

FLICKER MITIGATION IN INDUSTRIAL SYSTEMS

LEON DE WIT



Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Electronic Engineering with Computer Science at the University of Stellenbosch.

Study Leader: Prof. H. du Toit Mouton

April 2006

DECLARATION

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Ek, die ondergetekende, verklaar hiermee dat die werk in hierdie tesis vervat, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik by enige universiteit ter verkryging van 'n graad voorgelê het nie.

Signature/Handtekening:

.....

Date/Datum:

.....



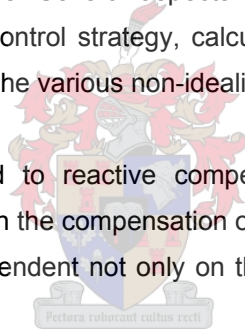
SUMMARY

This thesis investigates the compensation of voltage flicker in an industrial environment. Industrial loads draw progressively less sinusoidal currents. These currents cause non-sinusoidal voltage drop over the line impedance, causing a distorted line voltage. The light output of incandescent electric lighting systems is quadratically proportional to the line voltage, and thus variations in the line voltage cause irritating variations in the output of such systems.

Two tools to analyse flicker problems are developed: A USB data logger is built to log measured waveforms to computer hard disk. These data are analysed using a MATLAB implementation of the IEC-specified flicker meter.

A converter-based flicker compensator is found to be the only compensator capable of compensating general flicker loads. Such a compensator is developed using the synchronous reference frame filtering technique. Several aspects of the compensator are dealt with in detail including selection of a current control strategy, calculation and implementation of the converter duty cycles and compensation of the various non-idealities in such a controller.

Full compensation is contrasted to reactive compensation – the second option being less expensive but also less effective in the compensation of certain loads. The effectiveness of reactive compensation is found to be dependent not only on the type of load, but also on the type of line feeding the flicker load.



Three industrial flicker loads are measured and analysed: a three-phase welder, a sawmill and a rock crusher. These loads are simulated, and the compensation proposed confirmed via these simulations. The compensation of the three-phase welder is tested using a hardware scale model. This verified the operation of the proposed flicker compensator in practice.

OPSOMMING

Hierdie tesis ondersoek die kompensasie van spanningsflikker in 'n industriële omgewing. Industriële laste trek toenemend minder sinusvormige strome. Hierdie strome veroorsaak 'n nie-sinusvormige spanningsval oor die lynimpedansie, wat lei tot 'n vervormde lynspanning. Die intensiteit van lig wat uit 'n gloeilamp gestraal word is kwadraties proporsioneel tot die lynspanning, sodat die variasies in lynspanning irriterende variasies in sulke beligtingstelsels se ligintensiteit teweegbring.

Twee gereedskapstukke word ontwikkel: 'n USB dataopnemer word gebou sodat gemete golfvorms op 'n rekenaarhardeskyf gestoor kan word. Hierdie data word verwerk m.b.v. 'n MATLAB-implimentasie van die IEC-gespesifiseerde flikkermeter.

Daar word bevind dat 'n omsettergebaseerde flikkerkompenseerder die enigste kompenseerder is wat algemene flikkerlaste kan kompenseer. So 'n kompenseerder word ontwikkel deur gebruik te maak van die sinchrone verwysingsraamwerk filteringstechniek. Verskeie aspekte van die kompenseerder word in besonderhede bespreek, veral die kies van 'n stroombeheerstrategie, die berekening en implementering van die omsetter dienssiklusse en die kompensasie van die verskeie nie-idealiteite in die beheerder.

Volle kompensasie word teenoor reaktiewe kompensasie gestel. Hier is die tweede opsie goedkoper, maar ook minder effektief in die kompensasie van spesifieke laste. Daar word bevind dat die effektiwiteit van reaktiewe kompensasie afhang nie net van die las nie, maar ook van die soort lyn waardeur die las gevoer word.

Drie industriële flikkerlaste word gemeet en ontleed: 'n driefase sweismasjien, 'n saagmeule en 'n klipvergruiser. Hierdie laste word gesimuleer en die voorgestelde kompensasie word deur simulاسie bevestig. Die kompensasie van die driefase sweismasjien word ook getoets deur 'n hardeware skaalmodel. Dit bewys die korrekte werking van die kompenseerder prakties.

DANKBETUIGINGS

Graag rig ek die volgende dankbetuigings aan persone en instansies waarsonder hiedie tesis nie moontlik sou gewees het nie:

- Eerstens, aan my Hemelse Vader alle eer. Van Hom alleen ontvang ons alles wat ons het, en deur Hom alleen bereik ons alles wat ons bereik.
- Aan Prof. Toit Mouton, my studieleier, vir sy kritiese leiding en ondersteuning gedurende die doen van die navorsing en die skryf van hierdie tesis.
- Aan die lede van die Drywingselektronikanavorsingsgroep by die Universiteit van Stellenbosch. Baie dankie aan elkeen wat deur raad en kritiek 'n bydrae gelewer het tot hierdie tesis. Spesifiek dankie aan Wernher Swiegers en Dr. Jaques du Toit wat my gehelp het om die DPQC goed te verstaan en te gebruik. Dankie ook aan Pieter Henning, by wie ek boekdele geleer het oor ingebedde programmering; en aan Aniel le Roux, wie se kennis van beheeralgoritmes van onskatbare waarde was.
- Aan my gesin en vriende, wat my deur hul ondersteuning bygestaan het. Dankie veral aan die mense van die BTK, by wie ek soveel geleer het, en van wie ek soveel ontvang het.
- Aan Hennie Mostert en Yusuf Martin van Eskom, Johan Smit van Cape Sawmills en Luca Clayton van Holcin vir geleenthede om werklike data te bekom en die probleem uit die industrie se oogpunt te sien.
- Laastens, dankie aan Eskom, die NNS en THRIP vir finansiële ondersteuning.



Pectora roborant cultus recti

CONTENTS

Declaration.....	i
Summary.....	ii
Opsomming.....	iii
Dankbetuigings.....	iv
Contents.....	v
List of Figures.....	viii
List of Tables.....	x
Glossary.....	xi
CHAPTER 1: Introduction.....	1
1.1. Flicker principles.....	1
1.2. Compensation strategies.....	2
1.2.1. Compensation via a new line.....	3
1.2.2. Capacitive compensation.....	3
1.2.3. Compensation via an SVC.....	4
1.2.4. Converter-based compensators.....	6
1.3. Problem definition.....	7
1.4. Thesis outline.....	7
CHAPTER 2: Metering flicker.....	9
2.1. Motivation for a flicker meter.....	9
2.2. Available flicker meters.....	10
2.3. The IEC flicker meter.....	10
2.3.1. Block 1: Calibration and scaling.....	11
2.3.2. Block 2: Model of an incandescent lamp.....	11
2.3.3. Blocks 3 and 4: Model of the response of the human visual system.....	12
2.3.4. Block 5: Statistical evaluation.....	14
2.4. Implementation of the flicker meter in MATLAB.....	15
2.5. Verification.....	16
2.6. Chapter summary.....	17
CHAPTER 3: A USB data logger.....	18
3.1. Motivation for a data logger.....	18
3.2. Overview.....	19
3.3. Measurement system.....	21
3.4. PEC33 code and hardware.....	22
3.5. PC software.....	23
3.6. Results.....	24

3.7. Chapter summary	25
CHAPTER 4: Finding a compensation strategy	27
4.1. Overview of compensation theory	27
4.2. p - q Theory.....	29
4.2.1. α - β Coordinates	30
4.2.2. p - q Theory in α - β coordinates	31
4.3. Synchronous reference frame filtering.....	34
4.4. Comparison of p - q and synchronous reference methods.....	37
4.5. Chapter summary	39
CHAPTER 5: Implementation	40
5.1. General notes on current-controlled PWM converters	40
5.2. Proportional integral control.....	41
5.3. Hysteresis Control.....	42
5.4. Deadbeat control.....	43
5.5. Selection of deadbeat control	43
5.6. Practical considerations	44
5.7. Time delays.....	45
5.8. Space vector modulation	47
5.9. Dead time effects	50
5.10. Generation of switching signals	57
5.11. Chapter summary	58
CHAPTER 6: Case studies	60
6.1. Model of the compensator	60
6.2. Case study I: Three-phase arc welder.....	61
6.3. Case study II: Sawmill.....	69
6.4. Case study III: Rock crusher.....	76
6.5. Conclusions	80
6.6. Chapter summary	81
CHAPTER 7: Comparison with an SVC.....	82
7.1. Principles of operation	82
7.2. Simulation in Simpler.....	83
7.3. Simulation results.....	85
7.4. X/R sweep.....	87
7.5. Chapter summary	88
CHAPTER 8: Hardware simulation and results.....	89
8.1. Motivation for doing a hardware simulation	89
8.2. The DPQC as compensator.....	89
8.3. Selection and modelling of the three-phase load	91

8.4.	Compensation results	94
8.5.	Chapter summary	101
CHAPTER 9: Conclusion		102
9.1.	Thesis conclusion	102
9.2.	Future work	103
9.3.	Thesis contribution	104
References		105
APPENDIX A: Selected proofs		109
A.1.	Active vs. reactive compensation	109
A.2.	Derivation of the SVC firing angle – impedance relationship	110
A.3.	SVC current harmonics	112
APPENDIX B: MATLAB flicker meter simulation script		114
APPENDIX C: USB data logger code & hardware		117
C.1.	PEC33 DSP code	117
C.2.	PEC33 FPGA VHDL code	138
C.3.	USB daughter board schematic	146
C.4.	C++ code for PC	147
APPENDIX D: Simulation scripts		154
D.1.	Simplorer model of the DPQC controller	154
D.2.	MATLAB simulation of jaw crusher compensation	160
APPENDIX E: Flicker generator		164
E.1.	Controller VHDL code	164
E.2.	Controller schematic	166
APPENDIX F: DPQC code		167
F.1.	Extract from moddec.c	167
F.2.	Extract from space.c	168

LIST OF FIGURES

Fig. 1-1: Single-line diagram of a load causing flicker	2
Fig. 1-2: Compensation via a new line	3
Fig. 1-3: Capacitive compensation	3
Fig. 1-4: Compensation by an SVC.....	4
Fig. 1-5: Resistive load with reactive line impedance	5
Fig. 1-6: Resistive load with resistive line impedance.....	5
Fig. 1-7: Compensation using a power electronics converter	6
Fig. 2-1: Traditional flicker curve	9
Fig. 2-2: Blocks in the IEC flicker meter	11
Fig. 2-3: Implementation of block 1	11
Fig. 2-4: Block two is simply a squaring operator	12
Fig. 2-5: Block 4 output	16
Fig. 3-1: Using the data logger	18
Fig. 3-2: Data logger block diagram	20
Fig. 3-3: Measurement card schematic.....	21
Fig. 3-4: Data logger filters (normalised).....	23
Fig. 3-5: Data logger results.....	24
Fig. 4-1: Fryze's orthogonal currents	29
Fig. 4-2: Traditional three-phase voltage representation and α - β voltage representation.....	30
Fig. 4-3: q does not lie in the α - β plane.....	32
Fig. 4-4: Vector representation in the α - β plane.....	35
Fig. 4-5: Vector representation in the synchronous d - q plane	36
Fig. 4-6: Simpler simulation comparing control strategies	37
Fig. 4-7: Simulation results – p-q Method	38
Fig. 4-8: Simulation results – d-q Method	38
Fig. 5-1: Control loop.....	40
Fig. 5-2: P-I Control	41
Fig. 5-3: Hysteresis control	42
Fig. 5-4: Deadbeat control.....	43
Fig. 5-5: Single phase of the converter output filter	44
Fig. 5-6: Delays in reference calculation	45
Fig. 5-7: Rotation in the α - β plane.....	47
Fig. 5-8: SVM sectors and vectors	48
Fig. 5-9: Decomposition of reference voltage	49
Fig. 5-10: Single phase arm of a three-phase converter.....	51

Fig. 5-11: Converter switching waveforms 52

Fig. 5-12: Three-phase converter..... 53

Fig. 5-13: PWM generation from duty cycle reference..... 58

Fig. 6-1: Simplorer simulation of the compensator 61

Fig. 6-2: Single-line diagram of the grid around the welder 62

Fig. 6-3: Voltage and current in Blackheath..... 62

Fig. 6-4: Simplorer simulation of the three-phase welder as load..... 64

Fig. 6-5: Comparison of measured and simulated current..... 65

Fig. 6-6: Effect on the voltage at the PCC..... 66

Fig. 6-7: Simulation of three-phase welder with compensator..... 66

Fig. 6-8: Three-phase welder simulation results 67

Fig. 6-9: Interaction between saw flywheel and blade 70

Fig. 6-10: Sawmill RMS current 70

Fig. 6-11: Sawmill current spectrum..... 71

Fig. 6-12: Sawmill current waveform..... 72

Fig. 6-13: Implementation of the disturbance function 73

Fig. 6-14: Simulated sawmill current waveform 74

Fig. 6-15: Sawmill simulation results 76

Fig. 6-16: Two types of rock crushers..... 77

Fig. 6-17: Crusher RMS currents 78

Fig. 6-18: Jaw crusher simulation results..... 79

Fig. 7-1: Simplorer simulation of an SVC 83

Fig. 7-2: SVC compensation range 84

Fig. 7-3: SVC currents..... 85

Fig. 7-4: Compensation sweeping the X/R ratio 87

Fig. 8-1: Block diagram of the DPQC..... 89

Fig. 8-2: Hardware simulation of the three-phase welder 92

Fig. 8-3: Photo of the hardware simulation setup 93

Fig. 8-4: Measured compensation results – Basic compensation 94

Fig. 8-5: Basic compensation - Spectrum and zoom 95

Fig. 8-6: Measured compensation results – Energy 96

Fig. 8-7: Measured compensation results – Time compensated 97

Fig. 8-8: Time compensated – Spectrum and zoom..... 97

Fig. 8-9: Measured compensation results – Time and dead time compensated 98

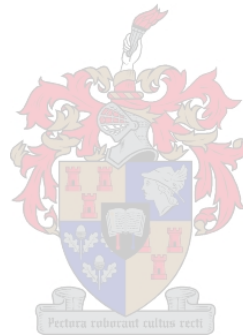
Fig. 8-10: Time and dead time compensated – Spectrum and zoom 99

Fig. 8-11: Measured compensation results - Reactive compensation 100

Fig. 8-12: Measured compensation results – Energy in reactive compensation 100

LIST OF TABLES

Table 2-1: Low pass filter parameters.....	13
Table 2-2: Weighing filter parameters.....	13
Table 4-1: Parameter values from simulation.....	38
Table 6-1: Parameter values for Simpler simulation of the compensator.....	61
Table 6-2: Power variations measured at Blackheath.....	63
Table 6-3: Parameter values for welder simulation.....	63
Table 6-4: Fourier coefficients for sawmill simulation.....	72
Table 6-5: Sawmill simulation parameters.....	74
Table 7-1: P_{st} values from X/R sweep.....	87
Table 8-1: DPQC parameter values.....	90
Table 8-2: Measurement equipment.....	90
Table 8-3: Load simulation parameter values.....	92



GLOSSARY

ADC	Analog to digital converter
DSP	Digital signal processor
EEPROM	Electrically erasable programmable read only memory
EPLD	Electronically programmable logic device
EMF	Electromotive force
FIFO	First in first out
FPGA	Floating point gate array
IEC	International electrotechnical commission
IGBT	Insulated gate bipolar transistor
PC	Personal computer
PCC	Point of common coupling
P-I	Proportional integral
PLL	Phase lock loop
POC	Point of compensation
PWM	Pulse width modulation
RMS	Root mean square
THD	Total harmonic disturbance
UPS	Uninterruptible power supply
USB	Universal serial bus
VHDL	Very high speed integrated circuit hardware description language

MATLAB is a registered trademark of The Mathworks Inc.

Simplorer is a registered trademark of Ansoft Corporation.

CHAPTER 1: INTRODUCTION

Utilities worldwide face ever increasing problems in power quality. The days of mostly static loads drawing sinusoidal currents are long gone. In their place has come a seemingly endless array of dynamic, non-linear and generally non-sinusoidal load types. Aggravating the problem is the growing divergence between generated power and demand. Economic and environmental constraints simply do not allow utilities to generate power with the wide margin of oversupply that was once the accepted norm. Busses are weaker, while having to supply quality power to increasingly demanding loads.

The results of power quality problems are well known. Harmonics and unbalance are major causes of inefficient power transfer and can cause severe damage to electrical machines experiencing torque pulsations. Failure of an overstressed phase in an unbalanced situation can be a very costly affair. Dips in the supply voltage disrupt plant control systems and can lead to expensive loss of production. Distortion of the voltage waveform is another problem, often amplified by the presence of inter-harmonics and current noise. These phenomena are especially troublesome as their effects are often stochastic or, even worse, chaotic.

A foremost effect of voltage distortion is flicker. Incandescent lamps have a light output proportional to the square of the voltage applied, making them very sensitive to variations in the voltage waveform. The human eye can detect such variations up to frequencies of about 40 Hz, with the most sensitive region being the 6 – 10 Hz range. In this very sensitive area, frequency modulation of the supply voltage of less than 0.5 % can be a severe irritation [01][02].

To solve the power quality problems they face, utilities increasingly look towards power electronics. No other technology can provide the benefits of almost infinite controllability with the ability to deliver power at the levels required. In addition, power electronics can provide much of the functionality of traditional power system components at much reduced losses. New applications are also possible. Many of the active filtering options available today cannot be implemented without using power electronics.

1.1. Flicker principles

As a rule, the voltage at the point of generation in a network can be taken as being practically sinusoidal. Designers of generators go to great lengths to minimise the amount of distortion and the harmonics present in the output of power stations. The role of generation in the flicker effect may therefore safely be ignored.

The primary cause of flicker is loads drawing non-sinusoidal currents with frequency components in the visible spectrum. The non-sinusoidal currents drawn cause a non-sinusoidal voltage drop over the line impedance (Z_{line} in Fig. 1-1). This causes a flicker problem at the point of common coupling (PCC) by distorting the line voltage (V_{pcc} in Fig. 1-1). Loads both upstream and downstream from the PCC will also be affected by the flicker, although the effect on the upstream loads will diminish as the line impedance diminishes.

As flicker is a current-induced problem, many compensation strategies focus on correcting the non-sinusoidal current flow in the system. From a measurement perspective, it is much easier to measure the variations in current causing the flicker problem than measuring the voltage variation itself. Because of the low expected value of the line impedance, the variations in current will be much larger than the variations in voltage. In spite of both current and voltage measurements being affected by noise, the expected signal to noise ratio (SNR) is much better for current measurements. Thus current-based strategies are to be preferred in compensators reacting to a reference current calculated from measurements.

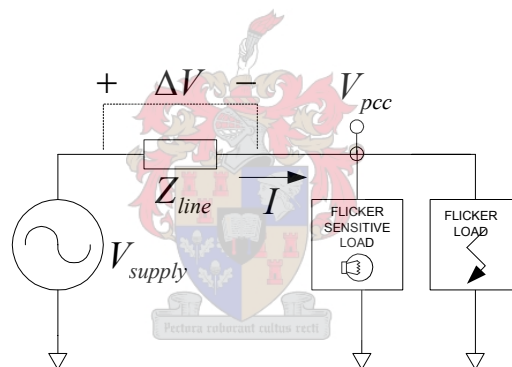


Fig. 1-1: Single-line diagram of a load causing flicker

1.2. Compensation strategies

Several strategies to mitigate flicker problems have been proposed and implemented. These strategies include new lines, capacitive compensation, static var compensators (SVCs) and the use of converter-based active filters.

1.2.1. Compensation via a new line

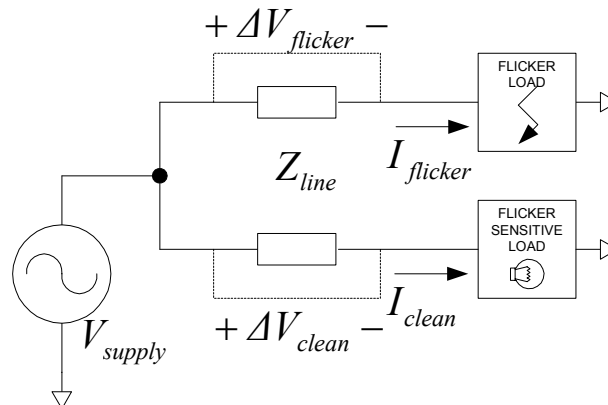


Fig. 1-2: Compensation via a new line

Providing the flicker-causing load with its own line from a strong bus is an effective mitigation strategy. As the flicker-causing currents drawn by the load no longer flow through the line impedance of the line feeding the flicker sensitive loads, these loads are unaffected by the operation of the flicker load. The obvious problem with this scheme is the cost of the extra line. Nevertheless, this has been the only workable solution for a number of high power flicker problems.

1.2.2. Capacitive compensation

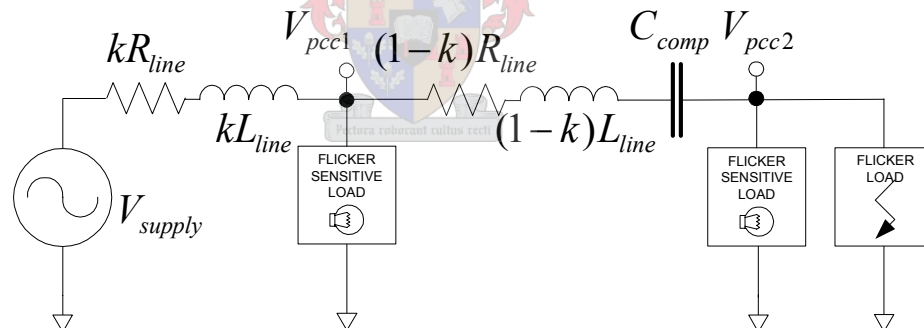


Fig. 1-3: Capacitive compensation

The compensating capacitor has no effect on the flicker problem at the PCC1

Capacitive compensation is aimed at decreasing the effective line impedance between source and PCC. As lines are usually at least partly inductive, this inductance may be cancelled by installing series capacitance. Unfortunately this is true only at one specific frequency. Capacitive compensation is also known to have lead to unwanted oscillations on power grids. This problem tends to be worse in the presence of flicker loads, as the non-linear nature of such loads usually implies the presence of a variety of frequencies in the load current. Capacitive compensation also compensates the flicker problem only at the point where the capacitors are inserted. Loads

upstream of the point of compensation (POC) will not be affected by the compensation, and may still experience intolerable levels of flicker.

1.2.3. Compensation via an SVC

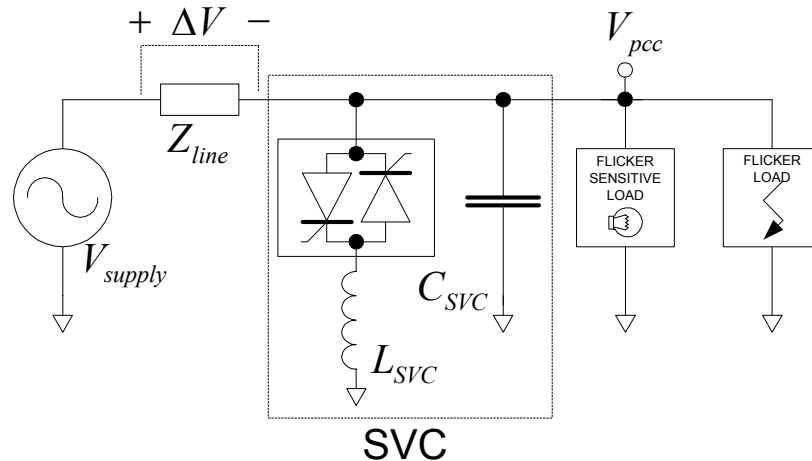


Fig. 1-4: Compensation by an SVC

Thyristor-controlled reactors (TCRs), thyristor-switched capacitors (TSCs) and static var compensators (SVCs) aim to control the reactive current drawn from the supply, thereby controlling the voltage drop over the line impedance and therefore also controlling the voltage at the PCC. This is done by connecting shunt impedance at the PCC, and varying this impedance to source or sink an appropriate measure of reactive power. The impedance is varied by adjusting the firing angle of thyristors connected in series with the compensator impedance.

In TCRs the impedance is formed by inductors, making a TCR an adjustable sink of reactive power. The left arm of the SVC in Fig. 1-4 is an example of a TCR. In TSCs the impedance is capacitive, making a TSC a variable source of reactive power. Combinations of TCRs, TSCs and fixed impedance produce a compensator capable of either sinking or sourcing reactive power. Such a device is called an SVC. SVCs have been used with success in the compensation of many flicker loads, but in many other cases they tend to be impractically large and expensive. In other cases they are simply ineffective [01]. The reason for these deficiencies is twofold: Because an SVC is switched only twice every cycle, the device frequency response is inherently limited. Also, an SVC is incapable of injecting real power.

To understand why reactive compensation does not provide adequate compensation of all flicker problems, a few special cases will be considered. Note that this discussion is not limited to SVCs, but contrasts reactive and active compensation in general.

If the load is purely reactive, a reactive power compensator can compensate the full variation in the current drawn. For a substantially resistive load this is not the case. This is illustrated in Fig. 1-5 and Fig. 1-6.

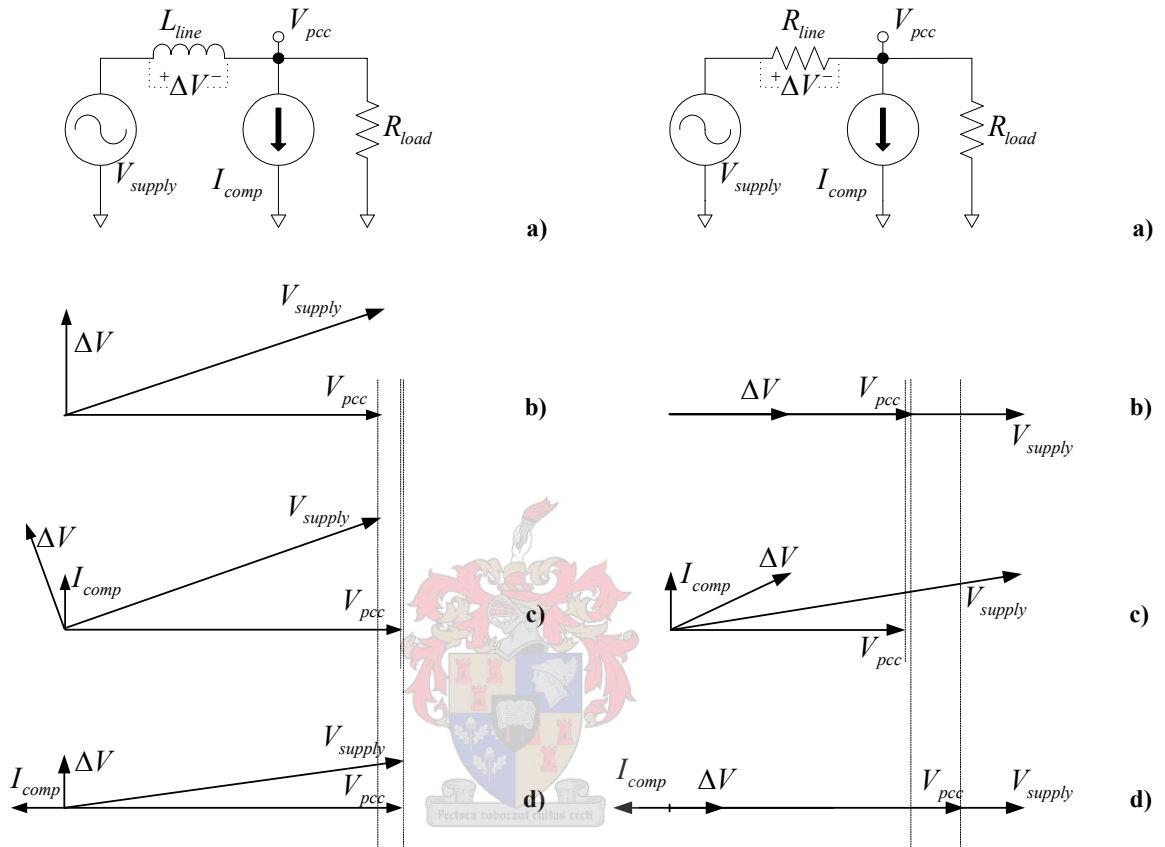


Fig. 1-5: Resistive load with reactive line impedance

a) Diagram, b) Uncompensated, c) Reactively compensated, d) Actively compensated

Fig. 1-6: Resistive load with resistive line impedance

a) Diagram, b) Uncompensated, c) Reactively compensated, d) Actively compensated

In the figures, the voltage V_{pcc} is assumed to lie at a specific point in the plane. Compensation will be evaluated by looking at how far a fixed amount of current is able to shift this voltage. The amount of shift is an indication of the controllability of the output voltage by a compensator with a given power rating.

In Fig. 1-5 the line impedance is mostly reactive. The influence of a certain amount of reactive compensation is shown in c). In d), the same amount of active compensation is applied. Neither active nor reactive compensation clearly shifts the voltage supply vector further than the other, indicating that both power components have a relevant influence on the output voltage.

In Fig. 1-6, the line impedance is mostly resistive. Active compensation produces a substantial shift in V_{pcc} , while reactive compensation has very little effect. In this case it is imperative that the compensator employed must be capable of supplying active power in order to compensate effectively.

From the preceding examples it is clear that a reactive power only compensator such as an SVC is unable to compensate general flicker loads. The more resistive a load and the associated line are, the less effective reactive compensation will be. A more mathematical approach leading to the same conclusion may be found in Appendix A.1.

1.2.4. Converter-based compensators

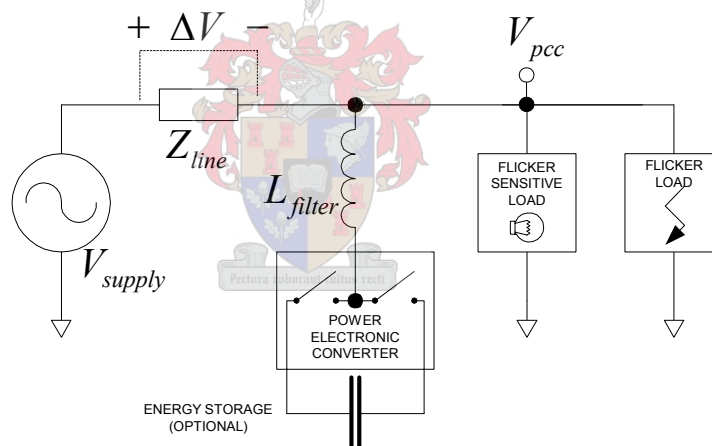


Fig. 1-7: Compensation using a power electronics converter

The solution to the problem of active power injection lies in the use of a power electronic converter. Such a compensator is capable of reactive power injection in the same way as was described for the SVC above. Connecting the compensator to an energy storage device gives the compensator the additional ability to regulate active power flow in the system. As a further advantage over SVCs, converter-based compensators have a much wider bandwidth.

The main disadvantage of converter-based compensators is cost – a suitable converter can cost up to several times the cost of an SVC with a similar power rating. Adding energy storage makes the compensator even more expensive. Thus, an SVC should be used whenever possible. In cases

where active compensation is needed or where SVC compensation is too slow, a converter-based compensator should be used.

1.3. Problem definition

From the previous section, it is clear that there is no single optimum solution for all flicker problems. A strategy that works excellently for one specific load may be quite ineffective or much more expensive than necessary in other circumstances.

New lines and capacitive compensation are methods relatively well understood. Similarly, SVCs have been used since the 1960s, and their operation is well documented in literature [03]. Much less work has been done in the field of converter-based compensators. Although compensators supplying only reactive power have been in use for some time, injecting active power to compensate flicker problems is a much more recent development. This thesis will focus on the use of such compensators, reviewing both the cases of reactive only compensation and compensation by supplying active and reactive power.

Because many practical flicker loads contain smoothing inductors for current control, inductance from motor windings, etc., these loads appear as current sources when viewed from the perspective of the line. This necessitates the usage of a parallel filter [04]. Parallel filters are also much easier to protect than series filters.

1.4. Thesis outline

This thesis starts with the development of two tools that are necessary for the study of the flicker problems to be compensated. Chapter 2 describes the development of a flicker meter. This is used to gauge the severity of flicker problems and the effectiveness of the proposed compensation throughout the thesis. Chapter 3 follows the development of a data logger capable of capturing voltage and currents over significant periods of time to an ordinary computer hard disk.

In Chapter 4 the compensation techniques used are motivated and developed. This is no straightforward matter, as the topic is still hotly debated in literature by various authors [05]. A choice of strategies is considered, and the $d - q$ static reference frame filtering technique is chosen as the most promising.

Chapter 5 deals with the implementation of the algorithms developed in Chapter 4 in a DSP based controller. Most converter-based compensators are designed to follow a reference current. The controller calculates the output voltage necessary to force the flow of this current, and switches the converter accordingly. Various aspects need to be taken into account when algorithms are

transferred from the mathematical to the physical domain. Timing delays and the need to translate a quasi continuous reference to discrete switching pulses impact on the effectiveness of the compensation. These issues have to be adequately addressed.

In Chapter 6, the developed techniques are comprehensively simulated using the Ansoft Simplorer simulation package. The prospects of the compensation method are reviewed under different load circumstances based on measurements taken in industry. In Chapter 7 a simulation of a static var compensator (SVC) is undertaken to contrast the developed methods with the capabilities of this popular device.

Chapter 8 describes a hardware simulation conducted in laboratory to verify the accuracy of the work done in the previous chapters.

Finally, in Chapter 9, the effectiveness of the proposed strategy is evaluated and assessed. Conclusions are drawn regarding the best way to compensate flicker under different conditions.

Each chapter will begin with a short introduction. The conclusions drawn in a chapter will be summarised at the chapter end.



CHAPTER 2: METERING FLICKER

In this chapter the usage of special measuring techniques when evaluating flicker will be briefly motivated. The International Electrotechnical Commission (IEC) proposed flicker meter will be dealt with in detail. The meter will be implemented in MATLAB, and the implementation will be shown to conform to the IEC standard.

2.1. Motivation for a flicker meter

As was noted in the introduction, flicker is a direct result of voltage variations. To evaluate the severity of the flicker problem at a specific point in the network, and to evaluate the effectiveness of any compensation applied, the expected irritation to humans has to be determined from the voltage waveform at the point of interest. As the human eye is more sensitive to some frequencies than others, a simple total harmonic disturbance (THD) figure does not provide sufficiently accurate information on how a human would perceive the effects of a polluted voltage signal. Weighing the harmonic components before calculating the distortion would be one way of incorporating the human visual spectrum in the measurement. This would still leave the problems of non-linearity and memory effect in human perception unaddressed.

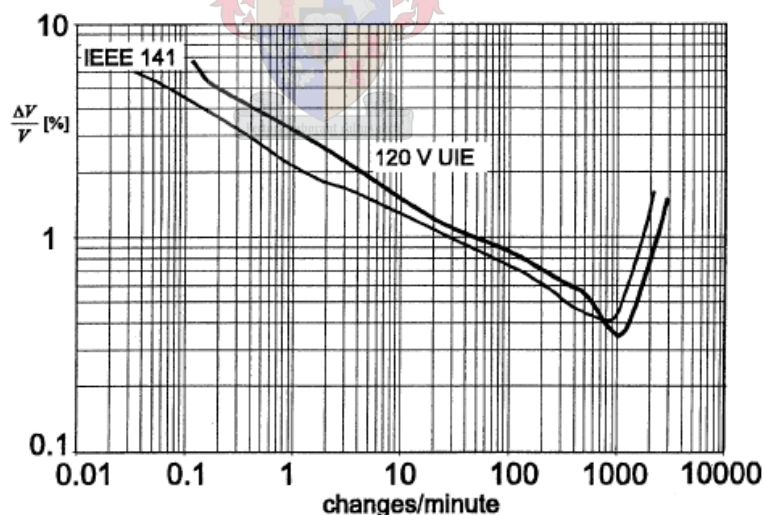


Fig. 2-1: Traditional flicker curve

Traditionally, flicker has been quantified using flicker curves. This is the approach taken in both IEEE 519-1992 [06] and IEEE 141-1995 [07], the two most relevant IEEE standards. An example of such a flicker curve is shown in Fig. 2-1, taken from [08]. The research these curves derive from is more than 50 years old. While incorporating some frequency information, this method shares the

problem of not adequately modelling the human visual response to flicker. This leads to the method being inaccurate in evaluating flicker in voltage waveforms modulated by the complex disturbance functions present in modern power systems. [08]

In recognition of these difficulties, various metering techniques have been developed to accurately measure the severity of flicker. The following paragraph will provide a brief overview of the development of the flicker metering standard used in this thesis, the IEC flicker meter specified in standard IEC 61000-4-15 [02].

2.2. Available flicker meters

The first flicker meter in general use was developed by the UK Electrical Research Association (ERA) in 1972. The meter was developed to enable an objective evaluation of the flicker problems associated with some large arc furnaces. Westinghouse Electric Corporation followed suit in 1974, developing a light weight flicker meter based on the work done by ERA. Meanwhile, another flicker meter was being developed in France by the Electricite' de France (EDF), the meter being introduced in 1976. Japan also developed a flicker meter - in 1978 the Central Research Institute of Electric Power Industry (CRIEPI) produced their ΔV_{10} meter.

With the multitude of flicker measuring techniques available at the start of early 1980s, the need for a generally accepted standard of flicker measurement was becoming apparent. The IEC defined an internationally agreed method of measurement along with accepted limits of tolerance. This work was formalised in a standard in 1986 [02].

The formalisation of flicker measurement in a standard does not imply that the work on flicker measurement is done. Considerable effort has been invested in improving the available techniques, prompting the IEC to temporarily update their standard flicker meter. Various authors continue to contribute to this effort. (As an example, see [09][10]) The latest, and most authoritative, version of the standard was published in 1997 and amended in 2003 [02]. This meter is recognised in South African standards by NRS 048-2:2003 par. 4.2.7 [11].

A flicker meter conforming to the IEC standard was constructed and tested in MATLAB. Extensive use was made of the work done by Rogóž [12], especially in the use of his parameter values for some of the flicker meter filters.

2.3. The IEC flicker meter

The IEC standard specifies a flicker meter that models the reaction of the human visual system to flickering light emitted by incandescent light sources. The severity of the disturbance is obtained as

perceived by humans. This data is then statistically evaluated to obtain two figures of merit: P_{st} , the short-term flicker severity, and P_{lt} , the long-term flicker severity, by which the flicker levels in a network may be quantified.

The standard itself discusses the flicker meter in five blocks. Each of these blocks will be treated in detail.

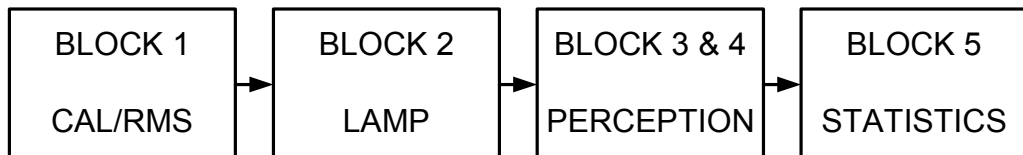


Fig. 2-2: Blocks in the IEC flicker meter

2.3.1. Block 1: Calibration and scaling

Block one fulfils two functions. Firstly a reference input is provided for calibration purposes. Such a reference is especially helpful in hardware implemented meters, where it often increases the accuracy of measurements if the meter is calibrated before use. In a software implementation the block is only needed to do the original calibration of the meter. Secondly the signal is scaled to a reference and represented as an RMS (root mean square) value. This enables the meter to accept input at any carrier amplitude and to provide flicker information on a percentage basis, rather than as an absolute variation.

Fig. 2-3 shows the implementation of this block as done by Rogóž. The input signal is scaled by the inverse of the average RMS value. The low-pass filter in the figure is used as Rogóž found this to be more efficient than computing the average from definition, and graphically proved the equivalence of the two methods in his paper.

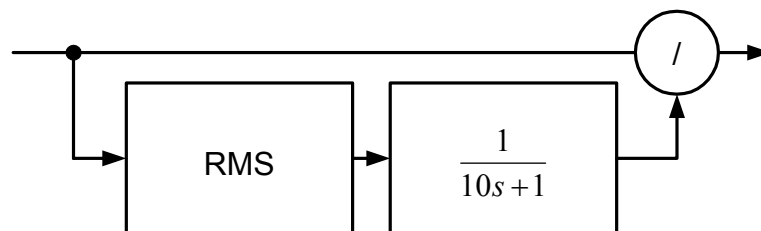


Fig. 2-3: Implementation of block 1

2.3.2. Block 2: Model of an incandescent lamp

The purpose of the second block is to simulate the light output of an incandescent lamp. If the lamp is modelled as being overbearingly resistive (a sound approximation for most household lamps) the

power dissipated in the lamp is directly related to the square of the voltage. Thus the lamp model is simply a squaring operator.

Strictly speaking, the standard defines the function of the second block as a “square law demodulator”. In this model demodulation is achieved by filtering in the next block.

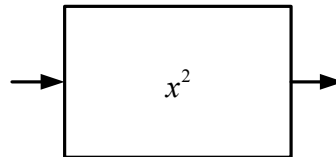


Fig. 2-4: Block two is simply a squaring operator

2.3.3. Blocks 3 and 4: Model of the response of the human visual system

The physical meaning of the distinction that the standard makes between blocks 3 and 4 is not clear. Therefore the two blocks will not be treated separately here.

The purpose of these blocks is to model the human visual system response to the varying output of the lamp in block 2. Firstly, as human eyes have a fixed bandwidth, the response of this block should also be bandwidth limited. Rogóz accomplishes this by building a band pass filter from a high and low-pass filter. The high pass filter has the simple transfer function

$$G_{high}(s) = \frac{s}{s + 2\pi 0.05}$$

Eq. 2-1

As can be seen from Eq. 2-1, the filter has a 3dB cutoff frequency of 0.05 Hz as specified in the standard.

The low-pass filter needs a sharper cutoff, which prompts the use of a 6th order Butterworth filter. A filter of this type and order is of the form

$$G(s)_{low} = \frac{1}{1 + b_1 \left(\frac{s}{\omega_c}\right) + b_2 \left(\frac{s}{\omega_c}\right)^2 + b_3 \left(\frac{s}{\omega_c}\right)^3 + b_4 \left(\frac{s}{\omega_c}\right)^4 + b_5 \left(\frac{s}{\omega_c}\right)^5 + b_6 \left(\frac{s}{\omega_c}\right)^6}$$

Eq.
2-2

The coefficients were found to be:

Coefficient	Value
b_1	3.864
b_2	7.464
b_3	9.141
b_4	7.464
b_5	3.864
b_6	1.000

Table 2-1: Low pass filter parameters

Again this is in conformance with the suggestion incorporated in the standard, both in the form of the filter as well as in the cutoff frequency of 35 Hz.

A weighing filter is applied to model the eye-brain response to flicker. The filter is of the form

$$G(s)_{\text{weighting}} = \frac{k\omega_1 s}{s^2 + 2\lambda s + \omega_1^2} \left(\frac{1 + \frac{s}{\omega_2}}{1 + \frac{s}{\omega_3}} \right) \left(\frac{1 + \frac{s}{\omega_4}}{1 + \frac{s}{\omega_4}} \right)$$

Eq. 2-3

with the parameters

Coefficient	Value
k	1.74802
λ	$2\pi 4.05981$
ω_1	$2\pi 9.15494$
ω_2	$2\pi 2.27979$
ω_3	$2\pi 1.22535$
ω_4	$2\pi 21.9$

Table 2-2: Weighing filter parameters

These parameters are specified in the standard. The filter has a resonant frequency of 8.8 Hz, which was found to be the frequency of maximum sensitivity for a 60 W, 230 V lamp. This lamp is

used extensively in Europe and South Africa. For other lamps, the weighing filter would have to be adjusted.

To model human nonlinear response to flicker, block 4 starts with a squaring operator. The signal is then averaged to model the so-called “storage effect” in the brain. This is done by low-pass filtering:

$$G(s)_{low} = \frac{1}{1 + \tau s}, \quad \text{Eq. 2-4}$$

$$\tau = 300ms .$$

The signal is scaled by a linear amplifier by a factor calculated to produce unity output at the agreed standard threshold of flicker perception.

The output at this stage represents the instantaneous flicker sensation as perceived by humans.

2.3.4. Block 5: Statistical evaluation

To produce the parameters P_{st} and P_{lt} a statistical evaluation of the instantaneous flicker sensation is done. The output of block four is sampled at a constant rate and divided into a number of classes. The minimum sampling rate is specified as 50 Hz. The classes are formed by specifying the base amplitude for each class, and a tolerance such that the classes span the entire range of possible output from block 4 without overlapping. Each time a new block four output value is recorded, the appropriate class counter is incremented. Thus a frequency distribution of flicker levels is obtained.

The short-term flicker severity can now be defined as

$$P_{st} = \sqrt{0.0314P_{0.1} + 0.0525P_{1.0s} + 0.0657P_{3.0s} + 0.28P_{10.0s} + 0.08P_{50.0s}}, \quad \text{Eq. 2-5}$$

where P_{xs} is the flicker level exceeded in x % of the observation period. An s in the subscript indicates the use of a smoothed value:

$$\begin{aligned}
 P_{50s} &= (P_{30} + P_{50} + P_{80})/3, \\
 P_{10s} &= (P_6 + P_8 + P_{10} + P_{13} + P_{17})/5, \\
 P_{3s} &= (P_{2.2} + P_3 + P_4)/3, \\
 P_{1s} &= (P_{0.7} + P_1 + P_{1.5})/3.
 \end{aligned}
 \tag{Eq. 2-6}$$

$P_{0.1}$ does not need smoothing as the low-pass filter in block 4 has a time constant of 300 ms, which prevents rapid variations in $P_{0.1}$.

The standard specifies the measuring period, T_{st} , for P_{st} to be selectable as 1, 5, 10 or 15 minutes.

P_{lt} is always measured over a period T_{lt} such that T_{lt} is an integer multiple of T_{st} . It is defined as

$$P_{lt} = \sqrt[3]{\frac{\sum_{i=1}^N P_{st_i}^3}{N}}
 \tag{Eq. 2-7}$$

Usually the short-term flicker severity is sufficient to describe the flicker situation in the region of a problem load if T_{st} is selected appropriately. For situations where many loads interact to contribute to the flicker problem or where the load has an exceptionally long (and possibly variable) duty cycle (such as is the case with arc furnaces), the long-term flicker value is more suitable to describe the magnitude of the flicker problem.

2.4. Implementation of the flicker meter in MATLAB

The IEC flicker meter was successfully implemented in MATLAB. The `lsim` function was used to simulate the response of the continuous filters. Using continuous to digital domain transformations to specify the whole system as a digital filter was attempted, but the loss of accuracy could not be justified by the increase in speed.

Block 5 was implemented somewhat differently from the specification. Instead of subdividing the output of block 5 into classes, the whole output vector was sorted using the very fast `sort` function. From this sorted vector, it was easy to find the precise flicker value exceeded in a specific percentage of T_{st} . As an example,

$$P_{10} = S \left[\text{round} \left(\frac{10}{100} N \right) \right]$$

Eq. 2-8

where N is the size of the vector and $\text{round}(x)$ rounds x to the nearest integer.

Using this method rather than step by step incrementing class counters improved the implementation speed dramatically without impacting on accuracy.

2.5. Verification

In [12] Rogó  provides figures showing the output of each block of the flicker meter for a predefined test signal, a 50 Hz sine wave modulated to a depth of 0.401 % by a 2 Hz square wave signal. The output obtained from the implemented flicker meter corresponded very well with the outputs as found by Rogó . Fig. 2-5 shows the output at block 4, the instantaneous perceived flicker. This corresponds closely to Fig. 10 in [12].

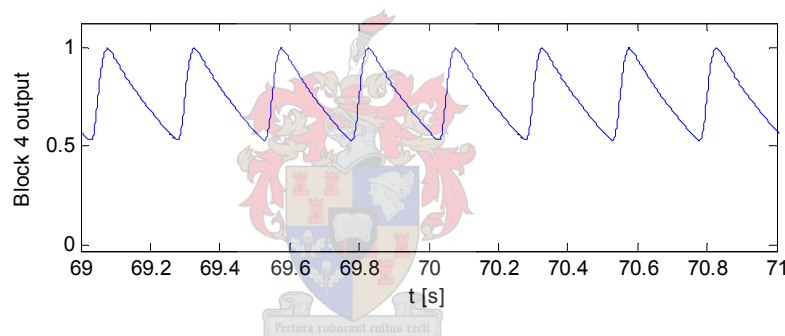


Fig. 2-5: Block 4 output

The flicker meter was further tested for conformance to the specifications in Tables 1 and 2 of the standard. These tables show the depth of modulation that should result in an instantaneous flicker value of unity at different frequencies. The implemented flicker meter was tested according to these specifications, and was found to be noticeably more accurate than the permitted 5 % deviation from the tabulated values.

In addition to these tests Rogó  also tested the flicker meter extensively on the basis of work done by Bień *et al.* as presented in [13]. He found the meter to be very accurate in the representation of flicker as perceived by humans.

2.6. Chapter summary

In this chapter human response to flicker was shown to be a complicated, highly non-linear process. This motivated the use of special techniques in the evaluation of flicker. After a quick review of flicker meter development, a flicker meter conforming to the IEC standard was developed in MATLAB. This meter models the whole flicker process, including the lamp emitting the flicker as well as a detailed model of the human perception of varying light output. Through the IEC the meter is recognised widely, especially in Europe, and also in South Africa through national standards. The meter was tested extensively and proved to conform to the standard as published.



CHAPTER 3: A USB DATA LOGGER

In this chapter the development of a universal serial bus (USB) data logger is motivated and described. Measured results are presented, verifying the success of the logger as a measurement tool.

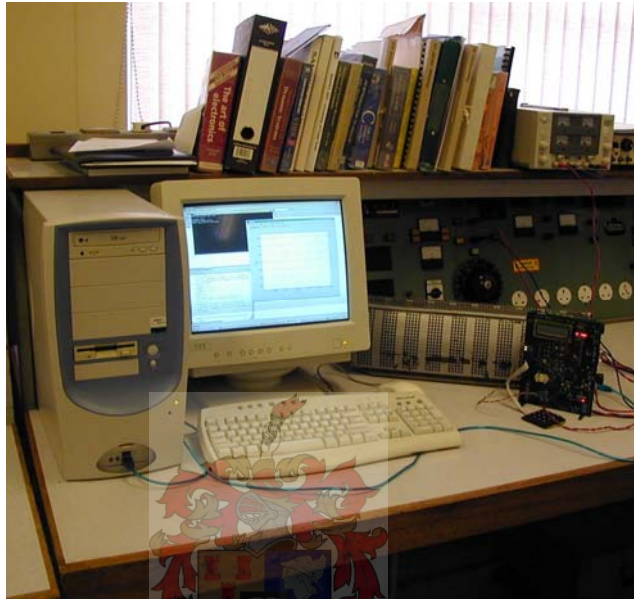


Fig. 3-1: Using the data logger

3.1. Motivation for a data logger

Flicker is a stochastic or chaotic phenomenon that incorporates significant variations occurring over periods of time very much longer than a line frequency fundamental period. The IEC flicker meter standard stipulates that short-term flicker severity should be evaluated over periods of 1, 5, 10 and 15 minutes. In contrast, the typical oscilloscope available in the laboratory can only store data spanning at most 10 s sampling at 1 kHz. To accurately analyse and simulate the studied flicker situations, a tool capable of recording waveforms for much longer periods of time was needed.

Several waveform recorders are available on the market. The decision to develop a logger was based on the following considerations:

- Implementing the logger in programmable hardware provided possibilities for customisation not possible if a logger had been purchased.
- Much of the hardware needed was available in the laboratory, making this implementation much cheaper than buying a ready made logger.

3.2. Overview

In designing the data logger, the first task was to identify a suitable storage medium. Two options are available, a hard drive or flash memory. A computer hard drive was selected. Rather than creating a dedicated storage system, the logger was implemented as an add-on to a normal personal computer (PC). This configuration provided a large, fast, reliable storage mechanism. The data was also available immediately for on site evaluation.

The logger itself was implemented on a PEC33 development board. The PEC33 was originally developed as a power electronics controller board [14], and therefore has more than enough computing power available to log data at the required rates. It incorporates a Texas Instruments TMS320VC33 digital signal processor (DSP) and two Altera ACEX EP1K500QC208-3 floating point gate arrays (FPGAs). The analog to digital converters (ADCs) are also from Texas instruments, the 12-bit TLV1570.

The measurement board was realised using a common differential measurement system utilising the Analog Devices AD623 differential amplifier. Voltage was sensed directly, and current was sensed using Universal Technic M1.UB current probes.

The logger was connected to the computer via a USB connection. USB was chosen as many laptops today do not have serial or parallel ports, making it difficult to connect them to equipment via RS232 or RS485. Also, the data rates possible with USB exceed those possible with these standards by far. When the PEC33 was developed originally it provided a USB port via a Philips PDIUSB12 USB driver. In this design, it was decided to discard this driver in favour of an FTDI FT232C. This choice will be motivated in section 3.5, where the development of the logger software is discussed.

A block diagram of the system is presented in Fig. 3-2.

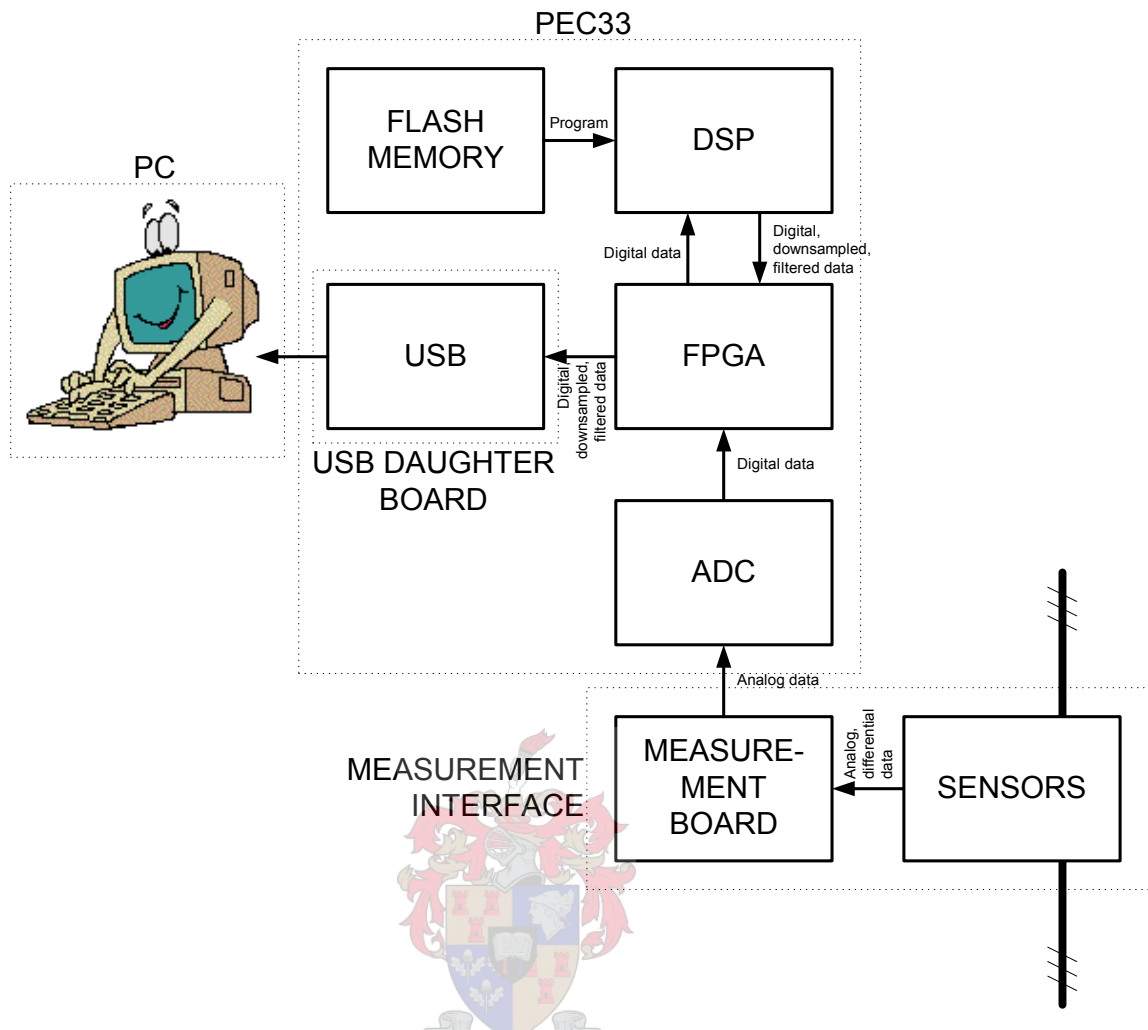


Fig. 3-2: Data logger block diagram

The basic data path can be seen from the diagram. The voltage signals from the voltage measurement points and the voltage output from the current sensors are fed, differentially, into the measurement board. This board contains the signal-conditioning circuitry and differential amplifiers. From here the signals are fed to the ADCs on the PEC33, where they are digitised and fed to the DSP via an interface in the FPGA. The DSP filters and downsamples the signal. The data is then sent to the PC via an FPGA interface to the USB driver.

The flash memory on the PEC33 was used exclusively to store the DSP program. It plays no direct part in the data stream.

The design of the components of the logger will now be dealt with in detail.

3.3. Measurement system

The purpose of the voltage measurement card is to convert the differentially measured high voltage input signal to a single-ended, scaled and filtered signal suitable as inputs to the PEC33 ADCs. Differential measurement is preferred as the area surrounding a power electronic converter is inherently electrically noisy. While single-ended systems are susceptible to common mode noise, differential systems cause common mode interferences to cancel, giving a much better common mode rejection ratio.

Another potential source of noise is aliasing. To ensure that the ADCs sample at at least two times the maximum frequency in the signal to be converted, the frequency content of this signal must be limited. This necessitates the inclusion of a low-pass filter. As this logger was developed especially for flicker measurements, information at frequencies outside the visible spectrum was irrelevant. Thus an even lower filter cutoff frequency could be chosen.

Lastly, the ADCs were protected by diode-clamping the measurement system output voltage.

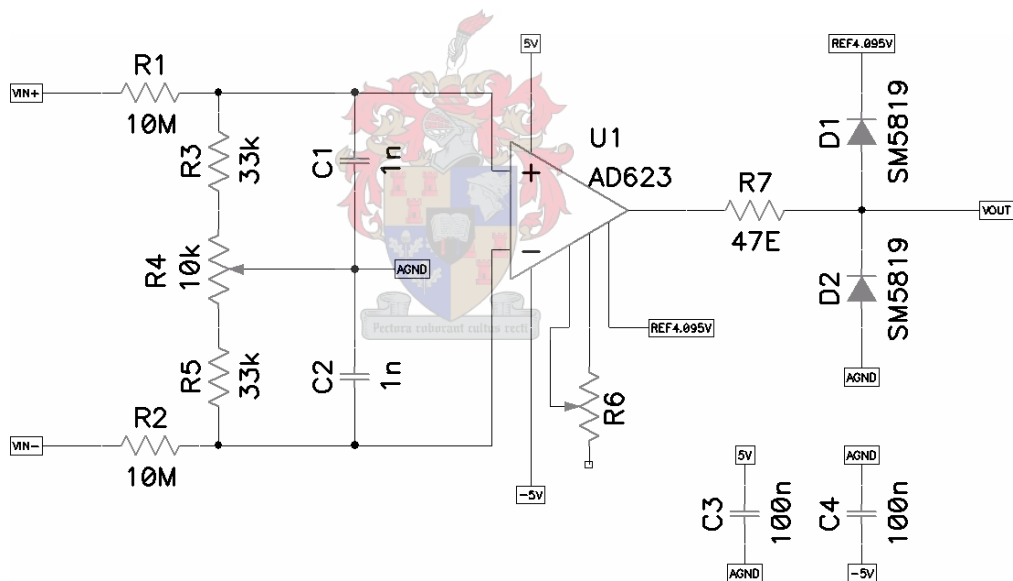


Fig. 3-3: Measurement card schematic

The input voltage scaling circuitry should ideally be specially calculated for each measurement range. Even though the AD623 enables the selection of the circuit gain via R_6 [15], amplifying a signal previously downscaled is poor practice as this noticeably increases system noise. Therefore R_6 was omitted from the circuit, resulting in the AD623 being programmed for unity gain.

With $R_1 = R_2$, $R_3 = R_5$ and $C_1 = C_2$ the transfer function of the input stage can be calculated to be

$$G = \frac{R_3 + 0.5R_4}{R_1(R_3 + 0.5R_4)C_1s + R_1 + (R_3 + 0.5R_4)} \quad \text{Eq. 3-1}$$

The diode clamping on the card output enforces an output range of 0.70 – 3.40 V, with the usual 0.70 V forward voltage drop assumption made for both diodes. The lower limit comes from the output being connected via reverse-biased diode to signal ground, while the higher limit comes from the diode connection to the ADC reference voltage. With the maximum input voltage specified as 250 V_{rms}, the input voltage range can be calculated as 707 V, giving a required scale factor of 3.82. This was easily realised by choosing $R_1 = 10 \text{ M}\Omega$, $R_4 = 10 \text{ k}\Omega$ and calculating $R_3 = 33 \text{ k}\Omega$. Because R_4 is a variable resistor, difference between the system analog ground and the ground of the input signal can be compensated for. Thus it is possible to guarantee that the input voltage per input pin (relative to analog ground) will not exceed the AD623 input range.

With C_1 selected as 1 nF the input stage cutoff frequency is 4.2 kHz, which is much lower than the 10 kHz sample rate selected for the PEC33 board. No aliasing should occur.

The current probe measurement system was developed in exactly the same way, remembering that the expected output voltage from the current probes is much lower than the input line voltages.

3.4. PEC33 code and hardware

On the PEC33 board the DSP commands the ADCs to sample via an interface implemented in the FPGA. The ADCs sample the input channels, and the values are sent to the DSP. The DSP clocks the ADCs at a rate of 10.0 kHz, much faster than the 1 kHz logging frequency. Thus only every tenth sample is needed. To prevent aliasing the data is digitally filtered before being down-sampled. The filter is of the form

$$G_z = \frac{z + 1}{z - 0.7285}, \quad \text{Eq. 3-2}$$

and was calculated from a 500 Hz continuous domain low-pass filter prototype using the Tustin transformation [16].

Again via an FPGA interface, the DSP sends the data to the FT2232C. The DSP also handles the logger user interface. It displays the current logger state on the PEC33 LCD, and offers the user limited control of the logging process through the PEC33 keyboard.

Used in “first in first out” (FIFO) mode, the FT2232C buffers the data and sends it to the logging PC at the first opportunity. Many other options are available, including UART emulation, ‘bit bang’ and a variety of serial protocols, some implemented via an internal serial engine. FIFO mode was selected as this mode of communication provided robust communications via a simple parallel interface. Mode selection, device identification, etc. is done via an external EEPROM (Electrically erasable programmable read only memory). This EEPROM is programmed via the USB bus using custom software supplied by FTDI. The USB daughter board was designed from the FT2232C datasheet [17].

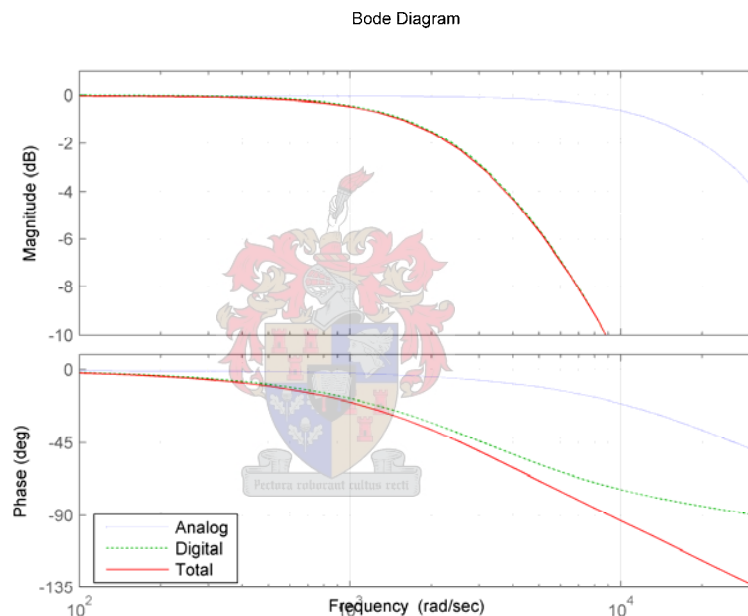


Fig. 3-4: Data logger filters (normalised)

3.5. PC software

One of the features of the FT2232C is the drivers FTDI provides with it. Most other USB protocol chip manufacturers leave the creation of PC side drivers up to the developer. This is a time consuming activity, requiring detailed knowledge of the specific operating system the drivers are written for.

Using the FTDI drivers, it was possible to quickly and easily create the C++ program that writes the data to hard disk. Extensive use was made of the programming manual provided by FTDI [18].

As the amount of data handled was relatively small compared to the memory available in a modern desktop computer, a single file was created for each run of the logger. (As an example of this, logging six channels at 16-bit resolution at the logger logging frequency of 1 kHz for an hour produces only 41.2 Mb of data.) After establishing a link with the logger, the program simply polled the FTDI data available flag to determine whether new samples were available. The data was then buffered in memory until the end of the logging run, at which time the whole file was written to disk.

MATLAB provides many tools for reading files and extracting data. Using some of these features, a MATLAB m-file was written to read the logged data and display the waveforms as a plot. The data was then available for evaluation, simulation, etc.

3.6. Results

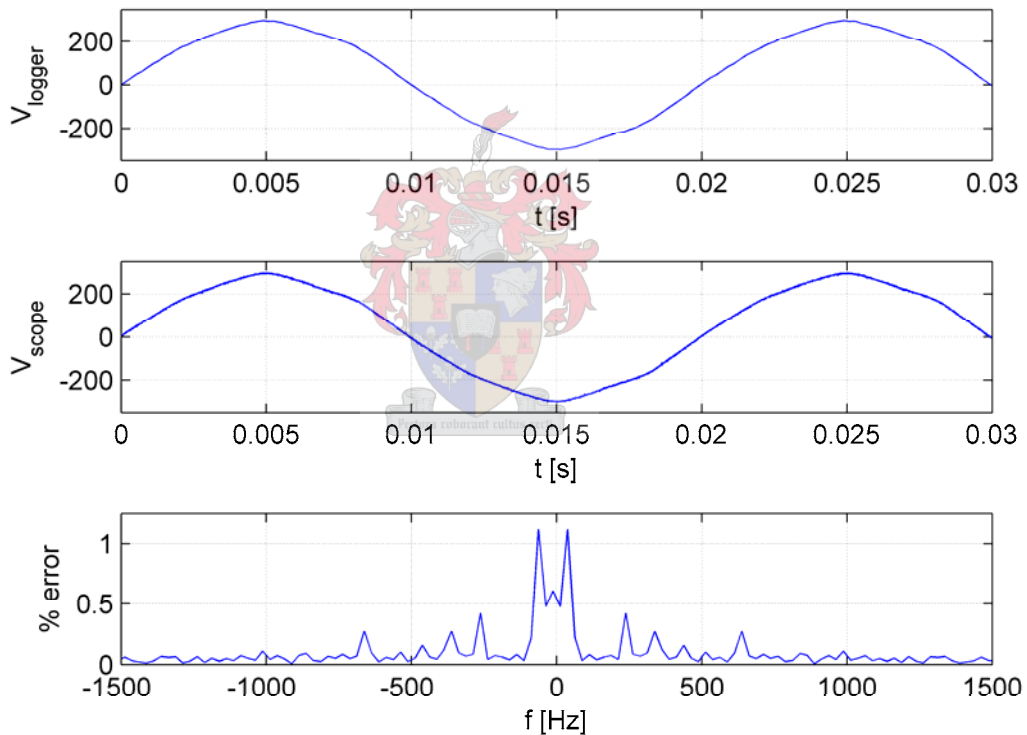


Fig. 3-5: Data logger results

Measurement by logger (top), Measurement by oscilloscope (middle), Fourier analysis of error (bottom)

Results from the data logger are shown in Fig. 3-5. From the top and bottom plots, the logger can be seen to follow the input waveform closely. Better insight into the logger behaviour is gained from

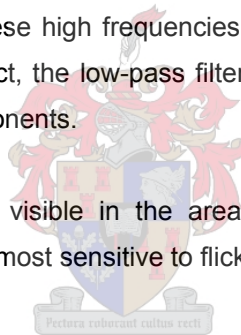
the Fourier analysis of the error signal. The error signal was computed by aligning the oscilloscope output and the data logger output in MATLAB and taking the difference.

Looking at the bottom plot in Fig. 3-5, the main error components tend to be at 50 Hz. This is to be expected, as some slight scaling error in the logging process is inevitable. Any slight misalignment of the oscilloscope and data logger signal will contribute to this error, although this does not represent any real errors in the logger. These errors should not impact on the flicker measured.

The next largest error component lies at zero frequency. As the logger logs only positive values, an arbitrarily chosen number represents zero. Changing this choice could eradicate this slight error. Again it has no bearing on flicker measurements.

The other noteworthy error components lie upwards of the fifth harmonic, or 250 Hz. The logger is expected to have a narrow pass band, given the slow sampling frequency and the numerous low-pass filters it contains, therefore the logged data is not expected to contain high frequency information. The oscilloscope measurement does contain this data, resulting in the apparent error. As no variations in voltage at these high frequencies are visible, they do not contribute to flicker and may safely be ignored. In fact, the low-pass filter character of the logger serves to suppress these high frequency noise components.

No large error components are visible in the area surrounding 50 Hz. The logger performs excellently in the frequency band most sensitive to flicker.



3.7. Chapter summary

In this chapter the development of a custom data logger was motivated. A computer hard drive was chosen as the most suitable storage medium, as it provides large capacity in an easy to access format. This motivated the development of the logger as an accessory to a standard desktop or laptop computer.

Most of the necessary logger hardware was found on the PEC33 power electronic controller board. As this controller was readily available, this provided a cost effective solution for much of the required logger hardware.

A USB connection was chosen as the means of communication between the PEC33 and the computer. USB was chosen because of its superior speed when compared to more traditional communication standards. In addition, it enables connectivity to nearly all modern computers. As no drivers were available for the original PEC33 USB interface, a daughter board was developed

using an FT2232C USB interface IC. This chip comes complete with all the necessary drivers to connect to a standard computer.

A C++ program was written to write the received data to disk. A MATLAB script was written to retrieve the data, enabling analysis such as flicker calculations.

Lastly the logger was verified and evaluated by measurement. The logger was found to perform quite satisfactorily in the frequency band needed in flicker analysis.



CHAPTER 4: FINDING A COMPENSATION STRATEGY

In this chapter, a compensation strategy for implementation in a converter-based compensator is sought. This starts with a short overview of the theory available, after which the two most applicable theories are covered in depth. Finally, the choice of compensation by synchronous reference frame filtering is motivated.

4.1. Overview of compensation theory

The world's first commercial power plant was built by Thomas Edison in 1880. This plant employed DC distribution, in which reactive power flow is not a problem. In 1886 George Westinghouse built his first power plant in Great Barrington, Massachusetts. This plant employed AC distribution in order to utilise transformers to alleviate voltage drop. Although AC presented major benefits over DC at the time, the problem of oscillating energy was noted and presented by Stanley and Shallenberger in their respective papers for the American Institute of Electrical Engineers and the Electric World [19][20]. Their explanation, attributing oscillating power in sinusoidal systems to the difference in phase angle between voltage and current is, with minor alterations, the same as that which can be found in textbooks and engineering standards today.

Work on general power theory was continued in the 1920s and 1930s by Budeanu and Fryze. In [21] Budeanu defined reactive power as:

$$Q_B \equiv \sum_{n=1}^{\infty} U_n I_n \sin \phi_n . \quad \text{Eq. 4-1}$$

The definition comes from a Fourier series decomposition of both the current and voltage. Multiplying the RMS value of each harmonic current (I_n) with the RMS value of the voltage harmonic at the same frequency (U_n) and the sine of the phase angle between the two quantities (ϕ_n) produces reactive powers in a manner analogous to that followed for traditional AC circuit analysis.

Hereby a new quantity, named the distortion power by Budeanu was introduced:

$$D_B \equiv \sqrt{S^2 - P^2 - Q_B^2}, \quad \text{Eq. 4-2}$$

where S is the apparent power, defined as the product of the RMS voltage and current, and P is the active power, defined as

$$P \equiv \sum_{n=1}^{\infty} U_n I_n \cos \phi_n. \quad \text{Eq. 4-3}$$

Despite objections from many authors, notably Fryze in [22], the Budeanu model gained wide acceptance both in academia and in industry. In 1977 Emanuel stated that “Budeanu’s model is today universally accepted” [23].

Budeanu’s theory presented a number of problems. Firstly, it was notoriously difficult to measure the distortion and reactive power, the first instruments capable of these measurements only appearing some fifty years after these powers were first defined [24]. Also, more and more authors questioned the validity of the definitions as proposed by Budeanu. Czarnecki in [24] and later also Emanuel in [25] finally proved that the Budeanu powers have no physical meaning.

Whereas Budeanu tried to define reactive power flow with the use of harmonics, Fryze’s definitions, as published in [22], are based on a wholly time-based approach. Fryze decomposed the current into a component having the same waveform as the source voltage, i_a , and a residual component, i_n , using the equations [05]:

$$i_a \equiv \frac{P}{\|u\|^2} u \quad \text{and} \quad \text{Eq. 4-4}$$

$$i_n \equiv i - i_a$$

where $\|u\|$ is the voltage RMS value and P is the average active power over a period.

The component i_a represents the active current component, while the component i_n represents the current associated with non-active power. These currents are mutually orthogonal, giving

$$\|i\|^2 = \|i_a\|^2 + \|i_n\|^2. \quad \text{Eq. 4-5}$$

This can be seen from the definition as well as from Fig. 4-1, presented below.

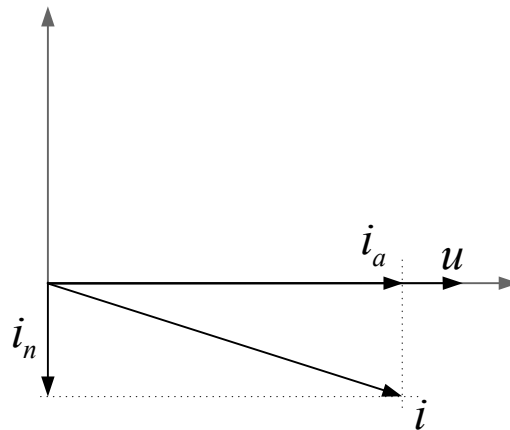


Fig. 4-1: Fryze's orthogonal currents

Multiplying Eq. 4-5 by $\|u\|^2$ produces the following definition of non-active power:

$$S^2 = P^2 + Q_F^2,$$

Eq. 4-6

$$Q_F \equiv \|u\| \|i_n\| = \sqrt{S^2 - P^2}.$$

Fryze's theory has been demonstrated to be correct [26], and has provided a basis for many other authors' theories.

Since the 1970s power electronics enabled the development of compensators that could follow a nearly arbitrary reference signal. This prompted renewed efforts to define non-active power for multiphase systems. An instantaneous method, rather than an average method such as the Fryze theory discussed above, was needed to control compensators providing real-time compensation with very fast response times.

Many theories have been put forward. Some have since been totally refuted, and some have been upheld for certain conditions but rejected in other cases. In almost all cases these theories have been extended and deliberated on by numerous authors. Only a few of these theories, relevant to the flicker compensator, can be discussed here. No less important, however, are the contributions of [27][28][29][30] and many others.

4.2. p-q Theory

In 1983 Akagi *et al.* introduced a novel definition of instantaneous active and imaginary powers [31]. Akagi followed Fryze's lead in defining the active current to be a dot product of the current and

voltage. In contrast to Fryze his definition of imaginary power was based on a cross product, and therefore completely different from any previous definition.

Akagi's theory is defined in the $\alpha - \beta$ coordinate system. This coordinate system is briefly described in the following paragraph.

4.2.1. $\alpha - \beta$ Coordinates

Traditionally, three-phase voltages are viewed as three voltage vectors in a plane, ideally separated by exactly 120° .

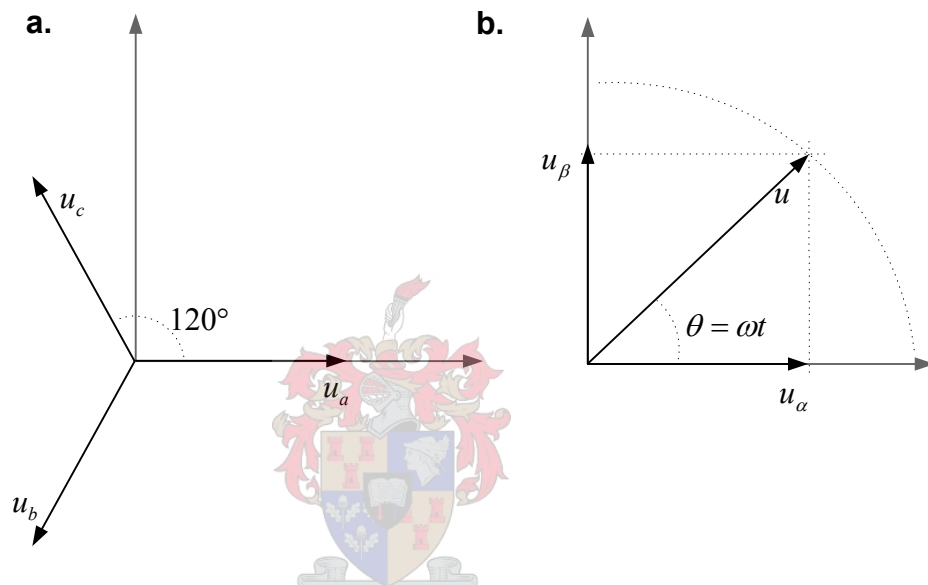


Fig. 4-2: Traditional three-phase voltage representation and $\alpha - \beta$ voltage representation

This representation contains redundant information. As the voltages are represented in a plane, only two variables should be necessary to uniquely describe the three-phase voltages.

As an example, it should be clear from Fig. 4-2a that the α -component of the voltage vector u_c may, via simple trigonometry, be found to be:

$$u_{c\alpha} = \cos(120^\circ) \times u_c = -\frac{1}{2} \times u_c.$$

Eq. 4-7

The full transform, known as the Clark transform, is given in Eq. 4-8:

$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} u_a \\ u_b \\ u_c \end{bmatrix}. \quad \text{Eq. 4-8}$$

The $\sqrt{\frac{2}{3}}$ scalar factor does not follow from the trigonometry, but is added for convenience in power calculations.

This transformation is also valid for three-phase currents, by virtue of the same arguments that were followed for voltages.

4.2.2. p - q Theory in α - β coordinates

Conventional instantaneous power in a three-phase circuit may be defined as:

$$p \equiv u \bullet i = u_a i_a + u_b i_b + u_c i_c, \quad \text{Eq. 4-9}$$

with the voltages referenced to an artificial common node such that

$$u_a + u_b + u_c = 0. \quad \text{Eq. 4-10}$$

It can be shown that

$$p = u_\alpha \bullet i_\alpha + u_\beta \bullet i_\beta \quad \text{Eq. 4-11}$$

also holds. This is an expected result, as Eq. 4-11 is basically a reaffirmation of Eq. 4-9, stated in a different coordinate system. Akagi goes further and defines a new quantity q :

$$q = u_\alpha \times i_\beta + u_\beta \times i_\alpha. \quad \text{Eq. 4-12}$$

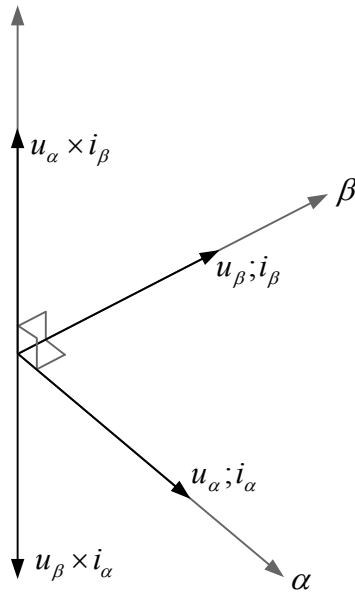


Fig. 4-3: q does not lie in the α - β plane

From Fig. 4-3 it can be seen that q does not lie in the $\alpha - \beta$ plain, but is perpendicular to it. Consequently, q cannot be regarded as being a conventional electrical quantity. The unit of q can be neither W, VA nor VAR. Akagi names this new quantity “instantaneous imaginary power”.

The current components in terms of the powers p and q are found by taking:

$$\begin{aligned}
 \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} &= \begin{bmatrix} u_\alpha & u_\beta \\ -u_\beta & u_\alpha \end{bmatrix}^{-1} \begin{bmatrix} p \\ q \end{bmatrix} \\
 &= \begin{bmatrix} u_\alpha & u_\beta \\ -u_\beta & u_\alpha \end{bmatrix}^{-1} \begin{bmatrix} p \\ 0 \end{bmatrix} + \begin{bmatrix} u_\alpha & u_\beta \\ -u_\beta & u_\alpha \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ q \end{bmatrix} \\
 &= \begin{bmatrix} i_{\alpha p} \\ i_{\beta p} \end{bmatrix} + \begin{bmatrix} i_{\alpha q} \\ i_{\beta q} \end{bmatrix}
 \end{aligned}$$

Eq. 4-13

where

$$i_{\alpha p} = \frac{u_{\alpha}}{u_{\alpha}^2 + u_{\beta}^2} p,$$

$$i_{\alpha q} = \frac{-u_{\beta}}{u_{\alpha}^2 + u_{\beta}^2} q,$$

$$i_{\beta p} = \frac{u_{\beta}}{u_{\alpha}^2 + u_{\beta}^2} p,$$

$$i_{\beta q} = \frac{u_{\alpha}}{u_{\alpha}^2 + u_{\beta}^2} q.$$

Eq. 4-14

The instantaneous real power can now be found in terms of the component powers p and q :

$$\begin{bmatrix} p_{\alpha} \\ p_{\beta} \end{bmatrix} = \begin{bmatrix} u_{\alpha} i_{\alpha} \\ u_{\beta} i_{\beta} \end{bmatrix} = \begin{bmatrix} u_{\alpha} i_{\alpha p} + u_{\alpha} i_{\alpha q} \\ u_{\beta} i_{\beta p} + u_{\beta} i_{\beta q} \end{bmatrix}$$

Eq. 4-15

$$p = p_{\alpha} + p_{\beta} = \frac{u_{\alpha}^2}{u_{\alpha}^2 + u_{\beta}^2} p + \frac{-u_{\alpha} u_{\beta}}{u_{\alpha}^2 + u_{\beta}^2} q + \frac{u_{\beta}^2}{u_{\alpha}^2 + u_{\beta}^2} p + \frac{u_{\alpha} u_{\beta}}{u_{\alpha}^2 + u_{\beta}^2} q.$$

Eq. 4-16

A number of important conclusions may be drawn from Eq. 4-16:

- The second and fourth terms in Eq. 4-16 sum to zero, indicating that q forms no part of the power flowing from the source to the load.
- Because q does not contribute to the power flow, but does increase line currents, (and therefore losses and possibly voltage distortion) q should be compensated.
- Because q is not associated with a flow of real power, a compensator compensating q only should not require any mechanism for energy storage.

The instantaneous real power p may further be divided into a constant component, \bar{p} , and a fluctuating component \tilde{p} . As p represents the flow of power from source to load, the component \tilde{p} must necessarily imply the variation in the power flow. For times when

$$-\tilde{p} > \bar{p}$$

Eq. 4-17

the net flow of power is negative (from load to source) which implies an exchange of energy between load and source.

Unfortunately neither \bar{p} nor \tilde{p} can be determined instantaneously but must be calculated by filtering \bar{p} as the DC component from p . A compensator attempting to cancel \tilde{p} would require an energy storage capability.

The instantaneous imaginary power q may also be divided into a constant component, \bar{q} , and a fluctuating component \tilde{q} . In a three-phase, balanced, sinusoidal circuit \bar{q} is numerically equivalent to the conventional reactive power Q . Filtering this component from the power to be compensated prevents the compensator from doing power factor correction. In cases where compensating harmonics and noise is important, but the power factor is not, this enables the design of an adequate compensator with a reduced power rating. In non-sinusoidal, unbalanced cases this property may still be utilised as an approximation. Care should however be taken as this may introduce substantial errors into the compensation effort.

Akagi's $p-q$ theory has been extensively implemented in industry, and generally performs well as a control philosophy in symmetrical systems. Under asymmetrical conditions the performance is not as good. The reason for this lies at the very heart of the theory. Akagi never motivates the choice of the definitions of q , other than by proving that q , as defined in his paper, causes currents to flow that does not contribute to the transmission of power. It is therefore completely possible that other power terms may be defined similarly, contributing to the currents flowing in the system but not contributing to the flow of real power. Czarnecki analyzes the non-symmetrical case in [32], and finds $p-q$ theory to be less than satisfactory in explaining the power phenomena in such cases. Kohata in [33] also reported such errors in an applied compensator.

In 1992 Willems [34] extended $p-q$ theory to multiphase systems. As the industrial flicker sources in this study are three-phase only, this extension is not investigated in this thesis.

Akagi extended $p-q$ theory in an appendix to [31] to three-phase four-wire systems. This extension completely attributes zero sequence components to active power. Peng generalised $p-q$ theory in [35][36] to allow for zero sequence components also contributing to non-active power. As the flicker sources studied in this dissertation are three-wire systems only, Peng's theory will not be discussed here.

4.3. Synchronous reference frame filtering

Synchronous reference frame filtering is based on a different compensation philosophy than the power theory described above. Most power theories calculate the part of the current in each phase that does not contribute to the flow of active power. By compensating these currents, the

compensated load is made to appear to absorb active power only. The compensated load therefore appears resistive.

In synchronous reference frame the goal is not to eliminate non-active power, but to force the compensated load currents to be sinusoidal. In a system with sinusoidal source voltages, sinusoidal currents guarantee sinusoidal voltage drop over the line impedance. The voltage at the point of common coupling (PCC) is therefore sinusoidal too. Furthermore, if the load and source are both balanced, the voltage drop in each line is equal. The voltage at the PCC would therefore also be balanced and free from distortion, although it is inevitable that there will be some voltage drop over the line.

The technique uses nearly the same $\alpha - \beta$ transform developed earlier in paragraph 4.2.1 to transform the line currents to a two dimensional vector representation.

Adding the α and β vectors produces a new current vector in the $\alpha - \beta$ plane. For balanced, sinusoidal currents, this vector has amplitude $\sqrt{\frac{3}{2}}$ times the amplitude of the composing abc currents and rotates at an angular frequency equal to the frequency of the abc sinusoids.

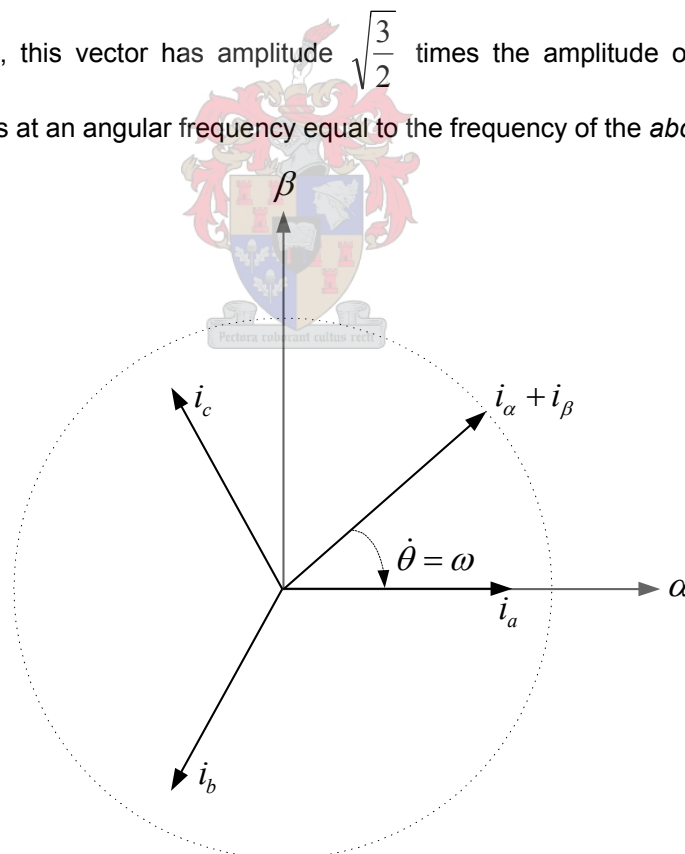


Fig. 4-4: Vector representation in the $\alpha - \beta$ plane

Under unbalanced conditions the circle in Fig. 4-4 becomes an ellipse. Non-sinusoidal conditions are manifested by a distortion of the circle.

Rotating the axes at an angular frequency equal to the angular frequency ($\dot{\theta}$ in Fig. 4-4), results in the locus of the current vector becoming stationary for the sinusoidal, balanced case. Unbalance or non-sinusoidal conditions are seen as noise around this stationary point. Because the rotation of the axes, or reference frame, is synchronised with the current vector this representation of the current is said to be in the synchronous reference frame.

As is the case in the α - β plane, the vector is described by its component vectors on the axes. These vectors can be found via Eq. 4-18. The resulting stationary vector is shown in Fig. 4-5.

$$\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} \cos(\omega t) & -\sin(\omega t) \\ \sin(\omega t) & \cos(\omega t) \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} \quad \text{Eq. 4-18}$$

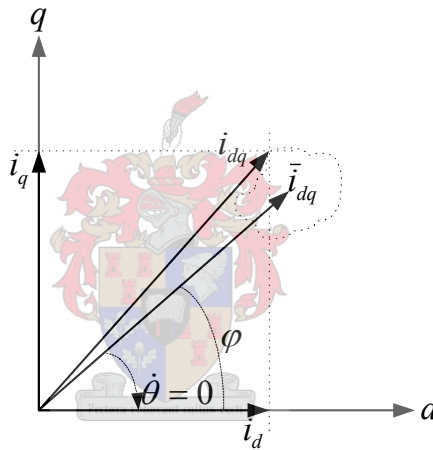


Fig. 4-5: Vector representation in the synchronous d - q plane

The d and q component vectors may be divided into stationary components, \bar{d} and \bar{q} , and varying components, \tilde{d} and \tilde{q} by low-pass filtering. Compensating the \tilde{d} and \tilde{q} components will cause the abc currents to be sinusoidal and balanced.

A phase lock loop (PLL) is necessary to compute the cosine and sine terms in Eq. 4-18. Although it is possible to lock this PLL to the currents, it is usually locked to the voltage. This causes the compensated vector \bar{i}_{dq} to appear at an angle, φ . This angle is numerically equal to the power angle, and the power factor in the system is equal to $\cos(\varphi)$. With φ so defined, compensating \bar{q} also forces the system to operate at unity power factor. Furthermore \tilde{d} now describes the

variation in *active* current. This enables the construction of a compensator without energy storage capability. Filtering q only, as was the case in p - q compensation, does not require energy storage.

Similar results may be obtained by filtering in the abc or $\alpha - \beta$ representations. Indeed, it has been proposed that for specific applications, filtering without the full synchronous reference frame transformation may be preferable [37]. Generally, filtering in the synchronous reference frame presents the following advantages:

- Only one or two filters are necessary depending on the compensator compensation goals. In the abc representation three filters are necessary.
- The filters necessary are simple low-pass filters. Often a first order filter is sufficient. Filtering in either the abc or $\alpha - \beta$ representations requires band pass filters of high order.

4.4. Comparison of p - q and synchronous reference methods

The methods investigated in the previous two sections were compared by simulation to find the most suitable strategy. A three-phase rectifier feeding an $R - L$ impedance was chosen as load. Flicker loads in general are non-sinusoidal and non-linear. The load chosen is an example of a simple load sharing these characteristics.

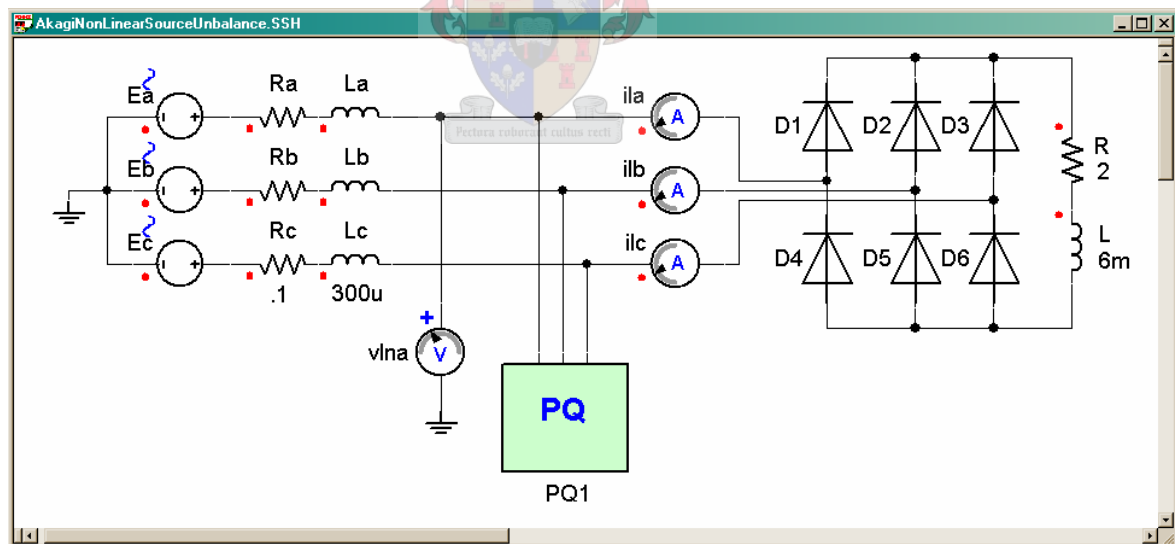


Fig. 4-6: Simplorer simulation comparing control strategies

Quantity	Value	Unit
Source voltages (Line to neutral)	230	V _{rms}
B-phase source voltage (Unbalanced case)	460	V _{rms}
Line resistance	0.1	Ω
Line inductance	300	μH
Load resistance	2	Ω
Load inductance	6	mH

Table 4-1: Parameter values from simulation

In Fig. 4-7 and Fig. 4-8 shows the results obtained from the simulation. Compensation was applied at 0.04 s.

Under balanced conditions with sinusoidal source voltages both strategies performed excellently. The load current was successfully reduced to the fundamental only. As a result the voltage drop over the line impedance should be sinusoidal too. This, together with sinusoidal source voltages, ensures nearly sinusoidal voltages at the PCC.

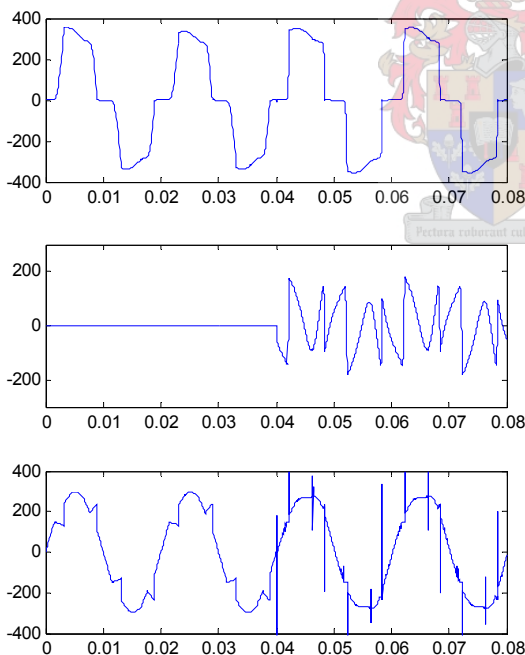


Fig. 4-7: Simulation results – p-q Method
 Load current (top), Compensator current
 (middle), PCC voltage (bottom)

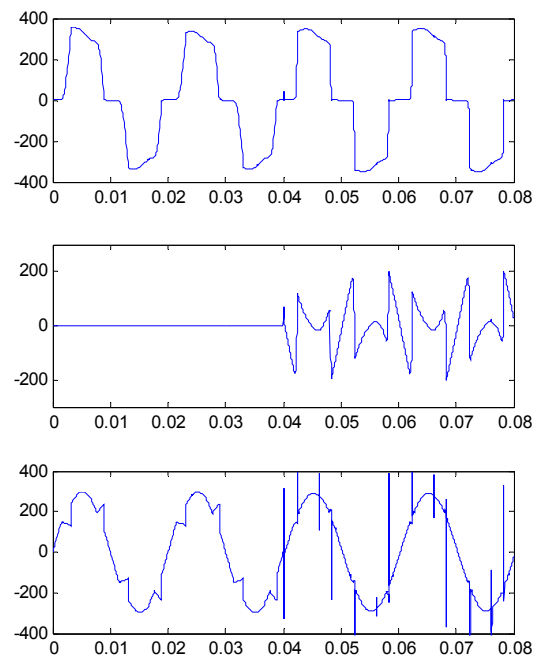
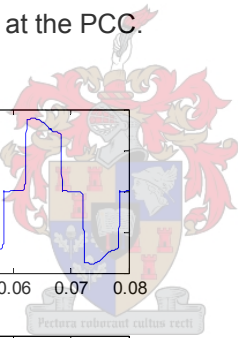


Fig. 4-8: Simulation results – d-q Method
 Load current (top), Compensator current
 (middle), PCC voltage (bottom)

In the case of an unbalanced load with sinusoidal source voltages, the results obtained above still hold and one method is not to be preferred above the other.

Based on these simulations and the preceding theory, it was decided that synchronous reference frame filtering would be used in this study to generate the controlling reference for the active filter developed. It provides all the benefits of $p-q$ theory: Compensation with/without energy storage and the ability to select whether power factor compensation is done (or omitted to reduce the compensator power rating). It has a solid theoretical backing based on clearly defined assumptions. These assumptions will be shown to be true for the loads investigated in this study. The strategy performs well under asymmetrical conditions, making it more generally applicable than $p-q$ theory. Furthermore it can reasonably easily be implemented in a DSP for practical implementation and evaluation.

4.5. Chapter summary

In this chapter a short overview of the development of compensating strategy was given. This provided a basis for the discussion of the two compensation strategies evaluated: Akagi's $p-q$ theory and synchronous reference frame filtering.

Akagi's $p-q$ theory was covered in detail, including both the necessary transformations and the calculation of the p and q power components on which the compensation strategy is based. The popularity of this method bears testimony to its effectiveness, yet there is little mathematical foundation for the specific definition of q . The theory is not valid in unbalanced or in four-wire applications.

Synchronous reference frame filtering was covered in the same detail as the $p-q$ method. In contrast with $p-q$ theory, the assumptions made using this method are easily defined. It is applicable to all cases in which $p-q$ theory can be used, with the added benefit of also being valid under asymmetrical conditions.

After evaluating Akagi's $p-q$ compensation and synchronous frame filtering in the $d-q$ plane by simulation, synchronous frame filtering was selected as the more promising strategy.

CHAPTER 5: IMPLEMENTATION

This chapter shows how the theory developed in the previous chapter may be implemented in a practical DSP-controlled converter. Some simple current control schemes are evaluated, and deadbeat control is selected for this compensator. The scheme is discussed in detail, with reference to compensation of dead time and other non-linear effects. State space vector modulation is discussed as a PWM (pulse width modulation) strategy.

5.1. General notes on current-controlled PWM converters

It is well known that current-controlled converters present major advantages over traditional open loop controllers. Kazmierkowski summarises these advantages as follows [39]:

- control of instantaneous current waveform and high accuracy;
- peak current protection;
- overload rejection;
- extremely good dynamics;
- compensation of effects due to load parameter changes;
- compensation of the semiconductor voltage drop and dead times of the converter;
- compensation of the dc-link and ac-side voltage changes.

As flicker loads are varying loads by definition open loop control is not a possibility, the fifth point above being indispensable in flicker compensation. Voltage control may be considered, but as the variations in current usually are far greater than the variation in voltage, current control gives better dynamic response and noise immunity.

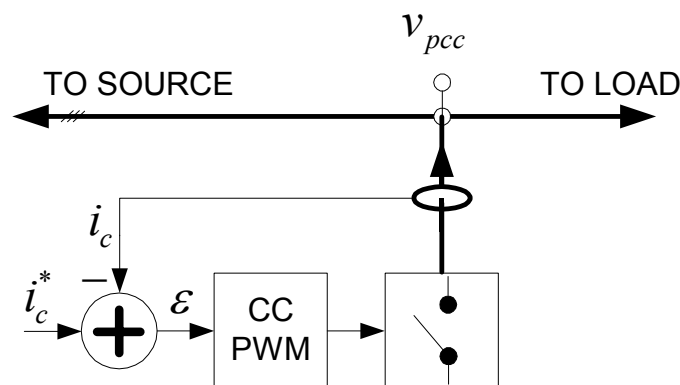


Fig. 5-1: Control loop

Fig. 5-1 shows the basic control loop in such a controller. The output current i_c is measured and compared to the reference current, i_c^* . This produces an error signal ε . It is the task of the controller to control the converter in such a way as to minimise ε .

The next sections will briefly cover a few popular current control schemes. More detail may be found in [39][40][41].

5.2. Proportional integral control

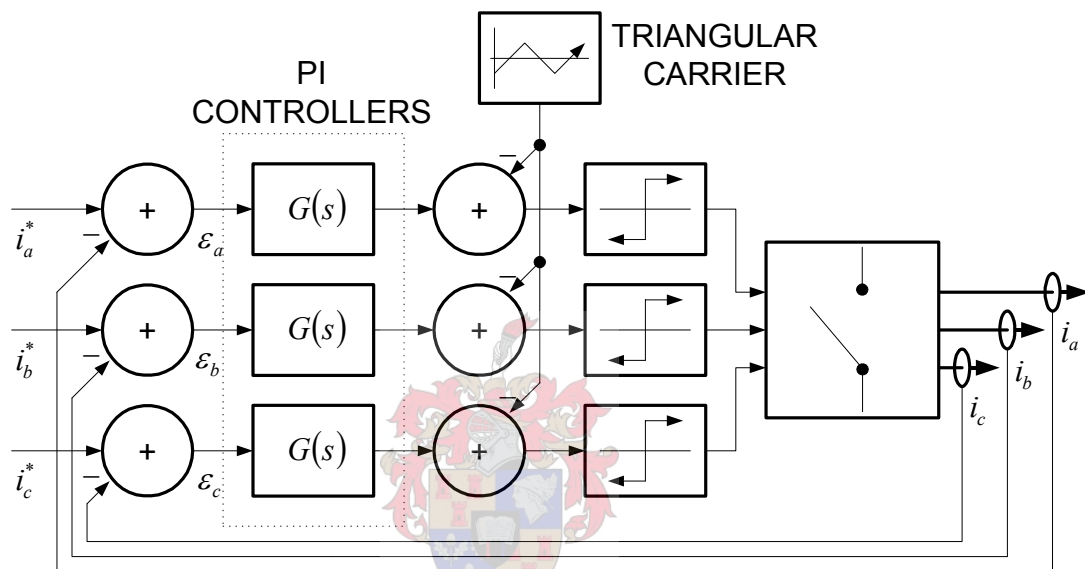


Fig. 5-2: P-I Control

This mode of control is directly derived from single phase proportional integral (P-I) control schemes. The controller behaves quite differently, however, as the fed back current ripple influences the switching times [39]. Three independent P-I controllers are used, one per phase generating reference voltages, u_a , u_b and u_c . These voltages are compared with a triangular wave to produce duty cycles d_a , d_b and d_c which are fed to the converter.

P-I control offers a well-defined harmonic spectrum, as all the generated harmonics must lie in the region of the carrier. This is usually well above the harmonic range of the load and source. The integral term in the compensator minimises the steady state error, although some tracking error is unavoidable. It is relatively easy to implement both digitally and in analog.

Difficulties arise when the slope of the command voltage exceeds the slope of the triangular carrier. Multiple crossings of the triangular boundaries can also cause problems. This limits the maximum

frequency of the load EMF (electromotive force) and command currents to a somewhat low $\frac{1}{9}$ th of the carrier frequency [42].

A variation on this scheme implements the controllers in the $d-q$ plane. Other modifications to the scheme include the use of PLL circuits and feed-forward correction to improve the tracking error.

5.3. Hysteresis Control

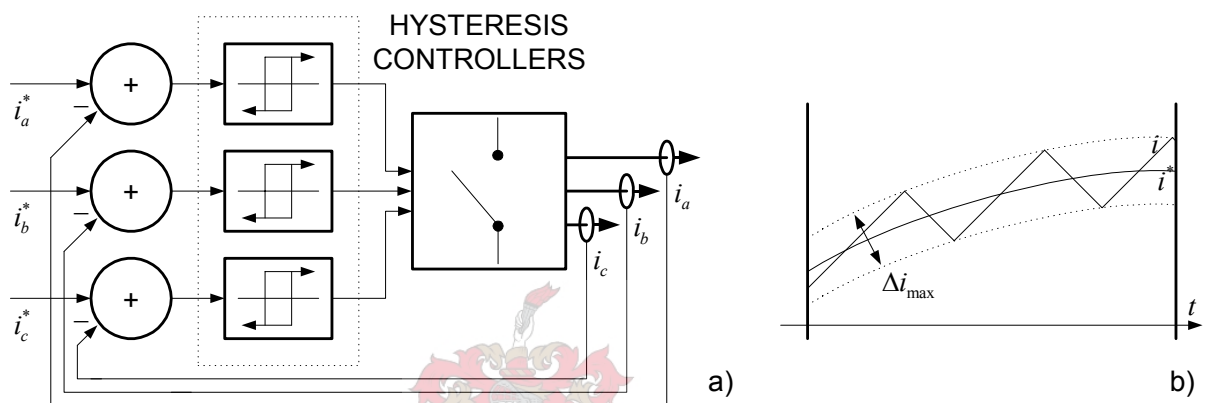


Fig. 5-3: Hysteresis control

a) Controller, b) Current following

A hysteresis controller does not implement a linear error minimization strategy. Instead, the reference current is directly compared to the output current. Two comparators are used to keep the current error in a narrow hysteresis band around the reference.

Hysteresis controllers are simple to implement, very robust, virtually independent of load parameters and offer excellent dynamic response. The ripple current is constant and easy to define from the hysteresis band.

The main disadvantage is the varying switching frequency. This causes switching harmonics that can be very difficult to analyze. This frequency is dependent on the load parameters and varies with the source voltage.

Many extensions of basic hysteresis operation have been suggested. The most drastic of these involves varying the hysteresis band to achieve a nearly constant switching frequency. Such a strategy solves the problem of unpredictable harmonics, but destroys one of the properties most attractive in hysteresis controllers – the simplicity. Nevertheless, such schemes are well suited to

applications where fast dynamic response is required, but stray harmonics cannot be tolerated. An example may be found in [43].

5.4. Deadbeat control

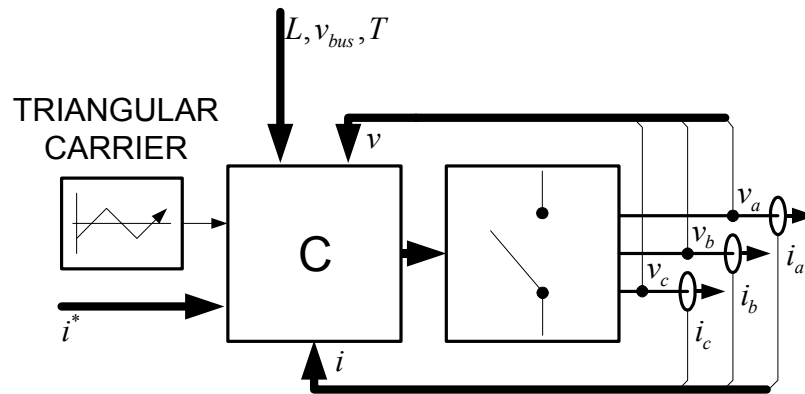


Fig. 5-4: Deadbeat control

Deadbeat control is a linear control method that uses known circuit parameters and measurements to calculate the future condition of the circuit. In this way reference voltages can be calculated to force the converter output currents to follow the references. Deadbeat controllers usually operate at constant frequency, with the output voltages calculated such that the desired output current is achieved at the end of each period, T .

Deadbeat controllers have better dynamic response than P-I controllers, although not as good as hysteresis controllers. Because the switching frequency is fixed, deadbeat controllers do not suffer from an unpredictable harmonic response like hysteresis controllers. The main disadvantage is the accurate circuit information required to do the necessary predictions.

5.5. Selection of deadbeat control

P-I controllers are inferior to hysteresis and deadbeat controllers in dynamic response. Hysteresis controllers can produce frequencies which may have an influence on the flicker situation in a network. Modified hysteresis controllers do not suffer from this difficulty, but are much more complex than the other methods considered. Accordingly it was decided to implement deadbeat control. As simulation results obtained later proved satisfactory, more complex algorithms such as modified hysteresis control are unnecessary.

5.6. Practical considerations

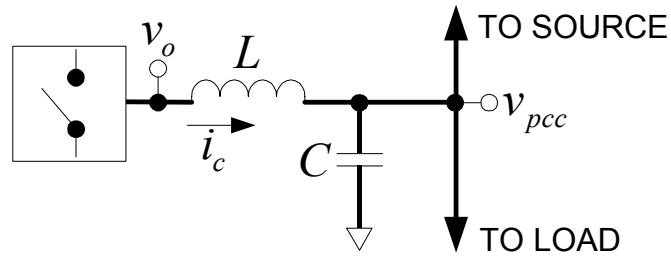


Fig. 5-5: Single phase of the converter output filter

Fig. 5-5 shows the output filter for a single phase of a three-phase converter. The implementation of deadbeat control will be explained from this figure.

If the circuit in Fig. 5-5 is viewed over a switching period, T , the current in the inductor at time t_0 may be found from basic circuit theory:

$$i_c(t_0) = i_c(t_0 - T) + T \frac{\overline{(v_o - v_{pcc})}}{L} \quad \text{Eq. 5-1}$$

with $\overline{(v_o - v_{pcc})}$ denoting the average voltage for the period, T ,

$$\overline{(v_o - v_{pcc})} = \frac{1}{T} \int_{t_0-T}^{t_0} (v_o - v_{pcc}) d\tau \quad \text{Eq. 5-2}$$

and all other symbols defined as in Fig. 5-5.

The following assumptions will now be made:

- T is small enough to make the variation in v_{pcc} negligible, that is, $\overline{v_{pcc}} = v_{pcc}$
- The capacitor current is comprised only of the ripple component of i_c .

The first assumption is generally valid for PWM-switched converters, as the converter switching frequency should be much higher than the fastest variation in the system.

The second assumption is a good approximation if the converter output filter works as expected. As a consequence, the converter output current into the PCC equals the average current, \bar{i}_c .

The required reference voltage can now easily be found from Eq. 5-1:

$$\bar{v}_o = \frac{L}{T}(i_c - i_c(t_0 - T)) - v_{pcc} \tag{Eq. 5-3}$$

by substituting the current by the reference current:

$$i_c(t_0) = i_c^*(t_0). \tag{Eq. 5-4}$$

Unfortunately Eq. 5-3 is not directly implementable because of unavoidable time delays in the compensator.

5.7. Time delays

One problem arises from the fact that the controller only realises current i_c^* at time $t_0 + T$. This means the controller will always lag the reference by exactly one period. Digital implementation introduces a calculation delay. This aggravates the problem and causes a lag of up to $2T$ in the compensator. Not only does this imply unsatisfactory reference following, but at time $t_0 + 2T$ the measurements taken at time t_0 would no longer be accurate, implying inaccuracy and possibly even instability. These problems are illustrated graphically in Fig. 5-6:

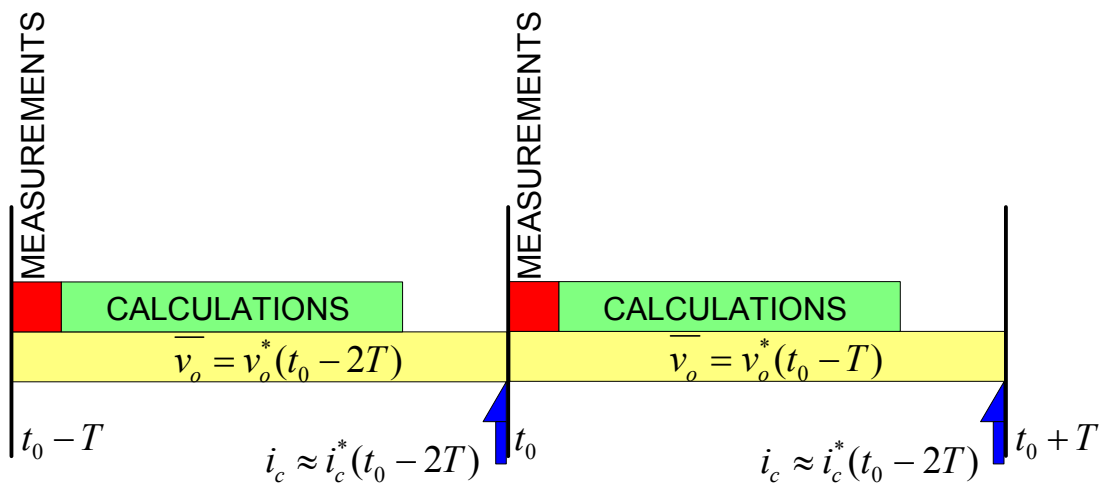


Fig. 5-6: Delays in reference calculation

From Eq. 5-3 and the preceding comments it is evident that the following terms must be predicted in order to overcome the difficulties discussed:

- i_c^* - The reference current will only be realised after time $2T$, therefore i_c^* should be predicted $2T$ ahead
- i_c - The current measured at time t_0 will be used in calculations in that period, but the resulting $\overline{v_o}$ will only take effect in the next period. Therefore i_c should be predicted one period ahead.
- v_{pcc} - For the same reason argued for i_c above, the v_{pcc} measured at time t_0 will effectively be used in the period $t_0 + T$ to $t_0 + 2T$. As Eq. 5-3 assumes v_{pcc} to be constant over a period, the best value to use would be the average value over the period in question. If v_{pcc} is approximated linearly, this would be the value at time $t_0 + 1\frac{1}{2}T$, $1\frac{1}{2}T$ after the measurement was taken.

Predicting i_c^* can be done in two ways: i_c^* can either be assumed to be linear with the prediction defined as

$$i_c^*(t_0 + 2T) \approx i_{cPred} \equiv i_c^*(t_0) + i_c^*(t_0 - T) - i_c^*(t_0 - 2T), \quad \text{Eq. 5-5}$$

or i_c^* can be assumed to be sinusoidal with frequency equal to the fundamental circuit frequency. If this second assumption is made, the three-phase i_c^* currents may be predicted by rotating the summed vector in the $\alpha - \beta$ plane:

$$\psi_{rotation} = 2\pi f t_{rotation}, \quad \text{Eq. 5-6}$$

$$i_{cPred\alpha\beta}^* = i_{cPred\alpha\beta}^* \begin{bmatrix} \cos \psi & \sin \psi \\ -\sin \psi & \cos \psi \end{bmatrix}. \quad \text{Eq. 5-7}$$

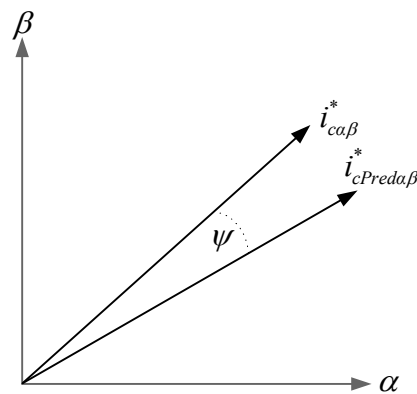


Fig. 5-7: Rotation in the α - β plane

For the a prediction of the reference current over two periods $t_{rotation}$ should be set to $2T$.

In the implemented controller this second method was used. As all the measured load currents showed sinusoidal behaviour, it can be expected that the compensation reference will be sinusoidal too.

Similarly v_o should be sinusoidal, and therefore $\overline{v_o}$ may be predicted by setting $t_{rotation}$ to $1\frac{1}{2}T$.

Two methods can be used to predict i_c . A sinusoidal i_c may be assumed, and i_c predicted as above. Alternatively the compensator may assume successful compensation in the following period, that is, the current i_c will follow the calculated reference current.

In the implemented controller the second means of compensation was used. It is more general than the first method, holding even under non-sinusoidal conditions.

5.8. Space vector modulation

Having obtained reference voltages in the previous section, a method is needed to generate the gating signals for the compensator. This section will deal with the (very popular) space vector modulation (SVM) method used in the compensator [42].

In a three-phase converter with the switches in each phase arm switching complementarily eight switching states are possible. Fig. 5-8 shows these vectors in the $\alpha - \beta$ plane. In the figure, the vectors are labelled in sequence (a, b, c), with a 1 denoting a closed top switch, and a -1 a closed bottom switch.

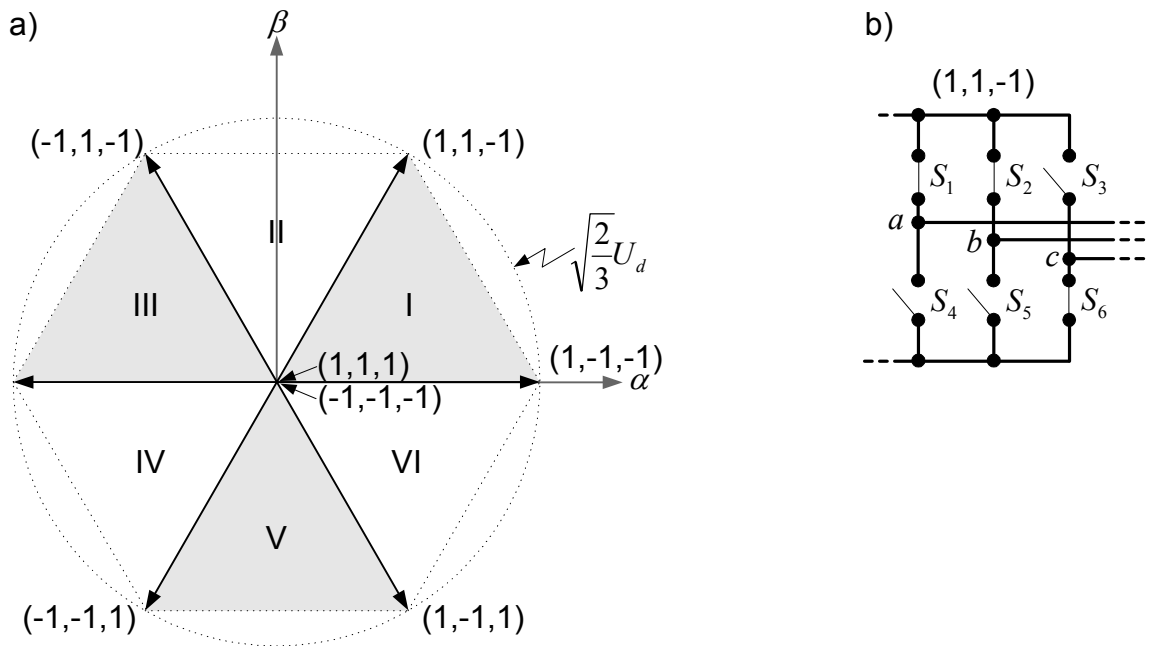


Fig. 5-8: SVM sectors and vectors

a) Vector plane, b) Example of a switching vector implementation (1, 1, -1)

The vectors divide the plane into six even sectors. A reference vector in any sector can be formed by decomposing the vector into three components: one component along each of the state vectors on the sector boundary and one component as the zero voltage vector. The zero vector may be realised by using either of the two possible switching vectors, but it will be shown that using both vectors minimises losses.

As an example, the decomposition of a vector in the first sector is shown in Fig. 5-9.

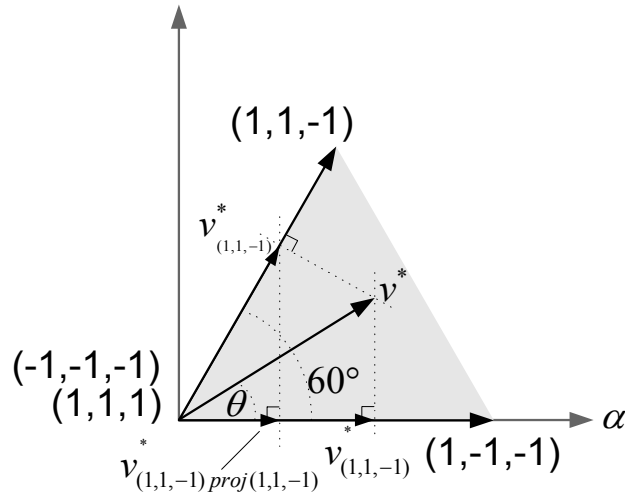


Fig. 5-9: Decomposition of reference voltage

The projection of the reference voltage, v^* , onto the $(1, 1, -1)$ vector is:

$$\|v_{(1,1,-1)}^*\| = \cos(60^\circ - \theta) \|v^*\|, \tag{Eq. 5-8}$$

and the projection onto the $(1, -1, -1)$ vector is:

$$\|v_{(1,-1,-1)}^*\| = \cos(\theta) \|v^*\|. \tag{Eq. 5-9}$$

Adding the vectors calculated in Eq. 5-8 and Eq. 5-9 will not produce the reference vector as the composing vectors are not orthogonal. The $v_{(1,1,-1)}^*$ vector clearly has a component in the α direction. Subtracting this component produces the required reference vectors:

$$\begin{aligned} \|(1,-1,-1)^*\| &= \cos(\theta) \|v^*\| - \cos(60^\circ) \|v_{(1,1,-1)}^*\| \text{ and} \\ \|(-1,-1,-1)^*\| &= \|v_{(1,1,-1)}^*\|. \end{aligned} \tag{Eq. 5-10}$$

Normalising these voltage vectors by the maximum possible vector length, $\sqrt{\frac{2}{3}}U_d$, produces the associated duty cycles.

The last step in duty cycle generation depends on the position of the reference vector. Three possibilities exist:

- If the reference vector lies within the shaded area in Fig. 5-9 the duty cycles will not add to unity, and the converter should be switched to the zero states for the remaining time.
- If the reference vector lies on the boundary of the shaded area, switching the converter to the zero state is not required and the calculated duty cycles may be used as is.
- If the reference vector lies outside the shaded area, the reference cannot be followed exactly because the compensator bus voltage is too low. In this case the best solution is to scale the duty cycles to produce maximum output voltage at the required angle.

The above algorithm results in optimal switching in the sense that a switching sequence can always be found in which only one phase arm has to switch at a time. This reduces the switching losses in the compensator. For the example above, the correct switching sequence would be:

$(-1,-1,-1)(1,-1,-1)(1,1,-1)(1,1,1)(1,1,-1)(1,-1,-1)(-1,-1,-1)$.

Note that both zero vectors have been used to obtain this result.

In the developed compensator the preceding method was implemented in the $\alpha - \beta$ plane. As the α and β axes are fixed relative to the eight compensator switching vectors, it is possible to decompose the reference voltage vector by precomputed sin and cos terms. This is much more computationally efficient, especially since the $\alpha - \beta$ representations of all the required vectors are already known from the reference generation algorithm.

Note that the method makes no assumptions as to the voltage of the bus relative to the rest of the system. Only the line to line voltages are of importance. This makes it possible to use the common mode voltage present between all the phases and earth to the advantage of the compensator. This results in better bus utilisation than would otherwise have been possible. For this scheme to be effective, the bus voltage must not be fixed relative to the rest of the system. Specifically, it would inhibit the operation of the method if the centre point of the bus was to be connected to ground.

5.9. Dead time effects

A severe pollutant of the output voltage, dead time, is not taken into account in the SVM algorithm. This section will explain and compensate dead time effects.

If both switches in a phase arm of a converter were to be set to the conducting state, this would short circuit the DC-bus and destroy the converter. Ensuring that the control circuit never attempts to switch both switches on together is not sufficient to guarantee this will not happen, as the turn-on

time, T_{on} , and turn-off time, T_{off} , of the switches results in the switches not being exactly controllable. Therefore, a short time, T_d , is usually included during which the controller tries to switch off both switches. If T_d is selected such that

$$T_d > T_{off} - T_{on}, \tag{Eq. 5-11}$$

the safety of the compensator is ensured. Because both T_{on} and T_{off} depend on circuit parameters and cannot exactly be known in advance, T_d is usually selected much larger than Eq. 5-11 implies.

Dead time is studied, and the following method of compensation suggested in [44] by Oh et al.

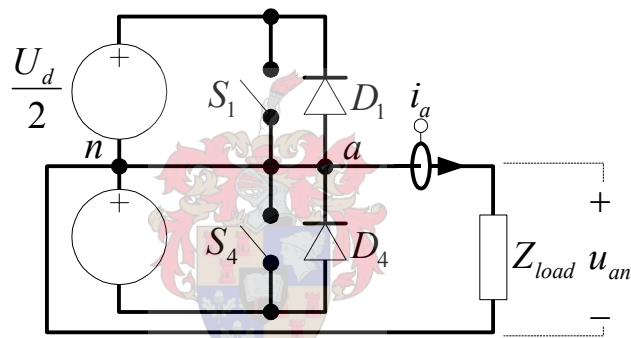


Fig. 5-10: Single phase arm of a three-phase converter

During dead time, both switches in Fig. 5-10 are in the off state. This means that the output voltage, u_{an} , depends only on the state of the diodes. The diode states depend on the current

direction. If i_a is positive, D_4 will be conducting fixing the output voltage at $-\frac{U_d}{2}$. Conversely, if

i_a is negative D_1 will be conducting fixing the output voltage at $\frac{U_d}{2}$.

During normal operation, the current direction also influences the output voltage. If i_a is positive,

either D_4 or S_1 must be conducting, producing an output voltage of either $-\frac{U_d}{2}$ or $\frac{U_d}{2}$. If i_a is

negative, either D_1 or S_4 must be conducting, producing an output voltage of either $\frac{U_d}{2}$ or $-\frac{U_d}{2}$.

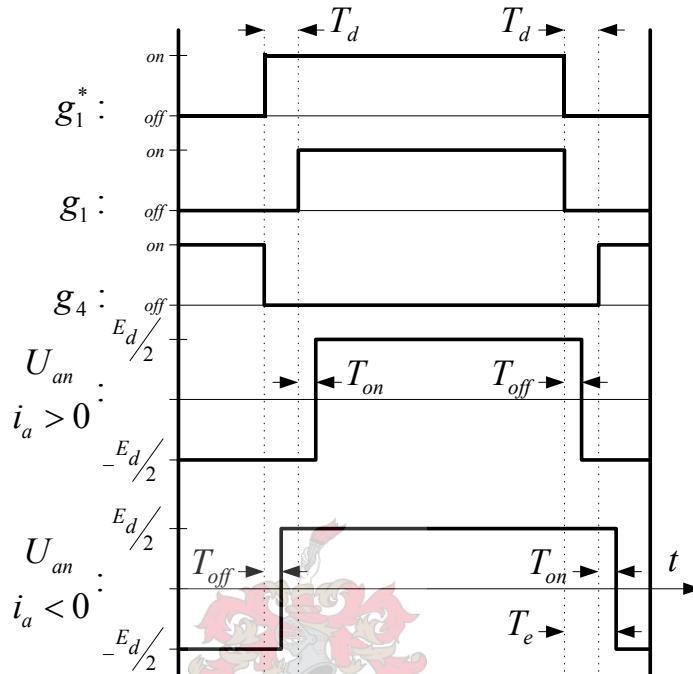


Fig. 5-11: Converter switching waveforms

Fig. 5-11 shows the influence of dead time on the converter switching waveforms. In the figure, g_i is the gate signal for switch i . g_i^* denotes the reference gate signal for switch i .

When the current is positive, the total time S_1 is on is less than that commanded by the reference. This results in a smaller output voltage than desired. The time S_1 is on is:

$$t_{a+} = t_a^* - ((T_d + T_{on}) - T_{off}), \tag{Eq. 5-12}$$

where t_a^* is the reference time associated with the reference gate signal g_i^* . As noted earlier, this is also the time that u_{an} is fixed at $\frac{U_d}{2}$, which motivates the choice of subscript in Eq. 5-13.

When the current is negative, the total time the voltage is fixed to the positive rail is given by

$$t_{a+} = t_a^* + ((T_d + T_{on}) - T_{off}) \tag{Eq. 5-13}$$

Oh generalises the above results by defining:

$$T_e \equiv (T_d + T_{on}) - T_{off} \tag{Eq. 5-14}$$

and stating that

$$t_{p+} = t_p^* - \text{sigum}(i_p)T_e \tag{Eq. 5-15}$$

where $p = a, b$ or c . The average phase output voltage can then be found to be

$$\overline{u_{pn}} = U_d \left(\frac{T_{p+}}{T_s} - \frac{1}{2} \right) \tag{Eq. 5-16}$$

with T_s as the switching period.

The above equations are valid for all time when $t_p^* > T_e$. If $t_p^* < T_e$, the reference is effectively too short for the converter to follow. The result is that the converter does not switch. The output voltage is determined by the current direction, and is constant for the whole switching period. As can be expected, the compensation of dead time is different for cases where such a narrow pulse exists in the control signal and cases where there is no such short pulse. Oh refers to the first case as “mode 1” and to the second of “mode 2”. This convention will be used in this discussion too.

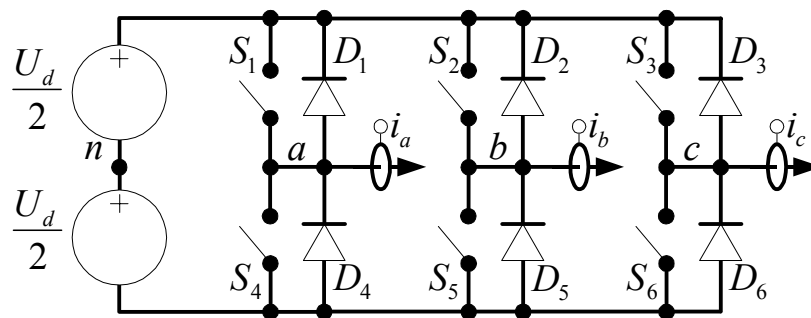


Fig. 5-12: Three-phase converter

Mode 1

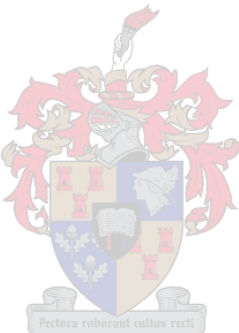
As an example, consider the case where

$$i_a > 0, i_b > 0 \text{ and } i_c < 0. \quad \text{Eq. 5-17}$$

The times when the compensator is driving high may be found from Eq. 5-12, Eq. 5-13 and Eq. 5-14:

$$\begin{aligned} t_{a+} &= t_a^* - T_e, \\ t_{b+} &= t_b^* - T_e \text{ and} \\ t_{c+} &= t_c^* + T_e. \end{aligned} \quad \text{Eq. 5-18}$$

The average phase to neutral voltages may now be found from Eq. 5-16

$$\begin{aligned} \overline{u_{an}} &= U_d \left(\frac{t_a^* - T_e}{T_s} - \frac{1}{2} \right), \\ \overline{u_{bn}} &= U_d \left(\frac{t_b^* - T_e}{T_s} - \frac{1}{2} \right) \text{ and} \\ \overline{u_{cn}} &= U_d \left(\frac{t_c^* + T_e}{T_s} - \frac{1}{2} \right). \end{aligned} \quad \text{Eq. 5-19}$$


The line to line voltages can be found from Eq. 5-19:

$$\begin{aligned} \overline{u_{ab}} &= U_d \left(\frac{t_a^* - t_b^*}{T_s} \right) = \overline{u_{ab}^*}, \\ \overline{u_{bc}} &= U_d \left(\frac{t_b^* - t_c^* - 2T_e}{T_s} \right) = \overline{u_{bc}^*} - U_d \left(\frac{2T_e}{T_s} \right) \text{ and} \\ \overline{u_{ca}} &= U_d \left(\frac{t_c^* - t_a^* + 2T_e}{T_s} \right) = \overline{u_{ca}^*} + U_d \left(\frac{2T_e}{T_s} \right). \end{aligned} \quad \text{Eq. 5-20}$$

From Eq. 5-20 the following may be noted:

- The average line to line voltage appearing between two phases that have the same current sign is not distorted by dead time effects;

- Compensation of the remaining two line to line voltages may be achieved by adding a compensating term to the reference switching time for each phase.

Setting

$$t_{aComp}^* = t_a^* + T_e, \quad \text{Eq. 5-21}$$

$$t_{bComp}^* = t_b^* + T_e \text{ and}$$

$$t_{cComp}^* = t_c^* - T_e$$

will provide the necessary compensation in this case. Following the same procedure as above for each possible combination of current directions it can be found that, in general, compensation in mode 1 follows Eq. 5-22:

$$t_{pComp}^* = t_c^* + \text{signum}(i_p)T_e. \quad \text{Eq. 5-22}$$

Eq. 5-22 differs from the compensating calculations in [44]. Both methods follow from the theory and are mathematically correct. The above equation allows for simpler implementation. Eq. 5-22 is also noted and supported by [45].

Mode 2

The compensation is slightly more involved in the presence of narrow pulses. Again compensation will be calculated following an example. The current directions are taken as the same as in Eq. 5-17, but a narrow pulse is said to exist in phase a.

The times the converter is driving high in each phase are:

$$t_{a+} = 0, \quad \text{Eq. 5-23}$$

$$t_{b+} = t_b^* - T_e \text{ and}$$

$$t_{c+} = t_c^* + T_e.$$

The average phase voltages may be calculated to be:

$$\overline{u_{an}} = -\frac{U_d}{2}, \quad \text{Eq. 5-24}$$

$$\overline{u_{bn}} = U_d \left(\frac{t_b^* - T_e}{T_s} - \frac{1}{2} \right) \text{ and}$$

$$\overline{u_{cn}} = U_d \left(\frac{t_c^* + T_e}{T_s} - \frac{1}{2} \right);$$

and the average line to line voltages:

$$\overline{u_{ab}} = U_d \left(\frac{-t_b^* + T_e}{T_s} \right) = \overline{u_{ab}}^* - U_d \left(\frac{t_a^* - T_e}{T_s} \right), \quad \text{Eq. 5-25}$$

$$\overline{u_{bc}} = U_d \left(\frac{t_b^* - t_c^* - 2T_e}{T_s} \right) = \overline{u_{bc}}^* - U_d \left(\frac{2T_e}{T_s} \right) \text{ and}$$

$$\overline{u_{ca}} = U_d \left(\frac{t_c^* + T_e}{T_s} \right) = \overline{u_{ca}}^* + U_d \left(\frac{t_a^* + T_e}{T_s} \right).$$

As the reference t_a^* is too short to be switched, compensation must be done in the other two phases. From Eq. 5-25 it is evident that the necessary compensation is:

$$t_{bComp}^* = t_b^* - (t_a^* - T_e) \text{ and} \quad \text{Eq. 5-26}$$

$$t_{cComp}^* = t_c^* - (t_a^* + T_e).$$

Proceeding in this way, the necessary compensation terms for all possible situations can be generated. In general, when the narrow pulse is in a phase with positive current, the compensation for phases with positive current should be

$$t_{pComp}^* = t_p^* - (t_{narrow\ pulse}^* - T_e), \quad \text{Eq. 5-27}$$

and the compensation for phases with negative current:

$$t_{pComp}^* = t_p^* - (t_{narrow\ pulse}^* + T_e). \quad \text{Eq. 5-28}$$

When the narrow pulse is in a phase with negative current, the compensation for phases with positive current is:

$$t_{pComp}^* = t_p^* - (t_{narrow\ pulse}^* - 3T_e), \quad \text{Eq. 5-29}$$

and the compensation for phases with negative current is:

$$t_{pComp}^* = t_p^* - (t_{narrow\ pulse}^* - T_e). \quad \text{Eq. 5-30}$$

This correlates with the results in [44]. Gous [45] goes on to consider the case of a three-phase four-wire system. The systems under consideration here is three-phase three-wire only, thus the four-wire case need not be investigated in this thesis.

Gous also explains that, in dead time calculations, the reference currents should be used rather than the measured currents. This problem is analogous to the one found in 5.7, where the measured values were demonstrated to be outdated by the time they were used and implemented. This is important in the region of current zero points, as the compensation varies drastically with current direction.

5.10. Generation of switching signals

Having computed the compensated duty cycles, the last task is the generation of the compensator gating signals. In the developed compensator, this is done simply by comparing the reference duty cycles with a unity amplitude triangular wave. When the duty cycle reference is larger than the triangular wave, the top switch is switched on. When the triangular wave is larger, the bottom switch is switched on.

Switching the compensator in this way guarantees that the average output currents will follow the references. This can be seen from Fig. 5-13. When the top switch is on, the current in that phase increases. When the top switch is off, the current in that phase decreases. Assuming that the compensator followed the reference in the previous switching period, the current will start at the reference, and end on it. During the period the average current equals the average reference current from symmetry.

When the current is not equal to the reference at the start of the period, the average current will not equal the average reference current. From the next period onwards operation should again be as described above.

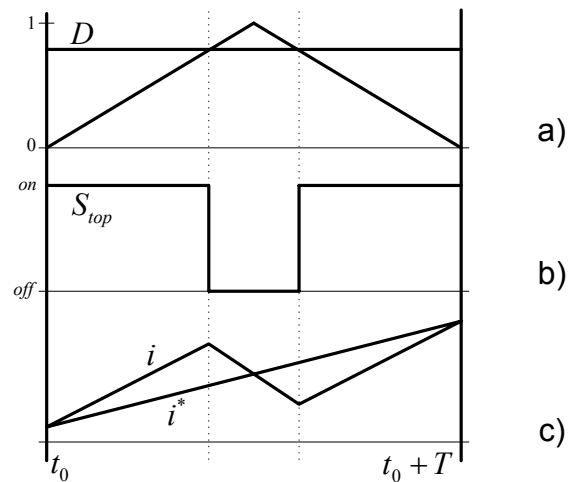


Fig. 5-13: PWM generation from duty cycle reference

a) Duty cycle (D) and triangular wave, b) Switch gating signal, c) Reference and output current

5.11. Chapter summary

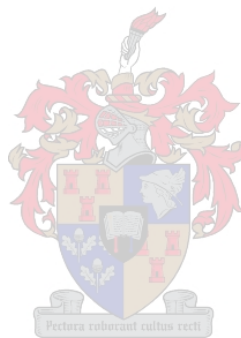
In this chapter the practical side of the implementation of the algorithms developed in the previous chapter was investigated. Several methods for generating switching signals from reference currents were evaluated.

P-I control offers a well defined harmonic spectrum, and is relatively easy to implement. Their dynamic response is inferior to both hysteresis and deadbeat controllers. Hysteresis controllers are very robust and offer excellent response, but are unpredictable in frequency. Deadbeat control offers dynamic response nearly as good as that of hysteresis control combined with the well defined harmonic qualities of P-I control. Accordingly, deadbeat control was selected for use in this compensator.

The implementation of deadbeat control was treated in detail. Special consideration was given to the effects of implementing this method in a discrete controller. It was shown how the current references, voltage and current measurements can be predicted to minimise the effect of time delays within the controller.

Space vector modulation was presented as a method to compute duty cycles from voltage references. It was noted that this method allows good DC bus utilization by allowing the injection of common mode voltage. The effects of the unavoidable dead time in the switching process were investigated and compensated.

Finally, the generation of switching signals by comparing duty cycles to a reference triangular wave was described. This method allows the converter output current to follow the reference in an average sense over each switching period.



CHAPTER 6: CASE STUDIES

Having developed a flicker compensator in the previous chapters, the compensator had to be tested for applicability and effectiveness in actual flicker load situations. In this chapter, the proposed compensator is simulated compensating three practical loads. Each load is either a source of flicker problems currently, or is an example of a load type that could present problems if such a load was to be installed on a weak line. For each load, a short background is given. A model of the load is built in the Simplorer/MATLAB simulation environment, and the load is simulated in detail. The developed compensator is used to compensate the load using different compensation parameters. Finally, the results of the simulations are presented and evaluated.

6.1. Model of the compensator

The simulation of the compensator controller follows the theory presented in the previous chapters closely. In this simulation it was easier to use the Simplorer internal modelling language than to build the whole controller using the graphical interface. Special care was taken to model the time delays in the controller accurately.

The converter hardware simulated was based on a three-phase controllable converter available in the laboratory. This converter was later also used in hardware simulations conducted, reported in Chapter 8.

In order to compensate at high voltages (such as the 11 kV found in the first load study below) an injection transformer had to be inserted between the compensator and the line. To enable the injection of active current, the compensator was supplied with a large bus capacitor. This capacitor fulfilled both the roles of bus regulator and energy storage. The converter IGBT (insulated gate bipolar transistor) modules were modelled using ideal switches and diodes. The switch gate signals were suitably delayed to model dead time effects.

The converter simulation is presented in Fig. 6-1, with the simulation parameters summarised in Table 6-1. Where the value of a parameter is only given for one phase, the simulation may be assumed to be symmetrical. The simulation script for the controller can be found in Appendix D.1.

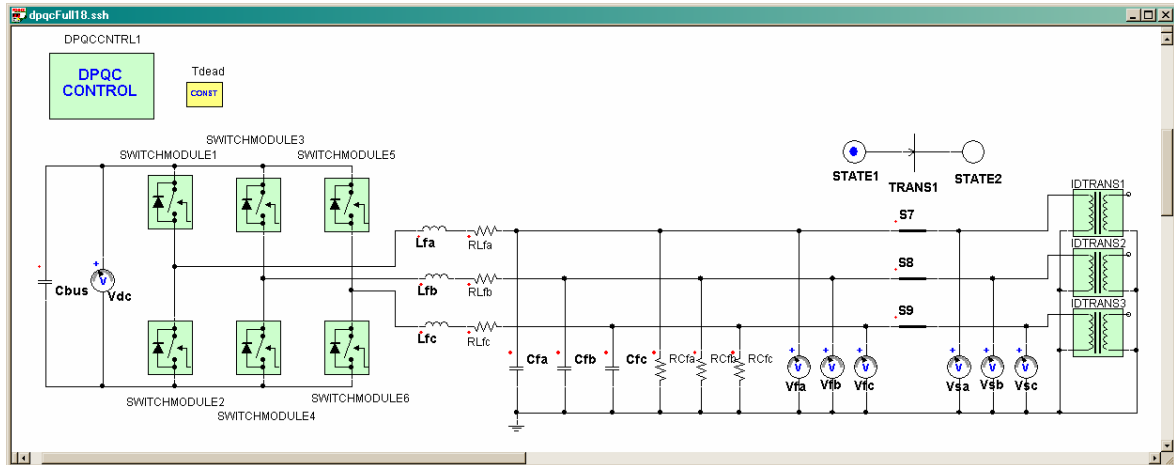


Fig. 6-1: Simplorer simulation of the compensator

Parameter	Value
L_{fa}	400 μ H
R_{Lfa}	0 Ω
C_{fa}	200 μ F
R_{Cfa}	10.0 k Ω
C_{bus}	2.00 F
Transformer ratio	221 : 8
T_{dead}	5.00 μ s

Table 6-1: Parameter values for Simplorer simulation of the compensator

6.2. Case study I: Three-phase arc welder

The first industrial load to be studied was a three-phase welder in the Blackheath industrial zone near Cape Town. This welder is used to weld wire for fencing, etc. The welder consists of 48 electrodes mounted on a moveable welding arm. The wire to be welded is rolled underneath the welder, and the electrodes are lowered to weld the joints as necessary. This happens at a frequency of very nearly 1 Hz.

Each time the electrodes are lowered to weld, the arcs formed cause a surge in the power drawn from the utility. Even though the lines feeding Blackheath have a relatively high fault level of 338 MVA, the line into Blackheath itself has a fault level of only 120 MVA. This implies relatively high line impedance, which, in turn, means that the surges in power drawn by the welder pollute the

voltage waveform over the whole Blackheath area [46]. A single-line diagram of the situation is presented in Fig. 6-2.

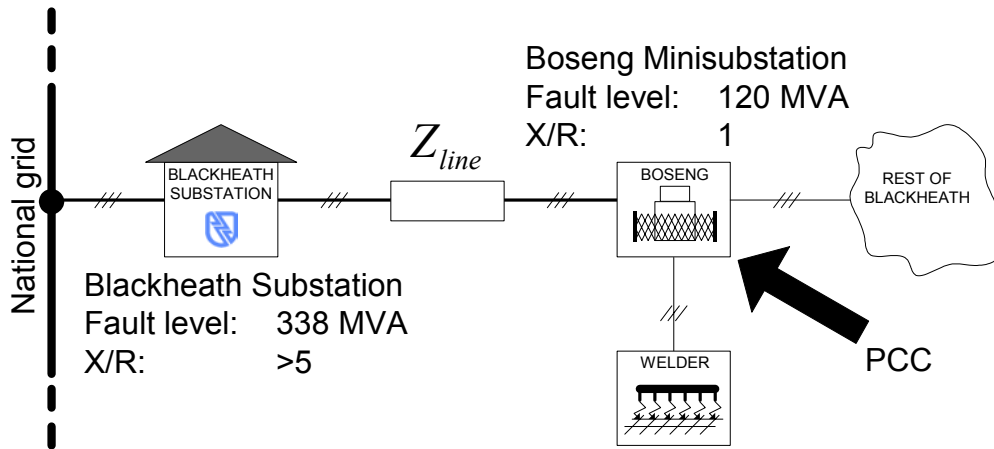


Fig. 6-2: Single-line diagram of the grid around the welder

The actual plant was not accessible, while Boseng mini-substation had no facilities for measuring. Measurements were therefore taken at Blackheath substation itself. Although this meant that the full Blackheath load was measured instead of the flicker load only, the influence of the welder on the Blackheath distribution network is so marked that its effect could be appreciated without difficulty. One line to line voltage and one line current are shown in Fig. 6-3.

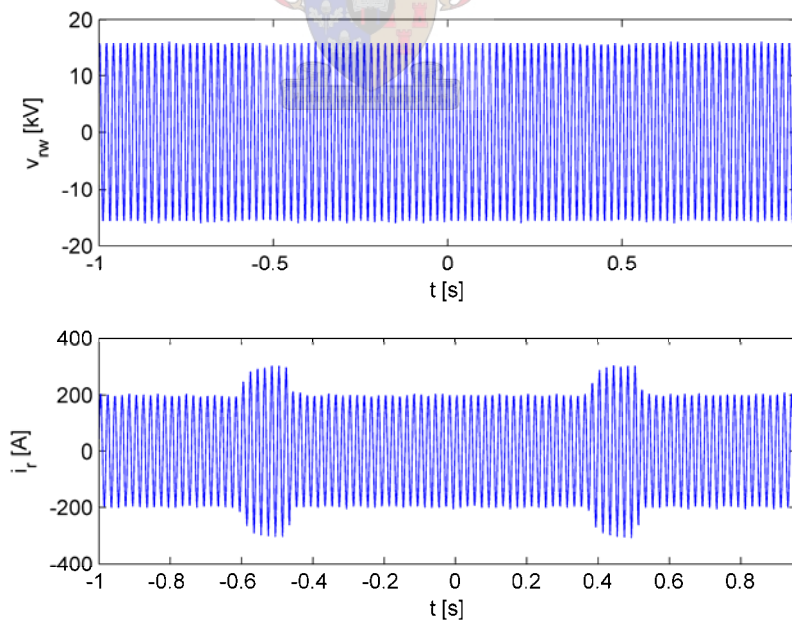


Fig. 6-3: Voltage and current in Blackheath

In Fig. 6-3 the rises in load current can be seen very clearly. It varies between about $140 A_{\text{rms/phase}}$ when the welder is not welding to $210 A_{\text{rms/phase}}$, a variation of $70 A_{\text{rms/phase}}$. Although the fault level at the substation is quite high, a small depression can be seen in the voltage each time the welder commences a welding cycle. The effect at a point in the system with reduced fault level such as Boseng, the PCC, can be expected to be much worse.

Two other observations are relevant:

- The load was found to be periodic with period 1.02 Hz.
- The load is piecewise sinusoidal, enabling the usage of traditional load parameters such as S and Q in their usual sense.

Because of the second point above the power variation can be summarised in Table 6-2, where the powers have been computed and decomposed in the usual way using the angle difference between current and voltage to compute the power angle.

	P	Q	S
Not welding	2.5 MW	1.3 MVar	2.8 MVA
Welding	3.1 MW	2.5 MVar	4.0 MVA
Δ	0.6 MW	1.2 MVar	1.2 MVA

Table 6-2: Power variations measured at Blackheath

Table 6-2 shows that a substantial part of the variation in power is reactive. This leads one to suspect that adequate compensation might be possible with little or no energy storage.

The load could be simulated relatively easily by calculating the load parameters under welding and non-welding conditions. These values were implemented by switching impedance in and out of the circuit in the manner observed in Fig. 6-3.

Parameter	Value
R_{line}	0.710
X_{line}	0.710
Z_{perm}	$43.95 \angle 27.21^\circ$
Z_s	$77.17 \angle 57.68^\circ$

Table 6-3: Parameter values for welder simulation

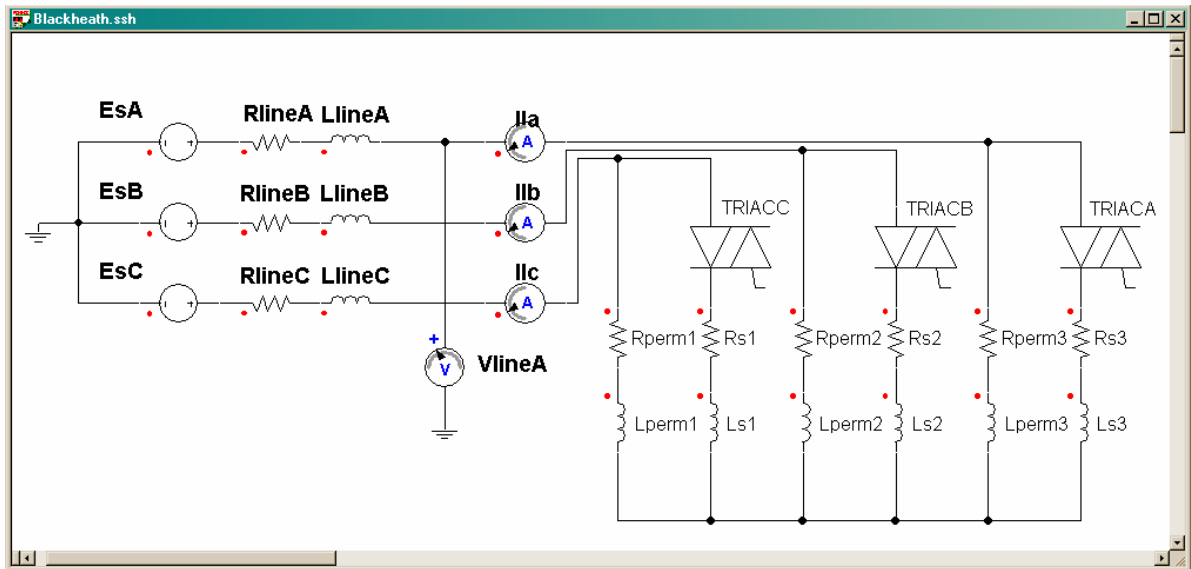


Fig. 6-4: Simplorer simulation of the three-phase welder as load

The line impedance is easily found from the three-phase fault level:

$$|Z_{line}| = \frac{|V_{ln}|^2}{|S_{1\phi fault}|} = \frac{\left(\frac{1}{\sqrt{3}} V_{ll}\right)^2}{\left|\frac{1}{3} S_{3\phi}\right|} = 1.008\Omega \tag{Eq. 6-1}$$

$$|R_{line}|^2 + |X_{line}|^2 = |Z_{line}|^2 \text{ and } \left|\frac{X_{line}}{R_{line}}\right| = 1, \tag{Eq. 6-2}$$

$$\Rightarrow 2|R_{line}|^2 = 2|X_{line}|^2 = |Z_{line}|^2$$

Solving for R_{line} and X_{line} gives the values in Table 6-3.

With Z_{line} known, the load parameters could be calculated:

$$Z_{perm} = \frac{V_{ln}}{I_l} - Z_{line} = \frac{\frac{1}{\sqrt{3}} V_{ll}}{I_l} - Z_{line}, \tag{Eq. 6-3}$$

with the non-welding current value used for I_l . Using the welding current in Eq. 6-3 gives the welding impedance value:

$$Z_{welding} = \frac{V_{ln}}{I_l} - Z_{line} = \frac{1}{\sqrt{3}} \frac{V_{ll}}{I_l} - Z_{line} . \tag{Eq. 6-4}$$

The switched impedance Z_s may be found by impedance division.

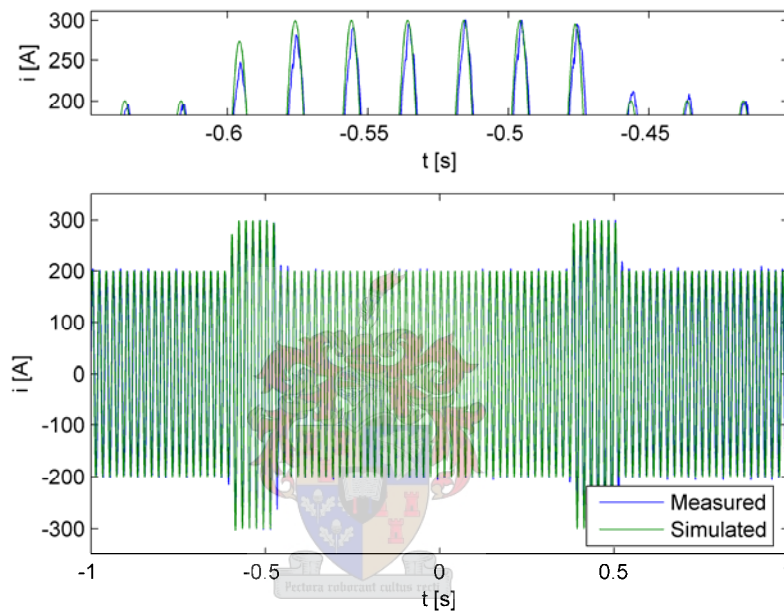


Fig. 6-5: Comparison of measured and simulated current

The validity of the simulation is shown in Fig. 6-5. The simulated current closely emulates the measured current. The chief difference is that the simulated current shows a sharper step in the current when the welding commences. This should generate slightly worse flicker problems than experienced in Blackheath. The simulation may therefore be regarded as conservative.

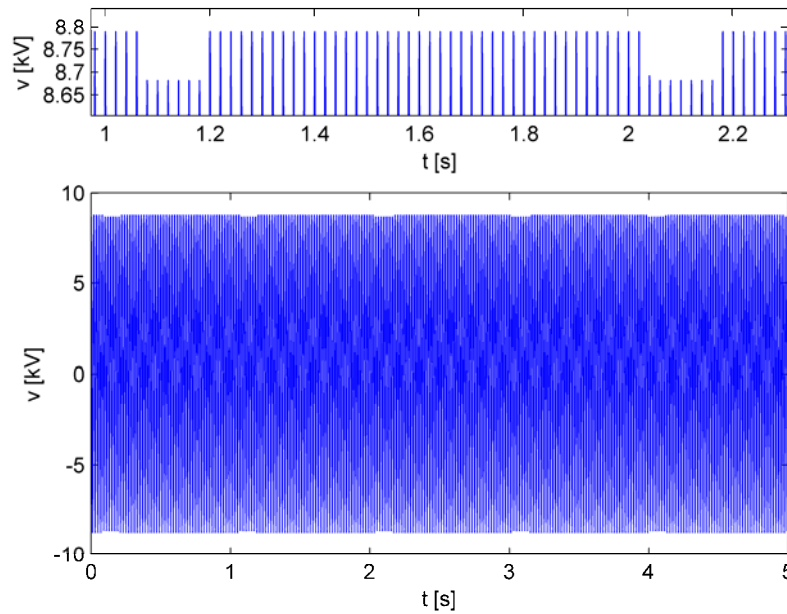


Fig. 6-6: Effect on the voltage at the PCC

Using the simulation it is possible to compute the voltage variation to be expected at Boseng mini-substation, the PCC. This is shown in Fig. 6-6. The voltage is clearly depressed each time the welder operates. The depression is of the order of 1.732%, well above the threshold of 0.471% specified in Table 2 of the IEC flicker meter standard [02] as the threshold of perception for humans. Severe flicker is to be expected. This is confirmed by the P_{st} value of 1.42 calculated using the software flicker meter developed in 2.3. It confirms that the welding plant is the major contributor to the flicker problem in the Blackheath area. Compensating this load will mitigate the flicker in the area to acceptable levels.

The whole simulation is presented in Fig. 6-7. Simulation results are presented in Fig. 6-8.

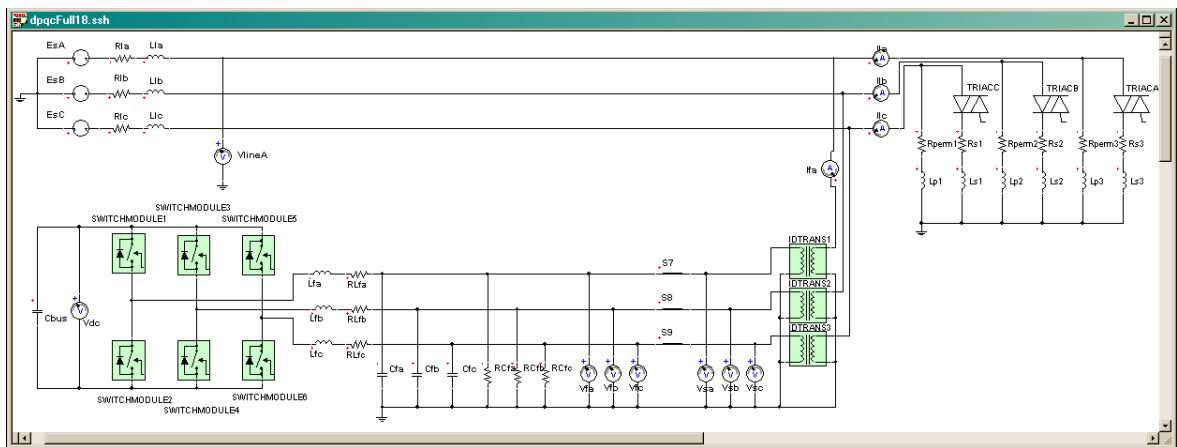


Fig. 6-7: Simulation of three-phase welder with compensator

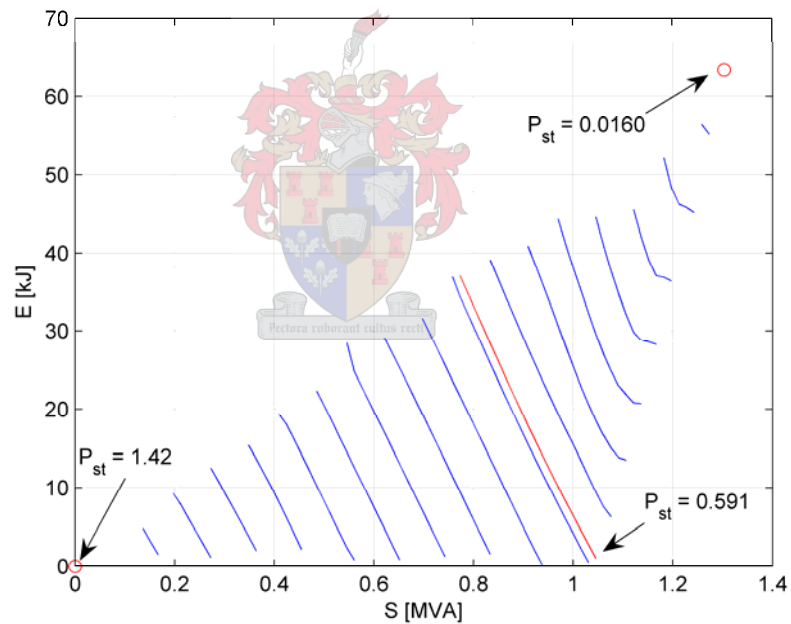
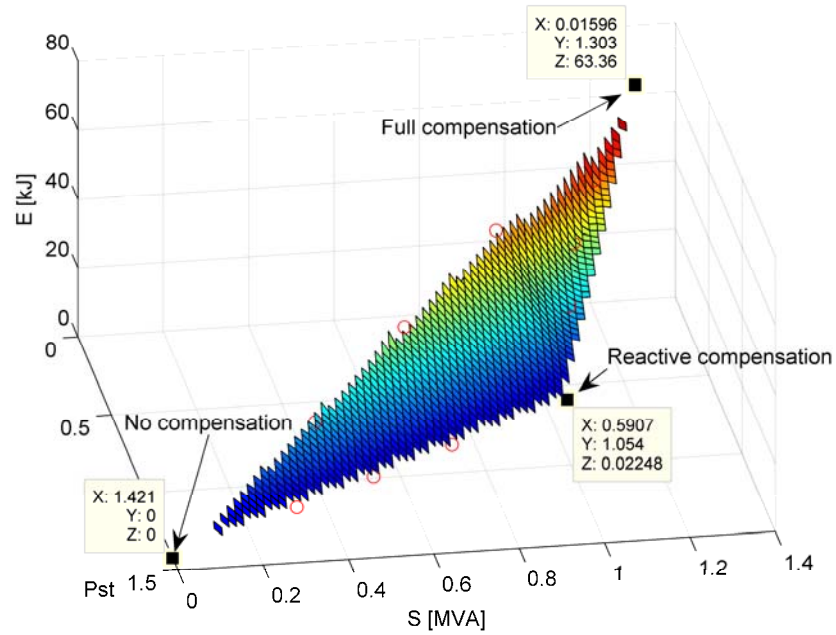


Fig. 6-8: Three-phase welder simulation results

In the top figure in Fig. 6-8 P_{st} , power and energy requirements are plotted against each other in a three-dimensional coordinate system. The plot was generated by running the simulation several times, each time with a different gain set on the compensator d (active power) and q (reactive power) channels. Simulation started with both D (the d channel gain) and Q (the q channel gain) set to zero. Ramping Q to unity evaluated the reactive power compensation abilities of the

compensator. This gave the points on a straight line between the points marked “No compensation” ($D = 0, Q = 0$) and “Reactive compensation” ($D = 0, Q = 1$). Ramping D too gave points between the points marked “Reactive compensation” ($D = 0, Q = 1$) and “Full compensation” ($D = 1, Q = 1$). These simulations show the influence of active power in the compensation efforts. Random values of D and Q gave points in a plane bounded by these two lines. This plane represents the compensation options possible using synchronous reference frame filtering. The plane was rendered using the `meshgrid`, `griddata` and `surf` functions in MATLAB.

The surface in the figure is expected to be smooth. The IEC flicker meter incorporates many non-linearities, but is not discontinuous. Also, progressively better compensation is expected as the compensation effort, and therefore the power and energy ratings, is increased.

A three-dimensional plot can provide good insight into the behaviour of the compensator. Unfortunately, the orientation of the plane in three-dimensional space can be difficult to visualise if only the three-dimensional plot is provided. To ease the interpretation of the simulation results, the bottom plot in Fig. 6-8 was drawn by taking constant P_{st} contours on the surface of the three-dimensional plot. In this plot the rightmost point on each line represents compensation with the minimum amount of active power possible at a given level of reactive compensation ($D = 0, Q = Q_1$). As active power injection is increased, the locus follows the specific contour to the left. As the active power injected increases, the total power needed decreases. Thus, at any point to the left of the rightmost point on any contour we can state: $D = D_2, Q = Q_2 < Q_1$. The red line represents the P_{st} level of the contour starting on $D = 0, Q = 1$, that is to say, full reactive compensation.

The simulation shows that excellent compensation can be attained using the methods simulated. The power values from the simulation correlate very well with the values in Table 6-2. As an example: For full compensation, the simulation gives the compensator rating as 1.05 MVA, which is in the region of the 1.20 MVA power variation measured allowing for losses and compensation error. Full compensation reduces the flicker levels from a P_{st} of 1.42 to a very low 0.016 (98.8 % compensation).

The compensator can be implemented with practically zero energy storage. A P_{st} of 0.591 could be obtained (58.3% compensation). This shows that the $d - q$ method effectively decouples active and reactive energy flow. In a practical compensator inevitable losses in the compensator would

degrade the performance, causing the compensator to inject some active power even when set to inject reactive power only ($D = 0, Q = 1$). This was verified by noting the compensator response when increasing R_{Lfa} .

The compensator can be forced to inject reactive power only by tightly regulating the DC bus. This can be done by adding a term to the compensator reference causing the bus to be charged from the line when undercharged, or discharged into the line when overcharged. Such regulation will prevent energy flow to or from the bus, but will impact negatively on the compensation achieved.

Note that the angle of the contour lines in the bottom plot is nearly -45° . This means that, for the same P_{st} level, a reactive compensator will have approximately two times the power rating of a compensator compensating both actively and reactively.

By using Fig. 6-8 a compensator can be designed to meet specific requirements. For instance, selecting a specific value for the P_{st} gives a plot of power vs. energy. The power rating, energy storage needed and compensator parameters may thus be found. Using a cost function, the locus of minimum cost per unit of flicker reduction can be found, giving the most cost effective solution to the problem.

As an example of the economic insight gained from the simulation: Although full compensation reduces the flicker level by 98.8 %, this requires a power rating of 1.30 MVA. Compensation of 58.3 % can be attained with a power rating of only 1.05 MVA, and very little energy storage (0.0225 kJ vs. 63.4 kJ for the previous case). As sufficient compensation ($P_{st} = 0.591$) is attainable using minimal energy storage, it is recommended that the compensation be implemented in this way.

6.3. Case study II: Sawmill

The second load to be investigated was a mechanical saw in a sawmill outside Stellenbosch. The sawmill uses frame and rotary saws, and this saw is of the frame type. A set of vertical blades are moved up and down to saw the wood inserted from a horizontal direction. The saw is mechanically constructed such that sawing only occurs when the blades are moving downwards.

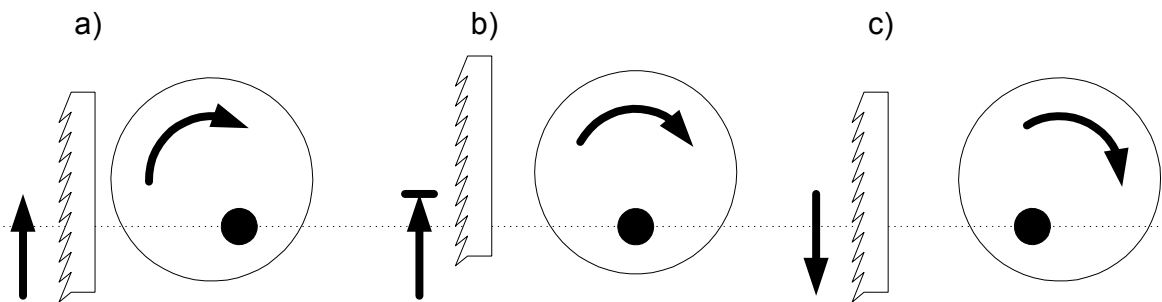


Fig. 6-9: Interaction between saw flywheel and blade

Power is delivered by a three-phase induction motor connected to an off-centre flywheel. The flywheel is used to store mechanical energy in an effort to reduce the peak power needed from the motor during sawing. When the saw is not sawing, (blades moving upwards) the flywheel is rotated to the point of maximum stored mechanical energy. When the saw is sawing, this energy is released into the load.

As the flywheel acts as a power filter, it reduces the peak currents drawn by the motor when sawing. Unfortunately the reverse is true when the saw is running without wood.

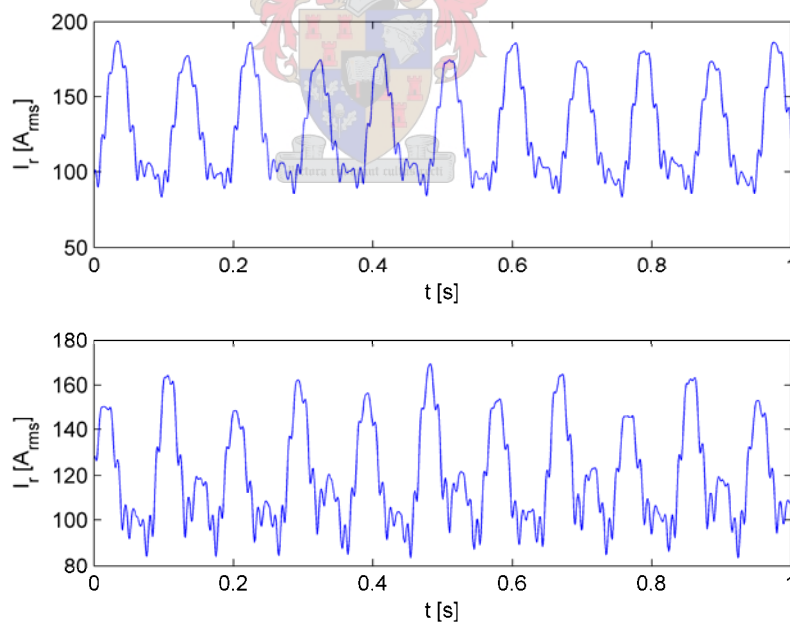


Fig. 6-10: Sawmill RMS current
While sawing wood (top), without wood (bottom)

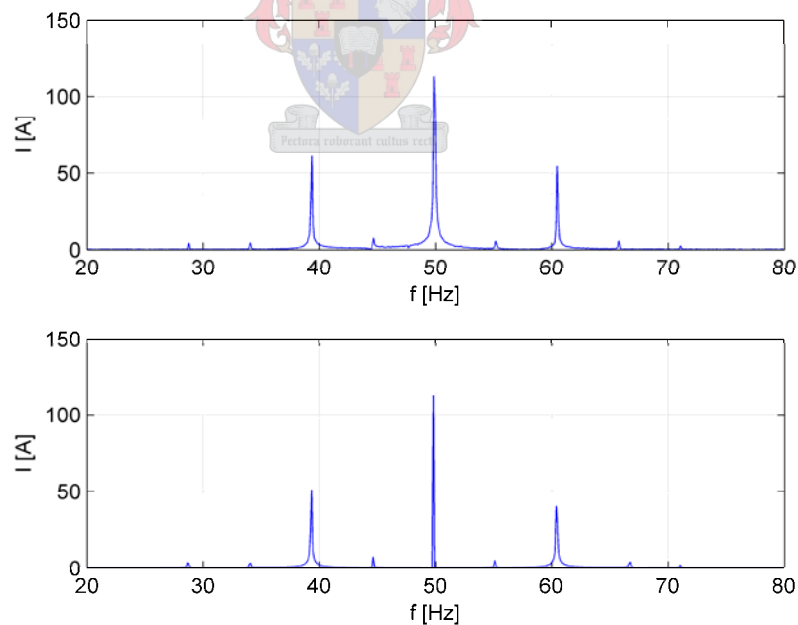
Fig. 6-10 clearly shows that the flywheel causes considerable distortion in the current drawn when the saw is without wood. Also, when the saw is sawing wood, the flywheel does not remove all

current variation. This is because of an inevitable compromise in the mechanical design: Had the flywheel been placed further off-centre the load variation when sawing would have been reduced, but the variation when not sawing would have been even worse than at present.

The saw described is connected to a relatively strong bus, and therefore does not constitute a flicker problem at present. Such a load may cause flicker problems, however, if connected to a weaker bus. This motivates the study of possible compensation. As the expected flicker is worse when the saw is sawing, this case will be considered in the rest of this discussion.

Fig. 6-12 shows the sawmill current to be quasi-periodic with a period of about 10 Hz. This is to be expected: All phenomena associated with the rotating flywheel should be periodic, and the wood fed to the saw is of a roughly constant diameter. The only non-periodic element in the load is the quality of the wood.

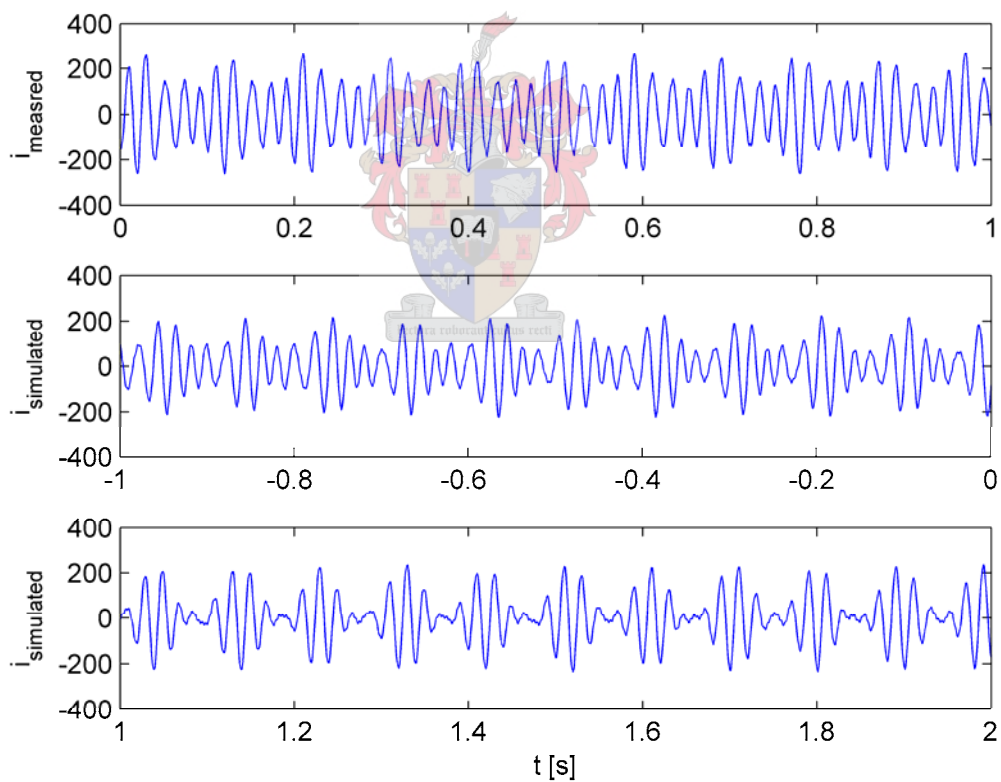
The quasi-periodic nature of the load lead to an analysis in the frequency domain. This produces the spectrum in Fig. 6-11. From this Fourier analysis a simulation could be built, resulting in the waveforms in Fig. 6-11 and Fig. 6-12. The terms in the Fourier analysis were calculated in MATLAB using the `fft` function, and are presented in Table 6-4. The simulation itself was run in Simplorer.



**Fig. 6-11: Sawmill current spectrum
Measured (top), Fourier simulated (bottom)**

f [Hz]	Amplitude [A]	Angle [°]
28.75	4.621	348.6
34.05	4.778	144.3
39.35	60.81	324.0
44.65	12.00	7.168
49.85	113.0	0.000
55.16	10.00	284.3
60.46	54.62	330.5
66.76	5.895	142.9
71.06	2.740	164.6
249.7	6.691	358.8

Table 6-4: Fourier coefficients for sawmill simulation



**Fig. 6-12: Sawmill current waveform
Measured (top), Fourier simulated (middle & bottom)**

The Fourier simulation produced good results at some times, but unsatisfactory results when the simulation was extended over longer simulation periods. Viewing the middle graph in Fig. 6-12, the

simulation can be seen to agree closely with the measured signal. The simulation models the load current accurately under both normal current and disturbance current conditions. Looking at the same simulation a second later, (bottom graph) a very unsatisfactory representation of the load is seen. While the disturbance current still seems to be modelled to some degree of accuracy, the load under normal conditions does not resemble the measured current at all. This is because of the difficulty in determining the Fourier coefficients accurately. The fact that the load is not completely periodic contributed to this difficulty.

One way of improving the Fourier analysis is to do the analysis on a larger data set with a much higher sampling frequency. Even better results were obtained simulating the load quite differently.

The load may be modelled as a constant rotational load modulated by a disturbance. This disturbance corresponds to the combination of the off-centre flywheel effect and the sawing action of the saw. In this second simulation, the load current was therefore modelled as a base current modulated by such a disturbance, D . The current was calculated using

$$i_{load} = (A + DB)\sin(2\pi ft + D\delta), \quad \text{Eq. 6-5}$$

with A the base current amplitude, B the disturbance amplitude and δ the disturbance phase change. The disturbance current was modelled as an on/off function filtered by a low-pass filter. Even though the load changes rapidly when the machine switches from sawing to not sawing, the filter was included as the flywheel filters out all the higher frequencies in this sudden change.

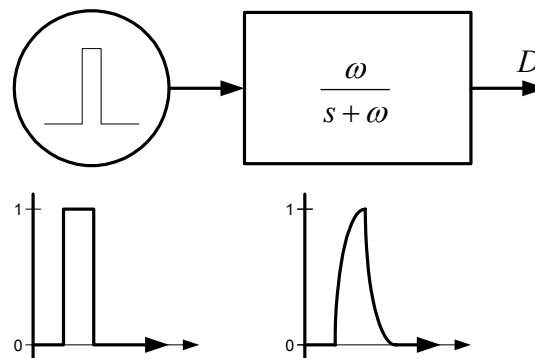


Fig. 6-13: Implementation of the disturbance function

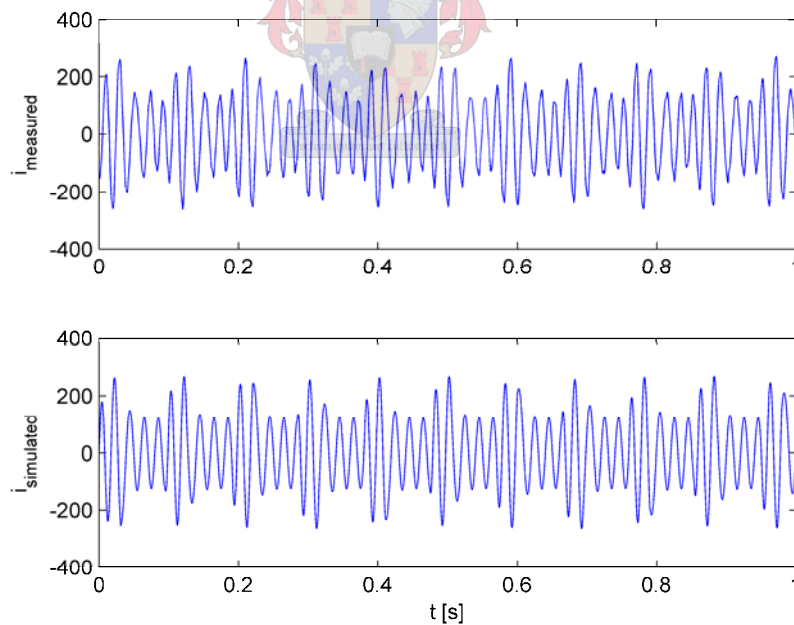
It can be seen in Fig. 6-13 that the disturbance function has four defining parameters: t_{onD} , the length of the disturbance pulse; f_d , the disturbance frequency and ω , the low-pass filter cutoff

frequency. The values for these parameters and the parameters in Eq. 6-3 were found by tuning the simulated load current to the measured load current, and are presented in Table 6-5.

Parameter	Value
f	50 Hz
A	122
B	148
δ	56.25°
t_{onD}	31 ms
f_d	10.55 Hz
ω	$20 \cdot 2\pi$ rad./s

Table 6-5: Sawmill simulation parameters

The simulated current is shown in Fig. 6-14. Comparing this figure with the Fourier simulation in Fig. 6-12 shows that this method is clearly superior in the accurate representation of the sawmill load. The simulated current was also compared to the measurements in the frequency domain, and in the time domain using RMS signals. These comparisons further validated the simulation results.



**Fig. 6-14: Simulated sawmill current waveform
Measured (top), Simulated (bottom)**

The compensating methods developed in this thesis were applied to the simulated load in Simplorer. As the sawmill is not a flicker problem at present, the line impedance was chosen to produce flicker of approximately one unit P_{st} . The X/R ratio was taken as unity, a common value for industrial sites like the sawmill under consideration. Unity P_{st} was reached at a line impedance value of 6.01 mΩ. The bus voltage at the sawmill is 400 V line to line. No injection transformer is required at this relatively low voltage. In all other respects, the simulated compensator was identical to the one used in the previous simulation.

The simulation results are presented as Fig. 6-15. The plots in the figure were drawn following the same methodology as was used for the welder results.

The simulation results bear strong resemblance to the results achieved when compensating the three-phase welder. Again the compensation options form a plane in three-dimensional space.

The most marked difference lies in the application of reactive compensation. The figure shows it that reactive compensation alone has very little effect on the flicker situation. Adding active compensation dramatically improves the compensation effort. The maximum compensation attainable when using only reactive compensation is a P_{st} of 0.845, which is only marginally better than the uncompensated load (15.5 % compensation). A compensator supplying active and reactive compensation produces a flicker level of approximately 0.0239 (97.6% improvement) at a power level of 73.4 kVA and requiring a very small energy storage of 1.084 kJ. In this load, it is imperative that the compensator applied should be able to supply both active and reactive power to the load to mitigate flicker effectively.

It could be argued that the inability of reactive compensation alone to mitigate flicker successfully is not a result of any intrinsic property of the load, but is rather a result of the large area of compensation options that the synchronous reference frame method is unable to provide. It may be meaningful to investigate more complex control for the compensation of this load if such controllers would enable a greater choice of power/energy combinations.

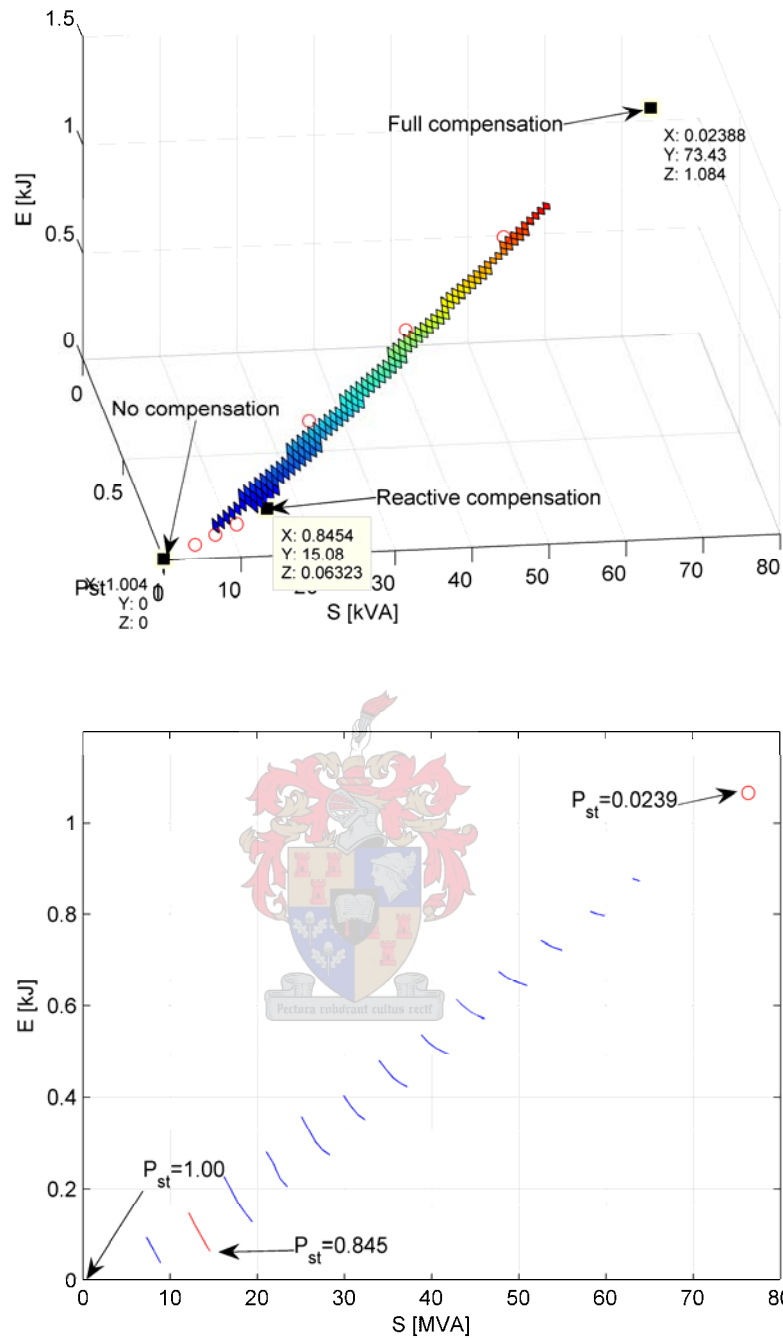


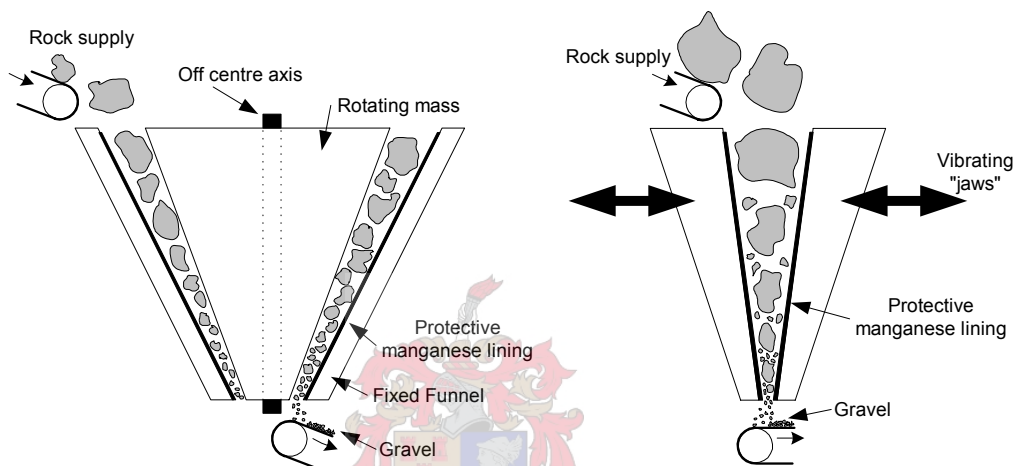
Fig. 6-15: Sawmill simulation results

6.4. Case study III: Rock crusher

The third load studied is a rock crusher in a gravel production plant. Different types of gravel demand the use of different types of crushers. The plant studied used two types of crushers common in industry.

Rotary crushers consist of a rotating mass within a conical housing. The mass is rotated round an off-centre axis. As the mass moves away from the housing, new rock falls into the cavity created. When the mass moves towards the housing the rock is crushed. As the rock moves downwards, the pebble size becomes smaller and smaller until the desired size is reached and the gravel falls from the crusher.

As the size of the rocks that have to be crushed varies continuously, the load on the crusher motor is continuously varying too. The rotating mass forms a flywheel that filters some of this distortion, reducing the varying currents drawn and the flicker produced.



**Fig. 6-16: Two types of rock crushers
(Rotary crusher (left); Jaw crusher (right))**

As its name suggests, a jaw crusher consists of two mechanical jaws that “chew” the rocks supplied to it to form gravel. The crusher jaws vibrate, crushing the rock between them. As is the case with the rotary crusher, the pebble size is reduced progressively as the rock descends through the crusher. The jaw crusher studied employs a large flywheel to help filter the load on the motor in a manner reminiscent of the flywheel and saw interaction in 6.3. When the crusher jaws are opening the flywheel speeds up, implying storage of mechanical energy. When the jaws are closing, this energy is used to aid the motor in the crushing process.

Of the two crushers studied, the jaw crusher produced the most severe flicker, and therefore it is for this crusher that compensation will be developed. The flicker measured at the machines themselves was 4.411 and 3.459 for the jaw and rotary crushers respectively. The measured RMS currents are shown in Fig. 6-17.

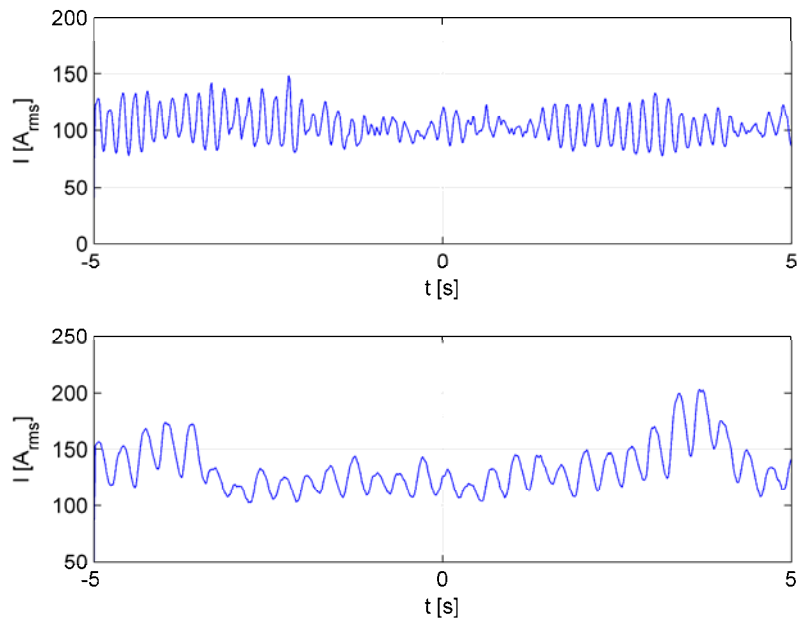


Fig. 6-17: Crusher RMS currents
Rotary crusher (top); Jaw crusher (bottom)

In contrast with the sawmill, the jaw crusher does not exhibit quasi-periodic behaviour. The flywheel and the associated “chewing” process do produce periodic effects, which can be seen as the 3 Hz variation in Fig. 6-17. In the case of crushers, however, the variation in the load to be crushed contributes significantly to the variation in load current. This variation cannot be predicted as it is dependent not only on the size of the rocks being crushed, but also on the rate at which the crusher is supplied. This is controlled by human intervention.

The unknowns and unknowables described above complicate the accurate simulation of the load. Various techniques were attempted including the Fourier and disturbance modelling techniques described in the other case studies. Unfortunately the results were less than satisfactory. Accordingly it was decided to use the measured load currents directly in a MATLAB simulation. This implies a less accurate simulation, as it was found to be impractical to model the system to the same level of detail that was possible using Simplorer. Also, certain simulated quantities (such as the line voltage at the infinite bus) had to be synchronised with the measured data. This could only be done to a finite degree of accuracy. In spite of these limitations the simulation was accurate enough to draw conclusions pertaining to the required power and energy storage capabilities of the compensator required. The simulation file is listed in Appendix D.2.

As in the previous simulation, an X/R value of unity was selected for the simulation, with the line impedance adjusted to produce about near unity flicker when no compensation is applied. The line

impedance in this simulation was taken as 52.3 mΩ. The bus voltage was measured as 300 V line to line.

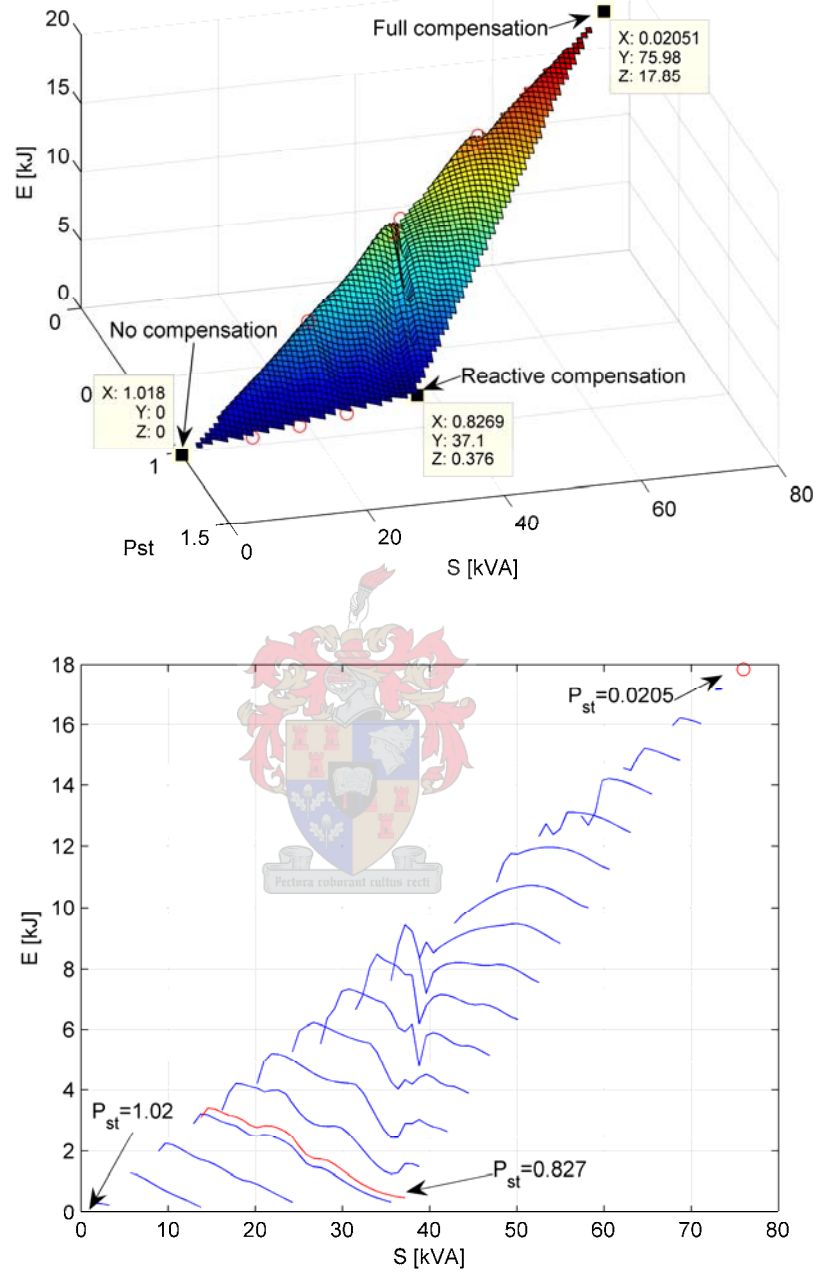


Fig. 6-18: Jaw crusher simulation results

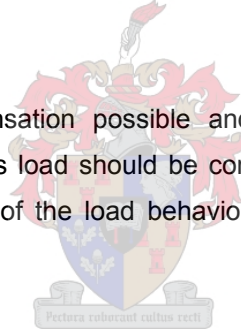
Fig. 6-18 follows the pattern set by the previous two loads. Again the compensation options form a plane in three-dimensional space. Full compensation offers compensation of 97.5% ($P_{st} = 0.025$) at a power level of 75.95 kVA and requiring an energy storage of 17.85 kJ. Reactive compensation

can only provide 17.3 % compensation ($P_{st} = 0.827$). The power level is 37.1 kVA and the energy storage needed 376 J.

Two properties of the results are interestingly different from the other two loads: Firstly, the plane seems to be 'folded'. The effect of the 'fold' can be seen in the contour plot where the contour lines are no longer straight. This effect can be explained by remembering that this simulation was done directly on raw data. In addition, the simulation of the compensator was not done as accurately as in the previous simulations. For the reasons explained in 6.2 one expects the surface in each of the three-dimensional plots to be smooth. Thus it would seem that the distortion of the plane in this figure is a result of inaccuracies in the simulation process.

The second interesting difference is the angle of the P_{st} contours. In previous cases the angle was of the order of -45° . In this case the angle is closer to -30° . This means that, for the same value of P_{st} , there will be a large variation in required compensator size depending on the compensation options selected. A reactive power only compensator would have to be very much overrated compared to a full compensator.

Because of the limited compensation possible and the high power rating necessary when compensating only reactively, this load should be compensated by supplying active and reactive power. This property is a result of the load behaviour, and not simply a result of errors in the $d - q$ algorithm.



6.5. Conclusions

A few observations can be made from the case studies presented in this chapter:

- The suggested compensator is capable of effectively mitigating flicker caused by disruptive loads.
- On lines on which the line impedance is significantly resistive feeding loads that have large variations in active power drawn, it is imperative that the compensation employed is capable of delivering active power. A solely reactive compensator will not be able to mitigate flicker successfully. This is in agreement with the theory presented in 1.1.
- The d-q filtering method provides excellent decoupling of the active and reactive load current components. Unfortunately inevitable losses in the compensator will degrade this performance. Nevertheless, compensation with only very little energy storage is possible.
- Compensation without energy storage is possible if the bus is regulated more tightly. Such regulation will impact negatively on the quality of compensation. The influence of different

bus parameters can be easily investigated using the simulations and simulation principles developed.

- Accurate simulation of the load can provide economical insights not only as to the type of compensation that must be employed, but also as to the expected cost of both the converter (from the power rating) and the energy storage. Different options can be evaluated to determine the effect on the P_{st} at the PCC studied, relative to compensation costs.

6.6. Chapter summary

In this chapter a three-phase welder, a sawmill and a rock crusher were investigated as examples of loads producing flicker problems. These loads were compensated in a simulation in either Simpler or MATLAB. To do these simulations an implementation of the theory presented in previous chapters had to be built in both environments. Accurate models were also built in Simpler representing the welder and sawmill loads. From these simulations, conclusions could be drawn as to the applicability of the compensation developed. The conclusions were summarised in 6.5.



CHAPTER 7: COMPARISON WITH AN SVC

This chapter discusses the basic principles of operation of a static var compensator (SVC). A Simpler simulation is done to compare this method of compensation with the methods developed in this thesis. Conclusions are drawn as to the applicability of this method of compensation.

7.1. Principles of operation

An SVC consists of a number of capacitors and reactors (inductors) controlled by sets of back to back thyristors. By controlling the thyristor firing angles, the total SVC impedance may be varied over a wide range. The SVC may thus be controlled to source/sink a controllable amount of reactive power to/from the grid.

By controlling reactive power flow, a wide variety of power quality problems can be addressed. SVCs are commonly used for voltage regulation, power factor correction and also flicker compensation.

SVCs are very robust, and can be built to very large power ratings. They are also much cheaper to install than most other controllable compensation devices. In spite of these desirable qualities, their slow response speed makes them inappropriate for many compensation problems. Because all control is actuated by controlling the firing angle, it is not possible to guarantee a response of faster than one half cycle of the line frequency.

A wide variety of control strategies are available for SVCs. These strategies respond to changes in the measured line voltage, line current or both. In voltage regulation applications the voltage is usually the measured quantity. Reactive power is injected in response to a sag in the measured voltage. This boosts the voltage and prevents line under voltage and voltage dropout.

Power factor correction is accomplished by measuring both voltage and current. This is necessary to calculate the reactive power flowing in the system. By injecting this current the SVC becomes the source of reactive power, and thus the power is not drawn from the bus.

As flicker is a voltage phenomenon, it seems obvious that the measured voltage amplitude should be the main controller input. As was found in 1.1, this is not necessarily the best option. As the

voltage error signal is likely to be much smaller than the current error, measuring the current is much less prone to noise and compensation error.

7.2. Simulation in Simplorer

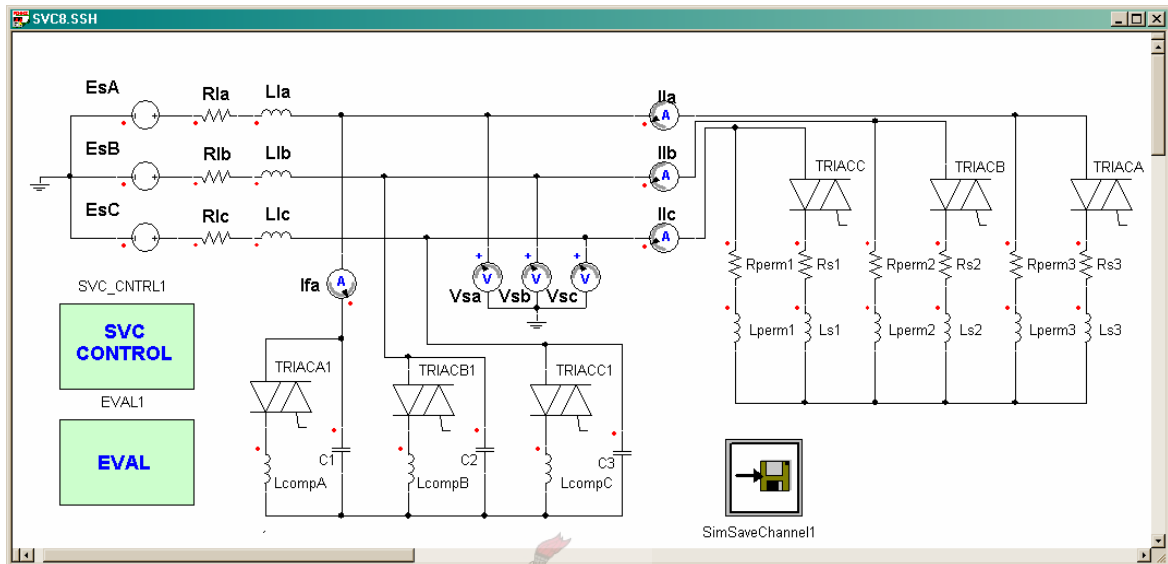


Fig. 7-1: Simplorer simulation of an SVC

Fig. 7-1 shows the load and SVC as modelled in Simplorer. Comparing this figure with Fig. 6-7, the reader will recognise the load to be the simulated three-phase arc welder load investigated in 6.2. The SVC is comprised of three triacs (TRIACA1, B1 and C1 in the figure) and the associated reactance and capacitance.

Design of the SVC was accomplished by utilising the procedures set out in [47]. From the investigation into properties of the welder load the variation in reactive power was known to be about 400 kVAR. The compensator should be able to deliver at least this amount of reactive power in order to compensate the variations on reactive current drawn by the load. Thus the value of the SVC capacitors could be calculated:

$$X_C = \frac{(V_{bus})^2}{Q_{SVC}} \quad \text{Eq. 7-1}$$

With a bus voltage of 11 kV line to line, this easily gives a capacitive reactance of 100 Ω , or 31.8 μF in a 50 Hz system. When the SVC firing angle is set to the maximum (180°) the reactors are effectively never switched into operation. This gives the maximum reactive power injection of 400 kVAR.

Choosing the value of the SVC reactance determines the SVC operating range. One simple choice is to keep the range of injectable power values symmetrical. This means

$$\max(Q_{SVC}) = -\min(Q_{SVC}) \quad \text{Eq. 7-2}$$

and is achieved when

$$X_L = \frac{X_C}{2}. \quad \text{Eq. 7-3}$$

This gives the value of the SVC inductive reactance as 50 Ω , or 160 mH.

In [47], the relationship between firing angle and SVC susceptance is given by:

$$B_{SVC} = \frac{X_C [2(\pi - \alpha) + \sin 2\alpha] - \pi X_L}{\pi X_C X_L} \quad \text{Eq. 7-4}$$

where α is the SVC firing angle. Eq. 7-4 is derived in Appendix A.1. This relationship allows a plot of the firing angle versus the injected reactive power. This was done in Fig. 7-2, showing the effective SVC operating range. At a firing angle of 115°, the capacitive and inductive sides of the SVC balance and the injected current becomes zero.

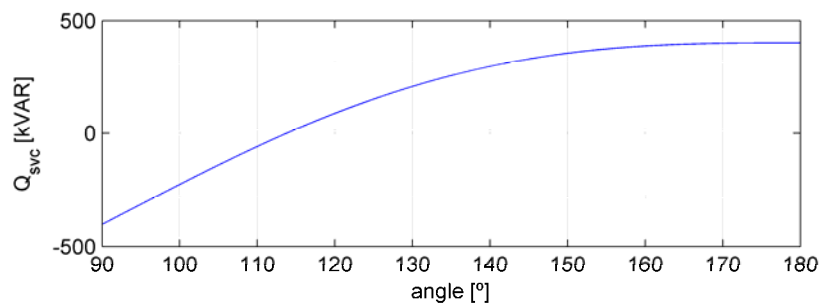


Fig. 7-2: SVC compensation range

To find the compensation reference, the measured load current was decomposed into its d and q components as described in 4.3. By virtue of the same arguments presented there, the q component of the current can be said to represent the reactive power flow to the load. Dividing this current by the measured bus voltage gave the required SVC susceptance.

The calculated susceptance then had to be translated to a thyristor firing angle using Eq. 7-4. This was not trivial, as solving Eq. 7-4 for α is difficult. To avoid this problem a numerical method was used. Taking α as a vector ranging from 90° to 180° , a range of α - B data pairs could be computed in MATLAB. Calculating susceptance instead of reactance avoided a discontinuity at the zero crossing seen in Fig. 7-2 which would have presented numerical difficulties. A 40th order polynomial was fitted to the data pairs, swapping the α and B values to produce an analytical expression relating B to α . Evaluating this polynomial over the range of expected susceptance values produced a reference table which was imported into the simulation. The SVC controller calculated firing angles by looking up the required susceptance in this table, linearly interpolating the data as necessary. Once the firing angle was known, the calculation of the switching delays was easy.

7.3. Simulation results

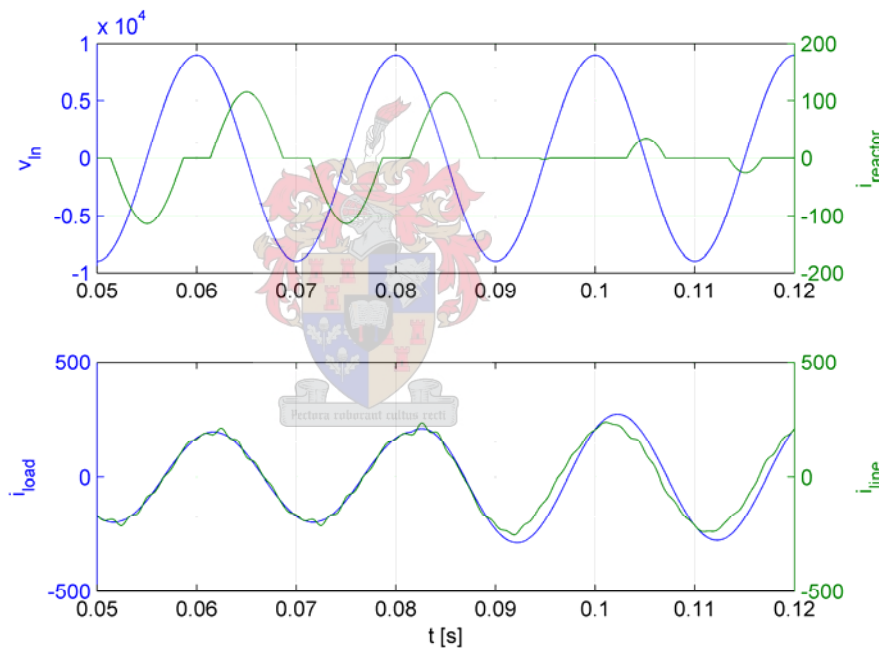


Fig. 7-3: SVC currents

Fig. 7-3 (top) shows the current through the SVC reactors on the same plot as the line voltage. Note that the firing angle is small at first. When the welder is not welding the reactors are switched into operation for most of each cycle. They nearly fully cancel the effect of the SVC capacitor. When the welder welds, the expected load step is seen in the welder load current Fig. 7-3 (bottom). The SVC compensates by increasing the reactor firing angle, thereby decreasing the effect of the reactor and increasing the total reactive power injected. As expected the injection of an appropriate amount of reactive power provides compensation, clearly visible in the bottom plot in the figure.

Around 0.1 s the firing angle is nearly zero. This is because the SVC is actually too late in compensating the load step. At approximately 0.085 s, when the load step occurs, the reactor triacs have already been switched. The SVC is unable to alter the compensation until the next half cycle when the firing angle may be set to a new value. The SVC compensates for this by overcompensating the next half cycle. The reactive power injected is maximised by maximising the firing angle. This allows the SVC to follow the reference in an average sense.

Because an SVC is not able to compensate immediately, it is ineffective in compensating loads with rapid current variations. The time lag inherent in its operation caused instability when the SVC was applied to the sawmill and crusher loads. While a more complex firing algorithm may improve the situation, the harmonic response of an SVC will always be inferior to that of a switch mode solution. Also, if the compensation objectives cannot be met by injecting reactive power only, an SVC is unable to inject active power from an energy storage device as was done with the switch mode solution.

In spite of these limitations the SVC was very effective in compensating the simulated welder. Flicker was reduced to $P_{st} = 0.613$, only slightly worse than the $P_{st} = 0.0591$ possible with the switch mode solution in 6.2. The compensation possible is still much less effective than the switch mode solution injecting active and reactive power ($P_{st} = 0.0160$).

An SVC will always represent a source of harmonic currents, as the current drawn by the switching SVC inductors are far from sinusoidal. All odd harmonics will be present. The amplitude of the harmonics will be given by Eq. 7-5. This equation is derived in Appendix A.3.

$$I_n = \frac{4V}{\pi X_L} \left[\frac{\sin(n+1)\alpha}{2(n+1)} + \frac{\sin(n-1)\alpha}{2(n-1)} - \cos\alpha \frac{\sin n\alpha}{n} \right] \text{ with } n = 3,5,7\dots \quad \text{Eq. 7-5}$$

Third order harmonics are often filtered by using a wye-delta transformer configuration. Other harmonics may be reduced by passive filters. Harmonic frequencies fall outside the human visible spectrum, and therefore present no problem in flicker compensation.

Generally, an SVC is to be preferred above a switch mode solution where possible, as the cost associated with an SVC is much lower than that of a converter-based compensator.

7.4. X/R sweep

As was noted in 1.2.3, the ability of a reactive power compensator to compensate a flicker load is influenced by the line impedance. Specifically, for loads drawing a substantial amount of active current, the more resistive the line is the less effective reactive compensation will be.

The three-phase welder does draw a substantial amount of active power (see Table 6-2). This provided an opportunity for verifying the statements made in 1.2.3 and proven in Appendix A.1. The result of this simulation experiment can be seen in Fig. 7-4 and Table 7-1.

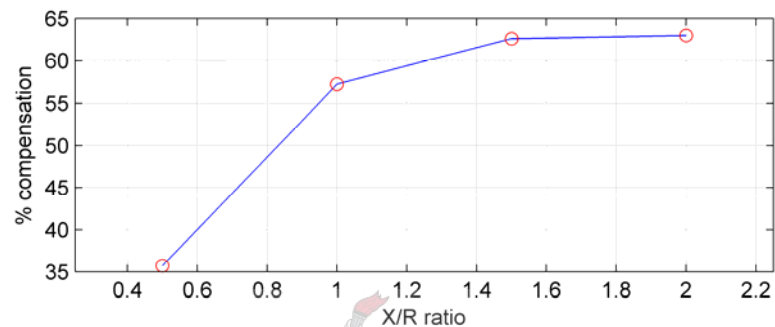


Fig. 7-4: Compensation sweeping the X/R ratio

X/R	P_{st} (Uncompensated)	P_{st} (Compensated)	% Compensation
0.5	1.23	0.792	35.7
1.0	1.43	0.613	57.2
1.5	1.52	0.567	62.6
2.0	1.52	0.564	63.0

Table 7-1: P_{st} values from X/R sweep

In the simulation the magnitude of the line impedance was kept constant at 0.995, the value used in all the three-phase welder simulations in this thesis. By changing the angle of the line impedance, different ratios of line susceptance to resistance (X/R) could be obtained. Simulating each of these scenarios and calculating the flicker before and after compensation by SVC, resulted in the figure shown.

In Table 7-1 the uncompensated flicker in the system can be seen to increase with X/R ratio. This is to be expected: As the load draws both active and reactive power, both the active and reactive current components flowing through the line impedance will cause flicker. As is noted in Appendix A.1, active power flowing through a resistive line impedance will cause a larger voltage drop than active power flowing through a reactive line impedance. The converse is true for reactive current.

Thus a load drawing substantial active and reactive current will cause maximum voltage drop over the line impedance at a specific X/R ratio. This will also be the X/R ratio of maximum flicker.

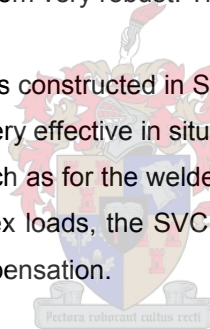
In spite of the uncompensated flicker increasing with X/R ratio, improvement is seen in the compensated flicker and percentage compensation achieved. This supports the statements of 1.2.3 and Appendix A.1 by showing a reactive power compensator to be much better at compensating a system fed by a reactive line than by a resistive line.

The plot in Fig. 7-4 “flattens out” for large X/R values. This implies that better than 63 % compensation is not possible using reactive power only. Also, the plot “drops off” sharply for low X/R values. This implies that a reactive power only compensator will be very ineffective in compensating such loads on such lines.

7.5. Chapter summary

In this chapter the operating principles of an SVC were briefly covered. It was noted that SVCs are fairly simple to operate, making them very robust. They can be constructed to high power ratings.

A simulation model of an SVC was constructed in Simplorer, and was used to compensate a flicker load. The SVC was found to be very effective in situations where the load changes slowly, or where the changes are well defined (Such as for the welder). The effectiveness decreases rapidly as load complexity increases. For complex loads, the SVC response time of up to half a cycle is not fast enough to provide adequate compensation.



An SVC can never supply active power. In situations where active power is needed to reach the compensation objectives, an SVC is not an option. The SVC also represents a harmonic source. Extra filtering is needed to rid the system of these harmonics. Harmonics do not impact on flicker, as they fall outside the human visible spectrum.

SVCs are much cheaper to install than converter-based compensators. This suggests that it will be more economical to employ an SVC in cases where an SVC can provide adequate compensation. In many other cases an SVC will not be able to compensate effectively. In these cases converter-based compensators, such as the compensator developed in this thesis, is the only viable option.

CHAPTER 8: HARDWARE SIMULATION AND RESULTS

To verify the correct operation of the developed compensator, the three-phase welder from 6.2 was simulated in hardware and experimentally compensated. This chapter highlights the practical considerations involved in such a compensation effort, and presents the results obtained.

8.1. Motivation for doing a hardware simulation

Given that the theory developed in this thesis was rigorously researched and then proved to be correct by simulating a variety of loads, the necessity of implementing the theory in hardware may be questioned. As anyone involved in practical power electronics will assert however, simulating a converter and implementing it in a realistic system are two very different things. Simulation, while very suitable in the development of concepts, does not model nearly enough of the phenomena in a system to give conclusive results as to its operation. Very often a carefully simulated control system will prove to be totally unstable in a practical environment. Only by physically implementing a system can its operation be fully verified.

8.2. The DPQC as compensator

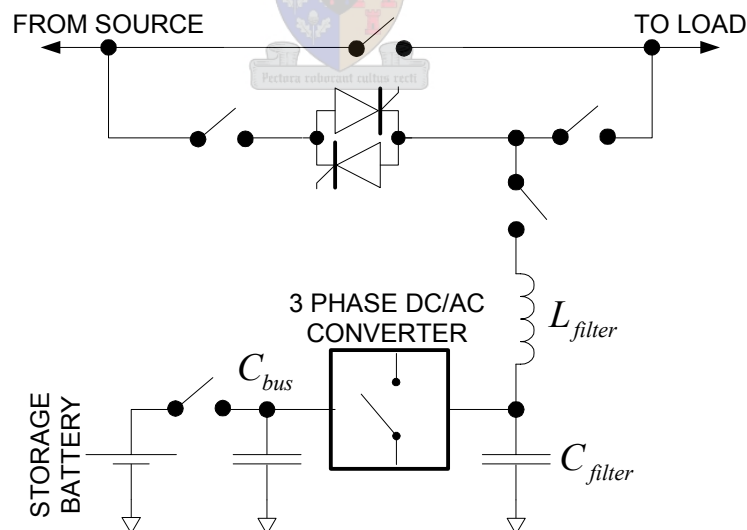


Fig. 8-1: Block diagram of the DPQC

Rather than building a new three-phase inverter from scratch, a previously developed compensating device was used. This device, the DPQC (Dynamic Power Quality Compensator)

contains a three-phase inverter, measurement and controlling hardware and the switchgear necessary to safely switch the compensator in and out of operation. By changing the controller software, the DPQC could be turned into a flicker compensator. Re-using this device was much more economical than building a new device from scratch.

The DPQC is shown in block diagram form in Fig. 8-1. The most important DPQC parameters are summarised in Table 8-1. Where applicable these parameters can be seen to correspond to the parameters used in the simulations of Chapter 6. The measurement equipment used in this section and throughout this chapter is listed in Table 8-2.

Parameter/Component	Value	Description
DC bus voltage	735 V	
DC bus capacitance	20×4.7 mF	SEMIKRON 4M7-45
Batteries (Energy storage)	58×12 V _{nominal}	Deltec High Cycle 1250
IGBT modules	1200 V, 270 A	SEMIKRON SKIM5 400GD128D
Filter capacitance (per phase)	4×50 μ F, 440 V _{AC}	AFCAP
Filter inductance (per phase)	400 μ H	ATM SA
Main controller DSP		Texas Instruments TMS320C31PQL50
Main controller FPGA		ALTERA FLEX EPF81500AQC240-3
Sampling frequency f_s	20 kHz	
Switching frequency f_{sw}	5 kHz	

Table 8-1: DPQC parameter values

Equipment	Type
Oscilloscope	Tektronix TDS 3014B
Voltage probe	Tektronix TEK P5100 100 \times
DC current probe	Tektronix P6303 with AM503B current probe amplifier
AC current probe	Tektronix TCP202 10 \times current probe used with a CT4 clip on current transformer

Table 8-2: Measurement equipment

In previous projects the DPQC had been used as an uninterruptible power supply (UPS), active filter, dip-protector etc. Most of these applications charged the DPQC energy storage during fairly long periods of "normal" load and line conditions and then used the stored energy during

comparatively short “fault” conditions. When energy storage is used in flicker compensation, the mitigation of the flicker implies a continuous exchange of energy between the line and the energy storage. This difference in mode of operation necessitated the revision of much of the DPQC protection software.

The DPQC was connected to a lead acid battery bank in the experiments in this thesis. Such batteries are in fact not a very good choice of energy storage, as the fast charge/discharge cycles associated with mitigating flicker wear out the batteries prematurely. Supercapacitors are much more robust in this regard, and should be considered a better option in a commercial design [48]. In spite of their long-term deficiency the batteries available were quite adequate for the testing of the algorithms developed in this thesis.

Considering the DPQC bus regulation, it should be apparent from the preceding chapters that no net flow of energy is expected into or out of the bus (and energy storage) under compensation with or without active power. Nevertheless, it cannot be accepted that the DPQC bus will automatically regulate itself at a constant voltage. Controller error and the need to charge the bus to some value in the first place necessitate the usage of a bus regulator.

In order to regulate the bus current needs to be drawn through the DPQC converter, interfering with the DPQC compensation action. Although this interference cannot be avoided, the effects can be minimised by low-pass filtering the regulator reference. By slowing the regulation its effects can be made so gradual as to be invisible. As the regulator only compensates for controller error, this slow response time should not cause the bus to vary unacceptably.

As was noted in 6.5, the bus regulator implies a design trade-off between better regulation (implying smaller energy storage) and less interference from the regulator (implying better compensation). For the implementation studied here a cutoff frequency of 0.317 Hz was chosen by iteration. The results will show this to have been an adequate choice.

8.3. Selection and modelling of the three-phase load

A model of the three-phase welder of 6.2 was used as load. The decision to use this load as basis for the experiment was taken for two reasons:

- Comparing the three-phase welder to the other loads studied in the previous chapter, the welder is obviously easier to simulate because of the on/off nature of the welding process. In fact, it is possible to simulate the load practically exactly as was done in Simplorer (shown in Fig. 6-4.).

- The compensation algorithm does not exploit any of the load characteristics peculiar to the welder. Therefore, it is not unreasonable to conclude that a compensator capable of successfully compensating the large load steps in the welder will also be able to compensate the slower load variations in other loads.

To accurately investigate the success of the compensator, both when compensating fully and when compensating reactively only, the poor power factor of the welder had to be attained using discrete inductors. As only limited inductance was available in the laboratory, the load was not simulated exactly as was done in the Simplorer simulation. Instead of parallel load branches representing the welder welding and non-welding state, series impedance was switched in and out of the load circuit. While having only minimal effect on the load characteristics, this greatly reduced the number of inductors needed.

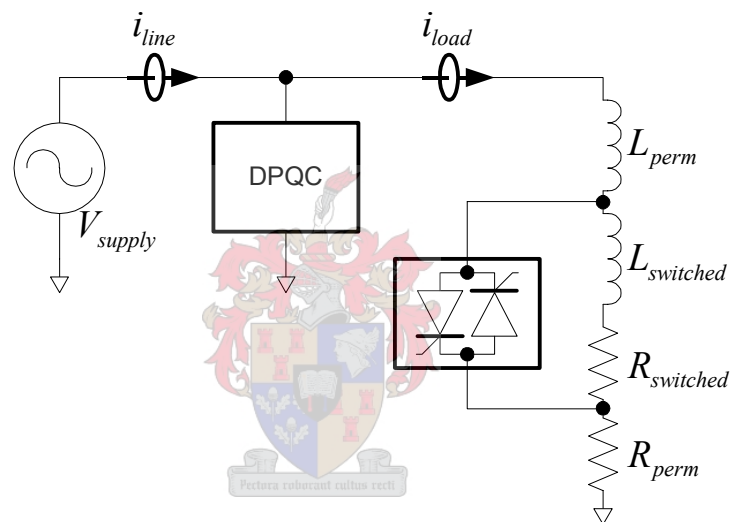


Fig. 8-2: Hardware simulation of the three-phase welder

The simulation parameters are given in Table 8-3.

Quantity	Value
V_{supply}	230 V _{RMS}
L_{perm}	3.68 mH
$L_{switched}$	3.82 mH
R_{perm}	2.09 Ω
$R_{switched}$	1.54 Ω

Table 8-3: Load simulation parameter values

Back-to-back SEMIKRON SKKT 253/16E thyristors were used to switch the load impedance. These thyristors were controlled using an ALTERA EPM7064 electronically programmable logic device (EPLD). The EPLD implemented a counter, by which the thyristors could be switched at a set duty cycle and frequency. The thyristors were switched via Texas Instruments SN75471 drivers driving pulse transformers. Schematics and VHDL code for the controller may be found in Appendix E.

The switching frequency was set to 1.00 Hz, with a duty cycle of 0.123. This is the same frequency and duty cycle measured at the three-phase welder load, and used in 6.2. The measured switching frequency and duty cycle were 0.990 Hz and 0.128 respectively, matching very closely with the set parameters.



Fig. 8-3: Photo of the hardware simulation setup

Fig. 8-3 shows the hardware setup in the laboratory. The orange enclosure to the left housed the load switching thyristors and their controller. The set of load inductors can be seen behind the enclosure. The load resistors were housed in an outdoor compartment.

The blue device to the right of the picture is the DPQC. The enclosure shown houses the three-phase converter and its controller. The DPQC batteries were housed in a separate battery room.

Modelling the welder load as explained in this section produced currents of $93 \angle -29^\circ A_{\text{rms}}$ during normal load conditions, and $114 \angle -38^\circ A_{\text{rms}}$ during simulation of the welding process. This corresponds to the three-phase welder by a scaling factor of roughly 0.5 in current.

8.4. Compensation results

Evaluation of the compensation methods developed started with the compensation being implemented in its most basic form. The time delays inherent in the system (as described in 5.7) and the effects of dead time compensation (as described in 5.9) were originally not compensated. This yielded the results in Fig. 8-4.

By tuning the compensator gain, very good compensation of flicker could be obtained. By iteration the optimum gain setting was reached at $D = Q = 1.3$. Using this gain the variation in load current after compensation was minimal. The flicker produced should be decreased significantly. Scaling back the compensated current to levels expected at the welder, and using the line impedance calculated there, the expected flicker could be calculated as 0.073. This implies compensation of 94 %. Comparing this figure with the 98 % compensation from the simulation in 6.2, shows that the practical compensator compensates the flicker very nearly as well as in theory.

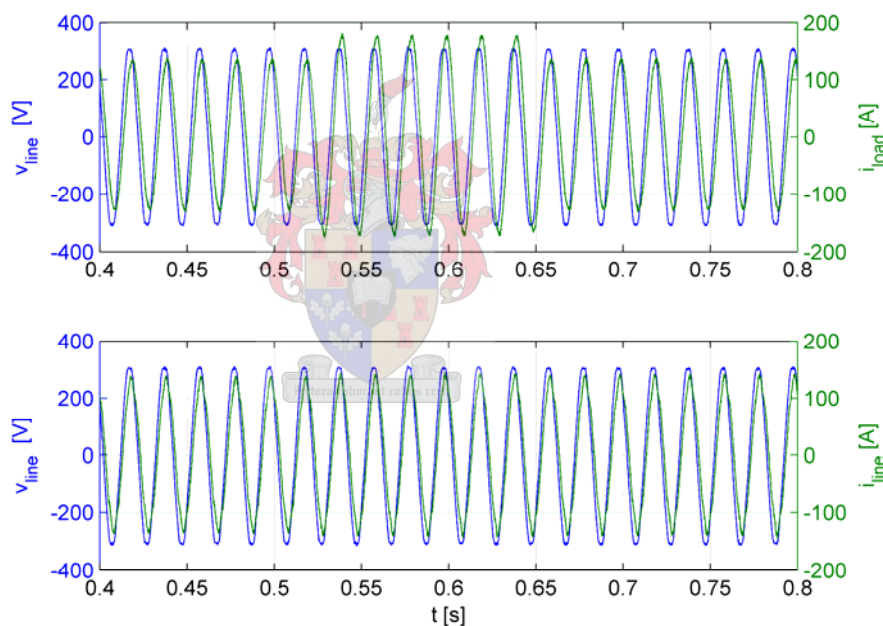


Fig. 8-4: Measured compensation results – Basic compensation

Evaluation of the quality of compensation also had to be done in the frequency domain. Looking at a Fourier decomposition of one cycle of the load current during compensation (Fig. 8-5), the fundamental frequency is seen to be the largest contributing component by far, meaning that the compensated current should be close to completely sinusoidal. Small components at the fifth and seventh harmonics are visible, explaining the slight distortion visible in the waveform. These high frequency components do not contribute to flicker as they are imperceptible to the human vision system.

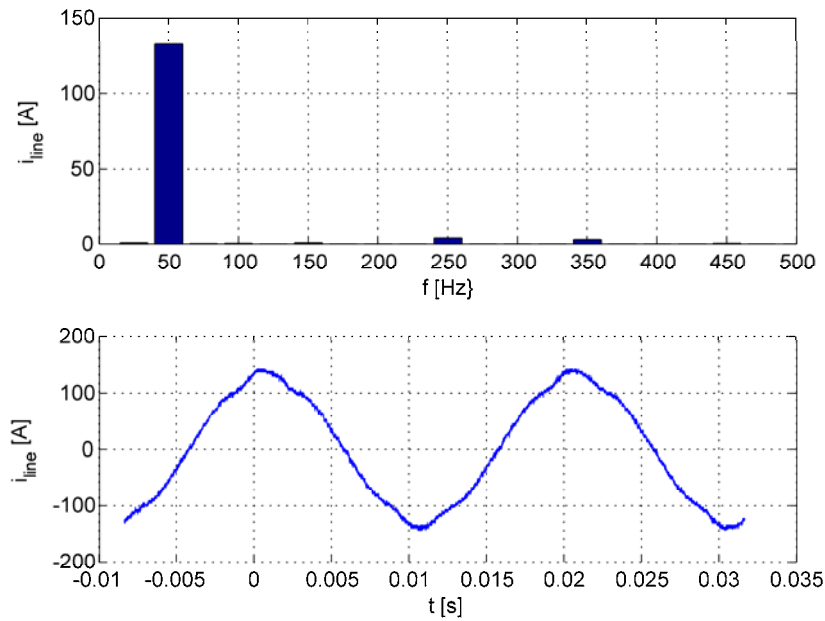
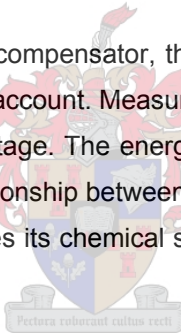


Fig. 8-5: Basic compensation - Spectrum and zoom

To evaluate the energy flow in the compensator, the energy in both the battery bank and the DC bus capacitors had to be taken into account. Measuring the energy in the capacitors is easy, as this is directly related to the DC bus voltage. The energy stored in the battery bank is more difficult to calculate, as there is no simple relationship between the battery voltage and the stored energy. The electric history of a battery influences its chemical state, which in turn influence the battery energy capacity.



Fortunately, knowledge of the total energy stored in the compensator is not required. To determine the size of the energy storage needed, only the energy needed to compensate each load step has to be calculated. This could be done by calculating the energy variation in the capacitor bank and adding it to the energy variation in the battery bank.

The energy in the bus capacitors can be calculated by taking

$$E_{cap}(t) = \frac{1}{2} C [v_{cap}(t)]^2. \quad \text{Eq. 8-1}$$

The energy variation in the bus capacitors is then calculated by

$$\tilde{E}_{cap}(t) = E_{cap}(t) - \int_T E_{cap}(\tau) d\tau. \quad \text{Eq. 8-2}$$

The energy from the battery bank could be calculated by subtracting the average battery current from the total battery current, multiplying the resulting variation in current with the battery voltage to give variation in battery power, and integrating this power to give the variation in battery energy:

$$\tilde{E}_{bat}(t) = \int_0^t \left[v_{bat} \left(i_{bat} - \frac{1}{T} \int_T i_{bat} d\tau \right) \right] d\tau . \quad \text{Eq. 8-3}$$

The average battery current charges the compensator energy storage. This part of the current becomes negligible as the battery bank reaches its nominal value. It does not contribute to the exchange of energy as part of the compensation process.

The total variation in compensator energy could be calculated by summing the variation in capacitor energy and the variation in battery energy. The result is presented in Fig. 8-6.

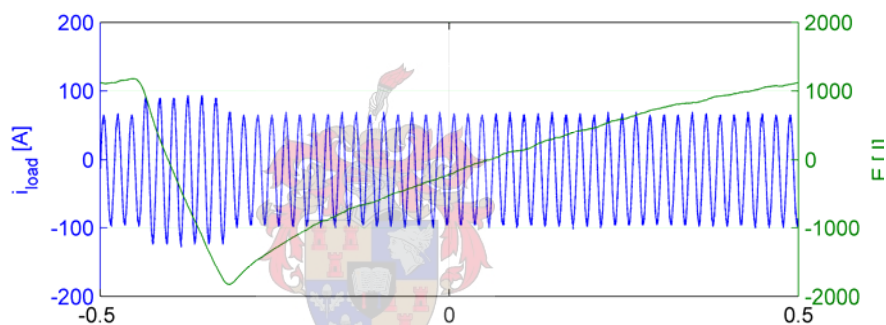


Fig. 8-6: Measured compensation results – Energy

In the figure, the drop in the compensator energy can be seen very clearly at the start of the load step. The compensator uses this energy to inject active power into the bus. After the load step has passed, the compensator gradually restores the energy used. This is done slowly, as to minimise the current drawn.

The total energy variation, and therefore the size of the required energy storage, is 3.01 kJ. Scaling this value back to the real welder load gives an energy requirement of 99.8 kJ. Comparing this value with the 63.4 kJ from the simulation shows that the compensator required much larger energy storage than indicated by the simulation. This is expected, because of the unavoidable losses in the compensator and in the storage elements themselves.

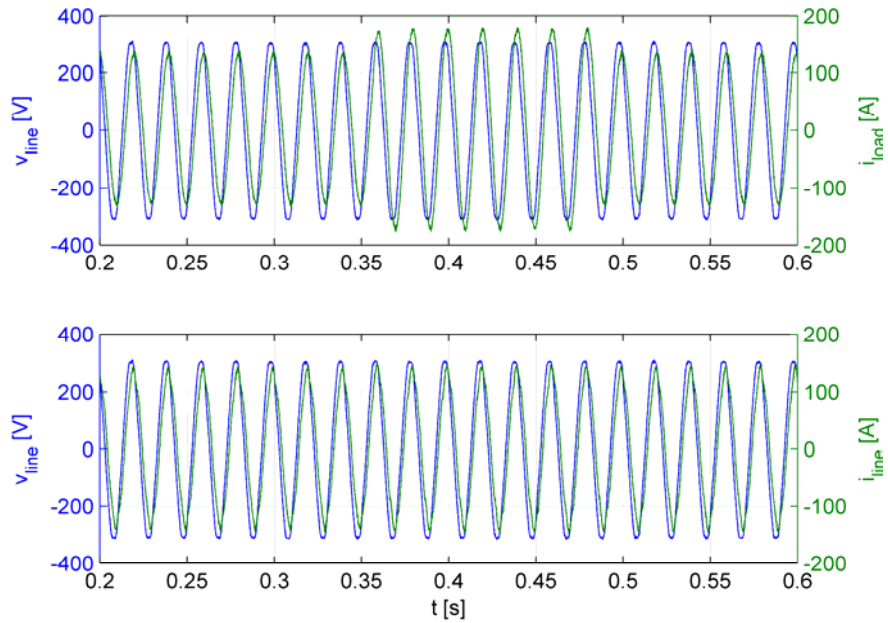


Fig. 8-7: Measured compensation results – Time compensated

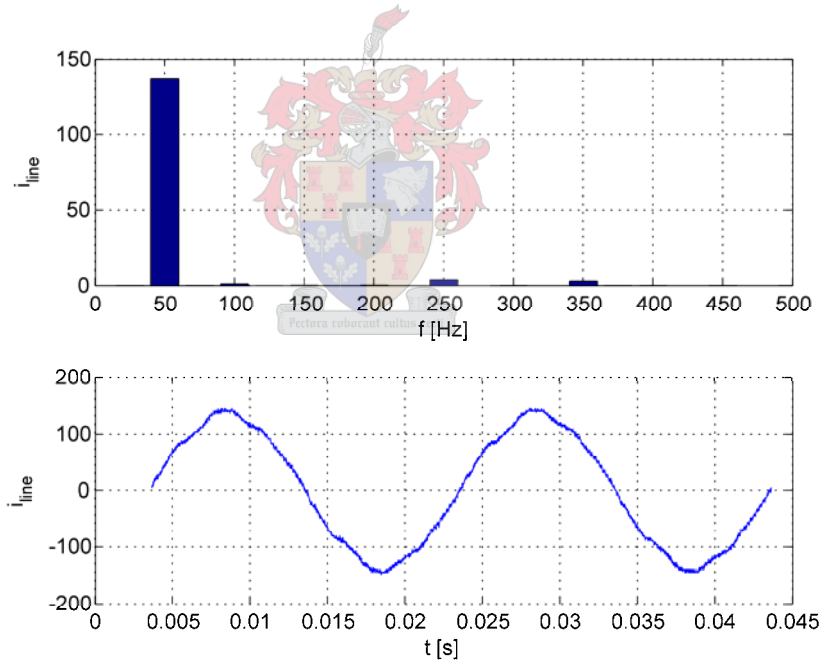


Fig. 8-8: Time compensated – Spectrum and zoom

In an attempt to correct the slight harmonic contamination the effects of the discrete, time delayed controller were compensated (as described in 5.7). Excellent compensation of the flicker problem could again be achieved, the compensation being nearly identical to that measured in the previous case.

Looking at the harmonic spectrum (Fig. 8-8), little improvement can be seen over the uncompensated case. Although the fifth and seventh harmonic components are decreased slightly (~5%) a second harmonic component is added, making this implementation slightly inferior to the first. It is recommended that the compensation be implemented as in the first case. If instability problems should arise, the compensation of controller effects could be attempted.

Applying the dead time compensation developed in 5.9, led to the results shown in Fig. 8-9. The effectiveness of the dead time compensation can be seen from the fact that the desired compensation could be achieved at a much smaller loop gain. Whereas a gain of $D = Q = 1.3$ had been used in the previous cases, a gain of only 1.1 was needed after dead time compensation. This shows dead time to be the largest contributor to error in the compensator current loop calculations.

Comparing the results with the previous two cases show them to again appear almost similar. In the spectrum, the second harmonic component disappeared, to be replaced by a slight third harmonic. A small increase in the fifth harmonic component can also be seen.

As was the case for time delay compensation, the overall effect of dead time compensation is so small as to render it unnecessary. Adequate compensation of the controller inaccuracies can be obtained by simple gain tuning. If the increased system gain was to lead to instabilities in a specific implementation, the developed dead time compensation may be employed, allowing the reduction of the loop gain. This should improve the system stability.

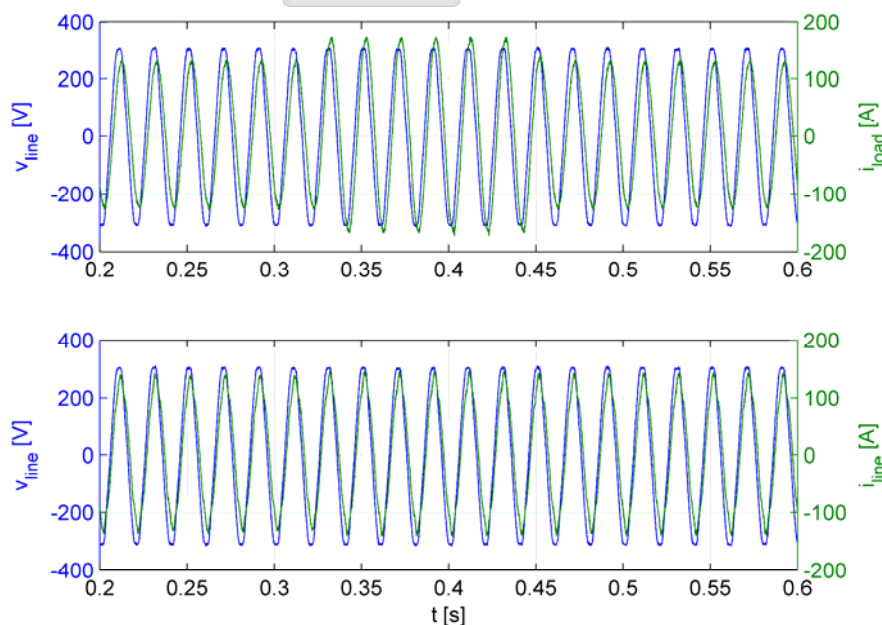


Fig. 8-9: Measured compensation results – Time and dead time compensated

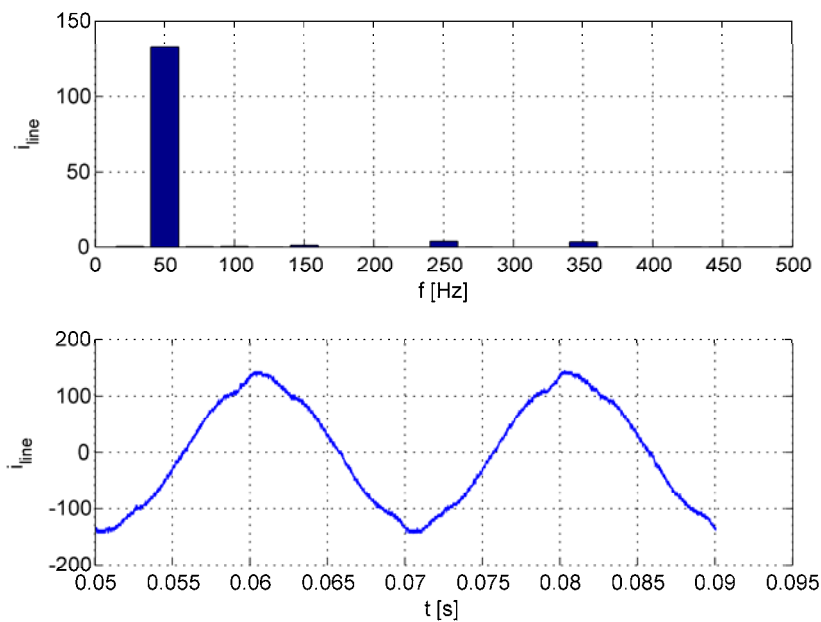


Fig. 8-10: Time and dead time compensated – Spectrum and zoom

The last variation in the compensation applied is compensation with minimal energy storage. Given the results of the previous experiments, reactive compensation was applied without time or dead time compensation being included in the controller algorithm. Results are presented in Fig. 8-11.

From Fig. 8-11 it is clear that reactive compensation alone is not enough to filter the whole load step. A clear variation in line current is visible at about 0.55 s, when the simulated welder commenced a welding cycle. The filter does smooth the step to a certain extent.

The filter is able to filter the phase of the line current. The large step in phase visible in the load current is not visible in the compensated current.

The compensation effort was measured to reduce the line flicker from an uncompensated value of $P_{st} = 1.1$ to $P_{st} = 0.46$. This implies 58 % compensation. These results are in accordance with the results of the simulation in 6.2 and the theory developed in 1.1.

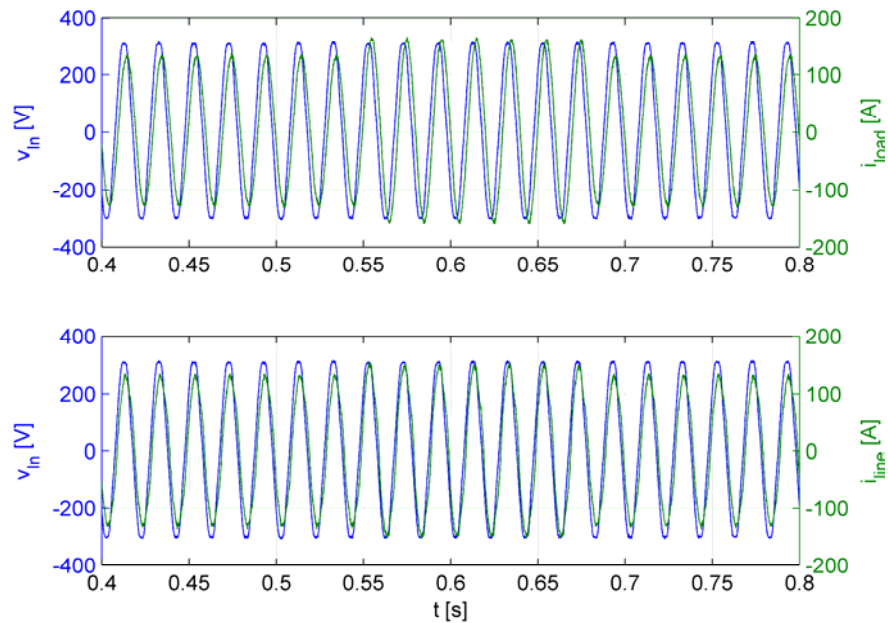


Fig. 8-11: Measured compensation results - Reactive compensation

Evaluating the energy used in the compensation effort produced Fig. 8-12. The same method was used as was used earlier in this section. Scaling back the energy required to the levels expected at the welder, the compensator is seen to require an energy storage capability of 15.2 kJ. As was the case when evaluating active compensation, this figure differs substantially from the 0.22 kJ required in the simulation. Again, this can be explained by taking the losses in the compensator and storage elements into consideration. These results highlights the importance of accurately modelling losses when a simulation is done for design purposes.

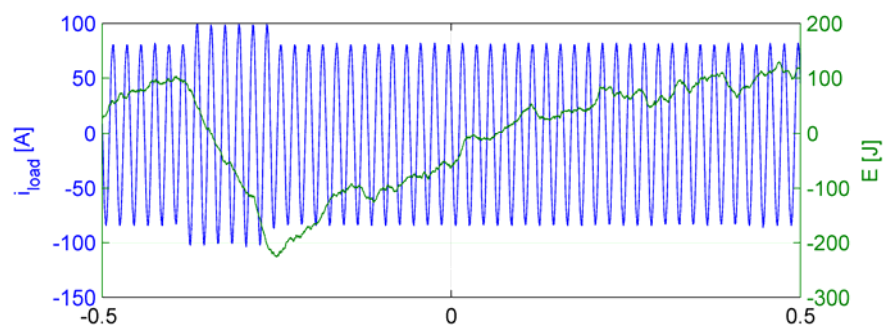


Fig. 8-12: Measured compensation results – Energy in reactive compensation

8.5. Chapter summary

In this chapter, the modelling of the three-phase welder from 6.2 was motivated and discussed in detail. The compensation algorithms developed in the rest of the thesis were implemented in a practical three-phase converter, and used to compensate the flicker load model. The results obtained were found to agree excellently with the results from the simulations undertaken in 6.2. In this way the theory presented and the accuracy of the simulations were proven.

Compensating the load with the most basic implementation of the synchronous reference frame algorithm produced excellent compensation results. Scaling the d and q channel reference currents using a fixed gain overcame errors in the compensation calculation and current loop.

Implementing the prediction algorithms of 5.7, designed to overcome time delays inherent in the compensator, did not improve the compensation significantly. Implementing these algorithms does not seem justified in this case. If stability problems should arise in compensating some other load, these algorithms could be employed to facilitate stability.

Implementing dead time compensation as developed in 5.9 did improve the compensator response. The extra loop gain inserted into the compensation calculation could be reduced significantly. For this load, the extra compensation effort required to implement dead time compensation does not seem worthwhile as the same results can be achieved by simply increasing the loop gain. If the compensator should be unstable compensating some other load, the loop gain may be decreased by implementing dead time. This would aid system stability.

Lastly, compensation with minimum energy storage was attempted. The compensator performed as was expected from the simulations in 6.2.

CHAPTER 9: CONCLUSION

This chapter summarises the conclusions drawn from the work presented in this thesis. No new material will be presented.

9.1. Thesis conclusion

The term 'flicker' describes voltage variations in the sub line-frequency range. These variations can be of a periodic, stochastic or chaotic nature. The term is taken from the visible flicker in the output of electric lighting systems connected to such lines. Flicker is caused by loads drawing non-sinusoidal currents. These error currents cause a non-sinusoidal voltage drop over the line impedance.

One way to solve flicker problems is to connect the problem load to a separate line. This is an effective, but very expensive problem. This leads to the study of methods of compensating flicker effects.

Compensation of flicker is best achieved by utilising a parallel connected compensation device. This device should provide the non-sinusoidal component of the current partially or in full to achieve partial or total compensation. This scheme prevents the unwanted current components from flowing through the line impedance, making the voltage drop closer to sinusoidal.

Passive compensation alone is usually inadequate. Thus flicker compensation is normally implemented by using active devices such as power electronic converters or SVCs. A converter-based compensator has a wide bandwidth, while SVCs have an inherently slow response. The lag in response from the SVC renders it unsuitable for compensating rapidly varying loads.

Converter-based compensators can compensate both active and reactive current components. For active compensation it is necessary to connect the compensator to a means of energy storage. An SVC can provide only reactive compensation. Reactive compensation is likely to be sufficient if both the load and line X/R ratios are high.

SVCs are much cheaper than converters, making them the preferred solution. Even in cases where active power is necessary to achieve the compensation objectives, an SVC might be combined with a converter-based compensator with energy storage if the reduction in the ratings of the converter makes this solution more economical.

Excellent control of the compensator can be achieved by using the synchronous reference frame filtering technique. This technique provides for the decoupling of active and reactive current terms, making different compensation implementations possible. The references obtained may be actuated in an SVC via firing angle control, or in a converter by using deadbeat control and space vector modulation.

Various techniques are available to compensate time delays in the controller, and to compensate the errors introduced by dead time in an inverter. In practise these methods have little impact on the flicker perceived. They may improve system stability, and should be considered in an unstable system.

In solutions using energy storage, the energy stored needs to be controlled in such a way that it will not interfere with the compensating operation of the compensator. This can be done by making the regulator slow enough. A too slow regulator will cause the required energy storage to be unduly large.

These conclusions are backed by a hardware simulation, in which the theory from which they derive was proven to be correct.

9.2. Future work

In the simulation of some of the loads investigated in this thesis it was necessary to resort to intuitive techniques in the time domain. Although these methods proved effective, a deeper understanding of flicker loads may be possible utilising so called *wavelets*. Such models could lead to better prediction of load behaviour, which could aid in the calculation of the compensator power and energy ratings. It could also impact on the compensation algorithms used [49].

A more complex control algorithm should be considered. It may be possible to reduce the flicker perceived substantially by compensating only selected components of the flicker-causing current. Thus the compensator power and energy ratings of the compensator may be reduced, making compensation much more economical. It may be possible to use adaptive control and fuzzy logic to improve the controller [50].

Flicker compensation is often needed at voltages much higher than those possible using currently available power electronic semi-conductors. In this thesis, the problem was solved by means of an injection transformer. Multilevel converters offer an alternative solution, which may be superior to the usage of a large, lossy, line frequency transformer.

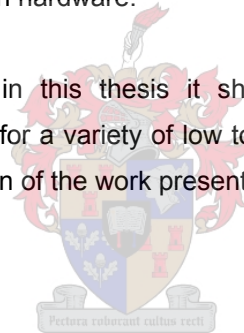
Interleaved, multilevel compensators make compensation possible at very high power levels. Such compensation schemes should be investigated, specifically to compensate large power flicker sources, such as arc furnaces.

Much work is still to be done in the modelling of arc furnaces. There is no reliable model of such furnaces which could be used in simulations such as presented in this thesis. On many lines, the only current solution for arc furnace flicker is the (very expensive) installation of a separate line from a strong grid.

9.3. Thesis contribution

Several potential industrial sources of flicker were investigated. A customizable data logger and a software flicker meter were developed to aid in this investigation. The theory behind the suggested compensation was covered in depth, including the compensation of possible controller non-idealities. It was shown how these loads may be simulated, and how design parameters such as compensator rating and energy storage size may be determined from these simulations. The compensation was implemented in hardware.

Using the methods developed in this thesis it should be possible to design effective and economical flicker compensation for a variety of low to medium power problem loads. Other loads may be compensated by extension of the work presented.



REFERENCES

- [01] O. Ozgun and A. Abur, "Flicker study using a novel arc furnace model", *IEEE Trans. on Power Delivery*, vol. 17, no. 4, Oct. 2002.
- [02] IEC 61000-4-15, "Electromagnetic compatibility (EMC) – Part 4: Testing and measurement techniques, Section 15: Flicker meter - Functional and design specifications", IEC, 2003.
- [03] Z. Zhang, N. R. Fahmi and W. T. Norris, "Flicker analysis and methods for electric arc furnace flicker mitigation", *IEEE Porto Power Tech Conference*, pp. 21 – 30, Sep. 2001.
- [04] F. Z. Peng, "Application issues of active power filters", *IEEE Industry Applications Magazine*, Sept./Oct. 1998.
- [05] L. M. Tolbert and T.G. Habetler, "Comparison of time-based non-active power definitions for active filtering", *Proc. VII IEEE International Power Electronics Congress*, pp. 73-79, Oct. 2000.
- [06] IEEE 519-1992, "Recommended Practice for Power Distribution in Industrial Plants", IEEE, 1993.
- [07] IEEE 141-1995, "Recommended Practices and Requirements for Harmonic Control in Electrical Power Systems", IEEE, 1993.
- [08] S. M. Halpin *et al*, "Voltage and lamp flicker issues: Should the IEEE adopt the IEC approach?", *IEEE Trans. on Power Delivery*, vol. 18, no. 3, Jul. 2003.
- [09] W. Xu, "Deficiency of the IEC flicker meter for measuring interharmonic caused voltage flickers", *Proc. Power Engineering Society general meeting*, 12-16 Jun. 2005.
- [10] E.W. Gunther, "A proposed flicker meter test protocol", *Proc. Int. Symp. on Quality and Security of Electric Power Delivery Systems*, 2003. 8-10 Oct. 2003.
- [11] NRS 048-2:2003, "Electricity supply - Quality of supply Part 2: Voltage characteristics, compatibility levels, limits and assessment methods", NRS 2003.
- [12] M. Rogóż, "The IEC flicker meter model", Unpublished, Sep. 2003.
- [13] A. Bień, Z. Hanzelka, M. Hartman, M. Piekarz and M. Szlosek, "Comparative test of flicker meters", *Proc. International conference on Harmonics and Quality of Power*, 2002.

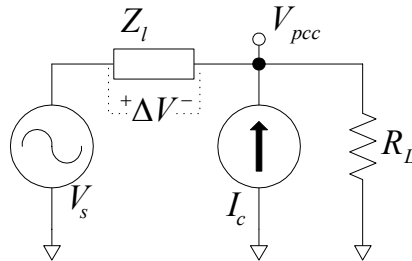
- [14] G. J. van Heerden, "Design and implementation of a DSP based controller for power electronic applications", *University of Stellenbosch M.Sc Eng. thesis*, Apr. 2003.
- [15] AD623 datasheet rev. C, Analog Devices, 1999.
- [16] G. F. Franklin, J. D. Powell and A. Emami-Naeini, "Feedback control of dynamic systems", *Prentice Hall*, 2002.
- [17] FTDI FT2232C Dual USB UART / FIFO I.C., FTDI, 2004.
- [18] FTDI FTD2XX programmer's guide, FTDI, 2004.
- [19] W. Stanley Jr., "Phenomena of Retardation in the Induction Coil", *American Institute of Electrical Engineers*, vol. 5, no. 4, pp 97-115, Jan. 1888.
- [20] O. B. Shallenberger, "The Distribution of Electricity by Alternative Current", *Electrical World*, pp. 114-115, 3 Mar. 1888.
- [21] C. I. Budeanu, "Pruissances reactives et fictives", *Institut Romain de l'Energie, Bucharest, Romania*, 1927.
- [22] S. Fryze, "Active, reactive and apparent power in circuits with sinusoidal voltage and current", *Przegląd Elektrot.*, no. 7, pp. 193-203, 1931.
- [23] A. E. Emanuel, "Suggested definition of reactive power in non-sinusoidal systems", *Proc. of the Institute of Electrical Engineers*, vol. 121, no. 7, Jul. 1974.
- [24] L. S. Czarnecki, "What is wrong with the Budeanu concept of reactive and distortion power and why it should be abandoned", *IEEE Trans. on Instrumentation and Measurement*, vol. 36, no. 3, Sept. 1987.
- [25] A. E. Emanuel, "Powers in non-sinusoidal situations - A review of definitions and physical meaning," *IEEE Trans. on Power Delivery*, vol. 5, pp. 1377-1389, Jul. 1990.
- [26] H. Supronowicz and J. Janczac, "Compensation of the reactive power drawn from the line by nonlinear consumers", *Proc. IEEE IAS Annual Meeting*, pp 1093-1098, 1990.
- [27] L.S. Czarnecki, "Orthogonal Decomposition of the currents in a 3-phase nonlinear asymmetrical circuit with a non-sinusoidal voltage source", *IEEE Trans. on Instrumentation and Measurement*, vol. 37, no. 1, pp.30-34, Mar. 1988.
- [28] L. Rossetto and P. Tenti, "Using ac-fed PWM converters as instantaneous reactive power compensators", *IEEE Trans. on Power Electronics*, vol. 7, no. 1, pp. 224-230, Jan. 1992.

- [29] A. Ferrero and G. Superti-Furga, "A new approach to the definition of power components in three-phase systems under non-sinusoidal conditions", *IEEE Trans. on Instrumentation and Measurement*, vol. 40, pp. 568-577, Jun. 1991.
- [30] A. Nabae and T. Tanaka, "A new definition of instantaneous active-reactive current-based on instantaneous space vectors on polar coordinates in three-phase circuits", *Conf. Record IEEE PES*, 1996.
- [31] H. Akagi, Y. Kanazawa and A Nabae, "Instantaneous power compensators comprising switching devices without energy storage components", *IEEE Trans. on Industry Applications*, vol. 20, no. 3, pp. 625-630, May/June. 1984.
- [32] L. S. Czarnecki, "On some misinterpretations of the instantaneous reactive power $p-q$ theory", *IEEE Trans. on Power Electronics*, vol. 19, no. 3, May 2004.
- [33] M. Kohata et al, "A novel compensator using static induction thyristor for reactive power and harmonics", *Proc. IEEE Industry Applications Society Annual Meeting*, vol. 35 no. 6, pp. 843-849, 2-7 Oct. 1988.
- [34] J. L. Willems, "A new interpretation of the Akagi-Nabae power components for non-sinusoidal three-phase situations", *IEEE Trans. on Instrumentation and Measurement*, vol. 41, pp. 523-527, Aug. 1992.
- [35] F. Z. Peng and J. S. Lai, "Generalised instantaneous reactive power theory for three-phase power systems", *IEEE Trans. on Instrumentation and Measurement*, vol. 45, no. 1, pp. 293-297, Feb. 1996.
- [36] F. Z. Peng, G. W. Ott, Jr., and D. J. Adams, "Harmonic and reactive power compensation based on the generalised instantaneous reactive power theory for three-phase four-wire systems," *IEEE Trans. on Power Electronics*, vol. 13, pp. 1174-1181, Nov. 1998.
- [37] M. J. Newman, D. N. Zmood and D.G. Holmes, "Stationary reference frame harmonics generations for active filter systems", *IEEE Trans. on Industry Applications*, vol. 38, no. 6, pp. 1591-1599, Nov./Dec. 2002.
- [38] G. D. Marques, "A comparison of active power filter control methods in unbalanced and non-sinusoidal conditions", *Proc. IEEE Industrial Electronics Society Annual Conference, IECON*, vol. 1, pp. 444-449, 31 Aug.-4 Sep. 1998.
- [39] M. P. Kazmierkowski and L. Malesani, "Current control techniques for three-phase voltage-source PWM converters: A survey", *IEEE Trans. on Industrial Electronics*, vol. 45, no. 5, Oct. 1998.

- [40] S. Buso, L. Malesani and P. Mattavelli, "Comparison of current control techniques for active filter applications", *IEEE Trans. on Industrial Electronics*, vol. 45, no. 5, Oct. 1998.
- [41] L. Malesani and S. Buso, "On the applications of active filters to generic loads" *Proc. 8th Int. Conf. on Harmonics and Quality of Power*, 14-16 Oct. 1998.
- [42] H. W. van der Broeck, H. C. Skudelny and G. Stanke, "Analysis and realization of a pulse width modulator based on voltage space vectors", *IEEE Trans. on Industry Applications*, vol. 24, pp. 142-150, Jan./Feb. 1988.
- [43] A. Kawamura and R. G. Hoft, "Instantaneous feedback controlled PWM inverters with adaptive hysteresis", *IEEE Trans. on Industry Applications*, vol. 20, pp. 769-775, Jul./Aug. 1984.
- [44] W. S. Oh, Y. T. Kim, H. J. Kim, "Dead time compensation of current-controlled inverter using space vector modulation method", *Proc. International conference on Power Electronic Systems and Drives*, vol.1, pp. 374-378, Feb. 1995.
- [45] M. G. Gous, "Shunt active power filtering algorithms for unbalanced, non-linear loads", *University of Stellenbosch M.Sc Eng. thesis*, p. 76, Dec. 2003.
- [46] Private discussion with H. Mostert, ESCOM Quality of Supply, 3 Mar. 2005.
- [47] E. Acha, V.G. Agelidis, O. Anaya-Lara and T.J.E. Miller, "Power electronic control in electrical systems", *Newnes*, 2002.
- [48] M. G. Becker, "Transformerless series dip/sag compensation with ultracapacitors", *University of Stellenbosch M.Sc Eng. thesis*, p. 42-52, Oct. 2003.
- [49] S. Huang and C Hsieh, "Application of continuous wavelet transform for study of voltage flicker generated signals", *IEEE Trans. on Aerospace and Electronic Systems*, vol. 36, is. 3, part 1, Jul. 2000.
- [50] A. Elnady and M.M.A. Salama, "Unified approach for mitigating voltage sag and voltage flicker using the DSTATCOM" *IEEE Trans. on Power Delivery*, vol. 20, is. 2, part 1, pp. 992 – 1000, Apr. 2005.

APPENDIX A: SELECTED PROOFS

A.1. Active vs. reactive compensation



In 1.2.3 the effect of reactive and active compensation was evaluated for a load drawing a large active power component. The same results obtained from the graphical derivations can be found by calculating the voltage at the PCC.

If the line is inductive, that is

$$Z_l = jX_l,$$

then the amplitude of the voltage at the PCC without compensation may be found as

$$|V_{pccX_l}| = |V_s| \frac{|R_L|}{|R_L + jX_l|}.$$

For the case where the line is resistive,

$$Z_l = R_l$$

the PCC voltage amplitude is

$$|V_{pccR_l}| = |V_s| \frac{|R_L|}{|R_L + R_l|}.$$

Using the triangle inequality, and assuming $X_l = R_l$ we can state

$$|V_{pccX_l}| \geq |V_{pccR_l}|.$$

This proves the important fact that active current flowing through a resistive line will cause a greater voltage drop than active current flowing through an inductive line. The converse is also true, and may be proven using the same method as above.

Adding a reactive compensator to the resistive case produces a line voltage magnitude

$$|V_{pccR_l I_{reactive}}| = \frac{|V_s R_L + jI_c R_l|}{|R_L + R_l|}$$

while active compensation produces

$$|V_{pccR_l I_{active}}| = \frac{|V_s R_L + I_c R_l|}{|R_L + R_l|}.$$

Again using the triangle inequality, we find

$$|V_{pccR_l I_{reactive}}| \leq |V_{pccR_l I_{active}}|$$

and thereby prove that active compensation will give better compensation than reactive compensation in this case. How much better the active compensation will be will depend on the exact circuit parameters.

A.2. Derivation of the SVC firing angle – impedance relationship

In Chapter 7.2 the relationship between the SVC thyristor firing angle, α , and the SVC susceptance is given as:

$$B_{SVC} = \frac{X_c [2(\pi - \alpha) + \sin 2\alpha] - \pi X_L}{\pi X_c X_L}$$

in Eq. 7-4. The derivation of this equation follows from the triac current waveform, an example of which can be found in Fig. 7-3. Remembering that α is defined as the angle difference between a voltage transition and the gate pulse of the triac and taking the time origin ($t = 0$), to fall on a

voltage peak, ($v = v_{\max}$) the instantaneous current i_{reactor} in any steady state half cycle can be written as

$$i_{\text{reactor}}(t) = \frac{\sqrt{2}V}{X_L}(\sin \omega t + \cos \alpha), \text{ where } \alpha - \frac{\pi}{2} < \omega t < \alpha + \frac{5\pi}{2}$$

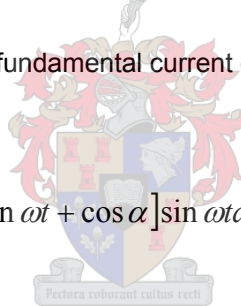
$$i_{\text{reactor}}(t) = 0 \text{ elsewhere.}$$

The amplitude of the fundamental current component may be found by Fourier series expansion. Again referring to Fig. 7-3, the current waveform can be seen to be odd, with half and quarter wave symmetry, allowing the fundamental Fourier coefficient to be written as

$$b_{\text{fundamental}} = \frac{8}{T} \int_0^{\frac{T}{4}} f(t) \sin \omega t dt$$

Calculating this coefficient by substituting $f(t) = i_{\text{reactor}}(t)$ and writing the angle bounds as time bounds using $\theta = \frac{2\pi t}{T}$ gives the fundamental current coefficient:

$$b_{\text{fundamental}} = \frac{8}{\sqrt{2}VTX_L} \int_{\frac{T(2\alpha-\pi)}{4\pi}}^{\frac{T}{4}} [\sin \omega t + \cos \alpha] \sin \omega t dt .$$



Evaluating this integral is done by using the integral solution

$$\int \sin px \sin qxdx = \frac{\sin(p - q)x}{2(p - q)} - \frac{\sin(p + q)x}{2(p + q)}$$

found in all good integration tables, and leads to the solution

$$b_{\text{fundamental}} = \frac{1}{\sqrt{2}V\pi X_L} [\sin 2\alpha + 2(\pi - \alpha)].$$

The RMS value of the fundamental current can be found by dividing this amplitude by $\sqrt{2}$. The susceptance of the inductor/triac combination at the fundamental frequency can be found by dividing by the RMS value of the fundamental current by the RMS voltage:

$$B_{\text{fundamental}} = \frac{1}{\pi X_L} [\sin 2\alpha + 2(\pi - \alpha)].$$

The last step in the derivation of the total SVC susceptance is calculating the total susceptance of the triac/inductor combination in parallel with the SVC capacitance. This is done using the well known formula:

$$B_{\parallel} = \frac{B_1 + B_2}{B_1 B_2}$$

Substituting the calculated inductor/triac combination susceptance as B_1 and the capacitor susceptance as B_2 gives Eq. 7-4.

A.3. SVC current harmonics

In A.1 the amplitude of the fundamental current component of an SVC was used to calculate the SVC susceptance. The inductor current was seen to be odd and half and quarter wave symmetrical. Thus the inductor current will consist only of odd current harmonics.

The SVC capacitance is fixed. This makes the capacitor current a function only of the line voltage. Assuming the line voltage to be sinusoidal, the capacitive arm of the SVC does not contribute to the harmonic currents drawn by the SVC. This causes finding the SVC inductor current harmonics to be equivalent to finding the SVC current harmonics.

Proceeding as was done in A.1, but calculating the general n^{th} Fourier coefficient by using

$$b_n = \frac{8}{T} \int_0^{\frac{T}{4}} f(t) \sin n\omega t dt$$

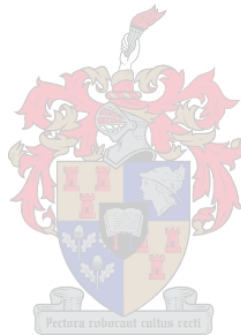
gives the coefficient integral as

$$b_n = \frac{8}{\sqrt{2} V T X_L} \int_{\frac{T(2\alpha-\pi)}{4\pi}}^{\frac{T}{4}} [\sin \omega t + \cos \alpha] \sin n\omega t dt.$$

The integration is straightforward, following the same pattern as the previous derivation. Collecting terms and simplifying leads to the general equation for SVC current harmonics:

$$I_n = \frac{4V}{\pi X_L} \left[\frac{\sin(n+1)\alpha}{2(n+1)} + \frac{\sin(n-1)\alpha}{2(n-1)} - \cos\alpha \frac{\sin n\alpha}{n} \right] \text{ with } n = 3, 5, 7, \dots$$

This is the same equation stated as Eq. 7-5.

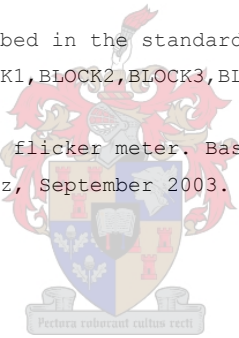


APPENDIX B: MATLAB FLICKER METER SIMULATION SCRIPT

```

%FLICKERMETER Evaluate the flicker in a signal.
% [FLICKERLEVEL,PST] = FLICKERMETER(X,BASEFREQUENCY,SAMPLEFREQUENCY)
% returns the flicker in the signal X.
%
% Frequencies should be specified in Hz.
%
% The function extrapolates the data X in the -t direction to calculate
% initial conditions for the flickermeter filters. In some cases the
% extrapolation may not be accurate enough, and some of the flicker meter
% output may have to be discarded in the statistical evaluation. The user
% has access to the index of the start of this evaluation by specifying it
% as a fourth parameter:
% FLICKERMETER(X,BASEFREQUENCY,SAMPLEFREQUENCY,EVALSTARTINDEX)
%
% The block outputs, as described in the standard, can be accessed by
% using [FLICKERLEVEL,PST,BLOCK1,BLOCK2,BLOCK3,BLOCK4] = FLICKERMETER(...)
%
% The function emulates an IEC flicker meter. Based on "The IEC
% Flickermeter Model" - M Rogoz, September 2003.
%
% Author: Leon de Wit
% University of Stellenbosch
% 20/08/2004

```



```

function [flickerLevel, Pst, block1, block2, block3, block4] = ...
    flickerMeter(X, baseFrequency, sampleFrequency, evalsIndexIn);

% Calibration
cConstant = 304.5e3;
prerunTime = 180;

% Sanity checks
if baseFrequency > sampleFrequency
    error('Base frequency can not be larger than sample frequency.');
```

```

end
if (length(X)-1)/sampleFrequency < 10*60
    warning('Pst not defined for data over a shorter than 10min period');
```

```

end

% Set up prerun to inhibit initial transients in calculated data
if nargin > 3

```

```

    evalOffset = evalSIndexIn - 1;
else
    evalOffset = 0;
end;
% prerunTime = length of the prerun in seconds
startIndex = 1;
data = X;
insertBlock = X(1:round(sampleFrequency/baseFrequency));
insertBlockLength = length(insertBlock);
for i = 1 : ceil(prerunTime*sampleFrequency/insertBlockLength);
    data = [insertBlock data];
    startIndex = startIndex + insertBlockLength;
end;

% Generate the time vector
t = 0 : 1/sampleFrequency : (length(data)-1)/sampleFrequency;

% Block 1
block1 = data./(max(0.5, lsim(tf([1],[10 1]), RMS(data,...
    baseFrequency, sampleFrequency), t)))');

% Block 2
block2 = block1.^2;

% Block 3
block3_1 = lsim(tf([1/2/pi/0.05 0], [1/2/pi/0.05 1]), block2, t);
wc = 2*pi*35;
block3_2 = lsim(tf([1],...
    [1/wc^6 3.864/wc^5 7.464/wc^4 9.141/wc^3 7.464/wc^2 3.864/wc 1]),...
    block3_1, t);
block3 = lsim(tf([1.74802*2*pi*9.15494 0],...
    [1 2*2*pi*4.05981 (2*pi*9.15494)^2])*tf([1/2/pi/2.27979 1],...
    [1/2/pi/1.22535/2/pi/21.9 (1/2/pi/1.22535 + 1/2/pi/21.9) 1]),...
    block3_2, t);

% Block 4
N = [cConstant];
D = [0.3 1];
block4 = lsim(tf(N,D), block3.^2, t);

% Remove prerun data
noPrerun = block4(startIndex + evalOffset : end);

% Results and Pst evaluation
flickerLevel = noPrerun;
noPrerun = sort(noPrerun);
lengthV = length(noPrerun);
P001 = noPrerun(round(lengthV/100*99.9));
P007 = noPrerun(round(lengthV/100*99.3));
P010 = noPrerun(round(lengthV/100*99));

```

```

P015 = noPrerun(round(lengthV/100*98.5));
P022 = noPrerun(round(lengthV/100*97.8));
P030 = noPrerun(round(lengthV/100*97));
P040 = noPrerun(round(lengthV/100*96));
P060 = noPrerun(round(lengthV/100*94));
P080 = noPrerun(round(lengthV/100*92));
P100 = noPrerun(round(lengthV/100*90));
P130 = noPrerun(round(lengthV/100*87));
P170 = noPrerun(round(lengthV/100*83));
P300 = noPrerun(round(lengthV/100*70));
P500 = noPrerun(round(lengthV/100*50));
P800 = noPrerun(round(lengthV/100*20));

P500s = (P300 + P500 + P800)/3;
P100s = (P060 + P080 + P100 + P130 + P170)/5;
P030s = (P022 + P030 + P040)/3;
P010s = (P007 + P010 + P015)/3;

Pst = sqrt(0.0314*P001 + 0.0525*P010s + 0.0657*P030s + 0.28*P100s + ...
          0.08*P500s);

%RMS RMS value of signal.
% RMS[X,BASEFREQUENCY,SAMPLEFREQUENCY] returns the RMS values
% of X.
%
% The averaging period for the RMS calculation is derived from
% the specified base frequency. The sample period of each sample
% is derived from the sample frequency.
%
% Frequencies may be specified in Hz, rad/s etc., but the two frequency
% parameters must share the same unit.

% Author: Leon de Wit
% University of Stellenbosch
% 19/08/2004

function rmsValues = rms1(X, baseFrequency, sampleFrequency)
delta = round(sampleFrequency/baseFrequency);
rmsValues = sqrt(filter(ones(1,delta),1,(X.^2)/delta));

```


APPENDIX C: USB DATA LOGGER

CODE & HARDWARE

C.1. PEC33 DSP code

```

/*****
/* DSP_Datalogger                                     */
/*****
/* DSP Kode vir die gebruik van die PEC33 bord as 'n USB datalogger      */
/*                                                                 */
/* DSP Code for using the PEC33 as a USB data logger                    */
/*                                                                 */
/* Ver 1.0 01/2005                                                       */
/* Leon de Wit                                                           */
/*****
/* External files */
#include <stdlib.h>
#include <stdio.h>
#include "A2D.h"
#include "LCD.h"
#include "timer0.h"
#include "address.h"
#include "USB.h"

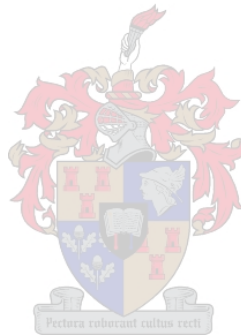
#define log_cycles 72000

inline void init(void);
void wait(void);
void log(void);

void (*execute_state)(void);
unsigned long cycles_logged = 0;

/*****
/* main                                               */
/*****
/* Implemented as a state machine                    */
/*****
main(void) {
    init();
    timer0_reset();
    while (1) {
        (*execute_state)();
    }
}

```



```

inline void init(void) {
    *prim_bus_ctrl = 0x1078;
        /* 1068(3wait,SWW=01) 10C8(6wait,SWW=01) 1078(3wait,SWW=11) */
    timer0_setup();
    LCD_setup();
    A2D_setup();
    execute_state = wait;
    USB_send_command(USB_idle);
}

void wait(void) {
    timer0_run_to_clocks(60000); /* 625 Hz */
    timer0_reset();
    LCD_disp();
    LCD_writeBuffer_line0("Waiting ");
    LCD_writeBuffer_line1("Press 0 to start. ");
    USB_send_command(USB_idle);
    cycles_logged = 0;
    if (KB_key_pressed() && (KB_get_key() == 0)) {
        USB_send_command(USB_start_log);
        execute_state = log;
    }
}

void log(void) {
/*    Filter implemented as
    Vout = Vin*K* z + b1
            -----
            z + a1
*/
    const float K = 0.1358, b1 = 1, a1 = -0.7285;
    float ch1 = 0, ch2 = 0, ch3 = 0, ch4 = 0;
    static float ch1z = 0, ch2z = 0, ch3z = 0, ch4z = 0;
    static float ch1y = 0, ch2y = 0, ch3y = 0, ch4y = 0;
    static int cntr = 0;
    timer0_run_to_clocks(3750); /* 10000 Hz */
    timer0_reset();
    LCD_disp();
    LCD_writeBuffer_line0("Logging ");
    LCD_writeBuffer_line1("Press 0 to stop. ");

    ch1 = K*(float)(adc0[adc_seq0_addr]);
    ch1y = ch1 + b1*ch1z - a1*ch1y;
    ch1z = ch1;

    ch2 = K*(float)(adc1[adc_seq0_addr]);
    ch2y = ch2 + b1*ch2z - a1*ch2y;
    ch2z = ch2;

```



```

ch3 = K*(float)(adc0[adc_seq4_addr]);
ch3y = ch3 + b1*ch3z - a1*ch3y;
ch3z = ch3;

ch4 = K*(float)(adc1[adc_seq4_addr]);
ch4y = ch4 + b1*ch4z - a1*ch4y;
ch4z = ch4;

if (++cntr == 2) {
    cntr = 0;
    USB_tx((unsigned short)(255)); /* Sync */
    USB_tx((unsigned short)(ch1y/4));
    USB_tx((unsigned short)(ch2y/4));
    USB_tx((unsigned short)(ch3y/4));
    USB_tx((unsigned short)(ch4y/4));
    ++cycles_logged;
}

A2D_sample();
if ((KB_key_pressed() && (KB_get_key() == 0)) ||
    cycles_logged == log_cycles) execute_state = wait;
}

/*****
*/ A2D.h */
/*****
*/ Algemene A na D biblioteek. */
/*
*/ General A to D library. */
/*
*/
/* Use this library by first calling A2D_setup. At present this function is */
/* empty, but to ensure future code compatibility it is recommended that this */
/* function be included in the setup procedure of the calling program. */
/* A2D_sample should be called next, instructing the A2D's to read a sample. */
/* The sample may then be read by calling A2D_read_x_x. */
/*
*/
/* Ver 1.0 11/2004 */
/* Leon de Wit */
/*****

#ifndef A2D
#define A2D

/*****
*/ A2D_setup */
/*****
*/ Set the A2D unit to the required configuration. */

```

```

/*****/
extern void A2D_setup(void);

/*****/
/* A2D_sample */
/*****/
/* Command all A2D's to commence sampling cycle. Must be called for each */
/* sampling cycle. */
/*****/
extern void A2D_sample(void);

/*****/
/* A2D_read_x_x */
/*****/
/* Return the last value read by the relevant A2D. */
/*****/
extern float A2D_read_0_0(void);
extern float A2D_read_0_1(void);
extern float A2D_read_0_2(void);
extern float A2D_read_0_3(void);
extern float A2D_read_0_4(void);
extern float A2D_read_0_5(void);
extern float A2D_read_0_6(void);
extern float A2D_read_0_7(void);
extern float A2D_read_1_0(void);
extern float A2D_read_1_1(void);
extern float A2D_read_1_2(void);
extern float A2D_read_1_3(void);
extern float A2D_read_1_4(void);
extern float A2D_read_1_5(void);
extern float A2D_read_1_6(void);
extern float A2D_read_1_7(void);
extern float A2D_read_2_0(void);
extern float A2D_read_2_1(void);
extern float A2D_read_2_2(void);
extern float A2D_read_2_3(void);
extern float A2D_read_2_4(void);
extern float A2D_read_2_5(void);
extern float A2D_read_2_6(void);
extern float A2D_read_2_7(void);

#endif /* A2D */

/*****/
/* A2D.c */
/*****/
/* Algemene A na D biblioteek. */
/* */

```



```

/* General A to D library. */
/*
/* Use this library by first calling A2D_setup. At present this function is
/* empty, but to ensure future code compatibility it is recommended that this
/* function be included in the setup procedure of the calling program.
/* A2D_sample should be called next, instructing the A2D's to read a sample.
/* Ensure that sufficient time elapses before reading the sample by calling
/* A2D_read_x_x.
/*
/* Ver 1.0 11/2004
/* Leon de Wit
/*****

/* External files */
#include "address.h"
#include "A2D.h"

/*****
/* A2D_setup
/*****
/* Set the A2D unit to the required configuration.
/*****
void A2D_setup(void) {}

/*****
/* A2D_sample
/*****
/* Command all A2D's to commence sampling cycle. Must be called for each
/* sampling cycle.
/*****
void A2D_sample(void) {
    *adc_samp_cmd = 1;
}

/*****
/* A2D_read_x_x
/*****
/* Return the last value read by the relevant A2D.
/*****
float A2D_read_0_0(void) {
    return (float)((adc0[adc_seq0_addr] & 0x3FF);
}

float A2D_read_0_1(void) {
    return (float)((adc0[adc_seq1_addr] & 0x3FF);
}

float A2D_read_0_2(void) {

```



```
        return (float)((adc0[adc_seq2_addr]) & 0x3FF);
    }

float A2D_read_0_3(void) {
    return (float)((adc0[adc_seq3_addr]) & 0x3FF);
}

float A2D_read_0_4(void) {
    return (float)((adc0[adc_seq4_addr]) & 0x3FF);
}

float A2D_read_0_5(void) {
    return (float)((adc0[adc_seq5_addr]) & 0x3FF);
}

float A2D_read_0_6(void) {
    return (float)((adc0[adc_seq6_addr]) & 0x3FF);
}

float A2D_read_0_7(void) {
    return (float)((adc0[adc_seq7_addr]) & 0x3FF);
}

float A2D_read_1_0(void) {
    return (float)((adc1[adc_seq0_addr]) & 0x3FF);
}

float A2D_read_1_1(void) {
    return (float)((adc1[adc_seq1_addr]) & 0x3FF);
}

float A2D_read_1_2(void) {
    return (float)((adc1[adc_seq2_addr]) & 0x3FF);
}

float A2D_read_1_3(void) {
    return (float)((adc1[adc_seq3_addr]) & 0x3FF);
}

float A2D_read_1_4(void) {
    return (float)((adc1[adc_seq4_addr]) & 0x3FF);
}

float A2D_read_1_5(void) {
    return (float)((adc1[adc_seq5_addr]) & 0x3FF);
}

float A2D_read_1_6(void) {
```



```

        return (float)((adc1[adc_seq6_addr]) & 0x3FF);
    }

float A2D_read_1_7(void) {
    return (float)((adc1[adc_seq7_addr]) & 0x3FF);
}

float A2D_read_2_0(void) {
    return (float)((adc2[adc_seq0_addr]) & 0x3FF);
}

float A2D_read_2_1(void) {
    return (float)((adc2[adc_seq1_addr]) & 0x3FF);
}

float A2D_read_2_2(void) {
    return (float)((adc2[adc_seq2_addr]) & 0x3FF);
}

float A2D_read_2_3(void) {
    return (float)((adc2[adc_seq3_addr]) & 0x3FF);
}

float A2D_read_2_4(void) {
    return (float)((adc2[adc_seq4_addr]) & 0x3FF);
}

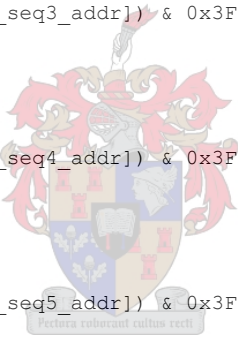
float A2D_read_2_5(void) {
    return (float)((adc2[adc_seq5_addr]) & 0x3FF);
}

float A2D_read_2_6(void) {
    return (float)((adc2[adc_seq6_addr]) & 0x3FF);
}

float A2D_read_2_7(void) {
    return (float)((adc2[adc_seq7_addr]) & 0x3FF);
}

/*****
/* LCD.h */
/*****
/* Algemene LCD biblioteek. */
/* Bevat fragmente kode van Pieter Henning. */
/* */
/* General LCD library. */
/* Fragments of this code was written by Pieter Henning. */
/* */

```



```

/* Use this library by first calling LCD_setup. */
/* Data is written to the two line LCD display by calling either */
/* LCD_writeBuffer_line0 or LCD_writeBuffer_line1. These functions write the */
/* strings to be displayed to the LCD output buffer. Place LCD_disp in the */
/* main program loop to have this data displayed on the LCD whenever the LCD */
/* is available. */
/* The LCD uses INT2 (IF4). This resource is therefore unavailable to the */
/* calling program */
/* */
/* Ver 1.0 11/2004 */
/* Leon de Wit */
/*****

#ifndef LCD
#define LCD

/*****
/* LCD_setup */
/*****
/* Set up the LCD by writing the initialization buffer to the LCD local */
/* memory. */
/*****
extern void LCD_setup(void);

/*****
/* LCD_writeBuffer_line0 */
/*****
/* Display a string on the top LCD line. */
/*****
extern void LCD_writeBuffer_line0(char* data);

/*****
/* LCD_writeBuffer_line1 */
/*****
/* Display a string on the bottom LCD line. */
/*****
extern void LCD_writeBuffer_line1(char* data);

/*****
/* LCD_disp */
/*****
/* Display data on LCD if LCD not busy, else exit. */
/*****
extern void LCD_disp(void);

#endif /* LCD */

/*****

```



```

/* LCD.c */
/*****
/* Algemene LCD biblioteek.
/* Bevat fragmente kode van Pieter Henning.
/*
/* General LCD library.
/* Fragments of this code was written by Pieter Henning.
/*
/* Use this library by first calling LCD_setup.
/* Data is written to the two line LCD display by calling either
/* LCD_writeBuffer_line0 or LCD_writeBuffer_line1. These functions write the
/* strings to be displayed to the LCD output buffer. Place LCD_disp in the
/* main program loop to have this data displayed on the LCD whenever the LCD
/* is available.
/* The LCD uses INT2 (IF4). This resource is therefore unavailable to the
/* calling program
/*
/* Ver 1.0 11/2004
/* Leon de Wit
*****/

```

```

/* External files */
#include <string.h>
#include <stdio.h>
#include "LCD.h"
#include "address.h"

```



```

/* Function prototypes */
void poll_IF4(void);

```

```

/* LDC initialisation and data buffers */

```

```

const unsigned int    lcd_init_buffer[]    =
{
    0x013c,           /* Initialise by instruction 1 */
    0x013c,           /* Initialise by instruction 2 */
    0x013c,           /* Initialise by instruction 3 */
    0x013c,           /* 8 bit, 2 lines, 5x10 */
    0x0108,           /* Display off, no cursor */
    0x0101,           /* Clear & home */
    0x0106,           /* Cursor incr, no shift */
    0x010c,           /* Display cursor, noblink on */
    'H','D','4','4','7','8','0','U', 0x0000
};

static struct{
    const unsigned int    clear0;           /* Cursor goto position 0 */
    char                  line0[32];       /* Line 0 data */
    const unsigned int    clear1;           /* Cursor goto position 40 (Line1) */
}

```

```

        char                line1[32];    /* Line 1 data */
    } lcd_data_buffer = {
        0x180 + 0,
        "                                ",
        0x180 + 40,
        "                                "};

/*****
/* LCD_setup                                                    */
/*****
/* Set up the LCD by writing the initialization buffer to the LCD local    */
/* memory.                                                         */
/*****
void LCD_setup(void) {
    int i = 0;
    while (i < sizeof(lcd_init_buffer)-1) {
        *lcd_load_cmd = lcd_init_buffer[i++];
        poll_IF4();
    }
    asm(" or 0004h, IF");    /* Set INT2 flag */
}

/*****
/* poll_IF4                                                    */
/*****
/* Loop until IF INT2 is set, then clear flag.                    */
/* Approximately 38us elapses from when data is written to the LCD until the */
/* LCD generates the interrupt.                                    */
/* The poll procedure clears the INT2 flag. Therefore the flag must be reset */
/* (in software) before control returns to main().                */
/*                                                                 */
/* Pieter Henning wrote:                                         */
/* procedure genereer loop tot IF se int 2 gestel word, dan word vlag */
/* geclear. Dit neem +- 38us vir LCD om int te generer na na lcd geskryf */
/* word.Die poll prosedure clear altyd die int2 flag, so voor main kan check */
/* vir int2 om te besluit of LCD besig is moet IF vir int 2 weer in */
/* sagteware gestel word.                                        */
/*****
void poll_IF4(void) {
    asm("data_loop");
    asm(" tstb 0004h, IF");
    asm(" BZ data_loop");
    asm(" ANDn 0004h, IF");
}

/*****
/* LCD_disp                                                    */
/*****
/* Display data on LCD if LCD not busy, else exit                */

```

```

/*****/
void LCD_disp(void) {
static int k = 0;
static const unsigned int *buffer = (const unsigned int*)&lcd_data_buffer;

static int LCD_status = 0;

asm("    tstb 0004h, IF");
asm("    BZ   LCD_nRDY");
asm("    ANDn 0004h, IF");
if (k < sizeof(lcd_data_buffer)) {
    *lcd_load_cmd = buffer[k];

    k++;
}
if ( k == sizeof(lcd_data_buffer)) { k = 0; }

asm("LCD_nRDY");
}

/*****/
/* LCD_writeBuffer_line0 */
/*****/
/* Display a string on the top LCD line. */
/*****/
void LCD_writeBuffer_line0(char* data) {
    strcpy(lcd_data_buffer.line0, data);
}

/*****/
/* LCD_writeBuffer_line1 */
/*****/
/* Display a string on the bottom LCD line. */
/*****/
void LCD_writeBuffer_line1(char* data) {
    strcpy(lcd_data_buffer.line1, data);
}

/*****/
/* timer0.h */
/*****/
/* Algemene tydhouer. */
/* */
/* General timer. */
/* */
/* Use this library by first calling timer0_setup. Hereafter waits can be */
/* implemented anywhere in the program by calling either timer0_wait_clocks */
/* or timer0_wait_ms */
/*****/

```

```

/*                                                    */
/* Ver 1.0 11/2004                                    */
/* Leon de Wit                                       */
/*****

#define TIMER0

/*****
/* timer0_setup                                     */
/*****
/* Set up the timer by writing the initialization constants to the predefined */
/* memory mappings.                               */
/*****
extern void timer0_setup(void);

/*****
/* timer0_wait_clocks                             */
/*****
/* Wait a user defined number of timer clock cycles. This function should be */
/* faster than timer0_wait_ms because the arithmetic necessary to translate */
/* required time to number of cycles is eliminated. However, should the timer */
/* frequency be changed all calls of this procedure would have to be updated. */
/*                                                    */
/* The timer (DSP) frequency is currently 75MHz.    */
/*****
extern void timer0_wait_clocks(int clocks);

/*****
/* timer0_wait_ms                                  */
/*****
/* Wait a user defined amount of time (specified in ms). This function is    */
/* slower than timer0_wait-clocks, but is more general as the response is not */
/* influenced by the timer period selected. The timer period can therefore be */
/* adjusted independently.                               */
/*****
extern void timer0_wait_ms(int ms);

/*****
/* timer0_reset                                    */
/*****
/* Reset the counter. It is not necessary to call this function before each */
/* call to any of the other timer functions.          */
/*****
extern void timer0_reset(void);

/*****
/* timer0_run_to_clocks                             */

```

```

/*****
/* Wait until the timer reaches the specified value. This may only be on      */
/* timer overflow. In conjunction with timer_reset it is possible to time    */
/* specific sections of code.                                               */
*****/
extern void timer0_run_to_clocks(int clocks);

#endif /* TIMER0 */

/*****
/* timer0.c                                                                    */
*****/
/* Algemene tydhouer.                                                         */
/*                                                                            */
/* General timer.                                                            */
/*                                                                            */
/* Use this library by first calling timer0_setup. Hereafter waits can be   */
/* implemented anywhere in the program by calling either timer0_wait_clocks */
/* or timer0_wait_ms                                                         */
/*                                                                            */
/* Ver 1.0 11/2004                                                           */
/* Leon de Wit                                                                */
*****/

/* External files */
#include "address.h"
#include "timer0.h"

/* Internal parameters */
#define TIMER0_PERIOD 0xFFFFFFFF /* Timer0 clocking at 37.5MHz. */
#define TIMER0_SETUP 0x3c0      /* 0x340 = 11 0100 0000 hold timer */

/*****
/* timer0_setup                                                                    */
*****/
/* Set up the timer by writing the initialization constants to the predefined */
/* memory mappings.                                                            */
*****/
void timer0_setup(void) {
    *timer0_period      =    TIMER0_PERIOD;
    *timer0_gc          =    TIMER0_SETUP;
    *timer0_counter = 0;
}

/*****
/* timer0_wait_clocks                                                                    */
*****/
/* Wait a user defined number of timer clock cycles. This function should be */

```

```

/* faster than timer0_wait_ms because the arithmetic necessary to translate */
/* required time to number of cycles is eliminated. However, should the timer */
/* frequency be changed all calls of this procedure would have to be updated. */
/*****/
void timer0_wait_clocks(int clocks) {
    *timer0_counter = 0;
    while (*timer0_counter < clocks) {}
}

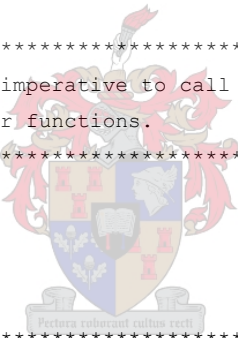
/*****/
/* timer0_wait_ms */
/*****/
/* Wait a user defined amount of time (specified in ms). This function is */
/* slower than timer0_wait-clocks, but is more general as the response is not */
/* influenced by the timer period selected. The timer period can therefore be */
/* adjusted independently. */
/*****/
/* ! Not implemented ! */

/*****/
/* timer0_reset */
/*****/
/* Reset the counter. It is not imperative to call this function before each */
/* call to any of the other timer functions. */
/*****/
void timer0_reset(void) {
    *timer0_counter = 0;
}

/*****/
/* timer0_run_to_clocks */
/*****/
/* Wait until the timer reaches the specified value. This may only be on */
/* timer overflow. In conjunction with timer_reset it is possible to time */
/* specific sections of code. */
/*****/
void timer0_run_to_clocks(int clocks) {
    while (*timer0_counter < clocks) {}
}

/*****/
/* address */
/*****/
/* Algemene adreslys. */
/* Hoofsaaklik kode van Pieter Henning. */
/* */
/* General address list. */
/* Most of this code was written by Pieter Henning. */

```



```

/*                                                                    */
/* Ver 1.0 11/2004                                                    */
/* Leon de Wit                                                         */
/******                                                              */

#ifndef ADDRESS
#define ADDRESS

/* Pieter Henning wrote:
/* Header file containing register address declarations and constant definitions */
/* exbus declarations is bygevoeg 10-03-2003 ph2                       */

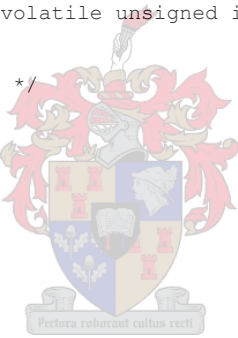
/* DSP timer0 register address declarations                            */
#define timer0_gc ((volatile unsigned int *)0x808020)
/* Pieter Henning wrote: laai timer global controll register se adress */
#define timer0_period ((volatile unsigned int *)0x808028)
#define timer0_counter ((volatile unsigned int *)0x808024)
/* DSP timer1 register address declarations                            */
#define timer1_gc ((volatile unsigned int *)0x808030)
#define timer1_period ((volatile unsigned int *)0x808038)
#define timer1_counter ((volatile unsigned int *)0x808034)

/* DSP Main constant definitions */
/* Serie port */
#define global_ctrl 0
#define tx_ctrl 2
#define rx_ctrl 3
#define timer_ctrl 4
#define timer_cntr 5
#define timer_period 6
#define tx_data 8
#define rx_data 0xC

/* DSP register address declarations */
#define serie0 ((int *)0x808040)
#define prim_bus_ctrl ((volatile unsigned int *)0x808064)

/* FPGA Main constant definitions */
/* RTC register address declaration
base *rtc=(int *)0x500010; */
#define rtc_status 0x0
#define rtc_wr_addr 0x1
#define rtc_wr_data 0x2
#define rtc_read_sec 0x3
#define rtc_read_min 0x4
#define rtc_read_hour 0x5
#define rtc_read_day 0x6
#define rtc_read_date 0x7

```



```

#define rtc_read_month      0x8
#define rtc_read_year      0x9
#define rtc_load           0xB

#define rtc_write_sec      0x60
#define rtc_write_min      0x61
#define rtc_write_hour     0x62
#define rtc_write_day      0x63
#define rtc_write_date     0x64
#define rtc_write_month    0x65
#define rtc_write_year     0x66
#define rtc_write_ctrl     0x67

/* Optical sender & resiver addres declarations
base *optic=(int *) 0x500068; */
#define opt_rx_0           0x0      /*lees 0 as lig af is          */
#define opt_tx_0           0x1      /*skakel lig aan as 1 skyf na opt_tx */
#define opt_rx_1           0x2
#define opt_tx_1           0x3

/* FPGA Main register address declarations */
#define RS232Port          ((volatile int *)0x500000)
#define fram_reg           ((volatile int *)0x500008)
#define rtc                ((volatile int *)0x500010)
#define optic              ((volatile int *)0x500068)

/* #define rtc_wr_data      ((int *)0x500012) */
/* volatile int *rtc_rd_data ((int *)0x500013) */

#define main_cmd0          ((volatile int *)0x500066)
#define main_cmd1          ((volatile int *)0x500067)

/* USB interface */
#define USB_tx_reg         ((volatile unsigned int *)0x500080)
#define USB_rx_reg         ((volatile unsigned int *)0x500082)

/* FRAM constant definitions */
/* #define TX_Data 0 #define RX_Data 1 #define Status 2 #define BaudRate 3*/
/* base *fram_reg=(int *)0x500008 register wat fram RDYnBSY seine ontvang;
bit 0 = fram0 RDYnBSY
bit 1 = fram1 RDYnBSY */
#define fram_status       0

/* FRAM register address declarations */
#define fram0              ((volatile int *)0x400000)
#define fram1              ((volatile int *)0xC00000)

/* FPGA Analog constant definitions */

```



```

/* DAC registers */
#define conf_addr          0
#define a_data_addr       1
#define b_data_addr       2
#define load_dac          3

/* ADC Registers */
/* die data word gestoor vlg waar in die sekwensie dit
/* gesample is */
#define conf_adc          0
#define adc_seq0_addr    1
#define adc_seq1_addr    2
#define adc_seq2_addr    3
#define adc_seq3_addr    4
#define adc_seq4_addr    5
#define adc_seq5_addr    6
#define adc_seq6_addr    7
#define adc_seq7_addr    8

/* config data vir dac. ( ch# bit 9,8,7 ) */
/* seq low data (16 bit) "seq3|seq2|seq1|seq0" example
/* "x011|x010|x001|x000" ->
/* ch3| ch2| ch1| ch0
/* seq0 word eerste gesample dan 2 .... */
#define adc_conf_seq_low ((volatile int *)0x600064)
#define adc_conf_seq_high ((volatile int *)0x600065)

/* FPGA Analog register address declarations */
#define dac0 ((volatile int *)0x600000)
#define dac1 ((volatile int *)0x600004)
#define dac2 ((volatile int *)0x600008)
#define dac3 ((volatile int *)0x60000C)

#define adc0 ((volatile int *)0x600010)
#define adc1 ((volatile int *)0x600019)
#define adc2 ((volatile int *)0x600022)
#define adc3 ((volatile int *)0x60002B)

#define adc_samp_cmd ((volatile int *)0x600034)
#define lcd_load_cmd ((volatile int *)0x600035)
#define keypad_load_cmd ((volatile int *)0x600045)
#define soft_status ((volatile int *)0x600046)
#define breaker_status ((volatile int *)0x600047)
#define LCD_RDYnBSY ((volatile int *)0x600048)

```

```

/* dig meetings se register adresse */
#define dig_Vac0_reg ((volatile int *)0x600036)
#define dig_Vac1_reg ((volatile int *)0x600037)
#define dig_Vac2_reg ((volatile int *)0x600038)
#define dig_Vac3_reg ((volatile int *)0x600039)
#define dig_Vac4_reg ((volatile int *)0x60003A)
#define dig_Vac5_reg ((volatile int *)0x60003B)
#define dig_Vac6_reg ((volatile int *)0x60003C)

#define dig_Vdc1_reg ((volatile int *)0x60003D)
#define dig_Vdc2_reg ((volatile int *)0x60003E)
#define dig_Vdc3_reg ((volatile int *)0x60003F)
#define dig_Vdc4_reg ((volatile int *)0x600040)
#define dig_Vdc5_reg ((volatile int *)0x600041)
#define dig_Vdc6_reg ((volatile int *)0x600042)
#define dig_Vdc7_reg ((volatile int *)0x600043)
#define dig_VdcTOT_reg ((volatile int *)0x600044)

/* exbus adresse en data def, periferal swiches and contacters
/* net 6 bits beskikbaar vir adresse
/*
-- adress 0000H vlag_enable sein data(0) 1 -> enable
/*
-- adress 0001H dienssiklus A data(9-0)
/*
-- adress 0002H dienssiklus B data(9-0)
/*
-- adress 0003H dienssiklus C data(9-0)
/*
-- adress 0004H sw fan data(0) 0 -> sw aan
/*
-- adress 0005H dc breeker data(0) 0 -> sw aan
/*
-- adress 0006H ac breeker data(0) 0 -> sw aan
/*
-- adress 0007H dc dump "data bits 6 tot 0" data(6-0) 0 -> sw aan
/*
-- adress 0008H sw SoftStrater data(0) 0 -> sw aan

/* adresse PWMv7b & relay v3 */
#define PWM_enable 0
#define PWM_Diens_A 1
#define PWM_Diens_B 2
#define PWM_Diens_C 3
#define sw_Fan 4
#define sw_DC_breeker 5
#define sw_AC_breeker 6
#define sw_DC_Dump 7 /*48*/
#define sw_softStarter 8
#define Lees_Fout 9

/* EPLD ExBus0 register address declarations */
#define ExBus0 ((volatile int *)0xA00000)

/* EPLD ExBus1 register address declarations */
#define ExBus1 ((volatile int *)0xB00000)

```

```
#endif /* ADDRESS */

/*****
/* USB.h
/*****
/* USB konneksie vir die PEC33.
/* Gebruik hierdie funksies om met die PC te kommunikeer.
/*
/* USB connction for the PEC33
/* Use these function to communicate with a PC.
/*
/* Ver 1.0 11/2004
/* Leon de Wit
/*****

#ifndef USB
#define USB

#include "address.h"

#define USB_idle          0
#define USB_set_log_time  1
#define USB_start_log     2
#define USB_log_end       3

void USB_tx(unsigned short data);
void USB_send_command(unsigned short com);

#endif

/*****
/* USB
/*****
/* USB konneksie vir die PEC33.
/* Gebruik hierdie funksies om met die PC te kommunikeer.
/*
/* USB connction for the PEC33.
/* Use these function to communicate with a PC.
/*
/* Ver 1.0 11/2004
/* Leon de Wit
/*****

#include "USB.h"

void USB_tx(unsigned short data) {
    *USB_tx_reg = data;
    asm("        NOP");
```



```
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");

asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");

asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");

asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");
asm("    NOP");

}

void USB_send_command(unsigned short com) {
    USB_tx(com);
}
```



```

/*****
/* KB.h
/*****
/* Algemene sleutelbord biblioteek.
/* Bevat fragmente kode van Pieter Henning.
/*
/* General keyboard library.
/* Fragments of this code was written by Pieter Henning.
/*
/* Use this library by calling KB_key_pressed to determine whether a new key
/* is available. If a key has been pressed the relevant scan code may be
/* retrieved by calling KB_get_key.
/* The KB uses INT0. This resource is therefore unavailable to the calling
/* program.
/*
/* Ver 1.0 11/2004
/* Leon de Wit
/*****

#ifndef KB
#define KB

/*****
/* KB_key_pressed
/*****
/* Return 1 if key has been pressed, 0 otherwise.
/*****
extern int KB_key_pressed(void);

/*****
/* KB_get_key
/*****
/* Return the value of the last key pressed.
/*****
extern int KB_get_key(void);

#endif /* KB */

/*****
/* KB.c
/*****
/* Algemene sleutelbord biblioteek.
/* Bevat fragmente kode van Pieter Henning.
/*
/* General keyboard library.
/* Fragments of this code was written by Pieter Henning.
/*
/* Use this library by calling KB_key_pressed to determine whether a new key

```



```

/* is available. If a key has been pressed the relevant scan code may be      */
/* retrieved by calling KB_get_key.                                          */
/* The KB uses INT0. This resource is therefore unavailable to the calling    */
/* program.                                                                    */
/*                                                                            */
/* Ver 1.0 11/2004                                                            */
/* Leon de Wit                                                                */
/*****

/* External files */
#include "KB.h"
#include "address.h"

/*****
/* KB_key_pressed                                                                    */
/*****
/* Return 1 if key has been pressed, 0 otherwise.                                */
/*****
int KB_key_pressed(void) {
    asm(" LDIU 00h, R0");
    asm(" tstb 0001h, IF");          /* Test INT0 flag */
    asm(" BZ  keypad_poll");

    asm(" LDIU 01h, R0");
    asm("keypad_poll");
}

/*****
/* KB_get_key                                                                    */
/*****
/* Return the value of the last key pressed.                                    */
/*****
int KB_get_key(void) {
    asm(" ANDn 0001h, IF");          /* Clear INT0 flag */
    return(*keypad_load_cmd & 0xf);
}

```



C.2. PEC33 FPGA VHDL code

```

-- USB controller for FTDI FT2232C in 245 FIFO mode

-- Leon de Wit
-- Desember 2004

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

```

```

-----
entity USB_controller is
    port( clk, not_reset                : in std_logic;
          -- FT2232C Command bus
          not_TXE, not_RXF              : in std_logic;
          not_RD, WR, SI_WU              : out std_logic;
          -- FT2232C Data bus
          FT_data                        : inout std_logic_vector(7 downto 0);
          -- Adress controller interface
          data_av                         : in std_logic;
          tx_data                         : in std_logic_vector(7 downto 0);
          rx_data                         : out std_logic_vector(8 downto 0);

          state_out                      : out std_logic_vector(7 downto 0)
    );
end entity USB_controller;
-----

```

```

-----
architecture a1 of USB_controller is
    type state_type is (reset, waiting, wait_for_TXE, tx1, tx2, tx_end, rx1,
                        rx2, rx3, rx4, rx5, rx_end1,
                        rx_end2, rx_end3);
    signal state, next_state              : state_type;
    signal tx_data_buffer                 : std_logic_vector(7 downto 0);
    signal data_av_buffer, not_TXE_buffer : std_logic;
    signal rx_data_buffer                 : std_logic_vector(8 downto 0);
    signal not_RXF_buffer                 : std_logic;
begin
-----

```

```

sequencer : process (clk, not_reset)
begin
    if (not_reset = '0') then
        state <= reset;
    elsif rising_edge(clk) then
        state <= next_state;
    end if;
end process sequencer;
-----

```

```

sequence : process (state)
begin
    case state is
        when reset =>
            WR <= '0';
            not_RD <= '1';
            rx_data_buffer <= "000000000";
            FT_data <= "ZZZZZZZZ";
            next_state <= waiting;

```

```
state_out <= "00000001";

when waiting =>
  WR <= '0';
  not_RD <= '1';
  FT_data <= "ZZZZZZZZ";
  if (not_RXF_buffer = '0') then
    next_state <= rx1;
  else
    if ((data_av_buffer = '1') and (not_TXE_buffer = '0')) then
      next_state <= tx1;
    elsif ((data_av_buffer = '1') and (not_TXE_buffer = '1')) then
      next_state <= wait_for_TXE;
    else
      next_state <= waiting;
    end if;
  end if;

state_out <= "00000010";

when wait_for_TXE =>
  WR <= '0';
  not_RD <= '1';
  FT_data <= "ZZZZZZZZ";
  if (not_TXE_buffer = '0') then
    next_state <= tx1;
  else
    next_state <= wait_for_TXE;
  end if;

state_out <= "00000100";

when tx1 =>
  WR <= '1';
  not_RD <= '1';
  FT_data <= tx_data_buffer;
  next_state <= tx2;

state_out <= "00000100";

when tx2 =>
  WR <= '1';
  not_RD <= '1';
  FT_data <= tx_data_buffer;
  next_state <= tx_end;

state_out <= "00000100";
```



```
when tx_end =>
    WR <= '0';
    not_RD <= '1';
    FT_data <= tx_data_buffer;
    next_state <= waiting;

    state_out <= "00000100";

when rx1 =>
    WR <= '0';
    not_RD <= '0';
    FT_data <= "ZZZZZZZZ";
    next_state <= rx2;

    state_out <= "00001000";

when rx2 =>
    WR <= '0';
    not_RD <= '0';
    FT_data <= "ZZZZZZZZ";
    next_state <= rx3;

    state_out <= "00001000";

when rx3 =>
    WR <= '0';
    not_RD <= '0';
    rx_data_buffer(7 downto 0) <= FT_data;
    rx_data_buffer(8) <= not rx_data_buffer(8);
    next_state <= rx4;

    state_out <= "00001000";

when rx4 =>
    WR <= '0';
    not_RD <= '0';
    rx_data_buffer(7 downto 0) <= FT_data;
    rx_data_buffer(8) <= not rx_data_buffer(8);
    next_state <= rx5;

    state_out <= "00001000";

when rx5 =>
    WR <= '0';
    not_RD <= '0';
    rx_data_buffer(7 downto 0) <= FT_data;
    rx_data_buffer(8) <= not rx_data_buffer(8);
    next_state <= rx_end1;
```

```

        state_out <= "00001000";

    when rx_end1 =>
        WR <= '0';
        not_RD <= '1';
        FT_data <= "ZZZZZZZZ";
        next_state <= rx_end2;

        state_out <= "00001000";

    when rx_end2 =>
        WR <= '0';
        not_RD <= '1';
        FT_data <= "ZZZZZZZZ";
        next_state <= rx_end3;

        state_out <= "00001000";

    when rx_end3 =>
        WR <= '0';
        not_RD <= '1';
        FT_data <= "ZZZZZZZZ";
        next_state <= waiting;

        state_out <= "00001000";

    when others =>
        WR <= '0';
        not_RD <= '1';
        FT_data <= tx_data_buffer;
        next_state <= reset;

    end case;
end process sequence;
-----
tx_data_buffering : process (clk, state)
begin
    if rising_edge(clk) then
        if (state = waiting) then
            tx_data_buffer <= tx_data;
        end if;
    end if;
end process tx_data_buffering;
-----
data_av_buffering : process (clk, data_av)
begin
    if rising_edge(clk) then
        data_av_buffer <= data_av;

```

```

        end if;
    end process data_av_buffering;
-----
not_TXE_buffering : process (clk, not_TXE)
begin
    if rising_edge(clk) then
        not_TXE_buffer <= not_TXE;
    end if;
end process not_TXE_buffering;
-----
-- RX SECTION
-----
not_RXF_buffering : process (clk, not_RXF)
begin
    if rising_edge(clk) then
        not_RXF_buffer <= not_RXF;
    end if;
end process not_RXF_buffering;
-----
-- GENERAL
-----
    SI_WU <= '1';
    rx_data <= rx_data_buffer;
-----
end architecture a1;

```

The address decoding for the USB block defined above had to be added to the existing address decoder. Only the relevant code extract is shown here:

```

-----
USB_sequencer: process (nReset, H1)
begin
    if (nReset = '0') then
        USB_state <= USB_reset;
    elsif rising_edge(H1) then
        USB_state <= USB_next_state;
    end if;
end process USB_sequencer;
-----
USB_process: process (USB_state, USB_tx_en, DSP_RnW, Data)
begin
    case USB_state is
        when USB_reset =>
            USB_data_av_buffer <= '0';
            USB_next_state <= USB_waiting;
        when USB_waiting =>
            USB_data_av_buffer <= '0';

```

```

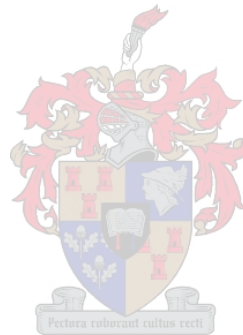
        if ((USB_tx_en = '1') and (DSP_RnW = '0')) then
            USB_next_state <= USB_load;
        else
            USB_next_state <= USB_waiting;
        end if;
    when USB_load => -- The data is loaded in a separate process,
        USB_data_av_buffer <= '0';
-- USB_data_buffering
        USB_next_state <= USB_tx_data1;
    when USB_tx_data1 =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_tx_data2;
    when USB_tx_data2 =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_tx_data3;
    when USB_tx_data3 =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_tx_data4;
    when USB_tx_data4 =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_tx_data5;
    when USB_tx_data5 =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_tx_data6;
    when USB_tx_data6 =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_tx_data7;
    when USB_tx_data7 =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_tx_data8;
    when USB_tx_data8 =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_tx_end;
    when USB_tx_end =>
        USB_data_av_buffer <= '1';
        USB_next_state <= USB_waiting;
    when others =>
        USB_data_av_buffer <= '0';
        USB_next_state <= USB_reset;
    end case;
end process USB_process;

-----
USB_data_buffering : process (USB_state, H1)
begin
    if rising_edge(H1) then
        if (USB_state = USB_load) then
            USB_tx_buffer <= Data(7 downto 0);
        end if;
    end if;
end process;

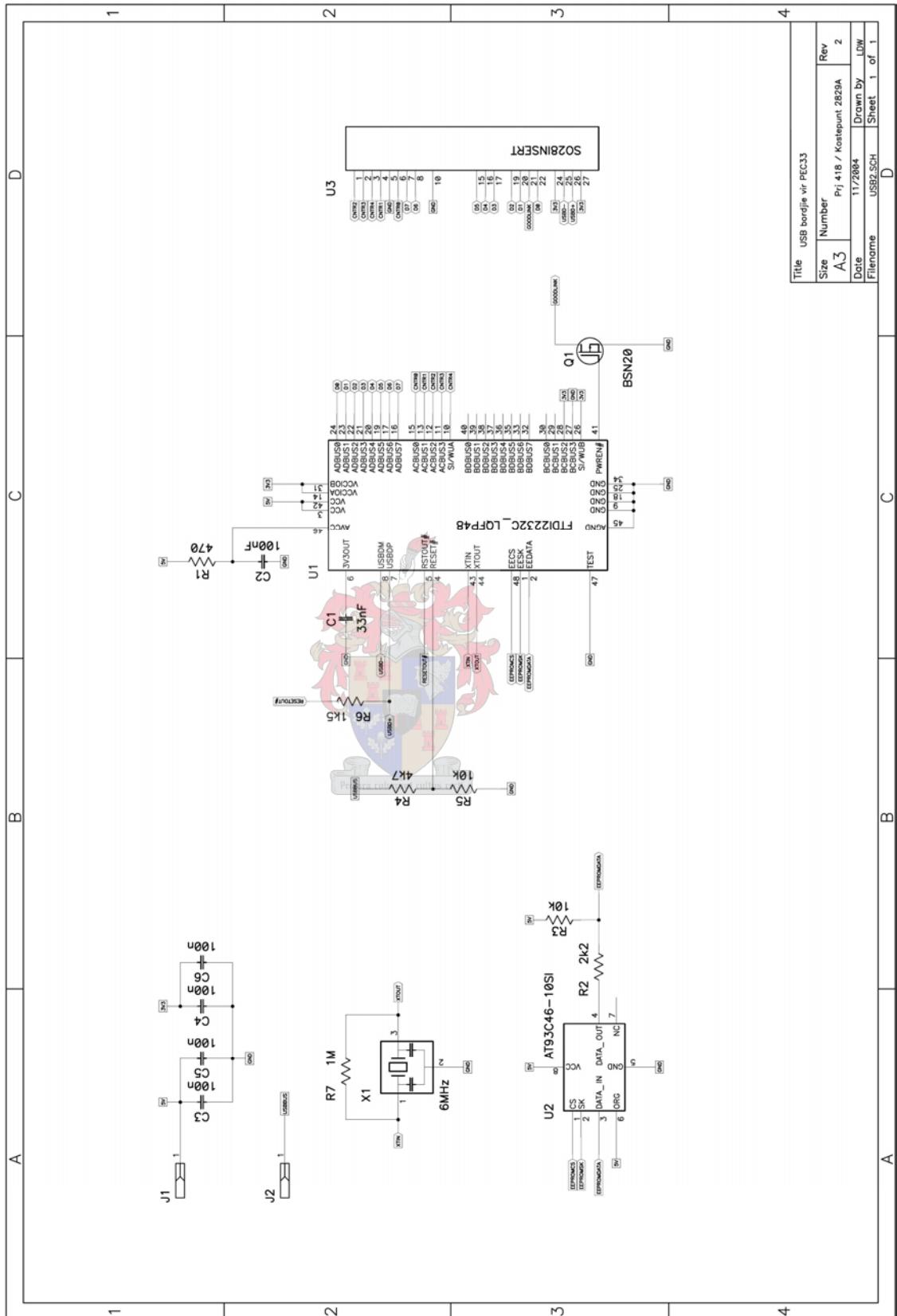
```

```
        end if;
    end process USB_data_buffering;
-----
USB_data_av_buffering : process (H1)
begin
    if (nReset = '0') then
        USB_data_av <= '0';
    elsif falling_edge(H1) then
        USB_data_av <= USB_data_av_buffer;
    end if;
end process USB_data_av_buffering;
-----

USB_data_tx <= USB_tx_buffer;
```



C.3. USB daughter board schematic



C.4. C++ code for PC

```
/* *****  
/* DataProcessor.cpp */  
/* *****  
/* Ver 1.0 01/2005 */  
/* Leon de Wit */  
/* *****  
  
#include <cstdlib>  
#include <iostream>  
#include <fstream.h>  
#include "USBCon.h"  
#include "windows.h"  
  
void waitForLogger();  
void waitForLogStart();  
void saveData();  
void quitOnError(char *message);  
  
const DWORD maxBytesReadAtATime = 64; // FTDI chip buffer size is 128 bytes  
const unsigned long dataBufferSize = 0x4000;  
const int numDataBuffers = 0x2; // Equals the number of files that will be written //db  
char filename[64];  
fstream outfile[numDataBuffers];  
  
USBCon Con1("USB Datalogger A");  
  
int main(int argc, char* argv[]) {  
    std::cout << "USB Datalogger Ver 1.0\n\n";  
  
    // Open all the files required for writing  
    for (int i = 0; i < numDataBuffers; i++) {  
        sprintf(filename, "PEG%i.dat", i);  
        outfile[i].open(filename, ios::out | ios::binary);  
    }  
  
    Con1.purgeRxQueue();  
    waitForLogger();  
    waitForLogStart();  
    saveData();  
    std::cout << "Logging successfull\n";  
  
    // Close all the opened files  
    for (i = 0; i < numDataBuffers; i++) {  
        outfile[i].close();  
    }  
}
```

```

        return 0;
    }

void waitForLogger() {
    DWORD numDevices = 0;
    DWORD dummy;
    std::cout << "Waiting for logger\n";
    while (numDevices == 0) {
        Con1.listDevices((PVOID) (&numDevices), (PVOID) (&dummy), FT_LIST_NUMBER_ONLY);
    }
}

void waitForLogStart() {
    DWORD amountInRxQueue = 0;
    DWORD dummy;
    unsigned short dataBuffer = 0;
    DWORD bytesRead = 0;
    std::cout << "Waiting for logger startup\n";
    while (dataBuffer != 2) {
        while (amountInRxQueue == 0) {
            if (Con1.getDetails((LPDWORD) (&amountInRxQueue), (LPDWORD) (&dummy),
                               (LPDWORD) (&dummy)) != USB_OK) {
                quitOnError("Datalogger failed startup.");
            }
        }
        if (Con1.getData((LPVOID) (&dataBuffer), 1, (LPDWORD) (&bytesRead)) != USB_OK ||
            bytesRead != 1) {
            quitOnError("Datalogger failed startup.");
        }
    }
}

void saveData() {
    char dataBuffer[dataBufferSize];
    int bytesRequired, dataBufferPosition;
    DWORD bytesRead = 0;

    unsigned int zeroTimeouts = 0;
    const unsigned int MAX_ZERO_TIMEOUTS = 65000;

    USB_STATUS status;

    std::cout << "Logging data\n";

    // Purge the queue of commands and startup noise
    Con1.purgeRxQueue();

    // Read the incoming data and store to file

```



```

for (int i = 0; i < numDataBuffers; i++) {
    // Fill the databuffer
    bytesRequired = min(maxBytesReadAtATime, dataBufferSize);
    dataBufferPosition = 0;
    while (bytesRequired) {
        status = Con1.getData((LPVOID) (&dataBuffer[dataBufferPosition]),
                               bytesRequired, (LPDWORD) (&bytesRead));

        if (!bytesRead) zeroTimeouts++;
        if (status != USB_OK || zeroTimeouts >= MAX_ZERO_TIMEOUTS) {
            // Close all the opened files
            for (i = 0; i < numDataBuffers; i++) {
                outfile[i].close();
            }
            quitOnError("Error retrieving data");
        }
        dataBufferPosition += bytesRead;
        bytesRequired = min(maxBytesReadAtATime, dataBufferSize - dataBufferPosition);
    }
    // Save databuffer to file
    outfile[i].write(dataBuffer, dataBufferSize);
}

void quitOnError(char *message) {
    std::cout << "Error: " << message << "\n";
    std::cout << "Final USB error code: " << Con1.getStatus() << "\n";
    std::cout << "Data could not be logged\n";

    // Close all the opened files
    for (int i = 0; i < numDataBuffers; i++) {
        outfile[i].close();
    }

    exit(EXIT_FAILURE);
}

/*****
/* USBCon.h
/* Ver 1.0 01/2005
/* Leon de Wit
*****/

#ifndef USBCON_H
#define USBCON_H

#include "windows.h"
#include "FTD2XX.H"

```

```

/* Default timeouts */
#define TX_TIMEOUT 300
#define RX_TIMEOUT 300

enum USB_STATUS {
    USB_OK,
    USB_DLL_ERROR,
    USB_COULD_NOT_OPEN,
    USB_COULD_NOT_SET_TIMEOUTS,
    USB_COULD_NOT_PURGE_RX,
    USB_COULD_NOT_PURGE_TX,
    USB_COULD_NOT_READ,
    USB_COULD_NOT_WRITE,
    USB_COULD_NOT_CLOSE
};

class USBCon {
public:
    USBCon(char *descriptor);
    USB_STATUS getStatus(); // Returns the first change from USB_OK, else USB_OK
    USB_STATUS setTimeouts(ULONG txTimeout, ULONG rxTimeout);
                                // Timeouts in ms, 300ms default

    USB_STATUS purgeRxQueue();
    USB_STATUS getData(LPVOID lpBuffer, DWORD dwBytesToRead, LPDWORD lpdwBytesReturned);
    USB_STATUS purgeTxQueue();
    USB_STATUS sendData(LPVOID lpBuffer, DWORD dwBytesToWrite, LPDWORD lpdwBytesWritten);
    USB_STATUS getDetails(LPDWORD lpdwAmountInRxQueue, LPDWORD lpdwAmountInTxQueue,
                                LPDWORD lpdwEventStatus);

    USB_STATUS listDevices(PVOID pvArg1, PVOID pvArg2, DWORD dwFlags);
    ~USBCon();

private:
    void linkDLL();
    void updateStatus(USB_STATUS);
    HMODULE thisHModule;
    FT_HANDLE thisFtHandle;
    USB_STATUS status;
};

#endif

/*****
/* USBCon.c */
/*****
/* Ver 1.0 01/2005 */
/* Leon de Wit */
/*****

```

```

#include "USBCon.h"

/* Pointers to DLL functions */
typedef FT_STATUS (WINAPI *PtrToOpen)(PVOID, FT_HANDLE *);
PtrToOpen USB_Open;
typedef FT_STATUS (WINAPI *PtrToOpenEx)(PVOID, DWORD, FT_HANDLE *);
PtrToOpenEx USB_OpenEx;
typedef FT_STATUS (WINAPI *PtrToListDevices)(PVOID, PVOID, DWORD);
PtrToListDevices USB_ListDevices;
typedef FT_STATUS (WINAPI *PtrToClose)(FT_HANDLE);
PtrToClose USB_Close;
typedef FT_STATUS (WINAPI *PtrToRead)(FT_HANDLE, LPVOID, DWORD, LPDWORD);
PtrToRead USB_Read;
typedef FT_STATUS (WINAPI *PtrToWrite)(FT_HANDLE, LPVOID, DWORD, LPDWORD);
PtrToWrite USB_Write;
typedef FT_STATUS (WINAPI *PtrToResetDevice)(FT_HANDLE);
PtrToResetDevice USB_ResetDevice;
typedef FT_STATUS (WINAPI *PtrToPurge)(FT_HANDLE, ULONG);
PtrToPurge USB_Purge;
typedef FT_STATUS (WINAPI *PtrToSetTimeouts)(FT_HANDLE, ULONG, ULONG);
PtrToSetTimeouts USB_SetTimeouts;
typedef FT_STATUS (WINAPI *PtrToGetQueueStatus)(FT_HANDLE, LPDWORD);
PtrToGetQueueStatus USB_GetQueueStatus;
typedef FT_STATUS (WINAPI *PtrToGetStatus)(FT_HANDLE, LPDWORD, LPDWORD, LPDWORD);
PtrToGetStatus USB_GetStatus;

/* Constructors */
USBCon::USBCon(char *descriptor) {
    status = USB_OK;
    linkDLL();
    if (status == USB_OK) {
        if (USB_OpenEx(descriptor, FT_OPEN_BY_DESCRIPTION, &thisFtHandle) != FT_OK) {
            updateStatus(USB_COULD_NOT_OPEN);
        } else if (USB_SetTimeouts(thisFtHandle, TX_TIMEOUT , RX_TIMEOUT) != FT_OK) {
            updateStatus(USB_COULD_NOT_SET_TIMEOUTS);
        }
    }
}

/* Utilities */
USB_STATUS USBCon::getStatus() {
    return status;
}

USB_STATUS USBCon::setTimeouts(ULONG txTimeout, ULONG rxTimeout) {
    if (USB_SetTimeouts(thisFtHandle, txTimeout , rxTimeout) != FT_OK) {
        updateStatus(USB_COULD_NOT_SET_TIMEOUTS);
    }
}

```



```
    return status;
}

USB_STATUS USBCon::purgeRxQueue() {
    if (USB_Purge(thisFtHandle, FT_PURGE_RX) != FT_OK) {
        updateStatus(USB_COULD_NOT_PURGE_RX);
    }
    return status;
}

USB_STATUS USBCon::getData(LPVOID lpBuffer, DWORD dwBytesToRead, LPDWORD lpdwBytesReturned)
{
    if (USB_Read(thisFtHandle, lpBuffer, dwBytesToRead, lpdwBytesReturned) != FT_OK) {
        updateStatus(USB_COULD_NOT_READ);
    }
    return status;
}

USB_STATUS USBCon::purgeTxQueue() {
    if (USB_Purge(thisFtHandle, FT_PURGE_TX) != FT_OK) {
        updateStatus(USB_COULD_NOT_PURGE_TX);
    }
    return status;
}

USB_STATUS USBCon::sendData(LPVOID lpBuffer, DWORD dwBytesToWrite, LPDWORD
lpdwBytesWritten) {
    if (USB_Write(thisFtHandle, lpBuffer, dwBytesToWrite, lpdwBytesWritten) != FT_OK) {
        updateStatus(USB_COULD_NOT_WRITE);
    }
    return status;
}

USB_STATUS USBCon::getDetails(LPDWORD lpdwAmountInRxQueue, LPDWORD lpdwAmountInTxQueue,
LPDWORD lpdwEventStatus) {
    if (USB_GetStatus(thisFtHandle, lpdwAmountInRxQueue, lpdwAmountInTxQueue,
lpdwEventStatus) != FT_OK) {
        updateStatus(USB_COULD_NOT_WRITE);
    }
    return status;
}

USB_STATUS USBCon::listDevices(PVOID pvArg1, PVOID pvArg2, DWORD dwFlags) {
    if (USB_ListDevices(pvArg1, pvArg2, dwFlags) != FT_OK) {
        updateStatus(USB_COULD_NOT_WRITE);
    }
    return status;
}
}
```

```
/* Private function members */
void USBCon::linkDLL() {
    thisHModule = LoadLibrary("Ftd2xx.dll");
    if (thisHModule == NULL) updateStatus(USB_DLL_ERROR);
    USB_Write = (PtrToWrite)GetProcAddress(thisHModule, "FT_Write");
    if (USB_Write == NULL) updateStatus(USB_DLL_ERROR);
    USB_Read = (PtrToRead)GetProcAddress(thisHModule, "FT_Read");
    if (USB_Read == NULL) updateStatus(USB_DLL_ERROR);
    USB_Open = (PtrToOpen)GetProcAddress(thisHModule, "FT_Open");
    if (USB_Open == NULL) updateStatus(USB_DLL_ERROR);
    USB_OpenEx = (PtrToOpenEx)GetProcAddress(thisHModule, "FT_OpenEx");
    if (USB_OpenEx == NULL) updateStatus(USB_DLL_ERROR);
    USB_ListDevices = (PtrToListDevices)GetProcAddress(thisHModule, "FT_ListDevices");
    if(USB_ListDevices == NULL) updateStatus(USB_DLL_ERROR);
    USB_Close = (PtrToClose)GetProcAddress(thisHModule, "FT_Close");
    if (USB_Close == NULL) updateStatus(USB_DLL_ERROR);
    USB_ResetDevice = (PtrToResetDevice)GetProcAddress(thisHModule, "FT_ResetDevice");
    if (USB_ResetDevice == NULL) updateStatus(USB_DLL_ERROR);
    USB_Purge = (PtrToPurge)GetProcAddress(thisHModule, "FT_Purge");
    if (USB_Purge == NULL) updateStatus(USB_DLL_ERROR);
    USB_SetTimeouts = (PtrToSetTimeouts)GetProcAddress(thisHModule, "FT_SetTimeouts");
    if (USB_SetTimeouts == NULL) updateStatus(USB_DLL_ERROR);
    USB_GetQueueStatus = (PtrToGetQueueStatus)GetProcAddress(thisHModule,
                                                              "FT_GetQueueStatus");
    if (USB_GetQueueStatus == NULL) updateStatus(USB_DLL_ERROR);
    USB_GetStatus = (PtrToGetStatus)GetProcAddress(thisHModule, "FT_GetStatus");
    if (USB_GetStatus == NULL) updateStatus(USB_DLL_ERROR);
}

void USBCon::updateStatus(USB_STATUS newStatus) {
    if (status == USB_OK) status = newStatus;
}

/* Destructor */
USBCon::~USBCon() {
    if (USB_Close(thisFtHandle) != FT_OK) {
        updateStatus(USB_COULD_NOT_CLOSE);
    }
    FreeLibrary(thisHModule);
}
```

APPENDIX D: SIMULATION SCRIPTS

D.1. Simplorer model of the DPQC controller

```

MODELDEF DPQCCNTRL {
    PORT REAL IN : loopGain = 1;
    PORT REAL IN : loopGainD = 1;
    PORT REAL IN : loopGainQ = 1;
    PORT REAL IN : sampleT = 50e-6;
    PORT REAL IN : switchingF = 5000;
    PORT REAL IN : prm_Supply_Lser_i = 400u;
    PORT REAL IN : prm_Op_APF_clpfA = 0.00001;
    PORT REAL IN : prm_Op_APF_clpfR = 0.00001;
    PORT REAL IN : filterStartA = 0;
    PORT REAL IN : filterStartR = 0;
    PORT REAL IN : shapingFilterP = 0.45;
    PORT REAL IN : shapingFilterStart = 0;
    PORT REAL IN : tDead = 40u;

    PORT REAL IN : Vsa;
    PORT REAL IN : Vsb;
    PORT REAL IN : Vsc;
    PORT REAL IN : Vdc;
    PORT REAL IN : Ila;
    PORT REAL IN : Ilb;
    PORT REAL IN : Ilc;
    PORT REAL IN : Ifa;
    PORT REAL IN : Ifb;
    PORT REAL IN : Ifc;

    PORT REAL OUT : topAOn = topAOnInt;
    PORT REAL OUT : botAOn = botAOnInt;
    PORT REAL OUT : topBOn = topBOnInt;
    PORT REAL OUT : botBOn = botBOnInt;
    PORT REAL OUT : topCOn = topCOnInt;
    PORT REAL OUT : botCOn = botCOnInt;

    /*****
    /* DSP model */
    /*****

    /* S&H's to model discrete controller */
    INTERN SAH mVsa (INPUT:=Vsa, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
    INTERN SAH mVsb (INPUT:=Vsb, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
    INTERN SAH mVsc (INPUT:=Vsc, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
    INTERN SAH mVdc (INPUT:=Vdc, NH:=1, NO:=0, Y0:=0, TS:=sampleT);

```



```

INTERN SAH mIla (INPUT:=Ila, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
INTERN SAH mIlb (INPUT:=Ilb, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
INTERN SAH mIlc (INPUT:=Ilc, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
INTERN SAH mIfa (INPUT:=Ifa, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
INTERN SAH mIfb (INPUT:=Ifb, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
INTERN SAH mIfc (INPUT:=Ifc, NH:=1, NO:=0, Y0:=0, TS:=sampleT);

/* Delays */
INTERN SAH mIl_a_z (INPUT:=mIl_a, NH:=1, NO:=0, Y0:=0, TS:=sampleT);
INTERN SAH mIl_r_z (INPUT:=mIl_r, NH:=1, NO:=0, Y0:=0, TS:=sampleT);

/* Initial conditions */
INTERN EQU {
    mIl_a := filterStartA;
    mIl_r := filterStartR;
    r_Vd := 0;
    r_Vq := 0;
} DST: SIM(Type := SFML, Sequ := INIT);

INTERN EQU {
    /* Measurements */
    mIld := mIla - 0.5*(mIlb + mIlc);
    mIlq := 0.8660*(mIlb - mIlc);

    mIfd := mIfa - 0.5*(mIfb + mIfc);
    mIfq := 0.8660*(mIfb - mIfc);

    imVdc := 1.0/mVdc;

    mVsd := mVsa - 0.5*(mVsb + mVsc);
    mVsq := 0.8660*(mVsb - mVsc);
    ImVs := 1.0/sqrt(mVsd^2 + mVsq^2);

    coswts := mVsd*ImVs;
    sinwts := mVsq*ImVs;

    /* Basic reference generation */
    mIl_aA := mIld*coswts + mIlq*sinwts;
    mIl_rA := -mIld*sinwts + mIlq*coswts;

    mIl_a := mIl_a_z + prm_Op_APF_clpfA*(mIl_aA - mIl_a_z);
    mIl_r := mIl_r_z + prm_Op_APF_clpfR*(mIl_rA - mIl_r_z);

    Icomp_a := loopGain*loopGainD*(mIl_aA - mIl_a);
    Icomp_r := loopGain*loopGainQ*(mIl_rA - mIl_r);

    rIfd := Icomp_a*coswts - Icomp_r*sinwts;
    rIfq := Icomp_a*sinwts + Icomp_r*coswts;

```

```

/* Prediction */
rIfdp := rIfd*0.9921147013144779 - rIfq*0.1253332335643043;
rIfqp := rIfd*0.1253332335643043 + rIfq*0.9921147013144779;

mIfdp := mIfd*0.9980267284282716 - mIfq*0.0627905195293133;
mIfqp := mIfd*0.0627905195293133 + mIfq*0.9980267284282716;

r_VdU := switchingF*prm_Supply_Lser_i*(rIfdp - mIfdp) +
(mVsd*0.9955619646030800 - mVsq*0.0941083133185143);
r_VqU := switchingF*prm_Supply_Lser_i*(rIfqp - mIfqp) +
(mVsd*0.0941083133185143 + mVsq*0.9955619646030800);

/* Filtering */
/* NOTE: Not implemented in real controller. */
r_Vd := r_Vd_z + shapingFilterP*(r_VdU - r_Vd_z);
r_Vq := r_Vq_z + shapingFilterP*(r_VqU - r_Vq_z);
} DST: SIM(Type := SSGM, Sequ := POST );

/* Calculation delays */
INTERN GZ r_Vd_z ( D:=1,P[0]:=1,Q[1]:=1,INPUT:=r_Vd,N:=2,TS:=50u,Q0[1]:=0);
INTERN GZ r_Vq_z ( D:=1,P[0]:=1,Q[1]:=1,INPUT:=r_Vq,N:=2,TS:=50u,Q0[1]:=0);

/* Sace vector PWM */
INTERN EQU {
/* sector 1 */
IF ((r_Vq_z >= 0) AND (r_Vq_z <= (1.732*abs(r_Vd_z))) AND (r_Vd_z >= 0)) {
D2 := imVdc*(r_Vd_z + r_Vq_z*((-0.5)/0.86602540378));
D1 := imVdc*(r_Vq_z*(( 1.0)/0.86602540378));
iD := (1.0)/(D1+D2);
IF (iD < 1.0) {
D2 := D2*iD;
D1 := D1*iD;
D0 := 0.0;
} ELSE {
D0 := 0.5*(1.0-D1-D2);
}
s_Dc := D0;
s_Db := s_Dc + D1;
s_Da := s_Db + D2;
}

/* sector 2 */
IF ((r_Vq_z >= 0) AND (r_Vq_z > (1.732*abs(r_Vd_z)))) {
D2 := imVdc*(r_Vq_z*(( 0.5)/0.86602540378) - r_Vd_z);
D1 := imVdc*(r_Vq_z*(( 0.5)/0.86602540378) + r_Vd_z);
iD := (1.0)/(D1+D2);
IF (iD < 1.0) {

```



```

        D2 := D2*iD;
        D1 := D1*iD;
        D0 := 0.0;
    } ELSE {
        D0 := 0.5*(1.0-D1-D2);
    }
    s_Dc := D0;
    s_Da := s_Dc + D1;
    s_Db := s_Da + D2;
}

/* sector 3 */
IF ((r_Vq_z >= 0) AND (r_Vq_z <= (1.732*abs(r_Vd_z))) AND (r_Vd_z < 0)) {
    D2 := imVdc*(r_Vq_z*(( 1.0)/0.866025403780));
    D1 := imVdc*(r_Vq_z*((-0.5)/0.86602540378) - r_Vd_z);
    iD := (1.0)/(D1+D2);
    IF (iD < 1.0) {
        D2 := D2*iD;
        D1 := D1*iD;
        D0 := 0.0;
    } ELSE {
        D0 := 0.5*(1.0-D1-D2);
    }
    s_Da := D0;
    s_Dc := s_Da + D1;
    s_Db := s_Dc + D2;
}

/* sector 4 */
IF ((r_Vq_z <= 0) AND (r_Vq_z > (-1.732*abs(r_Vd_z))) AND (r_Vd_z < 0)) {
    D2 := imVdc*(r_Vq_z*((-1.0)/0.86602540378));
    D1 := imVdc*(r_Vq_z*(( 0.5)/0.86602540378) - r_Vd_z);
    iD := (1.0)/(D1+D2);
    IF (iD < 1.0) {
        D2 := D2*iD;
        D1 := D1*iD;
        D0 := 0.0;
    } ELSE {
        D0 := 0.5*(1.0-D1-D2);
    }
    s_Da := D0;
    s_Db := s_Da + D1;
    s_Dc := s_Db + D2;
}

/* sector 5 */
IF ((r_Vq_z < 0) AND (r_Vq_z <= (-1.732*abs(r_Vd_z)))) {
    D2 := imVdc*(r_Vq_z*((-0.5)/0.86602540378) - r_Vd_z);

```

```

D1 := imVdc*(r_Vq_z*((-0.5)/0.86602540378) + r_Vd_z);
iD := (1.0)/(D1+D2);
IF (iD < 1.0) {
    D2 := D2*iD;
    D1 := D1*iD;
    D0 := 0.0;
} ELSE {
    D0 := 0.5*(1.0-D1-D2);
}
s_Db := D0;
s_Da := s_Db + D1;
s_Dc := s_Da + D2;
}

/* sector 6 */
IF ((r_Vq_z < 0) AND (r_Vq_z > (-1.732*abs(r_Vd_z))) AND (r_Vd_z > 0)) {
    D2 := imVdc*(r_Vd_z + r_Vq_z*(( 0.5)/0.86602540378));
    D1 := imVdc*(r_Vq_z*((-1.0)/0.86602540378));
    iD := (1.0)/(D1+D2);
    IF (iD < 1.0) {
        D2 := D2*iD;
        D1 := D1*iD;
        D0 := 0.0;
    } ELSE {
        D0 := 0.5*(1.0-D1-D2);
    }
    s_Db := D0;
    s_Dc := s_Db + D1;
    s_Da := s_Dc + D2;
}

/* Dead time compensation */
De := tDead*switchingF;
rIfa := 2/3*rIfd;
rIfb := 2/3*(-0.5*rIfd + 0.886*rIfq);
rIfc := 2/3*(-0.5*rIfd - 0.886*rIfq);

IF ((s_Da > De) AND (s_Db > De) AND (s_Dc > De)) {
/* Mode 1*/
    IF (rIfa >= 0) { s_Da := s_Da + De }
    ELSE          { s_Da := s_Da - De }
    IF (rIfb >= 0) { s_Db := s_Db + De }
    ELSE          { s_Db := s_Db - De }
    IF (rIfc >= 0) { s_Dc := s_Dc + De }
    ELSE          { s_Dc := s_Dc - De }
/* Mode 2 */
} ELSE IF ((s_Da <= De) AND (rIfa >= 0)) {
    IF (rIfb >= 0) { s_Db := s_Db - (s_Da - De) }
}

```

```

ELSE                                { s_Db := s_Db - (s_Da + De) }
IF (rIfc >= 0) { s_Dc := s_Dc - (s_Da - De) }
ELSE                                { s_Dc := s_Dc - (s_Da + De) }
} ELSE IF ((s_Da <= De) AND (rIfa < 0)) {
  IF (rIfb >= 0) { s_Db := s_Db - (s_Da - 3*De) }
  ELSE          { s_Db := s_Db - (s_Da - De) }
  IF (rIfc >= 0) { s_Dc := s_Dc - (s_Da - 3*De) }
  ELSE          { s_Dc := s_Dc - (s_Da - De) }
} ELSE IF ((s_Db <= De) AND (rIfb >= 0)) {
  IF (rIfa >= 0) { s_Da := s_Da - (s_Db - De) }
  ELSE          { s_Da := s_Da - (s_Db + De) }
  IF (rIfc >= 0) { s_Dc := s_Dc - (s_Db - De) }
  ELSE          { s_Dc := s_Dc - (s_Db + De) }
} ELSE IF ((s_Db <= De) AND (rIfb < 0)) {
  IF (rIfa >= 0) { s_Da := s_Da - (s_Db - 3*De) }
  ELSE          { s_Da := s_Da - (s_Db - De) }
  IF (rIfc >= 0) { s_Dc := s_Dc - (s_Db - 3*De) }
  ELSE          { s_Dc := s_Dc - (s_Db - De) }
} ELSE IF ((s_Dc <= De) AND (rIfc >= 0)) {
  IF (rIfa >= 0) { s_Da := s_Da - (s_Dc - De) }
  ELSE          { s_Da := s_Da - (s_Dc + De) }
  IF (rIfb >= 0) { s_Db := s_Db - (s_Dc - De) }
  ELSE          { s_Db := s_Db - (s_Dc + De) }
} ELSE IF ((s_Dc <= De) AND (rIfc < 0)) {
  IF (rIfa >= 0) { s_Da := s_Da - (s_Dc - 3*De) }
  ELSE          { s_Da := s_Da - (s_Dc - De) }
  IF (rIfb >= 0) { s_Db := s_Db - (s_Dc - 3*De) }
  ELSE          { s_Db := s_Db - (s_Dc - De) }
}
}
} DST: SIM(Type := SSGM, Sequ:=POST );

/*****/
/* FPGA SIMULATION */
/*****/

/* Triangular wave for FPGA simulation */
INTERN TRIANG TRIANG1 (
  FREQ:=switchingF, TPERIO:=200u, AMPL:=0.5, PHASE:=0, PERIO:=1, OFF:=0.5, TDELAY:=0 );
INTERN EQU {
  IF (TRIANG1.VAL > s_Da) {
    topAOnInt := 0;
    botAOnInt := 1;
  } ELSE {
    topAOnInt := 1;
    botAOnInt := 0;
  }
  IF (TRIANG1.VAL > s_Db) {
    topBOnInt := 0;

```

```

        botBOnInt := 1;
    } ELSE {
        topBOnInt := 1;
        botBOnInt := 0;
    }
    IF (TRIANG1.VAL > s_Dc) {
        topCOnInt := 0;
        botCOnInt := 1;
    } ELSE {
        topCOnInt := 1;
        botCOnInt := 0;
    }
} DST: SIM(Type := SSGM, Sequ := POST );
}

```

D.2. MATLAB simulation of jaw crusher compensation

```

% GENERATE Generate the compensation results for the measurements in this
% directory
% This Matlab script calls the local calculation function (dataAdd.m) for
% each search set of compensation options. The calculated data is stored in a
% local data file (data.dat). The results are plotted.

```

```

% Author: Leon de Wit
% University of Stellenbosch
% 13/09/2005

```



```

dataGrid = [];
save dataGrid.mat dataGrid;

```

```

warning off;

```

```

dataAdd(0,0);
dataAdd(0,1);
dataAdd(1,1);

```

```

dataAdd(0,0.3);
dataAdd(0,0.5);
dataAdd(0,0.7);

```

```

dataAdd(0.3,0.3);
dataAdd(0.5,0.5);
dataAdd(0.7,0.7);

```

```

dataAdd(0.3,1);
dataAdd(0.5,1);
dataAdd(0.7,1);

```

```

warning on;

load dataGrid.mat;
plot3(dataGrid(:, 1), dataGrid(:, 2)/1e3, dataGrid(:, 3)/1e3, 'ro'); hold on;
[XI,YI] = meshgrid(linspace(0,1.1,100),linspace(0,80,100));
ZI = griddata(dataGrid(:, 1),dataGrid(:, 2)/1e3,dataGrid(:, 3)/1e3,XI,YI,'cubic');
surf(XI,YI,ZI);
xlabel('Pst');
ylabel('S [kVA]');
zlabel('E [kJ]');
grid on;
hold off;

figure(2);
for i = 1 : 5 : 100
    plot(YI(:,i),ZI(:,i)); hold on;
end;
grid on;
xlabel('S [kVA]');
ylabel('E [kJ]');
hold off;

% DATAADD Add compensation data to the data file
% For use with generate.m.

% Author: Leon de Wit
% University of Stellenbosch
% 13/09/2005
function dataAdd(loopGainD, loopGainQ)
    loopGain = 1;
    filterStartD = 125;
    filterStartQ = -200;

    k = 10e-6;
    R = 0.037;
    L = R/2/pi/50;

    [Vab, tm] = loadtek('TEK00008.ISF'); Vab = Vab - sum(Vab)/length(Vab);
    Vbc = -loadtek('TEK00009.ISF'); Vbc = Vbc - sum(Vbc)/length(Vbc);
    Vca = -(Vab + Vbc);
    Ia = loadtek('TEK00011.ISF'); Ia = Ia - sum(Ia)/length(Ia);
    Ic = -loadtek('TEK00010.ISF'); Ic = Ic - sum(Ic)/length(Ic);
    Ib = -(Ia + Ic);
    tm = tm + 4.9990e+000;

    t = 0 : 50e-6 : 10 - 2e-3;
    Vab = interp1(tm, Vab, t);
    Vbc = interp1(tm, Vbc, t);

```



```

Vca = interp1(tm, Vca, t);
mIa = interp1(tm, Ia, t);
mIb = interp1(tm, Ib, t);
mIc = interp1(tm, Ic, t);

mIld = mIa - 0.5.*(mIb + mIc);
mIlq = 0.8660.*(mIb - mIc);

mVsa = (450*sin(2*pi*49.915*t + 004.2/180*pi))';
mVsb = (450*sin(2*pi*49.915*t - 115.8/180*pi))';
mVsc = (450*sin(2*pi*49.915*t + 124.2/180*pi))';

mVsd = mVsa - 0.5.*(mVsb + mVsc);
mVsq = 0.8660.*(mVsb - mVsc);
ImVs = (1.0./sqrt(mVsd.^2 + mVsq.^2));

coswts = mVsd.*ImVs;
sinwts = mVsq.*ImVs;

coswts = min(1.1, coswts); coswts = max(-1.1, coswts);
sinwts = min(1.1, sinwts); sinwts = max(-1.1, sinwts);

% Reference generation
mIl_aA = mIld'.*coswts + mIlq'.*sinwts;
mIl_rA = -mIld'.*sinwts + mIlq'.*coswts;

B = [k];
A = [1 k-1];
mIl_a = filter(B, A, mIl_aA, [-filterStartD/(k-1)]);
mIl_r = filter(B, A, mIl_rA, [-filterStartQ/(k-1)]);

Icomp_a = loopGain*loopGainD*(mIl_aA - mIl_a);
Icomp_r = loopGain*loopGainQ*(mIl_rA - mIl_r);

rIfd = Icomp_a.*coswts - Icomp_r.*sinwts;
rIfq = Icomp_a.*sinwts + Icomp_r.*coswts;

rIfa = 2.0/3.0*rIfd;

rIla = mIa - rIfa';
v = mVsa' - R*rIla - L*[0 diff(rIla)]/50e-6;

% Pst
[Fl Pst] = flickerMeter(interp1(t, v, [0 : 200e-6: t(end)]), 50, 5000, 0.4e5); Pst

% S
S = 3/2*450*max(max(rIfa), -min(rIfa))

```

```
% E
p = v.*rIfa';
e = filter(ones(1,20000),1,p/20000);
E = max(e) - min(e)

% Save data
load dataGrid.mat;
dataGrid = [dataGrid; [Pst S E]];
save 'dataGrid.mat' dataGrid;
```



APPENDIX E: FLICKER GENERATOR

E.1. Controller VHDL code

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity flickerCntl is
    port ( clock, not_reset : in std_logic;           -- GCLK, GCLRn
          thr1, thr2, thr3 : out std_logic;          -- CH1, CH2, CH3
          not_buzzer : out std_logic;                -- CH4 reserved for future use
          stopGo : in std_logic;                     -- CH5
          dummy_in : in std_logic_vector(5 downto 0); -- S5..S0 reserved for f. use
          dummy_out : out std_logic_vector(10 downto 0)); -- CH15..6 reserved for f. use
end entity flickerCntl;

architecture al of flickerCntl is
    -- user parameters
    constant LONGCOUNT : integer := 17540;
    constant SHORTCOUNT : integer := 2460;
    -- end of user parameters
    type stateType is (waiting, firing);
    signal state : stateType := waiting;
    signal counter : integer range 0 to LONGCOUNT := 0;
    signal start : std_logic := '0';
begin
-----
process (clock, not_reset) is
begin
    if (not_reset = '0') then
        state <= waiting;
        start <= '0';
        counter <= 0;
    elsif rising_edge(clock) then
        case state is
            when waiting =>
                if (counter = LONGCOUNT) then
                    state <= firing;
                    start <= '1';
                    counter <= 0;
                else counter <= counter + 1;
                end if;
            when firing =>
                if (counter = SHORTCOUNT) then

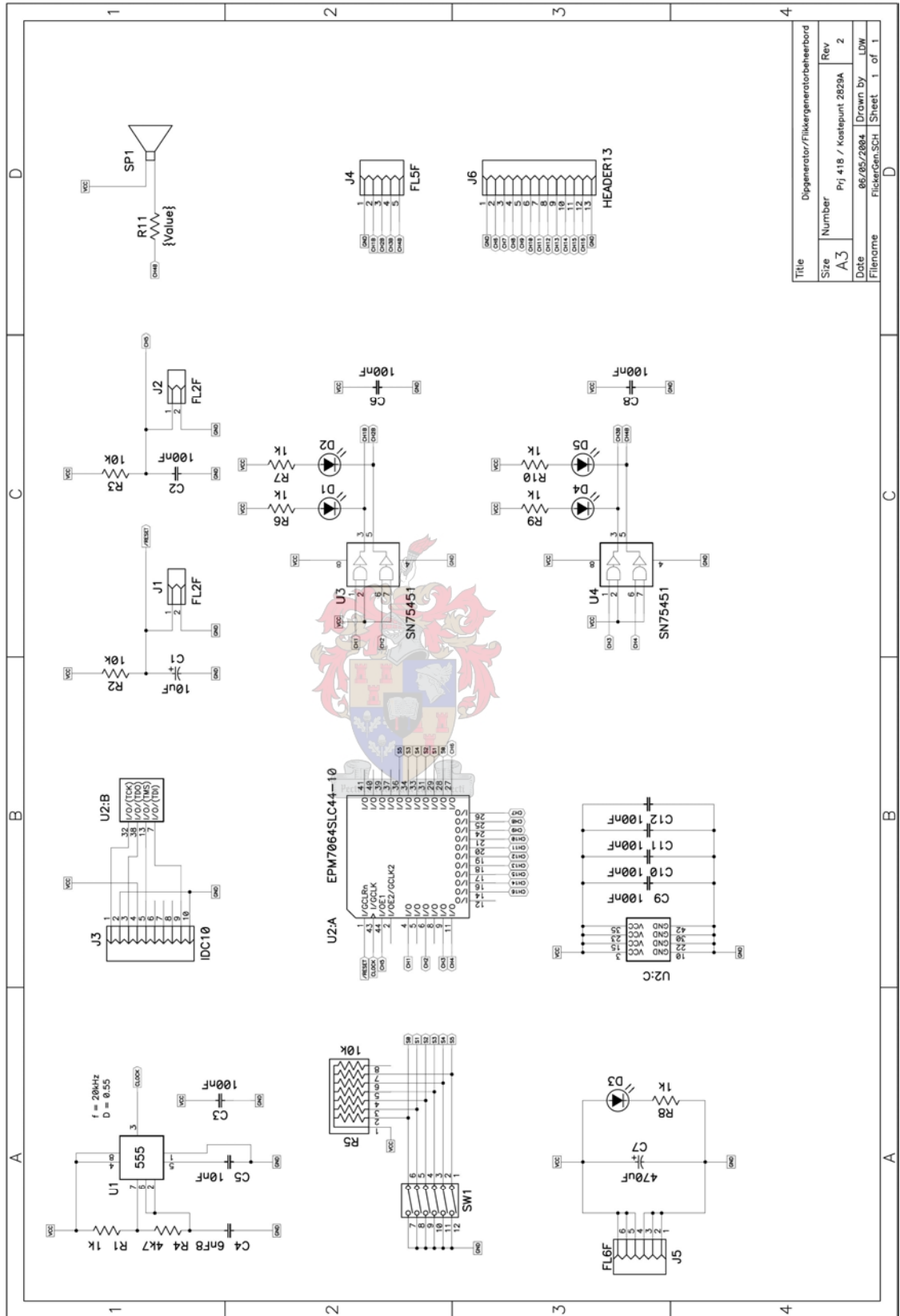
```



```
        state <= waiting;
        start <= '0';
        counter <= 0;
    else
        counter <= counter + 1;
        start <= not start;
    end if;
end case;
end if;
end process;
-----
thr1 <= start and stopGo;
thr2 <= start and stopGo;
thr3 <= start and stopGo;
not_buzzer <= '1';
dummy_out(10 downto 0) <= "0000000000";
end architecture a1;
```



E.2. Controller schematic



APPENDIX F: DPQC CODE

F.1. Extract from modec.c

```

static inline void CalcAPFref() {
    double rIfd, rIfq;
    double d_Id, d_Iq;
    double pComp, pCompMaxCharge, pCompMaxDischarge;
    double mIldd, mIlqq, rIfdd, rIfqq;
    double mIfdp, mIfqp;

    /* Translate currents to synchronous reference frame */
    mIldd = mIld*coswtf + mIlq*sinwtf;
    mIlqq = -mIld*sinwtf + mIlq*coswtf;

    /* Filter the active and reactive dc components */
    Il_a += prm_Op_APF_clpf*(mIldd-Il_a);
    Il_r += prm_Op_APF_clpf*(mIlqq-Il_r);

    /* Bus regulation */
    rIf += 0.0001*((-3.0)/(4.0))*mVdc*imVf*2.0*cID*cDV*(r_Vdc-mVdc) - rIf;

    /* Calculate compensation
    #if COMPENSATE_REACTIVE
    rIfdd = 0;
    rIfqq = 1.3*cAPF*(mIlqq - Il_r);
    #else
    rIfdd = 1.3*cAPF*(mIldd - Il_a);
    rIfqq = 1.3*cAPF*(mIlqq - Il_r);
    #endif

    /* Translate to rotating reference frame, adding bus regulation
    rIfd = rIfdd*coswtf - rIfqq*sinwtf + coswtf*rIf;
    rIfq = rIfdd*sinwtf + rIfqq*coswtf + sinwtf*rIf;

    /* Battery protection */
    /* Power to generate compensation current & bus regulation */
    pComp = ((2.0)/(3.0))*(mVfd*rIfd + mVfq*rIfq);
    /* Battery maximum discharge power. Limited to 60kW or 300 A DC, whichever is smaller. */
    pCompMaxDischarge = min(( cID*cVD*60.0e3), ( mVdc*cID*300.0));
    /* Battery maximum discharge power. Limited to 60kW or 300 A DC*/
    pCompMaxCharge = max((-1.0*cID*cVD*60.0e3), (-1.0*mVdc*cID*300.0));
    if (pComp > pCompMaxDischarge) {
        rIfd = (pCompMaxDischarge/pComp)*rIfd;
        rIfq = (pCompMaxDischarge/pComp)*rIfq;
    } else if (pComp < pCompMaxCharge) {

```

```

    rIfd = (pCompMaxCharge/pComp)*rIfd;
    rIfq = (pCompMaxCharge/pComp)*rIfq;
}

/* Sinusoidal prediction */
rIfdp = rIfd*0.9921147013144779 - rIfq*0.1253332335643043;
rIfqp = rIfd*0.1253332335643043 + rIfq*0.9921147013144779;

mIfdp = mIfd*0.9980267284282716 - mIfq*0.0627905195293133;
mIfqp = mIfd*0.0627905195293133 + mIfq*0.9980267284282716;

/* Dead beat current control
r_Vd = 5000.0*prm_Supply_Lser_i*(rIfdp - mIfdp) + (mVfd*0.9955619646030800 -
                                                    mVfq*0.0941083133185143);
r_Vq = 5000.0*prm_Supply_Lser_i*(rIfqp - mIfqp) + (mVfd*0.0941083133185143 +
                                                    mVfq*0.9955619646030800);
}

```

F.2. Extract from space.c

```

/* Dead time compensation */
De = 0.025; /* tDead*fs = 5e-6*5000 */
rIfa = 2.0/3.0*rIfdp;
rIfb = 2.0/3.0*(-0.5*rIfdp + 0.886*rIfqp);
rIfc = 2.0/3.0*(-0.5*rIfdp - 0.886*rIfqp);

if ((s_Da > De) && (s_Db > De) && (s_Dc > De)) {
/* Mode 1 */
    if (rIfa >= 0.0) { s_Da = s_Da + De; }
    else { s_Da = s_Da - De; }
    if (rIfb >= 0.0) { s_Db = s_Db + De; }
    else { s_Db = s_Db - De; }
    if (rIfc >= 0.0) { s_Dc = s_Dc + De; }
    else { s_Dc = s_Dc - De; }
/* Mode 2 */
} else if ((s_Da <= De) && (rIfa >= 0.0)) {
    if (rIfb >= 0.0) { s_Db = s_Db - (s_Da - De); }
    else { s_Db = s_Db - (s_Da + De); }
    if (rIfc >= 0.0) { s_Dc = s_Dc - (s_Da - De); }
    else { s_Dc = s_Dc - (s_Da + De); }
} else if ((s_Da <= De) && (rIfa < 0.0)) {
    if (rIfb >= 0.0) { s_Db = s_Db - (s_Da - 3.0*De); }
    else { s_Db = s_Db - (s_Da - De); }
    if (rIfc >= 0.0) { s_Dc = s_Dc - (s_Da - 3.0*De); }
    else { s_Dc = s_Dc - (s_Da - De); }
} else if ((s_Db <= De) && (rIfb >= 0.0)) {
    if (rIfa >= 0.0) { s_Da = s_Da - (s_Db - De); }
    else { s_Da = s_Da - (s_Db + De); }
}

```

```
    if (rIfc >= 0.0) { s_Dc = s_Dc - (s_Db - De); }
    else
        { s_Dc = s_Dc - (s_Db + De); }
} else if ((s_Db <= De) && (rIfb < 0.0)) {
    if (rIfa >= 0.0) { s_Da = s_Da - (s_Db - 3.0*De); }
    else
        { s_Da = s_Da - (s_Db - De); }
    if (rIfc >= 0.0) { s_Dc = s_Dc - (s_Db - 3.0*De); }
    else
        { s_Dc = s_Dc - (s_Db - De); }
} else if ((s_Dc <= De) && (rIfc >= 0.0)) {
    if (rIfa >= 0.0) { s_Da = s_Da - (s_Dc - De); }
    else
        { s_Da = s_Da - (s_Dc + De); }
    if (rIfb >= 0.0) { s_Db = s_Db - (s_Dc - De); }
    else
        { s_Db = s_Db - (s_Dc + De); }
} else if ((s_Dc <= De) && (rIfc < 0.0)) {
    if (rIfa >= 0.0) { s_Da = s_Da - (s_Dc - 3.0*De); }
    else
        { s_Da = s_Da - (s_Dc - De); }
    if (rIfb >= 0.0) { s_Db = s_Db - (s_Dc - 3.0*De); }
    else
        { s_Db = s_Db - (s_Dc - De); }
}
```

