To the Graduate Council:

I am submitting herewith a dissertation written by Sunil M. Chhaya, entitled "Expert System Aided Automated Design of AC Drive System." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Electrical Engineering.

B. K. Bose, Major Professor

We have read this dissertation and recommend its acceptance:

W. L. Green

F. W. Symonds

B. R. Upadhyaya

Accepted for the Council:

Associate Vice Chancellor and
Dean of the Graduate School

EXPERT SYSTEM AIDED AUTOMATED DESIGN OF AC DRIVE SYSTEM

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Sunil M. Chhaya

August 1995

DEDICATION

To my wife, Sudha, for her love and care

and

To my parents, for their support and guidance

ACKNOWLEDGEMENTS

I would like to thank Dr. B. K. Bose, my advisor, for introducing me to the emerging technology of applied artificial intelligence to power electronics and AC motor drives, for providing expert guidance and motivation through my Ph. D. work. I would also like to express deep appreciation towards the members of my Ph. D. committee, Dr. F. W. Symonds, Dr. B. R. Upadhyaya and Dr. W. L. Green, for their time, interest and suggestions during my research.

I thank the support staff at the EPRI Power Electronics Applications Center, for their support with testing equipment and my colleagues at the Power Electronics Lab, Dr.s Debaprasad Kastha and Gilberto Sousa, as well as Mr. Marcelo Simóes for their help and their insight on various aspects of my work. Thanks are also due to the Tennessee Valley Authority for providing the financial support during my Ph. D. studies.

Lastly, but most importantly, I thank my wife, Sudha, for her love, cheerful support, and motivation, that have helped me succeed.

Sunil M. Chhaya

ABSTRACT

Traditionally, AC drive systems have been designed by the expert engineer using paper, pencil and calculator, and then tested through laboratory prototype. This procedure is expensive and time-consuming. Besides, the design is not necessarily optimum. An expert system aided automated design of AC drive systems is presented here. The comprehensive design package for AC drive systems first configures the AC drive system based on the user's application specifications, and designs the nameplate specifications of the AC drive system. Next, it designs and optimizes the power converter system and then designs and fine-tunes the controllers for the AC drive system. Once the complete design is over, it downloads the optimized DSP controller code to a target experimental drive system, for its validation.

The expert system for the AC drive configuration selection and motor design was developed embedding the expert's know-how on matching the application specs with the characteristics of AC drive configurations available. The voltage-fed type of configuration was selected for the 1-200 kW power range of the AC drive system. A wide range of application classes, such as pumps, fans, winders, crushers, conveyors, machine tool drives, robotic arms, elevators, electric vehicles and traction, were embedded for the design and configuration of AC drive systems, and the expert system knowledge base was made flexible enough to tackle even the custom-designed application specified by the user. The design criteria for the motor nameplate specs were incorporated in the expert system in terms of the rule base. Intelligent defaults were implemented to meet any unresolved or conflicting choices. The user interface of the expert system was in terms of the natural language dialog and menu-driven consultation.

The expert system based automated design of the power converter system involved development of the rule-base for designing the ratings of the power devices, heat sink, snubbers and filter capacitors. Rules were also included to permit expert system's access to databases and graphics files. The databases were developed to store the data sheets of power semiconductor devices, such as the diodes and transistors (BJT/IGBTs), which would store their ratings and switching times. Graphical data were implemented for the devices using the best-fitting curves obtained through TableCurve software.

For the power converter optimization, the models of the power converter and machine, as well as the half-bridge for snubber performance optimization were developed in SIMNON program. In order for the expert system to communicate with SIMNON simulation, C language based interface routines achieving the data and control format conversion were developed. The expert system rule-base was developed to control the simulation of the power converter system in a manner that would obtain optimum values of snubber components. The results showed successful applications of the expert system for power converter optimization.

The expert system was then developed to perform the initial design of the control gains of the AC drive system. This was demonstrated with the example of indirect vector controlled induction motor drive with synchronous current control. The next task was to achieve a hybrid simulation software for the AC drive system. The power converter and machine simulation model was implemented in SIMNON, whereas the controllers were implemented in DSP 320C25 assembly and C language, being executed in the simulator environment. Then, the communication protocol was developed between the expert system and DSP simulator, in terms of C-language based data format and control conversion routines. Next, expert system rule-base was developed which would control the simulation of the AC drive controllers so as to optimize the controllers for the AC drive system. The user interface in this case involved graphical display of intermediate results and natural language dialogues for status information. The hybrid simulation software was developed flexibly, so as to be usable in stand alone operation. Next, the complete software was tested for its tuning performance in all the modes of tuning. The results obtained demonstrated the expert system's ability and usefulness to perform off-line optimization of the AC drive controllers. Moreover, the optimized DSP-based controller code was now available, and was downloaded to the experimental drive system.

The next task involved the experimental validation of the expert system based control tuning of the indirect vector controlled induction motor drive system. The hardware and software for this IGBT inverter fed 5 hp induction servo drive were integrated in a DSP 320C25 based system, and the tests were performed on the AC drive, to obtain the results of the tuning. These were found to match well with the tuned gain values obtained through the expert system based control tuning.

TABLE OF CONTENTS

Chapter		Page no.
1.	INTRODUCTION	
	1.1 Overview	1
	1.2 Literature Review	2
	1.3 Outline of the Dissertation	5
2.	EXPERT SYSTEM PRINCIPLES AND SHELL-BASED DESIGN METHODOLOGY	
	2.1 Introduction	9
	2.2 Expert System Principles	11
	2.2.1 Knowledge Base	12
	2.2.2 Inference Engine	12
	2.2.3 User Interface	14
	2.2.4 User Programs	15
	2.3 Expert System Development Tools	15
	2.3.1 Features of PC Plus Shell	16
	2.4 Shell Based Expert System Development Methodology	20
	2.4.1 Knowledge Acquisition	22
	2.4.2 Knowledge Representation	22
	2.4.3 User Interface Design	25
	2.4.4 Testing	26

2.4.4 Testing 26

2.5 Summary 27

3. EXPERT SYSTEM BASED AUTOMATED SELECTION OF AC DRIVE CONFIGURATION AND AC MOTOR DESIGN

3.1 Introduction 28

3.2 AC Drive Configuration Selection Considerations 30

3.2.1 Selection Criteria for AC Motors 32

3.2.2 Power Converter Topology Selection Criteria 36

3.2.3 Selection Criteria for Suitable Control for AC Drive System 38

3.3 AC Drive System Configuration Methodology 44

3.4 AC Motor Design Considerations 45

3.4.1 Motor Power Rating 46

3.4.2 Motor Voltage Rating 48

3.4.3 Motor Current Rating 49

3.4.4 Motor Base Frequency 49

3.4.5 Motor Enclosure and Ventilation 49

3.5 Knowledge Representation for AC Drive Configuration Selection and AC Motor Design 50

3.5.1 Knowledge Base Development for AC Drive Configuration Selection 51

3.5.2 Knowledge Base Development for AC Motor Design 55

3.6 User Interface Design 57

3.7 Summary 57

4. EXPERT SYSTEM BASED AUTOMATED DESIGN OF POWER CONVERTER SYSTEM

4.1 Introduction 59

4.2 Converter System Design Considerations 60

4.2.1 Design of Diode Rectifier 62

4.2.2 DC Link Filter Design 66

4.2.3 Inverter Circuit Design 68

4.2.3.1 Inverter Current Oversizing Factor 68

4.2.3.2 Switching Device Rating 69

4.2.3.3 Turn-off Snubber Design 69

4.2.4 Inverter Power Loss 71

4.2.4.1 Conduction Losses 71

4.2.4.2 Switching Losses 72

4.2.5 Thermal Design 73

4.3 Knowledge Representation for Power Converter System Design 76

4.3.1 Knowledge Base Development 78

4.3.2 Development of Database 81

4.3.3 Graphical Data Representation using TableCurve® 81

4.4 User Interface Design 86

4.5 Summary 88

5. EXPERT SYSTEM BASED AUTOMATED SIMULATION AND DESIGN OPTIMIZATION OF POWER CONVERTER FED AC DRIVE SYSTEM

5.1 Introduction 92

5.2 Converter System Design 93

5.3 Expert System Interface With System Simulation Program 96

5.3.1 Generalized Methodology of Expert System Interface With External DOS Programs 96

5.3.2 Expert System Interface with PC-SIMNON Simulation Program 98

5.4 Knowledge Representation 102

5.4.1 Knowledge Base Structure 102

5.4.2 Expert System for Simulation and Optimization 102

5.5 System Performance 107

5.6 Summary 107

6. EXPERT SYSTEM AIDED AUTOMATED CONTROLLER DESIGN AND AUTO-TUNING

6.1 Introduction 112

6.2 AC Drive Control System Design and Auto-Tuning Methodology 114

6.1 Introduction 112

6.2 AC Drive Control System Design and Auto-Tuning Methodology 114

6.2.1 Control Gain Initialization 115

6.2.2 Control Tuning Methodology 120

6.3 Expert System Based Real-Time Simulation Software Development Methodology 125

6.3.1 Expert System and System Simulation Program Interface With DSP Simulator 126

6.3.1.1 ASCII Floating Point Decimal to IEEE Single Precision Floating Point Hex Format Conversion Routines 128

6.3.1.2 IEEE Single Precision Floating Point Hex to ASCII Floating Point Decimal Format Conversion Routines 129

6.3.2 Methodology of Autonomous Simulation of AC Drive System 130

6.4 Knowledge Representation 135

6.4.1 Knowledge Base Structure 135

6.4.2 Expert System for Control Design and Automatic Tuning 137

6.5 System Performance 139

6.6 Summary 143

7. EXPERIMENTAL VALIDATION OF DRIVE CONTROLLER DESIGN

7.1 Introduction 144

7.2 Hardware Description 144

7.2.1 Hardware PWM Modulator 145

7.2.2 TMS 320C25 DSP Board 151

7.2.3 Analog Current Sensor Interface 155

7.2.4 Speed Sensing and Control Interface 155

7.3 Real Time Software Design 158

7.3.1 Basic Vector Control Functions 158

7.3.2 Synchronous Current Control 160

7.3.3 Control of Modes of Tuning 160

7.3.4 Speed Computation 161

7.3.5 Speed Controller Routine 161

7.3.6 Host to DSP Communication Routine 163

7.4 Experimental Study 163

7.4.1 I_{ds} Loop Tuning Performance 164

7.4.2 I_{qs} Tuning Performance 166

7.4.3 Speed Controller Tuning 166

7.5 Summary 169

8. CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

8.1 Accomplishments of the Work Presented 171

8.2 Recommendations for Future Work 175

REFERENCES 177

VITAE 182

APPENDICES 183

A LISTING ON EXPERT SYSTEM FOR AC DRIVE CONFIGURATION SELECTION AND AC MOTOR DESIGN

A.1 Expert System Root Frame 184

A.2 Expert System for AC Drive Configuration Selection 202

A.3 Expert System for AC Motor Design 209

B LISTING OF EXPERT SYSTEM FOR POWER CONVERTER DESIGN

B.1 Expert System for Converter System Design 218

C LISTING OF EXPERT SYSTEM FOR POWER CONVERTER SIMULATION AND OPTIMIZATION

C.1 Expert System for Power Converter Simulation and Optimization 244

C.2 SIMNON Simulation Programs for Power Converter System 255

C.3 C-Language Based Interface Routines 260

D LISTING OF EXPERT SYSTEM FOR CONTROL DESIGN AND AUTO-TUNING

D.1 Expert System for Control Design and Auto-Tuning 270

D.2 Plant (Power Converter and Machine) Simulation Program in SIMNON 318

D.3 TMS320C25 Based Controller Simulation Routines 321

D.4 C-Based Interface Routines for Controller Tuning 334

E LISTING OF REAL-TIME TMS320C25 BASED IVC-IM CONTROL SOFTWARE 358

LIST OF TABLES

Table no.	Page no.
4.1 A Sample Record from Power Transistor Database File	82
6.1 Switch Positions for Various Modes of Tuning	124
6.2 Parameters of AC200 Induction Servomotor	140

LIST OF FIGURES

Figure no.	Page no.
1.1 AC Drive System Design Flowchart	6
2.1 PC Plus Based Expert System Components	13
2.2 Frame-Subframe Hierarchical Structure for PC Plus based Expert System	18
2.3 PC Plus Based Expert System Interface with External Files	19
2.4 Expert System Development Methodology	21
2.5 Knowledge Representation	23
3.1 Structure of an AC Drive System	31
3.2 AC Machine Type Classification	33
3.3 Rectifier-Inverter Topology	37
3.4 Rectifier-Boost Chopper-Inverter Topology	39
3.5 Dual PWM Inverter Topology	40
3.6 AC Drive Configuration Selection Tree	42
3.7 Motor Nameplate Specs Calculation Flowchart	47
3.8 Expert System Structure for AC Drive Configuration Selection and AC Motor Design	52
3.9 Menu of Choices Offered to the User to Indicate his Type of Application	54
3.10 AC Motor Design Results Screen Showing Designed Motor Nameplate Ratings	56

3.11 Help Message Explaining the Importance of the Acceleration / Deceleration Time for Design 58

4.1 Voltage-Fed PWM Transistor Inverter System for Induction Motor Drive 61

4.2 Diode Rectifier Current Waveforms 63

4.3 Diode Crest Factor and Form Factor Curves 64

4.4 Inverter Voltage and Current Waves at Worst-Case Capacitor Ripple Current 67

4.5 Inverter Phase Leg and Associated Switching Voltage and Current Waves 70

4.6 Thermal Equivalent Circuit for the Power Converter System 74

4.7 Expert System Based Power Converter System Design Flowchart 77

4.8 Expert System Structure for Power Converter Design 79

4.9 PC Plus Interface With Database Files 83

4.10 Conversion of Graphical Characteristics of Power Semiconductor to Picture and Equation Forms Using TableCurve 85

4.11 PC Plus Interface With Graphics Files 87

4.12 User Interface Screen Showing Graphical Prompt of Inverter Short Time Overload Characteristics 89

4.13 Contents of Design Conclusion Screen 90

5.1 Inverter Phase Leg Operating as DC-DC Converter 94

5.2 Loss and Voltage Overshoot Curves for a Transistor at Switch-off with Turn-off Snubber 95

5.3 PC Plus Expert System General Communication Protocol with a DOS File 97

5.4 PC Plus Based Expert System Interface With SIMNON Simulation Program Showing all the Interface Routines 99

5.5 Structure of Expert System Knowledge Base 103

5.6 Snubber Resistance Optimization Algorithm 106

5.7 Steady-State Machine Phase Voltage and Current Waves of a Simulated Drive System 108

5.8 Turn-on Voltage and Current Waveforms for Snubber Resistance Optimization Iteration 109

5.9 Turn-off Voltage and Current Waveforms for Snubber Capacitance Optimization Iteration 110

6.1 A Typical Closed-Loop AC Drive System Control Structure for Servo Speed Control Application 116

6.2 Control Structure of the Decoupled Synchronous Current Control Loops 117

6.3 Speed Control Loop Structure for Servo Speed Control Drive System 121

6.4 Control Tuning Methodology 123

6.5 Software Structure for Expert System Based Controller Design and Auto-Tuning 127

6.6 Pseudo-Multitasking Operation with Continuous Simulation of Plant and Controller from a Batch File 132

6.7 Expert System based Controller Tuning Flowchart Showing Details of Software Execution 134

6.8 Structure of the Expert System Knowledge Base for Controller Design and Auto-Tuning 136

6.9 Passes of Expert System Aided Auto-Tuning of Controllers 142

7.1 Block Diagram of the TMS 320C25 Based Experimental Drive System 146

7.2 Hardware PWM Modulator Block Diagram 147

7.3 Triangular Wave Generator Circuit Diagram 148

7.4 Circuit Diagram of the Sine-Triangle Comparator 149

7.5 Circuit Diagram of the Pulse Separator 150

7.6 Dead Time Generator Circuit Diagram 152

7.7 Output Enable Circuit Diagram 153

7.8 Toshiba Inverter Block Diagram, with the new Hardwired PWM Generator 154

7.9 Motor Line Current Interface Circuit 156

7.10 Circuit Diagram of Speed Signal Interface 157

7.11 Software Structure of the IVC-IM Drive Control 159

7.12 Speed Measurement Flowchart 162

7.13 Experimental Passes of I_{ds} Current Loop Tuning Process 165

7.14 Experimental Passes of I_{qs} Current Loop Tuning Process, with Tuned Values of K_{pd} and K_{id} employed 167

7.15 Experimental Passes of Speed Loop Tuning, with Tuned Values of K_{pd}, K_{id}, K_{pq} and K_{iq} used 168

7.16 Steady-State Performance of the Indirect Vector Controlled Induction Motor Drive System showing Angle θ_e and sin θ_e and Line Currents i_b and i_c waves 170

CHAPTER 1

INTRODUCTION

1.1 Overview

Adjustable speed AC motor drives have evolved tremendously over the past two to three decades. As an interdisciplinary technology, the advances in the area of power electronics and adjustable speed AC drives have been benefited by the progress made in the other allied disciplines such as the power semiconductor device technology, control systems, VLSI technology and the introduction of fast and powerful microprocessors and digital signal processors. Although all of these developments have made the adjustable speed AC drives an essential and integral component of most of the modern industrial automation and control equipment, the task of designing the AC drive systems has become progressively more complicated. The AC drive design engineer has to keep himself up-to-date with the advances and latest developments in the AC motor drives and its component technologies, so that he can incorporate these in his product design, for its market competitiveness.

Traditionally, the AC drive systems in industry have been designed by expert designers using paper, pencil and calculator, often assisted by computer programs. The design is normally conservative and is often guided by empirical and rule-of-thumb principles. Once the design is complete, the values are used to set up a breadboard prototype, to test the design experimentally, and the results of this testing are interpreted and used to refine and fine-tune the design, before the final prototype is built for production. Such a process, however, is time-consuming and expensive. It also requires the expert's know-how for designing as well as testing the AC drive system.

Expert systems, a branch of artificial intelligence (AI), have shown tremendous potential for applications in a variety of disciplines, such as CAD (computer-aided design) of VLSI (very large scale integration) systems and multivariable control systems design. The slightly slower execution time of the expert system software makes its applications limited to the area of real-time control of slowly-changing systems such as chemical process control systems and nuclear power plant control systems. For faster control applications, other artificial intelligence techniques such as fuzzy logic and neural networks have shown considerable promise. However, in the off-line applications such as design, diagnostics, intelligent information retrieval and management systems, expert systems are very well-suited due to their ability to access and process vast amount of knowledge in diverse forms quickly and efficiently. The difficult task of learning the principles and programming in expert systems in languages such as PROLOG and LISP has been made much easier with the recent availability of the development environments, or shells. The expert program developed with such a shell can provide a user-friendly environment such that even a semi-skilled user can easily design a complex AC drive system, with the same ease and quality of output as the design performed by the expert. Expert systems, due to their modular nature, also allow the easy alteration and update of knowledge and data bases to accommodate the newly emerging technological advancements. This allows optimum design of the AC drive system at minimum time and cost.

1.2 Literature Review

Although the expert systems have shown tremendous promise for applications as well as have been applied successfully in diverse areas, their use towards power electronics and AC drives technology remains relatively unexplored. This can be attributed to a general unfamiliarity of the AC drives designer community with the expert system development techniques. The work presented here shows the application of expert systems to the design of AC drive systems for the first time. However, a survey of the recent publications shows that expert system applications in the areas of AC drives and other areas of power electronics applications are growing at a slow but steady rate. Due to the virgin nature of the work being presented here, however, there are very few references available which are directly related to this work.

Fraught[10] identifies the areas in engineering to which AI can be applied. A comprehensive control design package is presented in [4] by Spang, where a computer-aided control system design and

optimization package is presented. The software is a loose combination of a diverse variety of stand alone softwares, and many of the essential ingredients of a federated computer aided design package, such as data communication, design management, memory sharing, supervisory software development, etc. are highlighted. The CAD tool achieves modelling, analysis/design, simulation and implementation of control systems. An expert system based approach to this problem is discussed by Taylor [5]. The benefits of using AI approach towards providing a high-level design environment that is powerful, supportive, flexible, broad-based and readily accessible to non-expert users, are discussed. Basu et al [11] discuss the aspects of applying an expert system approach to control analysis and design, in the case of multivariable control systems. The expert system CONEX presented has the ability to combine both the analytical knowledge embedded in it in terms of models and the heuristic knowledge in terms of the rule base. Through efficient implementation of the inference strategy, it achieves the design of control systems effectively. Subramanyam [8] presents the application of expert systems to VLSI design, developed at AT&T Bell Labs. CAD of VLSI systems is one of the areas for which expert systems are very well-suited. The design task involves behavioral, structural and physical modelling of the system and design, requiring several levels of abstractions. Another area is the placement and routing of various components on the VLSI, which require help of heuristic based search techniques, as exhaustive search is impossible for all the possible solutions. The expert system, Synapse, applies many of the trade-offs involved in the VLSI design to arrive at solutions much quicker and more efficiently.

As for the application of expert systems in power electronics, in one of the earliest references [1] Tam describes the basic philosophy behind using expert systems towards design of static power converters. It presents a broad literature review and the discussion on the merits of applying AI techniques to CAD of power converters. A comprehensive discussion on the application of AI to power electronics can be found in [2] where the state-of-the-art in this advancing frontier of power electronics is covered in detail. Some of the recent applications are discussed. A variety of tools have been developed both by academia as well as industry for the CAD of power electronic circuits. One approach used at General Electric, described by Kuo and Ngo [3] presents a CAD tool for analysis and design of dc-dc converters, where a schematic capture facility is interfaced at the front end of the dynamic system simulator implemented in SIMNON, for the design of dc-dc converters. The paper, however, focusses more on the details on the interfacing than on the actual simulation and optimization. Lorenzo[6] *et al* describe the PECADS software CAD package, which is a PC-based

tool, for the design, analysis and simulation in power electronics. The chief drawback of this system is that although providing the user with extensive modelling and simulation capability, it still lacks in its user-friendliness, in the sense that the user still has to be a knowledgeable designer himself, and also know how to use the software, before he can exploit its abilities. This restricts the use of the software to few expert designers, who might as well do the design themselves. Moreover, the approach to design power converters seems more academic than practical, further reducing its applicability. This package thus becomes just another in line of some already available popular softwares such as SPICE, SABER and Analog WorkBench, to name a few. Kelkar[7] *et al* present a package used at Bell Labs, which allows simulation of power electronic circuits at the device level. It provides detailed performance information such as the device voltage and current waveforms and power dissipations. This is a very low-level software, which only the expert designers can utilize. It again lacks any intelligence, and is useful only as a designer's assistant. Many packages, even based on expert systems have been available for the design of switched-mode power supplies. One example is the package presented by Holtz [14], which designs the high frequency transformer for the SMPS application, by first designing the proper high-frequency magnetic circuit and then selecting the right core material from the database. Among other applications of expert systems to power electronics, Chen and Bose [12] present the expert system based automated selection of industrial AC drive products. The expert's knowledge on matching the application specs with the features available on industrial AC drive products and selecting the one suitable from the database is embedded in the expert system. The intelligent information retrieval system thus provides customizing of the industrial drive product for the user's application. Rajagopalan [13] presents the expert system application to fault diagnostics and monitoring of the AC drive system. The paper, however, does not give any details on the actual implementation of the expert system.

The limited nature of the literature available on the expert system applications to CAD of power electronic systems, including the AC motor drive systems, indicates that there is scope for work to be done on applying expert system principles to the automated design of AC drive systems, which is presented in this dissertation.

1.3 Outline of the Dissertation

The purpose of this project is to develop an expert system based program for the automated design of AC drive systems. The expert system embeds the know-how of the expert design engineer, and based on the details of the application available from the user, configures the AC drive system in terms of suitable power converter topology, type of control and the type of AC motor. Next, it designs the AC motor in terms of its nameplate ratings. Based on the load and motor mechanical and electrical parameters, it next performs the design of the selected power converter circuit. This is followed by the design of the controllers for the closed loop AC drive system. A unique simulation software is developed, performing hybrid simulation of the AC drive system, with the plant, i.e., the converter and AC motor, simulated in a dynamic simulation program, SIMNON and the controllers implemented in actual DSP 320C25 assembly and C language. Once the controllers are fine-tuned, the optimized controller code is downloaded to an experimental AC drive system, an indirect vector controlled induction motor drive, for example, and the control tuning is validated with the help of the experimental setup in the laboratory. The flowchart in Fig. 1.1 clearly outlines the AC drive system design process. The flow of discussion in this dissertation also follows the same sequence.

Chapter 2 broadly reviews the expert system principles and shell based design methodology, since the area of expert systems is somewhat new for the power electronics and AC motor drives community. The terminology related to the expert system is explained in detail. The process of expert system development is also outlined, keeping in mind the discussion in the following chapters. In the later sections of the chapter, details are given of the expert shell used in this project, Personal Consultant Plus (PC Plus) from Texas Instruments. The complete shell-based design methodology for the expert system is explained, with examples, wherever appropriate.

Once the reader is familiarized with the expert system concepts, Chapter 3 introduces the methodology of expert system based AC drive configuration selection and AC motor design. It begins with a broad overview of the AC drives technology. Next, it proceeds to explain various choices available to the AC drive designer for configuring the AC drive, in terms of the types of AC motors, the power converter topologies and control techniques, their relative merits and application criteria. Then, details are given about the methodology of configuring the AC drive system for a number of applications spanning the entire application spectrum of AC drives. A detailed discussion

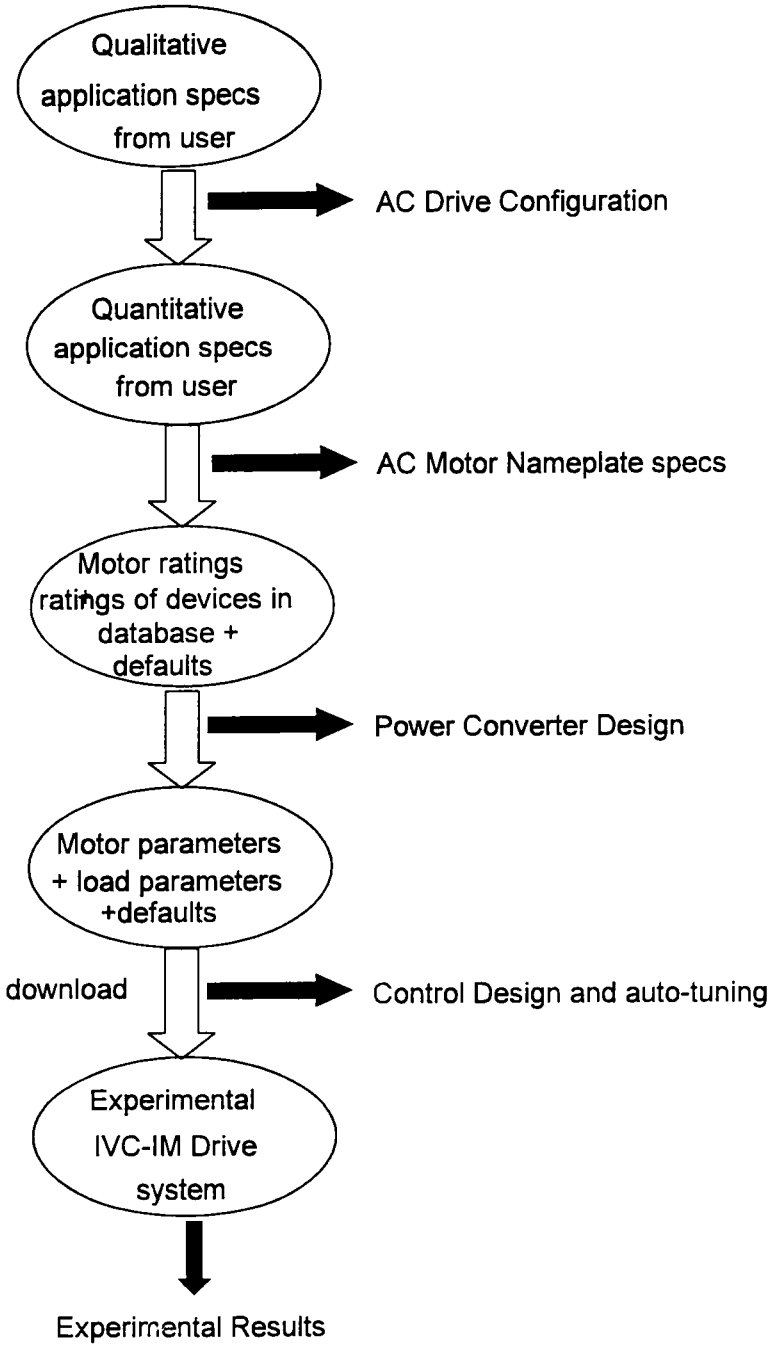


Fig. 1.1 AC Drive System Design Flowchart

on the design of the AC motor nameplate specs based on the user's application specs follows next. Now, the expert system based implementation for the AC drive configuration and AC motor design is discussed in terms of its knowledge representation and user interface design. The results of the expert AC drive configuration selection and motor design clearly show a good correlation with the established practices in these areas.

Once the AC drive system is configured and the AC motor nameplate specifications are calculated, the next task performed by the expert system is the design of power converter system. Chapter 4 begins with the details on design considerations of power converter system. It then describes how the expert's design knowledge on power converter system is embedded in the expert system in terms of the knowledge base and data base. Details on the expert system's interface with database and graphics files are given to explain the complexities involved in the software development. The user interface design is explained with the examples of natural language dialogues and graphics interfaces. The results of the power converter system design are presented at the end of Chapter 4, which show that based on the details provided by the expert system, an actual power converter system can be built in the lab.

Once the various power converter design parameters are available, based on the expert system's knowledge, it is often necessary to fine-tune some of these parameters. This process of optimization can be applied to many facets of the design of power converter system, such as the thermal design, packaging, total cost, efficiency, or reliability of the converter. This requires some converter modelling and simulation, so that the performance optimization can be based on the results of the simulation. The expert system, however, is proficient in performing the symbolic processing type of functions, with limited amount of math capability. and requires the help of system simulation program for optimization of the power converter performance. The methodology and implementation of the expert system for design, simulation and optimization of the power converter system are described in Chapter 5 in detail. The example of turn-off snubber design is used to demonstrate the expert system based optimization. The chapter begins with the details on the snubber design and optimization considerations. Next, the methodology of communication protocol development between the expert system and simulation program is described. The knowledge representation of the converter simulation and optimization is next, followed by a discussion on the system performance, in which iterative tuning of snubber parameters, such as the resistance and capacitance

is demonstrated.

Chapter 6 begins with the discussion on the control design considerations for the AC motor drive system. This is followed by details on software development methodology for the hybrid simulation of AC motor drive and its DSP based control, for real-time controller tuning. The discussion on expert system knowledge-base development follows next. The results at the end prove the applicability and usefulness of this software towards the AC drive control design and tuning. Throughout the chapter, the discussion uses the example of TMS 320C25 based indirect vector controlled induction motor drive system to explain the principles of simulation software and expert knowledge base development. This is to demonstrate the expert system based controller design and tuning. However, a similar approach can be applied to design and tune the control of a variety of AC drive system control configurations. The optimized control software code is later downloaded to an actual experimental drive system, which achieves significant time and cost saving towards the design of AC drive system.

Chapter 7 essentially covers the experimental validation of the controller design and tuning. Details of the hardware development in the beginning are followed by the real-time software design issues. The control software downloaded from the expert system is integrated within the framework of the rest of the I/O and host-to-DSP communication routines. The results of the experimental validation are presented, and the discussion basically focusses on their correlation with the results obtained earlier through the expert system based controller tuning. A close matching between the two is observed, thereby validating and justifying the expert system based controller tuning.

In the end, Chapter 8 summarizes the accomplishments of the work presented here, as well as points towards the direction in which the future efforts in this area should be focussed.

CHAPTER 2

EXPERT SYSTEM PRINCIPLES AND SHELL BASED DESIGN METHODOLOGY

2.1 Introduction

Expert system, a branch of Artificial Intelligence (AI), has been the focus of interest ever since the idea of artificial intelligence was formally conceptualized about four decades ago. Through the use of other AI techniques, such as fuzzy logic, artificial neural networks, genetic algorithms and probabilistic reasoning, the expert system essentially embodies the domain expertise of a human expert. As a result, with the help of the expert system, a skilled, semi-skilled or even a novice user is able to solve complex, real-world problems, with the same ease, speed and quality of an expert. The really outstanding feature of expert system that distinguishes it from other more traditional computer applications is its ability to deal with challenging tasks and produce results reflecting an expert's judgement and intuition.

Before any discussion on expert systems can begin, it is important to realize their special features which distinguish them from procedural programs written in languages such as Fortran, Pascal, Basic, etc. As explained earlier, both types of programs differ not only in their implementations but also in their uses. The main difference between the expert systems and the procedural programs arises from the type of data both of them operate on. The expert systems operate on symbols, or character-strings, as variables, whereas the procedural programs mainly deal with numbers in fixed or floating point numbers. Since most of the experience and know-how of the domain expert is in terms of rules-of-thumb or *heuristics*, and most of the analytical skills based on qualitative reasoning, both of which easy to implement using linguistic variables, it is very convenient to capture the domain expertise using the expert systems. Another key characteristic of expert systems

is the way in which they operate, i.e., there is no fixed procedure to solve problems because the solution(s) is (are) highly problem-dependent, and the program execution follows in the manner in which the values of goals, or result variables, are set. Thus, the order of writing expert system program code is noncritical. This frees the programmer from the straight-jacket of structuring the code very rigidly, as in the case of procedural programs. Because of the same loosely structured and modular nature of expert system programs, they are easy to upgrade and maintain, as the updating of one cluster of knowledge does not affect the other portions of the program. This also helps drive the concurrent engineering of the expert system, in which a number of groups can work on different components of the same expert system program independently and concurrently. Another distinguishing characteristic of the expert systems is their ability to handle uncertain or vague information. This can be of two types: the client may specify the data in terms of qualitative or uncertain values, or the reasoning to arrive at the results may also involve some amount of uncertainty. Expert systems implement uncertainty management using Bayesian (probability) or Dampster - Schafer (possibility) based reasoning. Recently, fuzzy logic based uncertainty management has also been available with expert systems, where membership functions can be created and rules written to manipulate fuzzy variables, during expert system's operation. Last, and most importantly, the expert systems can be designed to have self-learning capability, by using meta-knowledge, so that the program learns from its own experience, and executes the rules in the order of their usefulness to the problem at hand. Expert systems also allow the user to interactively solve the problem, by justifying and explaining the problem solving process , thereby educating the client. All of these features make the expert systems a very powerful tool to use in solving problems that defy the framework of a fixed procedure, which could be either too complicated or case-based and ill-defined. Thus, it is also important to know where expert systems are proper candidates to problem - solving, as against the procedural programs. The following criteria should be considered [16] to determine the applicability of expert system approach to solving a specific problem:

- The knowledge to solve the particular problem is highly centralized
- The procedure or flowchart to solve the problem is available, but is too long, detailed or filled with exceptions to actually be put to use
- The problem needs to be solved very frequently, with varying input conditions
- The solution of the problem needs to be updated frequently, which requires considerable time in training the operators in handling the solution, which can be a product or a process

- A variety of information from heterogeneous sources must be monitored and assimilated to determine the possibility of an important event, e.g., the alarm handling in a nuclear power plant
- Problem is too difficult to solve optimally due to its complexity, and many what-if scenarios need to be tried, in order to arrive at an acceptable solution, for example, placement and routing of a VLSI design layout.

Due to their inherent expert-like information-processing capability, expert systems have been applied in many diverse areas, such as diagnosis, planning, forecasting, design, language interpretation, control, status monitoring and instruction. The expert system based programs are somewhat slow in execution speed, when implemented on today's computers, (whose hardware is optimized to implement numeric processing), so, their application to real-time control is limited to controlling relatively slow control systems such as chemical process control or nuclear power plant control. However, for applications which can be performed off-line, such as the ones enumerated earlier, expert systems are the best suited.

In this chapter, a brief overview of the expert system principles is given in the beginning. The terminology related to the expert system is explained in detail. The process of expert system development is also outlined, keeping in mind the discussion in the following chapters. In the later sections of the chapter, details are given of the expert shell used in this project, Personal Consultant Plus (PC Plus) from Texas Instruments [17]. The complete shell-based design methodology for the expert system is explained, with examples, wherever appropriate.

2.2 Expert System Principles

The expert system can be defined as a set of intelligent computer programs, which embed the know-how of an expert in a very well-defined, restricted area or domain, and solve complex, real-world problems emulating expert's reasoning. The domain expert possesses all the key aspects of expertise, namely, the *knowledge* in the area of his expertise, and the *inference*, or reasoning, that he applies to process the facts, or data pertaining to a complex problem, to produce a result that satisfies all the constraints in the problem. Besides the knowledge and inferencing capability, the expert also needs to communicate or *interface with the user* so that he can acquire the problem

description in terms of specifications, justify and explain his reasoning, as well as provide the solution(s) to the client in the form he can understand and use. Similarly, the expert system has three key elements, which are, the knowledge base, the inference engine, the user interface and external programs, as depicted in Fig. 2.1. These will now be described in detail.

2.2.1 Knowledge Base

Expertise stems from knowledge. The expert draws heavily upon the years of his experience in his domain. The knowledge that he possesses is of two types, one is just the facts and their relationships. This knowledge is called *declarative* or fact-like. This can be likened to a database, in which the facts are stored. The other type of knowledge helps him combine these facts, use his reasoning and rules-of-thumb, to obtain the solution of the problem. This knowledge is called *procedural*, or method-like. Apparently, both type of knowledges are complementary to each other, and are used extensively to solve a problem in the domain of expertise.

Within the expert system, the portion of the program storing the expert's knowledge is called the *knowledge-base*. Knowledge base comprises of a number of components, namely, the elements or facts that are relevant to the problem, the production rules, in the form of IF ... THEN structures, that describe their relationships, rule-clusters, which allow functional partitioning of the knowledge, and activation mechanism, which governs the execution or instantiation of various groups of rules. Clearly, the knowledge base is at the core of the expert system, and it is because of this, that expert systems are sometimes also referred to as *knowledge-based systems.*

2.2.2 Inference Engine

As important as the knowledge is the ability of the expert to reason with it, use the facts, apply problem-solving strategies, to produce the solution. Depending on the problem as well as the data available on it, the expert adapts his problem-solving strategy, applying intelligent assumptions, omitting obviously contradictory information, using the relevant exceptions to the rules of the procedure, and searching for the best set of solutions that satisfactorily meet the constraints imposed by the problem. This efficient manipulating of facts and his know-how is also a characteristic of expertise, and is also an integral part of the expert's know-how. Interestingly and importantly, the

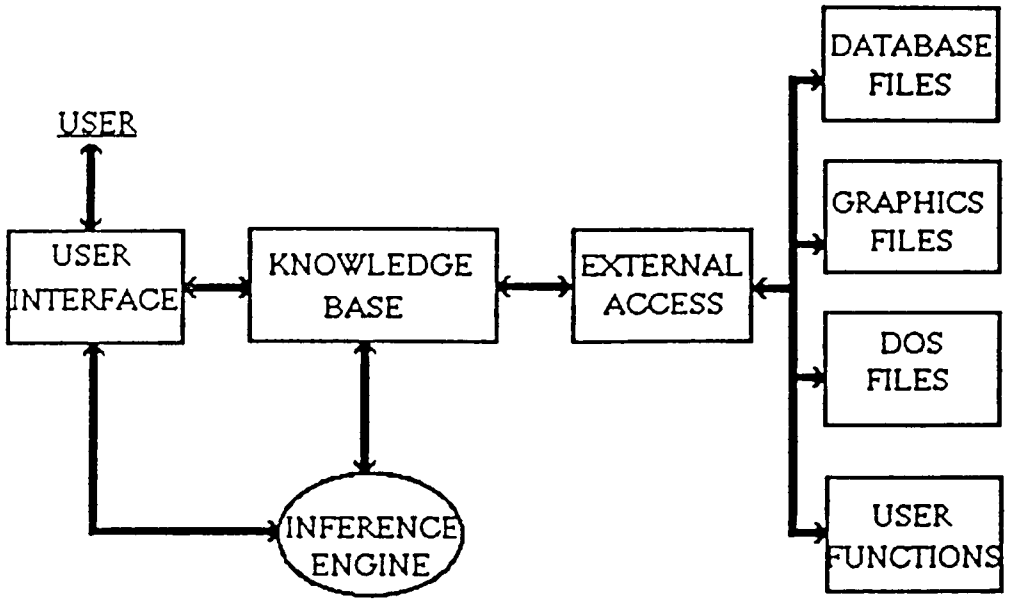


Fig. 2.1 PC Plus Based Expert System Components

problem - solving strategy is general in nature, i.e., the expert is able to apply similar techniques to use the data to produce solutions, regardless of the actual problem at hand.

In an expert system, the strategy of using the information from the client and applying the knowledge base in a manner that produce meaningful solution to the problem is embedded in the inference engine. As explained, the inference engine directs the implementation of knowledge. During the expert system's execution, it is the inference engine that decides which inference paradigm or heuristic search techniques (the techniques to search for the best possible way to execute rules to approach the solution, based on experience). The inference engine actually "runs" the expert system, by determining the rules to be invoked and tried, tracing the parameters of the problem, so that the result values are obtained from the data. In essence, the inference engine searches for problem-specific knowledge, and *infers* new knowledge from it, and continues this process till it has reached the solution. By its very nature, the inference engine is independent of the problem to be solved, and the same can be used for solving a number of problems.

2.2.3 User Interface

Communication with the user is another very important job of the expert. However knowledgeable the expert might be, without his ability to communicate, he is of no importance to the client. During a problem solving process, the expert *interfaces* with the user to determine the exact specifications for the problem, interactively performs the problem-solving to indicate to user the current status of the solution as well as giving the user some indications for special processing, such as irrelevant or inconsistent data, cautioning messages on the possible results, giving the user special instructions and advice pertaining to the solution, and finally the solution set of the problem. The communication proceeds mainly in natural language, such as English. Sometimes, graphics or pictures also help enhance the explanation capabmlity of the expert. Thus, being able to communicate with the user, in a manner that the user can understand, is an essential characteristic of an expert.

In an expert system, it is the user interface that communicates with the user. The communication performed by the user interface is bidirectional. At the simplest level, the client should be able to describe his problem to the expert system, and the expert system must be able to respond with its recommendations. In practice, a user interface generally is expected to perform additional functions.

The client may ask the expert system to explain its "reasoning", for example, or the expert system may request additional information about the problem from the user. Also, although the expert system developer is generally greatly experienced with computers as well as the problem solution, the intended user of expert system is frequently a novice. It is, therefore, critically important for the expert system to be easy to use. The user interface is in the form of natural, or English language dialogues, and images, in the form of computer graphics. The user interface thus enhances the expert system's understandability as well as acceptability by the user.

2.2.4 User Programs

Despite possessing the knowledge, inference strategy and communication skills, the expert sometimes has to perform some preliminary calculations, using calculator, or for a more detailed analysis, he may have to refer to some procedural or simulation programs. Sometimes, he may even refer to data sheets for selection of suitable components matching his problem solution. The domain expert is quite conversant with all of this procedure, and hence knows exactly what softwares to use or where to look for the specific information he wants. Through his years of experience, he has also developed proficiency in using and applying these utilities as a part of problem solving.

The expert systems are very good at handling symbolic data but their math capabilities are limited. Most of the programs or languages support basic math and scientific functions. However, most of the expert system implementation tools have capabilities to interface with external programs, such as procedural programs written in languages like Fortran or Pascal, databases, spreadsheets, graphics programs and simulation programs. These are termed as user programs, and any realistic expert system application will contain some or all of them.

2.3 Expert System Development Tools

Just as the way in which expert system manipulates information is different, so are the languages that enable the expert system development. Commonly used languages are, LISP (LISt Processing), PROLOG (PROgramming LOGic), OPS5 (Operator's Production System), etc. Thus, development of any application in expert system requires programming proficiency in one of these languages, which is difficult to achieve.

Fortunately, the recent availability of expert system development environments, or shells, has made the task of building an expert system to be somewhat easier [15]. These shells offer most of the commonly used development features, such as rule compilation, a higher level, English-like programming language, inference engine, uncertainty management schemes, debugging and run-time application generation. This enables the developer to concentrate on the actual tasks involved in the expert system development The ease in development translates into faster prototyping of the expert system program. Many shells are available commercially, as well as in public-domain (free-of-cost at some ftp site on the Internet). The shells are also available on a wide variety of platforms, such as IBM PCs, Apple - Macintosh, UNIX based Workstations, mainframes and LISP machines, such as the Texas Instruments "Explorer" series. Some of the popularly used shells are, KEE (Knowledge Engineering Environment) from Intellicorp, ART-IM (Automated Reasoning Tool - Information Management), a dialect of CLIPS (C Language Integrated Production System, developed at NASA) from Inference Corp., Knowledge Craft, from Carnegie Group, Inc., S.1 from Teknowledge, SUPERBRAINS, developed at Toyo Information Systems, Japan, EXSYS from Exsys Corporation, Nexpert Object, from Bechtel AI Institute, the Aion Development System (ADS), the KBMS (Knowledge Base Management System) from AI Corporation, Guru, from Micro Data Base Systems, Inc. and G2, a real-time expert system shell available from Gensym Corporation. A detailed description of each of these shells is available through the FAQ (Frequently Asked Questions), on the Internet Newsgroup called *comp.ai.shells.* The shell that is used extensively in this project, Personal Consultant Plus, or PC Plus, from Texas Instruments, has also been widely used for diverse applications. It characterizes all the essential features of the development environments, and hence will be described briefly.

2.3.1 Features of PC Plus Shell

The PC Plus provides a development environment that supports inference mechanisms, database access, natural language dialog interfaces, procedural interfaces, explanation facilities and graphics interfaces. Essential components of a PC Plus based expert system are shown in Fig. 2.1. The development system resides in PC Scheme, a modern dialect of LISP, and provides a user-friendly environment allowing user to develop the expert system using higher level Abbreviated Rule Language (ARL) similar to English. This enables fast and easy program development interactively without the programmer knowing the programming intricacies of Scheme. The ease in expert system

development with PC Plus shell reduces development time at the cost of slight reduction in the execution speed of the program. But for the off-line application such as design, the speed is not critical, and hence, this shell is quite suitable. The run-time delivery software can be in either LISP or C code, as per the user preference. This facilitates easy portability of the program, for PC type or LISP machines. The features of PC Plus can be summarized as follows:

- Provision of a tree-like structuring of knowledge-base (Fig. 2.2), for rule clustering in terms of frames and subframes, and rulegroups. This makes the program execution more efficient by activating or instantiating only the relevant cluster of rules at a time. Each frame / subframe has its own parameters, rules, etc, and the child-parent frame hierarchy allows a child frame access to the parent's parameters, and the parent frame access to the child frame's rules.

- The inference engine follows the backward chaining for goal tracing, i.e., it begins by identifying the goals, or result quantities, and traces the rules that set the values of the goals, in a backward or data- driven manner, asking for the parameter values to the user if it cannot set it through internal processing. There is also a provision to force forward chaining as inferencing strategy, for individual rules or cluster or rules.

- PC Plus also supports an explanation subsystem which helps client while inputting data, explaining *why* it is necessary and justifying how it reached a certain conclusion in simple English language using HELP, WHY and HOW commands.

- Although PC Plus has limited math capability, (supports basic arithmetic and logic functions), it has excellent interfacing capabilities with external files (Fig. 2.3), such as DOS executable or command files, user's customized functions in Scheme, databases in dBASE[18] files, spreadsheets in Lotus 1-2-3, and graphics developed with editors such as OrCAD, Dr. Halo III[19], graphics outputs of SIMNON and TableCurve softwares [20], for display during consultation.

- It also provides means of handling uncertainty by assigning and processing certainty factors to various parameter values during consultation.

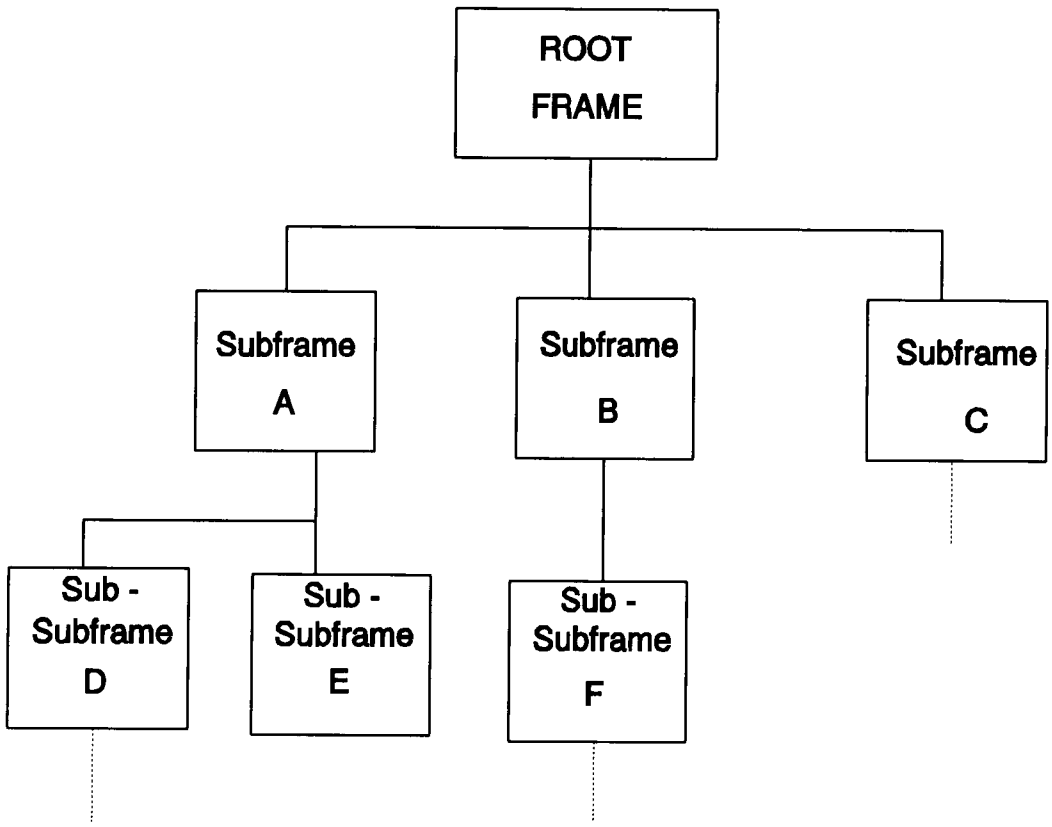


Fig. 2.2 Frame-Subframe Hierarchical Structure for PC Plus based Expert System

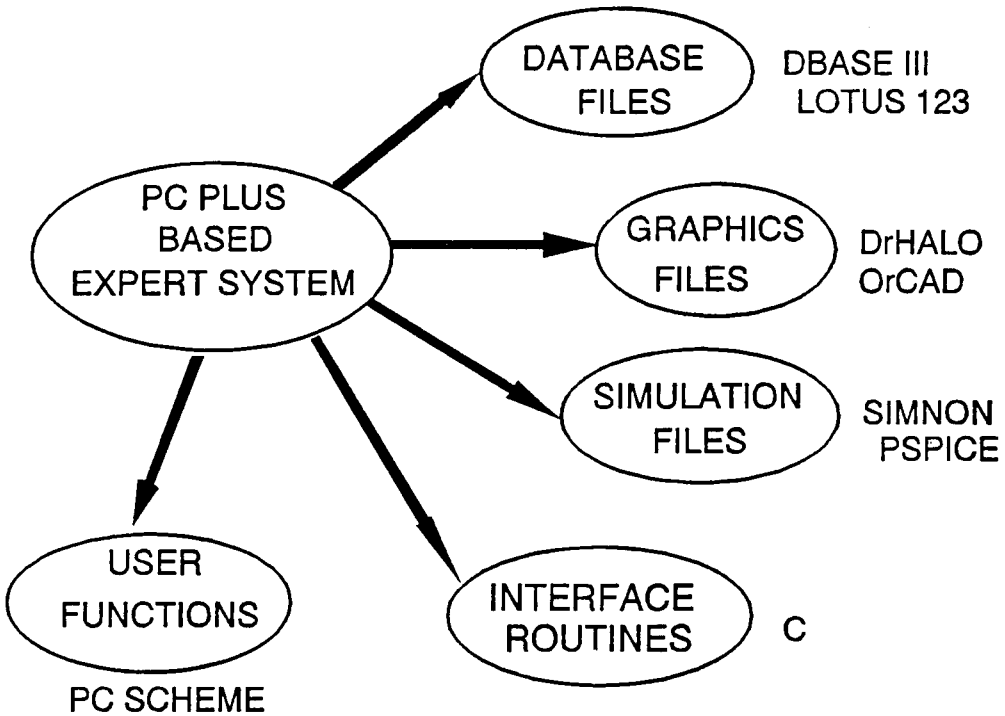


Fig. 2.3 PC Plus Based Expert System Interface with External Files

- The menu-driven system provides multiple windows for user-friendly development support assisting the user with on-line services such as full-screen editor and debugger.

- It provides learning mechanism to handle meta-knowledge using meta-rules, which control and modify the inferencing and instantiation process during a consultation to make the execution more efficient.

- The development support also provides means of generating run-time application version for final delivery, in LISP or C, as per client requirement and for easy portability. The stand alone operation of the delivery version obviates any need for the client to have any compilers or development programs, such as database, spreadsheets, graphic editors etc.

- It provides printed or screen output for the developed knowledge base in ARL, English and LISP version, although the development is done in ARL.

The development system requires one floppy and one hard disk drive, with minimum of 1.5 MB space, a minimum of 640 kB RAM, with 1.5 MB recommended, MS-DOS version 2.1 or higher, and an EGA / VGA with color monitor and printer. For the client, however, the system requirements are a fixed disk drive or a high density drive, 640 kB of RAM, DOS version 2.1 or higher and EGA / VGA with color monitor and printer.

2.4 Shell Based Expert System Development Methodology

Once the programmer has decided to use the expert system approach for solving his problem, using the considerations described above, as well as chosen an appropriate expert system development tool for the size and nature of his application, he is all set to begin the development of expert system. Typically, the process goes through several stages, before the version of the expert program emerges for application to the intended problem. These stages can be roughly classified as knowledge acquisition, knowledge representation, user interface design and testing, as shown in Fig. 2.4. These will now be discussed in detail.

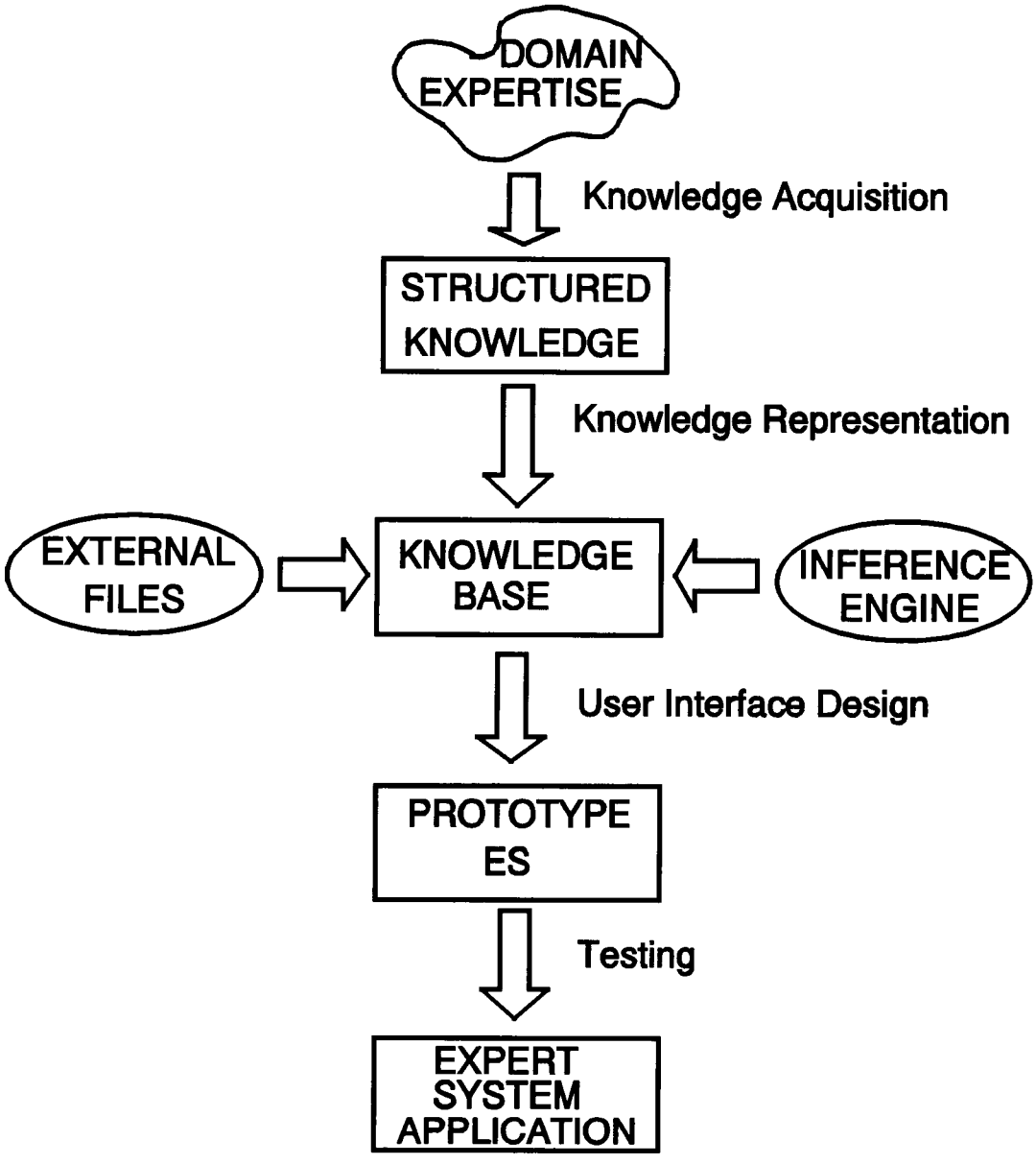


Fig. 2.4 Expert System Development Methodology

2.4.1 Knowledge Acquisition

The first task that the developer performs is to collect all the facts associated with the application, as well as the methodology that the domain expert employs to solve the problem. This is called knowledge acquisition. The process of knowledge acquisition is often difficult, as the expert system developer is proficient with programming skills of the expert system, and the domain expert knows his domain very well, but has no knowledge of expert system programming. Thus, extracting the knowledge from domain expert is an extremely important task deciding the effectiveness of the expert system developed to mimic the expert. The expert information is generally obtained by the knowledge engineer. It is a knowledge engineer's job to interview the expert, identify its key elements, iron out inconsistencies or contradictions, formalize and organize the knowledge in a hierarchical form that can be programmed later, and then later proceed to develop the expert system based on the knowledge obtained from the expert. The ideal case is when the domain expert himself is an expert programmer. In such a case, he doesn't need to go through this phase. At the end of the knowledge acquisition phase, the expert system developer has all the knowledge about the problem domain at his fingertips.

2.4.2 Knowledge Representation

Knowledge representation can be defined as the task of casting expert knowledge within the computer in a framework such that inferences or conclusions can be drawn. It deals with presenting the knowledge effectively and implementing the inferencing efficiently. As explained earlier, the expert knowledge can be categorized as declarative knowledge and procedural knowledge (Fig. 2.5). Depending on the type of knowledge representation technique used, the two types of knowledges can be separate or integrated. In the most commonly used *rule-based* approach for knowledge representation, both the types are integrated. Another way in which the knowledge is classified is in terms of the level of expert's understanding of the domain of expertise. For example, some of the disciplines, such as materials science, life sciences etc., are highly experimental, and all the knowledge acquired by the expert is highly empirical in nature, and is in the form of cause - effect relationships, without the actual understanding of the process that produces the effect from the cause. This type of knowledge is called "shallow", or "case-based" knowledge. This is one of the reasons for the heuristic rules in the expert system. On the other hand, some other processes, such

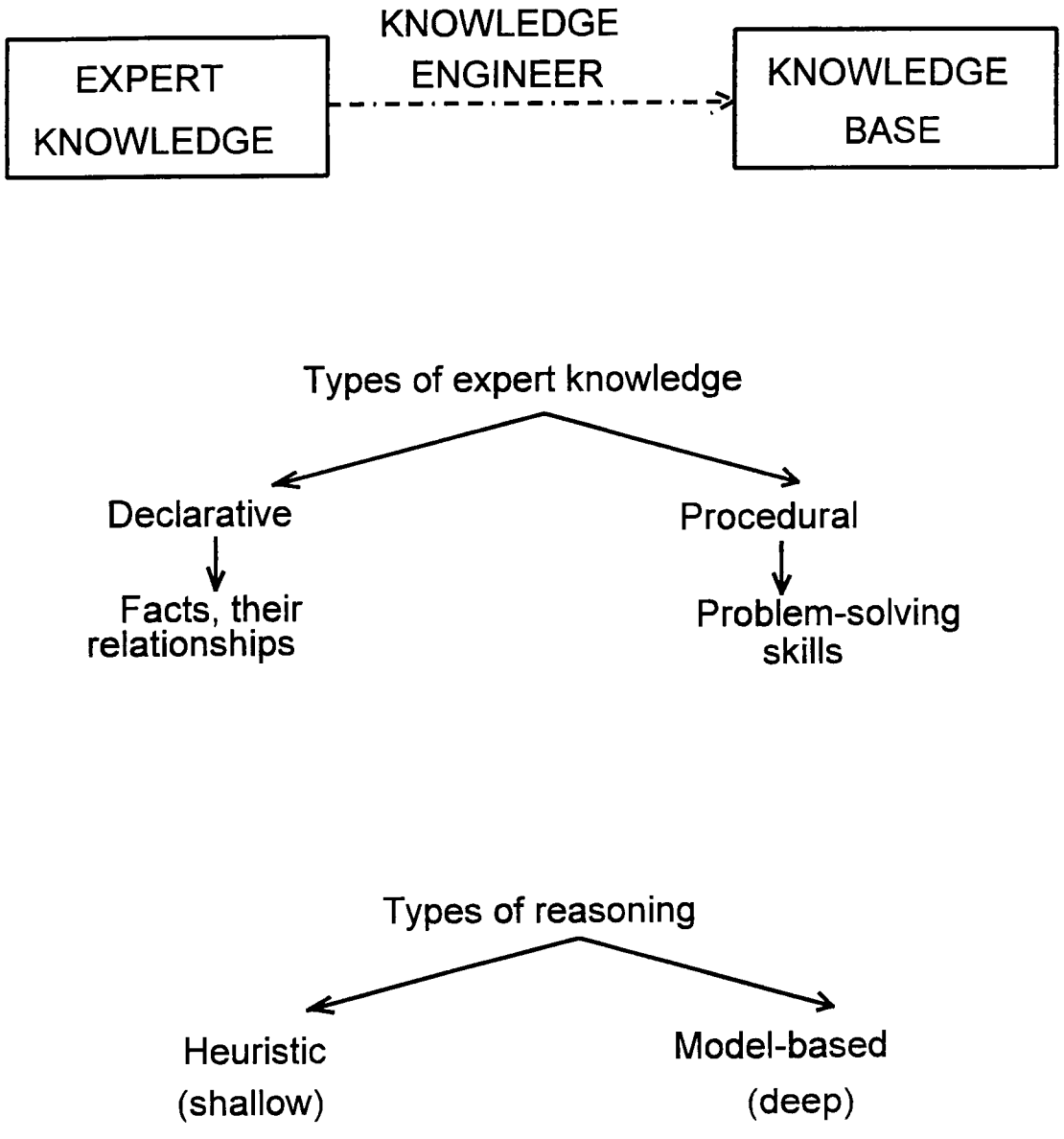


Fig. 2.5 Knowledge Representation

as electrical circuits, mechanical engineering design, thermodynamics, etc., are very well understood, and the models for the processes are very well known. In such cases, the expert knows exact parameters in the process, and the nature of their interactions that affect the behavior of the complete system. This type of knowledge possessed by the expert is classified as "deep" or "model-based" knowledge. The well-defined models can be programmed by a procedural program, to be used to provide the assistance to the expert system during its operation. In any practical expert system, the knowledge represented is a good combination of both types of knowledge.

Knowledge representation in the context of PC Plus based development involves identifying the major tasks involved in solving the problem, and developing a structure for the expert system comprising of sections pertaining to each of these tasks. For example, in an expert system performing automobile diagnostics, the tasks can be diagnosis of problems in the engine, transmission, electrical system, brakes and air-conditioning systems. Further division of diagnostic tasks in engine system can be valve timing, ignition system, fuel pump, cooling system, etc. Once the knowledge is logically partitioned into modules, each of these expert system modules can be developed by first identifying all the possible problems that can occur in it, and preparing a set of symptoms and their possible causes, as well as the relationships used to arrive from possible causes to symptoms. Once this is done, the required data can be identified to pin-point the problem and its cause - effect relationship. This, in essence, is at the core of knowledge representation process.

Next task of knowledge representation is to decide on the type of inference strategy, which operates on the knowledge encoded in the knowledge base. Inference strategies in rule-based expert systems can be goal-driven (a method called *backward chaining*) or data-driven (termed as *forward chaining*). Backward chaining inference strategy starts by hypothesizing a goal value and tries to use the expert system's rules that set that goal value. In contrast to this, the forward chaining technique reacts to externally or internally set data or conditions, and invokes the rules in which these values are a part of premise, or IF ... block. Evidently, both of these strategies are suitable for implementing different class of expert systems. Forward chaining is a natural direction for problem solving when data or ideas are a starting point. Hence it has been used in expert systems for analysis, design, diagnosis and concept formation. On the other hand, backward chaining is more applicable in diagnostics type of applications. In any real-life application, a combination of forward and backward chaining techniques is used. The PC Plus shell, for example, allows implementation of

both types of inferencing techniques, but defaults to backward chaining. In most of the cases, the shell chosen to develop the expert system has provision for implementing both types of reasoning strategies, and hence the user does not have to actually design the inference engine.

Once the logical partitioning of knowledge and selection of search strategy is done, the last task is to develop the control structure, in which is embedded the procedural knowledge. It controls execution of different functional subunits of knowledge base, in a manner which leads to efficient approach towards the goals or results of the problem. In the context of PC Plus based expert system, the functional subunits are called *frames* and *subframes,* the quantities that represent units of knowledge are called *parameters*. The parameters can be of the type *initial data* or *goal*. The core of the knowledge is in the form of *rules*, which are mainly IF ... THEN constructs. In PC Plus, the control knowledge, or knowledge about knowledge or *meta* knowledge is implemented by *meta rules*, which essentially assign *utility values* to rules. The utility values are numbers telling the expert system the relative importance of the specific rule in approaching the goal. These values are modified depending on the frequency with which the rules are tried during various executions of the expert system. Thus, the expert system approaches the solution to a particular problem more efficiently with time. Thus, the expert systems can be made to *learn* from their experience to some extent.

2.4.3 User Interface Design

As explained in the earlier section, the key to the expert system's acceptance by the user, who can be a novice or semi-skilled person, is in providing it with an excellent user interface. Thus, the goal of user interface design is to place more emphasis on the effective communication between the end user and the machine expert in those tasks where user needs assistance - more specifically, on communication between the end user and the information handled by the expert system. A good user interface exploits advances in other companion AI technologies, such as natural language processing, cognitive science, etc. Essentially, the user interface should be able to

- Understand fragmentary and ambiguous requests
- Present the results in a meaningful manner
- Deal effectively with exceptional situations

- Guide the user through complex dialog sequences
- Adapt easily to changing requirements, problem conditions and user's level of understanding

As described in earlier section, PC Plus offers a variety of ways to develop an effective user interface. Thus, in the context of PC Plus based expert system development the user interface design consists of determining the phraseology of the questions being asked during the dialog, the menu choices being offered, messages to appear as "help" during the consultation, generating and transporting the graphical images to the expert system to be displayed at appropriate places. Strategically placed help messages and images, as well as graphical prompts enhance the user's understanding of the problem being solved.

2.4.4 Testing

Once the expert system is designed design completely, it goes through extensive testing cycles. The testing process affects each of the expert system development stages. All the possible conditions of inputs are tried systematically, and the results are scanned for possible discrepancies in setting the goal values, silly inferences, wrong sequences of operations, improper prompts to the user, etc. These are eliminated one by one. Sometimes, wrong assumptions during the knowledge acquisition can cause incorrect operation of the expert system. In such a case, the domain expert is consulted again for clarifications. Sometimes, expert's own beliefs are in contradiction with each other, and then the trade-offs are applied intelligently to obtain a satisfactory expert system performance. After this, the expert system is tested with the actual clients, on a small scale, and their inputs are embedded in the final product, before it is distributed for actual use.

Most of the times, the knowledge representation also involves development of databases, which the expert system accesses during the consultation. The database contains information which, by itself, does not make much sense. But when the expert system uses this information in the context of obtaining data corresponding to the problem being solved, they become indispensable as ordered storage places for enormous amount of data. Standard database programs, such as dBASE, or spreadsheets such as Lotus 1-2-3 can be employed to develop databases, related to the problem being solved. Also, at this time, the models of the well-defined subsystems used for simulation

during the expert system execution are also created using suitable procedural or simulation software.

2.5 Summary

This chapter introduces the reader to the concept of expert systems technology. The distinguishing features of expert systems are discussed in the beginning, and so are criteria for the suitability of expert system approach to problem solving. Shell based expert system development methodology is described clearly in the next section. The features of the expert system development shell, PC Plus are explained in the later section, which also characterize most of the shells available today. The essential ingredients of an expert system program, such as the knowledge base, inference engine and the user interface are explained in details, assuming little or no artificial intelligence background for the reader.

CHAPTER 3

EXPERT SYSTEM BASED AUTOMATED SELECTION OF AC DRIVE CONFIGURATION AND AC MOTOR DESIGN

3.1 Introduction

The electric drives technology has undergone tremendous evolution over the past two to three decades. This rate of rapid development has been possible mainly due to enormous advances in the allied areas such as power semiconductors, control technologies, electric motors, microelectronics and VLSI technology. DC motor drives technology was the first to be introduced mainly due to the emergence of SCRs (Silicon Controlled Rectifiers) and the simplicity and superiority of DC motor control. However, the problems associated with this type of drives, such as high amount of harmonics in the source and load as well as low lagging power factor at the source, due to the phase control, and the limitations of DC motors such as their bulkiness and need for frequent maintenance of commutators and brushes, necessitated the development of AC motor drives. However, AC motors are much more complex to control, since the quantities involved are AC, varying in magnitude and phase with time.

Fortunately, the advances in control theory as well as availability of low-cost microprocessors and DSPs, as well as VLSI circuits for implementation of sophisticated control algorithms and electronics, has enabled real-time control of AC motor drives at competitive cost and performance. The prices of power semiconductors have also been falling steadily, while offering improved power handling capabilities with reliability. Recently, integrated power modules have been introduced having all of the control, protection and power electronics integrated on single chip, for power ratings up to 10s of kWs, pushing the level of power electronics integration still higher. Although

the AC drives technology has reached some level of maturity, the design of an AC drive system for a particular application remains quite a complicated task. This is because the designer has to be conversant with a diverse variety of disciplines outlined above, should have a sound knowledge of the load mechanics and dynamics, electric motors and their application characteristics, the power converter topologies, power semiconductor device technologies, control algorithms for the AC drives as well as their software and hardware implementations using the advanced DSPs and microprocessors. Moreover, all of the above areas are themselves evolving at extremely fast pace, making it even more difficult for the design engineer to keep abreast of the latest techniques and incorporate them in the design of AC motor drive systems.

As explained earlier, expert systems have shown tremendous promise towards solving problems which require integration, assimilation and application of knowledge from diverse sources, and emulating the expert designer's reasoning process to arrive at a solution to the problem. This chapter details the expert system based AC drive configuration selection as well as design of AC drive motor. The expert system conducts a question-and-answer dialog session with the user in simple English language in the beginning. The user is asked to choose the class of his application from a menu of AC drive application classes. If the user's application is not contained in the menu choices, the expert system then asks detailed questions to determine the AC drive configuration. Based on the user's qualitative and quantitative application specifications thus obtained, the expert system selects the type of drive motor, the type of power converter topology and the type of control for the AC drive. Then, it designs the nameplate specs of the AC motor selected, such that the user can later use them to buy a machine off-the-shelf or get it custom-designed.

The chapter begins with a broad overview of AC drives technology. Then, it describes various choices available to the AC drive designer for configuring the AC drive, in terms of the AC motors, the power converter topologies and control techniques, their relative merits and application criteria. Then, details are given about the methodology of configuring the AC drive system for a number of applications spanning the entire application spectrum of AC drives. A detailed discussion on the design of the AC motor nameplate specs based on the user's application specs follows next. Now, the expert system based implementation for the AC drive configuration and AC motor design is discussed in terms of its knowledge representation and user interface design. Wherever appropriate, the example of a sample AC drive application, the *servo speed control drive system*, is used to

demonstrate the configuration of AC motor drive system and the design of the AC motor. The results in the end show the performance of the expert system based AC drive configuration.

3.2 AC Drive Configuration Selection Considerations

AC drive system consists mainly of three components: the AC motor, the power converter and the control, as shown in Fig. 3.1. Hence, the selection of AC drive configuration involves selection of suitable type of AC drive motor, the type of power converter topology and the type of control, for the specific application [24]-[32]. As explained earlier, the selection of suitable AC drive configuration requires a good amount of information on the type of application, the operating conditions and any special requirements from the user. For this reason, the user is asked to furnish the following qualitative specs in the beginning [42]:

- Class of application
- User's choice, if any, for type of machine, converter and control
- The control variable, i.e., speed, position or torque
- Need for speed reversals
- Quadrants of operation, i.e., one, two or four
- Need for drive operation at zero speed
- Required unity input power factor
- Need for high efficiency, compact-sized machine
- Need for low acoustic noise operation

The design process takes into account various distinguishing characteristics of an AC drive system. These are,

- Vector control provides fast and frequent speed reversals and zero speed operation.
- Open loop V / Hz control is standard for applications where high performance is not expected
- A boost chopper in the DC link provides active waveshaping of the supply input current, providing a unity power factor operation
- High switching frequency, in the range of 10 - 20 kHz is necessary, with IGBTs as

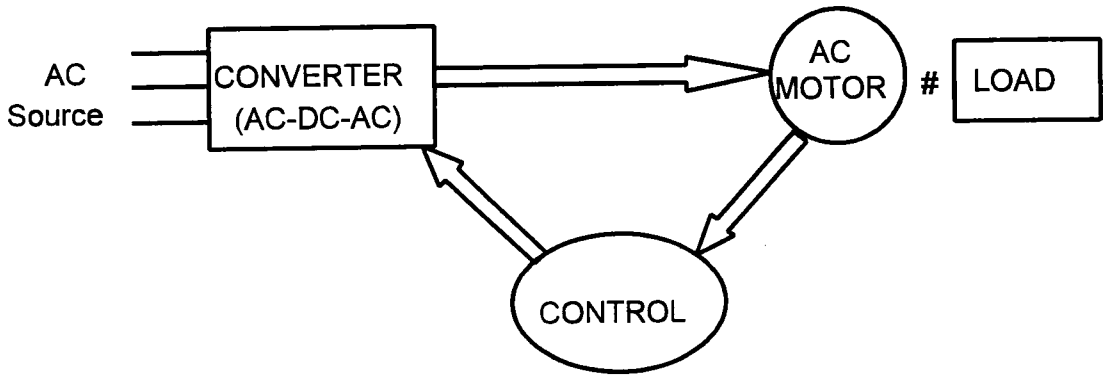


Fig. 3.1 Structure of an AC Drive System

the switching devices, for low acoustic noise operation

- Dual PWM converter provides easy regeneration for four quadrant operation, as well as sinusoidal current at the input, with high power factor.
- Permanent magnet (PM) AC motors offer high efficiency operation, with compact size, but have a relatively higher cost, with low payback period
- Although brushless DC motor drives have the disadvantage of 6th harmonic torque pulsations, they have extremely simple control, and do not need expensive, shaft-mounted speed sensors.
- Permanent magnet synchronous motors, with their sinusoidal airgap flux, allow very smooth control and offer high performance operation with vector control
- Induction motors, especially cage motors, offer rugged construction, wide speed range of operation, moderately high efficiency and high performance control.
- Indirect vector control offers zero speed operation, but requires speed sensor on the shaft, as well as some form of machine parameter adaptation
- Direct vector control offers speed sensorless operation, and has built-in motor parameter adaptation, but fails at zero or near-zero speeds.
- The permanent magnet AC motors are best suited for operation at or below base speed, however, their speed range is extendable to about 160% - 180% of base speed using advanced phase control techniques

The complete process of AC drive configuration selection will be discussed in the following sections:

3.2.1 Selection Criteria for AC Motors

As shown in Fig. 3.2, AC motors are mainly of two types: induction (sometimes also referred as asynchronous) and synchronous. The induction motors can be further classified based on the rotor construction to be wound-rotor type and cage rotor type. The synchronous motors are classified based on the method of producing the magnetic field, to be wound-field synchronous motors or permanent magnet synchronous motors. The permanent magnet synchronous motors can be further classified based on the shape of the rotor mmf, i.e., trapezoidal or brushless DC motors, and sinusoidal permanent synchronous motors (PMSMs). The power range for which the AC drive

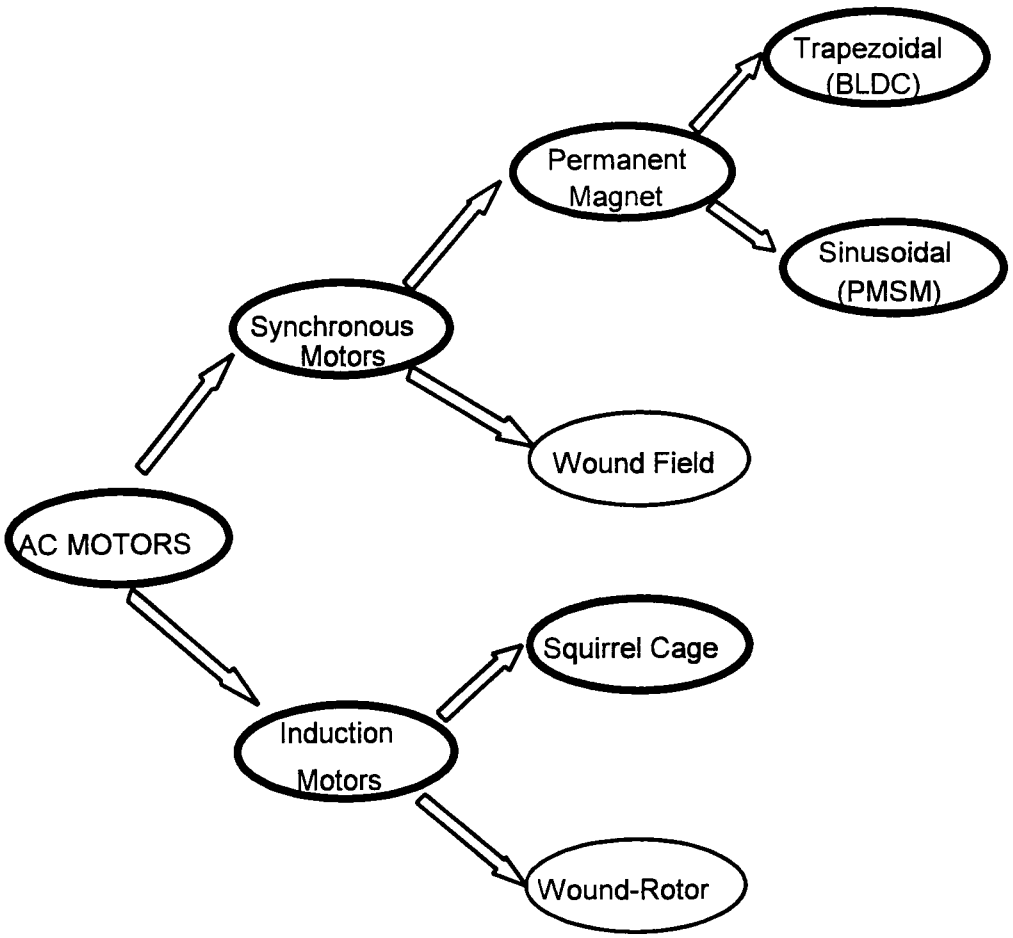


Fig. 3.2 AC Machine Type Classification

applications are considered is from 1 kW to 200 kW, which eliminates the application of drive types such as the static Scherbius drive or LCI fed synchronous motor drive (both used for higher power ratings). Thus, the two most commonly used types of AC motors for drive applications in this power range are cage induction motors and permanent magnet AC motors and are considered for AC motor selection.

The cage induction motors have the advantages of simple and rugged construction, their ability to operate over a wide speed range, moderately high efficiency, low inertia and small size. However, the stator current has also to provide for magnetization of the machine airgap, which allows the flux to be controlled externally and hence the field-weakening operation is possible. But, at the same time, the machine draws lagging VARs from the inverter, and hence increases the kVA rating of the inverter supplying power to the motor. The induction motors can be controlled both in open-loop V/Hz mode as well as in high-performance vector control mode. Induction motors, with a suitable converter topology, can also regenerate to the DC link or to the AC supply, in case of braking or overhauling load.

The PMAC motors [23], on the other hand, have the rotor mounted with permanent magnets, meaning that for the same power rating, the stator current rating is less than that of an induction motor, since the stator current does not contain the magnetizing component. This reduces the inverter current rating for the same output power of the motor. However, the rotor flux now is fixed and cannot be controlled or weakened easily from stator, unlike in the case of induction motors. Thus, the speed range of the PM motors is generally within the base speed. This is because the stator current becomes hard to control beyond base speed, when the motor counter emf begins approaching the DC link voltage as the speed of the motor increases. Recently, however, advances have been made in the control of PM motors, using the technique known as phase angle control for flux weakening, advancing their speed range to 1.6 - 1.8 times base speed. This, however, requires sophisticated, and hence, costly control. However, the power densities of both of these types of machines are significant, resulting in very compact sizes. The power ratings for which the PMAC motors are available commercially are up to 100 kw. The compact rotor size with the permanent magnets allows these machines to have high Torque / Amp capability. The rare earth magnetic materials used such as samarium-cobalt (Sm-Co), and more recently, neodymium-iron-boron (Nd-Fe-B) cause them to cost higher than induction motors. However, since the rotor rotates at

synchronous speed, there are no currents in rotor, resulting in zero rotor copper losses. This is the reason for the PMAC machines having a very high efficiency operation. Thus, although their initial cost is higher, the payback period is reasonably small. The permanent magnet AC motors are classified as the trapezoidal type, which is also known as brushless DC type, and the sinusoidal permanent magnet synchronous type motors.

For their control, the brushless DC type of motors require rotor position sensing, which is easily obtained by three stator-mounted, low-cost, Hall effect sensors, which sense the trapezoidal magnetic field position every 120° electrical. These signals can be decoded through a simple logic circuit to produce the gate signals for the inverter, resulting in a very simple control. The only drawback in using brushless DC motors is that due to the trapezoidal nature of the flux and back emf as well as six - step square waveshape of impressed stator currents, the motor develops torque with 6th harmonic pulsations. This is not a serious consideration if the load inertia is sufficiently large to filter it out, or if the load can withstand slight amount of speed pulsations. Moreover, the magnets are surface-mounted to achieve non-salient rotor design, which is strongly favored by the trapezoidal excitation due to their phase inductances remaining constant with respect to the rotor position. This makes the machine inoperable at higher speeds due to the centrifugal force tending to weaken the mechanical bonding between the surface-mounted magnets and the rotor surface.

The other type of PMAC motors, the permanent magnet synchronous motors (PMSMs), which exhibit exactly the same characteristics as synchronous motors, but with a permanent magnet, have sinusoidal mmf distribution around the airgap. Due to this, the rotor can have salient pole structure, which can be buried under the rotor surface. This gives the rotor more mechanical strength, allowing higher speed operation. PWM switching is used for the inverter connected to the PMSM, which facilitates smooth torque production. The PMSMs can be operated in open-loop V/Hz with embedded squirrel cage in the rotor, and no rotor position sensor is required. This can be employed in high efficiency pump and fan type of drives. However, synchronous-induction motors are not considered for application here. For high performance applications, vector controlled PMSM drive operation is desirable. The vector control of PMSMs is implemented in the way similar to the control of induction motors, the only difference being the absence of flux or magnetizing current control, due to the permanent magnets. The vector control and the control of PMSM, however, requires a costly absolute position sensor on the rotor shaft.

3.2.2 Power Converter Topology Selection Criteria

For adjustable speed AC drives, regardless of the type of application, a variable voltage variable frequency (VVVF) operation is required. Due to their widespread use and popularity, DC link, voltage-fed type configurations are considered for the power range of the AC drive systems. In order to convert a fixed voltage, fixed frequency supply to the VVVF required at the motor terminals, the power converters require two stages of operation, one stage (rectifier) converting the AC input supply to DC, with the help of a capacitive filter and the other stage (inverter) converting the DC voltage to VVVF AC at the output. Thus, the For the type of AC motors considered for applications, i.e., for PMAC and induction motors, three possible power converter topologies exist. These are, the rectifier-inverter, rectifier-boost-chopper-inverter and dual PWM inverter. The dynamic braking resistor in the DC link is used frequently, depending on the type of application for which the power converter is being used.

The rectifier-inverter topology (Fig. 3.3) is the most widely used for majority of AC drives. This is due to its simplicity of operation, very compact integration on the same heat sink and versatility in control implementation. A diode bridge rectifier at the supply input converts the AC fixed voltage fixed frequency supply to fixed DC voltage, with the help of an electrolytic capacitive filter in the DC link. This DC voltage is then converted with the help of transistorized bridge inverter, employing BJTs or IGBTs as switches, to VVVF source for the AC motor. Pulse-width modulation (PWM) is employed to produce a smooth sinusoidal current waveshape in the output. The only disadvantage with this scheme is that the supply input current flowing into the rectifier is in the form of pulses as the inverter appears as a capacitive load to the source. This is alleviated somewhat by employing series reactors either in the DC link or at the input terminals. Due to the rectifier's capability to allow only unidirectional power flow, this topology is applicable mainly for single quadrant operation applications, and with the help of a dynamic brake resistor in the DC link, allows two quadrant operation of the drive, provided the braking energy can be absorbed by the dynamic brake.

As outlined above, the diode rectifier-inverter topology has the drawback of producing a very poor supply current waveform in the form of pulses. In case the application demands a unity power factor operation at the input, this arrangement is unacceptable. Even with the additional reactors, the improvement is not satisfactory, and some means of active waveshaping of input current is needed.

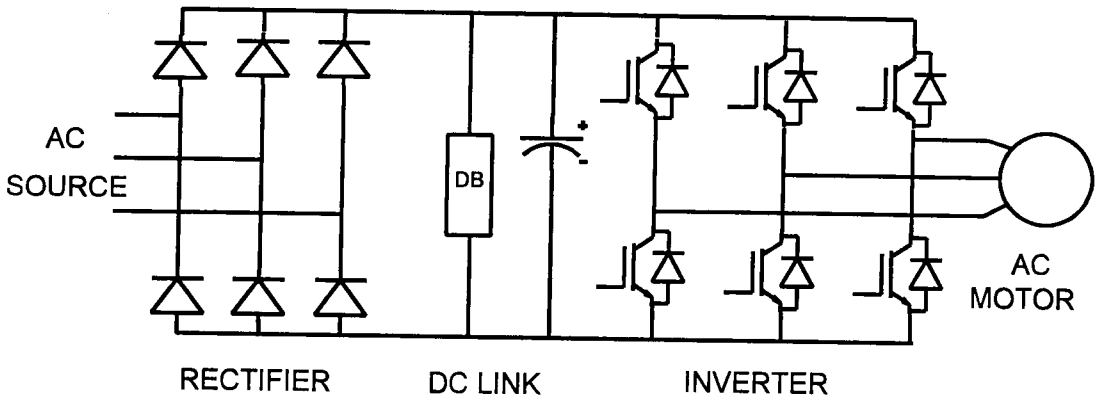


Fig. 3.3 Rectifier-Inverter Topology

This is achieved by the rectifier-boost-chopper-inverter topology (Fig. 3.4), with an extra stage in the DC link, in the form of a boost chopper, which increases or boosts slightly the DC link voltage, to force a sinusoidal DC link current. This is reflected as a sinusoidal supply current at the input. Thus, active current waveshaping and unity input power factor are obtained. However, the additional stage requires extra components and adds to the cost and complexity of control. Also, the overall conversion efficiency of the power converter is degraded by additional cascading stage. Hence, this type of converter topology is employed only if the application requirements impose it. It should be noted here that this configuration, as in the case of rectifier-inverter topology, allows single quadrant operation, which can be extended to two quadrants in some applications by adding a dynamic braking resistor in the DC link.

The rectifier-inverter or rectifier-boost chopper-inverter topologies, as explained above, allow two quadrant operation of the AC drive at the most. However, for some applications requiring four quadrant operation, it is necessary for the rectifier at the front end to have bidirectional power flow capability. This is achieved by employing a controlled PWM rectifier using the self-commutated switches such as IGBTs at the front end. The power converter topology resulting from this is called a dual PWM inverter (Fig. 3.5). An important side effect of this arrangement is automatic sinusoidal waveshaping of the supply current resulting in a unity power factor operation. Since the supply side and load side converters can now exchange roles between conversion and inversion very easily, the converter topology allows bidirectional power flow, resulting in four quadrant drive operation, which is suitable for elevator or traction type of drives. The superior performance, however, comes at a cost of additional power transistor bridge, adding not only to the cost of the total system, but also to the control complexity. Thus, its use should be justifiable in terms of the amount of energy regenerated.

3.2.3 Selection Criteria for Suitable Control for AC Drive System

An important consideration for the design of AC drive system is the type of control employed. The combination of power converter and induction machine can be seen as multi-input, multi-output nonlinear feedback system, with parameter variation problem. Thus, the control system that achieves speed, position or torque control has access to the feedback variables, such as the stator voltages and currents as well as rotor speed or position. Depending on the application, the type of control necessary can be classified as scalar and vector control. In the case of scalar control, only the

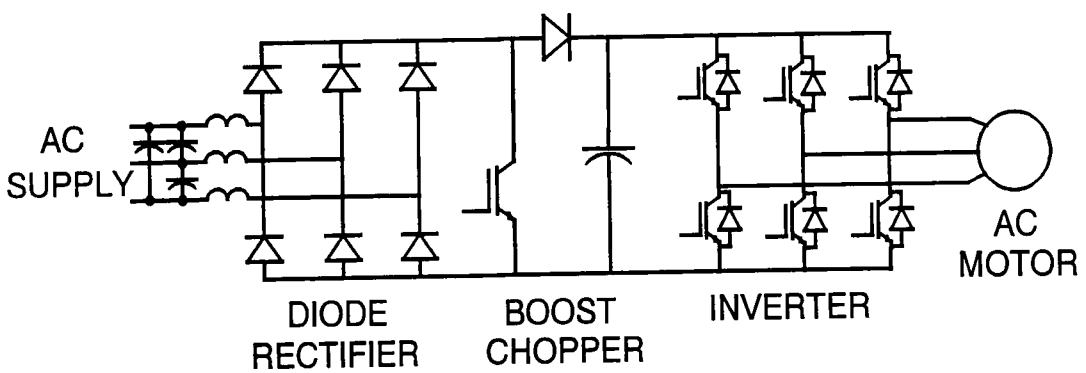


Fig. 3.4 Rectifier-Boost Chopper-Inverter Topology

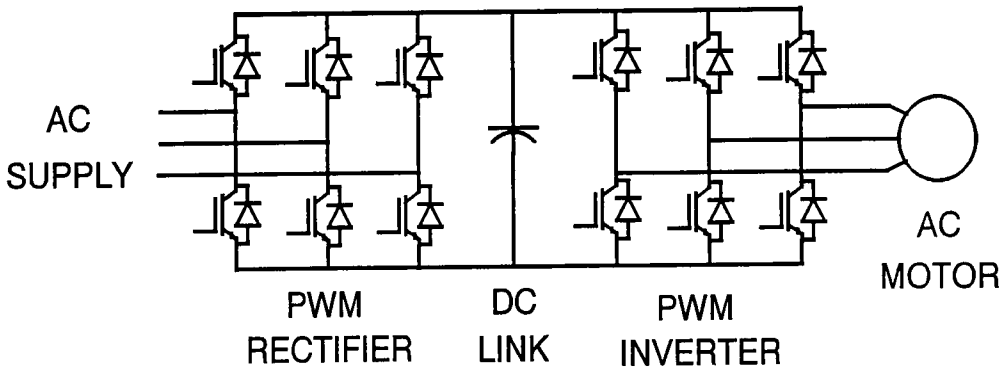


Fig. 3.5 Dual PWM Inverter Topology

magnitude of the control variable is controlled, whereas in case of vector control, both the magnitude and phase of the control variable, which is now a vector, are controlled.

For applications such as pumps or fans, the load requires variable speed operation for energy saving. However, no precise control of the speed is necessary, and in such a case, a simple, open-loop system is sufficient. In such a case, the magnitude of stator voltage is varied in proportion with its fundamental frequency, resulting in a constant Volts / Hz operation (Fig. 3.6). The airgap flux, which is proportional to the ratio of stator voltage and stator electrical angular frequency, is held roughly constant, and the speed of the motor is varied by varying the frequency. Although this type of arrangement offers satisfactory performance in steady state, its transient response is poor due to a reduction in flux during load disturbance or fast-changing speed command. However, this is not an issue for non-critical applications such as pumps and fans as described earlier. At very low speeds, the Volts / Hz ratio is increased, by adding a boost voltage to compensate for a larger component of the supply voltage being absorbed by the stator resistance.

In order to achieve excellent transient response characteristics, as well as fast speed reversals and zero speed operation, another form of control, described as vector control, is employed. In such a scheme, the magnetizing and torque-producing components of stator current, or, alternately, both the magnitude and phase of the stator current vector are controlled. This allows the flux in the airgap to be held constant, at its rated value, allowing the maximum torque sensitivity in the motor. In terms of the d-q axis model of the machine, when the d axis is aligned, at all instants, with the rotor flux vector, the d-axis component of the stator current controls the flux, analogous to the field current in the DC machine, whereas the q-axis component of stator current controls the machine torque, which is analogous to the armature current in the DC machine. A precondition to this behavior of the AC machine is that the electrical angle θ_e of the rotor position be known at all instants, so that the d-axis component of the stator current can be aligned with the rotor flux at all times. If this is achieved, the transient response achieved from the AC drive system becomes comparable to that of a separately-excited DC machine. Moreover, the stability of the AC machine is always maintained by the control, never allowing the breakdown torque limit to be exceeded. Also, the four quadrant operation of the AC drive is easily accomplished. This is the reason why vector controlled drives can be employed for high performance applications, which were previously using DC machines. However, as mentioned earlier, in order that a perfect decoupling of flux and torque producing

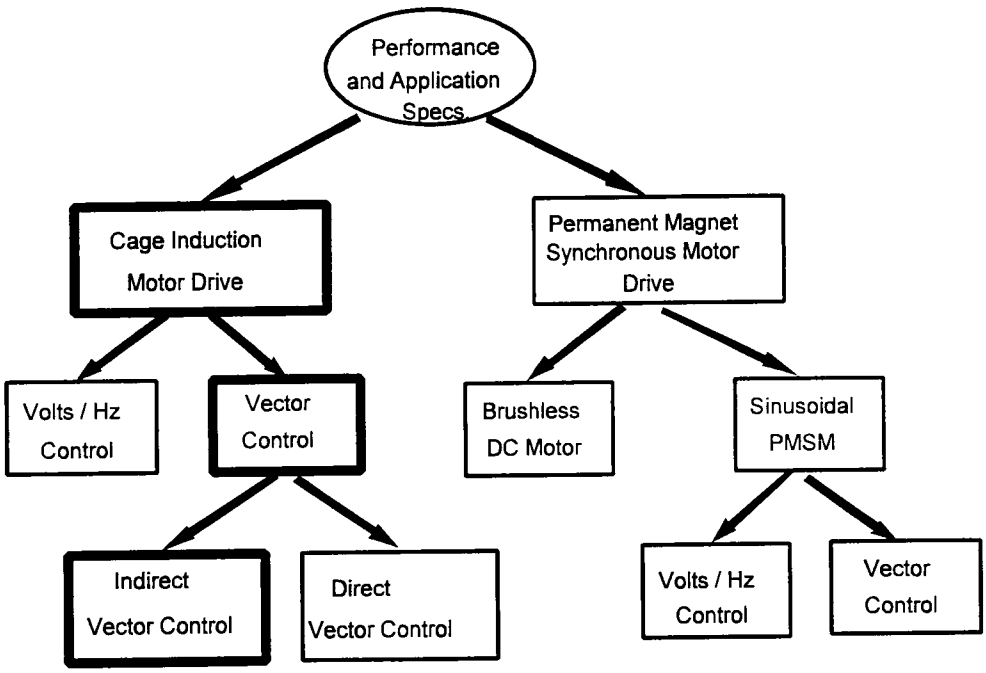


Fig. 3.6 AC Drive Configuration Selection Tree

components of stator current is achieved, the rotor electrical angle should be known at all times. It is the way in which the rotor electrical angle is generated that distinguishes between two principal techniques employed for vector control. These are called the direct vector control (DVC) and the indirect vector control (IVC)[42].

In the case of DVC, the θ_e value is generated directly from the d and q- axis components of stator fluxes, which can be measured directly from the stator - mounted flux sensors, or indirectly through the integration of induced emf across the airgap, from the stator voltages and currents. This technique has several advantages. Since the θ_e information is derived without actually sensing the rotor position, a rotor shaft encoder is not required in this implementation. This reduces the cost of control hardware and improves the reliability of the system. Also, the computation of stator fluxes does not require rotor parameters, and hence the scheme is immune to variation in rotor parameters such as the rotor resistance. The only major limitation of this type of vector control is that the range of operation of the drive is limited to up to 10% of base speed. This is because the noise in the stator signals becomes predominant at lower speeds, making it difficult to achieve accurate integration of voltages to obtain the fluxes. Thus, for applications not requiring near-zero speed operation this type of vector control is ideally suited.

If, however, the application requires zero speed operation, the DVC cannot be employed. In such a case, a different, *indirect*, method of rotor position sensing is necessary. This is achieved by using a shaft-mounted rotor speed (or, occasionally rotor position) sensor. The rotor electrical angular velocity ω_e is computed indirectly by adding to the rotor angular velocity ω_r, the slip frequency ω_{sl} calculated in feedforward manner from the command value of synchronously rotating reference frame q-axis component of stator, I_{qs}^*, by multiplying it with the quantity known as the slip gain K_{sl}, which is given by the following equation:

$$K_{sl} = \frac{L_m R_r}{|\Psi_r| L_r} \qquad 3.1$$

where, L_m is the magnetizing inductance of the machine, R_r is the rotor resistance, L_r is the rotor inductance, $|\Psi_r|$ is the magnitude of the rotor flux.The angle θ_e is computed as follows:

$$\theta_e = \int \omega_e \, dt = \int (\omega_r + \omega_{sl}) \, dt = \int (\omega_r + K_{sl} I^*_{qs}) \, dt \qquad 3.2$$

It is from this indirect method of computation of θ_e, that this type of vector control is termed as indirect vector control, or IVC. As is evident from the way the angle θ_e information is derived, IVC implementation mandates the shaft-mounted rotor position or speed sensor to be used. This adversely affects the cost and reliability of the system. Moreover, as seen from equation 3.1, the decoupling of d- and q-axis current components is also dependent on the machine parameters such as the rotor time constant and the rotor flux magnitude, and a change in any of them, if uncorrected, will also degrade the performance of the drive. Thus, rotor parameter adaptation should be employed in the form of slip gain tuning. The advantage of this form of vector control, as identified before, is its ability to operate at zero speed with ease.

3.3 AC Drive System Configuration Methodology

With the years of experience in designing AC drive systems for various applications, the expert follows a methodology, or procedure, as shown in the flowchart in Fig. 3.6. To start with, the expert has exact knowledge on configuring the AC drive system for a number of specific applications, for example, for pumps, fans, elevators, servos, etc. So, the first information he obtains from the user is whether the user's application falls within any of these applications for which he has a great deal of design knowledge. If this is the case, he doesn't need to ask any more questions about the configuration of the AC drive, except for the user's preferences for the AC machine, the power converter and the control strategy. In the case of servo speed control drive, for example, the expert system automatically infers the following information:

- The drive requires precise speed control
- Fast transient response (acceleration / deceleration) is required, with frequent reversals
- The speed range of the drive operation is between zero and base speed
- The drive requires zero speed operation

If, on the other hand, the user's application is new for the expert, he needs to know some application-specific qualitative specifications, such as whether the application needs zero speed operation, whether the application requires precise control of position or torque or speed of the motor, whether high input power factor operation is needed, etc. This information is obtained so that it can be

matched with the characteristics available with the AC motors, the power converters and the control. In case of incomplete information from the user or when more than one choices qualify, trade-offs are made by choosing intelligent defaults to simplify the AC drive configuration process. For example, induction motor is chosen as the default motor type, the power converter topology is defaulted to voltage-fed rectifier-inverter, and depending on whether the application is high-performance or open-loop type, indirect vector control or open-loop V/Hz control are defaulted for type of control respectively. The expert then presents his recommendation for the AC drive configuration appropriate for the user's application. As explained, the complete AC drive configuration selection process involves not only the expert's knowledge on the various types of AC drive components available, but also his judgement and intuition as to what type of configuration best meets the application's characteristics.

Once the AC drive configuration is selected, the expert uses some simple calculations, along with some criteria for oversizing of motor and inverter, to design the motor nameplate specifications, as will be discussed next.

3.4 AC Motor Design Considerations

Normally, the design of the AC motor involves design of its nameplate ratings, and based on various other considerations, the electromagnetic and mechanical design of the motor in details. The latter part is accomplished by the use of some excellent motor design programs available commercially without any difficulty. Hence, the design of the AC motor here is restricted to finding out the nameplate ratings of the AC motor.

Before the nameplate ratings of the AC motor are calculated, the following defaults are used:

- Motor service factor = 1.15
- Motor number of poles = 4
- Motor NEMA class = Class B
- Motor power factor = 0.85 lagging
- Motor efficiency = 90%

Also, before the nameplate ratings are calculated, the following information is asked from the user:

- The load power rating, or steady-state torque demand at base speed
- The base speed of the load
- The maximum speed of the load
- The supply voltage and number of phases
- In case of a battery-fed system, the battery voltage
- The load inertia
- The minimum acceleration time for the load to accelerate from minimum to maximum speed

Based on the above ratings, the motor nameplate specs are calculated in the following manner, which is depicted by the flowchart in Fig. 3.7

3.4.1 Motor Power Rating

Two types of motor power ratings are calculated: the motor steady-state power rating and the motor peak power rating. The steady-state power rating of the motor is the same as the load steady-state power rating. However, the motor peak power rating is determined from the acceleration-deceleration requirements of the load. The motor steady state power rating may be given by the user, in terms of the steady-state power demand of the load. However, if the user specifies the steady-state torque at base speed, the steady-state power rating of the motor can be calculated as follows:

$$P_{ss} = T_L \omega_b \qquad 3.3$$

where ω_b is the base speed of the load in mechanical radian/sec, T_L is the load torque in Nm at the base speed and P_{ss} is the steady-state power rating of the AC motor.

Once the steady-state power rating of the motor is determined, the next task is the computation of its peak power rating. As mentioned earlier, this is governed by the transient overloading considerations of the load, in terms of short-time acceleration and deceleration requirements. The maximum power rating of the AC motor is calculated as follows:

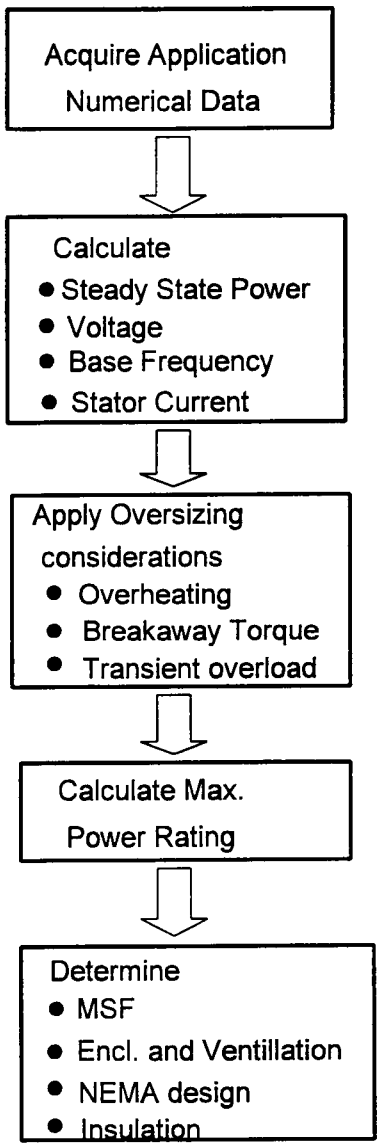


Fig. 3.7 Motor Nameplate Specs Calculation Flowchart

$$P_{max} = J\frac{(\omega_{max} - \omega_{min})}{t_{Amin}} \qquad 3.4$$

where, P_{max} = the peak power rating of the AC motor in Watts, J = combined load and motor moment of inertia in Kg-m^2, ω_{max} and ω_{min} are the maximum and minimum speeds in which range the load accelerates. t_{Amin} is the minimum acceleration time required by the load to accelerate.

When the motor is connected to the inverter, it is the motor peak power that 150% of the inverter steady-state power rating should match. Thus, the motor peak power decides the inverter steady-state power, and not the motor steady-state power.

3.4.2 Motor Voltage Rating

The motor voltage rating mainly depends on the DC link voltage, which, in turn, could depend on the supply voltage in the case of an AC drive system being fed from the AC source. On the other hand, if the inverter is battery-fed, it is the battery voltage that decides the motor voltage rating. Another important consideration for the motor voltage rating is whether the drive is operated in open-loop V/Hz mode or vector control mode. If the drive is operated in open-loop V/Hz mode, then the output voltage transitions to square-wave mode at base speed. Hence, the V/Hz control is lost at base speed, allowing the motor to operate at slightly higher voltage. However, in the case of vector control, it is necessary to stay in the PWM mode of operation even at base speed. This reduces the motor voltage rating by a small amount, as will be shown later.

In the case of applications requiring open-loop V/Hz mode, the motor voltage rating V_L is the same as the AC supply voltage V_s, i.e.,

$$V_L = V_s \qquad 3.5$$

However, in the case of a vector-controlled drive, the motor voltage rating is given by,

$$V_L = m\sqrt{3}\frac{V_s}{2} \qquad 3.6$$

where m = the maximum value of the modulation index for the PWM, which varies between 0.85-

0.95. V_s = the source rms voltage.

For a battery-fed drive system, such as in the case of an electric vehicle drive, the motor rms voltage rating is given by

$$V_L = m\sqrt{3}\frac{E_b}{2\sqrt{2}} \qquad 3.7$$

where E_b = battery voltage, m = maximum modulation index of the PWM voltage waveform.

3.4.3 Motor Current Rating

The motor rms current rating can be obtained once the motor power and voltage ratings are available, from the following relationship,

$$I_L = \frac{P_{ss}}{\sqrt{3}\, V_L\, \eta_M \cos\,\phi} \qquad 3.8$$

where I_L = motor current rating, V_L = motor voltage rating, cos ϕ = the motor rated power factor which varies in the range 0.85-0.9 and η_M = AC motor efficiency in the range of 0.85 - 0.9

3.4.4 Motor Base Frequency

The motor base frequency is obtained from the following relation, assuming that the motor number of poles = 4.

$$f_b = \frac{N_b}{30} \qquad 3.9$$

where f_b = base frequency in electrical Hz of the motor, N_b = base speed of motor in mechanical rpm

3.4.5 Motor Enclosure and Ventilation

For the application requiring the motor operation at low speeds (i.e., below 50% base speed), delivering constant or rated torque, forced ventilation is necessary. For hazardous environments, the

motor should be totally enclosed, and in the event the motor cannot be ventilated, the motor insulation should be a class higher for temperature rise (Class H). For applications having variable torque load, however, the enclosure or ventilation are not a consideration.

3.5 Knowledge Representation for AC Drive Configuration Selection and AC Motor Design

The knowledge for the AC drive configuration selection and AC motor design, as outlined in earlier sections, is embedded in the expert system in the form of the knowledge base. The declarative knowledge describing the application characteristics of different types of AC motors, power converter topologies and control is in the form of IF ... THEN type rules. For example, the following rule selects the power converter topology to be dual PWM inverter for an application requiring four quadrant operation, such as the elevator drive:

IF: APPLICATION = ELEVATOR OR FOUR-QUAD-OPER-REQUIRED
THEN: TYPE-OF-POWER-CONVERTER = DUAL-PWM-CONVERTER

The knowledge base also contains rules that impose on the expert system not to ask redundant or meaningless questions, if the information can be inferred from available data. For example, in the case of servo speed control drive application requiring operation at zero speed, the minimum speed of the drive is obviously zero. In such a case, the following rule directs the expert system to set the minimum speed of operation to be zero, and not to ask for it from the user:

IF: APPLICATION = SERVO_SPEED_CONTROL
THEN: MIN-SPEED = 0 AND
NOFINDOUT MIN-SPEED AND
NOPROMPT MIN-SPEED

The function NOPROMPT directs the expert system not to prompt for the minimum speed and the NOFINDOUT function imposes on the expert system not to find out the value of the minimum speed through trial of other rules, as it is already known.

The procedural knowledge in the form of the AC drive configuration selection methodology, is implicitly implemented in the expert system. The inference engine inherent to the PC Plus shell based expert system guides the AC drive configuration selection by searching for the goal values using backward chaining, as it tries the rules that set the values for the type of machine, converter topology and control. The procedural knowledge also contains information on the design of motor nameplate specifications, such as the output power, voltage rating, base frequency, etc. The equations 3.3 through 3.9 are implemented in this manner. Since the nameplate rating calculations employ simple arithmetic operations, the expert system handles them without any problems. For example, the following rule calculates the motor current rating from the motor voltage and power ratings, assuming the motor power factor = 0.85 and motor efficiency = 90%:

IF: MOTOR-VOLTAGE-RATING IS KNOWN AND MOTOR-POWER-RATING IS KNOWN

THEN: MOTOR-CURRENT-RATING = MOTOR-POWER-RATING / (1.732 * MOTOR-VOLTAGE-RATING * 0.85 * 0.9)

3.5.1 Knowledge Base Development for AC Drive Configuration Selection

The knowledge base for the AC drive system configuration, as explained, contains all the essential information for AC drive configuration. The basic elements of the PC Plus based knowledge base are the frames, subframes, parameters and rules. The frames and subframes represent a group or cluster of rules that perform certain function, such as AC drive configuration or AC motor design. Each frame is associated with a group of parameters, which represent bits of knowledge. The parameters are processed by IF...THEN type of rules. The frame - subframe hierarchical structure of knowledge base allows its functional partitioning, and only the rules belonging to the active frame are considered for instantiation at any given time, making the execution of the expert system efficient.

The motor power considered for the present design is in the range of 1-200 kW. The knowledge for the AC drive configuration selection is represented in the form of two frames. The structure of the knowledge-base with all the associated elements is shown in Fig. 3.8. The root frame, AC-DRIVE-SYSTEM-DESIGN, essentially contains the knowledge on determining the type of application, and

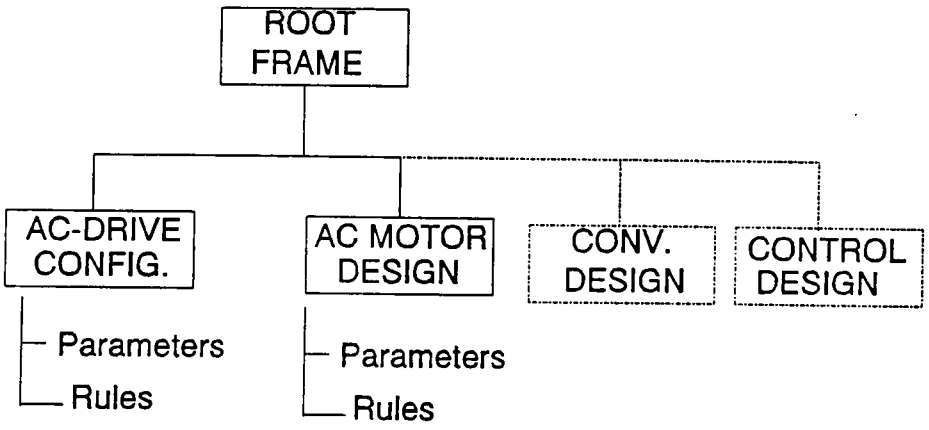


Fig. 3.8 Expert System Structure for AC Drive Configuration Selection and AC Motor Design

control knowledge on the order in which the tasks should be executed, to obtain complete AC drive system design. It achieves this by instantiating or activating the subframes pertaining to different tasks in a specific, logical order. Root frame also contains the goals or the result quantities that are to be found, i.e., information indicating completion of the AC drive configuration, the AC motor design, and the design of power converter and control. In the very beginning, the root frame asks the user the type of his application for the AC drive system in the form of a menu as shown in Fig. 3.9, and invokes the frame AC-DRIVE-CONFIGURATOR, to determine the configuration of the AC drive system, using the following rule:

IF : APPLICATION IS KNOWN
THEN : CONSIDERFRAME AC-DRIVE-CONFIGURATOR

In the subframe AC-DRIVE-CONFIGURATOR is embedded the knowledge on matching the application characteristics with that of a suitable AC drive configuration. As indicated earlier, if the user's application is one from the applications already known to the expert system, the AC drive configuration for that application is performed readily, keeping user's preferences in mind. However, if the user chooses USER_DEFINED, as the application choice, all the information to determine the type of AC machine, power converter and control is asked from the user. This is later used to determine the configuration of the AC drive. The inferencing uses backward chaining to trace the goals. The declarative and procedural knowledge on AC drive configuration is incorporated in this subframe. The data that the expert system asks from the user initially is used by various rules in the subframe, in a manner that assigns values to the goals of this frame, which are essentially the type of converter, type of machine and type of control. In certain cases, however, the user may have to be informed about some inconsistency in the data he has input. A rule performing such a function is not intended to set any value for the goal parameters. Thus, it will not be tried in the normal course of goal-driven backward chaining process for problem solution. However, the expert system allows such actions to be performed with the help of forward-chaining or antecedent rules. For example, if the user's application is elevator drive, which essentially requires the operation of the high-performance AC drive system at zero speed, and if the user indicates his choice of vector control to be direct vector control, the expert system warns him using the following antecedent rule:

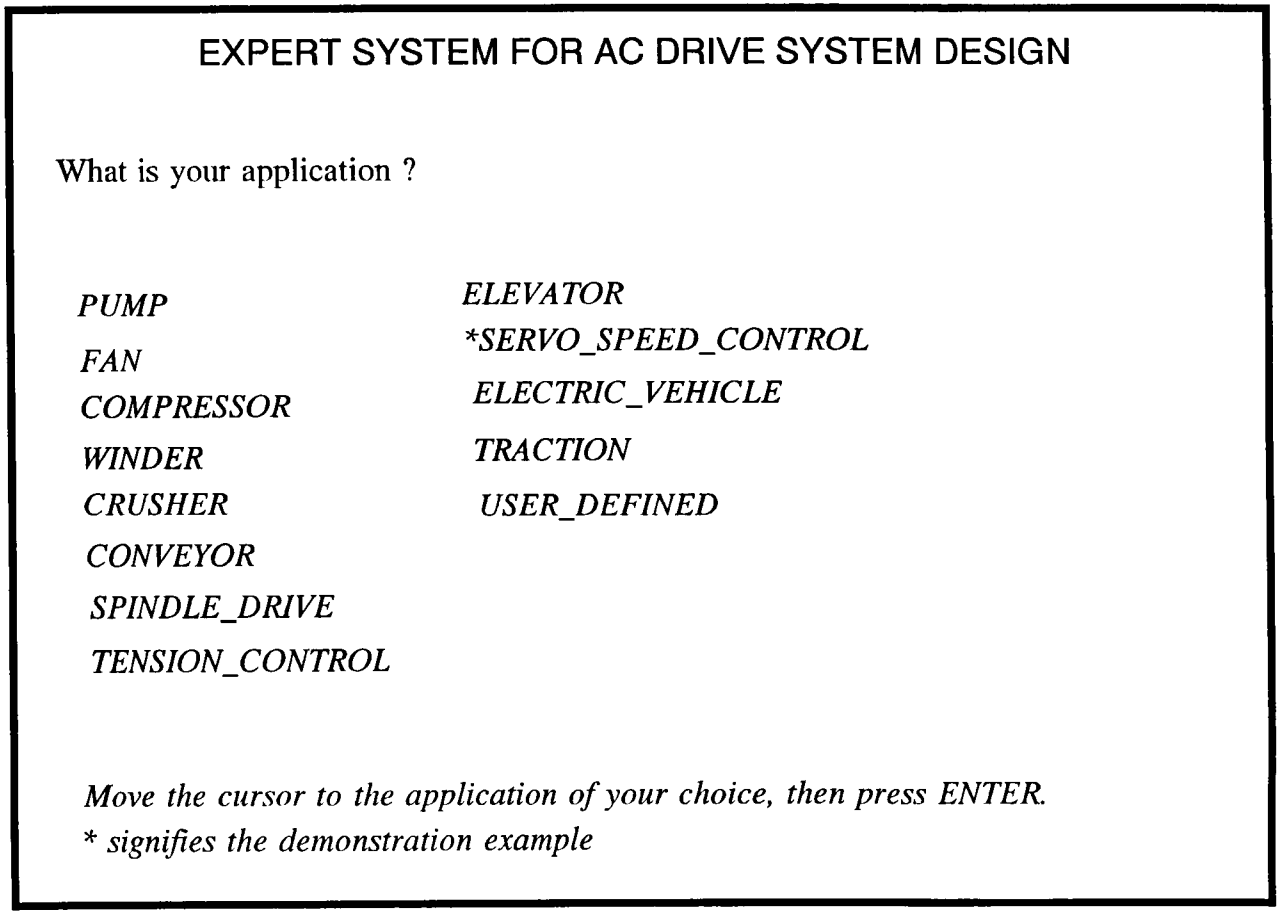


Fig. 3.9 Menu of Choices Offered to the User to Indicate his Type of Application

IF : APPLICATION = ELEVATOR AND CHOICE-FOR-CONTROL = DIRECT-VECTOR-CONTROL

THEN: PRINT "Your application requires zero speed operation, and you have chosen your type of control to be direct vector control, which cannot perform well at zero speed. Please change your response for type of control to be INDIRECT vector control, so that the consultation can continue"

ANTECEDENT: YES

Once all the information about the application is collected, the suitable type of AC motor is first selected. Then, the power converter topology is chosen and then the type of control is determined. This information is now displayed as intermediate results.

3.5.2 Knowledge Base Development for AC Motor Design

The expert's know-how on designing the nameplate ratings of AC motor is embedded in the subframe AC-MOTOR-DESIGN. Once the AC drive system configuration is done, the root frame invokes the subframe AC-MOTOR-DESIGN, to design the nameplate ratings of the AC motor, which can be an induction motor or permanent-magnet AC motor. Equations 3.3 through 3.9 are implemented in the form of rules in this knowledge base. Certain other motor design considerations, such as the motor oversizing due to fast acceleration / deceleration, etc. are also embedded in various rules. For example, the following rule calculates the maximum torque required by the motor during acceleration of inertia-type load:

IF : MIN_SPEED IS KNOWN AND
MAX_SPEED IS KNOWN AND
INERTIA IS KNOWN AND
MIN_ACCN_TIME IS KNOWN

THEN : ACCN_TORQUE =
INERTIA * (MAX_SPEED - MIN_SPEED) / MIN_ACCN_TIME

The expert system displays the results of the AC motor design, once all the nameplate ratings of the AC motor are found. A picture of the AC motor design results screen is shown in Fig. 3.10.

EXPERT SYSTEM FOR AC DRIVE SYSTEM DESIGN

The results of expert system based AC motor design are as follows:

Type of AC motor selected	*: Three Phase Induction Motor*
Motor Steady-State Power Rating	*: 10 kW*
Motor Peak Power Rating	*: 15 kW*
Motor Voltage Rating	*: 220 V*
Motor Current Rating	*: 35 A*
Motor Base Frequency	*: 73 Hz*
Motor Number of Poles	*: 4*
Motor NEMA Design	*: Class B*
Motor Enclosure and Ventillation	*: OSV*
Motor Service Factor	*: 1.15*

Press ENTER to Continue

Fig. 3.10 AC Motor Design Results Screen Showing Designed Motor Nameplate Ratings

3.6 User Interface Design

As highlighted in the introductory chapter on the expert systems, an effective user interface of the expert system is very important. The function of the user interface is to guide the skilled or semi-skilled user through the complicated process of designing the complete AC drive system by providing him with useful suggestions, comments, tips and on-line help messages to explain and simplify the AC drive design process to the user. Without the user interface, clearly, the user will be completely lost as to how to proceed with using the expert system for AC drive system design.

The user interface in the case of AC drive system design is in the form of natural language dialog and help messages presented to the user during the consultation. PC Plus allows the development of both text and graphical form of interface for the user, through built-in functions. Fig. 3.11 shows a sample help message appearing on the screen, with the expert system's prompt asking the user to input the minimum value of acceleration / deceleration time. The introductory message at the beginning of each major stage in design of AC drive system outlines the procedure followed by the expert system to achieve the goals of that stage. User interface is also facilitated in the form of periodic status messages on the screen, and intermediate results presented to the user at appropriate stages in the design. The user is thus kept informed on the current status of the AC drive system design.

3.7 Summary

This chapter presents expert system based AC drive configuration selection as well as AC motor design. The discussion begins with broad-based AC drive system design considerations, outlining characteristics of options available for the type of AC machine, power converter topology and control of AC drive. AC motor design methodology is presented in detail next. This is followed by the explanation of expert system based AC drive system design methodology, the knowledge base development for AC drive configuration and AC motor design, and its user interface design. Wherever appropriate, examples are given to explain the expert system development. The methodology of expert system based AC drive configuration selection is explained with the example of servo speed control drive application, which will be continued in the future chapters.

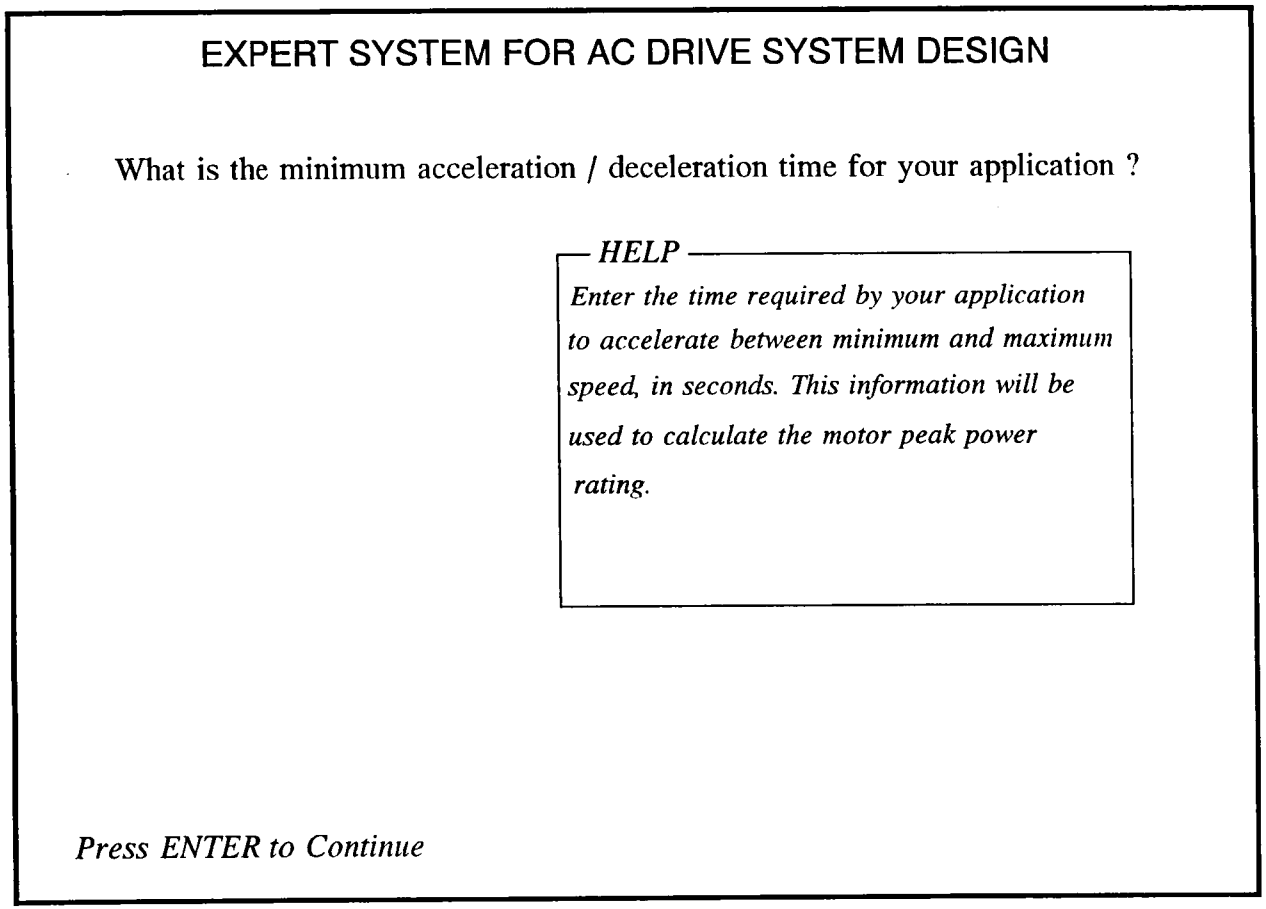


Fig. 3.11 Help Message Explaining the Importance of the Acceleration / Deceleration Time for Design

CHAPTER 4

EXPERT SYSTEM BASED AUTOMATED DESIGN OF POWER CONVERTER SYSTEM

4.1 Introduction

Traditionally, power converters are designed by power electronics engineer with the help of paper, pencil, calculator and often assisted by a computer program. The design is normally conservative and is done often by rule-of-thumb principles. This approach of designing is expensive and time-consuming, and the design is not necessarily optimum. Moreover, the power semiconductor device, converter and the control technologies have been advancing at a fast pace in the recent years and it is being difficult for a general power electronic engineer to keep abreast with these developments and incorporate the latest technology in the optimum design of the products. As explained earlier, expert systems offer an excellent means of transporting the designer's expertise into the software program, which even a semi-skilled person can understand and use, as well as get educated in the process. Again, as explained earlier, the knowledge and data bases can be easily altered or updated as the converter technology advances and new power devices are introduced. This chapter discusses in detail the converter system design considerations, the knowledge representation of the expert system, the user interface design and finally the system performance. The work discussed in this chapter was published and presented[33] in IEEE Industry Applications Society Annual Meeting at Houston in 1992.

4.2 Converter System Design Considerations

The expert system program considers the automated design of the most popular power circuit configuration, i.e., a voltage-fed PWM inverter system, shown in Fig. 4.1. The configuration consists of a front-end diode rectifier, dc link electrolytic filter capacitor and PWM inverter that uses either bipolar junction transistors (BJTs) or insulated gate bipolar transistors (IGBTs). A dynamic brake can be used in the dc link, but it is not considered. The circuit can be used for up to several hundred kilowatts of power.

The design of the power converter can be resolved into the design of diode rectifier, inverter with associated snubbers, dc link electrolytic capacitive filter and the system heat sink ratings. Before the design can begin, the following specs are to be supplied by the user:

- Power of the load machine
- Line and load rated voltage and frequency
- Source inductance or short circuit ratio
- Number of line phases
- Load acceleration / deceleration requirements
- Motor service factor

The design procedure should also take into consideration the typical power converter system characteristics, such as,

- The input line current wave is in the form of discontinuous pulses. Therefore, calculation of its rms, average and peak values is somewhat difficult.
- The inverter output current contains PWM carrier frequency and its side band related harmonic ripples. This causes additional loading of the machine and the inverter.
- The inverter input current contains complex harmonic frequency ripple. This, along with the supply - injected ripple, should be absorbed by the filter capacitor.
- The filter capacitor should maintain a relatively smooth dc link voltage.
- The motor service factor, (usually 1.1-1.15) determines the continuous motor rms current capability with the inverter injected harmonics. For continuous inverter

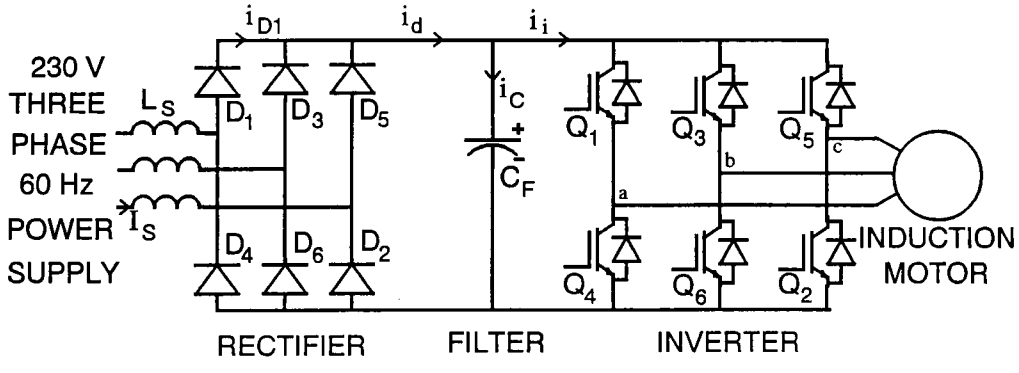


Fig. 4.1 Voltage-Fed PWM Transistor Inverter System for Induction Motor Drive

operation, this constraint has to be maintained.

- The general-purpose inverters are designed to have 150% short-time overcurrent rating according to NEMA standard ICS-3-301
- For safe inverter operation, the dc current rating (I_{dc}) of the power switches should not be exceeded (except on 1% duty cycle basis). Again, the devices should operate within the rated maximum junction temperature (T_{jmax}). For high reliability applications such as military equipment, the ratings of power devices are selected much higher than the designed values, so that they operate at much lower junction temperature, resulting in improved reliability.

The design of power converter system components will be described in the following sections:

4.2.1 Design of Diode Rectifier

The typical line current, rectifier diode current and the corresponding form factor (FF) and crest factor (CF) for a three phase rectifier are shown in Fig. 4.2 and 4.3 respectively. The shapes of curves in Fig. 4.3 are influenced by the parameter $x = L_s I_d / V_s$[35], where L_s= source inductance (mH), I_d = average dc link current (A) and V_s = rms line voltage of source (V). For a single phase rectifier, the shape of the current pulses remains the same, except that a single current pulse is contributed per half cycle. Again, since only two diodes conduct in both single phase and three phase rectifier case, the equivalent circuit will contain $2L_s$ inductance, but the conduction duty cycle of a diode will be reduced in three phase case. The diode and line currents for a single phase rectifier can be given as,

$$I_D = \frac{I_d}{2} \qquad 4.1$$

$$I_s = FF \cdot I_d \qquad 4.2$$

$$I_{Dr} = \frac{I_s}{\sqrt{2}} \qquad 4.3$$

$$I_{Dp} = CF \cdot I_s \qquad 4.4$$

Where I_D= diode average current, I_{Dr} = diode rms current, I_{Dp} = diode peak current and I_s = line rms

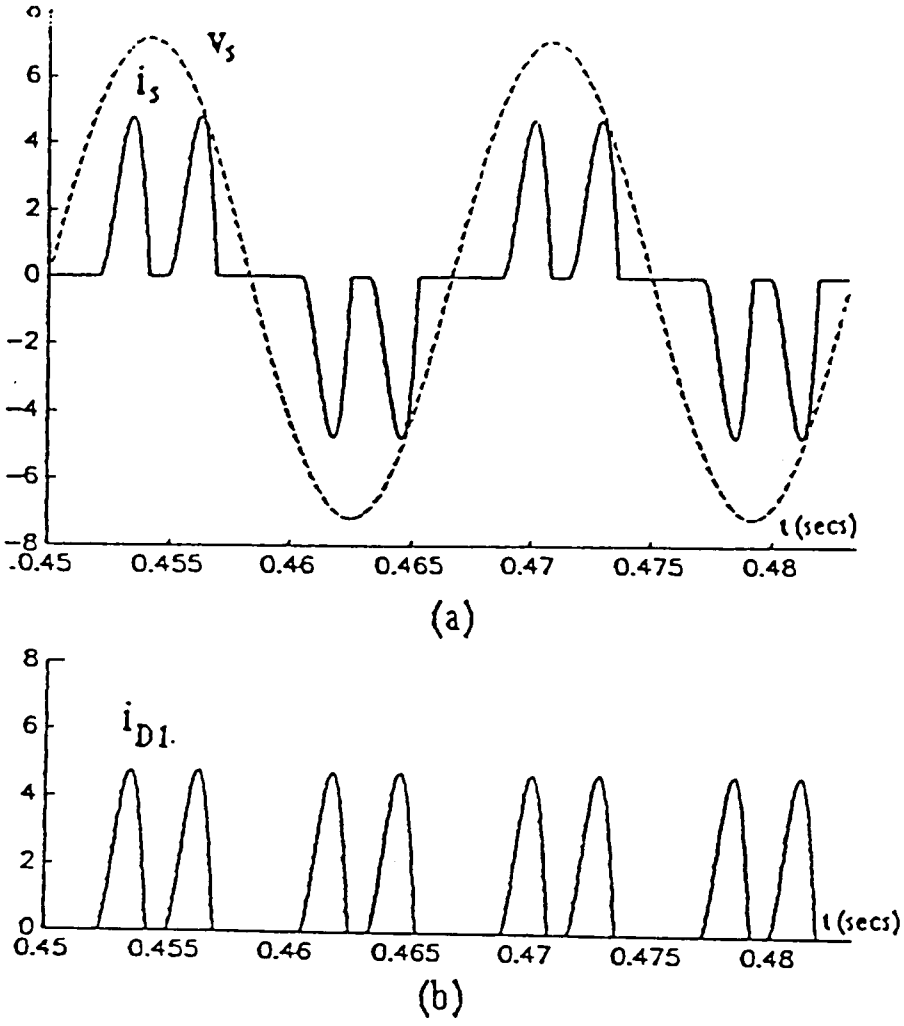


Fig. 4.2 Diode Rectifier Curent Waveforms

(a) Line Voltage and Current

(b) Diode Current

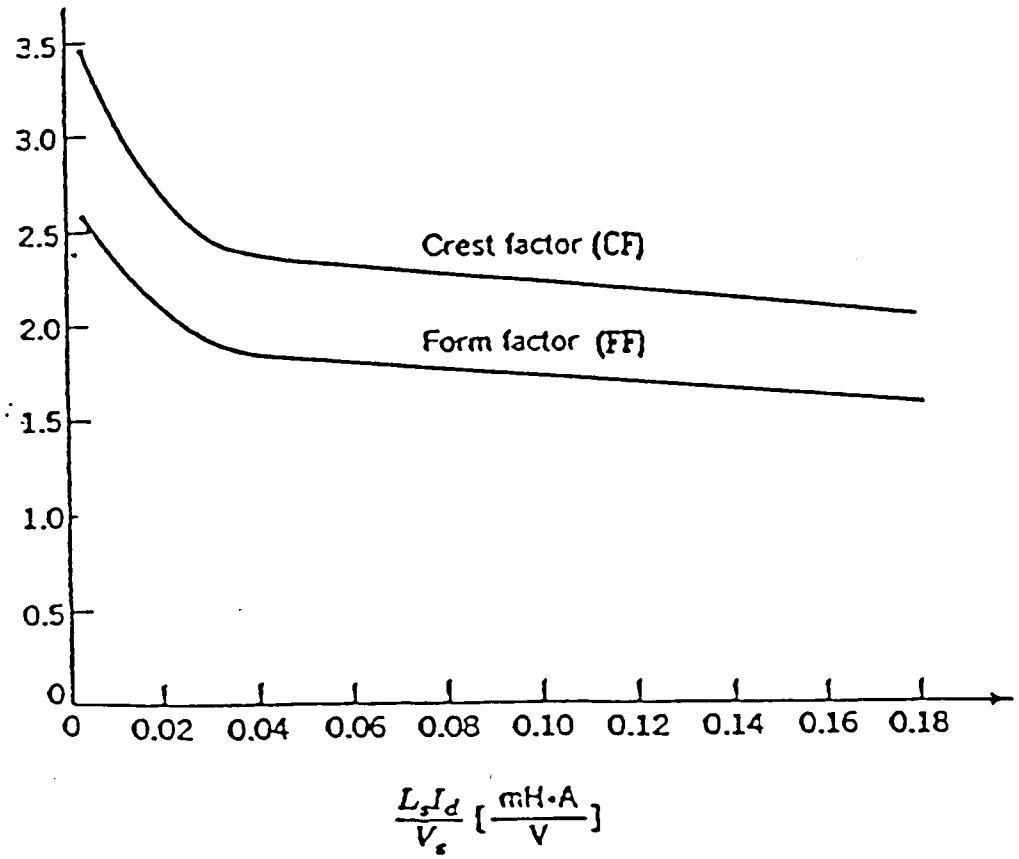


Fig. 4.3 Diode Crest Factor and Form Factor Curves

current. The corresponding expressions for three phase rectifier are,

$$I_D = \frac{2}{3} \cdot I_d \qquad \textbf{4.5}$$

$$I_s = \frac{1}{\sqrt{2}} \cdot FF \cdot I_d \qquad \textbf{4.6}$$

$$I_{Dr} = \frac{I_s}{\sqrt{3}} \qquad \textbf{4.7}$$

$$I_{Dp} = CF \cdot I_s \qquad \textbf{4.8}$$

For a given maximum I_d value, and knowing the L_s and V_s parameters, the FF and CF values can be determined from Fig. 4.3, and correspondingly the diode current ratings can be determined. For convenience of computer implementation, the FF and CF parameters can be expressed by linear segments as follows:

$$\begin{aligned} FF &= -30\,x + 2.7 && \text{for } 0 \le x < 0.02 \\ &= -12.5\,x + 2.35 && \text{for } 0.02 \le x < 0.04 \\ &= -1.786\,x + 1.92 && \text{for } x \ge 0.04 \end{aligned} \qquad \textbf{4.9}$$

and

$$\begin{aligned} CF &= -45\,x + 3.65 && \text{for } 0 \le x < 0.02 \\ &= -17.5\,x + 3.1 && \text{for } 0.02 \le x < 0.04 \\ &= -1.43\,x + 2.457 && \text{for } x \ge 0.04 \end{aligned} \qquad \textbf{4.10}$$

The peak inverse voltage (PIV) rating of the rectifier diodes is,

$$PIV = \sqrt{6} \cdot V_{smax} \qquad \textbf{4.11}$$

where V_{smax} is the maximum value of per-phase supply voltage.

4.2.2 DC Link Filter Design

It was mentioned earlier that the filter capacitor absorbs the ripple currents injected by the source side as well as the load side so as to decouple the output from the input. Although the filter bandwidth is the important parameter, the electrolytic capacitor is designed conservatively on the thermal limitation basis because of high equivalent series resistance (ESR). Therefore, the design criterion does not care about the distribution of the harmonics in the spectrum and the dc voltage ripple is maintained well within the limit.[38]

The source side current contains the dominant sixth harmonic ripple for the three-phase rectifier and second harmonic ripple for single phase. The analysis of source ripple is somewhat simple, but the inverter fed ripple current is very complex [37] because of the presence of Bessel function terms. Instead of theoretically analyzing this current wave, we decided to simulate a typical 5 hp drive and search for the worst case inverter input ripple current with maximum load current. Fig. 4.4 shows the inverter voltage and current waves in the worst case. It was found that the worst case ripple current in the load, absorbed by the filter capacitor is,

$$I_{CL} = 0.35 \cdot I_{L1} \qquad \textbf{4.12}$$

where I_{L1} = maximum rms fundamental load current.
Whereas the source side harmonic current absorbed by the filter capacitor can be given by,

$$I_{CS} = \sqrt{I_s^2 - I_d^2} \qquad \textbf{4.13}$$

where, I_{CS} = source-side harmonic ripple contribution to the filter capacitor rms current, I_S = rms source current and I_d = average dc link current.

Combining 4.12 and 4.13 gives us the filter capacitor rms current rating to be,

$$I_C = \sqrt{I_{CL}^2 + I_{CS}^2} \qquad \textbf{4.14}$$

The maximum capacitor voltage occurs at no load condition and is given as,

$$V_{Cm} = \sqrt{6} \cdot V_{smax} \qquad \textbf{4.15}$$

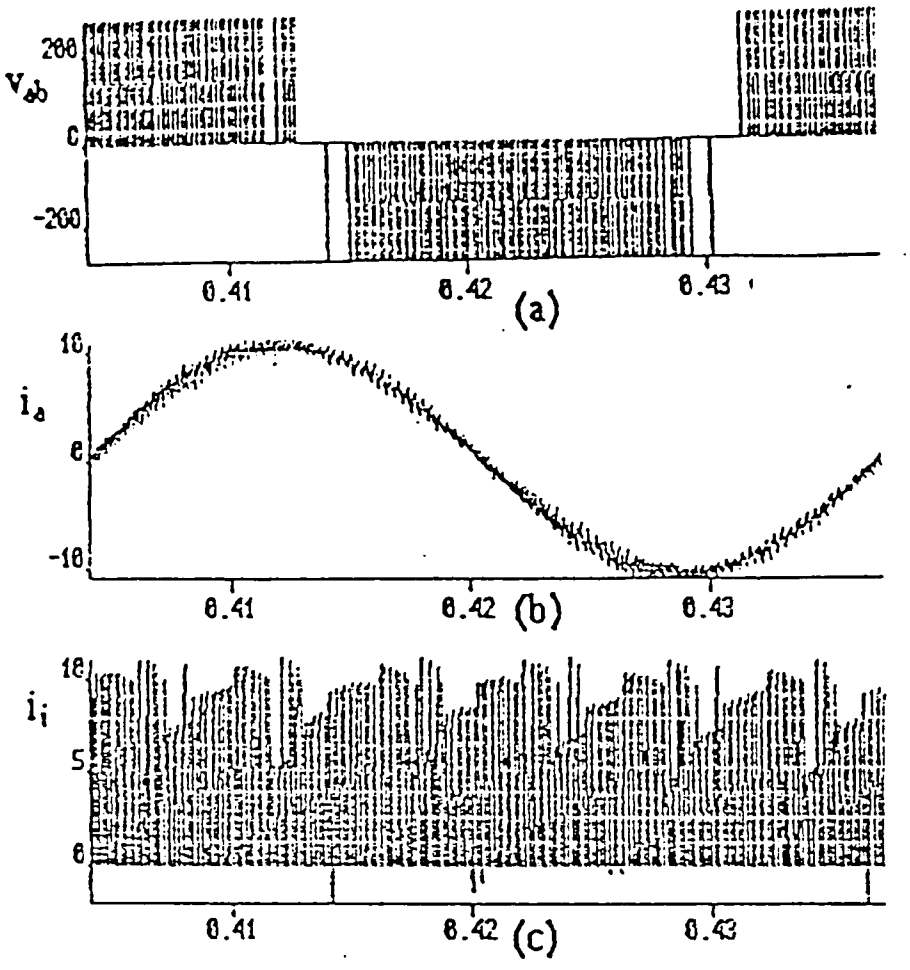


Fig. 4.4 Inverter Voltage and Current Waves at Worst-Case Capacitor Ripple Current

(a) Output Voltage

(b) Output Current

(c) Inverter Input Current

4.2.3 Inverter Circuit Design

The inverter circuit can be designed on the basis of the worst case loading and is described in the following sections:

4.2.3.1 Inverter Current Oversizing Factor

The acceleration / deceleration needs of the drive may demand more than rated torque from the machine. The large thermal time constant and pull-out torque capability of the machine easily permits this load demand. However, the inverter often needs oversizing to meet this demand because of short thermal time constant and limited continuous current rating (I_{dc}). The NEMA standard limited design, as mentioned before, will permit the inverter to carry 150% rated current for one minute during acceleration. The developed torque and the corresponding inverter current needed for acceleration / deceleration may need higher continuous rating of the inverter. If the load torque T_L is constant, then the required acceleration torque T_{mar} is given as, ([21],[22])

$$T_{mar} = \frac{W k^2 \cdot \Delta N}{308\ t_a} \qquad 4.16$$

where Wk^2 = load inertia in lb/ft^2, ΔN = rated speed in rpm and t_a = acceleration / deceleration time in seconds, needed for the load. Speed-dependent load torque requires solution of nonlinear differential equation [12]. If the motor rated torque is T_r, then with 150 % overloading of the inverter, the maximum acceleration / deceleration torque (T_{mai}) that the inverter can allow is T_{mai} = 1.5 T_r - T_L, where it is assumed that the torque is proportional to inverter current. Therefore, to satisfy the needed torque T_{mar} for acceleration, the continuous current rating has to be boosted by the oversizing factor given as,

$$IOS = \frac{T_{mar}}{T_{mai}} \qquad 4.17$$

If the motor maximum power rating P_{max} is known, however, from earlier motor design considerations, then the inverter short time current, (150% of the steady-state value), the inverter oversizing factor can be calculated as,

$$IOS = \frac{P_{max}}{1.5\,P_{inv}} \qquad \textbf{4.17a}$$

where P_{inv} is the inverter steady-state power rating.

4.2.3.2 Switching Device Rating

The voltage rating of the power transistor can be determined on the basis of worst case voltage impressed across the device. This can be found from the line voltage, dc link overvoltage due to regenerative braking and the transient voltage overshoot during turn-off of the device. The device current rating (I_{dc}) is again determined by considering the peak current flowing through the device [37]. This can be found from the current wave corresponding to inverter continuous power rating, oversizing factor (IOS), short time overcurrent rating and peak ripple current. Once the voltage and current capabilities of the device are determined, the nearest higher rating device can be selected. [37]. It is assumed that a matching bypass diode is built in the device module.

4.2.3.3 Turn-off Snubber Design

Snubber circuit ensures that the operating point of the power device always remains within the specified safe operating area (SOA). Fig. 4.5(a) shows the phase leg of an inverter with turn-off snubber. The turn-off snubber should be designed so that it delays the rise of v_{ce} till i_c is reduced to zero, limits the overshoot in v_{ce} due to L_l - C_s underdamped circuit, minimizes the total switching loss in BJT and snubber and limits the dv_{ce}/dt during turn-off to prevent faulty BJT turn-on. McMurray [34] has done a thorough analysis for the turn-off and turn-on snubbers used for BJTs for dc-dc converters. The same approach is extended here and is used for the design of the turn-off shunt snubbers for the BJTs in three-phase inverter circuit. The optimum snubber capacitance value, corresponding to the minimum total power loss in snubber and transistor during turn-off, is found using this approach for "small" snubber to be,

$$C_s = \frac{0.0988 \cdot I_C \cdot t_f}{V_d} \qquad \textbf{4.18}$$

where I_C = average transistor collector current, t_f = fall time rating of the transistor, and V_d = dc link

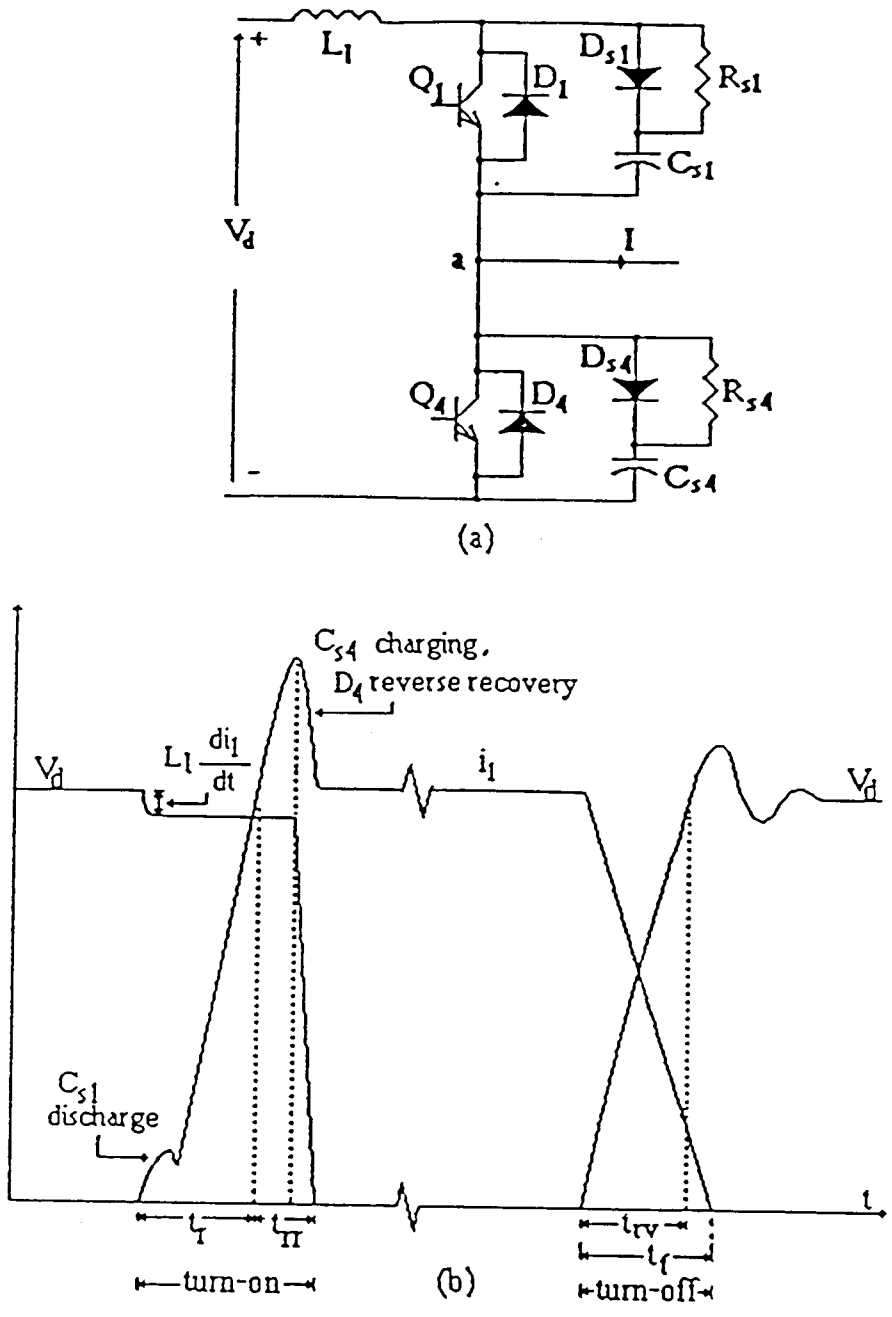


Fig. 4.5 Inverter Phase Leg and Associated Switching Voltage and Current Waves

(a) A Phase Leg of the Inverter
(b) Switching Voltage and Current Waves for Transistor Q_1

voltage. But choosing this small value of C_s causes large transient overshoot in v_{ce} during turn-off due to underdamped L_l-C_s circuit. So, a slightly higher value is chosen. The transistor switching losses are reduced further as a result, but the total losses in BJT and snubber increase slightly.

The snubber discharge time constant R_sC_s is kept much smaller than the minimum on-time of the transistor, to discharge C_s fully after each turn-on of transistor. The resistance R_s can thus be given by,

$$R_s = \frac{t_{ONmin}}{3 \cdot C_s} \qquad 4.19$$

where R_s = snubber resistance and t_{ONmin} = minimum ON time for the transistor. The stored energy in snubber capacitance is dissipated in R_s. Thus, its power rating is given by,

$$P_s = \frac{1}{2} C_s V_d^2 f_s \qquad 4.20$$

where P_s = snubber resistor power rating and f_s is the PWM inverter switching frequency.

The inductance L_l (generally stray leakage inductance) shown in Fig. 4.5(a) helps in preventing the turn-on inrush current and reduces the turn-on switching losses in the transistors.

4.2.4 Inverter Power Loss

The total power loss in the converter system consists of rectifier conduction losses in diodes, bypass diodes, and transistors, the switching losses in the power transistors during turn-on and turn-off as well as the snubber dissipation.

4.2.4.1 Conduction Losses

The total conduction loss in the system can be summarized as the conduction loss in the following:

- Diode rectifier
- Bypass diodes of inverter
- Inverter transistors

These can be found from the following equations:

For diode rectifier,

$$P_{cr} = 6\, V_D\, I_D \qquad 4.21$$

where V_D = forward conduction drop of diodes, I_D = diode average current and P_{cr} = total rectifier conduction loss.

For power transistors,

$$P_{ct} = 6\, \delta_{ct}\, V_{ce(sat)}\, I_C \qquad 4.22$$

where P_{ct} = total transistor conduction loss, δ_{ct} = average conduction duty cycle of transistors, $V_{cd(sat)}$ = average saturation voltage of transistors and I_C = average collector current of transistors.

For bypass diodes,

$$P_{cd} = 6\, (1 - \delta_{ct})\, V_{on}\, I_C \qquad 4.23$$

where P_{cd} = total conduction loss for bypass diodes and V_{on} = their forward conduction drop.

4.2.4.2 Switching Losses

Fig. 4.5(b) shows the typical turn-on and turn-off switching voltage and current waves. It is assumed that the load current I does not change during switching.

During turn-on, the transistor Q_1 carries part of the load current I which was initially flowing through diode D_4, the discharge current of snubber across Q_1, the recovery current through D_4 in the beginning and later the charging current of the lower snubber. The switching energy dissipation per pulse of output voltage is given by,

$$W_{s(on)} = 0.9\, I_C \left(V_d - \frac{L_l \cdot I_C}{t_r} \right) (0.3325\ t_r + 2\, t_{rr}) \qquad 4.24$$

where $W_{s(on)}$ = peak transistor turn-on energy per pulse, I_C = average transistor current, t_r = rated transistor current rise time, t_{rr} = rated bypass diode reverse recovery time and L_l = stray lead inductance in the inverter leg. The total turn-on loss in the transistor can be found out considering that there are actually $f_s/4$ lossy switchings of transistor power output voltage cycle, we get,

$$P_{s(on)} = \left(\frac{1}{\pi}\right) W_{s(on)} \left(\frac{f_s}{4}\right) \qquad 4.25$$

The turn-off loss is reduced by the presence of turn-off snubber as discussed earlier. The results of the analysis applied in [34] are used here, and the turn-off energy in the transistor per pulse is,

$$W_{s(off)} = \frac{1}{2} C_s V_d^2 f_s \qquad 4.26$$

where C_s is the snubber capacitance, V_d is the average dc link voltage, f_s is the switching frequency of the transistors. The total turn-off losses can be given as,

$$P_{s(off)} = \left(\frac{1}{\pi}\right) K_{off} W_{s(off)} \left(\frac{f_s}{4}\right) \qquad 4.27$$

where $P_{s(off)}$ = turn-off loss per transistor, K_{off} = the turn-off loss reduction factor, (dependent on the type of snubber, i.e., "small", "medium" or "large", according to McMurray's classification [34]), f_s = switching frequency.

The total power dissipation for converter, used for heat sink design, can now be given by equations 4.21, 4.22, 4.23, 4.25 and 4.27 as,

$$P = P_{ct} + P_{cr} + P_{cd} + 6\,(P_{s(off)} + P_{s(on)}) \qquad 4.28$$

4.2.5 Thermal Design

Once the power device is selected, it is necessary to design a suitable cooling system so that for a specified maximum ambient temperature, the device junction temperature remains within the safe limit ($T_{jmax} < 150$ °C*. The device modules are normally mounted on a heat sink which can be cooled either by natural convection or by fan [37]. Fig. 4.6 (a) shows a simplified thermal equivalent circuit representing the losses in diode rectifier, inverter transistors and bypass diodes, with associated thermal resistances and capacitances.

The junction temperature profile of a device has two types of ripples. The switching frequency ripple, which is negligible due to the filtering effect of the junction capacitances, and the

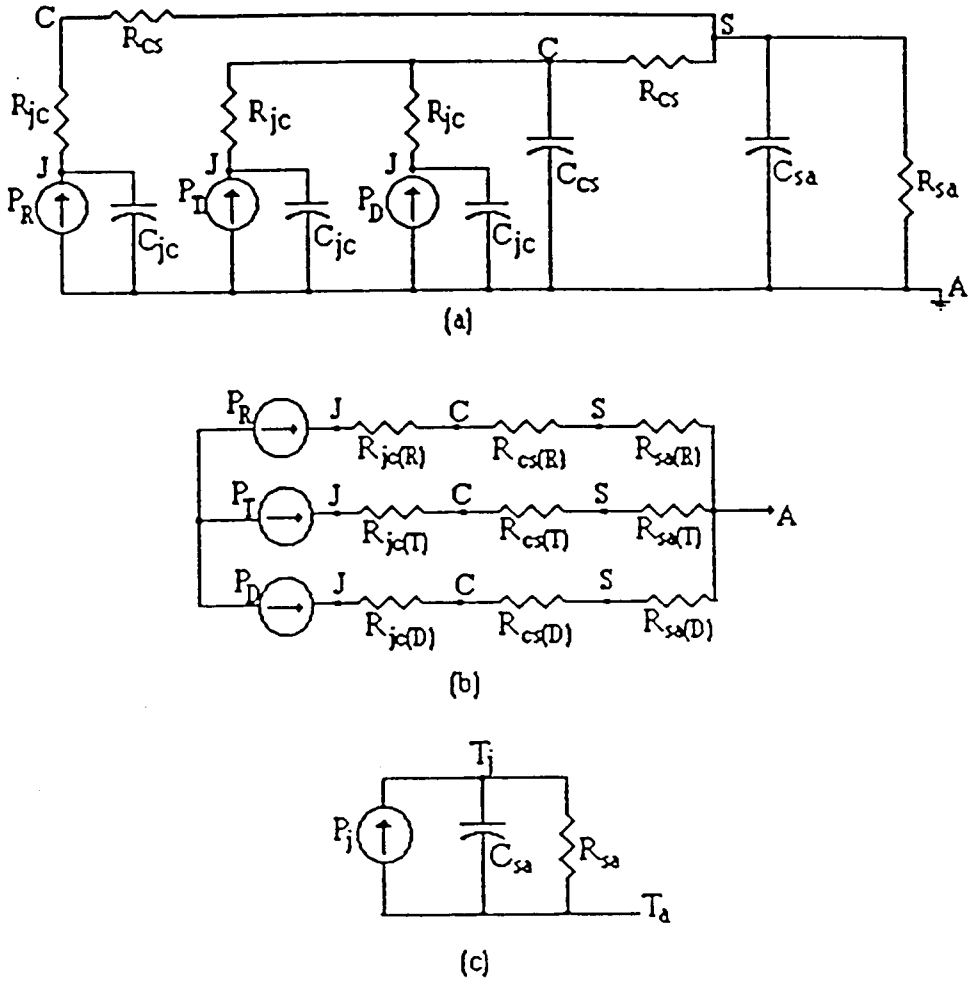


Fig. 4.6 Thermal Equivalent Circuit for the Power Converter System

(a) Distributed Thermal Equivalent Circuit of the Converter System
(b) Simplified Equivalent Circuit for Steady-State R_{SA} Calculation
(c) Transient Thermal Equivalent Circuit

fundamental frequency ripple, which is considerable. (But the temperature rise is found using the average value). Usually, the heat sink thermal resistance R_{SA} is calculated for steady state power dissipation, based on continuous overload rating of the inverter. The thermal equivalent circuit shown in Fig. 4.6(b) is used to calculate the contribution of R_{SA} due to the losses taking place in the rectifier diodes, inverter transistors and bypass diodes. For example, $R_{SA(R)}$ is given by,

$$R_{SA(R)} = \frac{T_j - T_a}{P_{cr}} - (R_{jc} + R_{cs}) \qquad 4.29$$

where $R_{SA(R)}$ = the thermal resistance contribution to R_{SA} due to the power dissipation in rectifier diodes. T_j = junction temperature, T_a = worst-case ambient temperature, P_{cr} = steady state power dissipation in rectifier diodes, R_{jc} = junction-to-case thermal resistance of rectifier diodes and R_{cs} = rectifier case-to-sink thermal resistance. The total R_{SA} of the heat sink can be calculated as,

$$R_{SA} = \frac{R_{SA(R)} R_{SA(D)} R_{SA(T)}}{R_{SA(R)} R_{SA(T)} + R_{SA(T)} R_{SA(D)} + R_{SA(R)} R_{SA(D)}} \qquad 4.30$$

where $R_{SA(T)}$, $R_{SA(R)}$ and $R_{SA(D)}$ are the contributions to R_{SA} due to the power dissipations in transistors, rectifier diodes and bypass diodes in that order.

The heat sink thermal capacitance C_{SA} is important in designing short-time overload rating of the inverter. Due to large value of C_{SA}, the heat sink delays the junction temperature rise, when it is subjected to transient heat dissipation resulting from short-time overloading (usually 150 % for 60 secs) of the inverter power circuit. The thermal equivalent circuit of the heat sink during this period, shown in Fig. 4.6 (c) allows us to find out the heat sink thermal capacitance to be,

$$C_{SA} = \frac{60}{R_{SA} \ln \left[\frac{0.4 P_j R_{SA}}{0.4 P_j R_{SA} - (T_{jm} - T_j)} \right]} \qquad 4.31$$

where P_j = power dissipation in power converter, T_{jm} = junction temperature limit during short time overloading, and T_j = junction temperature limit during steady state operation. (Usually, $T_j < T_{jm} < T_{jmax} = 150\ ^\circ C$)

4.3 Knowledge Representation for Power Converter System Design

As explained in earlier chapter, the knowledge representation can be defined as the task of casting expert knowledge within the computer in a way such that inferences or conclusions can be drawn. It deals with presenting the knowledge effectively and implementing the inferencing efficiently. The expert knowledge can be categorized as declarative knowledge and procedural knowledge. The declarative or fact-like knowledge deals with structuring domain knowledge acquired for designing the power converter system and is described by facts and their relationships. This is represented by frames, parameters and rules in this expert system. The rules have basically IF....THEN type of structure as shown in the sample rule in the abbreviated rule language (ARL) used here. This rule calculates the value of continuous maximum current rating (I_{dc}) of the power transistor.

IF: OUTPUT_POWER IS KNOWN AND OUTPUT_VOLTAGE IS KNOWN AND
INVERTER_OVERLOAD IS KNOWN AND
INV_OVERSIZING_FACTOR IS KNOWN

THEN: I_DC = (7.733 * OUTPUT_POWER * INV_OVERSIZING_FACTOR * INVERTER_OVERLOAD) / OUTPUT_VOLT

The procedural or method-like knowledge is similar to an algorithmic structure, expressed as a sequence of commands or actions. This basically controls the inferencing process, while the expert system arrives at conclusions from the design data available. The flowchart describing the complete process of expert system based design is shown in Fig. 4.7. Complete design discussed in section 4.2 earlier is embedded in the expert system knowledge base. Thus, based on the user-fed steady state and transient loading conditions, the ratings of various power components are calculated, and using these appropriate power semiconductors are selected from the database. Selection of the type of type of power switch is based on either of the following two criteria:

- User preference to BJT / IGBT
- Load harmonic current / acoustic noise considerations

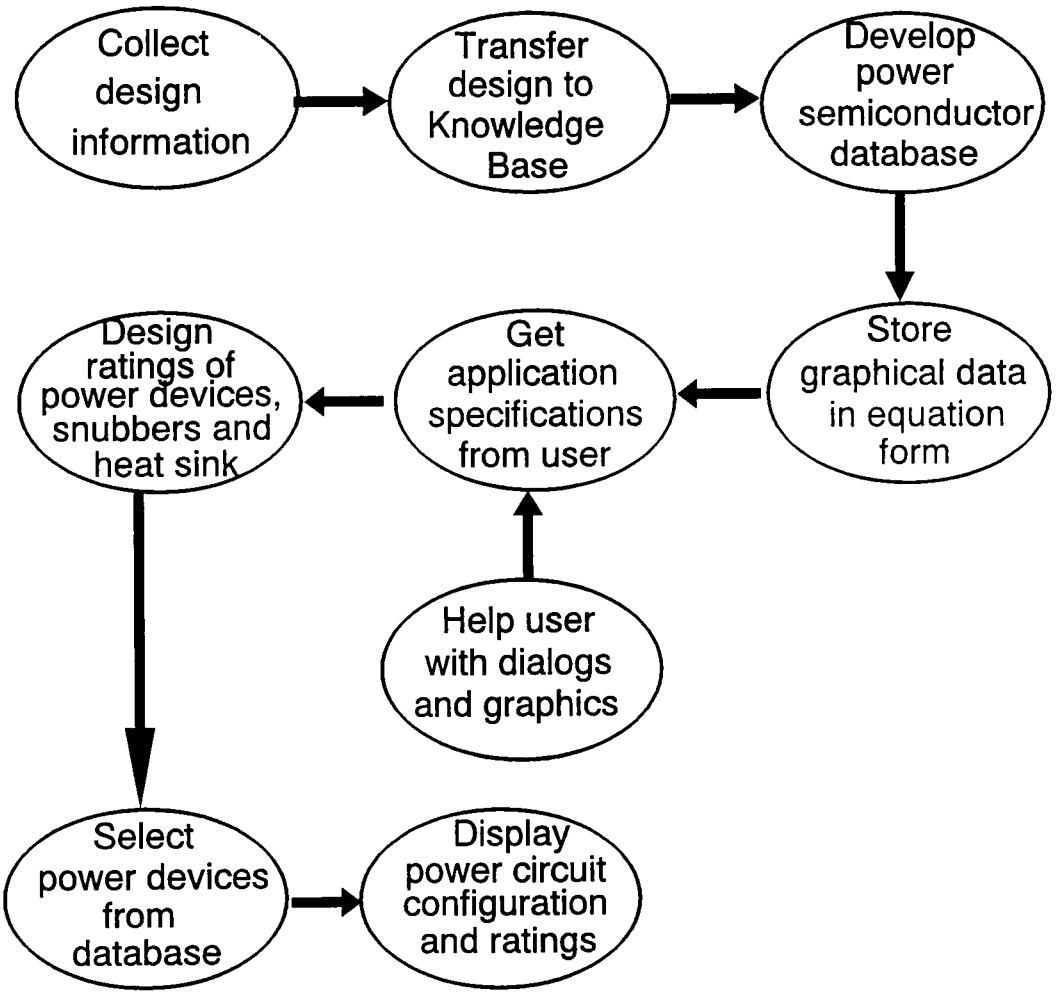


Fig. 4.7 Expert System Based Power Converter System Design Flowchart

4.3.1 Knowledge Base Development

The basic elements of the knowledge base in the expert system are frames, parameters, rules and functions. The frames represent a group or cluster of rules that perform certain function. Each frame is associated with a group of parameters, which represent bits of knowledge. The parameters are processed by a set of IF .... THEN type of rules. The hierarchy of frame structure allows functional partitioning of rule clusters, and only the rules belonging to the active frame are considered for instantiation at any given time, making the execution efficient.

The motor output power considered for the present design is in the medium power range, i.e., up to 200 kW. Only one frame, i.e., the root frame POWER-CONVERTER-DESIGN has been used for knowledge representation and it comprises of the associated parameters, rules, functions and variables, as shown in Fig. 4.8. The root frame contains also the initial data, such as inverter oversizing information, source inductance information and other data parameters, which are asked to the client at the beginning of consultation. The goals of the root frame are the quantities that are to be found, e.g., the power transistor type, rectifier diode type, snubber diode type, filter capacitor rms current rating, etc. As explained earlier, the inferencing is goal-driven and hence the rules are tried so as to set the parameter values affecting the goal values. When a certain parameter value cannot be set by any rule, expert system prompts for it to the user. Various rules in the root frame implement equations 4.1 through 4.31, in section 4.2. Also, the rules implement some warning messages for data inconsistency and conclusions and displaying graphic pictures during consultation. Rules also facilitate the interfacing of the expert system with external LISP functions as well as database files. When the above-mentioned actions are to be implemented, the rules cannot be expected to be fired during normal execution. Hence, they have to be specified to be "antecedent" or "forward-chaining" rules, which are data-driven. The following rule is an example of such rules:

IF: OUTPUT_POWER > 25 AND NO_OF_PHASES != THREE_PHASE
THEN: PRINT NAME " The output power level you specified requires three phase power supply, instead of single phase, as you have input. Please correct this input data, so that the consultation can continue"
ANTECEDENT: YES

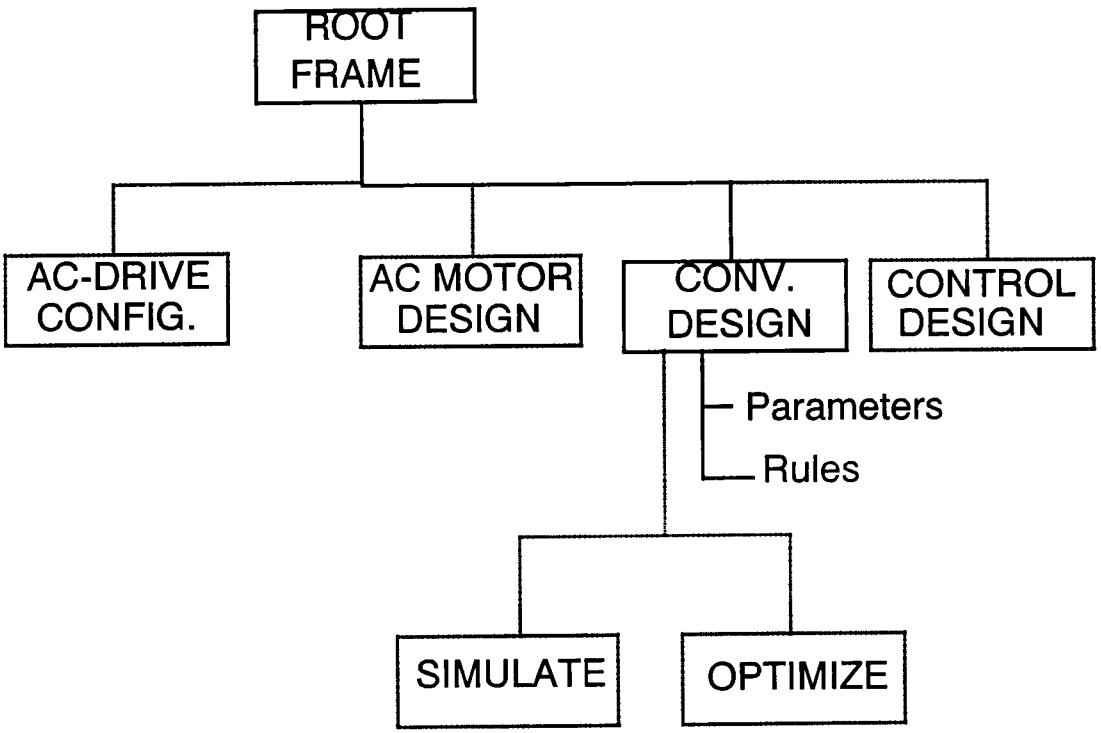


Fig. 4.8 Expert System Structure for Power Converter Design

As explained above, declaring this rule to be antecedent type is necessary as this rule does not help set the value of any parameter related to the goal parameter, and hence will not be tried by backward chaining during consultation. The programmer's task during the knowledge representation is usually to identify the key elements of the knowledge acquired and define them as parameters and goals, define rules that relate these parameters and the goals and then cluster these rules into functional subgroups in the form of rulegroups and then combine the rulegroups to form subframes and frames. This is the bottom-up design. The instantiation mechanism in general is fixed, inherent to the expert shell, but the programmer controls the logical structure in which the expert system should operate and instantiate various rulegroups and subframes.

Meta-knowledge, i.e., the knowledge about knowledge, usually derived from the user's experience with the consultation of expert system, provides the system with the understanding of knowledge as well as the ability to fill in missing information. In the expert system it is represented by meta-rules. It helps the expert system to acquire further procedural knowledge to work out answer apart from setting default values and controlling the instantiation of various rulegroups by assigning utility values to them.

As explained earlier, the expert shell has very limited math capability but can use all the scientific functions available in PC Scheme for its calculations. These are grouped under functions in the expert system knowledge base. One sample function calculating the value of filter capacitor rms current rating (Eqn. 4.15) is given below:

TRANSLATION:: Calculates the value of rms filter capacitor current. The function is called as follows:

```
        rms_filt_cap_current = (filt_cap_rms is_rms id_av il_fund_rms)
TEMPLATE:: (PARM PARM PARM)
TYPE:: EXPRESSION
SOURCE:: (LAMBDA (IS_RMS ID_AV IL_FUND_RMS)
            (SQRT (+ (- (EXPT (VAL1 FRAME IS_RMS) 2) (EXPT (VAL1 FRAME
              ID_AV) 2))
                (EXPT (* 0.35 (VAL1 FRAME IL_FUND_RMS)) 2 ))))
```

Here the Scheme functions SQRT (square root) and EXPT (exponentiation) are used. The source code is written in Scheme to implement the function. TRANSLATION, TEMPLATE, etc, are the compilation directives to the Scheme compiler, which compiles the source code finally for execution.

4.3.2 Development of Database

The data about the power semiconductor ratings are stored in an orderly manner in database files. Only one vendor, POWEREX Inc., has been considered for selection of power semiconductors, but more can be included later on. dBASE III+ [18] is used to create the database files for storing various ratings of power transistors, power diodes and three phase rectifier bridges. Each record in these files represents data about one particular device while each field in the record represents one particular rating of the device. Table 4.1 represents a sample record from the power transistor database. The database files can be accessed using various database interface functions in expert system rules anywhere in the knowledge base by rules using these functions. When encountered with the database access function, the expert system halts execution, loads the file specified, and applies the condition tests in the rule that called for database access. It then searches the first record that passes all the tests, as shown in Fig. 4.9. To avoid haphazard data retrieval, it is best to have the data files indexed for current and voltage ratings. Several such rules are used to retrieve various parameter values, as well as power semiconductor devices from the databases .

dBASE III+ is a very powerful tool to develop and manipulate databases with numerical and limited text data, but it has no means of storing data in graphical form. A different approach, therefore, is used to acquire, represent and use graphical data in the expert system.

4.3.3 Graphical Data Representation using TableCurve®

Often, the designer has to use the graphical data in the databook for design, as some of the data are not available in any other form. For example, for the current ratings ranging between 30 and 59 amperes, and voltage ratings ranging between 600 and 1000 volts, the same power diode will be selected (60 A/1000V). But, depending on the actual values of the currents flowing in the diode, its forward conduction drop can vary between 0.8V to 2.5 V, which, when used for calculation of diode conduction losses, will result in too conservative a design. To avoid this, the graphical characteristics

Table 4.1

A Sample Record from Power Transistor Database File

PACKAGE	SINGLE
TYPE	KS524503
VCE_SUS	600
I_DC	30
H_FE	75
VCE_SAT	2.0
T_ON	0.0000015
T_STG	0.000012
T_FALL	0.000003
R_CS	0.5
R_JC_BJT	0.15
R_JC_D	2.0

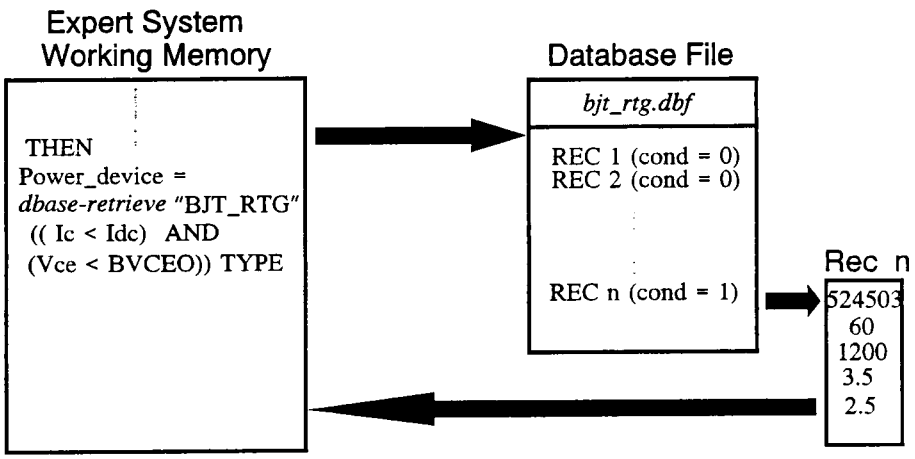


Fig. 4.9 PC Plus Interface With Database Files

can be employed by the expert system to calculate various parameter values for the power semiconductors, in the same way the designer would. The graphical data are usually available in the databooks, but have to be converted to both graphical and equation format for getting parameter values for any current value. This is done by subjecting the printed data in the databook to a number of conversion stages. An example converting $V_{CE(sat)}$ versus I_C characteristics of a power transistor from its printed data to graphical form and functional form is shown in Fig. 4.10. The three conversion stages can be described as follows:

- **Stage 1: Printed form → x-y table-**

The conversion from the printed data to x-y table can be done manually, but this will be approximate and time-consuming. Instead, the Tektronix Digitizer on mainframe VAX is used for faster and accurate conversion. A number of points spanning full range of I_C are entered from the characteristic curve and the digitizer converts them to their respective (x,y) coordinate pairs.

- **Stage 2: x-y table → Graphic image + Best-fitting curve equation-**

Now, the software TableCurve is used for converting this set of (x, y) coordinate pairs to the equation of the curve that best fits them. A number of statistical tests are performed and the equation of the curve best-fitting this set of points is found out, in the form of $V_{CE(sat)} = f(I_C)$. TableCurve also displays the points along with the bets-fitting curve for showing the best fit. Now we have both the picture and equation forms available from TableCurve for the printed graphical data in the databook. But these are still not usable by the expert system, and hence a third conversion stage is required.

- **Stage 3: TableCurve format → expert system compatible format-**

The GRAB utility from DrHalo graphic editor is used to convert the graphic picture in a graphic format (.PIC) compatible with the expert system, which then uses its SNAPSHOT utility (explained in section 4.3.4 next) to compress the graphical image to display it during consultation with the expert system. The equation form of the graphical characteristic is coded using abbreviated rule language (ARL) or Scheme or any other language, depending on complexity of calculation involved. This function is later called during consultation to calculate the $v_{CE(sat)}$ value.

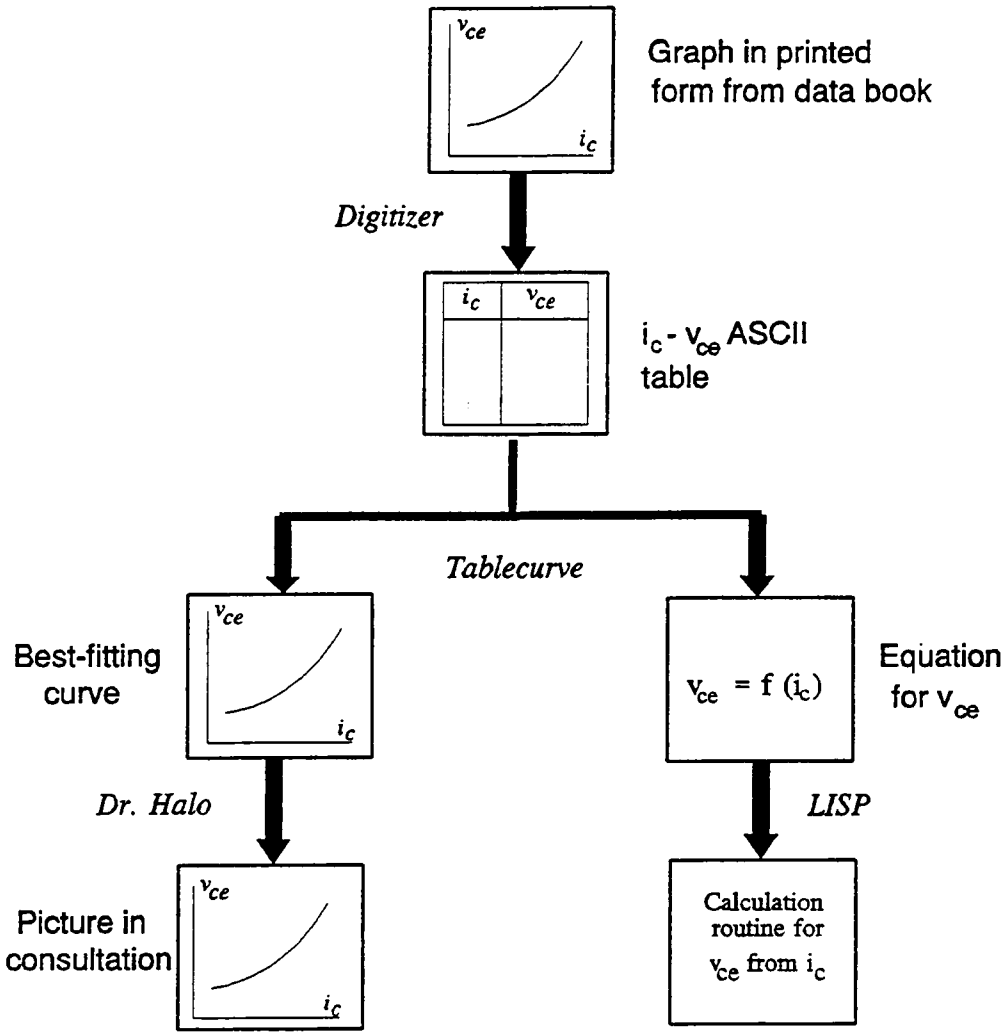


Fig. 4.10 Conversion of Graphical Characteristics of Power Semiconductor to Picture and Equation Forms Using TableCurve

4.3.4 Graphics Interface With Expert System

Intelligent use of graphics during consultation makes the task of communicating with the user easier and more effective while describing the circuit configurations, some typical characteristics of the system being designed, etc. Thus, graphics is an integral part of any expert system. In the present system too, graphics files are being used as a part of user interface.

The expert system itself does not have the capability of creating graphics images, but it can interface with external graphics files, created using third-party graphic editors. These images can then be displayed during consultation with the user. Expert system has a compression and expansion tool loaded in the form of SNAPSHOT utility, which converts graphics files in .PIC format into .GRI files, usable by the expert system.

A picture describing certain graphical information is generated by any graphic editor, such as OrCAD, SIMNON, TableCurve, DrHalo, etc, and then GRAB utility of DrHalo is used to convert this picture into .PIC format. DrHalo can now be used to edit this picture if necessary (Fig. 4.11). The SNAPSHOT utility inside the expert system is later used to capture this image in compressed format for usage by the expert system. Any time during consultation, when the picture is called, by a function such as PICTURE, GPROMPT, GHELP, etc, the expansion tool inside the expert system displays the image stored in the file referenced with the graphic function.

4.4 User Interface Design

Development of very effective user interface is very important, as user interface has the following important functions to do:

- To make the communication between the expert system and user, who could be a semi-skilled person, more effective
- To educate the user
- To make the expert system more user-friendly, and hence more user-acceptable

The development tools for the expert system allow several ways to implement the user interface as

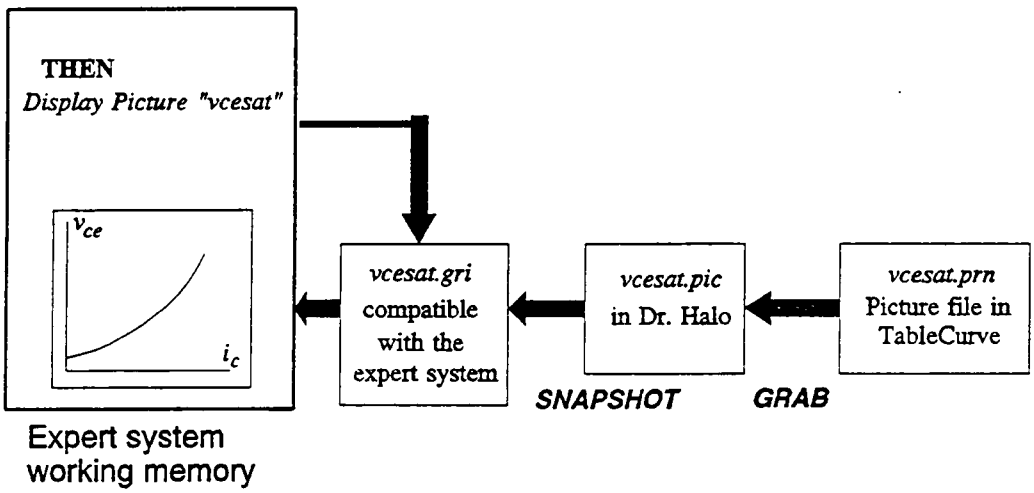


Fig. 4.11 PC Plus Interface With Graphics Files

highlighted in earlier chapter. All of these have been utilized while developing the expert system for design of power converter system. The dialog between the user and the expert system is conducted in plain English language, with the user being provided with precise questions, to obtain the data that the expert system needs for the design. The questions are also tailored according to user response. For example, in the beginning the user is asked information about the source inductance. Based on the user response, a suitable prompt is generated later to input value user indicated. This also avoids irrelevant questions being asked to the user, causing confusion. Sometimes, the semi-skilled user may be asked to furnish some value he cannot understand precisely. In such a case, user can avail of the on-line HELP facility, which can be in text form or in graphical form. In some other case, the user is asked a self-explanatory question, with the aid of graphic picture on the screen (Fig. 4.12). This eliminates ambiguities for the user in providing proper values to input parameters. Another method employed for making the consultation easier for the user is the provision of menus for selection of answers (Fig. 4.12). The availability of HOW command facilitates the mechanism for justifying the answers expert system provides in the end, and also, WHY command during consultation explains the user why a certain parameter value is necessary for consultation. This also gives the user insight into the inner workings of expert system, a part of the education process. The expert system execution, otherwise, is transparent to the user. The results displayed in the end are in the form of a graphical image showing the configuration of the power converter system, as well as values of the designed parameters, displayed in form of text message. This includes, as shown in Fig. 4.13, the selected power device types, such as the power transistor, rectifier diodes, snubber diodes, etc.

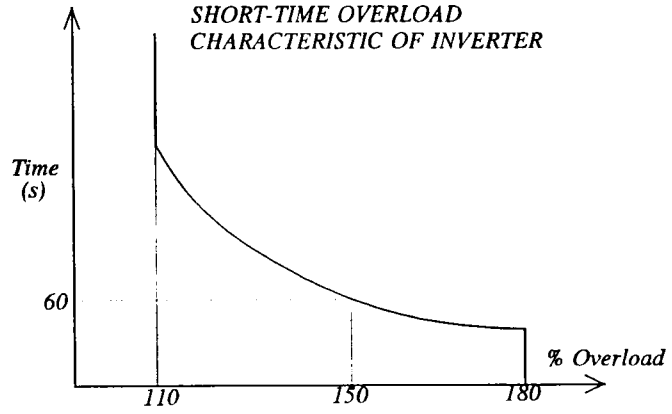
4.5 Summary

The chapter describes systematically the automated design technique of voltage-fed PWM inverter for induction motor drive using the expert system. All throughout the chapter, the emphasis has been on the design methodology of the expert system, rather than on the actual details of the expert system implementation. In the beginning, the power converter system design aspects that include the design of diode rectifier, filter capacitor, transistors, snubbers and heatsink have been reviewed because the knowledge base of the program is based on the power electronics knowledge. Both the BJTs and IGBTs have been considered for the design though the design of snubbers is primarily described with BJTs in mind. The knowledge base and database design have been described

EXPERT SYSTEM FOR AC DRIVE SYSTEM DESIGN

What is the short-time overload rating of the inverter ?

120 %
140 %
150 %
170 %



Select the value that applies
Press ENTER to continue

Fig. 4.12 User Interface Screen Showing Graphical Prompt of Inverter Short Time Overload Characteristics

EXPERT SYSTEM FOR AC DRIVE SYSTEM DESIGN

CONCLUSION SCREEN:

POWEREX Bipolar Junction Transistor selected	*: KS524503*
Bipolar Junction Transistor Module Type	*: Single Darlington*
Snubber Resistance	*: 10 Ω*
Snubber Resistance Power Rating	*: 26 W*
POWEREX Snubber Diode Selected	*: ME100603*
Snubber Capacitance	*: 0.15 μF*
Snubber Capacitor Voltage Rating	*: 500 V*
POWEREX Diode Bridge Rectifier Module Type	*: ME400402*
RMS Current Rating of Filter Capacitor	*: 43 A*
Voltage Rating of Filter Capacitor	*: 500 V*
Heat Sink Thermal Resistance	*: 0.78 °C/W*
Heat Sink Thermal Capacitance	*: 150 W-sec/°C*

Press ENTER to continue..

Fig. 4.13 Contents of Design Conclusion Screen

following the methodology description of representation of graphical data in picture and equation form using TableCurve curve-fitting software. Finally the user interface design is described. Intelligent prompts in the form of graphical pictures and text messages are designed to help explain the user the operation of the expert system based design. Also, default values are employed at strategic points during consultation, when user is not able to decide on a response. Although a design implementation with the rectifier-inverter system is shown, the same techniques can be extended to include other converter topologies, such as dual PWM converters, the resonant converters or the converter for brushless DC motor drives.

CHAPTER 5

EXPERT SYSTEM BASED AUTOMATED SIMULATION AND DESIGN OPTIMIZATION OF POWER CONVERTER FED AC DRIVE SYSTEM

5.1 Introduction

Over the years, the design of a power electronic converter system has always been the job of the expert designer. Based on the application specifications provided to him by the user, he designs the power converter, with the help of paper, pencil and calculator, using his domain knowledge and years of his experience. Occasionally, he also takes the help of some computer programs, but mainly the design is based on heuristics. However, once the design is complete, the engineer fabricates a prototype power converter system in the laboratory, for testing the design. He performs various tests on this prototype, and evaluates the performance of this breadboard prototype. The results of these tests are used to fine-tune the design further. This procedure, however, is expensive and time-consuming, as it requires the designer's expertise to design, breadboard, test and interpret the test results for refining the design.

The automated design approach has been applied successfully to analog and digital VLSI where the complete design - simulate - test iteration cycle is automated with the help of user-friendly CAD tools. In this chapter, the knowledge-based approach has been applied for CAD of power converter system. Here, the power converter designer's expertise has been embedded in the expert system, which provides a very powerful, flexible and user-friendly CAD tool in PC - based environment. The standalone run-time version can be easily used by a skilled or semi-skilled designer.

The chapter begins with a review of snubber design methodology because the converter design has

already been discussed. The bulk of the chapter then gives details of the developed interface between the expert system shell and the simulation program. The knowledge representation for the simulation based optimization of the power converter is then presented. The results, in the end, demonstrate the importance of the expert system-based approach. This work was originally published and presented [46] at the Industrial Electronics Society Annual Conference at Maui, Hawaii in November 1993 (IECON 93).

5.2 Converter System Design

In the beginning, the expert system makes a preliminary design of the converter system based on the expert design knowledge embedded in it. The candidate converter system selected here is a voltage-fed transistor inverter which feeds an induction motor and operates in open-loop volts/Hz control mode. The design considerations of the converter system and its expert system based design were described in chapter 4. The transistor turn-on and turn-off snubbers which have been optimized by the expert system will be reviewed here briefly. Fig. 5.1 shows an inverter phase leg operating as dc-dc converter with worst-case inverter load current. The circuit incorporates the standard turn-on and turn-off snubber elements. The leakage inductance L_l, contributed by the intentional routing of the busbars, controls the turn-on current profile, and is normally sufficient to control the turn-on switching loss for the low to medium power level inverters, which are the scope of design here. The design of turn-off snubber involves selection of proper resistance R_s and capacitance C_s. The resistance should be designed large enough to limit the device overshoot current within a safe value during turn-on, but again should be small enough so that the capacitor can discharge completely during the transistor on-interval. The design of C_s is a trade-off between the combined transistor and snubber losses and the voltage overshoot across the transistor during turn-off [44]. In this project, McMurray's mathematical model [34] for "small" snubber was used for design and simulation. Fig. 5.2 shows the transistor and capacitor energy loss distribution at turn-off as a function of capacitance C_s. It also indicates the device voltage overshot for various C_s values. It the capacitance value is chosen (point X) to yield the minimum total (transistor and snubber) loss, the voltage overshoot will be high. A higher C_s value results in lower voltage overshoot but increases the total loss, as indicated. This means that the efficiency of the inverter due to snubber can be improved at the cost of higher voltage rating of the devices, i.e., higher converter cost. During the preliminary design, the expert system selects the C_s value corresponding to the minimum total loss. Then the

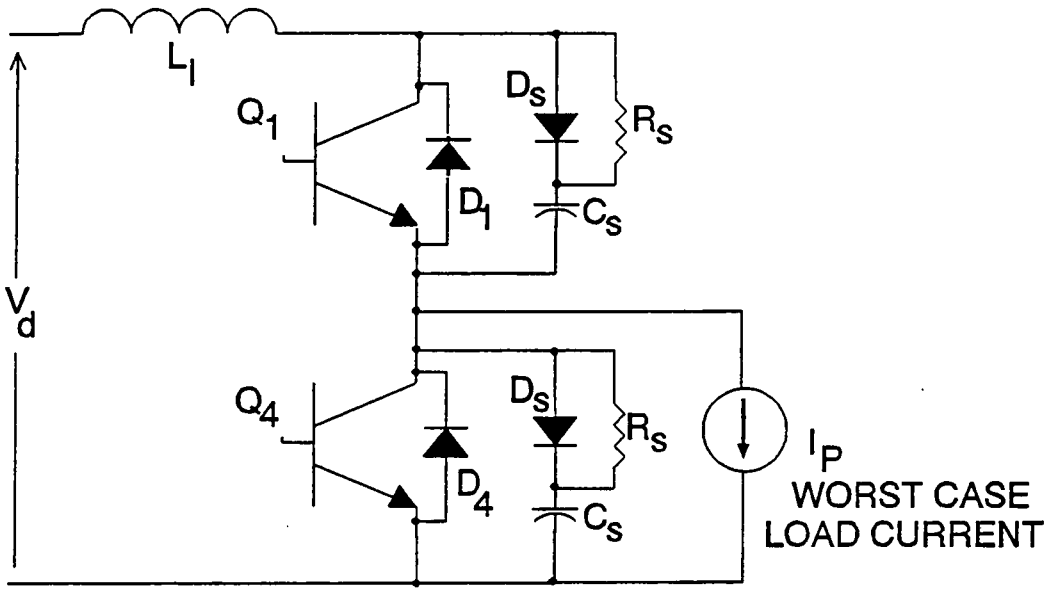


Fig. 5.1 Inverter Phase Leg Operating as DC-DC Converter

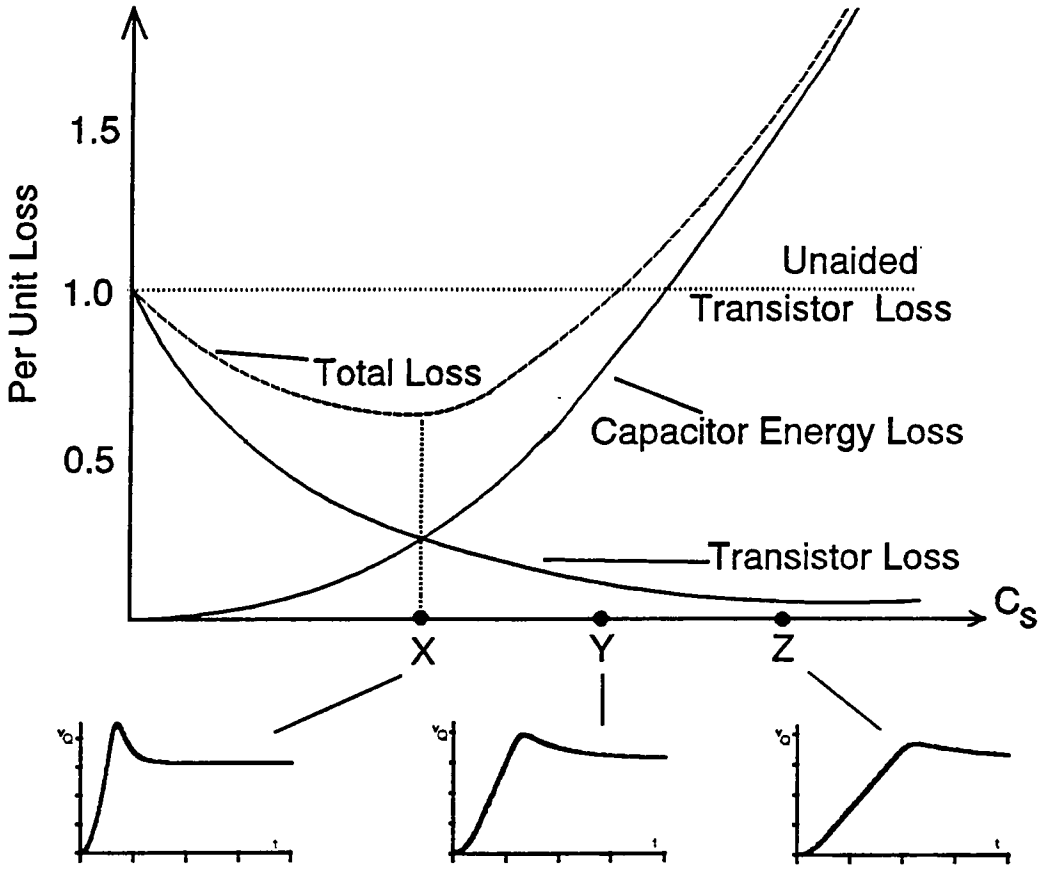


Fig. 5.2 Loss and Voltage Overshoot Curves for a Transistor at Switch-off with Turn-off Snubber

resistance R_s is designed so that the turn-on device current contributed by the snubber is a fraction of the worst-case load current.

5.3 Expert System Interface With System Simulation Program

The interface between the expert system and the simulation program can be better understood if we consider the procedure followed by the expert power electronics design engineer in the laboratory. Based on the user's specifications, he first designs the power converter system and selects suitable devices from the data books. Then, he builds the prototype of the designed system on a breadboard. In the case of the power converter system, for example, a three-phase PWM voltage-fed inverter is constructed in the lab. This inverter is now tested for various worst-case conditions, such as, currents in the transistors, voltages across the dc link, etc. The currents and voltages obtained through experimentation are then compared with their designed values, and if there are any discrepancies between the two, which would make the design inadequate for the loading or switching conditions, the design is modified to accommodate them. This is the design refinement based on laboratory experimentation.

The expert system approach used here mimics the same process as the one described above, but with the software breadboard, rather than actual hardware breadboard. Thus, there are no costly and hazardous component burnouts in this case. Moreover, many what-if scenarios can be tried on the software, before deciding on the best design in the end. As explained earlier, the expert system is best suited for the tasks involving symbolic processing, decision making and limited amount of calculations. However, PC Plus based expert system has the ability to interface with external programs, such as databases, procedural programs such as simulation files in PC-SIMNON, interface routines in C and graphics interfaces. Of particular interest here is the expert system's interface with DOS-based procedural programs, which is utilized extensively while the expert system communicates with the system simulation program, PC-SIMNON.

5.3.1 Generalized Methodology of Expert System Interface With External DOS Programs

The process of expert system's communication with external DOS program, which is in the form of a command or executable file is depicted in Fig. 5.3. PC Plus allows the use of its external interface

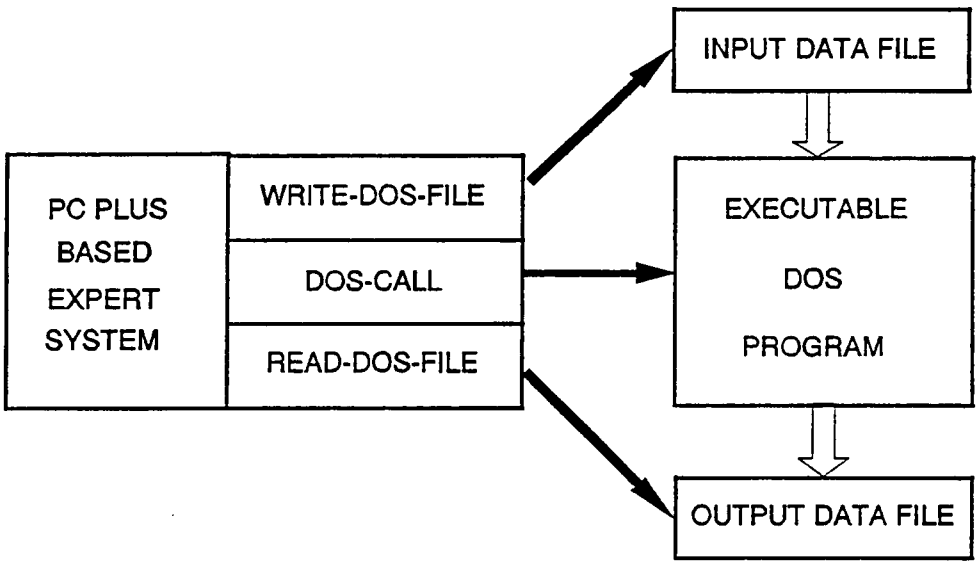


Fig. 5.3 PC Plus Expert System General Communication Protocol with a DOS File

functions that write or read from an external file, as well as call and execute an external DOS program. The process begins when the expert system exports the data for the DOS-based program in its input data file, using the command WRITE-DOS-FILE. Next, the expert system calls the DOS-based program using the function DOS-CALL. The DOS-based program reads the data from the input data file exported by the expert system earlier, and writes its results after creating its own output data file. These results are then read by the expert system when it uses the command READ-DOS-FILE. The expert system automatically performs the tasks such as opening and closing files, reading and writing data, and allocating memory for loading the DOS-based program, as well as calling and executing the program. So, the user need not worry about them.

5.3.2 Expert System Interface with PC-SIMNON Simulation Program

As explained earlier, the expert system can design the converter, and given the results of the simulation of the converter system, can optimize the design. However, it cannot model or simulate the converter system by itself. The system simulation programs, such as PC-SIMNON, Simulink, etc, on the other hand, can model and simulate a power converter system, but cannot interpret the results to produce the optimization. Thus, a combination of the expert system and the system simulation program has to be employed to achieve the purpose of simulation-iterated optimization of the power converter system.

The expert system hence performs the design of the power converter system and communicates with the external system simulation program, PC-SIMNON. In doing so, the expert system controls the simulation, and interprets the results of the simulation. It then uses the results of simulation to modify the design of the power converter and optimizes its performance. All of the above tasks are performed by the expert system with the help of C language based interface routines. For example, the simulation of the effect of turn-off snubber on the switching behavior of one transistor leg is conducted by the expert system in the steps outlined below:

- *The expert system exports the designed parameter data values to an external file*

In order to simulate one transistor leg with associated snubbers, (as shown in Fig. 5.1), the expert system exports the results of snubber design to the parameter data file, as shown in Fig. 5.4. These values correspond to the supply voltage of the inverter, the worst case load current flowing through

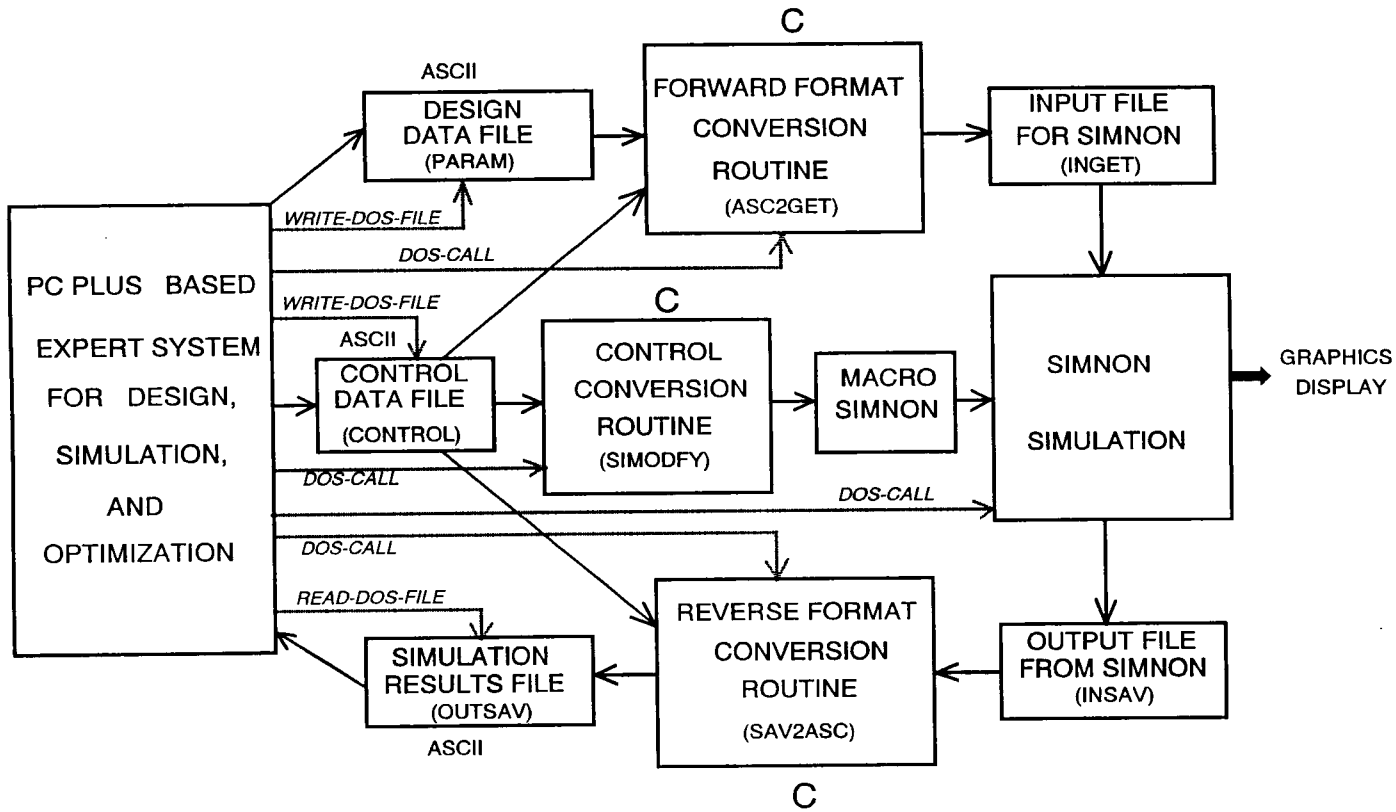


Fig. 5.4 PC Plus Based Expert System Interface With SIMNON Simulation Program Showing all the Interface Routines

the transistors, the transistor rise and fall times, the snubber resistance and capacitance values and the lead leakage inductance value. The format of this file is shown below:

220 20.9 1.19e-6 1.73e-6 0.15e-6 1.0e-6

- *The expert system exports the control data values to another external file*

Now, the expert system indicates that transistor HALF-BRIDGE is being simulated for THREE-PHASE inverter system, by outputting these two keywords to another file, called the control data file. (Fig. 5.4). This file, as will be seen later, is used by all the interface routines for different purposes. The contents of this file are shown below:

THREE-PHASE HALF-BRIDGE

- *The expert system invokes the forward format conversion routine*

The values supplied by the expert system to the parameter data file are meant for the system simulation program. However, the simulation program, PC-SIMNON[42] in this case, needs to have the parameters preceded by appropriate identifying labels showing the SIMNON model file, (ONOFF in this case) they belong to, and the parameter names they are assigned to. Since the expert system cannot provide this extra information, it invokes the C-based data format conversion routine. This forward format conversion routine reads the control data from the control data file, and immediately correlates the values in the parameter data file with their corresponding labels, opens the input data file for PC-SIMNON, and writes the labels along with their parameter values. Now, the design data values are in the format PC-SIMNON understands, and will be able to read, when directed by the simulation command file. The format-converted file now looks like,

[ONOFF]
Vs: **220**
I: **22.9**
tr: **1.19e-6**
tf: **1.73e-6**
Rs: **25.0**
Cs: **0.15e-6**
Ll: **1.0e-6**

- *The expert system generates simulation command file, invoking control conversion routine*

The expert system completely automates the simulation process, based on the user's choice of system

to be simulated, by generating the simulation command file for PC-SIMNON. This command file, or *macro*, is generated when the expert system invokes the control conversion routine, which is C-based[43]. This routine reads the information on the system to be simulated from the control data file, and automatically produces a string of PC-SIMNON simulation commands, which include commands for reading and writing data from input / output data files, simulation start and stop times, the system to be simulated, variables to be stored and displayed graphically.

- Expert system invokes the simulation program, PC-SIMNON

Having done the groundwork for invoking the simulation program, the expert system now invokes it, and executes the command file that it generated in the previous step. Now, the simulation command performs the simulation of the converter half-bridge after reading the data from the input data file. At the end of the simulation, the program not only displays the responses of voltages and currents, but also exports their critical values, (rise time, overshoot, etc.) to its output data file, which has essentially the same format as its input data file.

- Expert system now invokes the reverse data format conversion routine

The expert system now executes the reverse data format conversion routine, which acts in a manner exactly opposite to the forward data format conversion routine, and strips the simulation output data file of all its labels. It reads the control data file, to know which labels to remove. The format of the data file generated as a result is the same as the format of the design data file generated by the expert system, in the first step above.

- Expert system reads the simulation results

Since the simulation results are now available to the expert system in its compatible format, it can read them. It can use the results in a variety of ways. For example, if the snubber circuit is being optimized, the expert system is looking for the voltage or current overshoot in the transistor, so that it can modify the value of snubber capacitance or resistance (respectively), to get them a step closer to their optimum values. On the other hand, if the idea is just to verify the operation, the expert system displays the values of the parameters.

5.4 Knowledge Representation

The knowledge representation for converter simulation and optimization involves combining diverse types of knowledges, namely, the expert's know-how to interpret the simulation results to optimize the design essentially is based on heuristic rules, which signify "shallow" reasoning, which is also declarative. However, the knowledge about conducting the simulation as well as the simulation itself is highly procedural, as well as model-based and "deep".

5.4.1 Knowledge Base Structure

Fig. 5.5 shows the structure of the expert system presented in this chapter. The domain expertise of the design engineer in designing, breadboard testing and optimizing the system design is embedded in this expert system. The expert knowledge is stored in the expert system in the form of frames, parameters and rules. The three frames of the expert system achieve all the three major tasks mentioned above, i.e., design, simulation and optimization. The root frame performs the design of the converter system, based on user specs. Thus, it contains the parameters and rules pertaining to the design of the converter system. The expert know-how on simulation of the designed converter-machine system is embedded in the subframe 1, called SIMULATE. The subframe-2 is named OPTIMIZE and has the expert knowledge about optimizing the polarized snubber for transistor. The knowledge represented in these frames thus helps the expert system to achieve the task of designing and simulating the complete system, and optimizing the polarized snubber with the help of PC-SIMNON based simulation.

5.4.2 Expert System for Simulation and Optimization

During a typical consultation with the client, the expert system first obtains the critical data and performance specs of the converter system, and then designs the system. At the end of the design, it asks the user to choose independently between simulation of the designed system or optimization of the polarized snubber. The expert system then involves the subframe SIMULATE or OPTIMIZE based on user's choice. The rule activating subframe for simulation is shown below:

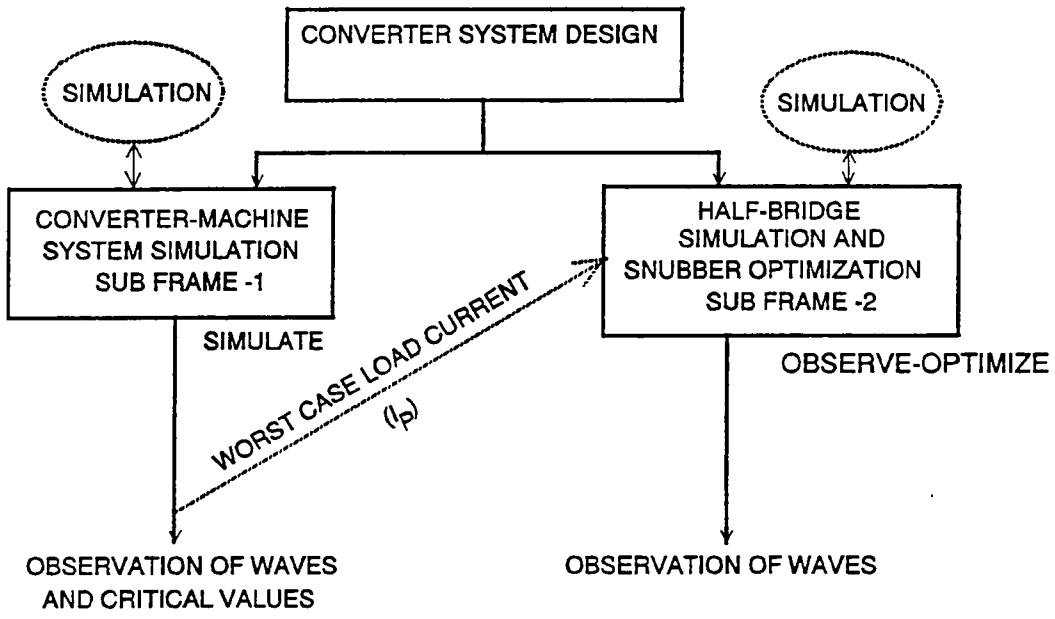


Fig. 5.5 Structure of Expert System Knowledge Base

IF : DESIGN-DONE AND USER-CHOICE = SIMULATION

THEN: CONSIDERFRAME SIMULATE

If the user chooses the simulation of the designed system, the expert system first outputs the results of the design and then invokes the simulation in PC-SIMNON environment. At the end of the simulation, the expert system directs PC-SIMNON to display the parameters, i.e., voltages and currents (of user's interest) graphically on screen. This is analogous to laboratory observation of the system performance by connecting the oscilloscope for display of key waveforms. At the end of the design, the expert system presents the user with the ratings of the power devices chosen. Thus, the user can compare these ratings with the worst case peak voltages and currents observed from the graphically displayed parameters. This helps the user in verifying the safety of the designed system. It should be noted here that while performing the simulation of the complete converter-machine system, the power devices are assumed to be ideal switches in order to save simulation time but without any loss of accuracy of the results.

If, on the other hand, the user chooses to optimize the snubber, the expert system invokes the subframe OPTIMIZE. The optimization of the polarized snubber involves selection of optimum values of R_s and C_s. The snubbers have to be designed for the worst case basis before any optimization can be attempted. Hence, the expert system first obtains the value of the worst case load current flowing in the transistors. It invokes PC-SIMNON simulation for the complete converter system, and directs it to supply the peak current value. The following rule achieves simulation and parameter passing:

IF : USER-CHOICE = OPTIMIZATION
AND CURRENT IS NOTKNOWN

THEN: MODE = COMPLETE-SYSTEM AND
WRITE-DOS-FILE "PARAM.DAT" ELECT-RAD/S INPUT-VOLT
SOURCE_L FILT_CAP T_RISE SNUB_R SNUB_C LEAD_L T_FALL
SWITCH-FREQ AND
WRITE-DOS-FILE "CONTROL.DAT" MODE NO-OF-PHASES AND
DOS-CALL "ASC2GET.EXE" AND
DOS-CALL "SIMODFY.EXE"AND
DOS-CALL "SIMNON.EXE" AND
DOS-CALL "SAV2ASC.EXE"AND
READ-DOS-FILE "OUTSAV.DAT" IPEAK

For the rest of the optimization process, the expert system conducts the simulation in the half-bridge mode. As explained earlier, the transistor half-bridge is operated in dc-dc converter mode, with worst-case load current and at 20 kHz switching frequency instead of at frequency of 5 kHz. This is because only the transient behavior near switching instants is of importance here, while observing the voltage or current overshoots during turn-off or turn-on interval. So, a higher switching frequency saves the simulation time.

The expert system now proceeds for snubber resistance optimization. As shown in Fig. 5.6, it feeds the preliminary design value of R_s to PC-SIMNON. It then simulates the half bridge for one switching cycle, and observes the peak overshoot current in transistor during turn-on. If this value is outside the tolerance band of 110% to 130% of worst case load current, the expert system notes the magnitude of this excursion, and its direction (beyond 130% or below 110%), and modifies the value of R_s by a small amount. The sample rule shown below increases the value of the snubber resistance by 5 ohms when the value of the current overshoot is more than 130%.

IF : CURRENT-OVERSHOOT > 130 AND ! OPT-DONE
THEN: SNUB-R = SNUB-R + 5.0

This process of modification of R_s and half bridge simulation is iterated till the expert system finds the current overshoot during turn-on within the tolerance band. The final modified value of R_s is not the optimized value.

Once the optimized R_s value is known, the expert system proceeds for the optimization of the snubber capacitance C_s. It starts once again with the worst case load current value, optimized value of R_s and designed C_s value. Optimization process for C_s closely resembles the optimization of R_s. The only difference is that the voltage overshoot for the transistor during turn-off is compared against 110% to 120% of dc link voltage, while C_s is varied. The optimum C_s value corresponds to the transistor voltage overshoot at turn-off falling within the tolerance band.

When both the snubber resistance and capacitance values are optimized, the expert system displays the results graphically in the forms of transistor voltage and current waveforms and numerically in the form of final values of voltage and current overshoots.

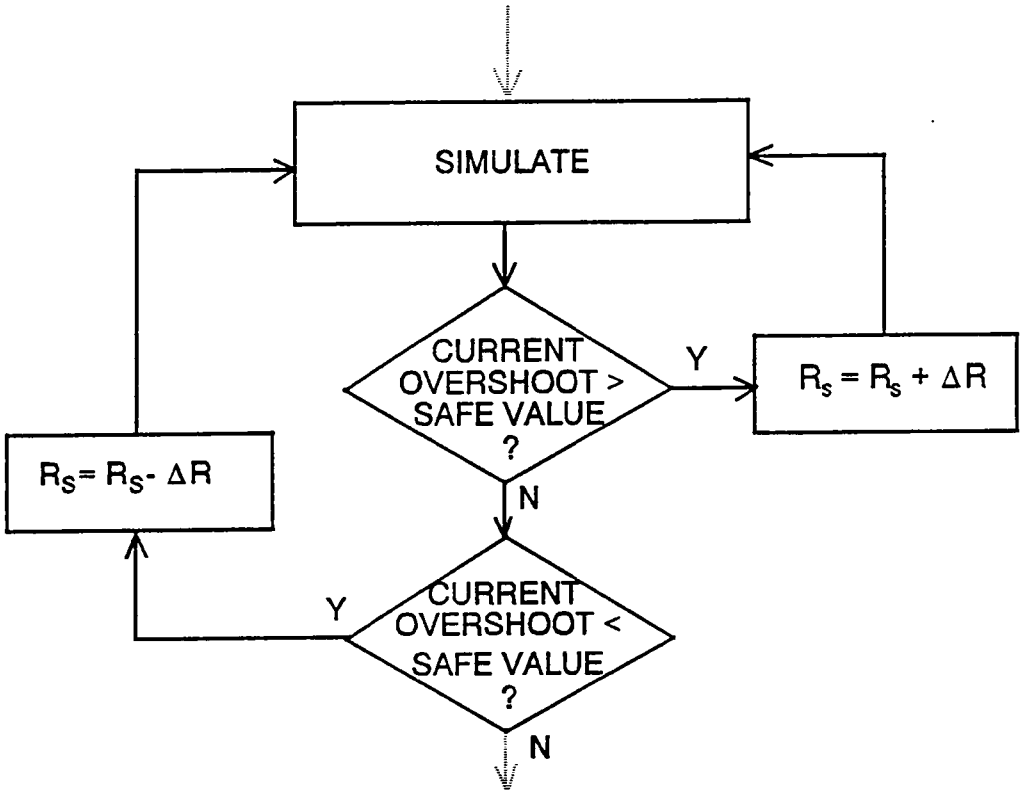


Fig. 5.6 Snubber Resistance Optimization Algorithm

5.5 System Performance

As mentioned earlier, the expert system mainly performs two major tasks according to user's choice: simulation of complete converter-machine system for steady-state performance observation and simulation of half-bridge dc-dc converter for optimization of snubber parameters. The second choice, however, automatically invokes the first choice to determine the worst case load current. Fig. 5.7 shows typical phase voltage and current waves for simulation of the complete system. Fig. 5.8 shows the voltage and current waves of transistor at turn-on during multiple passes of R_s optimization. The current tolerance band used for the design varies between 120% and 130% of the worst case load current of the inverter. The expert system also verifies that the 130% of the worst case load current lies within 70% of the I_{Cmax} rating of the transistor for safe operation. Note that the turn-on current transient occurs within 1% of the transistor duty cycle, and therefore, transistor peak current is limited within the safe limit. Fig. 5.9 shows the transistor voltage and current waves at turn-off switching during multiple passes of C_s optimization. As shown in the figure, the C_s value is increased until the voltage overshoot falls within the desired range of 110% to 120% of the dc bus voltage. Obviously, this results in lower device switching loss but higher snubber dissipation. The optimization procedure for snubber design illustrated here can be extended to more complex designs and optimizations.

5.6 Summary

The chapter describes the extension of the expert system for design of voltage fed PWM fed induction motor drive system to its simulation and design optimization. The bulk of the chapter deals with explaining the methodology of the expert system interface with the system simulation program, PC-SIMNON. The methodology used here is generalized enough to be used with any other types of softwares. In fact, the simulation and interface softwares are designed in a manner that allows them to be used as standalone, if the user chooses to supercede the expert system in decision making. Throughout the chapter, the emphasis has been on the methodology rather than the actual implementation. Although the polarized snubber of the inverter was chosen to demonstrate the expert system's capability to optimize, similar concept can be easily extended to more complex circuit or system optimization. Similarly, the converter design and simulation can be extended also for other devices, such as IGBT, GTO and power MOSFET. The principal accomplishment, as said

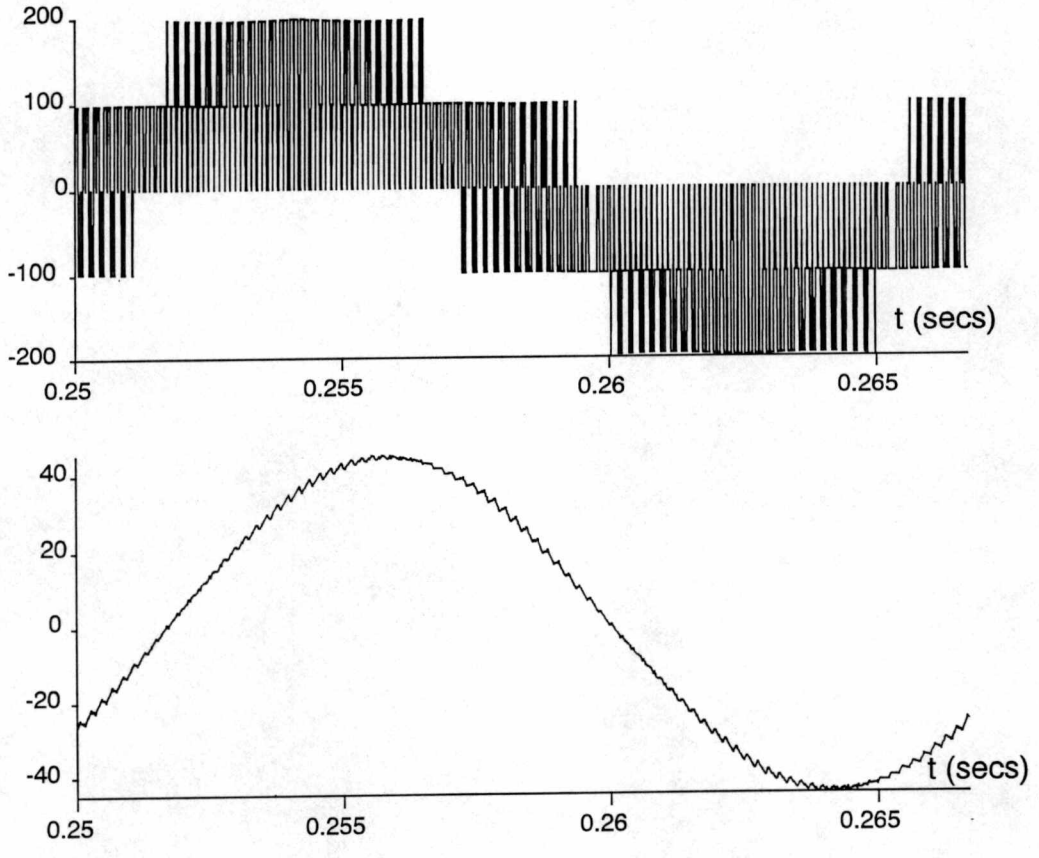


Fig. 5.7 Steady-State Machine Phase Voltage and Current Waves of a Simulated Drive System

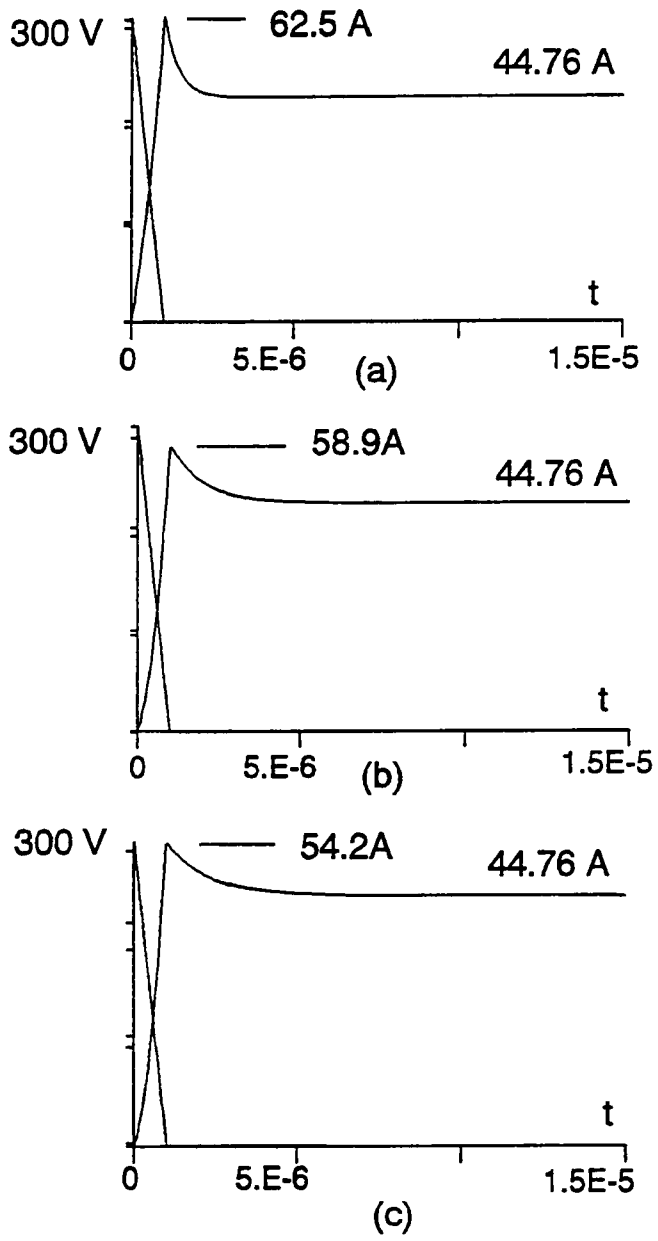


Fig. 5.8 Turn-on Voltage and Current Waveforms for Snubber Resistance Optimization Iteration, with

(a) R_s = 7 Ω, (b) R_s = 17 Ω and (c) R_s = 22 Ω,

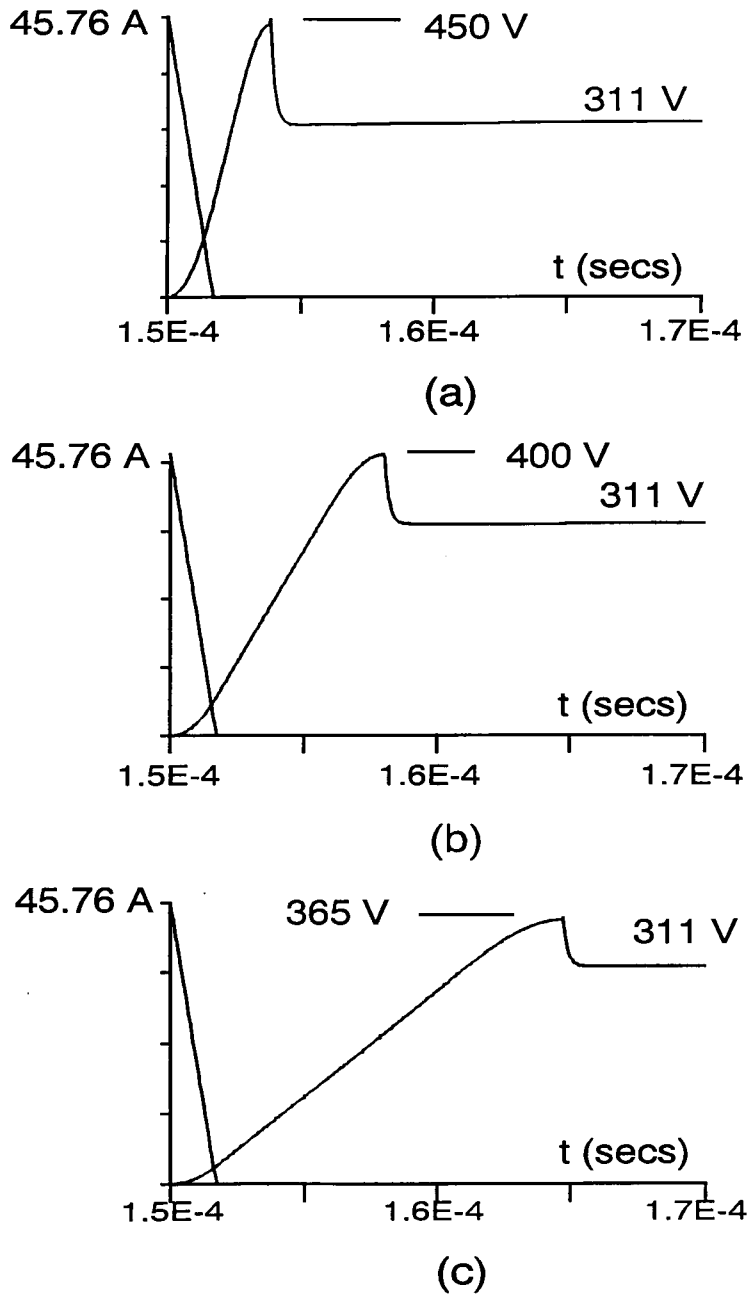


Fig. 5.9 Turn-off Voltage and Current Waveforms for Snubber Capacitance Optimization Iteration, with $R_s = 22\ \Omega$ and

(a) $C_s = 0.228\ \mu F$, (b) $C_s = 0.684\ \mu F$ and (c) $C_s = 1.5\ \mu F$

earlier, is the successful linking of PC Plus based expert system and PC-SIMNON based simulation programs which permits the automated simulation and optimization procedures.

CHAPTER 6

EXPERT SYSTEM AIDED AUTOMATED CONTROLLER DESIGN AND AUTO-TUNING

6.1 Introduction

An AC drive system, suitable for a high-performance application, can be seen to be a nonlinear, multi-input, multi-output system, with parameter variation problem. Apparently, to achieve with an AC drive the level of performance comparable to that of a DC drive requires complex control system. The recent trend has been to use digital control of AC drive systems, using fast and versatile fixed point or floating point microprocessors and digital signal processors (DSPs). Some form of vector control (DC or IVC) is employed routinely to achieve decoupled AC drive control, The control for the AC drives is thus invariably implemented in the form of assembly language or C based software, which, through the sensors, reads AC motor terminal quantities and shaft speed, and generates control voltages for the PWM inverter, in a manner that the AC drive exhibits high performance characteristics.

The complexity of the AC drive system to be designed, coupled with the rapid rate at which all the companion technologies such as microprocessors and DIPS, control systems, etc., are advancing, has made the expert AC drive design engineer's task extremely challenging. Usually, the expert designer's way to deal with the design complexities is to make simplifying assumptions, use rules of thumb and some preliminary calculations to design the control system for the AC drive. Once this is done, he writes the controller code in assembly language (for fast execution) and C language (for slower execution), debugs, assembles, links and downloads it to the processor memory, integrates it with the rest of the power converter and sensors. Now, he runs the drive system with the initial,

approximate values of the control parameters such as PI gains. Based on the results of the performance of AC drive, such as the speed and current responses, he fine-tunes the controllers iteratively, before putting the AC drive system in normal operation. Evidently, this process is expensive and extremely time-consuming, and requires a thorough knowledge of the state-of-the-art techniques in control and its digital implementation, as well as knowledge on AC motor drives.

Over the years, the expert systems have shown immense potential towards tasks involving the use of expert's know-how and judgement towards solving complicated problems defying a definite problem-solution method. Examples of expert systems applied to such applications are in design of multivariable control systems, computer-aided-design in mechanical engineering applications, VLSI CAD, etc. With the recent availability of expert system development shells, the task of fast development of expert systems has become somewhat easier. Expert systems, as discussed in earlier chapters, allow easy integration, expansion and distribution of expertise, enhancing their maintainability and upgradability.

This chapter discusses in detail the expert system based automated design and controller tuning for the AC drive system. This is in continuation with the design of the AC drive system discussed so far, in terms of the AC drive configuration selection, AC motor design and the power converter design and optimization. In the beginning, a discussion is presented on the control design considerations for the AC motor drive system. In the next section, software development methodology for the hybrid simulation of AC motor drive and its DSP based control, for real-time controller tuning is discussed. The discussion on expert system knowledge-base development follows next. The results at the end prove the applicability and usefulness of this software towards the AC drive control design and tuning. Throughout the chapter, the discussion uses the example of a TMS 320C25 based indirect vector controlled induction motor drive system to explain the principles of simulation software and expert knowledge base development. This is to demonstrate the expert system based controller design and tuning. However, a similar approach can be applied to design and tune the control of a variety of AC drive system control configurations. The optimized control software code is later downloaded to an actual experimental drive system, which achieves significant time and cost saving towards the design of AC drive system.

6.2 AC Drive Control System Design and Auto-Tuning Methodology

As outlined earlier, in terms of control, the AC drive system can be seen to be complex, nonlinear multi-input, multi-output system, with parameter variation problem. Also, because of the switching converter feeding the machine, the converter-machine system also has discrete-time characteristic. Additional complexity is added with the use of microprocessor / DSP based digital control for the AC drive. However, these discrete-time characteristics are somewhat overshadowed and mitigated by the sluggish nature of the AC machine response. In any case, to achieve control performance from an AC drive comparable to that of a DC drive requires immensely complicated control. A hierarchical control structure involving outer slower loops controlling the inner faster loops is usually employed. The recent trend has been to achieve this performance by employing vector control of stator current flowing in the motor. It can be shown that if the stator currents are transformed to the reference frame which rotates at the synchronous speed, and is fixed to the rotor flux vector position in real-time, the magnetizing and torque-producing components of the stator current appear as DC quantities, and are orthogonally decoupled. Hence, each can be controlled independently of the other, thereby achieving separately excited DC machine-like performance from the AC induction or permanent-magnet motor. The principles of direct and indirect vector control are explained in detail elsewhere.

To achieve the control of stator current would require using inverter topologies facilitating direct current control. One of the techniques used in the past was to use a current-fed inverter, which suffers from problems such as need for inverter switches with bidirectional capability, sluggish response, open-loop stability problem and torque pulsations. An improvement on this was the introduction of current controlled PWM inverters, which essentially had the PWM inverter switching based on the instantaneous current error which should stay within a hysteresis band. Although this is an improvement over current-fed inverters, it has the problems of loss of current control at higher speeds, as well as higher ripple in the load current due to varying PWM switching frequency. Both of these drawbacks have been eliminated by having the current control in the synchronous reference frame, where the quantities are DC, and using the current error to drive the voltage commands in d and q-axes through PI controllers. The PI controllers, although making the system operation slightly sluggish, ensure zero steady-state error and improve the response rise-time. They are also easier to implement. Hence they are preferred. Since the current control is achieved here in synchronous

reference frame, this type of control is referred to as synchronous current control. This maintains a constant PWM switching frequency as well as forces the desired currents by imposing the required voltages. Due to these features, synchronous current control has become popular with many of the applications. For example, if the application is servo speed control drive, the type of control chosen is indirect vector control, with synchronous current control in the inner loops and speed control in the outer loop for the induction motor drive, as shown in Fig. 6.1.

Once the type of control is ascertained, the next step is to determine the initial values of the control parameters. In the case of synchronous current controllers and the speed controller for servo drive application, the PI gains for each of the controllers need to be initialized. To ensure system stability, the initial values of the gains are chosen such that the system is overdamped. However, the initial values of these gains should be close enough to the optimum values, so that tuning can be achieved with minimum number of iteration steps. The selection of the initial values of control gains will now be discussed.

6.2.1 Control Gain Initialization

As mentioned above, P-I controllers are used to control the currents in d- and q-axis synchronous frames. The machine currents are controlled indirectly by imposing the voltages to the stator terminals corresponding to the current errors. In the case of servo speed control induction motor drive with indirect vector control and synchronous current control, the d-axis and q-axis equivalent circuits of induction motor can be considered independently due to the $i_{ds}^{e^*}$ and $i_{qs}^{e^*}$ decoupling in synchronous reference frame. Due to this, the control gains in d and q axis loops can be tuned independently. It is apparent that the values of these control gains will be dependent on the plant parameters, which are, in this case, the induction motor electrical parameters. Fig. 6.2 (a) and (b) show the d and q axis equivalent circuits of the induction motor in synchronous reference frame, with the current controllers imposing the stator voltages, which, in turn, control the currents in the motor. The motor currents are sampled every T_s seconds, transformed to synchronous frame currents, and the reference voltages $v_{ds}^{e^*}$ and $v_{qs}^{e^*}$ are generated by the synchronous current PI controllers, transformed to stationary frame phase voltages and impressed across the stator terminals. Normally, the sampling time of the current loops is kept much much smaller than the smallest time constant in the system (about 800 - 1000 times less), so that the discretization of the

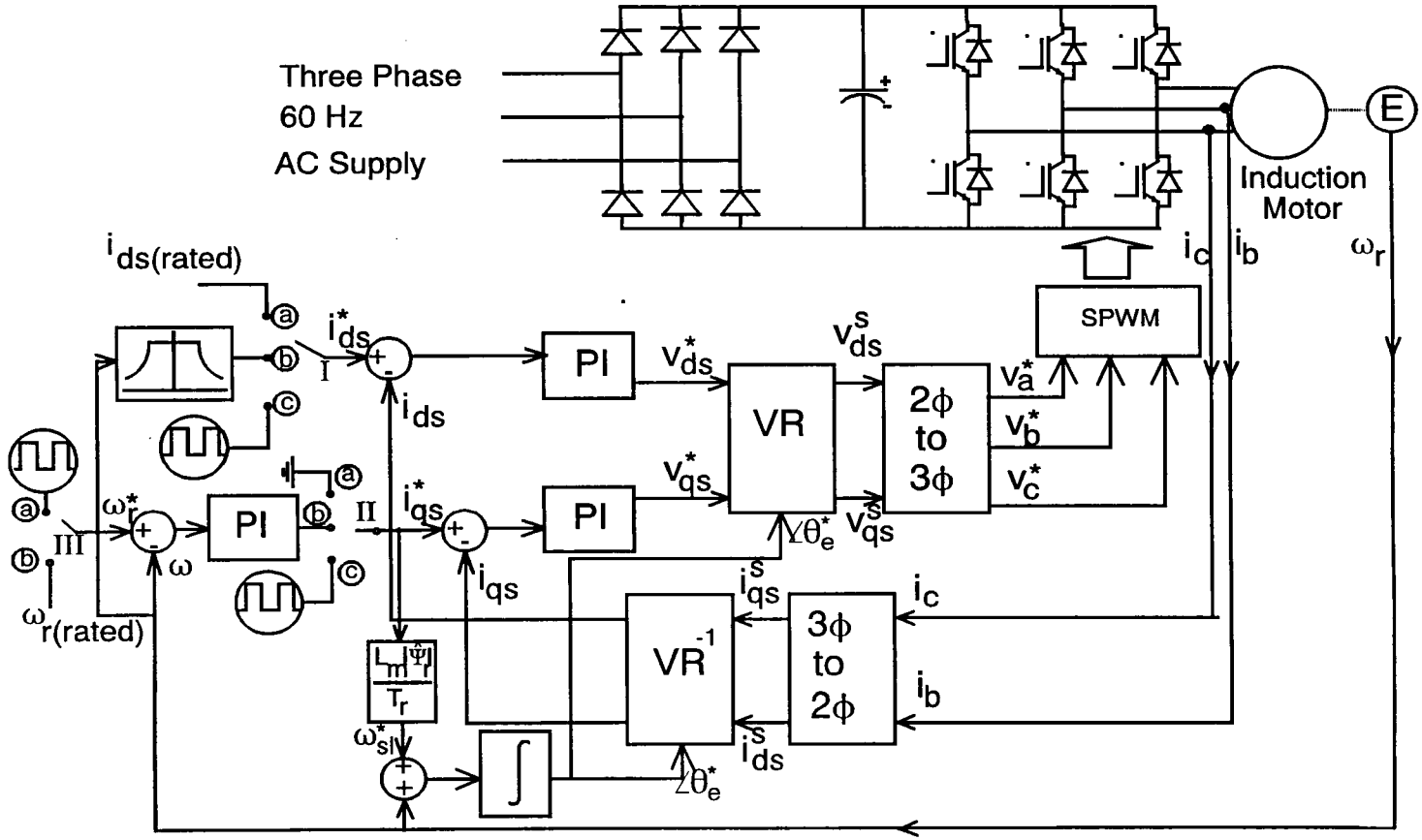
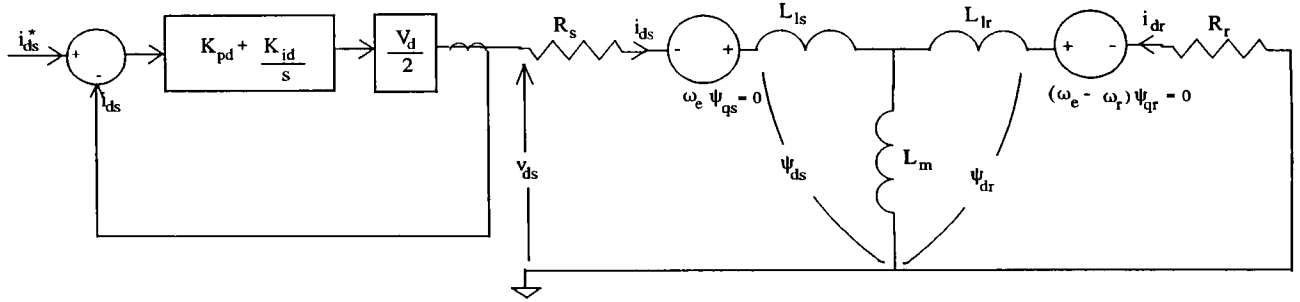
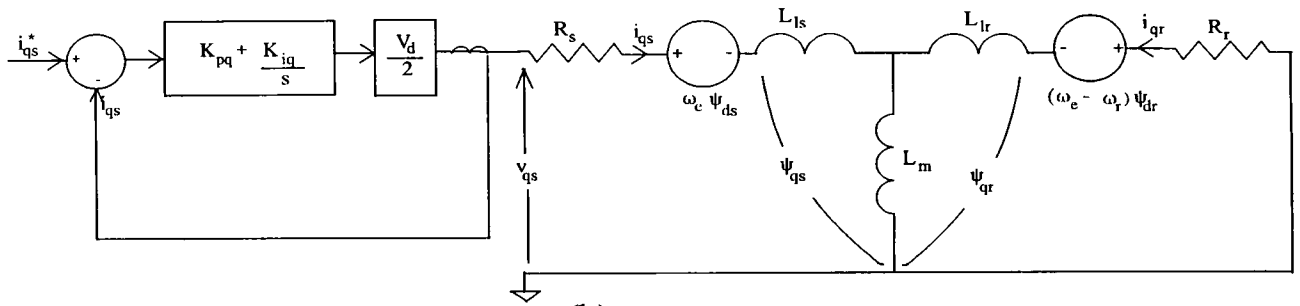


Fig. 6.1 A Typical Closed-Loop AC Drive System Control Structure for Servo Speed Control Application, with Various Modes of Tuning Implemented



(a)



(b)

Fig. 6.2 Control Structure of the Decoupled Synchronous Current Control Loops for

(a) D-axis Current Loop
(b) Q-axis Current Loop

system does not affect the system performance adversely. In fact, the w-domain transfer function resembles closely the s-domain transfer function at such a high sampling frequency. The transfer function of the d-axis current loop in the s-domain with the P-I controller and the plant (induction motor d-axis equivalent circuit), in terms of currents can be shown to be,

$$\frac{i^{e*}_{ds}}{i^{e}_{ds}} = \frac{V_d\left[K_{pd} + \frac{K_{id}}{s}\right]\left[\frac{Ls + R}{\left(s + \frac{R}{L + L_m}\right)\left(s + \frac{R}{L - L_m}\right)}\right]}{2 + V_d\left[K_{pd} + \frac{K_{id}}{s}\right]\left[\frac{Ls + R}{\left(s + \frac{R}{L + L_m}\right)\left(s + \frac{R}{L - L_m}\right)}\right]} \qquad 6.1$$

where K_{pd} = proportional gain of the P-I controller, K_{id} = integral gain of the P-I controller, L = $\sqrt{(L_m + L_{ls})(L_m + L_{lr})}$ = effective inductance of the induction motor , R = effective resistance of the induction machine = $\sqrt{R_s R_r}$, V_d= DC link voltage of the inverter, and s is the Laplacian operator. i^{e*}_{ds} = d-axis current command in synchronous frame, i^{e}_{ds} = the actual d-axis synchronous frame current. Since L_m » L_{ls} or L_{lr}, L ≈ L_m.

It is clear that the induction motor d-axis equivalent circuit model is second order, with one zero and two poles. However, one of the poles (at $\frac{R}{L + L_m}$) is dominant, with the other pole far away into the negative real axis. Thus, the transient behavior of the induction motor is decided by this dominant pole. As mentioned earlier, the presence of the P-I controller in the system improves the transient response and forces the steady-state error to be zero. Also, through the analysis of the characteristic equation of the transfer function in Eqn. 6.1, it can be shown that the values of the P-I gains K_{pd} and K_{id} depend solely on the induction motor equivalent circuit parameters, i.e., the stator and rotor resistances (R_s and R_r respectively) and leakage inductances (L_{ls} and L_{lr} respectively) as well as the magnetizing inductance L_m. It can also be shown that the value of L_m dictates the selection of the P-I gains. Although a closed form expression for the values of K_{pd} and K_{id} yielding slightly underdamped, close to critically damped response is quite involved to derive, initial values assigned to these gains need not be accurate, since the expert system performs the optimization

based on the actual responses obtained later. These initial values, corresponding to overdamped response, were based on the magnetizing inductance of the machine, L_m and were imposed to be,

$$K_{pd} = K_{id} = 3L_m \qquad 6.2$$

The q-axis induction motor equivalent circuit in synchronously rotating reference frame shown in Fig. 6.2(b), with the rated magnetic field established, is similar to the d-axis equivalent circuit (Fig. 6.2(a)), with one exception, i.e., the presence of the machine counter emf term $\omega_e\psi_{ds}$. However, for the sake of assigning the initial value of the K_{pq} and K_{iq}, the P and I gains in the q-axis controllers, this term can be ignored from the equivalent circuit, as in the very first pass of q-axis optimization, where the initial values are used, the rotor is at standstill, with only the flux established at rated value of i_{ds}. If the counter emf term is ignored in the beginning, and since the rest of the plant models, (i.e., the d-axis and q-axis equivalent circuits) are the same, the same initial values of K_{pq} and K_{iq} can be used as those assigned for K_{pd} and K_{id} in equation 6.2. Hence,

$$K_{pq} = K_{iq} = 3L_m \qquad 6.3$$

It should be noted, however, that once the optimization of the q-axis current controller begins, the rotor speed will no longer be zero due to a finite amount of torque produced by the command i_{qs}. As will be discussed later, the optimization passes take the finite rotor speed into account, while tuning the values of K_{pq} and K_{iq}.

In the hierarchical control loop structure, as mentioned earlier, the inner loops are normally the faster, current control loops. The slower control function, such as flux control, speed control or position control, is relegated to the outer loops. This is because they are required to interact with the relatively slower parameters of the AC drive control, such as the mechanical dynamics of the load and the slower electrical dynamics of the machine flux. In the case of the servo speed control drive, for example, the outer loop controls the speed by feeding the speed error through a PI controller as the torque command reference to the inner, faster, torque-control loop. Fig. 6.1 shows the loop structure. Obviously then, the design of the slower control loop PI gains is dependent on the motor and load mechanical dynamics, which is defined by the parameters such as the moment of inertia J and damping B. Of the two, the inertia is the parameter that directly affects the acceleration / deceleration characteristic of the motor, and hence, is used for initialization of the P-I gains for the

speed controller. As mentioned earlier, compared to the speed response, the inner, current loop response is instantaneous, and hence the i_{qs} loop dynamics is ignored while analyzing the mechanical, speed control system. In such a case, as shown in Fig. 6.3, the speed control response is solely governed by the motor shaft and load moment of inertia, J. The transfer function for the closed-loop speed control system can be easily shown to be,

$$\frac{\omega_r^*}{\omega_r} = \frac{\left[K_{ps} + \frac{K_{is}}{s}\right]\left[\left(\frac{3P}{4}\frac{L_m^2}{L_r} I_{ds(rated)}\right)\left(\frac{P}{2J}\frac{1}{s}\right)\right]}{1 + \left[K_{ps} + \frac{K_{is}}{s}\right]\left[\left(\frac{3P}{4}\frac{L_m^2}{L_r} I_{ds(rated)}\right)\left(\frac{P}{2J}\frac{1}{s}\right)\right]} \qquad 6.4$$

where, ω_r^* = speed command, ω_r = actual rotor speed (mechanical rad/s), K_{ps} = proportional gain of the speed loop P-I controller, K_{is} = integral gain of the speed P-I controller, P = number of poles, L_m = magnetizing inductance of the induction motor (Henry), $L_r = L_m + L_{lr}$, L_{lr} = rotor leakage inductance, J = total shaft moment of inertia (Kg-m^2), $I_{ds(rated)}$ = rated value of magnetizing component of stator current, s is the Laplacian operator.

An analysis of the characteristic equation of the transfer function (Eqn 6.4) shows that the response of the speed loop will be near critically damped when

$$\frac{K_{ps}^2}{K_{is}} = \frac{32\,JL_r}{3PL_m^2 I_{ds(rated)}} \qquad 6.5$$

where the symbols are already defined earlier with equation 6.4. The expert system initializes the gains K_{ps} and K_{is} to values which are slightly lower than the one given by the above relation.

6.2.2 Control Tuning Methodology

The initial values for the control parameters, such as the P-I gains, discussed in earlier section, are assigned so as to make the response of the corresponding control loop slightly overdamped, which

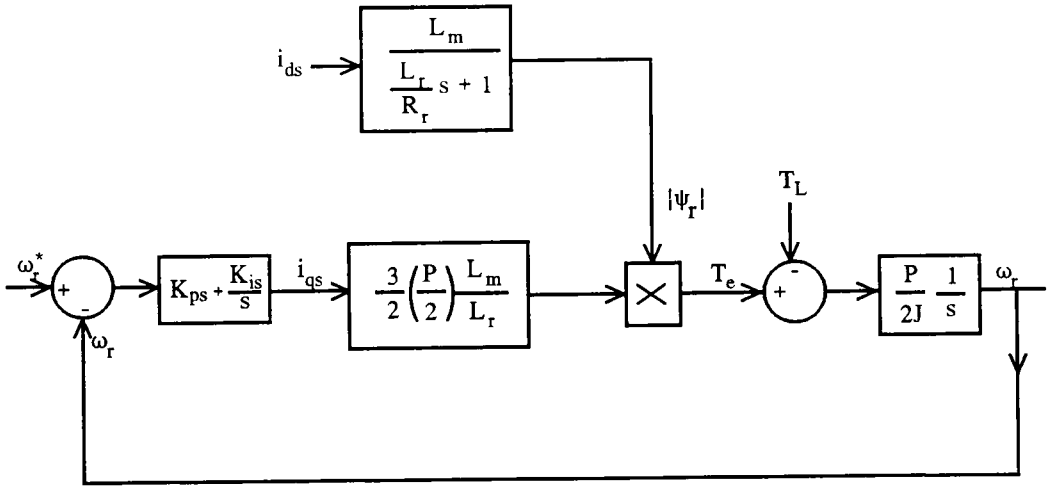


Fig. 6.3 Speed Control Loop Structure for Servo Speed Control Drive System

ensures system stability. However, the response of the system is sluggish in this case, and the control parameters need fine-tuning[47 - 52]. However, this can be done only on the basis of closed loop response obtained for each of the control loops. The tuning process is thus carried out in real-time, where the test signals are supplied to the AC drive system, for various loops in isolation, and the corresponding responses are examined to vary the control parameters, in the direction that achieves optimum response performance. Many sophisticated tuning algorithms have been developed and applied in practice. However, for the sake of simplicity of implementation, the algorithm used for tuning of the P-I gains is somewhat straightforward, and is shown in Fig. 6.4(a), where the controller response is examined for its rise-time and overshoot, and the gains are varied in such a manner that the response approaches its tuned profile, in the form of slight underdamping with a small amount of overshoot. Fig. 6.4 (b) shows the optimization search profile for K_p.

The tuning of the controllers is done one loop at a time. The first to be tuned are the inner, faster loops, followed by the outer, slower loops. This ensures stable operation of the drive during all phases of optimization. In the case of control tuning of the servo speed control drive, for example, the i_{ds} current loop is tuned first, with i_{qs} command set to zero. Table 6.1 shows the switch positions during the various modes of tuning, with reference to Fig. 6.1. Next to be tuned is the i_{qs} loop, with the rated i_{ds} established at all times, with the i_{ds} current loop PI gains set to their tuned values, and with the speed loop open. The outer speed loop is tuned next in the same manner. As indicated earlier, for tuning purposes, the reference signals for each of the loop are provided in the form of square waves of 50% duty cycle, so as to test the system performance against the worst-case current and speed variations that might arise during the normal operation of the system.

In practice, the tuning of the controllers is performed with the actual AC drive system in the laboratory, where the power converter is connected to the AC motor, whose terminal and shaft quantities are sensed and sent to a microprocessor or DSP, which, in turn, processes these variables and produces the control signals for the power converter switching. The controller is thus implemented in the assembly language as well as C language for the particular processor used. In order to have a very close matching between the performance of the controllers in the laboratory drive system and the expert system based tuning, it is desirable that the controllers used in the expert tuning be implemented in the same DSP assembly language and C language, and simulated in a manner analogous to real-time operation. This is exactly what is achieved by the expert system

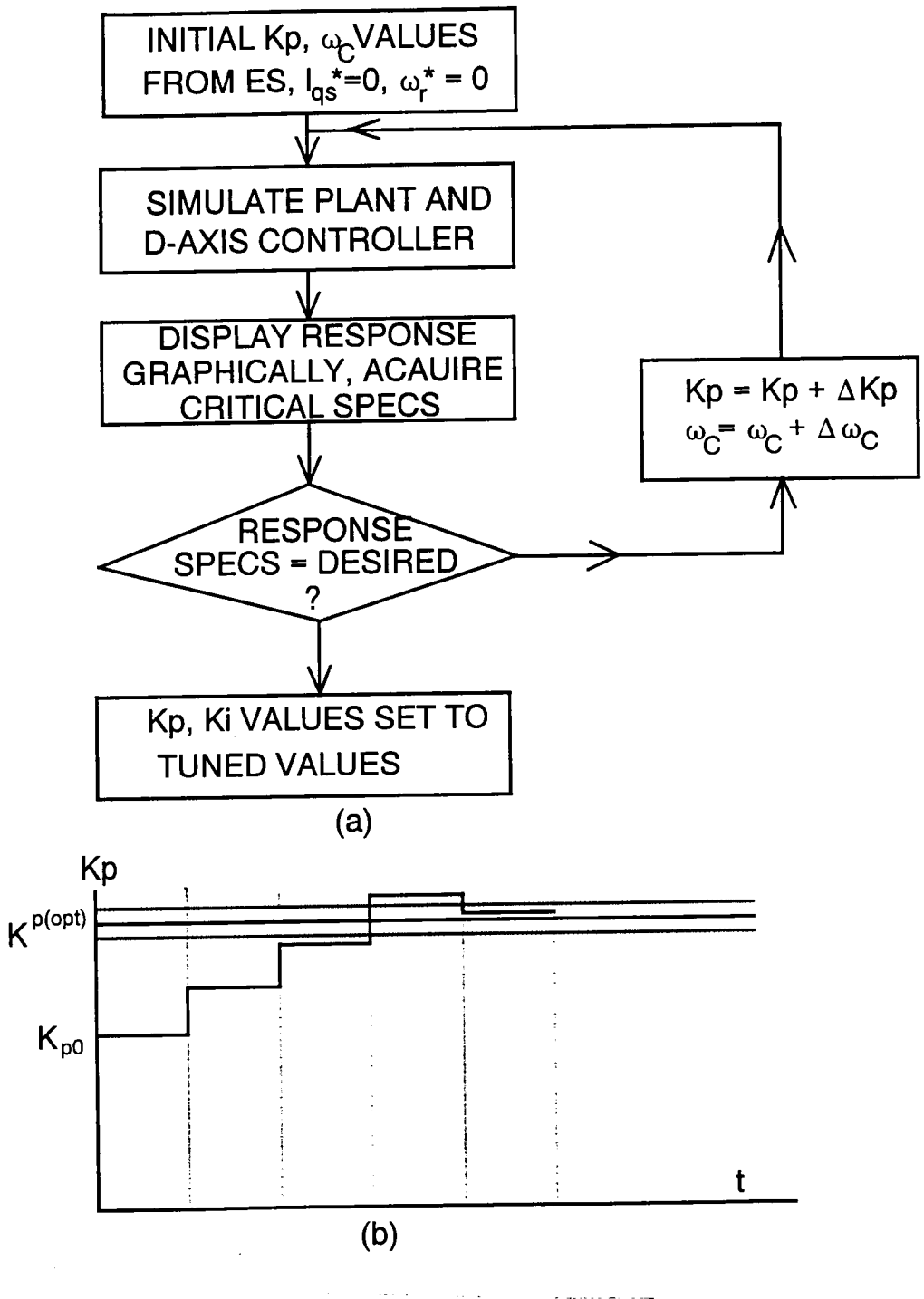


Fig. 6.4 Control Tuning Methodology

(a) Flowchart for K_P, K_I Tuning
(b) Search Profile for K_P

Table 6.1

Switch Positions for Various Modes of Tuning

	Switch I	Switch II	Switch III
I_{ds} Loop Tuning	c	a	N/A
I_{qs} Loop Tuning	a	c	N/A
Speed Loop Tuning	b	b	a

based controller tuning, with the help of the system and control simulation software developed specifically for the AC drive system design, as will be described next.

6.3 Expert System Based Real-Time Simulation Software Development Methodology

As discussed in the earlier chapter, expert system, though capable of performing symbolic processing and limited amount of arithmetic, is not suited for doing number crunching-intensive, procedural tasks such as simulation of dynamical systems. Fortunately, PC Plus based expert system has the ability to communicate with the external, DOS-based programs, for example, the dynamic simulation programs such as SIMNON, the simulator programs, such as the Texas Instruments 320C25 simulator[55-58], and various other executable files, which can be written in languages such as C.

In order to verify the performance of the initial controller design done by the expert system, it is necessary to simulate the design. To achieve this, the expert system takes the help of the simulation software. Also, the simulation must maintain a close correlation with the real-time performance of the drive. Thus, it will be ideal if the control is implemented in the actual microprocessor / DSP software code, to obtain reliable and realistic fine-tuning of the controllers. This arrangement also allows the downloading of the code to the actual drive system once the controllers are tuned. This obviates the user having to go through the tedious process of control program development and debugging. It is also advantageous to have the control simulation program autonomously and independently usable, if the user decided to supersede the expert system for the purpose of fine-tuning the controllers. All of the above requirements are met by the software being presented here.

Once the expert system completes the initialization of the control gains for the AC drive system, it exports these values to external simulation programs. The AC drive system comprises of the plant, i.e., the converter - machine combination, as well as the controllers, which perform the vector control and synchronous current control. The converter-machine simulation is performed by the system simulation program SIMNON, whereas the controller simulation is performed in assembly language and C language in the actual TI 320C25 DSP simulator environment. The simulator allows the execution of DSP code without actually using the physical DSP chip by making the CPU of the PC simulate the DSP behavior. Thus, it is a very useful tool for testing, debugging and simulation of the

DSP controller code. However, in order for the expert system to communicate as well as control the simulation of controllers and their fine tuning, it needs the help of the external C-based interface programs, which allow parameter passing back-and-forth between the expert system, the system simulation in SIMNON and the DSP based controller simulation. The interface between expert system and SIMNON for plant simulation has already been discussed earlier, and the same framework is extended further. Fig. 6.5 shows the structure of the expert system based control design and auto-tuning software, having the following four major components:

- Expert system module for design and auto-tuning of controllers
- System simulation programs in SIMNON, performing converter-machine simulation
- Controller simulation programs in DSP simulator
- C-based interface routines, which allow communication and control of different portions of software

As discussed earlier, the expert system, the system simulation program as well as the DSP simulator are programs meant to work standalone. Hence, in order to combine them to work together, the interface framework is necessary. The C-based interface routines achieve the following:

- Data communication between the expert system and simulator
- Data communication between the system simulation program and the simulator
- Data acquisition and display for the controller response, based on the mode of tuning, i.e., the i_{ds} loop tuning, i_{qs} tuning or complete system
- Auxiliary support functions, such as scanning of the response data to generate its rise time, overshoot, nature of response, etc; manipulating various data files for to enable smooth, logical functioning of the software

Each of these components will be discussed in detail.

6.3.1 Expert System and System Simulation Program Interface With DSP Simulator

The expert system embeds the know-how of the expert to conduct the tuning of controllers in the AC drive system by systematically performing various test modes with varying control parameters. The

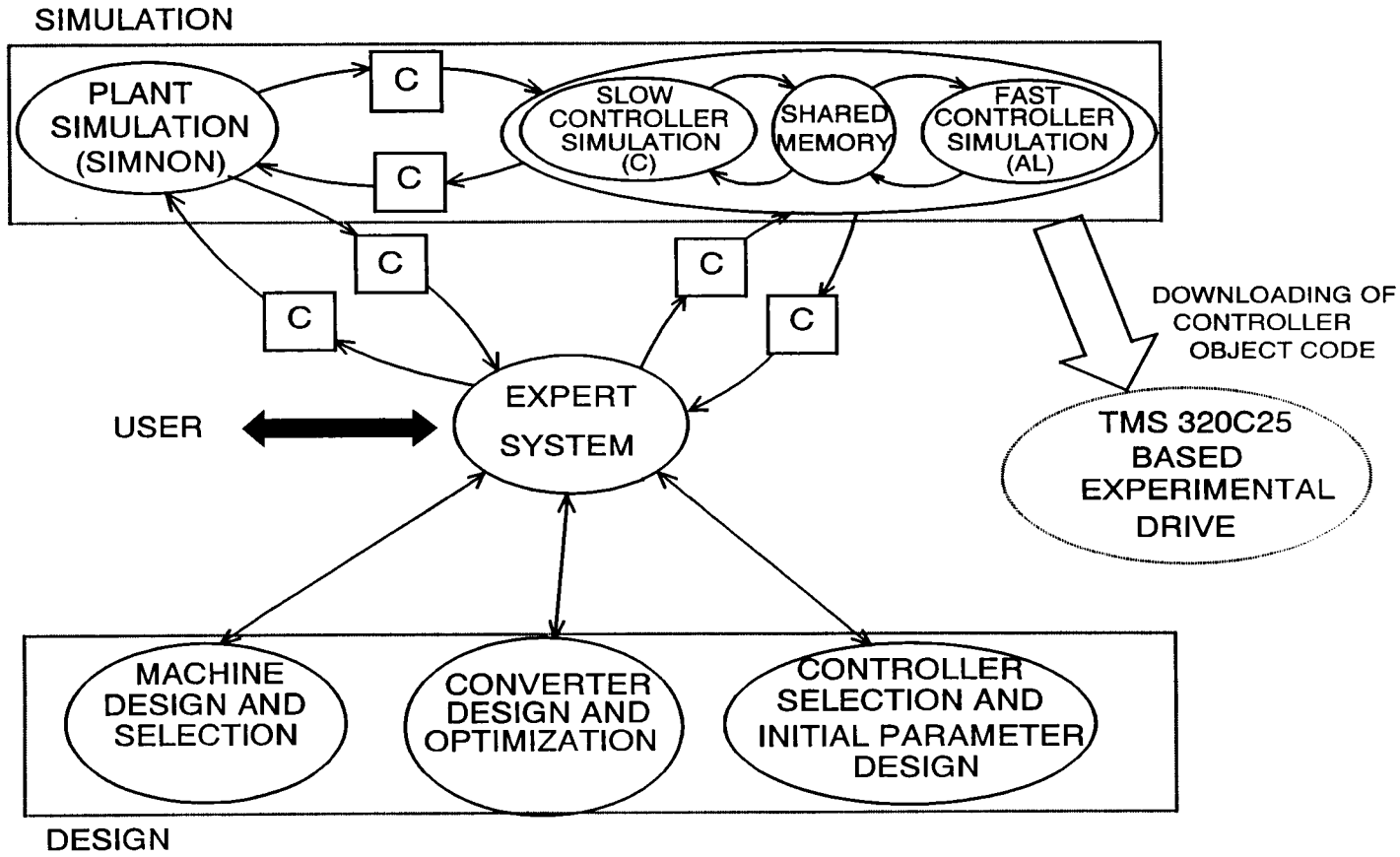


Fig. 6.5 Software Structure for Expert System Based Controller Design and Auto-Tuning

model for the converter-machine, or plant, is predeveloped in the system simulation program, SIMNON. Also, the controller programs are developed in C for slower portions and assembly language of DSP 320C25, using the standard debugging, compilation, assembly and linking tools available from Texas Instruments. Through the simulation command file for the simulator, the input/output ports of the DSP can be simulated to be input/output data files, which accept the data in IEEE single precision floating point hexadecimal format. The simulation command file also contains commands for the simulator to automatically load the executable controller routines, connect various DSP ports to different input/output files, execute the controller program, close all input/output files and exit.

During the control tuning process, the expert system varies the values of control gains, and outputs them to external ASCII files as floating point numbers. The system simulation program also outputs, at the end of every sampling interval, the values of motor quantities, such as the line currents and speed, in ASCII floating point decimal format. In order that the simulator is able to read these values, their format needs to be converted to IEEE single precision floating point hexadecimal format. Similarly, the results generated by the simulator are in this IEEE hex format, which need to be converted back to floating point decimal format, to be understandable by the expert system as well as the system simulation program. Thus, it is necessary that this format conversion be performed through some external means. This is done by C-based interface routines.

6.3.1.1 ASCII Floating Point Decimal to IEEE Single Precision Floating Point Hex Format Conversion Routines

These routines make possible the data communication between the expert system and the simulator as well as SIMNON and the DSP simulator. Although they are different routines, their format conversion logic is the same, and hence will be described together. As mentioned earlier, the both the expert system as well as the SIMNON program output the values for the DSP simulator in ASCII floating point decimal format, in data files. For example, during the tuning of i_{ds} current control loop, the expert system will output the values of the changed gains K_{pd}, K_{id}, as well as some system-specific information such as the current sampling time, the command value of I_{ds} and the slip gain value in the order shown below, in the decimal floating point format:

0.0768 0.0614 2.e-4 4.608 1.01099350522512

The expert system also outputs the information on the mode of control tuning, (Mode = 1 for i_{ds} loop tuning, Mode = 2 for i_{qs} loop tuning and Mode = 3 for complete system simulation). The controller program in DSP assembly language, however, requires the data above to be converted in the form shown below (internally, these are read as 1's and 0's):

4952	*; LSW, K_{pd}* (= 0.0764)
3d9d	*; MSW, K_{pd}*
cccd	*; LSW, K_{id} (= 0.0614)*
3f4c	*; MSW, K_{id}*
b717	*; LSW, Sampling time (= 2e-4)*
3951	*; MSW, Sampling time*
74bc	*; LSW, $I_{ds(ref)}$ (= 4.608)*
4093	*; MSW, $I_{ds(ref)}$*
683c	*; LSW, Slip gain (= 1.01099)*
3f81	*; MSW, Slip gain*

The number of parameters sent out by the expert system varies depending on the mode of tuning. The format conversion routine reads the current mode of tuning information, reads the parameter data file from the expert system, remembers their order, and converts them to the IEEE single precision floating point hex format shown above. The routine utilizes C functions manipulating character strings and format of numbers, to perform this format conversion.

The system simulation program, SIMNON, as indicated earlier, outputs the motor quantities such as the line currents and the speed values, in the floating point decimal format. This is converted to IEEE floating point hex format by another routine similar to the one converting the format for the expert system data. This routine is simpler, since the number of variables from the system simulation programs does not change with different modes of tuning.

6.3.1.2 IEEE Single Precision Floating Point Hex to ASCII Floating Point Decimal Format Conversion Routines

Once the DSP simulator implements the controller routines, it outputs the control variables, v_{ds}^{e*}, v_{qs}^{e*}, the angle θ_e^* for the system simulation program to use in the simulation spanning the next sampling interval, and some other variables, such as the controller responses, for storage and future display,

to a file. These variables are in the IEEE floating point hex format, as shown below:

```
0000        ; LSW, v_ds reference
3e0e        ; MSW, v_ds ref.
999a        ; LSW, i_ds error
3fb9        ; MSW, i_ds error
0000        ; LSW, θ_e
0000        ; MSW, θ_e
0000        ; LSW, v_qs ref.
0000        ; MSW, v_qs ref.
```

The number of variables stored in the data file above can change, depending on the mode of tuning. Thus, the C-based routine reads the information on the mode of tuning, and converts the above data from IEEE floating point hex format to the ASCII floating point decimal format as shown below:

```
0.138672    ; v_ds reference
1.450000    ; i_ds error
0.000000    ; θ_e
0.000000    ; v_qs reference
```

The format conversion of data from decimal to floating point hex and back, however, is not sufficient to achieve the interface between the expert system, SIMNON and DSP simulator. There are many complexities involved in the operation of the three of these programs, as will be highlighted next.

6.3.2 Methodology of Autonomous Simulation of AC Drive System

The PC Plus based expert system requires operation with the MS-DOS operating system. One of the problems that was faced early on was the bottleneck of 640 kB base memory, in which all the DOS-based programs must operate. This is impossible, since each of the expert system, SIMNON and the DSP simulator require the entire 640 kB of memory for its own execution. This is somewhat alleviated since the PC Plus based expert system allows some control over the computer's expanded or extended memory. This would allow it to remain active in the background. However, *only one* of the two simulation programs, i.e., SIMNON or DSP simulator would be allowed to remain in the base memory. Thus, methodology for achieving pseudo-multitasking capability in the software was developed, with which only one of the simulation programs would remain active in the memory, and

communicate with each other.

In practice, the real-time control is achieved by sampling the motor currents and speed at a certain rate, and supplying them to the DSP based controller program, which would operate on these variables, and perform the vector control and current control on these quantities to produce the control variables for the power converter switching at the next sampling instant. Thus, the operation of the AC drive system during any sampling interval is independent and in parallel of the DSP controller operation. Thus, both of them can be considered independently. In the AC drive control tuning software developed here, this principle is utilized. Since only one of the simulation programs can exist in the memory at one time, the simulation of the plant (i.e., the converter-machine combination) is carried out for one sampling interval, followed by the simulation of the controllers, in a cyclic manner.

Fig. 6.6 shows the process used to achieve the plant and control simulation. The expert system, once it has sent the control gain values to the simulator through the format conversion program, generates a DOS-batch file (a batch file is a file which contains DOS commands to be executed in the sequence they are written) with the help of one of the C-based routines. This file has the commands for invoking simulation in SIMNON, passing parameters to DSP simulator executing the format conversion routine, invoking simulation in simulator, converting the simulator results to decimal floating point values and passing them to SIMNON simulation for the plant, and perform these operations in a continuous, cyclic manner. The pseudo-multitasking is achieved by entering and exiting each of the simulation programs after completing the simulation of the system for one sampling interval. This ensures that only one of the simulation programs is loaded into the 640 kB of base memory in DOS. However, by doing so, all the intermediate state values for both the plant and controller are lost when the program is exit, which would be unacceptable. Moreover, by just leaving the continuous simulation task to an infinite loop batch file, the expert system would lose complete control of the controller tuning process. These problems are solved by a C-language based master control routine, which performs the following tasks:

- Generates SIMNON plant state initialization file at the beginning of every sampling interval, combining the previously stored plant states and newly generated controller states

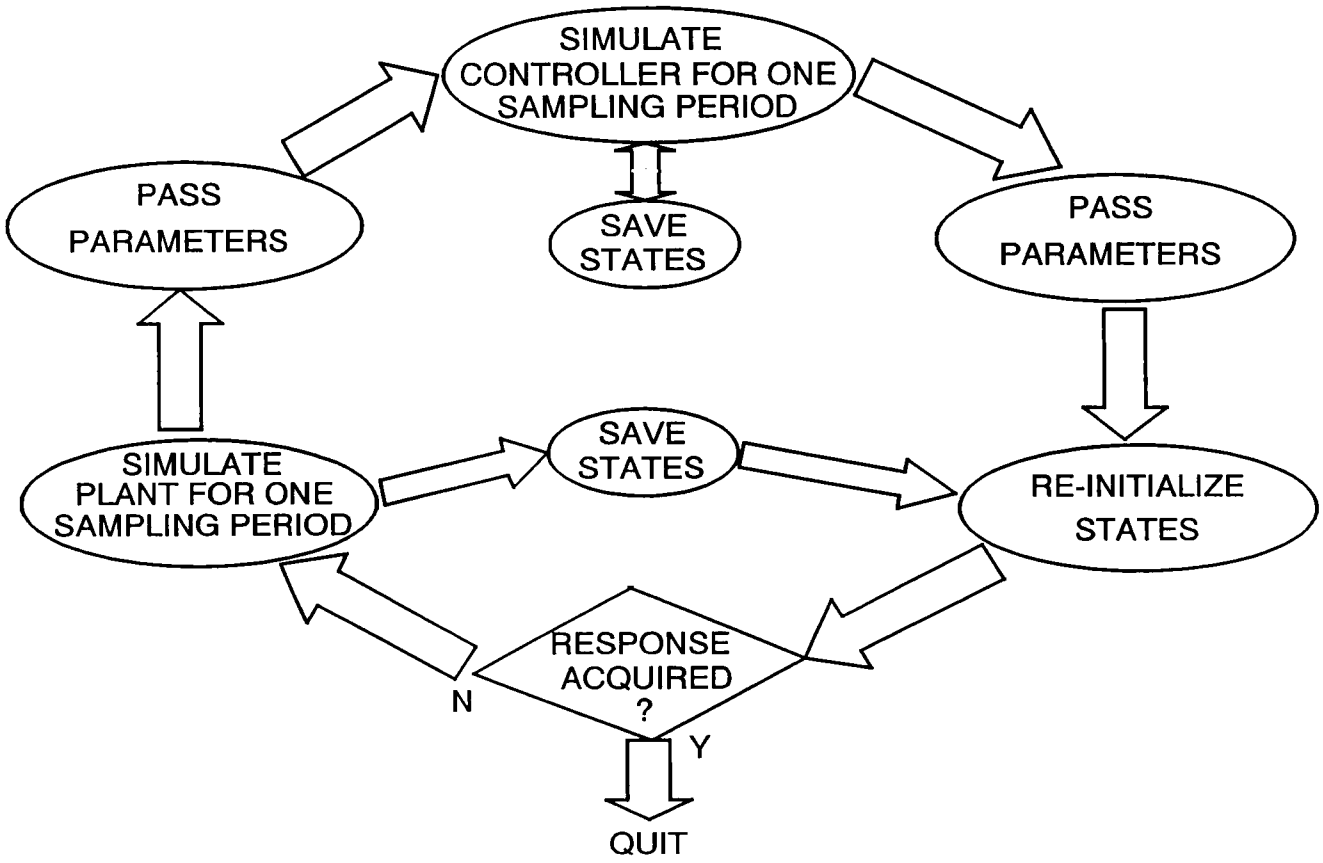


Fig. 6.6 Pseudo-Multitasking Operation with Continuous Simulation of Plant and Controller from a Batch File

- Stores controller response data values to buffer files for their later display and analysis
- Generates SIMNON plant simulation command file every sampling interval
- Keeps track of real-time and changes modes of software operation as the controller tuning progresses
- Enables the expert system to intervene the drive simulation at the end of each pass of controller tuning

Another C-based interface routine achieves the scanning of the stored controller responses and provides the information on its rise-time, overshoot and the nature of response to the expert system, so that it can proceed with the controller tuning by varying the PI gains based on this information. As highlighted earlier, the complete simulation and interface software framework is such that it can be decoupled from the expert system, to be used autonomously. The process of controller tuning will be described now, with the example of tuning the AC servo speed control drive controllers.

The tuning of the controllers progresses in three modes as explained earlier. Since all modes of operation are similar in terms of the software execution, the operation during the tuning of i_{ds} current control loop will be explained in detail. As shown in Fig. 6.7, the tuning process begins with the expert system calculating the initial values of the K_{pd} and K_{id} gains and exporting them to a data file. Then, the format conversion routine, (explained earlier) is called by the expert system to provide the control gain values to the DSP simulator in IEEE single precision floating point hex format. Now, the expert system also writes the parameters of the converter-machine model to the SIMNON program, as well as the information on the mode of tuning (mode =1) to a mode data file. Next the simulation batch file is created by the expert system with the help of another C-based routine, which is explained earlier. The value of the reference square wave for the i_{ds} command is also written to a file. Now, the simulation batch file is invoked by the expert system and the simulation of plant and controller continues in cyclic fashion, in a manner explained earlier, in Fig. 6.6. At the end of one period of controller simulation, the master control routine, which is C based, automatically breaks the AC drive controller simulation loop and transfers the control back to the expert system.

The expert system now uses the help of another C-based routine, also introduced earlier, to scan the response of the controller to obtain the nature of response as well as its rise-time and overshoot. It

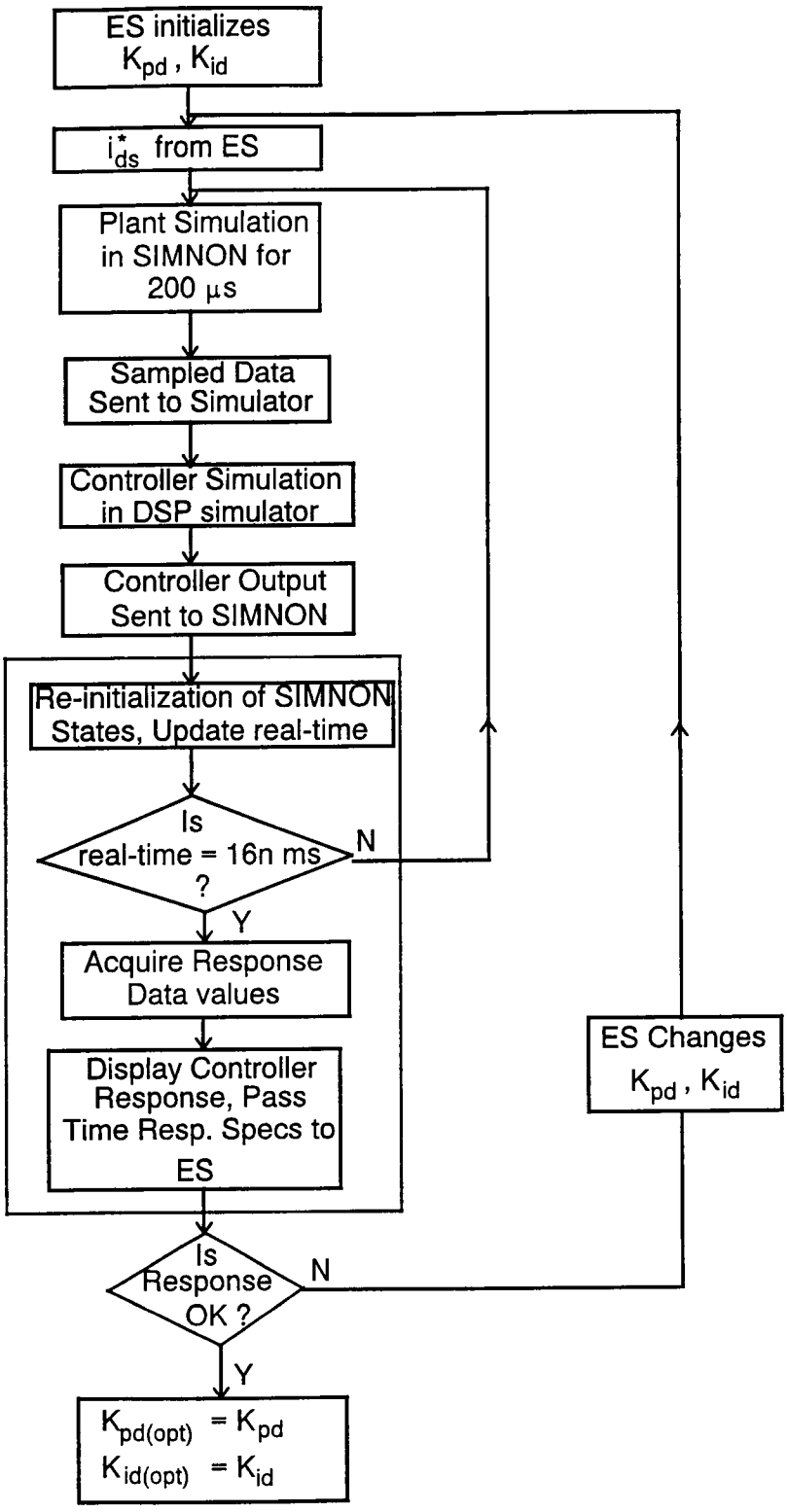


Fig. 6.7 Expert System based Controller Tuning Flowchart Showing Details of Software Execution

also displays the response graphically on the screen for the user to observe. Based on the rise-time and overshoot, the expert system alters the PI gain values slightly in the direction of achieving proper tuning of the controller, and writes these values as well as the new reference square wave command for the controller. Now, it repeats the procedure outlined so far, till it finds the response of the controller satisfactory. The optimization process then continues to tune the i_{qs} current loop controller and lastly the outer, speed controller is tuned in a similar manner.

6.4 Knowledge Representation

The expert system contains the knowledge both on the controller gain initialization and their tuning. The knowledge on controller gain initialization and the strategy for their tuning forms the core of the declarative or fact-like knowledge, whereas the process of manipulating various C-language based programs in a certain order represents the procedural knowledge. Also, the deep knowledge is embedded in the software in the form of plant and controller models in simulation programs, whereas the shallow knowledge is mainly the heuristic knowledge on selecting the values of control gains as well as varying them to achieve optimum controller performance.

6.4.1 Knowledge Base Structure

Fig. 6.8 shows the structure of the expert system developed for controller design and auto-tuning. The domain expertise of the AC drive control design engineer in initial design of control gains, the test-running of the AC drive system in various modes of tuning, and varying the control gains to achieve optimized control performance, is embedded in this expert system. The expert knowledge is stored in the expert system in the form of frames, parameters and rules. The four frames achieve all the four major tasks mentioned earlier, i.e., the initial design of control gains, and the auto-tuning of control in the three modes. The frame CONTROL-DESIGN essentially calculates the initial values for control gains by using the motor electrical parameters and load mechanical parameters, which can either be made available to it through the motor locked-rotor and no-load test or can be extracted from its nameplate ratings, using intelligent estimation techniques, such as genetic algorithms. For example, the following rule sets the values of PI gains of speed loop controller using Eqn. 6.5:

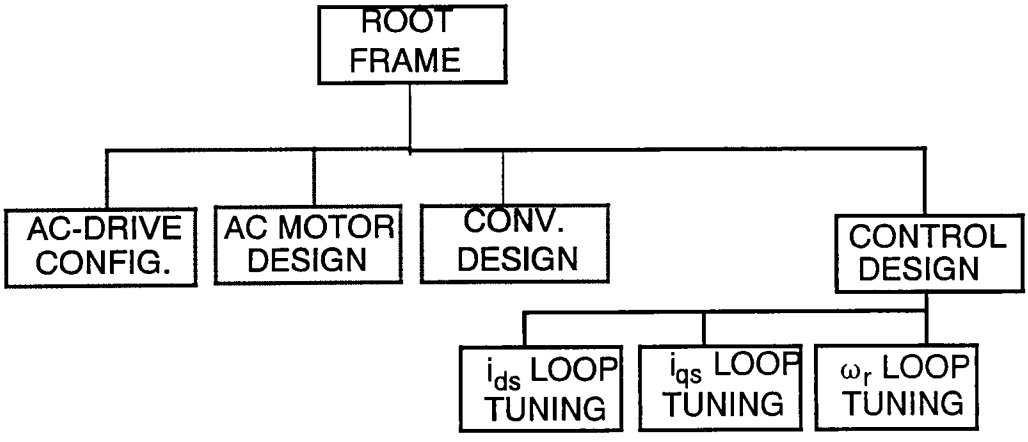


Fig. 6.8 Structure of the Expert System Knowledge Base for Controller Design and Auto-Tuning

IF : J IS KNOWN AND
LM IS KNOWN AND
LLR IS KNOWN AND
POLES IS KNOWN
IDS-RATED IS KNOWN
THEN : KPS = 32 * J * (LM + LLR) / (3 * POLES * LM * LM * IDS-RATED)
AND KIS = 1 * KPS

The expert's know-how on performing the simulation of the drive control system as well as its control tuning is embedded in three sub-frames, named IDS-TUNING, IQS-TUNING, SPEED-TUNING, which explain themselves. The procedure outlined in earlier section on the control tuning methodology is represented in the form of various parameters and rules that perform each of the tuning steps. With the help of these sub-frames, the expert system achieves the task of tuning each of the control loops.

6.4.2 Expert System for Control Design and Automatic Tuning

During a typical consultation with the client, the expert system, based on the user's application specs, configures the AC drive system, designs the nameplate specs of the drive motor, designs the power converter, and if the application requires closed-loop, vector control type of control, for example, in the case of servo speed control drive system, the expert system invokes the control design and automatic tuning by instantiating the control design subframe with the following rule:

IF : CONVERTER-DESIGN-DONE AND APPL-TYPE = HIGH-PERFORMANCE
THEN : CONSIDERFRAME CONTROL-DESIGN

The expert system then proceeds to assign the initial values of the control gains. Then, it automatically invokes the subframe IDS-TUNING, and invokes the plant and controller simulation, so that PI gains in the i_{ds} current control loop are tuned. This procedure is analogous to laboratory run of the IVC-IM drive system with initial values of PI gains, and observing the responses to vary their values till satisfactory type of response of i_{ds} controller is obtained. Next, the expert system follows the similar procedure to tune the i_{qs} current loop, by applying a square current reference in

q-axis current loop, with the i_{ds} reference value set to rated value. At the end of every pass of control tuning, the expert system also displays the controller responses to the user. In each mode of tuning, the controller tuning criteria used is the attainment of response which is slightly underdamped in nature, with about 5% overshoot, so as to ensure rapid rise-time and zero steady-state error. Once the two inner loops are tuned, the expert system proceeds to tune the outer speed loop. The following rule achieves one pass of i_{ds} loop controller tuning, through outputting the modified values of K_{pd}, K_{id} for overdamped response, conducting simulation of the IVC-IM drive system to acquire the i_{ds} controller response:

```
IF  :: (NAT-RES = OVERDAMPED AND ! IDOPT-DONE )
THEN :: (KPD = (E (+ (VAL1 FRAME DELP-KPD) (VAL1 FRAME KPD)))
        AND KID =
       (E (+ (VAL1 FRAME DELP-KID) (VAL1 FRAME KID)) )
         AND WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP
         AND DOS-CALL "par2hexn.exe" ""
         AND DOS-CALL "bfgen.exe" ""
            AND MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat"
                                                        MODSIM
               AND DOS-CALL "simodfy.exe" ""
         AND DOS-CALL "refresh.exe" ""
         AND PRINT " The response of Ids controller is still overdamped, so
       the simulation will continue with the new Kp and Ki .. " " "
       AND DOS-CALL "" "infsim" AND CD3 )
```

Once the simulation of the drive and controller is complete for one pass, the expert system regains control of the optimization process, displays the response graphically on the screen using the graphical display capabilities of SIMNON, scans the response data to obtain the transient response specs, such as the rise-time, overshoot, and the nature of response, using the following rule:

```
IF    :: IDS-LOOP-SIMULATION-DONE
THEN :: (MODSIM = "DACQDISP"
          AND WRITE-DOS-FILE "control.dat" MODSIM
        AND DOS-CALL "simodfy.exe" ""
          AND DOS-CALL "simnon.exe" ""
          AND DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat"
          AND READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES
          AND READ-DOS-FILE "respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3
                                          INDEX 4 X4
          AND IDRESP-ACQD )
```

The flag IDS-LOOP-SIMULATION-DONE is set when one pass of the i_{ds} current loop controller simulation is complete. Now, the expert system changes the mode of simulation in SIMNON to data acquisition and display of controller response by setting the flag DACQDISP, and then invoking the SIMNON program, which automatically knows to load the response data and display it graphically. Now, the expert system executes the C-based routine that scans the controller response, say i_{ds} profile, to obtain its rise time, overshoot and the nature of response, which it reads from another files at the end, and sets another flag called IDRESP-ACQD, to proceed with the tuning process further.

It is through the cascading of the rules shown above, that the expert system achieves the tuning of i_{ds}, i_{qs} and speed loop controllers.

Once all the controller gains are optimized, the expert system displays the results of the controller tuning in the form of tunes PI gains for i_{ds}, i_{qs} and speed control loops.

6.5 System Performance

The control design and tuning is demonstrated here in the example of a 5 hp indirect vector controlled induction motor drive system, with synchronous current control in the inner loops and with the outer speed control loop as shown in Fig. 6.1 earlier. As mentioned earlier, the vector control and current control functions are implemented in TI DSP 320C25 assembly and C language, whereas the plant simulation is implemented in SIMNON. The parameters of the AC200 induction motor are shown in Table 6.2. The process begins with the expert system calculating the initial values of the control gains for the inner current loop controllers as well as the outer speed loop controller. Next, these values are sent out to the simulation software, which performs the simulation of the converter-machine combination in the dynamic system simulation program, SIMNON, and the simulation of the controllers in the actual Texas Instruments DSP 320C25 simulator. The C-based interface routines not only pass the data parameters back and forth, by converting their formats but also help the expert system acquire, display and scan the responses of the controllers, to perform the optimization of the PI controllers. For the purpose of allowing the responses sufficient time to settle, the inner current controller responses are acquired for 16 ms each (31.25 Hz, 50% duty cycle square waves of 50% of rated amplitude), where the basic sampling time is 200 μs, whereas the outer, speed loop is supplied with the reference value of 10 r/s (96 rpm), and the

Table 6.2

Parameters of AC200 Induction Servomotor

5 hp 180 V 15 A		
4 poles 2200 rpm		
Equivalent Circuit Parameters		
$R_s = 0.177\ \Omega$	$R_r = 0.318\ \Omega$	$R_m = 121.4\ \Omega$
$L_{ls} = 0.00205$ H	$L_{lr} = 0.0013$ H	$L_m = 0.0328$ H

signal is allowed to settle for 2 secs. The speed reference value was chosen to be of small size so that the controller response settles fast in terms of the simulation time. On a 486-66 DX2 computer, the total process of tuning all the three loops took about 6 hours to simulate, mainly because of the small integration step size (1 μs) used to simulate the complete drive system for about 8-10 secs of real-time. The values of the gains K_{pd}, K_{id}, K_{pq} and K_{iq} are also dependent on the maximum values of v_{ds} and v_{qs}, which are 1.0 in this case, whereas the values of K_{ps} and K_{is} depend not only on the speed command value, but also on the reference value of the i_{qs} command, which is the output of the speed PI controller.

Fig. 6.9(a) shows the intermediate passes of expert system based auto-tuning of the i_{ds} loop controller, with i_{ds} command and actual i_{ds} in the machine, as the expert system varies the K_{pd} and K_{id} values to obtain optimized performance. Intentionally, the expert system assigns initial values of the gains to be lower than expected, so as to maintain a stable system. The response profile is slightly overdamped with the initial gain values. The increment and decrement in the gains is a small fraction of their initial values, rather than a fixed amount, since the actual gain values can vary widely, depending on the motor and load parameters. The decrement value for the gains is 50% of their increment value, so that same search profile is not repeated while incrementing and decrementing the gains. As the gain values are increased in small steps, the response approaches critically damped, and suddenly becomes slightly more underdamped than required. A small decrement in the gains yields near-optimum result, which is satisfactory.

Fig. 6.9 (b) shows a similar performance with the profiles of i_{qs} loop tuning, which is done with the rated value of i_{ds} established in the motor and a 50% of rated i_{qs} command applied with 50% duty cycle, at 31.25 Hz frequency. In order to shorten the time of tuning, the process can be started with K_{pq} and K_{iq} values to be slightly lower than the *optimized* values of the i_{ds} loop PI gains, due to the similarity of the i_{ds} and i_{qs} current loops, which, as discussed earlier, are close to identical at very low speeds. The optimum values of K_{pq} and K_{iq}, as seen from Fig. 6.9(b) are very close to the tuned K_{pd} and K_{id} values respectively, which further confirms the fact that the two loops share similar dynamic characteristics.

Fig. 6.9(c) depicts the results of speed loop tuning through its intermediate passes, as the K_{ps} and K_{is} values are gradually varied by the expert system, starting from the overdamped case.

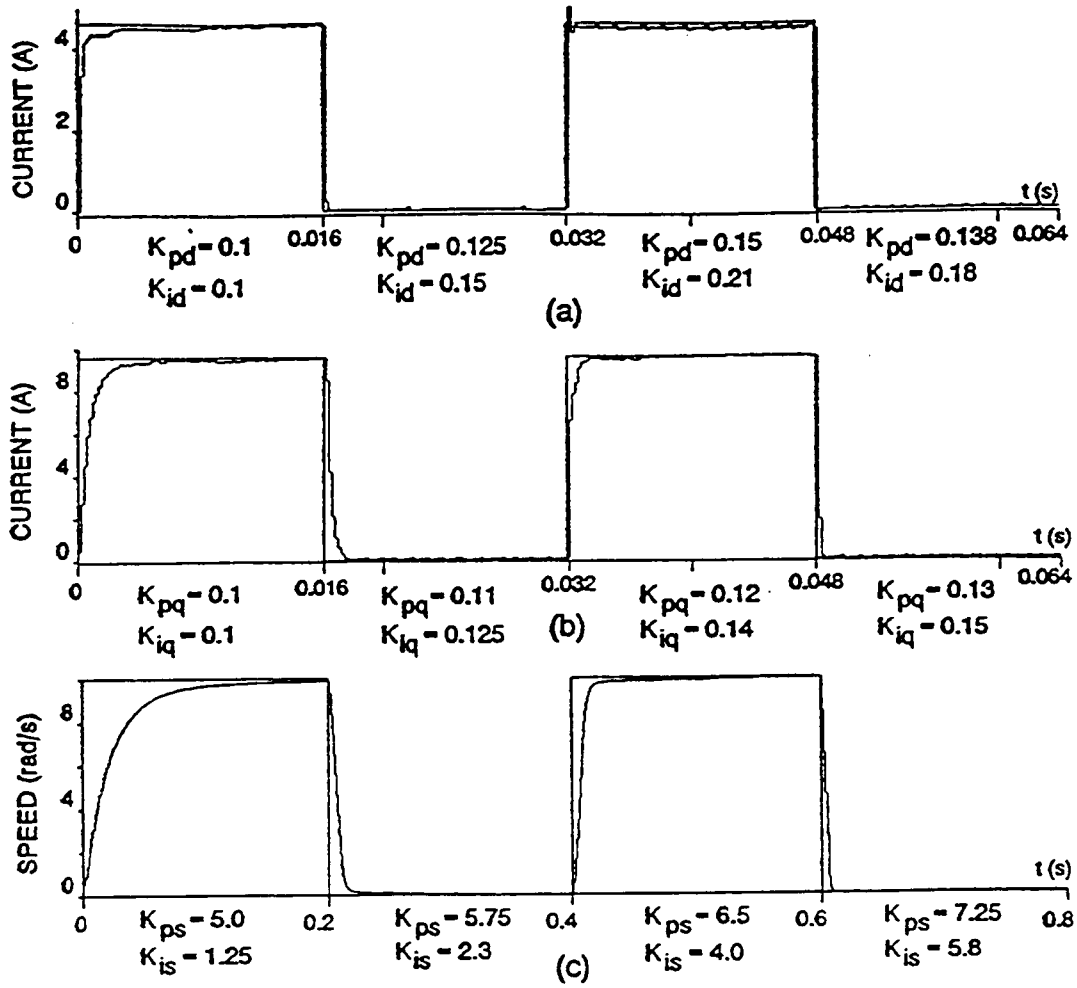


Fig. 6.9 Passes of Expert System Aided Auto-Tuning of Controllers

(a) I_{ds} Loop Tuning

(b) I_{qs} Loop Tuning

(c) Speed Loop Tuning

The inner loop PI gains are maintained at their tuned values during these optimization passes. As seen in the figure, the speed controller response slowly approaches the critically damped, as the gain values are tuned. The speed reference value has been chosen not only to maintain the simulation time low, but also to maintain the speed PI operation within its linear range, without saturating the output, where the closed loop control is lost.

6.6 Summary

The chapter describes the expert system aided automated controller design and auto-tuning for the AC drive system, with the example of the servo speed control application, where an indirect vector controlled induction motor drive is used. The chapter begins with a brief description on the control design methodology. Then the aspects of the tuning of controllers are described in detail. Next, the software structure for autonomous, hybrid simulation for the AC drive system, performed in SIMNON and DSP simulator, as well as the interface routines employed for achieving the simulation are presented. Rather than going into the actual implementation details of the software, the general methodology development aspects are highlighted, so as to explain the complexities of the software, as well as keeping the door open for applying the same principles to other types of AC drive applications. The knowledge representation in the expert system is discussed next in details, followed by the system performance, in which successful implementation of the expert system for automatic tuning of the controllers is demonstrated with the help of the results. The example of the IVC-IM drive system is used here to demonstrate the capability of the expert system to design and optimize the controllers for AC drive system, though the same principles can be extended, as said earlier, to other applications. One salient feature of this software is that the control software in C (internally compiled in DSP assembly language), and in DSP assembly language, once fine-tuned, can be directly downloaded to an actual experimental drive system, thereby freeing the AC drive designer from the tedious developmental tasks of the control software development. The simulation software is also designed to be autonomous so that if the user wishes to supersede the expert system in fine-tuning the controllers, he can still utilize the control simulation software, as it is modular and flexible in nature. It has been successfully shown that the expert system can be applied to perform the complex task of configuring the AC drive system for a particular application, designing the nameplate specs of the AC motor, designing and optimizing its power converter system and designing and fine-tuning its controllers.

CHAPTER 7

EXPERIMENTAL VALIDATION OF DRIVE CONTROLLER DESIGN

7.1 Introduction

Extensive discussion on the expert system based automated design of AC drive system, including the selection of drive configuration, design of motor nameplate ratings, design and optimization of power converter as well as controllers is presented in the previous chapters. Having designed the AC drive system, the expert system next downloads the optimized DSP code of the drive controllers to the actual experimental AC drive system. This chapter presents the experimental validation of the drive controller design. The AC drive system used for simulation study is used now for laboratory experimentation. A commercial 3 hp IGBT inverter was adopted for experimental study. Its original PWM modulator was replaced by a hardwired sine-triangle PWM modulator. The controller code in DSP 320C25, available from the expert system, was integrated with the rest of the I/O signal processing for currents and speed and host to DSP communication code developed later. The control voltage waves generated by the ARIEL DSP16 board were fed to the hardwired PWM modulator. The discussion begins with the description on the hardware design for the indirect vector controlled induction motor drive system. The various aspects of real-time software design are discussed next, followed by the results and discussion on the experimental performance.

7.2 Hardware Description

The experimental system contains a 3 hp commercial IGBT inverter (Toshiba model 130H1-2035), a 5 hp induction servomotor mounted with a speed encoder on the rotor shaft, a TMS320C25 based DSP board and associated interface circuit modules, such as the current low-pass filters and speed

sensor interface. Fig. 7.1 shows the experimental system in block diagram form.

7.2.1 Hardware PWM Modulator: In order to facilitate the PWM generation for the inverter, a hardware PWM modulator was employed, so that the drive control signals from the DSP can be supplied to the gate drivers of the IGBTs, achieving the DSP based control. For this purpose, The PWM modulator of the original inverter was replaced by the sinusoidal PWM modulator circuit shown in Fig. 7.2. The components of this circuit will be described in detail now.

Triangular wave generator: The triangular wave generator consists of an integrator and a hysteresis band comparator, as shown in Fig. 7.3. The zero crossing detector with hysteresis is fed the output of the integrator. The height of the triangular wave is determined by the width of the hysteresis band, and the triangular wave frequency is determined by the time constant of the integrator. The PWM frequency, which is the triangular wave frequency, is set to 10 kHz and its amplitude was preset to 4 V. The gain and offset adjustment are provided by a separate gain amplifier, whose positive and negative slope can be independently adjusted. The output of the triangular wave generator is fed to the sine-triangle comparator block through buffer stage.

Sine-triangle comparator: The DSP board generates the phase voltage references through software and outputs them through the DACs, which are subtracted from the triangular waveform and then fed to three zero crossing detectors with hysteresis. The outputs of the zero crossing detectors are processed in subsequent blocks to generate inverter gate switching signals. Differential input amplifiers are employed in this block to improve on the common mode noise rejection of the circuit. The FET-input wideband amplifiers are used in the circuit to avoid signal distortion and offset. Fig. 7.4 shows the circuit for the sine-triangle comparator.

Pulse Separator: The sine-triangle comparator generates three bipolar square waves. The negative halves of these waves are blocked and the positive half amplitudes are scaled to make them compatible with the digital logic circuits to which they are fed next. The pulse separator block, as shown in Fig. 7.5 has Schmidt trigger input stages avoiding any rise-time or fall-time-related incompatibility while interfacing the analog comparator with the digital logic circuits. The pulse separator outputs three *pairs* of switching waveforms for three phase legs of the inverter.

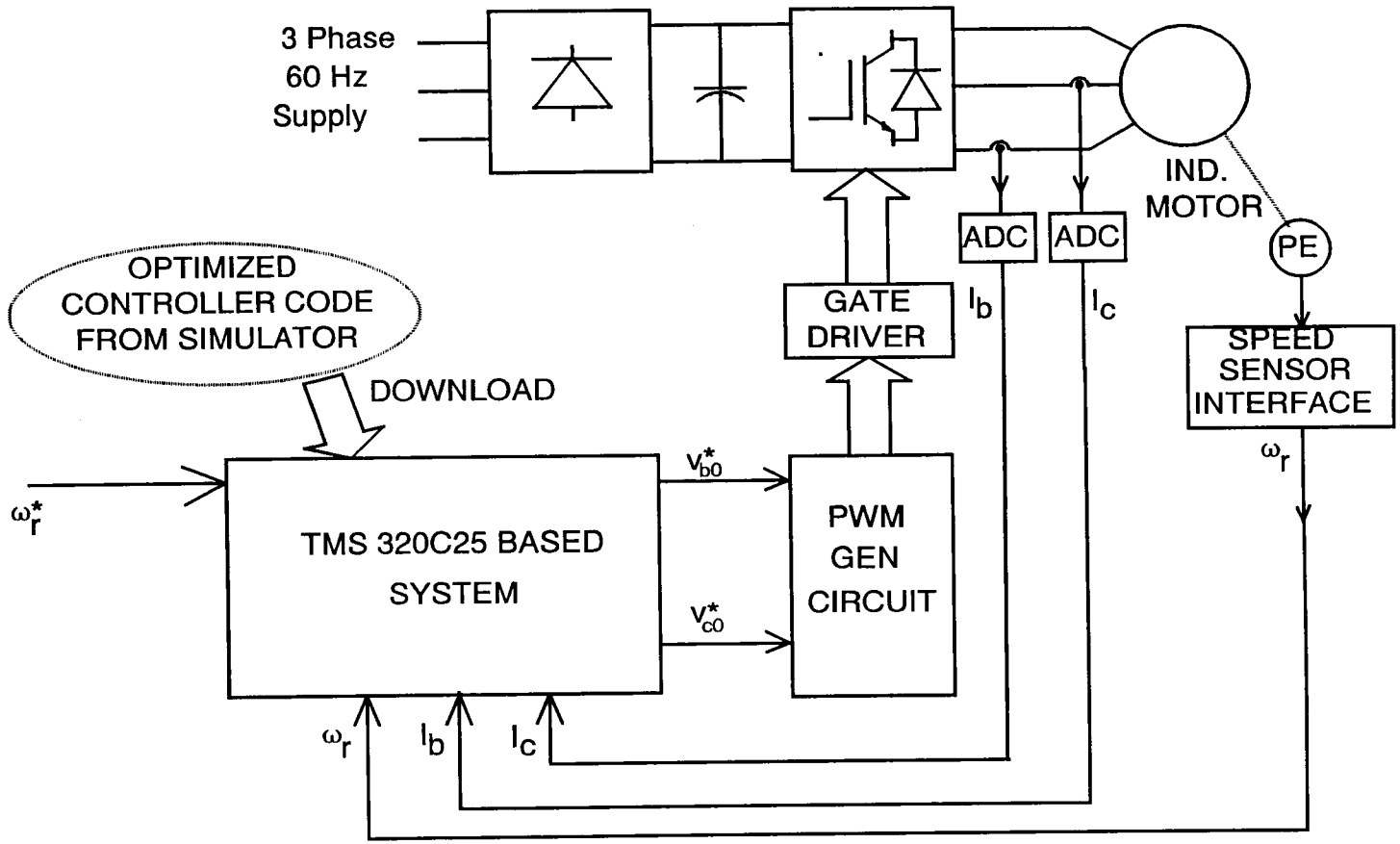


Fig. 7.1 Block Diagram of the TMS 320C25 Based Experimental Drive System

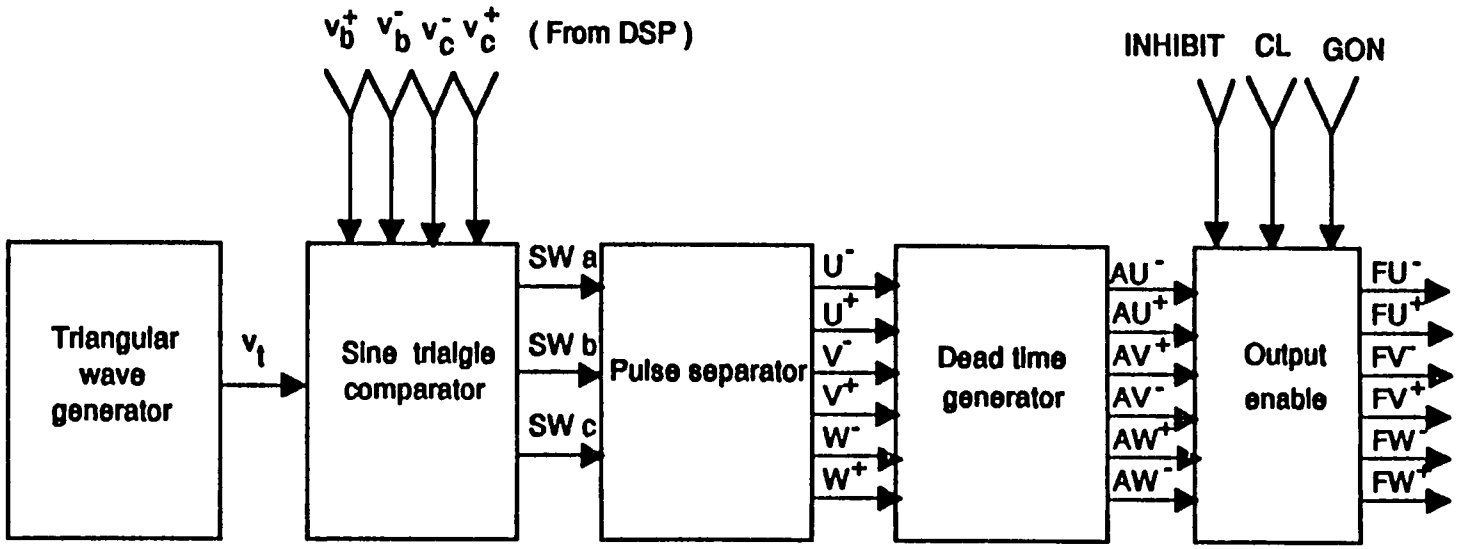


Fig. 7.2 Hardware PWM Modulator Block Diagram

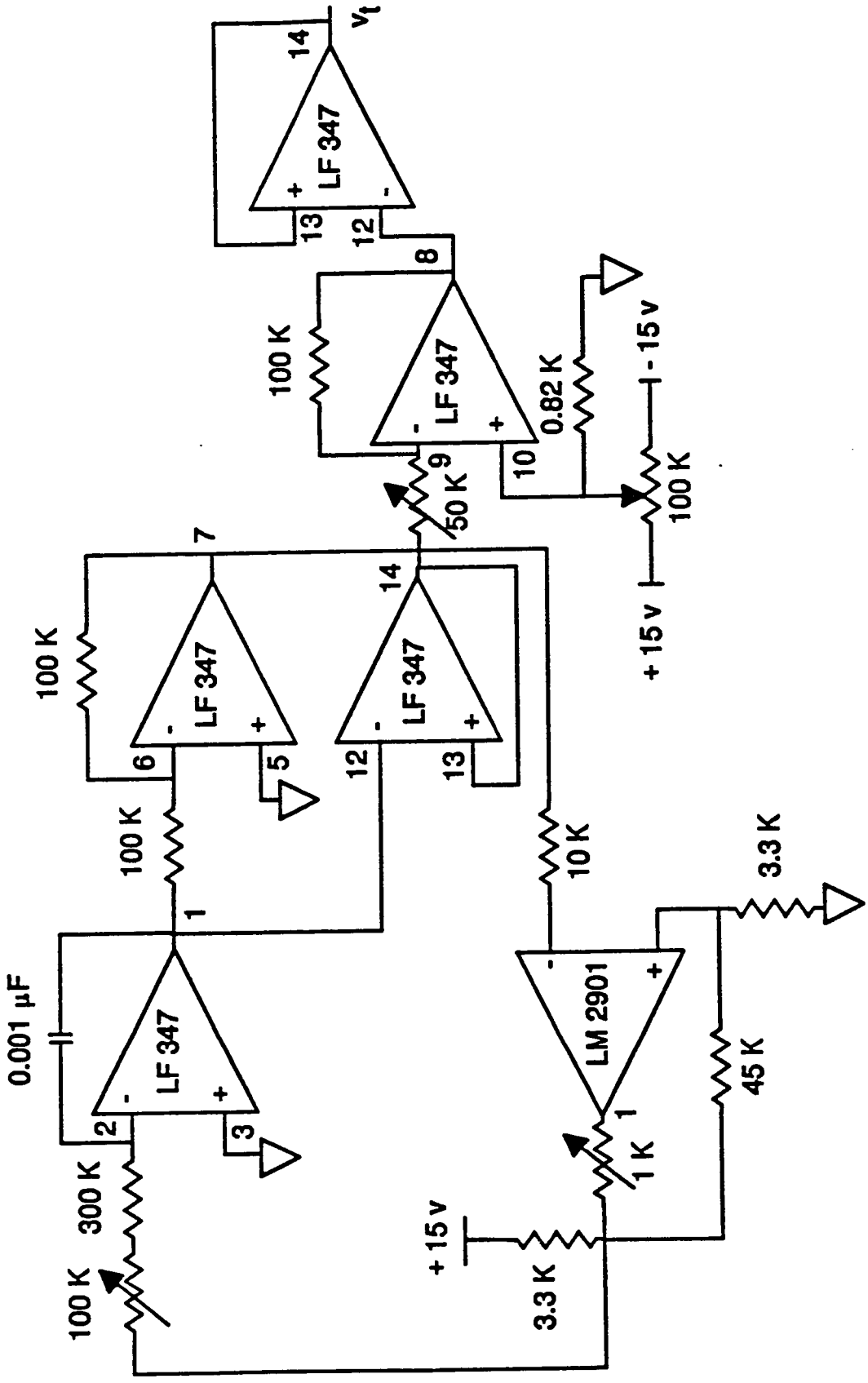


Fig. 7.3 Triangular Wave Generator Circuit Diagram

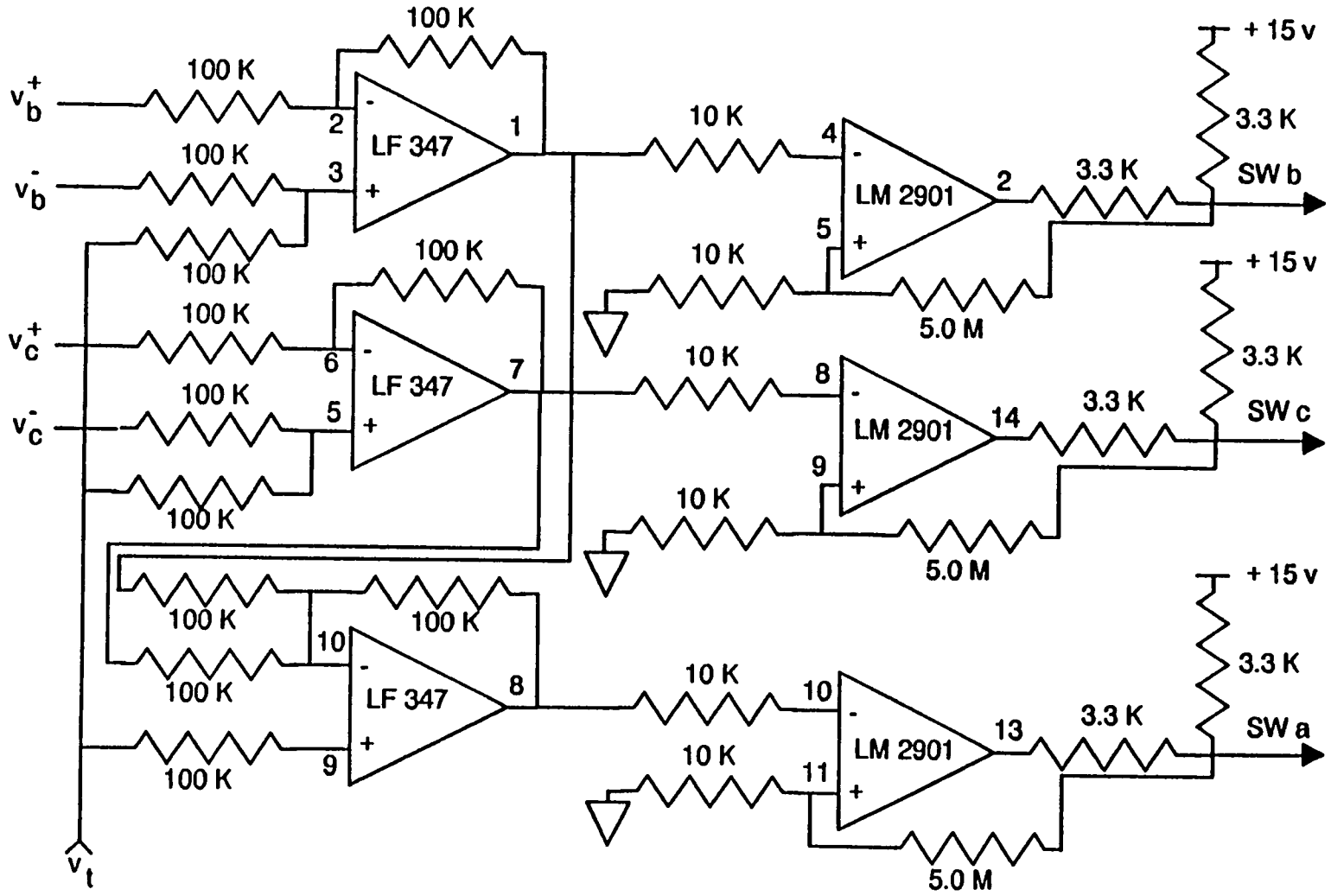


Fig. 7.4 Circuit Diagram of the Sine-Triangle Comparator

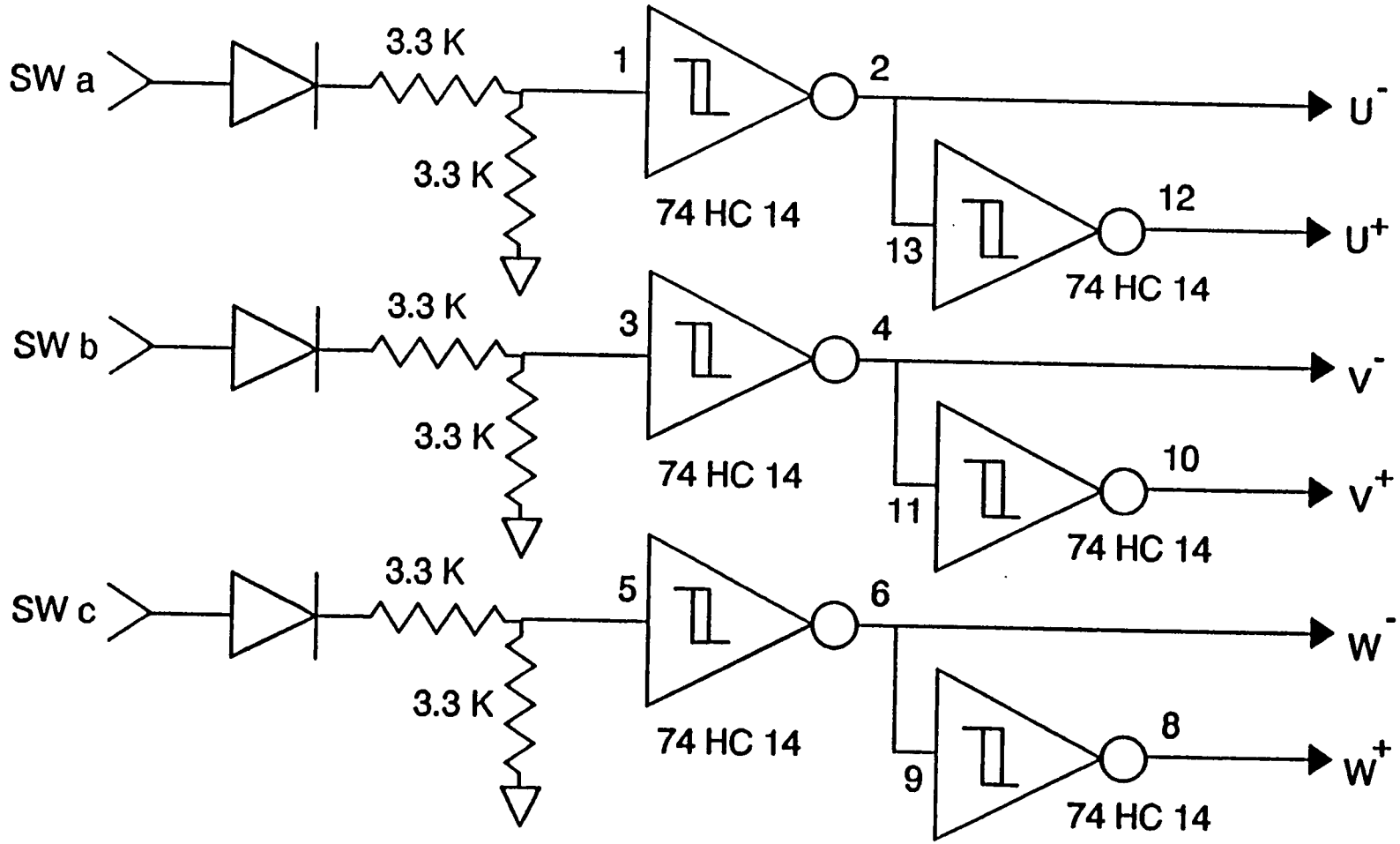


Fig. 7.5 Circuit Diagram of the Pulse Separator

Dead time generator: Before the base drive signals are fed to the upper and lower transistors, they should be separated by a short, dead-time, interval to avoid shoot-through type of fault. The dead-time is introduced at every rising edge of the base drive signals, and is about 3 μs. This delay is generated by feeding the outputs of the pulse-separator circuits to the inputs of rising-edge triggered monostables, after differentiating them, as shown in Fig. 7.6. The monostable time-period is set to 3 μs. The dead-time signals are then ANDed with the outputs from the pulse separator circuit to form the final base drive signals. In order to compensate for the propagation delay through the monostable stage, the outputs of the pulse separator are also delayed by passing them through two stages of non-inverting buffers before they are ANDed with dead-time signals.

Output enable: This block consists of a tri-stated buffer (Fig. 7.7) and its gate control circuit. The gate control circuit combines the protection signals from the original inverter to enable or disable the gate drives. The disabling of the gate signals is also achieved through DSP software. The GON signal from the inverter is latched to 0 when any of the earth fault, overvoltage or overcurrent conditions are detected. However, the signal CL is not latched, but is asserted to 1 only during the interval that the phase current of the inverter exceeds its current limit detection level. Thus, this signal is used to trigger a monostable with 50 μs pulsewidth, and this delayed output is fed to one of the control gates of the tristated buffer, whereas the GON signal is connected directly to another of the control gates. This ensures the disabling of the base drives of the inverter for at least 50 μs during the overcurrent condition. As mentioned earlier, the PWM modulator was interfaced to the base drive circuit of the original inverter. Fig. 7.8 shows the block diagram of the Toshiba inverter, with the PWM modulator interface, as the original PWM modulator outputs generated on-board the inverter are disabled and disconnected from the circuit. The protection signals GON and CL are tapped from the original inverter control board and used in the PWM modulator circuit as described earlier.

7.2.2 TMS 320C25 DSP Board

The major component of the control system of the AC drive is the DSP-16 [54] board, from Ariel Corporation. This is based on Texas Instruments TMS320C25, which is a 16-bit, fixed point, digital signal processor (DSP). The board is mounted inside the host PC, which is an IBM 286 compatible computer. The host computer facilitates the development and debugging environment, as well as the user interface during normal operation. The control board is equipped with two D/A as well as two A/D channels, which can be independently sampled to rates up to 50 kHz. The D/A channels are

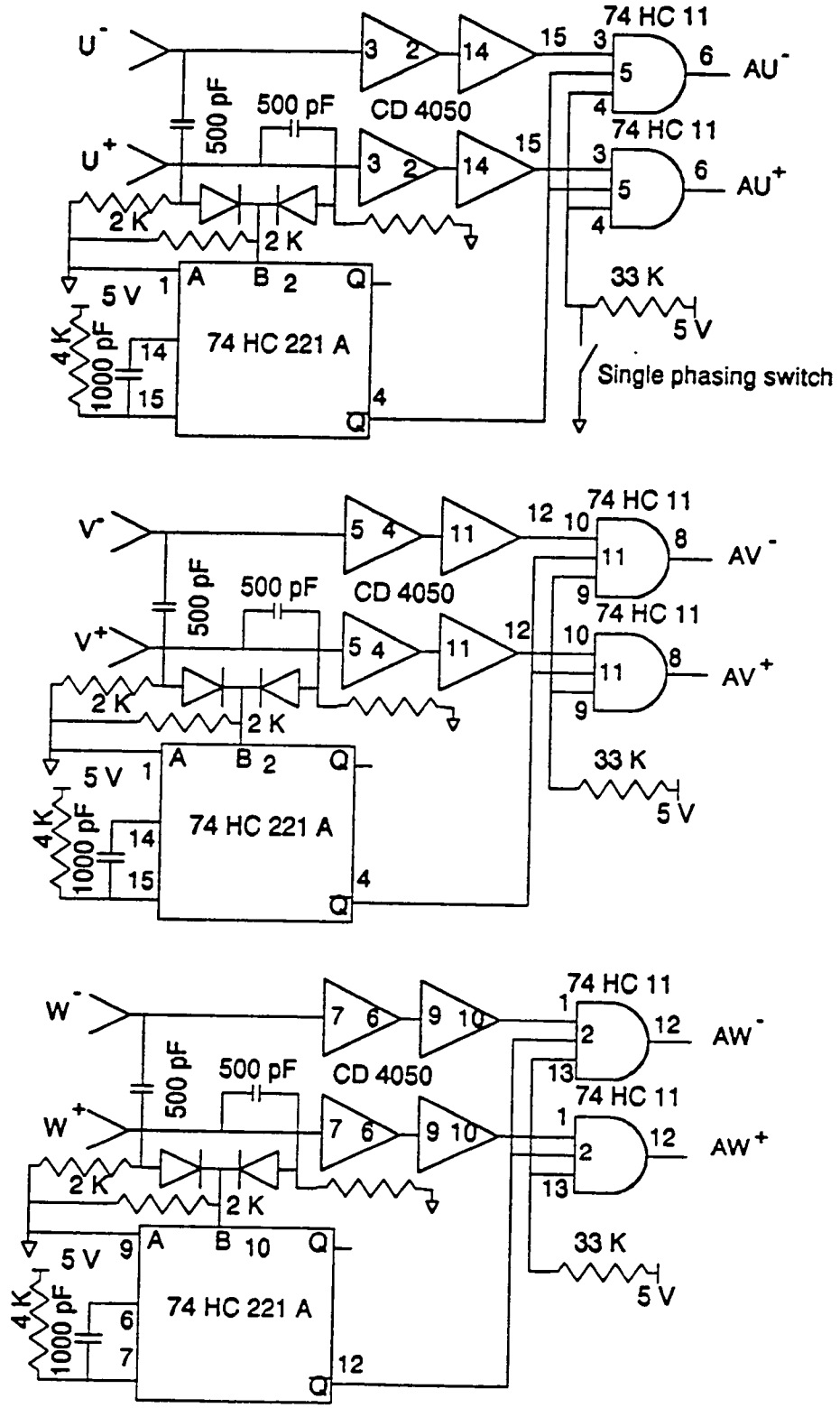


Fig. 7.6 Dead Time Generator Circuit Diagram

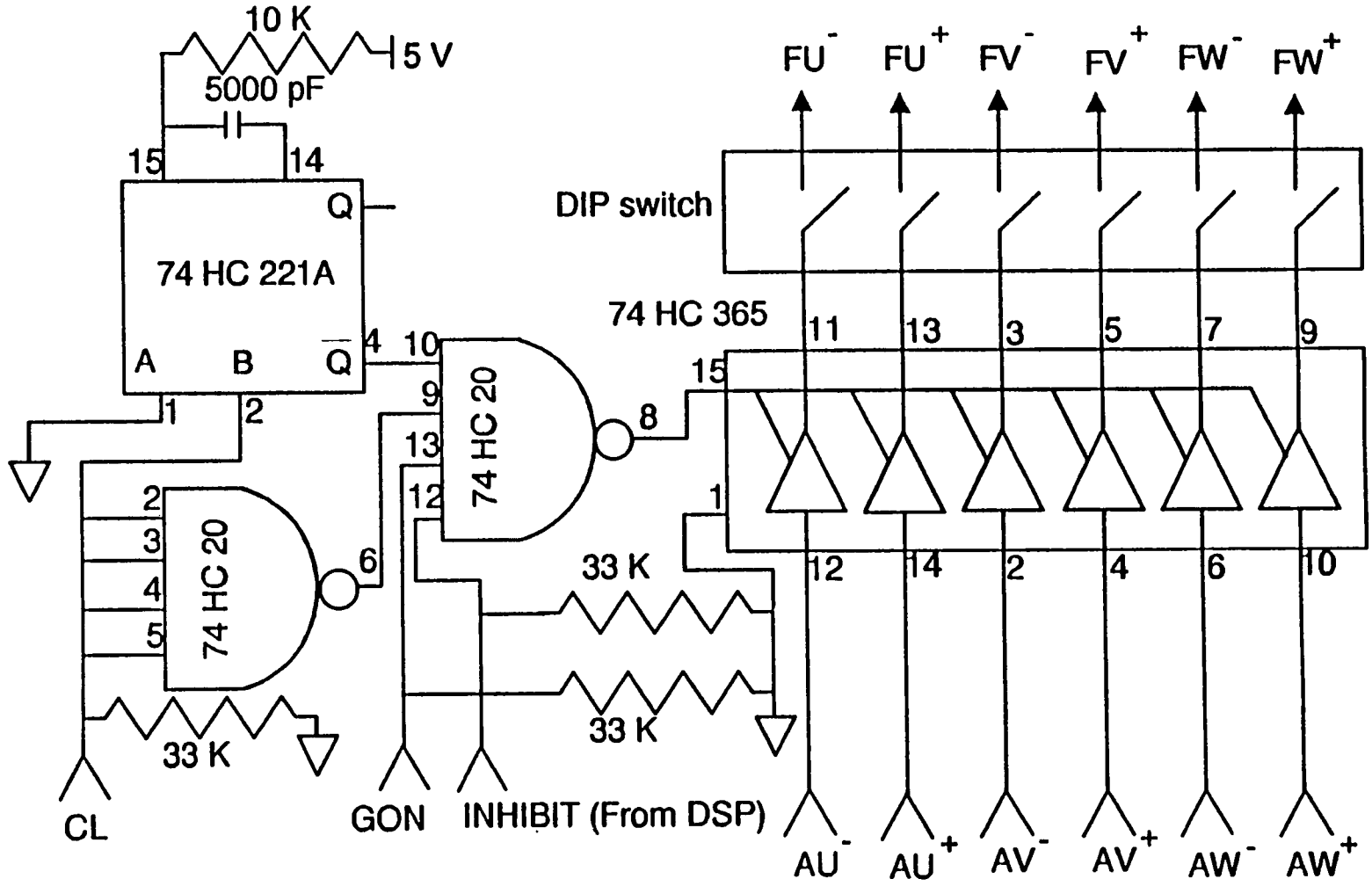


Fig. 7.7 Output Enable Circuit Diagram

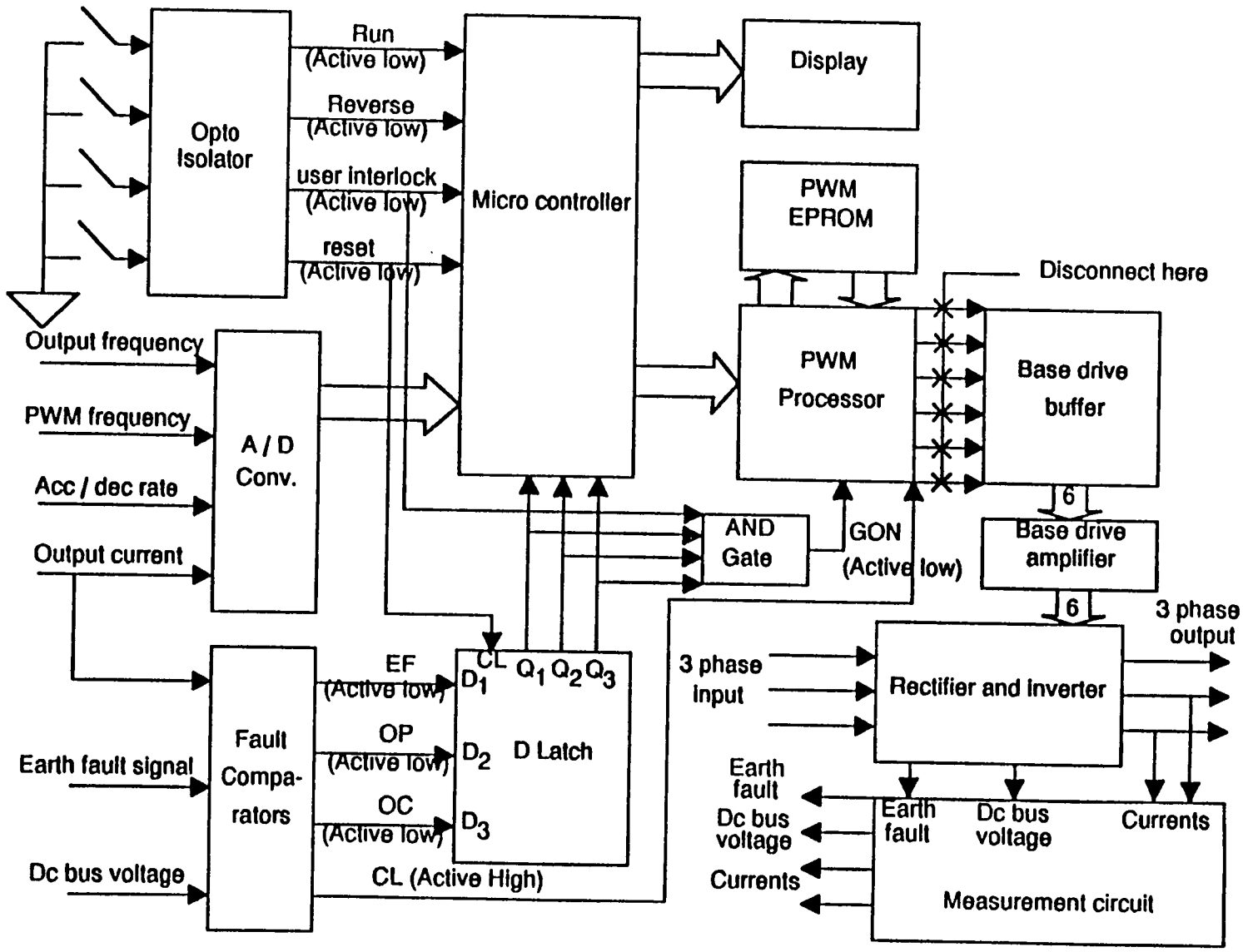


Fig. 7.8 Toshiba Inverter Block Diagram, with the new Hardwired PWM Generator

used to output the control voltages v_{b0} and v_{c0} for the PWM modulator, whereas the two A/D channels are tied to the motor current sensor circuits. The speed interface is directly connected to the TMS320C25 data, address and control bus, and is designed to operate as a parallel peripheral of the DSP. The DSP board provides 16 kilowords of program memory as well as 8 kilowords of data memory, as well as provides 256 kilobytes of buffer storage RAM, which can be used for data acquisition and display during the real-time control. The development and debugging environment also allows use of several host-to-DSP communication facilities such as loading the controller code, selectively enabling interrupts, setting the sampling rates for the I/O channels, as well as real-time communication with the DSP to change the parameter values, through the use of resident monitor / debugger software provided by the Ariel Corporation.

7.2.3 Analog Current Sensor Interface

As mentioned earlier, the DSP board allows two analog inputs, which are the motor line currents in this case. Although the analog board provides automatic offset null feature, the currents still require some preconditioning before being supplied to the DSP board. The preconditioning circuit is shown in Fig. 7.9. The inverter provides two Hall-effect current sensors which output voltage signals proportional to the line current. The sensitivity of the current sensors is approximately 3.3 V/A. The outputs of the current sensors are passed through low-pass filters with cutoff frequency at 5 kHz. The outputs of these current LPFs are connected to the DSP analog input channels through non-inverting buffer stages for impedance matching.

7.2.4 Speed Sensing and Control Interface

For the indirect vector controlled induction motor drive system, use of the speed encoder on the rotor shaft is inevitable. It is also critically important to have precise speed measurement for proper vector control operation. Actually, even an error of 1% in speed measurement can result in severe degradation of the drive performance.

The digital interface as well as measurement method described in this section provides high precision in a wide speed range. Fig. 7.10 shows the circuit diagram of the speed sensor interface. Basically, it consists of two 16-bit counters, one of which counts the shaft optical encoder pulses (θ) using a high frequency clock, whereas the other counter measures the time information (t). The speed is computed through software by dividing the encoder counter increment $\Delta\theta$ by the clock

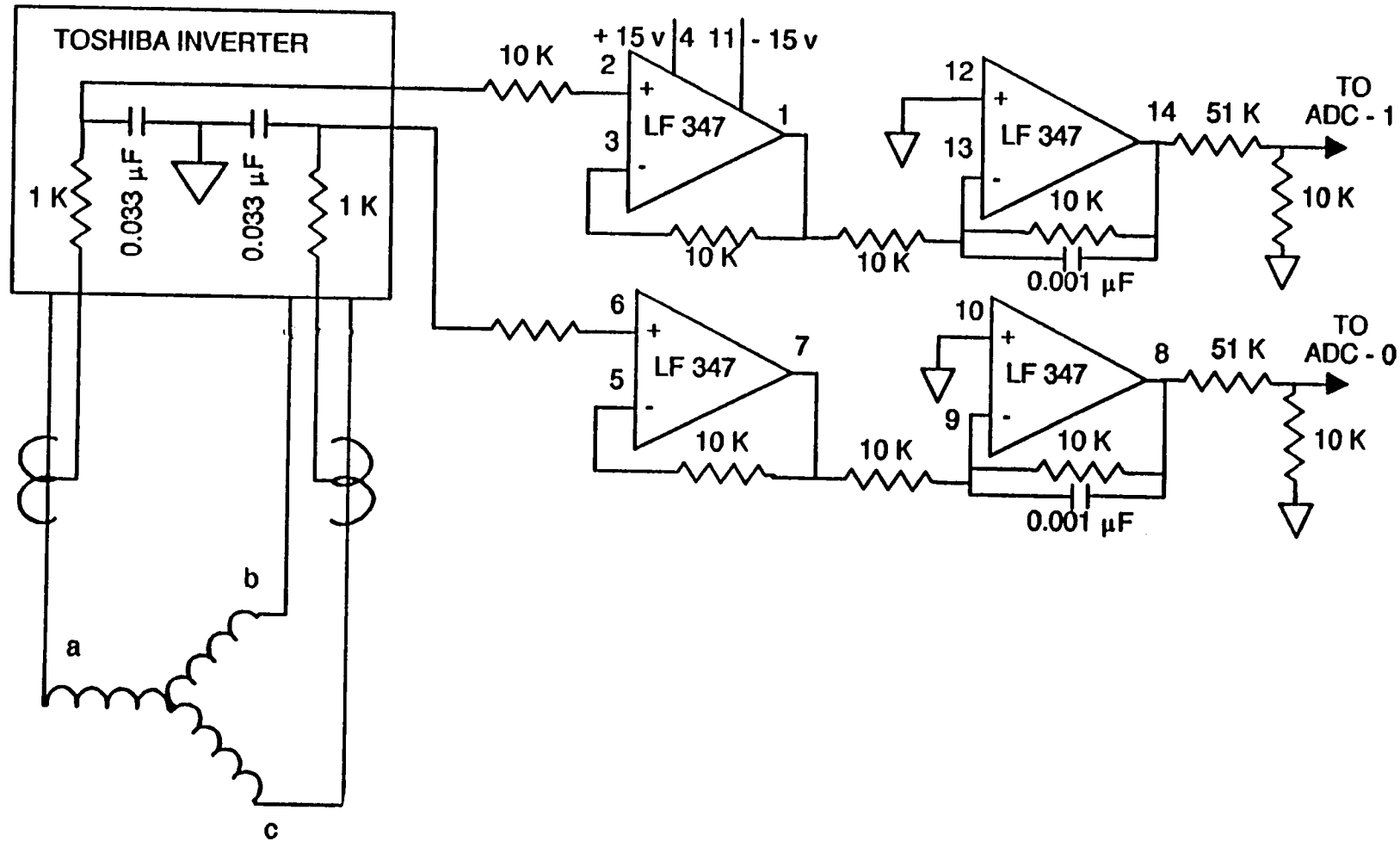


Fig. 7.9 Motor Line Current Interface Circuit

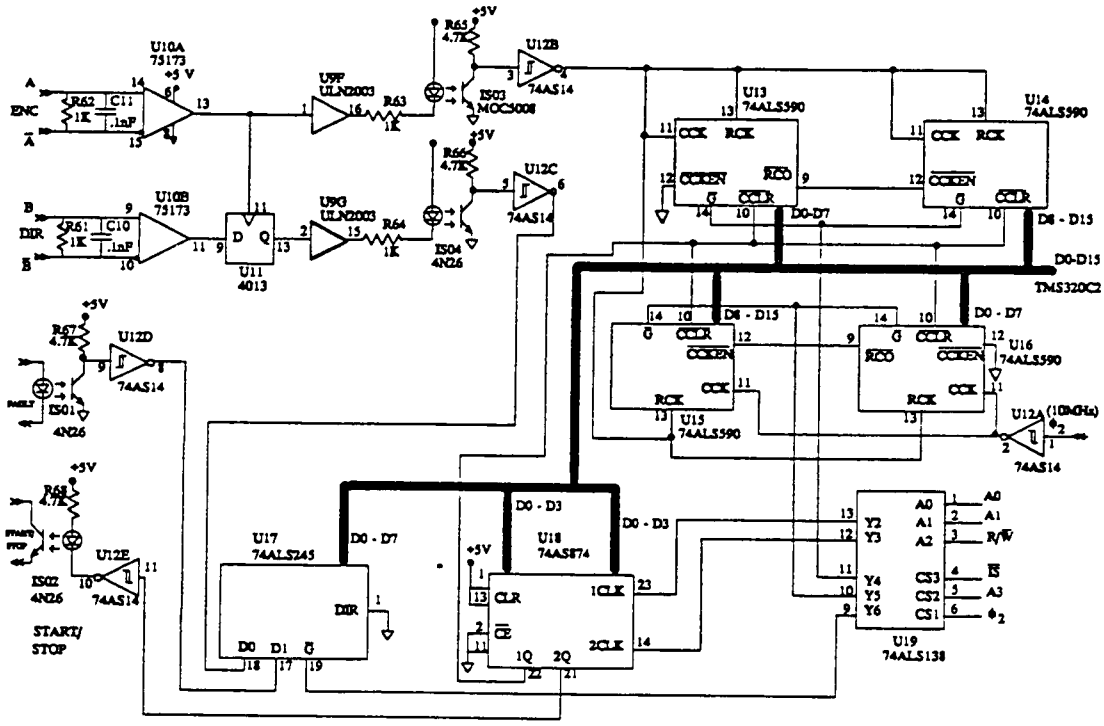


Fig. 7.10 Circuit Diagram of Speed Signal Interface

counter increment ΔT, with proper scaling. The software implementation will be explained later. Two 8-bit synchronous counters with latches at the outputs are cascaded for both the encoder and clock pulse counting. The counter contents are latched on to the output registers only at the positive edge of the register clock (RCK). Both the counters have their outputs tied to the 16-bit data bus of the DSP, whereas the port addresses for both of them are decoded through the use of a 3-8 decoder, 74LS138. The direction of rotation is also provided by the encoder, and the direction-sensing signal is read by the DSP through an octal bus transceiver, 74LS245.

7.3 Real Time Software Design

The complete drive control system, including the vector control, synchronous current control as well as the control for the modes of tuning, was implemented in DSP 320C25 assembly language and the speed control in C language, which was internally compiled to assembly language of DSP. Fig. 7.11 shows the software structure of the entire control system. Both the A/D and D/A converter operations are interrupt-driven and the maximum sampling frequency, as mentioned earlier, is 50 kHz per channel, and was employed to benefit from the embedded 20 kHz bandwidth input and output filters.

In order to allow sufficient time for the operation of the inner as well as the outer loop controllers plus the host-to-DSP communication, the sampling time for the execution of inner loops was chosen to be 200 μs, and the outer, speed controller was implemented with the sampling time of 1 ms, both of which are the same as the values employed by the expert system based controller simulation.

7.3.1 Basic Vector Control Functions

The implementation of the indirect vector control requires the generation of unit vectors, sin θ_e and cos θ_e, from the speed signal ω_r and the slip frequency ω_{sl}. The value of θ_e is obtained by the discrete integration of ω_e, using double precision to ensure accurate computation at low speeds. A sine table with a resolution of 5.689 bits/degree was initially constructed and stored in program memory. The sin θ_e and cos θ_e values are obtained by simply scaling down the θ_e value to the interval [-1024, 1024] bits and then retrieving the proper value from the sine table, using the table look-up method. The unit vectors are next used to calculate the stationary reference frame voltages v_{ds}^{s*} and v_{qs}^{s*}, from which the stationary reference voltages v_{b0} and v_{c0} are obtained. Phase a voltage is obtained by addition of the b and c phase reference voltages and negating the result, for the

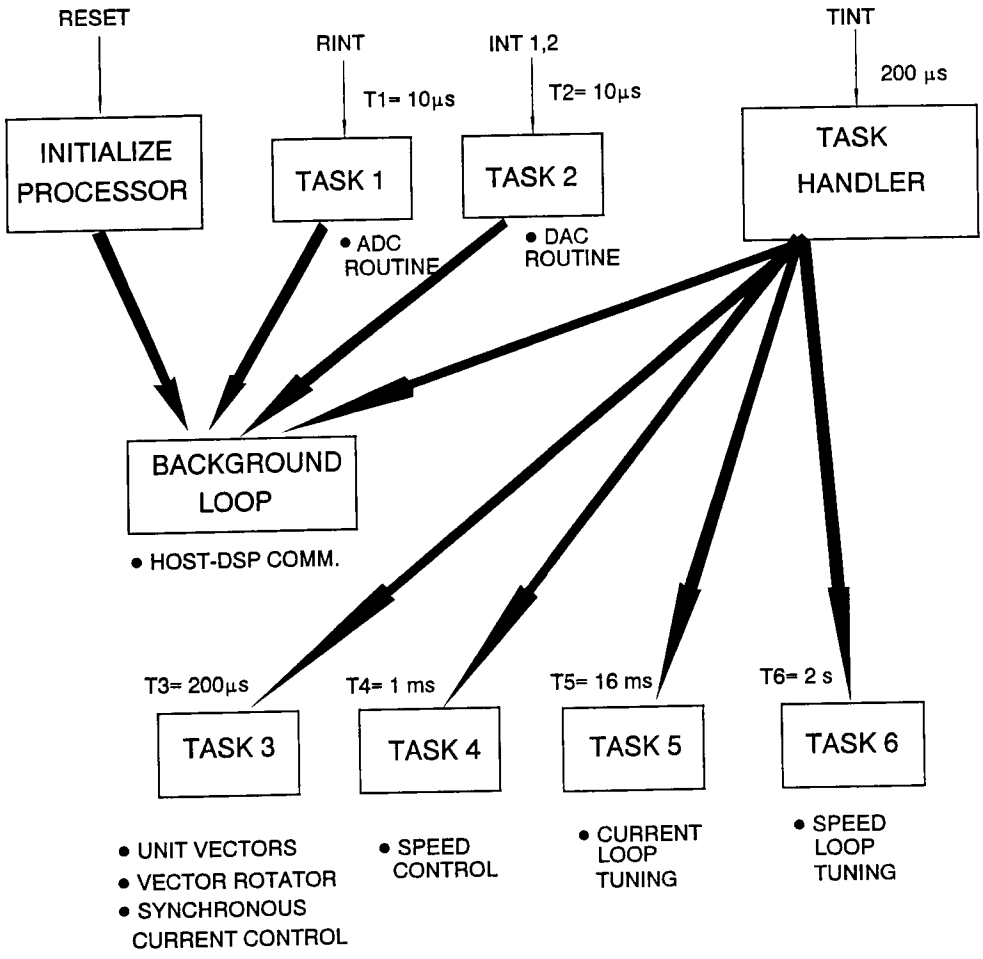


Fig. 7.11 Software Structure of the IVC-IM Drive Control

balanced phase voltage operation. The unit vectors are also employed to perform the inverse vector rotation of the motor currents, to obtain synchronous reference frame feedback currents, i_{ds}^{e} and i_{qs}^{e}.

7.3.2 Synchronous Current Control

As described above, the feedback current values in the synchronously rotating reference frame are produced by the vector control functions. The command values for these currents are fed to the system during the tuning through the keyboard. The PI controllers in the i_{ds} and i_{qs} current loops are also implemented in the DSP based control software. Actually, the optimized code for the vector control as well as current and speed control for the IVC-IM is downloaded from the expert system as is. In the implementation of the controllers, scaling of the variables is employed so that all the quantities remain within [-1, 1] interval in their per-unit values, for the fixed-point 16-bit implementation for the DSP 320C25. The modes of tuning, as well as the generation of the square wave command quantities is implemented in another mode control routine, which will be described later.

7.3.3 Control of Modes of Tuning

In the expert system based implementation, the expert system generates the command quantities for the controller being tuned during each pass of the optimization of the controllers. However, in the real-time experimental system, this operation is achieved by another routine, called the command square wave generator. This routine implements the following functions:

- Generation of square wave commands during the tuning of i_{ds}, i_{qs} and speed loops
- Implementation of the varying control topology during different modes of tuning
- Inhibition of gate control signals through software
- Computation of speed and current error inputs for the respective controllers
- Establishment of rated i_{ds} value in the synchronous reference frame, in the modes of i_{qs} and speed control tuning, before applying the i_{qs} and speed command square waves.

The frequency of the current square wave references is set to 31.25 Hz, which facilitates 16 ms of response acquisition time at every transition of the reference signal. The frequency of the speed reference signal is set to 0.25 Hz, allowing 2 seconds of response acquisition time at every speed

reference transition. The duty cycle of the reference waves is kept fixed at 50%, whereas their amplitude can be altered through the keyboard, with the help of the host-to-DSP communication routine.

7.3.4 Speed Computation

The basics of $\Delta\theta/\Delta T$ computation method for speed have already been explained in the discussion of the speed interface hardware. The same principles are used here, as indicated by the speed computation flowchart of Fig. 7.12. The basic sampling interval of 200 μs is expanded with the help of software timer to 1 ms, which is used to execute the speed control. The routine allows the extension of the usual sampling interval of 1 ms up to 5 ms, for accurate measurement of very low speeds. The speed is read to be zero if no change occurs in the angle count $\Delta\theta$ at the end of 5 ms. Under normal operation, $\theta_{new} > \theta_{old}$, and the speed computation proceeds as indicated. For operation in both directions, the direction signal must be probed, and the speed polarity corrected accordingly. Robust operation of the algorithm is achieved by imposing a pre-defined maximum value of speed, ω_{max}, to the speed change. This ensures the safe system operation even in the case of hardware malfunction in the form of failure of encoder or failure of one of the interface circuit components.

7.3.5 Speed Controller Routine

The optimized code for the C-based speed controller routine used for the experimental drive system was directly downloaded from the expert system for controller tuning. The C-based implementation was chosen since the controller execution rate was once every 1 ms. This time was sufficient for the slow-responding mechanical dynamics of the drive system, as discussed in earlier chapter. The C language based implementation allows usage of floating point numbers and quantities for PI controller, thereby avoiding the tedious and sometimes difficult scaling process. The control code was so designed as to implement the speed controller during the complete system mode, when the tuning of the inner current loops was complete and the outer speed loop was being tuned by varying the PI gain values. In order to facilitate smooth operation of the 5 hp induction machine fed from the 3 hp IGBT inverter, the limiter at the output of the speed PI controller was designed to limit the i_{qs} value to 6.5 Amps, which, when vectorially added to the 9.216 A of i_{ds} rated value, would yield the continuous maximum current capability of about 11 A from the 3 hp IGBT inverter.

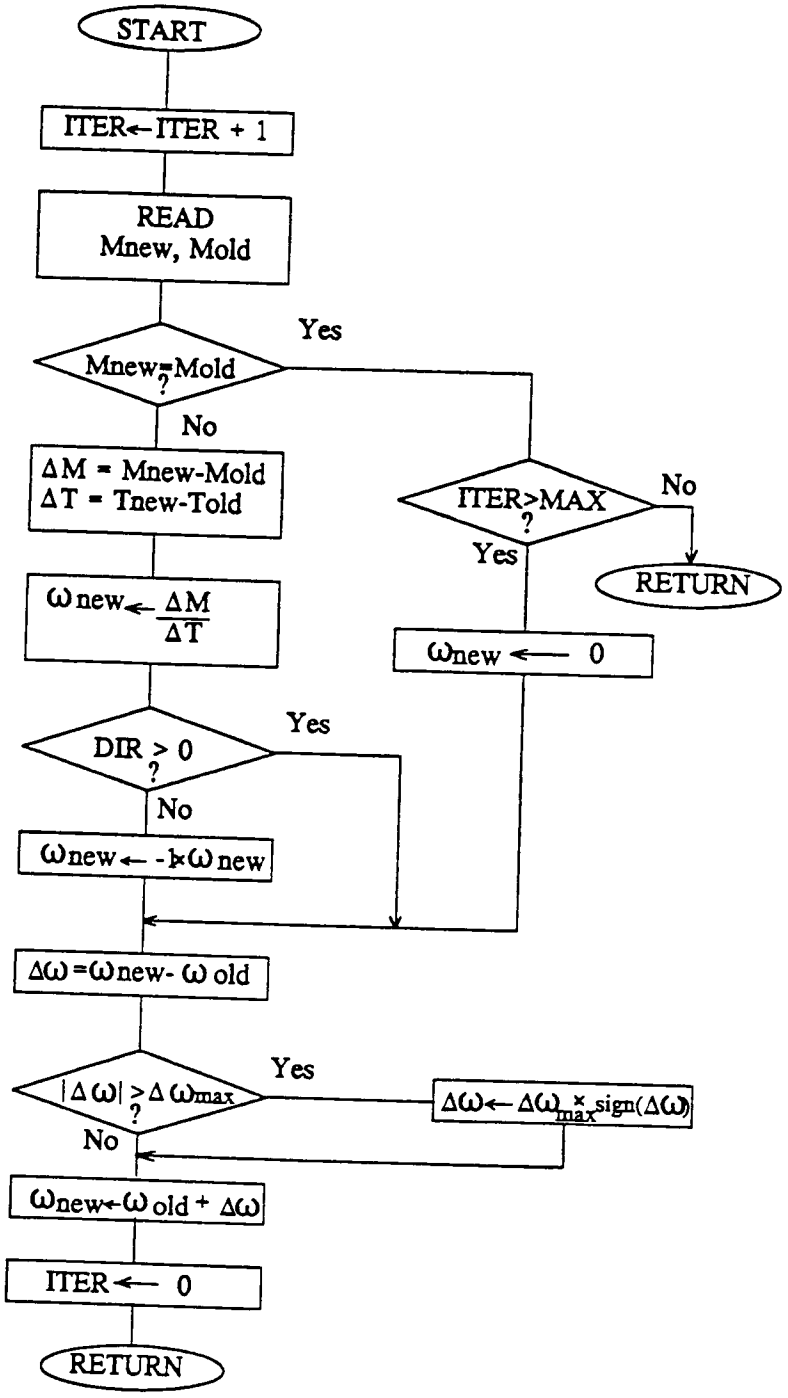


Fig. 7.12 Speed Measurement Flowchart

7.3.6 Host to DSP Communication Routine

This routine controls the data flow between the host PC, which is a 10 MHz 286-based computer, and the DSP board, without interrupting the real-time operation of the system. It was primarily designed to work under the development environment provided with the DSP 16 board. Some of the commands passed to the DSP are essentially debugging tools, whereas others are used as permanent control functions. This routine is used for passing to the DSP the following information:

- The start / stop command for the drive
- The command for the mode of controller tuning
- The information on the variables to be acquired during the real-time operation of the drive, for their later display
- The amplitudes of the command signals for i_{ds}, i_{qs} and the rotor speed
- The values of the control gains K_p and K_i, during the tuning of various controllers
- Command for the DSP to begin the data acquisition during the real-time operation of the drive

As shown by the Fig. 7.11, the host to DSP communication routine is executed in the background, as it has the least priority.

7.4 Experimental Study

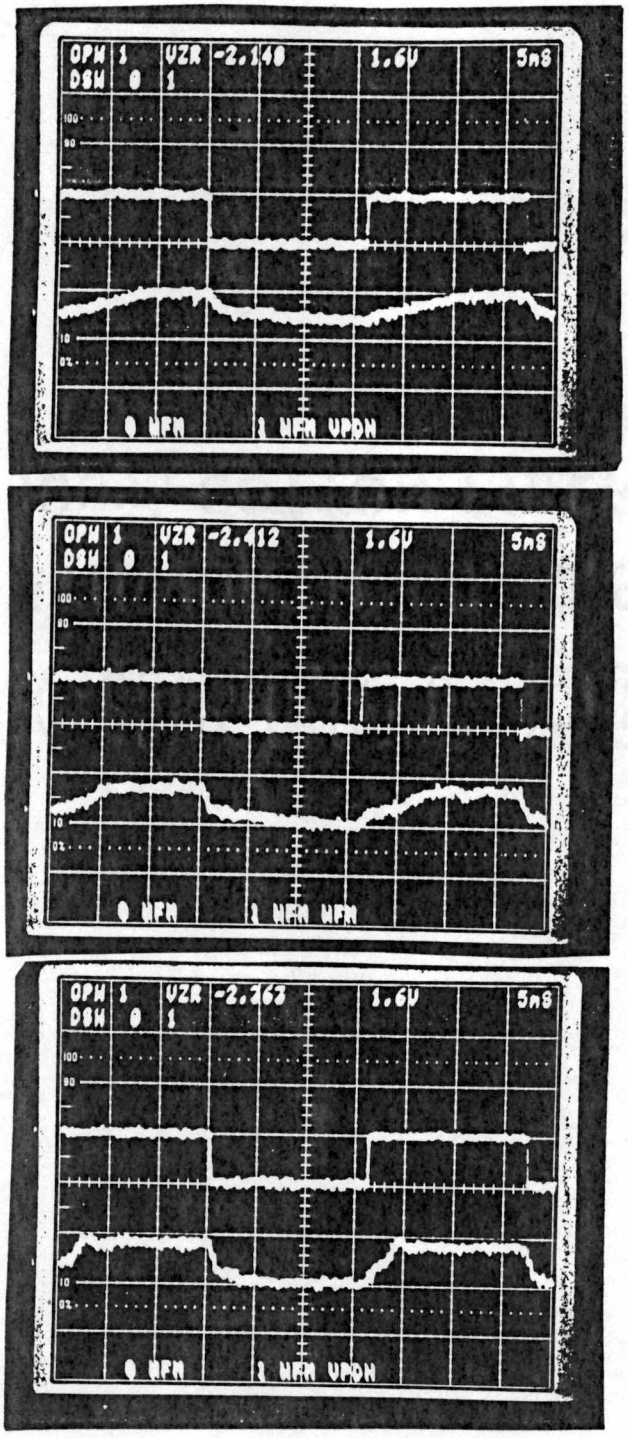
After careful software and hardware systems integration and debugging, the drive system was tested with a 5 hp induction servomotor, rated at 2200 rpm, 180 V, whose parameters were given in Table 6.1. The aim of the experimental study was to verify the control gain tuning for the current and speed controllers, which was earlier performed through expert system based software. Thus, as far as possible, one-to-one correspondence was maintained between the experimental verification procedure and the process followed by the expert system, so that the results obtained both through the design software and experimentation can be correlated. Thus, the laboratory system was tested for the three modes of tuning, namely, i_{ds} loop tuning, i_{qs} loop tuning and the speed loop tuning, in that order. Since the idea here was to validate the controller design, the drive testing was done at no load, to allow the operation of the 5 hp induction motor fed from the 3 hp IGBT inverter. During validation of each mode of tuning, the PI controller gains are varied in the same manner the expert system approaches the tuned values, and the responses of current loop controllers as well as the

speed loop controllers are obtained.

7.4.1 I_{ds} Loop Tuning Performance

In this mode, the i_{qs} command value is set to zero, whereas the i_{ds} command is generated by the DSP. The amplitude of the command current i_{ds} is set to 50% of its rated value, i.e., 4.608 A. Fig. 7.13 shows the i_{ds} current loop controller responses obtained during the passes of its tuning, when the P-I gain values are gradually varied. In all of the responses, the unfiltered component of switching frequency-related ripple is seen to be reflected, making the response slightly noisy. Fig. 7.13(a) shows the response clearly to be highly overdamped, with the rise time of 15 ms. The PI gain values were $K_{pd} = 0.08$ and $K_{id} = 0.085$. Fig. 7.13 (b) shows the response to be slightly improved, though still overdamped, with the increased PI gains to $K_{pd} = 0.1$ and $K_{id} = 0.106$. The rise-time of the response was seen to have improved to 10 ms. A slight distortion appeared in the actual i_{ds} current flowing in the machine near the transition from zero value. This was further deteriorated as the P-I gain values were increased, as shown in Fig. 7.13 (c) shows the optimized i_{ds} controller response profile. In this case, the PI gain values used were $K_{pd} = 0.128$ and $K_{id} = 0.16$. The response was slightly underdamped, and the rise-time markedly improved to 4 ms. The optimized values of K_{pd} and K_{id} found by the expert system were 0.138 and 0.18 respectively. The slight discrepancy in the experimental and simulated values can be attributed to the fact that the maximum values of v_{ds} and v_{qs} reference voltages, the outputs of the synchronous current PI controllers, were limited to 0.33 p.u., as against the 1.0 p.u. used in the simulation. A reduction in the values of v_{ds} and v_{qs} magnitudes means that even a slight change in the values of K_{pd} and K_{id} affects the controller responses greatly. Besides, the scaling of parameters in the experimental system is slightly different from the one used in the simulation, as in the case of the experimental drive system, the key scaling factors, such as the currents, speed, etc., are fixed by the hardware. This, combined with the fixed-point nature of the DSP's implementation of controllers, introduces errors in the calculations related to truncation and rounding off of quantities, which are reflected in slightly different values of PI gains to achieve similar type of response.

Another peculiar behavior in the i_{ds} controller response is observed distinctly in the cases of Fig. 7.13(b) and 7.13(c), where there is slight distortion in the actual i_{ds} wave when it is rising from zero at the rising edge of the command current. The effect is more enhanced in Fig. 7.13(c). This seems to be the result of the initial saturation of the P-I controller due to high current error, which prevents the controller from supplying adequate voltage in the d-axis equivalent circuit, except at the very



(a)

(b)

(c)

Fig. 7.13 Experimental Passes of I_{ds} Current Loop Tuning Process, Showing I_{ds}* (Upper Trace, 5 A / div) and I_{ds} (Lower Trace) with

(a) $K_{pd} = 0.08$, $K_{id} = 0.085$
(b) $K_{pd} = 0.1$, $K_{id} = 0.106$
(c) $K_{pd} = 0.128$, $K_{id} = 0.16$

beginning, to impress the required current, which rises in an open-loop manner. As the P-I gains increased, this behavior was observed earlier in real-time. This effect is observed only during repetitive application of square command pulses as a look at Fig. 7.14(c) reveals a smooth-rising i_{ds} current wave, with the rated i_{ds} command applied, before i_{qs} command is applied, as will be discussed later.

7.4.2 I_{qs} Tuning Performance

Once the i_{ds} current loop is tuned, the values of K_p and K_i for this loop are set to their optimum values, and the i_{qs} loop tuning begins. Fig. 7.14 shows the results of three passes of i_{qs} tuning process. As mentioned earlier, the software is so designed that every time a new i_{qs} command is given, i_{ds} is established at its rated value, i.e., 9.216 A, before i_{qs} square command is imposed on the circuit. The magnitude of the i_{qs} command is set to 6.6 A, which is the maximum permissible value with the limited current of the 3 hp inverter feeding the 5 hp motor (for which, the rated i_{qs} is about 12 A). The i_{qs} tuning is done with the rotor blocked, so that the machine is prevented from accelerating to dangerously high speeds in the torque control mode. Fig. 7.14 (a) shows the i_{qs} profile with the K_{pq} set to 0.09 and K_{iq} = 0.093. The response is seen to be overdamped, with the rise-time to be approximately 5 ms. This is expected since the torque sensitivity of the system is maximum with the rated i_{ds} value established. Fig. 7.14 (b) shows the i_{qs} response profile, when the K_{pq} is increased beyond its optimum value to 0.15, and K_{iq} is increased to 0.2. The response of the i_{qs} is considerably underdamped, with the overshoot of about 20%. The rise time is improved to be about 2 ms. However, the amount of underdamping may make the system operation jerky under transients. In order to correct this, the control gains are slightly reduced to K_{pq} = 0.13 and K_{iq} = 0.15, to obtain the responses in Fig. 7.14 (c), where the system behavior improves slightly, and the rise time increases to about 3 ms. This trace also shows the fact that the controller waits for 50 ms for the rated value of i_{ds} to be established, before applying the square command of i_{qs}. The upper trace shows the actual i_{ds} being established in the motor, and the lower trace is the actual i_{qs} in the motor.

7.4.3 Speed Controller Tuning

Once the tuning of the inner, faster, current control loops is done, the K_p and K_i values for these are set to the optimum values, and the output of the speed loop is the command value of the i_{qs} controller. The speed loop tuning begins with the speed K_{ps} and K_{is} set to values that produce overdamped response. Fig. 7.15 shows the profiles of the speed loop tuning, when the speed

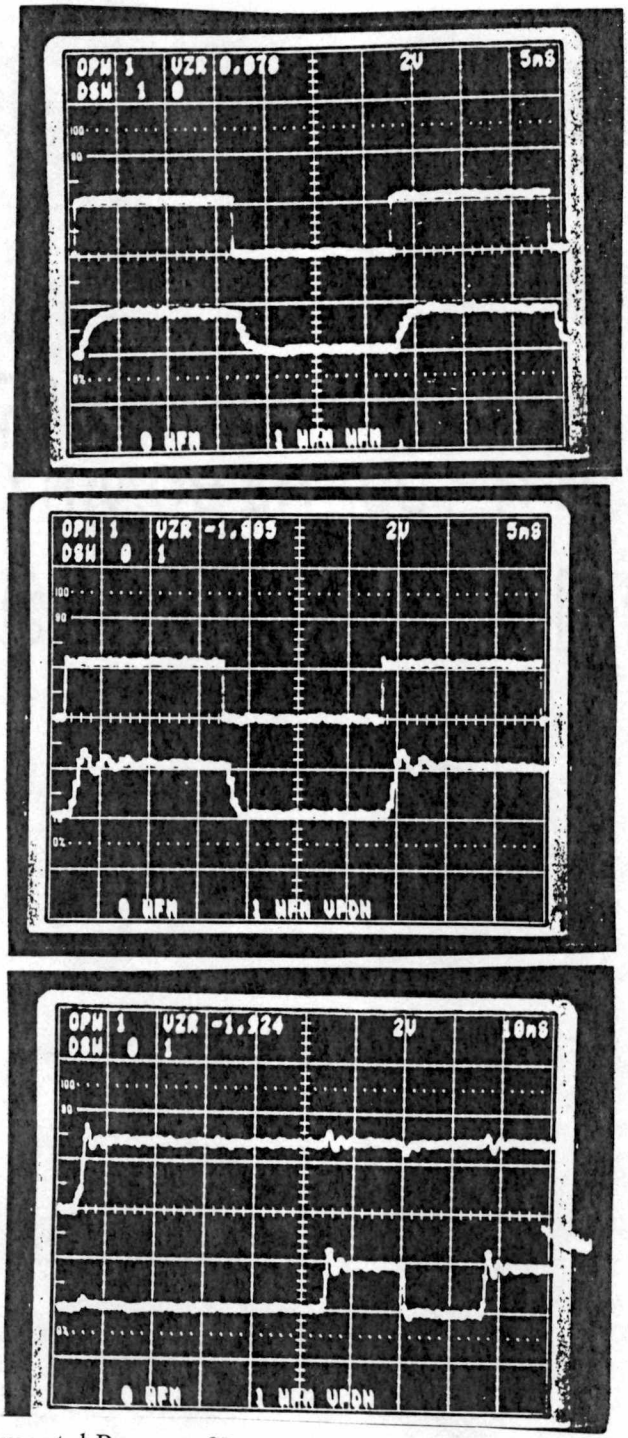


Fig. 7.14 Experimental Passes of I_{qs} Current Loop Tuning Process, with Tuned Values of K_{pd} and K_{id} employed. Upper Trace is I_{qs}* for (a) and (b) and $I_{ds(rated)}$ for (c), all 5 A/div, whereas Lower Trace is I_{qs}

(a) $K_{pq} = 0.09$, $K_{iq} = 0.093$
(b) $K_{pq} = 0.15$, $K_{iq} = 0.2$
(c) $K_{pq} = 0.13$, $K_{iq} = 0.15$

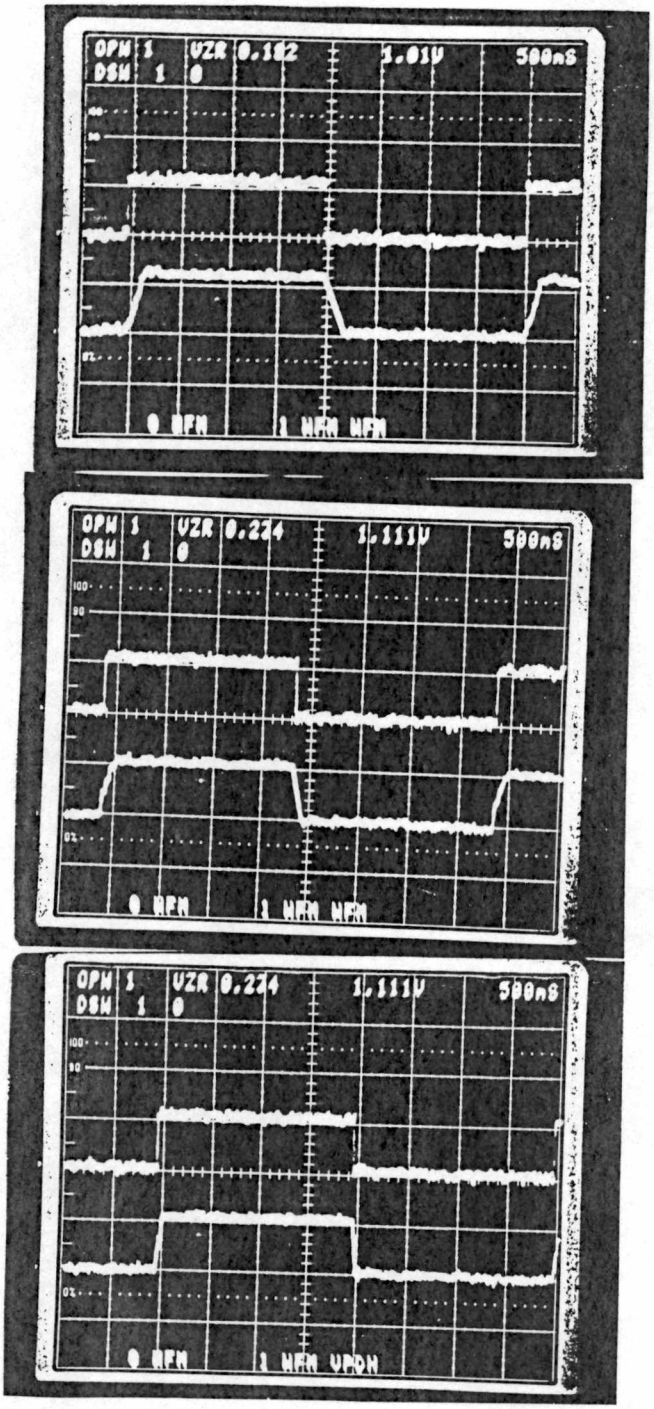


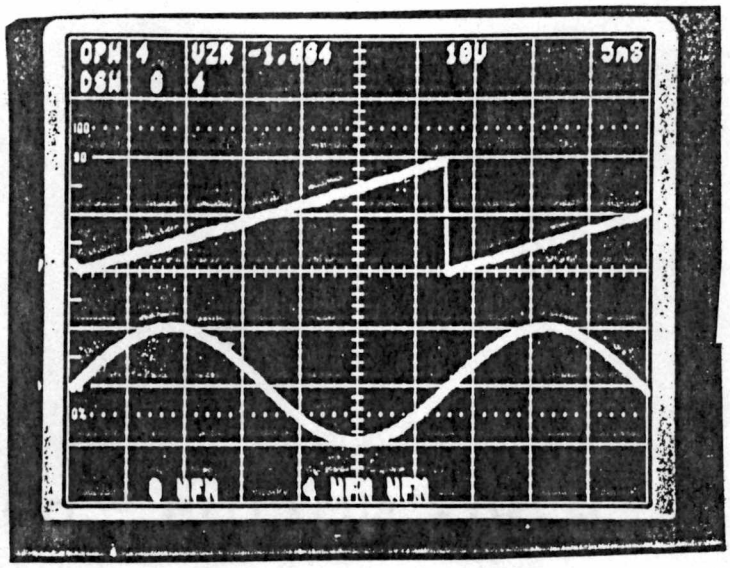
Fig. 7.15 Experimental Passes of Speed Loop Tuning, with Tuned Values of K_{pd}, K_{id}, K_{pq} and K_{iq} used, at 15 Hz (450 rpm)

(a) $K_{ps} = 1.0$, $K_{is} = 0.25$
(b) $K_{ps} = 1.3$, $K_{is} = 0.8$
(c) $K_{ps} = 1.5$, $K_{is} = 1.2$

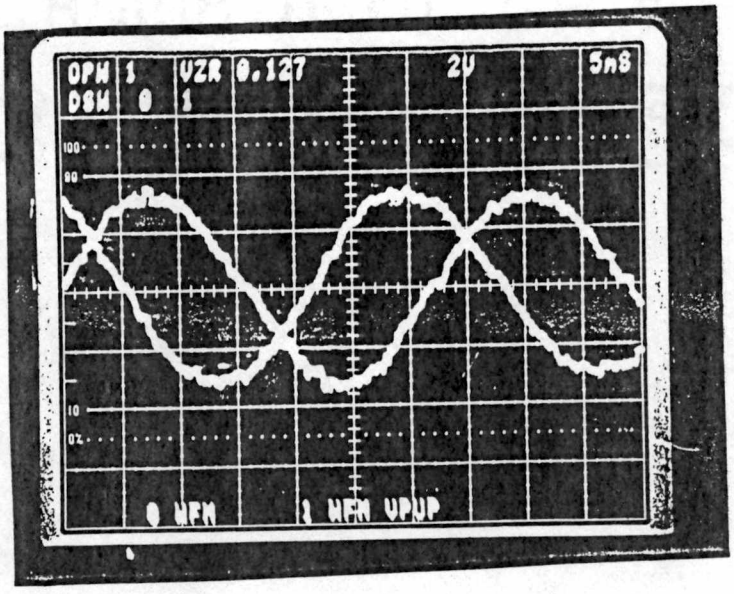
command of 450 rpm (47 r/s) is applied in the form of square wave pulses, of 0.25 Hz, as mentioned elsewhere. Fig. 7.15 (a) shows the speed controller response with the K_{ps} = 1.0 and K_{is} = 0.25. The response has the rise time of 300 ms, and is slightly overdamped. This is improved by increasing the K_{ps} to 1.3 and K_{is} to 0.8, as shown in Fig. 7.15 (b), where the response is closer to critically damped, with the rise time = 200 ms. The effect of the increased gains to K_{ps} = 1.5 and K_{is} = 1.2 brings the response to critically damped, with near-zero overshoot and the rise time of less than 100 ms. The values of the K_{ps} and K_{is} are matched closely to the ones obtained with the expert system based automatic tuning considering the fact that the speed command in the experimental system is 4.7 times the speed command used in the expert system based tuning. This brings the gains K_{ps} and K_{is} down correspondingly. Fig. 7.16 shows the steady-state performance of the indirect vector controlled induction motor drive system. Fig. 7.16(a) shows the electrical angle θ_e and one of the unit vectors, sin θ_e , at the drive operation at 450 rpm (30 Hz electrical). Fig. 7.16(b) shows the line currents i_a and i_b, also at the same frequency, on no load.

7.5 Summary

The experimental validation of controller design and tuning is described in this chapter. The details on the hardware development are discussed at sufficient length. The software organization and development is also described with particular emphasis on the implementation of various modes of tuning and methodology employed to obtain a one to one correspondence between the results obtained here and those with the expert system based tuning. The tuning process is illustrated by means of the results shown at the end of the chapter, in all the three modes, namely, tuning of i_{ds} current loop, tuning of i_{qs} current loop and the tuning of speed loop. The optimum gain values obtained through experimental results match closely with those obtained with the expert system based tuning. The differences in their values could be attributed to non-ideal sensors with noise and truncation and round-off errors introduced in the operation of the fixed-point DSP computation.



(a)



(b)

Fig. 7.16 Steady-State Performance of the Indirect Vector Controlled Induction Motor Drive System at 15 Hz, showing

(a) Angle θ_e, and sin θ_e, (0.5 pu / div)

(b) Line Currents i_b and i_c waves (5A / div)

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

8.1 Accomplishments of the Work Presented

The purpose of this project was to develop a comprehensive, expert system based, design program for AC drive systems, which embedded the know-how of expert design engineer. As far as possible, a close correlation was maintained between the expert system based development and the practices actually followed by the expert AC drive designer, to enhance the acceptability, usefulness and applicability of this software to the power electronics and AC drive design community. The methodology followed during the expert system development was as follows:

- The design problem was divided into smaller, manageable tasks, which can be tackled one at a time
- While solving each of the subtasks, the beginning phase was devoted to the validation of the new software concept being developed
- Next, the design knowledge related to the subtask at hand was collected, from various sources, such as the application notes, data books, literature and simulation studies. This can be likened to the knowledge acquisition phase of the expert system.
- The actual expert system implementation began with the knowledge representation, in which not only the expert system but all the allied models for simulation were developed. The user interface design was considered integral to the knowledge representation, and hence was performed side by side.
- Once the total system was in place, the expert system was put to test with a combination of input conditions representing a variety of user data, and the results were compared against their desired values. Any departures from these were corrected

by fine-tuning the expert system design.

- The development of the expert system based software was followed by the experimental validation to further verify and confirm the design done by the expert system

In the expert system based automated AC drive configuration selection and AC motor design, the following accomplishments were made:

- The design procedure for a list of applications was embedded in the expert system. The expert system based AC drive configuration selection covers a broad spectrum of AC drive applications, such as the pumps, fans, compressors, machine tool drives, robotics drives, elevator drives, electric vehicle drives, etc.
- The expert system offers the user also the choice to design a customized application, by providing all of the critical application specifications.
- The expert system was so designed as to offer the user the power to impose his choice on the type of converter, type of machine and type of control. Intelligent defaults were also implemented, in case the user's application information was not sufficient to arrive at a definite AC drive configuration.
- The expert system was also designed to calculate the motor nameplate specifications, such as the motor steady-state and peak power rating, motor voltage rating, motor current rating, number of poles, base frequency, NEMA design class, the class of insulation and type of enclosure and ventilation. These specs could be utilized by the user to get a suitable AC motor off-the-shelf or custom-designed

In the expert system based design of power converter system, the following was achieved:

- The expert system knowledge base was designed to implement the complete process of power converter system design, in the form of rating calculations of the power components, their selection from databases, the design of filter capacitor ratings, initial value selection for the turn-off snubber components and the thermal design for the power circuit.
- The databases were developed for the power BJTs, IGBTs and diode rectifier modules, and interfaced with the expert system so as to allow it to pick up the power devices corresponding to their calculated ratings.
- The graphical data of the power semiconductors in the form of their saturation voltage

characteristics and the switching time relationship with the collector currents was embedded in the expert system in the form of the equations of best-fitting curves, which were attained using TableCurve software.

- The graphical user interface was also developed for the expert system to communicate with the user effectively.

The development of expert system for power converter system simulation and design optimization achieved the following:

- The design and optimization considerations were embedded in the expert system, for the example of turn-off snubbers for the transistors.
- The methodology of expert system interface with the system simulation program, SIMNON, was developed in the form of C-based interface routines.
- The expert system was now assigned the additional task of being the design manager, when it not only performed the initial design and implemented simplified optimization algorithm for the snubber components, but also directed the simulation program to perform the simulation of the power converter system in the complete system or half-bridge mode.
- The graphical interface was also implemented so that the user could see the waveforms resulting from the expert system based simulation, for the quantities of his interest.
- Although the converter-machine model was embedded in the simulation program, with all the equivalent circuit parameters for the machine known to the simulation program beforehand, they could be easily extracted from the motor nameplate specifications, using, for example, the genetic algorithms.
- The snubber optimization process with the help of the expert system was demonstrated with the help of results of the snubber optimization.

The expert system for the automated design of controllers and their auto-tuning accomplished the following:

- Indirect vector controlled induction motor drive system with synchronous current control was chosen as the example for control design and initialization
- Methodology for initial value selection as well as tuning of the control gains for the current and speed loops was developed

- Simulation models were developed for the plant, i.e., the converter - machine combination, in SIMNON, and for the controllers in the simulator software for DSP 320C25. This type of simulation is uniquely offered by this software, and is not available with any other softwares simulation or design softwares.
- Software framework, for the expert system's interface with the DSP simulator and SIMNON's interface with the simulator was developed, using the C-based interface routines. The simulation software developed was developed in such a flexible manner that it can be autonomously used, even when decoupled from the expert system, if the user chose to supercede the expert system's design and auto-tuning algorithm.
- Expert system was developed for the initial design and auto-tuning of the controllers, with the help of the hybrid simulation performed in SIMNON and DSP simulator, with the help of a DOS-based batch file. The intermediate results of controller responses were displayed, as well as interpreted by the expert system to produce the tuned design of controllers.

During the experimental validation for the controller tuning on DSP 320C25 based indirect vector induction motor drive with synchronous current control, following was carried out:

- Hardware integration was done to obtain the DSP 320C25 interface with the IGBT inverter, through the hardware PWM modulator, and the sensors for currents and speed were interfaced with the DSP.
- Real-time software, consisting of the optimized controller code generated by the expert system, the I/O processing routines and the host-to-DSP communication routine was integrated, debugged and tested.
- In the real-time software, various modes of tuning were also implemented, by generating reference square wave commands for the variable corresponding to the mode of tuning.
- The process followed by the expert system to tune the controllers was emulated during the laboratory implementation, so as to obtain a one-to-one correspondence between the results obtained through software and hardware. The results of the three modes of tuning (i_{ds} current loop, i_{qs} current loop and speed loop) were obtained by running the drive system and were found to match those obtained with the expert system based auto-tuning.

8.2 Recommendations for Future Work

This project demonstrates the use of the expert system towards the detailed design of the AC drive systems for a variety of applications. The example of a popular AC drive configuration, the indirect vector controlled induction motor drive with synchronous current control, is used to demonstrate the usefulness and superiority of the expert system approach to other traditional, procedural approaches. Although the work completed for the development of the expert system based program is quite substantial, there is some scope of advancement. The features that can be used to upgrade this software are described below:

- The available software can be expanded, using the same principles, to include many more, non-drive applications such as UPS systems, active power line conditioners, etc.
- The range of power ratings for which the expert system operates can be expanded to include AC drive applications up to multimegawatt levels
- Different optimization criteria can be implemented, such as, cost, efficiency, packaging, size, reliability, thermal design, etc.
- Many of the design steps involve imprecise information to be processed, for which the expert system can use only the probabilistic reasoning. However, externally, fuzzy reasoning can be implemented through C-based fuzzy controllers, which can, for example, fine-tune the controllers in a better way.
- The main bottleneck of the system developed here is that it is DOS-based. As mentioned earlier, DOS imposes severe restrictions on the amount of memory and the number of programs that can be operated at the same time. Future efforts to improve this program should also include its implementation in a multi-tasking operating system such as the OS/2 or Unix.
- For experienced users, a schematic capture utility can be added at the front end, which would add to the capability of the expert system based package, to also allow design of user-defined AC drive system topologies.
- A genetic algorithm based program can be linked with the expert system, so that it can extract the motor parameters from the nameplate ratings designed by the expert system.

In conclusion, it is hoped that the expert system based automated AC drive system design program

will serve as the first step towards the total computer-integrated-manufacturing of the AC drive systems, by serving as the link between the designer's desk and the production floor, to achieve the same level of design automation as enjoyed by the VLSI systems today.

REFERENCES

[1] K. S. Tam, "Applications of AI techniques to design of static power converters", IEEE Ind. Appl. Soc. Annual Meeting Conf. Rec., pp 387 - 392, 1987.

[2] B. K. Bose, "Expert system, fuzzy logic and neural network applications in power electronics and motion control", Proceedings of the IEEE, pp. 1303 - 1323, vol. 82, August 1994.

[3] M. H. Kuo and K. D. T. Ngo, "Modelling and simulation of PWM dc-dc converters", Proc. of the IEEE Power Electronics Specialists Conference, pp. 184 - 192, 1986.

[4] H. Austin Spang III, "The federated computer-aided control design system", Proc. IEEE, vol. 72, no. 12, pp. 1724 - 1731, Dec. 1984.

[5] J. H. Taylor and D. K. Frederick, "Expert system architecture for computer-aided control engineering", Proc. IEEE, vol. 72, no. 12, pp. 1795 - 1805, Dec. 1984

[6] S. Lorenzo, J. M. Ruiz, F. Aldana and M. Shaker, "A new modelling and simulation CAD package for power converter design (PECADS)", IEEE Trans. on Industrial Electronics, vol. 37, no. 5, pp. 387 - 397, Oct. 1990.

[7] R. Kelkar, R. A. Wunderlich and L. J. Hitchcock, "Software breadboards for power electronic circuits", IEEE Trans. on Power Electronics, vol. 6, no. 2., pp. 170 - 178, April 1991.

[8] P. A. Subramanyam, "Synapse: An expert system for VLSI design", IEEE Computer Magazine, vol 19, no. 7, pp. 78 - 89, July 1986.

[9] D. C. Brown and B. Chandrasekaran, "Knowledge and control for a mechanical design expert system", IEEE Computer Megazine, vol. 19, no. 7, pp. 92 - 100, July 1986.

[10] W. S. Fraught, "Applications of AI in engineering", IEEE Computer Megazine, vol. 19, no. 7, pp. 17 - 27, July 1986.

[11] A. Basu, A. K. Majumdar and S. Sinha, "An expert system approach to control system design and analysis", IEEE Trans. on Systems, Man and Cybernatics, vol. 18, no. 5, pp. 685 - 694, Sept./Oct. 1988.

[12] D. Chen and B. K. Bose, "Expert system aided automated selection of industrial AC drives", IEEE Ind. Appl. Soc. Ann. Meeting Conf. Rec., pp. 387 - 392, 1992.

[13] V. Rajagopalan and K. Debebe, "Diagnosis and monitoring of AC drives", IEEE Ind. Appl. Soc. Conf. Rec., pp. 370 - 377, 1992.

[14] J. Holtz and H. Groschwitz, "An expert system on transformer design for switched-mode power supplies", Proc. of IEEE Industrial Electronics Conference, pp. 1413 - 1418, 1994.

[15] R. G. Vedder, "PC based expert system shells, some desirable and less desirable characteristics", Expert Systems Megazine, vol. 6, no. 1, pp. 28 - 42, Feb. 1989.

[16] D. Chorafas, "Knowledge Engineering", Van Nostrand Reinhold, New York, 1990.

[17] PC Plus User's Reference Manual, Texas Instruments, Nov. 1988.

[18] dBase III+ User's Manual, Ashton-Tate, 1988.

[19] DrHalo III User's Reference, Media Cybernatics, MD, 1991.

[20] TableCurve 3.0 User's Manual, Jandel Scientific, CA, 1991.

[21] Toshiba Drives Application Notes, 1990.

[22] Adjustable Frequency Controllers Application Guide, Square D Company, 1988.

[23] T. M. Jahns, "Motion control with permanent magnet AC machines", Proc. IEEE, vol. 82, no. 8, pp. 1350 - 1360, Aug. 1994.

[24] T. F. Lowrey and D. W. Petro, "Application considerations for PWM inverter-fed low voltage induction motors", IEEE Trans. on Ind. Applications, vol. 30, no. 2., pp. 286 - 293, March / April 1994.

[25] D. A. Dey, "Guidelines in selecting variable-speed drives from a user's viewpoint", IEEE Trans. on Ind. Applications, vol. 24, no. 6, pp. 1101- 1106, Nov./ Dec. 1988.

[26] G. E. Baumgart, "Sizing adjustable frequency AC induction motor controllers for high torque applications", IEEE Ind. Appl. Soc. Ann. Meeting Conf. Rec., pp 1769 - 1777, 1986.

[27] K. Yamaji, et. al., "Applications of induction motors and stepping motors in small-scale position control systems", Proc. of IEEE Industrial Electronics Conference, pp. 1002 - 1006, 1990.

[28] A. Fratta, A. Vagati and F. Villata, "On evolution of AC machines for spindle drive applications", IEEE Trans. on Ind. Applications, vol. 28, no. 5, pp. 1081 - 1086, Sept. / Oct. 1992.

[29] H. Tanaka, Y. Nagatani, M. Ehara, "Driving system incorporating vector control inverter for large-scale paper machine", IEEE Trans. on Ind. Applications, vol. 19, no. 3, pp. 450 - 460, May / June 1983.

[30] W. J. Jackson, "Mill motors for adjustable-speed AC drives", IEEE Trans. on Ind. Applications, vol. 29, no. 3, pp. 566 - 572, May / June 1993.

[31] J. L. Oldenkamp and S. C. Peak, "Selection and design of an inverter-driven induction motor for a traction drive system", IEEE Trans. on Ind. Applications, vol. 21, no. 1, pp.

259 - 265, Jan. / Feb. 1985.

[32] V. K. Modisetti and M. A. Ramulu, "Trends in electronic control of mine hoists", IEEE Trans. on Ind. Applications, vol. 22, no. 6, pp. 1105 - 1112, Nov. / Dec. 1986.

[33] S. M. Chhaya and B. K. Bose, "Expert system aided automated design technique for PWM voltage fed inverter for induction motor drive", IEEE Ind. App. Soc. Conf. Rec., pp. 770 - 778, 1992.

[34] W. McMurray, "Selection of snubbers and clamps to optimize the design of transistor switching converters", IEEE Trans. on Ind. Applications, vol. 16, no. 4, pp. 513 - 523, July / Aug. 1980.

[35] N. Mohan, T. Undeland and W. Robbins, "Power Electronics - Converters, Applications and Design", John Wiley and Sons, NY, 1994.

[36] T. Undeland et. al, "IGBT turn-off losses for hard-switching and with capacitive snubbers", Proc. of IEEE Power Electronics Specialists Conference, pp. 1501 - 1507, 1991.

[37] Heat dissipation components, EG&G Wakefield Engineering, 1990

[38] Power Semiconductor Databook, vol. 1-3, POWEREX Inc., 1988.

[39] J. W. A. Wilson and J. A. Yeamans, "Intrinsic harmonics of idelized inveter PWM systems", IEEE Ind. Appl. Soc. Ann. Meeting Conf. Rec., pp. 967 - 973, 1976.

[40] K. S. Rajasekara et al, "DC link filter design considerations in three-phase voltage source inverter fed induction motor drive system", IEEE Trans. on Ind. Applications, vol. 23, no. 4, pp. 673 - 680, July / Aug. 1987.

[41] S. M. Chhaya and B. K. Bose, "Expert system based automated simulation and design optimization of a voltage-fed inverter for induction motor drive", Proc. of IEEE Industrial Electronics Conference, pp. 1065 - 1070, 1993.

[42] H. Elmquist et al, "PC-SIMNON - User's Guide for MS-DOS Computers, Version 3.01", January 1990.

[43] Quick C Version 2.5 User's Reference, Microsoft Press, 1990.

[44] B. W. Williams, "Power Electronics, Devices, Drivers and Applications", John Wiley & Sons, New York, 1987.

[45] B. K. Bose, "Power Electronics and AC Drives", John Wiley & Sons, NY, 1988.

[46] S. M. Chhaya and B. K. Bose, "Expert system based automated design, simulation and controller tuning of AC drive system", Accepted for publication in IEEE

Industrial Electronics Conference, Orlando FL, 1995.

[47] B. Porter et al, "Real-time expert tuners for PI controllers", IEE Proceedings, vol. 134, pt. D, no. 4, pp. 260 - 263, July 1987.

[48] T. Kudor et al, "Self commissioning of vector controlled induction motors", IEEE Ind. Appl. Soc. Ann. Meeting Conf. Rec., pp. 528 - 535, 1993.

[49] L. J. Garces, "Parameter adaption for the speed-controlled static AC drive with a squirrel cage induction motor", IEEE Trans. on Ind. Applications, vol. 16, no. 2, pp. 173 - 178, Mar. / Apr. 1980.

[50] A. Nabae et al, "An approach to flux control of induction motors operated with variable frequency power supply", IEEE Trans. on Ind. Applications, vol. 16, no. 3, pp. 342 - 350, May / June 1980.

[51] Hakata et al, "Application strategies for AC rolling mill drives", IEEE Ind. Appl. Soc. Ann. Meeting, Conf. Rec., pp. 1141 - 1147, 1991.

[52] H. Sumner and G. M. Asher, "Self-commissioning of voltage-referenced voltage fed vector controlled induction motor drives", Conf. Rec. of IEEE Power Electronics Specialists' Conference, pp. 139 - 144, 1992.

[53] B. K. Bose, ed. , "Microcomputer Control of Power Electronics and Drives", IEEE Press, 1987.

[54] DSP-16 User's Manual, Ariel Corporation, 1989.

[55] TMS320C2x User's Reference, Texas Instruments, 1991.

[56] TMS320 Fixed Point DSP Optimizing C Compiler User's Guide, Texas Instruments, 1991.

[57] TMS320C2x C Source Debugger user's guide, Texas Instruments, 1991.

[58] TMS320C2x Assembly Language Development Tools, Texas Instruments, 1991.

VITAE

Sunil Madhusudan Chhaya was born in Rajkot, India on May 14, 1966. He completed his high school from Saurashtra High School, Rajkot in 1983. He joined the Lukhdhirji Engineering College, Morbi, Gujarat and obtained the Bachelor of Engineering degree in Power Electronics Engineering in December 1987. He was awarded the University Alumni Gold Medal for securing the highest grades in the University.

From March to July 1988, he worked at Comrel Electronics, Rajkot, India, as an assistant engineer designing and testing custom-built power supplies, battery chargers and automatic AC voltage regulators for commercial and defense applications. In July 1988, he accepted admission from the Indian Institute of Technology at Kharagpur, India, and the Government of India scholarship for a Masters degree in Electrical Engineering. He completed the program with honors in January, 1990. From March to July 1990, he worked at Kirloskar Electric Company, Mysore, India, as the senior engineer in inverter design and testing for AC motor drives.

In August 1990, he joined as an adjunct faculty at Lukhdhirji Engineering College, Morbi, and taught Power Electronics courses for junior and senior year undergraduate students until November 1990.

In January 1991, he accepted admission with research assistantship to the Ph. D. program at the University of Tennessee, Knoxville and worked under Dr. B. K. Bose in the area of expert system based AC motor drive design. He completed his Ph. D. studies in May, 1995. He is a member of IEEE and IEEE Industry Applications Society and has authored three IEEE conference papers. His areas of interest are, artificial intelligence applications to AC motor drives and their microprocessor and DSP based control.

APPENDICES

APPENDIX A: Listing on Expert System for AC Drive Configuration Selection and AC Motor Design

APPENDIX A.1 ARL Listing of Expert System Root Frame

```
DOMAIN :: "AC DRIVE SYSTEM DESIGN EXPERT"
ROOT FRAME :: AC-DRIVE-SYSTEM-DESIGN

======================
  Global KB data
======================

FRAME STRUCTURE ::
   AC-DRIVE-SYSTEM-DESIGN
      CONVERTER-DESIGN
         SIMULATE
         OPTIMIZE
      CONTROL-DESIGN
         IDS-TUNING
         IQS-TUNING
         SPD-TUNING
         TEST-RUN
      AC-MOTOR-DESIGN
      AC-DRIVE-CONFIGURATOR

KB Files :: (AC-DRIVE-CONFIGURATOR "ACDESEXP.k11" AC-MOTOR-DESIGN "
          ACDESEXP.k10" IDS-TUNING "ACDESEXP.k1" ACD-SYS-DES "ACDESEXP.k1"
          OBSERVE "ACDESEXP.k2" OPTIMIZE "ACDESEXP.k3" SIMULATE "
          ACDESEXP.k4" CONVERTER-DESIGN "ACDESEXP.k5" CONTROL-DESIGN "
          ACDESEXP.k6" TEST-RUN "ACDESEXP.k7" SPD-TUNING "ACDESEXP.k8"
          IQS-TUNING "ACDESEXP.k9" )
Parameter groups :: (AC-DRIVE-CONFIGURATOR-PARMS AC-MOTOR-DESIGN-PARMS
                TEST-RUN-PARMS SPD-TUNING-PARMS IQS-TUNING-PARMS
                IDS-TUNING-PARMS CONTROL-DESIGN-PARMS
                CONVERTER-DESIGN-PARMS OPTIMIZE-PARMS SIMULATE-PARMS
                AC-DRIVE-SYSTEM-DESIGN-PARMS )
Rule groups :: (AC-DRIVE-CONFIGURATOR-RULES AC-MOTOR-DESIGN-RULES
            TEST-RUN-RULES SPD-TUNING-RULES IQS-TUNING-RULES
            IDS-TUNING-RULES CONTROL-DESIGN-RULES
            CONVERTER-DESIGN-RULES OPTIMIZE-RULES SIMULATE-RULES
            AC-DRIVE-SYSTEM-DESIGN-RULES META-RULES )
Number of rules :: 413
Number of meta-rules :: 0
Variables :: (DOMAIN POWER-BJT-TYPE)
TEXTAGS :: ()
Functions :: (CSA-CALC FILT_CAP_RMS TRR-IC TRV_CALC VD-ID VD-ID-FD)

  ===============
    VARIABLES
  ===============

DOMAIN
  VALUE :: "AC DRIVE SYSTEM DESIGN EXPERT"
POWER-BJT-TYPE
  ===============
    FUNCTIONS
  ===============
CSA-CALC
  TRANSLATION :: (calculates the thermal capacitance of the heat sink)
  TEMPLATE :: (PARM PARM PARM)
```

```
 TYPE :: EXPRESSION
 SOURCE :: (LAMBDA
         (DELTA-PJ DELTA-TJ HS-RSA)
         (/ 60
          (*
           (VAL1 FRAME HS-RSA)
           (LOG
            (/
             (*
              (VAL1 FRAME DELTA-PJ)
              (VAL1 FRAME HS-RSA))
             (-
              (*
               (VAL1 FRAME DELTA-PJ)
               (VAL1 FRAME HS-RSA))
              (VAL1 FRAME DELTA-TJ)))))))

FILT_CAP_RMS
 TRANSLATION :: (This user-defined function returns the value of the rms
             capacitor filter current. The function call looks like
             this: filt_cap_irms = (filt_cap_rms is_rms id_av
             il_fund_rms ) )
 TEMPLATE :: (PARM PARM PARM)
 TYPE :: EXPRESSION
 SOURCE :: (LAMBDA
         (IS-RMS ID-AV IL-FUND-RMS)
         (SQRT
          (+
           (-
            (EXPT
             (VAL1 FRAME IS-RMS) 2)
            (EXPT
             (VAL1 FRAME ID-AV) 2))
           (* 0.1225
            (EXPT
             (VAL1 FRAME IL-FUND-RMS) 2)))))

TRR-IC
 TRANSLATION :: (This function calculates the reverse-recovery time for
             the feedback diode from its Id value The function call
             looks like, fd-trr = (trr-ic power_bjt_idc) )
 TEMPLATE :: (PARM)
 TYPE :: EXPRESSION
 SOURCE :: (LAMBDA
         (POWER_BJT_IDC)
         (+
          (+
           (+ 0.42733515
            (* 3.8695039e-4
             (EXPT
              (VAL1 FRAME POWER_BJT_IDC) 1.5)))
           (/ -0.41234325
            (LOG
             (VAL1 FRAME POWER_BJT_IDC))))
          (/ 1.5327672
           (*
            (VAL1 FRAME POWER_BJT_IDC)
            (VAL1 FRAME POWER_BJT_IDC)))))

TRV_CALC
 TRANSLATION :: (This user-defined function returns the voltage rise time
             for the BJT during snubber-assisted turn-off )
```

```
 TEMPLATE :: (PARM PARM PARM PARM)
 TYPE :: EXPRESSION
 SOURCE :: (LAMBDA
          (SNUB-C T_FALL INPUT-VOLT POWER_BJT_IDC)
          (SQRT
           (/
            (* 2
             (*
              (VAL1 FRAME SNUB-C)
              (*
               (VAL1 FRAME T_FALL)
               (*
                (SQRT 2)
                (VAL1 FRAME INPUT-VOLT)))))
            (VAL1 FRAME POWER_BJT_IDC))))

VD-ID
 TRANSLATION :: (This function calculates the forward conduction drop for
              the rectifier diodes. The function call looks like,
              vd-on-r = (vd-id rect-d-idav) )
 TEMPLATE :: (PARM)
 TYPE :: EXPRESSION
 SOURCE :: (LAMBDA
          (RECT_D_IDAV)
          (+
           (+
            (+ 0.97651209
             (* 6.1768023000004e-9
              (EXPT
               (VAL1 FRAME RECT_D_IDAV) 3)))
            (* 0.011757748
             (*
              (SQRT
               (VAL1 FRAME RECT_D_IDAV))
              (LOG
               (VAL1 FRAME RECT_D_IDAV)))))
           (/ -0.18027971
            (SQRT
             (VAL1 FRAME RECT_D_IDAV)))))
VD-ID-FD
 TRANSLATION :: (This LISP function calculates the forward conduction drop
              of the feedback diode of BJT from its average current
              value the function call looks like: vd-on-fd = (vd-id-fd
              power_bjt_idc ) )
 TEMPLATE :: (PARM)
 TYPE :: EXPRESSION
 SOURCE :: (LAMBDA
          (POWER_BJT_IDC)
          (+
           (+
            (+ 0.76338445
             (* -3.6505402e-4
              (EXPT
               (VAL1 FRAME POWER_BJT_IDC) 1.5)))
            (* 0.026194347
             (SQRT
              (VAL1 FRAME POWER_BJT_IDC))
             (LOG
              (VAL1 FRAME POWER_BJT_IDC))))
           (/ -0.086179014
            (VAL1 FRAME POWER_BJT_IDC))))
```

```
Frame :: AC-DRIVE-SYSTEM-DESIGN

IDENTIFIER :: AC-DRIVE-SYSTEM-DESIGN-
TRANSLATION :: (Expert system for PWM inverter design)
GOALS :: (CONFIG-DONE MOTOR-DESIGN-DONE CONV-DES-DONE CONT-DES-DONE)
INITIALDATA :: (NAME APPLICATION)
PROMPTEVER :: ("This expert system designs an AC drive system. Based on
        user's application" :LINE "performance specs, the expert
        system first configures the AC drive system in" :LINE "
                terms of the type of AC machine, its power converter topology and the type"
            :LINE "of control." :LINE 2 "Now, the expert system
            computes the size of the AC machine by determining" :LINE "
            its nameplate specs. This enables the user to choose a
            suitable machine" :LINE "off-the-shelf or get it
            custom-designed." :LINE 2 "Then, it designs the power
            converter system, by calculating the ratings and" :LINE "

    selecting proper power devices from the database. It also computes the heat"
            :LINE "sink size based on the power dissipation in the
            converter. Then, it finds" :LINE "the initial values of the
            snubber components. Now, the expert system" :LINE "exports
            the design parameters to the system simulation program
            PC-SIMNON," :LINE "and then performs the steady-state
            simulation of the power converter system." :LINE "The
            expert system also optimizes the power converter system,
            examplified" :LINE "here by optimization of snubber
            circuit, simulating iteratively its turn-on" :LINE "and
            turn-off behavior with varying snubber parameters." :LINE
            2 "Having completed the design-simulation-optimization of
            the power converter" :LINE "circuit, the expert system
            designs the control system, in the form of P-I" :LINE "

    gains, based on the machine and load parameters. It then supplies the" :LINE "
            designed parameters to the controllers implemented in
            TMS320C25 DSP assembly" :LINE "language or C language
            (compiled to DSP assembly language). The complete" :LINE "
            closed loop drive system, with the converter-machine model
            in PC-SIMNON and" :LINE "the controllers in the DSP 320C25
            Assembly Language, is simulated in" :LINE "automated
            fashion, by the expert system. The results of the
            simulation of" :LINE "controllers are again used for their
            optimization, with the inner (current)" :LINE "loops tuned
            first and then the outer (slower) loops are tuned." :LINE
            2 "The expert system displays the results of the design -
            optimization at the" :LINE "end." )
  DISPLAYRESULTS :: YES
  PARMGROUP :: AC-DRIVE-SYSTEM-DESIGN-PARMS
  RULEGROUPS :: (AC-DRIVE-SYSTEM-DESIGN-RULES)
  OFFSPRING :: (CONVERTER-DESIGN CONTROL-DESIGN AC-MOTOR-DESIGN
        AC-DRIVE-CONFIGURATOR )
  AC-DRIVE-SYSTEM-DESIGN-PARMS :: (ACCN-DCCN-TIME ACCN-TORQUE APPL-POWER
                    APPL-TYPE APPLICATION BASE-SPEED
                    BASE-SPEED-RANGE BREAKAWAY-TORQUE
                    BREAKAWAY-TORQUE-VALUE
                    CHOICE-FOR-CONTROL CHOICE-FOR-CONVERTER
                    CHOICE-FOR-MACHINE CHOICE-OF-SWITCH
                    CHOICE-OF-SYSTEM CONFIG-DONE
                    CONT-DES-DONE CONT-OPT-DONE
                    CONTROL-DESIGN? CONTROL-TYPE
                    CONTROL-VARIABLE CONV-DES-DONE
                    CONV-OPT-DONE CONV-TYPE
                    CONVERTER-DESIGN? CTRL-TYPE DESOPT-DONE
                    DRIVE-CONTROL-CHAR DS-DES-DONE
                    ENVIRONMENTAL-CONDITION
```

HIGH-INPUT-PF-REQD INFO-ON-POWER
INV-CONT-POWER-RATING IOSF IOSF_ACC
IOSF_BA LOAD-TYPE LOW-ACOUSTIC-NOISE
MACHINE-TYPE MAX-SPEED MIN-SPEED MOHF
MOSF MOSF_ACC MOTOR-BASE-FREQ
MOTOR-DESIGN-DONE MOTOR-DESIGN?
MOTOR-ENCLOSURE-VENTILLATION
MOTOR-MAX-POWER-RATING
MOTOR-NO-OF-POLES MOTOR-OUTPUT-POWER
MOTOR-RATED-CURRENT MOTOR-SS-POWER
MOTOR-VOLTAGE-RATING MSF NAME
NEMA-MOTOR-DESIGN NO-OF-PHASES
SPEED-RANGE SPEED-REVERSIBILITY
SS-TORQUE SUPPLY-INPUT-VOLT
TEST-RUN-DONE TORQUE-AT-MIN-SPEED
TOTAL-INERTIA TYPE-OF-CONTROL
TYPE-OF-CONVERTER TYPE-OF-MACHINE
TYPE-OF-PMAC-MACHINE TYPE-OF-PUMP
TYPE-OF-SWITCH ZERO-SPD-OPER)

AC-DRIVE-SYSTEM-DESIGN-RULES :: (RULE178 RULE179 RULE271 RULE350 RULE351
RULE353 RULE357 RULE359 RULE367 RULE375
RULE376 RULE383 RULE386 RULE396 RULE399
RULE400)

AC-DRIVE-SYSTEM-DESIGN-PARMS

ACCN-DCCN-TIME

TRANSLATION :: (time required for 0 to max speed acceleration)
PROMPT :: (:ATTR (CYAN HIGH) "What is the minimum" :ATTR (YELLOW HIGH) "
acceleration / deceleration time" :ATTR (CYAN HIGH) "(secs)
needed by the" :LINE "load for min speed <--> maximum speed
transition?" :LINE 2 "(This information determines the peak
power rating of the motor)")
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
ANTECEDENT-IN :: (RULE395 RULE314)
RANGE :: (0 1000)
CONTAINED-IN :: (RULE395 RULE314)

ACCN-TORQUE

TRANSLATION :: (acceleration / deceleration torque)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE395 RULE314)
USED-BY :: (RULE315 RULE316 RULE402 RULE410 RULE401)
RANGE :: (0 1000)
CONTAINED-IN :: (RULE316 RULE402 RULE401)

APPL-POWER

TRANSLATION :: (power required by the application)
PROMPT :: ("What is the power in kW required by your application in
steady state ?")
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE342)
ANTECEDENT-IN :: (RULE412)
USED-BY :: (RULE330)
CONTAINED-IN :: (RULE330)
RANGE :: (1 300)

APPL-TYPE

```
=========
 TRANSLATION :: (type of application)
 TYPE :: SINGLEVALUED
 EXPECT :: (HIGH-PERFORMANCE GENERAL-PURPOSE)
 UPDATED-BY :: (RULE328 RULE329 RULE397 RULE398)
 ANTECEDENT-IN :: (RULE271 RULE347)
 USED-BY :: (RULE299 RULE341 RULE335 RULE379 RULE402 RULE410 RULE401
        RULE403 RULE356 RULE413 SREFMARK RULE382 RULE381 )

APPLICATION
==========
 TRANSLATION :: (User's application)
 PROMPT :: ("What is the type of your application?" :LINE "Choose "USER
        DEFINED" if it does not match any of the choices." :LINE "(*
        indicates the demonstration example)" )
 TYPE :: SINGLEVALUED
 EXPECT :: (PUMP FAN COMPRESSOR CONVEYOR CRUSHER WINDER SPINDLE
        TENSION_CONTROL ELEVATOR *SPEED_CONTROL_SERVO ROBOTIC_ARM
        TRACTION ELECTRIC_VEHICLE USER_DEFINED )
 ANTECEDENT-IN :: (RULE338 RULE349 RULE359 RULE357 RULE358 RULE367
             RULE384 RULE392 RULE393 RULE371 RULE395 RULE396
             RULE399 RULE327 RULE314 RULE400 RULE383 RULE375
             RULE376 RULE409 )
 USED-BY :: (RULE331 RULE332 RULE294 RULE295 RULE350 RULE351 RULE335
        RULE394 RULE328 RULE329 RULE397 RULE398 RULE379 RULE380
        SREFMARK RULE365 RULE364 )

BASE-SPEED
==========
 TRANSLATION :: (base speed of the machine)
 PROMPT :: ("What is the maximum speed (base speed) in rpm at which the
        application" :LINE "requires max. torque ?" )
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 ANTECEDENT-IN :: (RULE355 RULE358 RULE308 RULE395 RULE409)
 USED-BY :: (RULE305 RULE306 RULE343 RULE402 RULE410 RULE401)
 RANGE :: (0 100000)
 CONTAINED-IN :: (RULE305 RULE306 RULE355 RULE358 RULE343 RULE308
             RULE395 RULE380 RULE402 RULE409 RULE410 RULE401 )

BASE-SPEED-RANGE
================
 TRANSLATION :: (ratio of base speed to min speed, used for motor
             overheating calc )
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-BY :: (RULE305)
 USED-BY :: (RULE310 RULE311 RULE312)
 RANGE :: (1 500)
 METHOD :: (FQUOTIENT (VAL1 FRAME BASE-SPEED) (VAL1 FRAME MIN-SPEED))

BREAKAWAY-TORQUE
================
 TRANSLATION :: (the breakaway torque required for the motor)
 TYPE :: SINGLEVALUED
 EXPECT :: (LOW MODERATE HIGH)
 UPDATED-BY :: (RULE331 RULE332 RULE294 RULE295 RULE350)
 USED-BY :: (RULE333)

BREAKAWAY-TORQUE-VALUE
======================
 TRANSLATION :: (value of breakaway torque needed by the application)
 PROMPT :: ("What is the value of the worst-case breakaway torque
        required by your" :LINE "application, as a % of full-load
        torque ?" :LINE 2 "(This information is necessary to
```

calculate inverter oversizing)")
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (RULE333)
CONTAINED-IN :: (RULE333)
RANGE :: (20 250)

CHOICE-FOR-CONTROL
==========
TRANSLATION :: (user's choice for drive control strategy)
PROMPT :: ("Do you have a choice over the control strategy of the drive
system ?" :LINE 2 "(Select the one that applies)")
TYPE :: SINGLEVALUED
EXPECT :: (OPEN-LOOP-V/HZ *VECTOR-CONTROL DON'T-CARE)

CHOICE-FOR-CONVERTER
==========
TRANSLATION :: (the user's choice for power converter topology)
PROMPT :: ("Do you have a preference for a particular power converter
topology ?" :LINE 2 "(Select the one that applies)")
TYPE :: SINGLEVALUED
EXPECT :: (*RECTIFIER-INVERTER RECTIFIER-BOOST_CHOPPER-INVERTER
DUAL-PWM-INVERTER DON'T-CARE)
UPDATED-IN :: (RULE399 RULE383 RULE376)
ANTECEDENT-IN :: (RULE344)
USED-BY :: (RULE298 RULE297 RULE335)

CHOICE-FOR-MACHINE
==========
TRANSLATION :: (user's choice for ac machine)
PROMPT :: ("Do you have a preference over the type of AC machine ?"
:LINE 2 "(select the choice applicable)")
TYPE :: SINGLEVALUED
EXPECT :: (*INDUCTION-MOTOR PERMANENT-MAGNET-AC-MOTOR DON'T-CARE)
ANTECEDENT-IN :: (RULE373 RULE352 RULE391)
USED-BY :: (RULE300 RULE296 RULE374 RULE301)

CHOICE-OF-SWITCH
==========
TRANSLATION :: (user's preference for power switch in inverter)
PROMPT :: ("Do you prefer to have one of the following power
semiconductor switches in" :LINE "your inverter ?")
TYPE :: SINGLEVALUED
EXPECT :: (BJT *IGBT DON'T-CARE)
USED-BY :: (RULE302 RULE303)

CHOICE-OF-SYSTEM
==========
TRANSLATION :: (Choice of design / optimization of system)
PROMPT :: ("Which of the following system do you want to design or
optimize?")
TYPE :: SINGLEVALUED
EXPECT :: (POWER-CONVERTER-DESIGN CONTROL-DESIGN)
HELP :: ("The expert system can perform the drive system" :LINE "design,
can design and optimize the power" :LINE "converter and can
also design and optimize the" :LINE "controllers.")

CONFIG-DONE
==========
TRANSLATION :: (ac drive configuration complete)
TYPE :: YES/NO
UPDATED-BY :: (RULE334)
ANTECEDENT-IN :: (RULE348 RULE353)

USED-BY :: (RULE178)

CONT-DES-DONE
==========
TRANSLATION :: (flag set when control design is done)
TYPE :: YES/NO
UPDATED-BY :: (RULE177)
USED-BY :: (RULE178)

CONT-OPT-DONE
==========
TRANSLATION :: (flag set when controller optimization is done)
TYPE :: YES/NO
UPDATED-BY :: (RULE292)
USED-BY :: (RULE178)

CONTROL-DESIGN?
==========
TRANSLATION :: (user's choice for design/optimization of ac drive
control system)
PROMPT :: ("The power converter system design is now complete. Do you
wish to proceed" :LINE "with the design of the control?")
TYPE :: YES/NO
ANTECEDENT-IN :: (RULE271)
USED-BY :: (RULE178)

CONTROL-TYPE
==========
TRANSLATION :: (Type of control strategy chosen for the ac drive system
)
TYPE :: YES/NO
USED-BY :: (RULE179)

CONTROL-VARIABLE
==========
TRANSLATION :: (the control variable for the application)
PROMPT :: ("What is the control variable for your application ?" :LINE "
(Select the one that applies)")
TYPE :: SINGLEVALUED
EXPECT :: (SPEED TORQUE POSITION)

CONV-DES-DONE
==========
TRANSLATION :: (Flag set when design of converter is done)
TYPE :: YES/NO
UPDATED-BY :: (RULE386 RULE388)
ANTECEDENT-IN :: (RULE271)
USED-BY :: (RULE178)

CONV-OPT-DONE
==========
TRANSLATION :: (Flag set when converter optimization done)
TYPE :: YES/NO
ANTECEDENT-IN :: (RULE271)
USED-BY :: (RULE178)

CONV-TYPE
==========
TRANSLATION :: (Type of power converter circuit topology)
TYPE :: SINGLEVALUED
USED-BY :: (RULE179)

CONVERTER-DESIGN?
==========
TRANSLATION :: (user's choice for designing converter system)

PROMPT :: ("The AC drive system configuration is selected, and the AC
motor is designed" :LINE "now. Do you wish to continue with
the design of power converter system?" :LINE 2 "(You could,
alternately, select an industrial drive product commercially
" :LINE "available from one of the vendors.)")
TYPE :: YES/NO
USED-BY :: (RULE178 RULE386)

CTRL-TYPE

TRANSLATION :: (type of control, dummy variable to store temporary
value of control)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE329)
ANTECEDENT-IN :: (RULE360 RULE371)
USED-BY :: (RULE372 SREFMARK RULE365 RULE364)

DESOPT-DONE

TRANSLATION :: (design and optimization of the drive system done)
TYPE :: YES/NO
UPDATED-BY :: (RULE178)

DRIVE-CONTROL-CHAR

TRANSLATION :: (Chars that determine ac drive system control of
user-defined appl)
PROMPT :: ("Which of the following characterize the AC drive for your
application ?" :LINE 2 "(Select ALL that apply." :LINE "This
information is used to determine the type of control, i.e.,
open-loop" :LINE "V/Hz or vector control for the AC drive for
your application, and a YES" :LINE "response on any of the
following will indicate the application needing VECTOR"
:LINE "CONTROL)")
TYPE :: ASK-ALL
EXPECT :: (FREQUENT-SPEED-REVERSALS ZERO-SPEED-OPERATION
PRECISE-SPEED-CONTROL PRECISE-TORQUE-CONTROL
PRECISE-POSITION-CONTROL FOUR-QUADRANT-OPERATION)
ANTECEDENT-IN :: (RULE399 RULE400)
USED-BY :: (RULE397 RULE398)

DS-DES-DONE

TRANSLATION :: (drive system design is done)
TYPE :: YES/NO
UPDATED-BY :: (RULE179)

ENVIRONMENTAL-CONDITION

TRANSLATION :: (the environmental condition of the motor installation)
PROMPT :: ("Does your application require the motor installation in
hazardous / explosive" :LINE "environment ?")
TYPE :: YES/NO
USED-BY :: (RULE309 RULE346)

HIGH-INPUT-PF-REQD

TRANSLATION :: (flag set to indicate user's need for high input pf)
PROMPT :: (Do you need to operate at near-unity input power factor ?)
TYPE :: YES/NO
UPDATED-IN :: (RULE344)
USED-BY :: (RULE298 RULE297)

INFO-ON-POWER

TRANSLATION :: (information on steady-state power requirement)
PROMPT :: ("Which of the following info can you provide on the load power requirement ?" :LINE "(You MUST know at least one of these)")
TYPE :: SINGLEVALUED
EXPECT :: (OUTPUT-POWER STEADY-STATE-TORQUE)
ANTECEDENT-IN :: (RULE339 RULE342 RULE308)
USED-BY :: (RULE330)

INV-CONT-POWER-RATING
TRANSLATION :: (continuous power rating of the inverter)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE403 RULE356 RULE413)

IOSF
TRANSLATION :: (inverter oversizing factor)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE324 RULE325)
DEFAULT :: (1)
RANGE :: (1 10)

IOSF_ACC
TRANSLATION :: (inverter oversizing factor due to acceleration)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE316)
USED-BY :: (RULE324 RULE325)
DEFAULT :: (1)
RANGE :: (1 10)
CONTAINED-IN :: (RULE325)

IOSF_BA
TRANSLATION :: (inv oversizing factor due to breakaway torque)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE333 RULE294)
USED-BY :: (RULE324 RULE325)
DEFAULT :: (1)
RANGE :: (1 10)
CONTAINED-IN :: (RULE324)

LOAD-TYPE
TRANSLATION :: (the type of load)
TYPE :: SINGLEVALUED
EXPECT :: (CONSTANT-TORQUE VARIABLE-TORQUE CONST-HP)
UPDATED-BY :: (RULE331 RULE332 RULE294 RULE295 RULE350)
UPDATED-IN :: (RULE375)
ANTECEDENT-IN :: (RULE347)
USED-BY :: (RULE341 RULE311)

LOW-ACOUSTIC-NOISE
TRANSLATION :: (user's requirement for low acoustic noise)
PROMPT :: (Does the application need to generate low acoustic noise?)
TYPE :: YES/NO
UPDATED-BY :: (RULE303)
USED-BY :: (RULE302)

MACHINE-TYPE
==========
TRANSLATION :: (Type of driven AC machine)
TYPE :: SINGLEVALUED
USED-BY :: (RULE179)

MAX-SPEED
========
TRANSLATION :: (maximum speed of the load)
PROMPT :: (What is the load maximum speed (rpm)?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE341 RULE380)
UPDATED-IN :: (RULE376 RULE409)
ANTECEDENT-IN :: (RULE304 RULE314)
RANGE :: (0 100000)
CONTAINED-IN :: (RULE314)

MIN-SPEED
========
TRANSLATION :: (minimum speed of the load)
PROMPT :: ("What is the minimum speed (rpm) at which the load operates ?"
)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE357 RULE358 RULE385 RULE400 RULE383 RULE375 RULE376)
ANTECEDENT-IN :: (RULE304 RULE395 RULE314)
USED-BY :: (RULE305)
RANGE :: (0 100000)
CONTAINED-IN :: (RULE305 RULE395 RULE314)

MOHF
====
TRANSLATION :: (motor oversizing factor due to overheating at low
speeds)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE310 RULE332 RULE294 RULE311 RULE312)
USED-BY :: (RULE340 RULE317 RULE318)
RANGE :: (1 10)
DEFAULT :: (1)

MOSF
====
TRANSLATION :: (motor oversizing factor)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE340 RULE317 RULE318)
USED-BY :: (RULE319)
CONTAINED-IN :: (RULE319)
RANGE :: (1 10)
DEFAULT :: (1)

MOSF_ACC
========
TRANSLATION :: (motor oversizing factor due to acceleration)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE315)
USED-BY :: (RULE340 RULE317 RULE318)
DEFAULT :: (1)
RANGE :: (1 10)

MOTOR-BASE-FREQ

TRANSLATION :: (base frequency of the motor)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE306)
USED-BY :: (SREFMARK RULE337)

MOTOR-DESIGN-DONE

TRANSLATION :: (design of motor nameplate specs complete)
TYPE :: YES/NO
UPDATED-BY :: (SREFMARK RULE337)
UPDATED-IN :: (RULE345)
ANTECEDENT-IN :: (RULE348 RULE353)
USED-BY :: (RULE178 SREFMARK RULE337)

MOTOR-DESIGN?

TRANSLATION :: (flag set when the user wants to design the motor)
PROMPT :: ("The configuration of the AC drive system is now over. The
results will be" :LINE "displayed after this message. Do you
wish to continue with the design of AC" :LINE "motor,
converter and control?" :LINE 2 "(If you want to select your
own ac drive system" :LINE "from among the industrial drives
commercially available, you can quit now.)")
TYPE :: YES/NO
ANTECEDENT-IN :: (RULE345 RULE348)

MOTOR-ENCLOSURE-VENTILLATION

TRANSLATION :: (enclosure and ventillation for the motor)
TYPE :: SINGLEVALUED
EXPECT :: (OSV TEFC TENV)
UPDATED-BY :: (RULE309 RULE346)
UPDATED-IN :: (RULE385)
USED-BY :: (SREFMARK RULE337)
DEFAULT :: (OSV)

MOTOR-MAX-POWER-RATING

TRANSLATION :: (Peak power rating of the AC motor)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE402 RULE410 RULE401)
ANTECEDENT-IN :: (RULE412)
USED-BY :: (RULE025 RULE001 RULE024 RULE034 RULE356 RULE413)
CONTAINED-IN :: (RULE025 RULE001 RULE024 RULE034 RULE356 RULE413)
METHOD :: (TIMES 1 (VAL1 FRAME MOTOR-OUTPUT-POWER))

MOTOR-NO-OF-POLES

TRANSLATION :: (number of stator poles for the AC motor)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (SREFMARK RULE337)
DEFAULT :: (4)
RANGE :: (4 10)

MOTOR-OUTPUT-POWER

TRANSLATION :: (output power rating of the motor)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE319)
ANTECEDENT-IN :: (RULE371)
USED-BY :: (RULE320 RULE321 RULE322 RULE323 RULE372 RULE403 RULE356

SREFMARK RULE337 RULE382 RULE381 RULE365 RULE364)
CONTAINED-IN :: (RULE326 RULE403)

MOTOR-RATED-CURRENT

TRANSLATION :: (rated motor current (Amps))
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE326)
METHOD :: (PLUS (FIX$ (FQUOTIENT (TIMES 680 (VAL1 FRAME
MOTOR-OUTPUT-POWER)) (VAL1 FRAME MOTOR-VOLTAGE-RATING)))
1)

MOTOR-SS-POWER

TRANSLATION :: (steady-state power of motor)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE330 RULE343)
UPDATED-IN :: (RULE308)
ANTECEDENT-IN :: (RULE355 RULE327)
USED-BY :: (RULE319 RULE326)
CONTAINED-IN :: (RULE355 RULE319)
METHOD :: (FQUOTIENT (TIMES (VAL1 FRAME SS-TORQUE) (VAL1 FRAME
BASE-SPEED)) 9550)

MOTOR-VOLTAGE-RATING

TRANSLATION :: (voltage rating of the motor)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE320 RULE321 RULE322 RULE323 RULE372 RULE394 SREFMARK
RULE382 RULE381 RULE365 RULE364)
USED-BY :: (RULE326 RULE001 RULE024 RULE034 SREFMARK RULE337 RULE382
RULE381)
CONTAINED-IN :: (RULE326 RULE001 RULE024 RULE034)

MSF

TRANSLATION :: (motor service factor)
TYPE :: SINGLEVALUED
USED-BY :: (SREFMARK RULE337)
DEFAULT :: (1.15)

NAME

TRANSLATION :: (Name of the client)
PROMPT :: (:ATTR (CYAN HIGH) "What is your" :ATTR (YELLOW HIGH) "name"
:ATTR (CYAN HIGH) "?")
TYPE :: SINGLEVALUED
EXPECT :: SINGLE-LINE-INPUT
HELP :: (:ATTR (RED HIGH) "Enter your name for" :ATTR (BLUE HIGH) "
identification" :ATTR (RED HIGH) "purpose")
CONTAINED-IN :: (RULE015 RULE042 RULE112 RULE412)

NEMA-MOTOR-DESIGN

TRANSLATION :: (NEMA class of motor design)
TYPE :: SINGLEVALUED
EXPECT :: (CLASS-A CLASS-B CLASS-C CLASS-D)
USED-BY :: (SREFMARK RULE337)
DEFAULT :: (CLASS-B)

NO-OF-PHASES

TRANSLATION :: (The number of phases)
PROMPT :: (:ATTR (CYAN HIGH) "What is the" :ATTR (YELLOW HIGH) "number

```
        of phases" :ATTR (CYAN HIGH) "of the supply?" )
  TYPE :: SINGLEVALUED
  EXPECT :: (SINGLE_PHASE THREE_PHASE)
  ANTECEDENT-IN :: (RULE015 RULE042 RULE081 RULE084 RULE412)
  USED-BY :: (RULE007 RULE016 RULE026 RULE028 RULE032 RULE033 RULE036
          RULE037 RULE038 RULE076 RULE082 RULE067 RULE075 )
  HELP :: (:ATTR (RED HIGH) "As a rule, Inverters" :ATTR (BLUE HIGH) "upto
        5 hp are fed from" :LINE 2 "the single phase supply," :ATTR (
        RED HIGH ) "but you can also have" :LINE 2 "inverters in this
        range supplied from 3-phase" :LINE 2 "supply. All" :ATTR (BLUE
        HIGH ) "higher power inverters are fed" :LINE 2 "from three
        phase supply only." :ATTR (RED HIGH) "So, decide" :LINE 2 "
        accordingly." )

SPEED-RANGE
===========
  TRANSLATION :: (range of speed of operation of load)
  TYPE :: SINGLEVALUED
  EXPECT :: POSITIVE-NUMBER
  RANGE :: (1 10000)

SPEED-REVERSIBILITY
===================
  TRANSLATION :: (load requirement for fast speed reversals)
  PROMPT :: ("Does the application need fast and frequent speed reversals
        ?" )
  TYPE :: YES/NO
  USED-BY :: (RULE335)

SS-TORQUE
=========
  TRANSLATION :: (steady-state torque of the load at base speed)
  PROMPT :: ("What is the steady-state torque (in N-m) for the load at
        base speed ?" )
  TYPE :: SINGLEVALUED
  EXPECT :: POSITIVE-NUMBER
  UPDATED-IN :: (RULE339 RULE355)
  ANTECEDENT-IN :: (RULE308)
  USED-BY :: (RULE343 RULE315 RULE316 RULE402 RULE410 RULE401)
  METHOD :: (FQUOTIENT (TIMES (VAL1 FRAME MOTOR-SS-POWER) 9550) (VAL1
        FRAME BASE-SPEED ) )
  RANGE :: (0 10000)
  CONTAINED-IN :: (RULE343 RULE308 RULE316 RULE402 RULE410)

SUPPLY-INPUT-VOLT
=================
  TRANSLATION :: (supply input voltage for the ac drive system)
  PROMPT :: ("What is the value of the supply input voltage for the AC
        drive system ?" )
  TYPE :: SINGLEVALUED
  EXPECT :: (220 460 DON'T-CARE)
  UPDATED-BY :: (SREFMARK RULE365 RULE364)
  UPDATED-IN :: (RULE383)
  ANTECEDENT-IN :: (RULE371 RULE327)
  USED-BY :: (RULE320 RULE321 RULE322 RULE323 RULE372 RULE002 RULE003
          RULE005 RULE006 RULE025 RULE066 RULE070 RULE073 RULE407
          RULE408 RULE411 SREFMARK RULE365 RULE364 )
  CONTAINED-IN :: (RULE323 RULE372 RULE002 RULE003 RULE005 RULE006
              RULE025 RULE066 RULE070 RULE073 RULE407 RULE408
              RULE411 )


TEST-RUN-DONE
=============
  TRANSLATION :: (Test run complete)
```

TYPE :: YES/NO
USED-BY :: (RULE292 RULE178)

TORQUE-AT-MIN-SPEED
TRANSLATION :: (torque at minimum speed)
PROMPT :: (What is the torque at minimum speed. as % of rated torque ?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE294 RULE380)
UPDATED-IN :: (RULE347 RULE395 RULE375)
USED-BY :: (RULE311 RULE312 RULE316)
RANGE :: (0 200)
CONTAINED-IN :: (RULE316)

TOTAL-INERTIA
TRANSLATION :: (inertia of the motor and load)
PROMPT :: ("What is the maximum inertia of the load (Kg-m2) ?" :LINE 2 "
(This information is necessary for peak power calculation of
the motor)")
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
ANTECEDENT-IN :: (RULE395 RULE314)
RANGE :: (0 10000)
CONTAINED-IN :: (RULE395 RULE314)

TYPE-OF-CONTROL
TRANSLATION :: (type of control of the ac drive system)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE299 RULE363 RULE362 RULE379)
USED-BY :: (RULE320 RULE321 RULE322 RULE323 RULE334)

TYPE-OF-CONVERTER
TRANSLATION :: (type of power converter topology for the application)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE298 RULE297 RULE335)
UPDATED-IN :: (RULE352 RULE392 RULE393 RULE399 RULE376)
USED-BY :: (RULE334)

TYPE-OF-MACHINE
TRANSLATION :: (type of ac machine for the application)
TYPE :: SINGLEVALUED
EXPECT :: (INDUCTION-MOTOR PERMANENT-MAGNET-SYNCHRONOUS-MOTOR
BRUSHLESS-DC-MOTOR)
UPDATED-BY :: (RULE300 RULE296 RULE374 RULE301)
ANTECEDENT-IN :: (RULE352 RULE392 RULE393)
USED-BY :: (RULE334)

TYPE-OF-PMAC-MACHINE
TRANSLATION :: (the type of pmac machine for the application)
PROMPT :: ("Do you have a choice for the type of permanent magnet ac
machine ?" :LINE 2 "(Select the one that applies)")
TYPE :: SINGLEVALUED
EXPECT :: (BRUSHLESS-DC-MOTOR PERMANENT-MAGNET-SYNCHRONOUS-MOTOR
DON'T-CARE)
UPDATED-BY :: (SREFMARK RULE369 RULE370)
UPDATED-IN :: (RULE391)
ANTECEDENT-IN :: (RULE368 RULE352)
USED-BY :: (RULE300 RULE374 RULE301 SREFMARK RULE369 RULE370)
HELP :: ("Brushless dc (or trapezoidal) PM motors are" :LINE "relatively

simple to control, and do not have" :LINE "costly shaft-mounted position sensor. They also" :LINE "cause 6th harmonic torque ripple. They require" :LINE "the stator-mounted hall-effect sensors for flux" :LINE "sensing." :LINE 2 "PMSMs, however, are necessary when high" :LINE "performance in the form of vector controlled" :LINE "operation is required. They need costly" :LINE "shaft-mounted absolute position sensor. The" :LINE " control complexity and the cost are also high.")
DEFAULT :: (BRUSHLESS-DC-MOTOR)

TYPE-OF-PUMP
TRANSLATION :: (the type of pump)
PROMPT :: (What is the type of your pump ?)
TYPE :: SINGLEVALUED
EXPECT :: (CENTRIFUGAL POSITIVE-DISPLACEMENT)
USED-BY :: (RULE294 RULE295)

TYPE-OF-SWITCH
TRANSLATION :: (the type of power switch selected for the inverter)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE302 RULE303)
ANTECEDENT-IN :: (RULE040 RULE405)
USED-BY :: (RULE039 RULE404 RULE064 RULE005 RULE070 RULE073 RULE407 RULE408 RULE411)
DEFAULT :: (IGBT)

ZERO-SPD-OPER
TRANSLATION :: (load requirement for zero spd operation)
PROMPT :: ("Does the application need zero or near-zero speed operation ?")
TYPE :: YES/NO
UPDATED-IN :: (RULE367 RULE366 RULE384 RULE400 RULE375)
ANTECEDENT-IN :: (RULE378 RULE377 RULE385)
USED-BY :: (RULE363)

AC-DRIVE-SYSTEM-DESIGN-RULES

RULE178
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
DESCRIPTION :: (drive system design and optimization done)
IF :: (CONFIG-DONE AND MOTOR-DESIGN-DONE AND ((CONVERTER-DESIGN? AND CONV-DES-DONE AND CONV-OPT-DONE) OR ! CONVERTER-DESIGN?) AND ((CONTROL-DESIGN? AND CONT-DES-DONE AND CONT-OPT-DONE AND TEST-RUN-DONE) OR ! CONTROL-DESIGN?))
THEN :: (DESOPT-DONE)

RULE179
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
DESCRIPTION :: (drive system design done)
IF :: (CONTROL-TYPE IS KNOWN AND CONV-TYPE IS KNOWN AND MACHINE-TYPE IS KNOWN)
THEN :: (DS-DES-DONE)

RULE271
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
ANTECEDENT :: YES

```
 DESCRIPTION :: (Invoke control design subframe)
 IF  :: (CONV-DES-DONE AND CONV-OPT-DONE AND APPL-TYPE =
      HIGH-PERFORMANCE AND CONTROL-DESIGN? )
 THEN :: (CONSIDERFRAME CONTROL-DESIGN)

RULE350
=======
 SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
 DESCRIPTION :: (load type and breakaway torque for crusher)
 IF  :: (APPLICATION = CRUSHER)
 THEN :: (LOAD-TYPE = CONSTANT-TORQUE AND BREAKAWAY-TORQUE = HIGH)

RULE351
=======
 SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
 DESCRIPTION :: (invokes ac drive configuration subframe)
 IF  :: (APPLICATION IS KNOWN)
 THEN :: (CONSIDERFRAME AC-DRIVE-CONFIGURATOR)


RULE353
=======
 SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (ask for user's choice for design/optimization of power
          converter )
 IF  :: (CONFIG-DONE AND MOTOR-DESIGN-DONE)
 THEN :: (E (VAL1 FRAME CONVERTER-DESIGN?))

RULE357
=======
 SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (min speed = base speed for winder (const hp appl))
 IF  :: (APPLICATION = WINDER OR APPLICATION = SPINDLE)
 THEN :: (NOPROMPT MIN-SPEED)

RULE359
=======
 SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (ask for type of pump, if appl = pump)
 IF  :: (APPLICATION = PUMP)
 THEN :: (E (VAL1 FRAME TYPE-OF-PUMP))

RULE367
=======
 SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (zero speed operation not reqd for spindle drive!)
 IF  :: (APPLICATION = SPINDLE)
 THEN :: (NOPROMPT ZERO-SPD-OPER AND NOFINDOUT ZERO-SPD-OPER AND !
      ZERO-SPD-OPER )

RULE375
=======
 SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (const torque high perf appl)
 IF  :: (APPLICATION = TENSION_CONTROL OR APPLICATION = ELEVATOR OR
      APPLICATION = *SPEED_CONTROL_SERVO OR APPLICATION = TRACTION
      OR APPLICATION = ROBOTIC_ARM )
 THEN :: (LOAD-TYPE = CONSTANT-TORQUE AND ZERO-SPD-OPER AND
      TORQUE-AT-MIN-SPEED = 100 AND NOPROMPT TORQUE-AT-MIN-SPEED AND
      NOPROMPT MIN-SPEED AND MIN-SPEED = 0 )
```

RULE376
=======
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (converter = dual-pwm for elevator or traction appl)
IF :: (APPLICATION = ELEVATOR OR APPLICATION = TRACTION)
THEN :: (NOPROMPT CHOICE-FOR-CONVERTER AND TYPE-OF-CONVERTER =
DUAL-PWM-CONVERTER AND MIN-SPEED = 0 AND NOPROMPT MAX-SPEED)

RULE383
=======
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (for EV appl, battery-fed conv)
IF :: (APPLICATION = ELECTRIC_VEHICLE)
THEN :: (NOPROMPT CHOICE-FOR-CONVERTER AND NOFINDOUT
CHOICE-FOR-CONVERTER AND NOPROMPT SUPPLY-INPUT-VOLT AND
NOFINDOUT SUPPLY-INPUT-VOLT AND NOPROMPT MIN-SPEED AND
MIN-SPEED = 0)

RULE386
=======
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
DESCRIPTION :: (invoke converter design on user's choice)
IF :: (! CONVERTER-DESIGN?)
THEN :: (CONV-DES-DONE AND NOFINDOUT CONV-DES-DONE)

RULE396
=======
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (Ask for drive control char to the user for user-defined
appl)
IF :: (APPLICATION = USER_DEFINED)
THEN :: (E (VAL1 FRAME DRIVE-CONTROL-CHAR))

RULE399
=======
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (for 4 quad oper, type of conv = dual pwm inv)
IF :: (APPLICATION = USER_DEFINED AND DRIVE-CONTROL-CHAR =
FOUR-QUADRANT-OPERATION)
THEN :: (TYPE-OF-CONVERTER = DUAL-PWM-INVERTER AND NOPROMPT
CHOICE-FOR-CONVERTER)

RULE400
=======
SUBJECT :: AC-DRIVE-SYSTEM-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (zero speed operation for the high-performance appl)
IF :: (APPLICATION = USER_DEFINED AND DRIVE-CONTROL-CHAR =
ZERO-SPEED-OPERATION)
THEN :: (ZERO-SPD-OPER AND MIN-SPEED = 0 AND NOPROMPT MIN-SPEED)

APPENDIX A.2 ARL Listing of Expert System for AC Drive Configuration Selection

```
==========================================
  Frame :: AC-DRIVE-CONFIGURATOR
==========================================

  IDENTIFIER :: "AC-DRIVE-CONFIGURATOR-"
  TRANSLATION :: (Configurator for the AC drive system)
  PARENTS :: (AC-DRIVE-SYSTEM-DESIGN)
  GOALS :: (TYPE-OF-MACHINE TYPE-OF-CONVERTER TYPE-OF-CONTROL
        TYPE-OF-SWITCH )
  INITIALDATA :: (CHOICE-FOR-MACHINE)
  PROMPTEVER :: ("The AC drive system suitable for your application will be
           configured now. The" :LINE "AC drive system consists of
           three main components, namely, the power" :LINE "electronic
           converter, the AC machine driving the load, and the control
           for" :LINE "the AC drive system. You will be asked
           questions so that the expert system" :LINE "can determine
           the TYPE of power converter topology, the TYPE of AC
           machine" :LINE "and the TYPE of control to be employed,
           depending on the type of" :LINE "application." )
  DISPLAYRESULTS :: YES
  PARMGROUP :: AC-DRIVE-CONFIGURATOR-PARMS
  RULEGROUPS :: (AC-DRIVE-CONFIGURATOR-RULES)
  AC-DRIVE-CONFIGURATOR-PARMS  :: (CHOICE-OF-VECTOR-CONTROL PMAC-MOTOR-CHAR
                    SPEED-SENSORLESS-OPER )
  AC-DRIVE-CONFIGURATOR-RULES  :: (RULE294 RULE295 RULE296 RULE297 RULE298
                    RULE299 RULE300 RULE301 RULE302 RULE303
                    RULE328 RULE329 RULE334 RULE335 RULE338
                    RULE341 RULE344 RULE345 RULE348 RULE352
                    RULE360 RULE361 RULE362 RULE363 RULE366
                    RULE368 RULE369 RULE370 RULE373 RULE374
                    RULE377 RULE378 RULE379 RULE384 RULE385
                    RULE391 RULE392 RULE393 RULE397 RULE398
                    )
   ==========================================
     AC-DRIVE-CONFIGURATOR-PARMS
   ==========================================
  CHOICE-OF-VECTOR-CONTROL
  ========================
   TRANSLATION :: (user's choice for type of vector control to be employed
            )
   PROMPT :: ("Your application is high-performance type, and requires
          vector control. Do" :LINE "you have a preference over the
          type of vector control to be employed?" )
   TYPE :: SINGLEVALUED
   EXPECT :: (DIRECT-VECTOR-CONTROL *INDIRECT-VECTOR-CONTROL NOT-SURE)
   ANTECEDENT-IN :: (RULE361 RULE366 RULE385)


  PMAC-MOTOR-CHAR
  ===============
   TRANSLATION :: (qualitative performance indices wanted from pmac motor)
   PROMPT :: ("Which of the following do you want from your permanent
          magnet AC motor drive?" :LINE 2 "(This information will
          decide the type of PMAC machine to be used." :LINE :ATTR (
          CYAN HIGH ) "You must select" :ATTR (CYAN REVERSE) "only one"
          :ATTR (CYAN HIGH) "of the choices.)" )
   TYPE :: ASK-ALL
   EXPECT :: (6TH-HARMONIC-TORQUE-PULSATIONS-BUT-SIMPLE-CONTROL
          COMPLEX-CONTROL-BUT-SMOOTH-OPERATION NOT-SURE )
   USED-BY :: (RULE374 RULE301 SREFMARK RULE369 RULE370)
```

SPEED-SENSORLESS-OPER

TRANSLATION :: (user's need for speed sensorless operation)
PROMPT :: ("Do you need to have speed sensorless operation for your high
performance" :LINE "application ?" :LINE 2 "(This information
is necessary to choose between Direct or Indirect Vector"
:LINE "Control)")
TYPE :: YES/NO
UPDATED-IN :: (RULE377 RULE366)
USED-BY :: (RULE362 RULE379)

AC-DRIVE-CONFIGURATOR-RULES

RULE294

SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
DESCRIPTION :: (load type = variable torque; breakaway torque = low -
for cf pump)
IF :: ((APPLICATION = PUMP AND TYPE-OF-PUMP = CENTRIFUGAL) OR
APPLICATION = FAN)
THEN :: (LOAD-TYPE = VARIABLE-TORQUE AND BREAKAWAY-TORQUE = LOW AND
NOPROMPT TORQUE-AT-MIN-SPEED AND MOHF = 1 AND IOSF_BA = 1)

RULE295

SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
DESCRIPTION :: (load type = constant torque; breakaway torque =
moderate for positive displ pump)
IF :: ((APPLICATION = PUMP AND TYPE-OF-PUMP = POSITIVE-DISPLACEMENT)
OR APPLICATION = COMPRESSOR OR APPLICATION = CONVEYOR)
THEN :: (LOAD-TYPE = CONSTANT-TORQUE AND BREAKAWAY-TORQUE = MODERATE)

RULE296

SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
DESCRIPTION :: (Type of mach = im, if user wants it or doesn't care)
IF :: (CHOICE-FOR-MACHINE != PERMANENT-MAGNET-AC-MOTOR)
THEN :: (TYPE-OF-MACHINE = INDUCTION-MOTOR)

RULE297

SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
DESCRIPTION :: (type of conv = r+i by user's choice or default)
IF :: (CHOICE-FOR-CONVERTER = *RECTIFIER-INVERTER OR (
CHOICE-FOR-CONVERTER = DON'T-CARE AND ! HIGH-INPUT-PF-REQD))
THEN :: (TYPE-OF-CONVERTER = RECTIFIER-INVERTER)

RULE298

SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
DESCRIPTION :: (type of conv = rect+boost chop + inv by user's choice
or pf need)
IF :: (CHOICE-FOR-CONVERTER = RECTIFIER-BOOST_CHOPPER-INVERTER OR
HIGH-INPUT-PF-REQD)
THEN :: (TYPE-OF-CONVERTER = RECTIFIER-BOOST_CHOPPER-INVERTER)

RULE299

SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
DESCRIPTION :: (type of control = open loop v/hz for low perf appl)
IF :: (APPL-TYPE = GENERAL-PURPOSE)
THEN :: (TYPE-OF-CONTROL = OPEN-LOOP-V/HZ)

RULE300

```
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (type of machine = brushless dc motor by choice)
 IF  :: (CHOICE-FOR-MACHINE = PERMANENT-MAGNET-AC-MOTOR AND
      TYPE-OF-PMAC-MACHINE = BRUSHLESS-DC-MOTOR )
 THEN :: (TYPE-OF-MACHINE = BRUSHLESS-DC-MOTOR)

RULE301
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (type of machine = permanent magnet synchronous motor by
          choice )
 IF  :: ((CHOICE-FOR-MACHINE = PERMANENT-MAGNET-AC-MOTOR AND
      TYPE-OF-PMAC-MACHINE = PERMANENT-MAGNET-SYNCHRONOUS-MOTOR ) OR
      (CHOICE-FOR-MACHINE = PERMANENT-MAGNET-AC-MOTOR AND
      TYPE-OF-PMAC-MACHINE = DON'T-CARE AND PMAC-MOTOR-CHAR =
      COMPLEX-CONTROL-BUT-SMOOTH-OPERATION ) )
 THEN :: (TYPE-OF-MACHINE = PERMANENT-MAGNET-SYNCHRONOUS-MOTOR)

RULE302
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (igbt selected as the type of switch)
 IF  :: ((CHOICE-OF-SWITCH != BJT AND LOW-ACOUSTIC-NOISE) OR (
      CHOICE-OF-SWITCH = DON'T-CARE AND LOW-ACOUSTIC-NOISE ) OR
      CHOICE-OF-SWITCH = IGBT )
 THEN :: (TYPE-OF-SWITCH = IGBT)

RULE303
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (bjt selected as the type of switch)
 IF  :: (CHOICE-OF-SWITCH = BJT)
 THEN :: (TYPE-OF-SWITCH = BJT AND NOPROMPT LOW-ACOUSTIC-NOISE AND
      NOFINDOUT LOW-ACOUSTIC-NOISE )

RULE328
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (application type = general purpose)
 IF  :: ((APPLICATION = PUMP OR APPLICATION = FAN OR APPLICATION =
      COMPRESSOR OR APPLICATION = CRUSHER OR APPLICATION = WINDER OR
      APPLICATION = CONVEYOR ) AND APPLICATION != USER_DEFINED )
 THEN :: (APPL-TYPE = GENERAL-PURPOSE)

RULE329
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (type of appl = high performance)
 IF  :: ((APPLICATION = SPINDLE OR APPLICATION = TENSION_CONTROL OR
      APPLICATION = ELEVATOR OR APPLICATION = *SPEED_CONTROL_SERVO
      OR APPLICATION = ROBOTIC_ARM OR APPLICATION = TRACTION OR
      APPLICATION = ELECTRIC_VEHICLE ) AND APPLICATION !=
      USER_DEFINED )
 THEN :: (APPL-TYPE = HIGH-PERFORMANCE AND CTRL-TYPE = VECTOR-CONTROL)

RULE334
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (sets configuration done flag)
 IF  :: (TYPE-OF-MACHINE IS KNOWN AND TYPE-OF-CONVERTER IS KNOWN AND
      TYPE-OF-CONTROL IS KNOWN )
 THEN :: (CONFIG-DONE)

RULE335
```

```
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (chooses type of converter = dual pwm)
 IF  :: (APPL-TYPE = HIGH-PERFORMANCE AND APPLICATION = USER-DEFINED
       AND (SPEED-REVERSIBILITY OR CHOICE-FOR-CONVERTER =
       DUAL-PWM-INVERTER ) )
 THEN :: (TYPE-OF-CONVERTER = DUAL-PWM-INVERTER)

RULE338
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (asks for type of pump, if appl = pump)
 IF  :: (APPLICATION = PUMP)
 THEN :: (E (VAL1 FRAME TYPE-OF-PUMP))

RULE341
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (don't ask for max speed for gen purpose, vtl / non
          const hp appl )
 IF  :: (LOAD-TYPE = VARIABLE-TORQUE OR (APPL-TYPE = GENERAL-PURPOSE
       AND LOAD-TYPE != CONST-HP ) )
 THEN :: (NOPROMPT MAX-SPEED)

RULE344
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (choice of inverter = r+i ==> high input pf not reqd)
 IF  :: (CHOICE-FOR-CONVERTER = *RECTIFIER-INVERTER)
 THEN :: (NOPROMPT HIGH-INPUT-PF-REQD AND NOFINDOUT HIGH-INPUT-PF-REQD
       AND ! HIGH-INPUT-PF-REQD )

RULE345
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (dont invoke ac motor design if user doesn't want to)
 IF  :: (! MOTOR-DESIGN?)
 THEN :: (DONTCONSIDERFRAME AC-MOTOR-DESIGN AND MOTOR-DESIGN-DONE)

RULE348
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (ask for converter system design if ac drive
          configuration and motor design done )
 IF  :: (CONFIG-DONE AND MOTOR-DESIGN? AND MOTOR-DESIGN-DONE)
 THEN :: (E (VAL1 FRAME CONVERTER-DESIGN?))

RULE352
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (special converter topology for brushless dc motor)
 IF  :: ((CHOICE-FOR-MACHINE = PERMANENT-MAGNET-AC-MOTOR AND
       TYPE-OF-PMAC-MACHINE = BRUSHLESS-DC-MOTOR ) OR TYPE-OF-MACHINE
       = BRUSHLESS-DC-MOTOR )
 THEN :: (NOPROMPT TYPE-OF-CONVERTER AND NOFINDOUT TYPE-OF-CONVERTER AND
       TYPE-OF-CONVERTER = "Brushless DC motor converter" )

RULE360
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
```

```
 ANTECEDENT :: YES
 DESCRIPTION :: (ask for user's choice of vector control)
 IF  :: (CTRL-TYPE = VECTOR-CONTROL)
 THEN :: (E (VAL1 FRAME CHOICE-OF-VECTOR-CONTROL))

RULE361
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Ask for 0 spd oper for vector control type)
 IF  :: (CHOICE-OF-VECTOR-CONTROL = NOT-SURE)
 THEN :: (E (VAL1 FRAME ZERO-SPD-OPER))

RULE362
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (for spd sensorless operation. dvc chosen)
 IF  :: (SPEED-SENSORLESS-OPER)
 THEN :: (TYPE-OF-CONTROL = DIRECT-VECTOR-CONTROL)

RULE363
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (indirect vector control for zero speed operation)
 IF  :: (ZERO-SPD-OPER)
 THEN :: (TYPE-OF-CONTROL = INDIRECT-VECTOR-CONTROL)

RULE366
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (dont ask for zero spd oper or speed sensorless oper for
        choice of VC known )
 IF  :: (CHOICE-OF-VECTOR-CONTROL != NOT-SURE)
 THEN :: (NOPROMPT SPEED-SENSORLESS-OPER AND NOPROMPT ZERO-SPD-OPER AND
      NOFINDOUT SPEED-SENSORLESS-OPER AND NOFINDOUT ZERO-SPD-OPER )

RULE368
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (find out user's preference on which pmac machine to use
        )
 IF  :: (TYPE-OF-PMAC-MACHINE = DON'T-CARE)
 THEN :: (E (VAL1 FRAME PMAC-MOTOR-CHAR))

RULE369
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DOBEFORE :: (373)
 DESCRIPTION :: (type of pmac machine = brushless dc)
 IF  :: (TYPE-OF-PMAC-MACHINE = DON'T-CARE AND PMAC-MOTOR-CHAR =
      6TH-HARMONIC-TORQUE-PULSATIONS-BUT-SIMPLE-CONTROL AND
      PMAC-MOTOR-CHAR != COMPLEX-CONTROL-BUT-SMOOTH-OPERATION )
 THEN :: (TYPE-OF-PMAC-MACHINE = BRUSHLESS-DC-MOTOR)

RULE370
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (type of machine = PMSM for user's need for shaft
        position encoder )
 IF  :: (TYPE-OF-PMAC-MACHINE = DON'T-CARE AND PMAC-MOTOR-CHAR !=
      6TH-HARMONIC-TORQUE-PULSATIONS-BUT-SIMPLE-CONTROL AND
      PMAC-MOTOR-CHAR = COMPLEX-CONTROL-BUT-SMOOTH-OPERATION )
 THEN :: (TYPE-OF-PMAC-MACHINE = PERMANENT-MAGNET-SYNCHRONOUS-MOTOR)
```

```
RULE373
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (find out user's choice for PMAC motor)
 IF  :: (CHOICE-FOR-MACHINE = PERMANENT-MAGNET-AC-MOTOR)
 THEN :: (E (VALI FRAME TYPE-OF-PMAC-MACHINE))

RULE374
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (default pmac machine = brushless dc motor)
 IF  :: (CHOICE-FOR-MACHINE = PERMANENT-MAGNET-AC-MOTOR AND
      TYPE-OF-PMAC-MACHINE = DON'T-CARE AND PMAC-MOTOR-CHAR =
      NOT-SURE )
 THEN :: (TYPE-OF-MACHINE = BRUSHLESS-DC-MOTOR)

RULE377
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (for zero spd oper, control = ivc and spd sensorless
          oper not possible )
 IF  :: (ZERO-SPD-OPER)
 THEN :: (! SPEED-SENSORLESS-OPER)


RULE378
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (ask for speed sensorless operation only if zero spd
          operation is not necessary )
 IF  :: (! ZERO-SPD-OPER)
 THEN :: (E (VALI FRAME SPEED-SENSORLESS-OPER))

RULE379
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (vector control defaults to ivc for no input from user)
 IF  :: (APPL-TYPE = HIGH-PERFORMANCE AND ! SPEED-SENSORLESS-OPER AND
      APPLICATION != USER_DEFINED )
 THEN :: (TYPE-OF-CONTROL = INDIRECT-VECTOR-CONTROL)

RULE384
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (zero speed operation for non-spindle and
          non-tension-control appl )
 IF  :: (APPLICATION != SPINDLE AND APPLICATION != TENSION_CONTROL)
 THEN :: (ZERO-SPD-OPER)

RULE385
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Warning if Zero spd operation and Direct Vector Control
          chosen )
 IF  :: (ZERO-SPD-OPER AND CHOICE-OF-VECTOR-CONTROL =
      DIRECT-VECTOR-CONTROL )
 THEN :: (PRINT "Your application requires operation of machine at zero
      speed, which cannot be provided by your choice of DIRECT VECTOR
      CONTROL. Please choose INDIRECT VECTOR CONTROL as type of
      control, for valid consultation. (For this, press F2, choose
```

```
      REVIEW and follow the instructions on the screen)" AND
      MIN-SPEED = 0 AND NOPROMPT MIN-SPEED AND
      MOTOR-ENCLOSURE-VENTILLATION = TEFC )

RULE391
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (don't find out type of pmac motor if im chosen!)
 IF  :: (CHOICE-FOR-MACHINE = INDUCTION-MOTOR)
 THEN :: (NOPROMPT TYPE-OF-PMAC-MACHINE AND NOFINDOUT
      TYPE-OF-PMAC-MACHINE )

RULE392
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (for ev and im or pmsm, battery-fed inverter chosen)
 IF  :: (APPLICATION = ELECTRIC_VEHICLE AND (TYPE-OF-MACHINE =
      INDUCTION-MOTOR OR TYPE-OF-MACHINE =
      PERMANENT-MAGNET-SYNCHRONOUS-MOTOR ) )
 THEN :: (TYPE-OF-CONVERTER = Battery-fed PWM inverter)

RULE393
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (for bldc ev drive, battery-fed bldc conv chosen)
 IF  :: (APPLICATION = ELECTRIC_VEHICLE AND TYPE-OF-MACHINE =
      BRUSHLESS-DC-MOTOR )
 THEN :: (TYPE-OF-CONVERTER = Battery-fed Brushless-DC motor converter)

RULE397
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (appl type = high-performance for user-defined appl)
 IF  :: (APPLICATION = USER_DEFINED AND (DRIVE-CONTROL-CHAR =
      FREQUENT-SPEED-REVERSALS OR DRIVE-CONTROL-CHAR =
      ZERO-SPEED-OPERATION OR DRIVE-CONTROL-CHAR =
      PRECISE-SPEED-CONTROL OR DRIVE-CONTROL-CHAR =
      PRECISE-TORQUE-CONTROL OR DRIVE-CONTROL-CHAR =
      PRECISE-POSITION-CONTROL OR DRIVE-CONTROL-CHAR =
      FOUR-QUADRANT-OPERATION ) )
 THEN :: (APPL-TYPE = HIGH-PERFORMANCE)

RULE398
=======
 SUBJECT :: AC-DRIVE-CONFIGURATOR-RULES
 DESCRIPTION :: (appl type = general-purpose for user-defined appl)
 IF  :: (APPLICATION = USER_DEFINED AND ! (DRIVE-CONTROL-CHAR =
      FREQUENT-SPEED-REVERSALS OR DRIVE-CONTROL-CHAR =
      ZERO-SPEED-OPERATION OR DRIVE-CONTROL-CHAR =
      PRECISE-SPEED-CONTROL OR DRIVE-CONTROL-CHAR =
      PRECISE-TORQUE-CONTROL OR DRIVE-CONTROL-CHAR =
      PRECISE-POSITION-CONTROL OR DRIVE-CONTROL-CHAR =
      FOUR-QUADRANT-OPERATION ) )
 THEN :: (APPL-TYPE = GENERAL-PURPOSE)
```

APPENDIX A.3 ARL Listing of Expert System for AC Motor Design

```
Frame :: AC-MOTOR-DESIGN

  IDENTIFIER :: "AC-MOTOR-DESIGN-"
  TRANSLATION :: (Design of the AC motor for the application)
  PARENTS :: (AC-DRIVE-SYSTEM-DESIGN)
  GOALS :: (MOTOR-OUTPUT-POWER MOTOR-MAX-POWER-RATING MOTOR-NO-OF-POLES
        MOTOR-VOLTAGE-RATING MOTOR-BASE-FREQ MOTOR-RATED-CURRENT MSF
        MOTOR-ENCLOSURE-VENTILLATION NEMA-MOTOR-DESIGN
        INV-CONT-POWER-RATING )
  INITIALDATA :: (INFO-ON-POWER SUPPLY-INPUT-VOLT BASE-SPEED MIN-SPEED
           ENVIRONMENTAL-CONDITION TOTAL-INERTIA ACCN-DCCN-TIME
           NO-OF-PHASES )
  PROMPTEVER :: ("The AC drive system for your application is now
           configured. The next few" :LINE "questions will be asked so
           that the AC motor selected can be designed in" :LINE "terms
           of its nameplate specs." :LINE 2 "The expert system expects
           you to know the following mechanical information" :LINE "

about your application:  (Metric system is used in the computations)" :LINE 2 "
                  The" :ATTR (CYAN HIGH) "steady-state power /
           steady-state" :ATTR (WHITE HIGH) :ATTR (CYAN HIGH) "torque"
           :ATTR (WHITE HIGH) "required" :LINE "       " :ATTR (CYAN
           HIGH ) "Base speed" :ATTR (WHITE HIGH) "of the application"
           :LINE "       " :ATTR (CYAN HIGH) "Minimum speed" :ATTR (
           WHITE HIGH ) "of operation" :LINE "       " :ATTR (CYAN
           HIGH ) "Load inertia" :LINE :ATTR (WHITE HIGH) "       "
           :ATTR (CYAN HIGH) "Minimum acceleration / deceleration
           time," :ATTR (WHITE HIGH) "for min <--> max speed" :LINE "
                    transition" :LINE 2 "The following
           defaults are employed, while designing the motor for your"
           :LINE "application:" :LINE 2 "       Number of poles = 4,
           Service Factor = 1.15, NEMA design = Class B" )
  DISPLAYRESULTS :: YES
  PARMGROUP :: AC-MOTOR-DESIGN-PARMS
  RULEGROUPS :: (AC-MOTOR-DESIGN-RULES)
  AC-MOTOR-DESIGN-PARMS  :: (BATTERY-VOLT)
  AC-MOTOR-DESIGN-RULES  :: (RULE020 RULE022 RULE304 RULE305 RULE306
                  RULE308 RULE309 RULE310 RULE311 RULE312
                  RULE314 RULE315 RULE316 RULE317 RULE318
                  RULE319 RULE320 RULE321 RULE322 RULE323
                  RULE324 RULE325 RULE326 RULE327 RULE330
                  RULE331 RULE332 RULE333 RULE337 RULE339
                  RULE340 RULE342 RULE343 RULE346 RULE347
                  RULE349 RULE355 RULE356 RULE358 RULE364
                  RULE365 RULE371 RULE372 RULE380 RULE381
                  RULE382 RULE394 RULE395 RULE401 RULE402
                  RULE403 RULE409 RULE410 RULE412 RULE413 )


   AC-MOTOR-DESIGN-PARMS


  BATTERY-VOLT
  ============
   TRANSLATION :: (battery voltage for battery-driven appl)
   PROMPT :: ("What is the voltage of the battery bank feeding the inverter
         ?" )
   TYPE :: SINGLEVALUED
   EXPECT :: POSITIVE-NUMBER
```

```
USED-BY :: (RULE394)
RANGE :: (1 1000)
CONTAINED-IN :: (RULE394)

==========
AC-MOTOR-DESIGN-RULES
==========

RULE020
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (warning if output power > 25 and no.of phases != 3
        phase )
INVALID :: YES
IF  :: (OUTPUT-POWER > 25 AND NO-OF-PHASES != THREE_PHASE)
THEN :: (PRINT NAME " -- The power rating for which the inverter is to
     be designed requires the input and output a.c. voltages of the
     inverter to be 3-phase . So please enter the correct value for
     this parameters before the valid consultation can continue.
     (Select New Start from the command menu available by pressing
     F2 or clicking right mouse button.)" )

RULE022
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (warning if output power < 5 and no.of phases != single
        phase )
INVALID :: YES
IF  :: (OUTPUT-POWER <= 5 AND NO-OF-PHASES = THREE_PHASE)
THEN :: (PRINT "It is not necessary for the drive to have a 3-phase
     supply at the inverter input or output to drive the motor of
     such low hp. If you wish to modify this input. select NEW START
     from the selection menu available by pressing F2 (or by
     clicking RIGHT mouse button). You may continue the consultation
     otherwise." )

RULE304
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (Prints message if min speed >= max speed!)
IF  :: (MIN-SPEED >= MAX-SPEED)
THEN :: (PRINT "The minimum speed you entered is MORE than the maximum
     speed you need. Please reenter the speed range information, and
     then continue the consultation" )

RULE305
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (base speed range = base speed / min speed)
IF  :: (MIN-SPEED IS KNOWN AND BASE-SPEED IS KNOWN)
THEN :: (BASE-SPEED-RANGE = (BASE-SPEED / MIN-SPEED))

RULE306
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (calculates motor base frequency)
IF  :: (BASE-SPEED IS KNOWN)
THEN :: (MOTOR-BASE-FREQ = (FIX (BASE-SPEED / 30) + 1))

RULE308
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
```

```
ANTECEDENT :: YES
DESCRIPTION :: (output power ( kW ) = torque * speed)
IF  :: (INFO-ON-POWER = STEADY-STATE-TORQUE AND SS-TORQUE IS KNOWN AND
      BASE-SPEED IS KNOWN )
THEN :: (MOTOR-SS-POWER = ((SS-TORQUE * BASE-SPEED) / 9550))

RULE309
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (enclosure = tenv for hazardous env)
IF  :: (ENVIRONMENTAL-CONDITION)
THEN :: (MOTOR-ENCLOSURE-VENTILLATION = TENV)

RULE310
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (motor overheating factor = 1 for speed range < 2)
IF  :: (BASE-SPEED-RANGE < 2)
THEN :: (MOHF = 1)


RULE311
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (motor oversizing factor computation due to cont load
        low spd oper )
IF  :: (BASE-SPEED-RANGE >= 2 AND BASE-SPEED-RANGE < 6 AND
      TORQUE-AT-MIN-SPEED IS KNOWN AND TORQUE-AT-MIN-SPEED > 60 AND
      LOAD-TYPE = CONSTANT-TORQUE )
THEN :: (MOHF = 1)


RULE312
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (mohf and motor encl and vent for spd range > 6
        non-hazardous env )
IF  :: (BASE-SPEED-RANGE >= 6 AND TORQUE-AT-MIN-SPEED > 60)
THEN :: (MOHF = 1)

RULE314
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (calculates acceleration torque)
IF  :: (TOTAL-INERTIA IS KNOWN AND MAX-SPEED IS KNOWN AND MIN-SPEED IS
      KNOWN AND ACCN-DCCN-TIME IS KNOWN AND APPLICATION !=
      ELECTRIC_VEHICLE )
THEN :: (ACCN-TORQUE = (((TOTAL-INERTIA * (MAX-SPEED - MIN-SPEED)) /
      9.55 ) / ACCN-DCCN-TIME ) )

RULE315
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (motor peak power calculated, and oversizing factor
        calculated )
IF  :: (ACCN-TORQUE IS KNOWN AND SS-TORQUE IS KNOWN)
THEN :: (MOSF_ACC = 1)

RULE316
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (inv oversizing factor due to acceleration computed)
IF  :: (ACCN-TORQUE IS KNOWN AND TORQUE-AT-MIN-SPEED IS KNOWN AND
```

```
      SS-TORQUE IS KNOWN )
 THEN :: (IOSF_ACC = ((ACCN-TORQUE + (0.5 * (((TORQUE-AT-MIN-SPEED / 100
      ) * SS-TORQUE ) + SS-TORQUE ) ) ) / (1.5 * SS-TORQUE) ) )

RULE317
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (motor oversizing factor = overheating factor if mohf >
           mosf_acc )
 IF  :: (MOHF IS KNOWN AND MOSF_ACC IS KNOWN AND MOHF > MOSF_ACC AND
      MOHF > 1. )
 THEN :: (MOSF = 1.)

RULE318
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (mosf = mosf_acc if mosf_acc >= mohf)
 IF  :: (MOHF IS KNOWN AND MOSF_ACC IS KNOWN AND MOHF <= MOSF_ACC AND
      MOSF_ACC > 1 )
 THEN :: (MOSF = 1.)

RULE319
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (calculates motor output power rating based on various
           oversizing factors )
 IF  :: (MOTOR-SS-POWER IS KNOWN AND MOSF IS KNOWN)
 THEN :: (MOTOR-OUTPUT-POWER = (FIX (MOSF * MOTOR-SS-POWER)))

RULE320
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (motor voltage rating = 220 for motor output power < 15
           kW )
 IF  :: (SUPPLY-INPUT-VOLT = DON'T-CARE AND MOTOR-OUTPUT-POWER < 15 AND
      TYPE-OF-CONTROL = OPEN-LOOP-V/HZ )
 THEN :: (MOTOR-VOLTAGE-RATING = 220)

RULE321
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (motor voltage rating = 220 for power range b/w 15 and
           20 kw by default )
 IF  :: (SUPPLY-INPUT-VOLT = DON'T-CARE AND MOTOR-OUTPUT-POWER >= 15
      AND MOTOR-OUTPUT-POWER < 20 AND TYPE-OF-CONTROL =
      OPEN-LOOP-V/HZ )
 THEN :: (MOTOR-VOLTAGE-RATING = 220)

RULE322
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (motor voltage rating = 460 V for power > 20 kW)
 IF  :: (SUPPLY-INPUT-VOLT = DON'T-CARE AND MOTOR-OUTPUT-POWER >= 20
      AND TYPE-OF-CONTROL = OPEN-LOOP-V/HZ )
 THEN :: (MOTOR-VOLTAGE-RATING = 460)

RULE323
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (for output power between 15 and 20 kW, user can dictate
           motor voltage rating )
 IF  :: (SUPPLY-INPUT-VOLT != DON'T-CARE AND MOTOR-OUTPUT-POWER >= 15
      AND MOTOR-OUTPUT-POWER < 20 AND TYPE-OF-CONTROL =
      OPEN-LOOP-V/HZ )
 THEN :: (MOTOR-VOLTAGE-RATING = (1 * SUPPLY-INPUT-VOLT))
```

```
RULE324
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (inverter oversizing factor = breakaway torque factor if
          ba > accn torque )
 IF  :: (IOSF_BA IS KNOWN AND IOSF_ACC IS KNOWN AND IOSF_BA > IOSF_ACC)
 THEN :: (IOSF = (1 * IOSF_BA))

RULE325
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (iosf = max (iosf_ba, iosf_acc))
 IF  :: (IOSF_BA IS KNOWN AND IOSF_ACC IS KNOWN AND IOSF_BA <= IOSF_ACC
      )
 THEN :: (IOSF = (1 * IOSF_ACC))

RULE326
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (calculates motor rated current)
 NOTE :: "754 = 1 / (1.732 * 0.85 * 0.9), 0.9 = motor eff, 0.85 = pf"
 IF  :: (MOTOR-SS-POWER IS KNOWN AND MOTOR-VOLTAGE-RATING IS KNOWN)
 THEN :: (MOTOR-RATED-CURRENT = (FIX ((754 * MOTOR-OUTPUT-POWER) /
      MOTOR-VOLTAGE-RATING ) ) )

RULE327
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Warning for too low a voltage for higher output power)
 IF  :: (SUPPLY-INPUT-VOLT = 220 AND MOTOR-SS-POWER > 20 AND
      APPLICATION != ELECTRIC_VEHICLE )
 THEN :: (PRINT "Your application demands more power than the low input
      voltage you have given. Please enter higher voltage value so
      that the consultation can continue. (For this, Press F2, select
      REVIEW and follow the instructions on the screen)" )

RULE330
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (motor ss power = appl power)
 IF  :: (INFO-ON-POWER = OUTPUT-POWER AND APPL-POWER IS KNOWN)
 THEN :: (MOTOR-SS-POWER = (1. * APPL-POWER))

RULE331
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (breakaway torque = high, load type = const torq for
          crusher )
 IF  :: (APPLICATION = CRUSHER)
 THEN :: (LOAD-TYPE = CONSTANT-TORQUE AND BREAKAWAY-TORQUE = HIGH)

RULE332
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (type of load = const hp, breakaway torque = moderate
          for winder )
 IF  :: (APPLICATION = WINDER)
 THEN :: (LOAD-TYPE = CONST-HP AND BREAKAWAY-TORQUE = MODERATE AND MOHF
      = 1 )

RULE333
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (Breakaway torque-related inverter oversizing computed)
```

```
 IF  :: (BREAKAWAY-TORQUE = HIGH AND BREAKAWAY-TORQUE-VALUE IS KNOWN)
 THEN :: (IOSF_BA = (BREAKAWAY-TORQUE-VALUE / 150))

RULE337
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (sets motor design done flag)
 IF  :: (MOTOR-OUTPUT-POWER IS KNOWN AND MOTOR-NO-OF-POLES IS KNOWN AND
       MOTOR-VOLTAGE-RATING IS KNOWN AND MOTOR-BASE-FREQ IS KNOWN AND
       MSF IS KNOWN AND NEMA-MOTOR-DESIGN IS KNOWN AND
       MOTOR-ENCLOSURE-VENTILLATION IS KNOWN AND (E (VAL1 FRAME
       IOSF_BA ) > 0 ) AND (E (VAL1 FRAME IOSF_ACC) > 0) AND (E (VAL1
       FRAME IOSF ) > 0 ) AND ! MOTOR-DESIGN-DONE )
 THEN :: (MOTOR-DESIGN-DONE)

RULE339
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (find output power if available, and DONT fine ss torque
           )
 IF  :: (INFO-ON-POWER = OUTPUT-POWER)
 THEN :: ((E (VAL1 FRAME APPL-POWER)) AND NOPROMPT SS-TORQUE)

RULE340
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (mosf = 1 if mohf <1, mosf_acc<1)
 IF  :: (MOHF IS KNOWN AND MOSF_ACC IS KNOWN AND MOHF < 1 AND MOSF_ACC
       < 1. )
 THEN :: (MOSF = 1)

RULE342
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (ask for ss torque if user knows it, DONT prompt for ss
           power )
 IF  :: (INFO-ON-POWER = STEADY-STATE-TORQUE)
 THEN :: ((E (VAL1 FRAME SS-TORQUE)) AND NOPROMPT APPL-POWER)

RULE343
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (calculate steady-state power for motor from T and w)
 IF  :: (SS-TORQUE IS KNOWN AND BASE-SPEED IS KNOWN)
 THEN :: (MOTOR-SS-POWER = (FIX ((SS-TORQUE * BASE-SPEED) / 9550)))

RULE346
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (enclosure = tenv for hazardous env)
 IF  :: (ENVIRONMENTAL-CONDITION)
 THEN :: (MOTOR-ENCLOSURE-VENTILLATION = TENV)

RULE347
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (torque at min speed = 100% for constant torque appl)
 IF  :: (LOAD-TYPE = CONSTANT-TORQUE OR APPL-TYPE = HIGH-PERFORMANCE)
 THEN :: (NOPROMPT TORQUE-AT-MIN-SPEED AND TORQUE-AT-MIN-SPEED = 100)

RULE349
=======
```

```
SUBJECT :: AC-MOTOR-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (evaluate breakaway torque requirement for crusher appl)
IF  :: (APPLICATION = CRUSHER)
THEN :: (E (VAL1 FRAME BREAKAWAY-TORQUE))

RULE355
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (calculates ss torque from power and speed)
IF  :: (MOTOR-SS-POWER IS KNOWN AND BASE-SPEED IS KNOWN)
THEN :: (SS-TORQUE = ((MOTOR-SS-POWER * 9550) / BASE-SPEED))

RULE356
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (calculates inverter continuous power rating for high
        perf appl )
IF  :: (MOTOR-MAX-POWER-RATING IS KNOWN AND APPL-TYPE =
     HIGH-PERFORMANCE AND MOTOR-MAX-POWER-RATING >
     MOTOR-OUTPUT-POWER )
THEN :: (INV-CONT-POWER-RATING = (FIX ((1.1 * MOTOR-MAX-POWER-RATING) /
     (1.5 * 0.9 ) ) ) )

RULE358
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (min speed = base speed for winder)
IF  :: ((APPLICATION = WINDER OR APPLICATION = SPINDLE) AND BASE-SPEED
     IS KNOWN )
THEN :: (MIN-SPEED = (1 * BASE-SPEED) AND (E (VAL1 FRAME MAX-SPEED)))

RULE364
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (motor voltage rating = 220 * 0.822 for <15 kW vector
        control appl )
IF  :: ((SUPPLY-INPUT-VOLT = DON'T-CARE OR SUPPLY-INPUT-VOLT = 220)
     AND MOTOR-OUTPUT-POWER < 15 AND CTRL-TYPE = VECTOR-CONTROL AND
     APPLICATION != ELECTRIC_VEHICLE )
THEN :: (MOTOR-VOLTAGE-RATING = (0.822 * 220) AND SUPPLY-INPUT-VOLT =
     220 )

RULE365
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
DESCRIPTION :: (motor voltage = 0.822 * 460 for power > 15kW or by user
        choice )
IF  :: ((SUPPLY-INPUT-VOLT = DON'T-CARE OR SUPPLY-INPUT-VOLT = 460)
     AND MOTOR-OUTPUT-POWER >= 15 AND CTRL-TYPE = VECTOR-CONTROL
     AND APPLICATION != ELECTRIC_VEHICLE )
THEN :: (MOTOR-VOLTAGE-RATING = (0.822 * 460) AND SUPPLY-INPUT-VOLT =
     460 )

RULE371
=======
SUBJECT :: AC-MOTOR-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (asks for proper input supply voltage for P > 20 kW, Vs
        = 220V )
IF  :: (MOTOR-OUTPUT-POWER IS KNOWN AND SUPPLY-INPUT-VOLT IS KNOWN AND
     CTRL-TYPE = VECTOR-CONTROL AND MOTOR-OUTPUT-POWER > 20 AND
     SUPPLY-INPUT-VOLT = 220 AND APPLICATION != ELECTRIC_VEHICLE )
```

```
 THEN :: (PRINT "Your voltage selection of 220 V is inapproriate for the
       power to be delivered to your application. Please enter the
       correct value of voltage to proceed with the design. (Press F2,
       select REVIEW and change supply input voltage to 460, from 220
       V, before proceeding further)" )

RULE372
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (motor voltage rating for 15 <= P < 20 as input by user)
 IF  :: ((SUPPLY-INPUT-VOLT = 220 OR SUPPLY-INPUT-VOLT = 460) AND
       MOTOR-OUTPUT-POWER >= 15 AND MOTOR-OUTPUT-POWER < 20 AND
       CTRL-TYPE = VECTOR-CONTROL )
 THEN :: (MOTOR-VOLTAGE-RATING = (0.822 * SUPPLY-INPUT-VOLT))

RULE380
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (max speed = base speed for non-const-hp appl)
 IF  :: (APPLICATION != SPINDLE AND APPLICATION != WINDER AND
       APPLICATION != ELECTRIC_VEHICLE )
 THEN :: (MAX-SPEED = (1 * BASE-SPEED) AND NOPROMPT MAX-SPEED AND
       TORQUE-AT-MIN-SPEED = 100 )

RULE381
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (for output power > 15 kW, unknown motor voltage rating
          set to 220 )
 IF  :: (MOTOR-OUTPUT-POWER < 15 AND MOTOR-VOLTAGE-RATING IS NOTKNOWN
       AND APPL-TYPE = GENERAL-PURPOSE )
 THEN :: (MOTOR-VOLTAGE-RATING = 220)

RULE382
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (motor power >= 15 ==> motor voltage rating = 460)
 IF  :: (MOTOR-OUTPUT-POWER >= 15 AND MOTOR-VOLTAGE-RATING IS NOTKNOWN
       AND APPL-TYPE = GENERAL-PURPOSE )
 THEN :: (MOTOR-VOLTAGE-RATING = 460)

RULE394
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (calculates motor voltage rating from battery voltage)
 IF  :: (APPLICATION = ELECTRIC_VEHICLE AND BATTERY-VOLT IS KNOWN)
 THEN :: (MOTOR-VOLTAGE-RATING = (FIX (0.58124 * BATTERY-VOLT) + 1))

RULE395
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (accleration torque for ev appl)
 IF  :: (TOTAL-INERTIA IS KNOWN AND BASE-SPEED IS KNOWN AND MIN-SPEED
       IS KNOWN AND ACCN-DCCN-TIME IS KNOWN AND APPLICATION =
       ELECTRIC_VEHICLE )
 THEN :: (ACCN-TORQUE = (((TOTAL-INERTIA * (BASE-SPEED - MIN-SPEED)) /
       9.55 ) / ACCN-DCCN-TIME ) AND TORQUE-AT-MIN-SPEED = 100 )

RULE401
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (Calculates peak motor power rating)
 IF  :: (ACCN-TORQUE IS KNOWN AND BASE-SPEED IS KNOWN AND APPL-TYPE =
       HIGH-PERFORMANCE AND ACCN-TORQUE > SS-TORQUE )
```

```
 THEN :: (MOTOR-MAX-POWER-RATING = (FIX ((ACCN-TORQUE * BASE-SPEED) /
      9550 ) ) )

RULE402
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (motor max power rating for non ev appl)
 IF  :: (ACCN-TORQUE IS KNOWN AND BASE-SPEED IS KNOWN AND SS-TORQUE IS
      KNOWN AND APPL-TYPE = GENERAL-PURPOSE )
 THEN :: (MOTOR-MAX-POWER-RATING = (FIX (((ACCN-TORQUE + SS-TORQUE) *
      BASE-SPEED ) / 9550 ) ) )

RULE403
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (inv cont power rating for gen purpose appl)
 IF  :: (MOTOR-OUTPUT-POWER IS KNOWN AND APPL-TYPE = GENERAL-PURPOSE)
 THEN :: (INV-CONT-POWER-RATING = (FIX ((1.1 * MOTOR-OUTPUT-POWER) / 0.9
      ) ) )

RULE409
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (max speed = base speed for elevator)
 IF  :: (BASE-SPEED IS KNOWN AND APPLICATION = ELEVATOR)
 THEN :: (MAX-SPEED = (1 * BASE-SPEED))

RULE410
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 IF  :: (ACCN-TORQUE IS KNOWN AND BASE-SPEED IS KNOWN AND APPL-TYPE =
      HIGH-PERFORMANCE AND ACCN-TORQUE <= SS-TORQUE )
 THEN :: (MOTOR-MAX-POWER-RATING = (FIX ((SS-TORQUE * BASE-SPEED) / 9550
      ) + 1 ) )

RULE412
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Prints warning if supply is single phase for > 2kW
          motor )
 IF  :: (NO-OF-PHASES = SINGLE_PHASE AND (MOTOR-MAX-POWER-RATING > 2 OR
      APPL-POWER > 2 ) )
 THEN :: (PRINT NAME "-- The power rating of the motor requires a three
      phase supply at the input, as against the single phase supply
      you have indicated. Please correct this input, before the
      design can proceed further. (To do so, press F2, choose REVIEW
      and follow instructions on screen)." )
RULE413
=======
 SUBJECT :: AC-MOTOR-DESIGN-RULES
 DESCRIPTION :: (inv cont pow rating for max power rating = motor o/p
          power )
 IF  :: (MOTOR-MAX-POWER-RATING IS KNOWN AND APPL-TYPE =
      HIGH-PERFORMANCE AND MOTOR-MAX-POWER-RATING =
      MOTOR-OUTPUT-POWER )
 THEN :: (INV-CONT-POWER-RATING = (FIX ((1.1 * MOTOR-MAX-POWER-RATING) /
      0.9 ) ) )
```

APPENDIX B ARL Listing of Expert System for Power Converter System Design

```
Frame :: CONVERTER-DESIGN

IDENTIFIER :: CONVERTER-DESIGN-
TRANSLATION :: (Power electronic converter design)
PARENTS :: (AC-DRIVE-SYSTEM-DESIGN)
GOALS :: (POWER_BJT BJT-IDC-RTG BJT-VOLT-RTG POWER_BJT_PACK RECT_D
      RECT-D-PACK FILTER-CAP-IRMS FILTER-CAP-VOLT SNUB-R SNUB-C
      SNUB_D SIM-OVER )
PREMISE :: ($AND
       (SAME FRAME CONVERTER-DESIGN?))
DISPLAYRESULTS :: YES
PARMGROUP :: CONVERTER-DESIGN-PARMS
RULEGROUPS :: (CONVERTER-DESIGN-RULES)
OFFSPRING :: (SIMULATE OPTIMIZE)
GPROMPTEVER :: "intro1"
CONVERTER-DESIGN-PARMS  :: (BJT-IDC-RTG BJT-MODULE-RCS BJT-MODULE-RCS-M
                BJT-P-COND BJT-P-OFF BJT-P-ON BJT-P-TOTAL
                BJT-RCS BJT-RJC BJT-RJC-M BJT-VOLT-RTG CHOICE
                CREST-FACTOR CUR-VALUE-KNOWN DELTA-PJ
                DELTA-TJ DES-DONE DUMMY-WE DUMMY-WE1
                DUMMY_BJT_TYPE FD-P FD-RCS FD-RJC FD-RJC-M
                FD-TRR FILT-C FILTER-CAP-IRMS FILTER-CAP-VOLT
                FORM-FACTOR FS HS-CSA HS-RSA ID-AV
                IL-FUND-RMS IMAX IS-FUND-RMS IS-RMS K K-OFF
                LEAD-L M OPT-DONE P POWER-BJ POWER-BJT-IC
                POWER-BJT-TY POWER_BJT POWER_BJT_BVCEO
                POWER_BJT_ICMAX POWER_BJT_IDC POWER_BJT_PACK
                QRR R-OPT R-OPT-DONE RECT-D-PACK RECT-D-RCS
                RECT-D-RCS-M RECT-D-RJC RECT-D-RJC-M RECT-DIO
                RECT-P RECT_D RECT_D_IDAV RECT_D_IDPEAK
                RECT_D_IDRMS RECT_D_PIV SIM-DONE SIM-OVER
                SINK-R-BJT SINK-R-FD SINK-R-RECT SNUB-C
                SNUB-C-OPT SNUB-C-VOLT SNUB-CA SNUB-CUR-OS
                SNUB-DIO SNUB-P-LOSS SNUB-R SNUB-R-P SNUB-RES
                SNUB-VOLT-OS SNUB_D SNUB_D_IDMAX SNUB_D_PIV
                STATUS T-AMB T-JUNCT T-RISE TJ-MAX TON-MIN
                T_FALL USER-CHOICE VCE-SAT VD VD-ON-FD
                VD-ON-R VD0 VOLT-OS-DES VOLT-OS-SIM
                VOLT-RISE-TIME WE WHAT-SNUB? X )
CONVERTER-DESIGN-RULES  :: (RULE001 RULE002 RULE003 RULE004 RULE005
                RULE006 RULE007 RULE015 RULE016 RULE024
                RULE025 RULE026 RULE027 RULE028 RULE029
                RULE030 RULE031 RULE032 RULE033 RULE034
                RULE035 RULE036 RULE037 RULE038 RULE039
                RULE040 RULE041 RULE042 RULE064 RULE065
                RULE066 RULE067 RULE068 RULE069 RULE070
                RULE072 RULE073 RULE074 RULE075 RULE076
                RULE077 RULE078 RULE079 RULE080 RULE081
                RULE082 RULE084 RULE085 RULE086 RULE087
                RULE088 RULE089 RULE090 RULE091 RULE092
                RULE093 RULE094 RULE095 RULE096 RULE098
                RULE112 RULE120 RULE124 RULE126 RULE127
                RULE128 RULE135 RULE136 RULE388 RULE404
                RULE405 RULE406 RULE407 RULE408 RULE411 )
```

CONVERTER-DESIGN-PARMS

BJT-IDC-RTG

TRANSLATION :: (Idc rating of the bjt selected from datasheet)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
RANGE :: (1 1000)
CONTAINED-IN :: (RULE040 RULE405 RULE095)

BJT-MODULE-RCS

TRANSLATION :: (single bjt + diode module case to sink thermal resistance)
TYPE :: SINGLEVALUED
USED-BY :: (RULE085)
CONTAINED-IN :: (RULE085 RULE040 RULE405)

BJT-MODULE-RCS-M

TRANSLATION :: (Thermal resistance, case to sink for BJT bridge)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE085)
USED-BY :: (RULE088)
CONTAINED-IN :: (RULE088)

BJT-P-COND

TRANSLATION :: (Conduction loss of BJT bridge)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE069)
USED-BY :: (RULE074)
RANGE :: (0 1000)
CONTAINED-IN :: (RULE074)

BJT-P-OFF

TRANSLATION :: (turn-off loss of the bjt power module)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE073 RULE408)
USED-BY :: (RULE074)
RANGE :: (0 1000)
CONTAINED-IN :: (RULE074)

BJT-P-ON

TRANSLATION :: (turn-on loss of bjt)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE070 RULE407)
USED-BY :: (RULE074)
RANGE :: (0 1000)
CONTAINED-IN :: (RULE074)

BJT-P-TOTAL

TRANSLATION :: (Total power losses in BJT module)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE074)
USED-BY :: (RULE087 RULE092 RULE094)

RANGE :: (1 5000)
LAST-MODIFIED-ON :: "Mar 18,'92"
CONTAINED-IN :: (RULE087 RULE092 RULE094)

BJT-RCS

TRANSLATION :: (BJT thermal resistance, case to sink)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE088)
USED-BY :: (RULE087)
CONTAINED-IN :: (RULE087)

BJT-RJC

TRANSLATION :: (BJT junction to case thermal resistance)
TYPE :: SINGLEVALUED
USED-BY :: (RULE085)
CONTAINED-IN :: (RULE085 RULE040 RULE405)

BJT-RJC-M

TRANSLATION :: (Thermal resistance of junction to case of BJT bridge
module)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE085)
USED-BY :: (RULE087)
CONTAINED-IN :: (RULE087)

BJT-VOLT-RTG

TRANSLATION :: (Rated maximum voltage of selected bjt)
TYPE :: SINGLEVALUED
CONTAINED-IN :: (RULE040 RULE405 RULE095)

CHOICE

TRANSLATION :: (the choice of the user for simulation)
PROMPT :: (:ATTR (CYAN HIGH) "Which of the following (sub)system do you
want to simulate ?")
TYPE :: SINGLEVALUED
EXPECT :: (RECTIFIER INVERTER-MACHINE COMPLETE-SYSTEM HALF-BRIDGE)
UPDATED-BY :: (RULE121)
UPDATED-IN :: (SREFMARK RULE153 RULE154 RULE157 RULE158 RULE150 RULE152
RULE159 RULE156)
USED-BY :: (RULE124 RULE126 RULE127 RULE128 RULE135 RULE136 RULE120)
HELP :: (:ATTR (YELLOW HIGH) "As explained earlier, the parameters
relevant" :LINE "to the system of your choice will be
displayed," :LINE "but complete system is simulated in the"
:LINE "background. Choose complete system for displaying"
:LINE "and passing all the parameters of complete" :LINE "
system." :LINE 2 "When you choose SNUBBER for simulation, one
phase" :LINE "leg of the bridge is simulated, with the" :LINE "
available snubber parameters used. This gives the" :LINE "user
an idea about whether the design is proper" :LINE "or not, but
does not attempt to optimize it.")

CREST-FACTOR

TRANSLATION :: (Crest factor for the supply current wave)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE029 RULE030 RULE031)
CONTAINED-IN :: (RULE032 RULE033)

CUR-VALUE-KNOWN

TRANSLATION :: (flag for known bjt collector current value)
TYPE :: YES/NO
UPDATED-IN :: (RULE098)
USED-BY :: (RULE121)

DELTA-PJ
TRANSLATION :: (excess junction power dissipation during transient overload)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE092)
USED-BY :: (RULE093)
CONTAINED-IN :: (RULE093)

DELTA-TJ
TRANSLATION :: (allowable change in junction temperature during transient overload)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE091)
USED-BY :: (RULE093)
CONTAINED-IN :: (RULE093)

DES-DONE
TRANSLATION :: (design completion flag)
TYPE :: YES/NO
UPDATED-IN :: (RULE095)
ANTECEDENT-IN :: (RULE096)
DEFAULT :: (NO)

DUMMY-WE
TRANSLATION :: (dummy value of supply angular frequency)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (RULE127)
RANGE :: (100 500)

DUMMY-WE1
TRANSLATION :: (another dummy value for supply angular frequency)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (RULE128 RULE136)
RANGE :: (100 500)

DUMMY_BJT_TYPE
TRANSLATION :: (the packing type retrieved from dbase of the power BJT)
TYPE :: SINGLEVALUED
USED-BY :: (RULE077)
CONTAINED-IN :: (RULE077 RULE040 RULE405)

FD-P
TRANSLATION :: (conduction loss of feedback diodes)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE068)
USED-BY :: (RULE089 RULE092 RULE094)
RANGE :: (0 500)
CONTAINED-IN :: (RULE089 RULE092 RULE094)

FD-RCS

======
TRANSLATION :: (feedback diode case to sink thermal resistance)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE088)
USED-BY :: (RULE089)
CONTAINED-IN :: (RULE089)

FD-RJC
======
TRANSLATION :: (feedback diode junction to case thermal resistance)
TYPE :: SINGLEVALUED
USED-BY :: (RULE085)
CONTAINED-IN :: (RULE085 RULE040 RULE405)

FD-RJC-M
========
TRANSLATION :: (per-module junction to case thermal resistance,
feedback diode)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE085)
USED-BY :: (RULE089)
CONTAINED-IN :: (RULE089)

FD-TRR
======
TRANSLATION :: (Reverse recovery time of feedback diode of BJT in
microsecs)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE064)
USED-BY :: (RULE070 RULE407)
RANGE :: (1.e-4 5)
CONTAINED-IN :: (RULE405 RULE070 RULE407)

FILT-C
======
TRANSLATION :: (filter capacitance value)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE095)

FILTER-CAP-IRMS
===============
TRANSLATION :: (RMS current rating for the filter capacitor (Cf) in
Amps.)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE035)
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE388)

FILTER-CAP-VOLT
===============
TRANSLATION :: (Voltage rating of filter capacitor (Cf) in Volts)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE002)
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE388)

FORM-FACTOR
===========
TRANSLATION :: (form factor for the supply current waveform)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE029 RULE030 RULE031)
USED-BY :: (RULE036 RULE037)
CONTAINED-IN :: (RULE036 RULE037)

FS
==
TRANSLATION :: (switching frequency of pwm)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE095)

HS-CSA
======
TRANSLATION :: (thermal capacitance value for heatsink, Watt-sec/C)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE093)
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE388)

HS-RSA
======
TRANSLATION :: (equivalent thermal resistance of the heatsink)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE090)
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE093 RULE388)
CONTAINED-IN :: (RULE093)

ID-AV
=====
TRANSLATION :: (average dc link current)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE025)
USED-BY :: (RULE007 RULE016 RULE035 RULE036 RULE037 RULE027)
CONTAINED-IN :: (RULE007 RULE016 RULE035 RULE036 RULE037 RULE027)

IL-FUND-RMS
==========
TRANSLATION :: (rms load fundamental current)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE034)
USED-BY :: (RULE026 RULE028 RULE035)
CONTAINED-IN :: (RULE026 RULE028 RULE035)

IMAX
====
TRANSLATION :: (maximum collector current value)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE121)
ANTECEDENT-IN :: (RULE098)
USED-BY :: (RULE126 RULE151)
RANGE :: (0 500)

IS-FUND-RMS
==========
TRANSLATION :: (fundamental supply rms current)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE026 RULE028)
USED-BY :: (RULE027)
CONTAINED-IN :: (RULE027)

IS-RMS
======
TRANSLATION :: (rms supply current value- fundamental + harmonics)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE036 RULE037)
USED-BY :: (RULE032 RULE033 RULE035)
CONTAINED-IN :: (RULE032 RULE033 RULE035)

K
=
TRANSLATION :: (multiplicant in diode for. cond. char.)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE095)

K-OFF
=====
TRANSLATION :: (reduction factor for snubber-assisted turn-off bjt
switching losses)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE072)
USED-BY :: (RULE073 RULE408)
RANGE :: (0.001 1)
LAST-MODIFIED-ON :: "Mar 18,'92"
CONTAINED-IN :: (RULE073 RULE408)

LEAD-L
======
TRANSLATION :: (Equivalent lead inductance in BJT turn-on circuit)
TYPE :: SINGLEVALUED
USED-BY :: (RULE070 RULE407)
DEFAULT :: (0.15)
CONTAINED-IN :: (RULE070 RULE407)

M
=
TRANSLATION :: (exponent in diode for. cond. char. equation)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE095)

OPT-DONE
========
TRANSLATION :: (optimization of the snubber circuit done)
TYPE :: YES/NO
UPDATED-BY :: (RULE174)
ANTECEDENT-IN :: (RULE144)
USED-BY :: (RULE120)

P
=
TRANSLATION :: (ratio of power dissipation in fb diode to power bjt)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE094)
USED-BY :: (RULE088)
CONTAINED-IN :: (RULE088)

POWER-BJ
========
TRANSLATION :: (power transistor chosen)
TYPE :: SINGLEVALUED

POWER-BJT-IC
============
TRANSLATION :: (Collector current value at continuous rated power
output of inverter)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE024)
USED-BY :: (RULE068 RULE069 RULE070 RULE073 RULE407 RULE408)
CONTAINED-IN :: (RULE068 RULE069 RULE070 RULE073 RULE407 RULE408)

POWER-BJT-TY
============
TRANSLATION :: (power bjt type)

TYPE :: SINGLEVALUED

POWER_BJT
=========
TRANSLATION :: (The POWEREX Transistor (Q1-Q6) type)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE039 RULE404)
ANTECEDENT-IN :: (RULE040 RULE405 RULE095)
USED-BY :: (RULE388)
CONTAINED-IN :: (RULE040 RULE405)

POWER_BJT_BVCEO
===============
TRANSLATION :: (Maximum voltage rating i.e. BVCEO or breakdown voltage,
for the power switcher)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE002)
USED-BY :: (RULE039 RULE404)
CONTAINED-IN :: (RULE039 RULE404)

POWER_BJT_ICMAX
===============
TRANSLATION :: (Icmax current rating for the power switching device (in
Amps))
TYPE :: SINGLEVALUED

POWER_BJT_IDC
=============
TRANSLATION :: (continuous maximum current rating of power transistor
in Amps)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE024)
USED-BY :: (RULE039 RULE404 RULE064 RULE003 RULE066)
CONTAINED-IN :: (RULE039 RULE404 RULE064 RULE003 RULE066)

POWER_BJT_PACK
==============
TRANSLATION :: (the package of Power BJT selected)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE078 RULE079 RULE080 RULE406)
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE388)

QRR
===
TRANSLATION :: (reverse recovery charge for the feedback diode)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE095)

R-OPT
=====
TRANSLATION :: (optimum turn-on snubber resistance)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE146)
ANTECEDENT-IN :: (RULE144 RULE161)
USED-BY :: (RULE121 RULE120)

R-OPT-DONE
==========
TRANSLATION :: (flag indicating Rs optimization over)
TYPE :: YES/NO
UPDATED-BY :: (RULE146)

RECT-D-PACK
===========

TRANSLATION :: (package of rectifier diode)
TYPE :: SINGLEVALUED
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE388)
CONTAINED-IN :: (RULE081 RULE084)

RECT-D-RCS
==========
TRANSLATION :: (Case to sink thermal resistance, rectifier module, "("
single/dual/quad/6-pack ")")
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (RULE082)
RANGE :: (0.001 5)
CONTAINED-IN :: (RULE081 RULE082)

RECT-D-RCS-M
============
TRANSLATION :: (thermal resistance, case to sink of rectifier bridge
module)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE082)
USED-BY :: (RULE086)
CONTAINED-IN :: (RULE084 RULE086)

RECT-D-RJC
==========
TRANSLATION :: (Junction to case thermal resistance, rectifier module (
single/dual/quad/6-pack))
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (RULE082)
RANGE :: (0.001 5)
CONTAINED-IN :: (RULE081 RULE082)

RECT-D-RJC-M
============
TRANSLATION :: (junction to case thermal resistance of rectifier bridge
module)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE082)
USED-BY :: (RULE086)
CONTAINED-IN :: (RULE084 RULE086)

RECT-DIO
========
TRANSLATION :: (rect dio chosen)
TYPE :: SINGLEVALUED

RECT-P
======
TRANSLATION :: (total conduction loss of rectifier module in Watts)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE067 RULE075)
USED-BY :: (RULE086 RULE092)
CONTAINED-IN :: (RULE086 RULE092)
RANGE :: (0 500)

RECT_D
======
TRANSLATION :: (the POWEREX Bridge rectifier module (D1-D6) type)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE038 RULE076)
ANTECEDENT-IN :: (RULE081 RULE084 RULE095)

USED-BY :: (RULE388)
CONTAINED-IN :: (RULE081 RULE084)

RECT_D_IDAV
==========
TRANSLATION :: (Average current rating for the input bridge rectifier
diodes in Amps)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE007 RULE016)
USED-BY :: (RULE038 RULE065 RULE076 RULE067 RULE075)
CONTAINED-IN :: (RULE038 RULE065 RULE076 RULE067 RULE075)

RECT_D_IDPEAK
==========
TRANSLATION :: (Peak current rating of the input bridge rectifier
diodes in Amps)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE032 RULE033)
USED-BY :: (RULE038 RULE076)
CONTAINED-IN :: (RULE038 RULE076)

RECT_D_IDRMS
==========
TRANSLATION :: (RMS current rating for the input bridge rectifier
diodes in Amps)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE032 RULE033)

RECT_D_PIV
==========
TRANSLATION :: (PIV rating of the input bridge rectifier diodes in
Volts)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE006)
USED-BY :: (RULE038 RULE076)
CONTAINED-IN :: (RULE038 RULE076)

SIM-DONE
========
TRANSLATION :: (simulation of designed system complete)
TYPE :: YES/NO
UPDATED-BY :: (RULE124 RULE126 RULE127 RULE128 RULE135 RULE136)
USED-BY :: (RULE120)

SIM-OVER
========
TRANSLATION :: (The complete process of simulation, optimization and
observation is over)
TYPE :: YES/NO
UPDATED-BY :: (RULE120)
ANTECEDENT-IN :: (RULE015 RULE042)

SINK-R-BJT
==========
TRANSLATION :: (sink thermal resistance due to bjt)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE087)
USED-BY :: (RULE090)
CONTAINED-IN :: (RULE090)

SINK-R-FD
==========
TRANSLATION :: (sink thermal resistance due to fb diode dissipation)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE089)

USED-BY :: (RULE090)
CONTAINED-IN :: (RULE090)

SINK-R-RECT
==========
TRANSLATION :: (Heat sink thermal resistance due to rectifier)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE086)
USED-BY :: (RULE090)
CONTAINED-IN :: (RULE090)

SNUB-C
======
TRANSLATION :: (The value of snubber capacitance (in MicroFarads))
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE003)
UPDATED-IN :: (SREFMARK RULE162 RULE163 RULE168 RULE169 RULE170 RULE171
RULE172 RULE173)
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE004 RULE388 RULE005 RULE066 RULE411)
CONTAINED-IN :: (RULE004 RULE005 RULE066 RULE411)

SNUB-C-OPT
==========
TRANSLATION :: (optimum snub. cap. value)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE174)
ANTECEDENT-IN :: (RULE144)
USED-BY :: (RULE120)

SNUB-C-VOLT
==========
TRANSLATION :: (The voltage rating of snubber capacitor in Volts)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE002)

SNUB-CA
=======
TRANSLATION :: (snubb cap chosen)
TYPE :: YES/NO

SNUB-CUR-OS
==========
TRANSLATION :: (current in BJT at turn-on)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE110)
UPDATED-IN :: (SREFMARK RULE153 RULE154 RULE157 RULE158 RULE150 RULE152
RULE159 RULE156)
ANTECEDENT-IN :: (SREFMARK RULE153 RULE154 RULE157 RULE158 RULE150
RULE152 RULE159 RULE156)
USED-BY :: (RULE135 RULE146)
RANGE :: (0 200)

SNUB-DIO
========
TRANSLATION :: (snubber diode chosen)
TYPE :: SINGLEVALUED

SNUB-P-LOSS
==========
TRANSLATION :: (Power loss in bjt and snubber during turn-off)
TYPE :: SINGLEVALUED
USED-BY :: (RULE124)

SNUB-R
======
TRANSLATION :: (The value of snubber resistance (Rs) in Ohms)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE004)
UPDATED-IN :: (SREFMARK RULE153 RULE154 RULE157 RULE158 RULE150 RULE152
RULE159 RULE156)
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE388)

SNUB-R-P
========
TRANSLATION :: (Power rating of snubber resistor in Watts)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE005 RULE411)

SNUB-RES
========
TRANSLATION :: (snubber resistance chosen)
TYPE :: SINGLEVALUED

SNUB-VOLT-OS
============
TRANSLATION :: (voltage overshoot of BJT due to snubber)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (RULE124)
RANGE :: (0 50)

SNUB_D
======
TRANSLATION :: (thr POWEREX snubber diode (Ds) type)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE041)
ANTECEDENT-IN :: (RULE095)
USED-BY :: (RULE388)

SNUB_D_IDMAX
============
TRANSLATION :: (Maximum current rating of snubber diode in Amps)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE001)
USED-BY :: (RULE041)
CONTAINED-IN :: (RULE041)

SNUB_D_PIV
==========
TRANSLATION :: (PIV rating of snubber diode in Volts)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE002)
USED-BY :: (RULE041)
CONTAINED-IN :: (RULE041)

STATUS
======
TRANSLATION :: (the status of the system according to observation)
TYPE :: SINGLEVALUED
EXPECT :: (O.K. UNHEALTHY)

T-AMB
=====
TRANSLATION :: (the ambient temperature for the thermal design)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (RULE086 RULE087 RULE089)

CONTAINED-IN :: (RULE086 RULE087 RULE089)
DEFAULT :: (40)

T-JUNCT
=======
TRANSLATION :: (Allowable power semiconductor junction temperature)
TYPE :: SINGLEVALUED
USED-BY :: (RULE086 RULE087 RULE089 RULE091)
DEFAULT :: (125)
CONTAINED-IN :: (RULE086 RULE087 RULE089 RULE091)

T-RISE
======
TRANSLATION :: (rated BJT Ic rise time, = 0.6*t (on))
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE064)
USED-BY :: (RULE070 RULE407)
RANGE :: (0 10)
CONTAINED-IN :: (RULE405 RULE070 RULE407)

TJ-MAX
======
TRANSLATION :: (The maximum allowable junction temperature during
transient overloading)
TYPE :: SINGLEVALUED
USED-BY :: (RULE091)
CONTAINED-IN :: (RULE091)
DEFAULT :: (135)

TON-MIN
=======
TRANSLATION :: (Switcher minimum rated turn-on time)
TYPE :: SINGLEVALUED
CONTAINED-IN :: (RULE040)

T_FALL
======
TRANSLATION :: (maximum switcher turn-off crossover time)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE064)
USED-BY :: (RULE072 RULE003 RULE066 RULE073 RULE408)
CONTAINED-IN :: (RULE072 RULE405 RULE003 RULE066 RULE073 RULE408)

USER-CHOICE
==========
TRANSLATION :: (user choice for simulation)
PROMPT :: ("The design of the system is now complete. What action do you
want taken ?")
TYPE :: SINGLEVALUED
EXPECT :: (SIMULATION-AND-WAVEFORM-DISPLAY OPTIMIZATION-OF-SNUBBER
NONE-OF-THE-ABOVE)
ANTECEDENT-IN :: (RULE096 RULE112 RULE161)
USED-BY :: (RULE124 RULE126 RULE127 RULE128 RULE135 RULE136 RULE110
RULE121 RULE120)
HELP :: (:ATTR (YELLOW HIGH) "The SIMNON simulation of the system just
designed" :LINE "can now be done. If you are interested, please"
:LINE "enter the relevant choice here.")

VCE-SAT
=======
TRANSLATION :: (bjt c-e saturation voltage)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-BY :: (RULE064)

```
 USED-BY :: (RULE069)
 LAST-MODIFIED-ON :: "Mar 18,'92"
 RANGE :: (0 5)
 CONTAINED-IN :: (RULE069 RULE405)

VD
==
 TRANSLATION :: (dc link voltage)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-BY :: (RULE121)
 USED-BY :: (RULE151)
 RANGE :: (200 700)

VD-ON-FD
========
 TRANSLATION :: (Forward conduction drop of feedback diode of BJT)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-BY :: (RULE064)
 USED-BY :: (RULE068)
 RANGE :: (0 5)
 LAST-MODIFIED-ON :: "Mar 18,'92"
 CONTAINED-IN :: (RULE068 RULE405)

VD-ON-R
=======
 TRANSLATION :: (Forward conduction drop of rectifier diode in V)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-BY :: (RULE065)
 USED-BY :: (RULE067 RULE075)
 CONTAINED-IN :: (RULE067 RULE075)
 RANGE :: (0.001 10)

VD0
===
 TRANSLATION :: (const. volt. for diode forward cond. char.)
 TYPE :: SINGLEVALUED
 UPDATED-IN :: (RULE095)

VOLT-OS-DES
===========
 TRANSLATION :: (desired voltage overshoot)
 PROMPT :: (What is the desired voltage overshoot for transistor?)
 TYPE :: SINGLEVALUED
 EXPECT :: (10 15 20 DEFAULT)
 ANTECEDENT-IN :: (SREFMARK RULE162 RULE163 RULE168 RULE169)
 USED-BY :: (RULE174)
 HELP :: ("Usually, voltage overshoots of around 15% in the" :LINE "dc
       link are allowable. Enter a number of your" :LINE "choice." )
 DEFAULT :: (15)

VOLT-OS-SIM
===========
 TRANSLATION :: (percentage voltage overshoot)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-IN :: (RULE161 SREFMARK RULE162 RULE163 RULE168 RULE169 RULE170
         RULE171 RULE172 RULE173 )
 ANTECEDENT-IN :: (SREFMARK RULE162 RULE163 RULE168 RULE169 RULE170
           RULE171 RULE172 RULE173 )
 USED-BY :: (RULE174)
 RANGE :: (0 100)
```

```
VOLT-RISE-TIME
==============
 TRANSLATION :: (Rise time of Vce for bjt in microseconds)
 TYPE :: SINGLEVALUED
 UPDATED-BY :: (RULE066)
 USED-BY :: (RULE072)
 CONTAINED-IN :: (RULE072)

WE
==
 TRANSLATION :: (electrical supply frequency)
 TYPE :: SINGLEVALUED
 UPDATED-IN :: (RULE095)

WHAT-SNUB?
==========
 TRANSLATION :: (User choice for turn-on or turn-off snubber for
        simulation)
 PROMPT :: (Which of the snubbers do you want to simulate ?)
 TYPE :: SINGLEVALUED
 EXPECT :: (TURN-ON TURN-OFF ONE-SWITCHING-CYCLE)
 UPDATED-BY :: (RULE110)
 UPDATED-IN :: (RULE161 SREFMARK RULE153 RULE154 RULE157 RULE158 RULE162
        RULE163 RULE168 RULE169 RULE170 RULE171 RULE172 RULE173
        RULE150 RULE152 RULE159 RULE156 )
 USED-BY :: (RULE124 RULE135 RULE136)
 HELP :: ("Here, half-bridge is simulated in the background," :LINE "but
      only the results for your choice of mode." :LINE "(turn-on,
      turn-off or both) are displayed." )

X
=
 TRANSLATION :: (parameter to calculate supply wave form factor and
        crest factor )
 TYPE :: SINGLEVALUED
 UPDATED-BY :: (RULE027)
 USED-BY :: (RULE029 RULE030 RULE031)
 CONTAINED-IN :: (RULE029 RULE030 RULE031)


 ============================
  CONVERTER-DESIGN-RULES
 ============================

RULE001
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Determines snubber diode maximum forward current rating
        )
 NOTE :: "worst case: rated minimum voltage, rated overload, snubber
      discharge current used"
 LAST-MODIFIED-ON :: "Mar 9,'92"
 IF  :: (MOTOR-MAX-POWER-RATING IS KNOWN AND MOTOR-VOLTAGE-RATING IS
      KNOWN )
 THEN :: (SNUB_D_IDMAX = (FIX (((1.5 * 1414.2) * MOTOR-MAX-POWER-RATING)
      / (((1.732 * MOTOR-VOLTAGE-RATING) * 0.85) * 0.9) ) ) )

RULE002
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Determines voltage ratings for power switcher, snubber
        diode, filter capacitor and snubber capacitor )
 NOTE :: (BVCEO = VDC (MAX) VOLTS)
 IF  :: (SUPPLY-INPUT-VOLT IS KNOWN)
 THEN :: (POWER_BJT_BVCEO = (FIX (2.15 * SUPPLY-INPUT-VOLT)) AND
```

```
      SNUB_D_PIV = (FIX (2.15 * SUPPLY-INPUT-VOLT)) AND
      FILTER-CAP-VOLT = (FIX (2.15 * SUPPLY-INPUT-VOLT)) AND
      SNUB-C-VOLT = (FIX (2.15 * SUPPLY-INPUT-VOLT)) )

RULE003
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates snubber capacitance value in microFarads)
 NOTE :: "C = 0.3952*Icmax * tcmax / Vcemax"
 LAST-MODIFIED-ON :: "Mar 9,'92"
 IF  :: (POWER_BJT_IDC IS KNOWN AND T_FALL IS KNOWN AND
      SUPPLY-INPUT-VOLT IS KNOWN )
 THEN :: (SNUB-C = (FIX (((395.2 * POWER_BJT_IDC) * T_FALL) / (3.11 *
      SUPPLY-INPUT-VOLT ) ) / 1000 ) )

RULE004
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates snubber resistance)
 NOTE :: "Snubber discharge time constant = min. on-time / 40"
 IF  :: (SNUB-C IS KNOWN)
 THEN :: (SNUB-R = (FIX (0.6 * (FIX (66.66666667 / SNUB-C) / 100))))

RULE005
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Determines snubber resistor power rating)
 NOTE :: "C in uF, fs = 4 kHz."
 LAST-MODIFIED-ON :: "Mar. 9,'92"
 IF  :: (SUPPLY-INPUT-VOLT IS KNOWN AND SNUB-C IS KNOWN AND
      TYPE-OF-SWITCH = BJT )
 THEN :: (SNUB-R-P = (FIX ((4 * SUPPLY-INPUT-VOLT ^ 2) * SNUB-C) / 1000)
      )

RULE006
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates PIV rating for the rectifier diodes)
 NOTE :: "PIV = 1.1 * sqrt(2) * Vin"
 IF  :: (SUPPLY-INPUT-VOLT IS KNOWN)
 THEN :: (RECT_D_PIV = (FIX (1.555634919 * SUPPLY-INPUT-VOLT)))

RULE007
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Finds 3-ph rectifier diode Average current rating)
 NOTE :: "rectifier average current rating derated because of diode
      conduction interval angle < 180"
 IF  :: (ID-AV IS KNOWN AND NO-OF-PHASES = THREE_PHASE)
 THEN :: (RECT_D_IDAV = (FIX (ID-AV / 3) + 1))

RULE015
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Prints message at conclusion for three phase inverter)
 IF  :: (SIM-OVER AND NO-OF-PHASES = THREE_PHASE)
 THEN :: (PRINT NAME "-- The results for design of the PWM inverter
      Induction Motor drive follow. All the power devices selected
      are from POWEREX catalog. :" " " AND PICTURE "config" )

RULE016
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Finds single phase rectifier diode average current
```

```
        rating )
 IF   :: (ID-AV IS KNOWN AND NO-OF-PHASES = SINGLE_PHASE)
 THEN :: (RECT_D_IDAV = (FIX ((ID-AV / 2.) + 1)))

RULE024
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates power transistor Idc rating)
 NOTE :: "Both continuous and transient overloading conditions considered
      here. Load ripple and snubber discharge also accounted for."
 LAST-MODIFIED-ON :: "Mar 24,'92"
 IF   :: (MOTOR-MAX-POWER-RATING IS KNOWN AND MOTOR-VOLTAGE-RATING IS
      KNOWN )
 THEN :: (POWER_BJT_IDC = (((1.8 * 1414.2) * MOTOR-MAX-POWER-RATING) / (
      ((1.732 * MOTOR-VOLTAGE-RATING) * 0.85 ) * 0.9 ) ) AND
      POWER-BJT-IC = (((1.2 * 1414.2) * MOTOR-MAX-POWER-RATING) / ((
      (1.732 * MOTOR-VOLTAGE-RATING ) * 0.85 ) * 0.9 ) ) )

RULE025
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates maximum average dc link current)
 NOTE :: "Motor eff = 0.85, inv. eff = 0.95, min. input voltage = 0.85 pu"
 LAST-MODIFIED-ON :: "Mar 9,1992"
 IF   :: (MOTOR-MAX-POWER-RATING IS KNOWN AND SUPPLY-INPUT-VOLT IS KNOWN
      )
 THEN :: (ID-AV = (MOTOR-MAX-POWER-RATING / ((((0.85 * 0.85) * 0.95) *
      1.4142135 ) * SUPPLY-INPUT-VOLT ) ) )

RULE026
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates supply fundamental rms current for single
        phase case )
 IF   :: (IL-FUND-RMS IS KNOWN AND NO-OF-PHASES = SINGLE_PHASE)
 THEN :: (IS-FUND-RMS = (1.4841 * IL-FUND-RMS))

RULE027
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates x value when user cannot supply source
        information )
 LAST-MODIFIED-ON :: "Feb 28,1992"
 IF   :: (ID-AV IS KNOWN AND IS-FUND-RMS IS KNOWN)
 THEN :: (X = ((0.132626 * ID-AV) / IS-FUND-RMS))

RULE028
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates rms fundamental supply current for three
        phase case )
 IF   :: (IL-FUND-RMS IS KNOWN AND NO-OF-PHASES = THREE_PHASE)
 THEN :: (IS-FUND-RMS = (1.0625 * IL-FUND-RMS))

RULE029
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates ff and cf for 0<x<=0.02)
 IF   :: (0 < X AND X <= 0.02)
 THEN :: (FORM-FACTOR = ((-30 * X) + 2.7) AND CREST-FACTOR = ((-45 * X)
      + 3.65 ) )

RULE030
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
```

```
 DESCRIPTION :: (calculates ff and cf for 0.02<x<=0.04)
 IF  :: (0.02 < X AND X <= 0.04)
 THEN :: (FORM-FACTOR = ((-12.5 * X) + 2.5) AND CREST-FACTOR = ((-17.5 *
      X ) + 3.1 ) )

RULE031
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates ff and cf for x > 0.04)
 IF  :: (0.04 < X)
 THEN :: (FORM-FACTOR = ((-1.786 * X) + 1.92) AND CREST-FACTOR = ((-1.43
      * X ) + 2.457 ) )

RULE032
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates rms and peak current ratings for 1 ph
          rectifier diodes )
 IF  :: (IS-RMS IS KNOWN AND NO-OF-PHASES = SINGLE_PHASE)
 THEN :: (RECT_D_IDRMS = (FIX (IS-RMS * 0.7071067812)) AND RECT_D_IDPEAK
      = (FIX (IS-RMS * CREST-FACTOR)) )

RULE033
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates rms and peak current ratings for 3 ph
          rectifier diodes )
 IF  :: (IS-RMS IS KNOWN AND NO-OF-PHASES = THREE_PHASE)
 THEN :: (RECT_D_IDRMS = (FIX (IS-RMS * 0.8164965809)) AND RECT_D_IDPEAK
      = (FIX (CREST-FACTOR * IS-RMS)) )

RULE034
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (finds fundamental load rms current)
 LAST-MODIFIED-ON :: "Mar 9,'92"
 IF  :: (MOTOR-MAX-POWER-RATING IS KNOWN AND MOTOR-VOLTAGE-RATING IS
      KNOWN )
 THEN :: (IL-FUND-RMS = ((1100 * MOTOR-MAX-POWER-RATING) / ((1.732 *
      0.85 ) * MOTOR-VOLTAGE-RATING ) ) )

RULE035
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates filter cap rms current rating)
 IF  :: (IS-RMS IS KNOWN AND ID-AV IS KNOWN AND IL-FUND-RMS IS KNOWN)
 THEN :: (FILTER-CAP-IRMS = (FIX (FILT_CAP_RMS IS-RMS ID-AV IL-FUND-RMS)
      ) )

RULE036
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates supply rms current for single phase case)
 IF  :: (ID-AV IS KNOWN AND FORM-FACTOR IS KNOWN AND NO-OF-PHASES =
      SINGLE_PHASE )
 THEN :: (IS-RMS = (FORM-FACTOR * ID-AV))

RULE037
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates rms supply current for three phase case)
 IF  :: (ID-AV IS KNOWN AND FORM-FACTOR IS KNOWN AND NO-OF-PHASES =
      THREE_PHASE )
 THEN :: (IS-RMS = ((FORM-FACTOR * ID-AV) / 0.7071067812))
```

```
RULE038
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Pops out the diode type for the calculated ratings for
        three phase )
 IF  :: (RECT_D_IDAV IS KNOWN AND RECT_D_IDPEAK IS KNOWN AND RECT_D_PIV
      IS KNOWN AND NO-OF-PHASES = THREE_PHASE )
 THEN :: (RECT_D = (DBASE-RETRIEVE "3ph_bri" (QUAL RECT_D_IAV >=
      RECT_D_IDAV AND RECT_D_IPK >= RECT_D_IDPEAK AND RECT_D_PIV >=
      RECT_D_PIV ) TYPE ) )

RULE039
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Selects BJT from database using calculated ratings)
 IF  :: (POWER_BJT_BVCEO IS KNOWN AND POWER_BJT_IDC IS KNOWN AND
      TYPE-OF-SWITCH = BJT )
 THEN :: (POWER_BJT = (DBASE-RETRIEVE "bjt_rat" (QUAL IC_DC >=
      POWER_BJT_IDC AND V_CEV_SUS >= POWER_BJT_BVCEO ) TYPE ) )

RULE040
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Retrieves switching and thermal ratings of selected BJT
        from dBASE )
 IF  :: (POWER_BJT IS KNOWN AND TYPE-OF-SWITCH = BJT)
 THEN :: (DBASE-BUFFER-APPEND AND DBASE-MRETRIEVE TON-MIN "bjt_rat" (
      QUAL TYPE = POWER_BJT ) T_ON DUMMY_BJT_TYPE "bjt_rat" (QUAL
      TYPE = POWER_BJT ) PACKAGE BJT-RJC "bjt_rat" (QUAL TYPE =
      POWER_BJT ) R_JC_BJT BJT-MODULE-RCS "bjt_rat" (QUAL TYPE =
      POWER_BJT ) R_CS FD-RJC "bjt_rat" (QUAL TYPE = POWER_BJT)
      R_JC_D BJT-IDC-RTG "bjt_rat" (QUAL TYPE = POWER_BJT) IC_DC
      BJT-VOLT-RTG "bjt_rat" (QUAL TYPE = POWER_BJT) V_CEV_SUS AND
      DBASE-EXECUTE )

RULE041
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Selects the snubber diode based on the ratings)
 IF  :: (SNUB_D_IDMAX IS KNOWN AND SNUB_D_PIV IS KNOWN)
 THEN :: (SNUB_D = (DBASE-RETRIEVE "diodes" (QUAL RECT_D_IPK >=
      SNUB_D_IDMAX AND RECT_D_PIV >= SNUB_D_PIV ) TYPE ) )

RULE042
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Prints message and conclusion picture for single phase
        supply. )
 IF  :: (SIM-OVER AND NO-OF-PHASES = SINGLE_PHASE)
 THEN :: (PRINT NAME "-- The results for design of the PWM inverter
      Induction Motor drive follow:" " " AND PICTURE "1prinv" )

RULE064
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates the t-fall, t-rise, vce (sat) of bjt and
        trr, vd (on) of feedback diode )
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (POWER_BJT_IDC IS KNOWN AND TYPE-OF-SWITCH = BJT)
 THEN :: (T_FALL = (((((0.17302852 + (0.01041419 * POWER_BJT_IDC)) + (
      0.48181426 / POWER_BJT_IDC ) ) + (8.6437379e-4 * POWER_BJT_IDC
      ^ 2 ) ) + (0.33552302 / POWER_BJT_IDC ^ 2) ) - (1.0067306e-5 *
      POWER_BJT_IDC ^ 3 ) ) AND T-RISE = (0.6 * ((((0.65861949 + (
```

```
      0.01803054 * POWER_BJT_IDC ) ) + (8.3457251e-5 * POWER_BJT_IDC
      ^ 2 ) ) - (0.41562869 / POWER_BJT_IDC) ) + (0.38884755 /
      POWER_BJT_IDC ^ 2 ) ) ) AND FD-TRR = (TRR-IC POWER_BJT_IDC)
      AND VCE-SAT = 1 AND VD-ON-FD = (VD-ID-FD POWER_BJT_IDC) )


RULE065
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates rectifier diode forward conduction drop)
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (RECT_D_IDAV IS KNOWN)
 THEN :: (VD-ON-R = (VD-ID RECT_D_IDAV))

RULE066
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates the trv for the snubber assisted turn-off of
          BJT. )
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (SNUB-C IS KNOWN AND POWER_BJT_IDC IS KNOWN AND T_FALL IS KNOWN
      AND SUPPLY-INPUT-VOLT IS KNOWN )
 THEN :: (VOLT-RISE-TIME = (TRV_CALC SNUB-C T_FALL SUPPLY-INPUT-VOLT
      POWER_BJT_IDC ) )

RULE067
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates rectifier conduction loss for 3-phase bridge
          )
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (RECT_D_IDAV IS KNOWN AND VD-ON-R IS KNOWN AND NO-OF-PHASES =
      THREE_PHASE )
 THEN :: (RECT-P = ((100 * ((6 * RECT_D_IDAV) * VD-ON-R)) / 150))

RULE068
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates total feedback diode conduction losses, W)
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (VD-ON-FD IS KNOWN AND POWER-BJT-IC IS KNOWN)
 THEN :: (FD-P = ((0.9 * VD-ON-FD) * POWER-BJT-IC))

RULE069
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates the conduction loss per BJT switch)
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (VCE-SAT IS KNOWN AND POWER-BJT-IC IS KNOWN)
 THEN :: (BJT-P-COND = ((0.85 * VCE-SAT) * POWER-BJT-IC))

RULE070
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates power bjt turn-on switching loss per switch)
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (POWER-BJT-IC IS KNOWN AND SUPPLY-INPUT-VOLT IS KNOWN AND
      LEAD-L IS KNOWN AND T-RISE IS KNOWN AND FD-TRR IS KNOWN AND
      TYPE-OF-SWITCH = BJT )
 THEN :: (BJT-P-ON = (0.005 * ((((0.9 * (0.25 / 3.141592654)) *
      POWER-BJT-IC ) * ((1.414213562 * SUPPLY-INPUT-VOLT) - ((LEAD-L
      * POWER-BJT-IC ) / T-RISE ) ) ) * ((0.3325 * T-RISE) + (2 *
      FD-TRR ) ) ) ) )
```

```
RULE072
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates k (off) for snubber assisted turn-off losses
        )
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (VOLT-RISE-TIME IS KNOWN AND T_FALL IS KNOWN)
 THEN :: (K-OFF = ((1 - ((4 / 3) * (VOLT-RISE-TIME / T_FALL))) + (0.5 *
      (VOLT-RISE-TIME / T_FALL ) ^ 2 ) ) )

RULE073
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates turn-off loss per BJT switch)
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (K-OFF IS KNOWN AND SUPPLY-INPUT-VOLT IS KNOWN AND POWER-BJT-IC
      IS KNOWN AND T_FALL IS KNOWN AND TYPE-OF-SWITCH = BJT )
 THEN :: (BJT-P-OFF = (0.005 * ((((((0.25 / 3.141592654) * K-OFF) * 0.5)
      * (1.414213562 * SUPPLY-INPUT-VOLT) ) * (1.6 * POWER-BJT-IC) )
      * T_FALL ) ) )

RULE074
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates total BJT bridge power dissipation for heat
        sink design )
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (BJT-P-COND IS KNOWN AND BJT-P-OFF IS KNOWN AND BJT-P-ON IS
      KNOWN )
 THEN :: (BJT-P-TOTAL = (6 * ((BJT-P-COND + BJT-P-ON) + BJT-P-OFF)))

RULE075
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Calculates rectifier conduction losses for single phase
        supply )
 LAST-MODIFIED-ON :: "Mar 18,'92"
 IF  :: (RECT_D_IDAV IS KNOWN AND VD-ON-R IS KNOWN AND NO-OF-PHASES =
      SINGLE_PHASE )
 THEN :: (RECT-P = ((100 * ((4 * RECT_D_IDAV) * VD-ON-R)) / 150))

RULE076
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (pops out the diode type from diodes database, single
        phase case )
 LAST-MODIFIED-ON :: "Mar 23,'92"
 IF  :: (RECT_D_IDAV IS KNOWN AND RECT_D_IDPEAK IS KNOWN AND RECT_D_PIV
      IS KNOWN AND NO-OF-PHASES = SINGLE_PHASE )
 THEN :: (RECT_D = (DBASE-RETRIEVE "diodes" (QUAL RECT_D_IAV >=
      RECT_D_IDAV AND RECT_D_IPK >= RECT_D_IDPEAK AND RECT_D_PIV >=
      RECT_D_PIV ) TYPE ) )

RULE077
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Sets value of power-bjt-type variable)
 LAST-MODIFIED-ON :: "Mar 24,'92"
 IF  :: (DUMMY_BJT_TYPE IS KNOWN)
 THEN :: (SET-VALUE POWER-BJT-TYPE (VALUE DUMMY_BJT_TYPE))

RULE078
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets the value of power bjt packing = transistor bridge
```

```
        )
 IF   :: ((VALUE-OF POWER-BJT-TYPE) EQUAL (VALUE-OF transistor_bridge))
 THEN :: (POWER_BJT_PACK = six transistor bridge module)

RULE079
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets power bjt module packing = dual darlington)
 IF   :: ((VALUE-OF POWER-BJT-TYPE) EQUAL (VALUE-OF dual_Darlington))
 THEN :: (POWER_BJT_PACK = dual darlington transistor module)

RULE080
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets power bjt package = single darlington module)
 IF   :: ((VALUE-OF POWER-BJT-TYPE) EQUAL (VALUE-OF single_darlington))
 THEN :: (POWER_BJT_PACK = single darlington transistor module)

RULE081
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (retrieves rectifier diode thermal and packing
        parameters )
 IF   :: (RECT_D IS KNOWN AND NO-OF-PHASES = SINGLE_PHASE)
 THEN :: (DBASE-BUFFER-APPEND AND DBASE-MRETRIEVE RECT-D-PACK "diodes" (
      QUAL TYPE = RECT_D ) PACKAGE RECT-D-RJC "diodes" (QUAL TYPE =
      RECT_D ) RECT_D_RJC RECT-D-RCS "diodes" (QUAL TYPE = RECT_D)
      RECT_D_RCS AND DBASE-EXECUTE )

RULE082
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates per-module single phase rect thermal
        resistances )
 IF   :: (NO-OF-PHASES = SINGLE_PHASE AND RECT-D-RJC IS KNOWN AND
      RECT-D-RCS IS KNOWN )
 THEN :: (RECT-D-RCS-M = ((VALUE RECT-D-RCS) / 4) AND RECT-D-RJC-M = ((
      VALUE RECT-D-RJC ) / 4 ) )

RULE084
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (retrieves thermal and packing info of three phase
        rectifier dbase )
 IF   :: (RECT_D IS KNOWN AND NO-OF-PHASES = THREE_PHASE)
 THEN :: (DBASE-BUFFER-APPEND AND DBASE-MRETRIEVE RECT-D-RJC-M "3ph_bri"
      (QUAL TYPE = RECT_D ) RECT_D_RJC RECT-D-RCS-M "3ph_bri" (QUAL
      TYPE = RECT_D ) RECT_D_RCS RECT-D-PACK "3ph_bri" (QUAL TYPE =
      RECT_D ) PACKAGE AND DBASE-EXECUTE )

RULE085
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (finds per-module thermal resistances for bjt and fb
        diode bridge )
 IF   :: (BJT-RJC IS KNOWN AND BJT-MODULE-RCS IS KNOWN AND FD-RJC IS
      KNOWN )
 THEN :: (FD-RJC-M = ((VALUE FD-RJC) / 6) AND BJT-RJC-M = ((VALUE
      BJT-RJC ) / 6 ) AND BJT-MODULE-RCS-M = ((VALUE BJT-MODULE-RCS)
      / 6 ) )

RULE086
=======
```

```
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates r-sa-r)
 IF  :: (T-JUNCT IS KNOWN AND T-AMB IS KNOWN AND RECT-P IS KNOWN AND
       RECT-D-RCS-M IS KNOWN AND RECT-D-RJC-M IS KNOWN )
 THEN :: (SINK-R-RECT = (((T-JUNCT - T-AMB) / RECT-P) - (RECT-D-RCS-M +
       RECT-D-RJC-M ) ) )

RULE087
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates r-sa-bjt)
 IF  :: (T-JUNCT IS KNOWN AND T-AMB IS KNOWN AND BJT-P-TOTAL IS KNOWN
       AND BJT-RJC-M IS KNOWN AND BJT-RCS IS KNOWN )
 THEN :: (SINK-R-BJT = (((T-JUNCT - T-AMB) / BJT-P-TOTAL) - (BJT-RCS +
       BJT-RJC-M ) ) )

RULE088
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (finds r-cs-t and r-cs-d)
 IF  :: (BJT-MODULE-RCS-M IS KNOWN AND P IS KNOWN)
 THEN :: (BJT-RCS = (BJT-MODULE-RCS-M * (1 + P)) AND FD-RCS = ((1 + P) /
       P ) )

RULE089
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (finds r-sa-d)
 IF  :: (T-JUNCT IS KNOWN AND T-AMB IS KNOWN AND FD-P IS KNOWN AND
       FD-RJC-M IS KNOWN AND FD-RCS IS KNOWN )
 THEN :: (SINK-R-FD = (((T-JUNCT - T-AMB) / FD-P) - (FD-RCS + FD-RJC-M))
       )

RULE090
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates heat sink thermal resistance on ss basis)
 IF  :: (SINK-R-FD IS KNOWN AND SINK-R-BJT IS KNOWN AND SINK-R-RECT IS
       KNOWN )
 THEN :: (HS-RSA = (FIX (1000 * (1 / (((1 / SINK-R-FD) + (1 / SINK-R-BJT
       ) ) + (1 / SINK-R-RECT) ) ) ) / 1000 ) )

RULE091
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates delta-tj)
 IF  :: (T-JUNCT IS KNOWN AND TJ-MAX IS KNOWN)
 THEN :: (DELTA-TJ = (TJ-MAX - T-JUNCT))

RULE092
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (finds transient overload power in terms of rated power)
 IF  :: (BJT-P-TOTAL IS KNOWN AND FD-P IS KNOWN AND RECT-P IS KNOWN)
 THEN :: (DELTA-PJ = (0.25 * ((BJT-P-TOTAL + FD-P) + RECT-P)))

RULE093
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates heat sink transient thermal capacitance,
           watt-sec/K )
 IF  :: (DELTA-TJ IS KNOWN AND DELTA-PJ IS KNOWN AND HS-RSA IS KNOWN)
 THEN :: (HS-CSA = (100 * FIX ((CSA-CALC DELTA-PJ DELTA-TJ HS-RSA) / 100
       ) ) )
```

```
RULE094
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (Finds value for P, the ratio of power dissipation in FD
        and BJT )
 IF  :: (BJT-P-TOTAL IS KNOWN AND FD-P IS KNOWN)
 THEN :: (P = (FD-P / BJT-P-TOTAL))

RULE095
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (sets design done flag)
 IF  :: (POWER_BJT IS KNOWN AND POWER_BJT_PACK IS KNOWN AND SNUB-R IS
      KNOWN AND SNUB_D IS KNOWN AND SNUB-C IS KNOWN AND RECT_D IS
      KNOWN AND RECT-D-PACK IS KNOWN AND FILTER-CAP-IRMS IS KNOWN
      AND FILTER-CAP-VOLT IS KNOWN AND HS-RSA IS KNOWN AND HS-CSA IS
      KNOWN )
 THEN :: (WE = 377 AND FILT-C = 0.0048 AND VD0 = 0.80767 AND K = 0.0454
      AND M = 0.5981 AND FS = 5000 AND QRR = 4.e-6 AND PRINT "The
      values of critical ratings of selected transistor are," AND
      SHOW BJT-IDC-RTG "The continuous current rating of the
      transistor is" IFUNKNOWN AND SHOW BJT-VOLT-RTG "The maximum
      voltage rating of the transistor is" IFUNKNOWN AND DES-DONE )

RULE096
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (invokes simulation subframe)
 IF  :: (DES-DONE AND USER-CHOICE = SIMULATION-AND-WAVEFORM-DISPLAY)
 THEN :: (DONTCONSIDERFRAME OPTIMIZE AND DONTCONSIDERFRAME OBSERVE AND
      CONSIDERFRAME SIMULATE )

RULE098
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (sets flag for known bjt collector current value)
 IF  :: (IMAX IS KNOWN)
 THEN :: (CUR-VALUE-KNOWN)

RULE112
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Invokes optimization of snub. cap. upon request)
 IF  :: (USER-CHOICE = OPTIMIZATION-OF-SNUBBER)
 THEN :: (PRINT NAME "-- The snubber capacitance optimization is now
      activated. Please wait for the simulation results .. --" AND
      DONTCONSIDERFRAME SIMULATE AND CONSIDERFRAME OPTIMIZE )

RULE120
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets the conclusion flag)
 IF  :: (USER-CHOICE = NONE-OF-THE-ABOVE OR (USER-CHOICE =
      SIMULATION-AND-WAVEFORM-DISPLAY AND (CHOICE = RECTIFIER OR
      CHOICE = INVERTER-MACHINE OR CHOICE = HALF-BRIDGE OR CHOICE =
      COMPLETE-SYSTEM ) AND SIM-DONE ) OR (USER-CHOICE =
      OPTIMIZATION-OF-SNUBBER AND R-OPT IS KNOWN AND SNUB-C-OPT IS
      KNOWN AND OPT-DONE ) )
 THEN :: (SIM-OVER)

RULE124
```

```
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets sim-done flag for turn-off)
 IF  :: (USER-CHOICE = SIMULATION-AND-WAVEFORM-DISPLAY AND CHOICE =
       HALF-BRIDGE AND WHAT-SNUB? = TURN-OFF AND SNUB-VOLT-OS IS
       KNOWN AND SNUB-P-LOSS IS KNOWN )
 THEN :: (SIM-DONE)

RULE126
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets sim-done flag for inverter-machine)
 IF  :: (USER-CHOICE = SIMULATION-AND-WAVEFORM-DISPLAY AND CHOICE =
       INVERTER-MACHINE AND IMAX IS KNOWN )
 THEN :: (SIM-DONE)

RULE127
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets sim-done when rectifier simulation is done)
 IF  :: (USER-CHOICE = SIMULATION-AND-WAVEFORM-DISPLAY AND CHOICE =
       RECTIFIER AND DUMMY-WE IS KNOWN )
 THEN :: (SIM-DONE)

RULE128
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets sim-done when complete system simulated)
 IF  :: (USER-CHOICE = SIMULATION-AND-WAVEFORM-DISPLAY AND CHOICE =
       COMPLETE-SYSTEM AND DUMMY-WE1 IS KNOWN )
 THEN :: (SIM-DONE)

RULE135
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets sim-done when simulation of turn-on snubber done)
 IF  :: (USER-CHOICE = SIMULATION-AND-WAVEFORM-DISPLAY AND CHOICE =
       HALF-BRIDGE AND WHAT-SNUB? = TURN-ON AND SNUB-CUR-OS IS KNOWN
       )
 THEN :: (SIM-DONE)

RULE136
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (simulation of one switching cycle done)
 IF  :: (USER-CHOICE = SIMULATION-AND-WAVEFORM-DISPLAY AND CHOICE =
       HALF-BRIDGE AND WHAT-SNUB? = ONE-SWITCHING-CYCLE AND DUMMY-WE1
       IS KNOWN )
 THEN :: (SIM-DONE)

RULE388
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets conv design done flag, one of the goals of the
           root frame )
 IF  :: (POWER_BJT IS KNOWN AND POWER_BJT_PACK IS KNOWN AND SNUB-R IS
       KNOWN AND SNUB_D IS KNOWN AND SNUB-C IS KNOWN AND RECT_D IS
       KNOWN AND RECT-D-PACK IS KNOWN AND FILTER-CAP-IRMS IS KNOWN
       AND FILTER-CAP-VOLT IS KNOWN AND HS-RSA IS KNOWN AND HS-CSA IS
       KNOWN )
 THEN :: (CONV-DES-DONE)

RULE404
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
```

```
 DESCRIPTION :: (selects suitable igbt from db)
 IF  :: (POWER_BJT_BVCEO IS KNOWN AND POWER_BJT_IDC IS KNOWN AND
      TYPE-OF-SWITCH = IGBT )
 THEN :: (POWER_BJT = (DBASE-RETRIEVE "igbt_rtg" (QUAL IC_DC >=
      POWER_BJT_IDC AND V_CEV_SUS >= POWER_BJT_BVCEO ) TYPE ) )

RULE405
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (retrieves from db thermal and switching ratings of
        selected igbt )
 IF  :: (POWER_BJT IS KNOWN AND TYPE-OF-SWITCH = IGBT)
 THEN :: (DBASE-BUFFER-APPEND AND DBASE-MRETRIEVE T-RISE "igbt_rtg" (
      QUAL TYPE = POWER_BJT ) T_RISE DUMMY_BJT_TYPE "igbt_rtg" (QUAL
      TYPE = POWER_BJT ) PACKAGE BJT-RJC "igbt_rtg" (QUAL TYPE =
      POWER_BJT ) R_JC_BJT BJT-MODULE-RCS "igbt_rtg" (QUAL TYPE =
      POWER_BJT ) R_CS FD-RJC "igbt_rtg" (QUAL TYPE = POWER_BJT)
      R_JC_D BJT-IDC-RTG "igbt_rtg" (QUAL TYPE = POWER_BJT) IC_DC
      BJT-VOLT-RTG "igbt_rtg" (QUAL TYPE = POWER_BJT) V_CEV_SUS
      T_FALL "igbt_rtg" (QUAL TYPE = POWER_BJT) T_FALL VCE-SAT "
      igbt_rtg" (QUAL TYPE = POWER_BJT) V_CE_SAT FD-TRR "igbt_rtg" (
      QUAL TYPE = POWER_BJT ) TRR_D VD-ON-FD "igbt_rtg" (QUAL TYPE =
      POWER_BJT ) VD_ON_FD AND DBASE-EXECUTE )

RULE406
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (sets power switch package to be dual igbt)
 IF  :: ((VALUE-OF POWER-BJT-TYPE) EQUAL (VALUE-OF dual_igbt))
 THEN :: (POWER_BJT_PACK = dual IGBT module)

RULE407
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates igbt turn-on switching losses)
 IF  :: (POWER-BJT-IC IS KNOWN AND SUPPLY-INPUT-VOLT IS KNOWN AND
      LEAD-L IS KNOWN AND T-RISE IS KNOWN AND FD-TRR IS KNOWN AND
      TYPE-OF-SWITCH = IGBT )
 THEN :: (BJT-P-ON = (0.01 * ((((0.9 * (0.25 / 3.141592654)) *
      POWER-BJT-IC ) * ((1.414213562 * SUPPLY-INPUT-VOLT) - ((LEAD-L
      * POWER-BJT-IC ) / T-RISE ) ) ) * ((0.3325 * T-RISE) + (2 *
      FD-TRR ) ) ) ) )

RULE408
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (calculates turn-off losses/igbt)
 IF  :: (K-OFF IS KNOWN AND SUPPLY-INPUT-VOLT IS KNOWN AND POWER-BJT-IC
      IS KNOWN AND T_FALL IS KNOWN AND TYPE-OF-SWITCH = IGBT )
 THEN :: (BJT-P-OFF = (0.01 * ((((((0.25 / 3.141592654) * K-OFF) * 0.5)
      * (1.414213562 * SUPPLY-INPUT-VOLT) ) * (1.6 * POWER-BJT-IC) )
      * T_FALL ) ) )

RULE411
=======
 SUBJECT :: CONVERTER-DESIGN-RULES
 DESCRIPTION :: (snubber resistor power rating for igbt)
 IF  :: (SUPPLY-INPUT-VOLT IS KNOWN AND SNUB-C IS KNOWN AND
      TYPE-OF-SWITCH = IGBT )
 THEN :: (SNUB-R-P = (FIX ((10 * SUPPLY-INPUT-VOLT ^ 2) * SNUB-C) / 1000
      ) )
```

APPENDIX C Software Listing for Power Converter System Simulation andOptimization

APPENDIX C.1 Expert System for Power Converter Simulation and Optimization

```
Frame :: SIMULATE

IDENTIFIER :: "SIMULATE-"
TRANSLATION :: (performs simulation of the designed system)
PARENTS :: (CONVERTER-DESIGN AC-DRIVE-SYSTEM-DESIGN)
GOALS :: (SIM-DONE)
PREMISE :: ($AND
        (NOTSAME FRAME USER-CHOICE NONE-OF-THE-ABOVE))
PARMGROUP :: SIMULATE-PARMS
RULEGROUPS :: (SIMULATE-RULES)
GPROMPTEVER :: "simint"
SIMULATE-PARMS  :: (BJT-IC-SIM BJT-VD-SIM)
SIMULATE-RULES  :: (RULE101 RULE102 RULE103 RULE104 RULE105 RULE134)

  SIMULATE-PARMS

BJT-IC-SIM

 TRANSLATION :: (value of the power transistor worst-case collector
          current from simulation )
 TYPE :: SINGLEVALUED
 EXPECT :: NUMBER
 RANGE :: (0 400)

BJT-VD-SIM

 TRANSLATION :: (value of the power transistor C-E voltage from
          simulation )
 TYPE :: SINGLEVALUED
 EXPECT :: NUMBER
 RANGE :: (0 1500)

  SIMULATE-RULES

RULE101

 SUBJECT :: SIMULATE-RULES
 DESCRIPTION :: (Simulates turn-on snubber upon user request.)
 IF  :: (CHOICE = HALF-BRIDGE AND WHAT-SNUB? = TURN-ON)
 THEN :: (WRITE-DOS-FILE "param.dat" INPUT-VOLT POWER-BJT-IC T-RISE
      SNUB-R SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
      WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "expdat.t"
      SNUB-CUR-OS )

RULE102

 SUBJECT :: SIMULATE-RULES
 DESCRIPTION :: (Invokes simulation of turn-off snubber)
 IF  :: (CHOICE = HALF-BRIDGE AND WHAT-SNUB? = TURN-OFF)
 THEN :: (WRITE-DOS-FILE "param.dat" INPUT-VOLT POWER-BJT-IC T-RISE
      SNUB-R SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
      WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "expdat.t"
```

```
      SNUB-VOLT-OS SNUB-P-LOSS )

RULE103
=======
 SUBJECT :: SIMULATE-RULES
 DESCRIPTION :: (Invokes simulation of rectifier)
 IF  :: (USER-CHOICE = SIMULATION-AND-WAVEFORM-DISPLAY AND CHOICE =
      RECTIFIER )
 THEN :: (WRITE-DOS-FILE "param.dat" INPUT-VOLT POWER-BJT-IC T-RISE
      SNUB-R SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
      CHOICE NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND DOS-CALL "
      simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND DOS-CALL "
      sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" DUMMY-WE )

RULE104
=======
 SUBJECT :: SIMULATE-RULES
 DESCRIPTION :: (Invokes simulation of inverter-machine system)
 IF  :: (CHOICE = INVERTER-MACHINE)
 THEN :: (WRITE-DOS-FILE "param.dat" INPUT-VOLT POWER-BJT-IC T-RISE
      SNUB-R SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
      CHOICE NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND DOS-CALL "
      simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND DOS-CALL "
      sav2asc.exe" "" AND READ-DOS-FILE "expdat.t" IMAX )

RULE105
=======
 SUBJECT :: SIMULATE-RULES
 DESCRIPTION :: (Invokes simulation of the complete system)
 IF  :: (CHOICE = COMPLETE-SYSTEM)
 THEN :: (WRITE-DOS-FILE "param.dat" INPUT-VOLT POWER-BJT-IC T-RISE
      SNUB-R SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
      CHOICE NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND DOS-CALL "
      simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND DOS-CALL "
      sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" DUMMY-WE1 )

RULE134
=======
 SUBJECT :: SIMULATE-RULES
 DESCRIPTION :: (simulates one complete switching cycle of BJT)
 IF  :: (CHOICE = HALF-BRIDGE AND WHAT-SNUB? = ONE-SWITCHING-CYCLE)
 THEN :: (WRITE-DOS-FILE "param.dat" INPUT-VOLT POWER-BJT-IC T-RISE
      SNUB-R SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
      WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat"
      DUMMY-WE1 )


==========================
 Frame :: OPTIMIZE
==========================

 IDENTIFIER :: "OPTIMIZE-"
 TRANSLATION :: (optimizes snubber circuit design)
 PARENTS :: (CONVERTER-DESIGN AC-DRIVE-SYSTEM-DESIGN)
 GOALS :: (R-OPT-DONE R-OPT SNUB-C-OPT OPT-DONE)
 INITIALDATA :: (VOLT-OS-DES)
 PROMPTEVER :: (Snubber optimization in progress .. Please wait --)
 DISPLAYRESULTS :: IFKNOWN
 PARMGROUP :: OPTIMIZE-PARMS
 RULEGROUPS :: (OPTIMIZE-RULES)
 OPTIMIZE-PARMS  :: (A1 A2 A3 A4 A5 A6 A7 A8 B1 B2 B3 B4 B5 B6 B7 B8
          F-CSIM-DONE F-RSIM-DONE L-NCSIM-DONE L-NRSIM-DONE
```

L-PCSIM-DONE L-PRSIM-DONE N-C-ITERATE N-R-ITERATE
P-C-ITERATE P-R-ITERATE POWER-LOSS READY?)
OPTIMIZE-RULES :: (RULE110 RULE121 RULE144 RULE146 RULE150 RULE151
RULE152 RULE153 RULE154 RULE155 RULE156 RULE157
RULE158 RULE159 RULE161 RULE162 RULE163 RULE168
RULE169 RULE170 RULE171 RULE172 RULE173 RULE174)

OPTIMIZE-PARMS

A1
==
TRANSLATION :: (iteration parm for snub res opt, #1)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE150)
ANTECEDENT-IN :: (RULE155 SREFMARK RULE153 RULE154 RULE152)
DEFAULT :: (NO)

A2
==
TRANSLATION :: (iteration parm for snub res opt, #2)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE152)
ANTECEDENT-IN :: (RULE155 SREFMARK RULE153 RULE154)
DEFAULT :: (NO)

A3
==
TRANSLATION :: (iteration parm for snub res opt, #3)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE153)
ANTECEDENT-IN :: (RULE155 SREFMARK RULE154)
DEFAULT :: (NO)

A4
==
TRANSLATION :: (iteration parm for snub res opt, #4)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE154)
ANTECEDENT-IN :: (RULE155)
DEFAULT :: (NO)

A5
==
TRANSLATION :: (iteration parm for snub res opt, #5)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE156)
ANTECEDENT-IN :: (SREFMARK RULE157 RULE158 RULE159)
DEFAULT :: (NO)

A6
==
TRANSLATION :: (iteration parm for snub res opt, #6)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE157)
ANTECEDENT-IN :: (SREFMARK RULE158 RULE159)
DEFAULT :: (NO)

A7
==
TRANSLATION :: (iteration parm for snub res opt, #7)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE158)
ANTECEDENT-IN :: (SREFMARK RULE159)
DEFAULT :: (NO)

A8
==
TRANSLATION :: (iteration parm for snub res opt. #8)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE159)
DEFAULT :: (NO)

B1
==
TRANSLATION :: (iteration parm, snub cap opt, #1)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE162)
ANTECEDENT-IN :: (SREFMARK RULE163 RULE168 RULE169)
DEFAULT :: (NO)

B2
==
TRANSLATION :: (iteration parm, snub cap opt. #2)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE163)
ANTECEDENT-IN :: (SREFMARK RULE168 RULE169)
DEFAULT :: (NO)

B3
==
TRANSLATION :: (iteration parm, snub cap opt, #3)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE168)
ANTECEDENT-IN :: (SREFMARK RULE169)
DEFAULT :: (NO)

B4
==
TRANSLATION :: (iteration parm, snub cap opt. #4)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE169)
DEFAULT :: (NO)

B5
==
TRANSLATION :: (iteration parm, snub cap opt, #5)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE170)
ANTECEDENT-IN :: (SREFMARK RULE171 RULE172 RULE173)
DEFAULT :: (NO)

B6
==
TRANSLATION :: (iteration parm, snub cap opt. #6)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE171)
ANTECEDENT-IN :: (SREFMARK RULE172 RULE173)
DEFAULT :: (NO)

B7
==
TRANSLATION :: (iteration parm, snub cap opt. #7)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE172)
ANTECEDENT-IN :: (SREFMARK RULE173)
DEFAULT :: (NO)

B8
==
TRANSLATION :: (iteration parm, snub cap opt. #8)

TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE173)
DEFAULT :: (NO)

F-CSIM-DONE

TRANSLATION :: (snub. simulation with initial Cs val. done)
TYPE :: YES/NO
UPDATED-IN :: (RULE161)
ANTECEDENT-IN :: (SREFMARK RULE162 RULE163 RULE168 RULE169 RULE170
RULE171 RULE172 RULE173)

F-RSIM-DONE

TRANSLATION :: (first Rs sim. done)
TYPE :: YES/NO
UPDATED-BY :: (RULE110)
ANTECEDENT-IN :: (RULE155 SREFMARK RULE153 RULE154 RULE157 RULE158
RULE150 RULE152 RULE159 RULE156)

L-NCSIM-DONE

TRANSLATION :: (last decremented Cs iteration done)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE173)
USED-BY :: (RULE174)

L-NRSIM-DONE

TRANSLATION :: (last decremented Rs iteration done)
TYPE :: YES/NO
USED-BY :: (RULE146)

L-PCSIM-DONE

TRANSLATION :: (last incremented Cs iteration done)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE169)
USED-BY :: (RULE174)

L-PRSIM-DONE

TRANSLATION :: (last incremented rs iteration done)
TYPE :: YES/NO
UPDATED-IN :: (RULE155)
USED-BY :: (RULE146)

N-C-ITERATE

TRANSLATION :: (iterate Cs in neg. dir.)
TYPE :: YES/NO

N-R-ITERATE

TRANSLATION :: (iterate Rs in neg. dir.)
TYPE :: YES/NO
UPDATED-IN :: (SREFMARK RULE159)

P-C-ITERATE

TRANSLATION :: (iterate Cs in pos. dir.)
TYPE :: YES/NO

P-R-ITERATE

```
 TRANSLATION :: (iterate Rs in pos. dir.)
 TYPE :: YES/NO

POWER-LOSS
==========
 TRANSLATION :: (power loss during turn-off of transistor)
 TYPE :: SINGLEVALUED
 EXPECT :: NUMBER
 RANGE :: (0 50)

READY?
======
 TRANSLATION :: (flag indicating snub. cap opt. possible)
 TYPE :: YES/NO
 UPDATED-BY :: (RULE151)
 USED-BY :: (RULE110)
 ==========================
   OPTIMIZE-RULES
 ==========================
RULE110
=======
 SUBJECT :: OPTIMIZE-RULES
 DESCRIPTION :: (invokes optimization routine for snubber res.)
 IF  :: (USER-CHOICE = OPTIMIZATION-OF-SNUBBER AND READY?)
 THEN :: (WHAT-SNUB? = TURN-ON AND WRITE-DOS-FILE "param.dat" INPUT-VOLT
       IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "
       control.dat" WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "
       " AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" ""
       AND DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat"
       INDEX 22 SNUB-CUR-OS AND F-RSIM-DONE )

RULE121
=======
 SUBJECT :: OPTIMIZE-RULES
 DESCRIPTION :: (Rs-opt, Icmax not available, user wants Cs-opt. Start
           inv-m/c sim. )
 IF  :: (USER-CHOICE = OPTIMIZATION-OF-SNUBBER AND ! CUR-VALUE-KNOWN
       AND R-OPT IS NOTKNOWN )
 THEN :: (CHOICE = INVERTER-MACHINE AND WRITE-DOS-FILE "param.dat"
       OUTPUT-VOLT POWER-BJT-IC T-RISE SNUB-R SNUB-C LEAD-L T_FALL
       AND WRITE-DOS-FILE "control.dat" CHOICE NO-OF-PHASES AND
       DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
       DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
       READ-DOS-FILE "outsav.dat" INDEX 2 VD INDEX 34 IMAX )

RULE144
=======
 SUBJECT :: OPTIMIZE-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (prints message at the end of optimization)
 IF  :: (R-OPT IS KNOWN AND SNUB-C-OPT IS KNOWN AND OPT-DONE)
 THEN :: (PRINT The results of snubber optimization are as follows )

RULE146
=======
 SUBJECT :: OPTIMIZE-RULES
 DESCRIPTION :: (assigns optimum snubber resistance value)
 IF  :: ((SNUB-CUR-OS > (E (* 1.2 (VAL1 FRAME IMAX))) AND SNUB-CUR-OS
       <= (E (* 1.3 (VAL1 FRAME IMAX))) ) OR L-PRSIM-DONE OR
       L-NRSIM-DONE )
 THEN :: (R-OPT-DONE AND R-OPT = (E (* 1 (VAL1 FRAME SNUB-R))))

RULE150
=======
 SUBJECT :: OPTIMIZE-RULES
```

```
ANTECEDENT :: YES
DESCRIPTION :: (first incremented Rs iteration)
IF  :: (F-RSIM-DONE AND SNUB-CUR-OS > (E (* 1.3 (VAL1 FRAME IMAX))))
THEN :: (SNUB-R = (E (+ 2 (VAL1 FRAME SNUB-R))) AND WHAT-SNUB? =
      TURN-ON AND CHOICE = HALF-BRIDGE AND WRITE-DOS-FILE "param.dat"
      INPUT-VOLT IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND
      WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
      DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
      DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
      READ-DOS-FILE "outsav.dat" INDEX 22 SNUB-CUR-OS AND A1 )

RULE151
=======
SUBJECT :: OPTIMIZE-RULES
DESCRIPTION :: (ready to simulate snubber)
IF  :: (IMAX IS KNOWN AND VD IS KNOWN)
THEN :: (READY?)

RULE152
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (second iteration with increased Rs value)
IF  :: (F-RSIM-DONE AND A1 AND SNUB-CUR-OS > (E (* 1.3 (VAL1 FRAME
      IMAX ) ) ) )
THEN :: (SNUB-R = (E (+ 2. (VAL1 FRAME SNUB-R))) AND WHAT-SNUB? =
      TURN-ON AND CHOICE = HALF-BRIDGE AND WRITE-DOS-FILE "param.dat"
      INPUT-VOLT IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND
      WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
      DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
      DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
      READ-DOS-FILE "outsav.dat" INDEX 22 SNUB-CUR-OS AND A2 )

RULE153
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (third iteration with increased Rs value)
IF  :: (F-RSIM-DONE AND A1 AND A2 AND SNUB-CUR-OS > (E (* 1.3 (VAL1
      FRAME IMAX ) ) ) )
THEN :: (SNUB-R = (E (+ 2 (VAL1 FRAME SNUB-R))) AND WHAT-SNUB? =
      TURN-ON AND CHOICE = HALF-BRIDGE AND WRITE-DOS-FILE "param.dat"
      INPUT-VOLT IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND
      WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
      DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
      DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
      READ-DOS-FILE "outsav.dat" INDEX 22 SNUB-CUR-OS AND A3 )

RULE154
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (fourth iteration with increased Rs value)
IF  :: (F-RSIM-DONE AND A1 AND A2 AND A3 AND SNUB-CUR-OS > (E (* 1.3 (
      VAL1 FRAME IMAX ) ) ) )
THEN :: (SNUB-R = (E (+ 2 (VAL1 FRAME SNUB-R))) AND WHAT-SNUB? =
      TURN-ON AND CHOICE = HALF-BRIDGE AND WRITE-DOS-FILE "param.dat"
      INPUT-VOLT IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND
      WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
      DOS-CALL "asc2get.exe\" "" AND DOS-CALL "simodfy.exe" "" AND
      DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
      READ-DOS-FILE "outsav.dat" INDEX 22 SNUB-CUR-OS AND A4 )

RULE155
=======
```

```
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (four incremented Rs iterations over, stop going
        further. )
IF  :: (F-RSIM-DONE AND A1 AND A2 AND A3 AND A4)
THEN :: (L-PRSIM-DONE)

RULE156
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (first decremented Rs iteration)
IF  :: (F-RSIM-DONE AND SNUB-CUR-OS < (E (* 1.2 (VAL1 FRAME IMAX))))
THEN :: (SNUB-R = (E (- 2 (VAL1 FRAME SNUB-R))) AND WHAT-SNUB? =
      TURN-ON AND CHOICE = HALF-BRIDGE AND WRITE-DOS-FILE "param.dat"
      INPUT-VOLT IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND
      WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
      DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
      DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
      READ-DOS-FILE "outsav.dat" INDEX 22 SNUB-CUR-OS AND A5 )

RULE157
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (second decremented Rs iteration)
IF  :: (F-RSIM-DONE AND A5 AND SNUB-CUR-OS < (E (* 1.2 (VAL1 FRAME
      IMAX ) ) ) )
THEN :: (SNUB-R = (E (- 2 (VAL1 FRAME SNUB-R))) AND WHAT-SNUB? =
      TURN-ON AND CHOICE = HALF-BRIDGE AND WRITE-DOS-FILE "param.dat"
      INPUT-VOLT IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND
      WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
      DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
      DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
      READ-DOS-FILE "outsav.dat" INDEX 22 SNUB-CUR-OS AND A6 )

RULE158
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (third decremented Rs iteration)
IF  :: (F-RSIM-DONE AND A5 AND A6 AND SNUB-CUR-OS < (E (* 1.2 (VAL1
      FRAME IMAX ) ) ) )
THEN :: (SNUB-R = (E (- 2 (VAL1 FRAME SNUB-R))) AND WHAT-SNUB? =
      TURN-ON AND CHOICE = HALF-BRIDGE AND WRITE-DOS-FILE "param.dat"
      INPUT-VOLT IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND
      WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
      DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
      DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
      READ-DOS-FILE "outsav.dat" INDEX 22 SNUB-CUR-OS AND A7 )

RULE159
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (fourth decremented Rs iteration)
IF  :: (F-RSIM-DONE AND A5 AND A6 AND A7 AND SNUB-CUR-OS < (E (* 1.2 (
      VAL1 FRAME IMAX ) ) ) )
THEN :: (SNUB-R = (E (- 2 (VAL1 FRAME SNUB-R))) AND WHAT-SNUB? =
      TURN-ON AND CHOICE = HALF-BRIDGE AND WRITE-DOS-FILE "param.dat"
      INPUT-VOLT IMAX T-RISE SNUB-R SNUB-C LEAD-L T_FALL AND
      WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
      DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
      DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
      READ-DOS-FILE "outsav.dat" INDEX 22 SNUB-CUR-OS AND A8 AND !
```

N-R-ITERATE)

RULE161
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (first snubber simulation with available Cs value,
optimized Rs)
IF :: (USER-CHOICE = OPTIMIZATION-OF-SNUBBER AND R-OPT IS KNOWN)
THEN :: (WHAT-SNUB? = TURN-OFF AND WRITE-DOS-FILE "param.dat"
INPUT-VOLT IMAX T-RISE R-OPT SNUB-C LEAD-L T_FALL AND
WRITE-DOS-FILE "control.dat" WHAT-SNUB? NO-OF-PHASES AND
DOS-CALL "asc2get.exe" "" AND DOS-CALL "simodfy.exe" "" AND
DOS-CALL "simnon.exe" "" AND DOS-CALL "sav2asc.exe" "" AND
READ-DOS-FILE "outsav.dat" INDEX 22 VOLT-OS-SIM AND
F-CSIM-DONE)

RULE162
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (first iteration for incremented Cs value)
IF :: (F-CSIM-DONE AND VOLT-OS-SIM > VOLT-OS-DES)
THEN :: (SNUB-C = (E (* 2.2 (VAL1 FRAME SNUB-C))) AND WHAT-SNUB? =
TURN-OFF AND WRITE-DOS-FILE "param.dat" INPUT-VOLT IMAX T-RISE
R-OPT SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" INDEX
22 VOLT-OS-SIM AND B1)

RULE163
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (second iteration with incremented Cs value)
IF :: (F-CSIM-DONE AND B1 AND VOLT-OS-SIM > VOLT-OS-DES)
THEN :: (SNUB-C = (E (* 2. (VAL1 FRAME SNUB-C))) AND WHAT-SNUB? =
TURN-OFF AND WRITE-DOS-FILE "param.dat" INPUT-VOLT IMAX T-RISE
R-OPT SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" INDEX
22 VOLT-OS-SIM AND B2)

RULE168
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (third incremented Cs iteration)
IF :: (F-CSIM-DONE AND B1 AND B2 AND VOLT-OS-SIM > VOLT-OS-DES)
THEN :: (SNUB-C = (E (* 1.5 (VAL1 FRAME SNUB-C))) AND WHAT-SNUB? =
TURN-OFF AND WRITE-DOS-FILE "param.dat" INPUT-VOLT IMAX T-RISE
R-OPT SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" INDEX
22 VOLT-OS-SIM AND B3)

RULE169
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (fourth incremented Cs iteration, and don't continue in
this direction)

```
IF  :: (F-CSIM-DONE AND B1 AND B2 AND B3 AND VOLT-OS-SIM > VOLT-OS-DES
    )
THEN :: (SNUB-C = (E (* 1.2 (VAL1 FRAME SNUB-C))) AND WHAT-SNUB? =
     TURN-OFF AND WRITE-DOS-FILE "param.dat" INPUT-VOLT IMAX T-RISE
     R-OPT SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
     WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
     DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" INDEX
     22 VOLT-OS-SIM AND B4 AND L-PCSIM-DONE )

RULE170
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (first Cs decremented iteration)
IF  :: (F-CSIM-DONE AND VOLT-OS-SIM < 5)
THEN :: (SNUB-C = (E (* 0.6 (VAL1 FRAME SNUB-C))) AND WHAT-SNUB? =
     TURN-OFF AND WRITE-DOS-FILE "param.dat" INPUT-VOLT IMAX T-RISE
     R-OPT SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
     WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
     DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" INDEX
     22 VOLT-OS-SIM AND B5 )

RULE171
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (second decremented Cs iteration)
IF  :: (F-CSIM-DONE AND B5 AND VOLT-OS-SIM < 5)
THEN :: (SNUB-C = (E (* 0.7 (VAL1 FRAME SNUB-C))) AND WHAT-SNUB? =
     TURN-OFF AND WRITE-DOS-FILE "param.dat" INPUT-VOLT IMAX T-RISE
     R-OPT SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
     WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
     DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" INDEX
     22 VOLT-OS-SIM AND B6 )

RULE172
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (third decremented Cs iteration)
IF  :: (F-CSIM-DONE AND B5 AND B6 AND VOLT-OS-SIM < 5)
THEN :: (SNUB-C = (E (* 0.8 (VAL1 FRAME SNUB-C))) AND WHAT-SNUB? =
     TURN-OFF AND WRITE-DOS-FILE "param.dat" INPUT-VOLT IMAX T-RISE
     R-OPT SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
     WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
     DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" INDEX
     22 VOLT-OS-SIM AND B7 )

RULE173
=======
SUBJECT :: OPTIMIZE-RULES
ANTECEDENT :: YES
DESCRIPTION :: (fourth decremented Cs iteration. stop iteration process
          )
IF  :: (F-CSIM-DONE AND B5 AND B6 AND B7 AND VOLT-OS-SIM < 5)
THEN :: (SNUB-C = (E (* 0.9 (VAL1 FRAME SNUB-C))) AND WHAT-SNUB? =
     TURN-OFF AND WRITE-DOS-FILE "param.dat" INPUT-VOLT IMAX T-RISE
     R-OPT SNUB-C LEAD-L T_FALL AND WRITE-DOS-FILE "control.dat"
     WHAT-SNUB? NO-OF-PHASES AND DOS-CALL "asc2get.exe" "" AND
     DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "sav2asc.exe" "" AND READ-DOS-FILE "outsav.dat" INDEX
```

22 VOLT-OS-SIM AND B8 AND L-NCSIM-DONE)

RULE174

SUBJECT :: OPTIMIZE-RULES
DESCRIPTION :: (volt overshoot within limits, stop Cs iteration)
IF :: ((VOLT-OS-SIM < VOLT-OS-DES AND VOLT-OS-SIM > 5.) OR
L-PCSIM-DONE OR L-NCSIM-DONE)
THEN :: (SNUB-C-OPT = (E (* 1 (VAL1 FRAME SNUB-C))) AND OPT-DONE)

APPENDIX C.2 SIMNON PROGRAM LISTING

APPENDIX C.2.1 Converter-Machine Simulation Programs

```
"*****************************************************************
 "AC POWER SUPPLY
"*****************************************************************
CONTINUOUS SYSTEM AC
OUTPUT vas vbs vcs
TIME t
"----Generate 3-phase sinusoidal voltages
vas=Vm*sin(wt)
vbs=Vm*sin(wt-2.0944)
vcs=Vm*sin(wt-4.1888)
wt=we*t
we=377
Vm:179.63      "179.605(220 v)
END

"*****************************************************************
 "Rectifier-Inverter system
"*****************************************************************
CONTINUOUS SYSTEM rfcin
"LOSSY DIODE CONSIDERED
INPUT vas vbs vcs Ii
OUTPUT Vd
STATE Vd1 Id x1 x2
DER DVd1 DId Dx1 Dx2
"
"BRIDGE OUTPUT VOLTAGE (RECTIFIED SINE WAVE)
VP=MAX(vas,MAX(vbs, vcs))
VN=MIN(vas,MIN(vbs, vcs))
Vr=VP-VN
"
"FILTERED DC LINK VOLTAGE
DVd1=(Id1-Ii)/Cf
VDD=VD0+K*(abs(Id1)^m)
AUX=(Vr-Vd-2*VDD)/(2*Ls)
DId=IF Id<0 AND AUX<0 THEN 0 ELSE AUX
Id1=IF Id>0 THEN Id ELSE 0
Vd=Vd1
"-------COMPUTATION OF RMS FILTER CAPACITOR CURRENT
Icf2=(Ii+Id1)*(Ii+Id1)
Icf=x1
Dx1=x2
Dx2=(-2*0.0167*x2-x1+Icf2)/(0.0167*0.0167)
Icr=sqrt(abs(Icf))

"
"DC LINK TO SUPPLY CURRENT TRANSFORMATION
Tid_isa=IF (vas>vbs) AND (vas>vcs) THEN 1 ELSE AUX1
A=(vas>vbs) and (vas<vcs)
B=(vas<vbs) and (vas> vcs)
AUX1=IF A OR B THEN 0 ELSE -1
ias=Tid_isa*Id1
"
"PARAMETERS
Ls:0.0005 "Ls=0.5mH
Cf:0.0048      "Cf=4800uF
Vd1:314
Id:30
VD0:0.80767
K:0.0454
m:0.5981
```

```
END

"***************************************************************
 "Sinusoidal PWM Inverter System
"***************************************************************
CONTINUOUS SYSTEM pvsiin
INPUT ia ib ic Vd
OUTPUT  van vbn vcn we Ii Nsaux sinwt coswt

TIME t

"TRIANGULAR WAVE GENERATOR
tau = 1/fs          "period of carrier wave
tb2 = tau/2
tsh = mod(t,tau)    "remainder of t/tau
m = 4*a/tau         "slope
y = if tsh<tb2 then (m*tsh-a) else (-m*(tsh-tb2)+a)

"PWM GENERATOR
"parameters
pi = atan(1)*4
w = 2*pi*fm             "modulating frequency in rad/s
we = w
wt = w*t
sinwt = sin(wt)         "next three variables are to be
coswt = cos(wt)          "used with IMS1
Nsaux = 2*fs/fm
vm = mf*a               "Modified (V/F control law)
phase = (2*pi)/3
"Sine modulating waveforms
vam = vm*SIN(wt)
vbm = vm*SIN(wt-phase)
vcm = vm*SIN(wt+phase)
"Output voltages
vao = if vam>y then vd else 0
vbo = if vbm>y then vd else 0
vco = if vcm>y then vd else 0
van = (2*vao-vbo-vco)/3   "output phase voltages
vbn = (2*vbo-vao-vco)/3
vcn = (2*vco-vao-vbo)/3
vab = van - vbn
"Inverter input current
Ii=(van*ia+vbn*ib+vcn*ic)/vd

a:10
fm:60             "modulating frequency
fs:5000           "carrier frequency
mf:1          "modulating factor
Tf = 1/fm
END

"***************************************************************
 "Induction Motor d-q axis Model
"***************************************************************
DISCRETE SYSTEM mxia
"This system supplies the maximum value of snubber voltage overshoot
INPUT abia
OUTPUT maxia
STATE x4
NEW  nx4
TIME t
TSAMP ts
nx4 = IF t < 0.35 THEN 0 ELSE AUX
AUX = IF (abia > x4) THEN abia ELSE x4
maxia = nx4
```

```
ts=t+h
h:10e-6
END

"*****************************************************************
 "Connecting System for Converter-Machine Simulation
"*****************************************************************
CONNECTING SYSTEM crfinv
"COMPLETE SYSTEM - SOURCE, RECTIFIER, FILTER, PWM VSI AND MACHINE
vas[RFCIN]=vas[AC]
vbs[RFCIN]=vbs[AC]
vcs[RFCIN]=vcs[AC]
"
Ii[RFCIN]=Ii[PVSIIN]
Vd[PVSIIN]=Vd[RFCIN]
"
VA[MACHIN]=VAN[PVSIIN]
VB[MACHIN]=VBN[PVSIIN]
VC[MACHIN]=VCN[PVSIIN]
"
IA[PVSIIN]=IA[MACHIN]
IB[PVSIIN]=IB[MACHIN]
IC[PVSIIN]=IC[MACHIN]
"
We[MACHIN]=We[PVSIIN]
Nsaux[MACHIN]=Nsaux[PVSIIN]
"
SINWT[MACHIN]=SINWT[PVSIIN]
COSWT[MACHIN]=COSWT[PVSIIN]
"
ABIA[MXIA] = ABIA[MACHIN]
MAXIA[MACHIN] = MAXIA[MXIA]
END
```

APPENDIX C.2.2 Listing for Converter Half-Bridge Simulation Programs

```
"****************************************************************
 "Half Bridge Model during Turn-on and Turn-off
"****************************************************************
CONTINUOUS SYSTEM onoff
INPUT oiC1 mx mxiq
OUTPUT iC1 vC i1
STATE ii vC1 Es
DER Dii DvC1 DEs
TIME t
B = IF(t<15e-6) THEN 1 ELSE 0
v1 = IF (t<tr) THEN Vd*(1-t/tr) ELSE AUX1
AUX1 = IF (B<1) AND (ii>0) THEN vC1 ELSE AUX12
AUX12 = IF (B<1) THEN vC1 + oiC1*R ELSE 0
iC1 = C*DvC1
vD4 = IF (ii<I) THEN 0 ELSE Vd
iD4 = I - ii
i1 = IF (B>0) THEN ii - oiC1 ELSE AUX11
AUX11 = IF (B<1) AND (t<15e-6+tf) THEN I*(1-(t-15e-6)/tf) ELSE 0
Dii = IF (B>0) AND (ii < I) THEN (Vd - v1)/Ll ELSE AUX2
AUX2 = IF (B>0) THEN 0 ELSE AUX3
AUX3 = IF (B<1) AND NOT(vC1>Vd-vD0b) THEN 0 ELSE AUX4
AUX4 = IF NOT (ii<0) THEN C1 ELSE C2
C1 = (Vd - vD0s)/Ll - (ks/Ll)*((abs(ii))^ms) - vC1/Ll
C2 = (Vd/Ll) - (R/Ll)*ii - (vC1/Ll)
DvC1 = IF (B>0) THEN (v1-vC1)/(R*C) ELSE AUX5
AUX5 = IF (B<1) AND NOT(i1<0) THEN (ii - i1)/C ELSE ii/C
vC = 100*(vC1 - Vd)/Vd
Ps = i1*v1
DEs = Ps
P = (fs/(4*PI))*Es
PI = 3.1415927
"Parameters
fs:5000
tf:1e-6
tr:1e-6
Vd:314
I:25
Ll:5e-6
C:220e-9
R:15
vD0s:0.80767
vD0b:0.80767
kb:0.0454
ks:0.0454
mb:0.5981
ms:0.5981

"INITIAL CONDITIONS
ii:0
vC1:314

END
"****************************************************************
 "Delay Subsystem for Avoiding Algebraic Loops
"****************************************************************
DISCRETE SYSTEM del
INPUT iC1
OUTPUT oiC1
STATE x1
NEW nx1
TIME t
```

```
TSAMP ts
nx1 = iC1
oiC1 = x1
ts = t + h
h:5e-9
x1:0
END

"*****************************************************************
 "Peak Voltage Detector for Turn-off Voltage Overshoot Calculation
"*****************************************************************
DISCRETE SYSTEM maxvc
"This system supplies the maximum value of snubber voltage overshoot
INPUT vC
OUTPUT mx
STATE x2
NEW  nx2
TIME t
TSAMP ts
nx2 = IF (vC > x2) THEN vC ELSE x2
mx = nx2
ts=t+h
h:1e-9
END

"*****************************************************************
 "Peak Current Detector for Turn-on Current Overshoot Calculation
"*****************************************************************
DISCRETE SYSTEM maxiq
"This system supplies the maximum value of snubber voltage overshoot
INPUT i1
OUTPUT mxiq
STATE x3
NEW  nx3
TIME t
TSAMP ts
nx3 = IF (i1 > x3) THEN i1 ELSE x3
mxiq = nx3
ts=t+h
h:1e-9
END

"*****************************************************************
 " Connecting System for Half-Bridge Switching Simulation
"*****************************************************************
CONNECTING SYSTEM conoff
iC1[del] = iC1[onoff]
oiC1[onoff] = oiC1[del]
vC[maxvc] = vC[onoff]
mx[onoff] = mx[maxvc]
i1[maxiq] = i1[onoff]
mxiq[onoff] = mxiq[maxiq]
END
```

Appendix C.3 C - Based Interface Routines Listing

```
/*****************************************************************/
/* asc2get.c -- C routine for format conversion from flat ASCII  */
/*          to SIMNON 'GET' parameter data file. This is the */
/*          routine where parameters obtained from the      */
/*          Expert System based design of PWM VSI are passed */
/*          onto the file PARAM.DAT, the format of which is  */
/*          then converted to SIMNON 'GET' file INGET.T,     */
/*          using the control information from another ES    */
/*          based control data file CONTROL.DAT.             */
/*                                                           */
/*          Author: Sunil M. Chhaya                  */
/*          Last Modified on : 03/16/94               */
/*****************************************************************/
#include <stdio.h>     /* For FILE definition */
#include <io.h>        /* for EOF definition */
#include <string.h>    /* for strcmp definition */
#include <math.h>      /* for scientific functions */
#define MAXBUF 128

main()
{
   static char fname_in1[] = "PARAM.DAT", fname_in2[] = "CONTROL.DAT", fname_out[] = "inget.t";
   FILE *input_fil1, *input_fil2, *output_file;
   float we, vs, l_s, cf, vd0, k, m, fs;
   float i, tr, qrr, tf, rs, cs, ll, u, v, x, y, uu, vv, xx, ww;
   float idse_ref, iqse_ref, wref, vdc, wb, rs1, rr, lr, ls, lm, j, b, pole, kll;
   char syst_of_interest[MAXBUF], no_of_phases[MAXBUF];
 /* Following section verifies the opening of input/output files */
   if ((input_fil1 = fopen(fname_in1, "r")) == NULL)
            {
            printf("Error opening parameter data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   if ((input_fil2 = fopen(fname_in2, "r")) == NULL)
            {
            printf("Error opening control data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   if ((output_file = fopen(fname_out, "w")) == NULL)
            {
            printf("Error opening SIMNON 'GET' file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   printf("PERFORMING FORWARD DATA FORMAT CONVERSION FOR SIMNON... PLEASE WAIT..\n");
   /* Following section reads the data from param.dat */
   fscanf(input_fil2, "%s %s", syst_of_interest, no_of_phases);
   ww = strcmp(syst_of_interest, "SIMPLNT");
   if (ww == 0)
            {
            fscanf(input_fil1, "%f %f %f %f", &idse_ref, &iqse_ref, &wref, &vdc);
            fscanf(input_fil1, "%f %f %f %f", &wb, &rs1, &rr, &lr);
            fscanf(input_fil1, "%f %f %f %f", &ls, &lm, &j, &pole);
            }
   else
            {
            fscanf(input_fil1, "%f %f %f", &vs, &i, &tr);
            fscanf(input_fil1, "%f %f %f %f", &rs, &cs, &ll, &tf);
```

```
        }
u = strcmp (syst_of_interest, "RECTIFIER");
v = strcmp (syst_of_interest, "INVERTER-MACHINE");
x = strcmp (syst_of_interest, "COMPLETE-SYSTEM");
uu = strcmp(syst_of_interest, "TURN-ON");
vv = strcmp(syst_of_interest, "TURN-OFF");
xx = strcmp(syst_of_interest, "ONE-SWITCHING-CYCLE");
/* Following section prints the above data in a format   */
/* readable by SIMNON, in the form of a 'GET' file when   */
/* the user wants rectifier/inverter/filter to be observed */

if (ww == 0)
        {
        fprintf(output_file, "[SYNSTA]\n\n");
        fprintf(output_file, "[PHASE23]\n\n");
        fprintf(output_file, "[SPWM]\n");
        fprintf(output_file, "fs:15000\n");
        fprintf(output_file, "Vd:%f\n", vdc);
        fprintf(output_file, "[MACHINDQ]\n");
        fprintf(output_file, "wb:%f\n", wb);
        fprintf(output_file, "rs:%f\n", rs1);
        fprintf(output_file, "rr:%f\n", rr);
        fprintf(output_file, "Lr:%f\n", lr);
        fprintf(output_file, "Ls:%f\n", ls);
        fprintf(output_file, "Lm:%f\n", lm);
        fprintf(output_file, "j:%f\n", j);
        fprintf(output_file, "pole:%f\n", pole);
        fprintf(output_file, "kFW:2.64e-5\n"); */
        fprintf(output_file, "[CONPLA]\n\n");
        }
else if (0 == (u && v && x))
        {
        fprintf(output_file, "[AC]\n");
        if (0 == (strcmp (no_of_phases, "THREE_PHASE")))
          {
          fprintf(output_file, "Vm:%7.2f\n", vs * sqrt(2) / sqrt(3));
          }
        else if (0 == (strcmp (no_of_phases, "SINGLE_PHASE")))
          {
          fprintf(output_file, "Vm:%7.2f\n", vs * sqrt(2));
          }

        fprintf(output_file, "[RFCIN]\n");
        fprintf(output_file, "Ls:%-7.4f\n", 0.0002);
        fprintf(output_file, "Cf:%-7.4f\n", 0.0048);
        fprintf(output_file, "VD0:%-5.4f\n", 0.80767);
        fprintf(output_file, "k:%-5.4f\n", 0.0454);
        fprintf(output_file, "m:%-5.4f\n",0.5981);
        fprintf(output_file,"Vd1:%-5.4f\n", vs*sqrt(2));
        fprintf(output_file, "[PVSIIN]\n");
        fprintf(output_file, "[MACHIN]\n");
        fprintf(output_file, "[MXIA]\n");
        fprintf(output_file, "[CRFINV]\n");
        }
        else if (0 == (uu && vv && xx))
        {
        fprintf(output_file, "[ONOFF]\n");
        fprintf(output_file, "I:%-7.2f\n", i);
        fprintf(output_file, "tr:%-4.2e\n", 1e-6);
        fprintf(output_file, "tf:%-4.2e\n", tf*1e-6);
        fprintf(output_file, "R:%-7.2f\n", rs);
        fprintf(output_file, "C:%-4.2e\n", cs * 1e-6);
        fprintf(output_file, "Ll:%-4.2e\n", ll * 1e-6);
        fprintf(output_file, "ii:%-4.2f\n", 0.00);
        fprintf(output_file, "vC1:%-4.2f\n", vs*sqrt(2));
```

```
        if (0 == (strcmp (no_of_phases, "THREE_PHASE")))
          {
          fprintf(output_file, "Vd:%-7.2f\n", vs * sqrt(2));
          }
        else if (0 == (strcmp (no_of_phases, "SINGLE_PHASE")))
          {
          fprintf(output_file, "Vd:%-7.2f\n", vs * sqrt(2));
          }
        fprintf(output_file, "vD0s:%-7.5f\n", 0.80767);
        fprintf(output_file, "vD0b:%-7.5f\n", 0.80767);
        fprintf(output_file, "ks:%-7.5f\n", 0.0454);
        fprintf(output_file, "kb:%-7.5f\n", 0.0454);
        fprintf(output_file, "ms:%-7.5f\n", 0.5981);
        fprintf(output_file, "mb:%-7.5f\n", 0.5981);
        fprintf(output_file, "[MAXVC]\n");
        fprintf(output_file, "[DEL]\n");
        fprintf(output_file, "[CONOFF]\n");
        }
  /* Following section verifies file closure */
  if (fclose(input_fil1) == EOF)
        {
        printf("Error closing parameter data file.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
  if (fclose(input_fil2) == EOF)
        {
        printf("Error closing control data file.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
  if (fclose(output_file) == EOF)
        {
        printf("Error closing SIMNON 'GET' file.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
}

/*******************************************************/
/* simodfy.c -- This is the C program achieving control conversion   */
/*        It reads control data from file CONTROL.DAT and     */
/*        generates a macro SIMNON.T, to automate simulation in */
/*        SIMNON environment                                  */
/* Program perfected: 3/25/93                                 */
/* Last options added: 3/11/94                                */
/*******************************************************/

#include <stdio.h>    /* for FILE definition */
#include <io.h>                  /* for EOF definition */
#include <string.h>   /* for strcmp() definition */
#include <malloc.h>    /* for malloc()       */
#include <ctype.h>    /* for isupper()      */
#include <errno.h>    /* for remove()       */
#define MAXBUF 128
#define HUNK 512

main()
{
  static char fname_in1[] = "CONTROL.DAT", fname_in2[] = "SIMNON.OLD";
  static char fname_out[] = "SIMNON.T";
  FILE *input_fil1, *input_fil2, *output_file;
```

```
char user_choice[MAXBUF], *cp, newname[128], *np;
int a=0, b=0, c=0, d=0, e=0, f=0, hunks = 0, bytes = 0, totbytes = 0;
int i, g=0, h=0, j=0, l=0, m=0;

/* The old SIMNON.T version is deleted first, before creating */
/* the new one.                                               */
remove(fname_out);
/* The following block of code copies the SIMNON.OLD file to  */
/* temporary file SIMNON.T, which will be appended later,     */
/* based on the user responses.                               */
if ((input_fil1 = fopen(fname_in1, "r")) == NULL)
        {
        printf("Error opening control data file.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }

if ((input_fil2 = fopen(fname_in2, "rb")) == NULL)
        {
        printf("\"%s\": can't open.\n", fname_in2);
        exit(1);
        }
        printf("PERFORMING CONTROL DATA CONVERSION .. \n");
if ((cp = malloc(HUNK)) == NULL)
        {
        printf("Malloc failed.\n");
        exit(1);
        }
while ((bytes = fread(cp + (HUNK * hunks), 1, HUNK, input_fil2)) == HUNK)
        {
        totbytes += bytes;
        ++hunks;
        if ((cp = realloc(cp, HUNK + (HUNK * hunks))) == NULL)
           {
           printf("Realloc failed.\n");
           exit(1);
           }
        }
if (bytes < 0)
        {
        printf("\"%s\": Error reading. \n", fname_in2);
        exit(1);
        }
totbytes += bytes;

for (i = 0; i < totbytes; ++i)
        if (islower(cp[i]))
           cp[i] = tolower(cp[i]);
(void)fclose(input_fil2);
if ((np = strrchr(fname_in2, '.')) != NULL)
        *np = '\0';
strcpy(newname, fname_in2);
strcat(newname, ".T");

if ((output_file = fopen(newname, "wb")) == NULL)
        {
        printf("\"%s\": Can't open.\n", output_file);
        exit(1);
        }
if (fwrite(cp, 1, totbytes, output_file) != totbytes)
   {
   printf("\"%s\": Error writing.\n", output_file);
   exit(1);
   }
```

```
(void)fclose(output_file);
/* The following block of code appends the intermediate SIMNON.T */
/* file with the control command block, based on user choice    */
if ((output_file = fopen(newname, "a+")) == NULL)
        {
        printf("Error opening SIMNON.T.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
fscanf(input_fil1, "%s", user_choice);
if (0 == strcmp (user_choice, "RECTIFIER"))
        a=1;
else if (0 == strcmp (user_choice, "INVERTER-MACHINE"))
        b=1;
else if (0 == strcmp (user_choice, "COMPLETE-SYSTEM"))
        c=1;
else if (0 == strcmp (user_choice, "TURN-OFF"))
        d=1;
else if (0 == strcmp (user_choice, "TURN-ON"))
        e=1;
else if (0 == strcmp (user_choice, "ONE-SWITCHING-CYCLE"))
        f=1;
else if (0 == strcmp (user_choice, "SIMPLNT"))
        g = 1;
else if (0 == strcmp (user_choice, "DACQDISP"))
        h = 1;
else if (0 == strcmp (user_choice, "QACQDISP"))
        j = 1;
else if (0 == strcmp (user_choice, "WACQDISP"))
        l = 1;
else if (0 == strcmp (user_choice, "TACQDISP"))
        m = 1;
else
        {
        a=0;
        b=0;
        c=0;
        d=0;
        e=0;
        f=0;
        g=0;
        h=0;
        j=0;
        l=0;
        m=0;
        }
fprintf(output_file, "LABEL ctrl\n");
if (a == 1)
        {
        fprintf(output_file, "LET a = 1\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 0\n");
        fprintf(output_file, "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }

else if (b == 1)
        {
```

```
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 1\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 0\n");
        fprintf(output_file, "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
else if (c == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 1\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 0\n");
        fprintf(output_file, "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
else if (d == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 1\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 0\n");
        fprintf(output_file, "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
else if (e == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 1\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 0\n");
        fprintf(output_file, "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
   else if (f == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 1\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 0\n");
```

```
        fprintf(output_file, "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
else if (g == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 1\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 0\n");
        fprintf(output_file, "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
else if (h == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 1\n");
        fprintf(output_file, "LET j = 0\n");
        fprintf(output_file, "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
else if (j == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 1\n");
        fprintf(output_file. "LET l = 0\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
else if (l == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
        fprintf(output_file, "LET e = 0\n");
        fprintf(output_file, "LET f = 0\n");
        fprintf(output_file, "LET g = 0\n");
        fprintf(output_file, "LET h = 0\n");
        fprintf(output_file, "LET j = 0\n");
        fprintf(output_file, "LET l = 1\n");
        fprintf(output_file, "LET m = 0\n\n");
        }
else if (m == 1)
        {
        fprintf(output_file, "LET a = 0\n");
        fprintf(output_file, "LET b = 0\n");
        fprintf(output_file, "LET c = 0\n");
        fprintf(output_file, "LET d = 0\n");
```

```
            fprintf(output_file, "LET e = 0\n");
            fprintf(output_file, "LET f = 0\n");
            fprintf(output_file, "LET g = 0\n");
            fprintf(output_file, "LET h = 0\n");
            fprintf(output_file, "LET j = 0\n");
            fprintf(output_file, "LET l = 0\n");
            fprintf(output_file, "LET m = 1\n\n");
            }
  else
            {
            printf("Invalid choice. Try again!\n");
            }
  fprintf(output_file, "IF a EQ 1 GOTO rectif\n");
  fprintf(output_file, "IF b EQ 1 GOTO pwmvsi\n");
  fprintf(output_file, "IF c EQ 1 GOTO filter\n");
  fprintf(output_file, "IF d EQ 1 GOTO bjtoff\n");
  fprintf(output_file, "IF e EQ 1 GOTO bjton\n");
  fprintf(output_file, "IF f EQ 1 GOTO hfbrid\n");
  fprintf(output_file, "IF g EQ 1 GOTO simplnt\n");
  fprintf(output_file, "IF h EQ 1 GOTO dacqdisp\n");
  fprintf(output_file, "IF j EQ 1 GOTO qacqdisp\n");
  fprintf(output_file, "IF l EQ 1 GOTO wacqdisp\n");
  fprintf(output_file, "IF m EQ 1 GOTO tacqdisp\n\n");
  fprintf(output_file, "LABEL endm\n\n");
  fprintf(output_file, "STOP\n");
   if (fclose(input_fil1) == EOF)
            {
            printf("Error closing control data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
  if (fclose(output_file) == EOF)
            {
            printf("Error closing SIMNON.T file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
}


/*************************************************************/
/* sav2asc.c -- This is the filter routine, which filters out
                bare ASCII data relevant to PC Plus expert system
                from the 'SAVE' file generated by SIMNON.
                All the data are filtered out from 'SAVE' file,
                and PC Plus decides, based on user choice, which
                data are to be used, and which ones to be ignored.
                This feature makes this routine general-purpose. */
/*************************************************************/

#include <stdio.h>    /* For FILE, BUFSIZ definition */
#include <io.h>      /* For EOF definition  */
#include <stdlib.h>   /* For atof() definition */
#include <string.h>   /* For strcpy() definition */
#include <conio.h>    /* For getche() definition */
#include <errno.h>    /* For remove()        */
#define MAXBUF 128

main()
{
  /* The following section initializes various variables and pointers */
  static char fname_in1[] = "INSAV.T", fname_in2[] = "EXPDAT.T";
  static char fname_in3[] = "CONTROL.DAT", fname_out[] = "OUTSAV.DAT", buf[BUFSIZ];
```

```
/* INSAV.T is the file generated by 'SAVE' command from SIMNON
   OUTSAV.DAT is the datafile filtered from the INSAV.T file.
   OUTSAV.DAT has just the numbers, readable directly from PC PLUS.
   EXPDAT.T has the exported data available from various simulations */
FILE *input_fil1, *input_fil2, *input_fil3, *output_file;
char var_str[10], ch, syst_of_interest[MAXBUF];
int  count = 0, len = 0, var_count = 0, max_len = 10;
int u = 0, v = 0, w = 0;
float num[33], volt_os, off_power, dat_len, peak_ia, mxiq, p_on ;

remove(fname_out);
/* This array (num) stores the ascii numbers filtered from
 the input data file outsav.t generated by SIMNON  */
/* Input and output files are opened in the following section */
if ((input_fil1 = fopen(fname_in1, "r")) == NULL)
        {
        printf("Error opening saved data file.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
if ((input_fil3 = fopen(fname_in3, "r")) == NULL)
        {
        printf("Error opening control data file.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
if ((output_file = fopen(fname_out, "a+")) == NULL)
        {
        printf("Error opening PC Plus input ASCII file.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }

printf("PERFORMING REVERSE DATA CONVERSION .. \n");
fscanf(input_fil3, "%s", syst_of_interest);
u = strcmp(syst_of_interest, "TURN-OFF");
v = strcmp(syst_of_interest, "TURN-ON");
w = strcmp(syst_of_interest, "ONE-SWITCHING-CYCLE");
if ((u && v && w) == 0)
        {
        dat_len = 20;
        }
else
        dat_len = 32;
        /* All variables are read from this point onwards */
        while (var_count <= dat_len)
          {
          while (((ch = fgetc(input_fil1)) != EOF) && ch != ':')
                  {
                  /* Appearance of a colon (:) is sensed for data read */

                  count++;
                  if (ch == '\n') /* reset count if newline encountered */
                    {
                    count = 0;
                    }
                  }
           /* A colon encountered meaning the next character string
            represents numerical data, truncated by a '\n' */
          count = 0;
          while ( len <= max_len && ch != '\n')
                  {
```

```
                ch = fgetc(input_fil1);
                if (ch == '\n')
                  {
                  break;
                  }
                if (ch == '\0')
                  {
                  ch = ' ';
                  }
                buf[count++] = ch;
                len++;
                }
            buf[count++] = '\0';
            strcpy (var_str, buf);
            /* Convert this character string to a floating point number */
            num[var_count++] = atof(var_str);
            fprintf(output_file, "%f\n", num[var_count - 1]);
            count = 0;
            len = 0;
            }
          if (strcmp(syst_of_interest, "TURN-OFF") == 0)
            {
            if ((input_fil2 = fopen(fname_in2, "r")) == NULL)
                {
                printf("Error opening snubber data file.\n");
                printf("Press any key to continue.\n");
                getche();
                exit(0);
                }
            fscanf(input_fil2, "%f %f", &volt_os, &off_power);
            fprintf(output_file, "%f\n", volt_os);
            fprintf(output_file, "%f\n", off_power);
            }
    else if (strcmp(syst_of_interest, "INVERTER-MACHINE") == 0)
            {
            if ((input_fil2 = fopen(fname_in2, "r")) == NULL)
                {
                printf("Error opening inv. current data file.\n");
                printf("Press any key to continue.\n");
                getche();
                exit(0);
                }
            fscanf(input_fil2, "%f", &peak_ia);
            fprintf(output_file, "%f\n", peak_ia);
            }
    else if (strcmp(syst_of_interest, "TURN-ON") == 0)
            {
            if ((input_fil2 = fopen(fname_in2, "r")) == NULL)
                {
                printf("Error opening inv. current data file.\n");
                printf("Press any key to continue.\n");
                getche();
                exit(0);
                }
            fscanf(input_fil2, "%f", &mxiq);
            fprintf(output_file, "%f\n", mxiq);
            }
      else { }
    fcloseall();
}
```

APPENDIX D Software Listing for Control Design and Auto-Tuning

APPENDIX D.1 ARL Listing of Expert System for Control Design and Auto Tuning

```
Frame :: CONTROL-DESIGN

  IDENTIFIER :: CONTROL-DESIGN-
  TRANSLATION :: (Design of the control system for the drive)
  PARENTS :: (AC-DRIVE-SYSTEM-DESIGN)
  GOALS :: (KPD WCD KPQ WCQ KPS WCS KPD-OPT WCD-OPT KPQ-OPT WCQ-OPT KPS-OPT
        WCS-OPT CONT-OPT-KNOWN )
  PROMPTEVER :: ("The closed loop ac drive system chosen here has two types
            of controllers:" :LINE "faster, current controllers in the
            inner loop, and the slower, speed" :LINE "controller in the
            outer loop. The inner controllers have a default sampling"
            :LINE "interval of 200 μs, whereas the outer controller has
            a slower sampling time" :LINE "of 1 ms." :LINE "      The
            inner controllers have been implemented in DSP 320C25"
            :LINE "assembly language, while the outer, slower
            controller has been implemented" :LINE "in C language,
            (later compiled to DSP assembly)." :LINE 2 "The controllers
            used here are of P-I type, the ones most commonly used in"
            :LINE "practice, for their simplicity of implementation and
            zero steady state error." :LINE 2 "The Kp and Ki values for
            current controllers are designed based on the" :LINE "

machine electrical parameters, such as magnetizing inductance and stator"
            :LINE "resistance, and the PI gains of the speed controller
            are designed based on" :LINE "load mechanical parameter, J,
            the moment of inertia. The load is chosen to" :LINE "be
            pure inertia. The controller gains are so designed as to
            make the" :LINE "response overdamped, to maintain stability
            of the system." :LINE 2 "The designed controller gain
            values are later fed to the controllers, for" :LINE "their
            optimization based on iterative simulation." )
    DISPLAYRESULTS :: YES
    PARMGROUP :: CONTROL-DESIGN-PARMS
    RULEGROUPS :: (CONTROL-DESIGN-RULES)
    OFFSPRING :: (IDS-TUNING IQS-TUNING SPD-TUNING TEST-RUN)
    CONTROL-DESIGN-PARMS  :: (BASE-SPD CONT-OPT-KNOWN CONT-OPT? DELN-KPD
                  DELN-KPQ DELN-KPS DELN-WCD DELN-WCQ DELN-WCS
                  DELP-KPD DELP-KPQ DELP-KPS DELP-WCD DELP-WCQ
                  DELP-WCS IDOPT-DONE IDS-RATED IDS-REF
                  IQOPT-DONE IQS-RATED IQS-REF J KPD KPD-OPT KPQ
                  KPQ-OPT KPS KPS-OPT KSLIP L-S LLR LLS LM LR
                  MODE MODSIM NAT-RES OPT-MODE POLE RATED-SPD RR
                  RS SOURCE-VOLT SPD-REF SPDOPT-DONE TEST-RUN?
                  TIME TRUN-KNOWN TSAMP VDC WCD WCD-OPT WCQ
                  WCQ-OPT WCS WCS-OPT X1 X2 X3 X4 )
    CONTROL-DESIGN-RULES  :: (RULE175 RULE176 RULE177 RULE272 RULE273 RULE274
                  RULE275 RULE276 RULE292 )


  CONTROL-DESIGN-PARMS


  BASE-SPD
  ========
```

TRANSLATION :: (base speed of the drive)
PROMPT :: (What is the base speed in rad/sec (elect) for the drive?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
HELP :: ("Enter the base speed in rad / s, for the driven" :LINE "
machine. Normally, corresponding to 60 Hz supply," :LINE "the
base speed is 377 r/s.")
DEFAULT :: (377)
RANGE :: (0 1000)

CONT-OPT-KNOWN

TRANSLATION :: (controller optimization choice known)
TYPE :: YES/NO
UPDATED-BY :: (RULE275)

CONT-OPT?

TRANSLATION :: (optimize the controller?)
PROMPT :: ("The design of controller is now over. Do you want to
optimize the" :LINE "controllers?")
TYPE :: YES/NO
ANTECEDENT-IN :: (RULE272)
USED-BY :: (RULE275 RULE177)

DELN-KPD

TRANSLATION :: (δKp decrement for d-axis loop tuning)
TYPE :: SINGLEVALUED
USED-BY :: (RULE177)
METHOD :: (TIMES 0.1 (VAL1 FRAME KPD))

DELN-KPQ

TRANSLATION :: (δKpq decrement value, q-axis loop)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 0.1 (VAL1 FRAME KPQ))

DELN-KPS

TRANSLATION :: (δKps decrement, speed loop tuning)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 0.075 (VAL1 FRAME KPS))

DELN-WCD

TRANSLATION :: (δWcd decrement)
TYPE :: SINGLEVALUED
USED-BY :: (RULE177)
METHOD :: (TIMES 0.1 (VAL1 FRAME WCD))

DELN-WCQ

TRANSLATION :: (δWcq decrement, iq-loop tuning)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 0.1 (VAL1 FRAME WCQ))

DELN-WCS

TRANSLATION :: (δWcs decrement, speed loop tuning)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 0.1 (VAL1 FRAME WCS))

DELP-KPD

TRANSLATION :: (δKp value in positive direction)
TYPE :: SINGLEVALUED
USED-BY :: (RULE177)
METHOD :: (TIMES 0.2 (VAL1 FRAME KPD))

DELP-KPQ
TRANSLATION :: (δKpq increment value)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 0.2 (VAL1 FRAME KPQ))

DELP-KPS
TRANSLATION :: (δKps increment, speed loop tuning)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 0.15 (VAL1 FRAME KPS))

DELP-WCD
TRANSLATION :: (δWcd increment)
TYPE :: SINGLEVALUED
USED-BY :: (RULE177)
METHOD :: (TIMES 0.2 (VAL1 FRAME WCD))

DELP-WCQ
TRANSLATION :: (δWcq increment, iq loop opt.)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 0.2 (VAL1 FRAME WCQ))

DELP-WCS
TRANSLATION :: (δWcs increment, speed loop tuning)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 0.2 (VAL1 FRAME WCS))

IDOPT-DONE
TRANSLATION :: (optimization of d-axis current controller done)
TYPE :: YES/NO
UPDATED-IN :: (RULE208 RULE207)
ANTECEDENT-IN :: (RULE289 RULE273 RULE180 RULE184 RULE188 RULE192
RULE196 RULE201 RULE203 RULE204 RULE205)
USED-BY :: (RULE292)

IDS-RATED
TRANSLATION :: (rated ids value)
PROMPT :: (What is the rated value of ids (the d-axis current) ?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
HELP :: ("This is a machine parameter, and you need to know" :LINE "the
value. Default here for the 5 hp machine is" :LINE "9.216 Amps."
)
DEFAULT :: (9.216)
RANGE :: (0 100)

IDS-REF
TRANSLATION :: (test reference value for ids)
TYPE :: SINGLEVALUED
USED-BY :: (RULE177)
METHOD :: (FQUOTIENT (VAL1 FRAME IDS-RATED) 2)

IQOPT-DONE

```
==========
 TRANSLATION :: (optimization of q-axis controller complete)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE235 RULE236)
 ANTECEDENT-IN :: (RULE290 RULE274 RULE209 RULE213 RULE217 RULE221
          RULE225 RULE230 RULE231 RULE232 RULE233 )
 USED-BY :: (RULE292)

IQS-RATED
==========
 TRANSLATION :: (rated value of iqs current (q-axis, torque component))
 PROMPT :: ("What is the rated value of torque component iqs of stator
       current ?" )
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 HELP :: ("Please enter the torque component Iqs of the" :LINE "stator
      current. Default is 19.11 A for the 5 hp" :LINE "machine." )
 DEFAULT :: (19.11)
 RANGE :: (0 100)

IQS-REF
=======
 TRANSLATION :: (test reference square signal for iqs tuning)
 TYPE :: SINGLEVALUED
 USED-BY :: (RULE177)
 METHOD :: (FQUOTIENT (VAL1 FRAME IQS-RATED) 2)

J
=
 TRANSLATION :: (Moment of inertia, load and machine, lb-ft2)
 PROMPT :: ("What is the moment of inertia of load and machine in kg-m2 ?"
       )
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 USED-BY :: (RULE175)
 HELP :: ("The moment of inertia in kg-m2 is a necessary" :LINE "
      parameter for the design of PI gains for speed" :LINE "
      controller." )
 CONTAINED-IN :: (RULE175)
 DEFAULT :: (0.1)
 RANGE :: (0 100)

KPD
===
 TRANSLATION :: (Kp for d-axis current controller)
 TYPE :: SINGLEVALUED
 UPDATED-BY :: (RULE176)
 UPDATED-IN :: (RULE184 RULE188 RULE192 RULE196 RULE201 RULE203 RULE204
          RULE205 )
 USED-BY :: (RULE177)

KPD-OPT
=======
 TRANSLATION :: (optimum kp for the d-axis current controller)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-IN :: (RULE208 RULE207)
 RANGE :: (0 10)

KPQ
===
 TRANSLATION :: (kp for the q-axis controller)
 TYPE :: SINGLEVALUED
 UPDATED-BY :: (RULE176)
 UPDATED-IN :: (RULE208 RULE207 RULE213 RULE217 RULE221 RULE225 RULE230
```

```
        RULE231 RULE232 RULE233 )
 USED-BY :: (RULE177)

KPQ-OPT
=======
 TRANSLATION :: (optimized q-axis kp)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-IN :: (RULE235 RULE236)
 RANGE :: (0 10)

KPS
===
 TRANSLATION :: (kp for the speed controller)
 TYPE :: SINGLEVALUED
 UPDATED-BY :: (RULE175)
 UPDATED-IN :: (RULE241 RULE245 RULE249 RULE253 RULE258 RULE259 RULE260
        RULE261 )
 USED-BY :: (RULE177)

KPS-OPT
=======
 TRANSLATION :: (optimized kp for speed loop)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-IN :: (RULE263 RULE264)
 RANGE :: (0 100)

KSLIP
=====
 TRANSLATION :: (rated slip gain)
 TYPE :: SINGLEVALUED
 USED-BY :: (RULE177)
 METHOD :: (FQUOTIENT (VAL1 FRAME RR) (TIMES (VAL1 FRAME LR) (VAL1 FRAME
        IDS-RATED ) ) )

L-S
===
 TRANSLATION :: (stator inductance)
 TYPE :: SINGLEVALUED
 METHOD :: (PLUS (VAL1 FRAME LLS) (VAL1 FRAME LM))

LLR
===
 TRANSLATION :: (rotor leakage inductance)
 PROMPT :: (What is the rotor leakage inductance in Henrys?)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 HELP :: ("The rotor leakage inductance is a necessary" :LINE "quantity
       for the machine model to be simulated." :LINE "Default here is
       0.00213 H, corresponding to 5 hp" :LINE "induction machine" )
 RANGE :: (0 5)

LLS
===
 TRANSLATION :: (stator leakage inductance)
 PROMPT :: (What is the stator leakage inductance per phase ?)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 HELP :: ("The stator leakage inductance (per phase) in" :LINE "Henrys is
       a necessary quantity for the machine" :LINE "d-q axis model
       simulation. The default here is" :LINE "0.00213 Henry." )
 RANGE :: (0 5)

LM
```

==
TRANSLATION :: (magnetizing inductance for the machine, in Henry)
PROMPT :: ("What is the magnetizing inductance of the machine (per
phase) in Henry?")
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
USED-BY :: (RULE176)
HELP :: ("The machine magnetizing inductance is necessary" :LINE "both
for the simulation of machine d-q axis" :LINE "model, and for
computation of initial values of" :LINE "current controller
gains. Default value here is" :LINE "0.032 H, per phase.")
CONTAINED-IN :: (RULE176)
DEFAULT :: (0.032)
RANGE :: (0 5)

LR
==
TRANSLATION :: (rotor inductance in eq. ckt)
TYPE :: SINGLEVALUED
METHOD :: (PLUS (VAL1 FRAME LLR) (VAL1 FRAME LM))

MODE
====
TRANSLATION :: (mode of tuning, for simulator)
TYPE :: SINGLEVALUED
EXPECT :: INTEGER
UPDATED-IN :: (RULE180 RULE209 RULE237 RULE265)
RANGE :: (0 4)

MODSIM
======
TRANSLATION :: (simnon simulation mode: plant sim / data acq & disp)
TYPE :: SINGLEVALUED
EXPECT :: (SIMPLNT DACQDISP QACQDISP WACQDISP TACQDISP)
UPDATED-IN :: (RULE279 RULE226 RULE197 RULE280 RULE214 RULE218 RULE222
RULE281 RULE282 RULE283 RULE284 RULE242 RULE250 RULE254
RULE286 RULE287 RULE288 RULE210 RULE266 RULE181 RULE185
RULE189 RULE193 RULE277 RULE278 RULE246 RULE238 RULE285
RULE180 RULE184 RULE188 RULE192 RULE196 RULE201 RULE203
RULE204 RULE205 RULE209 RULE213 RULE217 RULE221 RULE225
RULE230 RULE231 RULE232 RULE233 RULE237 RULE241 RULE245
RULE249 RULE253 RULE258 RULE259 RULE260 RULE261 RULE265
)

NAT-RES
=======
TRANSLATION :: (nature of response)
TYPE :: SINGLEVALUED
EXPECT :: (OVERDAMPED UNDERDAMPED)
UPDATED-IN :: (RULE279 RULE226 RULE197 RULE280 RULE214 RULE218 RULE222
RULE281 RULE282 RULE283 RULE284 RULE242 RULE250 RULE254
RULE286 RULE287 RULE288 RULE210 RULE181 RULE185 RULE189
RULE193 RULE277 RULE278 RULE246 RULE238 RULE285)
ANTECEDENT-IN :: (RULE194 RULE198 RULE195 RULE289 RULE212 RULE215
RULE211 RULE216 RULE219 RULE220 RULE223 RULE224
RULE227 RULE290 RULE239 RULE243 RULE244 RULE247
RULE248 RULE251 RULE252 RULE255 RULE240 RULE291
RULE208 RULE236 RULE182 RULE186 RULE190 RULE183
RULE187 RULE191 RULE264 RULE184 RULE188 RULE192
RULE196 RULE201 RULE203 RULE204 RULE205 RULE213
RULE217 RULE221 RULE225 RULE230 RULE231 RULE232
RULE233 RULE241 RULE245 RULE249 RULE253 RULE258
RULE259 RULE260 RULE261)

OPT-MODE

TRANSLATION :: (mode of tuning)
TYPE :: SINGLEVALUED
EXPECT :: (D-AXIS-LOOP Q-AXIS-LOOP SPEED-LOOP TEST-RUN)
UPDATED-IN :: (RULE272 RULE273 RULE274 RULE276)
ANTECEDENT-IN :: (RULE180 RULE209 RULE237 RULE265)

POLE

TRANSLATION :: (machine number of poles)
PROMPT :: (What is the number of poles of the driven machine ?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
HELP :: (Default here is 4.)
RANGE :: (0 50)

RATED-SPD

TRANSLATION :: (rated speed, rad/s)
PROMPT :: (What is the rated speed of machine in elect. rad/s ?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
HELP :: (Default here is 200 r/s.)
DEFAULT :: (377)
RANGE :: (0 1000)

RR

TRANSLATION :: (rotor resistance)
PROMPT :: (What is the rotor resistance in ohms?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
HELP :: (Default value per phase for the machine is 0.318Ω)
RANGE :: (0 10)

RS

TRANSLATION :: (stator resistance)
PROMPT :: (What is the stator resistance of the machine (per phase) ?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
HELP :: (The default value per phase is 0.37Ω)
DEFAULT :: (0.37)
RANGE :: (0 10)

SOURCE-VOLT

TRANSLATION :: (Input voltage of the system)
PROMPT :: (What is the input voltage of the system ?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
HELP :: (The default here is 220 V)
RANGE :: (100 600)

SPD-REF

TRANSLATION :: (test reference square signal for speed loop tuning)
TYPE :: SINGLEVALUED
USED-BY :: (RULE177)
METHOD :: (FQUOTIENT (VAL1 FRAME RATED-SPD) 20)

SPDOPT-DONE

TRANSLATION :: (speed loop optimization done)
TYPE :: YES/NO

UPDATED-IN :: (RULE263 RULE264)
ANTECEDENT-IN :: (RULE291 RULE276 RULE237 RULE241 RULE245 RULE249
RULE253 RULE258 RULE259 RULE260 RULE261)
USED-BY :: (RULE292)

TEST-RUN?
=========
TRANSLATION :: (Flag for test-running the control system)
PROMPT :: ("The optimization of complete IVC-IM drive control system is
complete. Do you" :LINE "now want to review the responses of
all the controllers by test-running the" :LINE "system?")
TYPE :: YES/NO
ANTECEDENT-IN :: (RULE293 RULE265)
USED-BY :: (RULE292)

TIME
====
TRANSLATION :: (real-time info)
TYPE :: SINGLEVALUED
EXPECT :: NUMBER
UPDATED-IN :: (RULE180 RULE265)
RANGE :: (0 100)

TRUN-KNOWN
==========
TRANSLATION :: (User choice for test-running the system is known)
TYPE :: YES/NO
METHOD :: (VAL1 FRAME TEST-RUN? () R)

TSAMP
=====
TRANSLATION :: (sampling time, inner loop)
PROMPT :: (What is the inner loop sampling interval in secs?)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
HELP :: (The default value is 200 μs)
RANGE :: (0 1)
DEFAULT :: (2.e-4)

VDC
===
TRANSLATION :: (DC link voltage of inverter)
TYPE :: SINGLEVALUED
METHOD :: (TIMES 1.4142 (VAL1 FRAME SOURCE-VOLT))

WCD
===
TRANSLATION :: (wc for the d-axis controller)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE176)
UPDATED-IN :: (RULE184 RULE188 RULE192 RULE196 RULE201 RULE203 RULE204
RULE205)
USED-BY :: (RULE177)

WCD-OPT
=======
TRANSLATION :: (optimized wc for d-axis current controller)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE208 RULE207)
RANGE :: (0 10)

WCQ
===
TRANSLATION :: (wc for the q-axis controller)

TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE176)
UPDATED-IN :: (RULE208 RULE207 RULE213 RULE217 RULE221 RULE225 RULE230
RULE231 RULE232 RULE233)
USED-BY :: (RULE177)

WCQ-OPT
=======
TRANSLATION :: (optimized q-axis controller wc)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE235 RULE236)
RANGE :: (0 10)

WCS
===
TRANSLATION :: (wc for the speed controller)
TYPE :: SINGLEVALUED
UPDATED-BY :: (RULE175)
UPDATED-IN :: (RULE241 RULE245 RULE249 RULE253 RULE258 RULE259 RULE260
RULE261)
USED-BY :: (RULE177)

WCS-OPT
=======
TRANSLATION :: (optimized wc for speed loop)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE263 RULE264)
RANGE :: (0 10)

X1
==
TRANSLATION :: (first qty read from o/p of trscan)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE279 RULE226 RULE197 RULE280 RULE214 RULE218 RULE222
RULE281 RULE282 RULE283 RULE284 RULE242 RULE250 RULE254
RULE286 RULE287 RULE288 RULE210 RULE181 RULE185 RULE189
RULE193 RULE277 RULE278 RULE246 RULE238 RULE285)

X2
==
TRANSLATION :: (second qty read from o/p of trscan)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE279 RULE226 RULE197 RULE280 RULE214 RULE218 RULE222
RULE281 RULE282 RULE283 RULE284 RULE242 RULE250 RULE254
RULE286 RULE287 RULE288 RULE210 RULE181 RULE185 RULE189
RULE193 RULE277 RULE278 RULE246 RULE238 RULE285)

X3
==
TRANSLATION :: (third qty read from o/p of trscan)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE279 RULE226 RULE197 RULE280 RULE214 RULE218 RULE222
RULE281 RULE282 RULE283 RULE284 RULE242 RULE250 RULE254
RULE286 RULE287 RULE288 RULE210 RULE181 RULE185 RULE189
RULE193 RULE277 RULE278 RULE246 RULE238 RULE285)

X4
==
TRANSLATION :: (fourth qty read from output of trscan.exe)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE279 RULE226 RULE197 RULE280 RULE214 RULE218 RULE222
RULE281 RULE282 RULE283 RULE284 RULE242 RULE250 RULE254
RULE286 RULE287 RULE288 RULE210 RULE181 RULE185 RULE189

RULE193 RULE277 RULE278 RULE246 RULE238 RULE285)

CONTROL-DESIGN-RULES

RULE175

SUBJECT :: CONTROL-DESIGN-RULES
DESCRIPTION :: (assigns initial kp and wc for speed loop)
IF :: (J IS KNOWN)
THEN :: (KPS = (80 * J) AND WCS = 1.)

RULE176

SUBJECT :: CONTROL-DESIGN-RULES
DESCRIPTION :: (prel vals of kpd, wcd, kpq and wcq)
IF :: (LM IS KNOWN)
THEN :: (KPD = (3. * LM) AND WCD = 1. AND KPQ = (3. * LM) AND WCQ = 1.)

RULE177

SUBJECT :: CONTROL-DESIGN-RULES
DESCRIPTION :: (controller design done)
IF :: (KPD IS KNOWN AND WCD IS KNOWN AND KPQ IS KNOWN AND WCQ IS KNOWN AND KPS IS KNOWN AND WCS IS KNOWN AND DELP-KPD IS KNOWN AND DELP-WCD IS KNOWN AND DELN-KPD IS KNOWN AND DELN-WCD IS KNOWN AND KSLIP IS KNOWN AND IDS-REF IS KNOWN AND IQS-REF IS KNOWN AND SPD-REF IS KNOWN AND CONT-OPT? IS KNOWN)
THEN :: (CONT-DES-DONE)

RULE272

SUBJECT :: CONTROL-DESIGN-RULES
ANTECEDENT :: YES
UTILITY :: 100
DESCRIPTION :: (Invoke Id-loop optimization)
IF :: (CONT-OPT?)
THEN :: (OPT-MODE = D-AXIS-LOOP AND CONSIDERFRAME IDS-TUNING)

RULE273

SUBJECT :: CONTROL-DESIGN-RULES
ANTECEDENT :: YES
UTILITY :: 80
DESCRIPTION :: (invoke iq-loop tuning subframe)
IF :: (IDOPT-DONE)
THEN :: (DONTCONSIDERFRAME IDS-TUNING AND OPT-MODE = Q-AXIS-LOOP AND CONSIDERFRAME IQS-TUNING)

RULE274

SUBJECT :: CONTROL-DESIGN-RULES
ANTECEDENT :: YES
UTILITY :: 70
DESCRIPTION :: (invoke speed loop tuning)
IF :: (IQOPT-DONE)
THEN :: (DONTCONSIDERFRAME IQS-TUNING AND OPT-MODE = SPEED-LOOP AND CONSIDERFRAME SPD-TUNING AND CONSIDERRULE FRAME RULE237)

RULE275

SUBJECT :: CONTROL-DESIGN-RULES
DESCRIPTION :: (flag set when controller optimization chosen)

IF :: (CONT-OPT?)
THEN :: (CONT-OPT-KNOWN)

RULE276
=======
SUBJECT :: CONTROL-DESIGN-RULES
ANTECEDENT :: YES
DESCRIPTION :: (Invokes test-running of complete system)
IF :: (SPDOPT-DONE)
THEN :: (DONTCONSIDERFRAME SPD-TUNING AND OPT-MODE = TEST-RUN AND
CONSIDERFRAME TEST-RUN AND CONSIDERRULE FRAME RULE265)

RULE292
=======
SUBJECT :: CONTROL-DESIGN-RULES
DESCRIPTION :: (Controller optimization done)
IF :: (IDOPT-DONE AND IQOPT-DONE AND SPDOPT-DONE AND TEST-RUN? IS
KNOWN AND TEST-RUN-DONE)
THEN :: (CONT-OPT-DONE)

Frame :: IDS-TUNING

IDENTIFIER :: IDS-TUNING-
TRANSLATION :: (d-axis current loop tuning performed here)
PARENTS :: (CONTROL-DESIGN AC-DRIVE-SYSTEM-DESIGN)
GOALS :: (WCD-OPT KPD-OPT IDOPT-DONE)
INITIALDATA :: (TSAMP POLE LLS LLR RR RS BASE-SPD SOURCE-VOLT IDS-RATED
DELP-KPD DELN-KPD DELP-WCD DELN-WCD)
PROMPTEVER :: (:LINE "The process of optimization of controllers begins
now. There are three" :LINE "controllers to be optimized
here, and the optimization is performed in three" :LINE "
stages:" :LINE 2 "Firstly, the d-axis controller is
optimized, with torque command Iq = 0, and" :LINE "the
speed of the machine, therefore, is also zero." :LINE 2 "

The d-axis controller is supplied with a unipolar square-wave test" :LINE "

current signal, with the amplitude of 50% rated value. The simulation is"
:LINE "performed in the TMS 320C25 simulator, and the
results are displayed" :LINE "graphically at the end of
each stage. The expert system varies the gains" :LINE "such
that the response profile approaches to slightly
underdamped response," :LINE "with very small overshoot."
:LINE 2 "The q-axis controller is optimized next, with
d-axis controller gains set to" :LINE "their optimized
values. Since there is not much difference in the
electrical" :LINE "circuit presented to the q-axis
controller, from that presented to the" :LINE "d-axis
controller. the optimized d-axis gains are used as the
starting" :LINE "values for q-axis controller gains, to
reduce the simulation time. The" :LINE "optimization now
proceeds, with the machine accelerating slightly, due to"
:LINE "the non-zero Iq command. The optimization proceeds
in a manner similar to" :LINE "that of d-axis controller."
:LINE 2 "Once the inner loops are tuned. the optimization
process continues with the" :LINE "outer. slower, speed
loop tuning. The inner loop gains are set to their" :LINE "
optimum values, and a speed reference square signal command
is given to the" :LINE "speed controller. The speed
controller is tuned in the same manner as the" :LINE "
current controllers." :LINE 2 "Once the optimization
process is over, you will be presented with a choice"
:LINE "to test-run the system, to observe the complete
system behavior, with the" :LINE "expert system running a
trial simulation with complete system." :LINE 2 "The
optimization of d-axis controller will now proceed. Please
wait for the" :LINE "results of the simulation pass...")
PARMGROUP :: IDS-TUNING-PARMS
RULEGROUPS :: (IDS-TUNING-RULES)
IDS-TUNING-PARMS :: (CD1 CD2 CD3 CD4 CD5 CD6 CD7 CD8 DRESP-ACQ1
DRESP-ACQ2 DRESP-ACQ3 DRESP-ACQ4 DRESP-ACQ5
DRESP-ACQ6 DRESP-ACQ7 DRESP-ACQ8 DRESP-ACQ9
DRESP-OS DTAU DTR DTS F-IDSIM-DONE FCONT-SIM-DONE
IDREF-CUR IDREF-PRE ODRESP-READ1 ODRESP-READ2
ODRESP-READ3 ODRESP-READ4 ODRESP-READ5 STOP-DEC-KPD
STOP-INC-KPD UDRESP-READ1 UDRESP-READ2 UDRESP-READ3
UDRESP-READ4 UDRESP-READ5)
IDS-TUNING-RULES :: (RULE180 RULE181 RULE182 RULE183 RULE184 RULE185
RULE186 RULE187 RULE188 RULE189 RULE190 RULE191
RULE192 RULE193 RULE194 RULE195 RULE196 RULE197

RULE198 RULE200 RULE201 RULE203 RULE204 RULE205
RULE206 RULE207 RULE208 RULE277 RULE278 RULE279
RULE280 RULE289)

IDS-TUNING-PARMS

CD1
===
TRANSLATION :: (flag, incremented kpd iteration)
TYPE :: YES/NO
UPDATED-IN :: (RULE184)
ANTECEDENT-IN :: (RULE200 RULE197 RULE185 RULE189 RULE193)

CD2
===
TRANSLATION :: (flag, incremented kpd iteration 2)
TYPE :: YES/NO
UPDATED-IN :: (RULE188)
ANTECEDENT-IN :: (RULE200 RULE197 RULE189 RULE193)

CD3
===
TRANSLATION :: (flag3, incremented kpd iteration)
TYPE :: YES/NO
UPDATED-IN :: (RULE192)
ANTECEDENT-IN :: (RULE200 RULE197 RULE193)

CD4
===
TRANSLATION :: (flag4, incremented kpd iteration)
TYPE :: YES/NO
UPDATED-IN :: (RULE196)
ANTECEDENT-IN :: (RULE200 RULE197)

CD5
===
TRANSLATION :: (flag5, kpd decremented iteration)
TYPE :: YES/NO
UPDATED-IN :: (RULE201)
ANTECEDENT-IN :: (RULE279 RULE280 RULE206 RULE277 RULE278)

CD6
===
TRANSLATION :: (flag6, kpd decr iter)
TYPE :: YES/NO
UPDATED-IN :: (RULE203)
ANTECEDENT-IN :: (RULE279 RULE280 RULE206 RULE278)

CD7
===
TRANSLATION :: (flag7, kpd decr iter)
TYPE :: YES/NO
UPDATED-IN :: (RULE204)
ANTECEDENT-IN :: (RULE279 RULE280 RULE206)

CD8
===
TRANSLATION :: (flag8, kpd decr iter)
TYPE :: YES/NO
UPDATED-IN :: (RULE205)
ANTECEDENT-IN :: (RULE280 RULE206)

DRESP-ACQ1
==========
TRANSLATION :: (first d-axis controller resp acquired)
TYPE :: YES/NO
UPDATED-IN :: (RULE181)
ANTECEDENT-IN :: (RULE182 RULE183)

DRESP-ACQ2
==========
TRANSLATION :: (id cont resp acq'd, case 2)
TYPE :: YES/NO
UPDATED-IN :: (RULE185)
ANTECEDENT-IN :: (RULE186 RULE183)

DRESP-ACQ3
==========
TRANSLATION :: (case 3 d-cont resp acquired)
TYPE :: YES/NO
UPDATED-IN :: (RULE189)
ANTECEDENT-IN :: (RULE190 RULE183)

DRESP-ACQ4
==========
TRANSLATION :: (id cont resp acq'd, sim pass #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE193)
ANTECEDENT-IN :: (RULE194 RULE183)

DRESP-ACQ5
==========
TRANSLATION :: (id cont resp disp and acq'd, pass 5)
TYPE :: YES/NO
UPDATED-IN :: (RULE197)
ANTECEDENT-IN :: (RULE198 RULE183)

DRESP-ACQ6
==========
TRANSLATION :: (d-axis cont resp acquired, underdamped case)
TYPE :: YES/NO
UPDATED-IN :: (RULE277)
ANTECEDENT-IN :: (RULE187)

DRESP-ACQ7
==========
TRANSLATION :: (d-axis cont resp acquired, und. #2)
TYPE :: YES/NO
UPDATED-IN :: (RULE278)
ANTECEDENT-IN :: (RULE191)

DRESP-ACQ8
==========
TRANSLATION :: (d axis cont underdamped resp read, pass #3)
TYPE :: YES/NO
UPDATED-IN :: (RULE279)
ANTECEDENT-IN :: (RULE195)

DRESP-ACQ9
==========
TRANSLATION :: (d-cont und resp read, pass 4)
TYPE :: YES/NO
UPDATED-IN :: (RULE280)
ANTECEDENT-IN :: (RULE289)

DRESP-OS
========

TRANSLATION :: (perc. overshoot of response of d-axis controller)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE195 RULE289 RULE183 RULE187 RULE191)
ANTECEDENT-IN :: (RULE208)
RANGE :: (0 100)

DTAU
====
TRANSLATION :: (time const, overdamped resp)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE194 RULE198 RULE182 RULE186 RULE190)
RANGE :: (0 50)

DTR
===
TRANSLATION :: (rise time of d-axis current controller response)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE195 RULE289 RULE183 RULE187 RULE191)
RANGE :: (0 50)

DTS
===
TRANSLATION :: (settling time, overdamped resp)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE194 RULE198 RULE195 RULE289 RULE182 RULE186 RULE190
RULE183 RULE187 RULE191)
RANGE :: (0 50)

F-IDSIM-DONE
============
TRANSLATION :: (first id loop simulation done)
TYPE :: YES/NO
UPDATED-IN :: (RULE180)
ANTECEDENT-IN :: (RULE197 RULE181 RULE185 RULE189 RULE193)

FCONT-SIM-DONE
==============
TRANSLATION :: (first id controller simulation done)
TYPE :: YES/NO

IDREF-CUR
=========
TRANSLATION :: (current value of ids ref)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE180)

IDREF-PRE
=========
TRANSLATION :: (previous value of ids ref)
TYPE :: SINGLEVALUED

ODRESP-READ1
============
TRANSLATION :: (response read, overdamped case 1)
TYPE :: YES/NO
UPDATED-IN :: (RULE182)
ANTECEDENT-IN :: (RULE184)

ODRESP-READ2
============
TRANSLATION :: (overdamped resp read case2)

TYPE :: YES/NO
UPDATED-IN :: (RULE186)
ANTECEDENT-IN :: (RULE188)

ODRESP-READ3
==========
TRANSLATION :: (Overdamped resp parms read, pass 3)
TYPE :: YES/NO
UPDATED-IN :: (RULE190)
ANTECEDENT-IN :: (RULE192)

ODRESP-READ4
==========
TRANSLATION :: (id cont resp read, overdamped case, pass #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE194)
ANTECEDENT-IN :: (RULE196)

ODRESP-READ5
==========
TRANSLATION :: (overdamped resp dat read, pass #5)
TYPE :: YES/NO
UPDATED-IN :: (RULE198)
ANTECEDENT-IN :: (RULE200)

STOP-DEC-KPD
==========
TRANSLATION :: (stop decrementing kpd, wcd)
TYPE :: YES/NO
UPDATED-IN :: (RULE206)
ANTECEDENT-IN :: (RULE207)

STOP-INC-KPD
==========
TRANSLATION :: (stop incrementing kpd, wcd)
TYPE :: YES/NO
UPDATED-IN :: (RULE200)
ANTECEDENT-IN :: (RULE207)

UDRESP-READ1
==========
TRANSLATION :: (underdamped d-axis resp data read, case 1)
TYPE :: YES/NO
UPDATED-IN :: (RULE183)
ANTECEDENT-IN :: (RULE184 RULE201)

UDRESP-READ2
==========
TRANSLATION :: (underdamped resp, id controller read case 2)
TYPE :: YES/NO
UPDATED-IN :: (RULE187)
ANTECEDENT-IN :: (RULE188 RULE203)

UDRESP-READ3
==========
TRANSLATION :: (underdamped resp parms read, pass 3)
TYPE :: YES/NO
UPDATED-IN :: (RULE191)
ANTECEDENT-IN :: (RULE192 RULE204)

UDRESP-READ4
==========
TRANSLATION :: (underdamped resp, id cont, pass #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE195)

```
 ANTECEDENT-IN :: (RULE196 RULE205)

UDRESP-READ5
============
 TRANSLATION :: (underdamped resp dat read, pass 5)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE289)
 ANTECEDENT-IN :: (RULE206)


 ======================
   IDS-TUNING-RULES
 ======================

RULE180
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 99
 DESCRIPTION :: (performs first id loop simulation, sets flag)
 IF  :: (OPT-MODE = D-AXIS-LOOP AND ! IDOPT-DONE)
 THEN :: (DOS-CALL "crecst.exe" "" AND MODSIM = "SIMPLNT" AND TIME = 0
       AND IDREF-CUR = 0 AND WRITE-DOS-FILE "timeinfo.dat" TIME AND
       MODE = 1 AND WRITE-DOS-FILE "mode.dat" MODE AND WRITE-DOS-FILE "
       conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND WRITE-DOS-FILE "
       param.dat" IDS-REF IQS-REF SPD-REF VDC BASE-SPD RS RR LR L-S
       LM J POLE AND WRITE-DOS-FILE "tstsig.dat" IDREF-CUR AND
       DOS-CALL "par2hexn.exe" "" AND DOS-CALL "asc2get.exe" "" AND
       WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "bfgen.exe" ""
       AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "log2ini.exe" ""
       AND DOS-CALL "" "infsim" AND F-IDSIM-DONE )

RULE181
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 98
 DESCRIPTION :: (acquires first response data through simnon)
 IF  :: (F-IDSIM-DONE)
 THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
       AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
       DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
       READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
       respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
       DRESP-ACQ1 )

RULE182
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 97
 DESCRIPTION :: (resp read, overdamped case, pass 1)
 IF  :: (DRESP-ACQ1 AND NAT-RES = OVERDAMPED)
 THEN :: (DTAU = (E (* 1 (VAL1 FRAME X1))) AND DTS = (E (* 1 (VAL1 FRAME
       X2 ) ) ) AND ODRESP-READ1 )

RULE183
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 88
 DESCRIPTION :: (reads resp specs for underdamped resp, case 1)
 IF  :: ((DRESP-ACQ1 OR DRESP-ACQ2 OR DRESP-ACQ3 OR DRESP-ACQ4 OR
       DRESP-ACQ5 ) AND NAT-RES = UNDERDAMPED )
 THEN :: (DRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND DTR = (E (* 1 (VAL1
```

```
      FRAME X3 ) ) ) AND DTS = (E (* 1 (VAL1 FRAME X4))) AND
      UDRESP-READ1 )

RULE184
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 96
 DESCRIPTION :: (simulation pass #2 with first modified kpd, wcd,
          overdamped case )
 IF  :: (NAT-RES = OVERDAMPED AND (ODRESP-READ1 OR UDRESP-READ1) AND !
      IDOPT-DONE )
 THEN :: (KPD = (E (+ (VAL1 FRAME DELP-KPD) (VAL1 FRAME KPD))) AND WCD =
      (E (+ (VAL1 FRAME DELP-WCD) (VAL1 FRAME WCD)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT "The response of id controller is overdamped. Both Kp and
      Ki are modified and the simulation will now proceed with these
      new values" "" AND DOS-CALL "" "infsim" AND CD1 )

RULE185
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 95
 DESCRIPTION :: (ids controller resp acquired, case 2)
 IF  :: (F-IDSIM-DONE AND CD1)
 THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      DRESP-ACQ2 )

RULE186
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 94
 DESCRIPTION :: (reads overdamped resp params, case 2)
 IF  :: (DRESP-ACQ2 AND NAT-RES = OVERDAMPED)
 THEN :: (DTAU = (E (* 1 (VAL1 FRAME X1))) AND DTS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND ODRESP-READ2 )

RULE187
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 85
 DESCRIPTION :: (case 2 underdamped resp parms read)
 IF  :: (DRESP-ACQ6 AND NAT-RES = UNDERDAMPED)
 THEN :: (DRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND DTR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND DTS = (E (* 1 (VAL1 FRAME X4))) AND
      UDRESP-READ2 )

RULE188
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 93
 DESCRIPTION :: (kpd incremented second time, simulation pass 3)
 IF  :: (NAT-RES = OVERDAMPED AND (ODRESP-READ2 OR UDRESP-READ2) AND !
      IDOPT-DONE )
```

```
 THEN :: (KPD = (E (+ (VAL1 FRAME DELP-KPD) (VAL1 FRAME KPD))) AND WCD =
      (E (+ (VAL1 FRAME DELP-WCD) (VAL1 FRAME WCD)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT " The response is overdamped for Ids controller, so the
      simulation will now resume with the gains Kp and Ki modified
      .... " " " AND DOS-CALL "" "infsim" AND CD2 )

RULE189
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 92
 DESCRIPTION :: (response data passed to ES via simnon, case 3)
 IF  :: (F-IDSIM-DONE AND CD1 AND CD2)
 THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      DRESP-ACQ3 )

RULE190
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 91
 DESCRIPTION :: (overdamped resp parms read, pass 3)
 IF  :: (DRESP-ACQ3 AND NAT-RES = OVERDAMPED)
 THEN :: (DTAU = (E (* 1 (VAL1 FRAME X1))) AND DTS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND ODRESP-READ3 )

RULE191
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 82
 DESCRIPTION :: (underdamped resp parms read, pass #3)
 IF  :: (DRESP-ACQ7 AND NAT-RES = UNDERDAMPED)
 THEN :: (DRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND DTR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND DTS = (E (* 1 (VAL1 FRAME X4))) AND
      UDRESP-READ3 )

RULE192
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 90
 DESCRIPTION :: (simulation pass 4 overdamped case, incremented kpd)
 IF  :: (NAT-RES = OVERDAMPED AND (ODRESP-READ3 OR UDRESP-READ3) AND !
      IDOPT-DONE )
 THEN :: (KPD = (E (+ (VAL1 FRAME DELP-KPD) (VAL1 FRAME KPD))) AND WCD =
      (E (+ (VAL1 FRAME DELP-WCD) (VAL1 FRAME WCD)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT " The response of Ids controller is still overdamped, so
      the simulation will continue with the new Kp and Ki .. " " "
      AND DOS-CALL "" "infsim" AND CD3 )

RULE193
=======
```

```
SUBJECT :: IDS-TUNING-RULES
ANTECEDENT :: YES
UTILITY :: 89
DESCRIPTION :: (d controller resp passed to ES via simnon, pass #4)
IF  :: (F-IDSIM-DONE AND CD1 AND CD2 AND CD3)
THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
     AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
     READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
     respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
     DRESP-ACQ4 )

RULE194
=======
SUBJECT :: IDS-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (d cont resp read, overdamped case, pass #4)
IF  :: (DRESP-ACQ4 AND NAT-RES = OVERDAMPED)
THEN :: (DTAU = (E (* 1 (VAL1 FRAME X1))) AND DTS = (E (* 1 (VAL1 FRAME
     X2 ) ) ) AND ODRESP-READ4 )

RULE195
=======
SUBJECT :: IDS-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (resp acqd, underdamped case, pass #4)
IF  :: (DRESP-ACQ8 AND NAT-RES = UNDERDAMPED)
THEN :: (DRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND DTR = (E (* 1 (VAL1
     FRAME X3 ) ) ) AND DTS = (E (* 1 (VAL1 FRAME X4))) AND
     UDRESP-READ4 )

RULE196
=======
SUBJECT :: IDS-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (incremented kpd, simulation pass #5)
IF  :: (NAT-RES = OVERDAMPED AND (ODRESP-READ4 OR UDRESP-READ4) AND !
     IDOPT-DONE )
THEN :: (KPD = (E (+ (VAL1 FRAME DELP-KPD) (VAL1 FRAME KPD))) AND WCD =
     (E (+ (VAL1 FRAME DELP-WCD) (VAL1 FRAME WCD)) ) AND
     WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND
     DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
     MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
     DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
     PRINT " The response is overdamped, so re-simulating with
     modified Kp and Ki for Ids loop controllers... " " " AND
     DOS-CALL "" "infsim" AND CD4 )

RULE197
=======
SUBJECT :: IDS-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (d cont resp disp and acq'd, pass 5)
IF  :: (F-IDSIM-DONE AND CD1 AND CD2 AND CD3 AND CD4)
THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
     AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
     READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
     respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
     DRESP-ACQ5 )

RULE198
=======
SUBJECT :: IDS-TUNING-RULES
ANTECEDENT :: YES
```

```
 DESCRIPTION :: (overdamped resp dat read, pass #5)
 IF  :: (DRESP-ACQ5 AND NAT-RES = OVERDAMPED)
 THEN :: (DTAU = (E (* 1 (VAL1 FRAME X1))) AND DTS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND ODRESP-READ5 )

RULE200
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (stop incrementing kpd at the end of 4 passes)
 IF  :: (CD1 AND CD2 AND CD3 AND CD4 AND ODRESP-READ5)
 THEN :: (STOP-INC-KPD)

RULE201
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 87
 DESCRIPTION :: (first decremented kpd pass of simulation)
 IF  :: (NAT-RES = UNDERDAMPED AND UDRESP-READ1 AND ! IDOPT-DONE)
 THEN :: (KPD = (E (- (VAL1 FRAME KPD) (VAL1 FRAME DELN-KPD))) AND WCD =
      (E (- (VAL1 FRAME WCD) (VAL1 FRAME DELN-WCD)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT "The response is more underdamped than necessary. So, Kp
      and Ki are varied towards their optimum value, and the
      simulation will begin again.." " " AND DOS-CALL "" "infsim"
      AND CD5 )

RULE203
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 84
 DESCRIPTION :: (second decremented kpd iteration)
 IF  :: (NAT-RES = UNDERDAMPED AND UDRESP-READ2 AND ! IDOPT-DONE)
 THEN :: (KPD = (E (- (VAL1 FRAME KPD) (VAL1 FRAME DELN-KPD))) AND WCD =
      (E (- (VAL1 FRAME WCD) (VAL1 FRAME DELN-WCD)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT " The response is more underdamped than desired. The
      simulation will now continue with Kp and Ki modified .. " " "
      AND DOS-CALL "" "infsim" AND CD6 )

RULE204
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 81
 DESCRIPTION :: (third decremented kpd iteration)
 IF  :: (NAT-RES = UNDERDAMPED AND UDRESP-READ3 AND ! IDOPT-DONE)
 THEN :: (KPD = (E (- (VAL1 FRAME KPD) (VAL1 FRAME DELN-KPD))) AND WCD =
      (E (- (VAL1 FRAME WCD) (VAL1 FRAME DELN-WCD)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT "The response is still more underdamped than necessary.
      Re-simulating with Kp and Ki changed to achieve desired
      response .. " "" :LINE 5 AND DOS-CALL "" "infsim" AND CD7 )
```

```
RULE205
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (fourth decremented kpd iteration)
 IF  :: (NAT-RES = UNDERDAMPED AND UDRESP-READ4 AND ! IDOPT-DONE)
 THEN :: (KPD = (E (- (VAL1 FRAME KPD) (VAL1 FRAME DELN-KPD))) AND WCD =
      (E (- (VAL1 FRAME WCD) (VAL1 FRAME DELN-WCD)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD WCD TSAMP IDS-REF KSLIP AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT "Re-simulating with modified Kp and Ki to achieve desired
      response.." " " AND DOS-CALL "" "infsim" AND CD8 )

RULE206
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (stop decrementing kpd at the end of four iterations)
 IF  :: (CD5 AND CD6 AND CD7 AND CD8 AND UDRESP-READ5)
 THEN :: (STOP-DEC-KPD)

RULE207
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (assign opt kpd if # iter > high limit)
 IF  :: (STOP-DEC-KPD OR STOP-INC-KPD)
 THEN :: (KPD-OPT = (E (* 1 (VAL1 FRAME KPD))) AND WCD-OPT = (E (* 1 (
      VAL1 FRAME WCD ) ) ) AND KPQ = (E (* 1 (VAL1 FRAME KPD))) AND
      WCQ = (E (* 1 (VAL1 FRAME WCD))) AND IDOPT-DONE )

RULE208
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (assign optimum kpd / kid when resp. specs are
          acceptable )
 IF  :: (NAT-RES = UNDERDAMPED AND DRESP-OS < 25)
 THEN :: (KPD-OPT = (E (* 1 (VAL1 FRAME KPD))) AND WCD-OPT = (E (* 1 (
      VAL1 FRAME WCD ) ) ) AND KPQ = (E (* 1 (VAL1 FRAME KPD-OPT)))
      AND WCQ = (E (* 1 (VAL1 FRAME WCD-OPT))) AND IDOPT-DONE )

RULE277
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 86
 DESCRIPTION :: (acquires d-axis cont underdamped response)
 IF  :: (CD5)
 THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      DRESP-ACQ6 )

RULE278
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 UTILITY :: 83
 DESCRIPTION :: (D-cont underdamped resp acquired, pass 2)
 IF  :: (CD5 AND CD6)
```

```
 THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      DRESP-ACQ7 )

RULE279
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (d-cont underdamped resp read, pass#3)
 IF  :: (CD5 AND CD6 AND CD7)
 THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      DRESP-ACQ8 )

RULE280
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (d axis underdamped resp read, pass 4)
 IF  :: (CD5 AND CD6 AND CD7 AND CD8)
 THEN :: (MODSIM = "DACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      DRESP-ACQ9 )

RULE289
=======
 SUBJECT :: IDS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (Underdamped response read, #5)
 IF  :: (NAT-RES = UNDERDAMPED AND DRESP-ACQ9 AND ! IDOPT-DONE)
 THEN :: (DRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND DTR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND DTS = (E (* 1 (VAL1 FRAME X4))) AND
      UDRESP-READ5 )
```

```
Frame :: IQS-TUNING

IDENTIFIER :: "IQS-TUNING-"
TRANSLATION :: (q-axis current loop tuning)
PARENTS :: (CONTROL-DESIGN AC-DRIVE-SYSTEM-DESIGN)
GOALS :: (KPQ-OPT WCQ-OPT IQOPT-DONE)
INITIALDATA :: (DELP-KPQ DELN-KPQ DELP-WCQ DELN-WCQ)
PROMPTEVER :: ("Q axis current loop tuning in progress.. Please wait for
        the simulation" :LINE "results.." )
PARMGROUP :: IQS-TUNING-PARMS
RULEGROUPS :: (IQS-TUNING-RULES)
ANTECEDENT :: YES
IQS-TUNING-PARMS  :: (CQ1 CQ2 CQ3 CQ4 CQ5 CQ6 CQ7 CQ8 F-IQSIM-DONE
            IQREF-CUR IQREF-PRE OQRESP-READ1 OQRESP-READ2
            OQRESP-READ3 OQRESP-READ4 OQRESP-READ5 QRESP-ACQ1
            QRESP-ACQ2 QRESP-ACQ3 QRESP-ACQ4 QRESP-ACQ5
            QRESP-ACQ6 QRESP-ACQ7 QRESP-ACQ8 QRESP-ACQ9
            QRESP-OS QTAU QTR QTS STOP-DEC-KPQ STOP-INC-KPQ
            UQRESP-READ1 UQRESP-READ2 UQRESP-READ3 UQRESP-READ4
            UQRESP-READ5 )
IQS-TUNING-RULES  :: (RULE209 RULE210 RULE211 RULE212 RULE213 RULE214
            RULE215 RULE216 RULE217 RULE218 RULE219 RULE220
            RULE221 RULE222 RULE223 RULE224 RULE225 RULE226
            RULE227 RULE229 RULE230 RULE231 RULE232 RULE233
            RULE234 RULE235 RULE236 RULE281 RULE282 RULE283
            RULE284 RULE290 )


  IQS-TUNING-PARMS


CQ1
===
 TRANSLATION :: (q-axis cont simulation done, pass 2)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE213)
 ANTECEDENT-IN :: (RULE229 RULE226 RULE214 RULE218 RULE222)

CQ2
===
 TRANSLATION :: (q-axis controller simulation pass #3 done)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE217)
 ANTECEDENT-IN :: (RULE229 RULE226 RULE218 RULE222)

CQ3
===
 TRANSLATION :: (iteration 3 q-axis cont simulation, kp inc'd)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE221)
 ANTECEDENT-IN :: (RULE229 RULE226 RULE222)

CQ4
===
 TRANSLATION :: (q-axis cont simulation with inc. kp, #4)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE225)
 ANTECEDENT-IN :: (RULE229 RULE226)
```

CQ5
===
TRANSLATION :: (cont simulation done, first iter. dec. kpq)
TYPE :: YES/NO
UPDATED-IN :: (RULE230)
ANTECEDENT-IN :: (RULE234 RULE281 RULE282 RULE283 RULE284)

CQ6
===
TRANSLATION :: (kpq-decremented simulation #2 done)
TYPE :: YES/NO
UPDATED-IN :: (RULE231)
ANTECEDENT-IN :: (RULE234 RULE282 RULE283 RULE284)

CQ7
===
TRANSLATION :: (#3, kpq-decremented controller simulation)
TYPE :: YES/NO
UPDATED-IN :: (RULE232)
ANTECEDENT-IN :: (RULE234 RULE283 RULE284)

CQ8
===
TRANSLATION :: (#4, kpq-decremented simulation done)
TYPE :: YES/NO
UPDATED-IN :: (RULE233)
ANTECEDENT-IN :: (RULE234 RULE284)

F-IQSIM-DONE
==========
TRANSLATION :: (first q-axis controller simulation done)
TYPE :: YES/NO
UPDATED-IN :: (RULE209)
ANTECEDENT-IN :: (RULE226 RULE214 RULE218 RULE222 RULE210)

IQREF-CUR
=======
TRANSLATION :: (current val of iq ref test signal)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE209)

IQREF-PRE
========
TRANSLATION :: (previous val of iq ref test signal)
TYPE :: SINGLEVALUED

OQRESP-READ1
==========
TRANSLATION :: (q-axis cont resp read by ES, pass #1)
TYPE :: YES/NO
UPDATED-IN :: (RULE211)
ANTECEDENT-IN :: (RULE213)

OQRESP-READ2
==========
TRANSLATION :: (q-axis cont overdamped resp read, pass #2)
TYPE :: YES/NO
UPDATED-IN :: (RULE215)
ANTECEDENT-IN :: (RULE217)

OQRESP-READ3
==========
TRANSLATION :: (q-axis cont overdamped resp read, pass #3)
TYPE :: YES/NO
UPDATED-IN :: (RULE219)

ANTECEDENT-IN :: (RULE221)

OQRESP-READ4
TRANSLATION :: (q-axis cont overdamped resp read, #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE223)
ANTECEDENT-IN :: (RULE225)

OQRESP-READ5
TRANSLATION :: (q-axis cont overdamped resp read, #5)
TYPE :: YES/NO
UPDATED-IN :: (RULE227)
ANTECEDENT-IN :: (RULE229)

QRESP-ACQ1
TRANSLATION :: (q-axis controller resp. displayed, passed to ES, pass #1)
TYPE :: YES/NO
UPDATED-IN :: (RULE210)
ANTECEDENT-IN :: (RULE212 RULE211)

QRESP-ACQ2
TRANSLATION :: (q-axis cont resp displayed. passed to ES. pass #2)
TYPE :: YES/NO
UPDATED-IN :: (RULE214)
ANTECEDENT-IN :: (RULE212 RULE215)

QRESP-ACQ3
TRANSLATION :: (q-axis cont resp displayed. passed to ES, pass #3)
TYPE :: YES/NO
UPDATED-IN :: (RULE218)
ANTECEDENT-IN :: (RULE212 RULE219)

QRESP-ACQ4
TRANSLATION :: (q-axis resp displayed, passed to ES, pass #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE222)
ANTECEDENT-IN :: (RULE212 RULE223)

QRESP-ACQ5
TRANSLATION :: (q-axis cont resp displayed. passed to ES, #5)
TYPE :: YES/NO
UPDATED-IN :: (RULE226)
ANTECEDENT-IN :: (RULE212 RULE227)

QRESP-ACQ6
TRANSLATION :: (q cont underdamped resp read. pass 1)
TYPE :: YES/NO
UPDATED-IN :: (RULE281)
ANTECEDENT-IN :: (RULE216)

QRESP-ACQ7
TRANSLATION :: (q cont und resp read #2)
TYPE :: YES/NO
UPDATED-IN :: (RULE282)
ANTECEDENT-IN :: (RULE220)

QRESP-ACQ8
==========
TRANSLATION :: (q axis cont underdamped resp read, pass 3)
TYPE :: YES/NO
UPDATED-IN :: (RULE283)
ANTECEDENT-IN :: (RULE224)

QRESP-ACQ9
==========
TRANSLATION :: (q cont und resp read, #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE284)
ANTECEDENT-IN :: (RULE290)

QRESP-OS
========
TRANSLATION :: (q-axis cont underdamped resp overshoot)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE212 RULE216 RULE220 RULE224 RULE290)
ANTECEDENT-IN :: (RULE236)
RANGE :: (0 100)

QTAU
====
TRANSLATION :: (q-axis cont resp time const. (overdamped))
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE215 RULE211 RULE219 RULE223 RULE227)
RANGE :: (0 50)

QTR
===
TRANSLATION :: (q-axis cont underdamped resp rise time)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE212 RULE216 RULE220 RULE224 RULE290)
RANGE :: (0 50)

QTS
===
TRANSLATION :: (q-axis cont resp settling time)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE212 RULE215 RULE211 RULE216 RULE219 RULE220 RULE223
RULE224 RULE227 RULE290)
RANGE :: (0 50)

STOP-DEC-KPQ
============
TRANSLATION :: (stop further kpq decrement)
TYPE :: YES/NO
UPDATED-IN :: (RULE234)
ANTECEDENT-IN :: (RULE235)

STOP-INC-KPQ
============
TRANSLATION :: (stop further kpq increments. iterations too high)
TYPE :: YES/NO
UPDATED-IN :: (RULE229)
ANTECEDENT-IN :: (RULE235)

UQRESP-READ1
============
TRANSLATION :: (q-axis cont underdamped resp read, pass #1)

```
 TYPE :: YES/NO
 UPDATED-IN :: (RULE212)
 ANTECEDENT-IN :: (RULE213 RULE230)

UQRESP-READ2
============
 TRANSLATION :: (q-axis underdamped resp read, pass #2)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE216)
 ANTECEDENT-IN :: (RULE217 RULE231)

UQRESP-READ3
============
 TRANSLATION :: (q-axis cont underdamped resp read, pass #3)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE220)
 ANTECEDENT-IN :: (RULE221 RULE232)

UQRESP-READ4
============
 TRANSLATION :: (q-axis underdamped resp read, #4)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE224)
 ANTECEDENT-IN :: (RULE225 RULE233)

UQRESP-READ5
============
 TRANSLATION :: (q-axis cont underdamped resp read, #5)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE290)
 ANTECEDENT-IN :: (RULE234)


=======================
  IQS-TUNING-RULES
=======================

RULE209
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (first pass of q-axis controller simulation)
 IF  :: (OPT-MODE = Q-AXIS-LOOP AND ! IQOPT-DONE)
 THEN :: (MODSIM = "SIMPLNT" AND MODE = 2 AND WRITE-DOS-FILE "mode.dat"
       MODE AND IQREF-CUR = 0 AND WRITE-DOS-FILE "tstsig.dat"
       IQREF-CUR AND WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ
       WCQ TSAMP IQS-REF IDS-RATED KSLIP AND DOS-CALL "par2hexn.exe" ""
       AND WRITE-DOS-FILE "CONTROL.DAT" MODSIM AND DOS-CALL "BFGEN.EXE"
"" AND DOS-CALL "SIMODFY.EXE" "" AND PRINT " SIMULATING THE IQS CONTROLLER
       WITH DESIGNED VALUES OF KP AND KI, WITH TUNED KP AND KI FOR IDS
       CONTROLLER.. " " " AND DOS-CALL "" "INFSIM" AND F-IQSIM-DONE )

RULE210
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (first q-axis cont. resp. displayed, passed to ES)
 IF  :: (F-IQSIM-DONE)
 THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
       AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
       DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
       READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
       respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
       QRESP-ACQ1 )
```

```
RULE211
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont overdamped resp parms read, pass #1)
 IF  :: (QRESP-ACQ1 AND NAT-RES = OVERDAMPED)
 THEN :: (QTAU = (E (* 1 (VAL1 FRAME X1))) AND QTS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OQRESP-READ1 )

RULE212
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont underdamped resp read, pass #1)
 IF  :: (NAT-RES = UNDERDAMPED AND (QRESP-ACQ1 OR QRESP-ACQ2 OR
      QRESP-ACQ3 OR QRESP-ACQ4 OR QRESP-ACQ5 ) )
 THEN :: (QRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND QTR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND QTS = (E (* 1 (VAL1 FRAME X4))) AND
      UQRESP-READ1 )

RULE213
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis controller simulation done, pass #2)
 IF  :: (NAT-RES = OVERDAMPED AND (OQRESP-READ1 OR UQRESP-READ1) AND !
      IQOPT-DONE )
 THEN :: (KPQ = (E (+ (VAL1 FRAME DELP-KPQ) (VAL1 FRAME KPQ))) AND WCQ =
      (E (+ (VAL1 FRAME DELP-WCQ) (VAL1 FRAME WCQ)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ WCQ TSAMP
      IQS-REF IDS-RATED KSLIP AND DOS-CALL "par2hexn.exe" "" AND
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND PRINT "Re-simulating with
      modified Kp and Ki, to correct for highly overdamped
      response.." " " AND DOS-CALL "" "infsim" AND CQ1 )

RULE214
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis resp displayed, passed to ES)
 IF  :: (F-IQSIM-DONE AND CQ1)
 THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      QRESP-ACQ2 )

RULE215
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont overdamped resp read. pass #2)
 IF  :: (QRESP-ACQ2 AND NAT-RES = OVERDAMPED)
 THEN :: (QTAU = (E (* 1 (VAL1 FRAME X1))) AND QTS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OQRESP-READ2 )

RULE216
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont underdamped resp read, pass #2)
 IF  :: (NAT-RES = UNDERDAMPED AND QRESP-ACQ6)
```

```
 THEN :: (QRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND QTR = (E (* 1 (VAL1
       FRAME X3 ) ) ) AND QTS = (E (* 1 (VAL1 FRAME X4))) AND
       UQRESP-READ2 )

RULE217
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis controller simulation, pass #3)
 IF  :: (NAT-RES = OVERDAMPED AND (OQRESP-READ2 OR UQRESP-READ2) AND !
       IQOPT-DONE )
 THEN :: (KPQ = (E (+ (VAL1 FRAME DELP-KPQ) (VAL1 FRAME KPQ))) AND WCQ =
       (E (+ (VAL1 FRAME DELP-WCQ) (VAL1 FRAME WCQ)) ) AND
       WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ WCQ TSAMP
       IQS-REF IDS-RATED KSLIP AND DOS-CALL "par2hexn.exe" "" AND
       DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
       WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
       " AND DOS-CALL "refresh.exe" "" AND PRINT "The response of Iqs
       controller is still overdamped. Changing Kp and Ki and
       re-simulating.." " " AND DOS-CALL "" "infsim" AND CQ2 )

RULE218
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont resp displayed, passed to ES, pass #3)
 IF  :: (F-IQSIM-DONE AND CQ1 AND CQ2)
 THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
       AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
       DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
       READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
       respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
       QRESP-ACQ3 )

RULE219
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont overdamped resp read, pass #3)
 IF  :: (QRESP-ACQ3 AND NAT-RES = OVERDAMPED)
 THEN :: (QTAU = (E (* 1 (VAL1 FRAME X1))) AND QTS = (E (* 1 (VAL1 FRAME
       X2 ) ) ) AND OQRESP-READ3 )

RULE220
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont underdamped resp read, pass #3)
 IF  :: (NAT-RES = UNDERDAMPED AND QRESP-ACQ7)
 THEN :: (QRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND QTR = (E (* 1 (VAL1
       FRAME X3 ) ) ) AND QTS = (E (* 1 (VAL1 FRAME X4))) AND
       UQRESP-READ3 )

RULE221
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (incremented kpq, third iteration of q-axis cont
            simulation )
 IF  :: (NAT-RES = OVERDAMPED AND (OQRESP-READ3 OR UQRESP-READ3) AND !
       IQOPT-DONE )
 THEN :: (KPQ = (E (+ (VAL1 FRAME DELP-KPQ) (VAL1 FRAME KPQ))) AND WCQ =
       (E (+ (VAL1 FRAME DELP-WCQ) (VAL1 FRAME WCQ)) ) AND
       WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ WCQ TSAMP
       IQS-REF IDS-RATED KSLIP AND DOS-CALL "par2hexn.exe" "" AND
```

```
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND PRINT " Changing Kp and Ki
      for Iqs controller to correct the overdamped response, and
      re-simulating.." " " AND DOS-CALL "" "infsim" AND CQ3 )

RULE222
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont resp displayed, passed to ES, #4)
 IF  :: (F-IQSIM-DONE AND CQ1 AND CQ2 AND CQ3)
 THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      QRESP-ACQ4 )

RULE223
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont overdamped resp read, #4)
 IF  :: (QRESP-ACQ4 AND NAT-RES = OVERDAMPED)
 THEN :: (QTAU = (E (* 1 (VAL1 FRAME X1))) AND QTS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OQRESP-READ4 )

RULE224
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont underdamped resp read, #4)
 IF  :: (NAT-RES = UNDERDAMPED AND QRESP-ACQ8)
 THEN :: (QRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND QTR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND QTS = (E (* 1 (VAL1 FRAME X4))) AND
      UQRESP-READ4 )

RULE225
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont simulation #4 with inc. kpq)
 IF  :: (NAT-RES = OVERDAMPED AND (OQRESP-READ4 OR UQRESP-READ4) AND !
      IQOPT-DONE )
 THEN :: (KPQ = (E (* 2 (VAL1 FRAME KPQ))) AND WCQ = (E (* 1.2 (VAL1
      FRAME WCQ ) ) ) AND WRITE-DOS-FILE "conpar.dat" KPD-OPT
      WCD-OPT KPQ WCQ TSAMP IQS-REF IDS-RATED KSLIP AND DOS-CALL "
      par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND MODSIM = "
      SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "
      simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND PRINT "Kp and
      Ki of Iqs controller are again modified to make response
      slightly underdamped.. The simulation will continue now.. " " "
      AND DOS-CALL "" "infsim" AND CQ4 )

RULE226
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont resp displayed, passed to ES, #5)
 IF  :: (F-IQSIM-DONE AND CQ1 AND CQ2 AND CQ3 AND CQ4)
 THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
```

```
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      QRESP-ACQ5 )

RULE227
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q-axis cont overdamped resp read, #5)
 IF  :: (QRESP-ACQ5 AND NAT-RES = OVERDAMPED)
 THEN :: (QTAU = (E (* 1 (VAL1 FRAME X1))) AND QTS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OQRESP-READ5 )

RULE229
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (stop incremented kpq iterations, limit reached)
 IF  :: (CQ1 AND CQ2 AND CQ3 AND CQ4 AND OQRESP-READ5)
 THEN :: (STOP-INC-KPQ)

RULE230
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (first decr. kpq iterative simulation)
 IF  :: (NAT-RES = UNDERDAMPED AND UQRESP-READ1 AND ! IQOPT-DONE)
 THEN :: (KPQ = (E (- (VAL1 FRAME KPQ) (VAL1 FRAME DELN-KPQ))) AND WCQ =
      (E (- (VAL1 FRAME WCQ) (VAL1 FRAME DELN-WCQ)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ WCQ TSAMP
      IQS-REF IDS-RATED KSLIP AND DOS-CALL "par2hexn.exe" "" AND
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND PRINT " The response is
      more underdamped than desired. The simulation will now continue
      with modified Kp and Ki.." " " AND DOS-CALL "" "infsim" AND
      CQ5 )

RULE231
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (kpq-decremented simulation done, pass #2)
 IF  :: (NAT-RES = UNDERDAMPED AND UQRESP-READ2 AND ! IQOPT-DONE)
 THEN :: (KPQ = (E (- (VAL1 FRAME KPQ) (VAL1 FRAME DELN-KPQ))) AND WCQ =
      (E (- (VAL1 FRAME WCQ) (VAL1 FRAME DELN-WCQ)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ WCQ TSAMP
      IQS-REF IDS-RATED KSLIP AND DOS-CALL "par2hexn.exe" "" AND
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND PRINT "The Iqs controller
      response is still underdamped and so. Kp and Ki are varied to
      achieve optimum response. The simulation will resume with these
      new values. .. " " " AND DOS-CALL "" "infsim" AND CQ6 )

RULE232
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (#3, kpq-decremented q-axis cont simulation)
 IF  :: (NAT-RES = UNDERDAMPED AND UQRESP-READ3 AND ! IQOPT-DONE)
 THEN :: (KPQ = (E (- (VAL1 FRAME KPQ) (VAL1 FRAME DELN-KPQ))) AND WCQ =
      (E (- (VAL1 FRAME WCQ) (VAL1 FRAME DELN-WCQ)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ WCQ TSAMP
      IQS-REF IDS-RATED KSLIP AND DOS-CALL "par2hexn.exe" "" AND
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
```

```
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND PRINT "The simulation will
      now continue to achieve optimum Iqs controller response, with
      modified PI gains, to correct more than adequate underdamped
      response. .. " " " AND DOS-CALL "" "infsim" AND CQ7 )

RULE233
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (kpq-decremented simulation #4 done)
 IF  :: (NAT-RES = UNDERDAMPED AND UQRESP-READ4 AND ! IQOPT-DONE)
 THEN :: (KPQ = (E (- (VAL1 FRAME KPQ) (VAL1 FRAME DELN-KPQ))) AND WCQ =
      (E (- (VAL1 FRAME WCQ) (VAL1 FRAME DELN-WCQ)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ WCQ TSAMP
      IQS-REF IDS-RATED KSLIP AND DOS-CALL "par2hexn.exe" "" AND
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND PRINT "The Iqs controller
      response is still underdamped. Modifying the PI gains and
      re-simulating.." " " AND DOS-CALL "" "infsim" AND CQ8 )

RULE234
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (stop further kpq decrement, upper iteration limit
          reached )
 IF  :: (CQ5 AND CQ6 AND CQ7 AND CQ8 AND UQRESP-READ5)
 THEN :: (STOP-DEC-KPQ)

RULE235
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (take current kp/ki as opt if iterations are too many)
 IF  :: (STOP-DEC-KPQ OR STOP-INC-KPQ)
 THEN :: (KPQ-OPT = (E (* 1 (VAL1 FRAME KPQ))) AND WCQ-OPT = (E (* 1 (
      VAL1 FRAME WCQ ) ) ) AND IQOPT-DONE )

RULE236
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (opt kp / ki found, stop simulation, next opt. mode is
          speed loop tuning )
 IF  :: (NAT-RES = UNDERDAMPED AND QRESP-OS < 10)
 THEN :: (KPQ-OPT = (E (* 1 (VAL1 FRAME KPQ))) AND WCQ-OPT = (E (* 1 (
      VAL1 FRAME WCQ ) ) ) AND IQOPT-DONE )

RULE281
=======
 SUBJECT :: IQS-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (q cont underdamped resp read, pass 1)
 IF  :: (CQ5)
 THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      QRESP-ACQ6 )

RULE282
=======
```

```
SUBJECT :: IQS-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (q controller underdamped resp read, #2)
IF  :: (CQ5 AND CQ6)
THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
     AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
     READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
     respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
     QRESP-ACQ7 )
```

RULE283

```
SUBJECT :: IQS-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (q axis cont und resp read, #3)
IF  :: (CQ5 AND CQ6 AND CQ7)
THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
     AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
     READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
     respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
     QRESP-ACQ8 )
```

RULE284

```
SUBJECT :: IQS-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (Q axis cont underdamped resp read, #4)
IF  :: (CQ5 AND CQ6 AND CQ7 AND CQ8)
THEN :: (MODSIM = "QACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
     AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
     DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
     READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
     respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
     QRESP-ACQ9 )
```

RULE290

```
SUBJECT :: IQS-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (reads underdamped resp specs. #5)
IF  :: (NAT-RES = UNDERDAMPED AND QRESP-ACQ9 AND ! IQOPT-DONE)
THEN :: (QRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND QTR = (E (* 1 (VAL1
     FRAME X3 ) ) ) AND QTS = (E (* 1 (VAL1 FRAME X4))) AND
     UQRESP-READ5 )
```

Frame :: SPD-TUNING

IDENTIFIER :: "SPD-TUNING-"
TRANSLATION :: (Tuning of the speed loop controller)
PARENTS :: (CONTROL-DESIGN AC-DRIVE-SYSTEM-DESIGN)
GOALS :: (KPS-OPT WCS-OPT SPDOPT-DONE)
INITIALDATA :: (DELP-KPS DELN-KPS DELP-WCS DELN-WCS)
PROMPTEVER :: ("Tuning of the speed loop PI controller in progress.. Wait
for the simulation" :LINE "to proceed....")
PARMGROUP :: SPD-TUNING-PARMS
RULEGROUPS :: (SPD-TUNING-RULES)
ANTECEDENT :: YES
SPD-TUNING-PARMS :: (CS1 CS2 CS3 CS4 CS5 CS6 CS7 CS8 F-SPDSIM-DONE
OSRESP-READ1 OSRESP-READ2 OSRESP-READ3 OSRESP-READ4
OSRESP-READ5 SPDREF-CUR SPDREF-PRE SRESP-ACQ1
SRESP-ACQ2 SRESP-ACQ3 SRESP-ACQ4 SRESP-ACQ5
SRESP-ACQ6 SRESP-ACQ7 SRESP-ACQ8 SRESP-ACQ9
SRESP-OS STAU STOP-DEC-KPS STOP-INC-KPS STR STS
USRESP-READ1 USRESP-READ2 USRESP-READ3 USRESP-READ4
USRESP-READ5)
SPD-TUNING-RULES :: (RULE237 RULE238 RULE239 RULE240 RULE241 RULE242
RULE243 RULE244 RULE245 RULE246 RULE247 RULE248
RULE249 RULE250 RULE251 RULE252 RULE253 RULE254
RULE255 RULE257 RULE258 RULE259 RULE260 RULE261
RULE262 RULE263 RULE264 RULE285 RULE286 RULE287
RULE288 RULE291)

SPD-TUNING-PARMS

CS1
TRANSLATION :: (spd cont incremented kps simulation, pass #1)
TYPE :: YES/NO
UPDATED-IN :: (RULE241)
ANTECEDENT-IN :: (RULE257 RULE242 RULE250 RULE254 RULE246)

CS2
TRANSLATION :: (incremented kps simulation pass #2 done)
TYPE :: YES/NO
UPDATED-IN :: (RULE245)
ANTECEDENT-IN :: (RULE257 RULE250 RULE254 RULE246)

CS3
TRANSLATION :: (incremented kps simulation iteration #3)
TYPE :: YES/NO
UPDATED-IN :: (RULE249)
ANTECEDENT-IN :: (RULE257 RULE250 RULE254)

CS4
TRANSLATION :: (incremented kps simulation iteration #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE253)
ANTECEDENT-IN :: (RULE257 RULE254)

CS5
===
TRANSLATION :: (first dec kps simulation iteration done)
TYPE :: YES/NO
UPDATED-IN :: (RULE258)
ANTECEDENT-IN :: (RULE262 RULE286 RULE287 RULE288 RULE285)

CS6
===
TRANSLATION :: (second decremented kps simulation iteration done)
TYPE :: YES/NO
UPDATED-IN :: (RULE259)
ANTECEDENT-IN :: (RULE262 RULE286 RULE287 RULE288)

CS7
===
TRANSLATION :: (third decremented kps simulation iteration done)
TYPE :: YES/NO
UPDATED-IN :: (RULE260)
ANTECEDENT-IN :: (RULE262 RULE287 RULE288)

CS8
===
TRANSLATION :: (fourth decremented kps iteration done)
TYPE :: YES/NO
UPDATED-IN :: (RULE261)
ANTECEDENT-IN :: (RULE262 RULE288)

F-SPDSIM-DONE
==========
TRANSLATION :: (first speed loop simulation done with designed kp, ki)
TYPE :: YES/NO
UPDATED-IN :: (RULE237)
ANTECEDENT-IN :: (RULE242 RULE250 RULE254 RULE246 RULE238)

OSRESP-READ1
==========
TRANSLATION :: (spd cont overdamped resp read, #1)
TYPE :: YES/NO
UPDATED-IN :: (RULE239)
ANTECEDENT-IN :: (RULE241)

OSRESP-READ2
==========
TRANSLATION :: (spd cont overdamped resp read, pass #2)
TYPE :: YES/NO
UPDATED-IN :: (RULE243)
ANTECEDENT-IN :: (RULE245)

OSRESP-READ3
==========
TRANSLATION :: (spd cont overdamped resp read, case #3)
TYPE :: YES/NO
UPDATED-IN :: (RULE247)
ANTECEDENT-IN :: (RULE249)

OSRESP-READ4
==========
TRANSLATION :: (spd cont overdamped resp parms read, #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE251)
ANTECEDENT-IN :: (RULE253)

OSRESP-READ5
==========

TRANSLATION :: (spd cont overdamped resp read. #5)
TYPE :: YES/NO
UPDATED-IN :: (RULE255)
ANTECEDENT-IN :: (RULE257)

SPDREF-CUR
==========
TRANSLATION :: (Current value of speed ref. test signal)
TYPE :: SINGLEVALUED
UPDATED-IN :: (RULE237)

SPDREF-PRE
==========
TRANSLATION :: (previous value of speed ref. test signal)
TYPE :: SINGLEVALUED

SRESP-ACQ1
==========
TRANSLATION :: (speed loop resp displayed, passed to ES, #1)
TYPE :: YES/NO
UPDATED-IN :: (RULE238)
ANTECEDENT-IN :: (RULE239 RULE240)

SRESP-ACQ2
==========
TRANSLATION :: (spd cont resp displayed, parms passed to ES, #2)
TYPE :: YES/NO
UPDATED-IN :: (RULE242)
ANTECEDENT-IN :: (RULE243 RULE240)

SRESP-ACQ3
==========
TRANSLATION :: (spd cont resp displayed, parms passed to ES, #3)
TYPE :: YES/NO
UPDATED-IN :: (RULE246)
ANTECEDENT-IN :: (RULE247 RULE240)

SRESP-ACQ4
==========
TRANSLATION :: (spd cont resp displayed, parms passed to ES, #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE250)
ANTECEDENT-IN :: (RULE251 RULE240)

SRESP-ACQ5
==========
TRANSLATION :: (spd cont resp displayed, parms passed to Es, #5)
TYPE :: YES/NO
UPDATED-IN :: (RULE254)
ANTECEDENT-IN :: (RULE255 RULE240)

SRESP-ACQ6
==========
TRANSLATION :: (spd cont und resp read. #1)
TYPE :: YES/NO
UPDATED-IN :: (RULE285)
ANTECEDENT-IN :: (RULE244)

SRESP-ACQ7
==========
TRANSLATION :: (spd cont und resp read. #2)
TYPE :: YES/NO
UPDATED-IN :: (RULE286)
ANTECEDENT-IN :: (RULE248)

SRESP-ACQ8
==========
TRANSLATION :: (spd cont und resp read, #3)
TYPE :: YES/NO
UPDATED-IN :: (RULE287)
ANTECEDENT-IN :: (RULE252)

SRESP-ACQ9
==========
TRANSLATION :: (spd cont und resp read, #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE288)
ANTECEDENT-IN :: (RULE291)

SRESP-OS
========
TRANSLATION :: (spd cont underdamped resp perc. overshoot)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE244 RULE248 RULE252 RULE240 RULE291)
ANTECEDENT-IN :: (RULE264 RULE258 RULE259 RULE260 RULE261)
RANGE :: (0 100)

STAU
====
TRANSLATION :: (speed loop overdamped resp time const.)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE239 RULE243 RULE247 RULE251 RULE255)
RANGE :: (0 500)

STOP-DEC-KPS
============
TRANSLATION :: (stop decrementing kps as upper iteration limit reached)
TYPE :: YES/NO
UPDATED-IN :: (RULE262)
ANTECEDENT-IN :: (RULE263 RULE264)

STOP-INC-KPS
============
TRANSLATION :: (stop incrementing kps, higher limit of iter.s reached)
TYPE :: YES/NO
UPDATED-IN :: (RULE257)
ANTECEDENT-IN :: (RULE263 RULE264)

STR
===
TRANSLATION :: (spd cont underdamped resp rise time)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE244 RULE248 RULE252 RULE240 RULE291)
ANTECEDENT-IN :: (RULE264 RULE241 RULE245 RULE249 RULE253)
RANGE :: (0 500)

STS
===
TRANSLATION :: (speed loop overdamped resp. settling time)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE239 RULE243 RULE244 RULE247 RULE248 RULE251 RULE252
RULE255 RULE240 RULE291)
RANGE :: (0 500)

USRESP-READ1
============

TRANSLATION :: (spd cont underdamped resp read, #1)
TYPE :: YES/NO
UPDATED-IN :: (RULE240)
ANTECEDENT-IN :: (RULE241 RULE258)

USRESP-READ2

TRANSLATION :: (spd cont underdamped resp read, pass #2)
TYPE :: YES/NO
UPDATED-IN :: (RULE244)
ANTECEDENT-IN :: (RULE245 RULE259)

USRESP-READ3

TRANSLATION :: (spd cont underdamped resp read, iter #3)
TYPE :: YES/NO
UPDATED-IN :: (RULE248)
ANTECEDENT-IN :: (RULE249 RULE260)

USRESP-READ4

TRANSLATION :: (spd cont underdamped resp parms read, #4)
TYPE :: YES/NO
UPDATED-IN :: (RULE252)
ANTECEDENT-IN :: (RULE253 RULE261)

USRESP-READ5

TRANSLATION :: (spd cont underdamped resp read, #5)
TYPE :: YES/NO
UPDATED-IN :: (RULE291)
ANTECEDENT-IN :: (RULE262)

SPD-TUNING-RULES

RULE237

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
UTILITY :: 69
DESCRIPTION :: (first speed loop simulation done, with designed kp, ki)
IF :: (OPT-MODE = SPEED-LOOP AND ! SPDOPT-DONE)
THEN :: (MODSIM = "SIMPLNT" AND MODE = 3 AND WRITE-DOS-FILE "mode.dat"
MODE AND SPDREF-CUR = 0 AND WRITE-DOS-FILE "tstsig.dat"
SPDREF-CUR AND WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT
KPQ-OPT WCQ-OPT KPS WCS TSAMP KSLIP IQS-RATED IDS-RATED
SPD-REF BASE-SPD AND DOS-CALL "par2hexn.exe" "" AND
WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "bfgen.exe" ""
AND DOS-CALL "simodfy.exe" "" AND PRINT "The speed loop
optimization will now begin, with designed Kp and Ki for speed
loop and optimized PI gains for inner current loops.." " " AND
DOS-CALL "" "infsim" AND F-SPDSIM-DONE)

RULE238

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (speed loop response displayed, passed to ES, #1)
IF :: (F-SPDSIM-DONE)
THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND

```
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      SRESP-ACQ1 )

RULE239
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont overdamped resp parms read #1)
 IF  :: (SRESP-ACQ1 AND NAT-RES = OVERDAMPED)
 THEN :: (STAU = (E (* 1 (VAL1 FRAME X1))) AND STS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OSRESP-READ1 )

RULE240
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont underdamped resp parms read, #1)
 IF  :: ((SRESP-ACQ1 OR SRESP-ACQ2 OR SRESP-ACQ3 OR SRESP-ACQ4 OR
      SRESP-ACQ5 ) AND NAT-RES = UNDERDAMPED )
 THEN :: (SRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND STR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND STS = (E (* 1 (VAL1 FRAME X4))) AND
      USRESP-READ1 )

RULE241
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont incremented kp simulation, pass #1)
 IF  :: ((NAT-RES = OVERDAMPED OR (NAT-RES = UNDERDAMPED AND STR > 0.05
      ) ) AND (OSRESP-READ1 OR USRESP-READ1) AND ! SPDOPT-DONE )
 THEN :: (KPS = (E (+ (VAL1 FRAME DELP-KPS) (VAL1 FRAME KPS))) AND WCS =
      (E (+ (VAL1 FRAME DELP-WCS) (VAL1 FRAME WCS)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ-OPT WCQ-OPT
      KPS WCS TSAMP KSLIP IQS-RATED IDS-RATED SPD-REF BASE-SPD AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT "The speed controller response is still unsatisfactory.
      Modifying the gains and re-simulating.." " " AND DOS-CALL "" "
      infsim" AND CS1 )

RULE242
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont resp displayed, parms passed to Es, #2)
 IF  :: (F-SPDSIM-DONE AND CS1)
 THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      SRESP-ACQ2 )

RULE243
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont overdamped resp read, pass #2)
 IF  :: (SRESP-ACQ2 AND NAT-RES = OVERDAMPED)
 THEN :: (STAU = (E (* 1 (VAL1 FRAME X1))) AND STS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OSRESP-READ2 )

RULE244
```

```
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont underdamped resp read, pass #2)
 IF  :: (SRESP-ACQ6 AND NAT-RES = UNDERDAMPED)
 THEN :: (SRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND STR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND STS = (E (* 1 (VAL1 FRAME X4))) AND
      USRESP-READ2 )

RULE245
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (incremented kps simulation pass #2 done)
 IF  :: ((NAT-RES = OVERDAMPED OR (NAT-RES = UNDERDAMPED AND STR > 0.02
      ) ) AND (OSRESP-READ2 OR USRESP-READ2) AND ! SPDOPT-DONE )
 THEN :: (KPS = (E (+ (VAL1 FRAME DELP-KPS) (VAL1 FRAME KPS))) AND WCS =
      (E (+ (VAL1 FRAME DELP-WCS) (VAL1 FRAME WCS)) ) AND
      WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ-OPT WCQ-OPT
      KPS WCS TSAMP KSLIP IQS-RATED IDS-RATED SPD-REF BASE-SPD AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      PRINT " The response of the speed controller is still
      unsatisfactory. Re-simulating with changed Kp and Ki," " " AND
      DOS-CALL "" "infsim" AND CS2 )

RULE246
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont resp displayed, parms passed to ES, #3)
 IF  :: (F-SPDSIM-DONE AND CS1 AND CS2)
 THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      SRESP-ACQ3 )

RULE247
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (speed controller overdamped resp read, case #3)
 IF  :: (SRESP-ACQ3 AND NAT-RES = OVERDAMPED)
 THEN :: (STAU = (E (* 1 (VAL1 FRAME X1))) AND STS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OSRESP-READ3 )

RULE248
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont underdamped resp #3 read)
 IF  :: (SRESP-ACQ7 AND NAT-RES = UNDERDAMPED)
 THEN :: (SRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND STR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND STS = (E (* 1 (VAL1 FRAME X4))) AND
      USRESP-READ3 )

RULE249
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (incremented kps simulation, iter #3)
 IF  :: ((NAT-RES = OVERDAMPED OR (NAT-RES = UNDERDAMPED AND STR > 0.02
```

```
      ) ) AND (OSRESP-READ3 OR USRESP-READ3) AND ! SPDOPT-DONE )
 THEN :: (PRINT "The response of speed controller is still
      unsatisfactory. The gains are now modified and simulation will
      resume.." " " AND KPS = (E (+ (VAL1 FRAME DELP-KPS) (VAL1
      FRAME KPS ) ) ) AND WCS = (E (+ (VAL1 FRAME DELP-WCS) (VAL1
      FRAME WCS ) ) ) AND WRITE-DOS-FILE "conpar.dat" KPD-OPT
      WCD-OPT KPQ-OPT WCQ-OPT KPS WCS TSAMP KSLIP IQS-RATED
      IDS-RATED SPD-REF BASE-SPD AND DOS-CALL "par2hexnexe" "" AND
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND DOS-CALL "" "infsim" AND
      CS3 )

RULE250
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont resp displayed, parms passed to ES, #4)
 IF  :: (F-SPDSIM-DONE AND CS1 AND CS2 AND CS3)
 THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      SRESP-ACQ4 )

RULE251
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont overdamped resp parms read, #4)
 IF  :: (SRESP-ACQ4 AND NAT-RES = OVERDAMPED)
 THEN :: (STAU = (E (* 1 (VAL1 FRAME X1))) AND STS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OSRESP-READ4 )

RULE252
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont underdamped resp parms read, #4)
 IF  :: (SRESP-ACQ8 AND NAT-RES = UNDERDAMPED)
 THEN :: (SRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND STR = (E (* 1 (VAL1
      FRAME X3 ) ) ) AND STS = (E (* 1 (VAL1 FRAME X4))) AND
      USRESP-READ4 )

RULE253
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (incremented kps simulation iteration #4)
 IF  :: ((NAT-RES = OVERDAMPED OR (NAT-RES = UNDERDAMPED AND STR > 0.02
      ) ) AND (OSRESP-READ4 OR USRESP-READ4) AND ! SPDOPT-DONE )
 THEN :: (PRINT "The response of the speed controller is unsatisfactory.
      The gains are changed and the simulation will resume" " " AND
      KPS = (E (* 2 (VAL1 FRAME KPS))) AND WCS = (E (* 1.2 (VAL1
      FRAME WCS ) ) ) AND WRITE-DOS-FILE "conpar.dat" KPD-OPT
      WCD-OPT KPQ-OPT WCQ-OPT KPS WCS TSAMP KSLIP IQS-RATED
      IDS-RATED SPD-REF BASE-SPD AND DOS-CALL "par2hexn.exe" "" AND
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND DOS-CALL "" "infsim" AND
      CS4 )

RULE254
=======
```

```
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont resp displayed, parms passed to Es, #5)
 IF  :: (F-SPDSIM-DONE AND CS1 AND CS2 AND CS3 AND CS4)
 THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
      AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
      DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
      READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
      respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
      SRESP-ACQ5 )

RULE255
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (spd cont overdamped resp read. #5)
 IF  :: (SRESP-ACQ5 AND NAT-RES = OVERDAMPED)
 THEN :: (STAU = (E (* 1 (VAL1 FRAME X1))) AND STS = (E (* 1 (VAL1 FRAME
      X2 ) ) ) AND OSRESP-READ5 )

RULE257
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (higher limit of iterations reached, stop incrementing
        kps )
 IF  :: (CS1 AND CS2 AND CS3 AND CS4 AND OSRESP-READ5)
 THEN :: (STOP-INC-KPS)

RULE258
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (first decremented kps simulation iteration done)
 IF  :: (NAT-RES = UNDERDAMPED AND SRESP-OS > 10 AND USRESP-READ1 AND !
      SPDOPT-DONE )
 THEN :: (PRINT " The response of speed controller is more underdamped
      than desired. Simulation will now continue with changed gains..
      " " " AND KPS = (E (- (VAL1 FRAME KPS) (VAL1 FRAME DELN-KPS)))
      AND WCS = (E (- (VAL1 FRAME WCS) (VAL1 FRAME DELN-WCS))) AND
      WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ-OPT WCQ-OPT
      KPS WCS TSAMP KSLIP IQS-RATED IDS-RATED SPD-REF BASE-SPD AND
      DOS-CALL "par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND
      MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND
      DOS-CALL "simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND
      DOS-CALL "" "infsim" AND CS5 )

RULE259
=======
 SUBJECT :: SPD-TUNING-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (decremented kps simulation iteration #2)
 IF  :: (NAT-RES = UNDERDAMPED AND SRESP-OS > 10 AND USRESP-READ2 AND !
      SPDOPT-DONE )
 THEN :: (PRINT " The response of the speed controller is still more
      underdamped than necessary. Re-simulating with changed Kp and
      Ki in the speed loop, " " " AND KPS = (E (- (VAL1 FRAME KPS) (
      VAL1 FRAME DELN-KPS ) ) ) AND WCS = (E (- (VAL1 FRAME WCS) (
      VAL1 FRAME DELN-WCS ) ) ) AND WRITE-DOS-FILE "conpar.dat"
      KPD-OPT WCD-OPT KPQ-OPT WCQ-OPT KPS WCS TSAMP KSLIP IQS-RATED
      IDS-RATED SPD-REF BASE-SPD AND DOS-CALL "par2hexn.exe" "" AND
      DOS-CALL "bfgen.exe" "" AND MODSIM = "SIMPLNT" AND
      WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "simodfy.exe" "
      " AND DOS-CALL "refresh.exe" "" AND DOS-CALL "" "infsim" AND
      CS6 )
```

RULE260

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (decremented kps simulation iteration #3)
IF :: (NAT-RES = UNDERDAMPED AND SRESP-OS > 10 AND USRESP-READ3 AND !
SPDOPT-DONE)
THEN :: (PRINT " The speed response is highly underdamped. The gains Kp
and Ki are modified and simulation will resume with these new
values.." " " AND KPS = (E (- (VAL1 FRAME KPS) (VAL1 FRAME
DELN-KPS))) AND WCS = (E (- (VAL1 FRAME WCS) (VAL1 FRAME
DELN-WCS))) AND WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT
KPQ-OPT WCQ-OPT KPS WCS TSAMP KSLIP IQS-RATED IDS-RATED
SPD-REF BASE-SPD AND DOS-CALL "par2hexn.exe" "" AND DOS-CALL "
bfgen.exe" "" AND MODSIM = "SIMPLNT" AND WRITE-DOS-FILE "
control.dat" MODSIM AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "
refresh.exe" "" AND DOS-CALL "" "infsim" AND CS7)

RULE261

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (decremented kps simulation iteration #4)
IF :: (NAT-RES = UNDERDAMPED AND SRESP-OS > 10 AND USRESP-READ4 AND !
SPDOPT-DONE)
THEN :: (PRINT " Speed controller response is more underdamped than
acceptable. So, gains Kp and Ki are now modified and simulation
will continue with these new values.." " " AND KPS = (E (- (
VAL1 FRAME KPS) (VAL1 FRAME DELN-KPS))) AND WCS = (E (- (
VAL1 FRAME WCS) (VAL1 FRAME DELN-WCS))) AND WRITE-DOS-FILE "
conpar.dat" KPD-OPT WCD-OPT KPQ-OPT WCQ-OPT KPS WCS TSAMP
KSLIP IQS-RATED IDS-RATED SPD-REF BASE-SPD AND DOS-CALL "
par2hexn.exe" "" AND DOS-CALL "bfgen.exe" "" AND MODSIM = "
SIMPLNT" AND WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "
simodfy.exe" "" AND DOS-CALL "refresh.exe" "" AND DOS-CALL "" "
infsim" AND CS8)

RULE262

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (stop decrementing kps further. upper iter limit reached
)
IF :: (CS5 AND CS6 AND CS7 AND CS8 AND USRESP-READ5)
THEN :: (STOP-DEC-KPS)

RULE263

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (#iter limit reached on either side. kp/ki values are
opt. vals)
IF :: (STOP-DEC-KPS OR STOP-INC-KPS)
THEN :: (KPS-OPT = (E (* 1 (VAL1 FRAME KPS))) AND WCS-OPT = (E (* 1 (
VAL1 FRAME WCS))) AND SPDOPT-DONE)

RULE264

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (opt values of kp, ki found. prepare for test run)
IF :: ((NAT-RES = UNDERDAMPED AND SRESP-OS < 10 AND STR < 0.05) OR
STOP-DEC-KPS OR STOP-INC-KPS)
THEN :: (KPS-OPT = (E (* 1 (VAL1 FRAME KPS))) AND WCS-OPT = (E (* 1 (
VAL1 FRAME WCS))) AND SPDOPT-DONE)

RULE285

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (speed cont underdamped resp read, #1)
IF :: (CS5)
THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
SRESP-ACQ6)

RULE286

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (Speed controller underdamped resp read, pass 2)
IF :: (CS5 AND CS6)
THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
SRESP-ACQ7)

RULE287

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (speed controller underdamped response read, #3)
IF :: (CS5 AND CS6 AND CS7)
THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
SRESP-ACQ8)

RULE288

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (Speed controller underdamped response read, #4)
IF :: (CS5 AND CS6 AND CS7 AND CS8)
THEN :: (MODSIM = "WACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
DOS-CALL "trscan.exe" "resp.t natres.dat respon.dat" AND
READ-DOS-FILE "natres.dat" INDEX 1 NAT-RES AND READ-DOS-FILE "
respon.dat" INDEX 1 X1 INDEX 2 X2 INDEX 3 X3 INDEX 4 X4 AND
SRESP-ACQ9)

RULE291

SUBJECT :: SPD-TUNING-RULES
ANTECEDENT :: YES
DESCRIPTION :: (spd cont underdamped resp #5 read)
IF :: (NAT-RES = UNDERDAMPED AND SRESP-ACQ9 AND ! SPDOPT-DONE)
THEN :: (SRESP-OS = (E (* 1 (VAL1 FRAME X1))) AND STR = (E (* 1 (VAL1
FRAME X3))) AND STS = (E (* 1 (VAL1 FRAME X4))) AND
USRESP-READ5)

Frame :: TEST-RUN

IDENTIFIER :: "TEST-RUN-"
TRANSLATION :: (Test run of the tuned system)
PARENTS :: (CONTROL-DESIGN AC-DRIVE-SYSTEM-DESIGN)
GOALS :: (TEST-RUN-DONE)
INITIALDATA :: (TEST-RUN?)
PROMPTEVER :: ("The optimization of the control system is now complete. A
test run showing" :LINE "the performance of the tuned
system will now be conducted... Please wait for" :LINE "the
simulation to proceed.")
PARMGROUP :: TEST-RUN-PARMS
RULEGROUPS :: (TEST-RUN-RULES)
ANTECEDENT :: YES
TEST-RUN-PARMS :: (COUNTER F-TSTSIM-DONE ID-OS ID-TR IQ-OS IQ-TR SPD-OS
SPD-TR TSTRUN-DONE)
TEST-RUN-RULES :: (RULE265 RULE266 RULE293)

TEST-RUN-PARMS

COUNTER

TRANSLATION :: (counter for the number of sampling instants)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE265)
RANGE :: (0 1100)

F-TSTSIM-DONE

TRANSLATION :: (first test-run simulation run done)
TYPE :: YES/NO
UPDATED-IN :: (RULE265)
ANTECEDENT-IN :: (RULE266)

ID-OS

TRANSLATION :: (id response overshoot)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE266)
RANGE :: (0 100)

ID-TR

TRANSLATION :: (id response rise time)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE266)
RANGE :: (0 50)

IQ-OS

TRANSLATION :: (iq response overshoot)
TYPE :: SINGLEVALUED
EXPECT :: POSITIVE-NUMBER
UPDATED-IN :: (RULE266)
RANGE :: (0 100)

IQ-TR

```
 TRANSLATION :: (iq response rise time)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-IN :: (RULE266)
 RANGE :: (0 50)

SPD-OS
======
 TRANSLATION :: (speed response overshoot)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-IN :: (RULE266)
 RANGE :: (0 100)

SPD-TR
======
 TRANSLATION :: (speed response rise time)
 TYPE :: SINGLEVALUED
 EXPECT :: POSITIVE-NUMBER
 UPDATED-IN :: (RULE266)
 RANGE :: (0 100)

TSTRUN-DONE
===========
 TRANSLATION :: (test run completed)
 TYPE :: YES/NO
 UPDATED-IN :: (RULE293 RULE266)
 DEFAULT :: (NO)
 ==================
  TEST-RUN-RULES
 ==================
RULE265
=======
 SUBJECT :: TEST-RUN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (first test run simulation done)
 IF  :: (OPT-MODE = TEST-RUN AND TEST-RUN?)
 THEN :: (PRINT "The optimization of the controllers for IVC-IM drive is
       now complete. This test run will allow you to review the
       responses of the tuned controllers.." " " AND MODE = 4 AND
       WRITE-DOS-FILE "mode.dat" MODE AND MODSIM = "SIMPLNT" AND
       WRITE-DOS-FILE "conpar.dat" KPD-OPT WCD-OPT KPQ-OPT WCQ-OPT
       KPS-OPT WCS-OPT TSAMP KSLIP IQS-RATED IDS-RATED SPD-REF
       BASE-SPD AND WRITE-DOS-FILE "control.dat" MODSIM AND DOS-CALL "
       bfgen.exe" "" AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "
       par2hexn.exe" "" AND COUNTER = 0 AND READ-DOS-FILE "
       timeinfo.dat" INDEX 1 TIME AND WRITE-DOS-FILE "timeinfo.dat"
       TIME COUNTER AND DOS-CALL "" "infsim" AND F-TSTSIM-DONE )

RULE266
=======
 SUBJECT :: TEST-RUN-RULES
 ANTECEDENT :: YES
 DESCRIPTION :: (displays graphically results of test simulation run,
          passes results to ES )
 IF  :: (F-TSTSIM-DONE)
 THEN :: (MODSIM = "TACQDISP" AND WRITE-DOS-FILE "control.dat" MODSIM
       AND DOS-CALL "simodfy.exe" "" AND DOS-CALL "simnon.exe" "" AND
       READ-DOS-FILE "tstrsp.dat" INDEX 1 ID-OS INDEX 2 ID-TR INDEX 3
       IQ-OS INDEX 4 IQ-TR INDEX 5 SPD-OS INDEX 6 SPD-TR AND
       TSTRUN-DONE )

RULE293
=======
 SUBJECT :: TEST-RUN-RULES
```

ANTECEDENT :: YES
DESCRIPTION :: (No test run, if user doesn't want)
IF :: (! TEST-RUN?)
THEN :: (PRINT "You chose not to test-run the optimized system. Hence,
the results of the complete optimization process will be
displayed now" AND TSTRUN-DONE)

APPENDIX D.2 Plant (Power Converter and Machine) Simulation Program Listing

```
"*****************************************************************
" Synchronous to Stationary Frame Voltage Transformation
"*****************************************************************
CONTINUOUS SYSTEM SYNSTA
"INPUT  vdser vqser thetae
OUTPUT Vqss Vdss cosref sinref
TIME t

cosref = cos(thetae)
sinref = sin(thetae)

"Synchronous to Stationary D-Q Ref. Frame
Vqss = vqser*cosref + vdser*sinref
Vdss = vdser*cosref - vqser*sinref

"PARAMETERS
vdser:0
vqser:0
thetae:0
END

"*****************************************************************
" Two Phase to Three Phase Voltage Transformation
"*****************************************************************

CONTINUOUS SYSTEM PHASE23
INPUT Vqss Vdss
OUTPUT Vac Vbc Vcc
TIME t

"Synchronous dq to Stationary ABC Ref. Frame
Vac=Vqss
Vbc=-(Vdss*sqrt(3)+Vqss)*0.5
Vcc=-(Vac+Vbc)

END

"*****************************************************************
" Sinusoidal PWM Inverter
"*****************************************************************
CONTINUOUS SYSTEM SPWM
"SINUSOIDAL PWM INVERTER
INPUT  Varef Vbref Vcref
OUTPUT  Van Vbn Vcn
TIME t
pi = 3.141592654

"********* Triangle Generator*****************
Ts = 1/fs              "Period of carrier wave
tn = MOD(t,Ts)*fs
trg = IF tn<0.25 THEN 4*tn ELSE IF tn<0.75 THEN -4*tn+2 ELSE 4*tn-4
y = amp*trg

"PWM Generator
Vam = Varef "Modulating voltages
Vbm = Vbref
Vcm = Vcref

Vao = IF Vam>y THEN Vd ELSE 0
Vbo = IF Vbm>y THEN Vd ELSE 0
Vco = IF Vcm>y THEN Vd ELSE 0
```

```
"Switching Signals => 1:Upper Switch, 0:Lower Switch
"A = IF Vao>0 THEN 1 ELSE 0
"B = IF Vbo>0 THEN 1 ELSE 0
"C = IF Vco>0 THEN 1 ELSE 0

"Output Current of Rectfier
"Idout = A*Ia + B*Ib + C*Ic

Van = (2*vao-vbo-vco)/3    "Output phase voltages
Vbn = (2*vbo-vao-vco)/3
Vcn = (2*vco-vao-vbo)/3

Vab = Vao-Vbo              "Output line voltages
Vbc = Vbo-Vco
Vca = Vco-Vao

"DC-Link Voltage
amp:1.0
Vd:300
fs:15000
END

"*****************************************************************
" Induction Motor D-Q Axis Current Model
"*****************************************************************
CONTINUOUS SYSTEM MACHINDQ
INPUT  Va Vb Vc cosref sinref "weref wb lr rs lm rr ls j pole kll

STATE Iqs  Ids  Iqr  Idr  w
DER  DIqs DIds DIqr DIdr Dw
TIME t
pi = 3.141592654
sqr3 = sqrt(3)
we = weref

"State Equations
Diqs = (Lr*(Vqs-rs*iqs)-fse*ids+Lm*rr*iqr-Lrm*w*idr)/kl
Dids = (fse*iqs+lr*(Vds-rs*ids)+Lrm*w*iqr+Lm*rr*idr)/kl
Diqr = (Lm*(rs*Iqs-Vqs)+w*Lsm*ids-iqr*rr*ls+fre*idr)/kl
Didr = (-w*Lsm*Iqs+Lm*(ids*rs-Vds)-fre*iqr-ls*rr*idr)/kl

"Electromechanical Equations
Dw = Te/j*pole/2
"Tl = IF wm>0 THEN Kl*wm*wm ELSE -KL*wm*wm
"KL:5E-3 "5.924E-4
Te = 0.75*pole*Lm*(Idr*Iqs-Iqr*Ids)
spd = w/pole/pi*60
wr = w          "Electrical rotor speed
wm = wr*2/pole  "Mechanical rotor speed

Fdss  = Lm*Idrs  + Ls*Idss
Fqss  = Lm*Iqrs  + Ls*Iqss
Fdrs =  Lm*Idss  + Lr*Idrs
Fqrs =  Lm*Iqss  + Lr*Iqrs

"Voltage Transformations
Vqss = va
Vdss = (vc-vb)/sqr3
Vqs  = vqss*cosref-vdss*sinref
Vds  = vqss*sinref+vdss*cosref

"Relationships of Induction Machine
delw = we-w
fse = lsr*we-delw*lmm
fre = we*lmm-delw*lsr
```

```
lrm = Lr*Lm
lsm = Ls*Lm
lsr = Ls*Lr
lmm = Lm*Lm
k1 = lsr-lmm

"Current Transformations
Iqss = iqs*cosref+ids*sinref
Idss = ids*cosref-iqs*sinref
Iqrs = iqr*cosref+idr*sinref
Idrs = idr*cosref-iqr*sinref

Ia = Iqss
Ib = -(0.5*Iqss)-(0.5*sqr3*Idss)
Ic = -(0.5*Iqss)+(0.5*sqr3*Idss)

"Parameters
Rs:0.370
Rr:0.318
Ls:34.13E-3
Lr:34.13E-3
Lm :32.00E-3
"Ls = Lls + Lm
"Lr = Llr + Lm
Tr = Lr/Rr
j:0.0175
pole:4
wb:376.991  "Electrical base speed
weref:0

"Initial Values
Iqs: 0.
Ids: 0.0
Iqr: 0.
Idr: 0
w: 0.

END

"*****************************************************************
" Connecting System for Plant (Converter + Machine) Simulation
"*****************************************************************
CONNECTING SYSTEM CONPLA
TIME t

Vqss[PHASE23] = Vqss[SYNSTA]
Vdss[PHASE23] = Vdss[SYNSTA]

Varef[SPWM] = Vac[PHASE23]
Vbref[SPWM] = Vbc[PHASE23]
Vcref[SPWM] = Vcc[PHASE23]

cosref[MACHINDQ] = cosref[SYNSTA]
sinref[MACHINDQ] = sinref[SYNSTA]
Va[MACHINDQ]=Van[SPWM]
Vb[MACHINDQ]=Vbn[SPWM]
Vc[MACHINDQ]=Vcn[SPWM]


END
```

APPENDIX D.3 TMS 320C25 Based Controller Simulation Routines

```
/*************************************************************************/
/* STIMER.C -- This is the master control routine, in which reside the */
/*          following modules:                                          */
/*          1: I/O of data from various files                           */
/*          2: Scaling factor computation                               */
/*          3: Floating to fixed pt conversion                          */
/*          4: C based slow PI controller for speed                     */
/*          5: AL modules for,                                          */
/*            a: 3-2ph conversion                                       */
/*            b: Angle generation                                       */
/*            c: Vector Rotation                                        */
/*            d: Ids PI controller                                      */
/*            e: Iqs PI controller                                      */
/*          6: Fix to floating pt conversion                            */
/*************************************************************************/

/* Input parameters */
extern int Mode, Counter;
extern float Ib_fl, Ic_fl, Wrr_fl, Wr_fl, Kpd_fl, Wcd_fl, Kpq_fl, Wcq_fl;
extern float Kps_fl, Wcs_fl, Idr_fl, Iqr_fl, Wbase;
extern float Ts1, Xs_fl, Xs_fl_n;
extern int SQR1O3 = 18918;
/* Ids_rated Supplied for Ids reference current generation      */
/* Kslip is the nominal slip gain (rated)                       */
extern float Ids_rated, Kslip, Iqs_rated;
/* Scaled parameters (fx pt) passed to AL controllers */
extern int Kpd_fx, Kid_fx, Kpq_fx, Kiq_fx, Ksl;
extern int Ib_fx, Ic_fx, Wr_fx, Ts1_fx;      /* Ts1_fx for we integration */
extern int Idr_fx, Iqr_fx, DT;
/* AL outputs (fixed pt) */
extern int Vdr_fx_n, Vqr_fx_n, Idr_fx_n, Iqr_fx_n;
extern int Eds_fx, Eqs_fx, YDH, YDL, YQH, YQL, We;
extern int Thetah, Thetal;
/* Scaled parameters (fl pt) sent out to SIMNON  */
extern float Vdr_fl_n, Vqr_fl_n, Idr_fl_n, Iqr_fl_n, We_fl;
extern float Eds_fl, Eqs_fl, Ews_fl, Thetae_fl;
/* Following are the AL functions called */
extern int Inmod(), Init();     /* Reads mode of simulation */
extern int Inidsdat(), Ouidsdat(); /* I/O for ids tuning mode */
extern int Iniqsdat(), Ouiqsdat(); /* I/O for iqs tuning mode */
extern int Inoldat(), Ouoldat();  /* I/O for outer loop mode */
extern int Phase32(), Thetagen(), Vectrot(), Sintab(); /* Basic vector control routines */
extern int Idscont(), Iqscont();  /* Current controllers     */
extern int Outstat();          /* Outputs buffer states      */
main()
{
  /* Ids reference current command slaved from speed reference command */
  float Idrfunc(float wr, float wb);
  void Spdpilim(void);
  /* Scaling factors defined below   */
  float si, skp, ski, se, sv, sw, st, stsamp, sksl;
  float ksmax, kpm = 10, kim, emax = 1638.35;
  float vmax, ymax, tsmax, femax = 266.66667;
  /* Derived scales   */
  tsmax = 1 / femax;
  vmax = kpm * emax;
  kim = kpm * femax / 128;
  ksmax = (6.283 * femax) / emax;
  /* Scale factor computation              */
  si = 32767.0 / emax;
  se = si;
  sv = 32767.0 / vmax;
```

```
sw = 32767.0 / (6.2832 * femax);
st = 5215.03;
stsamp = 32767 / tsmax;
sksl = 32767.0 / ksmax;
skp = 32767.0 / kpm;
ski = 32767.0 / kim;
/* Initialize variables and constants      */
Init();

/* Read mode status */
Inmod();
/* If mode = ids tuning, execute the following */
if (Mode == 1)
        {
        /* Input data from various files */
        /* Internal states read in fx. pt, external data in fl. pt. format*/
        Inidsdat();
        /* Floating pt to fixed pt conversion   */
        Ib_fx = Ib_fl * si;
        Ic_fx = Ic_fl * si;
        Idr_fx = Idr_fl * si;
        /* Ids tuning done at standstill       */
        Wr_fx = 0x0000;
        Ts1_fx = Ts1 * stsamp;
        Ksl = Kslip * sksl;
        DT = 2 * Ts1_fx;
        /* Iq ref. = 0, but has to be supplied for angle calc.  */
        Iqr_fx = 0x0000;
        /* Fixed pt gains for ids controller */
        Kpd_fx = Kpd_fl * skp;
        Kid_fx = Kpd_fl * Wcd_fl * ski;
        /* Call AL-based controllers     */
        Phase32();
        Thetagen();
        Vectrot();
        Idscont();
        /* Convert fix to floating pt */
        Vdr_fl_n = (float)Vdr_fx_n / (256 * sv);
        Eds_fl = (float)Eds_fx / (int)se;
        Thetae_fl = (float)Thetah / (int)st;
        We_fl = (float)We / (int) sw;
        /* Outputs to output data file(s)       */
        Ouidsdat();
        }

/* If mode = iqs tuning, execute following block    */
if (Mode == 2)
        {

        /* Input data from various files */
        Iniqsdat();

        /* Floating pt to fixed pt conversion   */
        Ib_fx = Ib_fl * si;
        Ic_fx = Ic_fl * si;
        Wr_fx = Wr_fl * sw;
        /* Id ref = rated during iqs loop tuning */
        Idr_fx = Ids_rated * si;
        Iqr_fx = Iqr_fl * si;
        Ts1_fx = Ts1 * stsamp;
        Ksl = Kslip * sksl;
        DT = 2 * Ts1_fx;
        /* Fixed pt gains for ids controller */
        Kpd_fx = Kpd_fl * skp;
        Kid_fx = Kpd_fl * Wcd_fl * ski;
```

```
        Kpq_fx = Kpq_fl * skp;
        Kiq_fx = Kpq_fl * Wcq_fl * ski;
        /* Call AL-based controllers    */
        Phase32();
        Thetagen();
        Vectrot();
        Idscont();
        Iqscont();
        /* Convert fix to floating pt */
        Vdr_fl_n = Vdr_fx_n / (256 * sv);
        Eds_fl = Eds_fx / se;
        Vqr_fl_n = Vqr_fx_n / (256 * sv);
        Eqs_fl = Eqs_fx / se;
        Thetae_fl = (float)Thetah / (int)st;
        We_fl = (float)We / (int) sw;
        /* Outputs to output data file(s)      */
        Ouiqsdat();
        }

/* If mode = complete system, execute following block    */
if ((Mode == 3) || (Mode == 4))
        {
        /* Input data from various files */
        Inoldat();
        /* Floating pt to fixed pt conversion   */
        Ib_fx = Ib_fl * si;
        Ic_fx = Ic_fl * si;
        Wr_fx = Wr_fl * sw;
        Ts1_fx = Ts1 * stsamp;
        Ksl = Kslip * sksl;
        DT = 2 * Ts1_fx;
        /* Fixed pt gains for ids and iqs controllers */
        Kpd_fx = Kpd_fl * skp;
        Kid_fx = Kpd_fl*Wcd_fl * ski;
        Kpq_fx = Kpq_fl * skp;
        Kiq_fx = Kpq_fl*Wcq_fl * ski;

        if (Counter == 0)
          {
          /* Speed PI controller and limiter implemented */
          Spdpilim();
          /* Id ref generated using function generator    */
          /* Idr_fl_n = Idrfunc(Wrr_fl, Wbase); */
          /* counter reset here                    */
          Counter = 5;
          }

        else if (Counter != 0)
          {
          /* Update counter and maintain previous current refs */
          Counter -= 1;
          /* Idr_fl_n = Idr_fl;          */
          Iqr_fl_n = Iqr_fl;
          Xs_fl_n = Xs_fl;
          }

        /* Fixed point current ref.s generated here */
        Idr_fx = Ids_rated * si;       /* Ids ref. always rated val */
        Iqr_fx = Iqr_fl_n * si;
        /* Call AL-based controllers    */
        Phase32();
        Thetagen();
        Vectrot();
        Idscont();
        Iqscont();
```

```
            /* Convert fix to floating pt */
            Vdr_fl_n = Vdr_fx_n / (256 * sv);
            Eds_fl = Eds_fx / se;
            Vqr_fl_n = Vqr_fx_n / (256 * sv);
            Eqs_fl = Eqs_fx / se;
            Thetae_fl = (float)Thetah / (int)st;
            We_fl = (float)We / (int) sw;
            /* Outputs to output data file(s)      */
            Ouoldat();
            }
  /* Internal States output to buffer file */
  Outstat();
}
float Idrfunc(float wr, float wb)
{
  /* Id reference function generator  */
  float idsr;
  if (wr <= wb)
            {
            idsr = Ids_rated;
            }
  else
            {
            idsr = Ids_rated * (wb / wr);
            }
  return(idsr);
}

void Spdpilim(void)
{
  /* Implement PI speed controller in C language    */
  Ews_fl = Wrr_fl - Wr_fl;
  Iqr_fl_n = Kps_fl * Ews_fl + Kps_fl*Wcs_fl * (5*Ts1) * Xs_fl;
  Xs_fl_n = Xs_fl + Ews_fl;

  /* Implement limiter below  */
  if (Iqr_fl_n > 0.95*Iqs_rated)
            {
            Iqr_fl_n = 0.95*Iqs_rated;
            }
  else if (Iqr_fl_n < -0.95*Iqs_rated)
            {
            Iqr_fl_n = -0.95*Iqs_rated;
            }
}

;*************************************************************
; Initialization Subroutine for Variables, Parameters and Constants
;*************************************************************
     .title "init.asm"
     .global _Init, _Inmod, _Inidsdat, _Iniqsdat, _Inoldat
     .global _Ouidsdat, _Ouiqsdat, _Ouoldat, _Phase32, _Thetagen
     .global _Vectrot, _Idscont, _Iqscont, _Sintab
     .global _Outstat
     .global _Mode, _Counter
     .global _Ib_fl, _Ic_fl, _Wrr_fl, _Wr_fl, _Kpd_fl, _Wcd_fl
     .global _Kpq_fl, _Wcq_fl, _Kps_fl, _Wcs_fl, _Xir_fl, _Idr_fl
     .global _Iqr_fl, _Wbase, _Ts1, _Xs_fl, _Xs_fl_n
     .global _Ksl, _Ksmax
     .global _Ids_rated, _Kslip, _Iqs_rated
     .global _Kpd_fx, _Kid_fx, _Kpq_fx, _Kiq_fx, _Xir_fx, _Ib_fx, _Ic_fx
     .global _Wr_fx, _Idr_fx, _Iqr_fx, Ts1_fx
     .global _Vdr_fx_n, _Vqr_fx_n, _Idr_fx_n, _Iqr_fx_n
     .global _Eds_fx, _Eqs_fx, _YDH, _YDL, _YQH, _YQL, _We
     .global _Vdr_fl_n, _Vqr_fl_n, _Idr_fl_n, _Iqr_fl_n
```

```
.global _Eds_fl, _Eqs_fl, _Ews_fl, _Thetae_fl, _We_fl
.global SIN, COS, TEMP0, TEMP1, We, _DT  ; For thetagen
.global _Thetah, _Thetal, Idse, Idss, Iqse, Iqss  ; for vectrot()
.bss   _Mode, 1
.bss   _Counter, 1
.bss   _Ib_fl, 2
.bss   _Ic_fl, 2
.bss   _Wrr_fl, 2
.bss   _Wr_fl, 2
.bss   _Kpd_fl, 2
.bss   _Wcd_fl, 2

.bss   _Kpq_fl, 2
.bss   _Wcq_fl, 2
.bss   _Kps_fl, 2
.bss   _Wcs_fl, 2
.bss   _Idr_fl, 2
.bss   _Iqr_fl, 2
.bss   _Wbase, 2
.bss   _Ts1, 2
.bss   _Xs_fl, 2
.bss   _Xs_fl_n, 2
.bss   _Ksl, 1
.bss   _Ksmax, 2
.bss   _Ids_rated, 2
.bss   _Kslip, 2
.bss   _Iqs_rated, 2
.bss   _Kpd_fx, 1
.bss   _Kid_fx, 1
.bss   _Kpq_fx, 1
.bss   _Kiq_fx, 1
.bss   _Xir_fx, 1
.bss   _Ib_fx, 1
.bss   _Ic_fx, 1
.bss   _Wr_fx, 1
.bss   _Idr_fx, 1
.bss   _Iqr_fx, 1
.bss   _Ts1_fx, 1
.bss   _Vdr_fx_n, 1
.bss   _Vqr_fx_n, 1
.bss   _Idr_fx_n, 1
.bss   _Iqr_fx_n, 1
.bss   _Eds_fx, 1
.bss   _Eqs_fx, 1
.bss   _YDH, 1
.bss   _YDL, 1
.bss   _YQH, 1
.bss   _YQL, 1
.bss   _We, 1
.bss   _Vdr_fl_n, 2
.bss   _Vqr_fl_n, 2
.bss   _Idr_fl_n, 2
.bss   _Iqr_fl_n, 2
.bss   _We_fl, 2
.bss   _Eds_fl, 2
.bss   _Eqs_fl, 2
.bss   _Ews_fl, 2
.bss   _Thetae_fl, 2
.bss   SIN, 1
.bss   COS, 1
.bss   TEMP0, 1
.bss   TEMP1, 1
.bss   _DT, 1
.bss   We, 1
.bss   _Thetah, 1
```

```
    .bss   _Thetal, 1
    .bss   Idse, 1
    .bss   Idss, 1
    .bss   Iqse, 1
    .bss   Iqss, 1
_Init:

TBL90:  .set   512          ; 90 deg offset in sin/cos table in thetagen
    RET


;*************************************************************
; Subroutine Reading Mode of Tuning
;*************************************************************
    .title "inmod.asm"
    .global _Mode
    .global _Inmod
    .text
_Inmod:
    LDPK   6
    IN     _Mode, PA0
    RET

;*************************************************************
; Subroutine Reading Ids Controller Tuning States and Parameters
;*************************************************************
        .title "inidsdat.asm"
        .mmregs
        .global _Init, _Inidsdat, _Idscont   ; Routines sharing data
        .global _Phase32, _Thetagen
        .global _Iqscont, _Vectrot, _Ouidsdat
        .global _Ib_fl, _Ic_fl, _Wr_fl, _Kpd_fl, _Wcd_fl, _Ts1
        .global _Thetal, _Thetah, _YDH, _YDL, _Idr_fl, _Kslip
        .text
_Inidsdat:
        LDPK   6
        IN     _Ib_fl, PA2
        IN     _Ib_fl+1, PA2
        IN     _Ic_fl, PA2
        IN     _Ic_fl+1, PA2
        IN     _Kpd_fl, PA1
        IN     _Kpd_fl+1, PA1
        IN     _Wcd_fl, PA1
        IN     _Wcd_fl+1, PA1
        IN     _Ts1, PA1
        IN     _Ts1+1, PA1
        IN     _Idr_fl, PA1
        IN     _Idr_fl+1, PA1
        IN     _Kslip, PA1
        IN     _Kslip+1, PA1
        IN     _Thetal, PA3      ; Thetae LSW
        IN     _Thetah, PA3       ; Thetae MSW
        IN     _YDH, PA3
        IN     _YDL, PA3
        RET

;*************************************************************
; Subroutine Reading Iqs Controller Paramters and States
;*************************************************************
        .title "iniqsdat.asm"
        .mmregs
        .global _Init, _Idscont, _Iniqsdat   ; Routines sharing data
        .global _Phase32, _Thetagen
        .global _Iqs_cont, _Vectrot, _Ouidsdat
        .global _Ib_fl, _Ic_fl, _Wr_fl, _Kpd_fl,_Wcd_fl, _Kpq_fl, _Wcq_fl
```

```
        .global _Xir_fl, _Ts1, _Thetae_fx, _Ids_rated, _Kslip
        .global _YDH, _YDL, _YQH, YQL, _Iqr_fl
        .text
_Iniqsdat:
        LDPK   6
        IN     _Ib_fl, PA2
        IN     _Ib_fl+1, PA2
        IN     _Ic_fl, PA2
        IN     _Ic_fl+1, PA2
        IN     _Wr_fl, PA2
        IN     _Wr_fl+1, PA2
        IN     _Kpd_fl, PA1
        IN     _Kpd_fl+1, PA1
        IN     _Wcd_fl, PA1
        IN     _Wcd_fl+1, PA1
        IN     _Kpq_fl, PA1
        IN     _Kpq_fl+1, PA1
        IN     _Wcq_fl, PA1
        IN     _Wcq_fl+1, PA1
        IN     _Ts1, PA1
        IN     _Ts1+1, PA1
        IN     _Iqr_fl, PA1
        IN     _Iqr_fl+1, PA1
        IN     _Ids_rated, PA1
        IN     _Ids_rated+1, PA1
        IN     _Kslip, PA1
        IN     _Kslip+1, PA1
        IN     _Thetal, PA3
        IN     _Thetah, PA3
        IN     _YDH, PA3
        IN     _YDL, PA3
        IN     _YQH, PA3
        IN     _YQL, PA3
        RET


;*************************************************************
; Subroutine Reading Speed Controller States and Parameters
;*************************************************************
        .title "inoldat.asm"
        .mmregs
        .global _Init, _Idscont, _Inoldat   ; Routines sharing data
        .global _Phase32, _Thetagen
        .global _Iqscont, _Vectrot, _Ouidsdat
        .global _Ib_fl, _Ic_fl, _Wr_fl, _Kpd_fl,_Wcd_fl, _Kpq_fl, _Wcq_fl
        .global _Kps_fl, _Wcs_fl, _Xir_fl, _Ts1, _Thetah, _Thetal, _Wrr_fl
        .global _YDH, _YDL, _YQH, _YQL, _Xs_fl, _Counter, _Kslip, _Iqs_rated
        .global _Ids_rated, _Wbase
        .text
_Inoldat:
        LDPK   6
        IN     _Ib_fl, PA2
        IN     _Ib_fl+1, PA2
        IN     _Ic_fl, PA2
        IN     _Ic_fl+1, PA2
        IN     _Wr_fl, PA2
        IN     _Wr_fl+1, PA2
        IN     _Kpd_fl, PA1
        IN     _Kpd_fl+1, PA1
        IN     _Wcd_fl, PA1
        IN     _Wcd_fl+1, PA1
        IN     _Kpq_fl, PA1
        IN     _Kpq_fl+1, PA1
        IN     _Wcq_fl, PA1
        IN     _Wcq_fl+1, PA1
```

```
        IN    _Kps_fl, PA1
        IN    _Kps_fl+1, PA1
        IN    _Wcs_fl, PA1
        IN    _Wcs_fl+1, PA1
        IN    _Ts1, PA1
        IN    _Ts1+1, PA1
        IN    _Kslip, PA1
        IN    _Kslip+1, PA1
        IN    _Iqs_rated, PA1
        IN    _Iqs_rated+1, PA1
        IN    _Ids_rated, PA1
        IN    _Ids_rated+1, PA1
        IN    _Wrr_fl, PA1
        IN    _Wrr_fl+1, PA1
        IN    _Wbase, PA1
        IN    _Wbase+1, PA1
        IN    _Thetal, PA3
        IN    _Thetah, PA3
        IN    _YDH, PA3
        IN    _YDL, PA3
        IN    _YQH, PA3
        IN    _YQL, PA3
        IN    _Xs_fl, PA3
        IN    _Xs_fl+1, PA3
        IN    _Idr_fl, PA3
        IN    _Idr_fl+1, PA3
        IN    _Iqr_fl, PA3
        IN    _Iqr_fl+1, PA3
        IN    _Ews_fl, PA3
        IN    _Ews_fl+1, PA3
        IN    _Counter, PA3
        RET

;*************************************************************
; Three-Phase to Two-Phase Current Conversion Routine
;*************************************************************
    .title "phase32.asm"
    .global _Phase32, _init, _thetagen, _vectrot
    .global _Ib_fx, _Ic_fx, Idss, Iqss
    .global _SQR1O3
    .text
_Phase32:
    SPM   1
    ZALH  _Ib_fx
    NEG
    SUBH  _Ic_fx
    SACH  Iqss              ; Iqss = -ib - ic (= ia)
    ZAC
    LT    _Ib_fx
    MPY   _SQR1O3
    APAC
    NEG
    LT    _Ic_fx
    MPY   _SQR1O3
    APAC
    SACH  Idss              ;Idss = (-ib + ic)/(sqrt 3)
    RET

;*************************************************************
; Rotor Electrical Angle and Unit Vector Generation Routine
;*************************************************************
        .title "thetagen.asm"
        .global _Thetagen, _Sintab, _init, _vectrot
        .global _Wr_fx, _Ksl, _Iqr_fx, _We, _Thetah, _Thetal
        .global SIN, COS, TEMP0, TBL90, TEMP1
```

```
        .global _DT
        .text
_Thetagen:
        SPM   0
        LDPK  6
        ZALH  _Wr_fx        ;We computation.
        LT    _Ksl
        MPY   _Iqr_fx
        APAC
        APAC
        SACH  _We           ;We = Wr + Kslip*Iqe

; ---------- Calc. angle theta
BP1:  ROVM              ;Reset overflow mode
        ZALH  _Thetah       ;
        ADDS  _Thetal
        LT    _We
        MPY   _DT
        APAC              ;Theta = Theta + We*DT
        SACL  _Thetal
        SACH  _Thetah
;
;---------- Get SIN(theta), COS(theta) from look-up table
        SSXM              ;Set sign extension
        LT    _Thetah
        MPYK  2
        PAC
        SACL  TEMP1
        LT    TEMP1
        MPYK  2048
        PAC
        SACH  TEMP0
        LAC   TEMP0
        BGEZ  POSIN
        ADLK  1024          ;If negative, add 180 degrees
        ANDK  1023
;
NESIN: SOVM
        ADLK  SINTBL
        TBLR  SIN
        ADLK  TBL90
        TBLR  COS
        LAC   SIN
        NEG
        SACL  SIN
        LAC   COS
        NEG
        SACL  COS
        B     SINEND

POSIN: SOVM             ;Set overflow mode
        ADLK  SINTBL
        TBLR  SIN
        ADLK  TBL90
        TBLR  COS

SINEND: RET

;*************************************************************
; Inverse Vector Rotation Routine
;*************************************************************
    .title "vectrot.asm"
    .global _Vectrot, _init, _sintab. _thetagen. _ids_cont. _iqs_cont
    .global SIN, COS, Idss, Iqss, Idse. Iqse. _Idr_fx. _Iqr_fx
    .global _Eds_fx, _Eqs_fx
```

```
    .text
_Vectrot:
    SPM   1
    ZAC
    LT    SIN
    MPY   Idss
    APAC
    NEG
    LT    COS
    MPY   Iqss
    APAC
    SACH  Iqse            ;Iqse = iqss*cos - idss*sin
    ZAC
    LT    SIN
    MPY   Iqss
    APAC
    LT    COS
    MPY   Idss
    APAC
    SACH  Idse            ;Idse = iqss*sin + idss*cos
    ZALH  _Idr_fx
    SUBH  Idse
    SACH  _Eds_fx           ; eds = idref - id
    ZALH  _Iqr_fx
    SUBH  Iqse
    SACH  _Eqs_fx           ; eqs = iqref - iq
    RET

;*************************************************************
; Ids Synchronous Current Controller Routine
;*************************************************************
        .title  "idscont.asm"
        .global _Idscont, _Init
        .global _Kpd_fx, _Eds_fx, _Vdr_fx_n. _Ts1_fx, _YDH, _YDL
        .global _Kid_fx
        .bss   Vdh, 1       ; Intermediate Vhigh
        .bss   Vdl, 1       ; Intermediate Vlow
        .bss   Vd, 1        ; Final V word
        .bss   Fd, 1        ; Sign flag for V
        .bss   VDH, 1
        .bss   VDL, 1
VLIM    .set   972      ; Set limiter at 95% of max. value
                        ; (972 = 2 * 0.95 * 512)
        .text
_Idscont:
        LDPK   6
        SPM    1
        SSXM
        LACK   1        ; Set positive sign flag by default
        SACL   Fd
        ZAC
        LT     _Kpd_fx
        MPY    _Eds_fx      ; vp = Kp * e
        LT     _Kid_fx
        MPYA   _YDH         ; vi = Ki * y (integral of error)
        SPM    3
        APAC            ; <ACC> = v = vp + vi
        SPM    1
        SACH   VDH
        SACL   VDL          ; Store V
        BGEZ   Nneg         ; Set sign flag if V < 0
        LACK   0
        SACL   Fd
        ZALH   VDH
        ADDS   VDL
```

```
Nneg:  ABS              ; Find range of V
       SACH   Vdh
       SACL   Vdl
       ZALS   Vdh
       SUBK   1
       BLEZ   Upscl      ; If |V| > 0.95, Set |V| = 0.95
       LACK   VLIM
       SACL   Vd
       B      Apsgn      ; Apply sign now
Upscl: ZALH   Vdh        ; Upscale the value of V in linear range
       ADDS   Vdl        ; For better precision
       RPTK   8
       SFL
       SACH   Vd         ; Vd has the value of |V|
Apsgn: LAC    Fd         ; Is the sign of V negative?
       BGZ    Ncmp       ; No ==> Don't negate |V|
       ZALH   Vd         ; Else V = -|V|
       NEG
       SACH   Vd
Ncmp:  ZALH   Vd         ; Store final limiter output
       SACH   _Vdr_fx_n   ;
       ZAC               ; Implement Integral of error
       LT     _Eds_fx    ; Dy = E * Ts
       MPY    _Ts1_fx
       APAC
       ADDS   _YDL       ; Y(new) = Y(old) + Dy
       ADDH   _YDH
       SACH   _YDH
       SACL   _YDL       ; Store new Y in buffer
       RET


;*************************************************************
; Iqs Synchronous Current Controller Routine
;*************************************************************
       .title "iqscont.asm"
       .global _Iqscont, _Init
       .global _Kpq_fx, _Eqs_fx, _Vqr_fx_n, _Ts1_fx, _YQH, _YQL
       .global _Kiq_fx
       .bss   Vqh, 1     ; Intermediate Vhigh
       .bss   Vql, 1     ; Intermediate Vlow
       .bss   Vq, 1      ; Final V word
       .bss   Fq, 1      ; Sign flag for V
       .bss   VQH, 1
       .bss   VQL, 1
VLIM   .set   972        ; Set limiter at 95% of max. value
                         ; (972 = 2 * 0.95 * 512)
       .text
_Iqscont:
       LDPK   6
       SPM    1
       SSXM
       LACK   1          ; Set positive sign flag by default
       SACL   Fq
       ZAC
       LT     _Kpq_fx
       MPY    _Eqs_fx    ; vp = Kp * e
       LT     _Kiq_fx
       MPYA   _YQH       ; vi = Ki * y (integral of error)
       SPM    3
       APAC
       SPM    1
       SACH   VQH
       SACL   VQL        ; Store V
       BGEZ   Nqneg      ; Set sign flag if V < 0
```

```
        LACK   0
        SACL   Fq
        ZALH   VQH
        ADDS   VQL
Nqneg:  ABS             ; Find range of V
        SACH   Vqh
        SACL   Vql
        ZALS   Vqh
        SUBK   1
        BLEZ   Upsclq       ; If |V| > 0.95, Set |V| = 0.95
        LACK   VLIM
        SACL   Vq
        B      Apsgnq       ; Apply sign now
Upsclq: ZALH   Vqh          ; Upscale the value of V in linear range
        ADDS   Vql          ; For better precision
        RPTK   8
        SFL
        SACH   Vq           ; Vd has the value of |V|
Apsgnq: LAC    Fq           ; Is the sign of V negative?
        BGZ    Nqcmp        ; No ==> Don't negate |V|
        ZALH   Vq           ; Else V = -|V|
        NEG
        SACH   Vq
Nqcmp:  ZALH   Vq           ; Store final limiter output
        SACH   _Vqr_fx_n    ;
        ZAC             ; Implement Integral of error
        LT     _Eqs_fx      ; Dy = E * Ts
        MPY    _Ts1_fx
        APAC
        ADDS   _YQL         ; Y(new) = Y(old) + Dy
        ADDH   _YQH
        SACH   _YQH
        SACL   _YQL         ; Store new Y in buffer
        RET

;*************************************************************
; Ids Tuning States and Parameters Output Routine
;*************************************************************
    .title "ouidsdat.asm"
    .global _Init, _Ouidsdat
    .global _Vdr_fl_n, _Eds_fl, _Thetae_fl
    .text
_Ouidsdat:
    LDPK   6
    OUT    _Vdr_fl_n, PA4
    OUT    _Vdr_fl_n+1, PA4
    OUT    _Eds_fl, PA4
    OUT    _Eds_fl+1, PA4
    OUT    _Thetae_fl, PA4      ;Thetae LSW
    OUT    _Thetae_fl+1, PA4    ;Thetae MSW
    OUT    _We_fl, PA4
    OUT    _We_fl+1, PA4
    RET

;*************************************************************
; Iqs Controller States and Parameters Output Routine
;*************************************************************
    .title "ouiqsdat.asm"
    .global _Init, _Ouiqsdat
    .global _Vdr_fl_n, _Vqr_fl_n, _Eds_fl, _Eqs_fl, _Thetae_fl
    .text
_Ouiqsdat:
    LDPK   6
    OUT    _Vdr_fl_n, PA4
    OUT    _Vdr_fl_n+1, PA4
```

```
    OUT   _Vqr_fl_n, PA4
    OUT   _Vqr_fl_n+1, PA4
    OUT   _Eds_fl, PA4
    OUT   _Eds_fl+1, PA4
    OUT   _Eqs_fl, PA4
    OUT   _Eqs_fl+1, PA4
    OUT   _Thetae_fl, PA4
    OUT   _Thetae_fl+1, PA4
    OUT   _We_fl, PA4
    OUT   _We_fl+1, PA4
    RET

;*************************************************************
; Speed Controller States and Parameters Output Routine
;*************************************************************
    .title "ouoldat.asm"
    .global _Init, _Ouoldat
    .global _Vdr_fl_n, _Vqr_fl_n, _Idr_fl_n, _Iqr_fl_n
    .global _Eds_fl, _Eqs_fl, _Ews_fl, _Thetae_fl
    .text
_Ouoldat:
    LDPK  6
    OUT   _Vdr_fl_n, PA4
    OUT   _Vdr_fl_n+1, PA4
    OUT   _Vqr_fl_n, PA4
    OUT   _Vqr_fl_n+1, PA4
    OUT   _Eds_fl, PA4
    OUT   _Eds_fl+1, PA4
    OUT   _Eqs_fl, PA4
    OUT   _Eqs_fl+1, PA4
    OUT   _Ews_fl, PA4
    OUT   _Ews_fl+1, PA4
    OUT   _Thetae_fl, PA4
    OUT   _Thetae_fl+1, PA4
    OUT   _We_fl, PA4
    OUT   _We_fl+1, PA4
    RET

/*************************************************************/
/* TMS 320C25 Simulator Simulation Command File              */
/*************************************************************/
load stimer
ma 0,2,1,iport
ma 1,2,1,iport
ma 2,2,1,iport
ma 3,2,1,ioport
ma 4,2,1,oport
mc 0,2,mode.dat,read
mc 1,2,inpar.dat,read
mc 2,2,insim.dat,read
mc 3,2,cstate.dat,read
mc 4,2,outsim.dat,write
go Outstat
mi 3,2,read
mc 3,2,cstate.dat,write
go c_int0-1
quit
```

APPENDIX D.4 C-Based Interface Routines for Controller Tuning

```
/*************************************************************/
/* PAR2HEXN.C - This routine converts floating point numbers to IEEE   */
/*          single precision floating point format, with LSW and   */
/*          MSW in reversed order, for being read by C25 Simulator */
/*************************************************************/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
char  ieeesp[9];
void main()
{
/* This function is called to perform floating point to hex format conversion */
char *f2x(float parm);
   /* parm  are the parameters in floating point format to be converted to hex */
   float parm1, parm2, parm3, parm4, parm5, parm6, parm7, parm8, parm9;
   float parm10, parm11, parm12, tstid, tstiq, tstwr ;
   int mode;
   /* num1 - num11 are the equivalent Hex number strings  */
   char num1[9], num2[9], num3[9], num4[9], num5[9], num12[9];
   char num6[9], num7[9], num8[9], num9[9], num10[9], num11[9];
   /* num1h, etc are the MSW and LSW s of the respective Hex numbers, separated */
   /* for being read in 16 bit format through input port of C25. This is a requirement  */
   char num1h[5], num1l[5], num2h[5], num2l[5], num3h[5], num3l[5];
   char num4h[5], num4l[5], num5h[5], num5l[5], num6h[5], num6l[5];
   char num7h[5], num7l[5], num8h[5], num8l[5], num9h[5], num9l[5];
   char num10h[5], num10l[5], num11h[5], num11l[5], num12h[5], num12l[5];
   int lenstr;
   /* CONPAR.DAT is the ES-exported parameter data file in floating point format */
   /* INPAR.DAT is the C25 Simulator input data file, in Hex format, */
   /*  to be connected to its input port for 16 bit data input     */
   /* MODE.DAT Stores the mode of simulation          */
   /* TSTSIG.DAT stores the prev. test (ids/iqs/wr) signal value */
   static char fname_in1[] = "CONPAR.DAT", fname_in2[] = "MODE.DAT";
   static char fname_in3[] = "TSTSIG.DAT", fname_out[] = "INPAR.DAT";
   FILE *input_fil1, *input_fil2, *input_fil3, *output_file;
   printf("PASSING CONTROL DATA FROM ES TO SIMULATOR IN HEX FORMAT\n\n");
   if ((input_fil1 = fopen(fname_in1, "r")) == NULL)
            {
            printf("Error opening ES parm. data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   if ((input_fil2 = fopen(fname_in2, "r")) == NULL)
            {
            printf("Error opening ES mode data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }

   if ((input_fil3 = fopen(fname_in3, "r")) == NULL)
            {
            printf("Error opening test signal data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   if ((output_file = fopen(fname_out, "w")) == NULL)
            {
            printf("Error opening Simulator input file.\n");
```

```
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
/* Mode of simulation read below:  */
fscanf(input_fil2, "%d", &mode);
/* Floating point data parameters read from input file below        */
        fscanf(input_fil1, "%f %f %f %f", &parm1, &parm2, &parm3, &parm4);
        fscanf(input_fil1, "%f", &parm5);
if ((mode == 2) || (mode == 3) || (mode == 4))
        {
        fscanf(input_fil1, " %f %f %f", &parm6, &parm7, &parm8);
        }
if ((mode == 3) || (mode == 4))
        {
        fscanf(input_fil1, "%f %f %f %f", &parm9, &parm10, &parm11, &parm12);
        }
if (mode == 1)
        {
        printf("Kp for this pass of Id loop opt. is. %f\n", parm1);
        printf("Ki for this pass of Id loop opt. is. %f\n", parm1 * parm2);
        fscanf(input_fil3, "%f", &tstid);
        fclose (input_fil3);
        fopen (fname_in3, "w");
        if (tstid != 0)
          {
          parm4 = 0;
          }
        fprintf(input_fil3, "%f", parm4);
        }
else if (mode == 2)
        {
        printf("Kp for this pass of Iq loop opt. is. %f\n", parm3);
        printf("Ki for this pass of Iq loop opt. is. %f\n", parm3 * parm4);
        fscanf(input_fil3, "%f", &tstiq);
        fclose (input_fil3);
        fopen (fname_in3, "w");
        if (tstiq != 0)
          {
          parm6 = 0;
          }
        fprintf(input_fil3, "%f", parm6);
        }

else if ((mode == 3) || (mode == 4))
        {
        if (mode == 3)
          {
          printf("Kp for this pass of speed loop opt. is, %f\n", parm5);
          printf("Ki for this pass of speed loop opt. is, %f\n", parm5 * parm6);
          }
        fscanf(input_fil3, "%f", &tstwr);
        fclose (input_fil3);
        fopen (fname_in3, "w");
        if (mode == 3)
          {
          if (tstwr != 10)
                {
                parm11 = 10;
                }
          else
                {
                parm11 = 0;
                }
          }
```

```
        else if (mode == 4)
          {
          if(tstwr == 0)
                  {
                  parm11 = 1*parm11;
                  }
          else
                  {
                  parm11 = 0;
                  }
          }
        fprintf(input_fil3, "%f", parm11);
        }
/* f2x function called below to convert the parameters to Hex long word format */
        f2x( parm1 );
        strcpy( num1, ieeesp );
        f2x( parm2 );
        strcpy( num2, ieeesp );
        f2x( parm3 );
        strcpy( num3, ieeesp );
        f2x( parm4 );
        strcpy( num4, ieeesp );
        f2x( parm5 );
        strcpy( num5, ieeesp );
if ((mode == 2) || (mode == 3) || (mode == 4))
        {
        f2x( parm6 );
        strcpy( num6, ieeesp );
        f2x( parm7 );
        strcpy( num7, ieeesp );
        f2x( parm8 );
        strcpy( num8, ieeesp );
        }

if ((mode == 3) || (mode == 4))
        {
        f2x( parm9 );
        strcpy( num9, ieeesp );
        f2x( parm10 );
        strcpy( num10, ieeesp );
        f2x( parm11 );
        strcpy( num11, ieeesp );
        f2x( parm12 );
        strcpy( num12, ieeesp );
        }
/* LSWs and MSWs are separated for each of the long words below, and output to the */
/* output file.                                                                    */
        strcpy( num1h , num1 + 4 );
        *(num1 + 4) = '\0';
        strcpy(num1l , num1);
        strcpy( num2h , num2 + 4 );
        *(num2 + 4) = '\0';
        strcpy(num2l , num2);
        strcpy( num3h , num3 + 4 );
        *(num3 + 4) = '\0';
        strcpy(num3l , num3);
        strcpy( num4h , num4 + 4 );
        *(num4 + 4) = '\0';
        strcpy(num4l , num4);
        strcpy( num5h , num5 + 4 );
        *(num5 + 4) = '\0';
        strcpy(num5l , num5);
        fprintf(output_file, "%4s\n", num1l);
        fprintf(output_file, "%4s\n", num1h);
        fprintf(output_file, "%4s\n", num2l);
```

```
            fprintf(output_file, "%4s\n", num2h);
            fprintf(output_file, "%4s\n", num3l);
            fprintf(output_file, "%4s\n", num3h);
            fprintf(output_file, "%4s\n", num4l);
            fprintf(output_file, "%4s\n", num4h);
            fprintf(output_file, "%4s\n", num5l);
            fprintf(output_file, "%4s\n", num5h);
   if ((mode == 2) || (mode == 3) || (mode == 4))
            {
            strcpy( num6h , num6 + 4 );
            *(num6 + 4) = '\0';
            strcpy(num6l , num6);
            strcpy( num7h , num7 + 4 );
            *(num7 + 4) = '\0';
            strcpy(num7l , num7);
            strcpy( num8h , num8 + 4 );
            *(num8 + 4) = '\0';
            strcpy(num8l , num8);
            fprintf(output_file, "%4s\n", num6l);
            fprintf(output_file, "%4s\n", num6h);
            fprintf(output_file, "%4s\n", num7l);
            fprintf(output_file, "%4s\n", num7h);
            fprintf(output_file, "%4s\n", num8l);
            fprintf(output_file, "%4s\n", num8h);
            }
   if ((mode == 3) || (mode == 4))
            {
            strcpy( num9h , num9 + 4 );
            *(num9 + 4) = '\0';
            strcpy(num9l , num9);
            strcpy( num10h , num10 + 4 );
            *(num10 + 4) = '\0';
            strcpy(num10l , num10);
            strcpy( num11h , num11 + 4 );
            *(num11 + 4) = '\0';
            strcpy(num11l , num11);
            strcpy( num12h , num12 + 4 );
            *(num12 + 4) = '\0';
            strcpy(num12l , num12);
            fprintf(output_file, "%4s\n", num9l);
            fprintf(output_file, "%4s\n", num9h);
            fprintf(output_file, "%4s\n", num10l);
            fprintf(output_file, "%4s\n", num10h);
            fprintf(output_file, "%4s\n", num11l);
            fprintf(output_file, "%4s\n", num11h);
            fprintf(output_file, "%4s\n", num12l);
            fprintf(output_file, "%4s\n", num12h);
            }
   fcloseall();
}
/* This is the function doing the conversion from floating point  to Hex   */
char *f2x( float parm )
{
char *hexstr8( unsigned long num, char *buffer );  /* Prototype */
   union
   {
    float f;          /* As float */
    unsigned long ul;   /* As unsigned long (for binary conversion) */
   } msbin, ieee;
   ieee.f = parm;
   /* Convert to Hexadecimal string in reversed word order */
   hexstr8( ieee.ul, ieeesp );
    return ieeesp;
}
/* Converts unsigned long to string of 8 hex characters.   */
```

```
/*  in reversed word order                          */
char *hexstr8( unsigned long num, char *buffer )
{
   char tmp[9];
   unsigned long len;
   memset( buffer, '0', 9 );
   len = strlen( ultoa( num, tmp, 16 ) );
   strcpy( buffer + 8 - len, tmp + 4 );
   *(tmp + 4) = '\0';
   strcpy(buffer + 12 - len, tmp);
   return buffer;
}

/*****************************************************************/
/* SMP2HEX.C - This routine converts floating point numbers to IEEE   */
/*          single precision floating point format, with LSW and   */
/*          MSW in reversed order, for being read by C25 Simulator */
/*****************************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
char  ieeesp[9];
int M;
void main()
{
/* This function is called to perform floating point to hex format conversion */
char *f2x(float parm);
   /* parm  are the parameters in floating point format to be converted to hex */
   float parm1, parm2, parm3;
   /* Mode is the Mode of simulation  */
   /*int mode; */
   /* num1 - num3 are the equivalent Hex number strings  */
   char num1[9], num2[9], num3[9];
   /* num1h, etc are the MSW and LSW s of the respective Hex numbers, separated */
   /* for being read in 16 bit format through input port of C25. This is a requirement  */
   char num1h[5], num1l[5], num2h[5], num2l[5], num3h[5], num3l[5];
   char nul_str[] = "00000000";
   int lenstr, i;
   /* EXPOR.T is the SIMNON exported sampled data file in floating point format */
   /* INSIM.DAT is the C25 Simulator input data file, in Hex format,    */
   /*  to be connected to its input port for 16 bit data input          */
   static char fname_in1[] = "EXPOR.T", fname_in2[] = "MODE.DAT", fname_out[] = "INSIM.DAT";
   FILE *input_fil1, *input_fil2, *output_file;
   if ((input_fil1 = fopen(fname_in1, "r")) == NULL)
            {
            printf("Error opening exported data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   if ((input_fil2 = fopen(fname_in2, "r")) == NULL)
            {
            printf("Error opening mode data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   if ((output_file = fopen(fname_out, "w")) == NULL)
            {
            printf("Error opening Simulator input port file.\n");
            printf("Press any key to continue.\n");
            getche();
```

```
        exit(0);
        }
  printf("CONVERTING SAMPLED PLANT DATA TO HEX FORMAT FOR CONTROLLER..\n");
  /* Mode read here  */
  fscanf(input_fil2, "%d", &M);
/* Floating point data parameters read from input file below       */
  if (M == 1)
        {
        fscanf(input_fil1, "%f %f", &parm1, &parm2);
        }
  else if ((M == 2) || (M == 3) || (M == 4))
        {
        fscanf(input_fil1, "%f %f %f", &parm1, &parm2, &parm3);
        }
  /* f2x function called below to convert the parameters to Hex long word format */
  if (M == 1)
        {
        if (parm1 == 0)
          {
          for (i=0; i<8; i++)
            {
            num1[i] = '0';
            }
          num1[8] = '\0';
          }
        else
          {
          f2x( parm1 );
          strcpy( num1, ieeesp );
          }
        if (parm2 == 0)
          {
          for (i=0; i<8; i++)
            {
            num2[i] = '0';
            }
          num2[8] = '\0';
          }
        else
          {
          f2x( parm2 );
          strcpy( num2, ieeesp );
          }
        }
  else if ((M == 2) || (M == 3) || (M == 4))
        {
        if (parm1 == 0)
          {
          for (i=0; i<8; i++)
            {
            num1[i] = '0';
            }
          num1[8] = '\0';
          }
        else
          {
          f2x( parm1 );
          strcpy( num1, ieeesp );
          }
        if (parm2 == 0)
          {
          for (i=0; i<8; i++)
            {
            num2[i] = '0';
            }
```

```
            num2[8] = '\0';
            }
          else
            {
            f2x( parm2 );
            strcpy( num2, ieeesp );
            }
          if (parm3 == 0)
            {
            for (i=0; i<8; i++)
              {
              num3[i] = '0';
              }
            num3[8] = '\0';
            }
          else
            {
            f2x( parm3 );
            strcpy( num3, ieeesp );
            }
    }
  /* LSWs and MSWs are separated for each of the long words below, and output to the */
  /* output file.                                                                    */
  if (M == 1)
          {
          strcpy( num1h , num1 + 4 );
          *(num1 + 4) = '\0';
          strcpy(num1l , num1);
          strcpy( num2h , num2 + 4 );
          *(num2 + 4) = '\0';
          strcpy(num2l , num2);
          fprintf(output_file, "%4s\n", num1l);
          fprintf(output_file, "%4s\n", num1h);
          fprintf(output_file, "%4s\n", num2l);
          fprintf(output_file, "%4s\n", num2h);
          }
  else if ((M == 2) || (M == 3) || (M == 4))
          {
          strcpy( num1h , num1 + 4 );
          *(num1 + 4) = '\0';
          strcpy(num1l , num1);
          strcpy( num2h , num2 + 4 );
          *(num2 + 4) = '\0';
          strcpy(num2l , num2);
          strcpy( num3h , num3 + 4 );
          *(num3 + 4) = '\0';
          strcpy(num3l , num3);

          fprintf(output_file, "%4s\n", num1l);
          fprintf(output_file, "%4s\n", num1h);
          fprintf(output_file, "%4s\n", num2l);
          fprintf(output_file, "%4s\n", num2h);
          fprintf(output_file, "%4s\n", num3l);
          fprintf(output_file, "%4s\n", num3h);
          }

  fcloseall();

}
/* This is the function doing the conversion from floating point  to Hex  */
char *f2x( float parm )
{
char *hexstr8( unsigned long num, char *buffer );  /* Prototype */
  union
  {
```

```
    float f;        /* As float */
    unsigned long ul;  /* As unsigned long (for binary conversion) */
   } msbin, ieee;
   ieee.f = parm;
   /* Convert to Hexadecimal string in reversed word order */
   hexstr8( ieee.ul, ieeesp );
    return ieeesp;
}
/* Converts unsigned long to string of 8 hex characters.  */
/*  in reversed word order                                 */
char *hexstr8( unsigned long num, char *buffer )
{
   char tmp[9];
   unsigned long len;
   memset( buffer, '0', 9 );
   len = strlen( ultoa( num, tmp, 16 ) );
   strcpy( buffer + 8 - len, tmp + 4 );
   *(tmp + 4) = '\0';
   strcpy(buffer + 12 - len, tmp);
   return buffer;
}

/*********************************************************/
/* HEX2FLO.C  -- Conversion routine for IEEE Single precision hex to */
/*          floating point ASCII format                  */
/*          Now also includes conversions for various mode of  */
/*          simulation.                                  */
/*********************************************************/

#include <stdio.h>
#include <math.h>
#include <io.h>
#include <string.h>
#include <stdlib.h>
void main()
{
   /* Function definition for hex to float conversion */
   float x2f(char *lsw_str, char *msw_str);
   /* parm are the converted float parameter values */
   float parm1, parm2, parm3, parm4, parm5, parm6, parm7, parm8;
   int mode;  /* Mode of simulation  */
   /* numh, numl are the two words (LSW, MSW) of the 32 bit hex number */
   char num1h[9], num1l[5], num2h[9], num2l[5], num3h[9], num3l[5];
   char num4h[9], num4l[5], num5h[9], num5l[5], num6h[9], num6l[5];
   char num7h[9], num7l[5], num8h[9], num8l[5];
   /* OUTSIM.DAT is the file output by TMS320C25 Simulator */
   /* IMPOR.T is the 'import' file used by SIMNON / ES    */
   /* OUTSIM.DAT has data in hex format, IMPOR.T has it in */
   /* floating pt ASCII format                           */
   /* MODE.DAT has the mode of simulation                */
   static char fname_in1[] = "OUTSIM.DAT", fname_in2[] = "MODE.DAT", fname_out[] = "IMPOR.T";
   FILE *input_fil1, *input_fil2, *output_file;
   if ((input_fil1 = fopen(fname_in1, "r")) == NULL)
            {
            printf("Error opening Simulator output data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   if ((input_fil2 = fopen(fname_in2, "r")) == NULL)
            {
            printf("Error opening Mode data file.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
```

```
        }
if ((output_file = fopen(fname_out, "w")) == NULL)
        {
        printf("Error opening SIMNON import file.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
/* Mode of simulation read below   */
fscanf(input_fil2, "%d", &mode);
/* Data values read below from the file OUTPUT.DAT  */
if (mode == 1)
        {
        fscanf(input_fil1, "%4s %4s %4s %4s", &num1l, &num1h, &num2l, &num2h);
        fscanf(input_fil1, "%4s %4s", &num3l, &num3h);
        }
else if (mode == 2)
        {
        fscanf(input_fil1, "%4s %4s %4s %4s", &num1l, &num1h, &num2l, &num2h);
        fscanf(input_fil1, "%4s %4s %4s %4s", &num3l, &num3h, &num4l, &num4h);
        fscanf(input_fil1, "%4s %4s %4s %4s", &num5l, &num5h, &num6l, &num6h);
        }
else if ((mode == 3) || (mode == 4))
        {
        fscanf(input_fil1, "%4s %4s %4s %4s", &num1l, &num1h, &num2l, &num2h);
        fscanf(input_fil1, "%4s %4s %4s %4s", &num3l, &num3h, &num4l, &num4h);
        fscanf(input_fil1, "%4s %4s %4s %4s", &num5l, &num5h, &num6l, &num6h);
        fscanf(input_fil1, "%4s %4s", &num7l, &num7h);
        }
/* x2f function is called below to convert two words of 32 bit data */
/* to Floating point ASCII values                                   */
if (mode == 1)
        {
        parm1 = x2f( num1l, num1h );
        parm2 = x2f( num2l, num2h );
        parm3 = x2f( num3l, num3h );
        }
else if (mode == 2)
        {
        parm1 = x2f( num1l, num1h );
        parm2 = x2f( num2l, num2h );
        parm3 = x2f( num3l, num3h );
        parm4 = x2f( num4l, num4h );
        parm5 = x2f( num5l, num5h );
        parm6 = x2f( num6l, num6h );
        }
else if ((mode == 3) || (mode == 4))
        {
        parm1 = x2f( num1l, num1h );
        parm2 = x2f( num2l, num2h );
        parm3 = x2f( num3l, num3h );
        parm4 = x2f( num4l, num4h );
        parm5 = x2f( num5l, num5h );
        parm6 = x2f( num6l, num6h );
        parm7 = x2f( num7l, num7h );
        }
/* The same values are written in the output file IMPOR.T below  */
if (mode == 1)
        {
        fprintf(output_file, "%f\n", parm1);
        fprintf(output_file, "%f\n", parm2);
        fprintf(output_file, "%f\n", parm3);
        }
else if (mode == 2)
        {
```

```
            fprintf(output_file, "%f\n", parm1);
            fprintf(output_file, "%f\n", parm2);
            fprintf(output_file, "%f\n", parm3);
            fprintf(output_file, "%f\n", parm4);
            fprintf(output_file, "%f\n", parm5);
            fprintf(output_file, "%f\n", parm6);
            }
    else if ((mode == 3) || (mode == 4))
            {
            fprintf(output_file, "%f\n", parm1);
            fprintf(output_file, "%f\n", parm2);
            fprintf(output_file, "%f\n", parm3);
            fprintf(output_file, "%f\n", parm4);
            fprintf(output_file, "%f\n", parm5);
            fprintf(output_file, "%f\n", parm6);
            fprintf(output_file, "%f\n", parm7);
            }
    fcloseall();
}
float x2f(char *instr1, char *instr2)
{
    /* Input: Two words, LSW and MSW of the 32 bit long word in character
                string format.
      Output: Single precision floating point number they represent, in
                ASCII format
      C Functions used: strcat(), strtoul()          */
    /* Following union defines different formats for the same number */
    union
    {
            float f;          /* As float */
            unsigned long ul;  /* As unsigned long (for binary conversion) */
    } parm;
    unsigned long num = 0;
    char  bufstr[11] = {'0', 'x'}, *stopstr;
    /* The LSW and MSW strings are catenated, to generated 32 bit long word */
    strcat(instr2, instr1);
    /*  The string 0x is added in front of this word, to identify the Hex */
    /*  format, for the function strtoul()                    */
    strcat(bufstr, instr2);
   /* parm is generated in unsigned long format, using ASCII string to */
    /* unsigned long conversion function. The number 16 is the base for the */
    /* Hex format.                                          */
    parm.ul = strtoul(bufstr, &stopstr, 16);
    /* The same parameter is returned in floating point format */
    return (parm.f);
}

/*****************************************************************************/
/* log2ini.C -- This single routine does the following tasks:          */
/*                                                                   */
/*        1. Generates SIMNON initialization file every beginning     */
/*          of sampling interval (including at time = 0).          */
/*                                                                   */
/*        2. Updates the buffer files storing responses for the three  */
/*          controllers, Ids, Iqs and speed.                    */
/*                                                                   */
/*        3. Generates SIMNON simulation macro every sampling interval */
/*                                                                   */
/*        4. Keeps track of real time, and changes modes of operation  */
/*          as the simulation and optimization progress           */
/*                                                                   */
/*        5. Determines whether ES should intervene the simulation    */
/*          for tuning Kp, Ki's of various loop or should it continue */
/*          to generate response data by sequential simulation in    */
/*          SIMNON and C25 simulator.                          */
```

```
/*                                                              */
/*      Author: Sunil M. Chhaya                                 */
/*      Last Modified and perfected: 03/16/94                   */
/*                                                              */
/*****************************************************************************/

#include <stdio.h>     /* for FILE definition     */
#include <io.h>        /* for EOF definition      */
#include <string.h>    /* for strcmp() definition */
#include <conio.h>     /* for getche()            */
#include <errno.h>     /* for remove()            */
#include <math.h>      /* for fmod()              */
#include <stdlib.h>    /* for div()               */
#define MAXBUF 80
/* Following are the flags set according to mode of operation */
int Mode, Idst, Iqst, Olt, Tstrun;
int Brbf, App_eids, App_eiqs, App_ewr, App_ilt, App_olt;
/* Following are the actual parameters passed from simulator to SIMNON */
float Vdse_ref, Vqse_ref, Thetae, Eids, Eiqs, Ewr, Wref, Idse_ref, Iqse_ref, We;
float Tsamp=0.0002;
/* Following are the machine parameters read from ES  */
float  Wb, Rs1, Rr, Lr, Ls, Lm, J, B, Kll, Vd;
int Pole;
div_t Divres;   /* For its use with the structure in the function div() */
/* Following files are processed by this routine       */
static char Fname_in1[] = "SIMNON.LOG", Fname_in2[] = "IMPOR.T";
static char Fname_in3[] = "TIMEINFO.DAT", Fname_in4[] = "MODE.DAT";
static char Fname_in5[] = "INFSIM.BAT", Fname_in6[] = "PARAM.DAT";
static char Fname_ou1[] = "INISTAT.T",Fname_ou2[] = "SIMUL.T";
static char Fname_ou3[] = "EIDS.T", Fname_ou4[] = "EIQS.T";
static char Fname_ou5[] = "EWR.T";
FILE *Input_fil1, *Input_fil2, *Input_fil3, *Input_fil4;
FILE *Input_fil5, *Input_fil6, *Output_fil1, *Output_fil2;
FILE *Output_fil3, *Output_fil4, *Output_fil5;
/* Calls various subroutines to implement sequential simulation control */
main()
{
  void Fileopen(void);        /* Opens all the i/o files   */
  void Set_time_mode(void);   /* Sets simulation time and mode  */
  void Init_sim_states(void); /* Generates INISTAT.T          */
  void Read_c25_ops(void);    /* Reads C25 relevant data       */
  void Break_bf(void);        /* Breaks from infinite loop of INFSIM.BAT */
  void Init_time_0(void);     /* Generates INISTAT.T at t = 0 */
  void Update_buffer(void);   /* Appends relevant buffers    */
  Fileopen();       /* Opens all i/o files */
  Set_time_mode();  /* Sets the mode (of tuning) and updates time */
  Read_c25_ops();   /* Reads data from 'C25 simulator o/p */
  switch(Mode)
          {
          case 1:
            {
            Init_time_0(); /* Initialize params and init. cond.s */
            break;
            }
          case 0:
            {
            Init_sim_states(); /* Generate INISTAT.T from SIMNON.LOG */
            Update_buffer();  /* Update buffers for waveform acquisition */
            Break_bf();       /* Break out of infinite loop when necessary */
            break;           /* for ES to intervene for Kp/Ki opt.   */
            }
          }
  fcloseall ();
}
void Fileopen(void)
```

```
{
  /* All I/O files are opened by this function  */
  /* Old version of INISTAT.T  removed below */
  remove(Fname_ou1);
  if ((Input_fil3 = fopen(Fname_in3, "r+")) == NULL)
          {
          printf("Error opening TIMEINFO.DAT.\n");
          printf("Press any key to continue.\n");
          getche();
          exit(0);
          }
  if ((Input_fil4 = fopen(Fname_in4, "r")) == NULL)
          {
          printf("Error opening MODE.DAT.\n");
          printf("Press any key to continue.\n");
          getche();
          exit(0);
          }
  if ((Input_fil5 = fopen(Fname_in5, "r")) == NULL)
          {
          printf("Error opening INFSIM.BAT.\n");
          printf("Press any key to continue.\n");
          getche();
          exit(0);
          }
  if ((Input_fil6 = fopen(Fname_in6, "r")) == NULL)
          {
          printf("Error opening PARAM.DAT.\n");
          printf("Press any key to continue.\n");
          getche();
          exit(0);
          }
  if ((Output_fil1 = fopen(Fname_ou1, "a+")) == NULL)
          {
          printf("\"%s\": can't open.\n", Fname_ou1);
          exit(1);
          }
  if ((Output_fil2 = fopen(Fname_ou2,"w")) == NULL)
          {
          printf("\"%s\": can't open.\n", Fname_ou2);
          exit(1);
          }
  if ((Output_fil3  = fopen(Fname_ou3, "a+")) == NULL)
          {
          printf("\"%s\": can't open.\n", Fname_ou3);
          exit(1);
          }
  if ((Output_fil4  = fopen(Fname_ou4 , "a+")) == NULL)
          {
          printf("\"%s\": can't open.\n", Fname_ou4);
          exit(1);
          }
  if ((Output_fil5  = fopen(Fname_ou5 , "a+")) == NULL)
          {
          printf("\"%s\": can't open.\n", Fname_ou5 );
          exit(1);
          }
}
void Set_time_mode(void)
{
  /* This function implements the basic mode generation logic,      */
  /* by doing the following:                                        */
  /* 1. By reading the time from file TIMEINFO.DAT. and mode of opt. */
  /*    from file CONTROL.DAT, to generate flags for,               */
  /*         (i) Appending the data storage buffer files, or       */
```

```
/*      (ii) Modifying INFSIM.BAT to break out of loop,        */
/* 2. By generating the new macro SIMUL.T, updating simulation    */
/*   in SIMNON                                                    */
/* 3. By updating the time counter by one Ts, for next operation. */
float rtime=0, st_iqt=0, st_olt=0, st_trn=0;
double c1=0, c2=0, c3=0, c4=0, c5=0, c6=0, c7=0, c12=0, c52=0, d=0,ee=0, ec=0, ef=0, e=0;        double ec12=0;
int interval, ig, mode_of_sim, imul1, imul2, imul3;
/* read real time info from the time counter       */
fscanf(Input_fil4, "%d", &mode_of_sim);
/* Following block of code sets mode of */
/* tuning/breaking loop/appending buf */
Idst = (mode_of_sim == 1);
Iqst = (mode_of_sim == 2);
Olt =  (mode_of_sim == 3);
Tstrun = (mode_of_sim == 4);
/* Mode set for Ids tuning here */
if(Idst == 1)
        {
        fscanf(Input_fil3, "%f", &rtime);
        if (rtime == 0)
          {
          Mode = 1;
          }
        else
          {
          Mode = 0;
          }
        c1 = (double)(rtime+Tsamp);
        c2 = 80* (double)(Tsamp);
        /* c2 = 2150 * (double) (Tsamp); */
        /* c12 = fmod(c1, c2); */
        imul3 = (int)(c1 / c2);
        c12 = c1 - (double) (imul3 * c2);
        ec12 = c12 - c2;
        if (fabs(ec12) < 1e-5)
          {
          imul3 += 1;
          c12 = c1 - (double)(imul3*c2);
          }
        App_eids = 1;
        App_eiqs = 0;
        App_ewr = 0;
        /* Set flags for breaking batch file loop or appending buffer for eid */
        if (fabs(c12) < 0.75 * (double)Tsamp)
          {
          Brbf = 1;
        /*   App_eiqs = 0; */
          }
        else if ((c12 > (0.75 * (double)Tsamp)) && (c12 < (double)Tsamp))
          {
          Brbf = 0;
          }
        else
          {
          Brbf = 0;
         /*  App_eiqs = 1; */
          }
        }
/* Mode set for Iqs tuning here */
else if (Iqst == 1)
        {
        fscanf(Input_fil3, "%f %f", &rtime, &st_iqt);
        c1 = (double)(rtime + Tsamp);
        c2 = (double)(80*Tsamp);
        c5 = (double)(c1 - st_iqt);
```

```
        imul1 = (int) (c5 / c2);
        c52 = c5 - (double) (imul1 * c2);
        ec = c52 - c2;
        if (fabs(ec) < 1e-5)
          {
          imul1 += 1;
          c52 = c5 - (double)(imul1*c2);
          }
        App_eids = 0;
        App_eiqs = 1;
        App_ewr = 0;
        if (fabs(c52) < 0.75 * (double)Tsamp)
          {
          Brbf = 1;
          }
        else if ((c52 > (0.75 * (double)Tsamp)) && (c52 < (double)Tsamp))
          {
          Brbf = 0;
          }
        else
          {
          Brbf = 0;
          }
        }
/* Mode set for outer loop tuning here */
else if (Olt == 1)
        {
        fscanf(Input_fil3, "%f %f", &rtime, &st_olt);
        c1 = rtime + Tsamp ;
        c2 = 80*Tsamp;
        /* c2 = 2150 * Tsamp; */
        c3 = c1 - st_olt;
        c4 = rtime - st_olt;
        c6 = 1000*Tsamp;
        c7 = 5*Tsamp;
        e = fmod(c4, c7);
        ee = e - c7;
        imul2 = (int) (c3 / c6);
        d = c3 - (double) (imul2 * c6);
        ef = d - c6;
        if (fabs(ef) < 1e-5)
          {
          imul2 += 1;
          d = c3 - (double)(imul2*c6);
          }
        if(fabs(ee) < 1e-5)
          {
          e = 0;
          }
        if (fabs(d) < 0.75* (double)Tsamp)
          {
          Brbf = 1;
          App_ewr = 0;
          }
        else if ((d > 0.75* (double)Tsamp) && (d < (double)Tsamp))
          {
          Brbf = 0;
          App_ewr = 1;
          }
        else if (fabs(e) < 0.75*(double)Tsamp)
          {
          App_ewr = 1;
          Brbf = 0;
          }
        else if((fabs(e) > 0.75* (double)Tsamp) && (fabs(e) < (double)Tsamp))
```

```
            {
            App_ewr = -1;
            Brbf = 0;
            }
         else
            {
            App_ewr = -1;
            Brbf = 0;
            }
         }
/* Mode set for test run here */
else if (Tstrun == 1)
         {
         fscanf(Input_fil3, "%f %d", &rtime, &interval);
         Divres = div(interval, 5);
         ig = Divres.rem;
         if (ig == 0)
            {
            App_olt = 1;
            }
         else
            {
            App_olt = -1;
            }
         if (interval < 80)
            {
            App_ilt = 1;
            }
         else
            {
            App_ilt = 0;
            }
         if (interval >= 999)
            {
            Brbf = 1;
            }
         interval += 1;
         }
/* delete the old real time info file */
remove(Fname_in3);
/* Generate new simulation command macro for SIMNON        */
fprintf(Output_fil2, "MACRO simul\n");
fprintf(Output_fil2, "SIMU %f %f 1e-6 /plndat\n", rtime, rtime+Tsamp);
fprintf(Output_fil2, "export expor<plndat ib[machindq] ic[machindq] wr[machindq] %f -notime\n", rtime+Tsamp);
fprintf(Output_fil2, "END\n");
/* The time is being updated below    */
rtime += Tsamp;
/* Open a new real time info file and store new simulation start time */
Input_fil3 = fopen(Fname_in3, "w");
printf("The current real time is: %f\n", rtime);
if (Idst == 1)
         {
         fprintf(Input_fil3, "%f", rtime);
         }
else if (Iqst == 1)
         {
         fprintf(Input_fil3, "%f %f", rtime, st_iqt);
         }
else if (Olt == 1)
         {
         fprintf(Input_fil3, "%f %f", rtime, st_olt);
         }
else if (Tstrun == 1)
         {
         fprintf(Input_fil3, "%f %d", rtime, interval);
```

```
        }
}
void Init_sim_states(void)
{
  /* This function scans the SIMNON.LOG file generated by SIMNON    */
  /* using the command "disp(lp) -state". It also scans the file    */
  /* IMPOR.T for the states generated by C25 Simulator, and combines */
  /* both of them, to generate a new SIMNON "GET" file, called      */
  /* INISTAT.T. The formats of SIMNON.LOG, IMPOR.T and INISTAT.T are */
  /* different. This routine is executed at every sampling instant, */
  /* except at the very beginning of the simulation (t = 0), where  */
  /* another function is used.                                      */
  /* tmpline is individual line read from SIMNON.LOG  */
  static char tmpline[MAXBUF];
  /* outline is the line written in INISTAT.T       */
  char  outline[22], tststr[6], buf_outline[22], *lb_outline;
  char *fbl_outline, tstr[6] ;
  /* a, b, c, d, aa, etc are flags checking various types of lines  */
  /* from SIMNON.LOG, based on their contents                       */
  int i = 1, a = 0 , b = 0, b1 = 0, b2 = 0, b3 = 0, c = 0, d = 0, num;
  int outcdloop = 0, aa = 0, e = 0;
  if ((Input_fil1 = fopen(Fname_in1, "r")) == NULL)
        {
        printf("Error opening SIMNON.LOG.\n");
        printf("Press any key to continue.\n");
        getche();
        exit(0);
        }
  while (strstr(tmpline, "CONNECTING SYSTEM") == NULL)
  /* execute the loop till SIMNON connecting system is reached */
        {
        /* Determine type of line.          */
        while ((a != 1) && (b != 1) && (c != 1) && (d != 1) && (e != 1))
          {
          fgets(tmpline, MAXBUF, Input_fil1);
          num = strlen(tmpline);
          strncpy(tststr, tmpline, 5);
          tststr[5] = '\0';
          a = (num <= 2 || (strcmp(tststr, "- - -") == 0));
          b1 = (strstr(tmpline, "CONTINUOUS SYSTEM") != NULL);
          b2 = (strstr(tmpline, "DISCRETE SYSTEM") != NULL);
          b3 = (strstr(tmpline, "CONNECTING SYSTEM") != NULL);
          b = (b1 || b2 || b3);
          c = (strchr(tmpline, ':') != NULL);
          if (strstr(tmpline, "SYNSTA") != NULL)
                e = 1;
                b = 0;
                }
          d = ((a == 0) && (b == 0) && (c == 0) && (e == 0));
          }
          /* Do nothing if the line is empty or contains '-------..' */
          if (a == 1)
                {
                a = 0;
                }
          /* Extract the label of the subsystem here          */
          if (b == 1)
                {
                strcpy(buf_outline, tmpline + 39);
                buf_outline[0] = '[';
                lb_outline = strchr(buf_outline, '\n');
                *lb_outline = ']';
                *(lb_outline + 1) = '\0';
                strcpy(outline, buf_outline);
                fprintf(Output_fil1, "%s\n", outline);
```

```
                b = 0;
                strset(outline, '\0');
                }
        /* Extract the states and their values here          */
        if ((c == 1) || (d == 1))
                {
                while ((outcdloop == 0) && (i <= 3))
                  {
                  strncpy (buf_outline, tmpline + 8 + 23*(i-1), 20);
                  strncpy(tstr, buf_outline, 5);
                  tstr[5] = '\0';
                  aa = (strcmp(tstr, "- - -") == 0);
                  if (aa == 1)
                          {
                          break;
                          }
                  if (strchr(buf_outline, '\n') != NULL)
                          {
                          outcdloop = 1;
                          }
                  fbl_outline = strchr(buf_outline, ' ');
                  *fbl_outline = ':';
                  buf_outline[20] = '\0';
                  strcpy(outline, buf_outline);
                  fprintf(Output_fil1, "%s\n", outline);
                  i+=1;
                  strset(outline, '\0');
                  }
                  c = 0;
                  d = 0;
                  outcdloop = 0;
                  i = 1;
                }
        if(e == 1)
                {
                fprintf(Output_fil1, "[SYNSTA]\n");
                fprintf(Output_fil1, "vdser:%7.2f\n", Vdse_ref);
                fprintf(Output_fil1, "vqser:%7.2f\n", Vqse_ref);
                fprintf(Output_fil1, "thetae:%7.2f\n", Thetae);
                fprintf(Output_fil1, "[MACHINDQ]\n");
                fprintf(Output_fil1, "weref:%7.2f\n", We);
                e = 0;
                }
        }
  }

void Read_c25_ops(void)
{
  /* This function makes available to the system, the values of the    */
  /* outputs of simulator, at every sampling instant, during different */
  /* modes of optimization.                                            */
  /* sets the values of all the inputs from C25 Simulator in all conditions */
  if (Mode == 1)
        {
        fscanf(Input_fil6, "%f %f %f %f", &Idse_ref, &Iqse_ref, &Wref, &Vd);
        fscanf(Input_fil6, "%f %f %f %f %f", &Wb, &Rs1, &Rr, &Lr, &Ls);
        fscanf(Input_fil6, "%f %f %d", &Lm, &J, &Pole);
        Vdse_ref = 0;
        Vqse_ref = 0;
        Thetae = 0;
        Eids = Idse_ref;
        Eiqs = Iqse_ref;
        Ewr = Wref;
        We = 0;
        }
```

```
    if (Mode != 1 && (((Input_fil2 = fopen(Fname_in2, "r")) == NULL)))
            {
            printf("Error opening IMPOR.T.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
    else if ((App_eids == 1) && (Mode != 1))
            {
            fscanf(Input_fil2, "%f\n", &Vdse_ref);
            fscanf(Input_fil2, "%f\n", &Eids);
            fscanf(Input_fil2, "%f\n", &We);
            Vqse_ref = 0;
            Eiqs = 0;
            Thetae = 0;
            }
    else if (App_eiqs == 1)
            {
            fscanf(Input_fil2, "%f\n", &Vdse_ref);
            fscanf(Input_fil2, "%f\n", &Vqse_ref);
            fscanf(Input_fil2, "%f\n", &Eids);
            fscanf(Input_fil2, "%f\n", &Eiqs);
            fscanf(Input_fil2, "%f\n", &Thetae);
            fscanf(Input_fil2, "%f\n", &We);
            }
    else if ((App_ewr == 1) || (App_olt == 1))
            {
            fscanf(Input_fil2, "%f\n", &Vdse_ref);
            fscanf(Input_fil2, "%f\n", &Vqse_ref);
            fscanf(Input_fil2, "%f\n", &Iqse_ref); */
            fscanf(Input_fil2, "%f\n", &Eids);
            fscanf(Input_fil2, "%f\n", &Eiqs);
            fscanf(Input_fil2, "%f", &Ewr);
            fscanf(Input_fil2, "%f\n", &Thetae);
            fscanf(Input_fil2, "%f\n", &We);
            }
    else if ((App_ewr == -1) || (App_olt == -1))
            {
            fscanf(Input_fil2, "%f\n", &Vdse_ref);
            fscanf(Input_fil2, "%f\n", &Vqse_ref);
            fscanf(Input_fil2, "%f\n", &Eids);
            fscanf(Input_fil2, "%f\n", &Eiqs);
            fscanf(Input_fil2, "%f\n", &Ewr);
            fscanf(Input_fil2, "%f\n", &Thetae);
            fscanf(Input_fil2, "%f\n", &We);
            }
}
void Break_bf(void)
{
  /* This function modifies the simulation batch file, INFSIM.BAT */
  /* to transfer control back from DOS to Expert System, whenever */
  /* ES needs to intervene, every time one full response is acquired */
  if(Brbf == 1)
            {
            remove(Fname_in5);
            Input_fil5 = fopen(Fname_in5, "w");
            fprintf(Input_fil5, ":iltsim\n");
            fprintf(Input_fil5, "echo off\n");
            fprintf(Input_fil5, "echo Invoking SIMNON simulation of plant..\n");
            fprintf(Input_fil5, "SIMNON\n");
            fprintf(Input_fil5, "echo Invoking TMS320C25 simulator for controller simulation..\n");
            fprintf(Input_fil5, "SIM2X\n");
            fprintf(Input_fil5, "echo Converting controller hex outputs to ASCII decimal...\n");
            fprintf(Input_fil5, "HEX2FLO\n");
            fprintf(Input_fil5, "echo Generating plant state initialization file...\n");
```

```
            fprintf(Input_fil5, "LOG2INI\n");
            fprintf(Input_fil5, "rem goto iltsim\n");
            fclose(Input_fil5);
            }
}
void Init_time_0(void)
{
   /* This function generates initial conditions and states for the */
   /* SIMNON simulation, at time = 0. It also writes the initial values */
   /* of buffer files, for their display later.                    */
   printf("Initializing SIMNON states for t = 0\n");
   fprintf(Output_fil1, "[SYNSTA]\n");
   fprintf(Output_fil1, "vdser:0\n");
   fprintf(Output_fil1, "vqser:0\n");
   fprintf(Output_fil1, "thetae:0\n\n");
   fprintf(Output_fil1, "[PHASE23]\n");
   fprintf(Output_fil1, "[SPWM]\n");
   fprintf(Output_fil1, "[MACHINDQ]\n");
   fprintf(Output_fil1, "iqs:0\n");
   fprintf(Output_fil1, "ids:0\n");
   fprintf(Output_fil1, "iqr:0\n");
   fprintf(Output_fil1, "idr:0\n");
   fprintf(Output_fil1, "w:0\n");
   fprintf(Output_fil1, "weref:0\n\n");
   fprintf(Output_fil1, "[CONPLA]\n");
   fprintf(Output_fil3, "%f\n", Eids);
   fprintf(Output_fil4, "%f\n", Eiqs);
   fprintf(Output_fil5, "%f\n", Ewr);
}
void Update_buffer(void)
{
   int i, j, k, l, m;
   /* Based on the flags set by routine Set_time_mode(), this function */
   /* appends the buffers with the values read from simulator outputs  */
   if (App_eids == 1)
            {
            for (i = 0; i < 5; i++)
               {
               fprintf(Output_fil3, "%f\n", Eids);
               }
            }
   else if (App_eiqs == 1)
            {
            for (j = 0; j < 5; j++)
               {
               fprintf(Output_fil4, "%f\n", Eiqs);
               }
            }
   else if ((App_ewr == 1) || (App_olt == 1))
            {
            for (k = 0; k < 5; k++)
               {
               fprintf(Output_fil5, "%f\n", Ewr);
               }
            }
   else if (App_olt == 1)
            {
            for (l = 0; l < 5; l++)
               {
               fprintf(Output_fil5, "%f\n", Ewr);
               }
            }
   if (App_ilt == 1)
            {
            for (m = 0; m < 5; m++)
```

```
            {
            fprintf(Output_fil3, "%f\n", Eids);
            fprintf(Output_fil4, "%f\n", Eiqs);
            }
          }}

/*****************************************************************/
/* TRSCAN.C -- Routine calculating the Transient Response Specs of a response   */
/*****************************************************************/

#include <stdio.h>     /* For FILE definition */
#include <io.h>        /* for EOF definition */
#include <math.h>      /* for scientific functions */
#include <stdlib.h>
#include <string.h>
main(int argc, char *argv[])
{
  int Sgn(float x1, float x2);          /* Determines sign of a float no. */
  FILE *input_file,  *output_fil1, *output_fil2;
  char ch;
  float e, t, tmax, tmin, emin = 5000, emax = -5000;
  float e0, t0, tr = 0, mp = 0, ts = 0, tau = 0, sign, ep, tp;
  int nat_of_resp = 0, tr_found = 0, signe0, signe;
  char nature[12] = "OVERDAMPED", under[12]="UNDERDAMPED";
  printf("SCANNING THE RESPONSE DATA TO PRODUCE RESPONSE SPECS..\n");
  if(argc != 4)
          {
          printf("Usage: trscan infile outfile1 outfile2 \n");
          exit(1);
          }
  if ((input_file = fopen(argv[1], "r")) == NULL)
          {
          printf("Error opening response data file.\n");
          printf("Press any key to continue.\n");
          getche();
          exit(0);
          }
  if ((output_fil1 = fopen(argv[2], "w")) == NULL)
          {
          printf("Error opening TRS nature file.\n");
          printf("Press any key to continue.\n");
          getche();
          exit(0);
          }
  if ((output_fil2 = fopen(argv[3], "w")) == NULL)
          {
          printf("Error opening TRS data file.\n");
          printf("Press any key to continue.\n");
          getche();
          exit(0);
          }
  /* Following section determines the nature of response (UD/OD) */
  fscanf(input_file, "%f %f", &t0, &e0);
  signe0 = Sgn(e0, 1);
  while (fscanf(input_file, "%f %f", &t, &e) != EOF)
          {
          signe = Sgn(e, e0);
          if (signe != signe0)
            {
            nat_of_resp = 1;
            strcpy(nature, under);
            break;
            }
          }
  fclose (input_file);
```

```
input_file = fopen(argv[1], "r");
/* Following section determines the t response specs for either case */
switch(nat_of_resp)
        {
        case 0:
          {
          while (fscanf(input_file, "%f %f", &t, &e) != EOF)
                  {
                  if (fabs(e) <= (0.362*fabs(e0)))
                    {
                    tau = (t-t0);
                    ts = 4*tau;
                    break;
                    }
                  }
          fprintf(output_fil1, "%s", nature);
          fprintf(output_fil2, "%f\n%f\n", tau, ts);
          break;
          }
        case 1:
          {
          while (fscanf(input_file, "%f %f", &t, &e) != EOF)
                  {
                  /* Following section finds rise time   */
                  if (tr_found != 1)
                    {
                    if (fabs(e) <= (0.1*fabs(e0)))
                            {
                            tr = t;
                            tr_found = 1;
                            }
                     }
                  /* Following section finds peak Mp and tp   */
                  if (signe0 == 0)
                    {
                    if (e < emin)
                            {
                            emin = e;
                            tmin = t;
                            }
                    ep = emin;
                    tp = tmin;
                    }
                  else
                    {
                    if (e > emax)
                            {
                            emax = e;
                            tmax = t;
                            }
                    ep = emax;
                    tp = tmax;
                    }
                  /* Following section finds settling time    */
                  if (ts == 0)
                    {
                    if (fabs(e) <= (0.05*fabs(e0)))
                            {
                            ts = t;
                            }
                    }
                  else if (fabs(e) > (0.05*fabs(e0)))
                    {
                    ts = 0;
                    }
```

```
                    }
                    /* Following block calculates actual TRS's  */
                    mp = fabs(ep/e0) * 100;
                    ts -= t0;
                    tr -= t0;
                    tp -= t0;
                    fprintf(output_fil1, "%s", nature);
                    fprintf(output_fil2, "%f\n%f\n%f\n%f\n", mp, ts, tr, tp);
                    break;
                }
            }
    fcloseall();
}
int Sgn(float x1, float x2)
{
    /* Returns 0 if positive no, or 1, if x is negative   */
    int sgn;
    if (x2 > 0)
            {
            if (x1 >= 0)
                {
                sgn = 0;
                }
            else
                {
                sgn = 1;
                }
            }
    else if (x2 < 0)
            {
            if (x1 > 0)
                {
                sgn = 0;
                }
            else
                {
                sgn = 1;
                }
            }
    return(sgn);
}

/*****************************************************/
/* BFGEN.C -- Generates simulation batch files for first sampling */
/*          instant as well as any other sampling instant.     */
/*****************************************************/

#include <stdio.h>
main()
{
    static char fname_ou1[] = "INFSIM.BAT";
    FILE *output_fil1;
    if ((output_fil1 = fopen(fname_ou1, "w")) == NULL)
            {
            printf("Error opening INFSIM.BAT.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
    printf("GENERATING THE SIMULATION BATCH FILE ...\n");
    fprintf(output_fil1, ":iltsim\n");
    fprintf(output_fil1, "echo off\n");
    fprintf(output_fil1, "echo Invoking plant simulation in SIMNON\n");
    fprintf(output_fil1, "simnon\n");
    fprintf(output_fil1, "echo Invoking controller simulation in TMS320C25 simulator\n");
```

```
   fprintf(output_fil1, "sim2x\n");
   fprintf(output_fil1, "echo Converting Hex controller outputs to ASCII decimal\n");
   fprintf(output_fil1, "hex2flo\n");
   fprintf(output_fil1, "echo Generating plant state initializing file\n");
   fprintf(output_fil1, "log2ini\n");
   fprintf(output_fil1, "goto iltsim\n");
   /* fprintf(output_fil1, ":end\n"); */
   fcloseall();
}

/*************************************************/
/* CRECST.C -- Creates initial intermediate buffer data file for */
/*          simulator                            */
/*************************************************/
#include <stdio.h>

main()
{
   int i, mode;
   char nul_str[] = "0000";
   static char  fname_in[] = "MODE.DAT", fname_out[] = "CSTATE.DAT";
   FILE  *input_file, *output_file;
void t_file = fopen(fname_in, "r");
   fscanf(input_file, "%d", &mode);
   printf("Generating buffer data file for TMS320C25 Simulator\n");
   if (mode == 1)
            {
            output_file = fopen(fname_out, "w");
            for (i=0; i<15; i++)
               {
               fprintf(output_file, "%s\n", nul_str);
               }
            }
   else if (mode == 3)
            {
            output_file = fopen(fname_out, "a+");
            for (i=0; i<9; i++)
               {
               fprintf(output_file, "%s\n", nul_str);
               }
            }
   fclose(output_file);
}

/*********************************************************/
/* REFRESH.C -- Refreshes buffer memory for acquired response data     */
/*********************************************************/
#include <stdio.h>     /* for FILE definition */
#include <io.h>        /* for EOF definition */
#include <errno.h>     /* for remove()        */
main()
{
   int mode;
   static char fname_in[] = "MODE.DAT";
   static char fname_ou1[] = "EIDS.T";
   static char fname_ou2[] = "EIQS.T", fname_ou3[] = "EWR.T";
   FILE *input_file, *output_fil1, *output_fil2, *output_fil3;
   if ((input_file = fopen(fname_in, "r")) == NULL)
            {
            printf("Error opening MODE.DAT.\n");
            printf("Press any key to continue.\n");
            getche();
            exit(0);
            }
   printf("INITIALIZING BUFFERS FOR RESPONSE DATA AGAIN ... \n");
```

```
    fscanf(input_file, "%d\n", &mode);
    if (mode == 1)
            {
            remove (fname_ou1);
            }
    else if (mode == 2)
            {
            remove(fname_ou2);
            }
    else if (mode == 3)
            {
            remove(fname_ou3);
            }
    else if (mode == 4)
            {
            remove(fname_ou1);
            remove(fname_ou2);
            remove(fname_ou3);
            }
    fcloseall();
}

/*************************************************************/
/* BREAK_BF.C -- Breaks the batch file loop for INFSIM.BAT when executed */
/*************************************************************/
#include <stdio.h>
#include <errno.h>
#include <io.h>
main()
{
   static char fname_in5[] = "INFSIM.BAT";
   FILE *input_fil5;
   remove(fname_in5);
   input_fil5 = fopen(fname_in5, "w");
   fprintf(input_fil5, ":sysim\n");
   fprintf(input_fil5, "SIMNON\n");
   fprintf(input_fil5, "SIM2X\n");
   fprintf(input_fil5, "LOG2INI\n");
   fprintf(input_fil5, "rem goto sysim\n"); */
   fprintf(input_fil5, ":end");
   fclose(input_fil5);
}
```

APPENDIX E Real - Time TMS320C25 Based IVC-IM Control Software Listing

```
/*******************************************/
/* CTIMER.C - Slow Controller and Host Routine */
/*******************************************/

extern int Tim_isr(), Initial(), Speed();
extern int Ctim, Vs_i, Thetae, Vs2, Delm, Delt, Wr1;
extern unsigned int Theta_i_h;
extern float Wrr_fl, Kps_fl, Kis_fl, Wr_fl;
extern float Ewr_fl, Iqr_fl_n, Xs_fl, LIM, Iqs_rated;
extern int Kps, Kis, Wrr;
main()
{
  void Spdpilim(void);                    /* Speed PI Controller */
  Initial();
  asm("   EINT");
  asm("L1:          ");
  while (1>0)
  asm("     CALL Hostdsp");  /* Background, Host to DSP communication task */
        {;}
  .copy "Hostdsp.asm"
}
void Spdpilim(void)
{
  LIM = 0.3;
  /* PI speed controller parameters   */
  Wrr_fl = Wrr*2.3981;        /* 2.3981=239.81/100, 239.81=2290/9.55 */
  Kps_fl = Kps*0.2;                   /* Kpmax = 20 = Kimax */
  Kis_fl = Kis*0.2;
  Wr_fl=(Wr1/136.64);                 /* in r/s, 136.64=(32767/(2290/9.55)) */
  /* PI controller equations */
  Ewr_fl = Wrr_fl - Wr_fl;
  Iqr_fl_n = Kps_fl * Ewr_fl + Kis_fl * 0.001 * Xs_fl;
  Xs_fl += Ewr_fl;
  /* Implement limiter below  */
  if (Iqr_fl_n > LIM*Iqs_rated)
        {
        Iqr_fl_n = LIM*Iqs_rated;
        }
  else if (Iqr_fl_n < -1*LIM*Iqs_rated)
        {
        Iqr_fl_n = -1*LIM*Iqs_rated;
        }
}

;*********************************************************************
* Timer Interrupt Service Routine, Implements basic vector control and current control
*********************************************************************
        .title      "Tim_isr.asm"
        .mmregs
        .global     _Tim_isr, _Initial, _Dac_A_isr, _Dac_B_isr, _Adc_isr
        .global _Sctable, SINTBL, _Speed, _Sqwavgen
        .global     _Phase23, _Synsta, _Hostdsp, _Bufstr, _Spdpilim
        .global     _Ctim, _counter, _dacabuf, _dacbbuf
        .global     _Vdss_i, _Vqss_i, _Vb0, _Vc0, _SQRT32, _HALF, _ZERO4
        .global     _OutbfA, _OutbfB, _Command, _Command2, _Disp, _Hostbuf, _DUMMY
        .global     _Bufal, _Bufah, _Col, _Low, _Bfadata, _Bfbdata, _Temp2
        .global     _Sacch, _Saccl, _ZERO, _ONE, _TEMP0, _Dummy, _BST_AUX
        .global     _Incomdata, _Mstep, _VAR1, _VAR2, _SWT0
        .global     _Stp, _Ssamp, _Mnew, _Mold, _Tnew, _Told, _Wr1
        .global     _Wrold, _Delm, _Delt, _Wr
        .global     _Swt5, _Ia, _Ic, _Start, _Idss, _Iqss
        .global     _Phase32, _Sqr1o3
```

```
        .global  _Thetagen, _Iqr, _Ksl, _We, _Sinth, _Costh
        .global _DT, _Thetah, _Thetal, _Theta, _Spdscl
        .global  _Invecrot, _Idse, _Iqse
        .global  _Idscont, _VDH, _VDL, _Vdh, _Vdl, _Vd, _Fd, _Scalf, _Vdr
        .global  _Eids, _Kpd, _Kid, _YDH, _YDL, _Dts, _Idr
        .global  _Iqscont, _VQH, _VQL, _Vqh, _Vql, _Vq, _Fq
        .global  _Vqr, _Eiqs, _Kpq, _Kiq, _YQH, _YQL
        .global  _Wref, _Ewr, _Fs, _Kps, _Kis, _YSH, _YSL
        .global _IQH, _IQL, _Iqh, _Iql, _Iqref, _Dtspd, _Mode
        .global _Idrated, _Nm2wre
        .global  _Swidest, _Swids, _Flids, _Swiqs, _Fliqs, _Swspd, _Flspd
        .global  _Idsq, _Iqsq, _Wrsq, _Flidest
; Volts/Hz variables
        .bss     _SQRT32, 1
        .bss     _HALF, 1
        .bss     _ZERO4, 1
        .bss     _Ctim, 1   ; Dummy timer interrupt counter
        .bss     _counter, 1        ; Counter used for generating DAC signals
        .bss     _dacabuf, 1        ; DAC A buffer
        .bss     _dacbbuf, 1        ; DAC B buffer
        .bss     _Dummy, 1          ; Dummy value in ADC ch B
        .bss     _Vdse_i, 1         ; Vdse, fx pt
        .bss     _Vqse_i, 1         ; Vqse, fx pt
        .bss     _TEMP0, 1
        .bss     _Vdss_i, 1         ; Vdss, fx pt
        .bss     _Vqss_i, 1         ; Vqss, fx pt
        .bss     _Vb0, 1            ; Ref. Vb
        .bss     _Vc0, 1            ; Ref. Vc
; Definition of data RAM
        .bss     _OutbfA, 1         ; Ch. A output buffer
        .bss     _OutbfB, 1         ; Ch. B output buffer
        .bss     _Command, 1        ; Host command word
        .bss     _Command2, 1       ; Value passed to 'C25
        .bss     _Disp, 1   ; Value passed to host
        .bss     _Hostbuf, 1        ; Buffer Host status
        .bss     _DUMMY, 1          ; Dummy reg
; Data buffer storage variables
        .bss     _Bufal, 1  ; Buffer addr. ptr, LSW
        .bss     _Bufah, 1  ; Buffer addr ptr, MSW
        .bss     _Col, 1            ; Column addr of DSP buffer
        .bss     _Low, 1            ; Row addr of DSP buffer
        .bss     _Bfadata, 1        ; Ch. A Buffer
        .bss     _Bfbdata, 1        ; Ch. B Buffer
        .bss     _Temp2, 1          ; Temporary variable
        .bss     _Sacch, 1  ; Push - Pop location of ACC MSW
        .bss     _Saccl, 1  ; Push / Pop location of ACC LSW
        .bss     _ZERO, 1 ; '0'
        .bss     _ONE, 1            ; '1'
        .bss     _BST_AUX, 1        ; Auxiliary variable
; Host-DSP communication variables
        .bss     _Incomdata, 1      ; Data read from Host
        .bss     _Mstep, 1  ; Main routine step
        .bss     _VAR1, 1 ; Buffer storage var, ch A
        .bss     _VAR2, 1 ; Buffer storage var, ch B
        .bss     _SWT0, 1 ; Software timer 0, for buffer storage
; Feedback speed computation variables
        .bss     _Ssamp, 1
        .bss     _Mnew, 1
        .bss     _Mold, 1
        .bss     _Tnew, 1
        .bss     _Told, 1
        .bss     _Wrold, 1
        .bss     _Wr1, 1
        .bss     _Wr, 1
        .bss     _Delm, 1
```

```
        .bss    _Delt, 1
;       Variables for IVC routines
        .bss    _Swt5, 1
; Acquired currents
        .bss    _Ia, 1
        .bss    _Ic, 1
        .bss    _Start, 1    ; Start / Stop flag
        .bss    _Idss, 1     ; Idss
        .bss    _Iqss, 1     ; Iqss
        .bss    _Sqr1o3, 1           ; Sqrt (1/3)
; Theta generation
        .bss    _DT, 1
        .bss    _Sinth, 1
        .bss    _Costh, 1
        .bss    _Thetah, 1
        .bss    _Thetal, 1
        .bss    _Ksl, 1
        .bss    _Iqr, 1
        .bss    _We, 1
        .bss    _Theta, 1
        .bss    _Spdscl, 1
; inverse vector rotation
        .bss    _Idse, 1
        .bss    _Iqse, 1
; IDS controller
        .bss    _Idr, 1
        .bss    _Fd, 1
        .bss    _Kpd, 1
        .bss    _Eids, 1
        .bss    _Kid, 1
        .bss    _VDH, 1
        .bss    _VDL, 1
        .bss    _YDH, 1
        .bss    _YDL, 1
        .bss    _Vdh, 1
        .bss    _Vdl, 1
        .bss    _Vd, 1
        .bss    _Vdr, 1
        .bss    _Dts, 1
        .bss    _Scalf, 1
; IQS controller variables
        .bss    _Fq, 1
        .bss    _Kpq, 1
        .bss    _Eiqs, 1
        .bss    _Kiq, 1
        .bss    _VQH, 1
        .bss    _VQL, 1
        .bss    _YQH, 1
        .bss    _YQL, 1
        .bss    _Vqh, 1
        .bss    _Vql, 1
        .bss    _Vq, 1
        .bss    _Vqr, 1
; Speed controller variables
        .bss    _Wref, 1
        .bss    _Ewr, 1
        .bss    _Fs, 1
        .bss    _Kps, 1
        .bss    _Kis, 1
        .bss    _YSH, 1
        .bss    _YSL, 1
        .bss    _IQH, 1
        .bss    _IQL, 1
        .bss    _Iqh, 1
        .bss    _Iql, 1
```

```
        .bss    _Iqref, 1
        .bss    _Dtspd, 1
        .bss    _Mode, 1
        .bss    _Idrated, 1             ; Rated Ids current
        .bss    _Nm2wre, 1              ; Mech rpm to elect r/s coeff
; Square wave ref. generator variables and flags
        .bss    _Swidest, 1
        .bss    _Swids, 1
        .bss    _Flids, 1
        .bss    _Swiqs, 1
        .bss    _Fliqs, 1
        .bss    _Swspd, 1
        .bss    _Flspd, 1
        .bss    _Idsq, 1
        .bss    _Iqsq, 1
        .bss    _Wrsq, 1
        .bss    _Flidest, 1
_Initial:
        .sect   "intvec1"
AINT:   B       _Dac_A_isr
BINT:   B       _Dac_B_isr
        .sect   "intvec2"
TINT:   B       _Tim_isr  ; Timer ISR
RCV:    B       _Adc_isr  ; ADC ISR

        .text
Mcnt:   .set    0
Tcnt:   .set    1
Incom:  .set    2                       ; Inverter command port
Instu:  .set    2                       ; Inverter status read
Dac:    .set    13                      ; DAC port addr, decoded by XF
TBL90:  .set    512                     ; offset for 90 deg, to pick Costhe value
Data:   .set    08                      ; Buffer data port
Addr:   .set    09                      ; Buffer address port
Readhost: .set  10                      ; Host read port
Refresh: .set   11                      ; Enable buffer refresh
Ssampc: .set    5                       ; 1 ms timer counter
Stmaxc: .set    25                      ; 5 ms timer counter
VLIM:   .set  207               ; Set Vds/Vqs limiter at 95% of max. value
ZEROP4: .set  10210
IQLIM:  .set    14                      ; (14/64)*30 = 6.6A, the set limit on IQ
SSV0    .usect  "svar0", 1
SSV1    .usect  "svar1", 1
SSTEMP1 .usect  "svar2", 1
SSTEMP2 .usect  "svar3", 1

INIT:   LDPK    0
        LALK    2000                    ; Timer = 200 us
        SACL    TIM
        SACL    PRD
        LACK    30                      ; IMR = 0000 0000 0001 1110 = TIM, RCV (ADC),
        SACL    IMR                     ;       INT2 (DAC-A) and INT1 (DAC-B) enabled
        SXF                             ; Initial setting of XF
        LDPK    6
        LALK    16384
        SACL    _HALF
        LALK    28377
        SACL    _SQRT32
        LALK    10210
        SACL    _ZERO4
        LACK    5
        SACL    _counter  ; Software timer for 1ms
        LACK    3
        SACL    _Bufah
        LACK    25
```

```
SACL    _Swt5           ; Software timer for 5ms
LALK    0FE00h
SACL    _Bufal
LALK    18918           ; Sqrt(1/3)
SACL    _Sqr1o3
LALK    3640
SACL    _DT             ; DT for theta integration
LALK    1186            ; Ksl=1.011, Kslmax=29.53
SACL    _Ksl
LALK    19191           ; Speed scaling factor =(2290/3910)*32767
SACL    _Spdscl
LALK    23930           ; Vdr/Vqr scaling factor
SACL    _Scalf          ;=(0.3115/(64/150))*32767
ZAC
SACL    _Vdr
SACL    _Vqr
SACL    _Idse
SACL    _Iqse
SACL    _Wr
SACL    _Theta
SACL    _YDH
SACL    _YDL
SACL    _YQH
SACL    _YQL
SACL    _Eids
SACL    _Eiqs
SACL    _ZERO
SACL    _VAR1
SACL    _VAR2
SACL    _Start          ; Start =0 in the beginning
SACL    _Iqr            ; Iq*=0
LACK    1
SACL    _ONE
LALK    7               ; DT for current error integration
SACL    _Dts
LALK    29320           ;
SACL    _SWT0
LACK    1
SACL    _Incomdata
OUT     _Incomdata, Incom   ; Clear M and T counters
ZAC
SACL    _Ssamp          ; 25 ms counter, not used anymore
SACL    _Wr1            ; Speed(new)
SACL    _Wr             ; Speed (old)
SACL    _Wrold
LDPK    _Wref           ; Ref speed
SACL    _Wref
SACL    _Ewr            ; Speed error
SACL    _YSH            ; Integral of speed error
SACL    _YSL
SACL    _IQH            ; Intermediate Iq*
SACL    _IQL
SACL    _Iqref          ; Iq* o/p of speed PI + limiter
LACK    20
SACL    _Dtspd          ; Speed error integral DT
LACK    5
SACL    _Mode           ; Mode of tuning, set to arbitrary value
LALK    10066           ; 10066=(9.216/30)*32767
SACL    _Idrated
LALK    29900           ; 30291 = 8.22/9, and 8.32=bits/mech rpm
SACL    _Nm2wre         ; (3938 mech rpm = 32767 bits). Thus
LDPK    _Mold           ; when multipled with 9, mech rpm -> p.u.
IN      _Mold, Mcnt     ; elect r/s conversion is achieved
IN      _Told, Tcnt     ; Initialize M(0) and T(0) counter values
LDPK    _Swidest    ; Initialize Id establishment timer = 50ms
```

```
        LALK    250
        SACL    _Swidest
        LACK    1               ; When FIRST 1 is subtracted, a 1 will be
        SACL    _Swids          ; generated, (initial 0 will be toggled)
        SACL    _Swiqs          ; to apply a square step wave in the first
        SACL    _Swspd          ; timer interrupt, wherever applicable
        ZAC                     ; ( Sw... -> 0 ==> Fl... -> 1, in the first
        SACL    _Flids          ; timer ISR execution, after Start=1
        SACL    _Fliqs
        SACL    _Flidest
        SACL    _Flspd
        RET
_Tim_isr: ; Context save        Ref to p 5-30, C2x users guide
        LARP    AR7
        MAR     *-
        SST1    *-
        SST     *-
        SACH    *-
        SACL    *-
        SPM     0
        SPH     *-
        SPL     *-
        MPYK    1
        SPL     *-
        RPTK    7
        POPD    *-
        SAR     AR0, *-
        SAR     AR1, *-
        SAR     AR2, *-
        SAR     AR3, *-
        SAR     AR4, *-
        SAR     AR5, *-
        SAR     AR6, *-
        LDPK    6
        LAC     _Ctim           ; Dummy counter incremented
        ADDK    1
        SACL    _Ctim
        CALL    _Sqwavgen
        LDPK    _counter
        LAC     _counter
        SUBK    1
        SACL    _counter
        BGZ     FCONT
        CALL    _Speed          ; Compute speed every 1 ms
        CALL    _Spdpilim ; Call speed regulator
        LACK    5
        SACL    _counter
FCONT:  CALL    _Idscont
        CALL    _Iqscont
        CALL    _Synsta
        CALL    _Phase23
        CALL    _Phase32
        CALL    _Thetagen
        CALL    _Invecrot
; Buffer storage functions
        SPM     1
; ------Select channel A variable to be stored
        LDPK    6
        LAC     _VAR1           ; VAR1 = 1, Store Sin(Theta)
        SUBK    1
        BNZ     VA2
        LAC     _Sinth
        SACL    _Disp
        B       VAEND
VA2:    SUBK    1               ; VAR1 = 2, Store Thetae
```

```
        BNZ     VA3
        LDPK    _Theta
        LAC     _Theta
        LDPK    _Disp
        SACL    _Disp
        B       VAEND
VA3:    SUBK    1
        BNZ     VA4             ;
        LAC     _Ia             ; VAR1 = 3, Store ADC A input, ib
        SACL    _Disp
        B       VAEND
VA4:    SUBK    1
        BNZ     VA5
        LDPK    _Iqsq           ; Store Iq square command for Var1=4
        LAC     _Iqsq
        LDPK    _Disp
        SACL    _Disp
        B       VAEND
VA5:    SUBK    1
        BNZ     VA6             ; Store Wr command for Var1 = 5
        LDPK    _Wrsq
        LAC     _Wrsq
        LDPK    _Disp
        SACL    _Disp
        B       VAEND
VA6:    SUBK    1
        BNZ     VA7             ;
        LDPK    _Idse
        LAC     _Idse
        ADLK    1000            ; VAR1 = 6, Store Idse
        LDPK    _Disp
        SACL    _Disp
        B       VAEND
VA7:    SUBK    1
        BNZ     VA8             ;
        LAC     _Eids           ; VAR1 = 7, Store Idse error
        SACL    _Disp
        B       VAEND
VA8:    SUBK    1
        BNZ     VBSTART                 ; VAR1 > 8 ==> Wrong value, check VAR2
        LAC     _Vdr            ; VAR1 = 8, Store Vdse*
        SACL    _Disp
VAEND:  SACL    _Bfadata  ; Store selected varaible in Ch A

; ------Select channel B variable to be stored
VBSTART: LDPK   6
        LAC     _VAR2           ; VAR2 = 1, Store costh
        SUBK    1
        BNZ     VB2
        LAC     _Costh
        SACL    _Disp
        B       VBEND
VB2:    SUBK    1               ; VAR2 = 2, Store We
        BNZ     VB3
        LDPK    _We
        LAC     _We
        LDPK    _Disp
        SACL    _Disp
        B       VBEND
VB3:    SUBK    1
        BNZ     VB4             ;
        LAC     _Ic             ; VAR2 = 3, Store ADC B input, ic
        SACL    _Disp
        B       VBEND
VB4:    SUBK    1
```

```
        BNZ     VB5             ;
        LAC     _Iqse           ; VAR2 = 4, Store Iqse
        SACL    _Disp
        B       VBEND
VB5:    SUBK    1
        BNZ     VB6             ;
        LDPK    _Wr1
        LAC     _Wr1
        SBLK    125
        LDPK    _Disp           ; VAR2 = 5, Store Rotor speed (elect)
        SACL    _Disp
        B       VBEND
VB6:    SUBK    1
        BNZ     VB7             ;
        LDPK    _Idsq
        LAC     _Idsq
        LDPK    _Disp           ; VAR2 = 6 Store Idse*
        SACL    _Disp
        B       VBEND
VB7:    SUBK    1
        BNZ     VB8             ;
        LAC     _Eiqs           ; VAR2 = 7 Store Iqse error
        SACL    _Disp
        B       VBEND
VB8:    SUBK    1
        BNZ     VAREND          ; VAR2 > 8 ==> Wrong value, quit
        ZALH    _Vqr            ; VAR2 = 8, Store Vqse*
        SACH    _Disp
VBEND:  SACL    _Bfbdata  ; Store selected variable in Ch A
VAREND:         LDPK    6
        LAC     _SWT0
        SUBK    1
        SACL    _SWT0
        BGZ     BUFRET
        LACK    2
        SACL    _SWT0
        CALL    _Bufstr
;       Context Restore
BUFRET: LARP    AR7
        MAR     *+
        LAR     AR6, *+
        LAR     AR5, *+
        LAR     AR4, *+
        LAR     AR3, *+
        LAR     AR2, *+
        LAR     AR1, *+
        LAR     AR0, *+
        RPTK    7
        PSHD    *+
        MAR     *+
        LT      *-
        MPYK    1
        LT      *+
        MAR     *+
        LPH     *+
        ZALS    *+
        ADDH    *+
        LST     *+
        LST1    *+
        EINT
        RET

        .copy   "daca.asm"
        .copy   "dacb.asm"
        .copy   "adcab.asm"
```

```
        .copy   "synsta.asm"
        .copy   "phase23.asm"
        .copy   "bufstr.asm"
        .copy   "speed.asm"
        .copy   "phase32.asm"
        .copy   "thetagen.asm"
        .copy   "invecrot.asm"
        .copy   "idscont.asm"
        .copy   "iqscont.asm"
        .copy   "spdcont.asm"
        .copy   "sqwavgen.asm"

;***************************************************************
*       DAC A Interrupt Service Routine
***************************************************************
        .title  "daca.asm"
; ISR for DAC channel A
_Dac_A_isr:
        LARP    AR7
        SAR     AR0, *-
        SAR     AR1, *-
        SAR     AR2, *-
        SAR     AR3, *-
        LDPK    0
        SST     SSV0
        LDPK    6
        OUT     _dacabuf, Dac
        NOP
        SXF
        LDPK    0
        LST     SSV0
        LARP    AR7
        LAR     AR3, *+
        LAR     AR2, *+
        LAR     AR1, *+
        LAR     AR0, *+
        EINT
        RET

;***************************************************************
*       DAC B Interrupt Service Routine
***************************************************************
        .title  "dacb.asm"
; DAC channel B ISR
_Dac_B_isr:
        LARP    AR7
        SAR     AR0, *-
        SAR     AR1, *-
        SAR     AR2, *-
        LDPK    0
        SST     SSV0
        LDPK    6
        OUT     _dacbbuf, Dac
        NOP
        RXF
        LDPK    0
        LST     SSV0
        LARP    AR7
        LAR     AR2, *+
        LAR     AR1, *+
        LAR     AR0, *+
        EINT
        RET
```

```
;*************************************************************
*           ADC Interrupt Service Routine
*************************************************************
          .title    "adcab.asm"
; ISR for ADC channels A and B
_Adc_isr:
          BIOZ      CHB
          LARP      AR7
          MAR       *-
          SST1      *-
          SST       *-
          SACH      *-
          SACL      *-
          SPM       0
          SPH       *-
          SPL       *-
          MPYK      1
          SPL       *-
          RPTK      7
          POPD      *-
          SAR       AR0, *-
          SAR       AR1, *-
          SAR       AR2, *-
          SAR       AR3, *-
          SAR       AR4, *-
          SAR       AR5, *-
          SAR       AR6, *-
          LDPK      0
          LAR       6, DRR
          LDPK      6
          SAR       6, _Ia              ; ADC Ch A --> Ia
          B         CHEXIT
CHB:      LARP      AR7
          MAR       *-
          SST1      *-
          SST       *-
          SACH      *-
          SACL      *-
          SPM       0
          SPH       *-
          SPL       *-
          MPYK      1
          SPL       *-
          RPTK      7
          POPD      *-
          SAR       AR0, *-
          SAR       AR1, *-
          SAR       AR2, *-
          SAR       AR3, *-
          SAR       AR4, *-
          SAR       AR5, *-
          SAR       AR6, *-
          LDPK      0
          LAR       6, DRR
          LDPK      6
          SAR       6, _Ic              ; ADC Ch B --> Ic
CHEXIT:
;         Context restore
          LARP      AR7
          MAR       *+
          LAR       AR6, *+
          LAR       AR5, *+
          LAR       AR4, *+
          LAR       AR3, *+
          LAR       AR2, *+
```

```
        LAR     AR1, *+
        LAR     AR0, *+
        RPTK    7
        PSHD    *+
        MAR     *+
        LT      *-
        MPYK    1
        LT      *+
        MAR     *+
        LPH     *+
        ZALS    *+
        ADDH    *+
        LST     *+
        LST1    *+
        EINT
        RET

;-------------------------------------------------
;   HOST COMMUNICATION ROUTINE
;-------------------------------------------------
; File:     HOSTDSP.ASM
; DATE: 12/07/93
; Date last modified: Feb 6, 95
; DESCRIPTION:
;         This routine controls the data flow between HOST and DSP,
; and was primarily designed to work under the DSPBUG environment.
; Some of the commands passed are essentially debugging tools, whereas
; others are used as permanent control functions.
;
;         From the DSPBUG environment, the SEND (S) function followed
; by a numeric code has the following effect:
;         CODE    ACTION
;         5       Start DSP buffer storage of selected variables (VAR1, VAR2)
;         6       VARIABLE A for buffer storage is given by the next value
;                 to be sent (Valid numbers are 1 through 8)
;         7       VARIABLE B (same as above)
;         8       Start / Stop command, 1=start
;         9       %Kp of Ids controller is given by the
;                 next value to be sent. (Max Kp=10)
;         10      %Ki value for Ids controller to be sent next, (max Ki=10)
;         11      %Kp value of Iqs controller next, (Max kp=10)
;         12      %Ki value of Iqs controller next, (Max Ki=10)
;         13      %Ids* value to be sent next, (Max Ids*=10)
;         14      %Iqs* value to be sent next, (Max Iqs*=20)
;         15      %Kp value for speed loop sent next, (Max Kp=20)
;         16      %Ki for speed loop sent next, (Max Ki = 20)
;         17      Command speed in mech. rpm sent next, (Max value=2200)
;
;         As observed above, some functions require only one command (e.g.,
; CODE 5), whereas other functions require two consecutive
; values to be sent to the program. The first value is the function code
; (e.g., 6 - 17), whereas the second is the actual value
; for the desired variable, such as torque reference, variable selected and
; so forth. A detailed description of the valid range of parameters is
; provided in the ASSEMBLY LANGUAGE SOFTWARE description.
;-------------------------------------------------------------
        .title  "hostdsp.asm"

_Hostdsp:
        LAC     _Mstep  ;Verify what is the current step.
        BZ      SBF0    ;_Mstep=0: Read function code.
        SUBK    1
        BZ      SBF1    ;_Mstep=1: Read parameter value.
        SUBK    1
        BZ      SBF2    ;_Mstep=2: Store information in proper variable.
```

```
        ZAC             ;_Mstep value is incorrect, make _Mstep=0
        SACL    _Mstep

; ---------- CASE _Mstep=0 : READ FIRST VALUE FROM HOST (FUNCTION Code)
SBF0:   IN      _Hostbuf, Hoststat  ;Read host status
        DINT
        BIT     _Hostbuf, 15                ;Test host status
        BBZ     L1                          ;Loop if no data is available
        EINT
        IN      _Command, Readhost          ;Read host 1st data (_Command)
        CALL    CASEA                       ;Single command functions
        B       L1

; ---------- CASE _Mstep=1 : READ SECOND VALUE FROM HOST (Par. value)
SBF1:   IN      _Hostbuf, Hoststat  ;Read host status
        DINT
        BIT     _Hostbuf, 15                ;Test host status
        BBZ     L1                          ;Loop if no data is avaiable
        EINT
        IN      _Command2, Readhost         ;Read host 2nd data (_Command2)
        LACK    2                           ;Make _Mstep=2
        SACL    _Mstep
        B       L1

; ---------- CASE _Mstep=2 : FUNCTIONS THAT REQUIRE 2 _CommandS.
;          IDENTIFY AND STORE INFORMATION IN PROPER PARAM./VAR.
SBF2:   ZAC                                 ;Set _Mstep=0
        SACL    _Mstep

S1:     LAC     _Command
        SUBK    6
        BNZ     S2              ;_Command=6 (Def. Buffer A var.)
        LAC     _Command2
        SACL    _VAR1           ;_VAR1 is the code for 1st var.
        B       CMDEND

S2:     LAC     _Command
        SUBK    7               ;_Command=7 (Def. Buffer B var.)
        BNZ     S3
        LAC     _Command2
        SACL    _VAR2           ;_VAR2 is the code for 2nd var.
        B       CMDEND

S3:     LAC     _Command
        SUBK    8
        BNZ     S4      ;_Command=8 (Start/ Stop)
        LAC     _Command2
        SACL    _Start
        B       CMDEND

S4:     LAC     _Command
        SUBK    9               ;_Command=9 (% Input Kp of Ids
        BNZ     S5              ; current controller)
        SPM     0               ; Kpmax = 5, %kp = (Kpact / Kpmax)*100,
        ZAC                     ; Internally, Kp(pu) = %kp(input)*328/32767
        LT      _Command2       ; %kp and Kp(pu) have different scales,
        MPYK    66              ; the one relevant to the user interface
        PAC                     ; is %kp, and the one relevant to the
        SACL    _Kpd            ; calculation is Kp(pu), these shouldn't be
        B       CMDEND                  ; confused with each other. Same goes for the
;                               ; rest of variables input at the screen
S5:     LAC     _Command
        SUBK    10              ;_Command=10 (% Input Ki of Ids
        BNZ     S6              ; current controller)
        SPM     0               ; Kimax=5.34, %ki=(kiact/kimax)*100,
```

```
        ZAC                     ; Internally, Ki(pu) = %ki(input)*62/32767
        LT      _Command2
        MPYK    62
        PAC
        SACL    _Kid
        B       CMDEND

S6:     LAC     _Command
        SUBK    11              ;_Command=11 (Input %Kp of Iqs
        BNZ     S7              ; current controller)
        SPM     0
        ZAC
        LT      _Command2
        MPYK    66
        PAC
        SACL    _Kpq
        B       CMDEND

S7:     LAC     _Command
        SUBK    12              ;_Command=12 (Input Ki of Iqs
        BNZ     S8              ; current controller)
        SPM     0
        ZAC
        LT      _Command2
        MPYK    62
        PAC
        SACL    _Kiq
        B       CMDEND
;
S8:     LAC     _Command
        SUBK    13              ;_Command=13 (% Input Ids reference for
        BNZ     S9              ; Ids current controller)
        SPM     0               ; Idmax=9.216, %id=idact/Idmax.
        ZAC                     ; Id(pu)=%id(input)*101/32767
        LT      _Command2
        MPYK    101
        PAC
        SACL    _Idr
        B       CMDEND
;;
S9:     LAC     _Command
        SUBK    14              ;_Command=14 (Input %Iqs reference
        BNZ     S10     ; of Iqs current controller)
        SPM     0               ; Iqsmax = 19.11A.
        ZAC
        LT      _Command2
        MPYK    209
        PAC
        SACL    _Iqr
        B       CMDEND

S10:    LAC     _Command
        SUBK    15
        BNZ     S11     ;_Command=15 (Mode of tuning)
        LDPK    _Command2       ; Mode=1 ==> Ids tuning
        LAC     _Command2       ; Mode=2 ==> Iqs tuning
        LDPK    _Mode           ; Mode=3 ==> Speed tuning
        SACL    _Mode           ; Mode=4 ==> Test run
        LDPK    _Command2
        B       CMDEND

S11:    LAC     _Command
        SUBK    16
        BNZ     S12     ;_Command=16 (%Kp for Speed loop,
        LDPK    _Command2       ; %kp=kp(act)/kpmax, kpmax=18
```

```
        LAC     _Command2       ; internally, kp(pu)=(kp(act)/18.33)*32767)
        LDPK    _Kps
        SACL    _Kps
        ZAC
        SPM     0
        LT      _Kps
        MPYK    322             ; 322=(18/100)*32767/18.33
        PAC
        SACL    _Kps
        LDPK    _Command2
        B       CMDEND

S12:    LAC     _Command
        SUBK    17
        BNZ     S13     ;_Command=17 (%Ki for speed loop
        LDPK    _Command2       ; %ki=100*ki(act)/kimax, kimax=11.2
        LAC     _Command2       ; internally, ki(pu)=(ki(act)/11.2)*32767)
        LDPK    _Kis
        SACL    _Kis
        ZAC
        SPM     0
        LT      _Kis
        MPYK    328             ; 54=(20/100)*32767/122.24
        PAC
        SACL    _Kis
        LDPK    _Command2
        B       CMDEND

S13:    LAC     _Command
        SUBK    18
        BNZ     CMDEND      ;_Command=18 (Speed command in mech rpm)
        SPM     1
        LDPK    _Command2       ; Internally converted to elect r/s,
        LAC     _Command2       ; by relation Wref(elect)=(Nref(mech))*2*Pi/30
        LDPK    _Wref           ; Pi/15=0.20944 = 6863(bit val)
        SACL    _Wref
        ZAC
        LT      _Wref
        MPY     _Nm2wre         ; Nm2wre=8.33/9, when multiplied by 9,
        APAC                    ; converts to p.u. elect r/s from mech rpm
        SACH    _Wref           ; See note in tim_isr for explanation
        SPM     0
        ZAC
        LT      _Wref
        MPYK    9
        APAC
        SACL    _Wref
        LDPK    _Command2
        B       CMDEND

CMDEND: ZAC                     ;If the instruction has been acknowledged,
        SACL    _Mstep          ; set _Mstep=0 and loop again.
        B       L1

; ---------- RESET BUFFER STORAGE SUBROUTINE
BUFSTO: DINT
        ZAC
        SACL    _Bufah          ;Used as a debug tool.
        SACL    _Bufal
        SACL    _Col
        SACL    _Low
        EINT
        RET

; ---------- CASE A: Single instruction routines
```

```
CASEA:  LAC     _Command                        ;_Command=0 :Stop system
        BNZ     CMD1
        LACK    1
        SACL    _Incomdata
        OUT     _Incomdata,Incom    ;INCON=1: Disable inverter gate
        B       AEND

CMD1:   LAC     _Command                :_Command=1 : Set rated flux
        SUBK    1
        BNZ     CMD4                    ; CMD = 3, 4 skipped
        LACK    3
        SACL    _Incomdata
        OUT     _Incomdata,Incom    ;Incom=3 : Enable inverter gate
        B       AEND

CMD4:   LAC     _Command                :_Command=5: Start buffer storage
        SUBK    5                       :_VAR1 and _VAR2 define the vars.
        BNZ     CMD5                    ; being stored
        CALL    BUFSTO
        B       AEND

CMD5:   LACK    1                       ;Case of 2 instr. commands
        SACL    _Mstep                  ; Make _Mstep=1 and return
        RET

AEND:   ZAC                             ;If the instruction was acknowledged
        SACL    _Mstep                  ; set _Mstep=0          and return
        RET

;-------------------------------------------------------------------------------
; Buffer Storage Subroutine for Data Acquisition and Display
;-------------------------------------------------------------------------------
        .title      "bufstr.asm"

_Bufstr: ; ---------- DATA STORAGE CH-A
        LDPK    6
        ZALS    _Bufal              ;Address of past data.
        ADDH    _Bufah
        ADDK    2
        SACH    _Bufah              ;Address of current data.
        SACL    _Bufal
        SACH    _BST_AUX,7          ;Shift by seven the linear address.
        ANDK    1FEH                ;Mask 9 lsb (bits 0-8) and
        SACL    _Low                ; store into LOW (ROW address).
        LAC     _BST_AUX                    ;Mask 9 msb (bits 9-17) and
        ANDK    1FFH                ;   store into COL (column address)
        SACL    _Col
        OUT     _ONE,11             ;Turn-off refresh mode.
        IN      _BST_AUX,8
        OUT     _Low,Addr           ;Send ROW and COL to PA 9
        OUT     _Col,Addr
        OUT     _Bfadata,Data       ;Write channel A: white line in GRAPH.
        OUT     _ZERO,11                    ;Enable refresh mode.
; ----------Verify if buffer memory is full.
        LALK    1FFH
        SUB     _Col
        BGZ     JMPFUL              :COL<1FFh --> not full.
        LALK    0FE00H              ;Buffer is full: freeze COL and ROW
        SACL    _Bufal              ;   values to last position in buffer.
        LACK    3
        SACL    _Bufah

; ----------DATA STORAGE CH-B
JMPFUL: LAC     _Low                ;Point to odd location in memory.
        ADDK    1
```

```
        SACL    _Low

        OUT     _ONE,11         ;Turn-off refresh mode.
        IN      _BST_AUX,8
        OUT     _Low,Addr       ;Send memory location (ROW,COL).
        OUT     _Col,Addr
        OUT     _Bfbdata,Data   ;Write channel B: red line in GRAPH.
        OUT     _ZERO,11        ;Enable buffer refresh.
        RET

;-----------------------------------------------------------------------
; Speed Measurement Subroutine
;-----------------------------------------------------------------------
        .title  "speed.asm"
        .global _Dir
        .bss    _Dir, 1

_Speed: LDPK    _Start
        LAC     _Start
        BZ      SFEXIT
        LDPK    _Mode
        LAC     _Mode
        SUBK    3
        BGEZ    Spdcalc
        LDPK    _Wrl
        ZAC
        SACL    _Wrl
        LDPK    _Iqref
        SACL    _Iqref
        B       SFEXIT
Spdcalc: LDPK   _Wref
        LAC     _Wref
        BNZ     Calspd
        LDPK    _Wrl
        ZAC
        SACL    _Wrl
        B       SFEXIT
Calspd: LDPK    6
        SPM     0

        IN      _Mnew,Mcnt      :Input M, T counters content
        IN      _Tnew,Tcnt

        LAC     _Mnew           ; Compute delta M
        SUB     _Mold
        SACL    _Delm,6  ; dM=32*(MNEW-MOLD)

        LAC     _Tnew
        SUB     _Told
        SACL    _Delt           ;delt = tnew - told

        ZALH    _Delm           ; wrtemp = delm / delt
        ABS                     ;
        RPTK    15
        SUBC    _Delt
        SACL    _Wrl
        ZAC
        LT      _Wrl
        LDPK    _Spdscl
        MPY     _Spdscl
        APAC                    ; SPM=0, so, added twice
        APAC
        LDPK    _Wrl
        SACH    _Wrl
        LDPK    _Wrsq
```

```
        LAC     _Wrsq
        BGZ     Noset
        LDPK    _Wrl
        ZALS    _Wrl
        LDPK    _Wref
        SUB     _Wref
        BLEZ    Noset
        LDPK    _Wrl
        ZAC
        SACL    _Wrl
Noset:  ZAC
        LDPK    _Dir
        SACL    _Dir
        LDPK    _TEMP0
        IN      _TEMP0,Instu    ;DIR CHECK
        BIT     _TEMP0,15
        BBNZ    STP12           ;If positive, proceed as is.
        LACK    1
        LDPK    _Dir
        SACL    _Dir
        LDPK    _Wrl
        ZALS    _Wrl
        NEG
        SACL    _Wrl

STP12:  DMOV    _Mnew           ; mold <-- mnew
        DMOV    _Tnew           ; told <-- tnew
        DMOV    _Wrl            ; WROLD <-- Wr
SFEXIT: LDPK    6
        RET

*************************************************************************
*       This routine implements various modes of optimization, by generating
*       command current / speed values, depending on the mode of tuning, as
*       follows: (Idsq, Iqsq and Wrsq are the ref outputs, to various PI's)
*
*       The mode definitions are:
*       Mode 1 = Ids tuning, Mode 2 = Iqs tuning,
*       Mode 3 = speed tuning, Mode 4 = Iqs tuning at non-zero speed
*
*       Mode 1: Idsq = Square wave, 31.25 Hz, with user-programmable magnitude
*               Iqsq = Wrsq = 0
*       Mode 2: Idsq = Id(rated), Iqsq = Square wave, 31.25 Hz, with prog. mag.
*               Wrsq = 0
*       Mode 3: Idsq = Id(rated), Iqsq = o/p of speed PI, Wrsq = sq. wave,
*               2.5 Hz, with prog. mag.
*       Mode 4: Same as Mode 2, with Wr != 0
*       For modes 2, 3 and 4, before the optimization begins, rated Id is
*       automatically established within the first 50 ms
*       LAST MODIFIED: March 24, '95
*       Author: Sunil M. Chhaya
*************************************************************************
        .title  "sqwavgen.asm"

_Sqwavgen:
        SPM     1
        LDPK    _Start          ; If Start=0, quit
        LAC     _Start
        BNZ     Chmode
        LDPK    _Idsq
        ZAC
        SACL    _Idsq
        LDPK    _Iqsq
        SACL    _Iqsq
        LDPK    _Wrsq
```

```
        SACL    _Wrsq
        B       Sqwexit
Chmode: LDPK    _Mode
        LAC     _Mode           ; Branch to mode-specific block
        BZ      Waidest
        SUBK    1
        BZ      Mod1
        SUBK    4               ; If Mode = 5 (initialized val), do nothing
        BZ      Sqwexit
Waidest: LDPK   _Idrated        ; If mode = 2/3/4, first establish Id(rated)
        ZALH    _Idrated        ; and wait for rated Id to be established
        LDPK    _Idsq
        SACH    _Idsq
        LDPK    _Flidest        ; Flidest is set to 1 when Id is established
        LAC     _Flidest        ; And after that, Id*=Id(rated) forever
        BNZ     Mod234
        LDPK    _Swidest        ; Swidest corresponds to 50 ms, the wait time
        ZALS    _Swidest
        SUBK    1
        SACL    _Swidest        ; If 50 ms is over, rated Id is established
        BNZ     Oiqwr           ; Now, jump to modes 2/3/4 related code
        LACK    1
        LDPK    _Flidest
        SACL    _Flidest
        B       Mod234
Oiqwr:  ZAC                     ; Otherwise,
        LDPK    _Iqsq           ; Iq* = Wr* = 0, while rated Id is established
        SACL    _Iqsq
        LDPK    _Wrsq
        SACL    _Wrsq
        B       Sqexit
Mod234: LDPK    _Mode           ; Mode = 2?
        LAC     _Mode
        BZ      Mod2            ; Mode = 0 is mode 2 extended into nonzero Wr
        SUBK    2
        BZ      Mod2            ; Branch to Mode 2
        SUBK    1
        BZ      Mod3            ; Mode = 3 ==> Branch to Mode 3
        SUBK    1
        BZ      Mod4            ; Mode = 4 ==> Branch to Mode 4
Mod1:   LDPK    _Iqsq           ; Iqr = Wref = 0, by setting Fliqs, Flspd = 0
        ZAC
        SACL    _Iqsq
        LDPK    _Wrsq
        SACL    _Wrsq
        LDPK    _Swids
        LAC     _Swids          ; Swids-=1
        SUBK    1               ; Is Swids = 0? (16 ms interval = 80*0.0002s)
        SACL    _Swids
        BNZ     Sqexit
        LACK    80              ; Yes ==> Reload Ids software timer
        SACL    _Swids
        LDPK    _Flids          ; And toggle Flids flag, to simulate square
        LAC     _Flids          ; wave reference
        BZ      One
        ZAC
        SACL    _Flids
        B       Apdsq
One:    LALK    32767           ; 1.0 p.u. = 32767! square wave is +10V-0V
        SACL    _Flids
Apdsq:  ZAC                     ; Multiply Id ref command with the square wave
        LT      _Flids          ; to get Id ref square wave
        LDPK    _Idr
        MPY     _Idr
        APAC
```

```
        LDPK    _Idsq
        SACH    _Idsq
        B       Sqexit          ; Mode 1 operation over, quit
Mod2:   LDPK    _Wrsq           ; Wref = 0. by setting Flspd = 0
        ZAC
        SACL    _Wrsq
        LDPK    _Swiqs
        LAC     _Swiqs          ; Swiqs -=1
        SUBK    1               ; Is Swiqs = 0? (16 ms interval = 80*0.0002s)
        SACL    _Swiqs
        BNZ     Sqexit
        LACK    80              ; Yes ==> Reload Iqs software timer
        SACL    _Swiqs
        LDPK    _Fliqs          ; And toggle Fliqs flag, to simulate square
        LAC     _Fliqs
        BZ      Oneq            ; wave reference
        ZAC
        SACL    _Fliqs
        B       Apqsq
Oneq:   LALK    32767
        SACL    _Fliqs
Apqsq:  ZAC                     ; Multiply Iq ref command with the square wave
        LT      _Fliqs          ; to get Iq ref square wave
        LDPK    _Iqr
        MPY     _Iqr
        APAC
        LDPK    _Iqsq
        SACH    _Iqsq
        B       Sqexit          ; Mode 2 operation over, quit
Mod3:   LDPK    _Idrated
        ZALH    _Idrated        ; Idref = Ids rated val
        LDPK    _Idsq
        SACH    _Idsq
        LDPK    _Iqref
        ZALH    _Iqref          ; Iq* = Output of Speed PI
        SACH    _Iqsq
        LDPK    _Swspd
        LAC     _Swspd          ; Swspd -= 1
        SUBK    1               ; Is Swspd = 0? (6 sec interval=30000*0.0002s)
        SACL    _Swspd
        BNZ     Sqexit
        LALK    30000           ; Yes ==> Reload Wref software timer
        SACL    _Swspd
        LDPK    _Flspd          ; And toggle Flspd flag, to simulate square
        LAC     _Flspd
        BZ      Ones            ; wave reference
        ZAC
        SACL    _Flspd
        B       Apssq
Ones:   LALK    32767
        SACL    _Flspd
Apssq:  ZAC                     ; Multiply Wref command with the square wave
        LT      _Flspd          ; to get Wref square wave
        LDPK    _Wref
        MPY     _Wref
        APAC
        LDPK    _Wrsq
        SACH    _Wrsq
        B       Sqexit          ; Mode 3 operation over, quit
Mod4:   LDPK    _Idrated
        ZALH    _Idrated
        LDPK    _Idsq
        SACH    _Idsq
        LDPK    _Iqref
        ZALH    _Iqref
```

```
        LDPK    _Iqsq
        SACH    _Iqsq
        LDPK    _Wref
        ZALH    _Wref
        LDPK    _Wrsq
        SACH    _Wrsq
Sqexit: ; Current comparators implemented below
        LDPK    _Idsq
        ZALH    _Idsq
        LDPK    _Idse
        SUBH    _Idse
        LDPK    _Eids
        SACH    _Eids
        LDPK    _Iqsq
        ZALH    _Iqsq
        LDPK    _Iqse
        SUBH    _Iqse
        LDPK    _Eiqs
        SACH    _Eiqs
Sqwexit: LDPK   6
        RET

;-----------------------------------------------------------------
; Ids Current Controller
;-----------------------------------------------------------------
        .title  "idscont.asm"
_Idscont:
        LDPK    6
        SPM     1
        SSXM
; If Start flag = 0, do not apply controller actions, clamp outputs to 0
        LDPK    _Start
        LAC     _Start
        BNZ     Aplact
        ZAC
        LDPK    _Vdr
        SACL    _Vdr
        SACL    _Vqr
        SACL    _YDH
        SACL    _YDL
        SACL    _YQH
        SACL    _YQL
        SACL    _Eids
        SACL    _Eiqs
        B       Conexit
Aplact: LACK    1       ; Set positive sign flag by default
        LDPK    _Fd
        SACL    _Fd
        ZAC
        LT      _Kpd
        MPY     _Eids           ; vp = Kp * e
        LT      _Kid
        MPYA    _YDH            ; vi = Ki * y (integral of error)
        APAC                    ; <ACC> = v = vp + vi
        SACH    _VDH
        SACL    _VDL            ; Store V
        BGEZ    Nneg            ; Set sign flag if V < 0
        LACK    0
        SACL    _Fd
        ZALH    _VDH
        ADDS    _VDL
Nneg:   ABS                     ; Find range of V
        SACH    _Vdh            ; Vdh-l contain |VDH-L|
        SACL    _Vdl
        ZALS    _Vdh
```

```
        SUBK   218
        BLEZ   Upscl        ; If |V| > 0.95, Set |V| = 0.95
        LACK   VLIM
        SACL   _Vdh
        ZAC
        SACL     _Vdl
Upscl:  ZALH   _Vdh         ; Upscale the value of V in linear range
        ADDS   _Vdl           ; For better precision
        RPTK   5
        SFL
        SACH     _Vdr                   ; Vdr = scaled Vd
Apsgn:  LAC    _Fd          ; Is the sign of V negative?
        BGZ    Calyd          ; No ==> Don't negate |V|
        ZALH   _Vdr           ; Else V = -|V|
        NEG
        SACH   _Vdr
Calyd:  LDPK     _Eids
        LAC      _Eids
        BLEZ     Edint
        SBLK     1000
        BLEZ     Conexit
Edint:  ZAC              ; Implement Integral of error
        LDPK     _Eids
        LT     _Eids       ; Dy = E * Ts
        MPY    _Dts
        APAC
        ADDS   _YDL          ; Y(new) = Y(old) + Dy
        ADDH   _YDH
        SACH   _YDH
        SACL   _YDL          ; Store new Y in buffer
Conexit:LDPK       6
        RET

;-------------------------------------------------------------------------
; Iqs Current Controller
;-------------------------------------------------------------------------
        .title  "iqscont.asm"

_Iqscont:
        LDPK   6
        SPM    1
        SSXM
; If Start flag = 0, do not apply controller actions, clamp outputs to 0
        LDPK     _Start
        LAC      _Start
        BZ       Conqexit
        LDPK     _Mode
        LAC      _Mode                ; If mode = 1, Vqr=0
        BZ       Chidest
        SUBK     1                    ; Else, Vqr = controller output
        BGZ      Chidest              ; Check for Id established.
Vqr0:   LDPK     _Vqr                 ;
        ZAC
        SACL     _Vqr
        SACL     _YQH
        SACL     _YQL
        LDPK     _Iqref
        SACL     _Iqref
        LDPK     _YSL
        SACL     _YSL
        SACL     _YSH
        B        Conqexit
Chidest: LDPK    _Flidest    ; If Id rated established, apply cont. action
        LAC      _Flidest    ; else, Vqr:= 0
        BZ       Vqr0
```

```
Aplqcont: LDPK    _Fq
          LACK  1          ; Set positive sign flag by default
          SACL  _Fq
          ZAC
          LDPK    _Kpq
          LT    _Kpq
          LDPK    _Eiqs
          MPY   _Eiqs
          LDPK    _Kiq      ; vp = Kp * e
          LT    _Kiq
          LDPK    _YQH
          MPYA  _YQH        ; vi = Ki * y (integral of error)
          APAC
          LDPK    _VQH
          SACH  _VQH
          SACL  _VQL        ; Store V
          BGEZ  Nqneg       ; Set sign flag if V < 0
          LACK  0
          LDPK    _Fq
          SACL  _Fq
          LDPK    _VQH
          ZALH  _VQH
          ADDS  _VQL
Nqneg:    ABS               ; Find range of V
          LDPK    _Vqh
          SACH  _Vqh
          SACL  _Vql
          ZALS  _Vqh
          SUBK  218
          BLEZ  Upsclq      ; If |V| > 0.95, Set |V| = 0.95
          LACK  VLIM
          SACL  _Vqh
          ZAC
          SACL    _Vql
Upsclq:   ZALH  _Vqh        ; Upscale the value of V in linear range
          ADDS  _Vql        ; For better precision
          RPTK  5
          SFL
          LDPK    _Vqr
          SACH    _Vqr
Apsgnq:   LDPK    _Fq
          LAC   _Fq         ; Is the sign of V negative?
          BGZ   Calyq       ; No ==> Don't negate |V|
          LDPK    _Vqr
          ZALH  _Vqr        ; Else V = -|V|
          NEG
          SACH  _Vqr
Calyq:    LDPK    _Eiqs
          LAC     _Eiqs
          BLEZ    Eiqint
          SBLK    1000
          BLEZ    Conqexit
Eiqint:   ZAC               ; Implement Integral of error
          LDPK    _Eiqs
          LT    _Eiqs       ; Dy = E * Ts
          LDPK    _Dts
          MPY   _Dts
          APAC
          LDPK    _YQL
          ADDS  _YQL        ; Y(new) = Y(old) + Dy
          ADDH  _YQH
          SACH  _YQH
          SACL  _YQL        ; Store new Y in buffer
Conqexit: LDPK    6
          RET
```

```
;-----------------------------------------------------------------
; Vector rotation performed here
;-----------------------------------------------------------------
        .title  "synsta.asm"
_Synsta:
        SPM     1
        LDPK    _Vqr
        ZAC
        LT      _Vqr
        MPY     _Sinth
        APAC
        NEG
        LT      _Vdr
        MPY     _Costh
        APAC
        SACH    _Vdss_i         ; Vdss = -Vqse*sin+Vdse*cos
        ZAC
        LT      _Vqr
        MPY     _Costh
        LT      _Vdr
        MPYA    _Sinth
        APAC
        SACH    _Vqss_i         ; Vqss = Vqse*cos+Vdse*sin
        RET


;-----------------------------------------------------------------
; Two-phase to three-phase conversion done below:
;-----------------------------------------------------------------
        .title  "phase23.asm"
_Phase23:
        LDPK    _Start
        LAC     _Start
        BNZ     Aptrans
        ZAC
        LDPK    _Vb0
        SACL    _Vb0
        SACL    _Vc0
        B       Sndbuf
Aptrans: LDPK   6
        SPM     1
        ZAC
        LT      _Vqss_i
        MPY     _HALF
        LT      _Vdss_i
        MPYA    _SQRT32
        APAC
        NEG
        SACH    _Vb0            ; Vb0 = -0.5*Vqss-0.866*Vdss
        ZAC
        LT      _Vqss_i
        MPY     _HALF
        APAC
        NEG
        LT      _Vdss_i
        MPY     _SQRT32
        APAC
        SACH    _Vc0            ; Vc0 = -0.5*Vqss+0.866*Vdss
        ZAC
        LT      _Vb0            ; apply scaling here
        MPY     _Scalf          ; Scalf = 0.7305 (23930) = 0.3115/0.4267
        APAC                    ; 0.4267 = 64/150, from id/iq cont.
        SACH    _Vb0
```

```
        ZAC
        LT      _Vc0            ; apply scaling here
        MPY     _Scalf
        APAC
        SACH    _Vc0
Sndbuf: ZAC
        LAC     _Vb0
        ADDK    164             ; DAC A error adjustment = 0.05 V
        SACL    _dacabuf
        ZAC
        LAC     _Vc0
        ADDK    131             ; DAC B error adjustment = 0.04 V
        SACL    _dacbbuf
        RET
```

```
;-------------------------------------------------------------------
; Three phase to two phase current conversion
;-------------------------------------------------------------------
    .title "phase32.asm"
_Phase32:
   LDPK         6
    SPM   1
    ZALH  _Ia
    NEG
    SACH  _Iqss         ; Iqss = ia
    LT    _Ic
    MPY   _Sqr1o3
    PAC
    LT    _Ia
    MPYA  _Sqr1o3
    APAC
    NEG                 ; both Ia and Ic are negated when read
    SACH  _Idss         ;Idss = (ia + 2ic)/(sqrt 3)
    RET
```

```
;-------------------------------------------------------------------
; Unit Vector and Electrical Angle Generation routine
;-------------------------------------------------------------------
        .title "thetagen.asm"
        .bss    TEMP4, 1
        .bss    TEMP5, 1
_Thetagen:
        LDPK    _Start
        LAC     _Start
        BNZ     Apint
Seth0:  LDPK    _Thetah
        ZAC
        SACL    _Thetah
        SACL    _Thetal
        SACL    _Wr1
        B       Getsc
Apint:  LDPK    _Mode           ; If Mode=Ids tuning (1), then theta=0
        LAC     _Mode           ; Else, theta=integral(We), We=Wr+Ksl*Iqs*
        BZ      Estid           ; For mode = 0, Iqs tuning done at a non-zero
        SUBK    1               ; speed! (after establishing Idref)
        BZ      Seth0           ; For Mode = 1, Theta = 0
        SUBK    1
        BNZ     Estid           ; For Mode > 2, Theta computed after Id est.
        ZAC                     ; For Mode = 2, Wr = 0, Theta = integ(Ksl*Iqs)
        LDPK    _Wr1
        SACL    _Wr1
        B       Compwe
```

```
Estid:    LDPK    _Flidest   ; Also, in Mode 2, Wr1=0
          LAC     _Flidest   ;
          BZ      Seth0
Compwe: SPM       0
          LDPK    _Wr1
          ZALH  _Wr1         ;We computation.
          LDPK    _Ksl
          LT    _Ksl
          LDPK    _Iqsq
          MPY   _Iqsq
          APAC
          APAC
          LDPK    _We
          SACH  _We          ;We = Wr + Kslip*Iqe

; ---------- Calc. angle theta
BP1:  ROVM              ;Reset overflow mode
          LDPK    _Thetah
          ZALH  _Thetah      ;
          ADDS  _Thetal
          LT    _We
          MPY   _DT          ; DT is twice its scaled val, for SPM = 0
          APAC             ;Theta = Theta + We*DT
          SACL  _Thetal
          SACH  _Thetah
;
;---------- Get  SIN(theta), COS(theta) from look-up table
Getsc:    SSXM             ;Set sign extension
          LDPK    _Thetah
          LT    _Thetah
          MPYK  1
          PAC
          SACL  TEMP4
          LT    TEMP4
          MPYK  2048
          PAC
          SACH  TEMP5
          LAC   TEMP5
          BGEZ  POSIGN
          ADLK  1024         ;If negative, add 180 degrees
          ANDK  1023
;
NESIGN: SOVM
          ADLK  SINTBL
          TBLR  _Sinth
          ADLK  TBL90
          TBLR  _Costh
          LAC   _Sinth
          NEG
          SACL  _Sinth
          LAC   _Costh
          NEG
          SACL  _Costh
          B     SINEEND

POSIGN: SOVM             ;Set overflow mode
          ADLK  SINTBL
          TBLR  _Sinth
          ADLK  TBL90
          TBLR  _Costh

SINEEND:RSXM
          ZALH    _Thetah
          ADDS    _Thetal
          SFL
```

```
        SACH    _Theta
        SSXM
        RET

;---------------------------------------------------------------
;   Performs stationary to synchronous frame current vector rotation
;---------------------------------------------------------------
        .title  "invecrot.asm"
_Invecrot:
    LDPK        6
    SPM   1
    ZAC
    LT    _Sinth
    MPY   _Idss
    APAC
    NEG
    LT    _Costh
    MPY   _Iqss
    APAC
    SACH  _Iqse             ;Iqse = iqss*cos - idss*sin
    ZAC
    LT    _Sinth
    MPY   _Iqss
    LT    _Costh
    MPYA  _Idss
    APAC
    SACH  _Idse             ;Idse = iqss*sin + idss*cos
        RET
```