Real-time Digital Simulation of Switching Power Circuits

by

Venkata R. Dinavahi

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

0-612-53721-8

Real-Time Digital Simulation of Switching Power Circuits

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
in the University of Toronto

Venkata R. Dinavahi

2000

Abstract

Sophisticated power electronic apparatus and their digital control systems are finding increasing applications in electric power systems at generation, transmission, distribution and utilization levels. It is essential to carry out rigorous performance evaluation of the power electronic equipment and their digital controllers prior to their commissioning on the host power system. The current trend to achieve that goal is to interface a real-time digital simulator representing the power electronic apparatus and the host power system with the digital controller. This thesis addresses the issue of synchronization between the output signals of the digital controller for firing power electronic switches and the discrete time-step of the real-time simulator. It is shown that lack of such synchronization can lead to severe inaccuracies in the simulation results.

A novel real-time simulation algorithm is proposed for accounting incoming switching events in fixed step-size digital simulation. This algorithm relies on the registration of the timing of the switching events and a subsequent *correction* procedure to calculate the system state. Off-line time domain simulations of a Pulse Width Modulated (PWM) Voltage Source Inverter (VSI) system demonstrate a ten-fold improvement in accuracy of the proposed algorithm over the fixed step-size algorithm using the same step-size. Practical feasibility of the proposed algorithm is demonstrated by implementation on a digital computing platform comprising of a DSP and a FPGA. The hardware and software design process follows a modular approach which makes it amenable for implementation on next generation processors. A 5kVA experimental set-up of the PWM VSC system is used to verify the results of the real-time simulation.

Acknowledgements

The author would like to express his thanks to Prof. Richard Bonert and Prof. Reza Iravani for their guidance, encouragement and financial support throughout the course of this work. Their discerning suggestions always proved very useful and were much appreciated.

The author would also like to acknowledge financial support from the University of Toronto.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Overall Perspective

Recent years have seen an increasing power demand coupled with an increasing usage of existing transmission and distribution resources. Various factors such as environmental concerns, right-of-way problems for new transmission lines and complex interconnections of networks are preventing an expansion and forcing a closer-to-limit operation of existing resources. The most recent trend is deregulation of the power industry which was initiated to increase competition in all sectors of the industry. All these developments demand a more efficient use of the existing resources and a better control of power flow. The industry solution to handle increasing power demand, better control of power flow and improve system stability at the same time, came in the form of new technologies such as FACTS (Flexible AC Transmission Systems) controllers and CP (Custom Power) controllers which are based on high-power electronics. Sophisticated power electronic controllers are also frequently employed for low-power industrial applications such as drive systems. The design and development of these power electronic apparatus is a complex and highly specialized process involving considerable investment. A comprehensive analysis of these technologies is essential to understand their operating characteristics, the range of control and their overall impact on the power system at large. In this respect only off-line transient simulation is not adequate, real-time simulation is needed for rigorous closed loop testing and correction of any potential problems such as harmonic injection and resonance between the power electronic controllers and the power system. A need therefore exists to develop techniques for accurate and efficient real-time simulation of systems containing conventional power system components and newer power electronic apparatus.

Traditionally, Transient Network Analyzers (TNAs), based on analog technologies, were used for real-time simulation and testing of new control strategies before they were installed on a power system. With the availability of low-cost and powerful digital computers and fast execution digital signal processors, analog simulators and controllers are giving way to fully *digital* real-time simulators and controllers. As a consequence of this trend new technical challenges are surfacing. One important problem, in particular, is that of interfacing a real-time digital simulator modeling power electronic systems with a digital controller. A major characteristic of digital simulation is that it is inherently discrete in nature. The system state can be computed not continuously but only at a certain time step either fixed or variable (a variable time step is allowed only in *off-line* digital simulation). Therefore, a real-time digital simulator is unable to respond immediately to external events coming in between two calculation steps. On the other hand, digital controllers for power electronic apparatus output firing pulses whose timing is important for the desired behavior of the apparatus. When a firing signal in the simulation is not taken into account at its proper time, switching is delayed which can have undesirable side effects on the output. Thus, when a real-time digital simulator is interfaced with a digital controller the overriding concern is that of synchronizing the operation of the simulator with the timing of the incoming firing signal. A lack of such synchronization can lead to errors in the simulation output and a possible run-off in the solution of system state due to accumulation of local errors over time. In an off-line simulation these spurious effects are usually avoided by using small integration time steps, however, such an approach is not feasible in real-time.

This thesis addresses the aforementioned problem. The interaction of a real-time digital simulator and a digital controller is investigated to find out the possible cause(s) of errors in simulation. The study system is a Pulse Width Modulated Voltage Source Inverter (VSI) system. Detailed off-line simulation is performed to examine the effects of limited time-steps, using the fixed step-size method on the simulation output. A new method is proposed for synchronization and alleviation of simulation errors. The proposed real-time simulation method has been implemented on a fully digital platform comprising of a Digital Signal Processor (DSP) and a Field Programmable Gate Array (FPGA). An experimental setup of the VSI system is used to validate the results from the real-time simulator.

1.2 Evolution of Electromagnetic Transient Simulation

Since the thesis topic is closely related to electromagnetic transient simulation it is worthwhile to examine briefly the past and current practices in this area. The tools [1] for electromagnetic transient simulation can be categorized into four main groups according to their order of evolution

1. Transient Network Analyzers (TNAs)
2. Analog and Hybrid Simulators
3. Fully Digital Off-line Simulators
4. Real-Time Digital Simulators

1.2.1 Transient Network Analyzers

A TNA is an assemblage of scaled down models of actual physical components operating at much lower voltages and currents. Generators, transformers, transmission lines, power electronic converters are connected as in a real network to assure real-time operation. Their real-time simulation capability made them crucial for testing control hardware before it was applied to the actual system. Because a TNA is expensive and complex to build only major utilities or research centers can afford to own one. Major drawbacks of TNAs are that they are hard to change or readjust and that all system parameters cannot be linearly scaled down based on physical components e.g. system resistance, machine mechanical parameters etc.

1.2.2 Analog and Hybrid Simulators

Analog simulators employed analog electronics comprising of integrators, summers and multipliers to solve the equations describing a system. This approach helped reduce the physical space and cost of the simulator many fold. With the development of digital computers hybrid simulators evolved. The term *hybrid* can be construed as a combination of analog computers and their digital counterparts or a combination of passive physical equipment and digital computer models. In either case the *hybrid* concept arose at all (instead of a direct leap from physical TNAs to fully digital simulators) was because during the course of simulator evolution people realized that either there were not enough digital models available or the models were not accurate enough to merit serving as a replacement to the analog components. In certain cases analog models were needed to simulate fast transients in real-time due to the digital models being too complex. Hybrid simulators can be thought of as an evolutionary stage which will quickly pass away in favor of fully digital simulators.

1.2.3 Fully Digital Off-line Simulators

A digital simulator emulates a physical component by solving the mathematical equations which describe the component behavior. Currently available digital simulation software programs can be classified into the following three classes.

1. Differential Equation Solver Programs
2. Circuit Oriented Programs
3. Hybrid Programs

1.2.3.1 Differential Equation Solver Programs

This category encompasses every mathematical software that can solve linear and non-linear differential equations. The most popular example is MATLAB and its graphical companion SIMULINK. For transient simulation in MATLAB, the user needs to describe the system in *state-variable form*. MATLAB is also a very convenient tool for controller design as it contains several toolboxes for control analysis, neural networks, fuzzy logic etc.

1.2.3.2 Circuit Oriented Programs

These software programs utilize *nodal method* of analysis for transient simulation. Popular programs in this category are SPICE, EMTP, EMTDC/PSCAD, and NETOMAC. SPICE was originally developed for the simulation of integrated circuits; its offspring PSPICE is now employed for the simulation of power electronics at low-power levels. The other three software packages are primarily used for power system and high-power electronics simulations. The user interface of each of these programs is unique.
EMTP and EMTDC are basically similar programs in terms of their internal workings with the exception of some modeling differences of certain power system components.

Both are fixed time step programs which make use of the trapezoidal numerical integration rule and the nodal method for network solution of power systems; a recently added feature of EMTDC/PSCAD is linear interpolation for some of its models. Both programs have extensive libraries of power equipment models, as well as analog and digital control models. In EMTP modeling of control systems can be achieved with TACS (Transient Analysis of Control Systems) or with newer MODELS package.
NETOMAC offers the user the choice of fixed step-size, variable step-size and interpolation features. It also provides a transient stability calculation feature absent in EMTP. There are some differences between NETOMAC and EMTP-type programs with regard to the modeling of control subsystem and the way it interacts with the network subsystem. A common feature of all these programs however, is that all of them are non-real-time off-line digital simulation tools.

1.2.3.3 Hybrid Programs

These programs utilize both nodal analysis and state-variable description. The nodal method is usually adopted for the network solution and the state variable approach is used to describe an auxiliary component, such as an SVC(Static Var Compensator) or a HVDC(High Voltage Direct Current) converter, which is eventually interfaced with the network using a Norton equivalent circuit. Some of these programs were developed as independent research efforts [2,3], however, the hybrid approach itself is extensively used in EMTP.

1.2.4 Real-time Digital Simulators

Rapid progress in computer technology and parallel processing is providing impetus for the development of fully digital real-time simulators which are posing increasing competition to existing TNAs and Hybrid simulators. The main examples in this category are:

- RTDS^{TM} from Manitoba HVDC Research Center [4]
- TEQSIM Simulator (A Hydro-Québec/Mitsubishi Electric Alliance) [5]
- dSPACE Simulator [6]

1.3 Literature Survey

The concept of real-time simulation using fully digital computers is quite a recent one; the first international conference on fully digital power system simulators [7] occurred in 1995. Since then there has been significant research effort to refine digital simulators with the goal of eventually replacing the analog and hybrid simulators with newer and lower costing fully digital simulators.

The flurry of research activity into real-time simulators was driven due to two main reasons. Firstly the advances in the development of discrete models for power system components (this aspect of research was pioneered by Dommel [8] with his EMTP program in the late 1960's, then as the popularity of EMTP grew detailed models of most of the power equipment were created). The improvements in accuracy and efficiency of these models had a significant effect on the real-time simulator technology. Secondly the enormous advancement in the computer technology, causing CPU and memory speeds to increase and the prices of desktop PCs, workstations and computer chips to drop every year, formed a major incentive that drove analog oriented thinking towards a digital mindset. Although concentrated on a short span of time, a so called "transient" in research activities has spawned a fairly extensive list of publications in the real-time simulation area but only those papers are cited here that were thought to serve as landmarks in this area. The literature can be broadly classified into three main categories : modeling, design and applications of real-time simulators.

1.3.1 Modeling for Real-time Simulators

The modeling aspect deals with the mathematical formulation of the power system components. Technically a model that has been developed for off-line simulation can be applied to real-time simulation as well but certain optimization of the model is needed with regard to its computational efficiency. The literature pertaining to this category can easily encompass models written for off-line simulation. The problem of interfacing a real-time digital simulator simulating power electronics with a digital controller has not been addressed in the literature, but problems related to modeling power electronic switches such as switching delay have been addressed with regard to off-line simulation. Variable step-size and order methods have been used for power electronic circuit simulation in programs such as SPICE but their application to power system transient simulation has been limited due to their large computation times compared to fixed step-size methods. Moreover, such methods have always been used for off-line simulation rather than real-time simulation.

Gole [2] developed a SVC model based on state variable approach to be interfaced with the EMTP program. To overcome spikes in the voltage waveform during thyristor switching, a variable time-step method is chosen for the SVC model where submultiples of the original fixed time step are used whenever switching occurred. In this method Adam's second order closed formula is used, which involves iterations instead of a single step solution using trapezoidal rule. Computation times reported for the SVC model are too large to be practical for real-time studies.

Kato [9] introduces a variable-order and variable step-size method using the Backward Differentiation Formulas for transient analysis. This method selects the step-size and the order which maximize the step-size within a given allowable error criterion. Results reported indicate better accuracy but higher CPU time compared to the fixed step-size trapezoidal rule.

Marti [10] reports a fully software approach for real-time transient simulation based on the EMTP. The problem of switching delay is not addressed in this paper. The focus is on achieving faster computational speeds for which the system admittance matrix is pre-calculated, inverted and stored unlike the EMTP where triangularization and Gauss reduction is used.

Crow [11] proposes multirate methods for the time domain simulation of systems containing fast components such as induction machine loads and FACTS controllers. Based on the behavior of the eigenvalues of the system the states are partitioned into different time response ranges and each variable is integrated with a step size which is sufficient for the required accuracy decided using a prediction of the local truncation error. The coupling between the different parts of the differential equations is maintained by approximating the slowly varying solution components.

Linear interpolation has been used effectively [12–14] in off-line simulation to estimate the switching instant. Its use in real-time simulation has not been reported yet. The effect of interpolation is demonstrated [14] in off-line simulation of HVDC systems for the removal of firing angle jitter and the elimination of spurious non-characteristic harmonics.

Acevedo [15] proposes a model to represent HVDC converters in real-time transient simulators. To meet the real-time requirements all the possible states of each 6 valve converter bridge are pre-calculated and stored. At each time step the correct state is picked up and solved together with the external power network for a simultaneous solution of the complete system.

1.3.2 Design of Real-time Simulators

The design aspect of real-time simulators deals with their hardware and software implementation. Several real-time simulators have been developed or are in the various stages of development. Some were developed to serve as large-scale, commercial, general-purpose simulators [16–18], while others were smaller-scaled, meant for the academic environment [19, 20] and tailored to meet certain system requirements [10, 20, 21].

Jakominich [18] reports the development of a hybrid simulator and its application in testing digital EHV-line relays. The developed simulator had the flexibility of modeling FACTS and HVDC converters fully digital by computer simulation and transient real-time data injection as well as analog by using physical simulation with original converter controllers for analyzing relays under real system conditions. Mercier [17] describes the evolution of the IREQ (Institut de Rechérche d' Hydro-Québec) simulator through a gradual transformation of its models from analog components to digital computers.

Some simulators were built around single processors [10, 21], but soon realization dawned that as the size of the systems modeled increased single CPUs were not able to cope with the resulting computational burden. Marti [10] demonstrates that using high-end, high speed computer workstations real-time operation can be achieved and sustained for a short period of time for relatively simple systems. The paper describes a completely software approach toward real-time simulation. Kezunovic [21] describes a design where a single processor is used for network simulation and multiple DSPs are used for instrument transformer and circuit breaker model implementation. The design has been optimized for power system protection device studies.

As the system complexity increased parallel processing was the next logical step in the evolution of real-time simulators. Durie [20] describes a digital microprocessor TNA based on the Inmos T800-20 transputer used for network modeling. One processor is assigned for each bus in the network being simulated. The topology of the *digital* TNA is identical to that of the network.

Recent advances in the digital signal processing technology has led to the development of RTDS^{TM} [16] a fully digital real-time simulator. The RTDS^{TM} is organized into individual racks of tightly coupled DSPs connected to one another using a common backplane. Each rack is identical and contains the necessary hardware for processing, inter-rack communication and user interface. The RTDS^{TM} uses PSCAD^{TM} for all user interaction. With its help the user can graphically enter the system under study, compile it and download the program to the RTDS^{TM}. The compiler produces all of the parallel processing code required by the DSPs and automatically assigns jobs to the individual DSPs.

1.3.3 Applications of Real-time Simulators

So far the two major application areas for real-time simulators have been:

1. Power system protection (relay testing) [18,21,22]
2. Testing of FACTS, Custom Power and HVDC controllers [19,23,24]

Other applications include simulation of transient stability [25], on-line security assessment [26], dispatcher training [27], etc.

1.4 Thesis Requirements

From a detailed inspection of the currently available digital simulation tools and a survey of the current literature it was concluded that :

1. Despite the fact that real-time digital simulators have been employed for various purposes for some time, their interaction with digital controllers has not been studied in sufficient detail to address the problems therein. Therefore, a thorough study related specifically to a real-time digital simulator simulating switching power circuits was needed. Although the problem of switching delay in digital simulation

has been addressed in the context of off-line simulation, its effects in real-time simulation have not been investigated, which rendered most of the off-line solutions inapplicable for real-time simulation.

2. Due to several modeling and implementation constraints off-line programs such as EMTP were unsuitable for simulating conditions encountered in the practical implementation of digital controllers. This hurdle demanded the development of new software, both off-line and real-time, addressing the specific modeling and implementation needs of the problem at hand.

3. The efficacy of any proposed solution needed to be tested by hardware implementation on a real-time digital processing platform. The goal here was not to develop a general purpose real-time digital simulator but rather to tailor the implementation to meet specific design and simulation requirements. None of the hardware implementations reported in the literature addressed the issues sought after in this thesis.

4. Simulation, be it off-line or real-time, is governed by the same underlying mathematical processes. Validation remains incomplete if a simulation cannot be verified by experimentation. An experimental setup of the study system was therefore needed to corroborate the simulation results and to provide closure.

1.5 Outline of Thesis

This thesis is composed of five chapters. Chapter 1 provided a general introduction to the area of real-time simulation. A review of literature is presented to bring the reader upto date with current research in this area.

Chapter 2 starts with the problem pertaining to switching events in digital simulation. Techniques to handle discrete events in off-line simulation and the reasons for their practi-

and a digital controller is investigated to get an insight into the the causes of simulation errors. The details of the proposed real-time simulation method are presented. The chapter ends with a discussion on numerical methods suitable for real-time simulation.

Chapter 3 is devoted to the modeling and control design of a three-phase PWM VSI system which is used as a study system to demonstrate the simulation results predicted in Chapter 2. Detailed off-line simulation technique is presented highlighting some key points where the adopted approach deviated from the convention. The simulation results also study the impact of limited time steps frequently encountered in real-time simulation using the fixed time step approach. A comparison of the results obtained using the fixed time-step approach and the proposed approach is presented to illustrate the benefits obtained from using the proposed approach.

Chapter 4 is devoted entirely to the experimental work done to validate the proposed approach. Firstly, the design and implementation of a real-time digital simulator based on a DSP-FPGA platform is presented. Then a description of an actual power circuit implementation of the VSI system is presented. Results reported in this chapter compare those obtained from an off-line simulation using fixed step-size approach with a small time-step, the real-time simulation using the proposed approach and those of the physical model.

Chapter 5 presents the conclusions and the main contributions of this thesis. Suggestions for future research work are also presented.

Chapter 2

Switching Events in Digital Simulation

This chapter addresses the problem related to switching events in digital simulation. Section 2.1 will define the problem. The conventional techniques to handle this problem in off-line simulation will be presented in section 2.2. All off-line strategies will be examined in detail and their particular advantages are discussed. Section 2.3 examines the reasons for the unfeasibility of some of these techniques for real-time simulation. Next, the interaction of a real-time digital simulator and a digital controller is studied, in detail in Section 2.4, to get an insight into the causes of simulation errors when proper synchronization is not used. Sampling theory will be used to model this interaction. Section 2.5 presents the details of a new method for synchronized operation of a real-time digital simulator and a digital controller. The proposed method relies on registration of the timing of the discrete gating pulses and uses a correction algorithm to modify the normal operation of the simulator. Selection of a proper numerical method is paramount in real-time simulation due to its influence over the speed, accuracy and stability of the simulation. Therefore the chapter will conclude with a discussion on the suitability of different numerical methods for real-time simulation.

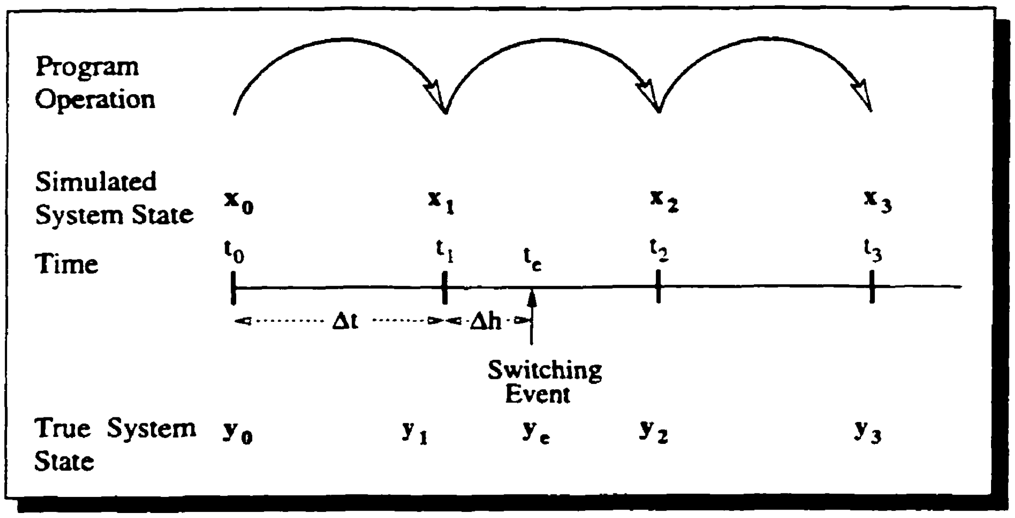


Figure 2.1: Switching event in digital simulation

2.1 The Problem

Digital controllers for power electronic systems output discrete firing signals which may not necessarily be in synchronism with the time step chosen for the real-time digital simulator. The error in the solution of the system state stems from the delay introduced in the switching due to improper synchronization of the two discrete processes. Unlike the actual physical system the digital simulator is unable to respond instantaneously to a firing signal that comes in between two calculation steps. Since the simulator can only respond at the end of a calculation step the actual switching may occur one time step too late in the worst case. Also, since the incoming firing signal does not always occur at the same instant on the time grid of the simulator, the switching is not always realized at the same moment and therefore the delay introduced is not constant. The larger the time-step of the simulator the larger is the delay in switching.

Figure 2.1 illustrates the problem on a time grid progressing from left to right $t_0 < t_1 < t_2 \ldots$ with a time step of $\Delta t = t_{i+1} - t_i$, $(i = 0, 1, 2, \ldots)$. A fixed step-size numerical integration program operation is depicted by the curved arrows. Let x_i, $(i = 0, 1, 2, \ldots)$ be the states of the system computed by the simulator at every time step Δt and y_i, $(i =$

$0, 1, 2, \ldots$) be the true states of the system obtained by taking the discrete events into account at their exact locations. The firing signal comes in at time t_e but is accounted for at time t_2 when the real-time simulator has already calculated the incorrect state x_2. The change in state requested by the discrete event is acknowledged at time t_2 and is used to calculate state x_3 at time t_3. The actual physical system would respond to the firing signal at time t_e with the state y_e and y_2 would be the true state of the system at time t_2.

2.2 Techniques for Handling Switching Events in Digital Simulation

Switching delay was first encountered in off-line simulation while simulating thyristor based power electronic circuits using line commutation. Since a thyristor turns off when the current through it goes to zero, switching is delayed by a fraction of one time step if the current zero occurs between two calculation steps. The techniques developed to solve this problem were later used for circuits using forced commutation and for circuits employing switches with gate turn-off capability. Although the current in a circuit equipped with switches such as GTOs or IGBTs can be interrupted at any time, switching delay in the simulation arises if the firing signal for the switch comes in between two time steps of the simulator. Figure 2.2 illustrates the algorithms that have been used to account for discrete switching events in off-line digital simulation. In a fixed time-step digital simulation program, such as the EMTP-MODELS, events that occur between two time steps are accounted in the next calculation step. The switching event coming in at time t_e is acknowledged at time t_2 and is used to calculate state x_3 at time t_3 . The traditional approach (Figure 2.2(A)) to alleviate the errors due to delay in switching is to carry out the entire simulation with a small step-size $\delta t = \Delta t/n$, (n integer), so as to reduce the

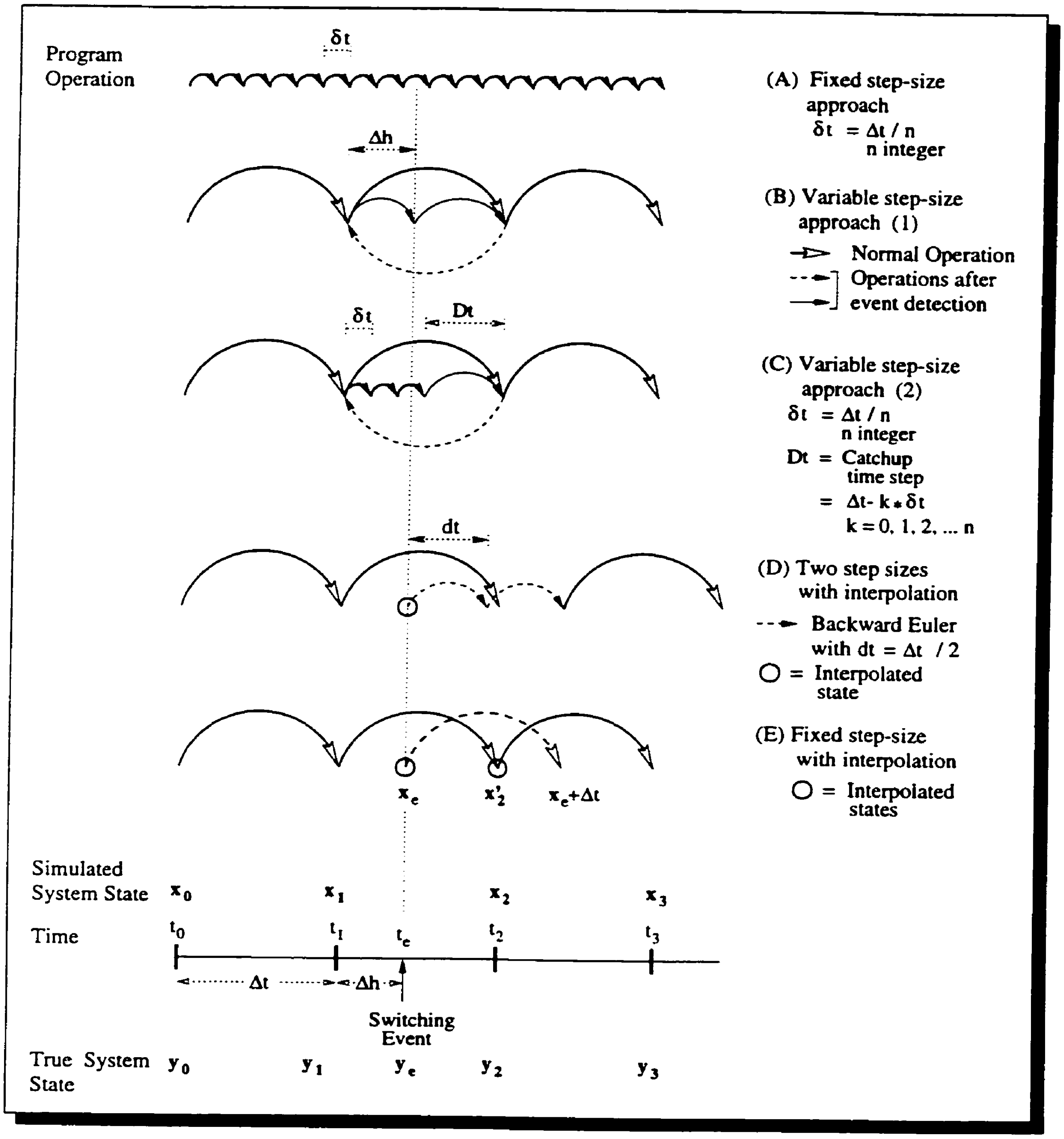


Figure 2.2: Techniques for handling switching events in off-line digital simulation

delay, but at the cost of a larger total simulation time.

In a variable time-step program (Figure 2.2(B)), as the name suggests, the time-step of the simulation is changed whenever a switching event comes in between two calculation steps. When the switching event is detected at time t_2 the algorithm backtracks to the state x_1 at the previous time step, takes a smaller time step Δh and calculates state y_e, takes another time step $(\Delta t - \Delta h)$ and calculates state y_2, and then proceeds with a fixed time step Δt till the next switching event is detected. The variable time-step process would involve a re-formulating of the admittance matrix during the course of program execution everytime the time-step changes. If the study system is large, this would mean a significant computational burden.

Another variation (Figure 2.2(C)) of the variable time-step method is a method where two time steps- one Δt and another $\delta t\,(= \Delta t/n)$ an integer submultiple of Δt are maintained. When the discrete event is detected at time t_2 the algorithm backtracks to time t_1 and starts calculating states every δt so that instead of accurately pinpointing the instant t_e it is finely straddled between two calculation steps δt. Once the event has been accounted for a catch-up time step $Dt\,(= \Delta t - k * \delta t,\ k = 0, 1, 2, \ldots n)$ is taken to time t_2. The advantage of this method is that the system admittance matrix can be pre-calculated and stored for the three time steps Δt, δt and Dt. This method was used for simulating an SVC (Static Var Compensator) circuit [2] using state variable formulation. Adam's second order integration rule was used to iterate around the event until adequate accuracy was achieved.

Linear interpolation has been used effectively in off-line digital simulation programs to accurately model switching instants. In the method of Figure 2.2(D) trapezoidal integration is used for normal operation with time step Δt. When a switching event is detected, the method interpolates to obtain circuit variables at time t_e. Then, Backward Euler (BE) integration is used for the subsequent two half time $(\Delta t/2)$ steps after which normal operation resumes. Notice that the original time grid has now been altered. The main motivation behind this approach is that using half time step BE rule the resulting system matrix is the same as that needed for a full time step trapezoidal rule. Any other step-size would entail a matrix reformulation. This method [12] has been implemented in the Microtran version of EMTP.

Figure 2.2(E) illustrates another method [14] which uses linear interpolation but the step-size is now fixed. Once the switching event is detected at time t_2 states x_1 and x'_2 are linearly interpolated to obtain state x_e at time t_e and the solution continues with the original time step $\triangle t$ yielding a new solution $x_{t_e+\triangle t}$ one time-step later. An additional interpolation step between t_e and $(t_e + \triangle t)$ is taken to find the correct state at t_2 i.e., x_e and $x_{t_e+\triangle t}$ are now linearly interpolated to get x_2. This second interpolation step is taken to put the solution back on the original time grid. So, there are two interpolation steps and two regular solution steps from the time the switching event is detected at t_2 till the time the state x_3 at t_3 is calculated. This approach again has the advantage that the admittance matrix need not be re-formulated since the time step is fixed.

2.2.1 Why Above Techniques Are Not Suitable for Real-Time Simulation

With regard to real-time computation the fixed step-size approach (Figure 2.2(A)) with a small $\triangle t$ is not feasible in the case of realistic size systems due to excessive computational speed requirement. The variable step-size approaches (1) and (2) in (Figure 2.2(B),(C)) also cannot be used because real-time simulation normally does not permit stepping back in time or iterations around the event. The main disadvantage of the method of Figure 2.2(D) is that it alters the original time grid every time a switching event is detected and thus becomes impractical for real-time operation. It is otherwise quite efficient to use two half time steps of BE integration which eliminates reformulation of the admittance

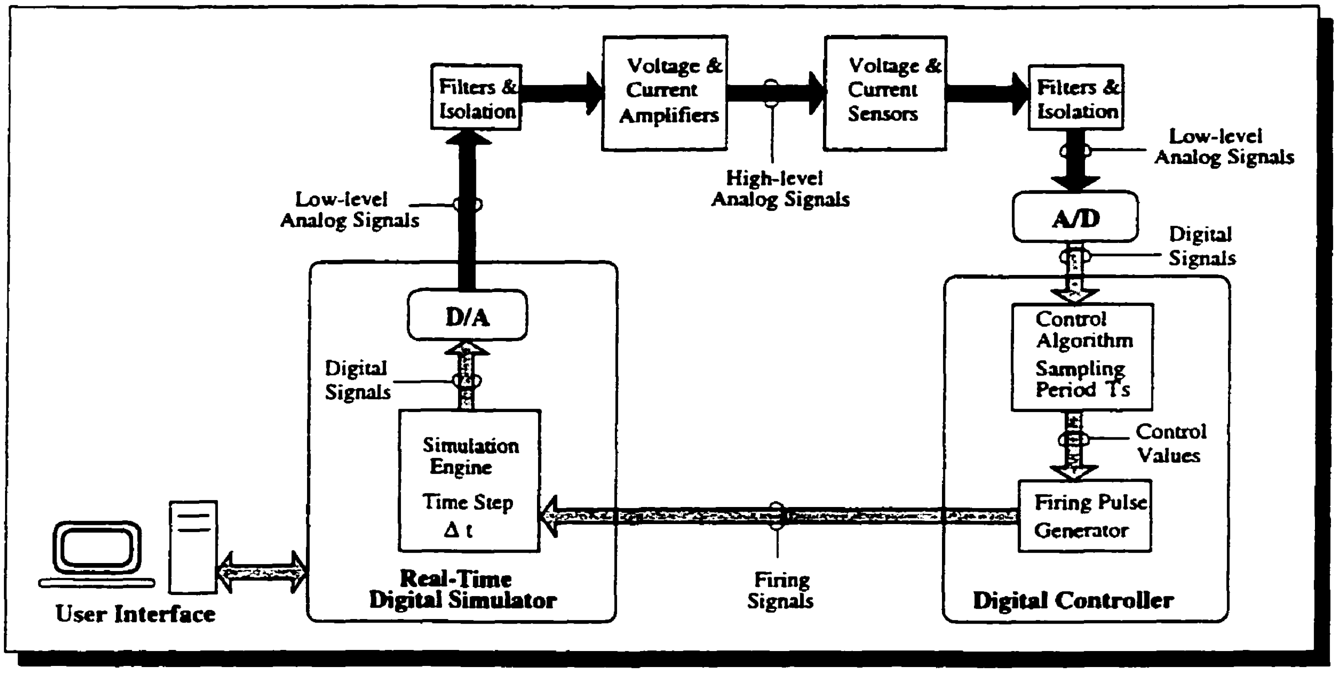


Figure 2.3: Generic setup of a real-time digital simulator interfaced with a digital controller

matrix. The approach of Figure 2.2(E) is conceptually attractive due to the fixed step-size but for real-time implementation it still needs significant computer time to compute two interpolation steps and two regular solution steps.

2.3 Interaction of a Real-Time Digital Simulator with a Digital Controller

Figure 2.3 illustrates a generalized setup of a real-time digital simulator interacting with a digital controller. The simulator generates digital signals internally which after D/A conversion are output as low-level analog signals. The low-level analog signals after adequate isolation and filtering are amplified before being applied to a test equipment or a controller. The digital controller setup usually has sensoring equipment and A/D converters as external peripherals to convert the high-level analog signals to low-level

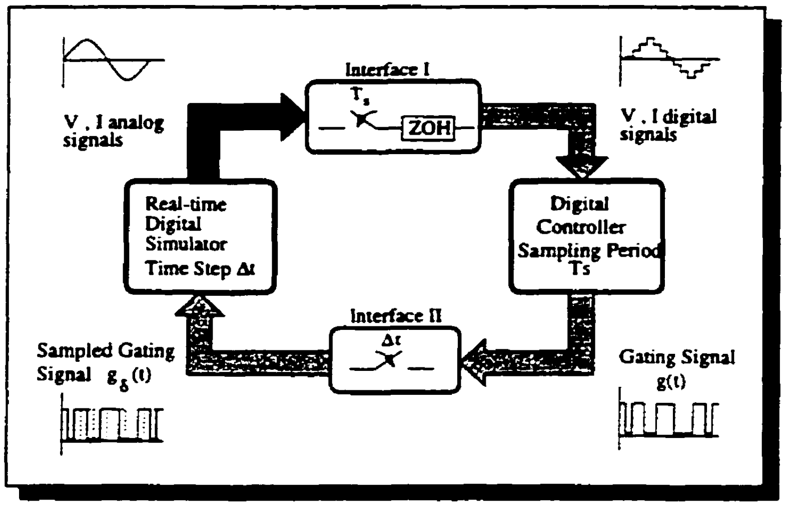

V, I analog signals
Interface I
T_s
ZOH
V, I digital signals
Real-time Digital Simulator Time Step Δt
Digital Controller Sampling Period Ts
Interface II
Δt
Sampled Gating Signal $g_\delta(t)$
Gating Signal g(t)

Figure 2.4: Interfaces between a real-time digital simulator and a digital controller

digital signals which are then fed to the control algorithm. A firing pulse generator inside the digital controller generates discrete gating pulses which are fed back to the simulator to control power electronic switching devices in the simulation.

Figure 2.4 illustrates a model which includes the essential elements taken from Figure 2.3. In deriving this model the quantization errors of A/D conversion are ignored and all amplifiers and sensors are assumed to be ideal without any bandwidth limitations. Two interfaces I and II are shown between the real-time digital simulator and the digital controller. Interface I represents the sampling action performed by the A/D converter on the power signals v, i and is modeled by a sampler which operates at a specific rate T_s (called the controller *sampling period*) and a zero-order hold. The simulator can acquire the gating signal from the digital controller only at finite intervals Δt due to the discrete nature of the numerical integration process it is carrying out. Interface II represents this Δt latency on the gating signal and is modeled by another sampler with sampling rate Δt. The problem defined in Section 2.1 is related to Interface II.

Let $g(t)$ represent the gating signal as the output of the digital controller and $g_\delta(t)$ represent the sampled gating signal as the input to the simulator. The action of sampling is mathematically represented by using *Impulse Modulation* [28]. $g_\delta(t)$ is obtained by multiplying $g(t)$ by a unit impulse train $\delta_T(t)$ with a period $T = \Delta t$.

$$g_\delta(t) = g(t) \cdot \delta_T(t) = g(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{2.1}$$

Since the impulse train is a periodic signal it can be expressed as an exponential Fourier series

$$\delta_T(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) = \sum_{n=-\infty}^{\infty} F_n \, e^{jn\omega_{\Delta t} t}, \quad \omega_{\Delta t} = 2\pi/T \tag{2.2}$$

where

$$F_n = \frac{1}{T} \int_{-T/2}^{T/2} \delta(t) \, e^{-jn\omega_{\Delta t} t} \, dt = \frac{1}{T} \tag{2.3}$$

so that

$$\delta_T(t) = \frac{1}{T} \sum_{n=-\infty}^{\infty} e^{jn\omega_{\Delta t} t}, \quad \omega_{\Delta t} = 2\pi/T \tag{2.4}$$

Substitution of $\delta_T(t)$ in (2.1) yields

$$g_\delta(t) = \frac{1}{T} g(t) \sum_{n=-\infty}^{\infty} e^{jn\omega_{\Delta t} t} \tag{2.5}$$

$$= \frac{1}{T} \sum_{n=-\infty}^{\infty} g(t) \, e^{jn\omega_{\Delta t} t} \tag{2.6}$$

The Fourier transform of both sides of (2.6) yields

$$g_\delta(j\omega) = \frac{1}{T} \sum_{n=-\infty}^{\infty} g[j(\omega - n\omega_{\Delta t})] \tag{2.7}$$

The right hand side of (2.7) represents the spectrum $g(j\omega)/T$ repeating periodically every $\omega_{\Delta t} = 2\pi/T = 2\pi/\Delta t$ rad/sec. As $T = \Delta t$ increases the distance between the individual $g(j\omega)/T$ spectrums decreases and they overlap.

Consider a rectangular pulse and its Fourier spectrum (Figure 2.5(a),(e)). Since the pulse is time limited its frequency spectrum is band unlimited. As the width of the pulse

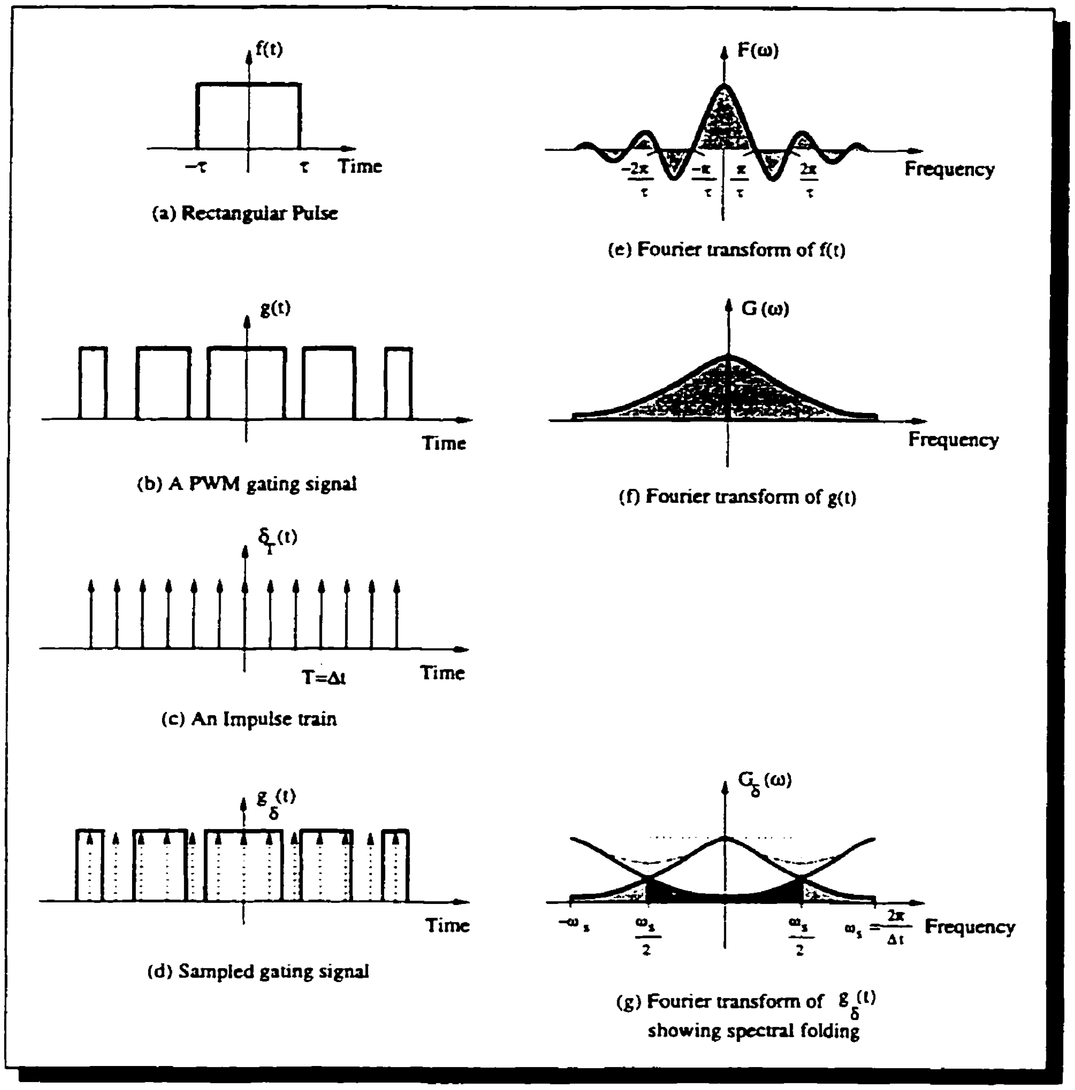


Figure 2.5: Sampling action on a gating signal

decreases, the amplitude of its spectrum decreases but its bandwidth increases.

A pulse width modulated gating signal $g(t)$ can be considered as a series of rectangular pulses with changing pulse widths. Its frequency spectrum shown in Figure 2.5(f) is also band unlimited not unlike that of a rectangular pulse. Sampling such a signal at a fixed rate will result in an *aliased* signal according to Shannon's sampling theorem [28]. The spectrum of $g_\delta(\omega)$ would consist of overlapping cycles of $g(\omega)$ repeating every $\omega_{\Delta t}$ rad/sec (Figure 2.5(g)). Aliasing arises due to the effect of spectral folding where a certain band of high frequency components get reflected in the low frequency region due to the limited sampling rate Δt. The resulting gating signal $g_\delta(t)$ thus is always an undersampled version of $g(t)$ and contains pulses having different positions and widths.

Now, it can be assumed that the spectrum of $g(t)$ decays significantly at high frequencies and contributes little to the reconstruction of the signal $g_\delta(t)$. Based on this assumption $g(t)$ can be bandlimited at a certain sufficiently large frequency $\omega_B = 2\pi/T_B$ and sampled at a rate in excess of $2\omega_B$. This is equivalent to using a sufficiently small time step $\Delta t < 1/2f_B$ to sample such a signal.

To illustrate the effects of limited sampling rates on the gating signal consider a PWM gating signal generated using a fundamental frequency ($60Hz$) sine wave and a triangular carrier at $1kHz$ using a time step of $10\mu s$. Figure 2.6 shows the changing nature of the gating signal when sampled at three different sample rates. For $\Delta t = 50\mu s$ and $\Delta t = 100\mu s$ the pulses are slightly shifted from their original positions and are slightly wider or narrower than their original widths. These rather insignificant differences have a dramatically significant impact when the altered gating signal is used to control a converter in a power circuit. As seen from Figure 2.6 at $\Delta t = 200\mu s$ entire negative pulses are missed. The above description is valid for any kind of gating signal but a PWM signal is chosen in particular because it is a ubiquitous signal for power electronics applications. The effect of the aliased gating signal can be studied by examining the

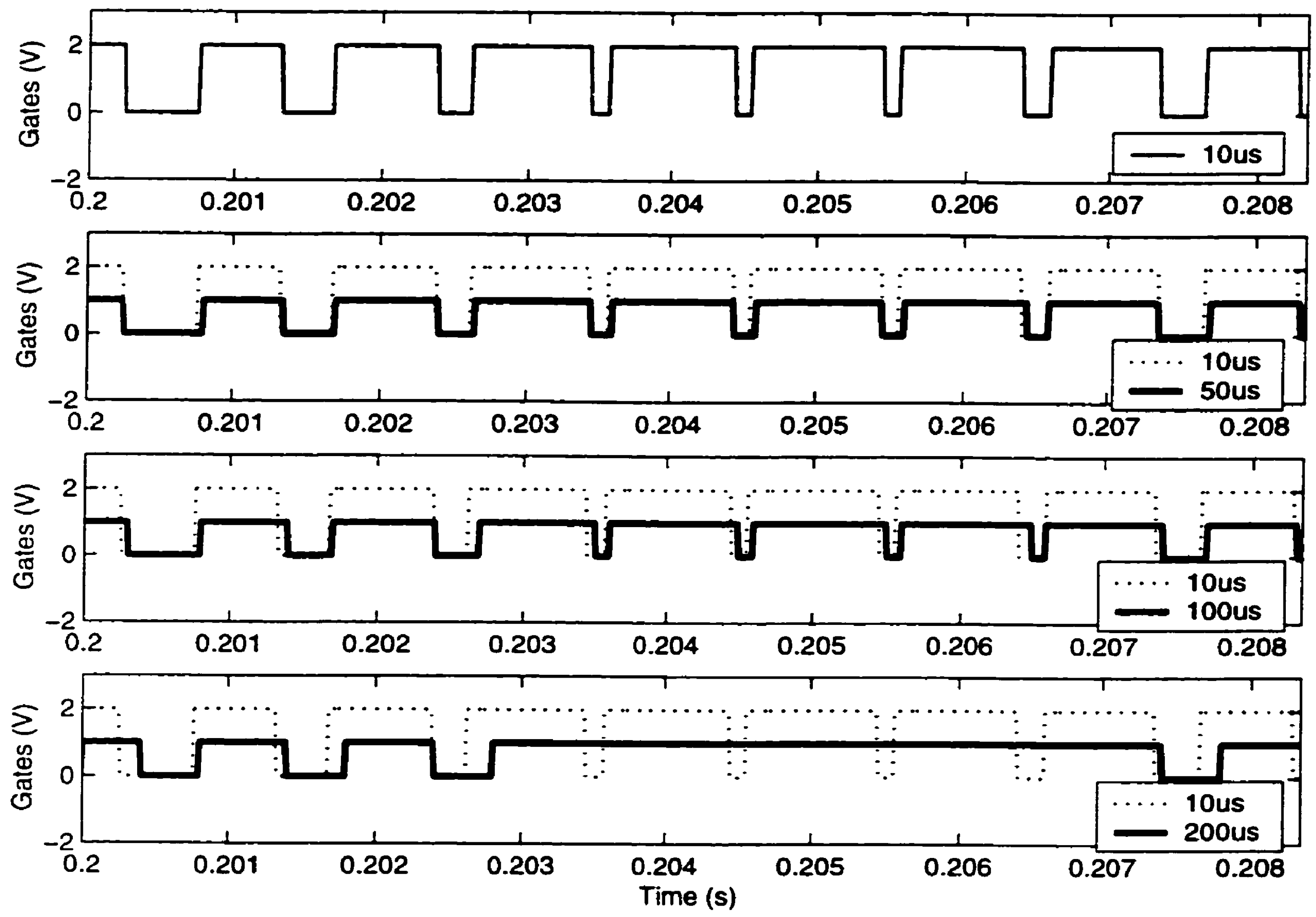


Figure 2.6: Effect of limited rate sampling on the gating signal

harmonic spectrum of the circuit variables in a power electronic circuit. As will be revealed from the simulation of a PWM VSI system in Chapter 3 an aliased gating signal can give rise to severely inaccurate results.

2.4 Proposed Approach for Real-Time Simulation

Based on the discussion in Section 2.2, clearly the existing techniques to handle switching events in digital simulation adopt one of the following practices that contradict real-time operation :

- Stepping back in time and changing states that have already been calculated.
- Altering the original time grid of the simulation.
- Iterations around the switching event.
- Varying the step size.

Of all the algorithms outlined in Section 2.2, the desirable approach to account for inter-step switching events is to use a small simulation time step. However, the main hurdle for its practical implementation is computer resource. Therefore, this approach is not feasible for real-time computation. It was concluded that a new real-time simulation algorithm was needed which conformed to the following requirements :

- A fixed simulation time step.
- No alteration of the original time grid during simulation.
- Registration of the timing of the external switching events independent of the simulation process.
- A computationally efficient way to utilize the registered timing to *correct* the simulated state.

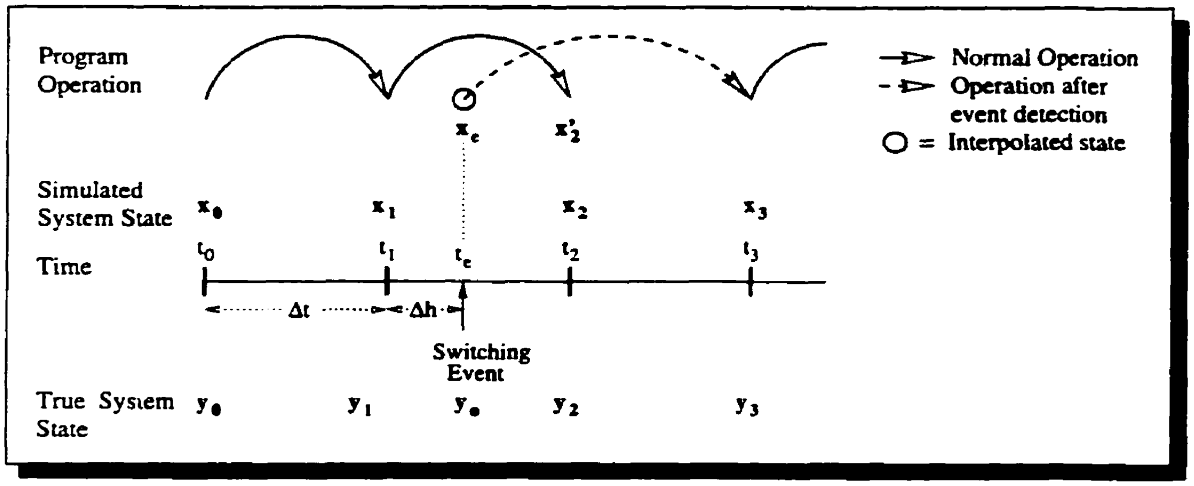


Figure 2.7: FICS algorithm for accounting switching events in real-time simulation

The proposed approach, shown in Figure 2.7, depends on registration of the timing of the discrete switching event and a subsequent correction algorithm before the next state calculation. This approach shall be referred to as the FICS (Fixed step with Interpolation and Clock Synchronization) algorithm henceforth.

At time t_2 the simulator has computed and sent out the state x_2 which is incorrect because a switching event arrived at time t_e while the simulator was still engaged in the calculation of x_2. Real-time operation does not allow recalling and changing state x_2. However, if the simulator were provided with the timing of the event relative to the simulation time grid, it can take certain *corrective* action before proceeding to calculate state x_3 at time t_3. An obvious question at this point is why doesn't the simulator take preemptive action before calculating x_2. The answer is that the simulator does not know *a priori* when the switching event will come in, the event being controlled by processes external to the simulator.

At the beginning of every time step the simulator looks for the switching event and its timing information. In the presence or absence of this knowledge one of the following operational paths are undertaken :

1. Normal Operation (when no event is detected).

2. Post-event Operation (when an event is detected).

2.4.1 Normal Operation

The system under study can be divided into two subsystems: one containing the Power electronic (PE) module and the other containing the network.

2.4.1.1 Power Electronic module

The PE part is comprised of switches whose state is governed be external firing pulses. Accordingly the output variable v of the PE module can be expressed as

$$v = f(G, x) \tag{2.8}$$

where G is the gating signal and x is a network state.

2.4.1.2 Network Solution

The network solution is similar to that adopted by a fixed step-size program such as the EMTP. A numerical integration technique is used to convert continuous time differential equations into discrete-time difference equations. This allows recursive solution of the state equations. All electrical elements are represented as equivalent admittances and current injections. Nodal analysis is then used to formulate a set of linear algebraic equations. The transient solution proceeds by solving the linear equations for the unknown node voltages at each time step. A simple first-order scalar formulation of the method is given below.

The state equation for a linear combination of lumped elements such as R, L or C can be written in general as

$$\dot{x} = a\,x + b\,u \tag{2.9}$$

subject to the initial conditions $x(t_0) = x_0$ and $u(t_0) = u_0$, where x is the circuit state and u is the input. The parameters a and b are dependent on the circuit constants R, L and C. With a time step of integration Δt, the solution at time $(t + \Delta t)$ can be expressed in terms of the solution at time t :

$$x(t + \Delta t) = x(t) + \int_{t}^{t+\Delta t} [a\,x(\varsigma) + b\,u(\varsigma)]\; d\varsigma \tag{2.10}$$

where ς is a variable of integration. Now, a numerical integration algorithm can be used to approximate the value of the integral in (2.10). Let the subscript n denote quantities at time t and $(n+1)$ quantities at time $(t + \Delta t)$.

$$x_{n+1} = x_n + f\,(\Delta t, a, b, x_{n+1}, x_n, \ldots, u_{n+1}, u_n, \ldots) \tag{2.11}$$

(2.11) forms the discrete time solution of (2.9). The number of history terms of x and u in the integral approximation will depend on the type of the chosen numerical method. A widely used numerical integration method is the Trapezoidal Rule employed in such programs as SPICE and EMTP. It uses a linear approximation of x over the interval t to $(t + \Delta t)$. Applying the Trapezoidal Rule to (2.10) we get

$$x_{n+1} = x_n + a\frac{\Delta t}{2}\,(x_{n+1} + x_n) + b\frac{\Delta t}{2}\,(u_{n+1} + u_n) \tag{2.12}$$

(2.12) can be re-arranged as

$$x_{n+1} = \alpha\, x_n + \beta\,(u_n + u_{n+1}) \tag{2.13}$$

where

$$\alpha = \left(1 + a\frac{\Delta t}{2}\right) / \left(1 - a\frac{\Delta t}{2}\right) = f_\alpha(\Delta t) \tag{2.14}$$

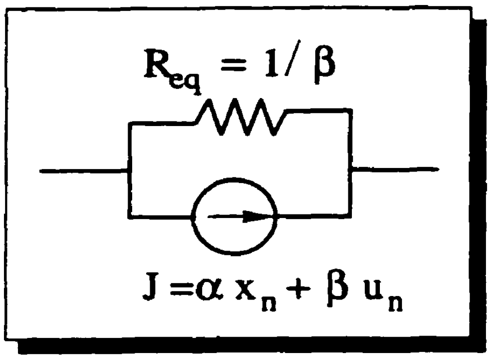


Figure 2.8: Discrete equivalent circuit of a lumped element

and

$$\beta = b\frac{\Delta t}{2} / \left(1 - a\frac{\Delta t}{2}\right) = f_\beta(\Delta t) \tag{2.15}$$

Recursive solution of (2.13) starting with $n = 0$ will yield the desired approximate solution of (2.9).

Network interpretation of (2.13) (2.13) can be re-written as

$$x_{n+1} = \beta u_{n+1} + J \tag{2.16}$$

where

$$J = \alpha x_n + \beta u_n \tag{2.17}$$

If the state x is a current and input u is a voltage source, then (2.16) represents an equivalent circuit (Fig 2.8) consisting of a current source J in parallel with a conductance β. J represents the *history* of the circuit since it depends on the state and input at the previous time step. α and β can be interpreted as the discrete time equivalent circuit parameters dependent not only on the original circuit parameters a and b but also on the time step Δt. Therefore, a change in the time step would directly affect the behavior of the circuit being simulated. The values of α and β for several commonly occurring lumped elements are given in Table 2.1.

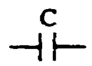
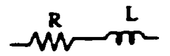
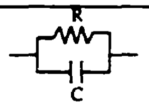
Element				
α	1	1	$\left(1+\frac{R}{L}\frac{\Delta t}{2}\right)/\left(1-\frac{R}{L}\frac{\Delta t}{2}\right)$	$\left(1+\frac{\Delta t}{2RC}\right)/\left(1-\frac{\Delta t}{2RC}\right)$
β	$\frac{\Delta t}{2L}$	$\frac{2C}{\Delta t}$	$\frac{\Delta t}{2L}/\left(1-\frac{R}{L}\frac{\Delta t}{2}\right)$	$\frac{\Delta t}{2C}/\left(1-\frac{\Delta t}{2RC}\right)$

Table 2.1: Parameters of discrete equivalent circuits for lumped elements

If the network has several branches of R,L and C, equations such as (2.16) can be written for every branch in the network. Nodal analysis can then be used to determine the node voltages. For a general network with n nodes, a system of n linear algebraic equations can be formed

$$[G]\cdot[v(t)] = [i_s(t)] - [J] = [h(t)] \tag{2.18}$$

where $[G]$ is the nodal conductance matrix, $[v(t)]$ is the column vector of the n unknown node voltages, $[i_s(t)]$ is the vector of current sources and $[J]$ is the vector of history current injections. In the normal execution of the EMTP, (2.18) is solved for the node voltages using the Gaussian Elimination technique. For the real-time simulator, however, a different approach has been adopted. A direct matrix inverse of $[G]$ is employed for the solution of (2.18).

$$[v(t)] = [G]^{-1}\cdot[h(t)] \tag{2.19}$$

In the real-time solution loop (2.19) is solved repeatedly for $[v(t)]$ at every time step. The matrix $[G]^{-1}$ is pre-calculated, stored and applied at every time step. Thus the normal operation can be summarized as :

1. Update the power electronic model using (2.8).

2. Calculate the history terms (2.17) and the voltage sources.

3. Calculate the state at the next time step using (2.16).

2.4.2 Post-Event Operation

At time t_2 the simulator possesses the following information:

- States x_2 and x_1
- Knowledge that the switching event occurred at time t_e relative to t_1. This information has two parts: one is a change in switch status and the other is the time interval Δh.

Next the simulator performs the following operations:

1. Linearly interpolates the network variables (all states and inputs) at time t_e based on x_1, x_2 and Δh. It is important that the variables interpolated comprise the state variables since they completely describe the system dynamic state at any time t.

$$\begin{aligned} x_i &= x_n + \tfrac{\Delta h}{\Delta t}\left(x_{n+1} - x_n\right) \\ u_i &= u_n + \tfrac{\Delta h}{\Delta t}\left(u_{n+1} - u_n\right) \end{aligned} \tag{2.20}$$

2. Computes the coefficients $\alpha_i = f_\alpha(2\Delta t - \Delta h)$ and $\beta_i = f_\beta(2\Delta t - \Delta h)$ based on the time step $(2\Delta t - \Delta h)$ instead of Δt.

3. Updates the power electronic model (2.8) based on the new gating information,. This state of the PE model remains unchanged till the next switching event is detected.

4. Calculates the history terms associated with the discrete time representation of the network components. During this calculation the results from steps 1 and 3 are used to properly initiate the history terms at time t_e. This operation incorporates the new changes made in the PE model and the network part acquired through linear interpolation. Normally, in the fixed step-size method, the history terms for

the calculation of state at time t_3 would be garnered from the incorrect state x_2 at time t_2.

5. Calculates the state of the system (2.16) at time t_3 based on the information from Step 2.

It is clear that calculation Steps 3, 4 and 5 are the same as the Normal Operation. The extra computation effort goes into Steps 1 and 2 after a switching event is detected. The *correction* indeed comprises of Steps 1 and 2 whose results are used in Steps 4 and 5 to properly initiate the history terms and to calculate the next state. Linear interpolation provides a computationally efficient way to estimate the circuit state without going through the full calculation procedure. Notice that during the *correction* procedure the original time grid is not altered. From the perspective of an external observer the apparent time step of the simulator is still fixed at Δt, there is only an internal adjustment of the time step to re-synchronize with the real-time clock whenever a switching event is detected. For the practical implementation of the FICS algorithm, however, there should be a provision to presicely capture the switching event and relay its timing information to the simulator at the beginning of the next time step.

2.5 Choice of a Numerical method for Real-time Simulation

Numerically, electromagnetic transient simulation is an Initial Value Problem (IVP). A numerical integration method provides an approximate solution to the IVP. The objective of this section is to find out which numerical methods are suitable for real-time simulation. Although Trapezoidal rule is popular and is widely used in EMTP-type programs, is it really a good choice for real-time simulation and if so why? This section seeks an answer to this question.

Class	Name	Attributes	Suitability
One-step Methods	Forward Euler	Low Order, Explicit, Slow convergence	No
	Runge-Kutta	Explicit, Accurate, Orders>2 computer intensive, Employ variable step-size	No
Multi-step Methods	Adams-Bashforth (A-B)	Explicit, Starting problem for orders>2	No
	Adams-Moulton (A-M)	Implicit, Good stability, Starting problem for orders>2	
	Backward Euler	A-M order 1	Yes
	Trapezoidal Rule	A-M order 2	Yes
	Gear's	Implicit,Good stability,Starting problem for orders>2	Yes
Predictor Corrector Methods	Combination of A-B and A-M	Variable-order, Variable step-size, Computer intensive	No
	Gear's Backward Difference Formula (BDF)		

Table 2.2: Suitability of numerical methods for real-time simulation

A general discussion on the various numerical methods for IVPs can be found in [29]. The three main criteria for selecting an integration algorithm are accuracy, speed, and stability. These criteria often contradict each other in terms of step-size and the order of the method selected which makes the selection process complicated. However, it is possible to eliminate many of the methods by careful examination of their properties. Table 2.2 summarizes the attributes of the available numerical methods. The choice of a good method was governed by the following guidelines :

1. Multi-step methods that use information at several previous time steps $t_{n-1}, t_{n-2}, \ldots$ are preferred over single step methods that use information only at the previous time instant t_{n-1} to calculate the state at t_n.

2. Multistep methods of orders greater than 2 have a starting problem due to the lack of enough initial conditions.

3. Methods having orders higher than 2 offer better accuracy but have shrinking stability regions and involve greater computation. Thus if computational efficiency is a requirement, as in real-time simulation, the orders have to be restricted to 2.

4. Implicit methods such as Adams-Moulton methods are preferred over explicit methods such as Adams-Bashforth methods due to their superior stability properties.

5. Computationally straight forward recursive schemes are better than schemes that involve iterations such as the Runge-Kutta fourth order method [29] using Merson's error criterion to vary step-size. In RKM-4 the function f is evaluated four times at each time step and these values are not used for any subsequent calculations.

6. For similar reasons as in (5) Predictor-Corrector schemes involving iterations cannot be employed for real-time simulation.

Based on the results presented in Table 2.2 and the above discussion the numerical integration methods that stand out as good candidates for real-time simulation are: Backward Euler's (BE) method, Trapezoidal rule and Gear's 2^{nd} order method. The following section examines these three methods in more detail from the perspective of accuracy, efficiency and stability.

2.5.1 Comparison of Backward Euler, Trapezoidal and Gear's 2^{nd} order methods

Consider the differential equation for an inductor

$$v(t) = L\frac{di(t)}{dt} \tag{2.21}$$

The corresponding discrete-time difference equation can be written as:

$$i(t) - i(t - \Delta t) = \frac{\Delta t}{L} v(t) \tag{2.22}$$

for the Backward Euler method,

$$i(t) - i(t - \Delta t) = \frac{\Delta t}{2L} \left(v(t) + v(t - \Delta t)\right) \tag{2.23}$$

for the Trapezoidal rule and

$$i(t) - \frac{4}{3} i(t - \Delta t) + \frac{1}{3} i(t - 2\Delta t) = \frac{2}{3} \frac{\Delta t}{L} v(t) \tag{2.24}$$

for the Gear's 2^{nd} order method. Applying Z-transform's property

$$Z\left\{f\left(t - k\Delta t\right)\right\} = z^{-k} Z\left\{f\left(t\right)\right\} \tag{2.25}$$

to (2.23)

$$I(z) - z^{-1} I(z) = \frac{\Delta t}{2L} V(z) + \frac{\Delta t}{2L} z^{-1} V(z) \tag{2.26}$$

from which the transfer function $H(z)$ of the discrete-time system can be derived as

$$H_{trap}(z) = \frac{I(z)}{V(z)} = \frac{\Delta t}{2L} \frac{z + 1}{z - 1} \tag{2.27}$$

The response to a sinusoidal excitation $v(t) = e^{j\omega t}$ is given by

$$i(t) = H_{trap}(z) e^{j\omega t} \tag{2.28}$$

where $z = e^{j\omega \Delta t}$. The corresponding transfer function for the Backward Euler method is

$$H_{be}(z) = \frac{\Delta t}{L} \frac{z}{z - 1} \tag{2.29}$$

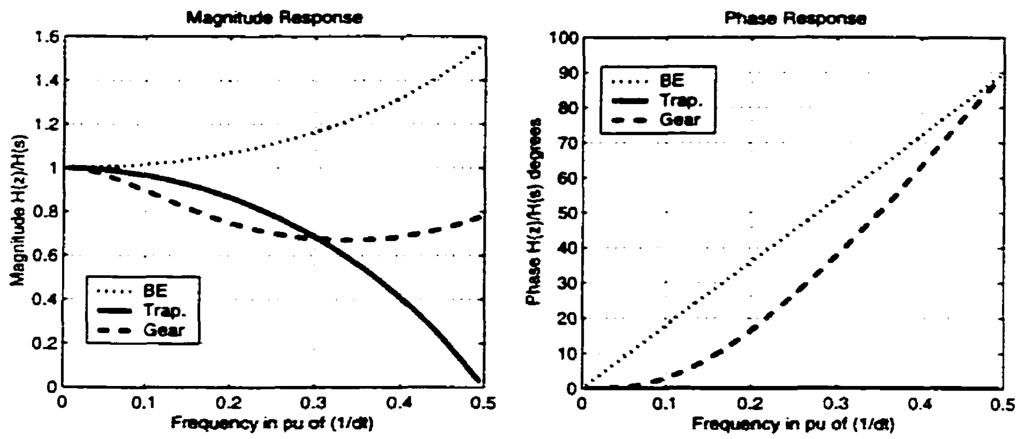


Figure 2.9: Magnitude and phase frequency response for the three numerical schemes.

and for the Gear's 2nd order method it is

$$H_{gear}(z) = \frac{\Delta t}{L} \frac{2z^2}{3z^2 - 4z + 1} \tag{2.30}$$

To evaluate the accuracy of the methods, their frequency response as integrators in the discrete-time domain $H(z)$ is compared with the exact frequency response of an integrator in the continuous-time domain $H(s) = 1/sL$ where $s = j\omega$. Fig 2.9 shows the magnitude and phase components of the ratio $H(z)/H(s)$ as a function of frequency in per unit of $1/\Delta t$. The magnitude response shows that BE and Trapezoidal rules have good accuracy upto about 1/5 of the Nyquist frequency ($1/2\Delta t = 0.5pu$), whereas Gear's method is less accurate than the other two.

The phase distortion of Trapezoidal rule is zero whereas BE and Gear's introduce a strong frequency dependent phase distortions. The phase errors in BE and Gear's distort the nature of the circuit element by introducing a fictitious resistance in its discrete-time representation. Due to the phase errors peaks in a waveform can be missed. Thus a smaller step-size has to be used for BE and Gear's than for Trapezoidal to obtain similar accuracy. Accordingly, BE and Gear's are slower than Trapezoidal.

The stability of the three methods as integrators is indicated by the poles of their transfer functions $H(z)$. All three methods have good stability properties as the stability condition $|z_i| \leq 1$ is met for all of them.

From the above discussion it can be concluded that Trapezoidal method is an overall good choice. As an integrator it is accurate, fast and stable. BE and Gear's can be used when damping is needed in a simulation. Gear's has an additional problem that it is not self-starting; (2.24) shows that two values of i, at $(t - \Delta t)$ and $(t - 2\Delta t)$ are needed to start the method. The Trapezoidal rule is also a good complement to the linear interpolation scheme adopted in the FICS real-time simulation algorithm of Section 2.4.

2.6 Conclusions

The objective of this chapter was to put the problem of asynchronous switching events in digital simulation into perspective. This problem commonly occurs when a real-time digital simulator simulating power electronic circuits is interfaced with a digital controller. The outputs of the digital controller namely gating pulses for firing power electronic switches, typically do not coincide with the discrete time step of the simulator resulting in switching delay which inturn causes errors in the simulation output. The main conclusions of this chapter can be summarized as follows :

1. Several conventional off-line techniques that deal with the problem of switching events in digital simulation have been studied with an intent for their rel-time application. It was found that these techniques could not be used for real-time simulation because they either violated real-time constraints or were too complex to implement.

2. The interaction of a real-time digital simulator with a digital controller has been modeled with the aid of sampling theory. This model gives an insight into the origin of simulation errors caused when switching events are not accounted at their

proper timing.

3. A new algorithm (FICS) for real-time digital simulation of switched power circuits has been presented. This algorithm depends on registration of the proper timing of the incoming switching events and applies a *correction* procedure to calculate the next system state based on that information. The *correction* procedure employs linear interpolation as a means to estimate the system state at the time of occurrence of the switching event.

4. Since the choice of a numerical integration scheme is central to any simulation problem, the numerical schemes for IVPs were analyzed from the perspective of their suitability for real-time simulation. The Trapezoidal Rule was found to be an overall good choice albeit Backward Euler and Gear's second order method were also considered suitable. The Trapezoidal Rule being a linear approximation itself was found to be an ideal compliment to the linear interpolation scheme adopted in the FICS real-time simulation algorithm.

Chapter 3

Modeling, Control and Off-line Digital Simulation of PWM VSI System

3.1 Introduction

Having developed an algorithm for taking switching events into account in digital simulation, it is necessary to employ this algorithm for simulating a switched power circuit. A three-phase Pulse Width Modulated (PWM) Voltage Source Inverter (VSI) is a commonly used switched circuit for a variety of power system applications. Therefore, this system was selected as a simulation example. The inverter system is simple enough not to clutter the simulation with high dimensional equations and it is sufficiently involved to give an insight into the switching processes. The low dimensionality of the system equations is of particular advantage when it comes to implementing the system on a real-time simulation platform.

The purpose of this chapter is to present the modeling and control system details of the VSI system. The inverter model is exact based on switching functions. Mathematical equations are derived to describe the system behavior. The control design is carried out in the synchronous *d-q* frame. In carrying out the off-line time-domain simulation of the system the main objective was to emulate the conditions which would be encountered during an experimental implementation. To that end several implementation issues are discussed in detail. The off-line simulation is carried out using two algorithms: the fixed step-size algorithm and the FICS algorithm proposed in Section 2.4. Simulation studies under both open loop and closed loop control are carried out to compare the performance of the two algorithms.

3.2 Modeling of the VSI System

The circuit diagram of the three phase VSI system is shown in Figure 3.1(A). The inverter consists of three legs, one for each phase. Each inverter leg is made up of two switching components S one each in the upper and lower halves, for example, components 1 and 2. Each switching component S is actually composed of a gate turn-off switch T such as a GTO or IGBT and an anti-parallel diode D. The output voltage of each leg with respect to the negative DC bus N, for example v_{AN} depends only on the DC bus voltage v_{dc} and the switch status and is independent of the output load current due to the presence of the anti-parallel diodes and complementary switchng of the switches in the same inverter leg. The inverter is connected to the AC source voltage E through a network comprising of an R-L series combination. All subsequent analysis and simulation is based on the following fundamental assumptions :

1. The AC source voltages are sinusoidal, balanced and harmonic free.
2. All switching devices T are ideal i.e., switches have zero 'on' resistance and infinite 'off' resistance and have zero turn-on and turn-off times.
3. All other circuit components are linear.

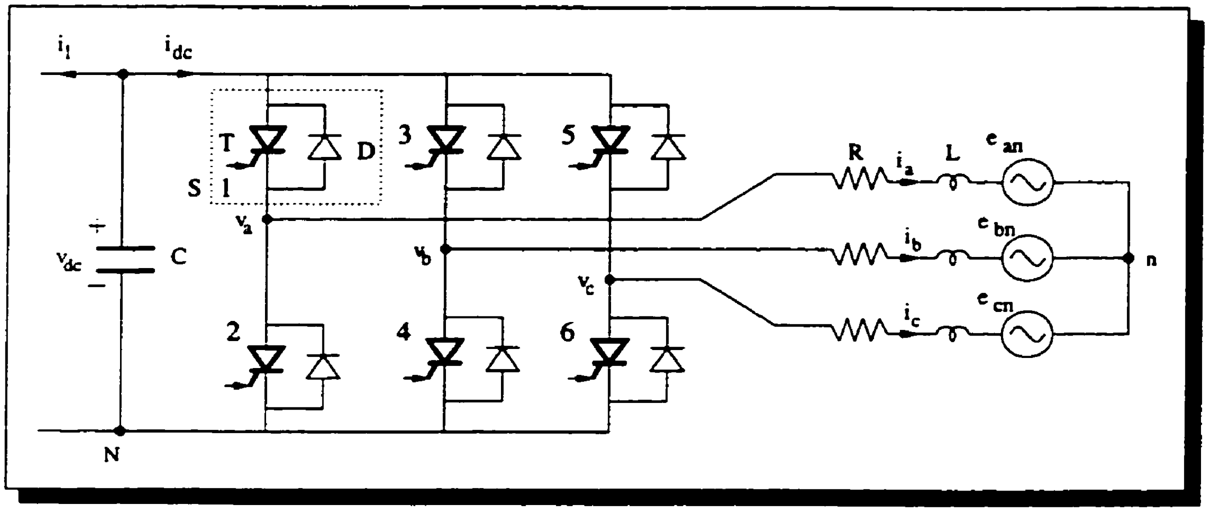


Figure 3.1: Three phase Voltage Source Inverter System

3.2.1 Switching Function Model of the Inverter

A switching function specifies the state of the switch i in the inverter. There are six switching functions corresponding to the six switches in Figure 3.1(A). The switching function S_i corresponding to switch i is defined as :

$$S_i = 1 \text{ if Switch } i \text{ is ON} \tag{3.1}$$

$$= 0 \text{ if Switch } i \text{ is OFF} \tag{3.2}$$

Since the two switches in each inverter leg are switched in a complementary fashion it is necessary to define only three switching functions S_a, S_b, S_c for the three inverter legs. For switches in the upper half $S_1 = S_a$, $S_3 = S_b$, $S_5 = S_c$ and for switches in the lower half $S_2 = S_a^{'}$, $S_4 = S_b^{'}$, $S_6 = S_c^{'}$. The switching functions S_k $(k = a, b, c)$ are determined by the modulation scheme. The output voltages of the inverter with respect to the negative DC bus N are given as :

$$v_{kN} = V_{dc} \quad \text{if } S_k = 1 \tag{3.3}$$

$$= 0 \quad \text{if } S_k = 0 \tag{3.4}$$

Thus,

$$v_{kN} = S_k * V_{dc} \quad k = a, b, c \tag{3.5}$$

3.2.2 System Equations

The differential equations governing the series R-L combination on the AC side are given as :

$$L\frac{di_a}{dt} + R\,i_a = v_{an} - e_{an} \tag{3.6}$$
$$L\frac{di_b}{dt} + R\,i_b = v_{bn} - e_{bn} \tag{3.7}$$
$$L\frac{di_c}{dt} + R\,i_c = v_{cn} - e_{cn} \tag{3.8}$$

For a three phase system without a neutral line

$$i_a + i_b + i_c = 0 \tag{3.9}$$
$$\frac{d}{dt}(i_a + i_b + i_c) = 0 \tag{3.10}$$

and since the supply voltages form a balanced three-phase set

$$e_{an} + e_{bn} + e_{cn} = 0 \tag{3.11}$$

From (3.6) through (3.11) we get

$$v_{an} + v_{bn} + v_{cn} = 0 \tag{3.12}$$

Under balance conditions the inverter output voltages v_{kn} can be expressed as

$$v_{kn} = v_{kN} - v_{nN} \quad {\scriptstyle k=a,b,c} \tag{3.13}$$

From (3.12) and (3.13) we get

$$v_{nN} = \frac{1}{3}\left(v_{aN} + v_{bN} + v_{cN}\right) \tag{3.14}$$

Substituting v_{nN} into equation 3.13 the inverter output voltages with respect to the system neutral n are given as

$$\begin{aligned} v_{an} &= \frac{2}{3}v_{aN} - \frac{1}{3}\left(v_{bN} + v_{cN}\right) && (3.15)\\ v_{bn} &= \frac{2}{3}v_{bN} - \frac{1}{3}\left(v_{aN} + v_{cN}\right) && (3.16)\\ v_{cn} &= \frac{2}{3}v_{cN} - \frac{1}{3}\left(v_{aN} + v_{bN}\right) && (3.17) \end{aligned}$$

The equations governing the DC side of the inverter are given by

$$\begin{aligned} -C\frac{dv_{dc}}{dt} &= i_{dc} + i_l && (3.18)\\ i_{dc} &= \sum_{k=a,b,c} S_k * i_k && (3.19) \end{aligned}$$

3.2.3 *abc*-$\alpha\beta$-*dq* Transformations

For control purposes the system voltages and currents are subjected to a change of coordinates from a three-phase system *a*-*b*-*c* to a synchronously rotating orthogonal two-phase system *d*-*q*. This transformation is performed in two stages. Figure 3.2 illustrates the three coordinate systems.

The first stage is a time invariant transformation that changes the three-phase system *a*-*b*-*c* into a stationary orthogonal two-phase system α-β. The change of coordinates for voltages and currents can be expressed as :

$$\begin{bmatrix} e_\alpha \\ e_\beta \\ e_o \end{bmatrix} = C_1 \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix}, \quad \begin{bmatrix} i_\alpha \\ i_\beta \\ i_o \end{bmatrix} = C_1 \begin{bmatrix} i_a \\ i_b \\ i_o \end{bmatrix} \tag{3.20}$$

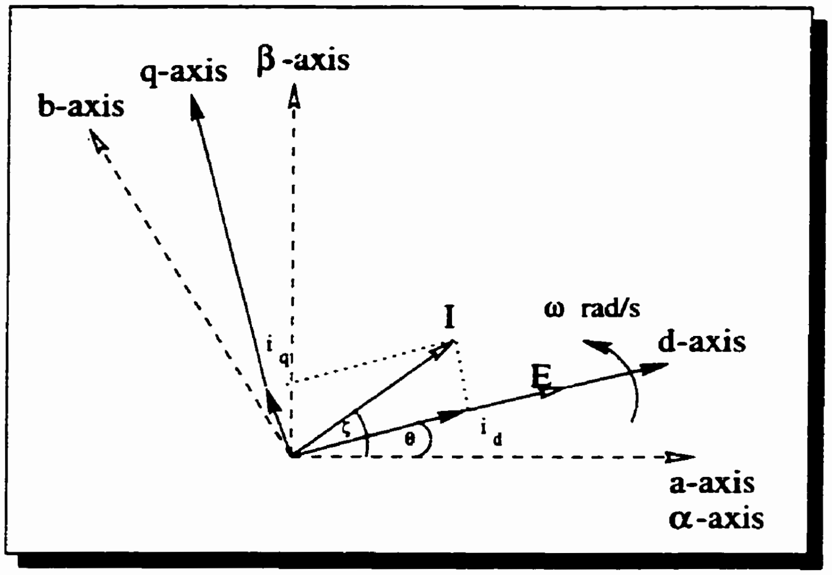


Figure 3.2: Synchronous *d*-*q* frame

where

$$C_1 = \frac{2}{3}\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \tag{3.21}$$

Since the system is assumed to be balanced the zero sequence components v_o and i_o are equal to zero and since $i_a + i_b + i_c = 0$ we get

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} 2/3 & -1/3 & -1/3 \\ 0 & 1/\sqrt{3} & 1/\sqrt{3} \end{bmatrix}\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1/\sqrt{3} & 2/3 \end{bmatrix}\begin{bmatrix} i_a \\ i_b \end{bmatrix} \tag{3.22}$$

and

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} 2/3 & -1/3 & -1/3 \\ 0 & 1/\sqrt{3} & 1/\sqrt{3} \end{bmatrix}\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} 2/3 & 1/3 \\ 0 & 1/\sqrt{3} \end{bmatrix}\begin{bmatrix} v_{ab} \\ v_{bc} \end{bmatrix} \tag{3.23}$$

(3.23) and (3.22) enable the three-phase voltages and currents to be expressed as complex space phasors in terms of their α and β components.

The space phasor notation can compactly represent all the information contained in the three phase system such as unbalance, harmonic distortion etc. At fundamental system frequency V and I move in a circular trajectory in the α-β plane.

The second stage is a time variant transformation that changes the stationary two phase system α-β into a rotating two phase system d-q. The d-axis is coincident with the voltage space phasor v (i.e.,$|v_d| = |v|$ and $|v_q| = 0$) and the q-axis is in quadrature with it. The change of coordinates can be expressed as

$$\begin{bmatrix} i_d \\ i_q \end{bmatrix} = C_2(t) \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix}, \begin{bmatrix} v_d \\ v_q \end{bmatrix} = C_2(t) \begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} \tag{3.24}$$

where

$$C_2(t) = \begin{bmatrix} \cos\theta(t) & \sin\theta(t) \\ -\sin\theta(t) & \cos\theta(t) \end{bmatrix} \tag{3.25}$$

Under balanced steady state conditions the components of the voltage and current space phasors in the d-q frame are constant quantities which facilitates a decoupled description of the two current components.

3.2.4 Pulse Width Modulation

In general a pulse width modulated switching signal is generated by comparing a constant or time varying modulating signal with a triangular signal. In many VSI applications, however, the inverter output is required to be sinusoidal with controllable magnitude and frequency. Therefore a sinusoidal modulating signal $v_{con} = V_{con}\sin(\omega t - \delta)$ is compared (Figure 3.3) with a triangular carrier signal v_{tri} and the intersection points are used to set the switching function S high or low. When $v_{con} > v_{tri}$, $S = 1$ and when $v_{con} < v_{tri}$, $S = 0$. In a three-phase inverter three sinusoidal modulating signals that are 120° out

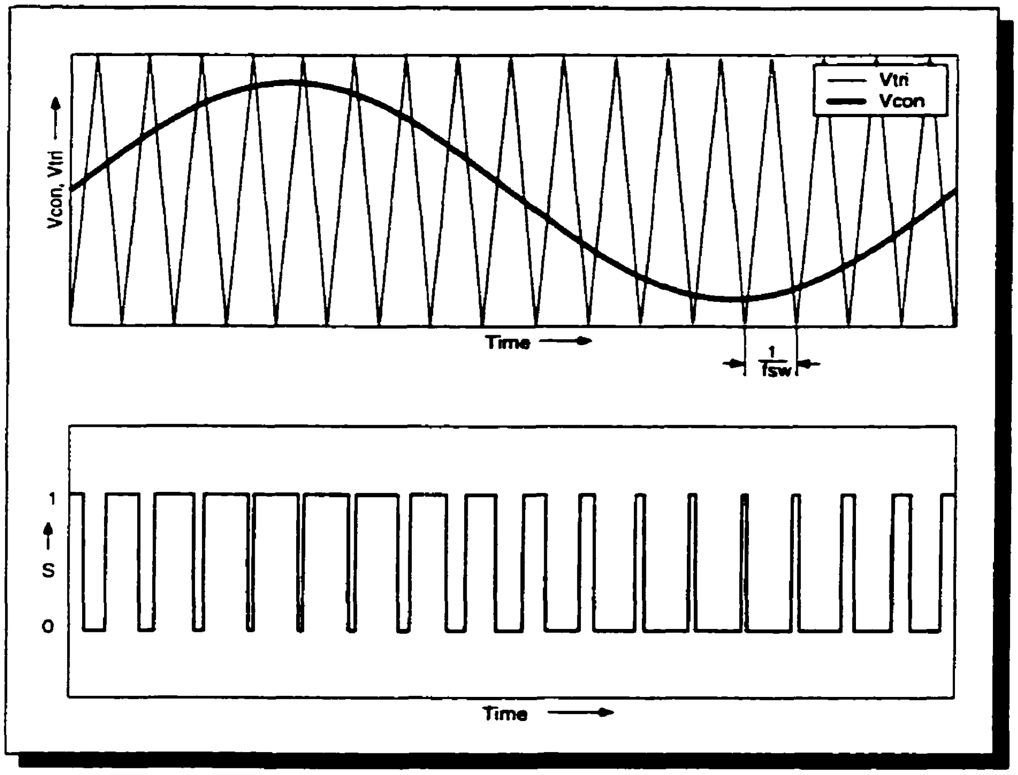


Figure 3.3: Sinusoidal Pulse Width Modulation

of phase are compared with a common triangular carrier signal of fixed amplitude and frequency to generate the three switching functions S_k $(k = a, b, c)$. The frequency of the triangular signal determines the inverter switching frequency f_{sw} and the frequency of the modulating signal determines the fundamental frequency f_1 of the inverter output. The frequency modulation ratio is expressed as $m_f = f_{sw}/f_1$ while the amplitude modulation index is defined as $m_a = V_{mod}/V_{tri}$.

With the DC bus voltage kept constant at V_{dc} the sinusoidal PWM scheme results in switched pulses of V_{dc} across the inverter AC terminals. Fourier analysis shows that the resulting output voltage waveform contains a predominant fundamental component and high frequency harmonics at the carrier and sideband frequencies. The peak value of the fundamental frequency component of voltage v_{aN} in the linear region $(m_a \leq 1)$ is given

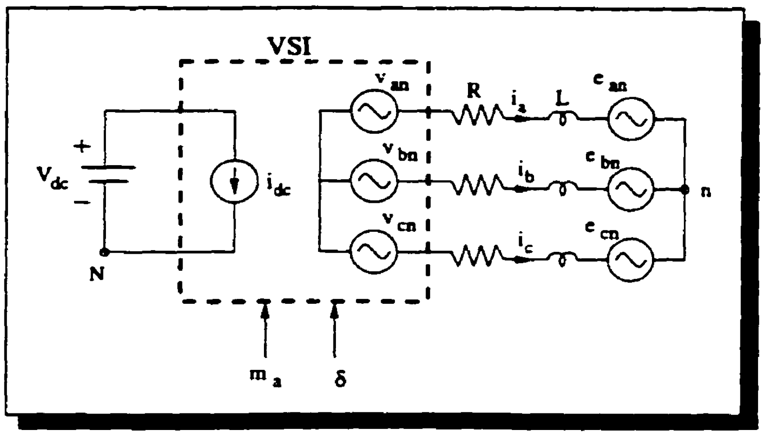


Figure 3.4: Equivalent circuit of the VSI system

by

$$v_{aN1} = \frac{m_a V_{dc}}{2} \tag{3.26}$$

Thus by varying m_a the amplitude of the fundamental Fourier series component of the switched voltage at the AC terminal is also varied proportionally. The phase angle of the fundamental component of the switched voltage can be changed by varying the phase angle δ of the modulating sine wave v_{con} with respect to the reference AC sinusoidal source voltage e. The harmonic spectrum of the inverter output is determined by m_f.

3.2.5 Principle of Operation of the PWM VSI with an AC Source

A simplified block diagram of the PWM VSI system connected to an AC source is shown in Figure 3.4. On the AC side the inverter behaves as a three-phase switched voltage source v_{kn} $(k = a, b, c)$ and on the DC side it behaves as a switched current source i_{dc}. The frequency of the switched voltage sources is synchronized with the frequency of the AC source voltage but their amplitude and phase are controllable using the two parameters m_a and δ. By using the principle of superposition the inverter output voltages can be expressed as

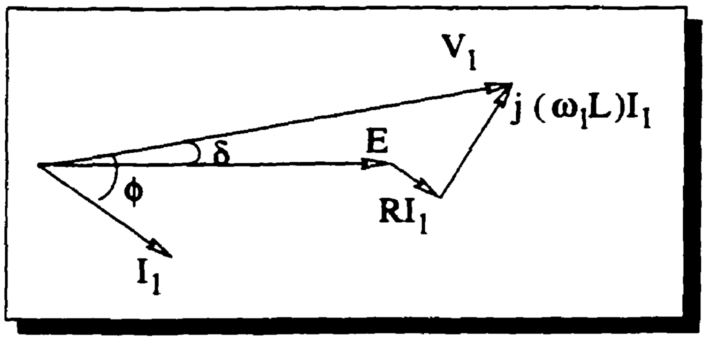


Figure 3.5: Fundamental frequency phasor diagram

$$v_{kn} = v_{kn1} + v_{knh} \quad , \; k = a, b, c \tag{3.27}$$

where v_{kn1} is the fundamental frequency component of v_{kn} and v_{knh} is the switching frequency component of v_{kn}. Similarly the current source i_{dc} on the DC side can be expressed as

$$i_{dc} = I_{dc} + i_{dch} \tag{3.28}$$

where I_{dc} is the average value of i_{dc} and i_{dch} comprises the harmonic component of i_{dc}. At fundamental frequency the dc voltage source acts as an open circuit for the average current source I_{dc} on the dc side. On the AC side the source acts as a short circuit for all harmonic frequencies. Since the AC source voltages are sinusoidal only the fundamental frequency component of the inverter voltage and current are responsible for the real power transfer to and from the AC source. The circuit operation can be expressed as

$$\vec{V_1} = \vec{E} + R\vec{I} + j\omega_1 L\vec{I_1} \tag{3.29}$$

where ω_1 is the fundamental angular frequency of the AC source. The phasor diagram of (3.29) is shown in Figure 3.5. It is apparent that be changing V_{an1} i.e., its amplitude using m_a or its phase shift δ from E_{an} both the line current I_{a1} and the power factor angle ϕ can be controlled. The power balance across the inverter can be expressed as:

$$p_{dc} = p_{ac} \tag{3.30}$$

$$v_{dc} * i_{dc} = \sum_{k=a,b,c} (v_{knl} * i_{knl} + v_{knh} * i_{knh}) \tag{3.31}$$

3.3 Inverter Control in *d*-*q* Frame

As seen from Section 3.2 by varying the modulation index m_a and the phase angle δ the inverter output voltage can be varied thereby controlling the output current and its power factor. This section describes the control strategy to obtain m_a and δ from the sampled system voltages and currents. The control principle is developed in the synchronous *d*-*q* frame which, as stated earliar, simplifies the design by decoupling the state variables. The control effort is two fold: one that controls the inverter current components i_d and i_q and the other that regulates the DC bus voltage v_{dc}. By controlling i_q the reactive power supplied by the inverter can be varied and controlling v_{dc} and i_d in turn the real power exchange can be varied. Proportional-Integral compensators are employed for both the current and voltage controllers. Before describing the control strategy, however, the system model in *d*-*q* frame needs to be developed.

3.3.1 VSI Model in *d*-*q* Frame

Applying the transformation of (3.22-3.23) to (3.6-3.8) and recognizing that v_o , i_o and e_o are equal to zero, the AC side differential equations in the α-β frame can be expressed as

$$\frac{d}{dt} i_{\alpha\beta} = -\frac{R}{L} i_{\alpha\beta} + \frac{1}{L} (v_{\alpha\beta} - e_{\alpha\beta}) \tag{3.32}$$

Applying the transform of (3.24), (3.32) can be expressed in the *d*-*q* frame as

$$\frac{d}{dt}\left(C_2(t)^{-1} i_{dq}\right) = -\frac{R}{L} C_2(t)^{-1} i_{dq} + \frac{1}{L} C_2(t)^{-1} (v_{dq} - e_{dq}) \tag{3.33}$$

$$\Rightarrow \quad C_2(t)^{-1}\frac{d}{dt}i_{dq} + \omega C_3(t) i_{dq} \quad = \quad -\frac{R}{L}C_2(t)^{-1}i_{dq} + \frac{1}{L}C_2(t)^{-1}\left(v_{dq} - e_{dq}\right) \qquad (3.34)$$

where $C_3(t) = \begin{bmatrix} -\sin\theta(t) & -\cos\theta(t) \\ \cos\theta(t) & -\sin\theta(t) \end{bmatrix}$ and $\omega = \frac{d\theta}{dt}$. Pre-multiplying (3.34) by $C_3(t)^{-1}$ we get

$$C_4(t)\frac{d}{dt}\begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega \begin{bmatrix} i_d \\ i_q \end{bmatrix} = -\frac{R}{L}C_4(t)\begin{bmatrix} i_d \\ i_q \end{bmatrix} + \frac{1}{L}C_4(t)\left(\begin{bmatrix} v_d \\ v_q \end{bmatrix} - \begin{bmatrix} e_d \\ e_q \end{bmatrix}\right) \qquad (3.35)$$

where $C_4(t) = C_3(t)^{-1}C_2(t)^{-1} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$. Now, for the DC side of the inverter the power balance (3.31) can be employed to get an expression for i_{dc} in the d-q frame. (3.31) can be rewritten as

$$i_{dc} = \frac{v_{abc}^t * i_{abc}}{v_{dc}} \qquad (3.36)$$

Applying the transform of (3.21) to the right hand side of (3.36) we get

$$i_{dc} = \frac{v_{\alpha\beta o}^t C_1^{-1t} C_1^{-1} i_{\alpha\beta o}}{v_{dc}} = \frac{3}{2}\frac{v_{\alpha\beta}^t i_{\alpha\beta}}{v_{dc}} \qquad (3.37)$$

since $C_1^{-1t}C_1^{-1} = \frac{3}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$ and $v_o = i_o = 0$. Thus,

$$i_{dc} = \frac{3}{2}\frac{v_{dq}^t C_2(t)^{-1t} C_2(t)^{-1} i_{dq}}{v_{dc}} = \frac{3}{2}\frac{v_{dq}^t i_{dq}}{v_{dc}} \qquad (3.38)$$

since $C_2(t)^{-1t}C_2(t)^{-1} = I_{2\times 2}$. From (3.35) and (3.38) the complete system model in the $d - q$ frame can be written as

$$\frac{di_d}{dt} \quad = \quad -\frac{R}{L}i_d + \omega i_q + \frac{1}{L}\left(v_d - e_d\right) \qquad (3.39)$$

$$\frac{di_q}{dt} \quad = \quad -\frac{R}{L}i_q - \omega i_d + \left(v_q - e_q\right) \qquad (3.40)$$

$$\frac{dv_{dc}}{dt} = -\frac{3}{2}\frac{(v_d i_d + v_q i_q)}{C v_{dc}} - i_l \tag{3.41}$$

In the above equations e_d and e_q are the d and q components of the AC source voltages $\vec{e}(e_d = |\vec{e}|$ and $e_q = 0$) since the d-axis is coincident with $\vec{e}$. Using (3.26) and the phasor diagram in Figure 3.5 the d and q components of the inverter output voltage can be expressed as

$$v_d = 0.5\, m_a\, v_{dc} \cos\delta \tag{3.42}$$

$$v_q = 0.5\, m_a\, v_{dc} \sin\delta \tag{3.43}$$

3.3.2 P-I Control of i_d , i_q and v_{dc}

From inspection it is apparent that equations (3.39) and (3.40) are cross-coupled in terms of i_d and i_q. Introducing two new variables O_d and O_q, the inverter output voltage components can be written as

$$v_d = L\,(O_d - \omega i_q) + e_d \tag{3.44}$$

$$v_q = L\,(O_q + \omega i_d) + e_q \tag{3.45}$$

Substituting (3.44-3.45) in (3.39-3.40) we get simple first order decoupled equations for i_d and i_q

$$\frac{di_d}{dt} = -\frac{R}{L} i_d + O_d \tag{3.46}$$

$$\frac{di_q}{dt} = -\frac{R}{L} i_q + O_q \tag{3.47}$$

O_d and O_q can be defined in terms of proportional-integral compensators and feedback variables as

$$O_d = G_i(s) * (i_d^* - i_d) \tag{3.48}$$

$$O_q = G_i(s) * \left(i_q^* - i_q\right) \tag{3.49}$$

where $G_i(s) = \frac{K_i(1+sT_i)}{sT_i}$ is the transfer function of the current controllers.

The reactive current reference i_q^* can be derived from the reactive power requirement of the inverter while the real current reference i_d^* is obtained from the output of an outer feedback control loop that regulates the DC bus voltage as

$$i_d^* = G_v(s)\,(v_{dc}^* - v_{dc}) \tag{3.50}$$

where $G_v(s) = \frac{K_v(1+sT_v)}{sT_v}$ is the transfer function of the PI compensator for v_{dc}.

For digital implementation the s-domain transfer functions of the PI controllers need to be converted into the z-domain. Substituting Tustin's formula $s = \frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}}$ in $G_i(s)$ we get

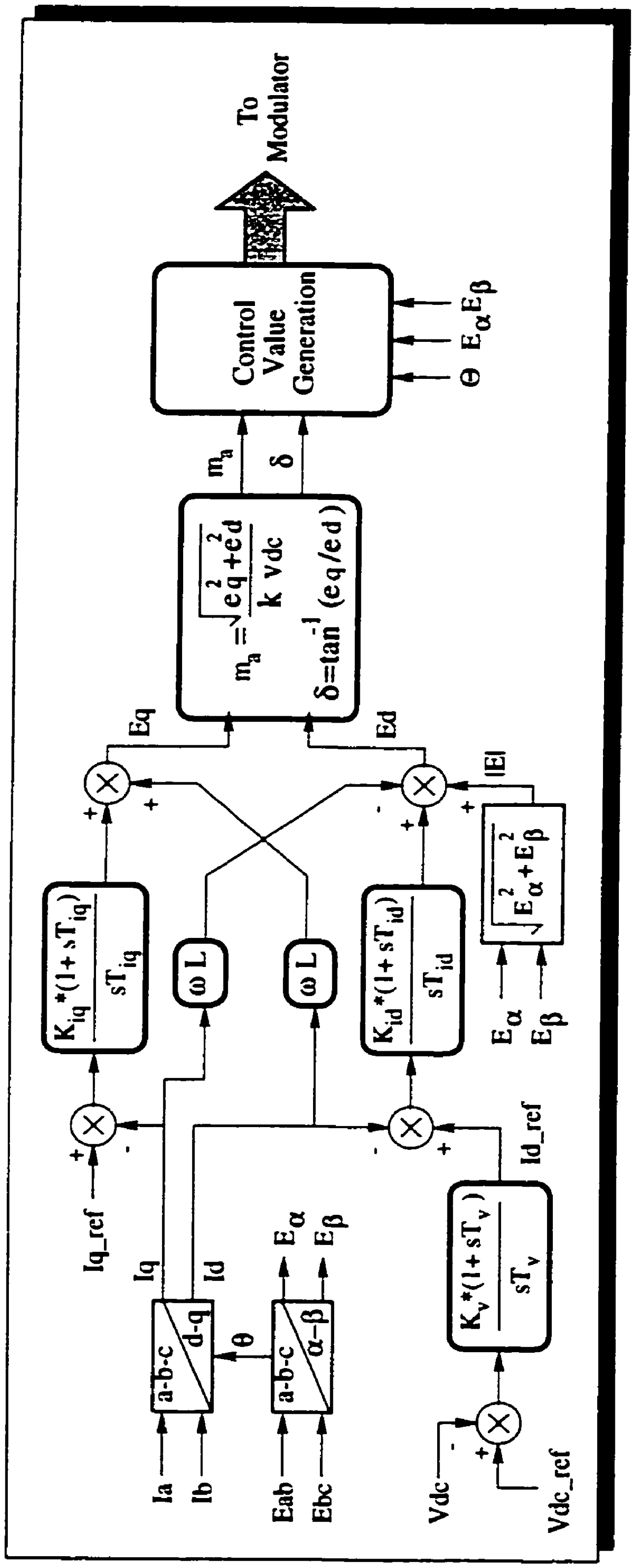
$$G_i(z) = \frac{K_i\left[\left(1 + \frac{T_s}{2T_i}\right)z - \left(1 - \frac{T_s}{2T_i}\right)\right]}{z-1} \tag{3.51}$$

where T_s is the controller sampling period. The discrete time recursive equation for the PI output O_k thus is given as

$$O_{kn} = O_{k(n-1)} + K_{i1}\,(i_{kn}^* - i_{kn}) + K_{i2}\left(i_{k(n-1)}^* - i_{k(n-1)}\right)\ ,\quad \text{k} = d, q \tag{3.52}$$

where $K_{i1} = K_i\left(1 + \frac{T_s}{2T_i}\right)$ and $K_{i2} = -K_i\left(1 - \frac{T_s}{2T_i}\right)$. Similarly, the output of the PI controller on v_{dc} is given by

$$i_{dn}^* = i_{d(n-1)}^* + K_{v1}\,(v_{dcn}^* - v_{dcn}) + K_{v2}\left(v_{dc(n-1)}^* - v_{dc(n-1)}\right) \tag{3.53}$$



Ia
Ib
Eab
Ebc
a-b-c
d-q
a-b-c
α-β
θ
Iq
Id
Eα
Eβ
Vdc
Vdc_ref
Iq_ref
Id_ref
$K_v*(1+sT_v)/sT_v$
$K_{iq}*(1+sT_{iq})/sT_{iq}$
$K_{id}*(1+sT_{id})/sT_{id}$
ωL
ωL
$\sqrt{E_\alpha^2+E_\beta^2}$
|E|
Eq
Ed
$m_a=\sqrt{eq^2+ed^2}/k\ vdc$
$\delta=\tan^{-1}(eq/ed)$
m_a
δ
Control Value Generation
θ Eα Eβ
To Modulator

Figure 3.6: Control block diagram for the VSI system

The complete control algorithm is summarized in the block diagram of Figure 3.6. The system samples consist of two line currents i_a and i_b , two line to line voltages v_{ab} and v_{bc} and the DC bus voltage v_{dc}. A coordinate transformation is performed from abc to dq frame. Depending on i_q^* and v_{dc}^* the inverter voltage vector $\overrightarrow{v_{dq}}$ is then determined. Finally using m_a and δ the switching functions S_k $(k = a, b, c)$ are derived by the PWM generator.

3.4 Off-line Digital Simulation

This section presents the details of off-line time domain digital simulation of the PWM VSI system. The objective is to emulate the implementation conditions of an experimental setup as closely as possible. To that end the two major issues related to practical implementation of a digital controller namely, the implementation of Pulse Width Modulation and the controller dead time are discussed in detail. A third equally important issue but related to simulation, the selection of an appropriate time step, is also discussed. Two approaches were used for off-line simulation: the fixed step-size approach and the FICS algorithm presented in detail in Section 2.4. Given the difference in the simulation method of the FICS approach from that of the fixed step-size approach, standard programs such as EMTP or PSCAD could not be used for off-line simulation. Even for the fixed step-size approach several modeling and implementation differences including those that are mentioned in this section precluded the use of such programs. The hurdles encountered while simulating the system using EMTP or PSCAD prompted the development of new off-line simulation programs.

3.4.1 Implementation of SPWM

As seen from Section 3.2.4, generation of switching functions S_k $(k = a, b, c)$ requires that a high frequency (f_{sw}) triangular carrier signal be compared with a low frequency

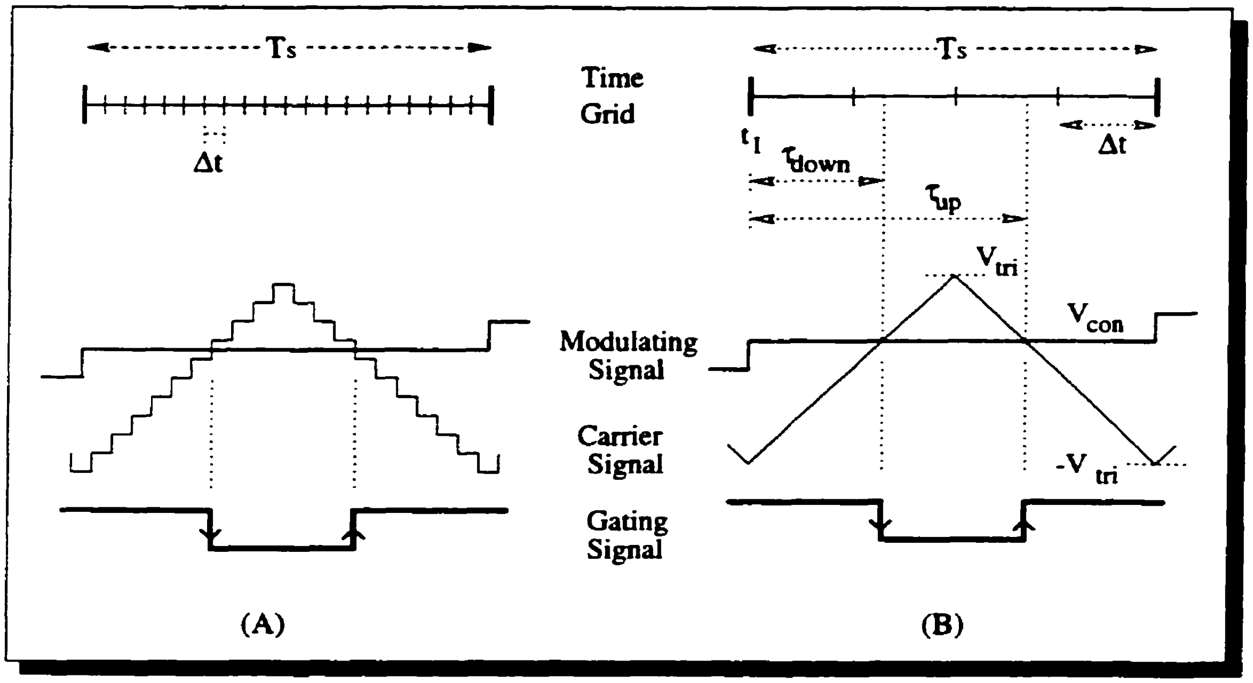


Figure 3.7: Comparison of PWM implementations in digital simulation

(f_1) sinusoidal modulating signal. The way in which the carrier and modulating signals are implemented in the digital simulation program significantly influences the simulation output. In practical digital controllers, the modulating signal is generated at a fixed sampling period T_s and the carrier signal at the desired frequency, generated by special digital hardware, has a very fine resolution (typically the time resolution of the carrier wave is in tens of nanoseconds). Furthermore, the controller sampling period is always synchronized with the switching frequency of the inverter i.e., $f_{sw} = 1/nT_s$, $n = integer$. In a standard off-line simulation program such as EMTP, since the PWM pattern generator is part of the entire system simulation, the carrier wave is generated with the same time step Δt at which the system simulation is carried out. This method has two undesirable side effects related to the speed and accuracy of the simulation.

1. It restricts the choice of Δt for the system simulation due to the fact that a very small Δt has to be chosen so as to have a high carrier resolution which results in a longer simulation time.

2. If a larger Δt were chosen based on the simulation requirements of the system transients, although the total simulation time is kept low, the resolution of the carrier signal in the PWM generator suffers which in turn affects the PWM output and the accuracy of the simulation.

As an example, a system simulation that normally requires only a time step of $50\mu s$ for capturing the required system transients would need a time step of $1\mu s$ or less if it includes a PWM generator with a carrier frequency of $1KHz$ or above. Thus if $\Delta t = 1\mu s$ the entire simulation slows down considerable whereas if $\Delta t = 50\mu s$ the resolution of PWM generator is affected yielding inaccurate results.

The sampling technique approach [30] used here circumvents the above dilemma. In keeping with the requirements of a practical digital controller the sinusoidal modulating signal is generated only at periodic time intervals T_s and the triangular carrier signal is defined using straight line segments of pre-defined slope in that sampling interval; the triangular carrier does not really exist as a signal. The switching instants are then computed, in the course of the program execution, as the respective points where the triangular slope reaches the control value. Figure 3.7 compares the conventional approach (A) with the adopted approach (B); in both approaches the switching frequency f_{sw} is set equal to the controller sampling frequency $1/T_s$. In Figure 3.7(A) the resolution of the carrier is dependent on Δt; the only way to increase the carrier resolution is to reduce the time step. In Figure 3.7(B) the carrier resolution is essentially independent of Δt thereby allowing an exact calculation of the switching times. The switching times in Figure 3.7(B) are given as:

$$\tau_{down} = t_1 + \frac{T_s}{4V_{tri}}(V_{con} + V_{tri}) \tag{3.54}$$

$$\tau_{up} = t_1 + T_s - \tau_{down} \tag{3.55}$$

3.4.2 Digital Controller Dead Time

The implementation of a practical digital controller mainly involves three stages :

1. The sampling of analog signals from the system and their subsequent conversion to digital form by A/D converters.
2. The execution of the control algorithm.
3. Generation of the switching signals.

Every sampling period T_s all the above functions are performed. The execution of these functions by a microprocessor or DSP takes a finite amount of time. Functions 1 and 3 usually take up a fixed amount of time but the time taken by function 2 may vary depending on the length of the control algorithm and whether or not it incorporates blocks of code, activated by the user on-line, which are not activated during normal execution. The total execution time may therefore be less than or even equal the sampling period T_s . It is thus not possible to implement the switching functions in the same sampling interval in which they are being computed. A delay of one sampling period is therefore introduced in the implementation even if the execution time is less than T_s. In order to emulate the behavior of a digital controller faithfully this dead time must be taken into account during simulation. In the developed off-line simulation programs the switching signals from the controller are delayed by one T_s before being sent to the system simulation module.

3.4.3 Choice of the Simulation Time Step Δt

Traditionally, the criterion for choosing a time step for digital simulation of power system transients was the following :

`` **the time step Δt must be at least two times smaller than the time period of the fastest transient being simulated** ''

This criterion is a direct consequence of the Sampling Theorem [28] which sets the Nyquist frequency $(1/2\Delta t)$. A universally agreed upon time step for off-line as well as real-time simulation of power systems is $\Delta t = 50\mu s$. This requirement pertains to power system transients that have a high frequency component of $10KHz$ or lower. Therefore, with $\Delta t = 50\mu s$, theoretically transients containing frequencies of up to $10KHz$ can be faithfully reproduced in the simulation. This premise is based on the assumption that the behavior of the adopted numerical method is independent of frequency. As seen from Section 2.6, however, most commonly used numerical methods have strongly frequency dependent magnitude and phase properties. As frequency increases these properties deteriorate rapidly even before the Nyquist limit is reached. As an example Table 3.1 shows the magnitude error $H(z)/H(s)$ of the Trapezoidal method as a function of frequency for $\Delta t = 50\mu s$. Clearly with such a time step transients of only upto $4KHz$ can be reliably reproduced.

The above criterion is no longer valid when power circuits containing switching elements are simulated. The choice of the time step, in this case, is governed by the resolution of the switching waveforms. From the experience gained by simulations carried out in this thesis, the following criterion emerges :

`` **using a fixed step-size numerical method the time step Δt must be at least 50 to 100 times smaller than the switching time period T_s**''

For example, if the switching frequency is $1KHz$ ($T_s = 1ms$), the time step for the simulation must be $20\mu s$ or smaller to obtain fairly accurate and reliable results. As will be seen later, the FICS real-time simulation algorithm proposed in Section 2.4 relaxes this stringent requirement on the time step.

f (kHz)	1	2	4	6	8	10
Error (%)	0.82	3.3	13.5	31.5	59.2	100

Table 3.1: Magnitude error variation with frequency for Trapezoidal rule with $\Delta t = 50\mu s$

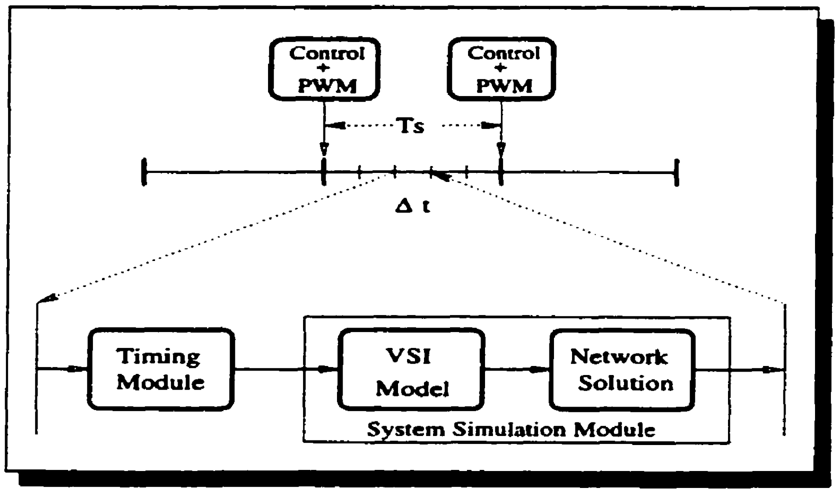


Figure 3.8: Off-line simulation program structure

3.4.4 Off-line Simulation Programs

Two separate programs were developed for the off-line simulation of the PWM VSI system. The program `nor_c11.c` uses the fixed step-size approach while program `intrp_c11.c` uses the FICS algorithm detailed in Section 2.4. Both programs were written in ANSI C for easy portability and their code can be found in the Appendix D. Figure 3.8 illustrates a common template for either programs on the simulation time grid. Two different time scales can be recognized : Δt is the simulation time step and T_s is the controller sampling interval. Every time step mainly two blocks of code are executed, one is the switch Timing Module and the other is the System Simulation module. The function of the Timing module is to determine the switching times of firing signals coming from the controller and to activate `flag_intrp` if a switching event was detected in the previous time step. This information is used in the FICS simulation algorithm to determine the time instant for interpolation. In the fixed step-size approach the System Simulation module mainly consists of the exact VSI model based on switching functions (3.3-3.5) followed by the solution of equations in Section 3.2.2; Trapezoidal numerical method is employed for the network solution. Using the FICS algorithm, however, the System Simulation module consists of two paths of execution : normal operation (Section 2.5.1) and post event operation (Section 2.5.2) depending on the status of `flag_intrp`. The Control module is identical for both programs and consists of the functions shown in Figure 3.6.

3.5 Off-line Digital Simulation Results

As seen in Section 3.4.3, in the simulation of switched circuits the selection of the time step Δt depends primarily on the desired switching resolution. Switching delay is always present when a fixed step-size algorithm is used to account for firing pulses coming in between two calculation steps. It is the time step Δt that decides whether this delay is small or large and thereby influences the simulation output. The focus of this section is therefore to examine the effects of varying time steps on the simulation output. Results are presented for off-line simulations carried out with programs developed using the fixed step-size method (`nor_c11.c`) and the FICS algorithm (`intrp_c11.c`). Both open loop and closed loop control studies were undertaken on the PWM VSI system. The parameters of the system are given in the Appendix B. The objective of the simulations was to address the following critical issues :

1. The nature and magnitude of errors encountered using the fixed step-size algorithm.
2. Performance of the proposed algorithm in comparison to the fixed step-size algorithm.
3. Suitability of the proposed algorithm for real-time simulation.

3.5.1 Open Loop Results

Under open loop conditions the output of the VSI can be varied by varying the modulation index m_a and the phase angle δ. According to the criterion suggested in Section 3.4.3, a time step of $\Delta t = 10\mu s$ that is 100 times smaller than the switching time period Ts

($= 1ms$) was selected to serve as a standard. The simulation results obtained with $\Delta t = 10\mu s$ were used as a reference against which the results obtained with limited time steps $\Delta t = (50, 100, 150, 200\mu s)$ were compared. Time steps lower than $10\mu s$ were not selected because they only led to increase in the simulation time without offering any noticeable change in the output. For those time steps the difference in simulation errors with respect to $\Delta t = 10\mu s$ asymptotically approached zero.

Figures 3.9 and 3.10 show steady state current waveforms obtained using the fixed step-size and the FICS algorithm using $m_a = 0.8$ and $\delta = 10°$. In Figure 3.9 the effect of increasing the step-size can be observed; the peaks of the waveforms increase and in the $\Delta t = 150\mu s$ and $\Delta t = 200\mu s$ cases a 1/3 harmonic of the fundamental can be clearly seen. In Figure 3.10 the integrity of the waveforms is adequately maintained till $\Delta t = 150\mu s$. The bar graphs of Figures 3.11 and 3.12 compare the first 20 harmonics of the currents for the different time steps and the two simulation approaches. The two major effects observed due to an increase in Δt were :

1. A significant change in the fundamental component.
2. A marked crowding of the frequency spectrum with non-characteristic harmonics.

Table 3.2 gives the fundamental magnitudes errors and the Total Harmonic Distortion (THD) for different time steps using the two simulation approaches. Clearly a time step of $50\mu s$ is inadequate for simulation using the fixed step-size approach due to a high error of 14%. The error roughly doubles as the time step is doubled. An increase in THD, with respect to the $\Delta t = 10\mu s$ case, can also be seen as the time step is increased. In contrast, for the proposed FICS approach the error stays around 3% till $\Delta t = 150\mu s$. The THD, however, is higher in some cases compared to the fixed step-size method. This result can be explained using Figure 2.7 in which state x_2 is left uncorrected after the interpolation step since the time t_2 is taken to have passed in real-time. State x_2 is deliberately not corrected to conform to the action of a real-time simulator which cannot

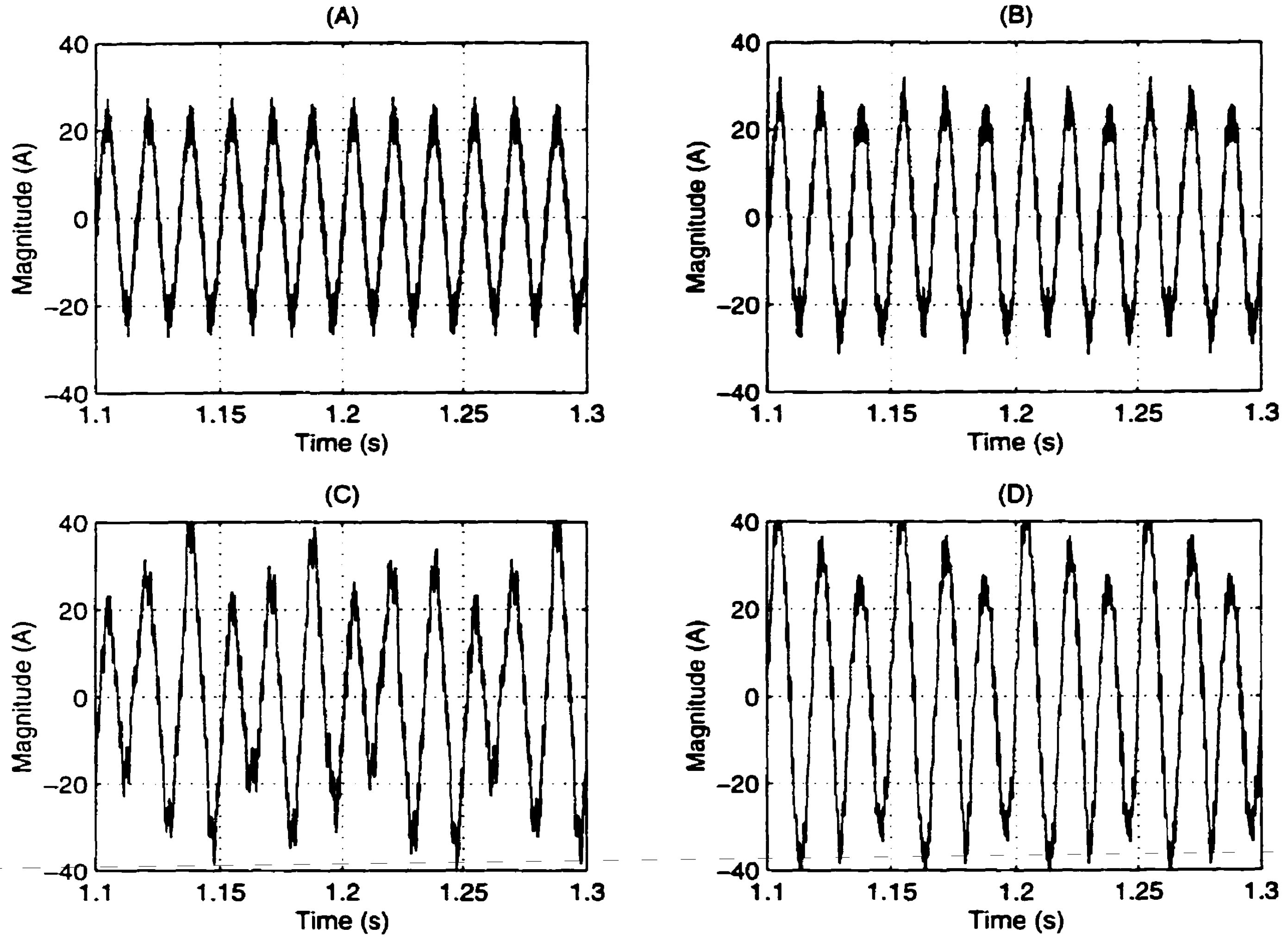


Figure 3.9: Steady state current I_a simulated using the fixed step-size algorithm with (A)$\Delta t = 50\mu s$ (B)$\Delta t = 100\mu s$ (C)$\Delta t = 150\mu s$(D)$\Delta t = 200\mu s$

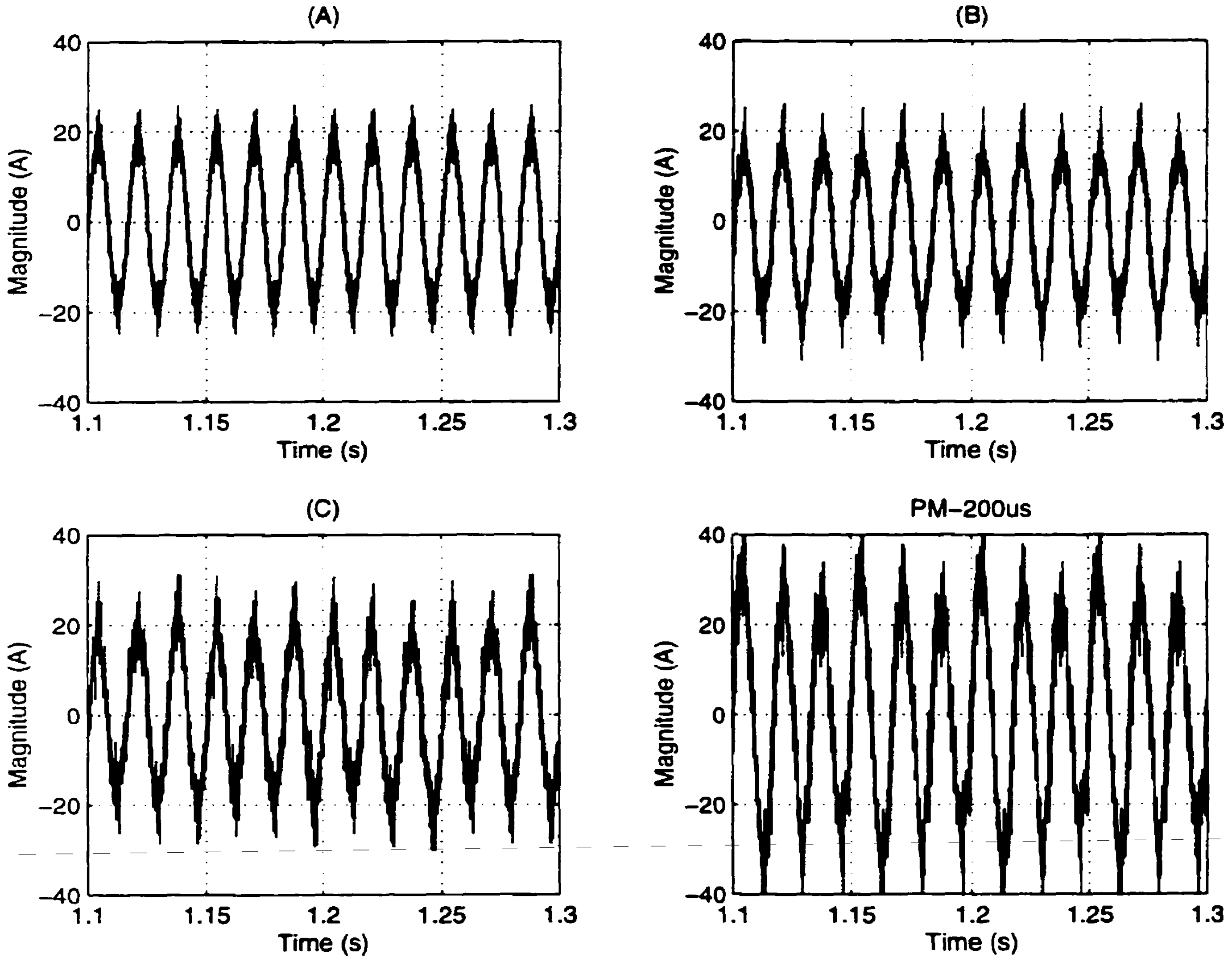
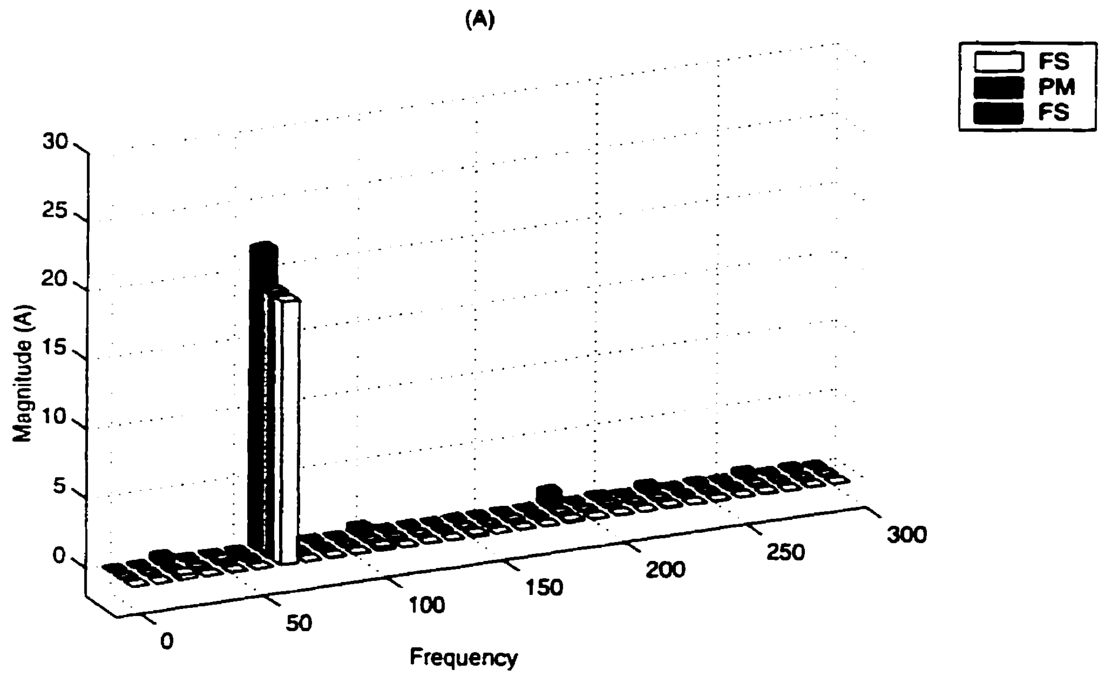


Figure 3.10: Steady state current I_a simulated using the proposed FICS algorithm with (A) $\Delta t = 50\mu s$(B) $\Delta t = 100\mu s$ (C) $\Delta t = 150\mu s$ (D) $\Delta t = 200\mu s$



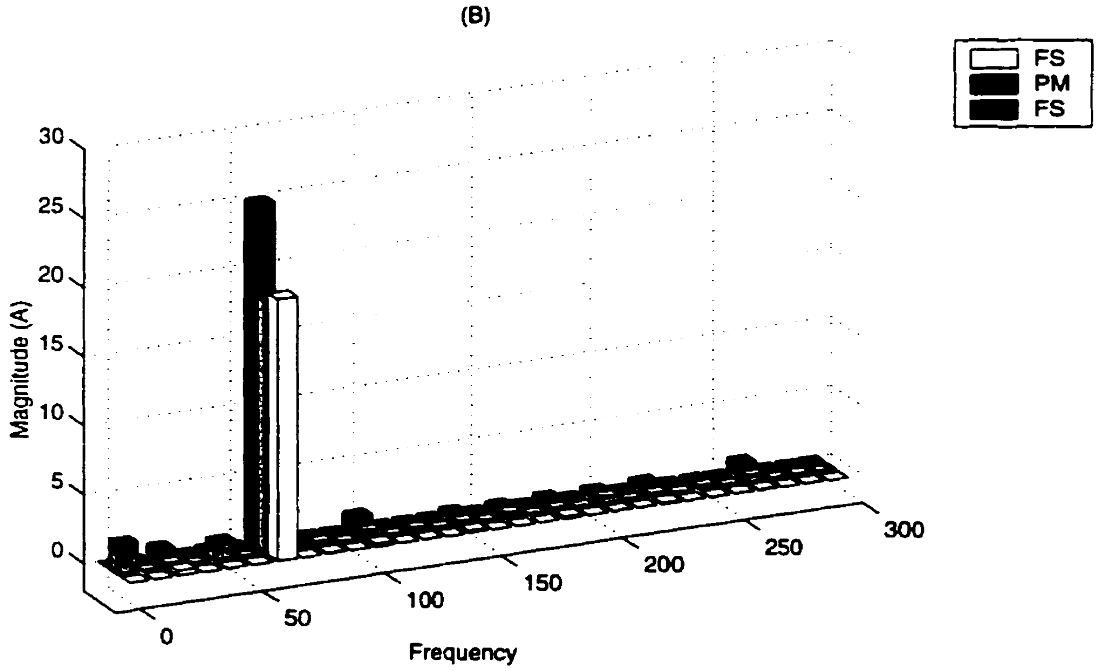
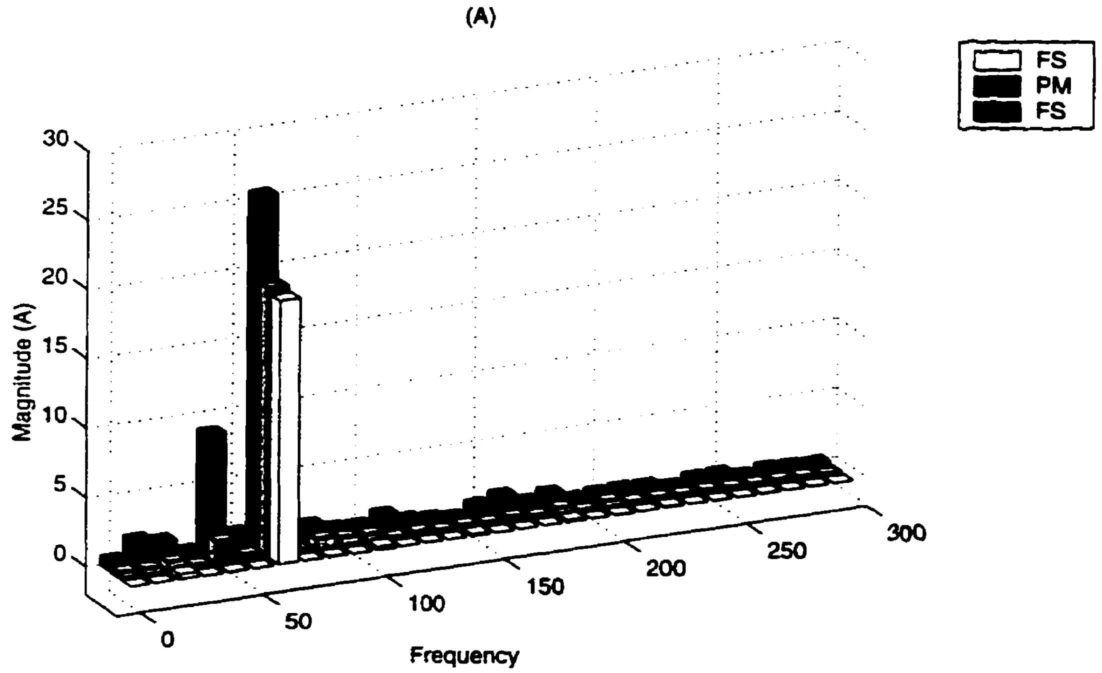


Figure 3.11: Frequency spectrum of I_a. FS : Fixed step-size method, PM : Proposed (FICS) method. (A)FS with $\Delta t = 10\mu s$ and $\Delta t = 50\mu s$, PM with $\Delta t = 50\mu s$ (B) FS with $\Delta t = 10\mu s$ and $\Delta t = 100\mu s$, PM with $\Delta t = 100\mu s$



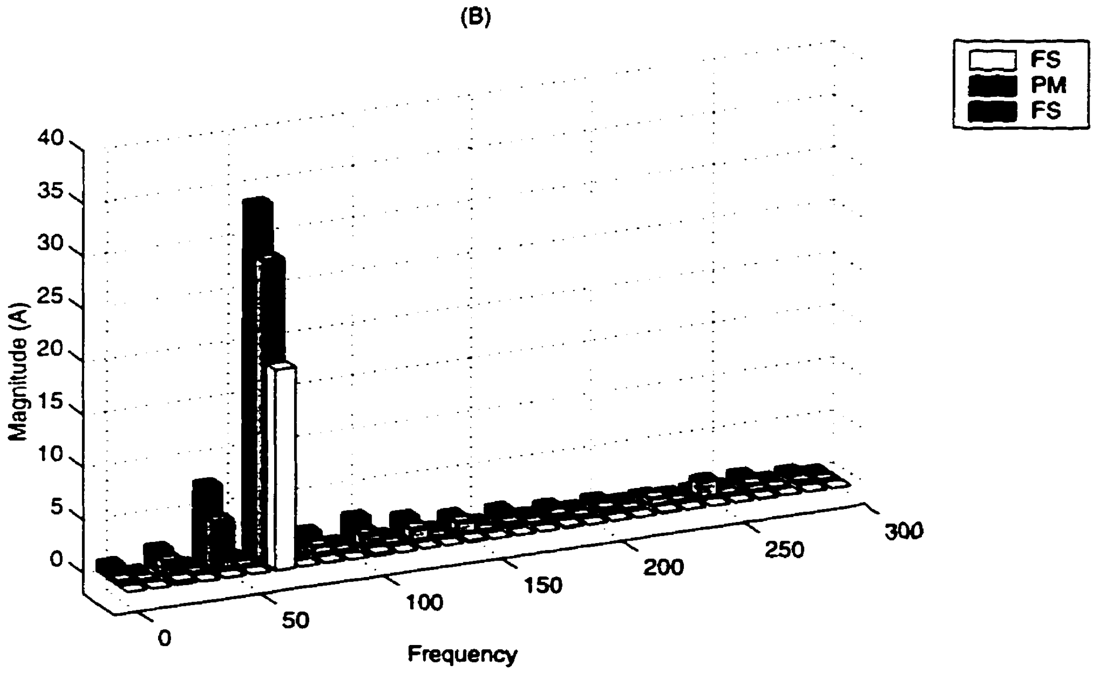


Figure 3.12: Frequency spectrum of I_a. FS: Fixed step-size method, PM: Proposed (FICS) method. (A) FS with $\Delta t = 10\mu s$ and $\Delta t = 150\mu s$, PM with $\Delta t \approx 150\mu s$ (B) FS with $\Delta t = 10\mu s$ and $\Delta t = 200\mu s$, PM with $\Delta t = 200\mu s$

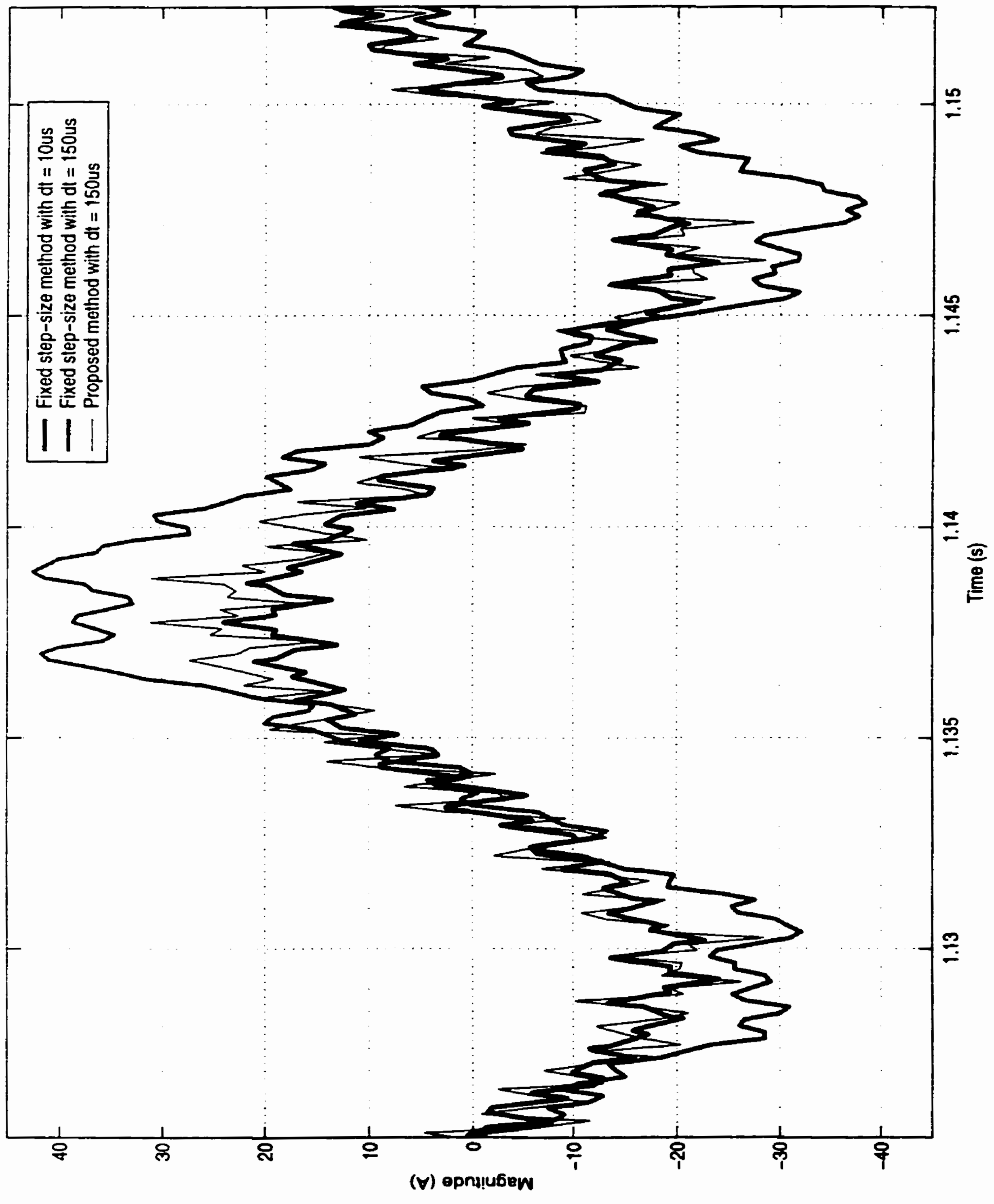


Figure 3.13: Detailed view of the steady state current I_a simulated using the fixed step-size method and the proposed FICS method

retrieve and correct states that have already been computed and sent out. To determine the breakdown point of the proposed FICS approach a time step of $\Delta t = 200\mu s$, that grossly violates the criterion of Section 3.4.3, was selected. In this case, owing to the time step being overwhelmingly large, the method lost its efficacy, however, as seen from Table 3.2 and Figure 3.12 the error is still significantly lower than that for the fixed step-size approach. Figure 3.13 shows a detailed view of the steady state current waveforms at $\Delta t = 10\mu s$ and $\Delta t = 150\mu s$. This picture essentially summarizes the performance of the FICS approach in comparison to the fixed step-size approach. The FICS approach gave atleast a 10 fold improvement over the fixed step-size approach. This estimate can be considered conservative given that the results for the proposed approach were valid even for $\Delta t = 150\mu s$. The higher accuracy derived with the FICS approach can be attributed to an accurate detection of the gating signals together with the interpolation and variable step-size solution.

Δt (μs)	**Fixed step-size algorithm**		**Proposed FICS algorithm**	
	$Error(\%)$	$THD(\%)$	$Error(\%)$	$THD(\%)$
50	14.0	15.15	0.3	16.7
100	30.0	15.2	3.4	28.14
150	34.0	42.0	2.0	33.6
200	75.0	27.5	50	36.7

Table 3.2: Fundamental magnitude error and THD of I_a simulated using the fixed step-size and the FICS algorithms for varying Δt in comparison to $\Delta t = 10\mu s$

3.5.2 Closed Loop Results

Figure 3.14 shows the VSI closed loop response. A step response in i_q was simulated. The simulations were carried out for $\Delta t = 10\mu s$ and $\Delta t = 100\mu s$ using the fixed step-size and the proposed FICS algorithms. The predominant effect of an increase in the time step was the appearance of jitter in the control quantities. As seen from this figure the traces of i_q for $\Delta t = 10\mu s$ are smooth whereas those for $\Delta t = 100\mu s$ contain a $\pm 4A$ jitter superimposed on the average value. The jitter effect was observed in all the control signals simulated using both the fixed step-size and the proposed FICS algorithms. The fact that the jitter disappears when the simulation time step is reduced indicates that its source is the numerical integration process in the simulation. For $\Delta t = 100\mu s$ it comes as no surprise that the FICS approach should behave similar to the fixed step-size approach since the underlying control algorithm is the same in both approaches. In a practical set-up of the digital controller the jitter may be reduced by filtering the system signals using low-pass filters before they are sampled. In an off-line simulation program the jitter in the control quantities may also be substantially reduced by using linear interpolation in the control algorithm. Such an approach could result in nearly smooth traces for increasing Δt using the fixed step-size approach or the proposed FICS approach. However, no such attempt was made in the developed off-line simulation programs since the primary objective of the study was to emulate a practical digital controller.

Figure 3.15 shows the time domain waveforms fo the VSI line current i_a under closed loop operation and Figure 3.16 shows the Fourier spectrum. Comparing the spectrum of i_a obtained using the fixed step-size approach at $\Delta t = 10\mu s$ and $\Delta t = 100\mu s$ we can see that the dramatic effect (i.e., a significant change in the fundamental component) observed under open-loop condition may not be apparent at all or may be obvious to a lesser degree. The reason is that under closed-loop control the current controller tends to regulate the output current thus masking the effect. The non-characteristic harmonics, however, are apparent. The proposed FICS algorithm results in a reduction in the number of non-characteristic harmonics yielding a lower THD compared to that of the fixed step-size approach. Figure 3.15 it is clear that the current waveform simulated using the proposed FICS algorithm at $\Delta t = 100\mu s$ appears closer to the one obtained using the fixed step-size algorithm at $\Delta t = 10\mu s$ than the one obtained using the fixed step-size

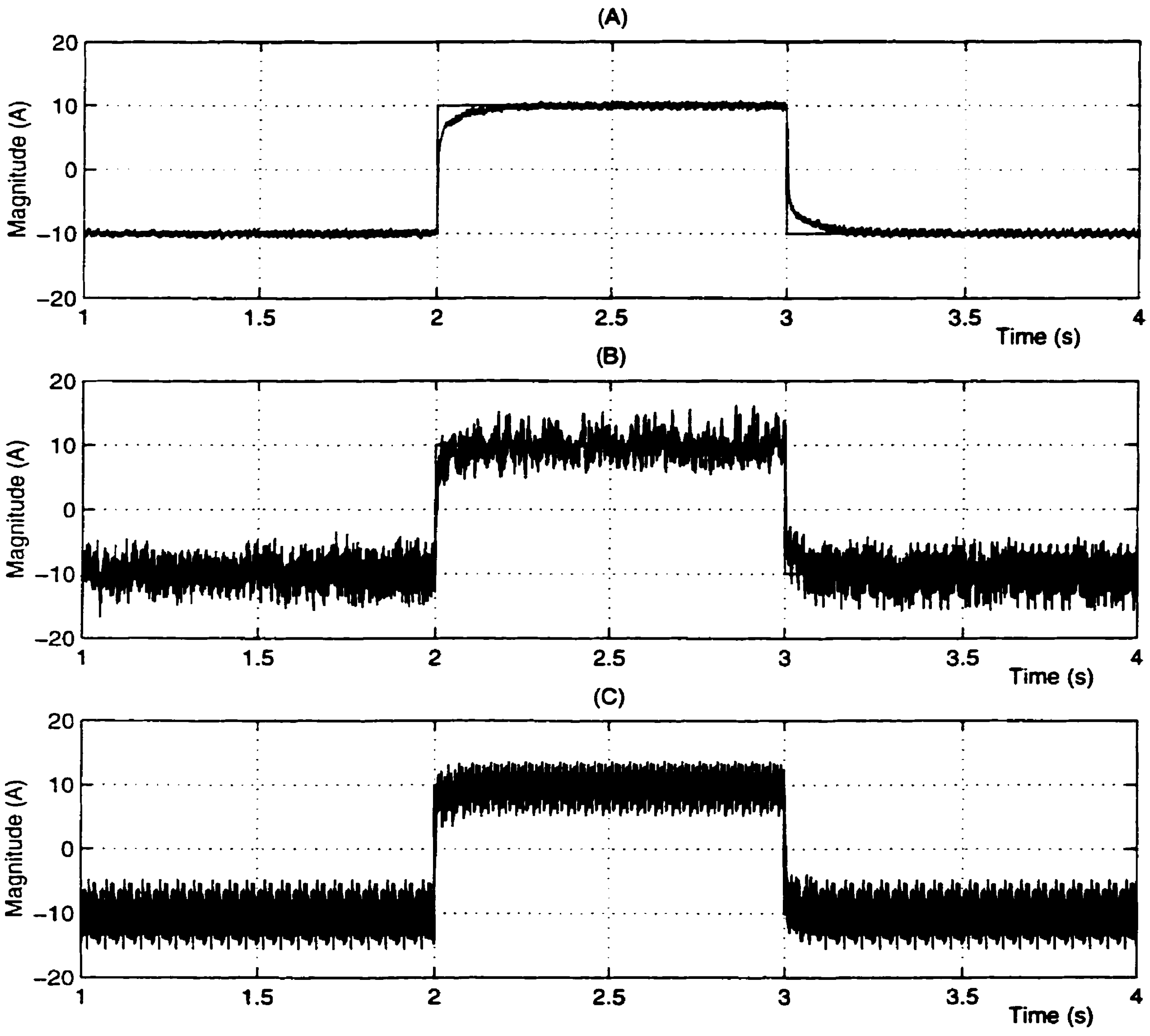


Figure 3.14: Step response in i_q simulated using the (A) Fixed step-size algorithm with $\Delta t = 10\mu s$ (B) Fixed step-size algorithm with $\Delta t = 100\mu s$ and (C) Proposed FICS algorithm with $\Delta t = 100\mu s$

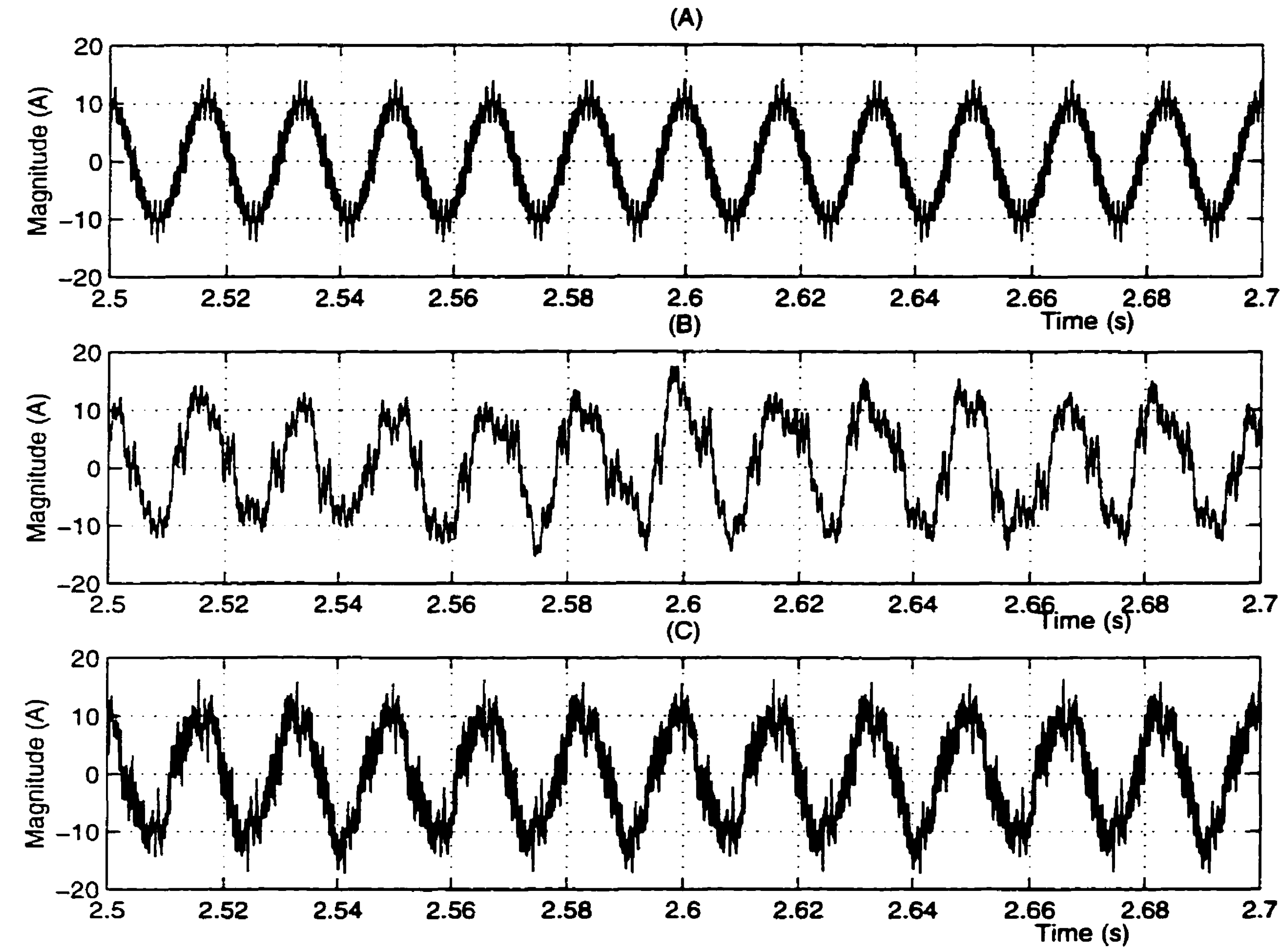


Figure 3.15: Current I_a under closed-loop control simulated using the (A) Fixed step-size algorithm with $\Delta t = 10\mu s$ (B) Fixed step-size algorithm with $\Delta t = 100\mu s$ and (C) Proposed FICS algorithm with $\Delta t = 100\mu s$

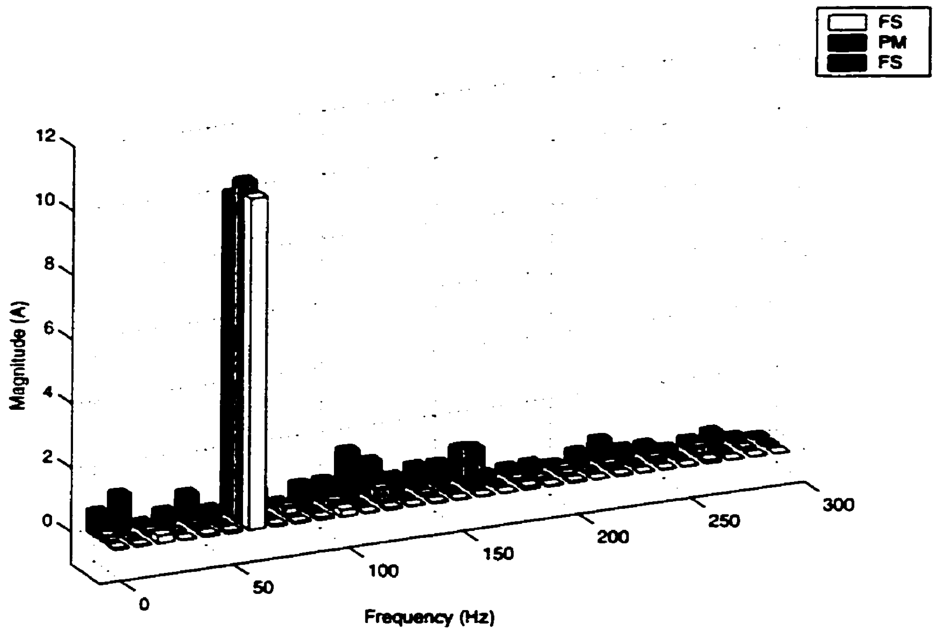


Figure 3.16: Frequency spectrum of I_a under closed-loop control. FS : Fixed step-size algorithm, PM : Proposed (FICS) algorithm. FS with $\Delta t = 10\mu s$ (THD=3.7%) and $\Delta t = 100\mu s$ (THD=40%), PM with $\Delta t = 100\mu s$ (THD=20%)

method at $\Delta t = 100\mu s$.

3.5.3 Execution Time

The off-line simulation programs using the fixed step-size and the proposed FICS algorithms were run on a $266MHz$ Intel Pentium processor running the RedHat 5.1 Linux operating system. Table 3.3 shows the CPU times measured for the two programs for a $1s$ simulation time. For $\Delta t = 50\mu s$ and $\Delta t = 100\mu s$ the execution time for the FICS approach is only 4% higher than that for the fixed step-size algorithm. To obtain similar accuracy the fixed step-size algorithm using $\Delta t = 10\mu s$ took $8.7s$ for the $1s$ run, a 5 to 8 fold increase in execution time. These results show that the time penalty for using linear interpolation and step-size variation as in the FICS algorithm is minimal compared with a reduction in the time step for the fixed step-size approach to obtain similar accuracy.

Δt (μs)	**Fixed step-size algorithm** (`nor_c11.c`)	**FICS algorithm** (`intrp_c11.c`)
10	8.68	8.84
50	1.88	1.96
100	1.05	1.09

Table 3.3: Execution time measured in seconds for a $1s$ simulation run on a Pentium $266MHz$ processor

3.6 Conclusions

The objective of this chapter was to provide modeling and control details of a three-phase PWM VSI system which was used as a study system for of-line time domain simulations in this chapter and will be used for designing the real-time simulator in Chapter 4. The VSI configuration proved to be suficiently complex to provide an insight into the switching processes without being excessively cumbersome for practical implementation. The contents of the chapter can be summarized as follows :

1. The derived inverter model is exact based on switching functions. The principle of operation of the VSI system was described giving the necessary details of PWM and the required coordinate transformations. The control of the VSI system was perormed in the synchronous $d-q$ frame using PI compensators.

2. The main objective during off-line time domain simulation was to emulate conditions encountered in the experimental setup. For that purpose the two major issues related to the practical implementation of digital controllers, namely the implementation of PWM and the controller dead-time were discussed. The choice of the simulation time step Δt was also discussed. On account of the traditional criterion of selecting Δt being inadequate for the simulation of power electronic circuits, a new criterion based on empirical evidence has been proposed. The details of the off-line simulation programs developed using the fixed step-size algorithm and the

proposed FICS algorithm (Section 2.4) were presented. Both off-line simulation programs were written in ANSI C language for ease of implementation on different computing platforms.

3. Off-line simulation results presented illustrate the performance of the FICS algorithm in comparison to the fixed step size algorithm. The outcome of increasing the simulation time step Δt translated into the inability of the fixed step-size method to account for inter-step switching events was illustrated with the following effects on the simulation output.

   - A significant change in the fundamental quantities.
   - An increase in non-characteristic harmonics.
   - Apperance of jitter in the control quantities.

   Based on a conservative estimate the FICS algorithm was found to yield at least a ten fold improvement in accuracy over the fixed step-size method using the same time step.

4. The execution time of the FICS algorithm was found to be comparable to that of the fixed step-size algorithm making it suitable for real-time computation.

Chapter 4

Real-time Digital Simulation and Experimental Verification of the PWM VSI System

4.1 Introduction

This chapter presents the design and implementation of a real-time digital simulator for the PWM VSI system. The objective of the design is to demonstrate the practical feasibility of the FICS real-time simulation algorithm. The design of the simulator is based on UHP40 (Universal High Performance) digital processing platform developed in the Power Group of the Department of Electrical and Computer Engineering at the University of Toronto. The UHP40 platform consists of the TMS320C40 DSP (Digital Signal Processor) and the EPF8000 FPGA (Field Programmable Gate Array) as the major processing elements. For real-time simulation of any dynamic system the simulation task must be completed within a given time increment. Given a hardware computing system the simulation algorithm is thus limited by what can be implemented in that time interval. Therefore, the design has to be properly tailored to meet the constraints of the available hardware resources and to meet the simulation objectives. The design in this chapter follows a systematic modular approach which makes full use of the computing resources available on the UHP40 platform. An experimental setup of the PWM VSI system was built to verify the real-time simulation. A detailed description of the experimented setup and its digital controller is presented.

The chapter begins with a brief overview of the UHP40 platform. Thereon, the hardware and software design and implementation details are thoroughly described. Finally, a comparison of results from the real-time simulation and the experimental setup, under similar operating conditions, is presented.

4.2 Overview of the UHP40 Digital Processing Platform

The structure of the UHP40 platform is shown in Figure 1. The main hardware and software features of the platform are described below.

4.2.1 Hardware of UHP40

4.2.1.1 TMS320C40 DSP

The main processing unit on the UHP40 platform, it is a 32 bit floating point DSP from Texas Instruments. It has two separate busses : a local bus which connects to static RAM modules (Banks 1 and 2) and a global bus which is connected to other peripheral devices on the UHP40 board. This processor is responsible for executing the user program, written in C, as an interrupt service routine in real-time.

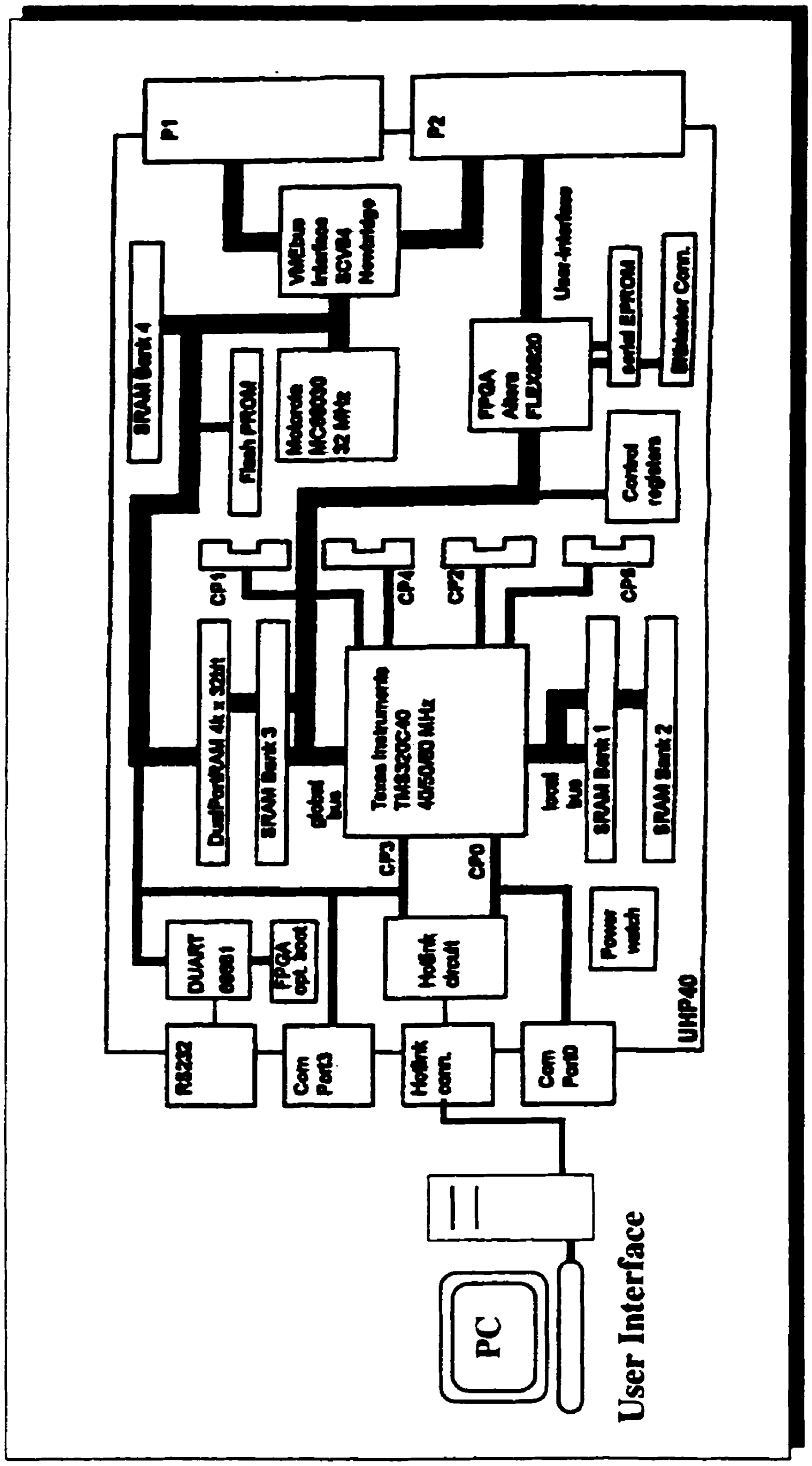


Figure 4.1: Structure of digital processing platform UHP40

4.2.1.2 EPF8000 FPGA

The FPGA used on the UHP40 is a FLEX8000 device from ALTERA and its internal architecture resembles a matrix structure of macro cells which are the basic computational units. Each cell comprises of a flip-flop and a look-up table in which any combination of logic primitives of four input variables can be stored. In total the FPGA on UHP40 has 672 macrocells and shares the global address and data space of the DSP. A global bus interface has to be programmed in the FPGA for communication with the DSP. The hardware logic design for the FPGA can be carried out using ALTERA's MAXPLUS [31] software.
The FPGA can be programmed, while prototyping using the BitBlaster hardware from ALTERA by connecting it to the serial interface of the PC. Once a design is finished it can be programmed into the EPROM so that the FPGA is automatically booted up after each power up. This device is usually employed as a gating pattern generator when the UHP40 is used as a controller.

4.2.1.3 Comports

The Comports provide 8 bit data transfer paths in and out of the UHP40. Comports 1,2,4 and 5 can be accessed through four on-board connectors. These are mainly used for connecting A/D boards or connecting other UHP40's in a multi-DSP system. Comports 0 and 3 are used by the ATM Interface Hotlink [31] circuit for two way communication between the UHP40 and the PC.
The UHP40 also contains a Motorola CPU MC68030 along with a Newbridge SCV64 chip which consists of a complete VME bus interface. This facility was however not used in the present design of the real-time simulator.

4.2.2 Software for UHP40

The software requirements for the UHP40 include :

1. A user program
2. DSP Kernel [31] program
3. DSP Monitor [31] program

The user program and the DSP Kernel program run on the DSP while the Monitor program runs on the PC. Once a user program has been developed, it is linked with the kernel program during compilation to produce an executable code which can be downloaded to the DSP through the Hotlink [31] cable. The user program consists of two functions, `user_function()` and `user_init()`. The `user_function()` contains the main simulation/control code that the user wants to run in real-time. The `user_init()` contains the initialization variables and a pointer to the `user_function()`. For initialization the Kernel program calls the `user_init()` function.

User programs typically require several signals to be monitored or parameters and switches to be set etc. This requires a two way communication between the UHP40 board and the PC. This communication is managed by the Kernel program on the DSP end while the Monitor program takes care of the PC end. Once initialized the Kernel program mainly performs two operations :

1. In the service mode it transfers signals back and forth between the UHP40 and the PC.
2. On receiving an interrupt programmed by an on-board timer, it executes the `user_function()`. The timing of the interrupt is set by the user and determines the time step of a simulation or the sampling period of a control algorithm.

The Monitor program running on the PC emulates a real-time oscilloscope. With its help the user can not only display the signals internal to the DSP but also change the parameters of the system on-line while the real-time simulation/control is in progress. Further details on the UHP40 hardware or software can be found in reference [31].

4.3 Design of the Real-time Digital Simulator and Controller

This section describes the implementation details of a real-time digital simulator for the VSI system described in Chapter 3. The design is based on the UHP40 digital processing platform described in Section 4.2. First an overview of the entire design is presented. Thereafter, the hardware and software features of the design are systematically described.

4.3.1 Overall Design

In order to implement a real-time digital simulator for the VSI system using the simulation approach detailed in Chapter 2 four major modules are needed.

1. System Simulator Module
2. Digital Controller Module
3. PWM Module
4. Switching Event Capture (SEC) Module

The System Simulator module is where the mathematical equations describing the system are solved in real-time. The Digital Controller implements the control algorithm. The PWM module generates the firing signals for the VSI and the SEC module determines the timing of those signals. Two UHP40 boards can be employed for this design as shown

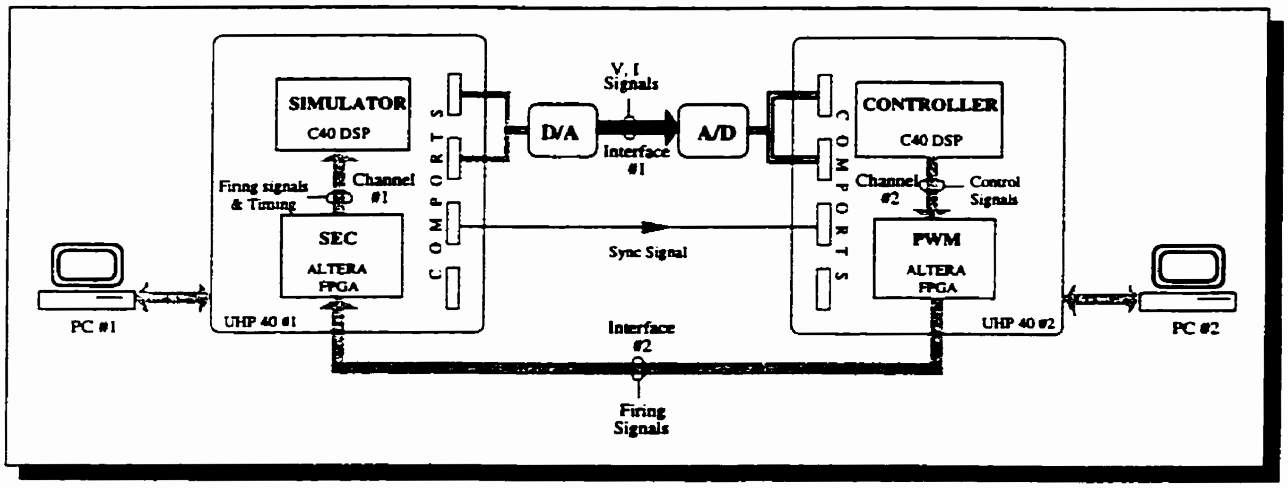


Figure 4.2: Real-time digital simulator and controller setup for the VSI system

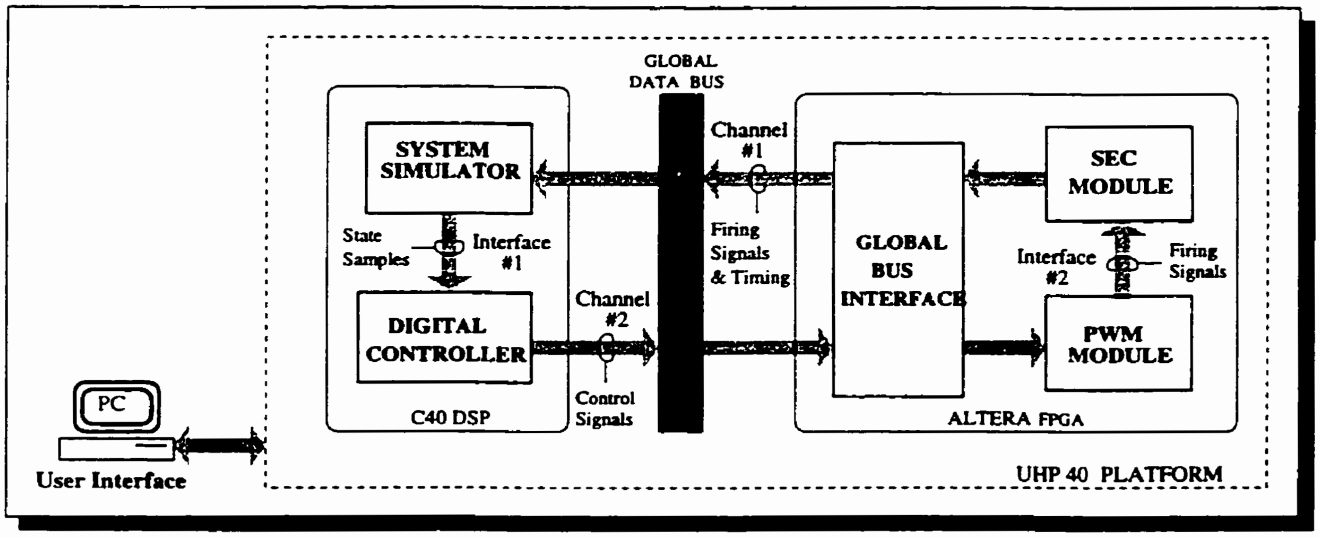


Figure 4.3: Final design of the real-time digital simulator and controller for the VSI system

in Figure 4.2. UHP40 #1 consists of the System Simulator and the SEC Modules while UHP40 #2 houses the Controller and the PWM modules. Each UHP40 is monitored by a separate PC. The C40 DSP on UHP40 #1 is configured as the System Simulator while the C40 DSP on UHP40 #2 serves as the Controller. Although the PWM and SEC modules can also be implemented on the DSP's, however, these functions are assigned to the FPGA's to minimize the computational overhead on the DSP's. Since PWM is originally part of the control algorithm, it is therefore assigned to the FPGA on UHP40 #2 while the SEC module is configured in the FPGA on UHP40 #2. The two UHP40 boards have two main data transfer interfaces : Interface #1 comprises of the D/A and A/D converters to transfer the system samples to the controller while Interface #2 simply serves to transfer the firing signals from the PWM module to the SEC module. In addition to the two interfaces a synchronizing signal **sync** is needed between the two boards to align their respective time grids. In this design although externally the data transfer is bidirectional between the two boards, internally on each UHP40 board the data transfer is unidirectional (Channels 1 and 2). On UHP40 #1 the data transfer is from the FPGA to the DSP while it is in the opposite direction on UHP40#2.

Owing to several hardware constraints the final design, however, has been optimized, without any loss of functionality, to resemble Figure 4.3. This design comprises of one UHP40 platform and one PC. The Simulator and the Controller modules are still configured in the DSP while the PWM and SEC modules are assigned to the FPGA. Interface#1 is now internal to the DSP and serves the same purpose of ferrying system samples from the simulator to the controller while Interface #2 is internal to the FPGA. In contrast to the design of Figure 4.2, the on-board data transfer (Channels 1 and 2) between the DSP and the FPGA is now bi-directional managed by the Global Bus Interface on the FPGA. Comparing Figures 4.2 and 4.3 it is apparent that none of the original functions have been compromised. The final design offers the following two advantages :

1. It eliminates additional D/A and A/D hardware. This has the salutary effect of not introducing the two non-ideal effects related to signal conversion : quantization of the signal values and the finite conversion time delay.
2. All operations can be easily synchronized without the need for the sync signal thereby eliminating unnecessary design complexity.

4.3.2 Digital Hardware Design for the FPGA

The digital hardware design for the FPGA in the real-time simulator setup shown in Figure 4.3 was carried out using the schematic design method of the Maxplus software from Altera to allow easy reconfiguration for different implementations. The main graphic design file is **design16n.gdf**. It consists of three subdesigns for :

1. Global Bus Interface (**16busint.gdf**)
2. Pulse Width Modulator (**16pwm.gdf**)
3. Switching Event Capture (**16capture.gdf**)

All design files can be found in the Appendix C.

4.3.2.1 Global Bus Interface

The primary function of this module is to arbitrate bi-directional data transfer (Channels 1 and 2 in Figure 4.3) between the DSP and the FPGA. In addition to the communication signals it provides Δt and T_s signals to the PWM and SEC modules to synchronize their respective operations.

For data transfer from the controller module in the DSP to the PWM module in the FPGA, the control values are first clocked into a set of registers whenever the DSP finishes its calculation and sends them. A second set of registers takes the control values from the first set at the beginning of each sampling period T_s. This operation involves a delay of one T_s and represents the controller dead time described in section 3.4.2. A watchdog macro function makes sure that the control values coming in from the DSP are refreshed every T_s.

The data transfer from the SEC module in the FPGA to the Simulator module in the DSP utilizes a tri-state buffer where the gating signals and their timing are stored before being transferred to the DSP at the beginning of each time step Δt.

4.3.2.2 Pulse Width Modulator

Sinusoidal pulse width modulation is implemented in this sub-design. A modified triangular carrier waveform synchronized with the controller sampling period T_s is compared with the control value sent from the Controller module in the DSP. Gating pulses are generated at the intersection of the carrier and control waves. The carrier wave is represented by a 16 bit counter value; it cannot be directly accessed as a signal.

At the beginning of every T_s the counter is pre-loaded with a positive/negative limit. The counter starts counting down/up till the opposite limit is reached when it holds the count for $4\mu s$. The hold operation creates a flat top on the carrier wave and determines the minimum pulse width of the resulting gating signal. At the next sampling instant T_s an opposite limit is pre-loaded and the counter counts in the other direction till the limit is reached when it again holds for $4\mu s$. The counter limit u is initialized in the DSP program and is given as

$$u = T_s \star f_{clock}/4 \tag{4.1}$$

where $T_s = 1/f_{sw}$ and $f_{clock} = 20MHz$. The gating pulses are stored in an output buffer in the Global Bus Interface and transferred to the Simulator module at the beginning of every Δt. The output pins of the FPGA are accessible for the user to examine the firing signals using an oscilloscope.

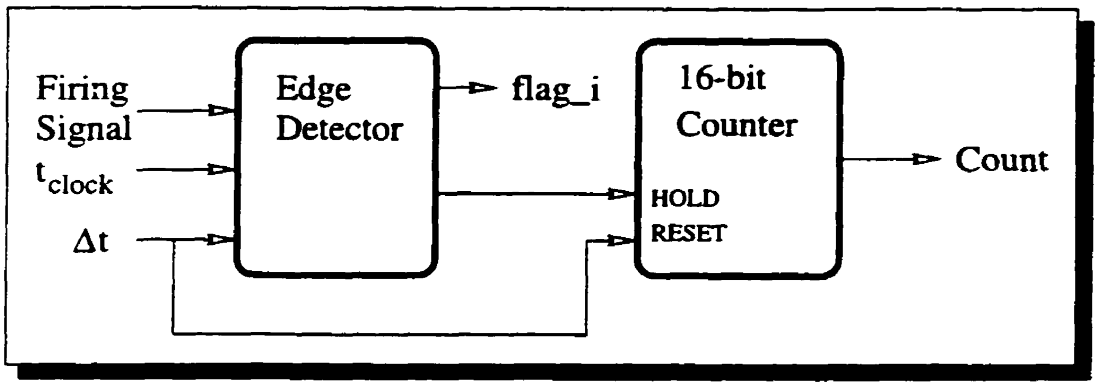


Figure 4.4: Switching Event Capture circuit

4.3.2.3 Switching Event Capture

The purpose of this module is to put a time stamp on the firing pulses coming from the Modulator. It registers the location of the firing pulses with respect to the beginning of the sampling instant T_s. The capture mechanism simply consists of an edge detection circuit and a 16 bit counter as shown in Figure 4.4. Figure 4.5 shows the hierarchy of time scales T_s, Δt, and t_{clock}. The resolution of the counter is $50ns/count$ based on the system clock frequency of $20MHz$. Every rising edge of the Δt signal the counter is reset and starts counting up. When a switching event is detected by the edge detector, the `flag_i` is set and a **hold** signal is sent to the counter. The counter value and the `flag_i` are held in an output buffer in the Global Bus Interface till the end of the time period Δt. They are sent to the Simulator module in the DSP at the beginning of the next simulation time step. Based on the state of the firing pulses from the PWM module and the counter value and `flag_i` from this module, the simulator can then determine the occurrence and location or any event in the previous time step.

4.3.3 Software Design for the C40 DSP

The software program for real-time simulation and control of the VSI system is `rts_110.c` and can be found in Appendix D. This code is executed as an interrupt service routine

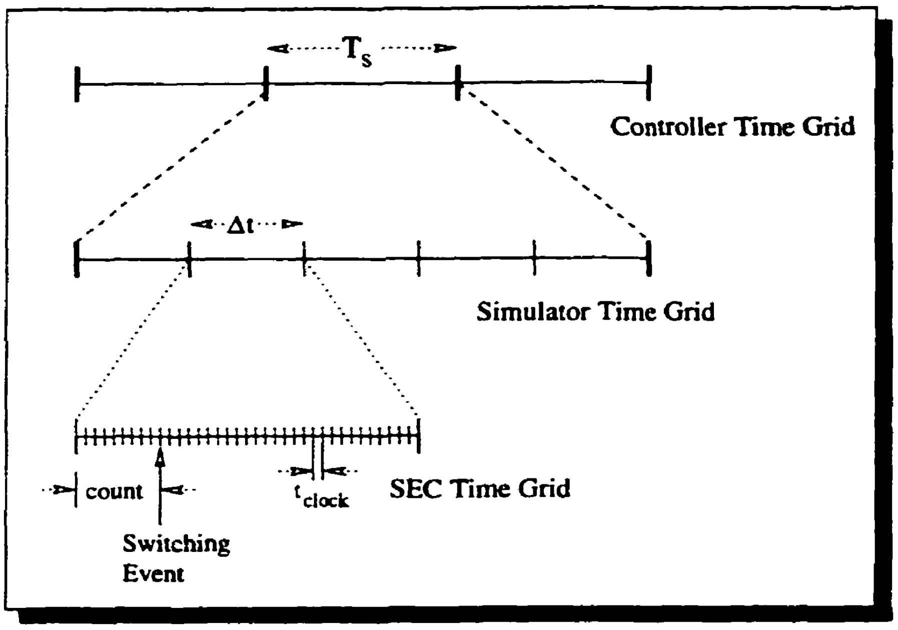


Figure 4.5: Hierarchy of time scales for the real-time simulator

`intfunc1` on the DSP at every time step Δt. Once initialized the real-time simulation can run indefinitely on the DSP. The user has full control of all the parameters of the simulation and can view the simulation variables using the Monitor program on the PC. The user can alter the system parameters or initiate transients on-line and see how the system evolves over time. The program is divided into two parts : the System Simulator Module and Controller Module. The real-time program structure is shown in Figure 4.6. This diagram only shows the sequence of operations carried out on the simulation time grid, their exact timing is given later in this chapter.

4.3.3.1 System Simulator

The code for this module performs three major functions. Firstly, it acquires the gating pulses and their timing information stored in the Global Bus Interface in the FPGA at the beginning of each time step Δt. Secondly, based on the gating signals the states of the switches in the inverter are derived. Then, depending on whether a switching event has been detected (indicated by **`flag_i`**) in the previous time step one of the two network

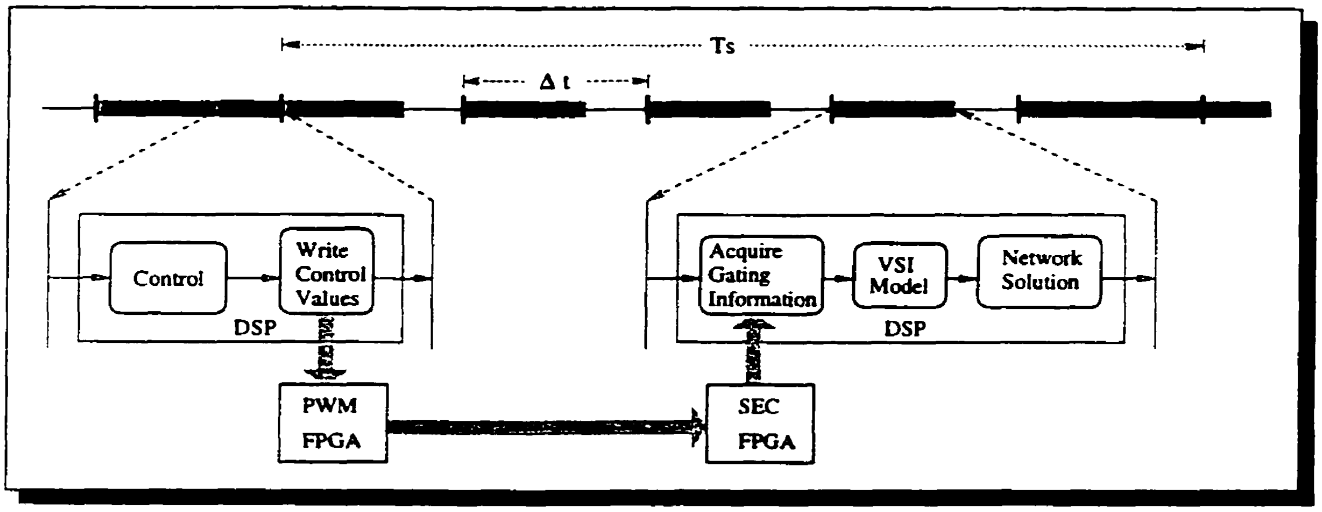


Figure 4.6: Real-time program structure

solution paths, described in Section 2.4, is undertaken. Once every T_s the simulator transfers voltage and current signals to the Controller module. This operation emulates Interface #1 in Figure 4.3.

Initialization The real-time simulator is initialized by a user defined function `user_init()`. This function specifies the initial values of the system and control variables, time step Δt, parameters, and PWM information such as frequency, resolution, etc. It also specifies the FPGA address space, memory addresses where control values can be written and addresses from where the gating information can be obtained; the address space may even be specified in the user interrupt function `intfunc1`. In addition, look-up tables are created for one period of sinusoidal functions which serve as sources in the simulation. Since power frequency ($60Hz$) voltage sources are periodical they need to be calculated for only one period, stored in a look-up table and applied at every subsequent cycle.

4.3.3.2 Controller

The code for this module is executed every T_s and performs the following three functions : acquisition of system samples from the Simulator module, control algorithm,and transfer of sinusoidal modulated control values to the FPGA. The control algorithm is the same as the one used for off-line simulation and is shown in Figure 3.6.

4.4 Experimental Setup

To verify the real-time simulation, a laboratory prototype of the VSI system was built. Figure 4.7 shows the experimental setup. The system parameters can be found in Appendix B. The VSI is composed of six IGBT switches with anti parallel diodes. The IGBTs are rated for a constant current of 25 A and a maximum DC voltage of 300V. The digital processing platform UHP40 described in section 4.2 is used as the digital controller. The control algorithm is executed on the C40 DSP while the PWM is implemented on the FPGA. Data acquisition of six signals (3 voltages and 3 currents) is done by dedicated A/D boards connected to the Comports on the UHP40. The user has full control of the system using the Monitor program running on the PC.

4.4.1 Digital Hardware and Software Design for the Controller

The FPGA hardware design used for the experimental setup is very similar to the one used for the real-time simulator. The top graphic design file is `con_fpga.gdf` which consists of three subdesigns :

1. Global Bus Interface (`16busint.gdf`)
2. Pulse Width Modulator (`16pwm.gdf`)
3. Dead Time Insertion Module (`gating.gdf`)

All design files can be found in the Appendix C. The Global Bus Interface is the same as the one in Section 4.3.2 with the exception that now the data transfer is the unidirectional flow of control values from the DSP to the FPGA resulting in a simpler design. The Pulse

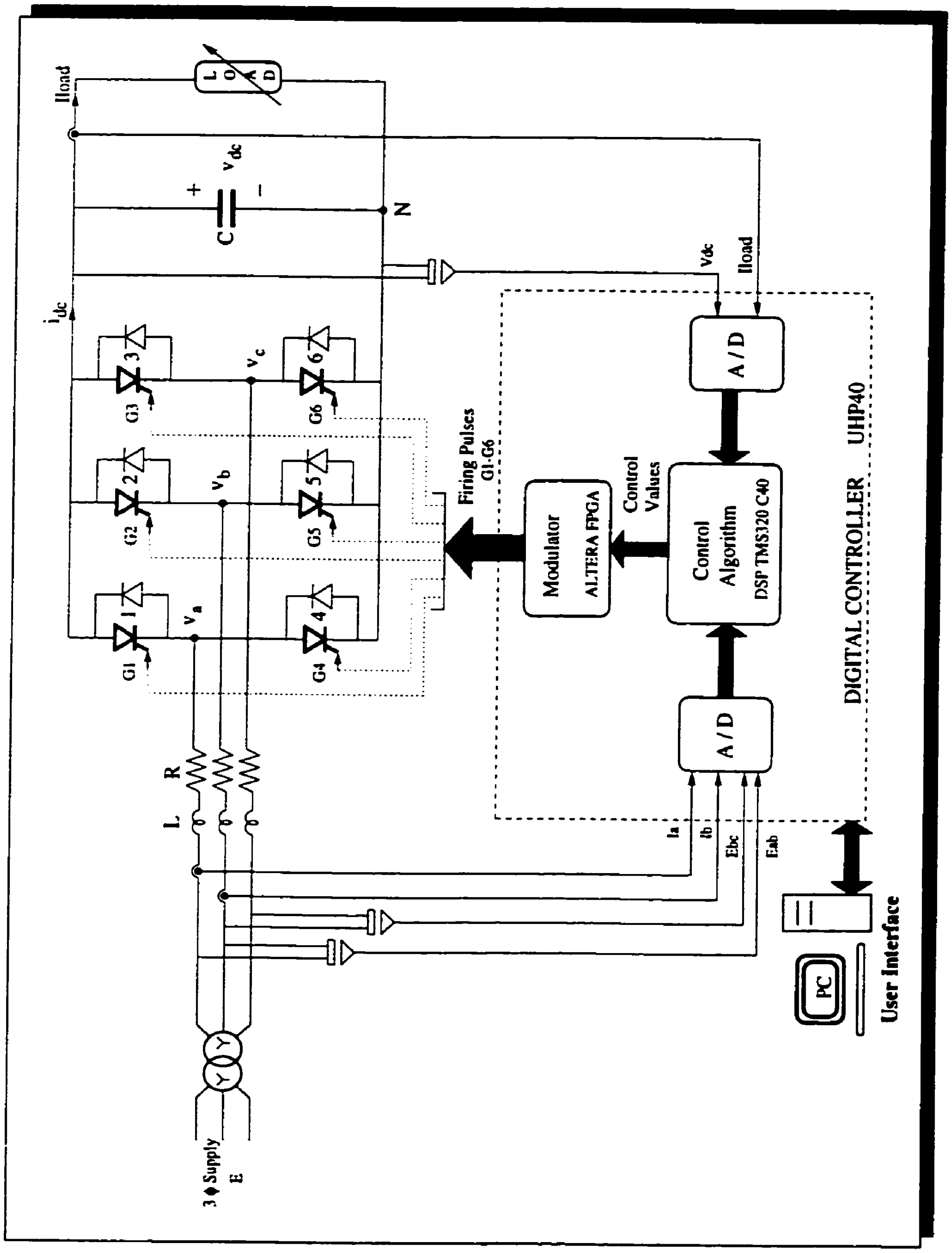


Figure 4.7: Experimental setup for the VSI system

Width Modulator which forms the central part of the overall FPGA design is identical to the one used for the real-time simulator. The Dead Time Insertion module inserts a $2\mu s$ dead time T_d between the gating signals of the upper and lower switches of each inverter leg in order to prevent a short circuit of the DC bus. During the dead time both switches are in the off position. The counter limit u for the Modulator is altered to include the dead time T_d and is initialized in the DSP program as :

$$u = \frac{(T_s - 2 * T_d) * f_{clock}}{4} \tag{4.2}$$

The Dead Time Insertion module is the only major difference between the FPGA designs for the experimental setup and the real-time simulator setup.

The software program executed on the DSP to control the VSI system is `con_110.c` found in the Appendix DThe control algorithm is identical to the one used for the real-time simulation with the exception of some extra code for signal acquisition using the A/D converters.

4.5 Real-time Simulation and Experimental Results

This section presents the results form the real-time simulation as well as the experimental setup of the PWM VSI system. Both open loop and closed loop control tests were conducted. For direct comparison the real-time simulation was carried out under the same operating conditions as those for the experimental setup. The digital control algorithm is identical in both cases. In one case the gating signals were sent to the physical setup while in the other they were sent to the simulated system. The real-time simulator employs the FICS simulation algorithm proposed in Section 2.4. A $100\mu s$ time step was chosen for the real-time simulation. This choice was governed by the capability of the C40 DSP.

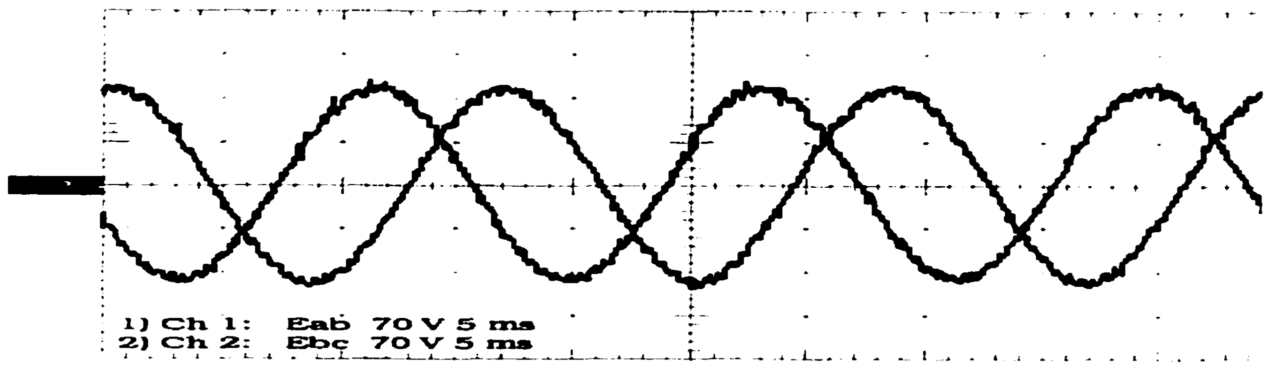


Figure 4.8: Steady state AC source voltages e_{ab} and e_{bc} - Experiment ($110V_{ll-rms}$)

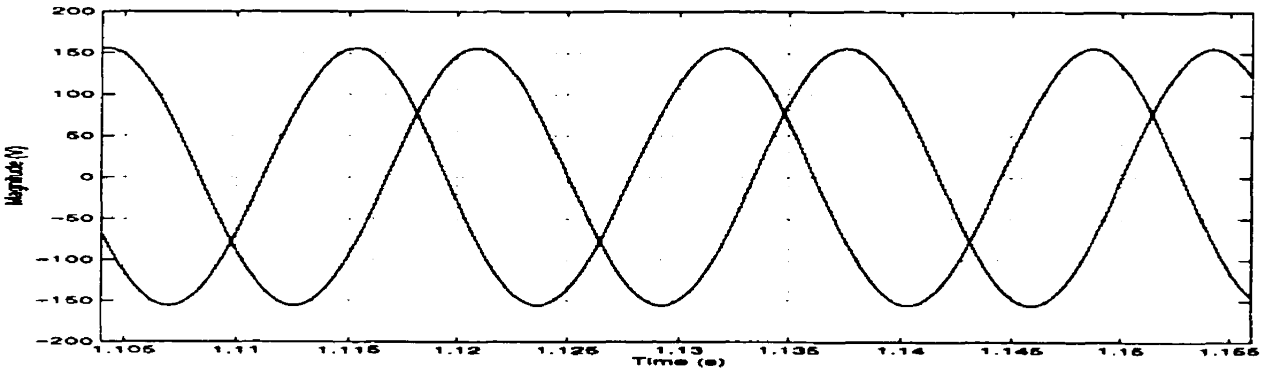


Figure 4.9: Steady state AC source voltages e_{ab} and e_{bc} - RTS ($110V_{ll-rms}$)

A smaller time step could not be chosen under the present design conditions.

Figures 4.8 through 4.29 show two sets of results, one from the real-time simulation and the other from the experiment. Under open loop conditions with $m_a = 0.8$ and $\delta = 20^\circ$, steady state waveforms of all circuit voltages and currents were observed. Fourier analysis of these waveforms is presented to demonstrate their close agreement. Under closed loop conditions, step responses of the i_q and v_{dc} controllers are shown in figures 4.23 through 4.29. The closed loop results demonstrate adequate tracking and stability in both sets of results. An off-line time domain simulation with $\Delta t = 10\mu s$ was also carried out under the same operating conditions and the results were found to be in good agreement with the experimental and the real-time simulation results. In addition to the fact that the adopted real-time simulation algorithm only approximates the behavior of

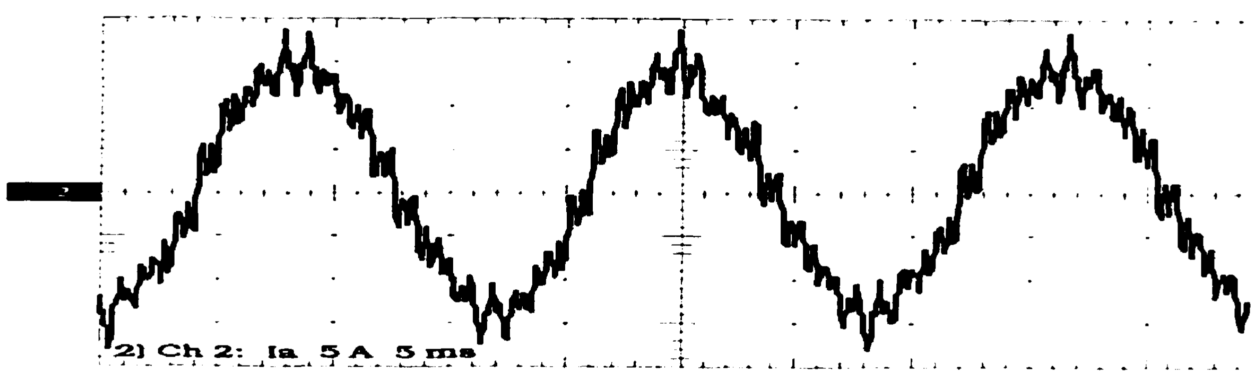


Figure 4.10: Steady state current i_a under open loop control-Experiment

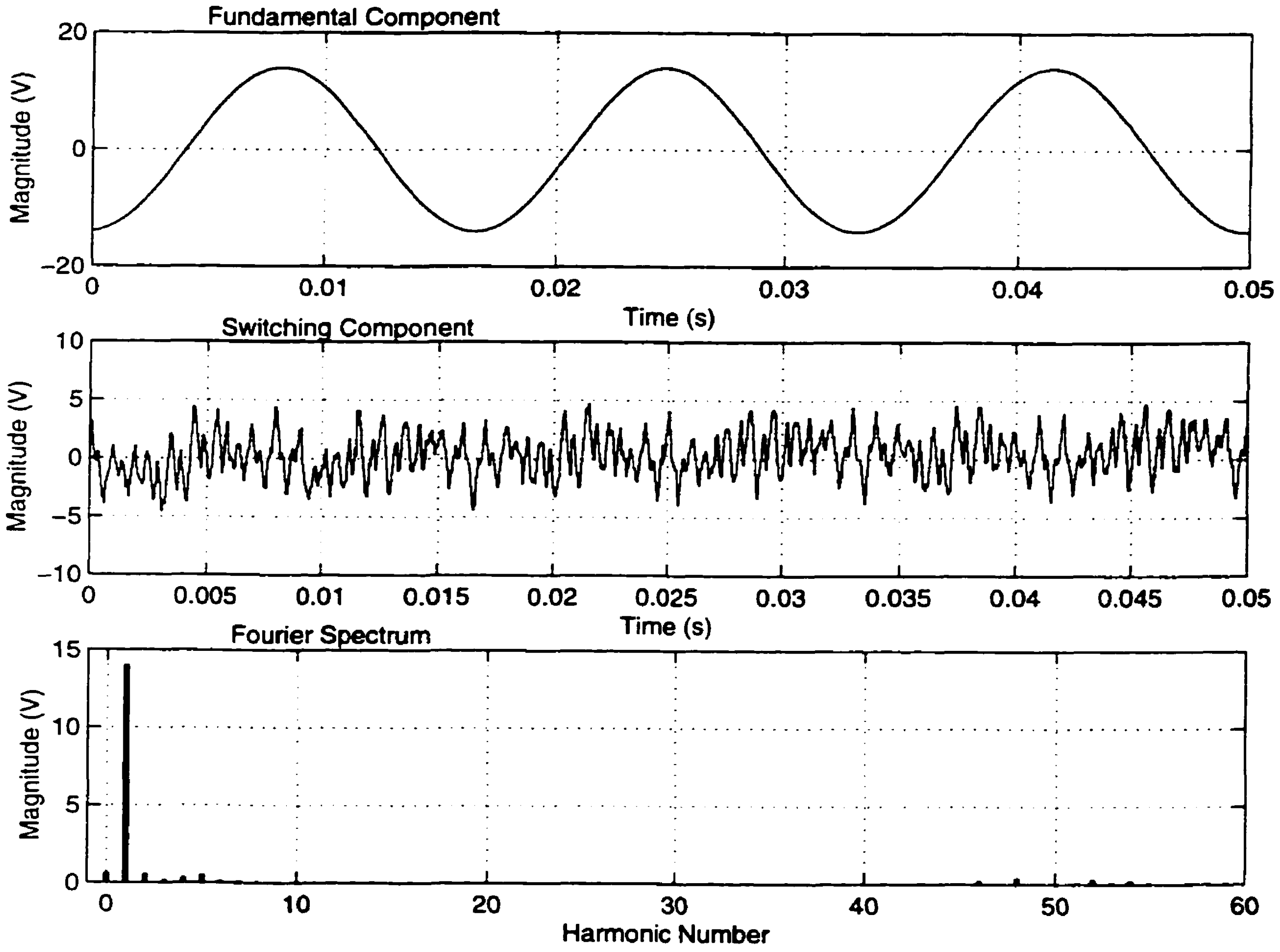


Figure 4.11: Fundamental, switching and fourier spectrum ofi_a (I_{a1} = 14.0A, THD = 7.0%)

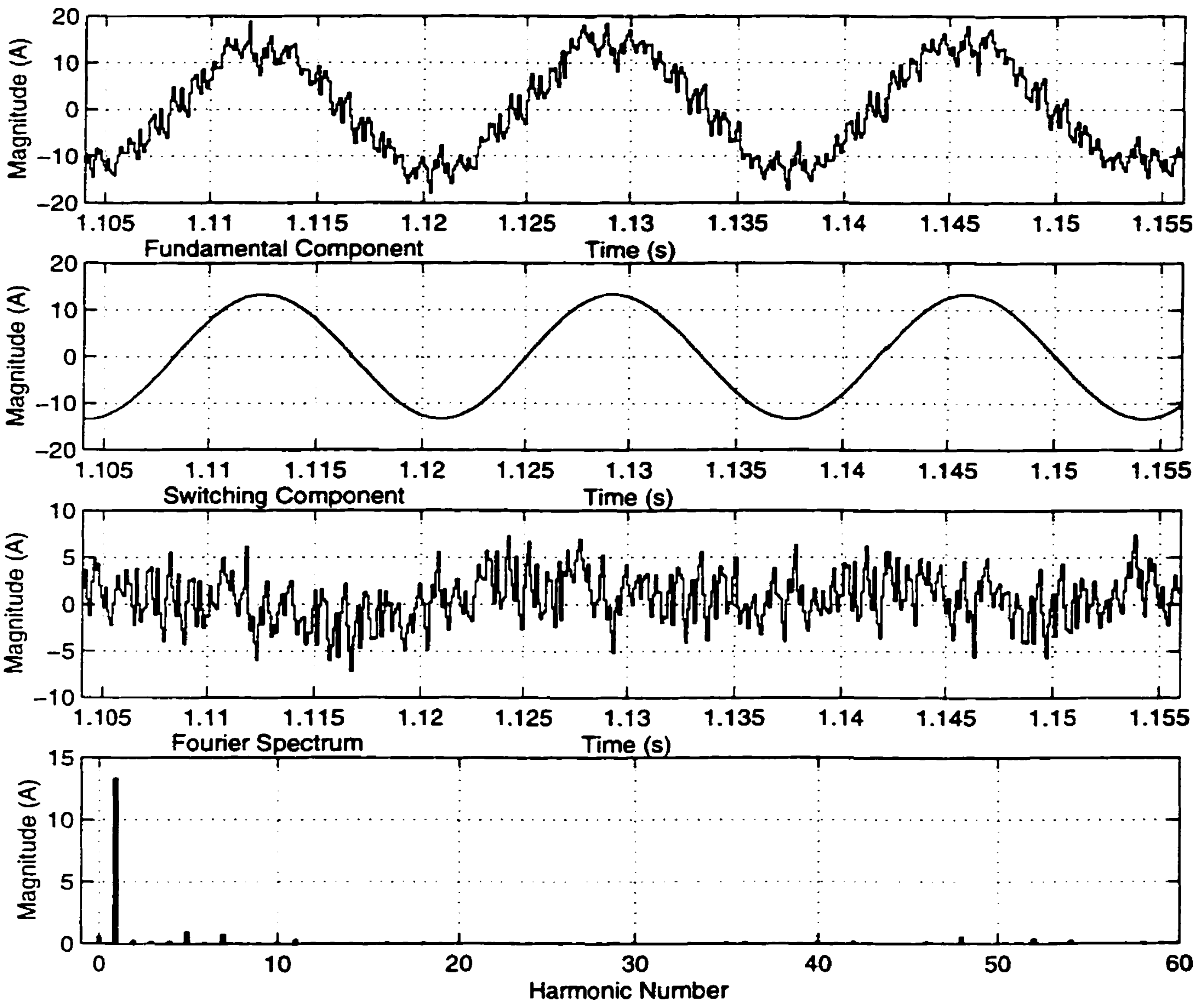


Figure 4.12: Steady state current i_a under open loop control (RTS) (I_{a1} =13.3A, THD = 11%)

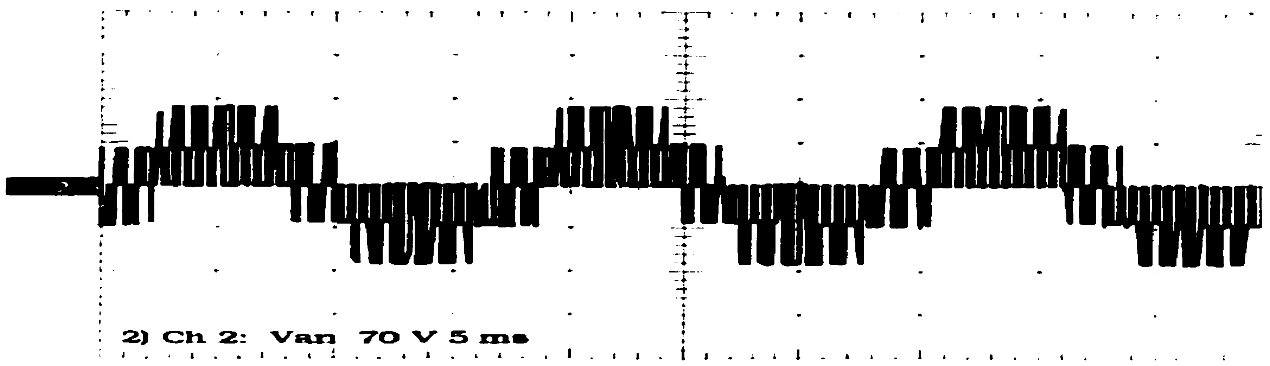


Figure 4.13: Steady state voltage v_{an} under open loop control (Experiment)

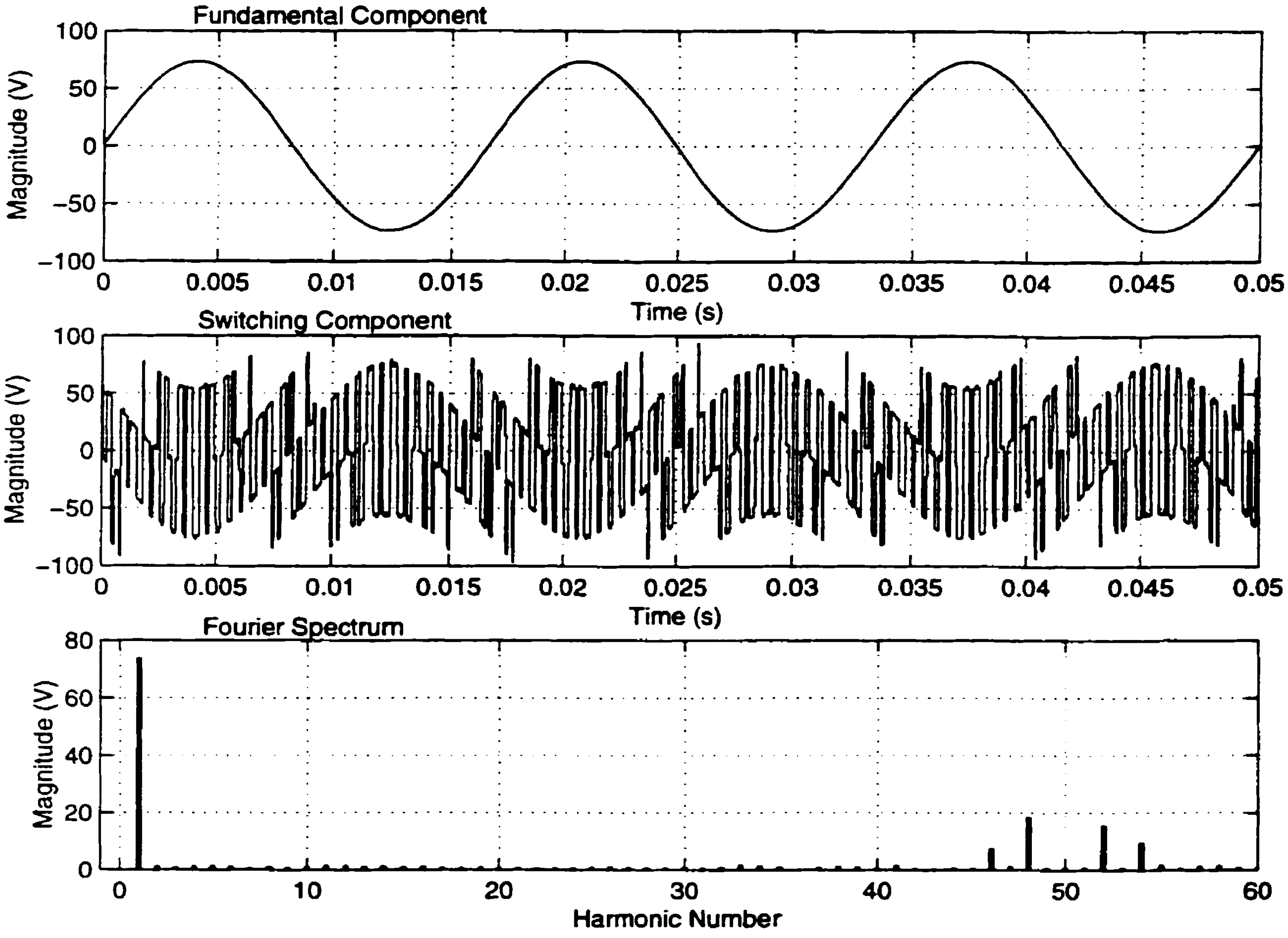


Figure 4.14: Fundamental, switching and fourier spectrum of v_{an} (V_{an1} = 73.5V, THD = 37.5%)

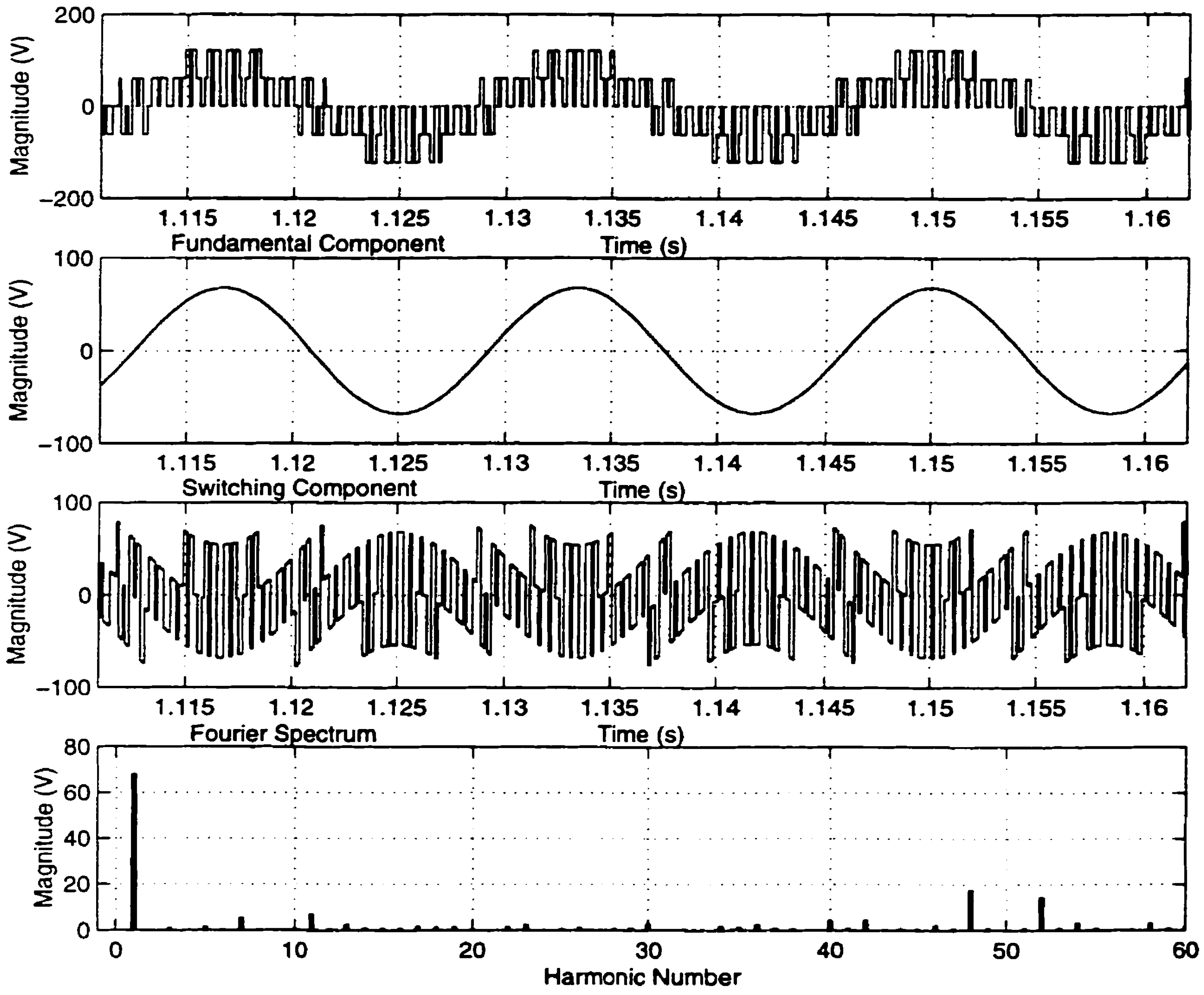


Figure 4.15: Steady state voltage v_{an} under open loop control (RTS) (V_{an1} = 68V, THD = 39.6%)

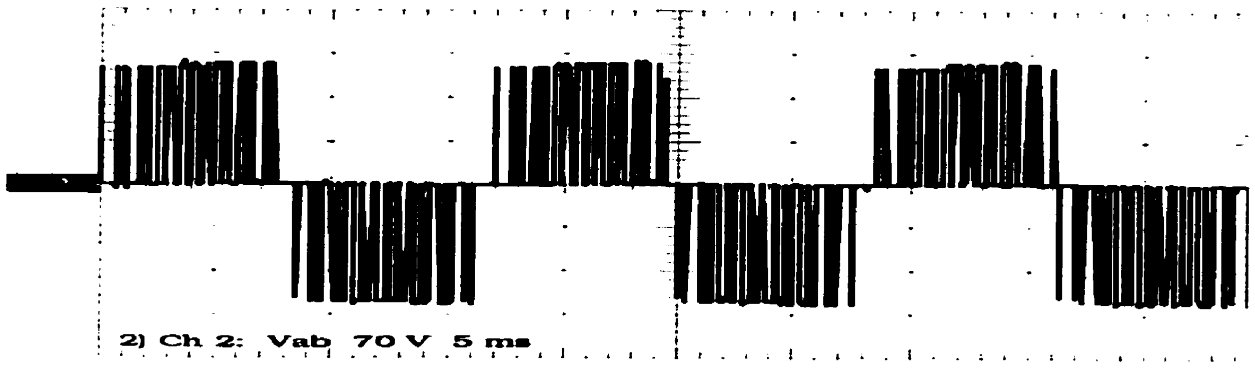


Figure 4.16: Steady state voltage v_{ab} under open loop control (Experiment)

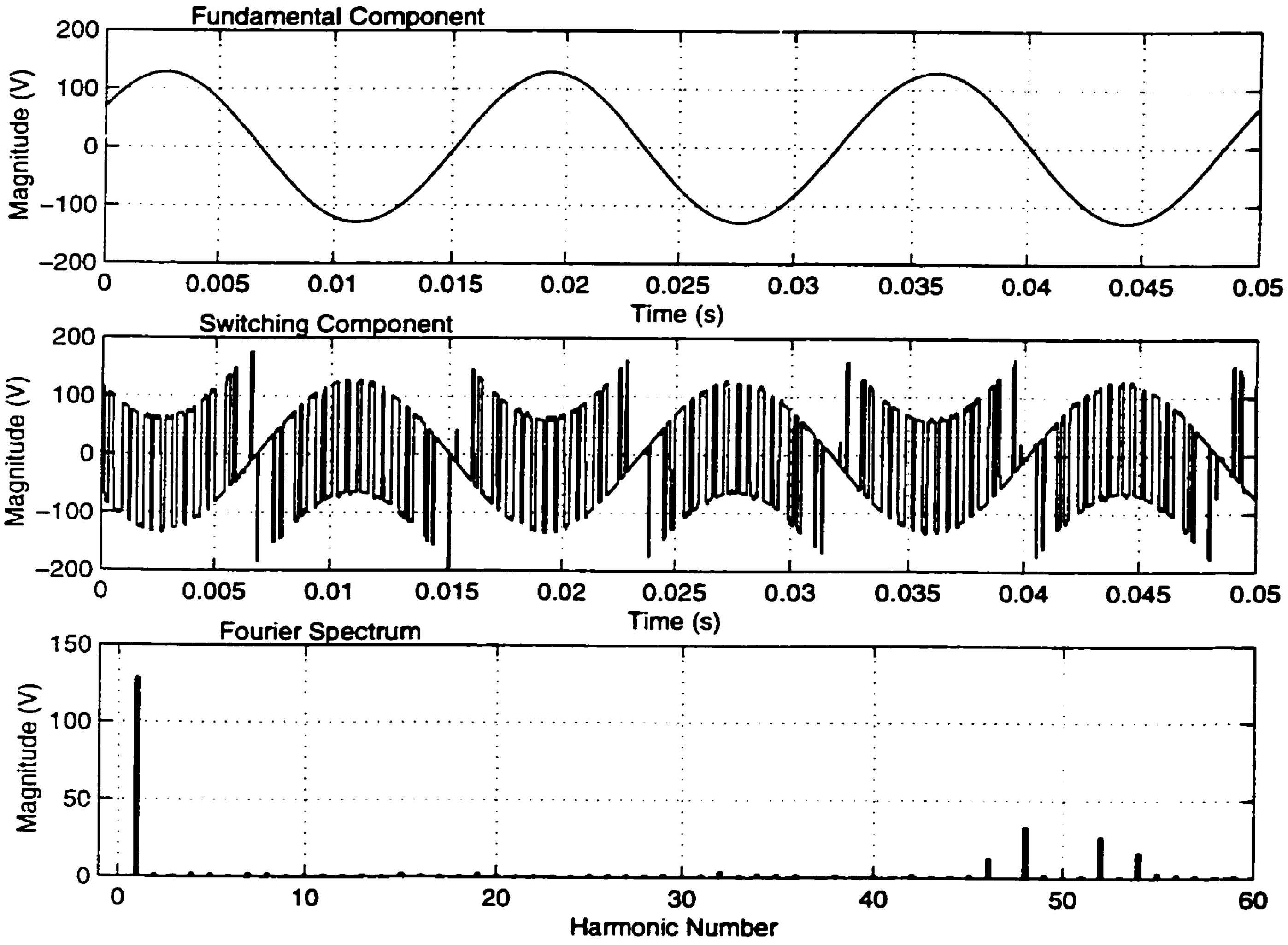


Figure 4.17: Fundamental, switching and fourier spectrum of v_{ab} (V_{ab1} = 128.8V, THD = 37.5%)

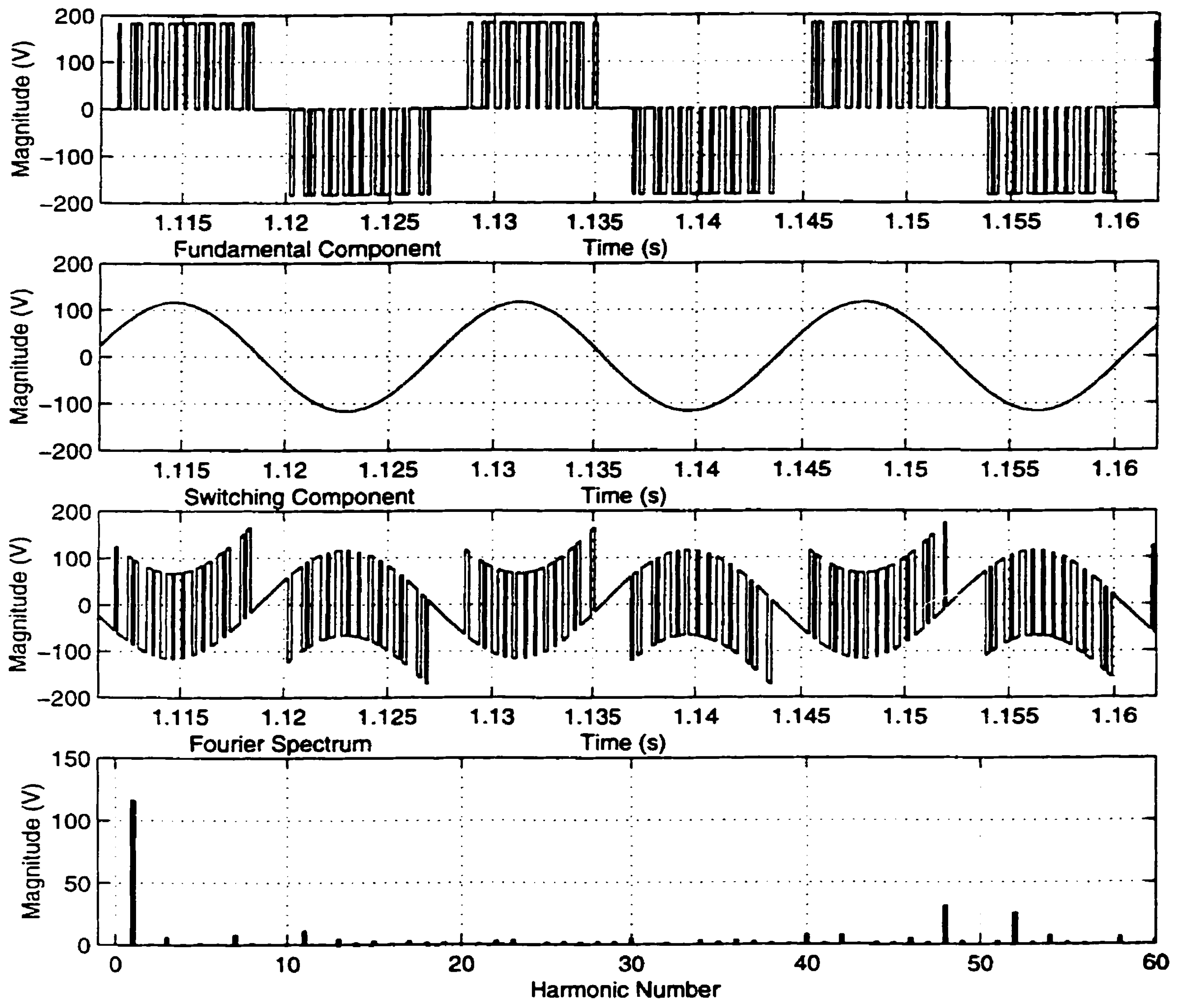


Figure 4.18: Steady state voltage v_{ab} under open loop control (RTS) (V_{ab1} =116.2V, THD = 40.6%)

a fixed step-size algorithm with a small Δt, the difference in the two sets of results can be attributed to the following factors :

1. In the real-time simulator switches in the VSI are modeled ideal and switches in the same inverter leg were turned on/off simultaneously. The real IGBT switches have finite turn-on and turn-off times and there is a $2\mu s$ deadtime when both switches in one leg are off simultaneously. This effect could not be simulated due to the difficulty of accounting a $2\mu s$ shift in the pulses on a time step of $100\mu s$. For low switching frequencies it is harmless to ignore this effect but for higher switching frequencies when the deadtime becomes comparable to the switching period, this effect can no longer be ignored. Although the PWM dead-time could have been compensated in the experimental setup such an effort was precluded by limited FPGA resources.

2. Asynchronous PWM i.e., $m_f = f_{sw}/f_1$ is a non-integer, is used for the experimental setup. To implement synchronous PWM the switching frequency f_{sw} must be made variable as the line frequency f_1 changes to keep the ratio m_f an exact integer. Since f_{sw} is synchronized to the controller sampling period T_s that would mean changing T_s on-line. However, changing T_s was not feasible due to a fixed sampling period which needs to be specified during the DSP initialization. Although the PWM implemented for the real-time simulator was also asynchronous the sources were considered ideal and frequency was fixed at $60Hz$. Whereas in the experimental setup the line frequency had constant variations of small amounts which could have variable effects on the results.

3. Unbalance in the system voltages and parameters or stray inductances and capacitances could also contribute to a little offset in the experimental results.

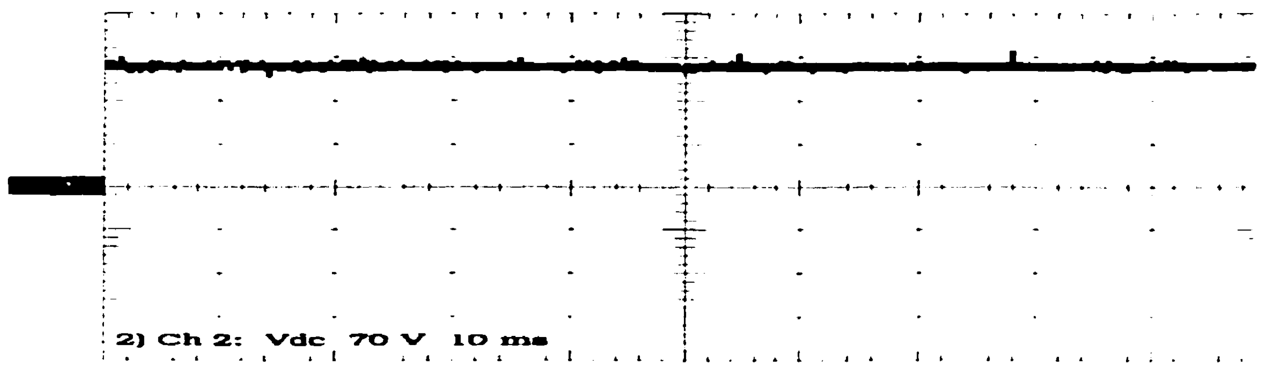


Figure 4.19: Dc link voltage v_{dc} under open loop control (Experiment) ($V_{dc-av} = 191$V)

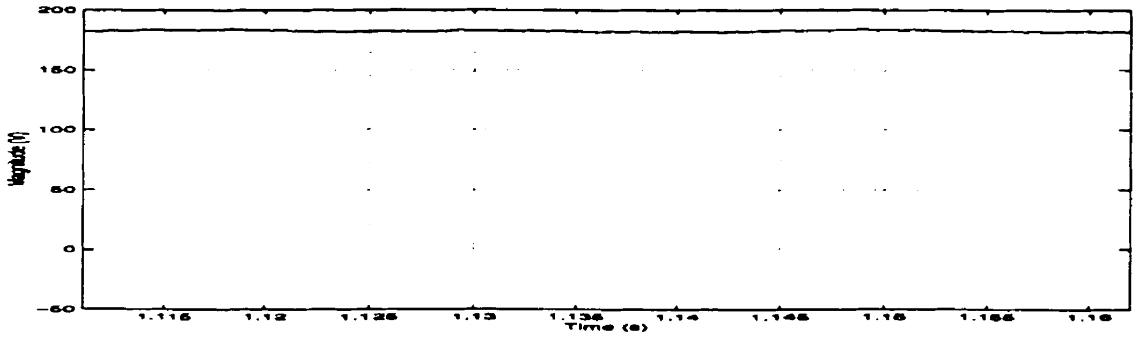


Figure 4.20: Dc link voltage v_{ab} under open loop control (RTS) ($V_{dc-av} = 183$V)

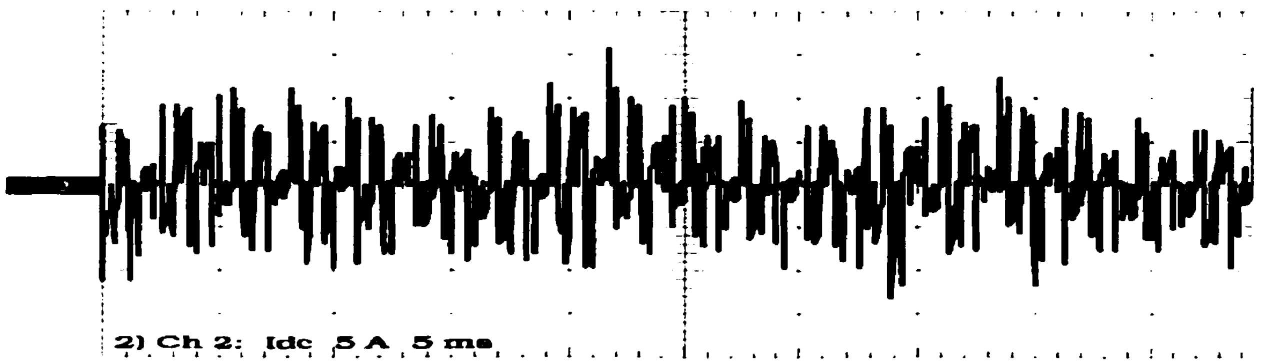


Figure 4.21: Steady state dc link currenti_{dc} under open loop control (Experiment)

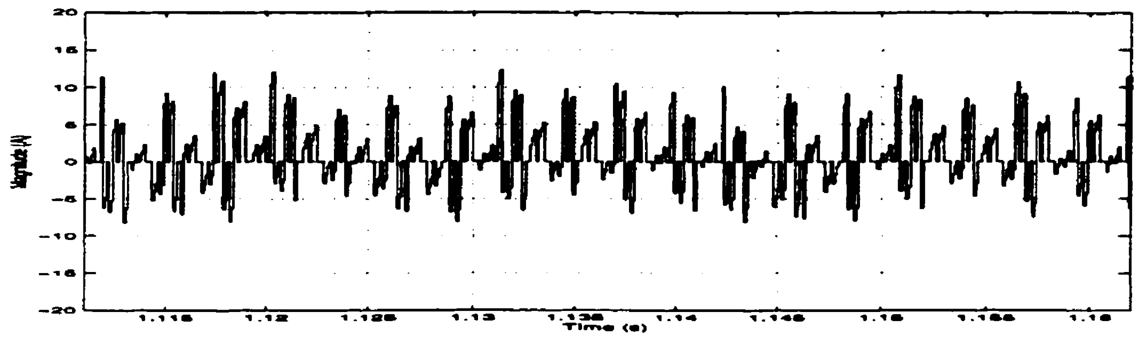


Figure 4.22: Steady state dc link current i_{dc} under open loop control (RTS)

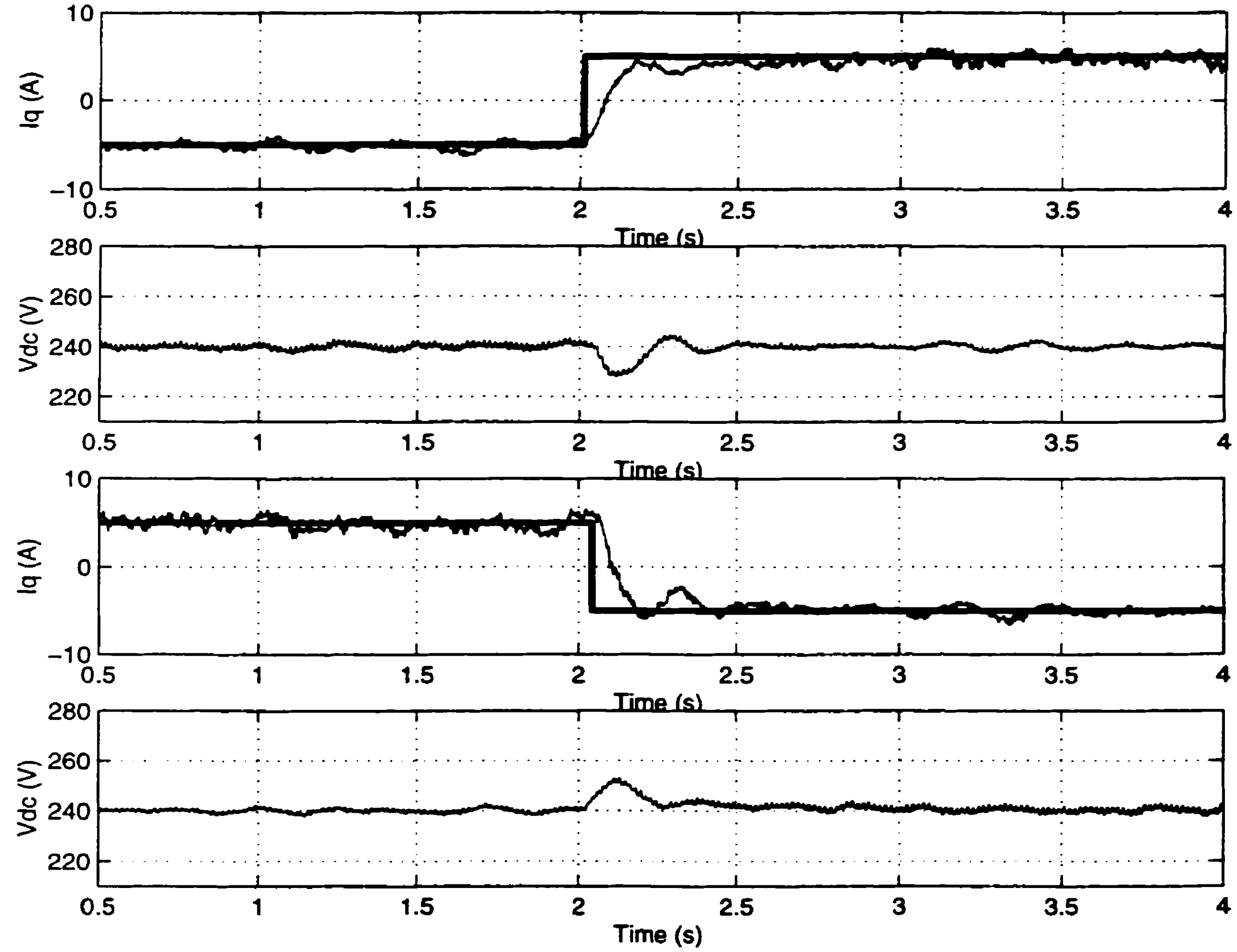


Figure 4.23: Step response of the i_q current controller (Experiment)

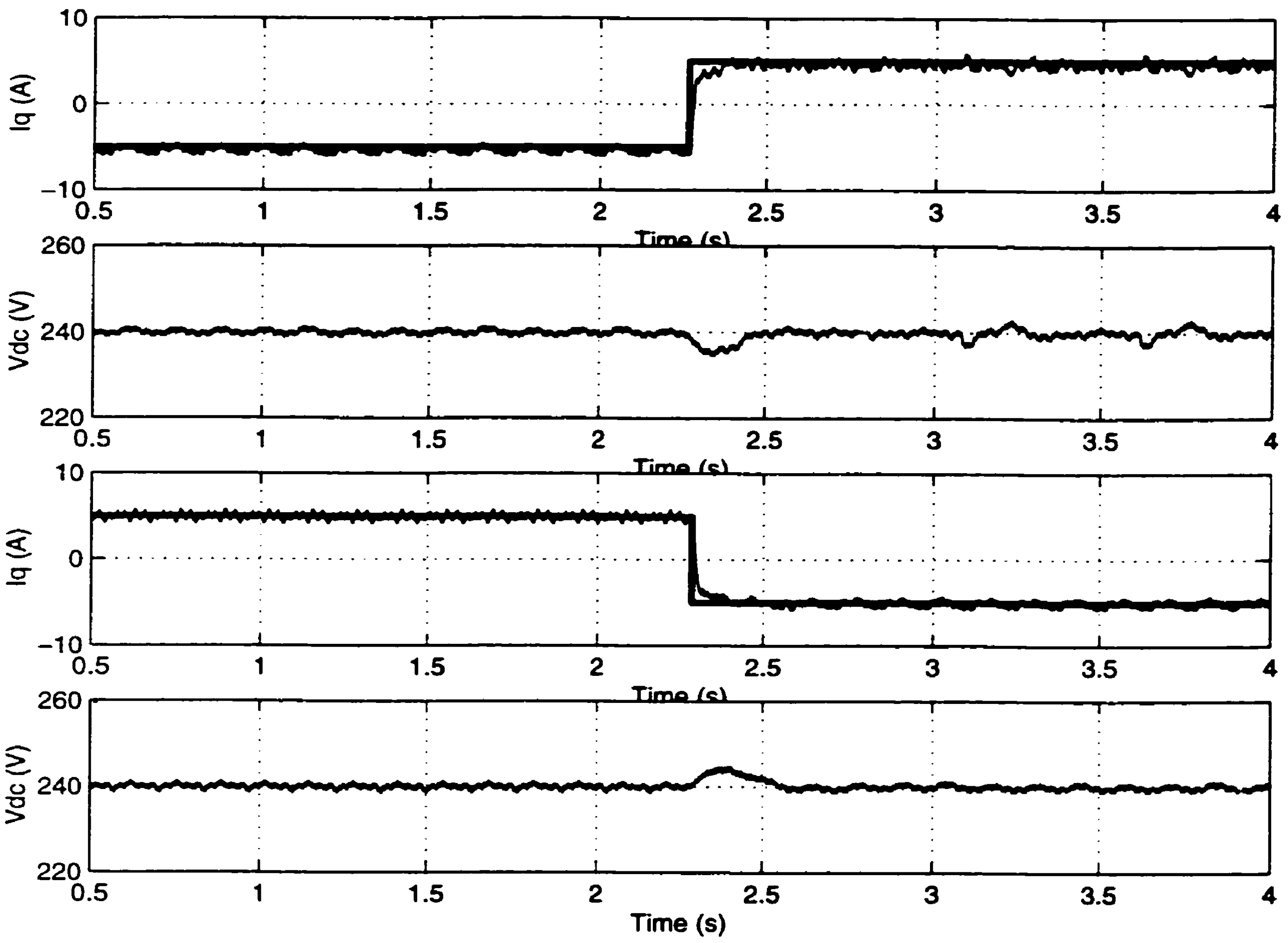


Figure 4.24: Step response of the i_q current controller (RTS)

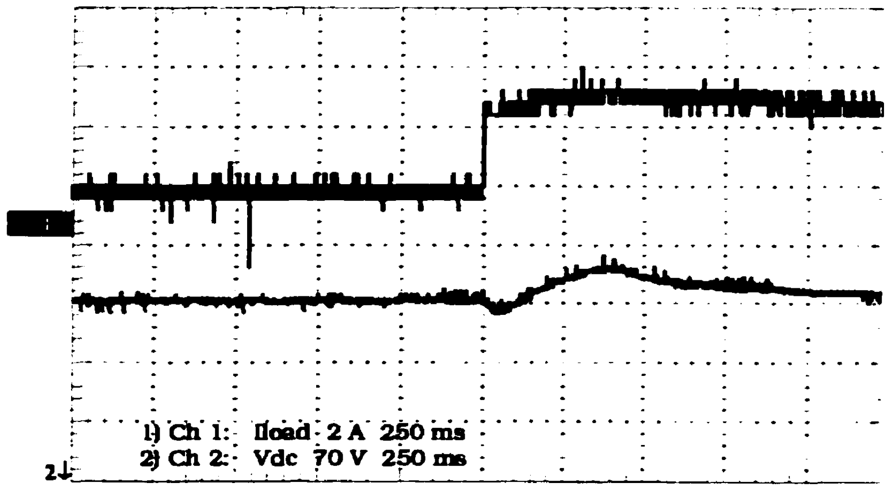


Figure 4.25: Dc side transient with 3A load switched on (Experiment)

es

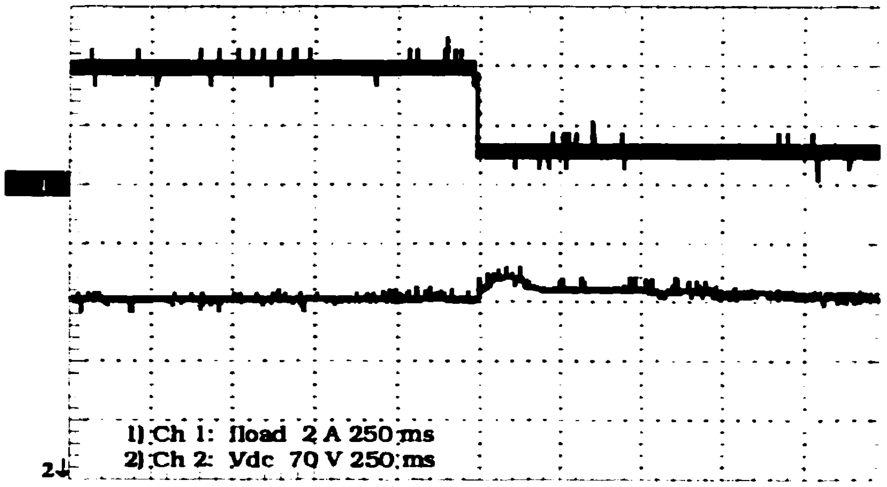


Figure 4.26: DC side transient with 3A load switched off (Experiment)

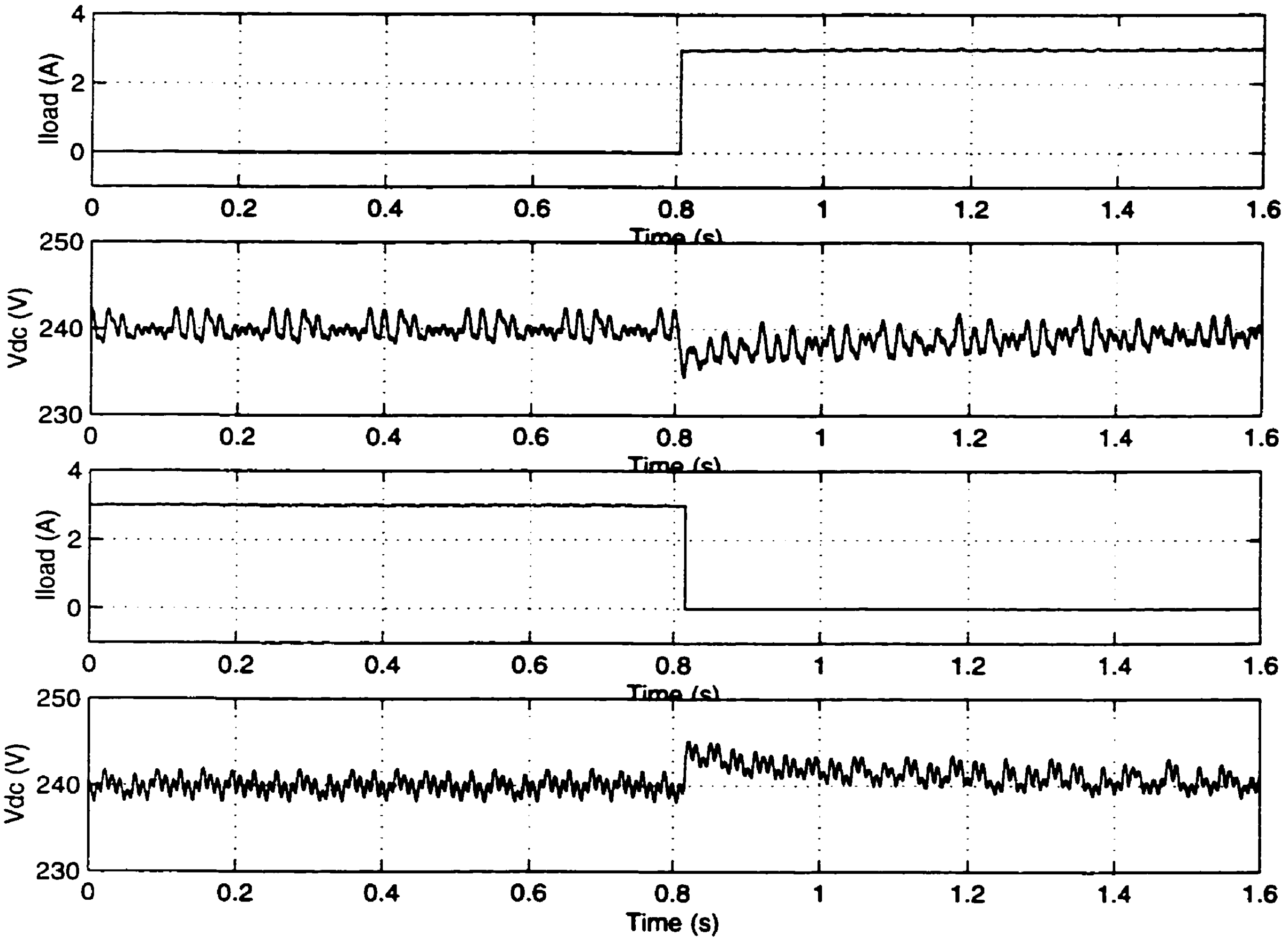


Figure 4.27: DC transient with 3A load switched on and off (RTS)

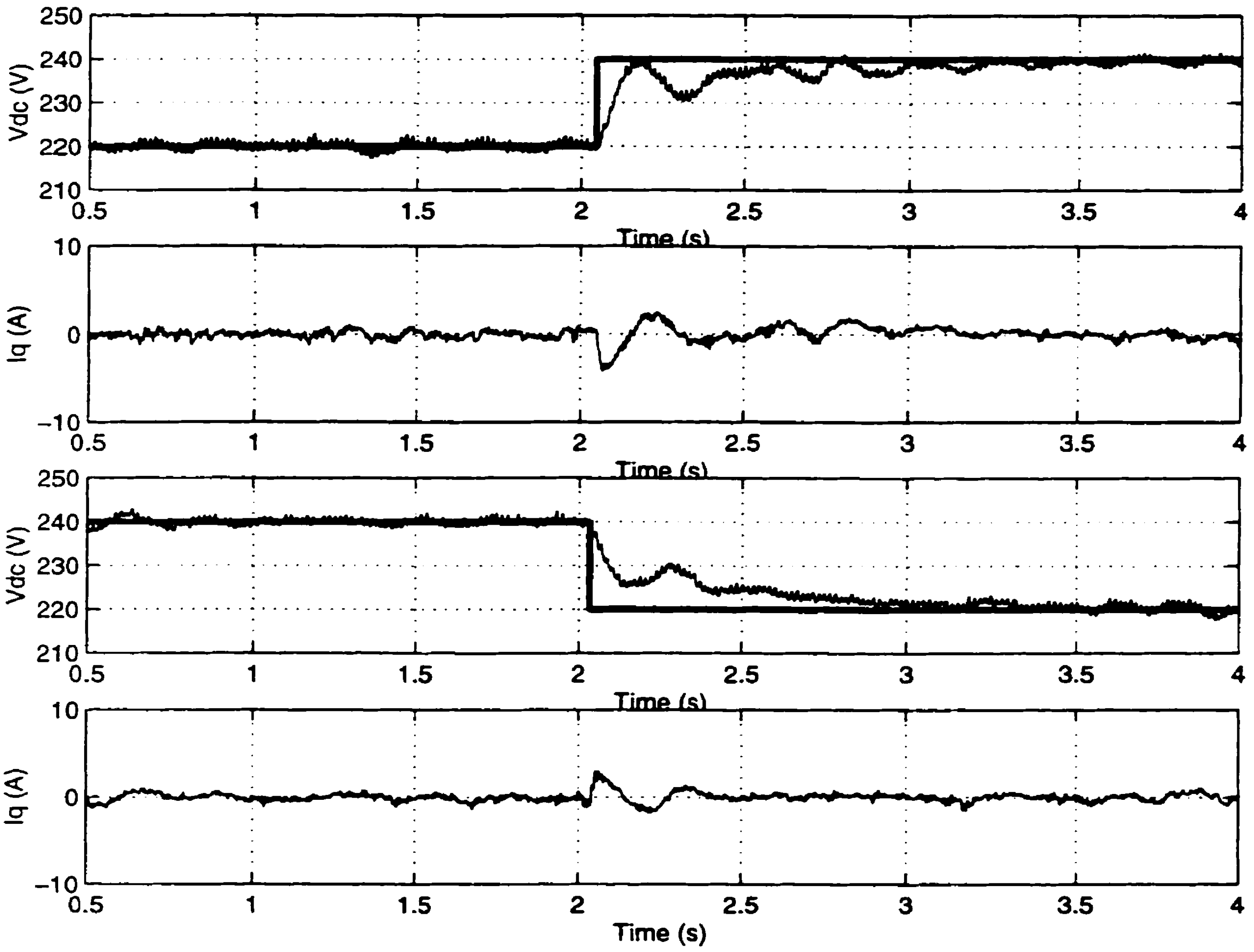


Figure 4.28: DC side transient showing step response of the v_{dc} controller (Experiment)

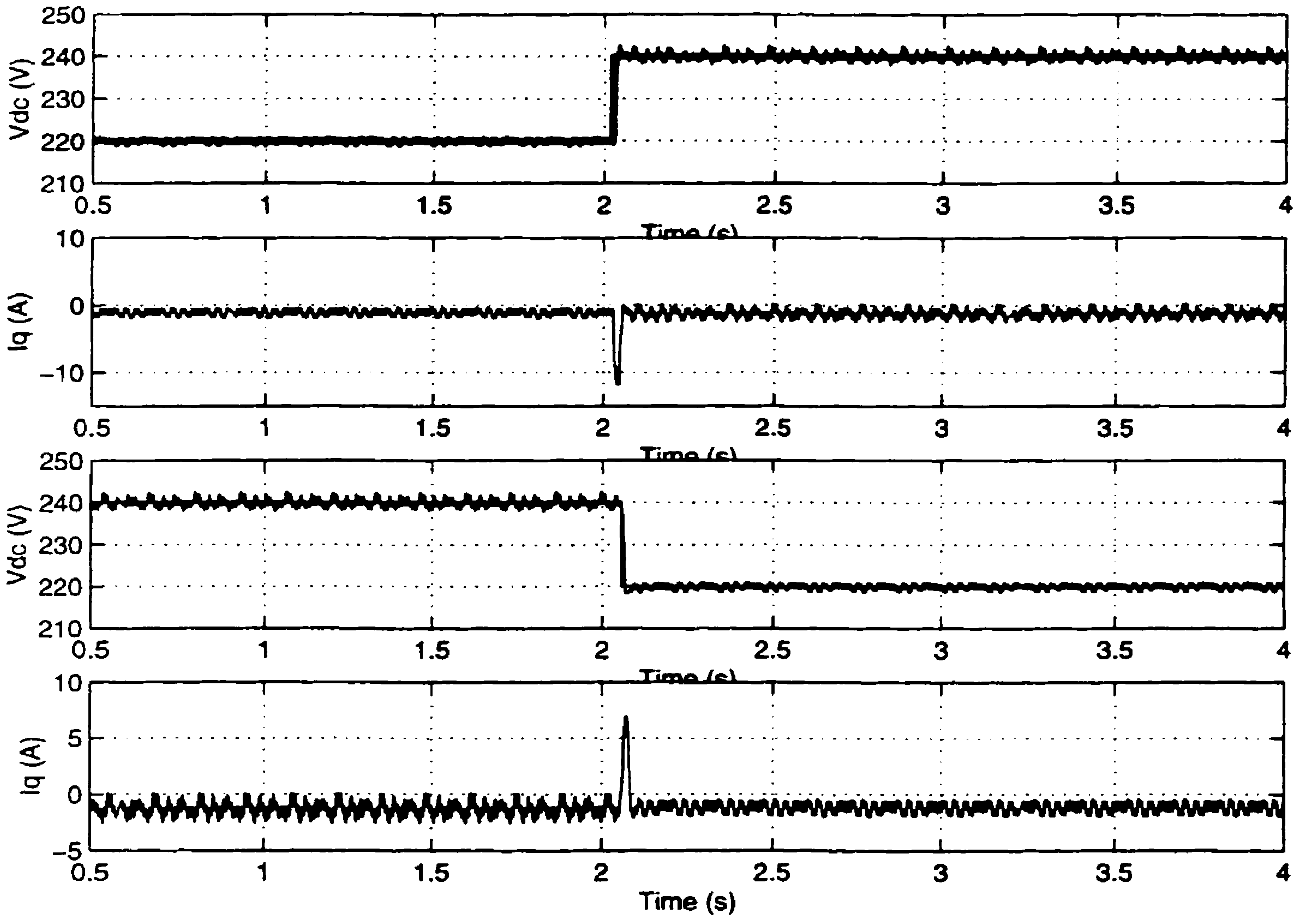


Figure 4.29: DC side transient showing step response of the v_{dc} controller (RTS)

4.6 Processor Timing

In order to determine the lower limit of the time step or to determine how much more system complexity can be included while still retaining real-time execution, it is helpful to examine the processor timings. The time taken by the C40 DSP to execute each of the modules in the user interrupt function were measured. The real-time simulation program `rts_110.c` uses a time step of $100\mu s$ out of which $84\mu s$ is the total simulation time and $16\mu s$ is the overhead for tasks such as initializing the DSP Kernel program ($4\mu s$) and managing data acquisition and display of upto three signals ($12\mu s$) on the PC. The simulation time comprises of $36\mu s$ used by the Simulator module and $48\mu s$ used by the Controller module. Inside the Simulator the linear interpolation and variable step-size code takes $26\mu s$ while the normal fixed step-size code takes $16\mu s$ to execute. The Controller module runs at a frequency of $2kHz$ which amount to a control period of $500\mu s$. The FPGA operates at a clock frequency of $20MHz$. The PWM and the SEC in the FPGA are carried out in parallel with the simulation in the DSP. The Switching Event Capture module takes about $15ns$ to detect a firing pulse and transfer its timing information to the simulation. From the point of view of the simulation in the DSP the FPGA response times can be considered to be instantaneous.

Under the present design conditions the resources of the C40 DSP and the FPGA were completely utilized. Simulation of a more complex system or adopting a higher switching frequency for the same system would require a smaller time step necessitating a change of design or the use of faster DSP's.

4.7 Conclusions

This chapter presented the design and implementation of a real-time digital simulator for the PWM VSI system. The hardware and software design details were presented. In addition an experimental setup of the VSI system was described. The contents of the chapter can be summarized as follows.

1. The design of the real-time digital simulator was based on a digital platform comprising of a C40 DSP and a FPGA. The Simulator and Controller modules on the DSP were developed in software written in C language while the PWM and the Switching Event Capture (SEC) modules were implemented using digital hardware design on the FPGA. Such a design was adopted because during the course of a simulation the user mainly interacts with the software modules by changing the parameters for the simulation or the controller while the hardware design modules of PWM and SEC are seldom changed once a simulation is initialized.

2. The experimental setup of the PWM VSI system utilizes the same digital platform as that for the real-time simulator. The digital controller was implemented in software on the C40 DSP. The PWM generator implemented on the FPGA is identical to that of the real-time simulator setup with the exception of the Dead Time Insertion module.

3. The real-time simulation and the experiment were carried out under similar operating conditions. Both open-loop and closed-loop control studies were performed on the VSI system. In spite of some inevitable implementation constraints agreement between the real-time simulation results and the experimental results confirms the viability of the proposed (FICS) simulation algorithm.

Chapter 5

Conclusions

Real-time simulation is a crucial stage before a power apparatus is commissioned. Therefore it is imperative to identify problems and their solutions to inspire confidence in the simulation results and to assure reliable operation of the power apparatus. This thesis has contributed to advances in the real-time digital simulation of power electronic circuits interfaced with digital controllers. The advances have been made possible by developing an algorithm and by developing software for simulation under off-line as well as real-time conditions and for the experimental control of a PWM VSI system.

5.1 Summary

The main conclusions of this thesis can be summarized as follows:

1. Existing off-line simulation algorithms and tools were investigated from the point of view of their application to real-time simulation. It was found that several elegant off-line algorithms violated real-time constraints precluding their use for real-time simulation. Furthermore, currently available power system simulation tools such as EMTP could not be adapted for use under real-time conditions due to implementation constraints.

2. To study the effects of switching events in digital simulation the interactions of a real-time digital simulator with a digital controller has been modeled with the aid of sampling theory. This model not only provides an insight into the origin of simulation errors caused by asynchronous switching events but also helps in incorporating practical implementation conditions in off-line simulation programs.

3. A new real-time simulation algorithm has been developed for accounting asynchronous switching events in digital simulation. This algorithm relies on the registration of timing of the switching events as they arrive and subsequently applies a correction while calculating the next system state. The correction procedure employs linear interpolation as a means to estimate the system state at the instant of the event occurrence. The proposed algorithm can be readily programmed into an off-the-shelf DSP system to serve as a stand alone real-time simulator or it can be used to develop a separate processing module that simulates only part of the simulated system to be retrofitted to an existing real-time digital simulator. The only requirement is the need for additional hardware to capture the timing of the incoming switching events.

4. Detailed off-line time domain simulations were performed on a PWM VSI system. Simulation programs developed for the fixed step-size algorithm and the proposed algorithm incorporate the conditions found in the practical implementation of digital controllers. Off-line simulations using the fixed step-size algorithm demonstrate the following effects of not accounting switching events at their proper timing:

   - A drastic change in the fundamental quantities.
   - Increase of non-characteristic harmonics.
   - Appearance of jitter in control quantities.

   These effects were found to diminish when the simulation step-size was chosen to be sufficiently small. The proposed (FICS) algorithm offered at least a ten fold improvement over the fixed step-size algorithm employing the same step-size. In addition, the execution time of the proposed algorithm was found to be only slightly higher than that of the fixed step-size algorithm.

5. Practical feasibility of the proposed (FICS) real-time simulation algorithm was demonstrated by its implementation on a digital processing platform comprising of a DSP and a FPGA. The software for the simulator and the controller was developed in C language and is fully portable to next generation of processors with minimum modifications.

6. A 5kVA PWM VSI system was used for experimental verification. The experimental and the real-time simulation were carried out under similar operating conditions. Open-loop and closed-loop control studies performed on the VSI system indicated a close agreement between the two sets of results despite certain unavoidable implementation constraints.

5.2 Contributions

The main contributions of this thesis are the following:

1. The interaction of a real-time digital simulator interfaced with a digital controller has been modeled with an intent to delve into the simulation errors caused by inter-step switching events.

2. A novel real-time simulation algorithm (FICS) has been proposed to synchronize the operation of a real-time digital simulator with the output of a digital controller.

3. Practical feasibility of the proposed algorithm has been demonstrated by real-time implementation on a DSP-FPGA platform.

4. Real-time and off-line simulation results have been experimentally verified.

5.3 Future Work

Parallel processing is an emerging area which offers a wide variety of new computer architectures which can be harnessed for the real-time simulation and control of power systems. Appendix A presents simulation paradigms for the following three types of systems where the need for computational power and speed are expected to be the greatest.

- Real-time simulation of complex power electronic systems poses a challenge due to the multitude of switching events that the simulator has to cope with in every time step. A multi-processor simulator architecture, based on the FICS real-time simulation algorithm presented in Section 2.4, is proposed for such systems.
- Systems that are characterized by a wide range of eigenvalues are difficult to simulate in real-time due to their requirement of different time steps for efficient simulation. A multirate simulation method is proposed for such systems for efficient simulation in a multi-processor environment.
- Systems that contain non-linear or time-varying elements pose a formidable challenge for real-time simulation due to their requirement of iterating the system equations or reformulation of the system sdmittance matrix. The proposed simulatoı architecture utilizes the Network Equivalent's method which facilitates the separation of the linear part of a system from its non-linear or time-varying parts.

References

[1] T. Sakaguchi Y. Sekine, K. Takahashi, "Real-time simulation of power system dynamics", *Electrical Power and Energy Systems*, vol. 16, no. 3, pp. 145–156, June 1994.

[2] A. M. Gole, V. K. Sood, "A static compensator model for use with the electromagnetic transients simulation programs", *IEEE Transactions on Power Delivery*, vol. 5, no. 3, pp. 1398–1406, July 1990.

[3] J. M. Zahavir, J. Arrillaga, N. R. Watson, "Hybrid electromagnetic transient simulation with the state variable representation of HVDC converter plant ", *IEEE Transactions on Power Delivery*, vol. 8, no. 3, pp. 1591–1598, July 1993.

[4] "RTDS Technologies Inc.", Web address : http://www.rtds.com.

[5] "Teqsim International", Web address : http://www.teqsim.com/en/Products.html.

[6] "dSPACE Inc.", Web address : http://www.dspace.de/en/Products/Products.htm.

[7] *Proceedings of ICDS '95, First International Conference on Digital Power System Simulators*, May 1995.

[8] H. W. Dommel, "Digital computer solution of electromagnetic transients in single and multiphase networks", *IEEE Transactions on Power Apparatus and Systems*, vol. 88, pp. 388–395, April 1969.

[9] T. Kato, K. Ikeuchi, "Variable order and variable step-size integration method for transient analysis programs", *IEEE Transactions on Power Systems*, vol. 6, no. 1, pp. 206–214, February 1991.

[10] J. R. Marti, L. R. Linares, "Real-time EMTP based transient simulation", *IEEE Transactions on Power Systems*, vol. 9, no. 3, pp. 1309–1316, August 1994.

[11] M. L. Crow, J.G. Chen, "The multirate simulation of FACTS devices in power system dynamics", *IEEE Transactions on Power Systems*, vol. 11, no. 1, pp. 376–383, February 1996.

[12] A. E. A. Araujo, H. W. Dommel, J. R. Marti, "Converter simulations with the EMTP: simultaneous solution and backtracking technique", in *IEEE/NTUA Athens Power Tech Conference*, September 1993.

[13] P. Kuffel, K. Kent, G. Irwin, "The implementation and effectiveness of linear interpolation within digital simulation", in *IPST'95- International Conference on Power Systems Transients*, September 1995.

[14] A. M. Gole, I. T. Fernando, G. D. Irwin, O. B. Nayak, "Modeling of power electronic apparatus: additional interpolation issues", in *ICDS'97- International Conference o Power Systems Transients*, June 1997.

[15] S. Acevedo, L. R. Linares, J. R. Marti, Y. Fujimoto, "Efficient HVDC converter model for real time transients simulation", *IEEE Transactions on Power Systems*, vol. 14, no. 1, pp. 166–171, February 1999.

[16] R. Kuffel, J. Giesbrecht, T. Maguire, R. P. Wierckx, P. G. McLaren, "RTDS-a fully digital power system simulator operating in real-time", in *Proceedings of EMPD'95- International Conference on Energy Management and Power Delivery*, 1995, vol. 2.

[17] P. Mercier, C. Gagnon, M. Tetreault, M. Toupin, "Real-time digital simulation of power systems at Hydro-Quebec", in *ICDS '95, First International Conference on Digital Power System Simulators*, April 1995.

[18] D. Jakominich, R. Krebs, D. Retzmann, A. Kumar, "Real time digital power system simulator design considerations and relay performance evaluation", *IEEE Transactions on Power Delivery*, vol. 14, no. 3, pp. 773–781, July 1999.

[19] J. Reeve, S. P. Lane-Smith, "Integration of real-time controls and computer programs for simulation of direct current transmission", *IEEE Transactions on Power Delivery*, vol. 5, no. 4, pp. 2047–2053, October 1990.

[20] R. C. Durie, C. Pottle, "An extensible real-time digital transient network analyzer", *IEEE Transactions on Power Systems*, vol. 8, no. 1, pp. 84–89, February 1993.

[21] M. Kezunovic, V. Skendic, M. Aganagic, J. K. Bladow, S. M. McKenna, "Design, implementation and validation of a real-time digital simulator for protective relay testing", *IEEE Transactions on Power Delivery*, vol. 11, no. 1, pp. 158–164, January 1996.

[22] P. G. McLaren, R. Kuffel, R. Wierckx, J. Giesbrecht, L. Arendt, "A real time digital simulator for testing relays", *IEEE Transactions on Power Systems*, vol. 7, no. 1, pp. 207–213, January 1992.

[23] D. Brandt, R. Wachal, R. Valiquette, R. Wierckx, "Closed loop testing of a joint VAR controller using a digital real-time simulator", *IEEE Transactions on Power Systems*, vol. 6, no. 3, pp. 1140–1146, August 1991.

[24] K. Bergmann, K. Stump, W. H. Elliot, "Digital simulation, transient network analyzer and field tests of the closed loop control of the eddy county SVC", *IEEE Transactions on Power Delivery*, vol. 18, no. 4, pp. 1867–1873, October 1993.

[25] M. La Scala, R. Sbrizzai, F. Torelli, P. Scarpellini, "A tracking time domain simulator for real-time transient stability analysis", *IEEE Transactions on Power Systems*, vol. 13, no. 3, pp. 992–998, August 1998.

[26] C. G. Groom, K. W. Chan, R. W. Dunn, A. R. Daniels, "Real-time security assessment of electrical power systems", *IEEE Transactions on Power Systems*, vol. 11, no. 2, pp. 1112–1117, May 1996.

[27] F. Wang G. Janka, G. Schellstede, "Dynamic power system simulation for real-time dispatcher training", in *Third International Conference on Power System Monitoring and Control*, 1991.

[28] A. V. Oppenheim, *Signals and Systems*, Prentice Hall, NJ., 1997.

[29] C. W. Gear, *Numerical initial value problems in ordinary differential equations*, Prentice-Hall Inc., 1971.

[30] J. Holtz, "Pulse Width Modulation- a survey", *IEEE Transactions on Industrial Electronics*, vol. 39, no. 5, pp. 410–419, December 1992.

[31] S. Krebs, *UHP40 User's Manual*, University of Toronto, December 1995.

[32] An IEEE committee report by a task force of the computer and analytical methods subcommittee of the power systems engineering committee, "Parallel processing in power systems computation", *IEEE Transactions on Power Systems*, vol. 7, no. 2, pp. 629–636, May 1992.

[33] G. Aloisio, M. A. Bochicchio, M. LaScala, R. Sbirizzai, "A distributed computing approach for real-time transient stability analysis", *IEEE Transactions on Power Systems*, vol. 12, no. 2, pp. 981–987, May 1997.

[34] J. S. Chai, A. Bose, "Bottlenecks in parallel algorithms for power system stability analysis", *IEEE Transactions on Power Systems*, vol. 8, no. 1, pp. 9–15, February 1993.

[35] I. C. Decker, D. M. Falcao, E. Kaszkurewicz, "Parallel implementations of a power system dynamic simulation methodology using the conjugate gradient method", *IEEE Transactions on Power Systems*, vol. 7, no. 1, pp. 458–465, February 1992.

[36] H. W. Dommel, "Nonlinear and time-varying elements in digital simulation of electromagnetic transients", *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-90, no. 6, pp. 2561–2567, Nov./Dec. 1971.

Appendix A

Parallel Processing Solutions For Real-Time Simulation

A.1 Introduction

Sequential computers i.e., computers consisting of a single processor that performs one operation at a time are rapidly reaching a physical barrier beyond which their processing speed cannot be increased through the use of faster components. The time taken by single processor computers to simulate realistic size power systems is slow in the off-line mode let alone real-time. A practical way to solve the ever increasing need for faster and more powerful computation is to employ parallel computers that use several processors to solve a problem. There are several areas in power system reasearch that can benefit from the use of parallel processing techniques [32]. A reduction in computation time using parallel processing results in two main advantages. Firstly the ability to simulate a system in real-time and secondly the ability to increase the size and the amount of modeling detail of the simulated power system.
The purpose of this appendix is to explore the application of parallel processing for the real-time simulation of power systems. Three key simulation problems are addressed:

1. Power systems comprising of embedded complex power electronic circuits such as a multilevel converter system or a HVDC system.

2. Systems comprising of widely distributed eigenvalues for which a single simualtion time step is usually not efficient to represent all system dynamics.

3. Systems containing non-linear or time varying elements.

A.2 Applications of Parallel Processing to Power System Simulation

The main motivation in applying parallel processing techniques to power system simulation problems is to speed up computation for on-line applications. Most simulation problems are not naturally amenable for direct implementation on parallel machines. In comparison with sequential computing significant computational gains can only be achieved if the simulation algorithm is designed so as to take full advantage of the parallel hardware architecture of the simulator. The conventional approach in designing a parallel simulation algorithm is to decompose a given problem into sub-tasks that can be assigned to different processors. The two main limiting factors using such an approach are inter-processor communication cost and unsolvable sequentiality in the problem itself. Furthermore, any limitations on the part of an unsophisticated compiler that are invisible to the programmer may also seriously undermine computational efficiency of the algorithm.

An area of power system computation that has seen an extensive application [33–35] of parallel processing techniques is the transient stability simulation problem. This problem requires the solution of differential equations representing the dynamics of rotating machines together with the algebraic equations representing the network. Other potential application areas include on-line load flow, reliability and short circuit calculations. A common characteristic of the transient stability problem and the others that encourages the application of parallel processing techniques is the size of the problem; most of these problems require, very often, several hundred equations to be solved every time step. On the other hand the application of parallel processing for the real-time simulation of electromagnetic transients is motivated due to two different reasons other than the size of the simulated system. Firstly, the time scale of the transients and secondly the inclusion of power electronic apparatus which increase the modeling complexity of the simulation. These two factors force the designer to restrict the simulation time step to a small value and thereby explore alternate means to speed up the computation.

This section presents three simulation paradigms where significant computational gains can be achieved through the use of a multi-processor architecture for the power system simulator.

A.2.1 Simulation Paradigm #1

Systems comprising of embedded complex power electronic circuits present a particular challenge for real-time simulation. One example is a system comprising of a multi-pulse converter configuration, where several six-pulse converters are operated from a common DC bus to reduce the output harmonic content, employed in such FACTS controllers as the STATCOM and the UPFC. Another example is a back-to-back HVDC link with detailed models of the converters, transformers, filters, and the extended AC system. A real-time simulator simulating such systems is faced with the rigour of accounting incoming switching signals for the power electronic devices. As discussed in Chapter 2, during the simulation of power electronic circuits majority of simulation errors stem from the inability of the simulator to account for the switching signals at their proper time instants. Therefore, it is important to curtail the number of incoming switching events in a single calculation step of the simulator. There are two scenarios which can lead to an increase in the number of switching events in any given time step. First is an increase in the switching frequency of the power electronic apparatus and second is an increase in the number of basic power electronic units, such as the six-pulse converter configuration, each operating at a low switching frequency. Such modeling complexities necessitate a reduction in the simulation time step. Using a fixed step-size integration algorithm, the extent to which the time step can be reduced without compromising real-time operation depends on the computational capability of the simulator. The real-time simulation algorithm FICS proposed in Section 2.4 may be employed in those cases when a reduction in the step-size for a fixed step-size method entails a heavier computational burden than that for carrying out the operations in the proposed algorithm itself. Accordingly it is worthwhile to examine the limitations imposed by the FICS algorithm on computational requirements. The two bottlenecks that need to be overcome before the proposed algorithm can be applied to larger systems are

1. Linear interpolation of several system variables. Although linear interpolation is computationally less intensive than a full state calculation, it could still constitute a considerable portion (10 to 50 percent) of the total computation time.

2. Inversion of α and β. As seen from equations 2.14 and 2.15, α and β are functions of the network parameters a, b and the time step Δt. Assuming that the network in question is linear and time-invariant, α^{-1} and β^{-1} need to be re-calculated whenever Δt is changed. In a more general case, this would mean inverting G (2.18) whenever Δt is changed.

In contrast to using a single processor to carry out the tasks sequentially, a multiprocessor architecture may be used to overcome the above two limitations. Figure A.1 illustrates a simulator which employs three processors in a shared memory configuration. There is a main processor which performs the tasks of network solution and oversees the communication with the external controller and the user while the two auxiliary processors

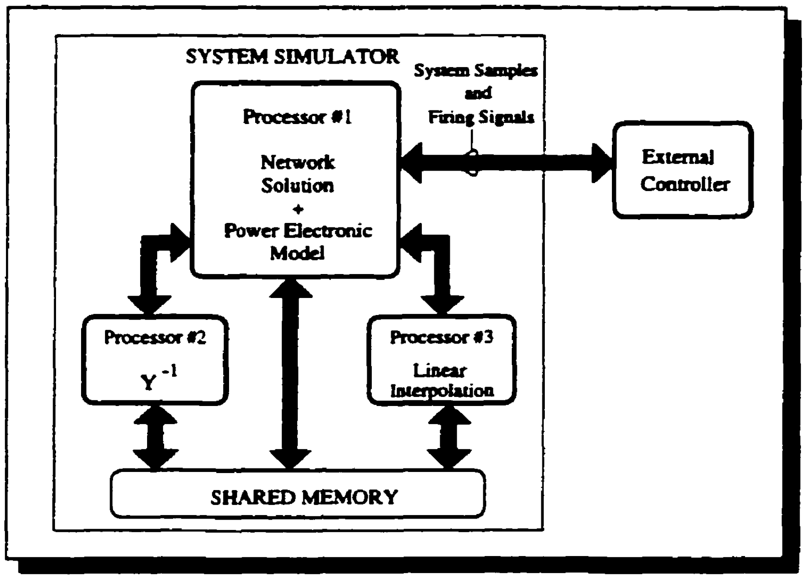


Figure A.1: Multi-processor simulator architecture for simulation example #1

solely perform the functions of linear interpolation and the inversion of α and β. When a switching event is detected the main processor can broadcast the relevant information to the auxiliaries and engage itself in other calculations while waiting for their response. A computationally efficient approach to circumvent the direct inversion of α and β when the step-size is changed is to store pre-calculated values of α^{-1} and β^{-1}, computed for several sub-multiples of the time step Δt, in a look-up table and apply them when needed. For example, if $\Delta t = 100\mu s$, α^{-1} and β^{-1} may be pre-computed at $5\mu s$ intervals resulting in 20 different values for each. Such an approach may not be precise as to the location of the switching event but it is sufficiently accurate compared with a calculation based on a single time step of $100\mu s$. The network solution in the main processor may itself be divided if needed into different sub-tasks. One practical scenario for simulating a three-phase system is to carry out the calculations concurrently in three separate processors one for each phase.

A.2.2 Simulation Paradigm #2

Power system transients are characterized by eigenvalues which can be of different orders of magnitudes. Transients corresponding to eigenvalues with large negative real parts decay rapidly (typical time constants are in the range of a few microseconds to tens of milliseconds) and are usually associated with electromagnetic phenomena. Examples of such transients include switching overvoltages, short-circuit transients, dynamics of FACTS apparatus and induction machine loads. On the other hand, transients corresponding to eigenvalues with smaller negative real parts decay slowly (typical time constants range from a few seconds to a few minutes) and are associated with phenomena that are electro-mechanical in nature. A prime example in this category is the dynamics associated with a generator model that includes the governor and the turbine. Certain loads may also be characterized by slow transients. Off-line transient programs such as the EMTP utilize a single time step for the simulation of a system that has both slow and fast transient components. The time step is usually chosen to be small based on the fastest transient component alone. Such an approach seriously undermines computational efficiency by slowing down the entire simulation and therefore becomes impractical for real-time applications. A more efficient approach would be to employ multiple time steps for the simulation based on the requirements of the individual components.

A multirate algorithm [11, 29] for integrating ordinary differential equations is one in which different equations are integrated using different step-sizes. Each system variable or a certain group of variables is integrated with a step-size which is sufficient for the required accuracy. The selection of the multiple step-sizes begins with dividing a given system into multiple subsystems. Such a division can be accomplished by examining the time responses of system eigenvalues at steady state. Figure A.2(A) shows an example system with two time-step related subdivisions. The slow subsystem S comprises of the generator, transmission lines and the loads while the fast subsystem F consists of a power

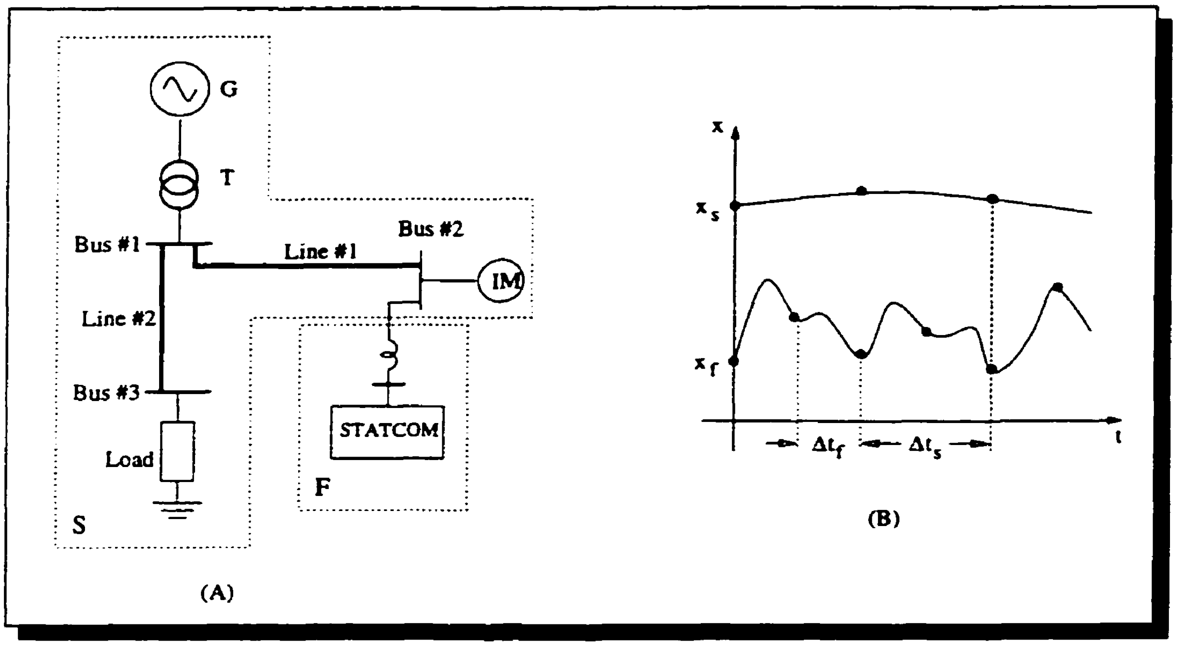


Figure A.2: (A) System for simulation example #2, F: fast Subsystem , S: Slow Subsystem, (B) Multi-rate simulation

electronic compensator. Let $x_s(t)$ represent a state of the slow subsystem e.g. voltage at bus#1 and $x_f(t)$ represent a state of the fast subsystem e.g. voltage at the STATCOM terminals. Figure A.2B illustrates the integration of $x_s(t)$ and $x_f(t)$ over time. $x_f(t)$ is integrated with a time step Δt_f whereas $x_s(t)$ is integrated with a time step $\Delta t_s = 2\Delta t_f$. Note that the values of the state $x_s(t)$ are not available at all instants of time; $x_s(t)$ needs to be approximated at the required time instant for the calculation of $x_f(t)$. Linear interpolation may be employed to estimate $x_s(t)$ without incurring undue computational burden. Computational efficiency is gained when the faster subsystem is smaller in size compared to the slower subsystem since the latter is only solved infrequently compared to the former.

The division of a system into smaller subsystems facilitates the use of several processors for its real-time simulation. Figure A.3 illustrates a multi-processor simulator for the system shown in Figure A.2(A). One processor is assigned each for the slow and fast subsystems. The fixed step-size integration algorithm may be used for both subsystems.

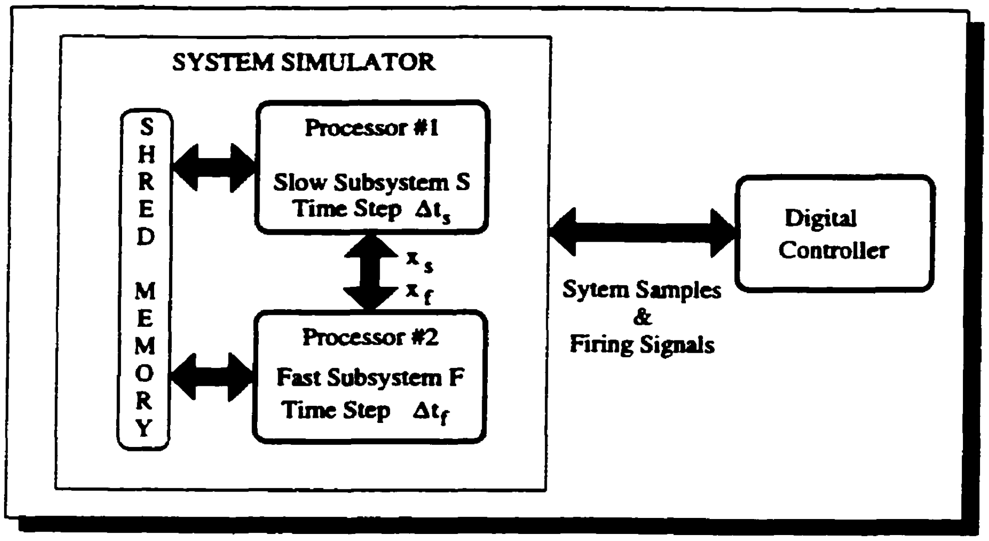


Figure A.3: Multi-processor architecture for simulation example #2

Alternately, the proposed approach of Section 2.4 may be employed for the faster subsystem F. Irrespective of the method chosen, the following two conditions must be met for real-time implementation:

1. The larger time step Δt_s must be an integer multiple of the smaller time step Δt_f to synchronize the operations of the two processors.

2. Real-time requirement i.e., each time step must be greater than the execution time of the algorithm for the respective subsystem.

It is possible to further divide the slower subsystem into smaller parts in which case a hierarchy of multiple step-sizes may be established for the whole system. Although the theoretical concepts of multirate methods such as the numerical properties of stability, convergence, etc, have been well established [29], their practical implementation for simulating complex power systems in real-time needs further research.

A.2.3 Simulation Paradigm #3

Transient simulation of power systems is carried out using the equation

$$[Y]\,[v(t)] = [J] \tag{A.1}$$

formulated using nodal analysis. The state of the system is found by repeated solution of (A.1) at every time step. Methods of solution for (A.1) include LU factorization, Gaussian Elimination and even a direct inversion of $[Y]$ in certain cases. Regardless of the method used, if the system in question is linear and time-invariant then $[Y]$ needs to be factorized or inverted only once in the preamble of the transient program before entering the time-step loop. The same factors of $[Y]$ may then be re-applied every time step since $[Y]$ remains essentially constant, a most desirable condition for real-time simulation. However, if the system in question consists of non-linear or time-varying elements then the $[Y]$ matrix does not remain unchanged any longer, its entries become functions of voltage, current or time. Commonly occurring nonlinearities include magnetic saturation and hysteresis in transformers and reactors. Examples of time-varying elements include circuit breakers, protective gaps, lightning arresters etc. If the $[Y]$ matrix is only a function of time, (A.1) forms a linear time-varying algebraic equation set and direct methods such as Gaussian Elimination may be used for its solution although $[Y]$ needs to re-triangularized at every time step of the transient simulation. However, if the elements of $[Y]$ are functions of voltages and/or currents, then the solution of (A.1) becomes non-trivial; an iterative numerical scheme such as the Newton-Raphson method must be adopted to solve (A.1) at each time step. Re-triangularizing $[Y]$ for the entire system or iterating (A.1) form serious bottlenecks if the simulation needs to be carried out in real-time. The simulation algorithm would be computationally efficient if the linear part of the system were separated from the non-linear and time-varying part. Two conventional methods [36] to accomplish such a separation are the Current Compensation method and the Network

Equivalent's method. The Current Compensation method based on the Compensation Theorem allows a non-linear or time-varying element between the nodes p and q to be excluded from the rest of the network and the $[Y]$ matrix and instead be simulated by a current source I_{pq}. In the Network Equivalent's method the re-factorization of the $[Y]$ matrix or iteration of (A.1) is confined to only those nodes of the network that are incident to the non-linear or time-varying elements. Such a confinement is accomplished by deriving an equivalent for the linear part of the network. Although both approaches are effective tools for simulation using sequential programming, the Network Equivalent's method is more suitable for implementation using multiple processors. Let subscript 1 denote the set of nodes which are incident to linear elements only and subscript 2 denote the set of nodes that are incident to non-linear or time-varying elements. Then (A.1) can be partitioned as

$$\begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22}(v,i,t) \end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \end{bmatrix} = \begin{bmatrix} J_1 \\ J_2 \end{bmatrix} \tag{A.2}$$

As seen from (A.2) only the submatrix Y_{22} is a function of v, i, or t. The vectors $v_1(t)$ and $v_2(t)$ can be obtained by solving the following equations

$$\left([Y_{22}(v,i,t)] - [Y_{21}]\,[Y_{11}]^{-1}\,[Y_{12}]\right)[v_2(t)] = [J_2] - [Y_{21}]\,[Y_{11}]^{-1}\,[J_1] \tag{A.3}$$

and

$$[Y_{11}]\,[v_1(t)] = [J_1] - [Y_{12}]\,[v_2(t)] \tag{A.4}$$

First (A.3) is solved for $v_2(t)$ either iteratively or by a direct method. Then $v_2(t)$ is substituted in (A.4) to obtain $v_1(t)$. Figure A.4 illustrates a simulator where solutions of (A.3) and (A.4) are carried out in two different processors. The matrix products $[Y_{21}]\,[Y_{11}]^{-1}\,[Y_{12}]$ and $[Y_{21}]\,[Y_{11}]^{-1}\,[J_1]$ in (A.3) are constant which allows them to be pre-calculated and stored in memory. The same holds true for $[Y_{11}]$ in (A.4), which may be solved by direct inversion of $[Y_{11}]$. The main computational burden here is the solution

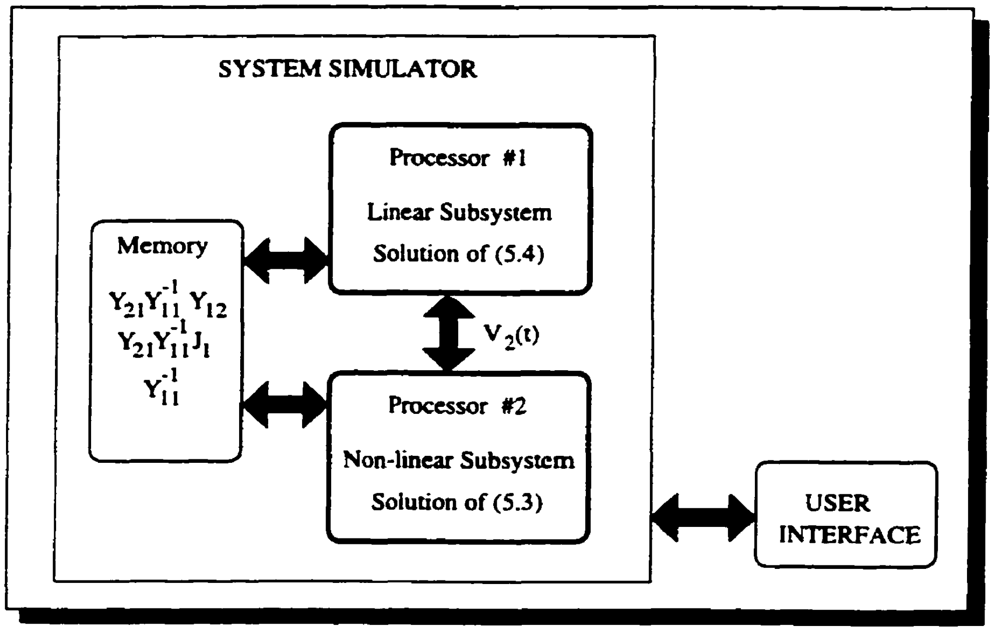


Figure A.4: Multi-processor simulator architecture for simulation example #3

solved by direct inversion of $[Y_{11}]$. The main computational burden here is the solution of $v_2(t)$ and it is kept to a minimum by restricting the iterative process only to the node subset 2. Computational efficiency is gained since instead of factorizing $[Y]$ for the whole network, only $[Y_{22}]$ needs to be changed. The computation of the linear part may be further divided into different processors depending on the size of the network and the computational speed requirements.

Appendix B

System Parameters

AC Source Voltage

Three-phase $E = 110Vrms\, l-l$, $60Hz$

Transformer

Three-phase $115V/115V$, $10kVA$, $X_T = 3.06\%$, $R_T = 2.71\%$

AC Circuit Parameters

$R = 0.5\Omega$, $L = 3.0mH$

DC Link Capacitance

$C = 4900\mu F$

Control Parameters

$K_{id} = 0.8$, $K_{iq} = 1.5$, $T_{id} = T_{iq} = 50ms$, $K_v = -2.0$, $T_v = 0.5s$

Switching Frequency

$f_{sw} = 1kHz$

Controller Sampling Frequency

$f_s = 2kHz$

Appendix C

FPGA Design Files

This appendix presents the FPGA design files for real-time simulation and experimental control of the three-phase PWM VSI system. The FPGA used is a FLEX 8000 device from ALTERA. The design has been carried out using the MAXPLUS II graphic design software. The list of the graphic design files (.gdf) and a brief description of each file is given below.

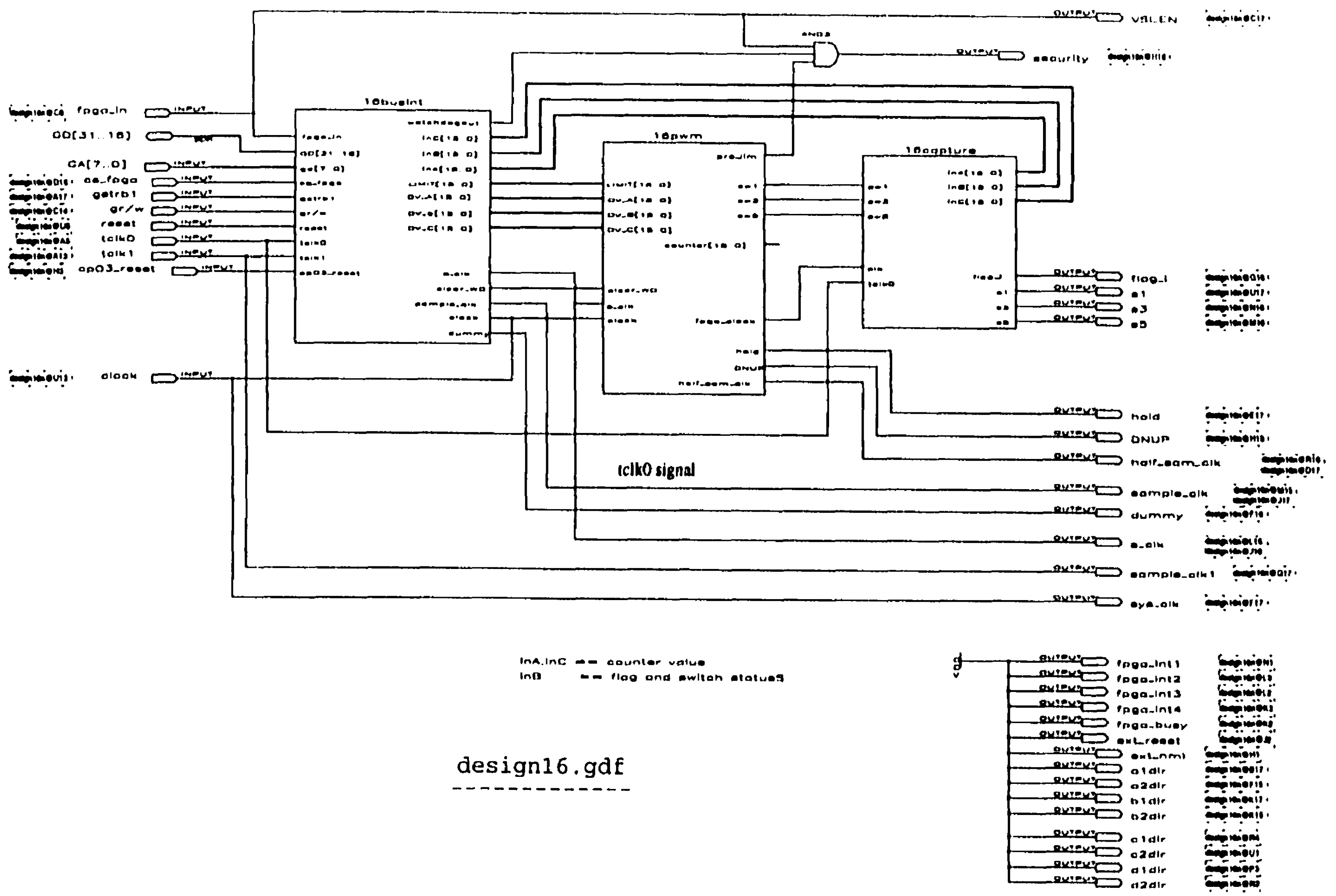
C.1 Real-Time Digital Simulator and Controller

- `design16.gdf` : Top design file
- `16busint.gdf` : Data Bus Interface to C40 DSP
- `16pwm.gdf` : Pulse Width Modulator
- `16capture.gdf` : Switching Event Capture
- `datadir2.gdf` : Bidirectional data buffer
- `compar16.gdf` : 16-bit Digital Comparator
- `freq_div.gdf` : Frequency Divider

- `reg16.gdf` : 16-bit register
- `reg8.gdf` : 8-bit register
- `swich16.gdf` : Multiplexer
- `switch8.gdf` : Multiplexer

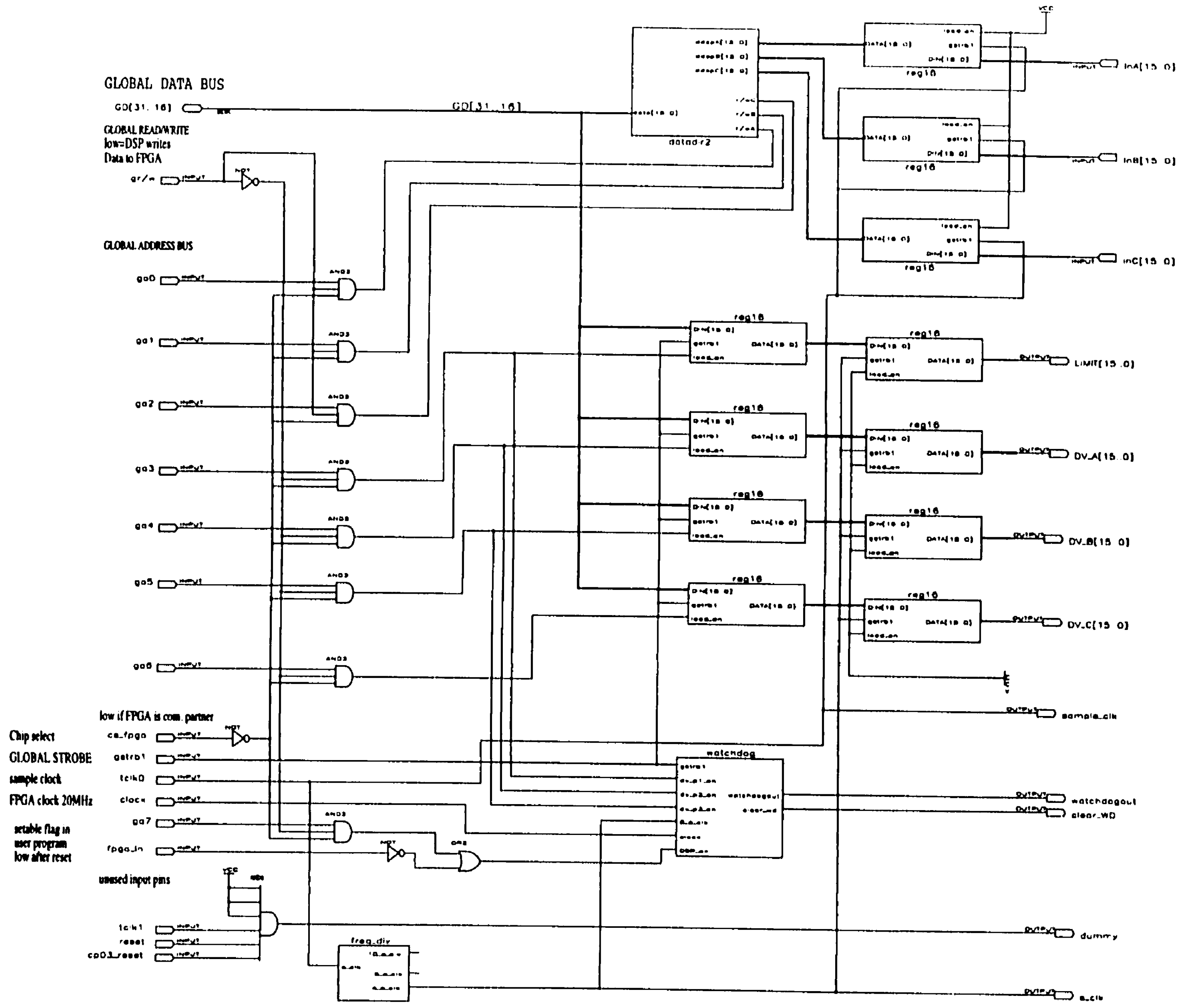
C.2 Experimental Setup

- `con_fpga.gdf` : Top Design File
- `16busint.gdf` : Data Bus Interface to C40 DSP
- `16pwm.gdf` : Pulse Width Modulator
- `gating.gdf` : PWM Dead Time Insertion
- `watchdog.gdf` : C40 Watchdog

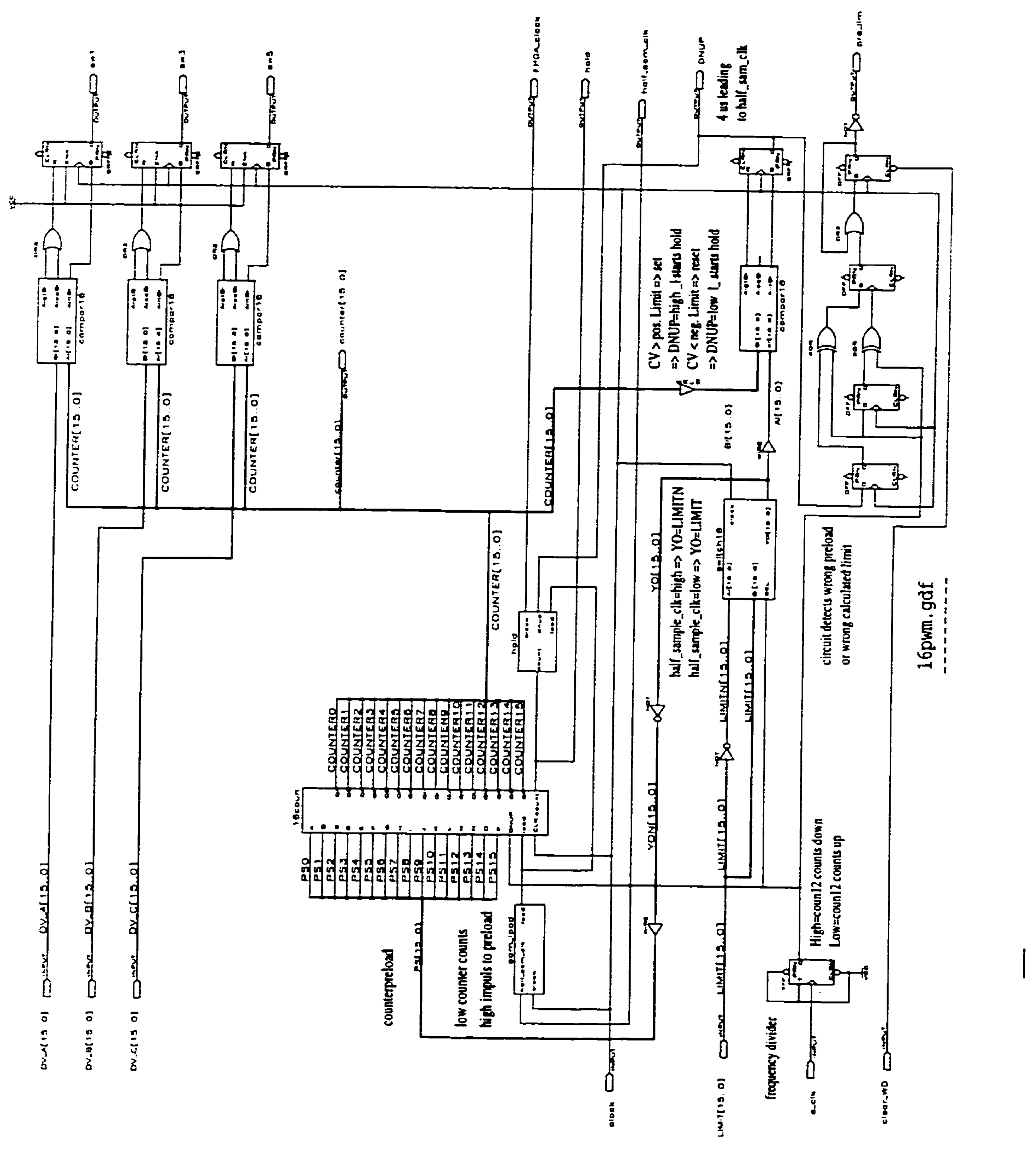


InA,InC == counter value
InB == flag and switch statuS

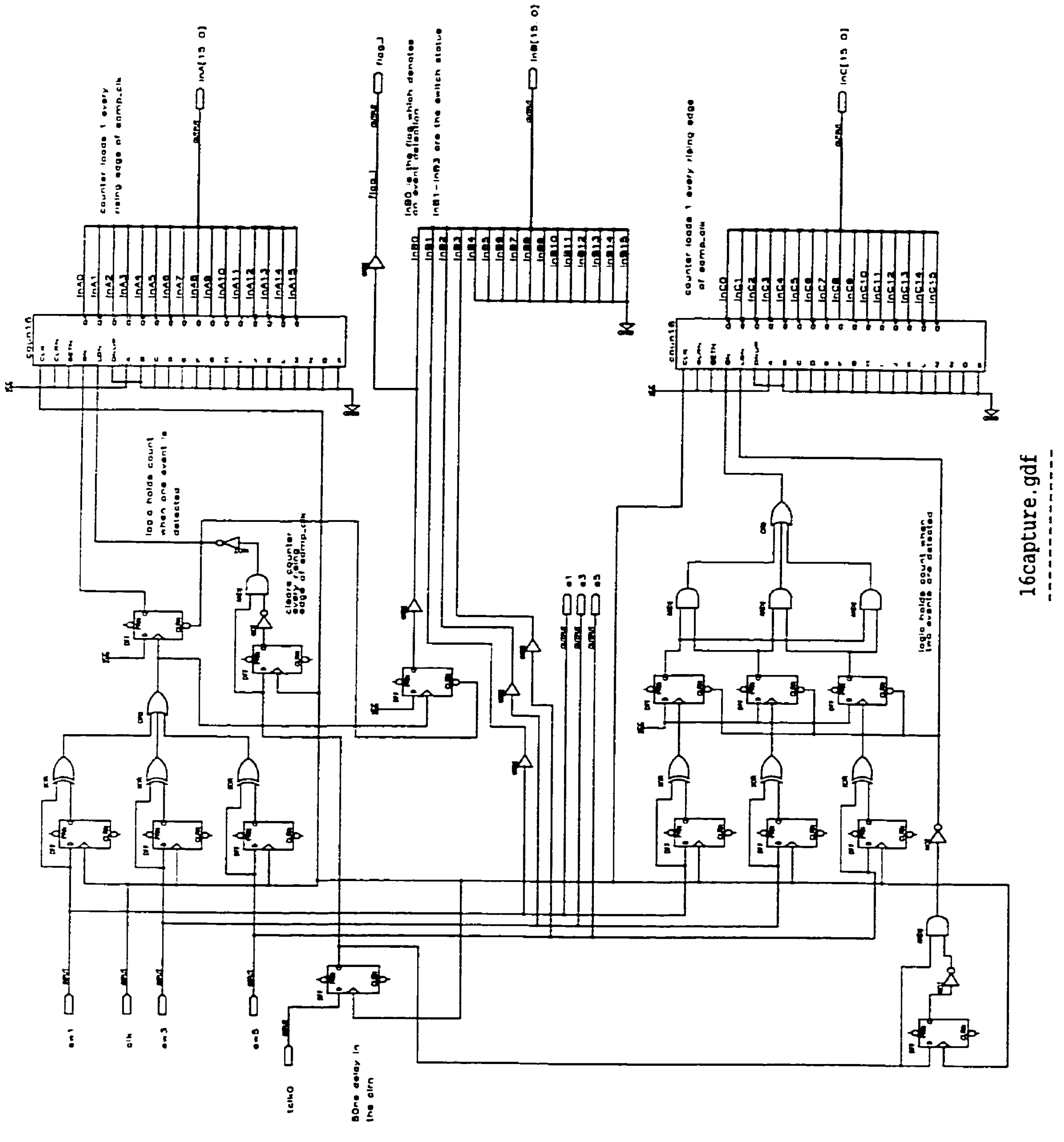
design16.gdf

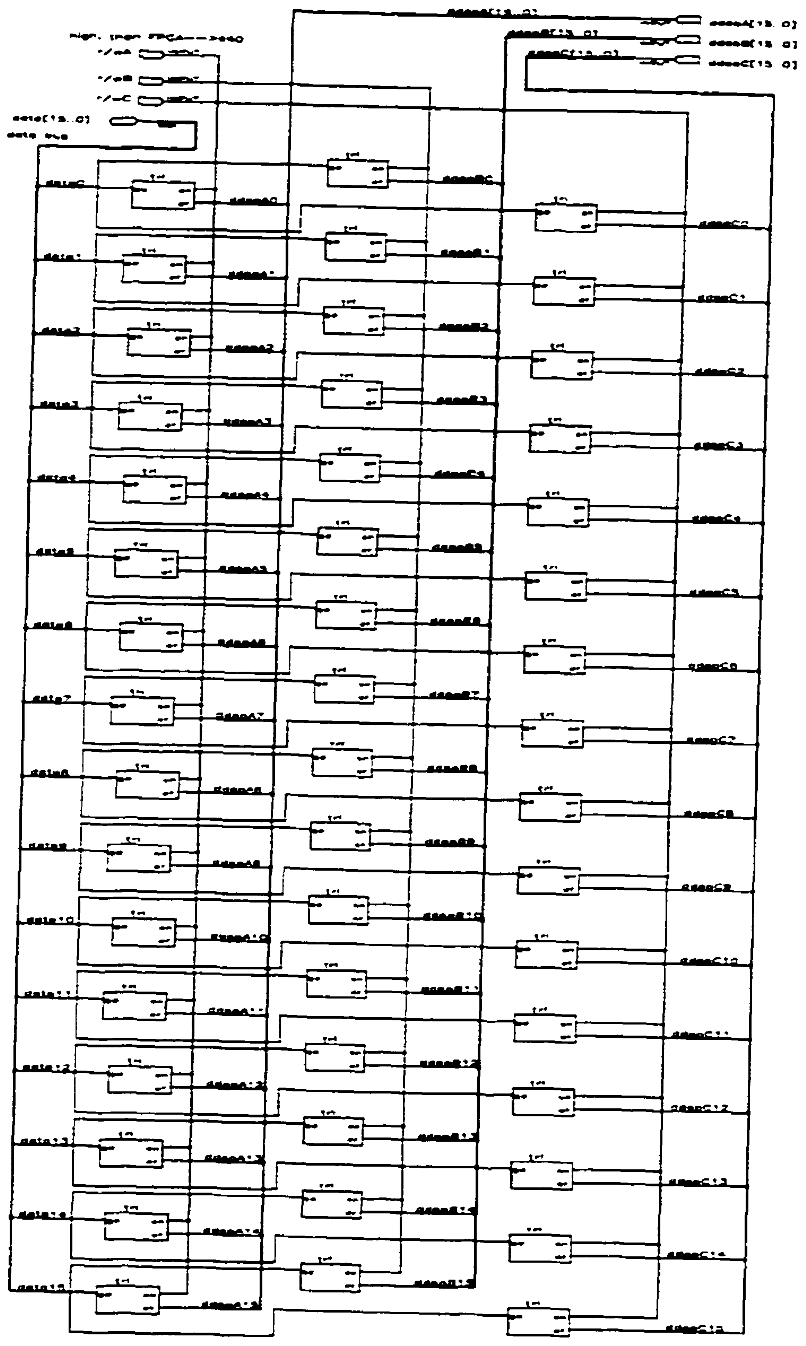


16busint.gdf

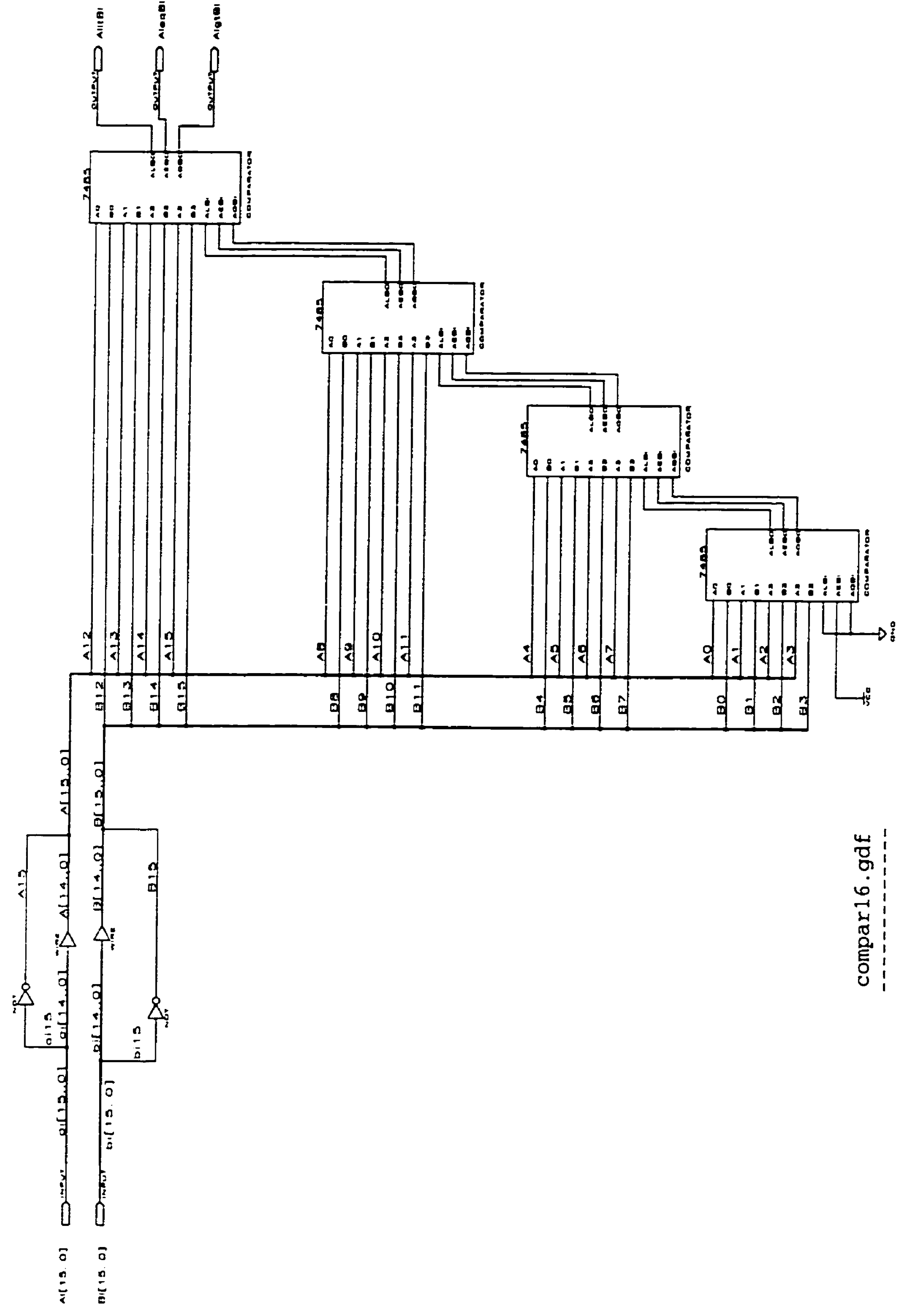


16pwm.gdf
counterpreload
low counter counts
high impuls to preload
frequency divider
High=coun12 counts down
Low=coun12 counts up
circuit detects wrong preload
or wrong calculated limit
half_sample_clk=high => YO=LIMITN
half_sample_clk=low => YO=LIMIT
CV > pos. Limit => set
=> DNUP=high_l starts hold
CV < neg. Limit => reset
=> DNUP=low l_starts hold
4 us leading
to half_sam_clk
DV_A[15..0]
DV_B[15..0]
DV_C[15..0]
COUNTER[15..0]
YO[15..0]
YON[15..0]
LIMIT[15..0]
LIMITN[15..0]
DNUP
hold
pre_lim
16coun
compar16

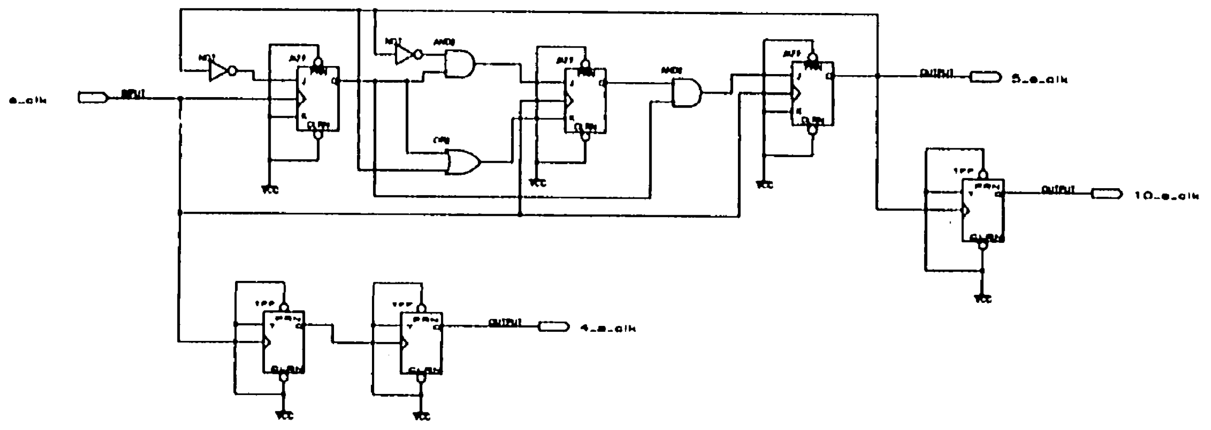

counter loads 1 every rising edge of samp_clk
InA[15..0]
flagJ
InB0 is the flag which denotes an event detection
InB1-InB3 are the switch status
InB[15..0]
counter loads 1 every rising edge of samp_clk
InC[15..0]
logic holds count when one event is detected
clears counter every rising edge of samp_clk
e1
e3
e5
logic holds count when two events are detected
16capture.gdf
ev1
clk
ev3
ev5
tclk0
80ns delay in the clrn



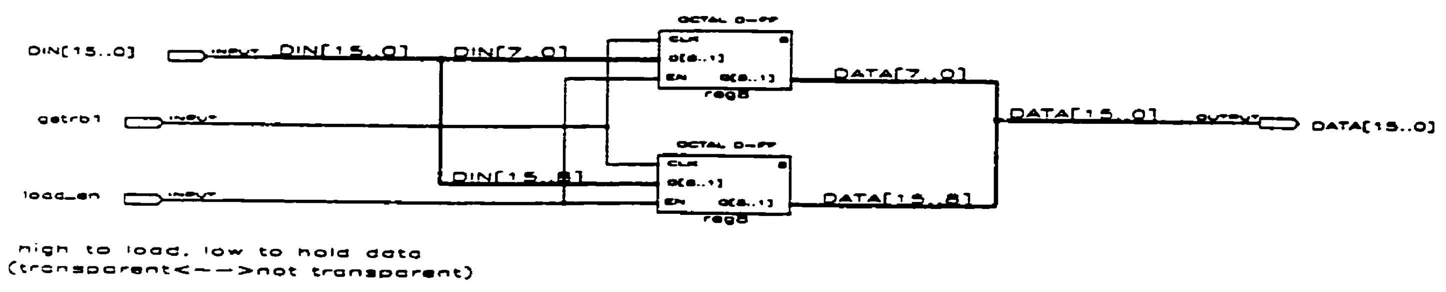
datadir.gdf



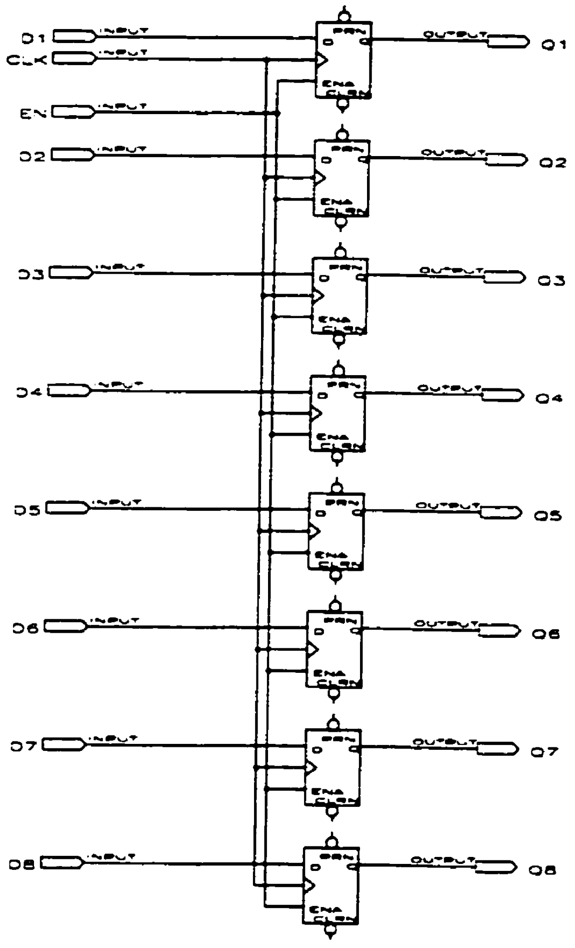
compar16.gdf


e_clk
INPUT
NOT
JKFF
AND2
OR2
VCC
OUTPUT
5_e_clk
TFF
PRN
CLRN
10_e_clk
4_e_clk

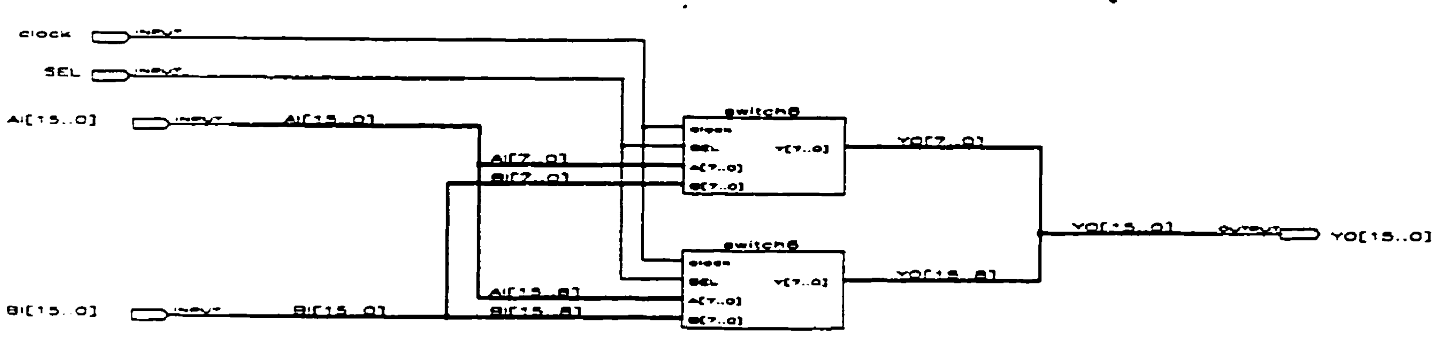
freq_div.gdf



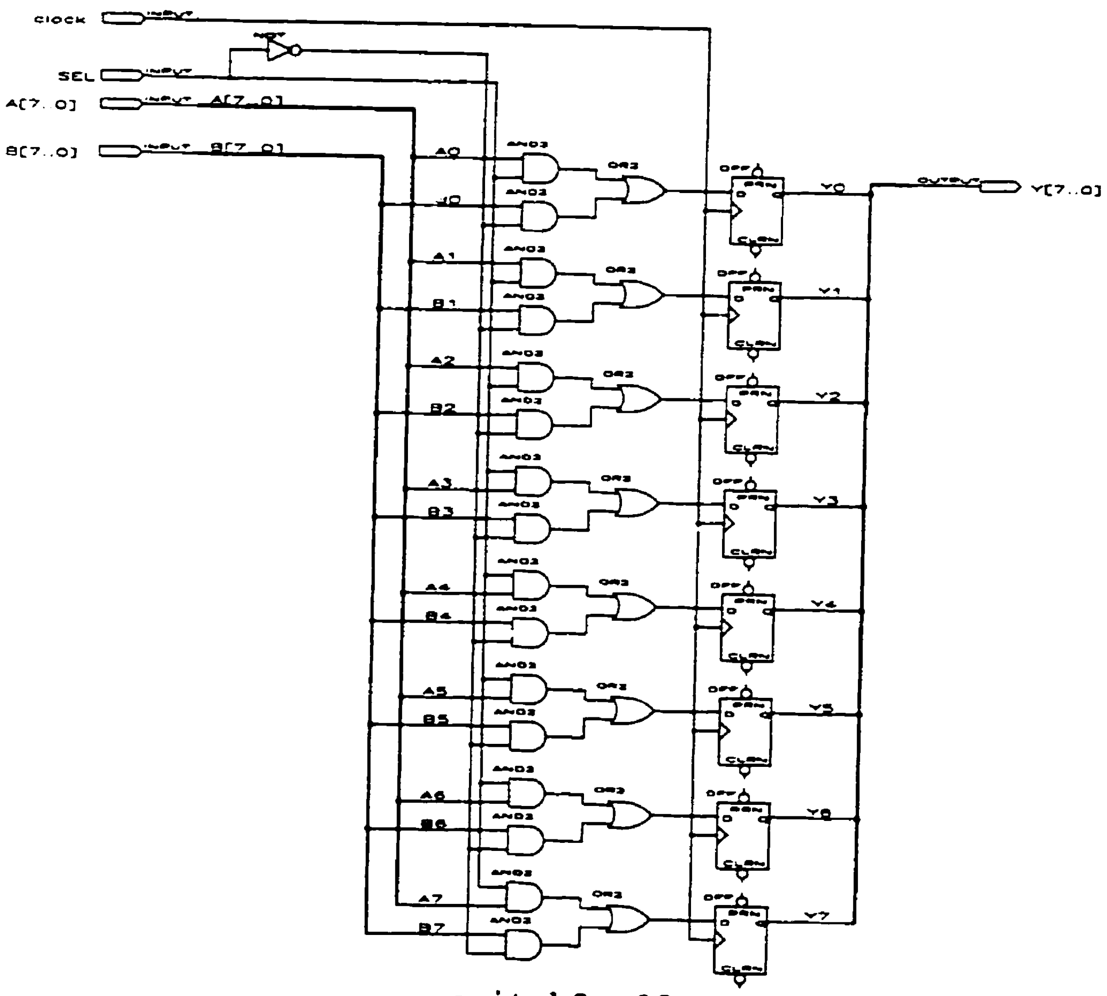
reg16.gdf



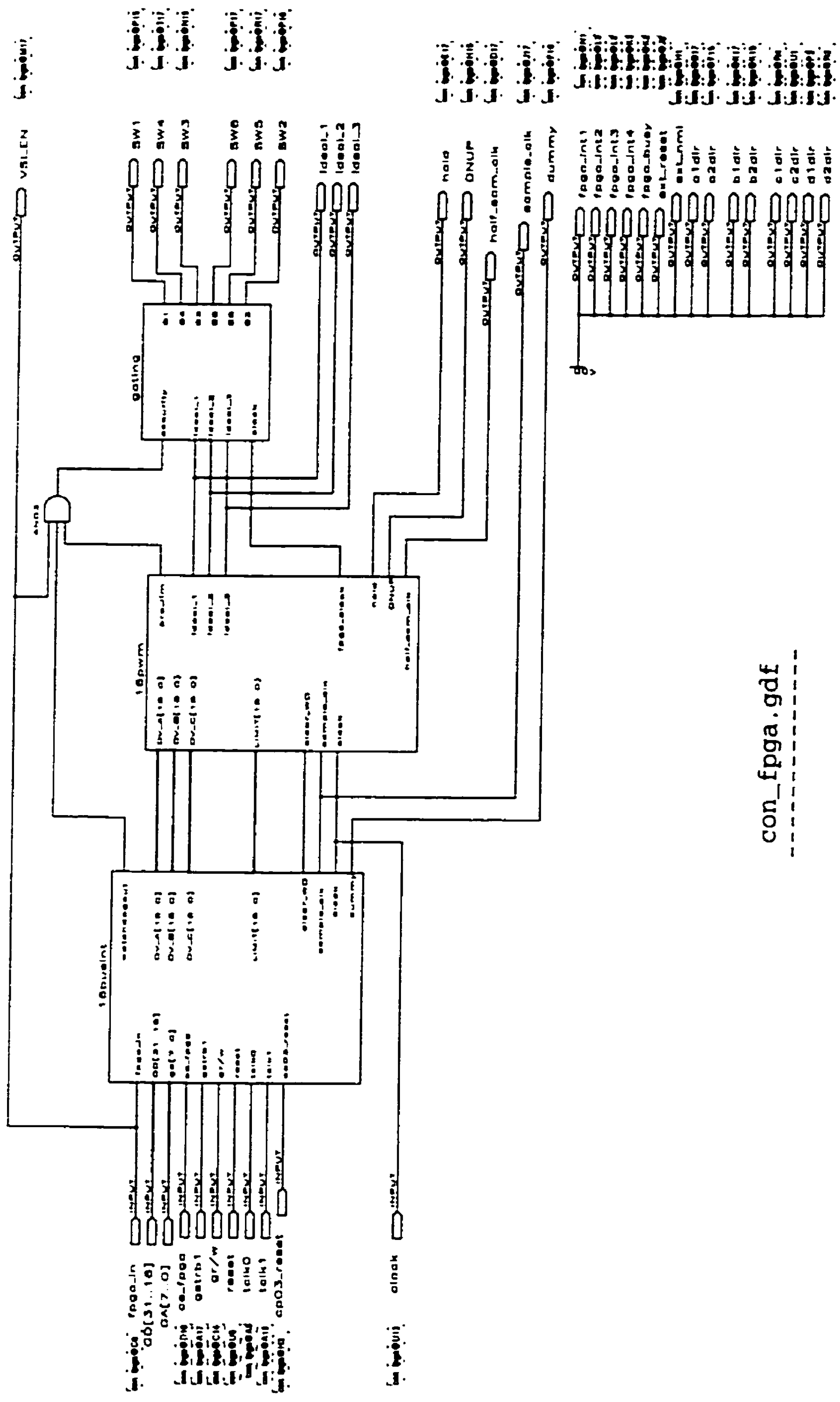
reg8.gdf



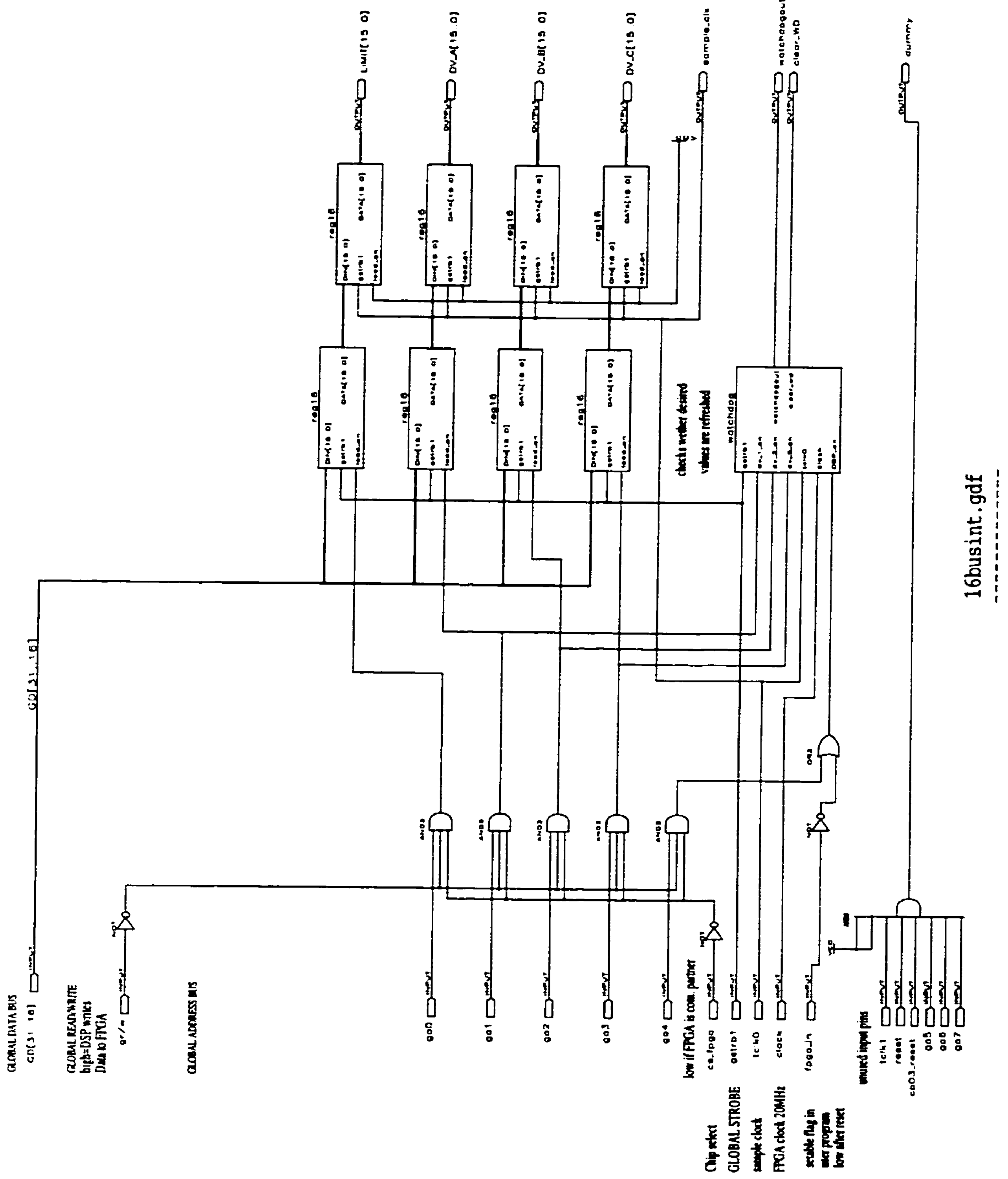
switch16.gdf

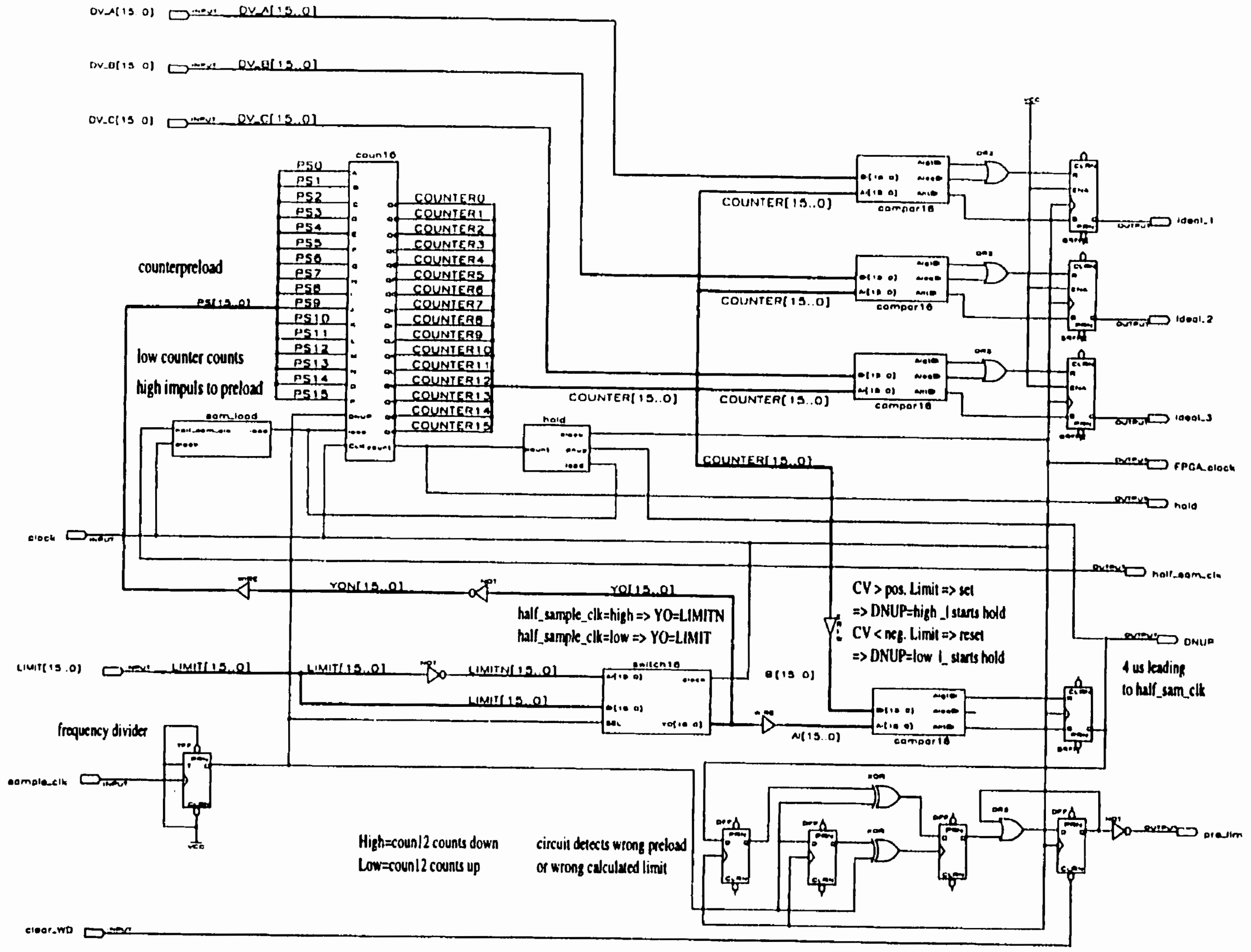


switch8.gdf

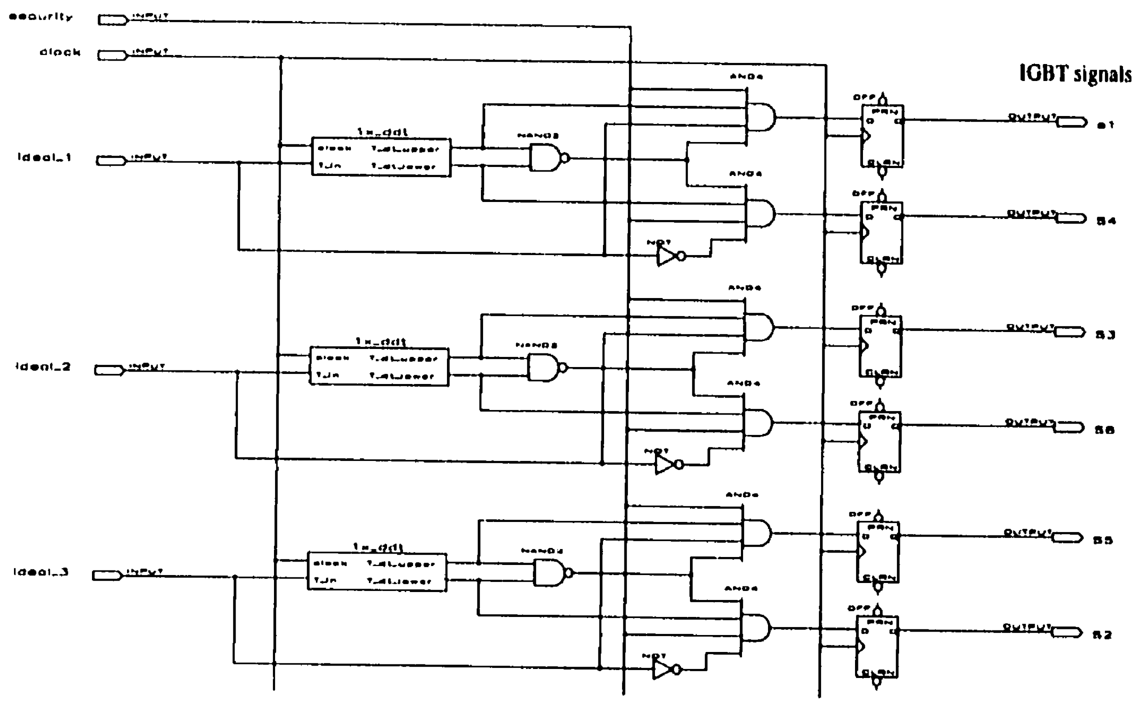


con_fpga.gdf

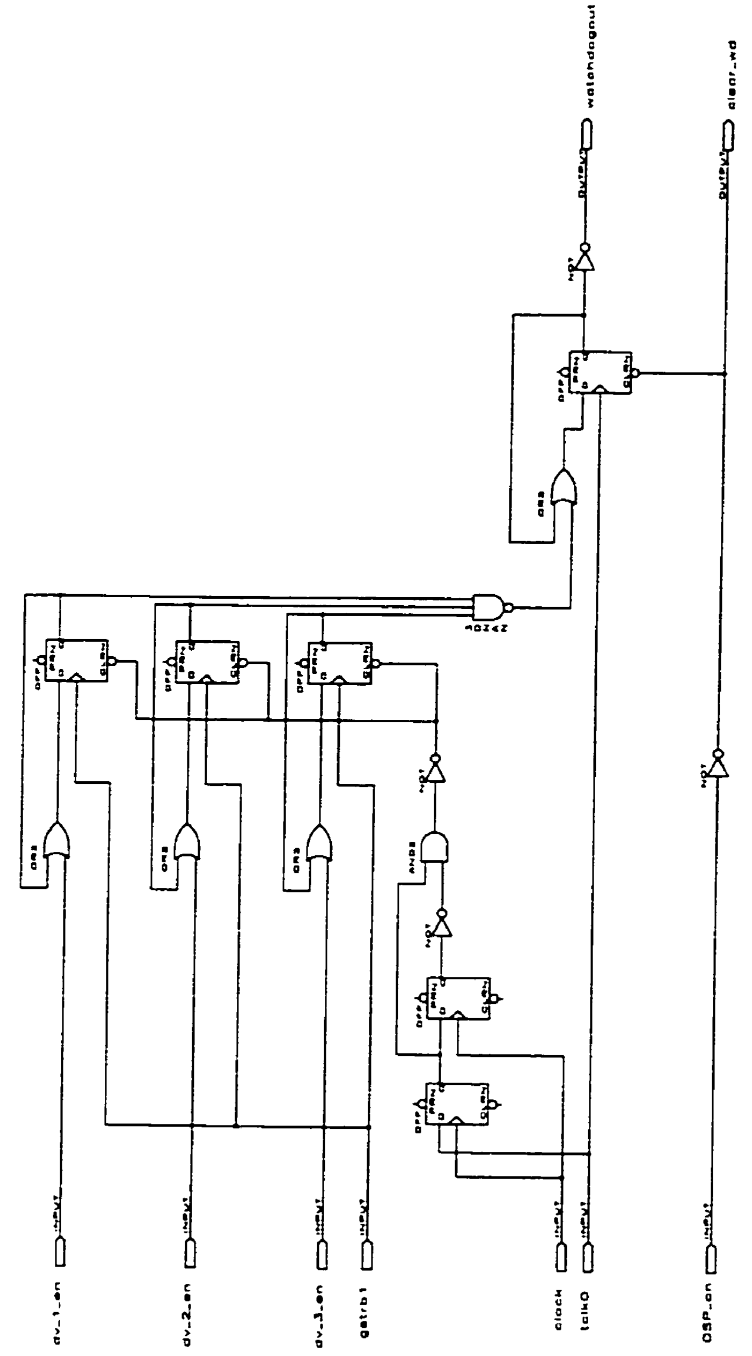

16busint.gdf
GLOBAL DATA BUS
GD[31..16]
GLOBAL READ/WRITE
high=DSP writes
Data to FPGA
gr/w
GLOBAL ADDRESS BUS
ga0
ga1
ga2
ga3
ga4
low if FPGA is com. partner
ce_fpga
gstrb1
tclk0
clock
fpga_in
Chip select
GLOBAL STROBE
sample clock
FPGA clock 20MHz
settable flag in
user program
low after reset
unused input pins
tclk1
reset
cpO3_reset
ga5
ga6
ga7
reg16
watchdog
checks weither desired
values are refreshed
LIMIT[15..0]
DV_A[15..0]
DV_B[15..0]
DV_C[15..0]
sample_clk
watchdogout
clear_WD
dummy


DV_A[15..0]
DV_B[15..0]
DV_C[15..0]
coun16
PS0
PS1
PS2
PS3
PS4
PS5
PS6
PS7
PS8
PS9
PS10
PS11
PS12
PS13
PS14
PS15
PS[15..0]
COUNTER0
COUNTER1
COUNTER2
COUNTER3
COUNTER4
COUNTER5
COUNTER6
COUNTER7
COUNTER8
COUNTER9
COUNTER10
COUNTER11
COUNTER12
COUNTER13
COUNTER14
COUNTER15
counterpreload
low counter counts
high impuls to preload
sam_load
hold
COUNTER[15..0]
compar16
ideal_1
ideal_2
ideal_3
FPGA_clock
hold
half_sam_clk
DNUP
clock
YON[15..0]
YO[15..0]
half_sample_clk=high => YO=LIMITN
half_sample_clk=low => YO=LIMIT
CV > pos. Limit => set
=> DNUP=high _| starts hold
CV < neg. Limit => reset
=> DNUP=low |_ starts hold
4 us leading
to half_sam_clk
LIMIT[15..0]
LIMITN[15..0]
switch16
B[15..0]
A[15..0]
frequency divider
sample_clk
High=coun12 counts down
Low=coun12 counts up
circuit detects wrong preload
or wrong calculated limit
pre_lim
clear_WD

16pwm.gdf


security
clock
ideal_1
ideal_2
ideal_3
INPUT
1x_ddt
clock
T_in
T_dt_upper
T_dt_lower
NAND2
AND4
NOT
DFF
PRN
CLRN
D
Q
OUTPUT
IGBT signals
s1
s4
s3
s6
s5
s2

gating.gdf



watchdog.gdf

Appendix D

Listing of Programs

This appendix lists software programs developed for off-line simulation, real-time simulation and experimental control of the three-phase PWM VSI system. All programs have been developed in the ANSI C language. The program list and a brief description is given below.

- `nor_c11.c` : Off-line simulation program using the fixed step-size method
- `intrp_c11.c` : Off-line simulation program using the proposed approach
- `rts_110.c` : Real-time simulation program using the proposed approach
- `con_110.c` : Program for experimental control

```
/**************************************************************
NOR_C11.C : Program for Off-line Simulation of a
Pulse Width Modulated (PWM) Voltage Source Inverter (VSI)
System Parameters : 110V(l-lrms), R=0.5ohm, L=3.0mH
Switching Frequency of VSI = 1kHz
Controller Sampling Frequency = 2kHz (Ts = 500us)
This program uses the Fixed Step-Size Algorithm for simulation
All arrays are cyclic buffers of length 3.
Program developed and tested by Venkata Dinavahi
Last updated on Feb. 13 '00
Compilation command : "cc intrp_c11.c -o intrp_c11 -lm"
"intrp_c11" is the object file
"sys_1.dat" file contains output data for the system
"con_1.dat" file contains output data for the controller
Note : This program was used to generate off-line
simulation results reported in Chapter #3
**************************************************************/

#include <stdio.h>
#include <math.h>
#include <time.h>

/*-------- Parameters of Simulation -----------*/

#define t 2.0
#define dt 50.e-6

#define PI 3.1415927
#define w 2*PI*60.0

#define R 0.5
#define L 3.0e-3
#define C 4900.e-6

#define Emag 110 * sqrt(2.0)/sqrt(3.0) /* Peak value of l-n voltage */
#define Vtri 1.0

#define Ts 500.e-6
#define Kid 0.8
#define Tid 50.e-3
#define ALPHA_d Kid*(Ts/(2.0*Tid)+1.0)
#define ALPHA0_d (ALPHA_d - 2.0*Kid)
#define Kiq 1.5
#define Tiq 50.e-3
#define ALPHA_q Kiq*(Ts/(2.0*Tiq)+1.0)
#define ALPHA0_q (ALPHA_q - 2.0*Kiq)
#define Kv -2.0
#define Tv 0.5
#define BETA Kv*(Ts/(2.0*Tv)+1.0)
#define BETA0 (BETA - 2.0*Kv)

#define Vd_ref 240.0
#define Iq_ref 0.0

#define N 3
#define M( x ) ( x % N )

main()
{
/*---------- Variable Declarations ----------*/

        clock_t start_clock, end_clock;
        int j = 0, k = 0;
        double t1 = 0.0;
        double A, B;
        double i;

        FILE *fp1, *fp2;

        int flag_a = 1, flag_b = 1, flag_c = 1, flag_down = 0, flag_pi = 0;
        int flag_control[N];
      /*int flag_dc_load = 0;
        int flag_clock = 0;*/
        int gat_1[N], gat_3[N], gat_5[N];
        int S1[N], S3[N], S5[N];
        double time[N], time_con[N];
        double crossing_a[N], crossing_b[N], crossing_c[N];
        double Xa[N], Xb[N], Xc[N];
        double grap_a[N], grap_b[N], grap_c[N];
        double control_a[N], control_b[N], control_c[N];
        double VaN[N], VbN[N], VcN[N];
        double Va[N], Vb[N], Vc[N], Vab[N];
        double Ea[N], Eb[N], Ec[N], Eab[N], Ebc[N];
        double I1[N], I2[N], I3[N], VD[N];
        double Ia[N], Ib[N], Ic[N], Idc[N], Vdc[N];
        double Eabs[N], Ebcs[N], Ias[N], Ibs[N], Vcaps[N];
        double I_re[N], I_im[N], Id[N], Iq[N], coswt[N], sinwt[N];
        double Vex_re[N], Vex_im[N];
        double Vs_re[N], Vs_im[N], Vs_abs[N];
        double Va_exc[N], Vb_exc[N], Vc_exc[N];
        double Vax_DV[N], Vbx_DV[N], Vcx_DV[N];
        double Id_err[N], Iq_err[N], out_d[N], out_q[N];
        double Ed[N], Eq[N], Edq[N];
        double ma_x[N], del_dc[N], ma[N], dc_ang[N];
        double out_vd[N], Vd_err[N], Idrefd[N];
        double Iq_ref1 = Iq_ref;
        double Iq1[N], Id1[N], Vcaps1[N], Idrefd1[N];
        int iter = 0;
        /*double Idc_load[N];*/

        fp1 = fopen("sys_n.dat","w");
        fp2 = fopen("con_n.dat","w");

/*---------- Initializations -------------------*/

        flag_control[j] = 0;
        gat_1[j] = 0; gat_3[j] = 0; gat_5[j] = 0;
        S1[j] = 0; S3[j] = 0; S5[j] = 0;
        time[j] = 0.0; time_con[k] = 0.0;
        crossing_a[k] = 0.0; crossing_b[k] = 0.0; crossing_c[k] = 0.0;
        Xa[k] = 0.0; Xb[k] = 0.0; Xc[k] = 0.0;
        grap_a[j] = 0.0; grap_b[j] = 0.0; grap_c[j] = 0.0;
        control_a[k] = 0.0; control_b[k] = 0.0; control_c[k] = 0.0;
        VaN[j] = 0.0; VbN[j] = 0.0; VcN[j] = 0.0;
        Va[j] = 0.0; Vb[j] = 0.0; Vc[j] = 0.0;
        Ea[j] = 0.0; Eb[j] = 0.0; Ec[j] = 0.0;
        Vab[j] = 0.0; Eab[j] = 0.0; Ebc[j] = 0.0;
        I1[j] = 0.0; I2[j] = 0.0; I3[j] = 0.0; VD[j] = 0.0;
        Ia[j] = 0.0; Ib[j] = 0.0; Ic[j] = 0.0; Idc[j] = 0.0; Vdc[j] = 220.0;
        Eabs[k] = 0.0; Ebcs[k] = 0.0; Ias[k] = 0.0; Ibs[k] = 0.0; Vcaps[k] = 220.0;
        I_re[k] = 0.0; I_im[k] = 0.0; Id[k] = 0.0; Iq[k] = 0.0;
        coswt[k] = 0.0; sinwt[k] = 0.0;
        Vs_re[k] = 0.0; Vs_im[k] = 0.0; Vs_abs[k] = 0.0;
        Vex_re[k] = 0.0; Vex_im[k] = 0.0;
        Va_exc[k] = 0.0; Vb_exc[k] = 0.0; Vc_exc[k] = 0.0;
        Vax_DV[k] = 0.0; Vbx_DV[k] = 0.0; Vcx_DV[k] = 0.0;
        Id_err[k] = 0.0; Iq_err[k] = 0.0; out_d[k] = 0.0; out_q[k] = 0.0;
        Ed[k] = 0.0; Eq[k] = 0.0; Edq[k] = 0.0;
        ma_x[k] = 0.8; del_dc[k] = 10.0; ma[k] = 0.8; dc_ang[k] = 10.0*PI/180.0;
        out_vd[k] = 0.0; Vd_err[k] = 0.0; Idrefd[k] = 0.0;
        Iq1[j] = 0.0; Id1[j] = 0.0; Vcaps1[j] = 220.0; Idrefd1[j] = 0.0;
        /* Idc_load[j] = 0.0;*/
        printf("Please stand by\n");

/* fprintf(fp2,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f\n",
   time_con[k],Eabs[k],Ebcs[k],Ias[k],I_re[k],I_im[k],Id[k],Id_err[k],
   Iq[k],Iq_err[k],out_d[k],out_q[k],Ed[k],Eq[k],ma[k],dc_ang[k],Vd_err[k],Vcaps[k]);

   fprintf(fp1,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f\n",time[j],
   Va[j],Ia[j],Vab[j],Vdc[j],Idc[j],grap_a[j],Iq_ref1,Iq1[j],Idrefd1[j],
   Id1[j],Vd_ref,Vcaps1[j],Idc_load[j]); */
```

```
        fprintf(fp2,"%f %f %f %f %f\n",
        time_con[k],Eabs[k],Ebcs[k],Id[k],Iq[k]);

        fprintf(fp1,"%f %f %f %f %f %f\n",time[j],Va[j],Ia[j],Vab[j],Vdc[j],Idc[j]);

/*------------ Definitions ----------------------*/

A = (2.0-(R/L)*dt)/(2.0+(R/L)*dt);
B = (dt/L)/(2.0+(R/L)*dt);

/*------------ Program Loop ----------------------*/

        printf( "Starting clock ...\n" );
        start_clock = clock();

        for (i = dt; i <= t;){

                j++;

                printf(".");

/*------------ Timing Block -------------------*/

                if (flag_a == 0){
                        if (time[M((j-1))] > crossing_a[M(k)]){
                                switch (flag_down) {
                                        case 1: gat_1[M(j)] = 0;
                                                flag_a = 1;
                                                break;
                                        case 0: gat_1[M(j)] = 1;
                                                flag_a = 1;
                                                break;
                                }
                } else{
                                gat_1[M(j)] = gat_1[M((j-1))];
                        }
                } else{
                        gat_1[M(j)] = gat_1[M((j-1))];
                }

                if (flag_b == 0){
                        if (time[M((j-1))] > crossing_b[M(k)]){
                                switch (flag_down) {
                                        case 1: gat_3[M(j)] = 0;
                                                flag_b = 1;
                                                break;
                        case 0: gat_3[M(j)] = 1;
                                flag_b = 1;
                                break;
                    }
                } else{
                    gat_3[M(j)] = gat_3[M((j-1))];
                  }
            } else{
                gat_3[M(j)] = gat_3[M((j-1))];
              }

            if (flag_c == 0){
                if (time[M((j-1))] > crossing_c[M(k)]){
                    switch (flag_down) {
                        case 1: gat_5[M(j)] = 0;
                                flag_c = 1;
                                break;
                        case 0: gat_5[M(j)] = 1;
                                flag_c = 1;
                                break;
                    }
             } else{
                    gat_5[M(j)] = gat_5[M((j-1))];
                  }
            } else{
                gat_5[M(j)] = gat_5[M((j-1))];
              }

/*------------ VSI Model for Time (j) ----------------*/

            if (gat_1[M(j)] == 1){
                VaN[M(j)] = Vdc[M((j-1))];
                        S1[M(j)] = 1;
            } else{
                VaN[M(j)] = 0.0;
                        S1[M(j)] = 0;
              }
            if (gat_3[M(j)] == 1){
                VbN[M(j)] = Vdc[M((j-1))];
                        S3[M(j)] = 1;
            } else{
                VbN[M(j)] = 0.0;
                        S3[M(j)] = 0;
              }
            if (gat_5[M(j)] == 1){
                VcN[M(j)] = Vdc[M((j-1))];
                        S5[M(j)] = 1;
            } else{
                VcN[M(j)] = 0.0;
                        S5[M(j)] = 0;
              }

/*------------- Solution of System Equations of Time (j) ----------*/

        I1[M(j)] = A * Ia[M((j-1))] + B * (Va[M((j-1))] - Ea[M((j-1))]);
        I2[M(j)] = A * Ib[M((j-1))] + B * (Vb[M((j-1))] - Eb[M((j-1))]);
        I3[M(j)] = A * Ic[M((j-1))] + B * (Vc[M((j-1))] - Ec[M((j-1))]);

                VD[M(j)] = -(dt/(2.0*C)) * Idc[M((j-1))] + Vdc[M((j-1))];

        Ea[M(j)] = Emag * cos(w*i);
        Eb[M(j)] = Emag * cos(w*i-(120.0*PI/180.0));
        Ec[M(j)] = Emag * cos(w*i+(120.0*PI/180.0));

                Va[M(j)] = (2.0/3.0) * VaN[M(j)] - (1.0/3.0) * ( VbN[M(j)] +
VcN[M(j)] );
                Vb[M(j)] = (2.0/3.0) * VbN[M(j)] - (1.0/3.0) * ( VaN[M(j)] +
VcN[M(j)] );
                Vc[M(j)] = (2.0/3.0) * VcN[M(j)] - (1.0/3.0) * ( VaN[M(j)] +
VbN[M(j)] );

        Ia[M(j)] = I1[M(j)] + (Va[M(j)]-Ea[M(j)]) * B;
        Ib[M(j)] = I2[M(j)] + (Vb[M(j)]-Eb[M(j)]) * B;
        Ic[M(j)] = I3[M(j)] + (Vc[M(j)]-Ec[M(j)]) * B;

                Idc[M(j)] = S1[M(j)] * Ia[M(j)] + S3[M(j)] * Ib[M(j)] + S5[M(j)]
* Ic[M(j)];

                Vdc[M(j)] = -(dt/(2.0*C)) * Idc[M(j)] + VD[M(j)];


                Vab[M(j)] = Va[M(j)] - Vb[M(j)];
                Eab[M(j)] = Ea[M(j)] - Eb[M(j)];
                Ebc[M(j)] = Eb[M(j)] - Ec[M(j)];

                time[M(j)] = i;

                /* For DC side transient */

                /* if (time[M(j)] >= 0.5){ flag_pi = 1; } */

                  if ((time[M(j)] >= 3.0) && (time[M(j)] <= 4.0)){
                  flag_dc_load = 1;
                  } else{
```

```
                flag_dc_load = 0;
                } */

              /* For Iq Transient */
              /* if (time[M(j)] >= 0.5){flag_pi = 1;}
              if ((time[M(j)] >= 2.0) && (time[M(j)] <= 3.0)){
                 Iq_ref1 = 10.0;
              }else{
                 Iq_ref1 = Iq_ref;}*/

/*----------- Control Algorithm -------------------*/

              t1 += dt;

              if (t1 > Ts){

                      k++;

                Eabs[M(k)] = Eab[M((j-1))];
                Ebcs[M(k)] = Ebc[M((j-1))];
                Ias[M(k)] = Ia[M((j-1))];
                Ibs[M(k)] = Ib[M((j-1))];
                Vcaps[M(k)] = Vdc[M((j-1))];

              Vs_re[M(k)] = (1.0/1.732) * ( 0.87 * Eabs[M((k-1))] + (
Ebcs[M((k-1))]*1.15 + Eabs[M((k-1))]*0.58 ) * 0.5);
              Vs_im[M(k)] = (1.0/1.732) * ( -0.5 * Eabs[M((k-1))] + (
Ebcs[M((k-1))]*1.15 + Eabs[M((k-1))]*0.58 ) * 0.87);
              Vs_abs[M(k)] = sqrt(Vs_re[M(k)]*Vs_re[M(k)] +
Vs_im[M(k)]*Vs_im[M(k)]);

              I_re[M(k)] = Ias[M((k-1))];
              I_im[M(k)] = 0.58 * Ias[M((k-1))] + 1.15 * Ibs[M((k-1))];

                    if (Vs_abs[M(k)] == 0.0){
                          coswt[M(k)] = 0.0;
                          sinwt[M(k)] = 0.0;
                    } else{
                          coswt[M(k)] = Vs_re[M(k)]/Vs_abs[M(k)];
                          sinwt[M(k)] = Vs_im[M(k)]/Vs_abs[M(k)];
                    }
              Id[M(k)] = I_re[M(k)] * coswt[M(k)] + I_im[M(k)] * sinwt[M(k)];
              Iq[M(k)] = -I_re[M(k)] * sinwt[M(k)] + I_im[M(k)] * coswt[M(k)];


                    Iq_err[M(k)] = Iq_ref1 - Iq[M(k)];
                    Vd_err[M(k)] = Vd_ref - Vcaps[M((k-1))];
```

```
                         if (flag_pi == 1){
                                   out_vd[M(k)] = out_vd[M((k-1))] + BETA
* Vd_err[M(k)] + BETA0 * Vd_err[M((k-1))];
                                   Idrefd[M(k)] = out_vd[M(k)];
                        } else{
                                   out_vd[M(k)] = out_vd[M((k-1))];
                                   Idrefd[M(k)] = Idrefd[M((k-1))];
                        }
                              Id_err[M(k)] = Idrefd[M(k)] - Id[M(k)];

                         if (flag_pi == 1){
                                   out_d[M(k)] = out_d[M((k-1))] +
ALPHA_d * Id_err[M(k)] + ALPHA0_d * Id_err[M((k-1))];
                                   out_q[M(k)] = out_q[M((k-1))] +
ALPHA_q * Iq_err[M(k)] + ALPHA0_q * Iq_err[M((k-1))];
                                   Ed[M(k)] = out_d[M(k)] - Iq[M(k)] * w *
L + Vs_abs[M(k)];
                                   Eq[M(k)] = out_q[M(k)] + Id[M(k)] * w *
L;
                                   Edq[M(k)] = sqrt(Ed[M(k)] * Ed[M(k)] +
Eq[M(k)] * Eq[M(k)]);
                                   ma_x[M(k)] = Edq[M(k)] / (0.7071 *
Vcaps[M((k-1))]);
                                   del_dc[M(k)] = atan(Eq[M(k)]/Ed[M(k)]) ;
                        } else{
                                   out_d[M(k)] = out_d[M((k-1))];
                                   out_q[M(k)] = out_q[M((k-1))];
                                   Ed[M(k)] = Ed[M((k-1))];
                                   Eq[M(k)] = Eq[M((k-1))];
                                   Edq[M(k)] = Edq[M((k-1))];
                                   ma_x[M(k)] = 0.8;
                                   del_dc[M(k)] = 10.0 ;
                        }

                        ma[M(k)] = ma_x[M(k)];
                        dc_ang[M(k)] = del_dc[M(k)]*PI/180.0;
                        /* For open loop del_dc has to be multiplied by
PI/180.
                             For closed loop del_dc is already in radians due to
the
                             atan function */

/*--------------- Generation of Control Signals --------------*/

                   Vex_re[M(k)] = Vs_re[M(k)] * cos(dc_ang[M(k)]) - Vs_im[M(k)] *
sin(dc_ang[M(k)]);
```

```
                Vex_im[M(k)] = Vs_re[M(k)] * sin(dc_ang[M(k)]) + Vs_im[M(k)]
* cos(dc_ang[M(k)]);

                Va_exc[M(k)] = Vex_re[M(k)];
                Vb_exc[M(k)] = -0.5 * Vex_re[M(k)] + 0.87 * Vex_im[M(k)];
                Vc_exc[M(k)] = - Va_exc[M(k)] - Vb_exc[M(k)];

                  if (Vs_abs[M(k)] == 0.0){
                      Vax_DV[M(k)] = 0.0;
                      Vbx_DV[M(k)] = 0.0;
                      Vcx_DV[M(k)] = 0.0;
                  } else{
                    /* Vax_DV[M(k)] = ma[M(k)] * Va_exc[M(k)] * 0.5 *
Vcaps[M(k)] / Vs_abs[M(k)];
                Vbx_DV[M(k)] = ma[M(k)] * Vb_exc[M(k)] * 0.5 * Vcaps[M(k)]
/ Vs_abs[M(k)];
                Vcx_DV[M(k)] = ma[M(k)] * Vc_exc[M(k)] * 0.5 * Vcaps[M(k)]
/ Vs_abs[M(k)]; */
                Vax_DV[M(k)] = ma[M(k)] * Va_exc[M(k)] / Vs_abs[M(k)];
                Vbx_DV[M(k)] = ma[M(k)] * Vb_exc[M(k)] / Vs_abs[M(k)];
                Vcx_DV[M(k)] = ma[M(k)] * Vc_exc[M(k)] / Vs_abs[M(k)];
                     }
                     control_a[M(k)] = Vax_DV[M(k)];
                     control_b[M(k)] = Vbx_DV[M(k)];
                     control_c[M(k)] = Vcx_DV[M(k)];

/*-------------- Calculation of Gate Timing --------------*/

                     if (flag_down == 1){

                         Xa[M(k)] = (-Ts/(2.0*Vtri)) * ( control_a[M(k)]
- Vtri);

                         Xb[M(k)] = (-Ts/(2.0*Vtri)) * (
control_b[M(k)] - Vtri);
                         Xc[M(k)] = (-Ts/(2.0*Vtri)) * ( control_c[M(k)]
- Vtri);


                         crossing_a[M(k)] = time[M((j-1))] + Xa[M(k)];
                         crossing_b[M(k)] = time[M((j-1))] + Xb[M(k)];
                         crossing_c[M(k)] = time[M((j-1))] + Xc[M(k)];

                         flag_down = 0;
                     } else{

                         Xa[M(k)] = (Ts/(2.0*Vtri)) * ( control_a[M(k)]
+ Vtri);

                         Xb[M(k)] = (Ts/(2.0*Vtri)) * ( control_b[M(k)]
```

```
+ Vtri);
                         Xc[M(k)] = (Ts/(2.0*Vtri)) * ( control_c[M(k)]
+ Vtri);

                         crossing_a[M(k)] = time[M((j-1))] + Xa[M(k)];
                         crossing_b[M(k)] = time[M((j-1))] + Xb[M(k)];
                         crossing_c[M(k)] = time[M((j-1))] + Xc[M(k)];

                         flag_down = 1;
                       }
                     grap_a[M(j)] = control_a[M(k)];
                     grap_b[M(j)] = control_b[M(k)];
                     grap_c[M(j)] = control_c[M(k)];

                     Iq1[M(j)] = Iq[M(k)];
                     Id1[M(j)] = Id[M(k)];
                     Idrefd1[M(j)] = Idrefd[M(k)];
                     Vcaps1[M(j)] = Vcaps[M(k)];

                     t1 -= Ts;

                     time_con[M(k)] = time[M((j-1))];

                     flag_a = 0;
                     flag_b = 0;
                     flag_c = 0;
                     flag_control[M(j)] = 1;

                     fprintf(fp2,"%f %f %f %f %f\n",
                     time_con[M(k)],Eabs[M(k)],Ebcs[M(k)],Id[M(k)],Iq[M(k)]);


            /* fprintf(fp2,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f\n",
                time_con[M(k)],Eabs[M(k)],Ebcs[M(k)],Ias[M(k)],I_re[M(k)],I_im[M(k)],
                Id[M(k)],Id_err[M(k)],Iq[M(k)],Iq_err[M(k)],out_d[M(k)],out_q[M(k)],
                Ed[M(k)],Eq[M(k)],ma[M(k)],dc_ang[M(k)],Vd_err[M(k)],Vcaps[M(k)]);
*/

                } else{
                    grap_a[M(j)] = grap_a[M((j-1))];
                    grap_b[M(j)] = grap_b[M((j-1))];
                    grap_c[M(j)] = grap_c[M((j-1))];

                    flag_control[M(j)] = 0;
```

```
                        Iq1[M(j)] = Iq1[M((j-1))];
                        Id1[M(j)] = Id1[M((j-1))];
                        Idrefd1[M(j)] = Idrefd1[M((j-1))];
                        Vcaps1[M(j)] = Vcaps1[M((j-1))];
                  }

            /* fprintf(fp1,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f\n",
                  time[M(j)], Va[M(j)],Ia[M(j)], Vab[M(j)], Vdc[M(j)],Idc[M(j)],
                  grap_a[M(j)],Iq_ref1,Iq1[M(j)],Idrefd1[M(j)],Id1[M(j)], Vd_ref,
                  Vcaps1[M(j)],Idc_load[M(j)]); */

                  fprintf(fp1,"%f %f %f %f %f %f\n",time[M(j)],Va[M(j)],Ia[M(j)],
                  Vab[M(j)],Vdc[M(j)],Idc[M(j)]);

            i += dt;
            iter++;
            }
      end_clock = clock();

      printf("...  done.  \n");
      printf("Iterations = %d\n",iter);

      fclose(fp1);
      fclose(fp2);
      printf("Processor time used :  %g sec.\n",
            (end_clock - start_clock) / (double) CLOCKS_PER_SEC);
      return 0;
      }
```

```
/***********************************************************
INTRP_c11.c : Program for Off-line Simulation of a
Pulse Width Modulated (PWM) Voltage Source Inverter (VSI)
System Parameters : 110V(l-lrms), R=0.5ohm, L=3.0mH
Switching Frequency of VSI = 1kHz
Controller Sampling Frequency = 2kHz (Ts = 500us)
The program incorporates the proposed algorithm for
accounting discrete firing pulses.
All arrays are cyclic buffers of length 3.
Program developed and tested by Venkata Dinavahi
Last updated on Feb. 13 '00
Compilation command : "cc intrp_c11.c -o intrp_c11 -lm"
"intrp_c11" is the object file
"sys_1.dat" file contains output data for the system
"con_1.dat" file contains output data for the controller
Note : This program was used to generate off-line
simulation results reported in Chapter #3
***********************************************************/

#include <stdio.h>
#include <math.h>
#include <time.h>

/*________ Parameters of Simulation ____________*/

#define t 2.0
#define dt 50.0e-6

#define PI 3.1415927
#define w 2.0*PI*60.0

#define R 0.5
#define L 3.0e-3
#define C 4900.e-6

#define Emag 110.0 * sqrt(2.0)/sqrt(3.0) /* Peak value of l-n voltage */
#define Vtri 1.0

#define Ts 500.e-6
#define Kid 0.8
#define Tid 50.e-3
#define ALPHA_d Kid*(Ts/(2.0*Tid)+1.0)
#define ALPHA0_d (ALPHA_d - 2.0*Kid)
#define Kiq 1.5
#define Tiq 50.e-3
#define ALPHA_q Kiq*(Ts/(2.0*Tiq)+1.0)
#define ALPHA0_q (ALPHA_q - 2.0*Kiq)
```

```
#define Kv -2.0
#define Tv 0.5
#define BETA Kv*(Ts/(2.0*Tv)+1.0)
#define BETA0 (BETA - 2.0*Kv)

#define Vd_ref 240.0
#define Iq_ref -10.0

#define N 3
#define M( x ) ( x % N )

main()
{
/*__________ Variable Declarations __________*/

        clock_t start_clock, end_clock;
        int j = 0, k = 0;
        double t1 = 0.0;
        double A, B, A_o, B_o;
        double step_1, step_2;
        double i;

        FILE *fp1, *fp2;

        int flag_a = 1, flag_b = 1, flag_c = 1, flag_down = 0, flag_pi = 0;
        int flag_control[N];
        int flag_ab = 1, flag_bc = 1, flag_ac = 1, flag_interp = 0;
        int gat_1[N], gat_3[N], gat_5[N];
        int S1[N], S3[N], S5[N];
        double time[N], time_con[N];
        double crossing_a[N], crossing_b[N], crossing_c[N];
        double Xa[N], Xb[N], Xc[N];
        double grap_a[N], grap_b[N], grap_c[N];
        double control_a[N], control_b[N], control_c[N];
        double VaN[N], VbN[N], VcN[N];
        double Va[N], Vb[N], Vc[N], Vab[N];
        double Ea[N], Eb[N], Ec[N], Eab[N], Ebc[N];
        double I1[N], I2[N], I3[N], VD[N];
        double Ia[N], Ib[N], Ic[N], Idc[N], Vdc[N];
        double Eabs[N], Ebcs[N], Ias[N], Ibs[N], Vcaps[N];
        double I_re[N], I_im[N], Id[N], Iq[N], coswt[N], sinwt[N];
        double Vex_re[N], Vex_im[N];
        double Vs_re[N], Vs_im[N], Vs_abs[N];
        double Va_exc[N], Vb_exc[N], Vc_exc[N];
        double Vax_DV[N], Vbx_DV[N], Vcx_DV[N];
        double Id_err[N], Iq_err[N], out_d[N], out_q[N];
        double Ed[N], Eq[N], Edq[N];
        double ma_x[N], del_dc[N], ma[N], dc_ang[N];
        double Vai = 0.0, Vbi = 0.0, Vci = 0.0, Iai = 0.0, Ibi = 0.0, Ici = 0.0;
        double Eai = 0.0, Ebi = 0.0, Eci = 0.0, Vdci = 0.0, Idci = 0.0, VDi = 0.0;
        double Iqf[N], Iqf0[N], Iq0[N];

        double out_vd[N], Vd_err[N], Idrefd[N];
        double Iq_ref1 = Iq_ref;
        double Iq1[N], Id1[N], Vcaps1[N], Idrefd1[N];
        int iter = 0;

        fp1 = fopen("sys_i.dat","w");
        fp2 = fopen("con_i.dat","w");

/*-------- Initializations ------------------*/

        flag_control[j] = 0;
        gat_1[j] = 0; gat_3[j] = 0; gat_5[j] = 0;
        S1[j] = 0; S3[j] = 0; S5[j] = 0;
        time[j] = 0.0; time_con[k] = 0.0;
        crossing_a[k] = 0.0; crossing_b[k] = 0.0; crossing_c[k] = 0.0;
        Xa[k] = 0.0; Xb[k] = 0.0; Xc[k] = 0.0;
        grap_a[j] = 0.0; grap_b[j] = 0.0; grap_c[j] = 0.0;
        control_a[k] = 0.0; control_b[k] = 0.0; control_c[k] = 0.0;
        VaN[j] = 0.0; VbN[j] = 0.0; VcN[j] = 0.0;
        Va[j] = 0.0; Vb[j] = 0.0; Vc[j] = 0.0;
        Ea[j] = 0.0; Eb[j] = 0.0; Ec[j] = 0.0;
        Vab[j] = 0.0; Eab[j] = 0.0; Ebc[j] = 0.0;
        I1[j] = 0.0; I2[j] = 0.0; I3[j] = 0.0; VD[j] = 0.0;
        Ia[j] = 0.0; Ib[j] = 0.0; Ic[j] = 0.0; Idc[j] = 0.0; Vdc[j] = 220.0;
        Eabs[k] = 0.0; Ebcs[k] = 0.0; Ias[k] = 0.0; Ibs[k] = 0.0; Vcaps[k] = 220.0;
        I_re[k] = 0.0; I_im[k] = 0.0; Id[k] = 0.0; Iq[k] = 0.0;
        coswt[k] = 0.0; sinwt[k] = 0.0;
        Vs_re[k] = 0.0; Vs_im[k] = 0.0; Vs_abs[k] = 0.0;
        Vex_re[k] = 0.0; Vex_im[k] = 0.0;
        Va_exc[k] = 0.0; Vb_exc[k] = 0.0; Vc_exc[k] = 0.0;
        Vax_DV[k] = 0.0; Vbx_DV[k] = 0.0; Vcx_DV[k] = 0.0;
        Id_err[k] = 0.0; Iq_err[k] = 0.0; out_d[k] = 0.0; out_q[k] = 0.0;
        Ed[k] = 0.0; Eq[k] = 0.0; Edq[k] = 0.0;
        ma_x[k] = 0.8; del_dc[k] = 10.0; ma[k] = 0.8; dc_ang[k] = 10.0*PI/180.0;
        out_vd[k] = 0.0; Vd_err[k] = 0.0; Idrefd[k] = 0.0;
        Iq1[j] = 0.0; Id1[j] = 0.0; Vcaps1[j] = 220.0; Idrefd1[j] = 0.0;
        Iqf0[k] = 0.0; Iq0[k] = 0.0; Iqf[k] = 0.0;

        printf("Please stand by\n");

/* fprintf(fp2,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f\n",
     time_con[k],Eabs[k],Ebcs[k],Ias[k],I_re[k],I_im[k],Id[k],Id_err[k],Iq[k],
     Iq_err[k],out_d[k],out_q[k],Ed[k],Eq[k],ma[k],dc_ang[k],Vd_err[k],Vcaps[k],Iqf[k]);
*/

        fprintf(fp2,"%f %f %f %f %f\n",time_con[k],Eabs[k],Ebcs[k],Id[k],Iq[k]);
        fprintf(fp1,"%f %f %f %f %f %f\n",time[j],Va[j],Ia[j],Vab[j],Vdc[j],Idc[j]);

/* fprintf(fp1,"%f %f %f %f %f %f %f %f %f %f %f %f %f\n",time[j],Va[j],
        Ia[j],Vab[j],Vdc[j],Idc[j],grap_a[j],Iq_ref1,Iq1[j],Idrefd1[j],Id1[j],
        Vd_ref,Vcaps1[j]); */

/*----------- Definitions ----------------------*/

A_o = (2.0-(R/L)*dt)/(2.0+(R/L)*dt);
B_o = (dt/L)/(2.0+(R/L)*dt);

/*----------- Program Loop ----------------------*/

        printf("Starting clock ...");
        start_clock = clock();

        for (i = dt; i <= t;){

                j++;

                printf(".");

/*----------- Timing Block ------------------*/

        if (flag_a == 0){

                if (time[M((j-1))] > crossing_a[M(k)]){

                        if ((time[M((j-1))] > crossing_b[M(k)]) && (flag_b ==
0)){

                                if (crossing_a[M(k)] > crossing_b[M(k)]){

                                        step_1 = crossing_b[M(k)] -
time[M((j-2))];

                                        step_2 = 2*dt - step_1;

                                        switch (flag_down) {
                                                case 1: gat_1[M(j)] = 0;
                                                        gat_3[M(j)] = 0;
                                                        break;
                                                case 0: gat_1[M(j)] = 1;
                                                        gat_3[M(j)] = 1;
                                        break;
                                }
                        }
                        if (crossing_a[M(k)] < crossing_b[M(k)]){

                                step_1 = crossing_a[M(k)] -
time[M((j-2))];

                                step_2 = 2*dt - step_1;

                                switch (flag_down) {
                                        case 1: gat_1[M(j)] = 0;
                                                gat_3[M(j)] = 0;
                                                break;
                                        case 0: gat_1[M(j)] = 1;
                                                gat_3[M(j)] = 1;
                                                break;
                                }
                        }
                        flag_a = 1; flag_b = 1; flag_ab = 1; flag_interp
= 1;

                } else if ((time[M((j-1))] > crossing_c[M(k)]) && (flag_c
== 0)){

                        if (crossing_a[M(k)] > crossing_c[M(k)]){

                                step_1 = crossing_c[M(k)] -
time[M((j-2))];

                                step_2 = 2*dt - step_1;

                                switch (flag_down) {
                                        case 1: gat_1[M(j)] = 0;
                                                gat_5[M(j)] = 0;
                                                break;
                                        case 0: gat_1[M(j)] = 1;
                                                gat_5[M(j)] = 1;
                                                break;
                                }
                        }
                        if (crossing_a[M(k)] < crossing_c[M(k)]){

                                step_1 = crossing_a[M(k)] -
time[M((j-2))];

                                step_2 = 2*dt - step_1;

                                switch (flag_down) {
                                        case 1: gat_1[M(j)] = 0;
```

```
                                                gat_5[M(j)] = 0;
                                                break;
                                        case 0: gat_1[M(j)] = 1;
                                                gat_5[M(j)] = 1;
                                                break;
                                }
                        }
                        flag_a = 1; flag_c = 1; flag_ac = 1; flag_interp
= 1;
                } else{

                        step_1 = crossing_a[M(k)] - time[M((j-2))];
                        step_2 = 2*dt - step_1;

                        switch (flag_down) {
                                case 1: gat_1[M(j)] = 0;
                                        break;
                                case 0: gat_1[M(j)] = 1;
                                        break;
                                } flag_a = 1; flag_interp = 1;
                        }
                } else{
                        gat_1[M(j)] = gat_1[M((j-1))];
                }
        } else{
                gat_1[M(j)] = gat_1[M((j-1))];
        }

/*------------- B-Group Timing ----------------*/

        if (flag_ab == 0){

                if (flag_b == 0){

                        if (time[M((j-1))] > crossing_b[M(k)]){

                                if ((time[M((j-1))] > crossing_c[M(k)]) &&
(flag_c == 0)){

                                        if (crossing_b[M(k)] >
crossing_c[M(k)]){

                                                step_1 = crossing_c[M(k)] -
time[M((j-2))];

                                                step_2 = 2*dt - step_1;

                                                switch (flag_down) {
```

```
                        case 1: gat_3[M(j)] = 0;
                                gat_5[M(j)] = 0;
                                break;
                        case 0: gat_3[M(j)] = 1;
                                gat_5[M(j)] = 1;
                                break;
                }
        }
        if (crossing_b[M(k)] < crossing_c[M(k)]){

                step_1 = crossing_b[M(k)] - time[M((j-2))];

                step_2 = 2*dt - step_1;

                switch (flag_down) {
                        case 1: gat_3[M(j)] = 0;
                                gat_5[M(j)] = 0;
                                break;
                        case 0: gat_3[M(j)] = 1;
                                gat_5[M(j)] = 1;
                                break;
                } }
        flag_b = 1; flag_c = 1; flag_bc = 1; flag_interp = 1;

} else{
        step_1 = crossing_b[M(k)] - time[M((j-2))];

        step_2 = 2*dt - step_1;

        switch (flag_down) {
                case 1: gat_3[M(j)] = 0;
                        break;
                case 0: gat_3[M(j)] = 1;
                        break;
                }
    flag_b = 1; flag_interp = 1;
}
                        } else{
                                gat_3[M(j)] = gat_3[M((j-1))];
                        }
                } else{
                        gat_3[M(j)] = gat_3[M((j-1))];
                }
        }
/*---------------- C-group Timing ---------------*/

        if ((flag_bc == 0) && (flag_ac == 0)){

                if (flag_c == 0){

                        if (time[M((j-1))] > crossing_c[M(k)]){

                                step_1 = crossing_c[M(k)] - time[M((j-2))];
                                step_2 = 2*dt - step_1;

                                        switch (flag_down) {
                                                case 1: gat_5[M(j)] = 0;
                                                        break;
                                                case 0: gat_5[M(j)] = 1;
                                                        break;
                                                } flag_c = 1; flag_interp = 1;
                        } else{
                                gat_5[M(j)] = gat_5[M((j-1))];
                        }
                } else{
                        gat_5[M(j)] = gat_5[M((j-1))];
                }
        }

/*------------- Interpolation Procedure ---------------*/

        if (flag_interp == 1){

/*------------ VSI Model for Time (j) ---------------*/

                if (gat_1[M(j)] == 1){
                        VaN[M(j)] = Vdc[M((j-1))];
                                S1[M(j)] = 1;
                } else{
                        VaN[M(j)] = 0.0;
                                S1[M(j)] = 0;
                }
                if (gat_3[M(j)] == 1){
```

```
                VbN[M(j)] = Vdc[M((j-1))];
                        S3[M(j)] = 1;
        } else{
                VbN[M(j)] = 0.0;
                        S3[M(j)] = 0;
                }
        if (gat_5[M(j)] == 1){
                VcN[M(j)] = Vdc[M((j-1))];
                        S5[M(j)] = 1;
        } else{
                VcN[M(j)] = 0.0;
                        S5[M(j)] = 0;
                }

        Iai = Ia[M((j-2))] + (step_1/dt) * ( Ia[M((j-1))] - Ia[M((j-2))] );
        Ibi = Ib[M((j-2))] + (step_1/dt) * ( Ib[M((j-1))] - Ib[M((j-2))] );
        Ici = Ic[M((j-2))] + (step_1/dt) * ( Ic[M((j-1))] - Ic[M((j-2))] );

        Vai = (2.0/3.0) * VaN[M(j)] - (1.0/3.0) * ( VbN[M(j)] +
VcN[M(j)] );
        Vbi = (2.0/3.0) * VbN[M(j)] - (1.0/3.0) * ( VaN[M(j)] +
VcN[M(j)] );
        Vci = (2.0/3.0) * VcN[M(j)] - (1.0/3.0) * ( VaN[M(j)] +
VbN[M(j)] );

        Eai = Ea[M((j-2))] + (step_1/dt) * ( Ea[M((j-1))] - Ea[M((j-2))]
);
        Ebi = Eb[M((j-2))] + (step_1/dt) * ( Eb[M((j-1))] - Eb[M((j-2))]
);
        Eci = Ec[M((j-2))] + (step_1/dt) * ( Ec[M((j-1))] - Ec[M((j-2))] );

        VDi = -(step_1/(2.0*C)) * Idc[M((j-2))] + Vdc[M((j-2))];
        Idci = S1[M(j)] * Iai + S3[M(j)] * Ibi + S5[M(j)] * Ici;
        Vdci = -(step_1/(2.0*C)) * Idci + VDi;

/*-------------------------------------------------------*/

        A = (2.0-(R/L)*step_2)/(2.0+(R/L)*step_2);
        B = (step_2/L)/(2.0+(R/L)*step_2);
/*-------------------------------------------------------*/

/*------------- Solution of System Equations of Time (j) ---------*/


    I1[M(j)] = A * Iai+ B * (Vai - Eai);
    I2[M(j)] = A * Ibi + B * (Vbi - Ebi);
    I3[M(j)] = A * Ici + B * (Vci - Eci);
```

```
        VD[M(j)] = -(step_2/(2.0*C)) * Idci + Vdci;

        Ea[M(j)] = Emag * cos(w*i);
        Eb[M(j)] = Emag * cos(w*i-(120.0*PI/180.0));
        Ec[M(j)] = Emag * cos(w*i+(120.0*PI/180.0));

                Va[M(j)] = (2.0/3.0) * VaN[M(j)] - (1.0/3.0) * ( VbN[M(j)] +
VcN[M(j)] );
                Vb[M(j)] = (2.0/3.0) * VbN[M(j)] - (1.0/3.0) * ( VaN[M(j)] +
VcN[M(j)] );
                Vc[M(j)] = (2.0/3.0) * VcN[M(j)] - (1.0/3.0) * ( VaN[M(j)] +
VbN[M(j)] );

        Ia[M(j)] = I1[M(j)] + (Va[M(j)]-Ea[M(j)]) * B;
        Ib[M(j)] = I2[M(j)] + (Vb[M(j)]-Eb[M(j)]) * B;
        Ic[M(j)] = I3[M(j)] + (Vc[M(j)]-Ec[M(j)]) * B;

                Idc[M(j)] = S1[M(j)] * Ia[M(j)] + S3[M(j)] * Ib[M(j)] + S5[M(j)]
* Ic[M(j)];

                Vdc[M(j)] = -(step_2/(2.0*C)) * Idc[M(j)] + VD[M(j)];

                Vab[M(j)] = Va[M(j)] - Vb[M(j)];
                Eab[M(j)] = Ea[M(j)] - Eb[M(j)];
                Ebc[M(j)] = Eb[M(j)] - Ec[M(j)];

                flag_ab = 0; flag_bc = 0; flag_ac = 0; flag_interp = 0;

        } else{

/*------------- Normal Procedure --------------------*/
\
/*----------- VSI Model for Time (j) ---------------*/

                if (gat_1[M(j)] == 1){
                        VaN[M(j)] = Vdc[M((j-1))];
                                S1[M(j)] = 1;
                } else{
                        VaN[M(j)] = 0.0;
                                S1[M(j)] = 0;
                        }
                if (gat_3[M(j)] == 1){
                        VbN[M(j)] = Vdc[M((j-1))];
                                S3[M(j)] = 1;
                } else{
                        VbN[M(j)] = 0.0;
                                S3[M(j)] = 0;
                }
                if (gat_5[M(j)] == 1){
                        VcN[M(j)] = Vdc[M((j-1))];
                                S5[M(j)] = 1;
                } else{
                        VcN[M(j)] = 0.0;
                                S5[M(j)] = 0;
                }

/*------------------------------*/
                A = A_o;
                B = B_o;
/*------------------------------*/

/*------------ Solution of System Equations of Time (j) ---------*/

        I1[M(j)] = A * Ia[M((j-1))] + B * (Va[M((j-1))] - Ea[M((j-1))]);
        I2[M(j)] = A * Ib[M((j-1))] + B * (Vb[M((j-1))] - Eb[M((j-1))]);
        I3[M(j)] = A * Ic[M((j-1))] + B * (Vc[M((j-1))] - Ec[M((j-1))]);
                VD[M(j)] = -(dt/(2.0*C)) * Idc[M((j-1))] + Vdc[M((j-1))];

        Ea[M(j)] = Emag * cos(w*i);
        Eb[M(j)] = Emag * cos(w*i-(120.0*PI/180.0));
        Ec[M(j)] = Emag * cos(w*i+(120.0*PI/180.0));

                Va[M(j)] = (2.0/3.0) * VaN[M(j)] - (1.0/3.0) * ( VbN[M(j)] +
VcN[M(j)] );
                Vb[M(j)] = (2.0/3.0) * VbN[M(j)] - (1.0/3.0) * ( VaN[M(j)] +
VcN[M(j)] );
                Vc[M(j)] = (2.0/3.0) * VcN[M(j)] - (1.0/3.0) * ( VaN[M(j)] +
VbN[M(j)] );

        Ia[M(j)] = I1[M(j)] + (Va[M(j)]-Ea[M(j)]) * B;
        Ib[M(j)] = I2[M(j)] + (Vb[M(j)]-Eb[M(j)]) * B;
        Ic[M(j)] = I3[M(j)] + (Vc[M(j)]-Ec[M(j)]) * B;

                Idc[M(j)] = S1[M(j)] * Ia[M(j)] + S3[M(j)] * Ib[M(j)] + S5[M(j)]
* Ic[M(j)];

                Vdc[M(j)] = -(dt/(2.0*C)) * Idc[M(j)] + VD[M(j)];

                Vab[M(j)] = Va[M(j)] - Vb[M(j)];
                Eab[M(j)] = Ea[M(j)] - Eb[M(j)];
                Ebc[M(j)] = Eb[M(j)] - Ec[M(j)];

        }
                time[M(j)] = i;
```

```
                /* For DC side transient */

                /* if (time[M(j)] >= 0.5){ flag_pi = 1; }

                   if ((time[M(j)] >= 3.0) && (time[M(j)] <= 4.0)){
                   flag_dc_load = 1;
                   } else{
                   flag_dc_load = 0;
                   } */

                /* For Iq Transient */
                /* if (time[M(j)] >= 0.5){flag_pi = 1;}
                if ((time[M(j)] >= 2.0) && (time[M(j)] <= 3.0)){
                  Iq_ref1 = 10.0;
                }else{
                  Iq_ref1 = Iq_ref;}*/

/*----------- Control Algorithm -------------------*/

                t1 = t1 + dt;

                if (t1 > Ts){

                        k++;

                 Eabs[M(k)] = Eab[M((j-1))];
                 Ebcs[M(k)] = Ebc[M((j-1))];
                 Ias[M(k)] = Ia[M((j-1))];
                 Ibs[M(k)] = Ib[M((j-1))];
                 Vcaps[M(k)] = Vdc[M((j-1))];

                Vs_re[M(k)] = (1.0/1.732) * ( 0.87 * Eabs[M((k-1))] + (
Ebcs[M((k-1))]*1.15 + Eabs[M((k-1))]*0.58 ) * 0.5);
                Vs_im[M(k)] = (1.0/1.732) * ( -0.5 * Eabs[M((k-1))] + (
Ebcs[M((k-1))]*1.15 + Eabs[M((k-1))]*0.58 ) * 0.87);
                Vs_abs[M(k)] = sqrt(Vs_re[M(k)]*Vs_re[M(k)] +
Vs_im[M(k)]*Vs_im[M(k)]);

                I_re[M(k)] = Ias[M((k-1))];
                I_im[M(k)] = 0.58 * Ias[M((k-1))] + 1.15 * Ibs[M((k-1))];

                        if (Vs_abs[M(k)] == 0.0){

                                coswt[M(k)] = 0.0;
                                sinwt[M(k)] = 0.0;
                        } else{
                                coswt[M(k)] = Vs_re[M(k)]/Vs_abs[M(k)];
```

```
                    sinwt[M(k)] = Vs_im[M(k)]/Vs_abs[M(k)];
                }
        Id[M(k)] = I_re[M(k)] * coswt[M(k)] + I_im[M(k)] * sinwt[M(k)];
        Iq[M(k)] = -I_re[M(k)] * sinwt[M(k)] + I_im[M(k)] * coswt[M(k)];


        Iqf[M(k)] = (2.2*Iqf0[M(k)] + Iq[M(k)] + Iq0[M(k)])/4.2;
        Iqf0[M(k)] = Iqf[M(k)];
        Iq0[M(k)] = Iq[M(k)];

                Iq_err[M(k)] = Iq_ref1 - Iq[M(k)];
                Vd_err[M(k)] = Vd_ref - Vcaps[M((k-1))];

            if (flag_pi == 1){

                        out_vd[M(k)] = out_vd[M((k-1))] + BETA
* Vd_err[M(k)] + BETA0 * Vd_err[M((k-1))];
                        Idrefd[M(k)] = out_vd[M(k)];
            } else{
                        out_vd[M(k)] = out_vd[M((k-1))];
                        Idrefd[M(k)] = Idrefd[M((k-1))];
            }
                Id_err[M(k)] = Idrefd[M(k)] - Id[M(k)];

            if (flag_pi == 1){

                        out_d[M(k)] = out_d[M((k-1))] +
ALPHA_d * Id_err[M(k)] + ALPHA0_d * Id_err[M((k-1))];
                        out_q[M(k)] = out_q[M((k-1))] +
ALPHA_q * Iq_err[M(k)] + ALPHA0_q * Iq_err[M((k-1))];
                        Ed[M(k)] = out_d[M(k)] - Iq[M(k)] * w *
L + Vs_abs[M(k)];

                        Eq[M(k)] = out_q[M(k)] + Id[M(k)] * w *
L;

                        Edq[M(k)] = sqrt(Ed[M(k)] * Ed[M(k)] +
Eq[M(k)] * Eq[M(k)]);


                        ma_x[M(k)] = Edq[M(k)] / (0.7071 *
Vcaps[M(k)]);

                        del_dc[M(k)] = atan(Eq[M(k)]/Ed[M(k)]) ;
            } else{
                        out_d[M(k)] = out_d[M((k-1))];
                        out_q[M(k)] = out_q[M((k-1))];
                        Ed[M(k)] = Ed[M((k-1))];
                        Eq[M(k)] = Eq[M((k-1))];
                        Edq[M(k)] = Edq[M((k-1))];
                        ma_x[M(k)] = 0.8;
```

```
                            del_dc[M(k)] = 10.0;
                        }
                ma[M(k)] = ma_x[M(k)];
                dc_ang[M(k)] = del_dc[M(k)] * PI/180.0;
                /* For open loop del_dc has to be multiplied by
PI/180.
                    For closed loop del_dc is already in radians due to
the
                    atan function */

/*--------------- Generation of Control Signals ---------------*/

            Vex_re[M(k)] = Vs_re[M(k)] * cos(dc_ang[M(k)]) - Vs_im[M(k)] *
sin(dc_ang[M(k)]);
            Vex_im[M(k)] = Vs_re[M(k)] * sin(dc_ang[M(k)]) + Vs_im[M(k)]
* cos(dc_ang[M(k)]);

            Va_exc[M(k)] = Vex_re[M(k)];
            Vb_exc[M(k)] = -0.5 * Vex_re[M(k)] + 0.87 * Vex_im[M(k)];
            Vc_exc[M(k)] = - Va_exc[M(k)] - Vb_exc[M(k)];

                if (Vs_abs[M(k)] == 0.0){

                    Vax_DV[M(k)] = 0.0;
                    Vbx_DV[M(k)] = 0.0;
                    Vcx_DV[M(k)] = 0.0;
                } else{

         /* Vax_DV[M(k)] = ma[M(k)] * Va_exc[M(k)] * 0.5 * Vcaps[M(k)] /
Vs_abs[M(k)];
            Vbx_DV[M(k)] = ma[M(k)] * Vb_exc[M(k)] * 0.5 * Vcaps[M(k)]
/ Vs_abs[M(k)];
            Vcx_DV[M(k)] = ma[M(k)] * Vc_exc[M(k)] * 0.5 * Vcaps[M(k)]
/ Vs_abs[M(k)]; */

            Vax_DV[M(k)] = ma[M(k)] * Va_exc[M(k)] / Vs_abs[M(k)];
            Vbx_DV[M(k)] = ma[M(k)] * Vb_exc[M(k)] / Vs_abs[M(k)];
            Vcx_DV[M(k)] = ma[M(k)] * Vc_exc[M(k)] / Vs_abs[M(k)];
                    }

                    control_a[M(k)] = Vax_DV[M(k)];
                    control_b[M(k)] = Vbx_DV[M(k)];
                    control_c[M(k)] = Vcx_DV[M(k)];

/*------------- Calculation of Gate Timing --------------*/

                    if (flag_down == 1){
```

```
                    Xa[M(k)] = (-Ts/(2.0*Vtri)) * ( control_a[M(k)]
- Vtri);
                    Xb[M(k)] = (-Ts/(2.0*Vtri)) * (
control_b[M(k)] - Vtri);
                    Xc[M(k)] = (-Ts/(2.0*Vtri)) * ( control_c[M(k)]
- Vtri);


                    crossing_a[M(k)] = time[M((j-1))] + Xa[M(k)];
                    crossing_b[M(k)] = time[M((j-1))] + Xb[M(k)];
                    crossing_c[M(k)] = time[M((j-1))] + Xc[M(k)];

                    flag_down = 0;
               } else{

                    Xa[M(k)] = (Ts/(2.0*Vtri)) * ( control_a[M(k)]
+ Vtri);
                    Xb[M(k)] = (Ts/(2.0*Vtri)) * ( control_b[M(k)]
+ Vtri);
                    Xc[M(k)] = (Ts/(2.0*Vtri)) * ( control_c[M(k)]
+ Vtri);


                    crossing_a[M(k)] = time[M((j-1))] + Xa[M(k)];
                    crossing_b[M(k)] = time[M((j-1))] + Xb[M(k)];
                    crossing_c[M(k)] = time[M((j-1))] + Xc[M(k)];

                    flag_down = 1;
                    }

               grap_a[M(j)] = control_a[M(k)];
               grap_b[M(j)] = control_b[M(k)];
               grap_c[M(j)] = control_c[M(k)];

               Iq1[M(j)] = Iq[M(k)];
               Id1[M(j)] = Id[M(k)];
               Idrefd1[M(j)] = Idrefd[M(k)];
               Vcaps1[M(j)] = Vcaps[M(k)];

               t1 = t1 - Ts;

               time_con[M(k)] = time[M((j-1))];

               flag_a = 0;
               flag_b = 0;
               flag_c = 0;
               flag_control[M(j)] = 1;
                              flag_ab = 0;
                              flag_bc = 0;
                              flag_ac = 0;

                    fprintf(fp2,"%f %f %f %f %f\n",time_con[M(k)],Eabs[M(k)],
                    Ebcs[M(k)],Id[M(k)],Iq[M(k)]);

          /* fprintf(fp2,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
               %f\n",time_con[M(k)],Eabs[M(k)],Ebcs[M(k)],Ias[M(k)],I_re[M(k)],
               I_im[M(k)],Id[M(k)],Id_err[M(k)],Iq[M(k)],Iq_err[M(k)],out_d[M(k)],
               out_q[M(k)],Ed[M(k)],Eq[M(k)],ma[M(k)],dc_ang[M(k)],Vd_err[M(k)],
               Vcaps[M(k)],Iqf[M(k)]); */

                    } else{
                         grap_a[M(j)] = grap_a[M((j-1))];
                         grap_b[M(j)] = grap_b[M((j-1))];
                         grap_c[M(j)] = grap_c[M((j-1))];

                          flag_control[M(j)] = 0;
                          Iq1[M(j)] = Iq1[M((j-1))];
                          Id1[M(j)] = Id1[M((j-1))];
                          Idrefd1[M(j)] = Idrefd1[M((j-1))];
                          Vcaps1[M(j)] = Vcaps1[M((j-1))];
                    }

               fprintf(fp1,"%f %f %f %f %f %f\n",time[M(j)],Va[M(j)],Ia[M(j)],
               Vab[M(j)],Vdc[M(j)],Idc[M(j)]);

          /* fprintf(fp1,"%f %f %f %f %f %f %f %f %f %f %f %f %f\n",time[M(j)],
               Va[M(j)],Ia[M(j)],Vab[M(j)],Vdc[M(j)],Idc[M(j)],grap_a[M(j)],Iq_ref1,
               Iq1[M(j)],Idrefd1[M(j)],Id1[M(j)],Vd_ref,Vcaps1[M(j)]); */

          i += dt;
          iter++;
          }

end_clock = clock();
printf("done.  \n");
printf("Iterations = %d\n",iter);

fclose(fp1);
fclose(fp2);
printf("Processor time used:  %g sec.\n",
          (end_clock - start_clock) / (double) CLOCKS_PER_SEC);
return 0;
}
```

```
/***********************************************************
RTS_110.C : PROGRAM FOR REAL-TIME SIMULATION OF A
PULSE WIDTH MODULATED (PWM) VOLTAGE SOURCE INVERTER (VSI)
SYSTEM PARAMETERS : 110V(L-LRMS), R=0.5OHM, L=3.0MH
SWITCHING FREQUENCY OF VSI = 1KHZ
CONTROLLER SAMPLING FREQUENCY = 2KHZ (TS = 500US)
SIMULATION AND CONTROL IS PERFORMED BY C40 DSP
PWM AND DISCRETE EVENT CAPTURE (DEC) IS PROGRAMMMED IN
THE ALTERA FPGA
DOWNLOAD DESIGN16N.SOF TO THE FPGA.
PROGRAM DEVELOPED AND TESTED BY VENKATA DINAVAHI
LAST UPDATED ON OCT. 20 '99
NOTE : THIS PROGRAM WAS USED TO GENERATE REAL-TIME
SIMULATION RESULTS REPORTED IN CHAPTER #4
***********************************************************/

#include <math.h>
#include <kernel.h>

/*==== SWITCH DECLARATIONS ======*/

#define on_sw (switches & 0x01)
#define pi_sw (switches & 0x02)
#define iq_sw (switches & 0x04)
/*#define il_sw (switches & 0x08) */

/*===== SIGNAL DECLARATIONS =====*/

#define Vdc signals[1]
#define Iq_av signals[2]
#define Iq_ref signals[3]
/*
#define Ea signals[1]
#define Eb signals[2]
#define Va signals[3]
#define Ia signals[4]
#define Idc signals[5]
#define Vdc signals[6]
#define Vab signals[7]

#define gat_1 signals[8]
#define gat_3 signals[10]
#define gat_5 signals[11]
#define step_1 signals[12]
#define count1_sig signals[13]
#define count2_sig signals[14]
#define flag_i signals[15]
#define con_a signals[16]
#define con_b signals[17]
#define con_c signals[18]

#define Id signals[10]
#define Iq signals[11]
#define Vs_re signals[13]
#define Vs_im signals[14]
#define ma_sig signals[12]
#define dc_ang_sig signals[13]

#define Iqf signals[14]
#define Idf signals[15]

/*#define Id_av signals[15] */

/*#define step_2 signals[19]*/
/*#define Vd_ref signals[16]
#define Iq_ref signals[4]
/*#define Icap signals[17]
#define Iload signals[5]
/*#define I_im signals[24]*/

/*#define control_a signals[10]
#define control_b signals[11]
#define control_c signals[12] */

/*======= PARAMETER DECLARATIONS ========*/

/*#define ma_par parameters[1]
#define dc_ang_par parameters[2]
#define Vd_ref parameters[3]
#define Ti parameters[4]
#define Kid parameters[5]
#define Kiq parameters[6]
#define Kv parameters[7]
#define Tv parameters[8] */

/*#define Iq_ref parameters[4]
#define Iload_par parameters[3]
#define beta_par parameters[4]
#define Kp_d_par parameters[5]*/

/*===== FPGA INFORMATION ======*/

#define Ts 5
```

```
#define FPGA_frequency ( long) 20 /* FPGA Clock frequency in MHz */
#define PWM_deadtime ( long) 2 /* deadtime in usec */
#define PWM_resolution ( long) 16 /* refers to 16-bit counter in 16pwm.gdf */
#define PWM_bs ( long) (32-PWM_resolution) /* bitshift of 32-bit word */
#define upper_limit ( long) ((long)(FPGA_frequency * (Ts*sampling_period)/2)
≪ PWM_bs)
/* The factor 5 in the upper_limit is Ts = 5 */

/*===== GLOBAL ADDRESS LINES ===== */

#define count_1_add ( long *) 0x83000001 /* ga0 read only*/
#define pulse_add ( long *) 0x83000002 /* ga1 read only*/
#define count_2_add ( long *) 0x83000004 /* ga2 read only*/
#define upperlimit_add ( long *) 0x83000008 /* ga3 write only*/
#define dvaluea_add ( long *) 0x83000010 /* ga4 write only*/
#define dvalueb_add ( long *) 0x83000020 /* ga5 write only*/
#define dvaluec_add ( long *) 0x83000040 /* ga6 write only*/
#define wd_res_add ( long *) 0x83000080 /* ga7 write only*/

#define PI (float) 3.141592654
#define FFT_SIZE ( int ) 133 /* FFT_SIZE = 16.67ms / sampling_period in us */

#define DELTA 2.0*PI/3.0
#define THETA 2.0*PI/FFT_SIZE

#define w 2.0*PI*60.0
#define D2R PI/180.0
#define R2D 180.0/PI

#define R 0.5
#define L 3.e-3
#define C 4900.e-6
#define Rdc 390.0
#define Rload 80.0

#define Emag 110.0 * sqrt(2.0)/sqrt(3.0) /* This is peak value of l-n voltage */

float sine_a[FFT_SIZE];
float sine_b[FFT_SIZE];
float sine_c[FFT_SIZE];

float k1,k2,k3,k4,k5,k7,A_o,B_o,dt;/*A_cap_o,B_cap_o*/
float scale;
int pulse, count_1, count_2,flag_interp=0;
float Kp, Kp_d, ALPHA_d, ALPHA_q, ALPHA0_d, ALPHA0_q, BETA, BETA0;

float Id_hist[33], Iq_hist[33];
```

```
int flag_av = 0, j = 0;

float ma_par, dc_ang_par, Vd_ref,Ti, Kid, Kiq, Kv, Tv;

void intfunc1(void);

void intfunc1(void)

{

        static int i = 0, t1 = 0;
        static int sw1=0, sw2=0, sw3=0;

        static int S1 = 0, S3 = 0, S5 = 0;

        static float A = 0.0, B = 0.0;
        static float A_cap = 0.0, B_cap = 0.0;

        static float step_1_dt = 0.0;

        static float VaN = 0.0, VbN = 0.0, VcN = 0.0;

        static float I1 = 0.0, I2 = 0.0, I3 = 0.0, VD = 0.0;

        static float Ec = 0.0, Vb = 0.0, Vc = 0.0, Ib = 0.0, Ic = 0.0;

        static float /*I_im = 0.0,*/ coswt = 0.0, sinwt = 0.0, cos_dc = 0.0,
sin_dc = 0.0;

        static float Vex_re = 0.0, Vex_im = 0.0;
        static float Va_exc = 0.0, Vb_exc = 0.0, Vc_exc = 0.0;
        static float ma = 0.0, dc_ang = 0.0;

        static float Iq_err = 0.0, Id_err = 0.0, Vd_err = 0.0, out_d = 0.0, out_q =
0.0;
        static float out_vd = 0.0, out_vd0 = 0.0, Idrefd = 0.0, out_d0 = 0.0,
out_q0 = 0.0;
        static float Ed = 0.0, Eq = 0.0, Edq = 0.0;

        static float Vai = 0.0, Vbi = 0.0, Vci = 0.0, Iai = 0.0, Ibi = 0.0, Ici = 0.0;
        static float Eai = 0.0, Ebi = 0.0, Eci = 0.0, Vdci = 0.0, Idci = 0.0, VDi
= 0.0;

        static float Va_j2 = 0.0, Va_j1 = 0.0, Vb_j2 = 0.0, Vb_j1 = 0.0, Vc_j2 =
0.0, Vc_j1 = 0.0;
        static float Ea_j2 = 0.0, Ea_j1 = 0.0, Eb_j2 = 0.0, Eb_j1 = 0.0, Ec_j2 =
0.0, Ec_j1 = 0.0;
```

```
        static float Ia_j2 = 0.0, Ia_j1 = 0.0, Ib_j2 = 0.0, Ib_j1 = 0.0, Ic_j2 = 0.0,
Ic_j1 = 0.0;
        static float Idc_j2 = 0.0, Idc_j1 = 0.0, Vdc_j2 = 0.0, Vdc_j1 = 0.0;


        static float Id_err_j1 = 0.0, Iq_err_j1 = 0.0, Vd_err_j1 = 0.0, Vs_abs = 0.0;
        static float Iqf0 = 0.0, Iq0 = 0.0, Idf0 = 0.0, Id0 = 0.0;
/* static float Ias0 = 0.0, Ias_f0 = 0.0, Ibs0 = 0.0, Ibs_f0 = 0.0; */

/* static float Vs_re = 0.0, Vs_im = 0.0, Vs_abs = 0.0; */
        static float Eabcs = 0.0, base = 0.0, Ias = 0.0, Ibs = 0.0;
        static float Ebcs = 0.0, Vcaps = 0.0;

        static float gat_3 = 0.0, gat_5 = 0.0, step_1 = 0.0, count1_sig = 0.0,
count2_sig = 0.0;
        static float con_b = 0.0, con_c = 0.0, flag_i = 0.0;
        static float Eabs = 0.0, Vs_re = 0.0, Vs_im = 0.0, I_im = 0.0, step_2 =
0.0;

        static float Ea = 0.0,Eb = 0.0, Va = 0.0,Ia = 0.0, Idc = 0.0;
        static float Vab = 0.0, gat_1 = 0.0, con_a = 0.0, Id = 0.0, Iq = 0.0;
        static float Id_av = 0.0, ma_sig = 0.8, dc_ang_sig = 0.0;
      /* static float Iq_av = 0.0; */

        /*========== PROGRAM LOOP =========*/

        if (on_sw)
        {

/*==== Acquisition of Gates and their timing from FPGA ====*/

        pulse = (*pulse_add) >> PWM_bs; /* bitshift for the pulses and counter
values */
        count_1 = (*count_1_add) >> PWM_bs;
        count_2 = (*count_2_add) >> PWM_bs;

        count1_sig = count_1;
        count2_sig = count_2;

        flag_interp = pulse & 0x0001; /* extracting flag and switch status */
        sw1 = (pulse & 0x0002) >> 1;
        sw2 = (pulse & 0x0004) >> 2;
        sw3 = (pulse & 0x0008) >> 3 ;

        gat_1 = sw1;
        gat_3 = sw2;
        gat_5 = sw3;
```

```
        flag_i = flag_interp;

        step_1 = (float)count_1/(FPGA_frequency*1.e6); /* step_1 is 50ns times
the counter value */
        step_2 = 2.0*dt - step_1;

/*if (iq_sw){Iload = 2.5;}else{Iload = 0.0;} */

        /*======= SYSTEM SIMULATION ========*/


         if (flag_interp == 1){

/*==== Interpolation Procedure ====*/

/*==== VSI Model for Time(j) =====*/

                if (gat_1 == 1.){
                        VaN = Vdc;
                        S1 = 1;
                } else{
                        VaN = 0.0;
                        S1 = 0;
                }
                if (gat_3 == 1.){
                        VbN = Vdc;
                        S3 = 1;
                } else{
                        VbN = 0.0;
                        S3 = 0;
                }
                if (gat_5 == 1.){
                        VcN = Vdc;
                        S5 = 1;
                } else{
                        VcN = 0.0;
                        S5 = 0;
                }

                step_1_dt = step_1/dt;

                Iai = Ia_j2 + step_1_dt * ( Ia_j1 - Ia_j2 );
                Ibi = Ib_j2 + step_1_dt * ( Ib_j1 - Ib_j2 );
                Ici = Ic_j2 + step_1_dt * ( Ic_j1 - Ic_j2 );

                Vai = k1 * VaN - k2 * ( VbN + VcN );
                Vbi = k1 * VbN - k2 * ( VaN + VcN );
```

```
            Vci = k1 * VcN - k2 * ( VaN + VbN );

            Eai = Ea_j2 + step_1_dt * ( Ea_j1 - Ea_j2 );
            Ebi = Eb_j2 + step_1_dt * ( Eb_j1 - Eb_j2 );
            Eci = Ec_j2 + step_1_dt * ( Ec_j1 - Ec_j2 );

            Idci = S1 * Iai + S3 * Ibi + S5 * Ici;

    /* if (iq_sw){
        A_cap = (1.0-(step_1/(2.0*Rload*C)))/(1.0+(step_1/(2.0*Rload*C)));
        B_cap = (step_1/(2.0*C))/(1.0+(step_1/(2.0*Rload*C)));
        VDi = -B_cap * Idc_j2 + A_cap * Vdc_j2;
        Vdci = -B_cap * Idci + VDi;
        }else{ */
            VDi = -(step_1*k3) * Idc_j2 + Vdc_j2;
            Vdci = -(step_1*k3) * Idci + VDi;

/*-------------------------------------------------------*/
            A = (2.0-k5*step_2)/(2.0+k5*step_2);
            B = (step_2/L)/(2.0+k5*step_2);
    /* A_cap = (1.0-(step_2/(2.0*Rload*C)))/(1.0+(step_2/(2.0*Rload*C)));
        B_cap = (step_2/(2.0*C))/(1.0+(step_2/(2.0*Rload*C))); */
/*-------------------------------------------------------*/

/*====== Solution of System Equations for Time(j) ======*/

        I1 = A * Iai + B * (Vai - Eai);
        I2 = A * Ibi + B * (Vbi - Ebi);
        I3 = A * Ici + B * (Vci - Eci);

    /* if (iq_sw){
        VD = -B_cap * Idc + A_cap * Vdc;
        }else{ */
        VD = -(step_2*k3) * Idc + Vdc;

        Ea = sine_a[i];
        Eb = sine_b[i];
        Ec = sine_c[i];

            Va = k1 * VaN - k2 * ( VbN + VcN );
            Vb = k1 * VbN - k2 * ( VaN + VcN );
            Vc = k1 * VcN - k2 * ( VaN + VbN );

        Ia = I1 + (Va-Ea) * B;
        Ib = I2 + (Vb-Eb) * B;
        Ic = I3 + (Vc-Ec) * B;
```

```
            Idc = S1 * Ia + S3 * Ib + S5 * Ic;

                /* if (iq_sw){
                    Vdc = -B_cap * Idc + VD;
                    Iload = Vdc/Rload;
                    }else{ */
                    Vdc = -(step_2*k3) * Idc + VD;
                /* Iload = 0.0;
                    } */

            Vab = Va - Vb;

            flag_interp = 0;

        } else{

/*============= Normal Procedure ==================*/

/*============= VSI Model for Time(j) ================*/

            if (gat_1 == 1.){
                    VaN = Vdc;
                    S1 = 1;
            } else{
                    VaN = 0.0;
                    S1 = 0;
                    }
            if (gat_3 == 1.){
                    VbN = Vdc;
                    S3 = 1;
            } else{
                    VbN = 0.0;
                    S3 = 0;
                    }
            if (gat_5 == 1.){
                    VcN = Vdc;
                    S5 = 1;
            } else{
                    VcN = 0.0;
                    S5 = 0;
                    }
/*----------------------------*/
            A = A_o;
            B = B_o;
        /* A_cap = A_cap_o;
            B_cap = B_cap_o; */
/*----------------------------*/
```

```
/*==== Solution of System Equations for Time(j) ====*/

        I1 = A * Ia + B * (Va - Ea);
        I2 = A * Ib + B * (Vb - Eb);
        I3 = A * Ic + B * (Vc - Ec);

    /* if (iq_sw){
        VD = -B_cap * Idc + A_cap * Vdc;
        }else{ */
        VD = -(dt*k3) * Idc + Vdc;

        Ea = sine_a[i];
        Eb = sine_b[i];
        Ec = sine_c[i];

                Va = k1 * VaN - k2 * ( VbN + VcN );
                Vb = k1 * VbN - k2 * ( VaN + VcN );
                Vc = k1 * VcN - k2 * ( VaN + VbN );

        Ia = I1 + (Va-Ea) * B;
        Ib = I2 + (Vb-Eb) * B;
        Ic = I3 + (Vc-Ec) * B;

                Idc = S1 * Ia + S3 * Ib + S5 * Ic;

         /* if (iq_sw){
                Vdc = -B_cap * Idc + VD;
                Iload = Vdc/Rload;
                }else{ */
                Vdc = -(dt*k3) * Idc + VD;
         /* Iload = 0.0;
              } */
                    Vab = Va - Vb;
        }

        /*======CONTROL ALGORITHM ========*/

                t1 = t1 + 1;

    if (t1 == Ts){

                 Eabs = Ea - Eb;
                 Ebcs = Eb - Ec;
                 Ias = Ia;
                 Ibs = Ib;
                 Vcaps = Vdc;
```

```
/* Ias_f = (2.2 * Ias_f0 + Ias + Ias0)/4.2;
        Ias_f0 = Ias_f;
        Ias0 = Ias;

        Ibs_f = (2.2 * Ibs_f0 + Ibs + Ibs0)/4.2;
        Ibs_f0 = Ibs_f;
        Ibs0 = Ibs; */


                Eabcs = Ebcs*1.15 + Eabs*0.58;
                Vs_re = k4 * ( 0.87 * Eabs + Eabcs * 0.5);
                Vs_im = k4 * ( -0.5 * Eabs + Eabcs * 0.87);
                Vs_abs = sqrt(Vs_re*Vs_re + Vs_im*Vs_im);

                I_im = 0.58 * Ias + 1.15 * Ibs;

                        coswt = Vs_re/Vs_abs;
                        sinwt = Vs_im/Vs_abs;

                Id = Ias * coswt + I_im * sinwt;
                Iq = -Ias * sinwt + I_im * coswt;


/* Iqf = (2.2 * Iqf0 + Iq + Iq0)/4.2;
        Iqf0 = Iqf;
        Iq0 = Iq;

        Idf = (2.2 * Idf0 + Id + Id0)/4.2;
        Idf0 = Idf;
        Id0 = Id; */

                  if (flag_av==0){ Id_av=Id_av+(float)(Id/33) ; /* 33 =
16.67ms/Ts/dt */
                        Iq_av=Iq_av+(float)(Iq/33) ;
                  }else{ Id_av = Id_av+((float)(Id-Id_hist[j])/33);
                        Iq_av = Iq_av+((float)(Iq-Iq_hist[j])/33);
                     }

                  Id_hist[j] = Id;
                  Iq_hist[j] = Iq;

                If(iq_sw){Iq_ref =5.0;}else{Iq_ref = -5.0;}

                        Iq_err = Iq_ref - Iq;
                        Vd_err = Vd_ref - Vcaps;
                        if (pi_sw){

        /* BETA = Kv * (Ts*dt/(2.0*Tv) + 1.0);
         BETA0 = BETA - 2.0*Kv; */
                                        out_vd = out_vd0 + BETA * Vd_err +
BETA0 * Vd_err_j1;
                                        Idrefd = out_vd;

                                        Id_err = Idrefd - Id;

   /* ALPHA_d = Kid *( Ts*dt/(2.0*Ti) + 1.0);
         ALPHA0_d = ALPHA_d - 2.0*Kid;

         ALPHA_q = Kiq * ( Ts*dt/(2.0*Ti) + 1.0);
         ALPHA0_q = ALPHA_q - 2.0*Kiq; */

                                        out_d = out_d0 + ALPHA_d *Id_err +
ALPHA0_d * Id_err_j1;
                                        out_q = out_q0 + ALPHA_q * Iq_err +
ALPHA0_q * Iq_err_j1;

                                        Ed = out_d - Iq *k7 + Vs_abs;
                                        Eq = out_q + Id * k7;
                                        Edq = sqrt(Ed *Ed + Eq * Eq);

                                        ma = Edq /0.7071/Vcaps;
                                        dc_ang = atan2(Eq,Ed) ;

                                        ma_sig = ma; /* output only */
                                        dc_ang_sig = dc_ang *R2D; /* output
only */
                        } else{
                                        ma_sig = ma_par;
                                        dc_ang_sig = dc_ang_par;
                                        dc_ang = dc_ang_par *D2R;

                            }

                            if (ma_sig > 1.0){
                                        ma_sig = 1.0; }
                         /* }else{ma_sig = ma_sig;} */

                            if (ma_sig < 0.0){
                                        ma_sig = 0.0; }
                        /* }else{ma_sig = ma_sig;} */

        /*===== GENERATION OF CONTROL SIGNALS ======*/
```

```
                    cos_dc = cos(dc_ang);
                    sin_dc = sin(dc_ang);

                    Vex_re = Vs_re * cos_dc - Vs_im * sin_dc;
                    Vex_im = Vs_re * sin_dc + Vs_im * cos_dc;

                    Va_exc = Vex_re;
                    Vb_exc = -0.5 * Vex_re + 0.87 * Vex_im;
                    Vc_exc = - Va_exc - Vb_exc;

                    base = ma_sig/ Vs_abs;

                    con_a = base *Va_exc;
                    con_b = base * Vb_exc;
                    con_c = base * Vc_exc;

/* control_a = ma * sin(THETA*i);
      control_b = ma * sin(THETA*i - DELTA);
      control_c = ma * sin(THETA*i + DELTA); */

      /*===== WRITE CONTROL VALUES TO FPGA =======*/

            set_out_port(FPGA_IN);

 /* if((fpga_sw)
                    {set_out_port(FPGA_IN);
                   set_out_port(LED3); */

                    *upperlimit_add = upper_limit; /* writes upperlimit to FPGA */
                    scale = (float)(upper_limit >> PWM_bs);

                    *dvaluea_add = ((long) (scale*con_a)) << PWM_bs;
                    *dvalueb_add = ((long) (scale*con_b)) << PWM_bs;
                    *dvaluec_add = ((long) (scale*con_c)) << PWM_bs;

/* }else
                    {res_out_port(FPGA_IN);} */

                              Vd_err_j1 = Vd_err;
                              Id_err_j1 = Id_err;
                              Iq_err_j1 = Iq_err;
                              out_d0 = out_d;
                              out_q0 = out_q;
                              out_vd0 = out_vd;

                              t1 = t1 - Ts;
```

```
                j++;
                if(j==33) {j=0; flag_av = 1;}
            }

/*===== Data buffer of length 2 ======*/


        Va_j2 = Va_j1; Va_j1 = Va;
        Vb_j2 = Vb_j1; Vb_j1 = Vb;
        Vc_j2 = Vc_j1; Vc_j1 = Vc;

        Ea_j2 = Ea_j1; Ea_j1 = Ea;
        Eb_j2 = Eb_j1; Eb_j1 = Eb;
        Ec_j2 = Ec_j1; Ec_j1 = Ec;

        Ia_j2 = Ia_j1; Ia_j1 = Ia;
        Ib_j2 = Ib_j1; Ib_j1 = Ib;
        Ic_j2 = Ic_j1; Ic_j1 = Ic;

        Idc_j2 = Idc_j1; Idc_j1 = Idc;
        Vdc_j2 = Vdc_j1; Vdc_j1 = Vdc;

        i++;
        if (i==FFT_SIZE) {i=0;}

        }else{ /* else for on-switch */
                        res_out_port(FPGA_IN);
                        *wd_res_add = 0x00;

        i = 0; t1 = 0;
        sw1=0; sw2=0; sw3=0;
        S1 = 0; S3 = 0; S5 = 0;
        A = 0.0; B = 0.0;
        step_1 = 0.0; step_2 = 0.0; step_1_dt = 0.0;
        VaN = 0.0; VbN = 0.0; VcN = 0.0;
        I1 = 0.0; I2 = 0.0; I3 = 0.0; VD = 0.0;
        Ec = 0.0; Vb = 0.0; Vc = 0.0; Ib = 0.0; Ic = 0.0;
        I_im = 0.0; coswt = 0.0; sinwt = 0.0; cos_dc = 0.0; sin_dc = 0.0;
        Vex_re = 0.0; Vex_im = 0.0;
        Va_exc = 0.0; Vb_exc = 0.0; Vc_exc = 0.0;
        ma = 0.0; dc_ang = 0.0;
        Iq_err = 0.0; Id_err = 0.0; Vd_err = 0.0; out_d = 0.0; out_q = 0.0;
        out_vd = 0.0; out_vd0 = 0.0; Idrefd = 0.0; out_d0 = 0.0; out_q0 = 0.0;
        Ed = 0.0; Eq = 0.0; Edq = 0.0;
        Vai = 0.0; Vbi = 0.0; Vci = 0.0; Iai = 0.0; Ibi = 0.0; Ici = 0.0;
        Eai = 0.0; Ebi = 0.0; Eci = 0.0; Vdci = 0.0; Idci = 0.0; VDi = 0.0;
```

```
        Va_j2 = 0.0; Va_j1 = 0.0; Vb_j2 = 0.0; Vb_j1 = 0.0; Vc_j2 = 0.0; Vc_j1 =
0.0;
        Ea_j2 = 0.0; Ea_j1 = 0.0; Eb_j2 = 0.0; Eb_j1 = 0.0; Ec_j2 = 0.0; Ec_j1 =
0.0;
        Ia_j2 = 0.0; Ia_j1 = 0.0; Ib_j2 = 0.0; Ib_j1 = 0.0; Ic_j2 = 0.0; Ic_j1 = 0.0;
        Idc_j2 = 0.0; Idc_j1 = 0.0; Vdc_j2 = 0.0; Vdc_j1 = 0.0;
        Id_err_j1 = 0.0; Iq_err_j1 = 0.0; Vd_err_j1 = 0.0; Vs_abs = 0.0;
        Iqf0 = 0.0; Iq0 = 0.0; Idf0 = 0.0; Id0 = 0.0;
        Vs_re = 0.0; Vs_im = 0.0; Vs_abs = 0.0;
        Eabcs = 0.0; base = 0.0;
        Ebcs = 0.0; Ias = 0.0; Ibs = 0.0; Vcaps = 0.0;
/* Ias0 = 0.0; Ias_f0 = 0.0; Ibs0 = 0.0; Ibs_f0 = 0.0;
Ias_f = 0.0; Ibs_f = 0.0; */
        Ea = 0.0; Eb = 0.0;
        Va = 0.0; Ia = 0.0; Idc = 0.0; Vdc = 0.0;
        Eabs = 0.0; Vab = 0.0;
        gat_1 = 0.0; gat_3 = 0.0; gat_5 = 0.0;
        step_1 = 0.0; step_2 = 0.0; count1_sig = 0.0; count2_sig = 0.0;
        flag_i = 0.0;
        con_a = 0.0; con_b = 0.0; con_c = 0.0;
        Id = 0.0; Iq = 0.0; /*Iqf = 0.0; Idf = 0.0; */
        Vs_re = 0.0; Vs_im = 0.0;
        ma_sig = ma_par; dc_ang_sig = dc_ang_par;
/* control_a = 0.0; control_b = 0.0; control_c = 0.0;
        Iload = 0.0; Icap = 0.0; */
        Iq_av = 0.0; Id_av = 0.0;flag_av = 0;j = 0;
                } /* closing brace for on_switch */

} /* closing brace for void intfunc1 */

void user_init (void)
{
        float theta,delta;
        int ii;
        theta=2.0*PI/FFT_SIZE;
        delta=2.0*PI/3.0;

        for(ii=0;ii<FFT_SIZE;ii++)

        {
                sine_a[ii]= Emag * sin(theta*ii);
                sine_b[ii]= Emag * sin(theta*ii-delta);
                sine_c[ii]= Emag * sin(theta*ii+delta);
        }

        sampling_period = 100.0;
        dt = sampling_period*1.e-6;
```

```
/*====== PWM initialization =========*/

            *count_1_add = 0x00;
            *pulse_add = 0x00;
            *count_2_add = 0x00;
            *upperlimit_add = 0x00;
            *dvaluea_add = 0x00;
            *dvalueb_add = 0x00;
            *dvaluec_add = 0x00;
            *wd_res_add = 0x00;

/*====== Parameter initializations ======*/

            ma_par = 0.8; dc_ang_par = 10.0;
/* beta_par = 0.001; Kp_d_par = -1.5;
            Iload_par = 2.5;*/

/*====== Signal initializations =======*/

      /* Ea = 0.0; Eb = 0.0;
         Va = 0.0; Ia = 0.0; Idc = 0.0;
         Eabs = 0.0; Vab = 0.0;
         control_a = 0.0; control_b = 0.0; control_c = 0.0;
         gat_1 = 0.0; gat_3 = 0.0; gat_5 = 0.0;
         step_1 = 0.0; step_2 = 0.0; count1_sig = 0.0; count2_sig = 0.0;
         flag_i = 0.0;
         con_a = 0.0; con_b = 0.0; con_c = 0.0;
         Id = 0.0; Iq = 0.0; Iqf = 0.0; Idf = 0.0;I_im = 0.0;
         Vs_re = 0.0; Vs_im = 0.0; Vs_abs = 0.0; Va_exc = 0.0;
         ma_sig = ma_par; dc_ang_sig = dc_ang_par;
         Ias = 0.0; Ias_f = 0.0; Ibs = 0.0; Ibs_f = 0.0;
         ma = 0.8; ALPHA = 0.0; */
         Vd_ref = 240.0; Iq_ref = -5.0;Vdc = 160.0;Iq_av = 0.0;
      /* Iload = 0.0; Icap = 0.0;
         Id_av = 0.0; */

/*======= Definitions of constants =======*/

         A_o = (2.0-(R/L)*dt)/(2.0+(R/L)*dt);
         B_o = (dt/L)/(2.0+(R/L)*dt);

/* A_cap_o = (1.0-(dt/(2.0*Rload*C)))/(1.0+(dt/(2.0*Rload*C)));
         B_cap_o = (dt/(2.0*C))/(1.0+(dt/(2.0*Rload*C)));
         Kp = 1.12;
         ALPHA = 374.0*Ts*dt/2.0; */
```

```
            Kid = 0.8;
            Kiq = 1.5;
            Ti = 50.e-3; /* Ti = 3ms */

/* ALPHA = Ki * ( Ts*dt/(2.0*Ti) + 1.0);
            ALPHA0 = ALPHA - 2.0*Ki; */

            ALPHA_d = Kid * ( Ts*dt/(2.0*Ti) + 1.0);
            ALPHA0_d = ALPHA_d - 2.0*Kid;

            ALPHA_q = Kiq * ( Ts*dt/(2.0*Ti) + 1.0);
            ALPHA0_q = ALPHA_q - 2.0*Kiq;

            Kv = -2.0;
            Tv = 0.5;
            BETA = Kv * (Ts*dt/(2.0*Tv) + 1.0);
            BETA0 = BETA - 2.0*Kv;

/* Kp = 1.2;
            AlPHA = 374.0*Ts*dt/2.0;
            Kp_d = -1.5;
            BETA = 0.001*Ts*dt/2.0; */

            k1 = 2.0/3.0;
            k2 = 1.0/3.0;
            k3 = 1.0/(2.0*C);
            k4 = 1.0/sqrt(3.0);
            k5 = R/L;
            k7 = w * L;
            user_interrupt = intfunc1;
}
```

```
/*************************************************************
CON_110.C : PROGRAM FOR OPEN-LOOP AND CLOSED-LOOP CONTROL OF A
PULSE WIDTH MODULATED (PWM) VOLTAGE SOURCE INVERTER (VSI)
SYSTEM PARAMETERS : 110V(L-LRMS), R=0.5OHM, L=3.0MH
SWITCHING FREQUENCY OF VSI = 1KHZ
CONTROLLER SAMPLING FREQUENCY = 2KHZ (TS = 500US)
A/D DATA ACQUISITION OF 6 SIGNALS :
VAB,VBC,IA,IB,VDC,IDC
DOWNLOAD CON_FPGA.SOF TO THE FPGA.
PROGRAM DEVELOPED AND TESTED BY VENKATA DINAVAHI
LAST UPDATED ON AUG 26 '99
NOTE : THIS PROGRAM WAS USED TO GENERATE EXPERIMENTAL
RESULTS REPORTED IN CHAPTER #4
*************************************************************/

#include <math.h>
#include <kernel.h>

/*===== SWITCH DECLARATIONS =====*/

#define on_sw (switches & 0x01)
#define fpga_sw (switches & 0x02)
#define pi_sw (switches & 0x04)
#define dc_pi (switches & 0x08)
#define iq_pi (switches & 0x10)
#define wd_res (switches & 0x20)
#define iq_sw (switches & 0x40)

/*===== SIGNAL DECLARATIONS =====*/

#define control_a signals[1]
#define control_b signals[2]
#define control_c signals[3]
#define Vsab signals[4]
#define Vsbc signals[5]
#define Vsabc signals[6]
#define Vs_re signals[7]
#define Vs_im signals[8]
#define Vs_abs signals[9]
#define Van signals[10]
#define Ia signals[11]
#define Ib signals[12]
#define I_re signals[13]
#define I_im signals[14]
#define Id signals[15]
#define Iq signals[16]
#define Vdc signals[17]
#define Idc signals[18]
#define Vex_re signals[19]
#define Vex_im signals[20]
#define Va_exc signals[21]
#define Vb_exc signals[22]
#define Vc_exc signals[23]
#define con_a signals[24]
#define con_b signals[25]
#define con_c signals[26]
#define coswt signals[27]
#define sinwt signals[28]
#define Vsp signals[29]
#define ma_sig signals[30]
#define dc_ang_sig signals[31]
#define Ia_f signals[32]
#define Ib_f signals[33]
#define Vdc_f signals[34]
#define ma_final signals[35]
#define dc_ang_final signals[36]
#define Vd_err signals[37]
#define Id_err signals[38]
#define Iq_err signals[39]
#define Ed signals[40]
#define Eq signals[41]
#define out_d signals[42]
#define out_q signals[43]
#define out_vd signals[44]
#define Id_av signals[45]
#define Iq_av signals[46]
#define step signals[47]
#define Idc_f signals[48]

/*===== PARAMETER DECLARATIONS =====*/

#define ma_par parameters[1]
#define dc_ang_par parameters[2]
#define Iq_ref_par parameters[3]
#define Vd_ref_par parameters[4]
#define Kv parameters[5]
#define Kid parameters[6]
#define Kiq parameters[7]
#define Tid parameters[8]
#define Tiq parameters[9]
#define Tv parameters[10]
#define Ki1f parameters[11]
#define Ki2f parameters[12]
#define Kv1f parameters[13]
```

```
#define Kv2f parameters[14]
#define out_d_par parameters[15]
#define out_q_par parameters[16]

/*===== FPGA INFOMATION ====== */

#define FPGA_frequency ( long) 20 /* FPGA Clock frequency in MHz */
#define PWM_deadtime ( long) 2 /* deadtime in usec */
#define PWM_resolution ( long) 16 /* refers to 16-bit counter in 16pwm.gdf */
#define PWM_bs ( long) (32-PWM_resolution) /* bitshift of 32-bit word */
#define upper_limit ( long) ((long)(FPGA_frequency * (sampling_period -
2*PWM_deadtime)/2-2) ≪ PWM_bs)

/*===== GLOBAL ADDRESS LINES =====*/

#define upperlimit_add ( long *) 0x83000001 /* ga0 write only*/
#define dvaluea_add ( long *) 0x83000002 /* ga1 write only*/
#define dvalueb_add ( long *) 0x83000004 /* ga2 write only*/
#define dvaluec_add ( long *) 0x83000008 /* ga3 write only*/
#define wd_res_add ( long *) 0x83000010 /* ga4 write only*/

/*===== GLOBAL CONSTANTS =====*/

#define PI (float) 3.141592654
#define FFT_SIZE ( int ) 33 /* FFT_SIZE = 16.67ms / sampling_period in us */
#define PHI 2.0*PI/3.0
#define DELTA 2.0*PI/3.0
#define THETA 2.0*PI/FFT_SIZE
#define w 2.0*PI*60.0
#define L 3.0e-3
#define R 0.5

/*====== GLOBAL VARIABLES =====*/

float sine_a[FFT_SIZE];
float sine_b[FFT_SIZE];
float sine_c[FFT_SIZE];
float sine[FFT_SIZE];
float cosine[FFT_SIZE];
float scale, Kvac, Kvdc, Kiac, Kidc;
float Kp, Kp_d, k7, ALPHA_d, ALPHA_q, BETA;
/*float K1f, K2f;
float Kv1f, Kv2f; */
int i, res_flag;
float ma, dc_ang;
float cos_dc, sin_dc;
float base;
float Iq_ref, Vd_ref, Idrefd;
float Edq;
float Iq_err0, Id_err0, Vd_err0, ma0, dc_ang0;
float Iq_ref0; Vd_ref0;
float Id_err_i, Id_err_i0;
float Ia0, Ia_f0, Ib0, Ib_f0, Vdc0, Vdc_f0, Idc0, Idc_f0;
float I_re_f, I_im_f;
int flag;
float out_d0, out_q0, out_vd0;
float Id_hist[FFT_SIZE], Iq_hist[FFT_SIZE];

/*===== A/D DATA ACQUISITION VARIABLES =====*/

float INT_MAX_FLOAT = 2147483646.0;
float ad_1, ad_2, ad_3, ad_4, ad_5, ad_6, ad_7, ad_8;
signed long data_1, data_2, data_3, data_4;
signed long ad_1temp, ad_2temp, ad_3temp, ad_4temp;
signed long ad_5temp, ad_6temp, ad_7temp, ad_8temp;

void intfunc1(void);
{
/*==== A/D DATA ACQUISITION =======*/

        data_1 = in_word(4);
        data_2 = in_word(4);
        data_3 = in_word(1);
        data_4 = in_word(1);

        ad_1temp = ((data_1 & 0x0000ffff) ≪ 16) & 0xffff0000;
        ad_2temp = ((data_1 & 0xffff0000));
        ad_3temp = ((data_2 & 0x0000ffff) ≪ 16) & 0xffff0000;
        ad_4temp = ((data_2 & 0xffff0000));

        ad_5temp = ((data_3 & 0x0000ffff) ≪ 16) & 0xffff0000;
        ad_6temp = ((data_3 & 0xffff0000));
        ad_7temp = ((data_4 & 0x0000ffff) ≪ 16) & 0xffff0000;
        ad_8temp = ((data_4 & 0xffff0000));

        ad_1 = (float) ad_1temp;
        ad_2 = (float) ad_2temp;
        ad_3 = (float) ad_3temp;
        ad_4 = (float) ad_4temp;
        ad_5 = (float) ad_5temp;
        ad_6 = (float) ad_6temp;
        ad_7 = (float) ad_7temp;
        ad_8 = (float) ad_8temp;
        Vsab = Kvac * 5.0 * ad_1/(INT_MAX_FLOAT);
        Vsbc = Kvac * 5.0 * ad_2/(INT_MAX_FLOAT);
        Vdc = Kvdc * 5.0 * ad_3/(INT_MAX_FLOAT);
        Van = Kvac * 5.0 * ad_4/(INT_MAX_FLOAT);
        Ia = Kiac * 5.0 * ad_8/(INT_MAX_FLOAT);
        Ib = Kiac * 5.0 * ad_7/(INT_MAX_FLOAT);
        Idc = Kidc * 5.0 * ad_6/(INT_MAX_FLOAT);

if(on_sw){

res_out_port(LED3);

        Ia_f = (Ki1f *Ia_f0 + Ia + Ia0)/Ki2f;
        Ia_f0 = Ia_f;
        Ia0 = Ia;

        Ib_f = (Ki1f *Ib_f0 + Ib + Ib0)/Ki2f;
        Ib_f0 = Ib_f;
        Ib0 = Ib;

        Vdc_f = (Kv1f *Vdc_f0 + Vdc + Vdc0)/Kv2f;
        Vdc_f0 = Vdc_f;
        Vdc0 = Vdc;

        Idc_f = (Kv1f *Idc_f0 + Idc + Idc0)/Kv2f;
        Idc_f0 = Idc_f;
        Idc0 = Idc;

/*==== Phase voltage is used as reference vector ====*/

        Vsabc = Vsbc*1.15 + Vsab*0.58;
        Vs_re = (1.0/sqrt(3.0)) * ( 0.87 * Vsab + Vsabc * 0.5);
        Vs_im = (1.0/sqrt(3.0)) * ( -0.5 * Vsab + Vsabc * 0.87);
        Vs_abs = sqrt(Vs_re*Vs_re + Vs_im*Vs_im);

/*==== Calculation of Id and Iq ====*/

        I_re_f = Ia_f;
        I_im_f = 0.58 * Ia_f + 1.15 * Ib_f;

/* I_re = I_re_f * 0.8944 - I_im_f * 0.4472; /* phase lead of 26.56 deg.
        I_im = I_re_f * 0.4472 + I_im_f * 0.8944; for fh = 120Hz */

        I_re = I_re_f * 0.9578 - I_im_f * 0.2873; /* phase lead of 16.7 deg. */
        I_im = I_re_f * 0.2873 + I_im_f * 0.9578; /* for fh = 200Hz */

  /* I_re = Ia;
        I_im = 0.58 * Ia + 1.15 * Ib; */

        coswt = Vs_re/Vs_abs;
        sinwt = Vs_im/Vs_abs;
        Vsp = Vs_re + Vs_im;

        Id = I_re * coswt + I_im * sinwt; /* This assumes that current space
phasor */
        Iq = -I_re * sinwt + I_im * coswt; /* leads the voltage space phasor */

         if (flag==0){ Id_av=Id_av+(float)(Id/FFT_SIZE) ;
                Iq_av=Iq_av+(float)(Iq/FFT_SIZE) ;
         }else{ Id_av = Id_av+((float)(Id-Id_hist[i])/FFT_SIZE);
                Iq_av = Iq_av+((float)(Iq-Iq_hist[i])/FFT_SIZE);
            }
                Id_hist[i] = Id;
                Iq_hist[i] = Iq;

/*===== CONTROL ALGORITHM ======*/


                        if (pi_sw){

                            if(iq_sw){Iq_ref = Iq_ref_par;step = 2.0;}
                                    else{Iq_ref = Iq_ref0;step = 0.0;}

                            /* if (Iq_ref_par < -20.0 || Iq_ref_par > 20.0)
                                    {Iq_ref = Iq_ref0;}
                                else{Iq_ref = Iq_ref_par;} */

                            /* if (Vd_ref_par < 90.0 || Vd_ref_par > 180.0)
                                    {Vd_ref = Vd_ref0;}*/
                                Vd_ref = Vd_ref_par;

                                Iq_err = Iq_ref - Iq;
                                Vd_err = Vd_ref - Vdc_f;

                            if (dc_pi){


                /*===== Vdc controller ======*/

                            /* out_vd = Kv * Vd_err; */

                                BETA = Kv *
(sampling_period*1.e-6/(2.0*Tv) + 1.0);
                                out_vd = out_vd0 + BETA * Vd_err +
(BETA - 2.0*Kv) * Vd_err0;
                                        if (out_vd > 20.0){out_vd = 20.0;}
                                        if (out_vd < -20.0){out_vd = -20.0;}


                                        out_vd0 = out_vd;

        /*====== Id controller =====*/

                                        Idrefd = out_vd;
                                        Id_err = Idrefd - Id;
                                        Id_err_i = Kid * Id_err;
                                        /* out_d = Kid * Id_err; */
                                        ALPHA_d = Kid *
(sampling_period*1.e-6/(2.0*Tid) + 1.0;
                                        out_d = out_d0 + ALPHA_d * Id_err +
(ALPHA_d - 2.0*Kid) * Id_err0; if (out_d > 70.0){out_d = 70.0;}
                                        if (out_d < -70.0){out_d = -70.0;} out_d0
= out_d;
                                /* if (iq_pi){ */

        /*===== Iq controller ======*/
                                    /* out_q = Kiq * Iq_err; */
                                        ALPHA_q = Kiq *
(sampling_period*1.e-6/(2.0*Tiq) + 1.0); out_q = out_q0 + ALPHA_q * Iq_err +
(ALPHA_q - 2.0*Kiq) * Iq_err0;
                                        if (out_q > 50.0){out_q = 50.0;}
                                        if (out_q < -50.0){out_q = -50.0;}
                                        out_q0 = out_q;
                                  /* }else{ out_q = 12.0;out_q0 = 0.0; } */
                                        }else{out_vd = 0.0; out_d = 40.0;out_q =
30.0; out_q0 = 0.0;
                                         out_vd0 = 0.0; out_d0 = 0.0;}

        /*======== Calculation of Mod.Index (ma) and Delta =====*/

                                        Ed = out_d - Iq * k7 + Vs_abs;
                                        Eq = out_q + Id * k7;
                                        Edq = sqrt(Ed * Ed + Eq * Eq); ma =
Edq / (0.7071 * Vdc_f);
                                        ma_sig = ma;
                                        dc_ang = atan2(Eq,Ed) ; /* dc_ang is now
in radians */
                                        dc_ang_sig = dc_ang * 180.0/PI;

        /*===== History values for integration =======*/
```

```
                                        Vd_err0 = Vd_err;
                                        Iq_err0 = Iq_err;
                                        Id_err0 = Id_err;
                                        Id_err_i0 = Id_err_i;
                                        Iq_ref0 = Iq_ref;
                                        Vd_ref0 = Vd_ref;

                                /* out_d0 = out_d;out_q0 = out_q; out_vd0 =
out_vd; */

                        } else{
                                        out_vd = 0.0; out_d = 40.0; out_q = 30.0;
                                        out_d0 = 0.0; out_q0 = 0.0; out_vd0 = 0.0;
Vd_err0 = 0.0;Iq_err0 = 0.0;Id_err0 = 0.0;Id_err_i0 = 0.0;
                                        Iq_ref0 = Iq_ref;Vd_ref0 = Vd_ref; ma =
ma_par;
                                        ma_sig = ma; /*--- only for output --- */
                                        dc_ang = dc_ang_par*PI/180.0;
                                        dc_ang_sig = dc_ang * 180.0/PI;/*--- only
for output --- */
                        }

        /*==== Limits on ma and dc_ang =====*/

                if (ma < 0.4 || ma > 0.98)
                        {ma = ma0;}
                /* if (ma < 0.4){ma = 0.4;}
                   if (ma > 0.98){ma = 0.98;} */
                if (dc_ang*180.0/PI < -70.0 || dc_ang*180.0/PI > 70.0)
                        {dc_ang = dc_ang0;}

                ma_final = ma; /*--- only for output --- */
                dc_ang_final = dc_ang*180.0/PI;/*--- only for output --- */
cos_dc = cos(dc_ang); /*-- inputs to cos and sin functions */
                sin_dc = sin(dc_ang); /*-- has to be in radians always --*/

        /*====== Generation of controls ======*/

                Vex_re = Vs_re * cos_dc - Vs_im * sin_dc;
                Vex_im = Vs_re * sin_dc + Vs_im * cos_dc;
                Va_exc = Vex_re;
                Vb_exc = -0.5 * Vex_re + 0.87 * Vex_im;
                Vc_exc = - Va_exc - Vb_exc;
                base = ma / Vs_abs;
                con_a = base * Va_exc;
                con_b = base * Vb_exc;
```

```
            con_c = base * Vc_exc;
            ma0 = ma;
            dc_ang0 = dc_ang;

/* control_a = ma * sine_a[i]; /* modulation index = 0.8
             control_b = ma * sine_b[i];
             control_c = ma * sine_c[i]; */

      /*======== Inputs to FPGA ===========*/

            *upperlimit_add = upper_limit; /* writes upperlimit to FPGA */
            scale = (float)(upper_limit >> PWM_bs);

            *dvaluea_add = ((long) (scale*con_a)) << PWM_bs;
            *dvalueb_add = ((long) (scale*con_b)) << PWM_bs;
            *dvaluec_add = ((long) (scale*con_c)) << PWM_bs;

      if(fpga_sw)
            {set_out_port(FPGA_IN);
           set_out_port(LED3);}
      else
            {res_out_port(FPGA_IN);}

      if (wd_res && (res_flag == 0))
            {*wd_res_add = 0x00; res_flag = 1;}
      else
            {res_flag = 0;}


      i++;
      if (i==FFT_SIZE) {i=0;flag=1;}

    } /*— closing brace for on_sw —*/

} /*— closing brace for intfunc1 —*/


void user_init(void)
{
      float theta,delta;
      int ii;
      theta=2.0*PI/FFT_SIZE;
      delta=2.0*PI/3.0;
      for(ii=0;ii<FFT_SIZE;ii++)

      {
            sine_a[ii]=10.0 * sin(theta*ii);
            sine_b[ii]=10.0 * sin(theta*ii-delta);
            sine_c[ii]=10.0 * sin(theta*ii+delta);
            sine[ii] = (float) sin(ii*theta);
            cosine[ii] = (float) cos(ii*theta);
      }

      sampling_period = 500.0;

/*===== PWM initialization ==== */

        *upperlimit_add = 0x00;
        *dvaluea_add = 0x00;
        *dvalueb_add = 0x00;
        *dvaluec_add = 0x00;
        *wd_res_add = 0x00;

/*===== Signal Initializations =====*/

      control_a = 0.0; control_b = 0.0; control_c = 0.0;
      Vsab = 0.0; Vsbc = 0.0;
      Vsabc = 0.0; Vs_re = 0.0; Vs_im = 0.0; Vs_abs = 0.0;
      Van = 0.0; Vdc = 0.0;
      Ia = 0.0; Ib = 0.0; Idc = 0.0;
      I_re = 0.0; I_im = 0.0; Id = 0.0; Iq = 0.0;
      Vex_re = 0.0; Vex_im = 0.0;
      Va_exc = 0.0; Vb_exc = 0.0; Vc_exc = 0.0;
      con_a = 0.0; con_b = 0.0; con_c = 0.0;
      coswt = 0.0; sinwt = 0.0; Vsp = 0.0;
      Ia_f = 0.0; Ib_f = 0.0; Vdc_f = 0.0;
      ma_final = 0.0; dc_ang_final = 0.0;
      Iq_err = 0.0; Vd_err = 0.0; Id_err = 0.0;
      Ed = 0.0; Eq = 0.0;
      out_vd = 0.0; out_d = 40.0; out_q = 30.0;
      Id_av = 0.0; Iq_av = 0.0;
      step = 0.0; Idc_f = 0.0;

/*===== Parameter Initializations =====*/

      ma_par = 0.8; dc_ang_par = 14.0;
      Iq_ref_par = 0.0; Vd_ref_par = 220.0;
      Ki1f = 2.183; Ki2f = 4.183;
      /* Ki1f = (2*tau/Ts) - 1, Ki2f = (2*tau/Ts) + 1;
        cutoff freq = 200Hz tau = 7.96e-4s */
      Kv1f = 20.22; Kv2f = 22.22;
      /* cutoff freq = 30Hz tau = 5.3e-3s */
      out_d_par = 40.0; out_q_par = 30.0;
      /* out_d_par = 5.0/40.0, out_q_par = 20.0/30.0
```

```
          Vll = 57.5/110 */

/*===== Float Initializations ======*/
          i = 0; res_flag = 0;
          ma = ma_par;ma_sig = ma_par;
          dc_ang = dc_ang_par*PI/180.0; dc_ang_sig = dc_ang_par;
          Iq_err = 0.0; Vd_err = 0.0; Id_err = 0.0;
          Iq_err0 = 0.0; Id_err0 = 0.0; Vd_err0 = 0.0; Idrefd = 0.0;
          Edq = 0.0; Iq_ref = Iq_ref_par; Vd_ref = Vd_ref_par;
          ma0 = 0.0; dc_ang0 = 0.0;
          Iq_ref0 = Iq_ref; Vd_ref0 = Vd_ref;
          flag = 0;
          out_vd = 0.0; out_d0 = 0.0; out_q0 = 0.0;
          Id_err_i = 0.0; Id_err_i0 = 0.0;

/*===== Constants =====*/

          Kvac = 71.0; Kvdc = 71.0; Kiac = 10.0; Kidc = 10.0;

/* Kp = 1.12;
          ALPHA = 374.0*sampling_period*1.e-6/2.0;
          Kp_d = 1.5;
          BETA = 0.001*sampling_period*1.e-6/2.0;
          k7 = w * L; */
          Kid = 0.8;
          Kiq = -1.5;
          Tiq = 50.e-3; /* Ti = 3ms */
          Tid = 50.e-3;
          ALPHA_d = Kid * (sampling_period*1.e-6/(2.0*Tid) + 1.0);
          ALPHA_q = Kiq * (sampling_period*1.e-6/(2.0*Tiq) + 1.0);
          Kv = -2.0;
          Tv = 0.5;
          BETA = Kv * (sampling_period*1.e-6/(2.0*Tv) + 1.0);
          k7 = w * L;

          user_interrupt = intfuncl;
}
```