Real Time Digital Control System for Power System Analog Simulator

by

G Henry Yogendran

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirement for the degree of

Master of Science

in the Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

(c) August 2000

0-612-53246-1

Real Time Digital Control System for Power System Analog Simulator

BY

G Henry Yogendran

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree

of

Master of Science

ACKNOWLEDGEMENTS

First, the author wishes to convey innumerable gratitude to Professor Ani. Gole for his guidance, counsel and his encouragement toward the completion of this thesis and the Masters program at the University of Manitoba.

The author would also like to thank Manitoba HVDC Research Centre for providing the opportunity to work on such an interesting thesis project and for providing financial and technical support during the course of this work.

My sincere thanks goes to the friendly technical staff at the Power Tower and at the Electrical Engineering department, for their technical support and advice.

ABSTRACT

With increasing system complexity, engineers use different tools to save money and time during the design and development cycle. Simulation is one of the important tool engineers use very often. These simulation tools are becoming more powerful because of the faster computers available today.

This thesis presents a design of a Real Time Control System (RTCS) simulation concepts using famous Power System simulation tool PSCAD and the off the shelf Digital Signal Processing (DSP) board with the Analog simulator at the University of Manitoba.

This RTCS simulation concept is validated for known HVDC system networks in real time. The benefit of the design is to achieve simple design of a real time controls with available hardware and software such as PSCAD and DSP boards.

Table of Contents

List of Figures

List of Tables

CHAPTER ONE

INTRODUCTION

The term "simulation" is used rather broadly today to cover several concepts in the computing world. This chapter deals specifically with "real-time" simulation. The term "real-time", as it relates to simulation, requires that the computer program execution of a modelled dynamic process must occur in the real-world time. It represents or simulates a real, dynamic phenomenon as it occurs. A real-time simulation in electrical power system is usually characterized by a control panel, interface hardware, computers, and an observer, all of which are linked together in a closed loop system.

Simulation is the technique by which a physical system can be represented mathematically by a computer program for the solution of a problem. This technique of problem solving is used when it is not feasible due to time, cost, or safety to conduct specific tests using the actual physical system, such as a high voltage power system. A mathematical model is developed for the physical system using knowledge of the physical laws describing the problem. This model is then programmed on the computer to generate the problem solution. The digital computer program represents a discrete approximation of the real world system (which is usually continuous). Within the overall task of simulation, there are three primary sub-fields: model design, model execution and model analysis.

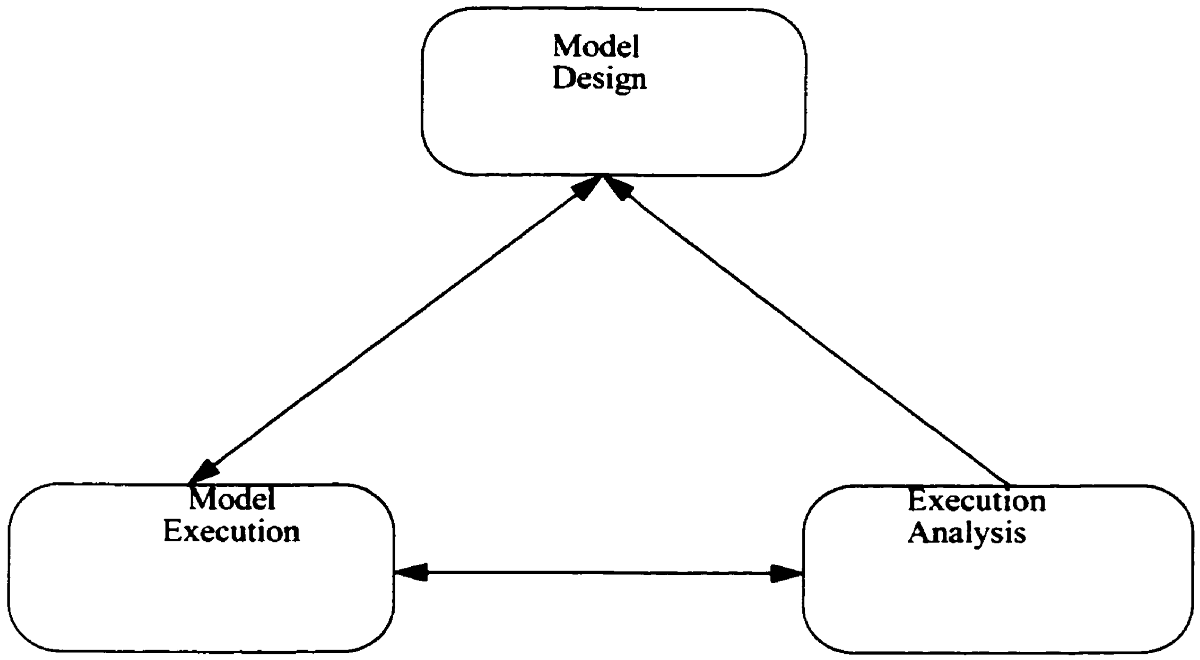


Figure 1.1

Models can take many forms including declarative, functional, constraint, spatial or multimodal. The next task, once a model has been developed, is to execute the model on a computer. We need to create a computer program which steps through time while updating the state and event variables in our mathematical model. There are many ways to "step through time" , for instance, leap through time using event scheduling or we can employ small time increments using time slicing. We can also execute/simulate the program on a massively parallel computer. This is called parallel and distributed simulation. For many large-scale models, this is the only feasible way of getting answers back in a reasonable amount of time.

When the control inputs to the system can be predetermined and are programmable, batch processing of the simulation program is possible. For batch processing, the computer program will be submitted to the computer and it runs as fast as the computer will allow. In this type of processing the running-time of the computer program (as it is running) is not related to the real world time.

In the event that the control inputs that are necessary for the testing procedure are dynamic in nature or cannot be predetermined, such as fault response in a power system, the term simulation takes on a new dimension known as real-time simulation. This new dimension calls for strict correspondence between the computer running-time (as it is running) and the real world time. Inputs and outputs to the hardware devices must be synchronized to a real-time clock and cannot be time-scaled as in the batch computing environment. A typical power system real time simulation involves a real-time computer program, an observer and appropriate interfaces that are all synchronized and running in real world time.

Simulation studies have historically been used as a tool in manufacturing operations to determine the effect of proposed changes before making commitments to capital expenditure or changes in operational policy.

1.1 Benefits of Simulation

There are many advantages of using simulation in lieu of other methods of research. For example, expensive prototypes need not be built and possibly destroyed during tests. A real systems with humans aboard need not be used for unknown experimental results thus jeopardizing life and expensive equipment. With simulation it is easy to extend the testing beyond the normal safety thresholds. Simulation permits rapid change from one set of conditions to another in a controlled environment, thus greatly extending the versatility of an experimental setup. Real-time simulation allows a more realistic representation of the physical system being studied and permits both quantitative and qualitative evaluation. In addition, users can be trained for various operational concepts without using expensive electrical systems, although most simulators are used for research and development purposes rather than for training.

In simulation, several alternative revisions may be proposed and there is uncertainty as to how each will behave operationally. System changes are invariably costly in terms of time, money and lost production while system revisions are incorporated. Thus it is frequently necessary to simulate the entire system in its original configuration and with each proposed alternative revision.

Engineers can create every conceivable scenario of contingencies and re-enact them to find solutions before they can actually occur in real life. Engineers can test equipment they plan on acquiring and make sure beforehand that it will meet their real needs.

Furthermore, Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behaviour of the system and evaluating various strategies for the operation of the system. Simulation has proven to be a cost-effective analysis tool which assists engineers and managers make decisions quicker and reliably. There are a lot of benefits of applying simulation technology for productivity improvement initiatives, some of these benefits includes: avoiding costly mistakes, estimating crucial parameters, experimenting on model rather than actual system.

1.2 Digital Simulation

Modern real-time digital simulators for power systems can have all the flexibility needed to carry out simulation of complex systems. Its graphical interface, coupled with the powerful latest generation of processors, means the user can design, modify and control the simulation of an electric power system in real time using a simple mouse, thus adding to the amazing possibilities digital simulation offers. There are new real-time digital simulators using the latest processor and programming language technology. They feature, graphical interface enables the user to graphically define a power system network with all its parameters. It also offers maximum flexibility, allowing the user to change the parameters or establish a new list of signals for observation, through the simple manipulation of the mouse, while a simulation is running.

1.3 Analog Simulation

Although the new digital technology is attractive in many ways, analog models still offer some important advantages in power systems, especially for environments where the user needs the, robustness and credibility on its simulation, including highly non-linear components such as arresters and thyristor valves. Also, for some users, it is very important to have a real feel of the simulated network through real voltage while other users already have an analog simulator and wish to expand their facilities with a similar technology. The hybrid technology represents a gradual evolution and is still the most reliable and accurate simulation method.

Currently, transient stability is investigated either by large discrete analog simulators that emulate the exact behaviour of the real network, or by numerical calculations that simulate the network behaviour using strong computers. The discrete analog simulators are actually a scale-down model of the real network, where the elements/components of the simulator are of the exact nature as the corresponding elements in the real network, for example generators, transformers, lines, cables, etc. Analog and digital simulators have their respective advantages and disadvantages. The main advantage of the analog simulators is their shorter computation time. On the other hand numerical simulators are easier to handle. Both of them however are very expensive and cumbersome.

This thesis will show simple implementation of a Real Time Control System using a digital signal processor, for a power system analog simulator.

CHAPTER TWO

SIMULATION CONCEPTS

Simulation is an important tool that allows the engineer to investigate system behaviour prior to real-world implementation. Simulation can be divided into two main categories: as off-line simulation and real-time simulation. In software based simulation, engineers can observe system behaviour with parameter changes. In real-time simulation, the system behaviour is achieved in real time. This method is important say, for testing real equipment. Real time simulation can be digitally based using very fast computers or it can be analog based where physical electronic components are used.

2.1 Analog and Digital Simulation

As discussed above design, analysis and planning studies of modern power systems are performed by using simulation techniques. These techniques fall into two main classes:

1. Off-Line computer programs.
2. Real-Time simulation

Real-time simulation for electrical networks can be divided into two groups such as analog simulation and real-time digital simulation. In analog simulation, the system can be scaled down to small system which consist of smaller scaled down identical components and scaled down parameters such as voltage, current etc. Setting up the scaled down analog simulation takes more time and effort because real hardware must be arranged for each case. On the other hand, real time digital simulation is faster to set up as it can be carried out using a conventional computer interface.

In real-time digital simulation, data are presented to the system from external environment, in synchronism with the external phenomena. In non-real time software based simulation, the software obtains the next input data only when it is ready to do so. Many real time systems rely upon interrupt service routines to notify the main processing software that input data are available. There is no notion of a "right time" and a "bad time" to service interrupts in a non real-time systems. However, in the real time system, the interrupt service routine can run at any time, and it must share data structures, like input buffers, with the main software.

Effective communications require support from both hardware, like integral I/O, and software driver routines. In real-time digital simulation large amounts of data must be shifted between data acquisition, data display, and processor boards. Furthermore, the architecture of digital signal processing systems must be sufficiently flexible to adapt to different configurations, which means the user can reuse both hardware and software, thus reducing development cost and time.

2.2 Digital Processors

Special purpose Digital Signal Processors (DSPs) are more powerful for simulation purposes when compared with conventional computer CPUs since they are tailor made to handle signal processing applications. Typically digital signal processors manipulate continuous data flow in real-time. They perform a single task with minimal latency and with limited memory and peripheral devices. Digital signal processors tend to be specialized devices whereas conventional processors tend to be generalized.

The requirements for digital signal processing are:

Real-time (deterministic) operation: The greatest challenge in DSP processing has been to develop systems that can handle all of these requirements with flexibility and precision in real time.

Computational power: In power systems, high speed computation is also required, because, although the sampling rate is lower than for other DSP applications, the algorithms are typically complex.

Fast I/O: The key requirement for I/O is that there should be wide I/O bandwidth. Wide I/O bandwidth mainly refers to the communication of the processors with the external world.

2.3 PSCAD/EMTDC:

EMTDC is a popular non-real time simulation engine for power system studies based on detailed mathematical models and numerical techniques for time domain simulation. This computer program works off line and the computation time of an event turns out to be orders of magnitude longer than the actual event time. The EMTDC engine is invoked from a graphical user interface called PSCAD. PSCAD allows the user to make schematic drawings of the power net-work using one of its module called DRAFT (Fig 2.1). Parameter entry is achieved using user-friendly pop-up menus. After schematic compilation, the Fortran simulation code is produced when the user clicks the COMPILE button on the DRAFT module. This Fortran code also facilitate some output file structure to view the simulation result in the plotting environment. For the simulation, the produced Fortran code is imported to another module called RUNTIME (Figure. 2.2).

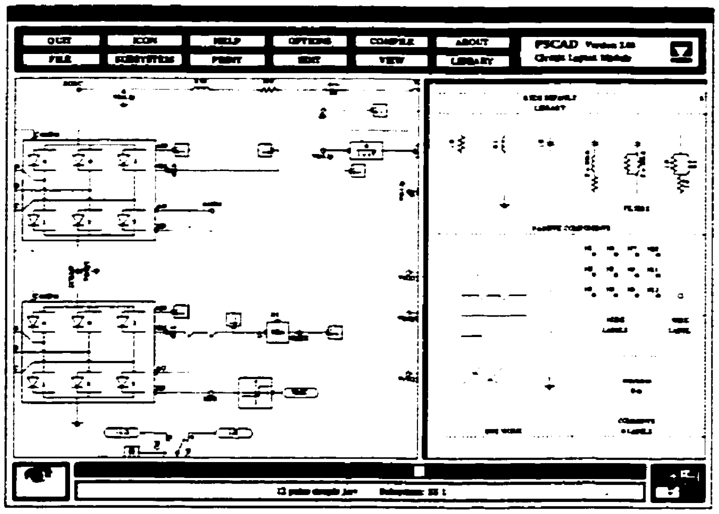


Figure 2.1

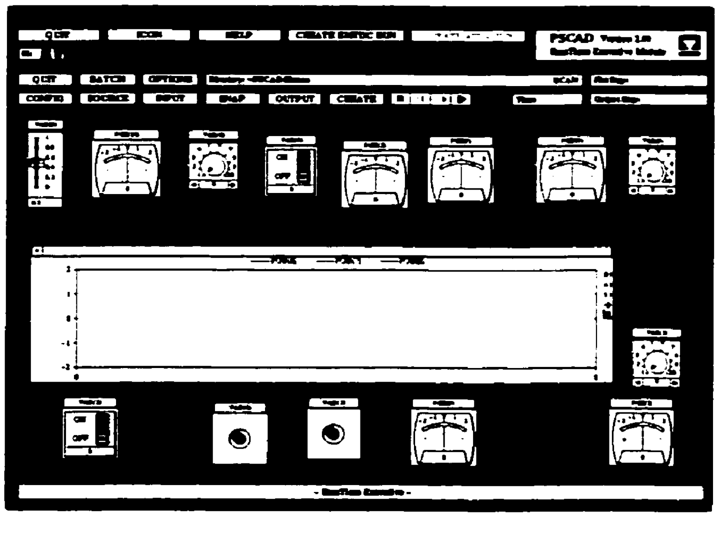


Figure 2.2

The Fortran code is further compiled into an executable program. The user controls the simulation and observes waveforms using the RUNTIME module. This allows the user to make changes to some system parameters during the run, in much the same manner as an operator at a system control console. After the simulation ends, output results can also be saved for later investigation.

Because of its non real-time nature, PSCAD/EMTDC is not the appropriate tool for testing real control or protection devices. This simulator is chiefly used to study the behaviour of different network and control topology.

2.4 Real Time Digital Simulators (RTDS):

The PSCAD/RTDS, originally designed at the Manitoba HVDC Research Centre, is a real time digital simulator to study the behaviour of power system devices in a real time environment.

At the hardware level, the multiprocessor control system is made up of various plug-in boards optionally arranged in a rack for HVDC and power system applications. Each processor board has it's own digital signal processor, data memory and output ports. The output ports interface the processor to external systems and signals with normalized output levels. The output signal has to be amplified to levels necessary for testing the real system component which is under test.

As for the software part, the programming of control or generation of signals, including all control parameter setting, is carried out in the PSCAD DRAFT module, which contains the graphical RTDS component library. The library consists of components such as control blocks, arithmetic blocks, logic and switch blocks and output ports. The schematic produced in the draft environment is then down-loaded in the RTDS memory as an executable digital signal processing program. During run-time, the output signals from RTDS can be fed to any real component under test through amplifiers and the system behaviour can be monitored using real instruments

2.5 Real Time Analog Simulators

Some investigators prefer to retain a direct correspondence of voltage, current and other derived variables between an actual Power system and its simulator. Analog simulators are favoured because, not only do they retain the system identity, the current and voltage variables of the real system are scaled down in the analog simulators.

Real time analog simulators are best described as hybrid simulators because these simulators generally contain a range of models such as, reduced scaled down physical models, equivalent circuit models including operational-amplifier based analog differential equation solvers. Modelling of power system on analog simulators is accomplished after connecting a complex arrangement of electrical components. The analog simulator at the University of Manitoba Power System Laboratory was developed to study HVDC systems behaviour under different conditions. It is a scaled down version of a large HVDC systems and consists of real components as well as variable power source and monitoring devices. This simulator can be used to study limited amount of HVDC topology using scaled down parameters.

Implementing different HVDC control algorithms for a study can be quite a challenging task as it requires the construction of the actual control circuit. This simulator is useful to study only conventional HVDC operations.

2.6 Limitation of Analog and Digital Simulators

As HVDC control systems improve, the performance benefit of the above discussed simulators are challenged in the following ways:

1. PSCAD/EMTDC is only good for non-real time simulation and theoretical studies.
2. Analog simulators allow limited number of HVDC topology and any changes to controls take time and effort.
3. RTDS simulators are expensive for simple simulation.

2.7 Proposed Real Time Control System (RTCS):

This thesis will show a simple implementation of Real Time Digital Control (RTCS) system environment for HVDC simulation. In this approach, the main power network is still modelled on the analog simulator, but the controls are digitally simulated in the manner of the RTDS.

* Control schematics are developed in the PSCAD draft environment.

* A Run-time controller is developed for a personal computer.

* Digital signal processing system can take input from external devices and send computed output signals to the other external devices in the analog simulator.

It is expected that due to the improved technique for implementing the controls, various different control strategies can be studied in a short time. The network topology is however still constructed in analog hardware. It is felt that this is an inexpensive method to improve the capability of the analog simulator without having to purchase the prohibitively expensive RTDS platform.

CHAPTER THREE

METHODOLOGY AND IMPLEMENTATION

This chapter describes the flow of Real Time Control System (RTCS) and how the hardware and the software for the RTCS were designed. The RTCS is described first, and the later sections explain hardware and software involved in the design.

The problems discussed here are:

1. Description of the RTCS
2. PC based Graphical User Interface (GUI) for RTCS.
3. Interface between RTCS and real time analog simulation.

The RTCS uses the PSCAD/DRAFT, Texas C60 based DSP board and the analog simulator in the Electrical engineering department for graphical schematic capture, digital signal processing and analog simulation respectively.

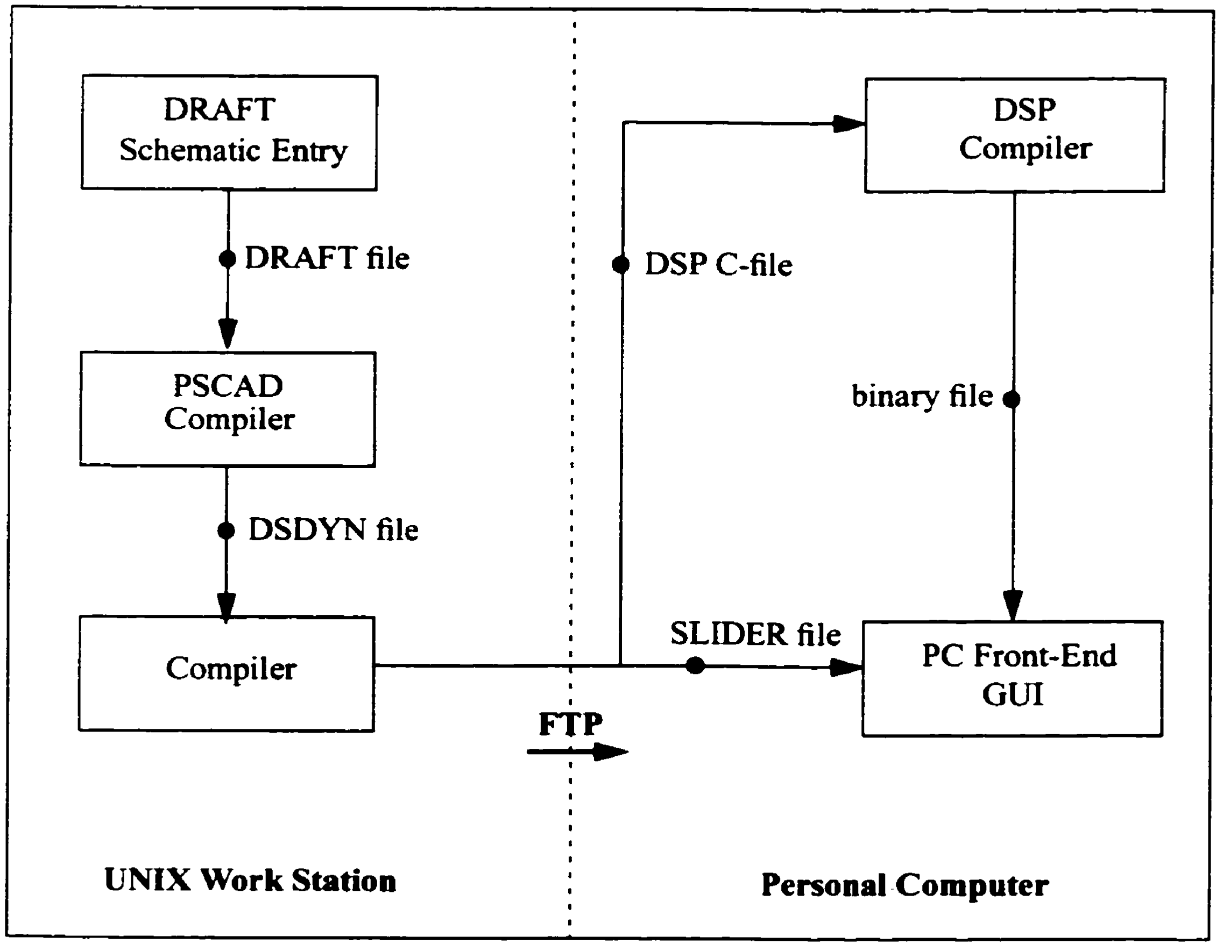


Figure 3.1

The actual RTCS implementation is shown in Figure 3.1. The PSCAD/ DRAFT front-end is used to enter and compile the schematics. The compiled schematic is translated into DSP C-code and an information file which consists of ranges and settings of variables. Both files are then downloaded to the personal computer where the DSP board resides and interfaced to the analog simulator. The DSP C-code is compiled in the PC environment and executed in the DSP boards using a GUI which runs under the Windows 3.1. operating system.

PSCAD/DRAFT is a very powerful graphical user front-end for the EMTDC simulation program. It permits the user to make schematic drawings of the control circuit. Components for the schematic can be copied from the vendor-supplied component library, or composed from basic elements. Each component has a definition file defining its graphical appearance, input and output ports and underlying FORTRAN simulation code. The circuit developed in DRAFT is compiled using the compiler attached to the DRAFT module. The compilation process results in a list of errors if any, and if none; produce the files necessary for run-time. One of the files produced for the run-time module is called the DSDYN file and contains FORTARN code for the run-time simulation. Rather than design a newer graphical front-end for the RTCS control layout, we chose to use PSCAD/DRAFT, as it is a tool which is gaining increased popularity.

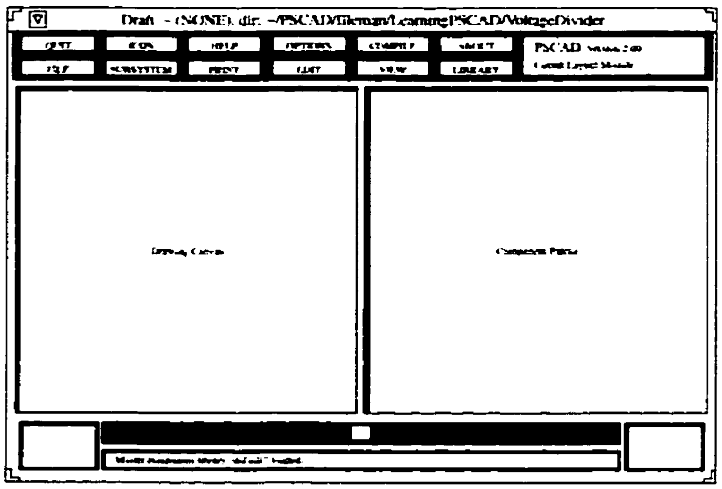

Draft - (NONE), dir: ~/PSCAD/Tiernan/LearningPSCAD/VoltageDivider
HELP
PRINT
LIBRARY
PSCAD

Figure 3.2

Using PSCAD/DRAFT features a comprehensive component library has been developed for the real time simulation. In these components definition file, FORTRAN code was replaced by DSP C code to implement the control system on the DSP board in real-time. Figure 3.3 shows one of the schematics of the PSCAD DRAFT module and its component library, in which real-time control system is simulated.

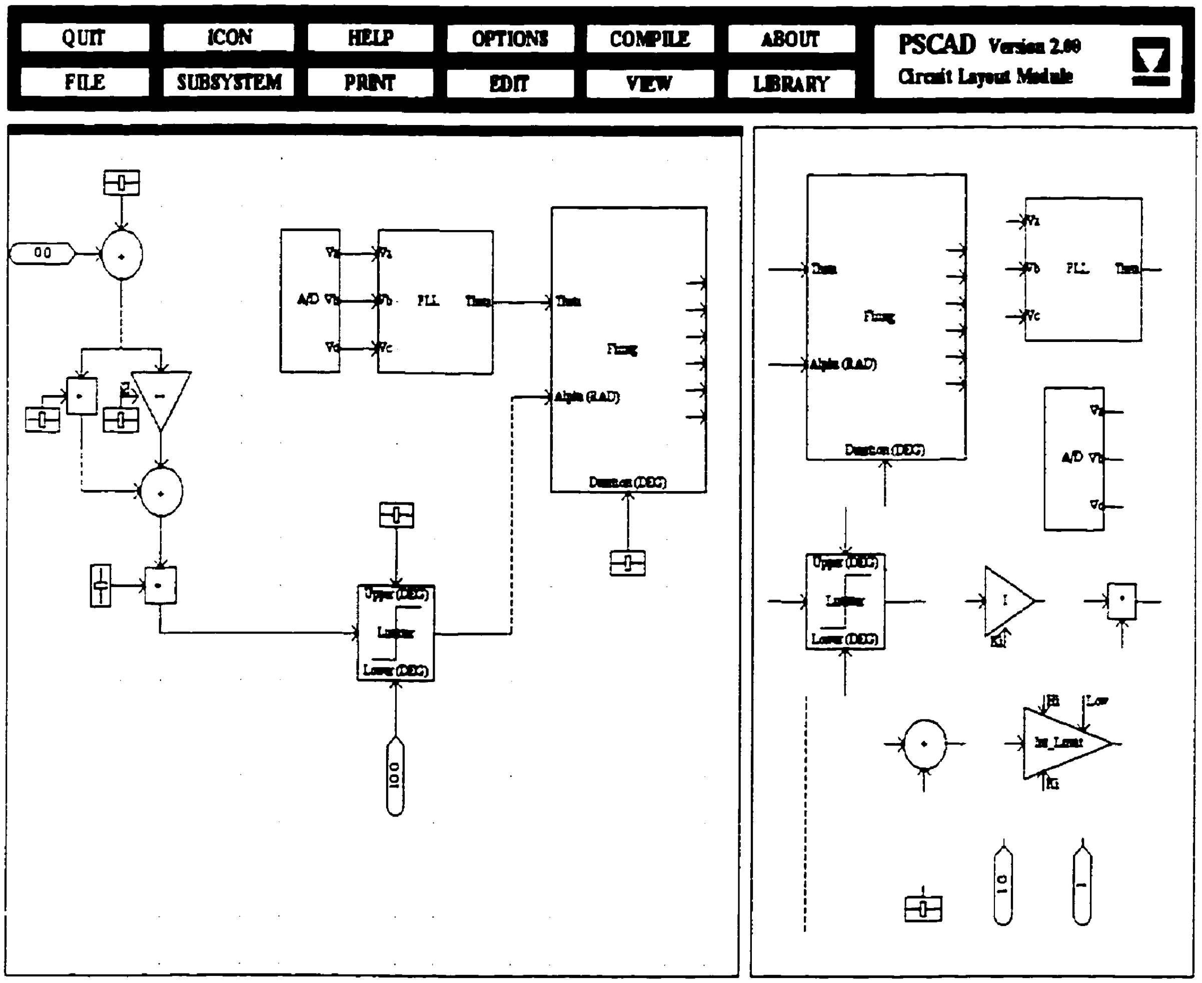


Figure 3.3

3.1 Draft Components for RTCS:

To use the PSCAD/DRAFT environment for the RTCS, components must be developed that generate C code for the DSP board. PSCAD/DRAFT components are defined using definition files containing the information required to display an icon representing the component and include the component in the circuit simulation. PSCAD/DRAFT is designed to generate FORTRAN code. The technique developed by me insert DSP code in this "FORTRAN" file. The FORTRAN file is then post-processed to strip off any remaining FORTRAN statements retaining only the pure DSP code for downloading to the DSP board.

Additional information in the designed PSCAD/DRAFT block contains the drawing of the component as it would appear on DRAFT palette as well as an information file indicating the settings and limits of variables that are required to be changed on runtime.

RTCS components also used all the features in the PSCAD component definition file except that the FORTRAN section was replaced by DSP C code with appropriate input and output parameters defined in the NODES section. The actual definition file for the RTCS component file is illustrated in the following Figure 3.4.

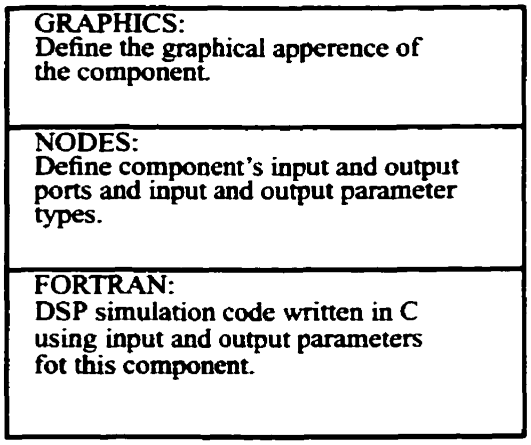

GRAPHICS:
Define the graphical apperence of the component.
NODES:
Define component's input and output ports and input and output parameter types.
FORTRAN:
DSP simulation code written in C using input and output parameters fot this component.

Figure 3.4

These parameters used in each RTCS components are defined and initialized in the FORTRAN section. Draft compiler uses the parameters defined in the NODES to interconnect the whole network and produce one syntactically unarranged programme file for simulation.

3.2 DSDYN Compilation

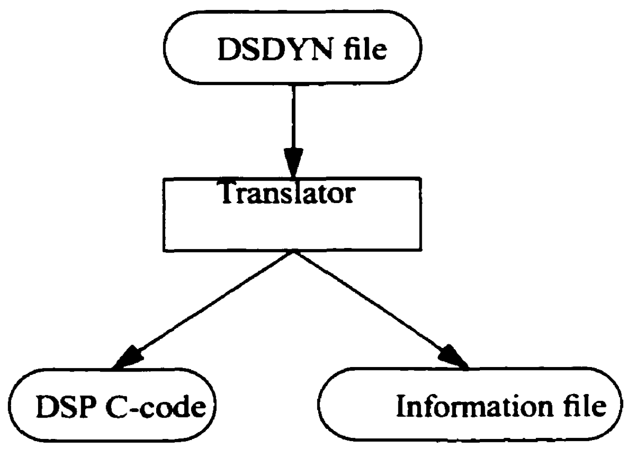


Figure 3.5

The DSDYN file produced from the DRAFT compiler is translated to pure DSP C-code using a translator. First, this translator removes all FORTRAN comments and other lines from the DSDYN file. Secondly, it converts some necessary FORTRAN lines into C-code syntax. Finally, it rearranges parameter definitions, declarations and produces DSP C-code for the schematic in the DRAFT canvas. In addition an information file is produced which contains the information required by the Run-time Control panel, such as the initial settings for the set-pot sliders and, initial gains, etc. Flowchart of the translator which produce DSP C-code from DSDYN file is shown in the following Figure 3.6.

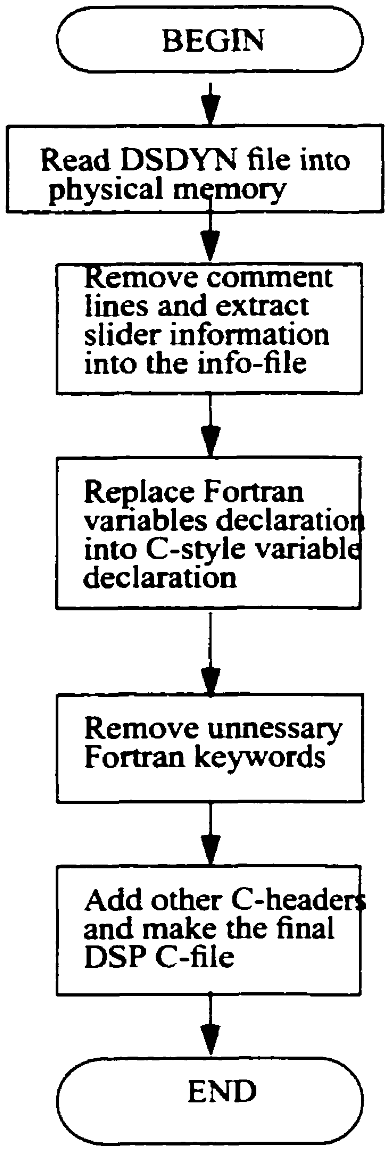

BEGIN
Read DSDYN file into physical memory
Remove comment lines and extract slider information into the info-file
Replace Fortran variables declaration into C-style variable declaration
Remove unnessary Fortran keywords
Add other C-headers and make the final DSP C-file
END

Figure 3.6

3.3 Real-Time Graphical User Interface

As mentioned earlier, the graphical schematic entry for generating DSP C-code and generating necessary files are done on Unix work station. The DSP C-code and the information files are sent to the PC where the DSP board and the other hardware were arranged.

3.3.1 The Graphical Runtime Control Panel

Once the developed model is down-loaded to the DSP board, a Run-time Graphical User Interface is used to communicate with the control system when it is running. Using this interface, changes of control settings can be made on-line. This panel automatically displays sliders and their parameters which were placed in PSCAD draft canvas in the initial drawing. During initialization, it reads the slider information file and place sliders and their parameters such as maximum, minimum, and initial values in the main GUI dialog boxes as shown in Figure 3.7. The user can also slide the appropriate slider within its specified range and pass its new parameter value to the DSP board during real-time. Whenever the user change a slider's position in the GUI dialogue, the GUI recalculate the parameter's value and passes all parameters to the DSP as an array without halting the DSP execution.

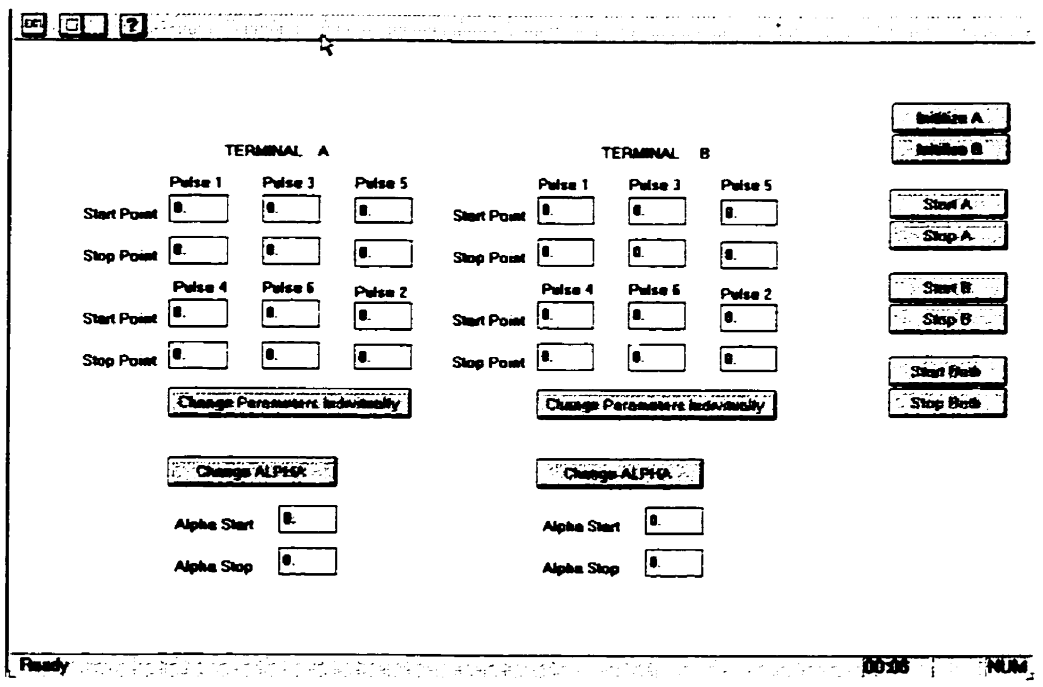


Figure 3.7

The above GUI is designed to control two independent DSP board based firing circuits. The two DSP board based firing circuits are identical in structure and in characteristics of operation. Each one has six output lines for 6-pulse firing circuits and three input lines to measure 3-phase AC line voltage.

The GUI is designed for the Microsoft Window 3.1 platform. Microsoft Visual C++ and Microsoft Foundation Classes (MFC) were used in the GUI development. Additional classes CDlgToolBar, CDlgStatusBar, CModelessDialog, and CModelessMain, which are written by Microsoft Product Support Service, were also used in the development. GUI class hierarchy is shown in following figure 3.8. Each rectangle represents a class and the arrow between classes represents the association between them.

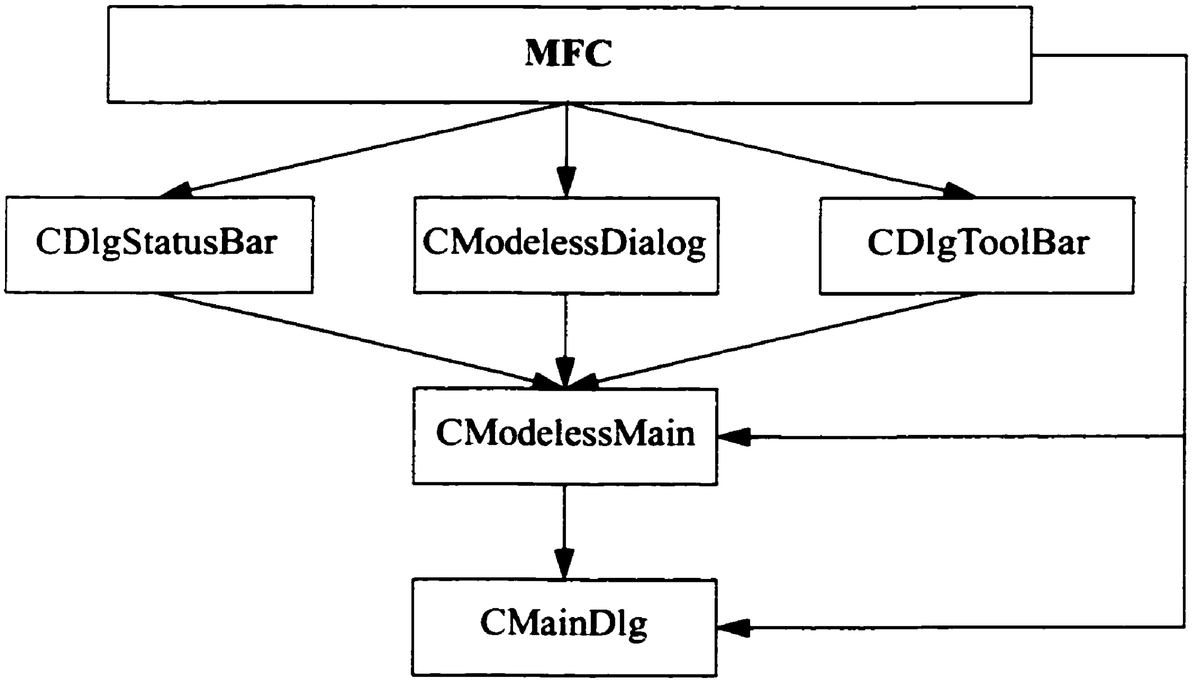


Figure 3.8

In the class hierarchy diagram shown in Figure 3.8. the CMainDlg class is derived from classes MFC and CModeleesMain. This class assigns parameters and functions to communicate with DSP boards. The parameters and functions are shown in following Table 3.1.

alpha_start_A[1]		alpha_start_B[1]	
alpha_stop_A[1]		alpha_stop_B[1]	
start_A[6]	stop_A[6]	start_B[6]	stop_B[6]
loadstat_A		loadstat_B	
start_vec_loc_A		start_vec_loc_B	
stop_vec_loc_A		stop_vec_loc_B	
Create()	DoDataExchange()	OnUpdateTime()	OnClose()
OnClickedInitA()		OnClickedInitB()	
OnClickedStartA()		OnClickedStartB()	
OnClickedReloadA()		OnClickedReloadB()	
OnClickedSameloadA()		OnClickedSameloadB()	
OnClickedStartBoth()			
OnClickedStopBoth()			

Table 3.1

Parameters in the class CMainDlg can be divided into two groups. The first group is used for user input and the second group for the initialization between GUI and the DSP boards. An explanation of the parameters is shown in Table 3.2.

Parameters	Meaning
alpha_start_A[1], alpha_start_B[1]	Firing start position in degree for both terminals
alpha_stop_A[1], alpha_stop_B[1]	Firing stop position in degree for both terminals
start_A[6], start_B[6]	Firing start position converted into radian
stop_A[6], stop_B[6]	Firing stop position converted into radian
loadstat_A, loadstat_B	DSP program load flag
start_vec_loc_A, start_vec_loc_B	Holds the 24 bit address to put start parameters
stop_vec_loc_A, stop_vec_loc_B	Holds the 24 bit address to put stop parameters

Table 3.2

Functions Create(), DoDataExchange(), OnUpdateTime() and OnClose() in the class CMainDlg are used for the data exchange between user input (GUI) and the CMainDlg class. Other ten functions are activated by clicking buttons in the GUI environment. OnClickedInitA() and OnClickedInitB() loads the executable COFF Object file into DSP board's memory and set all six output pins (six firing pulses) to zero state. Furthermore, these two functions will display error messages for unsuccessful loading of the object file into the memory, otherwise disable the assigned button in the GUI. OnClickedStartA() and OnClickedStartB() will start the DSP program in the assigned DSP board or OnClickedStartBoth() will start both DSP board simultaneously.

3.4 Hardware Arrangement

A TMSC30 based DSP board is used to generate six firing pulse in real time and a nine channel Analog to Digital (A/D) converter card is used to read data from three-phase line voltage and DC line current. The A/D converter card and the DSP board were connected through a special parallel expansion (DSPLINK) port in the DSP card. All the communication between PC and the DSP board is done via the PCs I/O space in the ISA bus. To avoid the I/O port conflict between other standard boards in the PC, 290 hex and 390 hex base address locations are used by the two DSP boards respectively. The whole system is controlled by PC based front-end software. Figure 3.10 shows the block diagram for the this system. The three-phase line is isolated by using step-down transformers to get zero to five volts input to the A/D converters. The A/D converter is triggered from the DSP board from it's timer-0 interrupt pulse line.

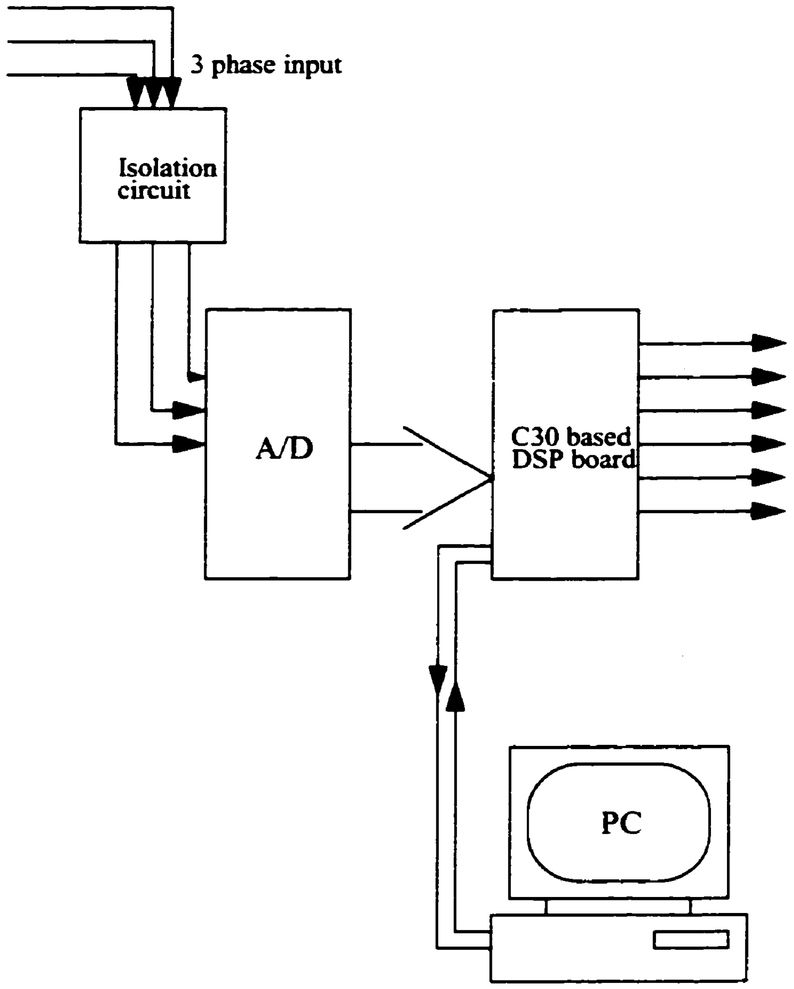


Figure 3.10

Six pins in the DSP serial port such as CLKX0, FSX0, DX0, CLKX1, FSX1 and DX1 are assigned as firing output pins. These pins are connected to each valve in the 6-pulse Converter/Inverter assembled in the analog simulator. To avoid data loss from the A/D converter, the DSP's interrupt routine and the external A/D converter are triggered by same TIMER0 peripheral module in the TMS320C30. The main program always waits until it gets an interrupt pulse from TIMER0 to activate the interrupt routine and the A/D converter. All the main processes such as the Phase Lock Loop (PLL), ramp generation and firing decisions are carried out in the interrupt routine and its flowchart is shown in Figure 3.11

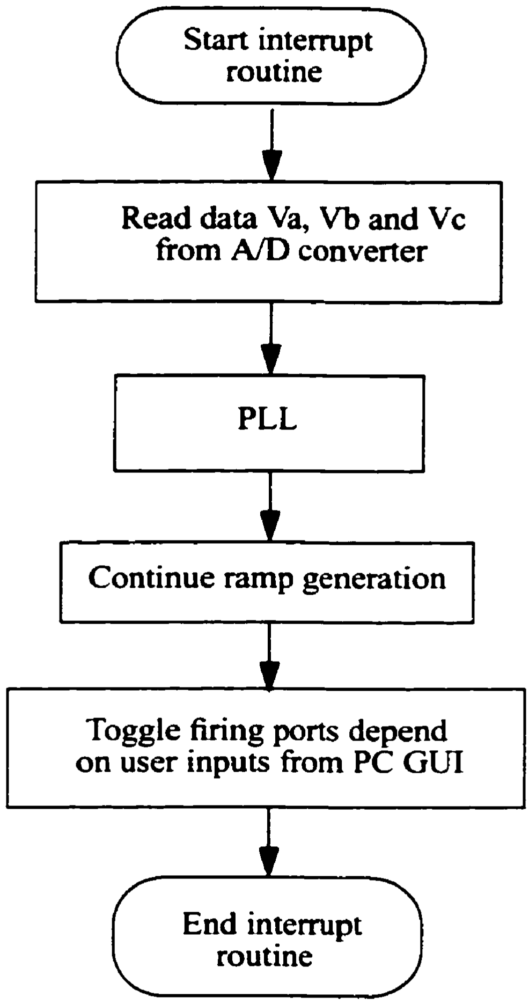

Start interrupt routine
Read data Va, Vb and Vc from A/D converter
PLL
Continue ramp generation
Toggle firing ports depend on user inputs from PC GUI
End interrupt routine

Figure 3.11

3.5 Phase Locked Loop

A Phase Locked Loop (PLL) was employed to follow the system voltage to determine the precise phase of the source. The phase locked loop looks at the line to neutral voltage of the bus and produces six ramps which are then compared to the required firing angle of a particular valve. When the ramps exceeds the values of the firing order, a firing pulse will be given for the particular thyristor gate. This pulse will be continuous throughout the duration that the valve is required to turn on to ensure that no premature extinction takes place, although the ideal thyristor does not require a continuous pulse once it is already on. The employed phase locked loop control circuit is shown in Figure 3.12.

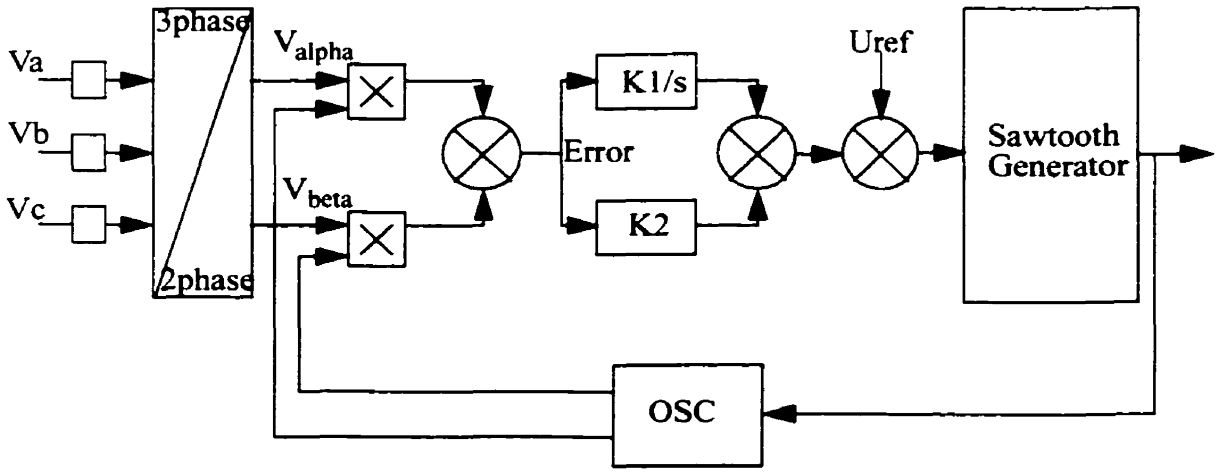


Figure 3.12

This phase lock loop is of the "*Transvektor*" type, i.e.; it uses sequence transformation to obtain the phase difference between the positive sequence (phase a) component of the three phase voltage and the VCO output. One advantage of this type of grid control circuit is its superior immunity to disturbances and harmonic distortion on the AC synchronising voltage [1].

The three phase voltage derived from the A/D converter are Va, Vb and Vc. Using a 3-phase to 2-phase transformation the direct and quadrature axes voltage, V_{alpha} and V_{beta} respectively, are derived according to the following equations[1].

$$V_{alpha} = \frac{1}{3} \cdot (2.Va - Vb - Vc) \text{ ———————————— } (3.1)$$

$$V_{beta} = \frac{1}{\sqrt{3}} \cdot (Vb - Vc) \text{ ———————————— } (3.2)$$

An Error signal is generated according to the equation and is acted upon by a PI controller with proportional gain K1 and integral gain K2.

$$Error = V_{alpha} \cdot \cos\Theta + V_{beta} \cdot \sin((\Theta) \text{ ————————— } (3.3))$$

It can be shown that this error is sinφ, where φ is the phase angle between the positive sequence component of the ac voltage and the VCO output. The nominal frequency of the Sawtooth Generator is controlled by a reference voltage Uref. The output of the Sawtooth Generator, which is limited between 0 and $2.\pi$, generates the timing Sawtooth waveform and it's utilised to derive the firing pulses.

The sawtooth output from the Sawtooth Generator is used to generate another five similar sawtooth waves 60^o apart from each other. These six waveforms and the user inputs from the PC based GUI are used to activate the six different firing pulses through the output pins in the serial port. Each pulse's starting position and the duration are determined from the six set of user input parameters. On the other hand, the user can input one set of parameter, to generate six firing pulses with equal duration and 60^o apart from each other.

3.6 Current Controller

In rectifier current control mode, the converter's firing angle α is controlled with a feedback control system. The DC voltage of the converter increases or decreases according to equation 3.4. The DC voltage variation with firing angle α is used as a mechanism to adjust the DC current to it's set-point I_{ref}. For example, if the DC current exceeds I_{ref}, then I_{Error} become greater than zero, and the firing angle will be increased. According to equation 3.4, V_d decreases in an attempt to decrease the value of I_d.

$$V_d = \frac{3 \cdot \sqrt{2}}{\pi} \cdot V_l \cdot \cos\alpha - \frac{3}{\pi} \cdot X_c \cdot I_d \text{———————————} (3.4)$$

The firing angle α is constrained to lie between limits α_{min} and α_{max}. No further control on α is possible once the smallest possible angle α_{min} or maximum possible angle α_{max} is reached.

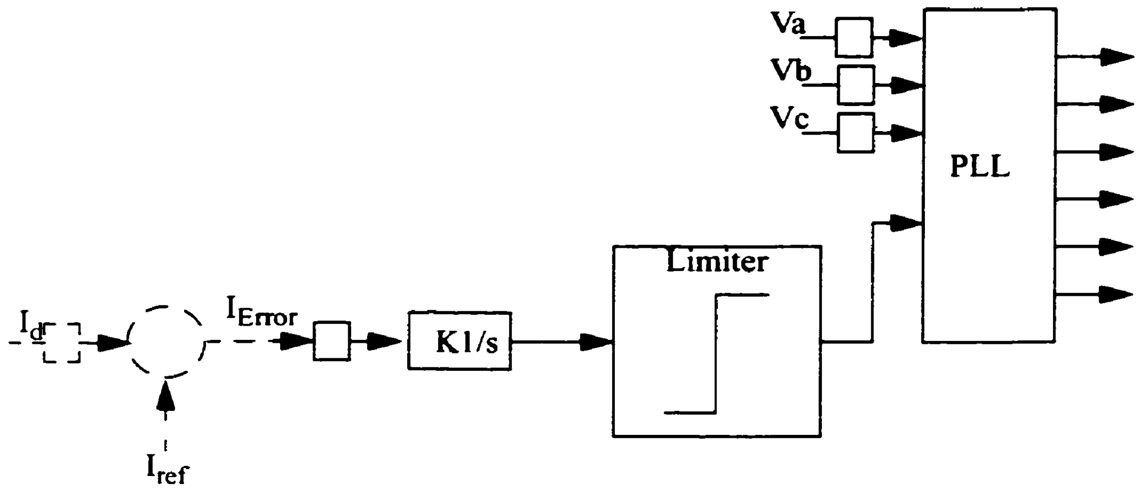


Figure 3.13

The user inputs the parameters such as α upper limit (α_{max}), α lower limit (α_{min}) and pulse duration using the run-time GUI.

3.7 RTCS Components

As mentioned earlier, all RTCS components were developed using DRAFT component features.

PLL Block:

One of the components developed to demonstrate this thesis was a phase locked loop (PLL). It takes the three phase input voltage and calculates the (theta) precise phase of the input source based on the PLL discussed in section 3.5. This phase locked loop produces a theta ramp reference to one reference to the positive sequence of the ac bus voltage (phase a). Its PSCAD graphical appearance is shown in Figure 3.14.

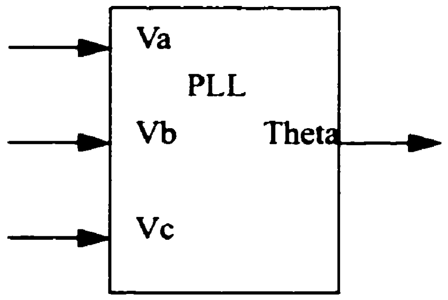


Figure 3.14

Firing Pulse Generation Block:

Another major components developed is the firing block. This is a DSP-board specific component and takes phase angle (theta), firing angle (alpha) and firing duration as inputs and outputs six firing pulses based on these inputs. Outputs from this block cannot be used in the Draft canvas. Its only output is through the DSP-board serial output-port and is the actual firing pulse sequence to the thyristors. Its PSCAD appearance is shown in Figure 3.15.

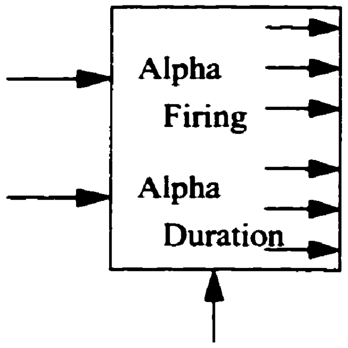


Figure 3.15

Combination of PLL and Firing blocks produce six firing pulses to turn ON a particular valve. When the ramps from PLL exceeds the value of the alpha order, the firing pulse will be given to the valve. This pulse will continue throughout the duration specified to the firing block. This will ensure that no premature extinction takes place.

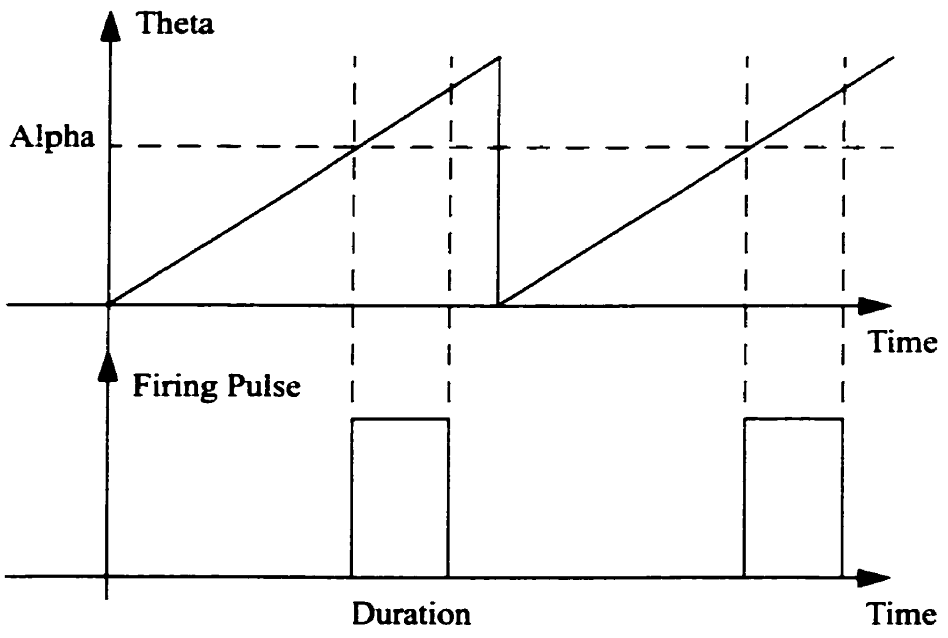


Figure 3.16

Other Components:

Similarly several other control components were developed. These include: an integrator, a multiplier, an adder and measurement blocks to interface to the real world.

The integrator block was modelled by converting the Laplace Transform of the transfer function (Y(s) = (1/s).X(s)) to a suitable difference equation using Eulers approximation i.e.

$$y(t) = y(t-\Delta t) + \frac{u(t) + u(t-\Delta t)}{2} \cdot \Delta t \text{------------------------} (3.4)$$

The measurement blocks for voltage and current interface the A/D converters with the digital control model running on the DSP board.

3.8 Hardware Interfacing

The analog simulator is connected to the DSP board via nine channel A/D converter and DSP-board serial port. Three phase line voltage from the analog simulator pass through isolation transformers to the A/D converter. The rectifier DC current in the analog simulator is passed through a current transducer and converted into voltage reference. The converted DC current's voltage reference is also connected to one of the channels in the A/D converter. Six output pins in the DSP serial port are connected to each of the valves in the 6-pulse Converter/Inverter assembled in the analog simulator.

3.9 Real-Time Implementation.

The DSP C-Code file and the slider information file which were extracted from DSDYN file are transferred to the personal computer where the DSP cards and other hardware resides. The DSP C-code file is again compiled using the compiler designated for the DSP board which resides in the PC. Finally, the binary output file from the DSP compiler and the slider information files are used by the run-time graphical user interface to run the actual system in real time. After activation, the graphical user interface (GUI) in the PC reads the slider information file and it automatically places sliders in the GUI canvas. The number of sliders and their settings will be the same as in the PSCAD draft canvas and these settings are passed through the slider information file.

CHAPTER FOUR

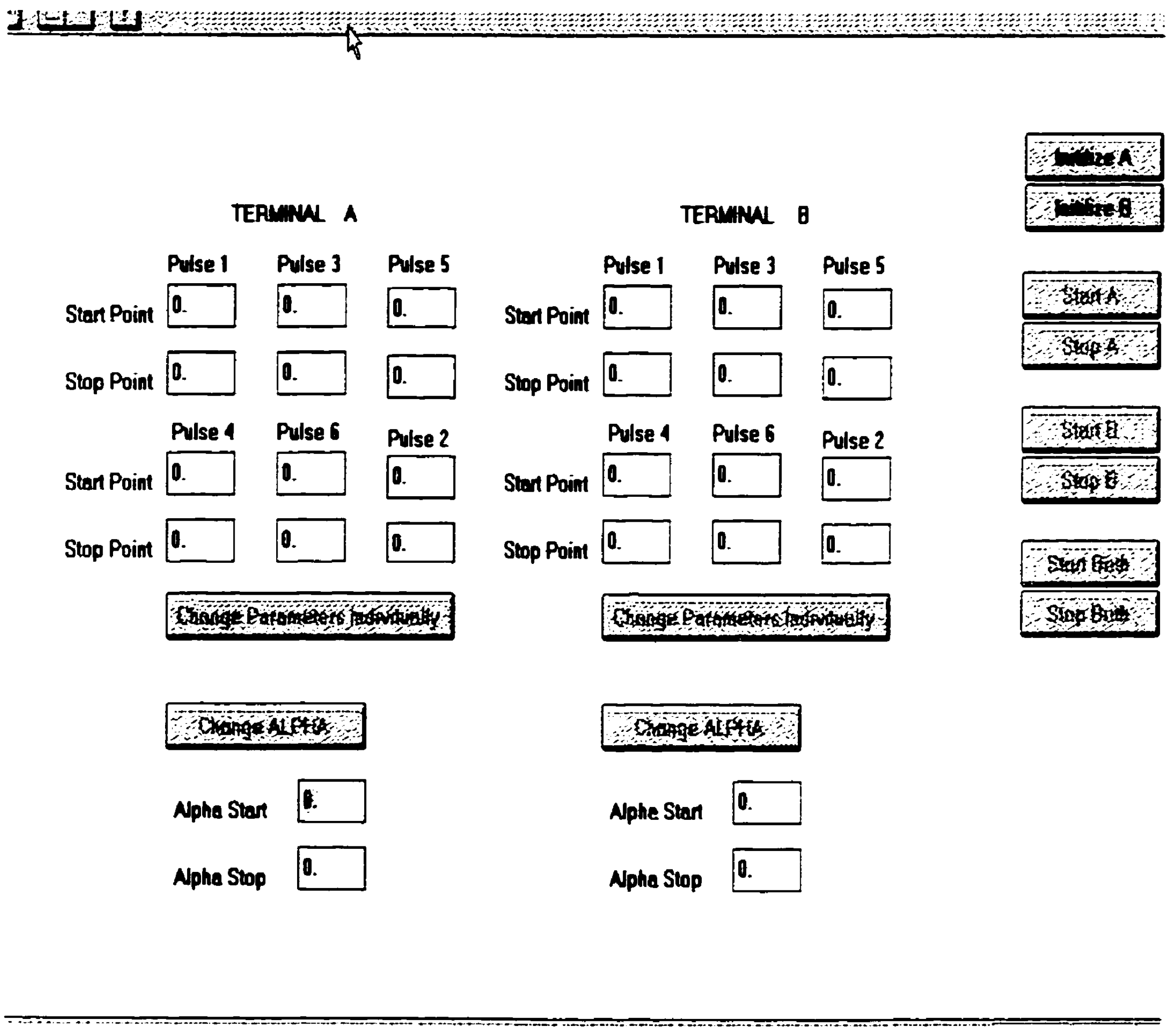
TESTING AND REAL TIME SIMULATION

In the previous chapters, it has been described how the hardware and the software were developed and has been shown that this hardware was designed to interface with the existing HVDC analog simulator and PSCAD computer simulation tool. In this chapter, results are presented for the control system which was tested independently and also together with the analog simulator. Furthermore, the PSCAD and RTCS interface were also tested. These tests are listed below:

1. Verification of interface between PC based GUI, two DSP boards and A/D converter boards.
2. Testing the real-time functionality of the PLL and the firing pulse generation.
3. Verification of the smallest possible firing pulse.
4. Generating all six firing pulses from one DSP board.
5. Testing the interface between the DSP-System and the analog simulator.
6. Verification of the RTCS as an entire system a) Testing PSCAD with RTCS b) PSCAD and RTCS combination tested using a simple feed-back loop network.

Test 1: Interface Verification

First the interface between the front-end Graphical User Interface (GUI), two C30 based DSP boards and two A/D converter boards was tested. The interface appears as shown in Figure 4.1. The DSP program is loaded by clicking "INITIALIZE A" or "INITIALIZE B" buttons in the GUI environment. If these DSP programs successfully load into the designated DSP boards, the INITILIZE button will be greyed and error message won't appear in the GUI environment. Different panel screens appear for various phases of tests.



TERMINAL A

	Pulse 1	Pulse 3	Pulse 5
Start Point	0.	0.	0.
Stop Point	0.	0.	0.

	Pulse 4	Pulse 6	Pulse 2
Start Point	0.	0.	0.
Stop Point	0.	0.	0.

Alpha Start 0.
Alpha Stop 0.

TERMINAL B

	Pulse 1	Pulse 3	Pulse 5
Start Point	0.	0.	0.
Stop Point	0.	0.	0.

	Pulse 4	Pulse 6	Pulse 2
Start Point	0.	0.	0.
Stop Point	0.	0.	0.

Alpha Start 0.
Alpha Stop 0.

Figure 4.1

The above GUI can communicate with individual DSP boards and the user can set the firing angle starting and ending points. The user can also test an individual pulse or set of six pulses. START and STOP buttons start and stop the DSP programs in the designated DSP boards. Change buttons are active after starting the program execution and when pressed will change the parameters in the DSP program, according to the values on the sliders.

Test 2: Real-Time Functionality

After successful initialization, parameters were entered in GUI to check the functionality of the Phase Lock Loop and firing pulses which originated from the serial port in the DSP-board. The firing angle α was entered and the system was started by clicking the "START" buttons. In this testing, one line-line voltage Vac and one firing pulse was observed on the Tektronix TDS 420A digital oscilloscope and the data were recorded and shown in Figure 4.2. The line frequency was 60Hz, the entered α was 15 degree with a pulse width of 30 degree.

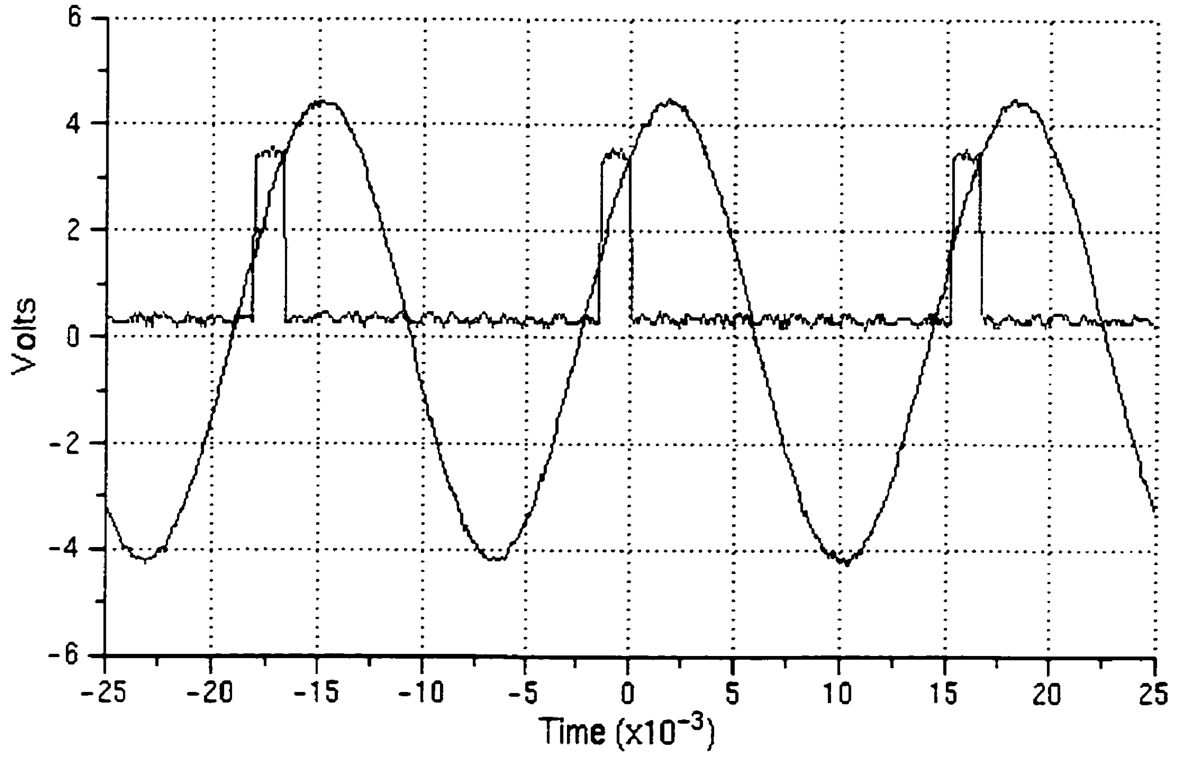


Figure 4.2

The above Figure 4.2 shows the obtained result as we expect. This test result verifies the real-time functionality of the PLL and the firing pulse generator which were running on the DSP board.

Test 3: Narrow Firing Pulse

Possible firing duration was tested by entering firing angle α to 15 degree and the pulse duration to 1 degree. The following Figure 4.3 shows line-line voltage Vac and the firing pulse with duration one degree. Firing duration below one degree couldn't be achieved during this test, due to the 44.4μs (22.5KHz) time step used.

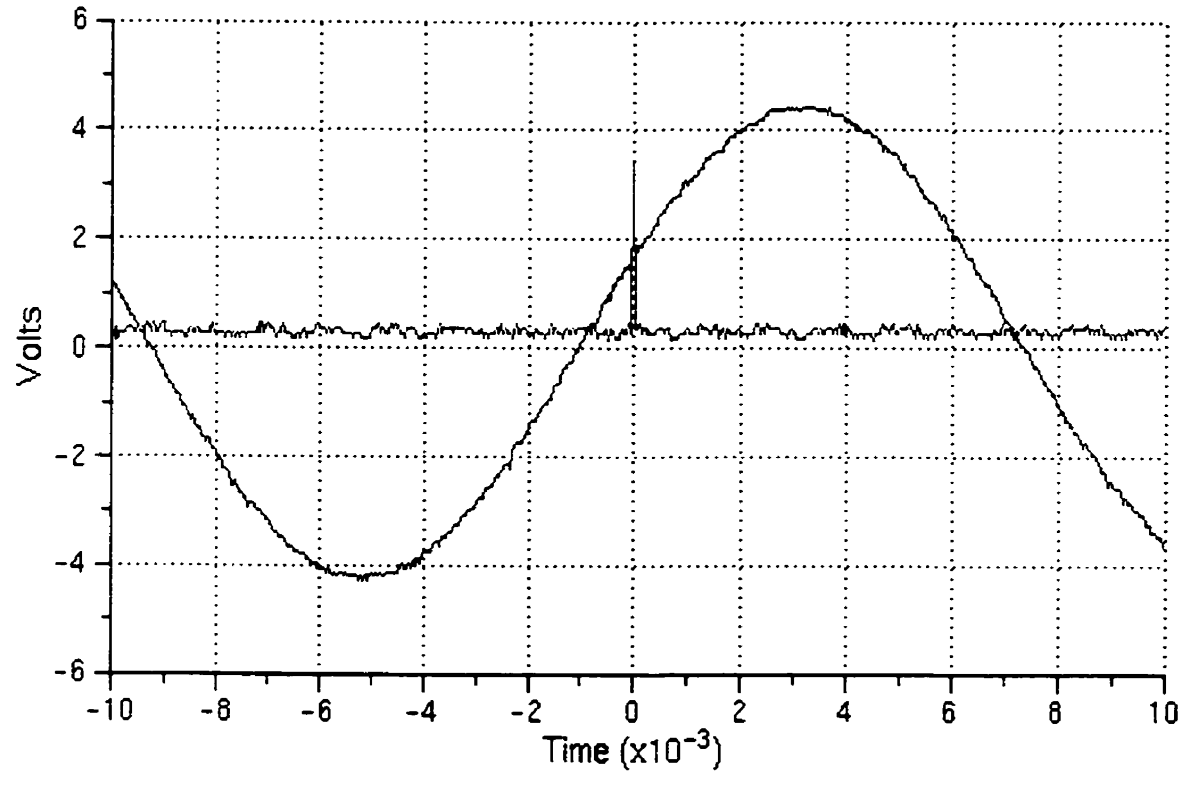


Figure 4.3

Test 4: Generating a set of firing pulse

All six pulse lines was tested for various firing angle and pulse duration. Figure 4.4 shows one phase voltage Va measured before the A/D converter and all six pulse lines which were observed on the Tektronix TDS420A digital oscilloscope.

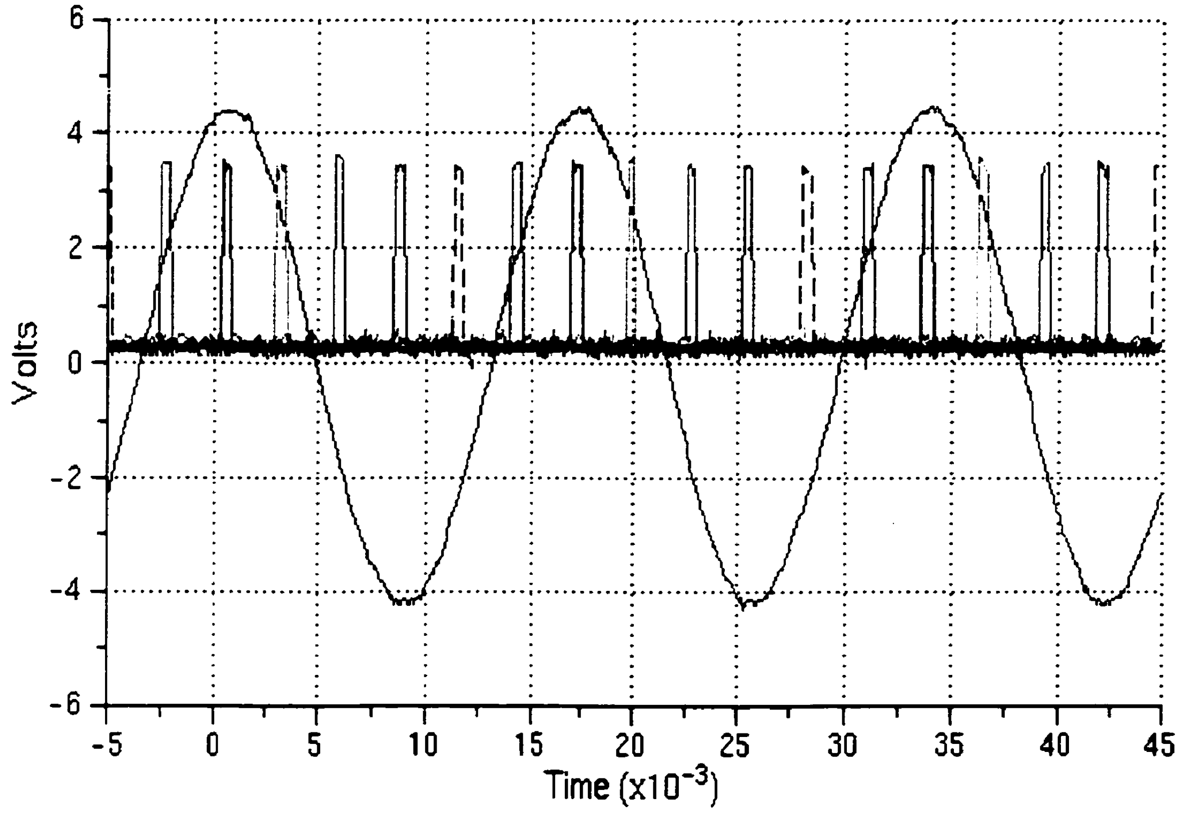


Figure 4.4

This test result shows that the DSP board can handle the necessary pulse generation process in real-time for a user given firing angle.

Test 5: DSP and the Analog Simulator Interface

The entire system was connected with the analog simulator to check the functionality of the firing circuit with the 6-pulse rectifier operation. The structure of the tested 6-pulse rectifier is shown in Figure 4.5. The firing pulses from the DSP board are routed through an optocoupler to the thyristor cards where they are amplified individually and sent to the thyrister gates. The optocoupler is used for electrical isolation between the DSP system and the power system model. Output DC voltage from the rectifier was recorded for different firing angles α.

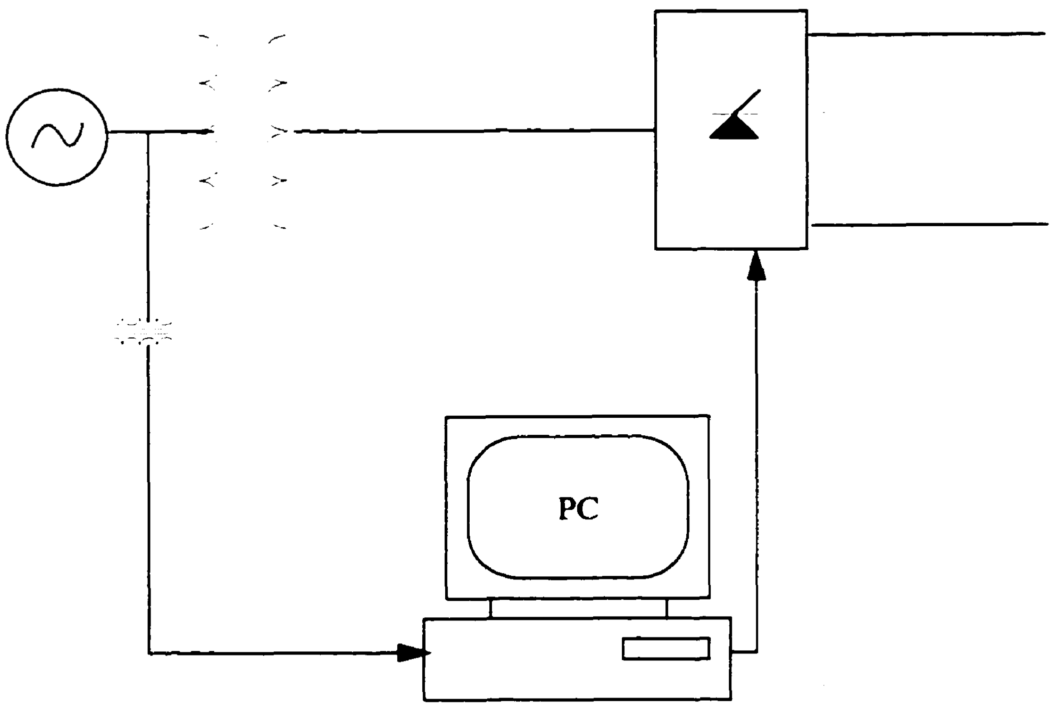


Figure 4.5

The line-neutral voltage after the transformer was adjusted to about 60 volts. The firing angle α and its duration was loaded to zero degree and 120 degree respectively from the PC's GUI. Figure 4.6 shows the line- neutral voltage and the DC voltage from the rectifier.

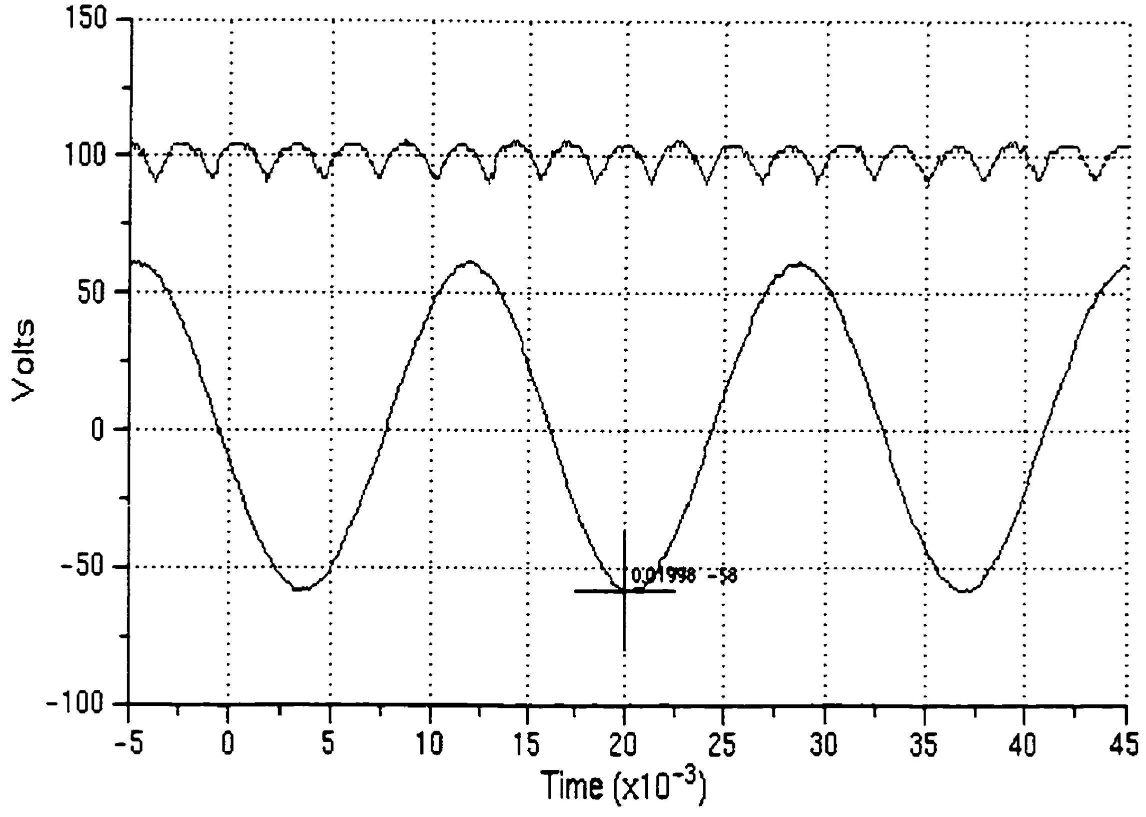


Figure 4.6

The real time output from the analog rectifier matches the theoretical result of a six pulse rectifier output. This result also demonstrates that the interfacing between PC based GUI and DSP board with the analog simulator works well and the run time GUI, DSP programs and the hardware interface for this thesis were working correctly for a simple case.

4.1 Validation of Interface between PSCAD and RTCS:

In the previous chapter, it has been shown that the system was designed to interface the PSCAD draft to the real world control system RTCS. In this section, several test cases will be presented to validate the practical usability of this system.

The basic steps involved in the interface between PSCAD/DRAFT and the RTCS are as follows.

1. Draw a schematic diagram of the HVDC system from the specially designed component library and compile using PSCAD/DRAFT compiler to generate the DSDYN file. The schematic diagram can also contain sliders to change parameters during real-time simulation. This slider parameter name, initial value, minimum value and maximum value can be initialized in the PSCAD/DRAFT.

2. The generated DSDYN file passed through another compiler/translator to generate real DSP C-code and the information file about the sliders in the DRAFT. The information file used in the RTCS front end and contain the slider parameter names, minimum, maximum and initial values which are displaced in the real-time control dialog. The above two steps are done in the Unix environment and finally, the DSP C-code file and the silider information files are transferred to the PC via a file transfer program.

3. The transferred DSP C-code again is compiled in the DOS environment using a compiler designated for the DSP board and produces an executable binary file. The above DSP executable file and the slider information file are transferred to the directory where the window based run-time executable file resides.

4. Final real-time simulation is done by executing a window based dialog environment. The real-time simulation process in the dialog is as follows.

•A) Sliders which are placed in the PSCAD/DRAFT canvas will be placed in the run-time dialog environment by clicking the button **Load-Info**. In this process, the front program reads the information file and places all the sliders and its parameters which were initialized in the PSCAD/DRAFT.

•B) By clicking the **Initialize** button, the executable DSP binary file is loaded in the DSP board where the real time execution takes place.

•C) Buttons "START" and "STOP" are designated to start and stop the real-time simulation. During the simulation, parameters can be changed by clicking arrows at the end of the sliders. Changed parameter values appear in small windows near the appropriate slider.

Test 6a: Testing PSCAD and RTCS interface

The first test was carried out to check the PSCAD and RTCS interface. This test verified the working of the PSCAD components and compiler as well as the real time control system. The block diagram of this system is shown in Figure 4.7. A simple schematic diagram of the control system was drawn in the PSCAD/ DRAFT to generate firing pulse with a range of firing angle α. Figure 4.8 shows the control schematic diagram for the tested system in the PSCAD/DRAFT environment. Since current feedback is not used, we manually enter the current error. This ramps the integrator output to a limit.

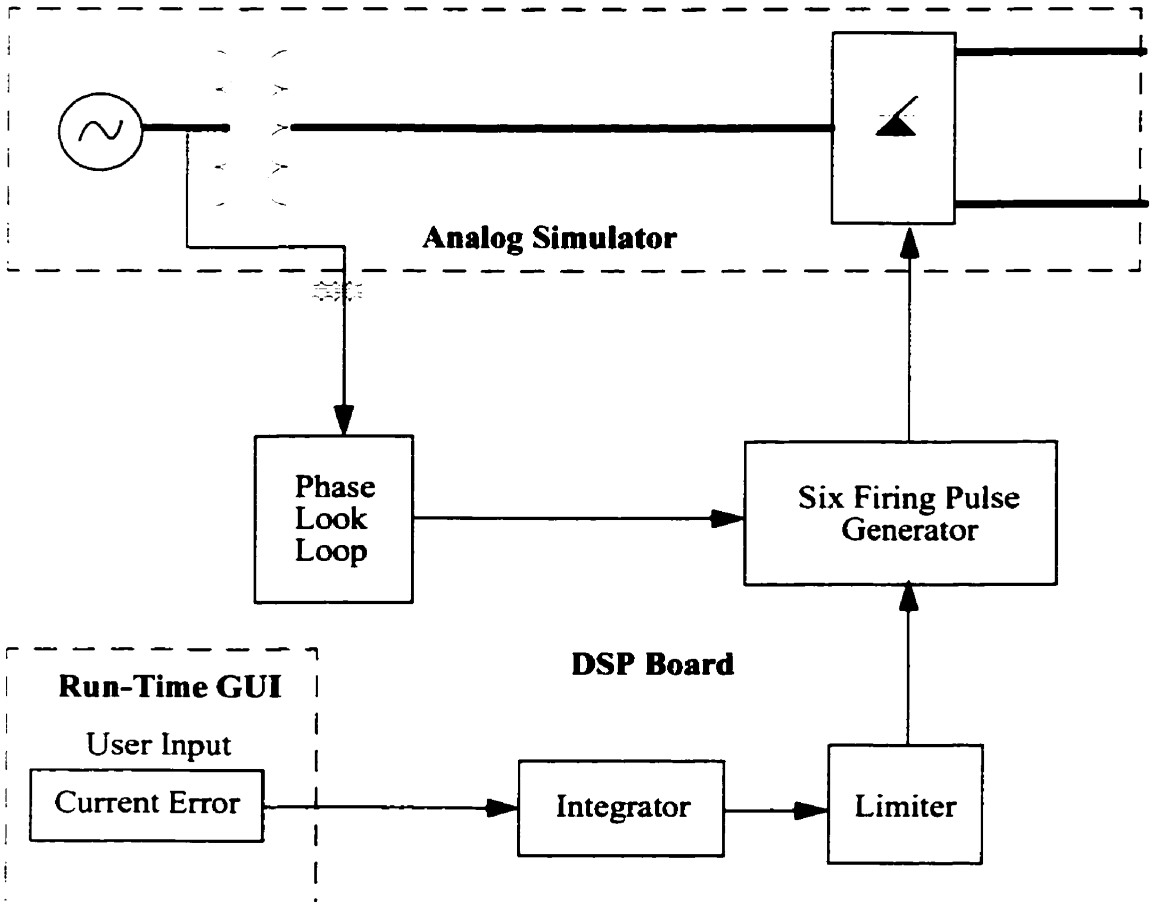


Figure 4.7

In this test the system is initialised with a given α upper limit (α_{max}), α lower limit (α_{min}) and pulse duration. The current error can be fed by sliding the slider in the run-time GUI. Whenever the system receives positive current error, the firing pulse (α) should move towards the upper limit (α_{max}), and for the negative current error should move towards the lower limit (α_{min}). For zero current error, the pulse should stay in its current position, ie the firing angle should be constant.

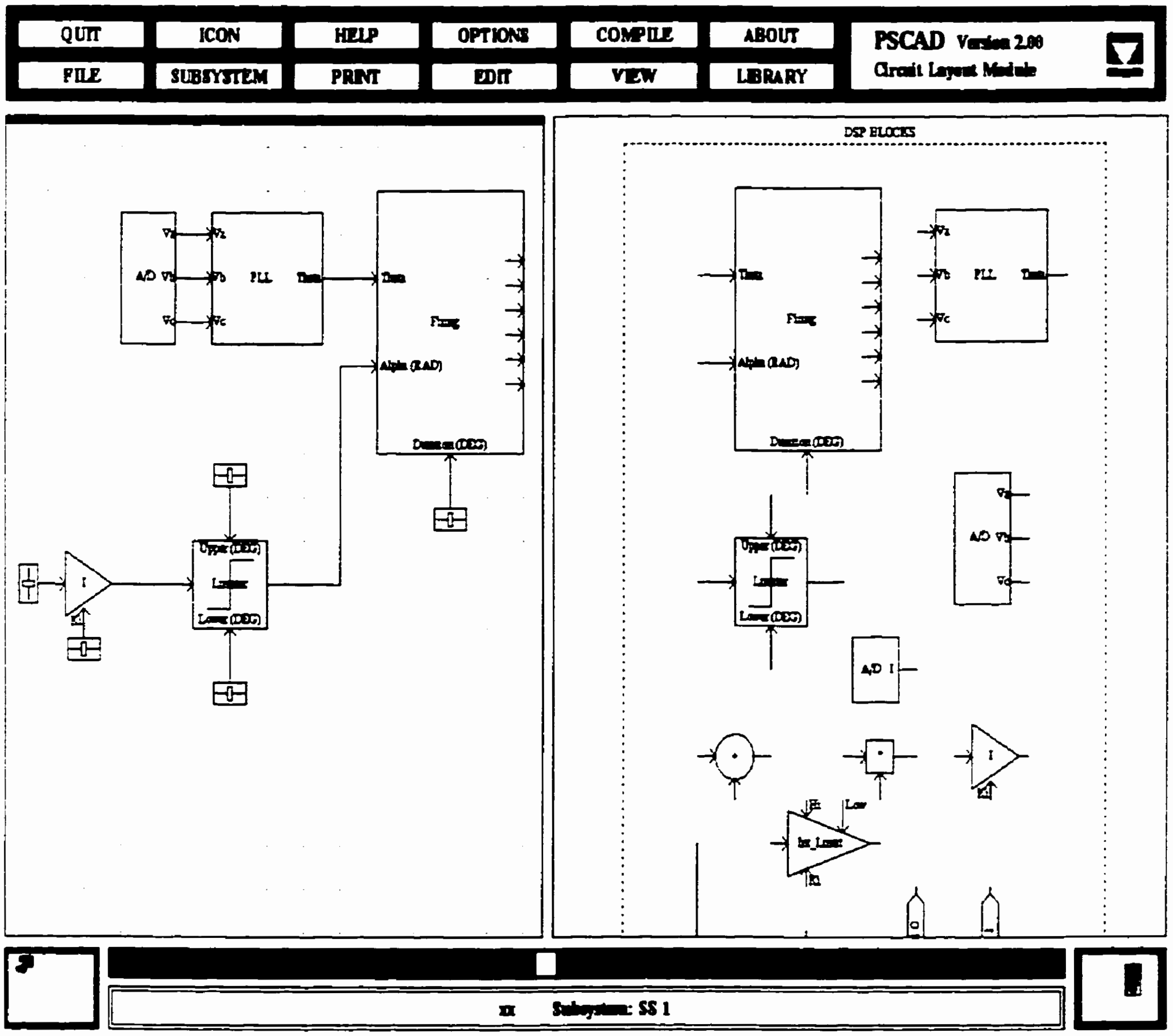
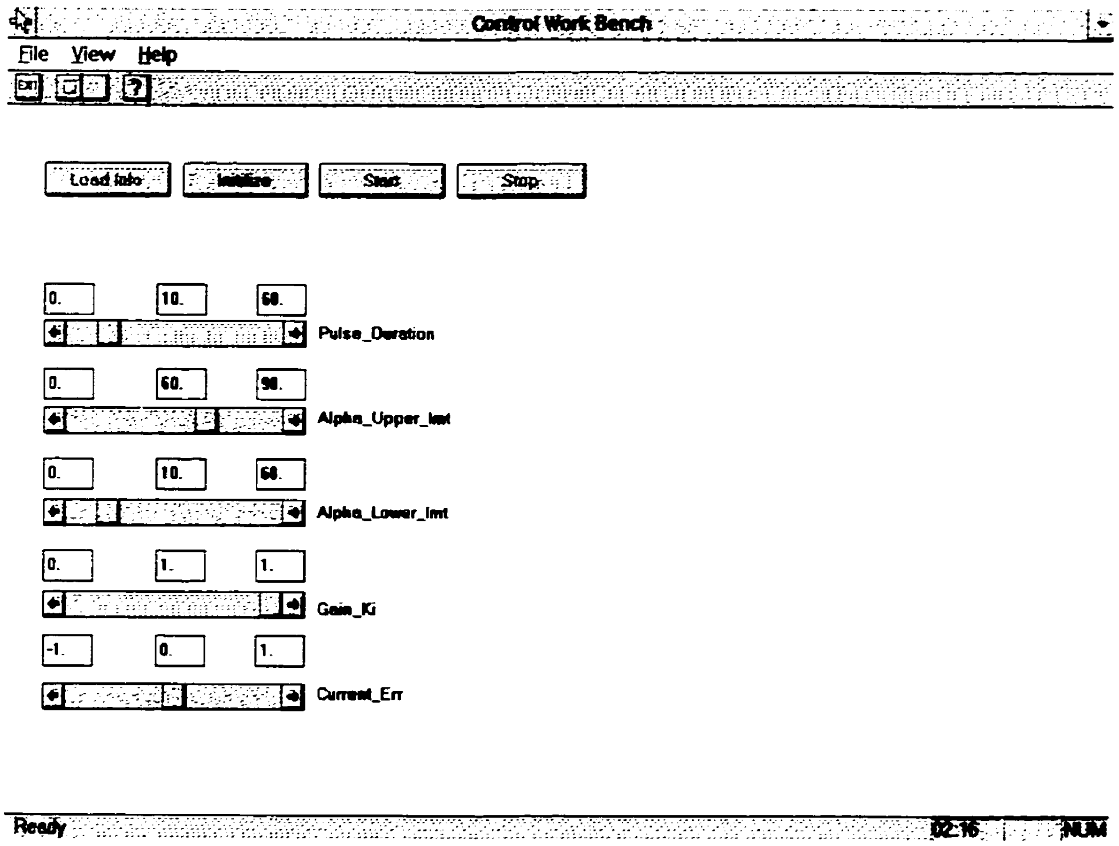


Figure 4.8

As explained earlier, the DRAFT circuit passed through the PSCAD compiler to produce the DSDYN file. This DSDYN file is passed through the second compiler/translator and produce DSP C-code and the slider information file. Generated, DSP C-code also compiled by a DSP compiler without any errors. This ensures that the designed PSCAD components and the second compiler made the proper C-code which is used to run in the DSP board.

Next, we used the windows based run time environment to execute the DSP code in real time. The run time environment automatically load the sliders in the GUI window. The resulting run time panel is shown in Figure 4.9 with the same number of sliders in the PSCAD/DRAFT.



0. 10. 60. Pulse_Duration

0. 60. 90. Alpha_Upper_lmt

0. 10. 60. Alpha_Lower_lmt

0. 1. 1. Gain_Ki

-1. 0. 1. Current_Err

Figure 4.9

The system was started and one line voltage, one pulse were observed on the oscilloscope. The current error was changed between a positive and a negative value. During this change, the observed firing pulse moved toward α upped limit and α lower limit as expected. The above step ensure the proper interfacing between run time environment and the DSP board. The oscilloscope output was obtained for one instance and its shown in Figure 4.10.

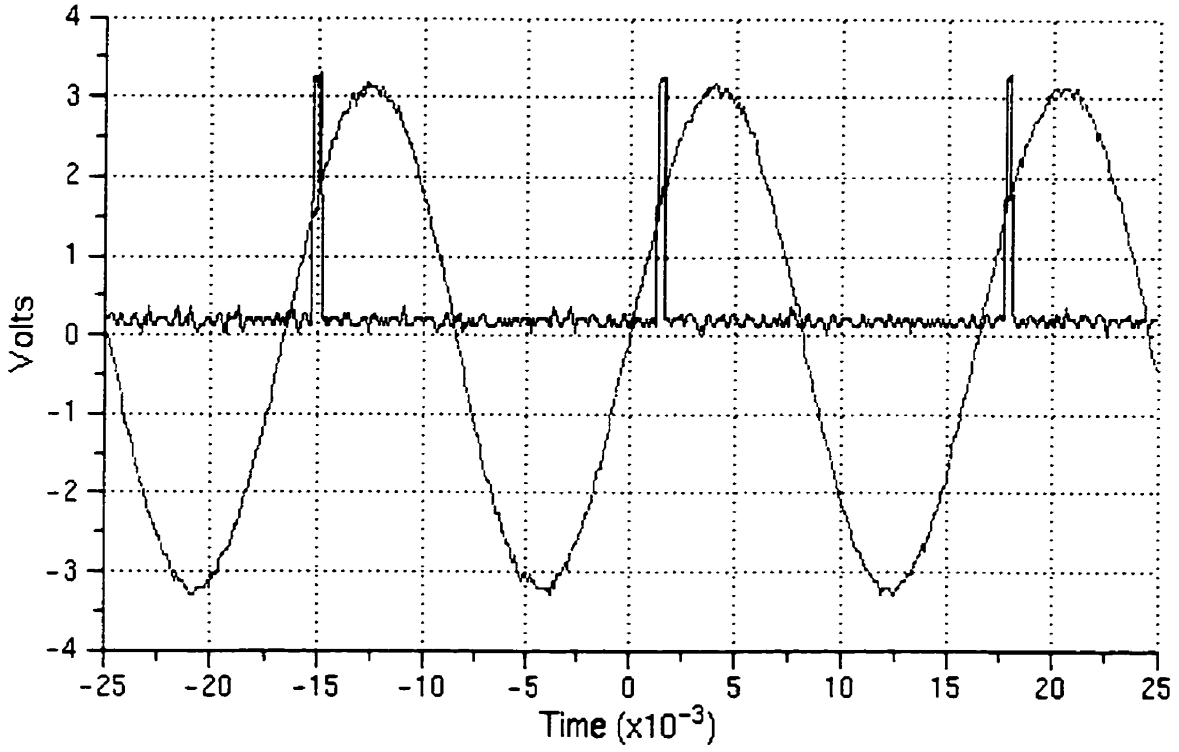


Figure 4.10

Test 6b: Testing PSCAD and RTDS using simple HVDC network

In this test, a feedback loop was tested using the PSCAD and RTCS interface in the HVDC analog simulator. A schematic diagram was drawn in the PSCAD/DRAFT with a current feedback control loop for the HVDC rectifier. A slider placed in the DRAFT to vary the current reference in the rectifier output (DC current). The whole schematic and the PSCAD/DRAFT are shown in the following Figures 4.11. and 4.12 This exercise tested the system in its most complete form.

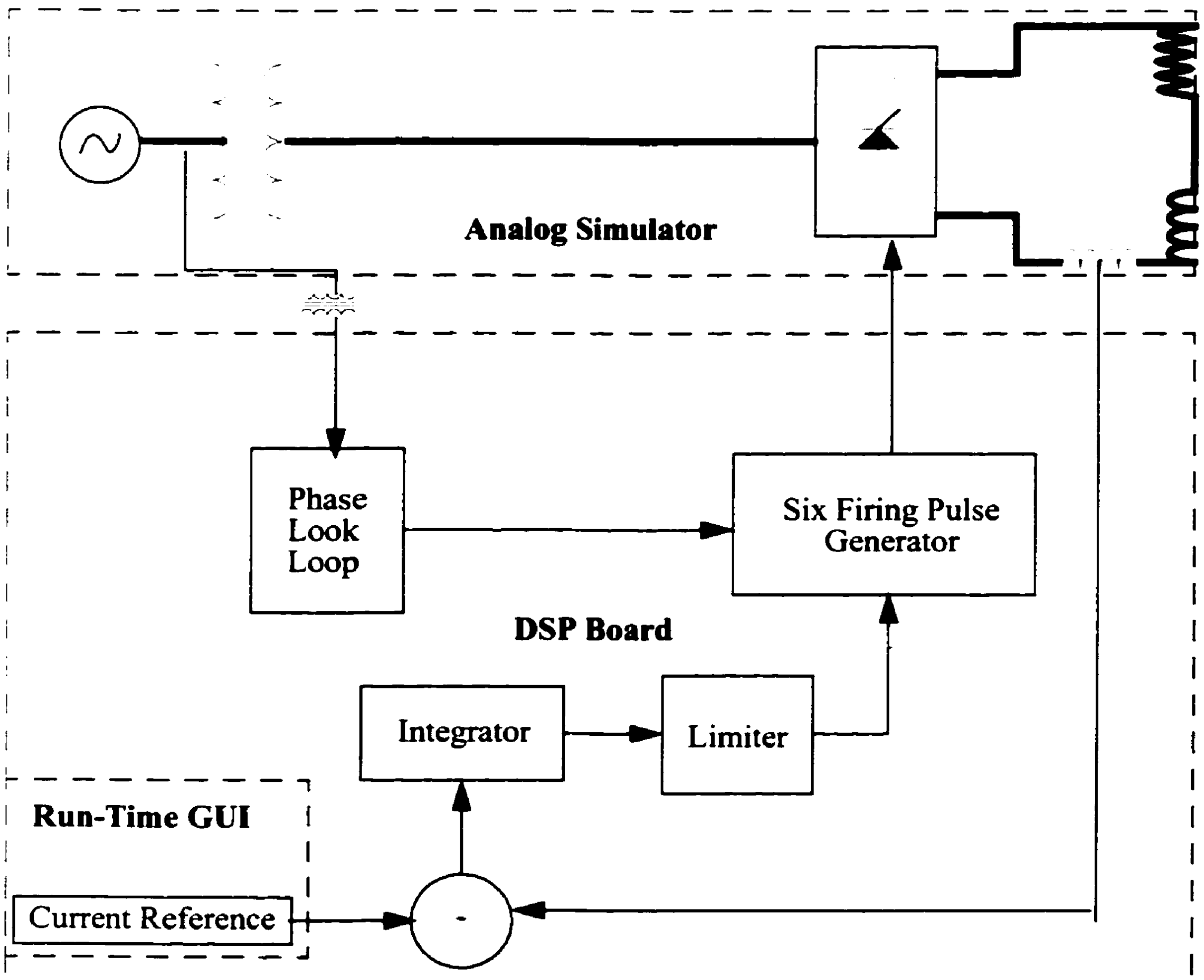


Figure 4.11

In this network, the real time control monitors the output current from the rectifier. The difference between the rectifier output current and the user input is used to adjust the firing angle of the rectifier automatically. The proportional integrator-block ramps the firing slowly for current variation. The limiter-block limits the firing angle between user set upper and lower limits.

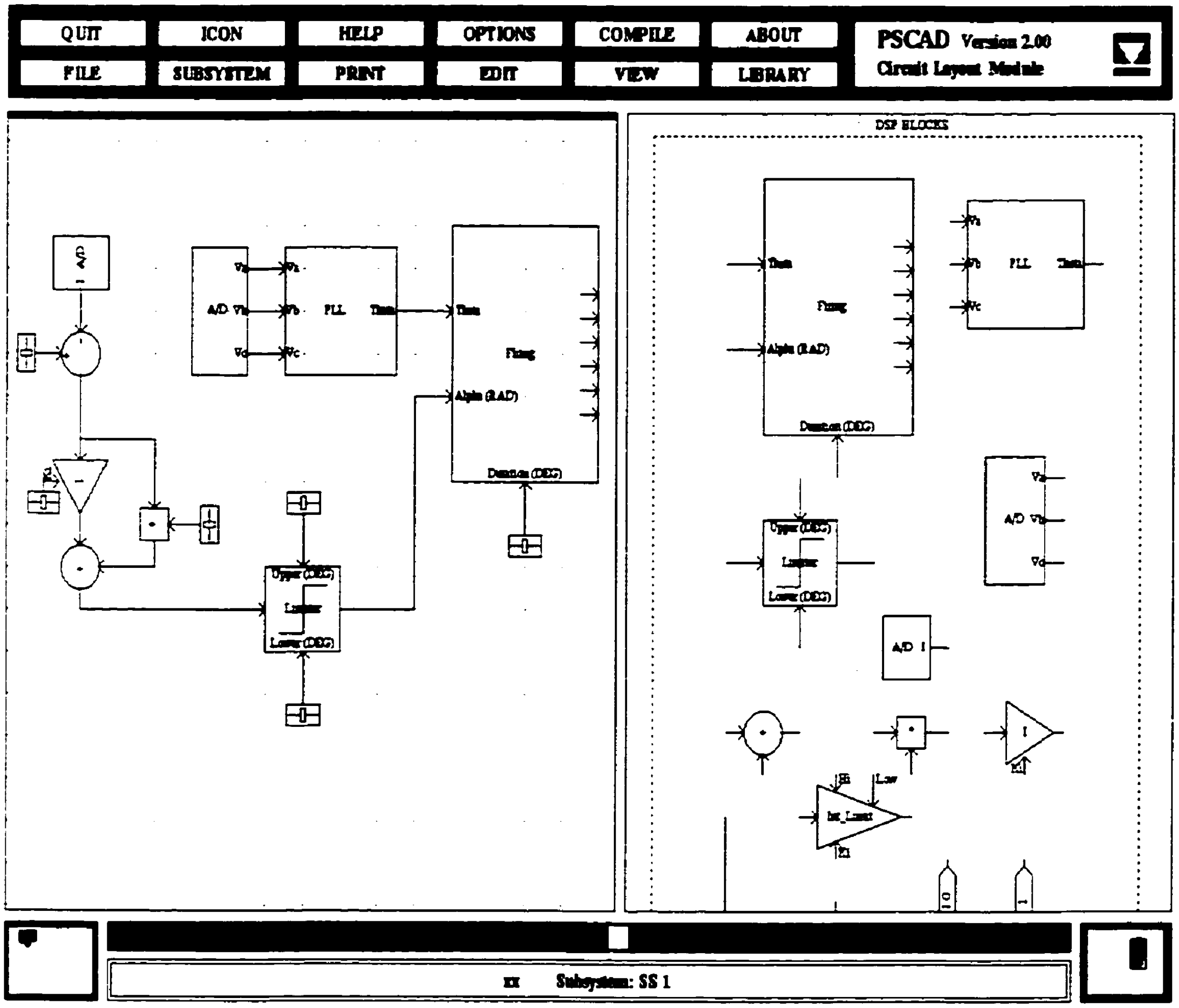


Figure 4.12

The hardware circuit was wired in the HVDC analog simulator according to the block diagram in Figure 4.11. As in test 1, the process from PSCAD/DRAFT to execution of real-time environment were followed.

The run-time GUI placed all six original sliders in its window during initialisation. Each parameter in the real-time DSP program was changed using GUI sliders and the system behaviour was observed. During this test, changing the current reference within its limit in the GUI, gradually changed the real current output from the rectifier. This indicates that the whole control system in the DSP board worked correctly in real-time.

CHAPTER FIVE

CONCLUSIONS

This thesis has shown that a real time control system for power system simulation study can be developed using off-the shelf components such as a Digital Signal Processing Board with the addition of the popular PSCAD tool. In this environment most of the control systems for HVDC topology can be implemented through real time simulation. Furthermore, parameters involved in the HVDC control system can also be varied within their limits, during real time simulation.

Features in the PSCAD allow us to develop a component library with DSP C-code and parameter initialization. They also allow us to create a C programming language based executable DSP-code and other information files necessary for the real time simulation. Users also can create their own component blocks to add in the component library.

Today DSPs are very powerful and allow extremely fast number crunching. This permits easy implementation of fully digital controllers. The use of the off-the shelf simple DSP boards proved their capability to handle complex algorithms in real time, which are involved in the HVDC control system.

The following is a brief summary of the RTCS:

- The popular GUI of the PSCAD has been exploited to make a programmable Real Time Control System (RTCS).
- A off-the shelf Digital Signal Processing Board was used to implement the design.
- Typical blocks required for a power electronic system were identified and implemented.
- A translator program required to convert DSDYN file from the PSCAD to DSP C-code was developed.
- PC based GUI developed to run the RTCS in real-time.

5.1 Recommendations for Future Work

In order to utilize the concept developed in this thesis, the following recommendations need to be adapted.

•A) A multiprocessing DSP board can be used to increase real-time processing power. It will facilitate the user with more simulation power for complicated, processor intensive control systems. Even if the multi-DSP board is found to be uneconomical at the present time, dramatically lowering semiconductor costs and progress in technology could make it viable soon.

•B) VME or VXI bus based PC and the multi-DSP boards combination can be used to increase data bandwidth between graphical user environment in the PC and the real time signal processing. The designer can increase the capability of the GUI to view some necessary HVDC signals in real time. Furthermore, VME or VXI based Analog to Digital converter cards also can be used and are readily available.

•C) Freely available operating systems such as Linux can be used for PSCAD/DRAFT and run-time environment. These operating systems allow the developer to have more control over the Kemal, since their source code is also freely available and can be modified to maximize the performance.

References

[1] A M Gole, V K Sood and L Mootoosamy, "Validation and Analysis of a Grid Control System Using d-q-z Transformation for Static Compensator Systems" Canadian Conference on Electrical and Computer Engineering, September 1989, pp 745-748.

[2] Third-Generation TMS320 User's Guide, Texas Instrument Incorporated.

[3] A M Gole, "HVDC Transmission Course 24.799" Notes, University of Manitoba 1995

[4] EMTDC User Manual - Manitoba HVDC Research Centre, Winnipeg, Manitoba

[5] J D Ainsworth, "The Phase Locked Oscillator - A New Control System for Controlled Static Converters", IEEE Trans. on PAS-87, No.3 March 1968, pp 859-864.

[6] Gunnar Asplund, Lennart Carlsson and Alf Persson "New concepts in HVDC" ABB Power Systems AB (Sweden)

[7] John Reeve, John A Baron and G A Hanley "A Technical assessment of artificial commutation of HVDC converters with series capacitors" IEEE Transactions on power apparatus and systems, Vol. pas-87, No 10, October 1968.

[8] Samuel P Harbison and Guy L Steele Jr, C Reference Manual, Prentice Hall, Englewood Cliffs, New Jersey.

[9] Microsoft Visual C++ Development system for windows, Class Library Refence, Microsoft Corporation.

[10] Microsoft Visual C++ Development system for windows, Class Library Refence, Visual work bench users guide, Microsoft Corporation.

Appendix A

```
/*****************************************************************************************/
/*This program tranlate the DSDYN file into a compilable DSP program to run in the DSP board.    */
/*****************************************************************************************/
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>

struct entry {  char *lexptr; };
/* Keywords used in the DSDYN file */
struct entry keywords[] = {"SUBROUTINE", "INCLUDE", "COMMON", "REAL", "DATA", "INTE-
GER", "VAR", "END", "RETURN"};
struct entry keys[] = {"DD", "FF", "MM"};

/*****************************************************************************************/
/*This function read the file in to a charecter buffer                                  */
/*****************************************************************************************/

int read_file(FILE *infile, char *buffer)
{
char temp, *temp1;
int flag;
int i = 0;
temp1 = buffer;
while(!feof(infile))
  {
  temp = getc(infile);
  if(!feof(infile))
    {
    temp1[i] = temp;
    i++;
    }
  }
 return i;
}

/*****************************************************************************************/
/*This function concatinat a string into a designated buffer.                           */
/*****************************************************************************************/

void copy_str(char *buffer, char *str, int last)
{
for(int i = 0; i < strlen(str); i++)
 {
 buffer[last+i] = str[i];
 }
}
```

```
/***************************************************************************************/
/*This function extract Slider information from the DSDYN file and write it into a separate file.     */
/***************************************************************************************/
void slider_info(char *buffer_in, int *size, char *buffer_info, int *size_info)
{
int t;
char str_buffer[15];

(*size)++;

do
 {
  t = buffer_in[(*size)];
  if(isalpha(t))
  {
   int a = 0;
   while (isalpha(buffer_in[(*size)]))
    {
     str_buffer[a] = buffer_in[(*size)];
     a++;
     (*size)++;
    }
    str_buffer[a] = '\0';
    if(strcmp(str_buffer, "Slider") == 0)
     {
     do
       {

       if ((buffer_in[(*size)]) == '=')
          {
          (*size)++;
          do
           {
            if(buffer_in[(*size)] != '"')
              {
              buffer_info[(*size_info)] = buffer_in[(*size)];
              (*size_info)++;
              }
            (*size)++;
           if (buffer_in[(*size)] == '\n')
             break;
           }while (buffer_in[(*size)] != ' ') ;
           buffer_info[(*size_info)] = ' ';
           (*size_info)++;
          }
       else
         {
         (*size)++;
         }
      }while  (buffer_in[(*size)] != '\n');
     buffer_info[(*size_info)] = '\n';
    (*size_info)++;
    return;
    }
   }
  else
    (*size)++;
 } while (buffer_in[(*size)] != '\n');
return;
}
/******************************************************************************************/
/*This function remove cooments from the file and return new file size                    */
/******************************************************************************************/
int comment(char *buffer_in, int size, char *buffer_out, FILE *file_info)
{
char *temp1, *temp2;
temp1 = buffer_in;
temp2 = buffer_out;
int j = 0, t, new_size = 0, temp;
char str_buffer[15];
char buffer_info[1024];
int size_info = 0;
for(int i =0; i <= size;i++ )
 {
 t = buffer_in[i];
 if (t == '\n' || t == '\t' || t == ' ')
  {
  buffer_out[new_size] = t;
  new_size++;
  }
 else if (!isalnum(t))
  {
  buffer_out[new_size] = t;
  new_size++;
  }

 else if (isalnum(t))
  {
  int a = 0;
   while (isalnum(t))
    {
    str_buffer[a] = t;
    a++;
    i++;
    t = buffer_in[i];
    }
   i--;
   str_buffer[a] = '\0';
   temp = new_size;
   if(strcmp(str_buffer, "C") == 0)
    {
    slider_info(buffer_in, &i, buffer_info, &size_info);
    }
   else
    {
    copy_str(buffer_out, str_buffer, new_size);
    new_size =  new_size + strlen(str_buffer);
```

```
      }
     }
    }
   for(int k=0; k < size_info; k++)
    {
     putc(buffer_info[k], file_info);
     printf("%c", buffer_info[k]);
    }
new_size--;
return new_size;
}
//*********************************************************************************************/
/*This function   lookup for keywords and return its size  */
/*********************************************************************************************/
int lookup(char s[],struct entry *keywords, int length)
{
int temp_return = 0;
for(int p = 0; p < length; p++)
  {
   if(strcmp(keywords[p].lexptr, s) == 0)
     temp_return = p+1;
  }
 return temp_return;
}

*********************************************************************************************/
/*This function copy a whole line from on buffer to another buffer.                        */
/*********************************************************************************************/
int copy_lines(char *buffer_in, int  *size_in, char *buffer_out, int size_out, int key_flag)
{
int t, flag;
char str_buffer[256];

  for(int j=0;j<(*size_in);j++)
       {
       t = buffer_in[j];
       if (isspace(t) == 0)
         {
         int a = 0;
         while (isalnum(t))
          {
         str_buffer[a] = t;
         a++;
         j++;
         t = buffer_in[j];
          }
         str_buffer[a] = '\0';
        flag = lookup(str_buffer, keys,3);
        if (flag == key_flag)
        {
         if (key_flag == 0)
           j = j - strlen(str_buffer);
          do
           {
            buffer_out[size_out] = buffer_in[j];
            size_out++;
            j++;
           }while (buffer_in[j-1] != '\n');
           j--;
          }

        else
          {
          do
           {
           if (j >= (*size_in))
             return size_out;
           j++;
           }while (buffer_in[j] != '\n');

          }

        }
       }
return size_out;/
*****************************************************************************************/
/*This function  extract C code from one file to another  */
/*****************************************************************************************/
}

int c_code(char *buffer_in, char *buffer_out, int size)
{
/*char *temp1, *temp2;
temp1 = buffer_in;
temp2 = buffer_out;*/
int new_size = 0;
int t, temp, flag, real_flag = 0, newline_size = 0;
char str_buffer[256];

for(int i =0; i <= size;i++ )
 {
 t = buffer_in[i];
 if (t == '\n' || t == '\t' || t == ' ')
  {
  buffer_out[new_size] = t;
  new_size++;
  if (t == '\n')
   {
   if(isdigit(buffer_in[i-1]) != 0)
    {
    buffer_out[new_size - 1] = ';';
    buffer_out[new_size] = '\n';
    new_size++;
    }
   newline_size = new_size;
   }
  }
 else if (!isalnum(t))
```

```
  {
  buffer_out[new_size] = t;
  new_size++;
  }

else if (isalnum(t))
 {
 int a = 0;
  while (isalnum(t))
    {
    str_buffer[a] = t;
    a++;
    i++;
    t = buffer_in[i];
    }
  i--;
  str_buffer[a] = '\0';
  temp = new_size;
  /*printf("%d \n", new_size);*/
  flag = lookup(str_buffer,keywords,9);
   if(flag == 4)
    real_flag++;
  if(flag > 0)
   {

     if((flag == 4) && (real_flag > 2))
      {
      copy_str(buffer_out, "DD  float", new_size);
      new_size =  new_size + strlen("DD  float");
      do
        {
        i++;
        buffer_out[new_size] = buffer_in[i];
        new_size++;
        } while (buffer_in[i] != '\n');
       new_size--;
       copy_str(buffer_out, " ;\n", new_size);
       new_size =  new_size + strlen(" ;\n");
       newline_size = new_size;
      }
    else if ((flag == 7) && (buffer_in[i+1] == '('))
      {
      copy_str(buffer_out, str_buffer, new_size);
      new_size =  new_size + strlen(str_buffer);
      buffer_out[new_size] = '['; new_size++;i++;
      do
        {
         i++;
         if(buffer_in[i] == ')')
            buffer_out[new_size]=']';
         else
           buffer_out[new_size]=buffer_in[i];
         new_size++;
        } while (buffer_in[i] != '\n');
```

```
         new_size--;
         copy_str(buffer_out, ";\n", new_size);
         new_size =  new_size + strlen(";\n");
         newline_size = new_size:

        }
       else
        {
         do
          {
          i++;
          } while (buffer_in[i] != '\n');
         /*buffer_out[new_size] = '\n';*/
        new_size= newline_size;

        }
      }
     else if (flag == 0)
      {
       copy_str(buffer_out, str_buffer, new_size);
       new_size =  new_size + strlen(str_buffer);
       newline_size = new_size;
       }
      /*printf("%d \n", new_size);*/

   }
  }
new_size--;
buffer_out[new_size] = '\n';
return new_size;
}/*******************************************************************************************/
/*This function rearrange the C code into combilable code                                  */
/*******************************************************************************************/

/* 1-->main, 2-->xx.c */
int main_code(char *buffer_in1, char *buffer_in2, char *buffer_out, int size1, int size2)
{
int new_size = 0;
int t, t1, temp, flag, flag1, newline_size = 0;
char str_buffer[256];

for(int i =0; i <= size1;i++ )
 {
 t = buffer_in1[i];
 if (t == '\n' || t == '\t' || t == ' ')
  {
   buffer_out[new_size] = t;
   new_size++;
  }
 else if (!isalnum(t))
  {
   buffer_out[new_size] = t;
   new_size++;
   }
  else if (isalnum(t))
   {
   int a = 0;
    while (isalnum(t))
      {
      str_buffer[a] = t;
      a++;
      i++;
      t = buffer_in1[i];
      }
    i--;
    str_buffer[a] = '\0';
    temp = new_size;
    flag = lookup(str_buffer, keys,3);

    switch(flag)
    {
    case 1:
     {
      new_size = copy_lines(buffer_in2, &size2, buffer_out, new_size, 1);
      break;
     }

    case 2:
     {
      new_size = copy_lines(buffer_in2, &size2, buffer_out, new_size, 0);
      break;
     }
    case 3:
     {
      new_size = copy_lines(buffer_in2, &size2, buffer_out, new_size, 3);
      break;
     }
    case 0:
     {
      copy_str(buffer_out, str_buffer, new_size);
      new_size =  new_size + strlen(str_buffer);
      newline_size = new_size;
      break;
      }
    }
  }
 }

return new_size;
}

main()
{

FILE *infile, *outfile, *mainfile, *infofile;
char in_buffer[8192], *temp;
char in_buffer1[8192], in_buffer2[8192];
```

```
int file_size, file_size2, final_size;
char str_system[15], str_outfile[15], str_infile[15], str_infofile[15];

/*infile = fopen("xx.dsd.f","r");
outfile = fopen("xx.c","w");
mainfile = fopen("main.c", "r");
infofile = fopen("xx.inf", "w");*/

scanf("%s", str_system);
strcpy(str_infile, str_system);
strcat(str_infile, ".dsd.f");
strcpy(str_outfile, str_system);
strcat(str_outfile, ".c");
strcpy(str_infofile, str_system);
strcat(str_infofile, ".inf");
printf("%s %s %s \n",str_infile, str_outfile,  str_infofile);
infile = fopen(str_infile,"r");
outfile = fopen(str_outfile,"w");
mainfile = fopen("main.c", "r");
infofile = fopen(str_infofile, "w");


file_size = read_file(infile, in_buffer);
file_size2 = read_file(mainfile, in_buffer2);
fclose(infile);
fclose(mainfile);

file_size = comment(in_buffer, file_size, in_buffer1, infofile); /*buffer ---> buffer1 */
printf("%d \n", file_size);
file_size = c_code(in_buffer1, in_buffer, file_size); /*buffer1 ---> buffer */
printf("%d \n", file_size);
final_size = main_code(in_buffer2, in_buffer, in_buffer1, file_size2, file_size); /*buffer --->buffer1 */
for(int i=0; i <final_size; i++)
  {
  putc(in_buffer1[i],outfile);
  }

fclose(infofile);
fclose(outfile);
return 0;
}
```

Appendix B.1

```
PARAMETERS:
GRAPHICS:
        Box(-48,-48,48, 48)
        Line( -64, -32, -48, -32)  Arrow_R(-48, -32)  FText( -42, -32,"Va")
        Line( -64, 0, -48, 0)  Arrow_R(-48, 0)  FText( -42, 0,"Vb")
        Line( -64, 32, -48, 32)  Arrow_R(-48, 32)  FText( -42, 32,"Vc")
        Line( 48, 0, 64, 0)  FText( 36, 0, "Theta")
        FText(-5, 0, "PLL")
NODES:
        Va -2 -1 INPUT  REAL
        Vb -2  0 INPUT  REAL
        Vc -2  1 INPUT  REAL
        theta 2  0 OUTPUT  REAL
FORTRAN: DSD
        C ========================
        C  Phase Lock Loop
        C ========================
        DD  float Vac, Vba, Vcb, Valpha, Vbeta, Prev_Theta, Pres_Theta, delta_t;
        DD  float Prev_Err, Pres_Err, Prev_Err_s;
        DD  float Pres_Err_s, Prev_VCO_input, Pres_VCO_input;
        DD  float K1, K2, Uref, delta_Uref;
        /*********pll*******************/
            Vac = $Va;
            Vba = $Vb;
            Vcb = $Vc;
            Valpha = 0.33333*(2.0*Vac - Vba - Vcb);
            Vbeta = 0.57735*(Vba - Vcb);
            Pres_Err = Valpha*cos(Prev_Theta) + Vbeta*sin(Prev_Theta);
            Pres_Err_s = (Prev_Err + Pres_Err)*0.5*delta_t + Prev_Err_s;
            Pres_VCO_input = K1*Pres_Err_s + K2*Pres_Err + Uref + delta_Uref;
            Pres_Theta = Pres_VCO_input*delta_t*6.28318 + Prev_Theta;
            if ( Pres_Theta > 6.28318)
                Pres_Theta = Pres_Theta - 6.28318;
            Prev_Err = Pres_Err;
            Prev_Err_s = Pres_Err_s;
            Prev_VCO_input = Pres_VCO_input;
            Prev_Theta = Pres_Theta;
            $theta = Pres_Theta;
        /********endof pll********************/
        MM  Prev_Err = 0.0;
        MM  Prev_Err_s = 0.0;
        MM  Prev_VCO_input = 0.0;
        MM  Prev_Theta = 0.0;
        MM  K1 = 20.0;
        MM  K2 = 10.0;
        MM  Uref = 60.0;
        MM  delta_Uref = 0.0;
        C ==================================
```

Appendix B.2

```
PARAMETERS:
GRAPHICS:
        Box(-64,-96,64, 96)
        Line( -96, -32, -64, -32)  Arrow_R(-64, -32)  FText( -50, -32,"Theta")
        Line( -96, 32, -64, 32)  Arrow_R(-64, 32)  FText( -34, 32,"Alpha (RAD)")
        Line( 0, 128, 0, 96) Arrow_U(0, 96) FText(0, 90, "Duration (DEG)")
        FText(-5, 0, "Firing")
        Line( 48, -45, 64, -45)   Arrow_R(64, -45)
        Line( 48, -27, 64, -27)   Arrow_R(64, -27)
        Line( 48, -9, 64, -9)     Arrow_R(64, -9)
        Line( 48, 9, 64, 9)       Arrow_R(64, 9)
        Line( 48, 27, 64, 27)     Arrow_R(64, 27)
        Line( 48, 45, 64, 45)     Arrow_R(64, 45)
NODES:
        theta -3 -1 INPUT    REAL
        alpha -3  1 INPUT    REAL
        duration 0 4 INPUT   REAL
FORTRAN: DSD
        C  Ramp generator and firing
        C  ========================
        DD  float Ramp[6], start, stop;
        /**********firing block*************************/
          Ramp[0] = $Theta;
          for (j = 1; j <= 5; j++)
            {
            Ramp[j] = Ramp[j-1] - 1.04719755;  /*  PI/3  */
            if (Ramp[j] < 0.0)
            Ramp[j] = Ramp[j] + 6.2831853;
            }
           start = $alpha;
           stop = start + $duration*(3.14159/180.0);
           /* pulse 1 at CLKX0 */
           if ((Ramp[0] > start) & (Ramp[0] < stop))
             up_CLKX0;
           else
             dn_CLKX0;
           /* pulse 2 at FSX0 */
           if ((Ramp[1] > start) & (Ramp[1] < stop))
             up_FSX0;
           else
            dn_FSX0;
           /* pulse 3 at DX0 */
           if ((Ramp[2] > start) & (Ramp[2] < stop))
             up_DX0;
           else
            dn_DX0;
           /* pulse 4 at DX1 */
           if ((Ramp[3] > start) & (Ramp[3] < stop))
             up_DX1;
           else
             dn_DX1;
           /* pulse 5 at FSX1 */
```

```
    if ((Ramp[4] > start) & (Ramp[4] < stop))
        up_FSX1;
    else
        dn_FSX1:
    /* pulse 6 at CLKX1 */
    if ((Ramp[5] > start) & (Ramp[5] < stop))
        up_CLKX1;
    else
       dn_CLKX1;
/***********endof firing block************************/
C ============================
```