FOUR QUADRANT COMPUTER BASED MOTOR TEST SYSTEM DEVELOPMENT

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

TOLGA İNAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2003

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all requirements as a thesis for the degree of Master of Science.

Prof. Dr. Mübeccel DEMİREKLER
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. H. Bülent ERTAN
Supervisor

Examining Committee Members

Prof. Dr. Muammer ERMİŞ (chairman)

Prof. Dr. H. Bülent ERTAN

Assoc. Prof. Dr. Işık ÇADIRCI

Asst. Prof. Dr. Cüneyt BAZLAMAÇCI

M. Sc. Tolga ÇAMLIKAYA

ABSTRACT

FOUR QUADRANT COMPUTER BASED MOTOR TEST SYSTEM DEVELOPMENT

İNAN, Tolga

M.Sc., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. H. Bülent ERTAN

August 2003, 138 pages

Development and research activities about electric motors require realistic feedback about the motor performance and efficiency. This feedback can be supplied by the help of the motor test systems without waiting for the end-user.

Throughout this study, a computer based motor test system with four quadrant loading capability is developed. The system is capable of entering user-defined test conditions, performing tests, acquisition of test data and displaying test results. The system has a visual user interface that can handle all tasks from a single computer.

Keywords: Motor test system, controlled load, data acquisition

ÖZ

DÖRT BÖLGELİ BİLGİSAYAR TEMELLİ MOTOR TEST SİSTEMİ GELİŞTİRİLMESİ

İNAN, Tolga

Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü

Tez Danışmanı: Prof. Dr. H. Bülent ERTAN

Ağustos 2003, 138 sayfa

Elektrik motorları hakkındaki araştırma ve geliştirme çalışmaları motor performansı ve verimi hakkında sağlıklı bir geribeslemeye ihtiyaç duyar. Bu geribesleme, son kullanıcı beklenmeksizin motor test sistemleri ile sağlanabilir.

Bu çalışma boyunca, dört bölgeli yükleme yeteneği olan bilgisayar temelli bir motor test sistemi geliştirilmiştir. Sistem, kullanıcı tarafından tanımlanan test koşullarının girilmesi, testlerin gerçekleştirilmesi, verilerin toplanması ve toplanan verilerin gösterilmesi yeteneklerine sahiptir. Sistemin, tüm işleri tek elden yürüten görsel bir arayüzü mevcuttur.

Anahtar Kelimeler: Motor test sistemi, kontrol edilebilir yük, veri toplama

ACKNOWLEDGEMENTS

I express sincere appreciation to my thesis supervisor Prof. Dr. H. Bülent Ertan for his valuable guidance throughout all stages of this study, which made it possible.

I wish to thank Prof. Dr. Muammer Ermiş for his continuous encouragement.

I also wish to thank Özgür Güven for his support and guidance at the beginning of this study.

I thank my roommates Bülent Dağ, Ertan Murat, Cüneyt Karacan , Serkan Paki Şedele and Orhan Seyfi Tosun for their deep understanding and support.

Finally, I would like to thank mom and dad, my sister and my dear wife for every step we had walked together.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Today, electrical machines are used on a large scale for the systems requiring mechanical work. There had been continuous research and development activities to produce electrical machines with higher performance and efficiency since energy-efficient electrical machines represent one of the largest opportunities for cost-effective electric savings around the world. Motor test systems are very critical at this point, because a motor test system with sufficient testing capabilities supply important information about motor design's success [1].

Electrical machine is a system that transforms electrical power to mechanical power in either direction. Therefore, a satisfactory motor test system requires mechanical measurements (torque, speed) as well as electrical ones. Properties of the electrical and mechanical transducers chosen for these measurements depend on the following issues: The type of the tested machine, maximum electrical and mechanical limits of the tested machine, parameters to be measured and the acceptable limits of error margins [9]. In addition to sensors, a motor test system requires a load for the tested machine. Passive loads (brakes etc.) were employed for this purpose in the older test beds. However today, with recent advances in the power electronic area, active loads are commonly used in such systems. Active loads equipped with static power controllers can be used as the master drive in the test bed, and they can easily control the operation speed. Therefore, the performance of the tested machine can be observed at any desired speed. As a result, a successful motor test bed should include a properly chosen sensor set, controlled load and suitable data acquisition hardware.

There are commercially available motor test systems with different testing capabilities in the world market [5,8]. These test beds have some common properties. They all employ direct measurements of shaft torque and angular speed. This corresponds to a realistic measurement of mechanical output power. Another common property among these test systems is the presence of a visual user-interface. This user-interface is used for controlling the load, acquisition of motor parameters and displaying the results.

This study stems from the need of such a motor test system in our country. It involves development of a computer based four-quadrant motor test system. This motor test system aims to control the load of the test machine dynamically and simultaneously measure the electrical and mechanical parameters of the test machine. This gives the system the ability of setting user-defined loads and controlling the operating conditions of the test machine.

This system can also be used as load simulator. If the machine under testis used generally with a load, which has known characteristic, motor test system can be programmed to simulate this load characteristic. Then the test of the machine is performed in conditions, which are very close to the actual operating conditions of the test machine.

Motor test system, which is capable of changing the load of test machine dynamically, can be used in testing vector controlled drives. Electrical machine drives using vector control algorithms are able to respond sudden changes in the load in a few milliseconds. Performance of such kind of drives cannot be measured by employing constant load tests. A machine test system which can suddenly change the machine load and record corresponding response of the tested electrical machine drive is required for testing vector controlled drives. The motor test system developed during this study can be used for testing vector controlled drives.

In the second chapter, an overview of the motor test system and introductory information about the system hardware will be given. System hardware consists of a load machine, a load controller, sensors, a data acquisition card and a personal computer. In order to create a user-friendly test environment, a visual user-interface is designed. This designed software handles all the tasks of test setup such as entering test conditions, performing tests and displaying results. This software will be introduced in the third chapter. Information about system operation is going to be given in the fourth chapter, which also includes results of sample tests performed by the motor test system.

CHAPTER 2

OVERVIEW & HARDWARE

2.1 General Description

The system constructed during the thesis is a dynamic motor performance tester. The system can change load of the tested motor dynamically and simultaneously measure the torque on the motor shaft, speed of rotation and electrical parameters of the motor.

The system hardware consists of the following parts:

i. Controlled load

ii. 4-Quadrant drive as load controller

iii. Sensors and signal conditioning equipment

iv. Data acquisition card

v. Computer for controlling the system

These parts will be presented in the following sections in detail. Here a basic idea about the operation principles of motor test system is given.

Motor test system requires a load for the test machine. This study aims to implement a load, which is user programmable. Controlled load and load controller combination is used to give the system ability of controlling load.

Programming the load is the first step for performing motor tests. Second step is running this user-defined load and acquiring of electrical and mechanical parameters of the test machine. Electrical parameters stand for voltage, current and power of the test machine. Mechanical parameters, on the other hand, stand for shaft speed and torque of the test machine. Sensors, signal conditioning equipment and data acquisition card are used at this stage. Finally, a personal computer is required for controlling the system hardware and establishing coordination during the tests.

2.2 Implementation Options

This section aims to present different implementation options for the system hardware. Discussion is focused on two points. The first point is options for load machine. AC and DC motor types and their advantages / disadvantages will be reminded. Second point is demonstrating basics of electrical measurements and possible measuring techniques. At the end of this section, implemented options will be presented with brief explanations.

2.2.1 AC and DC Motors

Motor Types

Industrial motors come in a variety of basic types. These variations are suitable for many different applications [12]. Naturally, some types of motors are more suited for certain applications than other motor types are.

2.2.1.1 AC MOTORS

The most common and simple industrial motor is the three-phase AC induction motor, sometimes known as the "squirrel cage" motor.

ADVANTAGES

- Simple Design
- Low Cost
- Reliable Operation
- Easily Found Replacements

Simple Design

The simple design of the AC motor -- simply a series of three windings in the exterior (stator) section with a simple rotating section (rotor). The changing field caused by the 50 or 60 Hertz AC line voltage causes the rotor to rotate around the axis of the motor.

The speed of the AC motor depends only on three variables:

1. The fixed number of winding sets (known as poles) built into the motor, which determines the motor's base speed.
2. The frequency of the AC line voltage. Variable speed drives change this frequency to change the speed of the motor.
3. The amount of torque loading on the motor, which causes slip.

Low Cost

The AC motor has the advantage of being the lowest cost motor for applications requiring more than about 1/2 hp (325 watts) of power. This is due to the simple design of the motor. For this reason, AC motors are overwhelmingly preferred for fixed speed applications in industrial applications and for commercial and domestic applications where AC line power can be easily attached. Over 90% of all motors are AC induction motors. They are found in air conditioners, washers, dryers, industrial machinery, fans, blowers, vacuum cleaners, and many, many other applications.

Reliable Operation

The simple design of the AC motor results in extremely reliable, low maintenance operation. Unlike the DC motor, there are no brushes to replace. If run in the appropriate environment for its enclosure, the AC motor can expect to need new bearings after several years of operation. If the application is well designed, an AC motor may not need new bearings for more than a decade.

Easily Found Replacements

The wide use of the AC motor has resulted in easily found replacements. Many manufacturers adhere to either European (metric) or American (NEMA) standards. (For Replacement Motors)

DISADVANTAGES

AC Motors have the following disadvantages:

- Expensive speed control

- Inability to operate at low speeds
- Poor positioning control

Expensive speed control

Speed control is expensive. The electronics required to handle an AC inverter drive are considerably more expensive than those required to handle a DC motor. However, if performance requirements can be met -- meaning that the required speed range is over 1/3rd of base speed -- AC inverters and AC motors are usually more cost-effective than DC motors and DC drives for applications larger than about 10 horsepower, because of cost savings in the AC motor.

Inability to operate at low speeds

Standard AC motors should not be operated at speeds less than about 1/3rd of base speed. This is due to thermal considerations. A DC motor should be considered for these applications.

Poor positioning control

Positioning control is expensive and crude. Even a vector drive is very crude when controlling a standard AC motor. Servomotors are more appropriate for these applications.

2.2.1.2 DC MOTORS

The brushed DC motor is one of the earliest motor designs. Today, it is the motor of choice in the majority of variable speed and torque control applications.

ADVANTAGES

- Easy to understand design
- Easy to control speed
- Easy to control torque
- Simple, cheap drive design

Easy to understand design

The design of the brushed DC motor is quite simple. A permanent magnetic field is created in the stator by either of two means:

- Permanent magnets
- Electro-magnetic windings

If the field is created by permanent magnets, the motor is said to be a "permanent magnet DC motor" (PMDC). If created by electromagnetic windings, the motor is often said to be a "shunt wound DC motor" (SWDC). Today, because of cost-effectiveness and reliability, the PMDC motor is the motor of choice for applications involving fractional horsepower DC motors, as well as most applications up to about three horsepower.

At five horsepower and greater, various forms of the shunt wound DC motor are most commonly used. This is because the electromagnetic windings are more cost effective than permanent magnets in this power range.

The section of the rotor where the electricity enters the rotor windings is called the commutator. The electricity is carried between the rotor and the stator by conductive graphite-copper brushes (mounted on the rotor), which contact rings on stator.

In most DC motors, several sets of windings or permanent magnets are present to smooth out the motion.

Easy to control speed

Controlling the speed of a brushed DC motor is simple. The higher the armature voltage, the faster the rotation. This relationship is linear to the motor's maximum speed.

The maximum armature voltage which corresponds to a motor's rated speed (these motors are usually given a rated speed and a maximum speed, such as 1750/2000 rpm) are available in certain standard voltages, which roughly increase in conjunction with horsepower. Thus, the smallest industrial motors are rated 90 V DC and 180 V DC. Larger units are rated at 250 V DC and sometimes higher.

Specialty motors for use in mobile applications are rated 12, 24, or 48 VDC. Other tiny motors may be rated 5 VDC.

Most industrial DC motors will operate reliably over a speed range of about 20:1 -- down to about 5-7% of base speed. This is much better performance than the comparable AC motor. This is partly due to the simplicity of control, but is also partly due to the fact that most industrial DC motors are designed with variable speed operation in mind, and have added heat dissipation features which allow lower operating speeds.

Easy to control torque

In a brushed DC motor, torque control is also simple, since output torque is proportional to current. If you limit the current, you have just limited the torque, which the motor can achieve. This makes this motor ideal for delicate applications such as textile manufacturing.

Simple, cheap drive design

The result of this design is that variable speed or variable torque electronics are easy to design and manufacture. Varying the speed of a brushed DC motor requires little more than a large enough potentiometer. In practice, these have been replaced for all but sub-fractional horsepower applications by the SCR and PWM drives, which offer relatively precisely control voltage and current.

Large DC drives are available up to hundreds of horsepower. However, over about 10 horsepower careful consideration should be given to the price/performance tradeoffs with AC inverter systems, since the AC systems show a price advantage in the larger systems. (But they may not be capable of the application's performance requirements).

DISADVANTAGES

DC Motors have the following disadvantages:

- Expensive to produce
- Can't reliably control at lowest speeds
- Physically larger
- High maintenance
- Dust

2.2.2 Measurement Options

2.2.2.1 Voltage measurements

The unit of measure for voltage is volt (V), its notation is U, u, V, v, *E* or e. Voltage is the most common electrical magnitude to be measured[11]. It is by far the most comfortable magnitude to measure and therefore all physical magnitudes are tried to be altered into electrical voltage. In contemporary IT-devices almost all information is expressed via electrical voltage. Digitally it is usually expressed by the so-called TTL-levels In analogue version it is usually expressed by any value of voltage.

All engineers must have a clear overview of measuring electrical voltage and respective faults. Both direct and alternating voltage must be measured. Direct voltage is defined as a voltage that does not change in time. Therefore measuring direct voltage is easy even with slow devices for measurement (like devices with mechanically turning pointer). The result of the measurements is the value of direct voltage. We may follow the changes in the value by repeated measurements. Alternating voltage is defined as relatively swiftly changing voltage. Measuring and following the changes of its instantaneous value requires swift devices: swift ADCs that record the values digitally, oscilloscopes for following the voltage curve etc. Measuring alternating current must be thus tackled more thoroughly than that of direct current, as the problem is far more complicated.

Let us first study the different forms of alternating current. It is easier to analyse periodic and aperiodic AC s separately. The so-called pure AC has zero mean or average value .

If the average value of AC is not zero, then it is expressed as a sum of pure AC and DC equaling the value of the mean. It enables us to tackle only these problems that are related to measuring pure AC. It is also important to know the shape of the voltage curve. The most common curve is sinusoidal AC The wide appearance of sinusoidal AC is explained by the fact that in the so-called linear electric circuits all currents and drops caused by some outer sinusoidal effect are always sinusoidal. Besides sinusoidal AC several other types of alternating currents, such as square voltage, triangular voltage, ramp or saw-tooth voltage, periodic rectangular pulses etc are used.

In case of AC we may measure its instantaneous values (by using ADC) and record the measured values digitally. It enables us to restore the curve on the monitor of our PC and study its details. Usually, however, measuring the so-called integral values is the limit. These integral values are:

1) Peak value

2) RMS or effective value

3) Rectified average

4) Average value

Any one of these measuring techniques can be selected for measuring AC voltage levels.

2.2.2.2 **Current measurements**

The unit of measure is ampere (A), its notation is *I* or i. In order to measure current the specific device – ammeter – must be connected to the current circuit in series with other elements in the circuit. Thus the circuit must be broken for a while. Such a breaking is allowed only on laboratory layouts during the elaboration of new circuits. The breaking must be carried out on an unpowered layout, thus avoiding the possible defects in electron devices caused by the voltage redistributions during disconnections of the circuit. After connecting the ammeter with the circuit the supply voltage of the layout is restored.

The resistance of an ideal ammeter is zero. In case of electronic ammeters this precondition is almost fulfilled, though the ammeter might be with an asymmetric input – i.e enables to measure the current only between voluntarily chosen point in circuit and circuit ground. Prepared breaking points for current measurements in industrial devices Typically direct currents are measured in order to estimate the working regime and heating of the elements within the circuits. D'Arsonval meter is successfully used during the process. This device is symmetric and able to measure direct current at any given point. It must be observed that the inner resistivity of the ammeter does not exceed 1-2% of the total resistance of the measuring circuit in order to ensure the low enough impact of the ammeter to the current in the circuit. The information concerning the measuring of integral values of alternating voltages presented is used for measuring the integral values of the alternating current.

2.2.2.3 Power Measurements

The unit of measure for power is watt (W); its notation is P or p; that for reactive power is Q or q. The instantaneous value of the power of electric signal is expressed as the product *of voltage and current.* If the current is in the direction determined by outer voltage, then $0 > p$. Here we have an example of electric energy loss – its temporary loading or transformation into heat, chemical or mechanical energy etc. If $0 < p$ we have the case of producing electric energy or getting it back from an accumulator.

The phase difference between current and voltage results in the case of alternating current in both energy saving and reproducing during a period. Instantaneous values of power are usually of no interest. We are interested in the average value of the power. It interests us in producing and selling electric energy; in electronics it interests us only in estimating the energy loss or heating of the device. In case of electromagnetic waves the radiated power or power falling on a unit of surface must be known.Measurement of power is complicated, as both current and voltage must be measured, their instantaneous values must be multiplied and the product averaged. Analog wattmeters have been elaborated for measuring power at low frequencies or at direct current.

2.2.2.4 Methods Of Torque Measurement

The growing popularity of instrumented couplings for continuous on-line torque monitoring has led to the widely used term torquemeter couplings. There are several varieties of instrumented torquemeter couplings currently available [20] . Each are capable of providing torque measurement through non-contacting means so there is no longer a need for the extra bearing supports associated with a .torquemeter. of years past. These torquemeters physically measure the torque being transmitted between the two machines of which they are connected. Since they also measure the speed, the typical preferred output of these torquemeters is power (torque multiplied by speed).

All torquemeter coupling designs are faced with the task of detecting a physical change in the coupling due to torsion while it is rotating, and getting this information to a stationary output device (generally in the control room). Over the years many methods have been devised to measure the torsional effects exhibited by the coupling. These methods range from measuring changes in the acoustics of coupling mounted piano wires to the application of magnetic circuits which sense changes in permeability as the coupling winds-up. Most of these methods have fallen short of the accuracy required for meaningful use as performance monitoring instrumentation. The challenge for accurate and reliable torque measurement is that each system is faced with determining those physical changes associated with torque alone while the coupling is subjected to a combination of torque, bending, and centrifugal loads. Discriminating the effects between these multiple loads has boiled down to two basic methods of detection:

1) Measurement of localized torsional strain,

2) Measurement of overall torsional deflection.

Strain Gage Type Torquemeters

There are several variations of the strain gage torquemeter system currently available. Each of them operates on the same general principle of:

1) Getting electrical operating power from an outside source to the coupling

2) Feeding that power through a four arm strain gage bridge located on the rotating coupling

3) Transmitting the resulting signal from the coupling back to a stationary receiver.

The strain gages are usually directly affixed to either the OD or the ID of a thinned down area on the coupling spacer (center spool piece) (For slower speed applications, some manufacturers provide a clamp-on split collar which contains the strain gages). As torque is applied, the localized twisting in the area of the strain gages creates a signal by the unbalancing of the strain gage bridge. Since the coupling spacer will be exposed to axial, centrifugal, and misalignment loads in addition to torque, the strain gages of the Wheatstone Bridge must be mounted precisely at 45° from the couplings axis in order minimize the strains from these extraneous loads.

In the past, the method of transmitting power to and receiving signals from the rotating instrumented torque-measuring coupling involved the use of contacting slip ring arrangements. This rendered them useful for only low-speed, high-torque applications and presented problems related to wear and foreign particulates. Today, most strain gage torquemeter systems have overcome these problems by using non-contacting, electro-magnetic induction techniques. The basic strain gage type torquemeter consists of a stationary component and a rotating component. Both components contain electronics. The stationary component (stator) provides power to the rotating component (rotor) via electromagnetic induction between windings contained on each component. The air gap between the stationary and rotating windings allows for relative axial, angular, and offset type excursions of the coupling during operation.

The rotating electronics condition the signal received from the stationary component and feeds it through the strain gage circuitry. (The rotating strain gage circuitry is usually provided by the manufacturer with a protective wrapping due to the sensitivity of the circuitry to handling damage and possible chemical contaminates.). The output of the rotating strain gage circuitry is amplified and transmitted back to the stationary component either by an FM (frequency modulated) signal, or by a second rotary transformer - depending on the manufacturer.

From the stationary component, the signal is typically sent back to the control room as an industry standard analogue signal for connection to the users data recorders or programmable logic controllers (PLC.s).

Torsional Deflection (Phase Shift) Type Torquemeters

As with stain gage types, there are several variations of the torsional deflection torquemeter system currently available. Each of them operates on the same general principle of measuring the torsional *wind-up* experienced when the coupling is exposed to torque by comparing the relative circumferential positions of different locations along the coupling's axis. The most practical way of measuring the coupling torsional wind-up has been found to be the measurement of phase shift of separate speed pickups mounted along the coupling's length. In this respect, *torsional deflection* torquemeters have become synonymous with *phase shift* torquemeters .

Each variation of the *phase shift* measuring *torsional defection* type torquemeter senses the relative positions of opposite ends of the coupling using a pair of toothed flanges which are made as an integral part of the coupling's spacer. The sensing devices are stationary, and work on the same principle as a typical speed pick-up - where the flux field around the pick-ups are changed every time a (steel) tooth on the rotating coupling passes it. By monitoring the phase relationship between toothed wheels affixed to each end of the coupling, an indication of coupling .twist. is obtained.

All phase shift type torquemeter couplings must find a solution to the problem that vertical and horizontal movements of the rotating coupling relative to the non-rotating pick-ups will also produce a phase shift. The method of distinguishing between torsion induced phase shift and those caused by these lateral movements form the basic differences of torsional deflection torquemeter systems.

For phase shift type torquemeters the voltage signals sensed by the pick-ups are typically sent to the control room where the signals are then processed. As with strain gage type torquemeters, the output of the processing unit is also typically made available as an industry standard analogue signal for connection to the user's data recorders or PLC.s. All phase shift torsional deflection torquemeters measure coupling rpm as a bi-product of torque determination, so the torque and speed signals are typically multiplied for a direct readout of power.

2.2.3 Implemented Options

Load and Load Controller

DC machine can be controlled easily. Controlling armature voltage, one can control the speed of the DC machine. Torque control is simply control of armature current under constant field excitation. Since the succeeding in torque and speed control in the dynamic motor tester is the main problem, a separately excited machine is chosen as load.

Suitable load controller for a separately excited DC machine is a four-quadrant DC machine drive with regeneration capability. Regeneration is worth to emphasize because it can send the energy absorbed from the shaft back to the utility grid. Therefore, system does not require energy dissipating elements such as braking resistors.

Sensors and Data Acquisition

Voltage and current sensors are chosen as RMS measuring transducers. These transducers generate DC output voltage proportional to the RMS value of the measured quantity.

Power transducer chosen for the motor test system, measures real power flow through itself. It generates DC output current proportional to the measured real power. This sensor has bi-directional measuring capability, so it can measure the power flow in both directions.

Shaft torque is measured by a strain-gage torquemeter. Torquemeter used in the system has rotary transformers, so it uses neither brushes, nor slip rings. It is used with a specific signal conditioner designed for strain-gage torquemeters. This signal conditioner generates analog voltage proportional to the measured shaft torque.

Finally, for acquisition of these analog signals 16 channel 16 bit A/D converter card is selected. This card is placed in the PC main board and it can easily be used in the Windows environment.

2.3 Theory of Operation

In the previous section, implemented hardware are told very briefly. Following sections of this chapter are going to tell operation principles of the implemented system and they will give further information about the system hardware.

Four-quadrant load controller and data acquisition card have their own software. It is clear that to obtain valid measurements, the most important things are coordination and timing of the hardware during the test. In order to solve coordination and timing problems, a user-friendly interface was designed. The software designed, as well as solving the issues mentioned above, allows easy setting of test conditions and accessing experimental results after the test.

The simplified block diagram of the system is shown below in Figure 2.1 Every element in Figure 2.1, except the tested machine and three-phase utility grid, belongs to the dynamic motor testing system. For the time being load machine is a DC machine and a four-quadrant DC motor drive is used as load controller. The motor, which was tested, is a three-phase induction machine; hence voltage and current sensors used for measurements are AC transducers.

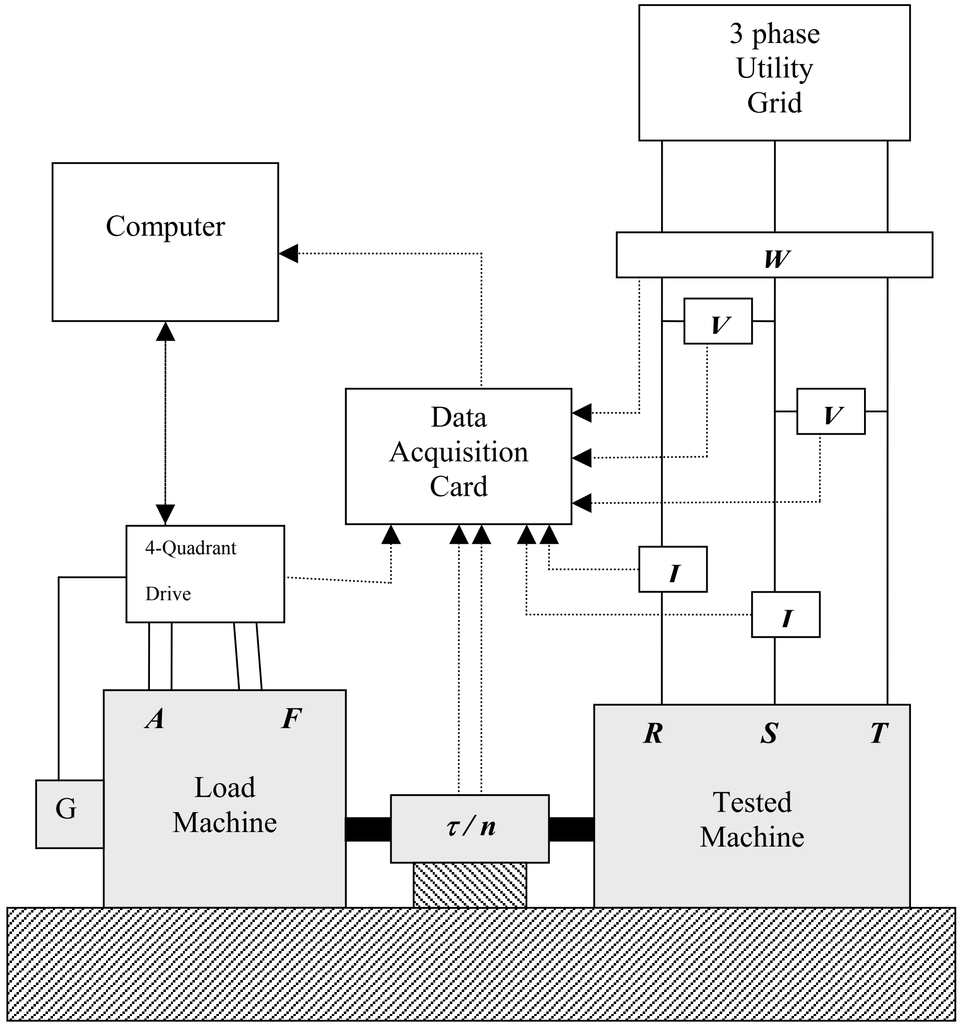

Figure 2.1 Simplified Block Diagram of the System

V : *Voltage Sensor*
I : *Current Sensor*
W : *Three-Phase Power Sensor*
τ / n : *Rotating Torque / Angular Speed Transducer*
A : *Armature Terminals of the Load Machine*
F : *Field Terminals of the Load Machine*
R ,S ,T : *Phase Terminals of the Tested Machine*
G : *Tachogenerator*

2.3.1 Test Conditions

First step of testing is entering the test conditions. Test conditions can be held in two main groups: Initial Conditions and Load Conditions.

A) Initial Conditions: Initial conditions are load side parameters and test motor parameters. These electrical parameters should be entered correctly to obtain safe operation of the system. Mechanical limits are also essential to protect load machine, torque / angular speed sensor and the machine under test. Maximum permissible speed is the mechanical limit to be set. Electrical protection of the load machine is done by declaring current and voltage limits of the armature and field terminals.

B) Load conditions: Second stage of declaring test conditions is defining load. Load can be programmed in two modes:

i) Speed-time load

ii) Torque-time load

User selects either mode of load and then identifies instantaneous values. For instance, user selects speed-time load mode and then enters the angular speed values for the desired instants. During the test, system will trace exactly predefined speed-time values. The load sets desired instantaneous speed independent of the torque produced by the tested machine. In Table 2.1, sample data for speed-time loading is shown. Data in Table 2.1 is plotted in Figure 2.2.

Speed (rpm)	Time (sec)
0	0
200	2
200	4
-400	6
-100	8
0	10

Table 2.1 Sample Speed-time load

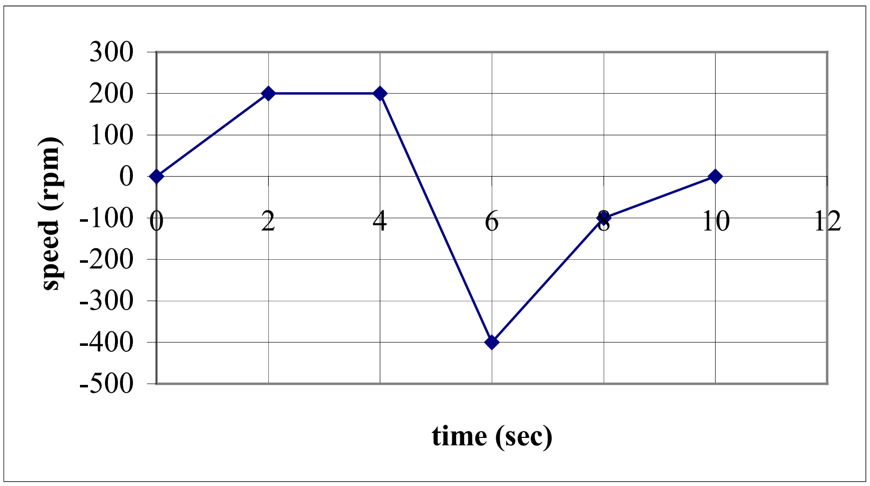

Figure 2.2 Sample Speed-time load

The given speed-time data are realized with a very little error margin as the 4-Quadrant drive employs a tachogenerator for closed loop speed feedback.

The other case is torque-time load mode. This mode of load tracks predefined torque-time values independent of the speed of the shaft.

2.3.2 Starting Test

After entering the test conditions, experiment can be started. As soon as "start" command is given, test conditions are converted to a form suitable for 4-Quadrant drive. Test Conditions are downloaded to the 4-Quadrant drive in this form.

The most important point after this step is synchronization. Load controller generates a timing signal for correct timing. In Figure 2.3 this timing signal and related time intervals are shown.

Experiment timing is achieved by this timing signal. For triggering data acquisition and for running the data acquisition program a delay time is required. Interval t_1 is selected to be larger than the required time. Similarly, time required for stopping data acquisition program and getting back into sleep mode corresponds to interval t_2.

In fact, data acquisition starts before test start-up. A voltage value of V_{ex} is generated during the experiment in order to define the start and stop instants of the test with precisely.

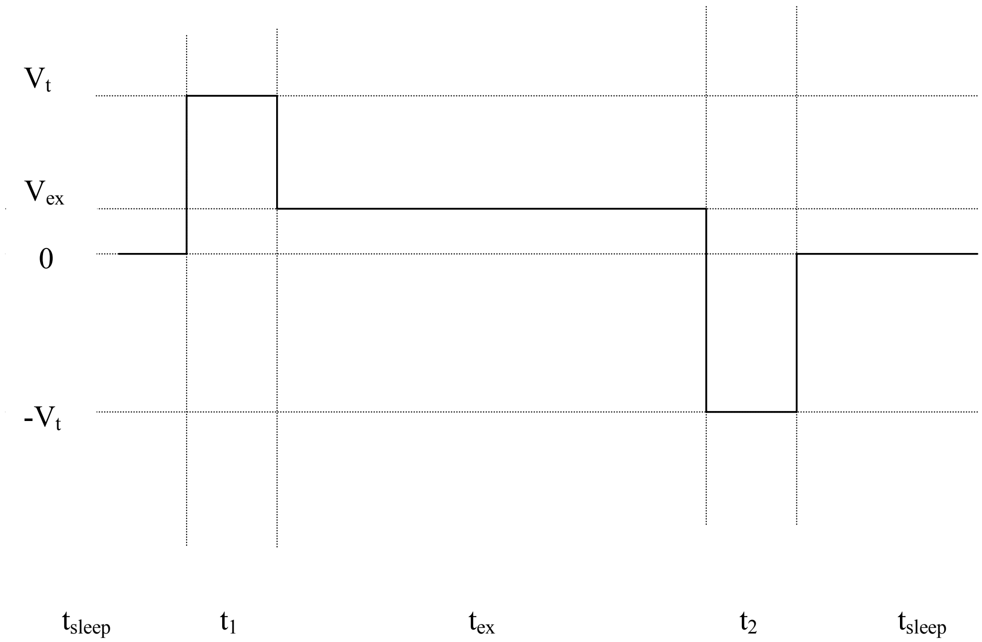

Figure 2.3 Timing signal generated by 4-Quadrant Drive

0 : *zero signal generated during sleep period*
V_t : *triggering signal starting experiment and data acquisition*
V_{ex} : *signal generated during the experiment*
$-V_t$: *triggering signal stopping experiment and data acquisition*
t_{sleep} : *sleeping time when no data acquired*
t_1 : *time interval when data acquisition started*
t_{ex} : *experiment time indicating the presence of valid data*
t_2 : *time interval when data acquisition stopped*

2.3.3 Demonstration of Acquired Data

Data acquired during the test are stored in an ASCII (*.asc) file. The file format consists of a header and data rows for each sampling time. Beginning part of the file is shown below:

DASYLab - V 5.01.10
WORKSHEET : veri
Recording Date : 08.08.2002, 23:07:52
Block Length : 64
Delta : 0.010000 sec.
Number of Channels : 8

Time	Ch #0	Ch #1	Ch #2	Ch #3	Ch #4	Ch #5	Ch #6	Ch 7
0,00	-0,01	-0,95	-1,40	-2,12	-0,00	0,17	0,29	4,88
0,01	-0,00	-0,60	-0,77	-1,45	0,00	0,18	0,41	4,88
0,02	0,01	-0,03	-0,44	-1,00	-0,00	0,14	0,42	4,88
0,03	0,01	-0,02	0,01	-0,53	0,00	0,15	0,49	4,88
0,04	0,00	0,42	0,19	-0,22	-0,00	0,10	0,47	4,88
0,05	0,00	0,27	0,51	0,12	0,00	0,12	0,51	4,88

First column stands for elapsed time. The other columns shows data acquired at each analog input channel. File length depends on data acquisition rate and experiment period. Data stored in this file are presented by tables and graphs to user.

2.4 Hardware

2.4.1 Load

Dynamic load is an essential part of the dynamic measurement system. Load used in the system is a separately excited DC machine. Such a machine serves well for both speed-time and torque-time type loads.

As far as closed loop speed feedback is employed, speed control and speed-time type load can be realized with many types of motors. However, this is not the case for torque-time type loads. Implementation of torque control is rather hard for some kind of motors. Under constant field excitation, separately excited DC machine is suitable for torque control. Well-known torque equation for DC machines is given below.

$$T_m = K_m * \phi_f * I_a \qquad (2.1)$$

T_m : Torque Generated (Nm)
K_m : Machine Constant (Nm / A*Weber)
ϕ_f : Air-Gap Flux per Pole (Weber)
I_a : Armature Current (A)

As clearly seen from the equation, torque control for separately excited DC machine has a very simple logic. Under constant field excitation, generated torque is directly proportional to the armature current. Hence controlling the armature current, user can obviously change torque reference.

Rated values and dimensions of DC motor are chosen in purpose to satisfy the needs of test conditions. Load machine is relatively larger than the tested machine to have the ability of dictating speed. In other words, load machine should overcome the torque generated by the tested machine.

2.4.2 4-Quadrant Drive

DC motor drives are widely used in applications that require regeneration, precise speed control, dynamic performance, and constant torque over wide speed ranges. A 4- quadrant commercial motor drive with regenerative braking is employed for the system. In Figure 2.4, the drive used in this study is shown.

Figure 2.4 4-Quadrant Drive: Mentor II

Dynamic load demands complete control of motor operation in both directions with the ability to reverse motor torque rapidly and frequently. Therefore, two anti-parallel thyristor bridges must be used, shown in Figure 2.5.

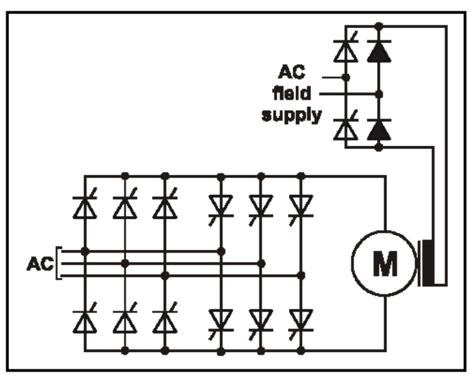

Figure 2.5 Dual bridge or parallel-pair 3-phase thyristor (SCR) Arrangement for 4-quadrant DC motor drive

This configuration provides full control of forward and reverse drive and forward and reverse braking without the reversing contactors. Therefore, it is called four-quadrant DC motor drive (Figure 2.6).

Regardless of whether a drive is single-quadrant or four-quadrant, motor response is fundamentally a function of voltage output, which is a function of the firing angle of the thyristor bridge, and this can be controlled precisely.

The quality of the response obtained from the motor is, therefore, dependent on the ability of the drive logic to receive, interpret and process a complete range of data concerning the state of the motor, and the desired state. Some of this data may be from external sources, such as the speed reference (demand), torque reference,

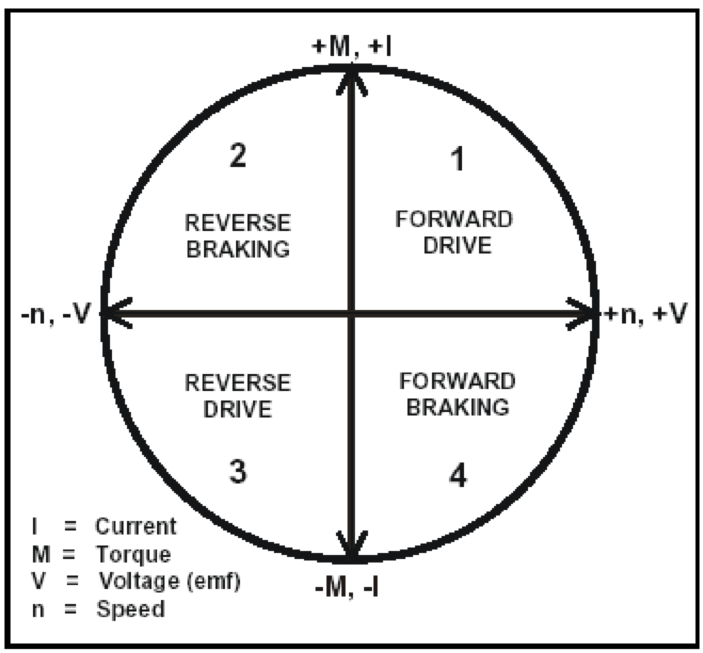

Figure 2.6 The four quadrants of the DC motor torque-speed diagram

motor speed feedback, and so on; some are derived internally by the drive logic itself. These are, for example, output voltage and current, and the demand condition of the logic system at various stages.

The logic system requires a set of instructions to allow it to undertake the process of interrogation, processing and signal-generation to control thyristor (SCR) firing. The instructions are provided in the form of data broken down into individual values or parameters.

Mentor II Drive is equipped with a dedicated microprocessor, and with software that is configured by the parameters written to it by the user. The parameters cover every significant factor related to motor performance, so that the user can set the drive up to meet the load requirements exactly.

Speed-time load is simply realized by setting speed parameter of the drive to a certain value. Load (that is DC machine) is coupled to a tachogenerator that produces a direct voltage proportional to shaft speed. This speed feedback is connected to the tachogenerator input of the DC drive. The closed loop speed feedback controls shaft speed with precision regardless of torque demand of the test machine. However, in practice it is observed that application of tachogenerator, gives speed control down to around 10 rpm. Speed control at lower speed values –even at zero speed— can be made by using an optic shaft encoder.

Current (i.e. torque) has a similar closed loop control. In the current control case however, no extra current measurement device is added to the system. Mentor II drive has built-in current sensors and achieves good average current control. Unfortunately, precisely controlled average does not mean pure direct current. Especially for small armature current values (when thyristors stay in conduction for a short period), armature current waveform is made up of periodic pulses. These current pulses indicate pulses in torque generated.

Mentor II has an application card, namely MD29 (Figure 2.7), which is used for RS232 communication and programming. The MD29 application card contains a microprocessor that provides a low-cost facility for the system designer to write application specific programs without a programmable logic controller.

The application card provides intimate high-speed bi-directional access. It can read and modify any parameter within the drive, enabling customized real-time calculations under a multi-tasking run-time environment. The Intel i960 32-bit RISC processor and 256K of user program FLASH memory (equivalent to >2000 lines basic instruction code) provide a powerful base for programming. Such a memory is sufficient to satisfy the needs of a dynamic load definition.

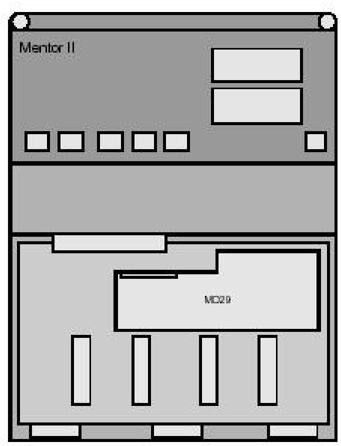

Figure 2.7 Location of the MD29 in the Mentor II Drive

4 – quadrant DC drive is used as the master drive in the test system. Load conditions are downloaded to DC and it controls the loading conditions. For example, in speed-time load case speed of the shaft is completely controlled by the 4-quadrant DC drive.

Load control performance of the DC drive is restricted by the capabilities of the static switches placed in it. Mentor II has full bridge of silicon-controlled rectifiers, which rectifies of 3-phase 50 Hz input voltage. Bridge has triggers 6 switches in each cycle, so the upper band limit of the load control is 300 Hz.

Finally, it must be noted that Mentor II drive has programmable analog outputs, one of which is used for generating timing signal shown in Figure 2.3.

2.4.3 Sensors and Signal Conditioning Equipment

Dynamic motor test requires dynamic measurement of desired signals. For this purpose, system includes various kinds of sensors. Sensors selected depend on the type of motor under test, expected accuracy and linearity. For the initial tests and demonstration, a squirrel cage induction machine is tested. Two line currents, two line-to-line voltages and real power flow into machine are the measured electrical parameters. Mechanical terminals of the tested machine are identified by shaft speed and torque. These parameters are also recorded to have a complete electromechanical perspective.

2.4.3.1 <u>Electrical Sensors</u>

AC Voltage Transducers

Two line-to-line voltages of tested squirrel cage induction machine are measured by two RMS transducers. In Figure 2.8, voltage transducer used is shown.

Figure 2.8 AC RMS Voltage Transducer: VTR-004D

The RMS value of the current or voltage is the effective or DC equivalent value of that current or voltage. The VTR voltage transducers calculate the effective value of current and voltage using a very close approximation of the root mean square integral.

VTR-004D is supplied by external 120 V ac at 50 Hz (terminals 3 and 4). Sensor generates 10 V dc (terminals 5 and 6) corresponding to full-scale input, which is 600V ac (terminals 1 and 2). In Figure 2.9, terminals of VTR-004D are shown. Current transducers and real power transducer also require a 120 V supply. We employed 220V /120V transformer with 70 VA power rating, serving as common power source for electrical sensors. Response time of the voltage transducers is 100msec (up to% 90 of the measured value)

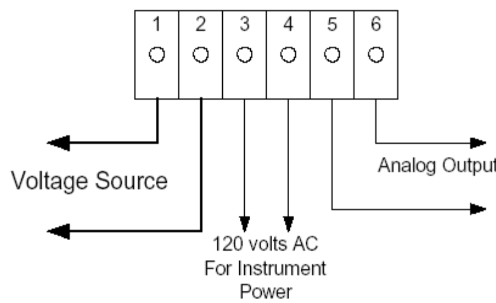

Figure 2.9 Terminals of voltage transducer VTR-004D

AC Current Transducers

Two line currents of the tested machine are measured by RMS current transducers. In Figure 2.10, current transducer used is shown.

Figure 2.10 AC RMS Current Transducer: CTRS-005D

CTRS-005D is supplied by external 120 V ac at 50 Hz (terminals 3 and 4). Sensor generates 10 V dc (terminals 5 and 6) corresponding to full scale input, which is 5 A AC (terminals 1 and 2). Figure 2.11 shows terminals of CTRS-005D. As the line currents are directly applied to terminals 1 and 2, protection of the sensor is essential. We applied 5A fuses to each line having a current transducer. Response time of the current transducers is 100msec (up to% 90 of the measured value).

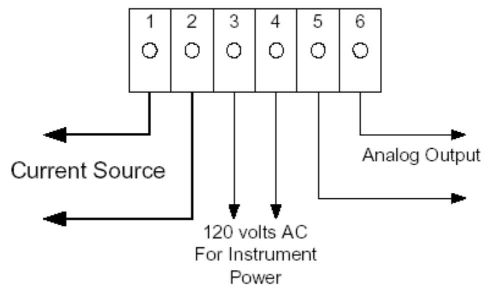

Figure 2.11 Terminals of voltage transducer CTRS-005D

Three-Phase Wattmeter

Real power input to the test machine is an important parameter. System has a 3-phase real power transducer. It is shown in Figure 2.12.

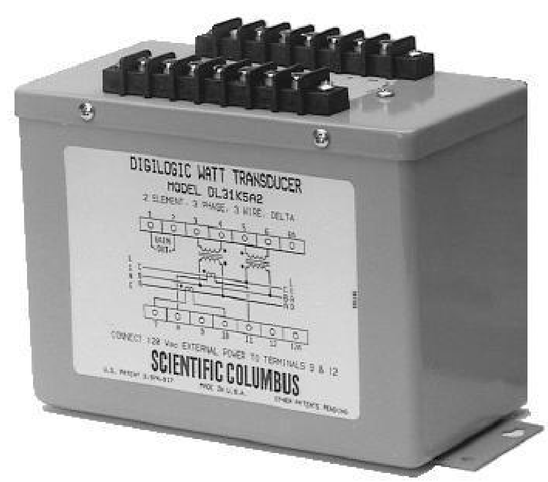

Figure 2.12 3-phase Real Power Transducer: Digilogic DL34-2K5-A2-2

Real power transducer is used to measure the real power flow from 3-phase utility grid to induction machine. Unlike the other sensors in the system, the power transducer has current output. It produces 1 mA dc output (output is bi-directional indicating power flow in both directions) corresponding to full scale (6000 kW) real power. 10 kΩ metal film resistor is connected parallel to output and it converts 1 mA output to 10 V. A dc voltage output, with a maximum of 10V, is appropriate for the data acquisition card. Response time of the power transducers is 1 sec (up to% 99 of the measured value)

Input and output connections for the real power transducer are shown in Figure 2.13. Terminals 1 and Terminal 2 supplies dc current output. Terminal 4-6 are reserved for voltage input, whereas terminals 7-12 stand for current measurement. Transducer is supplied 110 V ac from terminals 6A and 12A. The real power transducer used in the system has 480V / 5A voltage and current ratings respectively. Therefore, voltage and current transformers shown in Figure 2.13 are not used. Direct connection of the inputs increases the accuracy of power measurement.

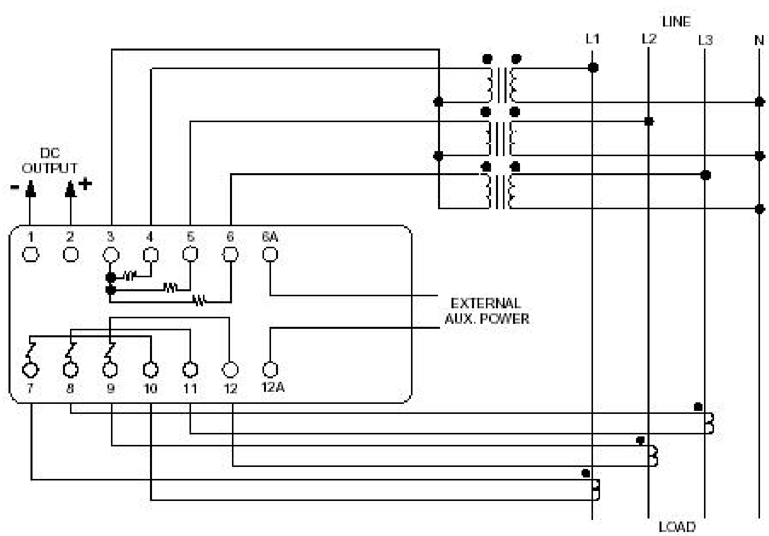

Figure 2.13 Terminals of Real Power Transducer

2.4.3.2 Mechanical Sensors

Torque and speed are measured by rotating type strain gage torquemeter. Torquemeter is shown in Figure 2.14. It is mounted in series between test motor and load motor.

Figure 2.14 Strain Gage Torquemeter: Himmelstein 9-02T (1-3)

Himmelstein 9-02T (1-3) strain gage torquemeter has a speed limit of 15000 rpm. The upper limit of torque, on the other hand, is 1000 lb-inches. Both limits of strain gage torquemeter are acceptable for the system built.

When external forces are applied to a stationary object, stresses and strains are the result. Stress is defined as the object's internal resisting forces, and strain is defined as the displacement and deformation that occur. It is known that metallic conductors subjected to mechanical strain exhibit a change in their electrical resistance. This phenomenon is used to convert mechanical effect (i.e. torque) to an electrical signal. Obviously, an external signal conditioner is needed for torque measurement to work cooperatively with strain gage.

Speed output of the torquemeter is 60 pulses/revolution. It generates a triangular wave whose frequency is proportional to speed rotation. For instance, triangular wave frequency is 1500 Hz for 1500 rpm shaft speed. As data acquisition card requires analog output, speed data stored in frequency form has to be converted to analog output. An external frequency-to-voltage converter should be connected.

Himmelstein 6-488B Torque and Speed Readout (Figure 2.15) has both strain gage amplifier for torque output and frequency-to-voltage converter for speed terminal. In addition to its digital displays, there also exist analog outputs corresponding to speed and torque. These analog outputs are directly connected to two channels of data acquisition card.

Figure 2.15 Torque and Speed Readout: Himmelstein 6-488B

2.4.4 Data Acquisition Card

System owns a data acquisition card, namely Advantech PCL-816 (Figure 2.16). In the previous sections, sensors used in the system were given. Each sensor is connected to a differential analog input channel of data acquisition card. Channel numbers and corresponding hardware are given in the Table 2.2.

Ch #	Hardware	Firm	Model	Analog Input
0	AC RMS Voltage Transducer	Ohio	VTR-004D	10 V dc / 480 V RMS
1	AC RMS Voltage Transducer	Ohio	VTR-004D	10 V dc / 480 V RMS
2	AC RMS Current Transducer	Ohio	CTRS-005D	10 V dc / 5 A RMS
3	AC RMS Current Transducer	Ohio	CTRS-005D	10 V dc / 5 A RMS
4	3-phase Real Power Transducer	Digilogic	DL34-2K5-A2-2	±10 V dc / 6000 W
5	Strain Gage Torquemeter	Himmelstein	9-02T (1-3)	1 V dc /1 Nm
6	Strain Gage Torquemeter	Himmelstein	9-02T (1-3)	2 V dc /1000 rpm
7	4-Quadrant Drive	C T	Mentor II	± 5 V dc

Table 2.2 Advantech PCL-816 analog input channels and connected hardware

As seen from Figure 2.16, Advantech PCL-816 is directly inserted into the main board of the personal computer. Analog inputs are connected through 37-pin female connector. PCL-816 accepts analog inputs of ± 10V. These inputs are converted to digital form in analog input module with 16 bits resolution. This resolution level is acceptable for the designed dynamic measurement system.

Data acquisition card can be run in two ways. First way is running an external data acquisition program. This data acquisition program automatically makes the necessary operations such as conversions and storing. Other method is proper use of associated files. During this study, we chose DasyLab as a data acquisition program. Details are going to be held be in the related parts of Software chapter.

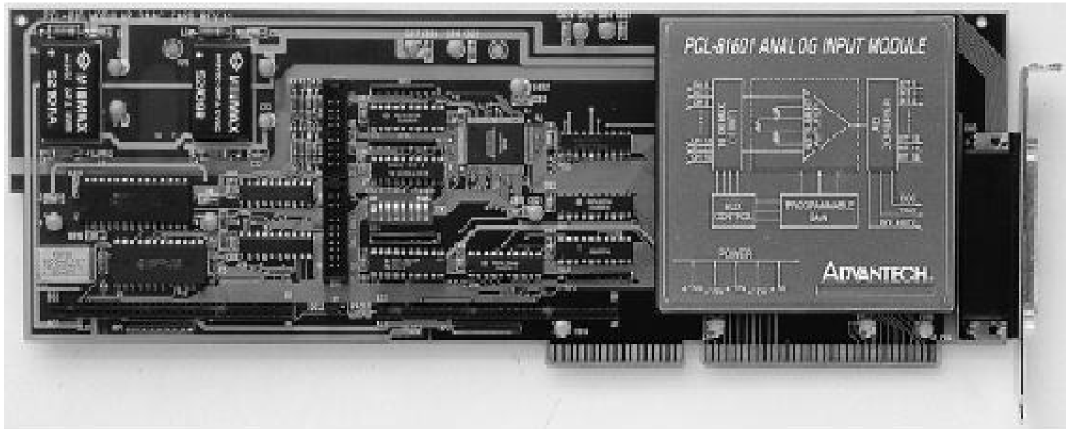

Figure 2.16 Data Acquisition Card: Advantech PCL-816

2.4.5 Computer

Personal computer used will not be explained in detail since a standard personal computer is used. Personal computer has a 200 MHz processor and 32 Mbytes RAM. Operating System used in the system is Microsoft Windows 98.

A serial port of the computer is reserved for RS-232 communication with Mentor II DC Drive. Desired test conditions are downloaded to DC Drive through this serial port.

System requires an empty slot for the data acquisition card (Advantech PCL-816). Data acquisition card is an internal one and is operated in the main board of the personal computer.

2.4.6 Tachogenerator

DC Drive used as load controller has closed loop speed control. In order to get speed feedback from system shaft, a tachogenerator is used. This tachogenerator produces a DC output voltage, which is directly proportional to value of shaft speed. Output of the tachogenerator has ripples on the average value. A low pass RC filter is used (R=220 Ohm, C= 47 microfarad) to filter these ripples on the tachogenerator output.

To observe the tachogenerator characteristic, speed profile shown in Table 2.3 is set as loading condition. This profile is shown at Figure 2.17. It is observed that tachogenerator used for speed feedback purpose has acceptable input-output characteristic.

Speed (rpm)	Time (sec)
0	0
250	1
250	5
500	6
500	10
750	11
750	15
1000	16
1000	20
0	25

Table 2.3 Speed Profile

Following results are obtained for this speed-time profile:

Analog Output of Speed at Himmelstein Torque-Speed Transducer.....Figure 2.18

Analog Output of Tachogenerator……………………………… ……………Figure 2.19

Analog Output of Tachogenerator (Low pass filtered)……..…………Figure 2.20

Analog Output of Tachogenerator at 1000 rpm……………………..…… Figure 2.21

Analog Output of Tachogenerator at 1000 rpm(Low pass filtered) ...Figure 2.22

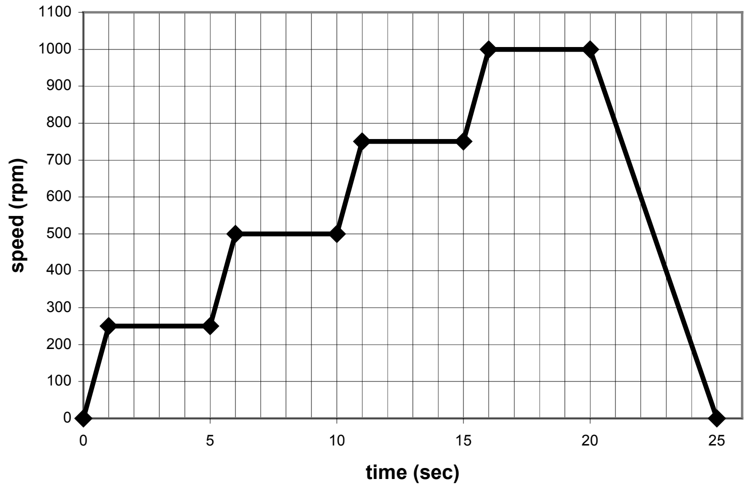

Figure 2.17 Speed Profile

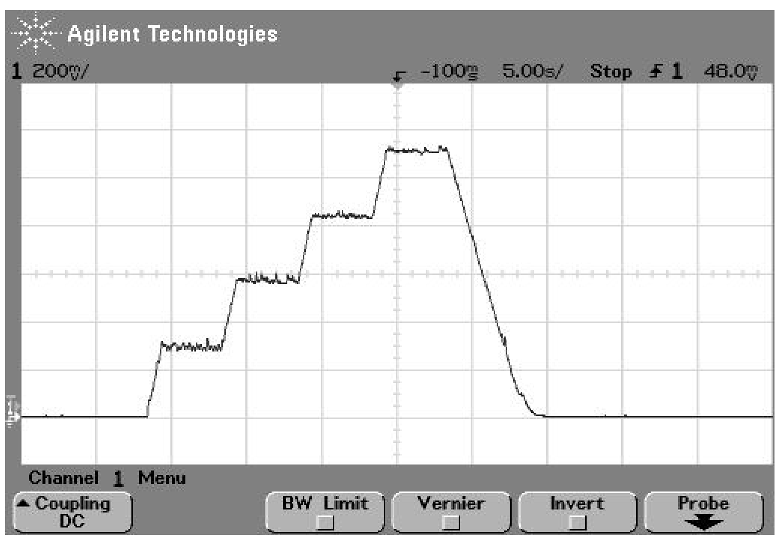

Figure 2.18 Analog Output of Speed at Himmelstein Torque-Speed Transducer

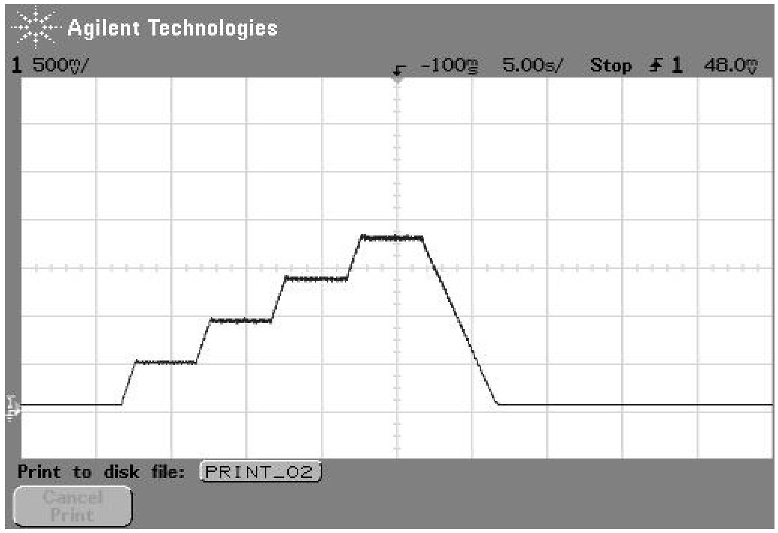

Figure 2.19 Analog Output of Tachogenerator

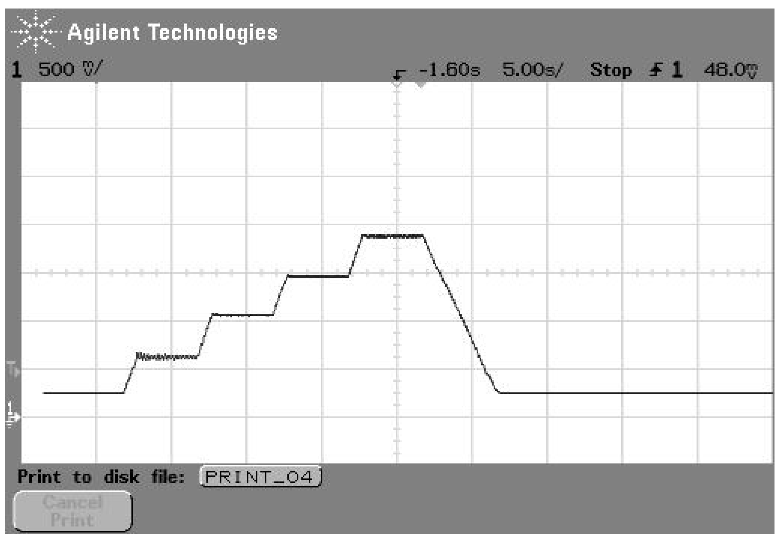

Figure 2.20 Analog Output of Tachogenerator (Low pass filtered)

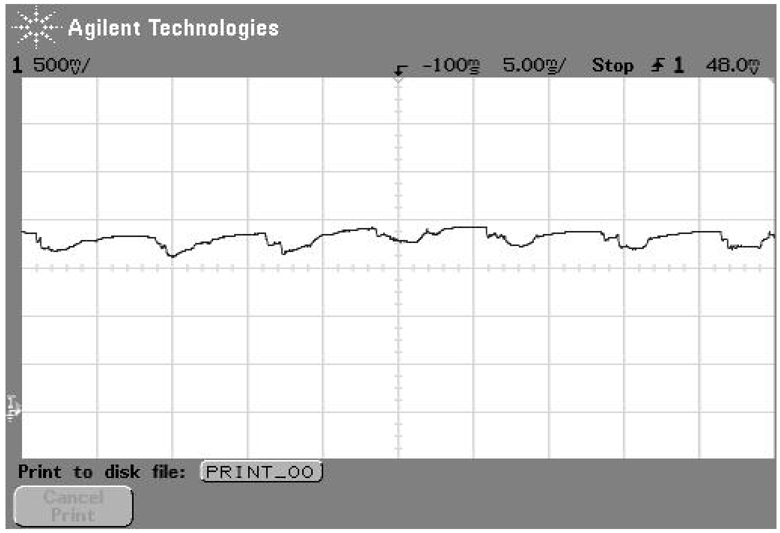

Figure 2.21 Analog Output of Tachogenerator at 1000 rpm

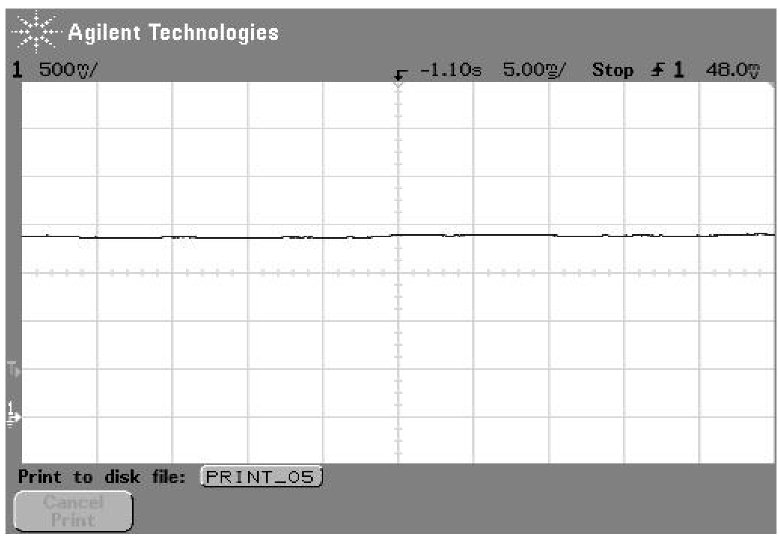

Figure 2.22 Analog Output of Tachogenerator at 1000 rpm(Low pass filtered)

2.5 Mechanical Considerations

The test machine in the test system is connected to the load machine via a torque-speed transducer. Therefore, mechanical part of the system has following components:

i) Test Machine Rotor

ii) Shaft between Test Machine Rotor and Coupling 1

iii) Coupling 1

iv) Shaft between Coupling 1 and Torque-Speed Transducer

v) Torque-Speed Transducer

vi) Shaft between Torque-Speed Transducer and Coupling 2

vii) Coupling 2

viii) Shaft between Coupling 2 and Load Machine Rotor

ix) Load Machine Rotor

These components are also shown in Figure 2.23 with their locations in the actual system. All these components have their own moment of inertia and finite rotational stiffness values and friction constants. Moment of inertia in a rotating system can store kinetic energy, whereas an element having torsional elasticity (i.e. has finite stiffness) can store potential energy. The result of these two energy storages may be oscillation[8].

Throughout this study, ideal transfer between load machine rotor and test machine rotor is assumed. For idealizing torque transfer, following assumptions are made:

Assumption 1 : All components are loss free.

Assumption 2 : Components ii,iii,iv,v,vi,vii and viii have zero moment of inetia and infinitive stiffness. So they can carry neither kinetic, nor potential energy.

A mechanical system with these assumptions has following properties for torque transfer:

i. Unity gain for all frequencies of torque input (no loss, no amplification)

ii. Zero phase response for all frequencies of torque input (no delay, no advance)

During this study, line fed (f=50 Hz) induction machine is used as test machine. A complete simulation of the system including mechanical considerations must be performed before running the system with a different frequency.

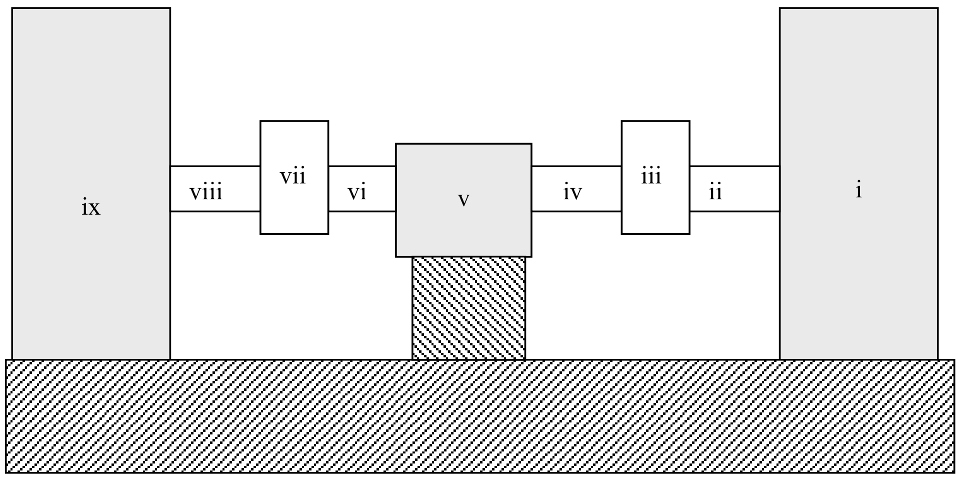

Figure 2.23 Locations of the mechanical components in the test system

CHAPTER 3

SOFTWARE

3.1 Introduction

In the previous chapter, hardware of the dynamic motor tester is introduced. System hardware consists of many different sections. From the software point of view, two sections are more important. These are 4- quadrant drive and data acquisition card. These two parts have direct connections to personal computer and they realize the communication and interaction between hardware and software. Sensors and signal conditioners are to be operated continuously when power is up. Third chapter has mainly two goals. First goal is the successful operation of the 4-quadrant drive and data acquisition card. Second is coordination for all hardware and design of a user-interface, which handles all the necessary tasks.

3.2 Software Overview

3.2.1 Systematic Approach

Designing and writing programs should have a systematic way[3]. These steps are referred to as the program life cycle and are summarized in Table 3.1.

These design steps have approximately equal weights. Successfully operating software starts with the true analysis of the problem; that is, the designer must determine what information is available and what is the intended outcome.

Afterwards the designer must design a plan to determine what must be done and in which order to achieve the required outcome. Except for very simple

Step	Procedure
1	Analyze the Problem
2	Design a Solution Plan
3	Construct an Algorithm
4	Implement the Algorithm
5	Test and Debug Algorithm

Table 3.1 Program Life Cycle

problems, the plan must be hierarchical, meaning the problem is divided into subgoals, which must be reached, and a plan is developed for each subgoal.

Once the solution plan has been established, the designer starts to concentrate on how each step of the plan can be achieved by the computer. This requires the designer to construct an algorithm for each goal/subgoal of the plan. An *algorithm* is a series of actions in a specific order, and programmers often represent their algorithms as *flowcharts*.

The fourth step in the program life cycle is the implementation of the algorithm in a particular programming language and its related syntax. Syntax refers to the rules of a language, including such things as the proper use of semicolons and the exact format of each statement.

When the program is free of syntax errors, the designer must run the program to test the results. To eliminate the errors, the process of debugging the program (identifying and correcting errors) begins, which involves reconsidering assumptions and decisions at each of the previous stages of the life cycle.

While designing our software, we obeyed the steps in the program life cycle. Functions of the software are going to be given in a general perspective, so the details of the program syntax are skipped. To be complete, full program code is presented in the Appendix.

3.2.2 Employed Software Tools

During program coding, three different software were employed. These software are tabulated in Table 3.2.

Software	Firm	Version
DPL Toolkit	Control Techniques	3.3
DasyLab	Dasytec	5.01.10
C ++ Builder Professional	Borland	4.0

Table 3.2 Employed Software

Brief information about each is going to be given below. First motivation of this briefing is to demonstrate roughly their basic properties. This gives the reader an idea about the platform where system software is designed. Another motivation is claiming that they are acceptable choices for programming the required tasks. Complete description of these programming languages is obviously out of the scope of this thesis.

3.2.2.1 **DPL Toolkit**

DPL Toolkit stands for Drive Programming Language Toolkit. Shortly speaking, DPL Toolkit is a Windows based developing environment for Drive Programming. In Figure 3.1 DPL Toolkit window is shown. This software is designed by Control Techniques to satisfy the needs of programming complicated motor drives. Throughout this study, Mentor II DC Motor Drive is used as the dynamic load controller. Mentor II has read/write parameters as well as read-only parameters. Drive Programming Language can easily access these parameters via MD29 application card.

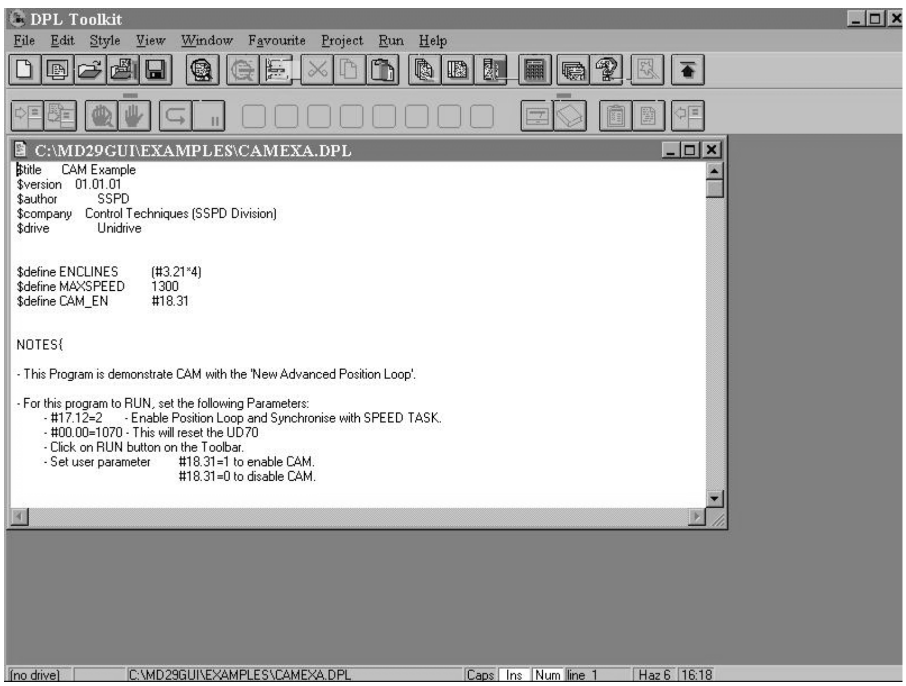

Figure 3.1 DPL Toolkit

DPL is in some means resembles to BASIC. It compiles files with extension *dpl.* The main workhorse of DPL is the set of factory defined drive parameters. A few parameters are tabulated in Table 3.3. These parameters –since they are factory defined— can be used without any declarations. In addition to these factory parameters, software allows introducing other user-defined parameters.

Parameter	Value
#01.06	Forward Maximum Speed
#01.07	Forward Minimum Speed
#01.08	Reverse Minimum Speed
#01.09	Reverse Maximum Speed
#02.04	Forward Acceleration
#02.06	Reverse Deceleration
#03.01	Final Speed Demand
#03.09	Speed loop Proportional gain
#03.10	Speed loop Integral gain
#03.11	Speed loop Derivative gain
#03.15	Maximum Armature Voltage
#04.05	Current Limit Bridge 1
#04.09	Current Offset

Table 3.3 Some of the Mentor II parameters

Simple conditional statements and while loops are available in Drive Programming Language. Variables, conditionals and finally loops are enough for coding simple programs. DPL programs are saved with *dpl* extension (such as *example.dpl*). Afterwards this file is compiled. If program is free of errors, then it is processed. File is converted through many steps and become to file format with *bin* extension (such as *example.bin*). For each step, a specific file is used and a new file format is generated. These steps and related files are shown in Figure 3.2. Ellipses stand for generated files, and rectangles are the executed files to generate these files. Finally, generated binary file (*example.bin*) is downloaded to Mentor II, which is the load controller of the system.

File flow mechanism in Figure 3.2 is important, because it makes external control of the Mentor II an easy task. “Dynamic Motor Tester” (software designed for the system) automatically generates the required *dpl* file (say *example.dpl*), which includes the user-defined test conditions. Then, it calls the executable files in Figure 3.2 one by one. At last, it executes *flasher.exe* file to complete the download process. By this way, the communication between “Dynamic Motor Tester” and DPL Toolkit is achieved. Moreover, all facilities of Drive Programming Language can be used, without awaking the DPL Toolkit environment. Executable files in file generation are called by the code written in designed user interface.

3.2.2.2 DasyLab

DasyLab is a data acquisition, process control and analysis system which takes full advantage of the features and graphical interface provided by Microsoft® Windows.

The most important design requirements for DasyLab were the integration of the important measuring and control devices on the market, a truly intuitive operating environment which offers help functions, a maximum signal processing speed and the effective graphical display of results.

Using DasyLab, a measuring, process control, or simulation task can be set up directly on the screen by selecting and connecting modular elements, which can be freely arranged.

Although DasyLab has extended programming capabilities, during our study it’s used only for handling data acquisition whenever desired. Data acquisition tasks require fast and continuous communication with the data acquisition card. We chose to overcome data acquisition problem with DasyLab rather than establishing a solution with C++ Builder.

Files created for data acquisition are “tetik.dsb” and “veri.dsa”. These files operate in a cyclic manner by the help of the control signal generated by DC Motor Drive.

Tetik.dsb (Figure 3.3) has only three modules: Analog Input Module, Trigger Module and Action Module. Trigger module checks whether valid trigger signal is present or not. If there exists valid trigger signal, it stimulates the Action Module. Action Module opens and runs the file veri.dsa and data acquisition starts.

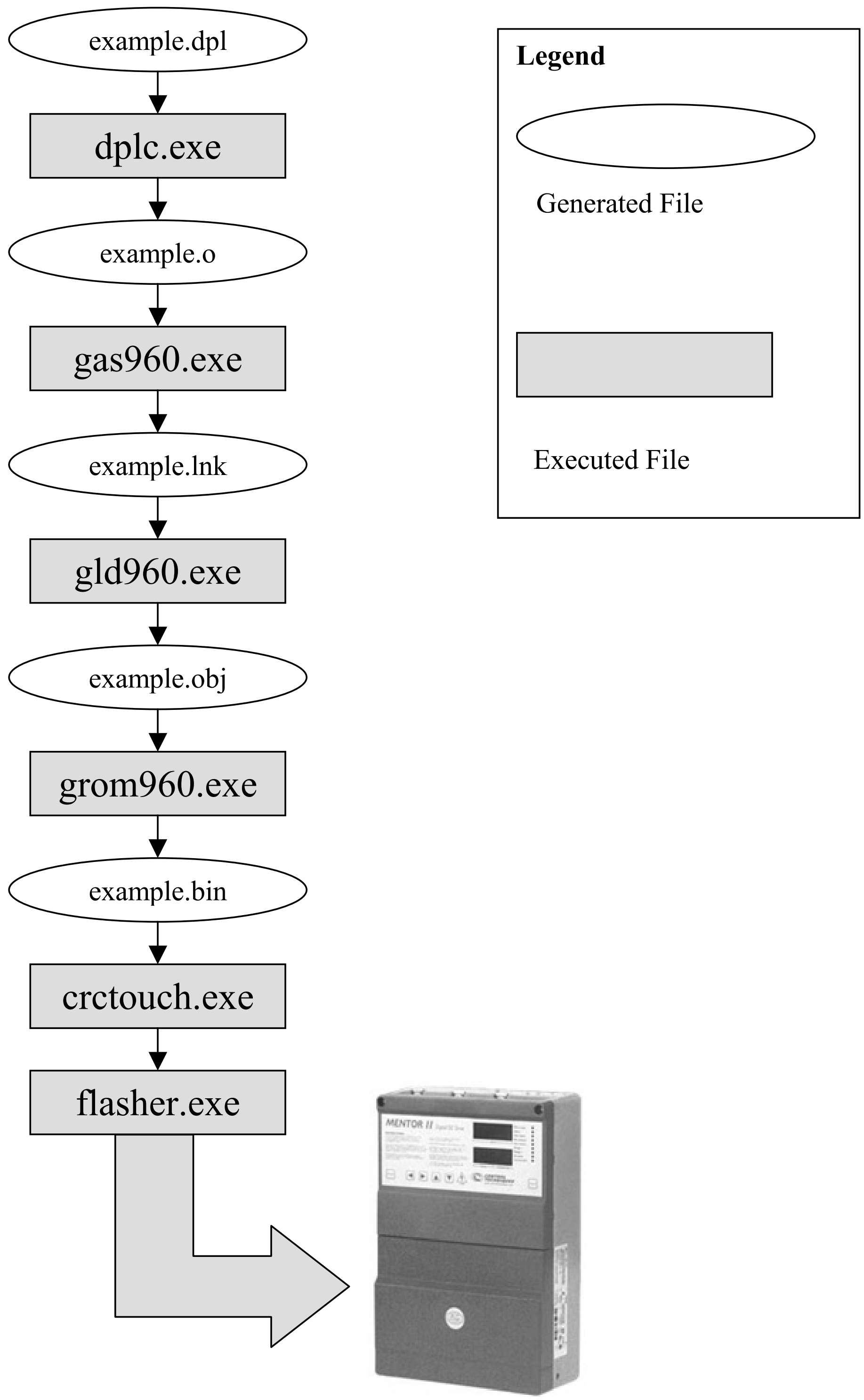

Figure 3.2 File generation and download process in DPL

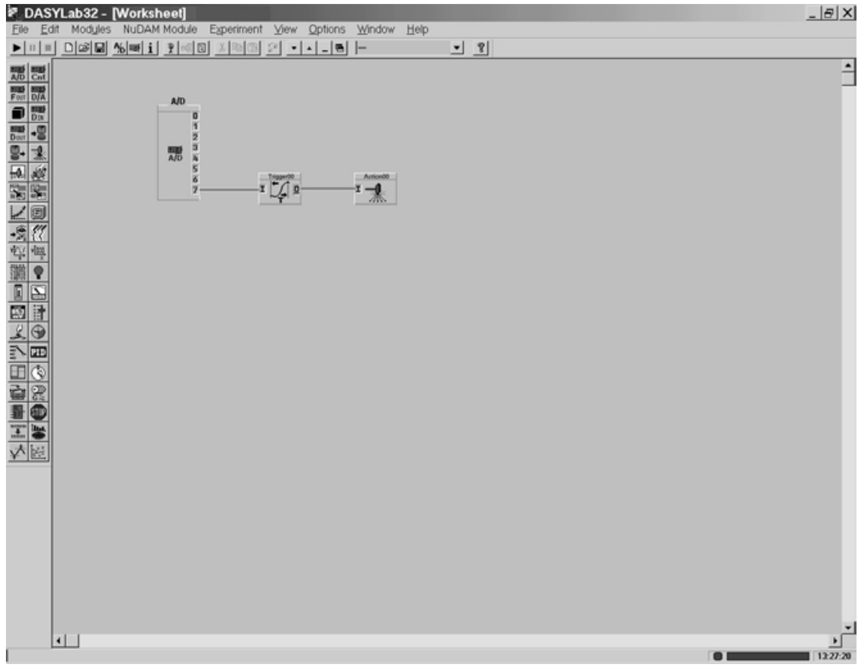

Figure 3.3   DasyLab "Tetik.dsb" file

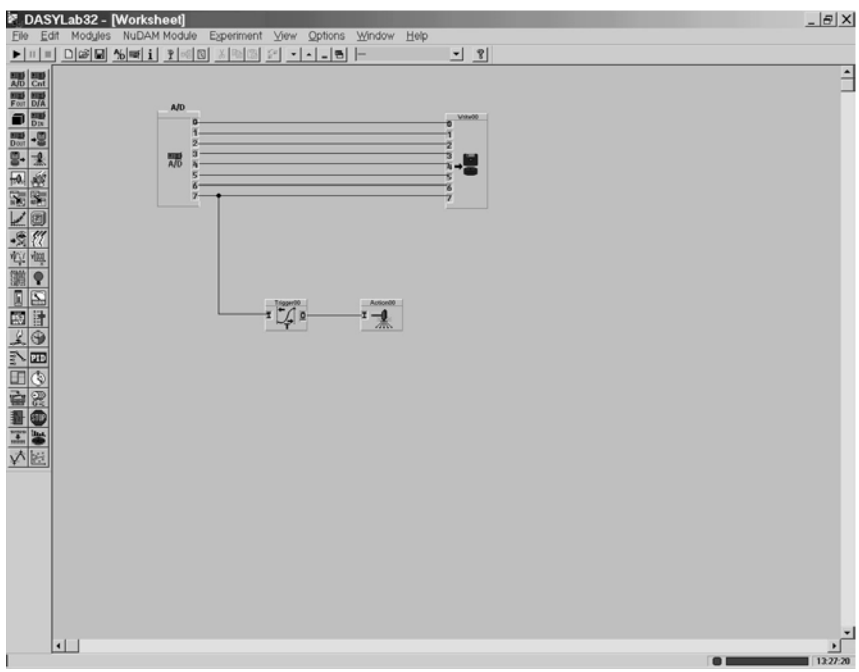

Figure 3.4   DasyLab "Veri.dsa" file

Veri.dsa (Figure 3.4) has four modules: Analog Input Module, Trigger Module, Action Module and Data Save Module. As soon as veri.dsa file is executed, data acquisition starts Trigger Module, this time, checks whether valid stop trigger is present or not. If there exists valid stop signal, it stimulates the Action Module. Action Module opens and runs the file tetik.dsb and data acquisition ends. The data acquisition card goes back to stand-by state.

3.2.2.3 C++ Builder

C++ Builder is an object-oriented programming language. Software designed by C++ builder has Windows compatible visual user-interface.

C++ Builder has a wide palette of built-in objects (Figure 3.6). These objects include common Windows objects such as buttons, combo-boxes etc. These objects make the designed software user-friendly for the end-user. Most of the properties of the objects are predefined. This gives the programmer a standard basis. Furthermore, most of the object properties are accessible through the object inspector (Figure 3.7). Any function or software procedure assigned to an object can be defined by the written code by the help of code window (Figure 3.5). Finally, C++ Builder creates portable executable file automatically for every compilation.

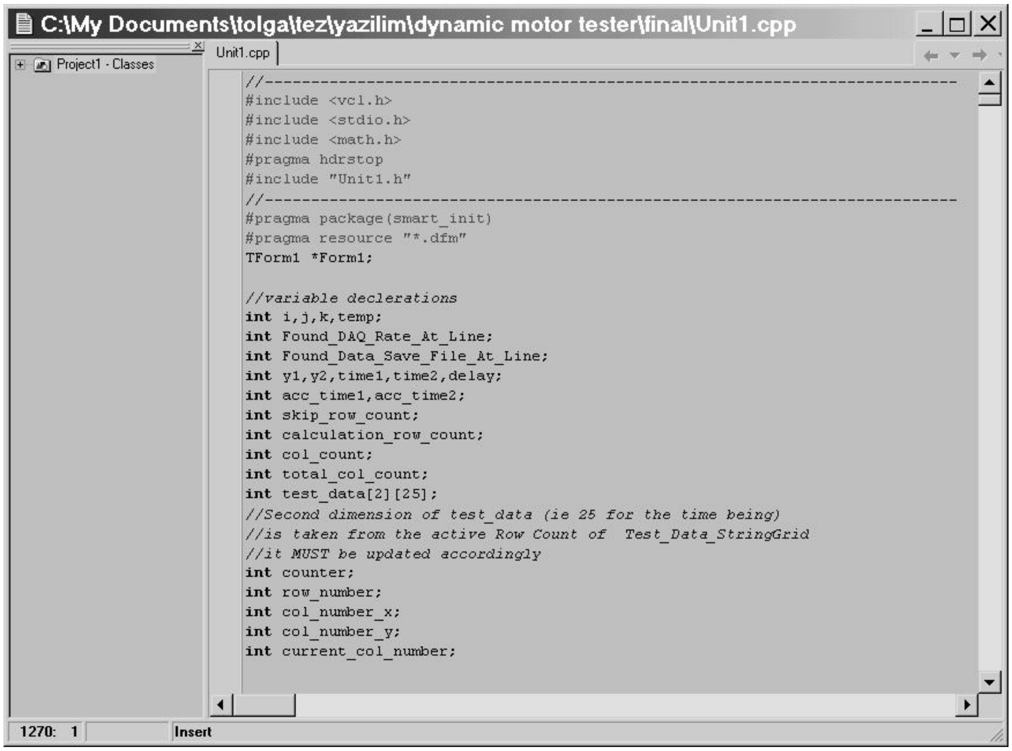

Figure 3.5 C++ Builder Code Window

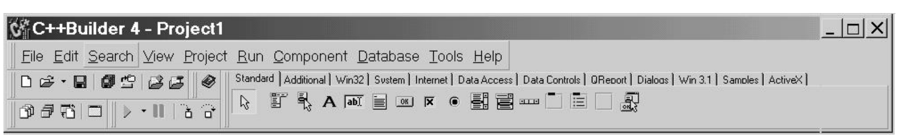

Figure 3.6 C++ Builder Object Palette

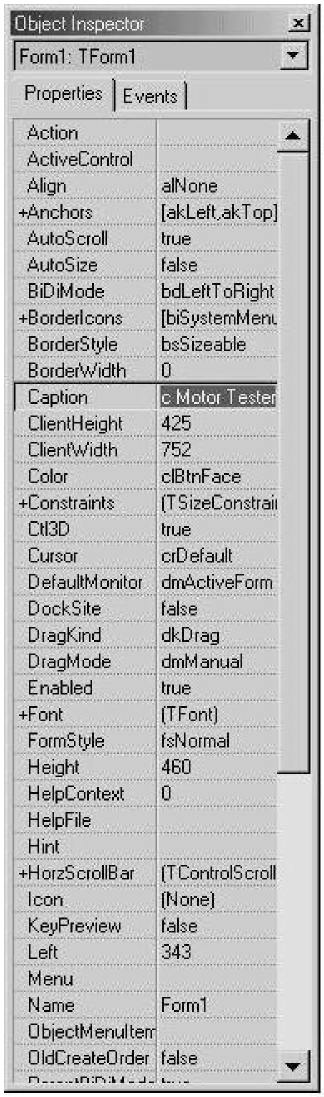

Figure 3.7 C++ Builder Object Inspector

3.2.3 Designed Software: "Dynamic Motor Tester"

Software designed for this system is called "Dynamic Motor Tester". This section aims to give a general layout about the system software.

Dynamic Motor Tester consists of some modules. These modules are as follows:

1) Load Side Parameters
2) Test Motor Parameters
3) Parameter Identification
4) Tests
5) Motor Characteristics
6) Reporting

These modules will be handled one by one and brief information is going to be given about each one. To have a very basic idea of the software, a simple flowchart of the software is given in *Figure 3.8*

3.2.3.1 Load Side Parameters

Design purpose of this module is to define the physical limitations and possible operating region of the DC machine used as the load. The setup parameters defined in this module are used in every stage whenever DC machine is driven by DC motor drive. This module is an input module and the inputs are shown in *Table 3.4.*

Maximum Armature Voltage
Maximum Armature Current
Maximum Field Current
Minimum Field Current
Maximum Speed
Torque / Current Ratio

Table 3.4   Inputs of "Load Side Parameters" module

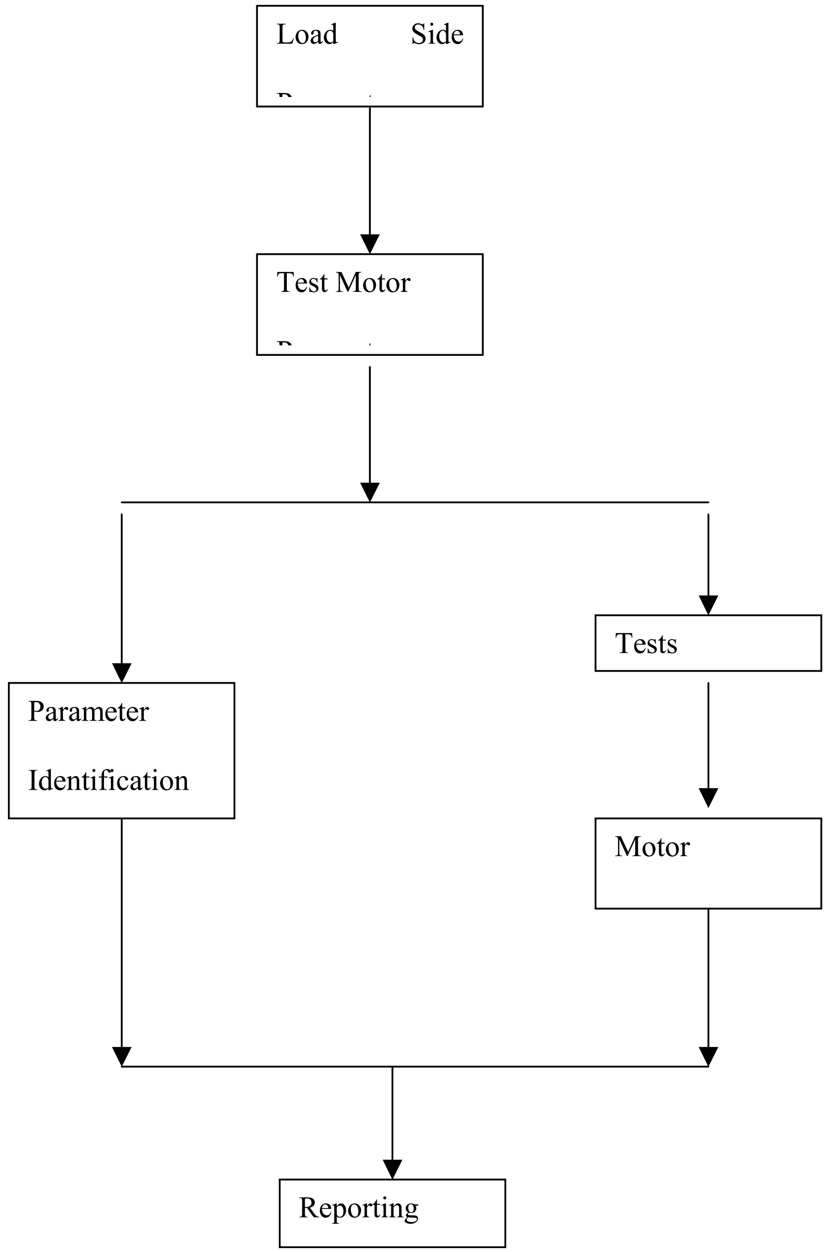

Figure 3.8 Simple Flowchart of "Dynamic Motor Tester" software

These parameters are entered only once and kept unchanged throughout the experiments. A “Notes” box is also introduced to give a possibility about extra information about the DC machine. The window layout of the Load Side Parameters is shown in *Figure 3.9.*

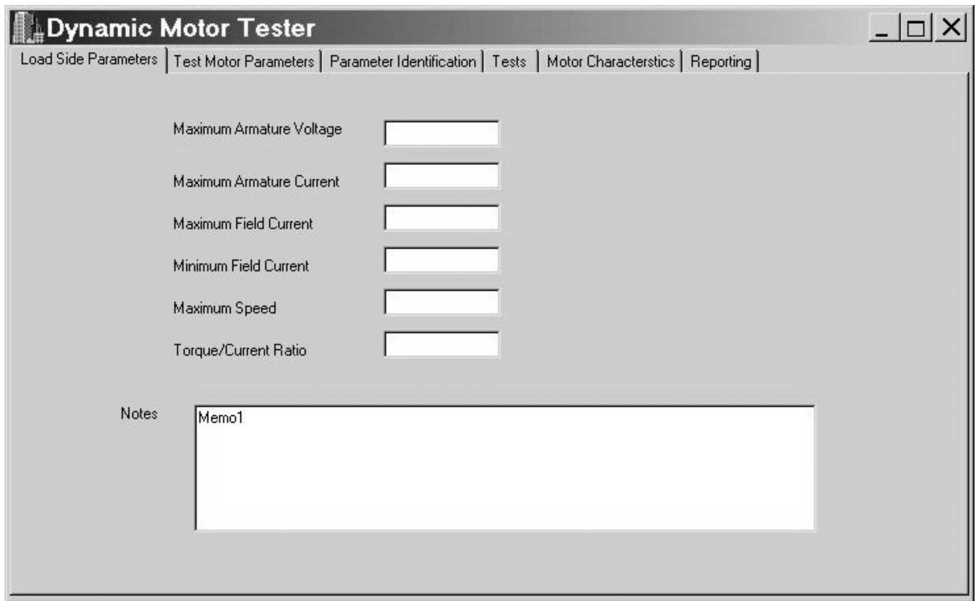

Figure 3.9   Window layout of the “Load Side Parameters” module

3.2.3.2 <u>Test Motor Parameters</u>

Dynamic Motor Tester software is an experimental motor test system user interface. Carrying out successful tests in the safety region requires an initial knowledge about the tested machine [5,6] . Test Motor Parameters module is designed for entering introductory nameplate data of tested machine.

This module is also an input module and its inputs are tabulated in Table 3.5.

Motor Title	Rated Line Voltage	HZ
Type	Rated Current	IP
kW	Serial No	Pole no
HP	Power Factor	Delta / Star
Rated Speed	Phase no	

Table 3.5 Inputs of "Test Motor Parameters" module

These values are used in the experiments when necessary. For instance, the synchronous speed of the test machine is calculated depending on these parameters. All nameplate data is used in the beginning of the reports for completeness.

It is common practice to test the same machine many times, so the Test Motor Parameters module has the option of saving nameplate data. Using "Save"; "Load" and "Save as.." Buttons user can save or reload these parameters. Window designed for Test Motor Parameters module is shown in Figure 3.10.

3.2.3.3 Parameter Identification

This module includes the commonly used tests for identification of parameters of linear circuit model for an induction machine. User can perform these tests easily by the help of this module. For simplicity, Parameter Identification module is divided into 3 sub-modules:

i. No Load Test

ii. Locked Rotor Test

iii. Parameters

The leading two sub-modules are special loading test with predefined conditions. Optional data, which can be entered to these modules, is very limited.

The last sub-module is simply output module for displaying calculated parameters.

Figure 3.10 Window layout of the "Test Motor Parameters" module

i. No Load Test

This sub-module has single input, namely "Test Voltage". No-Load Test is performed under this voltage value. This test has a predefined speed-time characteristic. Shaft speed is brought to the synchronous speed of test machine in 10 seconds. Speed control is completely done by the Load machine. Test machine is run at this speed for 30 seconds. "Test Voltage" is applied to the stator terminals of the test machine at the synchronous speed and electrical parameters of test machine are recorded to a file. Finally, shaft speed is again set to zero in

10seconds. Data recorded in No Load Test is used to calculate core loss resistance and magnetizing branch inductance. No Load Test sub-module window layout is given in Figure 3.11.

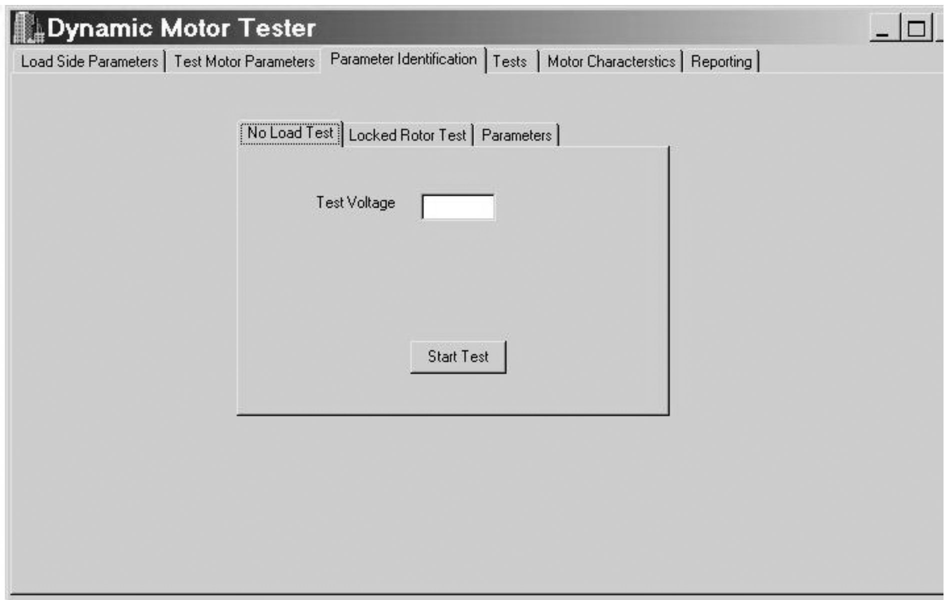

Figure 3.11 Window layout of the "No Load Test" sub-module

User can simply start the No Load Test with clicking the "Start Test" button. Test is performed automatically. Data acquisition frequency is 10 Hz at this test. All data received in No Load Test is saved to a file for calculating motor parameters.

ii. Locked Rotor Test

Limitation for the Locked Rotor Test is stator current of the test machine. User defines a "Maximum Current" for the stator current of the tested machine and this value is entered into the edit box present in the Locked Rotor Test sub-module. The experiment procedure is fixed and is performed as follows. Rotor of the induction machine is locked. Stator voltage is raised in steps up to the point where "Maximum Current" is reached. Tested machine is kept in this condition for 30 seconds and electrical parameters of the machine are stored in a file. Data recorded in Locked Rotor Test is used for calculating equivalent series resistance and equivalent leakage inductance. The window layout for Locked Rotor Test is given in Figure 3.12.

Locked Rotor Test sub-module is activated by single click on the "Start Test" button. Experiment is performed automatically. . Data acquisition frequency is 10 Hz at this test. Data received in Locked Rotor Test is saved to a file for calculating motor parameters.

iii. Parameters

Parameters sub-module is a simple output module for observing calculated parameters in the no load and locked rotor tests. Outputs of the Parameters sub-module are tabulated in Table 3.6. Outputs of this module are to be updated automatically. Window layout of this sub module is given in Figure 3.13.

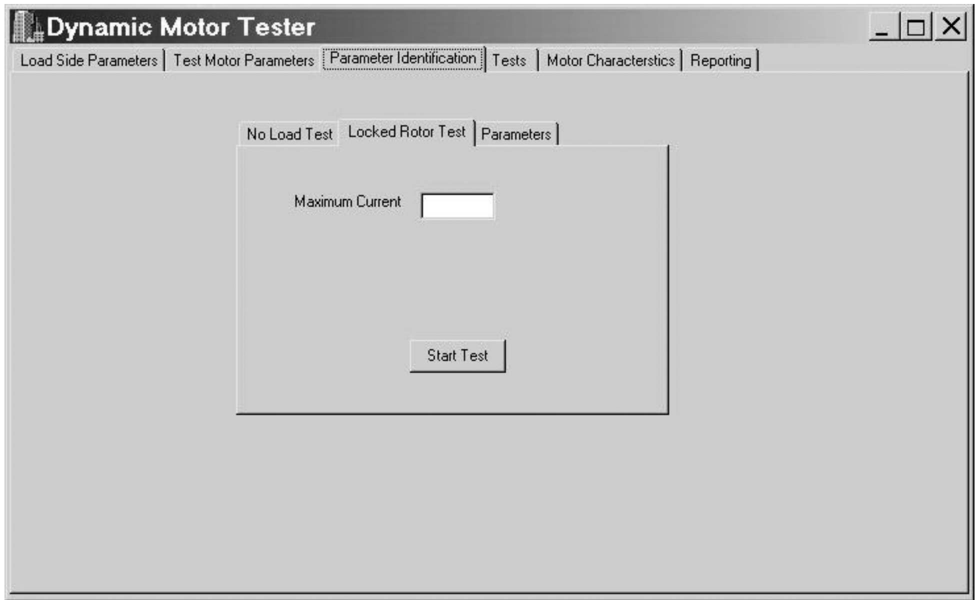

Figure 3.12 Window layout of the "Locked Rotor Test" sub-module

Core Loss Resistance
Magnetizing Branch Inductance
Equivalent Series Resistance
Equivalent Leakage Inductance

Table 3.6 Outputs of "Parameters" sub-module

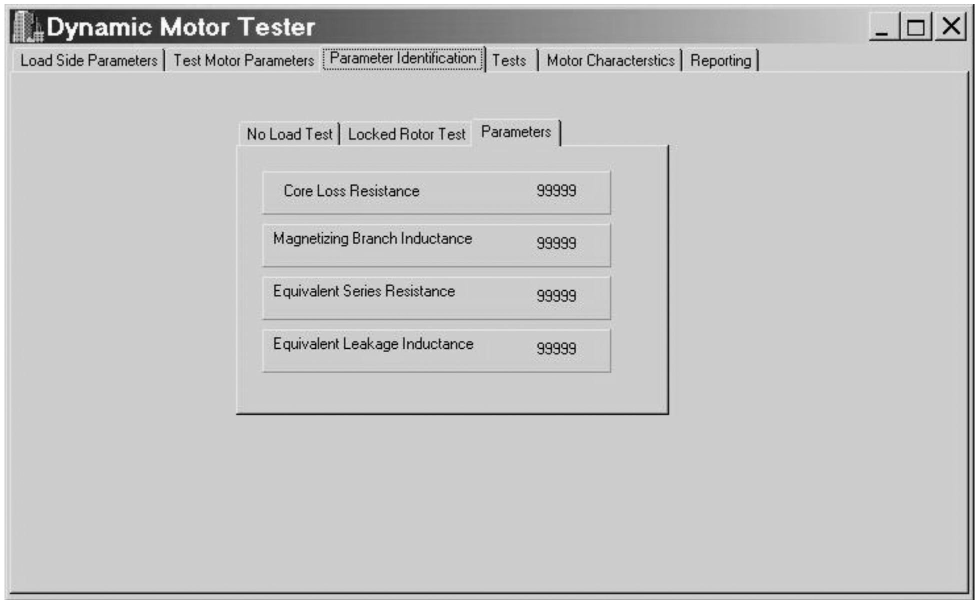

Figure 3.13 Window layout of the " Parameters "sub-module

3.2.3.4 Tests

Tests module is designed for giving the user opportunity of performing experiments with user-defined conditions. This module is separated into two sub-modules:

a. Automatic Loading
b. Manual Loading

Two sub-modules are the same in principle. If the experiment is done with the "Automatic Loading" sub-module, load machine drive controls the load automatically. In the "Manual Loading" sub-module however, user can set the load manually.

a.. Automatic Loading

In the Automatic Loading sub-module, there are two loading options. First, one is “Speed” controlled loading. The second is “Torque” controlled loading.

If “Speed” control is selected, user is expected to enter desired speed-time profile into the sub-module. Before starting the test, “Test Voltage” of the tested machine should be defined by user.

“Torque” control is the second option of loading. This time, user prepares a torque-time type load. Speed of the rotor in this test depends on the torque-speed characteristics of the tested machine.

Both control profiles can be plotted by “Plot” button before starting experiment. “Clear” button is present to clear all the data entered to this sub-module. Experiment time depends completely on the length of the load profile.

Finally, user clicks “Start Test” button to initialize the experiment. Load data entered to the screen is converted to appropriate format for the load machine drive and downloaded through RS232 to drive. Test machine experiences the defined load with “Test Voltage” applied to its terminals. All of the electrical and mechanical parameters (voltages, currents, real power input, torque, speed) are recorded to a file. . Data acquisition frequency is 20 Hz at this test. Recording length of file depends available space on the hard disc. However, too lengthy files can not be recalled in the graphic modules such as motor characteristics since available RAM in the PC is limited.

This file is saved as “Automatic Loading” file assigned to “Motor Title”. This file is recalled in the Motor Characteristics module. Window layout of the “Automatic Loading” sub-module is placed in Figure 3.14.

b.. Manual Loading

This sub-module is developed for satisfying the need of manual loading whenever needed. User selects either mode of loading (speed or torque). Then the load (speed or torque) can be with using precision potentiometer connected to Mentor II Drive used as load controller.

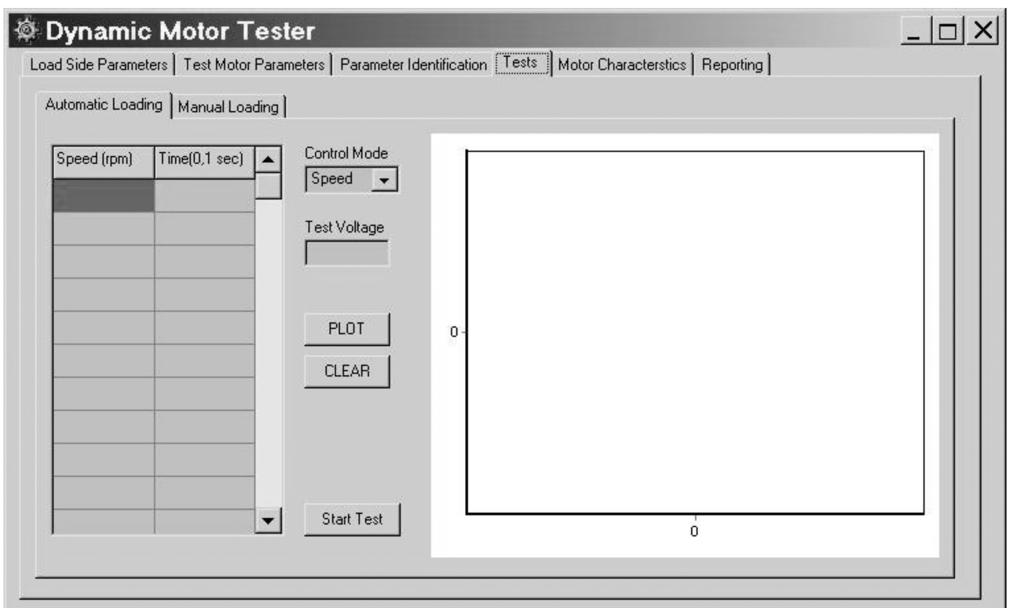

Figure 3.14 Window layout of the “ Automatic Loading ”sub-module

3.2.3.5 Motor Characteristics

This module is designed for observing motor characteristics. This module is mainly a graphical module. It can draw graphics of voltage, current, real power input, torque, speed, mechanical power output, power factor and efficiency with respect to any desired parameter. User selects “X-Axis” and “Y-Axis” parameters and clicks “Plot” and graphic is created. Default data source for this graphic is set as data file created by “Automatic Loading” sub-module. Window layout for the “Motor Characteristics” module is given in Figure 3.15.

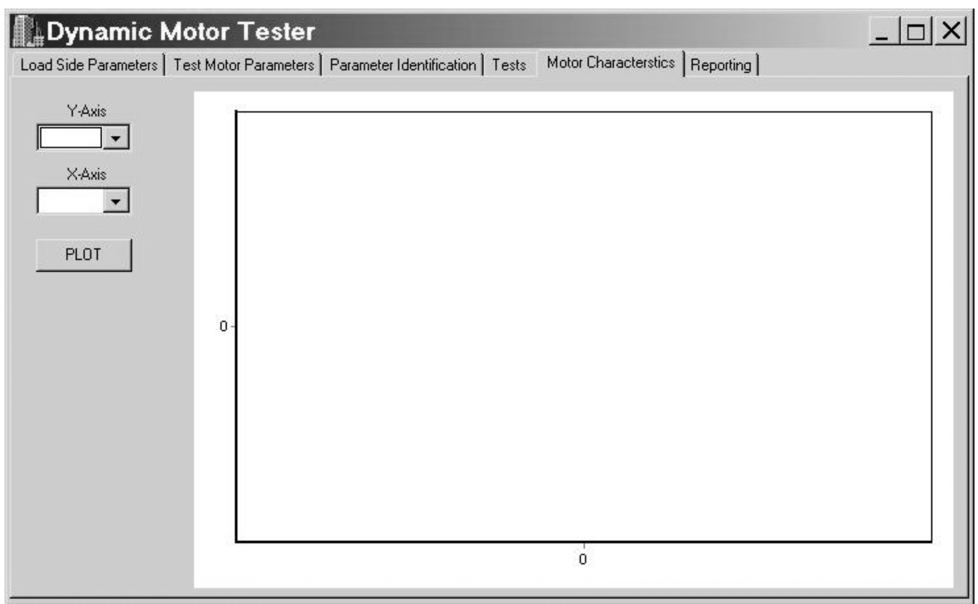

Figure 3.15   Window layout of the “ Motor Characteristics” module

Motor Characteristics module can be used to display torque-speed characteristic of the test machine. This characteristic shows the user starting torque of the test machine as well.

In this module, a curve fitting utility will be appropriate. This utility is thought as a future work and it can be as follows. User selects a curve type –such as polynomial, exponential etc—, then the user defines the parameters of curve fitting –degree of polynomial etc—,finally the user clicks a button and corresponding fitted curve is drawn on the original data. Constants of the fitted curve can also be displayed.

3.2.3.6 Reporting

This module is used for generating reports of the performed tests. These reports include data presented in table and graph formats. User can select any test performed and observe test results in screen in the form of tables and graphs.

In addition to screen display, reporting module can gather data from all performed tests and generate report file including motor specifications, test results and selected graphics.

CHAPTER 4

SYSTEM OPERATION AND SAMPLE TESTS

4.1 Introduction

In the second chapter general information about the system hardware was given, whereas the previous chapter introduced the software designed. The fourth chapter is about the successful operation of the system and performing sample tests.

Successful operation of the system requires starting up all of the system devices and running the system software. System start-up is important not only for successful system operation, but also for human safety. Dynamic Motor Tester includes both electrical and mechanical parts, all of which must be considered from the security perspective. Keeping this point in mind, in the first section of this chapter information about starting test environment will be given.

Second section of this chapter will describe the performing of sample tests and displaying the related results with Dynamic Motor Tester.

4.2 Starting – Up Test Environment

Hardware of Dynamic Motor Tester consists of many different parts. Therefore, before running the test software all related hardware must be turned on and must be brought to a state suitable for performing tests with Dynamic Motor Tester.

4.2.1 Running System Hardware

Hardware connections must be checked before power-up. Although these checks are straight forward, most of them are vital for proper operation and safety. These controls are summarized below and to remind the system, simplified block diagram of the system is given in Figure 4.1.

4.2.1.1 Mechanical Couplings

The very first thing is to check mechanical couplings of the system. These couplings are as follows:

i) Coupling between Load Machine & Torque-Speed Transducer

ii) Coupling between Torque-Speed Transducer & Test Machine

iii) Coupling between Test Machine & Tachogenerator

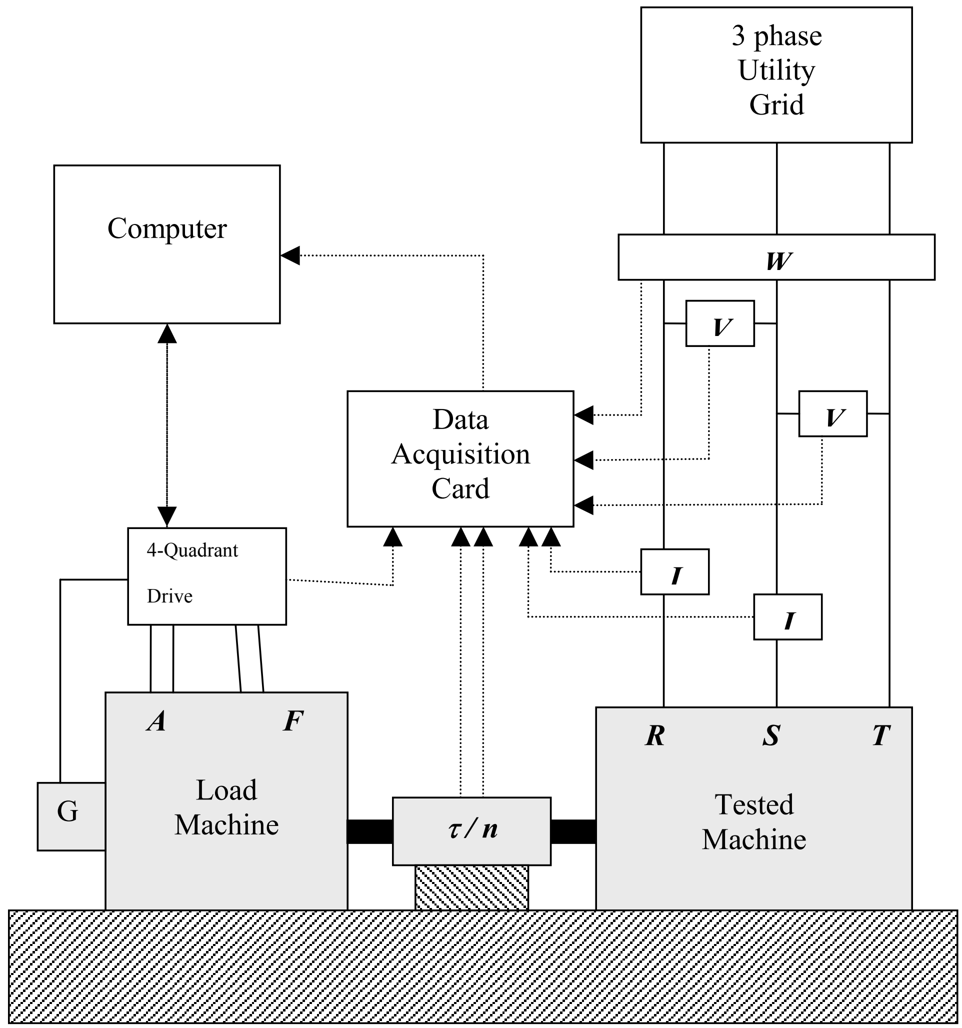

Figure 4.1 Simplified Block Diagram of the System

V : *Voltage Sensor*
I : *Current Sensor*
W : *Three-Phase Power Sensor*
τ / n : *Rotating Torque / Angular Speed Transducer*
A : *Armature Terminals of the Load Machine*
F : *Field Terminals of the Load Machine*
R ,S ,T : *Phase Terminals of the Tested Machine*
G **:***Tachogenerator*

4.2.1.2 <u>Electrical Connections</u>

Following the mechanical checkout, electrical connections of the system should be controlled. These connections are gathered in some groups (such as Power Supply Connections, Load Side Connections etc.) to follow the connections more easily.

Power Supply Connections

380V/50 Hz 3 Phase Supply

i) DC Motor Drive
ii) Test Motor Supply

220V/50 Hz 1 Phase Supply

i) Personal Computer
ii) Torque / Speed Transducer Amplifier
iii) Step-Down Transformer (220V/120V 70 VA)

120V/50 Hz 1 Phase Supply

i) Voltage Transducers
ii) Current Transducers
iii) Real Power Sensor

Load Side Connections

1) Field circuit (DC motor drive-DC machine)

2) Armature circuit (DC motor drive-DC machine)

3) Speed Feedback (DC motor drive-Tachogenerator)

4) Serial Connection (RS232 connection between PC and DC Motor Drive)

Test Motor Side Connections

1) Current paths (3 Phase Supply - Fuses - Current Transducers- Real Power Sensor-Induction machine)

2) Voltage measurement connections (Voltage Transducers-Real Power Sensor-Induction machine)

Torque / Speed Sensor Connections

1) Connection between Torque / Speed Sensor and Torque / Speed Transducer Amplifier

Data Acquisition Card Connections

1) Connection between Data Acquisition Card and Mainboard PCI slot

2) Analog Data Channel Connections

- i) Connection between Voltage Transducer 1 and Data Acquisition Card
- ii) Connection between Voltage Transducer 2 and Data Acquisition Card
- iii) Connection between Current Transducer 1 and Data Acquisition Card
- iv) Connection between Current Transducer 2 and Data Acquisition Card
- v) Connection between Real Power Sensor and Data Acquisition Card
- vi) Connection between Torque / Speed Transducer Amplifier and Data Acquisition Card (speed)
- vii) Connection between Torque / Speed Transducer Amplifier and Data Acquisition Card (torque)
- viii) Connection between DC Motor Drive and Data Acquisition Card (start–stop trigger)

4.2.1.3 Turning on the devices

After controlling all hardware connections, system devices can be turned on :

1) AC Voltage, Current and Power Transducers

2) Torque / Speed Transducer Amplifier

3) Load Controller

4) Personal Computer

Hardware of Dynamic Motor Tester is now ready for running the software and starting sample tests.

4.2.2 Running "Dynamic Motor Tester" Software

As mentioned in the third chapter, for data acquisition DasyLab commercial software is employed. In order to get the personal computer into a state waiting for data acquisition, *dasylab.exe* and *tetik.dsb* should be started.

Finally, system software "Dynamic Motor Tester" is activated and test setup is ready for performing tests.

4.3 Performing Sample Tests

Dynamic Motor Tester is now ready for performing sample tests. Initial conditions of the system are going to be introduced by the help of the Test Motor Parameters module. Afterwards sample tests are going to be performed. Finally, results obtained from these tests will be displayed as tables and graphics.

4.3.1 Loading Test Motor Parameters

The previously created motor information file Motor1.DMT is loaded. This file also includes the information required for the Load Side Parameters. Test Motor Parameters module loaded by this file is shown in Figure 4.2

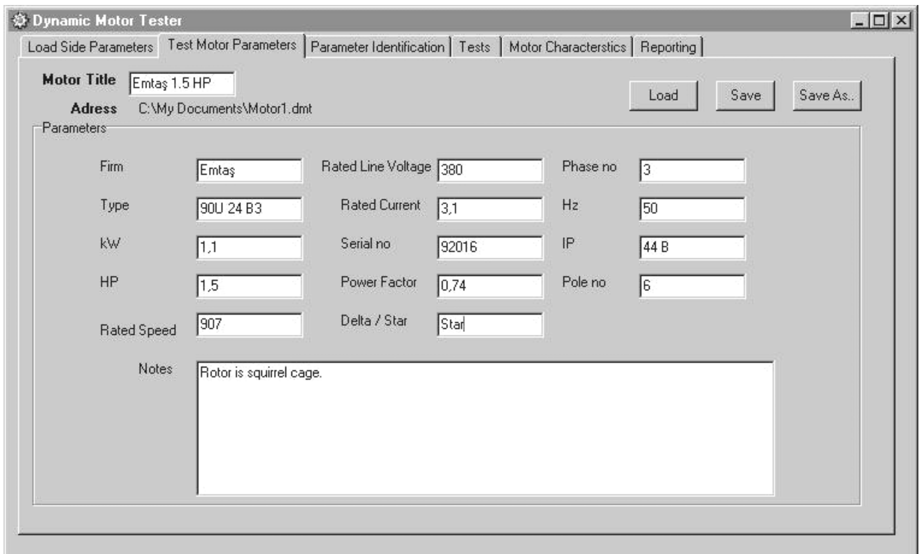

Figure 4.2 Test Motor Parameters Module Loaded

4.3.2 Parameter Identification

4.3.2.1 No Load Test

Performing No Load Test requires entering test voltage. Test voltage is entered in the related box. (Figure 4.3). After clicking "Start Test" button, no load test starts and test results are saved in a file named "noload.asc" in the same folder with Motor1.DMT.

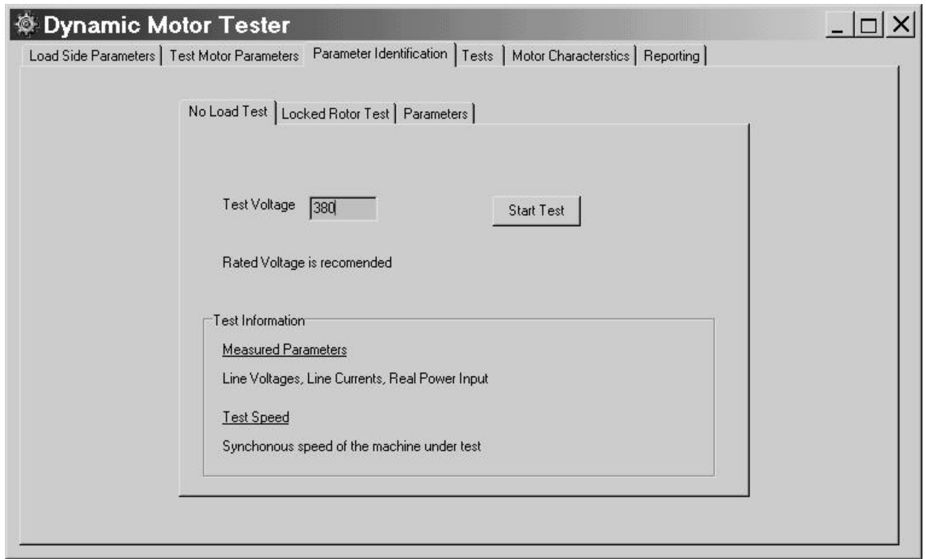

Figure 4.3 No Load Test Sub-Module Loaded

4.3.2.2 Locked Rotor Test

Performing Locked Rotor Test requires entering test current. Test current is entered in the related box. (Figure 4.4). After clicking "Start Test" button, no load test starts and test results are saved in a file named "locked.asc" in the same folder with Motor1.DMT.

4.3.2.3 Parameters

After performing No-Load Test and Locked Rotor Tests, Dynamic Motor Tester is ready for calculating motor parameters. By clicking "Calculate Parameters" button, results are displayed at parameters sub-module. Parameters sub-module with displayed results is shown in Figure 4.5, these results are also tabulated in Table 4.1.

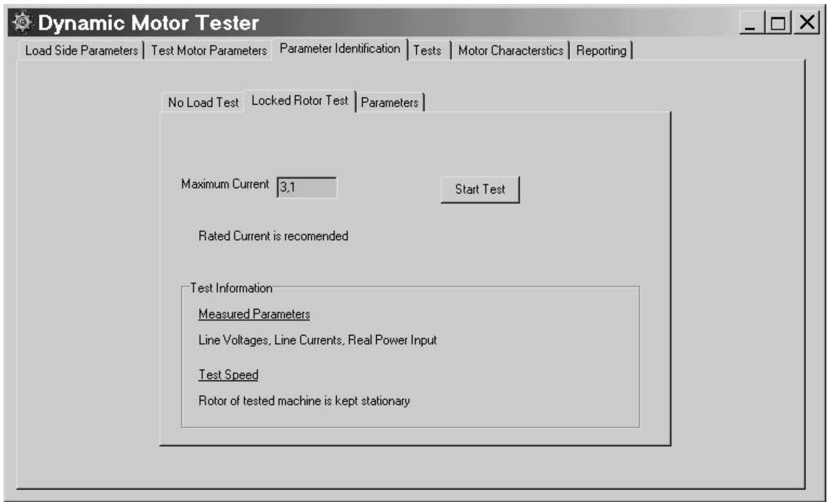

Figure 4.4 Locked Rotor Test Sub-Module Loaded

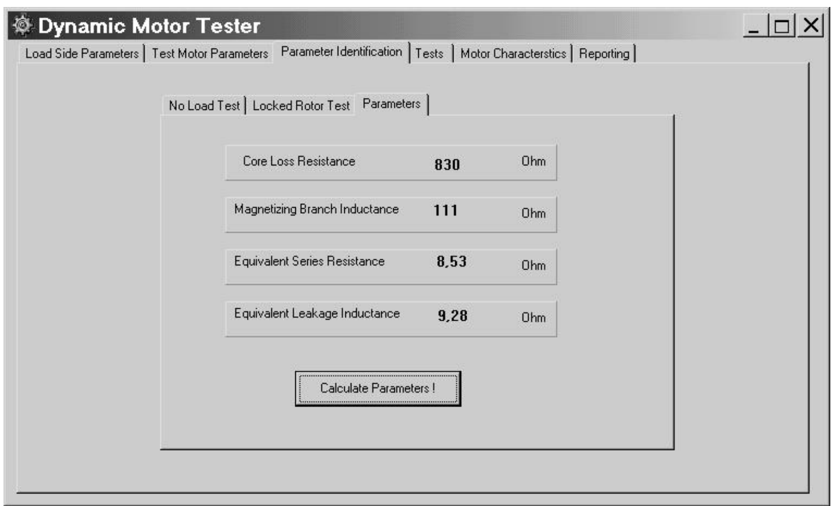

Figure 4.5 Parameters Sub-Module Loaded

Core Loss Resistance	830	Ohm
Magnetizing Branch Inductance	111	Ohm
Equivalent Series Resistance	8,53	Ohm
Equivalent Series Inductance	9,28	Ohm

Table 4.1 Calculated Parameters

4.3.3 Automatic Loading

In order to demonstrate an automatic loading example, following test is performed. Shaft speed is increased to synchronous speed of the test machine in three seconds. Shaft speed is then kept at synchronous speed for 7 seconds. Test voltage is going to be applied to the tested machine when it is running at synchronous speed. Afterwards, test machine is loaded down to zero speed in 20 seconds with constant deceleration. During all stages of the experiment, shaft speed is dictated by the DC motor drive used as load controller. Automatic Loading Window with conditions mentioned above is shown in Figure 4.6.

After completing the entries in Automatic Loading Window, "Start Test" button is clicked and automatic loading test starts. Test stops automatically at the end of the load profile. Data obtained throughout the experiment saved in the file "auto.asc". This file is used for drawing graphics and for reporting purpose later on.

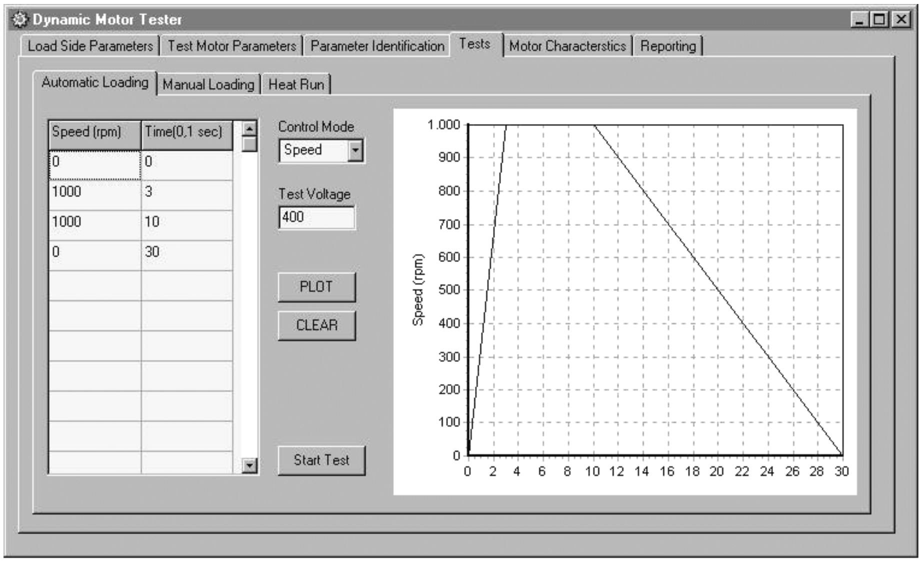

Figure 4.6 Automatic Loading Window Loaded

4.3.4 Motor Characteristics

Motor Characteristic module displays the graphics of the data obtained in the Automatic Loading Test. From the tested machine perspective, deceleration period (loading period) is more important. Data obtained in this period is tabulated in Table 4.2. Some rows of the data are skipped to prevent excessive space occupation.

Measured speed – time profile is displayed in Figure 4.7. It is easily observed that linear deceleration is achieved.

Characteristics of the motor are going to be displayed with respect to speed. Torque –speed characteristic is given in Figure 4.8.

Time[s]	V 1 (V)	V 2 (V)	I 1 (A)	I 2 (A)	Pin (W)	T(Nm)	n (rpm)	Pout (W)	Eff %	PF
9,3	403	406	2,29	2,1	460	2,61	980	268	58	0,30
9,65	404	407	2,42	2,2	915	2,4	985	248	27	0,56
10,3	404	405	2,33	2,22	656	3	975	306	47	0,41
10,65	406	407	2,83	2,6	1416	5,59	950	556	39	0,74
10,95	407	409	2,95	2,79	1326	7,85	941	773	58	0,65
11,3	407	408	3,1	3,07	1610	10,95	926	1061	66	0,74
12,2	403	403	4,32	4,19	2478	16,69	874	1528	62	0,83
12,6	402	403	4,83	4,76	2824	18,75	860	1687	60	0,84
13,2	401	403	5,39	5,32	3144	22,18	821	1905	61	0,84
13,85	401	403	6,14	5,74	3721	22,85	792	1893	51	0,90
14,2	405	403	6,21	6,13	3718	23,58	775	1913	51	0,86
14,75	404	402	6,71	6,63	4173	24,64	743	1917	46	0,90
15,1	408	402	7,06	6,98	4082	25,5	742	1982	49	0,83
15,65	402	401	7,41	7,33	4546	26,07	696	1899	42	0,89
16	402	401	7,64	7,53	4481	26,57	698	1942	43	0,85
16,6	404	404	8,05	7,97	4908	26,88	655	1843	38	0,88
17,2	404	399	8,55	8,28	4853	27,29	631	1801	37	0,83
17,55	401	407	8,79	8,77	5128	27,23	604	1721	34	0,83
18,15	400	408	8,87	8,84	5175	27,39	585	1676	32	0,84
18,5	400	404	9,16	9,04	5506	27,37	559	1600	29	0,87
19,1	400	403	9,34	9,07	5710	27,47	539	1550	27	0,89
19,45	402	401	9,45	9,93	5616	27,52	541	1560	28	0,83
20,1	401	404	9,56	9,69	5899	27,3	516	1474	25	0,88
20,5	399	401	9,93	9,93	5836	26,97	461	1302	22	0,85
21,1	399	401	10,03	10,13	5651	26,81	417	1170	21	0,81
21,45	399	401	10,15	10,13	5532	26,75	476	1331	24	0,79
22	399	401	10,29	10,18	5938	26,46	386	1069	18	0,84
22,35	400	399	10,43	10,31	5545	26,16	366	1003	18	0,77
23,05	404	407	10,9	10,77	6103	25,86	333	902	15	0,80
23,6	408	409	11,1	10,96	5734	25,76	303	818	14	0,74
24,7	406	408	11,41	11,29	6041	25,67	318	940	16	0,75
26	407	400	11,56	11,36	5867	24,91	267	697	12	0,73
26,9	399	401	11,33	11,57	5844	24,47	139	356	6	0,74
28,2	399	401	11,51	11,4	5635	24,21	92	233	4	0,71
28,55	399	401	11,55	11,45	5652	24,3	61	155	3	0,71
29,1	399	401	11,69	11,56	5915	23,84	35	87	1	0,74

Table 4.2 Data obtained at Automatic Loading Test

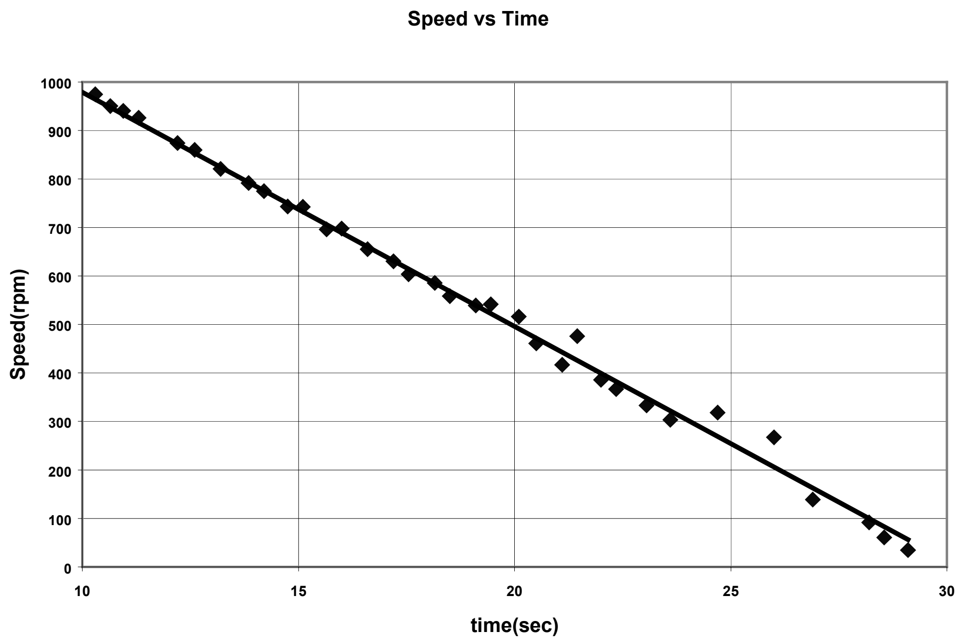

Figure 4.7 Measured Speed-Time Profile

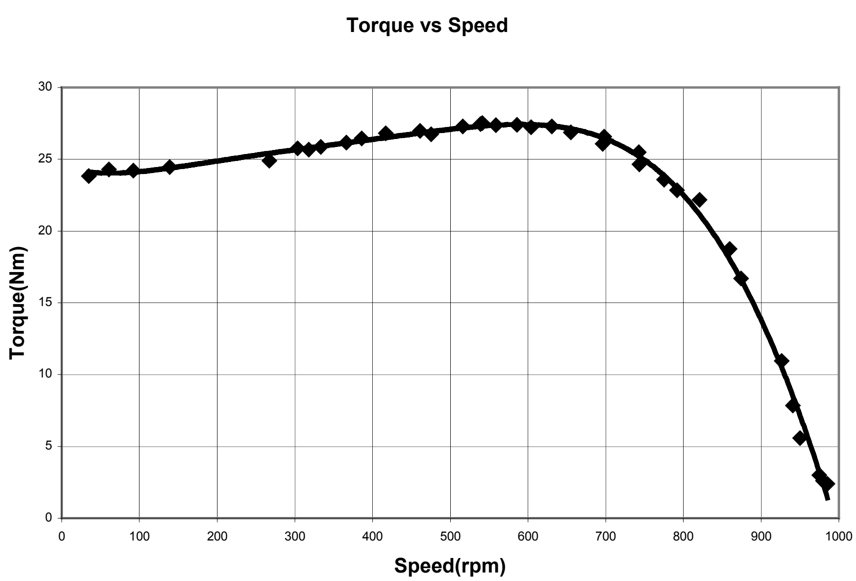

Figure 4.8 Torque—Speed Characteristic

Furthermore, according to the test results in Table 4.2, following characteristics are also displayed:

Current vs Speed ………………………………Figure 4.9

Current Balance………………………………. Figure 4.10

Input Power vs Speed……………………….… Figure 4.11

Output Power vs Speed………………………... Figure 4.12

Efficiency vs Speed……………………………. Figure 4.13

Power Factor vs Speed…………………………. Figure 4.14

Detailed discussion of these characteristics is out of the scope of this thesis. However, it should be noted that they are consistent with the well-known behavior of the induction machines.

At this chapter, basic information about the operation of “Dynamic Motor Tester” is given. Sample experiments with the motor system are done and finally the results obtained in these sample tests are displayed. These results verify that test system designed for this study is capable of performing the tests previously mentioned.

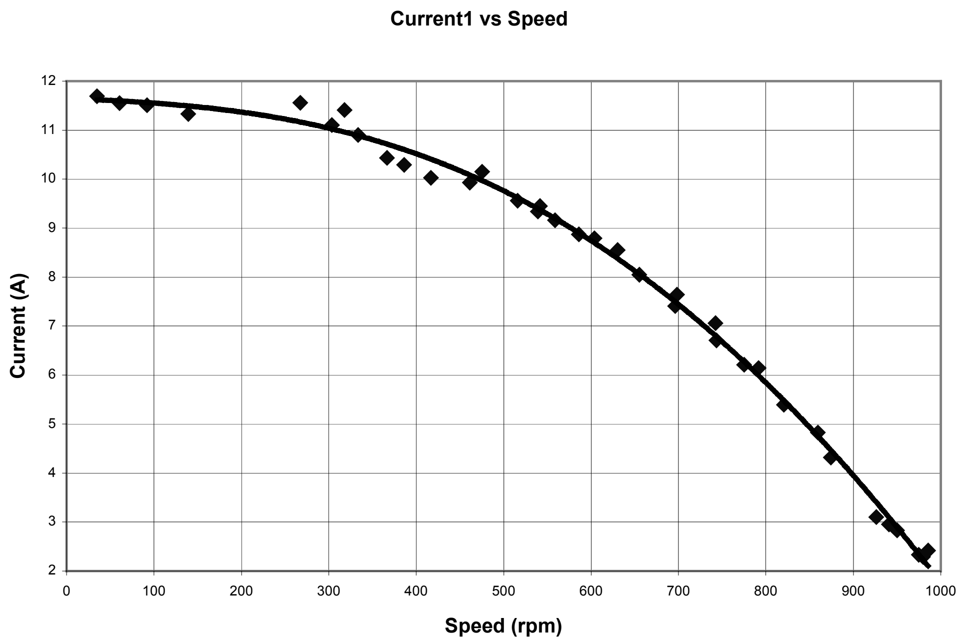

Figure 4.9 Current vs Speed

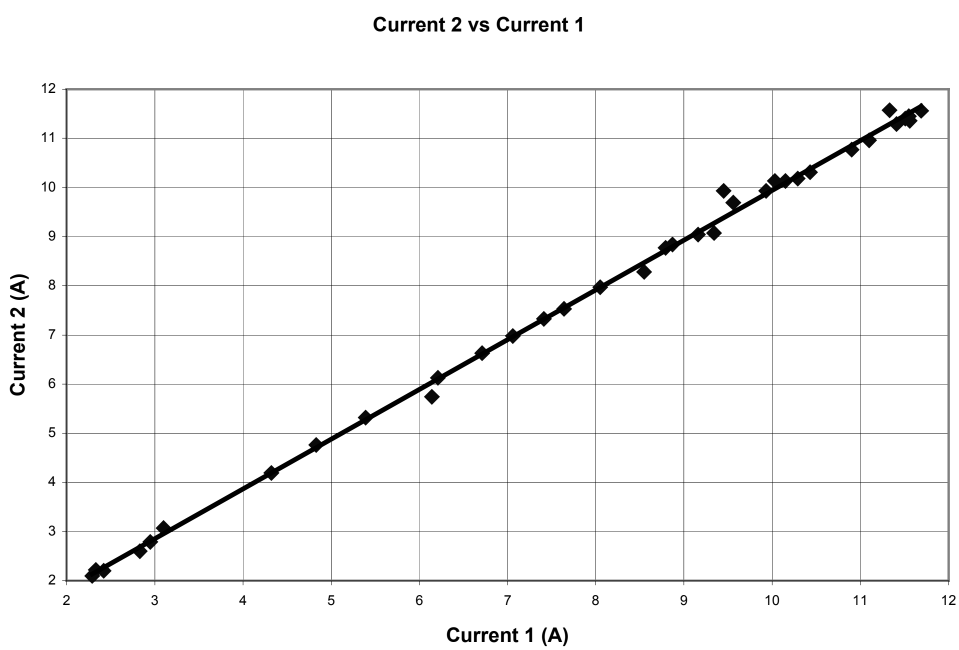

Figure 4.10 Current Balance

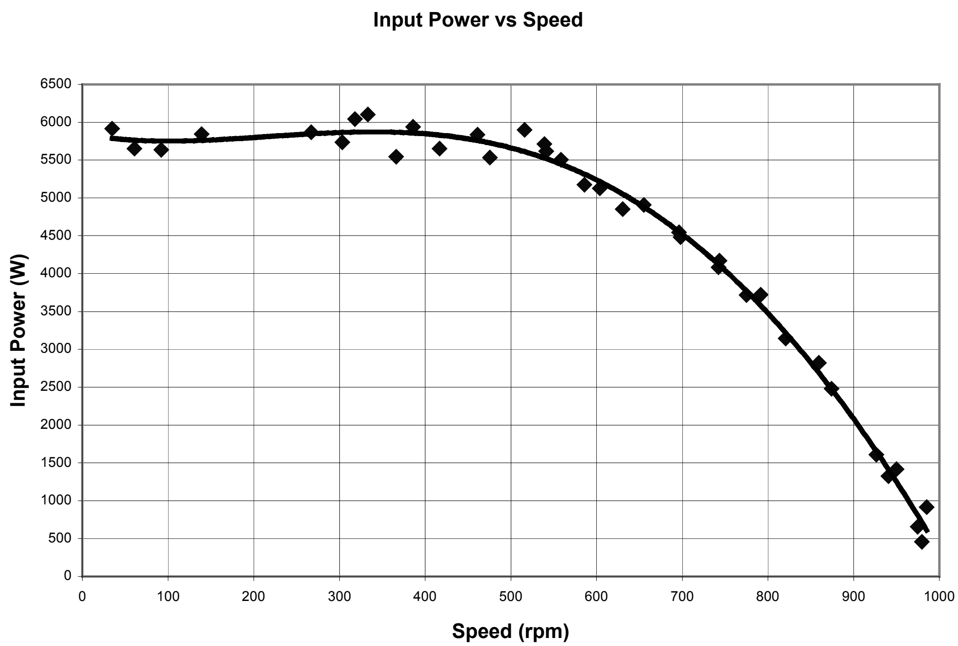

Figure 4.11 Input Power vs Speed

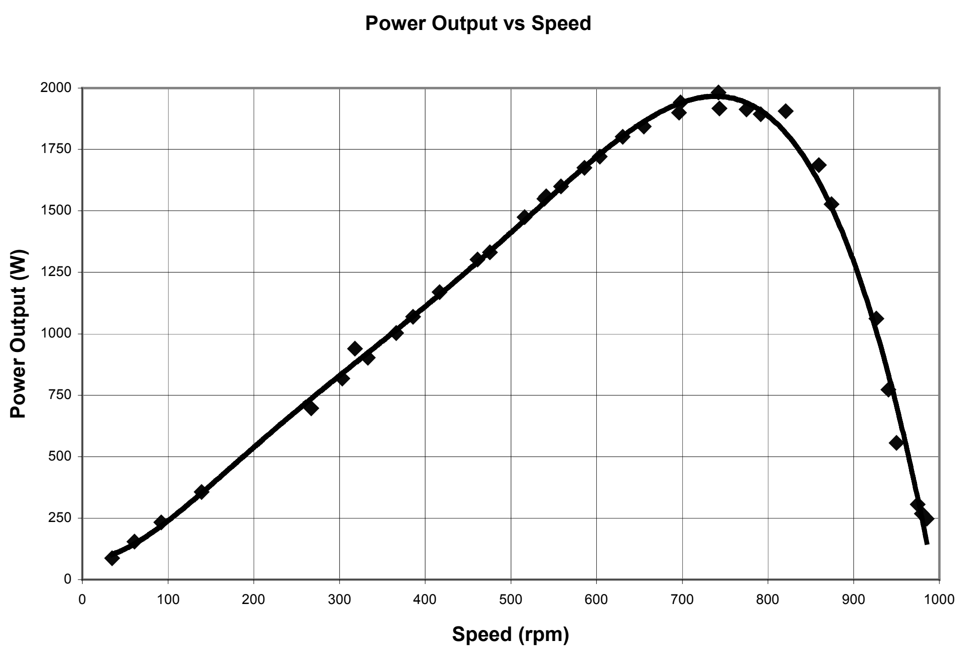

Figure 4.12 Output Power vs Speed

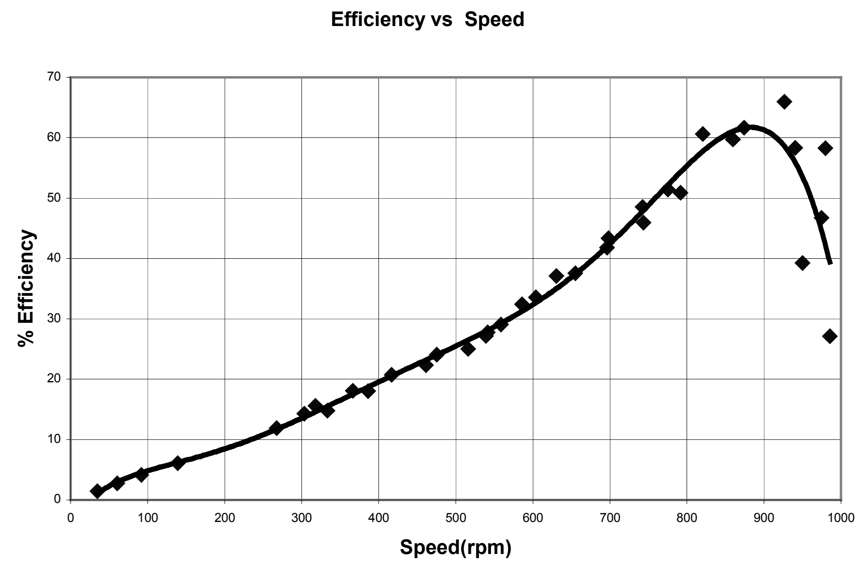

Figure 4.13 Efficiency vs Speed

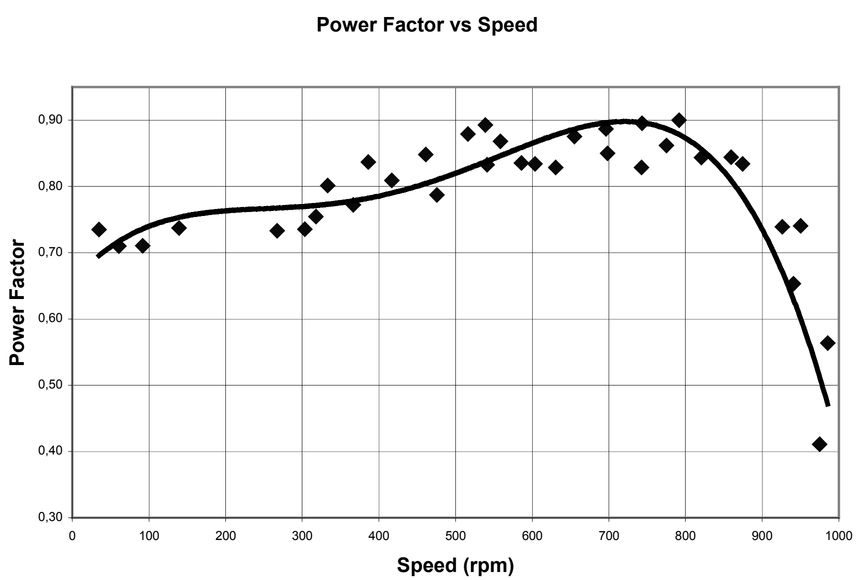

Figure 4.14 Power Factor vs Speed

CHAPTER 5

Throughout this study a computer based motor test system is developed. This system is equipped with electrical and mechanical sensors, data acquisition hardware and load conditioner. The motor test system employs a four-quadrant DC machine drive as a load controller. DC machine drive has a closed-loop speed control. This gives the system the ability of performing tests at the regions of the torque-speed plane where the test machine is naturally astable. When the test machine is loaded by the load machine, the DC machine drive operates in the regenerative region. Therefore, most of the electrical energy consumed by the test machine is supplied back to the utility grid. As a result, the overall test system efficiency is high.

Major contribution of this study is the designed software. The software is capable of load conditioning, performing tests, acquiring data and displaying results. All these actions can be performed with a single visual user-interface. This interface is designed with C++ Builder programming language. This programming language is chosen in purpose, since it contains commonly known built-in objects such as edit-boxes, combo-boxes, buttons etc. Software is separated into modules. Every module corresponds to a phase of the test procedure. The user faces different modules for different tasks. This creates a user-friendly environment for the user.

An important feature of the system software is easy load conditioning. A commercial motor drive usually requires configuring many setup parameters for single user-defined test. Designed software requires only the nameplate data for the test machine and desired load conditions. Having these inputs completed, the user can perform such loading tests with Automatic Loading sub-module easily. Another advantage of the software is performing data acquisition without difficulty. Acquisition of data starts simultaneously with the load conditioning. All received data is saved automatically to a file assigned to the test machine. This file can be recalled later for graphic generation and reporting purposes. No extra effort is required for this acquisition procedure.

The motor test system operates successfully with the current configuration, and the following issues can be done as future work to improve the system capabilities. Currently, system measures total series resistance in the locked rotor test. Therefore, it measures summation of stator resistance and referred rotor resistance. A dc resistance test can be added to system set-up to measure stator resistance independently and automatically. Then, stator resistance and referred rotor resistance can be measured and displayed separately. Another improvement can be made by upgrading data acquisition hardware. Motor Test System equipped with faster transducers and a better data acquisition card can give much better performance.

REFERENCES

[1] P. G. Cummings, W. D. Bowers, W. J. Martiny, “Induction Motor Efficiency Test Methods” , IEEE Transactions on industry Applications, Vol. 1A-17, No.3, May/June 1981, 253-272.

[2] A. T. de Almeida, F. J. T. E. Ferreira, J.F. Busch, P. Angers, “Comparative Analysis of IEEE 112-B and IEC 34-2 Efficiency Testing Standards Using Stray Load Losses in Low- Voltage Three-Phase, Cage Induction Motors”, IEEE Transactions on Industry Applications, Vol. 38, No. 2, March/April 2002, 608-614.

[3] T. Crews, U. Ziegler, “The Flowchart Interpreter for Introductory Programming Courses”, Department of Computer Science Western Kentucky University

[4] N. Ertugrul, “New Era in Engineering Experiments: an Integrated and Interactive Teaching/Learning Approach, and Real-Time Visualizations”, Int. J. Engng. Ed. Vol. 14, No. 5, 1998, pp. 344-355.

[5] Vogelsang & Benning Prozessdatentechnik Motor Test System Technical Documentation

[6] Dr. Staiger, Mohilo + Co GmbH, Motor Test-Rig Specifications, Schondorf

[7] M. E. El-Hawary, “Power Engineering Letters”, IEEE Power Engineering Review, July 2002, 35-37

[8] M. Ermiş, “Lecture Notes of Utilization of Electrical Energy Course”, Spring 2001, 8-13

[9] Digitorque® MTS-3800 Motor Test System, *2002 ,*Automation Technology Inc.

[10] Motor Testing Equipment,1995–2002, Magtrol, Inc.

[11] U. Tamm , P. Rang , Basics of Electrical Measurement, Tallinn Technical University, 2003

[12] B. Boley, Overview of Motor Types Tutorial , 1997

[13] IEEE Standard Test Procedure for Polyphase Induction Motors and Generators, IEEE Std 112-1996

[14] İ. Karagülle, Z. Pala, “C++ Builder 4.0”, Türkmen Yayınevi , 2001

[15] A. E. Fitzgerald, C. Kingsley Jr., S. D. Umans, “Electric Machinery”, Fifth Edition, Schaum Outline Series, 1990

[16] PC-LabCard Lab & Engineering Add-on’s for PC/XT/AT User’s Manual, 1993, by Advantech Co., Ltd.

[17] User’s Guide for Mentor II DC Drives 25 A to 1850 A output, 1994

[18] DASYLab Data Acquisition System Laboratory User Guide, 1993-1998 by DATALOG, Germany

[19] DPL Drive Programming Language Toolkit User Manual, November 1997, Control Techniques Drives Ltd.

[20] J. Corcoran, S. D'Ercole ,A New Development In Continuous Torque Monitoring Couplings, Proceedings of ASME DETC 2000 Power Transmission and Gearing Conference September 10-13, 2000 Baltimore, Maryland

APPENDIX

PART A
VIEWS FROM THE SYSTEM HARDWARE

Figure A-1   Load Controller—4 Quadrant DC Machine Drive

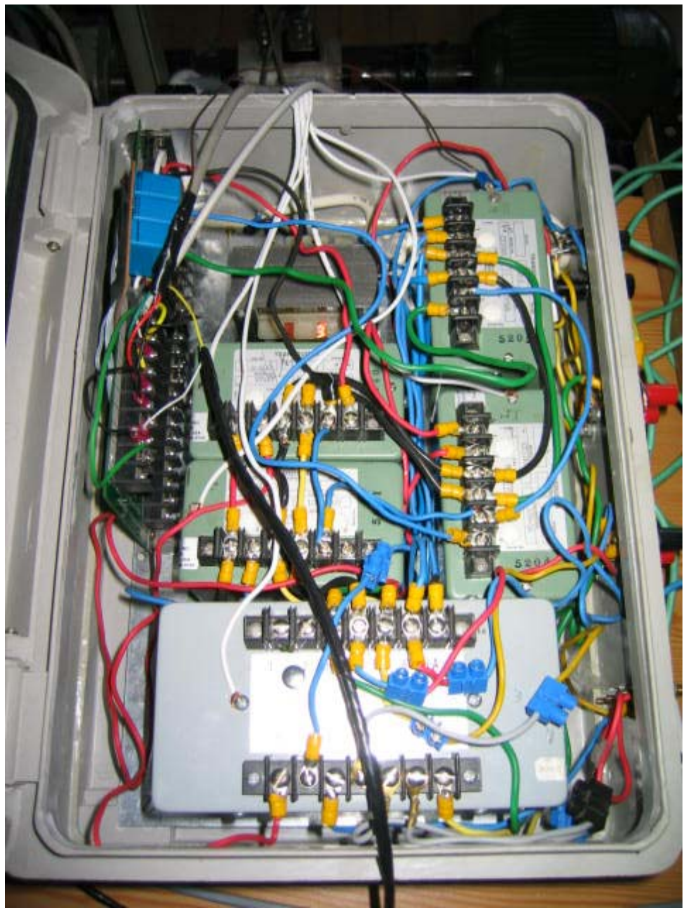

Figure A-2 Voltage, Current and Power Transducers

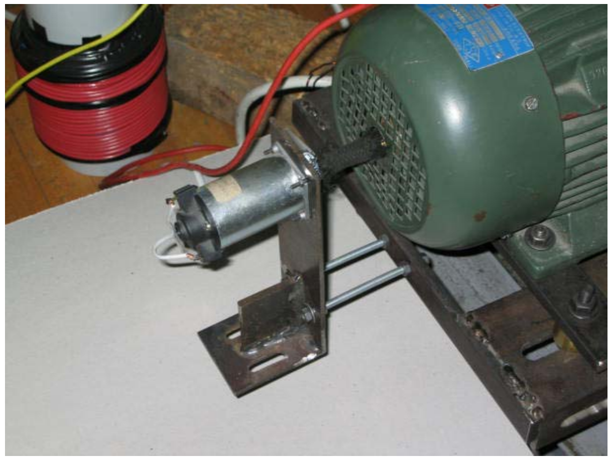

Figure A-3 Tachogenerator coupled to test machine

Figure A-4 Test Machine

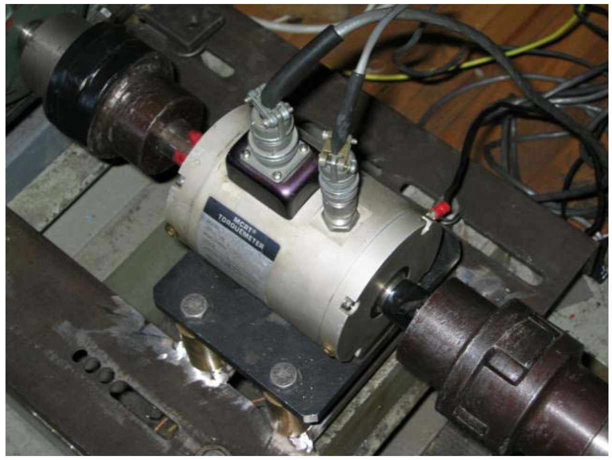

Figure A-5 Torque-Speed Transducer

Figure A-6 Load Machine

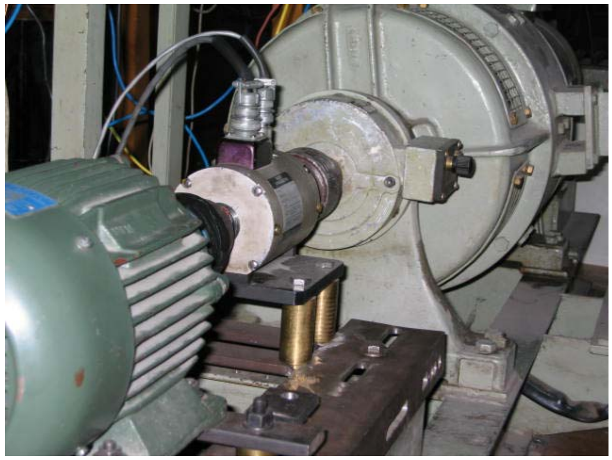

Figure A-7 Test Machine – Torque/Speed Transducer –Load Machine

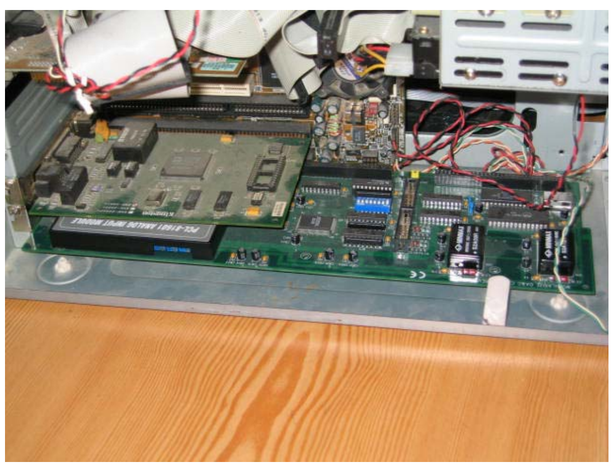

Figure A-8 Data Acquisition Card in the main board

PART B
DESIGNED SOFTWARE CODE

In the name of completeness, full program code of the “Dynamic Motor Tester” is given below.

Unit1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <math.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//variable declerations
int i,j,k,temp;
int Found_DAQ_Rate_At_Line;
int Found_Data_Save_File_At_Line;
int y1,y2,time1,time2,delay;
int acc_time1,acc_time2;
int skip_row_count;
int skip_row_count1;
int skip_row_count2;
int calculation_row_count;
int col_count;
int total_col_count;
int test_data[2][25];
//Second dimension of test_data (ie 25 for the time being)
//is taken from the active Row Count of  Test_Data_StringGrid
//it MUST be updated accordingly
int counter;
int row_number;
int col_number_x;
int col_number_y;
int current_col_number;

int No_Load_Start_Voltage;
int No_Load_Voltage_Increment;
int No_Load_Number_Of_Increments;


//declerations of parameters
float x,y;
float x_data,y_data;
float Maximum_Armature_Voltage;
float Maximum_Armature_Current;
float Maximum_Field_Current;
float Minimum_Field_Current;
float Maximum_RPM;
float Torque_Armature_Current_Ratio;
float synchronous_speed_of_tested_ac_machine_in_rpm;
float dTq_dt,dTq,Tq1,Tq2,t;
float Armature_Current_Reference_Armature_Current_Ratio;

float motor_voltage_at_no_load_test;
float motor_voltage_at_locked_rotor_test;
```

```
float motor_voltage_at_auto_load_test;

float a;
float b;



long double power_per_phase;
long double power_per_phase_set[20];

long double V_phase_to_neutral_average;
long double V_phase_to_neutral_average_set[20];

long double I_phase_average;
long double I_phase_average_set[20];

long double speed_avg;
long double speed_set[20];

long double iron_loss_resistance;
long double iron_loss_conductance;



long double admittance_Y;
long double magnetizing_branch_inductance_in_ohm;
long double magnetizing_branch_conductance_in_mho;
long double impedance_Z;
long double equivalent_series_resistance;
long double equivalent_leakage_inductance;

long double cumulative_power_per_phase;
long double cumulative_V_phase_to_neutral_average;
long double cumulative_I_phase_average;
long double cumulative_speed;

long double cumulative_iron_loss_resistance;
long double cumulative_magnetizing_branch_inductance_in_ohm;
long double cumulative_equivalent_series_resistance;
long double cumulative_equivalent_leakage_inductance;


long double temp1;
long double temp2;
long double data_set_param[11];


bool last_dialog_is_open_dialog;
bool last_dialog_is_save_dialog;
bool data_invalid_x,data_invalid_y;
bool x_new,y_new;

char dizi[20];
char dizi_plot[20];


AnsiString str1,str2,str3,str4,str5,str6,str7;
AnsiString DAQ_File_Name;
AnsiString DAQ_Rate;
AnsiString DAQ_Current_Data_Save_File;
AnsiString Parameter_Identification_Current_Data_Read_File;
AnsiString Motor_Characteristics_Current_Data_Read_File;

FILE *dosya;
FILE *file_read;
FILE *file_read_plot;


//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
```

```
{

//start trigerring  data acquisition
//WinExec("c:\\progra~1\\dasyla~1.0\\dasylab.exe c:\\data\\tetik.dsb",SW_SHOWMINIMIZED);


//Initial Settings
AC_Motor_Parameters_Motor_Title_Edit->Text="noname";
AC_Motor_Parameters_SaveDialog->DefaultExt="dmt";
AC_Motor_Parameters_OpenDialog->DefaultExt="dmt";
AC_Motor_Memo->Lines->Strings[0]="";
DC_Motor_Memo->Lines->Strings[0]="";
AC_DC_Motor_Memo->Lines->Strings[0]="";
DAQ_File_Update_RichEdit->Text="";

//****to be updated******
//****to be updated******
Armature_Current_Reference_Armature_Current_Ratio=10;
//****to be updated******
//****to be updated******


DAQ_File_Update_RichEdit->PlainText=true;
last_dialog_is_open_dialog=false;
last_dialog_is_save_dialog=false;


//Test_Data_StringGrid labeled
Test_Data_StringGrid->Cells[0][0]="Speed (rpm)";
Test_Data_StringGrid->Cells[0][1]="0";
Test_Data_StringGrid->Cells[0][2]="1000";
Test_Data_StringGrid->Cells[0][3]="1000";
Test_Data_StringGrid->Cells[0][4]="0";




Test_Data_StringGrid->Cells[1][0]="Time(sec)";
Test_Data_StringGrid->Cells[1][1]="0";
Test_Data_StringGrid->Cells[1][2]="5";
Test_Data_StringGrid->Cells[1][3]="7";
Test_Data_StringGrid->Cells[1][4]="15";

//No_Load_Test_Results_StringGrid labeled

No_Load_Test_Results_StringGrid->Cells[0][0]=" V (volts)";
No_Load_Test_Results_StringGrid->Cells[1][0]=" I (amps)";
No_Load_Test_Results_StringGrid->Cells[2][0]=" P (watts)";
No_Load_Test_Results_StringGrid->Cells[3][0]=" V square ";
No_Load_Test_Results_StringGrid->Cells[4][0]=" n (rpm)";

//initially speed control selected
Control_Mode_ComboBox->ItemIndex=0;

//select "Time" as X parameter
X_Parameter_ComboBox->ItemIndex=0;

//select "V1" as Y parameter
Y_Parameter_ComboBox->ItemIndex=1;

a=1.9;
b=-291;



}
//---------------------------------------------------------------------------
void __fastcall TForm1::AC_DC_Motor_Memo_Update_ButtonClick(
      TObject *Sender)
{
AC_DC_Motor_Memo->Text="";
```

```
for (i=0;i<=(20);i++)
AC_DC_Motor_Memo->Lines->Insert(i,"");

//DC motor parameters saved to memo
AC_DC_Motor_Memo->Lines->Strings[0]=DC_Motor_Maximum_Armature_Voltage_Edit->Text;
AC_DC_Motor_Memo->Lines->Strings[1]=DC_Motor_Maximum_Armature_Current_Edit->Text;
AC_DC_Motor_Memo->Lines->Strings[2]=DC_Motor_Maximum_Field_Current_Edit->Text;
AC_DC_Motor_Memo->Lines->Strings[3]=DC_Motor_Minimum_Field_Current_Edit->Text;
AC_DC_Motor_Memo->Lines->Strings[4]=DC_Motor_Maximum_Speed_Edit->Text;
AC_DC_Motor_Memo->Lines->Strings[5]=DC_Motor_Torque_Current_Ratio_Edit->Text;
//AC motor parameters saved to memo
AC_DC_Motor_Memo->Lines->Strings[6]=AC_Motor_Parameters_Motor_Title_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[7]=AC_Motor_Parameters_Firm_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[8]=AC_Motor_Parameters_Type_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[9]=AC_Motor_Parameters_kW_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[10]=AC_Motor_Parameters_HP_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[11]=AC_Motor_Parameters_Rated_Speed_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[12]=AC_Motor_Parameters_Rated_Current_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[13]=AC_Motor_Parameters_Rated_Voltage_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[14]=AC_Motor_Parameters_Serial_No_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[15]=AC_Motor_Parameters_Power_Factor_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[16]=AC_Motor_Parameters_Phase_no_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[17]=AC_Motor_Parameters_Hz_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[18]=AC_Motor_Parameters_IP_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[19]=AC_Motor_Parameters_Pole_no_Edit-> Text;
AC_DC_Motor_Memo->Lines->Strings[20]=" ";

//AC DC motor memo are saved
str1="";
str2="";
str3="";
str4="";
str5="";
str6="";
str7="";
str1=AC_DC_Motor_Memo->Text;
str2="b-r-e-a-k-1";
str3=DC_Motor_Memo->Text;
str4="b-r-e-a-k-2";
str5=AC_Motor_Memo->Text;
str6="b-r-e-a-k-3";
str7=str1+str2+str3+str4+str5+str6;
AC_DC_Motor_Memo->Text=str7;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::AC_DC_Motor_Parameters_Update_ButtonClick(
      TObject *Sender)
{
//DC motor parameters reloaded from memo
DC_Motor_Maximum_Armature_Voltage_Edit->Text=AC_DC_Motor_Memo->Lines->Strings[0];
DC_Motor_Maximum_Armature_Current_Edit->Text=AC_DC_Motor_Memo->Lines->Strings[1];
DC_Motor_Maximum_Field_Current_Edit->Text=AC_DC_Motor_Memo->Lines->Strings[2];
DC_Motor_Minimum_Field_Current_Edit->Text=AC_DC_Motor_Memo->Lines->Strings[3];
DC_Motor_Maximum_Speed_Edit->Text=AC_DC_Motor_Memo->Lines->Strings[4];
DC_Motor_Torque_Current_Ratio_Edit->Text=AC_DC_Motor_Memo->Lines->Strings[5];
//AC motor parameters reloaded from memo
AC_Motor_Parameters_Motor_Title_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[6];
AC_Motor_Parameters_Firm_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[7];
AC_Motor_Parameters_Type_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[8];
AC_Motor_Parameters_kW_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[9];
AC_Motor_Parameters_HP_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[10];
AC_Motor_Parameters_Rated_Speed_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[11];
AC_Motor_Parameters_Rated_Voltage_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[12];
AC_Motor_Parameters_Rated_Current_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[13];
AC_Motor_Parameters_Serial_No_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[14];
AC_Motor_Parameters_Power_Factor_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[15];
AC_Motor_Parameters_Phase_no_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[16];
AC_Motor_Parameters_Hz_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[17];
AC_Motor_Parameters_IP_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[18];
```

```
AC_Motor_Parameters_Pole_no_Edit-> Text=AC_DC_Motor_Memo->Lines->Strings[19];
AC_Motor_Parameters_Delta_Star_Edit->Text="Star";

i=AC_DC_Motor_Memo->Text.Pos("b-r-e-a-k-1");
j=AC_DC_Motor_Memo->Text.Pos("b-r-e-a-k-2");
k=AC_DC_Motor_Memo->Text.Pos("b-r-e-a-k-3");

DC_Motor_Memo->Text=AC_DC_Motor_Memo->Text.SubString(i+11,j-i-11);
AC_Motor_Memo->Text=AC_DC_Motor_Memo->Text.SubString(j+11,k-j-11);



}
//---------------------------------------------------------------------------


void __fastcall TForm1::AC_Motor_Parameters_Load_ButtonClick(
      TObject *Sender)
{
AC_Motor_Parameters_OpenDialog->Filter="Motor Setup Files (*.dmt)|*.DMT";
if (AC_Motor_Parameters_OpenDialog->Execute())
{
AC_DC_Motor_Memo->Lines->LoadFromFile (AC_Motor_Parameters_OpenDialog->FileName);
Adress_Label->Caption=AC_Motor_Parameters_OpenDialog->FileName;
last_dialog_is_open_dialog=true;
last_dialog_is_save_dialog=false;
}

AC_DC_Motor_Parameters_Update_Button->Click();

}
//---------------------------------------------------------------------------

void __fastcall TForm1::AC_Motor_Parameters_Save_ButtonClick(
      TObject *Sender)
{
AC_DC_Motor_Memo_Update_Button->Click();
if ((Adress_Label->Caption)==" ")
AC_Motor_Parameters_Save_As_ButtonClick(Sender);
else
AC_DC_Motor_Memo->Lines->SaveToFile(Adress_Label->Caption);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::AC_Motor_Parameters_Save_As_ButtonClick(
      TObject *Sender)
{

AC_DC_Motor_Memo_Update_Button->Click();
AC_Motor_Parameters_SaveDialog->FileName=Adress_Label->Caption;
if (AC_Motor_Parameters_SaveDialog->Execute())
{
AC_DC_Motor_Memo->Lines->SaveToFile(AC_Motor_Parameters_SaveDialog->FileName);
Adress_Label->Caption=AC_Motor_Parameters_SaveDialog->FileName;
last_dialog_is_open_dialog=false;
last_dialog_is_save_dialog=true;

}
}
//---------------------------------------------------------------------------




void __fastcall TForm1::DAQ_File_Update_ButtonClick(TObject *Sender)
{
str1="";
str2="";
str2="";
```

```
//*******************************************
//These values are unforunately constants
//they may be updated
//******************************
Found_DAQ_Rate_At_Line=48;
Found_Data_Save_File_At_Line=1012;
//**************************************
//File to be updated
DAQ_File_Name="c:\\data\\veri.dsa";

str1=";";
str2="   Sampling_Rate = ";
str3=str2+DAQ_Rate+str1;
//DAQ file loaded
DAQ_File_Update_RichEdit->Lines->LoadFromFile(DAQ_File_Name);
DAQ_File_Update_RichEdit->Lines->Strings[Found_DAQ_Rate_At_Line]=str3;
DAQ_Update_Delay_Timer->Enabled=true;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::DAQ_Update_Delay_TimerTimer(TObject *Sender)
{
DAQ_File_Update_RichEdit->Lines-
>Strings[Found_Data_Save_File_At_Line]=DAQ_Current_Data_Save_File;
DAQ_Update_Delay_Timer2->Enabled=true;
DAQ_Update_Delay_Timer->Enabled=false;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::DAQ_Update_Delay_Timer2Timer(TObject *Sender)
{
DAQ_File_Update_RichEdit->Lines->SaveToFile(DAQ_File_Name);
DAQ_Update_Delay_Timer2->Enabled=false;
//free RAM
DAQ_File_Update_RichEdit->Text="";

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Parameter_Identification_No_Load_Test_Start_Test_ButtonClick(
      TObject *Sender)
{
No_Load_Start_Voltage=StrToInt(Parameter_Identification_No_Load_Test_Test_Voltage_Edit->Text);
No_Load_Voltage_Increment=StrToInt(No_Load_Increment_Edit->Text);
No_Load_Number_Of_Increments=StrToInt(Number_Of_Increments_Edit->Text);





if((last_dialog_is_open_dialog)||(last_dialog_is_save_dialog))
{
Delete_Old_Files_Button->Click();

//Data Acquision File Updated
//Data to be updated
str1="";
str2="";
str3="";
str4="";
str1="      File_Name = '";
//attention !!!!,this may be a save dialog also
if(last_dialog_is_open_dialog)
str2=ExtractFilePath(AC_Motor_Parameters_OpenDialog->FileName);
else
str2=ExtractFilePath(AC_Motor_Parameters_SaveDialog->FileName);

str3="noload.ASC';";
```

```
str4=str1+str2+str3;
DAQ_Current_Data_Save_File=str4;
DAQ_Rate="10.0";
DAQ_File_Update_Button->Click();
//Load File Prepared

//Load side parameters are loaded
Get_Load_Side_Parameters_Button->Click();
//clear old test data
Clear_Parameter_Identification_Test_Data_Button->Click();
//No Load Test data loaded
Parameter_Identification_Test_Data_StringGrid->Cells[0][1]="0";
Parameter_Identification_Test_Data_StringGrid->Cells[1][1]="0";
Parameter_Identification_Test_Data_StringGrid-
>Cells[0][2]=floor(synchronous_speed_of_tested_ac_machine_in_rpm);
Parameter_Identification_Test_Data_StringGrid->Cells[1][2]="100";
Parameter_Identification_Test_Data_StringGrid-
>Cells[0][3]=floor(synchronous_speed_of_tested_ac_machine_in_rpm);
Parameter_Identification_Test_Data_StringGrid->Cells[1][3]="400";
Parameter_Identification_Test_Data_StringGrid->Cells[0][4]="0";
Parameter_Identification_Test_Data_StringGrid->Cells[1][4]="500";
//********************
//NO LOAD FILE PREPARED
//********************
Dummy_Button->Click();
//file open

if ((dosya = fopen("c:\\md29gui\\bin\\sinusoid.dpl", "w+"))
     == NULL)
   return;

fprintf(dosya,"$TITLE no_load  \n");
fprintf(dosya,"$AUTHOR tolgainan       \n");
fprintf(dosya,"$COMPANY METU    \n");
fprintf(dosya,"$VERSION 1.0.0          \n");
fprintf(dosya,"$DRIVE Mentor\n");
fprintf(dosya,"INITIAL{       \n");

fprintf(dosya,"//NO LOAD TEST       \n");
fprintf(dosya,"//NO LOAD TEST       \n");
fprintf(dosya,"//NO LOAD TEST       \n");
fprintf(dosya,"#00.00=200          \n");
fprintf(dosya,"#01.14=1          \n");
fprintf(dosya,"#01.15=0         \n");
          //#01.18 is selected as speed reference
fprintf(dosya,"#01.10=1 \n");
//4Q, Bi-Polar Mode seleced
//-----------------
//------DEGISKEN-----------------------------------
temp=INT(Maximum_RPM/3);
fprintf(dosya,"#01.06=%d  \n",temp);
//Max. Speed Forward
temp=-1*temp;
fprintf(dosya,"#01.09=%d  \n",temp);
          //Max. Speed Reverse
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#01.18=0                  \n");
          //Speed reference initialized
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#02.02=1                  \n");
          //Enable ramps
fprintf(dosya,"#02.04=50          \n");
          //Group1 fwd acceleration time from zero speed to maximum speed
     //that's 5 seconds
fprintf(dosya,"#02.05=50                     \n");
          //Group1 fwd deceleration time
fprintf(dosya,"#02.06=50                              \n");
          //Group1 rev deceleration time
fprintf(dosya,"#02.07=50                                      \n");
          //Group1 rev acceleration time
fprintf(dosya,"#02.14=0  \n");
```

```
fprintf(dosya,"#02.15=0  \n");
fprintf(dosya,"#02.16=0  \n");
fprintf(dosya,"#02.17=0  \n");
fprintf(dosya,"#02.18=0  \n");
	//group1 ramps are selected
fprintf(dosya,"#02.18=0       \n");
		//Common ramp select, group1 ramps selected
fprintf(dosya,"#02.19=0  \n");
	//ramp scaling disabled
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#03.09=080                \n");
		//Speed loop proportional gain, default value
fprintf(dosya,"#03.10=040                          \n");
fprintf(dosya,"#03.11=0                                     \n");
			//Speed loop derivative gain, default value
fprintf(dosya,"#03.12=0                    \n");
			//Analog feedback from tachometer selected instead armature
		//of estimation from the armature voltage
fprintf(dosya,"#03.13=0                              \n");
			//Analog feedback selected
//-----------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Armature_Voltage);
fprintf(dosya,"#03.15=%d  \n",temp);
	//Maximum armature volts
fprintf(dosya,"#03.16=300                 \n");
		//Maximum speed (rpm) /10 with weakened field
	//used to calibrate #03.03 to show actual speed / 10
	//in rpm.
//-----------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Armature_Current*1000/75);
fprintf(dosya,"#04.04=%d  \n",temp);
		//Symmetrical current limitation for positive
					//and negative bridges.
					//full scale 1000 corresponds to drive rating 75A
					//200 corresponds to 15A
fprintf(dosya,"#04.05=%d  \n",temp);
		//Maximum current limit for positive bridge
fprintf(dosya,"#04.06=%d  \n",temp);
		//Maximum current limit for negative bridge
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#04.09=0              \n");
		//current offset
fprintf(dosya,"#04.11=0                         \n");
			//current offset select (1=enabled,0=disbaled)
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#04.12=0                                  \n");
			//speed control
fprintf(dosya,"#04.13=0                                            \n");
			//selected
fprintf(dosya,"#04.14=1 \n");
fprintf(dosya,"#04.15=1 \n");
fprintf(dosya,"#04.16=1 \n");
fprintf(dosya,"#04.17=1 \n");
		//al quadrants enabled
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#05.05=750                                              \n");
		//Maximum current scaling, 75A drive rating
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#05.06=140\n");
		//Threshold of armature current feedback beyond which
					//the current-time overload protection begins to integrate
//-----------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Field_Current*1000/2500);
fprintf(dosya,"#06.08=%d           \n",temp);
		//Maximum field current1,
					//1000 corresponds to 2.5A
					//920 for 2.3A field current
fprintf(dosya,"#06.09=0                     \n");
```

```
                //Maximum field current2=0 for economy an d prevent overheating

temp=INT(Minimum_Field_Current*1000/2500);
fprintf(dosya,"#06.10=%d           \n",temp);
                //Minimum field current, 2.5 * 0.350 = 0.875A field current
//-----------------------DEGISKEN----------------------------------
fprintf(dosya,"#06.11=205                                  \n");
                //Field current feedback scaling, field controller scaled to 2.5A
fprintf(dosya,"#06.12=010                                          \n");
                //Field economy timeout, 10 seconds before selecting field current2
//-------------------------------------------------------------------------
fprintf(dosya,"#06.13=1\n");
                                //Enable field control
fprintf(dosya,"#06.14=1           \n");
                                //Maximum field2 selector, enable field current2------------>Enabled
fprintf(dosya,"#06.15=1                      \n");
                                //Enable field economy timeout----------->Enabled
//****************************************************
//****************************************************

fprintf(dosya,"#07.08=1507         \n");
// set DAC1 source as general read/wrıte paramter 15.07
fprintf(dosya,"#07.09=1506 \n");
// set DAC2 source as general read/wrıte paramter 15.06
fprintf(dosya,"#07.21=1000 \n");
fprintf(dosya,"#07.22=1000 \n");
// default scaling for DAC1  and DAC2

//****************************************************
fprintf(dosya,"reinit    //to change set up parameter       \n");
fprintf(dosya,"#14.04=100\n");
                //Clock section timebase assigned to 100 ms
fprintf(dosya,"#14.06=1 \n");
    //Autorun mode
fprintf(dosya,"reinit    //to change set up parameter       \n");
fprintf(dosya,"}                    \n");
fprintf(dosya,"BACKGROUND{\n");


/*
//motor voltage
temp=INT((motor_voltage_at_no_load_test-b)/a);
// absolute value
if (temp<0)
temp=temp*-1;
//fprintf(dosya,"#15.07=%d  \n",temp);
*/

//motor voltage
temp=INT((No_Load_Start_Voltage-b)/a);
// absolute value
if (temp<0)
temp=temp*-1;
fprintf(dosya,"#15.07=%d  \n",temp);




//****************************************************
//*************tetık**************************************
fprintf(dosya,"#15.06=800 // start experiment at Dasy Lab ------mid point for 3V 7V trigger                  \n");
fprintf(dosya,"DELAY(30)          //delay for operation\n");

//****************tetık***********************************
//****************************************************
fprintf(dosya,"#15.06=100 // experiment goes on \n");
fprintf(dosya,"#15.21=1   // run unidrive           \n");

fprintf(dosya,"DELAY(200)        //delay \n");
```

```
for(i=1;i<No_Load_Number_Of_Increments+1;i++)
{   //begin for
//motor voltage
temp=INT(((No_Load_Start_Voltage+i*No_Load_Voltage_Increment)-b)/a);
// absolute value
if (temp<0)
temp=temp*-1;
fprintf(dosya,"#15.07=%d  \n",temp);

fprintf(dosya,"DELAY(100)        //delay \n");
} //end for



//****************************************************
//*************tetık***************************************
fprintf(dosya,"#15.21=0   // stop unidrive            \n");

fprintf(dosya,"DELAY(80)         //delay for readingdata \n");
fprintf(dosya,"#15.06=-500 // stop experiment at Dasy Lab ------mid point for -3V  -7V trigger         \n");

fprintf(dosya,"DELAY(60)         //delay for stopping experiment \n");
//***************tetık***********************************
//****************************************************
fprintf(dosya,"#15.06=0 // experiment ends \n");
fprintf(dosya,"#15.07=0 // motor voltage back to zero \n");

//we reset the field current
//after 3 seconds to avoid overheating

fprintf(dosya,"DELAY(30)  \n",temp);
fprintf(dosya,"#06.08=0  \n");
fprintf(dosya,"#06.09=0  \n");
fprintf(dosya,"#06.10=0  \n");

fprintf(dosya,"}  \n");






//file closed
fclose(dosya);
//********************
//NO LOAD FILE END*****
//********************



//Download Test Conditions

Download_Test_Conditions_Button->Click();



} //end if   (DIALOG CHECK)
else
{
No_Motor_Defined_Show_Message_Button->Click();
} //end else  (DIALOG CHECK)

}
//---------------------------------------------------------------------------


void __fastcall TForm1::Parameter_Identification_Locked_Rotor_Test_Start_Test_ButtonClick(
      TObject *Sender)
{
```

```
motor_voltage_at_locked_rotor_test=StrToFloat(Parameter_Identification_Locked_Rotor_Test_Test_Voltage_
Edit->Text);

if((last_dialog_is_open_dialog)||(last_dialog_is_save_dialog))
{
Delete_Old_Files_Button->Click();
//Data Acquision File Updated
//Data to be updated
str1="";
str2="";
str3="";
str4="";
str1="      File_Name = '";
//attention !!!!,this may be a save dialog also
if(last_dialog_is_open_dialog)
str2=ExtractFilePath(AC_Motor_Parameters_OpenDialog->FileName);
else
str2=ExtractFilePath(AC_Motor_Parameters_SaveDialog->FileName);

str3="locked.ASC';";
str4=str1+str2+str3;
DAQ_Current_Data_Save_File=str4;
DAQ_Rate="10.0";
DAQ_File_Update_Button->Click();
//Load File Prepared
//Load side parameters are loaded
Get_Load_Side_Parameters_Button->Click();
//clear old test data
Clear_Parameter_Identification_Test_Data_Button->Click();
//Locked rotor Test data loaded
Parameter_Identification_Test_Data_StringGrid->Cells[0][1]="0";
Parameter_Identification_Test_Data_StringGrid->Cells[1][1]="0";

Parameter_Identification_Test_Data_StringGrid->Cells[0][2]="0";
Parameter_Identification_Test_Data_StringGrid->Cells[1][2]="300";

//********************
//LOCKED ROTOR FILE****
//********************
Dummy_Button->Click();
//file open

if ((dosya = fopen("c:\\md29gui\\bin\\sinusoid.dpl", "w+"))
      == NULL)
   return;

fprintf(dosya,"$TITLE locked  \n");
fprintf(dosya,"$AUTHOR tolgainan       \n");
fprintf(dosya,"$COMPANY METU    \n");
fprintf(dosya,"$VERSION 1.0.0          \n");
fprintf(dosya,"$DRIVE Mentor\n");
fprintf(dosya,"INITIAL{       \n");

fprintf(dosya,"//NO LOAD TEST       \n");
fprintf(dosya,"//NO LOAD TEST       \n");
fprintf(dosya,"//NO LOAD TEST       \n");
fprintf(dosya,"#00.00=200          \n");
fprintf(dosya,"#01.14=1          \n");
fprintf(dosya,"#01.15=0         \n");
                //#01.18 is selected as speed reference
fprintf(dosya,"#01.10=1 \n");
//4Q, Bi-Polar Mode seleced
//------------------
//------DEGISKEN------------------------------------
temp=INT(Maximum_RPM/3);
fprintf(dosya,"#01.06=%d  \n",temp);
//Max. Speed Forward
temp=-1*temp;
fprintf(dosya,"#01.09=%d  \n",temp);
                //Max. Speed Reverse
//-----------------------DEGISKEN------------------------------------
```

```
fprintf(dosya,"#01.18=0                \n");
                //Speed reference initialized
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#02.02=1                \n");
                //Enable ramps
fprintf(dosya,"#02.04=50          \n");
                //Group1 fwd acceleration time from zero speed to maximum speed
        //that's 5 seconds
fprintf(dosya,"#02.05=50                      \n");
                //Group1 fwd deceleration time
fprintf(dosya,"#02.06=50                                  \n");
                //Group1 rev deceleration time
fprintf(dosya,"#02.07=50                                              \n");
                //Group1 rev acceleration time
fprintf(dosya,"#02.14=0  \n");
fprintf(dosya,"#02.15=0  \n");
fprintf(dosya,"#02.16=0  \n");
fprintf(dosya,"#02.17=0  \n");
fprintf(dosya,"#02.18=0  \n");
        //group1 ramps are selected
fprintf(dosya,"#02.18=0       \n");
                //Common ramp select, group1 ramps selected
fprintf(dosya,"#02.19=0  \n");
        //ramp scaling disabled
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#03.09=080               \n");
                //Speed loop proportional gain, default value
fprintf(dosya,"#03.10=040                          \n");
fprintf(dosya,"#03.11=0                                     \n");
                                //Speed loop derivative gain, default value
fprintf(dosya,"#03.12=0                 \n");
                                //Analog feedback from tachometer selected instead armature
              //of estimation from the armature voltage
fprintf(dosya,"#03.13=0                            \n");
                                //Analog feedback selected
//-----------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Armature_Voltage);
fprintf(dosya,"#03.15=%d  \n",temp);
        //Maximum armature volts
fprintf(dosya,"#03.16=300                \n");
                //Maximum speed (rpm) /10 with weakened field
        //used to calibrate #03.03 to show actual speed / 10
        //in rpm.
//-----------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Armature_Current*1000/75);
fprintf(dosya,"#04.04=%d  \n",temp);
                //Symmetrical current limitation for positive
                                                //and negative bridges.
                                                //full scale 1000 corresponds to drive rating 75A
                                                //200 corresponds to 15A
fprintf(dosya,"#04.05=%d  \n",temp);
                //Maximum current limit for positive bridge
fprintf(dosya,"#04.06=%d  \n",temp);
                //Maximum current limit for negative bridge
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#04.09=0              \n");
                //current offset
fprintf(dosya,"#04.11=0                        \n");
                                //current offset select (1=enabled,0=disbaled)
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#04.12=0                                  \n");
                                //speed control
fprintf(dosya,"#04.13=0                                                \n");
                                //selected
fprintf(dosya,"#04.14=1 \n");
fprintf(dosya,"#04.15=1 \n");
fprintf(dosya,"#04.16=1 \n");
fprintf(dosya,"#04.17=1 \n");
             //al quadrants enabled
```

```
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#05.05=750                                             \n");
            //Maximum current scaling, 75A drive rating
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#05.06=140\n");
            //Threshold of armature current feedback beyond which
                                    //the current-time overload protection begins to integrate
//------------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Field_Current*1000/2500);
fprintf(dosya,"#06.08=%d         \n",temp);
            //Maximum field current1,
                                    //1000 corresponds to 2.5A
                                    //920 for 2.3A field current
fprintf(dosya,"#06.09=0                  \n");
            //Maximum field current2=0 for economy an d prevent overheating

temp=INT(Minimum_Field_Current*1000/2500);
fprintf(dosya,"#06.10=%d         \n",temp);
            //Minimum field current, 2.5 * 0.350 = 0.875A field current
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#06.11=205                                  \n");
            //Field current feedback scaling, field controller scaled to 2.5A
fprintf(dosya,"#06.12=010                                        \n");
            //Field economy timeout, 10 seconds before selecting field current2
//--------------------------------------------------------------------------
fprintf(dosya,"#06.13=1\n");
                        //Enable field control
fprintf(dosya,"#06.14=1          \n");
                        //Maximum field2 selector, enable field current2------------>Enabled
fprintf(dosya,"#06.15=1                   \n");
                        //Enable field economy timeout----------->Enabled
//*****************************************************
//*****************************************************

fprintf(dosya,"#07.08=1507         \n");
// set DAC1 source as general read/wrıte paramter 15.07
fprintf(dosya,"#07.09=1506 \n");
// set DAC2 source as general read/wrıte paramter 15.06
fprintf(dosya,"#07.21=1000 \n");
fprintf(dosya,"#07.22=1000 \n");
// default scaling for DAC1  and DAC2

//*****************************************************
fprintf(dosya,"reinit    //to change set up parameter       \n");
fprintf(dosya,"#14.04=100\n");
            //Clock section timebase assigned to 100 ms
fprintf(dosya,"#14.06=1 \n");
    //Autorun mode
fprintf(dosya,"reinit    //to change set up parameter       \n");
fprintf(dosya,"}                    \n");
fprintf(dosya,"BACKGROUND{\n");

temp=INT((motor_voltage_at_locked_rotor_test-b)/a);
// absolute value
if (temp<0)
temp=temp*-1;
fprintf(dosya,"#15.07=%d  \n",temp);


//*****************************************************
//**************tetık**************************************
fprintf(dosya,"#15.06=800 // start experiment at Dasy Lab ------mid point for 7V 10V trigger                  \n");
fprintf(dosya,"DELAY(30)         //delay for operation\n");

//****************tetık***********************************
//*****************************************************
fprintf(dosya,"#15.06=100 // experiment goes on \n");
fprintf(dosya,"#15.21=1   // run unidrive            \n");
```

```
fprintf(dosya,"#2.04=0  \n");
fprintf(dosya,"#2.05=0    \n");
fprintf(dosya,"#2.06=0      \n");
fprintf(dosya,"#2.07=0       \n");
fprintf(dosya,"DELAY(300)     \n");

//****************************************************
//*************tetık**************************************
fprintf(dosya,"#15.21=0   // stop unidrive            \n");
fprintf(dosya,"#15.06=-500 // stop experiment at Dasy Lab ------mid point for -3V  -7V trigger
\n");
fprintf(dosya,"DELAY(20)          //delay for readingdata \n");

//****************tetık***********************************
//****************************************************
fprintf(dosya,"#15.06=0 // experiment ends \n");
fprintf(dosya,"#15.07=0 // motor voltage back to zero \n");

//we reset the field current
//after 3 seconds to avoid overheating

fprintf(dosya,"DELAY(30)  \n",temp);
fprintf(dosya,"#06.08=0  \n");
fprintf(dosya,"#06.09=0  \n");
fprintf(dosya,"#06.10=0  \n");

fprintf(dosya,"}  \n");




//file closed

fclose(dosya);

//********************
//LOCKED ROTOR END****
//********************


//Download Test Conditions
Download_Test_Conditions_Button->Click();
} //end if  (DIALOG CHECK)
else
{
No_Motor_Defined_Show_Message_Button->Click();
} //end else      (DIALOG CHECK)

}
//---------------------------------------------------------------------------

/*
//******************************************************
//******GENERAL TEST LAYOUT*********BEGIN****************
//******************************************************
//DAQ_RATE AND DAQ FILE CAN BE UPDATED
//FOR THE SPECIFIC TEST

if((last_dialog_is_open_dialog)||(last_dialog_is_save_dialog))
{
Delete_Old_Files_Button->Click();
//Data Acquision File Updated
//Data to be updated
str1="";
str2="";
str3="";
str4="";
str1="      File_Name = '";
```

```
//attention !!!!,this may be a save dialog also
if(last_dialog_is_open_dialog)
str2=ExtractFilePath(AC_Motor_Parameters_OpenDialog->FileName);
else
str2=ExtractFilePath(AC_Motor_Parameters_SaveDialog->FileName);

str3="locked.ASC';";
str4=str1+str2+str3;
DAQ_Current_Data_Save_File=str4;
DAQ_Rate="10.0";
DAQ_File_Update_Button->Click();

//Load File Prepared

//Load side parameters are loaded
Get_Load_Side_Parameters_Button->Click();

//********************
//TEST FILE **********
//********************
Dummy_Button->Click();
//file open

if ((dosya = fopen("c:\\md29gui\\bin\\sinusoid.dpl", "w+"))
      == NULL)
   return;




//file closed

fclose(dosya);

//********************
//TEST FILE END********
//********************

//Download Test Conditions
Download_Test_Conditions_Button->Click();

} //end if  (DIALOG CHECK)
else
{
No_Motor_Defined_Show_Message_Button->Click();
} //end else   (DIALOG CHECK)

//****************************************************
//******GENERAL TEST LAYOUT*********END*****************
//****************************************************
*/
void __fastcall TForm1::Download_Test_Conditions_ButtonClick(
      TObject *Sender)
{
//This function operates mainly
//with the help of the external timers
Timer0->Enabled=True;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Get_Load_Side_Parameters_ButtonClick(
      TObject *Sender)
{
// parameters are loaded below
Maximum_Armature_Voltage=StrToFloat(DC_Motor_Maximum_Armature_Voltage_Edit->Text);
Maximum_Armature_Current=StrToFloat(DC_Motor_Maximum_Armature_Current_Edit->Text);
Maximum_Field_Current=StrToFloat(DC_Motor_Maximum_Field_Current_Edit->Text);
Minimum_Field_Current=StrToFloat(DC_Motor_Minimum_Field_Current_Edit->Text);
Maximum_RPM=StrToFloat(DC_Motor_Maximum_Speed_Edit->Text);
Torque_Armature_Current_Ratio=StrToFloat(DC_Motor_Torque_Current_Ratio_Edit->Text);
```

```
synchronous_speed_of_tested_ac_machine_in_rpm=(120*StrToFloat(AC_Motor_Parameters_Hz_Edit->
Text))/StrToFloat(AC_Motor_Parameters_Pole_no_Edit-> Text);
//Armature_Current_Reference_Armature_Current_Ratio=10;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::No_Motor_Defined_Show_Message_ButtonClick(
      TObject *Sender)
{
ShowMessage("No Motor Setup File is loaded / saved! ");
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Delete_Old_Files_ButtonClick(TObject *Sender)
{
//we delete old files
DeleteFile("c:\\md29gui\\bin\\sinusoid.dpl");
DeleteFile("c:\\md29gui\\bin\\sinusoid.o");
DeleteFile("c:\\md29gui\\bin\\sinusoid.bin");
DeleteFile("c:\\md29gui\\bin\\sinusoid.src");
DeleteFile("c:\\md29gui\\bin\\sinusoid.p");
DeleteFile("c:\\md29gui\\bin\\sinusoid.sym");
DeleteFile("c:\\md29gui\\bin\\sinusoid.obj");
DeleteFile("c:\\md29gui\\bin\\sinusoid.lnk");
DeleteFile("c:\\md29gui\\bin\\sinusoid.dpp");
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer0Timer(TObject *Sender)
{
WinExec("c:\\md29gui\\bin\\dplc  c:\\md29gui\\bin\\sinusoid.dpl -r -1",SW_SHOWMINNOACTIVE);
Timer1->Enabled=True;
Timer0->Enabled=False;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
WinExec("c:\\md29gui\\bin\\gas960 -o c:\\md29gui\\bin\\sinusoid.o -ASA
c:\\md29gui\\bin\\sinusoid.src",SW_HIDE);
Timer2->Enabled=True;
Timer1->Enabled=False;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer2Timer(TObject *Sender)
{
WinExec("c:\\md29gui\\bin\\gld960 c:\\md29gui\\bin\\sinusoid.lnk",SW_HIDE);
Timer3->Enabled=True;
Timer2->Enabled=False;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer3Timer(TObject *Sender)
{
WinExec("c:\\md29gui\\bin\\grom960 c:\\md29gui\\bin\\sinusoid.obj -i -o
c:\\md29gui\\bin\\sinusoid.bin",SW_HIDE);
Timer4->Enabled=True;
Timer3->Enabled=False;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer4Timer(TObject *Sender)
{
WinExec("c:\\md29gui\\bin\\crctouch c:\\md29gui\\bin\\sinusoid.bin 1
c:\\md29gui\\bin\\sinusoid.dpl",SW_SHOWMINNOACTIVE);
Timer5->Enabled=True;
Timer4->Enabled=False;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer5Timer(TObject *Sender)
```

```
{
WinExec("c:\\md29gui\\bin\\flasher c:\\md29gui\\bin\\sinusoid.bin MD29 1 ",SW_SHOWMINNOACTIVE);
Timer5->Enabled=False;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Clear_Parameter_Identification_Test_Data_ButtonClick(
      TObject *Sender)
{
Dummy_Button->Click();
for(i=1;i<(Parameter_Identification_Test_Data_StringGrid->RowCount+1);i++)
for(j=0;j<(Parameter_Identification_Test_Data_StringGrid->ColCount+1);j++)
Parameter_Identification_Test_Data_StringGrid->Cells[j][i]="";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Parameter_Identification_Get_Parameters_ButtonClick(
      TObject *Sender)
{
No_Load_Number_Of_Increments=StrToInt(Number_Of_Increments_Edit->Text);

if(!((last_dialog_is_open_dialog)||(last_dialog_is_save_dialog)))
No_Motor_Defined_Show_Message_Button->Click();

//variables loaded
skip_row_count=80;
skip_row_count1=140;
skip_row_count2=80;
calculation_row_count=20;

total_col_count=11;

cumulative_iron_loss_resistance=0;
cumulative_magnetizing_branch_inductance_in_ohm=0;
cumulative_power_per_phase=0;
cumulative_equivalent_series_resistance=0;
cumulative_equivalent_leakage_inductance=0;
cumulative_power_per_phase=0;
cumulative_V_phase_to_neutral_average=0;
cumulative_I_phase_average=0;

//
/*
long double power_per_phase_set[20];
long double V_phase_to_neutral_average_set[20];
long double I_phase_average_set[20];
*/


//*********
//*********





//Locked Rotor  file is loaded
str1="";
str2="";
str3="";

if(last_dialog_is_open_dialog)
str1=ExtractFilePath(AC_Motor_Parameters_OpenDialog->FileName);
else
str1=ExtractFilePath(AC_Motor_Parameters_SaveDialog->FileName);

str2="locked.ASC";
str3=str1+str2;

Parameter_Identification_Current_Data_Read_File=str3;

//locked rotor test paramter calculations
```

```
 if (
(file_read=fopen(Parameter_Identification_Current_Data_Read_File.c_str(),"r")) != NULL)
{




// to remove the introduction part of data file
for(i=0;i<57;i++)
fscanf(file_read,"%s",dizi);

//skip unnecessary data
for (k=0;k<(skip_row_count1*total_col_count);k++)
fscanf(file_read,"%s",dizi);
col_count=0;
for (k=0;k<(calculation_row_count*total_col_count);k++)

{

fscanf(file_read,"%s",dizi);
if (col_count!=total_col_count)
data_set_param[col_count]=StrToFloat(dizi);
if (col_count==total_col_count)
{
//data complete
//calculate paremeters
power_per_phase=data_set_param[5]/3;
V_phase_to_neutral_average=(data_set_param[1]+data_set_param[2])*0.288675134;
I_phase_average=(data_set_param[3]+data_set_param[4])*0.5;
impedance_Z=V_phase_to_neutral_average/I_phase_average;
equivalent_series_resistance=power_per_phase/(I_phase_average*I_phase_average);
equivalent_leakage_inductance=sqrt( abs((impedance_Z*impedance_Z)-
(equivalent_series_resistance*equivalent_series_resistance)));
cumulative_equivalent_series_resistance=cumulative_equivalent_series_resistance+equivalent_series_resistanc
e;
cumulative_equivalent_leakage_inductance=cumulative_equivalent_leakage_inductance+equivalent_leakage_i
nductance;
cumulative_power_per_phase=cumulative_power_per_phase+power_per_phase;
col_count=0;
}
col_count=col_count+1;
}   // end for
fclose(file_read);

// get the avverage values from the cumulative
equivalent_series_resistance=cumulative_equivalent_series_resistance/(calculation_row_count-1);
equivalent_leakage_inductance=cumulative_equivalent_leakage_inductance/(calculation_row_count-1);
power_per_phase=cumulative_power_per_phase/(calculation_row_count-1);


//finally display the required value


Parameter_Identification_Equivalent_Series_Resistance_Label-
>Caption=(floor(equivalent_series_resistance*100))/100;
Parameter_Identification_Equivalent_Leakage_Inductance_Label-
>Caption=(floor(equivalent_leakage_inductance*100))/100;
//=(floor(power_per_phase*100))/100;

}//end if



}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Plot_Test_Conditions_ButtonClick(TObject *Sender)
{

//we use this Dummy_Button click in order to avoid  reading
//Test_Data_StringGrid data when it's being editted.
Dummy_Button->Click();
// clear RAM
Test_Chart->Series[0]->Clear();




Tq1=0;
t=0;
dTq_dt=1200;
// 120 A/s = 1200 #04.09/s

//*******************
//speed_time plot begin
if
(Control_Mode_ComboBox->Items->IndexOf(Control_Mode_ComboBox->Text)==0)
{

Test_Chart->LeftAxis->Title->Caption="Speed (rpm)";


for(i=1;i<(Test_Data_StringGrid->RowCount+1);i++)
{
data_invalid_x=false;
data_invalid_y=false;

if (Test_Data_StringGrid->Cells[1][i]=="")
data_invalid_x=true;
if (Test_Data_StringGrid->Cells[0][i]=="")
data_invalid_y=true;

if (Test_Data_StringGrid->Cells[1][i]==" ")
data_invalid_x=true;
if (Test_Data_StringGrid->Cells[0][i]==" ")
data_invalid_y=true;



if((data_invalid_x==false)&&(data_invalid_y==false))
{
x=StrToFloat(Test_Data_StringGrid->Cells[1][i]);
y=StrToFloat(Test_Data_StringGrid->Cells[0][i]);
Test_Chart->Series[0]->AddXY(x,y,"",clBlack);
}


}
// for end
}
//speed_time plot end
//*******************


//*******************
//torque_time plot begin
if
(Control_Mode_ComboBox->Items->IndexOf(Control_Mode_ComboBox->Text)==1)
{
Test_Chart->LeftAxis->Title->Caption="Torque (Nm)";

for(i=1;i<(Test_Data_StringGrid->RowCount+1);i++)
{
data_invalid_x=false;
data_invalid_y=false;

if (Test_Data_StringGrid->Cells[1][i]=="")
```

```
data_invalid_x=true;
if (Test_Data_StringGrid->Cells[0][i]=="")
data_invalid_y=true;

if (Test_Data_StringGrid->Cells[1][i]==" ")
data_invalid_x=true;
if (Test_Data_StringGrid->Cells[0][i]==" ")
data_invalid_y=true;



if((data_invalid_x==false)&&(data_invalid_y==false))
{
t=StrToFloat(Test_Data_StringGrid->Cells[1][i]);
Tq2=StrToFloat(Test_Data_StringGrid->Cells[0][i]);

x=t;
y=Tq1;
Test_Chart->Series[0]->AddXY(x,y,"",clBlack);

dTq=Tq1-Tq2;
// absolute value
if (dTq<0)
dTq=dTq*-1;

x=(t/10)+(dTq/dTq_dt);
y=Tq2;
Test_Chart->Series[0]->AddXY(x,y,"",clBlack);
Tq1=Tq2;
}

}
// for end
}
//torque_time plot end
//*******************



}
//---------------------------------------------------------------------------

void __fastcall TForm1::Clear_Test_Data_ButtonClick(TObject *Sender)
{

//we use this Dummy_Button click in order to avoid  reading
//Test_Data_StringGrid data when it's being editted.
Dummy_Button->Click();
for(i=1;i<(Test_Data_StringGrid->RowCount+1);i++)
for(j=0;j<(Test_Data_StringGrid->RowCount+1);j++)
Test_Data_StringGrid->Cells[j][i]="";
Plot_Test_Conditions_Button->Click();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Control_Mode_ComboBoxChange(TObject *Sender)
{


//speed-time test predefined conditions
 if
(Control_Mode_ComboBox->Items->IndexOf(Control_Mode_ComboBox->Text)==0)
{
Clear_Test_Data_Button->Click();
Test_Data_StringGrid->Enabled=true;
}


//torque-time test predefined conditions
 if
(Control_Mode_ComboBox->Items->IndexOf(Control_Mode_ComboBox->Text)==1)
```

```
{
Clear_Test_Data_Button->Click();
Test_Data_StringGrid->Enabled=true;
}

 if
(Control_Mode_ComboBox->Items->IndexOf(Control_Mode_ComboBox->Text)==1)
   Test_Data_StringGrid->Cells[0][0]="Torque (Nm)";
 else
    Test_Data_StringGrid->Cells[0][0]="Speed (rpm)";
Plot_Test_Conditions_Button->Click();

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Test_Data_StringGridGetEditMask(TObject *Sender,
      int ACol, int ARow, AnsiString &Value)
{
if  (ACol==1)
      Value="9999"; //mask for time


if  (ACol==0)

      {
       if (Control_Mode_ComboBox->Items->IndexOf(Control_Mode_ComboBox->Text)==0)
       Value="#9999";//mask for speed
       else
       Value="#999";//mask for torque
      }

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Tests_Automatic_Loading_Start_Test_ButtonClick(
      TObject *Sender)
{
motor_voltage_at_auto_load_test=StrToFloat(Automatic_Loading_Test_Voltage->Text);

//*****************************************************
//******AUTOMATIC LOADING TEST *****BEGIN****************
//*****************************************************
//DAQ_RATE AND DAQ FILE CAN BE UPDATED
//FOR THE SPECIFIC TEST

if((last_dialog_is_open_dialog)||(last_dialog_is_save_dialog))
{
Delete_Old_Files_Button->Click();
//Data Acquision File Updated
//Data to be updated
str1="";
str2="";
str3="";
str4="";
str1="      File_Name = '";
//attention !!!!,this may be a save dialog also
if(last_dialog_is_open_dialog)
str2=ExtractFilePath(AC_Motor_Parameters_OpenDialog->FileName);
else
str2=ExtractFilePath(AC_Motor_Parameters_SaveDialog->FileName);

str3="auto.ASC';";
str4=str1+str2+str3;
DAQ_Current_Data_Save_File=str4;
DAQ_Rate="20.0";
DAQ_File_Update_Button->Click();

//Load File Prepared

//Load side parameters are loaded
Get_Load_Side_Parameters_Button->Click();
```

```
//********************
//TEST FILE **********
//********************
Dummy_Button->Click();
//file open

if ((dosya = fopen("c:\\md29gui\\bin\\sinusoid.dpl", "w+"))
      == NULL)
   return;





//**********speed-time file begin**************************

if
(Control_Mode_ComboBox->Items->IndexOf(Control_Mode_ComboBox->Text)==0)
{
//********************************************************
//**********speed-time file begin**************************
//********************************************************
fprintf(dosya,"$TITLE spd_time   \n");
fprintf(dosya,"$AUTHOR tolgainan       \n");
fprintf(dosya,"$COMPANY METU    \n");
fprintf(dosya,"$VERSION 1.0.0          \n");
fprintf(dosya,"$DRIVE Mentor\n");
fprintf(dosya,"INITIAL{       \n");

fprintf(dosya,"//SPEED-TIME TYPE LOAD       \n");
fprintf(dosya,"//SPEED-TIME TYPE LOAD       \n");
fprintf(dosya,"//SPEED-TIME TYPE LOAD       \n");

fprintf(dosya,"#00.00=200          \n");
fprintf(dosya,"#01.14=1          \n");
fprintf(dosya,"#01.15=0         \n");
               //#01.18 is selected as speed reference
fprintf(dosya,"#01.10=1 \n");
//4Q, Bi-Polar Mode seleced
//-----------------
//------DEGISKEN-----------------------------------
temp=INT(Maximum_RPM/3);
fprintf(dosya,"#01.06=%d  \n",temp);
//Max. Speed Forward
temp=-1*temp;
fprintf(dosya,"#01.09=%d  \n",temp);
               //Max. Speed Reverse
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#01.18=0                 \n");
               //Speed reference initialized
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#02.02=1                 \n");
               //Enable ramps
fprintf(dosya,"#02.04=50          \n");
               //Group1 fwd acceleration time from zero speed to maximum speed
       //that's 5 seconds
fprintf(dosya,"#02.05=50                      \n");
               //Group1 fwd deceleration time
fprintf(dosya,"#02.06=50                                  \n");
               //Group1 rev deceleration time
fprintf(dosya,"#02.07=50                                            \n");
               //Group1 rev acceleration time
fprintf(dosya,"#02.14=0  \n");
fprintf(dosya,"#02.15=0  \n");
fprintf(dosya,"#02.16=0  \n");
fprintf(dosya,"#02.17=0  \n");
fprintf(dosya,"#02.18=0  \n");
       //group1 ramps are selected
fprintf(dosya,"#02.18=0       \n");
```

```
		//Common ramp select, group1 ramps selected
fprintf(dosya,"#02.19=0  \n");
	//ramp scaling disabled
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#03.09=080		\n");
		//Speed loop proportional gain, default value
fprintf(dosya,"#03.10=040			\n");

fprintf(dosya,"#03.11=0				\n");
			//Speed loop derivative gain, default value
fprintf(dosya,"#03.12=0		\n");
			//Analog feedback from tachometer selected instead armature
		//of estimation from the armature voltage
fprintf(dosya,"#03.13=0			\n");
			//Analog feedback selected
//------------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Armature_Voltage);
fprintf(dosya,"#03.15=%d  \n",temp);
	//Maximum armature volts

fprintf(dosya,"#03.16=300		\n");
		//Maximum speed (rpm) /10 with weakened field
	//used to calibrate #03.03 to show actual speed / 10
	//in rpm.
//------------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Armature_Current*1000/75);
fprintf(dosya,"#04.04=%d  \n",temp);
		//Symmetrical current limitation for positive
				//and negative bridges.
				//full scale 1000 corresponds to drive rating 75A
				//200 corresponds to 15A
fprintf(dosya,"#04.05=%d  \n",temp);
		//Maximum current limit for positive bridge
fprintf(dosya,"#04.06=%d  \n",temp);
		//Maximum current limit for negative bridge
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#04.09=0		\n");
		//current offset
fprintf(dosya,"#04.11=0			\n");
			//current offset select (1=enabled,0=disbaled)
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#04.12=0				\n");
			//speed control
fprintf(dosya,"#04.13=0					\n");
			//selected
fprintf(dosya,"#04.14=1 \n");
fprintf(dosya,"#04.15=1 \n");
fprintf(dosya,"#04.16=1 \n");
fprintf(dosya,"#04.17=1 \n");
		//al quadrants enabled
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#05.05=750						\n");
		//Maximum current scaling, 75A drive rating
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#05.06=140\n");
		//Threshold of armature current feedback beyond which
				//the current-time overload protection begins to integrate
//------------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Field_Current*1000/2500);
fprintf(dosya,"#06.08=%d	\n",temp);
		//Maximum field current1,
				//1000 corresponds to 2.5A
				//920 for 2.3A field current
fprintf(dosya,"#06.09=0		\n");
		//Maximum field current2=0 for economy an d prevent overheating

temp=INT(Minimum_Field_Current*1000/2500);
fprintf(dosya,"#06.10=%d	\n",temp);
```

```
        //Minimum field current, 2.5 * 0.350 = 0.875A field current
//-----------------------DEGISKEN----------------------------------
fprintf(dosya,"#06.11=205                              \n");
        //Field current feedback scaling, field controller scaled to 2.5A
fprintf(dosya,"#06.12=010                                          \n");
        //Field economy timeout, 10 seconds before selecting field current2
//-------------------------------------------------------------------------
fprintf(dosya,"#06.13=1\n");
                //Enable field control
fprintf(dosya,"#06.14=1         \n");
                //Maximum field2 selector, enable field current2----------->Enabled
fprintf(dosya,"#06.15=1                   \n");
                //Enable field economy timeout----------->Enabled
//****************************************************
//****************************************************

fprintf(dosya,"#07.08=1507        \n");
// set DAC1 source as general read/wrıte paramter 15.07
fprintf(dosya,"#07.09=1506 \n");
// set DAC2 source as general read/wrıte paramter 15.06
fprintf(dosya,"#07.21=1000 \n");
fprintf(dosya,"#07.22=1000 \n");
// default scaling for DAC1  and DAC2

//****************************************************
fprintf(dosya,"reinit    //to change set up parameter      \n");
fprintf(dosya,"#14.04=100\n");
        //Clock section timebase assigned to 100 ms
fprintf(dosya,"#14.06=1 \n");
   //Autorun mode
fprintf(dosya,"reinit    //to change set up parameter      \n");
fprintf(dosya,"}                    \n");
fprintf(dosya,"BACKGROUND{\n");



temp=INT((motor_voltage_at_auto_load_test-b)/a);
// absolute value
if (temp<0)
temp=temp*-1;
fprintf(dosya,"#15.07=%d  \n",temp);




//*************tetık*************************************
fprintf(dosya,"#15.06=400 // start experiment at Dasy Lab ------mid point for 3V 7V trigger                  \n");
fprintf(dosya,"DELAY(30)         //delay for operation\n");

//****************tetık**********************************
//****************************************************
fprintf(dosya,"#15.06=100 // experiment goes on \n");
fprintf(dosya,"#15.21=1   // run unidrive           \n");

for(i=1;i<(Test_Data_StringGrid->RowCount+1);i++)
{  // begin for
data_invalid_x=false;
data_invalid_y=false;

if (Test_Data_StringGrid->Cells[1][i]=="")
data_invalid_x=true;
if (Test_Data_StringGrid->Cells[0][i]=="")
data_invalid_y=true;

if (Test_Data_StringGrid->Cells[1][i]==" ")
      data_invalid_x=true;

if (Test_Data_StringGrid->Cells[0][i]==" ")
      data_invalid_y=true;

// valid test data are transferred to test data
```

```
        if((data_invalid_x==false)&&(data_invalid_y==false))
                {
                test_data[1][i]=10*StrToFloat(Test_Data_StringGrid->Cells[1][i]);
                test_data[0][i]=StrToFloat(Test_Data_StringGrid->Cells[0][i]);
                }
//invalid data replaced with zeros
        else
                {
                test_data[1][i]=0;
                test_data[0][i]=0;
                 }
}  //end for


time1=0;
y1=0;
acc_time1=50;
for(i=1;i<(Test_Data_StringGrid->RowCount+1);i++)
{ //begin for
time2=test_data[1][i];
delay=time2-time1;
y2=test_data[0][i];

//calculate the new accerelation time
// it is not infinity(<1999) and it is nonegative
if(y2!=y1)
acc_time2=INT(delay*3000/(y2-y1));

// absolute value
if (acc_time2<0)
acc_time2=acc_time2*-1;

//clamp accerelation time
if (acc_time2>1999)
acc_time2=1999;



//write the required acc.
//no need to write the same acc.

        temp=acc_time2;
                if (acc_time2!=acc_time1)
                {
                fprintf(dosya,"  \n");
                fprintf(dosya,"#2.04=%d  \n",temp);
                fprintf(dosya,"#2.05=%d  \n",temp);
                fprintf(dosya,"#2.06=%d  \n",temp);
                fprintf(dosya,"#2.07=%d  \n",temp);
                fprintf(dosya,"  \n");
                }


//write required speed
//no need to write the same  speed
                temp=INT(y2/3);
                if (y2!=y1)
                fprintf(dosya,"#1.18=%d  \n",temp);

//time is unidirectional, at least for now
// write required delay
        if (delay>0)
                {
                temp=INT(delay/1);
                fprintf(dosya,"DELAY(%d)  \n",temp);
                 }


//update time1 and y1
y1=y2;
time1=time2;
```

```
acc_time1=acc_time2;
} //end for

//**************tetık***************************************
fprintf(dosya,"DELAY(10)          //delay for stoping experiment \n");
fprintf(dosya,"#15.06=-500 // stop experiment at Dasy Lab ------mid point for -3V  -7V trigger        \n");
fprintf(dosya,"DELAY(20)          //delay for stoping experiment \n");

fprintf(dosya,"#15.21=0   // stop unidrive             \n");
//*****************tetık************************************
//****************************************************
fprintf(dosya,"DELAY(20)          //delay for readingdata \n");
fprintf(dosya,"#15.06=0 // experiment ends \n");
fprintf(dosya,"#15.07=0 // motor voltage back to zero \n");

//we reset the field current
//after 3 seconds to avoid overheating

fprintf(dosya,"DELAY(30)  \n",temp);
fprintf(dosya,"#06.08=0  \n");
fprintf(dosya,"#06.09=0  \n");
fprintf(dosya,"#06.10=0  \n");

fprintf(dosya,"}  \n");



//***********speed-time file  end***************************
//**********************************************************
}
//**********************************************************
//***********speed-time file  end***************************

//**********************************************************
//***********torque-time file begin***************************
//**********************************************************
if
(Control_Mode_ComboBox->Items->IndexOf(Control_Mode_ComboBox->Text)==1)
{
//**********************************************************
//***********torque-time file begin***************************
//**********************************************************
fprintf(dosya,"$TITLE trq_time   \n");
fprintf(dosya,"$AUTHOR tolgainan       \n");
fprintf(dosya,"$COMPANY METU    \n");
fprintf(dosya,"$VERSION 1.0.0          \n");
fprintf(dosya,"$DRIVE Mentor\n");
fprintf(dosya,"INITIAL{       \n");

fprintf(dosya,"//TORQUE-TIME TYPE LOAD       \n");
fprintf(dosya,"//TORQUE-TIME TYPE LOAD       \n");
fprintf(dosya,"//TORQUE-TIME TYPE LOAD       \n");

fprintf(dosya,"#00.00=200         \n");
fprintf(dosya,"#01.14=1          \n");
fprintf(dosya,"#01.15=0        \n");
                //#01.18 is selected as speed reference
fprintf(dosya,"#01.10=1 \n");
//4Q, Bi-Polar Mode seleced
//------------------
//------DEGISKEN------------------------------------
temp=INT(Maximum_RPM/3);
fprintf(dosya,"#01.06=%d  \n",temp);
//Max. Speed Forward
temp=-1*temp;
fprintf(dosya,"#01.09=%d  \n",temp);
                //Max. Speed Reverse
//------------------------DEGISKEN------------------------------------
fprintf(dosya,"#01.18=0                \n");
                //Speed reference initialized
//------------------------DEGISKEN------------------------------------
```

```
fprintf(dosya,"#02.02=1                  \n");
                //Enable ramps
fprintf(dosya,"#02.04=50          \n");
                //Group1 fwd acceleration time from zero speed to maximum speed
        //that's 5 seconds
fprintf(dosya,"#02.05=50                      \n");
                //Group1 fwd deceleration time
fprintf(dosya,"#02.06=50                                  \n");
                //Group1 rev deceleration time
fprintf(dosya,"#02.07=50                                              \n");
                //Group1 rev acceleration time
fprintf(dosya,"#02.14=0  \n");
fprintf(dosya,"#02.15=0  \n");
fprintf(dosya,"#02.16=0  \n");
fprintf(dosya,"#02.17=0  \n");
fprintf(dosya,"#02.18=0  \n");
        //group1 ramps are selected
fprintf(dosya,"#02.18=0       \n");
                //Common ramp select, group1 ramps selected
fprintf(dosya,"#02.19=0  \n");
        //ramp scaling disabled
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#03.09=080               \n");
                //Speed loop proportional gain, default value
fprintf(dosya,"#03.10=040                          \n");

fprintf(dosya,"#03.11=0                                    \n");
                                //Speed loop derivative gain, default value
fprintf(dosya,"#03.12=0                    \n");
                                //Analog feedback from tachometer selected instead armature
                //of estimation from the armature voltage
fprintf(dosya,"#03.13=0                              \n");
                                //Analog feedback selected
//------------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Armature_Voltage);
fprintf(dosya,"#03.15=%d  \n",temp);
        //Maximum armature volts

fprintf(dosya,"#03.16=300                 \n");
                //Maximum speed (rpm) /10 with weakened field
        //used to calibrate #03.03 to show actual speed / 10
        //in rpm.
//------------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Armature_Current*1000/75);
fprintf(dosya,"#04.04=%d  \n",temp);
                //Symmetrical current limitation for positive
                                                //and negative bridges.
                                                //full scale 1000 corresponds to drive rating 75A
                                                //200 corresponds to 15A
fprintf(dosya,"#04.05=%d  \n",temp);
                //Maximum current limit for positive bridge
fprintf(dosya,"#04.06=%d  \n",temp);
                //Maximum current limit for negative bridge
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#04.09=0               \n");
                //current offset
fprintf(dosya,"#04.11=1                          \n");
                                //current offset select (1=enabled,0=disbaled)
//------------------------DEGISKEN-----------------------------------
fprintf(dosya,"#04.12=0                                \n");
                                //torque control
fprintf(dosya,"#04.13=1                                            \n");
                                //selected
fprintf(dosya,"#04.14=1 \n");
fprintf(dosya,"#04.15=1 \n");
fprintf(dosya,"#04.16=1 \n");
fprintf(dosya,"#04.17=1 \n");
            //al quadrants enabled
//------------------------DEGISKEN-----------------------------------
```

```
fprintf(dosya,"#05.05=750                                    \n");
        //Maximum current scaling, 75A drive rating
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#05.06=140\n");
        //Threshold of armature current feedback beyond which
                        //the current-time overload protection begins to integrate
//-----------------------DEGISKEN-----------------------------------

temp=INT(Maximum_Field_Current*1000/2500);
fprintf(dosya,"#06.08=%d          \n",temp);
        //Maximum field current1,
                        //1000 corresponds to 2.5A
                        //920 for 2.3A field current
fprintf(dosya,"#06.09=0              \n");
        //Maximum field current2=0 for economy an d prevent overheating

temp=INT(Minimum_Field_Current*1000/2500);
fprintf(dosya,"#06.10=%d          \n",temp);
        //Minimum field current, 2.5 * 0.350 = 0.875A field current
//-----------------------DEGISKEN-----------------------------------
fprintf(dosya,"#06.11=205                              \n");
        //Field current feedback scaling, field controller scaled to 2.5A
fprintf(dosya,"#06.12=010                                       \n");
        //Field economy timeout, 10 seconds before selecting field current2
//------------------------------------------------------------------------
fprintf(dosya,"#06.13=1\n");
                //Enable field control
fprintf(dosya,"#06.14=1          \n");
                //Maximum field2 selector, enable field current2------------>Enabled
fprintf(dosya,"#06.15=1                  \n");
                //Enable field economy timeout----------->Enabled
//*****************************************************
//*****************************************************

fprintf(dosya,"#07.08=1507        \n");
// set DAC1 source as general read/wrıte paramter 15.07
fprintf(dosya,"#07.09=1506 \n");
// set DAC2 source as general read/wrıte paramter 15.06
fprintf(dosya,"#07.21=1000 \n");
fprintf(dosya,"#07.22=1000 \n");
// default scaling for DAC1  and DAC2

//*****************************************************
fprintf(dosya,"reinit    //to change set up parameter       \n");
fprintf(dosya,"#14.04=100\n");
        //Clock section timebase assigned to 100 ms
fprintf(dosya,"#14.06=1 \n");
    //Autorun mode
fprintf(dosya,"reinit    //to change set up parameter       \n");
fprintf(dosya,"}                  \n");
fprintf(dosya,"BACKGROUND{\n");

temp=INT((motor_voltage_at_auto_load_test-b)/a);
// absolute value
if (temp<0)
temp=temp*-1;
fprintf(dosya,"#15.07=%d  \n",temp);

//*****************************************************
//*************tetık*************************************
fprintf(dosya,"#15.06=800 // start experiment at Dasy Lab ------mid point for 3V 7V trigger                  \n");
fprintf(dosya,"DELAY(30)          //delay for operation\n");

//****************tetık**********************************
//*****************************************************
fprintf(dosya,"#15.06=100 // experiment goes on \n");
fprintf(dosya,"#15.21=1   // run unidrive           \n");

fprintf(dosya,"#1.18=100 \n");//nonzero speed ref.

for(i=1;i<(Test_Data_StringGrid->RowCount+1);i++)
```

```
{  // begin for
data_invalid_x=false;
data_invalid_y=false;

if (Test_Data_StringGrid->Cells[1][i]=="")
data_invalid_x=true;
if (Test_Data_StringGrid->Cells[0][i]=="")
data_invalid_y=true;

if (Test_Data_StringGrid->Cells[1][i]==" ")
      data_invalid_x=true;

if (Test_Data_StringGrid->Cells[0][i]==" ")
      data_invalid_y=true;

// valid test data are transferred to test data
      if((data_invalid_x==false)&&(data_invalid_y==false))
            {
            test_data[1][i]=10*StrToFloat(Test_Data_StringGrid->Cells[1][i]);
            test_data[0][i]=StrToFloat(Test_Data_StringGrid->Cells[0][i]);
            }
//invalid data replaced with zeros
      else
            {
            test_data[1][i]=0;
            test_data[0][i]=0;
            }
}  //end for


time1=0;
y1=0;
for(i=1;i<(Test_Data_StringGrid->RowCount+1);i++)
{ //begin for
time2=test_data[1][i];
delay=time2-time1;
y2=test_data[0][i];
//time is unidirectional, at least for now
// write required delay
      if (delay>0)
            {
            temp=INT(delay/1);
            fprintf(dosya,"DELAY(%d)  \n",temp);
            }
//write required torque
//no need to write the torque

temp=INT((y2*Armature_Current_Reference_Armature_Current_Ratio)/Torque_Armature_Current_Ratio);
            if (y2!=y1)
            fprintf(dosya,"#4.09=%d  \n",temp);

//update time1 and y1
y1=y2;
time1=time2;
} //end for
```

```
//****************************************************
//*************tetık***************************************
fprintf(dosya,"#15.21=0   // stop unidrive           \n");
fprintf(dosya,"#15.06=-500 // stop experiment at Dasy Lab ------mid point for -3V -7V trigger                  \n");
fprintf(dosya,"DELAY(20)          //delay for readingdata \n");

//****************tetık************************************
//****************************************************
fprintf(dosya,"#15.06=0 // experiment ends \n");
fprintf(dosya,"#15.07=0 // motor voltage back to zero \n");

//we reset the field current
//after 3 seconds to avoid overheating

fprintf(dosya,"DELAY(30)  \n",temp);
fprintf(dosya,"#06.08=0  \n");
fprintf(dosya,"#06.09=0  \n");
fprintf(dosya,"#06.10=0  \n");

fprintf(dosya,"}  \n");



//***********torque-time file  end***************************
//**********************************************************
}
//**********************************************************
//***********torque-time file  end***************************

//file closed

fclose(dosya);

//********************
//TEST FILE END********
//********************

//Download Test Conditions
Download_Test_Conditions_Button->Click();

} //end if  (DIALOG CHECK)
else
{
No_Motor_Defined_Show_Message_Button->Click();
} //end else   (DIALOG CHECK)

//*****************************************************
//****AUTOMATIC LOADING TEST********END*****************
//*****************************************************

}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::Plot_Results_ButtonClick(TObject *Sender)
{

if(!((last_dialog_is_open_dialog)||(last_dialog_is_save_dialog)))
No_Motor_Defined_Show_Message_Button->Click();

else
{

total_col_count=11;
//we use this Dummy_Button click in order to avoid  reading
//Test_Data_StringGrid data when it's being editted.
Dummy_Button->Click();

// clear RAM
Results_Chart->Series[0]->Clear();
Results_Chart_XY->Series[0]->Clear();

Results_Chart->Visible=false;
Results_Chart_XY->Visible=false;

col_number_x=X_Parameter_ComboBox->ItemIndex;
col_number_y=Y_Parameter_ComboBox->ItemIndex;
current_col_number=0;

x_new=false;
y_new=false;



str1="";
str2="";
str3="";

if(last_dialog_is_open_dialog)
str1=ExtractFilePath(AC_Motor_Parameters_OpenDialog->FileName);
else
str1=ExtractFilePath(AC_Motor_Parameters_SaveDialog->FileName);
str2="auto.ASC";
str3=str1+str2;

Motor_Characteristics_Current_Data_Read_File=str3;

//no-load test parameter calculations

 if (
(file_read_plot=fopen(Motor_Characteristics_Current_Data_Read_File.c_str(),"r")) != NULL)
{
// to remove the introduction part of data file
for(counter=0;counter<57;counter++)
fscanf(file_read_plot,"%s",dizi_plot);


//skip initial data
//skip 2,5 seconds for 20 Hz
for (k=0;k<(50*total_col_count);k++)
//fscanf(file_read_plot,"%s",dizi);


//if time is selected for X axis use LINE for plotting
//since evertthing is function of time,
//that is--it has unique value at each time instant.
//the other relations may not result in functions
//so we will use DOTS for plotting

//********************
//X-time plot with LINE
//********************
if(X_Parameter_ComboBox->ItemIndex==0)
```

```
{

Results_Chart->Visible=true;

if (col_number_x==0)
Results_Chart->BottomAxis->Title->Caption="Time (Sec)";

if (col_number_y==0)
Results_Chart->LeftAxis->Title->Caption="Time (Sec)";
if (col_number_y==1)
Results_Chart->LeftAxis->Title->Caption="Voltage (Volt)";
if (col_number_y==2)
Results_Chart->LeftAxis->Title->Caption="Voltage (Volt)";
if (col_number_y==3)
Results_Chart->LeftAxis->Title->Caption="Current (Ampere)";
if (col_number_y==4)
Results_Chart->LeftAxis->Title->Caption="Current (Ampere)";
if (col_number_y==5)
Results_Chart->LeftAxis->Title->Caption="Power Input (Watts)";
if (col_number_y==6)
Results_Chart->LeftAxis->Title->Caption="Torque (Nm)";
if (col_number_y==7)
Results_Chart->LeftAxis->Title->Caption="Speed (rpm)";
if (col_number_y==8)
Results_Chart->LeftAxis->Title->Caption="Power Output (Watts)";
if (col_number_y==9)
Results_Chart->LeftAxis->Title->Caption="Eff. %";
if (col_number_y==10)
Results_Chart->LeftAxis->Title->Caption="Cos Fi";

// data scan start
do
  {

fscanf(file_read_plot,"%s",dizi_plot);
if (current_col_number==col_number_x)
{
x_data=StrToFloat(dizi_plot);
x_new=true;
}



if (current_col_number==col_number_y)
{
y_data=StrToFloat(dizi_plot);
y_new=true;
}

if((x_new=true)&(y_new==true))
{
Results_Chart->Series[0]->AddXY(x_data,y_data,"",clBlack);
x_new==false;
x_new==false;
}

current_col_number=current_col_number+1;
if (current_col_number==total_col_count)
current_col_number=0;

 } while (!feof(file_read_plot));
// data scan end
//Results_Chart->->UndoZoom();
} //end if


//********************
//X-Y plot with DOTS
//********************

else
```

```
{

if (col_number_y==0)
Results_Chart_XY->LeftAxis->Title->Caption="Time (Sec)";
if (col_number_y==1)
Results_Chart_XY->LeftAxis->Title->Caption="Voltage (Volt)";
if (col_number_y==2)
Results_Chart_XY->LeftAxis->Title->Caption="Voltage (Volt)";
if (col_number_y==3)
Results_Chart_XY->LeftAxis->Title->Caption="Current (Ampere)";
if (col_number_y==4)
Results_Chart_XY->LeftAxis->Title->Caption="Current (Ampere)";
if (col_number_y==5)
Results_Chart_XY->LeftAxis->Title->Caption="Power Input (Watts)";
if (col_number_y==6)
Results_Chart_XY->LeftAxis->Title->Caption="Torque (Nm)";
if (col_number_y==7)
Results_Chart_XY->LeftAxis->Title->Caption="Speed (rpm)";
if (col_number_y==8)
Results_Chart_XY->LeftAxis->Title->Caption="Power Output (Watts)";
if (col_number_y==9)
Results_Chart_XY->LeftAxis->Title->Caption="Eff. %";
if (col_number_y==10)
Results_Chart_XY->LeftAxis->Title->Caption="Cos Fi";

if (col_number_x==0)
Results_Chart_XY->BottomAxis->Title->Caption="Time (sec)";
if (col_number_x==1)
Results_Chart_XY->BottomAxis->Title->Caption="Voltage  (Volt)";
if (col_number_x==2)
Results_Chart_XY->BottomAxis->Title->Caption="Voltage (Volt)";
if (col_number_x==3)
Results_Chart_XY->BottomAxis->Title->Caption="Current (Ampere)";
if (col_number_x==4)
Results_Chart_XY->BottomAxis->Title->Caption="Current (Ampere)";
if (col_number_x==5)
Results_Chart_XY->BottomAxis->Title->Caption="Power Input (Watts)";
if (col_number_x==6)
Results_Chart_XY->BottomAxis->Title->Caption="Torque (Nm)";
if (col_number_x==7)
Results_Chart_XY->BottomAxis->Title->Caption="Speed (rpm)";
if (col_number_x==8)
Results_Chart_XY->BottomAxis->Title->Caption="Power Output (Watts)";
if (col_number_x==9)
Results_Chart_XY->BottomAxis->Title->Caption="Eff. %";
if (col_number_x==10)
Results_Chart_XY->BottomAxis->Title->Caption="Cos Fi";

Results_Chart_XY->Visible=true;
// data scan start
do
  {

fscanf(file_read_plot,"%s",dizi_plot);
if (current_col_number==col_number_x)
{
x_data=StrToFloat(dizi_plot);
x_new=true;
}
```

```
if (current_col_number==col_number_y)
{
y_data=StrToFloat(dizi_plot);
y_new=true;
}


if((x_new=true)&(y_new==true))
{
Results_Chart_XY->Series[0]->AddXY(x_data,y_data,"",clBlack);
x_new==false;
x_new==false;
}

current_col_number=current_col_number+1;
if (current_col_number==total_col_count)
current_col_number=0;



 } while (!feof(file_read_plot));
// data scan end
//Results_Chart_XY->UndoZoom();

} //end else
fclose(file_read_plot);


} //end if  fopen

} //end else dialog check

}
//---------------------------------------------------------------------------


void __fastcall TForm1::No_Load_Test_Results_ButtonClick(TObject *Sender)
{
No_Load_Number_Of_Increments=StrToInt(Number_Of_Increments_Edit->Text);

if(!((last_dialog_is_open_dialog)||(last_dialog_is_save_dialog)))
No_Motor_Defined_Show_Message_Button->Click();

//variables loaded
skip_row_count=80;
skip_row_count1=140;
skip_row_count2=80;
calculation_row_count=20;

total_col_count=11;

cumulative_iron_loss_resistance=0;
cumulative_magnetizing_branch_inductance_in_ohm=0;
cumulative_power_per_phase=0;
cumulative_equivalent_series_resistance=0;
cumulative_equivalent_leakage_inductance=0;
cumulative_power_per_phase=0;
cumulative_V_phase_to_neutral_average=0;
cumulative_I_phase_average=0;
cumulative_speed=0;

//
/*
long double power_per_phase_set[20];
long double V_phase_to_neutral_average_set[20];
long double I_phase_average_set[20];
*/

//**********
//**********
```

```
//No Load Test file is loaded
str1="";
str2="";
str3="";

if(last_dialog_is_open_dialog)
str1=ExtractFilePath(AC_Motor_Parameters_OpenDialog->FileName);
else
str1=ExtractFilePath(AC_Motor_Parameters_SaveDialog->FileName);

str2="noload.ASC";
str3=str1+str2;

Parameter_Identification_Current_Data_Read_File=str3;

//no-load test parameter calculations

 if (
(file_read=fopen(Parameter_Identification_Current_Data_Read_File.c_str(),"r")) != NULL)
{


No_Load_Test_Power_Chart->Series[0]->Clear();
//file_read=fopen("c:\\dene\\motor\\m1\\noload.asc","r");

//data_set_param[0]---->Time
//data_set_param[1]---->Voltage1
//data_set_param[2]---->Voltage2
//data_set_param[3]---->Current1
//data_set_param[4]---->Current2
//data_set_param[5]---->Power in
//data_set_param[6]---->Torque
//data_set_param[7]---->Speed
//data_set_param[8]---->Power out
//data_set_param[9]---->Efficieny
//data_set_param[10]---->Cos fi

// to remove the introduction part of data file
for(i=0;i<57;i++)
fscanf(file_read,"%s",dizi);

//skip unnecessary data
for (k=0;k<(skip_row_count1*total_col_count);k++)
fscanf(file_read,"%s",dizi);

//***************************

//V1 BEGIN
//*********

for(i=0;i<No_Load_Number_Of_Increments+1;i++)

{ // begin grand for

col_count=0;

for (k=0;k<(calculation_row_count*total_col_count);k++)
{
fscanf(file_read,"%s",dizi);

if (col_count!=total_col_count)
data_set_param[col_count]=StrToFloat(dizi);

if (col_count==total_col_count)
{
//data complete
//calculate paremeters
power_per_phase=data_set_param[5];
V_phase_to_neutral_average=(data_set_param[1]+data_set_param[2])*0.5;
I_phase_average=(data_set_param[3]+data_set_param[4])*0.5;
speed_avg=data_set_param[7];
```

```
/*iron_loss_resistance=(V_phase_to_neutral_average*V_phase_to_neutral_average)/power_per_phase;
iron_loss_conductance=1/iron_loss_resistance;
admittance_Y=I_phase_average/V_phase_to_neutral_average;
temp1=1000000*admittance_Y*admittance_Y;
temp2=1000000*iron_loss_conductance*iron_loss_conductance;
magnetizing_branch_conductance_in_mho=sqrt(abs(temp1-temp2));
magnetizing_branch_inductance_in_ohm=1000/(magnetizing_branch_conductance_in_mho);


cumulative_iron_loss_resistance=cumulative_iron_loss_resistance+iron_loss_resistance;
cumulative_magnetizing_branch_inductance_in_ohm=cumulative_magnetizing_branch_inductance_in_ohm+magnetizing_branch_inductance_in_ohm;
*/
cumulative_power_per_phase=cumulative_power_per_phase+power_per_phase;
cumulative_V_phase_to_neutral_average=cumulative_V_phase_to_neutral_average+V_phase_to_neutral_average;
cumulative_I_phase_average=cumulative_I_phase_average+I_phase_average;
cumulative_speed=cumulative_speed+speed_avg;

col_count=0;
}
col_count=col_count+1;
}   // end for

/*// get the avverage values from the cumulative
magnetizing_branch_inductance_in_ohm=cumulative_magnetizing_branch_inductance_in_ohm/(calculation_row_count-1);
iron_loss_resistance=cumulative_iron_loss_resistance/(calculation_row_count-1);
*/

//finally save the required values
power_per_phase_set[i]=cumulative_power_per_phase/(calculation_row_count-1);
V_phase_to_neutral_average_set[i]=cumulative_V_phase_to_neutral_average/(calculation_row_count-1);
I_phase_average_set[i]=cumulative_I_phase_average/(calculation_row_count-1);
speed_set[i]=cumulative_speed/(calculation_row_count-1);
//finally save the required value

// display results
No_Load_Test_Results_StringGrid->Cells[0][i+1]=FloatToStr(floor(V_phase_to_neutral_average_set[i]));
No_Load_Test_Results_StringGrid->Cells[1][i+1]=FloatToStr(floor(I_phase_average_set[i]*100)/100);
No_Load_Test_Results_StringGrid->Cells[2][i+1]=FloatToStr(floor(power_per_phase_set[i]));
No_Load_Test_Results_StringGrid->Cells[3][i+1]=FloatToStr(floor((V_phase_to_neutral_average_set[i])*(V_phase_to_neutral_average_set[i])));
No_Load_Test_Results_StringGrid->Cells[4][i+1]=FloatToStr(floor(speed_set[i]));

y=StrToFloat(No_Load_Test_Results_StringGrid->Cells[2][i+1]);
x=StrToFloat(No_Load_Test_Results_StringGrid->Cells[3][i+1]);
No_Load_Test_Power_Chart->Series[0]->AddXY(x,y,"",clBlack);

//Parameter_Identification_Core_Loss_Resistance_Label->Caption=(floor(iron_loss_resistance*100))/100;
//Parameter_Identification_Magnetizing_Branch_Inductance_Label->Caption=(floor(magnetizing_branch_inductance_in_ohm*100))/100;

//=(floor(power_per_phase*100))/100;

//reset cumulators
```

```
cumulative_iron_loss_resistance=0;
cumulative_magnetizing_branch_inductance_in_ohm=0;
cumulative_power_per_phase=0;
cumulative_equivalent_series_resistance=0;
cumulative_equivalent_leakage_inductance=0;
cumulative_power_per_phase=0;
cumulative_V_phase_to_neutral_average=0;
cumulative_I_phase_average=0;
cumulative_speed=0;

//V1 END


//skip data between two voltages

if (i<No_Load_Number_Of_Increments)
            {

              for (k=0;k<(skip_row_count2*total_col_count);k++)
              fscanf(file_read,"%s",dizi);
            }


} // end grand for

fclose(file_read);
}  // end if

}
//--------------------------------------------------------------------------
```