

**Design of a Solution to Unit Commitment  
Using Visual Basic**

by

Muhammad Zamir B Abu Bakar

Dissertation submitted in partial fulfilment of  
the requirements for the  
Bachelor of Engineering (Hons)  
(Electrical & Electronics Engineering)

JUNE 2004

Universiti Teknologi PETRONAS  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan

t  
TK  
1005  
M941

2004

1. Electric power consumption --  
Forecasting -- Mathematical model
2. EEE -- Thesis

CERTIFICATION OF APPROVAL

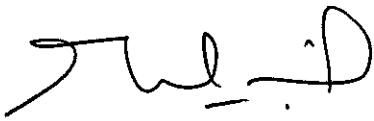
**Design of a Solution to Unit Commitment  
Using Visual Basic**

by

Muhammad Zamir B Abu Bakar

A project dissertation submitted to the  
Electrical & Electronic Engineering Programme  
Universiti Teknologi PETRONAS  
in partial fulfilment of the requirement for the  
BACHELOR OF ENGINEERING (Hons)  
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,



(Mr. Nursyarizal Bin Mohd. Nor)

Nursyarizal Mohd Nor  
Lecturer  
Electrical Engineering Department  
Universiti Teknologi PETRONAS  
31750 Tronoh  
Perak

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

June 2004

## CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



---

MUHAMMAD ZAMIR B ABU BAKAR

## **ABSTRACT**

This Final Year Project paper describes the application of the Visual Basic software in solving the Unit Commitment problem. The main objective is to find the most economical generator combination to accomplish the consumer load demand. Using this simulation software, it can reduce the budget of power company but maintain the power supplied to the customer. This project is continued from previous student. The author is needed to enhance the project by redesign the existing software to the new one that able to simulate more than 10 generators. Author is expected to design the programming using Visual Basic software and analyzed it. Knowledge in Unit Commitment problem and the understanding on how to implement the Visual Basic software are prerequisite to successful completion of this project.

## **ACKNOWLEDGEMENT**

All praises be upon Allah s.w.t. the Almighty and peace be upon Prophet Muhammad s.a.w. Firstly, the author would like to express his appreciation to the Electrical and Electronic Engineering Department of UTP for the support in completing this report. The author is grateful to Mr. Nursyarizal Mohd. Nor, for his many helpful discussions and suggestions. Also special thanks to Mr. Jale and Mr. Suhaimi, lecturer from IT&IS department for their willingness in sharing the knowledge and information. The author also appreciates Mr. Azizan for his valuable guidance and help, the appreciations also goes to the UTP's staff support and resources provided by the UTP Resource Center. Thanks and apologies to others whose contributions that have been overlooked. Last but not least, the continual encouragement and supports from Jamiatul Aqmar, the author's family and friends, which have lifted his spirits on countless occasions, are deeply and sincerely appreciated.

## TABLE OF CONTENTS

<b>CERTIFICATION</b>	. . . . .	i
<b>ABSTRACT</b>	. . . . .	iii
<b>ACKNOWLEDGEMENT</b>	. . . . .	iv
<b>TABLE OF CONTENTS</b>	. . . . .	v
<b>LIST OF FIGURES</b>	. . . . .	vi
<b>LIST OF TABLES</b>	. . . . .	vi
<b>CHAPTER 1:</b>	<b>INTRODUCTION</b>	1
	1.1 Background of Study	1
	1.2 Problem Statement	2
	1.3 Objectives and Scope of Study	5
<b>CHAPTER 2:</b>	<b>LITERATURE REVIEW / THEORY</b>	6
	2.1 Unit Commitment	6
	2.2 Visual Basic 6.0	8
<b>CHAPTER 3:</b>	<b>METHODOLOGY / PROJECT WORK</b>	11
<b>CHAPTER 4:</b>	<b>RESULTS AND DISCUSSIONS</b>	15
	4.1 Unit Commitment	15
	4.1.1 Equations	16
	4.1.2 Calculation	20
	4.2 Visual Basic	23
	4.2.1 Steps in program development	24
	4.3 Discussions	26
<b>CHAPTER 5:</b>	<b>CONCLUSION AND RECOMMENDATIONS</b>	37
	5.1 Conclusion	37
	5.2 Recommendations	38
<b>REFERENCES</b>	. . . . .	39
<b>APPENDIX</b>	. . . . .	41

## LIST OF FIGURES

- Figure 2.1 Discrete levels of system load for an example daily load cycle
- Figure 3.1 Flow Chart of the Project
- Figure 3.2 Steps of program development
- Figure 4.1 N thermal units committed to serve a load of  $P_{load}$
- Figure 4.2 Form for generator data entry
- Figure 4.3 Form for generator data searching
- Figure 4.4 Form for consumer load demand data entry
- Figure 4.5 Form for results
- Figure 4.6 Data control button
- Figure 4.7 Properties window for Data control
- Figure 4.8 DatabaseName dialog box
- Figure 4.9 Record source property windows
- Figure 4.10 ADODC button
- Figure 4.11 Components window
- Figure 4.12 ADODC property pages
- Figure 4.13 Record source for ADODC property pages
- Figure 4.14 Data fields in the database of generators

## LIST OF TABLES

- Table 4.1 Capacity of generators
- Table 4.2 Possible combination of 4 generators

# CHAPTER 1

## INTRODUCTION

### 1.1 Background of Study

Since human activities follow cycles, most systems supplying services to a large population will experience cycles. These include transportation systems and communication systems. The total loads on the system will generally be higher during daytime and early evening when industrial loads are high, lights are on, and so forth, and lower during late evening and early morning when most of the populations are asleep. In addition, the use of electric power has a weekly cycle, the load being lower over weekend than weekdays.

Nowadays, economic operation is very important for a power system to return a profit on the capital invested. One of the economic operation problems that are usually faced by the power system industries is called Unit Commitment problems. Shifting of the load demand on power supply system require a sufficient number of generating units to be committed to supply the required load. Note that, to "commit" a generating unit is to "turn it on", i.e. to bring the unit up to speed synchronize it to the system, and connect it so that it can deliver power to the network. The problem with "commit enough units and leave them on line" is one of economics. It is quite expensive to run too many generating units. A great deal of money can be saved by turning units off when they are not needed. Hence, electricity generating companies and power systems have the problem of deciding how best to meet the varying demand for electricity.

Therefore, the unit commitment problem is to schedule available generators (on or off) to meet the required loads at a minimum cost subject to system constraints. The most



important constraint is that the total generation must equal the forecast half-hourly demands for electricity. Unit commitment is a very challenging optimization problem, because of the astronomical number of feasible combinations of the on and off states of all the generating units in the power system over all the time-points in the study period.

The uses of programming software in solving the economic problem are no longer weird. Almost all of the system in our daily life used this type of solution. In solving Unit Commitment problems, we are trying to implement the Visual Basic (VB). VB is designed to allow the user to develop applications that run under windows without the complexity generally associated with windows programming. VB follow a relatively new type of programming called event-driven programming. In the event-driven model, programs are no longer procedural; they do not follow a sequential logic. Users, as a programmer, do not take control and determine the sequence of execution.

Visual Basic software is quite popular among the expert and beginner programmer. This is because Visual Basic can provide everything needed in order to develop any applications in an easy-to-use-and-learn Graphical User Interface (GUI). Visual Basic is easy to learn and understand and user friendly. Visual Basic provides a set of tools that makes it easy to develop powerful Windows applications-fast. That's not to say that there is nothing to learn, but the learning curve is small-even for those with no programming experience.

## **1.2 Problem Statement**

Electricity generating companies and power systems have the problem of deciding how best to meet the varying demand for electricity. The use of electrical power has a daily and weekly cycle. The demand or total load is lower over weekend than weekdays. As electricity cannot be stored, it is necessary to start-up and shut-down a number of generating units at various power stations each day.

Unit commitment optimizes the short-term scheduling of the generating units at power stations. Unit commitment involves determining which generators in a given facility will be committed to meet the expected load over the near term. Unit commitment is usually scheduled a day in advance, and a generator is said to be committed once it is started. Nearly all generators are rotating devices, so a committed generator is often referred to as a spinning generator. A committed unit can remain offline indefinitely without producing energy.

Electricity generating companies and power systems cannot just simply commit enough units to cover the maximum system loads and leave them running. The problem with "commit enough units and leave them on line" is one of economics. It is quite expensive to run too many generating units. A great deal of money can be saved by turning units off when they are not needed. A well done optimization could provide substantial annual savings in operational and fuel costs. It can minimize the total fuel cost or to maximize the total profit, over a study period of typically a day, subject to a large number of difficult constraints that must be satisfied.

The problem is to find the most economic production and trade program for power generation, when the power consumption and the technological and economical parameters of the power sources are known. Well done optimization could provide substantial annual savings in operational and fuel costs. Unit commitment was done by formulating it as a mixed integer problem, which included the most important economic and technological parameters. The unit commitment has often been solved by linear programming.

With the proliferation of computers into every part of our lives, the demand for software to accomplish everyday tasks grows at a rate that developers struggle to keep up with. In addition, many people have needs that are so unique it is impossible to buy software to meet them all. For those with programming experience, an easy solution is to write an application that meets all of their requirements. That way they get exactly what they

want and can change the program when their goals change. Visual Basic provides a set of tools that makes it easy to develop powerful windows applications-fast.

The dynamic nature of the unit commitment problem complicates its solution. Suppose that 10 units are available for scheduling within any one-hour interval, which is not unlikely in practice. Then, theoretically a total of  $2^{10} - 1 = 1023$  combinations can be listed. If it were possible to link each prospective combination of any one hour to each prospective combination of the next hour of the day, the total number of candidate combinations become  $(1023)^{24} = 1.726 \times 10^{72}$ , which is enormously large and unrealistic to handle. Fortunately, however, the multistage decision process of the unit commitment problem can be dimensionally reduced by practical constraints of system operations and by a search procedure based on the following observations:

- 1) The daily schedule has N discrete time intervals or stages, the durations of which are not necessarily equal. Stage 1 precedes stage 2, and so on to the final stage N.
- 2) A decision must be made for each stage k regarding which particular combination of units to operate during that stage. This is the stage k subproblem.
- 3) To solve for the N decisions, N subproblems are solved sequentially in such a way (called the principle of optimality) that the combined best decisions for the N subproblems yield the best overall solution for the original problem.

### 1.3 Objectives and Scope of Study

The objectives of this project are as follows:

1. To learn and study the application and concept of Visual Basic (VB) which acts as a controller to solve unit commitment problem.
2. To be exposed with the concept of Unit Commitment problem.
3. To redesign Visual Basic simulation software to control more than 10 generators.
4. Research and literature review on Unit Commitment problem.

The aim of the project is to concentrate more on Visual Basic in solving the Unit Commitment problem.

1. To rewrite and execute previous Visual Basic program.
2. To define and investigate in the errors that occurs during the execution.
3. To identify which line of the coding that needs to be change in order for the program to simulate more than 10 generators.
4. To enhance the interface of the program so that it is more users friendly and easy to understand.

The scope of study:

1. Literature review on Visual Basic in order to understand the existing program.
2. To refer with VB expert and discuss the problem encountered.
3. Internet surfing to gather additional information about Unit Commitment and to find information about Visual Basic.

## CHAPTER 2

### LITERATURE REVIEW / THEORY

#### 2.1 Unit Commitment

Because the total load of the power system varies throughout the day and reaches a different peak value from one day to another, the electric utility has to decide in advance which generators to start-up and when to connect them to the network and the sequence in which the operating units should be shut-down and for how long. The computational procedure for making such decisions is called *unit commitment* and a unit when schedule for connection to the system is said to be *committed*. [1]

To develop the concept of unit commitment, we consider the problem of scheduling fossil-fired thermal units in which the aggregate costs (such as start-up costs, operating fuel costs and shut-down costs) are to be minimized over a daily load cycle. The underlying principles are more easily explained if we disregard transmission loss in the system. Without losses, the transmission network is equivalent to a single plant bus to which all generators and all loads are connected, and the total plant output is equal to the total system load. [1]

The power system with  $K$  generating units (no two identical) must have at least one unit on-line to supply the system load which is never zero over the daily load cycle. If each unit can be considered either “on” (denoted by 1) or “off” (denoted by 0), there are  $2^K - 1$  candidate combinations to be examined in each stage of the study period. For example if,  $K = 4$ , so there are  $2^4 - 1 = 15$  combinations (see Table 4.2). Of course, not all combinations are feasible because of the constraints imposed by the load level and other

practical operating requirements of the system. For example, a combination of units of total capability less than 1400 MW cannot serve a load of 1400 MW or greater; any such combination is feasible and can be disregarded over any time interval in which that level of load occurs. [1]

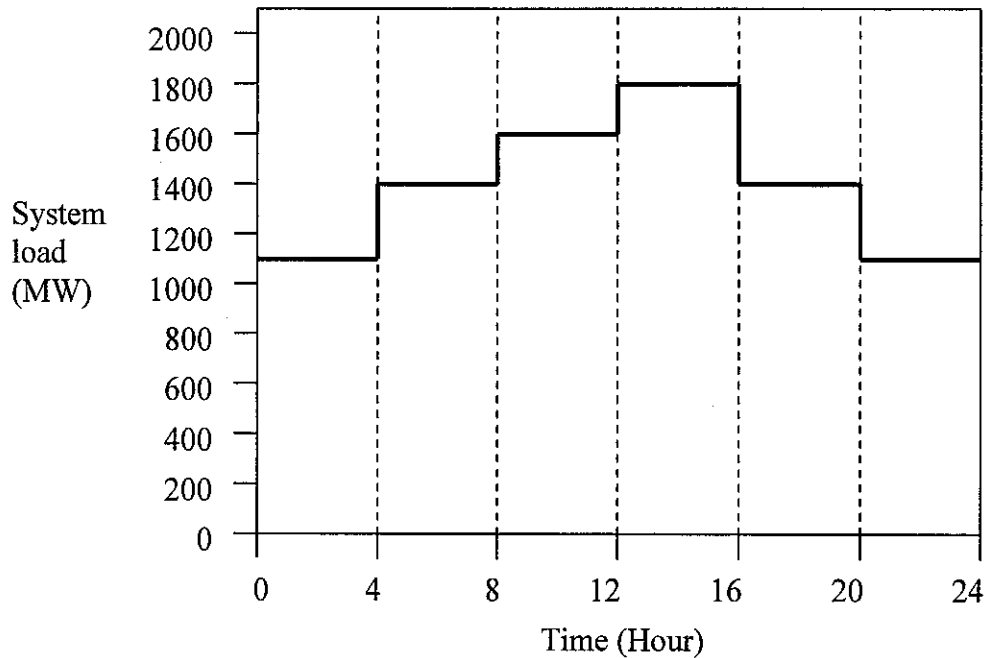


Figure 2.1: Discrete levels of system load for an example daily load cycle

We subdivide the 24-h day into discrete intervals or stages and the predicted load of the system will be considered constant over each interval, as exemplified in Figure 2.1. The unit commitment procedure then searches for the most economic feasible combination of generating units to serve the forecast load of the system at each stage of the load cycle. [1]

## 2.2 Visual Basic 6.0

Visual Basic is a powerful development environment because it allows applications to be developed visually. You drag and drop controls into forms. Then, you arrange and size controls on a form to create the interface for your Windows applications. However, the forms and controls by themselves do not create very useful applications. Code must be added to forms and controls to give them more complex functionality. Programming code is a set of instructions that tells the computer what to do.

When developing Windows applications, events are used as triggers to perform the actions of a program. Pressing a button, changing the size of a form, or just moving the mouse causes an event. This is where the real power of Visual Basic development comes from: combining events with code to produce powerful applications. These applications are called event-driven applications because the events that users perform make the application work. [4]

Most applications interact with data of some sort. It might be a simple text file that contains only a few lines of text, or it might be a large data base with millions of records. Regardless of the amount of data, visual basic provides a way to access these types of data. Text files have many uses. You could store text for a document, a log file, or even comma-delimited records. The **FileSystemObject** (FSO) objects supplies you with methods to easily open and manipulate text files. Databases have become invaluable in today's computing environment. They provide an extremely fast way to store, categorize, and retrieve large amount of data. To work with all types of databases, visual basic now uses the **ADO Data Control** (ADO is an acronym for **ActiveX Data Objects**). [3]

Many applications access information contained in database. Because of the variety of database systems, it would be difficult to implement a separate mechanism for accessing each type of database. Instead, an interface has been established that can interact with the different types of database. This interface is called **OLE DB**. While **OLE DB** is designed to interact with databases, it is not ideal for use in applications, which leads to

the need for something to connect the capabilities of **OLE DB** to applications. Microsoft **ActiveX Data Objects (ADO)** acts as that connection. **ADO** is compatible with any **OLE DB** data source and can be used to bind controls to that data source. Binding a control to a data source basically means that the control displays the data contained in one of the fields in a database. This can be done for display purposes only, or the user could also use the control to make changes to the data that is displayed. [4]

Many times, due to time constraints, people jump right into developing an application without planning the management of their development. This error can cause a lot of headaches. Careful planning can benefit even the smallest project. It will help protect your code and the time you have invested. While planning is very beneficial to your personal projects, it is absolutely essential to group projects. There must be a mechanism in place to manage who develops which files when. Communication between team members is crucial to avoid repeating work or, even worse, losing work. [5]

ActiveX Data Objects (ADO) access data from OLE DB providers. The Connection object is used to specify a particular provider and any parameters. To connect to a data source, you use a Connection object. Using that connection, you can create a new record set, and using the Recordset object's method and properties, you can work with your data. An ADO transaction marks the beginning and end of a series of data operations that are executed across a connection. ADO makes sure that changes to a data source resulting from operations in a transaction either all occur successfully, or not at all. If you cancel the transaction or one of its operations fails, then the result will be as if none of the operations in the transaction had occurred. [5]

For obvious reasons, Object Linking and Embedding (OLE) is a very popular programming topic. Using OLE you can give the users of your program direct access to OLE server programs like Microsoft Word or Excel. In fact, you can integrate all kinds of programs together using OLE, giving your program the power database, spreadsheet, word processor and even graphics programs all wrapped into one. Visual Basic lets you do this with the OLE control. This control can display OLE objects and those objects



appear as mini-versions of the programs connected to them. For example, if you display an Excel spreadsheet in an OLE control, the control displays what looks like a small version of Excel right there in your program. The program that creates the object displayed in the OLE control is an OLE *server*, and your program, which displays the OLE object, is called an OLE *container*. In fact, the proper name for the OLE control is the OLE container control. [5]

Rich text boxes (RTF) text support a variety of formats. For example, you can color text in a rich text box, underline it, bold it, or make it italic. You can select fonts and font sizes, as well as write the text out to disk or read it back in. RTF boxes can also hold a great amount of data, unlike standard text boxes, which are limited to 64K characters. RTF text was designed to be a step beyond plain text, and because many word processors let you save text in that format, it can provide a link between different types of word processors. Using RTF boxes, you can also create your own simple word processors, and that's exactly what the Visual Basic Application Wizard does if you create an application with it. [6]

## **CHAPTER 3**

### **METHODOLOGY / PROJECT WORK**

Methodology used at the first stage of this project was:

1. Internet surfing
  - To gather knowledge, information and data required on related websites regarding Visual Basic and Unit Commitment problems.
  
2. Secondary Source
  - To gather information through reading and observation from the reference book borrowed from Resource Centre.
  - To acquire relevant data and information as project is concerned.
  
3. Reference from the expert on Visual Basic.
  - To discuss with expert in solving the problem that occurs.
  - To investigate which line of coding that needs changes in order to achieve the main objectives of this project.
  - To make some changes in the users interface of previous program.

In performing the Visual Basic programming, author need to study the VB coding done by previous student. Before start studying the coding, author first has to gain knowledge about VB. The procedure used in dealing with VB was listed below:

1. Study the coding.

- In this stage, author studied all the coding. Determine and understand the function of each line coding.

2. Rewrite the coding.

- After completing the first stage, author then rewrite the overall coding and try to run it.
- If the program can be run, then it means that there is no problem in the coding and author can proceed to the next stage. But if there is an error occurs, means the program cannot be run, changes must be made to the coding.

3. Modify the coding.

- At this stage, the main objective of the project is going to be achieved. The existence coding is going to be changed or modified so that it will be able to simulate more than 10 generators. The existences coding only able to simulate up to 10 generators. If we exceed more than 10 generators, the program will not run.

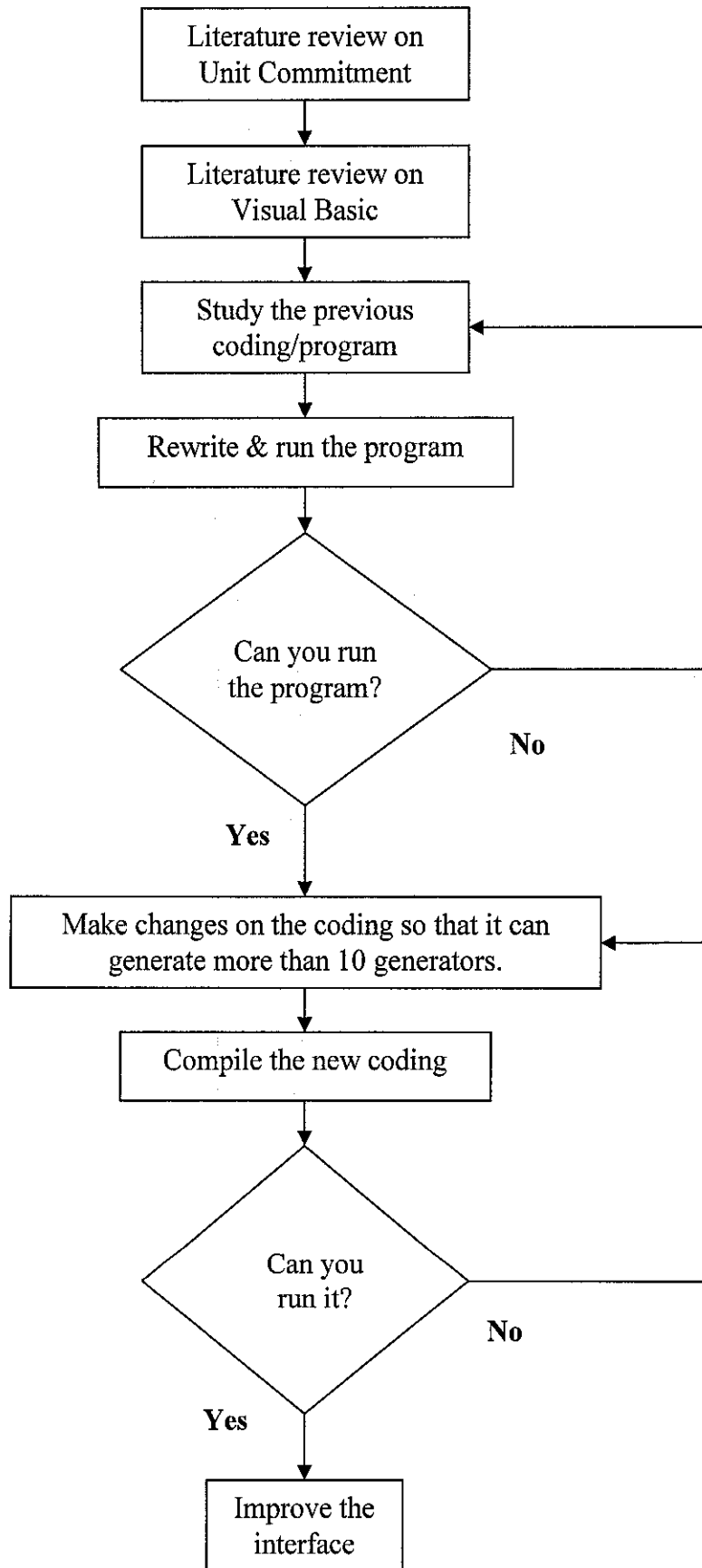


Figure 3.1: Flow Chart of the Project

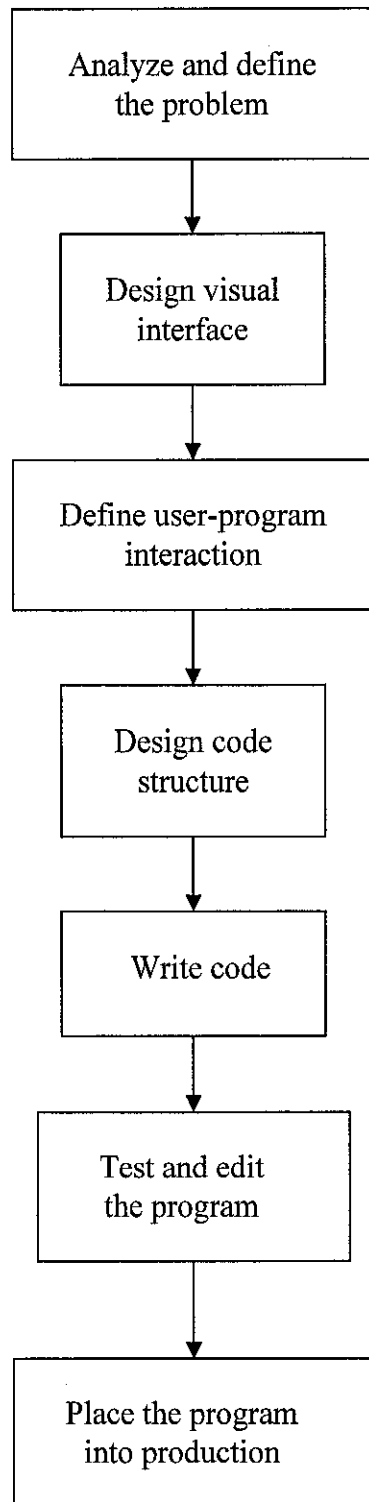


Figure 3.2: Steps of program development

## CHAPTER 4

### RESULTS AND DISCUSSIONS

#### 4.1 Unit Commitment

Economic operation is very important for a power system to return a profit on the capital invested. Rates fixed by regulatory bodies and the importance of conservation of fuel place pressure on power companies to achieve maximum possible efficiency. Maximum efficiency minimizes the cost of a kilowatthour to the consumer and the cost to the company of delivering that kilowatthour in the face of constantly rising prices for fuel, labor, supplies, and maintenance.

Operational economics involving power generation and delivery can be subdivided into two parts; one dealing with minimum cost of power production called economic dispatch and the other dealing with minimum-loss delivery of the generated power to the loads. For any specified load condition economic dispatch determines the power output of each plant (and each generating unit within the plant) which will minimize the overall cost of fuel needed to serve the system load. The minimum-loss problem can assume many forms depending on how control of the power flow in the system is exercised. The economic dispatch problem and also the minimum-loss problem can be solved by means of the optimal power flow (OPF) program.

### 4.1.1 Equations

Figure 4.1 shows the configuration that will be used in this project. This system consists of  $N$  thermal-generating units connected to a single bus-bar serving a received electrical load  $P_{load}$ . The input to each unit, shown as  $F_i$ , represents the cost rate of the unit. The output of each unit,  $P_i$ , is the electrical power generated by that particular unit. The total cost rate of this system is the sum of the costs of each of the individual units.

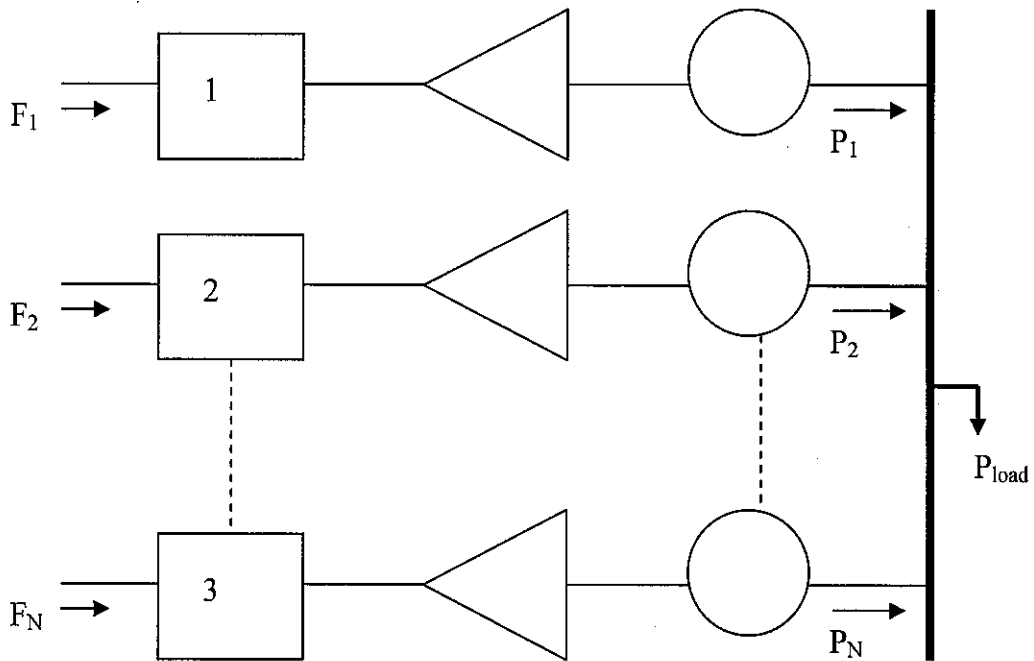


Figure 4.1:  $N$  thermal units committed to serve a load of  $P_{load}$ .

The essential constraint on the operation of this system is that the sum of the output powers must equal the load demand.

$$F_T = F_1 + F_2 + F_3 + \dots + F_N \quad (4.1)$$

$$P_T = P_1 + P_2 + P_3 + \dots + P_N \quad (4.2)$$

Considering this project, any transmission losses are neglected and any operating limits are not explicitly stated. To explain more easily on the unit commitment concept, a fixed start-up cost for each generator was assumed.

The incremental fuel cost of the unit in dollars per megawatthour is  $df_i / dP_i$ , whereas the average fuel cost in the same units is  $f_i / P_i$ . Hence, if the input-output curve of unit  $i$  is quadratic, we can write

$$f_i = (a_i / 2) P_i^2 + b_i P_i + c_i \quad \$ / h \quad (4.3)$$

and the unit has incremental fuel cost denoted by  $\lambda_i$ , which is defined by

$$\lambda_i = df_i / dP_i = a_i P_i + b_i \quad \$ / MWh \quad (4.4)$$

where  $a_i$ ,  $b_i$  and  $c_i$  are constant. The approximate incremental fuel cost at any particular output is the additional cost in dollars per hour to increase the output by 1 MW. Actually, incremental cost is determined by measuring the slope of the input-output curve and multiplying by cost per Btu in the proper units. Since mills (tenths of a cent) per kilowatthour is a very small amount of power in comparison with the usual output of a unit of a steam plant, incremental fuel cost may be considered as the cost of fuel in mills per hour to supply an additional kilowatt output.

We now have the background to understand the principle of economic dispatch which guides distribution of load among the units within one or more plants of the system. For instance, suppose that the total output of a particular plant is supplied by two units and that the division of load between these units is such that the incremental fuel cost of one is higher than that of the other. Now suppose that some of the load is transferred from the unit with the higher incremental cost to the unit with lower incremental cost. Reducing the load on the unit with the higher incremental cost will result in a greater reduction of cost than the increase in cost for adding the same amount of load to the unit with lower incremental cost. The transfer of load from one to the other can be



continued with a reduction in total fuel cost until the incremental fuel costs of the two units are equal. The same reasoning can be extended to a plant with more than two units.

Thus, for economical division of load between units within a plant, the criterion is that all units must operate at the same incremental fuel cost.

When the incremental fuel cost of each of the units in a plant is nearly linear with respect to power output over a range of operation under consideration, equations that represent incremental fuel costs as linear functions of power output will simplify the computations. An economic dispatch schedule for assigning loads to each unit in a plant can be prepared by:

- 1) Assuming various values of total plant output.
- 2) Calculating the corresponding incremental fuel cost  $\lambda$  of the plant.
- 3) Substituting the value of  $\lambda$  for  $\lambda_i$  in the equation for the incremental fuel cost of each unit to calculate its output.

A curve of  $\lambda$  versus plant load establishes the value of  $\lambda$  at which each unit should operate for a given total plant load.

Example, for a plant with two units operating under economic load distribution the  $\lambda$  of the plant equals  $\lambda_i$  of each unit, and so;

$$\lambda_1 = df_1 / dP_1 = a_1P_1 + b_1 \quad (4.5)$$

$$\lambda_2 = df_2 / dP_2 = a_2P_2 + b_2 \quad (4.6)$$

Solving for  $P_1$  and  $P_2$ , we obtain

$$P_1 = (\lambda - b_1) / a_1 \quad (4.7)$$

$$P_2 = (\lambda - b_2) / a_2 \quad (4.8)$$

Adding together these results and then solve for  $\lambda$  give

$$\lambda = a_T P_T + b_T \quad (4.9)$$

$$\text{where} \quad a_T = \left( \sum_{i=1}^2 \frac{1}{a_i} \right)^{-1} \quad (4.10)$$

$$b_T = a_T \left( \sum_{i=1}^2 \frac{b_i}{a_i} \right) \quad (4.11)$$

$$\text{and} \quad P_T = (P_1 + P_2) \quad (4.12)$$

is the total plant output. For instance, if the plant has  $K$  units operating on economic dispatch, then the coefficients of equation above are given by;

$$a_T = \left( \sum_{i=1}^K \frac{1}{a_i} \right)^{-1} = \left( \frac{1}{a_1} + \frac{1}{a_2} + \dots + \frac{1}{a_K} \right)^{-1} \quad (4.13)$$

$$b_T = a_T \left( \sum_{i=1}^K \frac{b_i}{a_i} \right) = a_T \left( \frac{b_1}{a_1} + \frac{b_2}{a_2} + \dots + \frac{b_K}{a_K} \right) \quad (4.14)$$

and the total plant output is given by

$$P_T = (P_1 + P_2 + \dots + P_K) \quad (4.15)$$

### 4.1.2 Calculation

To explain more on the calculation of unit commitment using all the equation above, refer to the example below. In this example it demonstrates only 4 generators. The specifications of all generators used in this example were taken from Power System Analysis book. [1]

Generating unit number	<u>Loading limits</u>		<u>Fuel cost parameters</u>		
	Min (MW)	Max (MW)	$a_i,$ $\left( \frac{\$ / (MW)^2}{h} \right)$	$b_i,$ (\$/MWh)	$c_i,$ (\$/h)
1	100	625	0.0080	8.0	500
2	100	625	0.0096	6.4	400
3	75	600	0.0100	7.9	600
4	75	500	0.0110	7.5	400

Table 4.1: Capacity of generators

$a_i$  = Fuel cost for start-up and shut-down

$b_i$  = Operating fuel cost

$c_i$  = Start-up and shut-down cost

Unit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
2	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
3	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
4	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 4.2: Possible combination of 4 generators

1 = ON

0 = OFF

a. Combination 10: Units 1 and 3 are in operating mode.

$$a_T = (a_1^{-1} + a_3^{-1})^{-1} = (0.0080^{-1} + 0.0100^{-1})^{-1} = 4.44 \times 10^{-3}$$

$$b_T = a_T \left( \frac{b_1}{a_1} + \frac{b_3}{a_3} \right) = a_T \left( \frac{8}{0.0080} + \frac{7.9}{0.0100} \right) = 7.876$$

Incremental fuel cost for the two units at load level of 1100 MW is equal to:

$$\lambda = a_T P_{gT} + b_T = 4.44 \times 10^{-3} (1100) + 7.876 = 12.76 \$ / MWh$$

$$P_{g1} = \frac{\lambda - b_1}{a_1} = \frac{12.76 - 8.0}{0.0080} = 595 MW$$

$$P_{g3} = \frac{\lambda - b_3}{a_3} = \frac{12.76 - 7.9}{0.0100} = 486 MW$$

The hourly production costs of the three units are calculated to be:

$$f_1 = 0.004P_{g1}^2 + 8.0P_{g1} + 500 \Big|_{P_{g1}=595} = \$6676.1 \text{ per hour}$$

$$f_3 = 0.005P_{g3}^2 + 7.9P_{g3} + 600 \Big|_{P_{g3}=486} = \$5620.38 \text{ per hour}$$

The total cost for this combination = **\$12 296.48 per hour**

b. Combination 13: Units 1, 2, and 4 are in operating mode.

$$a_T = (a^{-1} + a^{-1} + a^{-1})^{-1} = (0.008^{-1} + 0.0096^{-1} + 0.011^{-1})^{-1} = 3.1243 \times 10^{-3}$$

$$b_T = a_T \left( \frac{b_1}{a_1} + \frac{b_2}{a_2} + \frac{b_4}{a_4} \right) = a_T \left( \frac{8}{0.008} + \frac{6.4}{0.0096} + \frac{7.5}{0.011} \right) = 7.3374$$

Incremental fuel cost for the three units at load level of 1100 MW is equal to

$$\lambda = a_T P_{gT} + b_T = 3.1243 \times 10^{-3} (1100) + 7.3374 = 10.774 \$ / MWh$$

$$P_{g1} = \frac{\lambda - b_1}{a_1} = \frac{10.774 - 8.0}{0.008} = 374 MW$$

$$P_{g2} = \frac{\lambda - b_2}{a_2} = \frac{10.774 - 6.4}{0.0096} = 456 MW$$

$$P_{g4} = \frac{\lambda - b_4}{a_4} = \frac{10.774 - 7.5}{0.011} = 298 MW$$

The hourly production costs of the three units are calculated to be

$$f_1 = 0.004P_{g1}^2 + 8.0P_{g1} + 500 \Big|_{P_{g1}=347} = \$3758 \text{ perhour}$$

$$f_2 = 0.0048P_{g2}^2 + 6.4P_{g2} + 400 \Big|_{P_{g2}=456} = \$4317 \text{ perhour}$$

$$f_4 = 0.0055P_{g4}^2 + 7.5P_{g4} + 400 \Big|_{P_{g4}=298} = \$3123 \text{ perhour}$$

The total cost for this combination = **\$11 198 perhour**

## 4.2 Visual Basic

In this project, the Visual Basic software was used to solve the Unit Commitment problem. VB is a powerful programming software, easy to learn and very user friendly. The functions of VB are same like using other windows program. VB provides a set of visual objects that can be drawn easily onto a window (called a form). These controls eliminate the need to develop the code to construct the visual interfaces. The layout of the windows that contain the controls can be changed easily by dragging and dropping the controls to a new location, without necessitating a change in the code. Thus, the process for program development and revision becomes much easier and requires much less time and effort.

Visual Basic is an event-driven. An event-driven program does not dictate the sequence of operations. The user can instruct the computer to perform whatever operations the program is capable of, in any sequence he or she desires. This offers the user flexibility. Any changes in the sequence of operations will not call for revising the program. In this sense, an event-driven program is easier to develop and requires fewer revisions.

### 4.2.1 Steps in program development

In developing an application program, there is a fairly standard set of steps that must be followed in order to make sure the progress of the programming is smoother. These steps can be outlined as follows:

1. *Analyze and define the problem.* The first requirement of a program is that it must meet the needs of the applications. Thus, a clear understanding of the problem and goals is the first step in developing the program. Only when the needs and requirements of the program are clearly understood can we determine how the program is to look and act.
2. *Design the visual interface.* Based on analysis and understanding of the problem, one will be able to design the visual interface for the program and start to work with the VB Integrated Development Environment (IDE). At this stage, we will need to decide what data fields should appear on a form. This process can become quite involved. VB provides various visual objects (controls) that can be used to represent data fields. It takes a careful analysis to determine which VB controls will be the best given the nature of data field.
3. *Define user-program interaction.* The user interface consists of the visual aspect and the behavior in which the program responds to the user's actions and to what happens in the computer internally. We will need to determine what our program should do in detail. The user's actions and system activities are recognized as events. User actions that can trigger events include such things as pressing keys, clicking a control, or making a selection from a menu. System activities can also trigger events. Examples of these activities include loading and unloading a form. We must first be aware what events will be triggered when each of these actions occurs. Based on how we decide our program should react, we will place the code in the pertinent event to respond accordingly.

4. *Design the code structure.* On appearance, the code we will develop to respond to an action should be placed in the event that the action triggers. Thus, our code structure will simply be dictated by the responses we want our program to carry out. In reality, however, it can be much more complex. We will be introduced to various complex situations. Suffice it to say, it pays to analyze the complex situation thoroughly before writing any code. As the program grows into multiple **modules** (a module is a code window that contains code), we will also discover that many code blocks can be shared. Thus, we must choose the appropriate module in which to place these blocks. The design of the code structure can have far reaching implications on the maintainability of the code. The importance of this design phase cannot be overemphasized.
  
5. *Write code.* Based on the design, we will then develop the code to perform the activities that the program requires. In addition to ensuring that the code performs what is called for, particular attention should be pay to the coding style. This includes the mechanical aspects of indenting and following the naming conventions.
  
6. *Test and edit the program.* Testing and editing code constitute most of the effort in the coding activity. To minimize the possibility of encountering “mysterious” logic errors, the code were breaks into small steps and run test frequently. This makes it easier to identify the range of code that causes the error. A smaller number of statements make the error source easier to track down and correct. Program can have various kinds of errors:
  - a. It can have syntax errors, resulting from the failure to follow the rules to put various code elements together.
  
  - b. It can also have semantic errors, resulting from the difference between what the programmer codes and what he or she actually means (the failure to say what the programmer means). For example, the



programmer may code a “print” statement thinking it will output on a printer. However, the statement actually means to display output on a form.

- c. Finally, it can have logic errors, resulting from the difference between what the programmer believes a block of code will do and what the program actually does. This type of error is the trickiest and can take days or even weeks to resolve in some complex situations. Thus, in most cases, we will test run the program and discover some unexpected problems or results. We will then modify the code to solve the problems we have identified and test it again. The process was then repeat until no problem or unexpected result is encountered.
7. *Place the program into “production”*. After a program is thoroughly tested, it is ready to be placed in actual use. A program that works with live data and produces “real” results is called a production program. When we are developing and testing the program, we work in the VB IDE. A program to be placed in production should be compiled to produce a separate object program. This object program (an executable file) can then run without the IDE.

### **4.3 Discussions**

Mainly, this simulation software consists of four (4) forms:

1. Generator data form
2. Search form
3. Consumer form
4. Result form

Below is the layout view of the forms using the interactive graphical user interface (GUI) in the Visual Basic software. These forms were design using the toolbox provided in the VB software.

Generator Data

Database Generator Form

Generator Number: 11

Minimum Load Capacity (MW): 100

Maximum Load Capacity (MW): 625

Operating Fuel Cost (\$/MWh): 8.0

Startup and Shutdown Time ((\$/MW)(2/h)): 0.0080

Startup and Shutdown Cost (\$/h): 500

Buttons: First Record, Previous, Next, Last Record, New, Delete, Save, Search, Result, Close

Figure 4.2: Form for generator data entry

Search

Generator Number: 12

OK

Figure 4.3: Form for generator data searching

Figure 4.4: Form for consumer load demand data entry

GENERATOR CHARACTERISTIC DATA

Generator Number	Minimum Capacity	Maximum Capacity
5	200	750
7	200	750
8	150	700
9	75	500
10	100	625
11	100	625
12	200	750

GENERATOR RESULTS DATA

No.	Load Demand	IncrFuel

00000000000000  
00000000000001  
00000000000010  
00000000000011  
00000000000100  
00000000000101  
00000000000110  
00000000000111  
00000000001000  
00000000001001  
00000000001010

Back

Figure 4.5: Form for results

All the forms shown above was designed using command buttons and text boxes in order to make this project more interactive and user friendly. A **text box** is a VB control that enables user to enter any type of data. The text box control has a property called “text”. We can set this property to any text, which will then be displayed at a runtime. Text box also allows the user to enter and edit its contents at runtime. The text so obtained can then be further processed or simply saved for future use. **Command button** is the click event procedures most often seen, whose caption typically indicates to the user what to expect. For example, in this project, users can click “Last Record” command button to display the specification of the last generator in the database. The coding for “Last Record” command button and the text boxes are stated as below:

```
Private Sub cmdLast_Click()  
    With Data1.Recordset  
        .MoveLast  
        txtNo.Text = .Fields(0)  
        txtMin.Text = .Fields(1)  
        txtMax.Text = .Fields(2)  
        txtFuel.Text = .Fields(3)  
        txtTime.Text = .Fields(4)  
        txtCost.Text = .Fields(5)  
    End With  
End Sub
```

The coding for interface form in Figure 4.2 until Figure 4.5 is attached in *Appendix 1*.

**Data control** and **ActiveX data objects (ADO) data control** is the main function in this project. The data control is a two-way street; not only does it display database data, but user can modify the data that the Data control displays and the Data control makes sure that the changes are made to the underlying database through bound control. If we don't want user to be able to change data displayed from a Data control, we can use a

label and not a text box to display the database data. We can bind several other controls to the Data control and make the control read-only so that the user cannot change the underlying database. The procedure for setting the Data control properties can be seen as follows:

a. Connect property setting



Figure 4.6: Data control button

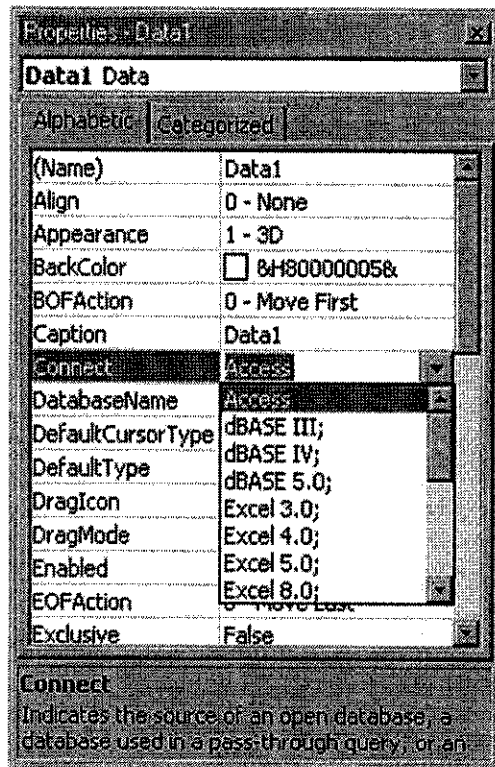


Figure 4.7: Properties window for Data control

In this project, the Data control was connected to the Access database because we are using the Microsoft Access Database.

b. Setting the database name property

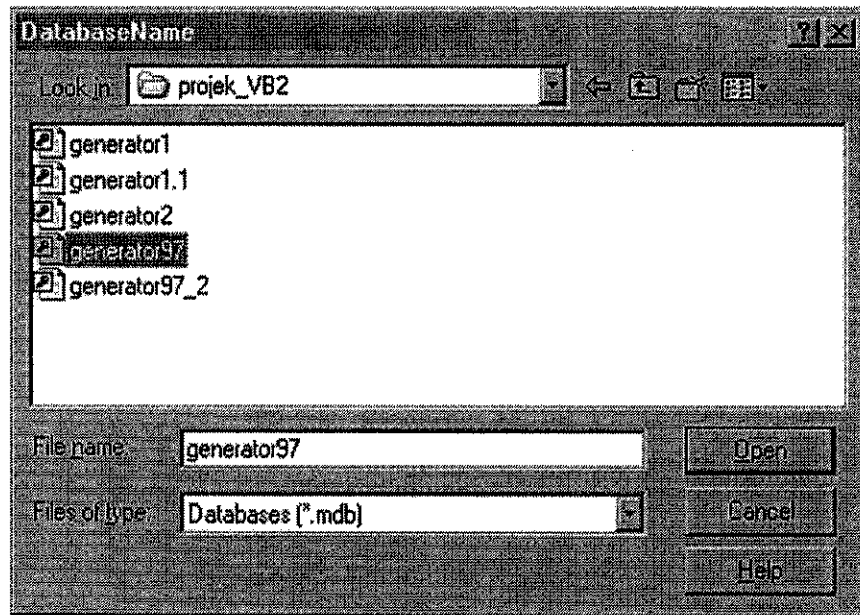


Figure 4.8: DatabaseName dialog box

Dialog box in Figure 4.8 above will occur when we click on the Database Name in the properties window.

c. Record source property setting

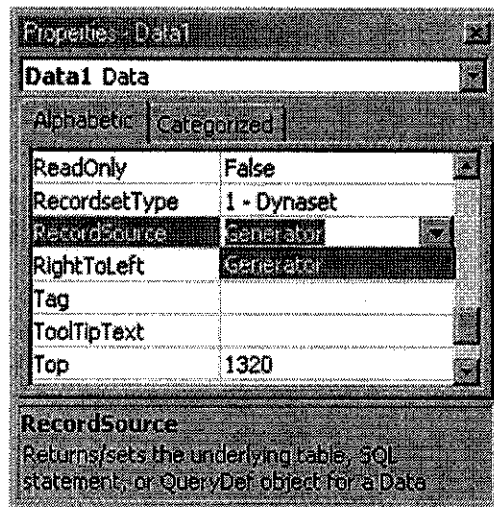


Figure 4.9: Record source property window

This procedure can be done by clicking on the record source property and choose the table name matching with the database name.

ActiveX data objects (ADO) data control is another technology introduces by VB. ADO can be used to handle not only the database (whether local or remote) but also various data “stores” (data that are stored in any form, not just the database format). These objects provide a standard programming interface to develop the code in handling data. The ADO provides a uniform set of interfaces (properties, events, and methods) that can be use to handle all kinds of data. From VB program, we can use the ADO to access the database via the ADO data control or by code directly. The ADO data control (ADODC) provides features that enable user to interact with the underlying database with bound controls and code. A VB control can be bound to data through the ADODC if that control has the DataSource property. Below are the procedures for setting the properties of ADODC:

- a. Before adding this data control to the form, we must first add the ADODC function by clicking on the “Project” toolbars and choose “Component” or just simply press “Ctrl+T” on the keyboard. Then components window as in Figure 4.11 below will appear. Click on the box provided to add the Microsoft ADO Data Control 6.0 (OLEDB) controls into the project.



Figure 4.10: ADODC button

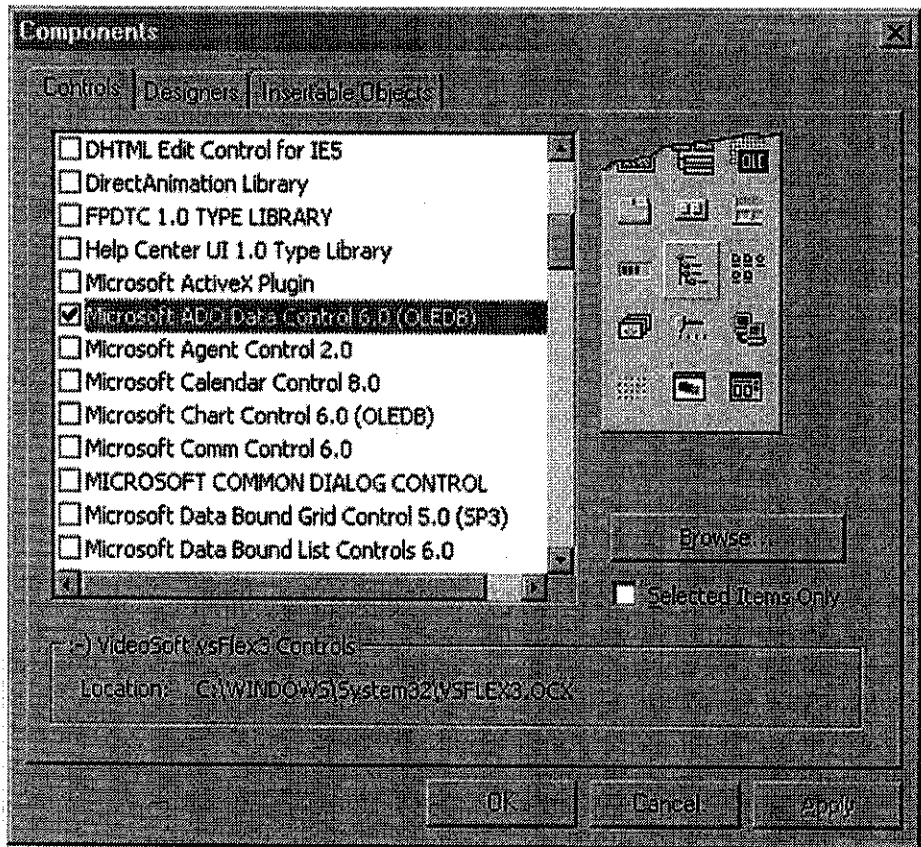


Figure 4.11: Components window

- b. Right click on the ADODC button and choose ADODC properties, then the property page as in Figure 4.12 below will appear. Choose the “Use connection string”, click the “Build” button and set the Microsoft Jet 4.0 OLE DB Provider as the data link properties. Click on the “Next” button, select the database name and test the connection between ADODC button and the database. If the test connection was succeeded, click “OK” button and precede to the Record source properties. Property pages as Figure 4.13 will then appear. Here we select the table using Structures Query Language (SQL) command.



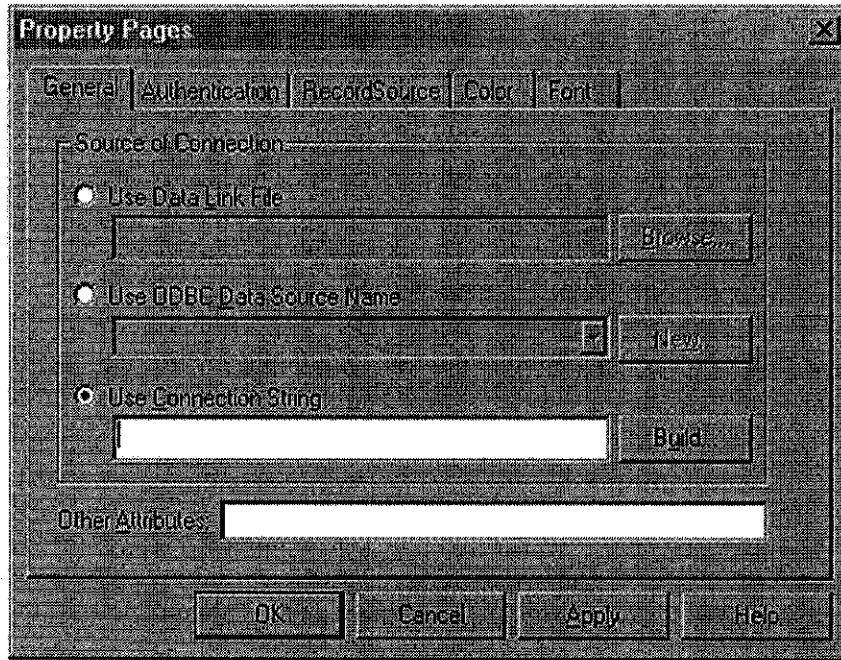


Figure 4.12: ADODC property pages

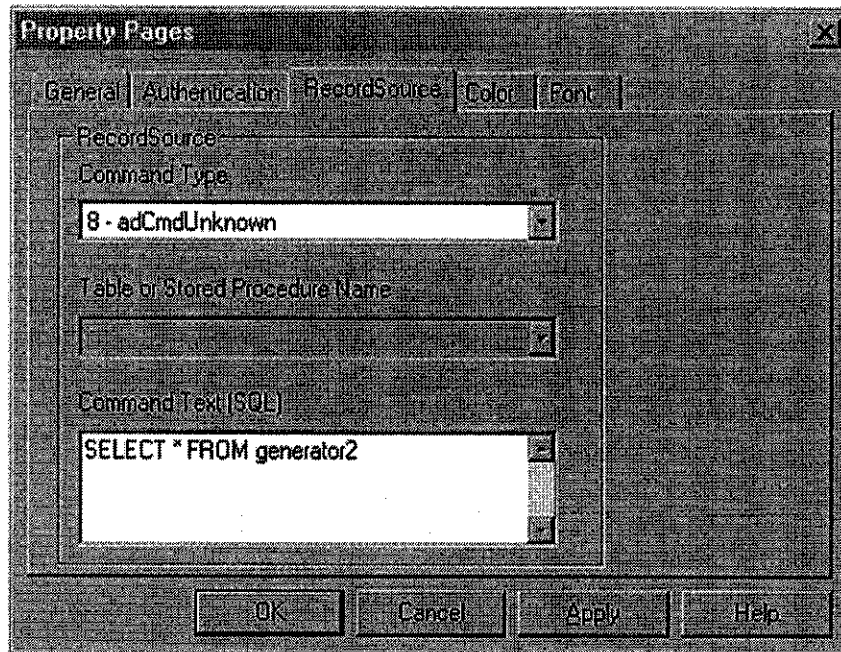


Figure 4.13: Record source for ADODC property pages

Another important feature in this project is the database. A database system is a program that organizes, manipulates, retrieves and reports data. Using database as a data, we can take advantage of Visual Basic's Data control to access the database from within the Visual Basic applications. The Data control makes it easy to retrieve data and display values from a database file without using any or VB's specific file-related commands. The Data control makes database access simple. A field is a column of data inside a file. A database application manages the data in a record and field format. The database however, doesn't necessarily store data in records and fields in a table-like format, but the database makes the data appear to program in that format. VB takes advantage of this format and retrieves data in the record and field format no matter how the database physically stores the data.

GeneratorNumber	MinimumCapacity	MaximumCapacity	OperatingFuel	Shuttime	ShutCost
1	100	625	8.0	0.0080	500
2	100	625	6.4	0.0096	400
3	75	600	7.9	0.0100	600
4	75	500	7.5	0.0110	400

Figure 4.14: Data fields in the database of generators

One challenge when using access is that we must often describe parts of the database to VB. VB cannot magically understand the database structure. When placing the Data control on the form, we'll have to tell the control the structure of the data and tell the Data control which parts of the data to access so that the control can properly retrieve data. For example, by setting appropriate property values, we must tell the Data control the name of database, the table and the fields to access.

A table is a logical collection of data in a database. A database might contain several tables. Some databases, such as Microsoft Access, store all the related database files in a single global file called the database file. Inside the database, the individual groups of records and fields are called tables. Other database systems, such as dBASE, keep track

of a database's data in multiple files. When we use a database such as Microsoft Access, we must describe both the overall database and the individual table name within the database that the Data control is to use.

A bound control is a control we can link to a database, via the Data control, that displays and updates database records if the user modifies the data in the bound control. In most database applications, code is required. If the user is to add new records and delete old ones, for example, code is needed. For simple displaying and updating of existing data, however, the Data control, labels and text boxes can do all the work. Below is the example of coding to delete the records.

```
Private Sub cmdDelete_Click()
    On Error GoTo DeleteErr
    With Data1.Recordset
        .MoveFirst
    Do While Not .EOF
        If .Fields(0) = txtNo Then
            .Delete
            Data1.Refresh
            Call cmdNew_Click
            a1 = MsgBox("Data has been remove", vbInformation)
        Exit Sub
    Else
        .MoveNext
    End If
    Loop
    End With
    Exit Sub
DeleteErr:
    MsgBox Err.Description
End Sub
```

## CHAPTER 5

### CONCLUSION AND RECOMMENDATIONS

#### 5.1 Conclusion

Unit commitment problem is a problem that must be frequently solved by a power utility company to economically determine a schedule of what combinations units will be used to meet the forecast demand and operating constraints such as spinning reserve requirements, over a short time horizon. In this project we are going to redesign the existing Visual Basic programming so that it can run and simulate more than 10 generators. In order to achieve this goal, the understanding in Visual Basic and knowledge in Unit Commitment are the main requirement. During the first semester of Final Year Design Project, the author is concentrating more on the literature review and gaining knowledge in both Unit Commitment and Visual Basic. For Visual Basic, the author learns on how to use it by referring to reference book and also learn from Information Technology (IT) student. In the first stage, authors rewrite the VB coding done by previous student and try to run it. During first stage, there are many errors occur in simulating the previous programming. Author overcomes the problem by referring to the Visual Basic expert form IT department.

Visual Basic software is suitable to use in solving the Unit Commitment problem because it is a powerful programming and easy to learn. The way we use VB software is similar to when we are using normal Windows program.

## **5.2 Recommendations**

In solving the Unit Commitment problem, we are not bounded to use only the Visual Basic software. There are many other programming software that can be used such as Visual Basic .NET, Microsoft Visual C++, and Macromedia Dream Weaver.

For the second recommendation, we can use another type of database software besides the Microsoft Access 97. In this project we only involve in simple and small data fields, so for more complex or big database, we can apply more powerful database software like Microsoft Access latest version, SQL Server, FoxPro, Dbase, and Btrieve.

## REFERENCES

### Secondary Sources

- [1] *Power System Analysis*, John J. Grainger and William D. Stevenson, Jr., McGraw Hill International Editions
- [2] *Power Generation Operation and Control*, Allen J. Wood and Bruce F. Wollenberg, Wiley-Interscience
- [3] *Teach Yourself Visual Basic 6*, Scott Warner, Osborne, McGraw Hill
- [4] *Visual Basic 6 Programming*, Blue Book, Peter G. Aitken, Corolis
- [5] *Visual Basic 6 Black Book*, Steven Holzner, Corolis
- [6] *Learning Visual Basic Through Applications*, Clayton E. Crooks II, Charles River Media
- [7] *Visual Basic 6*, Greg Perry with Sanjaya Hettihewa, SAMS Publishing
- [8] *Visual Basic 6 from the Ground Up*, Gary Cornell, McGraw Hill
- [9] *Visual Basic 6 Programming*, Business Application With A Design Perspective, Jeffrey J. Tsay, Prentice Hall
- [10] *Visual Basic "An Object Oriented Approach"*, McMonnies, Alistair, Pearson Education Limited

- [11] *Power System Analysis and Design* (Second Edition), Glover, J.D, PWS Publishing Company, Boston (1994)
- [12] *Control of Generation and Power Flow on Interconnected System*, Cohn N., John Wiley & Sons, Inc (1986)
- [13] *Power System Analysis*, Bergen, A.R., Prentice Hall

### **Internet**

- [1] <http://lemons.inescn.pt/artigo41.pdf>
- [2] <http://www.eepe.swan.ac.uk/upec/programme/abstract/a394.pdf>
- [3] [http://www.ima.mdh.se/tom/tom-papers/co-PSCC\\_1999\\_trondheim.ps](http://www.ima.mdh.se/tom/tom-papers/co-PSCC_1999_trondheim.ps)

**Coding for Generator Data form**

```
Private Sub cmdClose_Click()  
    Unload Me  
End Sub
```

---

```
Private Sub cmdDelete_Click()  
  
    On Error GoTo DeleteErr  
  
    With Data1.Recordset  
        .MoveFirst  
  
        Do While Not .EOF  
  
            If .Fields(0) = txtNo Then  
                .Delete  
                Data1.Refresh  
  
                Call cmdNew_Click  
                a1 = MsgBox("Data has been remove", vbInformation)  
                Exit Sub  
  
            Else  
                .MoveNext  
  
            End If  
  
        Loop  
    End With  
  
    Exit Sub  
  
DeleteErr:  
    MsgBox Err.Description  
  
End Sub
```

---

```
Private Sub cmdFirst_Click()  
    With Data1.Recordset  
        .MoveFirst  
        txtNo.Text = .Fields(0)  
        txtMin.Text = .Fields(1)
```



## APPENDIX I

```
txtMax.Text = .Fields(2)
txtFuel.Text = .Fields(3)
txtTime.Text = .Fields(4)
txtCost.Text = .Fields(5)
End With
End Sub
```

---

```
Private Sub cmdLast_Click()
With Data1.Recordset
.MoveLast
txtNo.Text = .Fields(0)
txtMin.Text = .Fields(1)
txtMax.Text = .Fields(2)
txtFuel.Text = .Fields(3)
txtTime.Text = .Fields(4)
txtCost.Text = .Fields(5)
End With
End Sub
```

---

```
Private Sub cmdNew_Click()

On Error GoTo AddErr

txtNo.Text = ""
txtMin.Text = ""
txtMax.Text = ""
txtFuel.Text = ""
txtTime.Text = ""
txtCost.Text = ""

Exit Sub
```

```
AddErr:
MsgBox Err.Description

End Sub
```

---

```
Private Sub cmdNext_Click()

current_position = Data1.Recordset.AbsolutePosition
Data1.Recordset.MoveLast
total_records = Data1.Recordset.RecordCount
If current_position = total_records - 1 Then
```

## APPENDIX I

```
Exit Sub
Else
    Data1.Recordset.AbsolutePosition = current_position
End If

With Data1.Recordset
    .MoveNext
    txtNo.Text = .Fields(0)
    txtMin.Text = .Fields(1)
    txtMax.Text = .Fields(2)
    txtFuel.Text = .Fields(3)
    txtTime.Text = .Fields(4)
    txtCost.Text = .Fields(5)
End With
End Sub
```

---

```
Private Sub cmdPrevious_Click()
    If Data1.Recordset.AbsolutePosition = 0 Then Exit Sub

    With Data1.Recordset
        .MovePrevious
        txtNo.Text = .Fields(0)
        txtMin.Text = .Fields(1)
        txtMax.Text = .Fields(2)
        txtFuel.Text = .Fields(3)
        txtTime.Text = .Fields(4)
        txtCost.Text = .Fields(5)
    End With
End Sub
```

---

```
Private Sub cmdResult_Click()
    frmConsumer.Show
    Unload Me
End Sub
```

---

```
Private Sub cmdSave_Click()

    On Error GoTo UpdateErr

    With Data1.Recordset
        .MoveFirst
        Do While Not .EOF
            If .Fields(0) = txtNo Then
```

## APPENDIX I

```
    MsgBox "Duplicate No. Data cannot be saved. Please enter new number",
vbCritical
    Exit Sub
    Else
        .MoveNext
    End If
    Loop
End With

If txtNo.Text = "" Or txtMin.Text = "" Or txtMax.Text = "" Or txtFuel.Text = "" Or
txtTime.Text = "" Or txtCost.Text = "" Then
    MsgBox "Please complete the fields properly", vbCritical
    Exit Sub
End If

With Data1.Recordset
    .AddNew

    .Fields(0) = txtNo.Text
    .Fields(1) = txtMin.Text
    .Fields(2) = txtMax.Text
    .Fields(3) = txtFuel.Text
    .Fields(4) = txtTime.Text
    .Fields(5) = txtCost.Text

    .Update
End With

Exit Sub

UpdateErr:
    MsgBox Err.Description

End Sub



---



Private Sub cmdSearch_Click()
    frmSearch.Show
End Sub
```

**Coding for Consumer Load Demand form**

```
Private Sub cmdOK_Click()  
    gConsumerLoad = txtInput.Text  
    frmResult.Show  
    Unload Me  
End Sub
```

**Coding for Search form**

```
Private Sub cmdFind_Click()  
  
    is_datafound = False  
  
    With frmGenerator1.Data1.Recordset  
        .MoveFirst  
        Do While Not .EOF  
            If .Fields(0) = txtGenNo.Text Then  
  
                frmGenerator1.txtNo.Text = .Fields(0)  
                frmGenerator1.txtMin.Text = .Fields(1)  
                frmGenerator1.txtMax.Text = .Fields(2)  
                frmGenerator1.txtFuel.Text = .Fields(3)  
                frmGenerator1.txtTime.Text = .Fields(4)  
                frmGenerator1.txtCost.Text = .Fields(5)  
  
                is_datafound = True  
  
                Exit Do  
            Else  
                .MoveNext  
            End If  
        Loop  
    End With  
  
    If is_datafound = False Then  
        MsgBox "Sorry, record not found", vbInformation  
    End If  
    Unload Me  
End Sub
```

**Coding for Result form**

Option Explicit

Dim mConsumerLoad As Integer  
Dim mGenNo As Integer  
Dim mTotRow As Integer  
Dim Time(1 To 15) As Currency  
Dim Cost(1 To 15) As Currency  
Dim Fuel(1 To 15) As Currency  
Dim pg(1 To 15) As Currency  
Dim f(1 To 15) As Currency  
Dim kira As Integer  
Dim index As Double  
Dim at As Double  
Dim lambda As Currency  
Dim Tot\_Prod\_Cost As Double  
Dim a(15) As Currency  
Dim x As Integer  
Dim b(15) As Currency  
Dim c(15) As Currency  
Dim bt As Currency  
Dim z As Integer

---

Private Sub cmdGenerator1\_Click()

    With adcGen.Recordset  
        Do Until .EOF  
            .MovePrevious  
        Loop  
    End With

frmGenerator1.Show  
Unload Me

End Sub

---

Private Sub Form\_Load()

Dim col As Integer  
x = 15  
mGenNo = adcGen.Recordset.RecordCount

## APPENDIX I

mConsumerLoad = gConsumerLoad

'If adcGen2.Recordset.BOF = False Then

```
' With adcGen2.Recordset
'   .MoveFirst
'   Do Until .EOF
'     .Delete
'     .MoveNext
'   Loop
' End With
'End If
```

GetData  
TotalRow

End Sub

---

Private Sub GetData()

Dim index As Integer  
Dim i As Integer

index = 15

If adcGen.Recordset.BOF = False Then

```
With adcGen.Recordset
  .MoveFirst
  Do Until .EOF
    Fuel(index) = .Fields(3)
    Time(index) = .Fields(4)
    Cost(index) = .Fields(5)
    .MoveNext
    index = index - 1
  Loop
End With
End If
End Sub
```

---

Private Sub TotalRow()

Dim i As Long  
Dim lup As Integer

## APPENDIX I

```
Dim start As Integer
Dim tmp As String
Dim row As Long
Dim balance As Integer
Dim Mx_col As Integer
Dim col As Integer
Dim h As Integer
Dim j As Integer
Dim y As Integer
Dim v As Integer
Dim bits As Integer
Dim ctr As Integer
Dim k As Integer
Dim number As Integer
Dim Total_loops As Currency
```

```
'Generate possible combination
```

```
Total_loops = 2 ^ Total_loops
```

```
For j = 1 To 15
```

```
List1.clear
```

```
    Mx_col = j
```

```
    ReDim s2(2 ^ Mx_col, Mx_col) As String * 1
```

```
    For col = 1 To Mx_col
```

```
        bits = 0
```

```
        ctr = 0
```

```
        For i = 1 To 2 ^ Mx_col
```

```
            s2(i, Mx_col - col + 1) = bits
```

```
            ctr = ctr + 1
```

```
        If ctr = 2 ^ (col - 1) Then
```

```
            If bits = 0 Then
```

```
                bits = 1
```

```
            Else
```

```
                bits = 0
```

```
            End If
```

```
            ctr = 0
```

```
        End If
```

```
    Next i
```

```
Next col
```

```
For row = 1 To 2 ^ Mx_col
```

```
    tmp = ""
```

```
    For i = 1 To Mx_col
```

## APPENDIX I

```
        tmp = tmp & s2(row, i)
    Next i
    List1.AddItem tmp
Next row
SendKeys "{END}+{HOME}"
Next j

End Sub
```

---

```
Private Sub calcat()
```

```
Dim v As Integer
Dim sumat As Currency
```

```
'For each c contain value i
```

```
at = a(1) + a(2) + a(3) + a(4) + a(5) + a(6) + a(7) + a(8) + a(9) + a(10) + a(11) + a(12) +
a(13) + a(14) + a(15)
```

```
If at > 0 Then
```

```
    at = 1 / at
```

```
    If x = 15 - kira Then
```

```
        For v = 15 To x
```

```
            sumat = Fuel(x) / Time(x)
```

```
            sumat = sumat + suamt
```

```
            x = x - 1
```

```
        Next v
```

```
    Else
```

```
        sumat = Fuel(x) / Time(x)
```

```
    End If
```

```
    bt = at * sumat
```

```
    lambda = (at * mConsumerLoad) + bt
```

```
    pg(x) = (lambda - Fuel(x)) / Time(x)
```

```
    f(x) = ((Time(x) / 2) * pg(x) ^ 2) + (Fuel(x) * pg(x)) + (Cost(x))
```

```
    Tot_Prod_Cost = Tot_Prod_Cost + f(x)
```

```
End If
```

```
End Sub
```

---



## APPENDIX I

Private Sub insertdb()

'input to database

Dim row As Long

Dim s2(32768, 40000) As Long

With adcGen2.Recordset.AddNew

adcGen2.Recordset("LoadDemand") = mConsumerLoad

adcGen2.Recordset("IncreFuel") = lambda

!C1 = s2(1, 1)

!C2 = s2(2, 1)

!C3 = s2(3, 1)

!C4 = s2(4, 1)

!C5 = s2(5, 1)

!C6 = s2(6, 1)

!C7 = s2(7, 1)

!C8 = s2(8, 1)

!C9 = s2(9, 1)

!C10 = s2(10, 1)

!C11 = s2(11, 1)

!C12 = s2(12, 1)

!C13 = s2(13, 1)

!C14 = s2(14, 1)

!C15 = s2(15, 1)

!PG1 = pg(1)

!PG2 = pg(2)

!PG3 = pg(3)

!PG4 = pg(4)

!PG5 = pg(5)

!PG6 = pg(6)

!PG7 = pg(7)

!PG8 = pg(8)

!PG9 = pg(9)

!PG10 = pg(10)

!PG11 = pg(11)

!PG12 = pg(12)

!PG13 = pg(13)

!PG14 = pg(14)

!PG15 = pg(15)

!F1 = f(1)

!F2 = f(2)

!F3 = f(3)

!F4 = f(4)

## APPENDIX I

```
!F5 = f(5)
!F6 = f(6)
!F7 = f(7)
!F8 = f(8)
!F9 = f(9)
!F10 = f(10)
!F11 = f(11)
!F12 = f(12)
!F13 = f(13)
!F14 = f(14)
!F15 = f(15)
!TotalProductionCost = Tot_Prod_Cost
```

```
adcGen2.Recordset.Update
adcGen2.Recordset.MoveNext
```

```
End With
End Sub
```

---

```
Private Sub clear()
```

```
at = 0
bt = 0
Tot_Prod_Cost = 0
lambda = 0
```

```
For z = 1 To 15
a(z) = 0
```

```
Next z
End Sub
```