

University of Jordan
Faculty of Graduate Studies

19
002


DESIGN AND CONSTRUCTION OF A PROTOTYPE
PROGRAMMABLE LOGIC CONTROLLER .

BY

Mohammad Usama Kamal Tahboub

SUPERVISOR


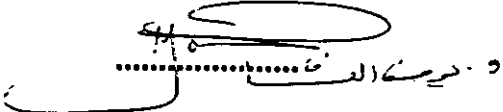
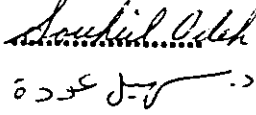
Dr. Khaled Toukan

معيد كلية الدراسات العليا


SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE IN INDUSTRIAL ENGINEERING ,
FACULTY OF GRADUATE STUDIES , UNIVERSITY OF JORDAN .

January , 1994

This thesis was defended successfully on January 10 , 1994 .

<u>Committee Names</u>	<u>Signature</u>
1- Dr. Khaled Toukan	
2- Dr. Yousef Al-assaf	
3- Dr. Souheil Odeh	

DEDICATION

اهـداء

الى أمي التي وهبتني الكثير ...

ACKNOWLEDGEMENTS

شكر و تقدير

الحمد لله أولا و آخرا على نعمائه و توفيقه .

أود أن أتقدم بالتقدير و الاحترام و الشكر الجزيل للدكتور خالد طوقان الذي أبدى اهتماما صادقا و جهدا مشكورا خلال مراحل العمل المختلفه ، كما أتقدم بالشكر للدكتور يوسف العساف و الدكتور سهيل عوده على حسن اهتمامهما و تقديرهما لهذا العمل . أود أن أتقدم بالشكر أيضا لكل من الدكتور خلدون طهبوب و الدكتور عبد الرحمن جرادات و الدكتور صباح البيرماتي لما أبدوا من ملاحظات مفيدة واهتمام مشكور . كما أود أن أشكر المهندس كمال عمار و المهندس سامح عليان وللسيد محمد فوزي و كل من أبدى جهدا و اهتماما ساعد على انجاز هذه الرسالة .

و أخيرا أتقدم بالشكر لعائلتي التي وقفت الى جانبي منذ اللحظات الأولى تدعمني و تحفزني و تدفعني الى الأمام .

CONTENTS	PAGE
Committee Decision	ii
Dedication	iii
Acknowledgments	iv
Contents	v
List of Tables	viii
List of Figures	x
Abstract (In English)	xii
Chapter One : Introduction	1
1.1. Historical Background	1
1.2. PLCs in Jordan	5
Chapter Two : PLC Structure	10
2.1. Central Processing Unit	11
2.1.1. The Processor	11
2.1.2. The Memory	13
2.1.3. The Power Supply	17
2.2. The Input / Output System	18
2.2.1. Discrete Inputs / Outputs	18
2.2.2. Numerical Inputs / Outputs	21
2.2.3. Special Inputs / Outputs	22
2.2.4. Remote Inputs / Outputs	24
Chapter Three : PLC Products and Applications	25
3.1. PLC Families	25
3.2. Defining the Control System	27
3.3. Steps to Implement a PLC System	30
3.4. PLC Applications	33

CONTENTS	PAGE
3.4.1. Rubber and Plastic	34
3.4.2. Chemical and Petrochemical	35
3.4.3. Power	35
3.4.4. Metals	36
3.4.5. Materials Handling	37
3.4.6. Automotive	38
3.4.7. Manufacturing / Machining	38
3.4.8. Examples	39
Chapter Four : Hardware Design	42
4.1. The Central Processing Unit (CPU)	42
4.2. Input / Output (I/O) Interface	43
4.2.1. Input / Output Interface Card	44
4.2.1.1. Buffer Stage	45
4.2.1.2. Decoding Stage	47
4.2.1.3. Peripheral Devices Stage ..	48
4.2.2. Power Amplification Card	60
4.3. Hardware Set Up	71
4.4. Reading The Inputs	72
4.5. Updating The Outputs	73
Chapter Five : Software Design	75
5.1. Programming Language Used	75
5.2. Menu System	75
5.3. Boolean Language	76
5.4. User Program Translation	77
5.5. Running User Program	79
5.5.1. Memory Configuration and Peripheral	

CONTENTS	PAGE
Devices Set Up	80
5.5.2. Scanning	82
5.5.2.1. Reading The Inputs	82
5.5.2.2. Executing The User Program	83
5.5.2.3. Updating The Outputs	85
5.6. Boolean Language Rules	86
5.7. Programs' Listing	89
Chapter Six : Testing and Equipment	90
6.1. PLC Components	90
6.2. Cost Estimation	90
6.3. Testing Basic Operation	93
6.4. Traffic Light System	93
6.5. Conclusion	94
APPENDIX A	96
APPENDIX B	103
APPENDIX C	113
APPENDIX D	118
APPENDIX E	131
References	140
Abstract (In Arabic)	142

List of Tables

1.1	Typical PLC Features / Benefits	6
2.1	Discrete Input / Output Devices	19
2.2	Standard I/O Voltage Ratings of Discrete I/O Interfaces	20
3.1	Steps for Selecting the Right PLC	31
4.1	Components of the Interface Card	45
4.2	8255A Basic Operation	52
4.3	Mode 0 Port Definition of the PPI	55
4.4	Register Selection Table	58
5.1	Boolean Language (Mnemonics and Operators) .	78
5.2	Sizes (in Bytes) of Programs Developed	89
6.1	Components Listing	91
A.1.1	Typical Standard Features of Small PLCs	96
A.2.1	Typical Standard Features of Medium PLCs	97
A.3.1	Typical Standard Features of Large PLCs	99
A.4.1	Typical Standard Features of Very Large PLCs	101
B.1.1	Function Table of the 74LS245	104
B.1.2	Recommended Operating Conditions of the 74LS245	104

B.2.1	Recommended Operating Conditions of the 74LS244	106
B.3.1	Function Table of the 74LS85	108
B.3.2	Recommended Operating Conditions of the 74LS85	109
B.4.1	Function Table of the 74LS138	111
B.4.2	Recommended Operating Conditions of the 74LS138	112
C.2.1	Data Sheets of the MOC3031 Opto-Coupled Triac	117

List of Figures

2.1	Block Diagram of the PLC	10
2.2	Scanning	11
2.3	A PLC Memory Map	14
2.4	Memory Area Table of the Data Table	16
2.5	Block Diagram of the AC/DC Input Circuit	21
3.1	Illustration of the PLC Product Range	26
3.2	Control Type Configuration	29
4.1	Block Diagram of the I/O Module	43
4.2	Schematic Diagram of the I/O Interface Card ..	46
4.3	Block Diagram of the PPI	50
4.4	Mode Definition Format of the PPI	54
4.5	Block Diagram of the 8253 Timer / Counter	56
4.6	Control Word Format of the PPI	59
4.7	Timing Diagram for Mode 0 Operation	61
4.8	MOC3031 Opto-Coupled Triac	62
4.9.a	Triac Symbol	63
4.9.b	Triac Structure	63
4.10	Voltage-Current Characteristics of the Triac .	64
4.11	Modes of Operation of the Triac	66
4.12	Generalized Power Converter Structure	66
4.13	Power Amplification Wiring	70
5.1	Menu System Options	76
5.2	Implementation Stages of a User Program	81
5.3	Boolean Language Syntax Diagrams	87

B.1.1	Pin Diagram of the 74LS245	103
B.2.1	Pin Diagram of the 74LS244	105
B.3.1	Pin Diagram of the 74LS85	107
B.4.1	Pin Diagram of the 74LS138	110

ABSTRACT

Design and Construction of a Prototype Programmable Logic Controller

*By : Mohammad Tahboub
Supervisor : Dr. Khaled Toukan*

In this thesis , a step towards increasing the national ' Know - How ' of Programmable Logic Controllers (PLCs) is undertaken . The concept of operation of the PLC is , first , described ; a discussion of the major components , different features and fields of application is also included . A prototype PLC is , then , designed and constructed through two parallel , interrelated procedures : hardware design and software development . A prototype PLC is designed and constructed using the CPU of a personal computer , to provide for the processor and power supply of the PLC , and an I/O interface module that provides twelve discrete output signals at 115 VAC and twelve discrete input signals at TTL level . The I/O interface module consists of two major parts : the I/O interface card and the DC / AC power converter . A Boolean Language environment is developed using Turbo Pascal (version 6) . The developed environment provides the user with the ability to write user programs in Boolean Language , translate the user program into Pascal , compile the translated Pascal code , and run the compiled programs . Running a user program results in two main activities : 1) set up of the prototype PLC and 2) performing continuous scanning . Thorough testing of the prototype PLC is performed at both TTL level and 115 VAC level to check for proper operation .

Chapter One

Introduction

1.1. Historical Background

Relay controlled systems were inflexible and costly .
The need for a system that provides for :

- 1- computer flexibility ,
- 2- easy programming and maintenance ,
- 3- reusability ,
- 4- robustness in the industrial environment , and
- 5- reduction in down time of machines ,

led to the design criteria of the first programmable logic controller (PLC) in 1968 by the Hydramatic division of the General Motors corporation [1] .

The first programmable logic controllers were limited to applications such as transfer lines and grinding machines since they were capable of on/off control only . However , they were easily installed , needed less space and energy than relays , and were reusable .

As time went , great enhancements were made to PLCs :

- 1- Early innovations (1970 - 1974) in microprocessor technology added greater flexibility and intelligence to the PLC . The PLC became capable of interfacing with the operator, manipulating data , performing arithmetic operations , and communicating with computers [1] .
- 2- Later enhancements (1975-1979) in hardware and software provided more flexibility to the PLC . The PLC provided larger memory capacity , remote Input/Output capabilities , analog control capability , operator

communication capability , and user friendly software environments [1] .

Larger memory capacity meant larger application programs and larger amounts of data that could be held and manipulated .

Remote input / output was done by signal multiplexing over two twisted pairs of wires . This made it possible for large systems to be divided into smaller systems ; smaller systems are more flexible and can be maintained easily .

Development of communication capabilities of the PLC made it possible for users to get their PLCs interconnected through Local Area Networks (LANs) in a step towards having a complete flexible manufacturing system .

Software enhancements provided easy utilization of hardware enhancements . Such software enhancements included the use of computer like statements and menu driven software .

431757

3- Later (1980 - 1989) , PLCs were greatly enhanced both in hardware and software capabilities .

Hardware enhancements include the achievement of faster scan times ; availability of special interfaces that allow certain devices to be connected directly to the controller , such as thermocouplers and strain gauges , and existence of a wide range of PLC sizes that provides the user with a variety of options .

Software enhancements include the use of high level languages such as Basic , diagnostics and fault detection of both the controller and the controlled machine , and support of floating point mathematics which facilitates.

performing complex computations [1] .

4- Nowadays (1990 - 1993) , Program generators are now being developed for PLCs . They help programmers to overcome the complications associated with data addressing and the inconveniences of bit addressing . The programmer has to provide the initial program specifications only and the generator performs the rest of job .

First , the programmer has to identify the devices and switches of the application . Second , he / she has to define the conditions under which each device will change state . Third , the PLC program generator takes over and generates the program code by first translating each state change condition into one or more rungs of the program ladder diagram and , next , assigning valid data addresses to each of the devices / switches defined [2] .

However , the above mentioned code generators do not provide portability of code between the different PLCs ; both the generator and generated code are PLC dependent . Another approach is to develop an automatic PLC program generator with a standard interface ; it aims at improving the portability of PLC programs developed on different PLCs [3] .

Such a generator was developed by Sangecta Bhatnager and Richard T. Lion in 1990 [3] . It consists of six modules , some of which are PLC dependent and some are not . The generator translates the user program specification into a Standard Control Logic Specification (SCLS) that is portable between different PLCs . Next , the generator generates a PLC specific program code to be

executed .

Bidirectional conversion of program code between SCLS specification and executable code is available ; thus , portability is improved .

PLC simulators is another field of concern for the time being . Different microcomputer based simulators of PLCs are developed that provide a safe , reliable testing environment of newly developed user programs . Such simulators are also useful for training purposes .

Testing user programs is time consuming ; on line testing would increase equipment down-time . Moreover , on line testing increases the probability of damage of hardware equipment connected to the PLC . Such disadvantages are overcome by using PLC simulators [4] .

PCL (Programmable Controller Language) is a high level , real-time programming language developed in the Technical University of Istanbul , Turkey . It is syntactically similar to Pascal . It provides for structured data types , efficient control structures , concurrent processing , exception handling , and asynchronous events . It is suitable for complex applications which require critical interrupt handling mechanisms [5] .

Generally speaking , the PLC is an essential part of factory automation . The PLC is , now , part of the integrated system combining robots , Computer Numerically Controlled (CNC) machines , CAD/CAM systems , and Management Information Systems (MIS) [1] .

When the PLC was compared versus relays , some

time ago , a trade-off was made between the cost of the PLC , relative to the relay system cost , and the features provided by the PLC such as flexibility , high reliability , small space requirements , and the ability to accommodate future expansion . However , as the cost of hardware is nowadays decreasing significantly and as the user is provided with the ability to choose the controller that suits him / her out of a wide range of controllers , the PLC appears to be the more probable choice .

Table 1.1 shows some of the PLC features and benefits .

1.2. PLCs in Jordan :

Industrial organizations in Jordan can be categorized into three major types according to the technology they employ :

(a) The first type of organizations uses up-to-date new technology in its production lines ; these represent a small percentage out of the total . They usually suffer from the lack of an expert staff that can support and maintain the technology used .

(b) The industrial organizations of the second type use what is called new-old technology ; this is old technology that is out of date and is no more used in the developed countries' industry ; thus ; they are imported to be used in third world countries . This type represents a good percentage out of the total . Such organizations might face problems with future expansion and lack of spare parts .

(c) The third type of organizations uses reconditioned old technology that usually requires extensive maintenance and

Table 1.1 Typical PLC Features / Benefits [1] .

Inherent Features	Benefits
Solid-State Components	High Reliability
Programmable Memory	Simplifies Changes Flexible Control
Small Size	Minimum Space Requirements
Microprocessor-Based	Communication Capability Higher Level of Performance Higher Quality Products Multi-functional Capability
Software Timers/Counters	Eliminate Hardware Easily Changed Presets
Software Control Relays	Reduce Hardware/Wiring Cost Reduce Space Requirements
Modular Architecture	Installation Flexibility Easily Installed Hardware Purchase Minimized Expandability
Variety of I/O Interface	Controls Variety of Devices Eliminates Customized Control
Remote I/O Stations	Eliminate Long Wire Problems
Diagnostic Indicators	Reduce Trouble-Shooting Time Signal Proper Operation
Modular I/O Interface	Neat Appearance of Control Panel Easily Maintained Easily Wired
All System Variables Stored in Memory	Useful Management / Maintenance Data Report Generation

suffers from the lack of spare parts in both the national and international markets . Future expansion and development might require discarding the old technology and

replacing it with new technology .

In Jordan , there are some authorized dealers of international companies that have PLC products such as AEG / Modicon , Fuji Electric , and Allen-Bradely .

Worldwide speaking , Siemens is number one in the world in the field of PLCs . They have a worldwide marketshare of 25% and a Middle East marketshare of 45% . Their worldwide sales in 92/93 was one Billion US dollars . Allen-Bradely and Mitsubishi are the second and the third in the world , respectively .

AEG / Modicon is an international company that has a good share of the world market . It is a large company that has around 16,500 - 17,000 employees .

In 1991, AEG sales in automation systems were approximately 3000 billion Deutsche marks ; automation systems include real time computers and PLCs as major items. AEG's annual report ensures that the AEG company is the leading supplier of integrated systems in Europe , North America , and South Asia . Foreign countries have a 44% share out of AEG product sales .

AEG's Jordanian authorized dealers have only installed 5-10 systems at different sites in Jordan . These include the Jordan Sewing and Weaving Company , the Sewage Treatment Plant in the Industrial City in Irbid , Al-Sufara Bakery , and the Arab Bank .

There are other sites in Jordan that use AEG's PLCs in automated factories ; however , these PLC sites were imported as parts of complete solutions from Europe and were completely installed by foreign experts . Jordanian

authorized dealers are not referenced except for spare parts .

Fuji Electric is also an international Japanese company specialized in electric and electronic products . It is considered to be one of the top ten in the world in this field .

U.S.A. has a 50% share of Fuji's sales , Europe has 20% share , and a 30% share is left for Japan's internal market .

Fuji PLCs' authorized dealers in Europe are done through Germany . They are responsible for the Fuji PLC installations in third world countries including Jordan . Such sites include the ' Jordan Electricity Authority ' .

As claimed by Fuji's authorized dealer in Jordan , Fuji company refuses to allow the Jordanian Fuji authorized dealer to deal with PLCs ; Fuji company thinks it is too early for third world countries to deal with automation provided , also , that there is no qualified people in Jordan to support such technology .

In fact , Jordanian PLCs authorized dealers are facing many problems :

(a) Owners of new factories prefer to implement total solutions granted ; they import ready-made factories to be installed in Jordan by foreign companies . This is more expensive and of less benefit to the Jordanian industry ; consultation of Jordanian experts and national cooperation would result in identical solutions with less cost and would provide great benefits to the Jordanian Industrial ' Know - How ' .

(b) The new technology implemented in limited aspects in Jordan requires a highly qualified staff for support and maintenance , the lack of which is a real problem . This can be solved by supporting efficient training courses outside and inside Jordan ; national co-operation is the key solution for such a problem .

(c) The majority of Jordanian factories are based on old technology , as mentioned above , which is a real problem . The process of updating old technology to use the new technology , part of which is the PLC , is a difficult process that requires special interface stages to be accommodated with old technology logic and power requirements .

In Europe , wide markets , promising industry , and large money capitals allow for discarding the old technology and replacing it with new technology . This is not the case in Jordan .

This is a short overview of the PLC market in Jordan. As time goes , the need for co-operation between the different industrial national organizations increases in order to put the different efforts on the right track . A good step forward is accomplished if a qualified staff ; that is capable of installing , testing , and maintaining PLCs in Jordan is being developed .

Chapter Two
PLC Structure

The programmable controller as defined by National Electrical Manufacturers Association (NEMA) is a : 'Digital electronic apparatus with a programmable memory for storing instructions to implement specific functions , such as logic sequencing , timing , counting , and arithmetics to control machines and processes ' [1] .

A PLC is composed mainly of two parts ; these are the Central Processing Unit (CPU) and the input / output interface . The CPU gets its inputs from the outside world using the I/O interface , executes the stored program loaded in the memory , and sends appropriate output commands to control devices through the I/O interface . Figure 2.1 shows the block diagram of the PLC [1] .

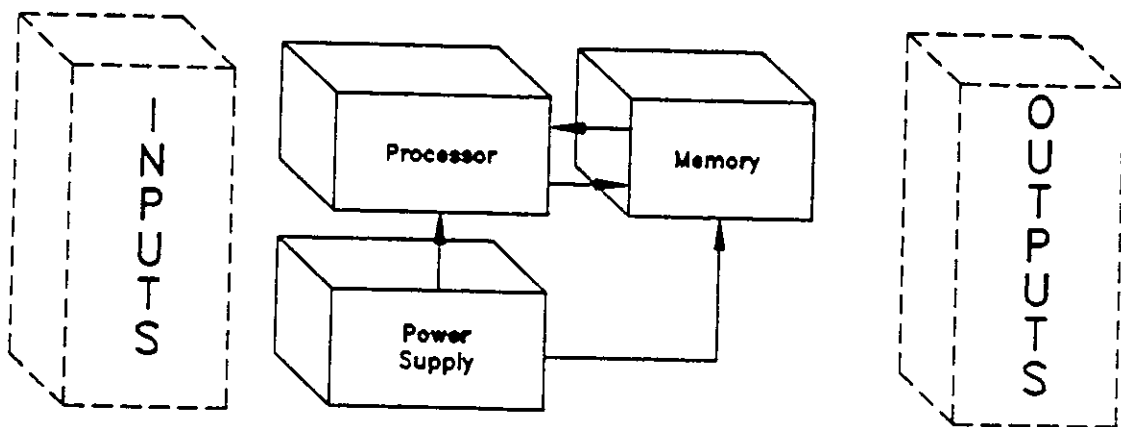


Figure 2.1 Block diagram of the PLC .

The process of reading the inputs , executing the loaded user program , and controlling outputs is called

'Scanning' and is performed in a continuous manner , as shown in figure 2.2 .

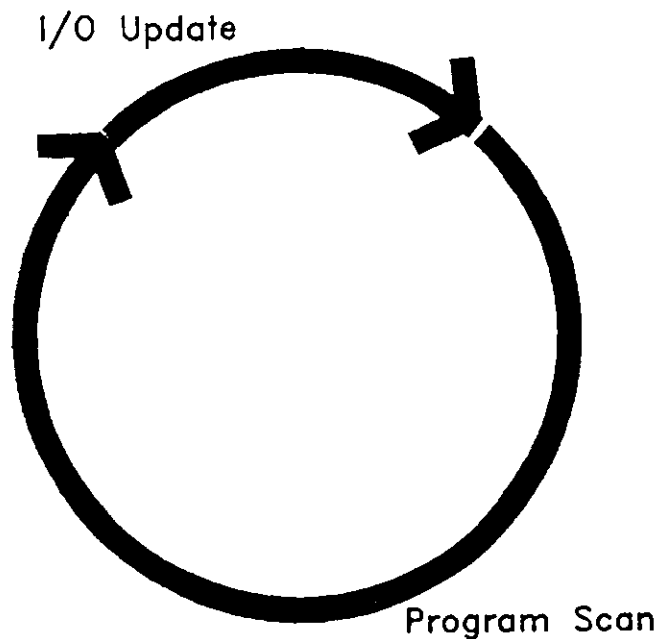


Figure 2.2 Scanning .

2.1. The Central Processing Unit (CPU) :

The central processing unit consists of three major components : the processor , the memory , and the power supply .

2.1.1. The Processor :

The processor performs many tasks including the scanning process , the different communication processes , and the system diagnostics process [1] .

Scanning is a cyclic process that is performed continuously with a cycle time depending on the power of the controller . A range of (1 - 100) msec is available , at different costs . Usually , manufacturers specify the scanning cycle time in terms of the amount of memory to be

scanned ; for example , 10 msec / 1 K byte of scanned memory. Other factors , such as the existence of remote I/O or the use of CRTs to monitor the program , affect the scanning time , also .

The processor performs the scanning process by executing a set of internal permanent programs called the Executive [1] .

Communication with remote I/O subsystems is another task of the processor [1] . Remote communication requires the existence of an I/O subsystem adapter module located in the CPU and the existence of a remote I/O processor module via which communication can be done .

Remote communication uses special cables such as twisted pair coaxial cables or fiber optic cables . Error detection and correction methods are implemented within the processor communication protocols ; possible methods are parity checking , check sum , and Hamming code methods.

The processor is capable of performing diagnostics tests for the memory , battery , power supply , and itself. Some processors are even capable of performing diagnostics tests on the controlled machine [1] .

Categorization of the processors in PLCs can be done according to the configuration of the processor system implemented . Multi-Processing category , for example , tends to have small processors working in parallel in such a way that the different tasks are distributed among them and the processing time is reduced .

The data word is another basis of classifying processors ; the data word represents the number of bits

that are treated as one unit by the processor .

2.1.2. The Memory :

The memory of a PLC can be divided , according to its usage , into four types ; these are the Executive , the Scratch Pad , the User Program Memory, and the Data Table [1] .

The Executive is that region of the memory that contains the ' Executive ' which is the supervisory permanent programs that direct the system activities .

The Scratch Pad is a temporary storage used by the CPU to store data that are continuously processed and needed instantaneously . Access of the Scratch Pad is faster than access to the main memory .

User Program Memory is that part of the main memory that contains the program of user instructions .

The Data Table is a memory area that is used to store any data required by the scanning process . Such data include any data associated with the control program , such as timers / counters preset values , and the data representing the input / output status of the system .

Figure 2.3 shows a simple memory map of the PLC . The executive and scratch pad are called the ' System Memory ' ; they are parts of the memory that are transparent to the user . Both the data table and the user program memory are called the ' Application Memory ' which is accessible to the user .

The data in the data table is treated in two different modes , status on / off mode and word (or byte) mode . In the status on / off mode , the data is treated

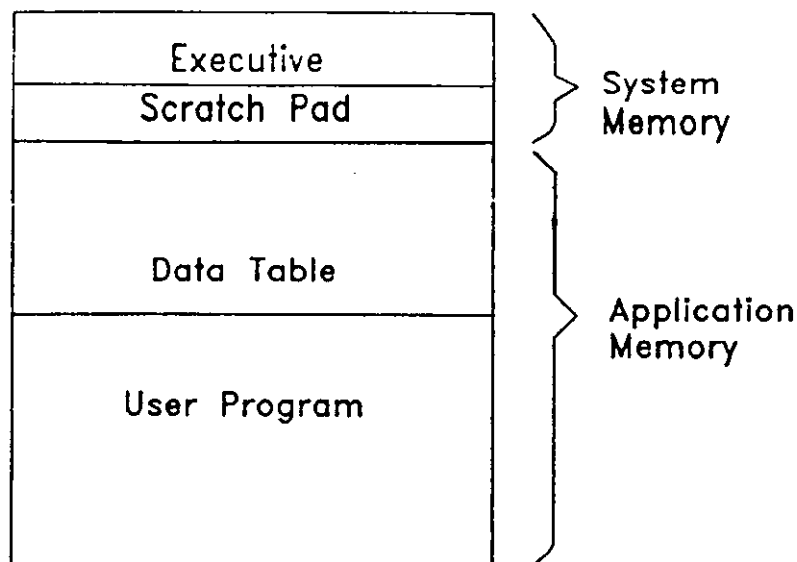


Figure 2.3 A PLC Memory Map [1] .

as single on / off bits each representing the status of a certain input , output , or memory address . In the word mode , bits of data are treated in groups representing words of data in Binary or Binary Coded Decimal (BCD) form [1] .

There are four types of data that are stored in the Data Table area according to which it can be functionally divided into [1] :

(a) Input table area : is a memory area (array of bits) used to reflect the status of input devices connected to the input interface ; each bit in the input table corresponds to an input device . When the input device is switched ' ON ' , the corresponding address bit in the input table is switched to ' 1 ' (ON / True) . The address bit is switched to ' 0 ' (OFF / False) when the input device is switched off . These input addresses are used by the ' Executive ' when performing its scanning

process .

(b) Output table area : is a memory area (array of bits) that corresponds to each output device connected to the output interface . During the scanning process , the processor reads the inputs and executes the user control program ; consequently , the output table is updated with ones and zeros .

Any output device with a memory bit set to ' 1 ' is switched ON . If the memory in the output table is ' 0 ' , the corresponding device is switched OFF .

(c) Internal storage bits : is a memory area (array of bits) used in control programs as interlocks . They are treated in a similar manner as the bits in the output table ; however , the main difference lies in that internal storage bits are not connected to output devices .

Internal storage bits are also called ' internal outputs ' or ' internal coils ' . They are usually used for internal purposes such as counters , timers , and intermediate output addresses .

(d) Storage registers : are an important part of the data table area since they allow for storage of data words (groups of bits) in BCD or binary format ; they are used to represent input and output data that cannot be represented by single ' 1 / 0 ' bits . Three types , according to their functions , of storage registers exist .

The first type is Input Registers that are used to store multi-bit data received via input interfaces from devices such as shaft encoders or thumb wheel switches . Input registers are also useful for storage of signals from

analog input devices which are converted into equivalent multi-bit binary representation .

The second type is Holding Registers that are used to store constants , entered by the user , or variables , needed during the execution of the user program . Such constants include the timer preset values , counter preset values , and ASCII characters. Timer accumulated values , counter accumulated values, BCD inputs , and BCD outputs are good examples of stored variables .

The third type is Output Registers that are used to store multi-bit data to be transmitted to analog or multi-bit output devices such as speed controllers and control valves .

Figure 2.4 below indicates the four types of memory in the data table areas .

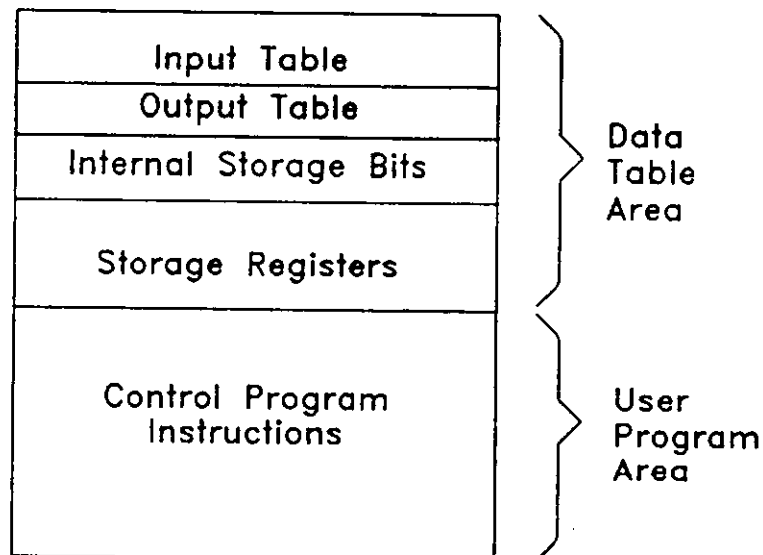


Figure 2.4 Memory Area Types of the Data Table [1] .

2.1.3. The Power Supply :

The power supply is a major part of the PLC . It supplies the DC power required by the CPU and input / output modules ; it regulates power supply input to the CPU in order to protect the different components and have a consistent operation of the PLC .

PLCs usually operate with 220 VAC power source . Some PLCs , depending on the application , accept 24 VDC power supply as a voltage source .

There are many factors that might affect the voltage supply in a factory and , thus, affect the voltage input level to the PLC . These include the start up / shut down of some heavy equipment such as large motors or compressors in the site , the power losses due to the various line distances between different substations , and the power losses due to the poor connections in the site [1] .

A PLC should be provided with a good power supply that will tolerate some range of variation in the input line voltages , 10 - 15 % for example . However, if the disturbances in the input voltage is higher than that range, the need for a constant voltage transformer, to be connected between the voltage source and the power supply , might arise . The constant voltage transformer will maintain a relatively constant output voltage although its input might be variable .

Moreover , an isolation transformer might be needed to isolate the PLC from the electromagnetic interference (EMI) in the factory that might cause misoperation of the PLC .

Another subject to be considered when thinking of power supplies is the maximum load that can be tolerated ; power supplies are capable of providing a maximum amount of current at its terminals , at a given voltage level . If the current required by the different input / output modules , at a moment of time , is greater than the maximum current of the power supply , under-current conditions will result causing an unpredictable operation of the Input / Output system . A previous good knowledge of the application requirements is vital to the selection of the proper power supply .

2.2. The Input / Output System :

The second part of the PLC is the input / output system which constitutes the physical means of communication between the CPU and the outside world . Early PLCs were limited to on / off inputs / outputs . This limitation prevented PLCs from having complete control over some processors which required analog inputs / outputs .

Nowadays , PLCs provide the user with a variety of interfaces suitable for both discrete and analog inputs / outputs . Four different kinds of inputs / outputs will be considered ; discrete inputs / outputs , numerical input / outputs , special inputs / outputs , and remote inputs / outputs [1] .

2.2.1. Discrete Inputs / Outputs :

These are the simplest types of inputs / outputs used with PLCs . Discrete input / output interfaces allow the PLC to receive on / off signals from input devices and transmit on / off signals to output devices . A list of

some discrete input / output devices is shown in table 2.1 below .

Table 2.1. Discrete Input / Output Devices .

Input Devices	Output Devices
<ul style="list-style-type: none"> - Selector switches - Push buttons - Circuit breakers - Relay contacts - Motor starter contacts 	<ul style="list-style-type: none"> - Alarms - Fans - Valves - Solenoids - Control relays

Various input / output interface circuits are available for different standard AC and DC voltage ratings. Table 2.2 indicates some of these ratings .

Different AC / DC input interface ratings are available such as 24 , 48 , 120 , and 230 volts AC / DC . The TTL input interfaces allow the PLC to accept signals from TTL-compatible devices , 5 VDC level control devices , and several types of photoelectric sensors .

Non-voltage input interface circuits allow the PLC to accept inputs from devices that are not connected to a power supply . Such interfaces contain their own internal DC (24 VDC) power supply that will compensate for the external power supply .

AC output interface circuits allow the PLC to be able to deal with output devices requiring AC input signals .

Table 2.2 Standard I/O Voltage Ratings of Discrete I/O Interfaces [2] .

Input Interfaces	Output Interfaces
- 24 Volts AC/DC	- 12-48 Volts AC
- 48 Volts AC/DC	- 120 , 230 Volts AC
- 120 Volts AC/DC	- 12-48 Volts DC
- 230 Volts AC/DC	- 120 Volts DC
- TTL level	- 23 Volts DC
- Non-voltage	- Contact relay
- Isolated inputs	- TTL level

Other discrete input / output interfaces available are outputs .

A typical interface circuit is shown in figure 2.5 . The AC / DC input interface circuit normally consists of two DC outputs , contact outputs , TTL outputs , and isolated circuits , the power circuit and the logic circuit ; they are usually isolated from each other through a coupler, which can be an optical coupler or pulse transformer , for example .

The power section is responsible for the conversion of the AC input signal to a rectified DC signal via a bridge rectifier , filtering the resulting DC signal to eliminate the dangerous effect of electric noise and signal bouncing , and detecting the minimum threshold level for a valid input signal .

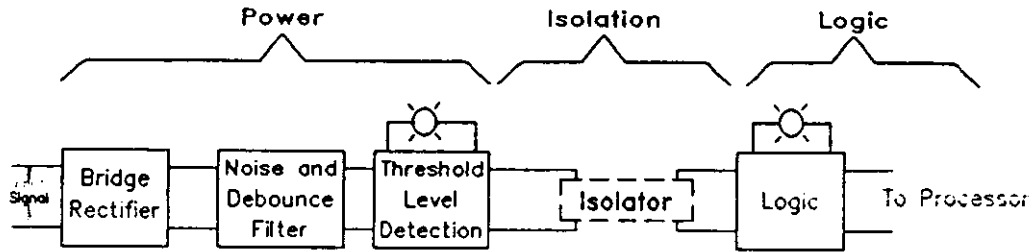


Figure 2.5 Block Diagram for AC / DC Input Circuit [1] .

The rectified , filtered signal is coupled through an optical coupler to the logic section of the PLC where it will be used in the control program . The coupler will protect the CPU side from any electrical voltage spikes that might occur .

The logic section of the interface is a digital circuit dealing with digital on / off signals .

2.2.2. Numerical-Data-Inputs-/-Outputs-:

Numerical data input / output interfaces makes it possible for the PLC to control input / output devices that deal with data bits in groups , each of which is considered to be a single unit of data . These devices can be classified into two type : analog devices and digital multi-bit devices . Analog devices transmit / receive analog signals as their outputs / inputs ; such signals include voltages and currents . Temperature transducers , pressure transducers , humidity transducers , and potentiometers are considered to be analog input devices . Analog output devices include analog valves and actuators , analog meters, and electric motor drivers .

Analog input devices need special analog interface

circuits containing analog-to-digital converters and having very high input impedance . Similarly , analog output devices receive signals from the processor through digital-to-analog converters ; analog output devices are usually optically coupled to the PLC .

Digital multi-bit devices produce discrete inputs / outputs that cannot be represented by single bits ; it rather needs a combination of bits to represent the different levels of its inputs / outputs . Bar code readers and seven-segment displays are good examples of such devices .

Data are received through the input interfaces in groups of bits and are stored in the input registers corresponding to the input devices . The program is executed and the output registers are updated with bits , groups of them , representing in BCD , the output level required . The input / output interfaces in this case provide the PLC with parallel communication capability with the input / output devices .

2.2.3. Special Inputs / Outputs :

Apart from discrete and numerical input / output devices which constitute , usually , a high percentage of the devices used in nature , some input / output devices require special input / output interface arrangements to be able to communicate with the PLC . The special input / output interfaces should adapt the signal received or transmitted in a way to suit the input / output device used. Example of signals to be adapted are very low level signals and fast input signals that cannot be detected by

the normal input / output interfaces [1] .

Special I/O arrangements may encounter the use of special I/O interface modules that contain their own processors and RAMs . This is usually called ' Distributed Processing ' since each of the processors at the interface modules accomplishes its tasks independently of the CPU .

The ASCII input / output interface module is a special circuit that requires special attention . Printers and displays , for example , are peripheral devices that send and receive ASCII code through RS-232 or RS-422 communications link . A special I/O interface should be implemented for such a case . If the ASCII interface uses the CPU as a processor, all the handshaking communication protocols are performed by the CPU ; this slows down the operation of the PLC since the CPU is interrupted by each input / output character . Moreover , if the scan rate is lower than the rate of communication , some data will be lost ; synchronization of the process , thus , might require slowing down the communication process .

Distributed processing solves such a problem ; at the start of each scan cycle , the inputs are read including the block of ASCII data accumulated in the I/O interface RAM . The handshaking protocol is performed by the I/O interface processor , and the data transmission between the CPU and the RAM of the I/O module is done at the I/O bus speed .

Similarly , a special network interface module is required to enable the PLC to interact with a network system of different devices in a factory such as robots ,

CNC machines , and CAD / CAM systems via a Local Area Network (LAN) . Communication parameters , such as baud rate , and communication protocols are network dependent [1] .

2.2.4 Remote Inputs / Outputs :

Large PLCs (512 inputs / outputs or more) provide the user with the capability of having remote input / output subsystems that are still under control of the CPU . Such a configuration requires rack-type enclosures in which the input / output modules are installed and provided with their own power supplies needed in the interface circuitry. A remote input / output adapter module is also needed to allow communications with the processor [1] .

Some limitations of remote input / output concept is the maximum number of input / output points in one subsystem and the distance allowed between the CPU and each subsystem .

Remote input / output modules offer great savings in wire , connections , and labor costs .

Chapter Three

PLC Products and Applications

A variety of PLC products and a variety of PLC applications exist . An overview of these is presented in this chapter .

3.1. PLC Families :

Nowadays , different PLC ranges are available with features depending on the size of the controller . These ranges are specified , usually , according to the number of input / output (I/O) points provided by the PLC ; this is called ' I/O count ' . As the I/O count increases , the complexity , and hence cost , of the system increases , thus requiring an increase in memory capacity , variety in I/O modules , and flexibility of the accompanying software (instruction set) [1] .

The PLC products are divided into four ranges (segments) 1 , 2 , 3 , and 4 depending on the variation of the features discussed above . The boundaries of these segments are overlapping with the intersection areas representing PLC products that have the features of the smaller PLC segment enhanced with some features found in the PLCs of the larger segment , thus providing the user with a great flexibility in choice . Figure 3.1 shows the four segments with their overlapping areas .

Segment 1 designates small PLCs which are suitable for simple on / off control . Table A.1.1 in Appendix A.1 shows some of their specifications (features) .

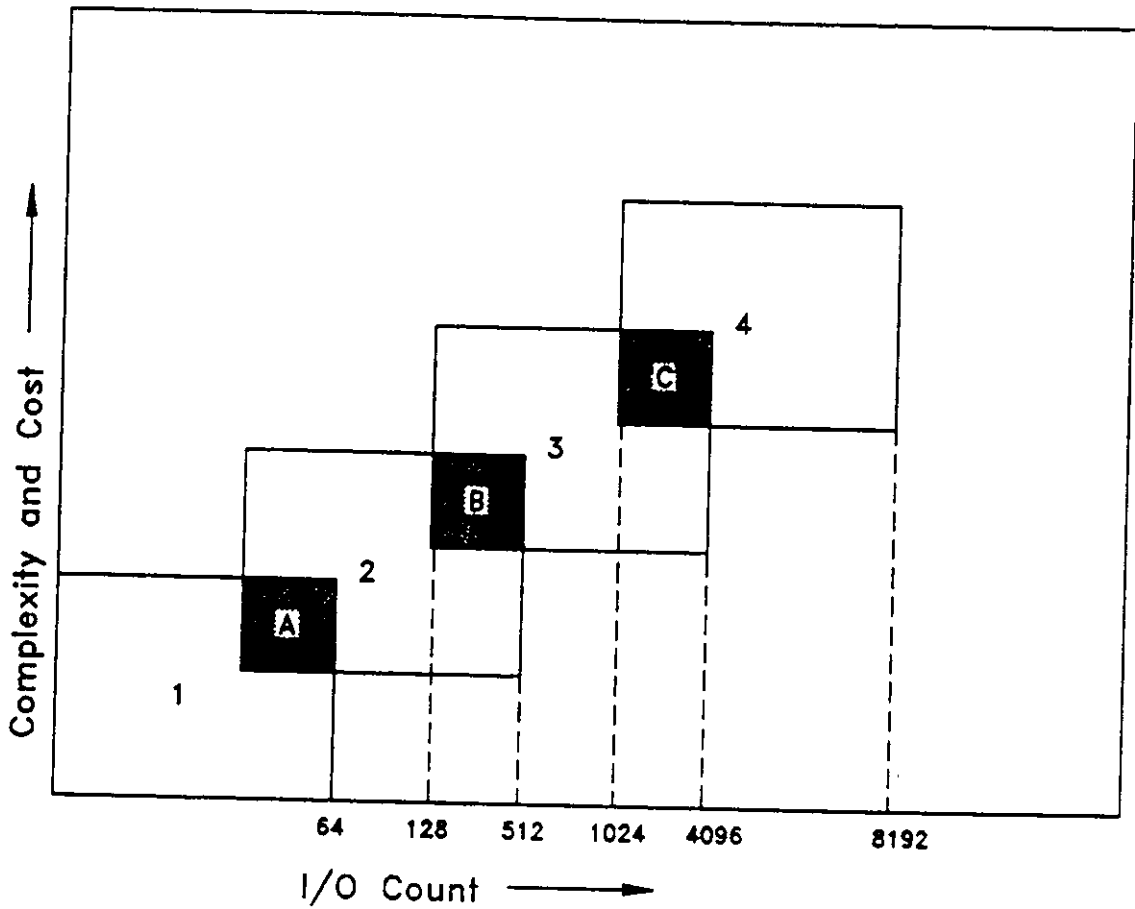


Figure 3.1 Illustration of the PLC Product Range [1].

Area ' A ' includes small PLCs that are enhanced with features from segment 2 medium controllers ; such features include analog control capability , basic mathematical operations handling , LANs compatibility , and remote I/O capability .

Segment 2 PLCs are of medium size and are more flexible than small PLCs . Table A.2.1 in Appendix A.2 indicates their features including analog control capability , data manipulation capability , and arithmetic

capabilities .

Area ' B ' Represents PLCs with features of segment 2 PLCs enhanced with more memory , PID control capability , subroutine handling capability , and more instructions for arithmetic and data handling .

Segment 3 designates large PLCs having the ability to perform extensive data manipulation , data acquisition , and reporting . Table A.3.1 in Appendix A.3 indicates their features .

Area ' C ' represents PLCs having the features of large PLCs with enhanced mathematical and data handling capabilities ; they are supported with larger memory and I/O capabilities .

Segment 4 represents very large PLCs that are used in complex control tasks where remote I/O and special I/O interfaces are needed . Table A.4.1 in Appendix A.4 indicates their features .

3.2. Defining the Control System :

Users usually face the problem of selecting the suitable PLC to their applications . The user decision should take into consideration many variables including the wide range of PLCs with different features and capabilities, the user's different applications , and thoughts of future expansion .

When a user is in a position to take a decision about the suitable controller , the following should be considered [1] :

1- The I/O modules suitable for the application should be determined . The following six key points should be

considered when determining the system I/O requirements :

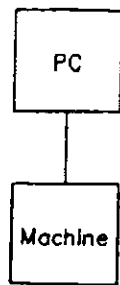
- (a) the size of the system in terms of the number of different inputs and outputs ,
- (b) the remote I/O requirements of the system , including a physical layout of the sensors and actuators , and information about the largest remote distance from the CPU ,
- (c) types of I/O modules required ,
- (d) input-to-output response time required ,
- (e) diagnostics required , and
- (f) redundancy requirements .

2- The user should , according to the knowledge of the application , determine the type of control to be implemented in his / her application . Different types of control exist , each of which has its own advantages , disadvantages , costs , and features ; these include individual machine control , centralized control , and distributed control . Figure 3.2 shows the different types of control mentioned [1] .

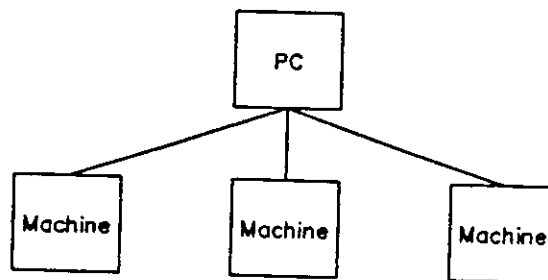
3- The user should make a good estimate of the memory size required based on the application size . The type of memory should also be determined ; for example , volatile with battery backup , non-volatile , or a combination of both .

Care should be given to limitations of memory provided by the manufacturer ; some memory may be limited in the number of internal outputs that can be used . Some restrictions might also be placed on the number of timers / counters used (50 , for example) .

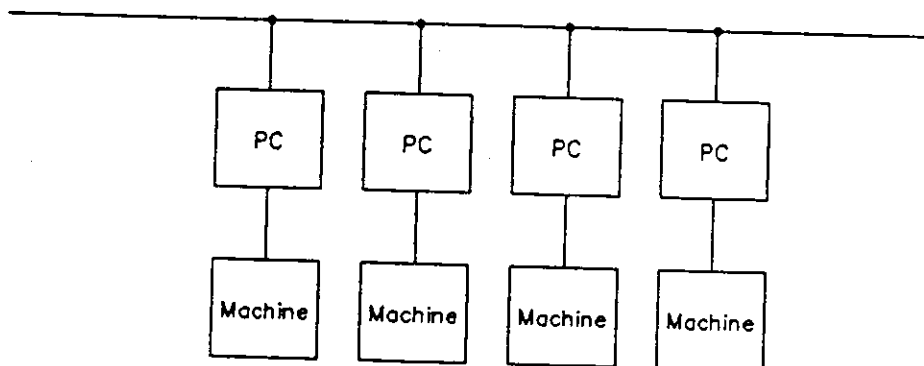
4- Software is an essential part of the decision making



a) Individual Machine Control



b) Centralized Control



c) Distributed Control

Figure 3.2 Control Type Configurations : a) Individual Machine Control , b) Centralized Control , c) Distributed Control .

process ; depending on the complexity of the application , the user should determine the instruction set to be used .

Good choice of software leads to ease of programming applications .

5- Determination of the different peripherals to be connected to the PLC is an important issue ; future expansion should be taken into consideration .

6- The physical and environmental conditions of the user site might affect the PLC . The PLC manufacturers specifications about the required operational environment should be studied.

To be more specific , effects of temperature , pressure , dust , and vibration on the PLC should be considered .

7- The existence of good vendor support is a must . Things to consider include the availability of a good support staff at the vendor's site , capability of the vendor to support training courses , and the provision of a good technical literature (manuals) by the vendor .

8- A knowledge of the product proven reliability is important ; lack of reliability increases machine downtime and affects product quality .

Table 3.1 indicates the major steps and considerations to be taken when selecting a PLC .

3.3. Steps to Implement a PLC System :

The methodical and logical approach is the key to success in the implementation of a PLC system . No matter how large the system is , a methodical approach yields reliable outcomes .

First , define your systems functional specifications . Second , draw the process and

Table 3.1 Steps for Selecting the Right PLC [1] .

- Step 1 : Know the process to be controlled .
- Step 2 : Determine the type of control .
- a- Distributed control
 - b- Centralized control
 - c- Individual machine control
- Step 3 : Determine the I/O interface requirement .
- a- Estimate digital and analog inputs and outputs
 - b- Check for Input/Output specifications
 - c- Determine if remote I/O is required
 - d- Special I/O requirements
 - e- Allow for future expansion
- Step 4 : Determine software language and functions .
- a- Ladder , Boolean , and/or high level
 - b- Basic instructions (timers , counters , etc)
 - c- Enhanced instructions / functions (math , PID)
- Step 5 : Consider the type of memory .
- a- Volatile (R/W)
 - b- Non-volatile (EEPROM , EPROM , Non-volatile RAM , CORE)
 - c- Combination of volatile and non-volatile
- Step 6 : Consider memory capacity .
- a- Estimate memory usage based on memory utilization per instruction
 - b- Allow extra memory for complex programming and future expansion .
- Step 7 : Evaluate processor scan time requirement
- Step 8 : Define programming and storage device requirements .
- a- CRT
 - b- Computer
 - c- Cassette or disk storage
 - d- Mini-programmer
 - e- Consider the functional capabilities of programming devices

continue

Table 1.1 Continued .

<p>Step 9 : Define peripheral requirements .</p> <ul style="list-style-type: none"> a- Graphic display b- Operator interface c- Line printers d- Documentation systems e- Report generation systems <p>Step 10 : Determine any physical and environmental constraints .</p> <ul style="list-style-type: none"> a- Available space of system b- Ambient conditions <p>Step 11 : Evaluate other factors that can affect selection .</p> <ul style="list-style-type: none"> a- Vendor support b- Product proven reliability c- Plant goals for future (e.g. , standardization)

instrumentation (PI) diagrams . Third , draw an I/O list . Fourth , draw the PLC configuration diagrams . Next , write the PLC program . Last , document the whole process [6] .

Definition of the systems functional specification is the initial and most important step ; other steps greatly depend on it . It should include a clear definition of the final achievements required in terms of general and specific goals . Moreover , a description of the system operation should be included .

A modular approach can be used to produce the system specification definition . First, a general description of the nature and performance of the plants equipment and of the systems total operation is made . Next , the general

description is broken into smaller functional blocks containing more details . Finally , a sequence list is obtained by providing a real-time description of the sequence of events within each functional block .

The PI diagrams include the process logic and process layout schematic . They show the different plant items , instruments , devices , and PLC I/O . Interconnections in the layout are also shown .

An I/O list is built on the information provided by the PI diagrams . It shows all the devices , their field wiring numbering , and their terminal numbers they will occupy on that PLC .

The PLC configuration diagrams support the I/O list ; they show the hardware layout and connections of the racks, CPU , power supply , and I/O cards of the PLC .

The modular approach for writing a PLC program proved to be the most professional ; testing of the written code is easy , diagnosis of the code problems is straight forward , and the size of the code is , usually , smaller than other programming methods . Thus , commissioning time is reduced .

Documentation includes all the documents that are made in the previous steps . It also includes an operator's manual . The documentation can be reproduced into a user-friendly set of manuals that will last for future use .

3.4. PLC Applications:-

Nowadays , PLCs are successfully applied in most industrial fields including steel milling , food

processing , and chemical and petrochemical treatment . Following is a quick glimpse of some of the PLC applications that would give good insight of the benefits of using PLCs and would stimulate new ideas about where and how to use PLCs [1] .

3.4.1. Rubber and Plastic :

Tire Curing Press Monitoring :

PLCs are used to perform individual press monitoring for time , pressure , and temperature during each press cycle . Machine status information is stored for later report generation and to alert the operator of any press malfunctions [1] .

Tire Manufacturing :

PLCs are used to control the sequencing of events which must occur to transform the raw tire into a tire fit for the road . Such events , in tire press / cure systems , include molding the thread pattern and curing the rubber to obtain the road-resistant characteristics . Use of PLCs reduces space requirement and increases reliability of the system and quality of the product [1] .

Plastic Injection Molding :

PLCs are used to control such variables as temperature and pressure for the optimization of the injection molding process . The PLC system provides the closed loop injection to provide good quality ; the system can also accumulate production data for future use [1] .

3.4.2. Chemical and Petrochemical :

Ammonia and Ethene Processing :

PLCs monitor and control large compressors that are used during the manufacturing of ammonia , ethylene , and other chemicals . The PLC monitors bearing temperatures , operation of clearance pockets , compressor speed , power consumption , vibration , discharge temperatures , pressure , suction flow , and gas composition [1] .

Dyes :

PLCs monitor and control the dye processing used in the textile industry . Accurate processing of color blending is provided [1] .

Gas Transmission and Distribution :

PLCs monitor and regulate pressures and flows of gas transmission and distribution systems [1] .

Fan Control :

PLCs automatically control fans based on levels of toxic gases in a chemical production environment . Control of fan start / stop , cycling , and speed is performed by the PLC so as to prevent contamination and to minimize energy consumption [1] .

3.4.3. Power-:

Plant Power System :

PLCs regulate the proper distribution of available electricity , gas , or steam. Monitoring power house facilities , scheduling distribution of energy , and generating distribution reports is also accomplished by PLCs [1] .

Energy Management :

PLCs control heating and cooling units in a manufacturing plant . Control of load cycles is also performed . Scheduled reports on the amount of energy used by the heating and cooling units are provided [1] .

Compressor Efficiency Control :

PLCs are used to control several compressors by handling safety interlocks, start up / shut down sequences , and compressor cycling . Using non-linear curves of the compressors , they are controlled to run at maximum efficiency [1] .

3.4.4. Metals :

Steel Making :

PLCs are used to control furnaces to produce metals according to preset specifications . PLCs also calculate oxygen and power requirements [1] .

Loading and Unloading of Alloys :

PLCs are used to control and monitor quantities of coal , iron ore , and limestone to be melted ; this is done through accurate weighing and loading sequences [1] .

Cold Rolling :

PLCs are used to control the conversion of semi-finished products into finished products through cold rolling machines . Speed of the motors need to be controlled to obtain correct tension and provide adequate gauging of the rolling material [1] .

3.4.5. Materials Handling :

Storage and Retrieval Systems :

PLCs are used to load and unload parts in storage and retrieval systems . PLCs keep track of necessary information such as storage lane numbers , the parts assigned to specific lanes , and the quantity of parts in any particular lane . This provides quick storage and retrieval of parts . Inventory reports can also be produced [1] .

Automatic Plating Line :

PLCs are used to control set patterns for the automated hoist which can traverse left , right , up, and down through the various plating solutions . PLCs keep track of the hoist location at all times [1] .

Conveyor systems :

PLCs control all the sequential operations , alarms , and necessary logic for circulating parts on a main line conveyor . Optimization of palletizer duty can also be obtained by scheduling lane sorting using PLCs [1] .

Automated Ware-Housing :

PLCs control the movement of stacking cranes ; high turn around of materials requests in an automated high cube vertical house is also provided . Aisle conveyors and case palletizers are also controlled by PLCs to reduce manpower requirements significantly [1] .

3.4.6. Automotive :

Internal Combustion Engine Monitoring :

PLCs monitor engines through monitoring water temperature , oil temperature , RPMs , torque , oil , pressure , and timing ; special sensors are located at the internal combustion engine to acquire data [1] .

Carburetor Production Testing :

PLCs are used to test variables such as pressure , vacuum , and flow to provide on-line analysis of automotive carburetors in a production assembly line. Significant reduction in test time , greater yield , and better quality are thus obtained [1] .

Power Steering Assembly and Test :

PLCs are used to control machines that ensure proper balance of valves and maximize left and right tuning ratios [1] .

3.4.7. Manufacturing /-Machining :

Production Machines :

Control and monitoring of automatic production machines at high efficiency rates is performed by PLCs . Monitoring also encounters machine status and piece count production [1] .

Wire Machines :

The time and on / off cycles of wire drawing machines are monitored using PLCs . Ramping control and synchronization of electric motor drives are also provided by the PLC system [1] .

Tool Changing :

PLCs are used to control synchronous metal cutting machines with several tool groups . Based on the number of parts that a tool manufactures, the PLC system keeps track of when to change the tool [1] .

3.4.8. Examples :

(a) Conveying Systems :

Conveyor Dynamics , Inc. (CDI) in U.S.A. has developed a modular approach to PLC programming over several years of accumulated experience in control of crushing and conveying systems . CDI has many successful sites at Canada , Utah , Mexico , Arizona, and Chile in which PLCs were used to efficiently control conveying and crushing systems [7] .

CDI has its own control philosophy that aims at five major goals . First is to maximize system safety. Second is to minimize commissioning down-time. Third is to minimize production down-time and maximize availability . Next is simplifying the operator interface and control . Last is providing fast diagnostics .

Large conveyors control systems should be precisely designed to insure safe operation ; dynamic instabilities of the belt , belt drift in curve conveyors , and fast natural acceleration of decline conveyors under loaded conditions are points of issue in such systems .

PLC control systems has proved to be efficient

in this field : they provide precise control of conveyor startup and stoppage sequences , they are flexible to field customization and tuning , and they are capable of monitoring the different components of the system . Thus , a high level of protection is guaranteed .

Smooth transitions from old existing relay control systems to new PLC control systems can be obtained by careful planning , full simulation of the control programs , and efficient debugging of problems , all of which are provided by the new PLC systems . Thus commissioning down-time is reduced .

The development of a reliable , complete diagnostics systems leads to minimizing the production down-time and , hence , maximizing system availability . PLC systems are completely flexible for such developments .

PLC systems can be interfaced to numerous control panels , a combination of which provides a user-friendly , efficient operator interface . Such an operator interface would minimize training time and operator errors .

Proper diagnostics of problems , a feature of PLC systems , results in fast fault finding ; down-time is minimized . It also helps organizing preventive maintenance .

(b) Monitoring of Analyzer Systems :

Syncrode Canada Ltd. is using PLCs in monitoring the operation of different analyzer systems .

Abnormal operation conditions are detected and corrective actions are taken to provide proper operation [8] .

PLCs are used with flue gas analyzer applications in which the status of the instrument air supply status has to be recorded . Syncrode Canada Ltd. is using PLCs in boiler control systems to monitor inlet flow rate of both sample and combustion air . Moreover , they are using PLCs with analyzer systems to monitor steam quantity in a hydrogen reformer plant .

Syncrode Canada Ltd. is using PLCs due to their flexibility and adaptability . However , it faces some problems since operators are unfamiliar with PLCs and ladder diagram . Many more applications of PLCs in industry really exist , and much more applications appear , every now and then , due to the great facilities that PLCs provide .

Chapter Four Hardware Design

Based on the information in the previous chapter , the steps towards constructing a prototype PLC are separated into two interrelated procedures to be carried out in parallel . These are the hardware design and software development and programming .

Hardware design is concerned with the construction of a hardware environment similar to that of the PLC . The software procedure is concerned with the development of a suitable software environment related to the hardware part and similar to that of the PLC . This chapter is devoted to the description of hardware design . Chapter five is devoted to software development and programming .

As was described in chapter two , the PLC structure is composed of two major parts : The Central Processing Unit (CPU) and the Input / Output (I/O) interface . Hardware wise , the construction of a prototype PLC calls for the implementation of these two parts .

4.1. Central Processing Unit (CPU) :

A personal computer is used to provide the CPU part of a PLC . As it has been previously mentioned , the CPU is composed of three parts ; the processor , the power supply, and the memory ; which are all provided by the personal computer .

A personal computer with an 8086 or an 80286 microprocessor would be sufficient to operate the prototype PLC since our expected applications in the C.I.M. laboratory would never require a more powerful processor .

However , an 80386 or 80486 microprocessor based personal computer can be used , when needed , to get a higher performance .

4.2. Input/Output (I/O) Interface :

Since the input / output interface part of the PLC is an application dependent part , a prototype discrete I/O module is designed to provide twelve on / off output signals at 115 VAC and twelve on / off signals at TTL level . The I/O module consists of two major parts : the I/O interface card and the power amplification card .

Figure 4.1 is a block diagram of the I/O module implemented .

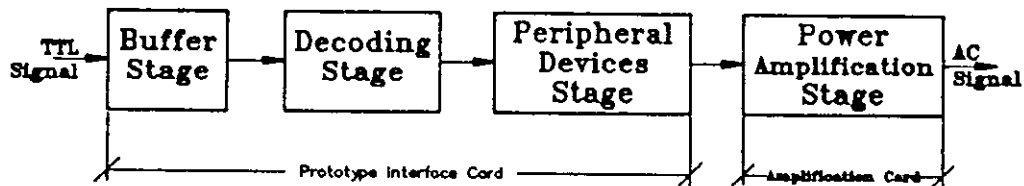


Figure 4.1 Block Diagram of the I/O Module .

The interface card is designed to be inserted in any of the personal computer available card slots and is considered to be a part of the personal computer . This prototype card is designed to be used in all personal computer models which use the 62-pin interface bus . This includes the PC-XT and PC-AT . However , this card does not support 16-bit data transfers in the PC-AT ; only 8-bit data transfers are supported .

A DC / AC converter (power amplification) stage is

designed to receive the output signals of the interface card and , accordingly , provide AC signals capable of driving AC loads . This stage represents , in addition to the I/O interface card , an I/O module suitable for 115 VAC applications .

4.2.1. Input /-Output Interface Card :

The card is composed of three hardware stages :

- (a) Buffer stage : the buffer stage is designed to protect the personal computer from high current leakages and voltage drops . All the address , data , and control lines are buffered using line drivers and bus transceivers , thus, peripheral devices can be installed in the card securely eliminating undesirable under-current conditions .
- (b) Decoding stage : the decoding stage is designed to decode the buffered address bus signals of the personal computer to determine the peripheral device to be accessed. The buffered control lines such as +AEN , -IOW , and -IOR are also used in the decoding process . Decoding is performed using a comparator , a decoder , AND gates , and OR gates .
- (c) Peripheral devices stage : this stage is designed to have two peripheral devices ; these are the 8255A Programmable Peripheral Interface (PPI) device and the 8253 Timer / Counter Circuit device . This stage allows for future expansion of six more peripheral devices .

Table 4.1 shows a list of the components used to build-up the interface card .

Figure 4.2 is a schematic diagram of the I/O interface card.

Table 4.1. Components of The Interface Card .

Component	Quantity
74LS00 NAND Gates	2
74LS08 AND Gates	1
74LS85 Comparator	1
74LS138 Decoder	1
74LS244 Line Driver	2
74LS245 Bus Transciever	1
8255A PPI	1
8253 Timer / Counter	1
1 MHz Crystal	1
4 Position Dip Switch	1
10 Micro Farad Tantalum Capacitor	2
0.1 Micro Farad Capacitor	5
4.7 kohm Resistance	5
1 Kohm Resistance	2
330 ohm resistance	1

4.2.1.1. Buffer Stage :

IC-1 (74LS245) is a bidirectional 8-bit bus transceiver . It operates in a three-state mode and controls the transfer of data between the 62 pin bus and the unique circuitry on the card . The 'Dir' (pin 1) on the chip determines the direction of data flow. If it is ' Low ' , data will flow from the card circuitry to the interface bus ; if it is ' High ' ,

data will flow from the interface bus to the card circuitry . The ' Output Enable ' (pin 19) controls data flow : when pin 19 is 'Active Low' , data will be permitted to flow ; else , the 74LS245 switches to its ' third ' state (high impedance) . The 74LS245 data sheets are shown in Appendix B.1 [9] .

ICs two and three (74LS244) are 8-bit wide line drivers whose ' Enable ' pins ($\overline{1G}$ & $\overline{2G}$) are connected to ground . They always pass the bus signals they buffer straight through since their enable signals are ' Active Low ' . Refer to Appendix B.2 for the 74LS244 data sheets [9] .

4.2.1.2. Decoding Stage :

IC-4 (74LS85) is a four-bit comparator that compares the signals' levels arriving from their logical sources , the address bus (A8 - A5) and the DIP switch , and outputs a 'High' signal (pin6) if they are equal . This chip is enabled by the address line (A9) . The 74LS85 data sheets are shown in Appendix B.3 [9] .

The sixteen different possible positions of the DIP switch provides a wide range of addresses that can be used to access the card . For example , if the DIP switch is positioned at (1000) , then the range of addresses available is Hex 300 - 31F . Similarly , if switch is positioned at (0000) then the range of addresses available is Hex 200 - 21F . IC-5 (74LS138) is a decoder that further decodes the address line (A4 - A2) . Each of the selected outputs can be used

to enable four consecutive addresses , using A1 and A0 . For example , when the DIP switch is positioned at (1000) the ' Select 1 ' pin will be Active Low for the address range Hex 304 - 307 .

The 74LS138 decoder is enabled when pin 4 ($\overline{G2A}$) is ' Active Low ' , this will occur if the following conditions are met :

- * Either -IOR or -IOW is active low ,
 - * +AEN is low ,
 - * and the comparator output is ' High ' (enabled) , thus , indicating the appropriate address selection .
- Data sheets for the 74LS138 decoder are included in Appendix B.4 [9] .

The same conditions are used to control data flow in IC-1 . These conditions are controlled using a combination of NAND / AND gates from ICs 8 and 9 (74LS00 and 74LS08 , respectively) .

4.2.1.3. Peripheral Devices Stage :

Two of the selected outputs , ' Select 0 ' and ' Select 1 ' , are used to chip select IC-7 and IC-6 , respectively .

IC-6 is an 8255A Programmable Peripheral Interface (PPI) chip that provides parallel data transfer (I/O) . It has 24 (3 ports A , B , and C ; 8 bits each) I/O lines which may be individually programmed in two groups of 12 lines each and used in three different modes of operation [10] .

Mode 0 , called ' Basic Input / Output ' mode , is implemented in this card in which each group of 12 I/O pins may be programmed in sets of 4 as either

input or output .

Mode 1 , called ' Strobed Input / Output ' mode, provides two 8-bit data ports that can be configured to operate in either input or output mode in conjunction with strobes or ' hand shaking ' protocols . Port C is divided into two parts , 4-bits each , that work as control and status signals ports for the PPI .

Mode 2 , called 'Strobed Bidirectional Bus I/O', provides the user with one 8-bit bidirectional port accompanied with five control / status signals that specify the instantaneous direction of data flow .

Mode 0 provides simple input and output operations that require no hand shaking procedures . It can be functionally defined as :

- * Having two 8-bit ports , A and B , and two 4-bit ports , C upper (PC4 - PC7) and C lower (PC0 - PC3) .
- * Any port can be input or output ; thus , sixteen different I/O configurations are possible .
- * Outputs are latched .
- * Inputs are not latched .

This card uses configuration 12 in which both ports A (PA7 - PA0) and C upper (PC7 - PC4) are used as inputs , and ports B (PB7 - PB0) and C lower (PC3 - PC0) are used as outputs .

Figure 4.3 shows a block diagram of the PPI ; it consists of the following blocks :

- 1- Data Bus Buffer (D7 - D0) :

This is a three-state bidirectional 8-bit buffer

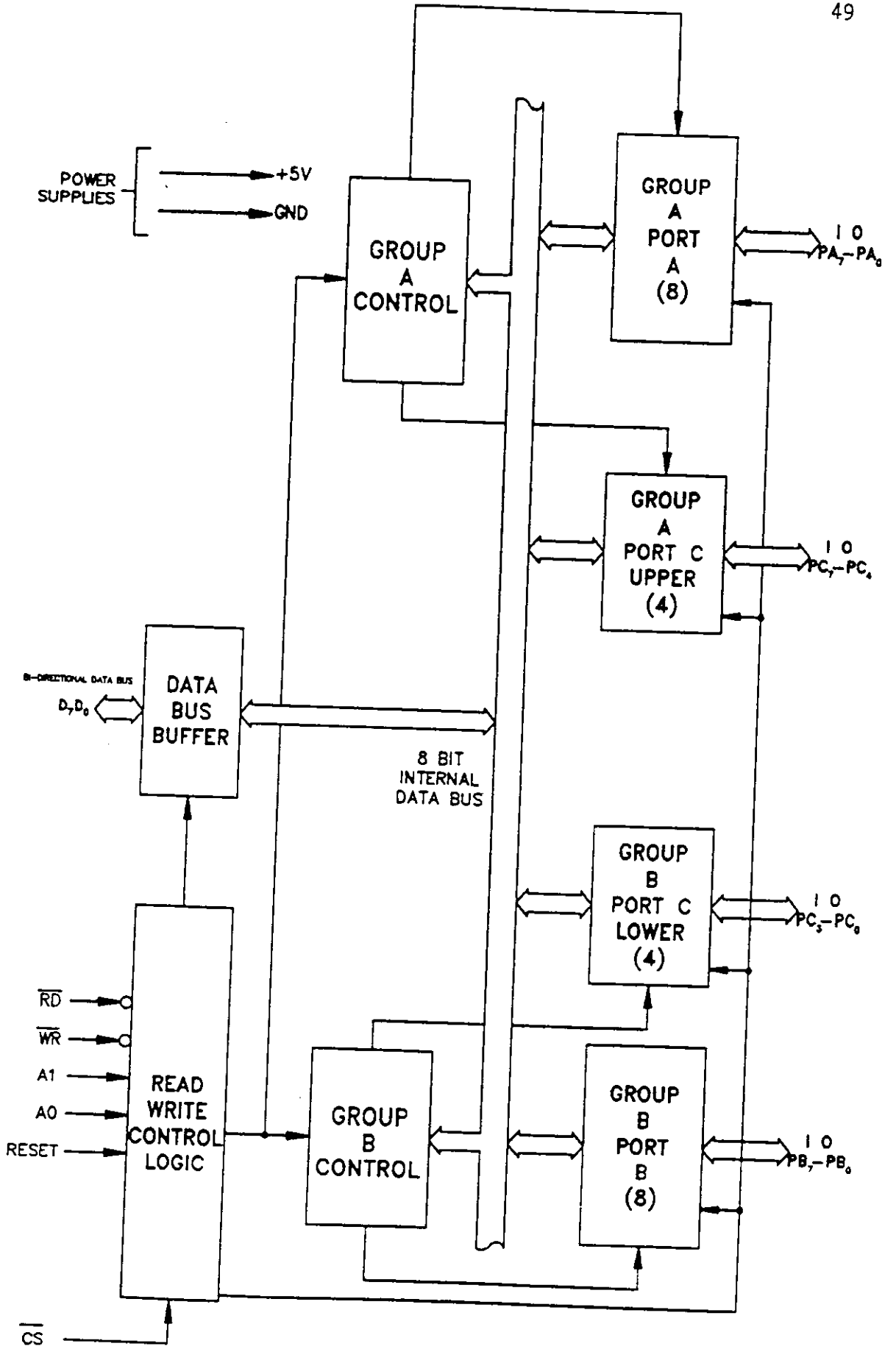


Figure 4.3 Block Diagram of the PPI [10] .

that is used to interface the 8255A to the system data bus . Direction and time of data flow is controlled by the MPU instructions . This bus is also used to transfer control and status information .

2- Read , Write , and Control Logic :

Upon receiving address and control signals from the Micro Processor Unit (MPU) , the control logic block issues commands to the control blocks of both groups , A and B , to manage the internal and external transfer of data and control words .

This block includes the Chip Select (\overline{CS}) pin which , when ' Active Low ' enables the 8255A . It also includes the Read (\overline{RD}) pin which allows the CPU to read from the 8255A and the Write (\overline{WR}) pin which allows the CPU to write to the 8255A . The Port Select 0 and Port Select 1 (A0 and A1) pins which , in coordination with the RD and WR pins control the selection of one of the three ports or the control word registers , are also included in this block . Moreover , this block includes the Reset pin which , when ' Active High ' , clears the control registers and sets all ports (A , B , and C) to operate at the input mode .

Table 4.2. shows the different possible combinations of the control and address signals and , hence , the resulting operation .

3- Group A and Group B Controls :

These control blocks are loaded by the CPU with control instructions that specify the functional

Table 4.2 8255A Basic Operation [10] .

A1	A0	\overline{RD}	\overline{WR}	\overline{CS}	Operation
					Input (READ)
0	0	0	1	0	Port A -----> Data Bus
0	1	0	1	0	Port B -----> Data Bus
1	0	0	1	0	Port C -----> Data Bus
					Output (WRITE)
0	0	1	0	0	Data Bus -----> Port A
0	1	1	0	0	Data Bus -----> Port B
1	0	1	0	0	Data Bus -----> Port C
1	1	1	0	0	Data Bus -----> Control
					Disable Function
X	X	X	X	1	Data Bus -----> 3-State
1	1	0	1	0	Illegal Condition
X	X	1	1	0	Data Bus -----> 3-State

configuration of each port . Upon receiving signals from the Read / Write Control Logic , each control block issues proper commands to its associated ports. Control Group A controls port A and C upper (PC7 - PC4) ; Control Group B controls port B and port C lower (PC3 - PC0) .

The control information is determined by a control word transmitted by the CPU . Figure 4.4 shows the mode definition format of the control word.

Our control word for mode 0 configuration is (10011000) .

4- Ports A , B , and C :

The PPI has three ports with the following specifications :

Port A : One 8-bit data output latch / buffer and one 8-bit data input latch .

PORT B : One 8-bit data input / output latch / buffer and one 8-bit data input buffer .

PORT C : One 8-bit data output latch / buffer and one 8-bit data input buffer (no latch for input) .

This port can be divided into two 4-bit ports under the mode control .

Table 4.3 shows the PPI port definition format at mode 0 operation .

IC-7 (8253) is a programmable interval timer / counter that includes three identical 16-bit counters which can operate independently in any of six different modes . To operate a counter , an (8 / 16) -bit count is loaded in its register and , on command, it begins to decrement the count until it reaches zero. At the end of the count , it generates a pulse that can be detected by the CPU at one of its available interrupt requests (IRQ6) or by any available input port (PC#5 of PPI is used in our case) . In addition , the count can be read by the cpu while the counter is decrementing [10] .

Figure 4.5 is a block diagram of the 8253 timer / counter device ; it consists of the following

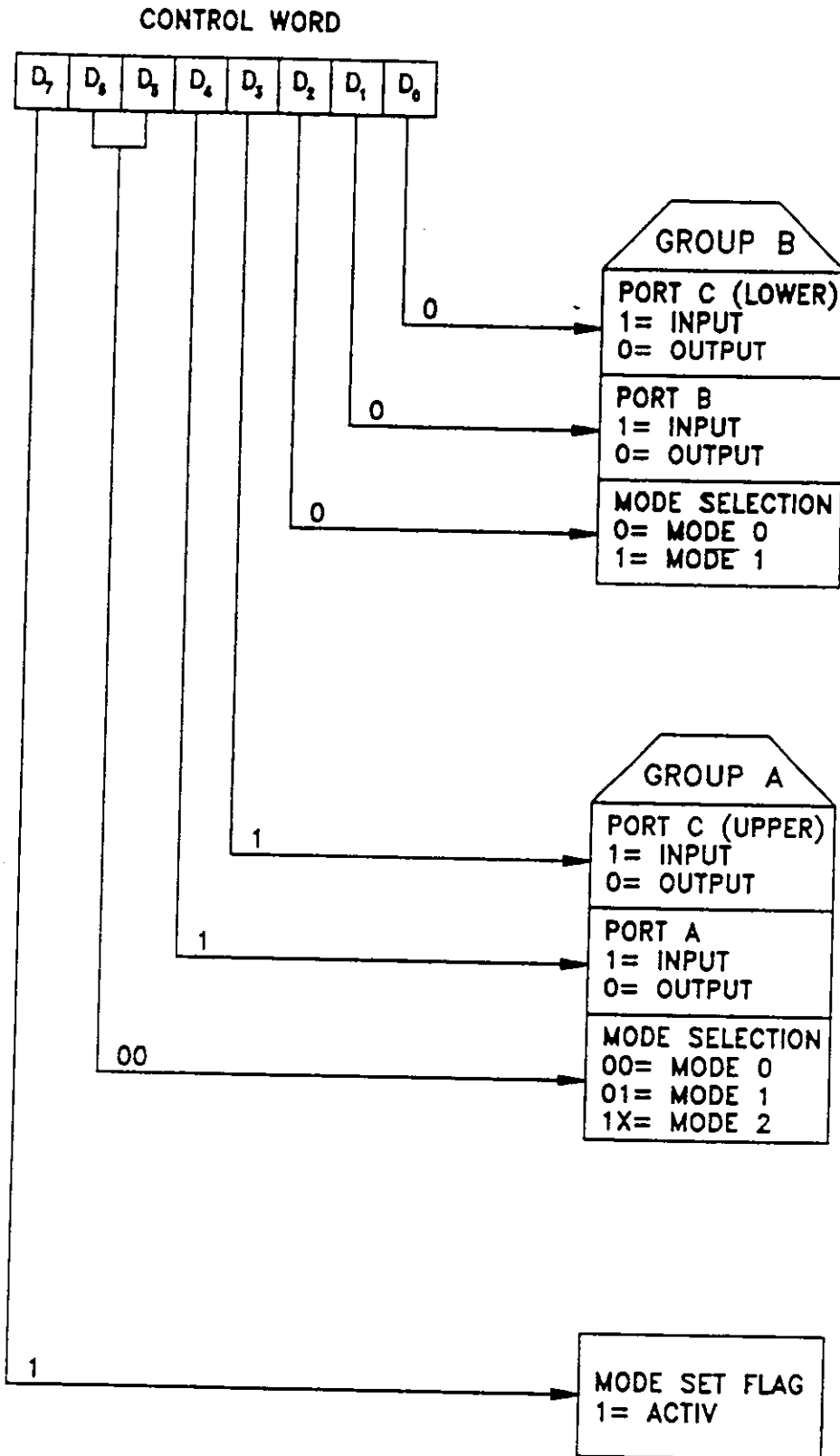


Figure 4.4 Mode Definition Format of the PPI [10] .

Table 4.3 Mode 0 Port Definition of The PPI [10] .
 (D2 = 0 , D5 = 0 , D6 = 0) .

A		B		Group A			Group B	
D4	D3	D1	D0	Port A	Port C (Upper)	#	Port B	Port C (Lower)
0	0	0	0	output	output	0	output	output
0	0	0	1	output	output	1	output	input
0	0	1	0	output	output	2	input	output
0	0	1	1	output	output	3	input	input
0	1	0	0	output	input	4	output	output
0	1	0	1	output	input	5	output	input
0	1	1	0	output	input	6	input	output
0	1	1	1	output	input	7	input	input
1	0	0	0	input	output	8	output	output
1	0	0	1	input	output	9	output	input
1	0	1	0	input	output	10	input	output
1	0	1	1	input	output	11	input	input
1	1	0	0	input	input	12	output	output
1	1	0	1	input	input	13	output	input
1	1	1	0	input	input	14	input	output
1	1	1	1	input	input	15	input	input

blocks:

1- Data Bus Buffer :

This is a tri-state , 8-bit , bidirectional buffer that is connected to buffered data bus of the

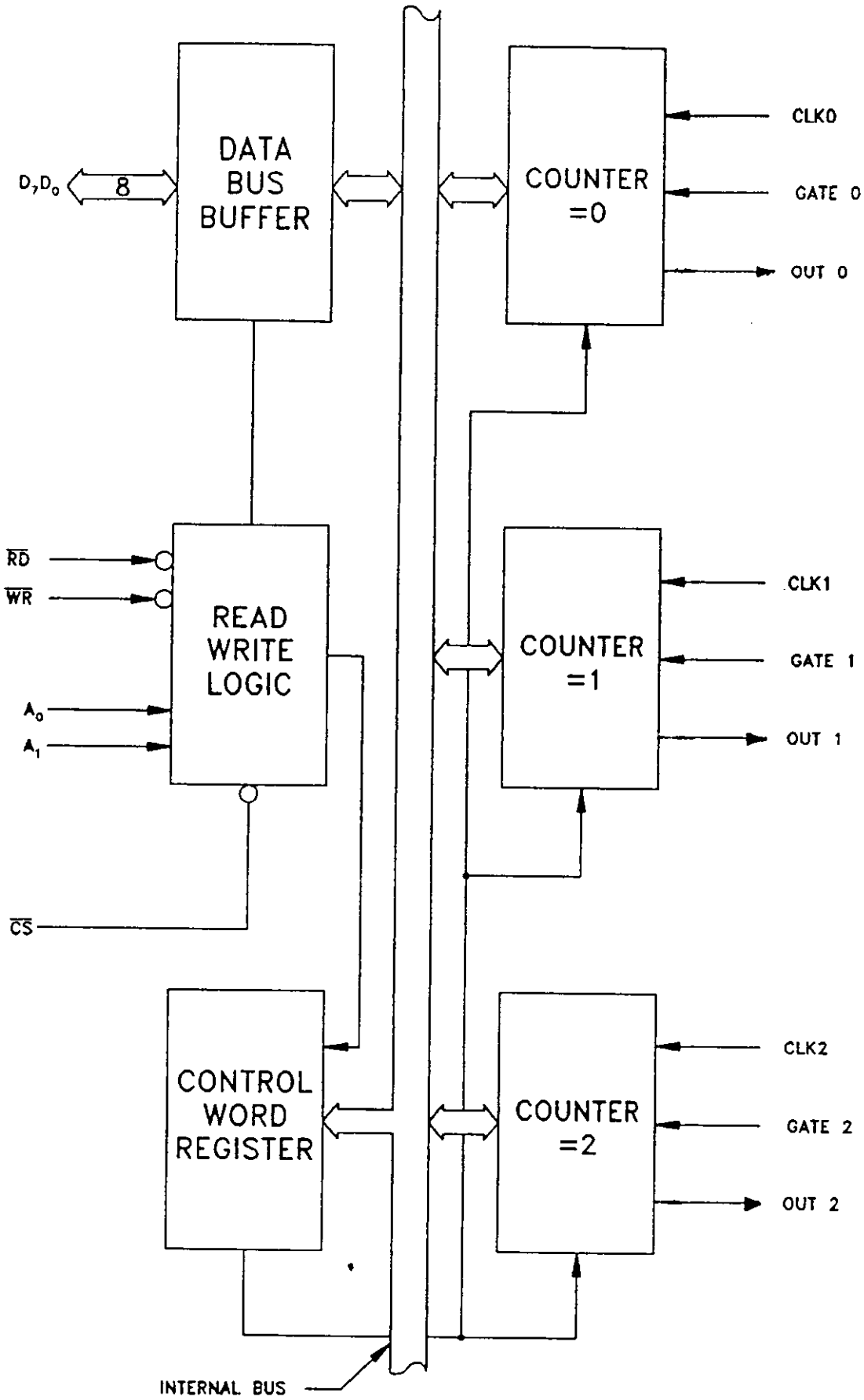


Figure 4.5 Block Diagram of the 8253 Timer / Counter [10] .

CPU .

2- Read / Write Control Logic :

This block includes five important signals that control the operation of the timer : first , the READ (\overline{RD}) signal that is connected to buffered -IOR signal of the CPU and , when ' Active Low ' , allows the CPU to read from the 8253 ; second , the WRITE (\overline{WR}) signal that is connected to the buffered -IOW signal of the CPU , and , when ' Active Low ' , allows the CPU to write to the 8253 ; third , the Chip Select (\overline{CS}) which , when ' Active Low ' , enables the CPU access to the 8253 ; and last the A1 and A0 signals that are connected to A1 and A0 of the MPU to specify the required counter control register to be accessed . Table 4.4 lists the logic selection table of the different timer registers .

3- Control Word Register :

The contents of this register are determined by the MPU , and they specify the counter to be used , its mode and either a read or write operation .

4- Counters 1 , 2 , and 3 :

Each of these has two input signals , clock (CLK) and gate (G) , and one output signal (OUT). Each counter can count either in binary or BCD .

Figure 4.6.a clarifies the control word format . Figure 4.6.b is the ' select counter ' logic table . Figure 4.6.c is the read / load logic table . Figure 4.6.d is the mode of operation format . Figure 4.6.e is the binary / BCD selection format .

Table 4.4 8253 Registers' Logic Selection Table .

$\overline{\text{CS}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	A1	A0	Register Access
0	1	0	0	0	Load Counter No. 0
0	1	0	0	1	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	0	1	1	Write Mode Word
0	0	1	0	0	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	0	1	1	0	Read Counter No. 2
0	0	1	1	1	No-Operation 3-State
1	x	x	x	x	Disable 3-State
0	1	1	x	x	No-Operation 3-State

The 8253 timer / counter is set to operate in mode 0 (interrupt on terminal count) ; this is done by loading a specific control word into the control register of the timer (01010000) . An external clock signal is provided for the timer using a special crystal circuitry in which NAND gates (IC-10) are used . The ' GATE ' input (pin 11) is set 'High' to enable counting . The output (pin 10) is connected to PC#5 (pin 12) of the the PPI . The timing diagram for mode 0 operation is shown in figure 4.7 .

The time period needed for any count is determined by the clock rate of the crystal which is

(a)

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

(b)

SC1	SC0	Select Counter
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

(c)

RL1	RL0	Read / Load
0	0	Counter Latching Operation
1	0	Read / Load Most Significant Byte Only
0	1	Read / Load Least Significant Byte Only
1	1	Read / Load Least Significant Byte First , Then Most Significant Byte

Figure 4.6. (a) Control Word Format , (b) Select Counter , (c) RL - Read/Load , (d) M - Mode , (e) Binary / BCD [10] .

(d)

M2	M1	M0	Mode
0	0	0	Mode 0
0	0	1	Mode 1
x	1	0	Mode 2
x	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

(e)

Binary / BCD	Count Type
0	Binary Counter 16 Bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Figure 4.6 Continued .

independent of the MPU ; thus , this interface card will provide fixed timing operations independent of the MPU processing unit .

4.2.2. Power-Amplification-Card :

The power amplification card converts the TTL level signal produced by the interface card into an AC signal capable of driving AC loads . This is considered to be a DC

MODE 0

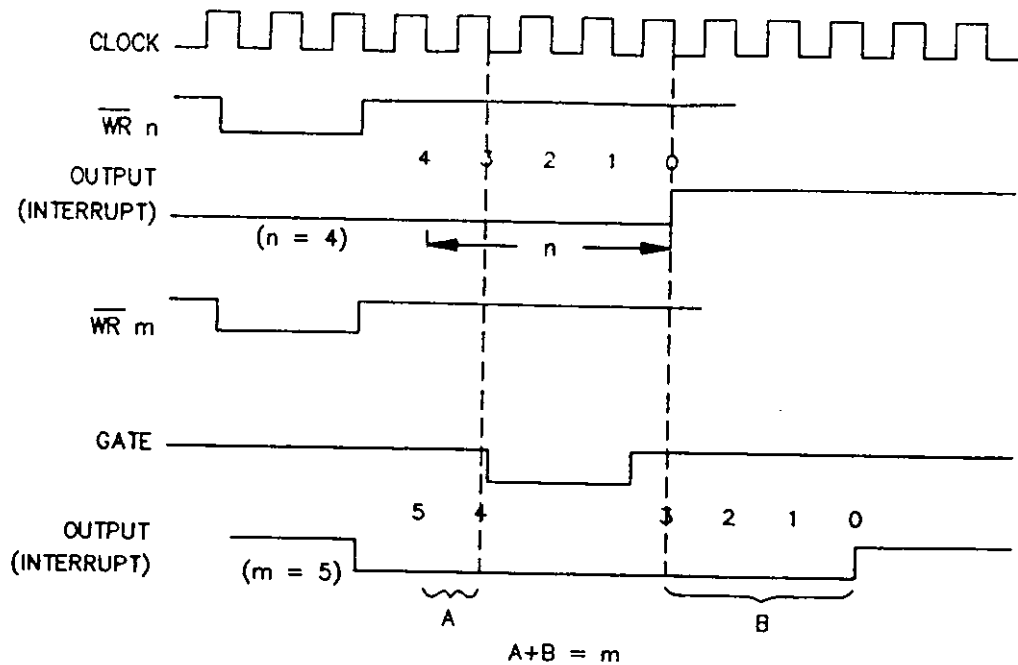


Figure 4.7 Timing Diagram for Mode 0 Operation of the 8253 Timer / Counter Device [10] .

to AC power converter . This stage is implemented using a single integrated circuit which is especially designed for TTL to 115 VAC conversion . The chip MOC3031 , which is manufactured by Motorola , can be analyzed into three major components (ports) : 1- triac, 2- zero crossing circuit , and 3- optical isolation . Figure 4.8 shows the internal components of the MOC3031 .

Thyristors are bistable semiconductor devices . Triacs are bidirectional thyristors capable of conduction in either direction as suggested by the opposing arrow heads shown in the device symbol [11] .

A triac has three terminals : 1- Main Terminal 1 (MT 1) , 2- Main Terminal 2 (MT 2) , and 3- Gate (G) . The polarities of MT 1 and MT 2 are insignificant ; however MT 2 and MT 1 are sometimes called the anode and the cathode ,

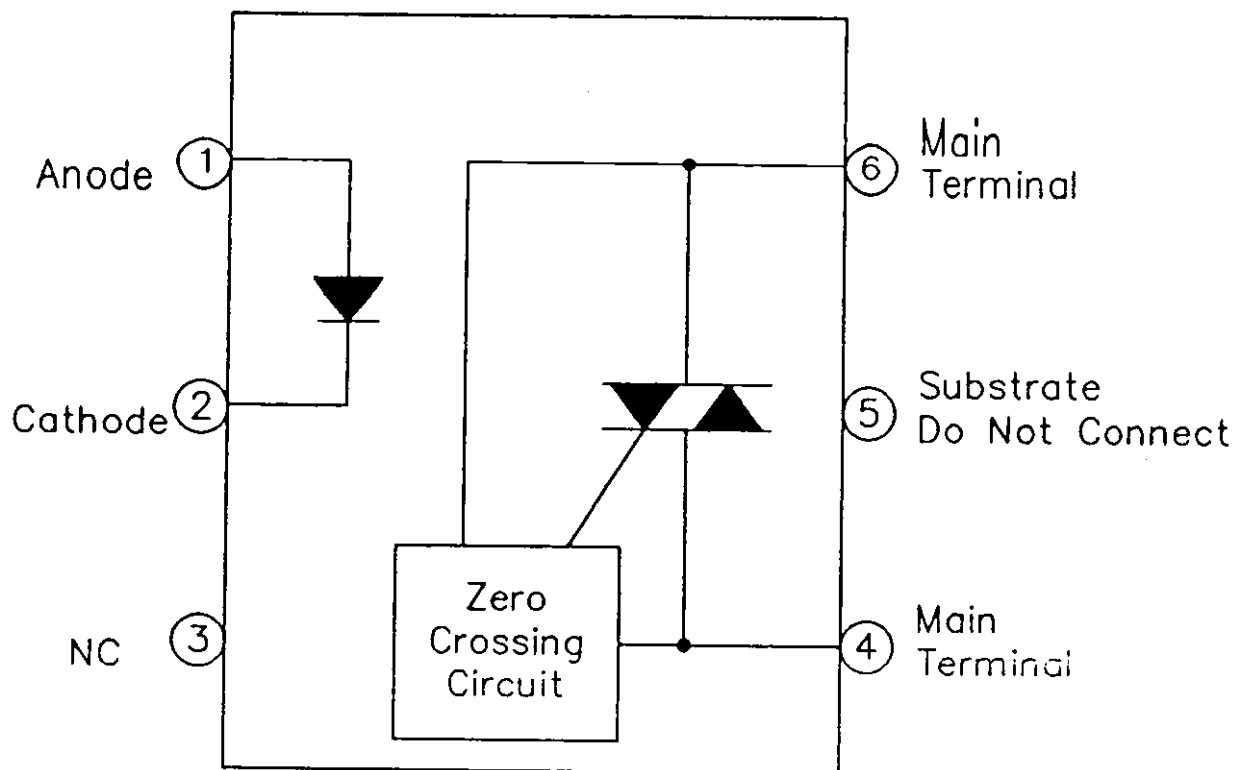


Figure 4.8 MOC3031 Opto-Coupled Triac [11] .

respectively , due to the naming convention followed in other thyristors .

The triac is usually thought of as two inverse parallel connected Silicon Controlled Rectifiers (SCR) . This can be simply represented by a symmetrical dashed line passing vertically through the triac structure as shown in figure 4.9 .

Figure 4.10 shows the voltage-current characteristics of the triac . As can be seen , if the voltage potential between the anode and the cathode exceeds the breakdown voltage (V_{BO}) then the device will switch from the high impedance (OFF) state through a negative resistance region to low impedance (ON) state . In the ON state , the voltage drop across the triac is very small , about 1 - 1.5 Volts , and a high main terminal current is allowed to flow through the device . For the triac device to keep in

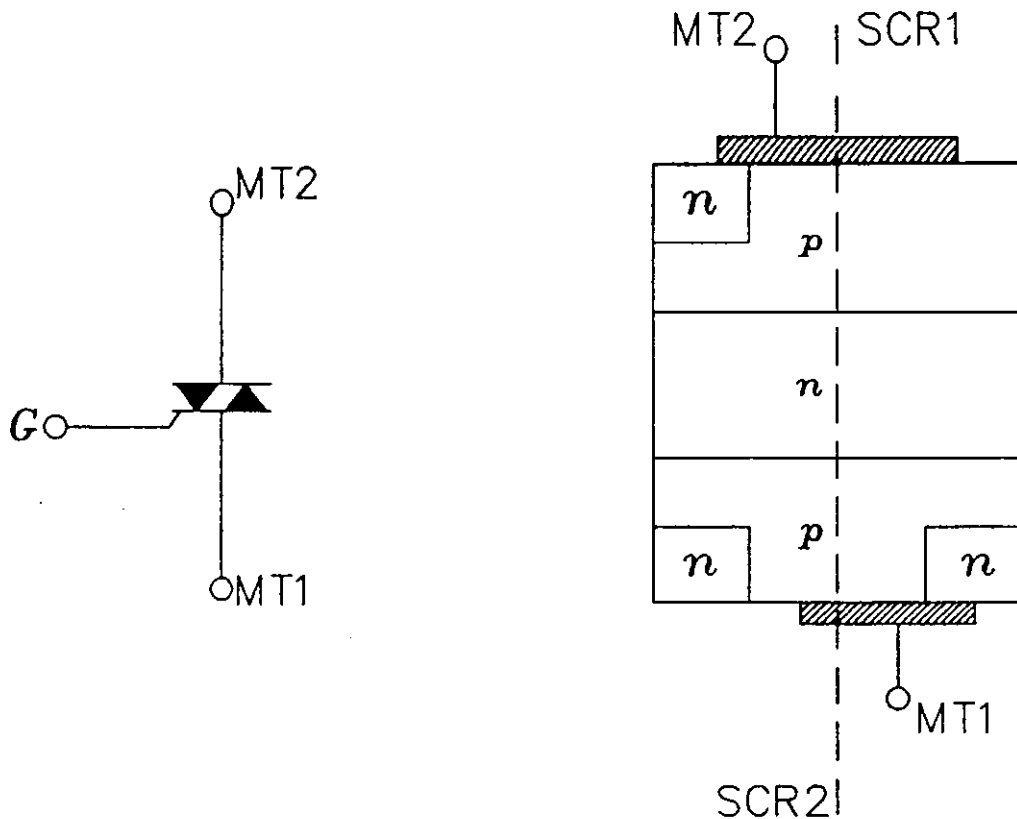


Figure 4.9 (a) Triac symbol , (b) Triac Structure [11].

the ' ON ' state , the main terminal current must keep a minimum value called ' Latching Current I_L ' .

Appendix C.1 contains some term definitions and performance parameters of the triacs .

The triac can operate in four modes , $I+$, $III+$, $I-$, and $III-$ [11] :

- 1- MT1 (cathode) is negative with respect to MT2 (anode) and the gate is positive with respect to MT1 . This mode is called ($I+$) mode .
- 2- MT1 is negative with respect to MT2 and gate is negative with respect to MT1 . This mode is also called ($I-$) mode.
- 3- MT1 is positive with respect to MT2 and gate is positive with respect to MT1 . This is also called ($III+$) mode .
- 4- MT1 is positive with respect to MT2 and gate is negative

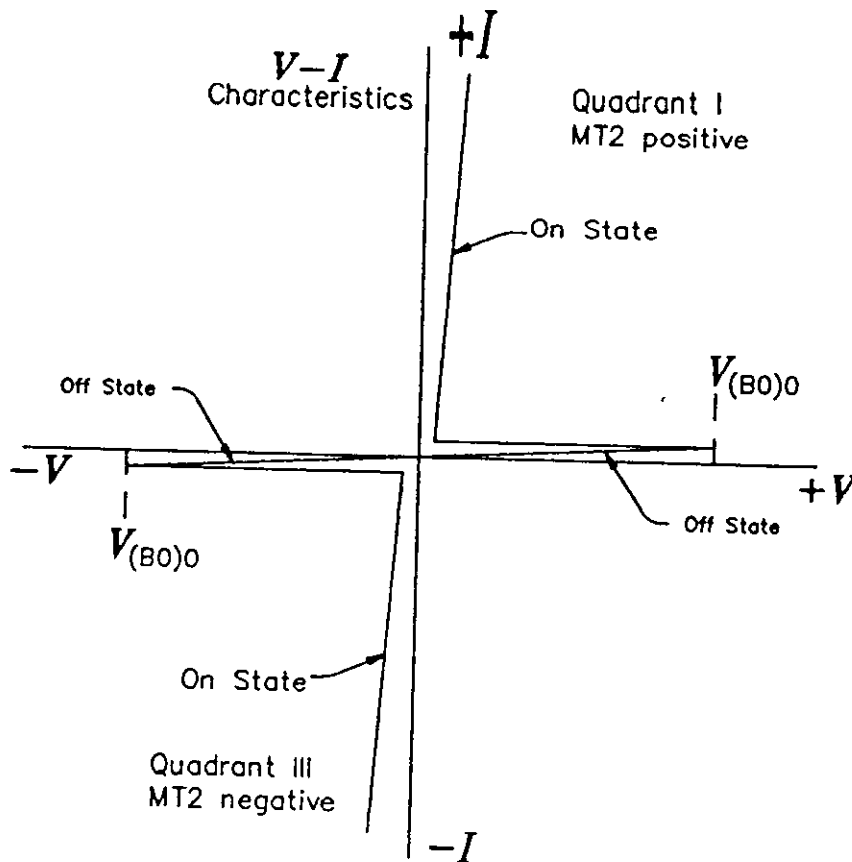


Figure 4.10 Voltage-Current Characteristics of the Triac [11] .

with respect to MT1 . This is also called (III-) mode .

The designation of modes I and III corresponds to quadrants I and III of the V-I characteristics and the (+) and (-) signs correspond to the polarity of the gate terminal with respect to MT1 .

The sensitivity of the triac is best in (I+) and (III-) modes in which the MT2 and the gate current polarities match . The operating modes in which both the main terminal (principal) and gate current flow through MT 1 in the same direction require less gate trigger current (more sensitive) than the other two modes in which the principal current and gate current oppose each

other . Figure 4.11 shows the four modes of operation of the triac with the corresponding current directions .

The break over voltage can be greatly reduced by injecting current at the gate terminal . However , when the triac switches into the ON state , the existence of the gate current becomes insignificant and the triac remains in that state until the main terminal current is reduced below the holding current value . Note that the triggering gate current can be either positive or negative irrespective of the main terminal voltage . If the main terminal current is reduced below the latching current to a value called ' Holding Current I_H ' the triac turns OFF [11] .

Power converter operation is based mainly on the switching of power semiconductor devices , a process that will introduce current and voltage harmonics into the supply system and on the output of the converters . As a result , problems of distortion of the output voltage , harmonic generation into the supply system , and interference with the communication and signaling circuits might appear . Hence , it is recommended that input and output filters are introduced to the converter to minimize the harmonic level . Figure 4.12 shows a generalized power converter structure .

A triac usually switches from one state to another in a small time interval . The rapid switch from high impedance to low impedance state will cause a rapid rise of the load current from zero to its final value each half cycle ; electrical noise or Radio Frequency Interference

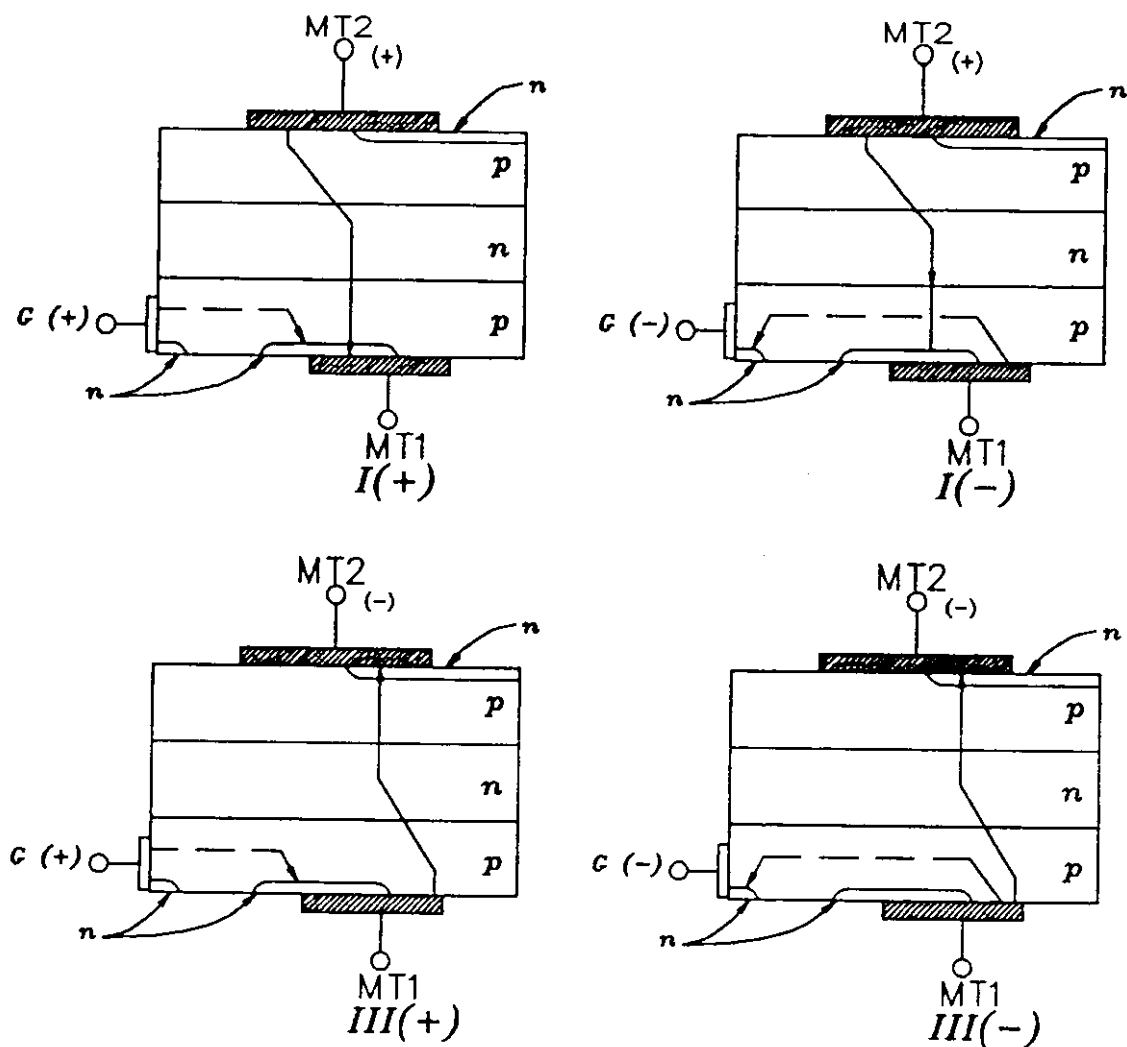


Figure 4.11 Modes of Operation of the Triac [11] .

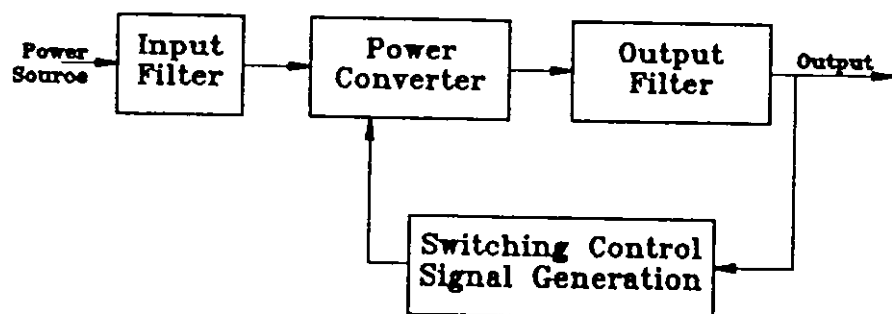


Figure 4.12 Generalized Power Converter Structure .

(RFI) is produced due to this rapid rise . The RFI is radiated through air and is conducted through power lines , thus affecting other equipment tied to the same power line . For resistive loads such as lamps, the peak load current can amount to several amperes . The RFI problem can be solved by using an inductance L which greatly attenuates the high frequency components of the RFI by limiting the rate of rise of the current . In addition , the use of a capacitance which bypasses the high frequency components of the RFI , prevents their propagation into the power line . This is called an RFI suppression network .

For high power applications , the load current is much larger than few amperes ; the use of large bulky inductors to suppress the noise is impractical . The solution to the RFI problem is obtained through a complex circuitry called ' Zero Voltage Switching ' or ' Zero Crossing Circuit ' .

The basic principle applied in the zero voltage switching circuit is to avoid the sudden high current jumps associated with triac switching . This is simply accomplished by turning the triac ON at the beginning of a zero voltage crossing ; thus , the starting turn on current is very small (approximately zero) .

The interface between thyristors and logic devices is a process in which the logic level signal is interpreted as a gate signal to trigger the device . Due to the electrical noise produced in the circuit , erroneous signals might be generated in the gating circuit .

The CMOS (Complementary Metal Oxide Semiconductor) IC logic family is most suitable for such an interface due to its high noise immunity . Noise immunity is a measure of how much noise voltage can a device tolerate before changing its output logic level unintentionally . Noise immunity is quantitatively specified as a noise margin in units of Volts . The CMOS IC logic family has a 45% of VDD noise margin where VDD is the supply voltage . Note , however that TTL logic level devices have a low noise margin and thus , might not be suitable for such an interface . However , due to the existence of the channel resistance , equivalent to the DC resistance between source and drain , the sink and the source current of the CMOS gate is limited to a maximum value of 10 mA . This current is not suitable for driving high power , AC loads that require powerful triacs with high gate currents . The solution to such a problem is to buffer the CMOS gate using a transistor ; the CMOS signal drives the transistor which will drive the triac [11] .

To compensate for the complex interface circuitry mentioned above , an optocoupler is used to provide the interface with a guarantee of high isolation between the logic system and the line driver load . It works as an input filter to the power converter and it protects the logic system against any high voltage or current leakage . Moreover , a high switching speed is provided .

Optocouplers , also known as Optically Coupled Isolators (OCI) , consist of a light emitting device , usually Infra-Red Light Emitting Diode (IRED) , and a

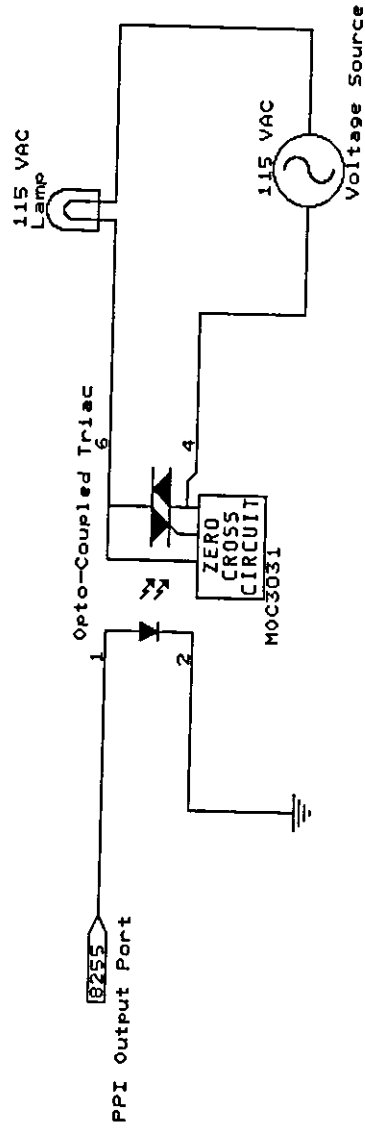
photo detector , a light sensitive triac in our case . The components are housed in a light excluding package , separated by a transparent insulation within the package .

Each PPI output signal is connected to a power converter IC to provide the required power amplification . The PPI output pin is connected to pin 1 of the MOC3031 IC which represents the anode terminal of the IRED . Pin 2 of the MOC3031 (cathode of the IRED) is connected to the ground point of the logic system . On the other insulated side of the power conversion IC the load is connected to pin 6 (MT 2 of triac) of the IC ; pin 4 (MT 1 of triac) is connected to the AC power supply . Figure 4.13 shows the MOC3031 connections with the PPI , load , and AC power supply .

When the PPI output is latched to ' High ' , it drives the IRED transmitter . These light emissions are detected by the light sensitive gate and the triac is switched ON allowing main terminal current passage through the load . The triac remains in the ON state (conduction state) as long as the PPI output remains ' High ' . If the PPI input changes state , the triac will turn OFF (commutate) with the next zero crossing of the AC load current .

Twelve opto-coupled triacs (MOC3031) are connected to the output ports of the PPI . The TTL level signals provided by the PPI are converted into 115 VAC signals capable of driving AC loads .

Appendix C.2 gives data sheets for the MOC3031 opto-coupled triac .



INDUSTRIAL ENGINEERING DEPARTMENT JORDAN UNIVERSITY	
Title	Power Amplification Wiring
Size	Document Number
A	Figure 4.13
REV	1
Date:	January 10, 1994
Sheet	of

4.3. Hardware Setup :

The prototype interface card includes two peripheral devices , the PPI and the timer / counter device . These are physically located at the addresses Hex 304 - 307 and Hex 300 - 303 , respectively , by positioning the DIP switch at (1000) .

At the startup of the personal computer , the 'Reset' signal goes ' High ' resulting in that all the PPI ports will be in the high impedance state . After the ' Reset ' signal is removed , all the ports of the PPI are set up to operate in the input mode [10] .

For the PPI to suit our application , operation in mode 0 should be selected and the ports should be configured to operate as mentioned in section 4.2. above . This can be done by issuing a single output instruction that loads the control register , located at Hex 307 , with the value (10011000) .

When the Hex 307 location is addressed , the address line ' A9 ' is ' High ' ; thus ; the comparator chip is enabled . The comparator output is also enabled since the (A8 - A5) values match the corresponding values of the DIP switch .

The set up step is a ' WRITE ' process ; the -IOW signal is ' Active Low ' . Hence , the outputs of the logical gates (ICs 8 and 9) are enabled , thus enabling the decoder (74LS138) and the bus transceiver (74LS245). Accordingly , the bus transceiver allows data flow from the CPU through the buffer stage , via the data lines to both the timer and the PPI . However , the output of the

decoder, ' Select 1 ' in this case , determines which peripheral device is to be enabled to receive the information sent via the data lines . The output of the decoder is determined by the value of the address lines (A4 - A2) , which is (001) in the case of the PPI and (000) in the case of the timer. The -IOW is also buffered to the ' WR ' pin of both the timer and the PPI to designate a ' WRITE ' operation . Moreover , the address lines A1 and A0 are also buffered to the A1 and A0 pins of both the timer and the PPI to specify which internal block of the chip is to be accessed ; the control register is accessed in the set up process since the address lines A1 and A0 have the value of (11) .

Similarly , the timer / counter device is set up using a suitable control word that is loaded at the address Hex 303 . In this case , the decoder ' Select 0 ' signal is ' Active Low ' and the timer / counter device is enabled to receive the control word sent via the data bus . A1 and A0 have a value of (11) that specify the control register of the timer .

4.4. Reading The Inputs :

This activity includes accessing the interface card at the locations Hex 304 (port A of the PPI) and Hex 306 (port C (upper)) through a ' READ ' operation , transferring the input data via the data bus and through the buffering bus transceiver to the CPU side , and storing the input data bits in their corresponding allocated memory bits .

Twelve inputs are available in our application ;

these are physically connected to port A pins and port C (upper) of the PPI . Two input instructions addressed to Hex 304 and Hex 306 accomplish the sequence of steps needed to access the interface card . The input process is a 'READ' activity that is similar in concept to the 'WRITE' activity mentioned above; however , the -IOR signal is 'Active Low ' during the ' READ ' activity rather than the -IOW signal . When ' Active Low ' , the -IOR signal reverses the direction of data flow through the bus transceiver ; data are transferred from the peripheral device , via the data bus , through the bus transceiver , and to the CPU side of the interface .

4.5. Updating The Outputs :

This activity includes reading the output data bits from their corresponding allocated memory bits , accessing the interface card at locations Hex 305 and Hex 306 (port B and port C (lower) of the PPI) through a ' WRITE ' operation and transferring the output data from the CPU through the buffering bus transceiver via the data bus to the data buffers of port B and port C (lower) .

Since ten outputs are needed in our application (Mercury robot) , two output instructions have to be performed . The first output instruction directs eight output data bits to port B at location Hex 305 via the 8-bit data bus available . The second , directs the remaining two output data pins to port C (upper) at the location Hex 306 .

In concept , updating the outputs is a ' WRITE ' process similar to setting up the devices ; however , the

type of data sent via the data bus and the addresses at which the data is sent differ .

Chapter Five

Software Design

The software design is concerned with the development of a ' Boolean Language ' environment that facilitates writing user programs in a simple user friendly manner and of a software program that will simulate the scanning process performed by the PLC . Set up of the different peripheral devices and configuration of the memory are performed by the software component .

5.1. Programming Language Used :

Turbo Pascal version 6 (TP-V6) is selected to implement the software programming required . It is a third generation , procedural programming language that supports object-oriented programming facilities . TP-V6 is selected due to the following reasons :

- 1) its efficiency in dealing with boolean logic and in performing bitwise operations ,
 - 2) the easiness through which different input / output devices can be accessed ,
 - 3) the powerful built-in library provided by TP-V6 , and
 - 4) TP-V6 provides an object-oriented application framework for windowing programs ; this is called Turbo Vision .
- Through Turbo Vision , sophisticated , consistent interactive application can be built in a short time . This can be useful for future development of an interactive user environment for the PLC [12] .

5.2. Menu System :

A simple menu system was developed with five

operations ; figure 5.1 shows the menu options provided for the user . The menu system can be encountered by typing 'MENU' at the operating system (DOS) prompt .

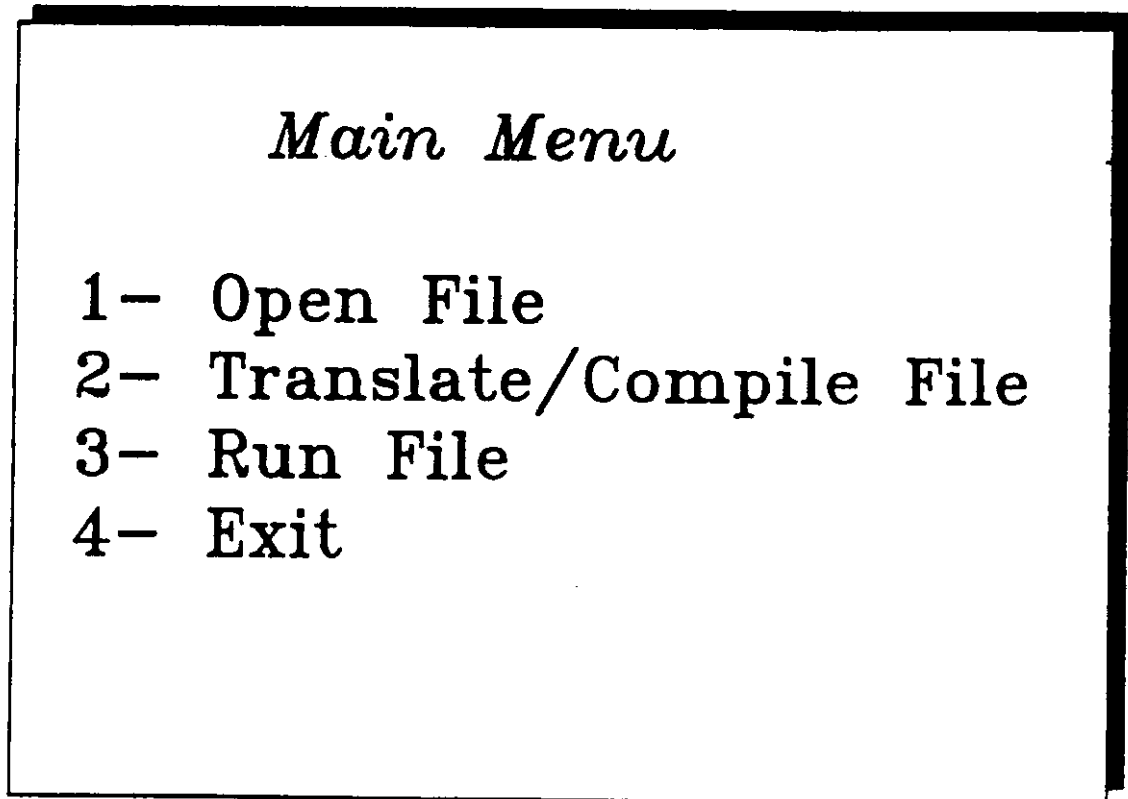


Figure 5.1. Menu system options .

Through selecting one of the five options provided , the user can open a user-program file , translate a user-program file into Pascal code , compile a Pascal program , run a compiled program , and / or exit the menu system .

5.3. Boolean Language :

The Boolean Language is a basic level language for the PLC that consists of a set of boolean operators such as AND, NOR , and OR and a set of mnemonic instructions that completely compensates for the different functionalities of the ladder diagram instruction set [1] . Texas Instruments

Model 510 PLCs use Boolean Language for user programs .

A Boolean Language environment is developed using TP-V6 ; this environment facilitates the entry of user programs through a three-letter mnemonic set which are , then , translated to Pascal statements to be compiled and executed.

A new user program can be entered by selecting option #1 in the menu system , which is ' Open File ' . The user will be transferred to the Integrated Development Environment (IDE) of TP-V6 through which the user can open a file , write his program and save it . The user program file should follow the DOS naming rules and must be ended with the extension ' .PLC ' . This environment is also used for making changes on any previously created files . Upon ' Exiting ' this development environment , the user will be provided with the menu system again .

Table 5.1 shows a list of the Boolean Language operators and mnemonics implemented . This instruction set can be further developed and enhanced to provide other functionalities that might be needed in certain applications . Section 5.7. is devoted to describe the Boolean Language rules .

5.4. User Program Translation Compilation :

After a user program is entered and saved in a Boolean Language code , it should be translated into a Pascal code with which the other parts of the developed software can communicate . This can be done by selecting the option #2 in the menu system , which is ' Translate File ' .

Table 5.1 Boolean Language (Mnemonics and Operators)

Mnemonic	Function
STR	Start
AND	And Point
OR	Or Point
AND NOT	And Not (NAND) Point
OR NOT	Or Not (NOR) Point
OUT	Energize Coil
OUT NOT	De-energize Coil
TMR	Timer
ENT	Enter Preset Value For Timer
RST	Reset Timer to Zero
MCR	Master Control Relay
END	End of Master Control Relay

The translator program prompts the user to enter the name of the file to be translated and the name of the new translated destination file , both of which must be entered without extensions . The extensions ' .PLC ' and ' .PAS ' will be provided by default for both files , respectively .

During the translation process , the user 'filename.PLC' program is checked for any syntax errors , the boolean code is translated into Pascal code , and a new intermediate file ' filename.PAS ' is created in which the new Pascal code is written .

Syntax errors in the user programs are checked

according to Boolean Language rules . Upon detecting a syntax error , a suitable error message is displayed that indicates the type of syntax error and the line at which it is detected ; the translation process is , then , 'Halted'. If there were no syntax errors detected , the translation is completed successfully resulting in Pascal code , corresponding to the translated boolean code, that will be written in a temporary file for further processing .

Actually , the translated Pascal code is not a complete program that can be compiled and executed ; it is only a group of properly coded Pascal statements that will constitute a part of a procedure in another Pascal program. Merging of Pascal code is done in a way that the merged product will be a valid Pascal program ; this is done automatically by a program called ' COMBINE ' .

The ' filename.PAS ' resulting from the merging process is a syntactically valid Pascal program that should be compiled and run .

After a user program file is translated and merged into a proper Pascal program , compilation is required to get an executable file to be run . Compilation is done , actually , by calling the TP-V6 compiler to compile the ' filename.PAS ' file into ' filename.EXE ' file which contains executable code .

The user should not face any problems during the compilation stage supposing that the user boolean program was properly translated and merged .

5.5. Running a User Program :

Running the user programs is , actually , using a

prototype PLC to control a certain process according to logical rules and instructions implied in the user program. Selecting option #4 in the menu system , which is ' Run File ' , will load the ' filename.EXE ' file into the memory to be executed . The user is prompted to enter the name of the file to be run ; a default extension of '.EXE' is assumed .

Figure 5.2 outlines the stages used to implement a user program for the prototype PLC starting with opening a file and ending with running that file .

The loaded executable file will mainly perform two operations :

- 1- Memory configuration and peripheral devices setup .
- 2- Scanning .

5.5.1. Memory Configuration and Peripheral Devices Setup :

The memory of the prototype PLC is configured by allocating the memory bits that correspond to both the real physical outputs and inputs of the system ; and by allocating the required memory to represent the internal outputs and timers . Memory allocations for the different program variables is also performed and initialization of all the above mentioned memory space is done .

The prototype card peripheral stage contains two peripheral devices that should be set up in specific modes of operation to suit the application . This is done at the startup of the prototype PLC . When the personal computer is switched ON or reset , the operating system sets up the different hardware configurations to work in certain specific modes of operation . For example , the main memory

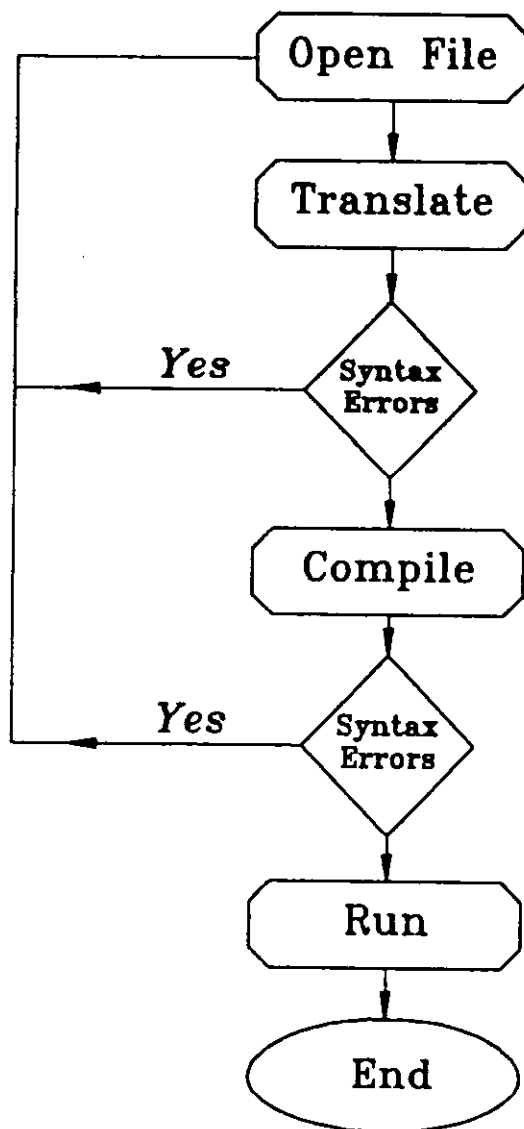


Figure 5.2 Implementation Stages of a User Program.

is configured and the serial port is set up to work at a certain specified dual rate .

When the ' Reset ' signal of the PPI goes 'High' , at startup or reset of the personal computer , all the ports of the PPI will be in the high impedance state . After the ' Reset ' signal is removed , all the ports remain in the

input mode [10] .

The software is designed to set up the PPI to operate in mode 0 with the different ports configured to operate as mentioned in the Hardware Design chapter above ; i.e , port A and port C (upper) are designed as input lines and port B and port C (lower) are designed as output lines . This can be done using a single output instruction that loads the control register of the PPI addressed at Hex 307 with the value (10011000) .

The software is also designed to set up the 8253 timer to operate in mode 0 , namely , interrupt on terminal count. This is done using a single output instruction that loads the control register of the timer , addressed at Hex 303 , with the value (01010000) .

5.5.2. Scanning-:

The ' Scanning ' process is a continuous activity that reads the inputs , executes the user program , and updates the outputs in an interactive manner . It also provides , apparently , " parallel execution " of user program statements .

5.5.2.1. Reading The Inputs :

This activity includes accessing the interface card at the addresses of port A and port C (upper) , Hex 304 and Hex 306 respectively , through a ' READ ' operation , transferring the input data via the data bus and through the buffering stage to the CPU . The input data bits are stored in their corresponding allocated memory bits . These bits represent the real

physical inputs of the system .

Since only 8-bit data transfers are supported , two input instructions are needed to read twelve inputs . The first instruction reads port A , thus , 8 input bits are read in parallel . The second reads the 4-bit input data at port C .

5.5.2.2. Executing The User Program :

The user program statements are combinations of boolean inputs that determine the states of specific outputs . The boolean inputs are evaluated using the corresponding allocated memory locations , and the internal bits corresponding to the outputs are updated accordingly .

When the user program logic activates a timer with a certain preset value , the prototype card is accessed at the location of the first timer , Hex 300, and loaded with the preset value . Down counting starts and continues till the count value equals 0 ; the allocated memory location corresponding to that timer is set ' ON ' , accordingly .

Simulation of " parallel statement execution " is accomplished by performing two passes on the user program logic . Thus , no matter where the user statement is located in the program , it is treated correctly and its result is reflected on the other related program statements .

Appendix D.1 is a program listing of the prototype PLC basic Pascal program which represents the ' Executive ' . Program ' PLC.PAS ' is written in

a modular format with each module (procedure) performing a specific , well-defined task indicated , normally , by its name .

This program provides for :

- 1- twelve physical inputs addressed at 1 - 8 and 101 - 104 ,
- 2- twelve physical outputs addressed at 11 - 18 and 113 - 116 ,
- 3- thirty-two timers addressed at 901 - 932 , and
- 4- one-hundred-sixty-eight internal outputs addressed at 701 - 868 .

The procedure ' Read_Inputs ' is responsible for reading the physical inputs addressed at Hex 304 and Hex 306 . It is also designed to detect any input changes that might happen during time delays .

The ' Get_Flag ' is a boolean function frequently invoked by the ' Executive ' to check the value of any of the internal memory bits representing the different inputs , outputs , timers , and / or internal outputs . ' Get_Flag ' is ' TRUE ' if the memory bit is ' ON ' (1) ; it is ' FALSE ' if the memory bit is ' OFF ' (0) .

Procedure ' Set_Flag ' is also used frequently by the ' Executive ' to set the different allocated internal memory bits to either ' ON ' or ' OFF ' according to the user program logic .

' Timer_Off ' is a boolean function with a 'TRUE' value indicating end of count of the 8253 timer / counter device .

Procedure ' Set_Timer ' initializes a time delay according to a ' Preset ' value determined by the user program . It loads the 8253 timer / counter device with the preset value and waits for the timer output response . Meanwhile , it detects any changes at the input state . When the time period elapses , the internal memory bit representing the timer is set ' ON ' .

Procedure ' Update_Outputs ' is responsible for updating the physical outputs at the end of the scan cycle .

As it is implied by its name , the procedure 'Exec_User' contains the user program translated Pascal statements . These are embedded into the 'PLC.PAS' program by the ' COMBINE.PAS ' program .

The set up process is performed by the procedure ' Initialize ' .

The ' Executive ' starts with initializing the memory and peripheral devices . Next , scanning is performed ; it continues until any key press is detected .

5.5.2.3. Updating The Outputs :

This activity includes reading the output data bits from their corresponding allocated memory bits and accessing the interface card at the location of port B and port C (lower) of the PPI , Hex305 and Hex306 respectively , through a ' WRITE ' operation . The output data bits are transferred from the CPU through the buffering stage via the data bus to the

data buffer of ports B and C (lower) .

Since only 8-bit data transfers are supported , two output instructions are used : the first directs 8 output data bits to port B , addressed at Hex 305 , and the second , directs the remaining output data bits to port C (lower) , addressed at Hex 306 .

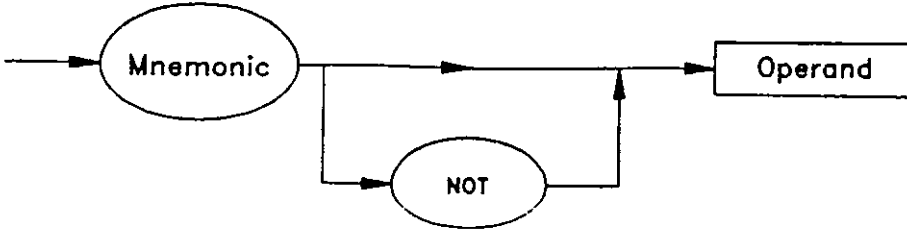
5.6. Boolean Language Rules :

Figure 5.4 shows the syntax diagrams for the Boolean Language instruction set implemented . To read a syntax diagram , follow the arrows .

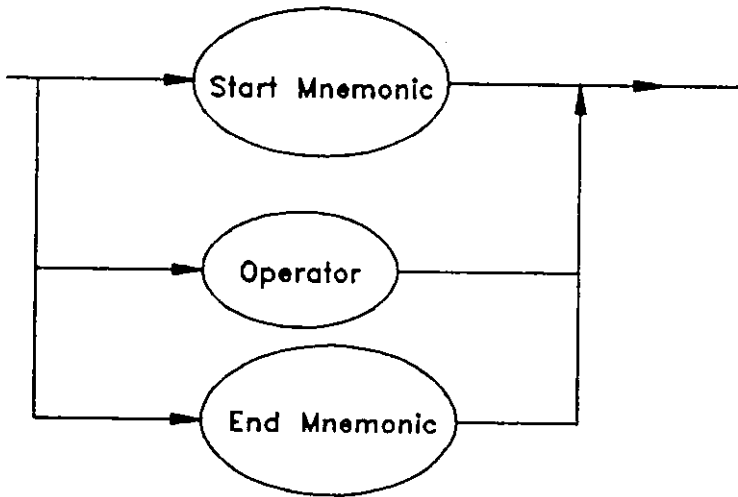
To write a user program in Boolean language , the following rules should apply :

- 1- Each mnemonic should exist on a separate line with its operand . This command line represents a part of a full command line consisting of one or more command lines . A full command line is equivalent to a rung in the ladder diagram .
- 2- A new full command line starts with one of the following mnemonics : STR , ENT , or END .
- 3- To continue with a started full command line , AND or OR mnemonics can be used repetitively to include related conditions . A limit exists on the number of nested ORings allowed , which is five .
- 4- To end a full command line , one of the following mnemonics is used : OUT , TMT , RST , or MCR .
- 5- The NOT operator is not allowed to be used with the mnemonics TMR , RST , ENT , and END .
- 6- A full command line ended with a TMR command line should be followed by a command line containing the ENT mnemonic

(a) Command Line Syntax



(b) Mnemonic



(c) Start Mnemonic

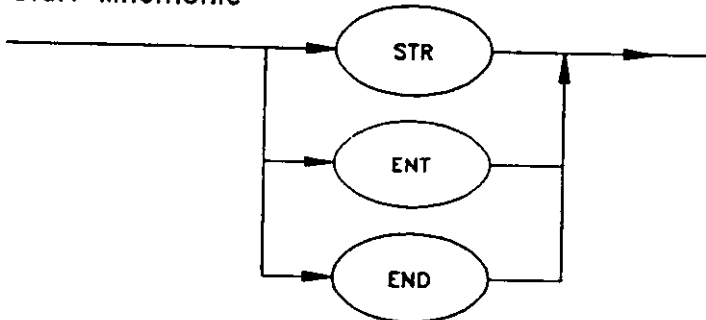
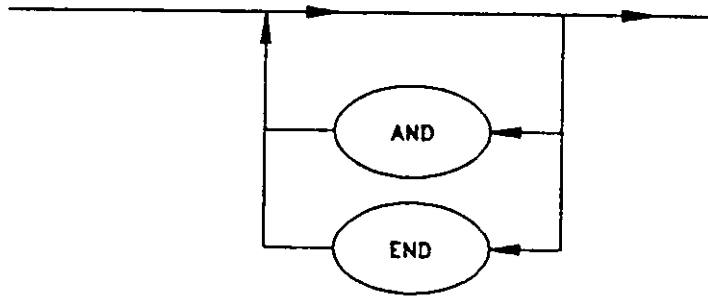
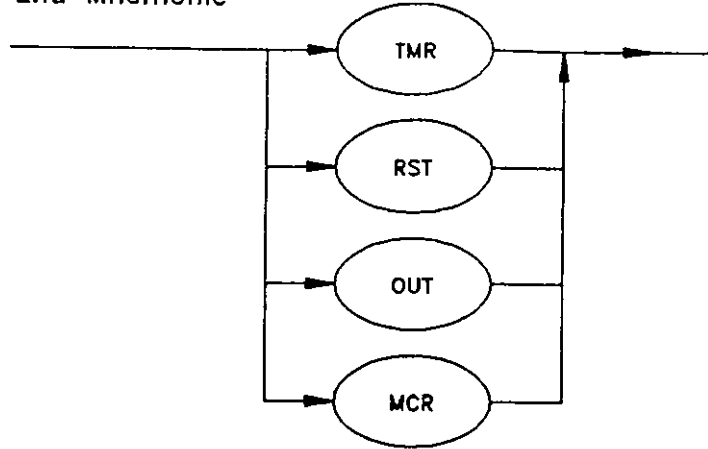


Figure 5.4 Syntax Diagram of Boolean Instruction set.

(d) Operator



(e) End Mnemonic



(f) Operand

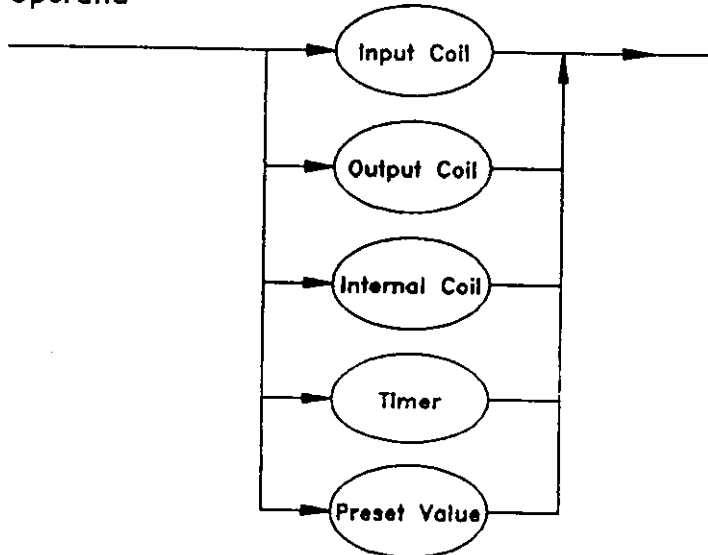


Figure 5.4 Continued .

to specify the preset value of the timer .

7- Each timer should be reset at some point in the program by an RST command line .

8- Each Master Control Relay (MCR) command line should be ended some point later in the program by an END command line . Nesting of MCRs is not allowed .

9- No limits , except those imposed by memory , are imposed on the number of full command lines used in the user program .

5.7. Program's Listing :

Appendix D.2 gives a listing of ' MENU.PAS ' program. Appendix D.3 gives a listing of the ' TRANSLATE.PAS ' program . Appendix D.4 gives a listing of the ' COMBINE.PAS ' program .

Table 5.2 is a list of the source and object code sizes , in bytes , of all the programs developed .

Table 5.2 Sizes (in Bytes) of Programs Developed .

Program Name	Source Code Size (Bytes)	Object Code Size (Bytes)
PLC	6667	8480
TRANSLATE	11829	11644
COMBINE	2299	6400
MENU	2625	6208

Chapter Six

Testing And Equipment

The prototype PLC is tested for proper operation using different programs . The testing process consists of two phases : 1- testing the basic operation of the PLC using a simulation board, 2- and testing the real operation of the PLC using the traffic light board through the power amplification stage . This is discussed in sections 4 and 5, respectively .

The following two sections include a list of the PLC components and a cost estimation of the project .

6.1. PLC Components :

Table 6.1 shows a listing of the components used to implement the different PLC stages ; the cost of each part is also included .

6.2. Cost Estimation :

Provided that the design of the PLC is made and that the software developed is available , the cost of construction of a PLC using a personal computer and a suitable I/O module would be , simply , the cost of the components used in constructing the PLC plus the man-power cost .

On average basis , the soldering of parts and wrapping of wires for the basic interface card is ten man hours . The construction of the power amplification stage is eight man hours .

Assuming that one man hour costs five J.D.s , the man-power cost would be 90 J.D.s to construct both the

Table 6.1 Components Listing .

No.	Item Description	# of Items	Cost/Item (J.D.)	Total Cost (J.D.)
<u>Interface Stage</u>				
1	JDR-PR2 Card	1	22.00	22.00
2	IC Sockets	8	00.20	01.60
3	Wire Wrapping Sockets	3	03.00	09.00
4	Line Drivers 74LS244	2	01.20	02.40
5	Bus Transceivers 74LS245	1	01.20	01.20
6	NAND Gates 74LS00	3	00.50	01.50
7	AND Gates 74LS08	1	00.50	00.50
8	Comparator 74LS85	1	00.60	00.60
9	3*8 Decoder 74LS138	1	00.70	00.70
10	4-Position Dip Switch	1	01.25	01.25
11	Timer / Counter 8253	1	04.00	04.00
12	Programmable Peripheral Interface PPI 8255A	1	05.00	05.00
13	Crystal 1 MHz	1	05.00	05.00
14	4.7 Kohm Resistors	5	00.05	00.25
15	1 Kohm Resistors	2	00.05	00.10
16	330 ohm resistor	1	00.05	00.05
17	10 micro Farad Tantalum Capacitors	2	00.25	00.50
18	0.1 micro Farad Capacitors	5	00.25	01.25
19	Wrapping Wires	3	08.00	24.00
20	1 Meter Ribbon Cable	1	03.50	03.50
				continue

Table 6.1 Continued

No.	Item Description	# of Items	Cost/Item (J.D.)	Total Cost (J.D.)
21	25-pin Connection (Plug-Socket)	1	01.50	01.50
22	Ribbon Cable Socket	1	02.50	02.50
	Sub Total			88.90
<u>Power Amplification Stage</u>				
1	Opto Coupled Triacs MOC3031	12	00.75	09.00
2	Connector (Plug)	1	01.00	01.00
3	Connectors (Socket)	10	00.60	06.00
4	Board Case	1	03.00	03.00
	Sub Total =			19.00
<u>Central Processing Unit</u>				
1	Personal Computer 80286	1	400.00	400.00
	Sub Total =			400.00
				=====
	Total =			507.90

interface circuit and the power amplification stage .

The total cost will be :

Total Cost = cost of components + man-power cost

Total Cost = 507.90 + 90.00 = 597.90 J.D.s

Note that the man-power cost can be reduced by manufacturing batches of printed circuit boards (PCBs) with the proper design .

6.3. Testing Basic Operation:-

A simple simulator board is constructed to test the basic operation of the prototype PLC at TTL level . The board is designed to simulate a discrete input stage represented by eight switches , and a discrete output stage represented by ten Light Emitting Diodes (LED's) . The above mentioned combination of eight inputs and ten outputs is selected with the Mercury Robot application in mind . With the existence of a suitable voltage level , the program tested using the simulator board can be used to control the Mercury Robot .

The operation of the prototype PLC is tested using different user programs that validate the proper functionalities of both the hardware and software aspects of the PLC .

Appendix E.1 gives a listing of a user program used to test the PLC functionality .

6.4. Traffic Light System:-

The PLC is tested for proper operation at a higher voltage level (115 VAC) using the traffic light simulator that was constructed at the Jordan University C.I.M. laboratory . Since the voltage operation level is 115 VAC , the constructed power amplification stage is used to provide the DC / AC conversion .

A previously written user program for the traffic

light system is translated into Boolean language and run to check for any problems in the prototype PLC ; the prototype PLC showed consistent operation . Appendix E.2 gives a listing of the traffic light system user program written using the boolean language .

6.5. Conclusions-:

The analysis , design , and construction of a prototype PLC is a basic step towards building robust knowledge in the automation field ; the amount of experience accumulated in the process is worth the effort .

The prototype PLC is considered to be a small PLC for the following reasons :

- 1- It can work with an 8-bit processor .
- 2- It is expandable to have 280 I/O points .
- 3- A maximum of twenty KBytes of memory is adequate for most of user programs .
- 4- It has a scan time of around 6 ms .
- 5- It provides for Boolean Language including timers and master control relays .

The prototype PLC can be further developed in terms of hardware and software :

- 1- Different I/O modules can be constructed ; for example , analog I/O modules .
- 2- A central processing unit (CPU) can be constructed in accordance with a power supply to provide for a dedicated PLC .
- 3- Proper manufacturing of reliable printed circuit boards (PCBs) that will survive the industrial environment can be established .

4- Further development of the Boolean Language to include sequencers , drums , counters , etc . is of vital importance .

At last , I would like to emphasize that we are aiming at accumulating sufficient knowledge about the environment we are dealing with . This is the basic step towards building good industrial basis for future development . The deeper our knowledge is , the greater our abilities to maintain , enhance , and further develop our environment , and the wider is the road heading to the future .

REFERNCES

- [1] Clarence , T. Jones , and Luis , A. Bryan ,
"Programmable Controllers" , International Programmable
Controls , Inc . , 1983 .
- [2] Bala , Ram and Steven , Lai , " The Development OF A
Program Generator For Programmable Logic Controllers " ,
Computers and Industrial Engineering , Volume 23 , No. 1-4,
pp. 335 - 339 , 1992 .
- [3] Sangeeta , Bhatnagar and Richard , J. Linn ,
"Autommatic Proramable Logic Controller Program Generator
With Standard Interface" , Manufacturing Review , Volume 3,
No. 2 , 1990 .
- [4] Hongzheng , Lu and Zhiguan , Ying , and . Warren ,
Liao , " Simulation Of Programmable Logic Controller " ,
Computers and Industrial Engineering , Volume 23 , No. 1-4,
pp. 351-354 , 1992.
- [5] Cengiz , Ozden and Bulent , Orenick , " Language
Facilities For Interrupt Handling on Programmable
Controllers " , Engineering Systems Design And Analysis ,
Volume 4 , 1992 .
- [6] David , Cranmer , " Implementing A PLC System " ,
Process and Control Engineering , Volume 44 , No. 10 ,
1991.

- [7] J. L. , Cornet , " A Modern Approach to PLC Programming in Mining Operations " , bulk solids holding , Volume 11 , No. 4 , November 1991 .
- [8] Ian , Verhappen , " Sample System Monitoring And Control Using Programmable Logic Controller " , Process-Control and Quality , Volume 3 , pp. 237-244 , 1992 .
- [9] " The TTL Data Book Volume 2 " , Texas Instruments , U.S.A., 1985 .
- [10] " Microprocessor Data Hand Book " , Micro-Tech Publications , Dubai , U.A.E. , 1991 .
- [11] Martin , Newman , " Industrial Electronics And Controls" , Spring Garden college , Prentice-Hall International , Inc. , 1986.
- [12] Tom , Swan , " Mastering Turbo Pascal " , Hayden Books, 1991 .
- [13] Muhammad , Harunur Rashid , " Power Electronics: Circuits, Devices, And Applications " , Purdue University Calumet , Prentice-Hall International , Inc. , Englewood Cliffs , New Jersey , 1988 .

APPENDIX AAPPENDIX A.1

Table A.1.1 Typical Standard Features of Small PLCs [1] .

- Up to 128 I/O .
- 4- or 8- bit processor .
- Relay replacing only .
- Memory up to 2K words.
- Digital I/O .
- Local I/O only .
- Ladder or Boolean Language only .
- Timers/Counters/Shift registers (TCS) .
- Master Control Relays .
- Drum Timers or Sequencers .
- Generally programmed with hand-held programmer .

APPENDIX A.2

Table A.2.1 Typical Standard Features of Medium Size PLCs [1] .

- Up to 1024 I/O .
- 8- bit processor .
- Relay replacing and analog control .
- Typical memory up to 4K words .
Expandable to 8K .
- Digital I/O .
- Analog I/O .
- Local and remote I/O .
- Ladder or Boolean language .
- Functional block/high level language .
- TCSs .
- MCRs .
- Jump .
- Drum Timers or Sequencers .
- Math capabilities .
 - Addition
 - Subtraction
 - Multiplication
 - Division
- Limited data handling .
 - Compare
 - Data conversion
 - Move register/file
 - Matrix functions
- Special function I/O modules .

continue

Table 2 Continued .

- RS-232C communication port .
- Local Area Networks (LANs) .
- CRT programmer .

APPENDIX A.3

Table A.3.1 Typical Standard Features of Large PLCs [1] .

- Up to 2048 I/O .
- 8- or 16- bit processor .
- Relay replacing and analog control .
- Typical memory up to 12K words ; Expandable to 32K .
- Digital I/O .
- Analog I/O .
- Local and remote I/O .
- Ladder or Boolean language .
- Functional block/high level language .
- TCSs .
- MCRs .
- Jump .
- Drum Timers or Sequencers .
- Math capabilities .
 - Addition
 - Multiplication
 - Square root
 - Subtraction
 - Division
 - Double precision
- Extended data handling .
 - Compare
 - Data conversion
 - Binary tables
 - ASCII tables
 - Move register/file
 - Matrix functions
 - Block transfer

continue

Table A.3.1 Continued .

- Special function I/O modules .
- PID modules or system software PID .
- One or more RS-232C communication port .
- Local Area Networks (LANs) .
- Host computer communications modules .
- CRT programmer .

APPENDIX A.4

Table A.4.1 Typical Standard Features of Very Large PLCs [1] .

- Up to 8192 I/O .
- 16- bit processor or multi-processors .
- Relay replacing and analog control .
- Typical memory up to 64K words ;
Expandable to 128K .
- Digital I/O .
- Analog I/O
 - Remote analog I/O .
- Remote special modules .
- Local and remote I/O .
- Ladder or Boolean language .
- Functional block/high level language .
- TCSs .
- MCRs .
- Jump .
- Subroutines , interrupts .
- Drum Timers or Sequencers .
- Math capabilities .

- Addition	- Subtraction
- Multiplication	- Division
- Square root	- Double precision
- Floating point	- Cosine functions
- Powerful data handling	
- Compare	- Data conversion

continue

Table 4 Continued .

- Move register/file
 - Block transfer
 - ASCII tables
 - Matrix functions
 - Binary tables
 - LIFO
 - FIFO
- Special function I/O modules .
 - PID modules or system software PID .
 - Two or more RS-232Ccommunication port .
 - Local Area Networks (LANs) .
 - Host computer communications modules .
 - Machine diagnostics .
 - CRT programmer .

APPENDIX BAPPENDIX B.1

Following are the data sheets of the 74LS245 bus transceiver chip . Figure B.1.1 is a pin diagram of the chip .

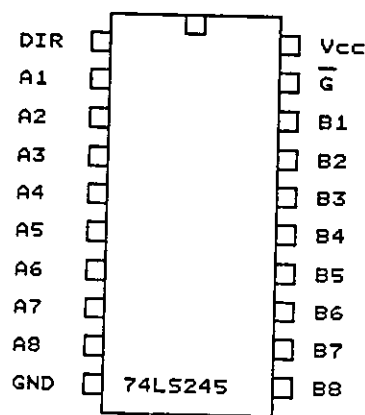


Figure B.1.1 Pin Diagram of the 74LS245 [9] .

These octal bus transceivers are designed for asynchronous two-way communication between data buses . The control function implementation minimizes external timing requirements .

Table B.1.1 is a function table of the 74LS245 ; table B.1.2 indicates the recommended operating conditions of the 74LS245 .

Table B.1.1 Function Table of the 74LS245 [9] .

Enable	Direction Control	Operation
G	Dir	
L	L	B Data to A Bus
L	H	A Data to B Bus
H	X	Isolation

Table B.1.2 Recommended Operating Conditions of the 74LS245 [9] .

Parameter	Min.	Nom.	Max.	Unit
V_{CC}	4.75	5	5.25	V
I_{OH}			-15	mA
I_{OL}			24	mA
T_A	0		70	°C

APPENDIX B.2

Following are the data sheets of the 74LS244 line drivers . Figure B.2.1 is a pin diagram of the chip .

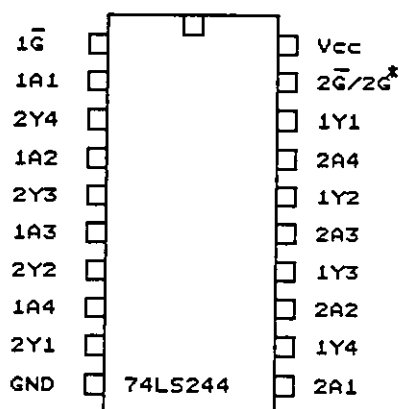


Figure B.2.1 Pin Diagram of the 74LS244 [9] .

These octal buffers and line drivers are designed specifically to improve both the performance and density of three-state memory address drivers , clock drivers , and bus oriented receivers and transmitters . These devices feature high fan-out , improved fan-in , and 4000 micro Volts noise margin .

Table B.2.1 indicates the recommended operating conditions of the 74LS244 .

Table B.2.1 Recommended Operating Conditions of the 74LS244 [9] .

Parameter	Min.	Nom.	Max.	Unit
V_{CC}	4.75	5	5.25	V
V_{IH}	2.00			V
V_{IL}			0.8	V
I_{OH}			-15	mA
I_{OL}			24	mA
T_A	0		70	$^{\circ}C$

APPENDIX B.3

Following are the data sheets of the 74LS85 comparator . Figure B.3.1 is a pin diagram of the 74LS85 .

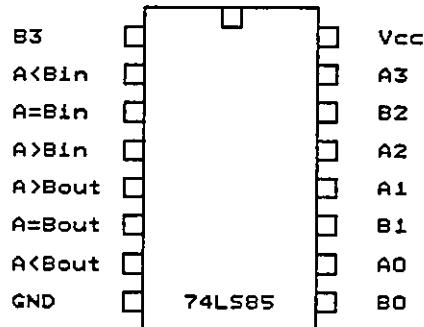


Figure B.3.1 Pin Diagram of the 74LS85 [9] .

These four bit magnitude comparators perform comparison of straight binary and straight BCD (8-4-2-1) codes . Three fully decoded decisions about two 4-bit words (A and B) are made and are extremely available at three outputs .

Table B.3.1 is a function table of the 74LS85 ; table B.3.2 indicates the recommended operating conditions of the 74LS85 .

Table B.3.1 Function Table of the 74LS85 [9] .

Comparing Inputs				Cascading Inputs			Outputs		
A3,B3	A2,B2	A1,B1	A0,B0	A>B	A<B	A=B	A>B	A<B	A=B
A3>B3	x	x	x	x	x	x	H	L	L
A3<B3	x	x	x	x	x	x	L	H	L
A3=B3	A2>B2	x	x	x	x	x	H	L	L
A3=B3	A2<B2	x	x	x	x	x	L	H	L
A3=B3	A2=B2	A1>B1	x	x	x	x	H	L	L
A3=B3	A2=B2	A1<B1	x	x	x	x	L	H	L
A3=B3	A2=B2	A1=B1	A0>B0	x	x	x	H	L	L
A3=B3	A2=B2	A1=B1	A0<B0	x	x	x	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	H	L	L	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	H	L	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	x	x	H	L	L	H
A3=B3	A2=B2	A1=B1	A0=B0	H	H	L	L	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	L	H	H	L

H = high level ; L = low level ; x = irrelevant

Table B.3.2 Recommended Operating Conditions of the 74LS85 [9] .

Parameter	Min.	Nom.	Max.	Unit
V_{CC}	4.75	5	5.25	V
I_{OH}			-400	mA
I_{OL}			16	mA
T_A	0		70	°C

APPENDIX B.4

Following are the data sheets of the 74LS138 decoder.
Figure B.4.1 is a pin diagram of the 74LS138 .

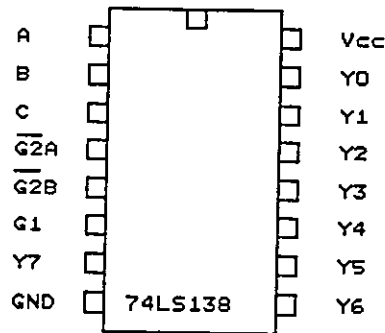


Figure B.4.1 Pin Diagram of the 74LS138 [9] .

These Schottky-clamped TTL MSI circuits are designed to be used in high-performance memory decoding or data-routing applications requiring very short propagation delay times . All these decoders feature fully buffered inputs , each of which represents only one normalized load to its driving circuit .

Table B.4.1 is a function table of the 74LS85 ; table B.4.2 indicates the recommended operating conditions of the 74LS85 .

Table B.4.1 Function Table of the 74LS138 [9] .

Inputs					Outputs							
Enable		Select										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
x	H	x	x	x	H	H	H	H	H	H	H	H
L	x	x	x	x	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	L	H	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

* $\overline{G2} = \overline{G2A} + \overline{G2B}$; H = high level ; L = low level ;
 x = irrelevant .

Table B.4.2 Recommended Operating Conditions of the
74LS138 [9] .

Parameter	Min.	Nom.	Max.	Unit
V_{CC}	4.75	5	5.25	V
V_{IH}	2.00			V
V_{IL}			0.8	V
I_{OH}			-1	mA
I_{OL}			20	mA
T_A	0		70	$^{\circ}C$

APPENDIX C

APPENDIX C.1

Following are some of the Silicon Controlled Rectifiers (SCRs) and Triac term definitions and performance parameters [13] :

- On-state current , $I_{T(AV)}$:

This is the average on-state current at a specified temperature . These data are normally quoted for a half-sine wave .

- RMS current , $I_{T(RMS)}$:

This is the root-mean-square value of on-state current. It signifies the heating effect due to (I^2R) dissipation and is limited due to thermal stress on device .

- Nonrepetitive rate of rise of on-state current , di/dt :

This is the maximum rate of rise of on-state current which the thyristor can withstand without being destroyed .

- Maximum repetitive peak reverse voltage , V_{RRM} :

This value defines the maximum permissible instantaneous value of repetitive applied reverse voltage that a thyristor can block .

- Maximum nonrepetitive peak reverse voltage , V_{RSM} :

This is the maximum instantaneous peak voltage of applied reverse voltage under transient conditions and for a specified time duration . V_{RSM} is typically 15% above V_{RRM} .

- Maximum repetitive peak off-state voltage , V_{DRM} :

This value defines the maximum permissible instantaneous value of repetitive applied forward voltage that a thyristor can withstand .

- Maximum nonrepetitive peak off-state voltage , V_{DSM} :

This is the maximum instantaneous peak value of applied forward voltage under transient conditions and for a specified time duration . V_{DSM} is typically 15% above V_{DRM} .

- On-state Voltage Drop , V_T :

This is the instantaneous value of on-state voltage drop and is dependent on the junction temperature , T_J . V_T can be considered as being made up of : (1) a value which is independent of the forward current and (2) a value which is proportional to the instantaneous forward current .

- Maximum peak on-state voltage , V_{TM} :

This is the maximum on-state voltage drop at a specified on-state current and junction temperature .

- Critical rate of rise of off-state voltage , dv/dt :

This is the minimum value of rate of rise of forward voltage which may cause switching from off-state to on-state.

- Latching current , I_L :

This is the minimum anode current which is required to maintain the thyristor in the on-state immediately after a thyristor has been turned on and the gate signal has been removed .

- Holding current , I_H :

This is the minimum anode current to maintain the thyristor in the on-state . The holding current is less than the latching current .

- Peak on-state current , I_p :

This is the peak instantaneous value of on-state current . It depends on the di/dt and width of current pulse . The switching losses would depend on the value of I_p , pulse width , and di/dt .

- Peak reverse current , I_{RM} :

This is the peak value of reverse current at the maximum junction temperature and maximum repetitive peak reverse voltage with gate open . This current would cause junction heating .

- Peak off-state current , I_{DM} :

This is the peak value of off-state current at the maximum junction temperature and maximum repetitive peak reverse voltage with gate open . This current would also cause junction heating .

- Junction-to-case thermal resistance , R_{thJC} :

This is the effective thermal resistance between the junction and outer case of the device .

- DC gate current trigger , I_{GT} :

This is the recommended value of the gate current , at a specified case temperature and anode voltage , required to trigger the triac .

- DC gate voltage trigger , V_{GT} :

This is the recommended value of gate voltage , at a specified case temperature , to trigger the triac .

- DC gate voltage not to trigger , V_{GD} :

This is the value of V_{GD} which does not cause the

thyristor to switch from on-state to off-state .

- Turn-off time , t_q :

This is the minimum value of time interval between the instant when the on-state current has decreased to zero and the instant when the thyristor is capable of withstanding forward voltage without turning on . t_q depends on the peak value of on-state current and the instantaneous on-state voltage .

APPENDIX C.2

Data sheet values of the opto-coupled triac MOC3031 are shown in table C.2.1 .

Table C.2.1 Data Sheets of MOC3031 Opto-Coupled Triac.

Parameter	Value
I_F	50 mA
V_T ($I_F = 30$ mA)	1.5 V
I_T	100 mA eff.
I_{GT}	15 mA (maximum)
V_{GT}	3.0 V
I_H	200 A
V_{TM}	3.0 V
I_{RM} ($V_R = 3$ V)	100 A
V_{RRM}	6 V

APPENDIX D.1

```

Program plc ;
Uses Crt ;

Label 25 ;

var
PPI_control ,           (* control word of PPI *)
Timer_control ,        (* control word of timer *)
Input_stat ,           (* internal bits corresponding to
Inputs_2 ,             input data *)
change_input_stat ,    (* detect any input changes during
change_inputs_2 ,     time delays *)
Outputs_1 ,            (* internal bits corresponding to
outputs_2              output data *)
                    : Byte ;
Interlocks : array[1..21] of Byte ;
    (* array representation of internal outputs *)
Timers : Array[1..4] of Byte ;
    (* array representation of timers *)
Timer_Preset : Array[1..32] of Integer ;
    (* array representation of timers preset values *)
procedure Read_inputs ;
begin
    input_stat := port[$304] or change_input_stat ;
    inputs_2 := (port[$306] div 16) or change_inputs_2 ;

```

```

end ;

Function check_ones : boolean ;
var
    output_test ,
        (* Intermediate byte to test physical outputs *)
    I      (* Counter of Ones *)
        : Byte ;

Begin
    Output_test := outputs_1 ;
    I := 0 ;
    (* counting number of 'ON' outputs in the first output byte

    While (output_test > 0) do
        begin
            Output_test := Output_test And (Output_test - 1) ;
            I := I + 1 ;
        end ;
        Output_test := Outputs_2 ;

    (* counting number of 'ON' outputs in the second output
    byte *)
    While (output_test > 0) do
        begin
            Output_test := Output_test And (Output_test - 1) ;
            I := I + 1 ;
        end;

    (* It is not allowed to have more than output 'ON' at the sam

```

```

time *)

  If ( I > 1 ) then Check_ones := False
  else Check_ones := True ;
End ;

Function get_flag (addr : Integer ) : boolean ;
var
  Test_byte ,      (* intermediate byte for check_up *)
  Element   ,      (* matrix element number *)
  Shift_count ,    (* # of shifts to the left to be made *)
  I : Byte ;       (* power index *)

begin
  test_byte := 1 ;

  If (addr > 0) And (addr <=8) then
    begin
      for I:= 1 to (addr-1) do
        test_byte := test_byte * 2 ;
        test_byte := test_byte and input_stat ;
      end

  Else if (addr > 10) And (addr <= 18) then
    begin
      for I:= 1 to (addr-11) do
        test_byte := test_byte * 2 ;
        test_byte := test_byte and outputs_1 ;
      end
    end
  end
end

```

```
Else if (addr > 100) And (addr <= 104) then
  begin
    for I:= 1 to (addr-101) do
      test_byte := test_byte * 2 ;
      test_byte := test_byte and inputs_2 ;
    end

Else if (addr > 112) And (addr <= 116) then
  begin
    for I:= 1 to (addr-113) do
      test_byte := test_byte * 2 ;
      test_byte := test_byte and outputs_2 ;
    end

Else if (addr > 700) And (addr <= 868) then
  begin
    Element := trunc((addr-700)/8) ;
    for I:= 1 to (addr-(element*8+701)) do
      test_byte := test_byte * 2 ;
      Test_byte := Test_byte and Interlocks[element+1] ;
    end

Else if (addr > 900) And (addr <= 932) then
  begin
    Element := trunc((addr-900)/8) ;
    for I:= 1 to (addr-(element*8+901)) do
      test_byte := test_byte * 2 ;
      Test_byte := Test_byte And Timers[element+1] ;
    end
  end
```

```
Else Exit ;          (* gets out of the function
                      immediately *)
```

```
If ( test_byte > 0 ) then
```

```
    get_flag := true
```

```
else
```

```
    get_flag := false ;
```

```
end ;
```

```
Procedure Set_flag ( addr : Integer ; status : boolean ) ;
```

```
var
```

```
    Oring_1 ,      (* value to be ORed to set a bit to
                   value = 'ON' *)
```

```
    Anding_1 ,    (* value to be ANDed to set a bit to
                   value = 'OFF' *)
```

```
    Result ,      (* intermediate variable *)
```

```
    Power ,       (* '2' to the 'power' = value of bit
                   required *)
```

```
    I ,           (* repetition Index *)
```

```
    Element       (* matrix element required *)
```

```
    : Byte ;
```

```
Begin
```

```
If (addr > 10) And (addr <= 18) then
```

```
    begin
```

```
        power := addr - 11 ;
```

```
        result := 1 ;
```

```
        for I := 1 to power do
```

```
result := result * 2 ;
    (* if power = 5 then result = 32 *)
If (status = true) then
    begin
        Oring_1 := result ;
        outputs_1 := outputs_1 Or Oring_1 ;
    end
else
    begin
        Anding_1 := 255 - result ;
        (* 255 = '11111111' *)
        outputs_1 := outputs_1 And Anding_1 ;
    end ;
end

Else if (addr > 112) And (addr <= 116) then
    begin
        power := addr - 113 ;
        result := 1 ;
        for I := 1 to power do
            result := result * 2 ;
            (* if power = 5 then result = 32 *)
            If (status = true) then
                begin
                    Oring_1 := result ;
                    outputs_2 := outputs_2 Or Oring_1 ;
                end
            else
                begin
```



```

        Anding_1 := 255 - result ;
            (* 255 = '11111111' *)
        outputs_2 := outputs_2 And Anding_1 ;
    end ;
end

Else If (addr > 700) And (addr <= 868) then
begin
    result := 1 ;
    element := trunc((addr - 700)/8) ;
    power := addr - (element*8 + 700) - 1 ;
    for I := 1 to power do
        result := result * 2 ;
            (* if power = 5 then result = 32 *)
    If (status = true) then
        begin
            Oring_1 := result ;
            Interlocks[element+1] :=
                interlocks[element+1] Or Oring_1 ;
        end
    else
        begin
            Anding_1 := 255 - result ;
                (* 255 = '11111111' *)
            Interlocks[element+1] :=
                interlocks[element+1] And Anding_1 ;
        end ;
    end
end
end

```

```

Else If (addr > 900) And (addr <= 932) then
  begin
    result := 1 ;
    element := trunc((addr - 900)/8) ;
    power := addr - (element*8 + 900) - 1 ;
    for I := 1 to power do
      result := result * 2 ;
      (* if power = 5 then result = 32 *)
    If (status = true) then
      begin
        Oring_1 := result ;
        Timers[element+1] :=
          Timers[element+1] Or Oring_1 ;
        Timer_Preset [ addr - 900 ] := 32767 ;
      end
    else
      begin
        Anding_1 := 255 - result ;
        (* 255 = '11111111' *)
        Timers[element+1] :=
          Timers[element+1] And Anding_1 ;
        Timer_Preset [ addr - 900 ] := 0 ;
      end ;
    end
  Else Exit ;
End ;

```

```

Function Timer_Off : Boolean ;

var
    Temp_Byte : Byte ;

begin

    Temp_Byte := Port[$306] div 16 ;
    Temp_Byte := Temp_Byte And $02 ;
                (* 00000010 --> PC5 of PPI *)
    Timer_Off := (Temp_Byte > 0 ) ;

end ;

Procedure Set_timer ( Addr : Integer ; Count : Integer ) ;
var

    temp_count , j : ShortInt ;
    i , Looper : Integer ;

begin

    Port[$303] := $50 ;
    for j := 1 to 70 do
        begin
            Looper := 52 * Count ;
            for i:=1 to Looper do
                begin
                    change_input_stat :=
                        port[$304] or change_input_stat ;

```

```

change_inputs_2 :=
    ( port[$306] div 16 ) or change_inputs_2 ;
Port[$301] := 255 ;
Repeat
Until Timer_Off ;
end ;
end ;

Set_Flag ( Addr , true ) ;
end ;

Procedure Update_Outputs ;
var
    I : Integer ;

begin
    port[$305] := outputs_1 ;
    port[$306] := outputs_2 ;
    for I := 1 to 32 do
        begin
            if (Timer_Preset [I] <> 0) and
                (Timer_Preset [I] <> 32767) then
                Set_Timer ( 900 + I , Timer_Preset [I] ) ;
            end ;
        end ;
    end ;

end ;

Procedure Exec_user ;
begin

```

USER PROGRAM IN PASCAL CODE

```
end ;

procedure initialize ;

var

    I : Byte ;

begin

    For I:= 1 to 3 do
        Timers[I] := 0 ;

    For I:= 1 to 21 do
        Interlocks[I] := 0 ;

    For I:= 1 to 32 do
        Timer_Preset[I] := 0 ;

    Input_stat := 0 ;
    inputs_2   := 0 ;
    change_input_stat := 0 ;
    change_inputs_2   := 0 ;
    outputs_1  := 0 ;
    outputs_1  := 0 ;
    Port[$305] := $00 ;
    Port[$306] := $00 ;

end ;
```

Begin

```
PPI_control := 98 ;      (* 10011000 *)
Port[$307] := PPI_control ;
      (* set_up mode and configuration *)
Timer_control := 50 ;    (* 00010000 *)
Port[$303] := Timer_control ;
      (* set_up of timer requested and mode *)
initialize ;
      (* initailize all internal bits to 0 *)
25 : Read_inputs ;
Exec_user ;
Exec_user ;
update_outputs ;
      (* If check_ones then update_outputs ; *)
if not KeyPressed then Goto 25 ;
End.
```

APPENDIX D.2

```
($M 16000 , 1000 , 2000)
```

```
(* 4 Kbyte stack , no Heap minimum or maximum *)
```

```
Program Menu ;
```

```
Uses Crt , Dos ;
```

```
var
```

```
Choice : char ;
```

```
FileName : String[15] ;
```

```
Compsec : String[80] ;
```

```
(* Center string Option at display line Row *)
```

```
Procedure Center ( Row : Integer ; Option : String ) ;
```

```
begin
```

```
GoToXY ( 40 - ( Length(Option) div 2 ) , Row ) ;
```

```
Write ( Option ) ;
```

```
end ;
```

```
Procedure DisplayMenu ;
```

```
begin
```

```
ClrScr ;
```

```
Center ( 8 , ' Main Menu' ) ;
```

```
Center ( 9 , '-----' ) ;
```

```
Center ( 10 , '1= Open File ' ) ;
```

```
Center ( 12 , '2= Translate File ' ) ;
```

```
Center ( 14 , '3= Compile File ' ) ;
```

```
Center (16 , '4= Run File      ') ;  
Center (18 , '5= Quit        ') ;  
end ;
```

```
Function Check_ExitCode : Boolean ;
```

```
var
```

```
Exit_Code : Integer ;
```

```
begin
```

```
Exit_Code := DosExitCode ;
```

```
if DosExitCode <> 0 then
```

```
begin
```

```
Check_ExitCode := False ;
```

```
GoToXY ( 1 , 24 ) ;
```

```
Writeln ;
```

```
Writeln ('Error #', DosExitCode) ;
```

```
Readln
```

```
end
```

```
else
```

```
Check_ExitCode := True ;
```

```
end ;
```

```
Begin
```

```
DisplayMenu ;
```

```
Repeat
```

```
Choice := ReadKey ;
```

```
Writeln ;
```

```
Writeln (Choice) ;
```

```
Case Choice of
```

```
'1' :
```



```
begin
    Exec ('Turbo.exe','') ;
    if not Check_ExitCode then halt ;
end ;
'2' :
begin
    ClrScr ;
    Write('File Name to Translate ? ') ;
    Readln(FileName) ;
    Exec ('Translate.exe',FileName) ;
    if not Check_ExitCode then halt ;
    Exec ('Combine.exe',FileName);
    if not Check_ExitCode then halt ;
end ;
'3' :
begin
    ClrScr ;
    Write('File Name to Compile ? ') ;
    Readln(FileName) ;
    Exec('Tpc.exe',FileName) ;
    if not Check_ExitCode then Halt ;
    Write('Press <Enter> to leave
           Compilation ...');
    Readln ;
end ;
'4' :
begin
    ClrScr ;
    Write('File Name to Run ? ') ;
```

```
    Readln(FileName) ;  
    FileName := FileName + '.EXE' ;  
    Center(15, ' Program is Running .. Press  
            Any key To Quit ');  
    Exec (Filename, '') ;  
    if not Check_ExitCode then Halt ;  
    end ;  
    '5' :  
        Halt ;  
    end ;  
    DisplayMenu ;  
    Until 'False' = 'True' ;  
    SwapVectors ;  
End.
```

APPENDIX D.3

```
<$M 4000 , 0 , 0 >
```

```
Program translate ;
```

```
type
```

```
String255 = string[255] ;
```

```
var
```

```
Infile , Outfile : Text ;
```

```
Line_Count : Integer ;
```

```
SavedExitProc : Pointer ;
```

```
Code_Line : string255 ;
```

```
ladder_flag ,
```

```
timer_flag ,
```

```
MCR_flag ,
```

```
bracket_count
```

```
: Byte ;
```

```
procedure CustomExit ; far ;
```

```
begin
```

```
if ( ExitCode <> 0 ) and ( ErrorAddr = nil ) then
```

```
begin
```

```
writeln ;
```

```
writeln('Program Halted ... ' ) ;
```

```
writeln('Exit Code = ',ExitCode) ;
```

```
write('Press <Enter> to leave ...');
```

```

    readln ;
end ;
exitproc := SavedExitProc ;
end ;

```

```

Procedure Syntax_Error ( Error_Num : integer ;
                        mnemonic : string ) ;

begin
    writeln('Syntax Error at line #',line_count) ;
    case Error_Num of
        1 : writeln('Mnemonic longer than allowed ...') ;
        2 : writeln('NOT or Operand expected ...') ;
        3 : writeln('Operand value missing ...') ;
        4 : writeln('MCR - END mismatch ...') ;
        5 : writeln (mnemonic,' is not allowed here ') ;
        6 : writeln (mnemonic,' is not allowed here ') ;
        7 : writeln ('ENT is expected after TMR ') ;
        8 : writeln (mnemonic,' is not allowed here ') ;
        9 : writeln ('MCR - END mismatch ') ;
        10 : writeln ('nested MCR is not allowed ') ;
        11 : writeln ('NOT is not allowed with ', mnemonic) ;
        12 : writeln('Ladder is incomplete ... ') ;
        13 : writeln('TMR command is not ended ... ') ;
        14 : writeln('MCR command is not ended ... ') ;
        15 : writeln('Invalid Value of Operand ... ') ;
        16 : writeln ('Syntax Error at line #',Line_count) ;
    else writeln ('Unrecognized Error !!!')
    end ;
end ;

```

```
Halt(1) ;

End ;

procedure OpenFiles ;
var
    FileName , PLC_File , PAS_File : string[14] ;

begin
    FileName := ParamStr(1) ;
    PLC_File := FileName + '.PLC' ;
    PAS_File := FileName + '.PAS' ;
    Assign(InFile,PLC_File) ;
    Reset (InFile) ;
    Assign (OutFile , PAS_File ) ;
    Rewrite (OutFile) ;
end ;

procedure ProcessString ( var Code_Line : string255 ) ;
var
    index , time_delay : Integer ;
    mnemonic , nnot , operand , MCR_Num : String[10] ;
    pascal_code : String ;

(*          Skip Spacse          *)

procedure Skip_Spaces ;

begin
```

```

while code_line[index] = ' ' do
inc (index) ;
end ;

```

```

(*                               Get Mnemonic                               *)

```

```

procedure Get_Mnemonic ;

```

```

var

```

```

    end_of_mnemonic : Boolean ;

```

```

begin

```

```

    mnemonic := '' ;

```

```

while ((code_line[index] <> ' ') and
      (index <= length(code_line))) do

```

```

begin

```

```

    mnemonic := mnemonic + upcase( code_line[index] ) ;

```

```

    inc (index) ;

```

```

end ;

```

```

if length (mnemonic) > 3 then

```

```

    Syntax_Error ( 1 , mnemonic ) ;

```

```

end ;

```

```

(*                               Check Operand Value                               *)

```

```

procedure Check_Operand ( operand : string ) ;

```

```

var

```

```

    i : integer ;
begin
    for i := 1 to length(operand) do
        if ( (ord(operand[i]) - $30 ) < 0 )
            or ( (ord(operand[i]) - $30 ) > 9 ) then
            Syntax_Error ( 15 , '!!!' ) ;
        end ;
    end ;

```

```

(*                      Check NOT                      *)

```

```

procedure Check_Not ;

```

```

var

```

```

    i : integer ;

```

```

begin

```

```

    nnot := '' ;

```

```

    operand := '' ;

```

```

    while ((code_line[index] <> ' ') and
           (index <= length(code_line))) do

```

```

        begin

```

```

            nnot := nnot + upcase( code_line[index] ) ;

```

```

            inc (index) ;

```

```

        end ;

```

```

    if length(nnot) > 3 then

```

```

        Syntax_Error ( 2 , mnemonic )

```

```

    else if nnot <> 'NOT' then

```

```

begin
    operand := '' ;
    for i := 1 to length(nnot) do
        operand := operand + nnot[i] ;
    nnot := '' ;
    end ;
end ;

```

```

(*                Get Operand                *)

```

```

procedure Get_Operand ;

```

```

begin
    operand := '' ;

    while ((code_line[index] <> ' ') and
           (index <= length( code_line ))) do
        begin
            operand := operand + code_line[index] ;
            inc (index) ;
        end ;

        if operand = '' then
            Syntax_Error ( 3 , mnemonic ) ;

        end ;

```

```

(*                Get Time Delay                *)

```



```

procedure Get_Time_Delay ;

begin
    time_delay := 0 ;

    while not
        (code_line[index] in [ chr($17) , chr($0) ] ) do
    begin
        time_delay := time_delay * 10 +
            ( ord ( code_line[index] ) - $30 ) ;
        inc (index) ;
    end ;

end ;

```

```

Procedure Translate ;

```

```

(*          Close Brackets          *)

```

```

procedure Close_Brackets ;

```

```

var

```

```

    i : Integer ;

```

```

begin

```

```

    For i := 1 to bracket_count do

```

```

        pascal_code := pascal_code + ')' ;

```

```

        bracket_count := 5 ;

```

```

end ;

```

```
procedure Check_MCR_Num ;
```

```
begin
```

```
  if operand <> MCR_Num then
```

```
    Syntax_Error ( 4 , mnemonic ) ;
```

```
end ;
```

```
procedure Lad_STR ;
```

```
begin
```

```
  if nnot = 'NOT' then
```

```
    pascal_code :=
```

```
      'if ( ( ( ( ( Not get_flag(' + operand + ')'
```

```
  else
```

```
    pascal_code :=
```

```
      'if ( ( ( ( ( get_flag(' + operand + ')'
```

```
    ladder_flag := 1 ;
```

```
    timer_flag := 0 ;
```

```
    bracket_count := 5 ;
```

```
end ;
```

```
procedure Lad_AND ;
```

```
begin
```

```
  if nnot = 'NOT' then
```

```
    pascal_code := ' And Not get_flag(' + operand + ')'
```

```
  else
```

```
    pascal_code := ' And get_flag (' + operand + ')'
```

```
end ;
```

```
procedure Lad_OR ;
```

```
begin
```

```
  if nnot = 'NOT' then
```

```
    pascal_code :=
```

```
      ') Or Not get_flag(' + operand + '))'
```

```
  else
```

```
    pascal_code :=
```

```
      ') Or get_flag (' + operand + '))' ;
```

```
  dec ( bracket_count ) ;
```

```
end ;
```

```
procedure Lad_OUT ;
```

```
begin
```

```
  if nnot = 'NOT' then
```

```
    begin
```

```
      close_brackets ;
```

```
      pascal_code := pascal_code + ' then set_flag(' +
```

```
        operand + ',false) ' + chr($0D) + chr($0A) +
```

```
        ' else set_flag(' + operand + ',true) ; ' ;
```

```
        (* Carriage Return + Line Feed *)
```

```
    end
```

```
  else
```

```
    begin
```

```
        close_brackets ;
        pascal_code := pascal_code + ' then set_flag(' +
        operand + ',true) ' + chr($0D) + chr($0A) +
        ' else set_flag(' + operand + ',false) ; ' ;
    end ;
    ladder_flag := 0 ;
end;

Procedure Lad_TMR ;
begin
    ladder_flag := 0 ;
    close_brackets ;
    pascal_code := pascal_code + ' then
        timer_preset(' + operand + ' - 900 ) := ' ;
    timer_flag := 1 ;
end ;

procedure Lad_ENT ;

begin
    timer_flag := 0 ;
    pascal_code := pascal_code + operand + ';' ;
end ;

Procedure Lad_MCR ;

begin
    ladder_flag := 0 ;
    close_brackets ;
```

```

    pascal_code := pascal_code + ' then begin ' ;
    MCR_flag := 1 ;
end ;

```

```

procedure Lad_END ;

```

```

begin
    ladder_flag := 0 ;
    pascal_code := ' end ; ' ;
    MCR_flag := 0 ;
end ;

```

```

procedure Lad_RST ;

```

```

begin
    ladder_flag := 0 ;
    close_brackets ;
    pascal_code := pascal_code +
        ' then set_flag(' + Operand + ',false)';
end ;

```

```

Begin

```

```

    if ( mnemonic = '' ) And ( nnot = '' )
        And ( operand = '' ) then
    else if mnemonic = 'STR' then Lad_STR
    else if mnemonic = 'AND' then Lad_AND
    else if mnemonic = 'OR' then Lad_OR
    else if mnemonic = 'OUT' then Lad_OUT

```

```

else if mnemonic = 'MCR' then Lad_MCR
else if mnemonic = 'TMR' then Lad_TMR
else if mnemonic = 'END' then Lad_END
else if mnemonic = 'ENT' then Lad_ENT
else if mnemonic = 'RST' then Lad_RST
else Syntax_Error ( 16 , mnemonic ) ;

```

```
End ;
```

```
procedure Error_Check ;
```

```
begin
```

```
(*          New Ladder Check          *)
```

```

if (ladder_flag = 1) and ( (mnemonic = 'STR')
    or (mnemonic = 'END') or (mnemonic = 'ENT') )
    then Syntax_Error ( 5 , mnemonic )

```

```

else if (ladder_flag = 0) and ( (mnemonic = 'AND')
    or (mnemonic = 'OR') or (mnemonic = 'OUT')
    or (mnemonic = 'MCR') or (mnemonic = 'TMR') )
    then Syntax_Error ( 6 , mnemonic ) ;

```

```
(*          Timer Check          *)
```

```

if ( timer_flag = 1 ) and ( mnemonic <> 'ENT' )
    then Syntax_Error ( 7 , mnemonic )

```

```

else if ( timer_flag = 0 ) and ( mnemonic = 'ENT' )
    then Syntax_Error ( 8 , mnemonic ) ;

```

```

(*                Master Control Relay Check                *)

```

```

if ( MCR_flag = 0 ) and ( mnemonic = 'END' )
    then Syntax_Error ( 9 , mnemonic )

```

```

else if ( MCR_flag = 1 ) and ( mnemonic = 'MCR' )
    then Syntax_Error ( 10 , mnemonic ) ;

```

```

(*                NOT Allowance Check                *)

```

```

if ( nnot = 'NOT' ) and ( ( mnemonic = 'TMR' )
    or ( mnemonic = 'MCR' ) or ( mnemonic = 'ENT' )
    or ( mnemonic = 'RST' ) or ( mnemonic = 'END' ) )
    then Syntax_Error ( 11 , mnemonic ) ;

```

```

End ;

```

```

begin

```

```

    index := 1 ;

```

```

    pascal_code := '' ;

```

```

    Skip_Spaces ;    (* to skip spaces starting code_line *)

```

```

    Get_Mnemonic ;

```

```

        (* to get the mnemonic at processed code_line *)

```

```

    if mnemonic <> '' then

```

```

begin
    Skip_Spaces ;
    (* to skip spaces between mnemonic and next operand

    Check_Not ;
    (* to read the next operand or NOT value *)

    if operand = '' then
        begin
            Skip_Spaces ;
            (* to skip spaces between NOT and operand *)
            Get_Operand ;
            (* To read operand value *) ;
        end ;
    Check_Operand ( operand ) ;

    if mnemonic = 'TMR' then
        begin
            Skip_Spaces ;
            (* to skip spaces between operand and time delay *)
            get_time_delay ;(* to read time delay value *)
        end ;
    Error_Check ;
    (* to check syntax errors in user program *)
    Translate ;
    (* Interpretation of user code into Pascal code *)
    end ;
    code_line := pascal_code ;
end ;

```

431757


```
Procedure ProcessFiles ;
```

```
var
```

```
code_line : string255 ;
```

```
begin
```

```
Line_Count := 1 ;
```

```
while not Eof(Infile) do
```

```
begin
```

```
readln(Infile , Code_line) ;
```

```
ProcessString ( Code_Line ) ;
```

```
writeln(Outfile, Code_Line ) ;
```

```
Inc(Line_Count) ;
```

```
end ;
```

```
Writeln(OutFile);
```

```
Writeln(OutFile);
```

```
Writeln(OutFile);
```

```
end ;
```

```
Procedure Check_Flags ;
```

```
begin
```

```
if ladder_flag = 1 then Syntax_Error ( 12 , '!!!!' ) ;
```

```
if Timer_flag = 1 then Syntax_Error ( 13 , '!!!!' ) ;
```

```
if MCR_flag = 1 then Syntax_Error ( 14 , '!!!!' ) ;
```

```
End ;
```

Begin

```
Timer_Flag := 0 ;
MCR_Flag   := 0 ;
Ladder_Flag := 0 ;
OpenFiles ;
ProcessFiles ;
Check_Flags ;
Close(Outfile) ;
Close(Infile) ;
SavedExitProc := ExitProc ; (* save ExitProc pointer *)
ExitProc := @CustomExit ;
write('Press <Enter> to end program ...') ;
Readln
```

End .

APPENDIX D.4

```
<$M 4000 , 0 , 0>
```

```
Program Combine ;
```

```
Uses Dos ;
```

```
var
```

```
    OutFile          : Text ;
```

```
    Filename         : String[20] ;
```

```
    Compsec          : String[80] ;
```

```
    SavedExitProc    : Pointer ;
```

```
Procedure CustomExit ; far ;
```

```
begin
```

```
    if ( ExitCode <> 0 ) and ( ErrorAddr = nil ) then
```

```
        begin
```

```
            Writeln ;
```

```
            Writeln('Program Halted ... ') ;
```

```
            Writeln('Exit Code = ',ExitCode) ;
```

```
            Write('Press <Enter> to leave ...');
```

```
            Readln ;
```

```
        end ;
```

```
        ExitProc := SavedExitProc ;
```

```
end ;
```

```
Procedure Instruct ;
```

```
begin
```

```
    writeln ;
```

```
Writeln ('Combine <output> <input1> <input2> ...<inputn>
```

```
    writeln ;
```

```
end ;
```

```
Function FileExists ( var Filename : Text ) : Boolean ;
```

```
begin
```

```
    <$I-> Reset (Filename) ; Close (Filename) ; <$I+>
```

```
    FileExists := ( IoResult = 0 ) ;
```

```
end ;
```

```
Procedure Combine_File ( FName : String ) ;
```

```
var
```

```
    InFile : Text ;
```

```
    S      : String[132] ;
```

```
begin
```

```
    Assign ( InFile , FName ) ;
```

```
    Reset (InFile) ;
```

```
    While not EOF (InFile) do
```

```
        begin
```

```
            Readln ( InFile , S ) ;
```

```
            Writeln ( OutFile , S ) ;
```

```
        end ;
```

```
    Close (InFile) ;
```

```
end ;
```

```
Procedure Process ;
```

```
begin
```

```

Assign ( OutFile , 'Temp.@@@' ) ;
Rewrite (OutFile) ;
Combine_File ( 'Start.@@@' ) ;
Combine_File ( FileName ) ;
Combine_File ( 'End.@@@' ) ;
Close (OutFile) ;
end ;

```

```

Procedure Rename_File ;

```

```

var

```

```

    Rename_Command , Delete_Command : String ;

```

```

begin

```

```

    Rename_Command := '/c rename temp.@@@ ' + Filename ;

```

```

    Delete_Command := '/c del ' + Filename ;

```

```

    Compsec := GetEnv('Compsec') ;

```

```

    if Length(Compsec) = 0 then

```

```

        Compsec := 'command.Com' ;

```

```

    {$I-} SwapVectors ;

```

```

    Exec (Compsec , Delete_Command) ;

```

```

    Exec (Compsec , Rename_Command) ;

```

```

    SwapVectors ; {$I+}

```

```

    if DosError <> 0 then

```

```

        Writeln('*** Error : Cannot Run Command.Com') ;

```

```

end ;

```

```

Begin

```

```

    SavedExitProc := ExitProc ;

```

```

    (* save ExitProc pointer *)

```

```
ExitProc := @CustomExit ;  
FileName := ParamStr(1) ;  
FileNAme := FileName + '.PAS' ;  
if ParamCount <= 0 then Instruct  else Process;  
    Rename_File ;  
write('Press <Enter> to end program ...') ;  
Readln  
End.
```

ملخص

تصميم و بناء نموذج أولي لجهاز تحكم منطقي مبرمج

الطالب : محمد أسامه طهبوب

المشرف : الدكتور خالد طوقان

في هذه الرسالة ، تم اتخاذ خطوة واسعة باتجاه بناء خيرة متراكمة في مجال أجهزة التحكم المنطقية المبرمجة . بداية تم أعداد دراسة عن مبدئية عمل هذه الأجهزة وشرح للأجزاء الأساسية ، و شملت هذه الدراسة أيضا اشكال ومجالات التطبيق . ثم تم تصميم وبناء نموذج أولي لجهاز تحكم منطقي مبرمج، حيث تم العمل على محورين متوازيين ومتراپطين معا هما محور التصميم والبناء للأجهزة والمعدات ومحور التصميم والتطوير للبرمجيات اللازمة.

على صعيد الأجهزة والمعدات تم بناء وتصميم نموذج أولي للجهاز بواسطة وحدة العمل المركزية لجهاز الحاسوب الشخصي لتوفير معالج ومصدر طاقة ، وأنجاز حزمة للأخراج والأدخال تتناسب مع مستوى فرق جهد متردد قيمته ١١٥ وحدة فرق جهد ويقوم بتوفير اثنتي عشرة إشارة كهربائية منفصلة صادرة، واثنتي عشرة إشارة كهربائية منفصلة داخلية. حزمة الإدخال والأخراج تتكون من قسمين أساسيين هما كرت اخراج/أدخال بيني وكرت لتحويل الطاقة من تيار ثابت الى متردد .

أما على صعيد البرمجيات ، فقد تم تطوير بيئة برمجية توفر للمستخدم إمكانية كتابة برامج مختلفة للتحكم بالجهاز وتنفيذها . أن تنفيذ هذه البرامج يقوم بمهمتين أساسيتين هما أعداد وترتيب النموذج الأولي للجهاز ثم إجراء عملية مسح متواصل .

في نهاية العمل تم فحص الجهاز بشكل مكثف لضمان جودة الأداء ودقته .