NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Electrical and Electronic Engineering

REMOTE PROCESS CONTROL

Graduation Project
EE - 400

Student: Ömer Akkanat (2001366)

Supervisor: Özgür Cemal Özerdem

Lefkoşa - 2002

TABLE OF CONTENTS

ACKNOWLEDGEMENT

First I want to thank Mr. Özgür Cemal Özerdem to be my Supervisor. Under his guidance, I successfully overcome many difficulties and learn a lot about PLC with EE 470 PLC course. PLC depends greatly on technology. In each discussion, he explained my questions patiently and answered my questions quickly and in detail. He always help me a lot either in my study.

Thanks to faculty of Engineering for having such a good computational environment. I want to thank all my student friends in NEU. Being with them make my during educational in NEU full of fun.

I also special thanks to Mr. Aziz Zenginses with their kind help. I could successfully to perform computational problems. He always help me a lot either in my studys

Finally I want to thank especially my parents. Without their endeless support and love for me. I wish daughter in the heaven be proud of me.

ABSTRACT

In this study, a system of Remote Process Control (REPCO) via telephone line with modem, network or internet is introduced. Minimum requirements to use this system are two computers: a server and a client connected through modem.

In this environment, the role of the server is to control the process via some sort of controller device. On the other hand, the client is to convey the control related commands and messages of the user to the server. Therefore, as long as there is a modem or network connection, the client can be anywhere.

In our research, we have used a Programmable Logic Controller (PLC) to transmit the equipment's data to the server computer and control the equipments by executing the commands received. In our approach, the server will process the incoming equipment related data and will send them, along with the associated screen snapshots, to the client. Server will also transmit the user and/or client-originated commands to the PLC. The fundamental idea of getting data and screen snapshots from the server and responding by sending the required commands will allow user/client to obtain the control authorization for monitoring the process remotely. Although the scope of this experiment contains a small infrastructure composed of one process, one server, and one client, the proposed method can easily be adapted to systems in different domains.

INTRODUCTION

The rapid developments in electronics have unstoppably revolutionized the Internet technology in recent years and have become a part of daily life, bringing in everything from Pampers to programmable logic devices. The rising usage of Internet has been complemented with developments in communication technology.

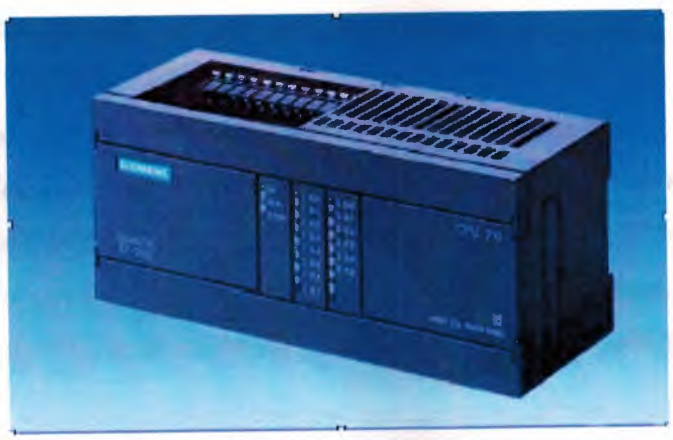


Fig. 1 Siemens S7 CPU212 PLC.

Moreover, Networking technologies start to invade the home to carry phone signals and TV programs, link computers and peripherals, and tap into the Internet (Data-Ray, 1999a and Dutta-Ray,1999b).These rapid developments show that Internet based systems have had a huge role in home and business solutions nowadays and will have more in future. Therefore, many software developers have been forced to find ways of developing systems more quickly than ever before.

Internet technology allows companies to overcome many of the physical constraints that often prevent them from doing business in distant markets, which means that an a commerce market is fundamentally global (Choi and Whinston, 1999).Communication Systems for Next-Generation Remote Education Using Asymmetrical Satellite and Terrestrial Networks (Yoshida et al., 1999) are a few examples of applications on Internet. Besides, in a recent study, a number of researchers (Mccalley et al., 1998) explained that

Internet technology provides a way to power engineering educators globally to combine their efforts in increasing course development resources.

For years research has focused on ways to allow remote access via standard communication. With the growth of Internet, one finds more and more devices such as coffee machines, telescopes, manipulators, and mobile robots connected to it. Open Access to a Mobile Robot on the Internet and Mobile Robots in Public Places are examples of remote control (Saucy and Mondana, 2000). A Feasibility Study for Internet Robots shows that controlling over Internet is becoming widespread.

The issue of process control has been effectively instrumental in raising the productivity in industrial automation for years. Remote control of processes is new era and only supported by a few companies, which are the giants of automation industry. Our proposal is also related to the control of industrial process with the exception of Internet. The idea proposed here comes with industrial automation for accessibility and controllability from anywhere.

The goal of this study is to discuss the details of the system that can control an industrial process over the Internet or modems with telephone lines. The remote processing has a bright future, especially in the companies with many branches where electronically controlled machines are deployed as a part of the workforce. While a computer connected to the automation devices can control and take data from processes at a production unit, the operator can process the data according to the rules and regulations and then issues commands necessary to control the system.

	Communication		Number of Process Connection	Modification
Commercial Methods	Server	Client	Server	Client
	Communicates with the client using web servers such as PWS™, IIS™	Using browsers, add-ons and software	Can connect only a process except Factory Suite	When an addition or changing is needed in process, they have to update/change server and client software
Proposed Method	No need to use web server	Only setup client software is enough	A client can connect more than one process	Changing server software is enough. When an error is occurred in server side, it could be corrected using mouse, keyboard, and screen utility at the client user.

Table 1 Characteristic differences in remote processing.

Technical Approach: For simulation purposes, Siemens S7 PLC process controller, shown in Figure 1, is used to control the processes of the equipments, and to bi-directionally communicate with the server. Additional tools are also developed to maintain the process control dialog between the server and the client.

Although some features of our method are similar to the methods available commercially in the industry, connecting to more than one process without using a web server is the unique feature of our approach. In other words, as far as monitoring and local control are concerned, the architecture of our work can be compared with the following software tools: WinCC from Siemens (Simens). For the remote control, the comparable software is PcANYWHERE from Symantec (Symatec).

Also our proposed monitoring and local control section will display a behavior similar to those mentioned above, the substantial difference will be in the remote control section as shown in Table 1. Therefore, we believe that the proposed work is an economic and useful solution for large and distributed companies, and workshops, which have to work together, even for institutions having health-critical or hazardous jobs.

The Architecture: A computer connected to the automation devices can control and get data from processes at a production unit. In the REPCO environment introduced this

process is done using PLC connection to a server, which can be controlled via telephone line or internet by a user who is far away from unit, as shown in Figure 2.

The REPCO environment can be technically decomposed into three parts:

I. Process

II. Server Program

III. Client Program.

The high level functional flow of the system is shown in Figure 3. The server software part is designed to handle three jobs. First it establishes a communication path with PLC. Second, the server maintains a smooth operation and records process related data. Lastly, it communicates with the client to obtain the process data and the server screen, according to which user commands are fed to the server. The communication between server and Client is handled through the use of Windows Socket, which supports TCP/IP. Note: Since HTTP and HTTPS1 protocols support only the connectionless communication, HTTP and related protocols are not considered at this stage.

Fig. 2 User process connections.

1. INTRODUCING THE PLC

1.1 What is PLC ?

A PLC (programmable logic controller) is a small industrial computer which originally replaced the necessary sequential relay circuits for machine control. The PLC works by looking at its inputs and depending upon their state, turning on/off its outputs. It contained a program which executed a loop, scanning the inputs and taking actions based on these inputs. The user enters a program, usually via software, that gives the desired results.

PLCs are used in many "real world" applications. If there is industry present, chances are good that there is a PLC present. If you are involved in machining, packaging, material handling, automated assembly or countless other industries you are probably already using them. If you are not, you are wasting money and time. Almost any application that needs some type of electrical control has a need for a PLC.

A PLC, basically consists of two elements:

I. the central processing unit
II. the input/output system

I. The Central Processing Unit

The central processing unit (CPU) is the part of a programmable controller that retrieves, decodes, stores, and processes information. It also executes the control program stored in the PLC's memory. In essence, the CPU is the "brains" of a programmable controller. It functions much the same way the CPU of a regular computer does, except that it uses special instructions and coding to perform its functions.

The CPU has three parts:

I. the processor
II. the memory system
III. the power supply

The processor is the section of the CPU that codes, decodes, and computes data. The memory system is the section of the CPU that stores both the control program and data from the equipment connected to the PLC. The power supply is the section that provides the PLC with the voltage and current it needs to operate.

II. The Input/Output System

The input/output (I/O) system is the section of a PLC to which all of the field devices are connected. If the CPU can be thought of as the brains of a PLC, then the I/O system can be thought of as the arms and legs. The I/O system is what actually physically carries out the control commands from the program stored in the PLC's memory.

The I/O system consists of two main parts:

I. the rack
II. I/O modules

The rack is an enclosure with slots in it that is connected to the CPU. I/O modules are devices with connection terminals to which the field devices are wired. Together, the rack and the I/O modules form the interface between the field devices and the PLC. When set up properly, Each I/O module is both securely wired to its corresponding field devices and securely installed in a slot in the rack. This creates the physical connection between the field equipment and the PLC. In some small PLC's, the rack and the I/O modules come prepackaged as one unit

1.2. Inputs And Outputs

All of the field devices connected to a PLC can be classified in one of two categories:

I. inputs
II. outputs

Inputs are devices that supply a signal/data to a PLC. Typical examples of inputs are push buttons, switches, and measurement de-vices. Basically an input device tells the PLC,

"Hey, something's happening out here…you need to check this out to see how it affects the control program.

Outputs are devices that await a signal/data from the PLC to perform their control functions. Lights, horns, motors, and valves are all good examples of output devices. These devices stay put, minding their own business, until the PLC says, "You need to turn on now" or "You'd better open up your valve a little more," etc.

There are two basic types of input and output devices:

I. discrete
II. analog

Discrete devices are inputs and outputs that have only two states: on and off. As a result, they send/receive simple signals to from a PLC. These signals consist of only 1's and 0's. A 1 means that the device is on and a 0 means that the device is off.

Analog devices are inputs and outputs that can have an infinite number of states. These devices can not only be on and off, but they can also be barely on, almost totally on, not quite off, etc. These devices send receive complex signals to from a PLC. Their communications consist of a variety of signals, not just 1's and 0's. Because different input and output devices send different kinds of signals, they sometimes have a hard time communicating with the PLC. While PLC's are powerful devices, they can't always speak the "language" of every device connected to them. That's where the I/O modules we talked about earlier come in. The modules act as "translators" between the field devices and the PLC. They ensure that the PLC and the field devices all get the information they need in a language that they can understand.

1.3. Control Programs

We talked a little bit earlier about the control program. The control program is a software program in the PLC's memory. It's what puts the control in a programmable controller. The user or the system designer is usually the one who develops the control program. The

control program is made up of things called instructions. Instructions are, in essence, little computer codes that make the inputs and outputs do what you want in order to get the result you need. There are all different kinds of instructions and they can make a PLC do just about anything (add and subtract data, time and count events, compare information, etc.). All you have to do is program the instructions in the proper order and make sure that they are telling the right devices what to do and voila!...you have a PLC-controlled system. And remember, changing the system is a snap. If you want the system to act differently, just change the instructions in the control program. Different PLC's offer different kinds of instructions. That's part of what makes each type of PLC unique.

However, all PLCs use two basic types of instructions:

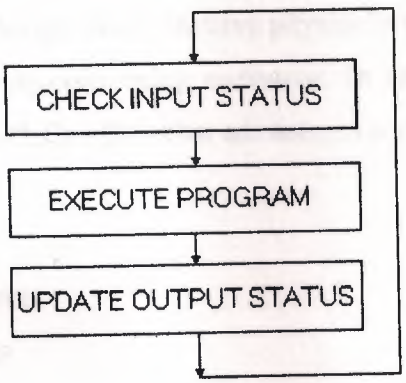
I. contacts
II. coils

Contacts are instructions that refer to the input conditions to the control program that is, to the information supplied by the input field devices. Each contact in the control program monitors a certain field device. The contact waits for the input to do something in particular (e.g., turn on, turn off, etc. this all depends on what type of contact it is). Then, the contact tells the PLC's control program, "The input device just did what it's supposed to do. You'd better check to see if this is supposed to affect any of the output devices."

Coils are instructions that refer to the outputs of the control program that is, to what each particular output device is supposed to do in the system. Like a contact, each coil also monitors a certain field device. However, unlike a contact, which monitors the field device and then tells the PLC what to do, a coil monitors the PLC control program and then tells the field device what to do. It tells the output device, "Hey, the PLC just told me that the switch turned on. That means that you're supposed to turn on now. So let's go!" In PLC talk, this three-step process of monitoring the inputs, executing the PLC control program, and changing the status of the

1.4. How PLC Work?

A PLC works by continually scanning a program. We can think of this scan cycle as consisting of 3 important steps. There are typically more than 3 but we can focus on the important parts and not worry about the others. Typically the others are checking the system and updating the current internal counter and timer values.



Step 1-CHECK INPUT STATUS-First the PLC takes a look at each input to determine if it is on or off. In other words, is the sensor connected to the first input on? How about the second input? How about the third... It records this data into its memory to be used during the next step.

Step 2-EXECUTE PROGRAM-Next the PLC executes your program one instruction at a time. Maybe your program said that if the first input was on then it should turn on the first output. Since it already knows which inputs are on/off from the previous step it will be able to decide whether the first output should be turned on based on the state of the first input. It will store the execution results for use later during the next step.

Step 3-UPDATE OUTPUT STATUS-Finally the PLC updates the status of the outputs. It updates the outputs based on which inputs were on during the first step and the results of executing your program during the second step.

1.5. WHY USE PLCS?

The software advantage provided by programmable controllers is tremendous. In fact, it is one of the most important features of PLCs. Software makes changes in the control system easy and cheap. If you want a device in a PLC system to behave differently or to control a different process element, all you have to do is change the control program. In a traditional system, making this type of change would involve physically changing the wiring between the devices, a costly and time-consuming endeavor. In addition to the programming flexibility we just mentioned, PLCs offer other advantages over traditional control systems. These advantages include:

- high reliability
- small space requirements
- computing capabilities
- reduced costs
- ability to withstand harsh environments
- expandability

1.6. Capabilities of the S7-212 CPUs

The S7-200 family includes a wide variety of CPUs. This variety provides a range of features to aid in designing a cost-effective automation solution. Table 2 provides a summary of the major features of each S7-200 CPU.

Table 2

CPU 212 DC power supply, DC inputs, DC outputs

Order Number: 6ES7 212–1AA01–0XB0

General Features	
Physical size (L x W x D)	160 x 80 x 62 mm (6.3 x 3.15 x 2.44 in.)
Weight	0.3 kg (0.7 lbs.)
Power dissipation	5 W at 1.75 A load
User program size/storage	512 words/EEPROM
User data size/storage Data retention	512 words/RAM 50 hr typical (8 hr minimum at 40° C)
Local I/O[1]	8 inputs/6 outputs
Maximum number of expansion modules	2
Digital I/O supported	64 inputs/64 outputs
Analog I/O supported	16 inputs/16 outputs
Boolean execution speed	1.2 µs/instruction
Internal memory bits	128
Timers	64 timers
Counters	64 counters
High-speed counters	1 software (2 KHz max.)
Analog adjustments	1
Standards compliance	UL 508 CSA C22.2 142 FM Class I, Division 2 VDE 0160 compliant CE compliant

Output Points		
Output type	Sourcing transistor	
Voltage range	20.4 VDC to 28.8 VDC	
Maximum load current	0 to 40° C	55°C[2]
Per single point	0.75 A	0.50 A
Per 2 adjacent points	1.00 A	0.75 A
All points total	2.25 A	1.75 A
Inductive load clamping	(per common)	
Single pulse	2A L/R = 10 ms 1A L/R = 100 ms	
Repetitive	1 W energy dissipation (1/2 LI^2 x switch rate < 1 W)	
Leakage current	100 µA	

Output Points (continued)	
Switching delay	25 µs ON, 120 µs OFF
Surge current	4 A, 100 ms
Voltage drop	1.8 V maximum at maximum current
Optical isolation	500 VAC, 1 min
Short circuit protection	None

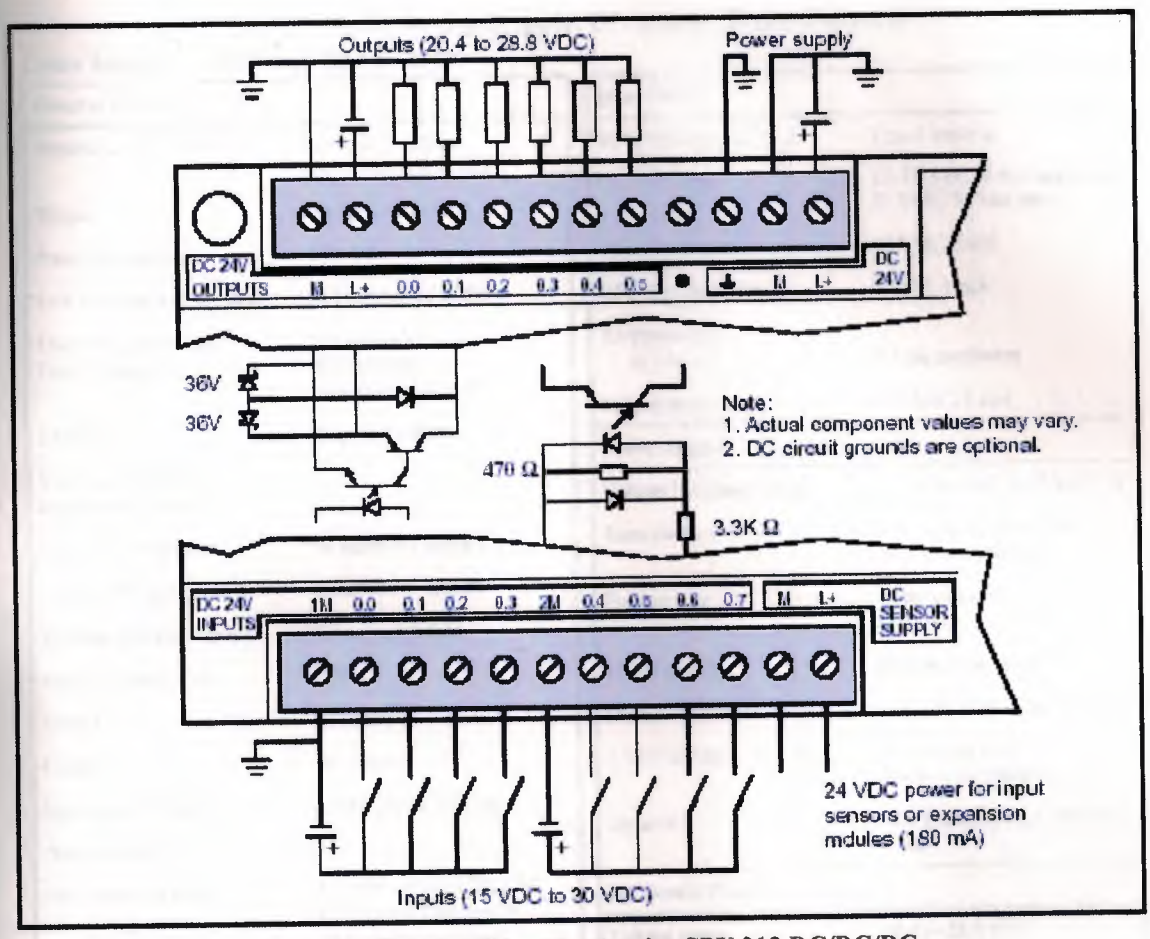
Input Points	
Input type (IEC 1131–2)	Type I sinking
ON state range	15–30 VDC, 4 mA minimum 35 VDC, 500 ms surge
ON state nominal	24 VDC, 7 mA
OFF state maximum	5 VDC, 1 mA
Response time I0.0 to I0.7	0.3 ms maximum
Optical isolation	500 VAC, 1 min

Power Supply	
Voltage range	20.4 to 28.8 VDC
Input current	60 mA typical, CPU only 500 mA maximum load
UL/CSA rating	50 VA
Holdup time	10 ms minimum from 24 VDC
Inrush current	10 A peak at 28.8 VDC
Fusing (non-replaceable)	1 A, 125 V, slow blow
5 VDC current	260 mA for CPU 340 mA for expansion I/O
Isolated	No

DC Sensor Supply	
Voltage range	16.4 to 28.8 VDC
Ripple/noise (< 10MHz)	Same as supplied voltage
24 VDC available current Short-circuit current limit	180 mA < 600 mA
Isolated	No

1 The CPU reserves 8 process-image input and 8 process-image output image register points for local I/O.

2 Linear derate 40 to 55° C. Vertical mount derate 10° C.



Connector Terminal Identification for CPU 212 DC/DC/DC

CPU 212 AC Power Supply, DC Inputs, Relay Outputs

Order Number: 6ES7 212–1BA01–0XB0

General Features	
Physical size (L x W x D)	160 x 80 x 62 mm (6.3 x 3.15 x 2.44 in.)
Weight	0.4 kg (0.9 lbs.)
Power dissipation	6 W
User program size/storage	512 words/EEPROM
User data size/storage Data retention	512 words/RAM 50 hr typical (8 hr minimum at 40° C)
Local I/O[1]	8 inputs/6 outputs
Maximum number of expansion modules	2
Digital I/O supported	64 inputs/64 outputs
Analog I/O supported	16 inputs/16 outputs
Boolean execution speed	1.2 µs/instruction
Internal memory bits	128
Timers	64 timers
Counters	64 counters
High-speed counters	1 software (2 KHz max.)
Analog adjustments	1
Standards compliance	UL 508 CSA C22.2 142 FM Class I, Division 2 VDE 0160 compliant CE compliant

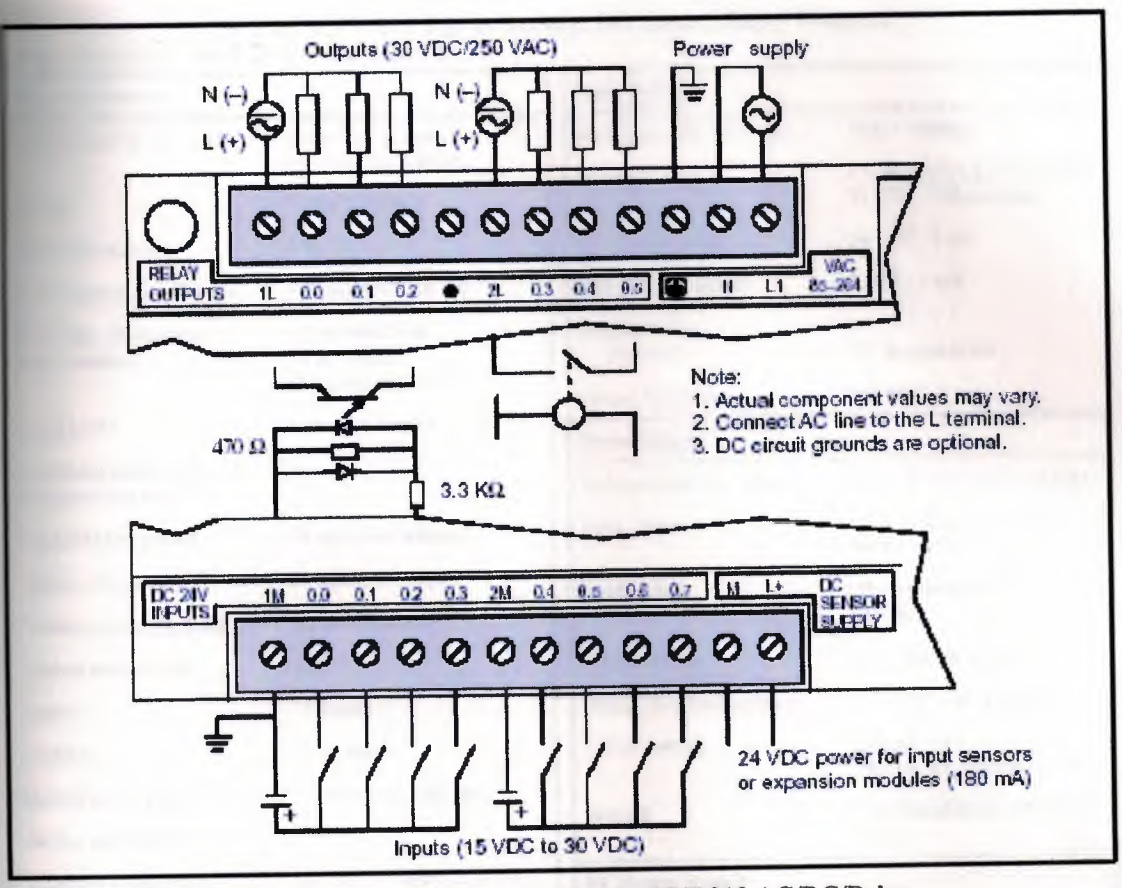
Output Points	
Output type	Relay, dry contact
Voltage range	5 to 30 VDC/250 VAC
Maximum load current	2 A/point, 6 A/common
Overcurrent surge	7 A with contacts closed
Isolation resistance	100 MΩ minimum (new)
Switching delay	10 ms maximum
Lifetime	10,000,000 mechanical 100,000 with rated load
Contact resistance	200 mΩ maximum (new)
Isolation coil to contact contact to contact (between open contacts)	 1500 VAC, 1 min 750 VAC, 1 min
Short circuit protection	None

Input Points	
Input type (IEC 1131–2)	Type 1 sinking
ON state range	15–30 VDC, 4 mA minimum 35 VDC, 500 ms surge
ON state nominal	24 VDC, 7 mA
OFF state maximum	5 VDC, 1 mA
Response time I0.0 to I0.7	 0.3 ms maximum
Optical isolation	500 VAC, 1 min

Power Supply	
Voltage/frequency range	85 to 264 VAC at 47 to 63 Hz
Input current	4 VA typical, CPU only 50 VA maximum load
Holdup time	20 ms minimum from 110 VAC
Inrush current	20 A peak at 264 VAC
Fusing (non-replaceable)	2 A, 250 V, slow blow
5 VDC current	260 mA for CPU 340 mA for expansion I/O
Isolated	Yes, Transformer, 1500 VAC, 1 min

DC Sensor Supply	
Voltage range	20.4 to 28.8 VDC
Ripple/noise (< 10MHz)	1 V peak-to-peak maximum
24 VDC available current Short circuit current limit	180 mA < 600 mA
Isolated	No

1 The CPU reserves 8 process-image input and 8 process-image output image register points for local I/O.



Connector Terminal Identification for CPU 212 AC/DC/Relay

24 VAC CPU 212 Power Supply, DC Inputs, Relay Outputs

Order Number: 6ES7 212–1FA01–0XB0

General Features	
Physical size (L x W x D)	160 x 80 x 62 mm (6.3 x 3.15 x 2.44 in.)
Weight	0.4 kg (0.9 lbs.)
Power dissipation	6 W
User program size/storage	512 words/EEPROM
User data size/storage Data retention	512 words/RAM 50 hr typical (8 hr minimum at 40° C)
Local I/O[1]	8 inputs/6 outputs
Maximum number of expansion modules	2
Digital I/O supported	64 inputs/64 outputs
Analog I/O supported	16 inputs/16 outputs
Boolean execution speed	1.2 µs/instruction
Internal memory bits	128
Timers	64 timers
Counters	64 counters
High-speed counters	1 software (2 KHz max.)
Analog adjustments	1
Standards compliance	UL 508 CSA C22.2 142 FM Class I, Division 2 VDE 0160 compliant CE compliant

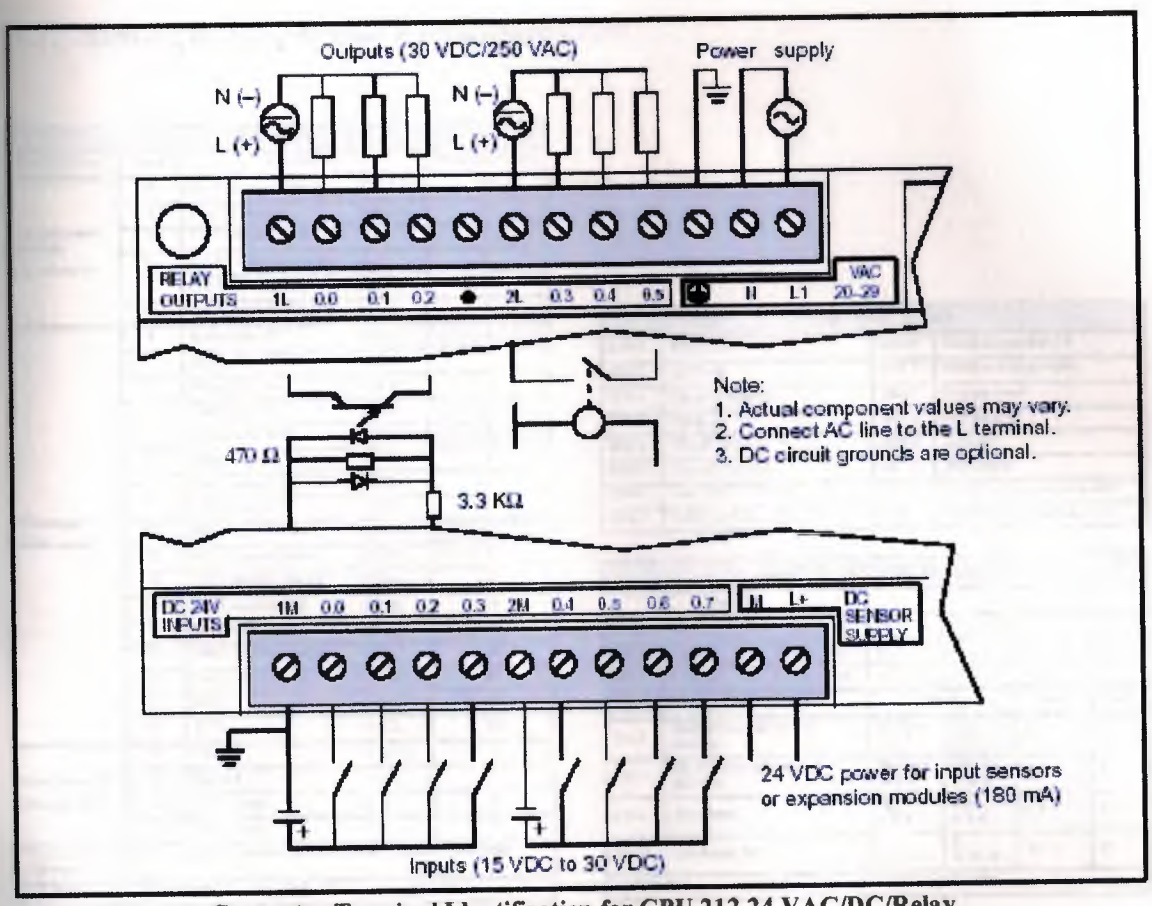
Output Points	
Output type	Relay, dry contact
Voltage range	5 to 30 VDC/250 VAC
Maximum load current	2 A /point, 6 A /common
Overcurrent surge	7 A with contacts closed
Isolation resistance	100 MΩ minimum (new)
Switching delay	10 ms maximum
Lifetime	10,000,000 mechanical 100,000 with rated load
Contact resistance	200 mΩ maximum (new)
Isolation coil to contact contact to contact (between open contacts)	 1500 VAC, 1 min 750 VAC, 1 min
Short circuit protection	None

Input Points	
Input type (IEC 1131–2)	Type 1 sinking
ON state range	15–30 VDC, 4 mA minimum 35 VDC, 500 ms surge
ON state nominal	24 VDC, 7 mA
OFF state maximum	5 VDC, 1 mA
Response time 10.0 to 10.7	 0.3 ms maximum
Optical isolation	500 VAC, 1 min

Power Supply	
Voltage/frequency range	20 to 29 VAC at 47 to 63 Hz
Input current	4 VA typical, CPU only 50 VA maximum load
Holdup time	20 ms minimum from 24 VAC
Inrush current	20 A peak at 29 VAC
Fusing (non-replaceable)	2 A, 250 V, slow blow
5 VDC current	260 mA for CPU 340 mA for expansion I/O
Isolated	Yes, Transformer, 500 VAC, 1 min

DC Sensor Supply	
Voltage range	20.4 to 28.8 VDC
Ripple/noise (< 10MHz)	1 V peak-to-peak maximum
24 VDC available current Short circuit current limit	180 mA < 600 mA
Isolated	No

1 The CPU reserves 8 process-image input and 8 process-image output image register points for local I/O.



Connector Terminal Identification for CPU 212 24 VAC/DC/Relay

1.7. SIMATIC S7-200 Quick Reference Card

Interrupts			
Priority Group	Event	Description	Priority in Group
Communication interrupts Highest priority	8	Port 0: Receive character	0
	9	Port 0: Transmit	0
	23	Port 0: Receive message	0
	24	Port 1: Receive message	1
	25	Port 1: Receive character	1
	26	Port 1: Transmit complete	1
I/O interrupts Middle priority	0	Rising edge, I0.0*	0
	2	Rising edge, I0.1	1
	4	Rising edge, I0.2	2
	6	Rising edge, I0.3	3
	1	Falling edge, I0.0*	4
	3	Falling edge, I0.1	5
	5	Falling edge, I0.2	6
	7	Falling edge, I0.3	7
	12	HSC0 = preset value*	0
	13	HSC1 = preset value	9
	14	HSC1 direction change	9
	15	HSC1 external reset	10
	16	HSC2 = preset value	11
	17	HSC2 direction change	12
	18	HSC2 external reset	13
	19	PLS0	14
	20	PLS1	15
Timed interrupts Lowest priority	10	Timed 0	0
	11	Timed 1	1
	21	T32 = preset	2
	22	T96 = preset	3
* If event 12 is attached to an interrupt, then event 0 and event 1 cannot be attached to interrupts.			
CPU 212 CPU 214 CPU 215 CPU 216			

Special Memory Bits			
SM0.0	Always On	SM1.0	Result of operation = 0
SM0.1	First Scan	SM1.1	Overflow or illegal value
SM0.2	Retentive data lost	SM1.2	Negative result
SM0.3	Power up	SM1.3	Division by 0
SM0.4	30 s off / 30 s on	SM1.4	Table full
SM0.5	0.5 s off / 0.5 s on	SM1.5	Table empty
SM0.6	Off 1 scan / on 1 scan	SM1.6	BCD to binary conversion error
SM0.7	Switch in RUN position	SM1.7	ASCII to hex conversion error

High-Speed Counter Modes					
Counter		Inputs			
HSC0	Maximum 2 kHz	I0.0			
HSC1	7 kHz 20 kHz	I0.6	I0.7	I1.0	I1.1
HSC2	7 kHz 20 kHz	I1.2	I1.3	I1.4	I1.5
Mode	Description	Clock		Reset	Start
0 to 2	Single phase with internal direction	Up/Down: 0, 1, 2		1, 2	2
3 to 5	Single phase with external direction	Up/Down: 3, 4, 5	Direction: 3, 4, 5	4, 5	5
6 to 8	Two phase	Up: 6, 7, 8	Down: 6, 7, 8	7, 8	8
9 to 11	Quadrature A/B	A: 9, 10, 11	B: 9, 10, 11	10, 11	11
CPU 212 CPU 214 CPU 215 CPU 216					

Description	Range Limit				Accessible as...			
	212	214	215	216	Bit	Byte	Word	DWord
User Program Size	512 W	2048 W	4096 W	4096 W				
User Data Size	512 W	2048 W	2560 W	2560 W				
Variable memory	0.1023	0.4095	0.5119	0.5119	V*x.y*	VB*x*	VW*x*	VD*x*
Input Image Register	0.7	0.7	0.7	0.7	I*x.y*	IB*x*	IW*x*	ID*x*
Output Image Register	0.7	0.7	0.7	0.7	Q*x.y*	QB*x*	QW*x*	QD*x*
Analog Inputs	0.30	0.30	0.30	0.30			AIW*x*	
Analog Outputs	0.30	0.30	0.30	0.30			AQW*x*	
Bit Memory	0.15	0.31	0.31	0.31	M*x.y*	MB*x*	MW*x*	MD*x*
Special Memory	0.45	0.85	0.194	0.194	SM*x.y*	SMB*x*	SMW*x*	SMD*x*
Retentive Timers 1 ms	0	0.64	0.64	0.64	T*x*		T*x*	
Retentive Timers 10 ms	1-4	1-4, 65-68	1-4, 65-68	1-4, 65-68	T*x*		T*x*	
Retentive Timers 100 ms	5-31	5-31, 69-95	5-31, 69-95	5-31, 69-95	T*x*		T*x*	
On/Delay Timers 1 ms	32	32, 96	32, 96	32, 96	T*x*		T*x*	
On/Delay Timers 10 ms	33-36	33-36, 97-100	33-36, 97-100	33-36, 97-100	T*x*		T*x*	
On/Delay Timers 100 ms	37-63	37-63, 101-127	37-63, 101-255	37-63, 101-255	T*x*		T*x*	
Counters	0-63	0-127	0-255	0-255	C*x*		C*x*	
High Speed Counter	0	0-2	0-2	0-2				HC*x*
Accumulators	0-3	0-3	0-3	0-3		AC*x*	AC*x*	AC*x*
Sequence Control Relay (SCR)	0-7	0-15	0-31	0-31	S*x.y*	SB*x*	SW*x*	SD*x*
Jump/Labels	0-63	0-255	0-255	0-255				
Call Subroutine	0-15	0-63	0-63	0-63				
Interrupt Routines	0-31	0-127	0-127	0-127				
Interrupt Events	0,1,2-10, 12	0-23	0-23	0-23				
PID Loops	N/A	N/A	0-7	0-7				
Ports	Port 0	Port 0	Port 0 , DP Port	Port 0, Port 1				

Boolean Instructions

LD	N	Load
LDI	N	Load Immediate
LDN	N	Load Not
LDNI	N	Load Not Immediate
A	N	AND
AI	N	AND Immediate
AN	N	AND Not
ANI	N	AND Not Immediate
O	N	OR
OI	N	OR Immediate
ON	N	OR Not
ONI	N	OR Not Immediate
LDB=	N1, N2	Load result of Byte Compare N1 (=, >=, or <=) N2
LDB>=	N1, N2	
LDB<=	N1, N2	
AB=	N1, N2	AND result of Byte Compare N1 (=, >=, or <=) N2
AB>=	N1, N2	
AB<=	N1, N2	
OB=	N1, N2	OR result of Byte Compare N1 (=, >=, or <=) N2
OB>=	N1, N2	
OB<=	N1, N2	
LDW=	N1, N2	Load result of Word Compare N1 (=, >=, or <=) N2
LDW>=	N1, N2	
LDW<=	N1, N2	
AW=	N1, N2	AND result of Word Compare N1 (=, >=, or <=) N2
AW>=	N1, N2	
AW<=	N1, N2	
OW=	N1, N2	OR result of Word Compare N1 (=, >=, or <=) N2
OW>=	N1, N2	
OW<=	N1, N2	
LDD=	N1, N2	Load result of DWord Compare N1 (=, >=, or <=) N2
LDD>=	N1, N2	
LDD<=	N1, N2	
AD=	N1, N2	AND result of DWord Compare N1 (=, >=, or <=) N2
AD>=	N1, N2	
AD<=	N1, N2	
OD=	N1, N2	OR result of DWord Compare N1 (=, >=, or <=) N2
OD>=	N1, N2	
OD<=	N1, N2	
LDR=	N1, N2 [1]	Load result of Real Compare N1 (=, >=, or <=) N2
LDR>=	N1, N2 [1]	
LDR<=	N1, N2 [1]	
AR=	N1, N2 [1]	AND result of Real Compare N1 (=, >=, or <=) N2
AR>=	N1, N2 [1]	
AR<=	N1, N2 [1]	
OR=	N1, N2 [1]	OR result of Real Compare N1 (=, >=, or <=) N2
OR>=	N1, N2 [1]	
OR<=	N1, N2 [1]	
NOT		Stack Negation
EU		Detection of Rising Edge
ED		Detection of Falling Edge
=	N	Assign Value
=I	N	Assign Value Immediate
S	S_BIT, N	Set bit Range
R	S_BIT, N	Reset bit Range
SI	S_BIT, N	Set bit Range Immediate
RI	S_BIT, N	Reset bit Range Immediate

Math, Increment, and Decrement Instructions

+I	IN1, OUT	Add Integer, DWord or Real IN1+OUT=OUT
+D	IN1, OUT	
+R	IN1, OUT [1]	
-I	IN1, OUT	Subtract Integer, DWord, or Real OUT-IN1=OUT
-D	IN1, OUT	
-R	IN1, OUT [1]	
MUL	IN1, OUT	Multiply Integer or Real IN1 * OUT = OUT
*R	IN1, OUT [1]	
DIV	IN1, OUT	Divide Integer or Real OUT / IN1 = OUT
/R	IN1, OUT [1]	
SQRT	IN, OUT [1]	Square Root
INCB	OUT [2]	Increment Byte, Word or DWord
INCW	OUT	
INCD	OUT	
DECB	OUT [2]	Decrement Byte, Word, or DWord
DECW	OUT	
DECD	OUT	
PID	Table, Loop [1]	PID Loop

Timer and Counter Instructions

TON	Txxx, PT	On Delay Timer
TONR	Txxx, PT	Retentive On Delay Timer
CTU	Cxxx, PV	Count Up
CTUD	Cxxx, PV	Count Up/Down

Real Time Clock Instructions

TODR	T [1]	Read Time of Day clock
TODW	T [1]	Write Time of Day clock

Program Control Instructions

END		Conditional End of Program
MEND		Main Program End of Program
STOP		Transition to STOP Mode
WDR		Watchdog Reset (300 ms)
JMP	N	Jump to defined Label
LBL	N	Define a Label to Jump to
CALL	N	Call a Subroutine
SBR	N	Define a Subroutine to be Called
CRET		Conditional Return from SBR
RET		Unconditional Return from SBR
FOR	Index, Initial, Final [1]	For/Next Loop
NEXT	[1]	
LSCR	N	Load, Transition, and End Sequence Control Relay Segment
SCRT	N	
SCRE		

Move, Shift, Rotate, and Fill Instructions

MOVB	IN, OUT	Move Byte, Word, DWord, Real
MOVW	IN, OUT	
MOVD	IN, OUT	
MOVR	IN, OUT [1]	
BMB	IN, OUT, N	Block Move Byte, Word, DWord
BMW	IN, OUT, N	
BMD	IN, OUT, N [2]	
SWAP	IN	Swap Bytes
SHRB	Data, S_bit, N	Shift Register Bit
SRB	OUT, N [2]	Shift Right Byte, Word, DWord
SRW	OUT, N	
SRD	OUT, N	
SLB	OUT, N [2]	Shift Left Byte, Word, DWord
SLW	OUT, N	
SLD	OUT, N	
RRB	OUT, N [2]	Rotate Right Byte, Word, DWord
RRW	OUT, N	
RRD	OUT, N	
RLB	OUT, N [2]	Rotate Left Byte, Word, DWord
RLW	OUT, N	
RLD	OUT, N	
FILL	IN, OUT, N	Fill memory space with pattern

Logic Operations

ALD		And for combinations
OLD		Or for combinations
LPS		Logic Push (stack control)
LRD		Logic Read (stack control)
LPP		Logic Pop (stack control)
ANDB	IN1, OUT [2]	Logical And of Byte, Word, and DWord
ANDW	IN1, OUT	
ANDD	IN1, OUT	
ORB	IN1, OUT [2]	Logical Or of Byte, Word, and DWord
ORW	IN1, OUT	
ORD	IN1, OUT	
XORB	IN1, OUT [2]	Logical XOr of Byte, Word, and DWord
XORW	IN1, OUT	
XORD	IN1, OUT	
INVB	OUT [2]	Invert Byte, Word and DWord (1's complement)
INVW	OUT	
INVD	OUT	

Table, Find, and Conversion Instructions

ATT	Data, Table [1]	Add data to table
LIFO	Table, Data [1]	Get data from table
FIFO	Table, Data [1]	
FND=	Scr, Pattn, Indx [1]	Find data value in table that matches comparison
FND<>	Scr, Pattn, Indx [1]	
FND<	Scr, Pattn, Indx [1]	
FND>	Scr, Pattn, Indx [1]	
BCDI	OUT	Convert BCD to Integer
IBCD	OUT	Convert Integer to BCD
DTR	IN, OUT [1]	Convert DWord to Real
TRUNC	IN, OUT [1]	Convert Real to DWord
ATH	IN, OUT, LEN	Convert ASCII to HEX
HTA	IN, OUT, LEN	Convert HEX to ASCII
DECO	IN, OUT	Decode
ENCO	IN, OUT	Encode
SEG	IN, OUT	Generate 7-segment pattern

Interrupt

INT	N	Beginning of Interrupt routine
CRETI		Conditional Return from Interrupt
RETI		Return from Interrupt
ENI		Enable Interrupts
DISI		Disable Interrupts
ATCH	INT, EVENT	Attach Interrupt routine to event
DTCH	EVENT	Detach event

Communication

XMT	TABLE, PORT	Freeport transmission
RCV	TABLE, PORT [2]	Freeport receive message
NETR	TABLE,PORT [2]	Network Read
NETW	TABLE,PORT [2]	Network Write

High Speed Instructions

HDEF	HSC, Mode	Define High Speed Counter mode
HSC	N	Activate High Speed Counter
PLS	X [2]	Pulse Output

Instructions are valid for the individual S7-200 PLCs as marked according to the following key:

[1] 214, 215, and 216 only
[2] 215 and 216 only

If not marked, the instructions are valid for all S7-200 PLCs.

2. PROCESS

2.1. Introduction

There is a need for a program for the PLC controller. STEP-7 Microwin PLC is a program editor prepared for Siemens PLC. The same program editor will be used for my research purposes. The unit that will be controlled by the Siemens S-7 CPU 212 PLC is a dry packaging unit. This can be seen in the fig.3. The flowchart for the PLC program can be seen fig.4.

Fig.3 dry packaging unit

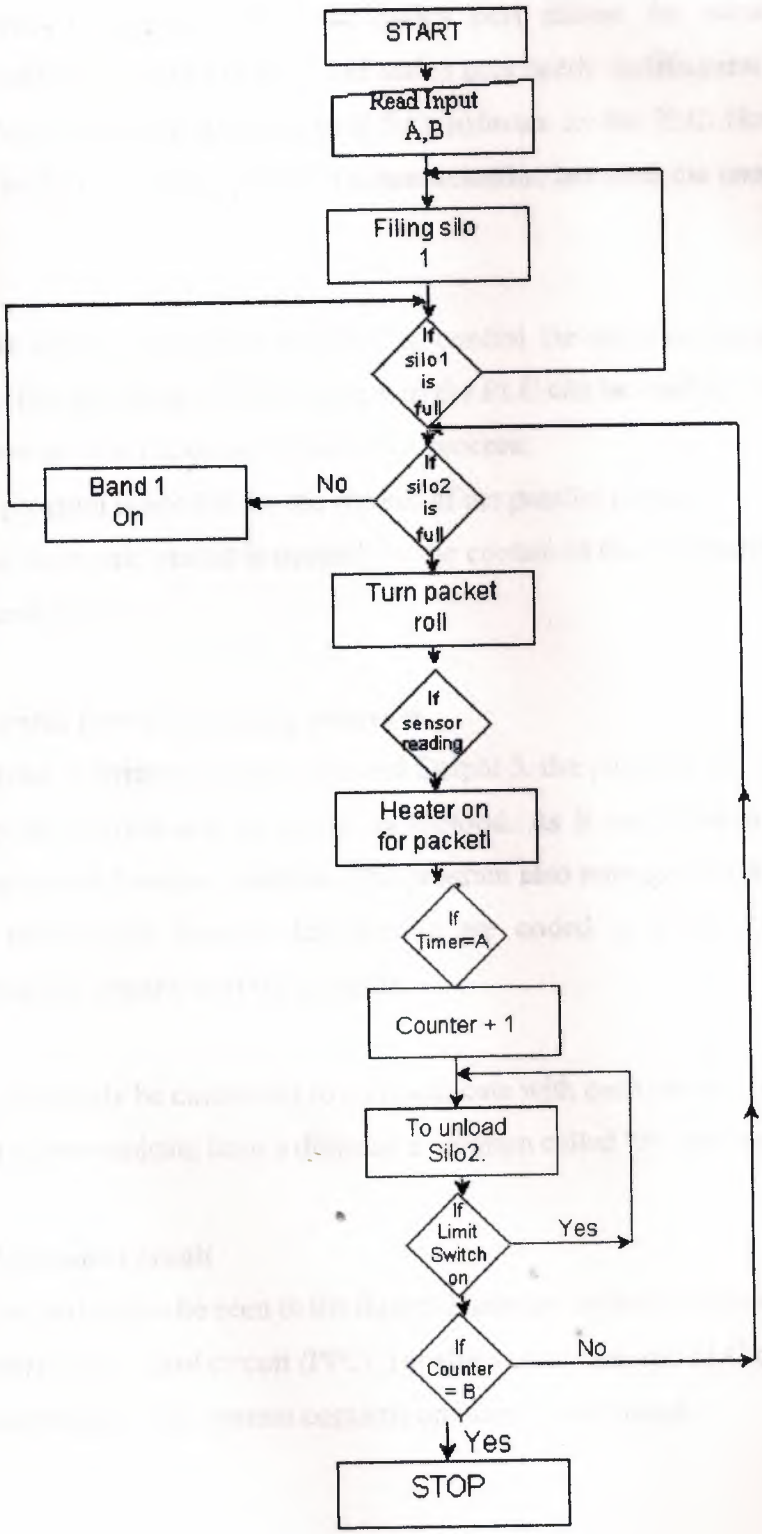


Fig. 4 PLC program flowchart

The program of the PLC and the series port allows for outside communication. Communication between the PLC and series port needs codification and de-codification systems. Apart from this there is a need for a software for the PLC. However the aim of the project is to find a simpler system of communication between the computer and the series port.

One of the aims is to prepare the PLC to control the input contacts and run the on/off functions. The incoming output contacts of the PLC can be read by the help of the parallel ports. There are two important steps in this process:

I. A program is needed for the control of the parallel ports.
II. An electronic circuit is needed for the control of the information received from the parallel ports.

2.2. Parallel Port Controlling program

This program is written by using Borland Delphi 5. the program was designed as shown to give easy recognition and be easily understood. As it can be seen from the figure this system can make 8 output controls. The program also manages the 4 inputs as well as the external information income. Information are coded is binaries. Fault warnings are displayed at the upper bar of the program.

Two PC can easily be connected to communicate with each other by using one as the host. In order to communicate from a distance a program called "PC anywhere" can be used.

2.3. Electronic Circuit

The circuit design can be seen in the figure. There are further explanations about the circuit. The parallel port control circuit (PPCC) enables computer and PLC connection without an electric connection. The system contains optocopler and relays.

2.4. Server Environment

Server environment encompasses a PC-Anywhere program which can initiate communication by warning the PC via a modem which is programmed earlier. When the connection is made one of the devises act as server and the other as client.

I. Server computer is the computer that manage the client computer from a distance.
II. Client computer is the computer that had been targeted by the server computer.

Two computers can be connected in the following fashion:

I. Direct
II. Via a Modem
III. Network
IV. Internet

I. Direct

Two computers can be connected directly by a UTP cable. An ethernet card is needed for the connection. Such a system can be adapted for the control of packaging from within the building.

II. Modem

A telephone line can be used for a modem connection. By using PC anywhere program it is possible to communicate with the client computer from anywhere. All we need is a computer, a telephone line and a modem. Distance is no problem with this system.

III. Network

some large organization can setup their own network systems. PC anywhere can be used for these systems as well.

IV. Internet

IP number of the client computer needs to be known to establish a connection via an internet provider. PC anywhere program can be connected to the internet and by registering an IP number the connection can be made.

2.5. Communication and Client Environment

Client environment consist of a computer, PPCD, pre-installed PC anywhere modem and the telephone line. Apart from these the program I wrote with the Delphi 5 is needed to control the PPCD. Since the Delphi Assembly language is easily acceptable it is much easier to make the parallel port control. Step-7 microwin program is needed if there is a desire to introduce other programs in future. Program editor is providing clear view of the steps of actions. Client can communicate with more than one servers shown fig.5. If desired the server can communicate with more than one client shown fig.6.

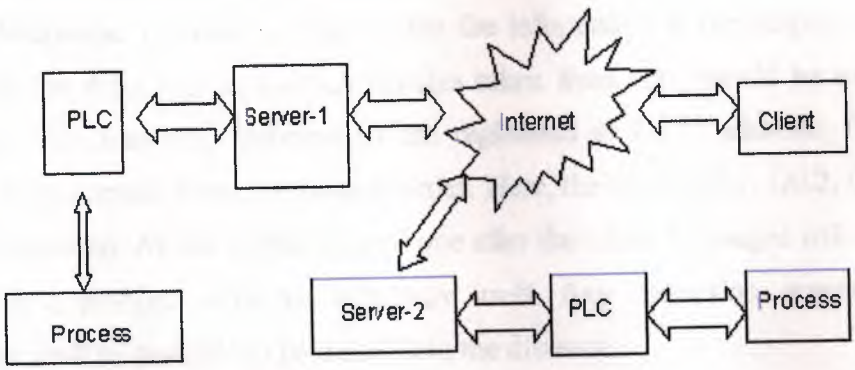


Fig.5 Client can view more than one process.

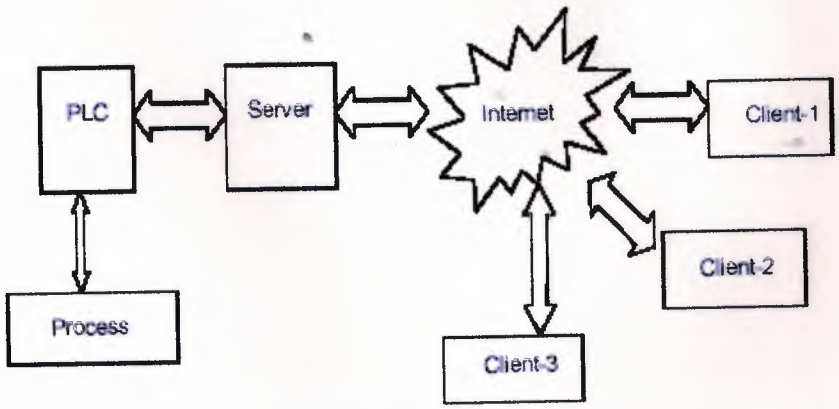


Fig.6 Process can be viewed by more than one client.

2.6. Connection for Chat

Need may arise to send and receive messages during the communication. A part is designed with Delphi which can be used as a chat shown fig.7.

Fig.7 Message line for chat

2.7. Process and Alarm

The system provides information about the progress of the work. System can be improved with extra functions. For example, it can be arranged that the system can send sms messages to our mobile telephones in case of emergencies. The program can be improved to include the capabilities of sms communication. Prewritten statements can then be sent to the mobile telephone. In order to achieve this the information at the outputs need to be transferred to the 4 bit inputs, and the binaries taken from here should be evaluated as hexadecimal. The incoming information are registered at H379 address. The control program of the computer then put these in order. Here, the Q0.0, Q0.1, Q0.2, Q0.3, Q0.4, output are controlled. All the output is send one after the other. Messages will reach us in case there is a problem with the computer itself. Any correction, amendment and configuration shall be possible to be made from the distance.

3. PRODUCTION UNIT

3.1. Introduction to the Production Unit

Dry packaging unit consist of several units as it can be seen in the figure 8. The system is controlled by relays. This is a system that requires extensive maintenance work. The system can be modified to be controlled by PLC system.

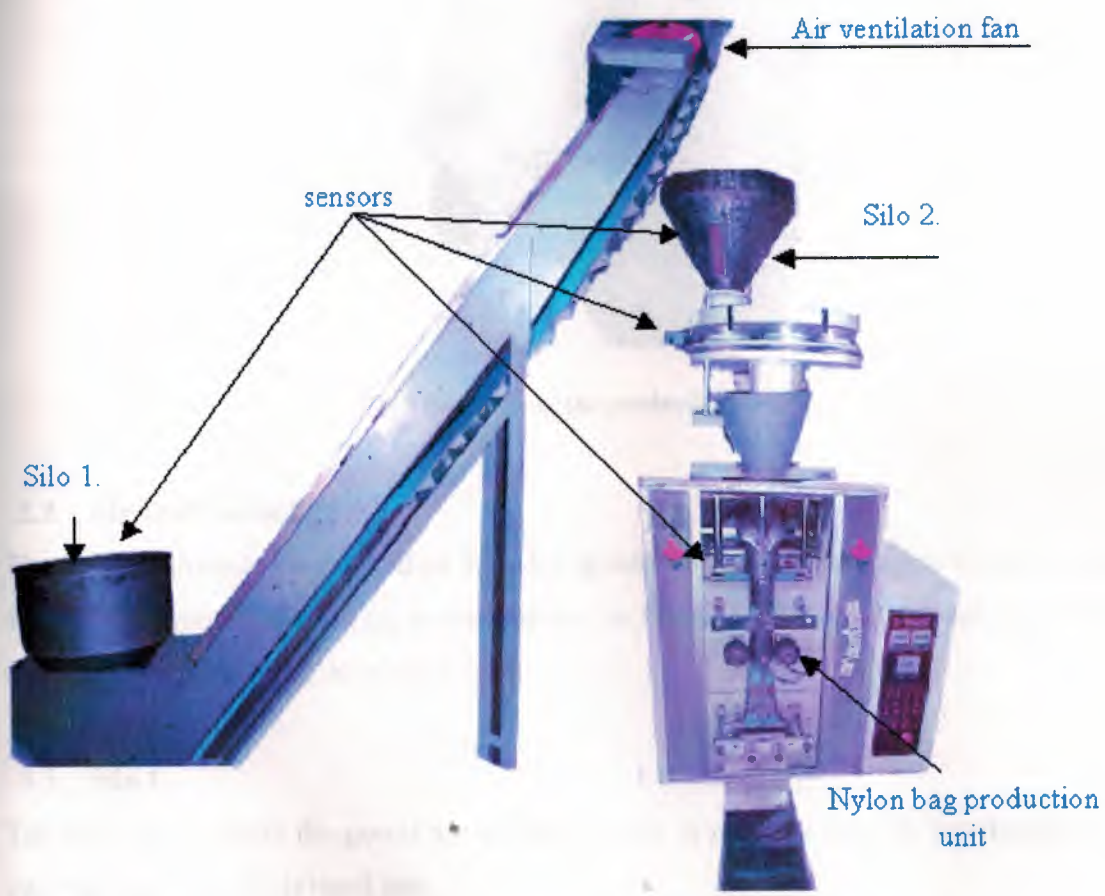


Fig.8 Dry packaging unit

Fig.9 Back side of the productions unit

3.2. Air ventilation fan

The fun functions to clean the dust from the goods that will be packaged. It consist a 3Hp 415 V 50Hz motor. According to regulations in TRNC a delta starter switch needs to be used for motors between 3Hp and 5.5HP.

3.3. Silo 1.

The first silo is where the goods are initially stored. A pallet is used for pouring the good into the silo. The silo is hand fed.

3.4. Silo 2.

After the first silo is full the second is started to be filled. There are switches that triggers the pallet at each silo. As it can be seen at the photos the two silos have height adjustments. The heights are arranged by a push button. There are also adjustments for the weight of the goods that will be packaged. Silo 1. and silo 2. are also facilitated with limit switch which control the discharge of the silos.

3.5. Nylon bag production unit

An electric and a mechanical system is to run nylon film and weld it to become a bag. The nylon tube has black markings, which are recognized by the sensors and these give the orders for action to seal the nylon. Figure 1.1

Fig.1.1 Nylon bag production unit

3.6. Sensors

The system contains 2 level switch and 1 limit switch. Role of these were just explained above. The sensors are made from photo-led and transistors. Shown fig.1.2a and 1.2b

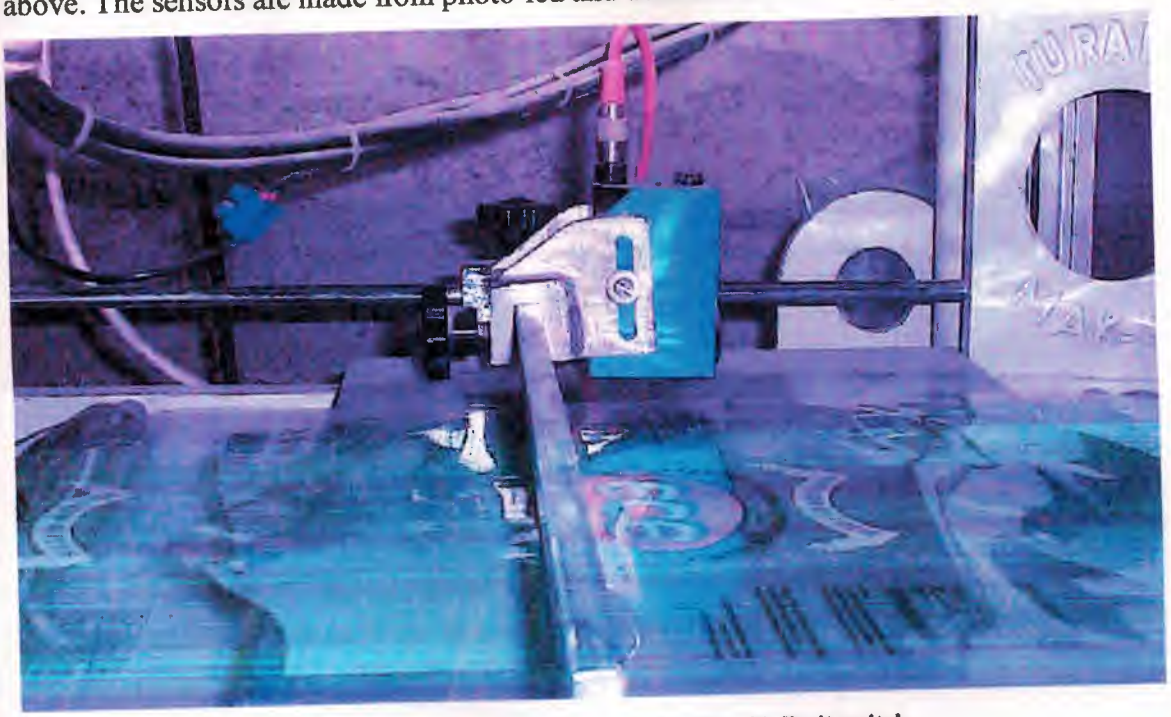


Fig.1.2a Black band reading sensor. Fig1.2b limit switch

4. PORT CONTROL UNIT

4.1. Parallel Port Control

There is a 25 pin D type parallel port on the computer. These can be used for sending information from the computer (output). It is also possible to obtain information from outside and feed these to the computer from outside using the pins (input). It's shown fig1.3.

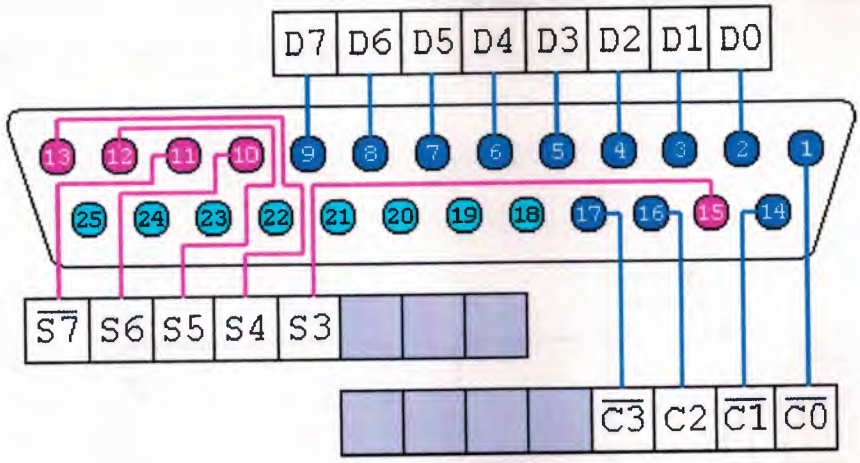


Fig.1.3 Parallel ports input and output

Standard signals sent and received from these ports are at the standard TTL level. As it can be understood the information exchange is made a by a parallel line the system allows more than one parallel port. These ports can be named as LPT1, LPT2, and LPT3. The IBM 25 D type connector numbering can be seen at the figure.

4.2. Port Register

In order to obtain data exchange the data need to be registered somewhere when the computer is connected to a parallel port. Data registration is temporarily made at the register section. Each register is represented by a hexadecimal address. 378H, 379H are such register addresses. As it can be seen the inputs are represented by S and the output are represented as D bites.

4.3. Parallel Port Control Circuit (PPCC)

Each bit taken from the parallel port is known to be at TTL level 5V. When a current connected to the system exceeds 1mA the computer can be damaged. In order to avoid such problem a circuit needs to be designed. It is shown figure1.4. The circuit should enable the controls to be made without any electrical connections. An optocopler system can be used for this purpose. Optocopler is a system which consist a led and photo transistors. This enables a secure and safe control. BC 148 or 2N2222 is activated by the information received from the computer. When the BC 148 is activated it activates the collector and the emitter which then energizes the relay.

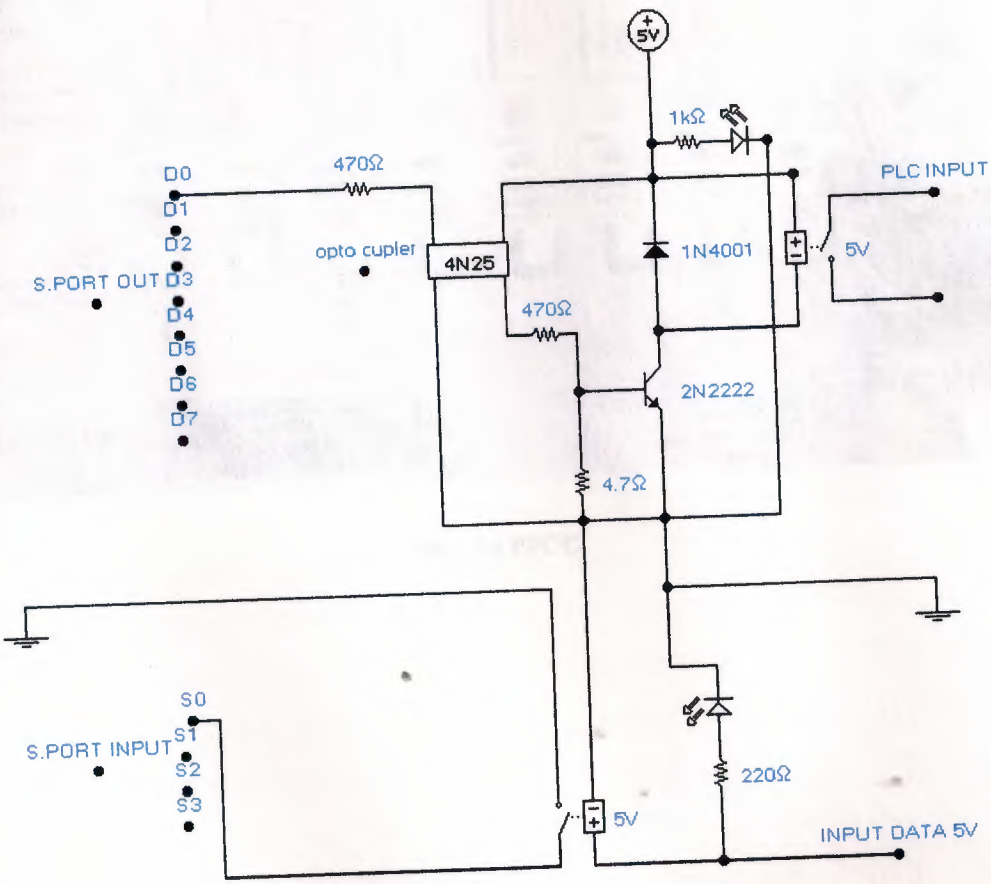


Fig.1.4 Port control circuits block diagram

The 470 Ohms resistance between the optocopler and the Parallel port is safeguard for the led. Led has a feature of 1.7 V. 470 Ohms resistance that is connected to the base of the Transistor limits the 0.7 V, which is necessary to activate it. Relay is activated by 25 V DC voltage, which then activates the PLC inputs. PLC outputs are communicated to the computer by the relay contacts. When the inputs are in connection 4 bit input is at the 0000 position. Energized relay opens the closed contacts and the output and input are evaluated with the program designed with Delphi.

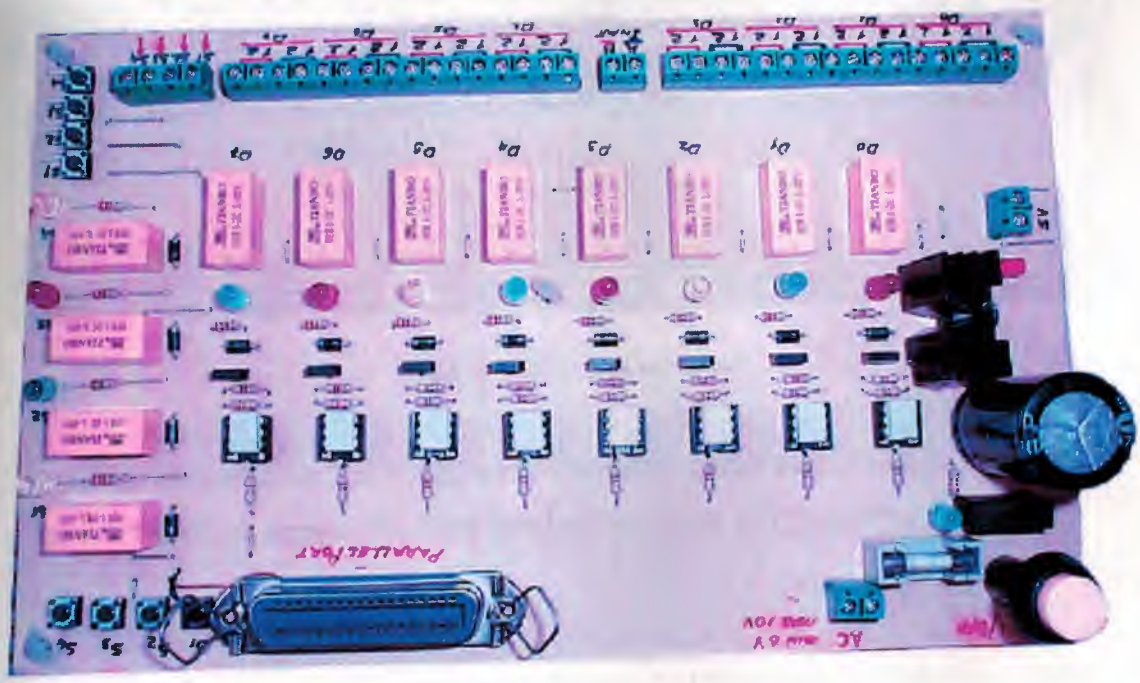


Fig.1.5a PPCC

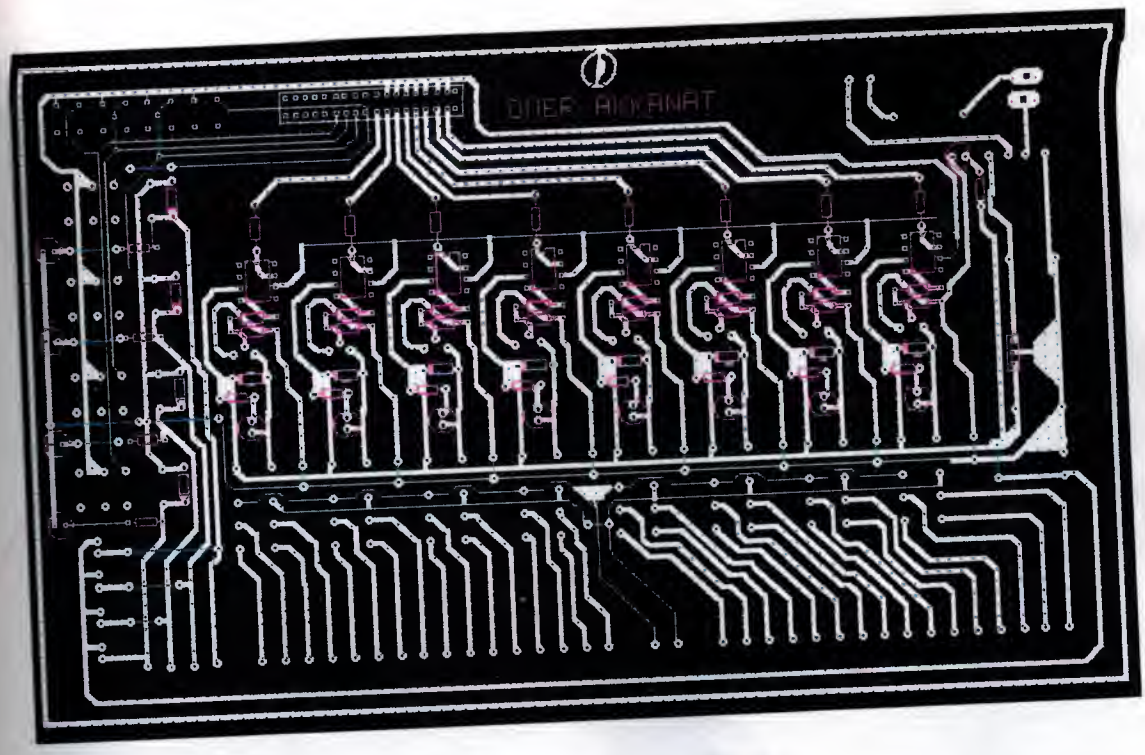


Fig.1.5b PPCC' printed board

I made the drawings for the PPCC with a program called Board Maker. then I made a printed board of the design. The circuit and the assembly can be seen fig.1.5a and b

4.4. Parallel Port Control Program

I selected Borland Delphi 5 for programming fig1.6. This is because it is very adequate to make a good visual design with this program. This program is perfectly in harmony with the assembly language. Assembly language is necessary for the port control. There is a necessary program under each button. The whole program is listed below. As it can be seen in the figure the program is designed for general purposes. The outputs can be controlled individually. There is also a section for the hexadecimal display. The inputs can individually be displayed as binaries as well as hexadecimal. There is a row for a message. The information from the inputs is displayed as messages at this row.

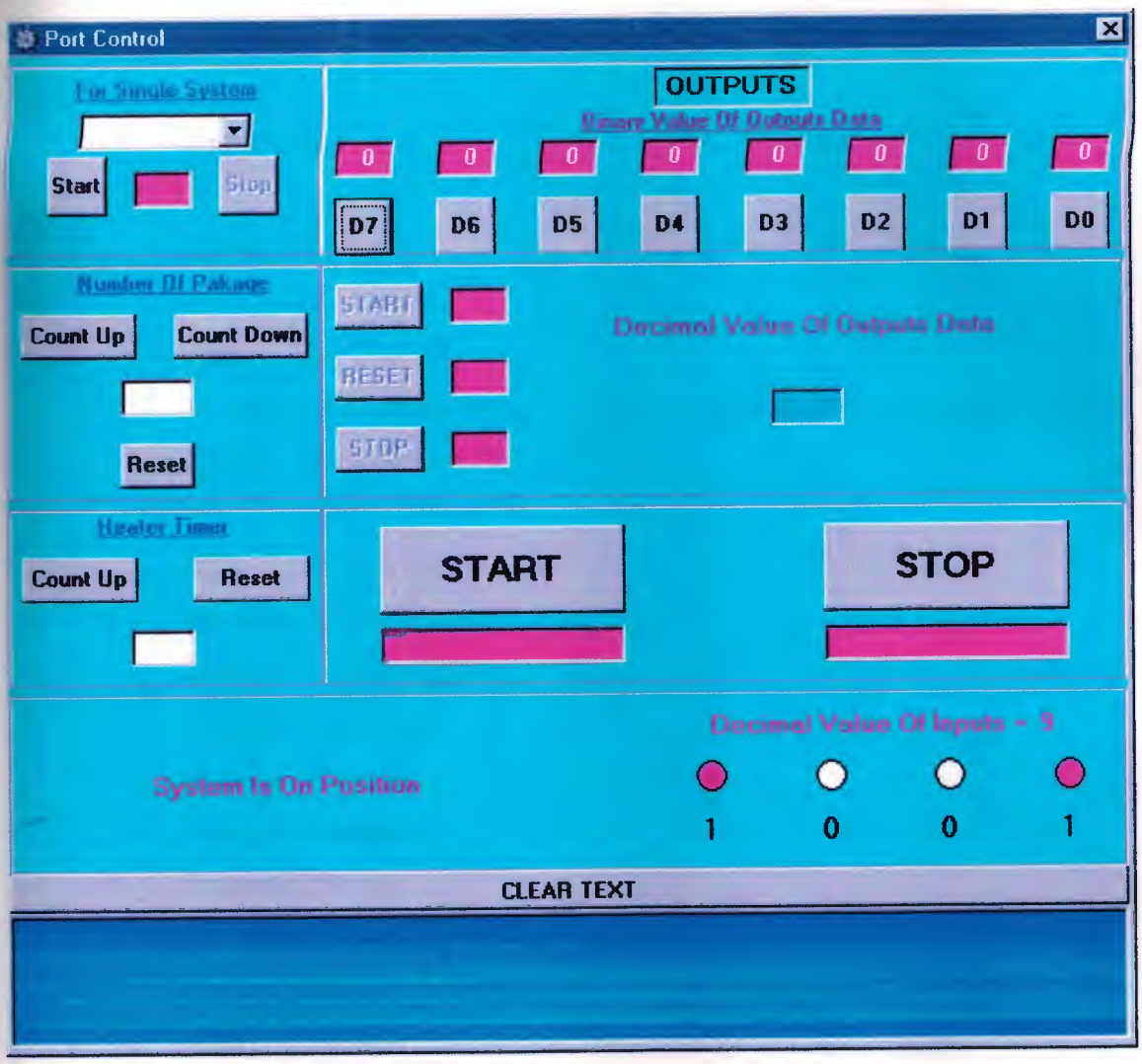


Fig.1.6 program of part control

Borland Delphi 5 PC's port control program

```
unit Unit1;

interface

uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, Buttons, ExtCtrls;

type
TForm1 = class(TForm)
GroupBox1: TGroupBox;
Button3: TButton;
Button4: TButton;
Button5: TButton;
Button6: TButton;
Button7: TButton;
Button8: TButton;
Button9: TButton;
Button10: TButton;
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
Edit5: TEdit;
Edit6: TEdit;
Edit7: TEdit;
Edit8: TEdit;
Edit9: TEdit;
Edit14: TEdit;
GroupBox2: TGroupBox;
```

```
ComboBox1: TComboBox;
Button1: TButton;
Edit1: TEdit;
Button2: TButton;
Label1: TLabel;
Label2: TLabel;
GroupBox3: TGroupBox;
Button14: TButton;
Button16: TButton;
Edit15: TEdit;
Button17: TButton;
Label4: TLabel;
GroupBox4: TGroupBox;
Button15: TButton;
Edit16: TEdit;
Button18: TButton;
Label5: TLabel;
GroupBox5: TGroupBox;
Edit10: TEdit;
Button11: TButton;
Edit12: TEdit;
Button13: TButton;
Button12: TButton;
Edit11: TEdit;
Label3: TLabel;
Edit13: TEdit;
GroupBox6: TGroupBox;
Button19: TButton;
Edit17: TEdit;
Button20: TButton;
Edit18: TEdit;
```

```
Label6: TLabel;
GroupBox7: TGroupBox;
Edit19: TEdit;
Timer1: TTimer;
Led1: TShape;
Led2: TShape;
Led3: TShape;
Led4: TShape;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Memo1: TMemo;
SpeedButton1: TSpeedButton;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure Button11Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure Button8Click(Sender: TObject);
procedure Button9Click(Sender: TObject);
procedure Button10Click(Sender: TObject);
```

```
procedure Button12Click(Sender: TObject);
procedure Button13Click(Sender: TObject);
procedure Button14MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button16MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button17MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button15MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button18MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button14MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button16MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button17MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button15MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button18MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button19MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button19MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button20MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button20MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure Button3MouseDown(Sender: TObject; Button: TMouseButton;
```

```
    Shift: TShiftState; X, Y: Integer);
    procedure Button13MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure Button21Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure Button22MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure Button22MouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure Button23MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure Button24MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure Button25MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure Button23MouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure Button24MouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure Button25MouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton1Click(Sender: TObject);
    procedure GroupBox2Click(Sender: TObject);
    private
    { Private declarations }
    public
    { Public declarations }
    end;

    var
    Form1: TForm1;
```

```
Veri:Integer;
D0:Integer;
D1:Integer;
D2:Integer;
D3:Integer;
D4:Integer;
D5:Integer;
D6:Integer;
D7:Integer;
Cup:Integer;
Toplam:Integer;
Up:Integer;
sonuc:Integer;
implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
if not (combobox1.text='') then
begin
GroupBox1.Enabled:=False;
GroupBox3.Enabled:=False;
GroupBox4.Enabled:=False;
GroupBox6.Enabled:=False;
edit1.color:=clGreen;
Button2.Enabled:=True;
If Combobox1.Text=('Device1') Then veri:=1;
If Combobox1.Text=('Device2') Then veri:=2;
If Combobox1.Text=('Device3') Then veri:=4;
If Combobox1.Text=('Device4') Then veri:=8;
```

```
If Combobox1.Text=('Device5') Then veri:=16;
If Combobox1.Text=('Device6') Then veri:=32;
If Combobox1.Text=('Device7') Then veri:=64;
If Combobox1.Text=('Device8') Then veri:=128;
asm
mov al,veri
mov dx,378h
out dx,al
end;
Button1.Enabled:=False;
Label1.Visible:=True;
//Combobox1.enabled:=False;
end
else
showmessage('Please select any device');
combobox1.setfocus;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
GroupBox1.Enabled:=True;
GroupBox3.Enabled:=True;
GroupBox4.Enabled:=True;
GroupBox6.Enabled:=True;
Button19.Enabled:=True;
Button20.Enabled:=True;
edit1.color:=clRed;
Button1.Enabled:=True;
Button2.Enabled:=False;
Label1.Visible:=False;
Combobox1.Text:='';
//ComboBox1.Enabled:=True;
```

```
asm
mov al,0
mov dx,378h
out dx,al
end;
end;


procedure TForm1.FormCreate(Sender: TObject);
begin
D7:=1;
D6:=2;
D5:=4;
D4:=8;
D3:=16;
D2:=32;
D1:=64;
D0:=128;
Edit15.Text:='';
Edit15.Text:='';
Up:=0;
CUp:=0;
end;

procedure TForm1.Button11Click(Sender: TObject);
begin
If Not (Edit2.Color=clGreen) Then D0:=0;
If Not (Edit3.Color=clGreen) Then D1:=0;
If Not (Edit4.Color=clGreen) Then D2:=0;
If Not (Edit5.Color=clGreen) Then D3:=0;
```

```
begin
Edit3.Text:=' 1';
Edit3.color:=clGreen;
Button11.Enabled:=True;
Button13.Enabled:=True;
Edit12.color:=clYellow;
Button4.enabled:=False;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
Edit4.Text:=' 1';
Edit4.color:=clGreen;
Button11.Enabled:=True;
Button13.Enabled:=True;
Edit12.color:=clYellow;
Button5.enabled:=False;
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
Edit5.Text:=' 1';
Edit5.color:=clGreen;
Button11.Enabled:=True;
Button13.Enabled:=True;
Edit12.color:=clYellow;
Button6.enabled:=False;
end;

procedure TForm1.Button7Click(Sender: TObject);
begin
```

```
Edit6.Text:='  1';
Edit6.color:=clGreen;
Button11.Enabled:=True;
Button13.Enabled:=True;
Edit12.color:=clYellow;
Button7.enabled:=False;
end;

procedure TForm1.Button8Click(Sender: TObject);
begin
Edit7.Text:='  1';
Edit7.color:=clGreen;
Button11.Enabled:=True;
Button13.Enabled:=True;
Edit12.color:=clYellow;
Button8.enabled:=False;
end;

procedure TForm1.Button9Click(Sender: TObject);
begin
Edit8.Text:='  1';
Edit8.color:=clGreen;
Button11.Enabled:=True;
Button13.Enabled:=True;
Edit12.color:=clYellow;
Button9.enabled:=False;
end;

procedure TForm1.Button10Click(Sender: TObject);
begin
Edit9.Text:='  1';
```

```
Edit9.color:=clGreen;
Button11.Enabled:=True;
Button13.Enabled:=True;
Edit12.color:=clYellow;
Button10.enabled:=False;
end;

procedure TForm1.Button12Click(Sender: TObject);
begin
D7:=1;
D6:=2;
D5:=4;
D4:=8;
D3:=16;
D2:=32;
D1:=64;
D0:=128;
Edit2.Text:='  0';
Edit3.Text:='  0';
Edit4.Text:='  0';
Edit5.Text:='  0';
Edit6.Text:='  0';
Edit7.Text:='  0';
Edit8.Text:='  0';
Edit9.Text:='  0';
Button3.Enabled:=True;
Button4.Enabled:=True;
Button5.Enabled:=True;
Button6.Enabled:=True;
Button7.Enabled:=True;
Button8.Enabled:=True;
```

```
Button9.Enabled:=True;
Button10.Enabled:=True;
Edit2.Color:=clRed;
Edit3.Color:=clRed;
Edit4.Color:=clRed;
Edit5.Color:=clRed;
Edit6.Color:=clRed;
Edit7.Color:=clRed;
Edit8.Color:=clRed;
Edit9.Color:=clRed;
Edit10.Color:=clRed;
Edit11.Color:=clRed;
Edit13.Text:='';
asm
mov al,0
mov dx,378h
out dx,al
end;
Button12.Enabled:=False;
GroupBox1.Enabled:=True;
//Button11.Enabled:=True;
end;

procedure TForm1.Button13Click(Sender: TObject);
begin
D7:=1;
D6:=2;
D5:=4;
D4:=8;
D3:=16;
D2:=32;
```

```
D1:=64;
D0:=128;
Edit2.Text:=' 0';
Edit3.Text:=' 0';
Edit4.Text:=' 0';
Edit5.Text:=' 0';
Edit6.Text:=' 0';
Edit7.Text:=' 0';
Edit8.Text:=' 0';
Edit9.Text:=' 0';
Button3.Enabled:=True;
Button4.Enabled:=True;
Button5.Enabled:=True;
Button6.Enabled:=True;
Button7.Enabled:=True;
Button8.Enabled:=True;
Button9.Enabled:=True;
Button10.Enabled:=True;
Edit2.Color:=clRed;
Edit3.Color:=clRed;
Edit4.Color:=clRed;
Edit5.Color:=clRed;
Edit6.Color:=clRed;
Edit7.Color:=clRed;
Edit8.Color:=clRed;
Edit9.Color:=clRed;
Edit12.color:=clRed;
Button11.Enabled:=False;
asm
mov al,0
mov dx,378h
```

```
out dx,al
end;
Button13.Enabled:=False;
end;


procedure TForm1.Button14MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);

begin
if (Up=100) Then Up:=-1;
Up:=Up+1;
Edit15.Text:=inttostr(Up);
asm
mov al,128
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button16MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
if (Up=0) Then Up:=101;
Up:=Up-1;
Edit15.Text:=inttostr(Up);
asm
mov al,64
```

```
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button17MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
Up:=0;
Edit15.Text:=inttostr(Up);
asm
mov al,192
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button15MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
CUp:=CUp+1;
if (CUp=11) Then CUp:=0;
Edit16.Text:=inttostr(CUp);
asm
mov al,32
mov dx,378h
out dx,al
end;
end;
```

```
procedure TForm1.Button18MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
Edit16.Text:='0';
CUp:=0;
asm
mov al,16
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button14MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,0
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button16MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,0
```

```
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button17MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,0
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button15MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,0
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button18MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
```

```
asm
mov al,0
mov dx,378h
out dx,al
end;
end;
procedure TForm1.Button19MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
Edit17.Color:=clGreen;
asm
mov al,1
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button19MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
Edit17.Color:=clRed;
asm
mov al,0
mov dx,378h
out dx,al
end;
end;
```

```
procedure TForm1.Button20MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
Edit18.Color:=clGreen;
asm
mov al,2
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button20MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
Edit18.Color:=clRed;
asm
mov al,0
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button3MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
GroupBox6.Enabled:=False;
GroupBox2.Enabled:=False;
GroupBox3.Enabled:=False;
```

```
GroupBox4.Enabled:=False;
end;

procedure TForm1.Button13MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
GroupBox6.Enabled:=True;
GroupBox2.Enabled:=True;
GroupBox3.Enabled:=True;
GroupBox4.Enabled:=True;
end;

procedure TForm1.Button21Click(Sender: TObject);
begin
asm
mov dx,379h
in al,dx
xor al,128
//mov cl,3
//shr al,cl
and al,11110000b
mov sonuc,al
end;
Edit19.Text:=InttoStr(sonuc);
end;


procedure TForm1.Timer1Timer(Sender: TObject);
Const
x:integer=0;
```

```
begin
asm
mov dx,379h
in al,dx
xor al,128
//mov cl,3
//shr al,cl
and al,11110000b
mov sonuc,al
end;
Edit19.Text:=InttoStr(sonuc);
If Edit19.Text=('0') Then
Begin
Led1.Brush.color:=ClWhite;
Led2.Brush.color:=ClWhite;
Led3.Brush.color:=ClWhite;
Led4.Brush.color:=ClWhite;
Label7.Caption:='0';
Label8.Caption:='0';
Label9.Caption:='0';
Label10.Caption:='0';
Label12.Caption:='0';
end;
If Edit19.Text=('16') Then
Begin
Led1.Brush.color:=ClRed;
Led2.Brush.color:=ClWhite;
Led3.Brush.color:=ClWhite;
Led4.Brush.color:=ClWhite;
Label7.Caption:='1';
Label8.Caption:='0';
```

```
Label9.Caption:='0';
Label10.Caption:='0';
Label12.Caption:='1';
end;
If Edit19.Text=('32') Then
Begin
Led1.Brush.color:=ClWhite;
Led2.Brush.color:=ClRed;
Led3.Brush.color:=ClWhite;
Led4.Brush.color:=ClWhite;
Label7.Caption:='0';
Label8.Caption:='1';
Label9.Caption:='0';
Label10.Caption:='0';
Label12.Caption:='2';
end;
If Edit19.Text=('48') Then
Begin
Led1.Brush.color:=ClRed;
Led2.Brush.color:=ClRed;
Led3.Brush.color:=ClWhite;
Led4.Brush.color:=ClWhite;
Label7.Caption:='1';
Label8.Caption:='1';
Label9.Caption:='0';
Label10.Caption:='0';
Label12.Caption:='3';
end;
If Edit19.Text=('128') Then
Begin
Led1.Brush.color:=ClWhite;
```

```
Led2.Brush.color:=ClWhite;
Led3.Brush.color:=ClRed;
Led4.Brush.color:=ClWhite;
Label7.Caption:='0';
Label8.Caption:='0';
Label9.Caption:='1';
Label10.Caption:='0';
Label12.Caption:='4';
end;
If Edit19.Text=('144') Then
Begin
Led1.Brush.color:=ClRed;
Led2.Brush.color:=ClWhite;
Led3.Brush.color:=ClRed;
Led4.Brush.color:=ClWhite;
Label7.Caption:='1';
Label8.Caption:='0';
Label9.Caption:='1';
Label10.Caption:='0';
Label12.Caption:='5';
end;
If Edit19.Text=('160') Then
Begin
Led1.Brush.color:=ClWhite;
Led2.Brush.color:=ClRed;
Led3.Brush.color:=ClRed;
Led4.Brush.color:=ClWhite;
Label7.Caption:='0';
Label8.Caption:='1';
Label9.Caption:='1';
Label10.Caption:='0';
```

```
Label12.Caption:='6';
end;
If Edit19.Text=('176') Then
Begin
Led1.Brush.color:=ClRed;
Led2.Brush.color:=ClRed;
Led3.Brush.color:=ClRed;
Led4.Brush.color:=ClWhite;
Label7.Caption:='1';
Label8.Caption:='1';
Label9.Caption:='1';
Label10.Caption:='0';
Label12.Caption:='7';
end;
If Edit19.Text=('64') Then
Begin
Led1.Brush.color:=ClWhite;
Led2.Brush.color:=ClWhite;
Led3.Brush.color:=ClWhite;
Led4.Brush.color:=ClRed;
Label7.Caption:='0';
Label8.Caption:='0';
Label9.Caption:='0';
Label10.Caption:='1';
Label12.Caption:='8';
end;
If Edit19.Text=('80') Then
Begin
Led1.Brush.color:=ClRed;
Led2.Brush.color:=ClWhite;
Led3.Brush.color:=ClWhite;
```

```
Led4.Brush.color:=ClRed;
Label7.Caption:='1';
Label8.Caption:='0';
Label9.Caption:='0';
Label10.Caption:='1';
Label12.Caption:='9';
end;
If Edit19.Text=('96') Then
Begin
Led1.Brush.color:=ClWhite;
Led2.Brush.color:=ClRed;
Led3.Brush.color:=ClWhite;
Led4.Brush.color:=ClRed;
Label7.Caption:='0';
Label8.Caption:='1';
Label9.Caption:='0';
Label10.Caption:='1';
Label12.Caption:='10';
end;
If Edit19.Text=('112') Then
Begin
Led1.Brush.color:=ClRed;
Led2.Brush.color:=ClRed;
Led3.Brush.color:=ClWhite;
Led4.Brush.color:=ClRed;
Label7.Caption:='1';
Label8.Caption:='1';
Label9.Caption:='0';
Label10.Caption:='1';
Label12.Caption:='11';
end;
```

```
If Edit19.Text=('192') Then
Begin
Led1.Brush.color:=ClWhite;
Led2.Brush.color:=ClWhite;
Led3.Brush.color:=ClRed;
Led4.Brush.color:=ClRed;
Label7.Caption:='0';
Label8.Caption:='0';
Label9.Caption:='1';
Label10.Caption:='1';
Label12.Caption:='12';
end;
If Edit19.Text=('208') Then
Begin
Led1.Brush.color:=ClRed;
Led2.Brush.color:=ClWhite;
Led3.Brush.color:=ClRed;
Led4.Brush.color:=ClRed;
Label7.Caption:='1';
Label8.Caption:='0';
Label9.Caption:='1';
Label10.Caption:='1';
Label12.Caption:='13';
end;
If Edit19.Text=('224') Then
Begin
Led1.Brush.color:=ClWhite;
Led2.Brush.color:=ClRed;
Led3.Brush.color:=ClRed;
Led4.Brush.color:=ClRed;
Label7.Caption:='0';
```

```
Label8.Caption:='1';
Label9.Caption:='1';
Label10.Caption:='1';
Label12.Caption:='14';
end;
If Edit19.Text=('240') Then
Begin
Led1.Brush.color:=ClRed;
Led2.Brush.color:=ClRed;
Led3.Brush.color:=ClRed;
Led4.Brush.color:=ClRed;
Label7.Caption:='1';
Label8.Caption:='1';
Label9.Caption:='1';
Label10.Caption:='1';
Label12.Caption:='15';
end;
If Edit19.Text=('0') Then Label13.Visible:=True
else
Label13.Visible:=False;
IF sonuc>0 Then
Begin
Label13.Visible:=False;
Label14.Visible:=True;
end
else
Label14.Visible:=False;
If sonuc=240 Then
Begin
Label15.Visible:=True;
end
```

```
Else
Label15.Visible:=False;
end;

procedure TForm1.Button22MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,2
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button22MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,0
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button23MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
```

```
mov al,1
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button24MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,4
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button25MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,8
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button23MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
```

```
begin
asm
mov al,0
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button24MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,0
mov dx,378h
out dx,al
end;
end;

procedure TForm1.Button25MouseUp(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
asm
mov al,0
mov dx,378h
out dx,al
end;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
```

```
begin
Memo1.Lines.Clear;
end;

procedure TForm1.GroupBox2Click(Sender: TObject);
begin

end;

end.
```

5. PLC PROGRAMS

5.1. Introduction

Siemens S7 200 212 CPU has 8 inputs ($I_{0.0}$, $I_{0.1}$.....$I_{0.7}$) and 6 output ($Q_{0.0}$, $Q_{0.1}$,.....$Q_{0.5}$). Since these output and inputs were insufficient for our purpose I made the modifications as it can be seen in the figures1.7a and b and turned the system into a single PLC.

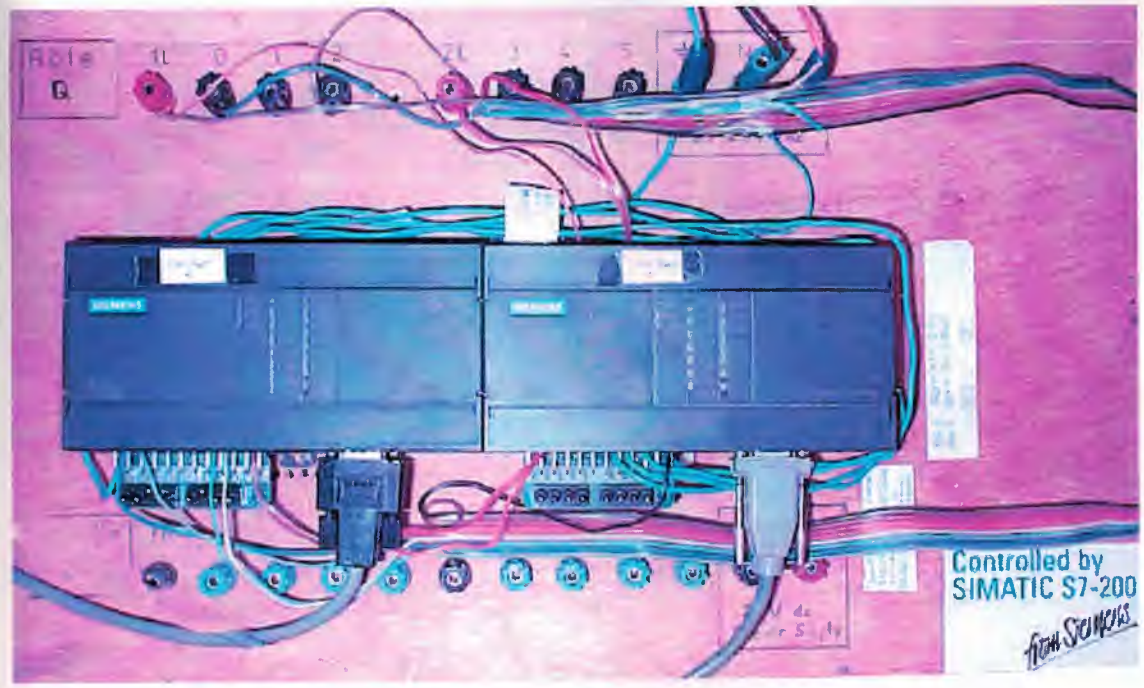


Fig.1.7aPLC's

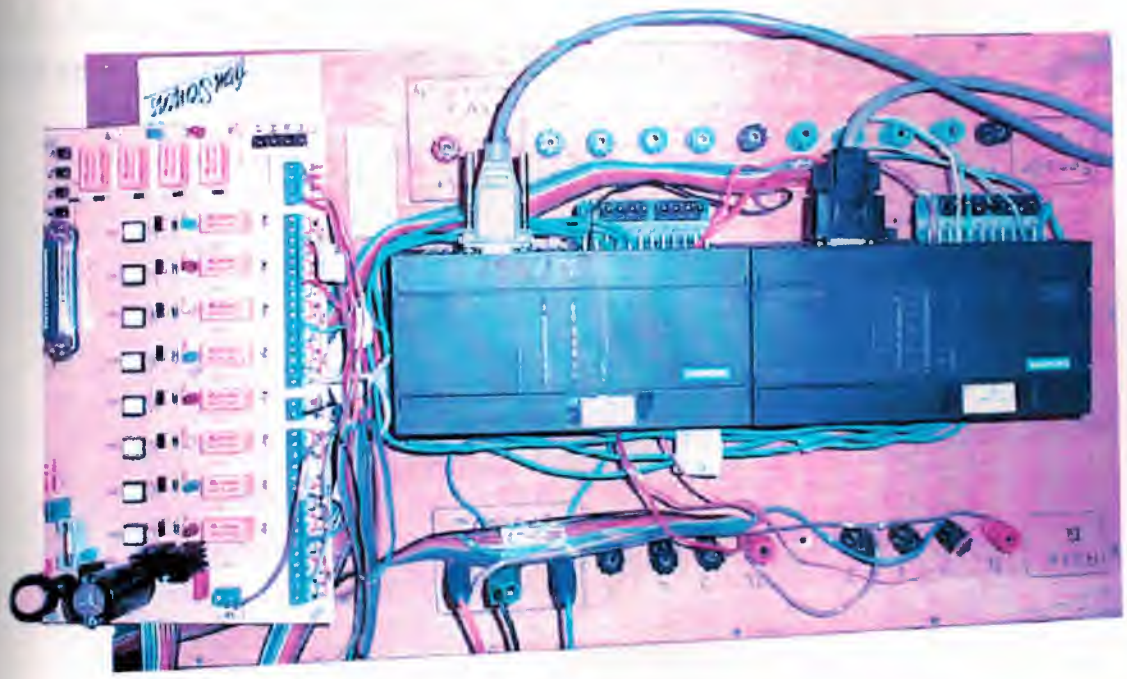


Fig.1.7b

5.2. Ladder Diagram

Each network of the Ladder program is individually explained. ladder diagram is shown.

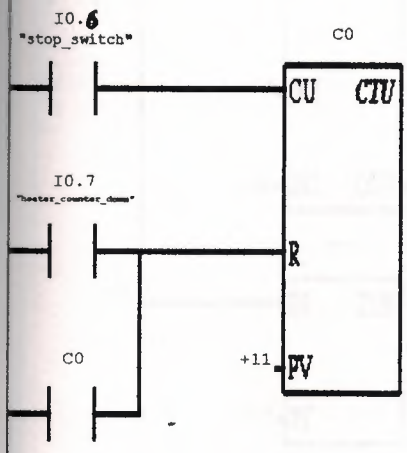
c:\microwin\projects\gproje1.ob1

PROGRAM TITLE COMMENTS

Press F1 for help and example program

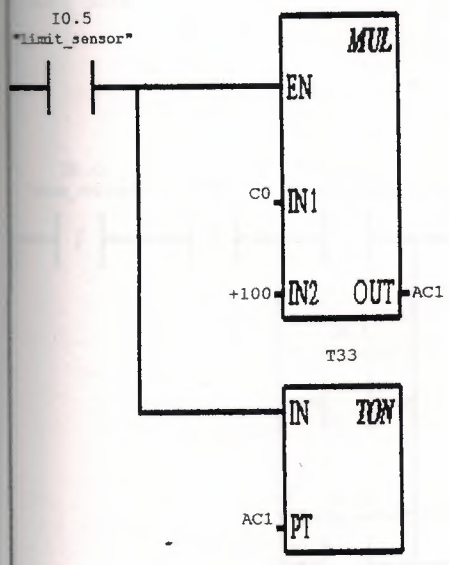
Network 1 Control İnput For The Heater

Counter counts from Co to maximum 11. Controls are made with two up and down buttons. I 0.6 and I 0.7 heater counters are adjusted to count up to ten by equalizing PV+1. The reset button is used in parallel with the heater count reset to allow the system to reset after 10.



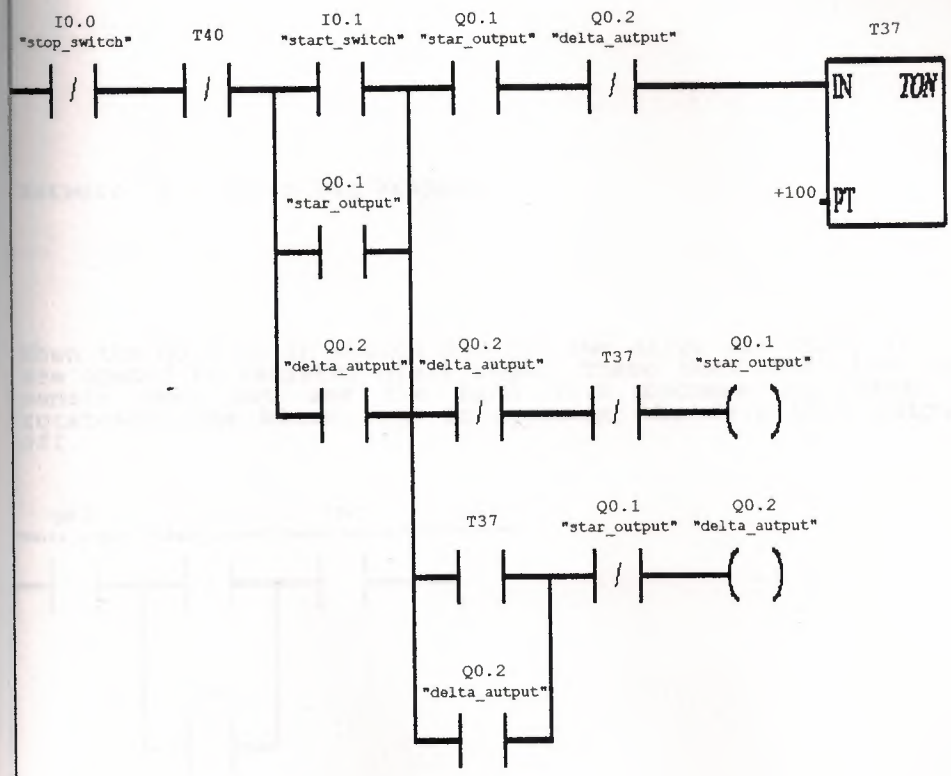
Network 2 Multiplication

Each value entered into counter is multiplied by 100. these are registered at AC1. This function is repeated each time by the I0.5 sensor. For each packaging the figures are entered to the multiplier at AC1. the values are multiplied by 100. the maximum figure that can be registers is 10. this is equal to 10 seconds



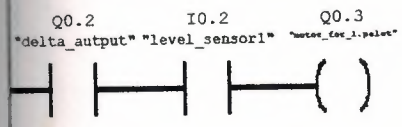
Network 3 Air Cleaner

There is a fan for cleaning the air of the packaging unit. The fan is switched on by a delta starter switch. This is the first unit that is run when the system starts to operate. All the security buttons of the whole system has a parallel connection to the I00 stop button. T40 and a sensor avoids the system to work when it is not necessary. Since there is no other output possibility star and delta are jointly used as the lock for the system. The passage time from star to delta is 10s.



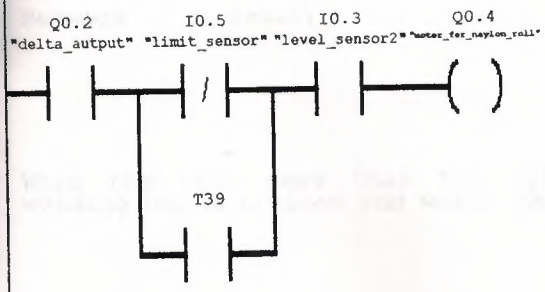
Network 4 First Pallet

When the system is initiated the star and delta outputs energize the unit 4. for the first pallet to work fan motor has to be used as delta. The buttons on the first and second silos should then be switched to on position. When the first silo is full up the sensor on the second silo is automatically turned to on. When the switches are on the pallet starts to work.



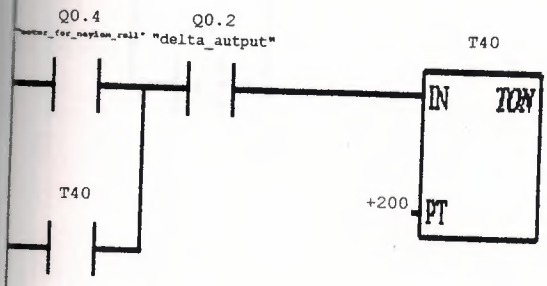
Network 5 Nylon Bag Wrapper

When the Q0.2 is in action and the two silos are full, I0.5 sensors are opened to register black signs. These then start the T39when the sensor does not see the band I0.5 becomes on. When the band rotatesand the black sign is apparent the I0.5 Q0.4 output is cut off.



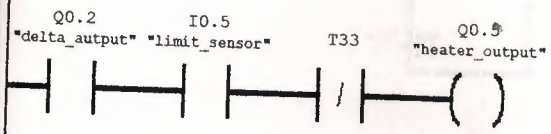
Network 6 Security Of The Nylon Roll

There is a danger that if the I0.5 sensor does not see the black marks the whole role of nylon can be run off and wasted. To avoid such a situation there is an emergency system that when the motor runs more than ten seconds non-stop, this means that there is something wrong and the system is automatically switched off.
Network 7: Heater



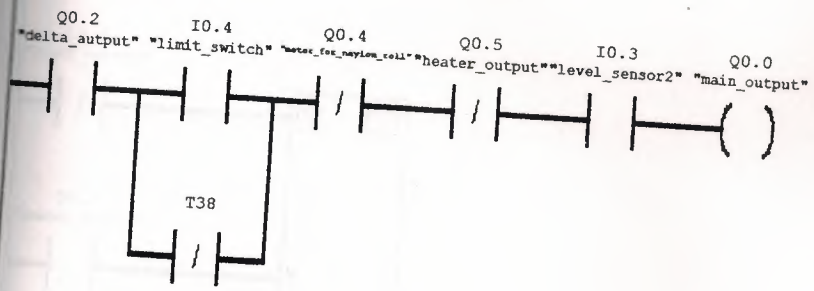
Network 7 Heater

When the I0.5 sees that the nylon is at the required place the welding bar activates and welds the nylon at the required place.



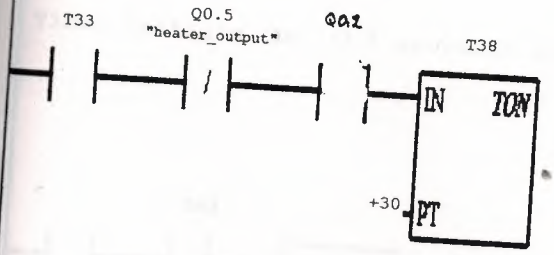
Network 8 Filling

The discharge notor Q0.0 is not active during the welding of nylon takes place. The Q0.0 motor becomes active when I0.3 limit switch is on and the silo 2 is full.



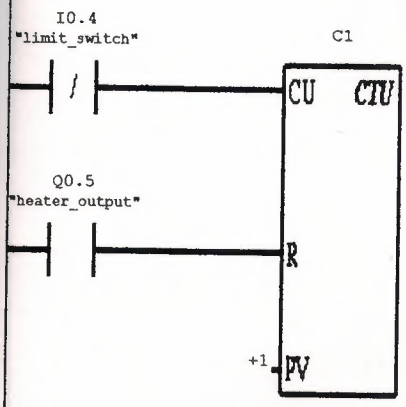
Network 9 Timing

I0.3 switch decides when the Q0.0 motor should stop. When the system is about to empty I0.3 is in off position. In order to activate the Q0.0 motor the I0.4 limit switch must be on. T38 30 ms gives the initial movement.



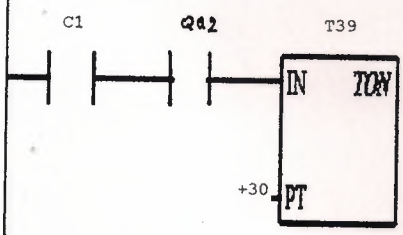
Network 10 Revolution Timing

After the discharge took place counter enables Q0.4 to place a new bag in the position. After the first revolution T39 counter starts counting. Safety mechanism avoids the system to continue when the I0.4 is not properly functioning.

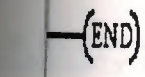


Network 11 Timer

Timer initiate the Q0.4 movement abd provides safety to Q0.0.



Network 12 End Of Ladder Program

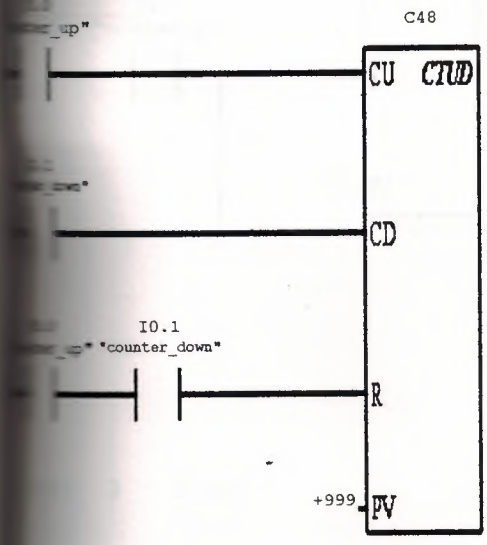


TITLE COMMENTS

F1 for help and example program

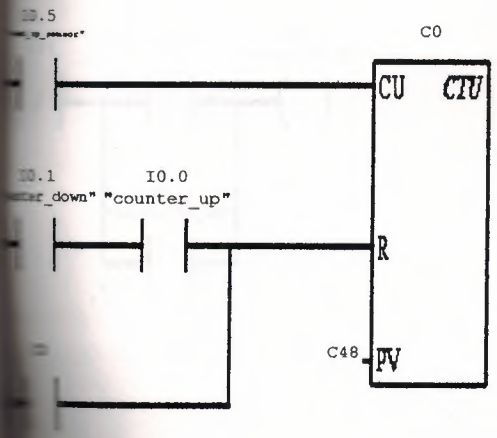
Network 1 Input For Counter

I0.0 is for count up and I0.1 is for count down. When the both buttons are pressed simultaneously the counter is reset.



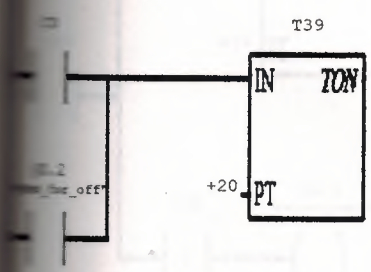
work 2 Limit

.5 of the first PLC counts the process in order to obtain as many output as icated at the C48 counter. A pulse is sent for each stage. And when I0.5 and .0 starts C0 becomes reset.



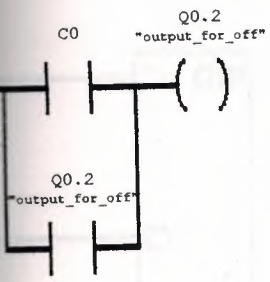
work 3 Timer

the system reaches the registered number of operations it needs to be switched . A timer is used to stop the system after a certain period.



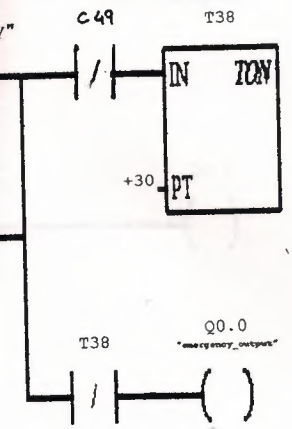
4 Output Of The Counter

desired number of packaging is made 24VDC is sent to the I0.0 output of the C. This increases the contact



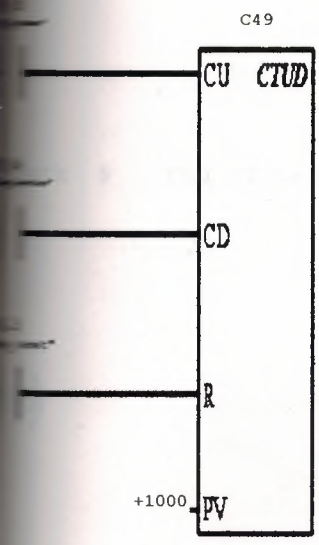
5 Emergency

e of emergency 30ms direct current is given to stop all the motors. PLC 1 I0.0 to off position by giving current to the Q0.0



Work 6 Service Counter

The required number of packaging is made the whole system stops.



Work 7 For Output

Network 8 For Output

Network 9 End Of Ladder Program

5.3. Statement List

First PLC Program Title Comments

NETWORK 1 //Control input for the heater

//Counter counts from Co to maximum 11. Controls are made with two up and down buttons. I O.6 and I 0.7 heater counters are adjusted to count up to ten by equalizing PV+1. The reset button is used in parallel with the heater count reset to allow the system to reset after 10.

```
LDI   "stop_switch"
LD    "heater_counter_down"
O     C0
CTU   C0, +11
```

NETWORK 2 //Multiplication

//Each value entered into counter is multiplied by 100. these are registered at AC1. This function is repeated each time by the I0.5 sensor. For each packaging the figures are entered to the multiplier at AC1. the values are multiplied by 100. the maximum figure that can be registers is 10. this is equal to 10 seconds

```
LD    "limit_sensor"
MOVW  C0, AC1
MUL   +100, AC1
TON   T33, AC1
```

NETWORK 3 //air cleaner

//There is a fan for cleaning the air of the packaging unit. The fan is switched on by a delta starter switch. This is the first unit that is run when the system starts to operate. All the security buttons of the whole system has a parallel connection to the I00 stop button. T40 and a sensor avoids the system to work when it is not necessary. Since there is no other

output possibility star and delta are jointly used as the lock for the system. The passage time from star to delta is 10s.

```
LDN   "stop_switch"
AN    T40
LD    "start_switch"
O     "star_output"
O     "delta_autput"
ALD
LPS
A     "star_output"
AN    "delta_autput"
TON   T37, +100
LRD
AN    "delta_autput"
AN    T37
=     "star_output"
LPP
LD    T37
O     "delta_autput"
ALD
AN    "star_output"
=     "delta_autput"
```

NETWORK 4 //First Pallet

//When the system is initiated the star and delta outputs energize the unit 4. for the first pallet to work fan motor has to be used as delta. The buttons on the first and second silos should then be switched to on position. When the first silo is full up the sensor on the second silo is automatically turned to on. When the switches are on the pallet starts to work.

```
LD    "delta_autput"
A     "level_sensor1"
=     "motor_for_1.palet"
```

NETWORK 5 //Nylon bag wrapper

//When the Q0.2 is in action and the two silos are full, I0.5 sensors are opened to register black signs. These then start the T39when the sensor does not see the band I0.5 becomes on. When the band rotatesand the black sign is apparent the I0.5 Q0.4 output is cut off.

```
LD    "delta_autput"
LDN   "limit_sensor"
O     T39
ALD
A     "level_sensor2"
=     "motor_for_naylon_roll"
```

NETWORK 6 //Security of the Nylon roll

//There is a danger that if the I0.5 sensor does not see the black marks the whole role of nylon can be run off and wasted. To avoid such a situation there is an emergency system that when the motor runs more than ten seconds non-stop, this means that there is something wrong and the system is automatically switched off. Network 7://Heater

```
LD    "motor_for_naylon_roll"
O     T40
A     "delta_autput"
TON   T40, +200
```

NETWORK 7 //Heater

//When the I0.5 sees that the nylon is at the required place the welding bar activates and welds the nylon at the required place.

```
LD    "delta_autput"
```

```
A    "limit_sensor"
AN   T33
=    "heater_output"
```

NETWORK 8 //Filling

//The discharge notor Q0.0 is not active during the welding of nylon takes place. The Q0.0 motor becomes active when I0.3 limit switch is on and the silo 2 is full.

```
LD   "delta_autput"
LD   "limit_switch"
ON   T38
ALD
AN   "motor_for_naylon_roll"
AN   "heater_output"
A    "level_sensor2"
=    "main_output"
```

NETWORK 9 //Timing

//I0.3 switch decides when the Q0.0 motor should stop. When the system is about to empty I0.3 is in off position. In order to activate the Q0.0 motor the I0.4 limit switch must be on. T38 30 ms gives the initial movement.

```
LD   T33
AN   "heater_output"
TON  T38, +30
```

NETWORK 10 //Revolution timing

//After the discharge took place counter enables Q0.4 to place a new bag in the position. After the first revolution T39 counter starts counting. Safety mechanism avoids the system to continue when the I0.4 is not properly functioning.

```
LDN   "limit_switch"
LD    "heater_output"
CTU   C1, +1
```

NETWORK 11 //Timer

//Timer initiate the Q0.4 movement abd provides safety to Q0.0.

```
LD    C1
TON   T39, +30
```

NETWORK 12 //End of lader program

```
MEND
```

Second PLC Program Title Comments

NETWORK 1 //Entry o pallet number

//I00 is for count up and I0.1 is for count down. When the both buttons are pressed simultaneously the counter is reset.

```
LD    "counter_up"
LD    "counter_down"
LD    "counter_up"
A     "counter_down"
CTUD  C48, +999
```

NETWORK 2 //Limit

//Q0.5 of the first PLC counts the process in order to obtain as many output as indicated at the C48 counter. A pulse is sent for each stage. And when I0.5 and I0.0 starts CO becomes reset.

```
LD    "count_up_sensor"
LD    "counter_down"
A     "counter_up"
O     C0
CTU   C0, C48
```

NETWORK 3 //Timer

//When the system reaches the registered number of operations it needs to be switched off. A timer is used to stop the system after a certain period.

```
LD    C0
O     "output_for_off"
TON   T39, +20
```

NETWORK 4 //Output of the Counter

//When the desired number of packaging is made 24VDC is sent to the I0.0 output of the first PLC. This increases the contact

```
LDN   T39
LD    C0
O     "output_for_off"
ALD
=     "output_for_off"
```

NETWORK 5 //Emergency

//In case of emergency 30ms direct current is given to stop all the motors. PLC 1 turns I0.0 to off position by giving current to the Q0.0

```
LD    "emergency"
O     C49
TON   T38, +30
AN    T38
=     "emergency_output"
```

NETWORK 6 //Service counter

//When the required number of packaging is made the whole system stops.

```
LD    "count_up_sensor"
LD    "count_down_service"
LD    "service_reset"
CTUD  C49, +1000
```

NETWORK 7 // For output

```
LD    "count_up_sensor"
=     Q0.4
```

NETWORK 8 // For output

```
LD    I0.6
=     Q0.5
```

NETWORK 9 //End of ladder program

```
MEND
```

CONCLUSION

The aim of the study is showing that any process can be managed remotely with ease. Need for remote managing could appear in health-critical or dangerous conditions, being far away from job, etc. It could be extremely useful for managers to check or administer or just for taking information as if using visual phone.

In fact, a production unit may have a PLC or other types of control device on their processes, so that they may not need this part of job. In this case, if some computers make the data collection and control the process with modem/internet availability, then server and client side programs are applicable for them.

In our study, since the work done in local area network, the process control by client realized in less than 1 second, but of course, in a wide area network connection would take more time due to distance and the quality of computers and their peripherals. Our study worked from modem to telephone line successfully. But operation occurred slowly. From local area network, more fast than wide area network.

Our study is possible to send mobile telephone message or e-mail but we must add something to the software. It is reality that my project supply extra security to my study area. We can increase security to use technology and we can control all system whatever we went from the away.

I believe that the proposed work is an economic and useful solution for large and distributed companies, and workshops, which have to work together, even for institutions having health critical or dangerous jobs.

REFERENCES

Lange, L., 1998. Analysis & Forecast: The Internet. IEEE Spectrum, 35(1): 37-42.

Dutta-Roy, A., 1999. Networks for Homes. IEEE Spectrum, 36(12): 26-33.

Dutta-Roy, A., 1999. Bringing Home the Internet. IEEE Spectrum, 36(3): 32-38.

Yourdon, 1997. Analysis & Forecast: Software Engineering Viewpoint, IEEE Spectrum, 34 (1):66-67

Choi, S.Y., and Whinston, A.B., 1999. The Future of E-Commerce: Integrate and Customize. IEEE Computer, 32(1):133-138.

Floyd. Electronic Devices

TAŞBAŞI Abdurrahman PC Donanımı

THOMAS L. Floyd. Digital Fundamentals Sixth Edition

THOMAS L. Floyd. Electronic Devices Fourth Edition

ALTINBAŞAK Orhan . Bilgisayar Uygulamaları

Electronics Workbench

Boardmaker

Rsview32: http://www.software.rockwell.com/rsview32.

Cimplicity: http://www.gefanuc.com/cimplicity/ındex.shtml.

Factory suit webserver: http://www.wonderware.com/products/fswebsrvr71.htm.

Wincc Siemens: http://www.wincc.de.

PC-Anywhere: http://www.symantec.com/pcanywhere/ındex.html