Design and Implementation of a Programmable Logic Controller Lab: An Internet Based Monitoring and Control of a Process

A Thesis Presented to
The Faculty of the
Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University
In Partial Fulfillment
Of the Requirement for the Degree
Master of Science
by
Aleksey Imaev

August, 2002
ACKNOWLEDGEMENTS

I would like to thank many people who have made this thesis possible:

Dr. Dennis Irwin, my supervisor, for all his support and encouragement throughout the preparation of this thesis;

Dr. Thomas Scott, for his great invaluable help. He has given me much instruction and has helped me to create lab exercises for a Programmable Logic Controller (PLC) course;

Mr. Victor Hanna, who had kindly provided me with his PLC class notes and experiments;

Dr. Robert Lipset, for the various bits of knowledge and his help in obtaining RSView32 WebServer software;

Dr. Trevor Hale, for reviewing the manuscript of this thesis, for his valuable comments on the thesis, and for his check and correction of the thesis.

Also I would like to pay tribute to people at Rockwell Software technical support, who provided me with an example of Active Server Pages program. This program was a great help in creating a demonstration of a remote process monitoring and control presented in this thesis.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Programmable Logic Controller</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 PLC components</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 PLC programming</td>
<td>3</td>
</tr>
<tr>
<td>1.1.3 Recent PLC enhancements</td>
<td>4</td>
</tr>
<tr>
<td>1.2 SCADA (Supervisory Control and Data Acquisition System) and HMI (Human Machine Interface)</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Objective and Scope of this Thesis</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Organization of this Thesis</td>
<td>8</td>
</tr>
<tr>
<td>2. LABORATORY DESIGN AND HARDWARE SETUP</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Overall Lab Design</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Advantages of the ControlLogix System</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Mechanical Design of the Trainer</td>
<td>13</td>
</tr>
<tr>
<td>2.4 Communication</td>
<td>15</td>
</tr>
<tr>
<td>3. SOFTWARE SETUP</td>
<td>17</td>
</tr>
<tr>
<td>3.1 RSLinx</td>
<td>17</td>
</tr>
<tr>
<td>3.1.1 Configuring Communication Drivers in RSLinx</td>
<td>18</td>
</tr>
<tr>
<td>3.1.2 RSLinx as OPC Server</td>
<td>25</td>
</tr>
<tr>
<td>3.1.2.1 What is OPC?</td>
<td>26</td>
</tr>
<tr>
<td>3.1.2.2 Configuring RSLinx as an OPC Server</td>
<td>26</td>
</tr>
<tr>
<td>3.2 RSLogix 5000</td>
<td>29</td>
</tr>
<tr>
<td>3.3 RSViwer32</td>
<td>39</td>
</tr>
<tr>
<td>4. LAB EXERCISES</td>
<td>42</td>
</tr>
</tbody>
</table>
5. MONITORING AND CONTROL IN A REMOTE ACCESS ENVIRONMENT: TRAFFIC LIGHTS EXAMPLE

5.1 Traffic Lights System Description

5.2 Ladder Logic Diagram

5.2.1 Routine MainRoutine

5.2.2 Subroutine Day_Mode

5.2.3 Subroutine Night_Mode

5.3 System Setup for the Remote Access Example

5.4 Software Interactions

5.5 Creating an OPC Automation Client

5.6 Web Interface

5.6.1 Page "View/Change Parameters"

5.6.2 Page "Traffic Lights Monitoring"

5.7 Concluding remarks

6. CONCLUSIONS

BIBLIOGRAPHY

APPENDIX

A. LAB EXERCISES

B. LADDER LOGIC PROGRAM

C. ASP CODE
LIST OF TABLES

Table 5.1: The association of the trainer lights with the signals on the traffic lights. 47
Table 5.2: The states of the system in the Day mode .................................................. 47
Table 5.3: The states of the system in the Night mode................................................. 48
Table 5.4: Elements of seq_array ............................................................................... 50
Table 5.5: The parameters that control the timings ..................................................... 52
Table 5.6: Elements of seq_array2 ............................................................................ 52
LIST OF FIGURES

Figure 1.1. Allen-Bradley SLC 100 trainer unit ......................................................... 7
Figure 1.2. Allen-Bradley ControlLogix System ................................................................. 8
Figure 2.1. Training Station ............................................................................................... 11
Figure 2.2. Basic ControlLogix5000 Demo ....................................................................... 11
Figure 2.3. Network setup of the PLC lab ......................................................................... 12
Figure 3.1. Configure Drivers dialog box .......................................................................... 19
Figure 3.2. Add New RSLinx Driver dialog box (RS-232) .................................................. 19
Figure 3.3. Configure RS-232 DF1 Devices dialog box ....................................................... 20
Figure 3.4. RSWho window: checking communication via RS-232 port ......................... 21
Figure 3.5. Assigning IP address to Ethernet module ......................................................... 22
Figure 3.6. 1756-ENET Configuration dialog box ............................................................... 22
Figure 3.7. Ethernet port configuration .............................................................................. 23
Figure 3.8. Add New RSLinx Driver dialog box (Ethernet) .................................................. 24
Figure 3.9. Configuring Ethernet module ........................................................................... 24
Figure 3.10. RSWho window: Checking communication via Ethernet ............................... 25
Figure 3.11. DDE/OPC Topic Configuration dialog box ...................................................... 27
Figure 3.12. Browsing to Logix5550 processor .................................................................. 28
Figure 3.13. DDE/OPC configuration: Data Collection tab ................................. 28
Figure 3.14. DDE/OPC configuration: Advanced Communication tab ................. 29
Figure 3.15. RSLogix 5000 screen layout ......................................................... 30
Figure 3.16. New Controller dialog box ............................................................. 31
Figure 3.17. New project in RSLogix 5000.......................................................... 32
Figure 3.18. Right-click on I/O configuration and select New Module .................... 33
Figure 3.19. Select Module Type window........................................................... 33
Figure 3.20. Module Properties – Local (1756-IB16D 2.1) window....................... 35
Figure 3.21. Module Properties – Local (1756-OB16D 2.1) window ...................... 35
Figure 3.22. Module Properties – Local (1756-IF6I 1.1) window ......................... 36
Figure 3.23. Module Properties – Local (1756-OF6VI 1.1) window ...................... 36
Figure 3.24. Module Properties – Local (1756-ENET /B 2.1) window ................... 38
Figure 3.25. Choosing communication path in Who Active dialog box ................. 38
Figure 3.26. The Project window....................................................................... 40
Figure 3.27. The Project window: System folder ............................................... 40
Figure 3.28. Channel Editor dialog box............................................................... 41
Figure 3.29. Node Editor dialog box ................................................................. 41
Figure 5.1. 4-way intersection controlled by traffic lights.................................... 45
Figure 5.2. The state diagram of the system in the Day mode............................. 48
Figure 5.3. The state diagram of the system in the Night mode ......................... 48
Figure 5.4. Time diagram: input to the sequencer .............................................. 51
Figure 5.5. Software Interactions..................................................................... 54
Figure 5.6. OPC Automation objects ............................................................... 55
Figure 5.7. The structure of an OPC client ................................................................. 56
Figure 5.8. Sample of the code that reads the tag value from the PLC memory .... 57
Figure 5.8. Screenshot of the Web page “View/Change Parameters” ............... 58
Figure 5.9. Screenshot of the Web page “Traffic Lights Monitoring” ............... 59
1. INTRODUCTION

1.1 Programmable Logic Controller

According to Simpson [1], a programmable logic controller (PLC) is a solid-state device, which uses a programmable memory to perform a controlling function. PLC's are capable of storing instructions, such as sequencing, timing, counting, arithmetic, data manipulation, and communication, to control industrial machines and processes [2]. Essentially, the PLC is an industrial strength microcomputer, which is designed to operate in harsh industrial environments and intended to perform its controlling functions flawlessly.

Before the advent of the PLC, manufacturing plants relied on electromechanical control systems. The development of the programmable logic controller in the late 1960's revolutionized the manufacturing industry. It was first introduced in 1968 by Bedford Associates, was called Modular Digital Controller (MODICON 084), and was used by the Hydramatic Division of the General Motors Corporation [3]. The advent of the PLC was a big step forward from relay-controlled systems. It was easy to install, used much less space and energy, had troubleshooting capabilities, and could easily be reprogrammed [2].
1.1.1 PLC components

All PLC’s have a CPU, memory, input and output interfaces (I/O interfaces), a programming interface, and a power supply. Some PLC’s have communication interfaces.

The input/output interface and the communication interface can be fixed or modular. The fixed I/O is typical of small PLC’s and come in one package while modular I/O is divided into separate modules that can be plugged. The modular I/O allows greater flexibility because the user can choose only the modules that are required for a given application [2]. The modular controller consists of a rack, power supply, CPU, I/O and/or communication modules, and an operator interface. The CPU and all the modules communicate via backplane, located at the rear of the rack.

The CPU executes PLC programs and controls communication among the PLC modules. The processor has memory for storing the program and results of the logical operations.

The I/O interface provides a way by which the PLC talks to field devices. The input interface converts signals received from the field devices into the form that the CPU can process. In contrast, the output interface takes signals from the CPU and converts them into forms that can produce control actions by external devices [1].

The programming interface is used to enter a program, usually relay ladder logic, into the memory of the PLC. A personal computer or a hand-held device can act as a program terminal.
The power supply supplies power to all the modules and the CPU.

The communication interface is an interface between the PLC and various types of industrial networks PLC can talk to such as ControlNet, Ethernet/IP, DeviceNet and Data Highway.

1.1.2 PLC programming

As described in Bryan and Bryan [2], the most commonly used PLC programming language is ladder logic. It looks very similar to hardwired logic circuits used to control machines and equipment. The ladder logic program represents electrical sequences of operation. It is essentially a symbolic set of instructions used to create controller program for a PLC. The basic elements of the ladder logic program are contacts and coils that are interconnected to obtain a desired control logic that is to be entered into the memory of the PLC.

A ladder logic program consists of one or more rungs of logic. A rung in the program contains a set of input and output instructions. The input instructions form a logic condition that is evaluated by the controller as being true or false. The output instructions such as ‘relays’, ‘timers’, ‘counters’ etc. are either enabled or disabled consistent with the input conditions.
1.1.3 Recent PLC enhancements

Since its advent PLC’s have improved a lot, they became more powerful and flexible. In the early 1970’s communication improvements begun to appear. Programmable controllers got the ability to talk to other PLC’s and to control the process remotely.

Technological advances in the PLC industry continue today. Changes include both hardware and software updates. Recent PLC hardware advances described in [2] include:

- Faster scan times using advanced microprocessor technology.
- Small, low cost PLC’s.
- Intelligent I/O interface modules such as PID (proportional-integral-derivative), network, communication, motion control and language modules (e.g., BASIC or Pascal).
- Mechanical design improvements.
- Special interfaces such as thermocouples and fast-response inputs can be plugged-in directly to the controller.
- Peripheral equipment has improved operator interface techniques.

Software enhancements according to [2] include:

- More powerful instructions and advanced functional block instructions provide powerful programming capabilities using simple programming commands.
- Controllers incorporating high-level languages such as BASIC and C.
- Diagnostics and fault detection allows to diagnose controller malfunctions as well as to diagnose failures of the controlled machine or process.
- Improvements in data handling and manipulation instructions.

All these enhancements have made the PLC a very powerful tool for today’s control needs.

1.2 SCADA (Supervisory Control and Data Acquisition System) and HMI (Human Machine Interface)

The early programmable logic controllers interfaced with the operator via push buttons and switches for control and lamps for indication. The interface was similar to a relay control panel [3].

Owing to the introduction of the personal computer (PC) in the 1980’s the development of a computer-based interface to the operator has begun. The tool for computer-based interface is called SCADA. SCADA stands for supervisory control and data acquisition system. As described in [4] SCADA is a purely software package that is positioned on top of the PLC to which it is interfaced. SCADA systems allow gathering of data in real time from remote locations in order to monitor and control process or equipment. The graphical display serves as an interface to the operator and shows the representation of the plant or equipment in graphical form. The
communication can be performed via a local area network (LAN), via proprietary networks that are used for control, or via the Internet.

The term Human Machine Interface (HMI) sometimes used in a similar content as the term SCADA used. HMI is a computer-based graphical interface that allows the operator to monitor or control plant or equipment. It simply serves as media between the plant and the operator that collects process data from the plant and displays this data in easy-to-read graphical form.

1.3 Objective and Scope of this Thesis

This thesis deals with the design and implementation of a programmable logic controller laboratory. The PLC lab is aimed to students who need practical knowledge in the broad area of process control. This includes ladder logic programming, control system design, data acquisition, and the development of HMI applications for automation. To assist student in learning, five ladder logic programming exercises and one HMI application exercise were developed herein.

The industrial digital control lab was completely redesigned as a part of this effort. Before the upgrade the lab had employed Allen-Bradley’s SLC 100 programmable controllers that were more than 15 years old and could not demonstrate many of the capabilities of modern PLC’s. The SLC 100 trainer unit is shown in Figure 1.1.

The main limitations of SLC 100 system are:
- Limited number of I/O points.
- Software running under DOS. Poor user interface.
- No communication.
- Limited variety of ladder logic instructions.

A need was identified for a more advanced and more up-to-date PLC system that would have the capabilities of modern PLC’s. An Allen-Bradley ControlLogix system (see Figure 1.2) was selected for this project. It provided the base for the lab design. It is a powerful control system that absorbed practically all advanced characteristics of the modern PLC’s.
A sub-objective of this project is to demonstrate an approach to real time monitoring and control of the process in a remote access environment. A web based interface was implemented that allows an operator to control a given system remotely by exchanging data with the PLC.

Figure 1.2. Allen-Bradley ControlLogix System

1.4 Organization of this Thesis

The thesis is organized as follows. Chapter 2 explains overall design of the lab and discusses advantages and technical characteristics of the ControlLogix System. Chapter 3 introduces software that has been used as the part of the project and describes configuration of that software. Chapter 4 presents the laboratory exercises developed to serve as training manual for those interested in learning the basics of PLC programming and HMI application development. Chapter 5 presents the demonstration of the Internet based remote monitoring and control concept using
traffic lights example. Finally, Chapter 6 summarizes the work presented herein and
draws some conclusions.
2. LABORATORY DESIGN AND HARDWARE SETUP

2.1 Overall Lab Design

The design of the laboratory is based on the Allen-Bradley’s ControlLogix programmable controller. The lab has 12 training stations and a server. The training station shown in Figure 2.1 includes:

- A Basic ControlLogix5000 Demo (referred herein as the “trainer”) as shown in Figure 2.2. It consists of the ControlLogix programmable controller connected to I/O base. The I/O base includes both digital and analog devices.
- A personal computer with RSLinx, RSLogix 5000 and RSView32 software.

All the training stations and the server are linked with Ethernet LAN (Local Area Network). The server computer has two network interface cards and is interfaced to both the Ethernet LAN and the Internet. It serves as a gateway between the Ethernet LAN and the outside world. The sketch of the network setup is shown in Figure 2.3.
Figure 2.1. Training Station

Figure 2.2. Basic ControlLogix5000 Demo
2.2 Advantages of the ControlLogix System

The ControlLogix system is one of the most advanced control platforms on the market and the most enhanced programmable controller Allen-Bradley has to offer today. It offers high performance in wide range of control applications.
According to the manufacturer [5] the advantages of the ControlLogix PLC include:

- Modularity of I/O, memory and communication interfaces.
- Integration with existing PLC systems. The ControlLogix system can exchange messages with other systems on other networks.
- Modules can be removed and inserted under power without disruption to other modules.
- Fast messaging between modules in the backplane, within network or between networks.
- Compact design.
- Motion control is integrated to ControlLogix processor.
- Distributed processing via EtherNet/IP, DeviceNet, or ControlNet networks.
- Distributed I/O at locations remote from the processor can be connected across EtherNet/IP, ControlNet, DeviceNet, and Universal Remote I/O links.
- Multiple processor modules can be plugged into the same backplane.

2.3 Mechanical Design of the Trainer

As mentioned above, the trainer is basically the ControlLogix PLC wired to I/O devices.
The ControlLogix control system consists of processor module, I/O modules and Ethernet/IP communication module in a single ControlLogix chassis with a power supply. It is rack-based modular PLC. The power supply mounts directly onto the ControlLogix I/O chassis. The controller modules are plugged in the ControlLogix I/O chassis (rack). The chassis are available in sizes of 4, 7, 10, 13, or 17 module slots. Any module can be plugged-in into any slot of the I/O chassis.

The trainer chassis has 7 slots. The following modules occupy six of these slots: a ControlLogix processor, DC input module, DC output module, analog input module, analog output module, and an Ethernet/IP communication module.

The DC input module (catalog #: 1756-IB16D), located in the slot 0, provides up to 16 inputs and has diagnostics for open circuit detection. It is wired to 12 pushbuttons (Trainer inputs DI0 – DI11) and to 6 toggle switches (Trainer inputs DI12 – DI15).

The DC output module (catalog #: 1756-OB16D), located in the slot 1, provides up to 16 outputs. It is wired to 12 output lights (Trainer outputs DO0 – DO11).

The ControlLogix 5550 processor (catalog #: 1756-L1) is located in the slot 2.

The analog input module (catalog #: 1756-IF6I), located in the slot 3, has 6 isolated analog inputs. It is wired to two potentiometers (Trainer potentiometers A10 and A11).
The analog output module (catalog #: 1756-OF6VI), located in the slot 4, has 6 isolated analog outputs. It is wired to two voltmeters (Trainer voltmeters A00 and AO1).

### 2.4 Communication

Communications provides the core for the ControlLogix system. Interfacing with communication networks is modular with the exception of an RS-232 (DF1/DH-485-protocol) port built into the processor. ControlLogix communication modules support EtherNet/IP, ControlNet, DeviceNet, Remote I/Om and Data Highway Plus networks. The modular approach to communication allows for more flexible system configuration: add as many or as few network modules as you need [5, 6, 7].

EtherNet/IP stands for Ethernet Industrial Protocol. According to [8] it is an open industrial networking standard that relies on:

- IEEE 802.3 Physical and Data Link standard.
- Ethernet TCP/IP protocol suite (Transmission Control Protocol/Internet Protocol), the Ethernet industry standard.
- Control and Information Protocol (CIP), the protocol that provides real-time I/O messaging and information / peer-to-peer messaging. ControlNet and DeviceNet networks also use CIP.
All the PLC’s in the lab are connected to the EtherNet/IP network. It offers high-speed communication between the PLC’s. EtherNet/IP communication interface modules on each trainer provide interface to the EtherNet/IP network. The EtherNet/IP network is local (limited to the boundaries of the laboratory) and can be used to deliver messages from one PLC to another or from any PLC to any personal computer.

Every PC and every PLC in the lab is assigned a unique IP address. IP address range is 10.10.10.1 through 10.10.10.26 (subnet mask is 255.255.255.0). Default gateway is set to 10.10.10.254. The server PC’s addresses are 10.10.10.1 (LAN) and 132.235.15.90 (the Internet). The training PC’s IP addresses end with the odd digit (such as in 10.10.10.3) and the PLC Ethernet interface module’s IP addresses end with the even digit (such as in 10.10.10.4).

The communication between a PLC and its programming interface (the PC located next to the trainer) can be accomplished in two ways: via a serial RS232 port or via an Ethernet network connection. The communication via Ethernet is often preferable because it provides faster communications.
3. SOFTWARE SETUP

This chapter introduces the software products that have been used as the part of the project and describes the configuration of that software.

The following software is installed on each training PC: RSLogix 5000, RSLinx and RSView32. RSLogix 5000 is used for PLC programming, RSLinx is used to setup communication and RSView32 is used for HMI development. All the software products operate in Microsoft Windows environment and are highly interoperable.

3.1 RSLinx

RSLinx is a software product used to communicate controller and various applications such as RSLogix 5000 and RSView32. It provides Allen-Bradley programmable controller access to a wide variety Rockwell Software and Allen-Bradley applications. In addition it can support many other software applications and talk to various devices and communication networks [9]. RSLogix 5000 and RSView32 both use RSLinx as a communication server to connect to PLC’s. For example when you download a ladder logic program to PLC, RSLogix 5000
automatically lanches RSLinx, which provides a communication path between PC and PLC. Graphical interface of RSLinx makes navigation through the network easy.

3.1.1 Configuring Communication Drivers in RSLinx

Communication drivers in RSLinx need to be configured before use. A communication driver is a software interface that is used to communicate between the ControlLogix 5550 and RSLinx. Two drivers need to be configured: one for communication via the RS232 port and another for communication via the Ethernet. The serial RS232 cable needs to be connected to the COM1 port on a PC and to the RS232 port on the Logix5550 processor module. Also, the PC and the EtherNet/IP communication interface module need to be connected to the Ethernet LAN.

To configure the RS232 driver run RSLinx, then select ‘Configure Drivers’ under the ‘Communications’ menu. The ‘Configure Drivers’ dialog box will be displayed as shown in Figure 3.1. From the ‘Available Driver Types’ list box select ‘RS-232 DF1 devices’ and click on the ‘Add New...’ button. The ‘Add new RSLinx Driver’ dialog box will be displayed as shown in Figure 3.2. Click on OK to close the dialog box. The ‘Configure RS-232 DF1 Devices’ window will be displayed, enter the configuration data as shown in Figure 3.3 and then click on ‘Auto-Configure’ button. The message “Auto Configuration Successful!” should appear on the right from the Auto-Configure button. Click on OK to close the ‘Configure RS-232 DF1 Devices’ window. Click on Close to close the ‘Configure Drivers’ window.
Figure 3.1. Configure Drivers dialog box

Figure 3.2. Add New RSLinx Driver dialog box (RS-232)
Figure 3.3. Configure RS-232 DF1 Devices dialog box

To check that the RS232 communication was setup correctly, select ‘RSWho’ under the ‘Communications’ menu and down from the RS232 driver you should be able to browse to all the modules on the ControllLogix backplane as shown in Figure 3.4.
Figure 3.4. RSWho window: checking communication via RS-232 port

When the RS232 driver is configured, the next step is to configure the communication via Ethernet. The Ethernet module requires its unique IP address. First we assign an IP address to the module and then we configure a driver for it.

To assign the IP address to the Ethernet interface module in RSLinx, select ‘RSWho’ under the ‘Communications’ menu and browse to the Ethernet module (1756-ENET/B). Right click on the Ethernet module icon as shown in Figure 3.5 and select ‘Module Configuration’. The ‘1756-ENET Configuration dialog’ box will be displayed as shown in Figure 3.6. Select Port Configuration tab and enter an IP address, subnet mask, and default gateway address (see example of 1756-ENET configuration as shown in Figure 3.7).
Figure 3.5. Assigning IP address to Ethernet module

Figure 3.6. 1756-ENET Configuration dialog box
To configure driver for communication via Ethernet in RSLinx, select ‘Configure Drivers’ under the ‘Communications’ menu. The ‘Configure Drivers’ dialog box will be displayed as shown in Figure 3.1. From the ‘Available Driver Types’ list box select ‘Remote Devices via Linx Gateway’ and click on ‘Add New...’ button. The ‘Add new RSLinx Driver’ dialog box will be displayed as shown in Figure 3.8. Click on OK to close the dialog box. The ‘Configure Remote Devices via Linx Gateway’ window will be displayed. In the ‘Server’s IP Address or hostname’ field enter the IP address of the Ethernet interface module as shown in Figure 3.9.
Figure 3.8. Add New RSLinx Driver dialog box (Ethernet)

Figure 3.9. Configuring Ethernet module
To check that the Ethernet communication was setup correctly, select ‘RSWho’ under the ‘Communications’ menu and down from the Ethernet driver you should be able to browse to all the modules on the ControlLogix backplane as shown in Figure 3.10.

![Figure 3.10. RSWho window: Checking communication via Ethernet](image)

3.1.2 RSLinx as OPC Server

RSView32 accesses data in the ControlLogix controller via RSLinx using OPC interface. The following subsections explain what OPC is and how to configure an OPC server in RSLinx.
3.1.2.1 What is OPC?

In the past the most common way for software applications to interface with the hardware was by developing drivers. However the main disadvantage of this method is that everyone must write a driver for a particular vendor's hardware. OPC standard helps to solve that problem.

OPC stands for OLE for Process Control. OLE is short for object linking and embedding. OPC is a standard that defines a method for exchanging real-time automation data among PC-based clients using Microsoft operating systems [11]. OPC is an integral part of Microsoft Windows operating system. An OPC server application serves as a link to data source and it can provide that data to OPC clients. An OPC client is a program that receives data from a server [10].

With OPC a vendor can develop a server with OPC interface to communicate to a data source. As a result any OPC client can access the device via OPC server.

For more information about OPC standard refer to official web site of the OPC Foundation [11].

3.1.2.2 Configuring RSLinx as an OPC Server

In RSLinx in order to access data from the PLC we need to create a topic. A topic represents a specific path to the Logix5550 processor.
To create a topic, run RSLinx and select ‘Topic Configuration...’ under the ‘DDE/OPC’ menu. The ‘DDE/OPC Configuration’ dialog box will pop up as shown in Figure 3.11. From this dialog box, click on New. From the data source tab, browse to the Logix5550 processor as shown in Figure 3.12. Enter a name for the new topic in the Topic List. Enter the specific information for the Data Collection tab as shown in Figure 3.13. Enter the specific information for the Advanced Communication tab as shown in Figure 3.14. Click on Apply to add a new topic. Click Yes when prompted with the message "Are you sure you want to update topic (topic_name)?". Click on Done to close the DDE/OPC Topic Configuration dialog box.

Figure 3.11. DDE/OPC Topic Configuration dialog box
Figure 3.12. Browsing to Logix5550 processor

Figure 3.13. DDE/OPC configuration: Data Collection tab
3.2 RSLogix 5000

RSLogix 5000 is a software package that is used to program and configure ControlLogix controllers. It offers symbolic programming with structures and arrays and instruction set that serves many types of applications [12]. RSLogix 5000 operates on Microsoft Windows operating systems. RSLogix 5000 software is used to train students the techniques of ladder logic programming. In addition to the ladder logic programming it also supports the programming with function block diagrams.
RSLogix screen layout is shown in Figure 3.15. Screen component definitions [13]:

- **Title Bar**: Shows the name of the program and the current opened project.
- **Menu Bar**: Used for selecting pull down menu options.
- **Tool Bar**: Used for selecting various functions.
- **Instruction Bar**: Used for quickly selecting various instructions.
- **Scroll Bar**: Used for scrolling through rungs of ladder logic.
• Project/Controller View: It is a visual representation of the projects file structure.

• Status Bar: Shows the time, cursor location, file number, describes menu selections and icons when the cursor is over them.

• Online Bar: Shows communication status and allows access to various online and offline operations.

To configure RSLogix 5000 run the program and create a new project from the File menu by clicking on New. The ‘New Controller’ dialog box will be displayed. Select the processor type, type in the name of the controller, select the chassis type and the slot where processor is located as shown in Figure 3.16. When you click OK, RSLogix 5000 will create a new project as shown on Figure 3.17.

Figure 3.16. New Controller dialog box
After the new project has been created I/O modules and communication module have to be configured. We need to create and configure the DC input, the DC output, the analog input, the analog output and the Ethernet modules.

To add a new module to the I/O configuration list right click on the I/O Configuration folder in Project/Controller view and select ‘New Module’ as shown in Figure 3.18. The ‘Select Module Type’ window will be displayed as shown in Figure 3.19. Use this screen to select a module type then click on OK. One should find module type information on the backside of the module lid.
Figure 3.18. Right-click on I/O configuration and select New Module

Figure 3.19. Select Module Type window
To create and configure the DC input module add a new module, the ‘1756-IB16D 16 Point 10V-30V DC Diagnostic Input’, to the I/O configuration list. The ‘Module Properties – Local (1756-IB16D 2.1)’ window will be displayed. Enter selections for this module as shown in Figure 3.20 (type in 0 in the slot field because the DC input module is located in slot 0). Click on Finish to continue.

To create and configure the DC output module add a new module, the ‘1756-OB16D 16 Point 19.2V-30V DC Diagnostic Output’, to the I/O configuration list. The ‘Module Properties – Local (1756-OB16D 2.1)’ window will be displayed. Enter selections for this module as shown in Figure 3.21 (type in 1 in the slot field because the DC output module is located in slot 1). Click on Finish to continue.

To create and configure the analog input module add a new module, the ‘1756-IF6I, 6 Channel Isolated Voltage/Current Analog Input’, to the I/O configuration list. The ‘Module Properties – Local (1756-IF6I 1.1)’ window will be displayed. Enter selections for this module as shown in Figure 3.22 (type in 3 in the slot field because the analog input module is located in slot 3). Click on Finish to continue.

To create and configure the analog output module add a new module, the ‘1756-OF6VI 6 Channel Isolated Analog Voltage Output’, to the I/O configuration list. The ‘Module Properties – Local (1756-OF6VI 1.1)’ window will be displayed. Enter selections for this module as shown in Figure 3.23 (type in 4 in the slot field because the Analog Output module is located in slot 4). Click on Finish to continue.
Figure 3.20. Module Properties – Local (1756-IB16D 2.1) window

Figure 3.21. Module Properties – Local (1756-OB16D 2.1) window
Figure 3.22. Module Properties – Local (1756-IF6I 1.1) window

Figure 3.23. Module Properties – Local (1756-OF6VI 1.1) window
To create and configure the Ethernet module add a new module, the ‘1756 Ethernet Bridge’, to the I/O configuration list. The ‘Module Properties – Local (1756-ENET /B 2.1)’ window will be displayed. Enter selections for this module as shown in Figure 3.24 (type in 5 in the slot field because the Ethernet module is located in slot 5). Click on Finish to continue.

The next step is to specify the path to the Logix5550 processor. Select Who Active under the Communications menu. The ‘Who Active’ dialog box will be displayed. At this point, if RSLinx is not already lunched, RSLogix 5000 will start RSLinx application automatically because it uses RSLinx to communicate with the PLC. There are two options for communication to the PLC: a path via the RS232 port and a path via the Ethernet connection. We want to select communication via the Ethernet because its data transfer rate is much faster than the data transfer rate of RS232. Navigate to Logix5550 processor, to make that path a default path check ‘Apply Current Path to Project’ check box as shown in Figure 3.25. Click on Apply button. Downloading a project to the PLC can now happen by clicking on the Download button.
Figure 3.24. Module Properties – Local (1756-ENET/B 2.1) window

Figure 3.25. Choosing communication path in Who Active dialog box
3.3 RSView32

RSView32 is integrated SCADA and HMI software package. It is used to create graphical displays for monitoring and controlling machinery and equipment [14]. RSView32 uses RSLinx as OPC (OLE for process control) server or DDE (Dynamic Data Exchange) server to get access to the information in the PLC.

RSView32 needs to be configured as an OPC client that uses RSLinx as OPC server to obtain data from the controller.

Run the RSView 32 program and create a new project from the File menu by clicking on New. The ‘Create New Project’ dialog box will be displayed. Enter the name of the project (e.g., “My_Project”), specify the directory where you want to save project files, and click on Open. RSView32 will create a new project. The ‘Project’ window will be displayed as shown in Figure 3.26. Open the ‘System’ folder in the ‘Project’ window as shown in Figure 3.27. Double click on Channel. The ‘Channel editor’ dialog box will be displayed. Select parameters as shown in Figure 3.28 and click on OK, to close the dialog box. Double click on Node in the Project window. The ‘Node editor’ window will pop up. Choose the data source – OPC Server. Specify the name of the node and the server name by clicking on “…” and selecting “RSLinx OPC Server”. Select parameters as shown in Figure 3.29 and click on Accept. Close the ‘Node editor’ window.
Figure 3.26. The Project window

Figure 3.27. The Project window: System folder
Figure 3.28. Channel Editor dialog box

Figure 3.29. Node Editor dialog box
4. LAB EXERCISES

Lab exercises are the part of a PLC course that is designed to give students a “hands-on” understanding of logic and operation of a PLC. The PLC course is taught using ControlLogix controllers, RSLogix 5000 programming software and RSView32 software. It gives students an opportunity to learn and experiment on real PLC systems.

The aim of lab exercises for students to learn the basics of the ladder logic programming in RSLogix 5000 environment as well as to introduce HMI design using RSView32. Each student works with his own simulated workstation, which includes end devices and the PLC.

Six lab exercises were developed to cover various topics. Five of them are focused on ladder logic programming with RSLogix 5000 and one exercise is aimed on HMI design with RSView32.

Here is a list of exercise topics:

- Program AND/OR conditions using Examine On and Examine Off statements.
- Program a bit latch commands and timers.
• Program counters.

• Program sequencers.

• Program an analog I/O using Math, Compare and Move instructions.

• Design a human-machine interface using RSView32.

The lab material is covered using examples that guide a student through the steps of solving a given problem. The use of examples is very important because it provides a student with a systematic approach to solving a given problem.

Students work with fully configured software. The configuration of the software is described in the Chapter 3. It is a time consuming process and only needs to be done once. With a pre-configured project a student can concentrate more on actual ladder logic programming and HMI design and less on the details of configuration. The exercises are included in APPENDIX A.
5. MONITORING AND CONTROL IN A REMOTE ACCESS ENVIRONMENT: TRAFFIC LIGHTS EXAMPLE

There is increasing trend towards remote monitoring and control of manufacturing operations. This enables process engineers with direct access to the process settings and controls from remote locations. This chapter explains an Internet based approach to monitoring and control of a process in a remote access environment using traffic lights example. The demonstration utilizes Active Server Pages (ASP) and OLE for Process Control (OPC) Automation Interface to access the data in the PLC and to display the results on the web page.

This chapter is organized as follows: First the description of the traffic lights system is given. Then we design a controller for the traffic lights using ladder logic programming language and the ControlLogix controller. Next the software required for the demonstration is described in detail. Then we design a web-based interface that will provide us with the ability to monitor and control the traffic lights system. Finally, we draw some conclusions and suggest improvements that can be made to the system.
5.1 Traffic Lights System Description

Consider a 4-way intersection controlled by traffic lights. Figure 5.1 shows the layout of road lanes and traffic lights. Street A is controlled by the traffic lights A1 and A2. Street B is controlled by the traffic lights B1 and B2. Lights A1 and A2 are operating synchronously as are lights B1 and B2. When the traffic lights switch from red to green to yellow in a regular repetitive manner we call it a day mode. A night mode is when the lights A1 and A2 that control traffic on the street A display a flashing yellow signal and the lights B1 and B2 that control traffic on the street B display a flashing red signal.

Figure 5.1. 4-way intersection controlled by traffic lights
There are five parameters that control the operation of the system:

- A Boolean variable that specifies whether the system is in the Day mode or the Night mode: `theMode`.
- The period of time the green signal is “ON” on the traffic lights A1 and A2: `timing_green_A`.
- The period of time the yellow signal is “ON” on the traffic lights A1 and A2: `timing_yellow_A`.
- The period of time the green signal is “ON” on the traffic lights B1 and B2: `timing_green_B`.
- The period of time the yellow signal is “ON” on the traffic lights B1 and B2: `timing_yellow_B`.

The timing parameters are sufficient to determine the timings for the red lights in the Day mode:

- The period of time (in seconds) the red signal is “ON” on the traffic lights A1 and A2:

  \[
  \text{timing\_red\_A} = \text{timing\_green\_B} + \text{timing\_yellow\_B} + 1 \quad (1)
  \]

- The period of time (in seconds) the red signal is “ON” on the traffic lights B1 and B2:

  \[
  \text{timing\_red\_B} = \text{timing\_green\_A} + \text{timing\_yellow\_A} + 1 \quad (2)
  \]
As mentioned earlier in the Chapter 2, the DC output module of the PLC is wired to the trainer lights DO0 through DO11. Let these trainer lights imitate the signals on the traffic lights as shown in the Table 5.1.

Table 5.1: The association of the trainer lights with the signals on the traffic lights

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>DO0</td>
<td>DO1</td>
<td>DO2</td>
</tr>
<tr>
<td>Yellow</td>
<td>DO4</td>
<td>DO5</td>
<td>DO6</td>
</tr>
<tr>
<td>Green</td>
<td>DO8</td>
<td>DO9</td>
<td>DO10</td>
</tr>
</tbody>
</table>

The day mode operation of the traffic light system can be described using a state diagram with 6 states. These states are defined in the Table 5.2. The state diagram for the day mode operation is given in Figure 5.2.

Table 5.2: The states of the system in the Day mode

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1 (S1)</td>
<td>Red</td>
<td>Green</td>
<td>Red</td>
<td>Green</td>
</tr>
<tr>
<td>State 2 (S2)</td>
<td>Red</td>
<td>Yellow</td>
<td>Red</td>
<td>Yellow</td>
</tr>
<tr>
<td>State 3 (S3)</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
</tr>
<tr>
<td>State 4 (S4)</td>
<td>Green</td>
<td>Red</td>
<td>Green</td>
<td>Red</td>
</tr>
<tr>
<td>State 5 (S5)</td>
<td>Yellow</td>
<td>Red</td>
<td>Yellow</td>
<td>Red</td>
</tr>
<tr>
<td>State 6 (S6)</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
</tr>
</tbody>
</table>
The night mode operation of the traffic light system can be described using a state diagram with 2 states. These states are defined in the Table 5.3 and the state diagram for the Night mode operation is given in Figure 5.3.

Table 5.3: The states of the system in the Night mode

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1 (S1)</td>
<td>Red</td>
<td>OFF</td>
<td>Red</td>
<td>OFF</td>
</tr>
<tr>
<td>State 2 (S2)</td>
<td>OFF</td>
<td>Yellow</td>
<td>OFF</td>
<td>Yellow</td>
</tr>
</tbody>
</table>

Figure 5.3. The state diagram of the system in the Night mode
5.2 Ladder Logic Diagram

The ladder logic program that controls the operation of the lights is attached in Appendix B. It consists of the main routine named MainRoutine and two subroutines named Day_Mode and Night_Mode.

5.2.1 Routine MainRoutine

The ladder diagram for the MainRoutine contains the logic that decides between the Day mode and the Night mode of operation. The decision is made upon the value of the Boolean tag (variable) called theMode. If theMode = 0 than the program will jump to subroutine Day_Mode. Else, if theMode = 1, the program will jump to subroutine Night_Mode.

5.2.2 Subroutine Day_Mode

The subroutine Day_Mode contains the logic that controls the operation of the lights in the day mode. It has 13 rungs numbered from 0 to 12. The key instruction in this subroutine is SQO (Sequencer Output) instruction. It uses a list of words contained in array seq_array. When enabled it recalls a word from the array and moves that word through the mask to another memory location Local:1:O.Data – DC
outputs. When the end of list is reached the sequencer will return to the first word and the process repeats. The mask specifies which bits to block or pass. The hexadecimal mask value 16#0FFF means that we want to pass only the bits that control the 12 trainer lights.

The words in the sequencer array seq_array represent the states of the traffic lights as shown in Table 5.4.

<table>
<thead>
<tr>
<th>State</th>
<th>Address</th>
<th>Binary Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>seq_array[1]</td>
<td>0000_0000_0000_0000_0000_0101_0000_1010</td>
</tr>
<tr>
<td>S2</td>
<td>seq_array[2]</td>
<td>0000_0000_0000_0000_0000_0000_0101_1010</td>
</tr>
<tr>
<td>S3</td>
<td>seq_array[3]</td>
<td>0000_0000_0000_0000_0000_0000_0000_1111</td>
</tr>
<tr>
<td>S4</td>
<td>seq_array[4]</td>
<td>0000_0000_0000_0000_0000_1010_0000_0101</td>
</tr>
<tr>
<td>S5</td>
<td>seq_array[5]</td>
<td>0000_0000_0000_0000_0000_0000_0101_0101</td>
</tr>
<tr>
<td>S6</td>
<td>seq_array[6]</td>
<td>0000_0000_0000_0000_0000_0000_0000_1111</td>
</tr>
</tbody>
</table>

The other instruction used is a timer TON instruction. It controls the operations based on time. When enabled TON instruction accumulates time. Timer accumulator value (.ACC) specifies the number of milliseconds that have elapsed since the TON instruction was enabled. Timer preset (.PRE) is the value (1 msec units) that the accumulated value must reach before the instruction sets the done (.DN) bit.

Tags sl, s2, s3, s4, s5 and s6 are Boolean variables. These tags make up the input condition to the sequencer. When sl becomes set it means that the system needs to switch to state S1, when s2 becomes set it means that the system needs to switch to
state S2, etc. Timers \textit{timer1}, \textit{timer2}, \textit{timer3}, \textit{timer4}, \textit{timer5} and \textit{timer6} control the period of time the system stays in states S1, S2, S3, S4, S5 and S6 respectfully. Timers \textit{timer7}, \textit{timer8}, \textit{timer9}, \textit{timer10}, \textit{timer11} and \textit{timer12} are used to clear bits \textit{s1}, \textit{s2}, \textit{s3}, \textit{s4}, \textit{s5} and \textit{s6} respectfully one millisecond second after these bits where set. Figure 5.4 shows how the input condition to the sequencer changes over time (sketched not to scale).

Tag \textit{start} is initially set to 1. It is used to make rung-condition-in for rung 0 true the first time subroutine is executed after it was downloaded to the PLC.

![Figure 5.4. Time diagram: input to the sequencer](image)

The preset values of timers: \textit{timer1.PRE}, \textit{timer2.PRE}, \textit{timer3.PRE}, \textit{timer4.PRE}, \textit{timer5.PRE} and \textit{timer6.PRE} are the periods of time that the system spends in each state. The parameters that control the timings for the traffic light system are shown in Table 5.5. These parameters can be changed during program run-time. The values of \textit{timer3.PRE} and \textit{timer6.PRE} are fixed and equal to 500 (milliseconds). They specify red light overlap delays (the period of time when the signals on all 4 traffic lights signals are red).
Table 5.5: The parameters that control the timings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>This tag controls the timing of:</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer1.PRE</td>
<td>The green signal on traffic lights A1 and A2</td>
</tr>
<tr>
<td>timer2.PRE</td>
<td>The yellow signal on traffic lights A1 and A2</td>
</tr>
<tr>
<td>timer4.PRE</td>
<td>The green signal on traffic lights B1 and B2</td>
</tr>
<tr>
<td>timer5.PRE</td>
<td>The yellow signal on traffic lights B1 and B2</td>
</tr>
</tbody>
</table>

5.2.3 Subroutine Night_Mode

The subroutine “Night_Mode” contains the logic that controls the operation of the lights in the night mode. It has two rungs and the end rung. The sequencer SQO instruction can switch between two states specified in array seq_array2 as shown in Table 5.6. The sequencer switches every 0.5 seconds, the delay is controlled by timer14.

Table 5.6: Elements of seq_array2

<table>
<thead>
<tr>
<th>State</th>
<th>Address</th>
<th>Binary Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Seq_array[1]</td>
<td>0000_0000_0000_0000_0000_0000_0101_0000</td>
</tr>
<tr>
<td>S2</td>
<td>Seq_array[2]</td>
<td>0000_0000_0000_0000_0000_0000_1010_0000</td>
</tr>
</tbody>
</table>

5.3 System Setup for the Remote Access Example

Two personal computers (PC’s) connected to the Internet are needed for the demonstration of remote control and monitoring approach. The PC that is connected
to the ControlLogix 5550 via Ethernet LAN is called “Server Station”. The Server Station provides the required data to all machines that connect to it. The Server Station has following software: RSLinx, RSLogix 5000, and Internet Information Services (IIS). Both RSLinx and RSLogix 5000 should be configured as explained in the Chapter 3.

The other personal computer used that connects to the server and provides an operator interface for the user is called “Client Station”. It can be any personal computer connected to the Internet. The Client Station must have a web browser installed such as Internet Explorer or Netscape Navigator.

IIS is a web server that runs on Windows NT and Windows 2000 operating systems. It supports Active Server Pages (ASP) technology that allows imbedding applications in web pages that modify content sent back to users [15]. Like a standard HTML file, an ASP file can contain HTML tags that will be interpreted by a standard web browser. The additional feature of ASP – the ability to include server side script, like Visual Basic script or JavaScript – can be used to create dynamic Web applications.

The base of this demonstration is composed of two ASP files that contain Visual Basic script (VBScript), which is a simplified version of the Visual Basic programming language. VBScript applications use OPC (overview of OPC standard was given in the Chapter 3) Automation Interface to access data in the ControlLogix PLC.
5.4 Software Interactions

Figure 5.5 shows how the software products installed on both the client and the server interact with each other and with the PLC. It can be seen that ASP (VBScript application) communicates to the RSLinx OPC Server through the specified OPC Automation interface, which provides ASP with access to the data in the PLC.
5.5 Creating an OPC Automation Client

Two ASP file were created: parameters.asp – to view or modify the parameters of the traffic lights system and monitor.asp – to monitor the current state of the traffic lights. The full source codes of these files are attached in Appendix C. Both files contain a Visual Basic Scripting code that implements an OPC client. It communicates to the RSLinx OPC server through the specified OPC Automation interface. Only the parts that relate to the OPC Automation interface are shown in the explanations below [16].

The OPC Automation objects are organized as shown in Figure 5.6 [16]. The structure of an OPC client is shown in Figure 5.7 [16].

![Diagram of OPC Automation objects](image-url)

Figure 5.6. OPC Automation objects
Figure 5.8 shows the sample VBScript code, which is a fragment of the file `monitor.asp`, that reads a value of the tag `Local:1:0.Data` from the PLC memory into a Visual Basic variable.

```vbscript
' Declaring Variables

Dim objServer  ' OPCServer Object
Dim objGroup   ' OPCGroup Object
Dim strDataAddress  ' String variable = "Local:1:0.Data"
Dim sItemID     ' String variable. Item ID for the tag
Dim objItem     ' OPCItem Object
Dim vItem       ' The value of the tag "Local:1:0.Data" is read into this variable

Set objServer = Server.CreateObject("opc.automation.1")
If objServer Is Nothing Then
  response.write("Could not create an OPC Server")
  response.end
End If

' Creating OPCServer Object, connecting to the OPC Server and adding OPCGroup Object
```
objServer.Connect "RSLinx OPC Server"

set objGroup = objServer.OPCGroups.Add("WebGroup1")
objGroup.IsSubscribed = False

'Creating OPC Item

strDataAddress = "Local:1:0.Data"
sItemID = ":[topic2]" & strDataAddress

Set objItem = objGroup.OPCItems.AddItem(sItemID, 1)

If objItem Is Nothing Then
    response.write "Could not create the OPC item: " & sItemID & "<p>"
    response.end
end if

'Connecting to RSLinx OPC Server
'Adding OPCGroup Object
'Disable server callback

'Creating OPC Item

strDataAddress = "Local:1:0.Data"
sItemID = ":[topic2]" & strDataAddress

Set objItem = objGroup.OPCItems.AddItem(sItemID, 1)

If objItem Is Nothing Then
    response.write "Could not create the OPC item: " & sItemID & "<p>"
    response.end
end if

'Connecting to RSLinx OPC
'Adding OPCGroup Object
'Disable server callback

Figure 5.8. Sample of the code that reads the tag value from the PLC memory

5.6 Web Interface

The web page based application consists of two web pages. One of the pages is titled “View/Change Parameters”. It is available at http://imaev.com/parameters.asp. The other Web page is titled “Traffic Lights Monitoring”. It is available at http://imaev.com/monitor.asp.

5.6.1 Page “View/Change Parameters”

The web page “View/Change Parameters” allows viewing and changing the parameters of the Traffic Lights system. The screen shot of this page is shown in
Figure 5.8. On load the page displays the current timings of the traffic signals and it tells the operator whether the lights are in the day or in the night mode by reading the data from the PLC.

![Screenshot of the Web page “View/Change Parameters”](image)

Figure 5.8. Screenshot of the Web page “View/Change Parameters”

After the page has loaded you may specify the following system parameters:

- The timing of the green signal on traffic light A (in seconds).
- The timing of the yellow signal on traffic light A (in seconds).
• The timing of the green signal on traffic light B (in seconds).
• The timing of the yellow signal on traffic light B (in seconds).
• Whether the system operates is in the day mode or in the night mode.

5.6.2 Page “Traffic Lights Monitoring”

The screen shot of the page web page “Traffic Lights Monitoring” is shown in the Figure 5.9.

![Traffic Lights Monitoring](image_url)

Figure 5.9. Screenshot of the Web page “Traffic Lights Monitoring”
The page enables the operator to monitor the states of all 4 traffic lights. It reads the current state of the traffic lights system and displays it in a graphical form.

5.7 Concluding remarks

An Internet based remote access control concept and implementation was presented in this Chapter. An OPC Automation interface, which provides software connectivity between server side Visual Basic scripting application and the programmable logic controller, proved to be a very powerful tool in creating remote access applications.

The system presented has the following main features:

- Human Machine Interface provides graphic displays, facilitates user input.
- No special software is required on the server side.
- No dependence on third party software that may be costly and may require license for each client machine connecting to a server.

Apart from features listed above the system suffers from some disadvantages. The major disadvantage is that the monitoring of the process is slow. The bottleneck of the system is the refresh rate of the web pages displayed in an Internet browser. The default refresh rate is set to 3 seconds. It is impractical to make the refresh rate lesser than 3 seconds, because a certain time is needed for the page to load (1 to 2 seconds depending on the client’s machine Internet connection). Creating a client-
server application – client side Visual Basic application exchanging data with Visual Basic application on the server side – can solve this problem. This would require installation of VB client on each client machine but would allow faster processes to be controlled and monitored over the Internet.

Embedding ActiveX controls into the Web pages could also greatly enhance the graphical interface of the system. ActiveX controls are interactive objects that can be contained inside a Web page. ActiveX is based on Component Object Model (COM), which is Microsoft’s open standard that specifies how software components interact with each other [17]. With ActiveX controls complex objects can be added to the Web interface such as data trends, animation, etc.
6. CONCLUSIONS

In this thesis we have presented design and implementation of the programmable logic controller lab. We discussed the software and hardware configuration of the lab, introduced the PLC training lab exercises and demonstrated an approach to Internet based monitoring and control of a process.

Six lab exercises were developed for the PLC training laboratory course. These were aimed to give students an opportunity to learn and experiment on real PLC systems. Students have been using these exercises for two quarters so far and, as the practice showed, they proved to be very helpful, assisting students in their learning of the PLC subject.

Future plans for the lab include creating more advanced lab exercises that will utilize unexplored features of the ControlLogix system, such as PID control and some communication features. For example the communication capability of the PLC can be used to control a process that requires several PLC’s to collaborate and share data.

The method of remote access to the PLC discussed in this document has many potential practical uses. It can be used to turn on and off various devices, to control PID processes, etc. – the utilizations of the method are vast and boundless.
BIBLIOGRAPHY


A. LAB EXERCISES

PLC Lab 1
BIT INSTRUCTIONS

❖ Objective
At the completion of this lab you should have understanding of concepts of ladder logic and will be able to create ladder logic program using RSLogix 5000 software. You will be able to:
- Create tags
- Understand bit instructions
- Understand I/O addressing
- Understand how to make edits off-line
- Understand how to download a program to the ControlLogix 5000

Instructions learned: Examine if Closed (XIC), Examine if Open (XIO) and Output Energize (OTE).

❖ Example problem
Let’s write a ladder diagram program using RSLogix 5000 software that accomplishes the following:

1. A push button (push button DI4 on the trainer) provides master power by activating an internal relay and the relay stays active after the button is released.
2. A second push button (push button DI5 on the trainer) can be activated to turn off the internal relay
3. The motor (simulated by trainer light DO0) is ON when the master power is ON and motor is OFF when the system is not powered.
I/O Addressing

As described in [16], I/O information is presented as a structure of multiple fields, which depend on the specific features of the I/O module. The name of the structure is based on the location of the I/O module in the system. Each tag is automatically created when you configure the I/O module through programming software. The I/O module configuration will not be discussed in this lab. Instead we are going to open already pre-configured project to save time. The tagname follows this format:

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

For example:

6.1.1.1 **Local:0:1.Data.4** addresses input push button DI4 on a trainer.

Local = Local chassis  
0 = Slot 0  
O = Output  
Data = Data. Specific data from I/O module; depends on what type of data the module can store.  
4 = we are addressing bit 4 of 32-bit word.

Ladder logic diagram

The ladder logic diagram for our example is provided below:

Notice that we are using an internal utility relay in this example. You can use the contacts of these relays as many times as required. Here they are used twice to simulate a relay with 2 sets of contacts. Remember, these relays do not physically exist in the PLC but rather they are bits in a register that you can use to simulate a relay.
Let’s see what happens as the program being scanned. Initially ‘Master ON’ button is OFF, ‘Master OFF’ button is OFF, ‘Internal Relay’ is FALSE (i.e. open) and, therefore, the ‘Motor’ is OFF.

When we activate ‘Master ON’ button the normally opened contact becomes closed and now there is true logic path from left to right. Therefore ‘Internal Relay’ becomes TRUE and turns ON the motor. Notice that even when the ‘Master ON’ becomes false (push button DI4 released) there is still a path of true logic from left to right. This is why we used an internal relay on the input side of the rung in parallel with normally open contact.

When we activate ‘Master OFF’ button the normally closed contact becomes open and now there is no true logic path from left to right. As a result ‘Internal Relay’ becomes FALSE and turns OFF the motor. When ‘Master OFF’ button is released the logic path from left to right will still be closed provided that ‘Master ON’ button is not activated.

At this point the logic will appear the same as in our initial step and the logic will repeat as shown above.

Procedure:

Let’s develop our program using RSLogix 5000 software.

Note: To get more information on using RSLogix 5000 software refer to “Getting Started with RSLogix 5000 Software”.

1. Start RSLogix 5000. Double mouse click on the RSLogix 5000 icon on the desktop to start the ladder logic editor.

2. Open the existing project named CL.ACD. For more information see “Opening an Existing Project”. The file CL.ACD has all necessary configuration, which means communication and I/O modules are already configured for you.

3. Select Save As... under file menu and save the project under name ‘lab1.ACD’

4. Right click on the MainRoutine in the Project/Controller View and choose open. RSLogix 5000 software opens the Ladder Editor Routine Window. If the ladder routine is empty, RSLogix 5000 automatically adds the first rung.

5. Add Examine On instruction to the rung. From the Ladder Instruction Toolbar click on the tab Bit, select Examine On instruction and drag it into
the ladder editor routine view. The instruction is added to the rung or branch in which you chose to put it.

Double click on the ? in the area above the instruction to open an entry field that will allow you to select an available tag addresses.

6. Click down arrow to select an available tag addresses. You should see structures for module-defined tags. These tags where created automatically for each I/O module when the system was configured.

7. Click on the + sign in front of the **Local:0:I** entry (the 1756-IB16D DC input module in slot 0).

8. Click on the **Local:0:I.Data** tag under the **Local:0:I** structure.

9. Click on the down arrow next to **Local:0:I.Data** and select ‘4” to reference bit ‘4’ of the tag (i.e. push button DI4).

10. Add Examine Off instruction in serial connection with the first instruction. Reference the instruction to the **Local:0:I.Data.5** tag (i.e. push button DI5).
11. Select the **Examine On** instruction and click on **Create a Branch** to create a parallel logic path. Drag the right end of the branch to the connection point on the left side of the instruction.

12. We need to create a tag for ‘Internal Relay’. To do that: right click on the Controller Tags folder in the Controller/Project View, and choose Edit Tags. The tag editor appears. At this point only I/O module structures appear in the editor.

13. In the create tag row, indicated by an asterisk (*), enter the name of the tag in the Tag Name column.

14. Enter the tag data type: BOOL. Enter a description of the tag, if desired. Press Enter.

15. To go back to the program right click on the **MainRoutine** in the Project/Controller View and choose open.
16. Click and hold mouse on the **Examine On** icon in the toolbar. Drag the mouse pointer and the **Examine On** until it is on the top of the blue line of the branch and a green dot appears on the blue line of the branch.

17. Double click on the ? in the area above the instruction, then click the down arrow and select ‘relay’ tag. Press enter.

18. Add the **OTE** (Output Energize) instruction. Assign the ‘relay’ tag to the instruction. Your first rung is complete.

19. Click Add Rung icon in the toolbox. The new rung will appear. Add instructions to the rung as shown below. Input tag: **relay**. Output tag: **Local:1:O.Data.0**.

20. You may want to add operand description by right clicking on instruction and selecting ‘Edit Main Operand Description’.
21. After you enter descriptions your program may look like this:

22. Right click on the Routine tab and choose Verify Routine. Any errors in your rung will be sent to the Results window.

23. Make sure controller is in REM (remote) mode. Turning the key on the ControlLogix 5550 processor changes the mode.

24. In the Online Toolbox click the down arrow on the right of Offline icon and select download.

25. After the program is downloaded to the processor set the controller to the run mode by either turning the key on the processor or by selecting Run Mode from the Online Toolbox.

26. Push the buttons DI4 (Master ON) and DI5(Master OFF) to see whether the motor (light DO0) is ON or OFF. Notice that the instructions on the screen change color depending on if they are true or false.

You have successfully created your first ladder logic program using RSLogix 5000 software.
PLC Lab 2
LATCH commands, and TIMERS

✧ Objective
At the completion of this lab you should be able to use latch commands, and timer instructions.

✧ About Latches
One of the available output functions is the latch (L) command. This command causes the output to be energized and remain energized until the unlatch (U) command is activate. The address of the latch and unlatch command must be the same. The use of this command is illustrated below in the timer example.

✧ About Timers
Timers control operations based on time. Let’s consider Timer On Delay (TON). TON accumulates time when the instruction is enabled (rung-condition-in is true). It simply delays x-seconds before turning on DN bit, which indicates that the ‘waiting time’ is over.

To use a timer you will need to enter operands, which are: Timer, Preset, Accum as can be seen on the picture above. You will need to create a tag of data type ‘TIMER’ (it is a predefined data type). Timer tag is a structure, which has the following fields:
### Mnemonic Data Description

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.EN</td>
<td>BOOL</td>
<td>The enable bit indicates that the TON instruction is enabled.</td>
</tr>
<tr>
<td>.TT</td>
<td>BOOL</td>
<td>The timing bit indicates that a timing operation is in process.</td>
</tr>
<tr>
<td>.DN</td>
<td>BOOL</td>
<td>The done bit indicates that .ACC &gt;= .PRE.</td>
</tr>
<tr>
<td>.PRE</td>
<td>DINT</td>
<td>The preset value specifies the value (1 msec units) that the accumulated value must reach before the instruction sets the .DN bit.</td>
</tr>
<tr>
<td>.ACC</td>
<td>DINT</td>
<td>The accumulated value specifies the number of milliseconds that have elapsed since the TON instruction was enabled.</td>
</tr>
</tbody>
</table>

Note: above table is reproduced from the RSLogix 5000 on-line help

**TON instruction operands:**

- **Timer On Delay**
- **Timer**
- **Preset**
- **Accum**

Timer: Enter the name of the tag (which should be of structure data type ‘TIMER’) that you created.

Preset: How long to delay (accumulate time) (in thousandths of a second)

Accum: Total msec the timer has counted. Initial value is typically 0.

Note: For more information refer to RSLogix 5000 on-line help.

---

### Example Problem

Write a ladder logic program that accomplishes the following:
Activating trainer push button Data 0 causes output light D03 to illuminate. The light stays on for 5 seconds. Then the light goes out. The process can then be repeated.
Ladder logic diagram

The ladder logic diagram is provided below

Instructions \textit{Output Latch (OTL)} and \textit{Output Unlatch (OUT)} are used in above ladder diagram.

\textbf{Output latch:}

\textbf{Output unlatch:}

When enabled, the OTL instruction sets (latches) the data bit. The data bit remains set until it is cleared, typically by an OTU instruction. When disabled, the OTL instruction does not change the status of the data bit.

Let’s see what happens as the program being scanned. When push button D10 is activated TON becomes enabled and output light DO3 becomes latched. Timer starts to accumulate time. However timer is enabled only when rung-condition-in is TRUE. To maintain rung-condition-in TRUE after push D10 is released we use hold-in branch in parallel with \textit{timer1.EN} bit, thus keeping timer1 enabled when pushbutton D10 is released.
After 5000 milliseconds (5 seconds) \textit{timer1.ACC} reaches preset value \((\text{ACC} = \text{.PRE})\) and timer sets its \textit{timer1.DN} bit. Done bit indicates that the 5 seconds have passed \((\text{i.e. } \text{ACC} >\text{= .PRE})\). Rung 1 condition-in becomes TRUE and we unlatch output light DO3 and reset the timer with Reset (RES) instruction. Reset instruction resets the \text{.ACC} value, \text{.DN} bit and \text{.TT} bit.

\textbf{Procedure:}

1. Create a tag for timer. Right click on Controller Tags in Controller/Project View and choose New Tag…. The New Tag window appears.
2. Enter the name of the tag: \text{timer1}
   Select data type: \text{TIMER}
   Enter the description of the tag if desired.

\begin{center}
\includegraphics[width=0.8\textwidth]{NewTag.png}
\end{center}

Click OK to close New Tag dialog window.

3. Now enter your ladder logic program. TON instruction can be entered as follows: From the Ladder Instruction Toolbar click on the tab Timer/Counter, select TON instruction and drag it into the ladder editor routine view. The instruction is added to the rung or branch in which you chose to put it.
4. TON instruction. Double click on the ? in the area to the right of Timer field, click down arrow and select tag ‘timer1’. Enter preset value: 5000 (preset value is in milliseconds).

5. Download the program to controller, run it and test how it works.
PLC Lab 3
COUNTERS

❖ Objective
At the completion of this lab you will be able to use counter instructions.

❖ About Counters
Counters control operations based on the number of events. Three available counter instructions are: Count Up (CTU), Count Down (CTD), Count Up and Count Down in function block (CTUD).

Let’s first examine the Count Up (CTU) instruction.

The instruction increments its \( ACC \) value by one each time the rung-condition-in goes from FALSE to TRUE.

As with TON instruction you will need to create a tag for your counter. The tag should be of predefined data type ‘COUNTER’. Below is a table (reproduced from the RSLogix 5000 on-line help) that explains counter structure fields.
COUNTER Structure:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.CU</td>
<td>BOOL</td>
<td>The count up enable bit indicates that the CTU instruction is enabled.</td>
</tr>
<tr>
<td>.DN</td>
<td>BOOL</td>
<td>The done bit indicates that .ACC &gt; .PRE.</td>
</tr>
<tr>
<td>.OV</td>
<td>BOOL</td>
<td>The overflow bit indicates that the counter exceeded the upper limit of 2,147,483,647. The counter then rolls over to -2,147,483,648 and begins counting up again.</td>
</tr>
<tr>
<td>.PRE</td>
<td>DINT</td>
<td>The preset value specifies the value, which the accumulated value must reach before the instruction sets the .DN bit.</td>
</tr>
<tr>
<td>.ACC</td>
<td>DINT</td>
<td>The accumulated value specifies the number of transitions the instruction has counted.</td>
</tr>
</tbody>
</table>

The Count Down (CTD) instruction is similar to the CTU instruction. The only difference is that the CTD decrements its .ACC value by one each time the rung-condition-in goes from FALSE to TRUE. It is typically used with a CTU instruction that references the same counter structure.

To learn more about counter instructions refer to the RSLogix 5000 on-line help.

**Example Problem**

Design a PLC program for a parking garage, which alerts an attendant when no more parking spaces are available. For demonstration, assume that there are 10 parking spaces. As cars leave the garage, the counter should count down and turn off the indicating light.

The ladder logic diagram is provided below:
Push button DIO (corresponding tag: car_in) – car enter the garage.
Push button DII (corresponding tag: car_out) – car leave the garage.

Let’s see what happens as the program being scanned. When we activate the button DIO the CTU instruction increments the value of counter1.ACC by one. When we activate the button DII the CTD instruction decrements the value of counter1.ACC by one. When the value of counter1.ACC reaches preset value, which is equal to 10, rung 2 condition-in becomes true and the indicating light turns on.

Procedure:

6. Create an alias for tag Local:0:1.Data.0 (it addresses the trainer push button DIO). Right click on Controller Tags in Controller/Project View and choose New Tag…. The New Tag window appears.

7. Enter the name of the tag: car_in
   Select the tag type: alias
   Select the alias for the tag from the drop down list: Local:0:1.Data.0
   Click OK to close the New Tag Window.

8. Similarly create an alias car_out for tag Local:0:1.Data.1 (it addresses the trainer push button DII).

9. Create an alias alert for tag Local:1:0.Data.4 (it addresses the trainer light DO4)

10. Create a tag for counter. Right click on Controller Tags in Controller/Project View and choose New Tag…. The New Tag window appears.

11. Enter the name of the tag: counter1
    Select data type: COUNTER
    Click OK to close the New Tag Window.
12. Now enter your ladder logic program. CTU and CTD instructions can be entered as follows: From the Ladder Instruction Toolbar click on the tab Timer/Counter, select CTU or CTD instruction and drag it into the ladder editor routine view. The instruction is added to the rung or branch in which you chose to put it.

13. CTU instruction. Double click on the ? in the area to the right of Counter field, click down arrow and select tag `counter1`. Set preset value to 10.

14. CTD instruction. Double click on the ? in the area to the right of Counter field, click down arrow and select tag `counter1`. Preset value should already be set to 10.

15. Download your program to the PLC.
PLC Lab 4
SEQUENCER

❖ Objective
At the completion of this lab you will be able to use sequencer and move instructions

❖ Move (MOV) instruction
MOV moves data from the source to the destination; it’s as simple as that. The source and the destination could be a number or it could be a tag. For more information on the MOV command refer to RSLogix 5000 on-line help.

❖ Sequencer Output (SQO) instruction
In this lab we will consider Sequencer Output (SQO) instruction. SQO instruction is used to set output conditions for each step in a sequence.

<table>
<thead>
<tr>
<th>SQO</th>
<th>&lt;EN&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequencer Output</td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td>?</td>
</tr>
<tr>
<td>Mask</td>
<td>?</td>
</tr>
<tr>
<td>Dest</td>
<td>?</td>
</tr>
<tr>
<td>Control</td>
<td>?</td>
</tr>
<tr>
<td>Length</td>
<td>?</td>
</tr>
<tr>
<td>Position</td>
<td>?</td>
</tr>
</tbody>
</table>

When enabled, the SQO instruction increments the position, moves the data at the position through the Mask, and stores the result in the Destination.

We will need to create special control structure for SQO instruction. The structure has data type CONTROL. The description of structure fields that can be used for SQO instruction is given next.

Also we will need to create an array also called sequencer table.

The use of the SQO instruction will be explained in an example later in this lab.
Operands:

<table>
<thead>
<tr>
<th>Operand</th>
<th>Type</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>DINT</td>
<td>array tag</td>
<td>Sequencer array Specify the first element of the sequencer array Do not use CONTROL.POS in the subscript</td>
</tr>
<tr>
<td>Mask</td>
<td>SINT INT DINT</td>
<td>tag immediate</td>
<td>Which bits to block or pass</td>
</tr>
<tr>
<td>Destination</td>
<td>DINT</td>
<td>tag</td>
<td>Output data from the sequencer array</td>
</tr>
<tr>
<td>Control</td>
<td>CONTROL</td>
<td>tag</td>
<td>Control structure for the operation Typically use the same CONTROL as the SQI and SQL instructions</td>
</tr>
<tr>
<td>Length</td>
<td>DINT</td>
<td>immediate</td>
<td>Number of elements in the Array (sequencer table) to output</td>
</tr>
<tr>
<td>Position</td>
<td>DINT</td>
<td>immediate</td>
<td>Current position in the array Initial value is typically 0</td>
</tr>
</tbody>
</table>

Note: above table was reproduced from the RSLogix 5000 on-line help

COUNTROL Structure:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.EN</td>
<td>BOOL</td>
<td>The enable bit indicates that the SQO instruction is enabled.</td>
</tr>
<tr>
<td>.DN</td>
<td>BOOL</td>
<td>The done bit is set when all the specified elements have been moved to the Destination.</td>
</tr>
<tr>
<td>.ER</td>
<td>BOOL</td>
<td>The error bit is set when .LEN ≤ 0, .POS &lt; 0, or .POS &gt; .LEN.</td>
</tr>
<tr>
<td>.LEN</td>
<td>DINT</td>
<td>The length specifies the number of steps in the sequencer array.</td>
</tr>
<tr>
<td>.POS</td>
<td>DINT</td>
<td>The position identifies the element that the controller is currently manipulating.</td>
</tr>
</tbody>
</table>

Note: above table was reproduced from the RSLogix 5000 on-line help.
Example Problem

The best way to illustrate the use of SQO instruction is by example. The example below will employ both timer and sequencer instructions.

Simulate a process that has 4 steps, each step takes 1 second.

Step 1. Outputs DO0, DO4, DO8 are activated. All other outputs are OFF.
Step 2. Outputs DO1, DO5, DO9 are activated. All other outputs are OFF.
Step 3. Outputs DO2, DO6, DO10 are activated. All other outputs are OFF.
Step 4. Outputs DO3, DO7, DO11 are activated. All other outputs are OFF.

After last step (step 4) the process is repeated starting with step 1.

The process is powered using a Master ON push button. Master OFF push button is provided to de-power the system.

Ladder logic diagram

The ladder logic diagram is provided below.

Let’s see what happens as the program being scanned.
When Master ON push button is activated Power (tag of data type BOOL) becomes TRUE and stays true until Master OFF push button is activated. The timer becomes enabled as a result.

After 1 second timer .DN bit becomes set and sequencer becomes enabled and it increments its position (.POS becomes 1) and moves data at the position through the Mask, and stores the result in the Destination (output word Local:1:0.Data). The timer resets (Rung 3) and then starts to accumulate time again (Rung 2). That completes the first step in our process.

Sequencer position is the position within sequencer array (sequencer data table) SQO_Array, which has 5 elements. For example position 3 points to the 32 bit word within an array at address SQO_Array[3]. The mask specifies which bits to block or pass. Prefix 16# in front of the mask value means that the value is represented in hexadecimal format (prefix 8# - octal format, prefix 2# - binary format, no prefix - decimal format). The mask value of '16#0000_0fff' specifies that we want to affect only 12 output bits (DO0 through DO11) and leave all other outputs unchanged.

At this point the logic will appear the same as in our initial step (except that sequencer position value will be different for each step) and the logic will repeat as shown above. If .POS > .LEN, the instruction wraps around to the beginning of the sequencer array and continues with .POS = 1.

If, at any time of the process, push button Master OFF is activated, Power becomes FALSE. In Rung 1 the normally closed contact enables two Move (MOV) instructions. First MOV instruction sets sequencer position value to zero and the second move instruction clears output bits (stops all devices).

In this example each step of the process takes the same amount of time. We have only one timer. That timer’s .DN bit triggers the sequencer. However, if each step takes different time to complete, the task becomes more complex because we will have to employ several timers with different time presets.

Procedure:


2. Enter the name of the tag: SQO_Array.

3. Click on the ... button on the left of Data Type field.
4. In Select Data Type Window enter the parameters as shown.

5. Click OK to close Select Data Type Window. Click OK to close New Tag Window.

6. Create a tag for timer. Name it Timer_1.

7. Create a tag named ‘Power’ of data type BOOL.

8. Create a tag named ‘SQO_Control’ of data type CONTROL.
9. Now we need to fill in values in sequencer table SQO_Array. Double click on Controller Tags in Controller/Project View. The Tags Window will appear. Select Monitor Tags tab on the bottom of the window.

![Tags Window]

- Make sure that the controller scope is selected.
- Make sure the Monitor Tags tab is selected.

10. Enter fields in sequencer table as shown:

![Fields Window]

- Select Binary to enter data in binary format.
- Enter values for sequencer table as shown.
11. Enter the ladder logic program as shown. SQO instruction can be found in the Ladder Instruction Toolbar under tab Sequencer. MOV instruction can be found in the Ladder Instruction Toolbar under tab Move/Logical.

12. Enter parameters for sequencer operands as shown:

13. Save your program and download it to controller.
PLC Lab 5
ANALOG I/O
Math and Compare Instructions

❖ Objective

At the completion of this lab you will learn how to program analog I/O devices. You will also learn several new instructions such as Math and Compare.

❖ Analog I/O

Along with DC input and DC output modules the ControlLogix trainer includes analog input and analog output modules. Analog input module is wired to the potentiometers AI0 and AI1. Analog output module is wired to the voltmeters AO0 and AO1.

❖ Math and Compare Instructions

Information on these instructions can be found in RSLogix 5000 on-line help. The use of this instructions will be demonstrated in an example below.

Example Problem

We are going to write a ladder logic program that uses analog I/O. Lets consider a simple program that accomplishes the following:

1. We want to be able to set output voltage on voltmeter AI0 with potentiometer AO0. In other words the voltage on potentiometer AI0 should be equal to the voltage on voltmeter AO0.

2. We want to be able to increment the voltage on voltmeter AO1 with push button DI0 and decrement the voltage on voltmeter AO1 with push button DI1. The voltage should be incremented or decremented by some small value.
Ladder Logic Diagram:

The analog input module and the analog output module each have 6 channels. Only two of these channels are used in the trainer. Channel 0 is wired to potentiometer A10 and voltmeter AO0. Channel 1 is wired to potentiometer A11 and voltmeter AO1.

'Local:3:1.Ch0Data' represents analog input word for channel 0, which is wired to the potentiometer A10. The data type of 'Local:3:1.Ch0Data' is 'REAL'.

'Local:4:0.Ch0Data' represents analog output word for channel 1, which is wired to the potentiometer A11. The data type of 'Local:4:0.Ch0Data' is 'REAL'.

'Local:4:0.Ch1Data' represents analog output word for channel 1, which is wired to the voltmeter AO1. The data type of 'Local:4:0.Ch1Data' is 'REAL'.

In rung 0 we simply move input data from the potentiometer A10 to the voltmeter AO0. That takes care of the first part of our problem.

Rungs 1 and 2 take care of the second part of the problem. Using ‘Less Than or Equal To’ (LEQ) instruction we make sure that the value of ‘Local:4:0.Ch1Data’ that represents output to voltmeter is less than voltmeter’s maximum desired value, which in our case is equal to 9.99875. Similarly using ‘Greater Than or Equal To’ (GEQ) instruction we check if the value of ‘Local:4:0.Ch1Data’ is greater than voltmeter’s minimum desired value.
With Addition (ADD) and Subtract (SUB) instructions we increment or decrement the value of ‘Local:4:0.Ch1Data’.

Procedure:

1. Open the project CL.ACD. This project already contains all the necessary configurations for the I/O and communication.

2. Save the project under different name.

3. Download the following program to the controller.
PLC Lab 6
CREATING OPERATOR INTERFACE

Objective

The objective of this lab is to demonstrate an example on how to develop Human Machine Interface (HMI) application using RSView32.

About HMI

The early programmable logic controllers interfaced with the operator via push buttons and switches for control and lamps for indication. Owing to the introduction of the personal computer (PC) in 1980’s the development of a computer-based interface to the operator had begun. The tool for computer-based interface through which operators control and receive information from a system or a process is called Human Machine Interface.

RSView32 is an HMI software product for Windows. RSView32 is a software package that is positioned on top of the PLC to which it is interfaced. With RSView32 you can represent the process in a graphical way for easy monitoring and control.

Example HMI Application

Let’s demonstrate the creation of HMI for the example that was presented in the Lab 3.

Design a PLC program for a parking garage, which alerts an attendant when no more parking spaces are available. For demonstration, assume that there are 10 parking spaces. As cars leave the garage, the counter should count down and turn off the indicating light.
First, we need program the PLC to perform the desired control function. Second, we need create a graphical display of our process. We will use the ladder logic program that we developed in the Lab 3. The program is reproduced below.

![Ladder Logic Diagram](image)

Download the program to the controller and proceed to the next step – creating an operator interface for the parking garage.

With the interface we’re about to create an operator will be able to see how many cars are currently in the garage and if the garage is full the indicating light will be turned on. All that information will be displayed on the computer screen.

RSView32 Procedure:

1. Start RSView32. Click on Start button then select Programs>Rockwell Software>RSView32 Works.

2. Open the existing project named DefaultProject.rsv. This project contains all the necessary configurations.

3. Now we need to create tags in RSView that we are going to use. All in all we need to create 2 tags. One will be associated with the RSLogix 5000 tag `counter1.ACC` (the current number of cars in the garage). Another will be associated with the RSLogix 5000 tag `alert (Local:1:O.Data.4)` that indicates that the number of cars in the garage is reached 10.

4. To create a tag, in the project window open the system folder as shown in Figure below:
5. Double click on the Tag Database. The following window will be displayed (you may need to maximize it).

6. Create a tag that corresponds to the number of cars in the garage. Enter tag options as described below.

   **Tag:**
   - **Name:** car_enter
   - **Type:** analog

   **Data Source:**
   - **Type:** Device
   - **Node Name:** Click on ... button and select - OPC_Connection
Address: Click on … button. OPC Address Browser window will be displayed. Browse to the tag `counter1.ACC` as shown below. Click OK to close the OPC Address Browser window.

7. Click on Accept to accept your changes. The Tag Database window will look as shown below.

Click on Next.

8. Create a tag that corresponds to the indication light. Enter the tag options as described below.
Tag:
   Name: alert_tag
   Type: digital

Data Source:
   Type: Device
   Node Name: Click on … button and select - OPC_Connection
   Address: Click on … button. OPC Address Browser window will be
displayed. Browse to the tag alert exactly as shown below. Click OK
to close the OPC Address Browser window.

9. Click on Accept to accept your changes. Click on Close to close the Tag
Database window.

10. In the Project window close the System folder and open the Graphics folder as
shown.
11. Double click on Display. The Display window will appear.

Take your time and get familiar with the toolbars.
12. First we draw the rectangle that is going to represent the light that indicates that the garage is full. When the rectangular is sketched right click on it, select Attributes and make sure that attribute Solid is selected.

13. Now we need to add animation to our indicating light. Right click on the rectangular, select Animation and click on Color. The Animation dialog box will be displayed as shown.
14. Click on Tags..., in the Tags window (shown below) select tag *alert_tag* that we created and click on OK to close the dialog box.

15. Enter the options that specify the color of the rectangular when the indication light is turned off (value = 0).
16. Similarly enter the options that specify the color of the rectangular when the indication light is turned on (value = 1).

![Animation window](image)

Click here to change the fill color.

Note that we have selected the option of flashing light (red-yellow) for *alert_tag = 1*. Click on Apply and then on Close to submit the changes and close the Animation window.

17. Next we want to add the Numeric Display to display the number of cars in the garage. To add the Numeric Display select the corresponding icon from the
Toolbar and draw a little rectangular area while holding your left mouse button. The Display Window will pop up. Enter the parameters as shown below.

![Numeric Display Window](image)

Click OK to close the Numeric Display window.

18. Add some text to your sketch using Text button on the Toolbar so your Graphics Display will look similar to one shown below.

![Sketch with Text and Numeric Display](image)
19. Run the project by clicking on Test Run button. Change the number of cars in the garage by activating push buttons DI0 and DI1. The Numeric Display should show the current number of cars in the garage. When the number of cars reaches 10 the indicating light should start flashing red and yellow until the number becomes less then 10.

\[\begin{align*}
\text{Conclusion} \\
\text{In this lab we presented a simple example of HMI. Other kinds of animation are available such as position animation; it is when object’s position on the computer screen is defined by the value of some tag. With RSView you can create complex operator interfaces, for example an HMI for a car assembly line.}
\end{align*}\]
B. LADDER LOGIC PROGRAM

B.1 MainRoutine

Diagram illustrating the flow of logic for MainRoutine, showing the transition between Day_Mode and Night_Mode based on theMode.
B.2 Routine Day Mode

Rungs 0 through 5:
Rungs 6 through 11:

Rung 12 and the end rung:
B.3 Routine Night Mode
C. ASP CODE

File parameters.asp

<% Option Explicit %>
<!--
parameters.asp
This Web page uses OPC to read/write data to the
ControlLogix 5550 PLC. On load it reads the process
parameters from the PLC.
When the user selects the his parameters for the process and
he clicks on 'Submit Changes'
the program writes data to the PLC.
-->

<HTML>
<HEAD>
<title>Parameters</title>
<base target="_self">
</HEAD>
<BODY link="#FFFFFF" vlink="#0000FF" alink="#FF0000" bgproperties="fixed">
<table border="0" width="500" cellspacing="0">
<tr>
<td width="500" bgcolor="#FFFFCC">
<h1 align="center"><b>View / Change Parameters</b></h1>
</td>
</tr>
</table>
<%'
'Declaring Variables

    dim objServer   'OPCServer Object
    dim objGroup    'OPCGroup Object
    dim strMode     'variable that specifies the mode of operation (day/night)
    dim strGreenA   'timing of the green signal on traffic lights A1 and A2
    dim strYellowA  'timing of the yellow signal on traffic lights A1 and A2
    dim strRedA     'timing of the red signal on traffic lights A1 and A2

-->
dim strGreenB  'timing of the green signal on traffic lights B1 and B2
dim strYellowB  'timing of the yellow signal on traffic lights B1 and B2
dim strRedB  'timing of the red signal on traffic lights B1 and B2
dim AddressMode  ' = "theMode"
dim ItemIDMode  'item ID for the tag
dim objMode  'OPCItem Object
dim AddressGreenA  ' = "timer1.PRE"
dim ItemIDGreenA  'item ID for the tag
dim objGreenA  'OPCItem Object
dim AddressYellowA  ' = "timer2.PRE"
dim ItemIDYellowA  'item ID for the tag
dim objYellowA  'OPCItem Object
dim AddressGreenB  ' = "timer4.PRE"
dim ItemIDGreenB  'item ID for the tag
dim objGreenB  'OPCItem Object
dim AddressYellowB  ' = "timer5.PRE"
dim ItemIDYellowB  'item ID for the tag
dim objYellowB  'OPCItem Object

'Creating OPCServer Object
set objServer = Server.CreateObject("opc.automation.1")
if objServer is nothing then
    response.write("Could not create an OPC Server")
    response.end
end if

'Connecting to the OPCServer
objServer.Connect "RSLinx OPC Server"

'Adding OPCGroup Object
set objGroup = objServer.OPCGroups.Add("WebGroup1")

'Disable server callback
objGroup.IsSubscribed = False

' Creating OPC items
AddressMode = "theMode"
ItemIDMode = "[topic2]" & AddressMode
Set objMode = objGroup.OPCItems.AddItem(ItemIDMode, 1)

AddressGreenA = "timer1.PRE"
ItemIDGreenA = "[topic2]" & AddressGreenA
Set objGreenA = objGroup.OPCItems.AddItem(ItemIDGreenA, 1)

AddressYellowA = "timer2.PRE"
ItemIDYellowA = "[topic2]" & AddressYellowA
Set objYellowA = objGroup.OPCItems.AddItem(ItemIDYellowA, 1)

AddressGreenB = "timer4.PRE"
ItemIDGreenB = "[topic2]" & AddressGreenB
Set objGreenB = objGroup.OPCItems.AddItem(ItemIDGreenB, 1)

AddressYellowB = "timer.S.PRE"
ItemIDYellowB = "[topic2]" & AddressYellowB
Set objYellowB = objGroup.OPCItems.AddItem(ItemIDYellowB, 1)

'Checking whether the OPC Items were created
If objMode Is Nothing Then
    response.write "Could not create the OPC item: &" & ItemIDMode & "<p>
    response.end
end if

If objGreenA Is Nothing Then
    response.write "Could not create the OPC item: &" & ItemIDGreenA & "<p>
    response.end
end if

If objYellowA Is Nothing Then
    response.write "Could not create the OPC item: &" & ItemIDYellowA & "<p>
    response.end
end if

If objGreenB Is Nothing Then
    response.write "Could not create the OPC item: &" & ItemIDGreenB & "<p>
    response.end
end if

If objYellowB Is Nothing Then
    response.write "Could not create the OPC item: &" & ItemIDYellowB & "<p>
    response.end
end if

'Reading/Writing data to the PLC
If Request.Form("isSubmitted") = "yes" Then

'if the user clicked on "Submit Changes" button: read the data
'that was entered and then write that data to the PLC

'reading the data from the form
    strMode = Request.Form("mode")
    strGreenA = Request.Form("GreenA")
    strYellowA = Request.Form("YellowA")
    strGreenB = Request.Form("GreenB")
    strYellowB = Request.Form("YellowB")

'writing the user data to the PLC
    If strMode = 1 Then
        objMode.write "1"
    else
        objMode.write "0"
    end if

    objGreenA.write strGreenA
```plaintext
objYellowA.write strYellowA
objGreenB.write strGreenB
objYellowB.write strYellowB

Else

'on the initial load of the page read all the parameters from the PLC

objGreenA.Read 1, strGreenA
objYellowA.Read 1, strYellowA
objGreenB.Read 1, strGreenB
objYellowB.Read 1, strYellowB
objMode.Read 1, strMode

End if

'Function "Cint" converts any number to the variant of subtype "Integer"

strGreenA = Cint(strGreenA)
strYellowA = Cint(strYellowA)
strGreenB = Cint(strGreenB)
strYellowB = Cint(strYellowB)

'Calculating the timings of the red signals for display on the web page

strRedA = strGreenB + strYellowB + 1000
strRedB = strGreenA + strYellowA + 1000
```

<font size=3 face="Times New Roman">This page displays the current timings of the traffic lights controlling the 4-way intersection (see the picture). To change the timings select the desired value for the green and yellow signals. The timings for the red signals will be changed accordingly. To switch to the night mode select the corresponding checkbox. Click on Submit Changes to submit your changes.</font>

```html
<form method="POST" action=change.asp>
<input type="hidden" name="isSubmitted" value="yes">
<table border="0" width="500">
  <tr>
    <td align="center">Current Timings of the Traffic Lights A1 and A2</td>
  </tr>
</table>
</form>
```
<td width="33%" bgcolor="#00AA00" height="19" >
<p align="center"><b><font color="#FFFFFF">GREEN</font></b></p>
</td>

<td width="33%" bgcolor="#FFFF00" height="19">
<p align="center">YELLOW</p>
</td>

<td width="34%" bgcolor="#FF0000" height="19">
<p align="center"><font color="#FFFFFF"><b>RED</b></font></p>
</td>

</tr>
<tr>
<td width="34%" height="22" bordercolor="#000080">
<p align="center"><select size="1" name="GreenA" style="font-weight: bold">
<option value="4000">
4 sec</option>
<option value="6000">
6 sec</option>
<option value="8000">
8 sec</option>
<option value="10000">
10 sec</option>
<option value="12000">
12 sec</option>
</select></p>
</td>
</tr>

<% 'Traffic lights A1 and A2: the VBScript code below allows this web 'page 'to "remember"the timings that were selected or, on initial 'load of the page, to display the current timing values of the 'system 'that were read from the PLC %>
<% if strGreenA = 4000 Then
response.write " selected"
end if
%>
4 sec</option>
<% if strGreenA = 6000 Then
response.write " selected"
end if
%>
6 sec</option>
<% if strGreenA = 8000 Then
response.write " selected"
end if
%>
8 sec</option>
<% if strGreenA = 10000 Then
response.write " selected"
end if
%>
10 sec</option>
<% if strGreenA = 12000 Then
response.write " selected"
end if
%>
12 sec</option>
<% if strGreenA = 14000 Then
%>
response.write " selected"
end if
%
>14 sec</option>
</select></td>

<td width="33%" height="22" bordercolor="#000080">
<p align="center">
<option value="1000"
<%
if strYellowA = 1000 Then
response.write " selected"
end if
%
>1 sec</option>
<option value="2000"
<%
if strYellowA = 2000 Then
response.write " selected"
end if
%
>2 sec</option>
<option value="3000"
<%
if strYellowA = 3000 Then
response.write " selected"
end if
%
>3 sec</option>
<option value="4000"
<%
if strYellowA = 4000 Then
response.write " selected"
end if
%
>4 sec</option>
</select></td>

<td width="33%" height="22" bordercolor="#000080">
<p align="center">
<b>
' display the timing of the red signal for the traffic lights A1
' and A2
'(this value was calculated)
response.write strRedA/1000 %>
sec </p> </b></td>
</tr>
</table>

<br>
<table border="0" width="500">
<tr>
<td>
Current Timings of the Traffic Lights B1 and B2

<table>
<thead>
<tr>
<th>Time</th>
<th>Green</th>
<th>Yellow</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Traffic lights B1 and B2: the VBScript code below allows this web page to "remember" the timings that were selected or, on initial load of the page, to display the current timing values of the system that were read from the PLC.

```vbscript
if strGreenB = 4000 Then
    response.write " selected"
end if

if strGreenB = 6000 Then
    response.write " selected"
end if

if strGreenB = 8000 Then
    response.write " selected"
end if

if strGreenB = 10000 Then
    response.write " selected"
end if
```

if strGreenB = 12000 Then
<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 sec</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2 sec</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3 sec</td>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>4 sec</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>12 sec</td>
<td>14000</td>
<td></td>
</tr>
<tr>
<td>14 sec</td>
<td>14000</td>
<td></td>
</tr>
</tbody>
</table>

`response.write strRedB/1000 %> sec`
<table width="500" border="1" bordercolorlight="#FFFFFF" bordercolordark="#FFFFFF">
    <tr>
        <td width="100%" colspan="3" bordercolor="#000080">
            <p align="center">
                <input type="checkbox" name="mode" value="1"
                %>
                        'making sure that the checkbox defined above is checked
                        'when the system is in the night mode or un-checked otherwise
                        
                        if strMode = 1 Then
                            Response.Write " checked=""checked"
                        end if
                        
                        <font size="3">Check this box to switch to the night mode</font>
            </td>
        </tr>
    </table>

    <table border="0" width="500">
        <tr>
            <td width="100%">
                <p align="center">
                    <input type="submit" value="Submit Changes" name="Submit"/>
                </td>
            </tr>
        </table>
    </form>

    <table border="0" width="700">
        <tr>
            <td width="100%" height="45">
                <p align="center">
                    'Displaying messages
                    
                    if strMode = 1 Then
                        'If the system is in the Night Mode
                    
                    <font size="3">The traffic lights are in the flashing yellow/red mode.</font> To switch back to the normal
                    <br> mode uncheck the corresponding checkbox and click on Change Parameters</b>.<\font>
                    
                    <% else if Request.Form("IsSubmitted") = "yes" Then
                        'If the system is in the Day Mode
                        'and the user parameters were submitted
                    %>
                        <font size="3">New timings was successfully written to the ControlLogix 5550</font>
controller!</font>

<% end if
end if %>
</td>
</tr>
<tr>
<td width="100%"><a href="auto.asp"><font color="#000080" face="Arial">
<br>Monitor the state of the traffic lights</a></font></td>
</tr>
</table>

File monitor.asp

<% Option Explicit %>

<!-- response.asp
This Web page displays the state of the traffic lights. It reads the data from the ControlLogix 5550 PLC using OPC Automation interface. The OPC Server is RSLinx. The page refreshes every 3 seconds.
-->

<HTML>
<HEAD>
<TITLE>Automatic Mode</TITLE>

<!-- Refresh rate for the page is defined here -->
<meta http-equiv="refresh" content="3;URL=auto.asp">
<base target="_self">
<meta name="Microsoft Border" content="none, default">

</HEAD>
<BODY>
<table border="0" width="500">
<tr>
<td width="100%" bgcolor="#FFFFCC">
<h1 align="center"><b>Traffic Lights Monitoring</b></h1>
</td>
</tr>
</table>

<p>
The page displays the current state of the traffic lights system. It refreshes automatically every 3 seconds. </p>
<% 
  dim objServer  'OPCServer Object
  dim objGroup  'OPCGroup Object
  dim strDataAddress  'String variable = "Local:1:0.Data"
  dim sItemID  'String variable. Item ID for the tag
  dim objItem  'OPCItem Object

  dim vItem 'The value of tag "Local:1:0.Data" is read into this variable

  set objServer = Server.CreateObject("opc.automation.1")
  if objServer is nothing then
    response.write("Could not create an OPC Server")
    response.end
  end if

  'Connecting to RSLinx OPC Server
  objServer.Connect "RSLinx OPC Server"

  'Adding OPCGroup Object
  set objGroup = objServer.OPCGroups.Add("WebGroup1")

  'Disable server callback
  objGroup.IsSubscribed = False

  'Creating OPC Item
  strDataAddress = "Local:1:0.Data"
  sItemID = "[topic2]" & strDataAddress
  Set objItem = objGroup.OPCItems.AddItem(sItemID, 1)
  If objItem Is Nothing Then
    response.write "Could not create the OPC item:"&ItemID&"<p>
    response.end
  end if

  'Reading the process data from the PLC into variable vItem
  'vItem will contain the value of the tag "Local:1:0.Data",
  'which is a 32-bit word.
  objItem.Read 1, vItem

  'The value of vItem is read as a decimal number.
  'Therefore we are translating this number to binary
  'and saving individual
  'bits in array D(32)
  Dim x
  Dim D(32)
  Dim N
x = Cint(vItem)
N = 0

'start translation to binary

Do
' Check whether it is 1 bit or 0 bit
If x Mod 2 Then
  D(N) = 1
Else
  D(N) = 0
End If

x = x \ 2
' Normal division  7/2 = 3.5
' Integer division  7\2 = 3

N = N + 1

If x < 1 Then Exit Do
Loop

%>

<table border="0" width="300" height="300" bordercolorlight="#FFFFFF"
bordercolordark="#FFFFFF" cellspacing="0" cellpadding="0">
<tr>
<td width="33%" height="100" bordercolorlight="#FFFFFF"
bordercolordark="#FFFFFF"> &nbsp; <b>
1</b> </td>
<td width="33%" align="center" height="100" bgcolor="#C0C0C0">
<p align="center"> &nbsp; </p> </td>
</tr>
<tr>
<td width="100%" align="center" &nbsp;>
</td>
</tr>
</table>

'The VBScript below specifies the colors of the desired table cells (12 cells in total)
'depending on the values of variables D(0) through D(11)

%>

<tr>
<td width="100%" align="center" %>
  if D(0)=1 Then
    response.write "bgcolor='#FF0000'"
  end if
  %>
</td>
</tr>
<tr>
<td width="100%" %>
  if D(4)=1 Then
    response.write "bgcolor='#FFFF00'"
  end if
  %>
</td>
<table>
<thead>
<tr>
<th></th>
<th>Light Bl</th>
<th>Light B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
&lt;table border="0" cellspacing="0" cellpadding="0" width="250"&gt;
  &lt;tr&gt;
    &lt;td width="29"&gt;
      &lt;%= if D(7)==1 Then response.write "bgcolor='#FFFF00'" end if %&gt;
      &lt;%= if D(11)==1 Then response.write "bgcolor='#00FF00'" end if %&gt;
    &lt;/td&gt;
  &lt;/tr&gt;
  &lt;tr&gt;
    &lt;td width="29"&gt;
      &lt;%= if D(2)==1 Then response.write "bgcolor='#0000FF'" end if %&gt;
    &lt;/td&gt;
  &lt;/tr&gt;
  &lt;tr&gt;
    &lt;td width="29"&gt;
      &lt;%= if D(6)==1 Then response.write "bgcolor='#FFFFFF'" end if %&gt;
    &lt;/td&gt;
  &lt;/tr&gt;
  &lt;tr&gt;
    &lt;td width="29"&gt;
      &lt;%= if D(10)==1 Then response.write "bgcolor='#00FF00'" end if %&gt;
    &lt;/td&gt;
  &lt;/tr&gt;
&lt;/table&gt;
<p align="left"><a href="change.asp"><font color="#000080" face="Arial">View/Change the parameters of the system</a></font></p>