8-1-2002

Systems integration using Siemens' PC based automation technology

Adwait Palsule

Systems Integration Using Siemens' PC Based Automation Technology

Adwait Arvind Palsule

M.S. (Computer Integrated Manufacturing)

Thesis submitted in partial fulfillment of the requirements

for the Master of Science in department of Manufacturing Engineering Technology

in the College of Applied Science and Technology

of the Rochester Institute of Technology

August 2002

COLLEGE OF APPLIED SCIENCE & TECHNOLOGY

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

MASTER OF SCIENCE DEGREE THESIS

The M.S. Degree Thesis of Adwait Arvind Palsule has been examined and approved by the thesis committee as satisfactory for the Thesis requirement for the Master of Science Degree in Computer Integrated Manufacturing

---

Dr. Sudhakar R. Paidy

Dept. of Industrial and Systems Engineering

Kate Gleason College of Engineering

Prof. S. Manian Ramkumar

Dept. of Manufacturing Engineering Technology

College of Applied Science and Technology

Permission granted

Systems Integration Using Siemens' PC Based Automation Technology

I, Adwait Arvind Palsule, hereby grant the permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or part. Any reproduction will not be for commercial use or profit.

Date: 8/29/2002 Signature of Author: _____

Dedication

To my parents and family,

because of who I am, what I am and where I am.

Acknowledgement

Throughout the time I have been working in this applied research study, many people have assisted me. It is impossible to acknowledge them all. Nevertheless, I would like to specifically thank the following individuals:

Dr. Sudhakar Paidy, for all the time he dedicated to my research work and for the knowledge he shared with me throughout the course of this study. This thesis would not have been possible without his help.

Prof. S. Manian Ramkumar, for his continuous support and valuable comments on my work.

Mr. Dennis Wilk, Mr. Barry Hawley, and Mr. Peter Stansky of Siemens' Energy and Automation for their constant help and technical support.

To Prakash Gandhi, fellow graduate research assistant, for his support, advice and comments throughout the course of this study.

To all my friends who helped me from the beginning of my thesis for giving valuable support and comments and those who helped me to prepare for the final presentation.

Table of Contents

List of Figures

Appendices

No.	Appendix
A	Siemens' Software Installation
B	Hardware Configuration
C	Communicating Step7 with WINCC
D	Accessing the WINCC tags via the OPC channel
E	Quickstep program for the conveyor control
F	Symbols table used in the Step7
G	LAD program for the Inquire and Pallet Controller module
H	WINCC Project Documentation

Abstract

Manufacturing Systems Integration is the progressive linking and combination of the various components of the system to merge their functional and technical characteristics into a comprehensive interoperable unit. It requires one to work with different hardware and software. There are a number of vendors providing a large number of products. Integrating these varieties of products provides a greater value than the sum of the value provided by the individual products. What hinders the effective integration of these components is the diversity in the design and the use of these products. Systems Integration is eased by well-established standards in data communication, programming languages, application development environments and computer operating systems. Many vendors have attempted to come up with standards that are relatively open. However, when one has to integrate data among multiple vendors' architecture, a new set of challenges emerge.

The Siemens' PC-based automation technology is an emerging technology that appears to provide robust architecture for integrating all elements of the manufacturing environment. Applications ranging from simple control to distributed control and full-fledged Manufacturing Execution Systems can be developed using Siemens' architecture. The primary focus of this applied research work is to develop a Manufacturing Execution System to control a flexible manufacturing system using Siemens PC-based automation technology. This technology is implemented in a Flexible Manufacturing cell named the CAMCELL. The CAMCELL consists of two CNC machining centers, assembly robots, and a vision system, all of which are interlinked by a material handling system. The software architecture of the CAMCELL is based on NIST's five level hierarchy, discussed briefly in the report. Specifically it contains functional modules for order entry, scheduling and routing. In addition to these functional modules, there are various support modules such as order entry module, scheduler, router etc, two of which named the Inquire and the Pallet Controller that are implemented in this study. Siemens' Step 7 and WINCC software are used for the control and monitoring of the cell.

Section 1. Introduction

A system is defined as a collection of elements or components that are organized together for a common purpose. [1] Manufacturing Systems Integration is the progressive linking and combination of the various components of the system to merge their functional and technical characteristics into a comprehensive interoperable unit. [2] System Integration requires us to work with different hardware and software. There are a number of vendors providing a large number of products. Integrating these varieties of products provides a greater value than the sum of the value provided by the individual products. What hinders the effective integration of these components is the diversity in the design and the use of these products. Computer systems are continuously evolving, operating systems are rapidly changing and the programming languages are getting more and more object oriented.

Systems Integration is eased by well-established standards in data communication, programming languages, application development environments and computer operating systems. Many vendors have attempted to come up with standards that are relatively open. For example, the *Simatic PCS 7* is a process control system based on standard technologies and thus offers openness and also allows it to be linked up to systems made by other manufacturers. It has a flexible architecture that facilitates horizontal integration among the processes as well as vertical integration in the company-wide information network and integration to field engineering. [3] *Industrial*^{IT} *Solutions* from ABB are designed to perform as a fully integrated set of solutions across common hardware, software, engineering, spare parts, project management, training and service components. They are scaleable and open and hence you can implement only what you require now, and add functionality over time, as needs evolve. [4] The *Westinghouse Distributed Processing Family (WDPF®)* distributed control and information system provides modulating control, sequential control, and data acquisition for a wide variety of process applications. It has a dual network architecture that provides effective and reliable communications and a deterministic network for real-time process control data and an open Ethernet Information Highway for file transfer, remote access, and enterprise integration. It also offers the flexibility to fully integrate plant process control, local and wide area SCADA systems, PLC networks, and maintenance management and laboratory computer systems in a single, unified architecture. [5] IBM's SiView Standard, an advanced manufacturing execution system for semiconductor manufacturers, provides a complete open, multi-platform environment with the industry's richest functionality and largest range of Manufacturing Execution Systems (MES) applications, including material management, equipment and process management, schedule management, process control, and factory automation. SiView Standard integrates with a variety of third-party software to offer an end-to-end MES solution. [6] However, when one has to integrate data among multiple vendors' architecture, a new set of challenges emerge.

The Siemens' PC-based automation technology is an emerging technology that appears to provide robust architecture for integrating all elements of the manufacturing environment. Applications ranging from simple control to distributed control and full-fledged MES can be developed using Siemens' architecture.

The primary focus of this thesis is applied research leading is to understand and develop a Manufacturing Execution System to control a manufacturing cell named the CAMCELL. CAMCELL is a flexible manufacturing system containing two CNC machining centers, assembly robots, and a vision system, all of which are interlinked by a material handling system. The software architecture of the CAMCELL is based on NIST' five level hierarchy [7] Specifically it contains functional modules for order entry, scheduling and routing. In addition to these functional modules, there are various support modules (order entry, scheduler, router etc.) two of which named the Inquire and the Pallet Controller are implemented in this study.

This study aims to evaluate the Siemens' Automation System in terms of its important features in the context of Systems Integration. There are many communication standards that govern the data communication within a manufacturing cell. Such communication standards range from device-to-device standards like the RS-232 serial communication standard to a distributed network communication standards like Industrial Ethernet. Similarly, there are varieties of programming languages standards that are popular in the process control field. From simple graphical programming languages like the Ladder Logic or the functional block diagram to Sequential Programming languages and complex Structured Control languages, there are a number of programming languages that a control solution developer can chose from. These two aspects of Systems Integration in addition to other problems faced in the multi-vendor integration are a huge challenge to bring about effective Systems Integration. This study aims at studying the Siemens' Automation System through these perspectives.

In section number two, we discuss the data communication standards popular within the process control field. Such standards include Serial Communication standards like RS-232, RS-422, RS-485, and Profibus networks communication standards, Parallel Communication standards like IEEE-488 communication standard and distributed network communication standards like Ethernet and TCP-IP. Here, we also discuss the ISO OSI 7-layer model, which specifies the functions that are to be fulfilled at each level in communication systems. In this section we also describe the Simatic NET Profibus network that is used in the CAMCELL.

In section number three, we discuss the programming languages currently in use in the process control field. Such languages include Graphical Programming languages like the Ladder Logic (LAD) and the Functional Block Diagram (FBD), Textual Programming language like the Statement Lists (STL), Sequential Programming languages like the Grafcet and Graph, and also Structured Control languages.

In section number four, we describe the General Architecture of the Siemens' Simatic PC-based Automation System. The various components of this architecture are Simatic Controllers, Simatic HMIs, Distributed I/Os, the Simatic NET communication network and the Step 7 configuration and programming software. In this section, we also describe the Simatic Manager architecture. Simatic Manager is the basic application for configuring the hardware and programming logic in a control application. Main components in this architecture are the Simatic PC Station, Operator Station, WinLC, which is the logic controller, WINCC, which is the Supervisory Control and Data Acquisition software HMI of the Siemens'

Automation system. This section contains a brief description of these components of the overall Simatic system architecture as well as the Simatic Manager architecture.

In section number five, we describe the CAMCELL through three different perspectives viz. the Manufacturing and Material Handling perspective, Product/Process flow perspective and the Computer hardware hierarchy perspective.

In section number six, we describe the material handling system of the CAMCELL, its components and its functions. We also describe the control program that runs the material handling system written in the Quickstep programming language.

In section number seven, we describe the CAMCELL automation system. Here we describe the actual implementation of the Siemens' automation technology in the CAMCELL. Here we describe the implementation of two functional modules of the CAMCELL viz. the Inquire and the Pallet Controller.

References:

[1] http://searchwin2000.techtarget.com/sDefinition/0,,sid1_gci213083,00.html

[2] www.its.bldrdoc.gov/fs-1037/dir-036/_5625.htm

[3] http://www.sea.siemens.com/process/product/pa7ovrvw.html

[4] http://www.abb.com/global/

[5] http://www.westinghousepc.com/wdpf-control/descriptions.cfm

[6] www.ibm.com

[7] Dr. Paidy. S.R., Dr. Reeve, Richard, "Software Architecture for a Cell Controller", HICCS, 1990.

Section 2. Data Communication for Process Control

2.1 Introduction

In this section, we will focus on various communication standards that are relatively more popular in the process control field. Communication standards are an agreed upon set of rules and guidelines that govern communication between two devices, or two systems. There are two aspects to any communication standard, viz. the hardware aspect and the software aspect. The hardware aspect of a communication standard deals with such things as physical characteristics, electrical characteristics, types of connectors and so on. On the other hand, the software aspect deals with the ways in which information is packaged and transferred. In this section we will describe both these aspects of the various communication standards leading to the discussion of the Simatic NET Profibus network, which is a part of the Siemens' System and is used in the CAMCELL application.

2.2 Data Transfer Modes [1]

There are two different kinds of data transfer modes,

1. Serial transfer mode, or serial communication,
2. Parallel transfer mode, or parallel communication.

2.2.1. Serial Communication

In Serial Communication, data is transferred from sender to receiver one bit at a time through a single line or circuit. The name serial communication comes from the fact that the serial port takes 8, 16 or 32 parallel bits from a computer bus and converts it as an 8, 16 or 32 bit serial stream. Each bit of information is transferred in series from one location to another. Each stream of bits is broken up to 8 bits called words. Serial communication is of two types.

Synchronous Serial Communication

In this type of serial communication, the sending and receiving ends of the communication are synchronized using a clock that precisely times the period separating each bit.

Asynchronous Serial Communication

By introducing a start bit that indicates the start of a short data stream, the position of each bit can be determined by timing the bits at regular intervals. The two systems then don't have to be synchronized by a clock signal. When the receiving end of the communication receives the start bit it starts a short-term timer. By keeping streams short, there's not enough time for the timer to get out of sync. This method is known as asynchronous communication because the sending and receiving end of the communication are not precisely synchronized by the means of a clock

2.2.2. Parallel Communication

Simultaneous transmission of the eight bit-voltages that constitute a byte is referred to as "parallel transfer". When bits have to be moved about within the computer itself, they are transmitted along wires. If the data to be transmitted is in 8-bits format bytes, then eight separate, discrete wires must simultaneously carry the eight representative electrical voltages between the two points. Parallel transfer, then, is done byte-by-byte. Since all eight bits arrive at their destination at the same instant, parallel data transfer can be accomplished at extremely high speeds.

2.3 Serial Communication Standards [2]

The most popular serial communication standards are,

1. RS 232 communication standard,
2. RS 485 communication standard,
3. RS 422 communication standard, and
4. Profibus networks

2.3.1. RS 232 communication standard

RS-232 communication standard was introduced in 1960 by the Electronic Industries Association (EIA), and is currently the most widely used communication protocol. It is simple and inexpensive to implement. Even though relatively slow, it is adequate for most simple serial communication devices such as keyboards and mice. RS-232 is a single-ended data transmission system, which means that it uses a single wire for data transmission. Since useful communication is generally two ways, a two-wire system is employed, one to transmit and one to receive. Because signals traveling this single wire are vulnerable to degradation, RS-232 systems are recommended for communication over short distances (up to 50 feet) and at relatively slow data rates (up to 20 kbps). RS-232 is used to connect only two systems. [12]

2.3.2. RS 422 communication standard

RS-422 (EIA RS-422-A Standard) is a serial communication standard used on Apple Macintosh computers. RS-422 uses a differential electrical signal, as opposed to unbalanced signals referenced to ground with the RS-232. Differential transmission, which uses two lines each to transmit and receive signals, results in greater noise immunity and longer distances as compared to the RS-232. The greater noise immunity and distance are big advantages in industries. RS-422, like RS-232 is also used to connect two systems. [12]

2.3.3. RS 485 communication standard

RS-485 (EIA-485 Standard) is an improvement over RS-422, because it increases the number of devices from 10 to 32 and defines the electrical characteristics necessary to ensure adequate signal voltages under maximum load. With this enhanced multidrop capability, you can create networks of devices connected to a single RS-485 serial port. The noise immunity and multidrop capability make RS-485 the serial connection of choice in industrial applications requiring many distributed devices networked to a

PC or other controller for data collection, HMI, or other operations. RS-485 is a superset of RS-422; thus, all RS-422 devices may be controlled by RS-485. RS-485 hardware may be used for serial communication for up to 4000 feet of cable.

2.3.4. Profibus [9]

2.3.4.1 Overview of Profibus Network

Profibus is an acronym for "**Pro**cess **F**ield **Bus**". It is a modified version of the RS 485 standard and follows the international standards EN 50170, EN 50254 and IEC 61158[10]. It is a vendor-independent standard documented in the volume 2 of EN 50170. The transmission medium is either a copper cable network based on a shielded 2-core cable or a fiber-optic cable network as shown in figure 2.1.

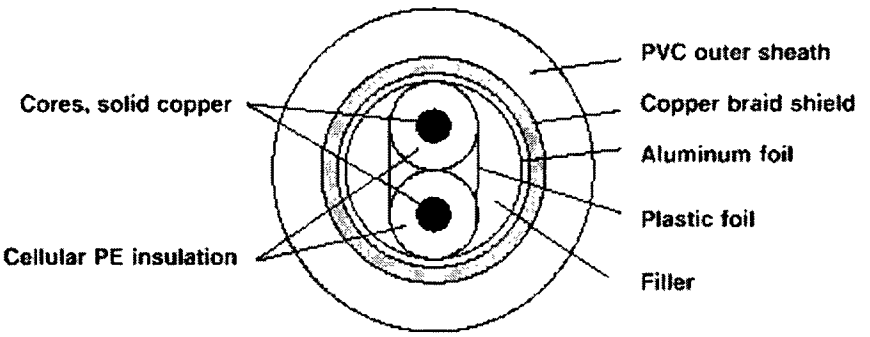


Fig 2.1 Structure of a shielded 2-core Profibus Cable

The transmission speed determines the length of the cable within a segment. The maximum length at the highest transmission speed (12 Mbit/sec) is 100m and the minimum cable length at the lowest transmission speed (9 kbit/sec) is 1000 m. The network can be expanded with repeaters.

The **P**rofibus supports the exchange of information between field devices and with systems at a higher system level. It is mainly used to transfer small to medium quantities of data. It offers a standardized interface for the transfer of process input and process output data between Simatic S7 stations and the field devices.

2.3.4.2 Access techniques

The network access technique for **P**ROFIBUS corresponds to the "Token bus" method specified by EN 50170, Volume 2 for active stations and the "Master/slave" method for passive stations. The access technique is independent of the transfer medium. Fig 2.2 shows the procedure used with active and passive stations. All active stations (masters) form, in a prespecified sequence, the "logical token ring" whereby each active station is aware of the other active stations and their sequence in the logical ring (the sequence is independent of the topological arrangement of the active stations on the bus). The right to access the medium (the "token") is passed from active station to active station in accordance with the sequence specified by the logical ring.

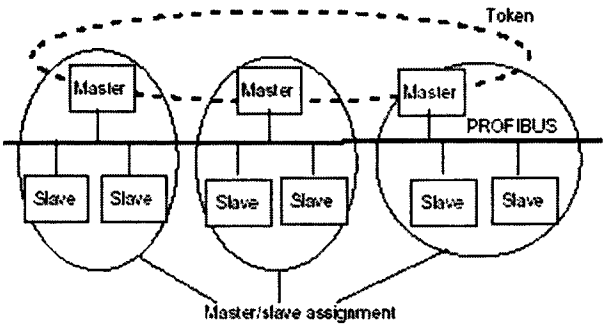


Fig 2.2. Profibus Access Technique

When a station receives the token (addressed to it), it has permission to send telegrams. The time allowed is specified by the so-called token holding time. Once this has elapsed, the station is only permitted to send one more high-priority message. If the station is not waiting to send a message, it passes the token onto the next station in the logical ring immediately. The corresponding token timers ("maximum token holding time", etc.) are configured for all active stations. If an active station is in possession of the token and connections to passive stations are configured for it (master/slave links), these passive stations are queried (e.g. variables are read) or data is sent to them (e.g. set point values). Passive stations never receive the token. This access technique allows stations to be added and removed under operating conditions.

2.4 Parallel Communication Standards

Parallel communication is characterized by multiple bits transfer that leads to higher data transfer rates, over relatively shorter distances.

The parallel communication greatly increases transfer speeds by using an eight-wire connector, which transmits the eight bits in a byte of data simultaneously, thus sending an entire byte of data in the time it takes to send a single bit in a serial system. This byte of data is supplemented by several other handshaking signals, each sent on its own wire, which ensure that data transfer takes place smoothly [3]

2.4.1. IEEE 488 (GPIB) [4]

The IEEE-488 bus was developed to connect and control programmable instruments, and to provide a standard interface for communication between instruments from different sources. Hewlett-Packard originally developed the interfacing technique, and called it HP-IB. Since the interface was so versatile, that the IEEE committee renamed it as GPIB (General Purpose Interface Bus). The IEEE-488 interface system consists of 16 signal lines and 8 ground lines. The 16 signal lines are divided into 3 groups (8 data lines, 3 handshake lines, and 5 interface management lines). The standard IEEE-488 cable has both a plug and receptacle connector on both ends.

The IEEE-488 bus specifies a maximum total cable length of 20 meters with no more than 20 devices connected to the bus and at least two-thirds of the devices powered on. A maximum separation of 4 meters between devices and an average separation of 2 meters over the full bus should be followed.

2.5 Network Interfaces

2.5.1. Network Topologies

There are two widely used network topologies

- Star Topology
- Ring Topology

2.5.1.1. Star Topology [5]

Star topology is a network configuration in which there is a central point to which a group of systems are directly connected. With the star topology, all transmissions from one system to another pass through the central point, which may consist of a device that plays a role in managing and controlling communications.

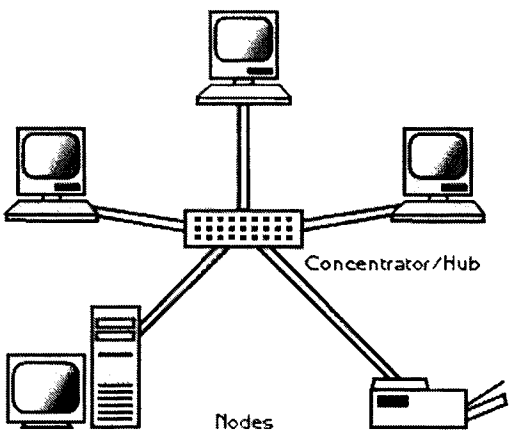


Fig. 2.3 Star Topology

2.5.1.2. Ring Topology [6]

A ring is a network topology or circuit arrangement in which each device is attached along the same signal path to two other devices, forming a path in the shape of a ring. Each device in the ring has a unique address. Information flow is unidirectional and a controlling device intercepts and manages the flow to and from the ring.

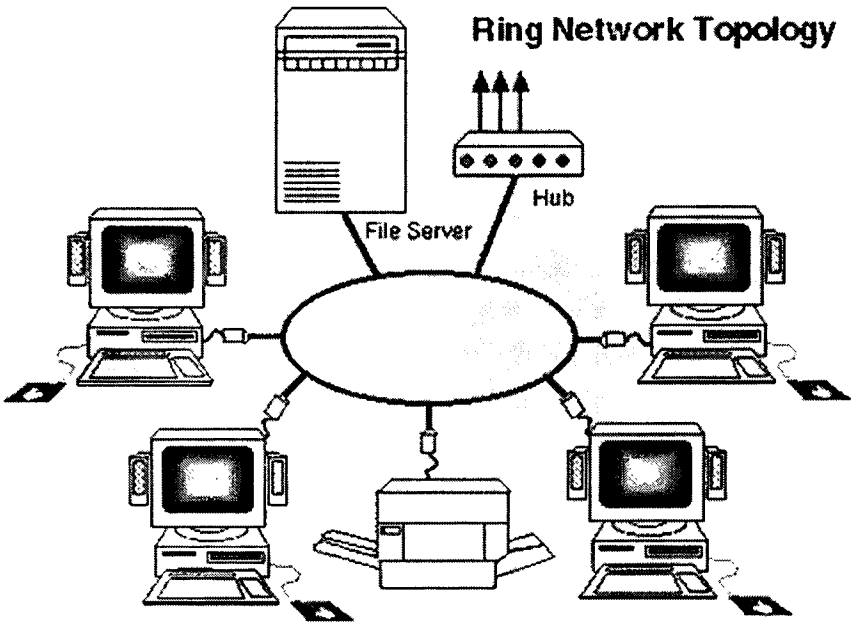


Fig 2.4 Ring Topology

2.5.2 TCP/IP

TCP and IP were developed by a Department of Defense (DOD) research project to connect a number of different networks designed by different vendors into a network of networks (the "Internet"). Several computers in a small department can use TCP/IP (along with other protocols) on a single LAN. The IP component provides routing from the department to the enterprise network, then to regional networks, and finally to the global Internet. As with all other communications protocol, TCP/IP is composed of layers:

IP - is responsible for moving packet of data from node to node. IP forwards each packet based on a four-byte destination address (the IP number).

TCP - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP is used by applications that require reliable data delivery. TCP uses a check-sum with the data sent that the TCP layer at the other end uses to verify that the data arrived undamaged. If the data checks out, the receiving TCP layer sends an acknowledgment back to the sender. If it arrives damaged, the receiver discards it and the sender, after an appropriate time-out period, resends it.

But before data is transmitted, the two hosts exchange a three-way handshake. The sender sends a signal telling the receiver it wants to set up a connection and how the data will be sequenced. The receiving host sends a signal back acknowledging receiving the signal. The sending host sends a signal back acknowledging the acknowledgement and begins data transmission. The connection at the end of the data transmission is closed with another 3-way handshake.

2.5.3 Ethernet

Ethernet is a local area network (LAN) technology that transmits information between computers at speeds of 10 and 100 million bits per second (Mbps). Currently the most widely used version of Ethernet technology is the 10-Mbps twisted-pair variety.

The 10-Mbps Ethernet media varieties include the original thick coaxial system, as well as thin coaxial, twisted-pair, and fiber optic systems. The most recent Ethernet standard defines the new 100-Mbps Fast Ethernet systems, which operates over twisted-pair and fiber optic media.

Each Ethernet-equipped computer, also known as a station, operates independently of all other stations on the network and there is no central controller. All stations attached to an Ethernet are connected to a shared signaling system, also called the medium. Ethernet signals are transmitted serially, one bit at a time, over the shared signal channel to every attached station. To send data a station first listens to the channel, and when the channel is idle the station transmits its data in the form of an Ethernet frame, or packet.

After each frame transmission, all stations on the network must contend equally for the next frame transmission opportunity. This ensures that access to the network channel is fair, and that no single station can lock out the other stations. Access to the shared channel is determined by the medium access control (MAC) mechanism embedded in the Ethernet interface located in each station. The medium access control mechanism is based on a system called Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

2.6. ISO OSI Seven-Layer Model [8]

The Open Systems Interconnection (OSI) 7 Layer Model is the product of years of work by many organizations. The International Standards Organization (ISO) has compiled all the results. This standard is called a reference model. It specifies the functions that are to be fulfilled at each level in communication between telephone systems, or computers. These 7 layers are thought to be adequate to encompass all communications functions.

Layer 1: The Physical Layer is concerned with transmitting raw bits, electrical impulses, light, and radio signals over a communications medium. The primary concern of a designer for this layer is mechanical, electrical, or optical matters. Specifications for wave shapes, signaling rates, cables and connectors are involved in this layer

Layer 2: The Data Link Layer takes the bits passed by the physical layer and creates and recognizes frame boundaries; by this technique it transfers units of information to the other end of the physical link It also contains the rules governing access to the network cable, **Media Access Control (MAC)** which are rules that are followed to move information into and out of the media.

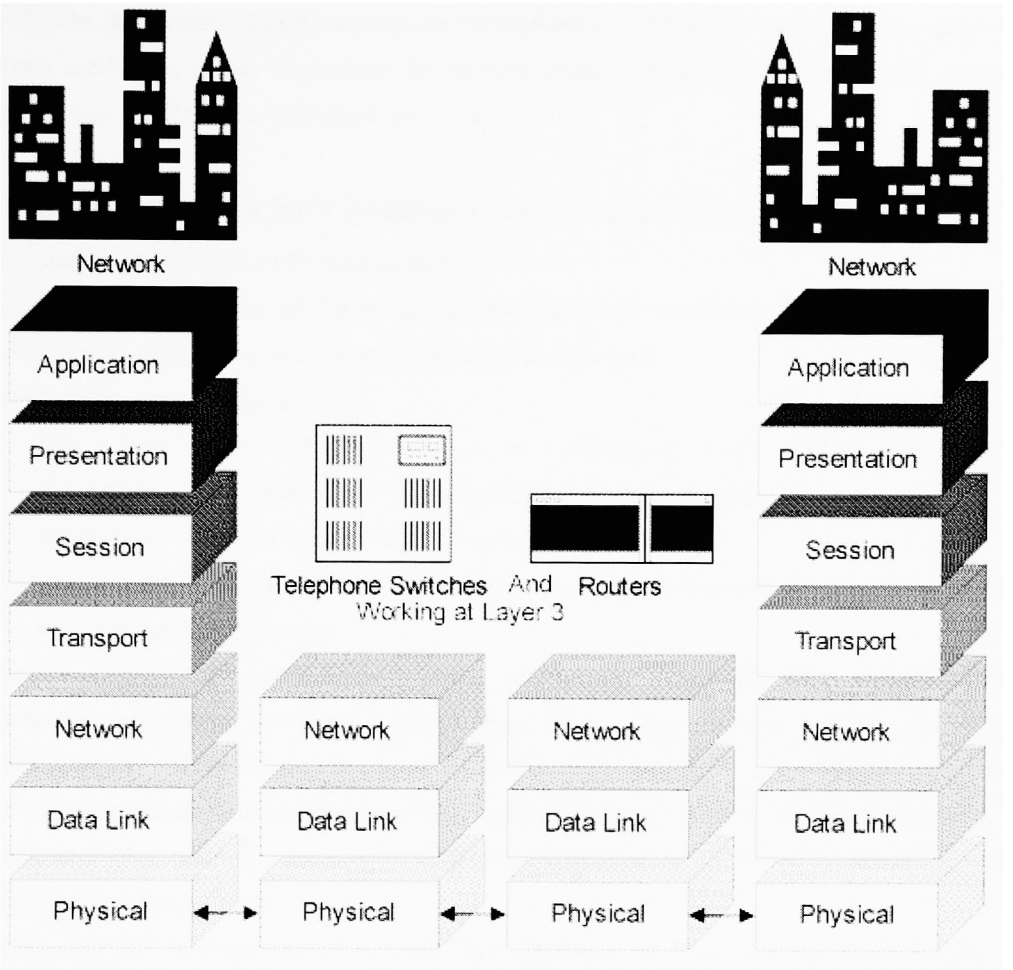


Fig 2.5 OSI ISO 7 layer model

Layer 3: The Network Layer controls operations of sub networks that might intervene between the two communication devices. This layer routes information among different networks.

Layer 4: The Transport Layer splits up information into appropriate sizes (segments information) so it will fit into packets of the right size for the networks being used. In many cases, it ensures all the packets arrive at the other end with errors and assembled in order and without duplication. Therefore, it provides end-to-end data integrity and quality of service.

Layer 5: The Session Layer controls the establishment and continuation of a particular communication between devices. It coordinates interaction between end-application processes, keeping the two devices talking to each other and maintaining a connection.

Layer 6: The Presentation Layer performs conversions on information. These include conversion of code sets, encryption, text compression, and protocol conversion for virtual terminal communication. This layer may also translate file formats that differ between devices.

Layer 7: The Application Layer contains the final particulars required for programs to communicate. This layer establishes means for making the network appear transparent to user devices, joining the communications stream to the individual device.

2.7 Topology of Simatic NET Profibus network used in the CAMCELL

2.7.1. Overview of the SIMATIC NET system

SIMATIC NET is the name of the communication networks connecting SIEMENS programmable controllers, host computers, workstations and personal computers.

SIMATIC NET includes the following:

- The communication network consisting of transmission media, network attachment and transmission components and the corresponding transmission techniques
- Protocols and services used to transfer data between the devices
- The modules of the programmable controller or computer that provide the connection to the communications processors.

To handle a variety of tasks in automation engineering, SIMATIC NET provides different communication networks to suit the particular situation. The topology of rooms, buildings, factories, and complete company complexes and the prevalent environmental conditions mean different requirements. The networked automation components also make different demands on the communication system.

The communication network used in the CAMCELL application is the Profibus network, which was discussed earlier in this section.

The SIMATIC NET topology used in the CAMCELL application is shown below in the figure 2.6

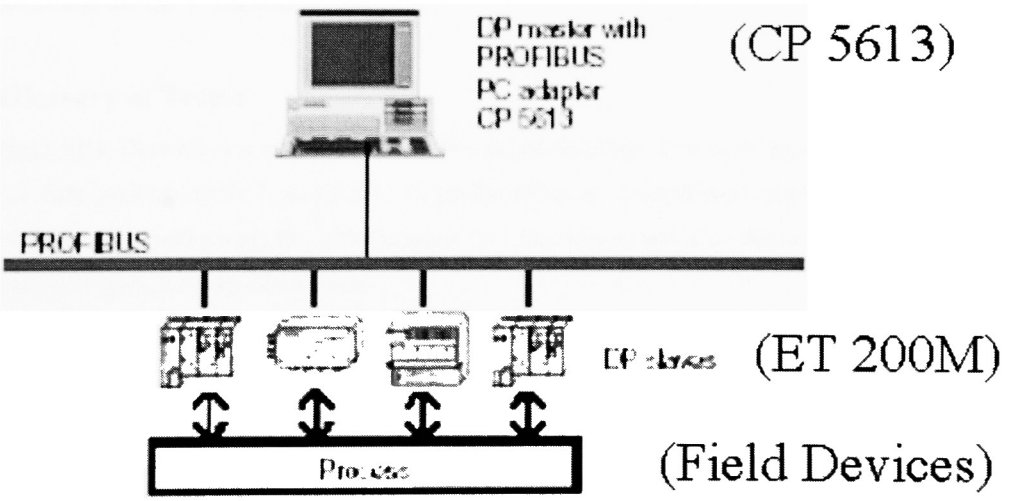


Fig 2.6 SIMATIC NET topology as used in the CAMCELL Application. [11]

2.7.2 CP 5613 Communication Processor

In any SIMATIC S7 application, a Communication Processor (CP) is always required for the Profibus network. In our CAMCELL system, we use the CP 5613 communication processor.

The CP 5613 is a PCI card with a microprocessor that allows connection of a SIMATIC programming device to Profibus under Windows NT 4.0 or Windows 2000. The CP 5613 can also be used to implement control tasks on a PC, thus facilitating PC based control.

The CP 5613 card is a short PCI card with a 9-pin sub D socket for connection to Profibus. It has diagnostic LEDs for installation and commissioning and during operation of the module. In order to provide access to the process data, the CP 5613 operates as the Profibus master module buffering the process image (input/output and diagnostic data) in the DP RAM (memory area on the CP). High performance data communications is handled autonomously by the CP 5613 hardware. The process data from the slaves are always consistent as they all originate from one and the same DP (decentralized peripheral) cycle.

2.7.3 ET 200 M distributed I/O module

The ET 200-M is a modular I/O station. It is the passive station of the Profibus field bus and has a maximum data transfer rate of 12Mbit/s. A typical ET 200M station consists of:

- Power Supply
- IM 153 interface module
- Up to 8 I/O modules

No specific slots are assigned to the I/O modules. Any combination of modules is possible. The ET 200M is connected to the PROFIBUS-DP via an IM 153 interface module. The inputs and outputs of the modular ET 200M I/O station can be accessed from the user program in the PLC in the same manner as the inputs and outputs of the central controller.

Communication via the bus system is handled completely by the master interface module in the central controller and the IM 153 interface module.

2.8 Glossary of Terms

- **Data bits**: Data bit is a measurement of the actual data bits in a transmission. Standard values for the data packets are 5, 7, and 8 bits. A packet refers to a single byte transfer, including start/stop bits, data bits, and parity. Since the number of actual bits depends on the protocol selected, the term packet is used to cover all instances
- **Stop bits**: Stop bits are used to signal the end of communication for a single packet. Typical values are 1, 1.5, and 2 bits. More the bits used for stop bits, greater is the lenience in synchronizing the different clocks, but slower the data transmission rate.
- **Parity**: Parity is a simple form of error checking that is used in serial communication. There are four types of parity: even, odd, marked, and spaced. The option of using no parity is also available. For even and odd parity, the serial port sets the parity bit (the last bit after the data bits) to a value to ensure that the transmission has an even or odd number of logic high bits. If the parity is odd, then the parity bit is 1, resulting in 3 logic-high bits. Marked and spaced parity does not actually check the data bits, but simply sets the parity bit high for marked parity or low for spaced parity. This

allows the receiving device to know the state of a bit to enable the device to determine if noise is corrupting the data or if the transmitting and receiving device clocks are out of sync.

- **Baud rate**: Baud rate is a measurement of speed for communication. It indicates the number of bit transfers per second. For example, 300 baud is 300 bits per second. Common baud rates for telephone lines are 14400, 28800, and 33600. Baud rates greater than these are possible, but these rates reduce the distance by which devices can be separated.
- **Bus**: In a computer, bus is the data path on the computers' motherboard that interconnects the microprocessor with attachments to the motherboard in expansion slots. Such attachments may be hard disk drives, CD-ROM drives etc. On a network, a bus is a transmission path on which signals are dropped off or picked up at every device attached to the line. Only the devices addressed by the signals pay attention to them, the others discard the signals. There are various types of bus available such as ISA bus, PCI bus, EISA bus, IDE bus, EIDE bus, VME bus etc.
- **Port**: On computer and telecommunication devices, a port is generally a specific place for being physically connected to some other device, usually with a socket and plug of some kind. Typically, a personal computer is provided with one or more serial ports and usually one parallel port. The serial port supports sequential, one bit-at-a-time transmission to peripheral devices such as scanners and the parallel port supports multiple-bit-at-a-time transmission to devices such as printers.
- **Standard**: An agreed upon set of rules and guidelines. Many standards are evolved from popular methods while others are crafted by professional groups (ANSI, ISO, etc.)
- **LAN**: A local area network (LAN) is a group of computers and associated devices that share a common communications line and typically share the resources of a single processor or server within a small geographic area (for example, within an office building).
- **WAN**: A computer network that spans a relatively large geographical area. Typically, a WAN consists of two or more local-area networks (LANs). The largest WAN in existence is the Internet.
- **Firewall**: Firewall is a system designed to prevent unauthorized access to or from a private network. Firewalls can be implemented in both hardware and software, or a combination of both. Firewalls are frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets. All messages entering or leaving the intranet pass through the firewall, which examines each message and blocks those that do not meet the specified security criteria.
- **Router:** Router is a network device that sends and receives Network protocol-data units (packets) and relays packets from one device to another through the network.
- **Hub:** Hub is local area network equipment that allows multiple network devices to be connected to the LAN cabling system through a central point.
- **Switch:** A LAN switch is a network interconnection device used to allow the network interface card in a computing device to be connected to one of multiple LAN transmission medium segments

- **Token Ring:** Token ring is a LAN data link technology, in which systems are connected to one another using a point-to-point twisted – pair cable segments to form a ring structure. A system is allowed to transmit only when it has the token, which is passed from one system to another around the ring.

References:

[1] http://www.rad.com/networks/1995/rs232/rs232.htm

[2] http://www.quatech.com/Application_Objects/FAQs/comm-over-rs-232.htm

[3] http://www.quatech.com/Application_Objects/FAQs/comm-over-parallel.htm

[4] http://www.techsoft.de/htbasic/tutgpib.htm

[5] http://fcit.coedu.usf.edu/network

[6] http://lrs.stcloudstate.edu/cim/courses/im644/ring.html

[7] http://www.ots.utexas.edu/ethernet/

[8] http://web.singnet.com.sg/~psylence/c3p03.htm

[9] Berger, Hans. Automating with SIMATIC. SIEMENS 2000.

[10] http://www.profibus.com/profibus.html

[11] http://www.sea.siemens.com/autogen/docs/net/pfb/man/DP Base Program Interface for 5613.pdf

[12] http://www.synchrotech.com/support/serial_standards.html

Section 3. Programming Languages for Process Control

3.1 Introduction

In this section we shall discuss the various programming languages used to write control logic for process control. There are various programming languages available for writing control logic. The various types of programming languages include graphical programming languages, statement lists, sequential programming languages, and structured Control Languages. First, we describe these programming languages and also discuss programming languages within each category. Next, we describe the Ladder Logic (abbreviated as LAD in the Siemens' Technology) and the Functional Block Diagram (abbreviated as FBD in the Siemens' Technology), which are graphical programming languages and Statement Lists, which is a textual programming language. Next, we will discuss the Grafcet, Graph, and the HiGraph programming languages, which are sequential programming languages. Next, we will discuss the Structured Control Language. We shall discuss the important features of each programming language and also discuss their advantages and disadvantages.

3.2 Graphical Programming Languages

Graphical programming languages make use of symbols and icons in order to build control logic. Graphical programming languages are especially suitable for representing bit logic, in a form either modeled on a relay ladder logic diagram (LAD) or resembling an electronic circuit diagram (FBD).

3.2.1 Ladder Logic (LAD)

Ladder logic programming is a graphical representation of the program designed to look like relay logic. [4] In LAD, user writes the program for the control application by arranging graphical program elements. These graphical program elements consist of contacts, coils and boxes, which are linked together in a form resembling a relay ladder logic diagram. In LAD, the entire program is contained in networks. [1] Each network consists of one rung, that is a logic operating in a coil or box. Each rung is a combination of input conditions (symbols) connected from left to right, with the symbols that represent the output at the far right. The symbols that are represented as inputs are connected in series, parallel, or some combination of the two to obtain the desired logic. [2]

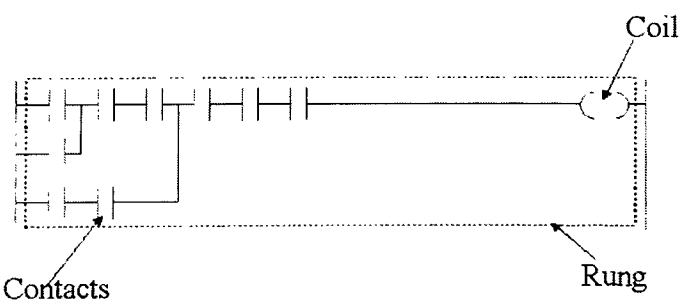


Fig 3.1 A rung in a ladder logic program [3]

3.2.1.1 Contacts in LAD

Contacts are used in LAD programming language to check the states of binary addresses, such as inputs. By arranging the contacts in series or parallel, logic operations can be performed. There are two types of contact; Normally Open Contacts and Normally Closed Contacts. In a normally open contact, the input is checked for a high state, the contact closes if the input turns high. In a normally closed contact, the input is checked for a low state, the contact closes if the input turns low.

3.2.1.2 Coils in LAD

Coils are used in LAD programs to set or reset bit addresses such as outputs. Coils energizes the bit address when the power flows through the coil, and de-energizes the bit address when the power stops flowing through it. Coils are also used to control timer and counter functions, call blocks without parameters, perform jumps to a different place in the program, and so on.

3.2.1.3 Example of an LAD rung

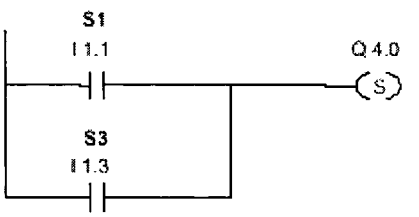


Fig 3.2 An example of a rung in a LAD program [5]

In the above example, an output S is energized or turned ON, when either input S1 or input S2 is energized or turned ON.

3.2.1.4 Advantages of LAD

The Ladder Logic Programming offers many advantages. Some of the advantages are as follows,

1. The LAD programming language is a part of International Electrotechnical Commission (IEC) 61131-3 standard for Industrial Control Programming. [6]
2. It is the most popularly used programming language for control programming
3. It is reasonably intuitive, especially for technicians with relay logic experience [2]
4. It is particularly effective in an on-line mode, when the PLC is actually performing control. The operation of the logic is apparent from the highlighting of the various relay contacts and coils on the screen, which identifies the logic state in real time. [2]
5. It is simple and easy to use

3.2.1.5 Disadvantages of LAD

Ladder Programming has become one of the most popular graphical languages for programming PLCs but unfortunately has a number of problems. [7]

1. The ladder symbols vary between different PLC products.
2. Poor facilities for structured or hierarchical program decomposition
3. Poor facilities for addressing and manipulating data structures,

4. Limited facilities for building complex sequences,
5. Limited control over program execution,
6. Facilities for arithmetic operations are cumbersome.

3.2.2 Functional Block Diagram (FBD)

Functional Block Diagram is another graphical language, popular in Europe. FBD program elements appear as blocks that are "wired" together, analogous to circuit diagrams. It is well suited for representing batch control applications. [9] In FBD, control programs are created by interconnecting AND and OR boxes. FBD provides function boxes for performing logic operations on signal states, simple boxes for processing the result of logic operations and complex boxes for non-binary functions.

3.2.2.1 Boxes in FBD

There are two types of boxes in FBD. Simple boxes operate on bit addresses, like outputs. These usually only have one input and may contain additional letters or symbols. There are simple boxes for setting or resetting bit addresses, evaluating signal edges, setting or resetting timer and counter addresses, and so on. Complex boxes are used for program elements with non-binary functions.

3.2.2.2 Binary Functions in FBD

Binary functions are used in FBD to check the signal states of bit addresses, such as inputs, and perform logic operations on them. The functions available are AND, OR, and XOR. All these functions can have more than two inputs. The function boxes can be interconnected to program complex logic operations in one network.

3.2.2.3 Example of an FBD Block

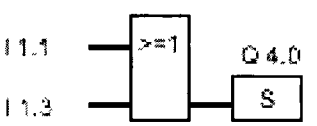


Fig 3.3 An example of a block in Functional Block Diagram

In the above example, the output S turns ON, when either the input I1.1 or I1.3 turns ON.

3.2.2.4 Advantages of FBD [8]

The Functional Block Diagram offers the following advantages,

1. Simultaneous programming and documenting (overview, comments, reliability, information flow)
2. Universal applicability (Signal processing, numeric, arithmetic, integer, floating point)
3. Structured programming (define and call subroutines)
4. Standardized set of Functions and Function Blocks

3.2.2.5 Disadvantages of FBD

Following are the disadvantages of the Functional Block Diagram

1. Poor facilities for structured or hierarchical program decomposition
2. Limited control over program execution
3. Facilities for arithmetic operations are cumbersome.

3.3 Statement List (STL)

STL is a text-oriented language in which the control task can be described in the form of a list. It very much looks like the short and simple lines of assembly codes. [11] Thus the control program is written in the form of a series of statements. Each statement contains an instruction that defines what is to be done. Depending on the type of instruction, this may be followed by an address that defines what action is to be done to. The elements of the statement lists closely resemble the mnemonics of microprocessor assembly languages. [11]

3.3.1 Example of an STL program

```
NETWORK 1
          LD      I0.0
          A       I0.1
          =       Q0.0
NETWORK 2
          LD      I0.4
          O       I0.5
          =       Q0.1
```

Fig 3.4 An example of STL instruction

A statement list provides another view of a set of instructions. The operation, what is to be done, is shown on the left. The operand, the item to be operated on by the operation, is shown on the right. In the above figure, in network 1, output Q0.0 turns high when the inputs I0.0 and I0.1 both turn high. On the other hand, the output Q0.1 turns high, when either the input I0.4 or I0.5 turns high. [12]

3.3.2 Advantages of STL programming

Statement Lists offer the following advantages, [13]

1. STL is very similar to an assembly code, hence the code can be easily read by everyone
2. STL's generic algorithms allow algorithms to be applied to many different structures
3. STL is easy to learn. The library is quite small owing to the high degree of generality.
4. STL containers are very close to the efficiency of hand-coded, type-specific containers.

3.3.3 Disadvantages of STL programming

Following are the disadvantages of the Functional Block Diagram, [13]

1. Error checking is difficult in STL. This means that if the programmer makes an error, an error may or may not occur when the erroneous instruction is made, but it will put the program in an unstable state (e.g. Writing off the end of an array).
2. Access to the exact source that one uses may be difficult, so if the user needs specific information about code details, they may not be accessible.

3.4 Sequential Programming Language

Sequential Programming Language is a method of representing the key elements of a sequential process, i.e. conditions required for passing from one state to another, and the effects present while in a particular state. [10]

In sequential control systems, unlike logic control systems, the static assignment of the input signals to the outputs is not of importance. It's the chronological sequence of these assignments that is important. The control processes that are executed one after the other are divided into individual steps. A step consists of one or more actions. Only the action in an active step (step currently being executed) is performed. The next step cannot be processed until the transition conditions are met. The transition can be dependent on the process, or can be dependent on a time factor. [1]

3.4.1 Grafcet

Grafcet is an international standard for sequential function charts that was released by the IEC in 1988. [17] It is a type of flow chart representation specifically developed for the needs of industrial control systems. Sequential operations are represented in Grafcet as a series of steps. The conditions required to move from one step to the next are referred to as transition conditions. The steps are represented by numbered squares, where the step comprises of one or more on-off or analog type actions. [19] A transition, which separates steps, and is represented by a horizontal hash, contains the logic condition for terminating the execution of the step. [19] The method can be used as a design tool for whatever technology is being used to control the system – relays, pneumatic logic, or PLCs. [18]

3.4.1.1 Example of a Grafcet program

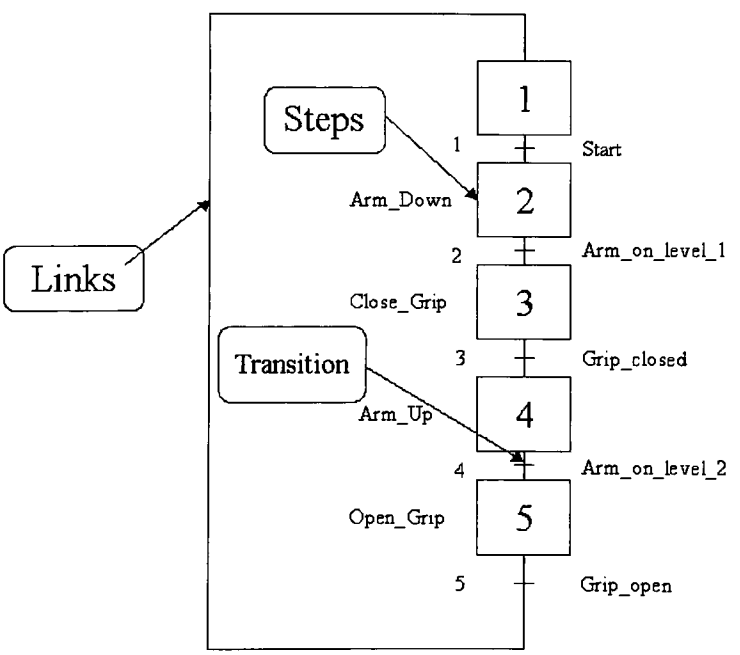


Fig 3.5 Grafcet Program

In the above example, there are steps and transitions taking place in sequence. The steps are shown in squares and transitions are shown in horizontal hashes.

3.4.1.2 Advantages of Grafcet

Following are the advantages of Grafcet,

1. Grafcet is supported by the IEC standard [17]
2. In executing the Grafcet, the PLC need only scan the active steps and the succeeding transitions. Hence scan times are reduced with Grafcet. [18]
3. Being developed for sequential processes, Grafcet has outstanding features such as parallel processing.
4. Grafcet application can be transferred into different controllers. With the help of software like Cadepa [19] the standard Grafcet code can be converted into machine code for most brands of PLCs. It can be transferred into ladder logic for Allen Bradley PLC-5, Modicon 984, Siemens S5 and others. [17]

3.4.1.3 Disadvantages of Grafcet

Following are the disadvantages of Grafcet

1. Not a very popular language

3.4.2 Graph

Graph is a sequential programming method for sequential control systems. [1] User can write the conditions enabling transitions from one step to the next in LAD or STL. Alternative or parallel branching extend the scope of the linear execution of consecutive steps. With Graph, operations within a process are configured and programmed in a standardized display mode (in compliance with IEC 61131-3). The process (e.g. the manufacture of a component) is divided into sequential or simultaneous steps. This division makes the structure of the PLC program easier to understand and also easier to analyze in the event of a malfunction. This is particularly useful in manufacturing to avoid costly downtimes. [14]

3.4.2.1 Example of a program in Graph

In the operation sequences, the individual steps of a process and the transitions to the next steps are represented as rectangles and lines as shown in figure 3.6. Qualifiers, within the steps, can initiate actions. (E.g. time-delay). If, for example, the operating sequence describes a drilling process, the "lower the drill" will be a step and the "motor ON" will be an action. The transitions describe the conditions under which a transition to the next step should occur. Moreover, interlocking and monitoring conditions can be defined for every step. An interlock can be used to prevent the execution of actions. Monitoring conditions enable the recognition of operational faults. All conditions (transitions, interlocking or monitoring) can be programmed either in LAD or FBD.

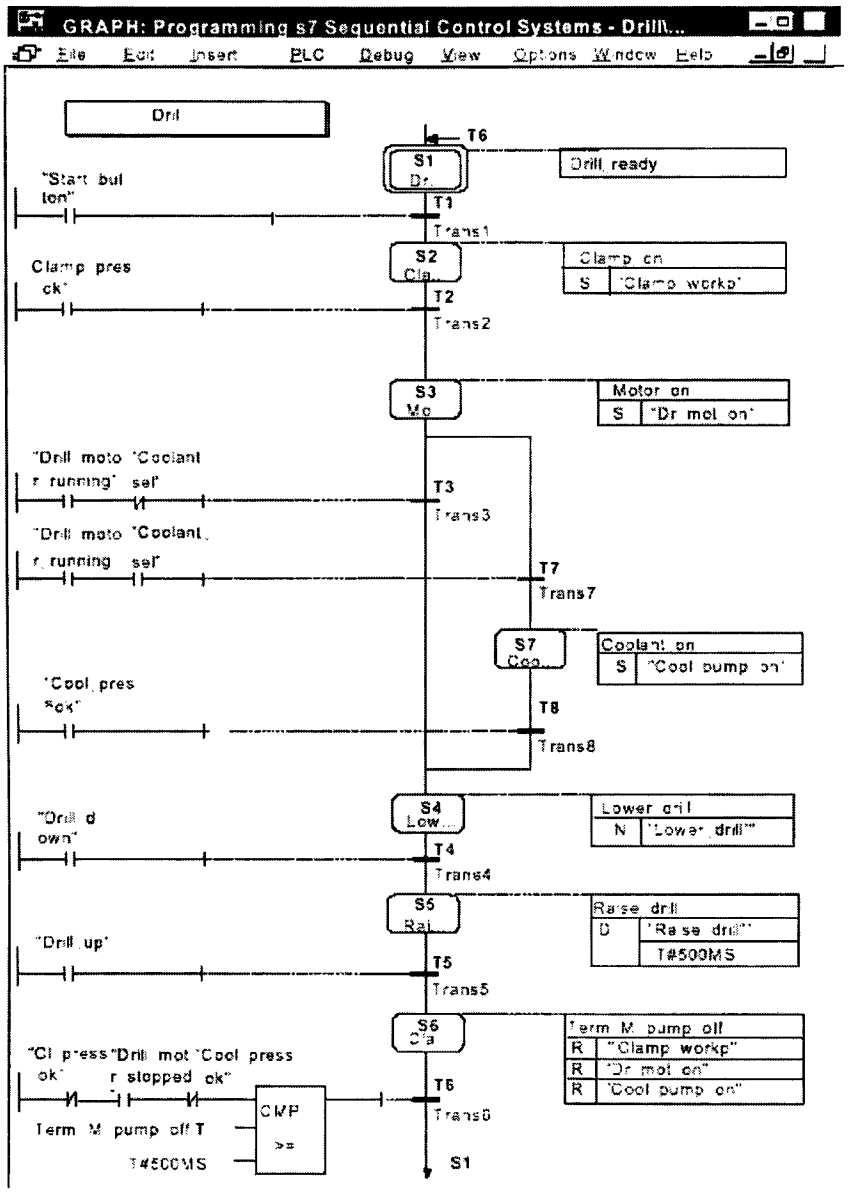


Fig 3.6 A Graph program

3.4.2.2 Advantages of Graph

Following are the advantages of the Graph programming language, [15]

1. Graph enables clear configuring of the process in the planning phase
2. Graph provides clear graphic representation of the process using sequence chains, resulting in easy maintenance and adaptation of programs
3. Easy troubleshooting with internal diagnostics functions, resulting in minimizing of expensive downtimes is possible with Graph.

3.4.2.3 Disadvantages of Graph

Following are the disadvantages of the Graph programming language,

1. Graph is suitable only for sequential control. Logic control cannot be programmed by Graph. [15]
2. The transitions and steps need to be programmed in either LAD or FBD. Hence Graph is not an independent language like LAD or FBD.

3.5 Structured Control Language

Structured Control Language (SCL) is a PASCAL-like high-level language optimized for programmable controllers. SCL is in compliance with IEC61131-3, and is especially suited to the programming of complex algorithms or for data processing projects. [14]

Apart from traditional control tasks, structured control language assists in performing data management tasks and complex mathematical operations. [16]

SCL statements consist of expressions, control statements, functions and block calls. Expressions assign values, which can also be result of arithmetic and logic operations or comparison functions. Control statements execute the program branches or repeat sections of a program.

```
FUNCTION_BLOCK FB27
VAR_INPUT
  SIG_SEL       : INT   := 0;
  GRP1_SEL      : BOOL  := 0;
  GRP2_SEL      : BOOL  := 0;
  GRP3_SEL      : BOOL  := 0;
END_VAR

VAR_OUTPUT
  SEL_OUT       : INT   := 0;
  GRP1_OUT      : BOOL  := 0;
  GRP2_OUT      : BOOL  := 0;
  GRP3_OUT      : BOOL  := 0;
END_VAR

VAR
  SELECT        : INT;
  MAX           : INT;
END_VAR

BEGIN
  SELECT        := SIG_SEL;
  MAX           := 3;
  IF SELECT  <  0 THEN                  //make it positive
    SELECT      := -SELECT;
  END_IF;
  IF SELECT  >  MAX THEN                //limit to MAX
    SELECT      := MAX;
  END_IF;
  SEL_OUT       := SELECT;
  GRP1_OUT      := GRP1_SEL;
  GRP2_OUT      := GRP2_SEL;
  GRP3_OUT      := GRP3_SEL;
END_FUNCTION_BLOCK
```

Fig 3.7 A function block in SCL

3.5.1 Advantages of Structured Control Language

Following are the advantages of Structured Control Language

1. SCL enables simple and fast program development for the user, by the application of powerful language elements such as IF.. THEN.. ELSE. [15]
2. Programs written in SCL have improved comprehensibility and improved structure. [15]
3. SCL programs are programmed as ASCII sources and are therefore easy to import and export. [15]

3.5.2 Disadvantages of Structured Control Language

Following are the disadvantages of Structured Control Language,

1. SCL requires the knowledge of basic high level programming language and hence not popular with technicians
2. SCL is not a suitable for simple logic controls.

References:

[1] Berger, Hans. Automating with SIMATIC. SIEMENS, 2000.

[2] Petruzella, Frank. Programmable Logic Controllers. Glencoe McGraw – Hill, 1996.

[3] http://xtronics.com/toshiba/llp.pdf, 1999.

[4] Gary A. Mintchell. http://www.controleng.com/archives/1998/ctl0402.98/04ebas.htm, 1998.

[5] Step 7- Ladder Logic for S7-300 and S7-400. Siemens Help Manual, 2000.

[6] http://www.plcopen.org/, 2000.

[7] http://www.searcheng.co.uk/selection/control/Articles/IEC61131/main.htm, 2001.

[8] http://www.plcopen.org/training_education/FBD2520Tutorial.ppt, 1999.

[9] http://www.controleng.com/archives/1996/02/issues/na/02c154.htm, 2001.

[10] N.J.Nelson. http://bournemouth.ac.uk/forth/euro/ef00/nelson00b.pdf, 2000.

[11] www.pcbasedautomation.com, 2001.

[12] http://www.sea.siemens.com/step/pdfs/plc_1.pdf, 2000.

[13] http://www.exciton.cs.rice.edu/comp410/STL/STL_Overview.htm, 2001.

[14] http://wwwlhc.cern.ch/IndCtrl/GUAPI/Siemens_Step7Technique.html, 2001.

[15] http://www.sea.siemens.com/sw/docs/s7pro.pdf, 2001.

[16] Structured Control Language for Step 7. Siemens Help Manual, 2000.

[17] Johnson, Hugh. Grafcet/Sequential Function Charts, 1992

[18] Lloyd, Michael. Grafcet Adds sequence dimension to ladder language, 1989

[19] Perron, Stephen. The New Generation Language for Control Design, 1992

Section 4. Simatic PC Based Automation System

4.1 Introduction

In this section, we shall discuss the Simatic Automation System. We shall discuss about the various components of the system and then describe them in brief. We shall discuss the various Simatic Controllers and Distributed I/Os and the HMIs. Next, we shall describe the Step 7 software in details. The Step 7 software is the central development tool in the Simatic Automation System. Next, we shall discuss the general architecture of the projects used in the Siemens' PC based automation technology. We shall present an overview of the project structure and discuss the main elements of the structure in detail. The main elements in this architecture are the SIMATIC Manager, the Hardware configuration, the Organization blocks, the symbols table, the operator stations, WINCC Explorer, and the WINLC, which is the PC-based logic controller in the family of S7 controllers. We shall discuss these elements in brief and also discuss the various programming languages available in the Simatic Automation System environment.

4.2 The General Architecture of the Simatic Automation System

The Simatic Automation system is a range of coordinated components with uniform methods of configuring, data management and data transmission. [1]

4.2.1 Various Components/General architecture of the Simatic Automation System

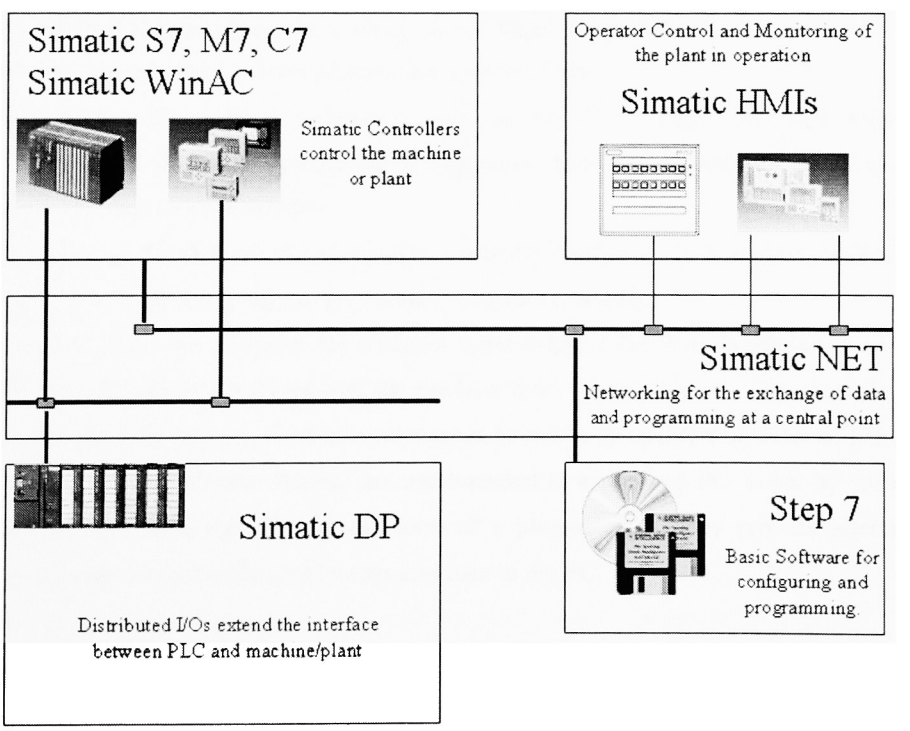


Fig 4.1 Components/Architecture of the Simatic Automation System

The entire Simatic Automation system consists of five major components, viz.

- Simatic Controllers
- Simatic Distributed I/Os (DP)
- Simatic HMI
- Simatic NET
- Step7

The Simatic Controllers control the machine or the plant. The Simatic Distributed I/Os provide the interface between the PLC and the machine or plant being controlled. Simatic NET provides the networking for the exchange of data and programming at a central point. Simatic HMIs are used for the operator control and monitoring of the plant in operation. Finally, Step 7 is the central development tool in the Simatic Automation system, consisting of the basic software for configuring and programming.

4.3 Simatic Controllers [1]

Simatic Controllers form the core of the automation system and help in controlling production machines, manufacturing plants or industrial processes.

There are three main families of controllers in the Simatic Automation System, viz. Simatic S7 controllers, Simatic M7 controllers and the Simatic C7 controllers

Simatic S7 programmable logic controllers form the basis of the automation system. There are three types of PLCs in this family,

S7-200 Micro PLC,

S7-300 modular mini PLC, for low-end an mid-range applications

S7-400 for high-end top-level performance requirements

PLCs consist of the CPU (central processing unit) and the I/O modules. The CPU stores the user program written in one of the PLC programming languages. The I/O modules provide the connection to the machine or the plant to be controlled.

The M7 families of controllers are AT-compatible computer modules, which expand the Simatic system by providing an open software platform for standard software products.

The C7 complete units are designed for machine control and offer PLC performance in a compact format. The operator controls and monitors the machine from the front panel. The controller inside the unit controls the machine using its built-in input/outputs or can be expanded with external I/O modules.

Controllers used with the Simatic WinAC are implemented in a standard PC, either by software in the form of soft PLC or as a slot PLC in the form of a plug-in card. They provide interfaces to PC applications and thus allow handling online process data in the PC.

4.5 Simatic Distributed I/Os

The Simatic distributed I/O system allows for installing the I/O modules connecting to the machine or the plant in the vicinity of the machine, at a distance from the PLC. The distributed I/Os are linked to the

central controller by the means of the Profibus network. Thus they provide an interface between the PLC and the plant/machine.

4.6 Simatic HMIs

HMI stands for Human Machine Interface. The Simatic HMIs are capable of providing text displays to the operator station. There are a variety of HMIs that are available in the Simatic Automation System such as Touch Panels, Text Displays, Push Button Panels, and Operator Panels.

4.4 Simatic NET

The Simatic NET links all the Simatic stations together and provides for the networking for the exchange of data and programming at a central point. A range of bus systems with different performance specifications make it possible to include a variety of devices in the network.

SIMATIC NET is the name of the communication networks connecting SIEMENS programmable controllers, host computers, workstations and personal computers.

SIMATIC NET includes the following:

- The communication network consisting of transmission media, network attachment and transmission components and the corresponding transmission techniques
- Protocols and services used to transfer data between the devices listed above
- The modules of the programmable controller or computer that provide the connection to the LAN (communications processors "CPs" or "interface modules").

To handle a variety of tasks in automation engineering, SIMATIC NET provides different communication networks to suit the particular situation. The topology of rooms, buildings, factories, and complete company complexes and the prevalent environmental conditions mean different requirements. The networked automation components also make different demands on the communication system.

4.7 Step 7 Software

The Step 7 software is the central development tool in the Simatic Automation system. It consists of the basic software for configuring and programming. The Step 7 software is used to configure the hardware, assign parameters to them, and program them. The Simatic programming languages and language representations integrated in Step 7 comply with the standard EN 61131-3 and IEC 131-3. The main functions of the Step 7 software are as follows

- Configuring the hardware Arranging the modules in the racks, assigning addresses to them and setting the module properties
- Configuring communication connections - Defining the communication partners and connection properties

- Writing user program for the PLC in the programming languages like LAD (Ladder Logic), STL (Statement Lists), and FBD (Functional Block Diagram), and testing the program online on the controller.

4.7.1 Combining the hardware and software using Step 7

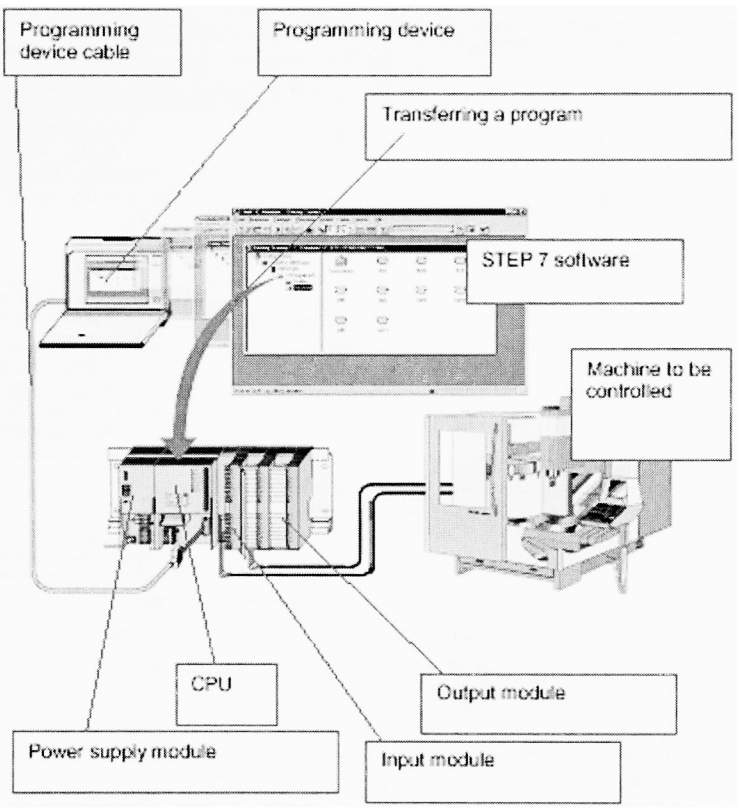


Fig 4.2 Using Step 7 to combine the hardware with the software

The Step 7 software plays the role of combining the plant hardware with the development software. The machine to be controlled is wired to the I/O module of the PLC. A Step 7 loaded computer forms a programming device, on which the program for controlling the process is developed. This programming device is connected to the I/O module or the CPU through a programming device cable like Profibus. The program developed in the Step 7 software is then loaded onto the CPU via the programming device cable and thus the machine control is enabled.

4.7.3 Basic Steps in Step 7 Automation

Whenever a control application is developed with Step7, there are a series of basic tasks. The following figure shows the tasks that need to be performed for any automation project using Step7.

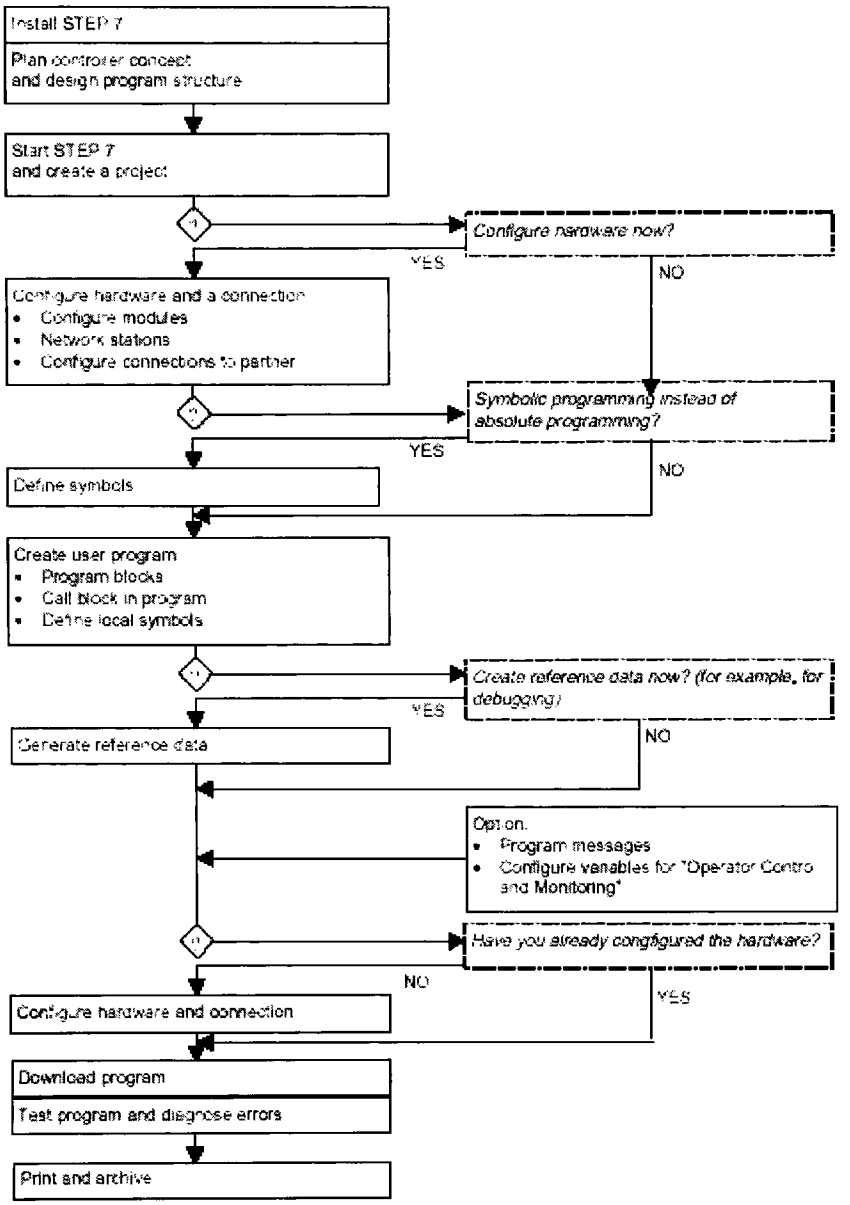


Fig 4.3 Basic tasks in automation project using Step 7

4.7.4 Simatic Manager Architecture

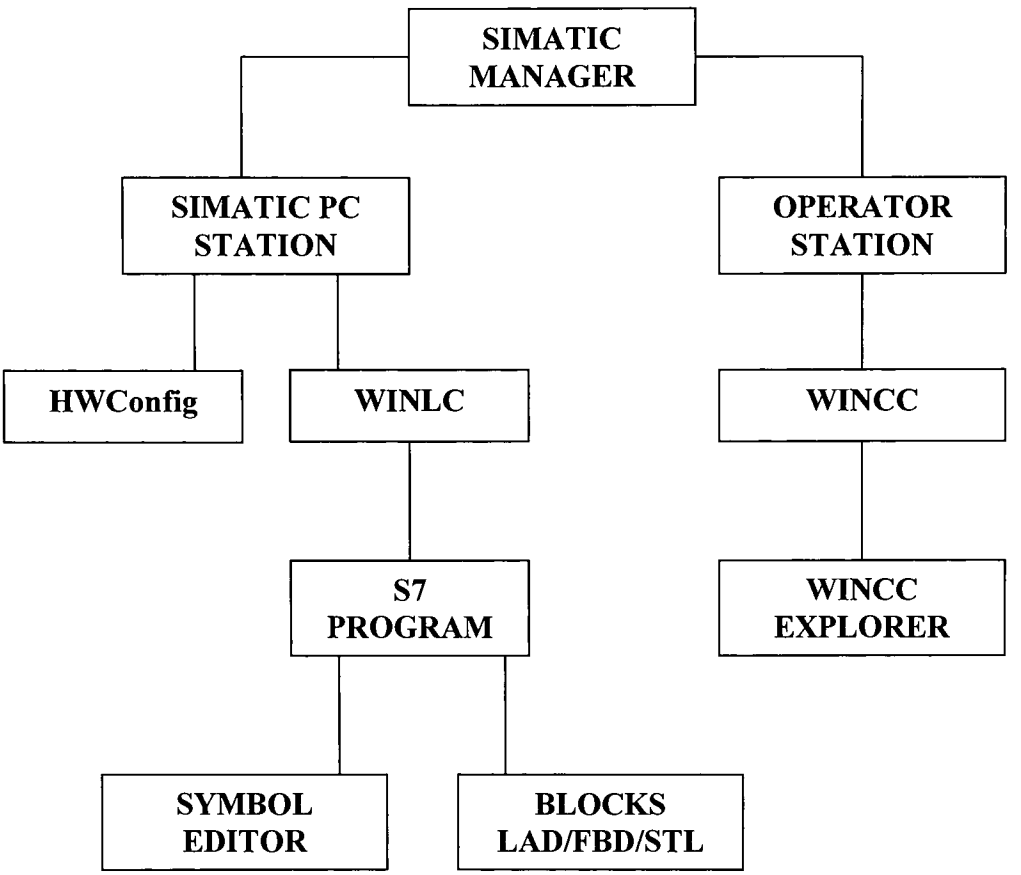


Fig 4.4 Basic architecture in a control application in the Siemens' PC Based Technology

4.7.4.1 Simatic Manager

The SIMATIC Manager is the basic application for configuring the hardware and programming logic in a control application. The SIMATIC Manager is the central window, which becomes active when STEP 7 is started. It manages all the data that belongs to an automation project. With Simatic Manager, users can work with the objects in the Step-7 world. The logical objects in the Simatic Manager represent the real objects of the plant/system. The project structure in the Simatic Manager is used to store and arrange all the data and programs in order. A brief project structure within the Simatic Manager is shown in the figure 4.4.

A new project for a control application is created within Simatic Manager. (Refer Appendix C for step by step instructions to create a project in Simatic Manager) Once a new project is created, a Simatic PC Station needs to be added to the project. This station contains the configuration and the parameter data of the hardware. Once a PC station is added to the project, it creates its own S7-Program. This S7 program comprises all the blocks with the programs necessary for controlling the application or the system.

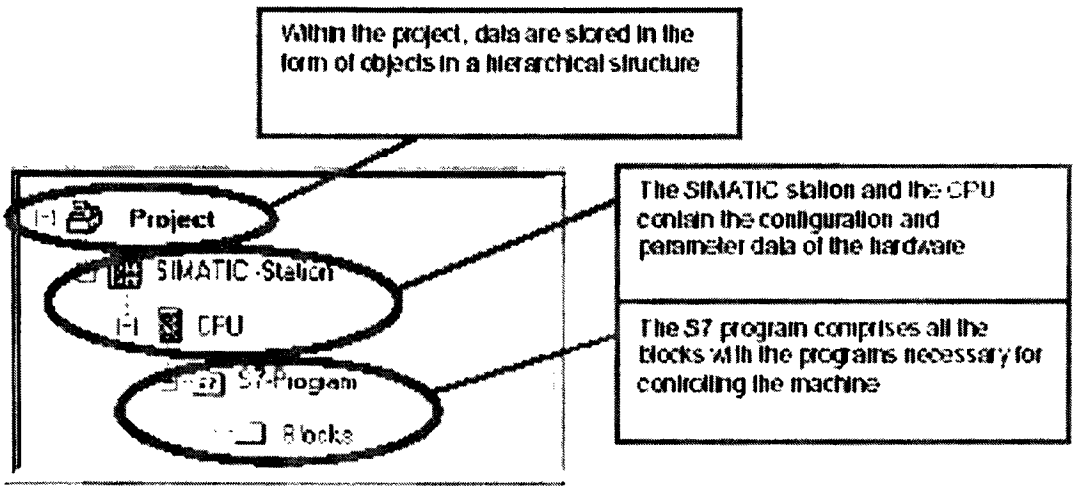


Fig 4.5 Project Structure within Simatic Manager

With Simatic Manager, the following functions can be performed

- Set up projects
- Configure and assign parameters to the hardware
- Configure hardware networks
- Program Blocks
- Debug and commission programs

A user can work in Simatic Manager in one of two ways

- Offline, without a programmable controller connected
- Online, with a programmable controller connected

4.7.4.2 Simatic PC Station

Simatic PC station represents the interface module that is used in a particular control application. It contains the configuration and the parameter data of the hardware.

4.7.4.3 WINLC

WINLC is the PC based controller in the family of S7 controllers. It provides process control from a computer thus facilitating a PC based control. It communicates with the Step 7 and the distributed I/O module viz. ET 200M over the Profibus network. Figure 4.6 shows the basic components of the WINLC. WINLC connects the PCI card CP5613 over the Profibus network to distributed I/O (say, ET 200M) that connects to the process or automation project.

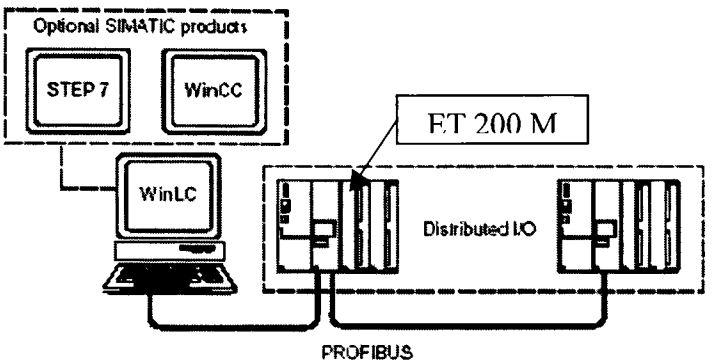


Fig 4.6 Components of WINLC

4.7.4.4 HWConfig

The HWConfig allows the configuration of the hardware that is used in the control application.

The term "configuring" refers to the arranging of racks, modules, distributed I/O (DP) racks, and interface sub modules in a station window. Racks are represented by a configuration table, which permits a specific number of modules to be inserted.

In the configuration table, STEP 7 automatically assigns an address to each module. The addresses of the modules in a station can be changed if the CPU in the station can be addressed freely (meaning an address can be assigned freely to every channel of the module, independent of its slot).

When the programmable controller starts up, the CPU compares the preset configuration created in STEP 7 with the actual configuration of the plant. Any errors are therefore recognized immediately and reported.

The configuration table for the hardware, once created for a specific hardware setup, can be imported into a .cfg file. This file can later be exported into a new project that uses the same setup, thus eliminating the need for performing the hardware configuration task every time a new project is created. (Refer Appendix B for a step by step instructions for configuring the hardware using HWConfig)

4.7.4.5 Symbol Editor

With the symbol editor, user can manage all the symbols used in an automation project. Symbol is a name defined by the user, taking syntax rules into consideration. Once defined, this name can be used in programming and in operating and monitoring.

In any control program, one has to work with addresses. These are inputs, outputs, and counter blocks for example. Every input and output has an absolute address predefined by the hardware configuration.

This address is specified directly; that is, absolutely. One can access the addresses absolutely (e.g. I1.0) or symbolically (e.g. Start). Symbolic addresses make use of names instead of absolute addresses. A control program can be made easier to read by using clearly descriptive names. When an S7 program is created, Simatic Manager also creates an empty symbols table. The user can open this table and then establish the symbols and assign the absolute addresses. There can be only one symbol table in an S7 program.

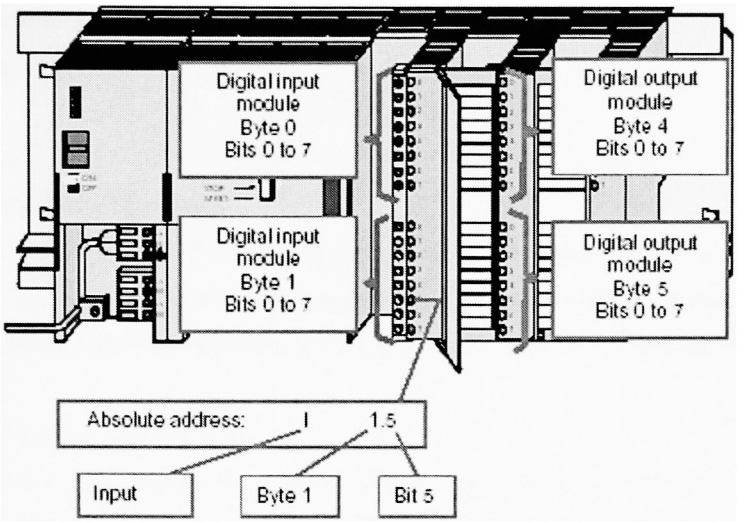


Fig 4.7 Absolute Addressing

When a symbol is established, an appropriate data type is also needed. This data type defines specific properties of the data that is hidden behind the symbol; essentially the representation of the data contents. For example, the data type BOOL characterizes a binary variable and the data type INT characterizes a digital variable whose contents represent a 16-bit integer.

4.7.4.6 S7 Program and Blocks

Once a PC station is added to the project, it creates its own S7-Program. This S7 program comprises all the blocks with the programs necessary for controlling the application or the system.

As described in the previous section, within a S7 program, logic blocks are created in standard languages like Ladder Logic (LAD), Statement List (STL) or Functional Block Diagram (FBD)

Blocks are objects in the S7 program, which contain the logic blocks, data blocks, user defined data types and variable tables. Logic blocks are the blocks that contain a part of Step-7 user program, for example an Organization Block or a Function Block. Organization Block forms the interface between the S7 and the user program. The sequence in which the user program should be processed is laid down here. On the other hand, a Function Block is a block that references a data block and contains static data (constant values). A Function Block allows user to pass parameters in user programs, which makes them suitable for programming complex functions that are required frequently. Data Blocks are areas in the user program that contains user data.

4.7.4.7 Programming Languages in Step 7

Programming languages are used to develop user programs. There are several PLC programming languages available in the Step 7 package. The Step 7 software supports most of the programming languages that we described in Section 3. The languages supported in the Step 7 package are,

- LAD, Ladder Logic
- FBD, Functional Block Diagram
- STL, Statement List
- Graph
- Hi-Graph
- Structured Control Language

An important feature of the Step 7 software is that the users can create program logic in LAD, FBD or STL and get the logic code in the other language. This gives the user the choice of writing the logic in his language of choice, and at the same time get the logic in the other two languages. [1]

4.7.4.8 Operator Station and WINCC

An Operator Station is an object that includes within a S7 project, a SIMATIC component such as WINCC, as an application for automation tasks. These objects are configured within the SIMATIC PC Station. An operator station facilitates for the transfer of the symbols used in the S7 program, as tags that can be used in the WINCC. Thus it acts as a link between WINCC and the Step 7.

WINCC is a Supervisory Control and Data Acquisition Software HMI (Human Machine Interface). It redefines the HMI as an integration platform for information based manufacturing. Developed with the cooperation of Microsoft® and the automation expertise of Siemens®, WINCC is an important tool in Siemens' PC based technology. PC-Based Automation involves the philosophy of bringing the shop floor and Corporate Level decisions closer together by the use of industry standards. WINCC meets this goal by deploying these open standards and open interfaces. WINCC is a tool for process visualization and information availability. Plant Floor Integration is realized with WINCC through its open connectivity architecture. In addition to its suite of internal drivers, WINCC can communicate with a variety of automation equipment using the OPC (Object Linking and Embedding for Process Control), which is an industry standard for Industrial Communications. WinCC can exchange information with plant floor hardware and software devices. It functions as both an OPC Client and an OPC Server. This enables WinCC to communicate with hardware devices such as PLCs, Drives, and Sensors. WINCC can also exchange information with various software devices such as Soft Controllers, and other HMIs.

The WinCC Explorer represents the topmost level within WINCC. Within the WinCC Explorer, a run-time module can be configured and started. From a functionality point of view, WinCC Explorer contains all the management functions for the entire WinCC system. All of the modules and editors available in WINCC are started from here.

Shown below in figure 4.5, is the general project structure for the control application project within WINCC Explorer.

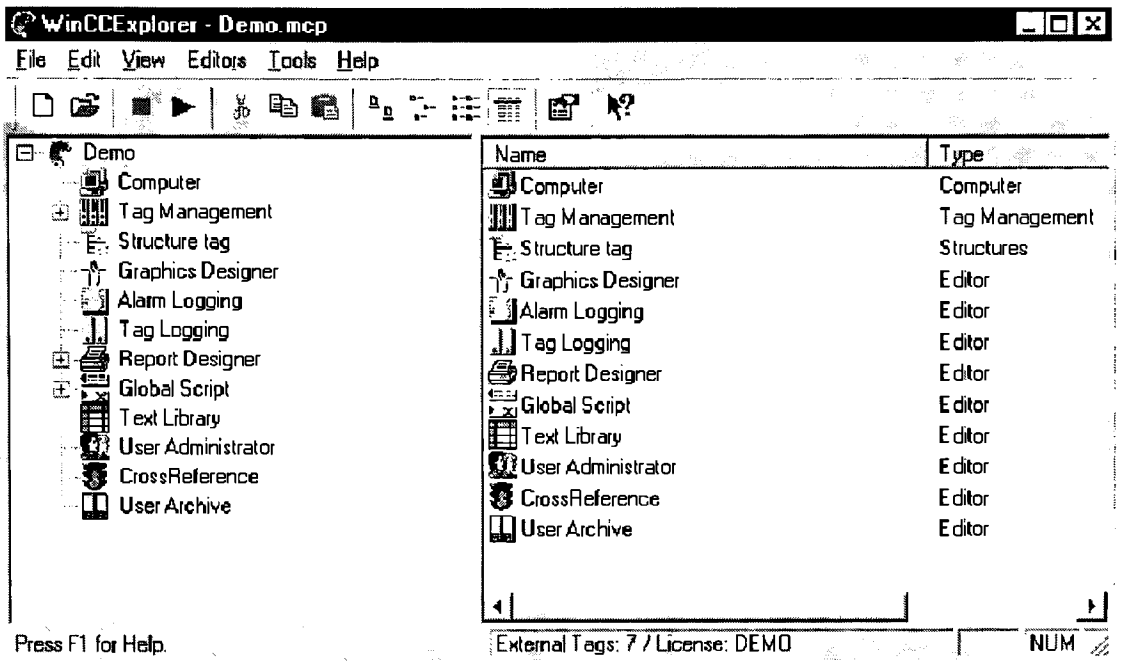


Fig 4.8 Project Structure within WINCC Explorer

The three main components of the project structure are

1. Computer,
2. Tag Management, and
3. Editors

a. Computer

The Computer component has all the workstations and servers assigned to a project. Only the control stations that are included in the configured computer list of the project can switch to runtime mode as well as configure mode. The Computer component has properties like Startup, Parameters and Graphics Runtime. With these properties, the startup behavior of the runtime for the local computer can be changed. Similarly, the language to be used in the runtime, window attributes, keyboard assignments and the behavior of cursor control can be defined.

b. Tag Management

The Tag Management component manages all the channels, logical connections, process tags, internal tags, and tag groups.

i. Tags

WINCC tags are the central elements for accessing process values. Within a WINCC project, they receive a unique name and a data type. A logical connection is assigned to a WINCC tag. This connection determines which channel delivers the process values to the tags using which connection. The WINCC tags are stored in a project wide database. When a WINCC machine starts, all of the tags belonging to the project are loaded and the corresponding run-time structures are set up.

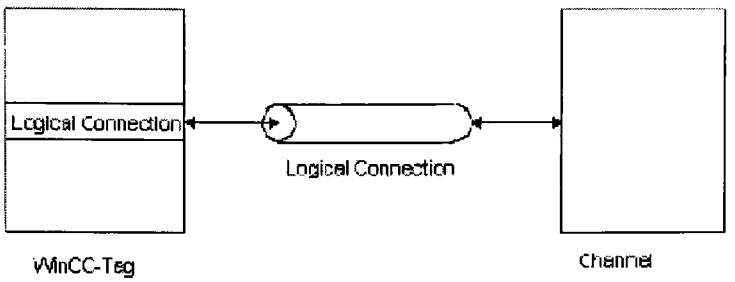


Fig 4.9 WINCC Tag

A WINCC tag is a data cell that is unique within a project.

The logical connection handles the communication with the PLC that enables access to the external tags in the process. The name of the connection is unique. For the logical connection, the tags of the remote PLC must be defined subsequently. Individual tags as well as groups of tags can be configured. The address information necessary for configuration always refers to the PLC that wants to access the tag by means of the logical connection. In addition to this, communication parameters for this connection (such as the bus address of the PLC) must also be specified. These parameters depend on the channel unit (driver connections) of the communication driver being used.

The Channel Unit handles the access to the communication interface of the operating system. It is a part of the communication driver.

ii. Communication Drivers

To access external tags, the communication driver for the communication with the remote PLC has to be installed. Depending on the driver, the communication driver contains one or more channel units. By means of the operating system interface and the hardware driver, a channel unit supports the hardware component (communication port or PC module) necessary for communication. A communication driver can support several channel units of the same type.

iii. Updating tags with Process Values

The WINCC Tag Management is responsible for providing the WINCC tags with the process value at run time. The tag management provides the process values to its WINCC tag and transfers the tags to be updated to their logical connection and thus to the appropriate channel unit. The channel unit executes the necessary communication steps by means of its process bus in order to assign values to all tags. Thus, the WinCC Tag management requests process values at run time from the remote PLC through the logical connection. The channel unit carries out the communication steps necessary for requesting the process values by means of the channel-specific connection and thus provides the WinCC data manager with process values. The data,

which are read in, are stored as a process image in the RAM of the computer. All WinCC components access this process image.

iv. Types of Tags

Within a project, tags can either be created as independent tags within a connection, or as tags in a tag group.

Generally there are two types of tags, internal tags and process tags. An internal tag is a tag that has no addressing at the Actuator-Sensor level. They are used to store general information like current date, time, current layer etc. In addition, internal tags enable a transparent data exchange between applications in order to implement inter-process communication in a centralized and optimized manner.

On the other hand, a process tag is a tag that has an address at the Actuator-Sensor level. Process tags contain information about the process values. They are linked to a logical connection. In order to mirror the address information of the various Actuator-Sensor (AS) systems, process tags consist of a general section, which contains information like name, type, and a connection-specific section whose interpretation depends on the logical connection.

c. WINCC Editors

Following is a list of the editors available in WINCC,

- Graphics Designer
- Alarm Logging
- Tag Logging
- Report Designer
- Global Scripts
- Text Library
- User Administrator
- Cross Reference
- User Archives

The Graphics Designer editor is a vector-oriented drawing program for creating process pictures. Numerous graphic objects, which are contained in an object and styles palette, can also be used to create complex process pictures. Dynamics can be added to individual graphic objects by means of action programming.

Alarm Logging provides displays and operating options for acquiring and archiving results. The message blocks, the message classes, the message type, the message display, and the report can be selected. The System Wizard and the configuration dialogs provide support during configuration. To display messages in runtime, the Alarm Control, which is contained in the object palette of the Graphics Designer, is used.

Tag Logging is used for acquiring data from running processes and for preparing them for display and archiving. The data formats of the archives and the acquisition and archiving timers can be freely selected. The display of the process values is made via the WinCC Online Trend and Table Controls, which display the data in trend and table form, respectively.

The Report Designer is an integrated reporting system for timer- or event-controlled documentation of messages, operations, archive contents, and current or archived data in the form of user reports or project documentation in freely selectable layouts. It provides a comfortable user interface with tool and graphic palettes and supports the various report types. Various standard system layouts and print jobs are available.

Global Script is the generic term for C functions and actions which, depending on type, can be used within a given project or within numerous projects. Scripts are used to configure actions to objects. They are processed using a system-internal C compiler. Global Script actions are used at runtime in process execution, which is initiated by a trigger.

In the Text Library, you can edit texts, which are used in runtime by the various modules. The foreign language output texts are defined in the Text Library for the configured texts. These are then output in the selected runtime language.

The User Administrator is used for allocating and controlling access authorizations of the users for the individual configuration- and runtime-system editors. When a user is set up, the access rights for the WinCC functions are set and individually assigned to the respective user. Up to 999 different authorizations can be allocated. User authorizations can be allocated at system runtime.

Cross Reference is used to find and display all points of use for objects, e.g. tags, pictures, and functions. With the "Linking" function tag names can be changed without causing inconsistencies in the configuration.

WinCC User Archives is a database system. Data from technical processes can in this way be continuously stored on a server PC and displayed online in runtime. Moreover, recipes and set point assignments for the connected controls can be stored in the User Archives and passed on to the controls as necessary.

References:

[1] Berger, Hans. Automating with SIMATIC. SIEMENS 2000.

Section 5. Description of the CAMCELL architecture [1]

5.1 Description of the CAMCELL

The CAMCELL is an integrated manufacturing facility that has evolved over the past several years within the department of Industrial and Manufacturing Engineering at Rochester Institute of Technology. It is a completely automated and fully operational production facility. In order to completely understand the various aspects of the CAMCELL, it has been described from the following three different perspectives in this section:

1. Manufacturing and Material Handling
2. Product/Process Flow
3. Computer Hardware Architecture

5.2 Manufacturing and Material Handling

The facility occupies about 600 sq.ft. in area and is comprised of two CNC machining centers (a mill and a lathe), a vision/inspection station and a load/unload dock. These four stations and four assembly stations are connected by two-closed loop conveyor system and are the eight main functional stations in the facility. The two conveyors are placed end to end in an "L" shape configuration.

The main conveyor loop connects the four stations identified above while the secondary conveyor that is perpendicular to the main loop connects the assembly stations. These conveyors act as a buffer and material transport system for up to twenty 8 in. x 8 in. size pallets, which hold the fixtures and materials needed in the CAMCELL operations.

Each pallet carries a Radio Frequency tag for identification that is tracked by 2 NAMCO radio frequency identification system placed on the two conveyors. Each tag has capability to write 16 bytes of data. The tags can be re-written in order to maintain additional work in process (WIP) status.

An industrial grade robot provides material handling for the mill center and the vision station. A robot and an elevator mechanism developed in house provide the load/unload capability at the lathe center. In addition to this, one Intellidex robot, one IBM robot and two Adept robots are used at the four assembly stations.

In summary, following is a list of primary manufacturing and material handling equipment in the facility:

- Two 10 feet long BOSCH conveyors
- 3-axis TERCO CNC vertical mill
- 19-inch ORAC CNC lathe with 8 station turret
- IRI vision inspection system
- 2-axis elevator
- 3-axis pneumatic robot (manipulator)
- NAMCO radio frequency identification system

- Two 5-axis Intellidex robots
- Two ADEPT robots
- One IBM robot

5.3 Product/Process Flow

As explained in the previous perspective, the two major production equipments in the facility are the two machining centers viz. a CNC lathe and a CNC Mill. Thus, facility is capable of producing any type of parts that can be manufactured by a lathe or a mill or combination of those two. Primary material used in manufacturing the parts is a special wax like material. However, the equipment is also capable of machining metal parts. Some of the sample parts that are produced in the facility are a screw, a nut, and an RIT key-chain. The facility illustrates the operations of a flexible manufacturing system (FMS) in scheduling of machines, assignment of pallets and flexibility in making any parts in any order with necessary setups and machine programming handled automatically by the CAMCELL control software. The materials and the tooling fixtures for parts are being produced are loaded at the dock. These will travel to the lathe and/or mill centers for machining and the vision station for inspection in order dictated by the process steps. All finished parts return to the dock for unloading. For example, if the part to be manufactured is a screw, then a cylindrical material will be first loaded at the dock. Then it would be routed to the lathe where the threads of the screw will be cut. Then the part will be routed to the mill center to cut the slot. Finally the part will be routed back to the dock for the unloading. Thus, this would be a typical material flow for manufacturing of the part. All equipment and the material transport system in the facility are fully integrated into a completely computer controlled manufacturing facility.

5.4 Computer Hardware Architecture

As defined by the National Institute of Standards and Technology (NIST), there is a five-level hierarchy in manufacturing automation [1]. RIT's CAMCELL computer hardware architecture has evolved into one that already implements such hierarchy. Most of the manufacturing equipment and related computer hardware used in the facility were acquired based on their individual merits for their use outside the CAMCELL environment for instructional purposes.

The compatibility among systems for total integration into the CAMCELL is accomplished through many in house developed hardware interfaces and control software. The five-level computer hardware architecture in the CAMCELL includes following five levels viz. tool level, station level, cell level, center level and plant level. At the **tool level**, several data acquisition systems are used to monitor and manipulate the status of various elements in the system such as the latches and sensors. At the **station level**, there are several machining centers, robots and vision systems that are primarily involved in the manufacturing of parts. The main activity at the **cell level** is routing jobs through the system for all low level stations. It deals with information and material handling in order to deliver right parts and material to the right place at the right time.

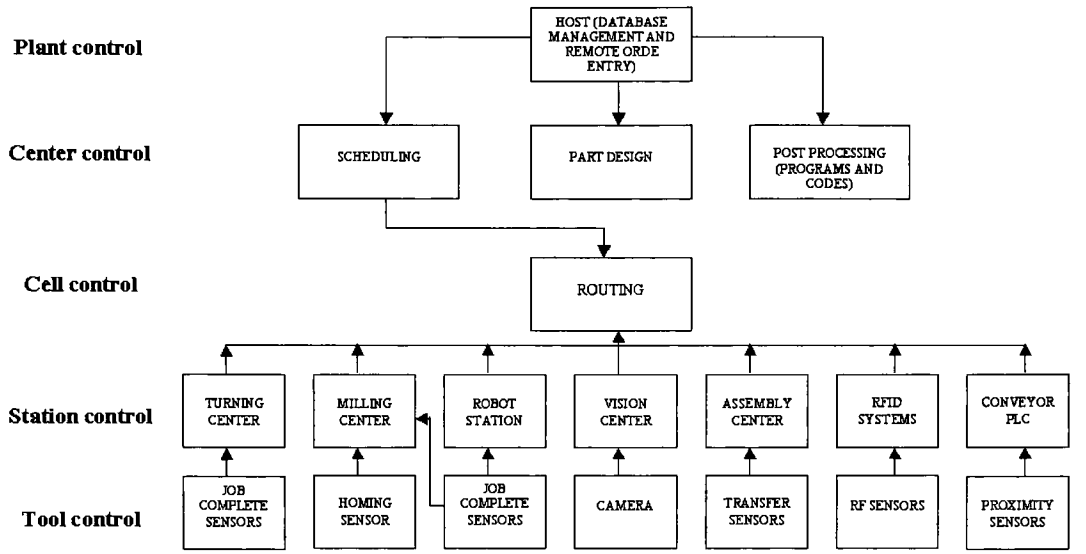


Fig 5.1. CAMCELL Hardware Architecture

At the **center level** scheduling of orders take place. This layer looks into job priorities, machine availability, personnel availability, material availability as well as machine capacity limits. At the uppermost level viz. the **plant level** orders from customers are entered. Other tasks dealt by this level are parts catalog management, initiation of design work, price negotiations, generation of material acquisitions, inventory management etc.

The low-level station control and device manipulation are achieved via Programmable Logic Controllers (PLCs), Data Acquisition Systems and computers. The lathe and the mill, during their machining cycle, operate under their own CNC controllers. However, their manufacturers originally designed these machines as stand alone units for manual keyboard control. They have been modified in the CAMCELL to allow remote computer controlled operation for automatic downloading of NC codes and device manipulation which is essential to integration of the mill and lathe machines into a flexible and automated manufacturing environment. The load/unload manipulator of the lathe has a dedicated computer for control. All the robots involved in the assembly operations have a controller of their own. The Intellidex robot has a programmable controller of its own. The IRI vision system is operated using a UNIX based microcomputer system. The NAMCO Radio Frequency Identification System is a host programmable intelligent peripheral. The results are delivered from the scheduler to the router on a once-a-day or once-a-shift basis, whereas the station level and tool level controllers have a real-time interaction between each other.

The integration of manufacturing systems can range from a relatively self contained machining cell to large automated factories which include support functions such as planning and scheduling. The type of architecture or combination of architectures used depends a great deal upon the application. Many of the

CIM systems in use today depend upon a combination of networked and hierarchical architectures. A hierarchical design has been used in the CAMCELL.

The hierarchical design architecture consists of varying levels of controls based upon the communications paths established between computers to form a control structure in the shape of a pyramid. At each level within the control structure, responsibilities are designated based upon the task to be performed and the time dependencies of that task. In this way the lowest level within the hierarchy, machine controllers or PLCs, is able to directly interact with the machines, which requires response time in the range of milliseconds. Higher-level computers are then able to co-ordinate a group of machine controllers in real-time. In this arrangement, the higher-level computer is directly linked to each of the lower level computers and sends out control messages to and receives some form of acknowledgement from them upon completion of the delegated tasks.

The hierarchical systems can be configured with powerful, inexpensive computers and, hence, are extremely cost effective. A networked architecture may be used to connect various manufacturing shop floor controllers in a larger facility, with the engineering planning functions, scheduling functions and business costing programs completing the idealized CIM system. The advantages of the hierarchical system are its quick response times, low cost and high reliability because they are not subject to unknown process sequence. Reliability can be designed into these systems at key points in the control structure. Hierarchical architectures are ideally suited for cell control activities and as such are found more frequently within complex flexible manufacturing systems.

Reference:

[1] Dr. Paidy. S.R., Dr. Reeve, Richard, "Software Architecture for a Cell Controller", HICCS, 1990.

Section 6. Control of the CAMCELL Material Handling System

6.1 Introduction

In this section, we will describe the components, function and the control of the material handling system used in the CAMCELL. The main components of the material handling system are two BOSCH conveyors placed end to end to form an L-shape, eight pneumatic latches and eight sensors at the eight workstations, four transfer lifts and four gates at the transfer lifts. In addition to these components, there are six-to-fourteen pallets that circulate along the two conveyors, carrying the unprocessed, semi-processed or processed material. The primary function of the material handling system is to carry the raw material, semi-processed material, and finished material along the conveyor from one station to another. In addition to this, the material handling system also has the function of stopping a particular requested pallet whenever a particular station makes a request for it. The Programmable Logic Controller used in the control of the material handling system is the CTC PLC from Control Technology Inc

6.2 Components of the Material Handling System

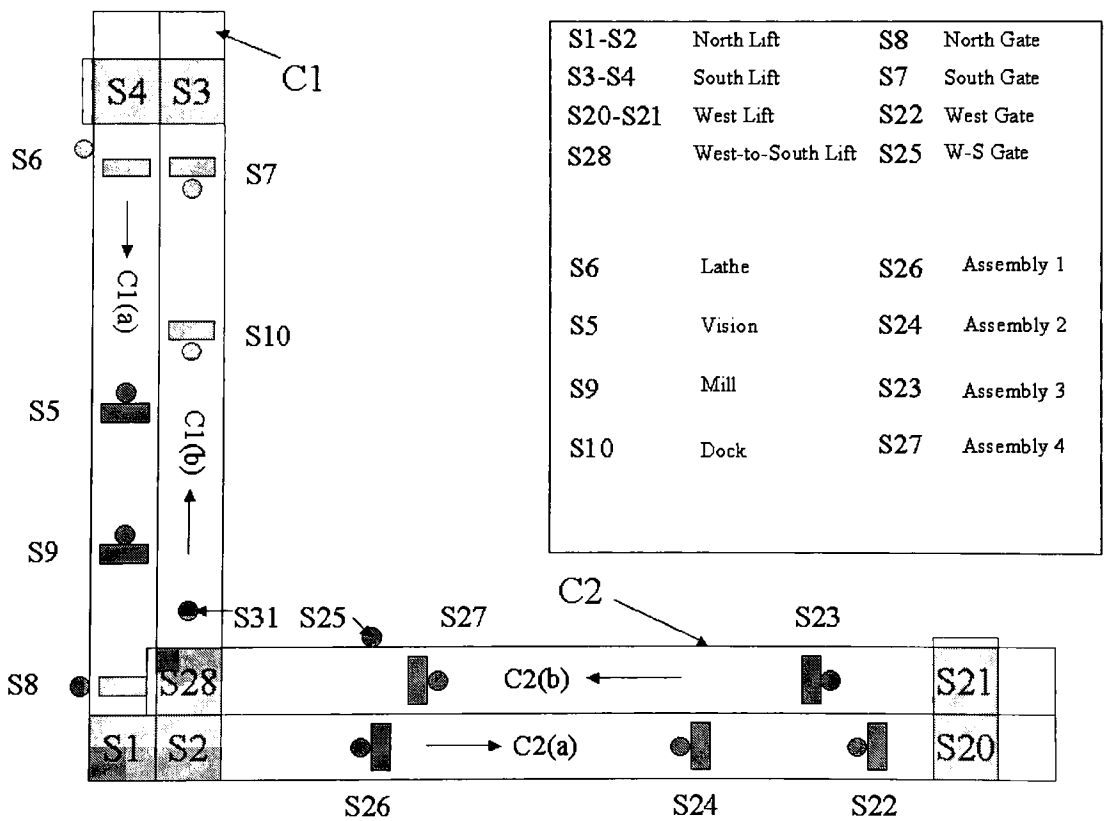


Fig 6.1 Schematic of the CAMCELL Material Handling System

Shown above in figure 6.1 is the schematic of the CAMCELL material handling system. The schematic shows the main components of the material handling system, which can be identified as follows,

- Two conveyors, placed end-to-end to form an L shape,
- Four transfer lifts,
- Four transfer gates,
- Eight pneumatic latches and eight sensors at the eight workstations.

The two conveyors, C1and C2, carry the pallets from one station to another. Each conveyor has two halves, C1 (a), C1 (b), C2 (a) and C2 (b), each moving in opposite directions to each other.

The four transfer lifts, viz. the South Lift (S3-S4), North Lift (S1-S2), West Lift (S20-S21) and the West-to-South Lift (S28), transfer the pallets from either one half of the conveyor to the other half, or from one conveyor to the other. Each lift, except for the West-to-South lift, has a load station and an unload station. The load station for the lift is the one through which the pallet enters the lift, whereas the unload station for the lift is the one through which the pallet leaves the lift. Thus stations S1, S3, and S20 are the load stations and stations S2, S4, and S21 are the unload stations for the North lift, South lift, and the West lift respectively. The West-to-South lift, on the other hand, has only one station.

There are four transfer gates, viz. the South Gate (S7), North Gate (S8), West Gate (S22) and the West-to-South Gate (S25), at each of the four transfer lifts that regulate the traffic at the transfer lifts. Each gate comprises of a pneumatic latch to hold or release the pallet and a sensor to detect the presence of the pallet. The gates hold the pallets from entering into the lift, while a pallet is being transferred by the lift, thus preventing the blocking at the lifts. The latches at the eight workstations perform the function of stopping and releasing the requested pallet at the station. The sensors at the workstations detect the presence of the pallet at the station.

6.3 Control of the CAMCELL Material Handling System

The CAMCELL material handling system is controlled by the CTC-2200 PLC from Control Technology Corporation. The control program for the material handling system is written in the Quickstep programming software from the Control Technology Corporation.

The control program for the material handling system consists of two main loops, viz. the transfer loops and the pass through loops. The transfer loops transfer the pallets from one half of the conveyor to the other, or from one conveyor to the other. On the other hand, the pass through loops transfers the logic of the inputs and the outputs on the CTC PLC to the Siemens PLC. With this pass through, the outputs on the CTC PLC are used as inputs in the Siemens PLC and the inputs in the CTC PLC are used as outputs in the Siemens PLC. The inputs and the outputs used in the program are shown in figure 6.2 and 6.4 respectively. The list of flags used in the program is shown in figure 6.3

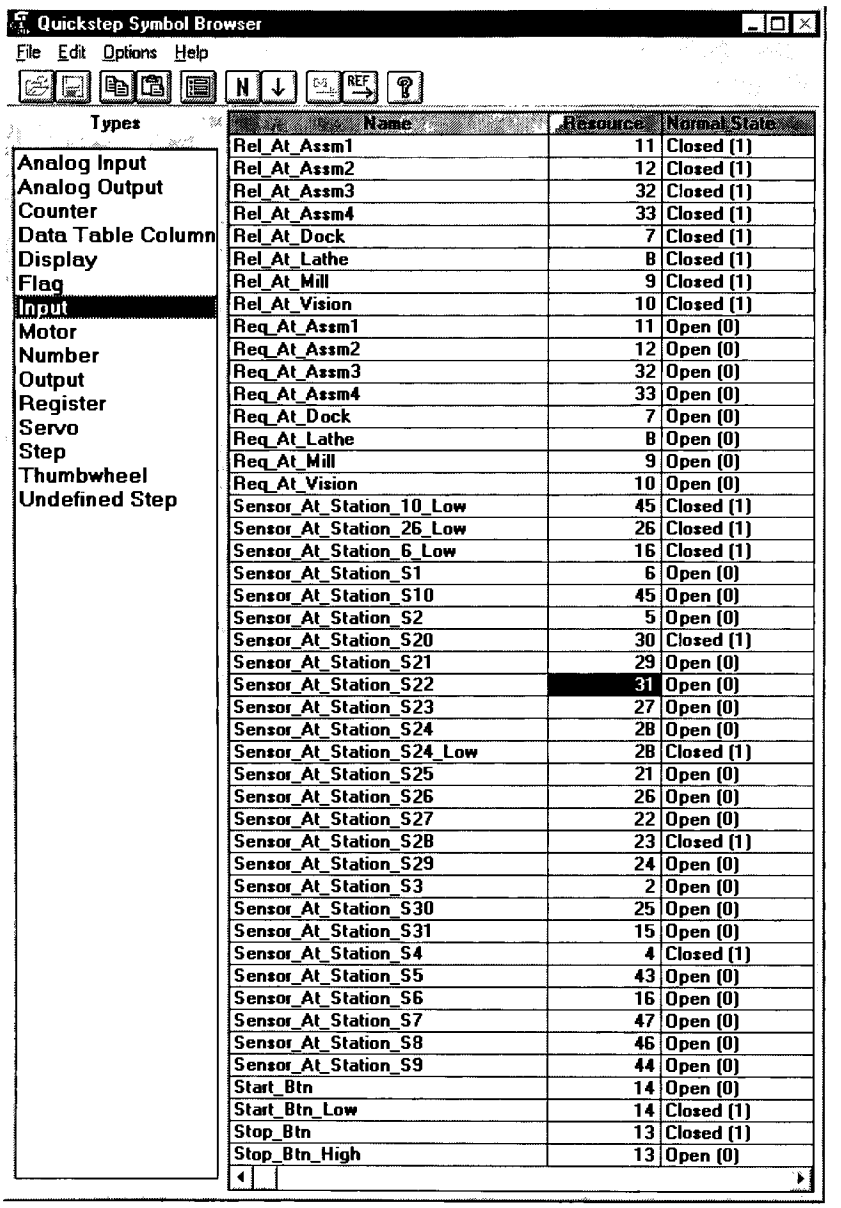


Fig 6.2 List of Inputs used in the Control Program

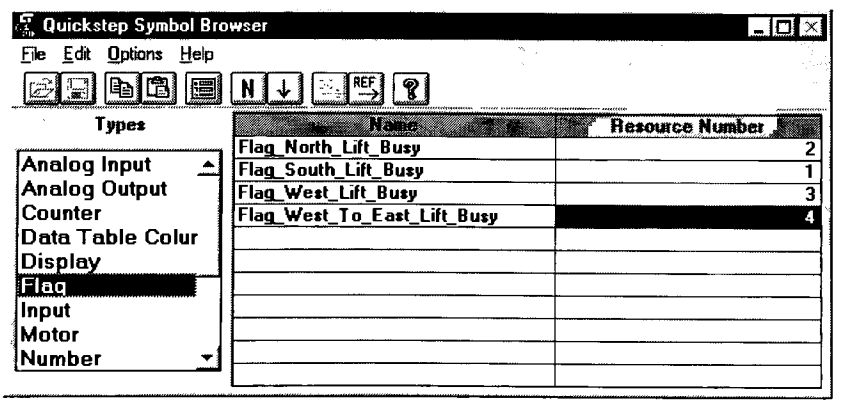


Fig 6.3 List of Flags used in the Control Program

- If the flag for the transfer lift is clear, the latch at the transfer gate is lowered to release the pallet. At this point, the flag for the transfer lift is set to busy.
- Next, the sensor at the load station of the lift is monitored. When the sensor at the load station of the lift goes high, it indicates the presence of the pallet at the load station of the pallet. At this point, the lift is raised to transfer the pallet from one half of the conveyor to the other half, or from one conveyor to the other. At the same time the latch at the transfer gate is raised to prevent the proceeding pallets from entering into the lift.
- Next, the sensor at the unload station of the lift is monitored. When the sensor at the unload station of the lift goes high, it indicates that the pallet has been transferred. At this point the lift is lowered and the pallet is thus allowed to leave the lift.
- Next, the sensor at the station following the unload station of the lift is monitored. When the sensor at this station goes high, it indicates that the transfer is complete. At this point the flag for the transfer lift is set to clear and the latch at the transfer gate is lowered to allow the proceeding pallet to enter the lift.

The transfer logic for the West-to-South transfer lift is slightly different, since there is only one station in the lift. The transfer logic for the West-to-South transfer lift is as follows,

- The latch at the assembly station #4 (S27) is raised to stop the pallet. At this point the sensor at the West-to-South transfer gate (S25) goes high.
- Next, the flag for the West-to-South transfer lift is monitored for the clear condition.
- When this flag is cleared, the lift is raised and the latch at station S27 is lowered to allow the pallet to enter the lift.
- Next, the flag is set to busy, and the latch at station S27 is raised, thus preventing the proceeding pallet from entering into the lift.
- Next, the sensor at the West-to-South transfer lift is monitored. When the sensor at the lift senses the pallet, the lift is lowered, thus allowing the pallet to leave the lift.
- Next, the sensor at the station S31 is monitored. When this sensor goes high, it indicates that the transfer is complete. At this point the flag for the transfer lift is set to clear and the latch at the transfer gate is lowered to allow the proceeding pallet to enter the lift

6.3.2 Pass Through Loops

The pass through loops, transfer the logic of the inputs and the outputs on the CTC PLC to the Siemens PLC. With this pass through, the outputs on the CTC PLC are used as outputs in the Siemens PLC and the inputs in the CTC PLC are used as inputs in the Siemens PLC.

The pass through between the CTC I/O lines and the Siemens I/O lines is accomplished through the registers in the Quickstep Software. With the help of registers, the value of an input line of the CTC PLC is stored into an output line of the CTC PLC. The input line of the CTC PLC is physically wired to the output line of the Siemens output module. With this, the output line of the CTC PLC can be used as an output line in the Siemens PLC. Similarly, with the help of registers, the value of the input line of the

CTC PLC is stored into an output line of the CTC PLC. This output line is physically wired to the input line of the Siemens input module. With this, the input line of the CTC PLC can be used as an input line in the Siemens PLC. Figures 6.6 and 6.7 show the list of pass through lines between the CTC PLC and the Siemens PLC

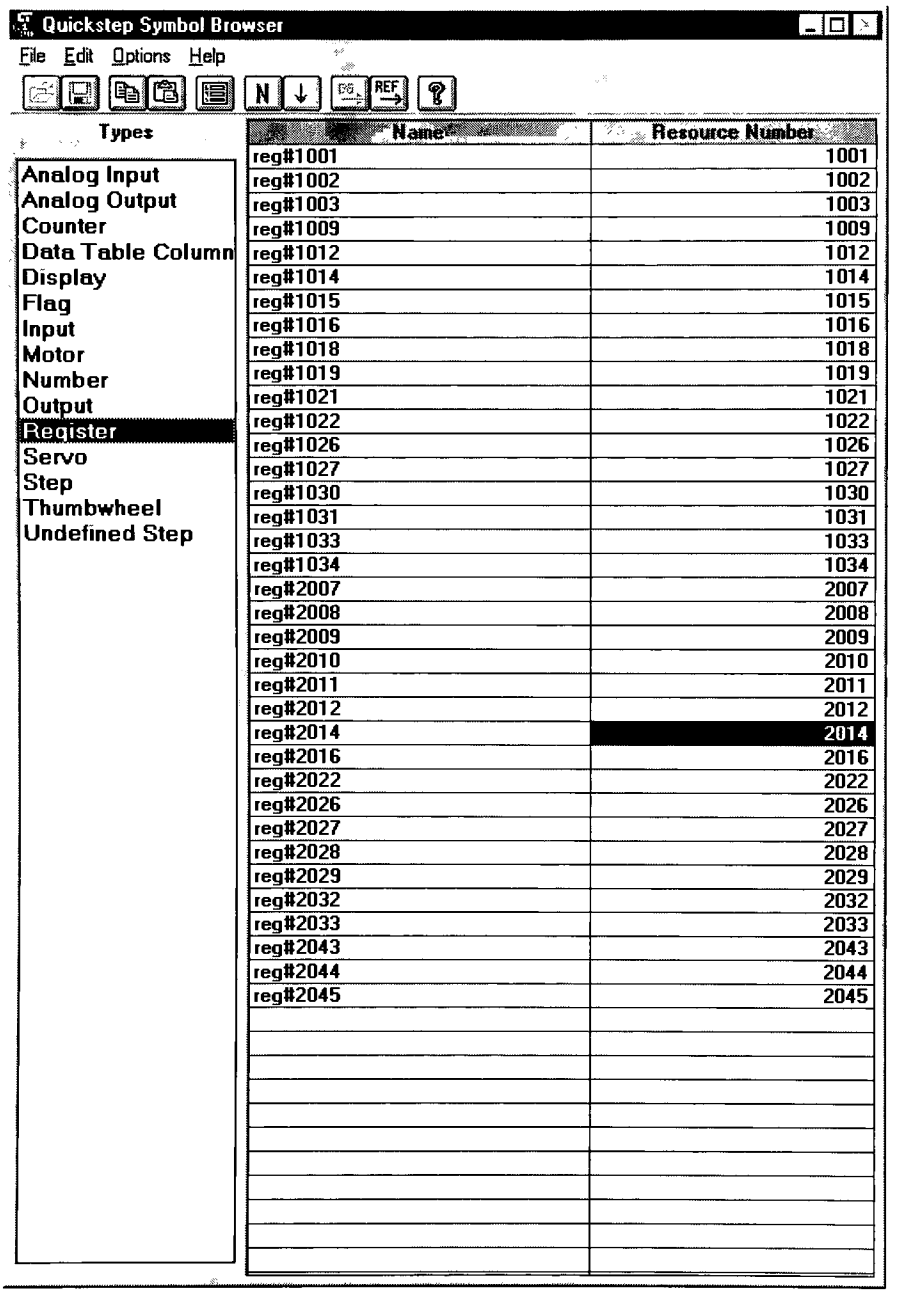


Fig 6.5 List of registers used in the Control Program

CTC Input	Line#	Register #	CTC Output	Line#	Register #	Siemens Input	Address
Sensor At Station S10	45	reg#2045	Set_Output_12	12	reg#1012	Pallet Passed Dock	I2.0
Sensor At Station S6	16	reg#2016	Set_Output_14	14	reg#1014	Pallet Passed Lathe	I2.2
Sensor At Station S9	44	reg#2044	Set_Output_15	15	reg#1015	Pallet Passed Mill	I2.3
Sensor At Station S5	43	reg#2043	Set_Output_16	16	reg#1016	Pallet Passed Vision	I2.4
Sensor At Station S26	26	reg#2026	Set_Output_18	18	reg#1018	Pallet Passed Asembly1	I2.6
Sensor At Station S24	28	reg#2028	Set_Output_19	19	reg#1019	Pallet Passed Asembly2	I2.7
Sensor At Station S23	27	reg#2027	Set_Output_30	30	reg#1030	Pallet Passed Asembly3	I3.1
Sensor At Station S27	22	reg#2022	Set_Output_31	31	reg#1031	Pallet Passed Asembly4	I3.2

Fig 6.6 List of Pass through Outputs

Siemens Output	Address	CTC Input Line#	Register #	CTC Output Controlled	Register #	CTC Output Line#
Request Assembly 1 Output	Q1.3	11	reg#2011	Latch/Release At Station S26	reg#1021	21
Request Assembly 2 Output	Q1.2	12	reg#2012	Latch/Release At Station S24	reg#1033	33
Request Assembly 3 Output	Q0.0	32	reg#2032	Latch/Release At Station S23	reg#1034	34
Request Assembly 4 Output	Q0.1	33	reg#2033	Latch/Release At Station S27	reg#1022	22
Request Dock Output	Q0.6	7	reg#2007	Latch/Release At Station S10	reg#1002	2
Request Lathe Output	Q0.7	6	reg#2008	Latch/Release At Station S6	reg#1003	3
Request Mill Output	Q1.0	9	reg#2009	Latch/Release At Station S9	reg#1001	1
Request Vision Output	Q1.1	10	reg#2010	Latch/Release At Station S5	reg#1009	9

Fig 6.7 List of Pass through Inputs

Section 7. CAMCELL Automation

7.1 Introduction

In this section, we shall discuss the CAMCELL automation system in detail. CAMCELL is a manufacturing cell that simulates a flexible manufacturing system. The two functional modules viz. the Pallet Controller and Inquire, implemented in this study, will be discussed in details. The WINCC tags used in the CAMCELL automation system are discussed. These tags include tags used for the Inquire module, tags used for the Pallet Controller module and tags shared by the two modules. Next, we discuss the function of the Inquire module, the WINCC screen developed for the Inquire module and the control program for the Inquire module. We also discuss the functions, WINCC screen and the control program for the Pallet Controller module.

7.2 Simulation of a Flexible Manufacturing System

CAMCELL is a manufacturing cell that simulates a flexible manufacturing system. As described in Section 5, it has eight stations served by a L-shaped material handling system. There are up to fourteen pallets that circulate in the system under the control of the CTC PLC. Figure 7.2 shows the schematic of the material handling system along with other data to be described later.

A software architecture based on NIST's five level hierarchy was discussed in [1] and further enhanced in a companion thesis work [2]. Specifically it comprises of functional modules for order entry, scheduling and routing. The routing module (Router) is responsible for sending specific part orders to each of the eight stations based on the process planning steps for the part. Each station from its process sequence data for the specific order learns and requests specific pallets (with the appropriate raw material and fixtures on them). These requests are funneled into a task named Pallet Controller. This task works with another task named Inquire.

In this study, the above-mentioned modules, viz. Pallet Controller and Inquire are implemented. Here, we use the OPC Client-Server methodology to implement these two modules on two different computers. The computer to which the Siemens PLC is connected is configured as the OPC Server. This computer runs the Inquire module and contains the repository of all the 'tags' of the project. Tags are the central elements for accessing process values and are available to the OPC, Step7 and WINCC. The Pallet Controller module is implemented on a different computer that is configured as an OPC Client. This client computer accesses the tags resident on the OPC server via the OPC channel. The figure below shows the brief schematic of the OPC Client-Server architecture.

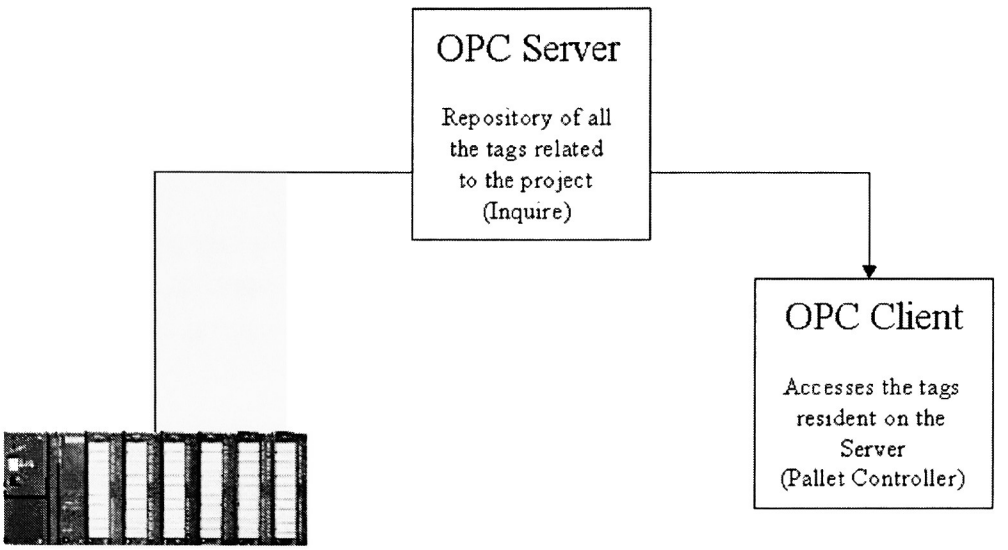


Fig 7.1 Schematic of the OPC Client-Server Architecture used in the CAMCELL

7.3 WINCC Tags used in the CAMCELL control software

As described in Section 4, WINCC tags are the central elements for accessing process values. In the CAMCELL, there are three groups of tags that together help in the functionality of the two modules implemented in this study, viz. the Inquire module and the Pallet Controller module. These three groups can be identified as follows,

1. WINCC tags for the Inquire module,
2. WINCC tags for the Pallet Controller module, and
3. WINCC tags shared between the Inquire and the Pallet Controller modules.

The WINCC tags for the Inquire module primarily access the process I/O data and are directly connected to the process I/O lines. In addition to accessing the process I/O data, this group contains tags to keep the pallet count and to indicate the next arriving pallet for each station. The WINCC tags for the Pallet Controller module give information about the order such as part name and also information about the station status. Although these tags belong to the Pallet Controller module, they are also displayed for the operator on the Inquire module interface. The third group of tags is the one that is shared between the Inquire and the Pallet Controller modules. It basically contains information passed by the Pallet Controller but processed and used by the Inquire module.

7.3.1 WINCC tags for the Inquire module

The WINCC tags for the Inquire module can be further divided into following sub-groups,

- Tags that are of memory word type (signed 16-bit) and are used to keep track of the pallet count,
- Tags that are of memory word type (signed 16-bit) and are used to indicate the next arriving pallet,

- Tags that are of binary type and are used as inputs to increment the counter for the pallet count,
- Tags that are of binary type and are used as inputs to raise or lower the latch at the workstations, and
- Tags that are of binary type and are used as outputs to raise or lower the latch at the workstation.

7.3.2 WINCC tags for the Pallet Controller module

The WINCC tags for the Pallet Controller module can be further divided into following sub-groups,

- Tags that are of binary type and are used to indicate whether a particular workstation is free or busy, and
- Tags that are of memory word type (16-bit character set) that indicates the name of the parts for a particular order.

Although these tags belong to the Pallet Controller module, they are also displayed for the operator on the Inquire module interface.

7.3.3 WINCC tags shared between the Inquire and the Pallet Controller modules

The WINCC tags that are shared between the Inquire and the Pallet Controller module are the memory word tags that contain the information about the pallet number requested by a particular workstation. This information is passed from the Pallet Controller module to the Inquire module. The Inquire module uses this information to make the decisions concerning the raising or lowering of a latch at a particular workstation.

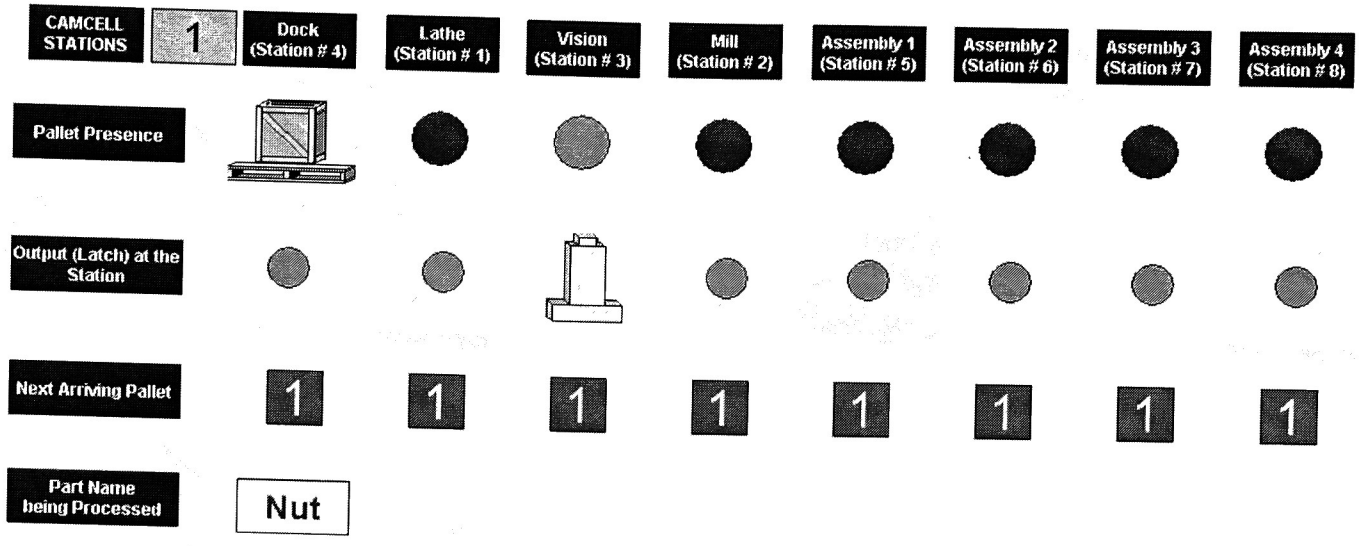
7.4 Inquire

In this study, we implemented the Inquire module using the Siemens' WINCC platform for developing the user interface and Step 7 ladder logic (LAD) for the control process. This module is implemented on the machine that is directly connected to the Siemens PLC and therefore connected to the process inputs and outputs. This machine is configured as the OPC server and is a repository of all process tags belonging to the project. The Inquire module is responsible for the following functions,

1. Monitor the inputs and the outputs from the CTC PLC,
2. Monitor the pallet travel across each of the eight stations, and
3. Raise or lower the latches to stop or release the pallets at the stations.

Figure 7.2 is the user interface screen for the Inquire module, which depicts the control panel with information regarding the following:

1. Presence or absence of a pallet at a station,
2. Status of the latch at the station,
3. Next arriving pallet number at the station, and
4. Arrival of the requested pallet.



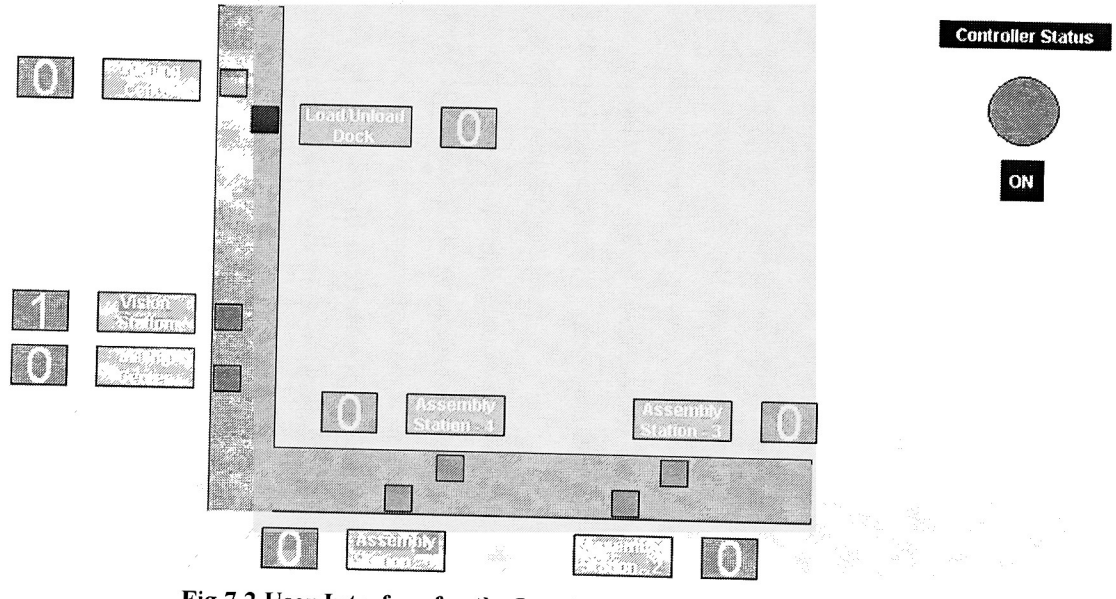


Fig 7.2 User Interface for the Inquire module

It also displays some order details at each of the station. The Inquire module is also responsible to track and process the data provided (via WINCC tags) by the Pallet Controller to note the requests to stop and release the pallets.

7.3.2 Control Program for the Inquire module

As described earlier in this section, the Inquire module has the task of monitoring the movement of the pallets across the eight stations, monitoring the status of the latches at the eight stations and providing the information about the next arriving pallet at each of the eight stations. The Inquire module also has the task of stopping or releasing a pallet at a particular station, when requested by the Pallet Controller. The Control program to implement these functions is built using the Siemens' Step7 Ladder Logic programming language. Appendix F contains the listing of the program for the Inquire module.

There are three main parts in the control program of the Inquire module. The first part of the program keeps track of the pallet movement across the eight stations. This is accomplished with the use of eight counters that increment by one every time a pallet crosses a particular station. Each counter is reset to zero when its count is equal to the number of pallets present in the system. With the help of the pass through logic discussed in the previous section, whenever the sensor at each of the eight workstations goes high (due to presence of a pallet at the station) a corresponding input in the Siemens' PLC goes high. These inputs are used to increment the value of the respective counter.

The second part of the control program processes the pallet request information that is provided by the Pallet Controller module. Here, the pallet number that is requested to be stopped at a station is stored in a memory variable. On the other hand, the value of the counter that keeps the track of the pallets across the stations is incremented by one using an "Add_Int" function of the ladder logic programming. This incremented value is stored in another memory variable. These two memory variables are then compared for equality condition using a "Comparator" function of the ladder logic programming. When the two memory variables have the same value, it indicates that the next arriving pallet at a station is same as the pallet number requested by the station. At this point, a memory bit is set high.

The third part of the control program monitors the condition of the memory bit for each of the eight workstations. When the memory bit for the particular station goes high, the respective latch is raised, thus stopping the requested pallet at the requesting station.

7.5 Pallet Controller

In this study, we implement the Pallet Controller module using the Siemens' WINCC platform and an OPC Client. Figure 7.3 shows the user interface screen for the Pallet Controller module, which contains the information regarding the following,

1. The pallet number requested by the individual station controllers. These Station requests are simulated via a manual data entry into the Pallet Controller interface,
2. The name of the part being processed at the particular station, and
3. The status of the station. (Busy or Free)

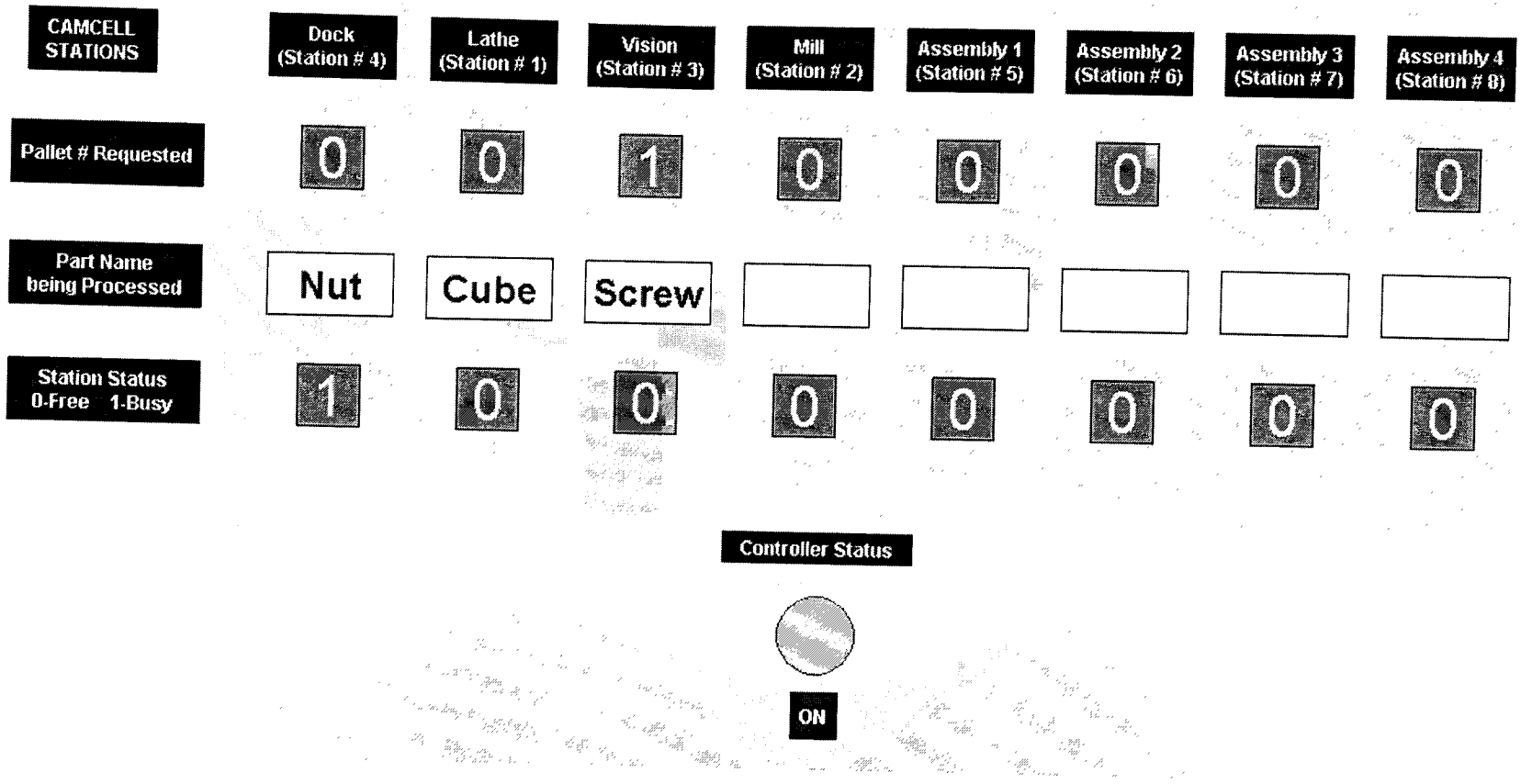


Fig 7.3 User Interface for the Pallet Controller module

The Pallet Controller module passes on the pallet request information to the Inquire module, which on the basis of the information about the next arriving pallet at the station, stops or releases the pallet at that station. However as noted earlier, all WINCC tags are centralized at the OPC server and are available to all OPC clients of this server.

Section 8. Conclusions

The primary focus of this applied research was to study the Siemens' PC based automation technology leading to the understanding and development of a Manufacturing Execution System to control a manufacturing cell. Systems Integration is the progressive linking and combination of various components of a system to merge their functional and technical characteristics into a comprehensive interoperable unit. Integrating these various components provides a value that is greater than the sum of the value provided by the individual components. System Integration is eased by well-established standards in data communications, programming languages, application development environment and computer operating systems.

In this study, we aimed at reviewing various data communication standards, programming languages and the application development environment offered by Siemens viz. the Step 7 and WINCC. Data Communication standards are agreed upon set of rules and guidelines that govern device to device communication as well as communication over the network. In this study, we reviewed the two main modes of data transfer viz. the serial and parallel communication as well as their communication standards. In addition to serial communication standards like the RS-232, RS-422 and RS-485 standard, parallel communication standard like the IEEE-488 standard and network communication standards like TCP/IP and Ethernet, we also discussed the ISO OSI seven-layer model. This model is called the reference model as it encompasses all the functions that are to be fulfilled at each level in communication. In addition to the above-mentioned standards, we also reviewed the Simatic NET Profibus network that is used in the Siemens' PC-based automation technology. Profibus is an acronym for Process Field Bus and is a modified version of the RS-485 serial communication standard. After working with the Profibus network in the CAMCELL application, we studied a number of features of the Profibus network that are portrayed by the Profibus International Association as its advantages. We confirmed such benefits of the Profibus network as open, vendor independent protocol, plug and play feature of the network, high data transmission rates up to 12Mbaud, high data security, and powerful diagnostic capabilities.

We also discussed the various programming languages used to write control logic for process control. There are various programming languages available for writing control logic. The various types of programming languages include graphical programming languages, statement lists, sequential programming languages, and structured Control Languages. Each type of language has some advantages and disadvantages. Graphical programming languages like the Ladder logic and Functional Block Diagrams are the most popular languages in use primarily because of their ease of use and they are reasonably intuitive, especially for technicians with relay logic experience. They are particularly effective in an on-line mode, when the PLC is actually performing control. The operation of the logic is apparent from the highlighting of the various relay contacts and coils on the screen, which identifies the logic state in real time. STL is a text-oriented language in which the control task can be described in the form of a list. It is easy to learn but has poor diagnostic capabilities. Sequential programming languages like Grafcet, Graph are suitable for sequential processes. However they need the transitions and control steps to be written in

either LAD or FBD. Structured Control languages enable simple and fast program development for the user, by the application of powerful language elements such as IF.. THEN.. ELSE. Programs written in SCL have improved comprehensibility and improved structure. However, they are not suitable for simple control applications, as they need the knowledge of basic high level programming language.

The application development environment offered by the Siemens' PC based automation technology basically comprises of Simatic Controllers such as WinLC, Simatic NET Profibus network, Simatic distributed I/O modules, Simatic HMIs such as WINCC and the standard development tool such as the Step 7. Simatic Controllers form the core of the automation system and help in controlling production machines, manufacturing plants or industrial processes. The controller used in the CAMCELL application is called the WinLC, which is the PC based controller in the family of S7 controllers. It provides process control from a computer thus facilitating a PC based control. The Simatic NET links all the Simatic stations together and provides for the networking for the exchange of data and programming at a central point. The Simatic distributed I/O system allows for installing the I/O modules connecting to the machine or the plant in the vicinity of the machine, at a distance from the PLC.

The Step 7 software is the central development tool in the Simatic Automation system. It consists of the basic software for configuring and programming. During our use of the Siemens' PC based automation system, we found their application development environment to be an ideal tool for developing an MES to control the CAMCELL. The two main tools used in the CAMCELL application were the Step 7 and the WINCC tools. Step 7 provides the complete development and configuration environment for developing a control solution. The project manager within the Step 7 known as the Simatic Manager manages all the data belonging to a project. One of the most important features observed about the Siemens' PC based technology was that all the data of a project was stored in a shared database. Therefore any tool within the system can access this data. This prevents data integrity problems between the configuration, programming, system setup, operation and debugging of a project. The hardware configuration using the Step 7 tool was found to be very easy. Step 7 provides integrated configuration of all the hardware across the entire WinAC and the Profibus spectrum.

Another helpful feature of the Step 7 tool was its multi language support. It provides three programming languages, viz. the LAD (Ladder Logic), FBD (Functional Block Diagram), and the STL (Statement Lists). In addition to these three basic languages, additional programming languages can be integrated into the Step 7. Such languages include Graph, HiGraph, and Structured Control language. The Step 7 was also found to have excellent diagnostics and interrupt capabilities. During the program execution, the logic can be viewed in the online mode, thus facilitating in diagnosis of any faulty logic or faulty hardware. Another important feature of Step 7 observed during this study was its support of different network and bus systems. It supports the Multi Point Interface (MPI) connection, the Profibus as well as Industrial Ethernet.

WINCC is a Supervisory Control and Data Acquisition Software HMI (Human Machine Interface). WINCC is a tool for process visualization and information availability. Plant Floor Integration is realized with WINCC through its open connectivity architecture. During our experience with WINCC in the

CAMCELL application, we found it to be very robust monitoring and data acquisition tool. WINCC can communicate with hardware devices such as PLCs, drives and sensors and exchange information with various software devices like soft controllers and other HMIs. Tags once defined in the Step 7 environment are available in WINCC for use. These tags can be read as well as written on. Thus the data integrity is maintained within the WINCC also. In addition to its suite of internal drivers, WINCC can communicate with a variety of automation equipment using the OPC (OLE for Process Control), which is an industry standard for Industrial Communications. Using the OPC technology, remote control of an application was accomplished in the CAMCELL application. Remote monitoring and data entry from a remote station was accomplished by the OPC connectivity offered by WINCC. WINCC was also found to be a very powerful tool in its capability of designing graphics to represent a schematic of a process. One can choose objects from a large objects library to develop an intuitive graphics of a process. Another powerful feature of WINCC is its Client Server architecture. Under this architecture, a central WINCC server is the only machine connected to the process and is a repository of all process related data. This data can then be accessed by a number of clients via the TCP/IP. Data management becomes easy when it is resident in one location. Thus WINCC client server architecture also contributes towards data integrity and ease of data management. In the CAMCELL application, the same task was accomplished by implementation of OPC client server architecture. Due to time constraints, WINCC client server architecture could not be implemented in the CAMCELL application, something that can be implemented in future applied research projects.

As a part of Siemens' Total Integrated Automation, the Siemens' PC based automation technology integrates systems that are best suited for a task. It integrates PLCs, distributed I/O modules, the HMIs, and the development tools into a common kernel. This kernel supports threefold integration viz. integrated configuration and programming, integrated data management and integrated data communication. Overall the technology was found to be an ideal application development environment for building an MES for controlling a manufacturing system and was true to its claim of having a totally "open" architecture.

Appendix A. SOFTWARE INSTALLATION

NOTE: Only members of SIMATIC group of ASI_ISE domain and local administrators can install this software.

The SIMATIC S7 PLC consists of software and hardware contained in following 8 boxes.

- PROFIBUS CP 5613
- WINAC BASIS
- WINCC / RC 256 V5.0

These are the three major software and hardware packages. In addition to this we also install a standard tool called

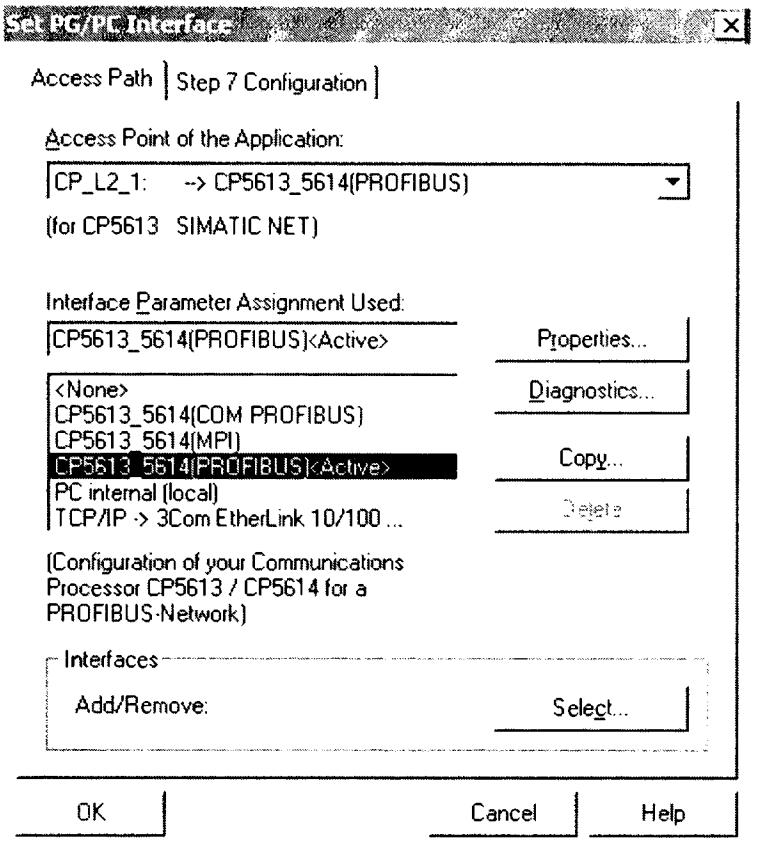
- STEP 7 V5.1

Then we also install some engineering tools such as

- S7 - SLC V5.1
- S7 - PLCSIM V5.0
- S7 - Graph V5.0
- S7 - Hi-Graph V5.0

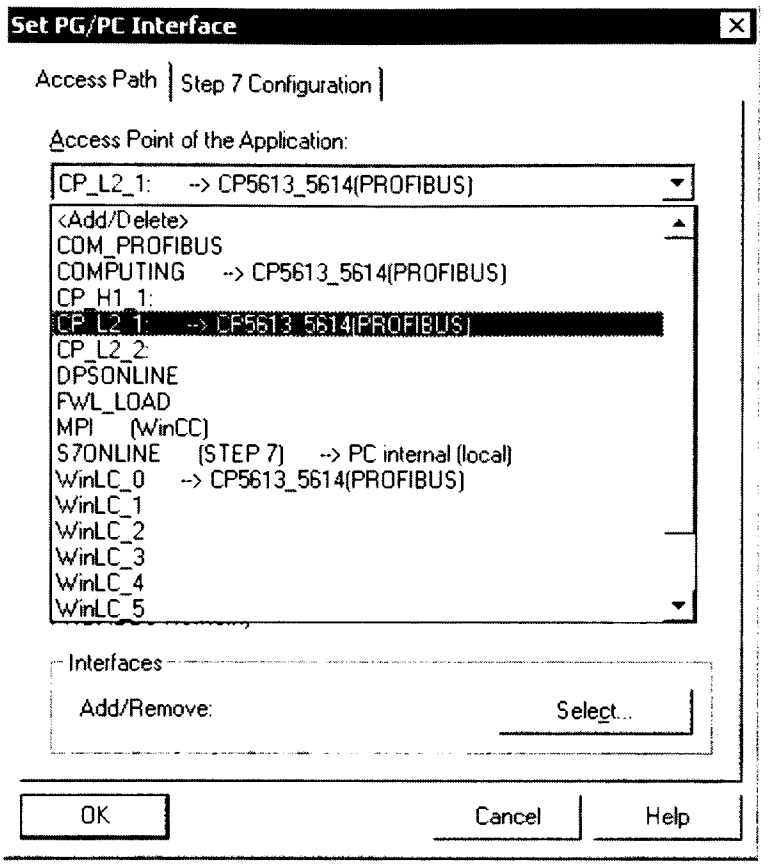
1. The first step is the installation of the CP5613 PCI card into the PCI slot of the computer. This card is found in the PROFIBUS CP 5613 box.
2. Once the card is installed and the computer is started a wizard for new hardware found pops up. Cancel this wizard at this stage.
3. Once the CP5613 PCI card is installed, we install the WINAC basis software.
4. For this, explore the WINAC CD and double click on the SETUP icon. There are different CDs for the NT machine and Win-2000 machine. Use the appropriate CD. Go through various windows and at the time of selection of language select English as the language. At the time of selection of target drive, select target drive as the C drive. (or D drive depending on the space remaining on the drive) RUN AUTOMATIC AUTHORIZATION. At this time you will have to insert the authorization diskette (that is provided with the software package) in the A drive and then continue with setup. The WINAC will now be installed.
5. During the installation of WINAC, select WINLC as an application
6. In the configuration of the Step S7 link Step S7 to PC Internal (Local)
7. Once the WINAC software is installed, reboot the computer. During the restart, remove the authorization diskette from the A drive. However keep the CD inside the CD-ROM drive.
8. When the computer reboots, the "New Hardware Found" wizard pops up again. This time click next in the windows that come up and finish the hardware installation.
9. The next package to be installed is SIMATIC NET. The CD for this is found in the PROFIBUS CP 5613 box.

1. Make sure that the WINLC is shutdown while performing the installation check.
2. Start the "PG/PC interface" tool (taskbar Start – Simatic – Simatic NET – Set PG/PC interface)
3. Select the appropriate access points of application as seen below:

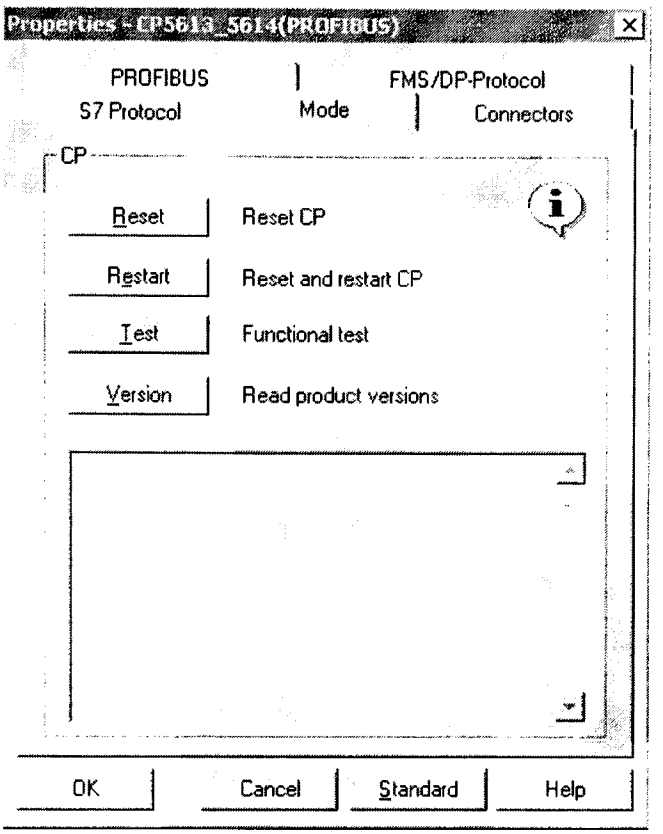


4. At this stage, set the access point of application for the following four
    a. Computing → CP5613_5614(PROFIBUS)
    b. CP_L2_1 → CP5613_5614(PROFIBUS)
    c. S7ONLINE (Step7) → PC internal (LOCAL)
    d. WINLC_0 → CP5613_5614(PROFIBUS)

    Te Interface parameter Assignment used for S7ONLINE (Step7) is the same as the one selected during the installation of the WINAC software. The access point of application should look as follows:

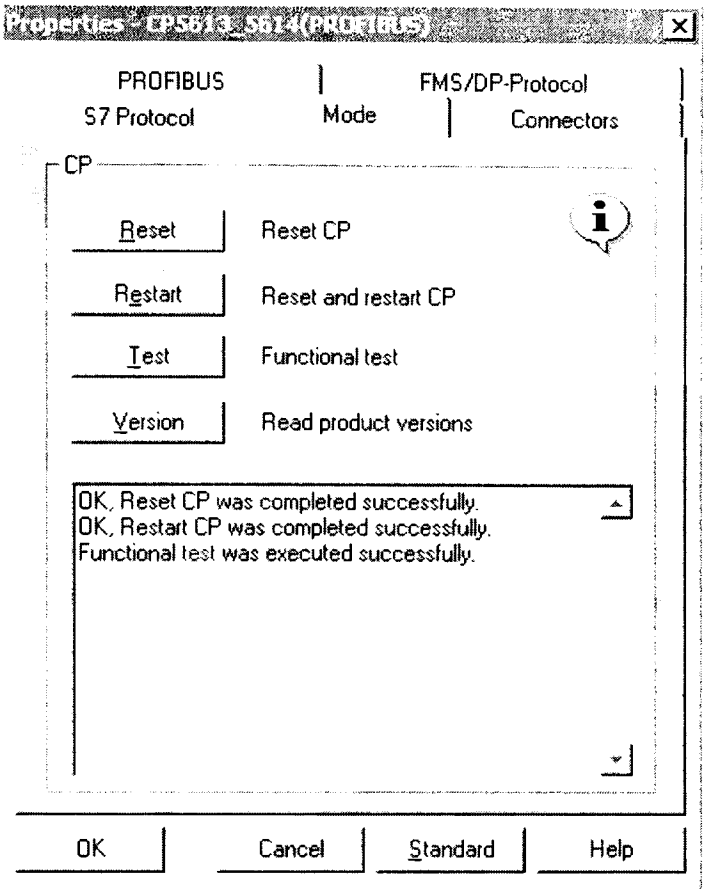


5. Click OK to save the setting.
6. Next click on the "Properties" button. A properties dialog appears with several tabs.
7. Select the Mode tab after which following window appears

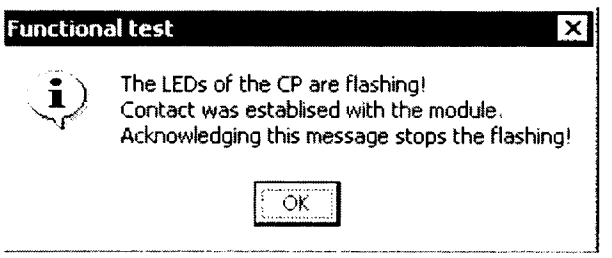


8. Click the Reset button followed by the Restart button.
9. Next, click the "Test" button, with which we can check whether the CP can be accessed and whether the installation was proper. If the installation is proper and if the CP can be accessed, then the LEDs of the module at the back of the CPU start flashing.

Following screenshot covers steps 8 & 9.



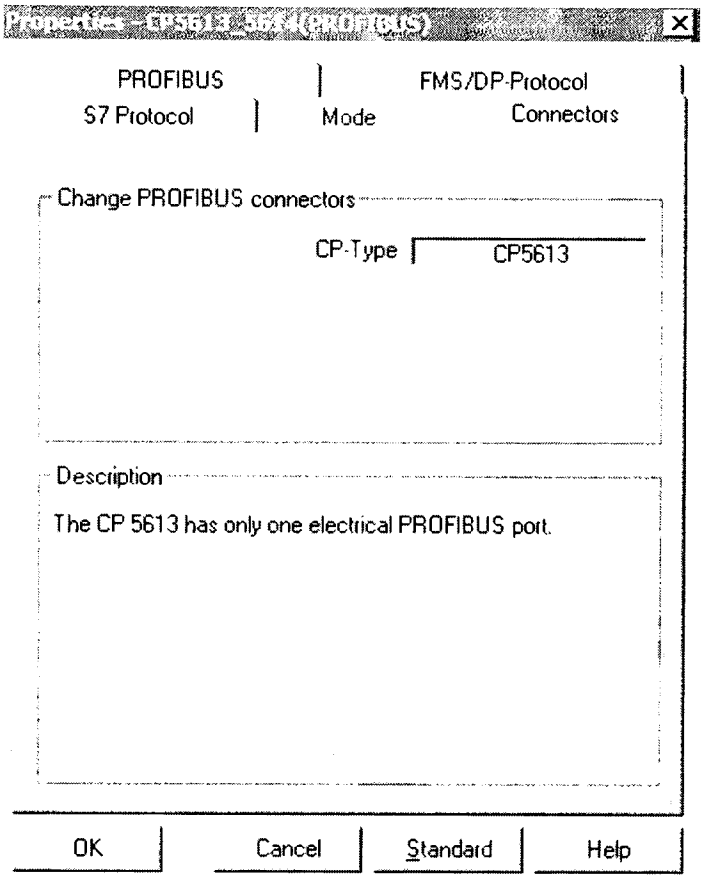
When the Test button is clicked, following message appears.



After this we can check the version of the software and the hardware by clicking the "Version" button.

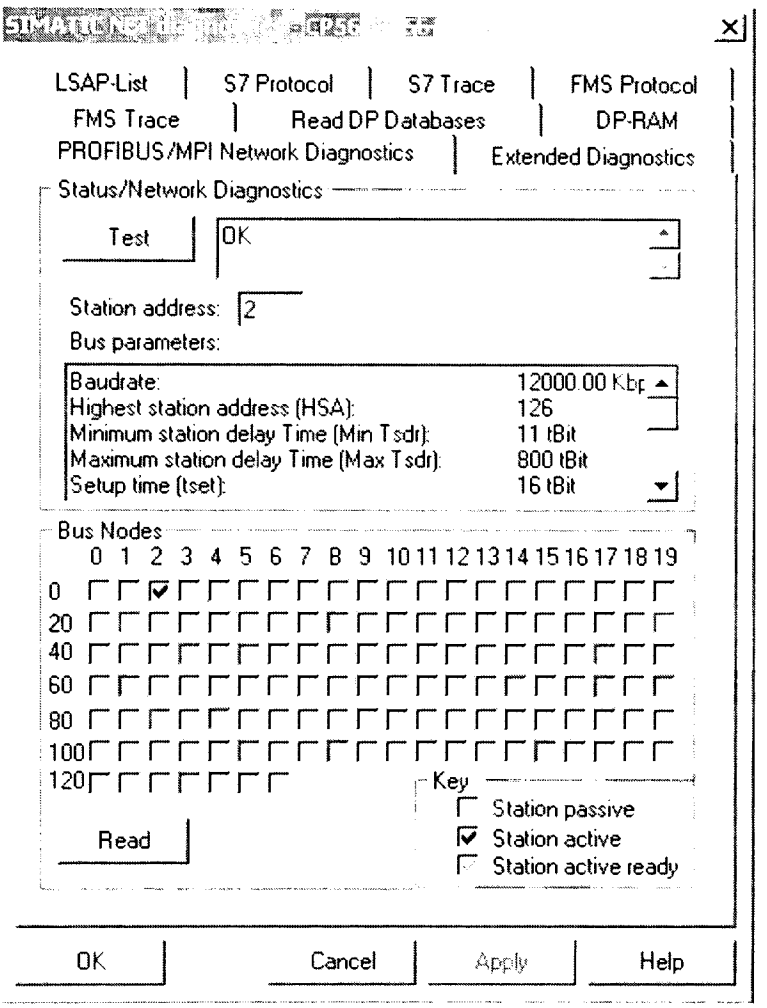
Setting the BUS parameters

1. Once the access points of application are set and once the installation is checked, we set the BUS parameters.
2. For this click the "Properties" button.
3. Since the Interface Parameter Assignment used is the CP 5613_5614, we select the "Profibus" tab.
4. Set the parameters as seen below



Checking Network Access

1. Checking the network access provides an overview of the following:
    a. The attached bus nodes
    b. The current bus parameters
2. Click the "Diagnostics" button in the "Set PG/PC interface" tool.
3. Select the "Profibus/MPI Network Diagnostics" tab.
4. Click the "Test" button at which the Mode is displayed and detailed information about the current bus parameters are obtained.
5. Click the "Read" button at which the mode is displayed and an overview of the attached passive and active nodes is obtained.



With this the Setting of the PG/PC interface is complete.

Appendix B. Hardware Configuration

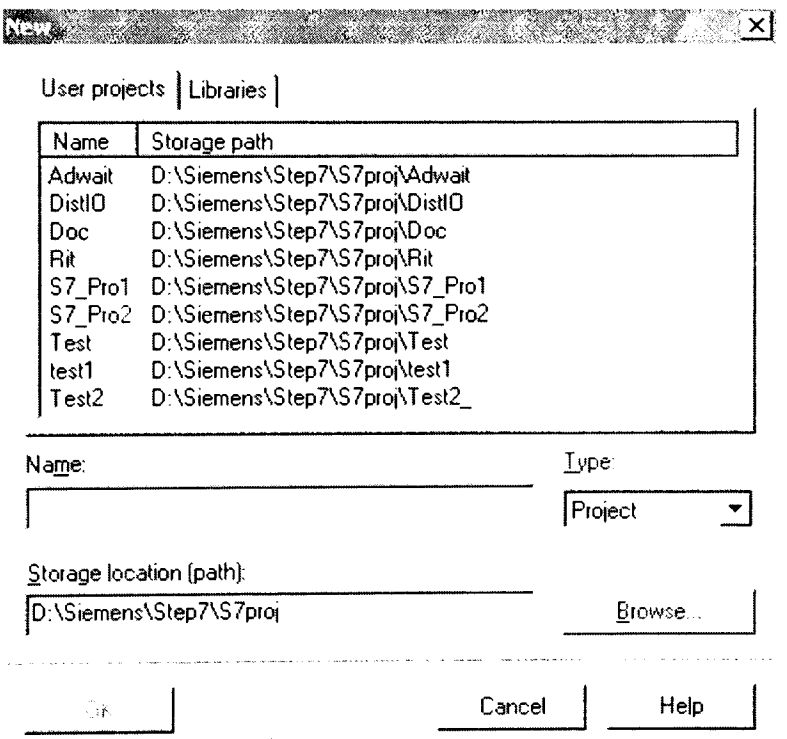
Introduction

The term configuring refers to arranging of racks, modules, and interface modules in a STATION window. Racks are represented by a configuration table that permits a specific number of modules to be inserted, just like a real rack.

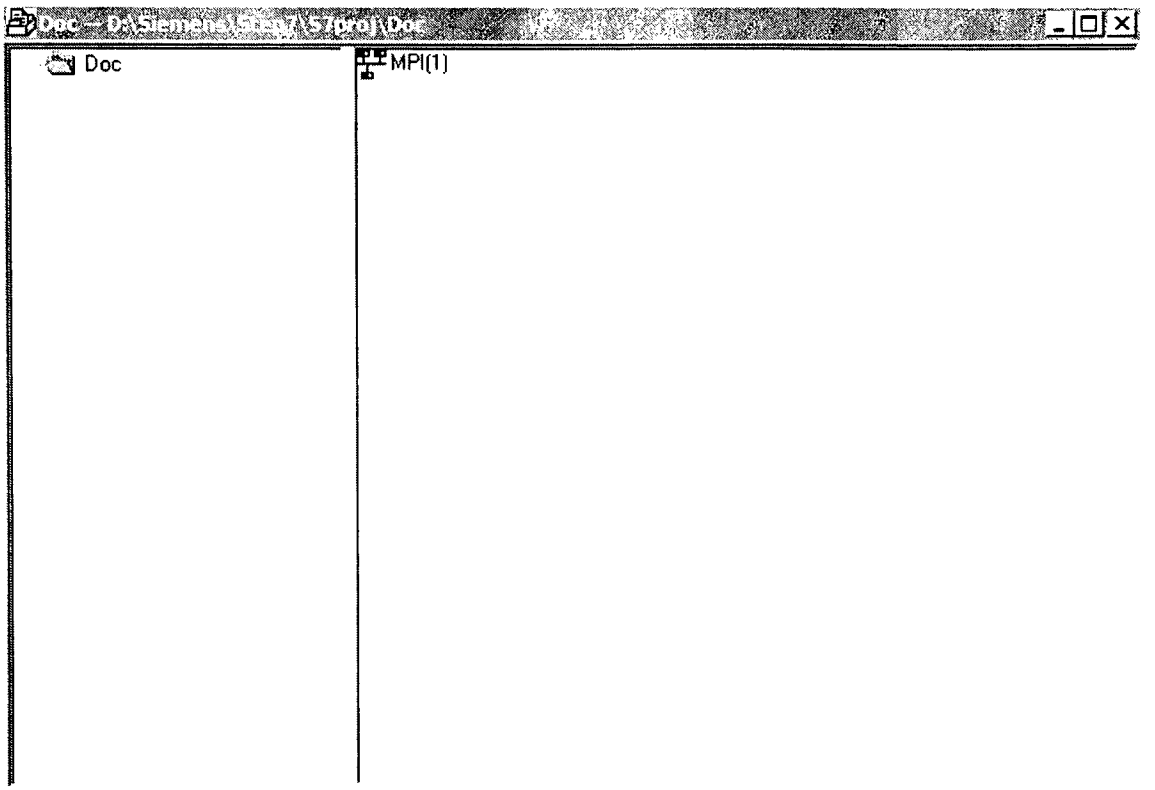
Steps in Configuring

Following steps are followed in the configuration of the hardware

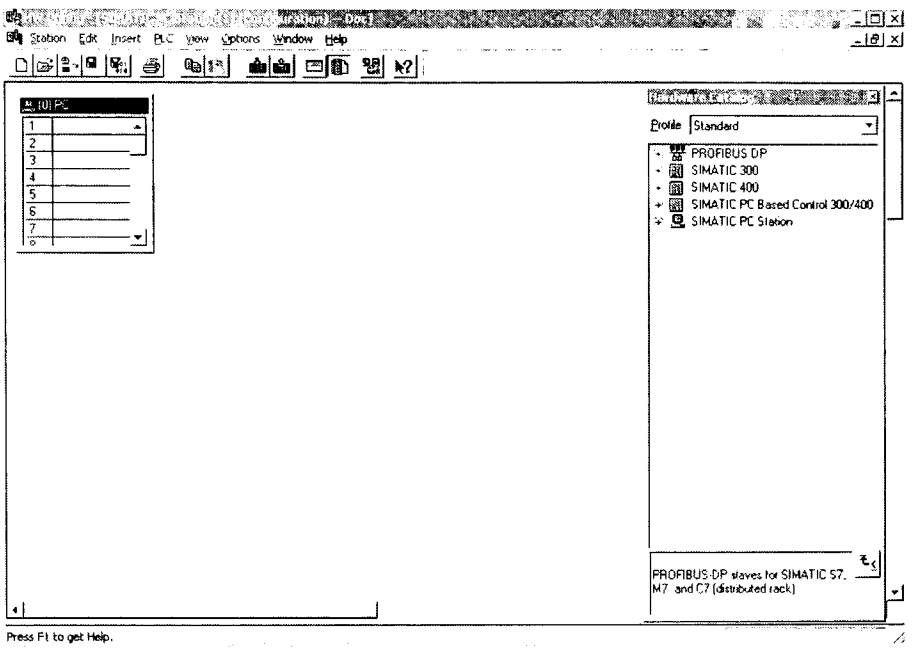
1. Open the SIMATIC MANAGER (Start → Simatic → Simatic Manager)
2. Open a new project (File → New). This will pop up the following window



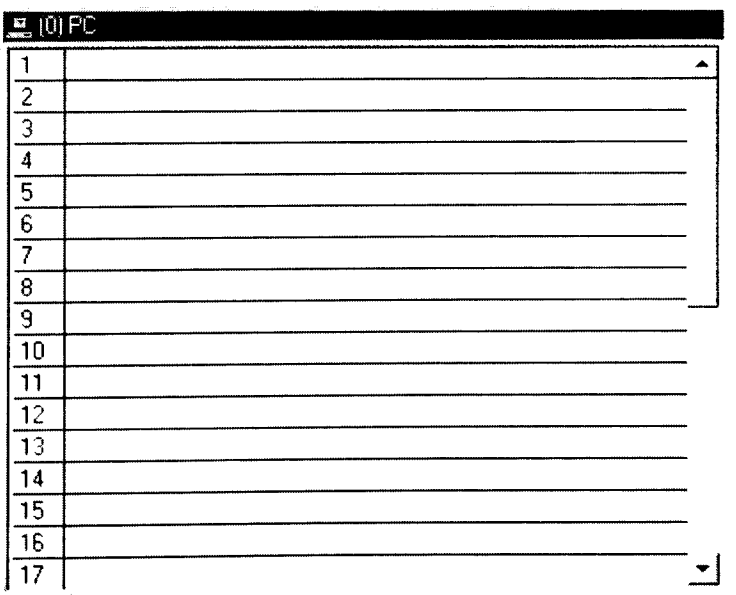
3. Enter the desired name of the project in the "Name" field. Select the type as "Project" and also select the desired storage location path. Click OK to open this new project. The project will look like this



4. Insert a new Simatic PC station (Insert → Station → Simatic PC Station)
5. Double click on the Simatic PC Station and this opens a window showing the icon for Configuration.
6. Double click on the Configuration icon to open the window for Hardware Configuration called the HWConfig as follows

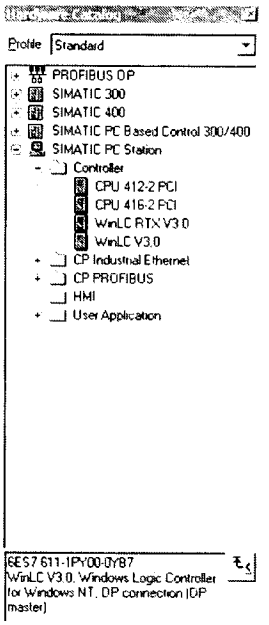


7. There are two windows in the HWConfig window
   a. The Station window
   b. The "Hardware Catalog" window from which we select the required hardware components. (If the hardware catalog window is not displayed then select the menu command View → Catalog)



Station Window

8. From the hardware catalog, select the WINLC V3.0 controller from the Simatic PC Station Option as shown below



Hardware Catalog

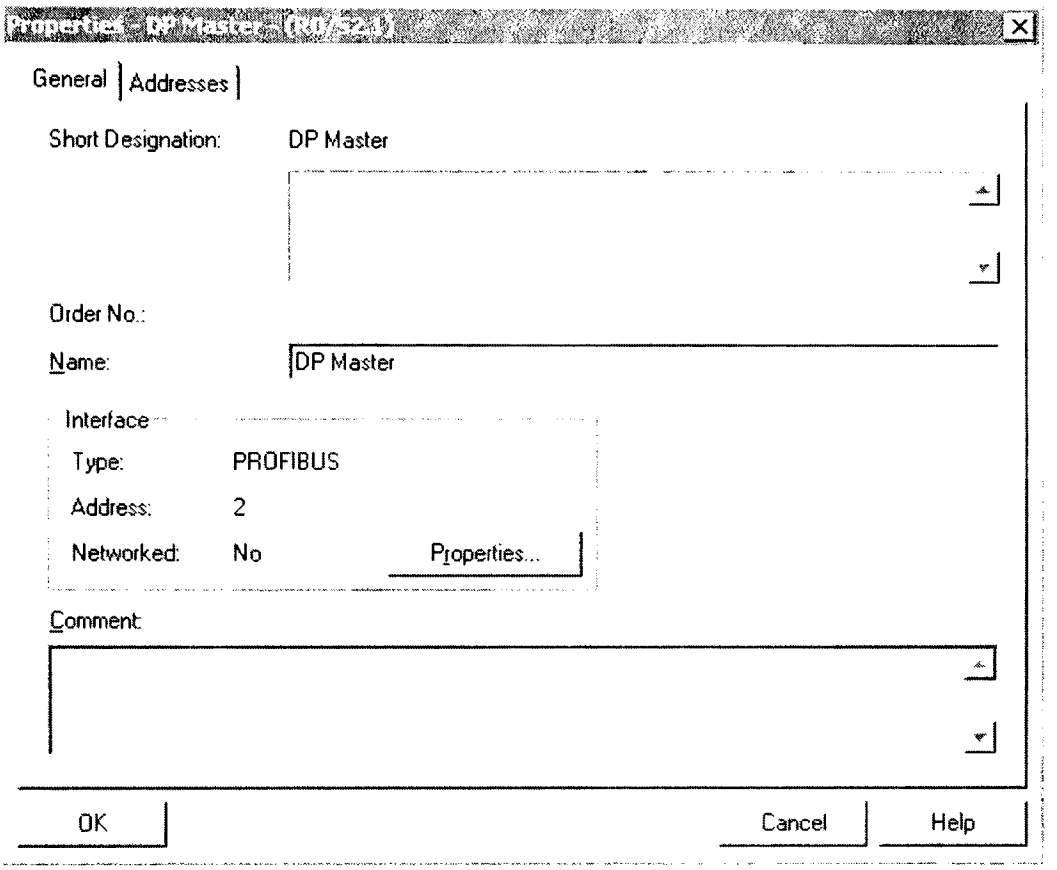
9. Click on the WINLC V3.0 icon and with the mouse click held down, drag it to the station window and release it.
10. This will insert the WINLC V3.0 controller in the station window.
11. At this stage, a window for the properties of Profibus interface DP master pops up. Ignore this window and click OK to continue.

Note:

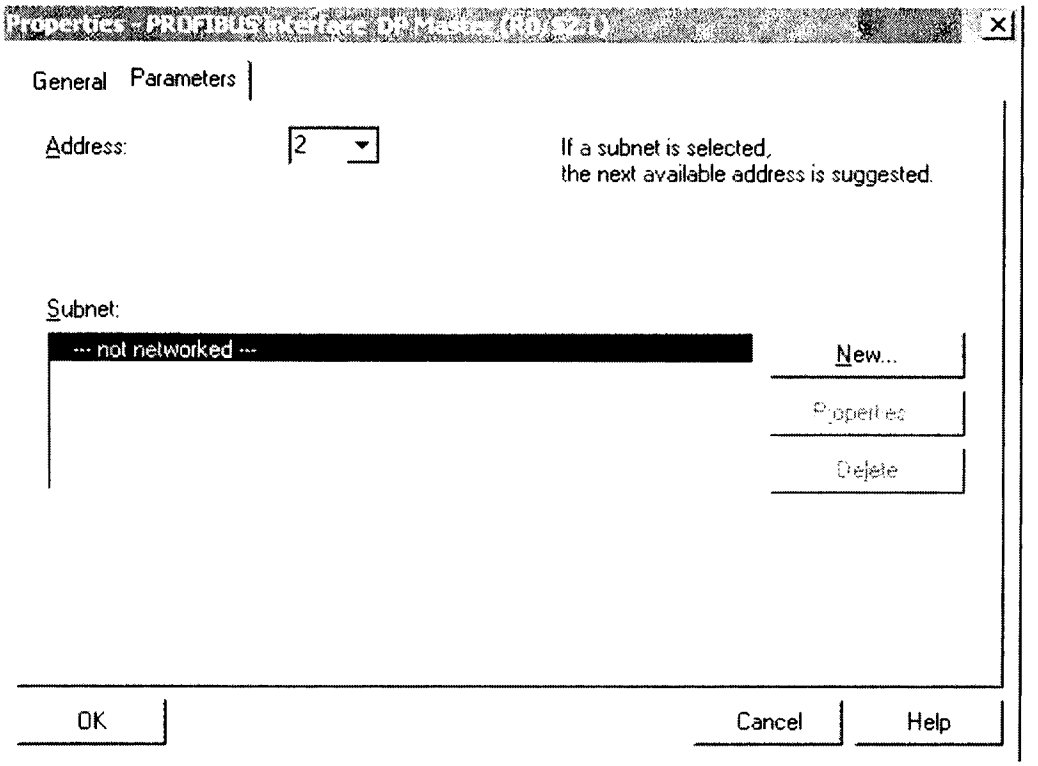
We cannot use the MPI node of WINLC V3.0 SP1 to configure the hardware. WINLC has no effect on any installed MPI card. The MPI address of the WINLC should be left at node address 2.

---

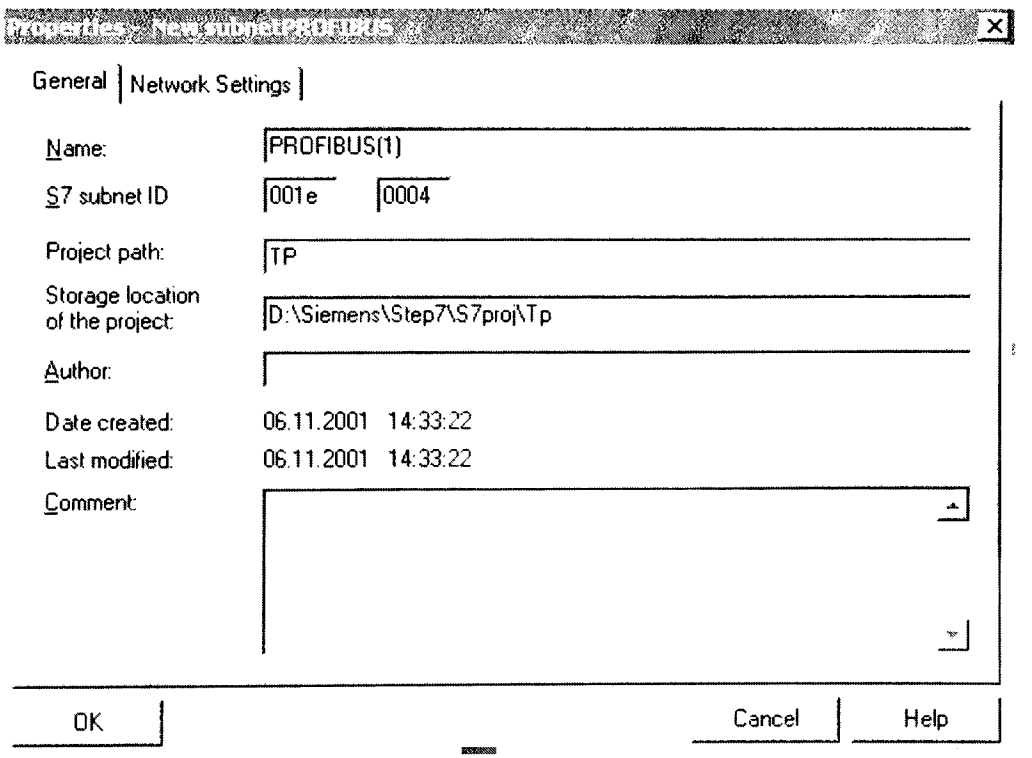
12. Next select (click on) the DP Master icon in the Station window to set its properties and double click on it to get the following window



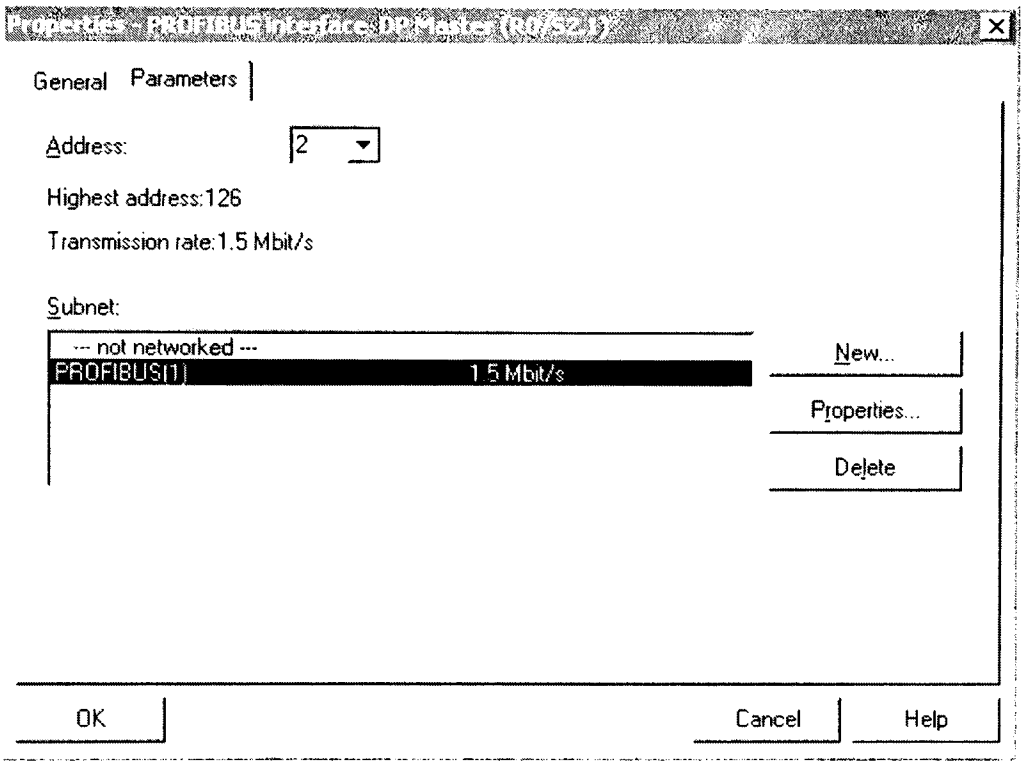
13. Click on the properties to get the window for Profibus interface DP Master that looks like this



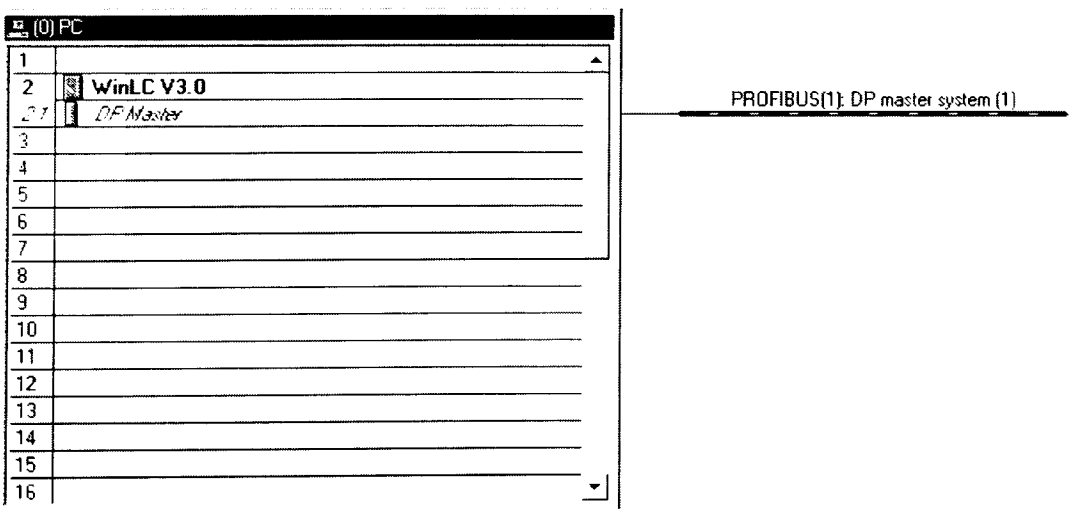
14. Click on the "New" button to get the following window



15. Click OK to accept the default of Profibus (1)
16. Next, click OK to enter default parameters for a PROFIBUS subnet and close the "Properties-PROFIBUS Node DP Master" dialog box. At this stage WINLC appears as the module in slot 2 of the rack.
17. At this stage if we again double click on the DP Master icon and click on properties we see the following window that shows the properties

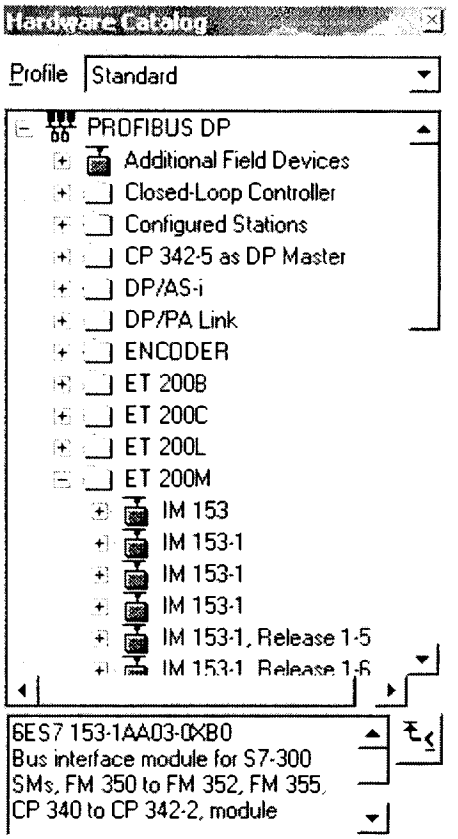


18. At the same time the station window looks as shown below

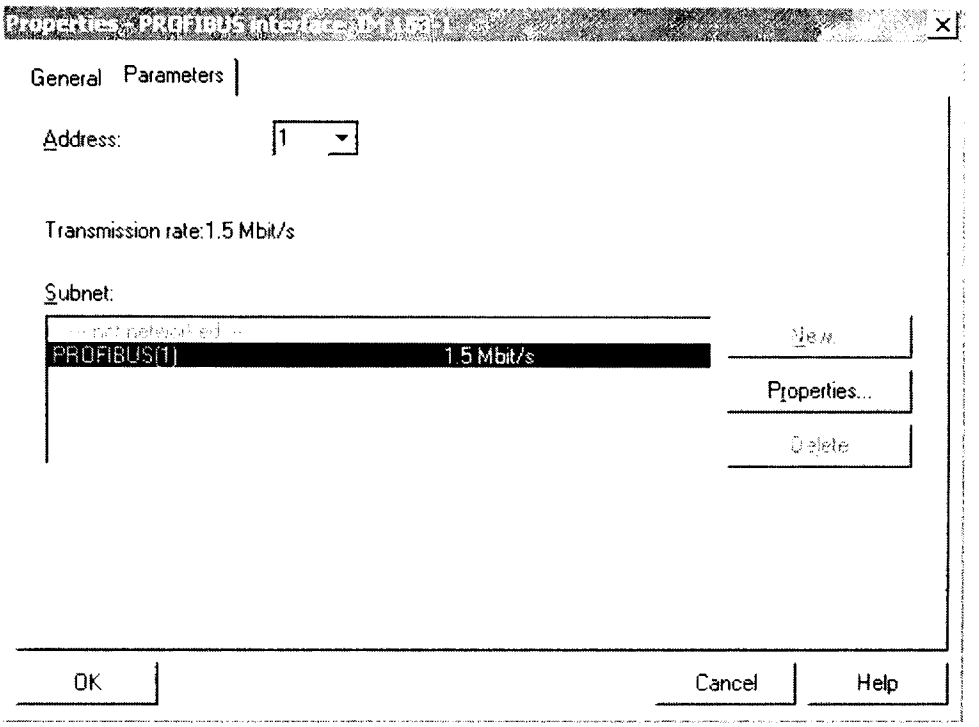


19. Select the Station → Save and Compile command.
20. Next we insert the interface module into the configuration.

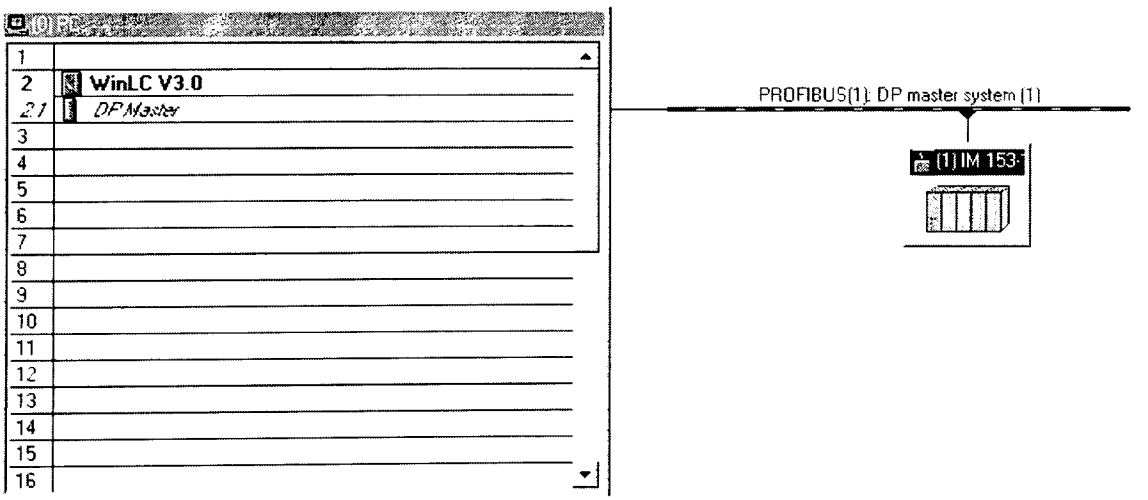
21. Select the Profibus DP from the Hardware Catalog and then select ET200M (the interface module ID that we have on the PLC unit). In this, select the IM 153-1 interface module that corresponds to the serial number 6ES7-153-1AA03-0XB0.



22. Drag this module to the Profibus (1): DP master system (1) and release it. At this stage following window pops up

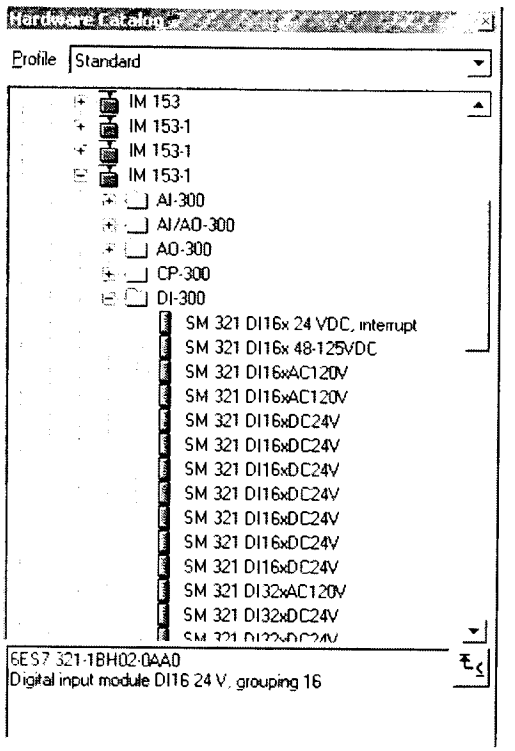


23. In this dialog box, select the Profibus address as 3 (in agreement with the station address set during the SET PG/PC interface done by adjusting the dip switches)
24. Click OK. We now see the station window as follows



25. Next step is to configure the I/O modules.

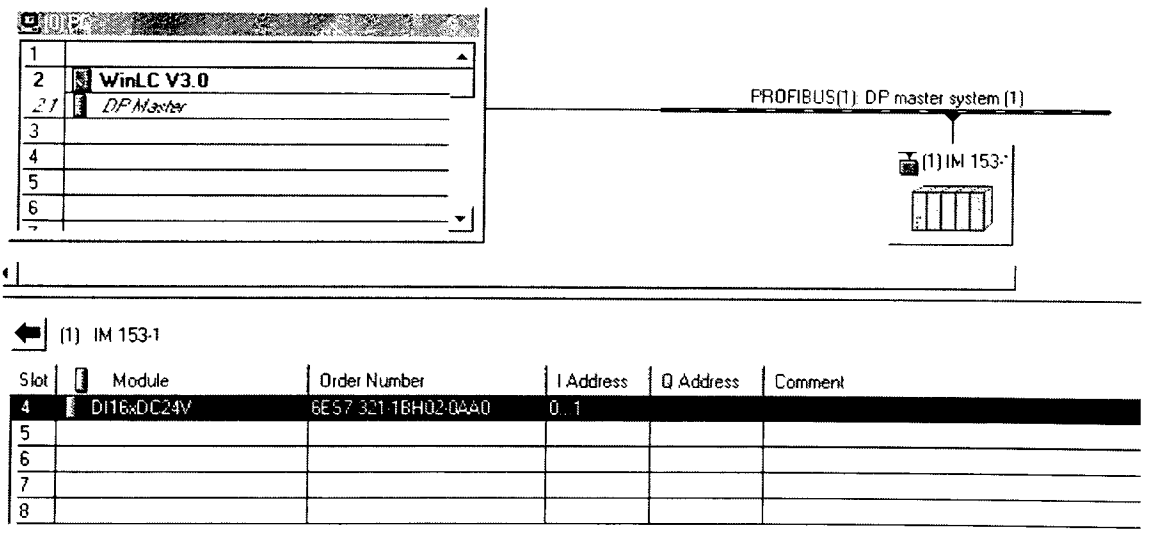
26. First select a DI/O simulator module. We can select any D16 x D24V module available in the DI-300 family. But to maintain uniformity we select a D16 x D24V module with order no.6ES7 321-



1BH02-0AA0.

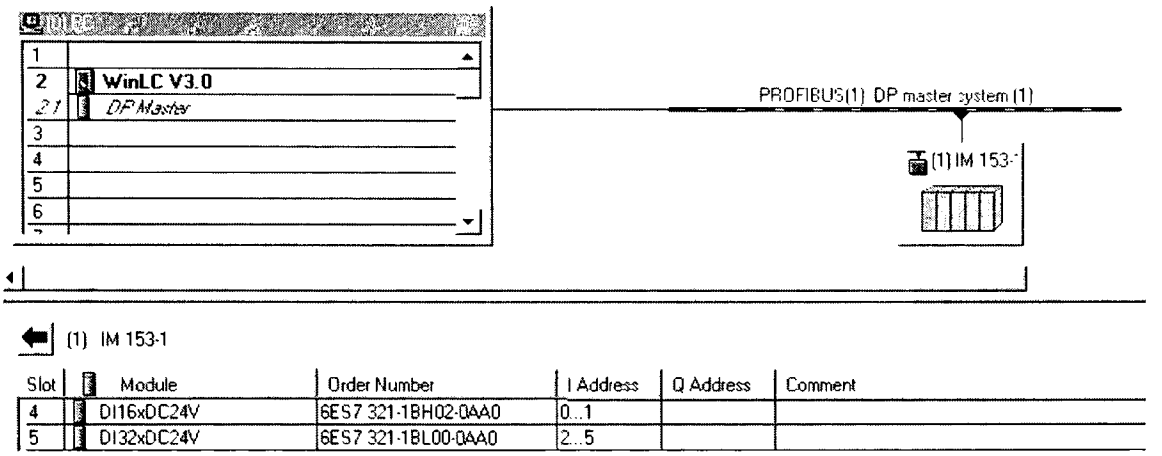
27. With the interface module IM 153-1 selected, just double click on the simulator module type and it will automatically enter into the slot 4 of the DP Slave system.

The Station window now looks as follows



28. Next, we also select the input and the two output modules.
29. Select the slot # 5 in the DP slave table and then select the input module (ET200M→IM153-1 →DI-300) corresponding to the following description

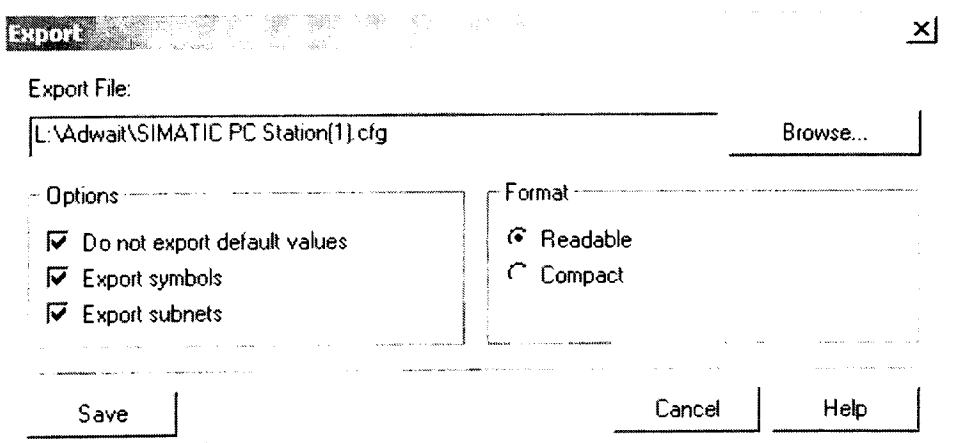
DI 32 x DC24V with order number 6ES7-321-1BL00-0AA0. Double click on this module and the input module automatically enters into the slot # 5 of the DP Slave table. The Station window then looks like this



30. We follow similar steps for the two output modules. We select slots 6 and 7 for them and select the modules (ET200m →IM 153-1 →DO-300) with following description

DO 8 x REL AC 230V with serial number 6ES7-322-1HF10-0AA0.

31. Save and Compile the configuration.
32. Once this configuration is saved, export the configuration into a .cfg file
33. For this, click on the Station > Export option in the Station menu, after which the following window pops up



34. Browse to the location where you wish to save this .cfg file.

Appendix C. Communicating Step-7 with WINCC

Purpose

The purpose of this tutorial-type document is to give step-by-step instructions to start, build, and manage a Step 7 project. With this document, one can create a Step 7 project, add an operator station, define symbols, write the control logic program in either LAD, FBD or STL programming language, download the program to the WinLC, define the symbols as WINCC tags and transfer them to WINCC, create graphics in the Graphics Designer editor of WINCC and activate the runtime module.

Pre-Requisite

In order to create, build and manage a project using this document, one should have done the complete hardware configuration using the document contained in Appendix B.

Procedure

The first step in working with Step 7 and WINCC is creating a project in Simatic Manager. The SIMATIC Manager is the basic application for configuring the hardware and programming logic in a control application. The SIMATIC Manager is the central window, which becomes active when STEP 7 is started. It manages all the data that belongs to an automation project.

1.1 Open the Simatic Manager from the Start button, Start > Simatic > **Simatic Manager**. Alternatively an icon for the Simatic Manager may be found on the desktop of the computer.

1.2 Once the Simatic Manager is opened, create a new project from the File menu of the Simatic Manager, **File > New**. Alternatively, the new project can be created from the New Project/Library icon 

1.3 Once the New option from the File menu is selected, or the New Project/Library icon is clicked, the following window appears

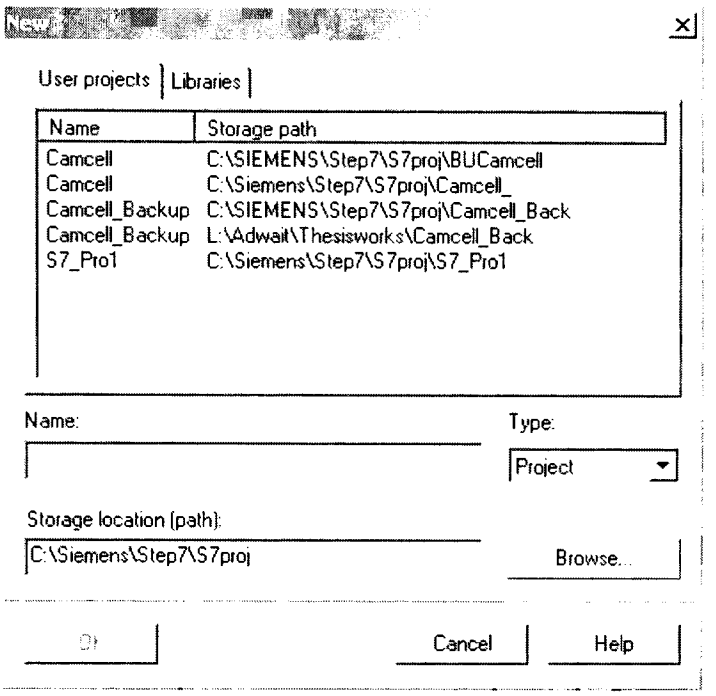


Fig C.1

1.4 In this window, type the name of the project in the Name field and select the Type as Project. (The other option in the Type field is the "Library") Select the default storage location for the project, i.e. C:\Siemens\Step7\S7proj.

1.5 Click OK to create the project.

Once the project is created, the next step is to add a Simatic PC station to this project and do the hardware configuration.

3.1 With the project name selected, click the Insert > Station > Simatic PC station menu.

2.2 This adds a Simatic PC station to the project, after which the project window looks as shown below

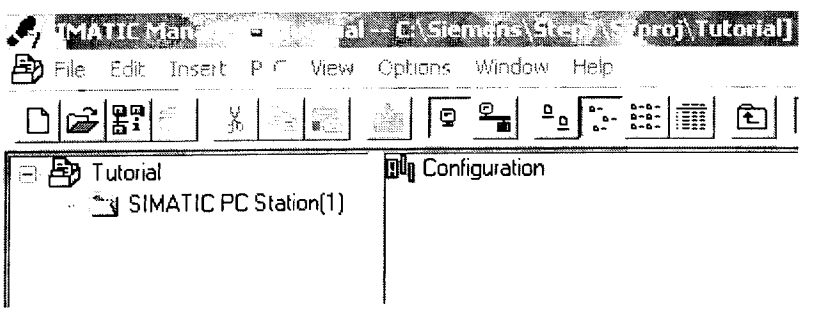


Fig C.2

2.3 Now, with the Simatic PC Station (1) selected, double click on the Configuration icon as seen in the figure above.

2.4 At this point, the HWConfig window opens. Here, import the configuration file that was created during the hardware configuration. For this, select the Station > Import option from the Station menu, after which the following selection window pops up

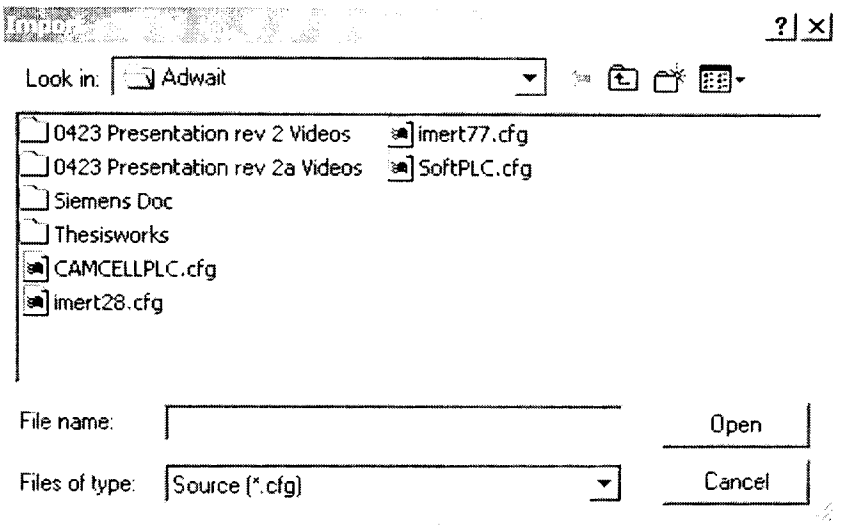


Fig C.3

2.5 Browse to the location where the configuration file was saved at the end of the hardware configuration step and select the .cfg file and click on Open to get the following window

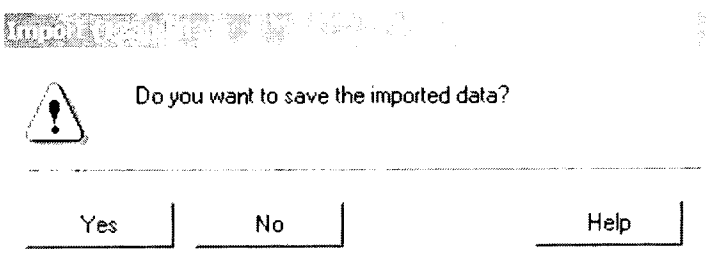


Fig C.4

2.6 Click on Yes to save the imported data. This saves the configuration data on the project.

2.7 Download this configuration onto the WinLC by clicking on the download icon

2.8 Close the HWConfig window.

The next step is to add the Operator Station to the project. An Operator Station is an object that includes within a S7 project, a SIMATIC component such as WINCC, as an application for automation tasks. These objects are configured within the SIMATIC PC Station. An operator station facilitates for the transfer of the symbols used in the S7 program, as tags that can be used in the WINCC. Thus it acts as a link between WINCC and the Step 7.

3.1 Right click on the name of the project in the project window. (In this case, the name of the project is Tutorial, refer Fig C.2)

3.2 Select the Insert Station option from the pop menu that appears upon right clicking the project name.

3.3 From the various options that pop up from this selection click on the OS option.

3.4 This adds an Operator Station to the project and the project window will now look as seen in the figure below

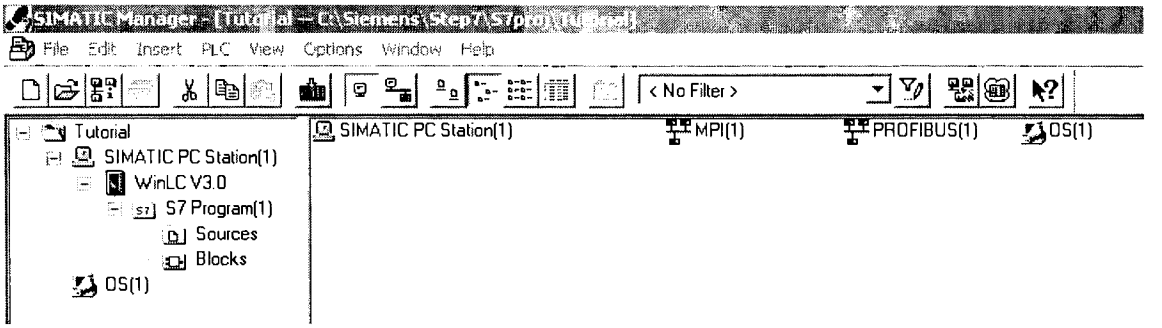


Fig C.5

3.5 Here, OS (1) is the operator station. One can rename the Operator Station to a more logical name. For this right click on the OS (1) and select the rename option from the menu that pops up.

Once the operator station is added, the next step is to define the symbols in the Symbols table

4.1 Click on the S7 Program (1) object to get the following window

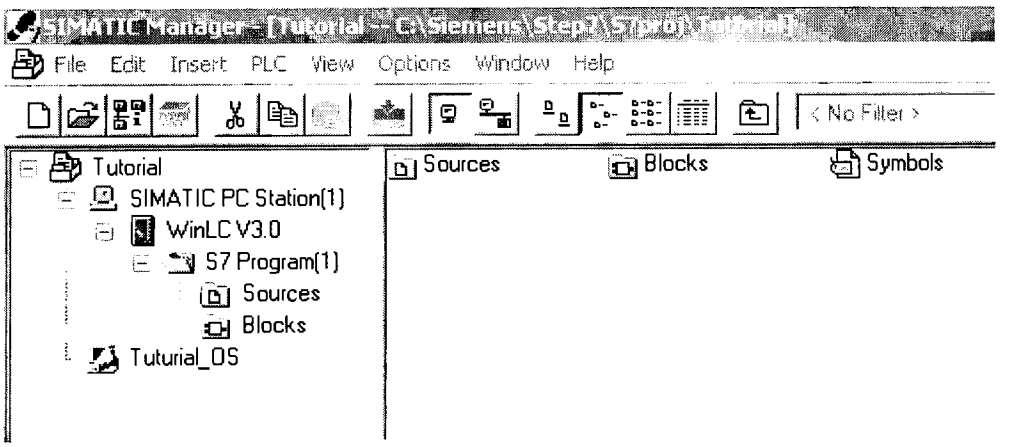


Fig C.6

4.2 Next, double click on the Symbols object to get the Symbols Editor that looks as seen below

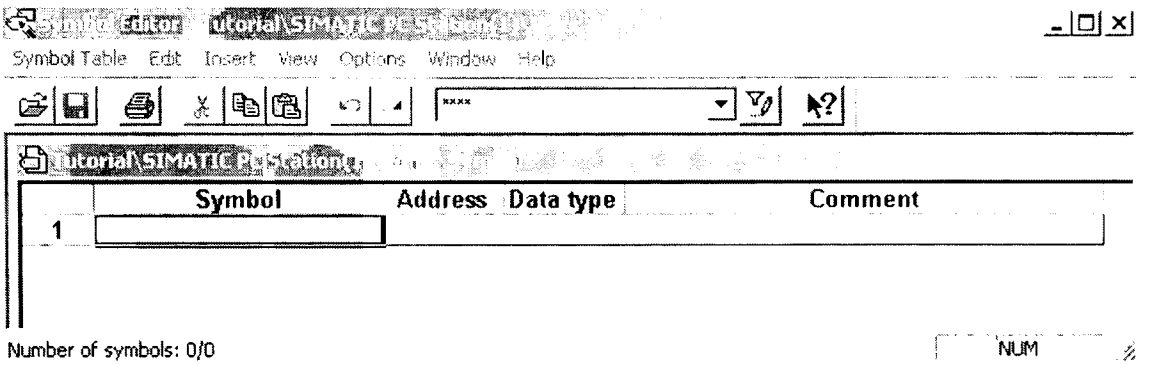


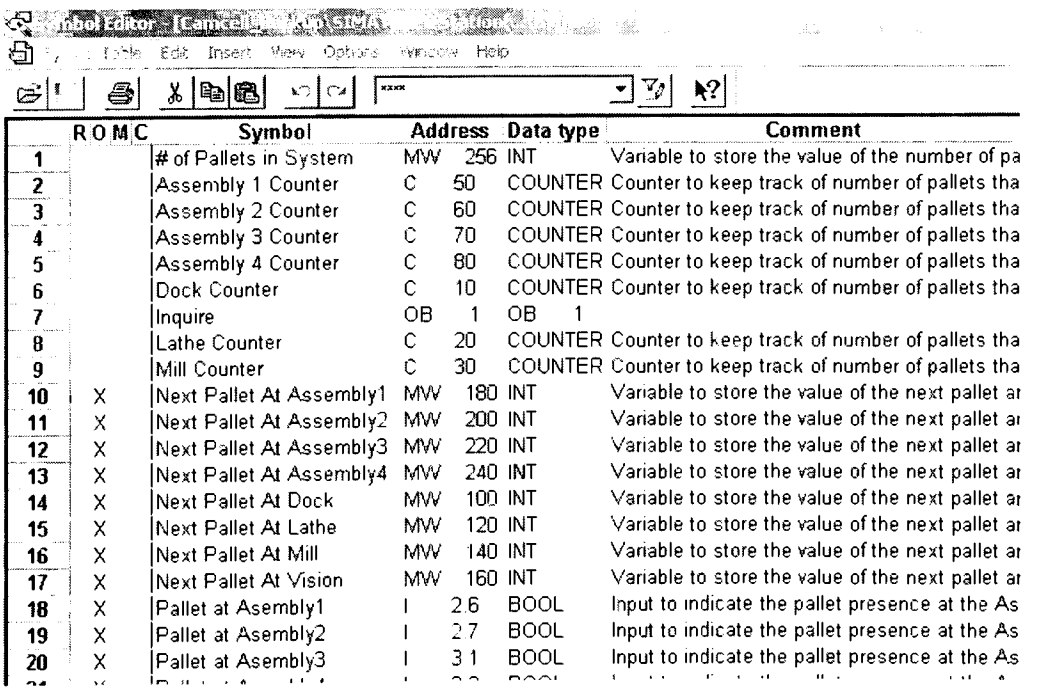
Fig C.7

4.3 Here one can define all the symbols that are required in the control application. For more details and help for using the Symbols Editor for symbols definition refer the Siemens' Online Help. (Go to Start > Simatic > Documentation > English. Refer the manuals Programming with Step 7 and Working with Step 7)

4.4 Once all the symbols are defined, save the symbols.

4.5 In order to transfer the Step 7 symbols as WINCC tags, these symbols need to be named into one of the four categories O (Operator Control and Monitoring), C (Communication), M (Messaging), R (Monitoring).

4.6 In the Symbols Editor, click the Columns O, C, M, R option from the View menu. At this point, the Symbols Editor looks as seen below



	R	O	M	C	Symbol	Address		Data type	Comment
1					# of Pallets in System	MW	256	INT	Variable to store the value of the number of pa
2					Assembly 1 Counter	C	50	COUNTER	Counter to keep track of number of pallets tha
3					Assembly 2 Counter	C	60	COUNTER	Counter to keep track of number of pallets tha
4					Assembly 3 Counter	C	70	COUNTER	Counter to keep track of number of pallets tha
5					Assembly 4 Counter	C	80	COUNTER	Counter to keep track of number of pallets tha
6					Dock Counter	C	10	COUNTER	Counter to keep track of number of pallets tha
7					Inquire	OB	1	OB 1	
8					Lathe Counter	C	20	COUNTER	Counter to keep track of number of pallets tha
9					Mill Counter	C	30	COUNTER	Counter to keep track of number of pallets tha
10		X			Next Pallet At Assembly1	MW	180	INT	Variable to store the value of the next pallet ar
11		X			Next Pallet At Assembly2	MW	200	INT	Variable to store the value of the next pallet ar
12		X			Next Pallet At Assembly3	MW	220	INT	Variable to store the value of the next pallet ar
13		X			Next Pallet At Assembly4	MW	240	INT	Variable to store the value of the next pallet ar
14		X			Next Pallet At Dock	MW	100	INT	Variable to store the value of the next pallet ar
15		X			Next Pallet At Lathe	MW	120	INT	Variable to store the value of the next pallet ar
16		X			Next Pallet At Mill	MW	140	INT	Variable to store the value of the next pallet ar
17		X			Next Pallet At Vision	MW	160	INT	Variable to store the value of the next pallet ar
18		X			Pallet at Asembly1	I	2.6	BOOL	Input to indicate the pallet presence at the As
19		X			Pallet at Asembly2	I	2.7	BOOL	Input to indicate the pallet presence at the As
20		X			Pallet at Asembly3	I	3.1	BOOL	Input to indicate the pallet presence at the As

Fig C.8

4.7 For every symbol that one wishes to transfer as a WINCC tag, select the symbol and right click on the symbol name.

4.8 From the menu that pops up, select the option called "Special Object Properties" in which there are three selections that one can make. Of these three, select the Operator Control and Monitoring option. At this point the following window will pop up

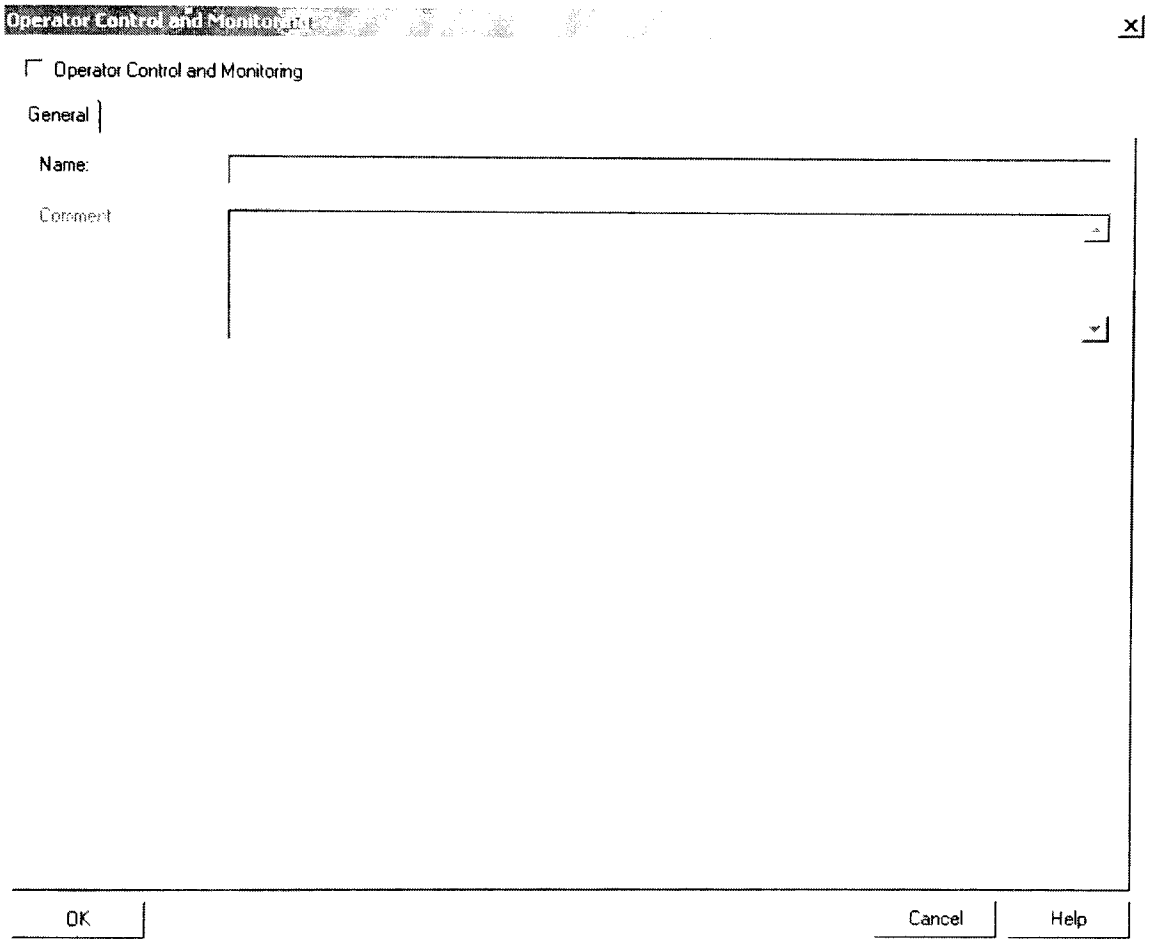


Fig C.9

4.9 Check the Operator Control and Monitoring check box and click OK.

4.10Save the Symbols Editor

Once the symbols are defined and categorized as Operator Control and Monitoring, the next step is to write the control logic for the application.

5.1 Click on the Blocks object in the objects window to get the OB1 object. Refer figure C.10

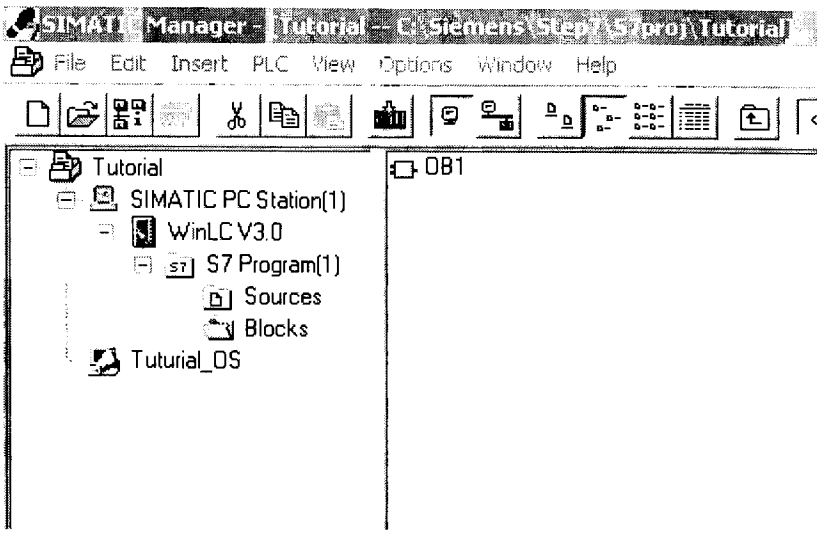


Fig C.10

5.2 Double click on the OB1 to get the LAD/FBD/STL window. This is the window in which the control logic for the application is written. The window looks as seen in the figure below

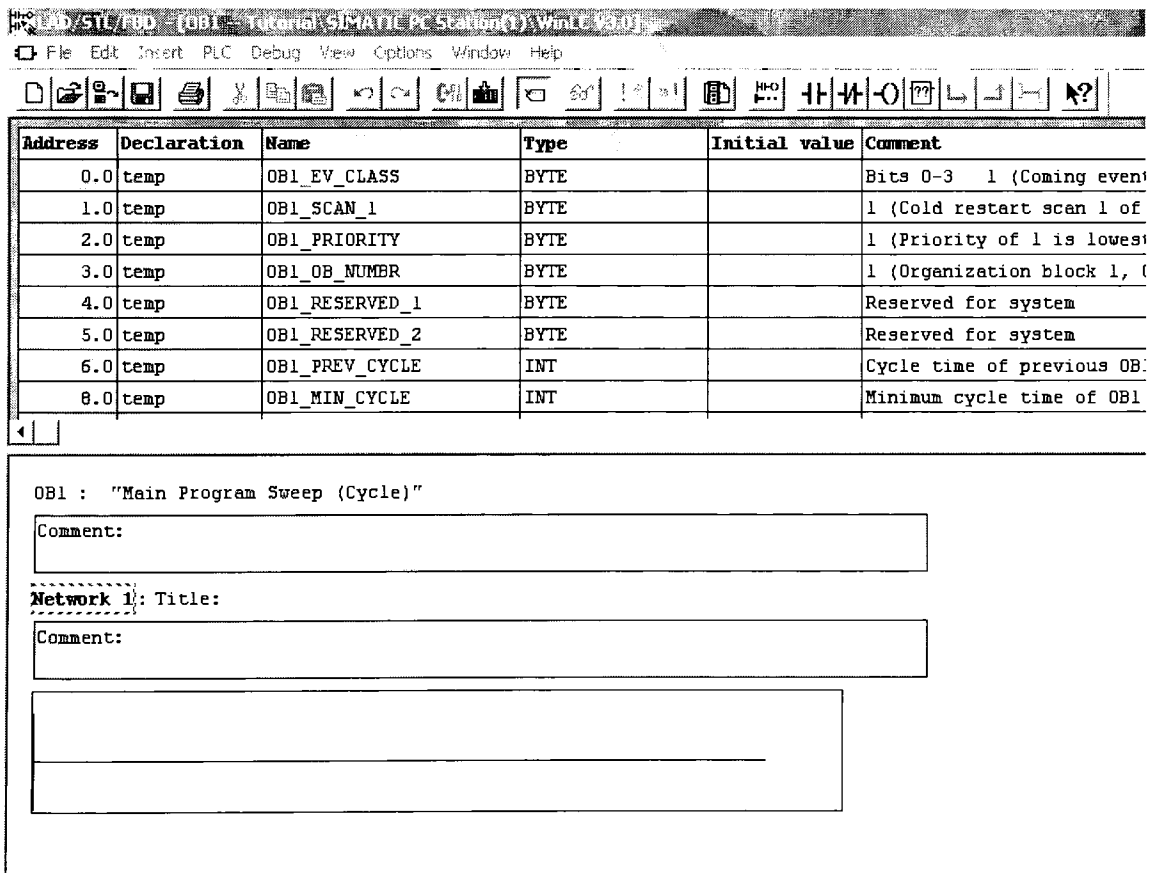


Fig C.11

5.3 Before starting to build the control logic, select the language in which you wish to build the logic. This can be done by checking either of the three options viz. **LAD, FBD,** STL option from the **View** menu of the LAD/FBD/STL window.

5.4 Now, one can build the control logic in any of the three programming languages. Although the logic is created in any of the three languages, the corresponding logic in the other two languages can be viewed by using the **View** menu.

5.5 For detailed help on programming with Step 7, refer the Siemens' Online Help (Go to Start > Simatic > Documentation > English. Refer the manuals Programming with Step 7 and Working with Step 7)

5.6 After the logic is completed, save the program and download it to the WinLC (with WinLC running in the Run-P mode)

5.7 The control logic is now executed on the field devices.

This completes the control program part of the project. Using Simatic WINCC software, the application can be monitored using various schematics and graphics. For this the symbols defined in the Step 7 environment have to be transferred as tags to the WINCC. This is the next step in the project.

17.1 In order to transfer the symbols defined in Step 7 to WINCC, select the **Transfer** option in the **PLC-OS Connection Data** sub-menu of the **Options** menu in the Simatic Manager window, as shown in the figure below.

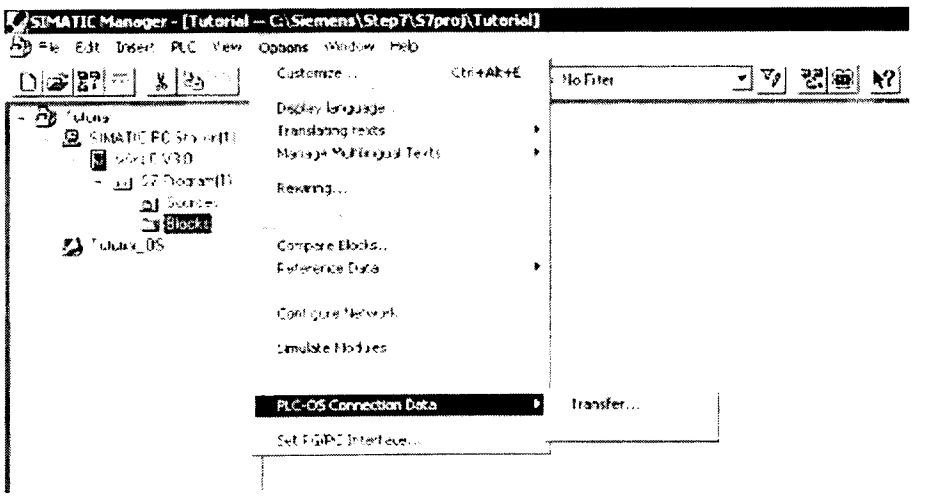


Fig C.12

6.2 When the Transfer option is clicked on, the following window pops up. This window is an introductory window, which lists the further steps that need to be taken for the transfer of symbols as tags.

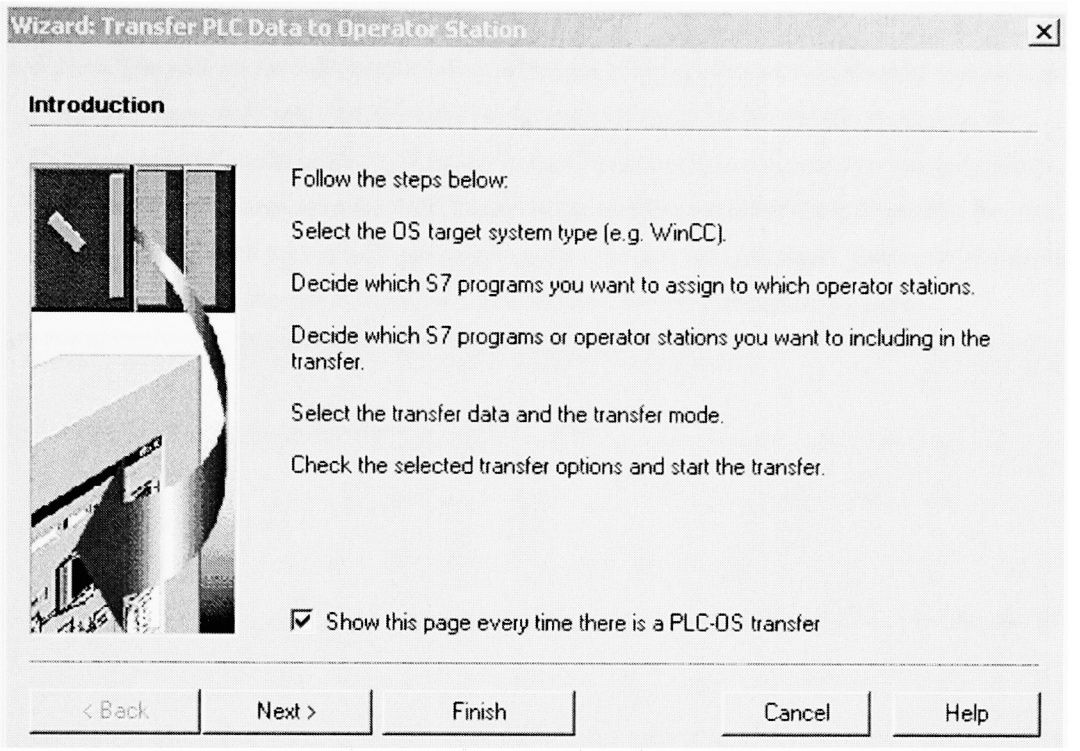


Fig C.13

6.3 Click on Next, to get the following window. In this window, select the program whose symbols

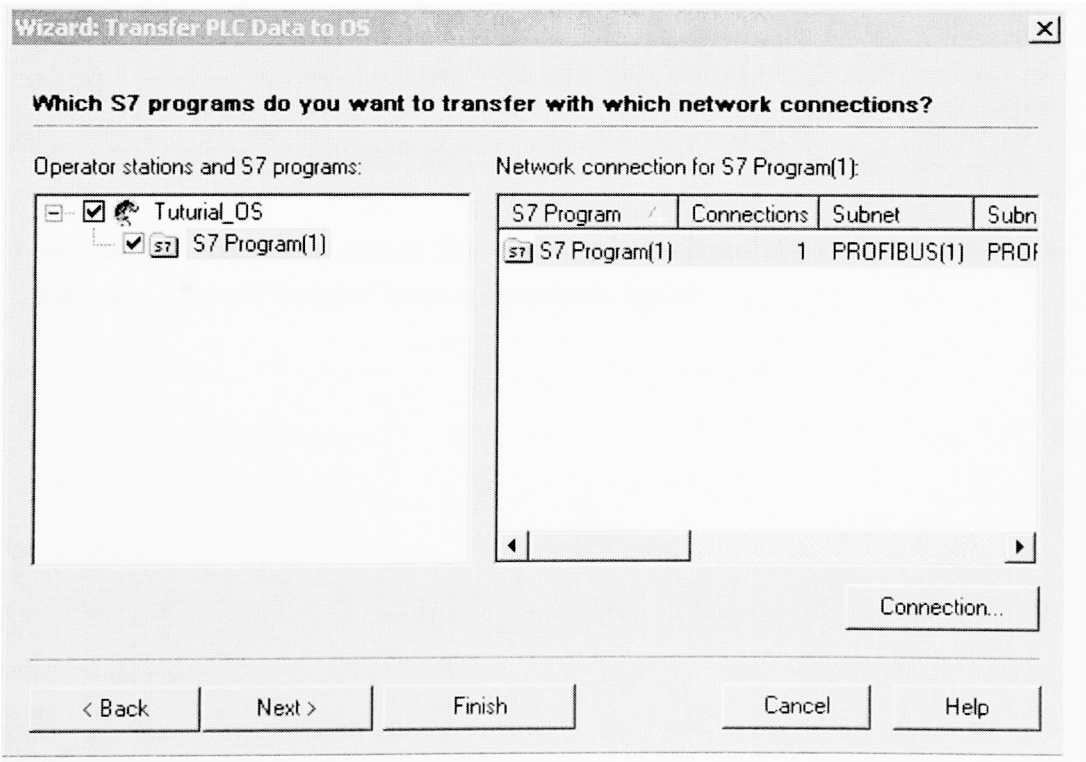


Fig C.14

you need to transfer as tags and then click Next.

6.4 Next, you will get the following window. Here check the box that says the transfer data as Tags and messages. Also select between the two options available for the transfer mode, viz. the "Changes only" mode or the "All" mode. In the Changes only mode, only the changed symbols are transferred, where as in the "All" mode, all the symbols are transferred. Typically, the "All" mode is used when the symbols are transferred for the first time and the "Changes only" mode is used during any subsequent transfers. Click on Next, once the selections are made.

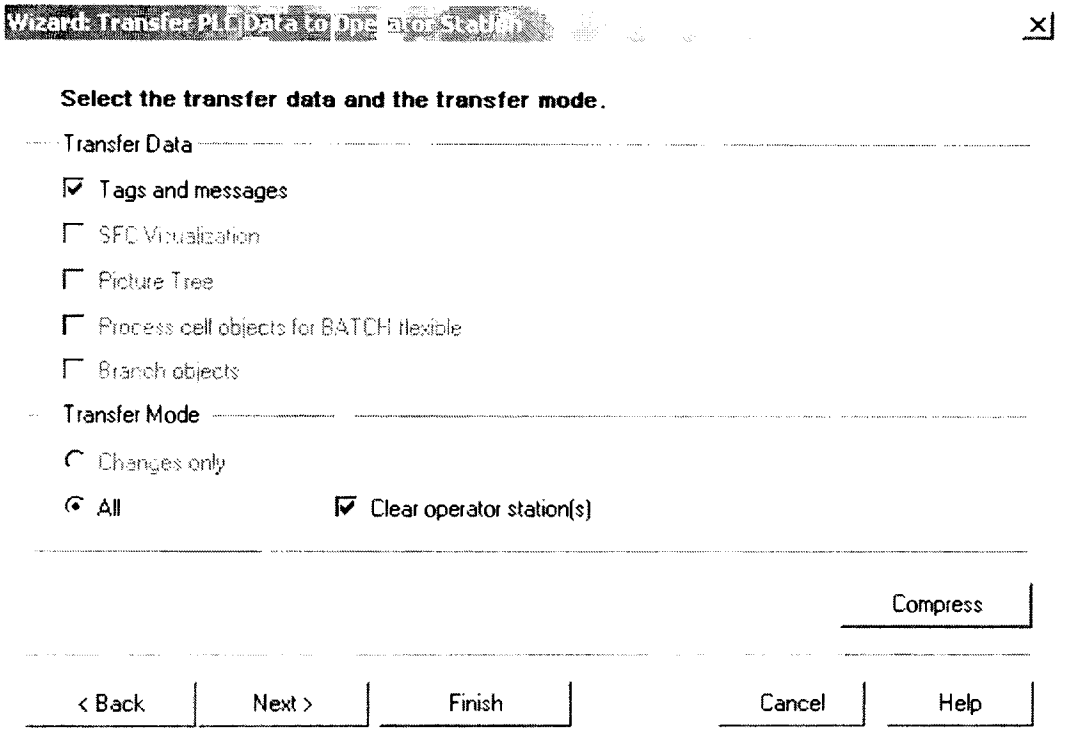


Fig C.15

6.5 Next, the following window pops up. This window gives a summary of the selections made in the last 3 steps. Click on "Transfer" button to complete the transfer.

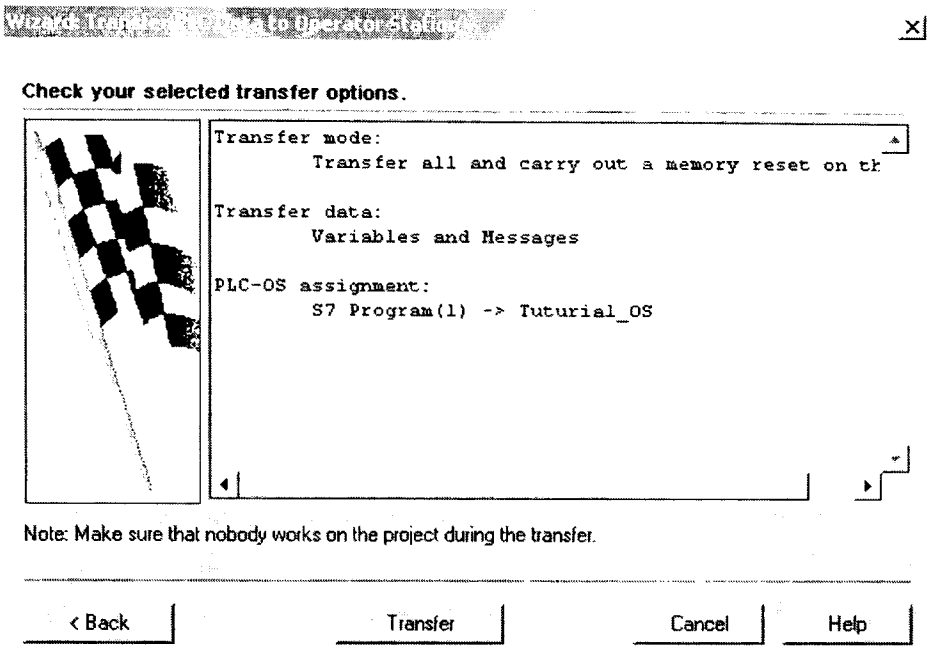


Fig C.16

Once the symbols are transferred as tags, these symbols are now available as tags in the WINCC. Next step is to open the Operator Station, which essential the WINCC window of the project. For this double click on the operator station object in the objects window of the project in the Simatic Manager. Altematively, you can right click on this object to open it.

Once WINCC is opened, the first step is to add a communication driver into the project.

7.1 On the left hand side of the project window in the WINCC Explorer window, there is a list of objects. One such object is the Tag Management object. Right Click on this object and select the "Add new driver" option from the menu that pops up. The following window will pop up.

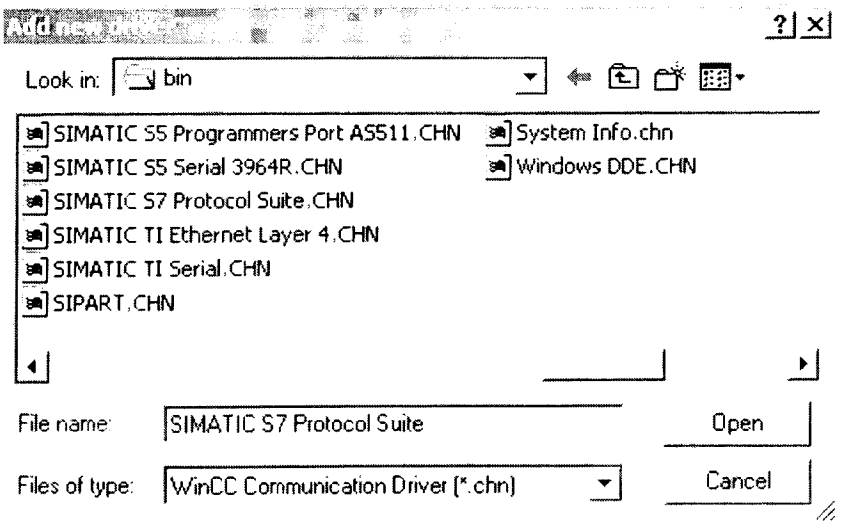


Fig C.17

7.2 From this window, select the "Simatic S7 Protocol Suite.CHN" driver. Adding this driver will enable the user to view the transferred tags.

7.3 Once the communication driver is added, the project window in the WINCC Explorer will look as below.

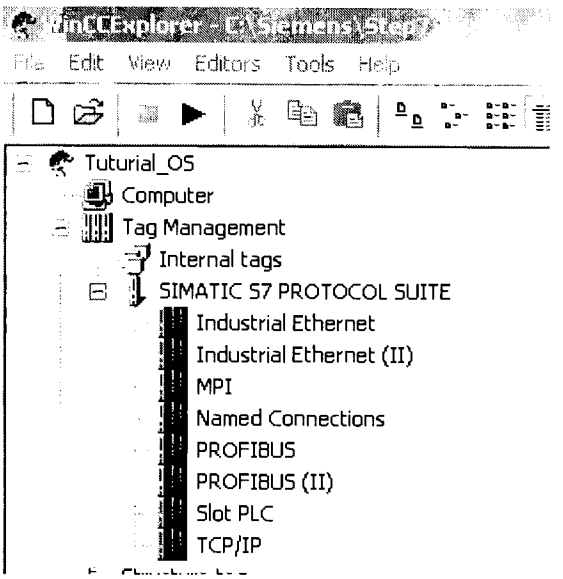
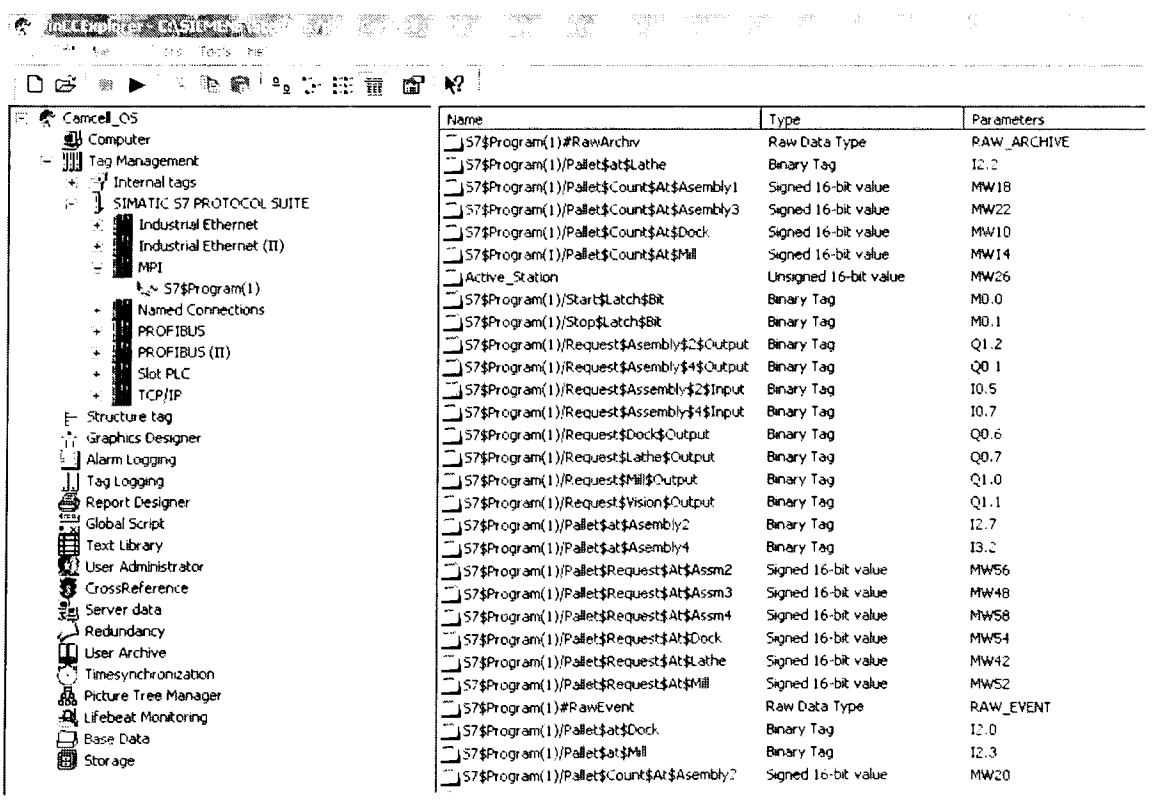


Fig C.18

7.4 If the MPI network is selected in the Simatic S7 Protocol suite, then the transferred tags can be viewed as seen in the figure below.



Next step is to create some graphics in the Graphics Designer Editor of WINCC to represent the process for which we built the control logic. For more information and help for using the Graphics Designer editor of WINCC, refer the WINCC online help manuals. (WINCC Doc#1- WINCC DOC#7, located in L:\Adwait\Siemens Doc) Once the graphic is created, activate the module using the icon. The module should be activated while the Step 7 program is running.

Appendix D. Accessing the WINCC tags via the OPC Channel

Purpose

The purpose of this tutorial-type document is to give step-by-step instructions to access the WINCC tags on the OPC server from a remote OPC client via the OPC channel.

Pre-requisite

In order to be able to access the tags resident on the OPC server, the machine on which the OPC server runs must be ON. Also the associated Step 7 and WINCC projects should also be running.

Procedure

The first step in order to access the WINCC tags on the OPC server from a remote OPC client via the OPC channel is to create a project in WINCC.

1.1 Open the WINCC from the Start button, **Start > Simatic > WINCC > WinCC Control Center 5.0**.

1.2 Once the Simatic Manager is opened, create a new project from the File menu of the Simatic Manager, **File > New**. Altematively, the new project can be created from the **New** icon

1.3 Once the New option from the **File** menu is selected, or the **New** icon is clicked, the following window appears

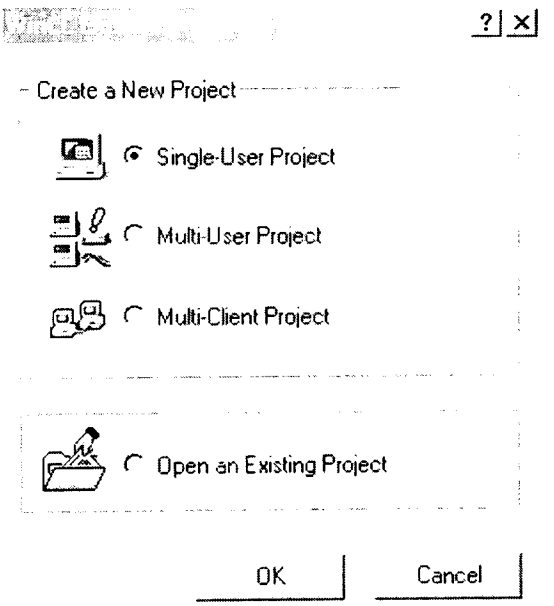


Fig D.1

1.4 In this window, select the project type as a Single-User project and click OK to continue.

1.5 Next, the following window pops up. In this window, type in the name of the project and also select the location to store the project

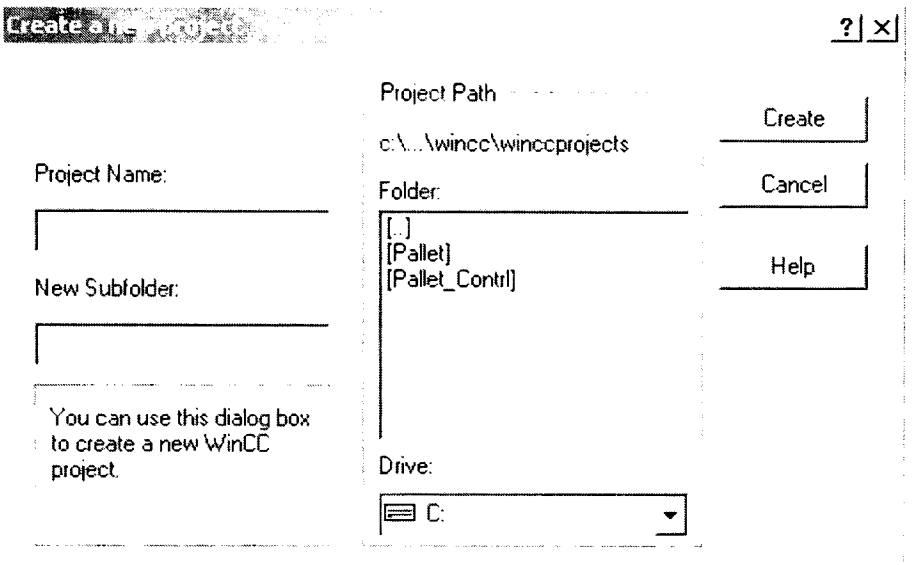


Fig D.2

1.6 Click on the **"Create"** button to create a project.

Once a project is created, the next step is to add a communication driver to this project. This communication driver, named the **"OPC.chn"**, enables the communication between the OPC client and the OPC server via the OPC channel.

2.1 Right click on the **Tag Management** object to get the following pop up menu

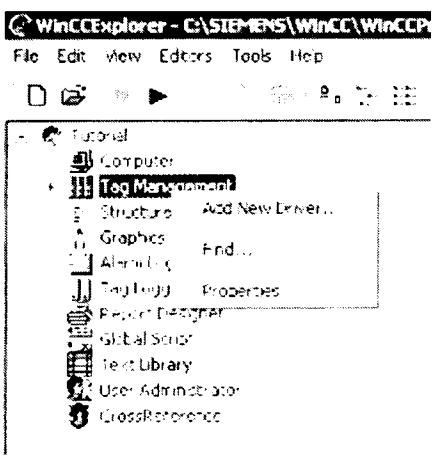


Fig D.3

2.2 From the three options available, select the **"Add New Driver"** option.

2.3 Once this option is selected the following window appears. This window has a list of all the communication drivers that can be added. The driver that we need to select here is the **"OPC.CHN"** as shown in the figure below.

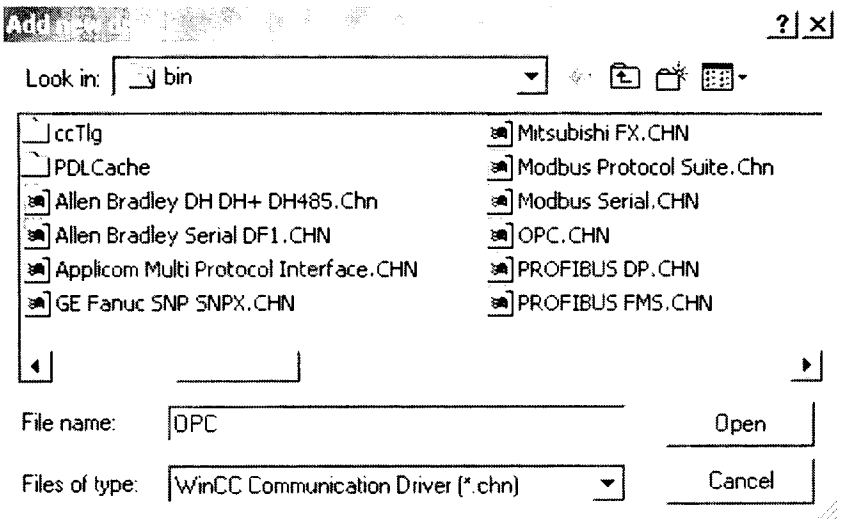


Fig D.4

2.4 Next, click on **Open** to select the driver.

Once the communication driver is selected, the next step is to browse through the network to the **OPC server (IMERT28)**

3.1 Within the Tag Management object browse down to the OPC groups.

3.2 Right click on the OPC Groups (OPCCHN Unit #1) object and select the **"System Parameters"** option from the pop menu that appears.

3.3 Once the **"System Parameters"** option is selected, the following window pops up.

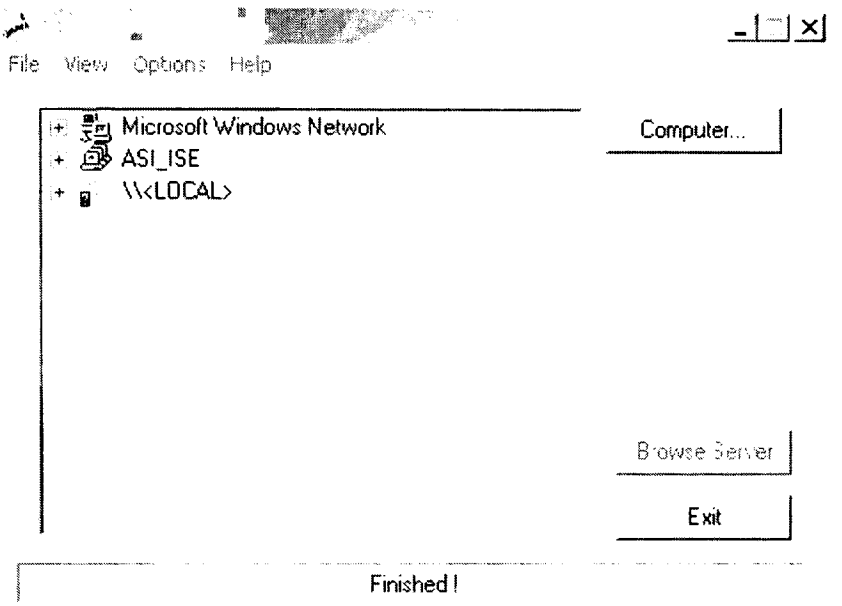


Fig D.5

3.4 In the OPC Item Manager window, there are three domains, viz. the Microsoft Windows Network, ASI_ISE, and the Local. Browse into the **ASI_ISE** domain and select the **IMERT 28** computer, which is the OPC server. This step is shown in the figure below

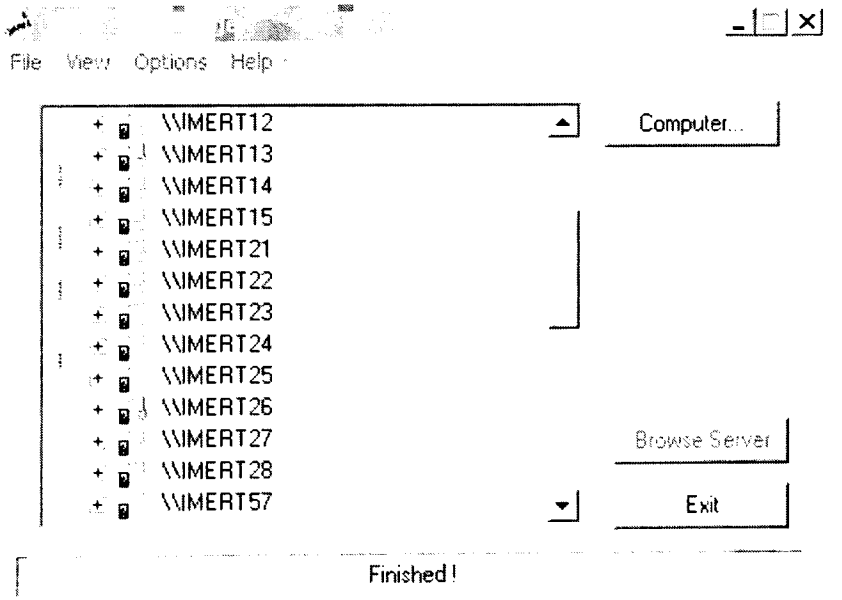


Fig D.6

3.5 Browse further into the **IMERT28**, and select the **OPC Server.WINCC** object and click on the **"Browse Server"** button on the right of the OPC Item Manager window. This is shown in the figure below.

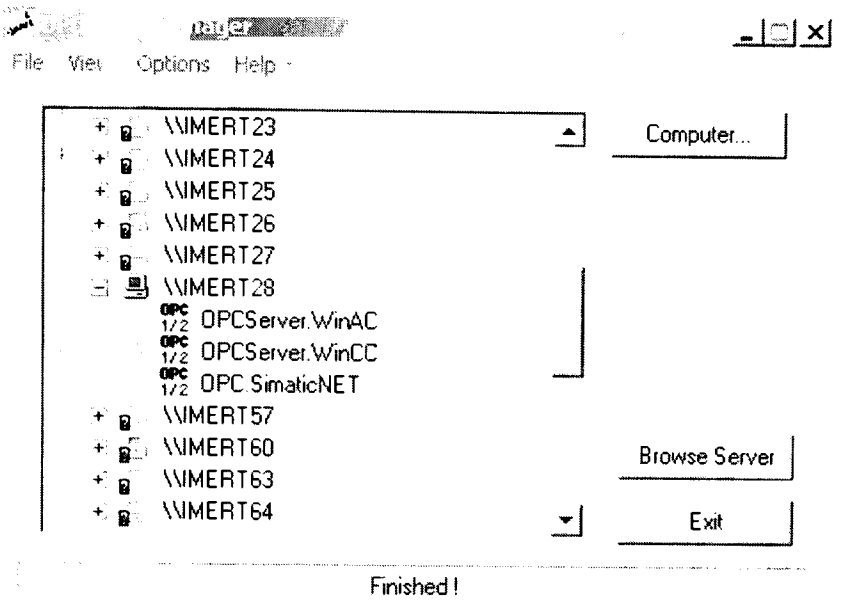


Fig D.7

Once you reach the OPC server, the next step is to browse it to access the tags within the server.

4.1 After clicking the "Browse Server" button in the previous step the following window will appear.

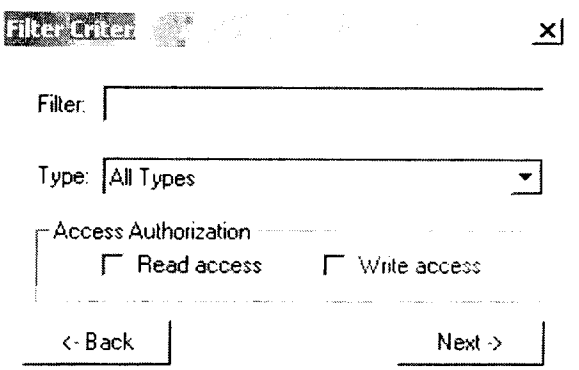


Fig D.8

4.2 Check the **"Read Access"** and the **"Write Access"** options and click **"Next"** to get the following window.

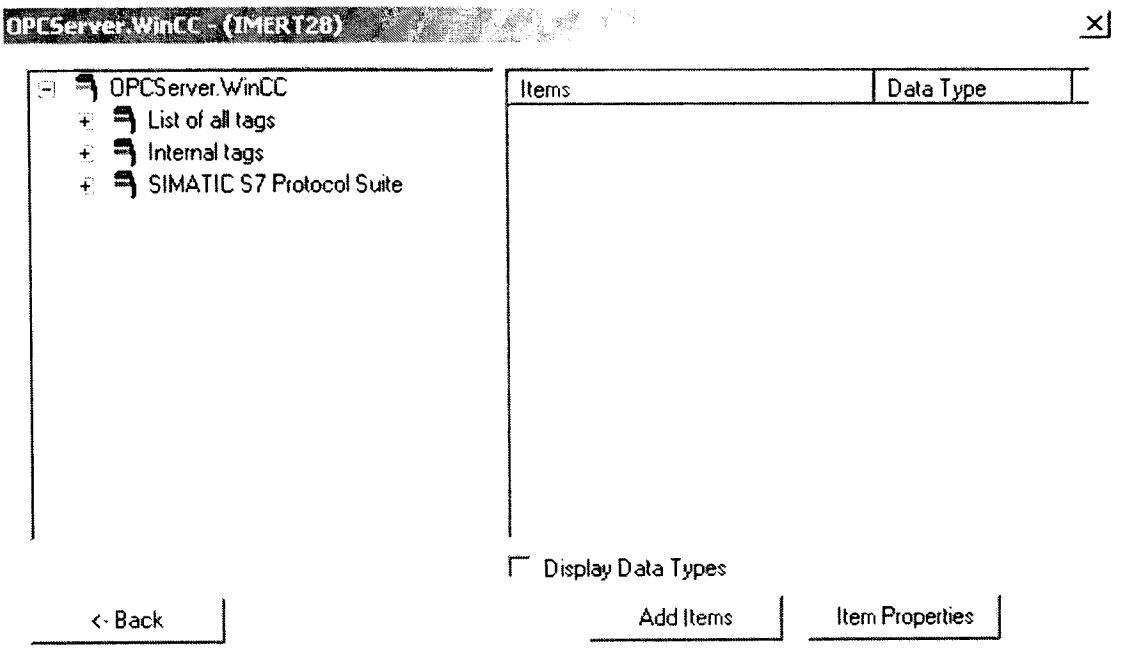


Fig D.9

4.3 Browse down in the **OPCServer.WINCC** object and select the **"List of all tags"** option so get all the tags resident in the OPC Server. This step is shown in the figure below.

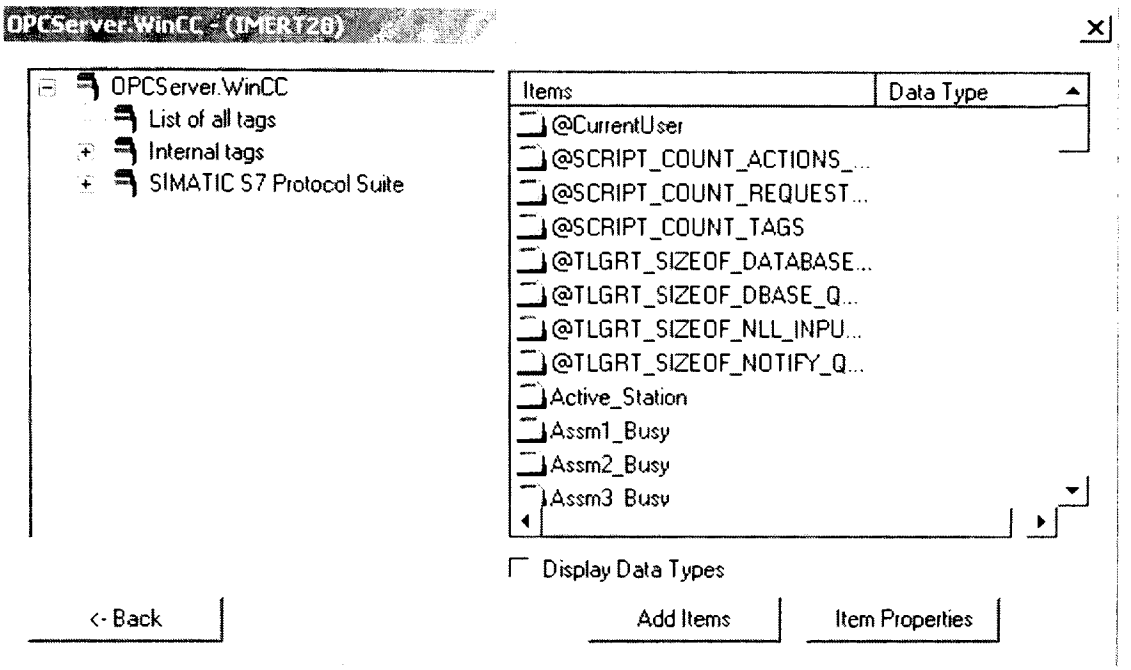


Fig D.10

4.4 The list of all tags resident in the OPC Server will be seen in the right pane of the window. Check the "Display data type" option to display the data type of the tags.

4.5 Finally, click on the "Add Items" button to add the tags into the current WINCC project.

Properties of symbol table

Name:	Symbols
Comment:	
Created on	09.04.2002 19:10:24
Last modified on:	25.04.2002 14:13:07
Last filter criterion:	Alle Symbole
Number of symbols:	73/ 73
Last Sorting:	Address Ascending

Symbol	Address	Data type	Comment
Dock Counter	C 10	COUNTER	Counter to keep track of number of pallets that passed the Dock.
Lathe Counter	C 20	COUNTER	Counter to keep track of number of pallets that passed the Lathe Station.
Mill Counter	C 30	COUNTER	Counter to keep track of number of pallets that passed the Mill Station.
Vision Counter	C 40	COUNTER	Counter to keep track of number of pallets that passed the Vision Station.
Assembly 1 Counter	C 50	COUNTER	Counter to keep track of number of pallets that passed the Assembly Station #1.
Assembly 2 Counter	C 60	COUNTER	Counter to keep track of number of pallets that passed the Assembly Station #2.
Assembly 3 Counter	C 70	COUNTER	Counter to keep track of number of pallets that passed the Assembly Station #3.
Assembly 4 Counter	C 80	COUNTER	Counter to keep track of number of pallets that passed the Assembly Station #4.
Request Dock Input	I 0.0	BOOL	Simulated Input to raise the latch at the Dock.
Request Lathe Input	I 0.1	BOOL	Simulated Input to raise the latch at the Lathe Station.
Request Mill Input	I 0.2	BOOL	Simulated Input to raise the latch at the Mill Station.
Request Vision Input	I 0.3	BOOL	Simulated Input to raise the latch at the Vision Station.
Request Assembly 1 Input	I 0.4	BOOL	Simulated Input to raise the latch at the Assembly Station #1.
Request Assembly 2 Input	I 0.5	BOOL	Simulated Input to raise the latch at the Assembly Station #2.
Request Assembly 3 Input	I 0.6	BOOL	Simulated Input to raise the latch at the Assembly Station #3.
Request Assembly 4 Input	I 0.7	BOOL	Simulated Input to raise the latch at the Assembly Station #4.
Start Button	I 1.0	BOOL	Simulated Input for the Start Button.
Stop Button	I 1.1	BOOL	Simulated Input for the Stop Button.
Pallet at Dock	I 2.0	BOOL	Input to indicate the pallet presence at the Dock.
Pallet at Lathe	I 2.2	BOOL	Input to indicate the pallet presence at the Lathe Station.
Pallet at Mill	I 2.3	BOOL	Input to indicate the pallet presence at the Mill Station
Pallet at Vision	I 2.4	BOOL	Input to indicate the pallet presence at the Vision Station.
Pallet at Asembly1	I 2.6	BOOL	Input to indicate the pallet presence at the Assembly Station #1.
Pallet at Asembly2	I 2.7	BOOL	Input to indicate the pallet presence at the Assembly Station #2.
Pallet at Asembly3	I 3.1	BOOL	Input to indicate the pallet presence at the Assembly Station #3.
Pallet at Asembly4	I 3.2	BOOL	Input to indicate the pallet presence at the Assembly Station #4.
Start Latch Bit	M 0.0	BOOL	Memory Bit for latching of the Start Button.
Stop Latch Bit	M 0.1	BOOL	Memory Bit for latching of the Stop Button.

Symbol	Address	Data type	Comment
Pallet Stop At Lathe	M 1.0	BOOL	Memory Bit to raise the latch at the Lathe Station.
Pallet Stop At Mill	M 2.0	BOOL	Memory Bit to raise the latch at the Mill Station.
Pallet Stop At Vision	M 3.0	BOOL	Memory Bit to raise the latch at the Vision Station.
Pallet Stop At Dock	M 4.0	BOOL	Memory Bit to raise the latch at the Dock.
Pallet Stop At Assm1	M 5.0	BOOL	Memory Bit to raise the latch at the Assembly Station # 1.
Pallet Stop At Assm2	M 6.0	BOOL	Memory Bit to raise the latch at the Assembly Station # 2.
Pallet Stop At Assm3	M 7.0	BOOL	Memory Bit to raise the latch at the Assembly Station # 3.
Pallet Stop At Assm4	M 8.0	BOOL	Memory Bit to raise the latch at the Assembly Station # 4.
Pallet Count At Dock	MW 10	INT	Memory variable to store the value of the counter C10.
Pallet Count At Lathe	MW 12	INT	Memory variable to store the value of the counter C20.
Pallet Count At Mill	MW 14	INT	Memory variable to store the value of the counter C30.
Pallet Count At Vision	MW 16	INT	Memory variable to store the value of the counter C40.
Pallet Count At Asembly1	MW 18	INT	Memory variable to store the value of the counter C50.
Pallet Count At Asembly2	MW 20	INT	Memory variable to store the value of the counter C60.
Pallet Count At Asembly3	MW 22	INT	Memory variable to store the value of the counter C70.
Pallet Count At Asembly4	MW 24	INT	Memory variable to store the value of the counter C80.
Pallet Request At Lathe	MW 42	INT	Variable to store value of pallet # requested to be stopped at Lathe Station.
Pallet Request At Vision	MW 44	INT	Variable to store value of pallet # requested to be stopped at Vision Station.
Pallet Request At Assm1	MW 46	INT	Variable to store value of pallet # requested to be stopped at Assm Station #1.
Pallet Request At Assm3	MW 48	INT	Variable to store value of pallet # requested to be stopped at Assm Station #3.
Pallet Request At Mill	MW 52	INT	Variable to store value of pallet # requested to be stopped at Mill Station.
Pallet Request At Dock	MW 54	INT	Variable to store value of pallet # requested to be stopped at Dock.
Pallet Request At Assm2	MW 56	INT	Variable to store value of pallet # requested to be stopped at Assm Station #2.
Pallet Request At Assm4	MW 58	INT	Variable to store value of pallet # requested to be stopped at Assm Station #3.
Next Pallet At Dock	MW 100	INT	Variable to store the value of the next pallet arriving at the Dock.
Next Pallet At Lathe	MW 120	INT	Variable to store the value of the next pallet arriving at the Lathe Station.
Next Pallet At Mill	MW 140	INT	Variable to store the value of the next pallet arriving at the Mill Station.
Next Pallet At Vision	MW 160	INT	Variable to store the value of the next pallet arriving at the Vision Station.
Next Pallet At Assembly1	MW 180	INT	Variable to store the value of the next pallet arriving at the Assm Station #1.
Next Pallet At Assembly2	MW 200	INT	Variable to store the value of the next pallet arriving at the Assm Station #2.
prakash	MW 210	INT	
Next Pallet At Assembly3	MW 220	INT	Variable to store the value of the next pallet arriving at the Assm Station #3.
Next Pallet At Assembly4	MW 240	INT	Variable to store the value of the next pallet arriving at the Assm Station #4.
# of Pallets in System	MW 256	INT	Variable to store the value of the number of pallets circulating in the system
Inquire	OB 1	OB 1	
Pallet Controller	OB 2	OB 2	

Symbol	Address	Data type	Comment
Request Asembly 3 Output	Q 0.0	BOOL	Output that raises/lowers the latch at the Assembly Station #3.
Request Asembly 4 Output	Q 0.1	BOOL	Output that raises/lowers the latch at the Assembly Station #4.
Request Dock Output	Q 0.6	BOOL	Output that raises/lowers the latch at the Dock.
Request Lathe Output	Q 0.7	BOOL	Output that raises/lowers the latch at the Lathe Station.
Request Mill Output	Q 1.0	BOOL	Output that raises/lowers the latch at the Mill Station.
Request Vision Output	Q 1.1	BOOL	Output that raises/lowers the latch at the Vision Station.
Request Asembly 2 Output	Q 1.2	BOOL	Output that raises/lowers the latch at the Assembly Station #2.
Request Asembly 1 Output	Q 1.3	BOOL	Output that raises/lowers the latch at the Assembly Station #1.
VAT_1	VAT 1		

OB1 - <offline>

"Inquire"
Name: **Family:**
Author: **Version:** 0.1
Block version: 2
Time stamp Code: 04/14/2002 06:46:34 PMPM
Interface: 02/15/1996 04:51:12 PMPM
Lengths (block/logic/data): 00696 00498 00020

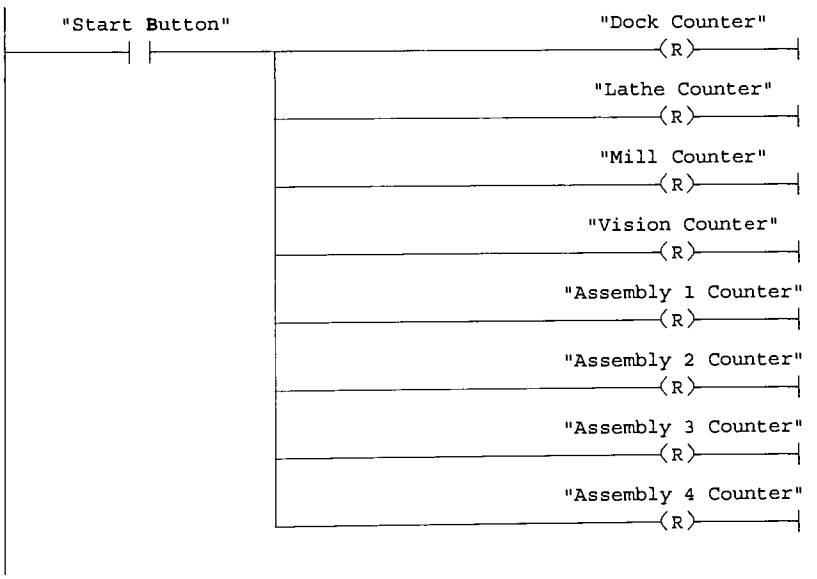
Address	Declaration	Name	Type	Initial value	Comment
0.0	temp	OB1_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
1.0	temp	OB1_SCAN_1	BYTE		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
2.0	temp	OB1_PRIORITY	BYTE		1 (Priority of 1 is lowest)
3.0	temp	OB1_OB_NUMBR	BYTE		1 (Organization block 1, OB1)
4.0	temp	OB1_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB1_RESERVED_2	BYTE		Reserved for system
6.0	temp	OB1_PREV_CYCLE	INT		Cycle time of previous OB1 scan (milliseconds)
8.0	temp	OB1_MIN_CYCLE	INT		Minimum cycle time of OB1 (milliseconds)
10.0	temp	OB1_MAX_CYCLE	INT		Maximum cycle time of OB1 (milliseconds)
12.0	temp	OB1_DATE_TIME	DATE_AND_TIME		Date and time OB1 started

Block: OB1 "Main Program Sweep (Cycle)"

Networks 1 through 28 are a part of the Inquire Program

Network: 1 COUNTER RESET

When the Start Button is started, all the counters used to keep track of the pallets at various stations are reset to zero.



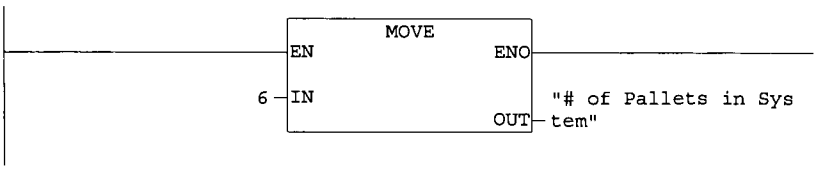
Symbol information

I1.0	Start Button	Simulated Input for the Start Button.
C10	Dock Counter	Counter to keep track of number of pallets that passed the Dock.
C20	Lathe Counter	Counter to keep track of number of pallets that passed the Lathe Station.
C30	Mill Counter	Counter to keep track of number of pallets that passed the Mill Station.
C40	Vision Counter	Counter to keep track of number of pallets that passed the Vision Station.
C50	Assembly 1 Counter	Counter to keep track of number of pallets that passed the Assembly Station #1.

C60 Assembly 2 Counter Counter to keep track of number of pallets that passed the Assembly Station #2.
C70 Assembly 3 Counter Counter to keep track of number of pallets that passed the Assembly Station #3.
C80 Assembly 4 Counter Counter to keep track of number of pallets that passed the Assembly Station #4.

Network: 2

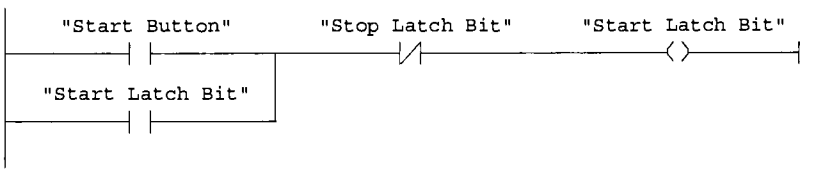
Move the value of # of pallets in the system (here, 6) in a memory variable MW256.



Symbol information
MW256 # of Pallets in System Variable to store the value of the number of pallets circulating in the system

Network: 3 START BUTTON

We use the Simulator Switch (#I1.0)as a start button to represent a push button for START. Hence, in order to provide for the latching of the start button, we set a bit for Start button high(viz. Start Latch Bit). The bit goes low when the Stop Latch Bit (turned high by the Stop Button)goes high.



Symbol information
I1.0 Start Button Simulated Input for the Start Button.
M0.0 Start Latch Bit Memory Bit for latching of the Start Button.
M0.1 Stop Latch Bit Memory Bit for latching of the Stop Button.

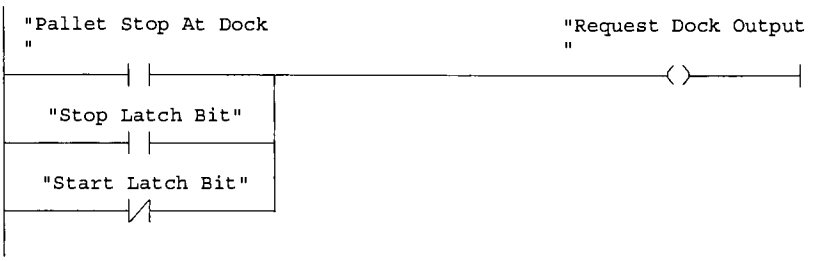
Network: 4 STOP BUTTON

When the Stop Button (represented by a simulator switch on the interface module) is pressed, the Stop Latch Bit is set to high.

Symbol information
I1.1 Stop Button Simulated Input for the Stop Button.
M0.1 Stop Latch Bit Memory Bit for latching of the Stop Button.

Network: 5 Pallet Stop Request at the Dock

The latch at Dock is normally raised. The latch is lowered when the start button is pressed. This latch is raised again when either a pallet request at the Dock is made or when the Stop button is pressed.



Symbol information

M4.0	Pallet Stop At Dock	Memory Bit to raise the latch at the Dock.
M0.1	Stop Latch Bit	Memory Bit for latching of the Stop Button.
M0.0	Start Latch Bit	Memory Bit for latching of the Start Button.
Q0.6	Request Dock Output	Output that raises/lowers the latch at the Dock.

Network: 6 Pallet Stop Request at the Lathe

The latch at the Lathe station is in a lowered state in its normal operation. This latch is raised only when a pallet request is made by the lathe station.

Symbol information

M1.0	Pallet Stop At Lathe	Memory Bit to raise the latch at the Lathe Station.
Q0.7	Request Lathe Output	Output that raises/lowers the latch at the Lathe Station.

Network: 7 Pallet Stop Request at the Mill

The latch at the Mill station is in a lowered state in its normal operation. This latch is raised only when a pallet request is made by the Mill station.

Symbol information

M2.0	Pallet Stop At Mill	Memory Bit to raise the latch at the Mill Station.
Q1.0	Request Mill Output	Output that raises/lowers the latch at the Mill Station.

Network: 8 Pallet Stop Request at the Vision

The latch at the Vision station is in a lowered state in its normal operation. This latch is raised only when a pallet request is made by the vision station.

Symbol information

M3.0	Pallet Stop At Vision	Memory Bit to raise the latch at the Vision Station.
Q1.1	Request Vision Output	Output that raises/lowers the latch at the Vision Station.

Network: 9 Pallet Stop Request at the Assembly #1

The latch at the Assembly Station #1 is in a lowered state in its normal operation. This latch is raised only when a pallet request is made by the Assembly Station #1.

Symbol information

M5.0	Pallet Stop At Assm1	Memory Bit to raise the latch at the Assembly Station #1.
Q1.3	Request Asembly 1 Output	Output that raises/lowers the latch at the Assembly Station # 1.

Network: 10 Pallet Stop Request at the Assembly #2

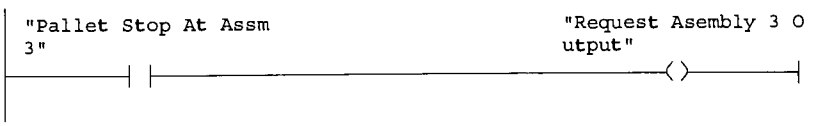
The latch at the Assembly Station #2 is in a lowered state in its normal operation. This latch is raised only when a pallet request is made by the Assembly Station #2.

Symbol information

M6.0	Pallet Stop At Assm2	Memory Bit to raise the latch at the Assembly Station #2.
Q1.2	Request Asembly 2 Output	Output that raises/lowers the latch at the Assembly Station # 2.

Network: 11 Pallet Stop Request at the Assembly #3

The latch at the Assembly Station #3 is in a lowered state in its normal operation. This latch is raised only when a pallet request is made by the Assembly Station #3.



Symbol information

M7.0	Pallet Stop At Assm3	Memory Bit to raise the latch at the Assembly Station #3.
Q0.0	Request Asembly 3 Output	Output that raises/lowers the latch at the Assembly Station # 3.

Network: 12 Pallet Stop Request at the Assembly #4

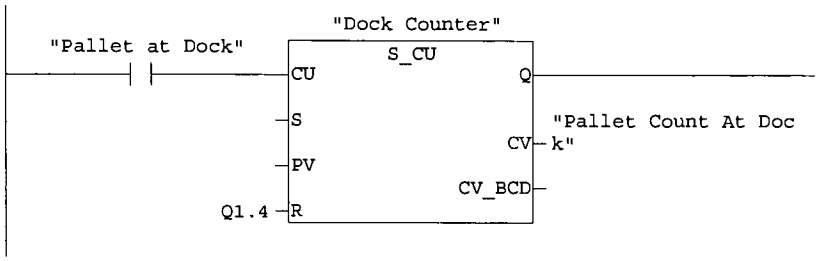
The latch at the Assembly Station #4 is in a lowered state in its normal operation. This latch is raised only when a pallet request is made by the Assembly Station #4.

Symbol information

M8.0	Pallet Stop At Assm4	Memory Bit to raise the latch at the Assembly Station #4.
Q0.1	Request Asembly 4 Output	Output that raises/lowers the latch at the Assembly Station # 4.

Network: 13 Pallet Counter (C10) for the Dock Station.

An Up-Counter (C10) is used to keep track of the pallets passing the Dock Station. The counter increments whenever a pallet leaves the Dock Station. The pallet count value is stored in a memory variable MW10.

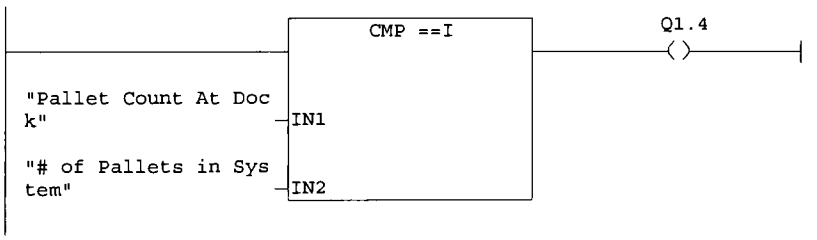


Symbol information

I2.0	Pallet at Dock	Input to indicate the pallet presence at the Dock.
C10	Dock Counter	Counter to keep track of number of pallets that passed the Dock.
MW10	Pallet Count At Dock	Memory variable to store the value of the counter C10.

Network: 14

The pallet count value for the Dock Station, stored in MW10 in the previous network is compared with the variable MW256 that contains the value for number of pallets in the system. When the value of these two variables is equal, an output Q1.4 turns high. This output is used to reset the counter C10 to zero in the previous network.

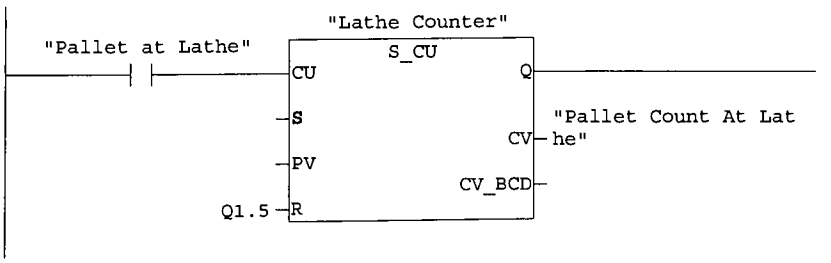


Symbol information

MW10	Pallet Count At Dock	Memory variable to store the value of the counter C10.
MW256	# of Pallets in System	Variable to store the value of the number of pallets circulating in the system

Network: 15 Pallet Counter (C20) for the Lathe

An Up-Counter (C20) is used to keep track of the pallets passing the Lathe Station. The counter increments whenever a pallet leaves the Lathe Station. The pallet count value is stored in a memory variable MW12.

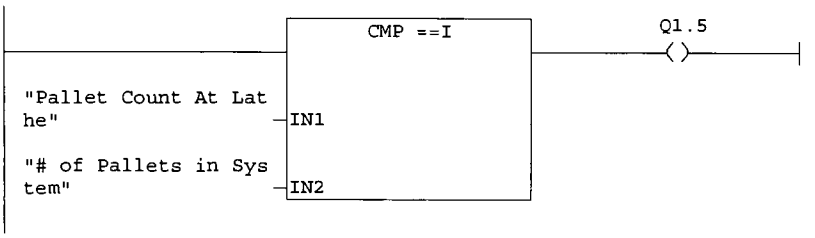


Symbol information

I2.2	Pallet at Lathe	Input to indicate the pallet presence at the Lathe Station.
C20	Lathe Counter	Counter to keep track of number of pallets that passed the Lathe Station.
MW12	Pallet Count At Lathe	Memory variable to store the value of the counter C20.

Network: 16

The pallet count value for the Lathe Station, stored in MW12 in the previous network is compared with the variable MW256 that contains the value for number of pallets in the system. When the value of these two variables is equal, an output Q1.5 turns high. This output is used to reset the counter C20 to zero in the previous network.

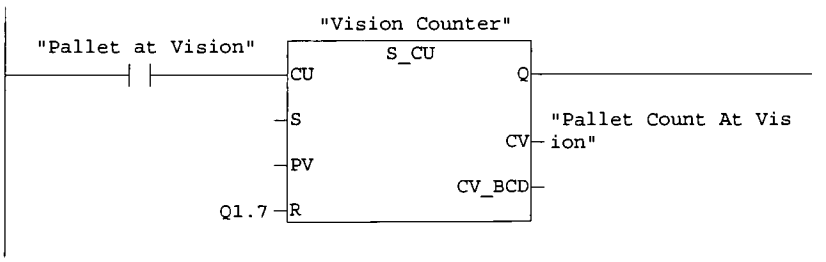


Symbol information

MW12	Pallet Count At Lathe	Memory variable to store the value of the counter C20.
MW256	# of Pallets in System	Variable to store the value of the number of pallets circulating in the system

Network: 17 Pallet Counter (C40) for the Vision

An Up-Counter (C40) is used to keep track of the pallets passing the Vision Station. The counter increments whenever a pallet leaves the Vision Station. The pallet count value is stored in a memory variable MW16.

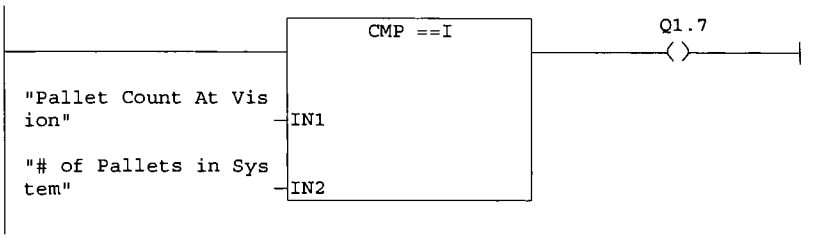


Symbol information

I2.4	Pallet at Vision	Input to indicate the pallet presence at the Vision Station.
C40	Vision Counter	Counter to keep track of number of pallets that passed the Vision Station.
MW16	Pallet Count At Vision	Memory variable to store the value of the counter C40.

Network: 18

The pallet count value for the Vision Station, stored in MW16 in the previous network is compared with the variable MW256 that contains the value for number of pallets in the system. When the value of these two variables is equal, an output Q1.7 turns high. This output is used to reset the counter C40 to zero in the previous network.

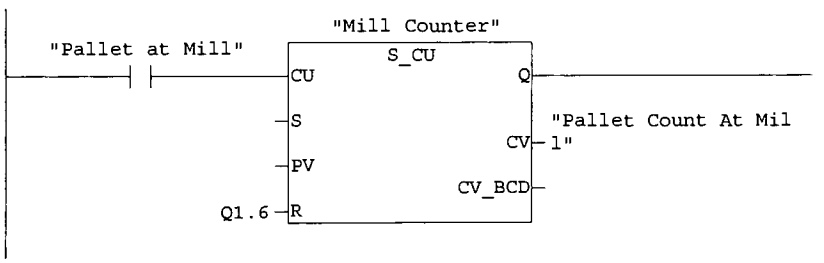


Symbol information

MW16	Pallet Count At Vision	Memory variable to store the value of the counter C40.
MW256	# of Pallets in System	Variable to store the value of the number of pallets circulating in the system

Network: 19 Pallet Counter (C30) for the Mill

An Up-Counter (C30) is used to keep track of the pallets passing the Mill Station. The counter increments whenever a pallet leaves the Mill Station. The pallet count value is stored in a memory variable MW14

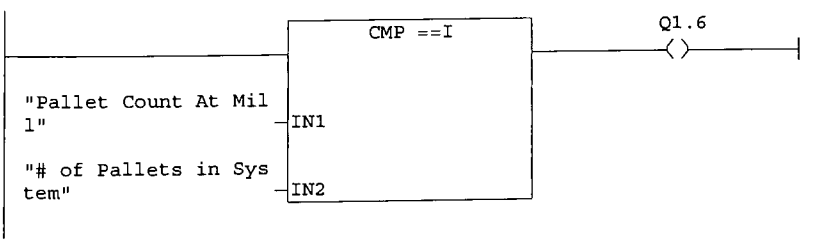


Symbol information

I2.3	Pallet at Mill	Input to indicate the pallet presence at the Mill Station
C30	Mill Counter	Counter to keep track of number of pallets that passed the Mill Station.
MW14	Pallet Count At Mill	Memory variable to store the value of the counter C30.

Network: 20

The pallet count value for the Mill Station, stored in MW14 in the previous network is compared with the variable MW256 that contains the value for number of pallets in the system. When the value of these two variables is equal, an output Q1.6 turns high. This output is used to reset the counter C30 to zero in the previous network.

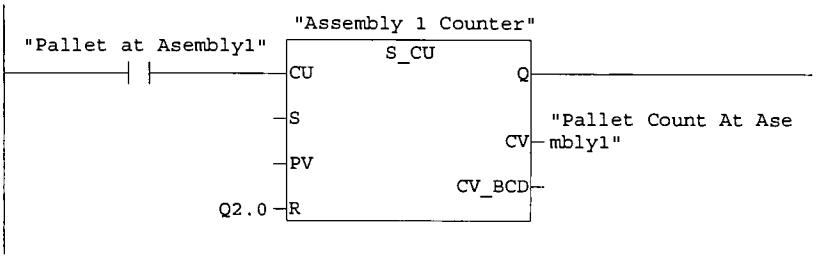


Symbol information

MW14	Pallet Count At Mill	Memory variable to store the value of the counter C30.
MW256	# of Pallets in System	Variable to store the value of the number of pallets circulating in the system

Network: 21 Pallet Counter (C50) for the Assembly 1

An Up-Counter (C50) is used to keep track of the pallets passing the Assembly Station #1. The counter increments whenever a pallet leaves the Assembly Station #1. The pallet count value is stored in a memory variable MW18.

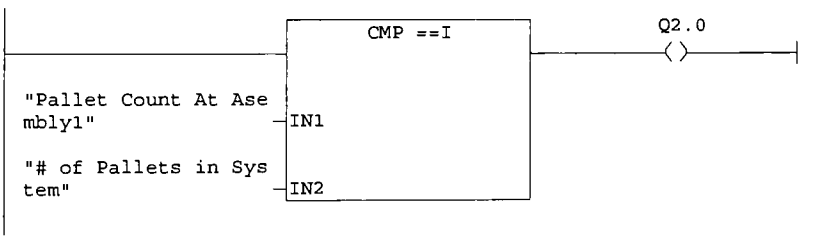


Symbol information

I2.6	Pallet at Asembly1	Input to indicate the pallet presence at the Assembly Station #1.
C50	Assembly 1 Counter	Counter to keep track of number of pallets that passed the Assembly Station #1.
MW18	Pallet Count At Asembly1	Memory variable to store the value of the counter C50.

Network: 22

The pallet count value for the Assembly Station #1, stored in MW18 in the previous network is compared with the variable MW256 that contains the value for number of pallets in the system. When the value of these two variables is equal, an output Q2.0 turns high. This output is used to reset the counter C50 to zero in the previous network.

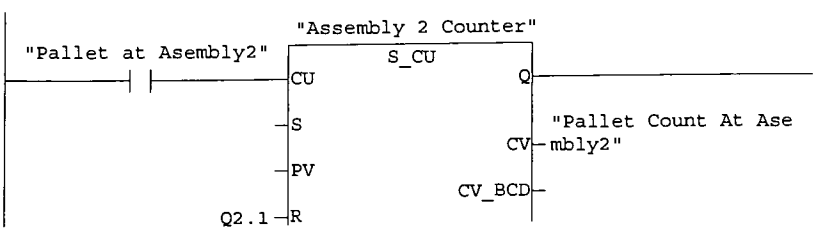


Symbol information

MW18	Pallet Count At Asembly1	Memory variable to store the value of the counter C50.
MW256	# of Pallets in System	Variable to store the value of the number of pallets circulating in the system

Network: 23 Pallet Counter (C60) for the Assembly 2

An Up-Counter (C60) is used to keep track of the pallets passing the Assembly Station #2. The counter increments whenever a pallet leaves the Assembly Station #2. The pallet count value is stored in a memory variable MW20.

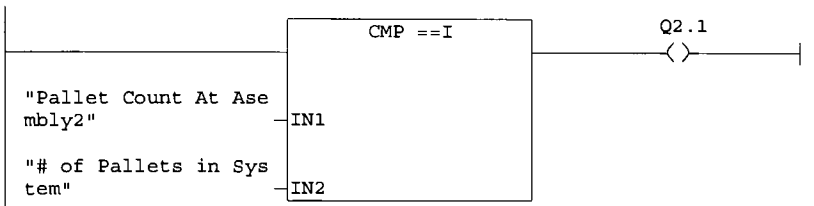


Symbol information

I2.7	Pallet at Asembly2	Input to indicate the pallet presence at the Assembly Station #2.
C60	Assembly 2 Counter	Counter to keep track of number of pallets that passed the Assembly Station #2.
MW20	Pallet Count At Asembly2	Memory variable to store the value of the counter C60.

Network: 24

The pallet count value for the Assembly Station #2, stored in MW20 in the previous network is compared with the variable MW256 that contains the value for number of pallets in the system. When the value of these two variables is equal, an output Q2.1 turns high. This output is used to reset the counter C60 to zero in the previous network.

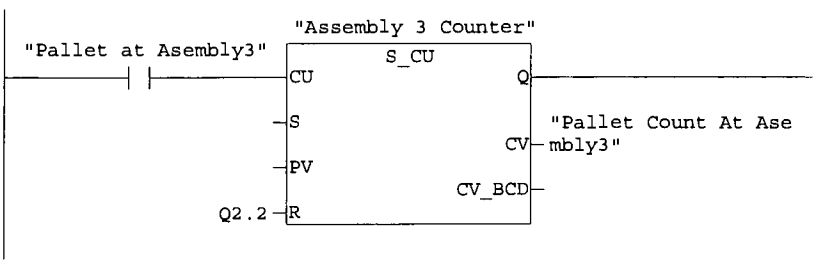


Symbol information

MW20	Pallet Count At Asembly2	Memory variable to store the value of the counter C60.
MW256	# of Pallets in System	Variable to store the value of the number of pallets circulating in the system

Network: 25 Pallet Counter (C70) for the Assembly 3

An Up-Counter (C70) is used to keep track of the pallets passing the Assembly Station #3. The counter increments whenever a pallet leaves the Assembly Station #3. The pallet count value is stored in a memory variable MW22.

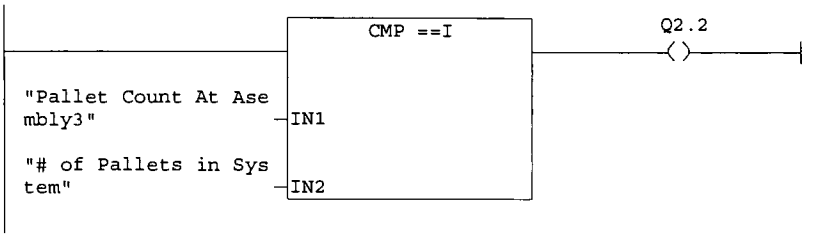


Symbol information

I3.1	Pallet at Asembly3	Input to indicate the pallet presence at the Assembly Station #3.
C70	Assembly 3 Counter	Counter to keep track of number of pallets that passed the Assembly Station #3.
MW22	Pallet Count At Asembly3	Memory variable to store the value of the counter C70.

Network: 26

The pallet count value for the Assembly Station #3, stored in MW22 in the previous network is compared with the variable MW256 that contains the value for number of pallets in the system. When the value of these two variables is equal, an output Q2.2 turns high. This output is used to reset the counter C70 to zero in the previous network.

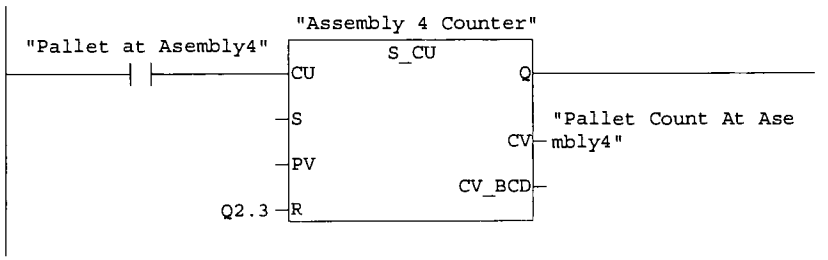


Symbol information

MW22	Pallet Count At Asembly3	Memory variable to store the value of the counter C70.
MW256	# of Pallets in System	Variable to store the value of the number of pallets circulating in the system

Network: 27 Pallet Counter (C80) for the Assembly 4

An Up-Counter (C80) is used to keep track of the pallets passing the Assembly Station #4. The counter increments whenever a pallet leaves the Assembly Station #4. The pallet count value is stored in a memory variable MW24.

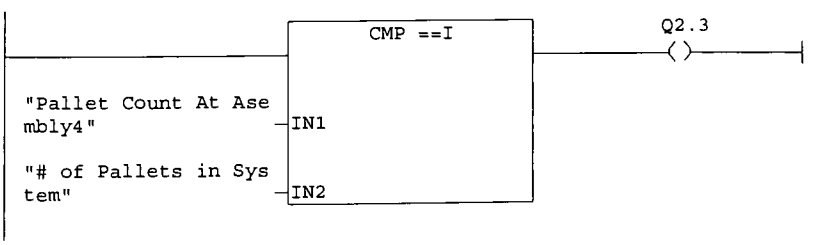


Symbol information

I3.2	Pallet at Asembly4	Input to indicate the pallet presence at the Assembly Station #4.
C80	Assembly 4 Counter	Counter to keep track of number of pallets that passed the Assembly Station #4.
MW24	Pallet Count At Asembly4	Memory variable to store the value of the counter C80.

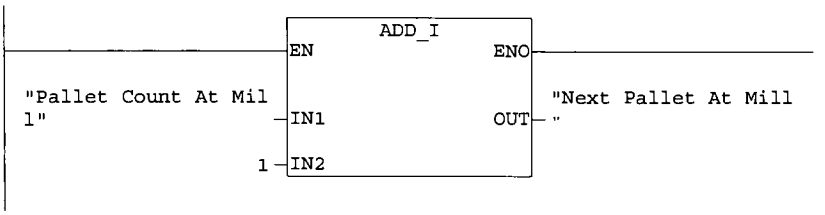
Network: 28

The pallet count value for the Assembly Station #4, stored in MW24 in the previous network is compared with the variable MW256 that contains the value for number of pallets in the system. When the value of these two variables is equal, an output Q2.3 turns high. This output is used to reset the counter C80 to zero in the previous network.



Network: 32 Next Pallet At the Mill Station.

In this network, we add 1 to the pallet count at the Mill (MW 14) to get the next pallet number at the Mill (MW 140)

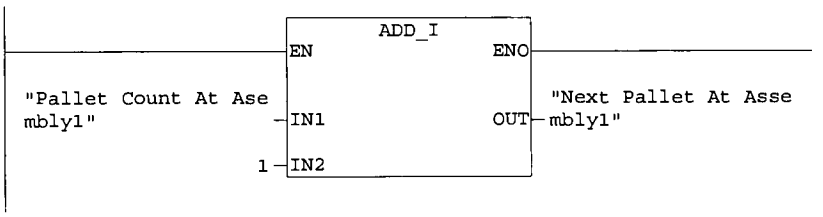


Symbol information

MW14	Pallet Count At Mill	Memory variable to store the value of the counter C30.
MW140	Next Pallet At Mill	Variable to store the value of the next pallet arriving at the Mi ll Station.

Network: 33 Next Pallet At the Assembly Station #1.

In this network, we add 1 to the pallet count at the Assembly #1 (MW 18) to get the next pallet number at the Assembly #1 (MW 180)

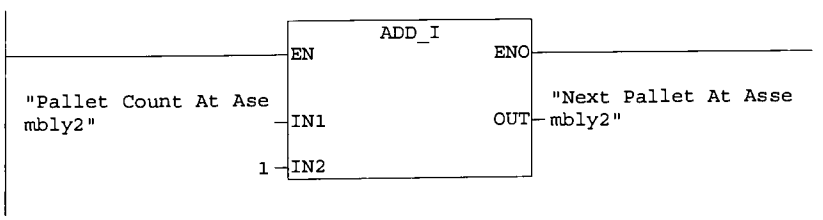


Symbol information

MW18	Pallet Count At Asembly1	Memory variable to store the value of the counter C50.
MW180	Next Pallet At Assembly1	Variable to store the value of the next pallet arriving at th e Assm Station #1.

Network: 34 Next Pallet At the Assembly Station #2.

In this network, we add 1 to the pallet count at the Assembly #2 (MW 20) to get the next pallet number at the Assembly #2 (MW 200)

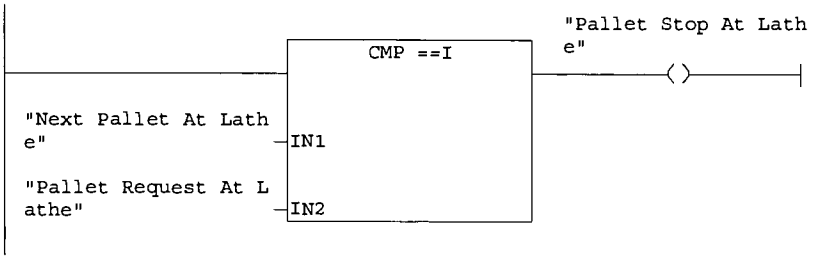


Symbol information

MW20	Pallet Count At Asembly2	Memory variable to store the value of the counter C60.
MW200	Next Pallet At Assembly2	Variable to store the value of the next pallet arriving at th e Assm Station #2.

Network: 38	Memory Bit To Raise Latch At the Lathe Station.

In this network, we compare the values of the Next Pallet At Lathe (MW 120) with the Pallet Request at Lathe (MW 42) to turn a Memory Bit (M1.0) high. This memory bit, in turn raises the latch at the Lathe Station.

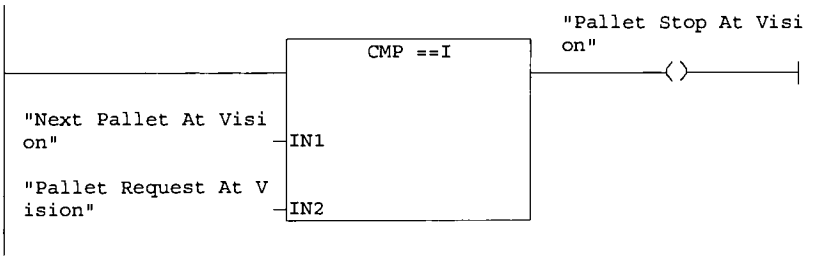


Symbol information

MW120	Next Pallet At Lathe	Variable to store the value of the next pallet arriving at the Lathe Station.
MW42	Pallet Request At Lathe	Variable to store value of pallet # requested to be stopped at Lathe Station.
M1.0	Pallet Stop At Lathe	Memory Bit to raise the latch at the Lathe Station.

Network: 39	Memory Bit To Raise Latch At the Vision Station.

In this network, we compare the values of the Next Pallet At Vision (MW 160) with the Pallet Request at Vision (MW 44) to turn a Memory Bit (M3.0) high.This memory bit, in turn raises the latch at the Vision Station.

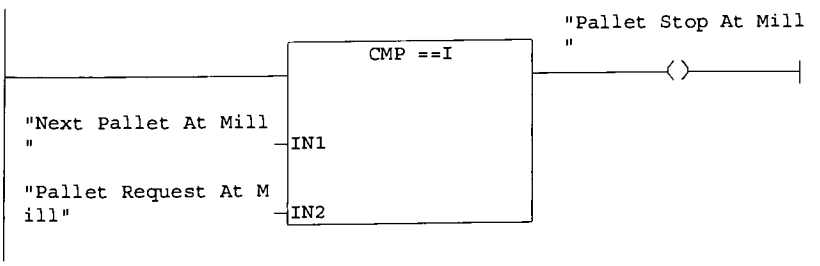


Symbol information

MW160	Next Pallet At Vision	Variable to store the value of the next pallet arriving at the Vision Station.
MW44	Pallet Request At Vision	Variable to store value of pallet # requested to be stopped at Vision Station.
M3.0	Pallet Stop At Vision	Memory Bit to raise the latch at the Vision Station.

Network: 40	Memory Bit To Raise Latch At the Mill Station.

In this network, we compare the values of the Next Pallet At Mill (MW 140) with the Pallet Request at Mill (MW 52) to turn a Memory Bit (M2.0) high.This memory bit, in turn raises the latch at the Mill Station.

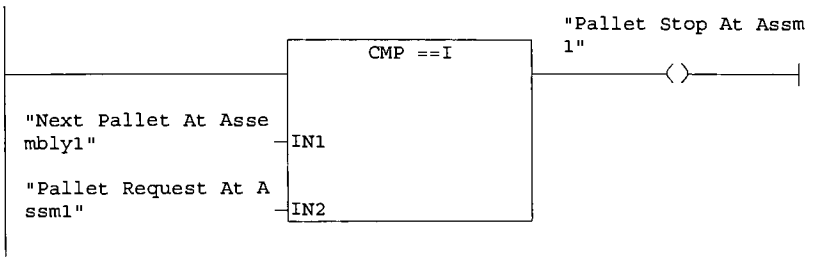


Symbol information

MW140	Next Pallet At Mill	Variable to store the value of the next pallet arriving at the Mill Station.
MW52	Pallet Request At Mill	Variable to store value of pallet # requested to be stopped at Mill Station.
M2.0	Pallet Stop At Mill	Memory Bit to raise the latch at the Mill Station.

Network: 41 Memory Bit To Raise Latch At the Assembly Station #1.

In this network, we compare the values of the Next Pallet At Assembly Station #1 (MW 180) with the Pallet Request at Assembly Station #1 (MW 46) to turn a Memory Bit (M5.0)high. This memory bit, in turn raises the latch at the Assembly Station #1.

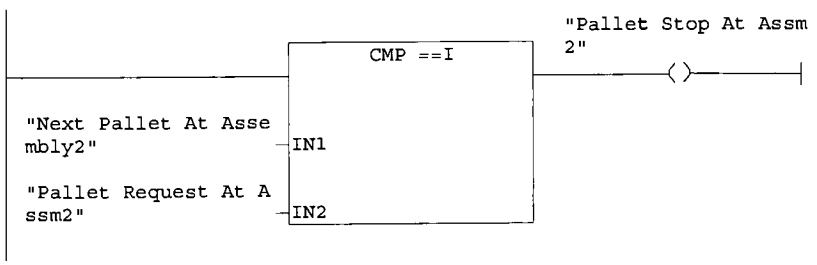


Symbol information

MW180	Next Pallet At Assembly1	Variable to store the value of the next pallet arriving at the Assm Station #1.
MW46	Pallet Request At Assm1	Variable to store value of pallet # requested to be stopped at Assm Station #1.
M5.0	Pallet Stop At Assm1	Memory Bit to raise the latch at the Assembly Station #1.

Network: 42 Memory Bit To Raise Latch At the Assembly Station #2.

In this network, we compare the values of the Next Pallet At Assembly Station #2 (MW 200) with the Pallet Request at Assembly Station #2 (MW 56) to turn a Memory Bit (M6.0) high. This memory bit, in turn raises the latch at the Assembly Station #2.

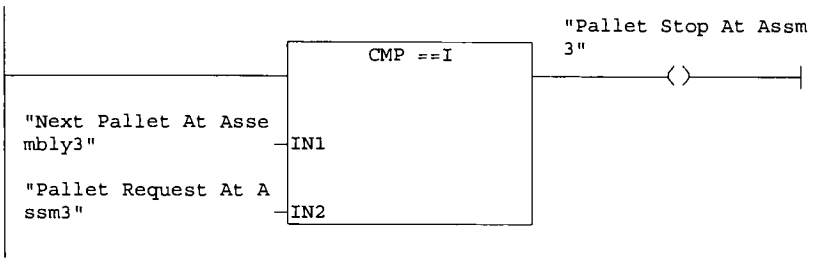


Symbol information

MW200	Next Pallet At Assembly2	Variable to store the value of the next pallet arriving at the Assm Station #2.
MW56	Pallet Request At Assm2	Variable to store value of pallet # requested to be stopped at Assm Station #2.
M6.0	Pallet Stop At Assm2	Memory Bit to raise the latch at the Assembly Station #2.

Network: 43	Memory Bit To Raise Latch At the Assembly Station #3.

In this network, we compare the values of the Next Pallet At Assembly Station #3 (MW 220) with the Pallet Request at Assembly Station #3 (MW 48) to turn a Memory Bit (M7.0) high. This memory bit, in turn raises the latch at the Assembly Station #3.

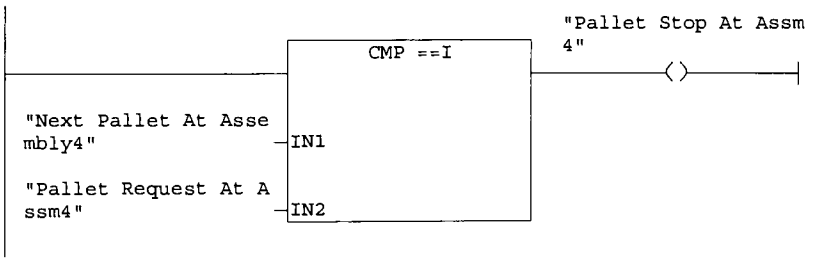


Symbol information

MW220	Next Pallet At Assembly3	Variable to store the value of the next pallet arriving at the Assm Station #3.
MW48	Pallet Request At Assm3	Variable to store value of pallet # requested to be stopped at Assm Station #3.
M7.0	Pallet Stop At Assm3	Memory Bit to raise the latch at the Assembly Station #3.

Network: 44	Memory Bit To Raise Latch At the Assembly Station #4.

In this network, we compare the values of the Next Pallet At Assembly Station #4 (MW 240) with the Pallet Request at Assembly Station #4 (MW 58) to turn a Memory Bit (M8.0)high. This memory bit, in turn raises the latch at the Assembly Station #4.



Symbol information

MW240	Next Pallet At Assembly4	Variable to store the value of the next pallet arriving at the Assm Station #4.
MW58	Pallet Request At Assm4	Variable to store value of pallet # requested to be stopped at Assm Station #3.
M8.0	Pallet Stop At Assm4	Memory Bit to raise the latch at the Assembly Station #4.

Computer		
IMERT28	Computer Type	Server

Tag Management		
@CurrentUser	Data Type	Text 8-Bit
	Parameters	internal tags
S7$Program(1)#RawArchiv	Data Type	Raw Data
	Parameters	RAW_ARCHIVE
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$at$Lathe	Data Type	Bit value
	Parameters	I2.2
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$CountAtAsembly1	Data Type	16-Bit signed
	Parameters	MW18
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$CountAtAsembly3	Data Type	16-Bit signed
	Parameters	MW22
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$CountAtDock	Data Type	16-Bit signed
	Parameters	MW10
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$CountAtMill	Data Type	16-Bit signed
	Parameters	MW14
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Active_Station	Data Type	16-Bit unsigned
	Parameters	MW26
	Start value	0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Start$Latch$Bit	Data Type	Bit value
	Parameters	M0.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Stop$Latch$Bit	Data Type	Bit value
	Parameters	M0.1
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Request$Asembly2Output	Data Type	Bit value
	Parameters	Q1.2
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Request$Asembly4Output	Data Type	Bit value
	Parameters	Q0.1

Tag Management		
	Parameters	MW54
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$RequestAtLathe	Data Type	16-Bit signed
	Parameters	MW42
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$RequestAtMill	Data Type	16-Bit signed
	Parameters	MW52
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)#RawEvent	Data Type	Raw Data
	Parameters	RAW_EVENT
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$at$Dock	Data Type	Bit value
	Parameters	I2.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$at$Mill	Data Type	Bit value
	Parameters	I2.3
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$CountAtAsembly2	Data Type	16-Bit signed
	Parameters	MW20
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$CountAtAsembly4	Data Type	16-Bit signed
	Parameters	MW24
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$CountAtLathe	Data Type	16-Bit signed
	Parameters	MW12
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$CountAtVision	Data Type	16-Bit signed
	Parameters	MW16
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Start$Button	Data Type	Bit value
	Parameters	I1.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Stop$Button	Data Type	Bit value
	Parameters	I1.1
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI

Tag Management		
S7$Program(1)/Pallet$at$Vision	Data Type	Bit value
	Parameters	I2.4
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Pallet_1	Data Type	16-Bit unsigned
	Parameters	MW28
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Part_Name_Lathe	Data Type	Text 8-Bit
	Parameters	MW60
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Part_Name_Vision	Data Type	Text 8-Bit
	Parameters	MW76
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$RequestAtAssm1	Data Type	16-Bit signed
	Parameters	MW46
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$RequestAtVision	Data Type	16-Bit signed
	Parameters	MW44
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$StopAtAssm1	Data Type	Bit value
	Parameters	M5.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$StopAtAssm2	Data Type	Bit value
	Parameters	M6.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$StopAtAssm3	Data Type	Bit value
	Parameters	M7.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$StopAtAssm4	Data Type	Bit value
	Parameters	M8.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$StopAtDock	Data Type	Bit value
	Parameters	M4.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$StopAtLathe	Data Type	Bit value
	Parameters	M1.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite

Tag Management		
	Channel unit	MPI
S7$Program(1)/Pallet$StopAtMill	Data Type	Bit value
	Parameters	M2.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Pallet$StopAtVision	Data Type	Bit value
	Parameters	M3.0
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Next$PalletAtAssembly1	Data Type	16-Bit signed
	Parameters	MW180
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Next$PalletAtAssembly2	Data Type	16-Bit signed
	Parameters	MW200
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Next$PalletAtAssembly3	Data Type	16-Bit signed
	Parameters	MW220
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Next$PalletAtAssembly4	Data Type	16-Bit signed
	Parameters	MW240
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Next$PalletAtDock	Data Type	16-Bit signed
	Parameters	MW100
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Next$PalletAtLathe	Data Type	16-Bit signed
	Parameters	MW120
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Next$PalletAtMill	Data Type	16-Bit signed
	Parameters	MW140
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
S7$Program(1)/Next$PalletAtVision	Data Type	16-Bit signed
	Parameters	MW160
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Part_Name_Mill	Data Type	Text 8-Bit
	Parameters	MW68
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Part_Name_Dock	Data Type	Text 8-Bit
	Parameters	MW84
	Connection	S7$Program(1)

Tag Management		
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Part_Name_Assm1	Data Type	Text 8-Bit
	Parameters	MW92
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Part_Name_Assm2	Data Type	Text 8-Bit
	Parameters	MW104
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Part_Name_Assm3	Data Type	Text 8-Bit
	Parameters	MW112
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Part_Name_Assm4	Data Type	Text 8-Bit
	Parameters	MW124
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Lathe_Busy	Data Type	Bit value
	Parameters	M1.4
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Mill_busy	Data Type	Bit value
	Parameters	M2.4
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Vision_Busy	Data Type	Bit value
	Parameters	M3.4
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Dock_Busy	Data Type	Bit value
	Parameters	M4.4
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Assm1_Busy	Data Type	Bit value
	Parameters	M5.4
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Assm2_Busy	Data Type	Bit value
	Parameters	M6.4
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Assm3_Busy	Data Type	Bit value
	Parameters	M7.4
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
Assm4_Busy	Data Type	Bit value
	Parameters	M8.4

Tag Management		
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI
@TLGRT_SIZEOF_DATABASE_QUEUE	Data Type	64-Bit IEEE 754
	Group	TagLoggingRt
	Parameters	internal tags
@TLGRT_SIZEOF_NOTIFY_QUEUE	Data Type	64-Bit IEEE 754
	Group	TagLoggingRt
	Parameters	internal tags
@TLGRT_SIZEOF_DBASE_QUEUE	Data Type	64-Bit IEEE 754
	Group	TagLoggingRt
	Parameters	internal tags
@TLGRT_SIZEOF_NLL_INPUT_QUEUE	Data Type	64-Bit IEEE 754
	Group	TagLoggingRt
	Parameters	internal tags
S7$Program(1)/prakash	Data Type	16-Bit signed
	Parameters	MW210
	Connection	S7$Program(1)
	Channel	SIMATIC S7 Protocol Suite
	Channel unit	MPI

Connections		
S7$Program(1)	Unit	MPI
	Parameters	MPI,2 0,,0,2,02

```
    ;;; Station 27(Assembly 4) input 22 to output 31
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    store reg#2045 to reg#1012
    store reg#2016 to reg#1014
    store reg#2044 to reg#1015
    store reg#2043 to reg#1016
    store reg#2026 to reg#1018
    store reg#2028 to reg#1019
    store reg#2027 to reg#1030
    store reg#2022 to reg#1031
    goto Step_Pass_Thru_Inputs

[10] Step_SGate
    ;;; This is the first step in the loop for the south lift.
    ;;; Here, we raise the latch at station S7.
    ;;; Next, we monitor the sensor at station S7 and then
    ;;; go to the next step.
    ------------------------------------------------------------------------
--------
    Latch_S7
    ------------------------------------------------------------------------
--------
    monitor Sensor_At_Station_S7 goto Step_To_Check_If_Ready

[11] Step_To_Check_If_Ready
    ;;; Here, we check the flag at the south lift for the
    ;;; clear condition. When the flag turns clear, we go to
    ;;; the next step.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    monitor Flag_South_Lift_Busy:clear goto Step_To_Release_Pallette

[12] Step_To_Release_Pallette
    ;;; Here, we lower the latch at station S7, permitting the
    ;;; pallet to enter the south lift,wait for 200ms
    ;;; and then go to the next step.
    ------------------------------------------------------------------------
--------
    Release_S7
    ------------------------------------------------------------------------
--------
    delay 200 ms goto Step_SLift

[13] Step_SLift
    ;;; Here, we raise the latch at station S7 to lock the
    ;;; proceeding pallets and then monitor the sensor at
    ;;; station S3 to turn high.
    ------------------------------------------------------------------------
--------
    Latch_S7
```

```
    ------------------------------------------------------------------------
--------
    monitor Sensor_At_Station_S3 goto Step_Raise_S3_S4

[14] Step_Raise_S3_S4
    ;;; Here we raise the south lift, thus transferring the
    ;;; pallet. We also set the flag for the south lift to busy
    ;;; so that proceeding pallets do not enter the lift.
    ;;; Next, we monitor the sensor at station S4 to ensure
    ;;; that the pallet is transferred and then go to next
    ;;; step.
    ------------------------------------------------------------------------
--------
    Raise_S3_S4
    ------------------------------------------------------------------------
--------
    set Flag_South_Lift_Busy
    monitor Sensor_At_Station_S4 goto Step_Lower_S3_S4

[15] Step_Lower_S3_S4
    ;;; Here, we lower the south lift, wait for 2 seconds and
    ;;; then go to the next step.
    ------------------------------------------------------------------------
--------
    Lower_S3_S4
    ------------------------------------------------------------------------
--------
    delay 2 sec goto Step_Klear_Flag

[16] Step_Klear_Flag
    ;;; Here, we clear the flag for the south lift so that the
    ;;; proceeding pallets can enter the lift.
    ;;; This is the last step in this loop. At the end of this
    ;;; loop we go back to the start of this loop at line #10.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    clear Flag_South_Lift_Busy
    goto Step_SGate

[30] Step_NGate
    ;;; This is the first step in the loop for the North Lift.
    ;;; Here, we raise the latch at station S8 and then monitor
    ;;; for the sensor at station S8 to turn high and then go
    ;;; to the next step.
    ------------------------------------------------------------------------
--------
    Latch_S8
    ------------------------------------------------------------------------
--------
    monitor Sensor_At_Station_S8 goto Step_Check_North_Lift_Ready

[31] Step_Check_North_Lift_Ready
    ;;; Here, we monitor the flag for the north lift for clear
    ;;; condition and then go to the next step, when the flag
```

```
    ;;; for north lift is clear.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    monitor Flag_North_Lift_Busy:clear goto Step_Release_North_Gate

[32] Step_Release_North_Gate
    ;;; Here, we lower the latch at station S8, thus permitting
    ;;; the pallet to enter the north lift and then monitor
    ;;; for the sensor at station S1 to turn High, after which
    ;;; we go to the next step.
    ------------------------------------------------------------------------
--------
    Release_S8
    ------------------------------------------------------------------------
--------
    delay 500 ms goto Step_NLift

[33] Step_NLift
    ;;; Once the pallet enters the north lift, we raise the latch
    ;;; at station S8 to prevent the proceeding pallets from
    ;;; entering the lift.
    ------------------------------------------------------------------------
--------
    Latch_S8
    ------------------------------------------------------------------------
--------
    monitor Sensor_At_Station_S1 goto Step_Raise_S1_S2

[34] Step_Raise_S1_S2
    ;;; Here we raise the north lift, thus transferring the
    ;;; pallet. We also set the flag for the north lift to busy
    ;;; so that proceeding pallets do not enter the lift.
    ;;; Next, we monitor the sensor at station S2 to ensure
    ;;; that the pallet is transferred and then go to next
    ;;; step.
    ------------------------------------------------------------------------
--------
    Raise_S1_S2
    ------------------------------------------------------------------------
--------
    set Flag_North_Lift_Busy
    monitor Sensor_At_Station_S2 goto Step_Delay

[35] Step_Delay
    ;;; We wait for 700 ms to ensure that the pallet has cleared
    ;;; the north lift and at the end of 700 ms we go to the
    ;;; next step.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    delay 700 ms goto Step_Lower_S1_S2
```

```
[36] Step_Lower_S1_S2
    ;;; Once the pallet has cleared the north lift, we lower
    ;;; the north lift and then we clear the flag for the
    ;;; north lift thus permitting the proceeding
    ;;; pallets to enter the north lift and then go to the
    ;;; next step.
    ;;; This is the last step in this loop, at the end of which
    ;;; we go back to start of this loop at line #30.
    ------------------------------------------------------------------------
--------
    Lower_S1_S2
    ------------------------------------------------------------------------
--------
    clear Flag_North_Lift_Busy
    goto Step_NGate

[50] Step_WGate
    ;;; This is the first step in the loop for the west Lift.
    ;;; Here, we raise the latch at station S22 and then monitor
    ;;; for the sensor at station S22 to turn high and then go
    ;;; to the next step.
    ------------------------------------------------------------------------
--------
    Latch_S22
    ------------------------------------------------------------------------
--------
    monitor Sensor_At_Station_S22 goto Step_Check_West_Lift_Ready

[51] Step_Check_West_Lift_Ready
    ;;; Here, we monitor the flag for the west lift for clear
    ;;; condition and then go to the next step, when the flag
    ;;; for west lift is clear.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    monitor Flag_West_Lift_Busy:clear goto Step_Release_West_Gate

[52] Step_Release_West_Gate
    ;;; Here, we lower the latch at station S22, thus permitting
    ;;; the pallet to enter the north lift and go to the next step.
    ------------------------------------------------------------------------
--------
    Release_S22
    ------------------------------------------------------------------------
--------
    delay 400 ms goto Step_WLift

[53] Step_WLift
    ;;; Once the pallet enters the north lift, we raise the latch
    ;;; at station S22 to prevent the proceeding pallets from
    ;;; entering the lift, and then monitor the sensor at
    ;;; station S21 to turn high, after which we go to the next
    ;;; step.
    ------------------------------------------------------------------------
--------
```

```
    Latch_S22
    ------------------------------------------------------------------------
--------
    monitor Sensor_At_Station_S21 goto Step_Raise_S21_S20

[54] Step_Raise_S21_S20
    ;;; Here we raise the west lift, thus transferring the
    ;;; pallet. We also set the flag for the west lift to busy
    ;;; so that proceeding pallets do not enter the lift.
    ;;; Next, we monitor the sensor at station S20 to ensure
    ;;; that the pallet is transferred and then go to next
    ;;; step.
    ------------------------------------------------------------------------
--------
    Raise_S20_S21
    ------------------------------------------------------------------------
--------
    set Flag_West_Lift_Busy
    monitor Sensor_At_Station_S20 goto Step_Lower_S21_S20

[55] Step_Lower_S21_S20
    ;;; Once the pallet is transferred we lower the west lift
    ;;; and then go to the next step.
    ------------------------------------------------------------------------
--------
    Lower_S20_S21
    ------------------------------------------------------------------------
--------
    delay 200 ms goto Step_Clear_Flag

[56] Step_Clear_Flag
    ;;; Once the pallet has been transferred,we clear the
    ;;; flag for the west lift thus permitting the proceeding
    ;;; pallets to enter the west lift and then go to the
    ;;; next step.
    ;;; This is the last step in this loop, at the end of which
    ;;; we go back to start of this loop at line #50.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    clear Flag_West_Lift_Busy
    goto Step_WGate

[70] Step_WEGate
    ;;; This is the first step in the loop for the west-to-south
    ;;; lift.
    ;;; Here we raise the latch at station S25, and then monitor
    ;;; the sensor at station S25 to turn high, after which
    ;;; we go to the next step.
    ------------------------------------------------------------------------
--------
    Latch_S25
    ------------------------------------------------------------------------
--------
    monitor Sensor_At_Station_S25 goto Step_Check_W_E_Lift_Ready
```

```
[71] Step_Check_W_E_Lift_Ready
    ;;; Here, we monitor the flag for the west-to-south lift
    ;;; for clear condition and then go to the next step, when
    ;;; the flag for west-to-south lift is clear.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    monitor Flag_West_To_South_Lift_Busy:clear goto
Step_Release_W_E_Gate

[72] Step_Release_W_E_Gate
    ;;; Here, we release the latch at station S25 to permit the
    ;;; pallet to enter the lift, and at the same time, raise
    ;;; the west-to-south lift. Next, we set the flag for the
    ;;; west-to-south lift to busy condition to prevent the
    ;;; proceeding pallets to enter the lift.
    ------------------------------------------------------------------------
--------
    Release_S25
    Raise_S28
    ------------------------------------------------------------------------
--------
    set Flag_West_To_South_Lift_Busy
    delay 200 ms goto Step_WELift

[73] Step_WELift
    ;;; Here we raise the latch at station S25 to prevent the
    ;;; proceeding pallets from entering into the lift.
    ------------------------------------------------------------------------
--------
    Latch_S25
    ------------------------------------------------------------------------
--------
    goto Step_Monitor_S28

[74] Step_Monitor_S28
    ;;; Here, we monitor the sensor at station S28 to ensure that
    ;;; the pallet has entered the lift.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    monitor Sensor_At_Station_S28 goto Step_Lower_S28

[75] Step_Lower_S28
    ;;; Once the pallet has entered the lift,we lower the lift
    ;;; and then monitor the sensor at station S31 to ensure
    ;;; that the pallet has cleared the lift.
    ------------------------------------------------------------------------
--------
    Lower_S28
    ------------------------------------------------------------------------
--------
```

```
    monitor Sensor_At_Station_S31 goto Step_Clr_Flag

[76] Step_Clr_Flag
    ;;; Once the pallet has cleared the lift, we clear the flag
    ;;; for the west-to-east lift to permit proceeding pallets
    ;;; to enter the lift.
    ;;; This is the last step in this loop, at the end of which
    ;;; we go to the first step in the loop at line #70.
    ------------------------------------------------------------------------
--------
    <NO CHANGE IN DIGITAL OUTPUTS>
    ------------------------------------------------------------------------
--------
    clear Flag_West_To_South_Lift_Busy
    goto Step_WEGate
```