

BRIDGEVIEW™ FOR HYBRID CONTROL SYSTEM:
AN APPLIED CASE STUDY OF AUTOMATING
AN INDUSTRIAL ASSEMBLY LINE

By

SYED M.M. MANZOOR

Bachelor of Electronic Engineering

NED University

Karachi, Pakistan

1992

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July 1998

OKLAHOMA STATE UNIVERSITY

BRIDGEVIEW™ FOR HYBRID CONTROL SYSTEM:

AN APPLIED CASE STUDY OF AUTOMATING

AN INDUSTRIAL ASSEMBLY LINE

Thesis Approved:

W. Dick Stout

Thesis Advisor

J. Chandler

John Hall

Wayne B. Powell

Dean of the Graduate College

PREFACE

Advancement in computer, sensors and controllers technology requires control engineers to deal with larger and more complicated control systems. It is no longer sufficient to choose between a discrete or analog system because both types are often needed in today's sophisticated systems. As discrete and analog control systems are both extending rapidly to accommodate crucial properties of each other, the line of demarcation between them is becoming fuzzy. A new type of control system – Hybrid Control System (HCS) – is emerging that combines the benefits of both. HCS's synergistic effect creates a control system that encompasses the features and functionality of each type and beyond.

This work describes a case study of such a system. It involves design and implementation of supervisory software for a propriety Hybrid Control System project involving a robotic assembly line for a cellular relay-station motherboard manufacturer. The scope of the project included system analysis of the robotic assembly line, selection of networking and communication protocols, database selection, and the design of an intuitive GUI or Man Machine Interface (MMI). Commercially available Supervisory Control and Data Acquisition (SCADA) packages were reviewed to select a suitable environment. The software implementation on one station of the line is shown as an example.

ACKNOWLEDGEMENTS

First and foremost, my family (Father, Syed M. Usman, Mother, Anis Fatima, brothers, and sisters) deserves credit for providing me the love and support throughout my life, especially during my graduate studies.

I am very thankful to my advisor, Dr. Nick Street, for providing me guidance, exercising patience, and extending encouragement on numerous occasions, like a friend more than an advisor. Dr. John Chandler and Dr. John Hatcliff also deserve credit for serving as thesis committee members and facilitating the thesis process despite of my being on the field a thousand mile away.

My special thanks goes to Spencer Sullivan (District Manager, VI Engineering) for providing me the opportunity to work on this exciting thesis project. His eternal optimism and trust in people make him one of the best managers I have encountered in my life. I learned a great deal working with him and his team of engineers. And of course, this thesis would not have been possible without the understanding and full support of Robert Jacobs (President, VI Engineering).

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
1.1 Abstract.....	1
1.2 Introduction.....	1
II. HYBRID CONTROL SYSTEM.....	4
2.1 Introduction to Control Systems.....	4
2.2 Robot Assembly Line Operation as an Example of Hybrid Control System.....	7
2.3 Hybrid Control System Architecture.....	8
2.4 Formal Definition of Hybrid Control System Automata.....	10
2.5 Hybrid Control System Advantages.....	12
III. SUPERVISORY CONTROL AND DATA ACQUISITION.....	13
3.1 What is SCADA.....	13
3.2 Where is it suitable?.....	13
3.3 Main Features of a SCADA System.....	14
1) Data Acquisition And Supervisory Control.....	14
2) Data Presentation.....	16
3) Network and Security.....	17
4) Database Connectivity.....	17
3.4 Overview of some widely used SCADA Packages.....	17
Cimplicity.....	18
InTouch.....	20
BridgeVIEW.....	22

IV.	TRX ASSEMBLY LINE PROJECT.....	27
	4.1 Project Overview.....	28
	4.2 System Analysis.....	31
	4.2.1 TRX Assembly Line Project Management.....	31
	a) Variables List.....	33
	b) Protocols.....	33
	Automatic and Fixes Board Mode Protocol.....	33
	Purge Protocol.....	35
	Board Re-entry Protocol.....	36
	Benchmark Test for Database.....	36
	Questionnaires.....	38
	c) Eventflow Diagrams.....	38
	4.2.2 Supervisory System Hardware.....	41
	4.2.3 Supervisory Software Architecture.....	41
	a) Hybrid Supervisory Control Software.....	42
	Graphical User Interface (GUI).....	42
	Handshaking.....	45
	Database.....	46
	1) Product Database.....	46
	Product Database Archiving.....	49
	2) Historian Database.....	49
	Communication.....	53
	b) Supervisory Software Algorithm.....	53
	c) SCADA Package Selection.....	54
V.	DESCRIPTION OF STATION 1.....	56
	5.1 Protocol Description through Eventflow Diagrams.....	56
	5.2 Supervisory Software Hierarchy.....	62
	1) GUI.....	62
	2) Handshaking.....	64
	Synchronization (Critical Timing) Issues.....	67

3) Database.....	69
4) Communication.....	71
5.3 Station 1 Block Diagrams.....	73
VI. SUMMARY OF OBSERVATION.....	86
VII. REFERENCES.....	89

LIST OF FIGURES

Figures	Page
Figure 2-1 Robot assembly line.....	8
Figure 2-2 Hybrid Control System Architecture.....	9
Figure 2-3 Hybrid Control System Automaton.....	11
Figure 3-1 An outline of a typical SCADA system.....	15
Figure 4-1 The TRX assembly line.....	29
Figure 4-2 A part of the list of the variables.....	34
Figure 4-3 A sample of Fix board mode protocol.....	35
Figure 4-4 A sample of Purge mode protocol.....	37
Figure 4-5 Table of Board reentry Flag definitions.....	37
Figure 4-6 Results of the benchmark tests on three databases using three front-ends...38	
Figure 4-7 A sample Questionnaire.....	39
Figure 4-8 A part of the eventflow diagram of one station.....	40
Figure 4-9a The "Main" GUI command center.....	43
Figure 4-9b The "Overview" GUI.....	44
Figure 4-10 A part of the product Database functionality requirements.....	47-48
Figure 4-11 Example of Database Field definitions.....	50
Figure 4-12 Access™ Database archiving procedure.....	51
Figure 4-13 Historian Database requirements.....	52
Figure 4-14 Networking and Data sharing Scheme.....	54
Figure 5-1a Station 1 Eventflow diagram.....	57
Figure 5-1b Station 1 Eventflow diagram.....	59
Figure 5-1c Station 1 Eventflow diagram.....	61
Figure 5-2 Station 1 GUI.....	63
Figure 5-3 Station 1 Automatic and Fixed Board Mode protocol.....	65
Figure 5-4 Station 1 Purge protocol.....	66

Figure 5-5	Station 1 Board reentry protocol.....	68
Figure 5-6	Product Database entries for Station 1.....	70
Figure 5-7	A part of BridgeVIEW engine configuration file.....	72
Figure 5-8	Station 1 Hierarchy.....	74
Figure 5-9	Station 1 Readin State Machine Icon and Block Diagram.....	75
Figure 5-10	States in Readin State Machine.....	76-78
Figure 5-11	Icon and Block diagram of the subprogram in state "Readin Board Info from Database".....	78
Figure 5-12	Icon and Block diagram of the subprogram that queries the database in program Readin State Machine.....	79
Figure 5-13	Icon and Block diagram of the subprogram that determines board status in Board Reentry protocol in program Readin State Machine.....	80
Figure 5-14	Icon and Block diagram of the subprogram that writes to Adept Controller 1 in Readin State Machine.....	81
Figure 5-15	Icon and Block diagram of the subprogram that reads from Adept Controller 1 in Readot State Machine.....	81
Figure 5-16	Station 1 Readot State Machine Icon and Block Diagram.....	82
Figure 5-17	Icon and Block diagram of the subprogram that updates the database in program Readot State Machine.....	83
Figure 5-18	Station 1 Purge State Machine Icon and Block Diagram.....	84
Figure 5-19	Icon and Block diagram of the subprogram that updates the database in program Purge State Machine.....	85

CHAPTER I

INTRODUCTION

1.1 Abstract

This thesis describes the supervisory software design and program for a hybrid control system industrial assembly line involving three robots controlling nine stations. It enables the software engineers to develop a hierarchical structure for integrating GUI, database, handshaking and network protocols in a single program. Its one-program-for-the-whole-system approach allows users to monitor the effect of a change in a process parameter of one robot over the whole assembly line. Moreover it stores and displays the product, process and statistical data for better optimization of the assembly line. The supervisory software is written in BridgeVIEW™ which utilizes the object oriented graphical programming environment for the development of the code.

1.2 Introduction

Automation is recognized as an integral part of the industrial world. Though its benefits extend to many areas such as office automation, it is most extensively used in the manufacturing industry for its indispensable benefits to areas such as production, information management, and safety features. It not only provides better data acquisition and control systems but also enhances the coordination between different instruments, thus reducing the need for manual intervention in batch processes. The net results are increased productivity, safer operations, and consequently, reduced cost of production.

Traditionally, two distinct types of control systems are employed in industry. Distributed Control Systems (DCS) are used for continuous control systems. DCS's are microprocessor-based replacements for the panel board controllers and recorders. DCS systems are extensively used, for example, in refineries and petrochemical plants. Programmable Logic Controllers (PLC), on the other hand, are used for discrete systems mostly involving On/Off or digital controls. PLC's are also microprocessor-based replacements for hardwired relays and mechanical timers.

Advancement in computer, sensors and controllers technology required control engineers to deal with larger and more complicated control systems. It was no longer sufficient to choose between a discrete or analog system because both types were often needed. Gradually, both continuous and discrete control systems were extended to accommodate crucial properties of each other [La Fauci 97], and the line of demarcation between them became fuzzy. A new type of control system – Hybrid Control System (HCS) – emerged that combines both Distributed Control Systems and Programmable Logic Controllers. The synergistic effect creates a control system that encompasses the features and functionality of each type and beyond.

The overlapping of continuous and discrete control system boundaries with one another increased the complexity of the resulting control systems. The combined controllers involve not only the complexity of continuous and discrete controllers but also the added complexity of mutual interaction. The increased complexity required the use of higher level (preferably graphic) languages. Desktop computers' role rapidly increased in automation and control systems because of their ability to run higher level (especially

visual) languages and capabilities such as networking and databases. The major hurdle in their use – reliability – was significantly reduced with the implementation of parallel architectures, multitasking and multiprocessing operating systems.

Canned commercial software is now available for desktop computers requiring very little programming; just configuring the I/O's and logic using built-in functions provided rapid development and reliability. However, extensive training is required to configure the system, and unique instrumentation and modification requires professional help from the manufacturers of the software package.

High level automation languages are also becoming popular. They are more flexible than canned packages and are relatively easy to use, especially if they have a graphical environment. An example is "iconic programming" in which functions are depicted as icons. Icons are copied from a built-in library, arranged, and wires drawn between them to show data flow path and sequence. This kind of programming is gaining popularity as it frees the programmer from mastering a text-based language.

How effective and suitable are these high level graphical languages for Hybrid Control Systems and what are their cons and pros? This thesis addresses this question with a case study using a popular graphical language, BridgeVIEW [National Instruments Corp 96], to complete a proprietary hybrid control project involving a robotic assembly line for a cellular relay station motherboard manufacturer

CHAPTER II

HYBRID CONTROL SYSTEMS

2.1 Introduction to Control Systems

The history of automation is as old as the history of human civilization. Right from the making of the first tool on earth there has been a constant trend towards automation. The first efforts for automation utilized stones and wood, followed by mechanical automation. With the introduction of electricity, mechanical parts were replaced by electrical parts. Today, there is a large number of automatic electrical controls in homes, businesses and industries.

Automation is the act of controlling processes with minimal human involvement. A typical control has the ability to start and maintain the process variables within their specified ranges, and eventually, stop the process when needed. Thus an automatic control system replaces the human input required for the process control.

A control system for any process can be designed by knowing the inputs, control actions, and outputs. Physical or chemical inputs to a process are generally transformed to electrical signals using transducers [Warnock 88]. Transducers generate an electrical signal proportional to the applied physical or chemical stimulus. These signals can be discrete (High/Low) or analog (continuous). For example, an On/Off switch produces a discrete signal while a thermocouple produces an analog signal (millivolts).

There are two major classes of electrical control systems. "Continuous control systems" involve addition, subtraction, differentiation and integration of the continuously incoming

signals (current/voltages) and provide correct output in real time. "Discrete control systems" output High/Low signals based on binary inputs (0 or 1) and Boolean control logic. Examples are "Proportional, Integral and Derivative" (PID) controllers for continuous control systems and "Programmable Logic Controllers" (PLC) for discrete control systems.

Initially, all controllers were hardwired and were applicable only for the process for which they were designed. These controllers would lose reliability if any design parameter had to be changed in the process. Integrated circuits gave software/control engineers the flexibility to use software routines for developing the logic for these controllers. A controller can now be used to handle different processes just by changing the software code. Controllers are now more robust, modular and portable.

Control system reliability is a primary concern for control engineers because of significant productivity (and financial) loss incurred by breakdowns. Whereas controllers for both discrete and continuous control systems were specifically designed for reliable operation as standalone independent units, desktop computers lagged behind in reliability, partly because their operating system was not designed with automation in mind. Operating systems of the early computers encountered frequent freeze-ups, usually because of viruses, low resource allocations, process disruptions, etc.

With the implementation of parallel architectures, multitasking and multiprocessing operating systems, desktop computers became more reliable. A problem in a single process does not freeze up the computer anymore, other processes continue to run, thus making computers reliable enough for uninterrupted control system operations. With the

increasing reliability of desktop computers, their role rapidly increased in automation because of their high potential for augmenting control system flexibility. Computers add capabilities such as networking, database, and ability to run higher level (especially visual) languages.

The advancement in computer, sensor and controller technology paved the way for the control engineers to deal with larger and more complicated control systems. It was no longer sufficient to choose between a discrete or analog system because both types were often needed. Thus, research was conducted in areas previously thought difficult to deal with [Lygeros 98]. As a result, both continuous and discrete control systems were extended to accommodate crucial properties of each other [La Fauci 97], and the line of demarcation between them became fuzzy.

The conventional design algorithms for discrete and continuous control systems were also modified. The overlapping of continuous and discrete control system boundaries with one another increases the complexity of the resulting control system. The combined controllers involve not only the complexity of continuous and discrete controllers but also the added complexity of mutual interaction. A hierarchy was developed to define the control structure for designing the combined continuous and discrete control system. In this hierarchy, continuous control systems carry out the lower level tasks and discrete control systems supervise these lower level tasks and issue the top-level commands. The control systems with the above hierarchical control structure are known as "Hybrid control systems"

2.2 Robot Assembly Line operation as an example of Hybrid Control System

A robot assembly line operation is a typical example of hybrid control system. In the example shown in Figure 2-1, a printed circuit board Y is moving on the conveyor belt. As it reaches position 1, Robot A picks it up and places it on position 2 inside the station #1, where specific parts are inserted on it. When insertion is complete at position 2, Robot A picks it up and puts it on position 3. Here too, some process is done on the board. Robot A then transfers it to position 4. After completion of the board process at position 4, the board is placed on the conveyor belt again, which takes the board to position 5. Here Robot B continues to advance the board to let all the processes at positions 6,7 and 8 inside station #2 be completed. The assembled board is then picked up by Robot B and is placed on the "Product box".

Each "position" in this assembly line operation represents a "Discrete Event System", while the work done on the board at each position is the "Continuous System". As the board reaches position 1, an event is triggered and Robot A is told to pick the board up. An actuator controls the movement of Robot A, governed by differential equations. When Robot A grips the board, another event is triggered commanding the Robot to put the board at position 2. Such discrete states and events govern the control algorithm for a complete assembly line operation, describing the movement of robots and conveyor belts at each state. The combination of such continuous and discrete systems makes the assembly line operation a hybrid control system.

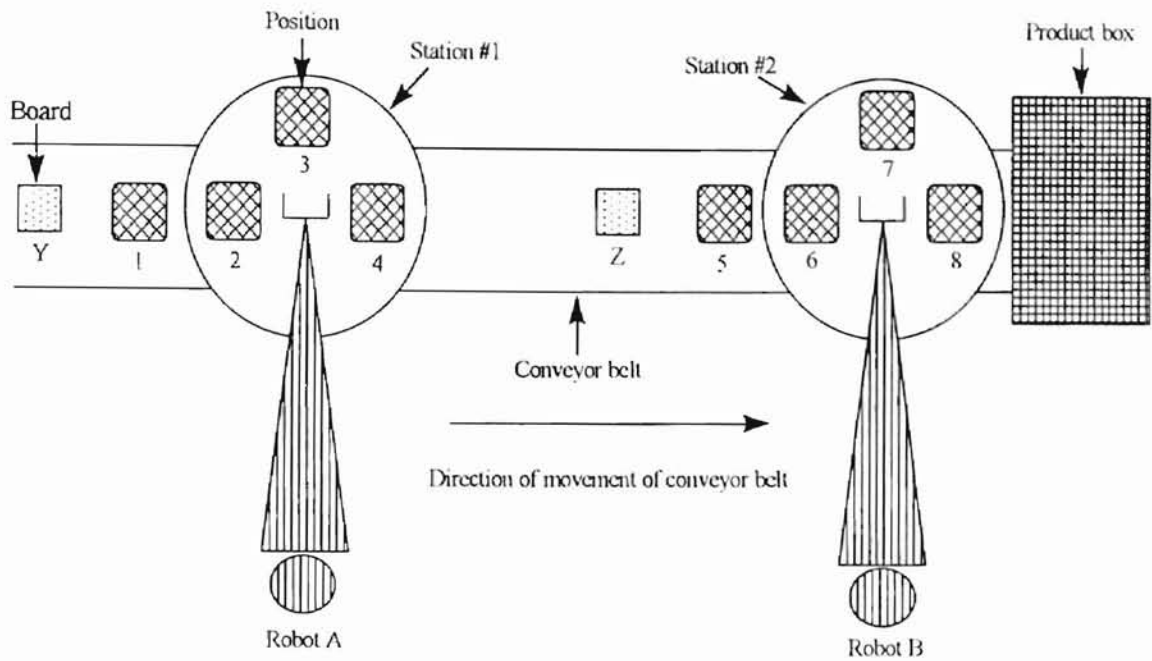


Fig 2-1: Robot assembly line.

2.3 Hybrid Control System Architecture

Hybrid control systems form a class of controllers typically built on a 3-layer hierarchical model. Though most industrial hybrid controllers today encompass a wider range of diversified control functions representing more than three layers of architecture, they can still be represented by the 3-layer abstract model [Godbole, Lygeros and Sastry 95] shown in Figure 2-2. The lowest layer of this architecture, "Continuous Systems", interacts directly with the plant control processes. This layer is modeled by using the integral, proportional, differential or difference equations. The upper layer, "Discrete Event System", acts as supervisory controller. The combination of the discrete events in this layer follows the hybrid system control algorithm. These algorithms are usually depicted by Finite State Machines, Petri Nets, etc. The middle or "Interface" layer does the communication between the continuous and discrete layers. It translates the action of

continuous layer to events for the discrete layer and conversely converts the responses (events) from the discrete layer to signals for the continuous system. In this 3-layer hybrid system architecture, any component not in the domain of the continuous system can be treated as a part of the discrete layer. This hierarchy of the multi-layer hybrid system is somewhat similar in concept to Open System Interconnection (OSI) [Tanenbaum 96] model of networking in which information gets condensed in the ascending order of layers. The topmost layer is the most abstract. In it, a single command carries all the information regarding the work performed by the subsequent layer and down all the way to the lowest layer.

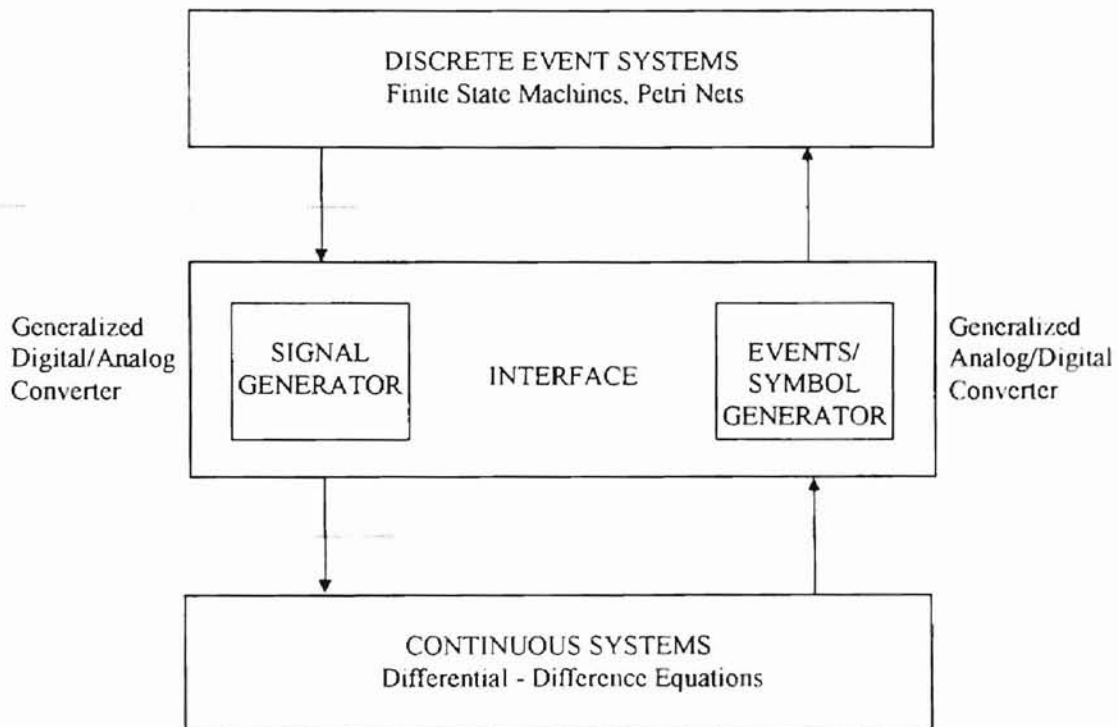


Fig 2-2: Hybrid Control System architecture ([Godbole, Lygeros and Sastry 95], page 167)

2.4 Formal Definition of Hybrid Control System Automata

“Hybrid control systems are continuous-variable and continuous-time systems with phased operations” [Deshpande and Varaiya 95]. Inside each phase of operation, the control system carries out the tasks set by static or dynamic rules of that phase. Transition between the phases takes place when the control system status within a phase evolves to a point where certain predefined transition criteria are met. At any instant, the control system can be viewed as a binary tuple of continuous and discrete states. Each phase is a discrete variable representing a single state within the state flow architecture of the control system. The continuous system defines the position of the control system within a single state.

The automata for the hybrid control systems were generalized by Deshpande and Varaiya [Deshpande and Varaiya 95]. They generalized the “hybrid system” as a tuple

$$H = (Q, R^n, \Sigma, E, \Phi)$$

where:

Q is a finite set of discrete states,

R^n is a set of continuous states,

Σ is a finite set of discrete events,

E is a finite set of edges. The edges model the discrete event system dynamics of the system.

An edge $e \in E$ is denoted as $(q_e, X_e, V_e, r_e, q'_e)$ and is enabled when the discrete state is q_e and the continuous state is X_e . When a transition through e is taken, the

event $V_e \in \Sigma$ is accepted by the system. The continuous state is then reset according to map r_e , and the system enters the discrete state q'_e ,

Φ is a set of differential inclusions that models the continuous dynamics of the system.

An example depiction of hybrid system automata is shown in Figure 2-4, with three discrete states namely 1, 2 and 3. A transition from state 1 to state 2 on edge e_1 can only take place if the continuous state is X_1 and discrete event is V_1 . Mapping it to r_1 then resets the continuous state. Similarly, the transition from state 2 to state 1 is possible on edge e_4 when the continuous state is X_4 , discrete event is V_4 and the map is r_4 . As a generalization each discrete state can go to another if the continuous state X respective to its discrete state is running and creates a discrete event V for transition to the other discrete state using its map r .

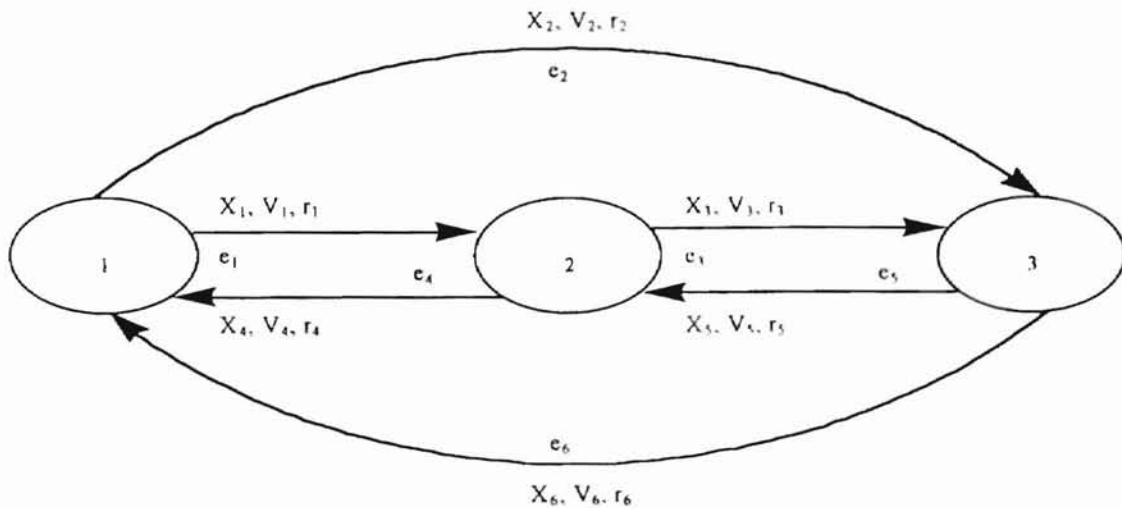


Fig 2-4: A Hybrid Control System automaton

2.5 Hybrid Control System Advantages

A hybrid control system is often the best solution for a process involving both discrete and continuous systems [Robinson and Salkas 95]. It allows all process I/O's to be configured within a single program, yielding a better organization for controlling the whole system. Also, networking is integrated at all levels – along with the system database – smoothly, securely and efficiently. Its one-program-for-the-whole-system approach allows maintenance engineers to maintain a single source code instead of multiple sources required for control systems configured separately as discrete and continuous.

An operator of a hybrid system is given a uniform interface to the processes, making it possible to recognize the problems more precisely. Since the configuration of all the processes is done within a single program, a process can be modified easily and its consequences on other processes can be seen readily. In summary, the diverse control requirements of a system can be met with reliability, flexibility, modularity and scalability by hybrid control algorithms within a single program

CHAPTER III

SUPERVISORY CONTROL AND DATA ACQUISITION

3.1 What is SCADA ?

Supervisory Control And Data Acquisition (SCADA) is defined as a system in which a central computer or operator measures and controls the processes which may be distributed at several locations, some of which may even be remote [Boyer 93]. An example is a central computer gathering information from all the processes, and based on that information, taking appropriate actions to control valves, switches, motors etc. A SCADA “package” is a set of software tools for industrial applications that facilitates acquisition, presentation, sharing, and storage of data and control equipment. Data is conditioned by SCADA packages to display real time process information in the form of graphs, tables, etc. and is passed to the control algorithms to provide a precision control to the industrial instruments. Networking protocols like TCP, UDP, and DDE are generally built into SCADA packages for sharing information. Relational databases are used for storing and viewing historical data. Databases can be programmatically accessed by Standard Query Language (SQL) queries. In summary, SCADA packages provide a comprehensive solution for industrial automation.

3.2 Where is it suitable?

SCADA packages are suitable for large-scale and enterprise-wide automation. They have built-in capabilities for configuring, data sharing over the network, error handling, storing

data, and generating reports. The SCADA packages can handle for a very large number of analog and digital inputs from instruments distributed plant-wide. The ability to handle complexities of large-scale automation makes the initial cost of installing the SCADA package quite high. The initial cost, however, is justified due to the ability of these packages to handle complexities involved in a large-scale automation project. Though SCADA packages can be used for smaller projects, their yield versus price ratio is much lower than lower-level software which could just as easily handle the same project

3.3 Main Features of a SCADA System

A typical SCADA system is shown in Figure 3-1. The main features of the SCADA system are listed below and discussed further in the following sections:

- 1) Data Acquisition and Supervisory Control
- 2) Data Presentation
- 3) Networking and Security
- 4) Database Connectivity

1) Data Acquisition And Supervisory Control

All SCADA packages have quite powerful and diverse capabilities in this area. They can acquire data from Programmable Logic Controllers, Distributed Control Systems, Hybrid Control Systems; besides a large number of analog, digital, text-string and boolean inputs and outputs. They usually include a library of device drivers (built-in programs that provide an interface between computer software and the device hardware) for the large number of instruments available in the market. Most of these device drivers can easily be configured for retrieving data and sending control

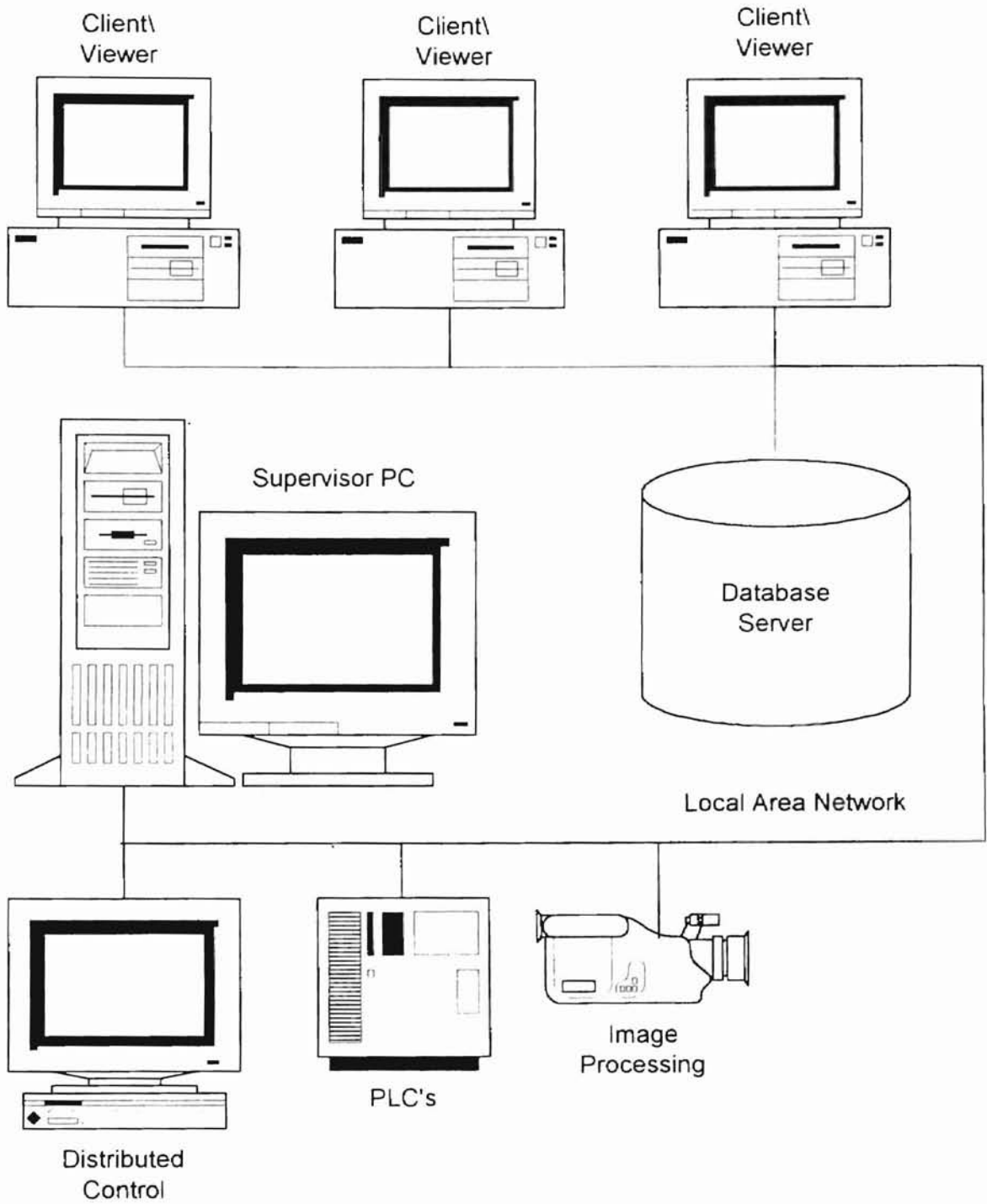


Fig 3-1 An outline of a typical SCADA system

values to the controllers. Apart from the built in drivers, SCADA packages allow the users to write their own code for customized hardware application in many text-based and graphical languages.

2) Data Presentation

The rich Graphical User Interfaces (GUI) in most SCADA packages make it easy to mimic the instruments or systems. GUI's are often called Man Machine Interface (MMI). MMI's contain a variety of graphical displays including switches, knobs, bulb indicators, meters, gauges, slide indicators, pumps and other digital/analog displays and controls. Using these graphical symbols, the process flow diagram of a large-scale factory can effectively be depicted. Objects can also be animated with or without linking them to actual data points.

Real-Time (recent) or historical (stored) data can also be viewed using graphical displays. The graphs are quite flexible and can be configured, e.g. for the scale and the units of data to be viewed beside many many other features. Or, an automatic option can be selected to let the program adjust them. Some SCADA packages even give users options to create their own graphical displays or edit existing ones. A report generation facility is provided to obtain current or historical data in spreadsheet word processor format. Reports can be configured to view the desired data in tabular or graphical forms

Alarm management is provided for viewing and acknowledging them. Different levels of alarm like low-low, low, high-high, high, etc can be set for input and output

data points. Alarms can be grouped together and viewed as a group or according to the priorities set for each group.

3) Networking and Security

SCADA packages provide the multilevel security options for different level of users. The security levels are hierarchical giving full privilege to the top level (administrator) and very little privilege to the bottom level (observer). Besides peer-to-peer networking, standard network protocols are supported for sharing data along with other users and/or nodes. SCADA packages usually support TCP/IP, UDP, and DDE for transferring data over different network topologies such as Ethernet. Many SCADA packages support the client/server model for supervisory control and management of distributed systems in large-scale industrial automations where equipment is spread all over the plant.

4) Database Connectivity

Data is logged by SCADA packages in a database on user-selected time intervals, from once a second to once a year. These databases support the open database connectivity (ODBC) at the back end of a relational database. The front end of the relational database uses standard query language to access data. Standard databases like Oracle and Access can also be plugged in to store or retrieve data from the database. When configured in sharing mode, remote users can also read and write to the database over the network.

3.4 Overview of some widely used SCADA Packages

SCADA packages popular in industrial applications share similar features described in the previous section. The difference between SCADA packages, lies mostly in user friendliness, richness of graphical user interface, network capabilities especially in handling large number of nodes, and database efficiency. Cimplicity® by GE Fanuc Automation, InTouch® by Wonderware Corporation and BridgeVIEW™ by National Instruments SCADA packages are described below.

CIMPLICITY®

CIMPLICITY® is the SCADA package made by GE Fanuc Automation. Its features are summed up below:

- It is easy to use for people familiar with Microsoft Windows™ environment
- It conforms to the 32-bit architecture of the PC's, fully capitalizing upon the inherent multiprocessing, multitasking, and multithreading capabilities [GE Fanuc Automation 95].
- Custom applications can be developed with Visual Basic language. Cimplicity's Program Editor provides an object interface to alarms, data values, etc. to create, edit, or browse for existing data points, and log status, etc. Data points can also be configured in a standard text editor or spreadsheet and saved as a comma-delimited file and imported to the Program Editor. By the same token, a configuration file can be exported to several third party packages. Changes in the configuration file can be made on the fly while running the Import/Export utility.

- The “Event Editor” in Cimplicity manages the events during the program run like acknowledging the alarms, logging data and triggering devices. Events can be invoked through a single preset action or through a combination of actions like changing values and alarm status or on reaching at a specific time. Actions can also be enforced as a result of more than one event
- A front end GUI can be created using “CIMEdit” editor. The editor has a set of drawing tool to create 2-d or 3-d graphics. It can embed OLE 2.0 objects, spreadsheets, video files etc. Objects can be animated for representing input or control actions via a dialogue box of “Object Properties” in the CIMEdit using geometric coordinates. Object can be placed at various locations on the screen. Cimplicity objects can be linked to data via “Cimplicity points” located anywhere on the network to view continuous changes
- Trending is provided to compare current and previous data from multiple sources
- Data values can be plotted by configuring different colors, lines, etc. Multi-axis plotting is achieved with unlimited numbers of pens per chart; each pen can have a different color, plotting rate, and separate axis [GE Fanuc Automation 96]
- Cimplicity allows scalability from a single Man Machine Interface all the way up to a large-scale supervisory system
- Cimplicity supports a distributed architecture that not only provides peer-to-peer networking but also a client/server model. Either the server, or each node in a peer to peer network must run Cimplicity. TCP/IP, DDE, Data Highway Plus, Modbus™ RTU, Modbus Plus, Siemens H1-TF, and serial RS-232 protocols are supported for peer-to-peer and client/server models.

- Alarms, events and other values can be logged dynamically, as configured via a dialog box with options to log values, alarm states, resources, time stamp etc. for any point. An Open DataBase Connectivity (ODBC) driver is provided for logging data to ODBC compliant databases like Microsoft Access and Microsoft SQL Server Database size is maintained by purging the older data, with an option to archive the purged data in a Comma Separate Variable (CSV) file
- Reports can be generated for data and alarms by retrieving them from the database. Data can be exported to spreadsheet files for further analysis.
- A remote computer on the network running Cimplicity can also be used to log data in the database to reduce cpu load or as a redundant backup.
- Security features allow only the authorized users to retrieve data from the database.

InTouch:

InTouch SCADA package is the product of Wonderware Corporation. It has the largest share of the SCADA packages in the industrial market.

- InTouch has two different versions for Windows 95 and Windows NT, but later versions (7.0) are compatible with both Windows 95 and NT.
- It has a library of wizards that create graphics and database tags automatically. The "WindowMaker" facility allows the user to define and configure tags, alarms, etc. using the "Special" tab. A Script Editor provides the necessary tools to write a custom base program with built-in logical, string, and mathematical expressions to help in writing syntactically incorrect code. Tag names and alarms can also be included in these expressions.

- Microsoft's ActiveX controllers can be integrated fully in the applications [Wonderware 97].
- InTouch uses the standard Windows GUI format. GUI's can be developed with a large number of objects using a variety of object-oriented design tools. InTouch supports any type of graphic resolution. Objects in the GUI can be animated with links to the tagged discrete, analog and string values and alarms. Though a library of pre-configured yet editable objects is present, more sophisticated wizards like "AutoCAD conversion" and "OEM's" wizard can also be utilized for creating data and graphic tags automatically.
- A large number of alarms can be configured with 999 alarm priorities and alarms organized into a hierarchical placement with eight levels of alarms group, each supporting up to sixteen subgroups. Alarms can be simultaneously watched from multiple remote applications using a dynamic referencing utility. Individual and group alarms can be acknowledged using a global alarm acknowledgment facility.
- The historical trending graphs are capable of simultaneously plotting sixteen charts from one or more data files. Though any number of historical or real-time data graphs can be displayed on the screen, only four charts can be displayed on a real-time graph at one time.
- InTouch supports both peer-to-peer and client/server models. Multi-platform connectivity between computers running Windows, VMS and UNIX is provided with InTouch's proprietary NetDDE protocol. TCP/IP, NetBIOS, Novell, Token Ring, Arcnet, DECnet and serial communication protocols are all supported. Communication protocols of several Programmable Logic Controllers are also

supported such as Allen-Bradley, Siemens, Modicon, Opto 22 and Square D [Wonderware 95].

- I/O data sources as well as the GUI of a process running on a remote PC can be viewed using DDE protocol or Wonderware SuiteLink protocol.
- Applications can be developed using Object Linking and Embedding (OLE) for Process Control (OPC), a Microsoft standard of communication to provide a standard interface between business systems, control systems and industrial devices [National Instruments Corp 97d].
- Discrete, real, integer and string values can be configured as database tags. Intouch can access databases such as ORACLE, Microsoft SQL Server, Industrial SQL server, Sybase, dBase and Microsoft's ODBC compliant databases. Any number of CSV spreadsheets can be used for the database tag configurations. The Dynamic Referencing utility allows to toggle between change database referenced tags and input /output tags on the fly

BridgeVIEW

National Instrument's SCADA package, BridgeVIEW, is somewhat unique among traditional packages. It provides the test and measurement capabilities in addition to the utilities of a typical SCADA package, and includes a fully functional, user-friendly yet powerful high-level graphical programming language called "G". This seamless integration of graphical programming environment along with the SCADA functionality provides an unprecedented flexibility to the end user for customizing their applications

BridgeVIEW (1.0) SCADA Properties

- BridgeVIEW also supports 32-bit architecture of Windows95 and NT
- The BridgeVIEW Engine is the heart of this SCADA software [National Instruments Corp 97c] It has a proprietary real-time database that keeps track of tag and process information. When the engine launches, it reads a configuration file (with an extension of scf - SCADA Configuration File) that contains all the required information for each tag in the system. The Engine runs separately from the rest of the SCADA application and device servers to increase data logging efficiency.
- BridgeVIEW Engine uses tags for data acquisition. The "Tag Configuration Editor" creates/edits a string, memory, discrete or analog tag to define its link with the physical I/O point, and associated process information such as name description, engineering units, scaling, alarming etc. The information regarding all the tags is saved in a configuration file, which can be exported or imported to/from delimited text files or spreadsheets.
- The Graphical User Interface (GUI) can support a large number of objects. BridgeVIEW has a built-in library of a variety of graphical displays, trend charts, push-buttons, alarms etc. Pictures and symbols from many commercial drawing packages can also be imported into the GUI. The MMI wizard allows user to create and configure the objects easily. Objects on the front panel can be linked to any of the tags (as configured in the Tag Configuration Editor) and can be animated as well.
- Alarms can be configured for any non-text tag using the Tag Configuration Editor. Alarms can be enacted for hi-hi, hi, lo, lo-lo conditions against the limits set for the tag. There are 15 priority levels and user groups in which alarms can be distributed.

BridgeVIEW alarm summary display facility allows for selective viewing, e.g. the alarms of highest priority level, or of a particular group. Besides alarms, events such as start, stop, and faults can be logged to the disk and viewed by the Event History Display facility.

- Historical data can be logged using a high throughput threaded Citadel™ database. Citadel utilizes the 32 bit Cyclic Redundancy Check (CRC) for data integration and compression techniques [National Instruments Corp 97a]. Historical Trend Viewer allows the user to display logged data for any number of tags. Each GUI can have several Historical Trend Viewers. Real-time data for any number of tags can be displayed on the GUI using the Real-Time Trend graph.
- Peer-to-peer networking and client/server model are both supported. Device servers are the link between the BridgeVIEW Engine and the hardware devices. Devices like PLC's, PC DAQ boards, and remote I/O's communicate with the device servers. BridgeVIEW Engine in turn communicates with the device server to get the I/O values associated with the tags. BridgeVIEW has the device servers for several PLC's like Allen Bradley, Siemens, Modbus, GE Fanuc, Omron etc and I/O networks like National Instrument DAQ, Optomux, Foundation Field Bus etc. Device servers can be configured using DDE, OPC, Industrial Automation Kernel (IAK), Virtual Instrument (VI), TCP/IP and UDP network protocols. Tags are assigned to device servers using the Tag Configuration Editor to get their respective values.
- Citadel™ database is used for logging the discrete, analog and string values of the tags. It timestamps each data value entered in the database, so data in Citadel can be accessed by defining the time limits and tag names. The data stored in the Citadel can

be exported to Microsoft Excel. Citadel is also accessible to other applications via SQL and ODBC interface.

Test And Measurement in BridgeVIEW (1.0)

- BridgeVIEW uses a graphical language for test, measurement and analysis, a very useful feature providing flexibility not available in other SCADA packages
- BridgeVIEW combines the graphical editing and execution system upon the object oriented, graphical, dataflow language G in which programs are created in block diagrams. As a dataflow language, any section of G code can execute as soon as it receives all of its input data, hence G applications have a potential of executing much faster than the sequential text based languages. This combination of object-oriented and dataflow programming style makes the G language easy to program, easy to debug, and faster. In addition, its GUI's can be very intuitive
- BridgeVIEW has the same programming concepts and style as LabVIEW. A program in LabVIEW (or BridgeVIEW) is written using three editors [Mahmood 96], front panel, block diagram and icon/connector editor. A "Virtual Instrument" can be created on the front panel by mimicking the actual electrical instrument. The input terminals of the actual instrument are depicted as "Controllers" in the Virtual Instrument and the display units as "Indicators". BridgeVIEW has an extensive library of controllers and indicators in the "Control" palette tool. Every Virtual instrument has a "Block Diagram" associated with it. The block diagram editor is used to write the actual graphical code. The function palette in the block diagram contains structures, string, array, mathematical, statistical analysis, file I/O, time, communication, data acquisition and several other functions. Any number of third

party functions can also be included in the functions palette library. Since the programs in BridgeVIEW are written using objects, there are various tools in the tools palette to help define the flow of the program. Arguments of one program can be passed to another using the connections made by an icon/connector editor, which allows a program to be used as a subroutine inside other programs.

- BridgeVIEW programs are hierarchical and modular [National Instruments Corp 96] Complex problems can be divided into subtasks and the programmer can create an object for every subtask in BridgeVIEW G code. By interconnecting these objects, the main program is developed just like a flowchart. Main programs in BridgeVIEW are much easier for the non-programmers to understand

CHAPTER IV

TRX ASSEMBLY LINE PROJECT

The TRX line assembles communication boards for the booster stations (repeaters) of a cellular (mobile) telephone network. The communication boards are processed through various stages in the assembly line where parts are automatically inserted and soldered into the board. Somewhere during the process, covers and labels are also put on the board. After assembly completion, boards are passed through test stations for functionality tests. The TRX assembly line is a hybrid control model. Each stage in the assembly line is unique in that each assembly line operation uses either continuous or discrete control system technique for gripping, holding, inserting, and soldering parts.

The BridgeVIEW SCADA package was selected to provide the supervisory control software for the TRX assembly line. The supervisory software's role was to create a link with several standalone automated units using Dynamic Data Exchange (DDE), track the location and process information of each board on the TRX assembly line, and provide this information to all units that require it. Supervisory control software was also responsible for updating the product database information in real time for review by authorized users on the network.

The primary goal of this TRX assembly line was to have a high production rate at economical cost. Ideally each board should take about eight minutes from start to finish. As multiple boards are processed concurrently the boards are produced successively after the initial startup of eight minutes. This TRX assembly line should be capable of producing five different types of communication boards as per demand and without

interruption. With regard to the complexity of TRX assembly line operation, this overall production rate goal is considered to be highly ambitious. The line is fully automated, eliminating human involvement, and reducing breakdown frequency and percentage, which is very important since a breakdown costs about five thousand dollars per minute and a rejected board costs \$500 because of gold and rare metal plating. The cost of running the line would be economical with high profit ratio if the goals are met.

4.1 Project Overview

The overall TRX assembly line is shown in Figure 4-1. The TRX assembly line can be divided in nine stations. A conveyor belt runs through the stations. There is a system computer for supervisory control and a plant computer for user interface. Stations 1, 3, and 6 are robotics driven by Adept microprocessor-based controllers. Though each station performs its job using its own proprietary control systems, they all depend on the supervisory computer to tell them what to do and then relay the processing information back to the system computer.

The TRX assembly line operation starts with the plant computer, where the operator provides a list of the type-coded serial numbers of the boards to be processed. This list is relayed to the system computer, which keeps these serial numbers in its database. This serial number is used to store and retrieve production information for a particular board to/ from the database. The board's sequential flow begins with the board coming from a loader through the conveyor belt and stopping at Station 1's prequeue stop. It then goes to the queue stop where it waits to be processed. It enters Station 1 as the station becomes available. First, its serial number is read by the bar code reader. This serial number is

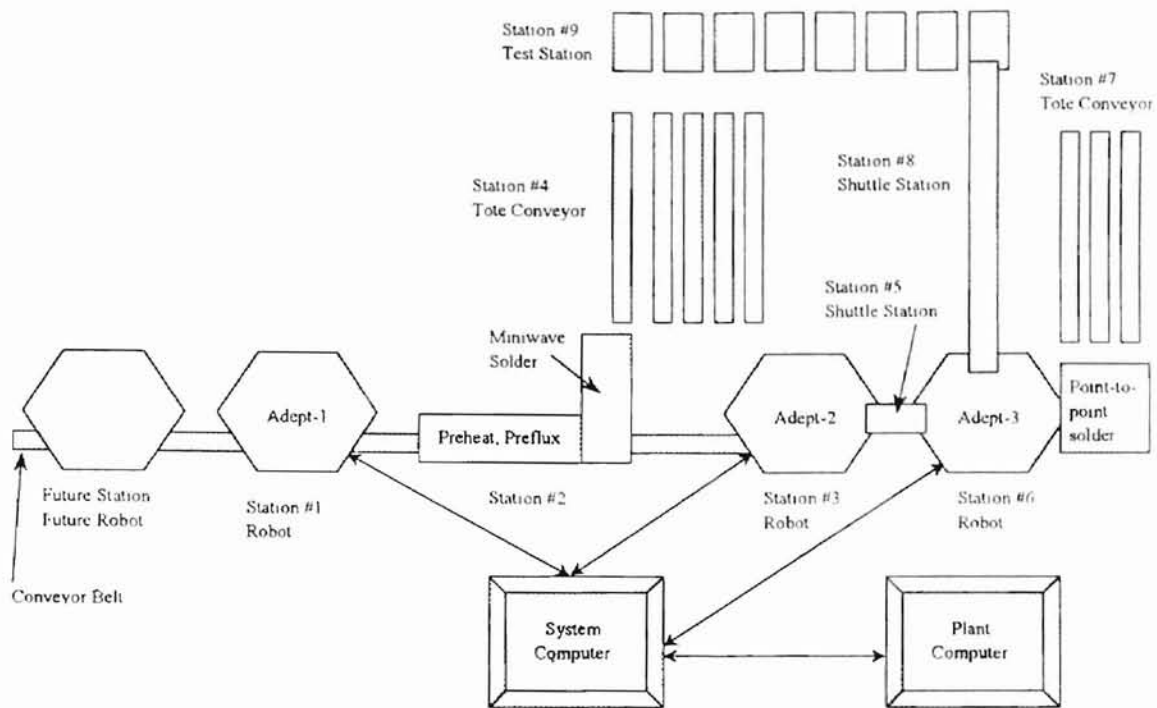


Fig 4-1 The TRX assembly line

passed on to the System Computer via Adept Controller 1, which returns important information needed by the Adept Controller 1 to process the board, such as board type, etc

The board is then picked up by the robot arm in Station 1 and put at the work stop. Based on the information provided by the System Computer about the board, Adept Controller 1 requests several machines associated with Station 1 to provide the right parts (mostly electrical) for that particular board type. The robot arm then goes to these machines one by one to get these parts and place them at the right location on the board. The exact location for insertion of parts is found using image processing techniques. After parts insertion is complete, the board is picked up again by the robot arm and placed on the conveyor belt for processing in Station 2.

Before entering Station 2, the board stops at prequeue and queue stops waiting for Station 2 availability. As board enters Station 2, its serial number is read again by the bar code reader and sent to the System Computer via Adept Controller 1 in a manner similar to the previous station. Station 2 takes the board now loaded with parts through a preheat and a preflux oven, where the board is heated and flux is put on the parts which are then soldered by the Miniwave soldering machine.

The board coming out of Station 2 on the conveyor belt waits at prequeue and queue stops prior to getting processed in Station 3. Like in previous stations, its serial number is read by the bar code reader as the board gets into Station 3. The serial number is passed to the Adept Controller 2 and System Computer in a way similar to Adept Controller 1. The information received back from System Computer is then transferred to Station 4's tote conveyor, which passes the frames for the particular board to assembly nest in Station 3 for placement by its robot arm. The frame is labeled and more electrical parts are inserted and screwed to the frame/board assembly, all using a robot arm which finally places the frame/board assembly to a shuttle (Station 5) that takes it to Station 6.

Having received related information about the board at Station 6 the serial number of the board is read again, and passed to Adept Controller 3. Station 6's robot arm picks the frame/board assembly up, inserts some more electrical parts, then put it into the Point-to-point soldering machine for soldering the electrical parts.

Station 7's tote conveyor gets the board information through Adept Controller 3 and passes the right cover for the board to the assembly nest at Station 6 to be put onto the

frame/board assembly and screwed to the frame. The robot arm in Station 6 takes the finished board out and puts it on the shuttle (Station 8) for delivery to Station 9

Station 9 reads the board serial number again and lets Adept Controller 3 get the board information from the System Computer in a way similar to other Adept Controllers. The finished board is then taken into Station 9's test fixture by the Gantry Robot controlled by Adept Controller 3 for a battery of tests. After tests are completed, the board is taken out from the test fixture by the Gantry Robot, and depending upon the test outcome, put into the pass or fail cart which thereby marks the end of the TRX assembly line operation.

Though narrated above as the sequence of operations on a single board, it is in reality an assembly line operation, where each machine is processing multiple boards not necessarily in serial order. Boards are manually taken in and out for various reasons. It is the System Computer's supervisory role to keep complete information about every board processed, in process, or to be processed. This supervisory control frees the system from keeping any serial order on the boards for proper identification

4.2 System Analysis

This section describes the approach taken to understand the project along with the software and hardware requirements.

4.2.1 TRX Assembly Line Project Management

The project management for the TRX assembly line project involved understanding the scope of the project, assigning roles to the team members in the project, compiling the variables list, establishing the handshaking protocols for

communication between Adept controllers and System Computer, benchmark testing, and numerous small issues.

The complexity and the size of the system warranted a team approach for handling the project. The project was divided in the following categories; Project Integration, GUI, Process Database, Communication, Handshaking, and Product Database. A team of three engineers was chosen by the management based on their experience, expertise and availability. Project Integration, GUI and Process Database categories were assigned to the two senior members of the team while Communication, Handshaking and Product Database categories were assigned to the author.

The author's primary role was to lay down the detailed supervisory control software through research and extensive communication with the client's engineers who were installing the TRX assembly line. Other roles included setting up the network communication protocol between the Supervisory System Computer, Adept controllers and the Plant Computer, define a Hybrid Control model based on the set of discrete events (handshaking protocols between System Computer and Adept Controllers), and developing the Product Database.

For time management (schedules and milestones), the project was set into three phases. In the first phase, a thorough study of TRX assembly line was done and protocols were set. The second phase involved writing the actual code and integrating different modules of the code. The third phase was to install and debug the program on the site.

a. Variables List

A list of variables thought to be needed for handshaking, communication, GUI and database was created. These variables were needed to exchange process information back and forth between the System Computer running supervisory software and various machines including robot arms inside the stations. Meaningful column names for System Computer's Product Database were also chosen to keep boards, status information linked to their serial numbers. Variables and column names were chosen to depict the flow of the boards through different stations in TRX assembly line. A part of the variables list is shown in Figure 4-2 as an example.

b. Protocols

Establishing protocols for handshaking between the Adept controllers and supervisory System Computer was a core requirement for the supervisory control software of this TRX assembly line. These handshaking protocols map to a set of discrete events in Hybrid Control model, that take the supervisory software from one state to another state. These protocols are basically the same but differ in details for each station. A general description of the protocols and database is provided below along with an example questionnaire used to set one of these protocols.

- Automatic and Fixed Board Mode Protocol

The automatic and fixed board mode protocols were defined to allow operation of the TRX assembly line in either of two modes (1) Automatic

(Normal), (2) Fixed (Debug) or semi-automatic. In Automatic mode, System Computer will communicate with the TRX assembly line and run supervisory software. In Fixed mode, the TRX assembly line will run with manual intervention to debug. The modes can be set separately for each station, giving the flexibility to debug the TRX assembly line station by station. A part of the automatic and fixed board protocol is shown in Figure 4-3.

DATABASE STORAGE INFORMATION			
STATION # 3 ADEPT ROBOT CELL (Stations 3 through 5)			
Polling Flag	Variable Name	Description	Variable Definition
Hot Flag	st3.need.readin	System computer needs to read the input values. A new board has arrived. Read the Adept data and write the data into Adept system.	0 = Do not read 1 = Read
Poll Always	st3.cur.step	Current step in AIM sequence (the array value equals the task assigned to the sequence). Main sequence only	String
St3 read in	st3.barcode.rd	Barcode read from product at que stop position to be read by BridgeView	1172046/1 1172046/2 1172046/3 1172068/3 1172296/3
St3 read out	brd.status.pass brd.status.fail	Status of the board leaving robot cell. One and only one of these two variables can be true.	0 = False 1 = True Note: Brd.status.fail and Brd.status.pass cannot both be 1.
As needed	user.log.name	User log on name	String of user name

Note:

- The information shown under Polling flag is as follows:
 - Hot Flag - BridgeView should constantly monitor these variables and take the appropriate action when variable is true
 - St3 read in - BridgeView should poll these variables whenever the Hot Flag st3.need.readin goes to true
 - St3 read out - BridgeView should poll these variables whenever the Hot Flag st3.need.readout goes to true
 - Poll Always - BridgeView should poll these variables at set intervals
 - As Needed - variables that BridgeView may need. Polling priority and interval dependent on frequency of use by BridgeView.

Fig 4-2: A part of the list of variables

Semi-Automatic (Fixed board) Mode Protocol

There will be two modes of operation for the TRX line. The most prominent mode being the Automatic mode where the line has direct communication with System Computer. The second method will be the semi-automatic or fixed board mode, which will have no communication with System Computer.

Automatic Mode

The automatic mode will allow System Computer to read and write to the Aim variables at each robot station. System computer will be able to take information from these variables and track the boards through the line along with storing data to the database and to the historian database. System computer will also be able to provide information to the adept controllers such as determining what product type to run based on the bar-code information and the pass/fail status from the upstream station.

Semi-Automatic or Fixed Board Mode

During debug of the machinery or if System Computer is not functional, fixed board may be used to run the TRX line.

In order for the station to be locked into fixed board mode the operator will need to go to the fixed board menu and select the proper board type to be run. This will have to be individually done for each station controller. After the operator has selected the fixed board mode the Adept Controller will no longer accept information from System Computer. The Adept Controller will still read the bar code and continue to track the board through the cell as in automatic mode. Attention should be taken that the board being processed matches the board selected by the operator. Otherwise the robot could try to place parts at undesired locations, causing the robot to crash.

Station 3 will be slightly different from station 1 and 6 in that it will need to read data from a floppy disk. The bar-code reader will read the bar code just as in the automatic mode. The difference is that now the Adept Controller will compare the bar code reading (printed circuit board serial number) with entries in a table which has been read from the floppy disk provided by the Plant Computer. At that time a search will be done to find the same serial number in the table. When a match has been found the TRX serial number, product number, revision number, and date will be read from the table and written to the label, which will be placed, on the board frame. The serial number could now be deleted from the table to reduce the time needed for future searching.

It should be noted that in fixed board mode the station 6 controller would not know if a board has passed station 9 tests. The station 9 pass/fail status is currently passed back to System Computer, not the station 6 Adept Controller.

Fig 4-3: A sample of Fix board mode protocol

- Purge Protocol

The purpose of purge protocol was to inform the supervisory System Computer about a board manually purged from the TRX assembly line because of a problem in the board such as some parts of it broke down. The

information regarding the purged board like serial number of board, time of purge, etc. is passed to the supervisory System Computer to update the product database and mark the board as purged. A part of the purge protocol is shown in Figure 4-4.

- Board Re-entry Protocol

This protocol was needed to be able to rerun a board through the TRX assembly line. Upon entering a station, the bar code reader reads the serial number of the board and the Adept controller uses it to query the supervisory System Computer about board's status information. If the board does not have status information, it implies that the board never ran through the assembly line; otherwise the board is reentering the station. The operator can either rerun or remove the board from the station. Figure 4-5 shows board re-entry flag definitions.

- Benchmark Test for Database

Benchmark tests were performed to select the commercial database with the fastest access time. A board typically gets through a station within two minutes during which several queries have to be made on the database. Some events allow only a few seconds for the query. The benchmark testing results for Microsoft® Access, BridgVIEW™ native databases "Citadel", and "dBASEV", using Access front-end, Microsoft® Excel, and National Instruments® Standard Query Language (SQL) Toolkit are shown in Figure 4-6.

Purge Protocol

The purge command has been introduced for the purpose of manually removing boards from the cell or assembly line. Without the purge, it would be impossible to remove a board for any reason and record such incidence to the database.

Thus, an operator will be allowed to purge by first removing the board from the work cell followed by pressing a custom purge button on the Adept menu. This in turn will bring up another menu where the operator will be able to specify the location from which the board is being purged. When the operator presses the button to remove the board the Adept Controller will prompt the operator "Are You Sure?" If the operator responds "Yes", the Adept Controller will purge the product number, serial number, revision number, and date/time for the board at that location. That unit will also be reinitialized for the next board (i.e.: clamps open, lift down, global variables reset, etc.). If the operator responds "No" the Adept Controller would skip to the end of the purge program and bypass the purge routine. No hot flags would be set.

System Computer will also be tracking the boards through the cell. The hot flags for the System Computer to look for will be st1.purge, st2.purge, st3.purge, st6.purge, and st9.purge. These variables will be set to "1" during a purge by the Adept Controller, who will also transfer the product number, revision number, serial number, and date/time of the board being purged to the stX.purge.prod, stX.purge.rev, stX.purge.ser and stX.purge.date variables (where X is the station number). This will allow System Computer not to miss recording the purged information. After a board has been taken out via the purge, the System Computer marks it as a failed board, stores other data, then sets the purge variables to null (" ") and the stX.purge equal to "0"

Fig 4-4: A sample of Purge mode protocol

Board Re-entry Flags Definition				
The following are the proposed board re-entry procedure flag conditions (The combination of "Pass", "Fail", and "Rerun" status depicts the processing history of the board on the line)				
Bar-code Reader At	Previous Status of Board Read	Pass	Fail	Rerun
Station #1	Station # 1 New Board	1		
Station #1	Station # 1 Failed Board			1
Station #1	Station # 2 Failed Board			1
Station #2	Station # 2 New Board	1		
Station #2	Station # 1 Failed Board		1	
Station #2	Station # 2 Failed Board			1
Station #3	Station # 3 New Board	1		
Station #3	Station # 2 Failed Board		1	
Station #3	Station # 3 Failed Board			1
Station #6	Station # 6 New Board	1		
Station #6	Station # 3 Failed Board		1	
Station #6	Station # 6 Failed Board			1

Fig 4-5 Table of Board reentry Flag Definitions

<p>Benchmark Tests (Citadel, DbaseV, and Access were used in testing. Worst case timings noted involving multiple tests)</p> <p><i>EXCEL/Query for Windows95:</i> Retrieve table (1000 rows by 30 columns Char*8 177Kb data) = 2 seconds Insert, delete, or update a row = faster than timer resolution of 0.1 seconds</p> <p><i>Microsoft Access for Windows 95</i> Results are similar to the <i>EXCEL/Query for Windows95</i>.</p> <p><i>National Instruments SQL Toolkit:</i> Retrieve with selection criteria 1000 rows by 30 columns of Char*8 177Kb data = 11.70 seconds Retrieve within a range 100 rows by 30 columns of data from above table = 1.7 seconds (mixed data types: Char*6, Date/Time, Char*32, Char*8, [8,2], [8,2], [8,2], Char*32) Insert Single Row of 30 columns = 0.17 seconds (In 1000 rows by 30 columns of Char*8 177Kb data table) Find & Retrieve Single Row from 1000 rows by 30 columns of Char*8 177Kb data table = 0.27 seconds Find & Update Single Row from 1000 rows by 30 columns of Char*8 177Kb data table = 0.17 seconds Delete Single Row from 1000 rows by 30 columns of Char*8 177Kb data table = 0.17 seconds Retrieve with selection criteria 1020 rows by 7 columns of mixed data types 1.1 MB data table = 3.85 sec</p>
--

Fig 4-6. Results of the benchmark tests on three databases using three front-ends

- Questionnaires

Questionnaires were extensively used to get a clear and precise understanding of the working of the TRX assembly line. Several questionnaires were sent to the client's engineers at different times to clarify the networking, handshaking and database protocols in a multiple-choice format. This was done to let the client's engineers explore all the possibilities of a situation before finalizing the design. Through these questionnaires, many small issues were resolved smoothly and with consensus of all the engineers involved in the project. A part of the questionnaire is shown in Figure 4-7

- c Eventflow Diagrams

Eventflow diagrams were drawn to depict the flow of board through each of the stations in the TRX assembly line. Eventflow diagrams differ from the software

- 1 Does truth table on page 1 of 5 in "Board Re-entry Procedure" lists all the possible situations?
 - A. Yes. If a situation outside this table occurs, send error signal.
 - B. No. If a board was passed from a station and reenters there again, write "Fail" so that it does not get processed again.
 - C. No. If a board was passed from a station and reenters there again, write "Rerun" so that operator could decide what to do.

2. Second last line, Paragraph 1, page 1 of 5 in "Board Re-entry Procedure" reads "System Computer will read this data and store the pass in the database then". Does System Computer also write board in/out date/time as well in the database along with pass/fail status?
 - A. Yes. BV should update variables as soon as they become available.
 - B. No. BV can hold some info in memory to combine several update operations when doing so makes it more efficient.

- 3 Last line, Paragraph 3, page 2 of 5 in "Board Re-entry Procedure" says "If the board did not pass station 2, the Adept will write a "I" in the station 2 variable brd.status.fail." Is it correct?
 - A. Yes. The variable referred to in "Station #1 Variables" as st1.opr.res is typo error.
 - B. No. This is a typo error. It should read "...will write a "0" in the station 2 variable combo.brd.stat." Note that for station 2, only one variable "combo.brd.stat" is used to indicate pass/fail status (0:fail, 1:pass) unlike for station 1 where the two variables "brd.status.pass" and "brd.status.fail" were used. Also note that possible inputs are 1,0 and not pass, fail as listed in "Station #1 Variables".

Figure 4-7: A Sample Questionnaire

flowcharts in a way that actual physical location is defined along with the software actions. Each station in the TRX assembly line has its own eventflow diagram because of their distinct physical locations. Links are made between different stations to depict board flow through each physical location in the TRX assembly line. The software parts of the eventflow diagrams are the actions or responses by the supervisory System Computer and/or the Adept Controller at a particular physical location. The eventflow diagrams in fact, provide the blueprint of the supervisory control software for the TRX assembly line. A part of the eventflow diagram for one of the station can be seen in Figure 4-8

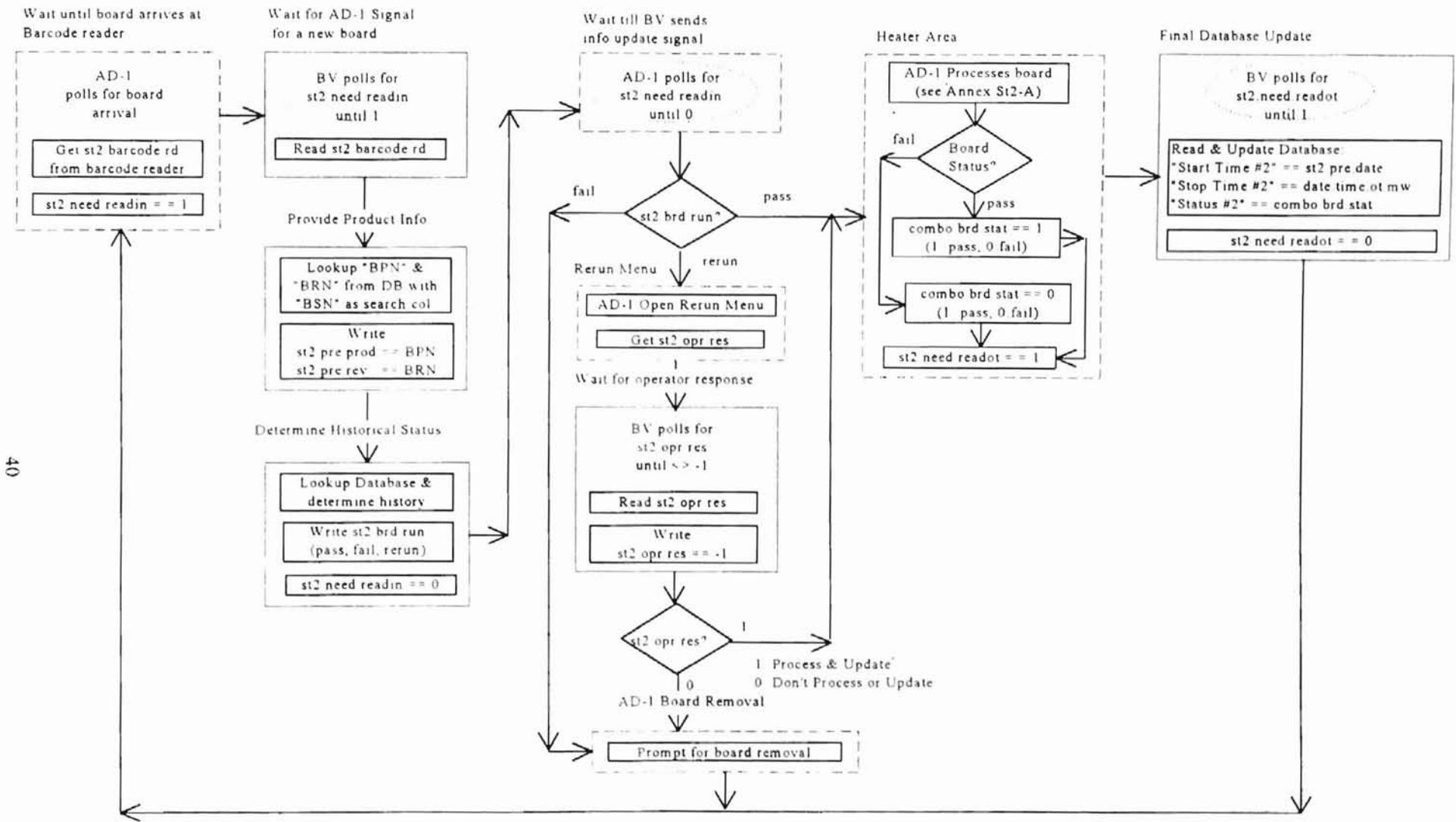


Fig 4-8. A part of eventflow diagram of one station

4.2.2 Supervisory System Hardware

A supervisory system was required to provide the controlling link from the Plant Computer to the three Adept controllers, besides the ability to monitor, analyze and troubleshoot the TRX assembly line. Intel 233 MHz Pentium processors with MMX technology, 64 Megabyte of RAM and 2.5 Gigabytes of hard drive were found to provide adequate speed of execution and storage for the supervisory system to achieve the above objectives

The computers and Adept controllers were interconnected and connected to the outside world through 100 Megabits/sec Ethernet adapter cards and hubs, RJ-45 connectors, and twisted pair cables

Windows NT was preferred over Windows 95 to be the System Computer's operating system because of its superior architecture. Its multithreading capability was needed to run different programs simultaneously and in a truly parallel architecture not on a time-sharing basis. A crash in a single program causing the whole operating system to crash was not acceptable for the TRX assembly line operation; therefore, Windows NT was once again preferred over Windows 95

4.2.3 Supervisory Software Architecture

The architecture of the supervisory software of the TRX assembly line was designed with modularity in mind, thus a layer was defined for each task. Each layer was encapsulated in a separate module, totally independent of, yet able to interact with, the others. These layers or modules, acting collectively, formed the supervisory software. Main considerations were. (a) Hybrid Supervisory Control

Software, (b) Supervisory Software Algorithm, (c) SCADA Package Selection. A brief description of each follows,

a. Hybrid Supervisory Control Software

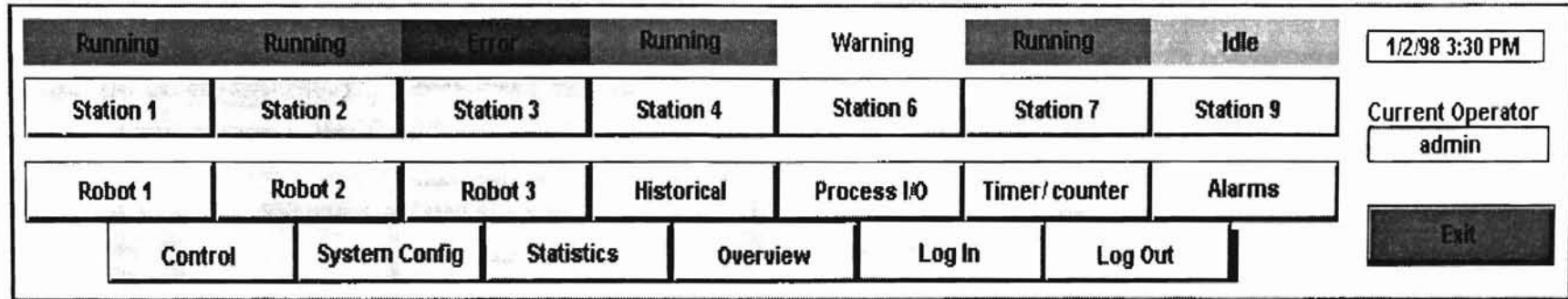
The hybrid supervisory control software or supervisory discrete event system (see Figure 2-1) for the TRX assembly line was made up of four different layers. These layers were GUI, handshaking, database and communication. All layers were developed independently and then incorporated into the main program. The top-down architecture style of the main program had the GUI at the top layer, handshaking at the middle layer, and database at the bottom layer. The communication layer in the main program was separate and was running in parallel to the other layers. Following is an explanation of the different layers in the supervisory control software.

- Graphical User Interface (GUI)

The GUI of the TRX assembly line was composed of several screens. There were screens showing historical data, process I/O, login, logout, alarms, statistics, timer/counter, system configuration, control and overview of the TRX assembly line. The main screen was able to call most but not all of the other screens directly. There were also screens for each station and robot. The main GUI command center is shown in Figure 4-9a.

The GUI screens provide the ability to control and display status information for a selected part of the TRX assembly line. For example, the Overview GUI

TRX Assembly Line



General Description:

- 1) This panel will be the main panel of the system and will reside either at the very top or very bottom of the page. The rest of the screen will be covered with any open panels.
- 2) Status buttons will indicate station status with colors and text: Red - Error; Green - Running; Blue - Idle.
- 3) Historical data and Process I/O will be enabled only after Station # or Robot # have been pressed. Global will be necessary to indicate which station is being used.
- 4) Dynamic loading and unloading will be used here. - Each button will unload the last program and reload the new one.
- 5) The built-in BridgeVIEW security system will be used. When a low-security-level operator logs into the system, buttons will be disabled and grayed out to limit their access.
- 6) Station 5 and 8 do not have variables being monitored by this system.
- 7) Indicators will show current day/time and current operator logged into system.

Note: One of the several screens will always be opened depending upon operator selection

Fig 4-9a: The "Main" GUI command center

screen gives an overview of boards process information in all the stations of the TRX assembly line, as shown in Figure 4-9b. The GUI screens were also assigned security levels to restrict access from operators with lower access levels. The GUI screen of login and logout required a valid name and password for security level verification before allowing the user to access the TRX assembly line.

The GUI screen's data was updated through different sources. Process I/O, Alarm and many other GUI's were updated in real time as soon as a change or event occurred on the TRX assembly line. The Historical GUI, on the other hand, reads historical data from the Citadel database for updating the screen

- Handshaking

The handshaking layer was the core of the supervisory control software as it was responsible for controlling the actions of the Adept controllers. In the context of the Hybrid Control model, the handshaking layer was the discrete event layer. The automatic and fixed board mode protocols, purge protocol, and the board reentry protocol were all implemented in this layer.

The handshaking layer continuously monitored for the events to activate various protocols on time response. For example, the board reentry protocol is activated whenever a new board enters the station for processing. The product database is then queried by another protocol to check the status of the entering board. The handshaking layer also monitors the mode of each station, and if it finds that the fixed board mode flag is set for one or more stations, stops the

supervisory control software for those stations. Similarly, if the purge protocol is set to true or the board is exiting the handshaking layer, it updates the product database.

- Database

The functionality of the database layer is to maintain the product and process databases of the TRX assembly line. The product database is queried or updated using SQL whenever a board enters or leaves a station, normally or upon manual purge. The process database is accessed to display the historical process data for one or more stations. Product and process databases are further described below:

1. Product Database

The product database keeps the record of each board ever entered through the TRX assembly line. The records were kept on a station-by-station basis for interrelated reasons. First, it prevented a board which had already processed through a station from rerun, which would crash the robot as it tries to assemble parts over the ones already assembled in previous run. Second, it alarms the operator of a station where boards are failing frequently. A part of the Product Database functionality requirement is shown in Figure 4-10.

The product database was developed using Microsoft® Access 97 database as per client's requests based on benchmark test results. The database field definitions for the Access 97 Product Database were Board

AUTOMATION PROJECT PRODUCT DATABASE INTERFACE

1.0 Introduction

This specification will describe the functional requirements needed to design and implement a run-time database (DB) for automated production. The DB will be an extension of a factory floor enabler (i.e. BridgeVIEW), and will be transparent to the user. The enabler will work interactively with several workcells constructing the products.

The line today will produce 5 different varieties of product, but must be easily expandable for addition types. The main function of the system (Enabler and DB) will be to provide product information to the various workcells in the line.

Each product produced will be labeled with a unique serial number via a bar code. This serial number will be logged in the DB with various additional information included but not limited to.

- Module product number
- Module serial number
- Module revision number
- Board product number
- Board serial number
- Board revision number
- Year and week
- Pass/fail status for each station
- Time in and time out for each station

Each workcell (except 1) will read the serial number when the product reaches the entry point, and convey it to the enabler. The enabler will then relay product information back to the workcell. This information will identify the product to be built, the serial number does not contain the product

Three (3) of the controllers will require product information only. That is, I have a serial number, browse the DB and tell me what product it is. The last controller will be placing the module serial numbers on the products and therefore requires additional information.

2.0 Database functions

The DB chosen will be an industry standard SQL DB capable of simultaneous users (Access 97). It shall be easily expandable to include additional product information such as SPC (Statistical Process Control) data from each workcell.

Capabilities to conduct a query on any field in the DB must be provided. For example, a query on all serial numbers related to a specific product and revision.

Though this functionality does not need to be provided within this workscope, it most likely will be needed in the future.

Several tags will also be logged to the DB after the product is complete

Fig 4-10 A part of Product Database functionality requirements

Pass/Fail status	Integer
Time in	String
Time out	String

Each time a query is conducted on the DB for a serial number, the current board location field will be updated. The product location values will be as follows:

- Cell 1 110
- Cell 2 120
- Cell 3 130
- Cell 6 160
- Auto. Unlimited 165
- Cell 9 190
- Final Test 200
- Test station 1-9 201-209

The product database contains integer fields for the location of each assembly and assembly status for each workcell

Location

The location is represented by a 8 bit unsigned integer. The integer is defined as given below

Value	Definition
0	Board not processed. An entry has been made into the database, but production of the assembly has not begun.
1-254	The value represents the defined location in the assembly line.
254	The assembly is assembled and failed final test.
255	Board complete. The assembly is assembled and has passed final test

Status

The status is represented by an 8 bit signed integer. The integer is defined as given below

Value	Definition
-1	Product has not entered the workcell. (Null)
0	Product failed workcell process. (Fail)
1	Product passed workcell process. (Pass)
2	Product passed workcell process after retry. (Retry)
3	Product passed workcell process after manual intervention. (Manual)

This will allow an easy query of the database. A value of <0 indicates all products not yet processed at an individual workcell. A value of 0 indicates a failed product. A value of >0 indicates a product which has passed either normally or with assistance. A value of >1 indicates products that required assistance to pass.

Fig 4-10 (Continued): A part of Product Database functionality requirements

serial number, Board product number, Board revision number, Module serial number, Module product number, Module revision number, Build date, and Board location. The database fields for each station were Pass/Fail status, Time in and Time out. A part of the field definitions can be seen in Figure 4-11.

- Product Database Archiving

The product database is archived to keep the size of the database reasonably small for fast queries and updating. The archiving was done on monthly, weekly, daily or hourly basis either automatically or manually. Archived data was saved in a separate database file, which was made available on the network for researchers and investigators interested in analyzing the performance of the TRX assembly line. A part of the archiving mechanism for the product database is shown in the Figure 4-12.

2. Historian Database

The real time process data from each of the stations was stored in the historian database. The historian database was created to improve the quality of the product and the performance of the TRX assembly line through off-line statistical process control (SPC). Continuous online process monitoring facilities are also included such as historical and real-time trend charts and statistical charts, all accessing data from the historian database for optimization of the TRX assembly line.

Database Table Definitions

Tuesday, September 09, 1997

Properties

Date Created: 8/12/97 5:11:06 PM
 Last Updated: 9/4/97 3:46:02 PM

Columns

Name	Type	Size
BPN	Text	26
Allow Zero Length: False Attributes: Variable Length Collating Order: General Column Hidden: False Column Order: Default Column Width: Default Description: Board Product Number Ordinal Position: 1 Required: False Source Field: BPN Source Table: Station number		
BSN	Text	15
Allow Zero Length: False Attributes: Variable Length Collating Order: General Column Hidden: False Column Order: Default Column Width: Default Description: Board Serial Number Ordinal Position: 2 Required: False Source Field: BSN Source Table: Station number		
BRN	
MPN	

User Permissions

Operator Read Data, Insert Data, Update Data, Delete Data
 Admin Operator permissions + Delete, Read Permissions, Set Permissions, Change Owner, Read Definition, Write Definition

Fig 4-11: Example of Database Field definitions

Product Database Archiving

The BridgeVIEW supervisor (System Computer) will be maintaining a "Product DB" (Access DB) to coordinate, monitor, and operate the line. Thus "Product DB" will be constantly polled and updated by the robots on the assembly line. When a product has completed assembly (and passed final test) the record associated with that product is no longer needed and should be archived. Only records of products that have been completed (and passed from the final test station) and records that have been marked "Expired" will be archived and purged from the "Product DB"

"Expired" is marked when = Current Date - "Build Date" (Field in the Product DB) > BridgeVIEW user settable and editable variable (in Days).

Manually deleting individual and sorted records will be allowed via Access front-end while the line is running, and must be proved that there is no conflict between Access and BridgeVIEW via field tests. Manual archiving should be on demand and via push button on the appropriate page.

Automatic archiving should be based on the system clock. The possible selections should be:

- Monthly* selected input the month day (1st to 31st) and time
- Weekly* selected input the day (Sun to Sat) and time
- Daily* selected: input the time
- Hourly* selected input the start time and a pick list for interval (Note: the start time may reset the archive clock if the duration does not divide evenly into 24 hours) and the pick list should support 1, 2, 4, 6, 8, 12, 24

File Storage Name should be user selectable. The file name and path should be in an editable field on-line. Shutting down the system and starting it should not change the file name. Also, during archiving, if the file does not exist create it (any newly created file must include the Field Names), if the file exists, append to it.

Note: The Plant Computer will delete the file after it has been archived

Fig 4-12 Access™ Database archiving procedure

The historian database was developed uses Citadel, a real-time database facility built into the BridgeVIEW™ SCADA package. The process values for each station were assigned tags (the variable name) in BridgeVIEW, their properties configured, and their logging into the database turned ON. The BridgeVIEW engine updates the Citadel database whenever the values are changed for any of the process tags. A part of the historian database requirements is shown in Figure 4-13. The archiving of the Citadel database is done by BridgeVIEW™ engine itself

Historian Tags

Any variables in BridgeVIEW can be logged in the historian. However, only the tags needed for monitoring and analyzing the line should be logged in Citadel.

XX Technology will select a group of tags to be logged based on their insight to the function of the System PC and the Adept robots with their field test of the Adept controllers performance during the debug phase. The idea of monitoring the process based on the different product types is essential for this multi-product line. In Addition, Cell timing tags (number of tags = # Cells X Per Product types) will be logged. These tags measure the product assembly time in each cell and for each product. The resolution for this tag should be in tenths of a second. The calculation is based on product type exit time (stop time) minus entry time (start time) at each cell. Again, each cell will have several logged tags (Different variables) based on product type to get an apples to apples comparison. With this information an engineer can determine whether an improvement is needed and where. Will have to manually navigate to find this timing based on Citadel approach.

Robot performance calculation (number of tags = # Cells) will be logged. The resolution for this tag should be in tenths of a second. The calculation should be = Up Time/ Total Time. The tenths of seconds is acceptable IF the Adept controller can accept these speeds.

Up Time = The cell is not in error (Green Light lit)

Total Time = Time since the counter was reset.

Robot Utilization (number of tags = # Cells) will be logged. The resolution for this tag should be in tenths of a second. The calculation should be = Busy Time/ Total Time (not including any Down Time).

Busy Time = Time the cell was working, actually assembling something.

Total Time = Time since the counter was reset not including any Down time.

Down Time = Time the cells red light was lit.

All timers and counters (up time, busy time, down time, etc.) shall be separate from each other and be capable of being reset one station at a time manually or automatically (up to 6 times a day). No Weekly or Monthly capabilities are needed at this time.

The following will define the light meanings:

Blue Light = Cell idle

Green Light = Cell running

Yellow Light = Operator attention

Red Light = Cell down

Note1 The adept may not be able to handle 125 variable being passed to BridgeVIEW. In addition to the words for the discrete I/O. If this seems to bog down Adept controllers, the variables marked "As Needed" could be taken out.

Fig 4-13: Historian Database requirements

A new archive file is created whenever the database size exceeds more than one megabyte.

- Communication

The communication layer manages the network communication between the System Computer, Adept controllers and the Plant Computer using Dynamic Data Exchange (DDE) over TCP/IP, a standard inter-process protocol for Windows NT. DDE was selected because of its ability to transfer data between two processes running on the same or remote computers by command level execution of programs. The network connections are shown in Figure 4-14

DDE-aware programs communicate by establishing a conversational channel [Wonderware 94]. A client program needing data requests the server program to provide data. "Application" name, "Topic" name and "Item" (variable name) are provided to identify the data being requested. For reading and writing to the Adept controllers, the communication layer uses the BridgeVIEW™ engine, which was configured to support DDE protocol.

- b. Supervisory Software Algorithm.

The finite state machine automaton [Pori 96] was applied in the supervisory control software of the TRX assembly line. The control laws for different stations were defined in the Continuous System Layer of the Hybrid Control model. The complete process flow was divided into different states in the supervisory discrete layer. Each state in the finite state machine automaton was responsible for dealing with events as per control laws governing the process at that particular instant.

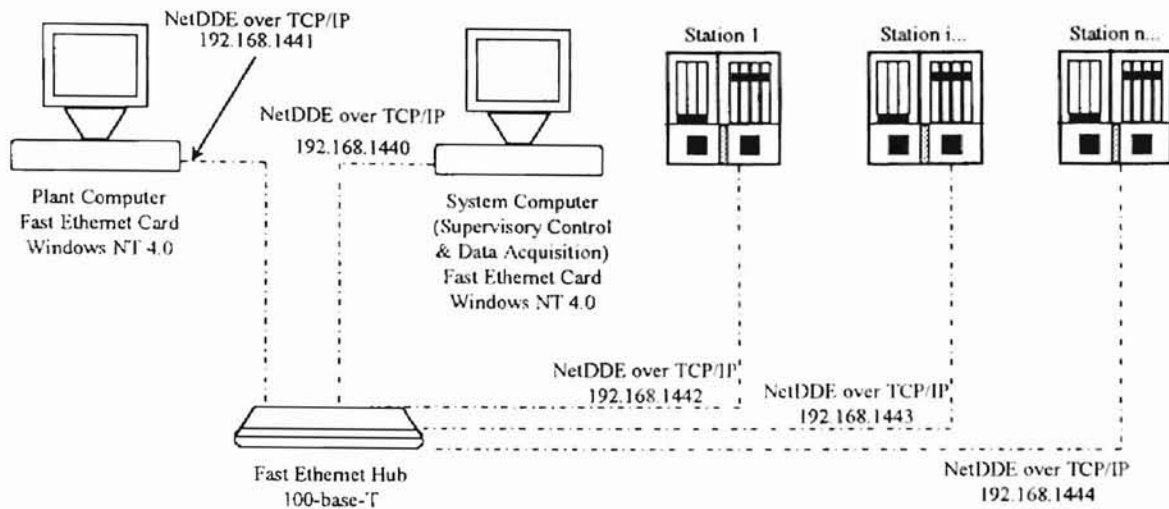


Fig 4-14: Networking & Data sharing Scheme

These control laws were applied at different times or at the same time on different stations, depending on the process flow of the TRX assembly line

c. SCADA Package Selection

The supervisory software architecture of the TRX assembly was implemented using a SCADA package. A SCADA package was preferred over a software language because of several in-built facilities such as password protection, rich graphical user interface, networking and database handling capabilities.

National Instrument's BridgeVIEW™ SCADA package was selected for implementing the TRX assembly line. BridgeVIEW has a built-in object-oriented graphical language, LabVIEW™, which allows programming flexibility within the SCADA package. Sophisticated test and measurement programs can be seamlessly integrated within BridgeVIEW's environment. Moreover programs in graphical languages are easy to write, understand, and debug [Mahmood 96]

These features gave BridgeVIEW an edge over the other SCADA packages considered for this application

CHAPTER V

DESCRIPTION OF STATION 1

A separate software code was developed for each station in the TRX assembly line. Though the algorithm was somewhat similar, the functional requirements of each station were different, warranting separate codes. The module for each station was seamlessly integrated into the main program within the supervisory layer of the TRX assembly line software. Since a similar design approach was taken for all stations, the following description of Station 1 will hopefully give a general idea of the overall approach taken for this project.

5.1 Protocol Description through Eventflow Diagrams

Board flow through a station was depicted with an eventflow diagram, which ties together the events, board locations, and software responses (actions) by the Supervisory Software (System Computer) or Adept Controller. The eventflow diagrams for Station 1 are shown in Figures 5-1a, 5-1b and 5-1c.

The processing in Station 1 begins when a board arrives at the "Queue stop" (see Figure 5-1a). First, the barcode reader reads the serial number of the board to identify the board and sends it to the Adept Controller (AD-1), which signals the board arrival to the Supervisory Software BridgVIEW (BV) by setting the flag "st1.need.readin".

BV, having been polling continuously for this flag to be set, gets the serial number from AD-1 and uses it to retrieve the product and status information of this board (i.e. its

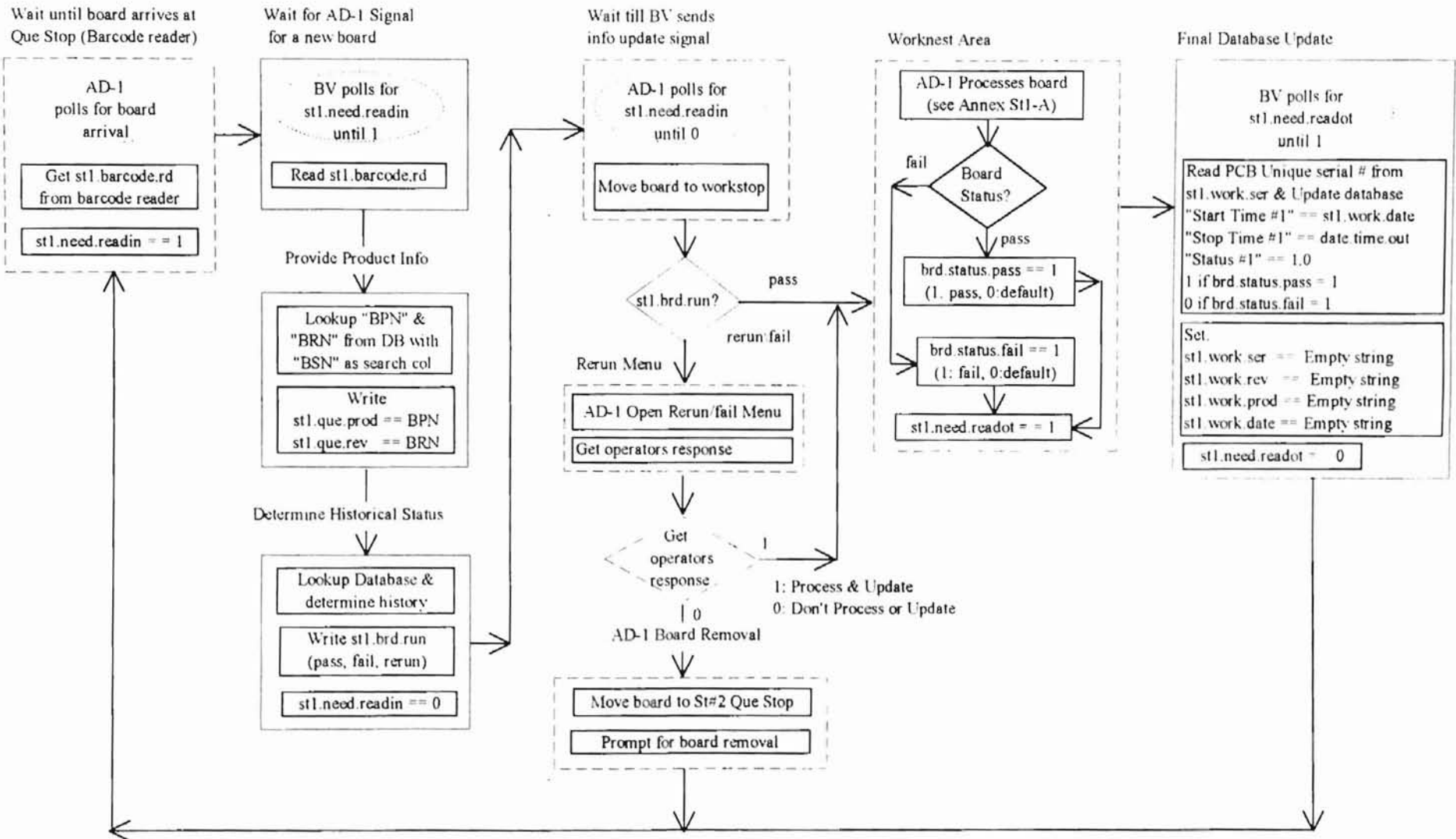


Fig 5-1a. Station 1 Eventflow diagram

previous processing history, if any, with the resulting outcome) from the Product Database (Microsoft® Access97). Using "Board reentry protocol", BV determines the status of the board (rerun, pass, or fail). It also updates database to the fact that this board has been entered into Station 1. At this point, it resets the flag "st1.need.readin" to signal AD-1 that the board status and product information it is waiting for is now ready to be read.

When AD-1 sees that "st1.need.readin" flag has been reset, it moves the board to the "Work stop". It then looks at the board status information provided by BV and starts processing the board if the status is "Pass". Otherwise, it pops out the "Rerun/Fail" menu to get operator input on whether to process or remove the board. If operator chooses to go ahead, it begins processing, else it moves the board to the "Queue stop" of the next station (Station 2) and signals it to remove the board from the assembly line.

A "Pass" board at the "Queue Stop" waits till there is no board in the "Pre Stop" (see Figure 5-1b). As "Pre Stop" gets empty, AD-1 moves the board to the "Pre stop" and also transfers its product information from the "Queue stop" variables to the "Pre stop" variables. At "Pre stop" the board waits for the "clearance signal" until there is no board in the "Work Stop", "Flip Stop" and Robot Arm Gripper. When clearance is granted, the board is moved to the "Work Stop", from where the Robot Arm Gripper picks the board up and puts it in the "Flip Stop" where processing on the board begins. When processing is finished, the Robot Arm Gripper picks the board up and puts it back on the "Work stop". AD-1 accepts no board at the "Work stop" until the previous board completely gets out of the "Work stop".

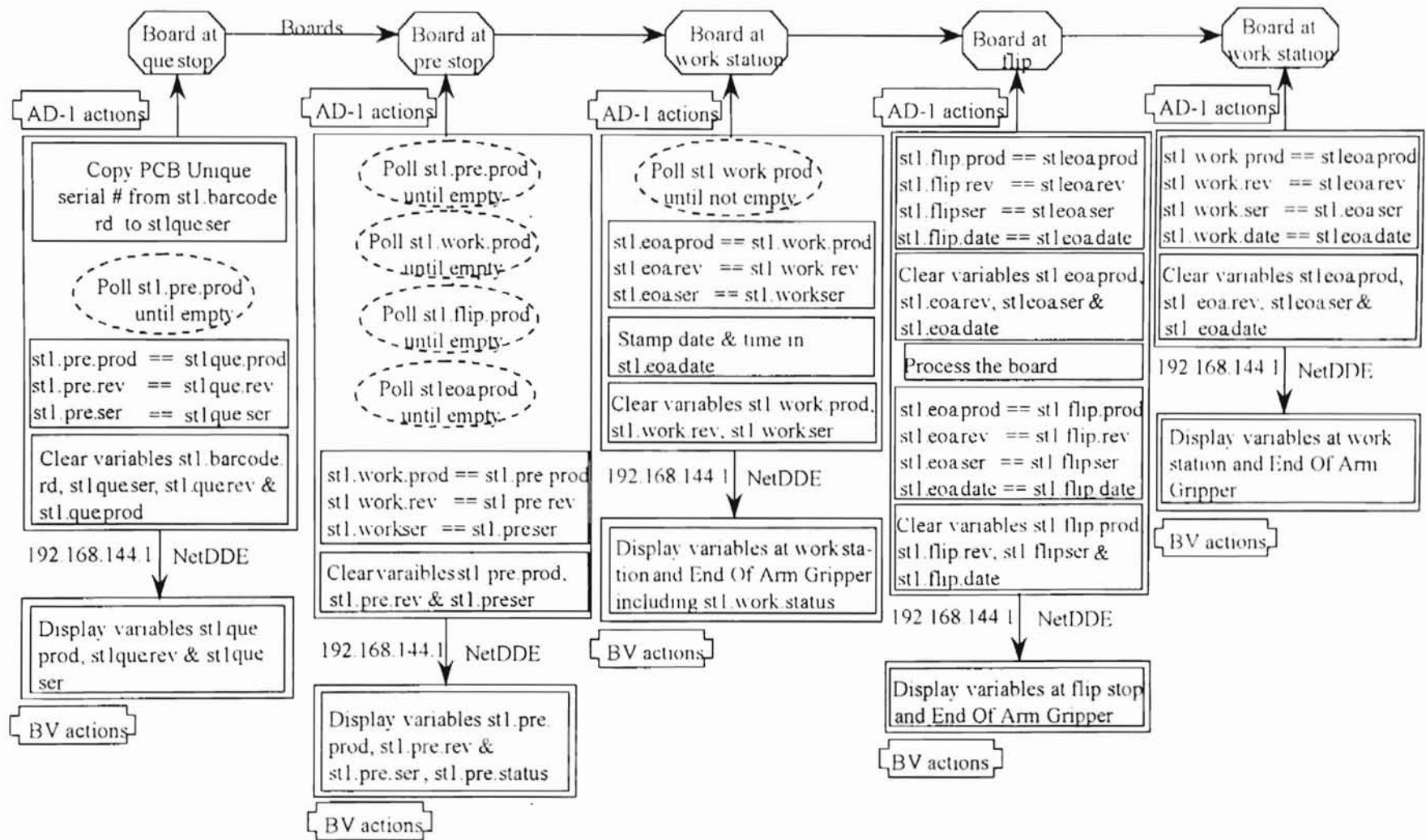
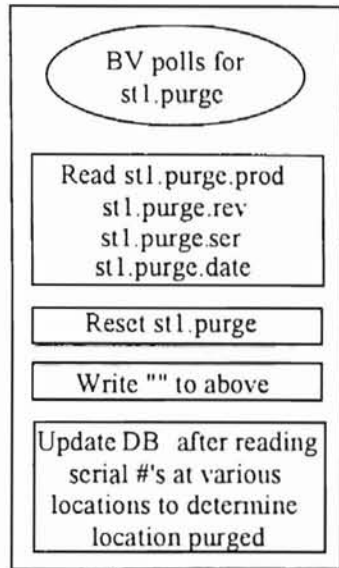


Fig 5-1b. Station 1 Eventflow diagram

BV keeps track of a board through each stop and displays it on demand. AD-1 determines the failure status of the board at the "Work Stop", updates its pass and fail variables accordingly, then sets the flag "st1.need.readot" to signal BV that it has finished processing the board (see Figure 5-1a). Upon getting this signal, BV reads the board's serial number along with the board's process start and stop times and status from AD-1 and updates product database. A separate record is kept by BridgeVIEW, which automatically records every event in its embedded real-time database (Citadel). Additionally, Supervisory Software also stores statistical information in Citadel such as the number of boards processed through Station 1, number of passed and failed boards in Station 1, etc

There are many smaller protocols running in parallel. Supervisory software runs the "Automatic" and "Fixed Board Mode" protocols separately. It continuously watches the flag "st1.fix.local" to determine which mode Station 1 is supposed to be in, i.e., normal or debug mode (see Figure 5-1c). In Purge Protocol, BV monitors the flag "st1.purge". As soon as it is set, BV reads board identification, purge location, and status information from AD-1, updates the product database, then re-initializes AD-1 variables to empty string. BV also coordinates with a remote computer (Plant Computer) to periodically receive information about new boards which will be processed in near future. Plant Computer's software takes operator entries, writes it into an ASCII file, and signals BV that new product data is now available. BV, which is continuously polling for this flag, then reads the file and updates the Product Database with the new board serial numbers and types. BV is also responsible for reading and displaying the user name, messages and current information from various locations inside Station 1.

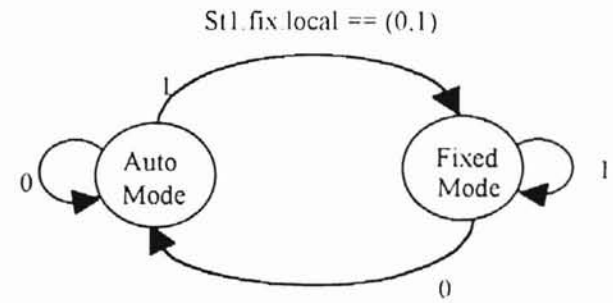
Purge Station #1



Read as needed

User.log.name (110)
User.log.num (111)
Statistical paras (112-119)

Read st1.cur.step st1.error.code st1.run.status	Read blue.lite green.lite red.lite yellow.lite
--	--



States:
Auto Mode: Run on normal mode
Fixed Mode: Debugging mode

61

AD-1 actions

Delete AIM-DB-1 once a week

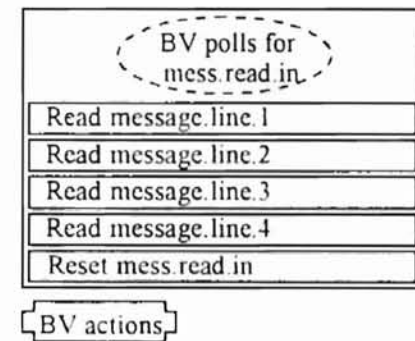
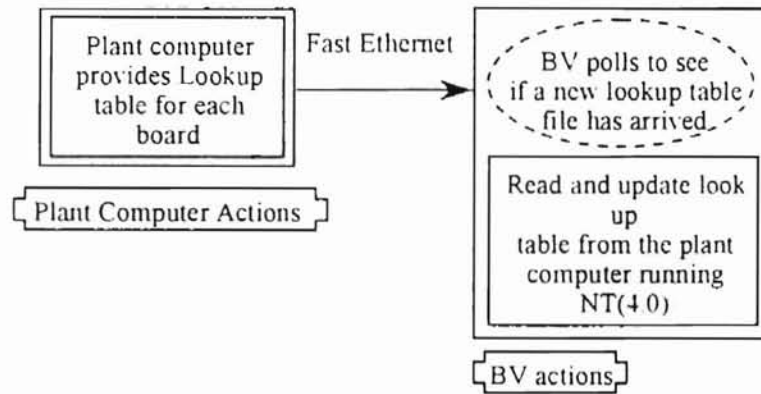


Fig 5-1c: Station 1 Eventflow diagram

5.2 Supervisory Software Hierarchy

The supervisory functions for Station 1 constitute an important part in the layered software architecture described in Chapter IV. The main program is run at the uppermost layer of the TRX assembly line supervisory software. This layer also runs, independent of each other, various subprograms like Log In, Log Out, Robot 1, Robot 2, Historical Trending etc. Most other subprograms run in the second layer including Adept 1, Adept 2, and Adept 3 (the three computers controlling robots) subprograms for all stations (1 to 9) separately. Handshaking protocols, however, are implemented in the third layer for all stations. Interfaces with the product database via SQL Queries and ODBC are in the lowermost layer of the software architecture.

The Station 1 supervisory software runs three subprograms in the second layer of hierarchy under the Robot 1 program. These programs run separately and describe the software actions on the board when the board gets in, gets out and is purged from Station 1. Inside each program, a finite state machine algorithm is applied to implement handshaking protocol (third layer) and database (fourth layer). Station 1 GUI, handshaking protocols layer, database, and communication are described below

1. GUI

The GUI of Station 1 (Figure 5-2) displays information about all boards at various stages of processing. It shows the serial numbers of the boards at each stop, serial number and the type of the board currently in production (at Flip Stop), start and stop times and Pass/Fail status of the last board processed through Station 1. It also shows if the fixed board mode is ON, which would take Station 1 into debugging mode.

Station 1 (Adept Robot 1) General Information

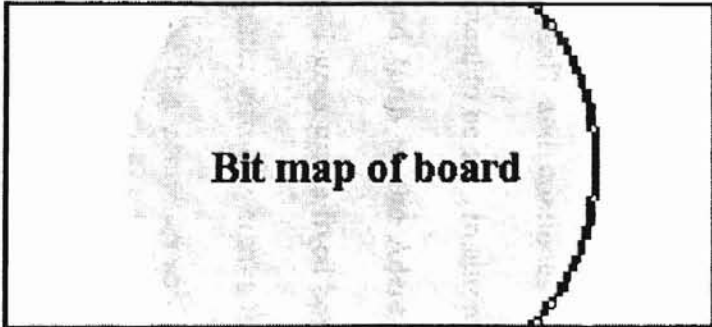
Board Information	Process Flags	Process Statistics
Queue Stop Board <input type="text" value="UA1004BF06"/>	Fix Board Mode <input type="checkbox"/>	Total Proc. <input type="text" value="1403"/>
Pre Stop Board <input type="text" value="UA1004BF45"/>		Passed <input type="text" value="1390"/>
Work Stop Board <input type="text" value="UA1004BF24"/>	Misc. Information Oscillator Tray Position <input type="text" value="0"/> Oscillator Tray Count <input type="text" value="0"/> Connector Tray Position <input type="text" value="0"/> Connector (213/104) Tray Count <input type="text" value="0"/> Connector (209/102) Tray Count <input type="text" value="0"/> Robot Speed (% Full Speed) <input type="text" value="0.00"/>	Failed <input type="text" value="13"/>
Board In Production <input type="text" value="UA1004BF05"/>		% Fail <input type="text" value="0.00"/>
Board Type In Production <input type="text" value="TRX 1.5 Watt"/>		TRX 1.5 W <input type="text" value="100"/>
Board Start Time <input type="text" value="1/6/98 8:00:00 AM"/>		TRX 10 W <input type="text" value="642"/>
Board Stop Time <input type="text" value="1/6/98 8:02:00 AM"/>		TRX 30 W <input type="text" value="345"/>
Board Cycle Time (sec) <input type="text" value="120"/>		DBC TRX 30 W <input type="text" value="202"/>
Pass/Fail Status <input type="text" value="PASS"/>		TRX 1900 MHz <input type="text" value="114"/>
 Bit map of board		Last Reset <input type="text" value="1/4/98 8:15 AM"/>
		Error Information
Board Type <input type="text"/>		Operator's Message <input type="text"/>

Fig 5-2: Station 1 GUI

The GUI also shows the statistical information like total number of boards processed through Station 1, number of boards passed or failed, numbers for different types of boards processed, and the time when this statistical information was last reset. Additionally, it displays several miscellaneous pieces of information such as the parts count in the tray, position of the tray, error information regarding the board in production and operators message. The bit-map of the actual board in production and some other information is blanked out or tampered because of their propriety nature.

2. Handshaking

The handshaking layer of the supervisory software was responsible for implementing the Automatic/Fixed Board Mode, Purge, and Board Re-entry protocols. All stations had separate implementations of these protocols, thus for example, if Station 1 goes into the fixed board mode, System Computer will continue to respond to other Adept controllers and continue operation of other stations.

Figure 5-3 shows the Automatic and Fixed Board protocols for Station 1. In brief, System Computer polls the flag "st1.fix.local" and continues running the line in automatic mode as long as this flag is not found to be set. In automatic mode, System Computer passed the information back and forth to the Adept Controller 1. If, however, the flag is found to be set, Station 1 goes to the fixed board mode, which is basically a do nothing loop. System computer virtually breaks the connection with the Adept Controller, except the polling of the flag continues so that the program can revert itself to the automatic mode protocol upon operator request.

Automatic & Fixed board Mode Protocol

There will be two modes of operation for the TRX line. The most prominent mode being the Automatic mode where the line has direct communication with System Computer. The second method will be the semi-automatic or fixed board mode, which will have no communication with System Computer.

Automatic Mode *Will apply to all stations, 1-9*

The automatic mode will allow System Computer to read and write to the Aim variables at each robot station. System Computer will be able to take information from these variables and track the boards through the line along with storing data to the database and to the historian. System Computer will also be able to provide information to the Adept controllers such as determining what product type to run based on the bar-code information and the pass/fail status from the upstream station.

Fixed Board Mode *Will apply to all stations, 1-9*

The fixed board mode has been developed for debug of the machinery and should be used only for debugging the TRX line and/or newly implemented software. In order for the station to be locked into fixed board mode the engineer will need to go to the fixed board menu and select the proper board to be run. This will have to be done for each station controller. After the board has been selected by the engineer the Adept controller will set the variable stX.fix.brd equal to "1" and will no longer accept information from System Computer. The Adept controller will not read the bar code. The variable stX.fix.brd will be reset to "0" when the engineer has taken the station out of fixed board mode.

Fig 5-3 Station 1 Automatic & Fixed Board Mode protocol

The purge protocol (Figure 5-4) is implemented in the supervisory software's subprogram "Station 1 Purge State Machine". In this protocol, supervisory software updates the product database for the purged board. Adept Controller 1 sets the flag "st1.purge" when a board gets manually purged by the operator of Station 1. The supervisory software, polling for "st1.purge", responds when it is found to be set by reading the purged board's serial number, product number, revision number and purge date from the Adept Controller 1.

Using the serial number as the reference key, the Product Database record for this purged board is updated with the purge date, purge location and failed board status. Each station

Purge Protocol

Station 1 purge

If the robot has tried to pick or place a part unsuccessfully three times an error will be called. The operator will be given a message to either retry to pick, do not retry to pick, reset the pallet, or purge the board location. If the operator chooses to purge the board location a purge menu will pop up on the monitor screen. At this time the operator will be given the option to purge the Adept end of arm, workstation or the flipper station. When the operator presses the purge Adept end of arm, purge workstation or purge flipper station button the Adept system will prompt the operator "Are you sure". If the operators response "Are you sure" is yes then the Adept system will transfer the product number, revision number, serial number, and date/time of the board at that station to the st1.purge.prod, st1.purge.rev, st1.purge.ser, and st1.purge.date variables then set the variable st1.purge equal to "1". This will alert System Computer that a purge has been made. After a board has been taken out via the purge, the board should be marked by System Computer as a failed board. System Computer will then store the purged data and reset the st1.purge.prod, st1.purge.rev, st1.purge.ser, and st1.purge.date variables to null (" ") and the st1.purge equal to "0". If the operator's response "Are you sure" is no then the Adept system would skip to the end of the program and bypass the purging boards. No hot flags would be set for System Computer

Fig 5-4: Station 1 Purge protocol

is given a different purge location number to identify the station from which the purge took place. The supervisory software then resets the flag "st1.purge" to signal Adept Controller 1 to reset itself and start processing the next board

The Board Re-entry protocols, one for each station, determine the status of a board entering into a station. Figure 5-5 shows the Board Reentry protocol for Station 1. It is implemented in subprogram "Station 1 Readin State Machine" of Station 1 supervisory software.

When a board enters Station 1, Adept Controller 1 reads the board's serial number and sets the flag "st1.need.readin". The Station 1 supervisory software polls for this flag, and upon finding it set, queries the Product Database (using board's serial number as search key) for board's status in Station 1 and Station 2. It then runs an algorithm to determine if the board is new or has already run through Station 1

The algorithm writes "Pass" to variable "st1.brd.run" if Station 1 Status field of the Product Database record is empty and Station 2 Status field is not "0". It writes "Rerun" if Station 1 Status field shows "0", or it is empty with the Station 2 Status field showing "0"

The supervisory software resets the flag "st1.need.readin". Adept Controller 1 then reads the variable "st1.brd.run". If the board is "Pass" (new board, never processed before), it begins processing through Station 1. In case of "Rerun", the operator is given a choice via a pop-up menu to take the board out of Station 1 or let it run through

- Synchronization (Critical Timing) Issues

The difference in the speed of execution between the System Computer and the Adept Controllers created the synchronization problems. Code execution at the System Computer was an order of magnitude faster than at the Adept controllers. This necessitated the use of robust handshaking protocols at the software level as well.

The problem is as follows. Let us say Unit A & Unit B want to communicate with each other with handshaking. Normally, a single tag protocol is sufficient. Unit A sets a flag X at Unit B's memory directly, then waits for Unit B to reset its flag X after finishing the assigned tasks. The problem arises when Unit B is much slower than Unit A, in which case there could be a significant time lapse between Unit A's attempt to set flag X and the moment it actually gets set. As Unit A

Board Re-entry Procedure

The following is the proposed board re-entry procedure for Station 1.

Bar-code Reader At	Previous Status of Board Read:	Pass	Fail	Rerun
Station #1	Station # 1 New Board	1		
Station #1	Station # 1 Failed Board			1
Station #1	Station # 2 Failed Board			1

How does the system process good boards.

The bar-code reader at Station 1 will store the serial number in the variable st1.barcode.rd. The Adept will then set the variable st1.need.readin equal to "1" and wait for the System Computer to reset the variable to "0". This will let the Adept know that System Computer read the variables. After the variable st1.need.readin is set to "1" System Computer will read st1.barcode.rd, scan the database and determine if this board has passed Station 1 previously. If the board has not run in Station 1 System Computer will write a "pass" in the variable st1.brd.run, write the product number in the variable st1.que.prod and the revision number in st1.que.rev, then set the st1.need.readin equal to "0". The board will continue to Station 2.

How does the system process bad boards

The operator may re-enter the failed board in front of Station 1. If the operator re-enters the board at Station 1 the bar-code reader will read the serial number and store the data in the variable st1.barcode.rd. System Computer will scan the database and determine if the board has been run at Station 1 or Station 2. Since this board has already been through Station 1, System Computer will write "Rerun" in the variable st1.brd.run. Once the board has traveled from the que-stop (bar-code reader) to the work stop the Adept will pop up a menu and give the operator the choice to rerun this board in Station 1 or to not rerun this board in Station 1. In order for the board to be able to rerun in Station 1 the board must be stripped of all parts initially placed by Station 1. The operator will either press the Rerun Board button or the Do Not Rerun Board button. If the Rerun Board button is pressed the operator will be promoted "Are You Sure". If the "Are You Sure" answer is yes the board will rerun the station. If the "Are You Sure" answer is no then the operator will again be given the choice to rerun the board or to not rerun the board.

If the Do Not Rerun Board button is pressed the operator will be promoted "Are You Sure". If the "Are You Sure" answer is yes the work stop will be lowered and the board will simply pass to Station 2 que-stop. The Adept will write a "0" in the variable st1.opr.res and System Computer will not change the data stored in the database for this board's status in Station 1. At this time a message and amber light will be displayed alerting the operator if he or she wants to run the failed board at Station 2. If the "Are You Sure" answer is no then the operator will again be given the choice to rerun the board or to not rerun the board.

Fig 5-5 Station 1 Board Re-Entry Protocol

begins polling after a fixed interval, it finds flag X to be reset and thinks Unit B is done with the tasks, while in reality the flag was still reset because it never got set due to Unit B's CPU backlog. Even though the possibility is remote, the

consequences for a robotics assembly line could be disastrous as it may try to install parts over existing parts causing a crash.

This problem was resolved by implementing a “Two Flag Scheme”, which is robust albeit complicated. In “Two Flag Scheme” Flag 1 is the original protocol flag while Flag 2 is the acknowledge flag. The protocol flag was set and reset by the Adept Controller while acknowledge flag was set and reset by the System Computer. Since both the Adept Controller and System Computer were not sharing the same flag (protocol flag) and writing to their own flags, the timing conflict was avoided.

The scheme works as follows: When the Adept Controller sets Flag 1, System Computer runs its routine and then it sets Flag 2 to signal completion. While System Computer’s routine is executing, the Adept controller waits for Flag 2 to get set. When Flag 2 gets set, Adept Controller resets Flag 1. Meanwhile, System Computer waits for Flag 1 to get reset. When Flag 1 gets reset, System Computer resets Flag 2, then goes back and starts waiting for Flag 1 to get set. The Adept Controller in the mean time waits for Flag 2 to get reset and after Flag 2 was reset it further processed the board in station #1.

3 Database

The Product Database was handled in the last layer of the supervisory software hierarchy. “Queries” were built in this layer for the Product Database (Microsoft® Access97) using SQL [National Instruments Corp 97b] via ODBC. A sample of Product Database entries for Station 1 is shown in the Figure 5-6. A typical query

BPN	BSN	BRN	MPN	MSN	MRN	Build	Board loc	Start Time #110	Stop Time #110	Status #110
ROA 117 2046/1	UA1004BF01	R1A	KRC 121 10/1	UA1005D301	R4A	97W3	110	1/30/98 13:32:17	1/30/98 13:33:46	1
ROA 117 2046/1	UA1004BF02	R1A	KRC 121 10/1	UA1005D302	R4A	97W3	110	1/30/98 13:33:17	1/30/98 13:34:43	1
ROA 117 2046/1	UA1004BF03	R1A	KRC 121 10/1	UA1005D303	R4A	97W3	110	1/30/98 13:34:43	1/30/98 13:36:16	1
ROA 117 2046/1	UA1004BF04	R1A	KRC 121 10/1	UA1005D304	R4A	97W3	110	1/30/98 13:36:10	1/30/98 13:37:37	1
ROA 117 2046/1	UA1004BF05	R1A	KRC 121 10/1	UA1005D305	R4A	97W3	110	1/30/98 13:37:36	1/30/98 13:39:07	1
ROA 117 2046/1	UA1004BF06	R1A	KRC 121 10/1	UA1005D306	R4A	97W3	110	1/30/98 13:39:03	1/30/98 13:40:35	1
ROA 117 2046/1	UA1004BF07	R1A	KRC 121 10/1	UA1005D307	R4A	97W3	110	1/30/98 13:40:29	1/30/98 13:42:02	1
ROA 117 2046/1	UA1004BF08	R1A	KRC 121 10/1	UA1005D308	R4A	97W3	110	1/30/98 13:41:55	1/30/98 13:43:23	1
ROA 117 2046/1	UA1004BF09	R1A	KRC 121 10/1	UA1005D309	R4A	97W3	110	1/30/98 13:43:22	1/30/98 13:44:52	1
ROA 117 2046/1	UA1004BF10	R1A	KRC 121 10/1	UA1005D310	R4A	97W3	110	1/30/98 13:44:48	1/30/98 13:46:15	1
ROA 117 2046/1	UA1004BF11	R1A	KRC 121 10/1	UA1005D311	R4A	97W3	110	1/30/98 13:46:15	1/30/98 13:47:47	1
ROA 117 2046/1	UA1004BF12	R1A	KRC 121 10/1	UA1005D312	R4A	97W3	110	1/30/98 13:47:41	1/30/98 13:49:15	1
ROA 117 2046/1	UA1004BF13	R1A	KRC 121 10/1	UA1005D313	R4A	97W3	110	1/30/98 13:49:07	1/30/98 13:50:38	1
ROA 117 2046/1	UA1004BF14	R1A	KRC 121 10/1	UA1005D314	R4A	97W3	110	1/30/98 13:50:34	1/30/98 13:52:01	1
ROA 117 2046/1	UA1004BF15	R1A	KRC 121 10/1	UA1005D315	R4A	97W3	110	1/30/98 13:52:00	1/30/98 13:53:31	1
ROA 117 2046/1	UA1004BF16	R1A	KRC 121 10/1	UA1005D316	R4A	97W3	110	1/30/98 13:53:27	1/30/98 13:55:00	1
ROA 117 2046/1	UA1004BF17	R1A	KRC 121 10/1	UA1005D317	R4A	97W3	110	1/30/98 13:54:53	1/30/98 13:56:26	1
ROA 117 2046/1	UA1004BF18	R1A	KRC 121 10/1	UA1005D318	R4A	97W3	110	1/30/98 13:56:19	1/30/98 13:57:46	1
ROA 117 2046/1	UA1004BF19	R1A	KRC 121 10/1	UA1005D319	R4A	97W3	110	1/30/98 13:57:46	1/30/98 13:59:14	1
ROA 117 2046/2	UA1004BF20	R1A	KRC 121 10/2	UA1005D320	R4C	97W3	110	1/30/98 13:59:12	1/30/98 14:00:39	1
ROA 117 2046/2	UA1004BF21	R1A	KRC 121 10/2	UA1005D321	R4C	97W3	110	1/30/98 14:00:39	1/30/98 14:02:05	1
ROA 117 2046/2	UA1004BF22	R1A	KRC 121 10/2	UA1005D322	R4C	97W3	110	1/30/98 14:02:05	1/30/98 14:03:34	1
ROA 117 2046/2	UA1004BF23	R1A	KRC 121 10/2	UA1005D323	R4C	97W3	110	1/30/98 14:03:31	1/30/98 14:05:05	1
ROA 117 2046/2	UA1004BF24	R1A	KRC 121 10/2	UA1005D324	R4C	97W3	110	1/30/98 14:04:58	1/30/98 14:06:29	1
ROA 117 2046/2	UA1004BF25	R1A	KRC 121 10/2	UA1005D325	R4C	97W3	110	1/30/98 14:06:24	1/30/98 14:07:55	1
ROA 117 2046/2	UA1004BF26	R1A	KRC 121 10/2	UA1005D326	R4C	97W3	110	1/30/98 14:07:51	1/30/98 14:09:24	1
ROA 117 2046/2	UA1004BF27	R1A	KRC 121 10/2	UA1005D327	R4C	97W3	110	1/30/98 14:09:17	1/30/98 14:10:44	1
ROA 117 2046/2	UA1004BF28	R1A	KRC 121 10/2	UA1005D328	R4C	97W3	110	1/30/98 14:10:43	1/30/98 14:12:18	1

Figure 5-6. Product database entries for Station 1

contained the source name (alias defined in ODBC to point to the database name), the table name (table within the database source containing targeted data). The “Select” command was frequently used with serial number as the search key to retrieve data from the selected fields (columns). “Update” was another command extensively used, again with serial number as the reference key, along with data and corresponding field names where update had to occur

The process information for all stations was also stored in the Citadel database, along with the statistical information. A tag (variable) in BridgeVIEW was defined for each field of information desired to be stored. Tags are configurable for how to store data in Citadel. The data could be stored continuously, at preset time intervals, upon a change in value, or a combination thereof. The Citadel database was accessed by the second layer programs like “Historical Trending” and “Statistical Process Control” (SPC) to display the station-by-station process and statistical data using charts and graphs

4. Communication

The communication between the Adept Controllers and the supervisory System Computer was established using the BridgeVIEW Engine. The tags (variables) were configured to use DDE as the communication protocol. Example configuration to establish communication with an Adept variable “st1.need.readin” was “Adept|adept1!Aim\trx1\st1.need.readin”, where “Adept”, “adept1”, and “aim\trx1\st1.need.readin” are application, topic, and item names, respectively. About fifty tags were configured for every station to handle alarms, networking,

Tag Name	Data Type	Group	Tag Access	Server	Device	Item	Length	Tag Status
st1_need_readot	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_need_readot	N/A	Good
st1_need_readin	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_need_readin	N/A	Good
st1_blue_lite	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\blue_lite	N/A	Good
st1_green_lite	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\green_lite	N/A	Good
st1_mess_read_in	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\mess_read_in	N/A	Good
st1_red_lite	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\red_lite	N/A	Good
st1_cur_step	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_cur_step	80	Good
st1_error_code	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_error_code	80	Good
st1_purge	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_purge	N/A	Good
st1_run_status	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_run_status	N/A	Good
st1_yellow_lite	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\yellow_lite	N/A	Good
st1_fix_local	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_fix_local	N/A	Good
st1_ack_readin	Analog	Station1	Input/Output	DDE Server	Adept adept1	aim\trx1\st1_ack_readin	N/A	Good
st1_ack_readot	Analog	Station1	Input/Output	DDE Server	Adept adept1	aim\trx1\st1_ack_readot	N/A	Good
st1_ack_purge	Analog	Station1	Input/Output	DDE Server	Adept adept1	aim\trx1\st1_ack_purge	N/A	Good
st1_purge_ser	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_purge_ser	80	Good
st1_purge_date	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_purge_date	80	Good
st1_que_prod	Analog	Station1	Input/Output	DDE Server	Adept adept1	aim\trx1\st1_que_prod	80	Good
st1_que_rev	Analog	Station1	Input/Output	DDE Server	Adept adept1	aim\trx1\st1_que_rev	80	Good
st1_brd_run	Analog	Station1	Input/Output	DDE Server	Adept adept1	aim\trx1\st1_brd_run	80	Good
st1_barcode_rd	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_barcode_rd	80	Good
st1_date_time_in	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\date_time_in	80	Good
st1_date_time_out	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\date_time_out	80	Good
st1_work_ser	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\st1_work_ser	80	Good
st1_brd_status_pass	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\brd_status_pass	N/A	Good
st1_brd_status_fal	Analog	Station1	Input	DDE Server	Adept adept1	aim\trx1\brd_status_fal	N/A	Good

Fig 5-7. A part of BridgVIEW engine configuration file

database, etc. Configuration information about all tags was stored in a single configuration file, part of which is shown in Figure 5-7.

The configuration file is the first to be loaded when BridgeVIEW Engine starts running providing all the information for networking, data logging, alarms etc. The engine, capable of updating thousand tags per second, was configured for specifications like the server input queue size, server output queue size, server shut down time, number of retries for broken connection etc.

5.3 Station 1 Block Diagrams

The Station 1 supervisory software constituted three subprograms. The block diagrams of these subprograms is shown in this section. The common elements in the block diagrams of these subprograms are shown only once.

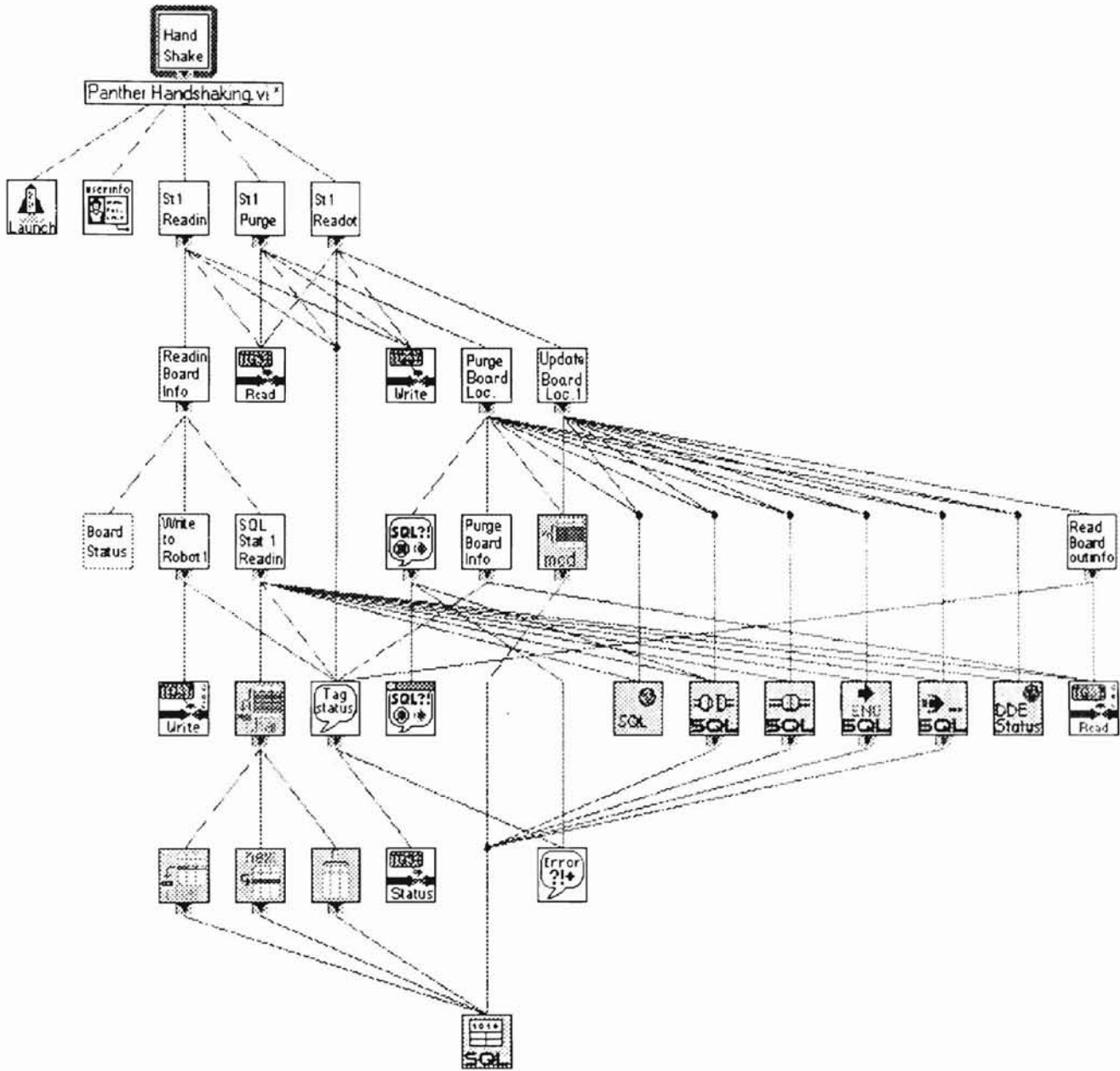


Fig 5-8: Station 1 Hierarchy

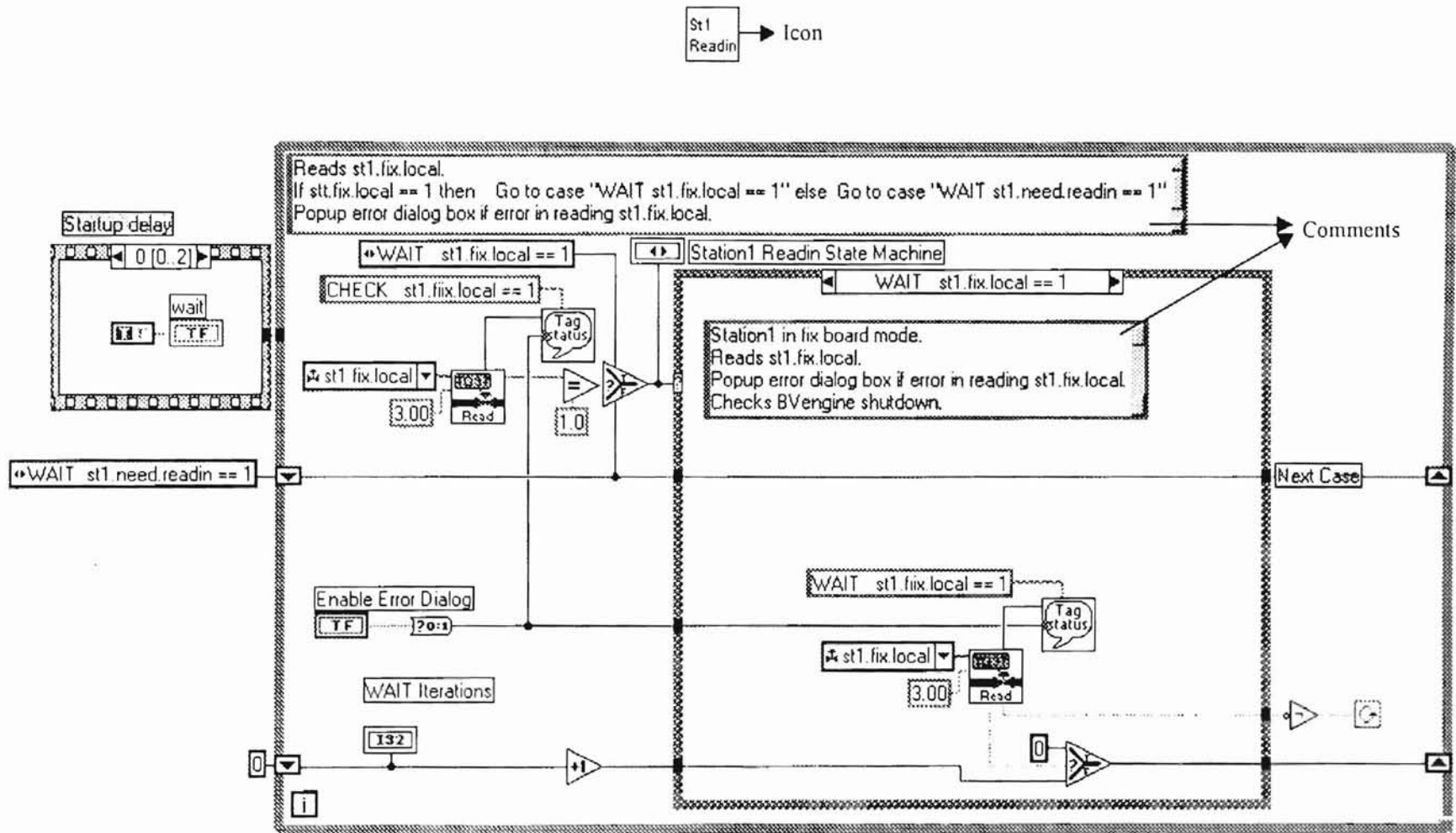


Fig 5-9. Station 1 Readin State Machine Icon & Block Diagram

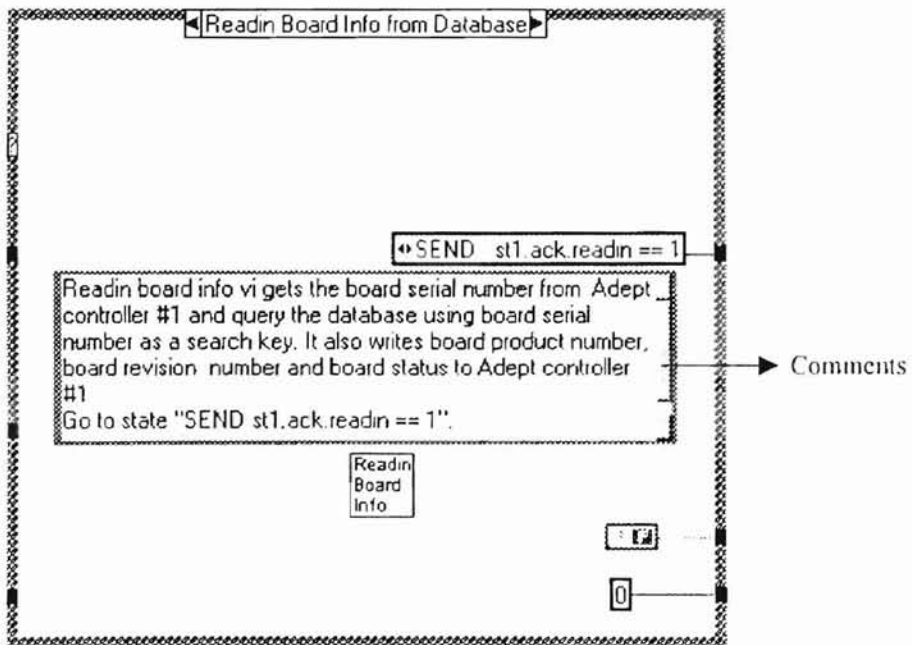
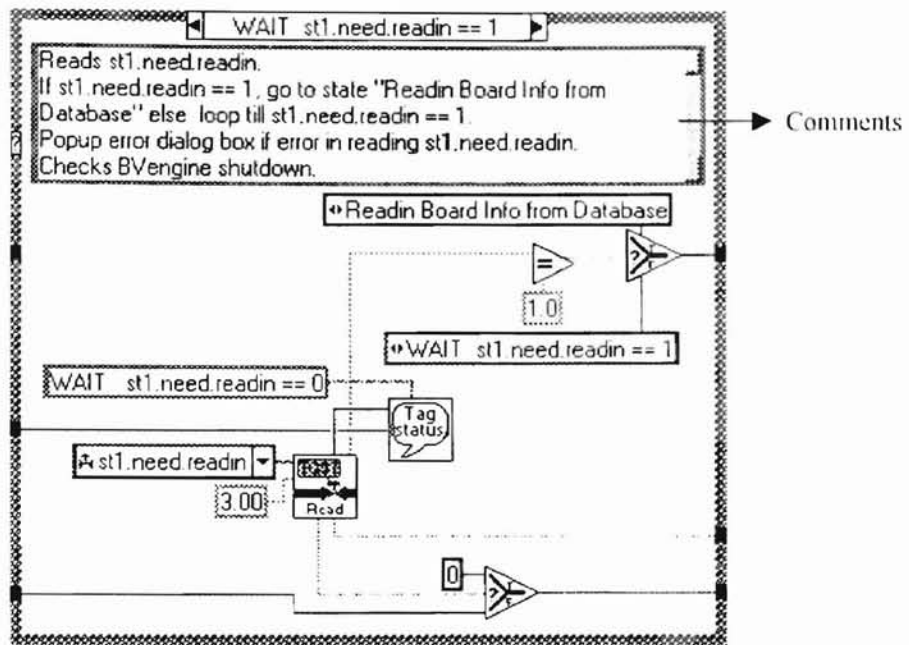


Fig 5-10 States in Readin State Machine

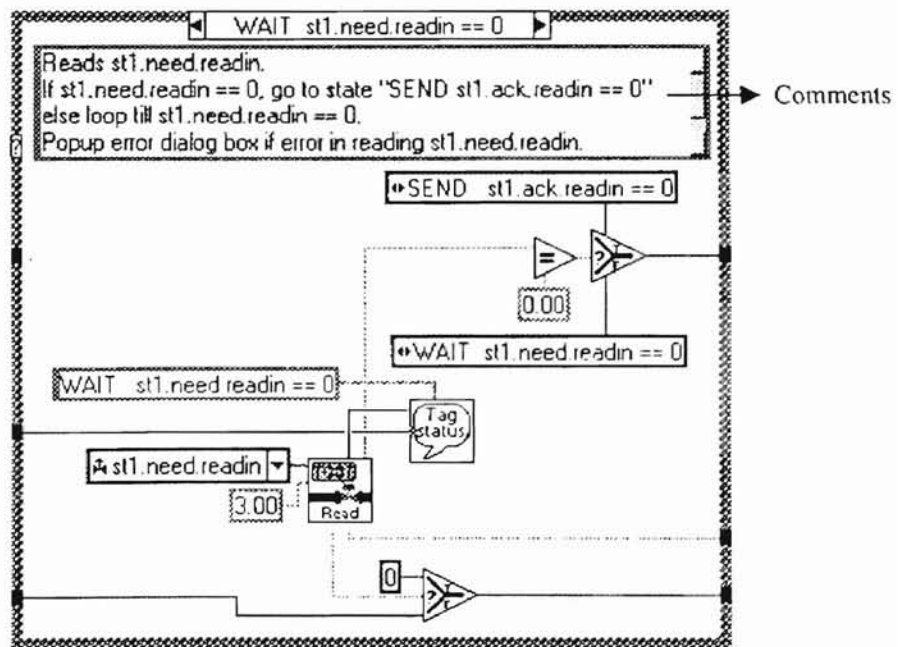
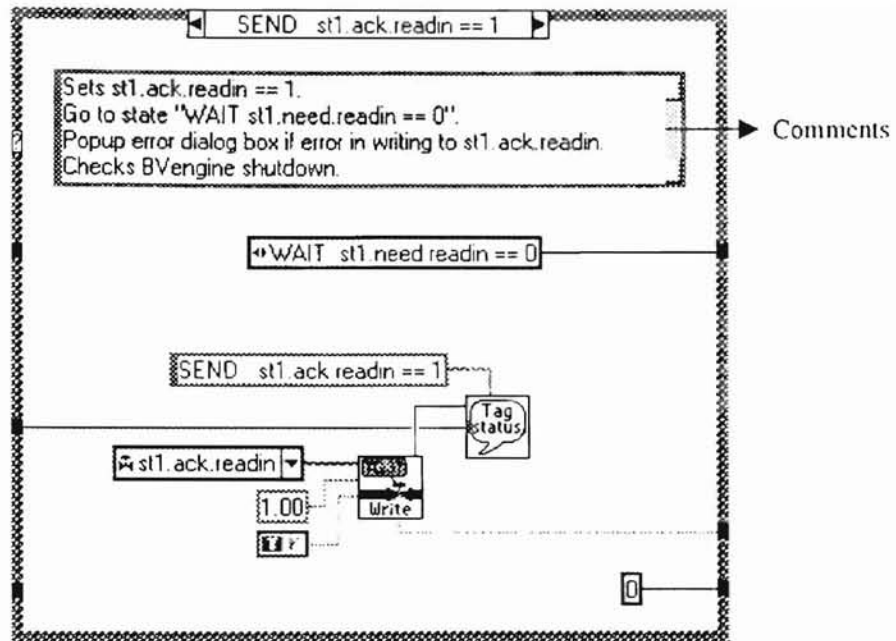


Fig 5-10 (continued): States in Readin State Machine

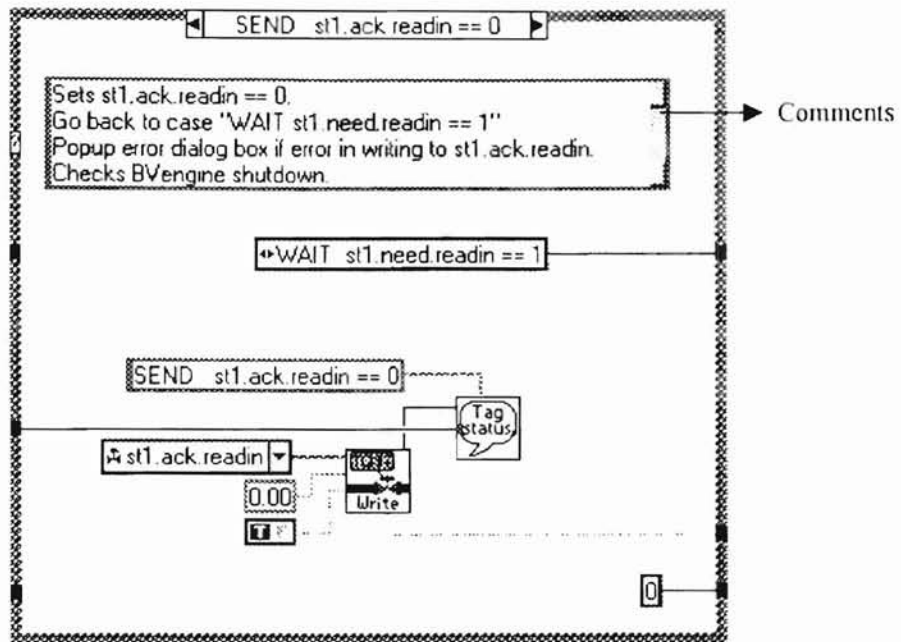


Fig 5-10 (continued) State in Readin State Machine

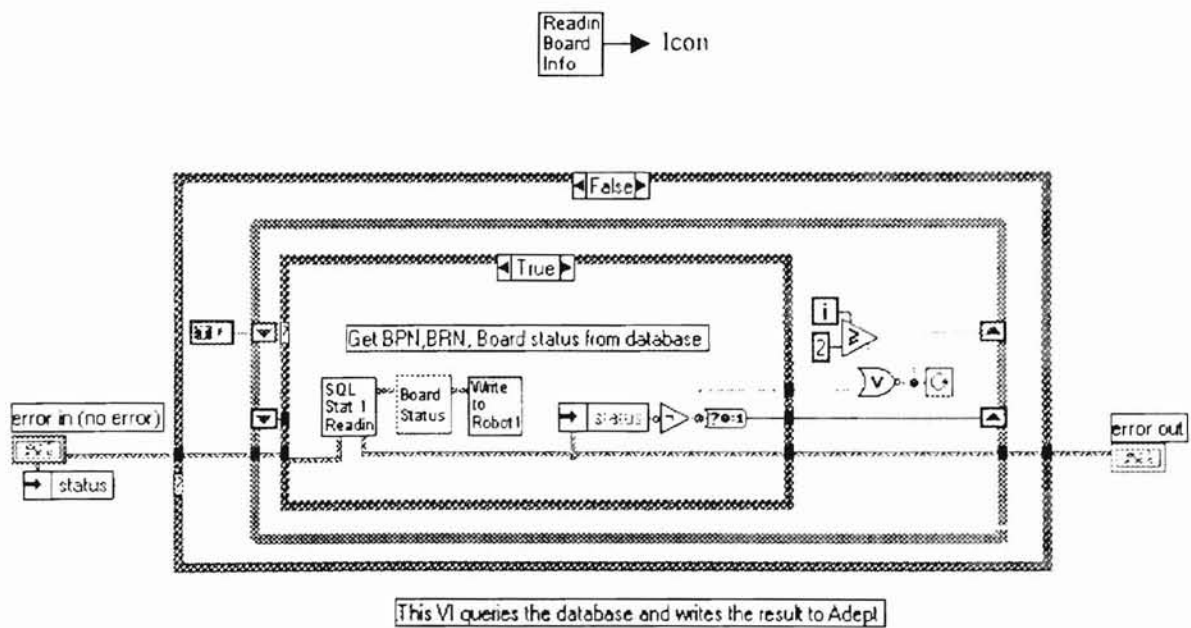


Fig 5-11. Icon and Block diagram of the subprogram in state "Readin Board Info from Database"

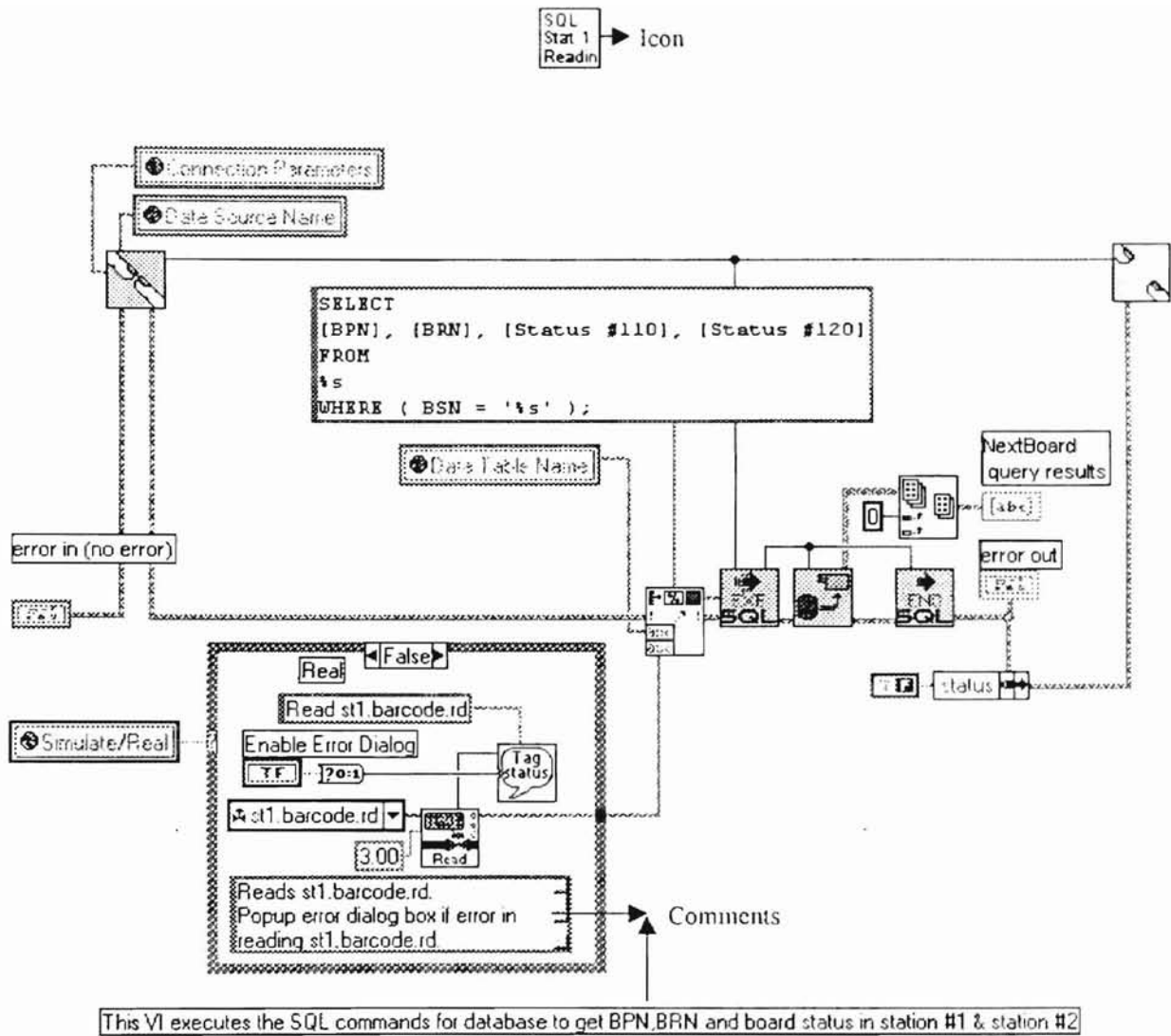


Fig 5-12: Icon and block diagram of a subprogram that queries the database in program Readin State Machine.

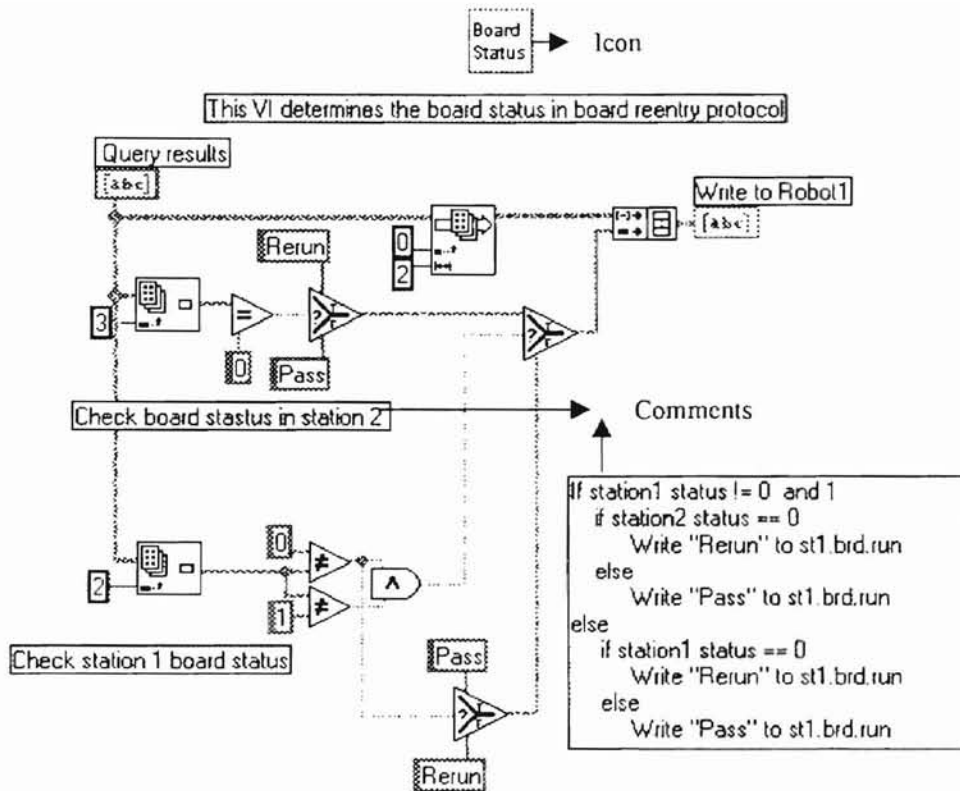


Fig 5-13: Icon and block diagram of a subprogram that determines board status in Board Reentry protocol in program Readin State Machine.

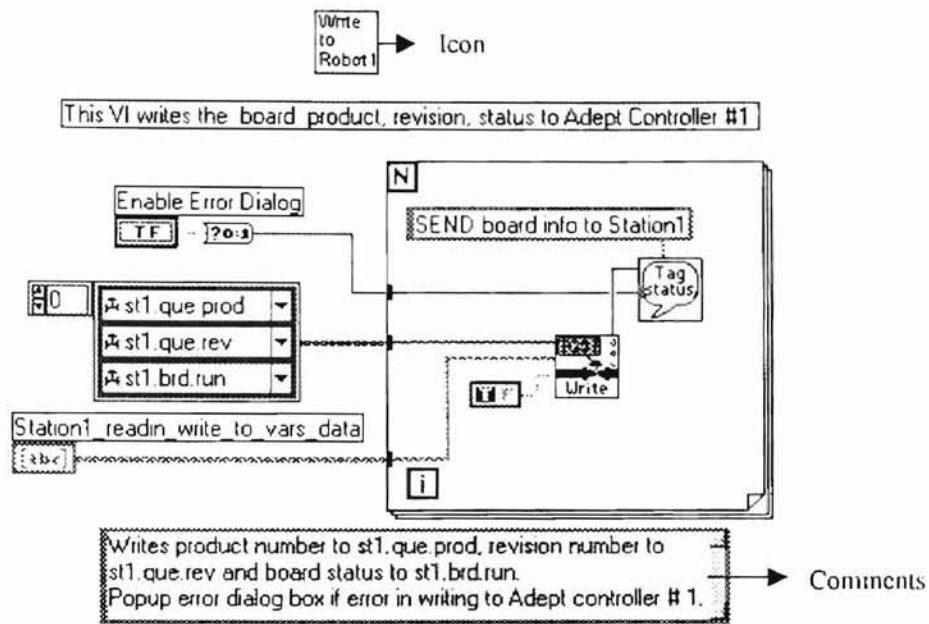


Fig 5-14 Icon and Block diagram of a subprogram that writes to Adept Controller 1 in Readin State Machine program

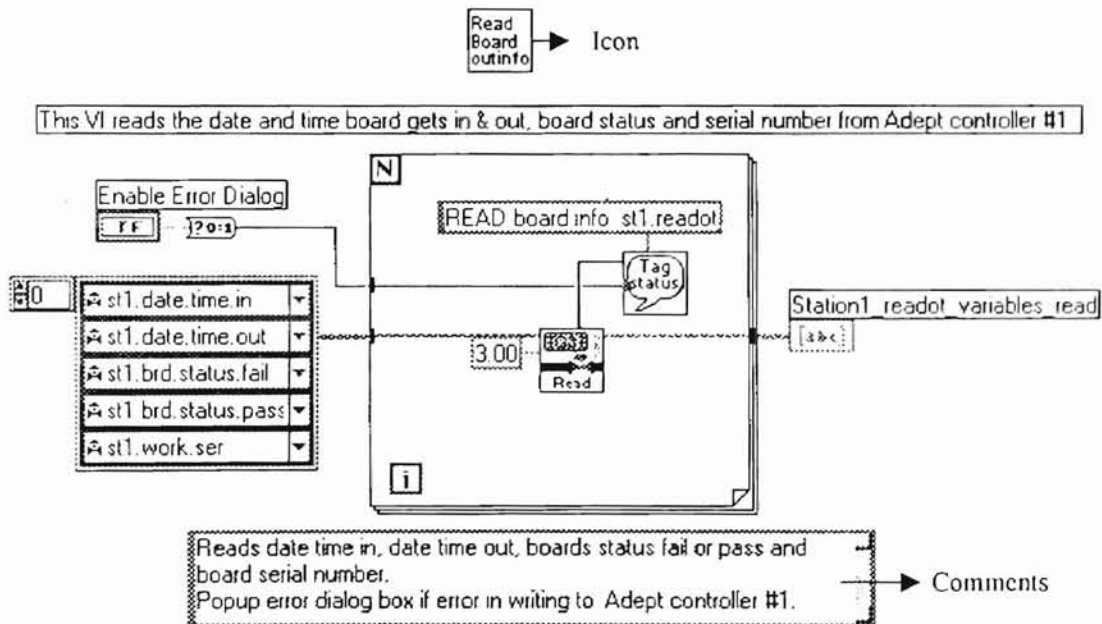
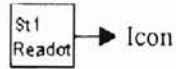


Fig 5-15. Icon and Block diagram of a subprogram that reads from the Adept Controller 1 in Readot State Machine program



82

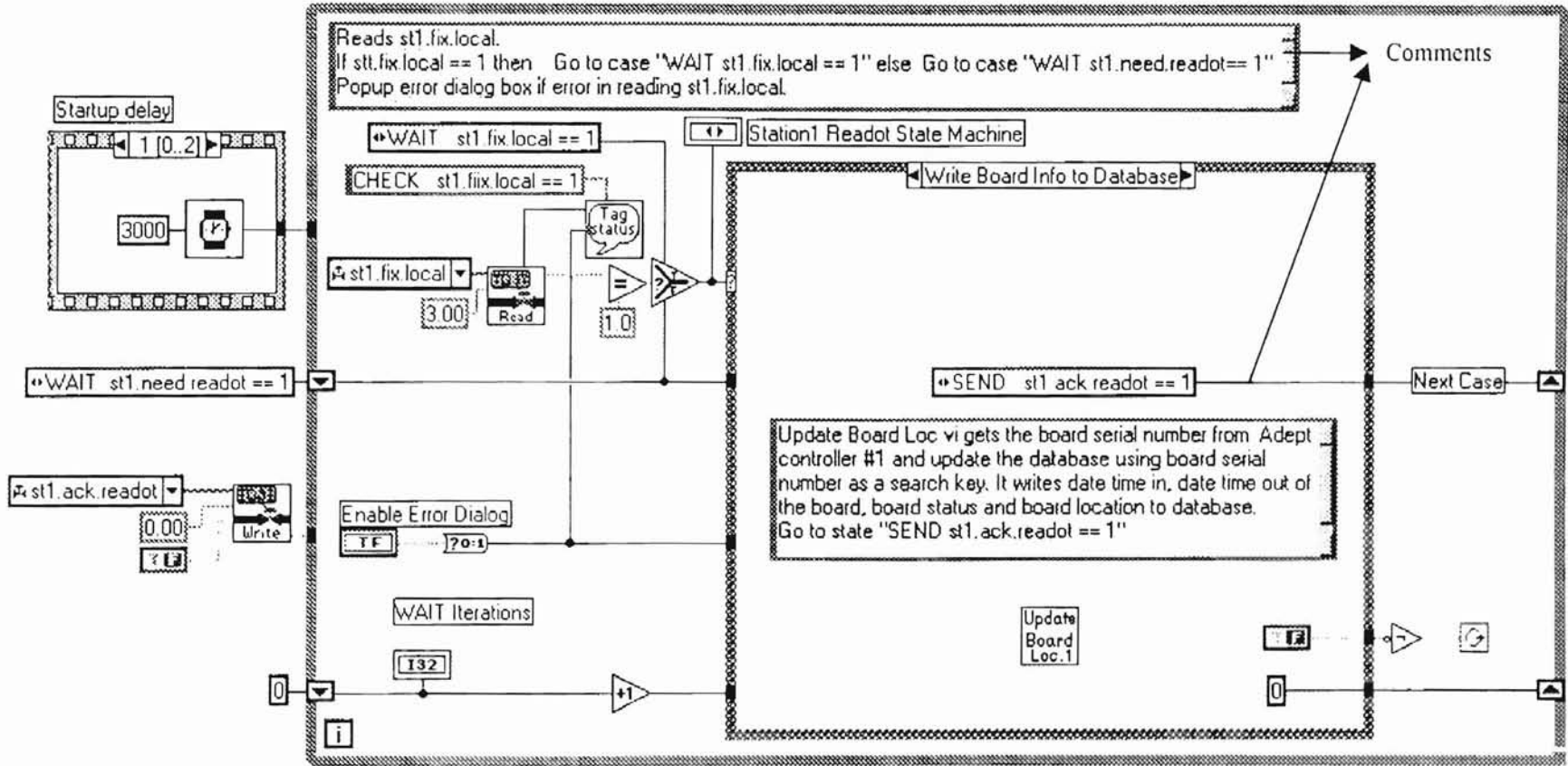


Fig 5-16: Station 1 Readot State Machine Icon & Block Diagram

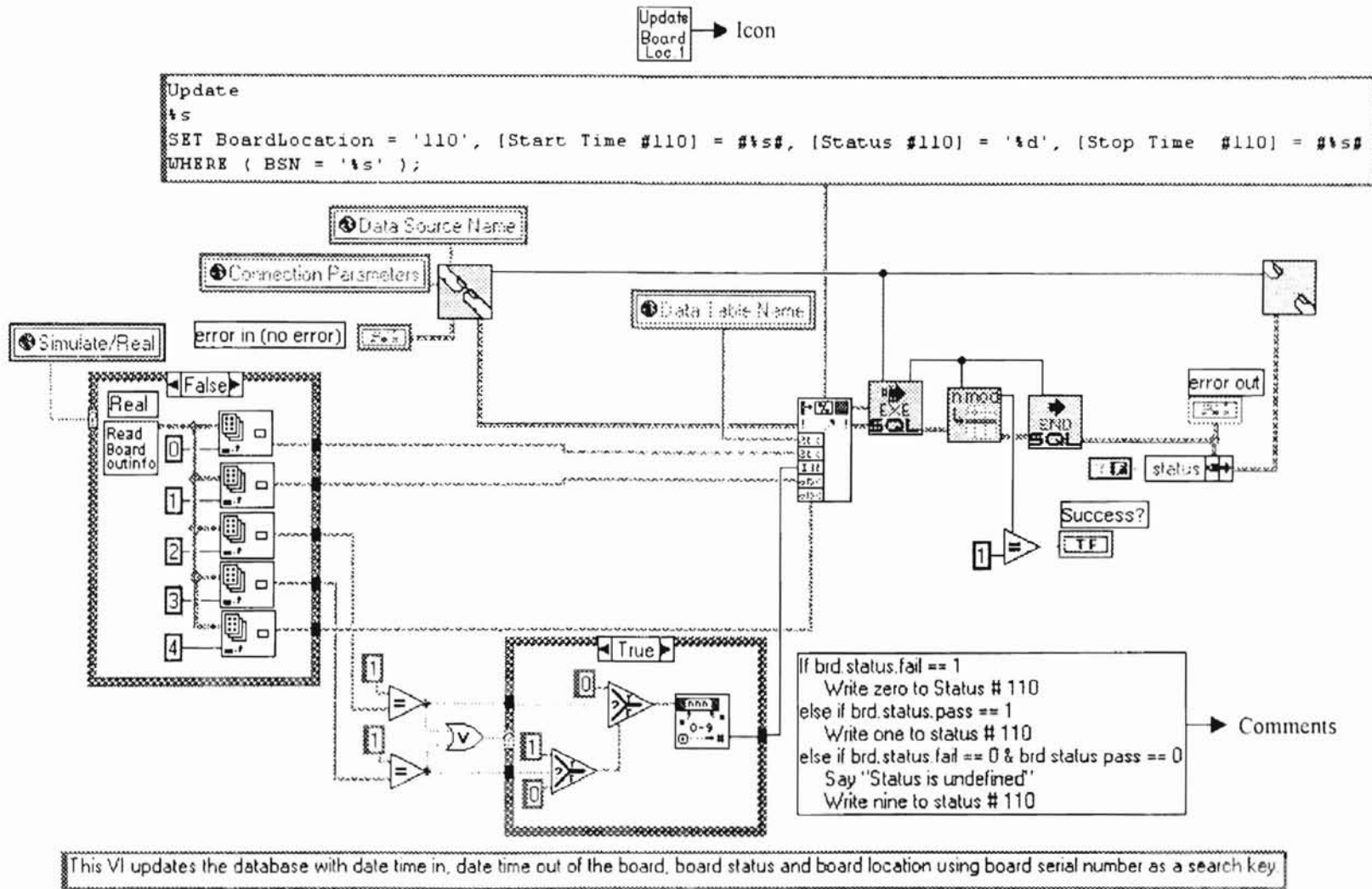


Fig 5-17. Icon and block diagram of a subprogram that updates the database in program Readot State Machine

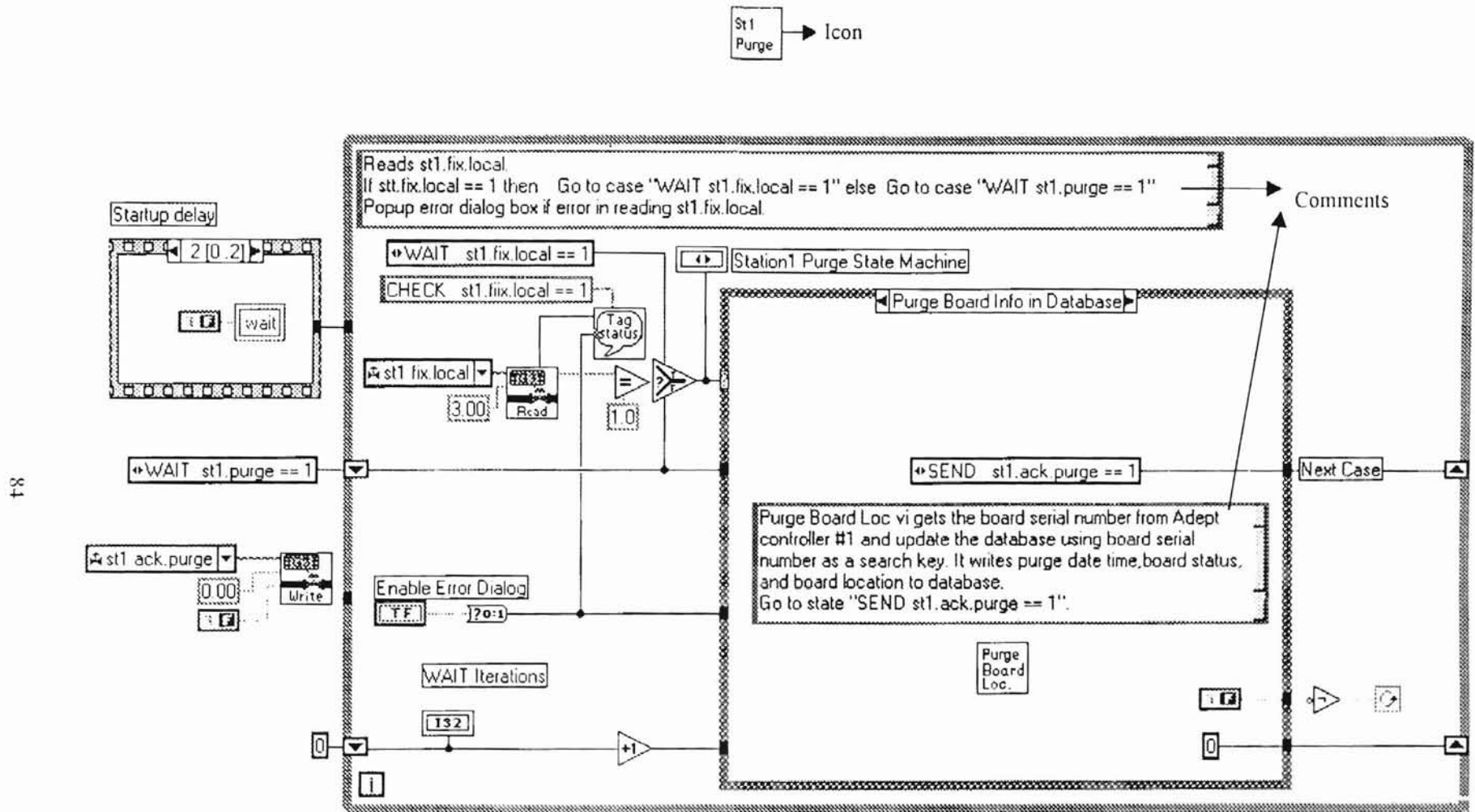


Fig 5-18: Station 1 Purge State Machine Icon & Block Diagram

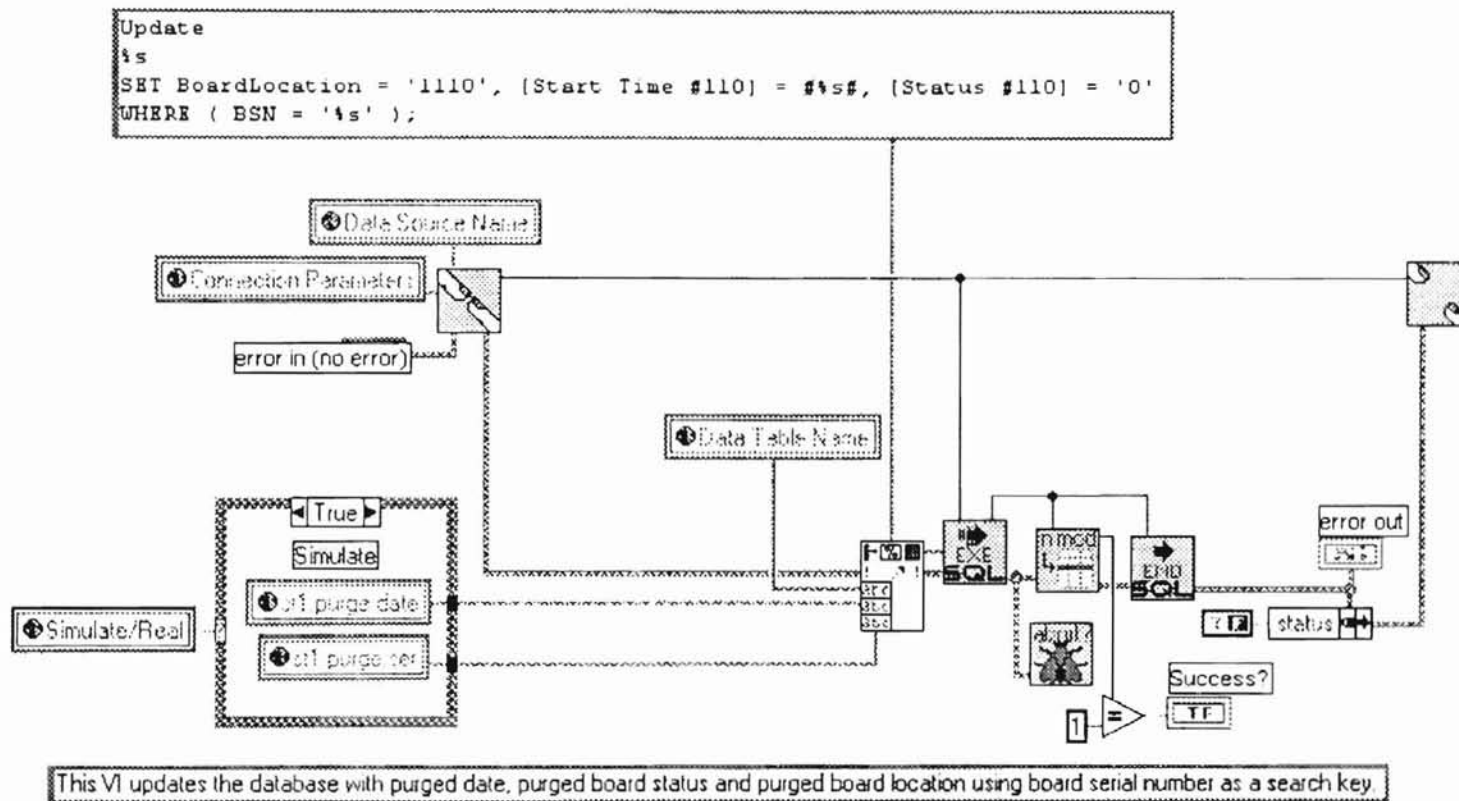
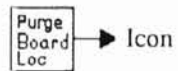


Fig 5-19 Icon and block diagram of a subprogram that updates the database in program Purge State Machine

CHAPTER VI

SUMMARY OF OBSERVATIONS

This report was intended to share my experiences with a Hybrid Control project of an industrial assembly line. It involved understanding, designing, coding, and debugging – almost all phases of the project. A literature review on the Hybrid Control model is included describing the evolution, architecture, and the utility of Hybrid Control models. An example application utilizing Hybrid Control model is also presented. SCADA systems are also briefly described, along a brief survey on the capabilities of three representative commercial SCADA packages.

The system analysis section includes design requirements and specifications. A layered software architecture was developed to implement various services and protocols, such as the Man Machine Interface (MMI) or GUI, communication, network, and database.

Only one station, Station 1, was selected as an example, though the methodology and philosophy is discussed for the entire line. For Station 1, samples of protocols and other engineering documents, including the block diagrams (BridgeVIEW code), are included. Admittedly, it is difficult to understand these documents without knowing the project in entirety, but they give some idea of the kind of activities involved in an industrial automation project of this scope.

The designing phase of the project took 60% of the project time while implementation took only 20%. The rest was in debugging and miscellaneous unaccountable activities such as coordination between our team and the client's team of engineers. Apparently, in

my opinion at least, the dataflow graphical environment of BridgeVIEW made the software development much easier, especially at debugging. The code, in reality, looked very similar to the Eventflow diagrams we had developed. An object was created for each design block, BridgeVIEW functions and structures were copied (all of which are also objects), and these objects were wired together. As compared to the text-based languages I have used, this approach of connecting objects together increases the readability and understandability of the program.

BridgeVIEW also saved a great deal of coding time by using several built-in functionalities. The historical trending charts were handy in displaying the process values of the entire assembly line by accessing data from its built-in real-time database (Citadel). BridgeVIEW engine also handled the I/O functions and communications with Adept Controllers, just by configuring some variables (Tags) to provide needed information. BridgeVIEW provided built-in modularity, a prerequisite for this supervisory program. For example, another station or function can be added just by inserting and wiring one more object to the program.

The performance of the Windows NT 4.0 platform exceeded our expectations. It performed flawlessly and kept running the software for the whole line, regardless of problems in one or more stations.

Of course, there were limitations and tradeoffs. BridgeVIEW™ can only handle about a thousand tags per second, beyond which its reliability is not guaranteed. Future additions, which certainly will occur, will have to work around this limitation.

The supervisory software did not have exception handling capabilities. For example, there were no provisions to react properly in cases of emergency shutdowns or power failures. The product database is currently available to local users only; it should be kept in a database server to give engineers or managers global access for real-time evaluation of line's production performance.

REFERENCES

- [**Boyer 93**] Stuart A Boyer, *SCADA Supervisory control and data acquisition*, Instrument Society of America, Research Triangle Park, NC, 1993.
- [**Deshpande and Varaiya 95**] Akash Deshpande and Pravin Varaiya, "Viable Control of Hybrid Systems", *Hybrid Systems II*, Lecture Notes in Computer Science 999, pp. 128-147, Springer-Verlag, Berlin Heidelberg, Germany, 1995
- [**GE Fanuc Automation 95**] GE Fanuc Automation, *Cimplicity[®] For Windows NT[™] Monitoring and Control Software Data Sheets*, Part No. GFT-168, Charlottesville, VA, Oct 1995
- [**GE Fanuc Automation 96**] GE Fanuc Automation, *Cimplicity[®] Monitoring and Control Products MMI & MES/SCADA*, Part No. GFW-0039, Charlottesville, VA, May 1996
- [**Godbole, Lygeros and Sastry 95**] Datta N Godbole, John Lygeros and Shankar Sastry, "Hierarchical Hybrid Control: A case study", *Hybrid Systems II*, Lecture Notes in Computer Science 999, pp. 166-190, Springer-Verlag, Berlin Heidelberg, Germany, 1995
- [**La Fauci 97**] Joseph La Fauci, "Users' demands narrow PLC-DCS gap", *InTech*, pp 36- 40, Feb 1997.
- [**Lygeros 98**] John Lygeros, *Hybrid System control*, <http://robotics.eecs.berkeley.edu/~lygeros/Research/hybrid.html>, (Jan 1998).
- [**Mahmood 96**] Syed M Mahmood, *General-Purpose Automation Programming Using A Graphical Language*, Masters Thesis, Oklahoma State University, Stillwater, Oklahoma, May 1996
- [**National Instruments Corp 96**] National Instruments Corporation, *BridgeVIEW[™] User Manual*, Part No. 321294A-01, Austin, TX, Oct 1996.
- [**National Instruments Corp 97a**] National Instruments Corporation, "Historical Trending", *BridgeVIEW[™] The Graphical Programming Approach to PC Automation*, Part No 350313A-01, Austin, TX, Feb 1997.
- [**National Instruments Corp 97b**] National Instruments Corporation, *BridgeVIEW[™] and LabVIEW[®] SQL Toolkit for G Reference Manual*, Part No. 321525A-01, Austin, TX, Feb 1997.

[**National Instruments Corp 97c**] National Instruments Corporation, "BridgeVIEW Architecture", *The Graphical Programming Approach to Industrial Automation*, Technical Seminar Series, Part No. 350331A-01, pp 9, Austin, TX, Mar 1997.

[**National Instruments Corp 97d**] National Instruments Corporation, "OLE for Process Control", *Industrial Solutions Using Advanced PC Technologies*, Technical Seminar Series, Part No 350368A-01, pp 12-15, Austin, TX, Aug 1997.

[**Pori 96**] Anuj Pori, *What are Hybrid Systems?*,
<http://www-path.eecs.berkeley.edu/~anuj/what-are-hybrid/what.html>. (Sep 1996)

[**Robinson and Salkas 95**] John J. Robinson and John P Salkas, "DCS vs. PLC: Why not a hybrid?", *InTech*, pp 40-43, Jul 1995

[**Tanenbaum 96**] Andrew S Tanenbaum, *Computer Networks*, Prentice Hall, Upper Saddle River, NJ, 1996.

[**Warnock 88**] Ian G Warnock, *Programmable Controllers Operation and Application*, University Press, Cambridge, UK, 1988

[**Wonderware 94**] Wonderware Corporation, *NetDDE Product Data Sheet*, Part No 15-306, Irvine, CA, July 1994.

[**Wonderware 95**] Wonderware Corporation, *InTouch for Process Automation*, Part No 15-309, Irvine, CA, Oct 1995

[**Wonderware 97**] Wonderware Corporation, *Visualization InTouch™ 7.0*, Part No 15-7000, Irvine, CA, Nov 1994

VITA

Syed M. M. Manzoor

Candidate for the Degree of

Master of Science

Thesis: BRIDGEVIEW™ FOR HYBRID CONTROL: AN APPLIED CASE STUDY
OF AUTOMATING AN INDUSTRIAL ASSEMBLY LINE

Major Field: Computer Science

Biographical:

Personal Data: Born in Karachi, Pakistan, May 15, 1970, son of Syed M. Usman and Anis Fatima.

Education: Received Bachelor of Engineering degree with major in Electronic Engineering from NED University at Karachi, Pakistan in 1992. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in July 1998.

Experience: Worked as Instrumentation Engineer in Gatron (1992-1994), a Ryan manufacturing and texturizing plant; Computer Science intern at BDM-Oklahoma Inc. (May, 1996- Aug, 1996); Computer Science intern at VI Engineering, North Carolina (Currently working since Jan 1, 1997).

Professional Membership: Engineering Council of Pakistan.