To the Graduate Council:

I am submitting herewith a thesis written by Brenda H. Barr entitled "Software Design for Real-Time Supervisory Control." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Seung-Chul Lee, Major Professor

We have read this thesis
and recommend its acceptance:

Bruce W. Bowan

Donald W. Roberds

Accepted for the Council:

Associate Vice Chancellor
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Brenda H. Barr

Date April 22, 1992

SOFTWARE DESIGN FOR REAL-TIME SUPERVISORY CONTROL

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Brenda H. Barr

May 1992

DEDICATION

To my loving parents, Annadell Ray Howard, whose prayers and encouragement have carried me over rough terrain, and the late Joe Lee Howard, whose confidence in me never wavered. Thank you for the wonderful gift of education.

ACKNOWLEDGMENTS

My committee chairman, Dr. Seung-Chul Lee, has my utmost gratitude for his superior guidance and infinite patience extended to me during the preparation of this thesis. Dr. Bruce Bomar has my sincere appreciation for his valuable counsel and significant contributions as a member of my committee. I am very grateful to Dr. Don Roberds for his comments and committee service.

A special acknowledgment goes to my friend and colleague, Mr. Wayne Belew. His tutelage over a course of several years has been the most significant contributing factor to my present knowledge of real-time system development. My deep appreciation is extended to my immediate supervisor, Mr. James Shelton, for whom I have had the privilege of working for the past ten years. I would not have been able to complete this endeavor without his long-standing support, patience, and encouragement. My thanks also go to Calspan Corporation for their generous support of higher education for their employees.

Not to be overlooked is my profound gratitude to my husband, Tom, for his reinforcement, consolation, and most of all for his patience during the seemingly endless quest of preparing this thesis.

ABSTRACT

Real-time software systems usually have more severe requirements and operate in more critical environments than conventional software systems. As a result, their development requires particular attention to certain aspects of the software that are not considered in traditional software development. While contemporary literature on the development of software systems offers meaningful insights for ordinary systems, it often advocates practices that are contrary to effective real-time systems. This thesis will address some of the issues that are germane to the development of real-time software systems and present a detailed example of such by describing a real-time supervisory control system that was developed and implemented at Arnold Engineering Development Center.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	
	1.1 Real-time Software Systems	1
	1.2 Overview	4
II.	REAL-TIME SUPERVISORY CONTROL APPLICATION	
	2.1 Application Background	6
	2.1.1 Plant and Compressors	7
	2.1.2 Surge Description	9
	2.1.3 Previous Surge Control	12
	2.1.4 Compressor Surge Study	14
	2.2 Surge Control System	16
	2.2.1 Valve Modification	19
	2.2.2 Controllers	19
	2.2.3 Signal Selections	26
	2.2.4 Manual Control Console	28
	2.2.5 Supervisory Computer	28
	2.3 Computer System	32
	2.3.1 Computer	33
	2.3.2 Terminal(s)	39
	2.3.3 I/O Subsystem	41
	2.3.4 Software	48
III.	SYSTEM REQUIREMENTS	
	3.1 System Analysis and Specifications	58
	3.2 Application Software Requirements	64
IV.	SOFTWARE SYSTEM DESIGN AND DEVELOPMENT	
	4.1 Design Considerations	73
	4.2 Application Preliminary Design	82
	4.2.1 Defining Data Storage	83
	4.2.2 Defining Programs	93
	4.3 Application Detailed Design	101
	4.3.1 Initialization	104
	4.3.2 Analogic I/O	105
	4.3.3 Controller Communications	107
	4.3.4 Data Validation and Alarms	110
	4.3.5 Data Conversion	115
	4.3.6 Valve Movement	116
	4.3.7 Function Control	118
	4.3.8 User Interface	121

4.3.9 Data Display 124
4.3.10 Data Base Transfers 125
4.3.11 Recording Data, Alarms, and Events 126
4.3.12 Formatting Files 129
4.4 Implementation and Testing 130
4.4.1 Testing Phases and Observations .. 132
4.4.2 Application Testing 136

V. CONCLUSIONS

5.1 System Evaluation 139
5.2 Recommendations for Future Improvements . 142

LIST OF REFERENCES 145

APPENDICES 150

A. Figures 151
B. I/O Subsystem Parameter List 207
C. Data Dictionary for the Software Specification Data Flow Diagram .. 212
D. Controller Constants File 216
E. Data Base File 218

VITA 225

LIST OF FIGURES

Figure		Page
1.	1-Stage Special Type B Configuration	152
2.	Tank Charge Configuration	152
3.	1-Stage Type B Configuration	153
4.	2-Stage Configuration	153
5.	3-Stage Configuration	154
6.	4-Stage Configuration	154
7.	5-Stage Configuration	155
8.	7-Stage Configuration	155
9.	9-Stage Configuration	156
10.	Typical Compressor Map	157
11.	Surge Control System	158
12.	Typical Axial Compressor Control Instrumentation	159
13.	Compressor Map with Antisurge Controller Limits	160
14.	Antisurge Controller Limit Lines for PI Circuit	161
15.	Compressor Operating Point to the Right of Surge Control Line	162
16.	Compressor Operating Point to the Left of Surge Control Line	163
17.	Compressor Operating Point on Recycle Trip Line	164
18.	Compressor Operating Point to the Left of Safety On Line	165
19.	22-Bit Memory Mapping	166

20. Virtual to Physical Address Mapping 166

21. Memory Management Registers 167

22. Memory Management Fields of a Virtual Address 168

23. Displacement Fields of a Virtual Address 168

24. Page Address Register 169

25. Page Description Register 169

26. Virtual to Physical Address Translation 170

27. ANDS5400 I/O Subsystem 171

28. ANDS5400 16-Bit Analog Address 172

29. ANDS5400 16-Bit Digital Address 172

30. ANDS5400 Q-Bus Interface Operating Modes 173

31. ANDS5400 Q-Bus Interface Registers 173

32. ANDS5400 Q-Bus Interface Control and Status Register 174

33. Task Mapping to RSX Executive 175

34. Response Time Versus Resource Demand 176

35. Data Flow Diagram Symbols 176

36. Data Flow Diagram of Main Phases of System Development 177

37. Sub-Level Data Flow Diagram of System Analysis Phase of System Development 178

38. Software Specification Data Flow Diagram 179

39. Specification of the Temperature Alarm Function 181

40. Preliminary Design Data Flow Diagram 182

41. System Data Flow Diagram 185

42. Off-Line Menu 188

43. Valve Function Menu 189

44. Print and/or Display Data Menu 190

45. On-Line Menu 191

46. I/O Subsystem Readings Menu 192

47. Data Base and System Constants Function Menu 193

48. Display and/or Edit System Constants 194

49. Display and/or Edit Controller Constants 195

50. Display and/or Edit Data Base 196

51. Servo Amplifier Power Readings 197

52. First Page of Pre-Operation Report 198

53. Second Page of Pre-Operation Report 199

54. Computer and Controller Override States 200

55. Select Valves for Span and Zero Function 201

56. Check Ambient Pressures and Temperatures 202

57. On-Line Status Display 203

58. Inlet Temperature Readings 204

59. Select Valves, Positions, and Ramp Time for Position Valve(s) Function 205

60. Relationship Between Testing Phases and Software Development Phases 206

CHAPTER I

INTRODUCTION

1.1 REAL-TIME SOFTWARE SYSTEMS

Real-time software systems are subject to the same objectives of good software design as other types of software systems, such as employing structured design methodologies and striving for a product that is modular, user-friendly, and easily maintained. They have additional requirements and constraints however that other types of software systems do not have. But before elaborating on what makes real-time software different, the term "real-time" could use some clarification.

The term "real-time" has been used, especially in the past, as a synonym for "timesharing" and even "online", which is an abuse of the term [1]. Real-time processing is indeed online processing but it may or may not use timesharing. Yet the term "real-time" has significance beyond and apart from the other terms. Real-time can be defined as the ability to process input data and respond to external events fast enough to permit the computer's results to influence the related external process within the required time interval [2,3].

Non-real-time applications do not have to be concerned about missing the occurrence of a change in their

environment. Software that generates a report from stored data or that performs number-crunching operations are examples of such applications. Real-time applications, on the other hand, must accommodate the rate of change in their environment. The rate of change that must be accommodated varies however with each application and the performance of the real-time system need not surpass that level of accommodation. Some applications, such as interactive systems like those used by financial institutions, mail order companies, or travel agencies, must respond within time intervals acceptable for human conversations which is in the range of one to fifteen seconds. Applications such as process control systems used in many types of plants usually impose more restrictive time constraints than the conversational applications, although they may allow response times exceeding one second. Other applications, like those used for aircraft flight controls, space shuttle control, or missile guidance control, require subsecond, immediate response to the environmental changes.

An inventory of all types of real-time systems would reveal that they are all interfaced to the outside world, event driven, and time constrained [3]. However, the systems that allow response times of three seconds or more do not usually encounter problems in meeting their

requirements, including response times, even without applying real-time design considerations. Normal practices of proper software design should yield an acceptable system. For systems with more stringent response times though, the requirements drive the design from the earliest phases [4]. The tactics applied to these systems differs from that of conventional systems development. The major challenges faced in developing real-time control software have led to significant contributions to software engineering in general.

The hardware, the operating system, and the application software are all factors that affect the system's performance. Poor choices during hardware configuration or the selection of an unsuitable operating system may render the system incapable of meeting its requirements regardless of the expertise applied in the application software. Because manpower costs tend to dominate the total cost of a system as its development and maintenance continues, the economic cost of performance problems introduced during a system's early phases increases as the system life cycle progresses [3]. Either extensive manpower must be expended in order to overcome the inadequacies of the hardware or the operating system or the system configuration must be changed (new hardware or a new operating system) which will most likely set back the application software development.

As stated before, the design of real-time application software should adhere to the standards of good software design, but the primary objective is meeting the user's requirements within the required response time. The speed with which the software accomplishes its requirements is then indirectly the designer's overriding concern, and the key to getting the speed he needs is efficiency. Taking efficiency measures to the extreme can result in a design and code that is difficult for others to follow. However, if the most time critical portions of the system are designed (and coded) with due emphasis on efficiency, other parts of the system may be able to lessen the emphasis with satisfactory results.

1.2 OVERVIEW

This thesis will address the development of real-time software systems by following the production of a real-time supervisory control system. This software system, chosen to illustrate the thesis topic, was developed and placed into operation at Arnold Engineering Development Center as part of a multi-year effort to automate the operation of plants supporting wind tunnel facilities. Mr. P. Wayne Belew, a Principal Engineer at Calspan Corporation, was the system project engineer. He performed a preliminary study and system analysis prior to preparing the system software

specifications. The author was assigned the role of Lead Programmer with responsibilty for all aspects of the system's software development. After the author completed the software design, Mr. Belew and another programmer, Mr. Michael Scott, assisted the author in generating the system's code. Each module was unit tested by its coder. The author then performed the remainder of the software testing and implementation phases.

Chapter Two presents background information pertinent to the supervisory control system and describes its purpose, application environment, and physical components.

Chapter Three discusses system analysis and the formulation of system requirements and specifications. The software requirements for the example real-time system are then presented.

Chapter Four addresses the design and development of real-time systems. Design philosophies are considered first, followed by the preliminary and detailed design phases of the supervisory control system. The final section of the chapter contains a brief discussion of implementation and testing, including a review of how the example control system was tested.

Chapter Five gives an evaluation of the example system and makes recommendations for improvements to both the supervisory control system and future real-time systems.

CHAPTER II

REAL-TIME SUPERVISORY CONTROL APPLICATION

2.1 APPLICATION BACKGROUND

During the course of developing a software system, the software designer must make many decisions about how the software will be implemented. These decisions span all levels of development, from software specifications to coding. The designer is constantly faced with a choice of methods with which to accomplish his specific goal. A specific goal may be a high level one, such as ensuring that all information that needs to be retained for later inspection is recorded in one or more disk files during the course of the software's execution. Alternatively a specific goal may be of the lowest level, such as comparing one memory location's contents to that of another.

In order to make informed and advantageous decisions about software implementation, the designer needs to have knowledge of the circumstances surrounding the process that the software is to carry out or affect. For example the software's response to detected errors in the system might be much more appropriate and well behaved if the designer understands the relative significance of an error to the software's physical environment beyond the computer.

Although knowledge of the process which the software is to influence is not a firm prerequisite for designing software, it is usually more important to the design of real-time software systems than to other types of applications. This is because the functions performed by real-time systems often involve the safety of equipment, resources, and people, which heightens the need for the designer to grasp the consequences of each decision he makes.

The following sections describe the background and purpose of the chosen application.

2.1.1 Plant and Compressors

The main compressor plant of the Von Karman Facility (VKF) at Arnold Engineering Development Center was the site of application for the subject software design. The Von Karman Facility is an aerodynamic testing facility. Operating conditions are simulated in wind tunnels, hyperballistic ranges, and space chambers for models of various types of aerodynamic craft (missiles, aircraft, spacecraft, etc.) and their components [5]. The main compressor plant of VKF, commonly referred to as the V-Plant, supplies the specially conditioned air that flows through the facility's three continuous flow wind tunnels for extended periods of time.

These three tunnels are known as Tunnel A, Tunnel B, and Tunnel C. Tunnel A is a supersonic wind tunnel that operates between Mach 1.5 and Mach 5.5 with pressures ranging from 1.5 to 200 psia [6]. Tunnel B and Tunnel C are hypersonic wind tunnels. Tunnel B operates at Mach 6 or Mach 8 at pressures of 20 to 900 psia. Tunnel C Mach numbers range from 4 to 10 with pressures of 20 to 1900 psia.

There are fifteen compressors in the V-Plant that allow one to ten stages of compressor operation, with inlet pressure as low as 0.1 psia and discharge pressure as high as 4000 psia [7]. Tunnel A uses one to five compressor stages; Tunnel B uses five or seven compressor stages; and Tunnel C uses seven or nine compressor stages. Ten compressor stages are used to supply high pressure (2000 to 3000 psia) air to other test facilities. The plant's first compressor stage consists of four parallel compressors, numbered 11, 12, 13, and 14. The second stage has two parallel compressors, numbered 21 and 22. The six compressors that make up the first two stages are axial flow compressors. The remaining eight stages use one compressor per stage. These compressors are numbered 31, 41, 51, 61, 71, 81, 91, 101, and 111. All of the compressors used for stages three through ten are centrifugal compressors.

There are nine different compressor configurations that are commonly used to provide the needed tunnel conditions. Figures 1 through 9* illustrate which compressors are used in each of the nine configurations. The figures also identify the compressors by number.

The fifteen V-Plant compressors are driven by six motor drive units known as S-1 through S-6. Each of the first five drive units has a 2500 HP wound rotor starting motor and a 16,000 HP synchronous main motor [7]. Drive unit S-6 has a 400 HP wound rotor starting motor and a 7000 HP wound rotor main motor. The axial compressors of stage one and stage two run at rotational speeds of 3600 RPM and are rated for pressure ratios up to 3.08. The centrifugal compressors run at fixed rotational speeds that range from 2847 RPM for the third stage to 15,075 RPM for the tenth stage with rated pressure ratios from 3.08 to 1.82. Each compressor has a flow recirculation valve for surge protection.

2.1.2 Surge Description

How a compressor reacts to a flow disturbance determines if it is stable or unstable [8]. If the compressor is capable of returning to its original set point, it is called stable. Otherwise it is deemed

---

* All figures are given in Appendix A.

unstable. Unstable systems are further classified as static or dynamic. Static instability results in a constant increase in deviation from the set point. Dynamic instability is an oscillating deviation from the set point with either a constant or increasing amplitude. Surge is a dynamic instability.

The condition called surge may or may not start out as a less severe condition called stall. Stall is caused by a reduction in the flow through a compressor. At some point in the reduction of flow, it becomes unstable and begins to oscillate [8]. If the unstable flow is restricted to localized regions around the compressor rotor, with the net flow through the compressor still being in a positive direction, the condition is called stall. The localized flow oscillations will be at some fraction of the compressor speed. If the instability develops into fluctuations of the entire flow region the condition is called surge. During surge the net flow through the compressor oscillates between positive and negative directions at large amplitudes and at frequencies between 0.5 and 10 hertz.

Figure 10 is a compressor map that shows the characteristic curve of a compressor for two operating speeds. Compressor manufacturers produce such maps for actual compressors, but the map in Figure 10 is only meant to be a typical example. Note that for the negatively

sloped portion of the characteristic curves, the pressure changes in the opposite direction of flow changes. A zero slope in the curves indicates no change of pressure for flow changes. For the remaining positively sloped portions of the curves, the pressure changes in the same direction as the flow. If a line is drawn through all the compressor's characteristic curves (the set of curves representing the full range of operating conditions) at their points of zero slope, the resulting line is known as the surge curve. Compressor stall and surge occur in the region to the left of the surge curve. The portion of the compressor's characteristic curve to the left of the dotted vertical line represents negative flow.

Suppose a compressor discharge valve fails in a closed position or some other unexpected occurrence causes an obstruction in compressor's discharge ducting, resulting in a reduction in flow through the compressor [7]. The operating point will then move to the left toward the zero slope. Unless countermeasures are taken, the operating point will pass the zero slope and enter a region where the compressor's air pressure is less than the discharge air pressure, causing the positive flow to stop. If the pressure differential is great enough, the flow will reverse direction. A negative flow through the compressor will draw down the air pressure in the discharge until it

is less than the compressor's air pressure. At that point forward flow will start again, but if the obstruction in the discharge ducting is still present, the cycle will be repeated [8].

The rapid flow oscillations of surge cause excessive vibration and runaway speed in the compressor. Temperatures within the compressor rise very quickly to extreme levels, particularly for axial compressors [7,8]. Both conditions can result in extensive damage to the compressor, even to the point of total loss. The cost of the damage to the compressor can be thousands to millions of dollars. The loss of revenue due to the compressor being inoperable may be greater than the cost of repair. Not all cases of surge result in obvious harm to the compressor; most do not. Nevertheless the surge may have altered the internal characteristics of the compressor such that it operates less efficiently. Repeated surge instances causing no apparent damage can significantly reduce the compressor's efficiency.

2.1.3 Previous Surge Control

Given the adverse effects of surge on compressors, it is clear that some form of surge protection is needed for the compressors in the VKF main plant. The chosen example of real-time software is part of an effort to provide an improved surge control system for those compressors.

Prior to this effort the surge protection for the compressors was minimal. Each of the six axial compressors had a thermocouple located at their inlet which provided detection of surge alarm level temperatures [7,9]. If the compressor's inlet temperature exceeded a preset limit, the bypass valve was automatically opened. Each of the nine centrifugal compressors had a discharge temperature thermocouple which provided only an alarm indication in the case of the measured temperature exceeding a preset limit; the compressor's bypass valve was not automatically opened. The automatic opening of an axial compressor's bypass valve did in most instances break the surge cycle. In some cases however it was necessary for the plant operating personnel to close the axial compressor's discharge valve, thus isolating the compressor from its parallel units, in order to alleviate the surge condition. Since the centrifugal compressors operate in a serial configuration, discharge valves are not pertinent to surge control of the centrifugal compressors.

The inadequacies of the prior system of surge protection may be condensed to two primary aspects. The first is that it provided no means of preventing the compressor from experiencing a surge condition; alarms occurred after the fact [10]. As noted in the previous discussion of compressor surge, allowing the compressor to

be subjected to the stress of surge is dangerous and costly. Secondly, valve movement essential to a compressor's recovery from surge was not automatic and hence not fast enough [9,10]. This aspect includes operation of bypass valves for the centrifugal compressors and operation of discharge valves for the axial compressors. Implicit to this aspect of valve movement is the need for the stroke time of a valve not to exceed some adequate response time.

2.1.4 Compressor Surge Study

A study of compressor surge in both the axial and centrifugal compressors of the VKF main plant was conducted by the project engineer to determine feasible methods of improving the surge protection of those compressors [7]. Conclusions and recommendations resulting from the study that are relevant to this discussion were:

1.) Detecting surge in its earliest form is not practical.

2.) A significant decrease in the time required to recognize alarm conditions was realized by locating the temperature alarm thermocouple in the second stage rather than the inlet. Compressor experts indicated they would expect the detection time to be even faster for thermocouples located

in the later stages since surge usually begins in that locale.

3.) Pressure ratio was found to be an unacceptable control variable for the centrifugal compressors because the pressure ratio varied only slightly for large variations in flow. Differential pressure versus flow was the recommended control mode for the centrifugal compressors.

4.) Accurate flow measurement was deemed impractical for the axial compressors due to the lack of straight ducting. Control of pressure ratio adjusted for suction temperature was recommended for surge prevention in the axial compressors due to the relative variance of the pressure ratio to that of inlet flow and inlet temperature.

5.) Instrumentation improvements were proposed in order to provide system response times adequate for surge protection. Those improvements included fast-response pressure transmitters for control of the bypass valves, faster hydraulics for the bypass valves, fast-response thermocouples for temperature measurements used for surge detection, and converting the parallel (axial) compressors' discharge valves to modulating valves. The discharge valve modification would allow the valves to be controlled by a surge control system.

6.) The method of surge protection should not interfere with the plant operating personnel's performance control of the compressor.

7.) Any controllers used as part of a surge control system should be capable of interfacing to the plant's network of computers.

Based upon the findings of the study, requirements for and specification of an improved surge control system were put forth and implemented.

2.2 SURGE CONTROL SYSTEM

The new surge control system that was implemented in the VKF main plant is described in this section. Three approaches to improving the surge protection of the compressors were examined [9]. They were a centralized control computer approach, a distributed anti-surge control approach, and a distributed performance control approach. The centralized control computer approach was discarded because the protection of all fifteen compressors would depend upon essentially one piece of hardware. A distributed approach would give each compressor its own piece of surge protection equipment and a spare piece of equipment could be made available. A distributed performance control system would be more flexible than a distributed anti-surge control system and therefore more

expensive. It would also require a greater degree of onsite programming because the performance controllers do not possess an anti-surge algorithm. The distributed anti-surge control system was determined to be capable of supplying the required level of surge protection without imposing any of the disadvantages of the other two approaches. It was thus chosen for the application in V-Plant.

The control strategy chosen for the V-Plant compressors was differential pressure across the compressor versus inlet flow differential pressure [9]. It may be noted that the previously mentioned study of surge on these compressors considered accurate flow measurement impractical for the axial compressors. However it was determined that the accuracy of the flow measurement was not as important as the repeatability of the measurement [7,10]. Subsequent evaluation of the axial compressor ducting revealed that the inlet mufflers also acted as flow straighteners allowing repeatable flow measurement. Furthermore the chosen control strategy does not use a direct flow measurement but rather a measurement of flow differential pressure.

The new surge control system required modifications of discharge valve controls for the axial compressors, a new manual control console, and new pressure and flow instrumentation on all the compressors. It also required an

anti-surge controller for each compressor's bypass valve and a supervisory minicomputer system interfaced to a network of other computers used for plant monitoring and control. Each anti-surge controller was to be interfaced to the supervisory computer and provision made for future interfacing of the controllers via the supervisory computer to a plant configuration computer system. The supervisory computer would provide secondary surge protection for all compressors in the event the primary system failed to prevent a compressor from surging. This secondary level of protection required software to be developed for that purpose. New instrumentation to interface several groups of I/O parameters to the supervisory computer was required. A specifically designed circuit board was necessary to incorporate some of the surge control requirements.

Figure 11 is a block diagram that shows the principal components and instrumentation groups of the new surge control system that will be discussed in the remainder of this chapter. Figure 12 illustrates the control instrumentation for any one of the six axial compressors. Individual features of the diagram will be discussed in the following sections. The control instrumentation for a centrifugal compressor is identical with the exclusion of the discharge valve control elements.

2.2.1 Valve Modification

The goal of valve modifications was to ensure that the hydraulics of all valves concerned with surge control were capable of a full stroke movement within the required response time and to make them compatible to computer control [9]. The new requirement for a full stroke was a maximum of two seconds. The bypass valves already had servo controls that allowed computer control [11]. The servo amplifiers were modified to decrease the valves' stroke times. The axial compressors' discharge valves did not already have servo controls and some of them required ten seconds to complete their full stroke movement [10]. Servo controls capable of delivering the required stroke time were installed for the axial discharge valves. That modification also allowed them to be closed by supervisory computer control in the event of a temperature alarm which would mean that the compressor was experiencing surge.

2.2.2 Controllers

The Compressor Controls Corporation (CCC) anti-surge controller was chosen to provide the primary level of surge protection for each of the fifteen compressors in the VKF main plant. These controllers are made for the express purpose of detecting, preventing, and recovering from compressor surge. They employ proprietary techniques within

their surge control algorithm called recycle trip and safety on. The CCC controllers have two modes of operation. In the automatic mode they use a proportional-integral (PI) algorithm to perform closed loop control on a compressor's bypass valve [11]. In the manual mode the user may control the bypass valve through buttons on the front panel of the controller.

The fundamentals of a controller's operation are more easily explained by referring to diagrams [12]. Figure 13 is a compressor map with the differential pressure across an inlet flow measuring device, labeled delta Po, as the horizontal axis. The vertical axis, labeled delta Pc, is the differential pressure across the compressor. The map shows a surge limit line which serves the same purpose as a surge curve as discussed in section 2.1.2, on pages 10 and 11. Several performance curves of various operating speeds (RPMs) are also shown. The distances labeled 1, 2, 3, and 4 will be described in reference to the operating point (large dot) shown on the RPM 1 performance curve. Distance 1 is the allowable increase in delta Pc if delta Po is to remain the same. Distance 2 is the allowable decrease in delta Po if delta Pc is to remain the same. Distances 3 and 4 are the allowable increase in delta Pc and decrease in delta Po if the rotational speed is to remain the same. Safe compressor operation would send the operating point away from the surge limit line toward the right side of the

map. Yet efficient compressor operation would send it toward the surge limit line. The objective of the anti-surge controller is to let the compressor operate as close to the surge limit line as possible while still providing adequate surge protection.

Computational circuits in the controller define two lines on a compressor map, shown in Figure 14, that are essential to the controller's prevention of compressor surge. The slope of the surge limit line is determined by the coefficient K. Coefficient K is defined as the cotangent of angle alpha. Figure 14 also shows where the surge limit line would be for a K equal to 0 and a K equal to 1. A typical compressor's value of K would fall somewhere between those two values. Thus the equation for the surge limit line is delta Po = K x (delta Pc). The value of K must be derived for a particular compressor from the manufacturer's compressor map or from empirical data. The second line in Figure 14 is called the surge control line. This line has the same slope as the surge limit line and is located a distance of b1, in the direction of increasing delta Po, from the surge limit line. The surge control line is defined by the equation delta Po = (K x (delta Pc)) + b1. The value of b1 must be determined for the particular compressor being controlled.

Figure 15 shows an operating point well to the right of the surge control line, representing completely safe

operating conditions. If the operating point moves far enough to the left to reach or pass over the surge control line (Figure 16), the controller's PI circuit will modify the controller's output to keep the operating point on the surge control line.

The recycle trip function of the anti-surge controller attempts to protect the compressor from an impending surge condition should the PI circuit be unsuccessful in keeping the operating point from moving too far to the left. The recycle trip line is shown in Figure 17. It is parallel to and located between the surge limit line and the surge control line on the compressor map. Its exact position is defined by the value RT. RT is the distance of the recycle trip line to the left of the surge control line along the delta Po axis. The recycle trip line's equation then is delta Po = (K x (delta Pc)) + b1 - RT. The value of RT must be determined for each controller's application. The recycle trip circuit of the controller is activated if the operating point of the compressor moves far enough to the left on the map to reach (or pass) the recycle trip line, as in Figure 17. When activated, the recycle trip function opens the compressor's bypass valve by amount C1, and continues to open the bypass valve by that amount for each time interval C2 in which the compressor's operating point is located on or to the left of the recycle trip line. When the bypass valve has been opened enough to move the

operating point to the right side of the recycle trip line, the recycle trip circuit exponentially decreases its output with time constant TL, to allow the bypass valve to slowly close. The values of C1, C2, and TL must be supplied by users of the controller.

The recycle trip attribute of the anti-surge controller is included to handle sudden, surge threatening disturbances experienced by the compressor that require faster and more acute responses than provided by the controller's PI control loop. It thus provides an extra level of surge protection above that of conventional control algorithms. Even if the controller is in the manual mode rather than the automatic mode of control, the controller will still recognize a recycle trip condition. If it does detect a recycle trip condition while in manual control mode, it will transfer itself to automatic control mode in order to take protective action. However, there is a separate feature of the controller called manual override which, if enabled, will prevent it from transferring itself from manual to automatic control mode, regardless of the compressor's condition.

The safety on feature of the anti-surge controller alerts the compressor operator to an occurrence of compressor surge and takes action to prevent the condition from reccurring. The safety on line, shown in Figure 18, is another line parallel to those previously discussed,

located a distance of SO to the left of the surge limit line. Therefore the line is defined by the equation delta Po = (K x (delta Pc)) - SO. SO is another value that must be determined for a particular compressor. If the compressor's operating point moves to the left of the safety on line, the controller interprets such a position as an actual surge condition. It responds by turning on the red safety on light on the controller's front panel to alert operating personnel. In addition it also increases the distance between the surge limit line and the surge control line by amount b2, thus moving both the surge control line and the recycle trip line further to the right along the delta Po axis.

The movement of the lines means that once the compressor has recovered from its present surge condition, it will not be allowed to come as close to the surge limit line as it was before the present surge condition. This increases the margin of safety because the previous margin was obviously not sufficient to prevent the compressor from surging. Should the compressor's operating point activate the safety on function again, the surge control and recycle trip lines will again be shifted to the right by amount b2, and likewise for any subsequent occurrences. The operator can cancel the actions taken by the safety on function by pressing the reset button, which will turn off the red safety on light and return the distance between the surge

limit line and the surge control line to its original value.

The control functions of the anti-surge controllers do not interfere with manual control of the bypass valve unless the compressor's operating point moves too close to the surge region of operation [9,10]. In such cases the controller would take overriding control of the bypass valve in order to protect the compressor from surge. Once the compressor has been returned to a safe region of operation, its functions would again be transparent to manual operation. The operator's options for overriding automatic valve control are given in section 2.2.3.

Each anti-surge controller accepts four channels of analog input and provides analog output for bypass valve position control [11]. Two of the analog input channels are for delta Po and delta Pc as defined above [12,13]. Due to the wide range of pressures that both of these measurements must cover, two pressure transducers (one low range and one high range) are used for delta Po and two are used for delta Pc (Figure 12). The third and fourth analog input channels are used to tell the controller which of the two transducer readings is being used for the first (delta Po) and second (delta Pc) channels respectively.

The CCC controller is also capable of communicating with a computer over an RS-232C standard serial interface

using Modicon Modbus protocol as described in Reference 13. This interface allows a host computer to read controller inputs, controller output, control algorithm constants, controller status modes (manual/auto mode, manual override on/off), and controller alarm conditions [9,13]. It also allows remote programming of a controller.

2.2.3 Signal Selections

A signal selection circuitry board was designed and built for the surge control system in order to implement some specific control aspects of the system [11]. Three categories of signal selection were provided.

The first type of signal selection chooses either the low range delta Po transducer or the high range delta Po transducer for input to an anti-surge controller. This function is duplicated for each of the fifteen CCC controllers. A similar selection is made between the low and high range delta Pc transducers for each controller. The function (Figure 12) chooses the high range transducer if the low range transducer signal is above ninety percent of its full scale.

Another type of signal selection is that of valve control. Bypass valves in this system have three sources of control output: an anti-surge controller, a manual control potentiometer, and an analog computer output. All three

sources are input to a function (Figure 12) that chooses the signal that provides the safest (for the compressor) position of the bypass valve. In other words it selects the most open position for the bypass valve. Discharge valves on the axial compressors have two sources of control output: a manual control potentiometer and an analog computer output. The two signals are input to a function (Figure 12) that also chooses the safest position, which is the signal for the most closed position of the discharge valve.

The last type of signal selection enables the operator to override the automatic control of bypass and discharge valves. A controller override switch is provided for each of the fifteen compressors. These switches will normally be left in the off position. Their main purpose is to allow the operator to override a CCC controller in the event of a controller malfunction. The controller's output will be excluded from signal selection for bypass valve output when the switch is in the on position. A computer override switch is also provided for each compressor. Like the controller override switches they will normally be left in the off position. When a switch is placed in the on position, the computer's output signals are excluded from the selection for bypass valve output and for discharge valve output if the compressor is axial.

2.2.4 Manual Control Console

A new manual control console was installed for this surge control system [9,11]. Each of the fifteen bypass valves and six discharge valves has a separate control module on this console. Each control module provides a manual control potentiometer and a valve position meter. Bypass valve modules include an alarm LED that will be turned on if the corresponding compressor's anti-surge controller activates the recycle trip function. This LED will flash if the controller detects a safety on condition or if the supervisory computer detects a temperature alarm for the corresponding compressor.

Controller and computer override switches were installed in a separate panel of instrumentation in the operator's area. This panel also contains switches for turning servo amplifier power off or on for each bypass valve and each axial discharge valve. Servo power for each valve is also indicated by an LED.

2.2.5 Supervisory Computer

A supervisory computer was specified for the surge control system in the VKF main plant in order to provide secondary protection for the fifteen compressors and in order to enhance the application of the CCC anti-surge controllers [9]. Should the anti-surge controllers fail to

prevent a compressor from going into surge, the secondary protection of the supervisory computer will take the appropriate control actions necessary to bring the compressor out of surge. It was noted earlier that the discharge valve of a surging axial compressor may have to be closed to isolate the compressor from its parallel units before it can recover from surge. The computer provides the only method of automatic control of the axial compressors' discharge valves since the anti-surge controllers only have control output to the bypass valves.

The central function of the computer's secondary protection is the detection of compressor temperature alarms. Thermocouples having a range of 0 to 400 degrees F were positioned at the inlet of each compressor (Figure 12). Thermocouples having a range of 0 to 600 degrees F were positioned in the second stage (internal) of compressors 11 through 61 (Figure 12) and at the discharge of compressors 71 through 111. If a compressor temperature alarm is detected by the supervisory computer it will open the compressor's bypass valve and flash an alarm light for that compressor on the operator's manual control console [9,11,14]. At some brief time interval (to be set in software) after the supervisory computer has commanded the bypass valve to open, it will check the position of that valve. If the bypass valve is not yet open the computer will remove the power from the servo amplifier for

that valve which will force it to a mechanically biased open position. In addition it will close the discharge valve of an axial compressor, should the temperature alarm not be removed by opening the bypass valve.

Instrumentation to accomplish the supervisory computer's functions includes two analog temperature inputs for each compressor, analog setpoint outputs to each of the fifteen bypass valves and to the six axial compressor's discharge valves, analog valve position indication inputs for each of the same, discrete relay outputs to the servo amplifier of each bypass valve, analog servo power inputs for each bypass valve and for axial discharge valves, discrete outputs to an LED alarm lamp for each compressor, and discrete inputs for each of the controller override and computer override switches [11,14]. Temperature inputs are checked for alarm levels and analog setpoint outputs are set accordingly. Valve position inputs are used to check the position of bypass valves after they have been commanded to open due to a temperature alarm. Both the analog outputs and the analog inputs for valves are also used for maintenance calibration and diagnostic functions during periods when the plant is not operating, and they furnish an alternative method of manual valve control. Discrete outputs to servo amplifiers provide the ability to remove power from an amplifier. In the future the status of servo power inputs will be sent to a configuration

system computer. The servo power status is presently made available to the user. The purpose of the alarm lamps was explained in the previous section. The states of the override switches are presently reported to the user but will in the future be reported to a configuration computer system.

The supervisory computer was interfaced to each of the fifteen CCC anti-surge controllers and software was written to service that interface. This software allows the controllers to be programmed through the computer and provides easy viewing access to any controller's input parameters, status modes, setpoints, control constants, and alarm conditions. However the main purpose of this hardware and software interface is to allow a future configuration computer system to communicate with the controllers, receiving information from the controllers and passing commands to them by way of the supervisory computer [9].

A flashing alarm lamp on the operator's manual console (indicating a controller safety on condition or a computer detected temperature alarm) will not cease flashing until the alarm is reset by the operator [11,14]. The supervisory computer's user interface supplies the reset function.

The supervisory computer will communicate with two other computer systems over network connections. Several aspects of communication with a configuration computer system to be implemented in the future have already been

iterated. In addition the supervisory computer will send the configuration system certain parameters of data, all alarm conditions, and various other status information items [9,14]. Recorded data and other files will be sent via the network to another computer system for archival purposes.

2.3 COMPUTER SYSTEM

Section 2.1 stated the need for a designer of real-time software systems to understand the application's process and physical environment. However the designer's foundation of knowledge would not be complete without adequate familiarity with the system's hardware components and the operating system under whose supervision the software will execute. Hardware components include the computer, standard peripheral devices, and any other devices interfaced to the computer. Limitations and capabilities of the hardware should be known and considered. If a software interface is not provided with nonstandard hardware, it must be developed. In such cases detailed information about the device's control and communication protocol is mandatory. Familiarity with the operating system is meant to include any layered software that also runs on the same computer, such as software that handles network communications.

When a software designer is well acquainted with the characteristics and unique aspects of the resources of the system with which his developed software will interact, he can make decisions that result in more efficient use of the hardware and the software. This type of information is usually more vital to the design of real-time software applications than to non-real-time applications. This is because real-time systems often place an emphasis on speed of data acquisition, system response to asynchronous events, and the processing of information, thus magnifying the value of efficiency at all levels of software design. Indeed, lack of attention to this issue frequently results in a failure to meet real-time specifications and subsequent redesign. At best it leaves the system with less capacity for future expansion. The computer, hardware devices, and software components of the real-time supervisory control application are described in the following sections.

2.3.1 Computer

A PDP-11/73 micro computer made by Digital Equipment Corporation (DEC) was the target computer of this design. DEC's line of PDP-11 computers is a very popular computer architecture, well-suited to real-time applications, and DEC has produced many successful operating systems for the machines [1]. The computer system consists of a KDJ11-A

processor, 1.5 megabytes of RAM, a 70 megabyte Winchester type (RD53) hard disk, two 5 1/4" (RX50) floppy disk drives, 16 serial I/O channels, a DEQNA Ethernet interface, and an I/O subsystem interface [11,14]. All of these components are Digital Equipment Corporation products except the serial I/O controller which is made by Emulex and the I/O subsystem controller which is made by Analogic.

The KDJ11-A processor performs self-diagnostics at boot-up and sets diagnostic LEDs accordingly [15]. The processor board also contains a floating-point processor and a serial I/O port which is the system console port. The KDJ11-A uses 22-bit memory addresses and the LSI-11 I/O bus (as opposed to DEC's other PDP-11 bus, the UNIBUS) which is commonly referred to as the Q-bus.

The floppy disk drives are used for installation of the operating system and other software in its distribution form. The hard drive contains the installed operating system and all user applications.

The Emulex 16-channel serial I/O controller emulates two DEC DHV11 communications controllers. This type of controller has a significant advantage for real-time environments over the more commonly used DLV11 controllers. Whereas the DLV11 controller interrupts the processor for each input character, the DHV11 uses DMA transfers and interrupts the processor only when transmission is complete.

The DEQNA interface is what places the computer on the V-Plant Ethernet network and allows it to communicate (given the necessary software connections) with any other computer on the network. It also allows users to log on to the computer through a terminal server rather than through a direct serial port.

Hardware interrupts are handled by the PDP-11/73 (or any other PDP-11 machine) through interrupt vectors. Each hardware device that is capable of interrupting the CPU has a vector address stored onboard its controller. Vector addresses use only three octal digits since they reside in low memory and their upper digits are always zeros. An interrupt vector consists of two consecutive words in memory. The first word, located at the vector address, contains the address of an interrupt service routine for the interrupting device [16]. The second word contains a processor status word to be used while the interrupt service routine is being executed. A processor status word is used to regulate various aspects of program execution, such as processor priority level, condition codes, protection mode, etc.

When a device interrupts the CPU the program counter and current processor status word are saved on a stack. Then the starting address of the interrupt service routine is loaded in the program counter from the first word of the interrupt vector and the processor status word is loaded

from the second word of the interrupt vector. The return from interrupt (RTI) instruction restores the program counter and processor status word of the program that was interrupted when the service routine is finished. Software interrupts are called traps and are handled in the same manner.

All interrupt service routines are expected to restrict the amount of processing time they use to a system defined limit. Should a service routine require more time, it must (by system rules) lower its priority to allow itself to be interrupted by other requests and complete its processing at the lower priority.

The KDJ11-A processor provides total memory management and protection for a multi-user environment [15]. PDP-11 addresses specify the location in memory of a 16-bit word or an 8-bit byte (ignoring floating-point data). Program instructions occupy 16-bit words. Memory management provides the translation from a 16-bit program address, called a virtual address, to the 22-bit physical memory address (Figure 19).

Programs are limited to 32 K words of address space unless they use the I (instruction) and D (data) option when the program is linked, in which case they have 32 K of I-space and 32 K of D-space. In both cases 32 K of program space is divided into eight pages of 4 K words. Memory

management registers called Page Address Registers (PARs) are used to map the virtual address pages of a program to 4 K (or less) pages of physical memory (Figure 20). If a program does not use the entire 4 K of a virtual page the memory management does not allocate the unused space (above the next integral multiple of 64 bytes).

There are three sets of 32 16-bit registers used for memory management (Figure 21). One of the three sets is selected by the two most significant bits of the processor status word which contain the current protection mode. The protection level is highest for user mode which has the lowest level of system privileges. Supervisor mode has more privileges and less protection than user mode. Kernel mode has all privileges and minimal protection. Each set of 32 registers consists of 16 registers for I space and 16 registers for D space. A 16 register group is further subdivided into 8 Page Address Registers (PARs) and 8 Page Descriptor Registers (PDRs), so there is one PAR and one PDR for each I space virtual address page and for each D space virtual address page of a program.

The three most significant bits of a virtual address determine which pair of PAR/PDR registers will be used for the address translation (Figure 22). The remaining bits of the virtual address are called the Displacement Field (Figure 22) and are divided into the block number (bits 6 through 12) of the current page and the displacement in

the block (bits 0 through 5) referred to by the block number (Figure 23). The PAR contains a Page Address Field (PAF) (Figure 24) which specifies the starting address of the page as a block number in physical memory. The PDR contains information about the page's length, expansion, and access control (Figure 25).

Figure 26 illustrates the construction of a physical address. The 7-bit block number field of the virtual address is added to the 16-bit Page Address Field in the PAR to obtain bits 6 through 21 of the physical address. The six least significant bits of the virtual address, the displacement in block field, are retained as the six least significant bits of the physical address.

The Q-bus is an asynchronous bus that uses a master-slave relationship between I/O devices [16]. Memory is treated as any other device. Since most I/O transfers involve data being stored to or retrieved from memory, this aspect of DEC's bus architecture serves to further unburden the CPU and simplify those transfers [17]. A device that requires bus service arbitrates with other devices for control of the bus. The current data cycle is overlapped with arbitration for the next cycle. The device with the highest priority request will become the next master of the bus and the device that is the object of its data transfer

will become the slave. The master device controls the bus until it releases it.

Devices on the bus are assigned one of five priority levels. This assignment is referred to as a device's vertical priority. For devices of the same priority level, the one that is physically closest on the bus to the CPU will have the highest priority within that level. Likewise the device farthest from the CPU on the bus will have the lowest priority within its own level. This type of priority is called horizontal priority. The vertical and horizontal aspects of a device's priority are determined by the user, which allows the flexibility to give real-time interfaces the level of priority required.

The uppermost 4 K words of memory (one page) are reserved for device control and status registers (CSRs) and I/O. This highest page of memory is called the I/O Page. The uppermost virtual address page of every program that requires any type of I/O is mapped to this page of physical memory.

2.3.2 Terminal(s)

It is not unusual for real-time systems to require terminals that provide processing speed and/or graphics resolution beyond the industry norm. However, such advanced technology terminals may pose a maintenance problem because

they are usually a one-of-a-kind piece of hardware and their price often prohibits purchasing spares. Therefore unless the requirements of the application cannot be satisfied otherwise, a terminal should be adopted that does not create future maintenance problems.

All video terminals used for the surge control system were Digital Equipment Corporation VT200 series terminals. They were the media object of the user interface software to be used by the plant operators, and they also were used as a development and documentation tool. The VT200 series was chosen as the user interface because of its widespread availability and its familiarity among software and hardware personnel.

There are three models in the VT200 line of terminals: VT200, VT240, and VT241. The VT200 is a monochrome terminal that does not support graphics [18]. The VT240 is a monochrome model that can emulate Tektronix 4010/4014 graphics and supports DEC's Remote Graphics Instruction Set (ReGIS). The VT241 is an RGB color terminal with the same graphics capability as the VT240. Since this application did not require color or graphics displays any VT200 model terminal can serve as the user interface.

Several models of DEC printers can be directly connected to any VT200 series terminal to print the text or graphics on the terminal screen. The screen display of the VT200 terminals is 24 lines by 80 or 132 columns. The

number of columns can be changed from the set-up menu. The baud rate (up to 19,200) can also be selected from the menu, as well as 7-bit or 8-bit ANSI control modes. All VT200 models have fifteen programmable function keys and video features that include bold, blink, underline, and reverse video on an individual character basis [19]. Lines may be set to double-height and/or double-width. The cursor may be positioned anywhere on the screen. There are many other selectable attributes too numerous to enumerate. All operating modes and features of the terminals can be set from the keyboard or through software commands.

2.3.3 I/O Subsystem

The I/O Subsystem is an Analogic Model ANDS5400. Excluding cabling the system has six components: a user card backplane, a programmable gain signal processor, an A/D converter, a control and front panel display unit, a power supply system, and an interface card (Model ITG-38) designed for the DEC 22-bit Q-bus and mounted on the PDP-11/73's backplane (Figure 27) [11,20].

Slots J1 through J16 in the user card backplane are used for any Analogic I/O card the user chooses. Slots J17 and J18 are reserved for a pair of expansion cards which are used by the I/O subsystem to channel all transfers to the correct chassis and user card. The subsystem may

include up to seven chassis added to the master chassis, each having the same structure as the master.

The scheme for addressing analog and digital channels is illustrated in Figures 28 and 29 respectively. Numbering of chassis, slots, and channels begins at zero and progresses upward. Three binary digits for the chassis number allow the maximum selection of 7 (decimal) for the eighth chassis in the system. Four binary digits for the slot number allow the maximum selection 15 (decimal) for slot J16. Five binary digits for an analog channel allow the maximum selection of 31 (decimal) for a 32-channel card. Analog input cards may have four, sixteen, or thirty-two channels depending upon the type of card and whether it is used in single-ended mode or differential mode. Two binary digits for a digital channel allow the maximum selection of 3 (decimal) for a 4-channel card. Digital I/O cards may have one, two, or four channel addresses. A digital channel may access eight or sixteen discrete bits. The four most significant binary digits of an analog address allow a mode and/or a gain selection, depending on the model of the signal processor. One signal processor model uses only the most significant bit to select either single-ended mode or differential mode. The signal processor used for the surge control system uses all four bits to select both the signal mode and gain. Valid gains range from one to thirty-two, incrementing in multiples of

two. The three binary digits labeled Special Function in Figure 29 are normally unused but may on occasion be used for specific system commands.

The surge control system has sixteen user I/O cards [14]. Appendix B lists the I/O parameters, their addresses, slot and channel numbers for analog parameters, and slot and bit numbers for discrete parameters. The appendix may help clarify the description of the system's I/O cards that follows.

Slots J1 through J6 each contain a 4-channel 12-bit analog D/A output card. Twenty-one of the twenty-four channels are used for computer output to bypass and discharge valves. Slots J7 and J8 each contain an 8-relay digital output card. Fifteen of the sixteen relays are used to control power to the servo amplifiers of the bypass valves. Slots J9 and J10 each contain a 16-channel 12-bit analog A/D card. Thirty of the thirty-two channels are connected to the inlet and internal/discharge compressor temperatures. Slot J11 contains a 32-bit digital output card with two channel addresses. Fifteen of the sixteen bits in one channel address are used to control the illumination of the bypass valve alarm lamps on the manual control console. Slot J12 contains a 32-bit digital input card with two channel addresses. Fifteen bits of the first channel address are used to input the states of the

computer override switches and fifteen bits of the second channel address are used to input the states of the controller override switches. Slots J13 through J16 each contain a 16-channel 12-bit analog A/D input card. Twenty-one of the thirty-two channels on the cards in Slots J13 and J14 are used to read the positions of the bypass and discharge valves. Twenty-one of the thirty-two channels on the cards in Slots J15 and J16 are used to read the power indications of the servo amplifiers for bypass and discharge valves.

The Q-bus interface card makes the data acquisition subsystem an extension of the computer hardware, thus allowing software access to the I/O subsystem's registers [20]. The base address of this interface is set by four switches on the card. The binary value of the four switches is added to 160000 (octal) to form the base address. The interface may be used for programmed I/O (one channel at a time) or DMA (Direct Memory Access) data transfers. The ANDS5400 subsystem is capable of transferring DMA data at a rate of 50 kilohertz [11]. Three operating modes, A/D input, digital output, and digital input, are used for both programmed I/O (PI/O) and DMA operations. Figure 30 shows the mode number used by the subsystem for each combination. Analog output is treated the same as digital output; the digital output modes are used to transfer analog output data. Figure 31 lists the

registers of the I/O subsystem that are accessible through the interface. Offset addresses given for the registers are added to the interface's base address to yield their actual memory addresses.

The master registers may be used to access digital (not analog) channels directly. Their offset addresses are the same as their card addresses, described above. They cannot be used with DMA transfers.

The DMAR1, DMAR2, and DMARX registers are used to perform DMA transfers. DMAR1 is loaded with the starting location of an address table. The address table is an array of channel addresses to be read for A/D or digital input or it is an array of channel addresses to be written to for digital output. A -1 placed at the end of the array of addresses tells the subsystem it has reached the end of the list. DMAR2 is loaded with the starting location of the data buffer. The data buffer will receive the data from an A/D or digital input operation or it will contain the data (set up by user software) to be output for a digital output operation. Since all the registers discussed are 16-bit registers, DMARX is used to provide the remaining six address bits needed to make DMAR1 and DMAR2 22-bit addresses. After the DMAR registers have been loaded, the mode of operation is set in the CSR (Figure 32), and then the GO bit in the CSR is set high. When the GO bit is set,

the READY bit in the CSR will go low until the DMA transfer is complete (or an error occurs, or the GO bit is cleared). At that point the GO bit will be reset, the READY bit will be set high, and an interrupt is generated by the interface.

The TRR register is used for the three PI/O modes. The first step of all PI/O operations is that of setting the mode in the CSR. The TRR is then loaded with the desired channel address to be read for A/D or digital input modes or the channel to receive data for the digital output mode. If the mode is A/D input the READY bit in the CSR will go low when the address is loaded in the TRR. When the data conversion is complete the READY bit will be set high and the interface generates an interrupt. The data may be read from the TRR as soon as the READY bit goes high. The READY bit is not involved in the digital input mode. The addressed digital data may be read from the TRR immediately after the channel address is loaded. The READY bit is not used for the digital out mode either but the PART 2 bit is. After the TRR has been loaded with a channel address, it is loaded again with the data to be sent to the channel address. In order to avoid any synchronization problem between the time the I/O subsystem gets the channel address out of the TRR and the time that the TRR is overwritten by the data being loaded, the PART 2 bit is set high after the I/O subsystem has acquired the address from the TRR. The

data should not be loaded in the TRR until the PART 2 bit goes high. The PART 2 bit will be reset when the data has been output to the addressed channel.

Interrupts generated by the I/O subsystem are not enabled unless the appropriate bit in the CSR is set. If the INT ENB bit of the CSR is set, the interrupts mentioned above (completion of a DMA transfer or of an A/D conversion) will be enabled along with interrupts generated as a result of an error condition. Setting the 54 ENB bit enables interrupts from user cards requesting service. Interrupts enabled by the INT ENB bit use a different interrupt vector from those enabled by the 54 ENB bit.

The CLR1 bit of the CSR is set to perform a clearing function in the interface. It must be set in order to clear error conditions and is helpful in clearing synchronization problems. Setting the CLR bit of the CSR causes the I/O subsystem to reset itself to its power-up conditions. Obviously this may not be desirable unless the software is prepared to start over.

The SSH bit of the CSR is used for sample and hold operations and is normally low.

The RMT bit may be read to determine if the I/O subsystem is in local or remote mode. It is high if the mode is remote and low if the mode is local. The mode is controlled by a button on the front panel of the subsystem.

Computer commands to the I/O subsystem are ignored while the local mode is selected. This bit may be checked by software and users alerted via some user interface if the mode has been changed to local when the subsystem should be under computer control. However the LOCK bit provides the ability to prevent that occurrence. If the LOCK bit is set while the subsystem is in remote mode, the front panel buttons will no longer have any effect. If it is set while the subsystem is in local mode it is ignored.

The DMA ERR bit of the CSR is set high if an error is detected during a DMA operation. The PI/O ERR bit is set high if an error is detected during a PI/O operation. The ERR bit is set high if either the DMA ERR bit or the PI/O ERR bit is set high. As stated before, the CLR1 bit must be set to clear the error bits.

2.3.4 Software

Software resources used to develop and implement the software design for the surge control system consisted of a Micro/RSX Version 3.2 operating system, a Micro RSX FORTRAN Version 5.0 compiler, a PDP-11 MACRO assembler, and DECnet-11M-PLUS for network communications.

Micro/RSX is based on the larger RSX-11M-PLUS operating system and is specifically tailored to the MicroPDP-11 line of processors [18]. The acronym RSX stands for Resource Sharing Executive. RSX literature most often

refers to the operating system code as the executive. RSX is a multi-tasking, multi-user operating system, allowing concurrent execution of many tasks (programs in their executable form are called tasks). It was designed as a very compact operating system, requiring a minimum of overhead processing. RSX is a very flexible system that supports any type of application although it has gained its strongest reputation in the real-time arena [1]. It has been said, "The PDP-11 is all things for all people and RSX-11M is the operating system that makes all this a reality" [3].

The RSX operating system was actually designed for real-time applications. Its scheduling algorithm is what makes it (or any other such system) an excellent real-time operating system. Its scheduling is driven by events and priorities. All system resources are managed on the basis of task priority which range from 1 (low) to 250 (high). Before elaborating on the real-time scheduling algorithm however it may be helpful to introduce some related aspects of the operating system.

RSX makes available to user-written programs certain system routines called system directives [21]. Allowing user programs to execute these system routines directly gives the programmer the capacity to create more powerful and more efficient programs than would otherwise be

possible. There are 94 directives that may be called from either FORTRAN or Macro programs. There are 13 additional directives that can only be called by Macro programs.

The directives may be grouped into ten categories which are: task execution control, task status control, informational, event associated, trap (software interrupt) associated, I/O and intertask communications, memory management, parent/offspring task associated, RSX-11M-PLUS and Micro/RSX, and command line interpreter support. Memory management directives allow a task to dynamically map and unmap parts of its virtual address space to different regions of memory, usually to access shared data regions. They also allow tasks to dynamically create regions in memory. Intertask communication directives set up a data buffer in the system through which a task can send information directly to another task.

One of the most widely used directives is Queue I/O Request (QIO). This directive allows the user to make I/O requests at the driver interface level, thus bypassing much of the system overhead associated with getting to that level and permitting the user to be much more specific about the way he wants the system to process the I/O request. The QIO is a very generic directive; it is used to make I/O requests of any device driver in the system regardless of the type of device.

In order to use some of the directives a task must be mapped to the executive, which actually makes the executive a part of the user task and only leaves the user 8 K words of address space to code the program (Figure 33). Of course the user is not allowed to map his task to the executive unless he is privileged to do so. This privilege places all the resources and power of the operating system at the disposal of the programmer.

The term significant event is given to a change in system status that causes the scheduler to review the list of tasks waiting to run [22]. Not all changes in system status are classified as significant events. Without listing all possible origins, the main sources of significant events are execution of any of a small group of system directives, the completion of an I/O operation, a round-robin scheduling pass, and the completion of a task's execution [3].

RSX provides 96 event flags for use by the executive and by user-written tasks [21]. They are divided into three groups of 32 flags. Flags 1 through 32 are called local flags and are recognized only within the task using them. Flags 33 through 64 are called global flags and are recognized by any task in the system. Flags 65 through 96 are called group-global flags and are recognized by tasks of the same group code. The state of an event flag is usually coupled with an event. Event flags may be used by

tasks for synchronization with internal (to the task) or external events. Using system directives, tasks can read the state of a flag, set it, clear it, or wait for a flag to be set by an another task.

The scheduler always grants the processor to the highest priority task that is ready to run [18,22]. However in order for a task to be considered ready to run it must pass several checks. It must be in memory, it must not be stopped, blocked, suspended, or in a wait for state, and it must have been allocated all the resources it requires.

Assuming a task has been requested to run by some means, it may not be in memory for a couple of reasons. First, since all tasks compete for all system resources, including memory, the requested task may not have been granted the memory space to begin its execution. Secondly, after the task has initially been loaded into memory and is executing, it may reach a point that it is unable to continue until some external event occurs, or perhaps until an internal timer expires. While the task is in a state of being unable to continue, it may be swapped back out to disk if a higher priority task is waiting on the disk for available memory. After the event that it was waiting for occurs, the task will renew its quest for memory resources. Thus although it may be an active task, meaning it has been requested to run and it has not completed, it may not be in

memory when the scheduler looks for the next task to run. This occurrence becomes less frequent as the amount of memory found in PDP-11/RSX systems increases.

A task becomes blocked, stopped, suspended or enters a wait for state by issuing a system directive (either implicitly or explicitly) that results in one of these states. Typical examples are a request to wait for the completion of an I/O transfer and a request to wait for an event flag to be set.

If a task passes all the ready to run checks and it is the highest priority task to do so, the scheduler will select it as the next task to run [3,21]. It will be allowed to run until it releases the CPU, either by completing its execution or by issuing a request that results in one of the non-ready states, or until a significant event occurs in the system. A task may itself issue a significant event through a system directive in order to voluntarily allow the executive to service pending requests. If the executing program is interrupted because of a significant event, the scheduler will go through the process again of selecting a task to run, but if the task that was just interrupted is still the highest priority one ready to run, the scheduler will return the processor to the same task.

Round-robin scheduling applies only to a specific range of the lower priorities [22]. The default range of

priorities subject to round-robin scheduling, which may be changed, is 1 to 150. Tasks of the same priority level within this range are rotated so as to share processing time allowed their priority level. When the scheduler looks at a list of tasks of the same priority, it schedules for execution the first one it encounters that is ready to run. The task that was selected to run is then moved to the end of the list. If a task in the list is not ready to run when the scheduler checks its status, it retains its position in the list. The general effect is to move I/O bound tasks to the front of the list and move CPU bound tasks to the rear of the list.

The round-robin scheduling applied to its range of priorities does not diminish the performance or effectiveness of real-time scheduling if task priorities are assigned appropriately [3]. Obviously time-sensitive, real-time tasks should be assigned priorities that are above the range of the round-robin scheduler so they will not be interrupted by the scheduler to check for other tasks of the same priority level that may wish to run. Moreover it should be understood that a task's priority level relative to that of other tasks is more important for real-time processing under RSX than is a task's absolute priority level. Real-time applications can execute concurrently with normal user activities such as copying

files, editing, compiling, etc. because user utilities and tasks used in program development are given a relatively low priority (50 to 60) so they will not hinder the system's response to real-time tasks.

Two separate file management services are furnished with Micro/RSX. Both file services permit block I/O as well as record I/O. The File Control Services (FCS) allows sequential and direct access to file records. The Record Management Services (RMS) was developed subsequent to FCS and it supports more complex file structures, allowing sequential, direct, keyed-index, and file address access to records. FCS uses only one type of file organization, which is sequential. The organization type of RMS files may be sequential, relative, or indexed. Records of either file service may be fixed-length or variable-length and they may be formatted or unformatted [23].

The FORTRAN statement used to open a new file allows these and other file attributes to be specified. Not all combinations of the attributes are allowed. For example if direct access is desired for an FCS file with sequential organization the records must be fixed-length; they cannot be variable-length. Unspecified attributes will be set by default. If the file already exists, the attributes need not be specified, but if they are they must agree with those of the file.

In spite of these restrictions, Micro/RSX offers a software designer considerable latitude to tailor his files to suit the application. The designer should appreciate, however, the trade-offs that must be made between ease of use and efficiency. For instance, using RMS is far more costly than using FCS in the amount of system overhead required, but FCS does not provide keyed-access indexed files. Also, block I/O operations are much faster than record I/O operations, but additional coding is required to make the information readable to users.

Each of the file services has a library of routines consisting of position-independent code that can reside anywhere in memory [18]. A task that requires file services may be linked to either or both of the libraries. This means that any number of tasks may share the same source code (in one or both libraries) for file services routines, thus conserving memory usage.

Micro/RSX has two command languages: MCR (Monitor Console Routine) and DCL (Digital Command Language). MCR is the original command language of RSX. The DCL command language available on Micro/RSX is a subset of that used by the VMS operating system which runs on DEC's VAX computers. Frequently repeated sequences of commands (such as compiling, linking, and purging versions of a program) may be put in an indirect command file and executed by entering one command to an Indirect Command Processor [24]. The

total set of commands intended for the Indirect Command Processor constitute a programming language that can execute any MCR or DCL command, create and read files, test and branch on conditions (including the status of tasks), carry on an interactive dialog with a user, and use symbols (variables) to substitute values into a command [25].

RSX provides many aids to system management in the form of system utilities [18,26]. There are extensive tools available to test, backup, restore, and manage offline media volumes (disks and tapes). An error logging utility can be used to retrieve information about a system failure or to aid in analysis of the health of system peripherals. Resource accounting tools help track and control resources. System management programs can create user accounts, grant or restrict privileges, and tune system performance. A Resource Monitoring Display (RMD) provides information updated once a second about all active tasks in the system.

User utilities include a full-screen editor that is capable of working on multiple files and takes advantage of the editing keypad on DEC's VT100 and VT200 series terminals to provide one stroke commands. Other user utilities offer a full range of file management and program development tools, including a library utility to create and maintain user source code libraries.

CHAPTER III

SYSTEM REQUIREMENTS

3.1 SYSTEM ANALYSIS AND SPECIFICATIONS

System analysis is the process of formulating a system specification from information about the present or previous system operations and from a set of user requirements for the new system [27]. The analysis phase of system development is preceded only by a feasibility study if such a study is conducted. It is a critical phase because the systems analysis determines the direction of all subsequent system development. Incomplete or incorrect specifications introduce unmanageable errors into the development process. They can make structured design impossible, create untestable requirements, and lead to a system that does not satisfy the user's needs [28]. Indeed, if the system is pointed in the wrong direction during this phase, then progress along that path is actually an accumulation of work that will have to be undone in order to rectify the system, which in turn can break both budgets and schedules.

The specifications that are produced by the system analysis define how the system will meet all its requirements. Many functions of real-time systems can be accomplished either by hardware or by software. Therefore

the analyst must make informed decisions about the partitioning of system requirements to hardware and software functions. He must be able to effectively weigh the cost of purchasing hardware (or a software package) against the cost of developing software.

Regardless of how the functions are assigned, the hardware configuration must be capable of the performance necessary to support the software requirements [3]. The resource capacity of the hardware in conjunction with the resource demands of the software jointly determine how long a single software request for a system resource will have to wait for service. In other words the resource capacity/demand relationship determines the system response time, which is a critical aspect of real-time systems. The increase in response time as the resource demands of the system approach the capacity limits of the resource is not linear as is shown in Figure 34. In the region between A and B the increase in demand results in a relatively small increase in response time. However in the region between B and C the same increase in demand produces a sharp rise in response time. In some cases software can be rewritten, redesigned, or otherwise modified to alleviate a response time problem caused by resource limitations, but in other cases it cannot solve the problem. Even when software modifications can rescue a system from failing to meet its

response time requirements, the changes are usually made during the later stages of system development which make them extremely expensive.

In the past the system analysis phase has been shortchanged, in part because it is one of the most difficult phases of a system's development, and in part because everyone concerned (analyst, user, and management) is eager to get on to more tangible results [27,28]. The process is difficult because in order to achieve a complete and correct description it requires a high level of interaction and agreement between several groups of people, as opposed to later phases that are almost exclusively concerned with technical aspects. Another difficulty of system analysis is accurately defining user requirements that may have never before been put forth formally and that may not lend themselves to technical terms. Ideally the analysis phase should be an iterative one where the analyst is constantly getting feedback from the user about the current state of the specification and making refinements to it. However, in the past, without the aid of functional decomposition tools similar to those used by system developers (top-down design methodologies, structure charts, etc.), the analyst often relied upon the English language as his only means of communicating requirements. This approach often consumed a considerable amount of time before the analyst produced a specification composed of

vast amounts of verbiage that neither user nor developer had the endurance to read in its entirety. Thus iterative feedback was nonexistent. Moreover the monolithic nature of the document made it difficult to modify, so it was rarely updated when the inevitable changes to the system requirements came along.

In recent years an alternative method of approaching the system analysis phase, called structured analysis, has been growing in acceptance and use. Structured analysis gives the analyst some documentation tools with which to partition his requirements and to communicate them in a concise form. Data flow diagrams along with a data dictionary provide the means to create a model of the system that is to be implemented. The basic symbols used in data flow diagrams are shown in Figure 35 [29].

The circle which is often referred to as a bubble (Figure 35) represents a function or process. In a software system it usually represents a program or functional module. A brief descriptive name and a reference number are used to label the function. The two parallel lines containing a data store (Figure 35) represent a place in the system where information can be deposited for retrieval at some other point in time. Typical software data stores are disk files and shared regions of memory. The name or title of the data store is written between the two lines.

The rectangle (Figure 35) represents a source of information coming into the system or a destination of information going out of the system. The source and/or the destination is considered an outside boundary of the system defined by the data flow diagram. A user interface terminal would be a common source and destination in a software system. The name of the source or destination is written inside the rectangle. The line containing data flow (Figure 35) is used to represent the transfer of information between any of the other three symbols. The arrow defines the direction of the transfer and a label identifying the information is written through or next to the line.

The data flow diagram allows the system specification to graphically depict all the functional components of the system and also show the interfaces between those components [27]. "The value of graphics is in getting information across effectively, making simple things obvious immediately and complex things comprehensible"[30]. The data dictionary will contain a definition of each interface, which is a labeled data flow on the diagram, in terms of connecting components. Data flow diagrams lend themselves to hierarchical levels which also helps the analyst to decompose the system into functional pieces. The highest level diagram would present only the main elements and processes of the system and their interfaces. In order to better understand any one of the processes on that

diagram, a separate data flow diagram may be created to show more detail about the process. For example Figure 36 shows a data flow diagram of the main phases of system development. To elaborate on what takes place in a particular bubble each phase may have its own sub-level diagram. Figure 37 is an expansion of bubble number 2, the system analysis phase. Each bubble in Figure 37 may be expanded by a separate diagram to show more detail about the process. The person using the diagrams to communicate decides how many levels are required. In structured analysis a mini-spec is written for each process (bubble) shown on the lowest level data flow diagram(s). The specification of an elemental process provides the detailed information on the process that remains to be communicated.

There are many more techniques and procedures pertaining to structured analysis than have been mentioned. The few basic concepts presented however are sufficient to indicate the advantages over the classical unstructured and undisciplined method of system analysis. The structured approach helps the analyst attack the problem of largeness through partitioning. The data flow diagrams can be generated in far less time than previous methods of documentation and are easily understood by the user. These qualities are conducive to the iterative feedback process between the analyst and the user. A system specification

composed of data flow diagrams, a data dictionary, and a set of mini-specs that correctly and thoroughly describes the system is a valuable asset to system designers and developers, since its form is concise and relatively easy to conceptualize. Such a specification is also a maintainable document.

3.2 APPLICATION SOFTWARE REQUIREMENTS

The format of the software specification for the surge control system follows the structured analysis format for specifications as closely as possible while still satisfying the format required by company software documentation standards [14]. It contains a top level data flow diagram that partitions the software requirements into their main functional processes and shows the interfaces between those processes as well as between hardware components, disk storage, and shared memory. A data dictionary is included that defines each type of interface information shown on the diagram. The purpose and requirements of each functional process shown on the diagram are described in a process mini-spec. Figure 38 (on two pages) is a reproduction of the software specification's data flow diagram and Appendix C contains the data dictionary for that diagram. This section will capsulize the contents of the mini-specs of the functional

processes identified on the diagram and also briefly discuss some additional goals of the software system development.

The terms online and offline will often be used in reference to the mode of operation of software modules and functions in this and successive sections. The term online implies the state of the software system while it is performing its surge control functions. The system is intended to operate in the online mode while the plant compressors are running. The term offline refers to the state in which the software system operates when the compressors are not running and the surge protection functions of the software are inactive.

The main function of Module 1.0 is to put tables of information prepared by the system developer into the shared memory region, henceforth called system common, before any system processing begins. The information should contain (but is not limited to) parameter addresses to be used by the Analogic I/O subsystem, scale factors, codes identifying parameters' engineering units, compressor temperature alarm limits, initial data for all output parameters (valve positions, alarm lamps, and servo power), and appropriate values for system status words or flags in the system common. The information should be kept in files to be read by Module 1.0 and to be modified by Module 8.0 as an offline activity. After Module 1.0 has performed its

initialization function it will start the execution of the other modules in the system that need to wait for the system common to be properly initialized before they begin their processing.

The purpose of Module 2.0 is to provide the interface to the Analogic I/O subsystem. It will handle acquiring data from all input parameters every 50 milliseconds using the DMA modes of analog and digital input. The status of the I/O subsystem will be transferred from the Analogic's CSR to a status word in system common. Digital output will be performed only during the module's initialization and when alarm conditions dictate a change in the output data.

Module 3.0 will communicate with the fifteen CCC anti-surge controllers. It will read each controller's status and four channels of data once a second and place that information in system common. It will also perform any remote programming of the controllers requested by the configuration computer via Module 4.0.

The purpose of Module 4.0 is to provide an interface to the configuration computer. The hardware and thus the protocol of the configuration computer's side of the interface is not yet defined. The communications link between the anti-surge system supervisory computer and the configuration computer will be a serial one whose details should not affect the module's functional requirements. The

primary function of Module 4.0 will be to make the configuration system aware of alarm conditions. When status words in system common indicate invalid temperature data, an anti-surge controller recycle trip or safety on condition, or a compressor temperature alarm, Module 4.0 will initiate a message to the configuration system reporting the abnormal status. In response to a request from the configuration computer the module will send all parameter data, all the arrays containing descriptive information about the parameters (ranges, scale factors, addresses, etc.), parameter validity and alarm limits, all arrays pertaining to valve stroke times, and all manner of status information residing in system common. Module 4.0 will also respond to commands from the configuration system to reset compressor alarms, to request Module 3.0 (controller interface) to perform a function, to move one or more valves to specified positions, to stroke one or more valves and report any abnormal results, and to request Module 8.0 (maintenance diagnostics) to perform a function.

The purpose of Module 5.0 is to fulfill the requirements of the surge control system for another separate software system that will be implemented on the V-Plant's VAX 11/750 computer (referred to as the archival computer by this system). The other system's main purpose will be to gather data from all the plant computers for integration, analysis and archival. All parameter data and

status information obtained by the surge control system will be sent over the plant network to an interfacing module on the the archival computer once a second by Module 5.0. Analog data will be sent in its original counts format. Parameter information needed to convert and display data at a later time will be sent prior to initiating the once-a-second transfers.

Module 6.0 will check for and respond to alarm conditions. The module will monitor the status in system common of all controllers and perform the compressor temperature alarming function. It will turn on the associated alarm lamp of any controller whose status indicates the recycle trip condition, and turn it off when the controller's status returns to normal. When a controller's status denotes the safety on condition this module will flash the associated alarm lamp and will continue to do so (even if the controller's status indicates the safety on condition has been cleared) until the operator resets the alarm. Figure 39 gives a graphical specification of the temperature alarm function. Since the flow of the figure is not unduly complex, the logic will not be repeated here for the reader. The function will be performed for all fifteen compressors every fifty milliseconds. The presence of a temperature alarm is determined by comparing the difference between the inlet

temperature and the internal or discharge temperature to a temperature alarm limit. A timer will be started when a temperature alarm is detected and the time in alarm will be compared to a time limit in system common. If the compressor's temperature has not returned to normal in this limited amount of time then measures beyond just commanding the compressor's bypass valve to open are taken to recover the compressor from surge (Figure 39). This module will maintain a temperature alarm status word in system common for the benefit of other modules in the system.

Module 7.0 will compare the raw (unconverted) counts of temperature data to high and low limits for the purpose of detecting invalid data and reporting the results to other modules through temperature validity status words in system common. Other modules in the system may request Module 7.0 to convert the temperature data to degrees Fahrenheit by setting a request flag in system common. This module will place the converted temperatures in an array in system common for retrieval by the requesting module(s).

Offline diagnostic functions will be performed by Module 8.0 at the request of the operator or the configuration system. Those functions are:

1.) Check (and report) ambient readings of the controllers' differential pressure transmitters and of the temperature parameters' thermocouples.

2.) Exercise the alarm lamps on the manual control console in order to verify their condition.

3.) Position valves at requested positions.

4.) Obtain the average stroke time of valves (for comparison during pre-operation instrumentation checks).

5.) Read and store the valves' positions for full range and zero outputs (to be done after manual valve calibration).

6.) Change any type of alarm limits in the system as requested by the operator or configuration system. The user interface is to be menu-driven.

7.) Send all information pertaining to the I/O parameters to the configuration system or to the archival computer for printing. The information will consist of parameter names, I/O subsystem hardware and software addresses, full scale and zero range limit, validity limits, alarm limits, type codes, and scale factors.

8.) Read the controllers' programmed values, store them in a file, and list them on request.

Module 9.0 will emulate the software system's interface to the configuration computer until the configuration system is implemented. All previously specified functions of interaction with the configuration system will be provided through a terminal interface. Data

and other information will be displayed on the terminal in a tabular fashion.

All parameter data (including anti-surge controller data) and alarm status words will be recorded once a second to a circular file by Module 10.0. The capacity of the circular file will be two hours of data. This function will provide an alternative method of storing operational data until the archival computer's data gathering software is ready and beyond that time when the archival computer is not running.

In addition to the requirements put forth in the mini-specs for the software specification's data flow diagram, the surge control system software will supply a data base of all system parameters that contains all the parameter tables that will be placed in system common at system initialization. A screen editing utility will be provided for ease of modifying the data base. Printing of the data base will be accomplished upon request from the configuration system.

The software for this system should be developed with three general goals in mind: efficiency, maintainability, and flexibility. Efficiency will promote software that executes in a minimum amount of time and that does not waste system resources, such as memory, disk space, I/O processing, and CPU processing. The software should leave

free and available as many resources as possible for future use. The software should be designed such that future modifications do not require system wide changes, but may be localized and require a minimum of change to the code. Flexibility will be especially important to the modules that will handle system interfaces that are presently undefined in protocol (Modules 4.0 and 5.0) and to the modules that interface directly to the user (Modules 8.0 and 9.0). These modules should be designed along functional lines with the ability to place requests for action and/or information from other modules as a general system level request, rather than making the request directly to the servicing module. Likewise these modules should be able to respond to general system level requests to perform a function. This flexibility and generality will ease the adaption of these modules to their future interface requirements. The software system designer should not be limited to these three points of consideration for the future of the software system, but should accommodate future capabilities wherever their possibility is recognized. However, the integrity of known and stated present requirements should not be compromised for unknown future requirements.

CHAPTER IV

SOFTWARE SYSTEM DESIGN AND DEVELOPMENT

4.1 DESIGN CONSIDERATIONS

The software system design translates the software specification into a blueprint for implementation. It specifies and describes the software system's architecture [3]. Yet the design phase has often been neglected and given only cursory treatment during the software development process. Software developers frequently experience pressure to produce something executable that others (predominately managers) can recognize as evidence of progress. They are thus influenced to get through the design phase as soon as possible, if they take the time to do it at all, in order to start coding. Studies have pointed out that design errors outnumber coding errors by roughly two to one when errors detected during the latter stages of software testing have been surveyed [28]. Furthermore it has been noted that design errors require exceedingly more time to diagnose and correct than coding errors. In light of these observations, it is obvious that the design phase of software development should be given due consideration.

Methodologies for improving the design process, referred to as structured design, have made significant

advances in the last ten to fifteen years as the software engineering community has come to realize the importance of the design phase. Unfortunately, their adoption has been rather slow due to the common temptation to rush through the design process and to the lack of software developers that have been schooled in its principles.

Structured design should not be confused with top-down design. Top-down design is an informal method of decomposing the overall problem into simpler components [31]. The process continues to make incremental refinements to the functional components until they are simple enough to be easily implemented. The opposite approach, called bottom-up design, begins with simple functions of the lowest level and proceeds to combine them into more complex functions until the system requirements have been met. A limited number of authors of software engineering literature hold that either of the top-down or bottom-up methods of design may be used during structured system design. However, the pioneers and recognized authorities of structured design, such as Edward Yourdon, Larry Constantine, Tom DeMarco, Michael Jackson, Glenford Myers, and others, do not consider bottom-up design to be compatible with the structured doctrine.

Neither should structured design be equated to structured programming. Structured programming is

considered to have begun in 1965 with Edsger Dijkstra [32]. It has since evolved into a well-defined method of applying top-down design to code development, using only three logical elements for code implementation: sequential instructions, the decision construct (if-then-else), and the loop construct (do-while). Structured programming applies to software modules or relatively small systems and does not address the overall structure of medium to large systems.

While structured design does not contradict any of the postulates of structured programming or top-down design, it is not a reiteration of those methods [31]. It is a major extension of those concepts that provides a disciplined approach to overcoming the complexity of large software systems. Structured design has one main objective, which is to produce a design of high quality, but its approach is multilateral.

Structured design postpones decisions about how a problem is to be solved until the problem has been sufficiently defined, thus allowing the nature of the software requirements to determine the form of the software system. It uses a variety of graphic and other forms of tools to make the design easier to understand. Many tools used for structured analysis, such as data flow diagrams, data dictionaries, structured English, decision trees, and decision tables, are also used for structured design, as

well as pseudocode and structure charts. Structured design attacks complexity by partitioning the system into black boxes and organizing them into hierarchies. The term black box is borrowed from the electronic engineering field and implies a piece of software whose inputs, outputs, and function are explicitly defined but whose internal processing is unknown. This partitioning into black boxes increases the flexibility of the system by increasing interchangeability and by making modifications easier [28]. Organizing the black boxes into hierarchies separates the working modules from the management modules and allows the working modules to be grouped (like departments of an organization) by their need for interaction with one another.

Evaluating a system design has traditionally been plagued by subjectivity. Structured design can help alleviate that problem by establishing a set of criteria by which the design should be judged [31]. For instance, how well the partitioning has been done can be based on the measure of coupling and cohesion that modules exhibit. Coupling is the degree of interdependence between modules. Minimal coupling is desired in order to allow a change in one module to affect as few other module interfaces as possible, in order to allow a module to be changed without being concerned about the impact on the internal details of

another module, and in order to minimize the risk of an error in one module appearing as a symptom in another. Cohesion is the degree to which activities inside a module are related to each other, also known as module strength. A high degree of cohesion is desired in order to keep modules partitioned along functional lines and thus increase the ease with which they can be modified. In essence, the criteria of coupling and cohesion determine how cleanly the modules are separated from each other.

Maintenance is a traditional problem of software systems, typically consuming twice the cost of development [31]. The problem of software maintenance has several sources, some more easily addressed than others. Transferring the responsibility of a system's maintenance to different software personnel is troublesome because the new maintenance person must accommodate a style of design and programming that is unfamiliar [33]. Different programmers will create different solutions to the same problem. If the maintenance programmer is not the same person that developed the software, he must take the time to adjust his thought processes to those of the developer and also take the time to understand how the system currently works. Failure to expend the time to do this before he begins to make modifications can result in unexpected repercussions. The changes may adversely affect a part of the system that the programmer does not

anticipate. But it is usually quite difficult for maintenance programmers to gain an adequate understanding of the present system and to accurately assess the impact of a proposed modification on the rest of the system when dealing with conventional systems. This difficulty arises from the typical neglect of system design and the lack of high-level documentation. Structured design addresses all these maintenance woes at the design phase, which is where most of them originate. Just as it provides a common ground for design evaluation, it provides a common method for arriving at a problem's solution, which helps to smooth the differences in personal styles. Most obviously it would eliminate the problem of the lack of system design. Finally, the tools used to create a structured design are a significant aid to high-level documentation.

Although no one static process of software design can satisfy the needs of all software development because of the vast spectrum of software products, the concepts of structured design can be applied to all types of software [34]. This discussion of structured design is meant only to give the reader a very brief glimpse of its variety of benefits. For details about applying the methods of structured design and a more comprehensive treatment of the subject, the reader is referred to the considerable amount of literature that has been published on structured design.

The one major criticism of structured design leads to the more specific concern of real-time systems design. That criticism is inefficiency, which is of primary concern to real-time designers. Proponents of structured design point out that in terms of production costs the method produces systems that are more efficient than traditional methods, or rather the lack of them [31]. That aspect of structured design efficiency can be verified, but the advocates do exhibit a disregard of efficiency aimed at processing speed. This is not to say, however, that real-time design and structured design are totally incompatible. The benefits of clarity, simplicity, functionality, documentation, maintainability, and flexibility to be gained through structured design are just as desirable and advantageous to real-time systems as any other type of software. However, the real-time designer must be aware of the characteristics of his system that will require special attention.

System primitives should be identified at the beginning of the design phase [3]. System primitives are low-level modules that interface to the operating system or more often to external hardware components, whose function and feasibility are critical to the system's requirements and implementation. Higher-level modules will be designed under the assumption that the primitives will work correctly, so that assumption should be verified as early

as possible. This is not a reversion to bottom-up design, but a necessary variation in the application of structured design to real-time systems.

The real-time designer should consider how the system is going to react to overloaded resources and take measures to prevent or handle those conditions without adverse effects on response times. A resource that reaches saturation can create a bottleneck that restricts the performance of the remainder of the system. Error conditions of all types should be identified and the software's response to them should be managed so as to allow the system to continue as normally as possible.

The relative priorities at which modules will be executed should be considered as they are designed, in association with the asynchronous aspects of the system. The designer should account for the occurrence of unexpected asynchronous events, the occurrence of too many expected asynchronous events, and an expected event that never occurs. Although he may not be able to control the occurrence of these situations, he can, through design, control the manner in which the system will respond to them. Because real-time systems are usually composed of more asynchronous activity than conventional software systems, they are particularly susceptible to race conditions. A race condition occurs if the order in which

two (or more) processes are executed can change the computational results of one or more of the processes [35]. Therefore the real-time designer should recognize potential race conditions and resolve them through the use of queues, semaphores, priorities, or other means of process synchronization [3].

Due to the time constraints placed on most real-time software systems, unnecessary I/O processing should be avoided. The processing of real-time data is of course necessary, but that too can be minimized by not performing any data conversion unless it is required. The overhead processing of normal I/O operations (disk drives, terminals, etc.) consumes an enormous amount of time relative to the real-time activities. Total avoidance not being practical, buffering the data that must be passed to or from I/O devices until significant quantities are stored can greatly reduce the amount of time spent on I/O accesses. In other words, the designer should make the I/O access accomplish as much as possible. Also, given that computer memory has become relatively inexpensive, resulting in ever increasing amounts of memory available on a given class of computer, the system should not have to read files once it has entered a state of responding to real-time events. The designer should arrange to have any information that will be needed from disk files transferred to memory before real-time execution begins.

The phases of software development that follow the design phase, coding and testing, are dependent upon the thoroughness and quality of the software design for their success [28]. The design phase presents the last chance to eradicate any deficiencies found in the software specification before they become exorbitantly expensive to resolve.

4.2 APPLICATION PRELIMINARY DESIGN

During preliminary design, the functional requirements that were set forth by the software specification are analyzed and software programs that will accomplish those requirements are identified. The programs' purposes and processing should be described on a high level. The relative priorities of the programs should be established, asynchronous activities (hardware interrupts or software events) that will affect the programs should be identified, and any necessary synchronization between the programs should be specified. The programs' interfaces should be defined well enough to identify and describe all of the software system's data storage articles and their contents (or at least their intended contents).

Data storage for the anti-surge control system took the form of reserved areas of computer memory and disk files. The memory storage areas will of course only exist

while the programs are running, and take the form of system commons. Disk files include those files that are created by the software designer for the primary purpose of system initialization, which may be subsequently modified by the system's users, and those files that are created as a function of task execution.

4.2.1 Defining Data Storage

A system common is an area of memory that is reserved for use by one or more programs, and may be defined as having read only access or read/write access. It may be allocated dynamically by a program or it may be what is called a resident common, in which case it is recognized by the RSX operating system as a task which must be linked and loaded into memory like all tasks. All common areas used by the surge control software system are resident commons. If a task will need access to a system common it must specify that at the time the operating system links all modules of the task for execution. Any number of tasks may link to the same system common and thus share that area of memory. A system common requires a minimum of one APR (Address Page Register) in a task's address space (4K words) regardless of how much of that address space is defined or used by the task. If the defined contents of the common exceed the size of one APR, then multiple APRs of the task's address space

are reserved for the common, the number of them dictated by the size of the contents of the common. Therefore any task that is going to link to a system common must have the required unused address space available. However, since the introduction of the I and D option (section 2.3.1, page 36) of linking tasks, most tasks do not have any problem accommodating that requirement.

The system common is the central and most important data storage facility in a real-time computer system executing under the RSX family of operating systems. That is not to say that application software cannot be written without using a system common, and it should be pointed out that the coupling criterion of structured design takes a negative view of the use of commons for the following reasons:[31]

1.) An error in one program may show up as a problem in any other program that uses the common.

2.) Programs that use variables in a common do so by name, so in order to use the program in another application it will have to use the same variable names.

3.) Commons may be abused by using the same areas to store different types of information by different programs.

4.) Maintenance of programs using large amounts of common data and variables is difficult due to the

uncertainty of what portion of the common is used by a particular program and the uncertainty of which variables in a particular program are shared by other programs.

5.) Changing a piece of common data in one program requires all other programs that use that piece of data to be changed also, and determining which programs use the data may be difficult and time consuming.

6.) FORTRAN commons refer to variables by their relative location from the beginning of the common area regardless of their name, which means that programs must know and comply with the size of all variables in the common that precede those used by the program.

All of these points have merit and bear recognition by the software developer. Nevertheless, proper and diligent management of the definitions of the variables that reside in the common can control these disadvantages. Many programming languages, such as FORTRAN, C, and others, provide the ability to insert variable declarations contained in a separate file. By restricting the definition of variables in the common to one such file, requiring all programs that access the common to use that definition file, and adequately documenting that file, many errors,

particularly those implied by the sixth point, can be avoided. Using the indirect command file feature of the RSX operating system, a file of commands to compile and link all of the software system's application programs can be created and executed any time a change to the common's variables is required, which will avoid the problems of the fifth and sixth points. Program code banners should contain a section that identifies all common variables accessed and/or changed by the program which, while not eliminating, will reduce the disadvantage of the fourth point. Problems arising from the third point should not occur, because it describes an inferior programming practice that seems to be aimed at saving memory space. The second point seems to be a relatively minor drawback. The problem of the first point will be minimized if the programs are properly partitioned and other aspects of low module coupling are observed.

Designing a real-time system that does not allow data to be shared is not practical in any sense. Methods of sharing data (and control flags), other than the use of commons, all require overhead processing by the operating system. System commons do not use any overhead processing; programs merely address the same memory location. Hence, while other types of software applications may have valid reasons to abstain from the use of common areas, neglecting the use of such an efficient structure for sharing data (and task-to-task communication) in real-time applications

would almost certainly degrade the system's performance.

Three system commons were defined for the surge control system. The common which contains the real-time data, status information variables, and task control variables is called RESCOM. The relatively large arrays to be read into memory from the data base file and other large infrequently used arrays were assigned to another common called DBSCOM. The third common, RCDCOM, is used to buffer data that is stored in the file that holds two hours' worth of once-a-second data. The common RESCOM was intentionally restricted to one address page in size (4K words) in order to not deplete the address space of the tasks, as all programs in the system will need to access this common. Using two address pages for this common would have had adverse effects on the system tasks which have to link to another system common in addition to RESCOM. The common DBSCOM needed two task address pages (8K words), but only a few programs would be linking to it. RCDCOM is the largest common, requiring three address pages (12K words), but only two tasks link to this common.

Five types of disk files were identified during preliminary design as being required for the anti-surge control system: a system common file, a controller constants file, a data base file, a system values file, and data log files.

The system common file was intended to provide a copy of the system common RESCOM as it existed at the beginning of the system's online mode of operation. It was designed as a sequential file whose primary purpose was to provide a means during system development to verify that the common contained the proper values.

The controller constants file was created to hold all the control parameters that can be read from the CCC anti-surge controllers, as well as the control constants to be downloaded to the controller when the compressor flow range changes from low to high or from high to low. It contains a set of parameters and constants for each of the fifteen controllers. An example of the file (the values are subject to change) is given in Appendix D. The file uses indexed organization with keyed access in spite of the fact that indexed, keyed access files require more file system processing than other types of files. The benefits to be gained from using a more efficient type of file access and organization would be nullified by the application program becoming larger and more complex, in order to restore the required flexibility of the file. Also, any access to the file while the system is running in the online mode will be for display or printing and the programs that perform those functions will execute at relatively low priorities.

The data base file contains descriptive information about the system's I/O parameters. The fundamental form of

the data base was taken from another system that was currently being used and it was modified and expanded for use by this system. Its format was directed toward compatibility with the commercial data base software known as dBASE which is marketed by Ashton-Tate to run on IBM or compatible personal computers. Direct compatibility with the dBASE software was not possible but programs were written to perform translation in both directions between the dBASE format and the RSX file format. The software utility called Kermit is used to transfer the files between the PDP-11/73 computer and an IBM compatible PC. The data base was designed to be used as a standard for future systems developed by the same engineering and software group, and thus is more general than was necessary for the surge control system. Like the controller constants file, this file uses indexed organization with keyed access in order to achieve the required flexibility. The basic format of the data base hinges on one file record for each parameter - a parameter being a distinct input or output channel of the Analogic I/O subsystem. Each record contains the same number of fields, with fields defined to accommodate both analog and digital types of channels, such that all record fields will not be applicable to any one type of parameter. The decision was made to combine analog and digital parameters in the same data base after

examining how the data base would be used. It was concluded that the convenience of combining the parameters outweighed the fact that additional storage space would be allocated but unused. Each record of the data base contains the following fields:

* A parameter number (for analog and digital channels),
* a conversion type code identifying what the parameter's engineering units are and what type of instrumentation the parameter uses (for analog and digital channels),
* an alphanumeric parameter description of up to thirty-two characters (for analog and digital channels),
* the parameter's Analogic I/O subsystem address separated into chassis, slot, and channel numbers (for analog and digital channels),
* a logical indication of whether the parameter uses the differential or single-ended input mode (for analog channels),
* a programmable input gain (for analog channels),
* the number of consecutive times the parameter must read above its alarm limit (for analog channels) or be in its abnormal state (for digital channels) before the alarm is recognized by the system,
* a logical indication of whether the parameter is

capable of generating an alarm condition (for analog and digital channels),

* a logical indication of whether the normal state of a parameter is zero or one (for digital channels),
* the parameter's minimum engineering units value based on its instrumentation (for analog channels),
* the parameter's maximum engineering units value based on its instrumentation (for analog channels),
* the parameter's minimum raw counts based on its instrumentation (for analog channels),
* the parameter's maximum raw counts based on its instrumentation (for analog channels),
* the scale factor for conversion from counts to engineering units (for analog channels),
* the parameter's warning alarm level (for analog channels, if applicable),
* the parameter's danger alarm level (for analog channels, if applicable), and
* a coefficient to be used in a software filtering algorithm (for analog channels).

Appendix E contains a printout of the data base file. All the fields of one record could not be printed horizontally across one page. Approximately half of a record's fields are displayed on the first three pages for the entire parameter list. The next three pages display the remainder

of the fields with the parameter number repeated at the beginning of each line for identification.

The system values file contains values that are not included in the data base file that need to be assigned to system common variables at the time of system initialization. They all pertain to stroking the bypass and discharge valves or to the surge alarm function. For example, the expected valve stroke times against which new stroke times will be compared and the differential temperature alarm levels (between compressor inlet and discharge temperatures) are kept in this file. The initial values of these kinds of variables are kept in a file because they will be modified from time to time by the user or by a program. The file uses sequential organization with fixed length records so that it may be accessed sequentially or directly.

Data log files hold all the system's parameter data saved at the rate of once a second, status variables saved out of system common, and a record of certain operator commands. The files provide a means of retaining this information until the software that is to send the same information to the archival computer over the plant network is further specified and developed. After that software has been implemented the files will serve a backup function in the event the archival computer cannot receive the information while the surge control system is online. The

once-a-second data is stored in a file that will hold two hours worth of such data. When the file is filled it will be overwritten in a circular manner. The file will be closed and saved at the user's command and a new version opened for continued recording of once-a-second data. The data log files are sequentially organized and accessed.

4.2.2 Defining Programs

In addition to defining and describing the system's data storage, preliminary software design must define the system's functions in terms of specific programs that will accomplish those functions. The manner in which the system's specified functions are allocated to separate programs will have a prevailing influence on all other phases of the software system's life cycle. It will be either an asset or a liability to the software's detailed design, coding, testing, and maintenance.

Because the software specification defined the system's requirements in terms of functional components, the ten major functions described by the software specification's mini-specs and data flow diagram (plus the requirement for a modifiable data base) provided a natural beginning for program definitions. Those definitions were analyzed with two main objectives in mind: a function should not be performed by more than one program, and all

I/O for a given device should be done by the same program, with the exception of disk I/O.

The data flow diagram of the anti-surge control system that was generated at the conclusion of the preliminary software design is shown in Figure 40 (on three pages). Comparing that data flow diagram to the one from the software specification (Figure 38) will illustrate the refinements that were made between the first attempt and the final form of program definition.

The initialization function was defined as a program with the addition of the requirement that it be executed again at the start of online processing in order to ensure that the system commons are in a known state and contain the desired values for the beginning of online operations.

The Analogic I/O subsystem functions were defined as a program with one change in specifications. Instead of output data being sent to the I/O subsystem only when a change in the data values is required, it will be sent at regular intervals (at the same rate as data is being input). The extra processing incurred by this change will not affect the system's overall performance, and the benefit is that of eliminating the need for communication between this program and any other program that needs to have new output data values sent out. Other programs can simply place the new output values in the output buffer in the system common area and they will automatically be sent

to the subsystem during this program's next execution cycle.

The CCC controller interface functions were defined as a program with a few refinements. A diagnostic test of controller communications will be performed when the program is started and the results of the test stored in system common. Reading of any controller values other than the status and four input channels (which are read and stored in common once a second) will not be done on regular intervals, but on a request basis.

The surge alarm functions and the data validation functions were defined as one program. In the software specification the alarm functions stood alone and the validation functions were combined with data conversion. The reason for the new combination of surge alarm and validation functions is that before checks can be made for alarm level temperatures, the validity of the temperature data needs to be established. More specifically, the severe responses brought about by the detection of a temperature alarm should not be allowed to happen because of faulty instrumentation. It may be noted that combining these two functions in the same program does not violate the previously stated goal of not having a function performed by more than one program. It does, however, breach the structured design criterion of module cohesion (section

4.1, page 77). But if the data validation functions were assigned to a separate program, that program would have to execute at the same rate as the Analogic I/O program and the surge alarm program, which is twenty times a second, in order to provide validated data for the alarm program. The added burden that would be placed on the system by the increase in context switching (operating system overhead of changing current tasks), created by another program executing every 50 milliseconds, was considered sufficiently undesirable to justify combining the functions in one program. A further refinement of this program's definition was the expansion of data validation to include the valve position readings and the servo amplifier power readings in addition to the temperature readings. The validity of the valve positions and servo power indications will be checked only once a second, rather than the twenty times a second rate of the compressor temperature data.

The function of converting data from digital counts to engineering units was defined as a program. Its definition was enlarged to include conversion of all analog data instead of just temperature data. The program will convert data on request basis, with the list of raw (counts) data and the list of converted data being passed through system common.

The software specification's requirements for the configuration computer functions, the emulation of those

functions on a terminal, the diagnostic functions, and a modifiable data base were closely related, although they contained relatively diverse lists of activities within each of their specifications. Therefore, their functional requirements were conceptually combined and the sum repartitioned to yield five programs: a configuration computer interface program, a program to handle terminal input, a program to handle terminal output, a program to perform functions that involved (non-alarm) valve movements, and a program to format data in a manner suitable for viewing on a terminal screen or for hardcopy.

The functions pertaining to valve movements were considered substantial enough to warrant a separate program. This isolation of the functions allowed requests for a particular type of valve movement to be placed through system common regardless of the origin of the request. The program will perform the following four functions:

1.) Obtain and store new zero and full-scale position readings of valves.

2.) Obtain and store new average stroke time limits for valves.

3.) Stroke valves and report.

4.) Position valves and report.

The program will place the results of any requested valve

movement function in system common for retrieval by the requesting program.

The keyboard input program will accept commands from the user's terminal, verify the command as being valid, and then carry out the requested function. The program will handle commands to display and/or modify the data base, commands to perform a diagnostic function, and commands that will later be made via the configuration computer interface. Most of the commands will be passed on to other programs to perform (controller functions, for example), but functions not implemented in another program, such as resetting alarms, will be performed by this program.

The terminal output program will display menus to aid the user in selecting valid commands and any information requested by the user. The type of information that can be displayed includes reports from valve movement functions, files stored on disk, controller parameters, system status conditions (validity, alarms, etc.), and current data as it is being read by the I/O subsystem. The program will essentially be controlled by the keyboard input program. A screen number placed in system common by the keyboard input program will dictate what type of display the terminal output program generates.

The keyboard input program and the terminal output program were first defined as one program, but it was determined that the processing required to generate and

update display screens would degrade the response time to users' requests to an unacceptable degree. As separate programs, the keyboard input program can perform a function that does not require any output to the terminal screen while the terminal output program updates the most recently requested display.

The software specification's requirements for the configuration computer interface was redefined as a program, with one refinement. Certain parameter data (to be specified in the future) and status information will be sent to the configuration computer at regular intervals, rather than only when an alarm condition is detected. Commands from the configuration system will be dispatched to other programs for execution in the same manner as commands from the terminal keyboard.

The function of formatting files for output to the user's terminal or to a printer was defined as a program. The program will respond to requests passed through system common to format the data base file, the system common file, the controller file, or a data log file. A flag in system common, set by the requesting program, will tell this program what to do with the formatted file - whether to send it to the printer or place it back on disk for the terminal output program. File formatting was assigned to a separate program because of its size and speed. The

formatting process will require rather lengthy and tedious code, and it may require more processing time than would be acceptable in other programs. The program's independence will allow formatting to be done at a low priority and also make the functions available to all other programs in the system.

A program was defined to record files on disk. This function does not pertain to files that are already stored on disk, such as the data base file or the system values file. The data logging function described in the software specification was assigned to this program with the refinement that certain operator commands will be recorded along with the once-a-second data and status words. The program will monitor a flag in system common and respond to the flag being set by saving the current data log and creating a new one. This program will also record the system common at the beginning of the system's online mode.

The software specification's requirements of sending data and status information to the archival computer over the plant network were defined as a program. They were refined to include certain operator input in order to be compatible with the recorded data logs, since the data logs will be used as a backup to these requirements when this program is implemented.

4.3 APPLICATION DETAILED DESIGN

After the anti-surge control software system's requirements were assigned during preliminary design to programs to be developed, the detailed design of those programs expanded their descriptions prior to code development.

The system's critical interfaces were identified as the Analogic I/O subsystem program and the CCC controller program. Those programs were designed and prototyped early in the detailed design phase to a degree sufficient to verify their feasibility.

Files containing information that may need to be accessed by the system were designed to be made available in memory, specifically in system commons. They will be transferred to memory during system initialization to avoid any further disk accesses to obtain the information.

All input to the system, from files, the I/O subsystem, the controllers, or the user is checked for errors before it is made available for processing. If time allows, as with the input from files at initialization and with user input, a program will notify the user of erroneous input and insist on correction before continuing. However, the more critical nature of the programs receiving input from the I/O subsystem and the controllers does not allow them to demand correct data before continuing.

Therefore, the erroneous input is discarded, the user is notified of the errors, and the programs continue to obtain new input data.

Synchronization between the programs is treated on an individual case of need, and takes the most appropriate of several forms. Using variables in system common as control flags is the method used most often, and these control flags may themselves take various forms. The next most often used method is the operating system's global event flags. In a few cases send/receive system directives are used to establish a private communication buffer between tasks. System directives are also used for a task to issue a command to run another task or to abort another task.

This design's approach to modularity within the programs was guided by the system's real-time considerations. Efforts were made to decrease the complexity of programs through modularity, but the methods of implementation are often different than those typical of structured programming. Structured programming advocates the use of subroutines and procedures to achieve modularity. However, the additional processing incurred by calling subroutines, particularly those with several arguments, can degrade the performance of a time critical program. Therefore, alternate methods of breaking code into sections were applied, such as computed GOTOs, when the program's execution time was crucial. The normal

application of subroutines was used for lower priority programs.

Another factor that strongly affected the detailed design of the programs was the intent to make them generic enough to be used in future systems with minimal modifications. That criterion influenced the design of all the programs in this system with the exception of the program that detects and responds to surge alarm conditions. The probability of the requirements of that program being applicable to another system was considered to be very low, and the time restrictions placed on the program reduced its flexibility.

The software system's final data flow diagram is shown in Figure 41 (on three pages). Although this diagram is quite similar to the one in Figure 40 that was developed during preliminary design, it does contain some notable program refinements that were a result of the detailed design. The reader may refer to the diagram as the programs are described in the following sections. The programs that will provide the interfaces to the configuration computer system and to the archival computer system are not included in the program descriptions below. The requirements of those programs had not been specified beyond those stated in this system's software specification, which was insufficient for further program development. They remain

on this system's data flow diagram, however, because they are still considered to be a part of this system that will be implemented as soon as their interfaces are defined.

4.3.1 Initialization

The initialization program executes prior to all other programs in the system. Execution time is not a concern because the program will execute once and then exit. Its priority will be relatively high. The main purpose of the initialization program is to put known values in all the variables of the RESCOM and DBSCOM system commons. The other purpose of the program is to initialize the global event flags used for system synchronization and to start the programs in the software system that will not exit until the entire system is stopped or until a new initialization is performed.

Values stored in the system values file are read into their appropriate locations in RESCOM and values stored in the CCC controller file are used to initialize many of the controller associated variables of RESCOM. The parameter information stored in the data base file is used to initialize some arrays in RESCOM directly and it is used to compute the values of other RESCOM arrays, such as the I/O subsystem addresses and scale factors. The data base file also directly provides the values used to initialize the vast majority of the DBSCOM common. After the program has

completed the initialization of the system commons and the setting of global event flags to their initial states, it starts all other programs in the system except the user interface program and then exits.

If the initialization is being done prior to the system running in the offline mode, the user interface program will be started by a command file after this program exits. If the initialization is being done prior to the system entering the online mode, the user interface program will already be active.

4.3.2 Analogic I/O

This program provides the software system's interface to the Analogic I/O subsystem. It is the highest priority task in the system and it executes every 50 milliseconds. The program is written in MACRO-11, which is the assembly language of the DEC PDP-11 computer family. The use of assembly language was necessary in order to directly address the registers of the Analogic I/O subsystem and to access the program's Page Address Registers, which are located in the executive's I/O Page (section 2.3.1, page 39).

The program initiates three DMA transfers of data during each execution cycle: analog input, digital input, and digital output. Analog data in the form of digital

counts is output using the digital output mode. Before each data transfer the DMA registers are loaded with the starting addresses of two arrays in the RESCOM common. One array contains the subsystem addresses of the channels to be read or to be output. The other array is the data array which will receive data in the case of DMA input or will contain outgoing data in the case of DMA output. When the registers have been loaded, the desired transfer mode is set in the Analogic CSR (Control and Status Register) and then the GO bit in the CSR is set high, which tells the subsystem to perform the indicated DMA operation. The program then checks the READY bit and error bits of the CSR to determine when the transfer completes successfully or to detect an error condition. If the transfer does not complete within the expected time interval the operation is considered unsuccessful. An error condition or unsuccessful transfer causes the program to move the CSR to the Analogic I/O status word in RESCOM, to alert other programs to the problem. If the transfer completes satisfactorily, the program continues on to perform the next type of data transfer. If all three types of transfers complete successfully, the Analogic status word in RESCOM is set to zero to indicate a normal status condition.

Both types of data input, analog and digital, use double buffers. This means there are actually two arrays in RESCOM for receiving analog data and two for receiving

digital data. Current buffer pointers, also in RESCOM, are toggled by this program at the end of its execution cycle to point to the most recently filled data arrays. The next execution of this program will transfer data to the alternate array (the one not being used by other programs). If for some reason the data acquisition of the current program cycle was not successful, the buffer pointers will not be toggled in order to allow other programs to continue to use the most recent "good" array of data. At the same time, the Analogic status word in RESCOM will inform the other programs that new data has been delayed.

The DMA registers require twenty-two bit physical addresses rather than the program's sixteen bit virtual addresses. Therefore the RESCOM array addresses that will be loaded into the registers are translated from virtual to physical addresses, using the method of address translation discussed in section 2.3.1 on pages 37 and 38, and stored in variables as part of this program's initialization.

4.3.3 Controller Communications

This program provides an interface to the fifteen CCC anti-surge controllers for this system and for the configuration system to be implemented in the future. It will execute once every second and will be the third highest priority task in the system, below the Analogic I/O

program and the data validation and alarm program. Communication with the controllers is divided into seven types of operations:

1.) Read a controller's mode byte (this byte contains the auto/manual status and the manual override on/off status).

2.) Read a controller's status byte (this byte contains the recycle trip status and the safety on status of the controller).

3.) Read a controller's inputs (differential pressure, differential flow, and a flow range reading for each).

4.) Perform the diagnostic communications test on a controller.

5.) Send a new setpoint value to a controller.

6.) Send a new control constant to a controller (one of eight types of constants).

7.) Read all of a controller's information (this includes the controller's inputs, mode byte, status byte, setpoint, process output variable, and all control constants).

All parameter information received from a controller as a result of any of these operations is passed on to the RESCOM common by this program.

The program will perform the diagnostic communications test on each of the controllers as part of its

initialization (and store the results in RESCOM). It will also read all information available for each controller into arrays in RESCOM during its initialization. This information may be presented for viewing and modification by other programs and it will be used for pre-operation checks to be performed prior to the system's online mode. The program will then enter a once-a-second execution loop during which it does three main activities.

The first action is to read each controller's mode, status, and inputs.

The second action is to examine an array in RESCOM to see if any other program has requested a controller operation to be performed. An array is used in order to allow this program to perform more than one request during its execution cycle. Thus any program requesting service will not have to wait longer than one cycle of this program to retrieve the results from system common. For each operation request the program finds in the array, it will obtain a list of controllers from a different but directly associated array in RESCOM for which the operation is to be performed. The program will perform operations as requested until it reaches the end of the request list. Any of the operations listed above may be requested by other programs, although to request one of the first three would be redundant since they are automatically performed each time

this program executes. Remote programming of the controllers is implemented through this structure of requesting operations.

The third main action of the program's cycle is to check the flow range input of each of the first and second stage (axial compressor) controllers for a change from high to low or from low to high. If the range has changed from the previous program cycle, some of the controller's control constants are changed to accommodate the change in flow range. The constants to be sent to a controller in this situation are stored in RESCOM. There is a set of high range constants and a set of low range constants for each of the fifteen controllers. These constants are also stored in controller file (Appendix D) for ease of modification.

The diverse and sometimes repetitious functions of this program were separated into subroutines that were then combined into a library for this program. Arguments are passed to these subroutines through a common area used only by this program. This was done in order to avoid passing numerous arguments in subroutine calls, thus yielding faster execution of a necessarily time critical task.

4.3.4 Data Validation and Alarms

The data validation and alarm checking program puts raw analog data through a software filter, checks the validity of all analog data, checks the alarm status of all

compressor temperatures, and provides the second level of surge protection for the plant compressors. It is the second highest priority task in the system, lower than only the Analogic I/O program, and it executes every 50 milliseconds. The program's execution is controlled by the Analogic I/O program in order to ensure the proper synchronization between the two programs. At the end of each pass of the program it waits for an event flag to be set by the Analogic I/O program before resuming execution.

All analog data input by the I/O subsystem is filtered with a first order filter and placed in an array in RESCOM for the benefit of programs that display data to the user, in order to reduce what is commonly called "noise" in the instrumentation signals. An array of filter constants in RESCOM, one for each analog input, allows the degree of filtering to be adjusted for individual parameters. The surge alarm code uses raw data rather than filtered data because the filter masks sudden changes in the data.

The voltage level inputs of the bypass and discharge valves' servo amplifiers are compared to a limit value. If the reading is below the limit value the servo amplifier's power is considered to be off. Two words in RESCOM are used to show the on/off status of the servo amplifiers' power, with one bit representing the condition of one servo amplifier. One word holds the status of the fifteen bypass

valve servos, and the other word holds the status of the six discharge valve servos. The program updates each servo amplifier's representative bit in its on/off status word once a second, as opposed to the rate of every 50 milliseconds at which the rest of the program is executed.

Valve position readings are compared to a high and a low validity value and their validity status is updated. If the position reading exceeds either the high or low validity limit, it will be considered invalid. Following the format of the servo power status words, two words in RESCOM are used to indicate valve position validity, one for bypass valves and one for discharge valves.

All compressor temperatures are compared to individual low range limits and their validity updated. There is no need to check the high end of the range because the instrumentation limit is the same as that of the A/D converter. Once again, two words in system common are used for the validity status of temperature data. One word holds the status of the fifteen inlet temperatures and the other holds the status of the fifteen internal and discharge temperatures (the first six stages are internal, the remaining stages are discharge).

After the analog data has been filtered and checked for validity, the program returns to the top of its execution cycle if the system is in the offline mode. If

instead the system is in the online mode, it continues on to perform the following alarm functions.

The program checks for temperature alarm conditions by comparing the difference between each compressor's inlet and internal/discharge temperatures to that compressor's differential limit value. If one or both of the compressor's temperatures is invalid, the temperature difference is not checked. If the compressor's differential temperature is greater than the limit, the compressor has a temperature alarm. The compressor's bit in the temperature alarm status word is not set, however, until the alarm has occurred N consecutive times, where N is the compressor's corresponding element in an array of integers.

The three alarm condition status words, the recycle trip status, the safety on status, and the temperature alarm status, are checked for any abnormal conditions. If a compressor's recycle trip status is on, the program turns on the corresponding console alarm lamp. The lamp is turned on by placing the appropriate value in the output data buffer that is sent to the Analogic subsystem. When the recycle trip status of the compressor returns to the off condition, the program turns off the console lamp by again changing the data in the Analogic output buffer. If a compressor's safety on status bit is set or if its temperature alarm status bit is set, the program flashes the corresponding console alarm lamp by alternating the

lamp's data value in the Analogic output buffer. The alarm lamp is flashed until that compressor's alarm reset status is set, even if the compressor's safety on status and temperature alarm status are both cleared.

When a compressor's temperature alarm status is recognized, the compressor's bypass valve is opened by placing the appropriate data value in the output buffer that is sent to the Analogic subsystem. In addition, an alarm timer for the compressor must be started.

Each compressor will have an alarm time limit stored in system common. If the compressor's temperature alarm has not been cleared within its alarm time limit, the position of the bypass valve is checked. If the bypass valve is not open (or at least 90% open), power is removed from the valve's servo amplifier by changing the amplifier's output data. Removing power from a valve's servo amplifier forces the valve to a mechanically biased open position.

When a first or second stage (axial) compressor exceeds its temperature alarm time limit, the compressor's discharge valve is closed as soon as the open position of the bypass valve is verified.

Any changes in the output data of a compressor's bypass valve, servo amplifier power, and discharge valve that are made in response to a temperature alarm are maintained until the compressor's alarm reset status is

set, regardless of the temperature alarm status being cleared. When the reset is received, the discharge valve is opened first, if it had been closed. When the discharge valve is opened, servo amplifier power is restored, if necessary, and the bypass valve is closed. The alarm reset status word is cleared as the program's last action.

4.3.5 Data Conversion

The data conversion program converts analog data from counts to engineering units. Other programs that want to obtain converted data will prepare a list of parameter array indices which is passed to this program through system common. The program will use the array indices to access corresponding elements of the parameter type code (degrees F, volts, etc.) array, the raw counts array, the minimum counts array, the minimum engineering units array, and the scale factor array, all stored in the system common area. The parameter's type code will direct a computed GOTO to the correct conversion equation. For linear conversions the equations will right-justify the raw counts, subtract the minimum counts from the raw counts, multiply the counts remainder by the scale factor, and add the minimum engineering units. For non-linear conversions, such as temperatures, a curve-fitting equation is applied after the counts have been right-justified and zeroed. The converted engineering units value is placed in an array in system

common to be used by the program that requested the conversion.

The data conversion program also converts engineering units data to counts data. Programs may request this type of conversion by using the appropriate request code. This type of conversion is primarily requested by the initialization program in order to obtain alarm limits in the form of counts. Alarm limits are given in engineering units in the data base file, but the alarm checking is done in counts to increase efficiency.

Once a second the program checks its request word in system common to see if any requests have been made.

4.3.6 Valve Movement

The program that moves the bypass and discharge valves will execute on a request basis and will perform one of four different functions. The four functions are:

1.) Obtain and store current zero and full-scale positions of valves.

2.) Obtain and store new average stroke time limit(s) for valve(s).

3.) Stroke valve(s) and report.

4.) Position valve(s) and report.

Function 1 will move the valve to its zero position, read the counts for that position, move the valve to its full-scale position, read the counts for that position, and put

the number of counts for both positions in the system common, superseding the previous values. Function 2 will stroke a valve some predefined (by a variable in system common) number of times and put the average stroke time in system common, again superseding the previous stroke limit time. Function 3 will stroke a valve one time, put the stroke time in system common, compare that stroke time to the valve's stroke limit time, and assign the pass/fail status of the stroke time. Function 4 will move a valve to a requested position by intervals determined by a ramp time and the distance between the current position and the requested position.

Any of the functions may be performed while the system is in the offline mode, but only the valve positioning function (function 4) may be performed when the system is in the online mode.

The program that requests a valve function will pass a function code along with a list of valve numbers to this program through system common. If function 4 is requested, a list of valve positions and a ramp time will also be passed. This program will pass a list of statuses for the results of the requested function for each valve back to the requesting program through the system common.

There are various status codes to signify what happened to each valve when the program tried to execute the requested function, such as:

* Valve did not close,
* Valve did not open,
* Valve did not move at all,
* Valve movement exceeded max time limit,
* Analogic subsystem error detected,
* Valve failed stroke time limit,
* Valve's endpoints not 40% apart,
* Requested valve position is less open than controller's valve position, and
* Differential pressure of compressor indicates compressor is active.

After performing the requested function, the program clears and then waits for a global event flag to be set before executing again. Therefore, another program that wants to request a valve movement function should set up the appropriate arrays and function request in system common and then set the above mentioned global event flag. The requesting program will know the valve movement program has completed processing the request when the same global event flag is cleared.

4.3.7 Function Control

The control program was added to the system during the detailed design phase, more specifically during the detailed design of the user interface program, as the need

for a system "traffic controller" became apparent. A single dispatcher of system function requests passed from one program to another was necessary to deal with the possibility of requests for the same function coming from multiple sources. Multiple sources are possible because up to three users are allowed to login to the offline surge control system and make interactive requests, in addition to requests that will be made by the future configuration system's interface program. By requiring all function requests to be passed to a control program that possesses the logic to validate, control, and queue those requests, the integration of the future configuration system program and the future archival computer program should be significantly simplified. Another benefit of creating a system control program was that of providing an appropriate location to place administrative functions of the system, such as managing the entry and exit of interactive users and managing a change in the system's online/offline mode. Some of the diagnostic functions that the preliminary design had assigned to the user interface program were transferred to this program, as it seemed a more expedient placement after this program was defined. A new diagnostic function, performing pre-operation status checks, was also added. When this function is requested, the control program will determine the state of the computer override and controller override switches; determine the state of output

data for bypass and discharge valves' servo amplifier power; determine the auto/manual mode of each CCC controller; make a health check of ambient readings of the controllers' differential pressure and flow inputs and of ambient readings for each compressor's inlet temperature and internal/discharge temperature; determine the validity of all bypass and discharge valves; determine if bypass valves are closed and if discharge valves are open; determine the status of all controllers (excluding alarm conditions); and determine the status of the Analogic I/O subsystem. The program compares the current conditions of all such status information to what is considered to be the normal conditions. The results of those comparisons are stored in status words in system common defined to hold those results. The data display program will use those results to generate the pre-operation report screens.

Since the configuration computer program was not implemented, the only program passing requests to the control program is the user interface program (one to three versions). User commands involving the controller communications program, the valve movement program, the recording program, the formatting program, resetting alarms, and diagnostic functions are passed to this program through system common in a function request code word. Any supplementary data, such as a list of valves, is passed through arrays in system common.

The control program has a high priority and its execution is tied to two global event flags. At the end of the program's execution cycle it waits for either of the event flags to be set before executing again. One of the event flags is set by a five second timer within the program. If a user interface program requires a response from the control program sooner than the five second time interval, it can set the second event flag, causing the control program to wake up immediately.

4.3.8 User Interface

The user interface program processes the user's input from the keyboard and provides the user with a menu-driven medium for giving commands to, or for requesting information from the surge control system. The screens presented to the user by this program and the data display program were designed to minimize the number of keystrokes required to obtain the system's response. Appropriate key selections are displayed in the lowermost region of non-menu screens to relieve the user of the need to memorize any keystrokes. Invalid input is discarded by this program and therefore is ignored by the system. Notification of invalid input was considered unnecessary with the presence of the valid input prompts on all screens.

Menu control is table (array) driven with each menu selection invoking subsequent menus or displays based upon

the input of function keys. Figures 42 through 47 illustrate the various user interface menus. Unused function keys are included on some menus in order to retain the relative position of the assigned function keys as they are physically grouped on the keyboard. Depending on the action required by the user's input, the program will either pass a function request to the control program, pass a request to the data display program, or respond to the input itself using one of many subroutines. The main type of requests that the program processes itself is the display and/or editing of the data base file, the system values file, and the controller constants file. Figures 48, 49, and 50 give a sample of each of those types of display/edit screens.

Regardless of whether the function is processed by this program or passed on to another program, this program will always determine if the display screen needs to be changed in response to the user's input. If it does the program clears the terminal screen and outputs a screen background. If further screen output is to be accomplished by the data display program, this background will probably consist of only a screen heading and a list of valid inputs given at the bottom of the screen. If the new screen is a menu, the background will contain the entire screen. These background screens are stored in files on disk and read

into a virtual array of 32,000 bytes during the program's initialization. A virtual array was used in order to avoid using an unusually large portion of task address space to store the background screens. Virtual arrays do not occupy task address space, but they have several restrictions on the manner in which they may be used. The background screen files on disk may be easily edited at any time.

The user interface program passes the number of a screen to be displayed and any additional information to the data display program through a private buffer using system directives. Since there may be one to three versions of the user interface program running at the same time, and each version of this program requires a complementary version of the data display program, this method of passing information guarantees a one to one communication between version pairs of these programs. The two programs also need to synchronize their output to a terminal in order to avoid sending output at the same time. The coordination is accomplished through global event flags.

The user interface program's execution is triggered by user input. When the program completes processing of all characters in its input buffer it will simply wait for more input. Its task priority is in the same relative range as the data conversion program and the valve movement program, which is lower than the hardware interface programs and higher than the file processing programs.

4.3.9 Data Display

The data display program produces all user terminal screens that are not menus and are not file display/edit screens. These include eight screens designed to aid the maintenance of the Analogic I/O instrumentation; two screens to display the pre-operation check report; nine screens designed to select valves, valve positions and ramp time, and provide reports for valve movement functions; a screen to display current readings and results of ambient checks for controller inputs and compressor temperatures; a screen to allow selective resetting of surge alarms; twenty-three screens to display the RESCOM system common; three screens to display the system values file; a variable number of screens to display the alarm log or the operator log; and a screen to display status information. Figures 51 through 59 show examples of some of the screens produced by this program.

Generation of the screens is partitioned into numerous subroutines and all the data and status information needed to display the screens is taken from system common. When the program is directed to generate the pre-operation check report, it also writes the report to a file on disk so that it may be displayed or printed at some later time.

This program is controlled by the user interface program as discussed in the previous section. In the

absence of a new screen number from the user interface program, this program will update its current screen twice a second. Its priority is very close to, but slightly lower than the user interface program's priority.

4.3.10 Data Base Transfers

The original data base file for the surge control system was created on an IBM compatible PC using the commercial software package called dBASE. The format of the dBASE file is foreign to both of the RSX operating system's file management services (FCS and RMS) and could only be read as an unformatted file. The location of file header information in the dBASE file was obtained by examining a dump, in both octal and ASCII forms, of several different dBASE files. The format of the data in each record was also obtained from the file dumps.

After the dBASE was transferred from a PC to the PDP-11/73 supervisory computer using the file transfer utility Kermit, it was opened as an FCS, unformatted file and read one block at a time into a 512 byte buffer. The number of records in the file, the length of each record, and the relative location in the file of the first data record are extracted from the file header block. A new RMS, indexed, keyed-access file is then opened to hold the newly formatted version of the file. One or more blocks are read from the dBASE file to get to the block containing the

first data record. After the buffer pointer is positioned at the first record, the bytes in that block and all remaining blocks are then separated into records and written to the RMS file. When the transfer is complete the data base program sets an event flag to notify the control program that it has finished.

Another similar program was developed to translate the RMS file back to the dBASE format prior to being transferred back to a PC. The purpose of being able to transfer the file back to a PC was to provide the capability to engineers to make file modifications in the more familiar dBASE format. By retrieving the file from the surge control system before making modifications, any previous changes made to the data base through this system's user interface will not be lost.

4.3.11 Recording Data, Alarms, and Events

During the detailed design of the program that records files on disk, two changes were made that differed from the program's preliminary design description. First the requirement of recording the system common was eliminated. Another program was written that gave the developers the capability to examine the contents of system common at any time. The second change was the separation of the data log into three files: an alarm message log, an operator input log, and the two hour data log.

The recording program will execute at a relatively low priority within the system, but higher than the data base transfer programs and the file formatting program. The program will control its own execution cycle with a one second timer.

Each time the program executes it will check the state of all controller status words (eight of them), the temperature alarm status word, the Analogic subsystem status word, the controller override digital inputs, and the computer override digital inputs. If any of these status words have changed from the previous second's check, they are examined on a bit by bit basis to determine exactly which parameters have changed state. Appropriate messages, including the time and date, are then recorded in the alarm log for those parameters.

Certain user requests and operation of the system will be recorded at the time they are performed in a file called an operator log. The types of system action recorded in this log include going from offline to online operation (or vice versa), user logins to the system, performing any of the valve movement functions, performing a pre-operation check, exercising the alarm lights, storing a two hour data log, remote programming of a controller, modifying a file, and resetting alarms.

A function code word in system common is checked to see if such an action has taken place. A zero function code

indicates no action is to be recorded. A non-zero code causes the program to record the action indicated by the code, along with the time and date, and then zero the function code word. The action function codes are placed in this word by either the control program or the user interface program.

This program also records the circular two hour data buffer. Once each second, all input data from both the CCC controllers and the Analogic I/O subsystem is saved in a buffer, along with the current time and date. When this buffer in the program is full (thirty seconds of data), it is written to the disk file. The buffers are written to the file in a circular manner so that it will contain the most recent two hours of data. This program will monitor a flag in system common and when the flag is set it will begin tracking elapsed time. After thirty minutes has elapsed since the flag was set, the two hour circular file will be closed and renamed, and the flag in system common will be cleared. A new two hour data file will be opened immediately and recording resumed in the new file. This flag that causes the data log to be closed and a new one opened is controlled by operator input through the control program. It is assumed that the operator will select this function whenever any event of consequence involving the compressors occurs. Saving the data in a separate file is

delayed thirty minutes after the flag is set in order to provide a window of data on each side of the time that the function was selected.

4.3.12 Formatting Files

This program's requirements were reduced during detailed design in three ways. First the requirement to format the system common file was removed due to the recording program no longer carrying the requirement to record such a file. Secondly the requirement to format the two hour data log was removed because that requirement was moved to the archival computer system. Thirdly the requirement to format files for display on the user's terminal was removed. The reason was twofold. Some of the files, the alarm log, the operator log, and the pre-operation report, are recorded in a format suitable for screen display. The remaining files, the data base file, the system values file, and the controller constants file, must be accessed and formatted by the user interface program in order to handle modification of those files, so it was logical to let that program perform the function of merely displaying those files as well.

This program's function, then, is to prepare files for output to a printer. The program does not execute unless requested by another program and it will be a low priority task. When another program wants to tell the formatting

program to format a file, a code representing the desired file is placed in a variable in system common and a global event flag is set. This program will examine the code word and proceed to format the requested file and send it to the printer. It then clears the code word and the event flag and waits for the event flag to be set again. Should there be multiple requests for this program's services, they will be queued by the control program.

4.4 IMPLEMENTATION AND TESTING

Implementation and testing is the final phase of software development prior to the system being delivered for operation in its destined application environment. However, it should be noted that the longer an error is allowed to remain in the software, the more expensive it is going to be to remove it, with the cost of correction escalating rapidly as the development progresses. Testing software after it has been specified, designed, and coded has been proven to be an inadequate means of producing reliable software. For example, a software system that meets every requirement of its specification will fail a user acceptance test if the system was not specified correctly [36]. Reliability, therefore, cannot be injected during the final stage of software development [37]. It is accomplished by the absence of errors in all of the

preceding stages of development. The role of testing should be that of locating the relatively small number of errors remaining in a well designed software system. The historically regular failure to consider software reliability or quality before reaching the testing phase has caused it to be an arduous and time consuming process [38].

Recognition of this major flaw in traditional software development by the software industry has led to a significant increase of emphasis on addressing software quality from its initial phases throughout its lifecycle. But while prodigious amounts of material have been published on this aspect of software development, a clear consensus of standards has not been reached, software quality being a complex and multidimensional attribute that is difficult to define. Other types of products usually have properties that are more easily measured and tested, like the power of a motor, the strength of building materials, or the dimensions of fittings. The intangible and often "one of a kind" characteristics of software have made standards of its quality elusive. Attempts to qualify the properties of software frequently apply adjectives such as reliability, flexibility, efficiency, usability, integrity, testability, functionality, portability, adaptability, and maintainability. Unfortunately, these

terms are almost as subjective as the word quality. Yet regardless of personal or organizational variations in the definitions of these attributes, if they are held forth as objectives throughout the software development process, the final software product will possess a higher degree of quality as a result of their consideration.

4.4.1 Testing Phases and Observations

Having acknowledged that the burden of producing quality software should not rest upon its latter stages, the implementation and testing phase of development remains a necessary and critical effort. Testing is the fundamental technique of exercising the software for the purpose of locating errors and establishing confidence in the software's correctness [37]. The testing phase is itself divided into phases of unit testing, integration testing, system testing, and acceptance testing. Implementation is achieved when all software modules work correctly, both individually and in unison. Figure 60 illustrates how each phase of testing relates to an earlier stage of software development.

Unit testing involves exercising each software module (unit) as a separate entity. This may require the development of driver modules, which pass test data to the module being tested and receive any module output, usually storing it for subsequent examination. Good programming

practice dictates this activity be performed as soon as the module is coded, but it is often postponed and combined with integration testing. During integration testing the modules are combined to execute together. Beginning with a single module, other modules are included in an ordered sequence, one at a time, to test the merger of their interfaces. Following the successful integration of all modules, system testing is performed to ensure that the software meets the objectives of its specifications and to validate the system's architecture. The system's control logic, timing, throughput, and performance are evaluated. After the software has passed a thorough system test, the last phase of testing, the acceptance test, should be a formality. Acceptance testing is based on the user requirements and generally includes the user in the performance of the test. Sometimes acceptance testing involves executing the system under different conditions than those of the system test. If the software's application environment was simulated during system testing, then acceptance testing may not be considered complete until the software has satisfactorily performed in its intended environment.

There is a wide range of opinion on the criteria that should be used to test software programs [37]. One end of the spectrum advocates designing test cases based on the

software specifications, without regard to the internal logic of the program, with the goal of testing every value and combination of input. The opposite approach is to design test cases based on the program's structure, with the goal of exercising every possible path of execution. The number of test cases required to fully carry out the first strategy makes it absurd if not impossible. The number of test cases required for the second method is equally impractical, with the added problem of being insufficient. Executing every path of a program would not identify any paths that were missing, and it would not guarantee that the paths worked correctly for all input values. The conclusion in these observations is that it is infeasible to prove that a piece of software is totally free of errors, and that neither extreme of applying test cases is desirable. Rather, test cases should be designed to demonstrate all the functions of the specifications and the inputs to those test cases should represent a class of inputs, such that if the test executes correctly it may be realistically assumed that other inputs of the same class will execute correctly. The effort applied to verify the software is determined by the criticality of the application [38].

Real-time systems, particularly control systems, are most often regarded as highly critical applications [3]. The cost of a software error in these systems routinely

involves property and frequently entails human safety to the point of costing human lives. Therefore a more intensive degree of testing is normally associated with real-time systems. The testing should verify that the system's ability to continue to operate in the face of bad inputs, to recover from non-fatal errors, and to avoid damage in the case of failure is sufficient for its application [36].

The critical nature of real-time applications also leads to the regular use of process simulation, since finding errors through the actual process can be disastrous. When the target computer of the system is not available during software development, a testbed may be prepared on a host computer to facilitate testing the software [39]. A testbed is an artificial computer environment designed to simulate the hardware on which the software is intended to execute. Likewise, when actual hardware devices to which the system must interface are not available during software development, they too may need to be simulated in order to provide a "real world" environment for testing. Needless to say, any form of process simulation or hardware simulation must truly represent its actual counterpart for the testing to be effective. The primary advantage of testing through the use of simulation is that of being able to test the software without the risk

of physical damage. But other advantages may be gained as well [3]. Simulation allows test inputs to be reproduced and it allows cases to be designed which test conditions that will occur only rarely, both of which could be a problem if the software were tested in its actual environment.

4.4.2 Application Testing

A software test plan was written for the anti-surge control software system during the preliminary design phase. Because the details of the system had not been designed at that point, the test plan did not provide detailed instructions for testing the software. However, it did provide a written plan for the informal and formal testing activities that would follow, giving preliminary consideration to hardware and software integration, formal test levels and classes, and simulation requirements. The test plan also identified functional requirements of the specification that would be tested and the resources needed to support all phases of testing. Near the end of detailed design, a software test description was drafted to describe the formal test cases in substantially more detail than was given in the test plan.

All software modules were unit tested as their code was completed. Test drivers were developed for the Analogic I/O program, the controller communications program, the

data validation and alarm program, and the valve movement program. Unit testing of the controller communications program revealed a timing problem. The operating system's terminal driver was used to communicate with the controllers over standard RS-232 serial connections. But the terminal driver, having been designed for keyboard input, could not process the input and output for all fifteen of the controllers in the specified one second time interval. The program's time interval had to be increased to one and a half seconds to accommodate the terminal driver's limitations. This deviation from the system's specifications was reviewed and considered an acceptable compromise.

Integration testing required minor changes in the relatively large amount of task synchronization between the control program, the user interface program, and the data display program. Only one significant integration problem was encountered, once again involving the controller communications program. The program intermittently failed to complete its processing within the allotted time interval. Rather than increase the controller program's time interval again, the Analogic I/O program's execution rate was slowed down from twenty times a second to fifteen times a second, which eliminated the controller program's intermittent failures to complete on time. The new

execution rate of the Analogic I/O program was appraised as remaining fast enough to continue to provide adequate surge protection for the plant compressors.

Following integration of all programs, the software system was exercised thoroughly in an attempt to verify all functional and performance requirements of the specification. New functional aspects that were not required by the specification, but were a result of the effort to make the programs suitable for use in future systems, were also verified. At this point in the testing process, the system was operated for extended periods of time in order to determine if any undetected errors would surface as a result of accumulation.

When the software had been system tested to the satisfaction of the author, a software test procedure was prepared to specify the final acceptance test. The test procedure was based on the previous test description and defined all test cases to be performed during the final test. Each test case was described with step-by-step instructions. A cross reference table was prepared as part of the test procedure to relate test cases to the requirements of the software specification. The future, intended users of the system performed the final acceptance test by following the steps in the test procedure.

CHAPTER V

CONCLUSIONS

5.1 SYSTEM EVALUATION

The anti-surge control system exhibits a significant improvement in design over other real-time systems that the author has developed and/or maintained. The improvement is primarily attributed to better, more thorough definition of the system at the system analysis and software specification phases. The improvement in these areas was carried through the remaining phases of software development, resulting in modules with a high degree of functionality. Much less time was spent in the software integration testing and system testing phases of implementation than had previously been expended on other less functionally defined systems.

Many improvements, too numerous to detail, were made within individual programs. However, three important characteristics of the preliminary, high level design of the system may deserve individual mention.

While most of the other systems use one or more files to serve the same purpose as the surge control system's data base, they lack the scope, flexibility, and ease of modification possessed by this system's data base. Parameter additions, deletions, and description changes are

menu driven in this system and none of these alterations require any code modification, unless an instrumentation type is introduced that has not been used with any previous system. Both of these attributes are a valuable improvement over the other systems.

The surge control system's treatment of data conversion also implemented more than one design improvement. The major reform was that of leaving data in its unconverted form until conversion is required for human viewing. This change in design is closely related to another design improvement, which was to perform all data comparisons such as validity and alarm level checks using raw data rather than converted data. The advantage comes from the considerable decrease in the amount of time required to perform arithmetic processes on arrays of integers compared to the same processing of arrays of floating-point numbers. The other systems converted all analog data immediately after it was acquired and the raw form of the data was not used in the systems thereafter. Leaving the data unconverted until there is a reason to convert it saves a great deal of conversion processing each acquisition cycle that is otherwise wasted. It is wasted because terminal screens displayed to the user only use a subset of the analog data and virtually all printing of converted data is done before or after online operation.

The data conversion program implemented by the surge control system allows any other program in the system that requires converted data to tailor its own list of parameters to be converted or to ask for the entire set of analog parameters to be converted.

Another noteworthy aspect of the surge control system's design is that of allowing more than one user to operate identical terminal interfaces. While other systems may have permitted more than one user to be logged on to the system, the software that handled different terminals was also completely different, such that certain functions could only be invoked from specific terminal ports. The surge control system allows up to three users (the number could conceivably be raised) access to the same type of interface with the only restriction being that only the first user can take the system into online mode. This feature provides minor enhancement of the system's usefulness to plant operators but it is a significant advantage to analysis and maintenance personnel by allowing them to monitor the system's operation over the plant network.

The effort to design the programs of the surge control system such that they would meet the functional requirements of other real-time systems developed and implemented at the same site has paid off handsomely. Three out of four software systems developed after the surge

control system have used between forty and ninety percent of the surge control system's programs. Using these programs in the new systems required minor modification in some cases and no modification in others. The time spent in development of the software systems was substantially reduced.

The recent integration of the configuration system interface program provided some practical evidence of the surge control system's clean, low degree of program coupling. The configuration system interface program was developed and integrated into the system as part of the configuration system project. It was accomplished by software personnel that did not have any previous experience with the surge control system, yet by acquainting themselves with the structure of how function requests were made to the control program and to the controller communications program they were able to implement the new interface program with relative ease.

5.2 RECOMMENDATIONS FOR FUTURE IMPROVEMENTS

When a software system is completed, one can always recognize aspects of the design or the method of development that could have been improved. In reflecting on the development of this system, three recommendations for improvement come to mind. One deals with the development

process, one with high level design, and the other with an additional feature for the user.

The user interface program, data display program, and control program were some of the last programs to be developed. It is clear that these programs should have been developed shortly, if not immediately, after the other interface programs. The status information, data displays, and diagnostic functions implemented by these programs could have considerably reduced the time required to unit test the remaining programs. While this recommendation comes too late to benefit the development of the surge control system, future development of software systems may benefit from the hindsight.

The second observation is that the non-real-time processing of the standard terminal driver used to communicate with the fifteen controllers essentially overloaded the system to the point that concessions in response time had to be made for controller communications, Analogic input and output, and alarm processing. Development of a driver to handle the controllers' interface would most likely have spared those concessions. However, the terminal driver's restrictions were not evident at the time of preliminary design and the additional time that development of a controller driver would have required were not within the software's schedule

or budget. Developers of future software systems should take note of the limitations imposed by using the terminal driver as an interface to relatively high speed hardware devices.

The final recommendation is to provide the capability of letting the user define a group of parameters to be displayed on a single screen, thus allowing him to select the type of information he would like to view at the same time. This would allow him to make combinations of data and status information that are the most meaningful and useful to his needs. This feature could be added to the surge control system at any point in the future that the necessary software resources are available.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Mitchell, Bruce R. and Walthers, Denny J., Application Development Under RSX, Unpublished draft, 1986.

[2] Lapedes, Daniel N. and others (eds.) McGraw-Hill Dictionary of Scientific and Technical Terms, 2nd ed., McGraw-Hill, New York, 1978.

[3] McGlinchey, James A. and Frisbee, Alan E., RSX-11M/M+ Real-Time System Design and Performance Management, Digital Equipment Corporation, 1985.

[4] Olsen, P.F. and Orrange, R.J., "Real-Time Systems for Federal Applications: A Review of Significant Technological Developments", IBM Journal of Research and Development, Vol. 25, No. 5, 405-416, September 1981.

[5] "Arnold Engineering Development Center", Brochure distributed by Air Force Systems Command, United States Air Force, Arnold Air Force Station, TN, Undated.

[6] "Arnold Engineering Development Center", Brochure distributed by Air Force Systems Command Test Center, Arnold Air Force Base, TN, Undated.

[7]* Belew, P. W., Report No. FSD-ER-B-84-13, Calspan Field Services, Inc., Arnold Air Force Station, TN, September 1984.

[8] McMillan, Gregory K., Centrifugal and Axial Compressor Control, Instrument Society of America, 1983.

[9]* Belew, P. W. and Bisby, T. L., Report No. FSD-ER-B-85-6, Calspan Field Services, Inc., Arnold Air Force Station, TN, March 1985.

---

* These documents are internal working documents that are not available outside Arnold AFB. They are referenced here to credit the authors, whose material is discussed in this thesis.

[10]* Bailey, R. L. and others, Report No. FSD-ER-B-85-3, Calspan Field Services, Inc., Arnold Air Force Station, TN, January 1985.

[11]* Belew, P. W., Report No. EMF-ER-B-88-21, Calspan Corporation/AEDC Operations, Arnold Air Force Base, TN, August 1988.

[12] Series II Antisurge Controller Instruction Manual, 3rd ed., Compressor Controls Corporation, September 1985.

[13] Series II Controller Computer Communications Specification, Revised ed., Compressor Controls Corporation, February 1986.

[14]* Belew, P. W., Report No. FSD-ER-B-86-15, Calspan Corporation/AEDC Division, Arnold Air Force Station, September 1986.

[15] KDJ11-A CPU Module User's Guide, Digital Equipment Corporation, Maynard, MA, May 1984.

[16] PDP-11 Architecture Handbook, Digital Equipment Corporation, Maynard, MA, 1983.

[17] Davis, Brian J., "DEC Bus Architecture", Proceedings of the Digital Equipment Computer Users Society, Anaheim, CA, 243-253, December 1984.

[18] RSX-11 Handbook, Digital Equipment Corporation, Maynard, MA, 1984.

[19] VT240 Series Programmer Reference Manual, 2nd ed., Digital Equipment Corporation, Maynard, MA, October 1984.

[20] Instruction Manual for ANDS5400, Book One (of two), Revised ed., Analogic Corporation, 1982.

[21] RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual, Revised ed., Digital Equipment Corporation, Maynard, MA, July 1985.

---

* These documents are internal working documents that are not available outside Arnold AFB. They are referenced here to credit the authors, whose material is discussed in this thesis.

[22] RSX-11M Executive Algorithms and Data Structures, Digital Equipment Corporation, Maynard, MA, November 1982.

[23] PDP-11 FORTRAN-77 Language Reference Manual, Digital Equipment Corporation, Maynard, MA, July 1983.

[24] Pieper, John F., RSX A Guide for Users, Digital Equipment Corporation, Maynard, MA, 1987.

[25] Johnson, Sharon L., "Programming with Indirect Command Files", Proceedings of the Digital Equipment Computer Users Society, New Orleans, LA, 427-430, 1985.

[26] RSX-11M/M-PLUS System Management Guide, Revised ed., Digital Equipment Corporation, Maynard, MA, July 1985.

[27] DeMarco, T., "Structured Analysis and System Specification", Classics in Software Engineering, Ed. Edward Yourdon, Yourdon Press, New York, 411-424, 1979.

[28] McClure, Carma L., Managing Software Development and Maintenance, Van Nostrand Reinhold, New York, 1981.

[29] Dickinson, Brian, Developing Structured Systems, Yourdon Press, New York, 1981.

[30] Mink, Catherine, "Speaking Without Words", DEC Professional, Vol. 6, No. 7, 32-42, July 1987.

[31] Page-Jones, Meilir, The Practical Guide to Structured Systems Design, Yourdon Press, New York, 1980.

[32] Yourdon, Edward N., Classics in Software Engineering, Yourdon Press, New York, 1979.

[33] Rogers, G. R., "A Simple Architecture for Consistent Program Design", IBM Systems Journal, Vol. 22, No. 3, 199-213, 1983.

[34] Radice, R. A. and others, "A Programming Process Architecture", IBM Systems Journal, Vol. 24, No. 2, 79-90, 1985.

[35] Madnick, Stuart E. and Donovan, John J., Operating Systems, McGraw-Hill, New York, 1974.

[36] William Hetzel and L. Kreuter Assoc., Structured Testing, Unpublished seminar literature, 1981.

[37] Myers, Glenford J., Software Reliability: Principles and Practices, Wiley, New York, 1976.

[38] Adrion, W. Richards and others, Validation, Verification, and Testing of Computer Software, National Bureau of Standards Special Publication 500-75, U. S. Government Printing Office, Washington, 1980.

[39] Presson, Edward, SOFTWARE TEST HANDBOOK Software Test Guidebook, RADC-tr-84-53, Vol II (of two), Boeing Aerospace Company, Griffiss Air Force Base, NY, March 1984.

APPENDICES

APPENDIX A

FIGURES

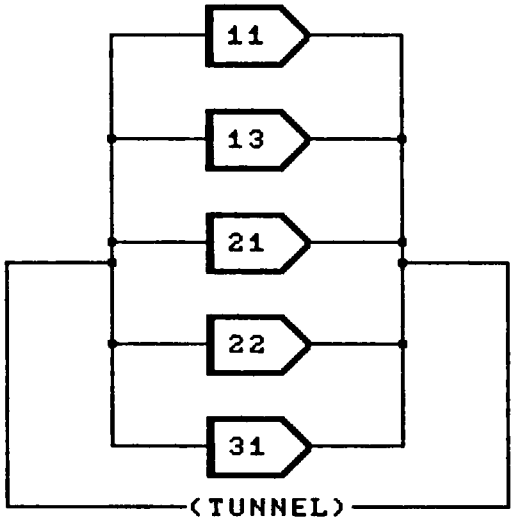


Figure 1. 1-Stage Special Type B Configuration

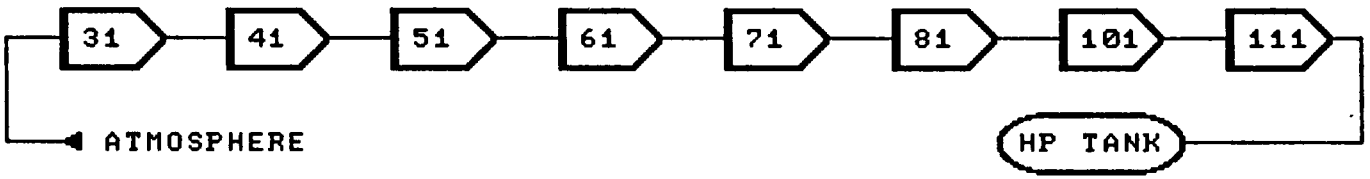


Figure 2. Tank Charge Configuration

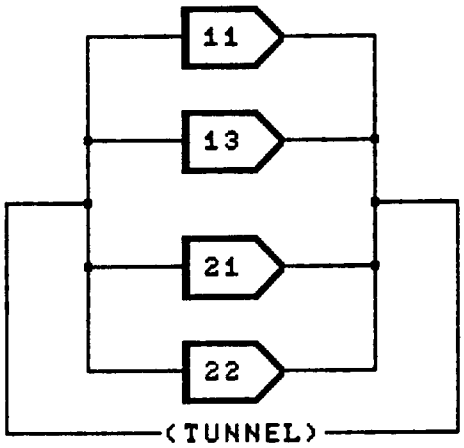


Figure 3. 1-Stage Type B Configuration

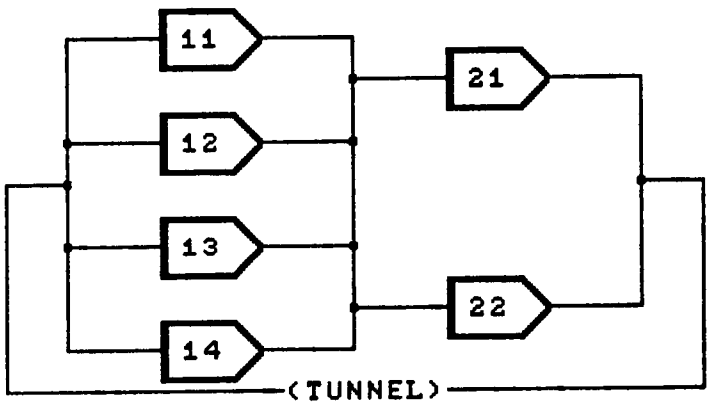


Figure 4. 2-Stage Configuration

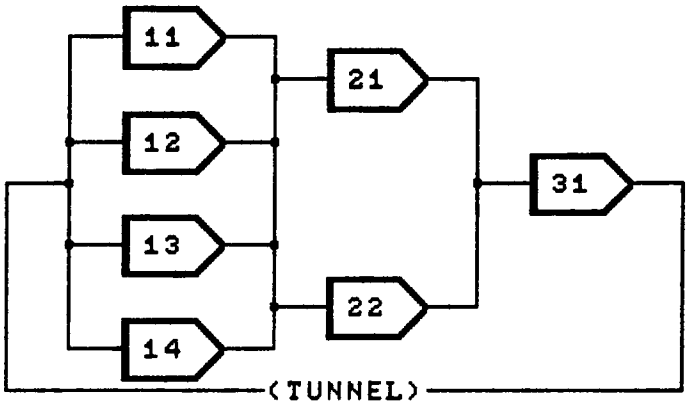


Figure 5. 3-Stage Configuration

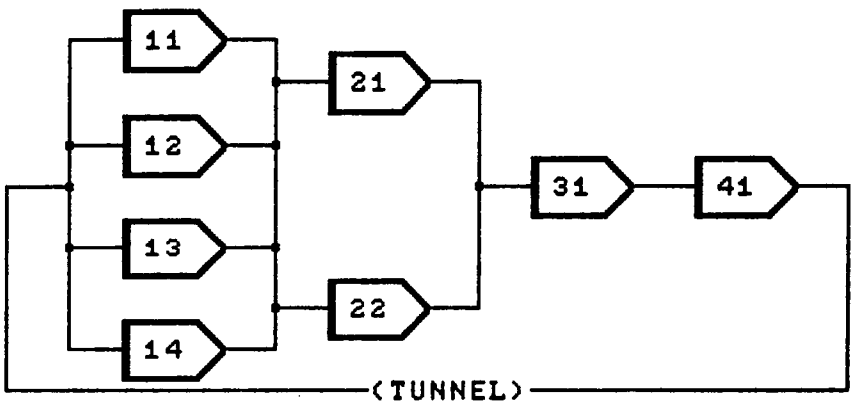


Figure 6. 4-Stage Configuration

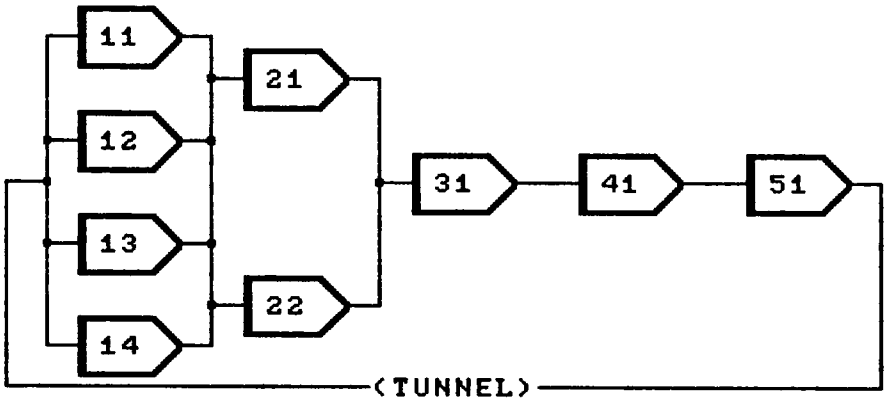


Figure 7. 5-Stage Configuration

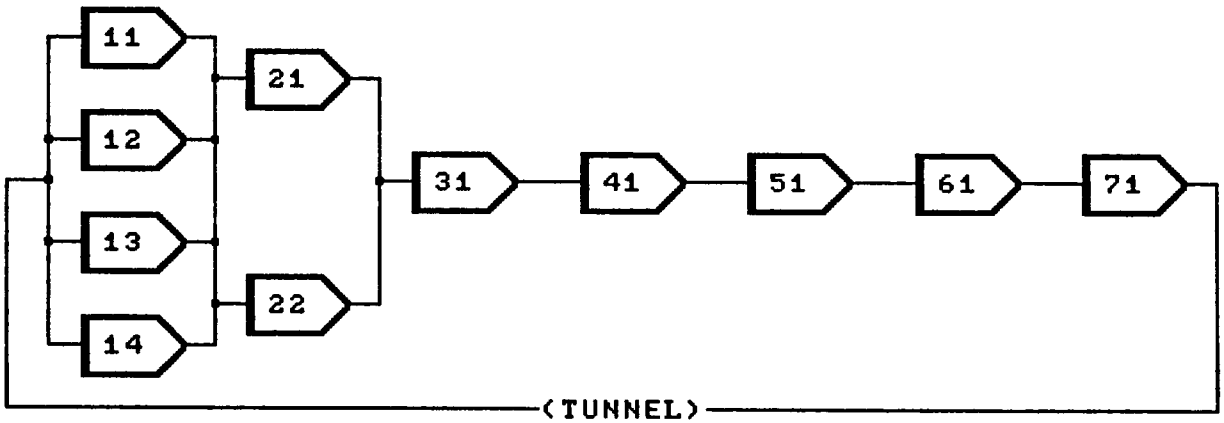


Figure 8. 7-Stage Configuration

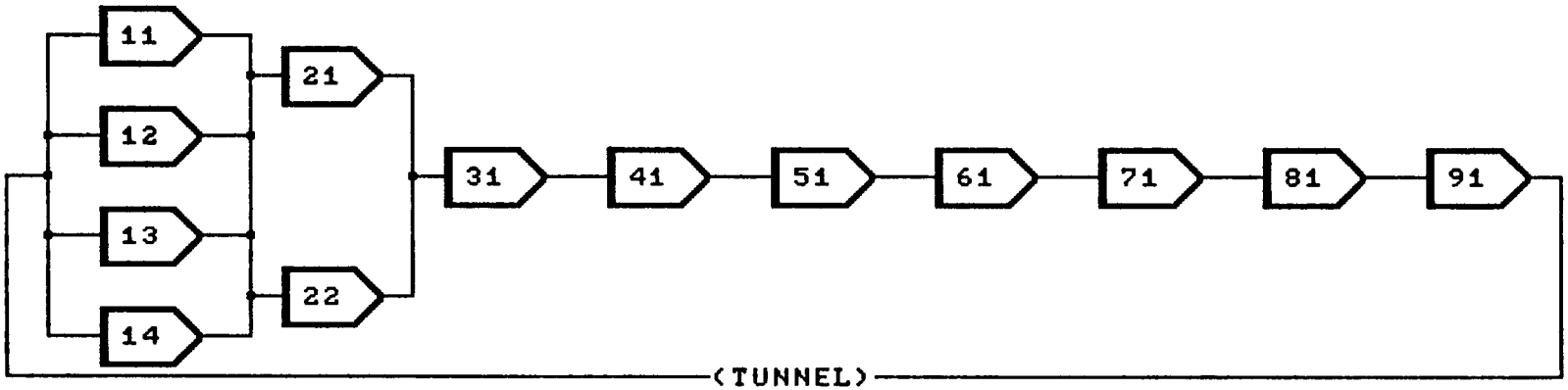


Figure 9. 9-Stage Configuration

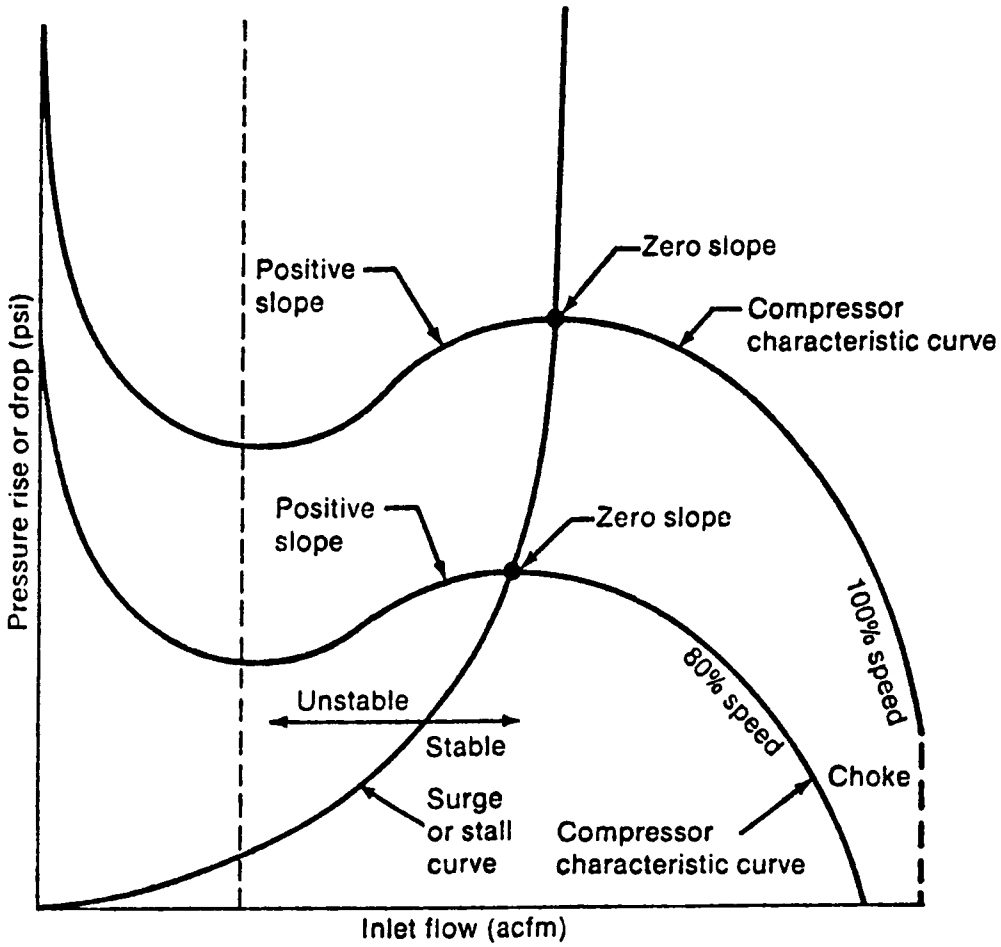


Figure 10. Typical Compressor Map

Source: McMillan, Gregory K., Centrifugal and Axial Compressor Control, Instrument Society of America, 1983.

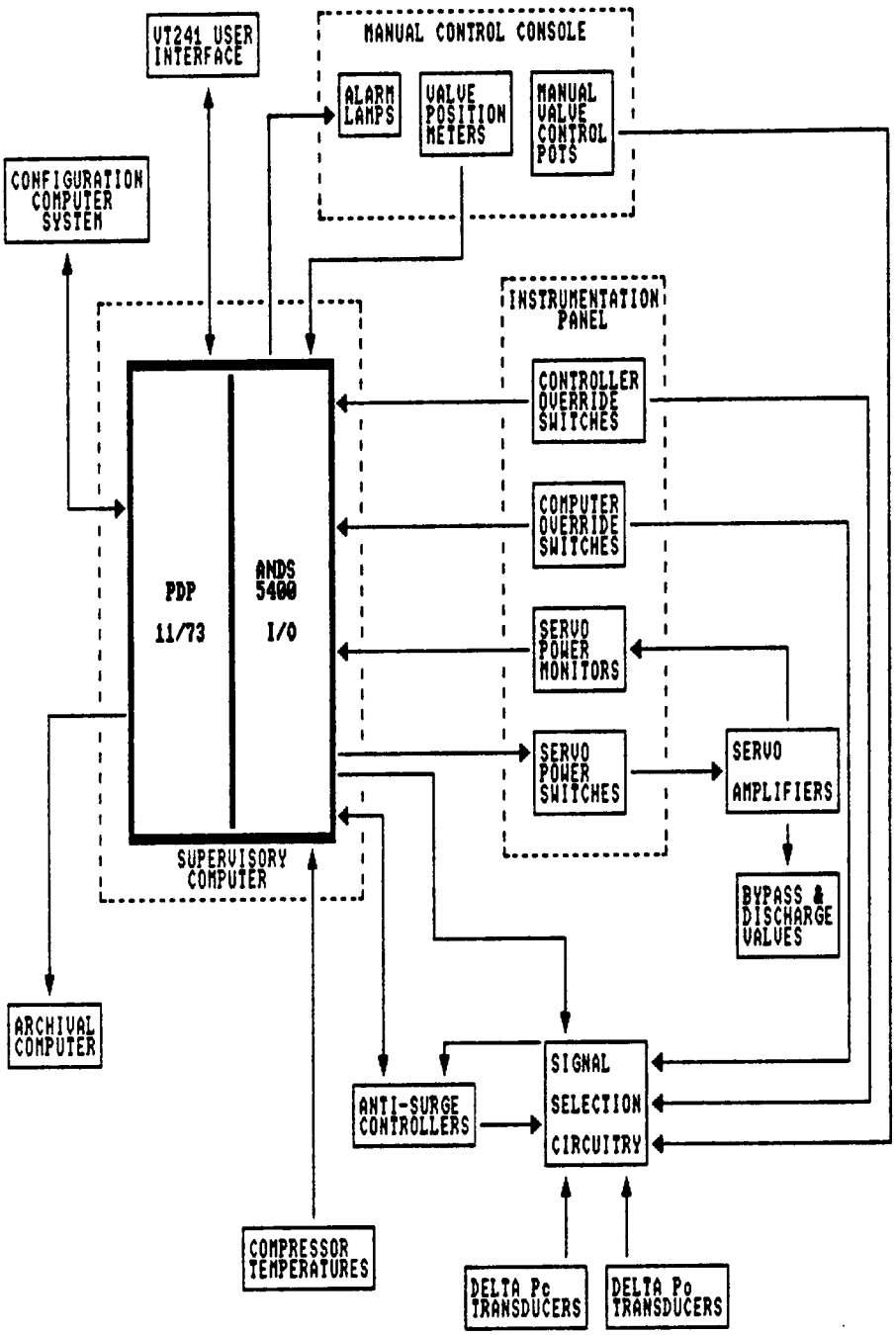


Figure 11. Surge Control System

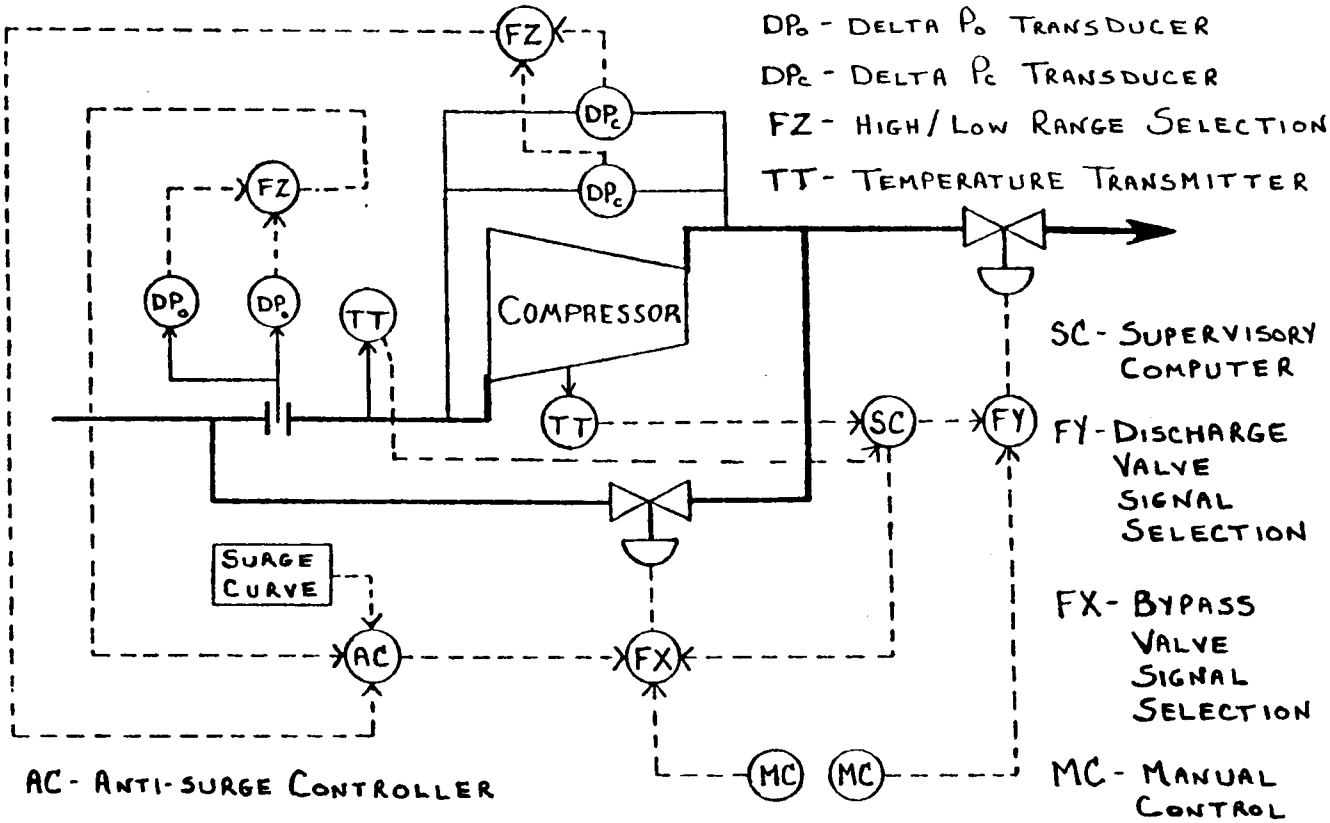


Figure 12. Typical Axial Compressor Control Instrumentation

Source: Belew, P. W., Report No. FSD-ER-B-86-15, Calspan Corporation/AEDC Division, Arnold Air Force Station, September, 1986. (Not available outside Arnold AFB.)

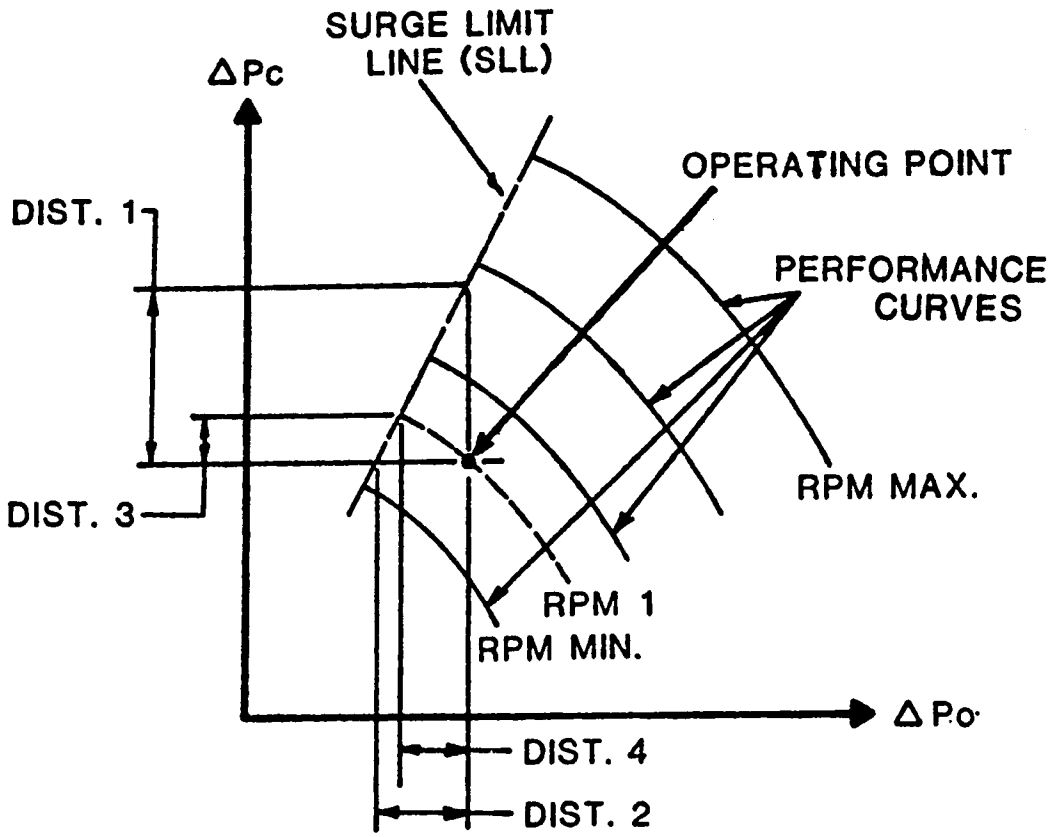


Figure 13. Compressor Map with Antisurge Controller Limits

Source: Series II Antisurge Controller Instruction Manual, 3rd ed., Compressor Controls Corporation, September, 1985.

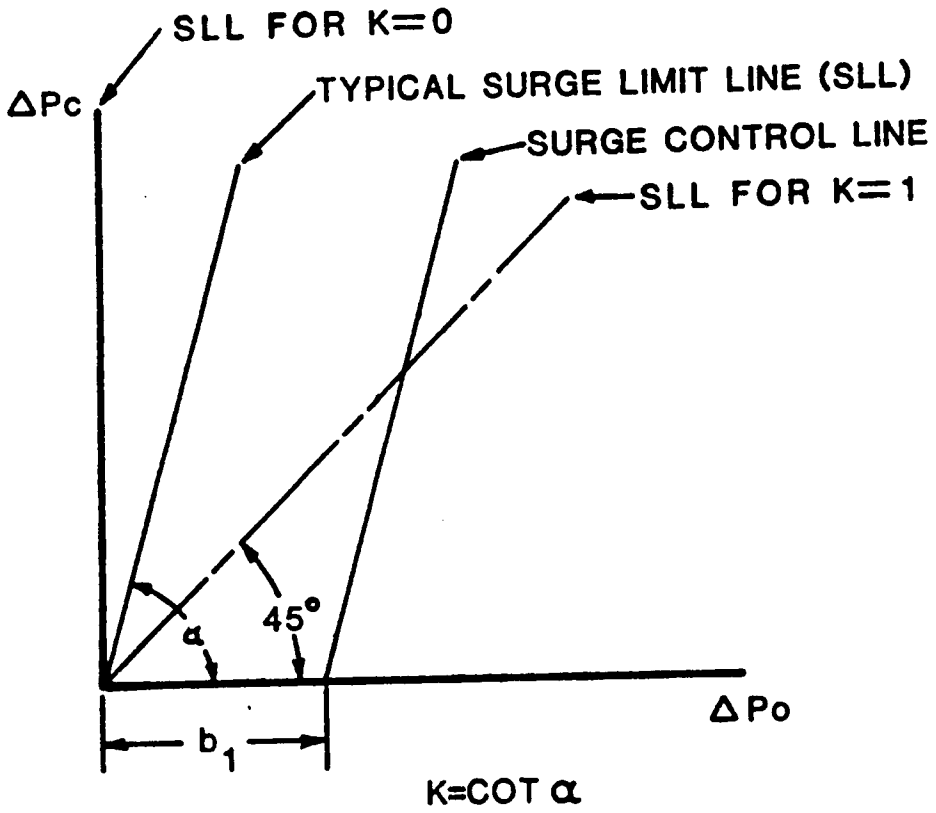


Figure 14. Antisurge Controller Limit Lines for PI Circuit

Source: Series II Antisurge Controller Instruction Manual, 3rd ed., Compressor Controls Corporation, September, 1985.

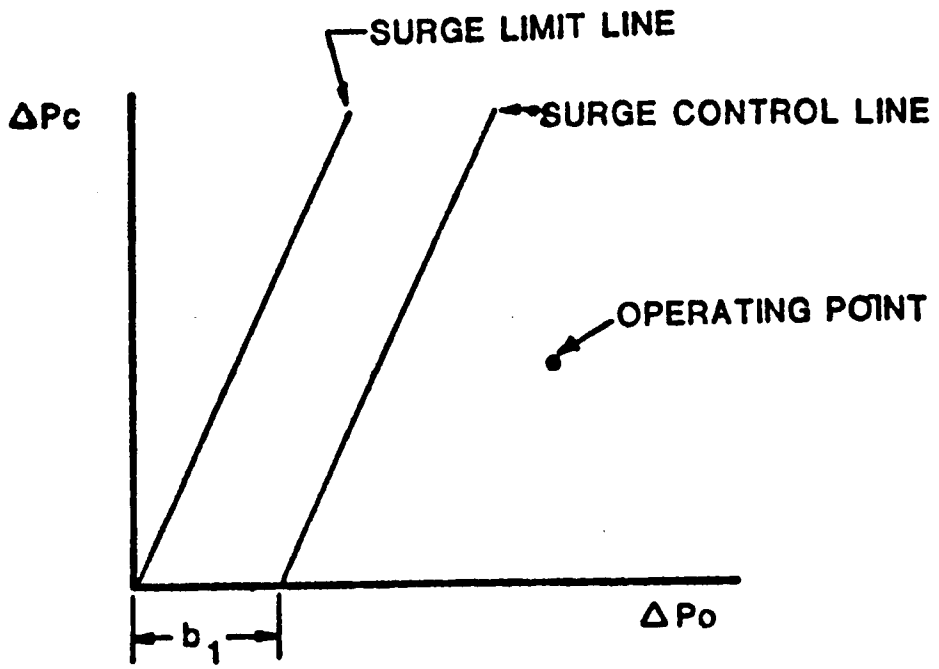


Figure 15. Compressor Operating Point to the Right of Surge Control Line

Source: Series II Antisurge Controller Instruction Manual, 3rd ed., Compressor Controls Corporation, September, 1985.

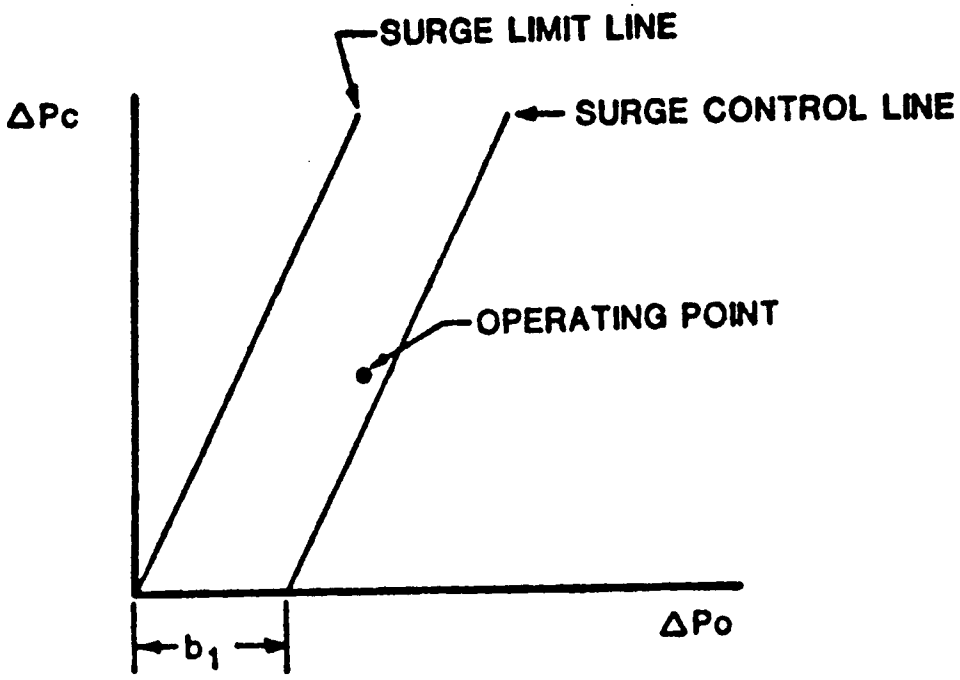


Figure 16. Compressor Operating Point to the Left of Surge Control Line

Source: Series II Antisurge Controller Instruction Manual, 3rd ed., Compressor Controls Corporation, September, 1985.

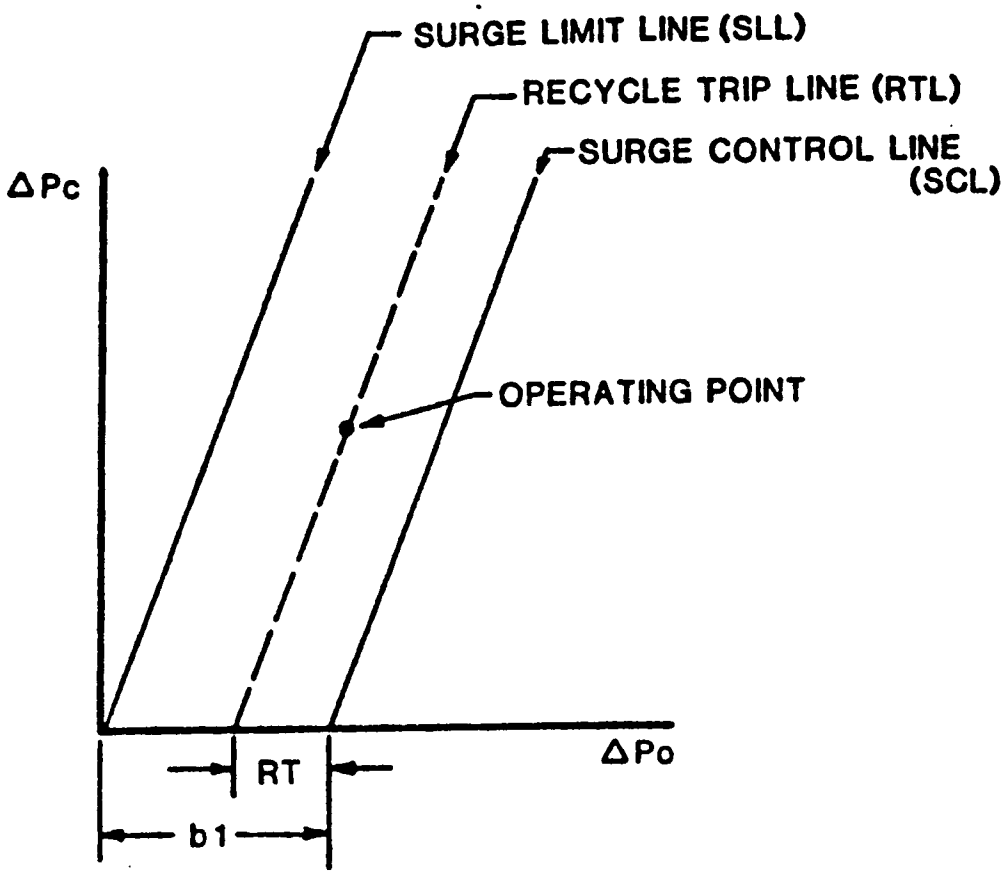


Figure 17. Compressor Operating Point on Recycle Trip Line

Source: Series II Antisurge Controller Instruction Manual, 3rd ed., Compressor Controls Corporation, September, 1985.

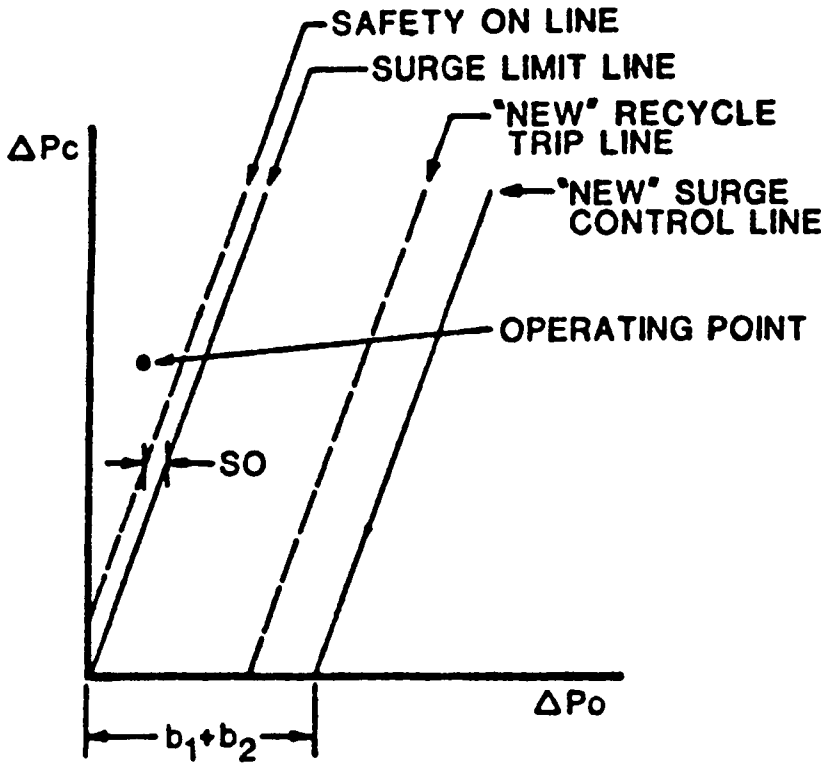


Figure 18. Compressor Operating Point to the Left of Safety On Line

Source: Series II Antisurge Controller Instruction Manual, 3rd ed., Compressor Controls Corporation, September, 1985.

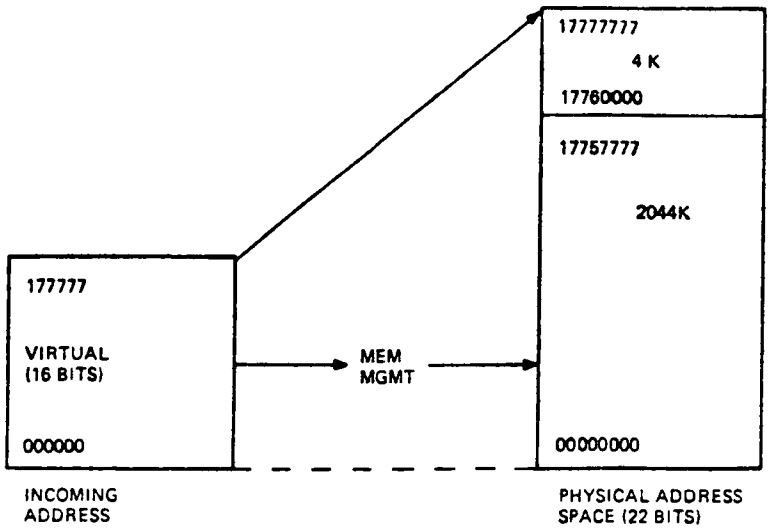


Figure 19. 22-Bit Memory Mapping

Source: PDP-11 Architecture Handbook, Digital Equipment Corporation, Maynard, MA, 1983.

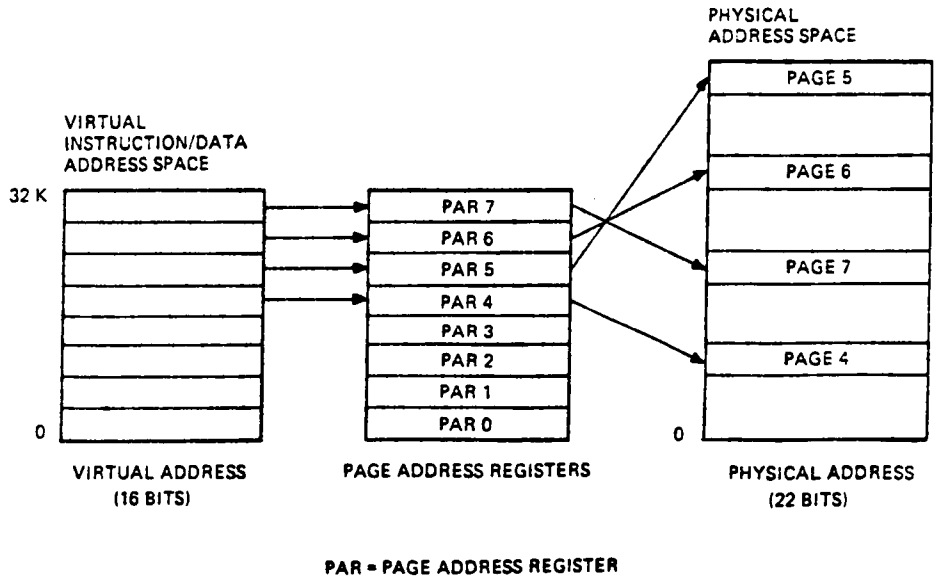


Figure 20. Virtual to Physical Address Mapping

Source: PDP-11 Architecture Handbook, Digital Equipment Corporation, Maynard, MA, 1983.

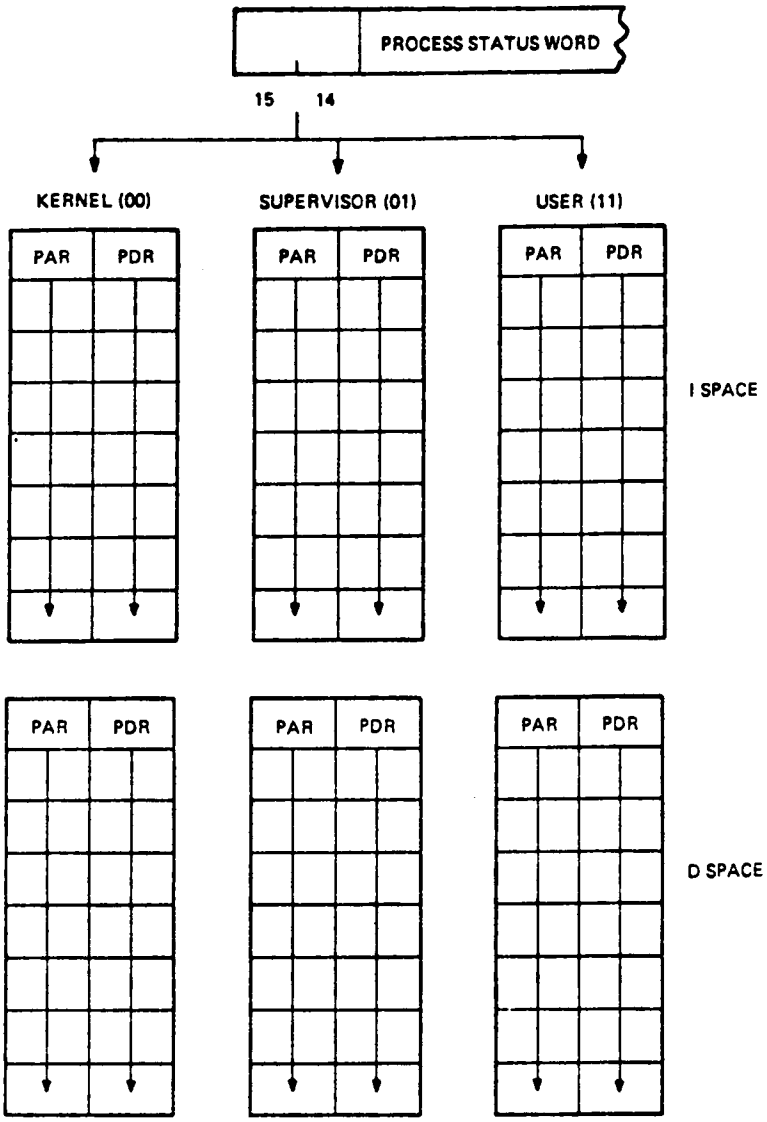


Figure 21. Memory Management Registers

Source: KDJ11-A CPU Module User's Guide, Digital Equipment Corporation, Maynard, MA, May 1984.

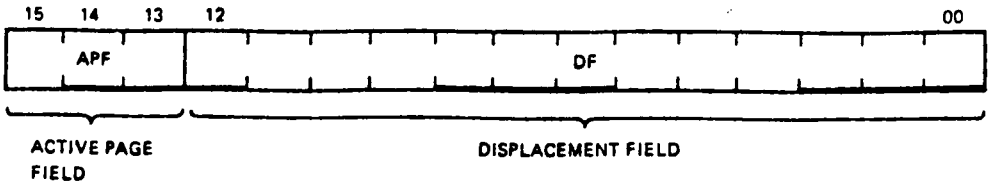


Figure 22. Memory Management Fields of a Virtual Address

Source: KDJ11-A CPU Module User's Guide, Digital Equipment Corporation, Maynard, MA, May 1984.

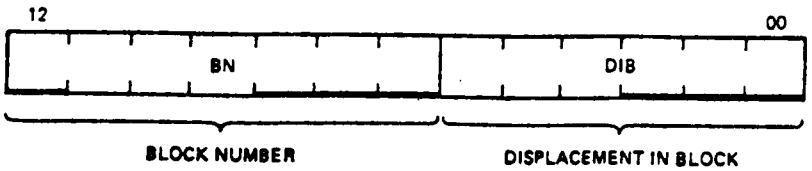


Figure 23. Displacement Fields of a Virtual Address

Source: KDJ11-A CPU Module User's Guide, Digital Equipment Corporation, Maynard, MA, May 1984.

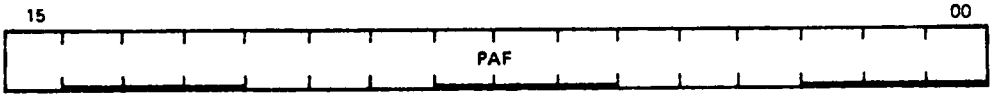


Figure 24. Page Address Register

Source: KDJ11-A CPU Module User's Guide, Digital Equipment Corporation, Maynard, MA, May 1984.

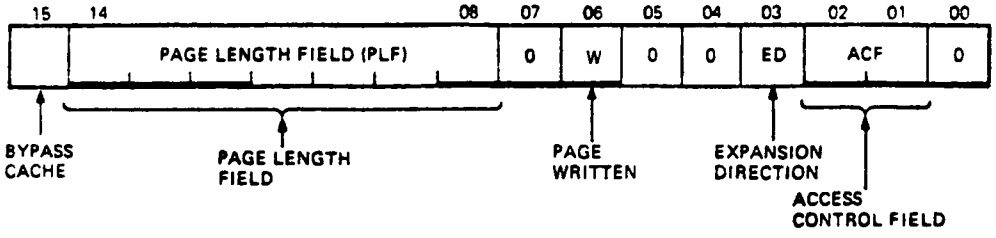


Figure 25. Page Description Register

Source: KDJ11-A CPU Module User's Guide, Digital Equipment Corporation, Maynard, MA, May 1984.

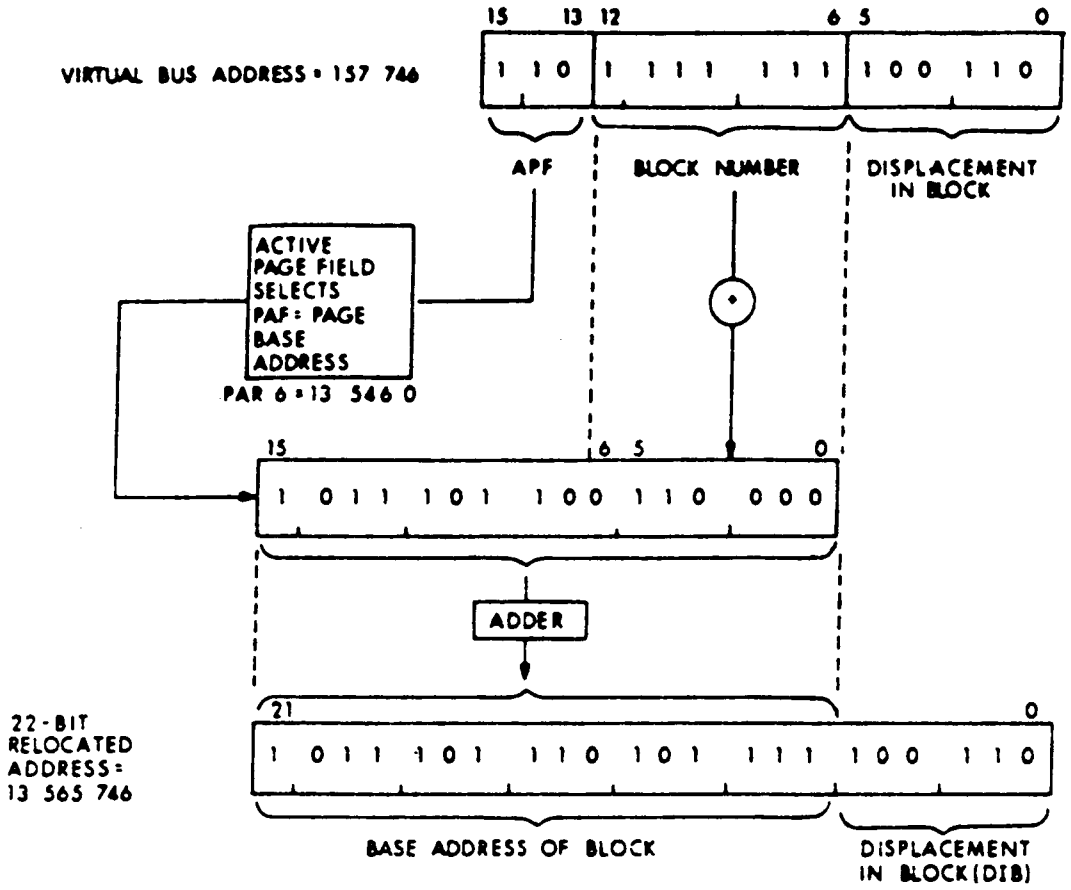


Figure 26. Virtual to Physical Address Translation

Source: PDP-11 Architecture Handbook, Digital Equipment Corporation, Maynard, MA, 1983.

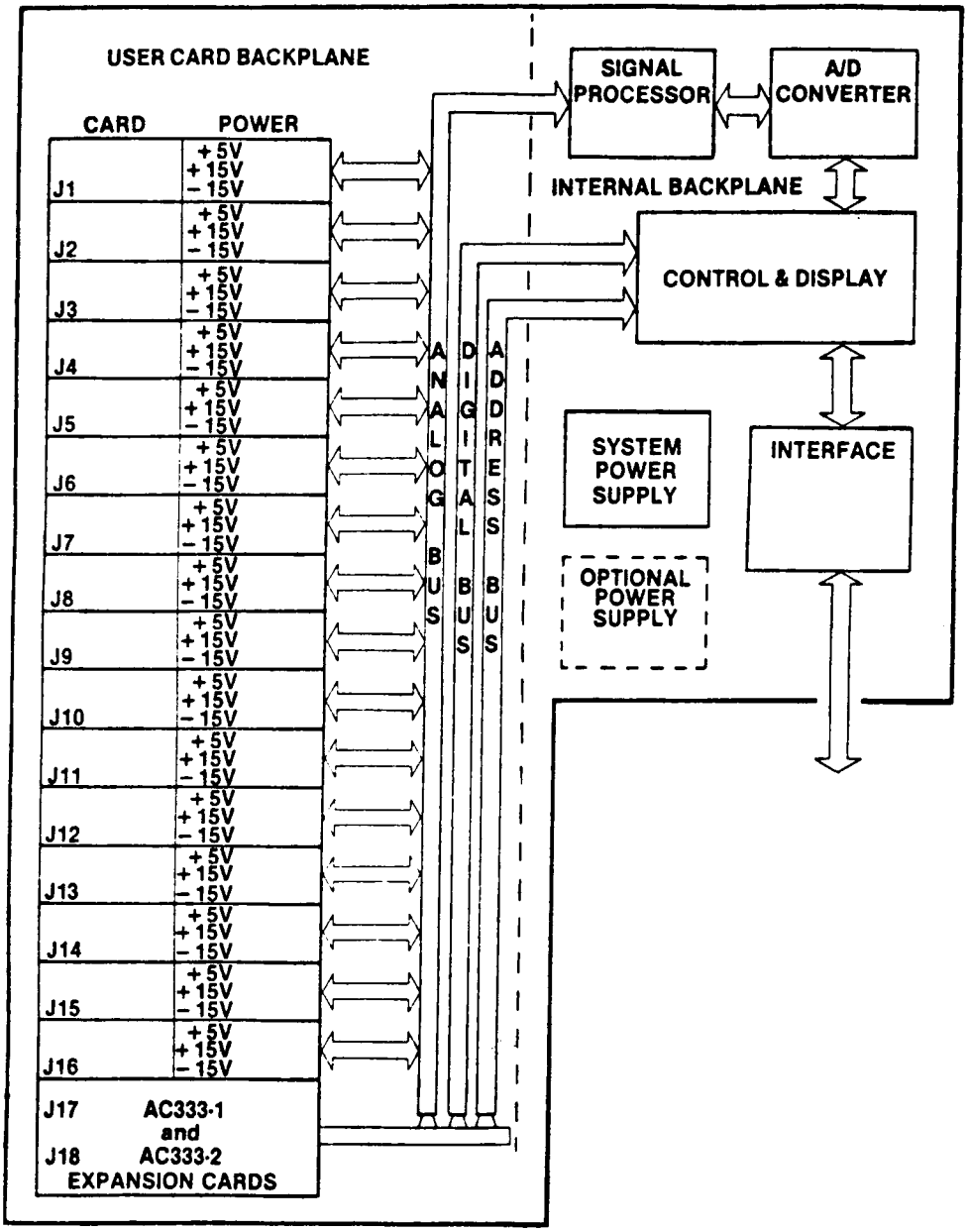


Figure 27. ANDS5400 I/O Subsystem

Source: Instruction Manual for ANDS5400, Book One (of two), Revised ed., Analogic Corporation, 1982.

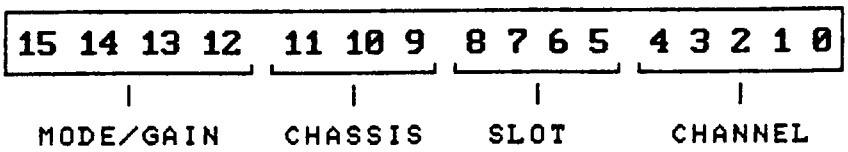


Figure 28. ANDS5400 16-Bit Analog Address

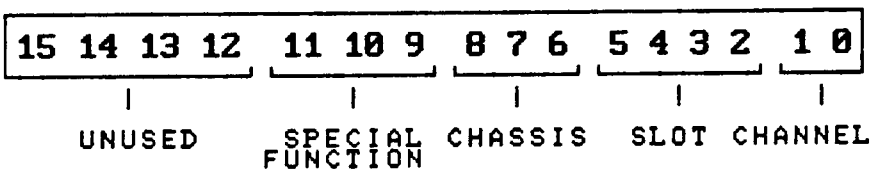


Figure 29. ANDS5400 16-Bit Digital Address

MODE	BIT 3	BIT 2	BIT 1	TYPE
7	1	1	1	DIG IN (DMA)
6	1	1	0	DIG OUT (DMA)
5	1	0	1	NOT USED
4	1	0	0	A/D (DMA)
3	0	1	1	DIG IN (P/O)
2	0	1	0	DIG OUT (P/O)
1	0	0	1	NOT USED
0	0	0	0	A/D (P/O)

Figure 30. ANDS5400 Q-Bus Interface Operating Modes

Source: Instruction Manual for ANDS5400, Book One (of two), Revised ed., Analogic Corporation, 1982.

NAME	ABBR.	READ/WRITE	BASE ADDRESS +
MASTER REGISTERS*	MR	R/W	(octal)- 000-376
CONTROL & STATUS	CSR	R/W	400
DIRECT MEMORY ACCESS 1	DMAR1	R/W	402
DIRECT MEMORY ACCESS 2	DMAR2	R/W	404
DIRECT MEMORY ACCESS EXTENSION	DMARX	R/W	406
TRANSACTION	TRR	R/W	410
FRONT PANEL	FPR	R	412

*If Master Registers are disabled, the board will not respond to relative addresses 000-376.

Figure 31. ANDS5400 Q-Bus Interface Registers

Source: Instruction Manual for ANDS5400, Book One (of two), Revised ed., Analogic Corporation, 1982.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERR	PI/O ERR	DMA ERR	LOCK	RMT	SSH	CLR	CLR1	READY	INT ENB.	54 ENB.	PART 2	MODE 2	MODE 1	MODE 0	G O
R	R	R	R/W	R	R	W	W	R	R/W	R/W	R	R/W	R/W	R/W	R/W

NOTE: R = Read Only W = Write Only R/W = Read/Write

Figure 32. ANDS5400 Q-Bus Interface Control and Status Register

Source: Instruction Manual for ANDS5400, Book One (of two), Revised ed., Analogic Corporation, 1982.

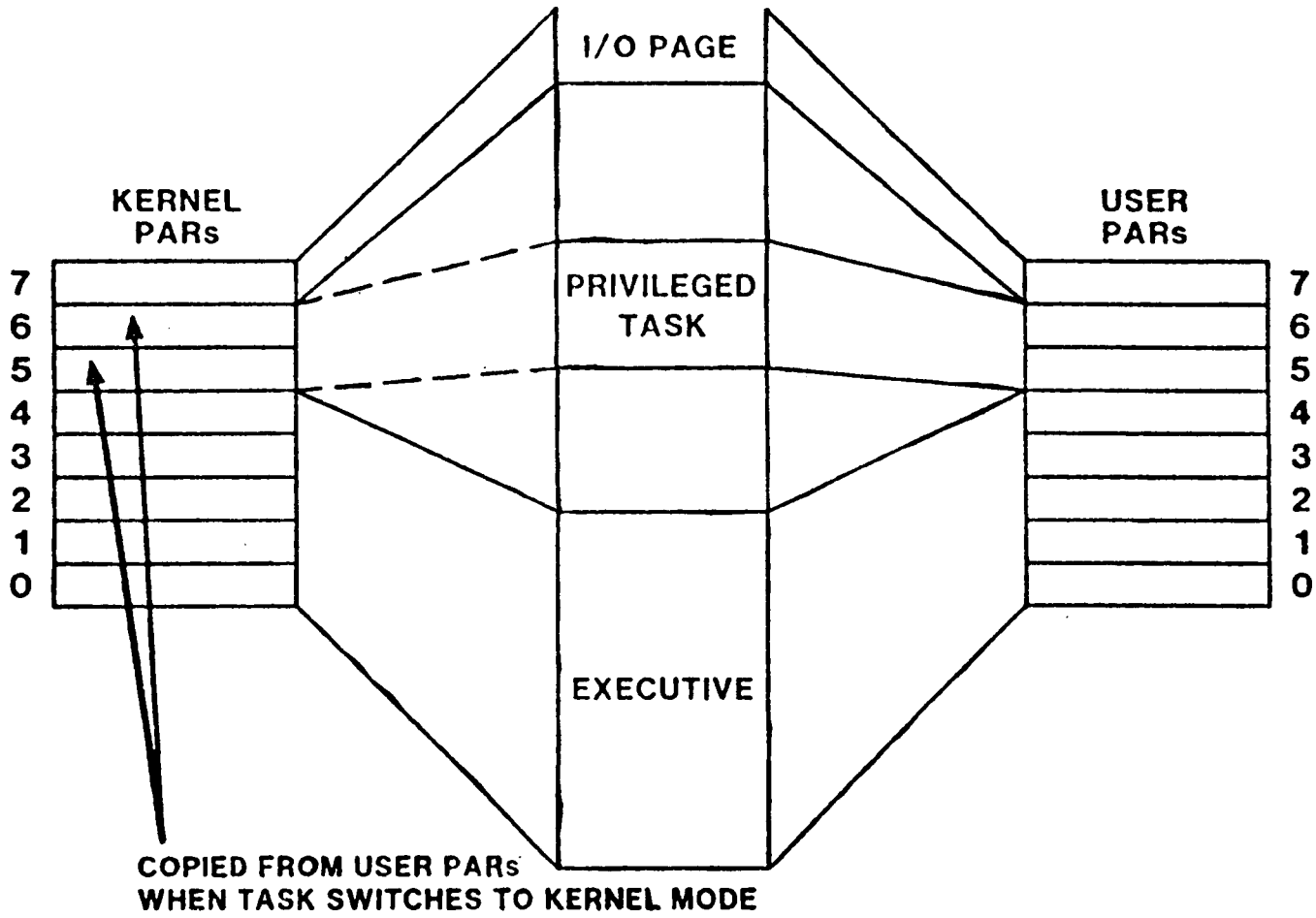


Figure 33. Task Mapping to RSX Executive

Source: RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual, Revised ed., Digital Equipment Corporation, Maynard, MA, July 1985.

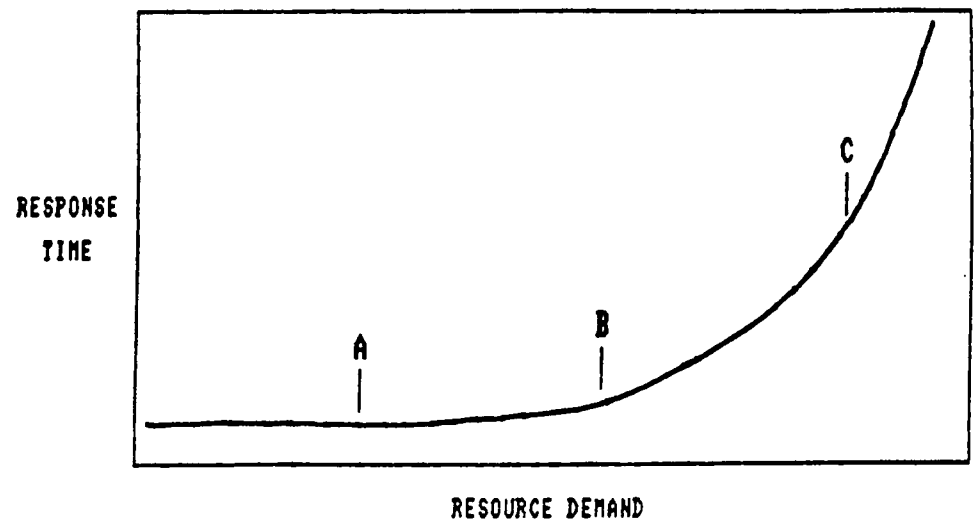


Figure 34. Response Time Versus Resource Demand

Source: McGlinchey, James A. and Frisbee, Alan E., RSX-11M/M+ Real-Time System Design and Performance Management, Digital Equipment Corporation, 1985.

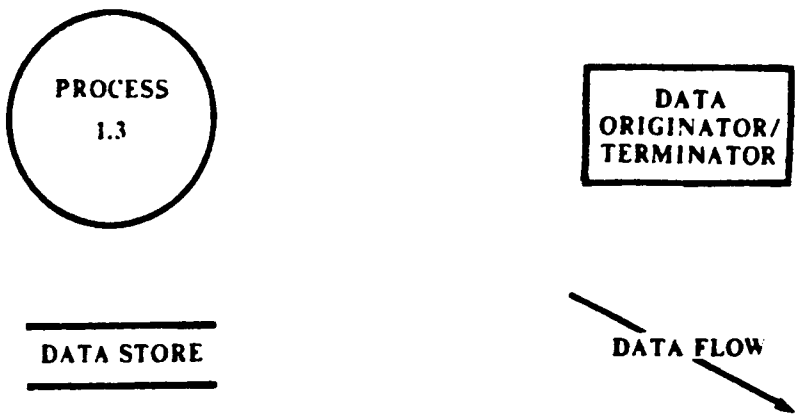


Figure 35. Data Flow Diagram Symbols

Source: Dickinson, Brian, Developing Structured Systems, Yourdon Press, New York, 1981.

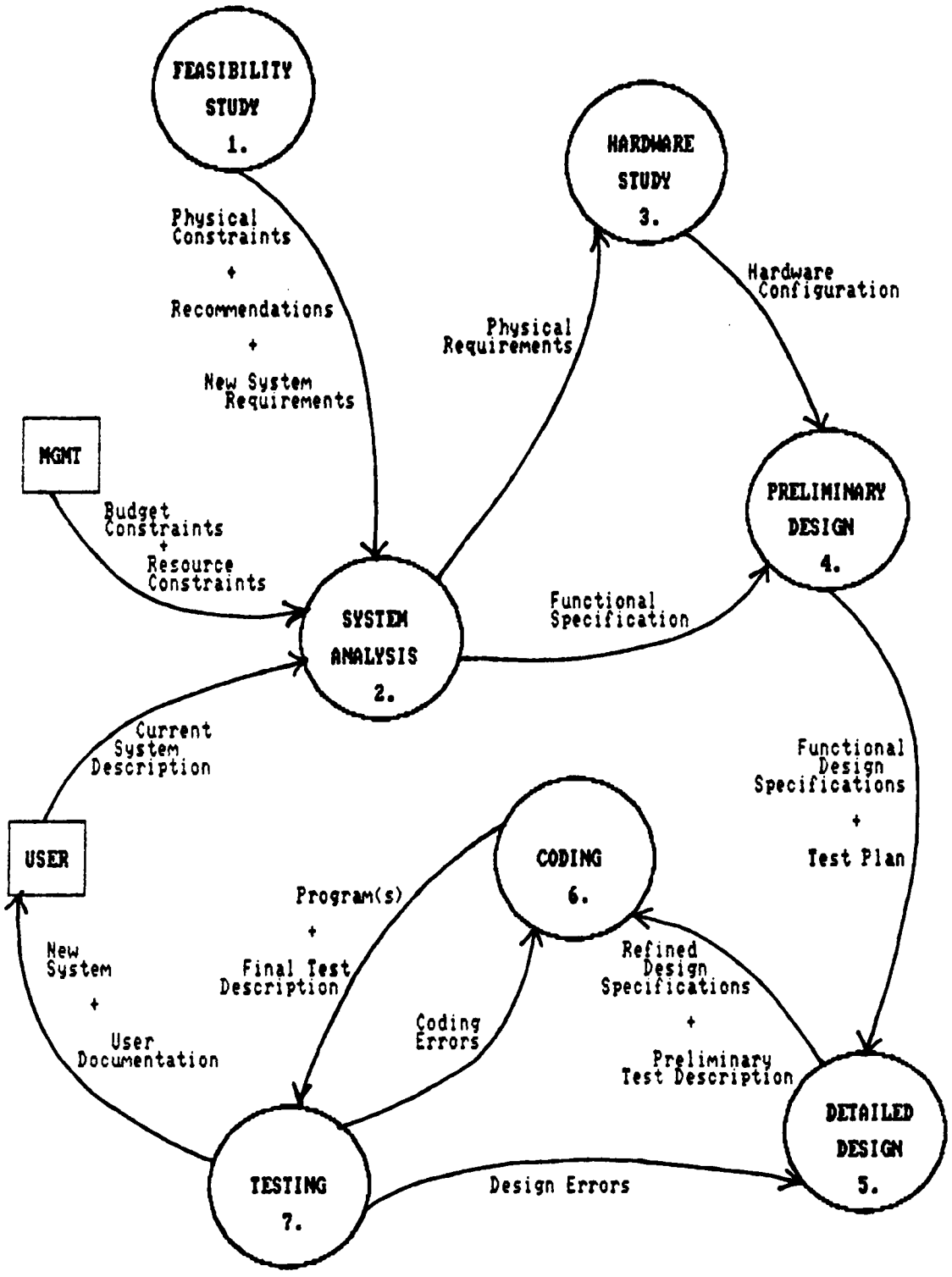


Figure 36. Data Flow Diagram of Main Phases of System Development

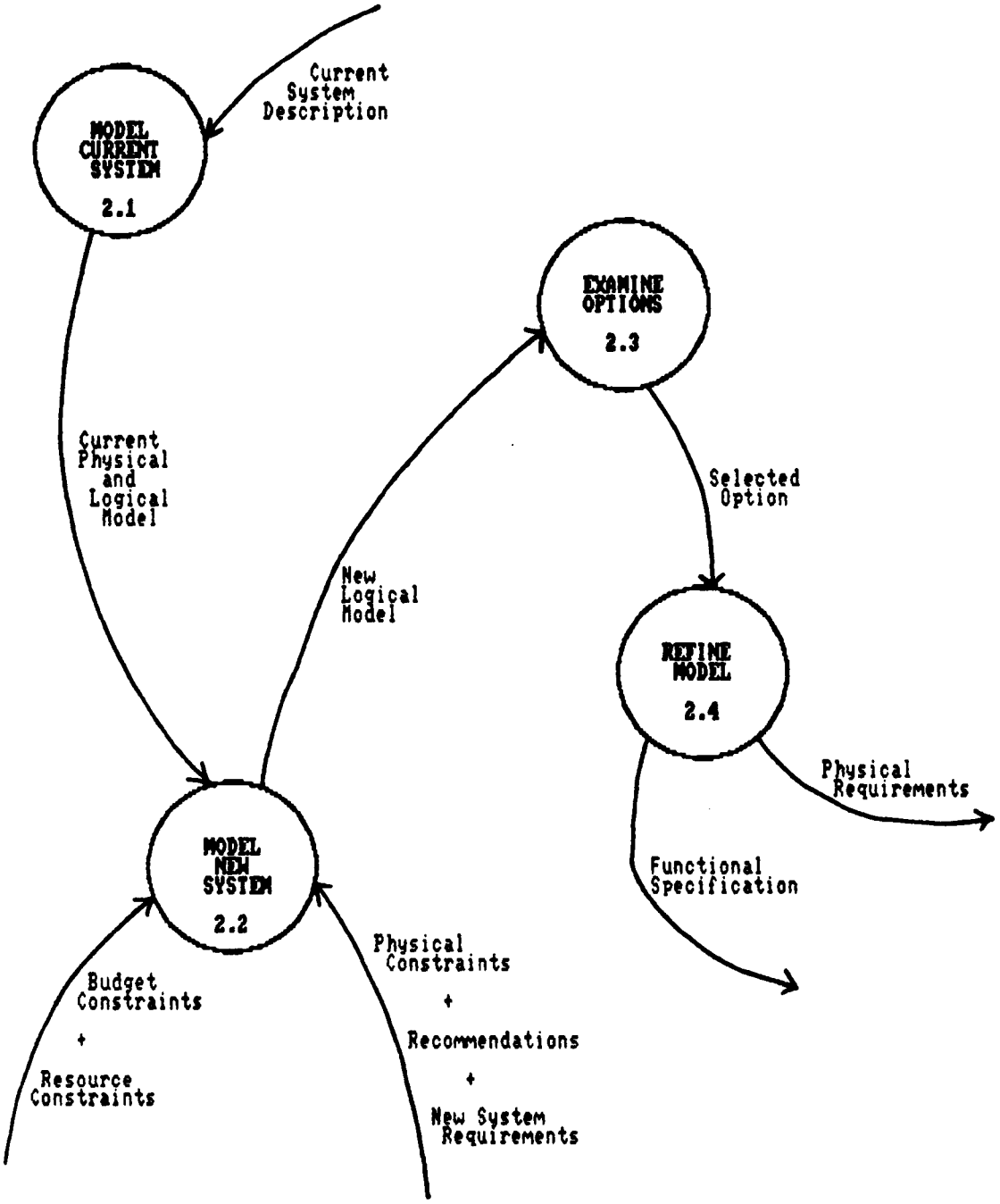


Figure 37. Sub-Level Data Flow Diagram of System Analysis Phase of System Development

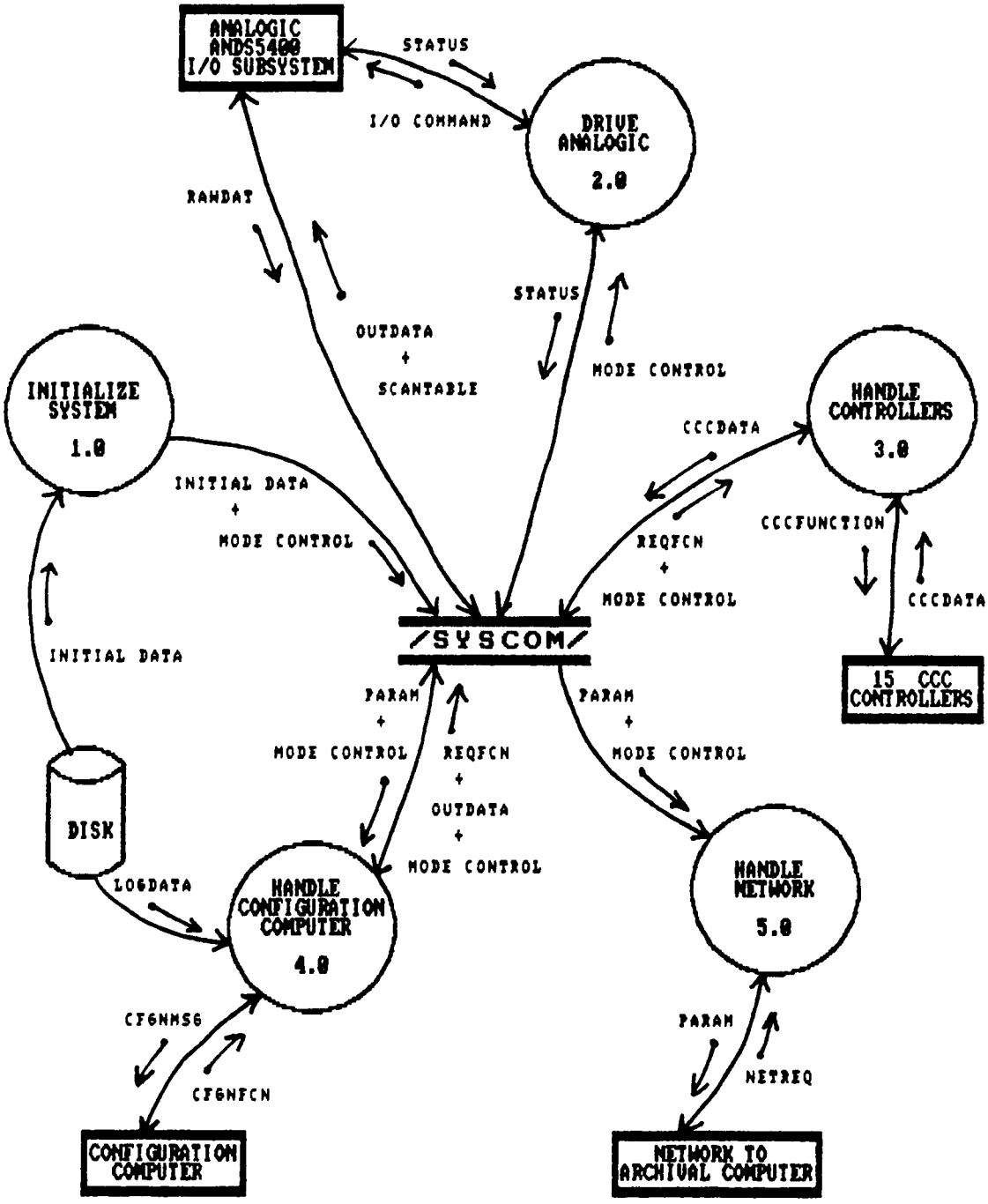


Figure 38. Software Specification Data Flow Diagram

Source: Belew, P. W., Report No. FSD-ER-B-86-15, Calspan Corporation/AEDC Division, Arnold Air Force Station, September 1986. (Not available outside Arnold AFB.)

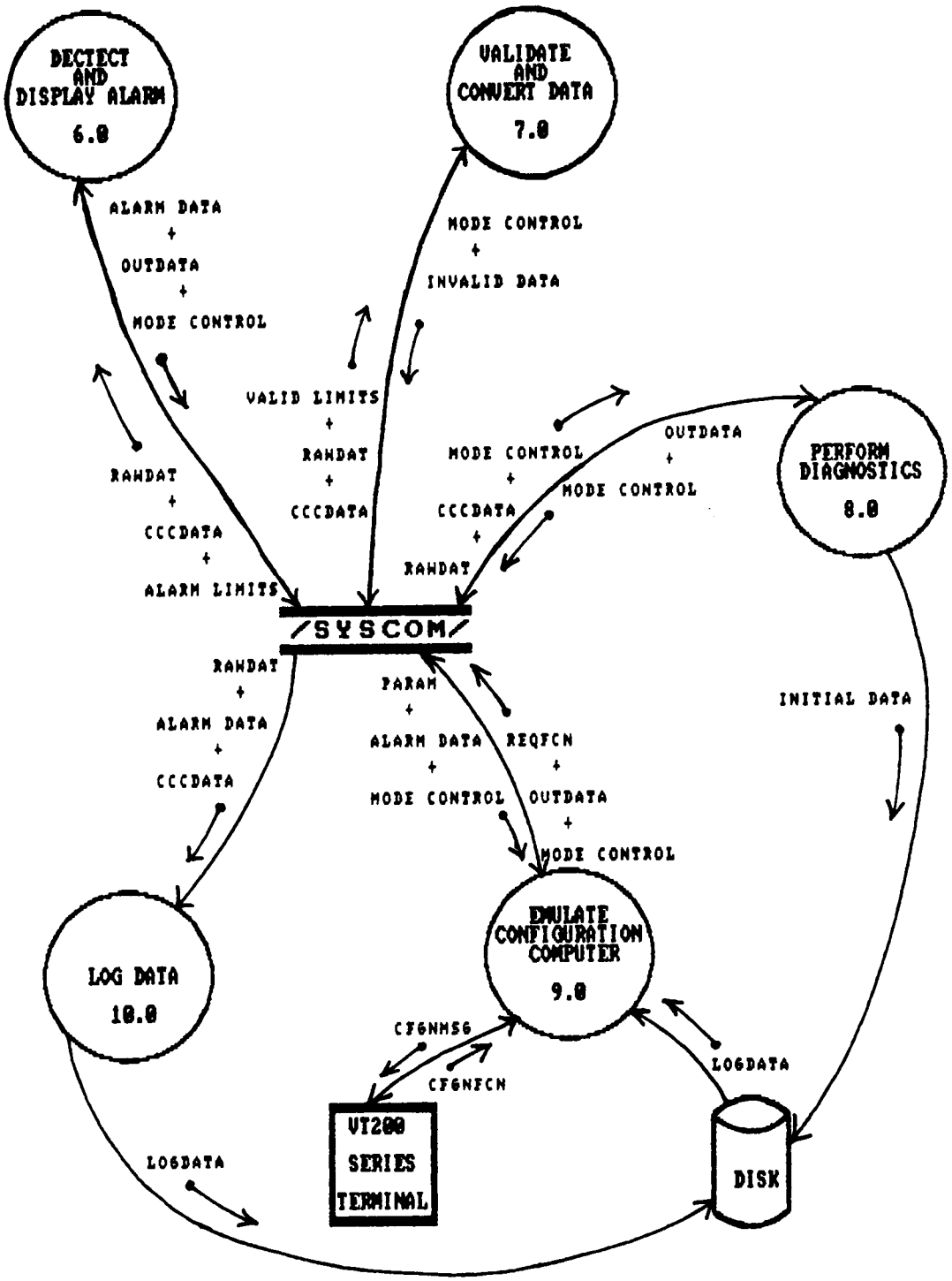


Figure 38. (continued)

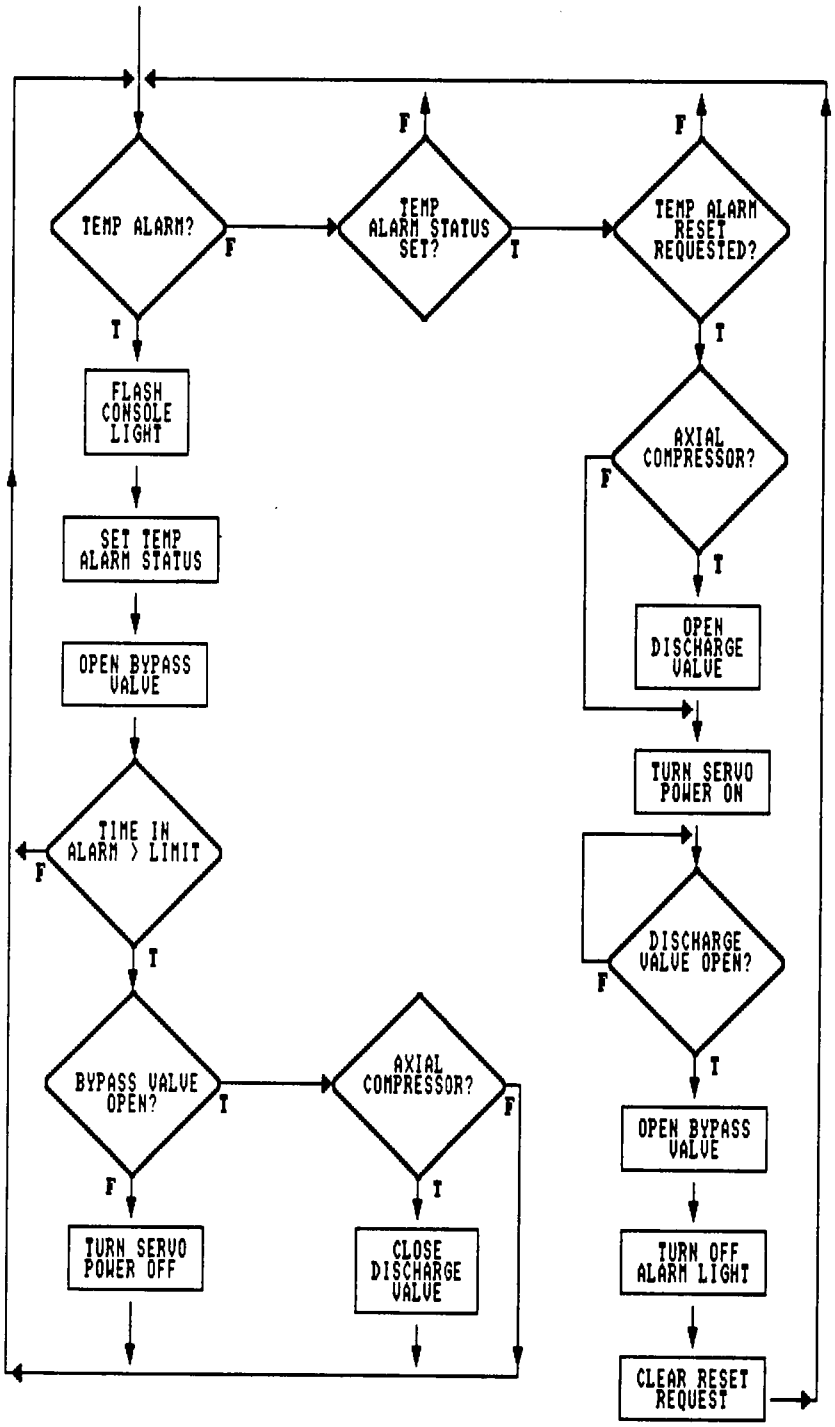


Figure 39. Specification of the Temperature Alarm Function

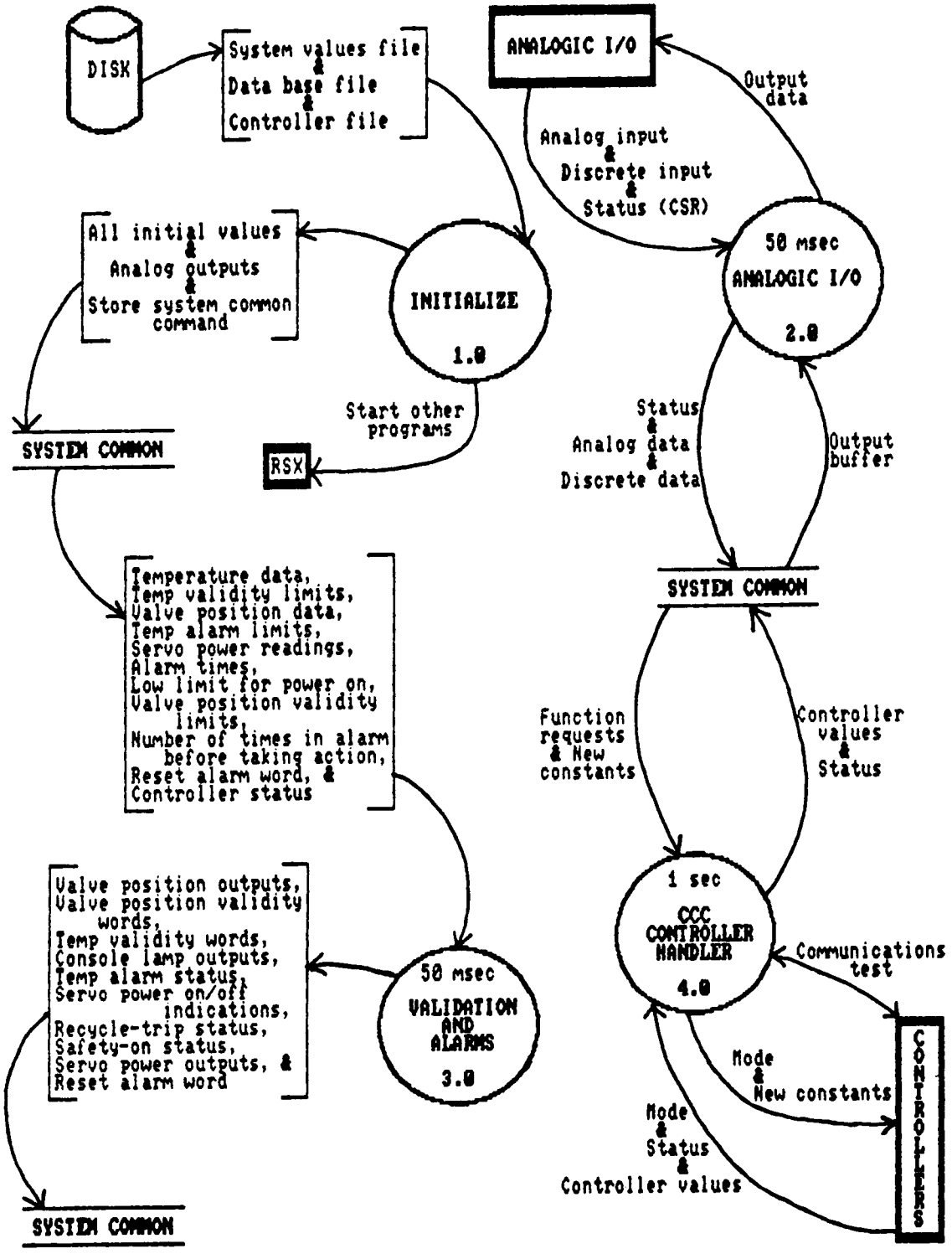


Figure 40. Preliminary Design Data Flow Diagram

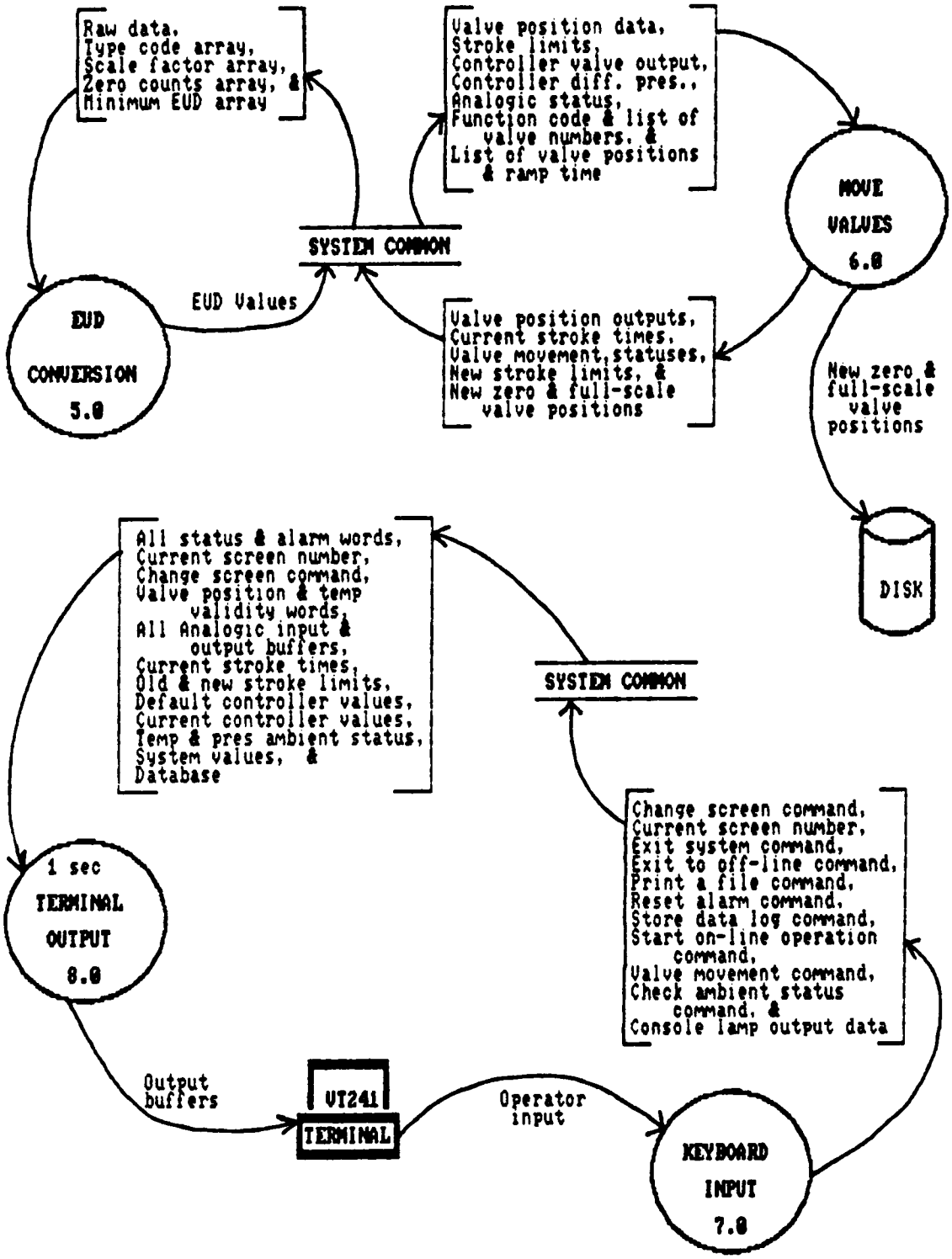


Figure 40. (continued)

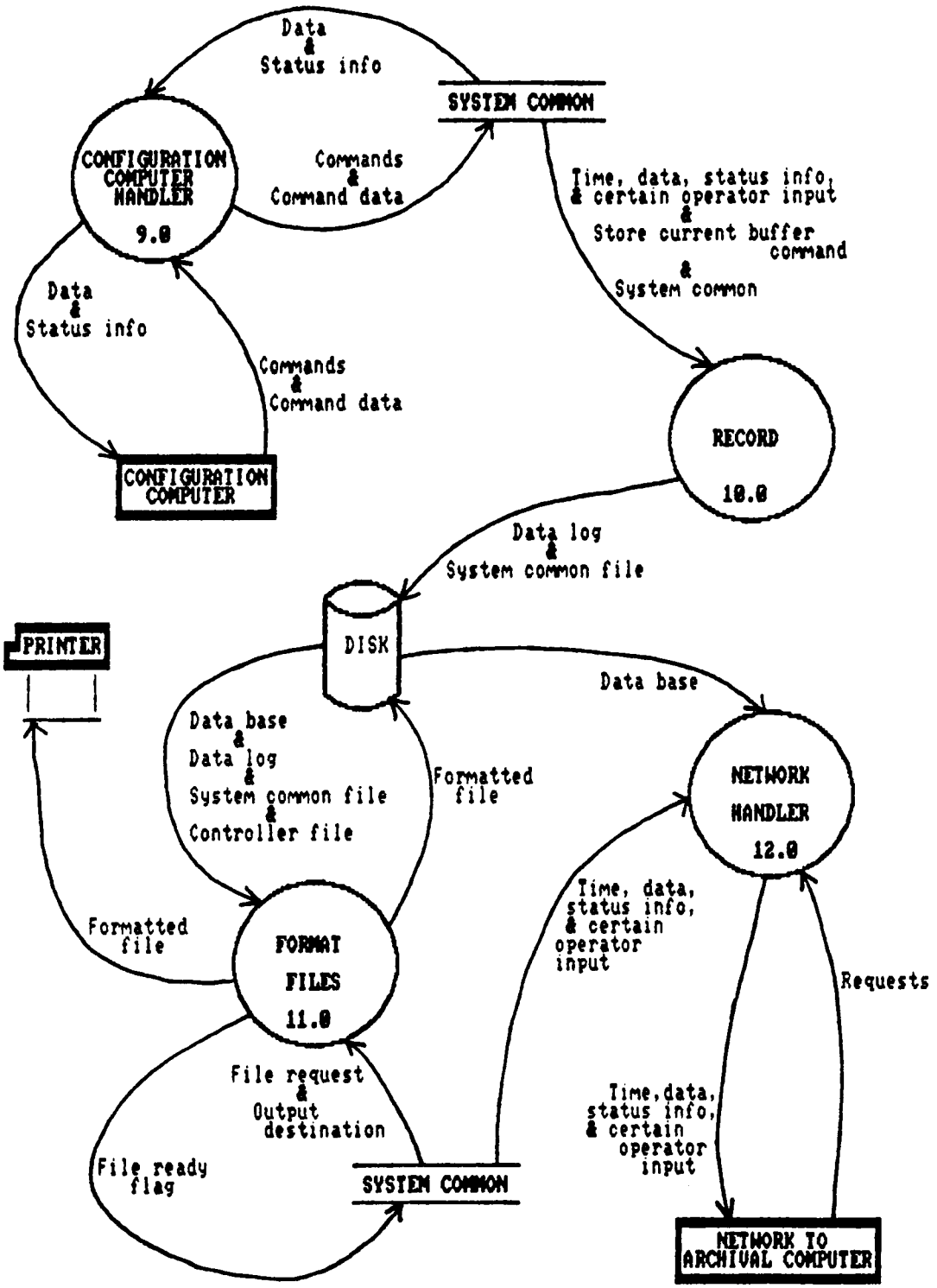


Figure 40. (continued)

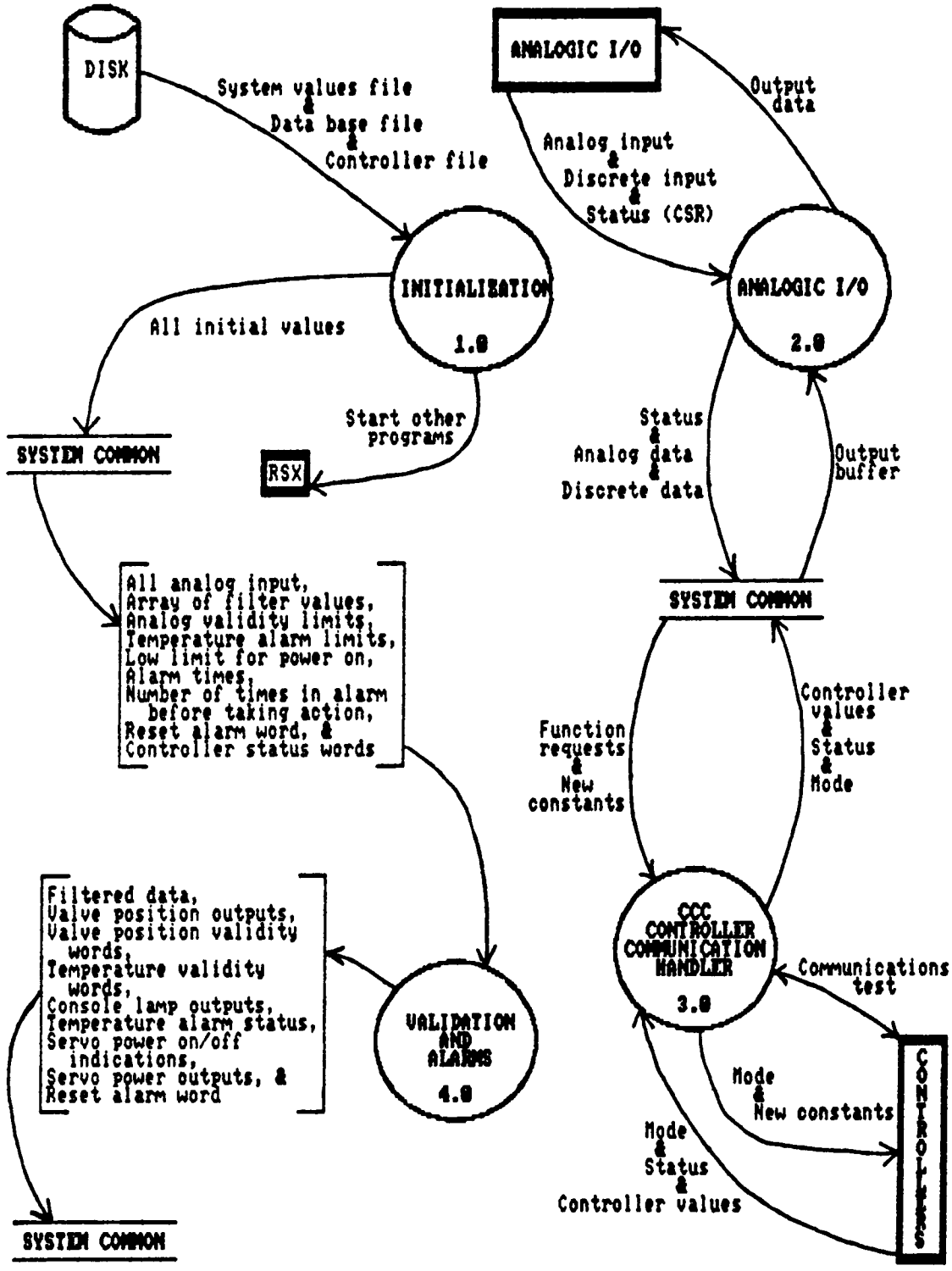


Figure 41. System Data Flow Diagram

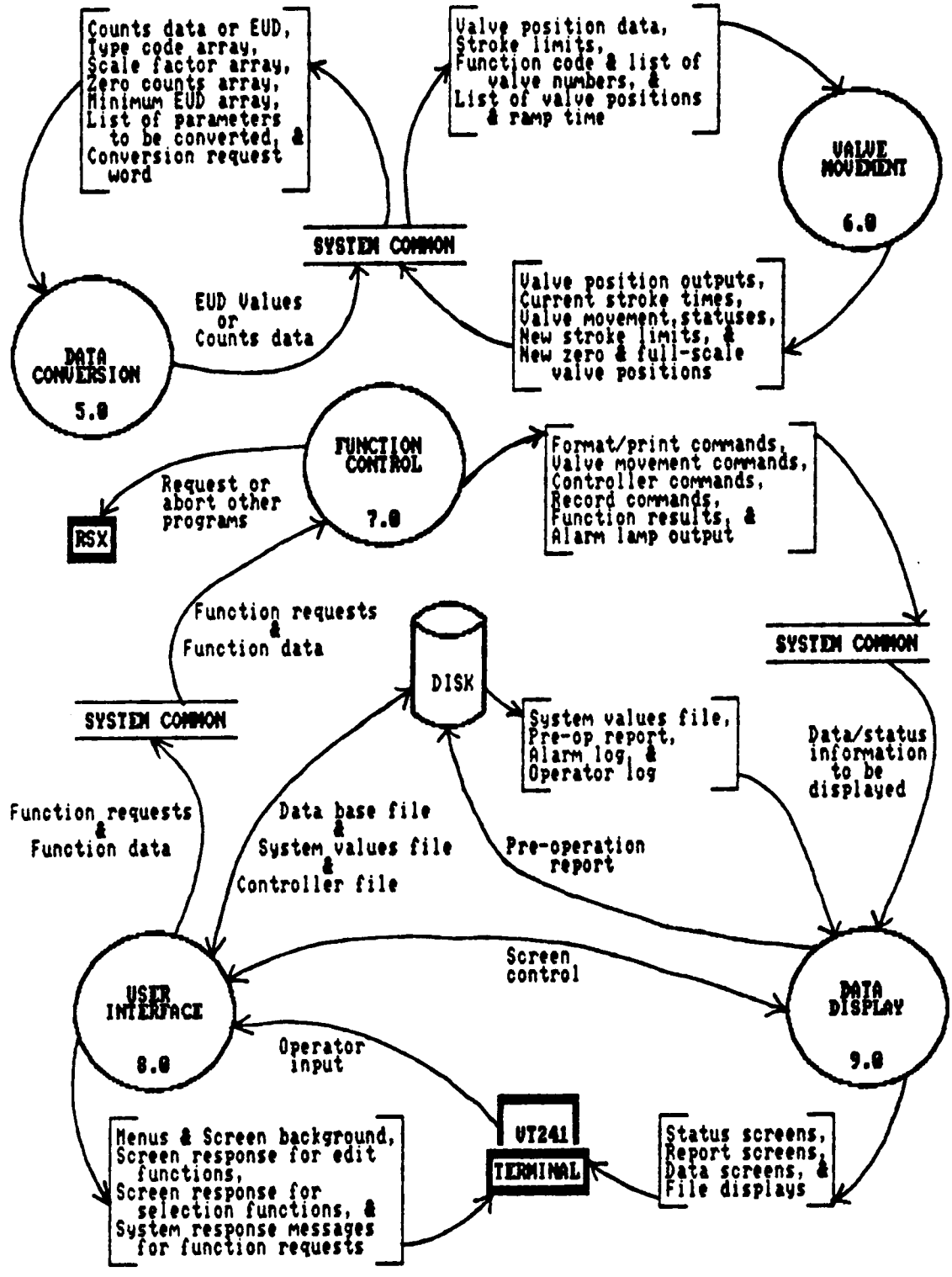


Figure 41. (continued)

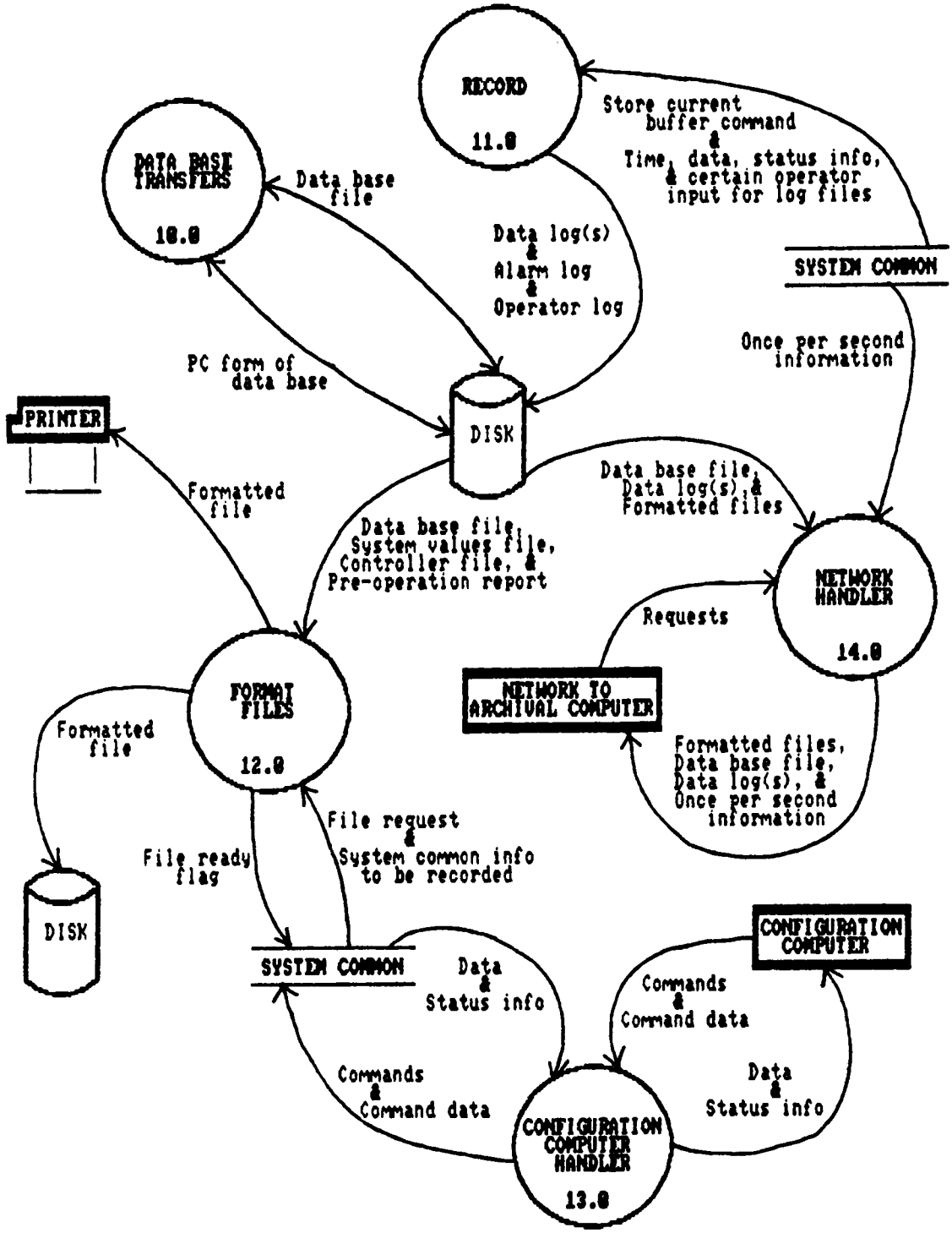


Figure 41. (continued)

V-PLANT OFF-LINE ANTI-SURGE CONTROL

F6 = PERFORM PRE-OPERATION CHECKS AND REPORT

F7 = START ANTI-SURGE CONTROL SYSTEM

F8 = EXERCISE ALARM LIGHTS

F9 = CHECK AMBIENT TEMPERATURES AND DIFFERENTIAL PRESSURES

F10 = VALVE FUNCTION MENU

F12 = CONTROLLER FUNCTION MENU

F13 = DATA BASE/SYSTEM CONSTANTS MENU

F14 = I/O SUBSYSTEM READINGS MENU

F17 = PRINT/DISPLAY DATA

F18 =

F19 = EXIT SURGE CONTROL SYSTEM

Figure 42. Off-Line Menu

VALVE FUNCTIONS

F6 = DISPLAY VALVE POSITION INPUTS

F7 = DISPLAY SERVO AMPLIFIER POWER INPUTS

F8 = DISPLAY VALVE POSITION OUTPUTS

F9 = DISPLAY SERVO AMPLIFIER POWER OUTPUTS

F10 = READ AND STORE CURRENT ZERO AND FULL-SCALE POSITIONS OF VALVES

F12 = ESTABLISH AND STORE NEW AVERAGE STROKE TIME FOR VALVE(S)

F13 = STROKE VALVE(S) AND REPORT

F14 = POSITION VALVE(S) AND REPORT

PRESS <ESC> FOR MAIN MENU

Figure 43. Valve Function Menu

PRINT/DISPLAY DATA

F6 = PRINT PARAMETER DATA BASE (FILE)
F7 = PRINT ALARM LOG
F8 = PRINT SYSTEM CONSTANTS
F9 = PRINT CONTROLLER CONSTANTS
F10 = PRINT RESIDENT COMMON
F12 = PRINT OPERATOR LOG
F13 = PRINT PREOP REPORT
F14 = DISPLAY ALARM LOG
HELP = DISPLAY OPERATOR LOG
DO = DISPLAY PREOP REPORT
F17 = DISPLAY RESIDENT COMMON
F18 = DISPLAY SYSTEM CONSTANTS

PRESS <ESC> FOR MAIN MENU

Figure 44. Print and/or Display Data Menu

V-PLANT ON-LINE ANTI-SURGE CONTROL

F6 = DISPLAY ALL STATUSES

F7 = STORE CURRENT DATA LOG

F8 = RESET ALARMS

F9 = DISPLAY ALL TEMPERATURES AND CONTROLLER PRESSURES

F10 = POSITION VALVE(S) AND REPORT

F12 = DISPLAY CURRENT AND STORED CONTROLLER PARAMETER VALUES

F13 = DATA BASE/SYSTEM CONSTANTS MENU

F14 = I/O SUBSYSTEM READINGS MENU

F17 = PRINT/DISPLAY DATA

F18 =

F19 = EXIT TO OFF-LINE MENU

Figure 45. On-Line Menu

I/O SUBSYSTEM READINGS

F6 = CONTROLLER AND COMPUTER OVERRIDE STATES

F7 = VALVE POSITION READINGS

F8 = VALVE POSITION OUTPUTS

F9 = SERVO AMPLIFIER POWER READINGS

F10 = SERVO AMPLIFIER POWER OUTPUTS

F12 = CONSOLE LIGHT OUTPUTS

F13 = INLET AND INTERNAL/DISCHARGE COMP TEMPERATURES

PRESS <ESC> FOR MAIN MENU

Figure 46. I/O Subsystem Readings Menu

DATA BASE/SYSTEM CONSTANTS FUNCTIONS

F6 = MODIFY DATA BASE

F7 = DISPLAY DATA BASE

F8 = PRINT DATA BASE

F9 = GENERATE SYSTEM DATA BASE FROM PC/DBASE FILE

F10 =

F12 = MODIFY SYSTEM VALUES

F13 = DISPLAY SYSTEM VALUES

F14 = PRINT SYSTEM VALUES

PRESS <ESC>FOR MAIN MENU

Figure 47. Data Base and System Constants Function Menu

Strokes/Avg	3	C81 Alarm Time	100
Valve Tolerance	0.10	C91 Alarm Time	100
Power On Level	3300	C101 Alarm Time	100
C11 Alarm Time	200	C111 Alarm Time	100
C12 Alarm Time	200	C11 Diff Alm Cts	950
C13 Alarm Time	200	C12 Diff Alm Cts	950
C14 Alarm Time	200	C13 Diff Alm Cts	950
C21 Alarm Time	200	C14 Diff Alm Cts	950
C22 Alarm Time	200	C21 Diff Alm Cts	950
C31 Alarm Time	100	C22 Diff Alm Cts	950
C41 Alarm Time	100	C31 Diff Alm Cts	950
C51 Alarm Time	100	C41 Diff Alm Cts	950
C61 Alarm Time	100	C51 Diff Alm Cts	950
C71 Alarm Time	100	C61 Diff Alm Cts	950

DISPLAY MODE
F7 - Edit mode
F11 [Esc] - Close display

EDIT MODE
F16, DO - Store an edited record
F11 [Esc] - Return to display
Arrow keys - Field movement

NOTE: Alarm time is in units of loop time (67.7 msec)

Figure 48. Display and/or Edit System Constants

COMPRESSOR 11	CURRENT VALUE MEMORY	CURRENT VALUE FILE	LOW RANGE MEMORY	LOW RANGE FILE	HIGH RANGE MEMORY	HIGH RANGE FILE
Analog 1, DeltaH	2.0					
Analog 3, DH range	1.8					
Analog 2, DeltaP	0.0					
Analog 4, DP range	1.8					
Sum B2	0.1	0.1	0.5	0.5	0.5	0.5
RT	1.9	1.1	1.1	1.1	1.9	1.9
SO	1.0	1.0	1.0	1.0	1.0	1.0
PV	8.7	5.4				
Set Point.	6.8	3.6				
FA Mode	8.	8.				
PB	100.	100.				
Kr	5.0	5.0				
Tf	3.0	3.0				
K coefficient	.240	.240	.240	.240	.240	.240
Control line, B1	2.0	1.0	1.0	1.0	2.0	2.0
RT amplitude, C1	10.0	10.0	20.0	20.0	20.0	20.0
RT time, C2.	1.00	1.00	0.52	0.52	0.52	0.52
MODE (1-AUTO 2-MOR)	2.	2.				
STATUS (1-RT 2-SO)	0.	0.				

DISPLAY MODE
F11-Exit F8-Select Controller
F7-Edit Mode

EDIT MODE
F11-Display Mode F8-Save Ranges
F7-Save Current F9-Load to Controller

Figure 49. Display and/or Edit Controller Constants

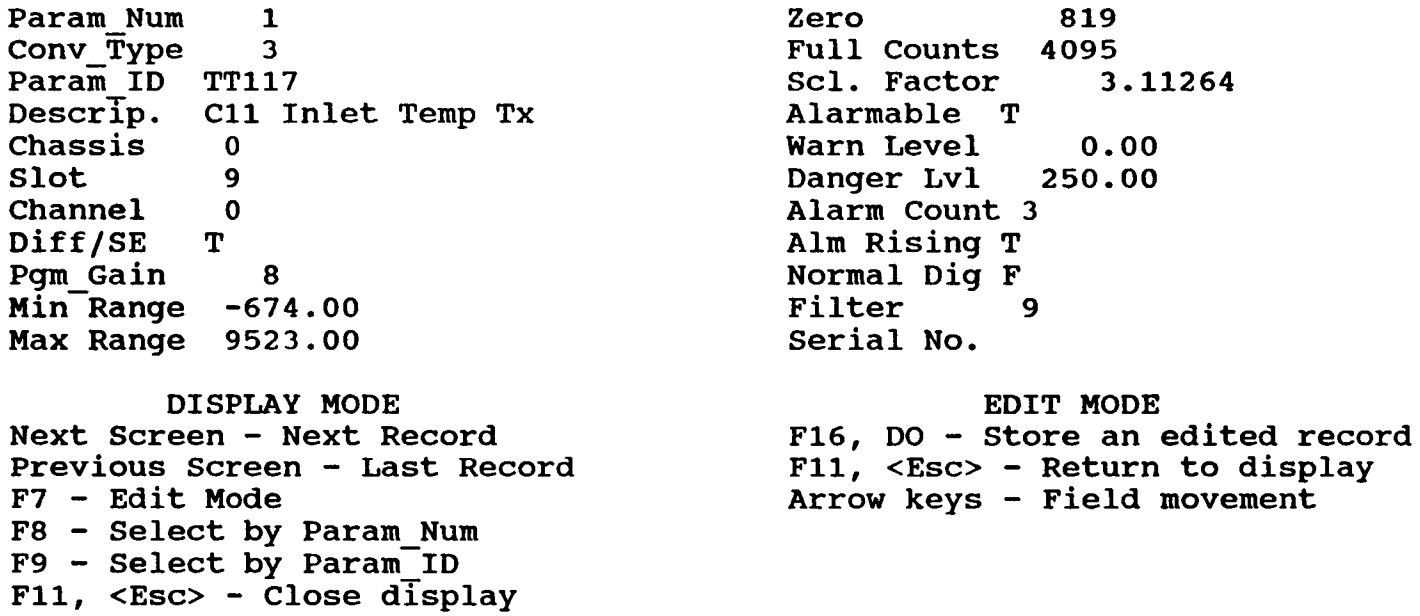


Figure 50. Display and/or Edit Data Base

VALVE/VALVE NO./ TYPE	PRESENT VOLTS	PRESENT COUNTS	COUNTS RANGE	VOLTS RANGE	CHAS-SLOT-CHAN			OCTAL ADDRESS
113 /Bypass	9.81	4016	0-4095	0.0-10.0	0	15	1	100702
123 /Bypass	9.80	4011	0-4095	0.0-10.0	0	15	3	100706
133 /Bypass	9.75	3991	0-4095	0.0-10.0	0	15	5	100712
143 /Bypass	9.83	4024	0-4095	0.0-10.0	0	15	7	100716
213 /Bypass	9.79	4010	0-4095	0.0-10.0	0	15	9	100722
223 /Bypass	9.85	4035	0-4095	0.0-10.0	0	15	11	100726
313 /Bypass	9.82	4020	0-4095	0.0-10.0	0	15	12	100730
413 /Bypass	9.76	3995	0-4095	0.0-10.0	0	15	13	100732
513 /Bypass	9.82	4022	0-4095	0.0-10.0	0	15	14	100734
613 /Bypass	9.71	3976	0-4095	0.0-10.0	0	15	15	100736
713 /Bypass	9.77	4000	0-4095	0.0-10.0	0	16	0	100740
813 /Bypass	9.83	4023	0-4095	0.0-10.0	0	16	1	100742
913 /Bypass	9.80	4013	0-4095	0.0-10.0	0	16	2	100744
1013 /Bypass	9.60	3930	0-4095	0.0-10.0	0	16	3	100746
1113 /Bypass	9.67	3960	0-4095	0.0-10.0	0	16	4	100750
115 /Discharge	9.83	4025	0-4095	0.0-10.0	0	15	0	100700
125 /Discharge	9.71	3977	0-4095	0.0-10.0	0	15	2	100704
135 /Discharge	9.73	3986	0-4095	0.0-10.0	0	15	4	100710
145 /Discharge	9.82	4021	0-4095	0.0-10.0	0	15	6	100714
215 /Discharge	9.68	3962	0-4095	0.0-10.0	0	15	8	100720
225 /Discharge	9.86	4039	0-4095	0.0-10.0	0	15	10	100724

PRESS <ESC> FOR I/O SUBSYSTEM READINGS MENU

Figure 51. Servo Amplifier Power Readings

PRE-OPERATION REPORT
19-DEC-91 15:47:01

	Compressor Number														
- is normal	1	1	1	1	2	2	3	4	5	6	7	8	9	10	11
X is abnormal	1	2	3	4	1	2	1	1	1	1	1	1	1	1	1
COMPUTER OVERRIDE	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X
CONTROLLER OVERRIDE	-	-	-	-	X	X	X	X	X	X	X	X	X	X	X
CCC IN MANUAL MODE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CCC AMBIENT DELTA-P	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-
CCC AMBIENT DELTA-H	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-
AMBIENT INLET TEMP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
AMBIENT INT/DIS TEMP	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X
BYPASS SERVO POWER OFF	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DISCHG SERVO POWER OFF	-	-	-	-	-	-									

PRESS <NEXT SCREEN> OR <PREV SCREEN> FOR SECOND PAGE PRESS <ESC> FOR MAIN MENU

Figure 52. First Page of Pre-Operation Report

PRE-OPERATION REPORT
19-DEC-91 15:47:01

	Compressor Number														
- is normal X is abnormal	1 1	1 2	1 3	1 4	2 1	2 2	3 1	4 1	5 1	6 1	7 1	8 1	9 1	1 0 1	1 1 1
INVALID BYPASS VALVE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
INVALID DISCHRG VALVE	-	-	-	-	-	-									
BYPASS VALVE OPEN	-	-	-	-	-	-	-	-	-	-	-	-	-	X	-
DISCHARGE VALVE OPEN	-	-	-	-	-	-									
CONTROLLER TIME-OUT	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X
CCC EXCEPTION RESPONSE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CCC COMMUNICATIONS	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X
CCC CURRENT VS. STORED	X	X	X	X	X	X	-	X	-	X	-	-	-	X	-

ANALOGIC CSR = 000000 * NO ERRORS

PRESS <NEXT SCREEN> OR <PREV SCREEN> FOR FIRST PAGE PRESS <ESC> FOR MAIN MENU

Figure 53. Second Page of Pre-Operation Report

COMP NO.	COMPUTER OVERRIDE	CHAS-SLOT-POINT			OCTAL ADDRESS	CONTROLLER OVERRIDE	CHAS-SLOT-POINT			OCTAL ADDRESS
11	0	0	12	0	000054	0	0	12	16	000056
12	0	0	12	1	000054	0	0	12	17	000056
13	0	0	12	2	000054	0	0	12	18	000056
14	0	0	12	3	000054	0	0	12	19	000056
21	0	0	12	4	000054	1	0	12	20	000056
22	0	0	12	5	000054	1	0	12	21	000056
31	0	0	12	6	000054	1	0	12	22	000056
41	0	0	12	7	000054	1	0	12	23	000056
51	0	0	12	8	000054	1	0	12	24	000056
61	0	0	12	9	000054	1	0	12	25	000056
71	0	0	12	10	000054	0	0	12	26	000056
81	0	0	12	11	000054	0	0	12	27	000056
91	0	0	12	12	000054	0	0	12	28	000056
101	1	0	12	13	000054	1	0	12	29	000056
111	1	0	12	14	000054	1	0	12	30	000056

0 -> NORMAL 1 -> OVERRIDE

PRESS <ESC> FOR I/O SUBSYSTEM READINGS MENU

Figure 54. Computer and Controller Override States

SELECT VALVES, POSITIONS, AND RAMP TIME

Span and Zero

	VALVE NO.		VALVE NO.
1	V113 - C11 Bypass		V813 - C81 Bypass
	V123 - C12 Bypass	1	V913 - C91 Bypass
	V133 - C13 Bypass		V1013 - C101 Bypass
1	V143 - C14 Bypass		V1113 - C111 Bypass
	V213 - C21 Bypass	1	V115 - C11 Discharge
	V223 - C22 Bypass		V125 - C12 Discharge
	V313 - C31 Bypass	1	V135 - C13 Discharge
1	V413 - C41 Bypass		V145 - C14 Discharge
	V513 - C51 Bypass		V215 - C21 Discharge
	V613 - C61 Bypass	1	V225 - C22 Discharge
	V713 - C71 Bypass		Ramp Time

F6 - Sel/Desel all bypass valves
F7 - Sel/Desel all discharge valves
F8 - Select/Deselect a valve

F11, <ESC> - Return to off line menu
F16, <DO> - Execute the selection
Arrow keys - Cursor position

Figure 55. Select Valves for Span and Zero Function

COMP NO.	% FULL SCALE DELTA P	FLOW	INLET DEG F	INT/DIS DEG F
11	0.0	2.0	108.1	134.7
12	0.0	0.0	101.1	143.1
13	0.0	0.9	104.7	126.1
14	0.2	0.0	103.3	135.1
21	0.4	0.0	105.3	148.0
22	0.0	0.4	88.2	115.7
31	0.0	0.0	97.6	124.5
41	0.7	0.1	93.6	153.6
51	0.0	0.3	92.7	148.5
61	0.0	0.0	69.8	95.2
71	4.0***	8.2***	78.2	73.0
81	0.2	0.6	107.5	98.0
91	0.6	0.7	91.8	99.5
101	0.0	0.0	78.5	600.1***
111	0.0	0.0	89.8	21.1***

*** -> OUT OF AMBIENT RANGE

PRESS <ESC> FOR MAIN MENU

Figure 56. Check Ambient Pressures and Temperatures

	COMPRESSOR NUMBER														
- is normal X is abnormal	1 1	1 2	1 3	1 4	2 1	2 2	3 1	4 1	5 1	6 1	7 1	8 1	9 1	0 1	1 1
COMPUTER OVERRIDE	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X
CONTROLLER OVERRIDE	-	-	-	-	X	X	X	X	X	X	X	X	X	X	X
INVALID TEMPERATURE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CONTROLLER ERROR	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X
FAIL FLOW RNG CHNGE	-	-	-	-	-	-									
SERVO POWER OFF	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RECYCLE TRIP	-	X	-	X	X	-	-	-	-	-	-	-	-	-	-
SAFETY ON	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TEMPERATURE ALARM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LATCHED ALARM	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

PRESS <ESC> FOR MAIN MENU

Figure 57. On-Line Status Display

COMP NO.	INLET DEG F	COUNTS	DEG F RANGE	COUNTS RANGE	VOLTS RANGE	CHAS-SLOT-CHAN			OCTAL ADDRESS
11	76.	1349	0-400	819-4095	0.2- 1.2	0	9	0	130400
12	76.	1347	0-400	819-4095	0.2- 1.2	0	9	2	130404
13	75.	1341	0-400	819-4095	0.2- 1.2	0	9	4	130410
14	75.	1345	0-400	819-4095	0.2- 1.2	0	9	6	130414
21	74.	1335	0-400	819-4095	0.2- 1.2	0	9	8	130420
22	73.	1326	0-400	819-4095	0.2- 1.2	0	9	10	130424
31	74.	1337	0-400	819-4095	0.2- 1.2	0	9	12	130430
41	76.	1349	0-400	819-4095	0.2- 1.2	0	9	14	130434
51	81.	1385	0-400	819-4095	0.2- 1.2	0	10	0	130440
61	78.	1360	0-400	819-4095	0.2- 1.2	0	10	2	130444
71	77.	1358	0-400	819-4095	0.2- 1.2	0	10	4	130450
81	91.	1356	0-400	819-4095	0.2- 1.2	0	10	6	130454
91	78.	1360	0-400	819-4095	0.2- 1.2	0	10	8	130460
101	74.	1335	0-400	819-4095	0.2- 1.2	0	10	10	130464
111	72.	1320	0-400	819-4095	0.2- 1.2	0	10	12	130470

PRESS <NEXT SCREEN> FOR INTERNAL/DISCHARGE TEMPERATURES

PRESS <ESC> FOR I/O SUBSYSTEM READINGS MENU

Figure 58. Inlet Temperature Readings

SELECT VALVES, POSITIONS, AND RAMP TIME

Position Valves

%OPEN	VALVE NO.	%OPEN	VALVE NO.
60	V113 - C11 Bypass		V813 - C81 Bypass
60	V123 - C12 Bypass		V913 - C91 Bypass
	V133 - C13 Bypass		V1013 - C101 Bypass
	V143 - C14 Bypass		V1113 - C111 Bypass
	V213 - C21 Bypass	25	V115 - C11 Discharge
40	V223 - C22 Bypass		V125 - C12 Discharge
	V313 - C31 Bypass		V135 - C13 Discharge
	V413 - C41 Bypass	50	V145 - C14 Discharge
	V513 - C51 Bypass		V215 - C21 Discharge
100	V613 - C61 Bypass		V225 - C22 Discharge
	V713 - C71 Bypass	30.	Ramp Time

F6 - Sel/Desel all bypass valves
F7 - Sel/Desel all discharge valves
F8 - Select/Deselect a valve
F11, <ESC> - Return to off line menu
F16, <DO> - Execute the selection
Arrow keys - Cursor position

Figure 59. Select Valves, Positions, and Ramp Time for Position Valve(s) Function

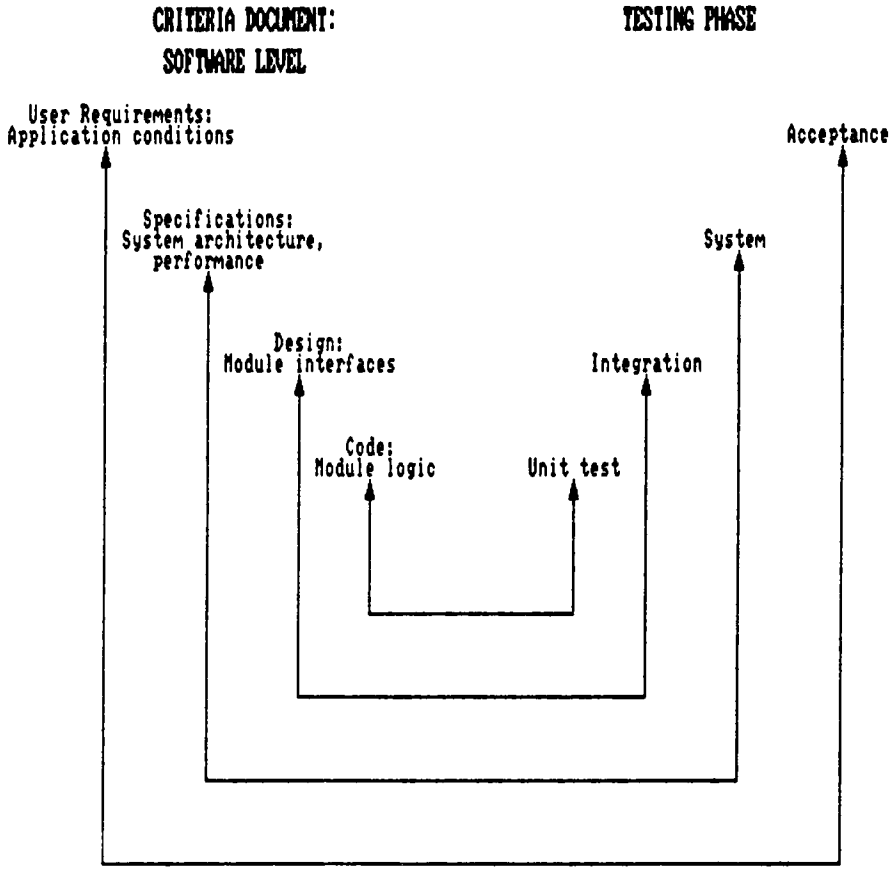


Figure 60. Relationship Between Testing Phases and Software Development Phases

APPENDIX B

I/O SUBSYSTEM PARAMETER LIST

Analog Parameters

Description	Analogic Slot/Channel	Address (octal)
C11 Dischg Valve Position Input	13/0	130600
C11 Bypass " " "	13/1	130602
C12 Dischg " " "	13/2	130604
C12 Bypass " " "	13/3	130606
C13 Dischg " " "	13/4	130610
C13 Bypass " " "	13/5	130612
C14 Dischg " " "	13/6	130614
C14 Bypass " " "	13/7	130616
C21 Dischg " " "	13/8	130620
C21 Bypass " " "	13/9	130622
C22 Dischg " " "	13/10	130624
C22 Bypass " " "	13/11	130626
C31 Bypass " " "	13/12	130630
C41 Bypass " " "	13/13	130632
C51 Bypass " " "	13/14	130634
C61 Bypass " " "	13/15	130636
C71 Bypass " " "	14/0	130640
C81 Bypass " " "	14/1	130642
C91 Bypass " " "	14/2	130644
C101 Bypass " " "	14/3	130646
C111 Bypass " " "	14/4	130650
C11 Dis. Valve Servo Power Input	15/0	100700
C11 Byp. " " " "	15/1	100702
C12 Dis. " " " "	15/2	100704
C12 Byp. " " " "	15/3	100706
C13 Dis. " " " "	15/4	100710
C13 Byp. " " " "	15/5	100712
C14 Dis. " " " "	15/6	100714
C14 Byp. " " " "	15/7	100716
C21 Dis. " " " "	15/8	100720
C21 Byp. " " " "	15/9	100722
C22 Dis. " " " "	15/10	100724
C22 Byp. " " " "	15/11	100726
C31 Byp. " " " "	15/12	100730
C41 Byp. " " " "	15/13	100732
C51 Byp. " " " "	15/14	100734
C61 Byp. " " " "	15/15	100736
C71 Byp. " " " "	16/0	100740
C81 Byp. " " " "	16/1	100742
C91 Byp. " " " "	16/2	100744
C101 Byp. " " " "	16/3	100746
C111 Byp. " " " "	16/4	100750

Analog Parameters

Description	Analogic Slot/Channel	Address (octal)
C11 Inlet Temperature Input	9/0	130400
C11 Internal " "	9/1	130402
C12 Inlet " "	9/2	130404
C12 Internal " "	9/3	130406
C13 Inlet " "	9/4	130410
C13 Internal " "	9/5	130412
C14 Inlet " "	9/6	130414
C14 Internal " "	9/7	130416
C21 Inlet " "	9/8	130420
C21 Internal " "	9/9	130422
C22 Inlet " "	9/10	130424
C22 Internal " "	9/11	130426
C31 Inlet " "	9/12	130430
C31 Internal " "	9/13	130432
C41 Inlet " "	9/14	130434
C41 Internal " "	9/15	130436
C51 Inlet " "	10/0	130440
C51 Internal " "	10/1	130442
C61 Inlet " "	10/2	130444
C61 Internal " "	10/3	130446
C71 Inlet " "	10/4	130450
C71 Discharge " "	10/5	130452
C81 Inlet " "	10/6	130454
C81 Discharge " "	10/7	130456
C91 Inlet " "	10/8	130460
C91 Discharge " "	10/9	130462
C101 Inlet " "	10/10	130464
C101 Discharge " "	10/11	130466
C111 Inlet " "	10/12	130470
C111 Discharge " "	10/13	130472

Analog Parameters

Description	Analogic Slot/Channel	Address (octal)
C11 Dischg Valve Position Output	1/0	000000
C11 Bypass " " "	1/1	000001
C12 Dischg " " "	1/2	000002
C12 Bypass " " "	1/3	000003
C13 Dischg " " "	2/0	000004
C13 Bypass " " "	2/1	000005
C14 Dischg " " "	2/2	000006
C14 Bypass " " "	2/3	000007
C21 Dischg " " "	3/0	000010
C21 Bypass " " "	3/1	000011
C22 Dischg " " "	3/2	000012
C22 Bypass " " "	3/3	000013
C31 Bypass " " "	4/0	000014
C41 Bypass " " "	4/1	000015
C51 Bypass " " "	4/2	000016
C61 Bypass " " "	4/3	000017
C71 Bypass " " "	5/0	000020
C81 Bypass " " "	5/1	000021
C91 Bypass " " "	5/2	000022
C101 Bypass " " "	5/3	000023
C111 Bypass " " "	6/1	000024

Discrete Parameters

Description	Analogic Slot-Bit	Address (octal)
C11 Bypass Valve Servo Power Control	7-0 (LSB)	30
C12 " " " " "	7-1	"
C13 " " " " "	7-2	"
C14 " " " " "	7-3	"
C21 " " " " "	7-4	"
C22 " " " " "	7-5	"
C31 " " " " "	7-6	"
C41 " " " " "	7-7 (MSB)	"
C51 " " " " "	8-0 (LSB)	34
C61 " " " " "	8-1	"
C71 " " " " "	8-2	"
C81 " " " " "	8-3	"
C91 " " " " "	8-4	"
C101 " " " " "	8-5	"
C111 " " " " "	8-6 (MSB)	"

Discrete Parameters

Description	Analogic Slot-Bit	Address (octal)
C11 Flow (Surge) Alarm Lamp	11-0 (LSB)	50
C12 " " " "	11-1	"
C13 " " " "	11-2	"
C14 " " " "	11-3	"
C21 " " " "	11-4	"
C22 " " " "	11-5	"
C31 " " " "	11-6	"
C41 " " " "	11-7	"
C51 " " " "	11-8	"
C61 " " " "	11-9	"
C71 " " " "	11-10	"
C81 " " " "	11-11	"
C91 " " " "	11-12	"
C101 " " " "	11-13	"
C111 " " " "	(MSB) 11-14	"
C11 Controller Override	(LSB) 12-32	56
C12 " "	12-31	"
C13 " "	12-30	"
C14 " "	12-29	"
C21 " "	12-28	"
C22 " "	12-27	"
C31 " "	12-26	"
C41 " "	12-25	"
C51 " "	12-24	"
C61 " "	12-23	"
C71 " "	12-22	"
C81 " "	12-21	"
C91 " "	12-20	"
C10 " "	12-19	"
C111 " "	(MSB) 12-18	"
C11 Computer Override	(LSB) 12-16	54
C12 " "	12-15	"
C13 " "	12-14	"
C14 " "	12-13	"
C21 " "	12-12	"
C22 " "	12-11	"
C31 " "	12-10	"
C41 " "	12-9	"
C51 " "	12-8	54
C61 " "	12-7	"
C71 " "	12-6	"
C81 " "	12-5	"
C91 " "	12-4	"
C101 " "	12-3	"
C111 " "	(MSB) 12-2	"

APPENDIX C

DATA DICTIONARY FOR THE SOFTWARE SPECIFICATION DATA FLOW DIAGRAM

Data Dictionary for the Software Specification
Data Flow Diagram

ALARM DATA Three alarm words indicating each compressor's status relative to "recycle trip", "safety on", or temperature alarm.

ALARM LIMITS Table of temperature differentials at which the compressors are considered to be in surge.

ALARMTIME The time in tenths of seconds a first or second stage compressor can be in temperature alarm before its discharge valve is closed.

CCCDATA Data obtained from the CCC controller consisting of Controller analog inputs (% of full scale), controller output (% of full scale), controller status information, and configuration constants(CFGNCON).

CCCFUNCTION Function code or command to the CCC controller as described in Controller Communications Specification (Reference 5).

CFGNCON Data obtained from the CCC Controllers representing the configuration constants stored in the controllers.

CFGNFCN Function code from the Configuration System (V-PPC3).

CFGNMSG Packets of alarm, status, printout, and stroketime data from this system to the Configuration System (V-PPC3).

INITIAL DATA Aggregate data loaded from disk to Syscom prior to running the system. It consists of SCANTABLE, SCALEFACTORS, ZEROES, TYPE CODES, ALARM LIMITS, ALARMTIME, STROKETABLE, TOLERANCE, and INITOUT.

INITOUT The default conditions for valve positions, Moog servo power, and alarm lamps.

INVALID DATA Status words indicating invalid data.

I/O COMMAND The sequence of register loading, mode selection and initiation of transfer to the Analogic subsystem.

LIST The aggregate data base comprising parameter names, and other parameter descriptive information including software and hardware channel numbers, alarm limits, type codes, validity limits, engineering units conversion constants, and ranges.

LOGDATA Data logged to a circular buffer capable of storing 2 hours of RAWDAT, ALARMDATA, and CCCDATA.

MODE CONTROL Status words in SYSCOM or System Common Flags whereby the mode of individual modules can be controlled by other modules.

NETREQ The request code from the PMC signifying what data is required. Requests are for combinations of data from PARAM.

OUTDATA Output data buffer consisting of valve position setpoints and discrete outputs for Moog servo power control and control console alarm lamps.

PARAM The complement of data and status information required by the V-Plant PMC. It consists of RAWDAT, CCCDATA, ALARM DATA, STROKETIMES, INVALID DATA, ZEROES, LIST, SCALE FACTORS, TOLERANCE, and STROKETABLE depending upon NETREQ.

RAWDAT Raw temperature data as acquired by the I/O subsystem.

REQFCN The function code word generated by the configuration computer handler module and used by the controller handler module. Functions are the remote programming of the CCC controllers. This word is used to generate CCCFUNCTION.

SCALE FACTORS A table with the conversion scale factors for raw counts to millivolts for thermocouple transmitter inputs.

SCANTABLE The table of addresses and gains used by the ANALOGIC in acquiring data.

STATUS Analogic status word (read from its CSR).

STROKETABLE A table of expected stroke times for the various surge control valves. This is determined by the machine as a diagnostic function.

STROKETIMES A table containing actual stroke times (both open and close) for the current run.

SYSCOM System common area of memory used for dynamic status and data.

TOLERANCE A table containing acceptable tolerance levels for variation of STROKETIMES from STROKETABLE.

TYPE CODES A table containing the type code for each of the analog temperature inputs. There are three type codes, Type T-400 Degrees F, Type T-600 Degrees F, and Type J-400 Degrees F.

VALID LIMITS A table of the validity limits (both high and low) for the temperature data.

ZEROES A table containing the zero offset counts for thermocouple input channels.

APPENDIX D

CONTROLLER CONSTANTS FILE

CCC DATA FILE

ACTUAL VALUES

COMP	B2	RT	SO	PV	SP	FA	PB	Kr	Tf	K	B1	C1	C2	Mode	Stat
11	0.1	1.1	1.0	5.4	3.6	8.	100.	5.0	3.0	.240	1.0	10.0	1.00	2.	0.
12	0.0	1.1	1.0	0.0	1.0	8.	100.	5.0	2.0	.240	1.0	10.0	1.00	2.	0.
13	0.0	1.1	1.0	0.7	1.0	8.	100.	9.0	3.0	.200	1.0	50.0	1.00	2.	0.
14	0.0	1.1	2.0	0.0	1.0	8.	100.	5.0	2.0	.220	1.0	10.0	1.00	2.	0.
21	2.0	4.1	1.0	0.0	4.0	8.	100.	5.0	3.0	.350	4.0	10.0	1.00	2.	0.
22	1.0	4.1	1.0	0.4	4.0	8.	100.	5.0	3.0	.200	4.0	10.0	1.00	2.	0.
31	2.5	15.0	5.0	0.0	15.0	8.	150.	15.0	2.0	.240	15.0	20.0	0.52	2.	0.
41	0.0	0.0	0.0	0.0	0.0	0.	0.	0.0	0.0	.000	0.0	0.0	0.00	3.	0.
51	2.5	9.9	5.0	0.3	7.0	8.	200.	10.0	0.5	.252	7.0	20.0	0.52	2.	0.
61	2.5	11.0	5.0	0.0	5.0	8.	250.	15.0	1.0	.120	5.0	20.0	0.52	2.	0.
71	2.5	11.0	5.0	1.0	10.4	8.	250.	15.0	0.2	.352	10.0	20.0	0.52	2.	0.
81	2.5	11.0	5.0	2.2	9.1	8.	250.	15.0	0.5	.472	8.0	20.0	0.52	0.	0.
91	2.5	11.0	5.0	0.7	10.0	8.	250.	15.0	0.7	.055	10.0	20.0	0.52	2.	0.
101	2.5	5.0	2.0	0.5	5.5	1.	100.	10.0	0.0	.499	5.0	20.0	0.12	3.	0.
111	0.0	0.0	0.0	0.0	0.0	0.	0.	0.0	0.0	.000	0.0	0.0	0.00	3.	0.

LOW RANGE VALUES

COMP	B2	RT	SO	K	B1	C1	C2
11	0.5	1.1	1.0	.240	1.0	20.0	0.52
12	0.5	1.1	1.0	.240	1.0	20.0	0.52
13	0.5	1.1	1.0	.200	1.0	20.0	0.52
14	0.5	1.1	1.0	.220	1.0	20.0	0.48
21	0.5	4.1	1.0	.250	4.0	20.0	0.52
22	0.5	4.1	1.0	.200	4.0	20.0	0.52
31	2.5	0.8	2.0	.240	5.0	50.0	5.00
41	2.5	0.8	2.0	.420	11.0	50.0	5.00
51	2.5	0.8	2.0	.252	5.0	50.0	5.00
61	1.0	0.8	2.0	.120	5.0	50.0	5.00
71	2.5	0.8	2.0	.352	5.0	50.0	5.00
81	2.5	0.8	2.0	.472	5.0	50.0	5.00
91	2.5	0.8	2.0	.055	5.0	50.0	5.00
101	1.0	0.8	2.0	.500	50.0	50.0	5.00
111	57.8	64.0	97.4	.000	0.0	20.5	2.05

HIGH RANGE VALUES

COMP	B2	RT	SO	K	B1	C1	C2
11	0.5	1.9	1.0	.240	2.0	20.0	0.52
12	0.5	1.9	1.0	.240	2.0	20.0	0.52
13	0.5	1.9	1.0	.200	2.0	20.0	0.52
14	0.5	1.9	1.0	.220	2.0	20.0	0.52
21	0.5	5.5	1.0	.250	5.5	20.0	0.52
22	0.5	5.5	1.0	.200	6.0	20.0	0.52
31	1.0	0.8	2.0	.500	50.0	50.0	5.00
41	1.0	0.8	2.0	.500	50.0	50.0	5.00
51	1.0	0.8	2.0	.500	50.0	50.0	5.00
61	1.0	0.8	2.0	.500	50.0	50.0	5.00
71	1.0	0.8	2.0	.500	50.0	50.0	5.00
81	1.0	0.8	2.0	.500	50.0	50.0	5.00
91	1.0	0.8	2.0	.500	50.0	50.0	5.00
101	1.0	0.8	2.0	.500	50.0	50.0	5.00
101	97.4	0.0	0.0	.000	1.0	0.0	0.00

APPENDIX E

DATA BASE FILE

DATA BASE FILE

PARM NUM	CONV TYPE	PARAMETER ID	PARAMETER DESCRIPTION	CH AS	SL OT	CH AN	D F	PR GAIN	TM AM	A P	N 1
1	3	TT117	C11 Inlet Temp Tx	0	9	0	T	8	3	T	F
2	3	TT127	C12 Inlet Temp Tx	0	9	2	T	8	3	T	F
3	3	TT137	C13 Inlet Temp Tx	0	9	4	T	8	3	T	F
4	3	TT147	C14 Inlet Temp Tx	0	9	6	T	8	3	T	F
5	3	TT217	C21 Inlet Temp Tx	0	9	8	T	8	0	T	F
6	3	TT227	C22 Inlet Temp Tx	0	9	10	T	8	0	T	F
7	3	TT317	C31 Inlet Temp Tx	0	9	12	T	8	0	T	F
8	3	TT417	C41 Inlet Temp Tx	0	9	14	T	8	0	T	F
9	3	TT517	C51 Inlet Temp Tx	0	10	0	T	8	0	T	F
10	3	TT617	C61 Inlet Temp Tx	0	10	2	T	8	0	T	F
11	3	TT717	C71 Inlet Temp Tx	0	10	4	T	8	0	T	F
12	3	TT817	C81 Inlet Temp Tx	0	10	6	T	8	0	T	F
13	3	TT917	C91 Inlet Temp Tx	0	10	8	T	8	0	T	F
14	3	TT1017	C101 Inlet Temp Tx	0	10	10	T	8	0	T	F
15	3	TT1117	C111 Inlet Temp Tx	0	10	12	T	8	0	T	F
16	16	TT118	C11 Internal Temp Tx	0	9	1	T	8	3	T	F
17	16	TT128	C12 Internal Temp Tx	0	9	3	T	8	3	T	F
18	16	TT138	C13 Internal Temp Tx	0	9	5	T	8	3	T	F
19	16	TT148	C14 Internal Temp Tx	0	9	7	T	8	3	T	F
20	16	TT218	C21 Internal Temp Tx	0	9	9	T	8	0	T	F
21	16	TT228	C22 Internal Temp Tx	0	9	11	T	8	0	T	F
22	3	TT318	C31 Internal Temp Tx	0	9	13	T	8	0	T	F
23	3	TT418	C41 Internal Temp Tx	0	9	15	T	8	0	T	F
24	3	TT518	C51 Internal Temp Tx	0	10	1	T	8	0	T	F
25	3	TT618	C61 Internal Temp Tx	0	10	3	T	8	0	T	F
26	3	TT718	C71 Discharge Temp Tx	0	10	5	T	8	0	T	F
27	3	TT818	C81 Discharge Temp Tx	0	10	7	T	8	0	T	F
28	3	TT918	C91 Discharge Temp Tx	0	10	9	T	8	0	T	F
29	3	TT1018	C101 Discharge Temp Tx	0	10	11	T	8	0	T	F
30	3	TT1118	C111 Discharge Temp Tx	0	10	13	T	8	0	T	F
31	22	ZIV113	Valve 113 Position Indication	0	13	1	T	8	0	F	F
32	22	ZIV123	Valve 123 Position Indication	0	13	3	T	8	0	F	F
33	22	ZIV133	Valve 133 Position Indication	0	13	5	T	8	0	F	F
34	22	ZIV143	Valve 143 Position Indication	0	13	7	T	8	0	F	F
35	22	ZIV213	Valve 213 Position Indication	0	13	9	T	8	0	F	F
36	22	ZIV223	Valve 223 Position Indication	0	13	11	T	8	0	F	F
37	22	ZIV313	Valve 313 Position Indication	0	13	12	T	8	0	F	F
38	22	ZIV413	Valve 413 Position Indication	0	13	13	T	8	0	F	F
39	22	ZIV513	Valve 513 Position Indication	0	13	14	T	8	0	F	F
40	22	ZIV613	Valve 613 Position Indication	0	13	15	T	8	0	F	F
41	22	ZIV713	Valve 713 Position Indication	0	14	0	T	8	0	F	F
42	22	ZIV813	Valve 813 Position Indication	0	14	1	T	8	0	F	F
43	22	ZIV913	Valve 913 Position Indication	0	14	2	T	8	0	F	F
44	22	ZIV1013	Valve 1013 Position Indication	0	14	3	T	8	0	F	F
45	22	ZIV1113	Valve 1113 Position Indication	0	14	4	T	8	0	F	F
46	22	ZIV115	Valve 115 Position Indication	0	13	0	T	8	0	F	F
47	22	ZIV125	Valve 125 Position Indication	0	13	2	T	8	0	F	F
48	22	ZIV135	Valve 135 Position Indication	0	13	4	T	8	0	F	F
49	22	ZIV145	Valve 145 Position Indication	0	13	6	T	8	0	F	F
50	22	ZIV215	Valve 215 Position Indication	0	13	8	T	8	0	F	F
51	22	ZIV225	Valve 225 Position Indication	0	13	10	T	8	0	F	F

DATA BASE FILE

PARM NUM	CONV TYPE	PARAMETER ID	PARAMETER DESCRIPTION	CH AS	SL OT	CH AN	D F	PR GAIN	TM AM	A P	N 1
52	51	JIV113	Valve 113 Servo Power Indication	0	15	1	T	1	0	F	F
53	51	JIV123	Valve 123 Servo Power Indication	0	15	3	T	1	0	F	F
54	51	JIV133	Valve 133 Servo Power Indication	0	15	5	T	1	0	F	F
55	51	JIV143	Valve 143 Servo Power Indication	0	15	7	T	1	0	F	F
56	51	JIV213	Valve 213 Servo Power Indication	0	15	9	T	1	0	F	F
57	51	JIV223	Valve 223 Servo Power Indication	0	15	11	T	1	0	F	F
58	51	JIV313	Valve 313 Servo Power Indication	0	15	12	T	1	0	F	F
59	51	JIV413	Valve 413 Servo Power Indication	0	15	13	T	1	0	F	F
60	51	JIV513	Valve 513 Servo Power Indication	0	15	14	T	1	0	F	F
61	51	JIV613	Valve 613 Servo Power Indication	0	15	15	T	1	0	F	F
62	51	JIV713	Valve 713 Servo Power Indication	0	16	0	T	1	0	F	F
63	51	JIV813	Valve 813 Servo Power Indication	0	16	1	T	1	0	F	F
64	51	JIV913	Valve 913 Servo Power Indication	0	16	2	T	1	0	F	F
65	51	JIV1013	Valve 1013 Servo Pwr. Indication	0	16	3	T	1	0	F	F
66	51	JIV1113	Valve 1113 Servo Pwr. Indication	0	16	4	T	1	0	F	F
67	51	JIV115	Valve 115 Servo Power Indication	0	15	0	T	1	0	F	F
68	51	JIV125	Valve 125 Servo Power Indication	0	15	2	T	1	0	F	F
69	51	JIV135	Valve 135 Servo Power Indication	0	15	4	T	1	0	F	F
70	51	JIV145	Valve 145 Servo Power Indication	0	15	6	T	1	0	F	F
71	51	JIV215	Valve 215 Servo Power Indication	0	15	8	T	1	0	F	F
72	51	JIV225	Valve 225 Servo Power Indication	0	15	10	T	1	0	F	F
73	201	OIY11	C11 Computer Override Ind.	0	12	0	F	0	0	F	F
74	201	OIY12	C12 Computer Override Ind.	0	12	1	F	0	0	F	F
75	201	OIY13	C13 Computer Override Ind.	0	12	2	F	0	0	F	F
76	201	OIY14	C14 Computer Override Ind.	0	12	3	F	0	0	F	F
77	201	OIY21	C21 Computer Override Ind.	0	12	4	F	0	0	F	F
78	201	OIY22	C22 Computer Override Ind.	0	12	5	F	0	0	F	F
79	201	OIY31	C31 Computer Override Ind.	0	12	6	F	0	0	F	F
80	201	OIY41	C41 Computer Override Ind.	0	12	7	F	0	0	F	F
81	201	OIY51	C51 Computer Override Ind.	0	12	8	F	0	0	F	F
82	201	OIY61	C61 Computer Override Ind.	0	12	9	F	0	0	F	F
83	201	OIY71	C71 Computer Override Ind.	0	12	10	F	0	0	F	F
84	201	OIY81	C81 Computer Override Ind.	0	12	11	F	0	0	F	F
85	201	OIY91	C91 Computer Override Ind.	0	12	12	F	0	0	F	F
86	201	OIY101	C101 Computer Override Ind.	0	12	13	F	0	0	F	F
87	201	OIY111	C111 Computer Override Ind.	0	12	14	F	0	0	F	F
88	202	OIC11	C11 Controller Override Ind.	0	12	16	F	0	0	F	F
89	202	OIC12	C12 Controller Override Ind.	0	12	17	F	0	0	F	F
90	202	OIC13	C13 Controller Override Ind.	0	12	18	F	0	0	F	F
91	202	OIC14	C14 Controller Override Ind.	0	12	19	F	0	0	F	F
92	202	OIC21	C21 Controller Override Ind.	0	12	20	F	0	0	F	F
93	202	OIC22	C22 Controller Override Ind.	0	12	21	F	0	0	F	F
94	202	OIC31	C31 Controller Override Ind.	0	12	22	F	0	0	F	F
95	202	OIC41	C41 Controller Override Ind.	0	12	23	F	0	0	F	F
96	202	OIC51	C51 Controller Override Ind.	0	12	24	F	0	0	F	F
97	202	OIC61	C61 Controller Override Ind.	0	12	25	F	0	0	F	F
98	202	OIC71	C71 Controller Override Ind.	0	12	26	F	0	0	F	F
99	202	OIC81	C81 Controller Override Ind.	0	12	27	F	0	0	F	F
100	202	OIC91	C91 Controller Override Ind.	0	12	28	F	0	0	F	F
101	202	OIC101	C101 Controller Override Ind.	0	12	29	F	0	0	F	F
102	202	OIC111	C111 Controller Override Ind.	0	12	30	F	0	0	F	F

DATA BASE FILE

PARM NUM	CONV TYPE	PARAMETER ID	PARAMETER DESCRIPTION	CH AS	SL OT	CH AN	D F	PR GAIN	TM AM	A P	N 1
103	101	ZVC-113	Valve 113 Position Setpoint	0	1	1	F	0	0	F	F
104	101	ZVC-123	Valve 123 Position Setpoint	0	1	3	F	0	0	F	F
105	101	ZVC-133	Valve 133 Position Setpoint	0	2	1	F	0	0	F	F
106	101	ZVC-143	Valve 143 Position Setpoint	0	2	3	F	0	0	F	F
107	101	ZVC-213	Valve 213 Position Setpoint	0	3	1	F	0	0	F	F
108	101	ZVC-223	Valve 223 Position Setpoint	0	3	3	F	0	0	F	F
109	101	ZVC-313	Valve 313 Position Setpoint	0	4	0	F	0	0	F	F
110	101	ZVC-413	Valve 413 Position Setpoint	0	4	1	F	0	0	F	F
111	101	ZVC-513	Valve 513 Position Setpoint	0	4	2	F	0	0	F	F
112	101	ZVC-613	Valve 613 Position Setpoint	0	4	3	F	0	0	F	F
113	101	ZVC-713	Valve 713 Position Setpoint	0	5	0	F	0	0	F	F
114	101	ZVC-813	Valve 813 Position Setpoint	0	5	1	F	0	0	F	F
115	101	ZVC-913	Valve 913 Position Setpoint	0	5	2	F	0	0	F	F
116	101	ZVC-1013	Valve 1013 Position Setpoint	0	5	3	F	0	0	F	F
117	101	ZVC-1113	Valve 1113 Position Setpoint	0	6	0	F	0	0	F	F
118	101	ZVC-115	Valve 115 Position Setpoint	0	1	0	F	0	0	F	F
119	101	ZVC-125	Valve 125 Position Setpoint	0	1	2	F	0	0	F	F
120	101	ZVC-135	Valve 135 Position Setpoint	0	2	0	F	0	0	F	F
121	101	ZVC-145	Valve 145 Position Setpoint	0	2	2	F	0	0	F	F
122	101	ZVC-215	Valve 215 Position Setpoint	0	3	0	F	0	0	F	F
123	101	ZVC-225	Valve 225 Position Setpoint	0	3	2	F	0	0	F	F
124	301	FA11	C11 Surge (Flow) Alarm	0	11	0	F	0	0	F	F
125	301	FA12	C12 Surge (Flow) Alarm	0	11	1	F	0	0	F	F
126	301	FA13	C13 Surge (Flow) Alarm	0	11	2	F	0	0	F	F
127	301	FA14	C14 Surge (Flow) Alarm	0	11	3	F	0	0	F	F
128	301	FA21	C21 Surge (Flow) Alarm	0	11	4	F	0	0	F	F
129	301	FA22	C22 Surge (Flow) Alarm	0	11	5	F	0	0	F	F
130	301	FA31	C31 Surge (Flow) Alarm	0	11	6	F	0	0	F	F
131	301	FA41	C41 Surge (Flow) Alarm	0	11	7	F	0	0	F	F
132	301	FA51	C51 Surge (Flow) Alarm	0	11	8	F	0	0	F	F
133	301	FA61	C61 Surge (Flow) Alarm	0	11	9	F	0	0	F	F
134	301	FA71	C71 Surge (Flow) Alarm	0	11	10	F	0	0	F	F
135	301	FA81	C81 Surge (Flow) Alarm	0	11	11	F	0	0	F	F
136	301	FA91	C91 Surge (Flow) Alarm	0	11	12	F	0	0	F	F
137	301	FA101	C101 Surge (Flow) Alarm	0	11	13	F	0	0	F	F
138	301	FA111	C111 Surge (Flow) Alarm	0	11	14	F	0	0	F	F
139	302	JCV113	Servo Amp Power Valve 113	0	7	0	F	0	0	F	F
140	302	JCV123	Servo Amp Power Valve 123	0	7	1	F	0	0	F	F
141	302	JCV133	Servo Amp Power Valve 133	0	7	2	F	0	0	F	F
142	302	JCV143	Servo Amp Power Valve 143	0	7	3	F	0	0	F	F
143	302	JCV213	Servo Amp Power Valve 213	0	7	4	F	0	0	F	F
144	302	JCV223	Servo Amp Power Valve 223	0	7	5	F	0	0	F	F
145	302	JCV313	Servo Amp Power Valve 313	0	7	6	F	0	0	F	F
146	302	JCV413	Servo Amp Power Valve 413	0	7	7	F	0	0	F	F
147	302	JCV513	Servo Amp Power Valve 513	0	8	0	F	0	0	F	F
148	302	JCV613	Servo Amp Power Valve 613	0	8	1	F	0	0	F	F
149	302	JCV713	Servo Amp Power Valve 713	0	8	2	F	0	0	F	F
150	302	JCV813	Servo Amp Power Valve 813	0	8	3	F	0	0	F	F
151	302	JCV913	Servo Amp Power Valve 913	0	8	4	F	0	0	F	F
152	302	JCV1013	Servo Amp Power Valve 1013	0	8	5	F	0	0	F	F
153	302	JCV1113	Servo Amp Power Valve 1113	0	8	6	F	0	0	F	F

DATA BASE FILE

PARM NUM	MIN RANGE	MAX RANGE	ZERO	FULL COUNT	SCALE FACTR	ALM ABL	WARNING LEVEL	DANGER LEVEL	FILTR
1	-674.00	9523.00	819	4095	3.11264	T	0.00	250.00	9
2	-674.00	9523.00	819	4095	3.11264	T	0.00	250.00	9
3	-674.00	9523.00	819	4095	3.11264	T	0.00	250.00	9
4	-674.00	9523.00	819	4095	3.11264	T	0.00	250.00	9
5	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
6	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
7	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
8	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
9	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
10	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
11	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
12	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
13	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
14	-674.00	9523.00	819	4095	3.11264	T	0.00	200.00	9
15	-674.00	9523.00	819	4095	3.11264	T	0.00	300.00	9
16	-885.00	17186.00	819	4095	5.51618	T	0.00	325.00	9
17	-885.00	17186.00	819	4095	5.51618	T	0.00	325.00	9
18	-885.00	17186.00	819	4095	5.51618	T	0.00	325.00	9
19	-885.00	17186.00	819	4095	5.51618	T	0.00	325.00	9
20	-885.00	17186.00	819	4095	5.51618	T	0.00	325.00	9
21	-885.00	17186.00	819	4095	5.51618	T	0.00	325.00	9
22	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
23	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
24	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
25	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
26	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
27	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
28	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
29	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
30	-674.00	15769.00	819	4095	5.01923	T	0.00	325.00	9
31	0.00	100.00	0	3490	0.02865	F	0.00	0.00	9
32	0.00	100.00	0	3467	0.02884	F	0.00	0.00	9
33	0.00	100.00	5	3462	0.02893	F	0.00	0.00	9
34	0.00	100.00	0	3465	0.02886	F	0.00	0.00	9
35	0.00	100.00	1	3483	0.02872	F	0.00	0.00	9
36	0.00	100.00	0	3178	0.03147	F	0.00	0.00	9
37	0.00	100.00	2	3459	0.02893	F	0.00	0.00	9
38	0.00	100.00	0	3466	0.02885	F	0.00	0.00	9
39	0.00	100.00	0	3472	0.02880	F	0.00	0.00	9
40	0.00	100.00	0	3385	0.02954	F	0.00	0.00	9
41	0.00	100.00	0	3479	0.02874	F	0.00	0.00	9
42	0.00	100.00	24	3475	0.02898	F	0.00	0.00	9
43	0.00	100.00	7	3266	0.03068	F	0.00	0.00	9
44	0.00	100.00	0	3282	0.03047	F	0.00	0.00	9
45	0.00	100.00	1	3252	0.03076	F	0.00	0.00	9
46	0.00	100.00	67	3230	0.03162	F	0.00	0.00	9
47	0.00	100.00	143	2791	0.03776	F	0.00	0.00	9
48	0.00	100.00	47	3194	0.03178	F	0.00	0.00	9
49	0.00	100.00	2	3251	0.03078	F	0.00	0.00	9
50	0.00	100.00	27	3256	0.03097	F	0.00	0.00	9
51	0.00	100.00	12	3303	0.03039	F	0.00	0.00	9

DATA BASE FILE

PARM NUM	MIN RANGE	MAX RANGE	ZERO	FULL COUNT	SCALE FACTR	ALM ABL	WARNING LEVEL	DANGER LEVEL	FILTR
52	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
53	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
54	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
55	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
56	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
57	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
58	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
59	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
60	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
61	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
62	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
63	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
64	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
65	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
66	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
67	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
68	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
69	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
70	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
71	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
72	0.00	10.00	0	4095	0.00244	F	0.00	0.00	9
73	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
74	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
75	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
76	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
77	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
78	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
79	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
80	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
81	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
82	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
83	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
84	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
85	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
86	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
87	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
88	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
89	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
90	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
91	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
92	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
93	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
94	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
95	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
96	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
97	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
98	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
99	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
100	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
101	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
102	0.00	0.00	0	0	0.00000	F	0.00	0.00	0

DATA BASE FILE

PARM NUM	MIN RANGE	MAX RANGE	ZERO	FULL COUNT	SCALE FACTR	ALM ABL	WARNING LEVEL	DANGER LEVEL	FILTR
103	0.00	100.00	3304	300	-0.03329	F	0.00	0.00	0
104	0.00	100.00	3489	269	-0.03106	F	0.00	0.00	0
105	0.00	100.00	3994	288	-0.02698	F	0.00	0.00	0
106	0.00	100.00	3302	128	-0.03151	F	0.00	0.00	0
107	0.00	100.00	3727	321	-0.02936	F	0.00	0.00	0
108	0.00	100.00	2992	183	-0.03560	F	0.00	0.00	0
109	0.00	100.00	3670	-10	-0.02717	F	0.00	0.00	0
110	0.00	100.00	3608	397	-0.03114	F	0.00	0.00	0
111	0.00	100.00	3977	97	-0.02577	F	0.00	0.00	0
112	0.00	100.00	3742	317	-0.02920	F	0.00	0.00	0
113	0.00	100.00	3668	672	-0.03338	F	0.00	0.00	0
114	0.00	100.00	4003	332	-0.02724	F	0.00	0.00	0
115	0.00	100.00	3740	1656	-0.04798	F	0.00	0.00	0
116	0.00	100.00	3532	576	-0.03383	F	0.00	0.00	0
117	0.00	100.00	3686	369	-0.03015	F	0.00	0.00	0
118	0.00	100.00	3712	262	-0.02899	F	0.00	0.00	0
119	0.00	100.00	3751	479	-0.03056	F	0.00	0.00	0
120	0.00	100.00	3798	712	-0.03240	F	0.00	0.00	0
121	0.00	100.00	3853	24	-0.02612	F	0.00	0.00	0
122	0.00	100.00	3784	843	-0.03400	F	0.00	0.00	0
123	0.00	100.00	3278	278	-0.03333	F	0.00	0.00	0
124	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
125	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
126	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
127	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
128	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
129	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
130	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
131	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
132	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
133	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
134	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
135	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
136	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
137	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
138	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
139	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
140	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
141	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
142	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
143	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
144	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
145	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
146	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
147	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
148	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
149	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
150	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
151	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
152	0.00	0.00	0	0	0.00000	F	0.00	0.00	0
153	0.00	0.00	0	0	0.00000	F	0.00	0.00	0

VITA

Brenda Carol Howard Barr was born in Grosse Pointe, Michigan on June 3, 1955. She moved to Tennessee with her parents in April, 1957 and graduated in June, 1973 from Putnam County Senior High School. She attended Tennessee Technological University to receive a Bachelor of Science degree in Computer Science in March, 1983. For the past ten years she has been employed by Calspan Corporation at Arnold Engineering Development Center (AEDC) as a Computer Scientist, developing and maintaining real-time control systems. AEDC is located on Arnold Air Force Base, in middle Tennessee.