Attachment 3

Model 956A-201 Failure History Summary

SYNCOR
RADIATION
MANAGEMENT
A Syncor Business

November 4, 2002

Rochester Gas and Electric Corporation
Ginna Station
1503 Lake Rd.
Ontario, N.Y. 14519

6045 Cochran Road
Cleveland
Ohio 44139 USA

phone 440 248.9300
fax 440 349.2307
srm.syncor.com

Attention: Mr. Paul Swift, Project Engineer

Reference: Rochester Gas and Electric Corp. P.O. No.: 4500008671
Ginna Station,
Syncor Radiation Management Sales Order No. 157033

Subject: Model 956A-201 Failure History Summary

Dear Paul:

As requested during our recent NRC visit, the following is a summary of the Model 956A-201 failure history:

Model 956A-201 Failure History:

The overall reliability of the UDR family is supported by our repair history data base, which has been in operation since 1988. On a total return basis, which includes returns for periodic re-calibrations for our 946 and 956 area monitor units, the UDR product line has exhibited an overall return rate of approximately 6% on a sales base of approximately 2,000 units. The vast majority of the units returned for repair are due to a failed dc power supply. The total 956A installed base also exhibits a total return rate of approximately 6%. When the returns are adjusted for re-calibrations, a much lower return percentage would be realized. The 956A-201 model supplied to RG&E, which was first sold in 1995, currently exhibits a gross return rate of approximately 3%. This reduced return rate may be attributed to the power supplies not yet reaching maturity. The 956A returns are distributed as follows:

The Model 956 Digital Ratemeter (UDR) is one of three microprocessor based digital ratemeters designed by Victoreen (now Syncor) in 1985 for the nuclear industry. The Model 956 is used with a Geiger-Mueller tube detector for Area Monitoring. The Model 942 is used with scintillation detectors for process monitoring, and the Model 946 is used with an eight decade ion chamber detector, also for Area Monitoring. Each of these UDR's share a common mother board that contains the microprocessor (and 8-bit Motorola 6802 and associated electronics) and differ in the specific functions configured into the operating firmware. The active electronic components are high reliability, 54LS logic Mil-Spec components. Over the installed life of the unit, the 54LS logic has proved to be a reliable device, with a low failure rate.

SYNCOR
RADIATION
MANAGEMENT
A Syncor Source

6045 Cochran Road
Cleveland
Ohio 44139 USA

phone 440.248.9300
fax 440.349.2307
srm syncor.com

956A-200 Total Returns:

Re-calibrations:	15	
Repair:	1	Corrosion damage
	1	Broken AC power switch
Other	2	Sales Demo

956A-201 Total Returns:

Re-calibrations:	1	
Repair: 1		Random failure, D/A converter replaced
Other:	1	Shipping damage
	1	Customer loan

Syncor/Victoreen has always been dedicated to the supply of the highest quality products to our Nuclear Power and Medical customers, and to continually improving the quality of our products. In May, 1998, we expanded our existing 10CFR21 reporting system and implemented a formal, database based, customer complaint tracking system, our QSP-14-01, Complaint Handling. This procedure tracks customer complaints, and requires management review and monitoring. Although customers are not required to input into the system, our internal engineering and customer service personnel are charged with the responsibility of entering customer complaints into the system. To date, there have been a total of 3 complaints logged to the entire Model 9XX UDR family.

The first two complaints, issued in Feb, 2000 and April, 2002, involved units that were modified for a specific customer. Both of these complaints have been satisfactorily closed. The third complaint was also issued in 2002, and involved testing documentation. A summary of the complaint, and action taken follows:

Feb, 2000: On a 946B Ion Chamber readout modified for use with a roll up ribbon cable rear panel connector interface, which is not the configuration or interface supplied for RG&E, the display value was found by the customer to sporadically change. When tested by Syncor, anomaly was not reproducible, and is not critical for the alarming function of the unit. After considerable in-house testing, a revised PROM was supplied from Syncor to the client. The customer was satisfied with the PROM provided, and the complaint was closed, with the understanding that a new complaint would be opened if the anomaly re-occurred.

Apr, 2002: On a 946A Ion Chamber readout used in a simulator, with an analog input used to simulate the radiation input signal from the detector, the customer advised it was possible to set the Warn alarm higher than the High alarm. This was contrary to the operation of the actual unit, where the firmware does not accept a Warn alarm set point higher than the High alarm set point. All other customers using the same PROM were then notified of the problem, and were offered an upgrade at a nominal charge. Since the ability of the unit to alarm was not compromised, there was no safety concern. None of the user's elected to pursue the firmware change, and the problem was closed.

Syncor Radiation Management
A Syncor Business

Oct, 2002: It was identified that EMI testing performed did not encompass the specific levels of ERRI TR-102323-R1. The customer was advised, and a program to re-test the equipment to the specific limits of the EPRI specification was initiated.

Concerning 10CFR21 Notices, there has been only one 10CFR21 Notification issued. It was issued, 1999, and dealt with a BNC connector provided by as an equivalent to an another connector, and was found, by Syncor, to not have the same cable retention properties as the original. All of the incorrect connectors were subsequently replaced.

6045 Cochran Road
Cleveland
Ohio 44139 USA

phone 440.248 9300
fax 440.349 2307
srm.syncor.com

We believe the clean operating history of the unit, the environmental and seismic qualification tests performed, and the V&V program performed for RG&E, will ensure reliable operation of the Model 956A-201 UDR for their safety related application.

Sincerely Yours,

Andrew Lasko
Project Manager
Phone: 440-542-3611
Fax: 440-542-3661
E-Mail: alasko@inovision.com

SOFTWARE VERIFICATION AND VALIDATION PLAN
FOR
PROM P/N 94095603
GM- AREA MONITOR

REV. LEVEL	ECN #	DESCRIPTION/PAGES AFFECTED

ENGINEERING	[signature]	DATE 9-20-02
MANUFACTURING	[signature]	DATE 9/20/02
QUALITY ASSURANCE	[signature]	DATE 9/20/02

SYNCOR RADIATION MANAGEMENT			DATE 9/11/02	TITLE SOFTWARE VERIFICATION AND VALIDATION PLAN, 94095603	
REV 1	ECN NO. 3044	RELEASED FOR PRODUCTION	DOC CTRL	SHEET 1 of 27	NO. 94095603VVP	SIZE

Table of Contents

PURPOSE

The purpose of this Verification and Validation plan is to develop a series of activities, and their associated inputs and outputs, that will demonstrate that the firmware in the P/N 94095603 EPROM, used in the Victoreen Model 956A Digital G-M Area Monitor Readout, manufactured by Syncor Radiation Measurements meets the monitor's design requirements and exhibits a high degree of reliability.

Note that although the base firmware was developed in the early 1980s, prior to the availability of the current industry software development standards, this V&V plan is intended to demonstrate that the existing firmware is suitable for use in safety related applications.

1 REFERENCE DOCUMENTS

The reference standards used for guiding the preparation of this document and for SV&V implementation are listed below:

1.1 IEEE Std 7-4.3.2-1993, Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations

1.2 IEEE Std 610.12-1990, Glossary of Software Engineering Terminology

1.3 IEEE Std 729-1983, Standard Glossary of Software Engineering Terminology

1.4 IEEE Std 829-1991, Standards for Software Test Documentation

1.5 IEEE Std 830-1993, Recommended Practice for Software Requirements Specifications

1.6 IEEE Std 1012-1996, Standard for Software Verification and Validation Plans

1.7 IEEE Std 1016-1987, Recommended Practice for Software Design Descriptions

1.8 IEEE Std 1074-1991, Standard for Developing Software Life Cycle Processes

1.9 EPRI Std TR-103291-CD Handbook for Verification and Validation of Digital Systems (12/1998)

1.10 EPRI Std TR-102348, Rev. 1, Guidelines on Licensing Digital Upgrade

1.11 Syncor Radiation Management Quality Assurance Manual, QSP-100, Version 004, Rev. 1/2/02, Implemented 3/14/02

1.12 Syncor Radiation Management Quality Procedure QSP-205, Document Control

1.13 Syncor Radiation Management Quality Procedure QSP-05-05, Engineering Change Notice

2.11 10CFR50, Appendix B, Quality Assurance Criteria for Nuclear Power Plants and Fuel Reprocessing Plants, Jan. 20, 1975

2.12 ANSI/ASME NQA1-1994, Quality Assurance Program Requirements for Nuclear Facility Applications

2 DEFINITIONS

2.1 Definitions

2.1.1 Acceptance testing – Formal testing conducted to determine whether or not the system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.

2.1.2 Anomaly – Anything observed in operation of the UDR that deviates from expectations based on previously verified software/firmware products or reference documents.

2.1.3 Development team – Team of qualified engineers in charge of applying software development life cycle.

2.1.4 Developer – Member of the development team.

2.1.5 Firmware – The combination of software and data that reside in read-only memory

2.1.6 Firmware component – Assembly language module (set of functions).

2.1.7 Hardware – Physical equipment used to process, store, or transmit computer programs and data.

2.1.8 Life-cycle phase – Any period of time during software development or operation that may be characterized by a primary type of activity (such as design or testing) that is being conducted. These phases may overlap one another; for V&V purpose, no phase is concluded until its development products are fully verified.

2.1.9 Safety related firmware – Firmware for the RMS safety related equipment.

2.1.10 Software – Computer programs and data pertaining to the operation of a computer system.

2.1.11 Software/firmware testing – The process of testing an integrated hardware and software/firmware system to verify that the system meets its specified requirements.

2.1.12 Software tools – A computer program used in the development, testing, analysis, or maintenance of a program or it's documentation. Examples include CASE software, decompiler, driver, editor, flow charter, monitor, test case generator, or timing analyzer.

2.1.13 Software verification and validation plan – A plan for the conduct of software verification and validation.

2.1.14 SSC – Systems, Structure and Components

2.1.15 Test procedure – Documentation that is part of the test report, specifying a sequence of actions for the execution of a test

2.1.16 Traceability – The degree to which a relationship is established between two or more products of the development process, especially product having a predecessor-successor or master-subordinate relationship to one another; for example the degree to which the requirements and design of a given software component match.

2.1.17 Validation – The process of evaluating software/firmware at the end of the software development process to ensure compliance with software requirements.

2.1.18 Validator – Member of the SV&V team who carries out validation.

2.1.19 Verification – The process of determining whether or not the products of a given phase of the software/firmware development cycle fulfill the requirements established during the previous phase.

2.1.20 Verifier – Member of the project team who carries out verification.

2.2 Abbreviations

ANSI – American National Standards Institute

ASCII - American Standard Code for Information Interchange

DOS – Disk Operating System

ECN – Engineering Change Notice

EPROM – Erasable Programmable Read Only Memory

IEEE – Institute of Electrical and Electronics Engineers

PC – Personal Computer

QA – Quality Assurance

RMS – Radiation Monitoring System

SRM – Syncor Radiation Management

SV&V – Software Verification and Validation

UDR - Universal Digital Ratemeter

VVTP - Verification and Validation Test Plan

VVTR - Verification and Validation Test Report

V&V - Verification and Validation

2.3 Acronyms & Notations

PE - Project Engineer

PM - Project Manager

QE – Quality Engineer

QM - Quality Management

PM - Project Manager

PE - Project Engineer

RE – Reliability Engineer

SE – Software Engineer

SM - Syncor Management

SRM - Syncor Radiation Management

TT – Test Technician

2.4 Documentation Names

SRS - Software Requirements Specification

SDD - Software Design Description

SVVP – Software V&V Plan

VVTP - Verification and Validation Test Plan

VVTR - Verification and Validation Test Report

3 Verification & Validation Overview

The overall objective of the V&V Plan for the 94095603 firmware is to assure the program promotes a quality and highly reliable product through an independent process of technical review and evaluation. Note the firmware does not contain an operating system, and performs specific functions on a cyclic basis. A flow chart of the firmware operation is provided in Addendum 1.

The embedded and operating system software and associated tools are predeveloped, or commonly known as legacy software. Like other predeveloped software, it is important to examine the development history to understand how the software has matured with time into the quality product it is today. When the Prom P/N 94095603 firmware was conceived, there was very little guidance in the way of industry standards to base the software development and design on. Good programming practices were used based on the objective of producing a highly reliable safety system.

As expressed in SRP 0800, Appendix 7.0A, the use of digital I&C systems presents the concern that minor errors in design and implementation can cause them to exhibit unexpected behavior. To minimize this potential problem, the design qualification of digital systems needs to focus on a high quality development process that incorporated disciplined specification and implementation of design requirements. Potential common-mode failures caused by software errors are also a concern. One of the protection means against – common-mode software failures is also accomplished by an emphasis on the quality process.

The Prom P/N software was initially developed approximately 15 years ago, evolving into the present day configuration. Within this time frame the product that matured to incorporate enhancements and facility improved hardware design. The evolutionary process will be evaluated to ensure that the pre-developed (Legacy) software is sufficiently reliable for use in nuclear safety related applications.

3.1 Organization

In order to ensure the program supports high quality and reliability, a process of independent technical reviews and evaluations will be performed. The project will be functionally organized under a Project Engineering Manager. The Project Manager will co-ordinate the V&V activities, schedule formal reviews, and document the results of the V&V reviews. The Project Engineering manager may also serve as a member of the V&V review team. A Quality Assurance Engineer will also participate in design reviews to ensure the overall quality of the project is maintained.

The software testing process was strengthened by designating the responsibility for the validation testing to an independent V&V engineer and technician.

An overall project organizational chart is provided below:

Management
-Organization
-Resources
-Follow-up

Qualification
-Quality Assurance
-Quality Control
-V&V

Project Team

Development
-Concept Phase
-Design Phase
-Manufacturing Phase

The project organization is described below:

```
Syncor Management (SM)----------------------------------------Quality Management (QM)
                 |
                 |---Project Manager (PM)
                 |
                 |---Project Engineer (PE)
                 |
                 | --------Independent SV&V Engineer (V&V)
                 |
                 |---------V&V Test Support Technician (V&V)
                 |
                 |---Quality Engineer (V&V)
```

The staff members that will participate in the V&V effort are:

Name:	Function	Team	Resource
J. Hale	Systems Business Manager	SM	SM
Zis Giatis	QA Manager	QM	QM
Judy Ellis	Software Engineer	V&V	SE
Andy Lasko	Project Manager	PM	PM
Dave Warner	Reliability Engineer	V&V	RE
Dave Smith	Quality Engineer	V&V	QE
George Buck	Test Technician	V&V	TT

The Project Manager will participate in the V&V reviews, and has the authority to resolve issues raised during the V&V.

The [illegible] has the authority to approve documents produced in the V&V process

3.2 Master Schedule

The V&V project is presently planned for completion on an expedited basis, over a 3-month period. At the end of the process, a formal SV&V report will be issued. Progress will be reported on a periodic basis, typically monthly.

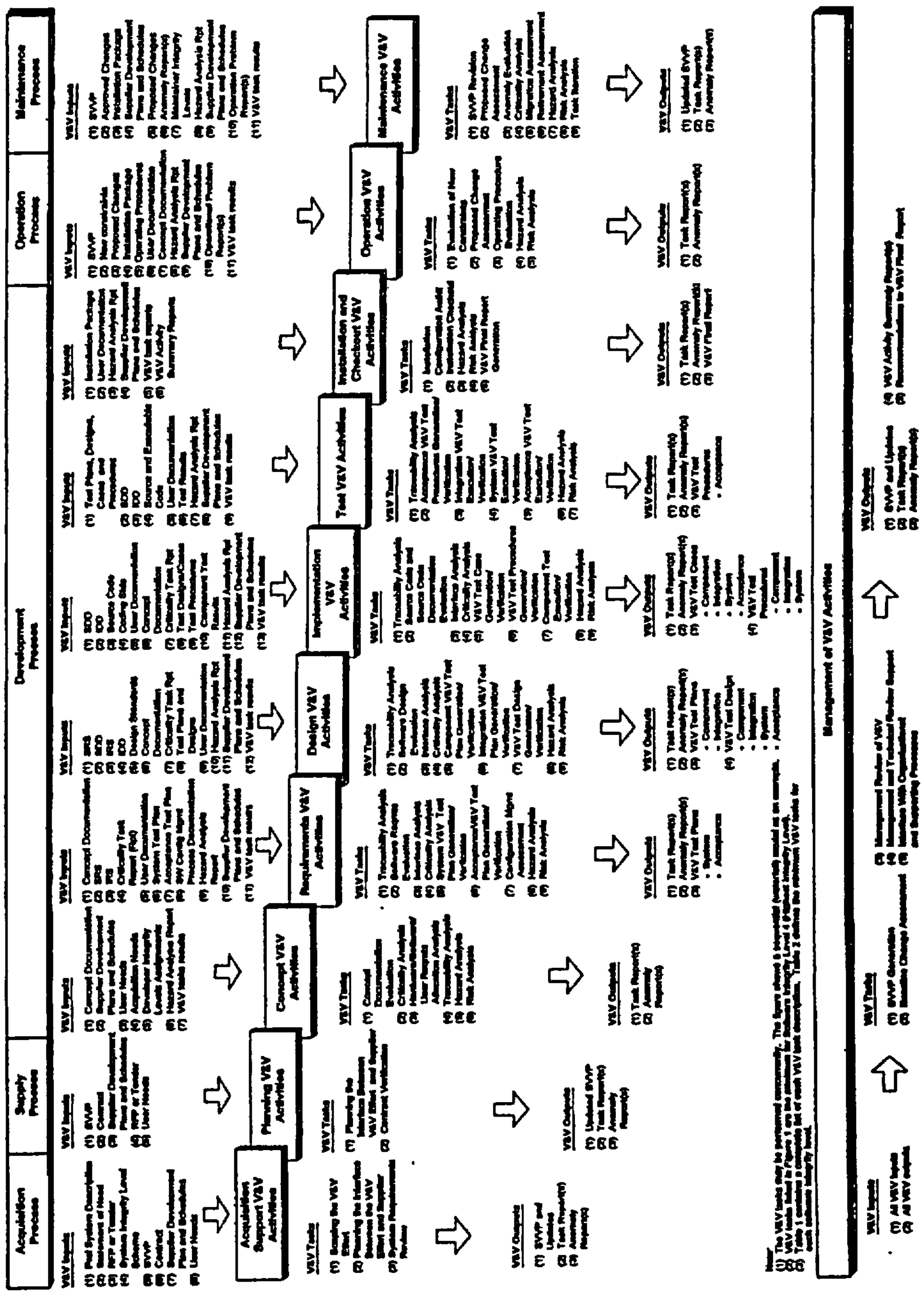
The SVV overview shown below summarizes the life-cycle model used for the project. It is based on the sample model defined in IEEE 1012, except as follows:

-For this project, the design phase has been previously completed, but has not been formally documented. This plan is designed to document the firmware that has been designed. The product is presently in the Maintenance phase.

-Installation, checkout, and operation are performed by the user

Major schedule milestones are listed below:

Complete SRS	Sep. 30, 2002
Complete SDD	Oct. 15, 2002
Complete VVTP	Oct. 30, 2002
Complete VVTR	Nov. 1, 2002



SYNCOR RADIATION MANAGEMENT	DATE 9/11/02	TITLE SOFTWARE VERIFICATION AND VALIDATION PLAN, 94095603				
REV	ECN NO.	RELEASED FOR	DOC CTRL	SHEET	NO.	SIZE
1	3044	PRODUCTION		10 of 27	94095603VVP	

3.3 Resources Summary

Resources available for this project will include:

3.3.1 Project Manager

3.3.2 Project Engineer

3.3.3 Software/Firmware Engineer

3.3.4 Test Technician

3.3.5 Quality Assurance Engineer

In addition to the above, the following equipment will be required:

3.3.6 1, 956A-201 UDR

3.3.7 1, Signal Generator

3.3.8 1, Digital Voltmeter

3.3.9 1, 94095603 EPROM

3.4 Responsibilities

3.4.1 The SRM Project Manager/Project Engineer is responsible for the implementation of this plan, identifying requirements, resolving problems, and ensuring compliance to the requirements identified by SRM personnel and any subcontractors employed.

3.4.2 The Software Engineer is responsible for reviewing the code, and providing the documents identified in the SV&V Plan. The Software Engineer is also responsible for implementing the V&V tests.

3.4.3 The Test Technician is responsible for assisting the Software Engineer with the V&V tests.

3.4.4 The Quality Assurance Engineer is responsible for reviewing the documents, and ensuring the quality requirements of the SV&V Plan are maintained.

3.5 Tools, Techniques, and Methodologies

The methods used in the V&V process will include review by cognizant engineering personnel, independent verification, and formal reviews.

The tools that will be used for the V&V process are as follows:

Document Preparation:

-Networked PC, Microsoft Word for Windows

-The documentation provided shall be written on a PC using a word processor program; e.g., Microsoft Word or a flat ASCII text editor, or similar. Each page of the document shall have a page header. The page header shall include the document name, part number, revision level and page number.

Target Hardware

-Model 956A-201 UDR with 94095603 EPROM

Test

Signal Generator

Software Testing

-American Arium Assembler/Linker

--DOS Based Personal Computer

For this project, Third Party Software is limited to assembly, emulation, linking and program development tools identified above. The Model 956 firmware is programmed assembly language, and does not include an operating system

4 Life-Cycle Verification and Validation

Outputs from phase tasks are used to develop corresponding V&V phase summary reports and are ongoing inputs to the SVVR. Outputs of V&V tasks become inputs to subsequent life-cycle V&V tasks.

5.1 Management of V&V

4.1.1 V&V Tasks, Inputs/Outputs, Resources and Responsibilities

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
Software Verification and Validation Plan (SVVP) Generation. Generate an SVVP for all life cycle processes. The SVVP may require updating throughout the life cycle. Outputs of other activities are inputs to the SVVP.	SVVP (previous update) Contract	SVVP and Updates	PM
Baseline Change Assessment. Evaluate proposed software changes (e.g., anomaly connections and requirement changes) for effects on previously completed V & V tasks. Plan iteration of affected tasks or initiate new tasks to address software baseline changes or iterative development processes. Verify and validate that the change is consistent with system requirements and does not adversely affect requirements directly or indirectly. An adverse effect is a change that could create new system hazards and risks or impact previously resolved hazards and risks	SVVP Proposed Changes Risks identified by V & V Tasks	Updated SVVP Task Report(s) – Baseline Change Assessment Anomaly Report(s)	PM
Management Review of V & V. Review and summarize the V & V effort to define changes to V & V tasks or to redirect the V & V effort. Recommend whether to proceed to the next set of V & V and development life cycle activities, and provide task reports, anomaly reports, and V&V Activity Summary Reports to the organizations identified in the SVVP. Verify that all V & V tasks comply with task requirements defined in the SVVP.	SVVP and Updates	Updated SVVP Task Report(s)-Recommendations V & V Activity Summary Reports Recommendations to the V&V Final Report	PM, SM, QE

4.1.2 Risks

The risks identified to date are:

4.1.2.1 V&V personnel requires capabilities and attitudes that differ from those encountered during software development.

Impact: A reduction in the motivation of the verifier/validator may have a negative effect on the quality of the product.

Action: Periodically, (each week), a meeting is held between the members of the V&V team and project manager. This meeting promotes teamwork:

- Each member of the V&V team to report work progress, to express any technical and personal communication problems encountered.

-Anticipation of events before they occur thus avoiding technical and motivational problems.

4.1.2.2 The projection of the workload involved in the V&V tasks may be incorrect (over- or underestimated, workload not well distributed).

Impact: adverse effect on schedule

Action: The periodic monitoring (monthly) perceives these shortcomings and defines corrective actions.

4.2 **Acquisition Support** (Acquisition Process) - Not Required; Customer inputs are specified in purchase documents, and customer review/approval of SVVP, SRS, and SDD will be obtained.

4.3 **Planning** (Supply Process) - Not Required; See 5.2 above

4.4 Development Process

4.4.1 Concept Phase of V&V

4.4.1.1 V&V tasks, Inputs/Outputs, Resources and Responsibilities

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
Concept Documentation Evaluation. Verify that the concept documentation satisfies user needs and is consistent with acquisition needs. Identify major constraints of interfacing systems and constraints or limitations of proposed approach. Assess criticality of each software item.	Concept Documentation User Needs Acquisition Needs	Task Report- Concept Documentation Evaluation Anomaly Report(s)	PM, QE, SM, SE

4.4.1.2 Risks

4.4.1.2.1 Product performance may not fully envelope customer requirements.

Impact: Be aware that initial performances may fall short of meeting all customer expectations.

Action: Anomalies will be identified and reviewed with the customer for ultimate disposition.

4.4.2 Requirements Phase of V&V

4.4.2.1 V&V tasks, Inputs/Outputs, Resources and Responsibilities

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
Traceability Analysis. Trace the software requirements (SRS) to system requirements (Concept Documentation) and system requirements. Analyze identified relationships for correctness, consistency, completeness, and accuracy.	Concept Documentation SRS	Task Report- Traceability Analysis Anomaly Report(s)	PM, QE, SM, SE
Software Requirements Evaluation. Evaluate the requirements (e.g., functional, capability, interface, qualification, safety, security, human factors, data definitions, user documentation, installation and acceptance, user operation, and user maintenance) of the SRS for correctness, consistency, completeness, accuracy, readability, and testability.	Concept Documentation SRS	Task Report(s)-Software Requirements Evaluation Anomaly Report(s)	PM, QE, SM, SE
Interface Analysis. Verify and validate that the requirements for software interfaces with hardware, user, operator, and other systems are connected, consistent, complete, accurate, and testable	Concept Documentation SRS	Task Report(s) - Interface Analysis Anomaly Report(s)	PM, QE, SM, SE
Criticality Analysis. Review and update any existing criticality analysis results from the prior Criticality Task Report using the SRS.	Task Report(s) – Criticality SRS	Task Report(s) – Criticality SRS	PM, QE, SM, SE
System V & V Test Plan Generation and Verification. *(For Software Integrity Levels 1 and 2)* Verify that developer's System Test Plans conform to Project defined test document purpose, format, and content (eg., see IEEE Std 829-1991). Validate that the System Test Plan satisfies the following criteria: 1) test coverage of system requirements; 2) appropriateness of test methods and standards used; 3) conformance to expected results; 4) feasibility of system qualification testing; and 5) capability to be operated and maintained.	Concept Documentation (System requirements) SRS User Documentation System Test Plan	Anomaly Report(s) System V&V Test Plan	PM, QE, SM, SE

4.4.2.2 Risks Not applicable

4.4.3 Design Phase of V&V

4.4.3.1 V&V tasks, Inputs/Outputs, Resources and Responsibilities

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
Traceability Analysis. Trace design elements (SDD), to requirements (SRS), and requirements to design elements. Analyze relationships for correctness, consistency, and completeness..	SRS SDD	Task Report(s)- Traceability Analysis Anomaly Report(s)	PM, QE, SM, SE
Software Design Evaluation. Evaluate the design elements (SDD) for correctness, consistency, completeness, accuracy, readability, and testability.	SRS SDD Design Standards (e.g., standards, practices, and conventions)	Task Report(s)- Software Design Evaluation Anomaly Report(s)	PM, QE, SM, SE
Interface Analysis. Verify and validate that the software design interfaces with hardware, user, operator, software, and other systems for correctness, consistency, completeness, accuracy, and testability.	Concept Documentation (System requirements) SRS SDD	Task Report(s) – Interface Analysis Anomaly Report(s)	PM, QE, SM, SE
V & V Test Design Generation and Verification. 1) system testing; and 2) acceptance testing. Continue tracing required by the V & V Test Plan. Verify that the V&V Test Designs comply with Project defined test document purpose, format, and content (e.g., see IEEE Std 829-1991). Validate that the V & V Test Designs satisfy the criteria in V&V tasks.	SDD User Documentation Test Plans Test Designs	System V&V Test Design(s) Acceptance V&V Test Design(s) Anomaly Report(s)	PM, QE, SM, SE

4.4.3.2 Risks Not applicable

4.4.4 Implementation Phase of V & V

4.4.4.1 V&V tasks, Inputs/Outputs, Resources and Responsibilities

The code will be reviewed for conventional indenting formatting. File headers, which includes the file name, the author, a description/purpose, definition of variables, sub-routines called, and the modification history, will be used for module modifications.

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
Traceability Analysis. Trace the source code components to corresponding design specifications(s), and design specification(s) to source code components. Analyze identified relationships for correctness, consistency, and completeness	SDD Source Code	Task Report(s) - Traceability Analysis Anomaly Reports	PM, QE, SM, SE
Source Code and Source Code Documentation Evaluation. Evaluate the source code components (Source documentation) for correctness, consistency, completeness, accuracy, readability, and testability.	Source Code SDD Coding Standards User Documentation	Task Report(s) – Source Code and Source Code Documentation Evaluation Anomaly Report(s)	PM, QE, SM, SE
Interface Analysis. Verify and validate that the software source code interfaces with hardware, user, operator, software, and other systems for correctness, consistency, completeness, accuracy, and testability.	Concept Documentation SDD Source Code User Documentation	Task Report(s) – Interface Analysis Anomaly Report(s)	PM, QE, SM, SE
V&V Test Case Generation and Verification. Verify that the developer's Test Cases conform to Project defined test document purpose, format, and content. Validate that the developer's Test Cases satisfy the criteria for system and acceptance testing.	SRS SDD User Documentation Test Design Test Cases	System V&V Test Cases Acceptance V&V Test Cases Anomaly Report(s)	PM, QE, SM, SE
V&V Test Procedure Generation and Verification. Verify that the developer's Test Procedures conform to Project defined test document purpose, format, and content. Validate that the developer's Test Procedures satisfy the criteria in V&V tasks for system and acceptance testing.	SRS SDD User Documentation Test Cases Test Procedures	System V&V Test Procedures Anomaly Report(s)	PM, QE, SM, SE
Hazard Analysis. Verify that the implementation and associated data elements correctly implement the critical requirements and introduces no new hazards. Update the hazard analysis.	Source Code SDD Hazard Analysis Report	Task Report(s) - Hazard Analysis Anomaly Report(s)	PM, QE, SM, SE
Risk Analysis. Review and update risk analysis using prior reports. Provide recommendations to eliminate, reduce or mitigate the risks.	Source Code Hazard Analysis Report V&V task results	Task Report(s) – Risk Analysis Anomaly Report(s)	PM, QE, SM, SE

4.4.4.2 Risks Not Applicable

4.4.5 Test Phase of V & V

4.4.5.1 V&V tasks, Inputs/Outputs, Resources and Responsibilities

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
Traceability Analysis. Analyze relationships in the V&V Test Plans, Designs, Cases, and Procedures for correctness and completeness. For correctness, verify that there is a valid relationship between the V&V Test Plans, Designs, Cases, and Procedures. For completeness, verify that all V&V Test Procedures are traceable to the V&V Test Plans.	V&V Test Plans V&V Test Designs V&V Test Procedures	Task Report(s) – Traceability Analysis Anomaly Report(s)	PM, QE, SM, SE
Acceptance V&V Test Procedure Generation and Verification. Verify that the developer's Acceptance Test Procedures conform to Project defined test document purpose, format, and content.	SDD Source Code User Documentation Acceptance Test Plan Acceptance Test Procedures	Acceptance V&V Test Procedures Anomaly Report(s)	PM, QE, SM, SE, RE
System V&V Test Execution and Verification. Use the developer's system test results to verify that the software satisfies the test acceptance criteria.	Source Code Executable Code User Documentation Acceptance Test Plan Acceptance Test Procedures Acceptance Test Results	Test Report(s) – Test Results Anomaly Report(s)	PM, QE, SM, SE, RE, TT
Hazard Analysis. Verify that the test instrumentation does not introduce new hazards. Update the hazard analysis	Source Code Executable Code Test Results Hazard Analysis Report	Task Report(s) – Hazard Analysis Anomaly Report(s)	PM, QE, SM, SE
Rick Analysis. Review and update risk analysis using prior task reports. Provide recommendations to eliminate, reduce, or mitigate the risks.	Hazard Analysis Report V&V task results	Task Report(s) – Risk Analysis Anomaly Report(s)	PM, QE, SM, SE

4.4.5.2 Risks Not Applicable

4.4.6 Installation and Checkout Phase of V & V

4.4.6.1 V&V tasks, Inputs/Outputs, Resources and Responsibilities

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
Installation Configuration Audit. Verify that all software products required to correctly install and operate the software are present in the installation package. Validate that all site-dependent parameters or conditions to verify supplied values are correct.	Installation Package (e.g.. Source Code, Executable Code, User Documentation, SDD, SRS, Concept Documentation, Installation Procedures, site-specific parameters, Installation Tests, and Configuration Management Data)	Task Report(s) - Installation Configuration Audit Anomaly Report(s)	PM, QE, SM, SE
Installation Checkout. Conduct analyses or tests to verify that the installed software corresponds to the software subjected to V & V. Verify that the software code and databases initialize, execute, and terminate as specified. In the transition from one version of software to the next, the V & V effort shall validate that the software can be removed from the system without affecting the functionality of the remaining system components. The V & V effort shall verify the requirements for continuous operation and service during transition, including user notification.	User Documentation Installation Package	Task Report(s) - Installation Checkout Anomaly Report(s)	PM, QE, SM, SE
Hazard Analysis. Verify that the installation procedures and installation environment does not introduce new hazards Update the hazard analysis	Installation Package Hazard Analysis Report	Task Report(s) – Hazard Analysis Anomaly Report(s)	PM, QE, SM, SE
Risk Analysis. Review and update risk analysis using prior task reports.	Installation Package Supplier Development Plans and Schedules V&V Task Results	Task Report(s) - Risk Analysis Anomaly Report(s)	PM, QE, SM, SE
V & V Final Report Generation. Summarize in the V & V final report the V&V activities, tasks and results, including Report (s) status and disposition of anomalies..	V & V Activity Summary Report(s)	V&V Final Report	PM, QE, SM, SE

4.4.6.2 Risks Not Applicable

4.5 Operation Phase of V & V

4.5.1 V&V tasks, Inputs/Outputs, Resources and Responsibilities

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
Evaluation of New Constraints. Evaluate new constraints (e.g., operational requirements, platform characteristics, operating environment) on the system or software requirements to verify the applicability of the SVVP. Software changes are maintenance activities (see 5.6 1).	SVVP New constraints	Task Report(s) – Evaluation of New Constraints	PM, QE, SM, SE
Proposed Change Assessment. Assess proposed changes (e.g., modifications, enhancements, or additions) to determine the effect of the changes on the system. Determine the extent to which V & V tasks would be iterated.	Proposed Changes Installation Package	Task Report(s)- Proposed Change Assessment	PM, QE, SM, SE
Operating Procedures Evaluation. Verify that the operating procedures are consistent with the user documentation and conform to the system requirements	Operating Procedures User Documentation Concept Documentation	Task Report(s) – Operating Procedures Evaluation Anomaly Report(s)	PM, QE, SM, SE
Hazard Analysis. Verify that the operating procedures and operational environment does not introduce new hazards. Update the hazard analysis.	Operating Procedures Hazard Analysis Report	Task Report(s) - Hazard Analysis Anomaly Report(s)	PM, QE, SM, SE
Risk Analysis. Review and update risk analysis using prior task reports. Provide recommendations to eliminate, reduce, or mitigate the risks.	Installation Package Proposed Changes Hazard Analysis Report Supplier Development Plans and Schedules Operation problem reports V&V task results	Task Report(s) - Risk Analysis Anomaly Report(s)	PM, QE, SM, SE
Installation and Operation. These tasks are assigned to Syncor Radiation Management.	Installation Package, Concept Documentation, SRS, Source Code Listings, Executable Code, User Documentation, SVVP, SVVR	Anomaly Report	PM, QE, SM, SE,, QM

4.5.1.1 Risks Not Applicable

4.6 Maintenance Phase of V & V

4.6.1 V&V tasks, Inputs/Outputs, Resources and Responsibilities

V & V Tasks	Required Inputs	Required Outputs	Resources Responsibilities
SVVP Revision. Revise the SVVP to comply with approved changes.	SVVP Approved Changes Installation Package	Updated SVVP	PM, QE, SM, SE
Proposed Change Assessment. Assess proposed changes (e.g., modifications, enhancements, or additions) to determine the effect of the changes on the system. Determine the extent to which V & V tasks would be iterated.	Proposed Changes Installation Package	Task Report(s)- Proposed Change Assessment	PM, QE, SM, SE
Anomaly Evaluation. Evaluate the effect of software operation anomalies.	Anomaly Report(s)	Task Report(s) – Anomaly Reports	PM, QE, SM, SE
Retirement Assessment. For software retirement, assess whether the installation package addresses: software support, impact on existing systems, software archiving, transition to a new software product, and user notification	Installation Package Approved Changes	Task Report(s) – Retirement Assessment Anomaly Report(s)	PM, QE, SM, SE
Hazard Analysis. Verify that software modifications correctly implement the critical requirements and introduce no new hazards Update the hazard analysis.	Proposed Changes Installation Package Hazard Analysis Report	Task Report(s) – Hazard Analysis Anomaly Report	PM, QE, SM, SE
Risk Analysis. Review and update risk analysis using prior task reports. Provide recommendations to eliminate, reduce, or mitigate the risks.	Installation Package Proposed Changes Hazard Analysis Report Supplier Development Plans and Schedules Operation problem reports V&V task results	Task Report(s) - Risk Analysis Anomaly Report(s)	PM, QE, SM, SE

4.6.1.1 During the maintenance phase, the developers may be assigned to other projects and may not be readily available to assist.

Impact: Lack of resources for immediate response to problems.

Action: Plan that resources familiar with the development be available to complete the maintenance phase work.

5 Software Verification and Validation Reporting

This section describes how the results of implementing the Plan will be documented.

5.1 Task Reporting

A report of each of the Tasks/Sub-tasks performed in the SVVP shall be developed and issued as they are completed. Listed below are the different reports to be generated.

Management	Progress reporting and internal notes
Documentation Evaluation	Documentation checking forms with review reports
Software/Firmware Testing	Software test report
Acceptance Testing	Acceptance Test Report
Others	Meeting reports or internal notes

5.2 V&V Phase Summary Report

A phase Summary Report shall summarize the results of V&V tasks performed in each of the following life-cycle phases: Requirements, Design, Implementation and Test. Each V&V Phase Summary report shall contain the following:

5.2.1 Description of SV&V tasks performed

5.2.2 Summary of test results

5.2.3 Summary of anomalies and resolutions

5.2.4 Recommendations

5.3 Anomaly Report

An anomaly report shall document each anomaly detected in the SV&V. The report content and administrative controls are provided in 7.1

5.4 Final Software Verification and Validation Report

The final report shall include a summary of the V&V activities and results. Deviation from the SV&V plan will be noted. Both positive and negative findings will be reported. Based on the results of the V&V, a conclusion and recommendations for further actions will be provided.

The format of the final report will be as follows:

Summary of each phase, to include:

5.4.1 Task results

5.4.2 Anomalies

5.4.3 Anomaly Resolution

5.4.4 Overall Quality Assessment

5.4.5 Conclusions

5.4.6 Recommendations

6 Verification and Validation Administrative Procedures

6.1 Anomaly Reporting and Resolution

As identified, anomalies will be written, and forwarded to the PM for logging. Each anomaly will be sequentially numbered by the PM.

Each anomaly will be presented to the review team for discussion and resolution. If mutual agreement cannot be reached, the PM will resolve the anomaly, and the process completed. Based on the severity of the anomaly, the originator may stop work, and request an immediate review meeting. Otherwise, the anomaly will be reviewed at the completion of the current V&V task, or phase.

6.2 Task Iteration Policy

A change request regarding a version results in the following processing with respect to the SV&V life cycle:

6.2.1 Analysis of the impact of the change (identification of items involved and the degree of the modification)

6.2.2 Repetition of the V&V cycle on items which change in order to check that the modifications have been taken into account in version n+1

6.3 Deviation Policy

When a deviation to the SVVP is identified, generation of an ECN, as described in QSP-05-08 will be required.

6.4 Control Procedures

All documents produced under the V&V program will be controlled and stored as any other engineering document, as described in QSP-05-08.

SRM classifies firmware as a drawing and therefore, follows SRM QSP-205 and QSP-05-08, Engineering/Document Change Notice Procedure, for its control. To this extent, the problem is documented using the Engineering Change Notice (ECN) procedure and sent to the Project Manager. Upon evaluation, the ECN will: 1) Be approved and implemented; 2) Be forwarded to the appropriate department for further action or; 3) Be returned with an explanation. Upon resolving the problem, the applicable documentation will be revised, and the corrected firmware will be released using the Engineering Change Notice (ECN).

Problems relating to monitor operation must be formally directed to the cognizant project engineer or Project Manager in the form of a field problem report. The format of the field problem report is not critical; however sufficient information (i.e., tag number, description of problem, operating mode, results observed, etc.) must be provided to permit the problem to be reproduced. The project engineer, or manager, will be responsible for resolving the problem report and, if required, initiate an internal ECN (per QSP –05-08) to revise the applicable firmware and documentation as required in this SVVP. Testing of revised firmware will be performed on hardware similar to that originally tested on.

Repetition of the affected portion of the V&V program will be required for and change affecting software that has been formally subjected to a V&V program.

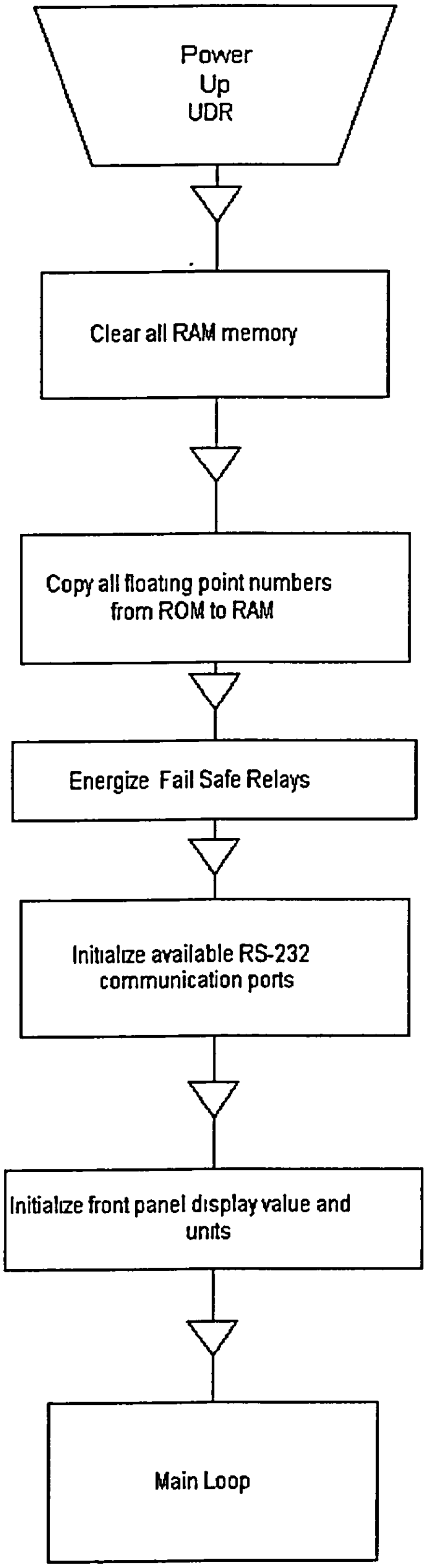
6.5 Standards, Practices, and Conventions

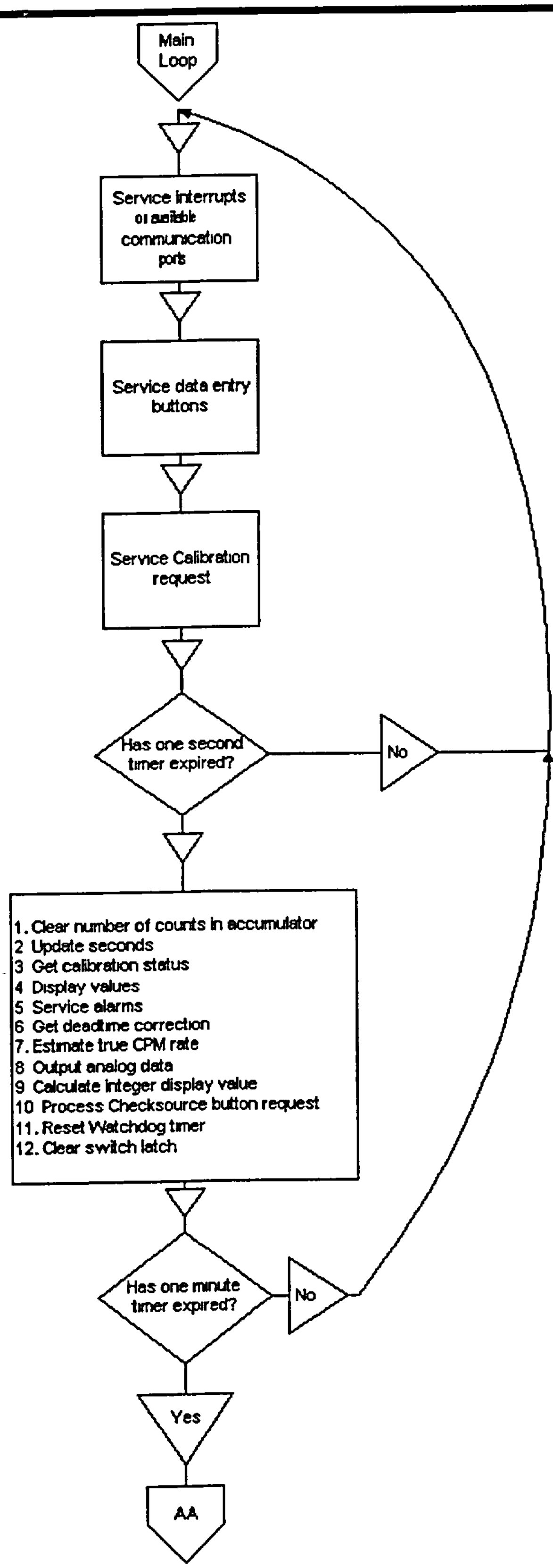
Refer to Section 4.0

APPENDIX A: List of all documents to be generated under this SVVP

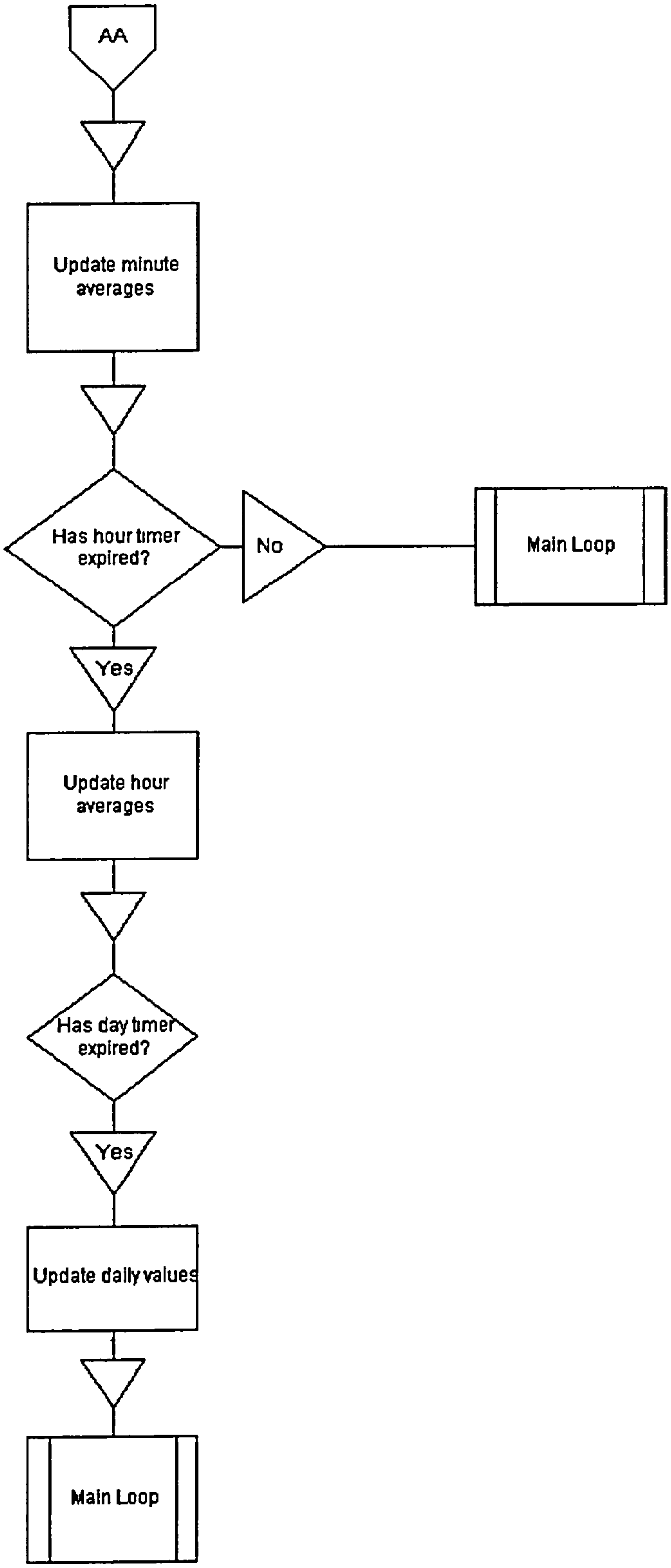
Document Number:	Description:
94095603SDD	Software Design Description
94095603SRS	Software Requirements Specification
94095603VVTP	Verification and Validation Test Procedure
94095603VVTR	Verification and Validation Test Report

ADDENDUM 1 – Firmware Flowchart, Page 1 of 3 pages





SYNCOR RADIATION MANAGEMENT			DATE 9/11/02		TITLE SOFTWARE VERIFICATION AND VALIDATION PLAN, 94095603	
REV 1	ECN NO. 3044	RELEASED FOR PRODUCTION	DOC CTRL	SHEET 2 of 3	NO. 94095603VVP	SIZE



SYNCOR RADIATION MANAGEMENT				9/11/02	TITLE SOFTWARE VERIFICATION AND VALIDATION PLAN, 94095603	
REV 1	ECN NO. 3044	RELEASED FOR PRODUCTION	DOC CTRL	SHEET 3 of 3	NO. 94095603VVP	SIZE

SOFTWARE REQUIREMENTS SPECIFICATION
FOR
PROM P/N 94095603
GM- AREA MONITOR

REV. LEVEL	ECO #	DESCRIPTION/PAGES AFFECTED
2	3076	Add backlight description, 3.1.1.1.2; Add optional to 3.1.3.1 and 3.1.4.1; Add clock interrupts to 3.1.3.2; Change range to 1.0E-2 to 1.0E+3, 3.2.1.2.1.11; Correct Aux relay operation, de-energize to trip, 3.2.1.3.2.1.2.4.4 and 3.2.1.3.2.1.2.5.2; Delete E0002 and E0008, 3.6.3
1	3052	Initial Release

ENGINEERING	[signature]	DATE 10-23-02
MANUFACTURING	[signature]	DATE 10-24-02
QUALITY ASSURANCE	[signature]	DATE 10-24-02

SYNCOR Radiation Management			10/23/02		TITLE SOFTWARE REQUIREMENTS SPECIFICATION 94095603	
REV 2	ECO NO. 3076	RELEASED FOR PRODUCTION	DOC CTRL RKM	SHEET 1 of 10	NO. 94095603SRS	SIZE PLN

Table of Contents

1. Introduction

This Software Requirements Specification is being developed to capture the firmware requirements for PROM P/N 94095603, used in the Model 956A G-M Digital Area Monitor Readout and to provide a benchmark to which designs will be validated.

1.1 Purpose of this document

The purpose of this document is to specify the functional and performance requirements for the Victoreen Model 956A Universal Digital Ratemeter (UDR), in accordance with Reference 1.4.2.

1.2 Scope of this document

The requirements were derived from the Installation, Operation, and Maintenance Instruction Manual for the Area Monitoring System Model 955A , published 5/96 by Victoreen, Inc, in accordance with Reference 1.4.2 and 1.4.4.

1.3 Definitions, acronyms, and abbreviations

1.3.1 Definitions

1.3.1.1 Checksource – radioactive source used to check detector and UDR operation

1.3.1.2 Contract – A legally binding document agreed upon by the customer and supplier. This includes the technical and organizational requirements, cost and schedule for a product. A contract may also contain informal but useful information such as the commitments or expectations of the parties involved.

1.3.1.3 Customer – The person, or persons, who pay for the product and usually (but not necessarily) decide the requirements. In the context of this recommended practice the customer and the supplier may be members of the same organization.

1.3.1.4 Detector – A plant mounted device that measures gamma radiation and converts it into an electrical signal

1.3.1.5 Fail safe - A condition for a relay where in normal operation the relay is energized

1.3.1.6 G-M – Geiger-Mueller tube radiation detector

1.3.1.7 Remote display device – An electronic device that displays information in a central location from a sensor, or detector, located within the plant

1.3.1.8 Set point – configuration parameter

1.3.1.9 Supplier – The person, or persons, who produce a product for a customer. In the context of this document, the customer and the supplier may be members of the same organization.

1.3.1.10 User – The person, or persons, who operate or interact directly with the product. The user(s) and the customer(s) are often not the same person(s).

1.3.1.11 VICO loop – Proprietary communication protocol

1.3.1.12 Watchdog Timer – A timer that must be reset on a repetitive basis, or it will time out and take a prescribed action.

1.3.2 Acronyms

1.3.2.1 DPDT - Double Pole Double Throw

1.3.2.2 EPRI - Electric Power Research Institute

1.3.2.3 IEEE - Institute of Electrical and Electronics Engineers

1.3.2.4 PROM - Programmable Read Only Memory

1.3.2.5 QAM - Quality Assurance Manual

1.3.2.6 RAM - Dynamic Random Access Memory

1.3.2.7 SPDT - Single Pole Double Throw

1.3.2.8 UDR - Universal Digital Ratemeter, Model 956A

1.3.3 Abbreviations

1.3.3.1 P/N - Part Number

1.3.3.2 V&V - Verification and Validation

1.4 References

1.4.1 IEEE Std 610.12-1990, Glossary of Software Engineering Terminology

1.4.2 IEEE Std 830-1993, Recommended Practice for Software Requirements Specifications

1.4.3 IEEE Std 1074-1991, Standard for Developing Software Life Cycle Processes

1.4.4 EPRI TR-103291 Handbook for Verification and Validation of Digital Systems (12/1998)

1.4.5 Software Verification and Validation Plan for Prom P/N 94095603, P/N 94095603VVP

1.4.6 SYNCOR QAM P/N QSP-100 Version 4.

1.5 Overview

The firmware described in this specification is specifically intended for use in the Model 956A UDR for the detection and measurement of ionizing radiation. When used with a Model 897A series G-M detector, the system monitors gamma radiation over a 5-decade range and provides indication when the radiation level increases above a high alarm set point, a warn alarm set point, an over range set point, or drops below a fail set point. Analog outputs are available for trend display on a strip chart recorder or a computer. The UDR also provides display, control, and annunciation functions. When equipped with the Model 942-200-80 Communications Loop Option circuit board, serial communications with a CRT terminal for status information, set point edit, and historical data retrieval is available.

2. Overall Description

This section describes the general factors that affect the firmware and its requirements.

2.1 Product Perspective

The Victoreen Model 956A UDR is a remote display device.

2.2 Product Functions

The Victoreen Model 956A UDR continuously displays radiation levels, indicates alarm status, updates analog outputs, and provides control and annunciation functions based on signals it receives from a specific detector. The UDR also provides a detector calibration and test function.

2.3 User Characteristics

The Victoreen Model 956A UDR should be used only by persons who have been trained in the proper interpretation of its readings and the appropriate safety procedures to be followed in the presence of radiation.

2.4 Constraints

The user needs to have the radiation levels continuously displayed, indication when alarm conditions exist, and be able to verify that the UDR is functioning correctly.

2.5 Assumptions and Dependencies

This firmware is specifically intended for use in the Model 956A UDR for the detection and measurement of ionizing gamma radiation. The system monitors gamma radiation over a 5-decade range and provides indication when the radiation level increases above a high alarm set point, a warn alarm set point, an over range set point, or decreases below a fail set point. The UDR must provide display, analog output, control, and annunciation functions.

2.6 General Constraints

All software source modules are written in Motorola 6802 assembly code using an ASCII text editor on a DOS based PC

3. Specific Functional Requirements
This section contains all the software requirements for the firmware.

3.1 External interface requirements
3.1.1 User interfaces
3.1.1.1 Front panel consisting of:
3.1.1.1.1 Seven segment, 3 digit display
3.1.1.1.2 3, Backlights, mR/h, R/h, and kR/h
3.1.1.1.3 5, Status indicators
3.1.1.1.4 24 segment Bargraph
3.1.1.1.5 4, Pushbuttons
3.1.1.1.5.1 Check Source
3.1.1.1.5.2 Alarm Acknowledge
3.1.1.1.5.3 HIGH alarm set point display
3.1.1.1.5.4 WARN alarm set point display
3.1.1.2 16, Position Function select switch
3.1.1.3 3, Data entry pushbuttons
3.1.1.3.1 Digit
3.1.1.3.2 Value
3.1.1.3.3 Enter
3.1.1.4 Calibration mode is entered manually by turning rotary set point switch to position 8 and pressing ENTER button
3.1.1.5 When is calibration mode, another calibration can be executed by pressing the ENTER key
3.1.1.5.1.1 System will again display calibration time for editing
3.1.1.6 Calibration in process can be stopped by pressing the ENTER key
3.1.1.7 Calibration can be stopped by moving the rotary switch to a position other than 8

3.1.2 Hardware interfaces
3.1.2.1 Power on/off button
3.1.2.2 Provide 1 analog output of 0 – 10 Vdc
3.1.2.3 Provide 2 analog outputs of 4 – 20 mA
3.1.2.4 Monitor shall have one failsafe SPDT high alarm output
3.1.2.5 Monitor shall have one failsafe DPDT warning output
3.1.2.6 Monitor shall have one failsafe DPDT fail output

3.1.3 Software interfaces
3.1.3.1 Message queues (optional)
3.1.3.2 Interrupts (clock interrupts)

3.1.4 Communications interfaces
3.1.4.1 RS-232 (optional)
3.1.4.2 VICO loop (optional)
3.1.4.3 RS-485 (optional)

3.2 System features

3.2.1 Continuously monitor radiation

3.2.1.1 To inform the user as to the presence and level of radiation within the monitored equipment or area.

3.2.1.2 Normal operation

3.2.1.2.1 Associated functional requirements

3.2.1.2.1.1 The radiation value is displayed as a three digit number

3.2.1.2.1.2 Backlit insert will display engineering units in mR/H, R/h, or kR/h.

3.2.1.2.1.3 The bargraph will display the value on the fixed mR/h scale

3.2.1.2.1.4 Any indicator lights that are on will be green

3.2.1.2.1.5 Analog outputs will track the displayed value.

3.2.1.2.1.6 Alarm outputs will be active.

3.2.1.2.1.7 The Warn relay will operate in Fail Safe Mode.

3.2.1.2.1.8 The High relay will operate in Fail Safe Mode.

3.2.1.2.1.9 The Fail relay will operate in Fail Safe Mode.

3.2.1.2.1.10 The serial port (optional)

3.2.1.2.1.11 Operates over a range of 1.00E-2 to 1.00E+3 mR/h

3.2.1.3 Inform when a predetermined limit is exceeded

3.2.1.3.1 Radiation value is below minimum .

3.2.1.3.1.1 Associated functional requirements

3.2.1.3.1.1.1 Fail outputs will be activated upon following conditions

3.2.1.3.1.1.1.1 No counts

3.2.1.3.1.1.1.2 Loss of power

3.2.1.3.1.1.1.3 MPU failure (Watch dog timer)

3.2.1.3.1.1.1.4 Detector anti-Jam occurs

3.2.1.3.2 Radiation value exceeds predetermined set point

3.2.1.3.2.1 Associated functional requirements

3.2.1.3.2.1.1 WARN condition:

3.2.1.3.2.1.1.1 Warn alarm condition is true when the display dose rate is greater than or equal to the WARN alarm set point.

3.2.1.3.2.1.1.2 WARN alarm logic is fail-safe

3.2.1.3.2.1.1.3 WARN alarm is manual reset

3.2.1.3.2.1.1.4 When WARN alarm is tripped:

3.2.1.3.2.1.1.4.1 Amber WARN alarm indicator begins flashing

3.2.1.3.2.1.1.4.2 Bagraph goes to amber

3.2.1.3.2.1.1.4.3 WARN alarm relay coil de-energizes

3.2.1.3.2.1.1.5 When WARN alarm is acknowledged:

3.2.1.3.2.1.1.5.1 WARN alarm indicator will be on steady

3.2.1.3.2.1.1.5.2 WARN relay will change state when the radiation value drops below the WARN alarm set point

3.2.1.3.2.1.1.6 The WARN alarm is normally inhibited in Check Source Mode

3.2.1.3.2.1.2 HIGH ALARM condition

3.2.1.3.2.1.2.1 HIGH alarm condition is true when the display dose rate is greater than or equal to the HIGH alarm set point.

3.2.1.3.2.1.2.2 HIGH alarm logic is fail safe

3.2.1.3.2.1.2.3 HIGH alarm is manual reset

3.2.1.3.2.1.2.4 When HIGH alarm is tripped:

3.2.1.3.2.1.2.4.1 Red HIGH alarm indicator begins flashing

3.2.1.3.2.1.2.4.2 Bargraph goes to red

3.2.1.3.2.1.2.4.3 HIGH relay coil de-energizes

3.2.1.3.2.1.2.4.4 The Auxiliary relay coil de-energizes

3.2.1.3.2.1.2.5 When HIGH alarm is acknowledged:
3.2.1.3.2.1.2.5.1 HIGH alarm indicator will be steady on
3.2.1.3.2.1.2.5.2 The Auxiliary relay will change state when the radiation value drops below the HIGH alarm set point.
3.2.1.3.2.1.2.5.3 HIGH alarm relay will change state when the radiation value drops below the HIGH alarm set point.
3.2.1.3.2.1.2.6 The HIGH alarm is normally inhibited in Check Source Mode
3.2.1.3.2.1.3 Range Alarm condition
3.2.1.3.2.1.3.1 When measured radiation field is below under range set point:
3.2.1.3.2.1.3.1.1 Front panel display will indicate 0.00 mR/h
3.2.1.3.2.1.3.1.2 Bargraph will indicate the actual radiation value
3.2.1.3.2.1.3.1.3 RANGE alarm indicator will illuminate in red.
3.2.1.3.2.1.3.1.4 The analog outputs are set to 0.00
3.2.1.3.2.1.3.2 When measured radiation field increases into the range of the detector.
3.2.1.3.2.1.3.2.1 RANGE alarm indicator will extinguish
3.2.1.3.2.1.3.2.2 Normal operation will begin
3.2.1.3.2.1.3.3 When measured radiation field is above over range set point:
3.2.1.3.2.1.3.3.1 WARN alarm is true
3.2.1.3.2.1.3.3.2 HIGH alarm is true
3.2.1.3.2.1.3.3.3 Red RANGE indicator is illuminated
3.2.1.3.2.1.3.3.4 Bargraph illuminates in red
3.2.1.3.2.1.3.3.5 Analog output reads full scale
3.2.1.3.2.1.3.3.6 Front panel display reads EEEEE
3.2.1.3.2.1.3.3.7 Over range alarm is reset by pressing acknowledge pushbutton
3.2.1.3.2.1.3.3.8 Over range alarm does not reset automatically
3.2.1.3.2.1.3.3.9 When detector output is above the electronic anti-jam circuit threshold the anti-jam circuit will be activated

3.2.1.4 Check Source Mode
3.2.1.4.1 Check source pushbuttons are provided to verify detector operation
3.2.1.4.1.1 When CHECK SOURCE pushbutton is held down:
3.2.1.4.1.1.1 Check source relay is energized
3.2.1.4.1.1.2 Check source indicator will illuminate (green)
3.2.1.4.1.1.3 Radiation value will be displayed on the front panel
3.2.1.4.1.1.4 Front panel High and Warn alarms status indicators are disabled
3.2.1.4.1.1.5 Analog outputs are set to zero
3.2.1.4.1.2 When CHECK SOURCE pushbutton is released:
3.2.1.4.1.2.1 Check source indicator will be extinguished
3.2.1.4.1.2.2 Normal UDR operation will resume

3.2.1.5 Calibration Mode
3.2.1.5.1 Current calibration time is displayed in seconds
3.2.1.5.2 First digit flashing to indicate the edit mode
3.2.1.5.3 Bargraph will turn off
3.2.1.5.4 High condition will clear
3.2.1.5.5 Warn condition will clear
3.2.1.5.6 Radiation units backlight will turn off
3.2.1.5.7 Calibration set point may be edited

3.2.1.5.8 The ENTER pushbutton will start the Calibrate timer
3.2.1.5.9 Whenever calibration is restarted the system will display calibration time for editing.
3.2.1.5.10 When the calibrate time expires, the backlight will be steady on
3.2.1.5.11 Once calibration mode is exited
3.2.1.5.11.1 UDR will reset
3.2.1.5.11.2 UDR will continue normal operation
3.2.1.5.12 Calibration mode can be entered with UDR in CHECK SOURCE mode
3.2.1.6 Data Entry Mode
3.2.1.6.1 Selected by pressing ENTER pushbutton while the rotary FUNCTION switch is in a valid set point position.
3.2.1.6.1.1 Selected set point is displayed in exponential format (e.g. 1.00E2)
3.2.1.6.1.2 Left most digit is flashing
3.2.1.6.1.3 Set point values are entered in exponential format (e.g. X..XXEN), where X is the mantissa and N is the exponent
3.2.1.6.1.4 X value may be any integer value between 0 and 9
3.2.1.6.1.5 Exponent may be positive or negative
3.2.1.6.1.6 Exponent may be any integer between 0 and 9
3.2.1.6.2 Function switch not on a valid set point has no effect.
3.2.1.6.3 Bargraph remains active
3.2.1.6.4 Analog outputs remain active

3.3 Performance requirements
3.3.1 Contact output logic shall be fail safe
3.3.2 Response time of the system for a change in radiation value is 60 seconds.
3.3.3 Display is updated once per second
3.3.4 Alarm is initiated within one second after the current one minute average exceeds the alarm setpoint.
3.3.5 After approximately 60 seconds the displayed value will indicate ambient radiation
3.3.6 Calibration mode will be reset to 60 seconds whenever unit is turned off
3.3.7 Calibration mode reset to 60 seconds when functional switch is moved from position 8
3.3.8 When checksource button is held down the check source relay will change state causing the mechanism in the detector to expose the check source
3.3.9 When checksource button is released, the check source relay will return to its normal state and the source capsule in the detector will be returned to its shielded position

3.4 Design constraints
3.4.1 Radiation rate range is 1.0E-2 to 1.0E+5 mR/hr
3.4.2 Operating temperature is 32 degrees F to 122 degrees F (0 degrees C to +50 degrees C)
3.4.3 Relative humidity is 0 to 99% non-condensing
3.4.4 Heat loading is approximately 96 BTU/hr
3.4.5 120Vac power requirement is approximately 28 watts
3.4.6 MC6802 Microprocessor
3.4.7 1MHz processor operation
3.4.8 8 bit word size
3.4.9 Interrupt capability
3.4.10 Software compatible with MC6800
3.4.11 32Kx8 UV erasable EPROM memory
3.4.12 8Kx8 RAM
3.4.13 64bytes, Electrically erasable memory (E^2)

3.5 Software system attributes

3.5.1 Reliability

The firmware will be designed/tested/controlled/maintained in accordance with the Verification and Validation Plan for PROM P/N 94095603.

3.5.2 Availability

Initialization will take place every time the system is powered up or reset.

3.5.3 Security

The operating program for the UDR resides on a PROM. It cannot be accidentally modified by the user.

3.5.4 Maintainability

The firmware will be written in assembly language. It will be modular by design. This is a simple system that does a limited set of functions.

3.6 Other requirements

3.6.1 There are to be nine user specified set points, switch selectable

- 3.6.1.1 HIGH alarm
- 3.6.1.2 WARN alarm
- 3.6.1.3 Resolving time detector dead time
- 3.6.1.4 Analog full scale limit
- 3.6.1.5 Over range limit
- 3.6.1.6 Conversion constant
- 3.6.1.7 Analog low scale
- 3.6.1.8 Calibrate mode
- 3.6.1.9 Under range value

3.6.2 UDR contains a series of eleven hardware jumpers for configuration purposes

- 3.6.2.1 JP1, Microprocessor reset
- 3.6.2.2 JP2, PROM type
- 3.6.2.3 JP3-1 - JP3-2, statistical Accuracy
- 3.6.2.4 JP3-3, Alarm acknowledge logic
- 3.6.2.5 JP3-4, Fail alarm
- 3.6.2.6 JP3-5, Check Source alarm
- 3.6.2.7 JP4 through JP7, Hardware options

3.6.3 Error codes are to be displayed

- 3.6.3.1 E0001 - Display value is negative
- 3.6.3.2 E0007 - Specific function is not implemented

3.6.4 Error codes are cleared automatically when initiating event is corrected.

SOFTWARE DESIGN DESCRIPTION

FOR

PROM P/N 94095603

G-M AREA MONITOR

REV. LEVEL	ECN #	DESCRIPTION/PAGES AFFECTED
2	3076	Add 24 bar graph segments and correlation to 4.2.13; Add Overrange frequencies and text to end of 4.2.31; Add SPDF30 used in 94095603 PROM, pg 106
1	3074	Initial Release

ENGINEERING	[signature]	DATE 10-23-02
MANUFACTURING	[signature]	DATE 10-24-02
QUALITY ASSURANCE	[signature]	DATE 10-24-02

SYNCOR RADIATION MANAGEMENT	DATE 10/23/02	TITLE SOFTWARE DESIGN DESCRIPTION 94095603

REV	ECN NO.	RELEASED FOR	DOC CTRL	SHEET	NO.	SIZE
2	3076	PRODUCTION		1 of 189	94095603SDD	PLN

Table of Contents

1. Introduction

This software design description document provides information on the architecture and coding of the UDR firmware PROM P/N 94095603, used in the Model 956A G-M AREA MONITOR. This document has been prepared in accordance with the requirements of ANSI/IEEE Std. 1016-1987.

1.1. Purpose

This document defines how the requirements specified in the SRS could be achieved using the software architecture defined in this document in accordance with Reference 2.2 and 2.6.

1.2. Scope

The system will be divided into modules, relationship between them and functionalities will be defined in accordance with Reference 2.2 and 2.6.

1.3. Definitions, acronyms, and abbreviations

1.3.1. Definitions

1.3.1.1. Architecture – The organizational structure of the system.

1.3.1.2. Commercial Grade Item Dedication – An acceptance process undertaken to provide reasonable assurance that a commercial grade item to be used as basic component will perform its intended safety functions and, in this respect, is deemed equivalent to an item designed and manufactured under a 10CFR Part 50, Appendix B, quality assurance program.

1.3.1.3. Component – A distinct part of a subsystem. A component may be decomposed into other components and computer software units.

1.3.1.4. Computer Program – A combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions.

1.3.1.5. Critical Characteristics – Those important design, material, and performance characteristics of a commercial grade item that, once verified, will provide reasonable assurance that the item will perform its intended safety function.

1.3.1.6. Data – A representation of facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automatic means.

1.3.1.7. Data Flow – The sequence in which data transfer, use, and transformation are performed during the execution of a computer program.

1.3.1.8. Deadtime – tau value, function of the detector and supplied with detector calibration sheet.

1.3.1.9. Dedicator – Refers to the dedicating entity: the organization that performs the dedication process.

1.3.1.10. Dependability – A broad concept incorporating various characteristics of digital equipment, including reliability, safety, availability, maintainability, and others.

1.3.1.11. Design Description – A document that describes the design of a system or component.
1.3.1.12. Design Entity – An element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced.
1.3.1.13. Design Method – A definition of a set of essential entities.
1.3.1.14. Design Methodology – A guideline identifying how to design software.
1.3.1.15. Design View – A subset of design entity attribute information that is specifically suited to the needs of a software project.
1.3.1.16. Digital Equipment – Equipment containing one or more computers.
1.3.1.17. Entity Attribute – A named characteristic or property of a design entity that provides a systematic procedure for the statement of fact about the entity.
1.3.1.18. Firmware – Software that resides in read-only memory.
1.3.1.19. Fail Safe – A condition for a relay where in normal operation the relay is energized.
1.3.1.20. G-M – Geiger-Mueller tube gamma radiation detector
1.3.1.21. Goto – Is to be coded as a jump or branch depending on conditions
1.3.1.22. Hardware – The physical equipment used to process, store, or transmit computer programs.
1.3.1.23. Include Module – A program file containing variable declarations and definitions that is included in one or more source files, to guarantee that all the source files will be supplied with the same definitions and declarations.
1.3.1.24. Microprocessor – See "Computer"
1.3.1.25. Model – A representation of one or more aspects of a system.
1.3.1.26. Module - A program unit that is discrete and identifiable with respect to assembling/compiling with other units and loading.
1.3.1.27. Notation – A set of symbols used to represent design entities and entity attributes.
1.3.1.28. Nuclear grade equipment – Basic components designed and manufactured under a quality assurance program complying with 10CFR50 Appendix B.
1.3.1.29. Regression Testing – Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.
1.3.1.30. Requirements - The statement of needs by a user that triggers the development of a program, system, or project.
1.3.1.31. Robustness – The ability of the digital equipment to function correctly in the presence of invalid inputs or stressful environmental conditions.
1.3.1.32. Routine - A defined objective or characteristic action of a system or component software module that performs a specific action, is invoked by the appearance of its name in an expression, may receive input values and return a value.

1.3.1.33. Safety Systems – Those systems that are relied upon to remain functional during and following design basis events to ensure (i) the integrity of the reactor coolant pressure boundary, (ii) the capability to shut down the reactor and maintain it in a safe shutdown condition, or (iii) the capability to prevent or mitigate the consequences of accidents that could result in potential offsite exposures comparable to the 10CFR Part 100 guidelines.

1.3.1.34. Software – Computer programs, procedures, and data pertaining to the operation of a computer system. This includes software that is implement ed in firmware.

1.3.1.35. Software Design Description – A representation of a software system created to facilitate analysis, planning, implementation, and decision making.

1.3.1.36. Software Design Document – The output of design process in a presentable format.

1.3.1.37. Software Design Process – Organized tasks and activities of design, having appropriate specifications.

1.3.1.38. Software Design Process Specification – Know-how, technology of design, that specify operationally how to use methodology of design (standardized itself) together with standards for evaluating design, tools to support design automation, and documentation required to represent design information.

1.3.1.39. Software Tool – A computer program used in the development, testing, analysis, or maintenance of a program or its documentation.

1.3.1.40. Source Module – A program file that contains the code that is used to generate the program.

1.3.1.41. System Integration – The process of combining software components, hardware components, or both into an overall system.

1.3.1.42. System Testing – Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

1.3.1.43. Thermister - Solid state sensor which changes voltage output with temperature; very reliable, more expensive than thermostats, rapidly becoming the industry standard

1.3.1.44. Traceability – (1) The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another. (2) The degree to which each element in a software development product establishes its reason for existing.

1.3.1.45. Traceability Matrix – A matrix that records the relationship between two or more products of the development process.

1.3.1.46. Unit – (1) A separately testable element specified in the design of a computer software component. (2) A logically separable part of a computer program. (3) A software component that is not subdivided into other components. Note: The terms "module", "component", and "unit" are used interchangeably.

1.3.1.47. Unit Testing – Testing of individual hardware or software units or groups of related units.

1.3.1.48. Varistor - variable resistor: a two-element semiconductor with nonlinear resistance in which the resistance drops as the applied voltage increases. Varistors are often used as a safety device to short circuit transient high voltages in electronic circuits

1.3.1.49. Vendor – The organization that holds information on one or more of the following: the design, design development process, testing, operating history, error reporting, and original equipment manufacturer.

1.3.1.50. Verification and Validation (V&V) – The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements. The activities involved in verification and validation are equivalent, for digital systems, of activities that have traditionally been performed for design verification and acceptance testing of other types of equipment used in nuclear safety-related applications.

1.3.1.51. VICO loop – proprietary communication protocol used by Syncor to communicate with a dumb terminal, or other another digital system.

1.3.1.52. Watchdog Timer – A timer that must be reset on a repetitive basis, or it will time out and take a prescribed action.

1.3.2. Acronyms

1.3.2.1. AC - Alternating Current
1.3.2.2. ACIA - Asynchronous Communications Interface Adapter
1.3.2.3. ANSI – American National Standard
1.3.2.4. ASCII – American Standard Code for Information Interchange
1.3.2.5. BCD – Binary-Coded Decimal
1.3.2.6. CCR – Condition Code Register
1.3.2.7. CPM - counts per minute
1.3.2.8. CS – Clear to Send
1.3.2.9. CSA - Canadian Standards Association
1.3.2.10. D/A – Digital to Analog
1.3.2.11. DC – Direct Current
1.3.2.12. EEPROM – Electrically Erasable Programmable Read-Only Memory
1.3.2.13. EMI - ElectroMagnetic Interference
1.3.2.14. EPRI – Electric Power Research Institute
1.3.2.15. ETX – End of Transmission Text
1.3.2.16. FCC - Federal Communications Commission
1.3.2.17. IEEE – Institute of Electrical and Electronics Engineers, Inc
1.3.2.18. IRQ – Interrupt Request
1.3.2.19. LPC – Linear Predictive Coding
1.3.2.20. LSB – Least Significant Byte
1.3.2.21. MID – Middle Byte
1.3.2.22. MPU – MicroProcessor Unit
1.3.2.23. MSB – Most Significant Byte
1.3.2.24. PCB – Printed Circuit Board

1.3.2.25. PROM – Programmable Read Only Memory
1.3.2.26. PTC – Positive Temperature Coefficient
1.3.2.27. QAM – Quality Assurance Manual
1.3.2.28. RAM – Random Access Memory
1.3.2.29. ROM – Read-Only Memory
1.3.2.30. SDD - Software Design Descriptions
1.3.2.31. SRS - Software Requirement Specification
1.3.2.32. STX – Start of Text
1.3.2.33. TTL - Transistor – Transistor Logic
1.3.2.34. UDR - Universal Digital Ratemeter
1.3.2.35. SVVP – Software V & V Plan
1.3.2.36. UL - Underwriters Laboratories, Inc
1.3.2.37. V&V – Verification and Validation
1.3.2.38. Vac - Voltage Alternating Current
1.3.2.39. VDE - Verband Deutscher Elektrotechniker
1.3.2.40. WR - Write

1.3.3. Abbreviations
1.3.3.1. HEX – hexadecimal number
1.3.3.2. Hz - Hertz
1.3.3.3. kR/h – kiloRoentgens per hour
1.3.3.4. mR/h – milliRoentgens per hour
1.3.3.5. MONSTAT – monitor status
1.3.3.6. msec – milliseconds
1.3.3.7. ns – nanosecond
1.3.3.8. opcode – operation code
1.3.3.9. OPTBRD – option board
1.3.3.10. P/N – part number
1.3.3.11. R/h – Roentgens per hour
1.3.3.12. R/W – Read Write
1.3.3.13. uC/cc –micro-Curies per cubic centimeter
1.3.3.14. UV – Ultra Violet

2. References

2.1 IEEE Std 610.12-1990, Glossary of Software Engineering Terminology
2.2 ANSI/IEEE Std 1016-1987, Recommended Practice for Software Design Descriptions
2.3 IEEE Std 1016.1-1993, Guide to Software Design Descriptions
2.4 IEEE Std 1074-1991, Standard for Developing Software Life Cycle Processes
2.5 IEEE Std 1219-1992, Standard for Software Maintenance
2.6 EPRI TR-103291-CD, Handbook for Verification and Validation of Digital Systems
2.7 EPRI TR-106439, Guideline on Evaluation and Acceptance of Commercial Grade Digital Equipment for Nuclear Safety Applications
2.8 Software Requirement Specification for PROM P/N 94095603, P/N 94085603SRS
2.9 Software Verification and Validation Plan for Prom P/N 94095603, P/N 94095603VVP

2.10 SYNCOR QAM P/N QSP-100 Version 4.

3. Decomposition Description

The section describes the major functions of each module that is used in the 956A UDR firmware.

3.1. Module Decomposition

3.1.1. Continuously monitor radiation

3.1.1.1. Main module

3.1.1.1.1. Handles initialization on power up and when reset line is signaled

3.1.1.1.1.1. HIGH alarm light off

3.1.1.1.1.2. WARN alarm light off

3.1.1.1.1.3. Fail light off

3.1.1.1.1.4. Check Source light off

3.1.1.1.1.5. Bar graph green

3.1.1.1.2. Clear all RAM memory

3.1.1.1.3. Copy floating numbers from ROM to RAM

3.1.1.1.4. Energize the high and warn alarm relays if fail safe

3.1.1.1.5. Main loop

3.1.1.1.5.1. Check communication option

3.1.1.1.5.2. Service interrupts

3.1.1.1.5.3. Service data entry buttons

3.1.1.1.5.4. Check for calibration mode

3.1.1.1.5.5. Every second

3.1.1.1.5.5.1. Clear number of counts in accumulator

3.1.1.1.5.5.2. Update seconds

3.1.1.1.5.5.3. Get calibration status

3.1.1.1.5.5.4. Display values in selected units

3.1.1.1.5.5.5. Service alarms

3.1.1.1.5.5.6. Get deadtime correction

3.1.1.1.5.5.7. Estimate true CPM rate

3.1.1.1.5.5.8. Output to chart recorder

3.1.1.1.5.5.9. Process check source button

3.1.1.1.5.5.10. Reset watchdog timer

3.1.1.1.5.5.11. Clear switch latch

3.1.1.1.5.5.12. Wait for interrupt

3.1.1.1.5.6. Every minute

3.1.1.1.5.6.1. Update minutes

3.1.1.1.5.7. Every hour

3.1.1.1.5.7.1. Update hours

3.1.1.1.5.8. Every day

3.1.1.1.5.8.1. Update days

3.1.2. Inform user when predetermined limit has been exceeded or not met
- 3.1.2.1. If no count
  - 3.1.2.1.1. de-energize fail relay
  - 3.1.2.1.2. illuminate fail light
- 3.1.2.2. If loss of power
  - 3.1.2.2.1. de-energize fail, WARN and HIGH relays
- 3.1.2.3. If hardware failure (i.e. watch dog timer time out)
  - 3.1.2.3.1. de-energize fail relay
  - 3.1.2.3.2. illuminate fail light
- 3.1.2.4. If detector anti-Jam
  - 3.1.2.4.1. de-energize fail relay
  - 3.1.2.4.2. illuminate fail light
- 3.1.2.5. WARN condition
  - 3.1.2.5.1. If in Check Source Mode, block WARN alarm
  - 3.1.2.5.2. If display dose rate is greater than or equal to WARN alarm set point
    - 3.1.2.5.2.1. Flash amber WARN alarm indicator
    - 3.1.2.5.2.2. Set bar graph color to amber
    - 3.1.2.5.2.3. De-energize WARN alarm relay coil
  - 3.1.2.5.3. WARN alarm is acknowledged
    - 3.1.2.5.3.1. Set WARN alarm indicator to steady on
    - 3.1.2.5.3.2. When radiation value drops below WARN set point
      - 3.1.2.5.3.2.1. Change state of WARN alarm relay
      - 3.1.2.5.3.2.2. Turn off WARN alarm indicator light
      - 3.1.2.5.3.2.3. Set bar graph to normal color
- 3.1.2.6. HIGH alarm condition
  - 3.1.2.6.1. If in Check Source Mode, block HIGH alarm
  - 3.1.2.6.2. If display dose rate is greater than or equal to HIGH alarm set point
    - 3.1.2.6.2.1. Flash red HIGH alarm indicator
    - 3.1.2.6.2.2. Set bar graph to red
    - 3.1.2.6.2.3. HIGH alarm relay coil is de-energized
    - 3.1.2.6.2.4. Set auxiliary output high (option)
  - 3.1.2.6.3. HIGH alarm is acknowledged
    - 3.1.2.6.3.1. Set HIGH alarm indicator to steady on
    - 3.1.2.6.3.2. Set auxiliary output low (option)
  - 3.1.2.6.4. When radiation value drops below HIGH set point
    - 3.1.2.6.4.1. Change state of HIGH alarm relay
    - 3.1.2.6.4.2. Change bar graph color to amber
    - 3.1.2.6.4.3. Turn off HIGH alarm light

3.1.2.7. Range alarm condition
- 3.1.2.7.1. Measured radiation is below Under Range set point then
  - 3.1.2.7.1.1. Display value of 0.00 mR/h
  - 3.1.2.7.1.2. Display actual value on bar graph with fixed mR/h scale
  - 3.1.2.7.1.3. Illuminate RANGE alarm indicator in red
    - 3.1.2.7.1.3.1. When radiation value increases into the range of the detector,
    - 3.1.2.7.1.3.2. Turn off RANGE alarm indicator
- 3.1.2.7.2. Measured radiation is above the Over Range set point,
  - 3.1.2.7.2.1. Set WARN alarm status to true
  - 3.1.2.7.2.2. Set HIGH alarm status to true
  - 3.1.2.7.2.3. Illuminate red RANGE alarm indicator
  - 3.1.2.7.2.4. Set bar graph color to red
  - 3.1.2.7.2.5. Set analog output to full scale
  - 3.1.2.7.2.6. Display 'EEEEE' for radiation value on front panel
  - 3.1.2.7.2.7. When acknowledged and the reading is below the set point
    - 3.1.2.7.2.7.1. Reset Alarm Over Range
  - 3.1.2.7.2.8. When detector output is above the electronic anti-jam circuit threshold, the hardware Anti-Jam circuit is activated

3.1.3. Check Source Mode
- 3.1.3.1. While button is depressed
  - 3.1.3.1.1. Energize check source relay to expose the source capsule
  - 3.1.3.1.2. Illuminate check source indicator
  - 3.1.3.1.3. Display radiation value on front panel
  - 3.1.3.1.4. Disable HIGH and WARN status indicators
  - 3.1.3.1.5. Set analog outputs to zero
- 3.1.3.2. When check source button is released
  - 3.1.3.2.1. Disable check source
  - 3.1.3.2.2. Clear averaging buffer
  - 3.1.3.2.3. Turn off check source indicator

3.1.4. Calibration mode

3.1.4.1. Set the select switch to position 8, and press the ENTER button to start the calibration mode

3.1.4.2. Display current calibration time in seconds on front panel

3.1.4.3. Flash first digit to indicate edit mode

3.1.4.3.1 Permit editing of the calibrate time

3.1.4.4. Turn off bar graph

3.1.4.5. The Analog output is set to the last displayed value

3.1.4.6. Clear HIGH alarm condition

3.1.4.7. Clear WARN alarm condition

3.1.4.8. Turn off radiation backlight

3.1.4.9. Start the calibrate timer by pressing the ENTER button

3.1.4.9.1. Set display to zero

3.1.4.9.2. Flash the backlight units

3.1.4.9.3. Update the counts each second

3.1.4.10. Stop calibration in process

3.1.4.10.1. When calibrate time has expired

3.1.4.10.2. When ENTER button is pressed

3.1.4.10.3. When rotary switch is moved out of position 8

3.1.4.10.4. Display the total counts received

3.1.4.10.5. Set backlights to steady on

3.1.4.11. If ENTER key is pressed when calibration is done

3.1.4.11.1. Display calibration time in seconds on front panel

3.1.4.11.2. Enable editing of calibration time

3.1.4.11.3. Wait for ENTER button to be pressed to start timer

3.1.4.12. Exit calibration mode when rotary switch is moved out of position 8

3.1.4.13. Reset UDR to normal operation when calibration mode is exited

3.1.4.14. Start Calibrate mode in check source

3.1.4.14.1 When the rotary switch is in position 8

3.1.4.14.2 When the check source button is pressed and held down and

3.1.4.14.3 When the ENTER button is pressed

3.1.4.14.4 The check source indicator will illuminate

3.1.4.14.5 When the check source button is then released

3.1.4.14.1 The check source will be locked on

3.1.4.14.2 The check source indicator will remain lit

3.1.4.14.6 Press the ENTER button to start the calibrate timer

3.1.4.14.7 Exit calibration mode when rotary switch is moved out of position 8

3.1.4.14.8 Clear check source condition

3.1.5. Data Entry Mode

3.1.5.1. If select switch is set to valid set point number and ENTER is pressed

3.1.5.1.1. Display selected set point in exponential format

3.1.5.1.2. Flash left most digit for editing

3.1.5.1.3. Read set point values in exponential format (X.XXEN)

3.1.5.1.3.1. X is the mantissa

3.1.5.1.3.2. X can be any integer value between 0 and 9

3.1.5.1.3.3. N is the exponent

3.1.5.1.3.4. Exponent can be positive or negative

3.1.5.1.3.5. Exponent is any integer between 0 and 9

3.1.5.1.4. Switch position on invalid set point report error E0007

3.1.5.1.5. Display radiation values on bar graph in fixed mR/h scale

3.1.5.1.6. Set analog outputs to monitored radiation values

3.1.5.1.7. Push ENTER button to enter the value into memory

3.1.6. Set Points

3.1.6.1. Set Point 0 – HIGH alarm

3.1.6.2. Set Point 1 – WARN alarm

3.1.6.3. Set Point 2 – Resolving Time (Dead Time)

3.1.6.4. Set Point 3 – Analog Full Scale

3.1.6.5. Set Point 4 – Overrange

3.1.6.6. Set Point 5 – Conversion Constant

3.1.6.7. Set Point 6 - Not Used

3.1.6.8. Set Point 7 – Analog Low Scale

3.1.6.9. Set Point 8 – Calibrate Mode

3.1.6.10. Set Point 9 – Underrange

3.1.6.11. Set Points A through F - Not Used

3.1.6.12. Error Code

3.1.6.12.1. E0001 – Negative display data

3.1.6.12.2. E0002 - Invalid set point value

3.1.6.12.3. E0007 - Invalid function

3.1.6.12.4. E0008 - Invalid analog scale value

3.1.6.12.5. EEEEE - Overrange condition

3.1.6.12.6. Error codes are cleared automatically when initiating event is corrected.

3.2. Concurrent Process Description

The UDR software/firmware is a simple system. On power up, initialization takes place. Then the system goes into a sleep state until the timing interrupt occurs. Then all of its selected functions are performed. The Main Loop timing cycle is 1 second.

3.3. Data Decomposition

Operating parameters (i.e. User set points) are stored in EEPROM. Data is passed between functions via accumulator A, accumulator B, index register X, calling parameter or public variable.

3.3.1. Conversion Constant Set Point

The conversion constant converts the detector pulse rate into an mR/h value, using the following equation:

D = (CPM x K) – BKG

Where: D = the calculated value in mR/h (used for alarm set point limit checks)
CPM = the current, true, count rate in CPM (the sum of the 60 most recent 1 seconds values, corrected for Tau).
K = the conversion constant in mR/h/CPM
BKG = 0 (not used)

The value of K is supplied on the detector calibration data sheet.

3.3.2. Resolving Time (Dead Time)

This constant is a correction for the resolving time of the detector. As the radiation field that the detector is viewing increases, the detector cannot count every pulse, because some are in coincidence or are so close together that two pulses may look like one. To correct for this nonlinearity, the resolving time is corrected by the following equation:

CPM = Ro/(1-(Ro x Dead Time))

Where: CPM = the true count rate
Ro = the observed count rate (the sum of the 60 most recent 1 second values)
Dead Time = the resolving time in minutes/count

The value of Dead Time is supplied with the detector and is found on the detector calibration data sheet. This value is identified as TAU on the data sheet.

3.3.3 Analog Output

The analog output is a logarithmic function of the displayed value. An 8 bit DAC is used to convert the displayed value to an 4-20 mA and a 0-10Vdc analog output. The analog output full scale and low scale set points are used to scale the output to the range of the detector. The following equation is used to calculate the analog output voltage or current:

$$P = \log(R / LSV) / [\log(FSV) - \log(LSV)]$$

And

$$V = P(V_{max} - V_{min}) + V_{min} \quad \text{or} \quad I = P(I_{max} - I_{min}) + I_{min}$$

Where:

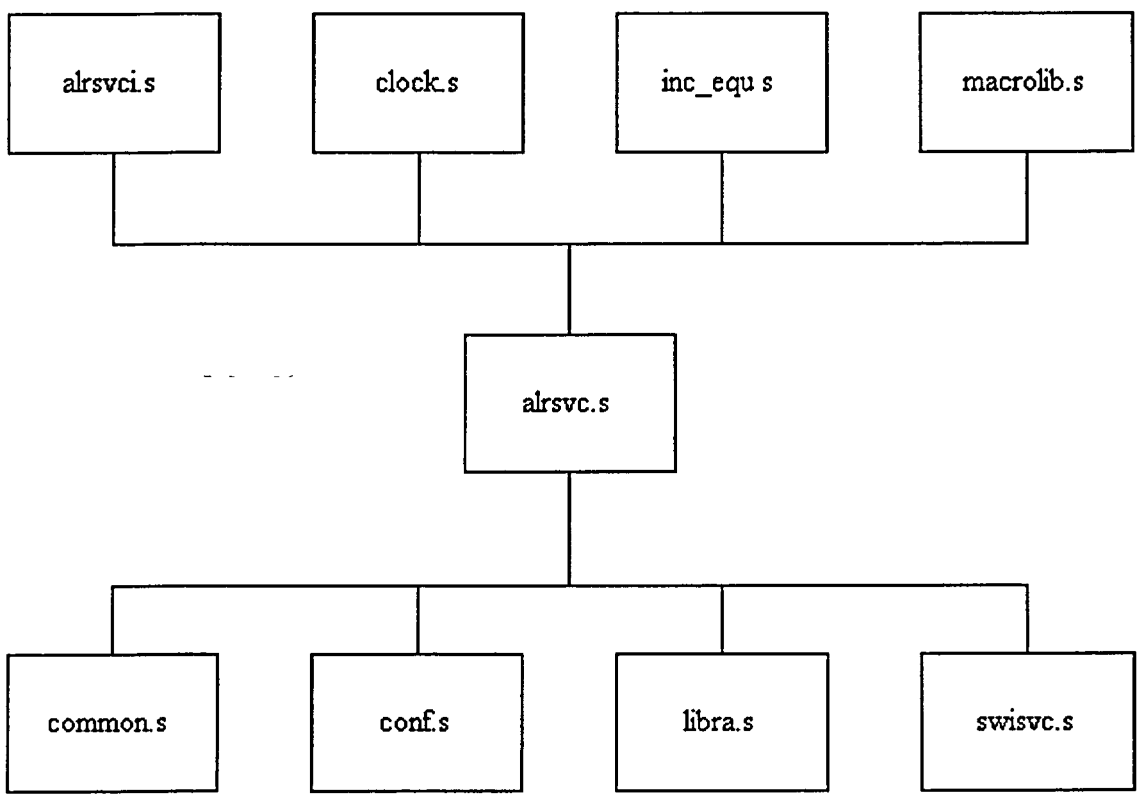
P =	Percent of scale, expressed as a decimal number
R=	Current reading
LSV=	Low scale value
FSV=	Full scale value
V=	Voltage output
I=	Current output
V_{max}=	Maximum voltage available (usually 10Vdc)
V_{min}=	Minimum voltage available (usually 0 Vdc)
I_{max}=	Maximum current available (usually 20 mA)
I_{min}=	Minimum current available (usually 4 mA)

4. Dependency Description
This is a simple system. It performs a finite set of functions every second. This section shows the dependency between modules.

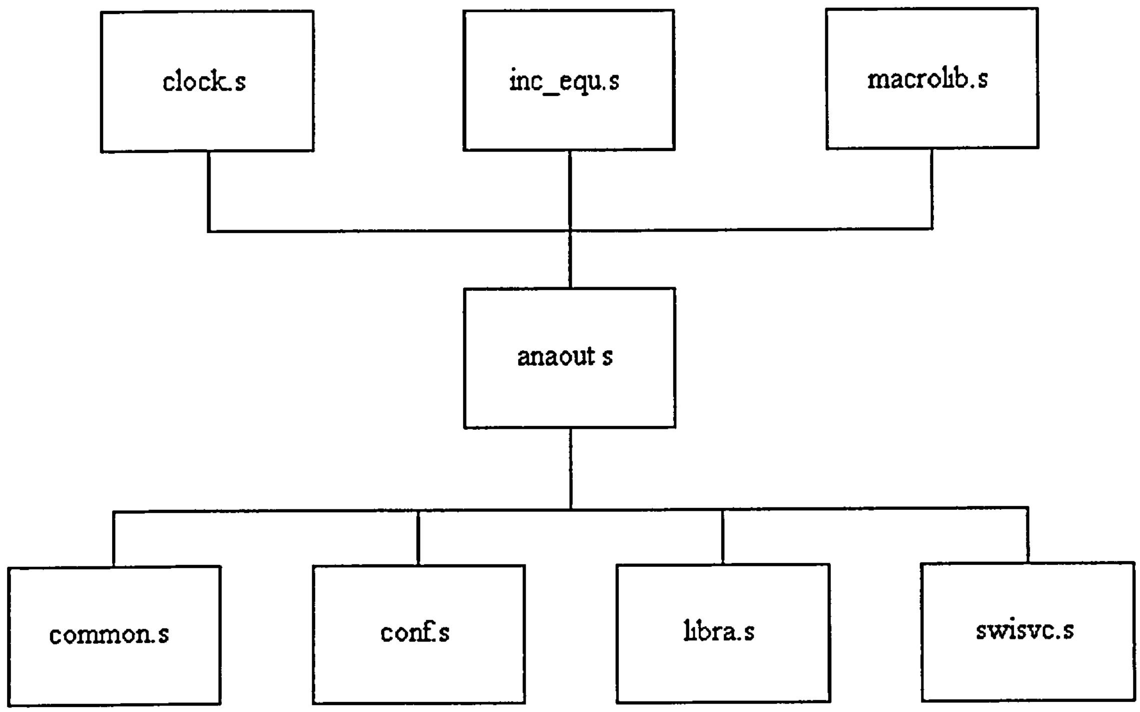
4.1. Intermodule Dependencies

All modules above the reference module are either modules that are included in the reference module or make calls to the routines within the reference module. All modules below the reference module either have routines that are called by the reference module or contain data that is used or modified by the reference module.

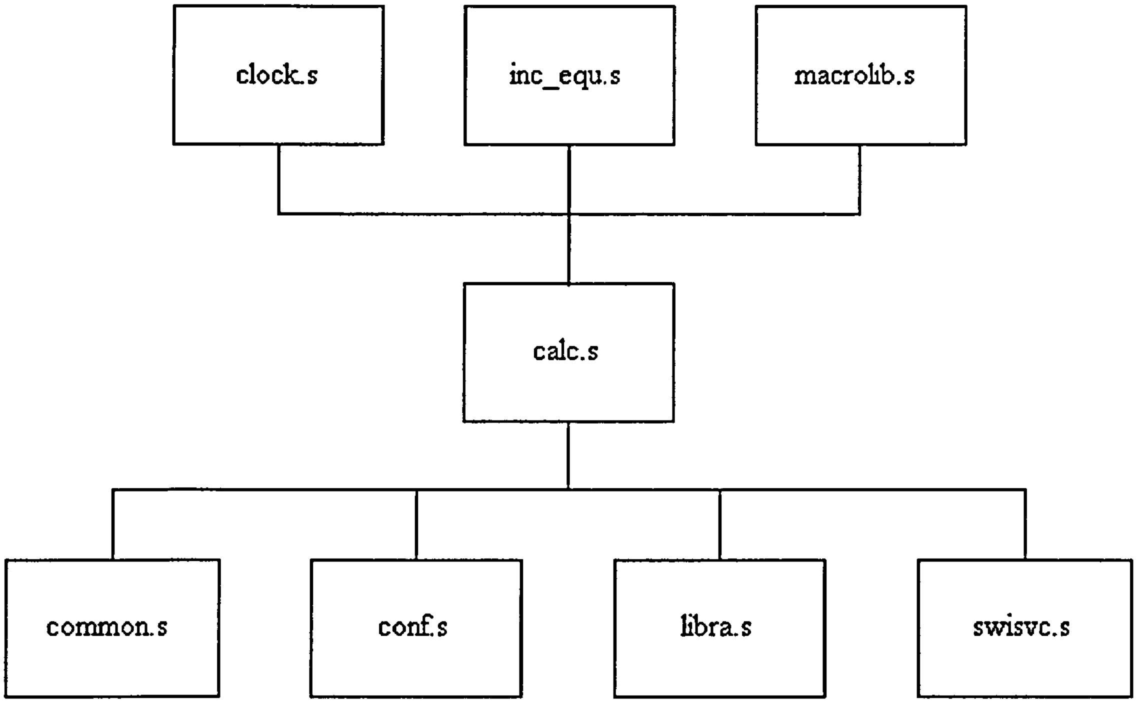
4.1.1. Alrsvc.s



4.1.2. Anaout.s

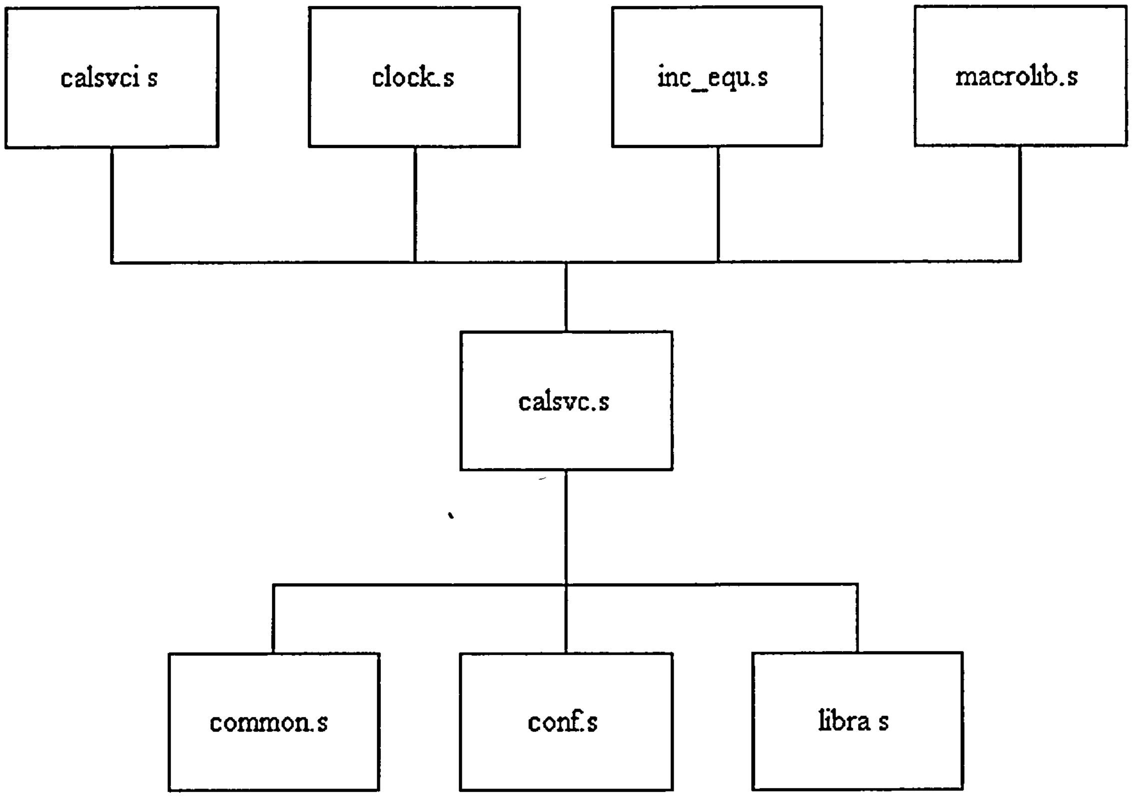

clock.s
inc_equ.s
macrolib.s
anaout s
common.s
conf.s
libra.s
swisvc.s

4.1.3. Calc.s

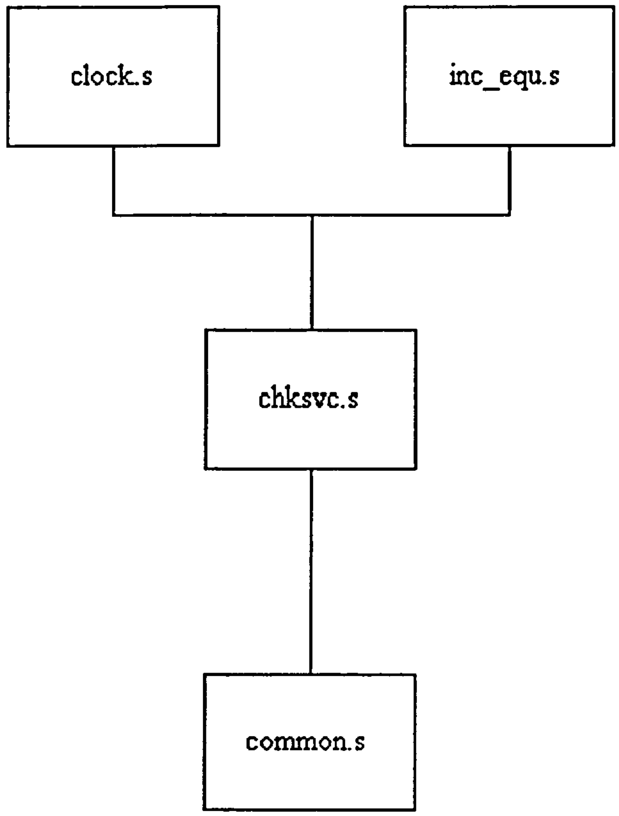


clock.s
inc_equ.s
macrolib.s
calc.s
common.s
conf.s
libra.s
swisvc.s

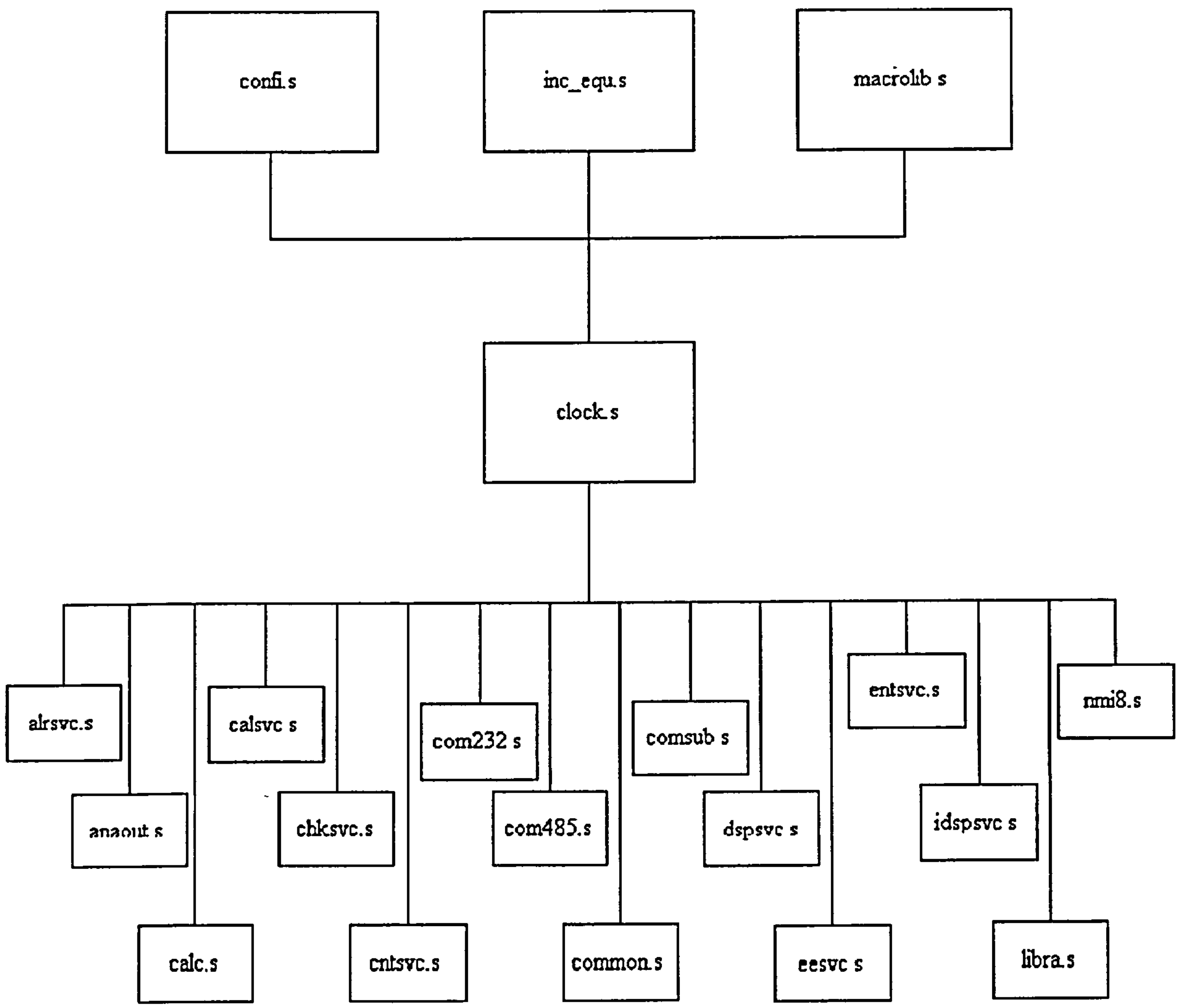
4.1.4. Calsvc.s


calsvci s
clock.s
inc_equ.s
macrolib.s
calsvc.s
common.s
conf.s
libra s

4.1.5. Chksvc.s

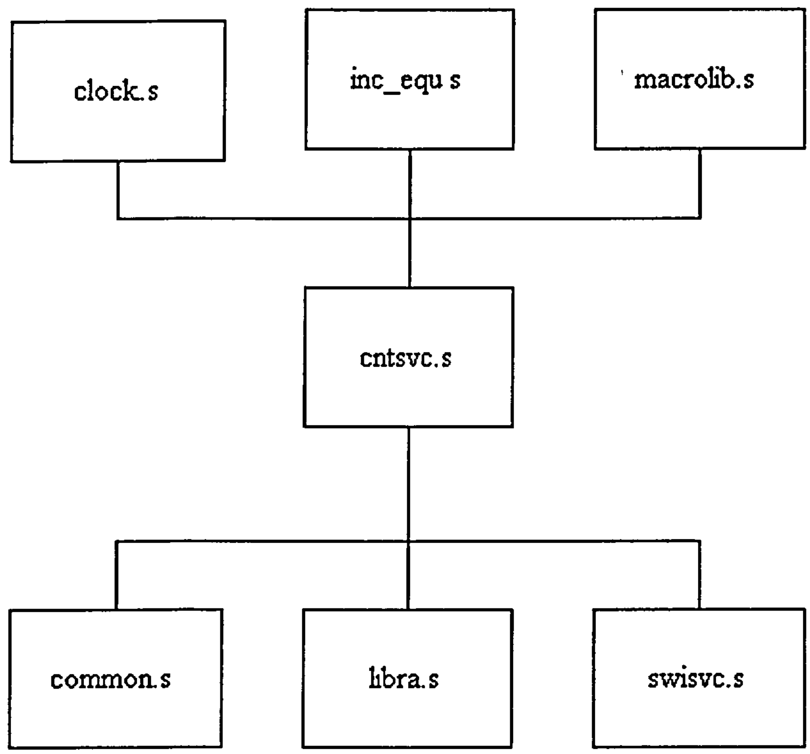


4.1.6. Clock.s

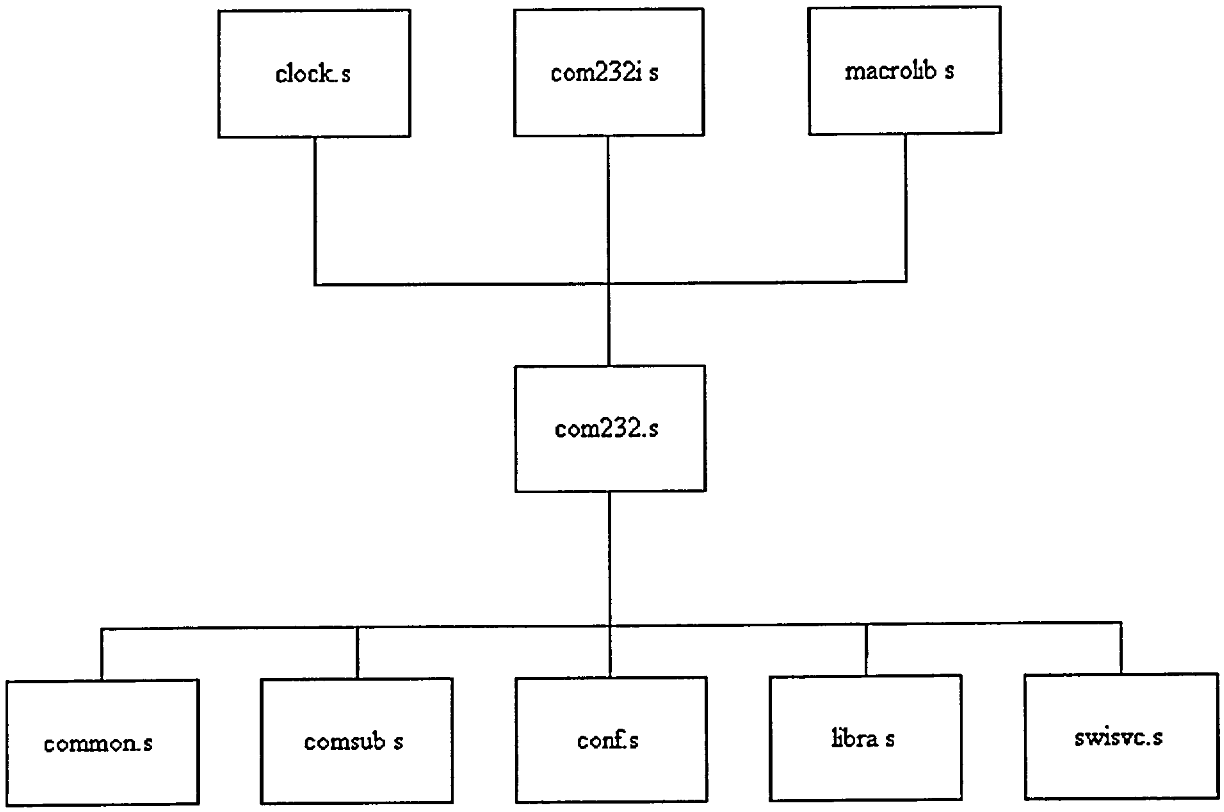


confi.s
inc_equ.s
macrolib.s
clock.s
alrsvc.s
anaout.s
calc.s
calsvc.s
chksvc.s
cntsvc.s
com232.s
com485.s
common.s
comsub.s
dspsvc.s
eesvc.s
entsvc.s
idspsvc.s
libra.s
nmi8.s

4.1.7. Cntsvc.s

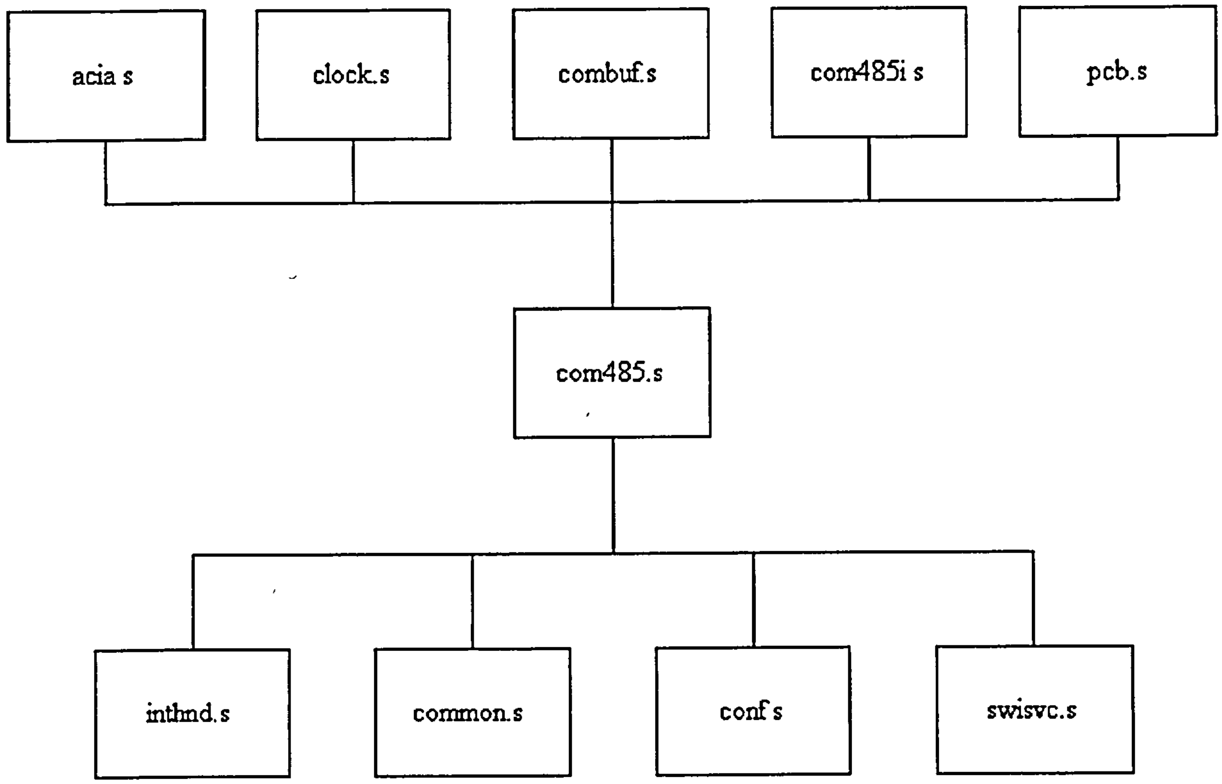


4.1.8. Com232.s



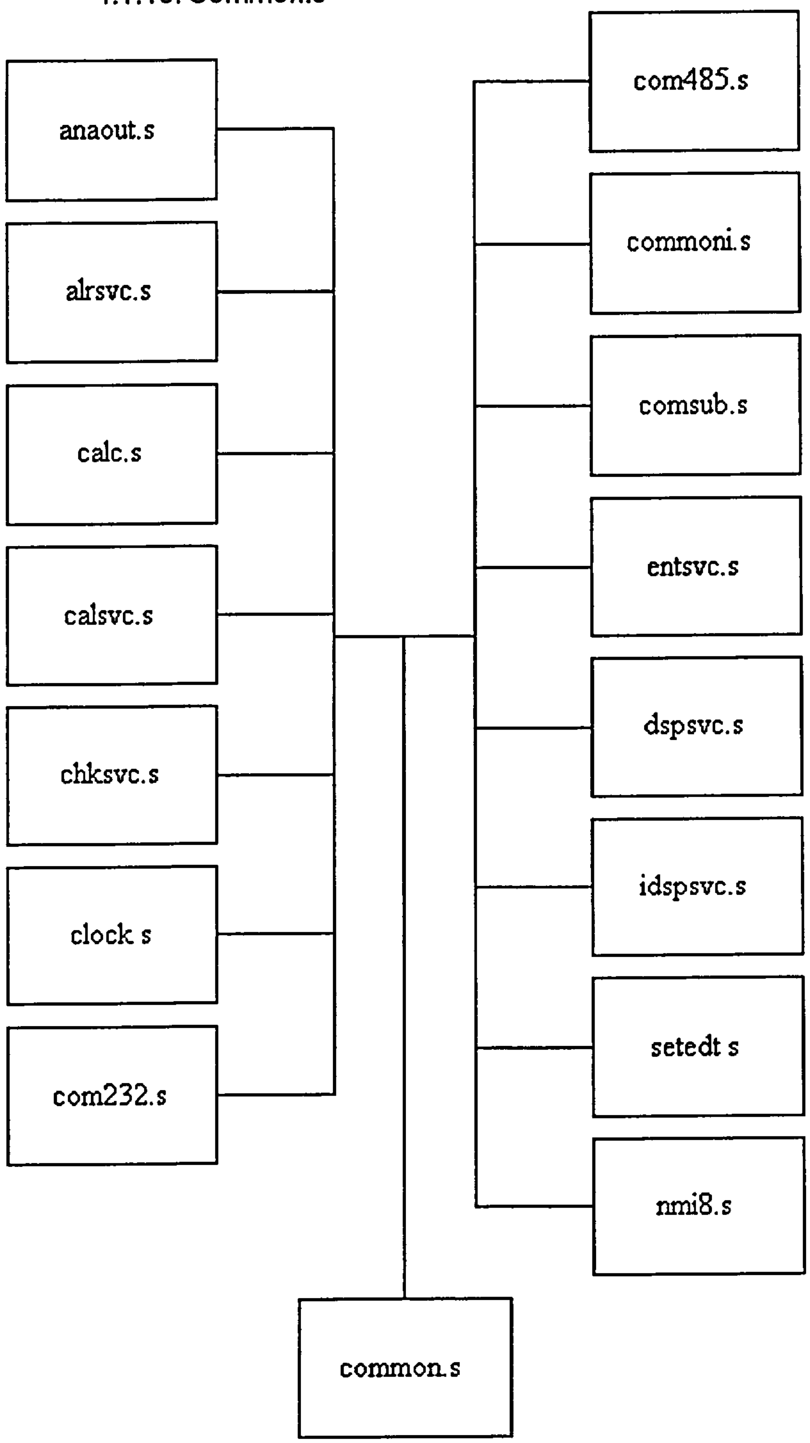
clock.s
com232i.s
macrolib.s
com232.s
common.s
comsub.s
conf.s
libra.s
swisvc.s

4.1.9. Com485.s

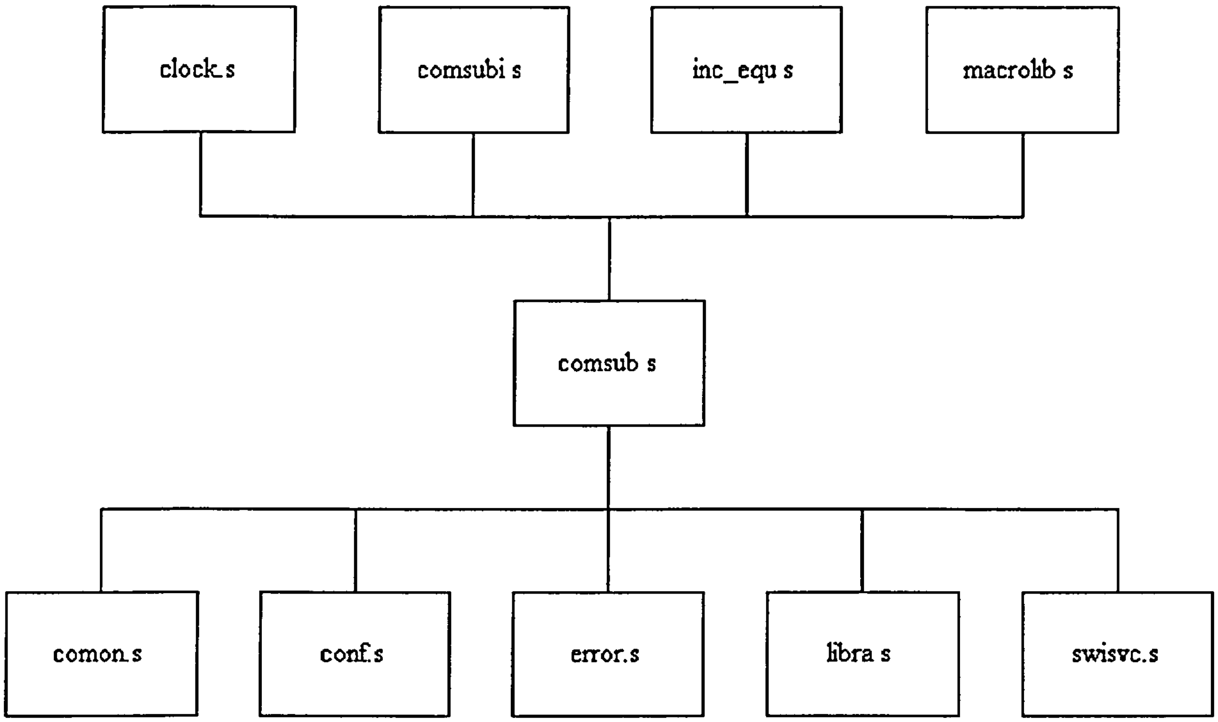


acia s
clock.s
combuf.s
com485i s
pcb.s
com485.s
inthnd.s
common.s
conf s
swisvc.s

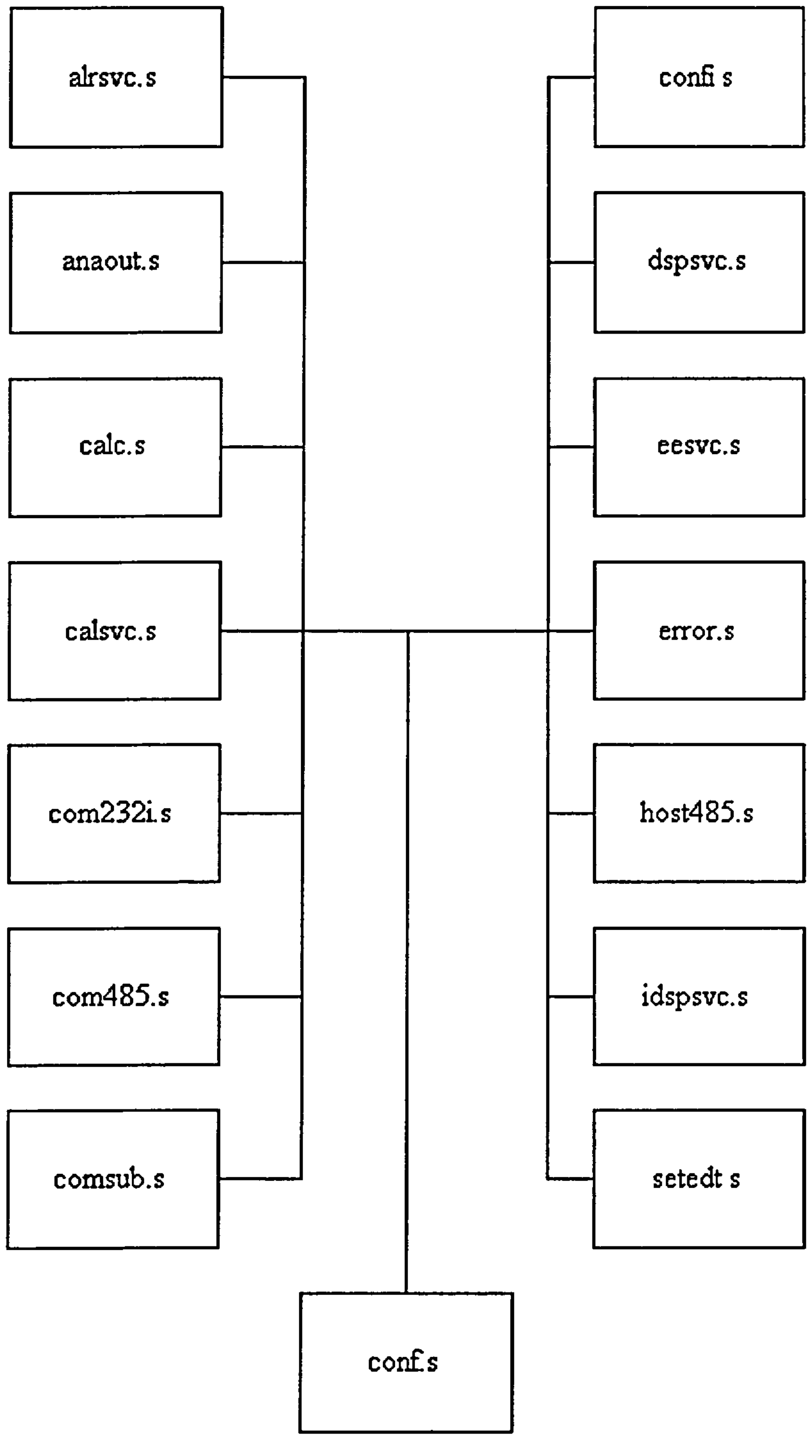
4.1.10. Common.s



4.1.11. Comsub.s

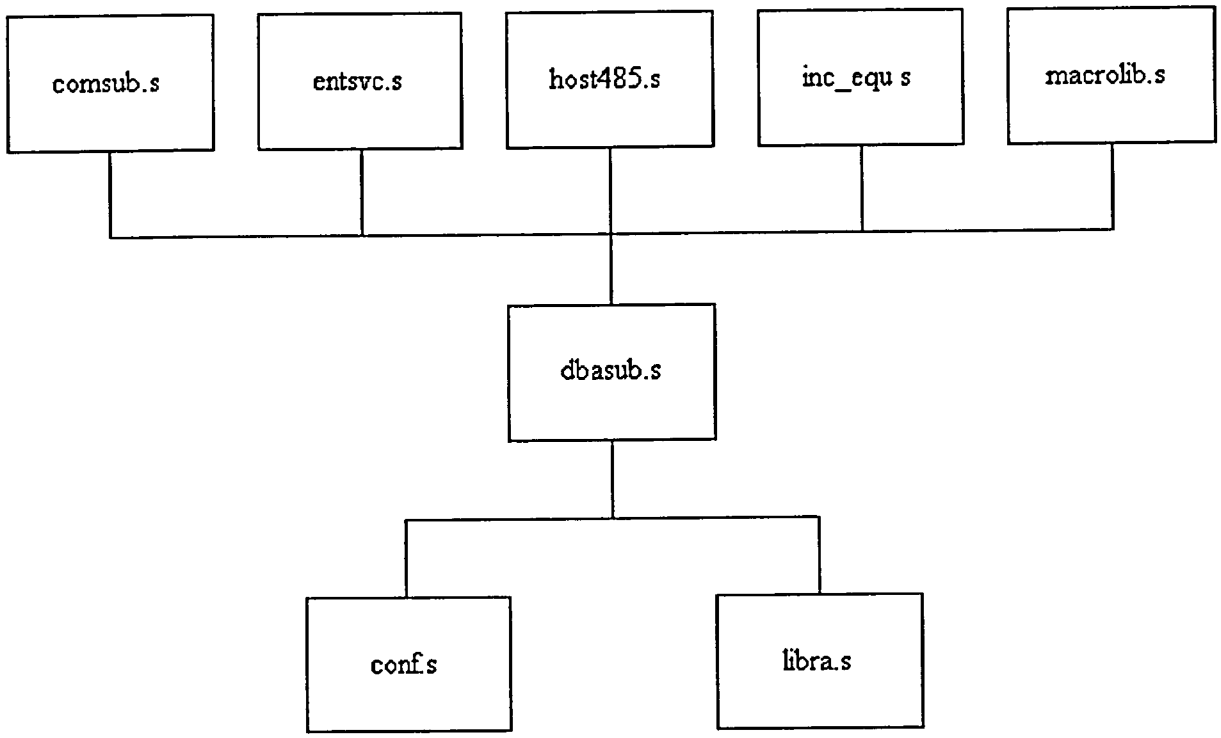


4.1.12. Conf.s

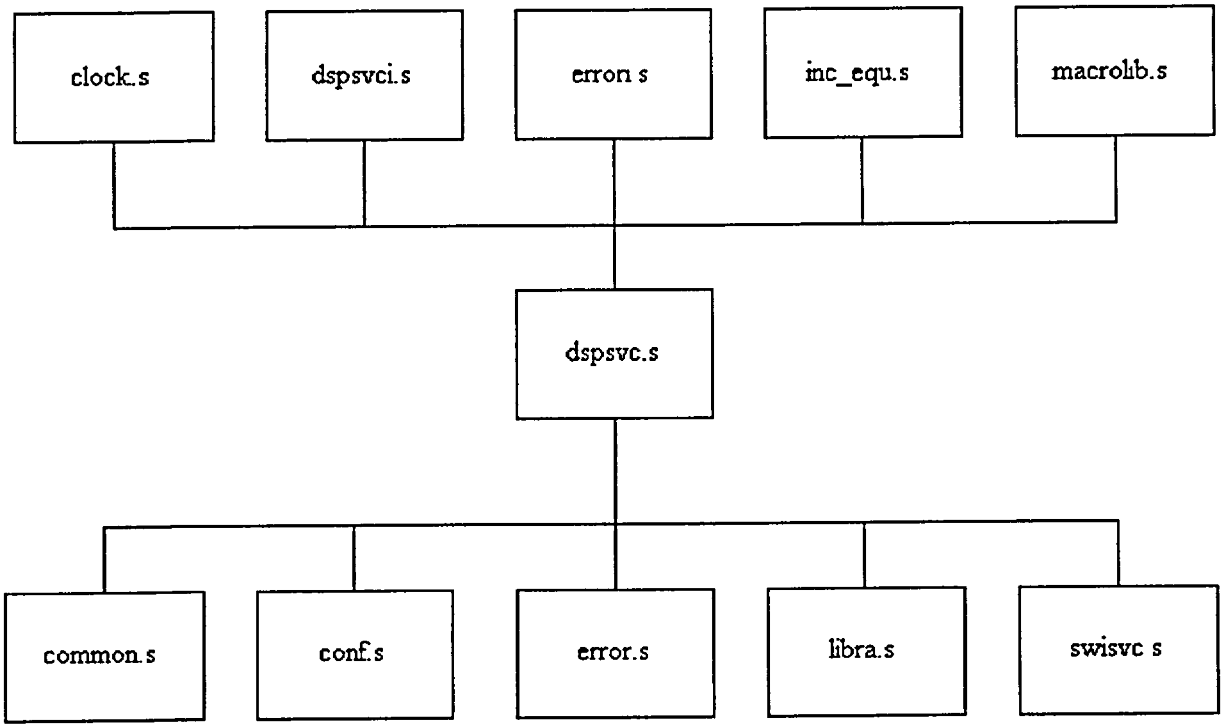


alrsvc.s
anaout.s
calc.s
calsvc.s
com232i.s
com485.s
comsub.s
confi s
dspsvc.s
eesvc.s
error.s
host485.s
idspsvc.s
setedt s
conf.s

4.1.13. Dbasub.s

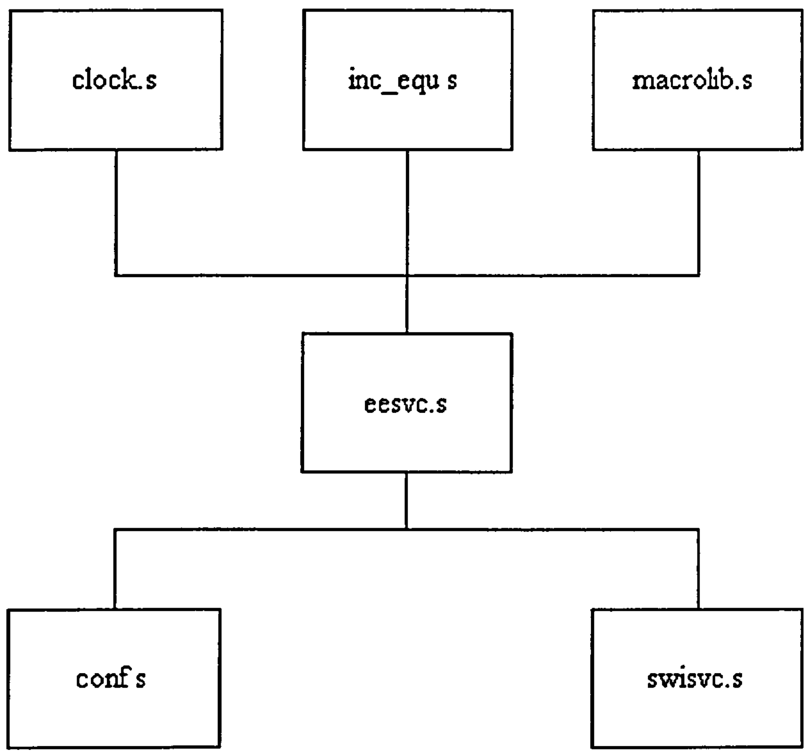


4.1.14. Dspsvc.s

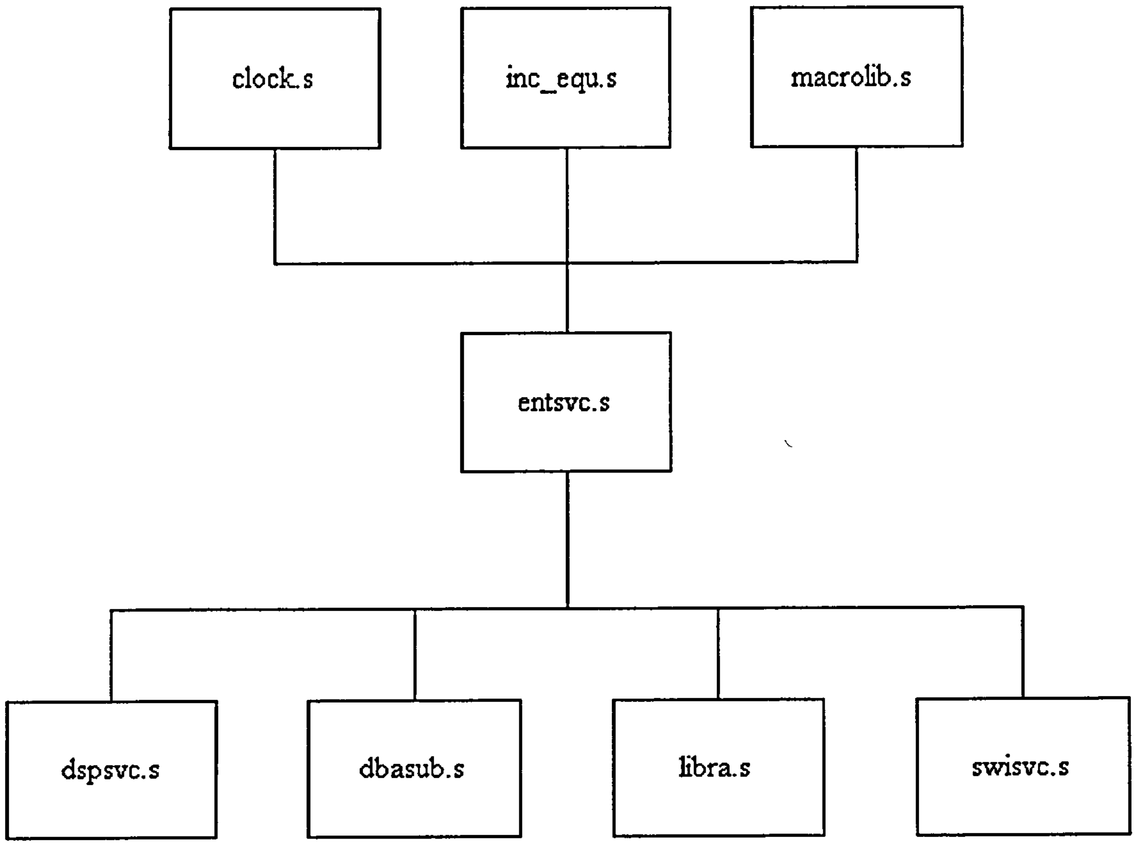


SYNCOR RADIATION MANAGEMENT			DATE 10/23/02		TITLE SOFTWARE DESIGN DESCRIPTION, 9405603	
REV 2	ECN NO. 3076	RELEASED FOR PRODUCTION	DOC CTRL	SHEET 30 of 189	NO. 94095603SDD	SIZE PLN

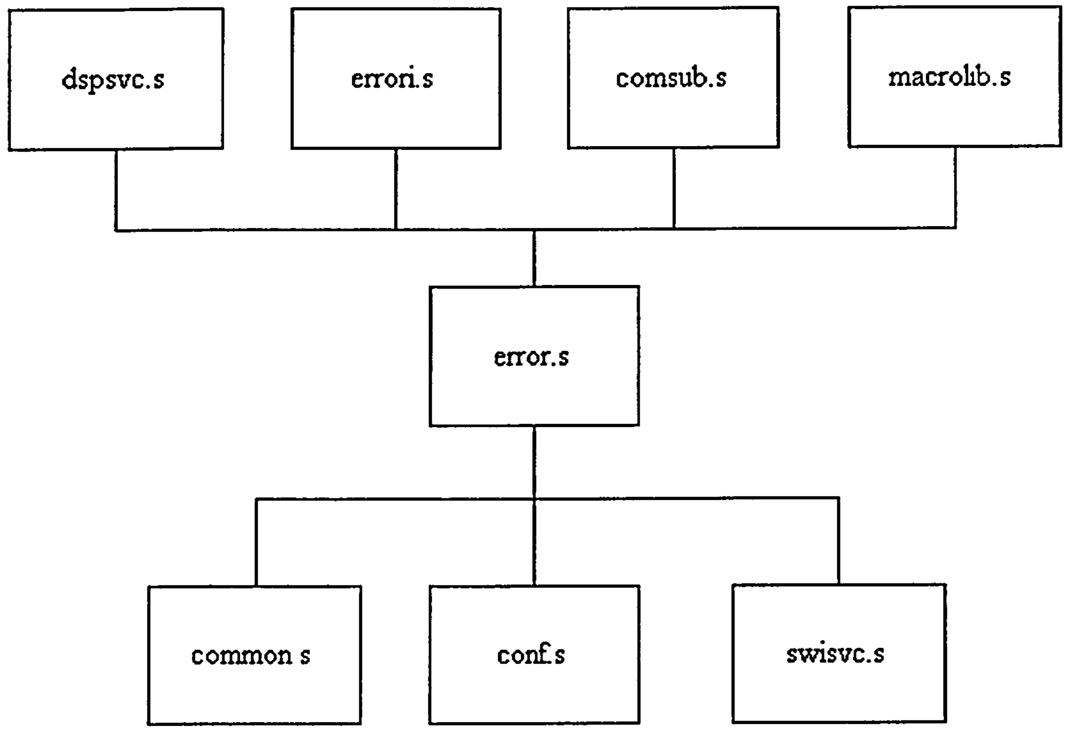
4.1.15. Eesvc.s



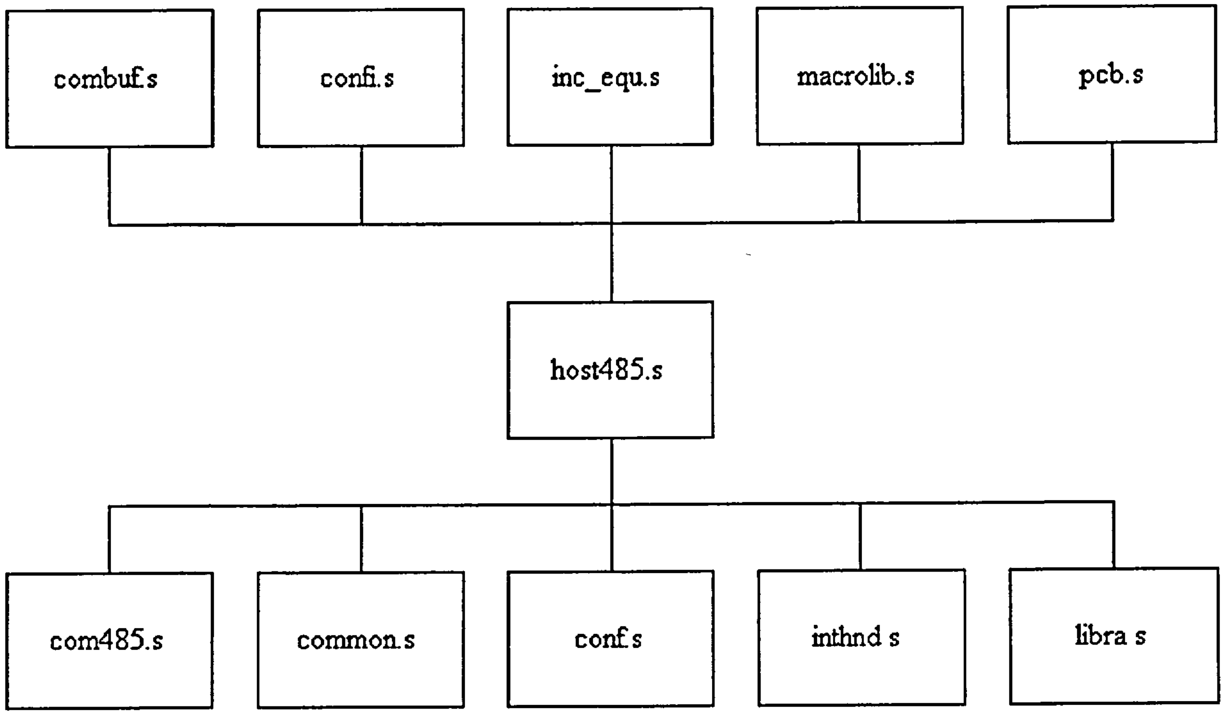
4.1.16. Entsvc.s



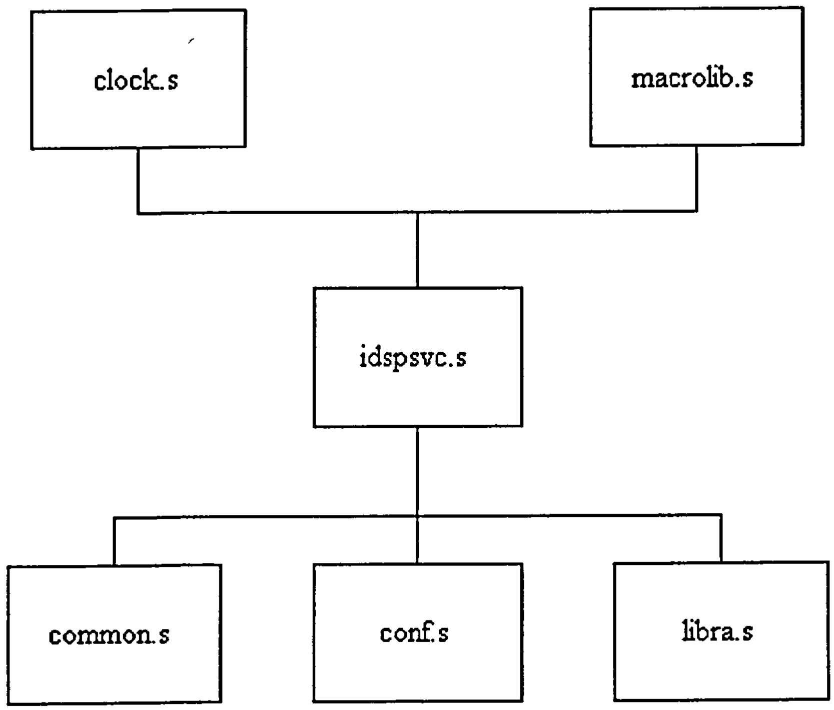
4.1.17. Error.s


dspsvc.s
errori.s
comsub.s
macrolib.s
error.s
common s
conf.s
swisvc.s

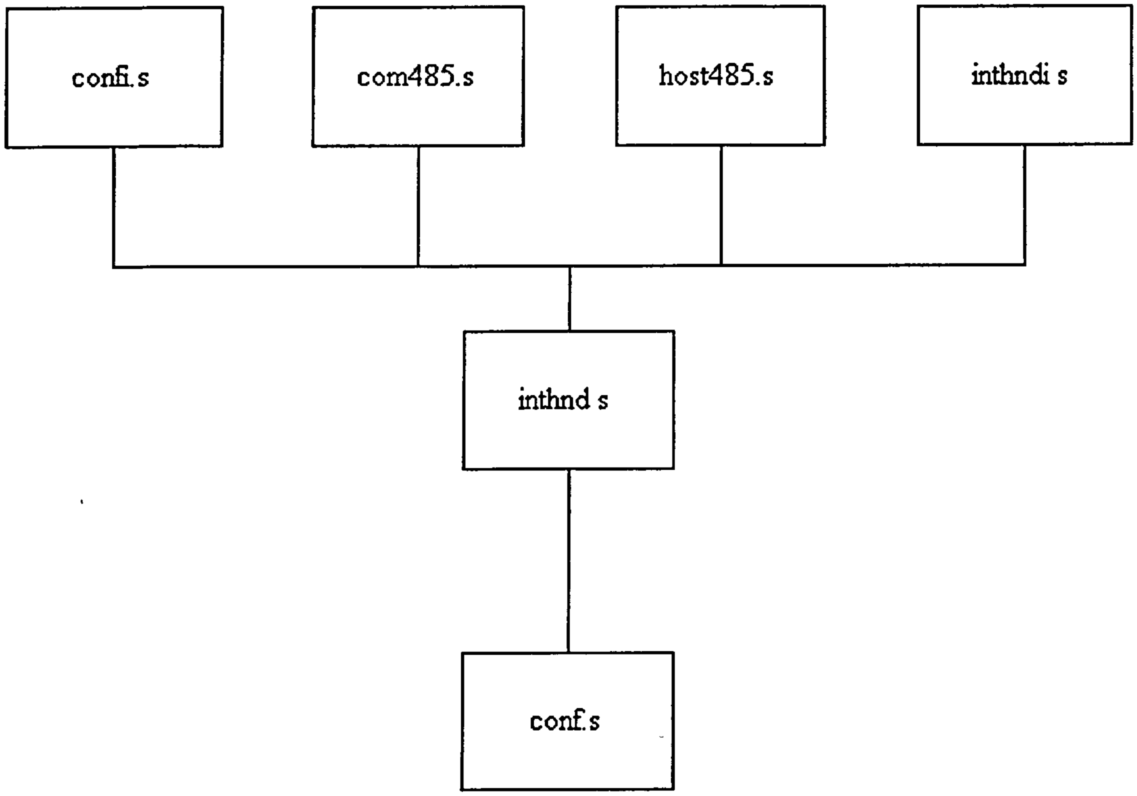
4.1.18. Host485.s



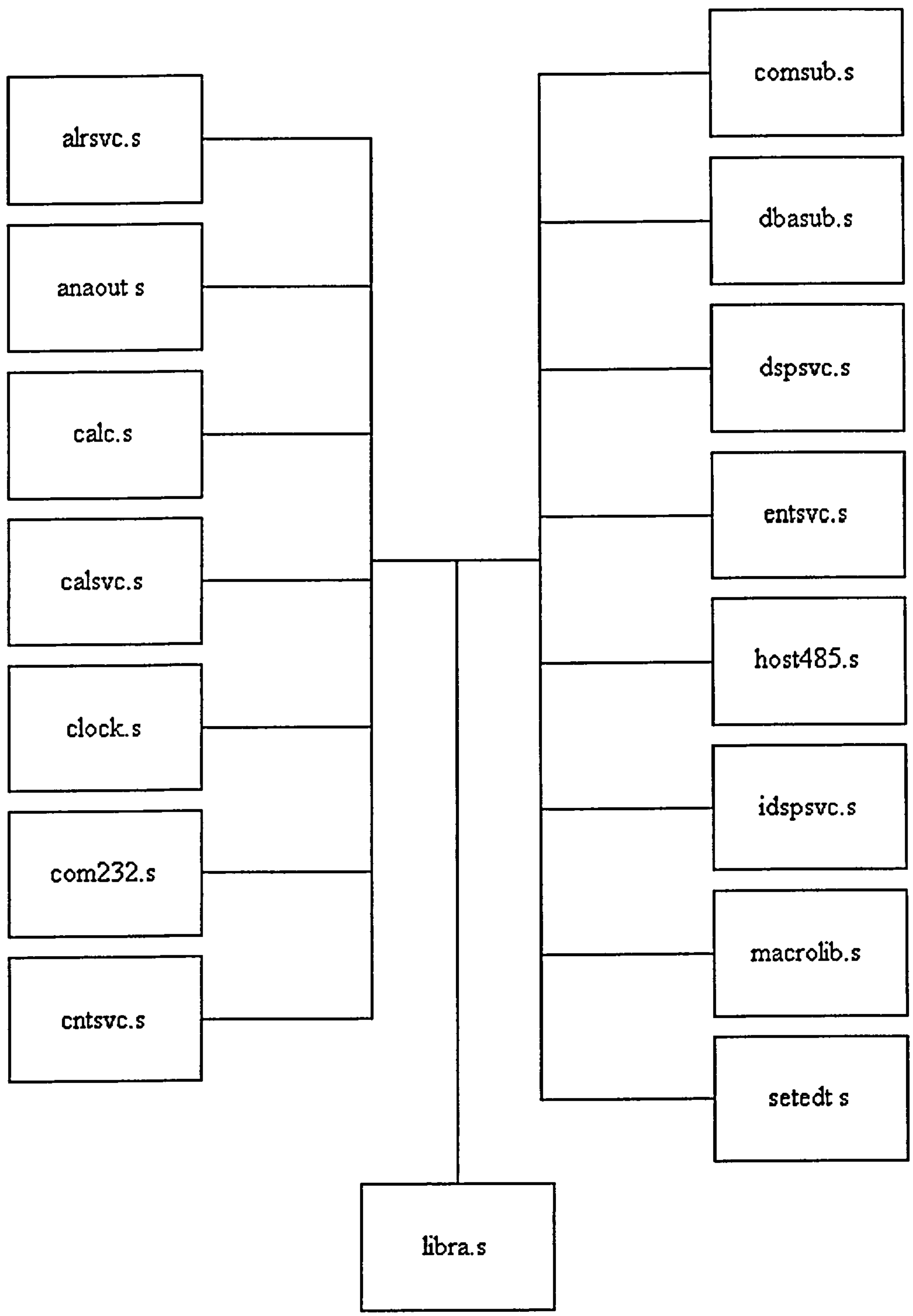
4.1.19. Idspsvc.s



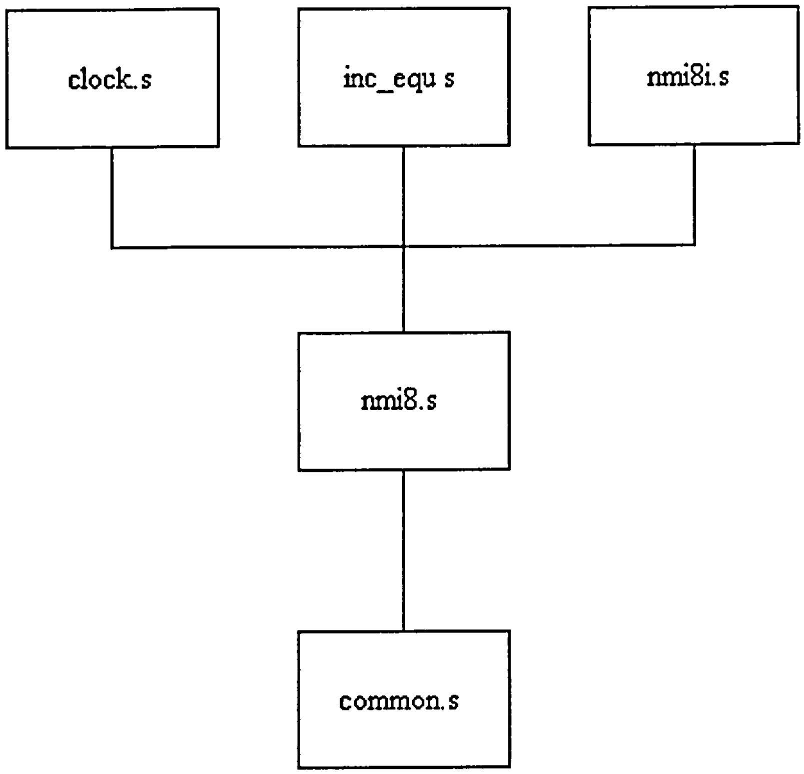
4.1.20. Inthnd.s



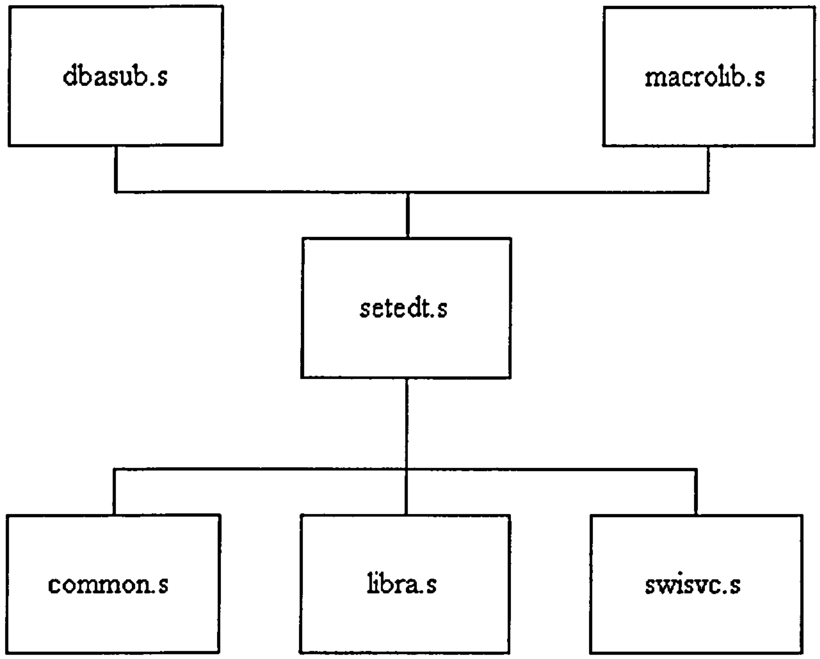
4.1.21. Libra.s


alrsvc.s
anaout s
calc.s
calsvc.s
clock.s
com232.s
cntsvc.s
comsub.s
dbasub.s
dspsvc.s
entsvc.s
host485.s
idspsvc.s
macrolib.s
setedt s
libra.s

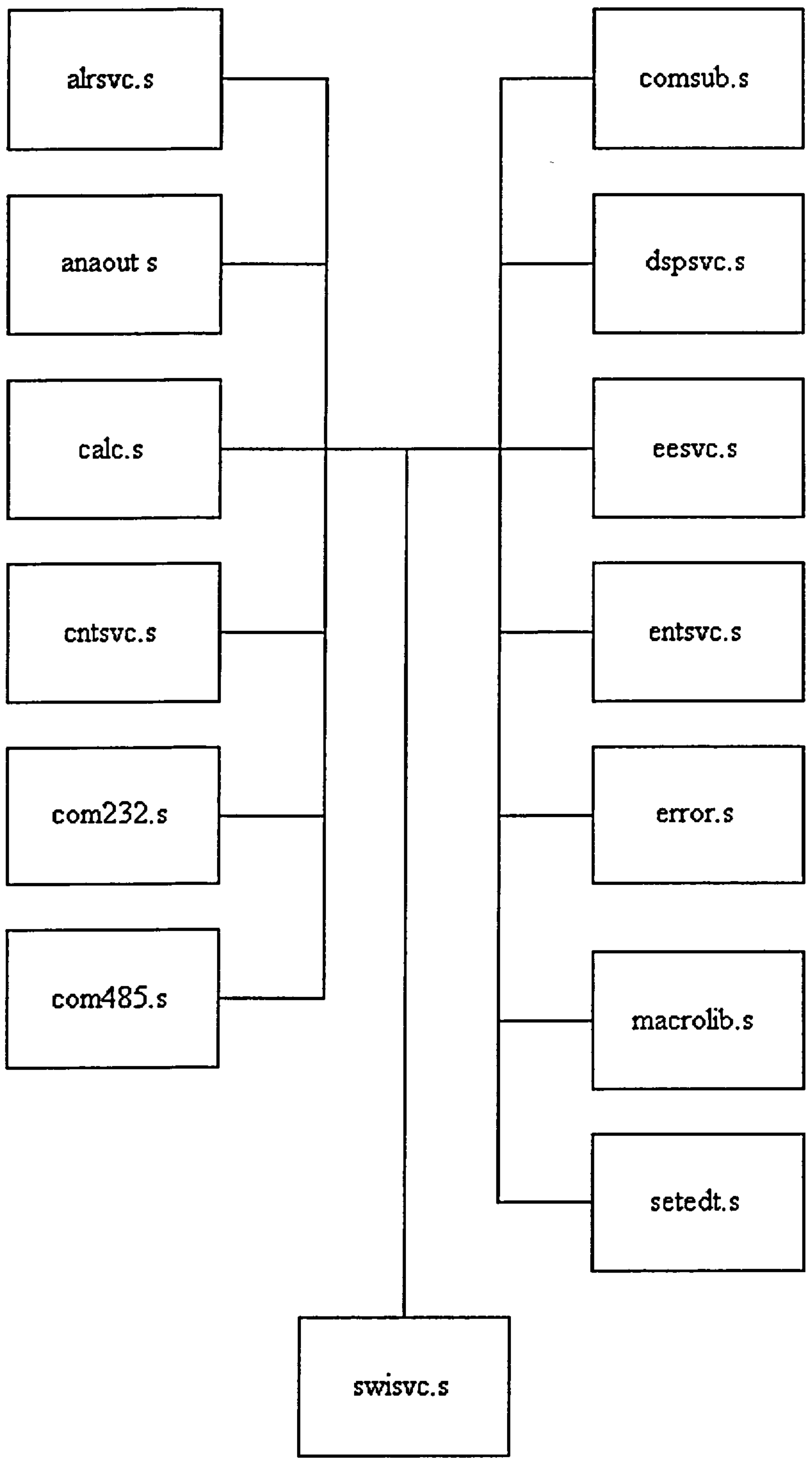
4.1.22. Nmi8.s



4.1.23. Setedt.s



4.1.24. Swisvc.s



4.2. Interprocess Dependencies
See Appendix A for drawings.

4.2.1. Reset Circuitry
The reset circuit generates a 650 ms wide low pulse to the MPU reset input and various external registers. The R1/C1 network generates a delayed trigger pulse to the U10 multivibrator. Upon power up, C1 charges through R1. When approximately 1.4 volts is reached, U11-8 goes low, triggering U10. U10-4 returns high and U10-13 returns low. The low to high transition on U10-4 signals the MPU to begin a reset sequence.

4.2.2. Clocks
The system clock is generated by the MPU using a 4 MHz crystal, CR1. The system clock, from which others are derived, is an output on the MPU pin 37 and operates at 1MHz.

4.2.3. NMI Clock
The NMI clock is generated by U30, U31, and U32 , which are dual decade counters. The 1 MHz system clock is applied to the U30-1 input. U30 is a divide by 100 counter, while U31 is a divide by 50 counter, with respect to the input frequency. U30-9's output is 10kHz and U31-9's output is 200 Hz. U32 is connected for operation as a divide by 25 counter, which produces an 8 Hz output on U32-9.

4.2.4. Write Cycle Clock
The Write Cycle Clock is generated by U19. The 1 MHz system clock is applied to the U19-2 input, which is adjusted via VR13 for a –225 ns delay from the falling clock edge. The second stage of U19 produces a 225 ns output pulse width.

4.2.5. Address Drivers
Line drivers U12 (low order addresses and U13 (high order addresses) provide signal buffering and capability to drive 15 TTL's unit loads for the address bus. The output drives are all internal devices utilizing address signals on the main circuit board as well as the J3 option interface bus connector for additional circuit boards. PROM 8000 directly drives the enable pins on the PROM. RAM 0000, RAM 2000, and RAM Read are logic OR'd with Clock from U17. RAM Read drives the RAM output. Enable pins RAM 0000 and RAM 2000 act as chip enables for the appropriate RAM. RAM Write is logic OR'd with Write 02 U17, which is connected to the write enable pins on the rams. The 5000 Block output signal is applied to driver U74 and connect to J3, the optional connector.

4.2.6. PROM
This is typically a 27256 which is a UV erasable 32K x 8 bit PROM. U23, which responds to address 8000-FFFF, is always present. U23 contains the operating program for the UDR (firmware).

4.2.7. RAM

U21 is utilized for temporary data storage. U21, which responds to address 0000-1FFF (8K x 8 bit), is always present. Data stored in the RAM is lost on power down.

4.2.8. EEPROM

Sixteen (16) monitor specific, operator entered set points are stored in 64 bytes of non-volatile electrically erasable memory. U33 provides storage for the set points (256 bytes max.). U35 is an 8-bit control register.

4.2.9. Data Transceivers

Data Transceiver U16 is an octal tri-state bi-directional transceiver which provides drive capability to the data bus.
Data transceiver U73 provides buffer and drive capability to the external data bus interface, available for optional circuit boards on J3.

4.2.10. Control Signal Buffer

Line driver U18 provides a signal drive for all system control signals and clocks utilized by circuitry within the main circuit board.
Line driver U74 provides the drive for control lines and clock signals for external circuitry utilizing the J3 option interface connector.

4.2.11. Address Decoding

The master decoder (U14) is an open-collector 32 x 8 bit bipolar PROM. Address lines A15, A14, A13, and A12 as well as read/write are used to decode memory and I/O addresses in 4K hex blocks.

4.2.12. Relay Control Register (Write Only)

The Relay Control Register (U44) is an 8 bit register with clear, and responds to address 400C. Upon initial power-up, the system reset signal sets all outputs low. Data written into U44 remains at the outputs until a reset occurs or new data is written. With the exception of the fail bit (D0), all outputs are applied to U48, which is an inverting open collector driver. The outputs of U48, including fail, drive (via the J2 connector) mechanical relays located on the relay board. The fail bit is used as an input to U47-3, which, when set high-low-high once per second, causes output U47-6 to remain low. Should this high-low-high sequence fail to occur (under MPU control), U47 will time out and set U47-6 high, causing the fail indicator on the panel and fail relay to de-activate. U48 also drives the remote indicator (when used) on the front panel.

4.2.13. Bar graph (Write Only)

The bar graph addresses are decoded by U5, which is a 1 of 8 decoder. Control line inputs to U5 are R/W, Write 02, and Bar graph. The bar graph contains 24 segments, with 3 segments allotted to each decade. Each of the three segments illuminates at approximately 25, 50, and 75% of each decade.

4.2.14. Read-Write Cycles

A read cycle is performed by sequencing RAM 0000 and RAM Read, while RAM Write is held high (inactive). The address (A0 – A12) are latched by the falling edge of RAM 0000. Data becomes valid approximately 250 ns later.

A write cycle is performed by sequencing RAM 000 and RAM Write, while holding RAM Read high (inactive). Identical to the read cycle, the address (A0 – A12) are latched by the falling edge of RAM 000. Data is strobed into RAM on falling edge of RAM Write.

4.2.15. Write Register Decoding

Decoding for write registers within the main circuit board is performed by U2, which is a 1 of 8 decoder. Control signals for U2 are Reg. Select, R/W (active high), Write 02 as well as addresses A1, A2, and A3. U2 decodes two addresses per output, starting at 4000, and ending with 400E. These outputs are active low.

4.2.16. Counter Control (Write Only)

The counter control register (U43) is an 8 bit register with clear, and responds to address 400A. Upon initial power-up, the system resets all outputs low. Data written into U43 remains at the outputs until a reset occurs or new data is written.

4.2.17. Display Control (Write Only)

Registers U71 and U72 are used to control and display data on the front panel 7-segment displays. Five digits are used along with two spare digit drive signals. The display control register (U71) is an 8-bit register, utilizing four data bits (D0-D3). The display data register (U72) is also an 8-bit register utilizing five data bits (D0-D4). Both U71 and U72 outputs are reset (low) upon initial power-up.

U71 is used to select the digit to be written as well as to set the WRITE bit input to U75, the display controller. U72 is used to enter the data to be written and a decimal point for the selected digit.

U75 is a universal eight digit 7-segment LED driver controller used with common anode devices. Address inputs (A0-A2), supplied by U71, are used to select the digit.

Data inputs D0-D3 and the decimal point, supplied by U72, are used to enter data in the selected digit.

The display controller contains all necessary circuitry including address decoding, static RAM, and multiplex oscillator for interdigit blanking.

4.2.18. Status Indicators (Write Only)

The status register (U60) is an 8-bit register with clear, and responds to address 4000. Data written into U60 remains at the outputs until a reset occurs or new data is written. Upon initial power-up, the system reset signal sets all outputs low.

U60 outputs, when high, control U61 inverter/driver to activate the appropriate front panel status indicators. U60 outputs, when low, control U61 to deactivate the appropriate front panel status indicators.

4.2.19. Data Entry (Read Only)

Octal buffer U36 functions as an interface to supply the status of the switches for data entry to the internal data bus address (4002).

4.2.20. Gross Counter (Read Only)

The gross counter register. Octal buffer/drivers U40 (4004) and U41 (4006) interface the low and high counter bytes to the data bus from U45 an U46 respectively. Dual module 16 counters (U45, U46) accumulate counts from the signal processing circuitry over a program controlled sample period and make this data available to the MPU. A high level on U45 pins 2 and 12, and U46 pins 2 and 12 cause the counters to clear in anticipation of a sample being initiated. A low level activates the counters to accept pulses from the signal processing circuitry. Maximum count for a sample period is 32,768. When this count is achieved, U46 pin 8 goes high which causes the pulse counting to stop. This condition is recognized by the MPU as an indication of counter overflow.

4.2.21. Digital to Analog – Converter and Output Circuitry (Write Only)

The D/A converter (U82) is an 8-bit buffered multiplying device which responds to address 4004. Data is written and latched by U82 when CS and WR are active (low). The converter is configured for unipolar operation with a voltage reference of + 10VDC.

U81 operation amplifier 1 buffers the converter output to drive the three analog output circuits. This voltage is also provided to J4, analog option connector. J4 is provided with the necessary supply voltages to configure a custom analog output range for special applications. U81 operational amplifier 3 provides the user with a 0 – 10 volt analog output. VR1 is the zero adjustment and VR2 is the gain adjustment for the 0 – 10 volts output.

The circuitry comprised of U80 operational amplifiers 1 and 2, as well as U81 operational amplifier 2, provides a 4 – 20 mA reading on the output. U81 operational amplifier 2 and Q1 are configured as a constant current source controlled by the output of U80 operational amplifier 1. The positive feedback circuitry, comprised of R41 and R43, ensures that the output current will remain constant regardless of the output load impedance. The maximum load impedance is 500 ohms.

The circuitry comprised of U80 operational amplifiers 3 and 4 as well as U81 operational amplifier 4 provides a second 4 – 20 mA user output.

VR6 is adjusted to obtain a 4 mA reading at the output and VR5 is adjusted to obtain a 20 mA reading at the output.

4.2.22. Read Register Decoding

Decoding for READ registers within the main circuit board is performed by U34, which is a 1 of 8 decoder. Control signals for U34 are REG SELECT, and R/W (active high) as well as address A1, A2, and A3. U34 decodes address per output, starting with 4000 and ending 400E. These outputs are active low.

4.2.23. Option Board Bus

The option board bus is available on connector J3. All address, data and control signals are provided to allow various digital/analog circuit boards to directly interface to the main circuit board. Decoded signals for asynchronous communications interface adapter option, and the general purpose interface bus option are available on the option board interface connector.

4.2.24. Signal Input Circuitry

Detector input pulse circuitry consists of an input buffer, high/low discriminators, signal detection, anti-jam and signal multiplexer circuits.

4.2.25. Buffer Amplifier

The detector input is connected to J6. The input impedance is 50 ohms to match the signal cable and the detector's output impedance. Jumpers JP4 and JP5 are used to select the proper pulse polarity.

The detector input signal (with appropriate polarity jumpers installed) is applied to unity gain buffer amplifier U90. VR8 is used to fine adjust for unity gain. Regardless of input signal polarity, U90-6 outputs positive going pulses. VR9 is a zero offset adjust for U90. The buffer amplifier output is provided to the J7 connector (for use by analyzer option circuitry) as well as the high and low discriminators. TP-Pulse is available as a test.

4.2.26. Discriminators

The low level discriminator is comprised of comparator U91 device 2 and associated circuitry. VR11 is used to set the trip threshold. The adjustment range is 50 mV to 1 volt, which can be measured at the low discriminator test jack. As the positive pulse, applied to the input, passes through the trip threshold, the output (U91-6) is forced low. When the pulse returns through the trip threshold, the output U91-6 returns high and ready to accept another input pulse. Pulses below the trip threshold do not trigger the output.

The high discriminator is comprised of comparator U91 device 1 and associated circuitry. VR10 is used to set the trip point. The adjustment range is 3.5 to 7.5 volts, which can be measured at the high discriminator test jack. As the positive pulse, applied to the input, passes through the trip threshold, the output (U91 device 1) is forced low. When the pulse returns through the trip threshold, the output (U91 device 1) return high and is ready to accept another input pulse. Pulses below the trip threshold do not trigger the output.

4.2.27. Option Jumper Select (Read Only)

Octal buffer U42 functions as an interface to provide the status of the option jumpers for sensitivity selection, alarm reset operation, alarm operation in check source, fail operation, anti-jam bit data, and serial data from EEPROM to the internal data bus and MPU. U42 responds to address 4008. The sensitivity read functions are in the table below.

Data Bit		Function
D0		Serial data from EEPROM
D1		Anti-Jam
D2		(not used)
D3	JP3-5	(IN) Inhibit alarms during check source operation (factory setting)
		(OUT) Alarms active during check source operation
D4	JP3-4	(IN) Fails in five minutes with no count (factory setting)
		(OUT) Does not fail
D5	JP3-3	(IN) Manual reset of alarms (factory setting)
		(OUT) Auto alarm acknowledge, after counts return to normal

JP3 jumper IN – MPU reads a low (0)
JP3 jumper OUT – MPU reads a high (1)

4.2.28. Switch Inputs

Octal inverting buffer U62 functions as an interface to provide the status of the front panel control switches to the internal data bus and MPU. U62 responds to address 4000.

When a switch is pressed, the appropriate input to U62 is pulled low. When U62 is read by the MPU, a high (1) is available on the data bus. When no switches have been pressed, all output (U62) will be low when read. U63 is a latch which latches switch data from the check source and alarm acknowledge switches. The MPU controls the clear switch latch signal to reset U63. The circuit comprised by S1, D1, and R11 is a remote alarm acknowledge. R11 is selected to allow a 20 mA signal to flow through the S1 infrared diode when a given voltage is present on the J9 remote acknowledge input. When this voltage is present, the S1 infrared diode is forward biased, causing the S1 phototransistor to conduct. This effectively forces a low (0) to U63-10, setting the alarm acknowledge bit.

4.2.29. Signal Detection

The circuitry comprised of flip-flop U93 and inverters U11 and U94, utilizes the low discriminator and high discriminator outputs to ensure that only input signals which peak between the discriminators are made available to the gross counters.

When U93-2 counter enable is brought high, under software control to initiate a sample period, and the low discriminator threshold is exceeded, a positive going edge on U93-3 clocks a high into flip-flop U93-5. Assuming the high discriminator has not bee exceeded, when the input pulse returns through the low discriminator

threshold, a positive going edge on U93-11 clocks the high on U93-12 to the output U93-9.
A delayed positive pulse produced by the R/C network (R80/C35) on U93-13 allows the flip-flop U93-9 to remain high until the delayed pulse onU93-13 returns low, which resets the output U93-9 low. U93-9 is connected to the signal multiplexer circuitry.
When an input pulse exceeds both the high and low discriminators , the high (U93-5) clocked by the transition through the low discriminator is reset by the low (U91-1) resulting from the transition through the high discriminator. This action causes no pulse to be generated at U93-9.

4.2.30. Signal Multiplexer

The signal multiplexer comprised of U101, U102, and U94 allows the MPU to select wither the radiation pulse or the frequency output representing the high voltage to be input to the gross counters. When counter enable is active (high), the signal detection circuit output (representing radiation) is routed to the gross counters. When HV select is active (high), the HV frequency is routed to the gross counters. The outputs connected to pull-up resistor R81, are open collectors allowing the most significant bit of the counters to force this node low, effectively terminating the pulse input to the counters and indicating an overflow condition.

4.2.31. Anti-Jam Circuitry

The anti-jam circuitry allows for the detection of rapid increase in pulses (due to a rapid increase in radiation at the detector) and provides a bit to the sensitivity select register. A detector will reach a point, in a very high radiation field, when it will no longer provide pulses, but conducts continuously. The absence of pulses would normally indicate a low radiation field, when in actuality this is not the case. The purpose of the anti-jam circuit is to detect that this situation is about to occur, and to indicate it to the MPU. The MPU will then shut down the high voltage.
The input to the anti-jam circuit is provided by the low discriminator output (U91-6). JP-7 selects detector type, 1-2 for scintillation detectors and 2-3 for GM type. Q3 turns ON/OFF with input pulses, which allows C39 to charge to an average DC level. VR12 (adjustment range 0 to 1.6 volts) is used to adjust the trip threshold on comparator U92-2. When the repetition rate of the input pulse causes C39 to charge and the DC level to exceed the threshold, comparator output U92-1 (low in normal operation) is forced high. When this occurs, U96-2 goes high (U96-1 is high after power-up) U93-3 goes low and U96-4 goes high. Diode D9 effectively latches this circuit in the jam mode. That is, if C39 discharges (due to absence of input pulses) and U92-1 goes low, D9 becomes forward biased which holds U92-2 high. The high, now on U96-4 causes Q4 to turn on driving Q5 on, forcing U96-8 & 9 node to ground. In normal operation, JP6 is in position 1-2 allowing high current flow through F1 (1/20 Amp fuse) causing it to blow. R82 will now hold U96-8 & 9 node to ground, causing U96-10 (anti-jam bit) to be active (high). At this point, normal operation can only be achieved by replacing fuse F1. Jumper JP6 – position 2-3 is for test purposes only and allows fuse F1 to be removed from the circuit and R79 provides pull-up to + 5 volts. In this mode, cycling of power resets the anti-jam circuit. R93/C41 on U96-1 provides a delay from power up to inhibit false tripping of the anti-jam circuit. The anti-jam set point is based on the maximum counting range of the detector, and is set at approximately 40,000Hz.

In the event the monitored radiation increases above the calibrated range of the detector, but below the anti-jam threshold, provisions are included in the firmware to indicate an OVERRANGE condition by displaying EEEEE on the display.

The OVERRANGE set point is based on the specific type og Geiger-Mueller tube detector used for each range. The G-M detector used for each range exhibit different response characteristics, in terms of the pulses, or counts, provided per mR/h. The OVERRANGE set point is equal to the maximum calibrated range for each type of detector. The set points for each range are as follows:

Range, (Max range is equal to OVERRANGE set point), mR/h:	Typical Conversion Constant, mR/h/CPM:	Equivalent Trip Frequency, Hz:
1.00E-2 to 1.00E+3	1.00E-3	16,667
1.00E-1 to 1.00E+4	1.00E-2	16,667
1.00E0 to 1.00E+5	1.00E-1	16,667

4.2.32. High Voltage Supply

The high voltage is utilized by a GM detector (typical range 500 volts to 650 volts). The adjustment range of the HV supply is 300 Vdc to 1800 Vdc. The HV output is short circuit proof in that it will current limit the oscillator section within ten seconds of the output being shorted.

R5 and associated circuitry provide the DC voltage adjustment to U1 device 3. The output U1-8 will vary under control of R5.

Short circuit protection is provided by the PTC thermister. The PTC resistance in normal operation is nominally 5 ohms. When the high voltage output is shorted, the control circuitry U1 device 1 attempts to maintain regulation by increasing the base drive for transistor Q1. Excessive current flows through the PTC, causing the internal temperature to increase. As the temperature increases, the PTC resistance also increases dramatically. The effect is that the control voltage to the oscillator is decreased to a minimum level. The response of the PTC is approximately ten seconds. Removal of the short circuit condition results in restoration of the high voltage to the preset level.

4.2.33. Relay Circuit Board

The relay circuit board contains five independently controlled mechanical relays. Each relay provides two Form C sets of contacts with the exception of the check source and High alarm relays, which provide a single Form C set of contacts for customer use. Interconnection is from J2 on the relay board to J2 on the main circuit board. The control signals (active low) and +15 volts common are provided. The relays perform the following functions: Spare, Check Source, Fail, Warn, and High Alarm. The relay contacts are provided to the user via rear panel connector P1. Varistors (V1-V16) provide transient protection across the contacts.

4.2.34. Front Panel Circuit Board

The front panel circuit board consists of the 7 segment display, backlights for engineering units, status indicators, switches, and bargraph assembly. The front panel interfaces to the main circuit board via interconnecting row 100, 200, and 300. The main power switch also mounts to the front panel circuit board.

4.2.35. Power Supply

The UDR power supply is rated at +5 volts @ 3 amps, +15 volts @2.0 amps, and –15 volts @ 0.5 amps. The input is user selectable at 115 Vac (92 to 132 Vac) or 230 Vac (180 to 264 Vac). The power supply is designed to meet safety requirements UL/CSA/VDE. EMI emissions comply with FCC/Class B requirements. The AC input to the power supply may range from 90 to 204 Vdc @ 50.60 Hz. The 956A must be configured for use at 125 Vac, 50/60 Hz as a factory option.
The power supply provides all internal UDR voltages as well as detector supply voltages. All outputs are protected with automatic recovery upon removal or short circuit condition.

4.2.36. Optional Circuit Boards

Option circuit boards are installed into the 50 pin J3 connector available on the main circuit board. As many as three option boards may be stacked into the J3 bus. The analog input, RS232 communications, and Single channel analyzer option boards must be configured into the code prior to assembly. The 94095603 PROM does not support the analog input and Single channel analyzer option boards.

4.2.37. Option Jumpers

Jumper	Function	Position	Operation
JP1	Microprocessor Reset	Out	Normal Operation (Factory setting)
		In	Not Applicable
JP2	PROM Type	1-2	PROM 27128
		2-3	PROM 27256 (Factory setting)
JP3-1 / JP3-2	Statistical Accuracy	Out/Out	2% Accuracy (9604 counts)
		Out/in	5% Accuracy (1537 counts)
		In/Out	10% Accuracy (384 counts)
		In/In	Fixed one second display update (Factory setting)
JP-3-3	Alarm Acknowledge	In	Manual Acknowledge (Factory setting)
		Out	Automatic
JP3-4	Fail Alarm	In	Enable no counts fail alarm (Factory setting)
		Out	Inhibit no counts fail alarm
JP3-5	Check Source Alarm	In	Alarm Inhibited (factory setting)
		Out	Alarm Enabled
JP4	Input Pulse Selection	1-2	Negative input pulse
		2-3	Positive input pulse – GM Detectors (Factory setting)
JP5	Shield Polarity Selection	1-2	Shield for negative pulse
		2-3	Shield for positive pulse (GM) (Factory setting)
JP6	Anti-Jam Fuse Selection	1-2	Enable for normal operation (Factory setting)
		2-3	Anti-Jam circuit fuse bypassed (testing)
JP7	Detector Type for Anti-Jam circuit timing	1-2	Scintillation
		2-3	GM Tube (Factory setting)
		Out	Anti-Jam circuit disabled (testing)

4.3. Data Dependencies

Data is entered via pushbuttons on front panel, a rotary function switch and data entry pushbuttons. It is then converted to the appropriate format.

5. Interface Description

This section describes the interface that each architectural component provides.

5.1. Module Interface

5.1.1. Include modules:

These modules define variables or constants that are used in more than one source module or throughout a source module.

5.1.1.1. Acia.s – included in:
- 5.1.1.1.1. Com485.s

5.1.1.2. Alrsvci.s – Included in:
- 5.1.1.2.1. Alrsvc.s

5.1.1.3. Calsvci.s – Included in:
- 5.1.1.3.1. Calsvc.s

5.1.1.4. Com232i.s – Included in:
- 5.1.1.4.1. Com232.s
- 5.1.1.4.2. Conf.s

5.1.1.5. Com485i.s – included in:
- 5.1.1.5.1. Com485.s

5.1.1.6. Combuf.s – Included in:
- 5.1.1.6.1. Com485.s
- 5.1.1.6.2. Host485.s

5.1.1.7. Commoni.s – Included in:
- 5.1.1.7.1. Common.s

5.1.1.8. Comsubi.s - Included in:
- 5.1.1.8.1. Comsub.s

5.1.1.9. Confi.s - Included in:
- 5.1.1.9.1. Clock.s
- 5.1.1.9.2. Conf.s
- 5.1.1.9.3. Host485.s
- 5.1.1.9.4. Inthnd.s

5.1.1.10. Dspsvci.s – Included in:
- 5.1.1.10.1. Dspsvc.s

5.1.1.11. Errori.s – Included in:
- 5.1.1.11.1. Dspsvc.s
- 5.1.1.11.2. Error.s

5.1.1.12. Inc_equ.s –Included in:
- 5.1.1.12.1. Alrsvc.s
- 5.1.1.12.2. Anaout.s
- 5.1.1.12.3. Calc.s
- 5.1.1.12.4. Calsvc.s
- 5.1.1.12.5. Chksvc.s
- 5.1.1.12.6. Clock.s
- 5.1.1.12.7. Cntsvc.s
- 5.1.1.12.8. Comsub.s
- 5.1.1.12.9. Dbasub.s
- 5.1.1.12.10. Dspsvc.s
- 5.1.1.12.11. Eesvc.s
- 5.1.1.12.12. Entsvc.s

5.1.1.12.13. Host485.s
5.1.1.12.14. Nmi8.s
5.1.1.13. Inthndi.s – Included in:
5.1.1.13.1. Inthnd.s
5.1.1.14. Macrolib.s –Included in:
5.1.1.14.1. Alrsvc.s
5.1.1.14.2. Anaout.s
5.1.1.14.3. Calc.s
5.1.1.14.4. Calsvc.s
5.1.1.14.5. Clock.c
5.1.1.14.6. Cntsvc.s
5.1.1.14.7. Com232.s
5.1.1.14.8. Comsub.s
5.1.1.14.9. Dbasub.s
5.1.1.14.10. Dspsvc.s
5.1.1.14.11. Eesvc.s
5.1.1.14.12. Entsvc.s
5.1.1.14.13. Error.s
5.1.1.14.14. Host485.s
5.1.1.14.15. Idspsvc.s
5.1.1.14.16. Libra.s
5.1.1.14.17. Setedt.s
5.1.1.14.18. Swisvc.s
5.1.1.15. Nmi8i.s – Included in:
5.1.1.15.1. Nmi8.s
5.1.1.16. Pcb.s – Included in:
5.1.1.16.1. Com485.s
5.1.1.16.2. Host485.s

5.1.2. Source Modules:
Modules made up of assemble code that enables the UDR to perform all the required functions.
5.1.2.1. Alrsv.s, Alarm Service – called in main loop
5.1.2.2. Anaout.s, Analog output – called in main loop
5.1.2.3. Calc.s, Display Calculation - called in main loop
5.1.2.4. Calsvc.s, Calibration Services – called in main loop
5.1.2.5. Chksvc.s, Checksource Services – called in main loop
5.1.2.6., Clock.s, Scheduling services – this is the main loop
5.1.2.7. Cntsvc.s, Deadtime Correction – called in main loop
5.1.2.8. Com232.s, RS232 Communication – called in main loop
5.1.2.9. Com485.s, RS485 Communication – called in main loop
5.1.2.10. Dbasub.s, Setpoint services – called in main loop
5.1.2.11. Dspsvc.s, Display Services – called in main loop
5.1.2.12. Eesvc.s, EEPROM Services – called in main loop
5.1.2.13. Entsvc.s, Service data entry buttons – called in main loop
5.1.2.14. Host485.s, Host Message Replay Services - called by the interrupt service routine

5.1.2.15. Intdsp.s; Display Conversion – called in main loop
5.1.2.16. Inthnd.s, Interrupt Handler – called in main loop

5.1.3 Support Modules

5.1.3.1 Common.s – This modules contains any variable or table that is either used by more than one module or is dependent on the monitor's configuration.
5.1.3.2 Comsub.s – The ACIA buffer is queued in the COM232 module and pulled in here to execute the command and store the response in the ACIA buffer.
5.1.3.3 Conf.s - The UDR MONSTAT option equates.
5.1.3.4 Error.s –
5.1.3.5 Libra.s – Subroutine library used by several functions.
5.1.3.6 Nmi8.s – 8 Hz interrupt service routine that is connected to a clock which ticks 4 times per second.
5.1.3.7 Setedt.s – Counts routines used with data entry button services.
5.1.3.8 Swisvc.s – Supervisory services – contains routine that do basic functions for several routines.

5.2. Process Interface

The Victoreen Model 956A UDR receives signals from a specific detector. The UDR also receives input from the user via the pushbuttons on the front panel and the data entry rotary switch and pushbuttons. Analog output is also generated for specific events.

5.3. Process description

The Victoreen Model 956A UDR continuously displays radiation levels, indicates alarms, and provides display, control and annunciation functions. The UDR also provides channel calibration and test functionality in combination with the detector.

6. Detailed Design
This section contains the details needed by a programmer prior to implementation.

6.1. Module Detailed Design

6.1.1. Include Modules:
- 6.1.1.1. Acia.s
  - 6.1.1.1.1. Defines offsets for Data, Status, Command and Control Registers.
  - 6.1.1.1.2. Defines status equates
  - 6.1.1.1.3. Defines command equates
- 6.1.1.2. Alrsvci.s – alarm serve equates
  - 6.1.1.2.1. Defines alarm service equates for 5 min. no counts timer and fail bit
- 6.1.1.3. Calsvci.s – calibrate service equates
  - 6.1.1.3.1. Offset into RAM
  - 6.1.1.3.2. Lights mask on/off
  - 6.1.1.3.3. Alarm high relay
  - 6.1.1.3.4. Warn lights
  - 6.1.1.3.5. Fail relay
- 6.1.1.4. Com232i.s – RS232 communication equates
  - 6.1.1.4.1. Offsets for ACIA registers
  - 6.1.1.4.2. ACIA status equates
  - 6.1.1.4.3. ACIA command equates
  - 6.1.1.4.4. ACIA buffer equates
  - 6.1.1.4.5. Dumb terminal equates
- 6.1.1.5. Com485i.s – RS485 Interrupt service equates
  - 6.1.1.5.1. Number of bytes in message types
  - 6.1.1.5.2. Equates for states
  - 6.1.1.5.3. Message error codes
- 6.1.1.6. Combuf.s – Communication buffer structure
  - 6.1.1.6.1. Equates for offset into communication buffer
- 6.1.1.7. Commoni.s – Common equates
  - 6.1.1.7.1. Line feed
  - 6.1.1.7.2. RAM upper bound
  - 6.1.1.7.3. End of text
- 6.1.1.8. Comsubi.s - RS232 Communication commands equates
  - 6.1.1.8.1. Offsets of ACIA register
  - 6.1.1.8.2. ACIA status equates
  - 6.1.1.8.3. ACIA command equates
  - 6.1.1.8.4. ACIA buffer equates
- 6.1.1.9. Confi.s – 956A UDR configuration equates
  - 6.1.1.9.1. Monitor display
  - 6.1.1.9.2. Conversion constant
  - 6.1.1.9.3. Background subtract
  - 6.1.1.9.4. Anti-Jam
  - 6.1.1.9.5. Fail safe
  - 6.1.1.9.6. Detector type
  - 6.1.1.9.7. Analog option board

6.1.1.9.8. RS-232 communication option
6.1.1.9.9. UDR type
6.1.1.9.10. ACIA1
6.1.1.9.11. ACIA2
6.1.1.9.12. SCANRAD option
6.1.1.10. Dspsvci.s – Display services equates
6.1.1.10.1. Defines High byte and Low byte
6.1.1.10.2. Defines Green registers
6.1.1.10.3. Defines Red registers
6.1.1.11. Errori.s – Setpoint Validity Check equates
6.1.1.11.1. Defines error codes
6.1.1.12. Inc_equ.s – Common equates
6.1.1.12.1. Defines JAM and OVERRANGE bits
6.1.1.12.2. Defines Scalar Module Addresses
6.1.1.12.3. Defines Status bits
6.1.1.12.4. Defines Relay control bits
6.1.1.12.5. Defines Counter Control bits
6.1.1.12.6. Defines Switch Input bits
6.1.1.12.7. Defines Setpoint EEPROM Address
6.1.1.12.8. Defines Configuration file equates
6.1.1.12.9. Defines display data equates
6.1.1.12.10. Defines sensitivity select/status
6.1.1.12.11. Defines status word bits
6.1.1.13. Inthndi.s – Interrupt handling equates
6.1.1.13.1. Defines IRQ status bit for polling
6.1.1.14. Macrolib.s – 6800 MACRO library
6.1.1.14.1. Store using x
6.1.1.14.2. Test argument to trigger logic analyzer
6.1.1.14.3. Task name
6.1.1.14.4. Bit set
6.1.1.14.5. Bit clear
6.1.1.14.6. Move X
6.1.1.14.7. Move to X
6.1.1.14.8. Push X
6.1.1.14.9. Pull X
6.1.1.14.10. Add B to X
6.1.1.14.11. Swap X
6.1.1.14.12. Add Constant to X
6.1.1.14.13. Set argument 2 to argument 1
6.1.1.14.14. Add argument 1 to argument 2 and store result in argument 3
6.1.1.14.15. Subtract argument 2 from argument 1 and store result in argument 3
6.1.1.14.16. Multiply argument 1 by argument 2 and store result in argument 3
6.1.1.14.17. Divide argument 2 by argument 1 and store result in argument 3
6.1.1.14.18. Compare argument 1 to argument 2
6.1.1.14.19. Convert floating point value to 32 bit integer
6.1.1.14.20. Converts 32 bit integer value to floating point

6.1.1.14.21. Converts to complement of argument
6.1.1.14.22. Normalize the argument
6.1.1.14.23. Convert BCD to float
6.1.1.14.24. Convert float to BCD
6.1.1.14.25. Televideo macros
6.1.1.14.26. Compare two 16 bit arguments

6.1.1.15. Nmi8i.s – 8 HZ NMI interrupt service equates

6.1.1.15.1. Enable scalar counter
6.1.1.15.2. Clear scalar counter

6.1.1.16. Pcb.s – Port control block equates

6.1.1.16.1. ACIA address
6.1.1.16.2. Input buffer address
6.1.1.16.3. Output buffer address
6.1.1.16.4. Scratchpad pointer
6.1.1.16.5. Device initialization routine
6.1.1.16.6. Interrupt service routine
6.1.1.16.7. Response routine
6.1.1.16.8. PCB length

6.2. Source Modules

6.2.1. Alrsvc.s – serves alarms, relays and front panel lights, range light (under and over range), fail and Jam conditions. Jam condition is not checked.

```
Initialize variables
ALR:
    Get checksource flag
    Combine with decay timer
    If checksource not active goto ALR20
    Get jumper status
    If alarms enabled during checksource then goto ALR20
    Combine warn mask and high alarm light
    Complement the result
    Mask all front panel lights
    Set high and warn lights to off
    Goto ALR40
    ALR20:
        Get the high alarm status
        Check for errors
        If no errors then goto ALR23
        Clear the status
        Save status
    ALR23:
        Get the states address table for high alarms
        Determine current high alarm state
        Goto current state
```

ALR30:
- Get warn alarm status
- Check for errors
- If no errors goto ALR33
- Clear status
- Save status

ALR33:
- Get the states table addresses for warn states
- Determine current warn state
- Goto current state

ALR40:
- If counting is disabled after power up then goto ALR80
- If counting is disabled then goto ALR45
- If not in under-range goto ALR41
- Set under-range flag
- Set range alarm
- Goto ALR60

ALR41:
- In under-range flag is not set then goto ALR42
- Clear under-range flag
- Turn under-range light off
- Indicate normal conditions

ALR42:
- If counting is not enabled then goto ALR45
- If not in over-range then goto ALR43
- Set range alarm
- Set Hyst to normal
- Goto ALR60

ALR43:
- Set the count flag
- Indicate over-range
- Set the range alarm
- Clear over-range flag
- Enable counting
- Clear the software acknowledge flag

ALR60:
- If no anti-jam option then goto ALR63
- If not hardware jam then goto ALR61
- Turn on fail light
- De-energize relay
- Turn on range light
- Set jam indicator
- Turn high voltage off
- Update_scalar_register
- Set fail flag
- Disable counting
- Goto ALR80

ALR61:
    If not option no-counts fail then goto ALR80
ALR63:
    If no-counts fail goto ALR64
    If timer equal zero goto ALR67
    Clear timer
ALR67:
    If fail flag is set then goto ALR80
    Reset range flag
    Reset fail flag
    Energize fail relay
    Set fail flag
    Save the status
    Goto ALR80
ALR64:
    If unit is in fail then goto ALR80
    Increment the timer
    Save the timer
    If unit is not in fail then goto ALR80
    Turn on fail light
    De-energize relay
    Turn on range light
    Get fail flag
    Set fail status
ALR80:
    Set/reset relays
    Save relay status
    Update lights
    Save light status
ALR99:
    Return to caller

STATE0: high alarm state 0 routine – no high alarm
    If not in high alarm then goto ST040
    If fail safe then goto ST010
    Release high alarm relay
    Update relay status
    Save relay status
    Goto ST012
    ST010:
        Activate high alarm relay
        Update relay status
        Save relay status

ST012:
    Turn high alarm light on
    Update light status
    Save light status
ST020:
    If not auto alarm then goto ST030
    Increment high alarm state
ST030:
    Increment high alarm state
ST040:
    Return to caller

STATE1: high alarm state 1 routine – unacknowledged high alarm
    If high alarm has not been acknowledged then goto ST110
    Turn high alarm light on
    Update light status
    Save light status
    Release the rate relay
    Update rate relay status
    Save rate relay status
    Increment high alarm state
    Enable counting
    Clear software acknowledge flag
    Save software acknowledge flag
    Goto ST120
    ST110:
        Flash the high alarm light
        Activate rate relay
        Update rate relay status
        Save rate relay status
    ST120:
        Return to caller

STATE2: high alarm state 2 routine – acknowledged high alarm
    If still in high alarm then goto ST230
    Turn high alarm light off
    Update high alarm light status
    Save high alarm light status
    If fail safe then goto ST210
    Activate high alarm relay
    Update high alarm relay status
    Save status
    Goto ST220

```
        ST210:
            Release high alarm relay
            Update relay status
            Save relay status
        ST220:
            Set high alarm state to zero
        SST230:
            Return to caller

STATW0: warn alarm state 0 routine – no warn alarm
    If not in warn then goto STW040
    If fail safe then goto STW010
    Release warn relay
    Update relay status
    Save relay status
    Goto STW012
    STW010:
        Activate warn relay
        Update relay status
        Save relay status
    STW012:
        Turn warn alarm light on
        Update warn light status
        Save warn light status
    STW020:
        If not auto alarm then goto STW030
        Increment warn alarm state
    STW030:
        Increment warn alarm state
    STW040:
        Return to caller

STATW1: warn alarm state 1 routine – warn alarm unacknowledged
    If the alarm has not been acknowledged then goto STW110
    Turn light on steady
    Update light status
    Save light status
    Release rate relay
    Update rate relay status
    Save rate relay status
    Increment warn alarm state
    Enable counting
    Clear software acknowledge flag
    Update software acknowledge flag
    Save software acknowledge flag
    Goto STW120
```

STW110:
Flash the warn light
Update warn light status
Save warn light status
Activate rate relay
Update rate relay status
Save rate relay status
STW120:
Return to caller

STATW2: warn alarm state 2 routine – warn alarm acknowledged
If in warn alarm then goto STW230
Turn warn light off
Update light status
Save light status
If fail safe then goto STW210
Activate rate relay
Update rate relay status
Save rate relay status
Goto STW220
STW210:
Release warn relay
Update relay status
Save relay status
STW220:
Set warn state to zero
STW230:
Return to caller

ALRTAB:
Define high alarm state table
STATE0 – no alarm
STATE1 – unacknowledged alarm
STATE2 – acknowledged alarm

WRNTAB:
Define warn alarm state table
STATW0 – no alarm
STATW1 – unacknowledged alarm
STATW2 – acknowledged alarm

6.2.2 Anaout.s –Analog output functions

```
Initialize variables
ANAOUT:
    Call CHKRNG to save low and high ranges
    Clear floating point total
    If analog output setpoints are not valid then go to SENDNUM
    If in calibration mode then goto SENDNUM
    If checksource timer is active then goto SENDNUM
    If checksource is on then goto SENDNUM
    If unit is in fail then goto SENDNUM
    If in jam then goto MAXNUM
    If over-range then goto MAXNUM
    If not under-range then goto CONTRN
    Goto SENDNUM
    CONTRN:
        If current value is not equal to high scale then goto NXREFPT
    MAXNUM:
        Maximize the floating point total
        Goto SENDNUM
    NXREFPT:
        If current value is greater than or equal to low scale then goto CONT1
        Goto SENDNUM
    CONT1:
        Calculate the quotient as current value divided by low scale
        Call FPLOG to get log of the quotient
        Divide the log by the number of decades
        Multiply the result by defined percentage
        Get the exponent
        If not valid then goto SENDNUM
        Get the mantissa, can only be one or two
        If two digits then goto SENDNUM
        Right justify the top nibble
        If greater than or equal to 8 then goto RND
        Goto SENDNUM
    RND:
        Put the integer in the proper byte
    SENDNUM:
        Get scalar address
        Get byte that represents current value
        Move byte into DAC
        Return to caller
```

```
CHKRNG:
    Save low scale
    Save high scale
    Set range to 1 decade
    Call GETRNG to get range of low scale
    If error occurred then goto RNGERR
    Call GETRNG to get range of high scale
    If error occurred then goto RNGERR
    Calculate range in decades by subtracting low range from high range
    If low range is greater than high range goto RNGERR
    Save integer range
    If range equals zero then goto RNGERR
    Clear analog error flag
    Clear low nibble
    If high nibble equals zero then goto NOHIN
    Increment range
    Get range mantissa
    Goto STORNG to store range mantissa
NOHIN:
    Get the range mantissa
    Justify to high nibble
STORNG:
    Store range mantissa
    Goto RNGEX
RNGERR:
    Set analog error flag
RNGEX:
    Return to caller

TABLE:  table used by the FPLOG routine

FPLOG: routine to compute the common logarithm (base 10)
    Initialize variables
    BIGLOOP:
        Call ST_X_Z to store initial 4 byte number (xarg) in shifted version (zarg)
        Make a copy of counter
        While counter not zero do begin
            Call SHIFT to shift zarg right by 1
            Decrement counter
        End while loop
    TSTEND:
        If first byte of xarg not equal to 41H then goto COMP
        If second byte of xarg  not equal to 10H then goto NOSTP
        Goto STOP
    NOSTP: (label needed for addressing)
```

COMP:
    Call FSUB to calculate difference of xarg and zarg
    Get exponent of difference
    If exponent is less than 41H then goto SHFT
REDVAL:
    Store difference
    Goto ST_X_Z
    Make a copy of counter
    While counter not equal to zero do begin
        Call SHIFT to shift zarg
        Decrement counter
    End while loop
    Get TABLE starting address
    Call BX4TOX to set X = (B*4) + X
    Save the results
    Get address of newv
    Load and store each byte
    Goto BIGLOOP
    SHFT:
        Call SHIFT to shift zarg right one bit
        Increment calculation parameter
        If calculation parameter equal 16 goto STOP
        Goto TSTEND
    STOP:
        Get the address of xarg
        Move each byte of yarg into xarg
        Return to caller

ST_X_Z: routine to store xarg in zarg
    Get the address of xarg
    Move each byte of xarg into zarg
    Return to caller

SHIFT: routine to shift zarg right one bit
    Save A and B on stack
    Get the address of zarg
    Shift the first byte of the mantissa to the right
    If normalization is not needed goto GO_ON
    Restore the first byte of the mantissa
    Initialize a counter

```
TAKOFF:
    Shift the bottom byte to the left
    Rotate byte left 2 bit positions
    Rotate byte left 1 bit position
    Decrement the counter
    If counter not equal to zero goto TAKOFF
    Goto OUTTA
GO_ON:
    Bring back the carry bit
    Rotate byte right 2 bit positions
    Rotate byte right 3 bit positions
OUTTA:
    Pull A and B off the stack
    Return to caller
```

```
GETRNG:  routine to get the range in decades, input is at X, return range is
         at A
    Initialize range to zero
    Save range
    Get start of powers of ten table
    While not end of table do begin
        If range number equal to entry in table then goto GOTRNG
        Increment table pointer
    End while
    GOTRNG:
        Save range decade
        Return to caller
```

PTENTBL: Table of powers of ten

6.2.3 Calc.s – Estimates the counts per minute to within 2, 5, or 10 percent statistical accuracy depending on option jumpers.

```
Initialize variables
Define target count table
CPMSVC:
    CPM1:
        If not end of minute then goto CPM9
        Add the new value to the 20 –minute buffer
        Call AVERAG
    CPM2:
        Round current value
        Calculate +3 SIGMA, high 3 SIGMA
        Calculate –3 SIGMA, low 3 SIGMA
        Set counter to 1
    CPM6:
        While counter is less than 19 do begin
            Point to 20minute data
            Call LOCATE
            If no data then go to CPM8
            If value is greater than or equal to high 3 SIGMA then goto CPM8
            If value is less than or equal to  low 3 SIGMA then goto CPM7
            Increment counter
            If target count reached then goto CPM8
        End while
    CPM8:
        Float the count
        Calculate the CPM
        Set the display update flag
        Goto CPM10
    CPM9:
        If the current value is not less than the target then goto CPM11
        Set the display update flag
        Goto CPM10
    CPM11:
        Clear the display update flag
    CPM10:
        If the display update flag is not set then goto CPM99
        If backgroung subtract option is present goto CPM12
        Save count values
        Goto CPM14
    CPM12:
        Subtract the background
        Normalize to zero
```

CPM14:

If conversion constant option is present then goto CPM17
Save values as MPH
Goto CPM99

CPM17:

Multiply in conversion constamt

CPM99:

Return to caller

RESETC: routine to reset channels

Get the second queue
Call CLEARQ to clear the queue
Get the minute queue
Call CLEARQ to clear the queue
Initialize current MPH to 4000H
Initialize current value to 4000H
Set timer to 59
Return to caller

CPMI:

Get 20 minute buffer
Call INITQ to initialize buffer
Initialize FP3 to 4130H
Select target counts according to option register setting
Return to caller

6.2.4 Calsvc.s – Calibration service

```
Initialize variables
CALSVC:
    If in state 1 then begin
        Blank system lights
        Save light status
        Turn High Alarm Off
        Turn Warn Alarm Off
        Check Monitor status flags
        If fail safe relays then begin
            De-energize warn relay
            De-energize high relay
            Goto  NOFS1
        End if
        If no fail safe relay then begin
            Energize warn relay
            Energize high relay
        End if
            NOFAIL1:
                De-energize the fail relay
                Save the relay status
                Set the calibration value to 0
                Goto XREFCAL
    End state 1
    If in state 2 then begin
        Turn on System lights
        Save light status
        Check calibration value
        If STP is okay then goto STPOK1
        Use maximum value
            STPOK1:
                Initialize scalars
                Set to run state
                Goto XREFCAL
    End in state 2
    If not in state 3 then goto CC4
    Get the timer
    If timed out then goto CALOUT
    Save the timer
    Convert to floating point
    Calculate the sum
    Save the sum
    Toggle the lights
    Goto XREFCAL
```

CALOUT:
  Turn lights off
  Save the lights status
  Indicate last state
  Goto XREFCAL
CC4:
  Turn on lights
  Save light status
XREFCAL:
  Check STP select switch
  If switch position 8 is set then goto LEVCAL
  Set calibrate flag to reset system end exit
LEVCAL:
  Return to caller

CALINI: routine to initialize calibration timer
  Initialize timer to '423CH' (60 seconds)
  Return to caller

CALMAX:
  Defines maximum value of 65,535 seconds

6.2.5 Chksvc.s – Checksource enable/disable. Checksource is kept on while button is held down

```
Initialize variables
CHKSVC:
    If decay timer is active then begin
    Decrement timer
    If timer has expired then goto CHK02
        Goto CHK99
    End if
    Else goto CHK01
    CHK02:
        Clear the queues
        Goto CHK99
    CHK01:
        Read the switch register
        Save the switch status to the old status
        Compare old status with the new status
        If the status has not changed then goto CHK99
        If new status is checksource off the goto CHKOFF
        If checksource flag is off then goto CHKOFF
        Call RESETC
        Save the status
        Indicate checksource is on
        Clear the decay timer
        Goto CHK99
    CHKOFF:
        Turn off relay
        Save status
        Update decay timer
    CHK99:
        Return to caller
```

6.2.6 Clock.s – Schedule and clock functions

Initialize variables

RESET: this executes whenever a reset occurs. This happens whenever power comes up, or the Reset line is signaled.

```
Set up 400H byte stack
Get memory starting address
While not at upper memory boundary do begin
    Clear memory
    Increment memory pointer
End while
Call RESETC to intialize all the rotating buffers
Set delay timer for 5 seconds
Inhibit counts for 5 seconds
Set units code
RST20:
    While not end of block do begin
        Copy floating point number from ROM to RAM
    End while
    Goto RST30
RST30:
    If fail option is not present then goto RST40
    Get the high alarm relay bit
    Get the warn alarm relay bit
    Get the rate relay bit
    Set the bits in the RAM buffer
    Light the backlight for the engineering units
RST40:
    Get the options byte
    If RS-232 option is not available then goto RST46
    Get pointer to ACIA table
    While not end of table do begin
        Get ACIA routine address
        If address is NULL then goto RST46
        Set parameter for initialization routine
        Get the ACIA buffer
        Get the initialization routine
        Execute the initialization routine
        If not SCANRAD option then goto NOTCOMSR
        Set offset to next communication port to 10
        Goto CALCOFST
        NOTCOMSR:
            Set offset to next communication port to 8
        CALCOFST:
            Call ATOX to calculate the next communication port address
    End while
```

```
RST46:
    Get the units of radiation
RST48:
    While not end of unit text do begin
        Move units's text into temporary buffer
    End while
    Turn on the high voltage
    Set counts information to normal (62H)
    Determine number of lines to be printed
    Set first digit to 1FH
    Set next to 00
    Set next to 10H
    Set next to 10H
    Set next to 1FH
    Set display count to 4
RST53:
    While not all digits displayed do begin
        Increment digit index
        Save display count
        Save display data
        Decrement display count
    End while

INITBL: initialization table
    Call COMINI
    Call INTINI
    Call ENTIN
    Call CALINI
    Call CNTINI
    Call CPMI
    Call DSPINI
    Save address of backup indicator as backup indicator
    Initialize button latch

CLOCK: routine to keep the time of day and does not run when counts are
inhibited
    If no communication option then goto CLCK10
    Call IRQSVC to check for communication commands
     CLCK10:
        Call ENTSVC
        Call EESVC
```

```
CLCK20:
    If not in calibration run mode then goto CLCK40
    If ticks not equal to zero then goto CLCK30
    Get the light status
    Or the status with 0E0H
    Save the light status
    Goto CLCK40
CLCK30:
    If ticks not equal to 6 then goto CLCK40
    Get the light status
    And the status with 01FH
    Save light status
CLCK40:
    If ready to run flag is not equal to zero then goto CLCK41
    Goto SLEEP
CLCK41:
    Clear the ready to run flag
    If the noise suppression timer is equal to zero then goto CLCK50
    Decrement noise suppression timer
    Clear the number of counts accumulator
    Goto CLCK83
CLCK50:
    Update the seconds counter
    Save the seconds counter
    If seconds counter is less than 60 then goto QGO
    Clear seconds counter
    Get minutes counter
    Increment minutes counter
    Save minutes counter
    If minutes counter is less than 60 then goto QGO
    Clear minutes counter
    Get hours counter
    Increment hours counter
    Save hours counter
    If hours counter is less than 24 then goto QGO
    Clear hours counter
    Get absolute days
    Increment days
    Save days
QGO:
    If calibrate status is equal to zero then goto CLCK80
```

CHKCAL:
    If calibrate is not over then goto NOVC1
    Clear calibrate status
    Get units
    Save units
    Goto RESET
NOVC1:
    Clear current units to indicate not normal operation
    Call DSPSVC – handle the display
    Call CALSVC – handle calibration
    Goto CLCK90
CLCK80:
    Call DSPSVC – handle the display
    Call ALR – detect alarms
CLCK83:
    Call CNTSVC – average/deadtime counts
    Call CPMSVC – estimate true CPM rate/rad value
    Call ANAOUT – output CPM to chart
    Call INTDSP – calculate integer display value
    Call CHKSVC – process checksource button
CLCK90:
    Call RESETF – reset watchdog timer
CLCK93:
    Call RESETA – clear switch latch
    Get checksource decay timer
SLEEP:
    Goto CLOCK to wait for interrupt

RESETF: routine to reset watchdog timer (by pulsing fail light) and sleep until Awakened by NMI. (Pulse is 20 microseconds negative)
    Get scalar registers
    Get fail bit
    Get present relay output
    Get complement of fail bit
    Clear fail bit
    Rewrite output register
    Call DELAY to wait for 20 microseconds
    Restore original output
    Return to caller

RESETA: routine to pulse latch clear line
- Get scalar registers
- Get clear switch latch
- Get complement of clear switch latch
- And with scalar control register
- Rewrite output
- Call DELAY to wait for 20 microseconds
- Restore original output
- Return to caller

DELAY: routine to delay for 20 microseconds
- Three no operation instructions
- Return to caller

6.2.7 Cntsvc.s – Calculates the dead time corrected CPM

```
Initialize variables
CNTSVC:
    If counts are enabled then goto OV1
    Goto EXOVC
    OV1:
        Check reset timer
        If timed out goto OV1_1
        Set scalar sum to zero
    OV1_1:
        Get range information
        If over range goto OV2
        Call RESETC to reset channel
        Clear CPM
        Goto DONE
    OV2:
        Get the scalar sum
        Call AVERAG
        Compute counts per minute
        Call DEADT to compute deadtime correction
        If correction not needed then goto DONE
        Calculate correct CPM for deadtime
    DONE:
        Save the CPM value
    EXOVC:
        Return to caller
    CNTINI:
        Call INITQ
        Reset timer to 5 seconds
        Return to caller
```

6.2.8 Com232.s – RS232 communication functions

Initialize variables

COM232:

Get ACIA buffer
Save ACIA butter
Get ACIA address
Save it in ACIA buffer
If interrupt was received then goto COM80
Point to ACIA
Point to ACIA
If transmit flag is equal to zero then goto COM10
Goto COM99

COM10:

If character is not backspace then goto COM20
If number of input characters is equal to zero then goto COM99
Decrement the number of input characters
Decrement input character pointer
If result not equal 0FFH then goto COM12
Decrement input buffer pointer

COM12:

Call ECHO to echo the received character
Goto COM99

COM20:

Call ECHO to echo the received character
Store the character in the ACIA buffer
Increment the number of characters
Increment the input buffer pointer
If not equal to zero then goto COM25
Increment input buffer pointer

COM25:

If character is carriage return then goto COM30
Call ECHO to echo the received character
Goto COM99

COM30:

Get the input buffer starting address
Disable receiver interrupt
Save command register copy
Print 'CR'
Save 'CR' in the data register
Get ACIA starting pointer
Call ABX
Save the buffer pointer
Get the buffer starting address
Get the entry point
If equal to zero then goto COM32

Set command number to 7
Save command number
Get the address of the command subroutine table
Call ABX
Store the command subroutine address in the ACIA buffer
Queue the ACIA buffer address in the IRQ queue
Set process flag to prevent transmission until response
Point to IRQ queue
Save ACIA buffer address in queue
Update IRQ queue pointer
Get the buffer starting address
Goto COM99

COM32:
Get the input pointer starting address

COM34:
Get next character
If character is a space then goto COM34

COM40:
Clear the digit counter
If character not equal to 'CR' then goto COM42
Set number of lines to '1'
Print prompt
Goto COM70

COM42:
If character is not '?' then goto COM41
Save help command
Goto COM48

COM41:
Call CHARC to check if alphabetical character
If alphabetical character goto COM44
Set error number to zero for syntax error
Goto COM70

COM44:
Make sure character is upper case
Save character in command
Get next character
Call CHARC to check if alphabetical character
If alphabetical character then goto COM46
Set error number to zero for syntax error
Goto COM70

COM46:
Make sure character is upper case
Save character in command

COM48:
Get next character
If character is a space then goto COM48

COM45:
If character not equal carriage return then goto COM60
COM50:
If command not equal 'AL' for display alarms then goto COM52
Set command number to '2'
Goto COM70
COM52:
If command not equal 'DS' for display radiation values then goto COM54
Set command number to '3'
Goto COM70
COM54:
If command not equal 'SP' for setpoint display then goto COM56
Set setpoint number to '0FFH' to display all setpoints
Set command number to '4'
Goto COM70
COM56:
If command not equal 'VR' for display program version then goto COM57
Set command number to '5'
Goto COM70
COM57:
If command not equal '?' for help command then goto COM58
Set command number to '6'
Goto COM70
COM58:
Set error number to zero for syntax error
Goto COM70
COM60:
Call VALID to check for digit
If a valid digit then goto COM61
COM61:
Save the digit
Increment the number of digits
Get the next character
Call VALID to check for digit
If not a valid digit then goto COM62
Save the digit
Increment the number of digits
COM63:
Get the next character
COM62:
If the character is a space then goto COM63
If the character is a carriage return then goto COM64
Indicate syntax error
Goto COM70

COM64:
    Get the command number
    If command is 'SP' for setpoint display then goto COM65
    Indicate syntax error
    Set error number to zero
    Goto COM70
COM65:
    Get the previous digit
    If the number of digits is '1' then goto COM66
    Multiply the previous digit by '10'
    Add the current digit to the result of the multiplication
COM66:
    Save the result as the setpoint number
    Get the ACIA buffer starting address
    Get the setpoint table starting address
    Call ABX to add the contents of accumulator B to the contents of IX
    If setpoint is not used goto COM69
    If setpoint number is greater than the maximum setpoint number then goto COM69
    If setpoint number is not equal to the maximum setpoint number then goto COM70
COM68:
    If setpoint number is not within the limit then goto COM69
    Set command number to '4'
    Goto COM70
COM69:
    Set error number '1' for setpoint out of range
    Save error number
COM70:
    Save command number
    Get command subroutine table starting address
    Store the command subroutine address in the ACIA buffer
    Queue the ACIA buffer address in the IRQ queue for processing command
    Goto COM99
COM80:
    Get ACIA status address
    If transmit buffer is not empty then goto COM81
    Goto COM99
COM81:
    If the process flag is not set then goto COM99
    Get the character to be output
    If end of message then goto COM84
    Set output pointer to next location
    If not equal to zero then goto COM82
    Increment high byte of output pointer

COM82:
  Send the character in turn
  Goto COM99
COM84:
  Decrement number of lines in response
  If number of lines equal to zero then goto COM90
  If one line is not left in response then goto COM87
  Print the prompt
  Store the command subroutine address in ACIA buffer
COM87:
  Set the process flag to prevent transmission until response
  Disable transmit interrupt
  Save command register copy
  Queue the command in IRQ queue
  Update the queue pointer
  Goto COM99
COM90:
  Clear the number of characters
  Get pointer to input buffer
  Goto ABAX
  Disable transmit interrupt
  Enable receive interrupt
  Save the new command register contents
  Write command to ACIA
  Point to ACIA buffer
  Clear transmit flag
COM99:
  Get ACIA buffer address
  Save in ACIA buffer
  Return to caller

ECHO: routine to echo received character, character is in accumulator B
  Get ACIA starting pointer
  Disable receiver interrupt
  Enable transmit interrupt
  Save command register copy
  Modify ACIA command register
  Disable transmit interrupt
  Enable receive interrupt
  Save the new command register contents
  Return to caller

VALID: routine to test an input digit
  Input: Digit to be tested is in accumulator A,
  Output: carry bit clear - valid decimal digit,
    carry bit set – invalid decimal digit

CHARC: routine to test input character
Input: Character to be validated is in accumulator A
Output: carry bit clear – valid alphabetic character
carry bit set – invalid alphabetic character

ABX: routine to add the contents of accumulator B to the contents of X
Input: value in accumulator B,
value in X
Output: results are in accumulator A and B
contents of accumulator B are unchanged

ABAX: routine to add the contents of accumulator B
Input: value in accumulator B
value in X
Output: results are in accumulator A and B
contents of X are unchanged

COMINI: routine to initialize ACIA and parameter blocks
Save ACIA buffer
Get the ACIA address
Store the ACIA address in the ACIA buffer
Get ACIA address
Read the data register to clear any interrupts
Read the DIP switch
Clear out unused bits
OR in status register
Write to control register
Save control register in copy
Save in control register copy
Set ACIA to receive interrupt
Set data terminal ready bit
Enable transmit interrupt
Get pointer to ACIA buffer
Save command register copy
Get prompt routine starting address
Get ACIA starting address buffer
Save subroutine address in buffer
Set the process flag
Get pointer to IRQ queue
Save ACIA buffer address in queue
Update queue pointer
Save queue pointer
Get offset to output buffer
Call ABAX

Save results in output buffer
Get input buffer offset
Call ABAX
Save results in input buffer
Return to caller

6.2.9 Com485.s – RS485 Interrupt service functions, the last character is the Linear Predictive Coding (LPC) character

```
Initialize variables
COM485:
    Save current values on the stack
    Point the port parameters
    Access ACIA address
    Read ACIA status register
    If receive register is not full then goto XMITINT
    Read input character
    Save input character
    If ACIA status is parity error then goto ERRFLG
    If ACIA status is overrun error then goto ERRFLG
    If ACIA status is no error then goto NOERR
    ERRFLG:
        Set error flag
        Save error flag in input buffer
    NOERR:
        Call DECODE to check if character is valid based on message format
    XMITINT:
        Access ACIA status register
        If transmit register is empty then goto EXSIT
        Call GETCHR to get the next character
        If transmit buffer is not empty then goto XMIT
        Initialize delay to 0FFH
        While delay is not equal to 0 do begin
            Decrement delay
        End while
        Access ACIA address
        Reset transmit interrupt request
        Goto EXSIT
    XMIT:
        Access ACIA status register
        Output character
    EXSIT:
        Restore variables from the stack
        Return to caller
```

```
DECODE: routine to check validity of character based on message format
    Access the input buffer address
    Read the last character received
    Get the message state table
    Goto service routine based on state of operation
    Call BTXS
    SHTAT0: received character must be 'STX'
        Access input buffer address
        If received character is not equal to start of transmission character
         ('STX') then goto BDMSG
        Call CLRBUF to clear the input buffer
        Increment state counter
        Goto FINI
    BDMSG:
        Set state counter to zero
        Goto FINI
    SHTAT1: received character must be number of bytes
        Access input buffer address
        If received character not equal to 'STX' then goto CHKBYT
        Set state to '1'
        Goto FINI
    CHKBYT:
        Update the last character
        Read the input character, this should be the number of data bytes
        Clear the sign bit
        If the number of data bytes is less than minimum number of data
         bytes then goto FIN
        If the number of data bytes is greater than maximum number of bytes
         then goto FIN
        Save number of data bytes
        Set state counter to '2'
        Goto FINI
    SHTAT2: received character must be channel id
        Access input buffer address
        Update last character
        Read input character
        Decrement number of data bytes
        If the number of bytes is greater than zero then goto STAT2A
        Set state to zero
        Goto FINI
    STAT2A:
        Save channel id in input buffer
        Set state to '3'
        Goto FINI
```

SHTAT3: received character must be 'CMD'
Access input buffer address
Update last character
Read input character
Decrement number of data bytes
If number of data bytes is greater than zero then goto OK3
If message is not for this channel then goto NOTMEE
Goto ER_LEN
OK3:
Save CMD in input buffer
Set state to '4'
FIN:
Goto FINI
NOTMEE:
Goto NOTME
SHTAT4: received character must be 'SBC'
Access input buffer address
Update last character
Read input character
Decrement number of data bytes
If number of data bytes is equal to zero then goto STA6
If number of data bytes is greater than zero then goto STA5
If message not for this channel then goto NOTME
Goto ER_LEN
STA6:
Set state to '6'
Goto FINI
STA5:
Set state to '5'
Goto FINI
SHTAT5: received character must be data or checksum
Access input buffer address
Update last character
Get the data length
Call ATXS
DONINC:
Save the data in the input buffer
Access the input buffer address
Increment the data counter
Decrement the number of bytes
If the number of bytes is equal to zero then goto STST6
OK5:
Set state to '5' to get more data
Goto FINI

```
STST6:
    Set state to '6'
    Goto FINI
SHTAT6:
    Access input buffer address
    If message is not for this channel then goto NOTME
    If LPC is not correct then goto ER_LPC
    If error status is set then goto CHKER to determine particular error
DOHICK:  good message received
    Access the address of message handling routine
    Goto message handling routine
NOTME:  message not for this channel
    Set state to zero
    Goto FINI
CHKER:  receiver communication error
    If overrun error has occurred then goto ER_OVR
    If parity error then goto ER_PAR
    If framing error then goto ER_FRM
FINI:  partial word received, wait for the rest
    Access input buffer address
    Update state counter
    Return to caller
ACKNOW:
    Set response to 'ACK'
    Call STOCHR
ACK_A:
    Call SNDMSG, respond with data
    Goto EKSREF
ER_PAR:
    Set error to parity error
    Goto ERR_E
ER_LPC:
    Set error to bad LPC
    Goto ERR_E
ER_OVR:
    Set error to data overrun
    Goto ERR_E
ER_FRM:
    Set error to framing error
    Goto ERR_E
ER-LEN:
    Set error to bad message length
    Goto ERR_E
ER_NUM:
    Set error to invalid data
    Goto ERR_E
```

ER_SBC:
Set error to invalid subcommand/function
Goto ERR_R
ER_NON:
Set error to invalid command/channel
Goto ERR_E
ER_MOD:
Set error to not in remote
ERR_E:
Save error code
Access output buffer address
Initialize response buffer
Save 'NAK' in output buffer
Call STOCHR
Get error code
Call STOCHR to store error code in output buffer
Call SNDMSG to initiate transmission
Goto EKSREF

SNDMSG: routine to initiate transmission of response
Get input buffer address
Clear input buffer
Get output buffer address
Get number of bytes in message
Mask in the 10000000B, the preset bit
Save the result as number of bytes
Add this number to the LPC
Call STOCHR to put this LPC in the output buffer
Access the output buffer address
Indicate character has been processed
Decrement the number of bytes, LPC not counted in number of bytes
Return to caller
EKSREF:
Point to port parameters
Enable transmit IRQ's for this ACIA
Save in command register
Return to caller

CLRBUF: routine to clear a communication buffer, Input: address of buffer to be cleared, Output: None
Clear error status
Set state to zero
Set data length to zero
Set LPC to zero
Set first data byte pointer to zero
Set last data byte pointer to zero
Return to caller

STOCHR: routine to store a character in the output buffer, Input: The character to be stored, Output: the carry bit is set if the buffer is full

```
Access the output buffer address
Increment the number of bytes in the message
Get the current message length
If the current message length is less than the total buffer length then goto
L1010
Set the carry bit
Goto L1099
L1010:
    Increment the current message length
    Save new length
    Update the LPC
ADDBUF:
    Call BTXS to store the character in the output buffer
    Increment the buffer pointer
    Clear the carry bit to indicate success
L1099:
    Return to caller
```

GETCHR: routine to get the next character to transmit, Input: none, Output: retrieved character and the carry bit is set if the buffer is empty

```
Access the output buffer address
If the state is equal to zero then goto NOCHAR
If the current buffer size is equal to zero then goto NOCHAR
Get the pointer to the next character
If the pointer to the end of the buffer is less than or equal to the pointer to the
next character then goto L1110
NOCHAR:
    Set the carry bit to indicate that the buffer is empty
    Goto L1199
L1110:
    Get the index to next character to send
    Call BTXS
    Get the character
    Access the output buffer address
    Increment the character pointer
    Clear the carry bit to indicate that the character was read
L1199:
    Return to caller
```

SETAP: setup routine to initialize the transmit buffer
- Access output buffer address
- Clear output buffer
- Get start of transmission string
- Store string in output buffer
- Set data end to '2'
- Return to caller

ATXS: routine add accumulator A to X
- Save accumulator A on the stack
- Save accumulator B on the stack
- Add accumulator A to X
- Restore accumulator A from stack
- Restore accumulator B from stack
- Return to caller

BTXS: routine to add accumulator B to X
- Save accumulator A on the stack
- Save accumulator B on the stack
- Add accumulator B to X
- Restore accumulator A from stack
- Restore accumulator B from stack
- Return to caller

COMINA: routine for ACIA initialization
- Access ACIA address
- Read data register
- Read baud rate from option board
- Clear out unused bits
- Set RTS to baud rate and 1 stop bit for WL 8
- Write to control register
- Set data terminal ready bit and parity mode enabled
- Write to command register
- Return to caller

COMINB: routine to initialize unused ACIA
- Access ACIA address
- Clear IRQ
- Clear receive status
- Reset device
- Return to caller

STATBL: decode message state table

6.2.10 Common.s – Contains any variable or table that is either used by more than one module or is dependent on the monitor's configuration.

Declare public variables for scalar output states
Declare public parameters for average and deadtime routines
Declare public detector related pointers
Declare public variables for clock updates
Declare public variables for scalar counts updated by GETCNT routine
Declare public variables for single channel and Americum channel accumulators buffers, channel counts and related tags
Declare public status word and status/setpoint change words
Declare public set indicator lights, horn and beacon
Declare public RS232 communication ACIA buffers
Declare public analog voltages
Declare public PCB scratchpad area
Define variables displayed in response to '?' or 'HE'
Define the structure of each queue
Stack declarations
Initializing queues
Define tables used by communication section

6.2.11 Comsub.s – Services the command subroutine of the serial communication.

Define temporary buffers

IRQSVC:

- Get pointer to interrupt queue
- If no interrupts then goto IRQ99
- Get the last entry in the table
- Dequeue the entry
- Save ACIA buffer address
- Execute the command
- Get the buffer starting address
- Clear the process flag
- Get ACIA buffer pointer
- Get offset to output buffer
- Call ABAX
- Save the character to be output
- Update output buffer index
- Get ACIA starting pointer
- Disable receiver interrupt
- Enable transmit interrupt
- Save command register copy
- Get pointer to ACIA buffer
- Set transmit flag
- Decrement queue pointer
- If not top of queue then goto IRQSVC
- IRQ99:
  - Return to caller

CMDTAB: define command table

ERROR	- Pointer to error routine
PROMPT	- Pointer to print prompt routine
ALCMD	- Pointer to alarm display command routine
DSCMD	- Pointer to display radiation routine
SPCMD	- Pointer to display setpoint command routine
VRCMD	- Pointer to display version command routine
HELP	- Pointer to display commands routine
ESPOINT	- Pointer to set setpoint command routine

HELP: routine to display the available commands
    Get ACIA buffer starting address
    If the number of lines equal zero then goto HLP10
    Get pointer to the test
    Save a copy
    Goto HLP20
    HLP10:
        Set the number of llines to '6'
        Get the text starting address
        Save a copy of the pointer
    HLP20:
        Call BLANK to blank the response line
        Get address of response buffer
        Save address of response buffer
        Call COPY to copy the text
        Get ACIA buffer starting address
        Save pointer to text
        Store ETX
        Get response buffer starting address
        Set number of characters to be transferred to '80'
        Call MOVST
    HLP99:
        Return to caller

ALCMD: routine to display the values associated with different alarms
    Get ACIA buffer starting address
    If the number of lines is not equal to zero then goto ALC20
    ALC10:
        Set the number of lines to '2'
    ALC20:
        Call BLANK to blank out the response line
        Print 'HI' for high alarm
        Save rounding flag
        Set pointer to high alarm
        Call FP2DEFB to convert floating point number to ASCII
        Retrieve the rounding flag
        Print the units
        Get the destination address
        Set the number of characters to be moved to '5'
        Call COPYN
        Print 'WA' for warn alarm
        Set pointer to warn alarm
        Call FP2DEFB to convert floating point number to ASCII
        Print the units
        Get the destination address
        Set the number of characters to be moved to '5'
        Call COPYN

ALC60:
- Get the destination address
- Put 'ETX' at destination address
- Set the number of characters to be transferred to '80'
- Call MOVST

ALC90:
- Return to caller

DSCMD: routine to display the radiation values and status
- Call BLANK to blank the response line
- Get ACIA buffer starting address
- If the number of lines is equal to '2' then goto DSC46
- If the number of lines is not equal to zero then goto DSC16

DSC10:
- Get ACIA starting address
- Set number of lines to '3'

DSC16:
- Set text to 'Radiation:'
- Get response address
- Save buffer pointer
- Call COPY
- Get rounding flag
- Save rounding flag
- Get the radiation values
- Call FP2DEFB to convert from floating point to ASCII
- Get the rounding flag
- Save the rounding flag
- Goto DSC22

DSC22:
- Get the text for the units
- Store units
- Get the destination address
- Set the number of characters to be moved to '5'
- Call COPYN

DSC36:
- Goto DSC76

DSC46:
- Get the text for 'Status'
- Get the destination address
- Call COPY

DSC50:
- Get warn alarm status
- If status is zero then goto DSC54

DSC51:
- Get destination address
- Print 'WA' for warn alarm

DSC54:
Get high alarm status
If status is equal to zero then goto DSC56
DSC55:
Print 'HI' for high alarm
DSC56:
If check source status is zero for channel 1 goto DSC58
Get text 'C/S'
Get destination address
Call COPY
DSC58:
If calibration status is equal to zero then goto DSC60
Get text 'Calib'
Get destination address
Call COPY
DSC60:
Get under range/no counts flag
If over range bit is zero then goto DSC62
Get under range text
Get destination address
Call COPY
DSC62:
Get over range flag
If under range bit is zero then goto DSC64
Get over range text
Get destination address
Call COPY
DSC64:
Get under range/no counts flag
OR in over range flag
If fail bit is equal to zero then goto DSC76
Get fail text
Get destination address
Call COPY
DSC76:
Get destination address
Store 'ETX'
Get response buffer starting address
Set the number of characters to be transferred to '80'
Call MOVST
DSC90:
Return to caller

SPCMD: routine to print the number, name and value of each setpoint stored in EEPROM when setpoint is undefined, and to display and possibly change the value if setpoint is defined.

```
Get ACIA buffer starting address
If setpoint number is defined then goto SPC40
If number of lines equal zero then goto SPC02
Get ACIA buffer starting address
Get the text pointer
Save the text pointer
Goto SPC10
SPC02:
    Get the setpoint pointers text address
    Set number of lines to zero
SPC04:
    Save setpoint text pointer
    If setpoint is not defined then goto SPC06
    Point to the data
    If setpoint is not initialized then goto SPC06
    Increment number of lines
SPC06:
    Get setpoint text pointer
    Point to next entry
    If not end of setpoint table then goto SPC04
    Point to ACIA buffer
    Increment number of lines
    Save number of lines
    Clear setpoint number
    Set pointer to setpoint text
SPC10:
    If the setpoint is not used then goto SPCB
    If setpoint is used then goto SPC11
SPCB:
    Set not-used-setpoint flag
    Goto SPC20
SPC11:
    Call BLANK to blank the response line
    Get ACIA buffer starting address
    Get setpoint number
    Call BCDDEFB to convert to ASCII digits
    If first digit is not zero then goto SPC12
    Set text to ' '(space)
```

SPC12:

Save ASCII digits in response buffer
Get setpoint text table pointer
Point to setpoint definitions
Point to the text
Save the text pointer
Get the destination address
Call COPY
Save setpoint text pointer
Get pointer to setpoint
Point to value
Save pointer
Get destination address
Call FP2DEFB
Get pointer to units
Get pointer to destination
Call COPY
Save setpoint text pointer
Get destination address
Set test to 'ETX'
Clear not-used-setpoint flag

SPC20:

Set the setpoint text pointer to the next setpoint
Get the ACIA buffer starting address
Increment the set point number
Save setpoint number
Set text pointer to the next setpoint
If this setpoint is not used then goto SPC10
Get ACIA buffer starting address
Save pointer to setpoint text
Goto SPC80

SPC40:

Get ACIA buffer starting address
Increment setpoint level to indicate modify setpoint
Set number lines equal to '1'
Save setpoint number

SPC45:

Call BLANK to blank the response buffer
Get the ACIA buffer starting address
Clear the carry bit
Get the setpoint number
Call BCDDEFB to convert to two ASCII digits
If first digit is not equal to zero then goto SPC48
Set first digit equal to ' ' (space)

SPC48:

Save the digits in the response buffer

```
MM1:
    Get the ACIA buffer starting address
MM2:
    Get the setpoint number
MM3:
    Shift the number left
MM4:
    Point to the setpoint definition table
MM5:
    Call ADBX
MM6:
    Save the entry pointer
MM7:
    Point to the definition
MM8:
    Point to the text
    Set copy source address
    Set destination address
    Call COPY
    Get the setpoint value
    Set destination address
    Call FP2DEFB
    Get the destination address
    Save 'ETX'  at destination address
SPC80:
    Get response buffer starting address
    Set number of bytes to be transferred to '80'
    Call MOVST to copy the buffer
SPC99:
    Return to caller

VRCMD:  routine to display software part number, version, and date
    Get the ACIA buffer starting address
    If the number of lines is equal to zero then goto VRC03
    Get pointer to the setpoint text
    Save pointer
    Goto VRC05
VRC03:
    Set number of lines to '4'
    Set the starting address of the code to be executed
VRC05:
    Call BLANK to blank the response line
    Get the starting address of the code to be executed
    Goto code
```

VRC10:
- Get the title
- Get the destination address
- Call COPY
- Get the text
- Get the destination address
- Set the number of characters to be copied to '14'
- Call COPYN
- Prepare to print the next lline
- Get pointer to ACIA buffer
- Save pointer to the text
- Goto VRC50

VRC20:
- Get the part number
- Get the destination address
- Call COPY
- Get the text
- Get the destination address
- Set the number of characters to be copied to '8'
- Call COPYN
- Prepare to print the next line
- Get pointer to ACIA buffer
- Save pointer to the text
- Goto VRC50

VRC30:
- Get the date
- Get the destination address
- Call COPY
- Get destination address
- Get month version/revision was released
- Call BCDDEFB to convert to ASCII
- If first digit is not zero then goto VRC32
- Set first digit to ' ' (space)

VRC32:
- Save the digits
- Save '/'
- Get year version/revision was released
- Call BCDDEFB to convert to ASCII
- If first digit is not equal to zero then goto VRC36
- Set first digit to ' ' (space)

VRC36:
- Save the year

VRC50:
Get location in response buffer to store 'ETX'
Save 'ETX'
Get response buffer starting address
Set number of characters to be transferred to '80'
Call MOVST
VRC99:
Return to caller

PROMPT: routine to display prompt
Set text to prompt definition
Set the number of characters to be transferred to '7'
Call MOVST
Return to caller

ERROR:
Get ACIA buffer starting address
Get error code
Get error message buffer starting address
Goto ABDX
Get the number of characters in the message
Call MOVST
Set number of lines to '1'
Return to caller

ESPOINT: routine to process setpoint commands
Get ACIA buffer starting address
Get the setpoint number
Save the setpoint number in local buffer
Set pointer to input buffer
Call ADBX
Set the input buffer pointer to buffer starting address
Clear the number of digits counter
Clear the decimal point location counter
Clear the exponent sign
ESP03:
Get the pointer to the input buffer
Get the character
Increment the character pointer
Save the pointer
If the character is a space then goto EXP03
If the character is not a carriage return then goto ESP04
Set the command to '1'
Call ESP80

ESP04:

If the character is not a decimal point then goto ESP05
Set decimal point location flag
Save decimal location
Goto ESP07

ESP05:

Call VALID to check if valid decimal digit
If not a valid digit then goto ESP60
Increment the number of digits counter
Save the digit
Update the digit pointer
Increment the decimal point location counter

ESP07:

Get the pointer to the input buffer
Get the character
Update the buffer pointer
Call VALID
If digit not valid then goto ESP12
Increment the number of digits counter
Save the digit
Update the digit pointer
If the decimal point flag is set then goto ESP10
Increment the decimal point location counter

ESP10:

If the number of digits is less than or equal to '5' then goto ESP07
Goto ESP60

ESP12:

If the number of digits is equal to zero then goto ESP60
If character not equal to decimal point then goto ESP15
If decimal point flag is set then goto ESP60
Set the decimal point flag
Goto ESP07

ESP15:

If the character is not a space then goto ESP20

ESP17:

Get the pointer to the input buffer
Get the character
Update buffer pointer
If character is a space then goto ESP17

ESP20:

Make sure character is upper case
If character is not equal to 'E" then goto ESP30

ESP25:
- Get the pointer to the input buffer
- Get the character
- Update the input buffer pointer
- If the character is a space then goto ESP25
- If the character is not a '+' then goto ESP30
- Clear the exponent sign flag
- Goto ESP35

ESP30:
- If character is not a '-' then goto ESP37
- Increment exponent sign flag

ESP35:
- Call VALID
- If not a valid digit then goto ESP60
- Save the digit
- If exponent flag is equl to zero then goto ESP40
- Make the exponent negative

ESP40:
- Save the exponent

ESP42:
- Get the pointer to the input buffer
- Get the character
- Update the input buffer pointer
- Save the pointer
- If the character is equal to space then goto ESP42
- If the character is not equal to a carriage return then goto ESP60
- Get pointer to the number
- Call ETOFP to convert to a floating point number
- Get setpoint number
- Indicate set function
- Call DBASUB to enter setpoint into RAM

ESP50:
- If error occurred in DBASUB then goto ESP70
- Set to print prompt on return
- Goto ESP80

ESP60:
- Get ACIA buffer starting address
- Clear error code to indicate syntax error
- Goto ESP80

ESP70:
- Get ACIA buffer starting address
- Set error code to '2' for invalid set point

ESP80:
- Get ACIA buffer address
- Set the number of lines to '1'
- Clear the error level

ESP90:
- Get the ACIA buffer address
- Save the command number
- Get the command subroutine table starting address
- Call ADBX
- Get the ACIA buffer starting address
- Save the subroutine address in the buffer
- Execute the command
- Return to caller

ETOFP: routine to convert E-format numbers to floating point. Input: number to be convert is pointed to by IX, 3 digits followed by exponent. Output: converted floating point number is pointed to by IX

```
Clear the floating point number buffer
Set the power of ten to '1'
Get the number to be converted
If the decimal point location is equal to the number of input digits then goto
EFP10
Get the exponent
If the exponent is negative then goto EFP06
EFP03:
    Decrement the exponent
    Increment the decimal location
    If decimal location is not equal to number of input digits then goto EFP03
    Goto EFP07
EFP06:
    Decrement the exponent
    Increment the decimal point location
    If decimal location not equal to number of input digits then goto EFP06
EFP07:
    Save the exponent
    Save the decimal location
EFP10:
    Get the exponent
    Save the exponent
    If exponent is equal to zero then goto EFP24
    If exponent is negative then goto EFP17
EFP12:
    While the exponent is not equal to zero do begin
        Multiply '1E1' by power of ten
        Set power of ten equal to the result
        Decrement the exponent
    End while
    Goto EFP24
```

EFP17:
While exponent is not equal to zero do begin
Divide '1E1' by power of ten
Set power of ten to the result
Increment exponent
End while
EFP24:
Set pointer to least significant digit
Decrement the number of input digits
Call ADBX
Save the digit pointer
EFP30:
While number of digits to be transferred is not equal to zero do begin
Get the digit
Convert digit to floating point
Multiply digit by power of ten
Add result to value
Multiply '1E1' by power of ten
Set power of ten equal to result
Get pointer to the number
Decrement pointer
Decrement number of digits to be transferred
End while
Save value
Return to caller

ADBX: routine to ad contents of accumulator B to contents of X, contents of B are unchanged

Save accumulators
Save IX
Add the contents of accumulator B to the least significant byte of IX
Add '0' to the most significant byte of IX
Save the result
Retrieve the registers
Return to caller

ABAX: routine to add the contents of accumulator B to the contents of IX, contents of X are unchanged. The result is in accumulator A and B.

Clear accumulator A
Store IX
Add the contents of accumulator B to the least significant byte of IX
Add '0' to the most significant byte of IX
Save the results
Get the original contents of IX
Return to caller

MOVST: routine to copy a string to the output buffer. The number of bytes to be copied is in accumulator B.

- Store characters in local buffer
- Save the number of bytes in local variable
- Get the address of the ACIA output buffer
- Call ADBX
- Save IX in local buffer
- Get the number of bytes to be transferred
- NXTCH:
  - Get a character
  - Update pointer to character buffer
  - Store character in output buffer
  - Update output buffer pointer
  - If number of characters to be transferred is not equal to zero then goto NXTCH
  - Return to caller

BLANK: routine to blank the response buffer

- Get pointer to response buffer starting address
- Store line feed in response buffer
- Update response buffer pointer
- Store carriage return in response buffer
- Update response buffer pointer
- Set counter to '78'
- Set character to ' ' (space)
- While counter not equal to zero do begin
  - Store character in response buffer
  - Update response buffer pointer
  - Decrement counter
- End while
- Return to caller

COPYN: routine to copy a string, Input: pointer to source, pointer to destination, number of bytes in accumulator B.

- Get the source pointer
- Get the character
- Update source pointer
- Get destination pointer
- Save the character
- Update the destination pointer
- Decrement the number of characters
- If number of characters to be transferred is not equal to zero then goto COPYN
- Return to caller

COPY: routine to copy text, Input: pointer to source buffer and pointer to destination buffer. The source text should be terminated by '\'

Get the source buffer pointer
Get the character
Update the source pointer
If the character is equal to '\' then goto COPYEX
Get destination buffer pointer
Save the character
Update destination pointer
Goto COPY
COPYEX:
Return to caller

BCDDEFB: routine to convert BCD byte to two ASCII digits, Input: BCD byte in accumulator A, Output: the two digits are in accumulator A and B with the most significant digit in accumulator A.

Get the most significant digit
Add in '30H' to convert to ASCII
Mask the most significant digit
Add in '30H' to convert to ASCII
Return to caller

6.2.12 Conf.s – 956A UDR configuration

Define ID to be UDR firmware part number
Define MONSTAT, the integer display conversion constant
Define the OPTBRD, to be UDR956A or COMOPT or DET210
Define display parameters
Define 3 sets of factory setpoints

SPDF10:

0: Alarm Limit (1E3)
1: Warn Limit (1E3)
2: Resolving Time (0 minutes/count)
3: High Scale Value (1E3)
4: Over Range Limit (1E5)
5: Conversion Constant (1E0)
6: Not Used
7: Low Scale Value (.01)
8: Calibrate Time Constant (6E1)
9: Under Range Limit (.01)
10: Not Used
11: Not Used
12: Not Used
13: Not Used
14: Not Used
15: Channel Id

SPDF20:

0: Alarm Limit (1E4)
1: Warn Limit (1E2)
2: Resolving Time (0 minutes/count)
3: High Scale Value (1E4)
4: Over Range Limit (1E4)
5: Conversion Constant (1E0)
6: Not Used
7: Low Scale Value (.1)
8: Calibrate Time Constant (6E1)
9: Under Range Limit (.1)
10: Not Used
11: Not Used
12: Not Used
13: Not Used
14: Not Used
15: Channel Id

SPDF30 (Used in 94095603 PROM):

- 0: Alarm Limit (1E3)
- 1: Warn Limit (1E1)
- 2: Resolving Time (0 minutes/count)
- 3: High Scale Value (1E4)
- 4: Over Range Limit (1E4)
- 5: Conversion Constant (1E-2)
- 6: Not Used
- 7: Low Scale Value (1E-1)
- 8: Calibrate Time Constant (6E1)
- 9: Under Range Limit (1E-1)
- 10: Not Used
- 11: Not Used
- 12: Not Used
- 13: Not Used
- 14: Not Used
- 15: Channel Id

Set up bar graph parameters

- Bar low value 1 : 0.01MRH
- Bar high value 1 : 1.00E+06
- Bar low value 2 : .0.01 MRH
- Bar high value 2 : 1.00E+07
- Bar low value 3 : 0.01 MRH
- Bar high value 3 : 1.00E+08
- Bar off value : 0

Define SPNTB table to be the pointer table to the setpoint nomenclature

- SPN0: High Alarm Limit mR./hr
- SPN1: Warn Alarm Limit mR/hr
- SPN2: Resolve Time Min/Count
- SPN3: Full Scale Value mR/hr
- SPN4: Over Range Limit mR/hr
- SPN5: Conversion Constant mR/hr/Count
- SPN6: Background Subtract
- SPN7: Low Scale Value mR/hr
- SPN8: Calibrate Time Constant sec
- SPN9: Under Range Limit mR/hr
- SPN15: Channel Id

Define SETTAB, setpoints indexed by THINPUT value
- Function switch position 15 – setpoint address
- Function switch position 7 – low scale value
- Function switch position 14 – not used
- Function switch position 6 – background subtract
- Function switch position 11 – not used
- Function switch position 3 – high scale value
- Function switch position 10 – not used
- Function switch position 2 – resolve time
- Function switch position 13 – not used
- Function switch position 5 – conversion constant
- Function switch position 12 – not used
- Function switch position 4 – over range limit
- Function switch position 9 – under range limit
- Function switch position 1 – warn alarm limit
- Function switch position 8 – calibration time constant
- Function switch position 0 – high alarm limit

Define RS-232 VICOLOOP – the specific locations are determined by the PCB include File. A software switch determines the type of communication, ScanRad or Dumb Terminal.

Define setpoint definition table indexed by function switch position
Set up setpoint definitions
Set up units text for the 'DS' command
- CPMUNT: 'CPM' selected when units = 1
- MRUNIT: 'mR/h' selected when units = 2
- UCUNIT: 'uC/cc' selected when units = 3

Initialize current units to '2'
Define ACIA tables
Set checksource decay constant to 5 seconds
Declare process interrupt vectors

6.2.13 Dbasub.s – Setpoint subroutines – This routine is used to read/write setpoints and is reentrant

THMSUB:
- Set data pointer
- Set read/write flag
- Check for valid function code
- If error condition goto SETERC
- Get the exponent and save value
- Get setpoint
- If not defined goto SETERC
- If setpoint is not initialized goto SETERC
- If read setpoint goto READ
- Else goto SET
- SET:
  - Get setpoint number
  - Call SETEDT for validity check
  - If error then goto SETERV
- REP:
  - Store setpoint value in table
  - Set modify flag
  - Goto SETNIL
- READ:
  - Read setpoint value
- SETNIL:
  - Clear carry bit
  - Clear overflow bit
  - Goto DONE
- SETERV:
  - Clear carry bit
  - Set V bit to indicate value out of range
  - Goto DONE
- SETERC:
  - Clear overflow bit
  - Set C bit to indicate illegal channel/.function
- DONE:
  - Save condition codes
  - Set condition code
  - Return to caller

6.2.14 Dspsvc.s – UDR display services

Initialize variables

DSPSVC: this routine drives all the displays

```
If display is not inhibited then goto CHANG1
Goto DONE
CHANG1:
    Get calibrate status
    If not in calibrate then goto NOINCL
    Get calibration counts
    Goto SETUP
NOINCL:
    Get switch inputs
    If not display high voltage then goto HIALRM
    Point to high voltage value
    Goto SETUP
HIALRM:
    If not display high alarm then goto WRNALR
    Point to high alarm value
    Goto SETUP
WRNALR:
    If not display warn alarm then goto RATE1
    Point to warn alarm value
    Goto SETUP
RATE1:
    If not display rate then goto CKDSP
    Point to rate value
    Goto SETUP
CKDSP:
    Call ERROR to check setpoint validity
    If no error then goto ARDY
    Goto DSPERR
ARDY:
    Get channel range info
    If channel in over range then goto D10PS1
    If not in JAM condition then goto DISPI
D10PS1:
    Get over range value
    Save over range value in display output
    Call DSPDT2 to display value
    Call BARFUL to light bar to the maximum
    Goto OUTDSP
DISPI:
    Get pointer to current value
    If not integer display then goto SETUP
    Set digit counter to '4'
    Point to display data
    Initialize decimal point at second digit
```

```
INTLUP:
    Get display digit
    Point to next digit
    If not decimal point then goto NODPOS
    Clear bit to display decimal point
NODPOS:
    Select correct digit
    Store the data
    Move decimal point position
    Decrement digit counter
    If digit counter is greater than or equal to zero then goto INTLUP
    Turn on unit's light
    Save light status
    Goto DONE
SETUP:
    Save display unit
    SETUP1:
        Get the value from the table
        Save the value in local variable
        Get the next byte
        Save it
        Get the third byte
        Save it
        Get the last byte
        Save it for conversion
        Normalize the floating point number
        Get the round flag
        Push accumulator A on stack
        Increment round flag
        Convert to hex
        Pull A off the stack
        Save A in round flag
        Get display value
        Get the units
        Call DSPDTA to display the data
DONE:
    If not in calibrate then goto LITTBR
    Turn lights off
    Goto OUTDSP
LITTBR:
    Call STRBAR to turn on bar
OUTDSP:
    Return to caller
```

DSPDTA: routine to write the display
- Save the units
- Set up display control
- If no error from setpoint entry routine then goto DT1
- Set error number
- Goto DSPERR, process error
- DT1:
  - Get floating point to hex result code
  - Get the least significant byte
  - If byte is equal to '2' then goto DSPDT4
  - If byte is not equal to 1 then goto DSPDT1
  - Set error number to 1, negative number
  - Goto DSPERR, process error
- DSPDT4:
  - Set error number to 6, number too large to be displayed
  - Goto DSPERR, process error
- DSPDT1:
  - Get the exponent
  - If exponent is not blank then goto DSPDT1A
  - Get the value
  - Goto STOREXP
- DSPT1A:
  - If exponent is not negative then goto PLUSEXP
  - Set exponent negative
  - Goto STOREXP
- PLUSEXP:
  - Set exponent positive
- STOREXP:
  - Move the exponent
  - Clear the decimal point
  - Get address of the display
- DSPDT2:
  - Point past the sign byte
  - Get the control value
  - Save the control value in the display control
  - OR in the decimal point
  - Write the data
  - Set decimal point to 010H
  - If all digits are displayed (display control value equal 0) then goto DSPDT9
  - Decrement control value
  - Get the value
  - Mask the next reference digit
  - Get the control value
  - OR in the decimal point
  - Save control value in display control value
  - Output the value

```
        Decrement the control value
        Goto DSPDT2
    DSPDT9:
        Get the current status of the lights
        Turn off unit lights
        Put in new lights
        Save status of lights
        Return to caller

DSPERR: routine to display error code in accumulator B
    Output selected error
    Prepare error code for display
    Initialize decimal point to 10H
    Initialize control value to '4'
    Get display value in X
    Call DSPDT2 to write the value to the display
    Goto DONE

STRBAR: bar graph routine
    If not in calibrate mode then goto NOINC3
    Get calibrate value
    Save it in bar graph value
    Goto ANOTH
    NOINC3:
        Get current bar graph parameter
        Save it in bar graph value
    ANOTH:
        Get break point value
        If bar graph value is less than breakpoint then goto FNDLO
        If bar graph value is equal to breakpoint then goto FNDEQ
        If the last value then goto FNDEQ
        Point to next reference entry
        Get next reference breakpoint table entry
        Get color code entry
        Save address of next color code entry
        Goto ANOTH
    FNDLO:
        X now points to breakpoint just higher than the current value
        Average breakpoint and next lower breakpoint
        If bar graph value is less than the average then goto FNDEQ and use
         lower breakpoint
        Else use the higher breakpoint
```

```
FNDEQ:
    X now points to the desired table entry
    Get first light group code
    Get second code
    Get third code
    If high alarm in state 1 or 2 then goto RED
    If warn alarm not in state 1 or 2 then goto GREEN
    Goto AMBER
AMBER:
    Call GON
    Call RON
    Goto BARDON
GREEN:
    Call ROFF
    Call GON
    Goto BARDON
RED:
    Call GOFF
    Call RON
BARDON:
    Return to caller
```

```
GON: routine to turn lights green
    Get bar register 1 value
    Save it in green register 1
    Get bar register 2 value
    Save it in green register 2
    Get bar register 3 value
    Save it in green register 3
    Return to caller

GOFF: routine to turn green off
    Set green register 1 to 0FFH
    Set green register 2 to 0FFH
    Set green register 3 to 0FFH
    Return to caller

RON: routine to turn red on
    Get bar register 1 value
    Save it in red register 1
    Get bar register 2 value
    Save it in red register 2
    Get bar register 3 value
    Save it in red register 3
    Return to caller
```

ROFF: routine to turn red off
    Set red register 1 to 0FFH
    Set red register 2 to 0FFH
    Set red register 3 to 0FFH
    Return to caller

BARFUL: routine to turn all bars red
    Get code table for all bars
    Save first code in bar register 1
    Save second code in bar register 2
    Save third code in bar register 3
    Call RED
    Return to caller

DSPINI: initialization routine
    Initialize variable to '2.0'
    Determine detector
    Get starting address of the pointers table
    Call ATOX
    Save bar graph starting value for this detector
    Save bar graph ending value for this detector
    Get the color code table
    Get number of bars lit on low value
    If none then goto COLDON
    COLLUP:
        Offset by 3 for each entry
        Decrement the number of bars
        If number of bars is not equal to zero then goto COLLUP
    COLDON:
        Increment offset by 3 to point to 1 just after
        Save address of first light code
        Get address of table begin
    BRILUP:
        Save the limit
        Compare the table entry to the low limit
        If table entry is greater than or equal to low limit then goto GOTLO
        Point to next entry
        Goto BRILUP
    GOTLO:
        Point to next entry in table, which is the first entry to use
        Save in first entry variable
    HIILUP:
        Compare table entry to the high limit
        If table entry is greater than or equal to the high limit then goto GOTHI
        Point to next entry in table
        Goto HIILUP

```
GOTHI:
    Save address of last value
    Set first color to green
    Goto DONE

BARPAR:
    Define C1MPH, the pointer to displayed value
    TABBEG:
        Define breakpoint table
    TABEND: end of breakpoint table
    COLTAB:
        Define color code table
        ALLON:
            Define all bars on
    COLEND:  end of color table
```

6.2.15 Eesvc.s – EEPROM update routine

```
Initialize variables
EESVC:
    While not end of setpoint table do begin
        Get setpoint
        If setpoint not modified then goto ENT2
        Make a copy of the new setpoint
        Call ERSET to erase location to be written into
        Call WRSET to write into EEPROM
        ENT2:
            Update table pointer
            Increment setpoint number
    End while
    Return to caller

RDSET:
    Get address of E2
    Call READ to read first two words
    Return to caller

READ:
    Call COMMAND
    Call CLK  to clock in next bit
    Initialize byte count to 2
    While byte count is greater than zero do begin
      Initialize bit count to 8
      While bit count is greater than zero do begin
        Get the bit
        Clock in next bit by calling CLK
          Decrement bit count
      End while loop on bit count
       Decrement byte count
       Clock in next bit by calling CLK
    End while loop on byte count
    Return to caller

WRSET:
    Get  the E2 address
    Call WRITE to write the first word
    Call W10M to wait 10 msec.
    Get address of nex word
    Call WRITE to write the second word
    Call W10M to wait 10 msec.
    Return to caller
```

```
WRITE:
    Call COMMAND
    Initialize byte count to 2
    While byte count is greater than zero do begin
        Initialize bit count to 8
        While bit count is greater than zero do begin
            Get the bit
            Clock in next bit by calling CLK
            Decrement bit count
        End while loop on bit count
        Decrement byte count
        Clock in next bit by calling CLK
    End while loop on byte count
    Return to caller

EWEN: Sets up EEPROM
    Initializes address to 30H
    Call COMMAND
    Set the bit
    Call CLK to clock in next bit
     Returns to caller

EWDS:
    Initialize address to 00H
    Call COMMAND
    Set  the bit
    Call CLK to clock in next bit
    Return to caller

ERSET:
    Get the E2 address
    Call ERASE to erase the first word
    Call W10M to wait 10 msec
    Increment the address
    Call ERASE to erase the second word
    Call W10M to wait 10 msec
    Return to caller

ERASE:
    Call COMMAND
    Set bit
    Call CLK to clock next bit
    Return to caller
```

```
ERAL: erases all setpoints
    Call COMMAND
    Set bit
    Call CLK to clock next bit
    Return to caller

WRAL: write to all registers
    Initialize address to 10H
    Call COMMAND
    Initialize byte count to 2
    While byte count is greater than zero do begin
        Initialize bit count to 8
        While bit count is greater than zero do begin
            Get the bit
            Clock in next bit by calling CLK
              Decrement bit count
         End while loop on bit count
         Decrement byte count
         Clock in next bit by calling CLK
    End while loop on byte count
    Return to caller

COMMAND: send command the the E2PROM
    Input B contains the command start bit plus opcode
    Input address contains the byte address

    Initialize bit count to 4 for 4-bit opcode
    Enable the chip
    Set the bit
    While bit count is greater than zero do begin
        Get the next bit
        Call CLK to clock in bit
        Decrement bit count
    End while
    Initialize bit count to 6 for 6-bit address
    Get the address
    While bit count is greater than zero do begin
        Send the next bit
        Call CLK to clock the bit
        Decrement bit count
    End while
    Return to caller
```

```
CLK: Clocks data bit
    Bring CE high
    Bring CE low
    Return to caller

W10M: wait for 10 msec
    Initialize counter to 995
    While counter greater than zero do begin
        Decrement counter
    End while
    Return to caller

ENTIN:  initialize setpoints
    Call EWEN to set up EEPROM
    Get data entry buttons
    Save only enter bit
    Set init flag
    If not defaulting EEPROM goto EN1
    Call ERAL to erase all setpoints
    Call W10M to wait for 10 msec
    EN1:
        Get top of setpoint table
        Initialize setpoint number to zero
        While not end of setpoint table do begin
            Save table pointer
            Get setpoint definition
            If  setpoint is unused goto EN4
            If storing default setpoint goto EN3
            Get default value
            Call WRSET to erase the location to be written into
            EN3:
                Get setpoint number
                Call RDSET to copy data from EEPROM to RAM
            EN4:
                Get table pointer
                Point to next entry
                Increase setpoint number
        End while
        Set loop address for COM422
        Return to caller
```

6.2.16 Entsvc.s – Services data entry buttons

Initialize variables

ENTSVC:

Get function switch position
Call ATOX
Set switch position
Get the previous status
Save previous status
Read data entry register
Determine which buttons were pushed – buttons are low true
If bit was entered then goto ENTBUT
If value was selected then goto SELBUT
If digit was selected then goto DIGBUT

RET:

If not enter mode then goto EDTX
Get quarter second timer
If timer not expired then goto EDT1
Move the value to work area
Blank the digit
Clear the units
Point to modified data
Call DSPDTA to display the data
Goto EDTX

EDT1:

Clear units indicator
Point to display data
Goto DSPDTA to display data

EDTX:

Return to caller

FUNSW: Function switch decode table

ENTBUT: enter button routine

If not already in enter mode then goto ENT1
Increment enter mode flag
Get switch position
Point to setpoint table
Indicate read function
Call DBASUB to read setpoint
If setpoint is defined goto ISBDEF
Indicate error
Return to caller

```
ISBDEF:
    Round up value
    Convert value to display format
    If no error occurred then goto ENTA
    Set to error value
ENTA:
    Point to digit parameters
    Initialize digit table pointer
    Set up blank digit mask
    Get current position
    If positon is not to be calibrated then goto NOICP
    Set calibrate status to 1

NOICP:
    Return to caller

ENT1: routine to enter new setpoint
    If setpoint is undefined then goto ERROR7
    Convert setpoiint to floating point value
    Put in setpoint table
    Set function code to switch position
    If not last function then goto SETCID
    Indicate write function
    Call DBASUB to store setpoint in RAM and EEPROM
    If not calibration mode then goto NINC6
    Update calibration status
    NINC6:
        Reset the enter flag
        Reset the error flag
        Return to caller

SETCID: routine to store new loop address
    If two digits goto SC2
    Set exponent to 41H for one digit
    Store the exponent
    Store the mantissa
    Right justify the mantissa
    Store integer loop address
    Set the modify flag
    Goto NINC6
    SC2:
        Store integer loop address
        Set exponent to 42H for two digits
        Store exponent
        Store mantissa
        Set modify flag
        Goto NINC6
```

DIGBUT: routine to select next digit to edit
- Get digit table pointer
- Point to next entry
- Get blank mask
- Store new table pointer
- Goto RET

SELBUT: routine to roll selected digit
- Get digital table pointer
- Point to roll routine
- If invalid address then goto RET
- RD1:
  - Get left and right digits
  - Increment left digit
  - If in range then goto RD1A
  - Sub zero first
  - RD1A:
    - Combine left and right digits
    - Update display data
    - Return to caller (RTS)

- RD2:
  - Get left and right digits
  - Increment right digit
  - If in range then goto RD2A
  - Sub low value
  - RD2A:
    - Combine left and right digits
    - Update display data
    - Return to caller (RTS)

- RD3:
  - Get left and right digits
  - Increment left digit
  - If in range then goto RD3A
  - Sub zero first
  - RD3A:
    - Combine left and right digits
    - Update display data
    - Return to caller (RTS)

```
RD4:
    Get left and right digits
    Increment right digit
    If in range then goto RD4A
    Sub low value
    RD4A:
        Combine left and right digits
        Update display data
        Return to caller (RTS)

RD5: (exponent)
    Get left and right digits
    Increment left digit
    If equal to +0 then goto RD5A
    Set to +0
    RD5A:
        Combine left and right digits
        Update display data
        Return to caller (RTS)
RD6:
    Get left and right digits
    Increment right digit
    If in range then goto RD4A
    Sub low value
    RD4A:
        Combine left and right digits
        Update display data
        Return to caller (RTS)

BLANK:
    Get blank mask
    Point to display data
    While not last digit pair do begin
        If left digit selected then begin
            Blank left digit
            Place digit in data
        End if
        If right digit selected then begin
            Blank right digit
            Modify display data
        End if
    End while
    Return to caller (RTS)

DP: digital table declaration
```

6.2.17 Error.s – Error routine

```
ERROR:
    If over range is less than under range then goto ERRV1
    If over range is less than warn then goto ERRV1
    If over range is less than high alarm then goto ERRV2
    ERRV1:
        Set error to invalid setpoint error code
        Goto ERRV3
    ERRV2:
        If not analog error goto ERRV3
        Set error to analog error code
    ERRV3:
        Return to caller
```

6.2.18 Host485.s – Message reply routine

Initialize variables

HOST485:

Get pointer to port parameters
Get pointer to output buffer address
Call SETAP to initialize output buffer
Get pointer to input buffer
Get command value from input buffer
Save command in local variable
Get subcommand value
Save subcommand value in local variable
If command not equal '10H' then goto NOTSTAT
If subcommand not equal '40H' then goto NOTSTAT
Goto CURSTAT, it is the current status command

NOTSTAT:

If command not equal zero then goto CMDERR
If subcommand is less than '0H' then goto SBCERR
If subcommand is less than or equal to '0FH' then goto REQSTSP
If subcommand is less than '10H' then goto SBCERR
If subcommand is greater than '1FH' then goto OTHERCMD
Goto SETSTSP

OTHERCMD:

If subcommand not equal to '22H' then goto CHKSOFF
Goto CKSON, to execute checksource on routine

CHKSOFF:

If subcommand not equal to '32H' then goto ALARMACK
Goto CKSOFF to execute checksource off routine

ALARMACK:

If subcommand not equal to '43H' then goto CURSTAT
Goto ACK to execute alarm acknowledge routine

CURSTAT:

If subcommand not equal to '40H' then goto CURRVAL
Goto CURSTA to execute current status command

CURRVAL:

If subcommand not equal to '41H' then goto ERRMSG
Goto CHVAL to execute current value command

ERRMSG:

Goto SBCERR

EXIT:

Return to interrupt service routine

DTAERR:

Get pointer to invalid data error
Goto EXIT

SBCERR:

Get pointer to invalid subcommand or function error
Goto EXIT

```
CMDERR:
    Get pointer to invalid command or channel error
    Goto EXIT
ACKA:
    Get pointer to acknowledge processor respond with data return point
    Goto EXIT
ACKN:
    Get pointer to acknowledge processor respond with 'ack' return point
    Goto EXIT

REQSTSP: routine read the specific setpoint value and moves the value to
output buffer
    Get pointer to setpont definition table
    Call ATOX
    Point to the setpoint
    If the setpoint value is equal to zero then goto SBCERR
    Get pointer to port parameters
    Get pointer to output buffer
    Call STORE
    Goto ACKA
    STORE:
        For all four parts of the setpoint value do begin
            Save setpoint value in local variable
            Call STOCHR
        End for
        Return to caller

SETSTSP: routine to set specific setpoint to the value specified by the data
    Get pointer to port parameters
    Get pointer to input buffer
    Get data offset
    Call STOX
    Get setpoint number from subcommand
    Get setpoint value
    Set carry to indicate write setpoint
    Call DBASUB to write setpoint
    If carry bit is set then goto SBCERR
    If overflow bit is set then goto DTAERR
    Goto ACKN, successful completion of a command

CKSON: enables checksource
    Turn on the checksource flag
    Update remote control commands status
    CKSONX:
        Goto ACKN
```

CKSOFF: disables checksource
    Turn off checksource flag
    Update remote control commands status
    CKSOFX:
        Goto ACKN

ACK: acknowledge alarms routine
    If high or warn alarm has been acknowledged then goto NOUNACK
    If acknowledge counter is greater than or equal to '1' then goto NOUNACK
    Increment acknowledge counter
    Turn on alarm acknowledge flag
    Save remote control commands
    Release the rate relay
    NOUNACK:
        Goto ACKN

CURSTA: this command retrieves the status
    Get pointer to ACIA port parameters
    Get pointer to output buffer
    Get the high alarm status, lower high of byte #0
    Call STOCHR to store BYTE #0 in the output buffer
    Get checksource status
    Shift checksource bit left 6 times for compatibility with U942
    OR in the warn status bit
    Call STOCHR to store byte #1 into output buffer
    Get under range/no counts fail flag
    Shift it left 4 times, this will be upper half of byte #2
    OR in the over range/jam fail flag for the lower half of byte #2
    Call STOCHR to store byte #2 in the output buffer
    Clear accumulator A
    Call STOCHR to store '0' in byte #3
    Call STOCHR to store '0' in byte #4
    Call STOCHR to store '0' in byte #5
    Call STOCHR to store '0' in byte #6
    Call STOCHR to store '0' in byte #7
    Call STOCHR to store '0' in byte #8
    Goto ACKA

CHVAL: this command retrieves the current one minute average value

```
Access the minute queue structure pointers
Set the first pointer to top pointer minus 5
Set the last pointer to the bottom pointer
Set pointer to the current pointer minus 5
If first pointer is not equal to pointer then goto NOTLIMIT
Set pointer equal to last pointer
NOTLIMIT:
    Transfer the contents of pointer to temp variable
    If background subtract option is present then goto BKGSUB
    Store temp as MPH
    Goto POSVAL
BKGSUB:
    Subtract the background
    If the result is greater than zero then goto POSVAL
    Set first byte of temp960 to '4000H'
    Set second byte of temp960 to '0'
    Goto FLPCNVRT to convert the hex value to floating point
POSVAL:
    Get the monitor status
    If the conversion constant is present then goto CNVCONST
    Save temp in temp960
    Goto FLPCNVRT, to convert the hex value to floating point
CNVCONST:
    Multiply in the conversion constant
    Result is 4 bytes of hex data
    Get first byte of result
    Call STOCHR
    Get second byte of result
    Call STOCHR
    Get third byte of result
    Call STOCHR
    Get fourth byte of result
    Clear lower nibble so that the quality tag is clean
    Get the checksource status
    If checksource is off then goto NOQ
    Load accumulator A with '02H'
    Goto STORED4
    NOQ:
        Get the tag
        Isolate the tag
        If tag is no data ('07H') then goto STORED4
        Clear the tag
    STORED4:
        OR the third byte with accumulator A
        Call STOCHR
```

FLPCNVRT: converted hex value to floating point in version A. Revision B leaves the
data in hex.

Goto ACKA

6.2.19 Idspsvc.s – Integer display conversion routine

```
Initialize variables
INTDSP:
    If display option is not present then return to caller
INOPR:
    Get current value
    If in under range goto INRNG1
        Else goto DSPZRO
    INRNG1:
        If in over range goto INRNG2
        Else goto DSPOVR
    INRNG2:
        If in JAM condition goto  INRN22
        Else goto DSPOVR

    INRN22:
        Set flag to display in mR/hr
        If number is greater than 99.9 goto RNGIST
        Set flag to display in R/hr
        Convert number to R/hr
        If number is greater than 99.9 goto RNGIST
        Set flag to display in kR/hr
        Convert number to kR/hr
    RNGIST:
        Set flag to round number
        Convert number to BCD
        Get sign of exponent
        Save a copy
        Isolate the sign
        Isolate the exponent digit
        If sign is negative goto NEGSIN
        Get first digit
        If  not digit the goto GOTSIN
        Decrement index by one
        Goto GOTSIN
    NEGSIN:
        Get exponent
    GNLUP:
        Decrement exponent
        If equal to zero goto GOTSIN
        Store a zero
        Increment index by one
        Goto GNLUP
```

GOTSIN:

- Get 1^{st} and 2^{nd} digits
- Save 1^{st} digit
- Save 2^{nd} digit
- Get 3^{rd} digit
- Save 3^{rd} digit

STOVAL:

- Store integer display value
- Store integer display units
- Return to caller

DSPZPRO:

- Set number value to zero
- Goto STOVAL

DSPOVR:

- Set all digits to E
- Goto STOVAL

INTINI:

- Get monitor option byte
- If integer display option then goto PINTOP
- Set flag to option not present

PINTOP:

- Blank the first digit
- Store value of zero
- Blank the last digit
- Initialize floating point variable for 1000 to 1000.0
- Set display flag to display in mR/hr
- Set old CMPH to 0
- Filter at 1.0
- Initialize floating point variable for 99p9 to 99.9
- Return to caller

6.2.20 Inthnd.s – Interrupt handler

```
INTHND:
    Get pointer to ACIA table address
    INT2:
        While not end of ACIA table do begin
            Get the address of ACIA pointed to
            If address is zero then goto NOINT
            Get the contents of the status register
            If this did not cause an interrupt then goto NOINT
            Get ACIA buffer
            If no SCANRAD then goto NOTSRCOM
            Set pointer to the active PCB
            NOTSRCOM:
                Get the subroutine address
                Service the ACIA in turn
            NOINT:
                Get the next location in the ACIA table
        End while
        Return to caller
```

6.2.21 Libra.s – routines to support standard RS232 communications. These routines are not to be used by interrupt driven functions such as: NMI, COMSVC, THMSUB or SWISVC.

Define variables
LIBRA: subroutine jump table
- AVERAG:
  - Goto AVGQ, to average queue
- FSPQ:
  - Goto FSP, to forward space pointer
- LOCATE:
  - Goto LOCQ, to find queue value
- INITQ:
  - Goto INIQ, to initialize queue
- CLEARQ:
  - Goto CLRQ, to clear queue
- BKSPQ:
  - Goto BKSP, to backspace queue
- SRCHQ:
  - Goto SEARCH, to search queue
- DEADT:
  - Goto DEAD, to do deadtime calculation
- SQRT:
  - Goto RADX, to calculate square root
- CLEAR:
  - Goto CLRMEM, to clear memory block

ATOX: routine to add value in accumulator A to X, accumulator value is preserved.
- Save value from accumulator A in local variable
- Save value from accumulator B in local variable
- ACCOM:
  - Get high order of X in accumulator B
  - Add accumulator A to low order of X
  - Add carry to high order
  - Save high order
  - Save low order
  - Put result in X
  - Restore accumulator A
  - Restore accumulator B
  - Return to caller
- AX2TOX: adds 2 times value in accumulator A to X
  - Save accumulator A in local variable
  - Save accumulator B in local variable
  - Multiple accumulator A by 2
  - Goto ACCOM

AX4TOX: adds 4 times value in accumulator A to X
Save accumulator A in local variable
Save accumulator B in local variable
Multiple accumulator A by 4
Goto ACCOM

BTOX: routine to add value in accumulator B to X, accumulator value is preserved.
Save accumulator B in local variable
Save accumulator A in local variable
BCCOM:
Get high order of X in accumulator A
Add accumulator B to low order of X
Add carry to high order
Save high order
Save low order
Put result in X
Restore accumulator B
Restore accumulator A
Return to caller
BX2TOX: adds 2 times value in accumulator A to X
Save accumulator B in local variable
Save accumulator A in local variable
Multiple accumulator B by 2
Goto BCCOM
BX4TOX: adds 4 times value in accumulator A to X
Save accumulator B in local variable
Save accumulator A in local variable
Multiple accumulator B by 4
Goto BCCOM

CLRMEM: routine to clear memory block. Input: index register X contains starting address, accumulator A contains length (1-128). Output: Memory block cleared to zero, accumulator A, accumulator B and index register X are wiped out.

While length is not equal to 0 do begin
Clear X
Increment X
Decrement length
End while
Return to caller

SETUP: routine to setup queue pointer locations. Input: index register X points to queue header

- Save X in local variable
- Get queue header
- Set FIRST equal to queue header
- Set LAST equal to queue header plus 2
- Set PTR (queue write pointer) equal to queue header plus 4
- Set TOTAL (queue total) equal to queue header plus 6
- Return to caller

NODATA: defines value for no data as double word '04000H', '0H'

INIQ: routine initialize queue, index register X points to queue descriptor

- Save the descriptor address in POINT1 (prototype header)
- Get queue header address
- Save it in POINT2 (queue header in RAM)
- Move prototype header to queue header in RAM
- CLRQ: set all values to no data
  - Save POINT1 on the stack
  - Save POINT2 on the stack
  - Save queue header address
  - Get pointer to first element
  - Save it
  - Get pointer to last element
  - Save it
  - Start with last element
  - While not first element do begin
    - Set element to zero
    - Set tag to NODATA
    - Decrement element pointer
  - End while
  - Pull POINTER1 off of stack
  - Pull POINTER2 off of stack
  - Return to caller

AVGQ: update historical section. Inputs: index register X points to buffer header,

- QVAL is tagged section.
- Output: queue total computed and QAVG (average value)
- Buffer format: (0) Top Pointer, (2) Bottom Pointer, (4) Oldest Value, (6) Total. The new value replaces the oldest value.
- Call SETUP to set up the queue pointers
- Point to the oldest value
- Replace it with the new value
- Replace the tag
- Call FSPA to bump queue pointer

Clear the counter
Point to the first value
Get the value
If value is not equal to zero then goto AVG1
Increment counter
AVG1:
    Point to last element
AVG2:
    Get the value
    Add the two values together
    Save the result
    If value is not equal to zero then goto AVG3
    Increment counter
AVG3:
    Backspace the value pointer
    If value pointer not equal to first value then goto AVG2
AVG4:
    Get pointer to queue total
    Rewrite it to local variable
    Get counter
    Update counter
    Initialize QAVG to NODATA
    Get good value count
    If counter equals zero then goto AVGX
    Clear temporary variable used to float counter
    Float the counter
    Divide the total value by the counter
AVGX:
Return to counter

LOCQ: calculate address of section circular buffer. Input: accumulator A contains the index, index register X points to buffer header. Output: index register X points to the result

Call SETUP
LOCQA:
    Save the index
    Calculate index times 5
    Save result in TEMP
    Get queue write pointer
    Get pointer to first element
    Subtract first element pointer from queue write pointer
    Save result in TEMP1
    If TEMP1 exponent is equal to TEMP exponent then goto LOC2
    If TEMP1 exponent is greater than TEMP exponent then goto LOC1
    If TEMP1 mantissa is less than TEMP mantissa then goto LOC2

```
LOC1:
    Subtract TEMP from TEMP1
    Save result in TEMP1
    Goto LOC3
LOC2:
    Subtract TEMP1 from TEMP
    Save result in TEMP
    Subtract TEMP from last element
    Save result in TEMP1
LOC3:
    Save TEMP1 in index register X
    Restore accumulator A
    Return to caller
```

SEARCH: searches queue from current value back until value is equal to QVAL. Input: QVAL. Output: index of value relative to newest value is in accumulator A.

```
Call SETUP
Clear counter
Get search argument
Save it
If it is equal to zero then goto SRCH3
Call BKSPA to point to nearest value
Call BKSPA to point to previous value
Set counter to '1'
SRCH1:
    If the current value is equal to zero then goto SRCH3
    Call COMPAR
    If the argument value and current value are equal then goto SRCH2
    Get pointer to header
    Get current pointer
    If the pointers are the same then goto SRCH3
    Increment the counter
    Call BKSPA to point to nearest value
    Goto SRCH1
SRCH2:
    Get the count in accumulator A
    Clear the carry bit to indicate success
    Return to caller
SRCH3:
    Get the count in accumulator A
    Set the carry bit to indicate not found
    Return to caller
```

BKSP: routine to backspace circular buffer pointer
- Call SETUP to get queue pointers
- BKSPA:
  - Get current pointer
  - If current pointer is equal to first pointer then goto BKSP1
  - Update current pointer to point to previous element
  - Goto BKSPX
- BKSP1:
  - Set current pointer to last element
- BKSPX:
  - Save current pointer in index register X
  - Return to caller

FSP:
- Call SETUP to setup queue pointers
- FSPA:
  - Get current pointer
  - If current pointer is not equal to last pointer then goto FSP1
  - Set current pointer to first pointer
  - Goto FSP2
  - FSP1:
    - Update the current to point to next element
  - FSP2:
    - Get pointer to the queue header
    - Get current pointer
    - Rewrite the slot pointer
    - Save new current pointer in index register X
    - Return to caller

COMPAR: compare two floating point numbers, positive exponents is assumed.

- If first byte of float 1 is not equal to first byte of float 2 then goto COMPX
- If second byte of float 1 is not equal to second byte of float 2 then goto COMPX
- If third byte of float 1 is not equal to third byte of float 2 then goto COMPX
- If fourth byte of float 1 is not equal to fourth byte of float 2 then goto COMPX
- COMPX:
  - Return to caller , carry set indicates numbers are equal

DEAD: routine to compute dead time correction factor, Input: index register X contains pointer to setpoints in RAM.

Save RAM pointer
Set up variable equal to floating point '1'
Get RAM pointer
Multiply counts per minute value by resolve time in minutes per count
Subtract result of multiply from floating '1'
Dead time correction factor is equal to floating '1' divided by result of subtraction
If division result is not negative then goto NONEGC
Set dead time correction factor to '0'
NONEGC:
    Restore index register X
    Return to caller

RADX: routine to calculate the square root of a floating point number in excess 64 notation

Initialize variable to floating point '2'
Save input number
Get the exponent
Get first two hex digits
Keep the first digit
Get pointer to square root table
Compute mantissa times '4'
Call ATOX to find root in the table
Save square root of mantissa
Get exponent in accumulator A
Set the carry bit
Subtracting '039H' and carry from accumulator A
Convert to table index
Get pointer to table
Call ATOX to find root
Save square root of exponent
Multiply square root of exponent by square root of mantissa
Save result in SQRTX
Set iterations to '4'
SQRLUP:
    Divide the input number by SQRTX
    Move original SQRTX to work area
    Clear least significant nibble
    Move result of divide to work area
    Clear least significant nibble
    If SQRTX is equal to result of divide then goto FOUND
    Decrement iterations

If iterations is equal to zero goto FOUND
NOFND:
Add SQRTX to result of the divide
Set SQRTX to result of addition
Call SHIFT to divide by two
Goto SQRLUP
FOUND:
Set index register X to address of SQRTX work area
Return to caller

SHIFT: routine shifts one bit, normalizing if necessary
Save accumulator A on stack
Save accumulator B on stack
Get address of SQRTX work area
If exponent is not zero then goto NOZ
If mantissa is not zero then goto NOZ
If equal to zero then goto OUTTA
NOZ:
Shift the first byte of the manissa to the right
Save the carry flag in accumulator A
Mask out bottom nibble
If not equal to zero then goto GO_ON
Decrement the exponent
Restore the first byte of the mantissa
Initialize counter to '3'
TAKOFF:
Shift the bottom byte left
Shift the next byte with the carry left
Shift the next byte left
Decrement the counter
If counter is not equal to zero then goto TAKOFF
Goto OUTTA
GO_ON:
Bring back th carry
Shift one bit to the right
Shift one bit to the right
OUTTA:
Restore the stack
DONE:
Return to caller

SR: defines the square root table for X = 0 to 16

SR16: defines the square roots of the powers of 16

FP2DEFB: routine to convert floating point number to ASCII

Input: F2HDR contains the address of floating point number to convert index register X contains address of buffer to store conversion. Output: Buffer is loaded with converted number, if number is illegal (a negative number), all zeros are stored. Buffer pointer is pointing to next space in buffer. Buffer format 100+1 = 1.00E+01. Buffer format 123-4 = 1.23E-04

Save buffer address
Get input floating point number address
Make a copy of the floating point number
Get flag to round result
Put rounding flag on the stack
Increment rounding flag
Get buffer address
Call FP2HEX to convet to BCD characters
Restore old rounding flag
Get the BCD sign
If sign is not equal to 0 then goto F2AERR
Get first digit
Isolate the first digit
Make it ASCII
Save it in buffer
Adjust the decimal point
Print the decimal point
Save it in the buffer
Adjust the buffer pointer
Get the second digit
Isolate it
Make it ASCII
Save it in the buffer
Update the buffer pointer
Get the third digit
Isolate it
Make it ASCII
Save it in the buffer
Print space
Save it in the buffer
Update buffer pointer
Print 'E'
Save it in the buffer
Update buffer pointer
Get sign, first exponent digit
Isolate it

If it equals '00H' then goto F2AERR
If it equals '08H' then goto F2AERR
If it does not equal '07H' then goto F2ANN0
Store a minus sign in accumulator A for negative exponent
Goto F2AOV1
F2ANN0:
    Store a space in accumulator A for positive exponent
F2AOV1:
    Save sign in the buffer
    Update buffer pointer
    Get second exponent digit
    Isolate it
    Make it ASCII
    Save it
    Update buffer pointer
    Goto F2AXREF
F2AERR:
    Store '000+0' in buffer to indicate error condition
    Update buffer pointer
F2AXREF:
    Return to caller

EXPON: routine to calculate the exponent of an argument
    Set FP1 equal to '1.0'
    Set FP0_2 equal to '0.2'
    Initialize result YVAL equal to FP1
    Initialize intermediate result TVAL equal to FP1
    EXP1:
        Initialize L to '0'
        Initialize L+2 to '1'
        Initialize 1200H to '1202H'
    EXP3:
        Convert L to floating point
        Save in N (counter)
        Mulitply TVAL by input value and store in RVAL
        Divide RVAL by N and store in TVAL
        Get pointer to 1200H
        Copy RVAL to pointed at location
        Update pointer
        Copy TVAL to pointed at location
        Update pointer
        Save pointer
        If TVAL is less than FP0_2 then goto EXP7
        Add YVAL to TVAL and save in YVAL
        Increment count
        Goto EXP3

6.2.22 Nmi8.s – 8 Hz non-maskable interrupt service routine. This interrupt is connected to a clock which ticks 4 times per second. Various time dependent actions are performed: 1) gets counts from all routines that count and adds the count to the sum in RAM, which is looked at and cleared whenever the filter routine is run. This happens typically every second. 2) activates the clock function by putting a value in the run flag.

```
Initialize variables
NMI:
    Get backup flag to see if initialization is complete
    If initialization is complete then goto NMI1
    Goto NMI99
    Get the scalar address
    Call GETSC to get the counts
    If High Voltage then goto NMI30
    Load the scalar counts per second
    Goto NMI34
    NMI30:
        Get the High Voltage counts per second
    NMI34:
        Store the counts per second
    Call ADDSUM to add current counts to running total
    Call STRSM to save the counts on the 7th tick
    Call TICKER to schedule task as required
    NMI99:
        Set run flag to execute setpoint edit routine and
            RS232 communication command routines
        Return to caller

GETSC: routine to get counts
    Disable counting
    Read the count
    Clear the counter
    Reset clear bit
    Restore select bit
    Enable counting
    Return to caller

ADDSUM: routine to total counts
    Value to be added is stored in TEMP
    The address of the sum is pointed to by X
    Add current count to sum
    Result is returned in X
    Return to caller

STRSM: routine to store the counts on the 7th tick
    Get ticks
    If ticks are less than 7 then goto PAST
```

Store the count
PAST:
Return to caller

TICKER: routine to schedule tasks
Get the ticks
Increment ticks by one
If ticks are not equal to 8 then goto TICKX
Reset tick counter
Set run flag
TICKX:
Return to caller

6.2.23 Setedt.s – Setpoint validity check routine
Define the lower limit for conditional assembly
Initialize variables
SETEDT: re-entrant code
Push all used variables on to the stack
Determine the subroutine
Execute the subroutine
Pull everything off of the stack
Return to caller
Defines limit values

EDIT:
Defines jump table indexed by function switch position

EDITRES: edit resolve time
Goto SETOK
EDTCNV: edit conversion constant
Goto SETOK

EDTBKG: edit background
Goto SETOK

EDTCAL: edit calibration time
Goto SETOK

EDTRAT: edit rate of rise setpoint
Goto SETOK

EDTCID: ensures the number is an integer. Ensure number is integer by converting to integer and back to floating point.

Call FCMPER to compare number to 255
If greater than 255 goto CIDERR
Convert number into a 1-byte integer
Goto SETOK

CIDERR:
Call SETERC to indicate invalid value

EDTALR: high alarm
Call FCMPER to compare number to defined high value
If not valid then goto ALRBAD
Else goto SETOK
ALRBAD:
Goto SETERC to report error

EDTWRN: warn alarm
Call FCMPER to compare number to defined high value
If not valid then goto WRNBAD
Else goto SETOK
WRNBAD:
Goto SETERC to report error

EDTLSV:
Call CHKRNG to check for an even power of 10
If not a valid value then goto LSVBAD
Copy new low scale setpoint
Copy full scale setpoint
Call FCMPER to compare full scale setpoint to low scale setpoint
If full scale is less than low scale then goto LSVBAD
Goto SETOK
LSVBAD:
Goto SETERC to indicate invalid value

EDTFSV – high scale analog output
Call CHKRNG to check for an even power of 10
If not valid then goto FSVBAD
Copy new high scale setpoint
Copy low scale setpoint
Call FCMPER to compare full scale to low scale
If full scale is less than low scale then goto FSVBAD
Goto SETOK
FSVBAD:
Goto SETERC to indicate invalid value

EDTOVR: over range
Copy new over range setpoint
Copy appropriate 'zero' as defined at the beginning of the module
Call FCMPER to compare over range and 'zero'
If over range is less than 'zero' then goto OVRBAD
Copy '1E9"
Copy new over range
Call FCMPER to compare the two
If '1E9' is less than over range then goto OVRBAD
Copy new over range
Copy high alarm setpoint

```
        Call FCMPER to compare the over range and high alarm setpoints
        If over range is greater than or equal to high alarm then goto OVROK
            OVRBAD:
            Goto SETERC to indicate invalid value
        OVROK:
            Goto SETOK

    SETERC: routine to
        Set ERROT_4 to '3'
        Indicate error condition
        Return to caller

    SETOK:
        Clear error indicator
        Return to caller

    CHKRNG: routine to check for even power of ten
        Get start of power of ten table
        RNGLUP:
            Look for match in most significant digit
            If match is found then goto GOTRNG
            Get next entry in the table
            If not end of table then goto RNGLUP
            Indicate range not found
        GOTRNG:
            Return to caller

    FCMPER: routine to compare two real numbers
              Input:  FLT11 and FLT21
              Output:  result returned in condition flags
                       FLT11 = FLT21   carry flag clear, zero flag set
                       FLT11 > FLT21   carry flag clear, zero flag clear
                       FLT11 < FLT21   carry flag set, zero flag clear

        Get FLT11, argument 1 exponent
        Isolate sign bit
        Move it over
        Get FLT21, argument 2 exponent
        Isolate sign of mantissa
        Move it over
        If signs are not equal then goto FX
        Get argument 1 exponent
        Clear the mantissa sign
        Get argument 2 exponent
        Clear the mantissa sign
        If the exponents are not equal then goto FX
        Get argument 1 mantissa
        Get argument 2 mantissa
        If mantissas are not equal then goto FX
```

FX:
Return to caller

6.2.24 Swisvc.s – Supervisory services

Initialize variables and offsets

SVCTBL: table of addresses. The order number of the subroutine should be the same as the service number in the appropriate macro.

(0)	SVCEX	- breakpoint routine
(1)	PUSHX	- QED
(2)	PULLX	- QED
(3)	ADDBX	- Add accumulator B to index register X
(4)	SWAPDX	- Swap index register X and accumulator A, B
(5)	ADDX	- Add signed byte to index register X
(6)	SVCEX	- Add signed byte to S (NA)
(7)	SVCEX	- Build MR/HR
(8)	FADD	- Floating Add
(9)	FADD	- Floating subtract (uses add routine)
(10)	FMUL	- Floating Multiply
(11)	FDIV	- Floating divide
(12)	SVCEX	- Floating log (E) (NA)
(13)	SVCEX	- Floating exp (E) (NA)
(14)	FCMP	- Floating compare
(15)	FGETI	- Convert integer to float
(16)	FPUTI	- Convert float to integer
(17)	HEX2FP	- Convert display to float
(18)	FP2HEX	- Convert float to display
(19)	SVCEX	- Read a message (NA)
(20)	SVCEX	- Write a message (NA)
(21)	SVCEX	- Add a task to the queue (NA)
(22)	SVCEX	- Remove a task from the queue (NA)
(23)	SVCEX	- Load accumulator A from address (X+accumulator B)
(24)	SVCEX	- Store accumulator A to address (X+accumulator B)
(25)	FSTR	- Store argument 1 into argument 2
(26)	CMPLI	- Compliment argument 1
(27)	FNORM	- Normalize argument 1

SWISVC:
- Clear interrupts for COMSVC
- Get the SVC service code byte
- Double it into an SVCTBL offset
- If offset is less than the maximum offset then goto L100
- Goto SVCER0
- L100: calculates the offset into SVCTBL
  - Get the table address
  - Save the address
  - Get the low byte in accumulator A

Save accumulator A on the stack
Get the high byte in accumulator A
Save accumulator A on the stack
Add in the offset
Restore the stack
Get pointer to address of routine
Clean up the stack
Get the address of the routine
Jump to the routine

SVCEX: parameter list will always contain at least two arguments. The first argument will always be the number of bytes in the argument list. The second argument will be the service code byte.

Get section code space
Get the parameter list length byte
Get the stack address
Add the length to the stacked PC
Restore to stacked PC
Return from interrupt

PUSHX: routine to save information on the stack, entered with the stack pointing to the stacked registers.

Make a two byte space on the stack
Set counter to '7'
L200:
Move the stacked registers down two bytes
Increment the index register X
Decrement the counter
If all seven bytes of register have not been moved then goto L200
Copy index register X into the space just created
Goto SVCEX

PULLX: routine entered with stack pointing to the stacked registers
Copy the last 2 bytes pushed into the stacked X
Move the register stack up 2 bytes
Set the counter to '7'
L210:
Move the register stack up to bytes
Decrement the index register X
Decrement the counter
If counter is greater than zero then goto L210
Release the two bytes of stack
Goto SVCEX

ADDBX: routine entered with stack pointing to the stacked registers
Move stacked X into accumulator A and B
Add stacked B to A,B

Restore accumulators A, B to index register X
Goto SVCEX

SWAPDX: routine entered with stack pointing to the stacked registers
Swap lower byte of index register X with accumulator B
Swap higher byte of index register X with accumulator A
Goto SVCEX

ADDX: routine entered with stack pointing to the stacked registers
Clear accumulator B
Get the signed 8 bit argument
If it is greater than or equal to zero then goto L220
Decrement accumulator B
L220:
Add stacked index register X to accumulator A,B
Store accumulator A, B to stacked index register X
Goto SVCEX

FSTR: routine to store the floating pointing number from argument 1 into argument 2.

Get first and second bytes from argument 1
Store bytes in first and second bytes of argument 2
Get third and fourth bytes from argument 1
Store bytes in third and fourth bytes of argument 2
Goto SVCEX

FNORM: routine to normalize a floating point number
Get stack pointer
Get offset to stacked program counter
Get the number
Get fourth byte of the number
If not equal to zero then goto CHKNOR
Get next reference byte
If it is not equal to zero then goto CHKNOR
Get next reference byte
If it is not equal to zero then goto CHKNOR
Set exponent to zero
Goto NEXIT
CHKNOR:
Get the first byte of the mantissa in accumulator A
Save a copy in accumulator B
Mask off lower nibble
If equal to hex zero then goto NEXIT
Restore accumulator A
Shift left 4 bits
Get the second byte of the mantissa in accumulator B
Shift right 4 bits
Add the two nibbles together

Save the result
Get the second byte of mantissa in accumulator A
Shift it left 4 bits
Get the third byte of mantissa in accumulator B
Shift it right 4 bits
Add the nibbles together
Save the result
Get the third byte of mantissa in accumulator A
Shift it left 4 bits
Save it
Decrement the exponent
Goto CHKNOR

NEXIT:
Goto SVCEX

CMPLI: routine to take floating number pointed at by the index register and complement it.

Get stack pointer
Push argument on the stack
Get the argument
Do one's complement of first byte of mantissa
Do one's complement of second byte of mantissa
Do one's complement of third byte of mantissa
Increment third byte
If not equal to zero then goto CEXIT
Increment second byte
If not equal to zero then goto CEXIT
Increment first byte

CEXIT:
Get the exponent
Complement the sign
Save the new exponent

CFINE:
Goto SVCEX

FADD: routine to add two floating point numbers. Input: argument 1, argument 2. Output: argument 3; argument 1 is added to argument 2.

Get top of stack pointer
Push argument 1 on the stack
Clear accumulator A
Push accumulator A on stack as guard byte
Get third byte of mantissa for argument 1
Put it on the stack
Get second byte of mantissa for argument 1
Put it on the stack

Get first byte of mantissa for argument 1
Put it on the stack
Get the exponent for argument 1
Put it on the stack
Get stack pointer
Update stack pointer
Push argument 2 on the stack
Clear accumulator A
Push accumulator A on stack as guard byte
Get third byte of mantissa for argument 1
Put it on the stack
Get second byte of mantissa for argument 2

Put it on the stack
Get first byte of mantissa for argument 2
Put it on the stack
Get the exponent for argument 2
Put it on the stack
Update stack pointer
If most significant byte of argument 2 is equal to zero then goto LPAGE
Get service code
If not subtract then goto LPAGE
Complement the sign of argument 2
Do one 'no operation' statement
LPAGE:
    Put most significant byte of argument 2 on the stack
    Clear accumulator A
    Put accumulator A on stack as guard byte
    Put accumulator A on stack three times for the mantissa of the result
    Put '40H' on the stack for exponent of zero for result
    Get the stack pointer
FIRST: offset of argument 1 set to '10'
SECN: offset of argument 2 set to '5'
RES: offset of result set to '0'
Clear swapping flag
If argument 1 mantissa is not zero then goto NZ1
Increment swap flag to indicate argument 2 is bigger
Goto ADD_GO
NZ1:
    If argument 2 mantissa is not equal to zero then goto NZ2
    Goto ADD_GO
NZ2:
    Get argument 1 exponent
    Remove the sign
    Get argument 2 exponent
    Remove the sign
    If argument 1 exponent is greater than or equal argument 2 exponent
        then goto NOSWP

```
        Put argument 1 on the stack
        Increment swap flag
        Transfer from accumulator B to accumulator A
        Pull accumulator B from the stack
    NOSWP:
        Subtract accumlator B from accumulator A
        Save the number of shifts
        If number of shifts is less than '7' then goto GO_SHF
        If swap flag is equal to zero then goto ZSEC
        Get offset for argument 1
        Call AINCX to get the values
        Goto ZEROA

    ZSEC:
        Get offset for second argument
        Call AINCX
    ZEROA:
        Set the exponent to '40H'
        Clear byte 1
        Clear byte 2
        Clear byte 3
        Clear guard byte
        Goto ADD_GO
    GO_SHF:
        Get top of stack pointer
        Get exponent for argument 1
        Get magnitude of exponent
        Get exponent for argument 2
        Get magnitude of exponent
        If argument 1 is greater than argument 2 then goto SHFT2
        Get the offset off of the stack for argument 1
        Call ACINX to get the values
        Goto SHF_GO
    SHF2:
        Get offset off of the stack for argument 2
        Call ACINX to get the values
    SHF_GO:
        Get the number of shifts
        If the number of shifts equal zero then goto SHF_X
    SHF_TP:
        Initialize counter to '4'
    SHFRN:
        Shift most significant bit right 4 bits
        Rotate right next byte
        Rotate right least significant byte
        Rotate right guard byte
        Decrement counter
        If counter not equal to zero then goto SHFRN
        Increment the exponent
```

Decrement the number of shifts
If number of shifts is not equal to zero goto SHF_TP

SHF_X:
Get top of stack pointer
If sign of exponent for argument 1 is equal to sign of exponent for argument 2 then goto ADD_GO
Set counter to '4'

CTOP:
If byte of argument 1 not equal byte of argument 2 then goto COUT
Increment argument byte offset
Decrement counter
If counter not equal to zero then goto CTOP
Get stack pointer
Set exponent of result equal to '40H'
Set all three bytes of mantissa of result equal to '0'
Set guard byte to '0'
Goto CLEAN

COUT:
Get top of stack pointer
If carry is set goto COM1
Get exponent for argument 1
Save exponent in result
Get offset for second argument
Call AINCX for argument 2 values
Goto COMA

COM1:
Get exponent and sign for argument 2
Save exponent in result
Get offset for first argument
Call AINCX for argument 1 values

COMA:
Complement most significant byte
Complement next byte
Complement least significant byte
Complement guard byte
Increment the guard byte
If the carry bit is clear then goto AD_AS
Get byte 3 of mantissa
If carry bit is clear then goto AD_AS
Add '1'
Save byte 3
If carry is clear then goto AD_AS
Get byte 2 of mantissa
Add '1'
Save byte 2
If carry bit is clear then goto AD_AS

```
        Get byte 1 of the mantissa
        Add '1'
        Save byte 1
        Goto AD_AS
    ADD_GO:
        Get top of stack pointer
        Get first argument
        If swap flag is equal to zero then goto AD_G1
        Get second argument
    AD_G1:
        Save in results
    AD_AS:
        Get stack pointer
        Get the exponent of result
        Store it
        Add argument 1 guard byte to argument 2 guard byte
        Save in result guard byte
        Add argument 1 LSB to argument 2 LSB
        Save in result LSB
        Add argument 1 MID to argument 2 MID
        Save in result MID
        Add argument 1 MSB to argument 2 MSB
        Save in result MSB
        If carry is clear then goto CLEAN
        Save condition codes
        Get sign of first argument
        Get sign of second argument
        If the signs are not equal then goto CLEAN
        Restore the condition codes
        Get stack pointer
        Get the service code
        If subtract operation then goto CLEAN
        Save the condition codes
        Set counter to '4'
        Restore the condition codes
    OVSHF:
        Get pointer to top of stack
        Rotate right result byte 1 of mantissa
        Rotate right result byte 2 of mantissa
        Rotate right result byte 3 of mantissa
        Rotate right result guard byte
        Clear the carry bit
        Decrement the counter
        If counter is greater than zero then goto OVSHF
        Increment exponent of result
```

CLEAN:
Get top of stack pointer
Get pointer to argument 3
Get result exponent from the stack
Save it in argument 3 exponent
Get MSB of result from the stack
Save it in argument 3 MSB
Get MID of result from the stack
Save it in argument 3 MID
Get LSB of result from the stack
Save it in argument 3 LSB
Set counter equal to '11' to clean up stack

CLT:
Get value from stack
Decrement counter
If counter is not equal to zero then goto CLT
Goto SVCEX

AINCX: routine to increment index register X to offset passed in register B
If B is equal to zero then goto AIEX
Decrement B
Increment index register X
Goto AINCX
AIEX:
Return to caller

FSUB: stub

FMUL: routine to multiply two floating point numbers. Input: multiplicand, multiplier, and result. Multiplicand and multiplier are left unchanged, result is normalized.
Get pointer to top of stack
Get address of argument 2
Get byte 1 of argument 2 mantissa
Save it in index register X
Put it on the stack
Get byte 2 of argument 2 mantissa
Save it in index register X
Put it on the stack
Get byte 3 of argument 2 mantissa
Save it in index register X
Put it on the stack
Get pointer to top of stack
Get address of argument 3
Get byte 3 off of the stack
Save it in argument 3 byte 3

Get byte 2 off of the stack
Save it in argument 3 byte 2
Get byte 1 off of the stack
Save it in argument 3 byte 1
Save a 0 in argument 3 exponent
Get pointer to top of stack
Get pointer to argument 1
Save argument 1 expont on the stack
Save argument 1 byte 1 on the stack
Save argument 1 byte 2 on the stack
Save argument 1 byte 3 on the stack

```
MULINT;
    Set the loop counter to '25'
MULSTR:
    Get pointer to top of stack
    Get address of argument 1
    Rotate to the right byte 1 of argument 1
    Rotate to the right byte 2 of argument 1
    Rotate to the right byte 3 of argument 1
    Decrement the loop counter
    If loop counter is equal to zero then goto MULDN
    If the shifted bit in argument 1 is not set then goto SHFTR
    Get pointer to top of stack
    Get address of argument 2
    Get byte 3 of argument 2
    Get address of argument 3
    Add argument 1 byte 3 to argument 3 byte 3
    Save result in argument 3
    Get byte 2 of argument 2
    Add with carry byte 2 of argument 2 to byte 2 of argument 3
    Save result in argument 3
    Get byte 1 of argument 2
    Add with carry byte 1 of argument 2 to byte 1 of argument 3
    Save result in argument 3
SHFTR:
    Get pointer to top of stack
    Get address of argument 3
    Rotate to the right byte 1 of argument 3
    Rotate to the right byte 2 of argument 3
    Rotate to the right byte 3 of argument 3
    Goto MULSTR
MULDN:
    Get pointer to top of stack
    Get address of argument 1
    Get byte 3 off the stack
    Store in byte 3 of argument 1
```

Get byte 2 off the stack
Store in byte 2 of argument 1
Get byte 1 off the stack
Store in byte 1 of argument 1
Get exponent off the stack
Store in exponent for argument 1
Get pointer to top of stack
Get address of argument 3
Store '0' in exponent for argument 3
Set counter to '5' for number of nibbles
Put counter on the stack

NCHK:
Get byte 1 of mantissa for argument 3
Shift right
Shift right
Shift right
Shift right
If not equal to '000H' then goto SETS
Set counter to '3'
Clear carry bit

SHIFT:
Rotate left byte 3 pointed at by the index register
Rotate left byte 2 pointed at by the index register
Rotate left byte 1 pointed at by the index register
Decrement counter
If counter is greater than or equal to 0 then goto SHIFT
Get the exponent pointed at by the index register
Increment the exponent
Save it
Decrement number of nibbles
If number of nibbles is greater than or equal to 0 then goto NCHK

SETS:
Restore the stack
Get pointer to top of stack
Get address of argument 1
Get signed exponent for argument 1
Get address of argument 2
Get signed exponent for argument 2
Get the sign of the exponent for argument 1
Get the sign of the exponent for argument 2
Get address of argument 3
Store signed exponent of argument 1 in the result
If sign of exponent for argument 1 is equal to sign of exponent for argument 2 then goto PAGEN

STNEG:
OR the exponent of argument 3 with '10000000B'
Store the result in argument 3

PAGEN:

Get pointer to top of stack
Get address of argument 1
Put exponent for argument 1 in accumulator A
Get pointer to top of stack
Get address of argument 2
Put exponent for argument 2 in accumulator B
Mask out the sign for argument 1 exponent
Mask out the sign for argument 2 exponent
Add the two exponents together
Subtract '040H' from accumulator A
Get pointer to top of stack
Get address of argument 3
Put argument 3 exponent in accumulator B
Mask out the sign
If it is not equal to '06H' then goto MSEXP
Set exponent equal to '040H', real zero
Goto MEXIT

MSEXP:

Subtract accumulator B from accumulator A
Put exponent for argument 3 in accumulator B
Mask all but the sign bit
Save only the sign bit in exponent of argument 3
Or accumulator A with exponent of argument 3
Save result in argument 3

MEXIT:

Goto SVCEX

FDIV: data space

Defines ADDR
Call FPDIV
Defines argument 1
Defines argument 2
Defines result
Goto SVCEX

FPDIV: routine to divide floating point numbers. During execution of this routine the index register points to a block defined as:

Byte	0	Scratch
Byte	1	Overflow indicator
Byte	2-6	Result
Byte	7-11	Argument 2
Byte	12-16	Argument 1
Byte	17-18	PC of caller

The extra reference byte added to each argument is a guard byte used to maintain accuracy and is referred to as the overflow byte.

Set accumulator B to '1' to indicate divide
FPARG:
- Get pointer to top of stack
- Get address of calling sequence
- Get address of argument 1
- Save address
- Get first byte of argument 1
- If not equal to zero then goto OK
- If byte 1 is not equal to zero then goto OK
- If byte 2 is not equal to zero then goto OK
- If byte 3 is not equal to zero then goto OK
- Get pointer to top of stack
- Get address of calling sequence
- Get address of argument 3
- Store floating point zero in argument 3
- Restore stack to before call status
- Goto SVCEX

OK:
- Restore the index register
- Clear the overflow byte
- Put argument 1 LSB of mantissa on the stack
- Put argument 1 MID of mantissa on the stack
- Put argument 1 MSB of mantissa on the stack
- Save the function code on the stack
- Put exponent on the stack
- Make a copy of argument 1
- Subtract '040H' from it
- Remove the sign
- Save it in the index register X
- Get the original argument 1
- Get the sign
- Get the value of the exponent
- Restore the exponent
- Get the code off the stack
- Put the exponent on the stack
- Get pointer to top of stack
- Adjust stack pointer to address before call status
- Get address of argument 2
- Save zero on the stack
- Get LSB of mantissa for argument 2
- Put it on the stack
- Get MID of mantissa for argument 2
- Put it on the stack
- Get MSB of mantissa for argument 2
- Put it on the stack
- Save function code on the stack
- Get the exponent for argument 2
- Make a copy of it

Subtract '040H' from exponent
Remove the sign
Save the result in index register X
Get original exponent
Get the sign
Get the value of the exponent
Save the exponent
Get back the function code from the stack
Put the exponent on the stack
Clear space on the stack for result
Clear space on the stack for scratch space
Get the final index register pointer
If function code is equal to '1' for divide then goto DVSUB1

FPOUT:
If rounding is not to be done then goto X015
Get LSB of mantissa of result
Increment it
If carry not equal to zero then goto X015
Get MID of mantissa of result
Increment it
If carry not equal to zero then goto X015
Get MSB of mantissa of result
Increment it
If carry not equal to zero then goto X015
Set counter to '04H'

X014:
Rotate MSB to the right
Rotate MID to the right
Rotate LSB to the right
Rotate guard byte to the right
Clear the carry
Decrement the counter
If counter is not equal to zero then goto X014
Increment the exponent
If exponent overflow then goto FPOVF1
Rotate scratch area to the left
Rotate exponent to the right

X015:
Get overflow flag in accumulator B
Get address of result
Save a copy of it
Get the exponent's sign
If sign is equal to zero the goto POSX , sign is positive

NEGX:
Get the original exponent
Remove the exponent sign
Goto POSX1

POSX:

Get the original exponent
AND it with '040H' to make the XS64 exponent

POSX1:

Save the exponent
Save the MSB of the mantissa
Save the MID of the mantissa
Save the LSB of the mantissa
Set counter to '11'

X020:

Pop the stack
Decrement the counter
If the counter is not equal to zero then goto X020
Get the address of the stack
Get the return program counter
Clean the stack
If the overflow flag is equal to zero then goto X030
Set the overflow bit

X030:

Goto address pointed to by program counter

FPOVF1:

Goto FPOVF

DVSUB1:

Goto DVSUB2

ADSUB1:

Goto ADSUB2

ARG1:

Set argument 1 offset to '12'

ARG2:

Set argument 2 offset equal to '7'

ARG3:

Set argument 3 offset equal to '2'

RESULT:

Set result offset equal to '2'

FPOUT2:

Goto FPOUT

MULSUB:

Get argument 1
Goto NORMX1 to normalize it
Get argument 2
Goto NORMX1 to normalize it

FPM20:

Set loop counter to '25'
Clear scratch area
Clear the LSB

FPM30:
- Rotate MSB of argument 1 to the right
- Rotate MID of argument 1 to the right
- Rotate LSB of argument 1 to the right
- Decrement counter
- If counter is equal to zero then goto FPM40
- If shifted bit of argument 1 was not set then goto FPM35
- Add in LSB of argument 2
- Add in MID of argument 2
- Add in MSB of argument 2

FPM35:
- Rotate MSB of the result to the right
- Rotate MID of the result to the right
- Rotate the LSB in accumulator B to the right
- Rotate the guard byte to the right to save shift out in overflow
- Goto FPM30

FPM40:
- Save LSB of argument 2
- Get the sign of the result
- Make a copy
- Clear local overflow flag
- Get exponent in arithmetic form
- Add the exponents

FPM45:
- If exponent overflow then goto FPMOVF
- Clear the carry
- Rotate accumulator to the right

FPM37:
- Add in the sign
- Get the address of the result
- Goto NORMX1
- If overflow flag is not set then goto FPOUT2
- Increment the exponent

ADSUB2:
- Goto ADDSUB

FPMOVF:
- If underflow then goto FPUND
- Get the result
- If the overflow flag is not equal to zero then goto FPOVF
- If normalizing is not needed then goto FPOVF
- Set the exponent to '03FH', maximum for exponent
- Set local overflow
- Goto FPM37

NORMX1:
- Goto NORMX2

DVSUB2:
- Goto DVSUB3

FPOUT1:
    Goto FPOUT2
FPOVF:
    Mask out the sign by AND with '080H'
    Set exponent to the maximum value
    Set the overflow
    Goto FPMUL10
FPUND:
    Clear accumulator B to set exponent to zero
    Clear accumulator A to set fraction to zero
FPMUL10:
    Set the result fraction to zero
    Set the result exponent to zero
    Goto FPOUT1
FPM456:
    Goto FPM45
ADDSUB:
    Put the LSB on the stack
    Get the address of argument 1
    Goto NORMX2 to normalize argument 1
    Get the address of argument 2
    Goto NORMX2 to normalize argument 2
    Set counter to '8'
    Get argument 1 exponent
    Left justify it
    Get argument 2 exponent
    Left justify it
FPA05:
    Decrement the counter
    If counter is less than zero then goto FPA25
    If exponent from argument 1 is equal to exponent from argument 2
        then goto FPA30
    If exponent from argument 1 is greater than exponent from argument
        2 then goto FPA20, because argument 1 is bigger
    Call FPAS1 to shift argument 1 four times
    Add '2' to exponent from argument 1
    Goto FPA05
FPA20: argument 2 is smaller than argument 1
    CAll FPAS2 to shift argument 2 four times
    Add '2' to exponent form argument 2
    Goto FPA05
FPAS1: shift argument 1 one bit to the right
    Clear the carry
    Rotate each byte of argument 1 to the right
    Return to caller

```
FPAS2: shift argument 2 one bit to the right
    Clear the carry
    Rotate each byte of argument 2 to the right
    Return to caller
FPA25:
    If argument 1 exponent is greater than argument 2 exponent then
        goto FPA30
    Transfer argument 2 exponent to accumulator A
    Goto FPA30
FPOUT3:
    Goto FPOUT1
DVSUB3:
    Goto DVSUB4
NORMX2:
    Goto NORMX3
FPUND5:
    Goto FPUND
FPM455:
    Goto FPM456
FPOVF3:
    Goto FPOVF
FPA30:
    Save new exponent in the result
    Call FPAS1 to make room for sign in fraction for argument 1
    Call FPAS2 to make room for sign in fraction for argument 2
    If argument 1 is not negative then goto FPA40
FPA40:
    If argument 2 is not negative then goto FPA50
    Get the address of argument 2 in accumulator A
    Goto COMX1 to complement it
FPA50:
    Get function code from the stack
    If function code is not equal to '2' (add) then goto FPS10
    Add argument 1 to argument 2 byte by byte
    Store in result
FPA55:
    If there was no overflow then goto FPA60
    Save the MSB of the result
    Set counter to '4'
    Rotate MSB to the right to bring in lost bit
    Goto FPA58
FPA57:
    Shift MSB of the result to the right
FPA58:
    Rotate MID of result to the right
    Rotate LSB of result to the right
    Rotate guard byte of result to the right
    Decrement the counter
```

If counter is not equal to zero then goto FPA57
Increment the exponent of the result
If there is no overflow then goto FPA61
Rotate left MSB of result to set the overflow

FPA59:
Rotate argument 2 to the right

FPOVF2:
Goto FPOVF3

FPA80:
Goto FPOUT3

NORMX3:
Goto NORMX4

DVSUB4:
Goto DVSUB5

FPUND4:
Goto FPUND5

FPM454:
Goto FPM455

FPA60:
Save the MSB in result

FPA61:
If the result is not negative then goto FPA70
Get the address of the result
Call COMX1
Set the carry
If the result is too big to compare then goto FPA59

FPA70:
Shift the sign into the result
Move the sign back to the exponent
If the result is normalized then goto FPA80
Call NORMX to normalize the result
Goto FPA80

NORMX4:
Goto NORMX

DVSUB5:
Goto DVSUB6

FPUND3:
Goto FPUND4

FPM453:
Goto FPM454

COMX1:
Goto COMX

FPS10: subtract the two quadruple arguments
Get argument 1 exponent
Subtract argument 2 exponent
Save result

```
        Get argument 1 LSB
        Subtract argument 2 LSB
        Save result
        Get argument 1 MID
        Subtract argument 2 MID
        Save result
        Get argument 1 MSB
        Subtract argument 2 MSB
        If the carry bit is set, then goto FPS20
        Set the carry bit
        Goto FPA55
FPS20:
        Clear the carry bit
        Goto FPA55, to check the overflow and normalize
FPOVF4:
        Gotot FPOVF2

NORMX:  Nomalization routine, where accumulator A contains the offset
        to argument from normal X.  All registers except X are
        destroyed. V is set if underflow would occur if number is
        normalized.  Z is set if argument is equal to zero.

        Save index register X on stack
        Clear byte
NL1:
        Increment index register X
        Decrement accumulator A
        If not equal to zero, then goto NL1
        Check for zero
        If equal to zero, then goto NRMZRO
        Save sign in accumulator B with offset
        Double the exponent
NRM01:
        If normalized goto NRMOUT
        Decrement argument
        If V bit set, then goto NRMOVF, next shift will cause underflow
        Decrement argument
        Goto NRMSH4, to shift 4 bits
        Goto NRM01
NRMOVF:
        Increment argument, to restore the exponent
        Check overflow bit after the shift
NRMOUT:
        Put the sign back
NRMXIT:
        Get index register X off the stack
```

NRMLP:
- Decrement index register X
- Decrement accumulator A
- Get current status
- Save the interrupt status
- Return to caller

NRMSH4: routine to shift triple byte left 4 bits
- Set counter to '4'
- NSHLOP:
  - Shift the overflow byte left
  - Rotate left the LSB
  - Rotate left the MID
  - Rotate left the MSB
  - Decrement the counter
  - If counter is not equal to zero, then goto NSHLOP
  - Return to caller
- NRMZRO:
  - Set the 'Z' bit
  - Clear the exponent
  - Goto NRMXIT

DVSUB6:
- Goto DIVSUB

FPUND2:
- Goto FPUND3

FPM452:
- Goto FPM453

FPOVF5:
- Goto FPOVF4

COMX: Two's complement routine, where accumulator A contains the offset to argument from present X (0-64). Registers are lost.
- Save offset in accumulator B
- COM10:
  - Increment index register X
  - Decrement accumulator A
  - If not equal to zero, then goto COM10
  - Save condition codes
  - Complement MSB
  - Complement MID
  - Complement LSB
  - Complement overflow byte
  - Increment overflow byte
  - If not equal to zero, then goto COMOUT

```
        Increment LSB
        If not equal to zero, then goto COMOUT
        Increment MID
        If not equal to zero, then foto COMOUT
        Increment MSB
        If 'V' bit is set, then goto COMXX
    COMYY:
        Save condition code
    COMOUT:
        Decrement index register X
        Decrement accumulator B
        If not equal to zero, then goto COMOUT
        Save condition codes
        Return to caller
    DIVSTP:
        Goto FPOVF4
    NORMX5:
        Goto NORMX
    FPOVF6:
        Goto FPOVF5
    FPM451:
        Goto FPM452
    FPUND1:
        Goto FPUND2
    COMXX:
        Set counter to '4'
    COM22:
        Clear carry bit
        Rotate right the MSB
        Rotate right the MID
        Rotate right the LSB
        Rotate right the overflow byte
        Decrement the counter
        Increment the exponent
        Increment the exponent
        Goto COMYY

DIVSUB: Divide argument 2 by argument 1, save result in argument 3
    Get argument 2
    Goto NORMX5
    Get argument 1
    Goto NORMX5
    If overflow bit is set, then goto DIVSTP
    If equal to zero, then goto DIVSTP
    FPD15:
        Set counter to '5'
```

DSHF5:
- Shift the MSB of argument 2 to the right
- Rotate the MID of argument 2 to the right
- Rotate the LSB of argument 2 to the right
- Rotate the overflow byte of argument 2 to the right
- Decrement the counter
- If counter is not equal to zero, then goto DSHF5
- Shift the MSB of argument 1 to the right
- Rotate the MID of argument 1 to the right
- Rotate the LSB of argument 1 to the right
- Rotate the overflow byte of argument 1 to the right
- Get the overflow byte of argument 2
- Set counter to '29', 26 bits to do with overflow

FPD40:
- Save count on the stack
- Shift argument 2 left one bit
- Rotate left LSB of argument 2
- Rotate left MID of argument 2
- Rotate left MSB of argument 2
- Shift result left one bit
- Rotate left LSB of result
- Rotate left MID of result
- Rotate left MSB of result
- Subtract LSB of divisor from LSB of dividend
- Subtract MID of divisor from MID of dividend
- Subtract MSB of divisor from MSB of dividend
- If carry bit is set, then goto FPD55
- Goto FPD50

FPUND0:
- Goto FPUND1

FPM450:
- Goto FPM451

FPOVF7:
- Goto FPOVF6

FPD50:
- Add LSB of dividend to LSB of divisor
- Add MID of dividend to MID of divisor
- Add MSB of dividend to MSB of divisor
- Goto FPD60

FPD55:
- Increment overflow byte of result

FPD60:
- Get counter from the stack
- Decrement counter
- If counter is not equal to zero, then goto FPD40
- Set shift counter to '3'

SHFR3:
Shift the overflow byte of the result left
Rotate the LSB of the result to the left
Rotate the MID of the result to the left
Rotate the MSB of the result to the left
Decrement shift counter
If shift counter is not equal to zero, then goto SHFR3
Get exponent of result
Save it for sign calculation
Exclusive OR it with argument 1
Extract the sign
Clear the local overflow
Double exponent for argument 2
Double exponent for argument 1
Subtract argument 1 exponent from argument 2 exponent
If there is no overflow, then goto FPD70
If there is an underflow, then goto FPD80
Goto FPOVF7, since there is a definite overflow

FPD70:
Add '2' to argument 2 exponent

FPD75:
Goto FPM450

FPD80:
Add '02H' to argument 2 exponent
If overflow, then goto FPD85
Goto FPUND0

FPD85:
Clear overflow bit
Goto FPD75

FCMP: Floating point compare routine. This routine assumes that any number compared will be previously normalized. This allows for the second byte alone to identify a zero, in which case that byte is used in the exponent checking.
Format is LABEL FCMP ARG1,ARG2. Where ARG1 and ARG2 are the starting addresses of the two numbers to be compared. The results of the compare will be in the CCR.

Get the first byte of argument 1 in accumulator B
Get the first byte of argument 2 in accumulator A
If the exponents are equal, then goto CPAGE1
Goto EXITC

CPAGE1:
Get the second byte of argument 1 into accumulator A
If it is equal to zero, then goto ZERO1
Get the exponent for argument 1 in accumulator A

ZERO1:
Get the second byte of argument 2 in accumulator B
If it is equal to zero, then goto ZERO2
Get the exponent for argument 2

ZERO2:
Compare just the exponents
If the exponents are equal, then goto CPAGE2
Goto EXITC

CPAGE2:
Get the second byte of argument 1 into accumulator A
Get the second byte of argument 2 into accumulator B
If the bytes are equal, then goto CPAGE3
Goto EXITC

CPAGE3:
Get the third byte of argument 1 in accumulator A
Get the third byte of argument 2 in accumulator B
If they are equal, then goto CPAGE4
Goto EXITC

CPAGE4:
Get the fourth byte of argument 1 in accumulator A
Get the fourth byte of argument 2 in accumulator B
Compare accumulator A and B

EXITC:
Get the CCR value
Save it on the stack
Goto SVCEX

HEX2FP: Routine to take the BCD encoded numbers from the thumbwheel and convert them to Victoreen floating point format.
This routine is not reentrant.
Exponent sign is represented as :

Representation	Sign Value
8	+1
3	+0
0	-1
7	-0

Get the exponent byte
Isolate the exponent representation
If the exponent is not equal to +1, then goto EXPZER
Get the exponent
Isolate the right half of the exponent
Add ten to exponent value
Set flag to indicate positive exponent
Goto TEXP

EXPZER:
  If exponent is not equal to zero, then goto EXPNON
  Get the exponent
  Isolate the right half of the exponent
  Set flag to indicate positive exponent
  Goto TEXP
EXPNON:
  If exponent is not equal to –1, then goto EXPNZE
  Get the exponent
  Isolate right half of the exponent
  Set flag to indicate negative exponent
  Goto EXPNEG

EXPNZE:
  Get exponent equal to –0
  Isolate right half of exponent
  Set flag to indicate negative exponent
EXPNEG:
  Add '2' to absolute value
  Goto COMEX1
TEXP:
  If exponent is equal to zero, then goto STRZER
  If absolute value of exponent is not equal to '1', then goto
    COMEXP
  Make the exponent equal to 1
  Goto COMEX1
STRZER:
  Make the exponent '2'
  Goto COMEX1
COMEXP:
  Decrement the exponent
  Decrement the exponent
COMEX1:
  Adjust index for 4 byte entries into table
  Save accumulator B
  Put the exponent into accumulator B
  Get the table address
  Add the offset to table address
  Get first byte of argument
  Store in first byte of multiplier
  Get second byte of argument
  Store in second byte of multiplier
  Get third byte of argument
  Store in third byte of multiplier
  Get fourth byte of argument
  Store in fourth byte of multiplier
  Get the stack pointer
  Get address of HEX number

Get ones digit and 'E'
Isolate digit
Get hundred and tens digit
Isolate the tens digit
Get address of tens conversion table
Add offset to table address
Get tens representation of value
Get tens and ones result in A
Get stack pointer
Get address of HEX number
Get hundreds and tens digit
Isolate hundreds digit
Double the index
Get address of hundreds conversion table
Add offset to table address
Get hundreds representation of value
Get tens and ones result in accumulator B
Add offset
Store HEX value
Get fixed exponent value
Store exponent
Zero out last byte
Save it
Get exponent flag
If flag is equal to zero, then goto POSEXP
Call FDIV
Goto STRSLT

POSEXP:
Call FMUL

STRSLT:
Get stack pointer
Get address of result
Get result byte by byte and store it
Goto SVCEX

TENTAB: Table with HEX representations of multiplies of ten, values from 00 to 90.

HUNTAB: Table with HEX representations of multiplies of hundred, values from 000 to 900.

HXREFAB: Table with floating point values for conversion, values from 1E0 to 1E21.

FP2HEX: Routine to convert floating point to BCD display format. This routine is not reentrant.

Error code = Byte 0 of second argument
0 = result is good
1 = value is negative, no conversion made
2 = exponent too large or too small
Calling protocol: LABEL FPEHEX FP_ADDR,DISP_ADDR

Define local variables
Get stack pointer
Get address of floating point number
Get the exponent
If exponent is valid, then goto EXPOK
Goto BADEXP
EXPOK:
    If exponent is not negative, then goto PEXP
    Goto NEGVAL
PEXP:
    Get exponent
    If the exponent is negative, then goto NEGEXP
    Subtract '040H' from the exponent
    Set the flag to indicate positive exponent
    Goto TSTEXP
NEGEXP:
    Subtract exponent from '040H'
    Put result in accumulator A
    Make a copy in accumulator B
    Set up flag to indicate negative exponent
    Add '2' to exponent
    Goto INDEX1
TSTEXP:
    If exponent is equal to zero, then goto STRZVL
    If exponent absolute value is not equal to one, then go to INDEX
    Set the value to '1' for indexing into table
    Set up the negative flag
    Goto INDEX1
STRZVL:
    Set the value to '2' for indexing into table
    Set the negative flag
    Goto INDEX1
INDEX:
    Adjust the index to comput address of value in HXREFAB
INDEX1:
    Adjust index for 4 byte entries
    Save negative flag
    Get address of table HXREFAB
    Add offset to table address
    Get the exponent from the table
    Store it

```
        Get the MSB from the table
        Store it
        Get the MID from the table
        Store it
        Get the LSB from the table
        Store it
        Get the stack pointer
        Get address of argument
        Get byte by byte and store it
        Get negative flag
        If not negative then, goto DIVEXP
        Call FMUL, to do a mulitply
        Goto EXPCK
    DIXEXP:
        Call FDIV, to do the divide
    EXPCK:
        Get stack pointer
        Get address of second argument
        Initialize internal flag
        Initialize flag for multiply
        Initialize value to '041A0H' for multiply and divide
    TSTEX:
        Call FCMP to compare result to floating point zero
        If it is equal to zero, then goto DIVDON
        Get exponent of result
        If exponent is too large, then goto CHNGE
        If the exponent is equal to '043H', then go to DIVDON
        Increment the exponent of argument 1
        Call FSTR to set up value to be multiplied
        Call FMUL, to do the multiplication
        Goto TSTEX, to check if exponent is equal to 43
    CHNGE:
        Get stack pointer
        Get address of temporary flag
        Increment flag
        Call FSTR to set up value to be divided
        Call FDIV to do the divide
        Goto to TSTEX to test for large exponent
    DIVDON:
        Call GETDIG, to isolate and round off good digits
    CNVRT:
        Set up counter on stack
        Set up another counter on stack
        Put result in accumulator A and accumulator B
        Get stack pointer
```

CHK_3E8:
Subtract '3E8H' from the value
If carry bit is set, then goto ADD_3E8
Increment the counter
Goto CHK_3E8
ADD_3E8:
Add '3E8' to the value to restore to positive
Clear the carry bit
Put thousand digit in the left nibble
CHK_64:
If value is not equal to '64H', then foto GRT_64
If lower byte is equal to '064H', then goto CMP_A
GRT_64:
Subtract 64 from the value
Increment the counter
Goto CHK_64
CMP_A:
Compare value to '00AH'
If carry bit is set, then goto LSTDIG to get last digit
Subtract 'AH' from the value
Increment the counter
Goto CMP_A
LSTDIG:
Put tens digit in the left nibble
Add ones digit
Put result on the stack
Get accumulator A off of stack, this contains units and tens
Get accumulator B off of stack, this contains hundreds and thousands
Save accumulator A
Clear out the hundreds
If value is zero, then goto NOSHFT
Initialize counter to '00H'
Restore accumulator A
Clear carry bit
JSTIFY:
Rotate value in accumulator B to the left
Rotate value in accumulator A to the left
Increment the counter
If counter is not equal to '04H', then goto JSTIFY
Goto PAGEA
NOSHFT:
Get stack pointer
Get address of argument 2
Increment exponent counter
Restore accumulator A to original value

PAGEA:

AND 'EH' with value in accumulator B
Inclusive OR accumulator B with '00001110B'
Get stack pointer
Get address of argument 2
Store hundreds and tens digit
Store result
Get stack pointer
Get address of argument 1
Get exponent of argument 1
Get stack pointer
Get address of argument 2
Add the number of overflow byte divides to exponent of argument 2
Clear the temporary flag
Subtract the number of overflow byte multiplies
Clear the temporary flag
If the exponent is not positive, then goto NEXP
If the exponent is greater than '053H', then goto BADEXP
Isolate exponent digit
If digit is less than 'AH', then goto SMLEXP
Subtract '00AH' from the exponent
OR in '10000000B' to use code 8 = +1
Goto STREXP, to store exponent

SMLEXP:

OR in '00110000B', to use code 3 = +0
Goto STREXP

NEXP:

If exponent is –19, the goto BADEXP
Put the exponent into accumulator A
Set exponent to '040H' minus exponent
If exponent is less than '00AH', then goto SMLNEX
Subtract '00AH' from exponent
Use code 0 = -1
Goto STREXP

SMLNEX:

Use code 7 = -0

STREXP:

Get stack pointer
Get address of argument 2
Store exponent
Goto FPDONE

BADEXP:

Set error code to '2'
Goto ERRCOD

NEGVAL:
Set error code to '1'
ERRCOD:
Get stack pointer
Get address of argument 2
Store error code
FPDONE:
Goto SVCEX

GETDIG: Routine to isolate valid digits and round off values.
Input: value in DEND
Output: index register X contains valid digits

Get the exponent
If the value is negative, then goto NEXPON
Isolate exponent value
Goto GOTEXP
NEXPON:
Put exponent into accumulator B
Subtract accumulator B from '040H', with result in accumulator A
GOTEXP:
Set counter to '4' for rotating
Store counter on the stack
Get stack pointer
Get most significant byte of argument
Get second most significant byte of argument
THRDIG:
Clear the carry bit
Rotate right accumulator A
Rotate right accumulator B
Decrement loop counter
If counter is not equal to zero, then goto THRDIG
Save result in index register X
Adjust stack to normal
Isolate determining digit
RND:
If digit is less than '8', then goto GOTDIG
Increment index register X
GOTDIG:
Return to caller

FGETI: Routine to get floating point number.
Byte 0 Future exponent
Byte 1 MSB of INT
Byte 2 MSB of MIDDLE
Byte 3 LSB of MIDDLE
Byte 4 LSB of INT

Get stack pointer
Get argument
Put each byte of argument on the stack
Put maximum possible exponent value ('048H') on the stack
Get stack pointer

TSTFG:
- If accumulator B is equal to zero, then goto FINFG

CONTC:
- Get MSB
- Remove the lower nibble
- If it is not equal to zero, then goto FINFG
- Decrement the exponent
- Save it
- Initialize shift counter to '4'

SL4X:
- Shift byte 4 to the left
- Rotate byte 3 to the left
- Rotate byte 2 to the left
- Rotate byte 1 to the left
- Decrement the shift counter
- If shift counter is not equal to zero, then goto SL4X
- Get exponent into accumulator B
- Goto TSTFG

FINFG:
- If accumulator B is equal to zero, then goto NORD
- Get byte 4
- Compare it to '080H'
- If the carry is set, then goto NORD
- Get byte 3
- Increment it
- Save it
- If carry is clear, then goto NORD
- Get byte 2
- Increment it
- Save it
- If carry is clear, then goto NORD
- Get byte 1
- Increment it
- Save it
- If carry is clear, then goto NORD
- Set counter to '4'

SR4X:
- Rotate byte 1 to the right
- Rotate byte 2 to the right
- Rotate byte 3 to the right
- Decrement counter
- Clear the carry bit

If counter is not equal to zero, then goto SR4X
Increment the exponent

NORD:

Get stack pointer
Get argument 2
Get the exponent off of stack
Store it
Get MSB off of stack
Store it
Get LSB off of stack
Store it
Get overflow byte off of stack
Goto SVCEX

FPUTI: Routine to put a floating point number is argument 2.
Input is a floating point number of 4 bytes. If the unsigned integer equivalent is out of range, zeros will be stored.

Get the stack pointer
Get argument 1
Clear accumulator B
Store accumulator B on the stack
Get byte 3 of argument 1 into accumulator B
Put byte 3 of argument 1 on the stack
Get byte 2 of argument 1 into accumulator B
Put byte 2 of argument 1 on the stack
Get byte 1 of argument 1 into accumulator B
Put byte 1 of argument 1 on the stack
Get byte 0 of argument 1 into accumulator B
Put byte 0 of argument 1 on the stack
Get the stack pointer
If the exponent is less than '049H', then goto CFPUT
Zero out five bytes in index register
Goto XFPUT

CFPUT:

Set accumulator A to '048H', number of shifts

CTES:

If accumulator A is equal to zero, then goto XFPUT
Set counter to '4'

RTOP:

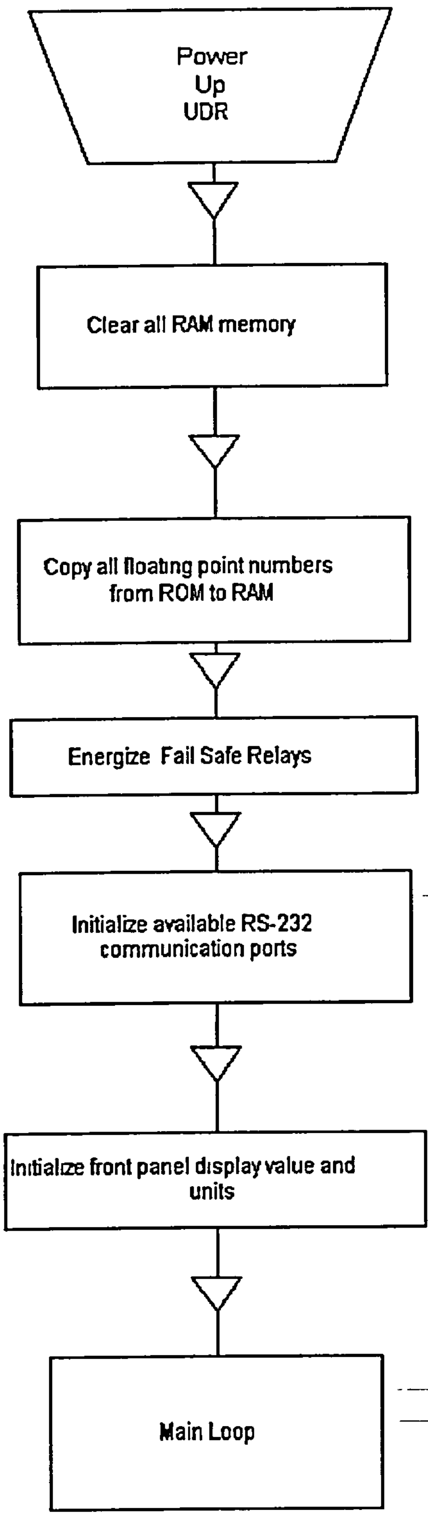
Shift byte 1 to the right
Rotate byte 2 to the right
Rotate byte 3 to the right
Rotate byte 4 to the right
Decrement counter
If counter not equal to zero, then goto RTOP
Decrement number of shifts

Goto CTES

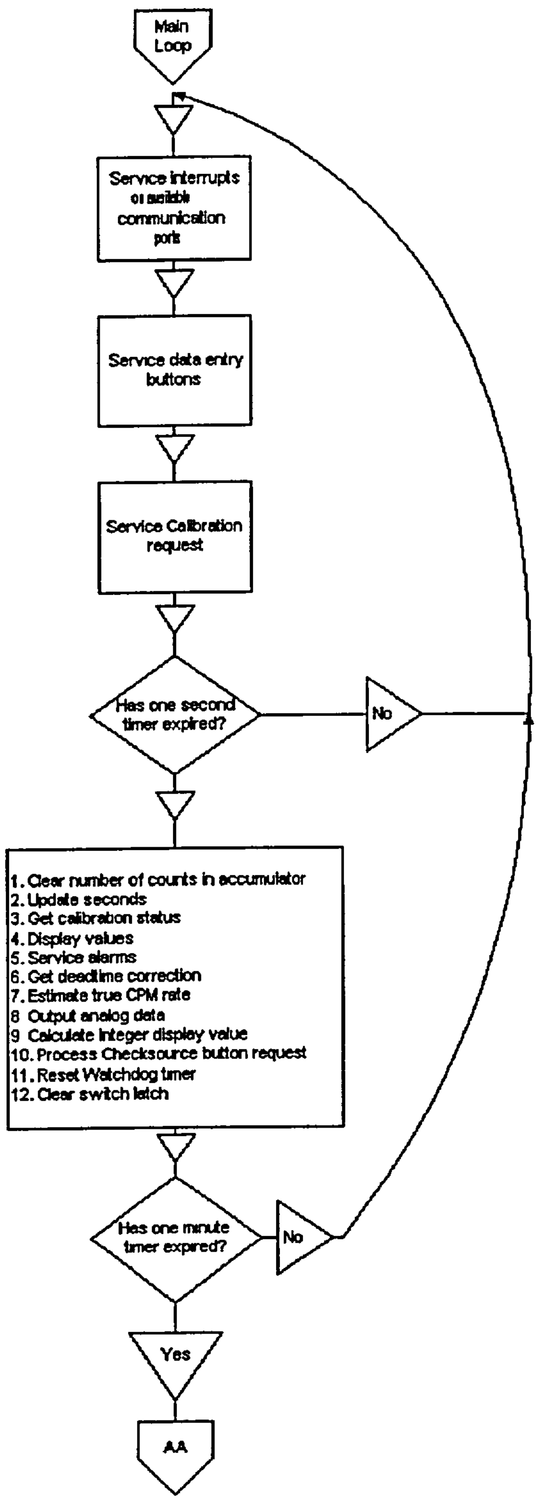
SVCER0: The SVC error handler. Input is number of argument bytes in accumulator B.

```
SVCER0:
SVCERR:
    Goto SVCEX
```

APPENDEX A – FIRMWARE FLOW CHART

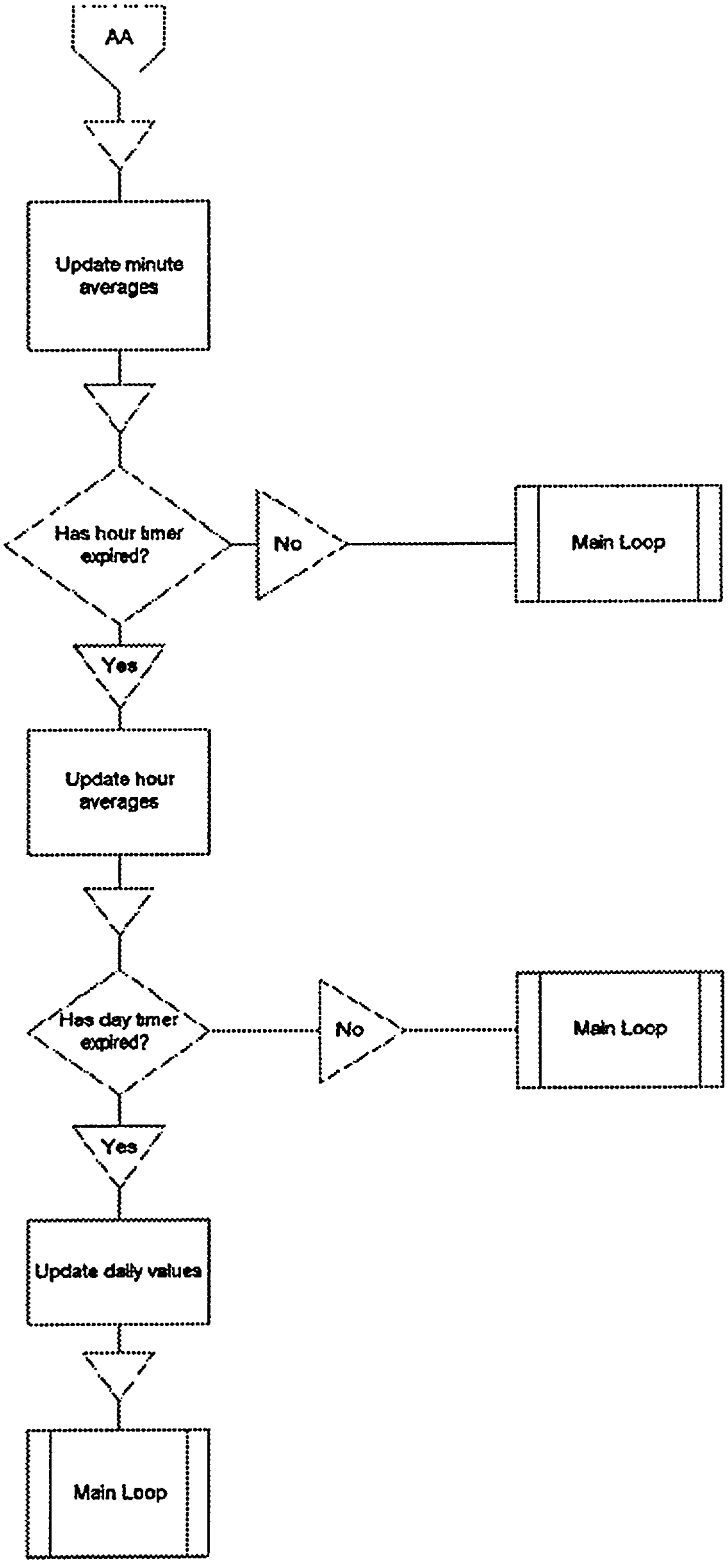


APPENDEX A – FIRMWARE FLOW CHART (Continued)

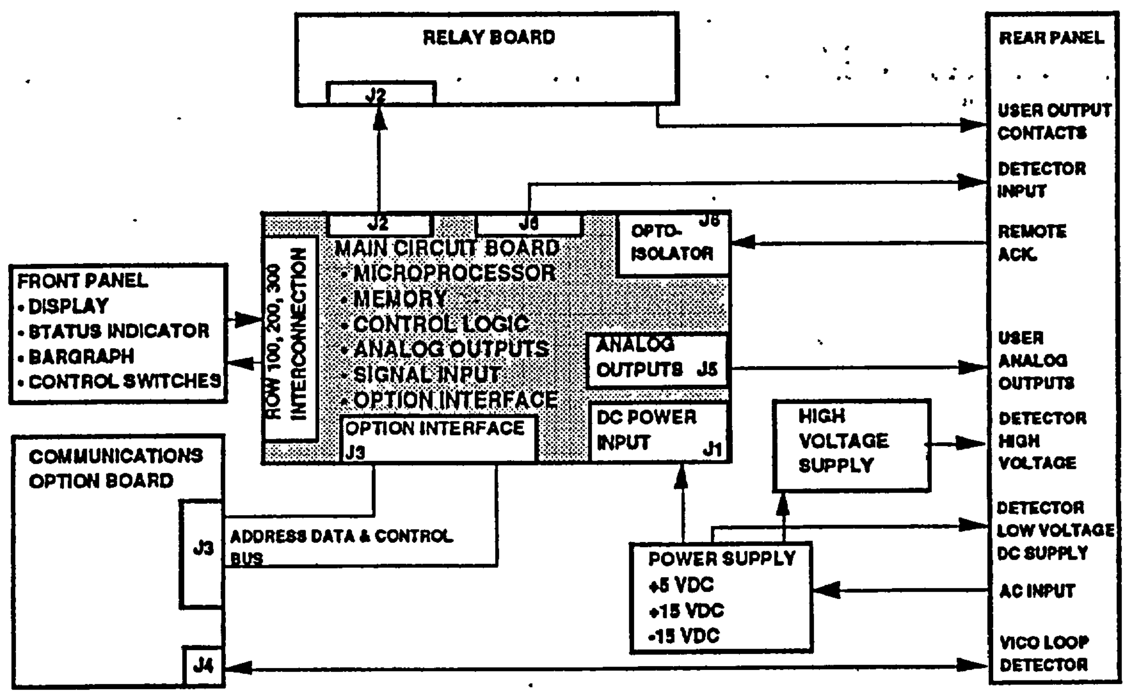


SYNCOR RADIATION MANAGEMENT		DATE 10/23/02		TITLE SOFTWARE DESIGN DESCRIPTION, 9405603	
REV 2	ECN NO. 3076	RELEASED FOR PRODUCTION	DOC CTRL RKM	SHEET 184 of 189	NO. 94095603SDD	SIZE PLN

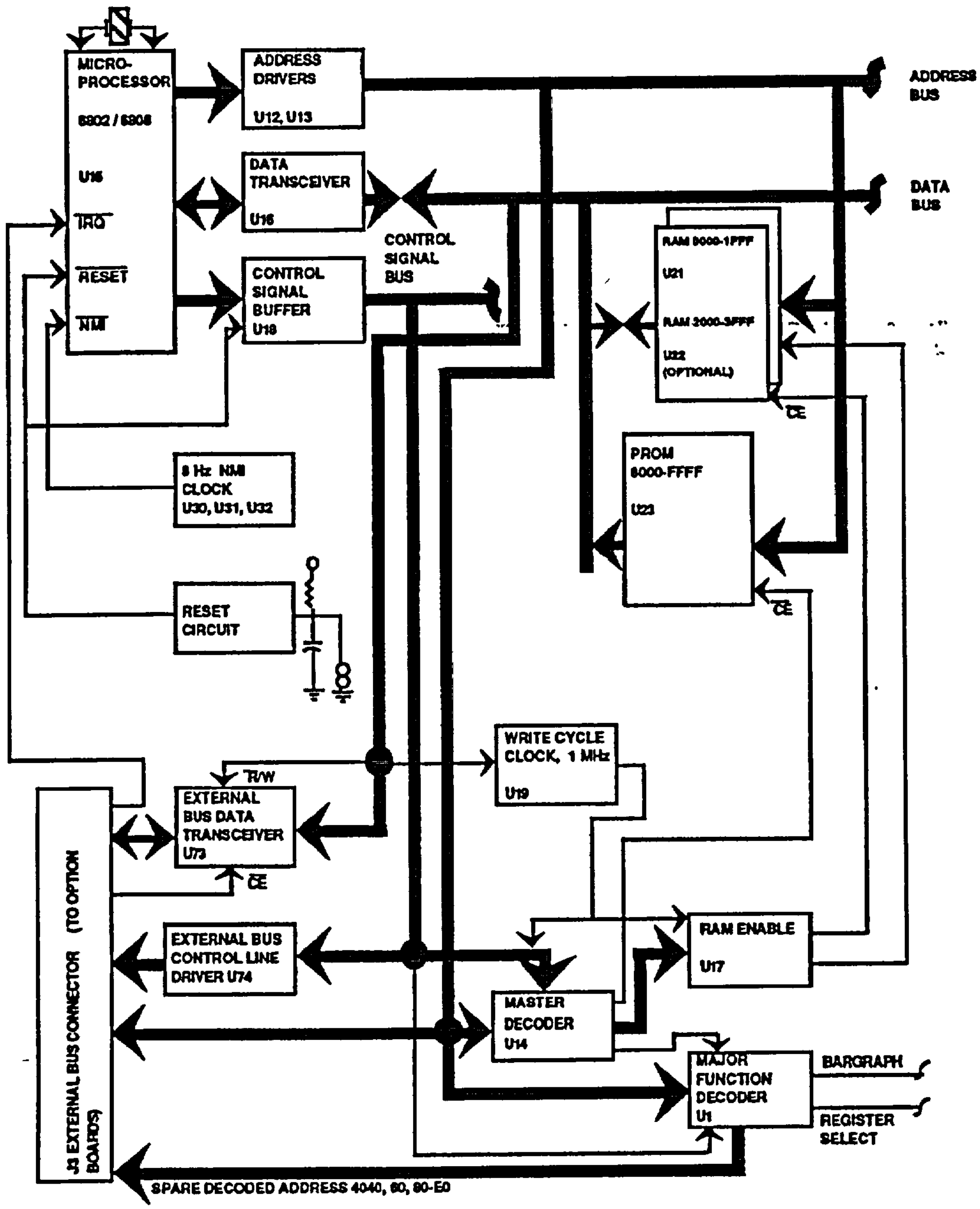
APPENDEX A – FIRMWARE FLOW CHART (continued)



APPENDIX B – Microprocessor Block Diagram

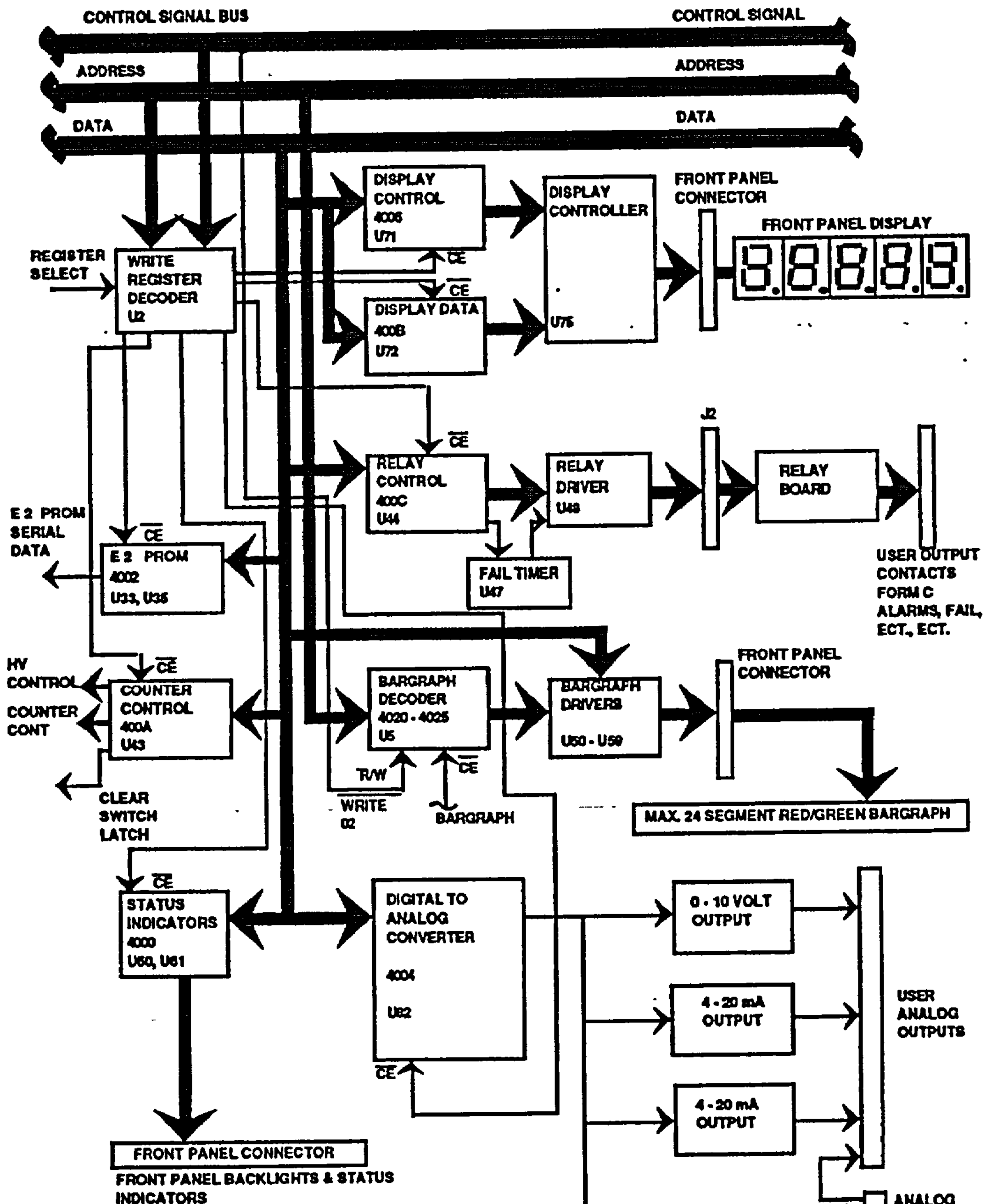


APPENDIX B – Microprocessor Block Diagram (Continued)

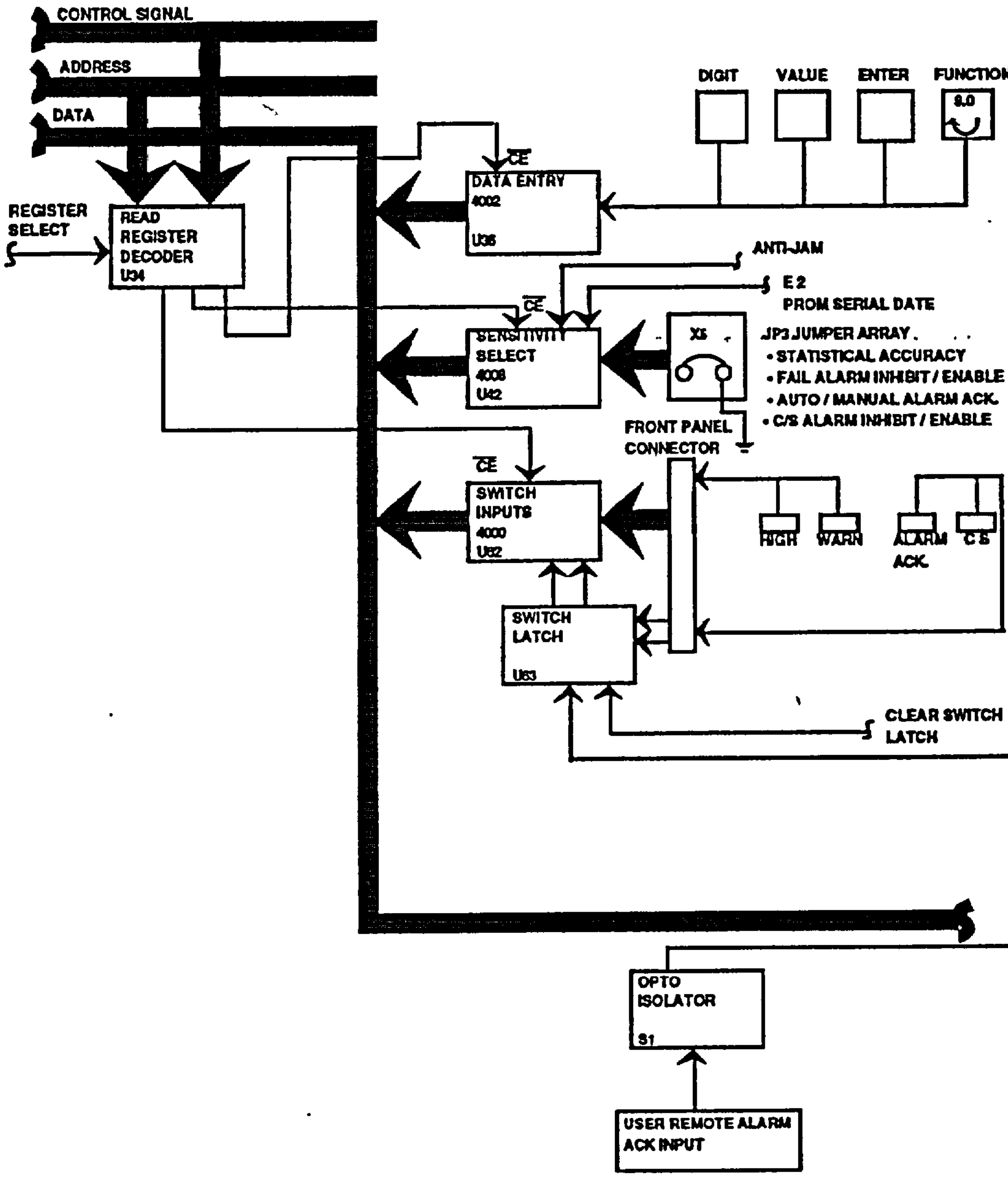


SYNCOR RADIATION MANAGEMENT			DATE 10/23/02		TITLE SOFTWARE DESIGN DESCRIPTION, 9405603	
REV 2	ECN NO. 3076	RELEASED FOR PRODUCTION	DOC CTRL [signature]	SHEET 187 of 189	NO. 94095603SDD	SIZE PLN

APPENDIX B – Microprocessor Block Diagram (Continued)



APPENDIX B – Microprocessor Block Diagram (Continued)



SYNCOR RADIATION MANAGEMENT			DATE 10/23/02		TITLE SOFTWARE DESIGN DESCRIPTION, 9405603	
REV 2	ECN NO. 3076	RELEASED FOR PRODUCTION	DOC CTRL RCHM	SHEET 189 of 189	NO. 94095603SDD	SIZE PLN