EARTH OBSERVING SYSTEM
DATA AND INFORMATION SYSTEM (EOSDIS)
TEST SYSTEM (ETS)
VME LOW-RATE SYSTEM (VLS)
DETAILED DESIGN SPECIFICATION

VOLUME 4

February 1996

DRAFT

Prepared by:

D. Nguyen,
Sr. Systems Engineering Specialist
Loral AeroSys

Reviewed by:

T. Voudouris
CSC

Reviewed by:

C. Mirchandani,
Sr. Systems Engineering Specialist
Loral AeroSys

Quality Assured by:

C. Pepersack, Principal Engineer
Unisys

Edited by:

S. George, Engineer
RMS Technologies, Inc.

Approved by:

N. Speciale, Head
System Applications Section
Code 521.2, NASA/GSFC

Goddard Space Flight Center
Greenbelt, Maryland 20771

PREFACE

This document is Volume 4 of a 5-volume set specifying the detailed design for the Earth Observing System Data and Information System (EOSDIS) Test System (ETS). This volume describes the detailed design specifications for the ETS Versa Module Eurocard (VME) Low-rate System (VLS) being developed by the Goddard Space Flight Center (GSFC) Microelectronic Systems Branch (MSB).

This document will serve as the single source document for configuration control of the ETS VLS system design, and will serve as the basis for design, implementation, and testing.

This document is under the configuration management of the Microelectronic Systems Branch Configuration Control Board (CCB). This document may be changed by Documentation Change Notice (DCN), reflected in text by change bars, or by complete revision.

Technical Publications Group
Microelectronic Systems Branch, Code 520.9
Goddard Space Flight Center
Greenbelt, Maryland 20771

CHANGE INFORMATION PAGE

List of Effective Pages

Page Number	Issue
Title	Original
Signature Page	Original
iii through xii	Original
1-1 through 1-5	Original
2-1 through 2-3	Original
3-1 through 3-4	Original
4-1 through 4-9	Original
5-1 through 5-7	Original
6-1 through 6-22	Original
7-1 through 7-11	Original
8-1 through 8-5	Original
9-1 through 9-18	Original
10-1 through 10-3	Original
AB-1 and AB-2	Original
A-1 through A-11	Original

Document History

Issue	Date	DCN No.
Original	February 1996	n/a

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION.....1-1
1.1 Purpose.....1-1
1.2 Overview.....1-1
1.3 Applicable Documents.....1-3
1.3.1 Parent Requirements Documents.....1-3
1.3.2 Standards Documents.....1-3
1.3.3 Reference Documents.....1-3
1.4 Document Organization.....1-4
1.5 Terminology.....1-4

SECTION 2 - SYSTEM DESIGN OVERVIEW 2-1
2.1 Design Methodologies.....2-1
2.2 Design Approaches.....2-2
2.2.1 Changing EDOS Baseline and Requirements.....2-2
2.2.2 Interface Issue.....2-2
2.2.3 Technical Challenge.....2-2
2.2.4 Return Link Data Processing Functions.....2-2
2.2.5 System Test.....2-2
2.2.6 User Interface.....2-3
2.3 Design Studies and Prototypes.....2-3

SECTION 3 - SYSTEM DESIGN 3-1
3.1 Introduction.....3-1
3.2 Functional Overview.....3-1
3.3 System.....3-2
3.4 Forward-Link (Uplink) CLTU Output Processing.....3-2
3.5 Return-Link Data (CADU) Processing.....3-3
3.5.1 Frame Processing.....3-3
3.5.2 Packet Processing.....3-3
3.6 System Architecture.....3-3

SECTION 4 - USER INTERFACE.....4-1
4.1 System Setup, Control, and Status.....4-1
4.2 Local Operations/OPMAN.....4-1
4.2.1 Activating a Catalog.....4-2
4.2.2 Looking at Status.....4-2
4.2.3 Shutting Off a Catalog.....4-3
4.2.4 Editing a Catalog.....4-3
4.3 Sample OPMAN Window.....4-3

SECTION 5 - OPERATIONS SCENARIO.....5-1
5.1 Return Link Operation.....5-1
5.2 Forward Link Operation.....5-1
5.3 FEP Card Modes of Operation.....5-2
5.3.1 Block Mode.....5-2
5.3.2 Frame Mode.....5-2
5.3.2.1 Reed-Solomon Throughput Mode.....5-2
5.3.2.2 Reed-Solomon Decode Mode.....5-3
5.3.2.3 Frame-Level Service Processing Mode.....5-3
5.3.3 Test Mode.....5-3

TABLE OF CONTENTS (CONT'D)

5.4 Service Processor Card Modes of Operation....5-3
5.4.1 Pipeline Input Interface....5-3
5.4.2 Input Rate Buffering....5-3
5.4.3 Tribuffer Subsystem....5-4
5.4.4 Reassembly RAM Subsystem....5-4
5.4.5 VSB Interface....5-4
5.4.6 Loopback FIFO....5-4
5.4.7 Pipeline Output Interface....5-4
5.4.8 VME Interface....5-5
5.4.9 Quality Processor....5-5
5.4.10 Header Processor....5-5
5.4.11 Output Processor....5-5
5.4.12 DPR Subsystem....5-6
5.5 Forward Link Interface Card Modes of Operation....5-6
5.5.1 Throughput Mode....5-6
5.5.2 Encode Mode....5-7

SECTION 6 - DETAILED HARDWARE DESIGN....6-1
6.1 System Overview....6-1
6.1.1 Introduction....6-1
6.1.2 System Components....6-1
6.2 Commercial Components....6-3
6.2.1 System Disk and Interface (Module)....6-4
6.2.2 Master Controller Card (MVME-167-032)....6-4
6.2.3 System Memory Card (MM6740CN)....6-4
6.2.4 Global Memory Card (MM6346D)....6-4
6.3 Custom Components....6-4
6.3.1 EOS Simulator Card....6-5
6.3.2 Front-End Processor Card....6-8
6.3.2.1 CPU/VMEbus/VSB....6-8
6.3.2.2 Input Processing....6-8
6.3.2.3 Frame Synchronization Processing....6-10
6.3.2.4 Annotation Processing....6-11
6.3.2.5 Error Detection and Correction....6-11
6.3.2.6 Frame Service Processor....6-11
6.3.2.7 Routing....6-11
6.3.2.8 Data Capture Subsystem....6-11
6.3.3 EOS Service Processor Card....6-12
6.3.4 Forward Link Interface Card....6-13
6.3.4.1 Microprocessor Decode And Control Logic....6-16
6.3.4.2 NCO Clock Generator....6-17
6.3.4.3 Timecode Interface....6-17
6.3.5 High-Rate Telemetry Backplane....6-17

SECTION 7 - DETAILED SOFTWARE DESIGN 7-1
7.1 Software Environment....7-1
7.1.1 VxWorks....7-1
7.1.2 Telemetry Processing Control Environment....7-1
7.1.3 MEDS....7-2
7.2 MCC Software....7-4
7.2.1 Gateway Interface Software....7-5

TABLE OF CONTENTS (CONT'D)

7.2.2 Command Block Processing....7-5
7.2.3 EDU Processing....7-6
7.3 EOS Simulator Card Software....7-6
7.4 EOS-FEP Card Software....7-7
7.5 EOS Service Processor Card Software....7-7
7.5.1 Header Process....7-7
7.5.2 Quality Process....7-8
7.5.3 Output Process....7-8
7.6 Forward Link Interface Card Software....7-8
7.6.1 Clock Generator....7-8
7.6.2 Timecode Interface....7-9
7.6.3 Telecommand Output Interface....7-9
7.6.4 Interrupts....7-9

SECTION 8 - LOCAL AREA NETWORK AND CONFIGURATIONS....8-1
8.1 LAN Configuration....8-1
8.1.1 ETS VLS Configuration....8-1
8.1.2 MCC and CDS Configuration....8-2
8.1.3 VME LRS and EOC Configuration....8-2
8.1.3.1 MCC and EOC Configuration....8-2
8.1.3.2 CDS and EOC Configuration....8-2
8.1.4 EOS Simulator Card and CDS configuration....8-2
8.2 Communication Protocols....8-2
8.2.1 TCP....8-2
8.2.2 UDP....8-3
8.2.3 IP....8-4

SECTION 9 - TELEMETRY PROCESSING CONTROL ENVIRONMENT....9-1
9.1 Introduction....9-1
9.2 TPCE Graphic User Interface Description....9-1
9.3 TPCE Software Architecture....9-14
9.3.1 VLSI Server....9-14
9.3.2 Status Collector....9-14
9.3.3 Message Collector....9-15
9.3.4 Process Initiator....9-15
9.3.5 External Status Server....9-15
9.3.6 Event Log....9-15
9.3.7 IPC Server....9-15
9.3.8 Parameter Server....9-16
9.3.9 Schedule Reader....9-16
9.3.10 Activity Schedule....9-16
9.3.11 Expert System....9-16
9.3.12 Data Set Distributor (HRS Only)....9-16
9.3.13 Configuration Set Editor....9-17
9.3.14 Preference Editor....9-17
9.3.15 Status Pages....9-17
9.4 Necessary VLS Modifications....9-17

SECTION 10 - SYSTEM TEST APPROACH....10-1
10.1 System Tests....10-2
10.2 Test Guidelines....10-2

TABLE OF CONTENTS (CONT'D)

10.3 Test Approval....10-2
10.4 Environmental Test Conditions....10-2
10.5 Discrepancies Reporting and Retest....10-3
10.6 Failure During Test....10-3
10.7 Retest and Regression Testing....10-3
10.8 Test Dependencies....10-3

ACRONYMS AND ABBREVIATIONS....AB-1

APPENDIX A - REQUIREMENTS TRACEABILITY MATRIX....A-1

FIGURES

Figure 1-1 ETS Low-rate System Block Diagram....1-2
Figure 1-2 Configuration 5 - EDOS Low Rate Simulator....1-2
Figure 3-1 VLS System Diagram....3-1
Figure 4-1 ETS-LRS Main OPMAN Window....4-2
Figure 4-2 ETS-LRS Main Setup Window....4-3
Figure 4-3 Synchronizer Card Setup Window....4-4
Figure 4-4 Frame Annotater Setup Window....4-5
Figure 4-5 Reed-Solomon Setup Window....4-6
Figure 4-6 Frame Server Setup Window....4-7
Figure 4-7 FEP Status Page 1....4-8
Figure 4-8 FEP Status Page 2....4-9
Figure 5-1 ETS-LRS Return Link Scenario....5-1
Figure 5-2 ETS-LRS Forward Link Scenario....5-2
Figure 6-1 ETS VLS Chassis....6-3
Figure 6-2 Typical Custom Card....6-5
Figure 6-3 EOS Simulator Card Data Flow....6-8
Figure 6-4 FEP Card Data Flow Diagram....6-9
Figure 6-5 EOS Service Processor Card Data Flow....6-13
Figure 6-6 Forward Link Interface Card Functional Block Diagram....6-15
Figure 6-7 Microprocessor Decode Logic....6-16
Figure 6-8 Control and Status Logic....6-17
Figure 6-9 HRTB Hardware....6-18
Figure 7-1 VxWorks and MEDS Architecture....7-3
Figure 7-2 MCC Context Diagram....7-5
Figure 8-1 Ethernet Application Layer....8-1
Figure 8-2 TCP Header....8-3
Figure 8-3 UDP Header....8-4
Figure 8-4 IP Header used with TCP....8-4
Figure 9-1 TPCE Main Window....9-1
Figure 9-2 VME Icon....9-1
Figure 9-3 VME Quick Status Information....9-2
Figure 9-4 VME Status Page Menu....9-3
Figure 9-5 Manual Commands Menu....9-3
Figure 9-6 File Menu....9-3
Figure 9-7 Editor Menu....9-4
Figure 9-8 View Menu....9-4
Figure 9-9 Event Log Annotation....9-5

TABLE OF CONTENTS (CONT'D)

Figure 9-10 Activity Schedule Editor....9-6
Figure 9-11 Activity Schedule Editor File Menu....9-6
Figure 9-12 Activity Schedule Editor View Menu....9-7
Figure 9-13 Session Editor Window....9-8
Figure 9-14 Data Set Distributor Editor....9-9
Figure 9-15 Data Set Retransmission....9-10
Figure 9-16 Example System Status Page....9-11
Figure 9-17 Example Subsystem Status Page....9-12
Figure 9-18 Configuration Set Editor....9-13
Figure 9-19 Preference Editor....9-13
Figure 9-20 TPCE Software Architecture....9-14
Figure 10-1 System Test Configuration....10-1

TABLES

Table 6-1 NRZ Decoding Scheme....6-10

SECTION 1
INTRODUCTION

This document is Volume 4 of a 5-volume set specifying the detailed design for the Earth Observing System (EOS) Data and Information System (EOSDIS) Test System (ETS).

1.1 PURPOSE

This document specifies the system level design for the ETS Low-rate Subsystem (LRS). The VME Low-rate System (VLS) is being developed by the Data Systems Technologies Division, Code 520 at Goddard Space Flight Center (GSFC), as a part of the ETS project to support EOSDIS system integration, testing, verification, and validation.

The LRS system will provide simulated processing functions of the EOS Data and Operations System (EDOS) to support the testing of EOS Operations Center (EOC), Spacecraft Integration and Test Facility (SCITF), and Spacecraft Simulator (SSIM).

This Detailed Design Specification (DDS) document establishes a formal understanding of the detailed software and hardware design. This document defines the formats and contents of data flowing between the EDOS and external facilities. The system user's guide will further document the capabilities of the LRS.

The designs detailed in this document were developed from preliminary designs contained in the handout at the ETS Design Review presentation held November 30, 1995.

1.2 OVERVIEW

Figure 1-1 depicts the ETS LRS block diagram. The LRS contains two major elements: VLS and Control & Display Subsystem (CDS). The VLS is responsible for data and commands processing; CDS handles system control and spacecraft command validation.

Together with CDS, the VLS will play a key role in supporting the ETS system test configuration five. It will provide the following major functions:

a. Perform command block validation.

b. Transmit Command Link Transmission Units (CLTU) that pass command block header validation.

c. Transmit Command Link Control Word (CLCW).

d. Perform S-band return-link processing with two channels.

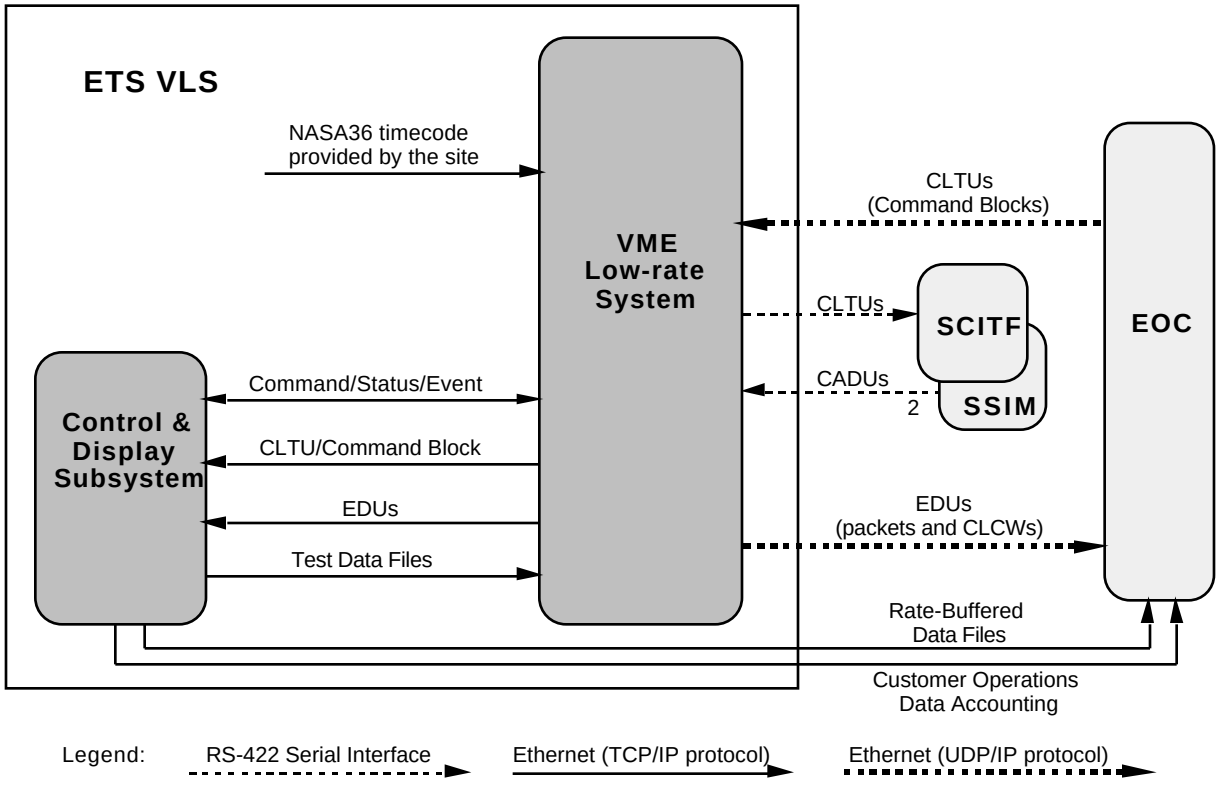


Figure 1-1. ETS Low-rate System Block Diagram

Figure 1-2 illustrates the data flow when the LRS performs EDOS forward-link and S-band return- link processing functions. In this configuration, the LRS will act as an EDOS to support interface testing between the EOC, SCITF, and SSIM.

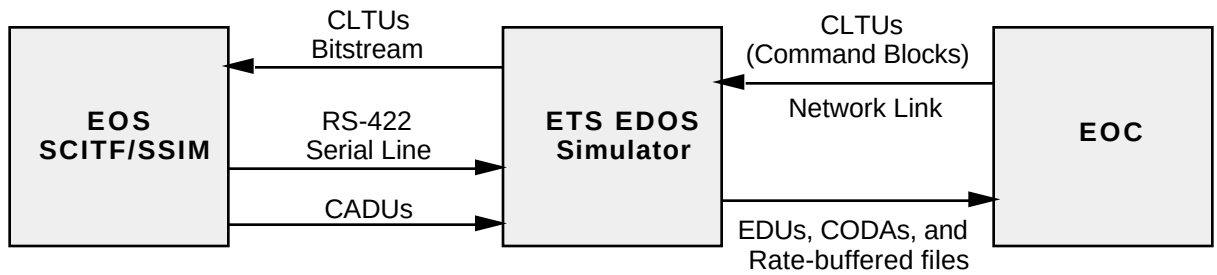


Figure 1-2. Configuration 5 - EDOS Low Rate

For the forward-link function, the VLS will receive CLTUs from the EOC in command data blocks through the EOSDIS Backbone Network (EBnet) interface. It will extract CLTUs from the blocks, add any necessary acquisition and idle sequences, and transmit the complete CLTUs to the SCITF/SSIM through the EBnet communication link. The VLS will discard forward-link CLTUs when the aggregate forward-link data rate exceeds the capacity of the selected physical channel. Optionally, the VLS will transfer the received CLTUs to the CDS where they can be logged for archival and post-test analysis.

For the return-link function, VLS will receive Channel Access Data Units (CADU) bitstreams from the SCITF/SSIM. It will simulate EDOS functions by performing frame synchronization, Reed-Solomon decoding and error correction, packet reassembly and quality annotation, CLCW extraction, EDOS Data Unit (EDU) generation. The VLS will transfer selected packets and all of the CLCWs in EDU format to the EOC as soon as they are received. It will also transfer selected EDUs to the CDS for rate-buffered file generation and delivery.

The LRS will provide processing status and statistics information to the ETS user with real-time updates.

1.3 APPLICABLE DOCUMENTS

The following documents were used as reference for the development of these requirements. They can be used to support, further define, and clarify information in this document.

1.3.1 PARENT REQUIREMENTS DOCUMENTS

a. Earth Observing System Data and Information System (EOSDIS) Test System (ETS) Functional and Performance Requirements, October, 1995.

b. ETS Operations Concept, May, 1995.

c. ETS System Design Specification, May, 1995.

d. ETS Low Rate System (LRS) System Requirements Specification (Volume 4), November, 1995.

1.3.2 STANDARDS DOCUMENTS

a. Consultative Committee for Space Data Systems (CCSDS), Recommendation for Space Data System Standards: Advanced Orbiting Systems, Networks and Data Links, CCSDS 701.0-B-2, November 1992.

b. CCSDS Recommendation for Space Data System Standards: Telecommand Part 1, Channel Service, CCSDS 201.0-B-1, January 1987.

c. CCSDS Recommendation for Space Data System Standards: Telecommand Part 2, Data Routing Service, CCSDS 202.0-B-2, November 1992.

d. CCSDS Recommendation for Space Data System Standards: Telemetry Channel Coding, CCSDS 101.0-B-3, May 1992.

1.3.3 REFERENCE DOCUMENTS

a. Presentation handout at ETS Design Review held November 30, 1995.

b. Interface Control Document (ICD) between EDOS and EOS Ground System (EGS), TRW/GSFC, January 1996.

c. EOSDIS Backbone Network (EBnet) and ETS ICD, February 1996.

d. Data Format Control Book for EOS-AM Spacecraft (ICD-106), Martin Marietta Corporation, Astro Space, April, 1994.

e. EDOS Functional and Performance Specification, GSFC, December 18, 1992.

f. EDOS External ICD Data Format Control Document (DFCD), Draft, July 1995.

g. DSTL-92-007, Human-Computer-Interface (HCI) Guidelines, August 1992.

h. Mission Operations and Data Systems Directorate (MO&DSD) Automated Information System (AIS) Security Policy, August 1990.

i. NASA Communications (Nascom) Access Protection Policy and Guidelines, Revision 3, August, 1995.

j. Front-End Processor Card Hardware Definition Document, 521-H/W-054, Code 521, GSFC/NASA, TBS.

k. EOS Service Processor Card Hardware Definition Document, 521-H/W-058, Code 521, GSFC/NASA, TBS.

l. Forward Link Interface Card Hardware Definition Document, Rev. B, 521-H/W-055, Code 521, GSFC/NASA.

m. EOS Simulator Card Hardware Definition Document, 521-H/W-051, Code 521, GSFC/NASA, TBS.

1.4 DOCUMENT ORGANIZATION

Section 2 presents an overview on system design, including design methodologies, design approaches, and design studies that led to the current baseline design.

Section 3 discusses VLS design, including a system block diagram, functional description, system architecture, and external interfaces.

Section 4 describes on a high-level the system user interface through which a user can control the system, and set up and monitor operations.

Section 5 provides scenarios for typical system operations.

Section 6 describes in detail the design of system hardware.

Section 7 describes an overview of application-specific software for each custom card.

Section 8 deals with issues in system Local Area Network (LAN) configuration and communications.

Section 9 gives a general description on system test approach and identifies test tools needed to carry out the system testing.

Appendix A includes a traceability matrix for mapping from the VLS System Requirements Specifications (SRS) to this document

1.5 TERMINOLOGY

Application Process Identifier (APID) – A space-borne experiment with a single APID may be defined as multiple sources; e.g., real-time and playback.

Configuration Set – Set of system parameters necessary to set up the VLS for a processing session.

Frame – Formatted data unit defined as CCSDS Version 1 Telemetry Transfer Frame, or Version 2 Coded Virtual Channel Data Unit (CVCDU), or CADU.

Processing Session – A scheduled time interval during which spacecraft data is received by the ETS through a single physical link. During a processing session, data may be received directly from the spacecraft simulator, a ground station playback, or a tape playback.

Source – A packet source is identified by a unique combination of Spacecraft Identifier (SCID), Virtual Channel ID (VCID), and APID. The VCID field can be a wildcard. The source is defined for sorting and grouping purposes.

Source Packet – A CCSDS data unit that contains engineering or payload data generated on board the spacecraft.

Test Data Base Set File – A file containing test data patterns, e.g., a set of CADUs with Multiplexing Protocol Data Units (M_PDU) in their data fields.

Test Data Update File – A file containing update information to a test data base set file. It is typically used to update Virtual Channel Data Units (VCDU), packet sequence count, and timecode information, as well as insert errors in the test data base set while the base set is being output repeatedly in a long duration test. By preserving integrity of protocol data in the base set such as sequence counts with real-time updates, a realistic long duration test can be achieved with a relatively small amount of data.

SECTION 2
SYSTEM DESIGN OVERVIEW

This section presents an overview of the design methodology and approaches taken during the VLS design process that led to the current baseline design as documented in this document.

2.1 DESIGN METHODOLOGIES

During the ETS system level design process, a comprehensive architectural study has been conducted by the ETS design team to identify a system architecture that will best suit the need of ETS. The result of this study is documented in the ETS System Design Specification (SDS). As its conclusion, Code 520's Very Large Scale Integration (VLSI) system architecture was chosen over other candidate architectures to be the basis for the VLS for its better performance, lower cost, lower risk, and shorter development cycle. The ETS SDS further allocates the ETS functional and performance requirements to the ETS VLS. The Code 520 development team then analyzed these requirements and further developed detailed system requirements for the ETS VLS, as documented in the ETS VLS SRS. These system design activities provided the ground for detailed design of the ETS VLS.

The goal of the ETS VLS design is to identify the most cost-effective and risk-free system configuration and components that will meet the requirements specified in the ETS VLS SRS. The system design process includes the following major phases:

a. Select a system configuration and define system external interfaces.

b. Identify functional components (Commercial Off the Shelf [COTS], Government Off the Shelf [GOTS], Hardware [H/W], Software [S/W]) in the selected system configuration.

c. Allocate requirements in the VLS SRS to the functional components.

d. Identify development tasks and estimate work load.

The selection of the ETS VLS configuration was based on two existing operational systems, Advanced Orbiting System (AOS) Test Bed and EOS Amplitude Modulated-1 (AM) Integration and Test (I&T) system, and a new developing front-end input processor board. The combination of these systems formed a solid basis for the ETS VLS. Most of their interfaces directly mapped to the ETS requirements. And 70 percent of the hardware and software components can be directly reused with minimal modification.

Having identified the system configuration, the ETS VLS development team allocated VLS requirements to individual functional components and analyzed the match and discrepancies. Where possible, the team adjusted system components to better meet ETS needs at the lowest cost.

Meanwhile, the VLS development team identified areas where COTS and existing GOTS components do not meet the requirements. The team then studied these areas and generated a set of additional hardware and software development tasks to cover these areas.

One of the main features of Code 521 VLSI technology is its modularity. The ETS VLS team carries this feature into the VLS design so that the system will be reconfigurable to accommodate changing requirements, be flexible to meet future mission's needs, and be upgradeable to maintain its technological edge.

2.2 DESIGN APPROACHES

The development of ETS VLS has encountered a series of development and technical risks that affect design options. This subsection presents a summary of these issues and their resolutions.

2.2.1 CHANGING EDOS BASELINE AND REQUIREMENTS

Most of the ETS test configurations and interfaces are centered around the EDOS. In the past year, the EDOS system baseline has gone through a number of major changes. Along with the baseline changes, there are also many significant requirements and interface changes. Even though the ETS architecture has been selected to be independent of EDOS architecture, these changes still have certain impacts on ETS requirements, configurations, and implementations.

To minimize the impact, the ETS VLS team has adopted a modular design approach with which all functional components, software or hardware, are modularized to the fullest extent possible. Each module can be integrated into a system in a plug-and-play fashion so that the system will be scaleable, flexible, upgradeable, and extendible. In most cases if a requirement changes, a VLS component may be added, removed, or modified to satisfy the requirement change without affecting the overall system design and configuration.

However, if the EDOS architecture, interface, performance, or schedule changes significantly, there may be ripple effect on the ETS. These development must be continuously monitored, with EOSDIS project support, and adjustments made if necessary.

2.2.2 INTERFACE ISSUE

One of the major risks in the ETS development is a lack of well-defined interfaces with the EOSDIS operational systems because its development schedule is ahead of EOSDIS systems. As a result, many of the preliminary designs are based on assumptions and informal user input. To mitigate the risk, the ETS VLS team has worked closely with the EDOS development team in their interface development efforts. The ETS VLS team has reviewed the working draft of the ICD at various stages and returned numerous corrections and suggestions, many of which have been adopted by the development team. With this approach, the development team can match their design closely to the evolving operational system.

2.2.3 TECHNICAL CHALLENGE

Although the ETS VLS is built on existing configurations and most of its components are reused from other projects, there are still a number of technical challenges that are yet to be met. These include the command data block validation, telecommand block forwarding, and CLTU processing. To understand these technical risks and find appropriate solutions, the ETS VLS team has undertaken a number of prototyping activities. Through these prototyping efforts, the team was able to quantify the risk, identify additional development, and select COTS components that could meet requirements at the lowest cost. A summary of this prototyping is given in Section 2.3.

2.2.4 RETURN LINK DATA PROCESSING FUNCTIONS

One of the VLS functional requirements calls for generation of EDUs from user-provided spacecraft test data. This requires the VLS to perform EOS return link data processing functions, including frame synchronization, Reed-Solomon decoding and error correction, packet extraction and reassembly, CLCW extraction, quality and statistics generation, and EDU generation. These are complex operations and have aroused some concern about the cost and schedule of the implementation.

2.2.5 SYSTEM TEST

The VLS is a complex system designed to fulfill complicated functional requirements and stringent performance requirements. It is a challenge to have all of these requirements fully exercised and tested. The ETS VLS team meets this challenge by having a test engineer participate in the design process. The test engineer's responsibility is to ensure that all requirements are verifiable and the system is testable. With an intimate knowledge of how the system

should work, the test engineer will be able to develop an effective and efficient test plan that will exercise all system components and verify their performance with minimum efforts. Also important status information will be collected through these tests that will help pinpoint a problem should it occur during a test.

2.2.6 USER INTERFACE

The Telemetry Processing Control Environment (TPCE) allows the operator to set up the VLS, flow data through the system, and monitor system status from a remote (or local) location that is connected through Ethernet. TPCE runs on an HP-755 workstation using X-windows.

TPCE receives and sends setup, status, and data through the Master Controller Card (MCC) of the VLS. Information is passed between the two systems packed in Modular Environment for Data Systems (MEDS) messages, which provide the translation of information in a format that VLS can process and produce. TPCE translates input from the VLS, creates displays, and updates screens accordingly. It also receives commands and data from the operator, and transfers that information to the VLS.

TPCE is a menu-driven window system. Multiple windows can be open, which provides the operator with a versatile and efficient method to access information, monitor status, and perform tasks. Windows may be opened and closed as needed, screen positions may be changed, and screens may be closed into icons.

2.3 SCSI-2 INTERFACE EVALUATION

The EOS Simulator Card will be using a Small Computer System Interface-2 (SCSI) Mezzanine Card to get the simulation data off a 4-GByte SCSI hard disk. The SCSI-2 Mezzanine Card uses a NCR53C710 SCSI processor chip to interface with the SCSI hard disk and the EOS Simulator Card. The NCR53C710 is a fast SCSI-2 controller chip with a maximum synchronous transfer rate of 10 Mbytes/sec. The requirements for the ETS LRS simulate and update data transfer is within the specifications of this SCSI processor chip.

SECTION 3
SYSTEM DESIGN

3.1 INTRODUCTION

The VLS is a high-performance forward-return link telemetry data simulation and processing system. It is based on the operational AOS Test-bed, EOS AM-1 LRS-VME, and Front-End Processor (FEP) system architectures using VLSI technology. It features VME open-bus architecture, NASA-developed telemetry processing Application Specific Integrated Circuits (ASIC) and boards, state-of-the-art COTS components, and real-time software.

The VLS is controlled through the ETS CDS, a UNIX-based control environment that supports both manual and automatic operations. The CDS features a Graphical User Interface (GUI), scheduling capability, and many standard operational functions. It is described in detail in Section 9.

Figure 3-1 illustrates a detailed system diagram of the VLS.

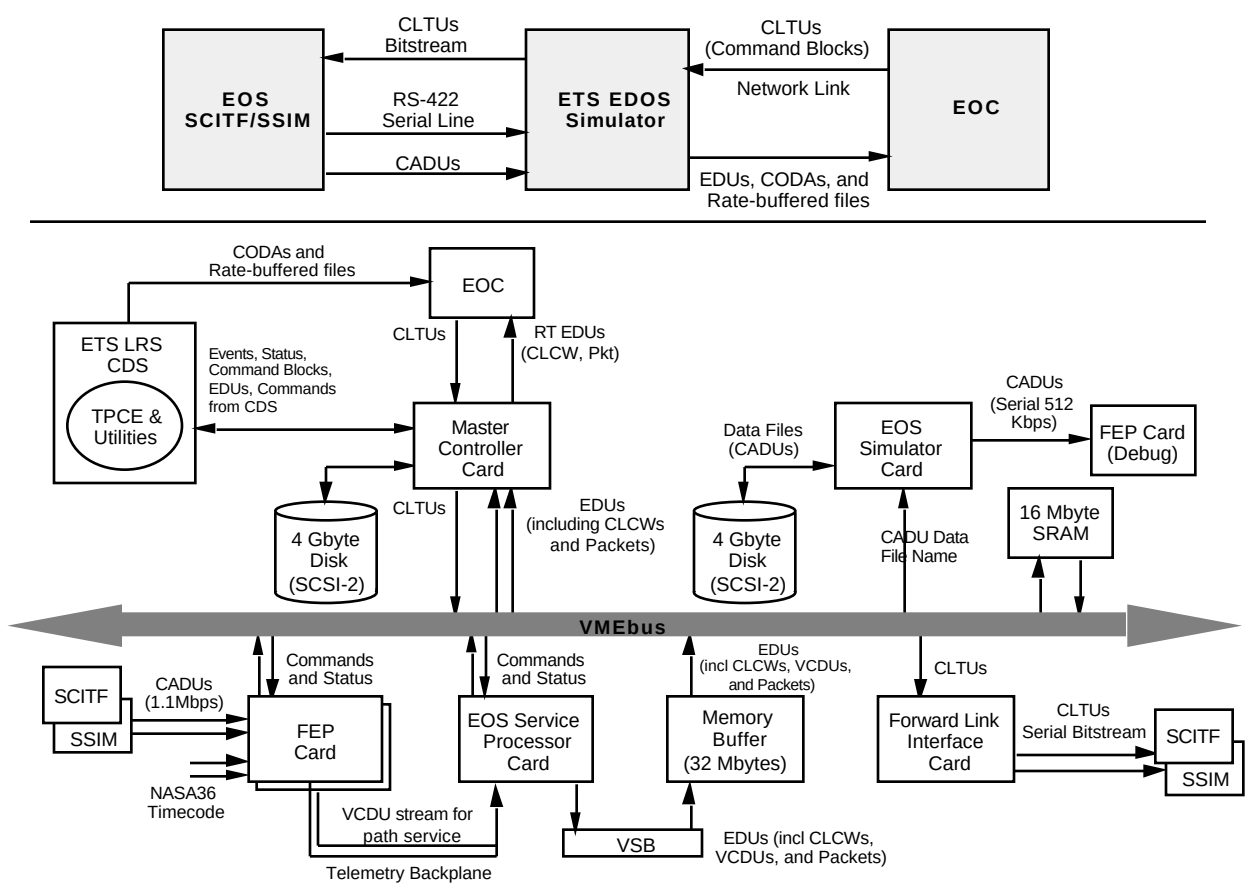


Figure 3-1. VLS System Diagram

3.2 FUNCTIONAL OVERVIEW

The VLS is designed to test the communication links between the space and ground network interfaces to EDOS for the normal mode of operation. The VLS will act as an EDOS simulator to support interface testing between the EOC and the spacecraft. The VLS will receive spacecraft commands in the form of command data blocks. The VLS

will receive low-rate S-band return-link data in the form of CADUs. The VLS will perform the EDOS processing functions on return-link data streams.

The main functions of the VLS are summarized into the following major functional areas:

a. System.

b. Forward-link (uplink) CLTU output processing.

c. Return-link data (CADU) processing.

NOTE: Refer to ETS LRS System Requirements Specification (Volume 4), listed in Section 1.3.1, Parent Requirements Documents, for detailed information.

3.3 SYSTEM

The LRS is composed of the VLS and the CDS. Most of the forward- and return-link processing functions listed below are provided by the VLS. The CDS supplies the user interface, provides real-time control, and is used to monitor operational status, functions provided by TPCE. In addition, the CDS support software, as part of the Utilities and Operations Management Data (OMD) software, provides the rate-buffered file production and Customer Operations Data Accounting (CODA) generation functions. The system provides the following functions:

a. Receive low-rate spacecraft data as CADUs in serial bitstream from the SCITF and the SSIM; perform frame synchronization, Reed-Solomon decoding and error correction, packet reassembly, command link control word (CLCW) extraction, and EDU construction.

b. Annotate packet (in appended EDOS service header) with actual quality and accounting information (rather than simulated header fields).

c. Transmit real-time EDUs (packets and CLCWs) to the EOC.

d. Provide rate buffering of low-rate playback data by creating a file of EDUs and performing a file transfer of the rate-buffered data file to the EOC.

e. Support two return link channels simultaneously.

f. Receive command data blocks from the EOC.

g. Perform EDOS forward-link processing; i.e., verify ground message headers, extract CLTUs and acquisition sequences, and output as serial bitstream.

h. Transmit forward link as serial bitstream (clock and data) to SCITF or SSIM.

i. Generate and transmit CODA reports in real time to reflect the actual return-link and forward-link data processing.

j. Transmit the rate buffered file production.

3.4 FORWARD-LINK (UPLINK) CLTU OUTPUT PROCESSING

In simulating EDOS, the VLS will receive the CLTUs in the form of EDOS command data blocks from EOC through EBnet using User Datagram Protocol (UDP) and Internet Protocol (IP). The VLS will check the command data blocks ground header. The VLS will send an event message to CDS for command data block status.

If the command header is validated, the VLS will strip out command data block ground message header and forward the rest of data (CLTUs) to SCITF/SSIM.

If a command header fails the validation, the VLS will send a message to CDS when it discards a command data block. The VLS will maintain a circular buffer to store discarded command data blocks for snapshot and debugging.

3.5 RETURN-LINK DATA (CADU) PROCESSING

In order to process the low-rate data generated by the EOS AM-1 spacecraft for use in tests with the EOC, the VLS will provide functions to capture and process CCSDS return-link telemetry. The operation is based on processing sessions. Statistics are collected and reported for each session.

A session begins when the VLS is set up and enabled by the host, and ends when it times out after a user-specified time period during which no data is present as input. A session can also be terminated by an explicit command from the host. The start and termination of sessions can be executed automatically by the host.

When serial data is ingested into the system, the VLS will perform frame synchronization to form CADUs and perform Reed-Solomon error correction on CVCDUs. It will then extract source packets contained in the VCDUs and construct EDUs. CLCWs may be extracted if they are present. The VLS will distribute EDUs from all or selected sources and CLCWs to EOC in real-time during the processing session. The VLS also will transmit all EDUs from a solid state recorder playback to the CDS for rate-buffered file generation. The VLS will periodically collect processing status and generate a CODA report, which will be sent to the EOC during the session.

At the completion of a session, the VLS will generate a summary report of accounting and data quality information for the telemetry that was processed.

Detailed requirements for each stage of telemetry processing are described below.

3.5.1 FRAME PROCESSING

The VLS will be able to receive return-link data from an external source via the RS-422 serial input port. The VLS will perform all frame-level processing in support of the CCSDS AOS VCDU Grade-2 service for return-link data.

The VLS will perform synchronization strategy, Reed-Solomon error detection and correction, and time-stamping functions on each CVCDU. The VLS will generate annotation for each VCDU with the following information:

a. Frame synchronization and Reed-Solomon decoding quality trailers.

b. Flags defining error condition(s).

The VLS will be able to process and provide statistics on 24 VCIDs, which is less than the ETS initial requirements, but sufficient to handle EOS AM-1 requirements.

3.5.2 PACKET PROCESSING

For each processing session, the VLS will extract and reassemble source packets from the VCDUs. Packet-level quality and accounting data is collected and reported to the user.

The VLS shall be able to identify independent sources by any combination of stream number, SCID, VCID, and APID. The VLS will discard and account for fill packets. The VLS will generate an EDOS Service Header (ESH) as specified in the EDOS External DFCD to form an EDU. The VLS will maintain and report the EDUs delivery status. The VLS shall maintain and report the CLCWs delivery status.

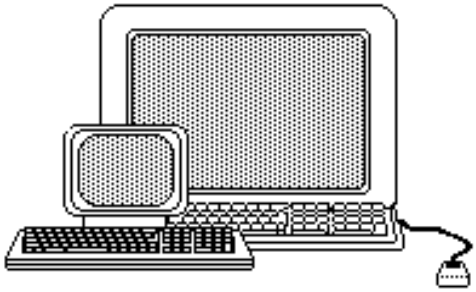
3.6 SYSTEM ARCHITECTURE

The VLS is based on VME standard open-bus architecture. COTS components are integrated together with the Code 521-developed telemetry processing functional components in a 10-slot VME rack along with SCSI disk drives and a power supply.

While the COTS components provide system base functions such as system control, network interface, data storage, and memory buffers, Code 521-developed functional components will perform most of the forward- and return-link processing functions. These functions include command block header validation, telecommand forwarding, low-rate

spacecraft telemetry data output simulation, frame synchronization, Reed-Solomon decoding and error correction, packet processing, and EDU generation.

SECTION 4
USER INTERFACE



This section discusses the MEDS-based Operations Manager (OPMAN). OPMAN provides a local user interface that allows for the overall configuration and status monitoring of each MEDS subsystem (card).

4.1 SYSTEM SETUP, CONTROL, AND STATUS

Catalogs are the foundation of system processing. They consist of files that define system configuration, data flow, and data output. The VLS is versatile and provides a wide range of capabilities; however, this flexibility also demands proper catalog setup to ensure correct data processing. System processing is begun by activating a catalog. OPMAN allows the operator to create, edit, and activate catalogs.

System processing is begun by activating a catalog. Catalog pages provide the setup parameters that must be defined for each card. Catalog pages allow the operator to define input source, output destination, and processing mode. Once a catalog is defined and tested, it is ready to be used for data processing.

When data is flowing, each card maintains status and counts, which the MCC gathers. The MCC makes this information available to OPMAN for display. Each operator interface has a system status screen and individual status screens for each card. These screens allow the operator to monitor system processing and output.

OPMAN provides menu-selectable commands that control the system, and that include catalog activation and access to status screens. The MEDS documentation should be referenced to learn the capabilities and choices available with each operator interface.

4.2 LOCAL OPERATIONS/OPMAN

OPMAN allows the operator to create, edit, and activate catalogs. OPMAN runs on a VT-100 compatible terminal that is connected to the MCC on the system. It allows the operator to set up the system, flow data, and monitor system status. OPMAN is thoroughly documented in the MEDS User's Guide, which provides a detailed description of each OPMAN screen and all of the parameters that the operator can define. OPMAN can be used as soon as the system boot sequence is complete.

OPMAN's main menu lists the primary commands available and provides a brief tutorial. From this screen, the operator accesses submenus that allow system command, setup, and monitoring. The following sections briefly review common procedures that are performed when operating the system. The OPMAN window (Figure 4-1) allows users to access to help information, setup individual subsystem, and display the FEP card status.

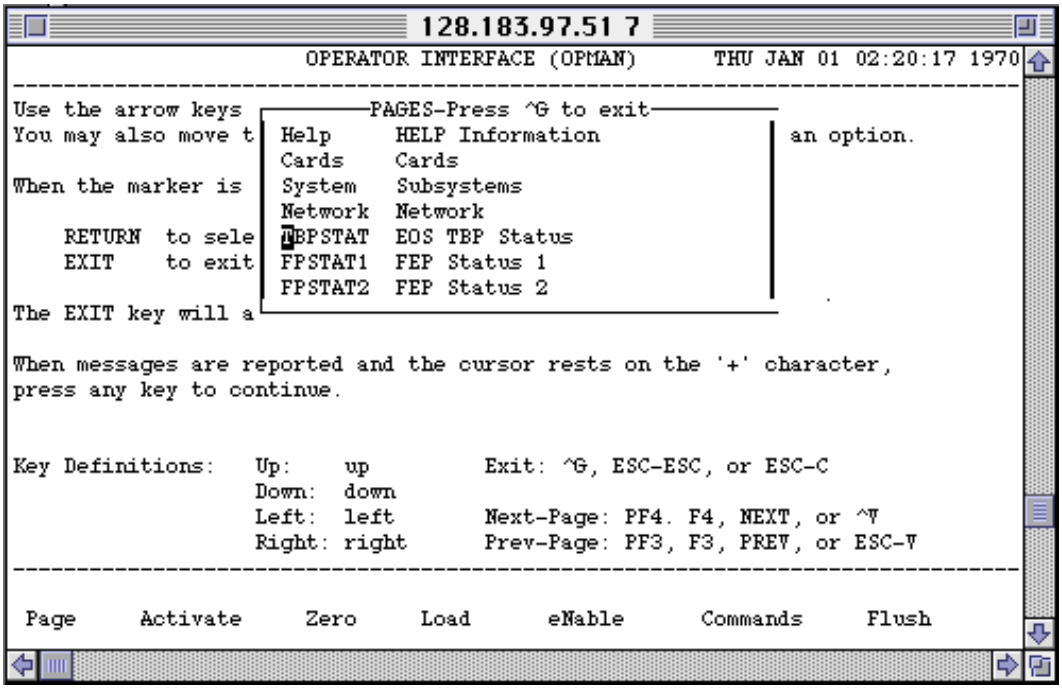


Figure 4-1. ETS-LRS Main OPMAN Window

4.2.1 ACTIVATING A CATALOG

The ACTIVATE command activates a catalog. This command provides two functions: it loads the catalog that contains the configuration information for the appropriate cards (subsystems), and it enables each of the cards, which allows the system to process data.

When the ACTIVATE command is implemented, it lists the current directory and asks the operator for the name of the catalog to be activated. The directory can be changed through the FILEINFO command.

4.2.2 LOOKING AT STATUS

OPMAN provides various status pages; each card in the system has its own status page and a system status page overviews the entire system. Both the system and the individual card status screens can be accessed by the PAGE command.

The system status screen lists cards in the system and each card's VME base address, Processor Communications Area (PCA) base address, and status base address. This page also indicates if each card is healthy (indicated by OK), enabled (ENA), or disabled (DSB). If a catalog is currently loaded into the subsystem, it is listed along with any comments pertaining to that catalog.

Each card-level status page is card-dependent. The MEDS User's Guide provides a complete description of each status screen. The operator watches these screens as data flows through the system, and uses them to ensure correct processing. For example, the following information about a card can be gathered from its status pages: enabled (the card is activated and processing data), the name of the catalog that is loaded, the number of blocks or frames that have entered the card, the number of blocks or frames that have exited the card, the mode in which the card is working, the number of sequence errors that have occurred, the size of blocks or frames, and the data's destination.

4.2.3 SHUTTING OFF A CATALOG

The SHUTDOWN command stops current catalog processing. It disables all cards that are currently enabled, and it clears the catalog name from system processing.

4.2.4 EDITING A CATALOG

The EDIT command provides access to all catalogs in the current directory, or allows the operator to create a new catalog. To correctly define a catalog, it is essential to be familiar with the data that the system is processing. The data defines many of the entries that are made throughout the catalog pages; however, there are several catalog entries that are dependent upon the VLS processing mode.

4.3 SAMPLE OPMAN WINDOWS

A sample LRS FEP main setup is shown in Figure 4-2. The main setup serves as both a data flow diagram and a top level of a hierarchical menu of setup windows. Data flow is viewed as entering the FEP from an entry in the far left “Input” column. Data flows to the right into the corresponding subsystem in the “Functions” column, and then flows down through all other activated subsystems in the column. Data then flows to the right to the two standard output channels and the High-rate Telemetry Backplane (HRTB).

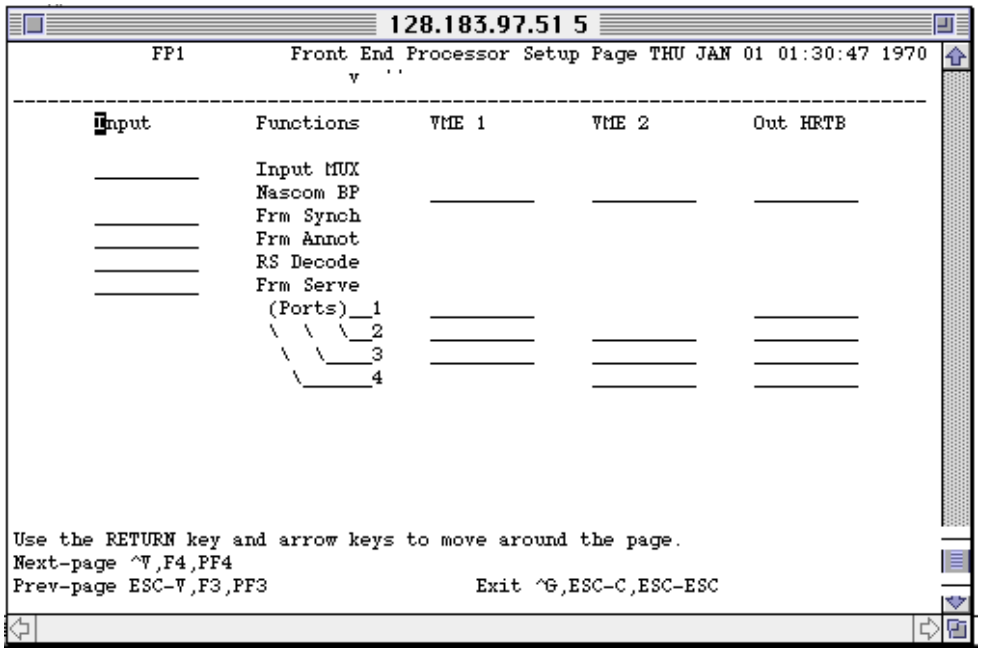


Figure 4-2. ETS-LRS Main Setup Window

To setup a subsystem (e.g., Synchronizer Card), use the cursor keys to select the desired subsystem on the main setup page and press Return. This will bring up the setup page for that subsystem. Refer to Figure 4-3 for the Synchronizer Card setup page and Figure 4-4 for the Frame Annotater setup page.

To edit the Synchronizer Card (or any other subsystem), move the cursor to the desired setup variables and enter the new values. Numeric values can be entered directly. Boolean values like “Enable/Bypass” and “Yes/No” are toggled by pressing Return. Pressing Return on variables that require one of a specific set of values (e.g., data direction can be forward, reverse, or both) brings up a small menu of the possible values. Some variables (e.g., Cycle Redundancy Check [CRC] and Pseudonoise [PN] decoding) require several variables to fully describe, and have their own setup windows. To exit the window, press ESC twice.

```
128.183.97.51 7
                                v  ''
-----------------------------------------------------------------------------
       Input          Functions       VME 1          VME 2          Out HRTB

       ________       Input MUX
                     ------------Frame Synchronizer Setup------------------------
       Accept  True__ & Forward_ Sync Patterns     Frame Size   : 0000
               Forward           Reverse           Data Compr   : Bypass_
       Sync  0x1ACFFC1D        0xB83FF358          Corr Invert  : None
       Mask  0xFFFFFFFF        0xFFFFFFFF          Reverse Data : No_
       Size    4 bytes                             Flywheel Dur : 00
       Bit Tol Fw-Tru   Fw-Inv   Rv-Tru   Rv-Inv   Check Dur    : 00
       Search  00       00       00       00       S/C Slip Tol : 00
       Check   00       00       00       00       L/F Slip Tol : 00
       Lock    00       00       00       00       Best match   : No_
       CRC Check :Bypass    PN Decoding: Bypass    Output On    : Lock_
                           ---------ESC-ESC to exit---------------------------


Use the RETURN key and arrow keys to move around the page.
Next-page ^V,F4,PF4
Prev-page ESC-V,F3,PF3                    Exit ^G,ESC-C,ESC-ESC
```

Figure 4-3. Synchronizer Card Setup Window

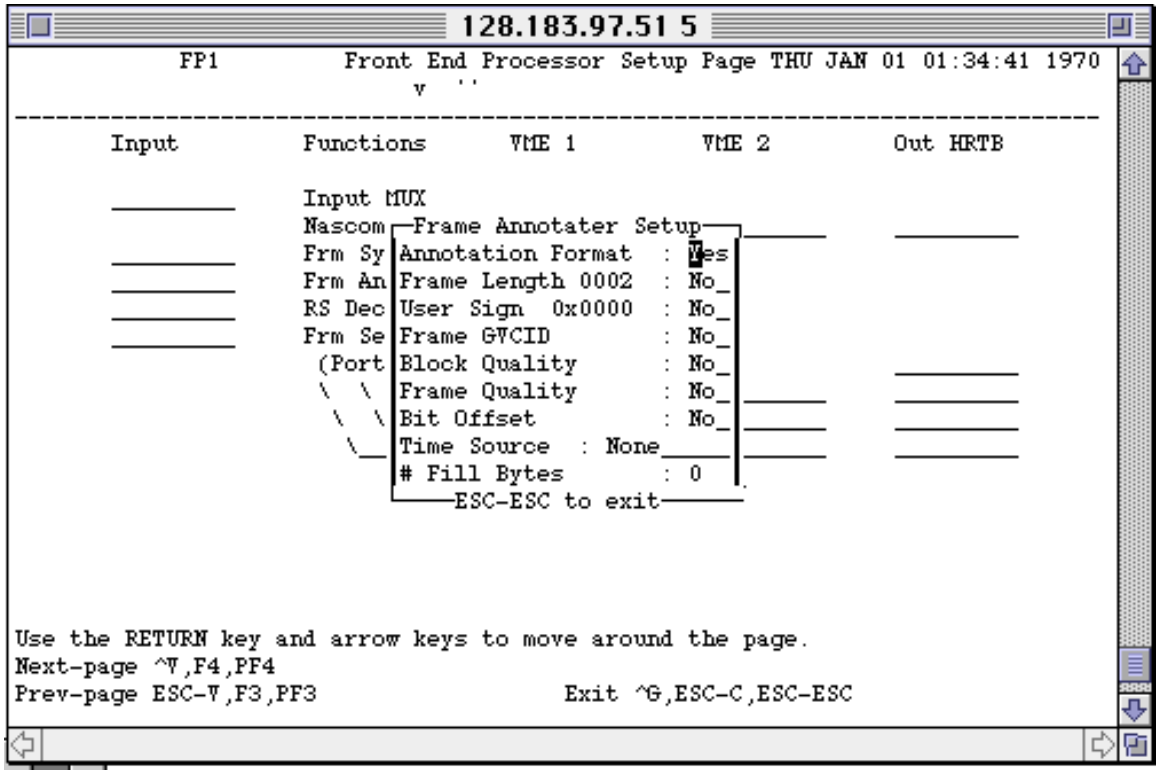


Figure 4-4. Frame Annotater Setup Window

Figure 4-5 depicts the setup page for Reed-Solomon decoding function. Some entries (e.g., Frame Size) are not editable if they can be calculated from the setups from previous subsystems. Output templates can be programmed for mission-specific needs, or custom designed by the user.

```
128.183.97.51 5
FP1          Front End Processor Setup Page THU JAN 01 01:36:33 1970
                  v  ''
-------------------------------------------------------------------------------
Input          Functions      VME 1          VME 2          Out HRTB
                         ┌──────Reed-Solomon Setup──────┐
_________      Inp       │Subsystem           : Enable  │
               Nas       │Encoded Data Offset : 02      │____   _________
_________      Frm       │Frame Length        : 0002    │
_________      Frm       │R-S (10,6) Header   : Bypass_ │
_________      RS        │R-S (255,223) Block : Bypass_ │
_________      Frm       │  Codeword Length   : 033     │
                (P       │  Interleave Level  : 1       │   _________
                \        │RS Qual Annotation  : Bypass  │____ _________
                 \       │ Static Ann: 0000000000000000 │____ _________
                         │Reject Unroutable   : No_     │____ _________
                         │    Uncorrectable   : No_     │
                         │             Short  : No_     │
                         │              Long  : No_     │
                         │Output Template: None_________│
                         └──────ESC-ESC to exit─────────┘

Use the RETURN key and arrow keys to move around the page.
Next-page ^V,F4,PF4
Prev-page ESC-V,F3,PF3                  Exit ^G,ESC-C,ESC-ESC
```

Figure 4-5. Reed-Solomon Setup Window

Figure 4-6 depicts the setup window for the Frame Server function. The routing table is stored as a separate file from the setup catalog. A routing table file editor is included with the FEP setup software. The editor uses setup information from the Frame Server subsystem to ensure that the routing table does not produce incompatible setups with the Frame Server setup.

```
128.183.97.51 5
FP1          Front End Processor Setup Page THU JAN 01 01:38:49 1970
                      v  ''
--------------------------┌──Frame Server Setup──┐-------------------------
      Input        Funct │1) Define Services    │2        Out HRTB
                         │   A. _____________   │
      _________    Input │   B. _____________   │
                   Nasco │   C. _____________   │_____    _________
      _________    Frm S │                      │
      _________    Frm A │2) Select Fixed       │
      _________    RS De │   Port 1: Disabled___│
      _________    Frm S │   Port 2: Disabled___│
                    (Por │   Port 3: Disabled___│         _________
                    \  \ │   Port 4: Disabled___│_____    _________
                     \   │                      │_____    _________
                      \_ │3) Select Routed      │_____    _________
                         │   Table:_____________│
                         │          Edit    Load │
                         │          Clear   Save │
                         │                      │
                         │4) VCID Status        │
                         └────ESC-ESC to exit───┘
Use the RETURN key and arrow keys to move around the page.
Next-page ^V,F4,PF4
Prev-page ESC-V,F3,PF3                  Exit ^G,ESC-C,ESC-ESC
```

Figure 4-6. Frame Server Setup Window

To display the descriptions of any setup variable in any setup window, simply move the cursor to the desired variable and press Ctrl-A; this will invoke a help window for the selected parameter.

The LRS FEP status is divided into two pages. Figure 4-7 displays a top-level overview of the setup and status of the dataflow. Input source, subsystem health and frame counters, and output counters are displayed alongside a diagram of the FEP dataflow. Error flags next to specific subsystems alert the user of problems within those subsystems.

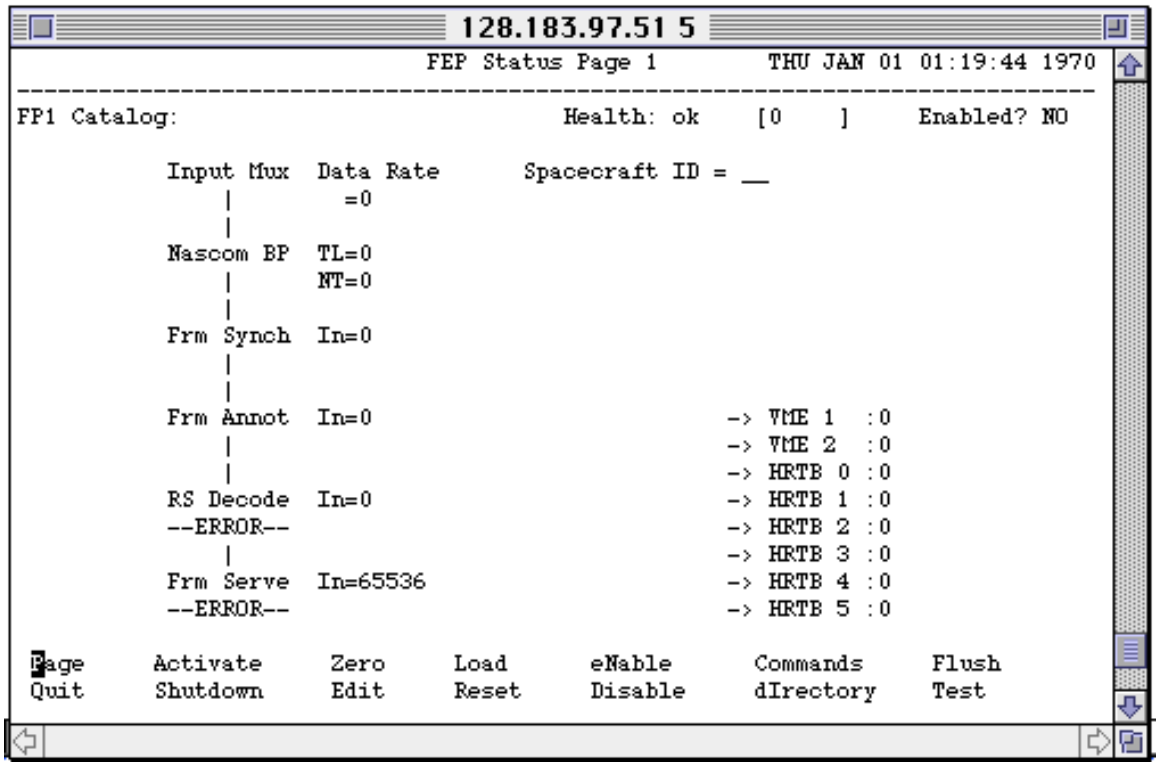


Figure 4-7. FEP Status Page 1

Figure 4-8 (FEP Status Page 2) displays a detailed breakdown of the status from the Frame Synchronizer, Reed-Solomon Decoder, and Frame Service VCID breakdown.

```
128.183.97.51 5
                                FEP Status Page 2        THU JAN 01 01:23:03 1970
-------------------------------------------------------------------------------
FP1 Catalog:                                Health: ok    [0    ]     Enabled? NO

  Frame Sync                    Reed Solomon              Frame Service
Frames In    0            Input Frm     0            Frame Count 65536
Frames Out   0            Long Frm      65536
Good Syncs   0            Short Frm     0            VCID V Count       Seq Err
Search       0            Filter Frm    131072        46  2 177         221
Check        0            Corr Frm      65536         46  2 177         223
Lock         0            Uncorr Frm    65792         46  2 177         225
Flywheel     0            Unroute Frm   0             46  2 177         227
B2Search     0            Corr Hdr      0             46  2 177         229
Forward CRC  0            Uncorr Hdr    65536         0   2 0           0
Reverse CRC  0            Corr Cdw      0             0   2 0           0
Blocks In    0            Uncorr Cdw    65536         0   2 0           0
Poly Err     0            Corr Frm Err  0             0   2 0           0
Slip Frm     0            Corr Hdr Err  65536         0   2 0           0
FrmSync Err  0            Corr Bit Err  0             0   2 0           0
Current      Inv-Rev                                  0   2 0           0


 Page      Activate      Zero      Load      eNable      Commands      Flush
 Quit      Shutdown      Edit      Reset     Disable     dIrectory     Test
```

Figure 4-8. FEP Status Page 2

SECTION 5
OPERATIONS SCENARIO

This section describes how the LRS will be set up, receive input, process data, and generate output for two distinct scenarios: return link processing and forward link processing. The VLS receives return-link data and transmits forward-link data from/to EOSDIS elements and other ETS components through two types of interfaces: the RS-422 serial port and the Ethernet network.

5.1 RETURN LINK OPERATION

Figure 5-1 illustrates a scenario of return link processing. In this configuration, return-link data is received and processed first by the Front-end Processor Card. The card performs frame-level processing (e.g., frame pattern synchronization, Reed-Solomon decoding, error detection and correction). The Front-end Processor Card also has a timecode interface to ingest external timecodes for time-stamping each frame during return-link data processing. Valid frames will then be passed on to the Service Processor Card for packet assembly service. All assembled packets and CLCW extracted from frames will be prepended with EDOS headers to create EDUs. The Service Processor Card will pass all EDUs to the MCC. The EDU handling task running on the MCC will determine the routing of real-time EDUs. All real-time EDUs will be forwarded to EOC. All EDUs will be sent to the CDS for archiving and rate-buffered data generation. The CDS will receive status information of all VLS subsystems and formulate CODA reports. The CDS will then send CODA reports and rate-buffered data to EOC.

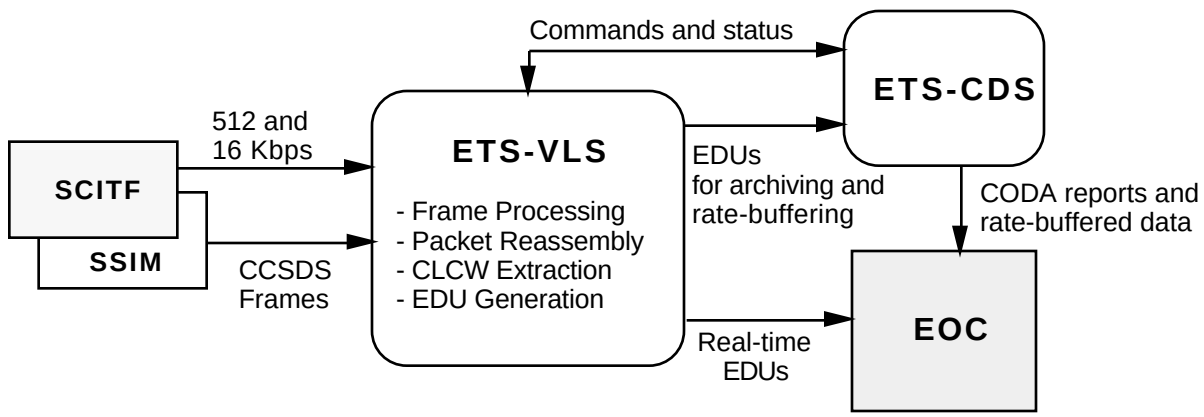


Figure 5-1. ETS-LRS Return Link Scenario

5.2 FORWARD LINK OPERATION

Figure 5-2 illustrates a scenario of forward link processing. The VLS receives command data blocks from the EOC. All command data blocks are passed on to the CDS for archiving. CLTUs in a command data block that passes header validation process are deposited in a mailbox. The Forward Link Interface Card periodically reads its assigned mailbox and passes CLTUs through to SCITF and SSIM. If the Forward Link Interface Card is initially setup to output CLTUs at a higher rate (10 Kbps), all subsequent lower rates will be supported. If the Forward Link Interface Card is initially setup to output at lower rate (125 bps), a new catalog has to be uploaded to change the output rate to a higher rate (10 Kbps). It is expected that EOC requests the change in output rate via a voice loop.

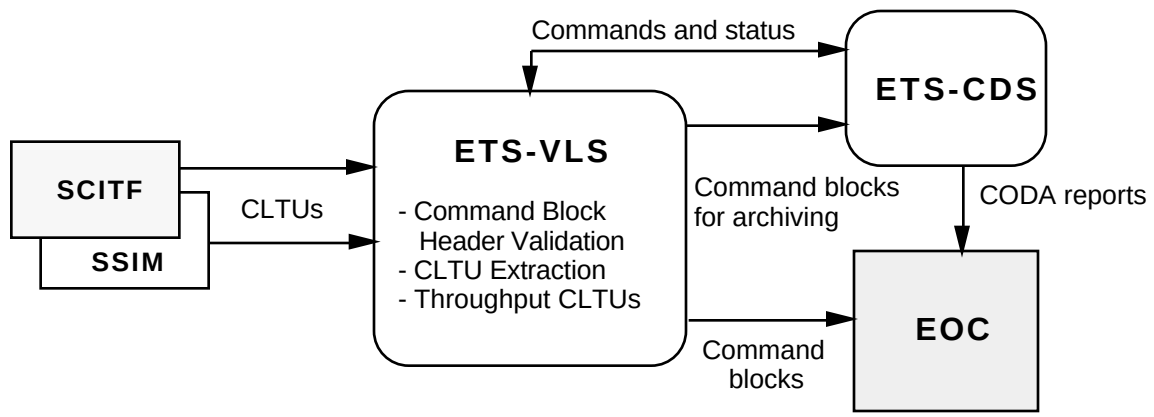


Figure 5-2. ETS-LRS Forward Link Scenario

5.3 FEP CARD MODES OF OPERATION

The FEP Card has three modes of operation: block, frame, and test. Frame mode is further distinguished by the frame processing enabled. Block and frame modes are operational modes; test mode is a self-diagnostic mode. Data can be processed simultaneously in both block and frame modes; however, different output channels must be selected for each mode.

5.3.1 BLOCK MODE

Block mode processing accepts input data as Nontelemetry (NTLM) data units. This processing mode requires serial input data from one of the FEP Card's external interfaces. From output processing, frames are output in parallel over the P3, or over the Versa Module Eurocard bus (VMEbus) from the data capture channels.

5.3.2 FRAME MODE

In frame mode, data processing can include error control and frame-level service processing functions. This processing mode requires serial input data from one of the FEP Card's external interfaces. The input data may be received in Nascom blocks, or as raw frames. Block data is first processed for block synchronization; all data input must receive frame synchronization. Frames are output in parallel over the P3, or over the VMEbus from the data capture channels. Processing input frames can include annotation, Reed-Solomon decoding, and frame-level service processing. Selection of the functions to be performed for each frame processing mode is independent.

5.3.2.1 Reed-Solomon Throughput Mode

Reed-Solomon throughput mode is available for frames that are not encoded at the transmitter. During throughput mode, the Reed-Solomon processing subsystem passes data unmodified from its input to its output.

5.3.2.2 Reed-Solomon Decode Mode

Reed-Solomon decode mode processes input frames with Reed-Solomon interleave levels of 1 through 8. Decode mode supports error detection and correction of input data streams prior to frame-level service processing.

5.3.2.3 Frame-Level Service Processing Mode

Five frame-level service processing functions are available, in addition to performing no service. Space Link Command (SLC) and insert processing functions are performed on all frames received as input. VCDU, Virtual Channel Access (VCA), and bit stream services are performed based on frame Global Virtual Channel Identifier (GVCID). Up to four modes of service may be selected for operation on a single frame, if each service is routed to a different output channel.

5.3.3 Test Mode

Test mode supports FEP Card self-testing. Input data is received from the Test Generator Mezzanine; output data is received by the data capture or test capture buffers. The input data format is dependent on the data entry point for the test being performed. The Test Generator Mezzanine supports the input format of each FEP Card subsystem. Validation of output data is performed by the Central Processing Unit (CPU) by its interface to the data capture and test capture buffers.

5.4 EOS SERVICE PROCESSOR CARD MODES OF OPERATION

The EOS Service Processor Card has two modes of operation: normal and test. Internal data processing is the same in both modes; however, the input source and output destination differ.

In normal mode, data is input from the HRTB. Data is output via the HRTB and/or the Direct Memory Access (DMA) circuitry.

In test mode, data is input from the header processor. The data content is software-dependent. Typically, the output is sent to the loopback First-In, First-Out (FIFO), and the data is examined via the output processor to ensure that no corruption has occurred.

5.4.1 PIPELINE INPUT INTERFACE

The pipeline input interface receives a parallel input data stream from the HRTB. The HRTB is the third bus connector on the VMEbus. The maximum sustained input rate for this portion of the card is over 500 Mbps.

To achieve higher transfer rates, the HRTB transfers two bytes (with the 9th bits) at a time. The HRTB input logic de-stacks these words into a 9-bit stream to the tribuffer subsystem. The nine bits in an input data unit are defined as follows: eight bits of data and a one bit End of Frame (EOF) mark (9th bit). The 9th bit is used to signal the end of a frame. On the last byte of a frame, the 9th bit is set to one; otherwise, it is zero. The expected data type is CCSDS-defined Version 1 transfer frames or Version 2 CVCDUs with a 9th-bit EOF mark; however, any data unit with a 9th bit EOF mark can be processed.

5.4.2 INPUT RATE BUFFERING

A 4K x 18-bit input synchronous FIFO in series with a 16K x 9 asynchronous FIFO provides rate buffering of data input to the EOS Service Processor Card. The asynchronous FIFO is connected to either the HRTB input bus (via an external P3/J3 connection), or to the header processor data bus (via an internal mezzanine-to-motherboard connection). Bit 3 in the header processor status/control register determines the input source (0 = external, 1 = internal). Header processor input is only used for testing purposes.

5.4.3 TRIBUFFER SUBSYSTEM

The tribuffer subsystem provides temporary storage of data frames; it includes a custom-designed GSFC, Code 521 VLSI Enhanced Tribuffer Controller ASIC and three 8-Kbyte Random Access Memory (RAM) buffers. The subsystem reads data frames from the input FIFO, stores each frame separately, and creates each frame as well as its associated packet primary header(s). After the header processor reads and processes a frame, it outputs that frame to the reassembly RAM subsystem.

5.4.4 REASSEMBLY RAM SUBSYSTEM

The reassembly RAM subsystem accepts data from the tribuffer subsystem, stores it, and outputs it from the EOS Service Processor Card. Data output, which is controlled by the Output Processor, can include fill and/or annotation. The output data destination can be the P3 pipeline, a loopback FIFO, and/or VME Subsystem Bus (VSB) DMA circuitry.

This subsystem consists of a 4-Mbyte RAM buffer and an enhanced RAM controller. The 4-Mbyte buffer is logically partitioned into thirty-two 128-Kbyte partitions that allow separate storage of frames from 32 different virtual channels.

5.4.5 VSB INTERFACE

The VSB interface provides a VSB interface for the output processor and a VSB DMA interface for the reassembly RAM subsystem output. The subsystem uses a Motorola MVSB2400 to provide a master A32/D32 interface to the VSB.

For DMA of reassembly RAM subsystem output data, a 4K x 32-bit FIFO buffers the data and converts it to 32-bit data. The FIFO is logically partitioned into four sections. Each 8-bit data output is stored in a partition, and then when all four partitions are non-empty, the four partitions are simultaneously transferred to the MVSB2400 chip for DMA.

5.4.6 LOOPBACK FIFO

The 16K x 9-bit loopback FIFO is connected to the RAM controller output or, in test mode, to the output processor. This FIFO is typically used for card self-testing.

To check a selected output, the output processor needs to reset the loopback FIFO to clear previous data, load an instruction set into the RAM controller instruction FIFO, and then repetitively read the captured data from the loopback FIFO. Only the first 16 Kbytes are captured; all subsequent bytes are dropped.

Alternatively, the output processor can source data directly into this FIFO and read it back to verify correct FIFO operation.

5.4.7 PIPELINE OUTPUT INTERFACE

The pipeline output interface outputs parallel data to the HRTB, which is the third (J3) bus connector. The typical data output is CCSDS-defined source packets with annotation.

The output from the RAM controller subsystem is nine bits with a write strobe. The nine bits are defined as follows: eight bits of data and a one-bit EOF mark (9th bit). The 9th bit is used to signal the end of the data unit. On the last byte of data and annotation, the 9th bit is set to one; otherwise, it is always zero.

The HRTB output logic stacks the asynchronous 9-bit output from the RAM controller into synchronous 18-bit data for transfer across the HRTB.

The HRTB output logic, which includes the output interface, can be reset in one of three ways: a manual card reset via the front-panel reset switch, a system reset from the VME master, or a software-generated pipeline reset via the header processor.

5.4.8 VME INTERFACE

The VME interface transfers setup data and commands to the quality processor, which then transfers the information to the header and output processors.

The quality processor subsystem can communicate with the VMEbus as a master or a slave via the VME interface. A Motorola MVME6000 provides the master/slave interface to the VMEbus. Programmable Logic Devices (PLD) provide the logic to arbitrate the quality processor's local bus, allowing the VMEbus master to access the local Static Random Access Memory (SRAM) of the quality processor.

5.4.9 QUALITY PROCESSOR

The quality processor subsystem processes quality information and communicates with the VMEbus interface. It also includes a serial Universal Asynchronous Receiver-Transmitter (UART) interface that provides direct terminal interaction for software development and debugging. The quality processor sets up the header and output processors by accessing their local memory. It can also communicate with the header and output processors through Dual-ported RAM (DPR) and may send interrupts to each.

This processor is dedicated to the analysis of extracted data for quality evaluation; it performs several typical functions:

a. Checks quality.

b. Generates a piece assembly list.

c. Generates quality/production accounting statistics.

5.4.10 HEADER PROCESSOR

The header processor subsystem controls the tribuffer RAM subsystem. It also communicates with the quality and output processors through DPR and interrupts. For testing purposes, the header processor can write data into the input FIFO, where the data is then processed and checked in the loopback FIFO. The header processor includes a serial UART interface that provides direct terminal interaction for software development and debugging.

The header processor reads and then processes data that is stored in the tribuffer subsystem RAM. Typically, it reads and processes frame or CVCDU headers and packet headers. It is the only processor with direct access to the input data, and it must provide any data-dependent information required by the quality processor and output processor functions:

a. The output processor requires location information for pieces to be reassembled for output. An indexed listing of this information is passed from the header processor to the output processor via DPR. For example, the header processor may provide start and end locations of packet pieces within each frame.

b. The quality processor also receives an indexed listing of information from the header processor via DPR. Typically, the quality processor requires header-derived information that describes both frames and packet pieces.

5.4.11 OUTPUT PROCESSOR

The output processor subsystem controls the reassembly RAM subsystem and tracks the location of frames and pieces of frames stored in the reassembly RAM. It also communicates with the quality and header processors through DPR and interrupts. It includes a serial UART interface that provides direct terminal interaction for software development and debugging.

The output processor receives an indexed list of data field piece locations within each frame from the header processor via shared DPR. The availability of new data can be signaled either by the tribuffer controller cycle interrupt, or by a message field defined in the DPR.

Based on the header processor data list, the output processor maintains a frame and piece directory. Using this and the assembly list from the quality processor, it generates instructions for data output, with a defined format for that output. Instructions are written to the instruction FIFO. A set of output instructions is generated whenever assembly data is available from the quality processor.

5.4.12 DPR SUBSYSTEM

The EOS Service Processor Card's three microprocessors are configured to be programmed and operated independently. Asynchronous communication between processors is provided either by direct interrupts, or through DPR.

The header/quality DPR is an 8K x 32-bit DPR for a total of 32 Kbytes. It provides the primary communication between the header and quality processor subsystems while the EOS Service Processor Card is running. The header processor passes tribuffer subsystem RAM information to the quality processor via this DPR; the quality processor passes data, status, and commands to the header processor via the same DPR.

The header/output DPR is an 8K x 32-bit DPR for a total of 32 Kbytes. It provides the primary communication between the header and output processor subsystems while the EOS Service Processor Card is running. The header processor passes tribuffer subsystem RAM information to the output processor via this DPR; the output processor sends and receives data to/from the header processor via the same DPR.

The quality/output DPR is an 8K x 32-bit DPR for a total of 32 Kbytes. It provides the primary communication between the quality and output processor subsystems while the EOS Service Processor Card is running. The quality processor passes data, status, and commands to the output processor via the same DPR; the output processor sends and receives data to/from the quality processor via the same DPR.

5.5 FORWARD LINK INTERFACE CARD MODES OF OPERATION

Once the Forward Link Interface Card is completely configured, the timecode interface drives the P3 telemetry pipeline with byte-wide continuous Parallel Binary (PB1) timecode and the Forward Link Card Controller (FLCC) simply waits for data. This data originates from the end-user, and is routed across an Ethernet network. It is stored in a mailbox, or in some allocated system memory specified to the FLCC in initial setup information.

The FLCC begins polling the allocated storage location, and when data is present, transfers the data to local Dynamic Random Access Memory (DRAM). At this point, the FLCC needs to determine whether the data is an AOS, telecommand, or Nascom data structure. The FLCC adds a ninth bit to the last byte of the data if it is an AOS data structure. If the structure is telecommand, it must be filled to ensure an integral number of telecommand codeblocks, which depends on the codeblock length. A ninth bit must be added to the last byte of the data structure if the interface is configured in encode mode. If the interface is configured in throughput mode, the ninth bit must be added to the second to last byte of the data structure.

With the type of data structure known, the FLCC transfers data to the input FIFO of the correct output interface. If the data structure is too large to fit into the FIFO, the FLCC must ensure that the data is cycled into the FIFO until the last byte of the data structure is written without the FIFO ever going empty before the ninth bit is reached. At this point, the hardware takes over and maintains a continuous data stream out of the Forward Link Interface Card until the output interface is turned off.

The Forward Link Interface Card can be configured to operate in either throughput mode or encode mode.

5.5.1 THROUGHPUT MODE

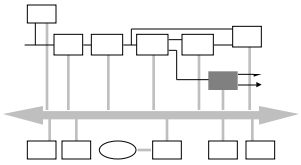
Configure interfaces as throughput, and supply valid CLTUs from the VME interface to the telecommand output interface for serial RS-422 transmission.

5.5.2 ENCODE MODE

Configure interfaces as encode, and supply raw telecommand data from the VME interface for serial RS-422 telecommand transmission.

NOTE: The VLS does not use this mode of operation.

SECTION 6
DETAILED HARDWARE DESIGN



This section provides detailed information on the major hardware components of the ETS VLS. It expands on the information provided in Sections 1 and 2. The following paragraphs describe each component's architecture and its role in the overall system structure. The discussion comprises both commercial and GSFC/NASA Code 521 custom components.

6.1 SYSTEM OVERVIEW

Figure 3-1 illustrates ETS VLS data flow and provides an overview of the relationships between the system components.

6.1.1 INTRODUCTION

The ETS VLS is packaged as a rack-mountable VMEbus-based system. Multiple telemetry processing cards, usually assigned specific functions, provide the system with standard telemetry processing. The primary functions that the system performs include:

a. Simulates and transmits spacecraft data for end-to-end system integrity checks at rates up to 10 Mbps.

b. Simulates two Transparent Asynchronous Transmitter/Receiver Interface (TAXI) streams of instrument packet data at rates up to 10 Mbps.

c. Processes CCSDS return link frames and packets; functions include frame synchronization, Reed-Solomon decoding, virtual channel routing, and fill frame removal at rates up to 1 Mbps.

d. Performs packet reassembly at rates up to 1 Mbps with a maximum peak rate of packets/second To Be Supplied (TBS).

e. Distributes a user-selectable subset of reassembled packets via a standard Ethernet Transmission Control Protocol/Internet Protocol (TCP/IP) network to spacecraft integration and test local area network and workstations at rates up to 1 Mbps.

f. Monitors data quality across all functions.

6.1.2 SYSTEM COMPONENTS

The ETS VLS architecture uses a building block approach that was pioneered at GSFC to facilitate the development of modular pipelined multiprocessing telemetry data processing systems. This approach uses modular COTS and custom processing building blocks in a "plug and play" open-bus architecture to create and configure mission-specific telemetry data systems. The ETS VLS is based on the industry standard VMEbus and the custom HRTB bus, that offloads telemetry data transfers from the VMEbus. The system also includes Ethernet network communications that allows a remote workstation to transfer setup and commands to the system and receive status and packet data from the system.

This combination of commercially available hardware (processors, Ethernet communication, disk controllers, and storage components) and custom logic cards comprise the ETS VLS. Communication between the wide variety of system components is made possible through the VMEbus, the custom HRTB, and generic software that is specifically developed for an environment with diversified hardware.

The major components of the ETS VLS are as follows:

a. A VME standard open-bus enclosure with a power supply and a 10-slot backplane, which includes a VMEbus (J1 and J2) and the HRTB bus (J3).

b. Two commercial VME cards and modules:

(1) MCC that consists of a Motorola MVME-167 single-board computer and interface connector paddle card.

(2) Global Memory (GM) Card (Micro Memory MM6740CN).

c. Five GSFC Code 521 custom cards:

(1) Two EOS Front-End Processor Cards (G1527430, Code 521, NASA/GSFC).

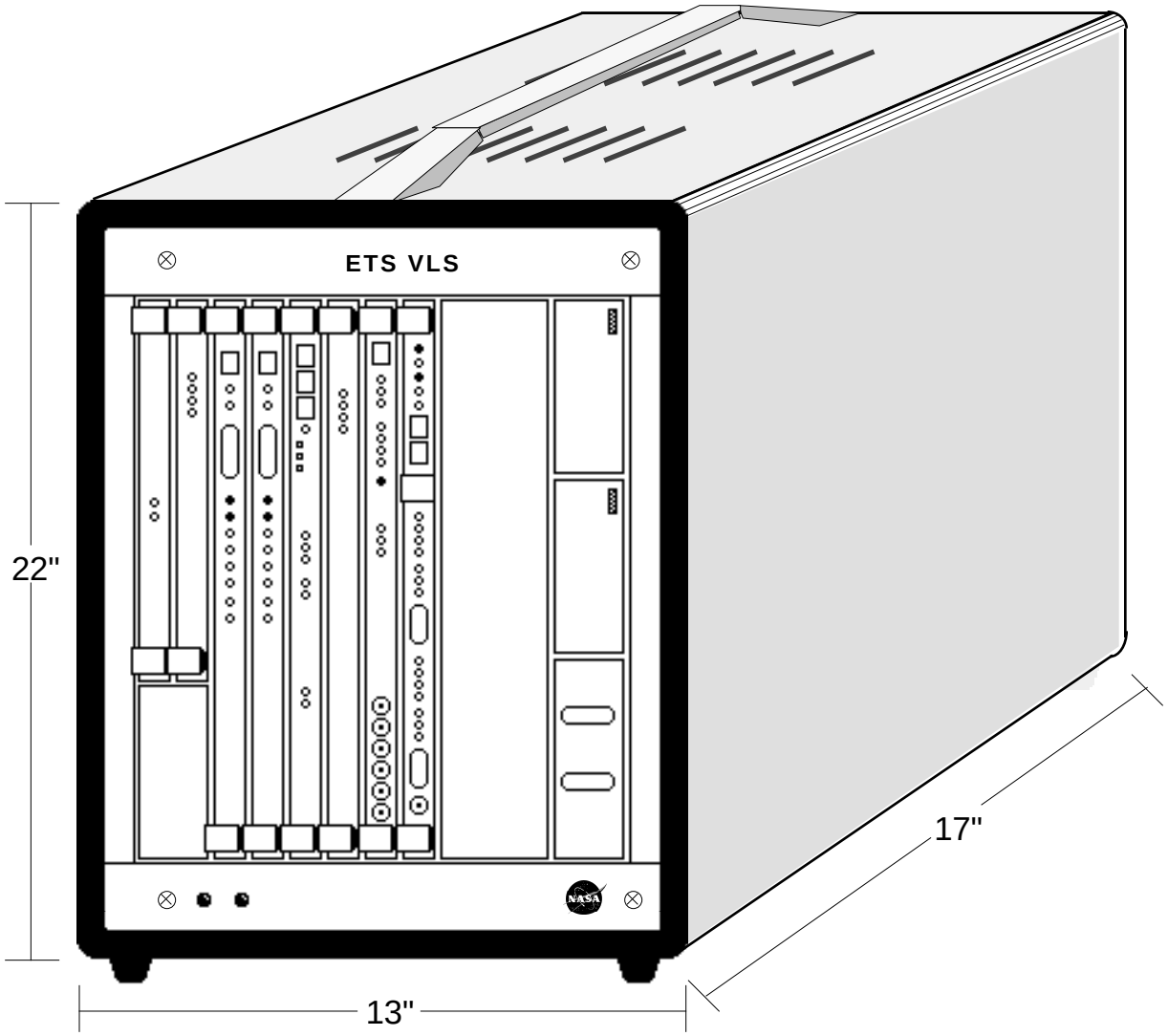
(2) One EOS Service Processor Card (G1527421, Code 521, NASA/GSFC).

(3) One Forward Link Interface Card, Revision B (G1490847, Code 521, NASA/GSFC).

(4) One EOS Simulator Card, Revision A (G1527414, Code 521, NASA/GSFC).

d. A real-time, multiprocessing, operating system environment, which includes VxWorks MEDS.

The ETS VLS is housed in a VME card chassis. The card cage is partitioned into two separate areas: a commercial area and a custom card area. The commercial area houses the COTS processors, disk modules, and interfaces. The custom area houses the special data processing cards that are necessary to implement the return link processing and instrument data simulation functions. The VMEbus transfers control and status information between the custom card area and the commercial area. Figure 6-1 illustrates the ETS VLS chassis.



Master Controller Card
SRAM Card
Front End Processor Cards
EOS Service Processor Card
DRAM Card
EOS Simulator Card
Forward Link Interface Card
System Disk
Data Storage Disk

NOTE: Cards are read from left to right.

Figure 6-1. ETS VLS Chassis

6.2 COMMERCIAL COMPONENTS

Commercial components refer to those functional elements, modules, and subsystems that are supplied by vendors external to Code 521 GSFC/NASA. In the ETS VLS, commercial cards provide for the exchange and storage of data, control, status, and quality information. They provide the processing, storage, and Input/Output (I/O) functions required by the overall ETS VLS application.

Section 1.1.3, Reference Documents, provides a list of commercial card documentation that should be referenced for detailed questions about each of the system's commercial hardware elements.

6.2.1 SYSTEM DISK AND INTERFACE (MODULE)

The system disk and interface is a commercial module with a 4-Gbyte hard disk. It provides storage for system configuration files and the software modules to be downloaded to each functional element card.

6.2.2 MASTER CONTROLLER CARD (MVME-167-032)

The MCC is a 33-MHz 68040-based Motorola MVME-167SA-2 CPU board. The MVME-167-032 is a 6U card with 32-bit access through both VME connectors (P1 and P2). The board has several built-in functions including 8 Mbytes of on-board shared parity-protected DRAM, four serial ports, parallel port, SCSI bus controller, tick timer, watchdog timer, time-of-day clock/calendar, and A32:D32 VMEbus interface with system controller functions. The serial ports provide a user interface specific to this card. Onboard functions include: 68882 floating point coprocessor, Ethernet interface controller, and SCSI bus interface with DMA channel. The card requires power supplies of +5 V @ 7 Amps maximum, +12 V @ 1 Amp maximum (100 mA maximum with no Ethernet LAN), and -12 V @ 100 mA maximum. A detailed description of the MVME-167 can be found in Motorola's MVME167S MPU VME Module User's Manual.

The Ethernet interface controls and moves data and command traffic over Ethernet via TCP/IP protocols; it interfaces the data flow between the ETS VLS and workstation(s). The Ethernet interface also controls and accounts for data flowing through the network to and from the ETS VLS.

It includes a 10-MHz Motorola 68010 microprocessor that autonomously handles the transmission and reception of TCP/IP packets from the network.

6.2.3 SYSTEM MEMORY CARD (MM6740CN)

The system memory card is a 6U Micro Memory MM6740CN high-speed 16-Mbyte SRAM memory board. The MM6740CN can transfer 32-, 16-, or 8-bit data over the VME and may be addressed using the 32- or 24-bit VME address paths. It can support VME block transfers. The board generates and stores a parity bit for each byte memory location during write cycles and checks it during read cycles. A status bit is set to indicate parity errors. It provides for the temporary logical buffering of data. The card requires a power supply of +5 V @ 7.75 Amps maximum. A detailed description of the MM6740CN can be found in Micro Memory's MM6740CN User's Guide.

6.2.4 GLOBAL MEMORY CARD (MM6346D)

The Global Memory Card is a 6U Micro Memory MM6346D high-speed 32-Mbyte DRAM memory board. The MM6346D can transfer 32-, 16-, or 8-bit data over the VME and may be addressed using the 32- or 24-bit VME address paths. It can support VME block transfers. The board generates and stores a parity bit for each byte memory location during write cycles and checks it during read cycles. A status bit is set to indicate parity errors. It provides for the temporary logical buffering of data. The card requires a power supply of +5 V @ 7.75 Amps maximum. A detailed description of the MM6346D can be found in Micro Memory's MM6346D User's Guide.

6.3 CUSTOM COMPONENTS

The ETS VLS includes custom cards with third bus connectors (J3/P3), which together with the HRTB comprise a telemetry data pipeline. The pipeline allows multiple simultaneous pipelined transfers of data between custom card elements without burdening the VMEbus. This "pipeline" technique allows great flexibility in the definition of new mission-specific telemetry processing systems, and provides for greatly increased data processing bandwidth.

The typical custom card (see Figure 6-2) is a 9U card that includes a 3U commercial controller (68030-based MZ 8130) and a 6U custom card section. The commercial CPU acts as the logic controller for the custom card; it controls the setup and execution of the custom hardware. It is memory-mapped into the VMEbus system environment, and it directly interfaces with all commercial components in the VME system. Each card runs programs loaded from the system disk module. The custom hardware is connected to the commercial CPU by a side

connector. The HRTB connector provides a standard pipeline interconnection between custom cards. Each card has one or more input and output ports on the HRTB connector. Each input and output provides 16 bits of data, a data strobe, and a bit that signifies the end of the data structure (i.e., frame, packet, or block) being transferred. The typical custom card has the ability to run, stand alone, and flow simulated input data using a test program module that runs at either system or card-level to thoroughly verify card operation. This avoids the need to bring in external test and simulation equipment.

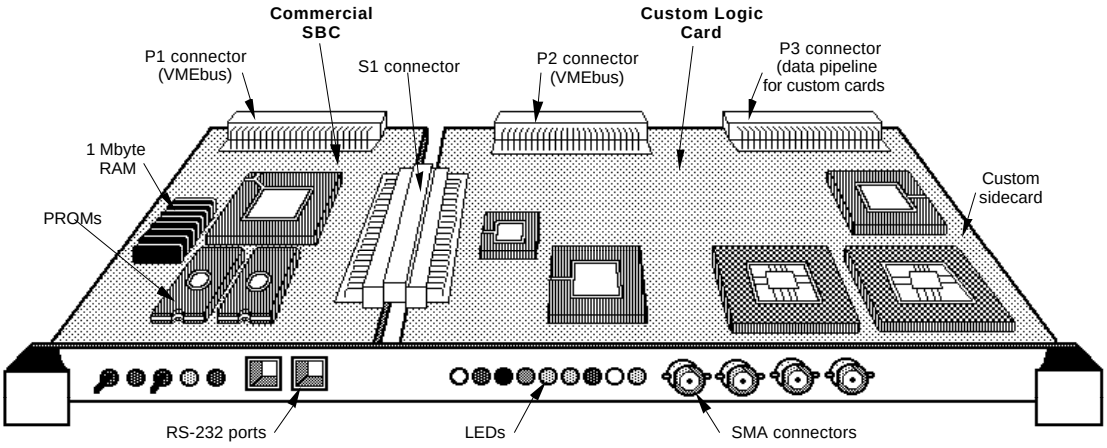


Figure 6-2. Typical Custom Card

The VxWorks/MEDS operating system described in Section 7 runs on each of the custom card controllers. VxWorks allows for the complete development of application programs on each custom card. MEDS links these application programs to the commercial cards.

The following sections describe the individual custom cards used as they apply to the LRS-VME system operation.

6.3.1 EOS SIMULATOR CARD

The EOS Simulator Card is a 9U VMEbus A32:D32 master/slave card using a 68040 CPU Mezzanine controller module mounted on the card. The custom logic card provides the specialized hardware functions for data simulation. The controller provides setup, hardware diagnostics, and control functions for the custom logic card. Refer to Section 7 for a description of the EOS Simulator Card software.

Figure 6-3 illustrates the EOS Simulator Card data flow. The functional elements of the custom logic card hardware are briefly defined as follows:

a. 68040 microprocessor mezzanine controller provides control circuitry to address and configure the different elements of the card. The CPU mezzanine controller transfers setup and control information to/from the MCC via the VMEbus, and writes frame pattern data to the 512-Kbyte memory, as well as updated sequence numbers to the update FIFO and Field-programmable Gate Array (FPGA) to support the output of simulated frames streams at rates up to 1.1 Mbps.

b. Dual-banked 4-Mbyte SRAM pattern memory used to hold the Simulated CCSDS Telemetry Generator (SCTGEN) data pattern. It is arranged as two memory banks that allow access from either the Code 521 Gallium Arsenide (GaAs) Test Data Generator controller chip set, update FIFO, or the CPU mezzanine controller. Each bank is implemented as two 256-Kbyte x 32 SRAM SIMM modules. Two stages of crosspoint switches allow bi-directional multiplexing of buses from all three sources. Multiplexing is controlled by the controller or the GaAs Test Data Generator. During memory loading, the controller operates the switches. Once output has started, the GaAs Test Data Generator controls the switches allowing ping-pong memory access between banks to provide updating on one bank while it outputs from the other bank. The RAM pattern memory can be expanded up to 16-Mbytes.

c. Update circuitry, under controller configuration, updates one of the memory banks with information previously loaded by the controller while the GaAs Test Data Generator outputs from the alternate bank. A 64-Kbyte update FIFO is used to decouple the controller from the dual-banked pattern memory during update mode. This decoupling allows the controller to burst out update information into the FIFO and resume other operations without having to wait for the updates to be read out by the update FPGA. The update FPGA reads and decodes the update address from the update FIFO to determine which module and bank is to be accessed. It then determines which memory bank the GaAs Test Data Generator is currently accessing and whether the update can be performed. If it can not be performed, the FPGA waits for the GaAs Test Data Generator to toggle to the other bank. Additional circuitry is included to interrupt the controller in case the GaAs Test Data Generator toggles into the bank currently being updated by the update FPGA. This would indicate that the GaAs Test Data Generator is operating at a faster rate than the updates can be loaded into the update FIFO.

d. Numerically Controlled Oscillator (NCO) provides programmable clock rates for the GaAs Test Data Generator chip set and the output serial-to-parallel conversion subsystem at rates from 1 kHz up to 150 MHz in 1-kHz increments.

e. GaAs Test Data Generator chip set, under controller configuration, alternates output of data from the two SRAM memory banks previously loaded by the controller using the SCTGEN-generated data files. The patterns may be output continuously, or the user may control the number of frames to be transmitted (up to 2^{24} frames) by switching between the two memory banks and repeating a programmable number of times until a specific number of patterns are output. The GaAs Test Data Generator may optionally output data in forward, reverse, true, or inverted formats. The ETS VLS only uses true forward data. The output streams are sent to the encoder section in byte-wide serial order. A built-in parallel-to-serial converter is provided on the chip to allow output as serial Emitter Coupled Logic (ECL) clock and data at rates up to 150 Mbps. Once the GaAs Test Data Generator has started a sequence, it assumes total control over data access and transfers.

f. Encoder Control FPGA provides control over the Reed-Solomon, CRC, and Bit Transition Density (BTD) encoders. It generates the timing signals required for the transfer of data from the GaAs Test Data Generator chip to the encoders. A single control register is programmed to determine the type of encoding to be performed, whether the frame synchronization pattern word is to be attached to the data, the Reed-Solomon interleave depth, and the data frame size to be encoded. In the ETS VLS, only Reed-Solomon encoding and frame synchronization word generation are used. The resulting data stream undergoes parallel-to-serial (clock and data) conversion at rates up to 150 Mbps.

g. EOS Reed-Solomon Card encoder consists of a byte-wide ASIC developed by the University of New Mexico, which implements the CCSDS-recommended (255, 223) code. It receives VCDU data from the GaAs Test Data Generator Chip in a byte serial fashion under the control of the FPGA. The ASIC is capable of supporting interleave depths (I) of one to eight. The Reed-Solomon code is a cyclic symbol error correcting code that is useful in correctly transferring data through a channel characterized by burst errors. The encoder operates on the 892-byte (223 * I = 223 * 4 = 892, for I = 4) long data space, and generates a 128-byte (32 * I = 32 * 4 = 128, for I = 4) long field of parity that is appended to the data to form a CVCDU as recommended for Grade-2 service in the CCSDS AOS Blue Book.

h. CRC-16 encoder consists of a byte-wide PLD implementation of the International Telegraph and Telephone Consultative Committee (CCITT) and CCSDS-recommended CRC-16 code. It receives data from the GaAs Test Data Generator chip in a byte serial fashion under the control of the FPGA. The CRC code provides the capability of detecting errors that may have been introduced into transmitted frames. The encoder operates on the data space to generate a 2-byte field of CRC code parity that is appended to the data as a frame data trailer field as recommended for Grade-3 service, or optionally for Grade-2 service in the CCSDS AOS Blue Book. The encoder is initialized to an all-ones condition between successive frames during the time allocated to frame synchronization mark insertion. The ETS VLS does not use the CRC-16 encoder.

i. The BTD encoder consists of PLDs implementing the CCSDS-recommended pseudo random sequence. It receives data from the error encoders in a byte serial fashion under the control of the FPGA. The pseudo random sequence is bit-wise exclusive-OR'ed with the data received in order to guarantee a minimum density of bit transitions in the data before it is modulated onto the physical space link channel. The encoder operates on the complete data space as recommended in the CCSDS AOS Blue

Book, but not on the sync marker. The encoder is initialized to an all-ones condition between successive frames during the time allocated to frame synchronization mark insertion. The ETS VLS does not use the BTD encoder.

j. Test FIFO is used to provide card self-test ability. This 16-Kbyte x 8 FIFO receives serial VCDUs from the error encoders under the control of the FPGA. During each session, the test FIFO is written to by the FPGA and GaAs Test Data Generator chip until full. The FIFO has a byte-wide interface to the controller for data verification.

k. The HRTB interface is used to synchronize the byte-wide data output from the encoders, and transmit it synchronously across the HRTB. This section is not used in ETS VLS.

l. The front-panel Subminiature Assembly (SMA) output interface includes a parallel-to-serial converter that serializes and formats data from the encoder section as a differential 100K ECL output made available to the external world through the front-panel SMA connectors.

m. The generic plug-in mezzanine module input interface is a generic interface that allows use of interface plug-in modules for access from external devices. Currently defined interfaces are: High-speed Interface (HSI) and SCSI. New interfaces can be accommodated through the creation of new plug-in modules.

n. The Light-emitting Diode (LED) control and display logic is used to provide a visible representation of the status of the card.

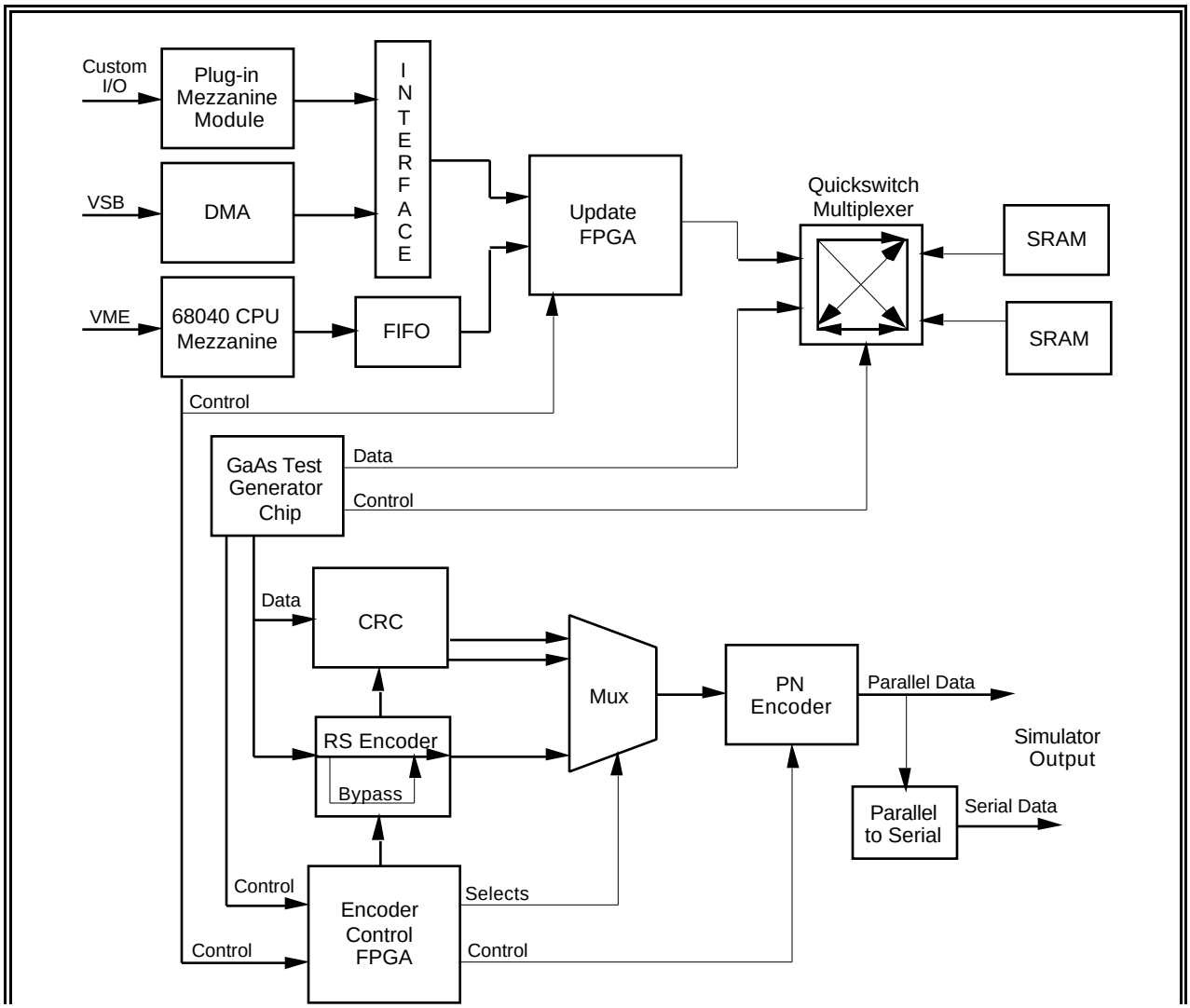


Figure 6-3. EOS Simulator Card Data Flow

6.3.2 FRONT-END PROCESSOR CARD

Figure 6-4 illustrates top-level data flow through the FEP Card. Each functional element is diagrammed; output processing applies to 9U versions of the card only. A detailed explanation of each element is provided in the following sections.

6.3.2.1 CPU/VMEbus/VSB

FEP Card setup, control, and status gathering is performed by the 68030 CPU Mezzanine. This mezzanine consists of a Motorola 68030 microprocessor operating at 33 MHz, 500 Kbytes of flash RAM for nonvolatile VxWorks boot image storage, and 2 Mbytes of SRAM for operating system and program memory.

The MC680x0 support chip supplies chip select and acknowledge generation for the motherboard, interrupt encoding, and UART operation for the terminal interface. The chip selects for all address spaces available as VME slave devices must be configured for asynchronous operation. This allows for the generation of DSACK signals back to the VME interface chip.

For complete details of the 68030 CPU Mezzanine, refer to the 68030 CPU Mezzanine, Revision C, Hardware Definition Document.

The VMEbus interface is implemented using the Newbridge Microsystems SCV64 VME chip. The SCV64 supports master and slave operations for VMEbus sizing up to A64/D64. VME transfers using A64/D64 and A32/D64 modes require the use of SCV64 block transfer mode. Block transfer mode is supported with the SCV64's internal DMA controller. During FEP Card operation, the DMA controller transfers data between the 32-bit wide data capture channels and a memory buffer located on the VMEbus. The SCV64 only supports DMA transfers between a local address space and a VMEbus address space.

For optimal system performance, the SCV64 should be configured to operate in decoupled mode. This mode allows accesses between the VME and local buses to be decoupled, which prevents slower VME transfers from occupying the local bus for an extended time. Further information pertaining to the SCV64 VME chip is available in the Newbridge Microsystems SCV64 User's Manual.

The VSB interface is implemented using a set of three Programmable Logic Arrays (PLA) to support limited VSB operation. This implementation only supports 32-bit longword-aligned master transfers in block write mode. Only serial request arbitration is supported; the FEP Card may be jumper-selected as the arbiter. Block transfers are supported for block sizes from 1 to 256 longwords. Any VSB device that supports the above transfer mode may be a destination; the transfer source must be one of the FEP Card's data capture channels.

The VSB transfer mechanism is implemented with a 10-state state machine clocked at 40 MHz. Maximum block transfer rate is controlled by the wait states required by the slave device. With the minimum of one wait state, the burst block transfer rate is 32 Mbytes/sec.

6.3.2.2 Input Processing

The input processing functional element accepts data from one of the external interfaces and formats it for further processing by either the Nascom block processing subsystem or the frame synchronization subsystem.

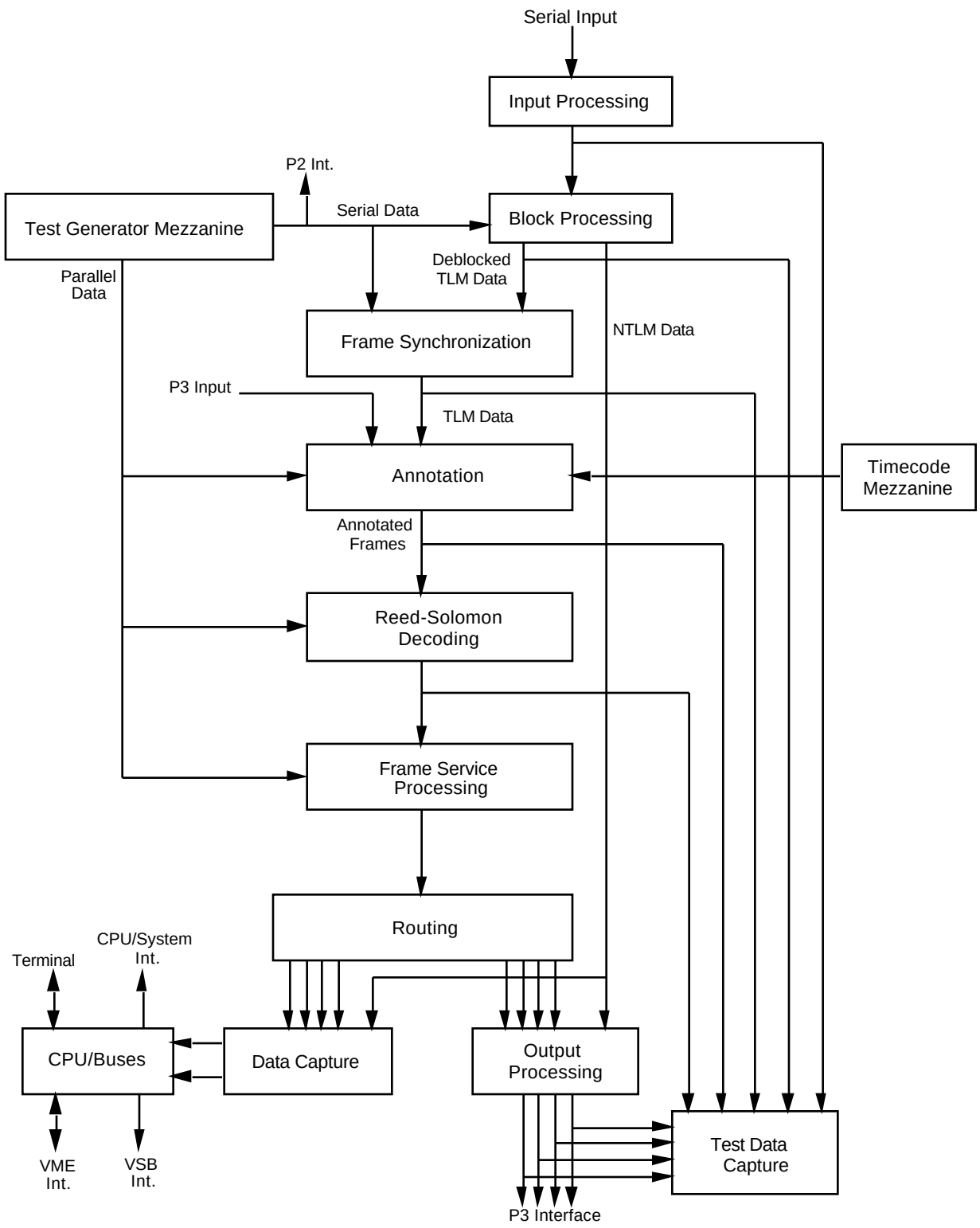


Figure 6-4. FEP Card Data Flow Diagram

Eight independent inputs are available for input processing. For each processing session, an input configuration must be selected and maintained for the entire session. Two RS-422 input interfaces are available on both the front-panel and P2 connectors. Each differential input pair is processed by the RS-422 receivers prior to forwarding to the input multiplexer. Two Transistor-Transistor Logic (TTL) input ports are available from the P3 connector. These inputs are fed directly to the input multiplexer for selection. A parallel input is available for processing data from the VMEbus. This processing mode allows the local CPU to read data from global memory on the VMEbus, and write it to an 8-bit wide First-in, First-out (FIFO) for processing. Once the desired data is written to the parallel input buffer, data flow is initiated by writing to the Start Par2Ser register. The last input port is serial data from the Test Generator Mezzanine. This input supplies TTL input data and clock directly to the input multiplexer.

The input multiplexer uses bits 0, 1, and 2 from the input control and status register to select the appropriate input source. All eight input sources are processed identically after selection at the input multiplexer. Optional clock inversion is available after the input multiplexer. This processing compensates for possible skew between the input clock and its data. Inversion of the input clock can adjust the hold and setup times between the data and the rising clock edge prior to input to Nonreturn to Zero (NRZ) processing. NRZ processing allows decoding of data formatted as NRZ on Level Conditions (NRZ-L), NRZ on Space Conditions (NRZ-S), or NRZ on Mark Conditions (NRZ-M). The NRZ decoding scheme is detailed in Table 6-1. A 1-bit time delay is incurred through the NRZ decoding process. Decoding modes are selected by using bits 4 and 5 of the input control and status register.

Table 6-1. NRZ Decoding Scheme

Input(x)	Input(x-1)	NRZ-L	NRZ-S	NRZ-M
0	0	0	1	0
0	1	0	0	1
1	0	1	0	1
1	1	1	1	0

Data output from input processing can be sent to the Nascom block processing subsystem or frame synchronization processing subsystem.

6.3.2.3 Frame Synchronization Processing

Frame synchronization is performed on data frames received either from one of the external data inputs or from Nascom block processing. Frame synchronization can be performed on true or inverted and forward or reverse frames. A frame is a fixed-length data unit with a synchronization pattern that signals the start of frame. Following the synchronization pattern, the frame consists of a header, data field, and trailer. A typical frame processed by the FEP Card conforms to CCSDS recommendations, as specified in either:

a. CCSDS Packet Telemetry, CCSDS 102.0-B-3, Blue Book.

b. CCSDS Advanced Orbiting Systems, Networks and Data Links, CCSDS 701.0-B-2, Blue Book.

Each frame consists of a frame synchronization pattern up to four octets in length and a variable VCDU.

6.3.2.4 Annotation Processing

Annotation processing attaches frame quality and status information to the header of each frame prior to shipment to Reed-Solomon decoding. Quality annotation is derived from frame quality checks performed by frame synchronization; for frames received in Nascom blocks, block quality information is derived from Nascom processing. Status annotation includes fields for annotation format, frame length, user signature, frame GVCID, frame offset, time, and fill data.

The annotation processor supplies annotation in a fixed order attached to each frame as a header. Fields to be included in the annotation are selectable during card setup to allow customization of the annotation. Annotation of Reed-Solomon quality information can be attached by the Reed-Solomon decoder; for Reed-Solomon encoded frames, the GVCID annotation must be attached by the Reed-Solomon decoder. Modification to annotation is also performed on frames receiving frame-level servicing. The format and user signature annotation fields are statistically configured at setup; if annotated, they will be the same for each output frame.

6.3.2.5 Error Detection and Correction

The FEP Card provides CCSDS grades of service 2 and 3 error detection and correction (refer to the CCSDS documents listed in Section 1.4 for a detailed explanation of grades of service). Grade 2 service provides error detection and correction for the entire VCDU. Grade 3 service provides VCDU header error detection and correction, and VCDU error detection.

Error detection services are intended to inform the user of data errors without the ability to quantify the extent of the corruption, and without attempting to correct data that is in error. Error correction services detect data errors, quantify the errors as correctable or uncorrectable, and restore correctable data units to an errorless condition.

6.3.2.6 Frame Service Processor

All data output from the Reed-Solomon Decoder is processed by the frame service processor subsystem. Each service is performed on every frame received by the frame service processor. The services performed are VCDU (identical functionality as SLC), VCA, insert, and no service. Output data from no service is identical to the input data. The processing algorithm for all other services is identical; the difference in functionality is implemented by programming the configuration registers. Output from each service consists of the header and data unit.

Three sets of three registers are available for programming of the VCDU, VCA, and insert services. In each set, registers are used to specify the byte counts associated with end-of-header, beginning-of-data unit, and end-of-data unit. Processing is initiated by the receipt of a beginning-of-frame mark, and is terminated with the receipt of an end-of-frame mark from the Reed-Solomon Decoder.

6.3.2.7 Routing

Routing functions are executed on each frame output. The routing hardware processes data between five input ports and four output ports. Valid routing configurations do not allow multiple frame service processor outputs to be routed to the same routing output for a single input frame. VCDU frame service processor output is input to the routing hardware in two ports to support VCDU and SLC services.

Each insert, SLC, and no service frame input to the subsystem is processed identically based on the insert, SLC, and no service routing registers. These registers must be configured prior to the initiation of data processing; all frames for a given service will be processed identically. Routing of VCDU and VCA services is GVCID dependent. The Reed-Solomon Decoder uses the GVCID received with each frame as an offset into a routing lookup table, and outputs to the routing hardware the 7-bit value from the lookup table with each byte of the frame.

6.3.2.8 Data Capture Subsystem

The data capture subsystem allows for the capture of data from various points in FEP Card data flow. This subsystem consists of data capture and test capture logic.

6.3.3 EOS SERVICE PROCESSOR CARD

The EOS Service Processor Card is a 9U VMEbus A32:D32 master/slave card using four mezzanines, three 68040 CPU mezzanine controllers, and a telemetry data pipeline mezzanine that are mounted on the card. The three 68040 mezzanines are designated as the header processor, quality processor, and output processor. These processors are configured to be independently programmed and operated. The fourth mezzanine, called the pipeline mezzanine, stores frames and assembles packets. The main board provides interfaces for the VMEbus, the VSB, the backplane, and the three 68040 mezzanines. Each of the three processors is linked to the others by DPR, and has a separate bus and executes independently. Each processor contains an operating system onboard.

In conjunction with the four mezzanines, the custom logic card provides the specialized hardware functions for service processing, including packet extraction from the incoming frames. The quality processor mezzanine acts as a card controller and provides setup, hardware diagnostics, and control functions for the custom logic card. Refer to Section 7 for a description of the EOS Service Processor Card software.

Figure 6-5 illustrates the EOS Service Processor Card hardware. The functional elements of the custom logic card hardware are briefly defined as follows:

a. The pipeline input interface receives parallel frame data from the FEP Card over the backplane or low-rate test frame data from the header processor at a maximum rate of 50 Mbps and peak packet rate of 15,000 packets per second. Data is received as byte-wide data with a 9th bit EOF marker signal. A 4-Kbyte x 9 input FIFO is used to buffer the data received and smooth any bursts in the data received.

b. The pipeline mezzanine contains two modules, controlled by the header and output processors, that manipulate the telemetry data: the reassembly RAM controller module and the tribuffer module. The tribuffer module includes a VLSI tribuffer controller and a triple buffer (4 Kbytes/buffer) for the temporary storage of frame data entering the card. It simultaneously inputs a frame from the input FIFO, and provides the previously stored frame data to the header processor, which outputs the frame previous to that to the reassembly RAM controller module. It also detects certain processor overload conditions. The reassembly RAM module consists of a 1-Mbyte RAM buffer and a RAM controller ASIC. It receives frames from the tribuffer module and stores them by VCID in one of eight 128-Kbyte virtual ring buffers in the RAM memory buffer. The RAM controller provides 8- to 16-bit transfer conversion and address counters for each of the eight partitions. Using commands provided by the output processor through a reassembly instruction FIFO, it outputs reassembled packets with optional annotation from the RAM buffer to the backplane, the loopback FIFO, or the VSB interface.

c. VSB interface provides a master DMA interface circuit to the VSB for the output processor, including the reassembly RAM data output.

d. The loopback FIFO temporarily holds packet data from the backplane output for system self-test by the output processor.

e. The pipeline output interface receives parallel packet data from the RAM controller module for output over the backplane. Data is received as byte-wide data with a 9th bit EOF marker signal.

f. The VMEbus interface provides a master and slave setup and command interface to the VMEbus for the quality processor. The quality processor then transfers the information to the header and output processors. The only memory accessible through the VME interface is the quality processor memory. It can also allow the EOS Service Processor Card to serve as the VMEbus system controller.

g. The quality processor communicates with the VMEbus as a master or a slave. It contains the control logic to set up the header processor and output processor local memories. It processes data flow quality information, and includes a local user terminal port for software development and debugging. It sets up the header and output processors by accessing their local DPR, and can send interrupts to each of them. The quality processor checks the quality of data, generates a packet piece assembly list, and maintains accounting statistics. The quality processor receives data from the header processor. The quality processor derives assembly information from the data it receives from the header processor, and passes it to the output processor.

h. The header processor controls the tribuffer RAM module. The header processor reads headers from the tribuffer RAM subsystem and passes them to the output and quality processors. It can communicate with the quality and output processors through interrupts or DPR. For testing purposes, it can write data into the input FIFO and read the results from the loopback FIFO. It includes a local user terminal port for software development and debugging. It is the only processor in the card with access to the input data, and provides data dependent information to the quality (header-derived information about frames and packet pieces) and output (location information for packet pieces) processors.

i. The output processor controls the reassembly RAM module. It receives metadata from the header processor and data commands from the quality processor. It uses this information to create instructions for the reassembly RAM to reconstruct packets and annotation for output to the backplane. It can communicate with the quality and output processors through interrupts or DPR. It includes a local user terminal port for software development and debugging. The output processor receives a list of packet piece locations in each frame from the header processor via the DPR. Based on this list, it maintains a frame and piece directory. Using this information, it generates instructions for data output to the reassembly RAM module.

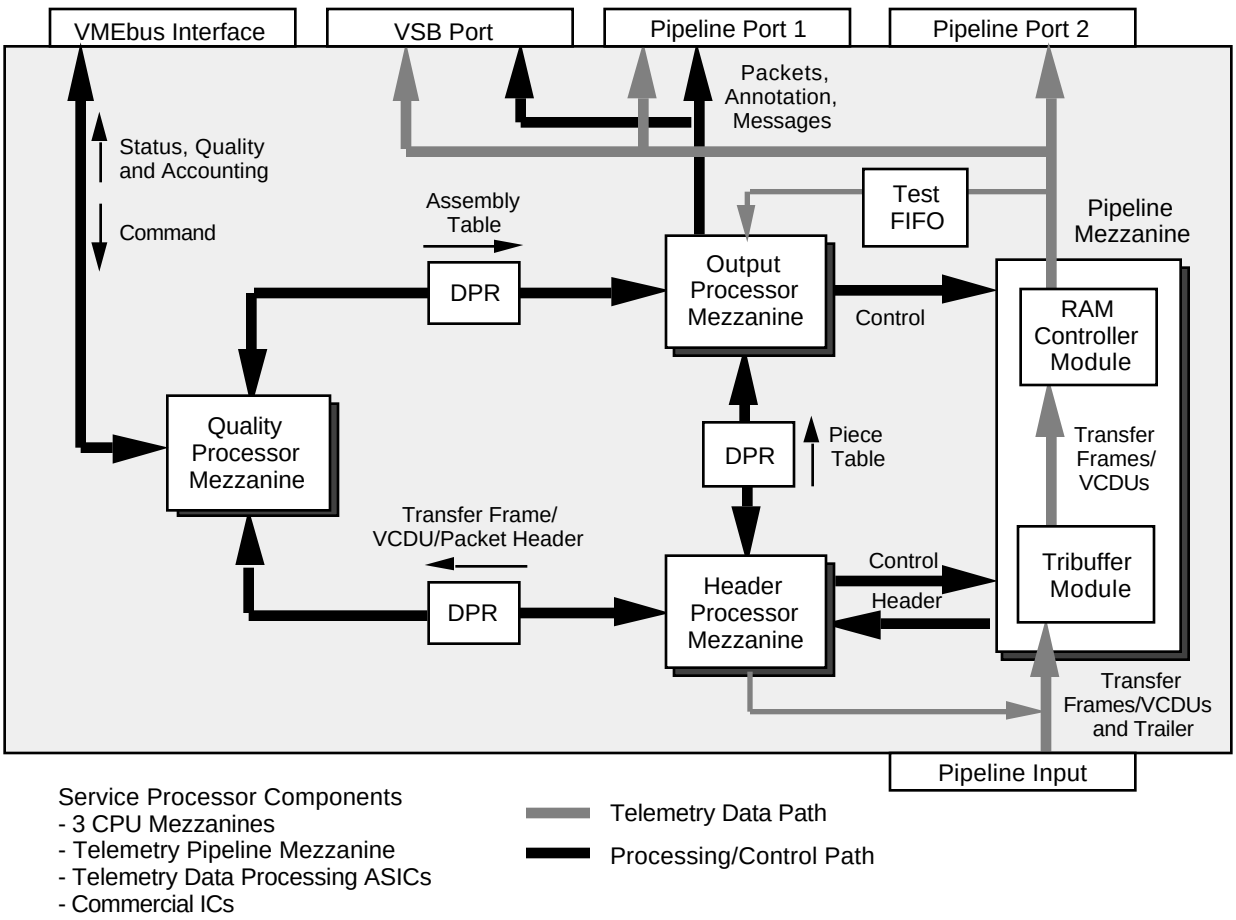


Figure 6-5. EOS Service Processor Card Data Flow

6.3.4 FORWARD LINK INTERFACE CARD

The Forward Link Interface Card is a 9U VME multi-function interface card that provides three separate independent types of telemetry interfaces and additionally contains a NASA36 timecode serial input interface and a parallel J3 pipeline timecode output interface. Refer to Figure 6-6. Telemetry data that is to be output is transferred over the

VMEbus to the appropriate interface where it may be encoded, converted and output as a synchronous serial stream. The timecode interface receives and decodes NASA36 AM timecode. This timecode is available for card- or system-use, or can be used to load a J3 pipeline timecode output interface.

The three independent telemetry interfaces are:

a. CCSDS Telecommand interface. This bi-directional interface receives data from the VMEbus and outputs it as a synchronous serial stream of CLTUs through either a front-panel DB-9 connector or one of two different ports on the P2 connector. One of the P2 output ports has data and clock and the other P2 output port has data, clock, and enable. Only one of these ports can be used at a time. A P2 connector input port receives serial CLTUs, processes, and transfers the CLTUs to the VMEbus. The output interfaces can operate in either encode or throughput mode. In throughput mode, anything that is input to the interface is output without any alteration. Therefore, in throughput mode the data that is to be output must contain both the start and tail sequences. In encode mode, the input data is Bose-Chaudhuri-Hocquenghem (BCH)-encoded and the hardware generates and inserts at the appropriate location both the start sequence and the tail sequence when the data is output to the serial interface. In both modes, a 12-bit programmable length (typically 132-bit) acquisition sequence of alternating ones and zeros is generated by hardware and output before the rest of the data.

b. AOS output interface. This interface outputs a serial stream of CADUs. The CADUs can be output from either an RS-422 DB-9 front-panel connector or the rear panel P2 connector. This interface can operate in either encode or throughput mode.

c. Nascom output interface. This interface accepts skeleton Nascom blocks, calculates and overlays the correct CRC onto the block, and outputs the data as a synchronous serial bit stream through either a front-panel DB-9 connector or through a P2 connector port. This interface can only operate in encode mode.

Each output interface has separate NCO that can be used as an internal clock source or can accept an external clock source to be used to generate the output clock.

The Forward Link Interface Card can receive and buffer CLTU echo command downlinks from the spacecraft to confirm uplink commands.

The Forward Link Interface Card contains a timecode interface, which accepts a 1-kHz AM NASA 36-bit timecode from a front-panel BNC connector or the P2 connector through the jumper. The specialized hardware on the Forward Link Interface Card decodes this signal to generate a timecode in Binary Coded Decimal (BCD) and PB1 formats that is available to the FLCC. PB1 format is also available to other cards in the system through the P3 telemetry pipeline.

Functionally, the Forward Link Interface Card accepts data from the VME interface. The FLCC determines whether the data is an AOS, telecommand, or Nascom block data structure, and transfers the block to the correct telemetry interface where the necessary encoding is performed.

Basic functions of the Forward Link Interface Card are as follows:

a. Accept valid CADUs from the VME interface and throughput to the AOS output interface for serial RS-422 transmission.

b. Accept valid CLTUs from the VME interface and throughput to the telecommand output interface for serial RS-422 transmission.

c. Accept raw telecommand data from the VME interface for encoding to generate CLTUs for serial RS-422 telecommand transmission.

d. Receive and buffer entire CLTU echo commands in a FIFO before outputting data to the FLCC for further processing, such as bit-error correction.

The Forward Link Interface Card is a 9U VME card. One third of the card is a commercially available MZ 8130 Single Board Computer (SBC) that is used as the FLCC. The other two-thirds are comprised of the custom logic card, which performs the specialized hardware functions.

The FLCC uses a 25-MHz Motorola 68030 microprocessor with 1 Mbyte of dual-access DRAM, and is a dedicated processor that provides setup, self-test, and control over the custom logic card. Extensive software to control the card and to provide complete setup and status reporting via the MCC is executed on the FLCC.

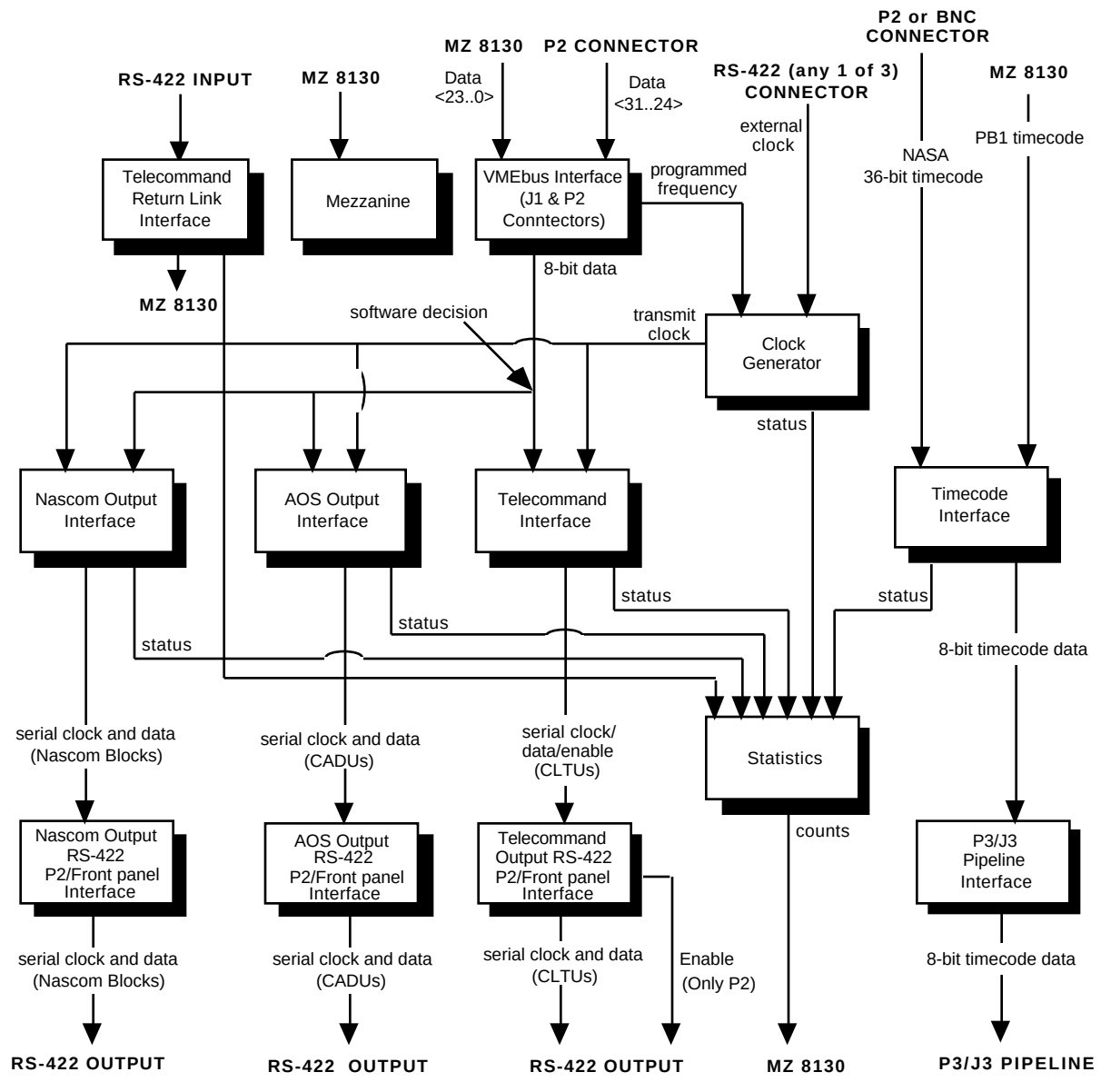


Figure 6-6. Forward Link Interface Card Functional Block Diagram

The custom logic card is connected to the MZ 8130/FLCC via an MX Bus (S1 connector), and contains the specialized hardware functions necessary to receive AOS, telecommand, and Nascom data from the VME interface, and process it to provide valid serial data streams for AOS, telecommand, or Nascom transmission. The custom logic is comprised of the following elements:

a. Microprocessor Decode and Control Logic (MDCL).

b. NCO clock generator.

c. NASA36 timecode interface.

d. Data quality chip.

e. AOS output interface.

f. Telecommand output interface.

g. Nascom output interface.

h. Telecommand return link interface.

6.3.4.1 Microprocessor Decode And Control Logic

The MDCL subsystem performs three main functions. First, it provides the interface of the custom logic card subsystems to the FLCC (through the S1 connector). The Microprocessor Decode Logic (see Figure 6-7) provides 24 chip select outputs, independent Read/Write (R/W) signals, and asynchronous acknowledge signals that are used by the FLCC to terminate the current bus cycle. Upper address lines from the FLCC are input to a programmable 16L8 and decoded, generating enable signals for other 16L8s that provide division of memory space into 24 equal segments.

Second, the MDCL buffers the address, data, and control signals on the S1 connector. This is necessary because the FLCC does not buffer its S1 signals.

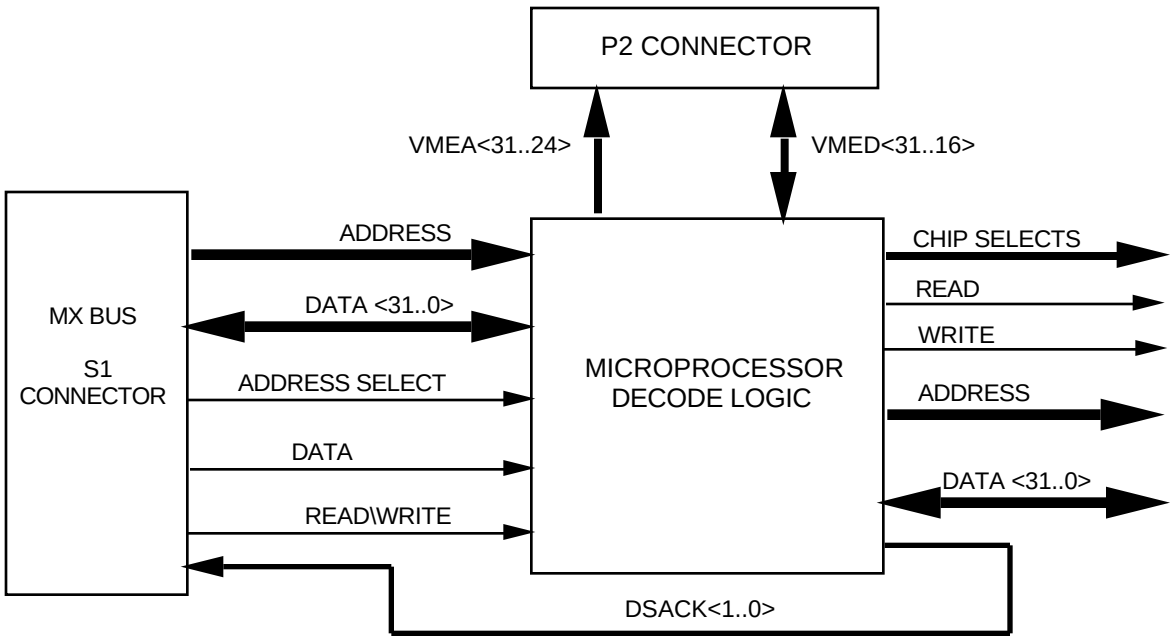


Figure 6-7. Microprocessor Decode Logic

Third, the MDCL contains control and status registers (see Figure 6-8). These latches are written to and read from by the FLCC; they provide hardware latch bits for control over the custom hardware, as well as status information from the output interfaces.

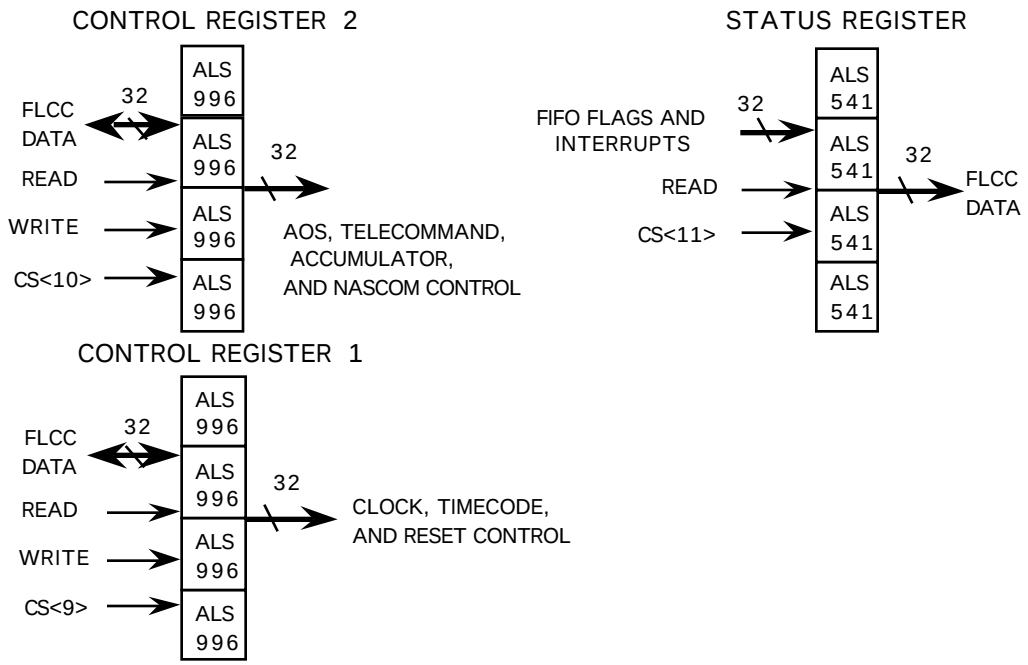


Figure 6-8. Control and Status Logic

6.3.4.2 NCO Clock Generator

The transmit clocks for the AOS, telecommand, and Nascom output interfaces are generated by the clock generator, which is set up by the FLCC. This logic allows for independent programmable clock selection from an NCO to provide variable clock rates of up to 1 MHz for each output interface. Selection of an external clock from either the front panel or the P2 connector is also provided.

6.3.4.3 Timecode Interface

The timecode interface provided on the Forward Link Interface Card accepts the 1-kHz, AM NASA36 timecode signal through either the front-panel BNC connector or rear P2 connector, and creates BCD or PB1 byte-wide data that is available to the FLCC; it also creates a byte-wide continuous PB1 timecode on the P3 telemetry pipeline. The BCD format is accurate to 1 µs and the PB1 format is accurate to 1 ms.

The data quality chip is implemented in an FPGA that contains twelve 16-bit counters. It is used to accumulate statistics on various processes. Refer to the ASIC Components Document, 521-SPEC-002, for detailed information.

The AOS, telecommand, and Nascom output interfaces are three independent forward link output interfaces that convert data from parallel to serial, with the option to encode the data passing through each interface (mandatory for the Nascom output interface) to provide valid serial data streams for AOS, telecommand, and Nascom transmission.

The telecommand return link interface correlates the CLTU start sequence and buffers the CLTU in the FIFO until it detects the tail sequence (or bad code block) and then generates an interrupt to FLCC to indicate that a CLTU is ready for processing. The ninth bit of the FIFO output indicates the last byte of CLTU.

6.3.5 HIGH-RATE TELEMETRY BACKPLANE

The HRTB (Figure 6-9) supports the following functions of the ETS VLS:

a. Telemetry transfer of data between custom cards in the system at rates up to 150 Mbps. It is capable of supporting transfers at rates up to 300 Mbps.

b. Ability to dynamically reconfigure connections to meet the needs of many different system interconnection topologies.

c. Ability to support “upstream” flow control. Receiving card can signal sender to hold transfers until ready.

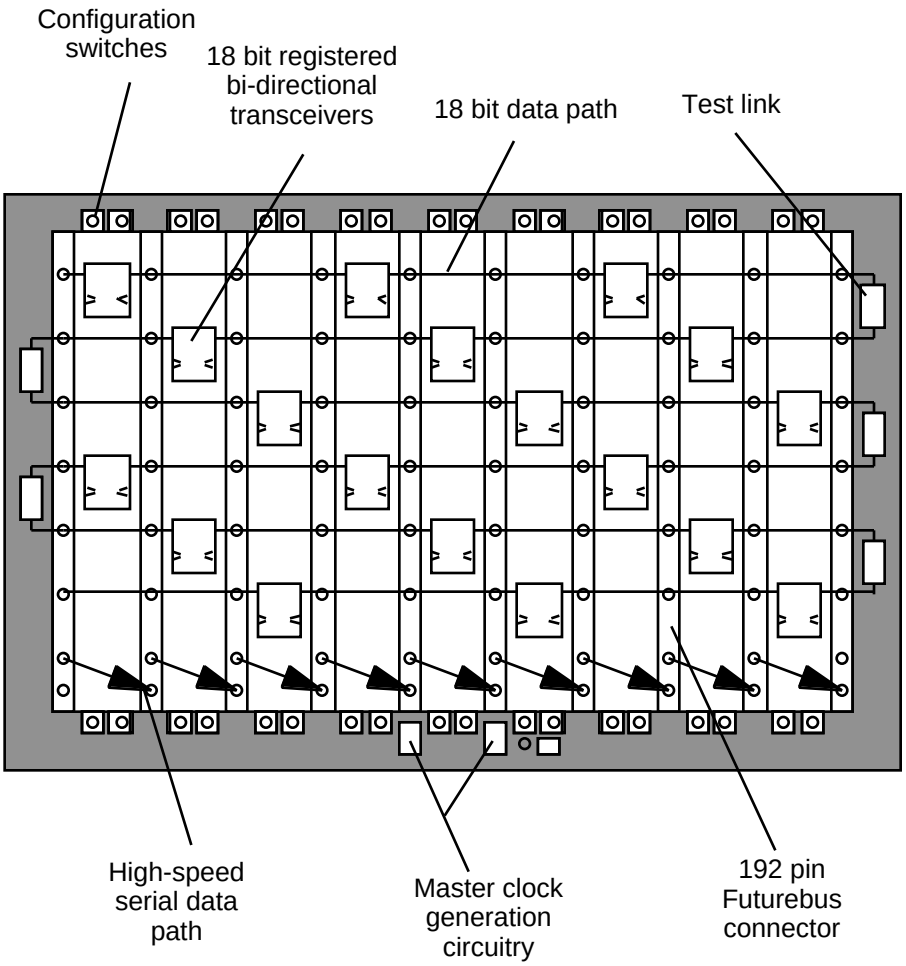


Figure 6-9. HRTB Hardware

The HRTB implements the following interconnection features:

a. Six to twenty data paths per system.

b. A nine-link high-speed serial daisy chain (not used in the ETS VLS).

c. Ten Futurebus-style card connectors.

d Eighteen repeater chips to reduce path physical lengths.

e. Thirty-seven interconnection configuration switches.

f. Onboard three-chip transfer clock generation.

g. Allows input for external transfer clock.

h. Includes ten chip “serpentine” test logic.

i. Supports up to twelve 6-Amp power taps.

The HRTB design may be divided into the following sections:

a. Six parallel data channels; each channel can be dynamically subdivided into up to four segments per channel. This provides from 6 to 20 data paths per system. Configuration of the connections is software-controllable from each of the cards.

A data channel consists of 18 bits per channel: 16 bits of data and 2-bit encoding of an EOF bit and a data valid bit to handle odd-byte length transfer. Each channel can support a 20+ MHz transfer and clocking rate. When transferring 16-bit words, this amounts to 300+ Mbps. Data transfers are performed synchronously using a “backplane” clock. A 20-MHz onboard oscillator is provided, but an external clock may also be used. The clock is distributed using two clock distribution Integrated Circuits (IC). One net is routed to each card, and one net is routed to each pair of transceivers. Each line is independently terminated using a resistor/capacitor termination.

The interconnection topology of the six parallel channels may be configured using the 37 configuration switches on the backplane. These switches provide a “default” configuration during system power up. All switches are readable by all cards connected to the HRTB. If a connected card has the appropriate driver hardware, its software may override its switch configuration dynamically. There are 18 repeater chips (FCT162501) spaced at a maximum of three slots to reduce path physical lengths. Each of these chips has two surface-mount control switches visible and changeable from the front of the backplane.

Each card connected to the HRTB has a 4-bit slot ID value (1 -10) that tells each card its position in the HRTB.

b. Nine-link high-speed serial daisy chain allows a differential serial input from the “upstream” card and an output to the “downstream” card. This function is not used in the ETS VLS.

c. Power pins. There are four VCC and 12 Ground (GND) power connector pins allocated. Each pin can handle 1 Amp. This is in addition to the VMEbus J1 connector that has three VCCs and eight GNDs, and the VMEbus J2 that has three VCCs and 13 GNDs. There are also four groups of “other” voltage pin. Each group consists of six pins (6 Amps). Each of the cards is connected to two groups: from left to right in the chassis, cards 1-5 are connected to pin groups one and two, and cards 6-10 are connected to pin groups three and four. The four groups can be mixed to provide one to four other voltages. Two of these groups are used in the ETS VLS to carry 100-Kbyte ECL power supplies (-2 V, -5 V) to the EOS Simulator Card, the FEP Card, and the EOS Service Processor Card. Unused groups may be redefined for other uses.

SECTION 7
DETAILED SOFTWARE DESIGN

The following section provides an overview of the ETS VLS application-specific software. The application-specific code is developed in a workstation environment using the VxWorks and MEDS platform.

7.1 SOFTWARE ENVIRONMENT

The variety of commercial and custom hardware used in the ETS VLS requires a very versatile software environment. Software environments include VxWorks, MEDS, and TPCE. They are used in the ETS VLS to accommodate a variety of changing hardware; they form the software platform on which all system software is built. VxWorks provides a real-time development environment; MEDS controls and monitors the ETS VLS and the movement of data throughout the system.

7.1.1 VXWORKS

The ETS VLS uses Wind River System's VxWorks that implements a high-performance real-time kernel. Software development is based on Unix workstations using Wind River, GNU and Green Hills' development tools. VxWorks provides extensive Unix-compatible networking utilities that allow seamless integration of VxWorks and Unix.

VxWorks is composed of three integrated components: a set of powerful cross-development tools that run on a host development system; a high performance scaleable real-time operating system that executes on a target processor; and a wide variety of communication options to connect the target and the host. Across all of these components, VxWorks adheres to industry standards including POSIX 1003.1b real-time Extensions, ANSI C, and TCP/IP networking.

The host tools consist of the Green Hills multisystem and standard GNU tools. The Green Hills Multisystem provides a complete cross-compiler and debug environment in a single tool. In addition, the standard host development tools included with VxWorks consist of a GNU cross-compiler toolkit and the VxGDB source debugger.

Target tools include extensive system diagnostics and performance monitoring, application prototyping facilities, comprehensive networking facilities, POSIX 1003.1b compliance, and dynamic linking and loading for rapid edit-test-debug cycles or dynamic run-time flexibility.

Also, the use of additional operating system accessories and productivity-enhancing tools can result in dramatically shorter time-to-market for embedded applications. In addition to the supported targets, VxWorks provides easy porting to custom hardware. Unique to VxWorks is a powerful interactive shell interface that allows users to interact with all VxWorks facilities. Unlike other "shells," the VxWorks shell provides one simple powerful capability: it can interpret and execute almost all C-language expressions, including calls to functions and references to variables whose names are found in the system symbol table.

7.1.2 TELEMETRY PROCESSING CONTROL ENVIRONMENT

The ETS VLS CDS provides mechanisms for controlling and monitoring system. The CDS automates operation of the ETS VLS based upon a schedule of system telemetry processing activities and user-initiated batch processing sessions. The operational schedule is a week-long schedule that identifies spacecraft pass events, including the ground station for downlink, the data rate for the downlink, and the start and stop times of each event. The CDS provides a display of this activity schedule to the user and allows the user to add and delete session events and initiate telemetry processing of data contained on tapes received from the ground stations.

The CDS is designed as a distributed, multi-process system; a number of simultaneously-executing processes cooperate to control and monitor operation of the ETS VLS. The subsystem may operate in a multi-host mode, wherein a specific process may be allocated to one of a number of host workstations, or on a single host workstation.

Processes are classified into two categories based on the temporal nature of their operations: static or transient. Static processes form the core of the CDS and are required to execute continuously, throughout the operation of the system. Termination of any static process is considered a failure of the CDS. Transient processes are initiated and terminated upon user request. For example, most processes that drive status and monitoring displays are transient.

The internal structure of almost every CDS process is virtually the same: a series of C++ class objects are initialized by the process and are then invoked through system event function callbacks. When some type of system event occurs ¾ for example, a user input, clock timeout, or file input ¾ an event is generated, causing a specific, pre-defined member function of some class to be called. This event-based callback structure is supported generically by a set of C++ classes that provide callback capabilities for specific types of system events. A number of examples include: WrEvent, which provides callbacks based upon user input to CDS displays; Alarm Clock, which provides callbacks based upon wall clock timers; Inter-process Communication (IPC) Server, which provides IPC callbacks; and Data Server, which provides status and monitoring data to registering objects on a regular basis through callbacks.

The CDS automatically sets up the ETS VLS for telemetry processing based on scheduled processing sessions of captured telemetry. The CDS provides displays that present the current status of the ETS VLS as it is processing. These displays present frame synchronization, and packet reassembly accounting and data quality information for on-going telemetry processing sessions, present summary data quality and accounting information for processed sessions, and present the status of system hardware and software subsystems. The CDS also automatically generates reports summarizing the data distribution to user sites, and quality and accounting information for completed telemetry processing sessions.

A graphic, Unix-based, computer workstation is configured to support control, status monitoring, and data distribution operations of the ETS VLS. Operating system processes on this workstation use the IPC facilities of Unix, including sockets, pipes, files, and shared memory.

Inter-workstation communications are supported via TCP/IP, providing reliable in-sequence delivery of data required for host-to-host communication. The standard File Transfer Protocol (FTP) is utilized for distribution of telemetry data sets and CODAs to EOC. FTP provides reliable, guaranteed-complete delivery of data files from host to host.

All CDS displays utilize the X Window system and the OSF Motif tool kit.

7.1.3 MEDS

MEDS manages the mix of subsystems (cards) that are part of the ETS VLS. MEDS is an integral part of the system design, development, and implementation because it was specifically designed to manage pipelines, multiprocessors, and a dual-bused hardware architecture.

MEDS provides the ETS VLS software platform upon which application-specific code is developed. Figure 7-1 reiterates the MEDS/VxWorks layers, and designates tasks. MEDS supports the basic software functions that are required in all ETS VLS subsystems: hardware and software processing, setup, control, and monitoring. It provides an infrastructure to pass data between systems, processors, and tasks, and it supports operator interface development. The following paragraphs provide a brief overview of MEDS.

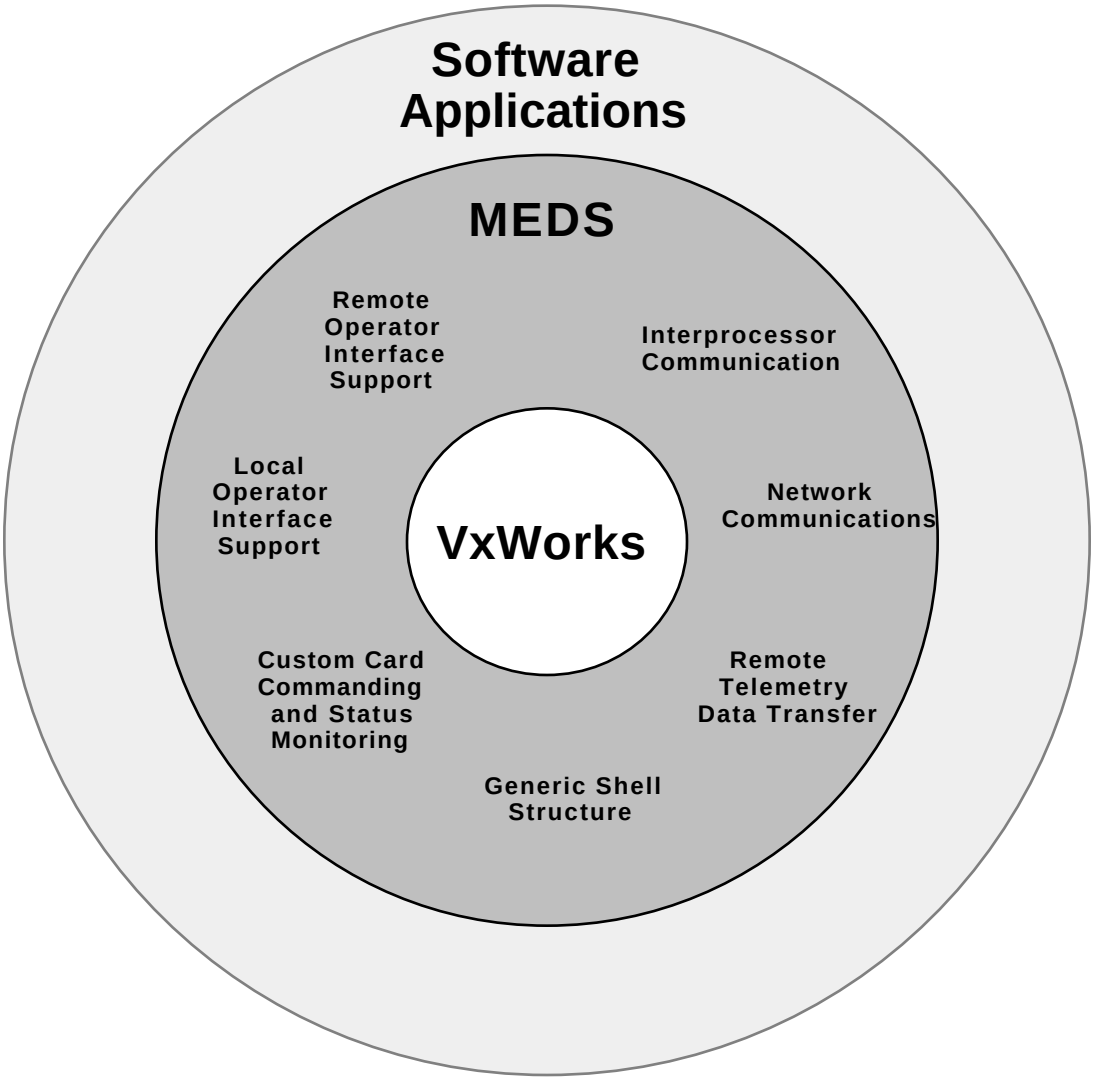


Figure 7-1. VxWorks and MEDS Architecture

The overall MEDS software design is modularized into packages that supply general purpose system functions such as operator interfaces, status gathering, command handling, interprocessor communications, and network communications. Each package implements a set of functions that serves as a resource to the application software, while hiding the details of its implementation. Consistent interfaces have been defined for each package, so code within a package can be changed without affecting the application code as long as the functions and interfaces remain constant.

Some MEDS packages exist as linkable libraries that the programmer uses to build a task, such as a command handler. Other packages exist as customizable source code files, where the applications programmer copies the source file, adds his application-specific code, and compiles, for example, a status task. In all cases, the programmer does not need to know the details of MEDS implementation; only the interfaces to it. MEDS software is primarily written in C; assembly language is used for interrupt handlers.

A MEDS-based system is built by adding custom code to the general purpose MEDS code, which spares application developers the burden of creating an infrastructure for each new system. It also adds consistency to the system design, implementation, and maintenance.

The ETS VLS software is a group of cooperating task built on MEDS software packages. The tasks are spread across the processors of the system.

7.2 MCC SOFTWARE

The MCC software supports the following functions:

a. Maintenance, formatting, and transfer of telemetry processing and system status information to the user workstation via Ethernet.

b. Translation of user commands for the ETS VLS into lower-level card commands.

c. Transfer of error tag records to the user workstation for the error tag record mode of operation.

d. Transfer of buffers to the user workstation for the erroneous frame capture mode of operation.

The MCC software consists of five tasks: MCBOOT, MCLOADER, MCRCMD, MCRSTS, and OPMAN.

a. The MCBOOT task carries out the system boot sequence during system reset. This process is executed once and then terminated.

b. The MCLOADER task monitors the health of all application cards and responds to a single-card reboot.

c. The MCRCMD task receives commands from the user workstation and then translates them into lower-level card commands.

d. The MCRSTS task gathers application cards status information and sends it to the user workstation.

e. The OPMAN task is the local operator interface. It allows the user to control and monitor the system using a local terminal.

Figure 7-2 depicts the MCC context diagram.

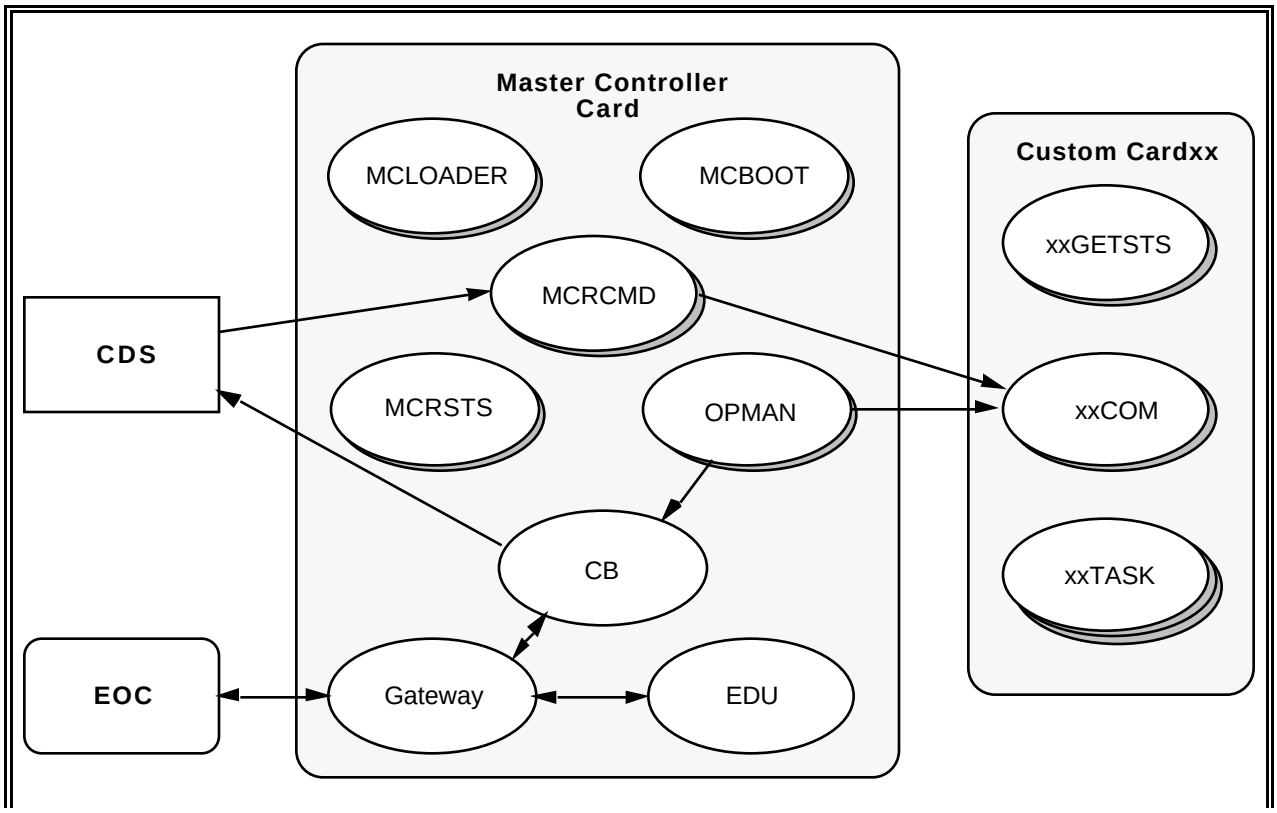


Figure 7-2. MCC Context Diagram

7.2.1 GATEWAY INTERFACE SOFTWARE

The Gateway Module moves data between the ETS VLS and the Ethernet connection to workstations. The physical link between the workstation and the system is Ethernet, and the communication protocol used for data transfer is TCP/IP or UDP/IP.

For TCP/IP, the Remote Interface software consists of two main program modules, the Remote Interface Command handler (RICOM) and the Remote Interface Data Mover (RIMOVER). RICOM is the Remote Interface Module controller; it handles initialization in conjunction with the MCC software. RIMOVER handles the actual transfer of data through the module.

There are multiple copies of RIMOVER; each runs as a separate task. Each RIMOVER task is given two parameters when it is created: the TCP/IP port number that it uses to communicate with the workstation, and the name of the configuration file that specifies what data to route between which Ethernet ports and MEDS mailboxes. RIMOVER reads the configuration file and builds an internal table that tells it how to process incoming and outgoing data. Next, it listens for a connection from the workstation on its given port number, and sets up mailboxes with ETS VLS subsystems.

To process data, RIMOVER checks for data incoming or outgoing through it mailboxes or Ethernet connection. It looks at the table to determine where to route the data, and then it moves it. At the Ethernet and TCP/IP levels, the data is indiscriminately sliced into packet-sized segments and sent across the network for reassembly. An acknowledgment is required from the receiving end to ensure that the data was transferred without error.

For UDP/IP, there is a suite of UDP utilities. The UDP UTIL package implements a high-level interface to UDP/IP network communications. Under UDP, processes send “datagrams” to each other. Unlike TCP, UDP has no concept of a “connection;” messages are individually routed to one or more destination endpoints through a single socket. Also unlike TCP, UDP does not guarantee reliable transmission; messages may be lost or sent out of order.

In the UDP UTIL package, the endpoints of a connection-less UDP “link” are called “endpoints.” A given program may create an anonymous UDP endpoint bound to a system-assigned network port, or it may create a UDP endpoint bound to a predetermined network port (specified by a name or port number).

Client processes generally create anonymous UDP endpoints for the purpose of sending messages to a server process at a predetermined network port. With an anonymous endpoint, a client must let the server(s) know its port number before the client can receive messages from the server(s). The act of sending a datagram to the server(s) automatically supplies the server(s) with the client’s port number and IP address.

By creating a UDP endpoint bound to a predetermined network port, a server is immediately ready to receive datagrams sent by clients to that port; the clients already know the port number, so there is no need for the server to send messages first.

To send a datagram from one endpoint to another, one must specify the network address of the destination endpoint. Since the destination endpoint probably belongs to another process, possibly on a remote host, the UDP_UTIL package requires a user to create a “proxy” endpoint for the destination endpoint. A proxy endpoint simply specifies the network address of the destination endpoint; the proxy endpoint is not bound to a network port and it has no operating system socket associated with it. The proxy endpoint is internally linked with its source endpoint, so, when a datagram is sent to the proxy, udpWrite automatically sends the datagram through the source endpoint to the destination. A source endpoint may have many proxy endpoints, but a given proxy endpoint is only linked to a single source. (If multiple endpoints are desired, a user can create multiple proxy endpoints for the same destination.)

7.2.2 COMMAND BLOCK PROCESSING

The command blocks will be received over the network using UDP/IP as described above in Section 7.2.1. Once a command block has been received, it will be processed. Processing: checks the command block header; sends the

entire command block to CDS for archiving; strips out the command block header and sends the CLTUs to the Forward Link Interface Card mailbox; monitors the input command block data rate; maintains a circular command block queue; generates event messages and statistical information.

The command block header will be checked for the proper message type, source ID, destination ID, version number, message length, and sequence count. If the command block fails the checking process, an error event message will be sent to the CDS indicating the type of failure.

All command blocks will be sent to the CDS for archiving regardless of pass or failure in the command block header checking process. The software used to send the command blocks to the CDS will be the gateway software. If the command block passes the header check, then the command block header will be stripped out and the rest of the data (the CLTU) will be sent to the Forward Link Interface Card through MEDS mailboxes.

If the incoming command block rate exceeds the output rate, the incoming blocks will not be sent. This means the incoming command blocks will be dropped and not sent to the CDS for archiving. Once the output command block has been sent, the incoming blocks will start to be processed again. An event message will be sent to the CDS if any command blocks are dropped.

A circular queue will be maintained in memory containing sent command blocks. Once the queue is filled, the next command block will replace the oldest command block in the queue. Besides error event messages, an event message will be sent to the CDS after each CLTU is sent to the Forward Link Interface Card.

The statistics will supply information about the total number of command blocks received, the number of good command blocks, number of bad command blocks, number of command blocks with bad message type, source ID, destination ID, version number, message length, and sequence count.

7.2.3 EDU PROCESSING

There are two EDU processing tasks: transfer of all types of EDUs to the CDS, and transfer of real-time EDUs to EOC. The transfers to the CDS will be made using TCP/IP. The EDUs are received from the EOS Service Processor Card, which supplies an ESH. The transfers to the EOC are made using UDP/IP. A Ground Management Header (GMH) is added to the real-time EDUs. Status reports indicate the number and total bytes of EDUs sent to the EOC, and the number and total bytes of EDUs sent to the CDS.

7.3 EOS SIMULATOR CARD SOFTWARE

The EOS Simulator Card outputs a single frame data stream. All data manipulation is handled by the card's hardware. The software is responsible for initializing and controlling the hardware, and for keeping track of status counts.

EOS Simulator Card software runs in MEDS. Thus, it must accept and respond to operator commands sent by the MCC, and it must prepare and store status information for possible display.

The EOS Simulator Card software is divided into three tasks: the command handler, status gathering, and the debug task.

a. The command handler task handles card initialization, setup, and self-test. During card setup the command handler is responsible for loading frame data sets generated by Simulated Telemetry Generator (STGEN) into the 4-Mbyte RAM.

b. The status gathering task periodically reads hardware status registers to accumulate status on data flow through the card.

c. The debug task is used for development and maintenance purposes.

7.4 FEP CARD SOFTWARE

The FEP Card reads a single input data stream to find frames. All data manipulation is handled by the card's hardware. The software is responsible for initializing and controlling the hardware, and for keeping track of status counts.

FEP Card software runs in MEDS. Thus, it must accept and respond to operator commands sent by the MCC, and it must prepare and store status information for possible display.

The FEP Card software is divided into three tasks: the command handler, status gathering, and the debug task.

a. The command handler task handles card initialization, setup, and self-test.

b. The status gathering task periodically reads hardware status registers to accumulate status on data flow through the card.

c. The debug task is used for development and maintenance purposes.

Frame processing statistics are gathered throughout the card during operation, and are made available to the CPU interface. These statistics are gathered on both a physical channel basis and a virtual channel basis. Physical channel statistics are calculated based on every frame received by the FEP Card, regardless of the GVCID associated with the frame. Virtual channel statistics are calculated for each GVCID processed by the FEP Card during a session.

7.5 EOS SERVICE PROCESSOR CARD SOFTWARE

The EOS Service Processor Card software runs on three MC68040 microprocessors: header processor, quality processor, and output processor. Although the three microprocessors are linked by DPR to share data with each other, and the quality processor is linked to the VMEbus, each microprocessor has an individual bus, runs independently, and has its own operating system. Each microprocessor also has an RS-232 port that allows it to be connected to a local terminal for debugging and testing.

This card includes two VLSI ASICs: the tribuffer controller and the RAM controller. The header processor controls the tribuffer controller, which manages three RAM buffers in the telemetry data path. The output processor controls the RAM controller, which manages a 1-Mbyte reassembly RAM buffer that holds data pieces.

The EOS Service Processor Card software can be summarized by five top-level processes: command process, status process, header process, quality process, and output process. The command process handles command transfers between the system MCC and the EOS Service Processor Card. The status process handles status transfers between the EOS Service Processor Card and the system MCC. The remaining three processes are telemetry data processes that convert frames into reassembled source packets. An overview of the header, quality, and output processes is provided in the following sections.

7.5.1 HEADER PROCESS

The frames, which the EOS Service Processor Card receives over the J3/P3 custom pipeline, first move into the frame tribuffer (managed by the header processor). The header process has three data streams outputs:

a. First, the header process extracts frame headers, trailers, and packet headers from the frames. Next, it validates the frames and packets, and then sends the processed information to the quality process via the frame and status buffers.

b. Second, the header process generates a piece list summarizing the location of each data piece in the transfer frame, and passes this list to the output process via a buffer.

c. Third, frame data is output to the telemetry data buffer that is controlled by the output process.

7.5.2 QUALITY PROCESS

Once the quality process receives the frame information from the header process, it checks the information against predefined criteria, and then updates status information in the quality and accounting tables in the status buffer.

The quality process also tracks the availability of data pieces in the telemetry data buffer. When all pieces of a packet are obtained, or when some pieces have been rejected, the process sends the output process a packet reassembly table that identifies the packet pieces. DRAM buffers the reassembly table data between the quality and output processes.

7.5.3 OUTPUT PROCESS

The output process is the final stage of reassembling source packets. It maintains a piece directory using the piece list generated by the header process. When a new packet reassembly table (identifying the packet pieces) arrives, the output process begins outputting the packet over the VSB piece by piece.

7.6 FORWARD LINK INTERFACE CARD SOFTWARE

The Forward Link Interface Card software can be summarized by four top-level processes:

a. Setup:
   (1) Initialize custom logic.
   (2) Receive setup information from the MCC.
   (3) Configure custom logic as specified by setup information.
   (4) Begin polling the allocated memory location or mailbox as specified by setup information.

b. Forward Link Data Processing:
   (1) Transfer data from the allocated memory location or from the mailbox to local DRAM.
   (2) Determine whether data is an AOS, telecommand, or Nascom data structure. Perform necessary functions (i.e., pad last octet of telecommand data).
   (3) Transfer data to appropriate FIFO.
   (4) Maintain statistics on the transmitted data.

c. Telecommand Return Link Data Processing:
   (1) Correlate the start sequence of CLTU up to the error tolerance specified by the initial setup.
   (2) Write the CLTU in byte format into the 8-Kbyte x 8 FIFO until it detects the end-of-tail sequence.
   (3) Generate an interrupt to signal the FLCC that a CLTU is ready to be read.

d. Time Decoding and Distribution:
   (1) Interface to NASA standard 36-bit timecode.
   (2) Decode standard timecode to an accuracy of 1 µs for BCD and 1 ms for PB1.
   (3) Provide decoded timecode to Forward Link Interface Card in BCD format (days, hours, minutes, seconds, milliseconds) and in PB1 format (days, milliseconds).
   (4) Provide decoded timecode to other cards in PB1 format through the P3 telemetry pipeline.

7.6.1 CLOCK GENERATOR

Specific clock requirements to be fulfilled by software are designated as follows:

a. Provide ability to enable/disable the output clock for each interface.

b. Provide for independent programmable selection of an internal clock at any rate up to 1 MHz for each output, or selection of an external clock source.

7.6.2 TIMECODE INTERFACE

The only specific timecode requirement to be fulfilled by software is that the card provide status information on the timecode.

7.6.3 TELECOMMAND OUTPUT INTERFACE

Specific requirements to be fulfilled by software are designated as follows:

a. Optionally, to provide an idle sequence after the first CLTU transmission is completed to maintain bit synchronization. The idle sequence pattern can be selected from one of four patterns: 55, AA, FF, 00.

   NOTE: Upon power-up there is no idle sequence.

b. Provide the ability to change the CLTU output data rate between two CLTUs to and from 125 bps, 1 Kbps, 2 Kbps, and 10 Kbps.

c. Maintain the following statistics:

   (1) Number of encoded telecommand codeblocks received.

   (2) Number of unencoded telecommand codeblocks received.

   (3) Number of CLTUs transmitted.

   (4) Time each CLTU was transmitted.

   (5) Logging of transmitted commands.

7.6.4 INTERRUPTS

The Forward Link Interface Card custom logic may generate any of the following interrupts. Each interrupt can be disabled as is the case at power up.

a. The BCD timecode interrupt is generated if the BCD NASA36 timecode counters are not incremented each millisecond, or if they are incremented more than once a millisecond.

b. The telecommand interrupt indicates a fatal error in the processing of telecommand data. This fatal interrupt is generated if the input FIFO to the telecommand output interface becomes empty before a ninth bit is reached (output under flow). The ninth bit is used to indicate the end of a data block.

c. The telecommand synchronization interrupt indicates the start of a new transmission sequence. It is generated when a telecommand start sequence is detected. This interrupt is generated so that the FLCC can maintain statistics that account for the time at which each transmission sequence begins. This interrupt was implemented for spacecraft integration and test.

SECTION 8
LOCAL AREA NETWORK AND CONFIGURATIONS

The ETS VLS communicates via the Ethernet interface. This section explains LAN configuration and includes a brief discussion of the network protocols used.

8.1 LAN CONFIGURATION

In this section, we will discuss all the network connections required to accomplish the ETS VLS functionalities. Each connection is listed and explained in detail, along with who uses it and what is transmitted over the connection.

The system consists of two major Ethernet nodes: CDS and ETS VLS. Outside of the VLS, EOC also communicates with the system via Ethernet. The communication between CDS and ETS VLS employs standard TCP/IP; between ETS VLS and EOC, communication follows standard UDP/IP.

The ETS VLS uses one Ethernet 10base2 (15-pin Attachment User Interface [AUI]) connection with seven IP addresses.

Commands and data have a bi-directional flow on the Ethernet interface between the ETS VLS and the CDS. The MCC manages this communication. Essentially, the software acts as a mailman that passes messages back and forth between the workstation and the ETS VLS. The communication protocol used for data transfer is TCP/IP. The information passed may include commands, status, and telemetry or nontelemetry data.

The CDS and ETS VLS maintain a handshaking relationship; every message transfer requires an acknowledgment that the message has been received without error. If an error occurs, the message is retransmitted.

ETS VLS uses layering, a method that uses protocols at different levels. MEDS, TCP, and IP are each separate layers that access the services of the level below it; the MEDS accesses the TCP library of routines, TCP accesses the IP library of routines, and IP calls the Ethernet routines. Figure 8-1 illustrates the format of data after it is processed by these levels.

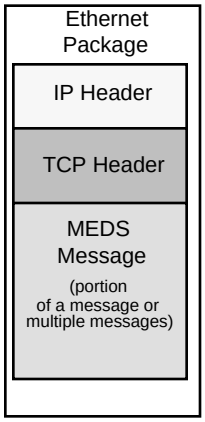


Figure 8-1. Ethernet Application Layers

8.1.1 ETS VLS CONFIGURATION

ETS VLS consists of CDS and VME LRS. A physical Ethernet connection exists between the two system. The VME LRS has five Internet address assigned to the VME rack. Each physical card needs an address to boot itself. Only the MCC has a physical Ethernet connection. All the other cards use the shared memory network (refer to the VxWorks v5.2 Programmer's Guide) scheme. The shared memory network works as the regular Ethernet, except that

it is implemented over VMEbus. All the CPU cards on the shared memory network are assigned an IP address and supports network protocols. The VME LRS contains an MCC, FRP Card, EOS Service Processor Card, EOS Simulator Card, and Forward Link Interface Card.

CDS is a Unix-based system with full TCP/IP support. Its address is known to the VME LRS and the EOC to establish connections as needed. CDS may employ more than one physical Ethernet connection and address, if performance is an issue. A priority scheme is implemented to support this case.

8.1.2 MCC AND CDS CONFIGURATION

This connection is used to transfer and receive commands, status, and event messages generated by CDS and the MCC. This is a primary communication between CDS and the VME LRS. Control of the VME LRS is accomplished through this connection. CDS makes three dedicated socket connections to the MCC at the beginning of a mission. These connections are for MCRCommand, MCRStatus, and MCREvent.

8.1.3 VME LRS AND EOC CONFIGURATION

EOC is not a node within the ETS VLS LAN. However, it is discussed here because of the close ties between the VME LRS and the EOC. A connection between the VME LRS and the EOC accommodates transfer of CLTUs and EDUs. This connection uses UDP/IP protocol.

8.1.3.1 MCC and EOC Configuration

EOC generates a stream of command blocks that is sent to the VME LRS for validation. UDP socket port on VME LRS and EOC handles this transfer. The realtime EDUs generated by he VME LRS are transmitted via the UDP socket. The socket connection exists between the MCC in the VME LRS and EOC.

8.1.3.2 CDS and EOC Configuration

EOC receives rate-buffered data files and CODA from CDS via this connection. The connection supports a full suite of TCP/IP protocol.

8.1.4 EOS SIMULATOR CARD AND CDS CONFIGURATION

This connection provides the way to transfer the test files from CDS disks to the SCSI-2 disk drive controlled by the EOS Simulator Card. A DOS file system installed on the SCSI device is cross-mounted to CDS over Ethernet using TCP/I. CDS mounts the remote SCSI disk at the beginning of a mission and downloads test files by copy.

8.2 COMMUNICATION PROTOCOLS

A protocol provides formulas for passing messages; it includes the message format and rules for error conditions. The ETS VLS uses TCP/IP and UDP/IP to provide a standard method to regulate data transmission between the ETS LRS and the EOC. Although TCP, IP, and UDP are used in conjunction throughout this document, they are three separate protocols that perform separate functions.

TCP ensures the successful transmission of commands to the receiver. It records what is sent, retransmits anything that was not successfully transmitted, and puts data back into the correct order after transmission. UDP is at the same level as TCP. IP is responsible for routing data to the correct destination.

In the following discussion, an octet is defined as eight bits and a datagram is a collection of data sent as a single message during a data transfer. Information in the following discussion was obtained from “Introduction to the Internet Protocols,” produced by Rutgers University; the material copyright belongs to Charles Hedrick.

8.2.1 TCP

TCP is summarized by the following capabilities:

a. Transfers a stream of octets in both directions between the ETS LRS and the EOC by packaging some number of octets into segments for transmission.

b. Handles errors: damaged, lost, or duplicated data, and data delivered out of order.

c. Allows many processes to simultaneously transfer data by assigning addresses (ports) to each process.

d. Maintains status information for each data stream.

TCP breaks the data stream into manageable segments and attaches a header to the front. This constitutes one datagram. In the ETS VLS, the data field of the datagram is either a MEDS message, a portion of a MEDS message, or multiple MEDS messages. The header fields of primary interest (Figure 8-2) are defined as follows:

a. Source and Destination Port numbers identify different conversations. For example, all data coming from the same module is sent to the same port. Each end of the communication assigns a different port number to the conversation; the source and destination port fields identify these port numbers.

b. Sequence Number ensures that data is assembled in the correct order and without missing segments when it arrives at the destination side of the transfer. The sequence is maintained by the number of octets passed—not the number of datagrams passed. Therefore, this number would increment by 500 if each datagram contains 500 octets.

c. Acknowledgment Number ensures that the datagram arrived at the receiving side. When a datagram arrives, the receiving side returns a datagram with an acknowledgment number that reflects the total number of octets received. The field value indicates that all octets have been received up to that number.

d. Checksum holds the sum of all octets in the datagram. The receiving end recomputes this number upon the datagram's arrival and checks it against the header data field. If the two numbers do not agree, the datagram is retransmitted.

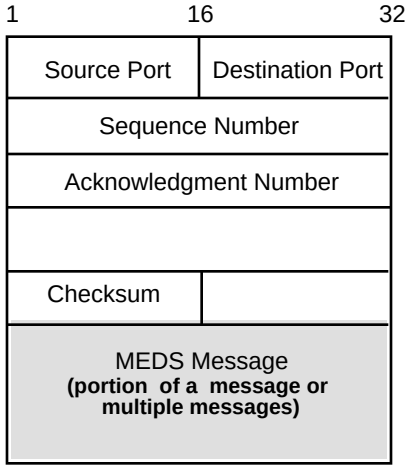


Figure 8-2. TCP Header

8.2.2 UDP

Each UDP message contains a UDP header and a UDP data field; each message is called a datagram. UDP does not use acknowledgments to ensure the safe arrival of data. It primarily identifies the exact location of the datagram recipient. UDP header (Figure 8-3) fields are defined as follows:

a. Source Port identifies port to which a reply should be sent (optional; set to zero if not used).

b. Destination Port provides exact port to which datagram is to be delivered when it arrives at the host.

c. Message Length in octets of the UDP datagram includes UDP header and data.

d. Checksum allows receiving end to verify that data was received without error (optional; set to zero if not used).

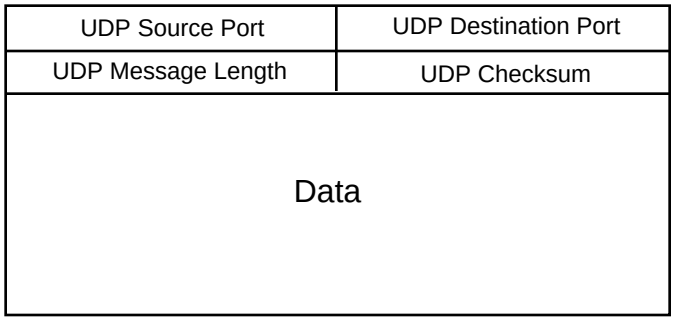


Figure 8-3. UDP Header

8.2.3 IP

TCP sends each datagram to IP. IP finds a route and transports the datagram to its destination. To accomplish this, IP must add its own header. Header fields of primary interest (Figure 8-4) are defined as follows:

a. Source and Destination Addresses reflect Internet addresses of sending and receiving machines.

b. Protocol number that instructs receiving end to send datagram to TCP.

c. Header Checksum is a sum of the IP header. The receiving end recomputes this value, compares it against the header value, and discards (which requires retransmission of datagram) any datagram for which the two values do not agree.

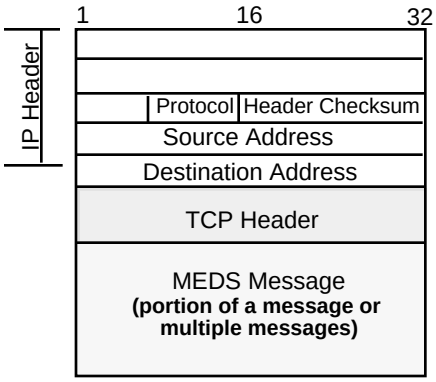


Figure 8-4. IP Header used with TCP

SECTION 9
TELEMETRY PROCESSING CONTROL ENVIRONMENT

9.1 INTRODUCTION

This section is prepared as a design description for the TPCE software developed by the Software and Automation Systems Branch, Code 522.2. The TPCE software will be used in the ETS VLS system to control the Code 521 VME system to be used in the VME LRS. This document introduces the TPCE project software and details the major components within it. Then the requirements that TPCE satisfies are listed with respect to the TPCE build/component that satisfies the requirement. The components that need some work to complete the ETS VLS system are mentioned to present an idea of the changes to be done.

The TPCE software is a set of reusable C++ classes that is used to control and monitor a telemetry processing system. The telemetry processing system is a VME-based system developed by the Microelectronics Systems Branch, Code 521. The VME system is composed of a set of hardware components that accomplish the telemetry processing. The VME system communicates with the TPCE using MEDS.

9.2 TPCE GRAPHIC USER INTERFACE DESCRIPTION

This section describes the GUI concepts used in TPCE. The major TPCE GUI panels are illustrated and explained. The panel illustration are taken from the TPCE library of generic user interfaces or from other specific projects for which TPCE has been used. The panels taken from other projects are intended to be illustrative and will be modified as necessary to meet the specific needs of the ETS system.

The TPCE Main Window is shown in Figure 9-1. This window consists of one or two VME icons (reference Figure 9-2) that represent the VME systems under TPCE control in the current configuration, a scrolled text window, and several pull-down menus. Since TPCE can control either the LRS VME or the HRS VME or both in any specific configuration, the window may have either one or two VME icons.

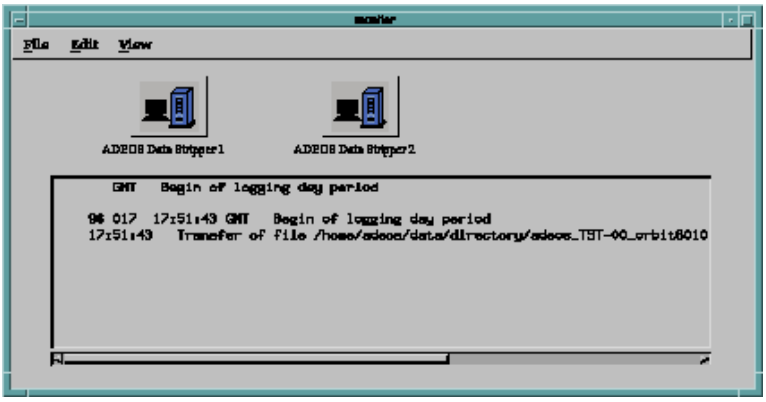


Figure 9-1. TPCE Main Window

Figure 9-2. VME Icon

The scrolled text window of the Main Window displays the current event log. Each message in the log is tagged with a time stamp and message severity code. Selecting either of the VME icons causes a pop-up menu of further information about the VME system (see Figure 9-3). This menu contains status information about the current session, and sub-menus for selecting detailed subsystem status pages (illustrated in Figure 9-4) or manual VME system commanding (see Figure 9-5).

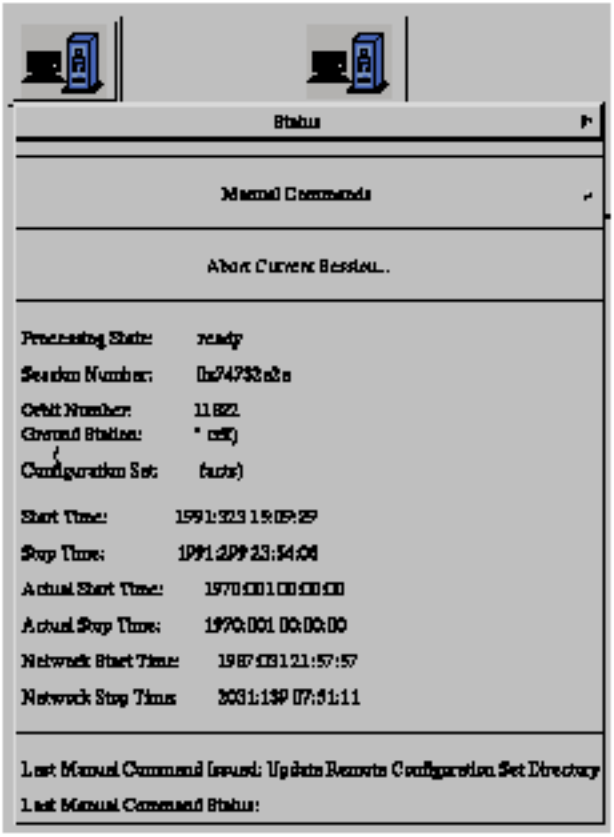


Figure 9-3. VME Quick Status Information

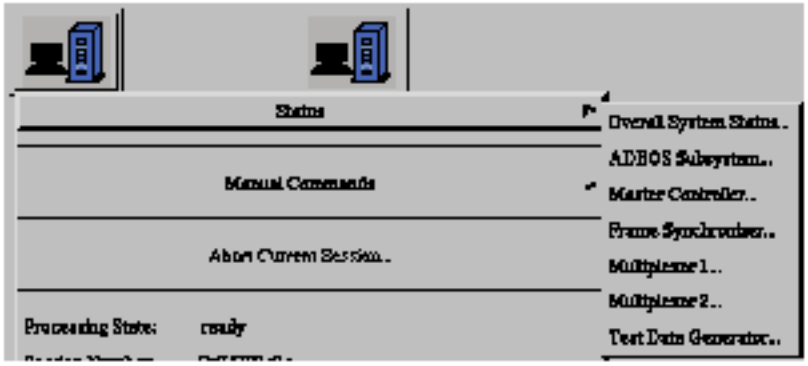


Figure 9-4. VME Status Page Menu

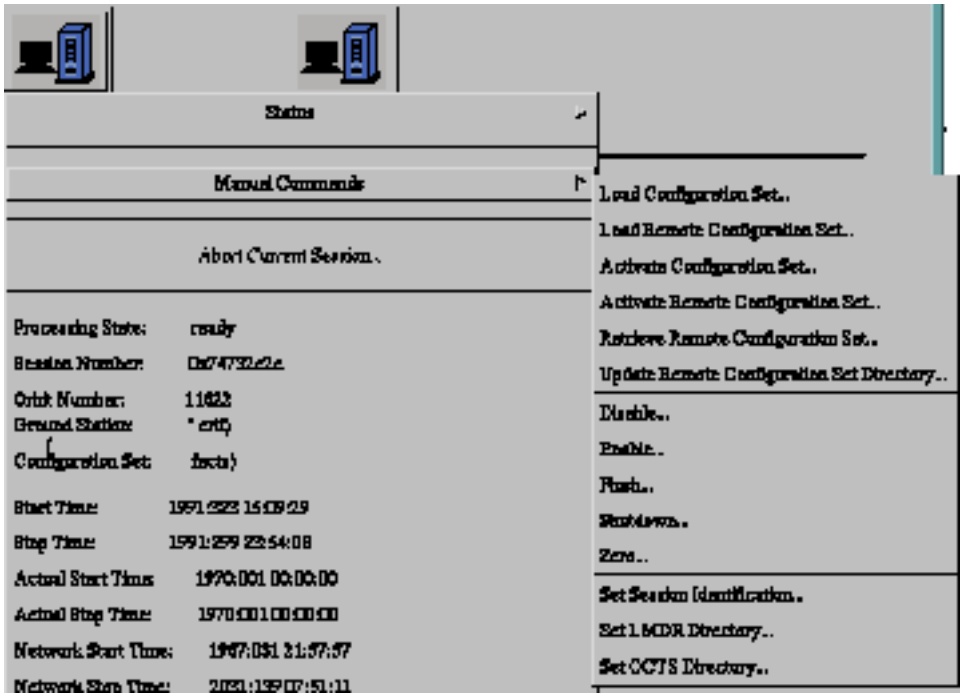


Figure 9-5. Manual Commands Menu

Three pull down menus from the toolbar provide access to various TPCE functions.

The File menu in Figure 9-6 allows the user several options: to request the start of a new event log file, causing the current file to be closed and saved; to request the current event log file to be printed; or to exit the TPCE system.

Figure 9-6. File Menu

The Edit menu in Figure 9-7 allows the user to access the GUI to edit the TPCE activity schedule, to add annotations to the current event log, to edit configuration sets, to edit TPCE system preferences, or to apply filters to the messages displayed in the event log.

Figure 9-7. Editor Menu

The View menu in Figure 9-8 allows the user to access the GUI to monitor and/or control data set distribution, to view selected data set contents, to view the contents of old event log files, to view the contents of Quality Assurance (QA) files from past sessions, and to toggle control of debug message display in the event log.

Figure 9-8. View Menu

Event log annotation, which is invoked from the Main Window Edit menu, uses the dialog shown in Figure 9-9. Each annotation is time stamped with the current time when the entry is made.

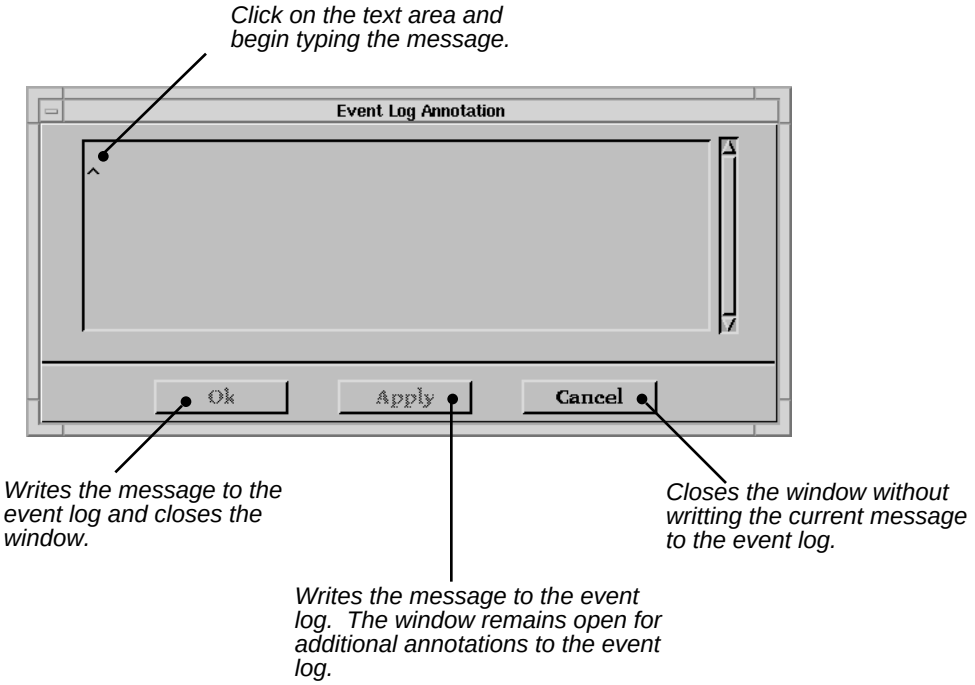


Figure 9-9. Event Log Annotation

The activity schedule editor, which is invoked from the Main Window Edit menu, uses the dialog shown in Figure 9-10. The main feature of this window is a set of graphic time lines, one for each VME system in the current configuration. The display can be re-sized to increase or decrease the time window displayed or it can be scrolled to display the time window of interest within the current activity schedule. Sessions are represented on the time line display as bars whose vertical size is proportional to the time length of the session

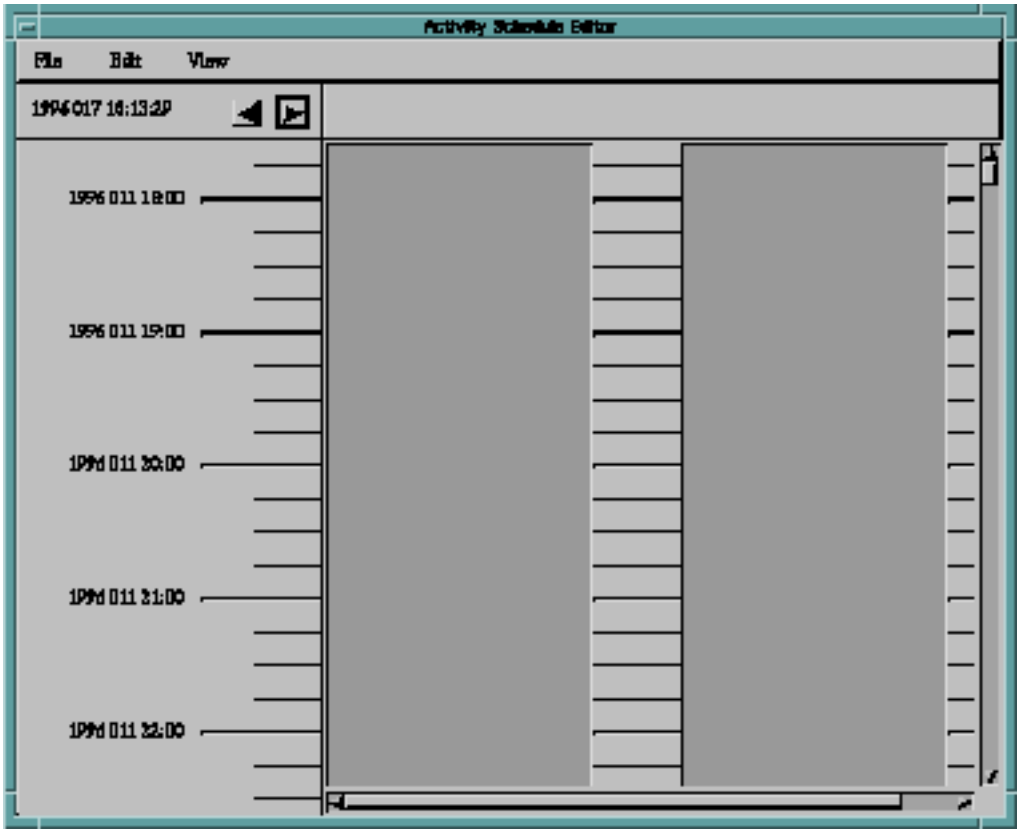


Figure 9-10. Activity Schedule Editor

Three pull down menus from the toolbar provide access to various schedule related functions.

The File menu in Figure 9-11 allows the user to save the current schedule to a file, to print the schedule in text form, or to close (exit) the activity schedule editor.

Figure 9-11. Activity Schedule Editor File Menu

The Edit menu allows the user to access the GUI to define a new session to be added to the schedule, to modify a session currently in the schedule, to delete a session currently in the schedule, or to re-assign a session from one VME to another (if the configuration includes multiple VMEs capable of performing the same session). To modify, delete, or re-assign a session, it must be selected by clicking its representation in the graphic time line.

The View menu in Figure 9-12 allows the user to access functions the automatically scroll the time lines to the current time, to view the QA report for a selected (past) session, or the view the attributes of a selected session.

Figure 9-12. Activity Schedule Editor View Menu

Invoking the New Session or Modify Session functions from the Edit menu causes the session editor window shown in Figure 9-13 to be displayed. The session editor allows the user to define the attributes of a session or modify the attributes of a previously defined session. These attributes include the session type, session start time, session stop time, VME configuration set, orbit number, and ground station.

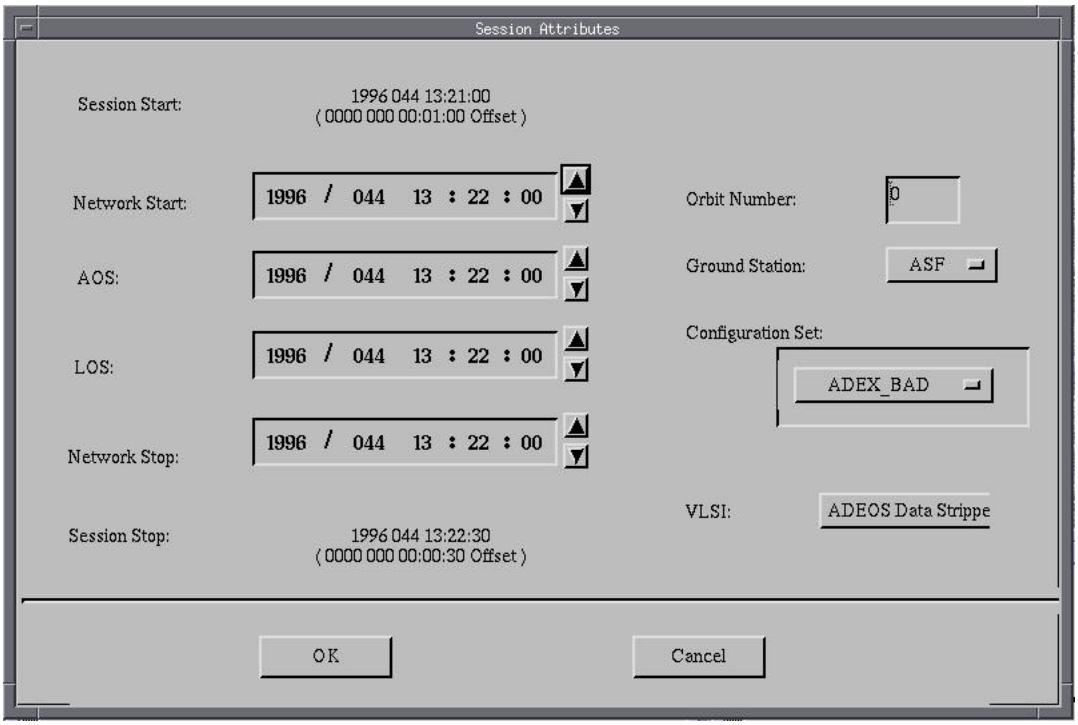


Figure 9-13. Session Editor Window

The data set distributor editor, which is invoked from the Main Window View menu uses the dialog shown in Figure 9-14. The main features of this window include a scrollable list of names of files queued for transmission, status of the data distribution process, attributes of the file currently being transmitted (if any), and a set of buttons for exercising limited control of the distribution process. The control buttons allow the user to display the attributes of a file (selected from the transmission queue), remove a file from the transmission queue, halt or resume transmission, and request re-transmission of a file that is still in the system.

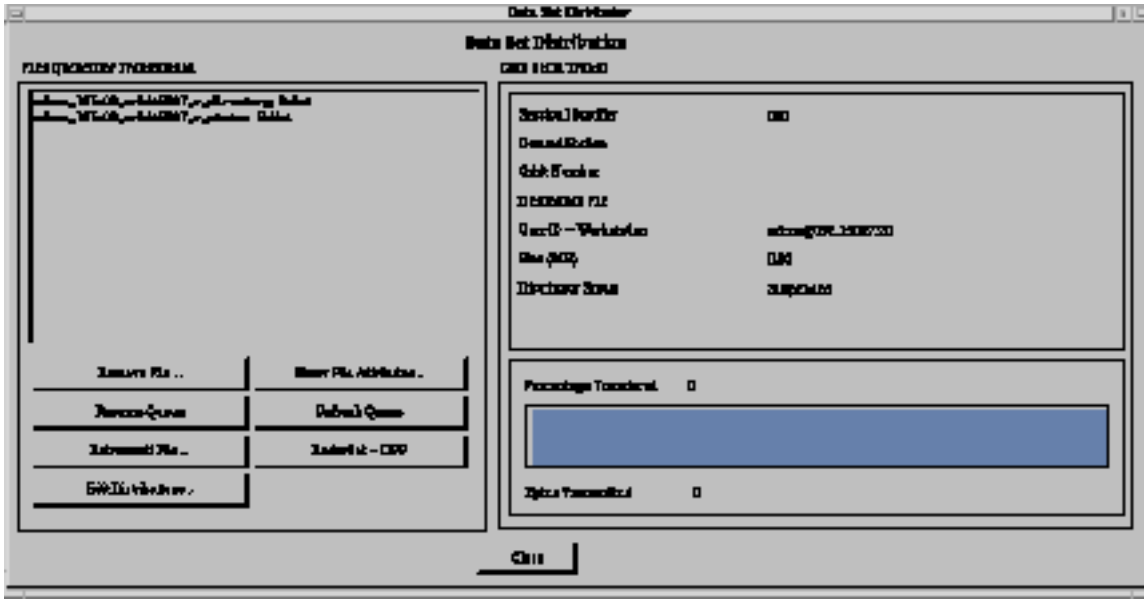


Figure 9-14. Data Set Distributor Editor

Requesting re-transmission of a file from the data set distributor editor causes the data set re-transmission dialog shown in Figure 9-15 to be displayed. The main features of this editor are three scrollable lists. The first list contains the names of the session for which data sets are still available for re-transmission. When a session is selected from this list, a set of files available for the session appears in the second scrollable list. The third scrollable list represents a queue of files to be re-transmitted.

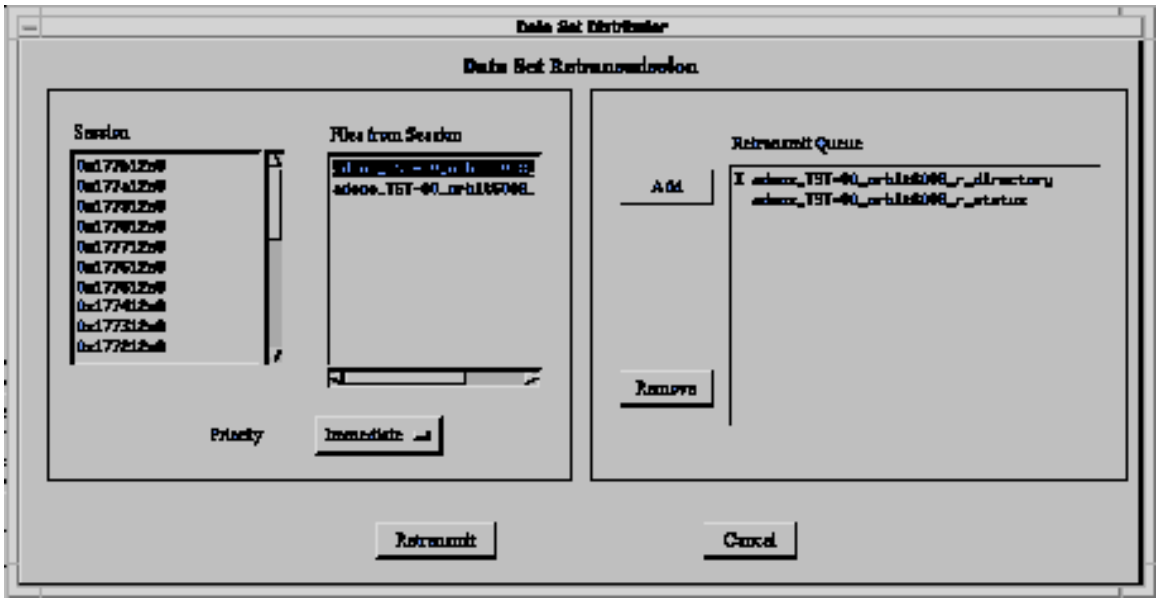


Figure 9-15. Data Set Retransmission

To request re-transmission of a file on this list, the user selects the file on the second list and clicks the add button. The file name should then appear on the third list. To delete a file erroneously placed on the retransmission queue, the user selects its name on the third list and then clicks the delete button. The name should disappear from the third list. When the third list correctly represents the list of files the user wishes to be re-transmitted, the user clicks the retransmit button causing the list to actually be queued for retransmission.

The VME system status pages for the various subsystems can be viewed by selecting corresponding items from the status pull-down menu of the VME icon shown in Figure 9-4. Examples of typical status page editors are shown in Figures 9-16 and 9-17. Figure 9-16 represents a summary type status compiled from various subsystems, while Figure 9-17 represents a typical detailed status page from a single subsystem.

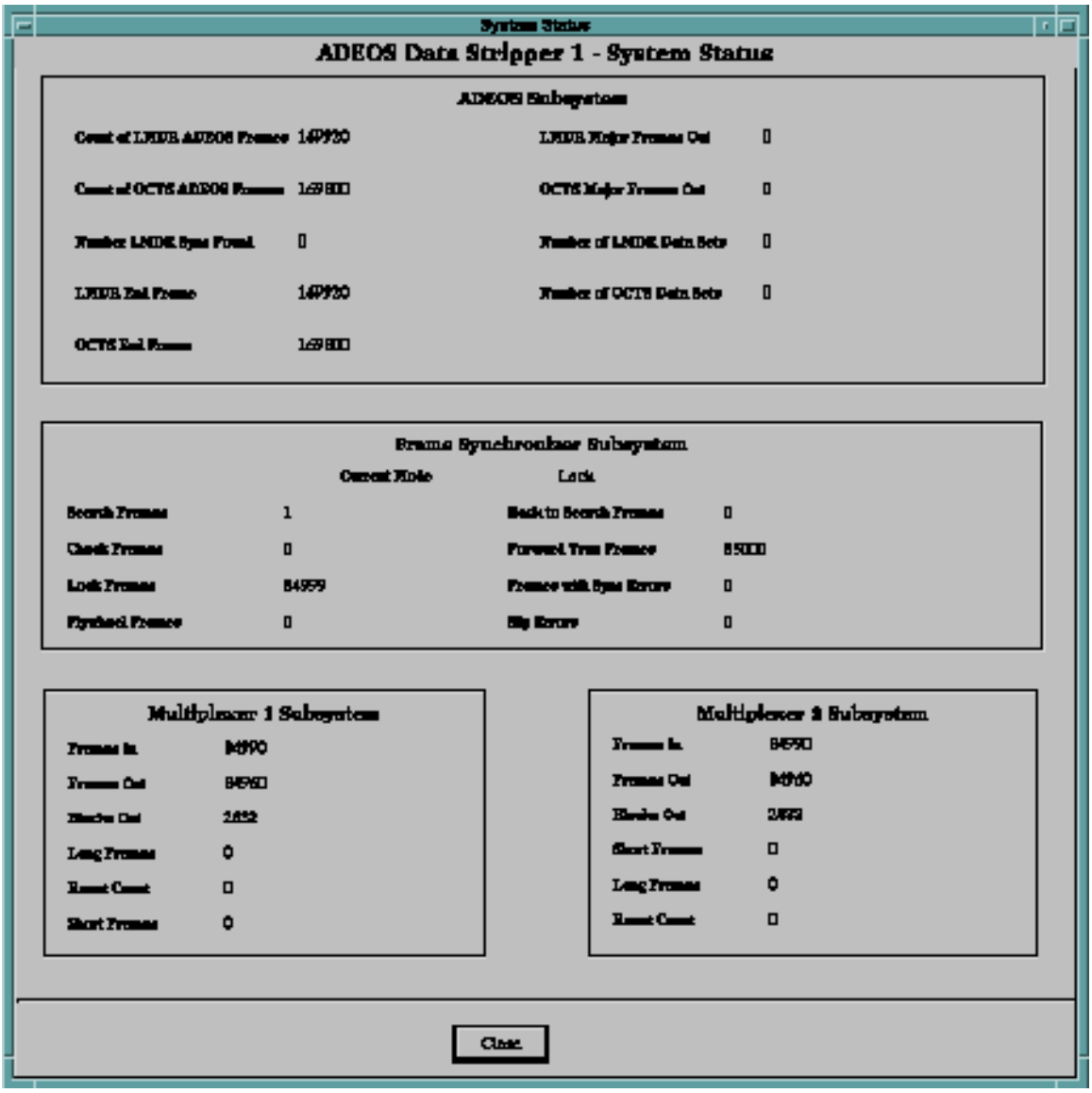


Figure 9-16. Example System Status Page

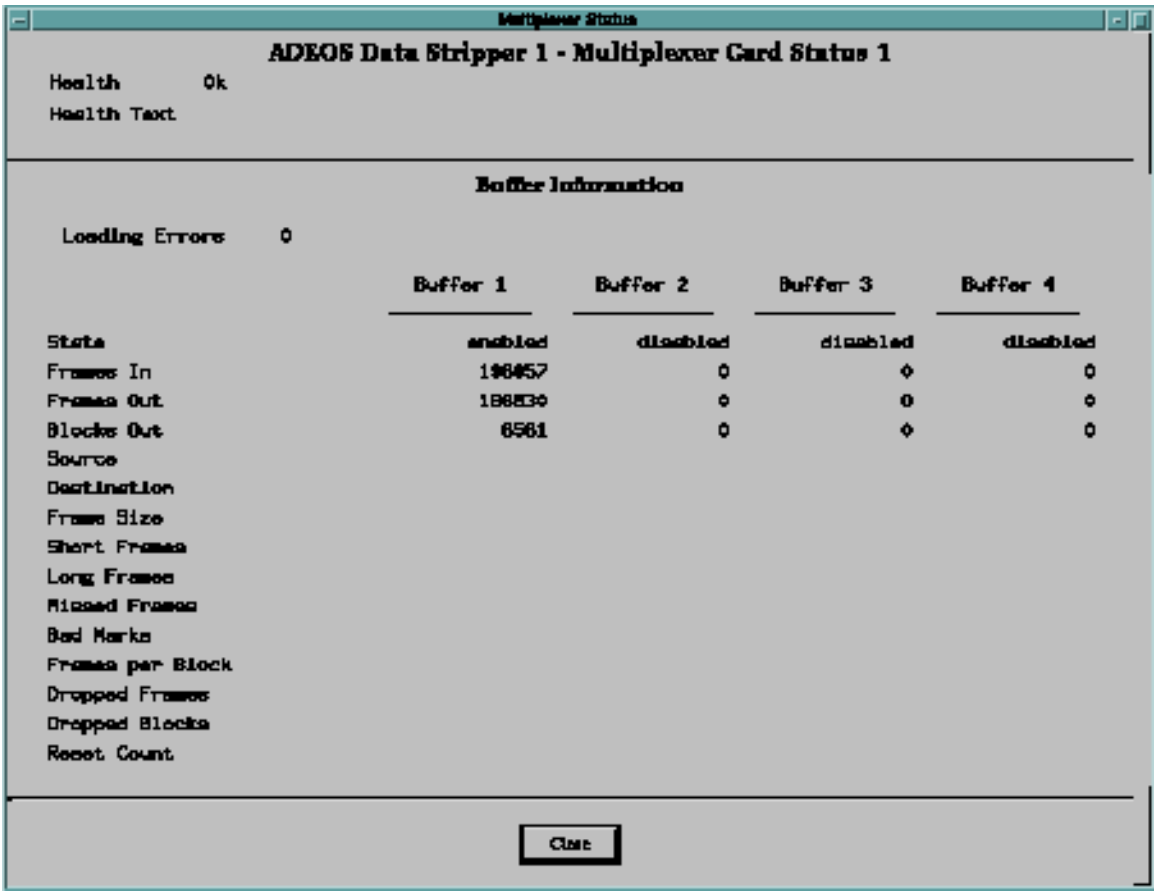


Figure 9-17. Example Subsystem Status Page

The configuration set editor, which is invoked from the Main Window Edit menu, uses the dialog shown in Figure 9-18. This editor uses a scrolled list of known subsystems from which the user selects the subsystem(s) to which the configuration set is to be applied. The parameters appropriate to the subsystem are then displayed in a second list. As each parameter is selected on the second list, its description and current value (in the configuration set) are displayed. The user may change the value to be the desired value, within the constraints defined for the parameter. When all values have been set to the desired values, the configuration set may be saved under the indicated configuration set name.

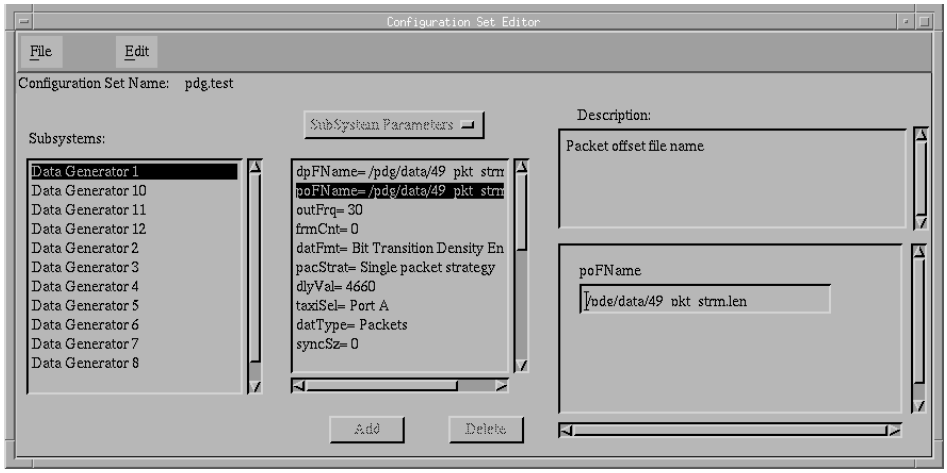


Figure 9-18. Configuration Set Editor

The preference editor, which is invoked from the Main Window Edit menu, uses the dialog shown in Figure 9-19. This editor displays a list of preference categories in a scrolled list. When a category is selected, a list of the preferences named for the category is displayed in a second scrolled list. When a preference name is selected, its description and current value are displayed. The user may change the value to be the desired value, within the constraints defined for the parameter. The changed value may be saved as the new preference to be used when the TPCE system is restarted.

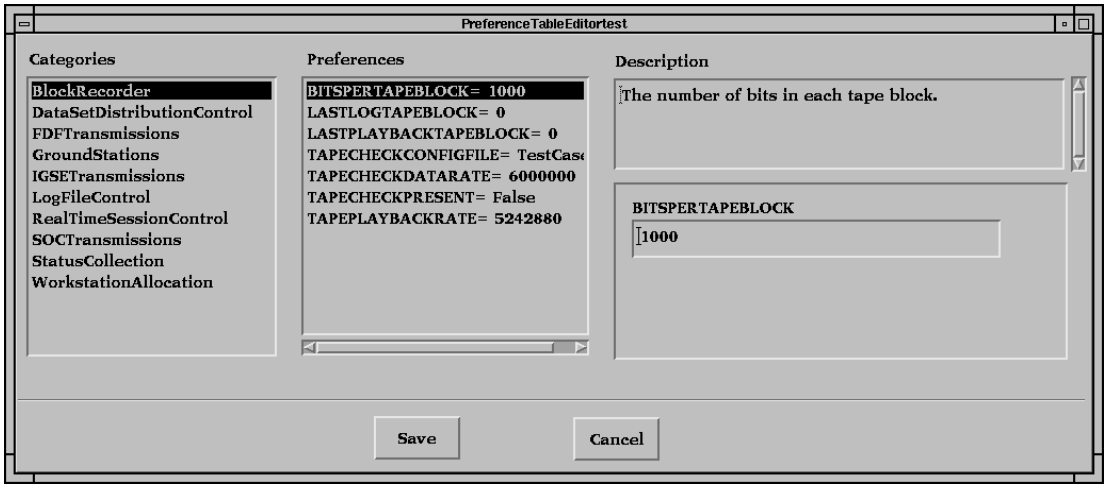


Figure 9-19. Preference Editor

9.3 TPCE SOFTWARE ARCHITECTURE

TPCE software is organized into fourteen major software components. Figure 9-20 shows the fourteen components and the communication between them. The following subsections describe each major component.

9.3.1 VLSI SERVER

The VLSI server acts as a command and response gateway to the VME. This component receives command requests, via the IPC server, from a component attempting to control the VME and packages commands into MEDS format. Once a MEDS command is created, a unique signature key is placed in the structure header. The command is then written to the VME command port. After the VME handles the MEDS command, a response is returned to the command port. The signature key is used to identify the command the response was for. The command response is unpacked and sent, via the IPC server, to the requesting component.

The VLSI server will periodically send a NOOP command to the VME to ensure the connection to the VME is active. If the command fails to return, then the connection is deemed down. The VLSI server will broadcast a message to the IPC server indicating that the connection is inactive. The VLSI server will attempt to reconnect. When reconnection is established, a message is broadcast to the IPC server indicating the connection is active.

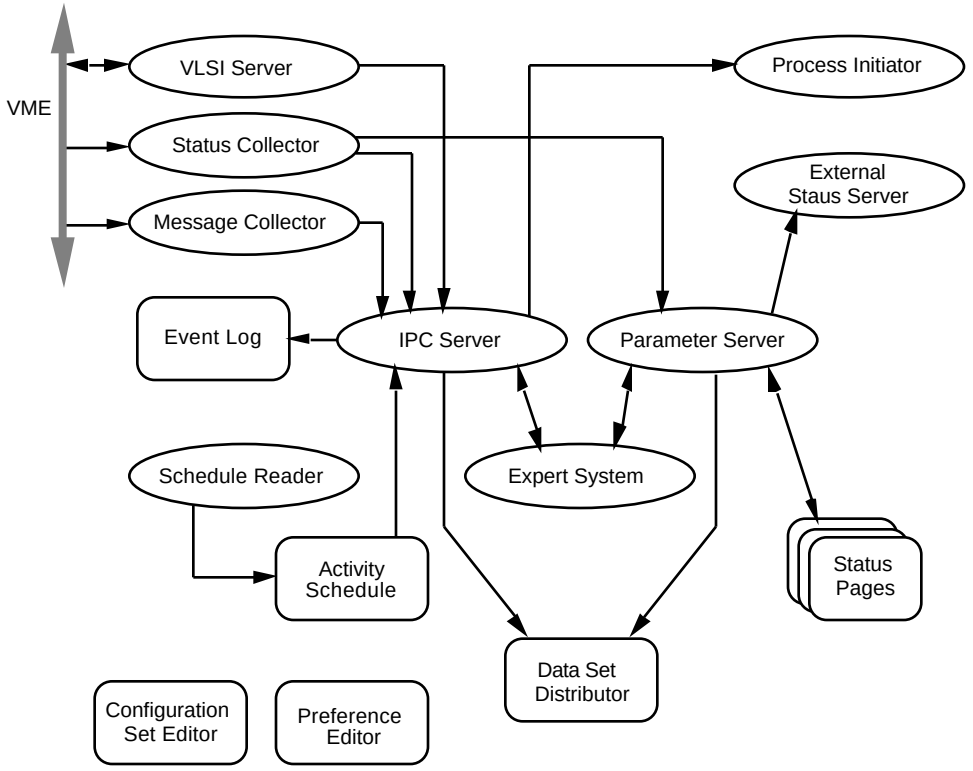


Figure 9-20. TPCE Software Architecture

9.3.2 STATUS COLLECTOR

The status collector receives all the VME status parameters. This component connects to a port on the VME to receive status. After connecting to the port, the status collector sends a return status command to the VLSI server via the IPC server. The return status command contains the status update rate the VME will use to send status. Once status comes into the status collector, each status item is extracted and written to the parameter server.

9.3.3 MESSAGE COLLECTOR

The message collector receives all VME unsolicited event messages. This component connects to a port on the VME to receive messages. The messages are sent whenever a pre-defined event occurs on the VME. The message is receive and sent to the event log via the IPC server. Before message processing is complete, the message opcode is examined to determine whether the message should be routed elsewhere. If the opcode matches a list of routable messages then it is broadcast to the IPC server with a special IPC server ID. Message processing is then completed. Any component may register to receive the specific messages that it requires to perform its function.

9.3.4 PROCESS INITIATOR

The process initiator component is used to control execution of Unix processes or to send special messages to other processes. The definition of what processes can be controlled is defined in a system control file. The control file contains process execution directives or keywords. A process execution directive can describe a process that should be started once the TPCE system starts or when indicated via an IPC message. For directives that are to be started via an IPC message, the process initiator receives a message from the IPC server with the directive identifier. The initiator reads the definition for the directive and acts on it. The action is to send an IPC message to open a User Interface (UI) component, spawn a Unix shell with the specified Unix command, or access the Unix system services to execute on another hosts.

The process initiator has the added functionality to route commands to another instance of the initiator software that resides on another host or to query on the executability of another host. Also the initiator can query the Unix display to determine whether a specified host can route X Windows displays on its console. This provides the mechanism to have X Window displays routed to an X Windows host on the network.

9.3.5 EXTERNAL STATUS SERVER

This component was created to provide status updates to non-TPCE software. This component will allow an external status client to connect to a TCP/IP socket on the host computer and receive specified status items. The client will receive only the status items that have been pre-defined in the TPCE system. This server alleviates the need for a software component to write MEDS translation software or to compile with TPCE code. All that is needed from a client is a connection to the Unix port specified in the system services.

9.3.6 EVENT LOG

The event log provides event messages for display on a UI or to a terminal, file, or printer. The events come from either user entry (annotation), VME generation, TPCE software, or external sources.

The user entry mechanism allows users to enter an informational message that get time-tagged and sent to the log. The VME generation mechanism is a connection, via the IPC server, to log all messages that are directed to the event log from the message collector. TPCE software messages are those events the have been deemed significant. These messages are generated by the software and are sent to the IPC server with event log direction. External sources can send event messages to the event log by sending IPC messages to the IPC server with event log direction. This capability implies external sources use TPCE software to send the messages.

The event log processes four types of messages: information messages, warning messages, error messages, and debug messages. By default, the debug messages are not displayed on the UI. Debug output can be turned on through a main window view menu item. Messages tagging depends on the message type. Normally, every logged message contains a time tag followed by the message itself. If the message is an error message, it is prefixed with the string “E->.” If the message is a warning message, it is prefixed with the string “W->.” Information and debug messages are output without a prefix.

9.3.7 IPC SERVER

The IPC server is the central communication component of the TPCE system. The server provides the communication mechanisms for one component to talk to another component. A message originator registers a connection to the IPC server for sending messages. As a message is sent to the IPC server, a message ID is used to

uniquely identify the message type. A client of the IPC server also registers itself with a connection to the IPC server. The client then registers a request for messages by message types.

When the IPC server receives a message, a search is done to determine whether any clients have requested to receive this message. If no clients have requested the message, then the message is stored but not propagated anywhere. If a client has requested the message, the message is routed to the client and the search continues. A message originator may direct a message to a specific client or may broadcast the message to any client registered for the message type. A client receiving a message is informed of the message originator and type, as well as the contents of the message.

9.3.8 PARAMETER SERVER

The parameter server acts very much like the IPC Server with the exception that messages are all status items. Messages are either a group of items or individual items. Registration is done by keywords rather than IDs. When a client receives an item, the item must then be extracted from the received message.

9.3.9 SCHEDULE READER

The schedule reader defines the contents of an operational schedule file that may be received from a source outside TPCE. The operational file is used to automatically setup activities to be performed at specified times with indicated activity parameters. As each activity is read from the file, a session event is created that is then passed to the activity schedule.

9.3.10 ACTIVITY SCHEDULE

The activity schedule receives a session event from the schedule reader and creates an alarm to go off at a pre-defined time prior to the start of the session event. When this alarm goes off, the session event is passed to the expert system to process the session. The activity schedule provides a set of UI software to allow a user to create a session event. The UI software also allows a user to modify session events. The actions in each session event are pre-defined and implemented in the TPCE software.

9.3.11 EXPERT SYSTEM

The expert system is a set of C++ software and C Language Integrated Productions System (CLIPS) rules that are used to create actions based on a defined rule base. Each rule is a definition of existing conditions and a set of actions required to progress from the current state to a new state. Together the rules define the state machines that guide VME systems through the execution of processing sessions. State machines are organized in a simple hierarchical structure. A master state machine monitors and controls the flow of session events to a set of VLSI (VME) state machines, one per VME system.

Different VLSI state machines can have different rule bases, which is necessary for the single TPCE system to concurrently control different VME systems, such as the ETS LRS and the ETS HRS.

When the master state machine receives a session event, it checks the current state of the intended VLSI state machine and communicates the event to that state machine if the machine is in a ready state.

When the VLSI state machine receives the event, it initiates the actions to sequence the VME system through the states that comprise the session. Commands are sent to the VLSI system via the VLSI server. Responses to the commands as well as unsolicited messages from the VLSI system are used as facts asserted to the state machine to synchronize the steps and assure proper set up, execution, and termination of the processing.

9.3.12 DATA SET DISTRIBUTOR (HRS ONLY)

The data set distributor allows TPCE and the user to monitor and control the transmission of data sets produced by the VME system. As data sets are produced, the data set distributor is notified of the progress via an unsolicited message from the VME.

The data set distributor maintains a data base of prescribed data distribution tables indicating which data set types should be transmitted to which users. It also maintains a queue of data sets waiting to be transmitted to users. At any time, it can also query the VME for directories of data sets available for transmission.

Based on this information, the data set distributor commands the VME to transmit data sets to intended users.

The data set distribution editor allows the user to monitor and intervene in the data set transmission operation. The user can also request that data sets be retransmitted when problems occur.

9.3.13 CONFIGURATION SET EDITOR

The configuration set editor essentially operates off-line from the rest of the TPCE software. This component defines the contents of a VME system configuration set. Each configuration set contains a group of VME subsystems each with a set list of parameters and values. The configuration set editor allows users to create or modify configuration files. This component allows users to add or remove defined subsystems. Within each subsystem is a set of parameters, each of which has a value with pre-defined constraints. The UI will allow the user to set each parameter to an acceptable value.

9.3.14 PREFERENCE EDITOR

The preference editor allows a user to modify the parameters used to configure the TPCE software. These parameters are critical to the normal operation of the TPCE system. The form of modifications are similar to the configuration set editor. If modifications are done to preferences, the user must take specific action (copying a file) to ensure that the modified preferences are used. The new preferences will then be used during the next execution of the TPCE software.

9.3.15 STATUS PAGES

A status page is a UI component that allows the user to view VME status items. Status pages are broken into logical organizations, such as VME card or subsystem pages. Each status page registers with the parameter server to receive status items. When a status item is received by the status page, the item is reflected on the display.

9.4 NECESSARY VLS MODIFICATIONS

The modifications needed to support the LRS are defined in the previous section. Most of the modifications deal with supporting new subsystems in the TPCE system. These include the FEP Card status, catalog, and command MEDS data structures.

SECTION 10
SYSTEM TEST APPROACH

The system test plan will be developed to validate the ETS VLS in a simulated environment that is close to a real-world scenario. System testing verifies that the system is ready for delivery. Testing includes tests for user interfaces and overall functional capabilities. System testing will be designed to determine whether the delivered software complies with functional, performance, operational, and interface requirements while operating in a test environment. The principal intent of system testing is to verify that a system satisfies the requirements allocated to that build or release as documented in the approved Requirements Document. Figure 10-1 depicts an approach of the system test configuration.

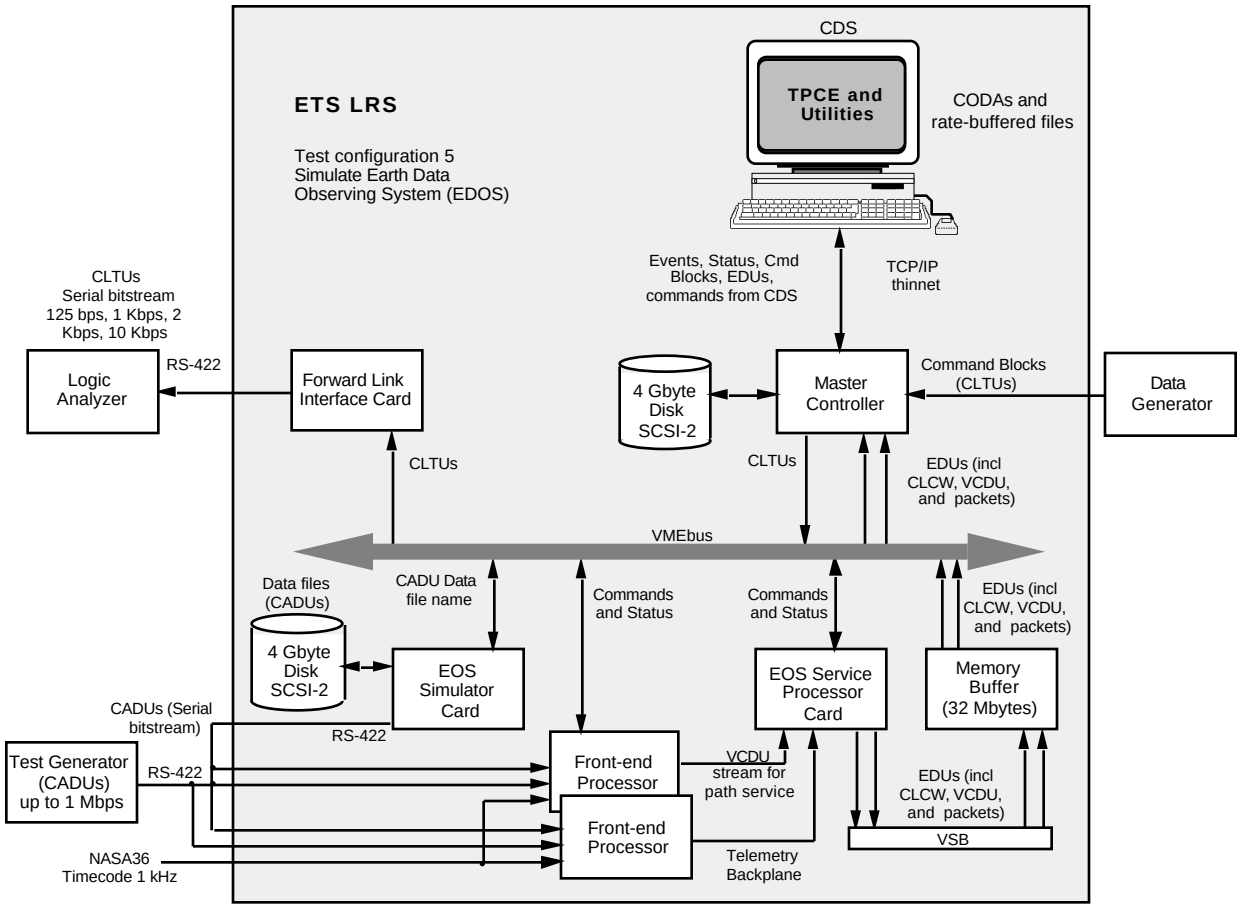


Figure 10-1. System Test Configuration

Initial tests will be concentrated on following individual cards:

a. MCC.

b. EOS Simulator Card.

c. FEP Card.

d. EOS Service Processor Card.

e. Memory Buffer.

The fully populated system will then be evaluated using the Test Data Generator Card as well as the external test equipment for total functional compliance.

10.1 SYSTEM TESTS

System test cases acceptance criteria are defined as a part of the system test plan. The system tests are conducted by the system test team. Test results are documented in a system test report prepared by the team. The system test team, in conjunction with the development team, uses verification criteria to evaluate the test results and determine system acceptability. Discrepancy reports are written for any problems or anomalies encountered.

System testing, as a minimum, will demonstrate that:

a. The system satisfies all operational requirements under nominal and capacity conditions.

b. Each system function affected by newly developed and newly modified software included in the system is correctly processed and produces responses consistent with the function.

c. The system responds correctly to simulated anomalies.

d. Outstanding problems found in previous software deliveries are resolved satisfactorily in the current release/build.

10.2 TEST GUIDELINES

Tests will be conducted to prove functional compliance with operational requirements of the referenced specifications. The following guidelines will be used to measure the success and thoroughness of a test:

a. All minimum functions defined by the established standard protocols are executed correctly at least once.

b. Verification tests will be automatically executed to accelerate testing and reduce errors.

c. Capability will be provided to perform step-by-step analysis for comparison between the expected and actual results.

d. Dynamic test modification capability will be provided to:

(1) Retest specific sequences that previously had problems.

(2) Enable manual inputting of augmented and/or new test sequences for the identification/evaluation of potential problem areas.

10.3 TEST APPROVAL

All test procedures, test configuration, and test equipment will be approved by the project manager.

10.4 ENVIRONMENTAL TEST CONDITIONS

System equipment will be tested in the Code 521 laboratory. The tests will be performed under the following environmental conditions:

a. The temperature at the test location will be within the range of 0°C to $+36.7^{o}$C.

b. The relative humidity at the test location will be within the range of 20% to 80%.

10.5 DISCREPANCIES REPORTING AND RETEST

Any constraints and irregularities observed during the prescribed test will be recorded on Discrepancy Report (DR) form. Every DR is evaluated and assigned a priority.

10.6 FAILURE DURING TEST

If a component of the system fails during the performance of any test, the following action will be taken:

a. The test will be immediately terminated and the failure will be recorded in the Problem Reporting System (PRS).

b. Project management will review problem reports and classify problems according to their severity.

10.7 RETEST AND REGRESSION TESTING

Upon completion of problem resolution, the test will be resumed at the point or portion of the test cycle where the failure occurred, or all necessary tests will be repeated as specified by the evaluator(s).

Prior to each new release, all necessary tests that were successfully executed under previous release may be repeated as determined by the evaluator(s).

10.8 TEST DEPENDENCIES

The test will be conducted after satisfying the following dependencies:

a. Availability of test equipment (e.g., local control terminal or workstation) and tools listed for each test.

b. Availability of the test procedures that have been reviewed and approved by the project manager.

c. Completion of mechanical and electrical assembly inspection after a replacement of a board.

ACRONYMS AND ABBREVIATIONS

Term	Definition
AIS	Automated Information System
AM	Amplitude Modulated
AOS	Advanced Orbiting Systems
APID	Application Process Identifier
ASIC	Application-specific Integrated Circuit
AUI	Attachment User Interface
BCD	Binary Coded Decimal
BCH	Bose-Chaudhuri-Hocqueghen
BTD	Bit Transition Density
CADU	Channel Access Data Unit
CCB	Configuration Control Board
CCITT	International Telegraph and Telephone Consultative Committee
CCSDS	Consultative Committee for Space Data Systems
CDS	Control and Display Subsystem
CLIPS	C Language Integrated Production System
CLCW	Command Link Control Word
CLTU	Command Link Transmission Unit
CODA	Customer Operations Data Accounting
COTS	Commercial Off the Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CVCDU	Coded Virtual Channel Data Unit
DCN	Documentation Change Notice
DDS	Detailed Design Specification
DMA	Direct Memory Access
DPR	Dual-ported RAM
DR	Discrepancy Report
DRAM	Dynamic RAM
DSB	Disabled
EBnet	EOSDIS Backbone Network
ECL	Emitter Coupled Logic
EDOS	EOS Data and Operations System
EDU	EDOS Data Unit
EGS	EOS Ground System
ENA	Enable
EOC	EOS Operations Center
EOF	End of File
EOS	Earth Observing System
EOSDIS	EOS Data and Information System
ESH	EOS Service Header
ETS	EOSDIS Test System
FEP	Front-End Processor
FIFO	First-in, First-out
FLCC	Forward Link Card Controller
FTP	File Transfer Protocol
GaAs	Gallium Arsenide
GM	Global Memory
GND	Ground
GMH	Ground Management Header
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
GVCID	Global Virtual Channel Identifier
H/W	Hardware
HCI	Human Computer Interface
HRTB	High-rate Telemetry Backplane
HSI	High-speed Interface
I&T	Integration and Test
I/O	Input/Output

Term	Definition
IC	Integrated Circuit
ICD	Interface Control Document
ID	Identifier
IEEE	Institute of Electrical and Electronic Engineers
IP	Internet Protocol
LAN	Local Area Network
LRS	Low-Rate System
M_PDU	Multiplexing Protocol Data Unit
MCC	Master Controller Card
MDCL	Microprocessor Decode and Control Logic
MEDS	Modular Environment for Data Systems
MO&DSD	Mission Operations and Data Systems Directorate
MSB	Microelectronics Systems Branch
Nascom	NASA Communications
NCO	Numerically Controlled Oscillator
NTLM	Nontelemetry
NRZ	Nonreturn to Zero
NRZ-L	Nonreturn-to-Zero on Level Conditions
NRZ-M	Nonreturn-to-Zero on Mark Conditions
NRZ-S	Nonreturn-to-Zero on Space Conditions
OMD	Operations Management Data
OPMAN	Main Operation Interface
PB1	Parallel Binary
PCA	Processor Communication Area
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PN	Pseudorandom Noise
PRS	Problem Reporting System
R/W	Read/Write
RAM	Random Access Memory
RICOM	Remote Interface Communication
RIMOVER	Remote Interface Data Mover
S/W	Software
SBC	Single Board Computer
SCID	Spacecraft ID
SCITF	Spacecraft Integration and Test Facility
SCSI	Small Computer System Interface
SCTGEN	Simulated CCSDS Telemetry Generator
SLC	Space Link Command
SMA	Subminiature Assembly
SRAM	Static RAM
SRS	System Requirements Specifications
SSIM	Spacecraft Simulator
STGEN	Simulated Telemetry Generator
TAXI	Transparent Asynchronous Transmitter-receiver Interface
TBS	To Be Supplied
TCP/IP	Transmission Control Protocol/Internet Protocol
TPCE	Telemetry Processing Control Environment
UART	Universal Asynchronous Receiver-Transmitter
UDP/IP	User Datagram Protocol/Internet Protocol
UI	TPCE User Interfaces
UTC	Universal Time Code
VC	Virtual Channel
VCA	Virtual Channel Access
VCDU	Virtual Channel Data Unit
VCID	Virtual Channel Identifier
VLS	VME Low-rate System
VLSI	Very Large Scale Integration
VME	Versa Module Eurocard
VMEbus	Versa Module Eurocard bus
VSB	VME Subsystem Bus

APPENDIX A
REQUIREMENTS TRACEABILITY MATRIX

The following requirements traceability matrix provides a reference from the Level 4 requirements contained in Volume 4 of the ETS Requirements Specification to hardware subsystems, software modules, and system builds.

In the matrix, build refers to the number in which a requirement is successfully delivered.

Rqmt. Nos.	ETS VLS Requirements Description	Build	Module
2.1.1.1	The LRS shall provide two input serial ports and one output serial port. All ports shall comply with RS-422 standard with differential data and clock signals.	1	FEP FLIC
2.1.1.2	The LRS shall use the two input ports for receiving and processing two streams of CADUs from SCITF/SSIM through EBnet. The maximum data input rate is 512 Kbps for one stream, and 16 Kbps for the other.	1	FEP
2.1.1.3	The LRS shall use the output port to test interface between EOC and spacecraft, as the EDOS Simulator. The LRS shall output spacecraft commands in CLTU bitstream to the SCITF/SSIM via EBnet at rates from 125 bps to 10 Kbps.	1	FLIC
2.1.1.4	The LRS interface with the EOS AM-1 SCITF shall be the same as specified for the spacecraft to EBnet interface, as contained in Reference 1.4.3.e.	1	FLIC
2.1.1.5	The LRS interface with the EOS AM-1 SSIM shall be the same as the LRS interface with the SCITF, as contained in Reference 1.4.3.f.	1	FLIC
2.1.2.1	The LRS shall provide an Ethernet interface. The Ethernet shall conform to 10Base2 of Institute of Electrical and Electronic Engineers (IEEE) 802.3 standard, also known as Thin Ethernet.	1	MC
2.1.2.2	The LRS shall interface with EOC through the Ethernet using the IP suite, including TCP/IP, and UDP/IP.	1	MC
2.1.2.3	The LRS shall, through the Ethernet interface, receive CLTU command data blocks from EOC and output EDUs (packets and CLCWs) to the EOC. All data transferred to this Ethernet interface are based on UDP/IP protocol.	1	MC
2.1.2.4	The LRS shall, through the Ethernet interface, output rate-buffered data files to the EOC. All data transferred to this Ethernet interface are based on FTP and TCP/IP protocol.	1	MC
2.1.2.5	The interface between the VLS and CDS shall use FTP and TCP/IP. The information includes followings: a) VLS commands from the CDS to the VLS. b) Status from the VLS to the CDS. c) Event messages from the VLS to the CDS. d) Command data blocks and CLTUs from the VLS to CDS for archives. e) EDUs from VLS to CDS. f) Test data files from CDS to VLS.	1 1 1 1 1 1 1	MC
2.1.2.6	The Ethernet interface shall provide an effective bandwidth of 2.5 Mbps.	1	MC
2.1.3.1	The VLS shall provide two NASA36 timecode interfaces with a BNC connector for each interface. The interfaces will be used for the VLS to ingest external timecodes for time-stamping return-link VCDUs.	1	FEP

2.1.3.2	The VLS shall ingest external timecode in 1-kHz time interval and output timecode in either in one millisecond (PB1 format) or one microsecond (PB5J format) resolution internally.	1	FEP
3.1.1	The LRS shall conform to the CCSDS Recommendations for Space Data Systems applicable to the EOSDIS project as specified in this document.	1	FEP SVP
3.1.2	The LRS shall provide the capability to accept and distribute test data by electronic transmission.	1	VLS
3.1.3	The LRS shall provide the capability to read and interpret flags in all headers (refer to Section 1.4.2).	1	FEP SVP
3.1.4	The LRS shall provide the capability to validate all headers of received data.	1	FEP SVP MC
3.1.5	The LRS shall provide the capability to generate static test data.	1	SIMCard STGEN
3.1.6	The LRS shall use the Universal Time Code (UTC) format for time-of-day-related data, as required by the test.	1	FEP
3.1.7	The LRS shall provide the capability to generate the summary status, quality, and accounting information.	2	MC
3.1.8	The LRS shall provide a local user interface through an RS-232 port.	1	VLS OPMAN
3.1.8.1	The local user interface shall allow users to issue commands, check system health, and monitor system processing status.	1	OPMAN TPCE
3.1.8.2	The local user interface shall allow users to edit system and session configuration sets.	1	OPMAN TPCE
3.1.8.3	The local user interface shall allow users to execute a processing session.	1	OPMAN TPCE
3.1.9	The LRS shall be able to support remote user interface through an Ethernet link.	1	TPCE VLSI_ Server
3.1.10	The LRS shall provide self-test and self-diagnostic capabilities.	2	TPCE VLS
3.1.10.1	The LRS shall perform self-test upon boot-up and provide results to the user.	2	TPCE VLS
3.2.1	The VLS shall be able to receive the CLTUs in the form of EDOS command data blocks from EOC through EBnet using UDP/IP protocol.	1	MC
3.2.1.1	The VLS shall maintain a buffer to store incoming command data blocks with the minimum size of 16 Kbytes.	1	FLIC MC

3.2.1.2	The VLS shall check the command data blocks ground header. The checking fields shall include: a) Message type. b) Source ID. c) Destination ID. d) Version number. e) Sequence count. f) Message length.	1 1 1 1 1 1 1	MC
3.2.1.2.1	The VLS shall send an event message to CDS for status.	1	MC
3.2.1.3	The VLS shall continue processing a command data block even it fails the validation of the EDOS header.	1	MC
3.2.1.4	The VLS shall maintain a circular buffer to store all input command data blocks with status for snapshot and debug.	1	MC
3.2.1.5	The VLS shall maintain and report status for command blocks. The status includes as a minimum: a) Number of received command blocks. b) Number of command data blocks pass checking. c) Number of command data blocks fail checking.	1 1 1 1	MC
3.2.1.6	The VLS shall send all received command data blocks to CDS for archival.	1	MC OMD_ Utility
3.2.2	The VLS shall strip out command data block Ground Message Header and forward rest of data (CLTUs) to SCITF/SSIM.	1	FLIC MC
3.2.3	The VLS shall be able to transmit CLTUs to EBnet through the RS-422 serial output interface.	1	FLIC
3.2.4	The VLS shall be able to output CLTUs in throughput mode (as they are received).	1	FLIC
3.2.5	The VLS shall optionally be able to output idle sequences to maintain the connection in the absence of CLTUs.	1	FLIC
3.2.6	The VLS shall be able to transmit CLTUs in 125 bps, 1 Kbps, 2 Kbps, or 10 Kbps rate. The rate can be changed in between CLTUs.	1	FLIC
3.2.7	The VLS shall discard command data blocks when the aggregate forward-link data exceeds the capacity of the physical channel.	1	MC
3.2.8	The VLS shall maintain a circular buffer to store failed command data block for snapshot and debug.	1	MC
3.3.1.1	The VLS shall be able to receive return-link data from an external source via the RS-422 serial input port.	1	FEP
3.3.1.2	The VLS shall be able to perform all processing in support of the CCSDS AOS VCDU Grade-2 service for return-link data.	1	FEP SVP
3.3.1.2.1	The VLS shall be able to perform Reed-Solomon error detection and correction on each CVCDU.	1	FEP
3.3.1.3	The VLS shall be able to support CCSDS AOS VCDU Service.	1	FEP

3.3.1.4	The VLS shall be able to detect and synchronize on the following modes of data: a) Normal polarity, forward bit order. b) Inverted polarity, forward bit order.	1 1 1	FEP
3.3.1.5	The VLS shall be able to correlate to synchronization pattern up to 32 bits in length.	1	FEP
3.3.1.6	The VLS shall perform search/check/lock/flywheel synchronization strategy with error tolerance of between 0 and 7 bits.	1	FEP
3.3.1.7	The VLS shall invert the bits of each CADU detected to have inverted polarity.	1	FEP
3.3.1.8	The VLS shall be able to correct bit slips, selectable between 0 and plus 3 bits, in a CADU, by padding to the proper length.	1	FEP
3.3.1.9	The VLS shall be able to correct bit slips, selectable between 0 and minus 3 bits, in a CADU by truncating to the proper length.	1	FEP
3.3.1.10	Upon detection of the first frame in a session, the VLS shall notify the CDS of synchronization.	1	FEP
3.3.1.11	The VLS shall be able to reject or discard, as a user option, any VCDU that has failed Reed-Solomon decoding.	1	FEP
3.3.1.12	The VLS shall discard and account for fill VCDUs.	1	FEP
3.3.1.13	The VLS shall be able to time-stamp each VCDU.	1	FEP
3.3.1.14	The VLS shall maintain CADU processing quality and accounting information for each processing session, including:: a) Frame-synchronization status. b) CADUs received. c) Back-to-search count. d) Search CADUs. e) Check CADUs. f) Lock CADUs. g) Flywheel CADUs. h) Forward inverted CADUs. i) Forward true CADUs. j) CADUs with bit slip detected. k) CADUs with errors in synchronization pattern. l) Fill CADUs.	1 1 1 1 1 1 1 1 1 1 1 1 1 1	FEP TPCE Status_ Collector
3.3.1.15	The VLS shall allow a user to zero CADU processing quality and accounting indicators.	1	FEP OPMAN
3.3.1.16	The VLS shall maintain frame Reed-Solomon quality and accounting information for each processing session, including as a minimum: a) Operation mode. b) Interleave depth. c) Codeword length. d) CADUs output from the passthrough path. e) CADUs filtered. f) CADUs output. g) Header error counts.	1 1 1 1 1 1 1 1	FEP OPMAN

	h) Headers corrected. i) Uncorrectable headers. j) Codeword error counts. k) Codewords corrected. l) Uncorrectable codewords. m) CADUs with corrections. n) Uncorrectable CADUs.	1 1 1 1 1 1 1	
3.3.1.17	The VLS shall allow a user to zero Reed-Solomon processing quality and accounting counts.	1	FEP OPMAN
3.3.1.18	The VLS shall also maintain counts for each VC, including as a minimum: a) VCDUs received. b) VCDUs with CRC errors. c) VCDUs with correctable Reed-Solomon errors. d) Reed-Solomon symbols corrected. e) VCDUs with sequence errors. f) Missing VCDUs.	1 1 1 1 1 1 1	FEP OPMAN
3.3.1.19	The VLS shall allow a user to zero VC data quality counts.	1	FEP OPMAN
3.3.1.20	The VLS shall be able to extract CLCWs from CADUs from selected VCs.	1	SVP
3.3.1.21	The VLS shall reject a VCDU that contains any of the following errors: a) Invalid SCID (optional). b) Invalid frame version number (optional). c) Invalid VCID. d) Invalid first header pointer error. e) Invalid APID. f) Packet length error. g) No packet header (including the secondary header) found. h) VCDUs with uncorrectable Reed-Solomon errors.	1 1 1 1 1 1 1 1 1	FEP SVP
3.3.1.22	The VLS shall reject a VCDU if it contains an error in the header or a packet piece. However, all packets before the erroneous packet piece shall be processed and output.	1	FEP SVP
3.3.1.23	The VLS shall generate annotation for each rejected VCDU with the following information: a) Frame synchronization and Reed-Solomon decoding quality trailers. b) Byte location of error in the VCDU. c) Flags defining error condition(s).	1 1 TBD 1	FEP
3.3.1.24	The VLS shall send to CDS information regarding rejected VCDUs in the order they are received	1	FEP MC
3.3.1.25	The VLS shall be able to identify and process VCDUs of VCs that request AOS VCDU service.	1	FEP
3.3.1.26	The VLS shall be able to connect up to two separate NASA36 timecode interfaces.	1	FEP
3.3.1.26.1	The VLS shall use either a BNC or SMA connector for each timecode interface.	1	FEP
3.3.1.26.2	The output of timecode shall be in PB5J format. It will be used for time-stamping received CADUs	1	FEP
3.3.2.1	The VLS shall be able to support the CCSDS AOS Path Service for return-link data.	1	SVP
3.3.2.2	The VLS shall be able to identify independent sources by any combination of stream number, SCID, VCID, and APID.	1	SVP
3.3.2.3	The VLS shall extract packet pieces from VCDUs requesting AOS path service and reassemble packets.	1	SVP

3.3.2.4	The VLS shall be able to process source packets with variable length.	1	SVP
3.3.2.5	The VLS shall be able to generate fill data using a selectable 16-bit fill pattern to complete packet fragments that have complete packet headers corresponding to the packet length defined by the valid packet header.	1	SVP
3.3.2.6	The VLS shall be able to discard and account for fill packets.	1	SVP
3.3.2.7	The VLS shall be able to discard any packet fragment without an associated valid header.	1	SVP
3.3.2.8	The VLS shall be able to check for packet sequence discontinuities and determine the minimum number of packets missing in sequence gaps.	1	SVP
3.3.2.9	The VLS shall be able to generate an ESH as specified in the EDOS External DFCD.	1	SVP
3.3.2.10	The VLS shall be able to generate an EDU by concatenating an ESH with each return-link VCDU for VCDU service.	1	SVP
3.3.2.11	The VLS shall be able to generate an EDU by concatenating an ESH with each return-link packet.	1	SVP
3.3.2.12	The VLS shall be able to generate an EDU by concatenating an ESH with each extracted return-link CLCW.	1	SVP
3.3.2.13	The VLS shall be able to identify forward-ordered playback VCDU data based on the replay flag in the VCDU primary header.	1	FEP
3.3.2.14	The VLS shall maintain cumulative quality and accounting records by VC and by source: <u>By Virtual Channel:</u> a) SCID. b) VCID. c) Number of VCDUs processed. d) Number of VCDUs rejected. e) Number of APIDs. f) Number of packets processed. g) Number of VCDU sequence discontinuities. h) Number of missing VCDUs. i) Number of invalid application packets rejected. j) Number of VCDUs with CRC errors. k) Number of Reed-Solomon uncorrectable VCDUs. l) Number of Reed-Solomon corrected VCDUs. m) Number of packet source sequence counter discontinuities. n) Number of missing packets. o) Number of packets with fill.	1 <u>1</u> 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	FEP SVP

	By Source: a) SCID. b) VCID. c) APID. d) Spacecraft time of first packet. e) Spacecraft time of last packet. f) Number of packets processed. g) Number of length errors. h) Number of packets from VCDUs with errors. i) Number of packets with fill. j) Number of packet source sequence counter discontinuities. k) Number of missing packets. l) Number of EDUs output during the current session.	1 1 1 1 1 1 1 1 1 1 1 1 1	
3.3.2.15	The VLS shall be able to zero the cumulative quality and accounting records as defined in 3.5.2.15.	1	OPMAN
3.3.2.16	The VLS shall be able to output all received CLCWs to EOC.	1	MC SVP
3.3.2.16.1	The sequence order of EDUs output shall be the same as received.	1	SVP
3.3.2.16.2	The VLS shall output selected EDUs to CDS for rate-buffered file generation.	1	MC SVP
3.3.2.16.3	The VLS shall output EDUs to a CDS-specified socket through Ethernet link using TCP/IP.	1	MC
3.3.2.16.4	The VLS shall be able to output all or user-selected EDUs to EOC	1	MC
3.3.2.16.5	The VLS shall be able to transfer either one stream of EDUs to a single EOC-specified destination or two separate streams of EDUs to two separate EOC-specified UDP ports through Ethernet link using UDP/IP.	1	MC
3.3.2.16.6	The VLS shall maintain and report the EDUs delivery status. The status shall include: a) Number of EDUs sent to EOC. b) Total bytes of EDUs sent to EOC. c) Number of EDUs sent to CDS. d) Total bytes of EDUs sent to CDS.	1 1 1 1	MC
3.3.2.17	The VLS shall be able to output all of CLCWs to EOC.	1	MC SVP
3.3.2.17.1	The sequence order of CLCWs output during the session shall be the same as extracted.	1	SVP
3.3.2.17.2	The VLS shall maintain and report the CLCWs delivery status. The status shall include the following: a) Number of CLCWs sent to EOC. b) Total bytes of CLCWs sent to EOC.	1 1 1	MC SVP
4.1.1	The VLS shall be able to support a file system of 2 Gbytes.	1	VLS
4.1.2	The LRS will be able to complete a warm boot in no more than 10 minutes.	1	CDS VLS
4.1.3	The LRS will be able to complete a cold boot in no more than 15 minutes.	1	CDS VLS
4.2.1	The VLS shall transmit spacecraft command to SCITF/SSIM at 125 bps, 1 Kbps, 2 Kbps or 10 Kbps rate.	1	FLIC
4.3.1	The VLS shall be able to perform return-link processing at data rates up to 1 Mbps.	1	FEP SVP
4.3.2	The VLS shall be able to transfer EDUs from selected sources to a user-specified destination at aggregate rates up to 50 Kbps using UDP/IP protocol.	1	MC SVP
4.3.2.1	The VLS shall be able to transfer EDUs to the CDS at aggregate rates up to 1.25 Mbps using TCP/IP protocol.	1	MC SVP

4.3.3	The VLS shall have the capability to process up to two input streams simultaneously.	1	FEP
4.3.4	The VLS shall have the capability to process up to 8 SCIDs with values from 0 to 255 for CCSDS VCDU.	1	FEP
4.3.5	The VLS shall have the capability to process up to 31 VCs with IDs 0 to 63 for CCSDS VCDU.	1	FEP (24 VCs)
4.3.6	The VLS shall have the capability to process up to 512 Application Processes with ID value from 0 to 2046.	1	SVP
4.3.7	The VLS shall have the capability to process up to 512 sources.	1	SVP
4.4.1	The VLS shall be able to internally output timecode in one-millisecond in PB1 format or 1 microsecond in PB5J format.	1	FEP
4.4.2	The VLS shall be able to ingest NASA36 timecode in 1 kHz resolution time signals.	1	FEP
5.1.a	The GUI shall conform to the HCI Guidelines (see Reference 1.4.3.j).	1	TPCE
5.1.b	The GUI shall provide interaction to enable the user to exercise the defined functions (refer to ETS GUI Guide) associated with the LRS.	2	TPCE
5.1.c	The GUI shall display an acknowledgment to a user directive within two seconds of directive entry.	1	TPCE
5.1.d	The GUI shall display a response regarding an execution of a user command within five seconds of command entry.	1	TPCE
5.1.e	The GUI shall provide interaction to enable the user to display status page(s) at system-level and subsystem-level.	1	TPCE Status_ Editor
5.1.f	The GUI shall provide interaction to enable the user to configure the LRS to simulate EDOS forward- and return-link processing.	1	TPCE VLS
5.1.1.1	The GUI shall provide interaction to enable the user to view the activity schedule.	1	Activity_ Schedule
5.1.1.2	The GUI shall provide interaction to enable the user to add new session events to the activity schedule.	1	Activity_ Schedule
5.1.1.3	The GUI shall provide interaction to enable the user to delete selected session events from the activity schedule.	1	UI Activity_ Schedule
5.1.1.4	The GUI shall provide interaction to enable the user to modify selected session events from the activity schedule.	1	UI Activity_ Schedule
5.1.1.5	The GUI shall check schedule conflict for any newly entered session event.	1	Activity_ Schedule
5.1.1.6	The GUI shall report to the user any detected schedule conflict.	1	Activity_ Schedule
5.1.2.1	The GUI shall provide interaction to enable the user to create a new configuration set.	2	Config_ Set_ Editor
5.1.2.2	The GUI shall provide interaction to enable the user to delete a configuration set.	2	Config_ Set_ Editor
5.1.2.3	The GUI shall provide interaction to enable the user to view a configuration set.	2	Config_ Set_ Editor
5.1.2.4	The GUI shall provide interaction to enable the user to edit a configuration set.	2	Config_ Set_ Editor
5.1.2.5	The GUI shall provide interaction to enable the user to associate a configuration set to a session event.	2	Activity_ Schedule

5.1.2.6	The GUI shall check a newly entered values against an authorized range.	2	Config_ Set_ Editor
5.1.2.7	The GUI shall alert the user any entry that is out of range.	2	Config_ Set_ Editor
5.1.2.8	The GUI shall allow the user to enter an out-of-range value once it receives an override from the user.	1	Config_ Set_ Editor
5.1.3.1	Status, data quality, and accounting information shall be available to users, with displays of status for on-going sessions updating continuously at least once every five seconds.	2	Status_ Collector
5.1.3.2	The GUI shall provide interaction to enable the user to view the current system level session status, including any system errors.	1	UI
5.1.3.3	The GUI shall provide interaction to enable the user to view the current subsystem level session status (e.g., VLS).	1	UI
5.1.3.4	The GUI shall provide interaction to enable the user to view the current card-level session status (e.g., Service Processor, Simulator).	1	UI
5.1.3.5	The GUI shall provide interaction to enable the user to view status of available ports on the Ethernet LAN.	1	MC Unix
5.1.3.6	The GUI shall provide interaction to enable the user to view status of LRS ports on the EBnet.	1	MC
5.1.3.7	The GUI shall generate session reports summarizing the session system status, quality, and accounting information.	2	Expert_ System
5.1.3.8	The GUI shall provide interaction to enable the user to view the session reports.	2	UI
5.1.3.9	The GUI shall provide interaction to enable the user to print the session reports.	2	UI
5.1.3.10	The GUI shall provide interaction to enable the user to save the session reports.	2	Expert_ System
5.1.4.1	The GUI shall provide the capability to log user directives in the session journal.	1	Event_ Log
5.1.4.2	The GUI shall provide the capability to log all commands sent to the VLS in the session journal.	1	Event_ Log
5.1.4.3	The GUI shall provide the capability to log all command responses received from the VLS in the session journal.	1	Event_ Log
5.1.4.4	The GUI shall provide the capability to log all system events received from the VLS in the session journal.	1	Event_ Log
5.1.4.5	The GUI shall provide the capability to log all system errors in the session journal.	1	Event_ Log
5.1.4.6	The GUI shall time-tag each system event.	1	Event_ Log
5.1.4.7	The GUI shall provide interaction to enable the user to view the session journal.	1	UI
5.1.4.8	The GUI shall provide interaction to enable the user to annotate individual events in the session journal.	1	Event_ Log
5.1.4.9	The GUI shall allow the user to print the session journal.	1	UI
5.1.4.10	The GUI shall allow the user to save the session journal.	1	UI
5.1.5.1	The GUI shall provide interaction to enable user to run card-level diagnostic tests, subsystem-level self-tests, and system-level self-tests.	2	UI
5.1.5.2	The GUI shall provide interaction to enable the user to view the results of selected card(s) or subsystem(s) self-tests.	2	UI Event_ Log

5.1.5.3	The GUI shall provide capability to enable the user to log the results of selected card(s) or subsystem(s) self-tests.	2	UI Event_ Log
5.1.5.4	The GUI shall provide capability to enable the user to annotate the self-test results.	2	UI Event_ Log
5.1.5.5	The GUI shall provide capability to enable the user to view the selected self-test results of in the log.	2	UI Event_ Log
5.2.1	The GUI shall allow the user to specify processing parameters through the configuration sets.	2	Config_ Set_ Editor
5.2.2	The GUI shall allow the user to specify timeout value for end-of-session determination.	2	Config_ Set_ Editor
5.2.3	The GUI shall allow the user to select a mode in which the system will remain in processing mode until manual termination.	2	Config_ Set_ Editor
5.2.4	The GUI shall allow the user to configure the VLS with the user-selected configuration set.	1	UI Expert_ System
5.2.5	The GUI shall allow the user to specify destinations for data delivery.	1	UI
5.2.6	The GUI shall provide interaction to enable the user to monitor data processing status.	1	UI
5.2.7	The GUI shall notify the user when the processing session ends.	1	UI Expert_ System
5.2.8	The GUI shall generate a session status report summarizing the status of processing session.	2	Expert_ System
5.2.9	The GUI shall allow the user to enable data archiving function.	2	UI Utility
5.2.10	The GUI shall allow the user to disable data archiving function.	1	UI Utility