BBN Systems and Technologies
A Division of Bolt Beranek and Newman Inc.

BBN Report No. 7632

SIMNET CVCC

Simulation of the SINCGARS Radio System Software Design Document

Report No. 7632

SIMNET
CVCC

SIMULATION OF THE SINCGARS RADIO SYSTEM SOFTWARE DESIGN DOCUMENT

JULY 1991

Prepared by:

BBN Systems and Technologies
Advanced Simulation
10 Moulton Street
Cambridge, MA 02138 USA

Prepared for:

Defense Advanced Research Projects Agency (DARPA)
1400 Wilson Blvd.
Arlington, VA 22209-2308

This research was performed by BBN Systems and Technologies under Contract Nos. MDA972-89-C-0060 and MDA972-90-C-0061 to the Defense Advanced Research Projects Agency (DARPA). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policy, either expressed or implied, of DARPA, the U.S. Army, or the U.S. Government.

Table of Contents

List of Figures

1 SCOPE

1.1 Identification

This BBN Software Design Document describes the SINCGARS radio simulation Computer Software Configuration Item (CSCI) for the SIMNET M1 tank simulator, as composed of Computer Software Components (CSCs) and Computer Software Units (CSUs).

1.2 System Overview

SIMNET is an advanced research project sponsored by the Defense Advanced Research Projects Agency (DARPA) in partnership with the United States Army. Currently in its sixth year, the goal of the program is to develop the technology to build a large–scale network of interactive combat simulators. This simulated battlefield will provide, for the first time, an opportunity for fully–manned platoon–, company–, and battalion–level units to fight force–on–force engagements against an opposing unit of similar composition. Furthermore, it does so in the context of a joint, combined arms environment with the complete range of command and control and combat service support elements essential to actual military operations. All of the elements that can affect the outcome of a battle are represented in this engagement, with victory likely to go to that unit which is able to plan, orchestrate, and execute its combined–arms battle operations better than its opponent. Whatever the outcome, combat units will benefit from this opportunity to practice collective, combined-arms, joint war fighting skills at a fraction of the cost of an equivalent exercise in the field.

While simulators to date have been shown to be effective for training specific military skills, their high costs have made it impossible to buy enough simulators to fully train the force. Further, because of the absence of a technology to link them together, they have not been a factor in collective, combined-arms, joint training. SIMNET addresses both of these problems by aiming its research at three high–payoff areas:

- Better and cheaper collective training for combined-arms, joint war fighting skills
- A testbed for doctrine and tactics development and assessment in a full combined-arms joint setting
- A "simulate before you build" development model

These payoffs are achievable because of recent breakthroughs in several core technologies that have been applied to the SIMNET program:

- High speed microprocessors
- Parallel and distributed multiprocessing
- Local area and long haul networking
- Hybrid depth buffer graphics
- Special effects technology
- Unique fabrication techniques

These technologies, applied in the context of selective fidelity and rapid prototyping design philosophies, have enabled SIMNET development to proceed at an unprecedented pace, resulting in the fielding of the first production units at Fort Knox, Kentucky, just three years into the development cycle.

In addition to the basic training applications, work is underway to apply SIMNET technology in the area of combat development to aid in the definition and acquisition of weapon systems. This is made possible because of the low cost of the simulators, the ease with which they can be modified, and the ability to network them to test the employment of a proposed weapon system in the tactical context in which it will be used, i.e., within the context of the combined arms setting.

Work on SIMNET is being carried out by co–contractors Bolt Beranek and Newman Inc. (BBN) and Perceptronics, Inc. Perceptronics is responsible for training analysis, overall system specification, and the physical simulators, and BBN is responsible for the data communication and computer–based distributed simulation and the computer image generation (CIG) subsystems. The project is a total team effort.

DARPA is the DoD agency chartered with advancing the state of the art in military technology by sponsoring innovative, high–risk/high–payoff research and development.

1.3 Documentation Overview

1.3.1 Purpose

This document is a representation of the software system, and is created to facilitate analysis, planning, implementation, and decision making. It is also used as the primary medium for communicating software design information.

The BBN Software Design Document is used by the customer to understand the detailed design of the CSCI.

1.3.2 Contents

The contents of this document include the CSCI's detailed design,and data structures.

2 REFERENCED DOCUMENTS

There are no applicable documents.

3 PRELIMINARY DESIGN

The following describes the preliminary design for the SINCGARS CSCI.

3.1 CSCI Overview

The SINCGARS radio simulator and radio interface unit (RIU) simulate the RT-1523 SINCGARS radio, either in manpack or vehicle-borne configuration. The system can reproduce the attenuation of a radio signal due to distance and intervening terrain, the effects of transmitter and receiver characteristics, radio interference and jamming, and detectability of emissions and source. They permit communication of both voice and data This paragraph identifies and describes the role of this CSCI within the system to which this document applies. Figure 3.1-1 is a system architecture diagram of the relationships between this CSCI and the other system CIs. The data being passed in figure 3.1-1 is defined in the NAME CSCI Data Dictionary (Appendix A). The following identifies and states the purpose of each external interface of this CSCI:

a. *RIU* Interface. The purpose of this interface is to resolve conflicts between voice and data transmission, to queue requests for transmission and reception on behalf of the IVIS simulation, and to handle overrides of queuing by a human operator. It transmits and received IVIS simulator messages over the simulation Ethernet. It fragments these messages for transmission over the simulated radio channel, recomposing them on the receiving end.

b. *SINCGARS* Interface. The purpose of this interface is to transmit and receive voice and data from other radios and CHS equipment in the system, both real and simulated. It provides support for two different hardware interfaces. One interface allows connection of the simulated SINCGARS radio via an AUD/DATA connector to CHS hardware, specifically, an APIU. The other interface allows connection of the simulated SINCGARS radio via an RXMT (retransmit) connector to a real SINCGARS radio. Both connectors accept bitstreams at MIL-STD-188 levels.

Figure 3.1-1 is a system architecture diagram of the relationships between the SINCGARS radio simulator and RIU and the other system CIs of the communications simulation.

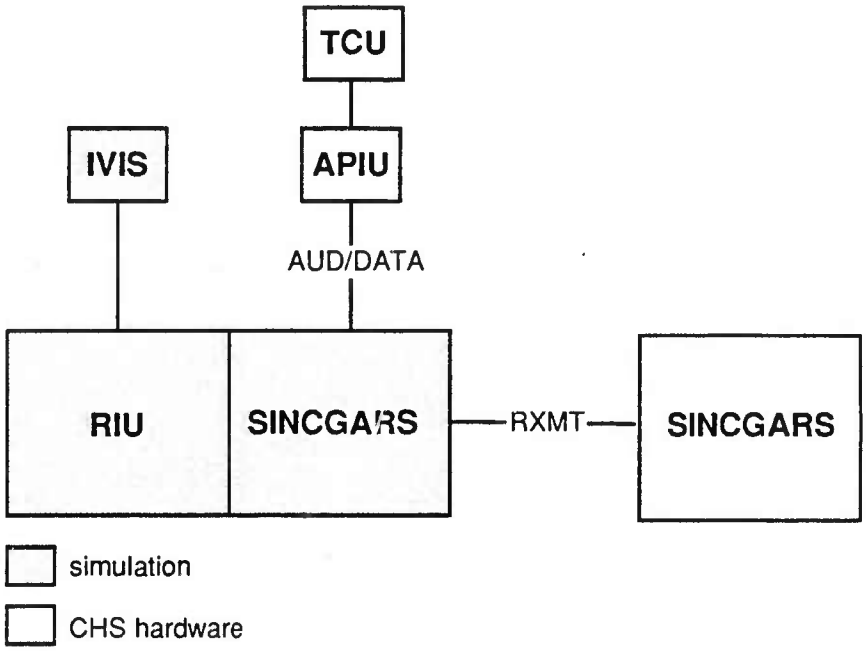

Figure 3.1-1 System Architecture of Relationships Between *SINCGARS* CSCI and the other CIs

3.1.1 CSCI Architecture

SINCGARS provides the following functions:

- maintenance of signal attenuation
- maintenance of vehicle, transmitter, and radio state information
- simulation of signal reception and output
- response to front panel and keyboard input
- response to voice input and simulation of signal transmission
- maintenance of interface to radio interface unit (RIU) simulation
- reporting of radio state information
- checking and resetting hardware interfaces

Figure 3.1-2 illustrates the top-level architecture.

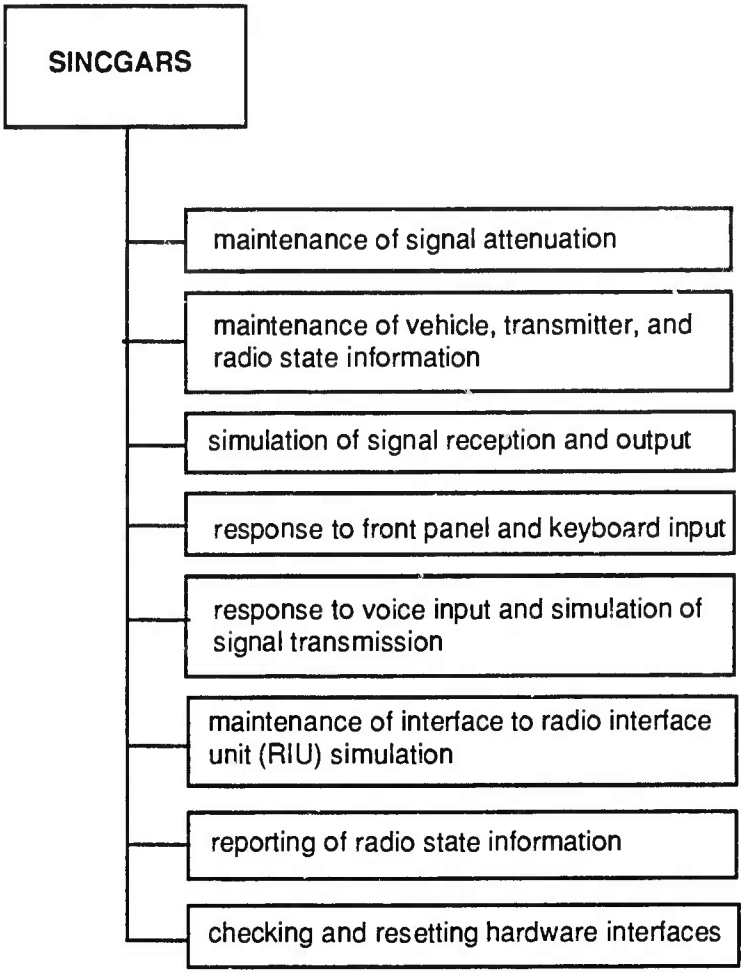

Figure 3.1.1-1 SINCGARS Functions

3.1.3 System States and Modes

Not Applicable

3.1.3 Memory and Processing Time Allocation

Not Applicable

3.2 CSCI Design Description

The following provides a design description of each CSC of this CSCI.

When not performing this interrupt–based processing, the host process is cycling through the function described in section 3.2.1.

Upon receiving an interrupt marking the start of a new tick, the host process performs the functions described in sections 3.2.2 to 3.2.8.

3.2.1 Maintenance of signal attenuation

The host process maintains current estimates of signal attenuation over each path, from every transmitter to each receiver it simulates. It recomputes signal attenuation for the paths between selected pairs of transmitters and receivers.

3.2.2 Maintenance of vehicle, transmitter, and radio state information

The host process maintains the state of each radio it simulates, knowledge of the vehicles within which those radios reside, and knowledge of what transmitters exist, where those transmitters are located in the simulated world, and what each transmitter's characteristics are.

3.2.3 Simulation of signal reception and output

The host process accepts signal PDUs from the network (and from the host's own transmitters), simulates signal detection and capture by each receiver, and supplies digitized speech signals to the voice I/O subsystem for output.

It checks for and processes any PDUs received from the network. For certain PDUs it updates its knowledge of simulated vehicles, transmitters, or the state of its own radios. The contents of signal PDUs bearing radio signal information may be passed to one or more speech I/O channels.

3.2.4 Response to front panel and keyboard input

The host process responds to inputs from radio front panel switches and keypads and updates the displays and internal radio state information accordingly;

Occasionally it determines whether any input has been received from a radio front panel. In response to such input, it determines that radio's new state, and perhaps outputs a new string of characters for display on the radio's front panel.

It checks for any terminal keyboard input. The terminal used to start the host process may be used to issue commands to it during its execution. These commands are meant primarily for debugging.

3.2.5 Response to voice input and simulation of signal transmission

The host process accepts digitized speech signals from the voice I/O subsystem and issues signal and intercom PDUs containing the signals;

3.2.6 Maintenance of interface to radio interface unit simulation

The host process supports an interface to the radio interface unit (RIU) simulation which allows each simulated RIU to sense channel activity, transmit data, and receive data.

It performs any periodic processing required for the simulation of RIUs.

3.2.7 Reporting of radio state information

The host process periodically sends PDUs that report the state of its radio transmitters and receivers.

3.2.8 Checking and resetting hardware interfaces

The host process checks that various hardware interfaces are responsive and resets those that are not.

4 DETAILED DESIGN

The following provides detailed design information for this CSCI.

4.1 CSCs

The following figures describe the relationships between the CSUs of each CSC.

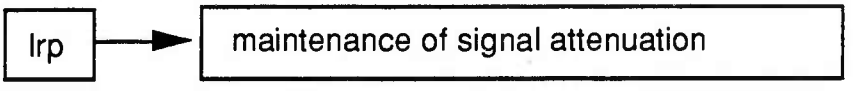

Figure 4.1-1 Maintenance of signal attenuation

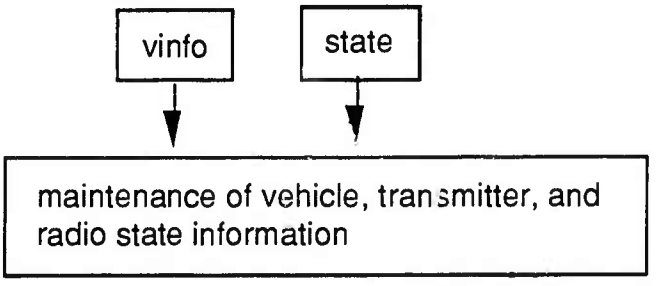

Figure 4.1-2 Maintenance of vehicle, transmitter and radio state information

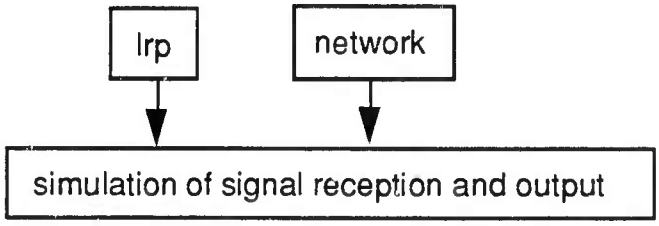

Figure 4.1-3 Simulation of signal reception and output

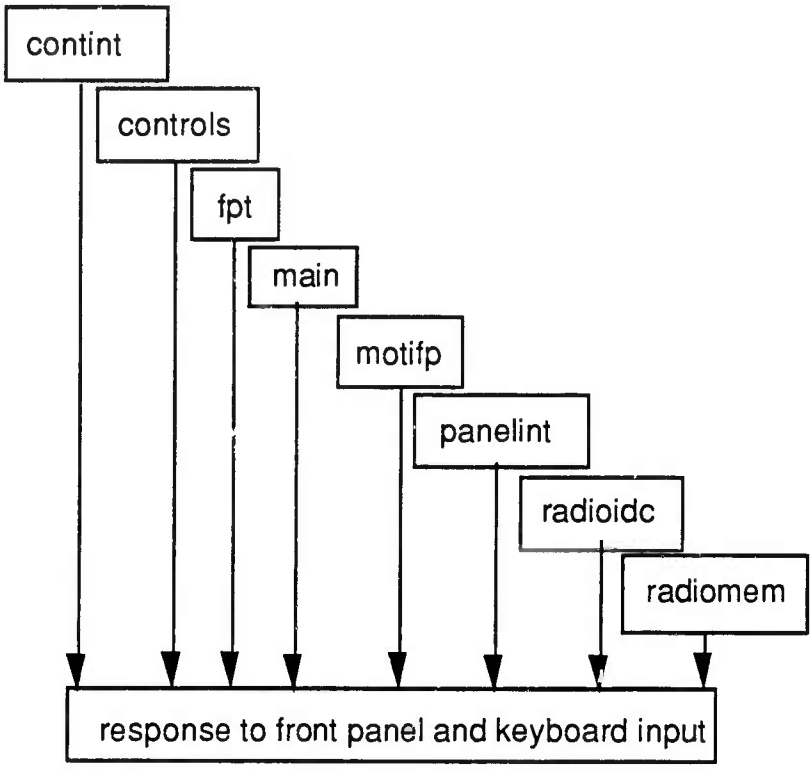

Figure 4.1-4 Response to front panel and keyboard input

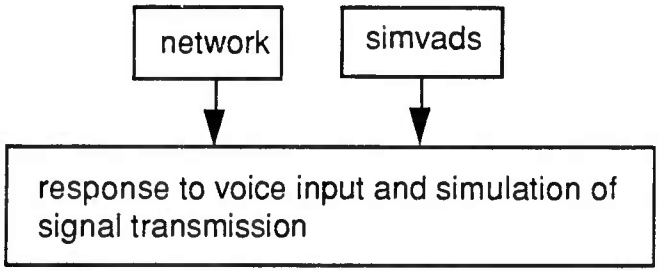

Figure 4.1-5 Response to voice input and simulation of signal transmission

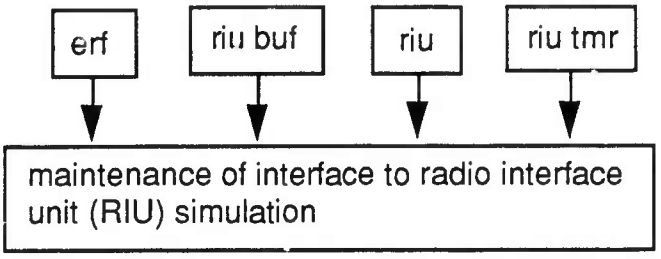

Figure 4.1-6 Maintenance of interface to RIU simulation

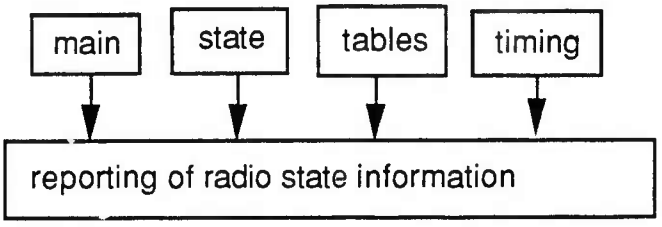

Figure 4.1-7 Reporting of radio state information

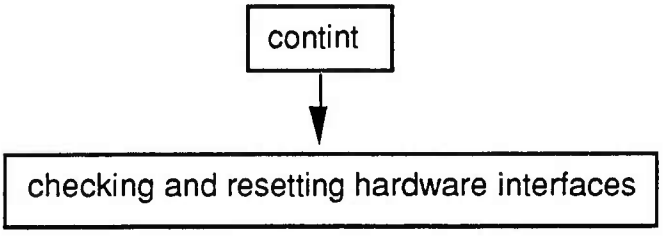

Figure 4.1-8 Checking and resetting hardware interfaces

4.1.1 Contint CSU

The Contint CSU performs controls initialization and maps the front panel controls to their corresponding functions (described in 4.1.2, "Controls CSU."

4.1.1.1 Contint CSU Design Specification/Constraints

Function Definition: controls_init
Call: cnt_init ("/simnet/data/sincgars/radio_dev.def")

Function Definition: controls_simul
Call: cnt_simul

Function Definition: controls_exit
Call: cnt_exit

Function Definition: controls_power_status
Return: (1)

Function Definition: chan_cue_0
Call: channel_control(0,CUE)

Function Definition: chan_man_0
Call: channel_control(0,MAN)

Function Definition: chan_1_0
Call: channel_control(0,1)

Function Definition: chan_2_0
Call: channel_control(0,2)

Function Definition: chan_3_0
Call: channel_control(0,3)

Function Definition: chan_4_0
Call: channel_control(0,4)

Function Definition: chan_5_0
Call: channel_control(0,5)

Function Definition: chan_6_0
Call: channel_control(0,6)

Function Definition: mode_sc_0
Call: mode_control(0,SC)

Function Definition: mode_fh_0
Call: mode_control(0,FH)

Function Definition: modem_fh_m_0
Call: mode_control(0,FH_M)

Function Definition: fctn_stby_0
Call: fctn_control(0,STBY)

Function Definition: fctn_tst_0
Call: fctn_control(0,TST)

Function Definition: fctn_ld_0
Call: fctn_control(0,LD)

Function Definition: fctn_sq_on_0
Call: fctn_control(0,SQ_ON)

Function Definition: fctn_sq_off_0
Call: fctn_control(0,SQ_OFF)

Function Definition: fctn_rxnt_0
Call: fctn_control(0,RXMT)

Function Definition: fctn_rem_0
Call: fctn_control(0,REM)

Function Definition: fctn_z_fh_0
Call: fctn_control(0,Z_FH)

Function Definition: fctn_off_0
Call: fctn_control(0,FCTN_OFF)

Function Definition: comsec_pt_0
Call: comsec_control(0,PT)

Function Definition: comsec_ct_0
Call: comsec_control(0,CT)

Function Definition: comsec_td_0
Call: comsec_control(0,TD)

Function Definition: comsec_rv_0
Call: comsec_control(0,RV)

Function Definition: comsec_z_0
Call: comsec_control(0,COM_Z)

Function Definition: rf_pwr_lo_0
Call: pwr_control(0,PWR_LO)

Function Definition: rf_pwr_m_0
Call: pwr_control(0,PWR_M)

Function Definition: rf_pwr_hi_0
Call: pwr_control(0,PWR_HI)

Function Definition: rf_pwr_pa_0
Call: pwr_control(0,PWR_PA)

Function Definition: comm_talk_a_0
Call: talk_control(0,positionCommander,A)

Function Definition: comm_talk_b_0
Call: talk_control(0,positionCommander,B)

Function Definition: comm_talk_int_0
Call: talk_control(0,positionCommander,INT)

Function Definition: loader_talk_a_0
Call: talk_control(0,positionLoader,A)

Function Definition: loader_talk_b_0
Call: talk_control(0,positionLoader,B)

Function Definition: loader_talk_int_0
Call: talk_control(0,positionLoader,INT)

Function Definition: driver_talk_a_0
Call: talk_control(0,positionDriver,A)

Function Definition: driver_talk_b_0
Call: talk_control(0,positionDriver,B)

Function Definition: driver_talk_int_0
Call: talk_control(0,positionDriver,INT)

Function Definition: gunner_talk_a_0
Call: talk_control(0,positionGunner,A)

Function Definition: gunner_talk_b_0
Call: talk_control(0,positionGunner,B)

Function Definition: gunner_talk_int_0
Call: talk_control(0,positionGunner,INT)

Function Definition: chan_cue_1
Call: channel_control(1,CUE)

Function Definition: chan_man_1
Call: channel_control(1,MAN)

Function Definition: chan_1_1
Call: channel_control(1,1)

Function Definition: chan_2_1
Call: channel_control(1,2)

Function Definition: chan_3_1
Call: channel_control(1,3)

Function Definition: chan_4_1
Call: channel_control(1,4)

Function Definition: chan_5_1
Call: channel_control(1,5)

Function Definition: chan_6_1
Call: channel_control(1,6)

Function Definition: mode_sc_1
Call: mode_control(1,SC)

Function Definition: mode_fh_1
Call: mode_control(1,FH)

Function Definition: modem_fh_m_1
Call: mode_control(1,FH_M)

Function Definition: fctn_stby_1
Call: fctn_control(1,STBY)

Function Definition: fctn_tst_1
Call: fctn_control(1,TST)

Function Definition: fctn_ld_1
Call: fctn_control(1,LD)

Function Definition: fctn_sq_on_1
Call: fctn_control(1,SQ_ON)

Function Definition: fctn_sq_off_1
Call: fctn_control(1,SQ_OFF)

Function Definition: fctn_rxnt_1
Call: fctn_control(1,RXMT)

Function Definition: fctn_rem_1
Call: fctn_control(1,REM)

Function Definition: fctn_z_fh_1
Call: fctn_control(1,Z_FH)

Function Definition: fctn_off_1
Call: fctn_control(1,FCTN_OFF)

Function Definition: comsec_pt_1
Call: comsec_control(1,PT)

Function Definition: comsec_ct_1
Call: comsec_control(1,CT)

Function Definition: comsec_td_1
Call: comsec_control(1,TD)

Function Definition: comsec_rv_1
Call: comsec_control(1,RV)

Function Definition: comsec_z_1
Call: comsec_control(1,COM_Z)

Function Definition: rf_pwr_lo_1
Call: pwr_control(1,PWR_LO)

Function Definition: rf_pwr_m_1
Call: pwr_control(1,PWR_M)

Function Definition: rf_pwr_hi_1
Call: pwr_control(1,PWR_HI)

Function Definition: rf_pwr_pa_1
Call: pwr_control(1,PWR_PA)

Function Definition: comm_talk_a_1
Call: talk_control(1,positionCommander,A)

Function Definition: comm_talk_b_1
Call: talk_control(1,positionCommander,B)

Function Definition: comm_talk_int_1
Call: talk_control(1,positionCommander,INT)

Function Definition: loader_talk_a_1
Call: talk_control(1,positionLoader,A)

Function Definition: loader_talk_b_1
Call: talk_control(1,positionLoader,B)

Function Definition: loader_talk_int_1
Call: talk_control(1,positionLoader,INT)

Function Definition: driver_talk_a_1
Call: talk_control(1,positionDriver,A)

Function Definition: driver_talk_b_1
Call: talk_control(1,positionDriver,B)

Function Definition: driver_talk_int_1
Call: talk_control(1,positionDriver,INT)

Function Definition: gunner_talk_a_1
Call: talk_control(1,positionGunner,A)

Function Definition: gunner_talk_b_1
Call: talk_control(1,positionGunner,B)

Function Definition: gunner_talk_int_1
Call: talk_control(1,positionGunner,INT)

Function Definition: chan_cue_2
Call: channel_control(2,CUE)

Function Definition: chan_man_2
Call: channel_control(2,MAN)

Function Definition: chan_1_2
Call: channel_control(2,1)

Function Definition: chan_2_2
Call: channel_control(2,2)

Function Definition: chan_3_2
Call: channel_control(2,3)

Function Definition: chan_4_2
Call: channel_control(2,4)

Function Definition: chan_5_2
Call: channel_control(2,5)

Function Definition: chan_6_2
Call: channel_control(2,6)

Function Definition: mode_sc_2
Call: mode_control(2,SC)

Function Definition: mode_fh_2
Call: mode_control(2,FH)

Function Definition: modem_fh_m_2
Call: mode_control(2,FH_M)

Function Definition: fctn_stby_2
Call: fctn_control(2,STBY)

Function Definition: fctn_tst_2
Call: fctn_control(2,TST)

Function Definition: fctn_ld_2
Call: fctn_control(2,LD)

Function Definition: fctn_sq_on_2
Call: fctn_control(2,SQ_ON)

Function Definition: fctn_sq_off_2
Call: fctn_control(2,SQ_OFF)

Function Definition: fctn_rxnt_2
Call: fctn_control(2,RXMT)

Function Definition: fctn_rem_2
Call: fctn_control(2,REM)

Function Definition: fctn_z_fh_2
Call: fctn_control(2,Z_FH)

Function Definition: fctn_off_2
Call: fctn_control(2,FCTN_OFF)

Function Definition: comsec_pt_2
Call: comsec_control(2,PT)

Function Definition: comsec_ct_2
Call: comsec_control(2,CT)

Function Definition: comsec_td_2
Call: comsec_control(2,TD)

Function Definition: comsec_rv_2
Call: comsec_control(2,RV)

Function Definition: comsec_z_2
Call: comsec_control(2,COM_Z)

Function Definition: rf_pwr_lo_2
Call: pwr_control(2,PWR_LO)

Function Definition: rf_pwr_m_2
Call: pwr_control(2,PWR_M)

Function Definition: rf_pwr_hi_2
Call: pwr_control(2,PWR_HI)

Function Definition: rf_pwr_pa_2
Call: pwr_control(2,PWR_PA)

Function Definition: comm_talk_a_2
Call: talk_control(2,positionCommander,A)

Function Definition: comm_talk_b_2
Call: talk_control(2,positionCommander,B)

Function Definition: comm_talk_int_2
Call: talk_control(2,positionCommander,INT)

Function Definition: loader_talk_a_2
Call: talk_control(2,positionLoader,A)

Function Definition: loader_talk_b_2
Call: talk_control(2,positionLoader,B)

Function Definition: loader_talk_int_2
Call: talk_control(2,positionLoader,INT)

Function Definition: driver_talk_a_2
Call: talk_control(2,positionDriver,A)

Function Definition: driver_talk_b_2
Call: talk_control(2,positionDriver,B)

Function Definition: driver_talk_int_2
Call: talk_control(2,positionDriver,INT)

Function Definition: gunner_talk_a_2
Call: talk_control(2,positionGunner,A)

Function Definition: gunner_talk_b_2
Call: talk_control(2,positionGunner,B)

Function Definition: gunner_talk_int_2
Call: talk_control(2,positionGunner,INT)

Function Definition: chan_cue_3
Call: channel_control(3,CUE)

Function Definition: chan_man_3
Call: channel_control(3,MAN)

Function Definition: chan_1_3
Call: channel_control(3,1)

Function Definition: chan_2_3
Call: channel_control(3,2)

Function Definition: chan_3_3
Call: channel_control(3,3)

Function Definition: chan_4_3
Call: channel_control(3,4)

Function Definition: chan_5_3
Call: channel_control(3,5)

Function Definition: chan_6_3
Call: channel_control(3,6)

Function Definition: mode_sc_3
Call: mode_control(3,SC)

Function Definition: mode_fh_3
Call: mode_control(3,FH)

Function Definition: modem_fh_m_3
Call: mode_control(3,FH_M)

Function Definition: fctn_stby_3
Call: fctn_control(3,STBY)

Function Definition: fctn_tst_3
Call: fctn_control(3,TST)

Function Definition: fctn_ld_3
Call: fctn_control(3,LD)

Function Definition: fctn_sq_on_3
Call: fctn_control(3,SQ_ON)

Function Definition: fctn_sq_off_3
Call: fctn_control(3,SQ_OFF)

Function Definition: fctn_rxnt_3
Call: fctn_control(3,RXMT)

Function Definition: fctn_rem_3
Call: fctn_control(3,REM)

Function Definition: fctn_z_fh_3
Call: fctn_control(3,Z_FH)

Function Definition: fctn_off_3
Call: fctn_control(3,FCTN_OFF)

Function Definition: comsec_pt_3
Call: comsec_control(3,PT)

Function Definition: comsec_ct_3
Call: comsec_control(3,CT)

Function Definition: comsec_td_3
Call: comsec_control(3,TD)

Function Definition: comsec_rv_3
Call: comsec_control(3,RV)

Function Definition: comsec_z_3
Call: comsec_control(3,COM_Z)

Function Definition: rf_pwr_lo_3
Call: pwr_control(3,PWR_LO)

Function Definition: rf_pwr_m_3
Call: pwr_control(3,PWR_M)

Function Definition: rf_pwr_hi_3
Call: pwr_control(3,PWR_HI)

Function Definition: rf_pwr_pa_3
Call: pwr_control(3,PWR_PA)

Function Definition: comm_talk_a_3
Call: talk_control(3,positionCommander,A)

Function Definition: comm_talk_b_3
Call: talk_control(3,positionCommander,B)

Function Definition: comm_talk_int_3
Call: talk_control(3,positionCommander,INT)

Function Definition: loader_talk_a_3
Call: talk_control(3,positionLoader,A)

Function Definition: loader_talk_b_3
Call: talk_control(3,positionLoader,B)

Function Definition: loader_talk_int_3
Call: talk_control(3,positionLoader,INT)

Function Definition: driver_talk_a_3
Call: talk_control(3,positionDriver,A)

Function Definition: driver_talk_b_3
Call: talk_control(3,positionDriver,B)

Function Definition: driver_talk_int_3
Call: talk_control(3,positionDriver,INT)

Function Definition: gunner_talk_a_3
Call: talk_control(3,positionGunner,A)

Function Definition: gunner_talk_b_3
Call: talk_control(3,positionGunner,B)

Function Definition: gunner_talk_int_3
Call: talk_control(3,positionGunner,INT)

Function Definition: chan_cue_4
Call: channel_control(4,CUE)

Function Definition: chan_man_4
Call: channel_control(4,MAN)

Function Definition: chan_1_4
Call: channel_control(4,1)

Function Definition: chan_2_4
Call: channel_control(4,2)

Function Definition: chan_3_4
Call: channel_control(4,3)

Function Definition: chan_4_4
Call: channel_control(4,4)

Function Definition: chan_5_4
Call: channel_control(4,5)

Function Definition: chan_6_4
Call: channel_control(4,6)

Function Definition: mode_sc_4
Call: mode_control(4,SC)

Function Definition: mode_fh_4
Call: mode_control(4,FH)

Function Definition: modem_fh_m_4
Call: mode_control(4,FH_M)

Function Definition: fctn_stby_4
Call: fctn_control(4,STBY)

Function Definition: fctn_tst_4
Call: fctn_control(4,TST)

Function Definition: fctn_ld_4
Call: fctn_control(4,LD)

Function Definition: fctn_sq_on_4
Call: fctn_control(4,SQ_ON)

Function Definition: fctn_sq_off_4
Call: fctn_control(4,SQ_OFF)

Function Definition: fctn_rxnt_4
Call: fctn_control(4,RXMT)

Function Definition: fctn_rem_4
Call: fctn_control(4,REM)

Function Definition: fctn_z_fh_4
Call: fctn_control(4,Z_FH)

Function Definition: fctn_off_4
Call: fctn_control(4,FCTN_OFF)

Function Definition: comsec_pt_4
Call: comsec_control(4,PT)

Function Definition: comsec_ct_4
Call: comsec_control(4,CT)

Function Definition: comsec_td_4
Call: comsec_control(4,TD)

Function Definition: comsec_rv_4
Call: comsec_control(4,RV)

Function Definition: comsec_z_4
Call: comsec_control(4,COM_Z)

Function Definition: rf_pwr_lo_4
Call: pwr_control(4,PWR_LO)

Function Definition: rf_pwr_m_4
Call: pwr_control(4,PWR_M)

Function Definition: rf_pwr_hi_4
Call: pwr_control(4,PWR_HI)

Function Definition: rf_pwr_pa_4
Call: pwr_control(4,PWR_PA)

Function Definition: comm_talk_a_4
Call: talk_control(4,positionCommander,A)

Function Definition: comm_talk_b_4
Call: talk_control(4,positionCommander,B)

Function Definition: comm_talk_int_4
Call: talk_control(4,positionCommander,INT)

Function Definition: loader_talk_a_4
Call: talk_control(4,positionLoader,A)

Function Definition: loader_talk_b_4
Call: talk_control(4,positionLoader,B)

Function Definition: loader_talk_int_4
Call: talk_control(4,positionLoader,INT)

Function Definition: driver_talk_a_4
Call: talk_control(4,positionDriver,A)

Function Definition: driver_talk_b_4
Call: talk_control(4,positionDriver,B)

Function Definition: driver_talk_int_4
Call: talk_control(4,positionDriver,INT)

Function Definition: gunner_talk_a_4
Call: talk_control(4,positionGunner,A)

Function Definition: gunner_talk_b_4
Call: talk_control(4,positionGunner,B)

Function Definition: gunner_talk_int_4
Call: talk_control(4,positionGunner,INT)

Function Definition: chan_cue_5
Call: channel_control(5,CUE)

Function Definition: chan_man_5
Call: channel_control(5,MAN)

Function Definition: chan_1_5
Call: channel_control(5,1)

Function Definition: chan_2_5
Call: channel_control(5,2)

Function Definition: chan_3_5
Call: channel_control(5,3)

Function Definition: chan_4_5
Call: channel_control(5,4)

Function Definition: chan_5_5
Call: channel_control(5,5)

Function Definition: chan_6_5
Call: channel_control(5,6)

Function Definition: mode_sc_5
Call: mode_control(5,SC)

Function Definition: mode_fh_5
Call: mode_control(5,FH)

Function Definition: modem_fh_m_5
Call: mode_control(5,FH_M)

Function Definition: fctn_stby_5
Call: fctn_control(5,STBY)

Function Definition: fctn_tst_5
Call: fctn_control(5,TST)

Function Definition: fctn_ld_5
Call: fctn_control(5,LD)

Function Definition: fctn_sq_on_5
Call: fctn_control(5,SQ_ON)

Function Definition: fctn_sq_off_5
Call: fctn_control(5,SQ_OFF)

Function Definition: fctn_rxnt_5
Call: fctn_control(5,RXMT)

Function Definition: fctn_rem_5
Call: fctn_control(5,REM)

Function Definition: fctn_z_fh_5
Call: fctn_control(5,Z_FH)

Function Definition: fctn_off_5
Call: fctn_control(5,FCTN_OFF)

Function Definition: comsec_pt_5
Call: comsec_control(5,PT)

Function Definition: comsec_ct_5
Call: comsec_control(5,CT)

Function Definition: comsec_td_5
Call: comsec_control(5,TD)

Function Definition: comsec_rv_5
Call: comsec_control(5,RV)

Function Definition: comsec_z_5
Call: comsec_control(5,COM_Z)

Function Definition: rf_pwr_lo_5
Call: pwr_control(5,PWR_LO)

Function Definition: rf_pwr_m_5
Call: pwr_control(5,PWR_M)

Function Definition: rf_pwr_hi_5
Call: pwr_control(5,PWR_HI)

Function Definition: rf_pwr_pa_5
Call: pwr_control(5,PWR_PA)

Function Definition: comm_talk_a_5
Call: talk_control(5,positionCommander,A)

Function Definition: comm_talk_b_5
Call: talk_control(5,positionCommander,B)

Function Definition: comm_talk_int_5
Call: talk_control(5,positionCommander,INT)

Function Definition: loader_talk_a_5
Call: talk_control(5,positionLoader,A)

Function Definition: loader_talk_b_5
Call: talk_control(5,positionLoader,B)

Function Definition: loader_talk_int_5
Call: talk_control(5,positionLoader,INT)

Function Definition: driver_talk_a_5
Call: talk_control(5,positionDriver,A)

Function Definition: driver_talk_b_5
Call: talk_control(5,positionDriver,B)

Function Definition: driver_talk_int_5
Call: talk_control(5,positionDriver,INT)

Function Definition: gunner_talk_a_5
Call: talk_control(5,positionGunner,A)

Function Definition: gunner_talk_b_5
Call: talk_control(5,positionGunner,B)

Function Definition: gunner_talk_int_5
Call: talk_control(5,positionGunner,INT)

Function Definition: chan_cue_6
Call: channel_control(6,CUE)

Function Definition: chan_man_6
Call: channel_control(6,MAN)

Function Definition: chan_1_6
Call: channel_control(6,1)

Function Definition: chan_2_6
Call: channel_control(6,2)

Function Definition: chan_3_6
Call: channel_control(6,3)

Function Definition: chan_4_6
Call: channel_control(6,4)

Function Definition: chan_5_6
Call: channel_control(6,5)

Function Definition: chan_6_6
Call: channel_control(6,6)

Function Definition: mode_sc_6
Call: mode_control(6,SC)

Function Definition: mode_fh_6
Call: mode_control(6,FH)

Function Definition: modem_fh_m_6
Call: mode_control(6,FH_M)

Function Definition: fctn_stby_6
Call: fctn_control(6,STBY)

Function Definition: fctn_tst_6
Call: fctn_control(6,TST)

Function Definition: fctn_ld_6
Call: fctn_control(6,LD)

Function Definition: fctn_sq_on_6
Call: fctn_control(6,SQ_ON)

Function Definition: fctn_sq_off_6
Call: fctn_control(6,SQ_OFF)

Function Definition: fctn_rxnt_6
Call: fctn_control(6,RXMT)

Function Definition: fctn_rem_6
Call: fctn_control(6,REM)

Function Definition: fctn_z_fh_6
Call: fctn_control(6,Z [illegible]I)

Function Definition: fctn_off_6
Call: fctn_control(6,FCTN_OFF)

Function Definition: comsec_pt_6
Call: comsec_control(6,PT)

Function Definition: comsec_ct_6
Call: comsec_control(6,CT)

Function Definition: comsec_td_6
Call: comsec_control(6,TD)

Function Definition: comsec_rv_6
Call: comsec_control(6,RV)

Function Definition: comsec_z_6
Call: comsec_control(6,COM_Z)

Function Definition: rf_pwr_lo_6
Call: pwr_control(6,PWR_LO)

Function Definition: rf_pwr_m_6
Call: pwr_control(6,PWR_M)

Function Definition: rf_pwr_hi_6
Call: pwr_control(6,PWR_HI)

Function Definition: rf_pwr_pa_6
Call: pwr_control(6,PWR_PA)

Function Definition: comm_talk_a_6
Call: talk_control(6,positionCommander,A)

Function Definition: comm_talk_b_6
Call: talk_control(6,positionCommander,B)

Function Definition: comm_talk_int_6
Call: talk_control(6,positionCommander,INT)

Function Definition: loader_talk_a_6
Call: talk_control(6,positionLoader,A)

Function Definition: loader_talk_b_6
Call: talk_control(6,positionLoader,B)

Function Definition: loader_talk_int_6
Call: talk_control(6,positionLoader,INT)

Function Definition: driver_talk_a_6
Call: talk_control(6,positionDriver,A)

Function Definition: driver_talk_b_6
Call: talk_control(6,positionDriver,B)

Function Definition: driver_talk_int_6
Call: talk_control(6,positionDriver,INT)

Function Definition: gunner_talk_a_6
Call: talk_control(6,positionGunner,A)

Function Definition: gunner_talk_b_6
Call: talk_control(6,positionGunner,B)

Function Definition: gunner_talk_int_6
Call: talk_control(6,positionGunner,INT)

Function Definition: chan_cue_7
Call: channel_control(7,CUE)

Function Definition: chan_man_7
Call: channel_control(7,MAN)

Function Definition: chan_1_7
Call: channel_control(7,1)

Function Definition: chan_2_7
Call: channel_control(7,2)

Function Definition: chan_3_7
Call: channel_control(7,3)

Function Definition: chan_4_7
Call: channel_control(7,4)

Function Definition: chan_5_7
Call: channel_control(7,5)

Function Definition: chan_6_7
Call: channel_control(7,6)

Function Definition: mode_sc_7
Call: mode_control(7,SC)

Function Definition: mode_fh_7
Call: mode_control(7,FH)

Function Definition: modem_fh_m_7
Call: mode_control(7,FH_M)

Function Definition: fctn_stby_7
Call: fctn_control(7,STBY)

Function Definition: fctn_tst_7
Call: fctn_control(7,TST)

Function Definition: fctn_ld_7
Call: fctn_control(7,LD)

Function Definition: fctn_sq_on_7
Call: fctn_control(7,SQ_ON)

Function Definition: fctn_sq_off_7
Call: fctn_control(7,SQ_OFF)

Function Definition: fctn_rxnt_7
Call: fctn_control(7,RXMT)

Function Definition: fctn_rem_7
Call: fctn_control(7,REM)

Function Definition: fctn_z_fh_7
Call: fctn_control(7,Z_FH)

Function Definition: fctn_off_7
Call: fctn_control(7,FCTN_OFF)

Function Definition: comsec_pt_7
Call: comsec_control(7,PT)

Function Definition: comsec_ct_7
Call: comsec_control(7,CT)

Function Definition: comsec_td_7
Call: comsec_control(7,TD)

Function Definition: comsec_rv_7
Call: comsec_control(7,RV)

Function Definition: comsec_z_7
Call: comsec_control(7,COM_Z)

Function Definition: rf_pwr_lo_7
Call: pwr_control(7,PWR_LO)

Function Definition: rf_pwr_m_7
Call: pwr_control(7,PWR_M)

Function Definition: rf_pwr_hi_7
Call: pwr_control(7,PWR_HI)

Function Definition: rf_pwr_pa_7
Call: pwr_control(7,PWR_PA)

Function Definition: comm_talk_a_7
Call: talk_control(7,positionCommander,A)

Function Definition: comm_talk_b_7
Call: talk_control(7,positionCommander,B)

Function Definition: comm_talk_int_7
Call: talk_control(7,positionCommander,INT)

Function Definition: loader_talk_a_7
Call: talk_control(7,positionLoader,A)

Function Definition: loader_talk_b_7
Call: talk_control(7,positionLoader,B)

Function Definition: loader_talk_int_7
Call: talk_control(7,positionLoader,INT)

Function Definition: driver_talk_a_7
Call: talk_control(7,positionDriver,A)

Function Definition: driver_talk_b_7
Call: talk_control(7,positionDriver,B)

Function Definition: driver_talk_int_7
Call: talk_control(7,positionDriver,INT)

Function Definition: gunner_talk_a_7
Call: talk_control(7,positionGunner,A)

Function Definition: gunner_talk_b_7
Call: talk_control(7,positionGunner,B)

Function Definition: gunner_talk_int_7
Call: talk_control(7,positionGunner,INT)

Function Definition: nil_proc

4.1.1.2 Contint CSU Design

The Contint CSU initializes the data structures and mapping.

4.1.2 Controls CSU

The various controls functions respond to individual switch positions. For example, for the mode control from squelch on to squelch off, the appropriate controls function would be called to disable squelching.

4.1.2.1 Controls CSU Design Specification/Constraints

Function Definition: SetHopInfo
Arguments: (fp, chan)
Call: SetTunerHopInfo(idcPortTable[fp - panelTable].i_radio, &hopinfo);

Function Definition: control_control
Arguments: (fp)
Calls:
```
clear_fp7(fp);
SetTunerFrequency(idcPortTable[fp - panelTable].i_radio,0,
    FALSE);
update_fp_power(fp, SMALL_RCVD_POWER, FALSE);
copy_in_fp7(fp->fp_display, " TST NYI");
SetTunerFrequency(idcPortTable[fp - panelTable].i_radio,fp-
    >internal.channel_freq[chan],chan == CUE);
copy_in_fp5(fp->fp_display, " COLD  ");
clear_fp5(fp);
SetHopInfo(fp, chan);
copy_in_fp5(fp->fp_display, " L7 L8  ");
copy_in_fp5(fp->fp_display, " L7  ");
set_update_flag(fp, TRUE);
copy_in_fp5(fp->fp_display, " L8    ");
```



```
                    set_update_flag(fp, TRUE);
                    clear_fp5(fp);
                    copy_in_fp7(fp->fp_display, " REMOTE ");
                    copy_in_fp7(fp->fp_display, " Z-FH   ");
                    SetTunerFrequency(idcPortTable[fp - panelTable].i_radio,0,
                        FALSE);
                    update_fp_power(fp, SMALL_RCVD_POWER, FALSE);
                    clear_lockouts(fp);
                    clear_channels(fp);
                    clear_tod(fp);
                    clear_comsec(fp);
                    cancel_delay(fp);
                    SET_MODE(fp, DEFAULT_MODE);
                    set_update_flag(fp, TRUE);

Function Definition: channel_control
    Arguments:      (idc_index,sw)
    Call:           control_control(fp);

Function Definition: mode_control
    Arguments:      (idc_index,sw)
    Call:           control_control(fp);

Function Definition: fctn_control
    Arguments:      (idc_index,sw)
    00188:          update_fp_power(fp, SMALL_RCVD_POWER, TRUE);
    Call:           control_control(fp);

Function Definition: comsec_control
    Arguments:      (idc_index,sw)

Function Definition: pwr_control
    Arguments:      (idc_index,swi)
    Calls:          print_at(&panelTable[idc_index],0,xchar(h_LVL2));
                    set_update_flag(&panelTable[idc_index],TRUE);
                    print_at(&panelTable[idc_index],0,xchar(h_LVL3));
                    set_update_flag(&panelTable[idc_index],TRUE);
                    print_at(&panelTable[idc_index],0,xchar(h_LVL5));
                    set_update_flag(&panelTable[idc_index],TRUE);
                    print_at(&panelTable[idc_index],0,xchar(h_LVL7));
                    set_update_flag(&panelTable[idc_index],TRUE);

Function Definition: talk_control
    Arguments:      (idc_index,who,sw)
    Calls:          update_talk_control(idc_index,who);
                    update_talk_control(pp - panelTable, who);

Function Definition: void key_radio
    Arguments:      (rp, new_key, speaker)
    Calls:          fprintf(astStream, "Transmitter keyed while receiving\n");
                    SendTransmitterPDU(rp, FALSE);
```

Function Definition: update_talk_control
Arguments: (idc_index,who)
Calls panelTable[idc_index].push_to_talk);
panelTable[idc_index].push_to_talk);
fflush(stdout);
key_radio(idcPortTable[idc_index].i_radio,R_KEYED_PREEMPT
,
idcPortTable[idc_index].i_radio->r_speaker);
desynchronize(idcPortTable[idc_index].i_radio);

Function Definition: soft_reset
Arguments: (idc_index,sw)
Call: set_light_val(reset_table[idc_index],sw);

Function Definition: set_beep
Arguments: (idc_index,sw)
Call: set_light_val(beep_table[idc_index],sw);

Function Definition: alert_operator
Arguments: (rp)
Call: beep_fp(rp->r_fps);

4.1.2.2 Controls CSU Design

The controls CSU has the functions mapped by contint.c via libcontrols to the front panel switches.

4.1.3 Data CSU

The data CSU contains the global data structures.

4.1.3.1 Data CSU Design Specification/Constraints

The Data file has no design specifications or constraints.

4.1.3.2 Data CSU Design

Not applicable.

4.1.4 Erf CSU

The erf CSU implements the electronic remote fill simulation for frequency hop mode. It permits presets on the radio to be loaded by a remote simulated radio.

4.1.4.1 Erf CSU Design Specification/Constraints

Function Definition: erf_receive_data
Arguments: (fp, src, snr, data, size)
Calls:
```
fprintf(astStream, "erf_receive_data: wrong size (%d)\n", size);
exit(1);
```

Function Definition: aerf_tick()
Calls:
```
copy_in_fp5(fp->fp_display, " L7 L8  ");
copy_in_fp5(fp->fp_display, " L7     ");
set_update_flag(fp, TRUE);
copy_in_fp5(fp->fp_display, " L8     ");
set_update_flag(fp, TRUE);
HLD_n_mode(fp, 0);
HLDln_mode(fp, 0);
fprintf(astStream, "erf_tick: Unknown ERF kind %d\n",
fp->internal.erf_msg.erf_kind);
key_radio(rp, 0, speakerUnknown);
HLD_n_mode(fp, 0);
HLDln_mode(fp, 0);
key_radio(rp, R_KEYED_ERF, speakerERF);
rt_transmit_erf(rp,&fp->internal.erf_msg,sizeof(fp-
    >internal.erf_msg),syncPreamble1);
rt_transmit_erf(rp,&fp->internal.erf_msg,sizeof(fp-
    >internal.erf_msg),fp->internal.erf_msg.erf_fragment
    ==fp->internal.erf_msg.erf_nfragments ?syncEOM
    :syncNormal);
```

Function Definition: erf_transmit_hopset
Arguments: (fp, hopset, tod_offset, lockout7, lockout8)

Function Definition: erf_transmit_lockout
Arguments: (fp, lockout)

4.1.4.2 Erf CSU Design

For the frequency hop mode, the erf simulates 1)transmission of electronic remote fill information from the front panel, and 2) reception, processing, and storage of electronic remote fill information with the simulated radio.

4.1.5 Fpt CSU

The fpt CSU is the basis of the front panel interface. It reads keypresses and updates the displays above the front panel's keypad.

4.1.5.1 Fpt CSU Design Specification/Constraints

Function Definition: u_char xchar
Arguments: (c)
Returns: val
-1

Function Definition:	set_mode
Arguments:	(fp, mode)
Calls:	sprintf(mode_bf, "strange mode %d", mode)
	mode_names[(int) mode]
	fprintf(astStream, "fpt %d: mode = %s.\n", fp - panelTable, s)
Function Definition:	open_fp
Arguments:	(pfn, fp)
Calls:	motif_panel(pfn, fp - panelTable)
Returns:	0,-1
Return:	1
Return:	-1
Return:	-1
Return:	-1,0
Function Definition:	close_fp
Arguments:	(pfd, pt)
Return:	-1
Return:	-1,pfd
Function Definition:	getcu
Arguments:	(kfd, ch)
Function Definition:	shift_r_fp
Arguments:	(dibuf, ch)
Function Definition:	shift_l_fp
Arguments:	(dibuf, ch)
Function Definition:	nfill_fp
Arguments:	(dibuf, num, ch)
Function Definition:	copy_in_fp
Arguments:	(dibuf, str)
Call:	ncopy_in_fp(dibuf, str, 0, 8)
Function Definition:	copy_in_fp5
Arguments:	(dibuf, str)
Call:	ncopy_in_fp(dibuf, str, 1, 5)
Function Definition:	copy_in_fp7
Arguments:	(dibuf, str)
Call:	ncopy_in_fp(dibuf, str, 1, 7)
Function Definition:	ncopy_in_fp
Arguments:	(dibuf, str, start, num)
Call:	xchar(str[i])
Function Definition:	copy_in
Arguments:	(dibuf, str)
Function Definition:	print_at
Arguments:	(fp, num, ch)

Function Definition: print_char_at
Arguments: (fp, num, ch)
Call: xchar(ch)

Function Definition: del_chars
Arguments: (dibuf, ch, rch)

Function Definition: print_fp
Arguments: (fp, ch)

Function Definition: print_char_fp
Arguments: (fp, ch)
Call: xchar(ch)

Function Definition: reset_cursor
Arguments: (fp)

Function Definition: set_cursor
Arguments: (fp, num)

Function Definition: erase_fp
Arguments: (fp, ch)
Call: print_at(fp, fp->internal.cursor, ch)

Function Definition: clear_fp7
Arguments: (fp)
Call: ncopy_in_fp(fp->fp_display, " ", 1, 7)

Function Definition: clear_fp5
Arguments: (fp)
Call: ncopy_in_fp(fp->fp_display, " ", 1, 5)

Function Definition: clear_tod
Arguments: (fp)

Function Definition: clear_channels
Arguments: (fp)

Function Definition: clear_lockouts
Arguments: (fp)

Function Definition: clear_comsec
Arguments: (fp)

Function Definition: clear_transec
Arguments: (fp)

Function Definition: set_tod_day
Arguments: (fp, channel, new_day)
Call: ftime(&now)

Function Definition: set_tod_sec
Arguments: (fp, channel, new_second)

Function Definition: standard_lockouts
Arguments: (fp)

Function Definition: standard_tod
Arguments: (fp)
Calls: time(&now)
set_tod_day(fp, i, 1)
set_tod_sec(fp, i, now % 86400)

Function Definition: standard_channels
Arguments: (fp)

Function Definition: standard_transec
Arguments: (fp)

Function Definition: standard_comsec
Arguments: (fp)

Function Definition: init_fp
Arguments: (fp, ch)
Calls: ftime(&now)
nfill_fp(fp->fp_display, 8, ch)
set_beep(fp-panelTable, FALSE)
reset_fps(fp)
standard_lockouts(fp)
standard_tod(fp)
standard_channels(fp)
standard_comsec(fp)
standard_transec(fp)
update_fp_power(fp, SMALL_RCVD_POWER, TRUE)

Function Definition: int update_fp
Arguments: (kfd_fp, fpbuf)
Call: display_codes(kfd_fp.u.motif_info, fpbuf)
Return: (write(kfd_fp.u.fd, fpbuf, 8))
Return: 8

Function Definition: set_update_flag
Arguments: (fp, val)

Function Definition: execute_fp
Arguments: (fp, ch)
Calls: fprintf(astStream, "fpt %d: execute %02x\n", fp - panelTable, ch)
freq_enter_mode(fp, ch);
erf_ofst_mode(fp, ch)
HLD___mode(fp, ch)
HFnnn_mode(fp, ch)
HLDl__mode(fp, ch)
HLnnn_mode(fp, ch)
STO___mode(fp, ch);
STOl__mode(fp, ch);
TOD_mode(fp, ch)
enter_TOD_mode(fp, ch)

hopset_display_mode(fp, ch)
hopset_enter_mode(fp, ch)
clr___mode(fp, ch)
clrl__mode(fp, ch)
fprintf(astStream, "execute_fp: Missing case for mode %d.\n",fp->internal.mode)
SET_MODE(fp, DEFAULT_MODE)
default_mode(fp, ch)

Function Definition: default_mode
Arguments: (fp, ch)
Calls: beep_fp(fp)
start_hopset_display_mode(fp)
start_freq_enter_mode(fp)
frequency_display_mode(fp)
start_erf_ofst_mode(fp)
HFnnn_mode(fp, ch)
HLnnn_mode(fp, ch)
start_TOD_mode(fp)
start_load(fp)
start_hopset_clear_mode(fp)

Function Definition: update_fp_power
Arguments: (fp, pwr, state)
Calls: print_char_at(fp, 0, ch)
set_update_flag(fp, TRUE)

Function Definition: update_transmit_power
Arguments: (fp, on)
Calls: print_char_at(fp, 0, ch)
set_update_flag(fp, TRUE)

Function Definition: start_hopset_clear_mode
Arguments: (fp)
Calls: copy_in_fp5(fp->fp_display, " CLR _ ")
set_update_flag(fp, TRUE)
SET_MODE(fp, CLR___MODE)

Function Definition: clr_n_mode
Arguments: (fp, digit)
Calls: copy_in_fp5(fp->fp_display, " CLR ")
print_at(fp, 5, digit)
set_update_flag(fp, TRUE)
beep_fp(fp)
SET_MODE(fp, DEFAULT_MODE)

Function Definition: clr___mode
Arguments: (fp, ch)
Calls: ch_to_digit(ch)
clear_fp7(fp)
set_update_flag(fp, TRUE)
set_delay(fp, TIME_BLINK, clr_n_mode, digit)
SET_MODE(fp, CLR_n_MODE)
print_char_at(fp, 4, (char) h_LOUT)


```
                    set_update_flag(fp, TRUE)
                    SET_MODE(fp, CLRl__MODE)

Function Definition clrln_mode
    Arguments:      (fp, digit)
    Calls:          copy_in_fp5(fp->fp_display, " CLR   ")
                    print_char_at(fp, 4, (char) h_LOUT)
                    print_char_at(fp, 5, '0' + digit)
                    set_update_flag(fp, TRUE)
                    beep_fp(fp)
                    SetHopInfo(fp, fp->channel_num)
                    SET_MODE(fp, DEFAULT_MODE)

Function Definition: clrl__mode
    Arguments:      (fp, ch)
    Calls:          ch_to_digit(ch)
                    clear_fp7(fp)
                    set_update_flag(fp, TRUE)
                    set_delay(fp, TIME_BLINK, clrln_mode, digit)
                    SET_MODE(fp, CLRln_MODE)

Function Definition: start_hopset_display_mode
    Arguments:      (fp)
    Calls:          SET_MODE(fp, HOPSET_DISPLAY_MODE)
                    hopset_display_mode(fp, FREQ)

Function Definition: hopset_display_mode
    Arguments:      (fp, ch)
    Calls:          start_hopset_enter_mode(fp)
                    clear_fp7(fp)
                    SET_MODE(fp, DEFAULT_MODE)
                    default_mode(fp, ch)
                    copy_in_fp5(fp->fp_display, s)
                    int_to_display3(fp->fp_display, n)
                    set_update_flag(fp, TRUE)

Function Definition: start_hopset_enter_mode
    Arguments:      (fp)
    Calls:          SET_MODE(fp, HOPSET_ENTER_MODE)
                    print_char_at(fp, 4, '_')
                    print_char_at(fp, 5, '_')
                    set_update_flag(fp, TRUE)
                    set_cursor(fp, 4)

Function Definition: hopset_enter_mode
    Arguments:      (fp, ch)
    Calls:          ch_to_digit(ch)
                    print_fp(fp, digit)
                    set_update_flag(fp, TRUE)
                    erase_fp(fp, xchar('_'))
                    set_update_flag(fp, TRUE)
                    display_to_int3(fp->fp_display)
                    copy_in_fp5(fp->fp_display, " STO _  ")
                    set_update_flag(fp, TRUE)
```



```
                    SET_MODE(fp, STO___MODE)
                    SET_MODE(fp, DEFAULT_MODE)
                    default_mode(fp, ch)

Function Definition: frequency_display_mode
    Arguments:      (fp)
    Calls:          copy_in_fp5(fp->fp_display, " 0     ")
                    int_to_display(fp->fp_display,fp->internal.channel_freq[fp-
                        >channel_num])
                    set_update_flag(fp, TRUE)

Function Definition: start_freq_enter_mode
    Arguments:      (fp)
    Calls:          SET_MODE(fp, FREQ_ENTER_MODE)
                    set_cursor(fp, 1)
                    copy_in_fp5(fp->fp_display, " FILL0  ")
                    print_char_at(fp, 5, '0')
                    print_char_at(fp, 5, 'C')
                    else print_char_at(fp, 5, '0' + (0x0f & fp->channel_num))
                    else int_to_display(fp->fp_display,
                    fp->internal.channel_freq[fp->channel_num])
                    set_update_flag(fp, TRUE)

Function Definition: cancel_mode
    Arguments:      (fp, mode_mask)
    Calls:          SET_MODE(fp, DEFAULT_MODE)
                    channel_control(fp - panelTable, fp->channel_num)

Function Definition: freq_enter_mode
    Arguments:      (fp, ch)
    Calls:          ch_to_digit(ch)
                    print_fp(fp, digit)
                    copy_in_fp5(fp->fp_display, " _____  ")
                    set_cursor(fp, 1)
                    else erase_fp(fp, xchar('_'))
                    set_update_flag(fp, TRUE)
                    SET_MODE(fp, DEFAULT_MODE)
                    cancel_delay(fp)
                    copy_in(temp, fp->fp_display)
                    del_chars(temp, xchar('_'), xchar(' '))
                    display_to_int(temp)
                    blink_display(fp)
                    clear_fp5(fp)
                    set_update_flag(fp, TRUE)
                    set_delay(fp, 7000, cancel_mode, 1 << FREQ_ENTER_MODE)
                    set_update_flag(fp, TRUE)

Function Definition: start_erf_ofst_mode
    Arguments:      (fp)
    Calls:          SET_MODE(fp, ERF_OFST_MODE)
                    set_cursor(fp, 1)
                    copy_in_fp5(fp->fp_display, ofst[fp-
                        >internal.ofst_index[channel]])
                    set_update_flag(fp, TRUE)
```


Function Definition: erf_ofst_mode
Arguments: (fp, ch)
Calls: clear_fp7(fp)
SET_MODE(fp, DEFAULT_MODE)
copy_in_fp5(fp->fp_display,ofst[fp->internal.ofst_index[fp->channel_num]])
set_update_flag(fp, TRUE)
set_update_flag(fp, TRUE)

Function Definition: start_load
Arguments: (fp)
Calls copy_in_fp5(fp->fp_display, " HLD _ ")
SET_MODE(fp, HLD___MODE)
set_update_flag(fp, TRUE)

Function Definition: HLDln_mode
Arguments: (fp, first_time)
Calls: copy_in_fp5(fp->fp_display, " HL___ ")
int_to_display3(fp->fp_display, fp->internal.holding_value)
set_update_flag(fp, TRUE)
beep_fp(fp)
blink_display(fp)
set_delay(fp, 7000, HLDln_mode, 0)
SET_MODE(fp, HLnnn_MODE)

Function Definition: HLDl__mode1
Arguments: (fp, lockout_num)
Calls: xchar('0' + lockout_num)
set_update_flag(fp, TRUE)
SET_MODE(fp, HLDln_MODE)
fp->internal.lockout[LKOUT(lockout_num)]
set_delay(fp, 500, HLDln_mode, 0)

Function Definition: HLDl _mode
Arguments: (fp, cl
Call: HLDl__mode1(fp, 1)
HLDl__mode1(fp, 2)
HLDl__mode1(fp, 3)
HLDl__mode1(fp, 4)
HLDl__mode1(fp, 5)
HLDl__mode1(fp, 6)
HLDl__mode1(fp, 7)
HLDl__mode1(fp, 8)

Function Definition: HLnnn_mode
Arguments: (fp, ch)
Calls: cancel_delay(fp)
erf_transmit_lockout(fp, fp->internal.holding_value)
copy_in_fp5(fp->fp_display, " SEND ")
set_update_flag(fp, TRUE)
SET_MODE(fp, ERF_SEND_MODE)
cancel_delay(fp)
copy_in_fp5(fp->fp_display, " STO _ ")

print_char_at(fp, 4, h_LOUT)
STOl__mode1(fp, lockout_num)
cancel_delay(fp)
SET_MODE(fp, DEFAULT_MODE)
default_mode(fp, ch)

Function Definition: STO___mode1
Arguments: (fp, channel)
Calls: print_char_at(fp, 5, '0' + channel)
set_update_flag(fp, TRUE)
blink_display(fp)
beep_fp(fp)
cancel_delay(fp)
SET_MODE(fp, DEFAULT_MODE)

Function Definition: STO___mode
Arguments: (fp, ch)
Calls: STO___mode1(fp, 1)
STO___mode1(fp, 2)
STO___mode1(fp, 3)
STO___mode1(fp, 4)
STO___mode1(fp, 5)
STO___mode1(fp, 6)

Function Definition: STOl__mode1
Arguments: (fp, lockout_num)
Calls: print_char_at(fp, 5, '0' + lockout_num)
set_update_flag(fp, TRUE)
blink_display(fp)
beep_fp(fp)
cancel_delay(fp)
SET_MODE(fp, DEFAULT_MODE)

Function Definition: STOl__mode
Arguments: (fp, ch)
Calls: STOl__mode1(fp, 1); break
STOl__mode1(fp, 2); break
STOl__mode1(fp, 3); break
STOl__mode1(fp, 4); break
STOl__mode1(fp, 5); break
STOl__mode1(fp, 6); break
STOl__mode1(fp, 7); break
case EIGHT: STOl__mode1(fp, 8); break

Function Definition: HLD_n_mode
Arguments: (fp, first_time)
Calls: copy_in_fp5(fp->fp_display, " HF___ ")
int_to_display3(fp->fp_display, fp->internal.holding_value)
set_update_flag(fp, TRUE)
beep_fp(fp)
blink_display(fp)
set_delay(fp, 7000, HLD_n_mode, 0)
SET_MODE(fp, HFnnn_MODE)

Function Definition: HLD___mode1
Arguments: (fp, channel)
Calls:
```
print_char_at(fp, 5, '0' + channel)
set_update_flag(fp, TRUE)
SET_MODE(fp, HLD_n_MODE)
set_delay(fp, 500, HLD_n_mode, 1)
```

Function Definition: HLD___mode
Arguments: (fp, ch)
Calls:
```
HLD___mode1(fp, 1); break
HLD___mode1(fp, 2); break
HLD___mode1(fp, 3); break
HLD___mode1(fp, 4); break
HLD___mode1(fp, 5); break
HLD___mode1(fp, 6); break
copy_in_fp5(fp->fp_display, " HLD _  ")
print_char_at(fp, 4, h_LOUT)
set_update_flag(fp, TRUE)
SET_MODE(fp, HLDl__MODE)
```

Function Definition: HFnnn_mode
Arguments: (fp, ch)
Calls:
```
cancel_delay(fp)
copy_in_fp5(fp->fp_display, " TOD  ")
SET_MODE(fp, DEFAULT_MODE)
set_update_flag(fp, TRUE)
erf_transmit_hopset(fp,fp->internal.holding_value,fp-
    >internal.holding_tod_offset,fp->internal.holding_lockout7,fp-
    >internal.holding_lockout8)
copy_in_fp5(fp->fp_display, " SEND   ")
set_update_flag(fp, TRUE)
SET_MODE(fp, ERF_SEND_MODE)
copy_in_fp5(fp->fp_display, " STO _  ")
set_update_flag(fp, TRUE)
SET_MODE(fp, STO___MODE)
SET_MODE(fp, DEFAULT_MODE)
default_mode(fp, ch)
```

Function Definition: channel_time
Arguments: (fp, channel, dayp, hourp, minutep, secondp, msecp)
Call:
```
ftime(&now)
```

Function Definition: update_TOD_mode
Arguments: (fp)
Calls:
```
channel_time(fp, channel, &day, &hour, &minute, &second,
    &msec)
clear_fp7(fp)
print_char_at(fp, 1, '0' + day / 10)
print_char_at(fp, 2, '0' + day % 10)
print_char_at(fp, 1, '0' + hour / 10)
print_char_at(fp, 2, '0' + hour % 10)
print_char_at(fp, 4, '0' + minute / 10)
print_char_at(fp, 5, '0' + minute % 10)
print_char_at(fp, 1, '0' + minute / 10)
```

```
                print_char_at(fp, 2, '0' + minute % 10)
                print_char_at(fp, 4, '0' + second / 10)
                print_char_at(fp, 5, '0' + second % 10)
                set_update_flag(fp, TRUE)
                set_delay(fp, 1000 - msec, update_TOD_mode)

Function Definition:  start_TOD_mode
    Arguments:        (fp)
    Calls:            SET_MODE(fp, TOD_DAYS_MODE)
                      update_TOD_mode(fp)

Function Definition:  TOD_mode
    Arguments:        (fp, ch)
    Calls:            ch_to_digit(ch)
                      update_TOD_mode(fp)
                      SET_MODE(fp, TOD_ENTER_DAYS_MODE)
                      copy_in_fp5(fp->fp_display, " __      ")
                      SET_MODE(fp, TOD_ENTER_HHMM_MODE)
                      copy_in_fp5(fp->fp_display, " __ __  ")
                      cancel_delay(fp)
                      set_cursor(fp, 1)
                      set_update_flag(fp, TRUE)
                      SET_MODE(fp, TOD_HHMM_MODE)
                      SET_MODE(fp, TOD_MMSS_MODE)
                      SET_MODE(fp, TOD_DAYS_MODE)
                      update_TOD_mode(fp)
                      cancel_delay(fp)
                      clear_fp7(fp)
                      SET_MODE(fp, DEFAULT_MODE)
                      default_mode(fp, ch)

Function Definition:  ch_to_digit
    Arguments:        (ch)
    Returns           1,2,3,4,5,6,7,8,9,0,-1

Function Definition:  enter_TOD_mode
    Arguments:        (fp, ch)
    Calls:            ch_to_digit(ch)
                      print_char_fp(fp, '0' + digit)
                      print_char_fp(fp, '0' + digit)
                      erase_fp(fp, xchar('_'))
                      SET_MODE(fp, TOD_DAYS_MODE)
                      set_tod_day(fp, fp->internal.tod_channel_num,fp->fp_display[1] *
                          10 +fp->fp_display[2])
                      SET_MODE(fp, TOD_HHMM_MODE)
                      set_tod_sec(fp, fp->internal.tod_channel_num,fp->fp_display[1] *
                          36000 +fp->fp_display[2] * 3600 +fp->fp_display[4] * 600
                          +fp->fp_display[5] * 60)
                      update_TOD_mode(fp)
                      blink_display(fp)
                      cancel_delay(fp)
                      clear_fp7(fp)
                      SET_MODE(fp, DEFAULT_MODE)
```


default_mode(fp, ch)
set_update_flag(fp, TRUE)

Function Definition: int is_zero
Arguments: (buf)
Returns (TRUE),(FALSE)

Function Definition: int display_to_int
Arguments: (buf)
Returns (0),(sum)

Function Definition: int display_to_int3
Arguments: (buf)
Returns (0),(sum)

Function Definition: int_to_display
Arguments: (buf, num)

Function Definition: int_to_display3
Arguments: (buf, num)

Function Definition: beep_fp
Arguments: (fp)
Calls: ftime(&etm)
set_beep(fp-panelTable, TRUE)

Function Definition: check_beep
Arguments: (fp)
Call: ftime(&etm)

Function Definition: set_delay
Arguments: (fp, delay, fn, arg)
Call: ftime(&etm)

Function Definition: cancel_delay
Arguments: (fp)

Function Definition: check_delay
Arguments: (fp)
Calls: ftime(&etm)
cancel_delay(fp)
(*fn)(fp, fp->internal.delay_arg)

Function Definition: fpt_receive_cue
Arguments: (fp)
Calls: copy_in_fp7(fp->fp_display, " CUE ")
set_update_flag(fp, TRUE)
beep_fp(fp)

Function Definition: reset_fps
Arguments: (fp)
Calls: ftime(&etm)
soft_reset(fp-panelTable, TRUE)

Function Definition: check_reset
Arguments: (fp)
Calls: ftime(&etm)
soft_reset(fp-panelTable, FALSE)

Function Definition: blink_display
Arguments: (fp)
Calls: ftime(&etm)
ncopy_in_fp(buf, " ", 0, 8)
update_fp(fp->internal.key_io, buf)
set_update_flag(fp, FALSE)

Function Definition: check_blink
Arguments: (fp)
Calls: ftime(&etm)
set_update_flag(fp, TRUE)

Function Definition: fp_show
Arguments: (argc, argv)
Calls: Rprintf("bad front panel index; must be 0 <= idx <= %d\n", MAX_IDC_CHANNELS-1)
Rprintf("Function: %s\n", fctn_names[(int) fp->fctn])
Rprintf(" Mode: %s\n", mode_names[(int) fp->mode])
Rprintf(" Power: %s\n", power_names[(int) fp->rf_power]);02055: Rprintf(" Comsec: %s\n", comsec_names[(int) fp->comsec])
Rprintf("Lockouts:")
Rprintf(" HL%3d", fp->internal.lockout[LKOUT(i)])
Rprintf("\n')
Rprintf(" Transec: TK%03d\n", fp->internal.transec)
Rprintf(" Channel: %d\n", fp->channel_num)
Rprintf(" Frequency Hopset L7 L8 COMSEC Day HHMM:SS\n")
channel_time(fp, channel, &day, &hour, &minute, &second, &msec)
sprintf(channel_name_buf, "chan %d", channel)
sprintf(hopset_buf, "HF%03d", fp->internal.hopset[channel])
Rprintf("%8s: %5d%3sKHz %s %s %s RK%03d %3d %02d%02d:%02d\n",channel_name,fp->internal.channel_freq[channel],ofst_names[fp->internal.ofst_index[channel]],hopset_string,fp->internal.lockout7[channel] ? "L7" : " ",fp->internal.lockout8[channel] ? "L8" : " ",fp->internal.comsec[channel],day, hour, minute, second)

Function Definition: fp_set
Arguments: (argc, argv)
Call: Rprintf("bad front panel index; must be 0 <= idx <= %d\n", MAX_IDC_CHANNELS-1)

4.1.5.2 Fpt CSU Design

The fpt CSU calls an open function, a close function, a keypad read function, and a set of display update functions, most of which are in controls.c.

4.1.6 Lrp CSU

The lrp (Longeley-Rice propagation model) CSU maintains the transmission loss table.

4.1.6.1 Lrp CSU Design Specification/Constraints

Function Definition: aknife
Arguments: (v2)
Returns: (6.02)
(6.02 + 9.11 * SQRT (v2) - 1.27 * v2)
(12.953 + 4.343 * LOG (v2))

Function Definition: fht
Arguments: (x, pk)
Return: retval

Function Definition: adiff
Arguments: (d, init)
Returns: 0.0
retval

Function Definition: alos
Arguments: (d, init)
Returns: 0.0
retval

Function Definition: h0f
Arguments: (r, et)
Return: retval

Function Definition: ahd
Arguments: (td)
Return: (a[i] + b[i] * td + c[i] * LOG (td))

Function Definition: ascat
Arguments: (d, init)
Returns: 0.0
1001.0
ss = (d - ad) / (d + ad)
(ahd (th * d) + 4.343 * LOG (47.7 * th * th * th * th * lrd.wn) - 0.1
* (lrd.ens - 301.0) * EXP (-th * d / 40.0e+3) + h0)

Function Definition: lrprop
Arguments: (d)

Function Definition: qtile
Arguments: (nn, aa, ir)
Return: q

Function Definition: zlsq1
Arguments: (pth, x1, x2, z0, zn)

Function Definition: dlthx
Arguments: (pth, x1, x2)
Returns: 0.0
retval

Function Definition: hzns

Function Definition: qlrpfl
Arguments: (pth, klimx, mdvarx)

Function Definition: qlrps
Arguments: (fmhz, zsys, en0, ipol, eps, sgm)

Function Definition: avar
Arguments: (zzt, zzl, zzc)
Return: retval

Function Definition: qlra
Arguments: (kst, klimx, mdvarx)

Function Definition: qerf
Arguments: (z)
Return: retval

Function Definition: qerfi
Arguments: (q)
Returns: retval
(-eno)
0

Function Definition: bld_trn
Arguments: (trn)
Returns: -eno
-eno
0

Function Definition: load_trn
Arguments: (trn, tfn)
Returns: -eno
-1
0

Function Definition: get_pfl
Arguments: (trn, pth)
Returns: -1

Function Definition: LoadTerrain

Function Definition: UpdateLoss
Arguments: (vpa, vpb, tp, pair)

4.1.6.2 Lrp CSU Design

The lrp CSU is a tree of functions invoked by a loop in main.c. It is a translation of the FORTRAN code described in NTIA Report 82-100, *A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode*.

4.1.7 Main CSU

The main CSU is the main loop and the entry point for the simulation.

4.1.7.1 Main CSU Design Specification/Constraints

Function Definition: main
Arguments: (argc, argv)
Calls:

```
perror("plock(PROCLOCK) failed");
fprintf(stderr, "Continuing anyway.\n");
fopen(errport, "a");
fprintf(stderr, "Using stderr, in place of %s, for
    astStream.\n",errport);
print_banner();
signal(SIGINT, exit_gracefully);
signal(SIGTERM, exit_gracefully);
tty_setup_modes();
atoi(optarg);
InitIDCs();
riu_init(& riuTable[idx]);
timing_init();
printf("Initializing fake simvad timer\n");
printf("tty_parser_init...\n");
tty_parser_init(command_table, "RADIO> ");
tty_tick();
simvads_restart();
```

Function Definition: static void ComputeLosses
Call: VehicleIDtoIndex(rp->r_radio_id.vehicle);

Function Definition: static int TickHandler

Function Definition: static void InitFakeSimvad

Function Definition: static int HandleAST
Calls:

```
timing_start(0);
timing_inter_ast(net_current_time(network_get_descriptor()));
timing_start(1);
timing_end(1);
timing_start(2);
```



```
                    riu_tick();
                    timing_end(2);
                    timing_start(3);
                    erf_tick();
                    timing_end(3);
                    timing_start(4);
                    simvads_service();
                    timing_end(4);
                    timing_start(5);
                    CheckFrontPanels();
                    timing_end(5);
                    timing_start(6);
                    (void) AssocTickAssocLayer();
                    (void) AssocTickAssocLayer(assocHandle);
                    timing_end(6);
                    fflush(astStream);
                    timing_end(0);

Function Definition: exit_gracefully
    Calls:          exit(0);
                    (void) setpri(sv_get_astpri());
                    motifp_exit();
                    simvads_uninit();
                    timing_uninit();
                    tty_restore_modes();
                    printf("\n");

Function Definition: print_banner
    Calls:          printf("\n");
                    printf("\n");
                    printf("HAVEQUICK Radio Simulation\n");
                    printf("SINCGARS Radio Simulation\n");
                    printf("%s\n", radio_version);
                    printf("BBN Systems and Technologies Corporation\n");
                    printf("Cambridge, MA, 02138\n");
                    printf("\n");
                    printf("\n");
                    printf("\n");
                    sleep(1);

Function Definition: Rprintf
    Arguments:      (va_alist)
    Calls:          va_start(args);
                    va_arg(args, char *);
                    setpri(sv_get_astpri());
                    vprintf(fmt, args);
                    setpri(pri);
    Return:         val;
```

Function Definition: xflush
Arguments: (f)
Calls: setpri(sv_get_astpri());
fflush(f);
setpri(pri);
Return: val;

Function Definition: main_need_simvads_restart

4.1.7.2 Main CSU Design

The main CSU is the main function. It calls various other functions in the simulation.

4.1.8 Motifp CSU

The motifp CSU is an X-windows simulation of the front panel, used for debugging.

4.1.8.1 Motifp CSU Design Specification/Constraints

Function Definition: motifp_display_codes
Arguments: (info, p)
Calls: SetArg(XmNlabelInsensitivePixmap, info->display_pmaps[c]);
XtSetValues(info->display_widgets[i], wargs, nargs);

Function Definition: static void keypad_button
Arguments: (w, data, cback)
Call: execute_fp(data->info->fp, data->data);

Function Definition: static void select_select
Arguments: (fn, w, data, cback)
Call: XqButtonListDeselectPos(w, cback->item_position);

Function Definition: static void power_select
Arguments: (w, data, cback)
Call: select_select(pwr_control, w, data, cback);

Function Definition: static void channel_select
Arguments: (w, data, cback)
Call: select_select(channel_control, w, data, cback);

Function Definition: static void mode_select
Arguments: (w, data, cback)
Call: select_select(mode_control, w, data, cback);

Function Definition: static void function_select
Arguments: (w, data, cback)
Call: select_select(fctn_control, w, data, cback);

Function Definition: static void comsec_select
Arguments: (w, data, cback)
Call: select_select(comsec_control, w, data, cback);

Function Definition: static void ptt_select
Arguments: (w, data, cback)
Calls: talk_control(info->fp - panelTable, crew_position,data->translations[prev_position - 1]);
talk_control(info->fp - panelTable, crew_position,data->translations[this_position - 1]);

Function Definition: static void quit_select
Arguments: (w, data, cback)
Call: exit_gracefully();

Function Definition: static Pixel mid_color
Arguments: (w, pixel1, pixel2)
Calls: XtDisplay(w);
XtScreen(w);
DefaultColormapOfScreen(scr);
XQueryColor(dpy, cmap, &color1);
XQueryColor(dpy, cmap, &color2);
XAllocColor(dpy, cmap, &color3);
Return: color3.pixel;

Function Definition: static Pixel led_color
Arguments: (w)
Calls: XtDisplay(w);
XtScreen(w);
DefaultColormapOfScreen(scr);
XAllocColor(dpy, cmap, &color);
Return: color.pixel;

Function Definition: static Widget create_selector
Arguments: (va_alist)
Calls: va_start(pvar);
va_arg(pvar, MOTIFP_INFOP);
va_arg(pvar, Widget);
va_arg(pvar, char *);
va_arg(pvar, XtCallbackProc);
va_arg(pvar, int);
va_arg(pvar, int);
va_arg(pvar, int);
va_end(pvar);
va_start(pvar);
(void) va_arg(pvar, MOTIFP_INFOP);
(void) va_arg(pvar, Widget);
(void) va_arg(pvar, char *);
(void) va_arg(pvar, XtCallbackProc);
(void) va_arg(pvar, int);
(void) va_arg(pvar, int);
SetArg(XmNorientation, XmVERTICAL);
SetArg(XmNisAligned, False);
XtCreateWidget(name, xmRowColumnWidgetClass, parent,wargs,


```
                    nargs);
                sprintf(label_name, "%s_label", name);
                sprintf(choices_name, "%s_choices", name);
                sprintf(frame_name, "%s_frame", name);
                XtCreateWidget(label_name, xmLabelGadgetClass,
selector,wargs,     nargs);
                XtCreateWidget(frame_name, xmFrameWidgetClass,selector,
                    wargs, nargs);
                va_arg(pvar, int);
                XmStringCreate(label, info->[illegible]harSet);
                va_end(pvar);
                SetArg(XmNitemCount, num_buttons);
                SetArg(XmNitems, items);
                SetArg(XmNselectionPolicy, XmSINGLE_SELECT);
                SetArg(XmNsingleSelectionCallback, callbacks);
                SetArg(XmNhilitePolicy, XmHILITE_WHEN_SELECTED);
                XqCreateButtonList(choices_frame, choices_name, wargs, nargs);
                XtFree(items);
                XtManageChild(choices);
                XtManageChildren(twidgets, ntwidgets);
Return:         selector;

Function Definition: static XmFontList FontListAppend
    Arguments:  (oldfl, font, charset)
    Returns:    XmFontListCreate(font, charset);
                XmFontListAdd(oldfl, font, charset);

Function Definition: static void InitFonts
    Arguments:  (info, dpy)

Function Definition: static Widget create_display
    Arguments:  (info, parent, name)
    Calls:      XtDisplay(parent);
                XtScreen(parent);
                RootWindowOfScreen(scr);
                SetArg(XmNbackground, &background);
                SetArg(XmNforeground, &foreground);
                XtGetValues(parent, wargs, nargs);
                led_color(parent);
                mid_color(parent, background, BlackPixelOfScreen(scr));
                SetArg(XmNbackground, background);
                SetArg(XmNforeground, foreground);
                XmCreateFrame(parent, "display_frame", wargs, nargs);
                SetArg(XmNorientation, XmHORIZONTAL);
                SetArg(XmNisAligned, False);
                SetArg(XmNbackground, background);
                XtCreateWidget(name, xmRowColumnWidgetClass,frame, wargs,
                    nargs);
                bzero(bmap, sizeof(bmap));
                XCreatePixmapFromBitmapData(dpy, win, bmap, 10, 16,red,
                    background, 8);
                XCreatePixmapFromBitmapData(dpy, win, bmap, 15, 24,red,
                    background, 8);
                SetArg(XmNlabelType, XmPIXMAP);
```

```
                    SetArg(XmNsensitive, False);
                    SetArg(XmNlabelInsensitivePixmap, info->display_pmaps[0]);
                    SetArg(XmNlabelPixmap, info->display_pmaps[1]);
                    SetArg(XmNbackground, background);
                    XtCreateWidget("", xmLabelGadgetClass, display,wargs, nargs);
                    XtManageChildren(info->display_widgets, 8);
                    XtManageChild(display);
    Return:         frame;

Function Definition: static Widget create_keypad
    Arguments:      (info, parent, name)
    Calls:          SetArg(XmNisAligned, False);
                    XtCreateWidget(name, xmRowColumnWidgetClass, parent,
    wargs,              nargs);
                    create_display(info, keypad_area,"display");
                    XtCreateWidget("keypad_frame", xmFrameWidgetClass,
                    keypad_area, wargs, nargs);
                    SetArg(XmNorientation, XmHORIZONTAL);
                    SetArg(XmNpacking, XmPACK_COLUMN);
                    SetArg(XmNnumColumns, 4);
                    SetArg(XmNadjustLast, False);
                    SetArg(XmNisAligned, False);
                    XtCreateWidget("keypad",
                        xmRowColumnWidgetClass,keypad_frame, wargs, nargs);
                    XmStringLtoRCreate(button_labels[i].l1, info->labelCharSet);
                    XmStringConcat(cs, XmStringSeparatorCreate(1));
                    XmStringConcat(cs,XmStringLtoRCreate(button_labels[i].l2,info-
                        >labelCharSet));
                    XmStringConcat(cs, XmStringSeparatorCreate(1));
                    XmStringConcat(cs,
                    XmStringLtoRCreate(button_labels[i].l3,nfo->digitCharSet));
                    SetArg(XmNlabelType, XmSTRING);
                    SetArg(XmNlabelString, cs);
                    SetArg(XmNdisarmCallback, callbacks);
                    XtCreateWidget(button_labels[i].name,xmPushButtonGadgetClass
                        keypad,wargs, nargs);
    ,
                    XtManageChildren(kwidgets, nkwidgets);
                    XtManageChild(keypad);
                    XtManageChildren(twidgets, ntwidgets);
    Return:         keypad_area;

Function Definition: static void CvtStringToWidget
    Arguments:      (args, nargs, fromVal, toVal)
    Calls:          XtParent((Widget) args[0].addr);

Function Definition: static MOTIFP_INFOP InitXt
    Arguments:      (argcp, argv, label, radio, fp, position, ptt)
    Calls:          XtCreateApplicationContext();
                    XtAppAddConverter(info->app, XtRString, XtRWindow,
                        CvtStringToWidget,cvt_args, XtNumber(cvt_args));
                    sprintf(panel_name, "radio_panel_%s", radio);
                    XtOpenDisplay(info->app, NULL, panel_name, "Radio",NULL, 0,
                        argcp, argv);
                    XtAppCreateShell(panel_name,
```



```
                    "Radio",applicationShellWidgetClass,info->dpy, NULL, 0);
                XtGetApplicationResources(info->frame, &info-
                    >application_data,application_resources,XtNumber
                    (application_resources),NULL, 0);
                InitFonts(info, XtDisplay(info->frame));
                XmCreateForm(info->frame, "panes", wargs, nargs);
                XmStringCreate(label, info->labelCharSet);
                SetArg(XmNlabelType, XmSTRING);
                SetArg(XmNlabelString, cs);
                XtCreateWidget("label", xmLabelGadgetClass, panes, wargs,
                    nargs);
                XtCreateWidget("sep", xmSeparatorGadgetClass, panes, wargs,
                    nargs);
                create_selector(info, panes, "power", power_select, 0, 0,"LO",
                    PWR_LO,"M",   PWR_M,"HI", PWR_HI,"PA",
                    PWR_PA,0);
                create_selector(info, panes, "channel", channel_select, 0, 0,"CUE",
                    CUE,"MAN", MAN,"1", 1,"2", 2,"3", 3,"4", 4,"5", 5,"6",
                    6,0);
                create_selector(info, panes, "mode", mode_select, 0, 0,"SC",
                    SC,"FH", FH,"FH-M", FH_M,0);
                create_selector(info, panes, "function", function_select, 0,
                    0,"STBY", STBY,"TST", TST,"LD", LD,"SQ ON",
                    SQ_ON,"SQ OFF", SQ_OFF,"RXMT", RXMT,"REM",
                    REM,"Z-FH", Z_FH,"OFF", FCTN_OFF,0);
                create_keypad(info, panes, "keypad");
                create_selector(info, panes, "comsec", comsec_select, 0, 0,"PT",
                    PT,"CT", CT,"TD", TD,"RV", RV,"COM-Z", COM_Z,0);
                create_selector(info, panes, "tc_ptt",
                    ptt_select,positionCommander, 0,"",-1,"A",A,"B",B,"INT",
                    INT,0);
                create_selector(info, panes, "l_ptt", ptt_select, positionLoader,
                    0,"",-1,"A",A,"B",B,"INT", INT,0);
                create_selector(info, panes, "g_ptt", ptt_select, positionGunner,
                    0,"",-1,"A",A,"B",B,"INT", INT,0);
                create_selector(info, panes, "d_ptt", ptt_select, positionDriver,
                    0,"",-1,"A",A,"B",B,"INT", INT,0);
                create_selector(info, panes, position == 0 ? "ptt0" : "ptt",ptt_select,
                    0, 0,"",-1,"Talk", A,0);
                create_selector(info, panes, "quit", quit_select, 0, 0,"RUN",
                    0,"PAUSE", 1,"QUIT", 2,0);
                XtManageChildren(widgets, nwidgets);
                XtManageChild(panes);
                XtRealizeWidget(info->frame);
                XFlush(info->dpy);
    Return:     info;

Function Definition: MOTIFP_INFOP motif_panel
    Arguments:  (name, idx)
    Calls:      XtToolkitInitialize();
                sprintf(riu_buf, "RIU %d", rp->r_riu);
                sprintf(ivis_buf, "IVIS %d", riuTable[rp->r_riu].u_ivis_index[0]);
                sprintf(ivis_buf, "no IVIS");
                sprintf(riu_buf, "no RIU");
```

```
                    sprintf(ivis_buf, "no IVIS");
                    sprintf(label, "RT-1523-X Radio %s, %s,
                        %s",RadioIDToString(rp->r_radio_id),riu_buf, ivis_buf);
                    sprintf(geom_buf, "+%d+%d", 100 - 20 * position, 20 * position);
                    iInitXt(&argc, argv, label, RadioIDToString(rp-
                        >r_radio_id),panelTable + idx, position, ptt);
    Return:         info;

Function Definition: motifp_input
    Arguments:      (info)
    Calls:          sprintf(fname, "/simnet/data/sincgars/panel%d.state",info->fp -
    panelTable);
                    fopen(fname, "r");
                    fscanf(f, "%d\n", &setting);
                    fclose(f);
                    XqButtonListSelectPos(w, info->current_settings[i], True);
                    XqButtonListHilitePos(w, info->current_settings[i]);
                    XtAppNextEvent(info->app, &event);
                    XtDispatchEvent(&event);

Function Definition: static save_settings
    Arguments:      (info)
    Calls:          sprintf(fname, "/simnet/data/sincgars/panel%d.state",
                    info->fp - panelTable);
                    fopen(fname, "w");
                    fprintf(f, "%d\n", info->current_settings[i]);
                    fclose(f);

Function Definition: motifp_exit
    Call:           save_settings(fp->internal.key_io.u.motif_info);
```

4.1.8.2 Motifp CSU Design

The motifp CSU is a set of X-windows functions which replace certain controls.c functions when the simulation is built for use with X-windows. It is *not* incorporated into the versions used at Fort Knox or Fort Monmouth, but is included for future use.

4.1.9 Network CSU

The network CSU is a set of functions designed to handle the sending and receiving of simulation network data on the Ethernet.

4.1.9.1 Network CSU Design Specification/Constraints

```
Function Definition: void InitSimNetwork

Function Definition: SignalVariant *AllocateBuffer
    Return:         buf;

Function Definition: void DeallocateBuffer
    Arguments:      (buf)
```


Function Definition: void saveSignalPDU
Arguments: (tp, bufp)
Calls: AllocateBuffer();
bcopy((char *) bufp, (char *) pdu,OFFSETA(SignalVariant, data)+ bufp->dataLength);

Function Definition: void ReadPDUs
Calls: AssocReceivePDU(&buf, &length, &group, &protocol, &primitive, &originator, &transID, &respondent);
AssocReceivePDU(assocHandle, &buf, &length, &group, &protocol, &primitive, &originator, &transID, &respondent);
fflush(astStream);
exit(0);
ProcessVehicleAppearancePDU(&(simPDU->variant.appearance));
VehicleDeactivated(VehicleIDtoIndex(simPDU->variant.deactivateRsp.vehicleID));
ProcessStatusChangePDU(&(dataPDU->variant.statusChange));
ProcessVehicleStatusPDU(&(dataPDU->variant.vehicleStatus));
ProcessTransmitterPDU(&(radioPDU->variant.transmitter));
ProcessSignalPDU(&(radioPDU->variant.signal));
ProcessAlertOperatorPDU(&(radioPDU->variant.alert));
ProcessIVISTransmitRequestPDU(&(ivisPDU->variant.transmitRequest));
ProcessSignalPDUs();

Function Definition: int VehicleIDtoIndex
Arguments: (vehicleID)
Calls: fprintf(stderr, "VehicleIDtoIndex: appearanceTable full\n");
exit(1);
Return: vidx;

Function Definition: void ProcessVehicleAppearancePDU
Arguments: (pdu)
Call: VehicleIDtoIndex(pdu->vehicleID);

Function Definition: void ProcessStatusChangePDU
Arguments: (pdu)

Function Definition: void ProcessVehicleStatusPDU
Arguments: (pdu)

Function Definition: void ProcessSignalPDU
Arguments: (pdu)
Calls: VehicleIDtoIndex(pdu->radio.vehicle);
saveSignalPDU(tp, pdu);

Function Definition: void ProcessTransmitterPDU
Arguments: (pdu)
Call: VehicleIDtoIndex(pdu->radio.vehicle);

Function Definition: int EqualFH
Arguments: (hi1p, hi2p)
Return: 0;
Return: 1;

Function Definition: void init_add_noise
Call: pow(10.0, -i/10.0);

Function Definition: int add_noise(a, b)
Calls: init_add_noise();
Return: a;
Return: a + s_dBm_sum[d];

Function Definition: FH_to_SC_noise
Arguments: (hip, frequency, power)
Return: power - 31;

Function Definition: FH_to_FH_noise
Arguents: (hip1, hip2, power)
Return: power - 31;

Function Definition: SC_to_FH_noise
Arguments: (frequency, hip, power)
Return: power - 31;

Function Definition: SC_to_SC_noise
Arguments: (freq1, freq2, power)
Return: power;
Return: power - 5 * (separation + 6);

Function Definition: desynchronize(rp)
Call: SendReceiverPDU(rp);

Function Definition: int finishPDU
Arguments: (rp, tp, snr, pdu)
Call: desynchronize(rp);
Return: 1;
Return: 0;
Calls: riu_receive_data(&riuTable[rp->r_riu],rp-radioTable,tp,snr,pdu->data,size);
simvads_data_frame(rp->r_voice_output);
erf_receive_data(rp->r_fps, tp, snr, (ERF_MSGP) pdu->data, size);
simvads_data_frame(rp->r_voice_output);
simvads_save_frame((short *) pdu->data,size,rp->r_voice_output);
Return: 0;

Function Definition: void ReceiveSC
Arguments: (rp)
Calls: add_noise(noise_power,FH_to_SC_noise(&tp->t_hopinfo,frequency,received_power));
add_noise(noise_power,SC_to_SC_noise(tp->t_frequency,frequency,received_power));
add_noise(noise_power, received_power);

```
                    desynchronize(rp);
                    finishPDU(rp, sync_tp, sync_snr, pdu);

Function Definition: void ReceiveSCSync
    Arguments:      (rp)
    Calls:          add_noise(noise_power,FH_to_SC_noise(&tp-
                        >t_hopinfo,frequency,received_power));
                    add_noise(noise_power,SC_to_SC_noise(tp-
                        >t_frequency,frequency,received_power));
                    add_noise(noise_power, received_power);
                    add_noise(noise_power, max_power);
                    SendReceiverPDU(rp);
                    SendReceiverPDU(rp);

Function Definition: void ReceiveCUE
    Arguments:      (rp)
    Calls:          add_noise(noise_power,FH_to_SC_noise(&tp-
                        >t_hopinfo,frequency,received_power));
                    add_noise(noise_power,SC_to_SC_noise(tp-
                        >t_frequency,frequency,received_power));
                    add_noise(noise_power, max_power);
                    add_noise(noise_power, received_power);
                    fpt_receive_cue(rp->r_fps);

Function Definition: void ReceiveFH
    Arguments:      (rp)
    Calls:          add_noise(noise_power,SC_to_FH_noise(tp->t_frequency,&rp-
                        >r_hopinfo,received_power));
                    add_noise(noise_power,FH_to_FH_noise(&tp->t_hopinfo,&rp-
                        >r_hopinfo,received_power));
                    add_noise(noise_power,FH_to_FH_noise(&tp->t_hopinfo,&rp-
                        >r_hopinfo,received_power));
                    desynchronize(rp);

Function Definition: void ReceiveFHSync
    Arguments:      (rp)
    Calls:          add_noise(noise_power,SC_to_FH_noise(tp-
                        >t_frequency,hopinfo,received_power));
                    add_noise(noise_power,FH_to_FH_noise(&tp-
                        >t_hopinfo,hopinfo,received_power));
                    add_noise(noise_power, received_power);
                    add_noise(noise_power, max_power);
                    SendReceiverPDU(rp);

Function Definition: resetTransmissions
    Call:           DeallocateBuffer(pdu);

Function Definition: ProcessSignalPDUs
    Calls:          ReceiveCUE(rp);
                    simvads_noise_frame(rp->r_voice_output);
                    ReceiveSCSync(rp);
                    ReceiveFHSync(rp);
                    ReceiveSC(rp);
```

```
                    ReceiveFH(rp);
                    resetTransmissions();

Function Definition: void ProcessAlertOperatorPDU
    Arguments:      (pdu)
    Call:           alert_operator(rp);

Function Definition: void ProcessIVISTransmitRequestPDU
    Arguments:      (pdu)

Function Definition: void SetupRadioPDU
    Arguments:      (pdu, kind)

Function Definition: void SetupSimulationPDU (pdu, kind)

Function Definition: void SetupIvisPDU
    Arguments:      (pdu, kind)

Function Definition: void SendTransmitterPDU
    Arguments:      (rp, periodic)
    Calls:          fprintf(astStream, "SendTransmitterPDU: AssocSendDatagram
                        %s\n",AssocError());
                    fprintf(astStream, "SendTransmitterPDU: AssocSendDatagram
                        %s\n",AssocError());

Function Definition: void SendReceiverPDU
    Arguments:      (rp)
    Calls:          fprintf(astStream, "SendReceiverPDU: AssocSendDatagram
                        %s\n",AssocError());
                    fprintf(astStream, "SendReceiverPDU: AssocSendDatagram
                        %s\n",AssocError());

Function Definition: void SendSignalPDU
    Arguments:      (rp, speaker, encoding, synchronization, duration, bitcount)
    Calls:          fprintf(astStream, "SendSignalPDU: AssocSendDatagram
                        %s\n",AssocError());
                    net_current_time(assocHandle);
                    fprintf(astStream, "SendSignalPDU: AssocSendDatagram
                        %s\n",AssocError());
                    VehicleIDtoIndex(siVar.radio.vehicle);
                    saveSignalPDU(tp, &siVar);
                    update_transmit_power(rp->r_fps, synchronization != syncEOM);

Function Definition: void SendIntercomPDU
    Arguments:      (vp, speaker, encoding, duration, bitcount)
    Calls:          net_current_time(networkInterface);
                    net_current_time(assocHandle);

Function Definition: void FlushPDUs

Function Definition: void SendVAPDU
    Arguments:      (vehicleID, location, marking)
    Calls:          net_current_time(networkInterface);
                    net_current_time(assocHandle);
```

```
                    fprintf(astStream, "SendVAPDU: AssocSendDatagram
                        %s\n",AssocError());

Function Definition: network_histogram_show
    Calls:          Rprintf("Histogram of PDUs received per tick:\n");
                    Rprintf("%d\t", i);
                    Rprintf(">%d\n", PDUS_PER_TICK_HISTOGRAM_SIZE - 2);
                    Rprintf("%d\t", pdus_per_tick_histogram[i]);
                    Rprintf("%d\n", pdus_per_tick_histogram
                        [PDUS_PER_TICK_HISTOGRAM_SIZE - 1]);

Function Definition: network_histogram_zero

Function Definition: network_show_cmcstats
    Calls:          net_stat_string(i, s);
                    Rprintf("%s %ld\n", s, stats[i]);
                    net_zero_statistics(networkInterface);
                    net_zero_statistics(assocHandle);
                    net_getaddr(networkInterface, &na);
                    net_getaddr(assocHandle, &na);
                    net_addr_bin_to_str(&na, eaddr);
                    net_addr_format_convert(eaddr, print_eaddr);
                    Rprintf("ethernet address:%s\n", print_eaddr);
    Return:         (networkInterface);
                    (assocHandle);
                    siVar.data;
    Calls:          fprintf(astStream, "%s to IVIS %s timed-
                        out\n",param,SimulationAddressToString(*respondent));
                    fprintf(astStream, "rt_receive_to_ivis: size %d > %d\n",size,
                        maxIVISMessageSize);exit(1);
                    bcopy((char *) msg, (char *) &irBuffer->variant.receive.message,
                        size);
                    fprintf(astStream, "rt_receive_to_ivis: AssocSendTransact
                        %s\n",AssocError());
                    fprintf(astStream, "rt_response_to_ivis: AssocSendTransact
                        %s\n",AssocError());
                    fprintf(astStream, "rt_transmit_data: size %d >
                        %d\n",size,MAX_DATA_BYTES);
                    exit(1);
                    bcopy(dg, siVar.data, size);
                    SendSignalPDU(&radioTable[radioNumber],speakerRIU,
                        signalData,synchronization,tickInterval,BITS(char) * size);
                    FlushPDUs();
                    bcopy((char *) erf_msg, siVar.data, size);
                    SendSignalPDU(rp,speakerERF,signalERF,synchronization,tickInt
                        erval,BITS(char) * size);
                    FlushPDUs();
                    sprintf(buf[buf_idx], "%d/%d", addr.site, addr.host);
    Return:         buf[buf_idx];
    Call:           sprintf(buf[buf_idx], "%d/%d/%d", vid.simulator.site,
                        vid.simulator.host,vid.vehicle);
    Return:         buf[buf_idx];
    Call:           sprintf(buf[buf_idx], "%d/%d/%d/%d", rid.vehicle.
                        simulator.site,rid.vehicle.simulator.host,
```

rid.vehicle.vehicle, rid.radio);
Return: buf[buf_idx];

4.1.9.2 Network CSU Design

The network CSU consists of an initialization function and a polling function.

4.1.10 Panelint CSU

The panelint CSU supports the front panel interface. It controls initialization of the IDC (inteactive device controller) board, which is responsible for sending messages to the simulation host about changes in the front panel. The panelint CSU also contains the front panel polling routine, which is invoked by the main loop.

4.1.10.1 Panelint CSU Design Specification/Constraints

Function Definition: InitIDCs
Calls: mem_assign_shared_memory()
idc_init()
controls_init();

Function Definition: PanelsInit
Arguments: (radio)
Call: set_update_flag(fp,TRUE);

Function Definition: CheckFrontPanels
Calls: check_delay(rp->r_fps)
motifp_input(rp->r_fps->internal.key_io.u.motif_info)
update_fp_power(rp->r_fps,rp->r_rcvd_power,FALSE)
set_update_flag(rp->r_fps,FALSE)
check_blink(rp->r_fps)
check_beep(rp->r_fps)
check_reset(rp->r_fps);

Function Definition: reset_fp
Call: reset_fps(rp->r_fps)

4.1.10.2 Panelint CSU Design

A change in the front panel switch causes the IDC board to send a message to the simulation host, which invokes the appropriate function in controls.c.

4.1.11 Param CSU

The param CSU controls aspects of the radio simulator's behavior.

4.1.11.1 Param CSU Design Specification/Constraints

```
Function Definition:  ConvertSimulationAddress
    Arguments:        (address, str)
    Return:           -1
    Return:           0;

Function Definition:  void ProcessParameters
    Arguments:        (filename)
    Calls:            exit(1)
                      fprintf(stderr, "ProcessParameters: AssocGetSimAddress failed --
                          \"%s\"\n", AssocError())
                      exit(1)
                      clear_tod(fp)
                      clear_channels(fp)
                      clear_lockouts(fp)
                      clear_comsec(fp)
                      clear_transec(fp)
                      ReportError(missingParameter)
                      ReportError(missingParameter)
                      ReportError(badRange, "Radio number",radioNumber, 0,
                          MAX_LOCAL_RADIOS-1)
                      ReportError("RADIO parameter must precede ATTACH")
                      ReportError("Duplicate ATTACH parameter")
                      ReportError(missingParameter)
                      ReportError(missingParameter)
                      ReportError(missingParameter)
                      ReportError("Bad marking char set: %c", chr)
                      ReportError(missingParameter)
                      ReportError("Bad marking char set: %c", chr)
                      ReportError(missingParameter)
                      ReportError("Incorrect attachment method: %s", str)
                      ReportError(missingParameter)
                      ReportError(missingParameter)
                      ReportError(missingParameter)
                      ReportError(missingParameter)
                      ReportError(missingParameter)
                      ReportError(badRange, "Radio number",radioNumber, 0,
                          MAX_LOCAL_RADIOS-1)
                      ReportError("Duplicate RADIO parameter")
                      ReportError(missingParameter)
                      ReportError(missingParameter)
                      ReportError(badVoiceChannel)
                      ReportError(badKeywordParameter, str)
                      ReportError(missingParameter)
                      ReportError("RADIO parameter must precede PRESET")
                      ReportError(missingParameter)
                      ReportError(badRange, "Channel number",channelNumber, 0,
                          NCHANNELS-1)
                      ReportError(badRange, "Frequency",frequency, 30000, 87750)
                      ReportError("PRESET frequency not a multiple of 250")
                      ReportError(missingParameter)
                      ReportError(badRange, "Channel number",channelNumber, 1, 6)
                      ReportError(badRange, "Hopset",hopset, 1, 999)
```



```
ReportError(missingParameter)
ReportError(badRange, "Lockout number",lockoutNumber, 1,
    numberFHLockouts)
ReportError(badRange, "Lockout",lockout, 1, 999)
ReportError(missingParameter)
ReportError(badRange, "Channel number",channelNumber, 0, 6)
ReportError(badRange, "Comsec", comsec, 1, 999)
ReportError(missingParameter)
ReportError(badRange, "Channel number",channelNumber, 1, 6)
ReportError(missingParameter)
ReportError(badRange, "Transec", transec, 1, 999)
ReportError(badKeywordParameter, str)
ReportError(missingParameter)
ReportError("Duplicate RIU parameter")
ReportError(missingParameter)
ReportError(badRange, "Radios/RIU", i, 0,
MAX_RADIOS_PER_RIU)
ReportError(missingParameter)
ReportError(badRange, "Radio number",radioNumber, 0,
    MAX_LOCAL_RADIOS-1)
ReportError("RADIO parameter must precede VEHICLE")
ReportError(missingParameter)
ReportError(badRange, "IVISes/RIU", i, 0,
MAX_IVISES_PER_RIU)
ReportError(missingParameter)
ReportError(badIvisNumber)
ReportError("IVIS parameter must precede RIU")
ReportError(missingParameter)
ReportError("Duplicate IVIS parameter")
ReportError(missingParameter)
ReportError(badAddress)
ReportError(missingParameter)
ReportError(missingParameter)
ReportError(badVehicleNumber)
ReportError("VEHICLE parameter must precede STATION")
ReportError(missingParameter)
ReportError("Duplicate station parameter")
ReportError(badVoiceChannel)
ReportError(pttConflict)
ReportError(missingParameter)
ReportError(missingParameter)
ReportError(badVehicleNumber)
ReportError("Duplicate VEHICLE parameter")
ReportError(missingParameter)
ReportError(missingParameter)
ReportError(badRange, "Radio number",0, radioNumber,
    MAX_LOCAL_RADIOS-1)
ReportError("RADIO parameter must precede VEHICLE")
ReportError("Duplicate VEHICLE parameter")
ReportError(badKeywordParameter, str)
ReportError(missingParameter)
ReportError(badVoiceChannel)
ReportError("Duplicate VOICECHANNEL parameter")
sprintf(vc->s_name, "sv%x:", voicechannelNumber)
```

sprintf(vc->s_name, "sv%x", voicechannelNumber)
sv_get_duration(SIXTEEN_KBITS_PER_SECOND)
sv_get_bitcount(SIXTEEN_KBITS_PER_SECOND)
sv_get_duration(THIRTYTWO_KBITS_PER_SECOND)
sv_get_bitcount(THIRTYTWO_KBITS_PER_SECOND)
sv_get_duration(SIXTEEN_KBITS_PER_SECOND)
sv_get_bitcount(SIXTEEN_KBITS_PER_SECOND)
ReportError("Unknown voice encoding")
ReportError("Parameter keyword not recognized")
ReportError("Line contains extraneous information")

Function Definition: static int ParseWord
Arguments: (pf, str)
Return: 0;
Return: 1;

Function Definition: static void ReportError
Arguments: (va_alist)
Calls: va_start(args)
va_arg(args, char *)
vfprintf(stderr, fmt, args)
va_end(args)
fprintf(stderr, "\n")
getc(f)
ungetc(ch, f)

4.1.11.2 Param CSU Design

The param CSU reads a parameter file (/simnet/data/sincgars/pars) at startup. The parameter file contains, for example, a definition of the number and type of radios attached to a given simulation host.

4.1.12 Radioidc CSU

The radioidc CSU initializes the connection between the radio simulator software and libidc, the idc library, which stores the functions for communications with the IDC board.

4.1.12.1 Radioidc CSU Design Specification/Constraints

Function Definition: idc_get_num_idcs
Call: return (NUM_IDCS);

Function Definition: idc_array_init

Function Definition: void idc_veh_spec_init

4.1.12.2 Radioidc CSU Design

Radioidc is a single initialization function.

4.1.13 Radiomem CSU

The radiomem CSU is a set of data structures connecting IDC boards to radio simulator software.

4.1.13.1 Radiomem CSU Design Specification/Constraints

Function Definition: mem_assign_other_ptrs

4.1.13.2 Radiomem CSU Design

Radiomem is a single function for mapping IDC boards to radio simulator software.

4.1.14 Riu_buf CSU

The riu buf CSU defines data buffers used in the riu simulation.

4.1.14.1 Riu_buf CSU Design Specification/Constraints

Function Definition: PUBLIC RIU_MSGP riu_buf_allocate
Arguments: (size)
Calls: fprintf(stderr, "riu_buf_allocate: Out of memory\n");
exit(1)
bzero(bf->m_data, size)
fprintf(astStream, "riu_buf_allocate: msg
%d@%6x\n",riu_buf_outstanding, bf)
Return: bf

Function Definition: PUBLIC void riu_buf_deallocate
Arguments: (bf)
Calls: fprintf(astStream, "riu_buf_dealloca: msg
%d@%6x\n",riu_buf_outstanding, bf)
FREE(bf)

4.1.14 2 Riu_buf CSU Design

Riu_buf has two functions, allocation and deallocation.

4.1.15 Riu_tmr CSU

The riu_tmr CSU controls the riu timing functions.

4.1.15.1 Riu_tmr CSU Design Specification/Constraints

Function Definition: riu_timer_init

Function Definition: riu_timer_periodic
Arguments: (elapsed)

Function Definition: riu_timer_cancel
Arguments: (if_what, what,if_arg1, arg1,if_arg2, arg2,if_arg3, arg3,if_arg4, arg4)

Function Definition: riu_timer_delay_
Arguments: (delay, what, nargs, arg1, arg2, arg3, arg4)
Calls: fprintf(stderr, "riu_timer_delay: ran out of memory\n");
exit(1);

4.1.15.2 Riu_mr CSU Design

Riu_tmr is called by riu.c.

4.1.16 Riu CSU

The riu CSU contains the actual riu simulation functions..

4.1.16.1 Riu CSU Design Specification/Constraints

Function Definition: riu_assert_fail
Arguments: (msg, line. file)
Calls: fprintf(astStream, "riu_assert failed at LINE %d, FILE %s: %s\n",line, file, msg)
exit(1)

Function Definition: riu_print_msg
Arguments: (riu, radio, label, msg, tag)
Calls: riu_radio_n(riu, radio)
sprintf(more, " src %s ber=%8f", RadioIDToString(id), msg->m_ber)
fprintf(astStream,"%*s: riu %d radio %d %d bytes%s%s%*s%s.\n",RIU_PRINT_LABEL_WIDTH, label,riu - riuTable, radio_n, msg->m_bytes,more,tag != NULL ? "\n" : "",tag != NULL ? RIU_PRINT_LABEL_WIDTH + 2 : 0,"",tag != NULL ? tag : "")

Function Definition: riu_print_pkt
Arguments: (riu, radio, label, pkt, src, power)
Calls: riu_radio_n(riu, radio)
sprintf(frag, " frg %d of %d", pkt->f_fragment+1, pkt->f_frags)
sprintf(xmitter, " src %s %ddBm", RadioIDToString(id), power)
fprintf(astStream, "%*s: riu %d radio %d%13s%s\n",RIU_PRINT_LABEL_WIDTH, label,riu - riuTable, radio_n, frag, xmitter)

Function Definition: riu_test
Arguments: (riu, serial, sender, radio_n)
Calls:
```
strncpy(pdu->message.messageID.originator.callSign.text,
    "TESTX",maxIVISCallSignLength)
riu_from_ivis(riu, riu->u_radio_index[radio_n], pdu)
riu_timer_delay4(3000 + riu_random_delay(7000),riu_test, riu,
    serial+1, sender,(radio_n + 1) % riu->u_num_radios)
```

Function Definition: riu_init
Arguments: (riu)
Calls:
```
riu_timer_init()
bzero((char *) s_zero_bits, sizeof(BITARRAY))
SETBIT(s_one_bits, i)
CLRBIT(s_zero_bits, i)
OFFSET(RIU_Datagram, type)
OFFSET(RIU_Datagram, sender)
riu_timer_delay4(3000, riu_test, riu, 0, &radioTable[riu-
    >u_radio_index[0]].r_radio_id.vehicle, 0)
riu_zero(riu)
```

Function Definition: riu_statistics
Arguments: (argc, argv)
Calls:
```
Rprintf("bad riu index; must be 0 <= idx <= %d\n", MAX_RIUS-
1)
Rprintf("RIU %d does not exist.\n", argv[0])
Rprintf("Statistics for RIU %d\n", argv[0])
Rprintf("\tAttached to radios:")
Rprintf(" none")
Rprintf(" %d", riu->u_radio_index[i])
Rprintf(".\n")
Rprintf("\tAttached to IVISes:")
Rprintf(" none")
Rprintf(" %d", riu->u_ivis_index[i])
Rprintf(".\n")
Rprintf("\tConnection from %s, serial #%d, at simulation time
    %d.\n",
VehicleIDToString(conn->c_remote),
Rprintf("\t%8d messages from IVIS.\n", riu->u_msgs_from_ivis)
Rprintf("\t%8d messages to IVIS.\n", riu->u_msgs_to_ivis)
Rprintf("\t%8d complete transmissions.\n", riu->u_transmissions)
Rprintf("\t%8d transmitted fragments.\n", riu-
    >u_transmitted_fragments)
Rprintf("\t%8d received fragments.\n", riu-
    >u_received_fragments)
Rprintf("\t%8d retransmissions.\n", riu->u_retransmissions)
Rprintf("\t%8d transmission preemptions.\n", riu->u_preemptions)
Rprintf("\t%8d reassembly failures.\n", riu-
    >u_reassembly_failures)
Rprintf("\t%8d messages with errors.\n", riu-
    >u_garbled_receptions)
Rprintf("\t%8d duplicates received.\n", riu->u_duplicates)
```

Function Definition: riu_zero
Arguments: (riu)

Function Definition: riu_zero_statistics
Call: riu_zero(riu)

Function Definition: riu_radio_n
Arguments: (riu, radio)
Return: radio_n
Return: -1

Function Definition: riu_random_delay
Arguments: (range)
Calls: double drand48()
Return: delay

Function Definition: riu_receive_data
Arguments: (riu, radio, source, snr, data, size)
Calls: riu_radio_n(riu, radio)
fprintf(astStream, "riu_receive_data: radio not attached\n")
exit(1)
riu_print_pkt(riu, radio, "riu_receive_data",pkt, source, snr)
fprintf(astStream, "%*s: reassembly failure\n",RIU_PRINT_LABEL_WIDTH, "riu_receive_data")
riu_buf_deallocate(msg)
riu_buf_allocate(pkt->f_frags * RIU_PKT_SIZE)
fprintf(astStream, "%*s: reassembly failure\n",RIU_PRINT_LABEL_WIDTH, "riu_receive_data")
fprintf(astStream, "%*s: reassembly failure\n",RIU_PRINT_LABEL_WIDTH, "riu_receive_data")
riu_buf_deallocate(msg)
fprintf(astStream, "riu_receive_data: incorrect packet size\n")
exit(1)
fprintf(astStream, "riu_receive_data: Message too big\n")
exit(1)
bcopy(pkt->f_data, dp + bytes_rcvd, pkt->f_bytes)

Function Definition: riu_receive_end
Arguments: (riu, radio, msg)
Calls: riu_print_msg(riu, radio, "riu_receive_end", msg,"unattached")
riu_buf_deallocate(msg)
riu_corrupt(msg)
FREE(detected_error)
riu_print_msg(riu, radio, "riu_receive_end", msg,"possible ACK w/errors...dropped")
riu_buf_deallocate(msg);
FREE(detected_error)
riu_print_msg(riu, radio, "riu_receive_end", msg,"error in sender...dropped")
riu_buf_deallocate(msg);
riu_find_remote(riu, dg->sender)
riu_merge(conn, msg, detected_error);
riu_print_msg(riu, radio, "riu_receive_end", msg,"with errors after merging")
riu_print_msg(riu, radio, "riu_receive_end", msg,"with errors, merged successfully")


```
                    riu_print_msg(riu, radio, "riu_receive_end", msg,"ACK")
                    riu_ack_to_ivis(riu, radio)
                    riu_buf_deallocate(msg)
                    riu_find_remote(riu, ((RIU_DatagramP) msg->m_data)->sender)
                    riu_flush_merge(conn)
                    sprintf(tag,"serial #%d to IVIS %s@%s",dg->serialNumber,
                        VehicleIDToString(local),SimulationAddressToStrin
                        g(to))
                    riu_print_msg(riu, radio, "riu_receive_end", msg, tag)
                    rt_receive_to_ivis(riu, radioTable[radio].r_radio_id,dg->network,
                        &dg->u.message.message,dg->u.message.message.length)
                    riu_print_msg(riu, radio, "riu_receive_end", msg,"duplicate")
                    riu_transmit_ack(riu, radio, conn->c_remote, local, dg->network,
                        dg->serialNumber)
                    riu_buf_deallocate(msg)

Function Definition: riu_from_ivis
    Arguments:       (riu, radio, pdu)
    Calls:           riu_ack_to_ivis(riu, radio);
                     riu_buf_allocate(size + OFFSET(RIU_Datagram, u.message))
                     bcopy((char *) pdu, (char *) &dg->u.message, size)
                     sprintf(tag, "serial #%d from IVIS %s",dg-
                         >serialNumber,RadioIDToString(pdu->radio))
                     riu_print_msg(riu, radio, "riu_from_ivis", msg, tag)
                     riu_transmit_start(riu, radio, msg)

Function Definition: riu_update_ber
    Arguments:       (riu, radio_n)
    Call:            riu_receive_end(riu, riu->u_radio_index[radio_n], msg)

Function Definition: riu_tick
    Calls:           riu_update_ber(riu, radio_n)
                     riu_timer_periodic(tickInterval)

Function Definition: double rint
    Arguments:       (d)
    Return:          floor(d + 0.5)

Function Definition: riu_corrupt
    Arguments:       (msg)
    Call:            double drand48()
    Return:          NULL
    Call:            SETBIT(bits, which_one)
    Return:          bits

Function Definition: riu_merge
    Arguments:       (conn, msg, detected_error)
    Calls:           !TSTBIT(detected_error, i)
                     !TSTBIT(&conn->c_merge_status, i)
                     CLRBIT(&conn->c_merge_status, i)
                     FREE(detected_error)
                     riu_buf_deallocate(msg)
    Return:          NULL
```


Call: riu_flush_merge(conn)
Return: msg

Function Definition: riu_flush_merge
Arguments: (conn)

Function Definition: riu_cancel_message
Arguments: (riu, msg)
Calls: riu_timer_cancel(true, riu_retry,true, (long) riu,false, 0,true, (long) msg,false, 0)
riu_timer_cancel(true, riu_transmit_start,true, (long) riu,false, 0,true, (long) msg,false, 0)
riu_timer_cancel(true, riu_transmit_continue,true, (long) riu,false, 0,true, (long) msg,false, 0)
riu_buf_deallocate(msg)

Function Definition: riu_ack_to_ivis
Arguments: (riu, radio)
Calls: fprintf(astStream, "%*s: ser#=%d sender=%s\n",RIU_PRINT_LABEL_WIDTH, "riu_ack_to_ivis",ack.serialNumber, RadioIDToString(ack.radio));rt_response_to_ivis(riu, &ack)
riu_cancel_message(riu, msg)
riu_cancel_message(riu, riu->u_transmit_msg[i])

Function Definition: Priu_transmit_ack
Arguments: (riu, radio, sender, recipient, network, serialNumber)
Calls: fprintf(astStream, "*s: ser#=%d sender=%s to recipient =%s\n",RIU_PRINT_LABEL_WIDTH,"riu_transmit_ack", serialNumber,IVIS_SystemIdentifierToString(recipient), IVIS_SystemIdentifierToString(sender))
riu_buf_allocate(size)
(RIU_DatagramP) msg->m_data
riu_timer_delay3(riu_random_delay(RIU_DELAY_RANGE), riu_transmit_start, riu, radio, msg)

Function Definition: riu_transmit_complete
Arguments: (riu, radio, msg, status)
Calls: iriu_radio_n(riu, radio)
key_radio(&radioTable[radio], 0, speakerUnknown)
riu_timer_delay3(riu_random_delay(RIU_DELAY_RANGE), riu_transmit_start, riu, radio, msg)
riu_buf_deallocate(msg)
riu_timer_delay3(s_retry_interval, riu_retry, riu, radio, msg)
riu_ack_to_ivis(riu, radio)

Function Definition: riu_transmit_continue
Arguments: (riu, radio, msg)
Calls: key_radio(&radioTable[radio], R_KEYED_DATA, speakerRIU)
rt_transmit_data(radio, (char *) &pkt, sizeof(pkt), syncPreamble1)
riu_timer_delay3(tickInterval, riu_transmit_continue, riu, radio, msg)
bcopy(dp + bytes_sent, pkt.f_data, pkt.f_bytes)
riu_print_pkt(riu, radio, "riu_transmit_continue",&pkt, NULL, 0)

fprintf(astStream,"riu_transmit_continue: m_fragments beyond m_frags\n")
exit(1)
rt_transmit_data(radio, (char *) &pkt, sizeof(pkt), syncNormal)
riu_timer_delay3(tickInterval, riu_transmit_continue,riu, radio, msg)
rt_transmit_data(radio, (char *) &pkt, sizeof(pkt), syncEOM)
riu_timer_delay4(tickInterval, riu_transmit_complete,riu, radio, msg, TRANSMIT_OK)
riu_transmit_complete(riu, radio, msg, TRANSMIT_PREEMPT)

Function Definition: riu_transmit_start
Arguments: (riu, radio, msg)
Calls: riu_radio_n(riu, radio)
riu_timer_delay2(tickInterval, riu_transmit_start, radio, msg)
sprintf(bf, "transmission %d", msg->m_transmit_count+1)
riu_print_msg(riu, radio, "riu_transmit_start", msg, bf)
riu_transmit_continue(riu, radio, msg)

Function Definition: riu_retry
Arguments: (riu, radio, msg)
Calls: riu_print_msg(riu, radio, "riu_retry", msg, "retransmit limit reached")
riu_ack_to_ivis(riu, radio)
riu_transmit_start(riu, radio, msg)

Function Definition: riu_find_remote
Arguments: (riu, remote)
Return: conn
Call: riu_flush_merge(conn)
Return: conn

4.1.16.2 Riu CSU Design

Riu is invoked by functions in network.c.

4.1.17 Rtu CSU

The rtu CSU is not used in the Fort Knox and Fort Monmouth versions. It provides certain UNIX functions needed by the radio simulation that are not available when the software is built for other computer systems.

4.1.17.1 Rtu CSU Design Specification/Constraints

Function Definition: setpri
Arguments: (pri)
Call: sv_get_astpri ();
Return: (set_pri);
Calls: sc_unlock ();
sc_lock ();
Return: ret_pri;

Function Definition:	getopt
Return:	EOF;
Return:	retval;
Return:	retval;
Return:	retval;
Calls:	fprintf (stderr,"%s: option requires argument -- %c\n", argv[0],argptr[1]);
Return:	'?';
Return:	'?';

Function Definition:	double drand48
Call:	rand ();
Return:	imed / base;

4.1.17.2 Rtu CSU Design

Rtu is a collection of functions normally found on UNIX systems.

4.1.18 Simvads CSU

The simvads CSU contains routines that handle the simvad (voice I/O) boards.

4.1.18.1 Simvads CSU Design Specification/Constraints

Function Definition:	simvads_init
Arguments:	(handler, busy_filename, noise_filename)
Calls:	printf ("simvads_init:opening busy file %s\n", busy_filename) noise_frame_buffer[idx][0], idx) close(fd) printf ("simvads_init:about fopen record file\n") record_file = fopen("record", "w") bzero(simstats, sizeof(struct simstats) * MAX_VOICE_CHANNELS)
Return:	(-1)
Return:	(0)

Function Definition:	broadcast_vc
Arguments:	(vc, pp, rp, real)
Calls:	SendSignalPDU(rp, vc->s_crew_station, vc->s_encoding,rp->r_keyed == 0 ? syncPreamble1 :real ? syncNormal :syncPreamble4,vc->s_duration, vc->s_bitcount)key_radio(rp, R_KEYED_VOICE, speaker) SendSignalPDU(rp, vc->s_crew_station, vc->s_encoding, syncEOM,vc->s_duration, vc->s_bitcount) key_radio(rp, R_KEYED_END, speaker) key_radio(rp, 0, speakerUnknown)

Function Definition: simvads_service
Calls:
GetBuffer()
bcopy(busy_frame_buffer[tickCount % BUSY_SIGNAL_FRAMES],buffer, 28 * sizeof(short))
main_need_simvads_restart()
broadcast_vc(vc, pp, rp, got_frame)
fwrite(buffer, 28, 2,record_file)
fclose(record_file)
broadcast_vc(vc, pp, rp, got_frame)
SendIntercomPDU(vp, vc->s_crew_station, vc->s_encoding,vc->s_duration, vc->s_bitcount)
broadcast_vc(vc, pp, rp, got_frame)
FlushPDUs()
fprintf(astStream, "%d %s==>%s\n", i, states[Ostate], states[vc->s_frame_state])

Function Definition: simvads_uninit
Call:
simvads_stop()

Function Definition: simvads_really_save_frame
Arguments:
(type, frame, size, channel)
Call:
bcopy(frame, vc->s_frames[vc->s_store_index].frame_buffer, size <= FRAMEBUFFER_SIZE ? size :FRAMEBUFFER_SIZE)

Function Definition: simvads_save_frame
Arguments:
(frame, size, channel)
Call:
simvads_really_save_frame(1, frame, size, channel)

Function Definition: simvads_data_frame
Arguments:
(channel)

Function Definition: simvads_noise_frame
Arguments:
(channel)
Call:
simvads_really_save_frame(3, noise_frame_buffer[frame],28 * sizeof(short), channel)

Function Definition: simvads_stop
Calls:
sv_ast_unsetup()
perror("simvads_stop:sv_restart")

Function Definition: int simvads_start
Calls:
printf("%s dummy simvad.\n", vc->s_name)
sprintf(errbuf, "simvads_start:error installing %s at 0x%x", vc->s_name, vc->s_address)
perror(errbuf)
printf("%s found at 0x%x.\n", vc->s_name, vc->s_address)
sprintf(errbuf, "simvads_start:can't open %s", vc->s_name)

```
                    perror(errbuf)
                    fprintf(stderr, "%s (desc = %d)", vc->s_name, vc->s_desc)
                    perror("simvads_start:sv_restart")
                    fflush (stdout)
    Return:         (-1)
    Call:           printf("simvads_start:no simvads found - can't init asts\n")
    Return:         (-1)
    Call:           fflush (stdout)
    Return:         (0)

Function Definition: int simvads_restart
    Calls:          simvads_stop()
                    simvads_start()

Function Definition: simvads_statistics
    Calls:          Rprintf("%s:\n", vc->s_name)
                    Rprintf("\tIO errors\t%d\n", simstats[i].IO_errors)
                    Rprintf("\tTicks with a particular number of frames:\n")
                    Rprintf("\t      frames")
                    Rprintf("\t%d", j)
                    Rprintf("\n")
                    Rprintf("\t      ticks(rcvd)")
                    Rprintf("\t%d", simstats[i].frames_rcvd_per_tick[j])
                    Rprintf("\n")
                    Rprintf("\t      ticks(sent)")
                    Rprintf("\t%d", simstats[i].frames_sent_per_tick[j])
                    Rprintf("\n")
                    Rprintf("\texcess frames
                        dropped\t%d\n",simstats[i].excess_output_dropped)
                    Rprintf("\tduplicated output
                        frames\t%d\n",simstats[i].duplicated_output)
                    Rprintf("\toutput FIFO
                        overflow\t%d\n",simstats[i].output_fifo_overflow)

Function Definition: simvads_zero_statistics
```

4.1.18.2 Simvads CSU Design

Simvads consists of a set of polling routines called by the main simulation every 26 milliseconds, an initialization routine, and a reset function.

4.1.19 State CSU

The state CSU is a set of functions that maintain and report the state of individual simulated radios.

4.1.19.1 State CSU Design Specification/Constraints

Function Definition: InitializeGlobalState

Function Definition: SetTunerFrequency
Arguments: (rp, frequency, cue)
Calls: BlockHandler("SetTunerFrequency")
SendTransmitterPDU (rp, 1)
UnblockHandler("SetTunerFrequency")

Function Definition: SetTunerHopInfo
Arguments: (rp, hopinfo)
Calls: BlockHandler("SetTunerHopInfo")
SendTransmitterPDU (rp, 1)
UnblockHandler("SetTunerHopInfo")

Function Definition: SetTransmitPower
Arguments: (rp, sw)
Calls: BlockHandler("SetTransmitPower")
UnblockHandler("SetTransmitPower")

Function Definition: AgeStates
Calls: BlockHandler("AgeStates")
UnblockHandler("AgeStates")
BlockHandler("AgeStates2")
UnblockHandler("AgeStates2")
BlockHandler("AgeStates3")
UnblockHandler("AgeStates3")

Function Definition: VehicleDeactivated
Arguments: (vidx)
Calls: fprintf(astStream, "Vehicle %s
deactivated.\n", VehicleIDToString(vehicleID))
fprintf(astStream, "Radio %d detached from vehicle %d.\n",i,
vehicleID)

Function Definition: BlockHandler
Arguments: (caller)
Call: sv_get_astpri()

Function Definition: UnblockHandler
Arguments: (caller)
Call: sc_unlock()

4.1.19.2 State CSU Design

State includes an initialization function and functions invoked each time a front panel changes state, for example in tuning, transmit power, or transmit mode.

4.1.20 Tables CSU

The tables CSU consists of functions used by the keyboard interface on the console.

4.1.20.1 Tables CSU Design Specification/Constraints

tables

```
Function Definition:  voicechannel_display
    Arguments:        (argc, argv)
    Calls:            Rprintf("bad voice channel number\n")
                      Rprintf("Voice Channel %d:\n", argv[0])
                      Rprintf("\t\tname = %s\n", vc->s_name)
                      Rprintf("\t\taddress = 0x%4x\n", vc->s_address)
                      Rprintf("\t\thardware exists\n")
                      Rprintf("\t\tin use\n")
                      Rprintf("\t\tinput radio number %d\n", vc->s_radio_input -
                            radioTable)
                      Rprintf("\t\tvehicle number %d crew station %d\n",vc-
                          >s_vehicle_input - vehicleTable, vc->s_crew_station)
                      Rprintf("\t\tencoding = %d\n", vc->s_encoding)
                      Rprintf("\t\tduration = %d\n", vc->s_duration)
                      Rprintf("\t\tbitcount = %d\n", vc->s_bitcount);

Function Definition:  tickcount_display
    Call:             Rprintf("elapsed ticks = %ld\n", tickCount);

Function Definition:  help
    Call:             Rprintf("try \"?\" for help\n");

Function Definition:  version
    Call:             Rprintf("Radio Simulator Version = %s\n", radio_version);

Function Definition:  DEFINE_TABLE
    Arguments:        (network_histogram_table)

Function Definition:  KEYWORD_SELECT
    Arguments:        ("Network Histogram Commands")

Function Definition:  KEYWORD
    Arguments:        ("show","- show PDU histogram")

Function Definition:  CALL
    Arguments:        (network_histogram_show)

Function Definition:  KEYWORD
    Arguments:        ("zero","- zero PDU histogram")

Function Definition:  CALL
    Arguments:        (network_histogram_zero)
```

```
Function Definition: DEFINE_TABLE
    Arguments:       (network_table)

Function Definition: KEYWORD_SELECT
    Arguments:       ("Network Commands")

Function Definition: KEYWORD
    Arguments:       ("getstats","- show cmc statistics")

Function Definition: CALL
    Arguments:       (network_show_cmcstats)

Function Definition: KEYWORD
    Arguments:       ("zerostats","- zero cmc statistics")

Function Definition: CALL
    Arguments:       (network_zero_cmcstats)

Function Definition: KEYWORD
    Arguments:       ("ethernetaddress","- display ethernet address")

Function Definition: CALL
    Arguments:       (network_geteaddr)

Function Definition: KEYWORD
    Arguments:       ("histogram","- network histogram commands")

Function Definition: DO_KEYWORD_TABLE
    Arguments:       (network_histogram_table)

Function Definition: DEFINE_TABLE
    Arguments:       (timing_table)

Function Definition: KEYWORD_SELECT
    Arguments:       ("Timing Commands")

Function Definition: KEYWORD
    Arguments:       ("show","- show timing data")

Function Definition: CALL
    Arguments:       (timing_display)

Function Definition: KEYWORD
    Arguments:       ("zero","- zero timing data")

Function Definition: CALL
    Arguments:       (timing_zero)

Function Definition: DEFINE_TABLE
    Arguments:       (simulation_table)

Function Definition: KEYWORD_SELECT
    Arguments:       ("Simulation Commands")
```

Function Definition: KEYWORD
Arguments: ("tickcount","- show tickcount")

Function Definition: CALL
Arguments: (tickcount_display)

Function Definition: KEYWORD
Arguments: ("voicechannel","- show voice channel data")

Function Definition: GETDECIMAL
Arguments: ("channel")

Function Definition: CALL
Arguments: (voicechannel_display)

Function Definition: DEFINE_TABLE
Arguments: (hardware_table)

Function Definition: KEYWORD_SELECT
Arguments: ("Hardware Commands")

Function Definition: KEYWORD
Arguments: ("resetsimvads","- reset ALL simvad cards")

Function Definition: CALL
Arguments: (simvads_restart)

Function Definition: KEYWORD
Arguments: ("simstats","- show statistics for simvads")

Function Definition: CALL
Arguments: (simvads_statistics)

Function Definition: KEYWORD
Arguments: ("resetfrontpanels","- reset ALL front panels")

Function Definition: CALL
Arguments: (reset_fp)

Function Definition: DEFINE_TABLE
Arguments: (riu_table)

Function Definition: KEYWORD_SELECT
Arguments: ("RIU Commands")

Function Definition: KEYWORD
Arguments: ("show","- show RIU statistics")

Function Definition: GETDECIMAL
Arguments: ("riu index")

Function Definition: CALL
Arguments: (riu_statistics)

```
Function Definition: KEYWORD
    Arguments:       ("zerostats","- zero RIU statistics")

Function Definition: CALL
    Arguments:       (riu_zero_statistics)

Function Definition: DEFINE_TABLE
    Arguments:       (fp_table)

Function Definition: KEYWORD_SELECT
    Arguments:       ("Front Panel Commands")

Function Definition: KEYWORD
    Arguments:       ("show","- show front panel state")

Function Definition: GETDECIMAL
    Arguments:       ("front panel index")

Function Definition: CALL
    Arguments:       (fp_show)

Function Definition: KEYWORD
    Arguments:       ("set","- set front panel parameter")

Function Definition: GETDECIMAL
    Arguments:       ("front panel index")

Function Definition: GETSTRING
    Arguments:       ("switch")

Function Definition: GETSTRING(
    Arguments:       "setting")

Function Definition: CALL
    Arguments:       (fp_set)

Function Definition: DEFINE_TABLE
    Arguments:       (simvads_table)

Function Definition: KEYWORD_SELECT
    Arguments:       ("SIMVADs Commands")

Function Definition: KEYWORD("show","- show simvad statistics")

Function Definition: CALL
    Arguments:       (simvads_statistics)

Function Definition: KEYWORD
    Arguments:       ("zero","- zero simvad statistics")

Function Definition: CALL
    Arguments:       (simvads_zero_statistics)
```

```
Function Definition:  DEFINE_TABLE
     Arguments:       (command_table)

Function Definition:  KEYWORD_SELECT
     Arguments:       ("Commands")

Function Definition:  KEYWORD
     Arguments:       ("network","- network functions")

Function Definition:  DO_KEYWORD_TABLE
     Arguments:       (network_table)

Function Definition:  KEYWORD
     Arguments:       ("simulation","- simulation status")

Function Definition:  DO_KEYWORD_TABLE
     Arguments:       (simulation_table)

Function Definition:  KEYWORD
     Arguments:       ("timing","- timing functions")

Function Definition:  DO_KEYWORD_TABLE
     Arguments:       (timing_table)

Function Definition:  KEYWORD
     Arguments:       ("riu","- RIU functions")

Function Definition:  DO_KEYWORD_TABLE
     Arguments:       (riu_table)

Function Definition:  KEYWORD
     Arguments:       ("fp","- front panel functions")

Function Definition:  DO_KEYWORD_TABLE
     Arguments:       (fp_table)

Function Definition:  KEYWORD
     Arguments:       ("simvads","- simvads functions")

Function Definition:  DO_KEYWORD_TABLE
     Arguments:       (simvads_table)

Function Definition:  KEYWORD
     Arguments:       ("hardware","- hardware functions")

Function Definition:  DO_KEYWORD_TABLE
     Arguments:       (hardware_table)

Function Definition:  KEYWORD
     Arguments:       ("help","- parser editor help")

Function Definition:  CALL
     Arguments:       (help)
```

Function Definition: KEYWORD
Arguments: ("version","- print simulator version information")

Function Definition: CALL
Arguments: (version)

Function Definition: KEYWORD
Arguments: ("exit","- exit program")

Function Definition: CALL
Arguments: (exit_gracefully)

4.1.20.2 Tables CSU Design

Tables functions are invoked by the main loop in main.c via the function tty_tick.

4.1.21 Timing CSU

The timing CSU functions do simulation performance timing, including recording and reporting performance.

4.1.21.1 Timing CSU Design Specification/Constraints

Function Definition: timing_start
Arguments: (which)
Call: net_current_time(network_get_descriptor());

Function Definition: timing_end
Arguments: (which)
Call: net_current_time(network_get_descriptor());

Function Definition: timing_display
Calls:
```
Rprintf("\n");
Rprintf("    %-40s%-10s%-10s%-10s (ms)\n", "", "  min", "  ave",
    "  max");
BlockHandler("timing_display");
UnblockHandler("timing_display");
Rprintf("    %-40s %-9.1f %-9.1f %-9.1f\n", ptv-
    >string,(float)tmin,(float)tsum / NUMSAVED,(float)tmax);
Rprintf("Inter-AST Interval Histogram\n");
Rprintf("0\t1\t2\t3-4\t5-8\t9-16\t>16\n");
Rprintf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
        inter_ast_histogram[0],inter_ast_histogram[1],
        inter_ast_histogram[2],inter_ast_histogram[3],
        inter_ast_histogram[4],inter_ast_histogram[5],
        inter_ast_histogram[6]);
```

Function Definition: timing_inter_ast
Arguments: (now)

Function Definition: timing_zero

Function Definition: timing_init
Call: timing_zero();

Function Definition: timing_uninit

4.1.21.2 Timing CSU Design

Timing functions are invoked from tables.c, main.c, network.c, simvads.c, and panelint.c.

4.1.22 Version CSU

The version CSU is a timestamp, reflecting when the current version of the software was assembled.

4.1.22.1 Version CSU Design Specification/Constraints

The file Version contains no function definitions.

4.1.22.2 Version CSU Design

Not applicable.

4.1.23 Vinfo CSU

The vinfo CSU is a set of data structures describing the vehicles with which radios are associated.

4.1.23.1 Vinfo CSU Design Specification/Constraints

The file Vinfo contains no function definitions.

4.1.23.2 Vinfo CSU Design

Vinfo contains no code, only data structure allocations. It is referenced in network.c.

5 CSCI DATA

Data .c contains the radio tables, front panel tables, and riu tables.

6 CSCI DATA FILES

There are no shared data files. The simulator reads its parameter file (discussed in 4.1.11) at start up time. It also looks to files for descriptions of the noise made when data is

transmitted (/simnet/data/sincgars/busy) and when squelch is disabled (/simnet/data/sincgars/noise).

6.1 DATA FILE TO CSC/CSU CROSS REFERENCE

This paragraph provides a mapping of each data file identified below to the CSCs and CSUs that use the data file.

6.2 NAME DATA FILE

Not applicable.

7 REQUIREMENTS TRACEABILITY

Not applicable.

8 NOTES

8.1 Acronyms/Abbreviations

CSC Computer Software Component
CSCI Computer Software Configuration Item
CSU Computer Software Unit
CDR Critical Design Review
CI Configuration Item
FCA Functional Configuration Audit
IDD Interface Design Document
NDS Non-Developmental Software
PCA Physical Configuration Audit
PDR Preliminary Design Review
SDD Software Design Document
SRS Software Requirements Specification

8.2 Notation

Not applicable.

This page is intentionally blank.