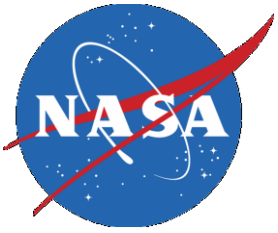
WVHTC FOUNDATION SM

Assessment Environment for Complex Systems Software Guide

Document ID: WVHTF-AECS-SG-F003-130116
Release Date: 16 January 2013
NASA-CR-2014-216654

Prepared For:



Prepared By:
West Virginia High Technology Consortium Foundation
Mission Systems Group
1000 Technology Drive, Suite 1000
Fairmont, WV 26554
http://www.wvhtf.org/

CONTENTS

FIGURES

TABLES

1 Scope

1.1 Identification

This Software Guide (SG) describes the software developed to test the Assessment Environment for Complex Systems (AECS) by the West Virginia High Technology Consortium (WVHTC) Foundation's Mission Systems Group (MSG) for the National Aeronautics and Space Administration (NASA) Aeronautics Research Mission Directorate (ARMD). This software is referred to as the AECS Test Project throughout the remainder of this document. AECS provides a framework for developing, simulating, testing, and analyzing modern avionics systems within an Integrated Modular Avionics (IMA) architecture. The purpose of the AECS Test Project is twofold. First, it provides a means to test the AECS hardware and system developed by MSG. Second, it provides an example project upon which future AECS research may be based. This Software Guide fully describes building, installing, and executing the AECS Test Project as well as its architecture and design.

The design of the AECS hardware is described in the *AECS Hardware Guide.* Instructions on how to configure, build and use the AECS are described in the User's Guide. Sample AECS software, developed by the WVHTC Foundation, is presented in the *AECS Software Guide*. The *AECS Hardware Guide*, *AECS User's Guide*, and *AECS Software Guide* are authored by MSG. The requirements set forth for AECS are presented in the *Statement of Work for the Assessment Environment for Complex Systems* authored by NASA Dryden Flight Research Center (DFRC).

The intended audience for this document includes software engineers, hardware engineers, project managers, and quality assurance personnel from WVHTC Foundation (the suppliers of the software), NASA (the customer), and future researchers (users of the software). Readers are assumed to have general knowledge in the field of real-time, embedded computer software development.

1.2 System Overview

The ARMD is responsible for achieving NASA Strategic Goal 3E, "Advance knowledge in the fundamental disciplines of aeronautics and develop technologies for safer aircraft and higher capacity airspace systems." To achieve this goal, ARMD conducts research across a breadth of core aeronautics competencies; research in key areas related to the development of advanced aircraft technologies and systems; and research that supports the Next Generation Air Transportation System (NextGen), in partnership with the Joint Planning and Development Office (JPDO).

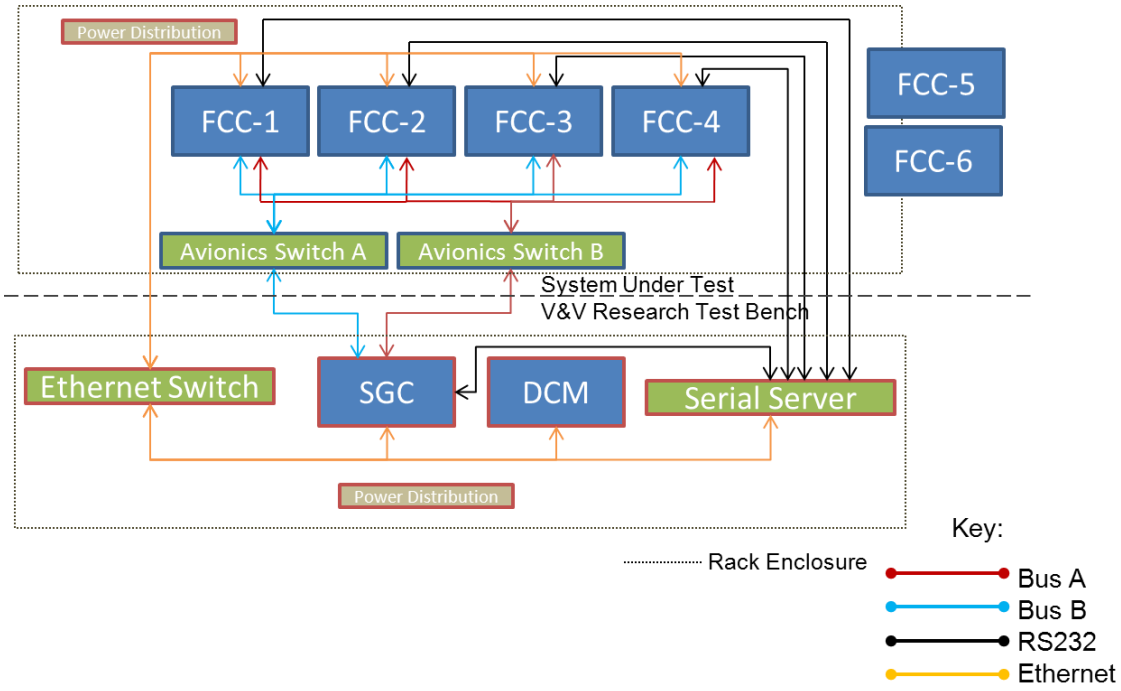
The JPDO has identified a lack of validation and verification methods needed to certify the safety of complex, NextGen systems as a critical gap. Validation confirms that NextGen systems and requirements meet the expectations of the customer of the system or other stakeholders throughout the system life cycle. Verification confirms that NextGen systems comply with requirements as the systems are designed, implemented, integrated, deployed, maintained, and retired. As a JPDO member agency, NASA is targeting this critical gap through the Verification and Validation of Flight-Critical Systems (VVFCS) initiative in ARMD's Aviation Safety Program. Research within VVFCS focuses on four

topics: (1) argument-based safety assurance, (2) distributed systems, (3) authority and autonomy, and (4) software intensive systems.

The Verification and Validation methods developed as a result of this research need to be evaluated and demonstrated in a realistic system. The system, called the Assessment Environment (AE), is comprised of a representative flight critical system and a system to provide test functions (input generation, output recording, etc.) commonly referred to as a test bench. The creation and use of the AE will allow researchers to evaluate research products beyond a purely theoretical environment.

The WVHTC Foundation has been tasked to build the Assessment Environment. The design will be based on a contemporary type of flight system known as an IMA architecture. This type of system architecture is currently being used in the latest aircraft. The integrated modular avionics architecture replaces boxes containing individual processors and applications with a single box performing all the functions on one or more processor boards. The AE supports both AFDX and TTE avionics buses.

Figure 1: AECS Architecture



1.3 Acronyms and Abbreviations

Appendix A lists the acronyms and abbreviations used throughout this document.

1.4 Document Overview

Section 2 lists other documents referenced from this Software Guide. It will be necessary to reference these additional documents for a complete understanding of certain features of the AECS Test Project.

Section 3 provides an overview of the AECS Test Project. This section provides an introduction to the components of the AECS Test Project. It defines key terms used throughout the remainder of the document.

Section 4 details the configuration management of the AECS Test Project. This section provides instructions on obtaining the AECS Test Project from the AECS source code repository.

Section 5 describes the installation and execution of the AECS Test Project. This section details installation of each AECS Test Project component. It also discusses starting of the AECS Test Project and monitoring via the monitor and control application.

Section 6 discusses the AECS Test Project software architecture. This section expands on the information presented in the overview. It provides additional details on the AECS Test Project and requires the reader to have a greater understanding of AFDX and the VxWorks 653 Operating System (OS).

Appendix B provides basic information on several additional projects created to aid in initial testing of the AE. These projects are simpler than the full AECS Test Project and are good starting points for someone unfamiliar with AECS, 653, AFDX, and TTE. As there are no formal requirements associated with these projects, they are documented in an Appendix.

2 Referenced Documents

2.1 Government Documents

2.1.1 Standards and Specifications

None.

2.1.2 Other Publications

Table 1: Other Government Publications

ID/Version	Description
Version 9.0, July 29, 2010	Statement of Work for the Assessment Environment for Complex Systems

2.2 Non-Government Documents

2.2.1 Standards and Specifications

None.

2.2.2 Other Publications

Table 2: Other Non-Government Publications

ID/Version	Description
WVHTF-AECS-HG	Assessment Environment for Complex Systems Hardware Guide (HG)
WVHTF-AECS-UG	Assessment Environment for Complex Systems User's Guide (UG)
WVHTF-AECS-TSP	Assessment Environment for Complex Systems Test Setup and Procedures (TSP)
WVHTF-AECS-TR	Assessment Environment for Complex Systems Test Results (TR)
V01.00 Rev. B, April 2011	AIM AFDX Lab-Switch 8/16/24 User Manual
DOC-16046-ND-01, 26 Oct 07	WindRiver Configuration and Build Guide for VxWorks 653, 2.2
DOC-15951-ND-02, 29 Oct 07	WindRiver Programmer's Guide for VxWorks 653, 2.2
DOC-15898-ND-00, 28 Sep 06	WindRiver Workbench Host Shell User's Guide for VxWorks, 2.6
DOC-16132-ND-00, 11 Oct 07	WindRiver Workbench User's Guide for VxWorks 653, 2.6.1
DOC-15844-ND-00, 1 Oct 06	WindRiver Workbench User's Guide for VxWorks, 2.6
D-TTE-G-01-002	TTE Build: The Node Design Tool for TTEthernet Networks
D-TTE-G-01-010	TTE Plan: The TTEthernet Scheduler
D-TTE-G-01-006	TTE Load: The TTEthernet Download Tool
D-TTE-G-01-005	TTE View: The TTEthernet Gigabit Monitoring User Guide
D-TTE-G-01-001	TTE Software Installer for TTE-Tools
D-TTE-SWITCH12-S-10-001	TTEthernet Development Switch 1Gbit/s 12 Ports Interface Control Document
D-TTEPMC-M-10-001	TTEthernet End System Interface Control Document (PMC/XMC Implementation)

3 AECS Test Project Overview

The AECS Test Project provides a sample project used to test the features and capabilities of the AECS System Under Test (SUT) which is composed of the Flight Control Computers (FCCs), AFDX switches, and TTE switches and the AECS Verification and Validation Research Test Bench (V&VRTB) which is composed of the Signal Generator Computer (SGC) and the Display Computer and Monitor (DCM). The AECS Test Project implements most of the software requirements called out in the *Statement of Work for the Assessment Environment for Complex Systems*.

The purpose of the AECS Test Project is to:

- Provide Test Software for the AECS System
- Exercise AECS Software and Hardware Features
- Provide Examples for Future AECS Researchers
- Provide a Framework for Future AECS Researchers

The goals of the AECS Test Project include:

- Test 653 Partitions
- Exercise Inter-Process Communications (IPC) Between 653 Partitions
- Operate at Varying Rates of Execution
- Test AFDX/TTE Messaging
- Test System Redundancy
- Mimic IMA Architecture

Figure 2 shows the AECS system architecture. Three FCCs communicate with the SGC via the Avionics Bus (which represents either the AFDX network or the TTE network). Both the FCCs and the SGC communicate with the DCM using the maintenance network. The FCCs and SGC execute VxWorks 653 while the DCM executes Windows XP.

The remainder of this section provides an overview of the AECS Test Project including basic architecture and key terms.

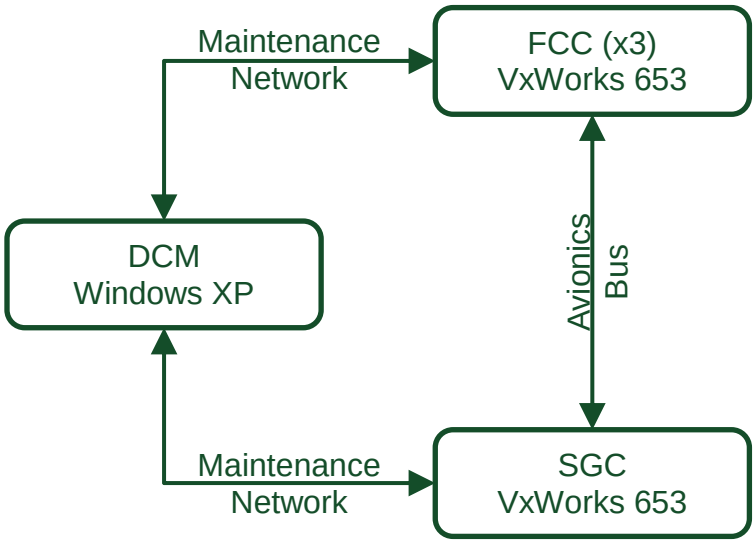


Figure 2: AECS Test Project Architecture

Appendix B provides details on several minimal projects that are simpler than the full AECS Test Project. For a user unfamiliar with AECS, 653, AFDX, and/or TTE, these simpler projects provide a good starting point for understanding the AE.

3.1 Avionics Bus

The AECS Test Project supports communications between the FCCs and the SGC via either AFDX or TTE (but not both simultaneously). The user chooses the desired network technology at build time and must ensure that all FCCs and the SGC are installed with images using the same network technology. See Section 4.3.1 for information on changing the build between AFDX and TTE.

Throughout the remainder of this document, the AFDX/TTE network connection will be generically referred to as the Avionics Bus.

The AECS Test Project avionics bus configuration is further detailed in Section 6.2.

3.1.1 Virtual Links

The AECS Test Project defines a single configuration with 10 virtual links. The virtual links are defined identically for both AFDX and TTE. The virtual links represent four distinct message types:

- FCC to FCC Cross-Channel Data Link (CCDL) Messages: Virtual Links (VL) 101, 201, and 301
- FCC to Components Response Messages: VLs 201, 202, and 302
- SGC to Components Command Message: VL 903
- Sensor/Actuator to FCC Messages: VLs 1004, 1104, and 1204

Figure 3 lists the four message types (sensor/actuator are combined as they are very similar), details their respective VLs, and provides an overview of the content of each message.

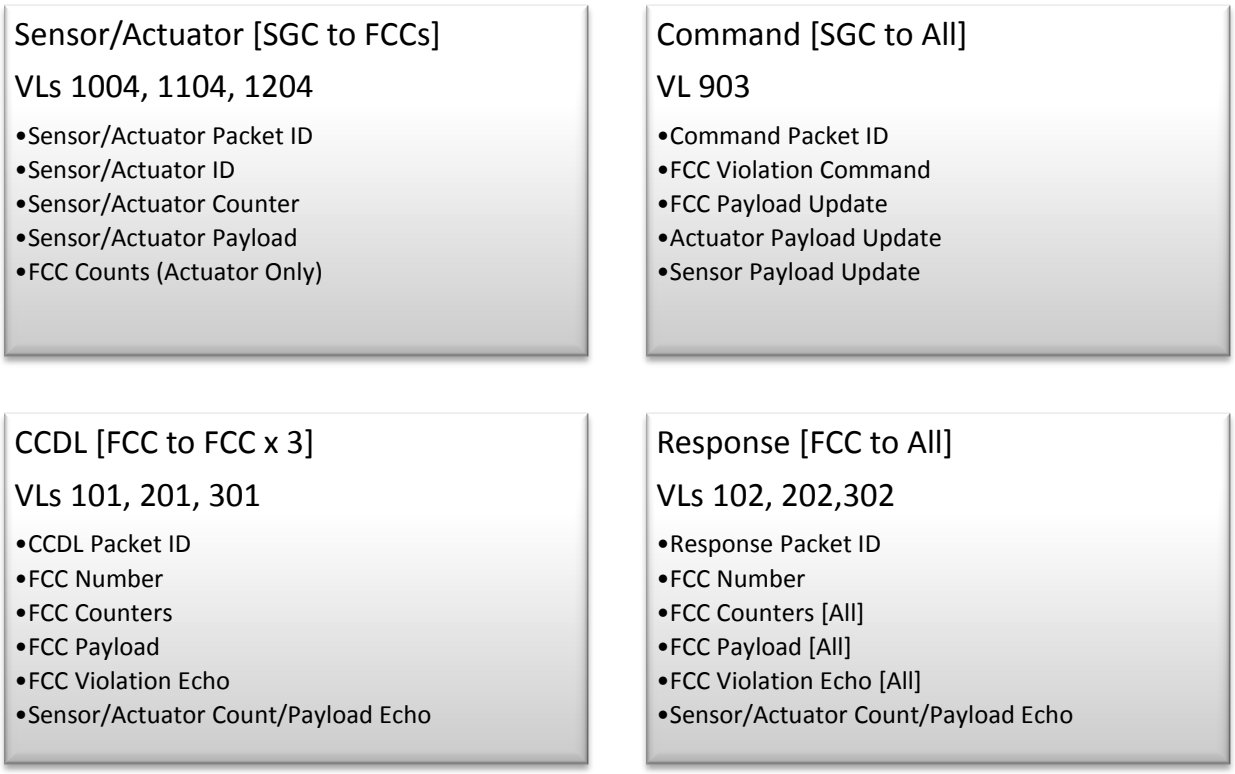


Figure 3: AECS Test Project AFDX Message Overview

3.1.2 AFDX

AFDX is one of the two primary avionics busses within the AECS system. AFDX provides real-time, reliable, redundant communications between the FCCs and/or the SGC. To support redundant operations, each AFDX Network Interface Card (NIC) has two ports (each FCC has one card, the SGC has four cards distributed between two SBCs) and the AECS system contains two switches. This provides two redundant AFDX buses, the Red bus and the Blue bus. Software in the AFDX device driver implements redundant operations.

Each AECS project must define one or more AFDX switch configurations. The switch configuration details how the AFDX switches will route packets. Each AFDX switch (there are two in AECS) is configured identically. General information related to the AFDX switches is available in the *AECS User's Guide*.

The AECS Test Project AFDX configuration is further detailed in Section 6.2.4.

3.1.3 TTE

TTE is one of the two primary avionics bus within the AECS system. TTE provides real-time, reliable, redundant communications between the FCCs and/or the SGC. To support redundant operations, each TTE NIC has three ports, two of which are used by the AECS Test Project (each FCC has one card, the SGC has four cards distributed between two SBCs) and the AECS system contains two switches. This provides two redundant TTE buses, the Purple bus and the Green bus. Software in the TTE device driver implements redundant operations.

Each AECS project must define one or more TTE switch configurations. The switch configuration details how the TTE switches will route packets. Each TTE switch (there are two in AECS) is configured identically. General information related to the TTE switches is available in the *AECS User's Guide*.

The AECS Test Project TTE configuration is further detailed in Section 6.2.5.

3.2 Flight Control Computers

The FCCs within the AECS system represent the primary flight control computers on an aircraft. In a representative system, they would receive air data from sensors, receive stick inputs from the pilot, perform control calculations, and send actuator commands. Three FCCs provide for a triplex system. Communications between the FCCs occurs via Avionics Bus.

For the AECS Test Project, each of the three Flight Control Computers executes two applications. Each application executes in its own partition. Figure 4 shows the AECS Test Project FCC partition layout.

Figure 4: AECS Test Project FCC Partitions

3.2.1 FCC Application 1

FCC Application 1 (fapp1) executes in FCC partition 1. fapp1 performs the following activities:

- Executes at 20 Hz
- Increments a Counter
- Receives FCC Payload and Violations from the SGC via the Avionics Bus
- Sends FCC Violations to fapp2 via Sampling Ports
- Sends FCC CCDL message to other FCCs via the Avionics Bus
- Receives FCC CCDL messages from other FCCs via the Avionics Bus
- Sends Counter and Payloads to SGC and Actuator via the Avionics Bus

The FCC Payload is a simple byte array, settable from the DCM, which represents the data generated by the FCC (for example, actuator commands). The content of the payload is not important within the AECS Test Project.

Figure 5 shows the fapp1 inputs and outputs.

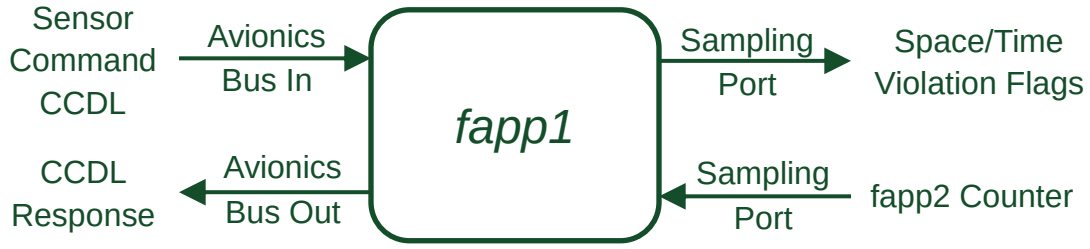


Figure 5: fapp1 Inputs/Outputs

fapp1 is further discussed in Section 6.4.1.

3.2.2 FCC Application 2

FCC Application 2 (fapp2) executes in FCC partition 2. FCC Application 2 performs the following activities:

- Executes at 40 Hz
- Increments a Counter
- Communicates with fapp1 via Sampling Ports

- Performs Time and Space Violations

Figure 6 shows the fapp2 inputs and outputs.

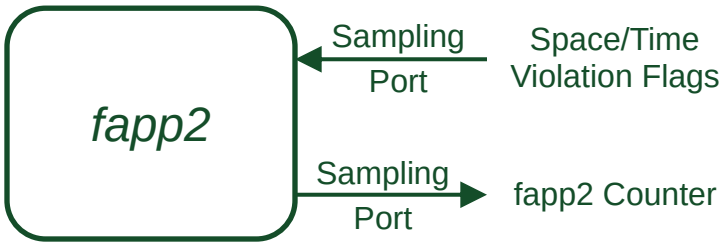


Figure 6: fapp2 Inputs/Outputs

fapp2 is further discussed in Section 6.4.2.

3.3 Signal Generator Computer

The SGC within the AECS system simulates the avionics components outside of the FCCs such as sensors, actuators, control inputs, and others. The SGC can also serve as a test recorder and system manager.

For the AECS Test project, the SGC executes four applications distributed over two SBCs (A and B). Each application executes in its own partition. Figure 7 shows the AECS Test Project SGC partition layout. Each partition has access to one of the four AFDX/TTE interfaces present on the SGC. This partition layout allows the AECS Test Project to mimic a typical IMA configuration.

Figure 7: AECS Test Project SGC Partitions

3.3.1 SGC Communications Daemon

Partition 1 on SGC-A, also referred to as the Communications Daemon (commd), contains two tasks, the Router and the Recorder.

The Router task performs the following activities:

- Receives Payload and Violation Updates from the DCM via Shared Memory
- Sends Payload Updates to the Sensors/Actuators via the Avionics Bus
- Sends Payload and Violation Updates to the FCCs via the Avionics Bus
- Receives Messages from FCCs/Sensors/Actuator via the Avionics Bus
- Forwards Messages Received via the Avionics Bus to Recorder via vThreads Message Queue

- Forwards Messages Received via the Avionics Bus to the DCM via Shared Memory

The Router is further discussed in Section 6.5.1. Figure 8 shows the Router inputs and outputs.

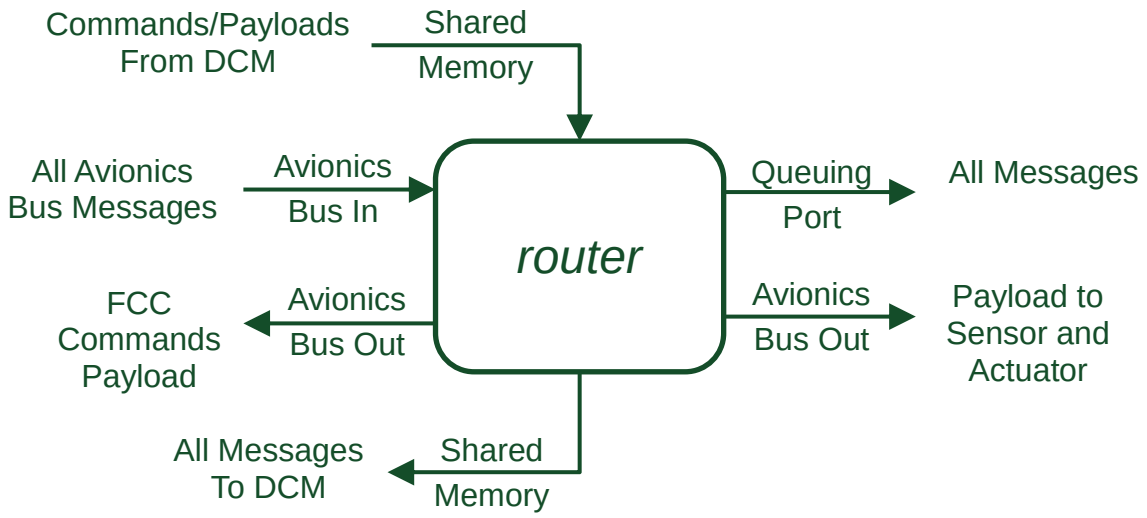


Figure 8: router Inputs/Outputs

The Recorder task performs the following activities:

- Receives Messages from Router via vThreads Message Queue
- Saves all Messages to Flash
  - Tags Packet with Time to Nearest Millisecond
  - Tags Packet with Source (Application Sending Packet)

Figure 9 shows the Recorder inputs and outputs.

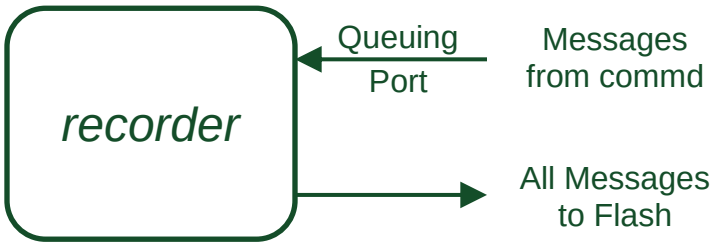


Figure 9: recorder Inputs/Outputs

Recorder records Avionics Bus messages to a single file. Recorder cannot record traffic from the redundant busses. Wireshark may be used to monitor both buses if desired (see the *AECS User's Guide* for additional information).

- `/sata9/recorder.pcap`: Packets received via the Avionics Bus.

Note that Recorder has a hard-coded 10MB file limit. This prevents excessive file sizes if the system is left on for a long period of time. At current data rates, the 10MB file limit represents approximately 3

minutes of data (around 43,000 packets). The file is truncated each time the SGC reboots so be sure to copy it prior to resetting or power cycling the SGC.

The Recorder is further discussed in Section 6.5.1.

3.3.2 SGC Actuator

SGC Actuator executes in SGC-A partition 2. Actuator performs the following activities:

- Executes at 20 Hz
- Increments an Actuator Counter
- Receives Actuator Payload from Router via the Avionics Bus
- Sends Actuator Counter and Actuator Payload to FCCs via the Avionics Bus
- Receives Actuator Commands from the FCCs

The Actuator Payload is a simple byte array, settable from the DCM, which represents the data generated by the Actuator. The content of the payload is not important within the AECS Test Project.

Figure 10 shows the Actuator inputs and outputs.

Figure 10: actuator Inputs/Outputs

Actuator is further discussed in Section 6.5.2.

3.3.3 SGC Sensor

SGC Sensor executes in SGC-B, partition 1 and partition 2; two separate instances of Sensor execute. Sensor performs the following activities:

- Executes at 20 Hz
- Increments a Sensor Counter
- Receives Sensor Payload from Router via the Avionic Bus
- Sends Sensor Counter and Sensor Payload to FCCs via the Avionics Bus

The Sensor Payload is a simple byte array, settable from the DCM, which represents the data generated by the Sensor. The content of the payload is not important within the AECS Test Project.

Figure 11 shows the Sensor inputs and outputs.

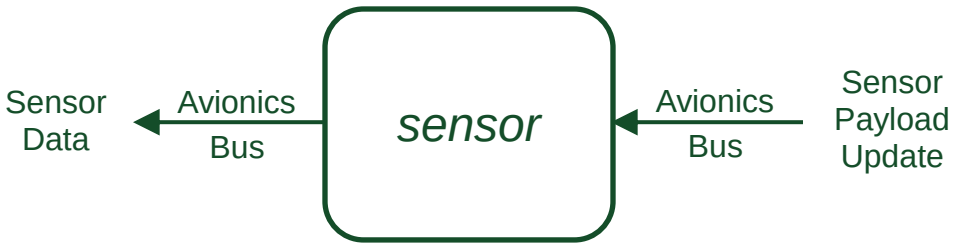


Figure 11: sensor Inputs/Outputs

Sensor is further discussed in Section 6.6.1.

3.4 Monitor and Control

The Monitor and Control (MonCon) user interface executes on the DCM. MonCon allows the user to perform the following activities:

- View Sensor Counter and Payload
- View Actuator Counter and Payload
- View FCC fapp1 and fapp2 Counters
- View FCC fapp1 Payload
- Change Sensor and Actuator Payloads
- Change FCC Payloads
- Change FCC Violation Commands

The MonCon user interface is shown in Figure 12.

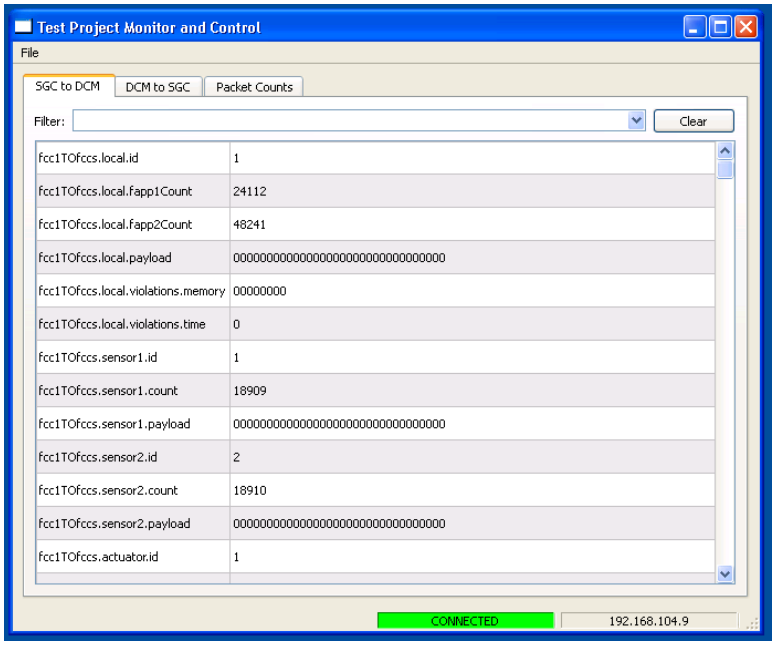


Figure 12: MonCon Application

Router, as noted above, places a copy of all AFDX/TTE packets into a shared data region accessible by both the core OS and partition 1. VxWorks shell commands within the core OS print the packet information stored in the shared data region. MonCon connects to the SGC VxWorks shell and thus the core OS via the Maintenance Network. MonCon can then execute the shell commands and retrieve the information to display. Figure 13 graphically depicts this communications path.

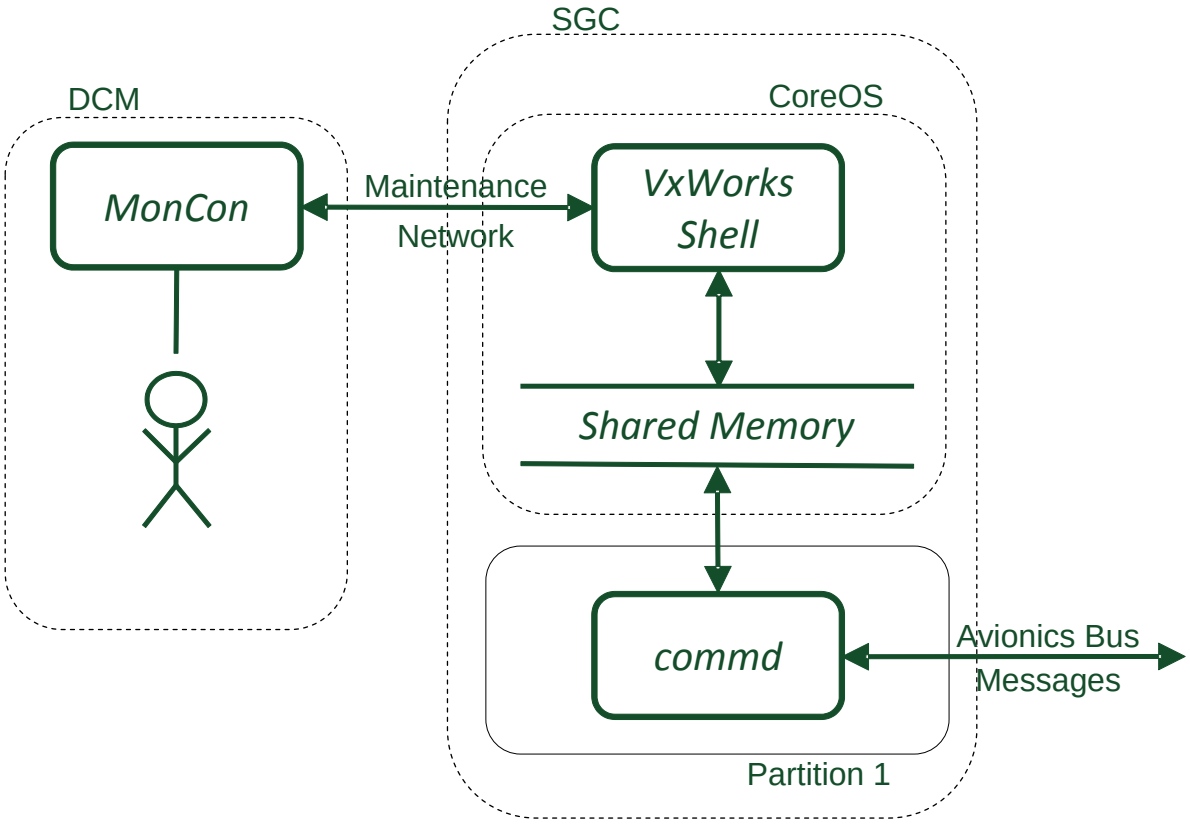


Figure 13: MonCon Communications Path

MonCon is further discussed in Section 5.3 and Section 6.7.

3.5 Testing

Testing of the AECS Test Project is described as part of the general testing of the AECS. See the *AECS Test Setup and Procedure* document for additional information on testing.

4 Configuration Management

4.1 Source Code Repository

The AECS Test Project is stored within the AECS subversion repository. Within the repository, it is located at:

```
/trunk/projects/test
```

This directory is considered the 'root' of the AECS Test Project. It can be checked out on the DCM using the Uniform Resource Locator (URL):

```
https://localhost/svn/AECS/trunk/projects/test
```

Additional information on subversion is available in the *AECS User's Guide*.

A copy of the AECS Test Project resides on the DCM in the directory:

```
E:\projects\test
```

The remainder of this document assumes you are working from this directory. The content is also applicable if you check the AECS Test Project out to a different location and work with it there.

4.2 Directory Layout

The root of the AECS Test Project directory tree contains the following files and subdirectories:

- `Makefile`: The top-level Makefile for the AECS Test Project; see Section 4.3 for additional details. Selection of the Avionics Bus occurs within this file.
- `afdx`: Contains files related to the AFDX switch configuration; see Section 4.2.1 for additional details.
- `build.mk`: Defines the build for the AECS Test Project, see Section 4.3 for additional details.
- `common`: Contains files shared across all subsystems; see Section 6.3 for additional details.
- `dcm`: Subdirectory containing the DCM components; see Section 4.2.2 for additional details.
- `doxygen`: Contains the files used to generate a portion of this document; see Section 4.2.3 for additional details.
- `fcc`: Subdirectory containing the FCC components; see Section 4.2.4 for additional details.
- `mos`: Subdirectory containing the module OS; see Section 4.2.5 for additional details.
- `sbcs`: Configuration mapping the SBCs to AECS Test Project components, see Section 4.2.6 for additional details.
- `sgc-a`: Subdirectory containing the SGC-A components; see Section 4.2.7 for additional details.
- `sgc-b`: Subdirectory containing the SGC-B components; see Section 4.2.8 for additional details.
- `tte`: Contains files related to the TTE switch and network configuration; see Section 4.2.9 for additional details.

The above files all reside in subversion. The build subsystem creates additional directories for storing files generated during the build. They may be present. They may be deleted. They will be recreated during the next build. They are not stored in subversion.

- `_build_afdx`: Files generated during the build when AVIONICS_BUS is set to 'afdx'. Contains the OS and partition images.
- `_build_tte`: Files generated during the build when AVIONICS_BUS is set to 'tte'. Contains the OS and partition images.

4.2.1 afdx Subdirectory

The `afdx` subdirectory contains the AFDX configuration file, `afdx.conf`, for the AECS Test Project. See Section 6.2.4 for additional information on AFDX configuration.

4.2.2 dcm Subdirectory

The `dcm` subdirectory contains the following files and subdirectories:

- `dcm/moncon`: Source code for the monitor and control user interface application; see Section 6.7 for additional details.

4.2.3 doxygen Subdirectory

The `doxygen` subdirectory contains configuration files required for generating portions of this document. See **Error! Reference source not found.** for additional information.

4.2.4 fcc Subdirectory

The `fcc` subdirectory contains the following files and subdirectories:

- `fcc/afdx`: Contains per FCC AFDX configuration files; see Section 6.2.4.2 for additional details.
- `fcc/build.mk`: Build file for the FCC subdirectory.
- `fcc/core`: Source code that executes in the core OS; see Section 6.4.6 for additional details.
- `fcc/fapp1`: Source code for the fapp1 application; see Section 6.4.1 for additional details.
- `fcc/fapp2`: Source code for the fapp2 application; see Section 6.4.2 for additional details.
- `fcc/image`: Files for configuring and building the FCC image; see Section 6.4.4 for additional details.
- `fcc/pos`: Files for configuring and building the FCC partition OS; see Section 6.4.3 for additional details.
- `fcc/tte`: Contains per FCC TTE configuration files; see Section 6.2.5 for additional details.

4.2.5 mos Subdirectory

The module OS subdirectory contains the makefiles for building the VxWorks 653 system. See 6.1 for additional details.

4.2.6 sbcs Subdirectory

The `sbcs` subdirectory contains configuration files mapping the SBCs to AECS Test Project components. See Section 4.3.3 for additional details.

4.2.7 sgc-a Subdirectory

The `sgc-a` subdirectory contains the following files and subdirectories:

- `sgc-a/actuator`: Source code for the actuator application; see Section 6.5.2 for additional details.
- `sgc-a/afdx`: Contains AFDX configuration files for SGC-A; see Section 6.2.4.2 for additional details.
- `sgc-a/build.mk`: Build file for the `sgc-a` subdirectory.
- `sgc-a/commd`: Source code for the communications daemon application; see Section 6.5.1 for additional details.
- `sgc-a/core`: Source code that executes in the core OS; see Section 6.5.6 for additional details.
- `sgc-a/image`: Files for configuring and building the SGC-A image; see Section 6.5.4 for additional details.
- `sgc-a/pos`: Files for configuring and building the SGC-A partition OS; see Section 6.5.3 for additional details.
- `sgc-a/tte`: Contains TTE configuration files for SGC-A; see Section 6.2.5 for additional details.

4.2.8 sgc-b Subdirectory

The `sgc-b` subdirectory contains the following files and subdirectories:

- `sgc-b/afdx`: Contains AFDX configuration files for SGC-B; see Section 6.2.4.2 for additional details.
- `sgc-b/build.mk`: Build file for the `sgc-b` subdirectory.
- `sgc-b/core`: Source code that executes in the core OS; see Section 6.6.5 for additional details.
- `sgc-b/image`: Files for configuring and building the SGC-B image; see Section 6.6.3 for additional details.
- `sgc-b/pos`: Files for configuring and building the SGC-B partition OS; see Section 6.6.2 for additional details.
- `sgc-b/sensor`: Source code for the sensor application; see Section 6.6.1 for additional details.
- `sgc-b/tte`: Contains TTE configuration files for SGC-B; see Section 6.2.5 for additional details.

4.2.9 tte Subdirectory

The `tte` subdirectory contains the TTE network description file, `network_description.xml`, for the AECS Test Project. See Section 6.2.5 for additional information on TTE configuration.

4.3 Build Subsystem

WindRiver provides example make files for building the 653 components such as the modules OS, the system image, the partition OS, and the application. These make files are described in the *VxWorks 653 Configuration and Build Guide*. These example make files include variables and rules from the Board Support Package (BSP) directory and provide a template for building 653 components. Unfortunately, WindRiver did not provide a unified build that constructs all of the different 653 components in a single step. Therefore, the WVHTC Foundation built a layer on top of the WindRiver make files to provide a single command for building all of the FCC, SGC, AFDX, and TTE components. This section provides an overview of the build subsystem. The build subsystem is well-commented; see the individual files for additional details. Additional information on the AECS build subsystem is available in the *AECS User's Guide*.

The build subsystem is based upon the standard make utility. A top-level `Makefile` residing in the root of the AECS Project Test directory is the entry point for building the project. The top-level `Makefile` recursively includes `build.mk` files in the component subdirectories. Both the top-level `Makefile` and the component `build.mk` files define the rules available to the user. All builds are executed by calling 'make' at the root of the AECS Project Test directory (typically `E:\projects\test`):

```
E:\projects\test > make <rule>
```

The most commonly used rules are listed in Table 3.

Table 3: Commonly Used Build Rules

Rule	Description
<nothing>	If no rule is specified, the default rule is used. This rule executes when 'Build' is selected from within Workbench.
default	The default rule builds the module OS, the FCC image and the SGC image. Note that the module OS is shared by the FCC and SGC images.
clean	Removes the _build_<bus> directory. This removes all generated files and will require all components to be rebuilt. This rule executes when 'Clean' is selected from within Workbench.
upload	This uploads the files for the system (FCC1, FCC2, FCC3, FCC4, FCC5, FCC6, SGC-A, and SGC-B). This rule is available from within Workbench under the 'Build Options' submenu. See Section 5.1 for upload details.
flash	This flashes the files for the core system (FCC1, FCC2, FCC3, FCC4, FCC5, FCC6, SGC-A, and SGC-B).). This rule is available from within Workbench under the 'Build Options' submenu. See Section 5.1 for flash details.
mos	Build just the module OS. This will be built automatically if required.
fcc<1|2|3|4|5|6>	Build just the FCC image for the give FCC. Use this if you are modifying FCC code, replace <#> with the FCC number.

sgc-<a|b>	Build just the SGC image for the give SGC. Use this if you are modifying SGC code, replace <#> with the SGC letter.
flash/fcc<1|2|3|4|5|6>	Flash a single FCC, replace <#> with the FCC number.
upload/fcc<1|2|3|4|5|6>	Upload a single FCC's files, replace <#> with the FCC number.
flash/sgc-<a|b>	Flash just the SGC, replace <#> with the SGC letter.
upload/sgc-<a|b>	Upload just the SGC's files, replace <#> with the SGC letter.
afdx/load/<red|blue>	Uploads the AFDX configuration file to either the red or blue switch. Only valid when NETWORK_BUS is set to AFDX. This rule is available from within Workbench under the 'Build Options' submenu.
tte/load/<purple|green>	Uploads the TTE configuration file to either the green or purple switch. Only valid when NETWORK_BUS is set to TTE. This rule is available from within Workbench under the 'Build Options' submenu.
help	Print a list of all available rules.

The WindRiver-provided makefiles are not ideal. They do not handle dependencies well and they ignore header file dependencies. If there seems to be a problem with the build, it is best to clean and rebuild.

Workbench uses the same makefile to build the AECS Test Project components. Workbench calls 'make' with the desired rule. Workbench, as delivered, supports the 'default', 'clean', 'afdx/load, 'tte/load', 'upload', and 'flash' rules. Additional rules can be added through the project properties dialog.

VxWorks 653 2.3 and Workbench 3.2 provide the user with the ability to use Eclipse features to define each VxWorks 653 component as an Eclipse project and combine building all of the projects together in a single workspace. As the AECS Test Project was originally developed against 2.2 and the build subsystem was in place and functional, no effort was expended to upgrade the AECS Test Project to use the new build features of 2.3.

4.3.1 Selection of Network Bus

Modify `Makefile` within the root of the project directory to select the desired Avionics Bus.

When first checked out of subversion, no Avionics Bus is selected within the AECS Test Project `Makefile`. If make is executed before selecting an Avionics Bus, an error will be displayed by the build subsystem.

```
AVIONICS_BUS must be defined, valid values are 'afdx' or tte', set in environment, provide on
command line, or uncomment value at beginning of Makefile.
```

To select an Avionics Bus, edit `Makefile` looking for the lines shown below (line numbers are provided, although these may change if the `Makefile` has been modified). Uncomment the line (remove the '#' symbol at the beginning) for the desired Avionics Bus.

```
51 #
52 # Uncomment ONE and only ONE to select a default NETWORK_BUS for the build.
53 #
54 #AVIONICS_BUS?=afdx
55 #AVIONICS_BUS?=tte
```

Note that it is also possible to specify the Avionics Bus in the environment and on the command line. Note that specifying the Avionics Bus in this manner will override the line within `Makefile`.

```
E:\projects\test > make AVIONICS_BUS=tte <rule>
```

Note that selection of the Avionics Bus is unique to the AECS Test Project and is not discussed with the *AECS User's Guide* documentation on the generic AECS build subsystem.

4.3.2 Changing the AECS Test Project Components to Build

By default, the AECS Test Project makefiles builds and installs all FCCs (1 through 6). If an FCC is removed from the system, it may be desirable to remove it from the build. The list of SBCs to build resides in the `build.mk` file in the root of the AECS Test Project. Look for the lines (note that shown line numbers may change if file has been modified):

```
45 #
46 # List of SBCs to install.
47 #
48 # Each SBC must have a matching configuration file relative to
49 # $(TOP)/$(SBC_CONFIG_DIR) of the development tree.  This configuration file
50 # indicates the image to install to this SBC, the flash method, and the IP
51 # address of the system.
52 #
53 SBC_CONFIG_DIR=sbcs
54 LIST_OF_SBCS=fcc1 fcc2 fcc3 fcc4 fcc5 fcc6 sgc-a sgc-b
```

And update as required to match the available SBCs. Note that the minimal system for testing all features is FCC1, FCC2, FCC3, SGC-A, SGC-B, and either TTE or AFDX. For example, to eliminate FCC5 and FCC6, change line 54 above to:

```
54 LIST_OF_SBCS=fcc1 fcc2 fcc3 fcc4 sgc-a sgc-b
```

4.3.3 SBC IP Addresses and Flash Method

For the AECS Test Project, mapping between IP address/flash method and individual SBCs is contained within the files within the `sbcs` directory off of the AECS Test Project root. This directory contains one file for each SBC (`fcc1.mk`, `fcc2.mk`, `fcc3.mk`, `fcc4.mk`, `fcc5.mk`, `fcc6.mk`, `sgc-a.mk` and `sgc-b.mk`). These file map each SBC to a VxWorks 653 image, Avionics Bus configuration, IP address, and flash method. The contents of `sbcs/fcc1.mk` are shown below.

```
34 #
35 # Image for this SBC.
36 #
37 SBC_IMAGE=fcc/image
38
39 #
40 # TTE configuration for this SBC.
41 #
42 ifeq ($(AVIONICS_BUS),tte)
43 TTE_CONFIG=fcc/tte/fcc1
44 endif
45
46 #
47 # AFDX configuration for this SBC.
48 #
```

```
49 ifeq ($(AVIONICS_BUS),afdx)
50 AFDX_CONFIG=fcc/afdx/fcc1
51 endif
52
53 #
54 # Flash method.
55 #
56 SBC_FLASH_METHOD=telnet:192.168.104.200:4001
57
58 #
59 # IP Address of the SBC.
60 #
61 SBC_IP_ADDR=192.168.104.1
```

If the IP address or flash method of an SBC within the AECS Test Project is changed, update these files to change the configuration of the AECS Test Project. See the build discussion in the *AECS User's Guide* for additional information on SBC IP address and flash method configuration.

4.4 Building the AECS Test Project

The AECS Test Project can be built from either the VxWorks 653 command shell or from with WindRiver's Workbench Integrated Development Environment (IDE). In both cases, the makefiles described in Section 4.3 build the software. The two methods for building the AECS Test Project should not be used concurrently.

4.4.1 From the VxWorks 653 Command Shell

To build the AECS Test Project from the command shell, start an AECS VxWorks 653 Command Shell as discussed in the *AECS User's Guide*. Note that this shell is slightly different than a standard VxWorks 653 Command Shell as the WIND_PATH environment variable must point to the AECS VxWorks kernel directory.

From the command shell, change to the AECS Test Project directory. This example assumes the AECS Test Project is checked out of subversion. This example uses `E:\projects\test` as the directory of the AECS Test Project.

```
E:
cd \projects\test
```

Once in the proper directory, execute the make command:

```
make
```

The build subsystem determines which components are out of date and builds them as needed. The build will stop if any errors are detected. Once the system is built, it must be installed to the targets. See Section 5.1 for installation details.

4.4.2 From Within Workbench

To build the AECS Test Project from within Workbench, start Workbench as discussed in the *AECS User's Guide*. Note that this is a slightly different procedure than starting a standard Workbench as the WIND_PATH environment variable must point to the AECS VxWorks kernel directory.

This example assumes the AECS Test Project is checked out of subversion and has been imported into Workbench. This example uses the project within `E:\projects\test` directory.

If Workbench is not using `E:\projects` as its workspace directory, use "File->Switch Workspace …" to change the workspace to `E:\projects`.

Once the correct workspace is selected, open the project navigator. From the project navigator, right click 'test' and select 'Build Project'. There are other methods to initiate a build within Workbench, see *WindRiver WorkBench User's Guide for VxWorks 653* and *WindRiver WorkBench User's Guide for VxWorks* for additional information on using Workbench.

The build subsystem determines which components are out of date and builds them as needed. The build will stop if any errors are detected. Once the system is built, it must be installed to the targets. See Section 5.1 for installation details.

Note that Workbench is configured to use the same Makefiles as the command line build and simply calls the WVHTC Foundation created Makefiles with the appropriate rule.

5 AECS Test Project Execution

5.1 Installing the AECS Test Project

Installation of the AECS Test Project is handled by the build subsystem. Rules within the Makefiles are used to both flash the images to Electrically Erasable Programmable Read-Only Memory (EEPROM) and upload the necessary files to the flash drive.

5.1.1 Installation from the VxWorks 653 Shell

To install the AECS Test Project from the command shell, start an AECS VxWorks 653 Command Shell as discussed in the *AECS User's Guide*. Note that this shell is slightly different than a standard VxWorks 653 Command Shell as the WIND_PATH environment variable must point to the AECS VxWorks kernel directory.

From the command shell, change to the AECS Test Project directory. This example assumes the AECS Test Project is checked out of subversion. This example uses `E:\projects\test` as the directory of the AECS Test Project.

```
E:
cd \projects\test
```

Once in the proper directory, execute the make commands:

```
make upload
make flash
```

The flash command takes up to several minutes to complete. The build subsystem will flash the SBCs and upload the required files. The above rules perform the flash/upload on FCC1, FCC2, FCC3, FCC4, FCC5, FCC6, SGC-A, and SGC-B (see Section 4.3.2 for details on removing missing SBCs from the build). An individual SBC can be done as well. For example, to flash/upload just FCC5:

```
make upload/fcc5
make flash/fcc5
```

The flash step is only necessary if the Extensible Markup Language (XML) configuration files or the core OS has changed.

If AVIONICS_BUS is set to afdx and the AFDX configuration file, `afdx/afdx.conf`, changes, upload it to the switches using the rules:

```
make load/afdx
```

If AVIONICS_BUS is set to tte and the TTE network description file, tte/network_description.xml, changes, upload it to the switches using the rules:

```
make load/tte
```

5.1.2 Installation from within Workbench

To install the AECS Test Project from within Workbench, start Workbench as discussed in the *AECS User's Guide*. Note that this is a slightly different procedure than starting a standard Workbench as the WIND_PATH environment variable must point to the AECS VxWorks kernel directory.

This example assumes the AECS Test Project is checked out of subversion and has been imported into Workbench. This example uses the project within `E:\projects\test` directory.

If Workbench is not using `E:\projects` as its workspace directory, use "File->Switch Workspace …" to change the workspace to `E:\projects`.

Once the correct workspace is selected, open the project navigator. From the project navigator, right click 'test' and select 'Build Options->flash'. Once flashing is complete, select 'Build Options->upload'. These shortcuts simply call the appropriate Makefile rules. Additional shortcuts to the Makefile rules can be added within the 'Properties' dialog, see *WindRiver WorkBench User's Guide for VxWorks 653* and *WindRiver WorkBench User's Guide for VxWorks* for additional information on using Workbench.

If AVIONICS_BUS is set to afdx and the AFDX configuration file, `afdx/afdx.conf`, changes, upload it to the switches using the 'Build Options->load/afdx menu item.

If AVIONICS_BUS is set to tte and the TTE network description file, `tte/network_description.xml`, changes, upload it to the switches using the 'Build Options->load/tte menu item.

5.1.3 Installation Details

5.1.3.1 Flashing Electrically Erasable Programmable Read-Only Memory

For the AECS Test Project, there are five regions of EEPROM that must be flashed:

- ROM Payload
- Core OS Symbol Table
- Core OS Image
- Partition OS Image
- Configuration Record

Table 4 lists the regions to be flashed, the flash address, and the location of the file within the `_build` tree.

Table 4: AECS Test Project EEPROM Regions

Region	Address	File Location
ROM Payload	0xE0000100	_build_<afdx|tte>/fcc/image/sms_romPayload.hex
Core OS Symbol Table	0xE0200000	_build_<afdx|tte>/fcc/image/coreOSsymbols.hex
Core OS Image	0xE0400000	_buid_<afdx|tte>/fcc/image/coreOs.hex
Partition OS Image	0xE0940000	_build_<afdx|tte>/fcc/image/partOs.hex
Configuration Record	0xE09B0000	_build_<afdx|tte>/fcc/configRecorder.hex

NOTE: Replace 'fcc' with 'sgc-a' or 'sgc-b' above to obtain the file locations for the SGC.

Flashing of the EEPROM is handled by the `trunk\aecs\install\flash_hex.py` and `trunk\aecs\install\flash_vx653.py` scripts. The actual commands are embedded within the Makefiles. While it is possible to flash regions individually, the Makefiles only support flashing all regions at the same time. The Makefile rules for flashing are:

- flash/fcc1: Flash all EEPROM regions for FCC1.
- flash/fcc2: Flash all EEPROM regions for FCC2.
- flash/fcc3: Flash all EEPROM regions for FCC3.
- flash/fcc4: Flash all EEPROM regions for FCC4.
- flash/fcc5: Flash all EEPROM regions for FCC5.
- flash/fcc6: Flash all EEPROM regions for FCC6.
- flash/sgc-a: Flash all EEPROM regions for the SGC-A.
- flash/sgc-b: Flash all EEPROM regions for the SGC-B.
- flash: Flash all EEPROM regions for FCC1, FCC2, FCC3, FCC4, FCC5, FCC6, SGC-A, and SGC-B.

Only the 'flash' rule is available from within Workbench. The other rules are available on the VxWorks 653 Shell command line.

Typically, the FCCs/SGC should be flashed anytime any of the XML configuration files change during development.

The `flash_hex.py` and `flash_vx653.py` scripts are described in detail in the *AECS User's Guide*.

5.1.3.2 Copying Files to the SATA Drives

Table 5 shows the files that need to be copied to the FCC Serial Advanced Technology Attachment (SATA) drive for the AECS Test Project.

Table 5: FCC Installation Files

File	Destination	Description
fcc/image/image_startup	/sata8	Startup file for FCCs.
build<tte|afdx>/fcc/core/core.o	/sata8	FCC object code specific to the coreOS.
build<tte|afdx>/fcc/image/fapp1.bin	/sata8	The fapp1 application.
build<tte|afdx>/fcc/image/fapp2.bin	/sata8	The fapp2 application.
_buid_afdx/fcc/afdx/fcc<#>/afdxConfig.o	/sata8	AFDX configuration file (different for each SBC).
_build_tte/fcc/tte/fcc<#>/tteConfig.o	/sata8	TTE configuration file (different for each SBC).

Table 6 shows the files that need to be copied to the SGC-A SATA drive for the AECS Test Project.

Table 6: SGC-A Installation Files

File	Destination	Description
sgc-a/image/image_startup	/sata8	Startup file for SGC.

build<tte|afdx>/sgc-a/core/core.o	/sata8	SGC object code specific to the coreOS.
build<tte|afdx>/sgc-a/image/commd.bin	/sata8	The commd application.
build<tte|afdx>/sgc-a/image/actuator.bin	/sata8	The actuator application.
_build_afdx/sgc-a/afdx/afdxConfig.o	/sata8	AFDX configuration file.
_build_tte/sgc-a/tte/tteConfig.o	/sata8	TTE configuration file.

Table 7 shows the files that need to be copied to the SGC-B SATA drive for the AECS Test Project.

Table 7: SGC-B Installation Files

File	Destination	Description
sgc-a/image/image_startup	/sata8	Startup file for SGC.
build<tte|afdx>/sgc-b/core/core.o	/sata8	SGC object code specific to the coreOS.
build<tte|afdx>/sgc-b/image/sensor.bin	/sata8	The sensor application.
_build_afdx/sgc-b/afdx/afdxConfig.o	/sata8	AFDX configuration file.
_build_tte/sgc-b/tte/tteConfig.o	/sata8	TTE configuration file.

Copying files to the FCCs and SGC is handled by the Makefiles using the `mput.py` script. The Makefile rules for uploading are:

- upload/fcc1: Copy all files needed for FCC1.
- upload/fcc2: Copy all files needed for FCC2.
- upload/fcc3: Copy all files needed for FCC3.
- upload/fcc4: Copy all files needed for FCC4.
- upload/fcc5: Copy all files needed for FCC5.
- upload/fcc6: Copy all files needed for FCC6.
- upload/sgc-a: Copy all files needed for the SGC-A.
- upload/sgc-b: Copy all files needed for the SGC-B.
- upload: Copy all files needed for FCC1, FCC2, FCC3, FCC4, FCC5, FCC6, SGC-A, and SGC-B.

Only the 'upload' rule is available from within Workbench. The other rules are available on the VxWorks 653 Shell command line.

5.2 Starting the AECS Test Project

Once installed, the AECS Test Project starts automatically when the system is booted. To restart the AECS Test Project, power-cycle the core system components. The core components include FCC1, FCC2, FCC3, SGC, and the two AFDX/TTE switches. Once powered on, the AECS Test Project runs continuously until powered off.

5.3 Monitor and Control of the AECS Test Project

Monitoring of the AECS Test Project is performed with the MonCon application. To start the MonCon application, click on the 'MonCon.bat' icon shown in Figure 14 on the AECS User's desktop. This assumes the MonCon application is built within E:\projects\test. See 6.7.3 for instructions on

Figure 14: MonCon Desktop Icon

building MonCon.

MonCon will start. The MonCon window is shown in Figure 15. MonCon provides three tabs:

- SGC to DCM: This tab shows the information sent from the SGC to the DCM. This includes the contents of all packets sent by FCC1, FCC2, FCC3, FCC4, FCC5, FCC6, and the SGC
- DCM to SGC: This tab shows the information sent from the DCM to the SGC. This includes all of the fields that the DCM can modify to affect the running AECS Test Project. See below for additional information.
- Packet Counts: This tab shows the number of packets received by the SGC. This tab is useful when debugging issues to see if a component of the system is sending packets.

All three tabs support filtering. Type the filter string in the 'Filter:' combo box at the top of the tab. Only fields matching the filter text will be displayed. For example, to only view the violations, type 'violations' in the 'Filter:' box. To clear the filter text, press the 'Clear' button. Filtering occurs dynamically as the text is typed.

The DCM to SGC tab allows the user to change the payload contents as well as the violation requests for the different components of the system. This tab adds an additional 'Edit Field:' combo box and 'Send' button below the fields list as shown in Figure 15.

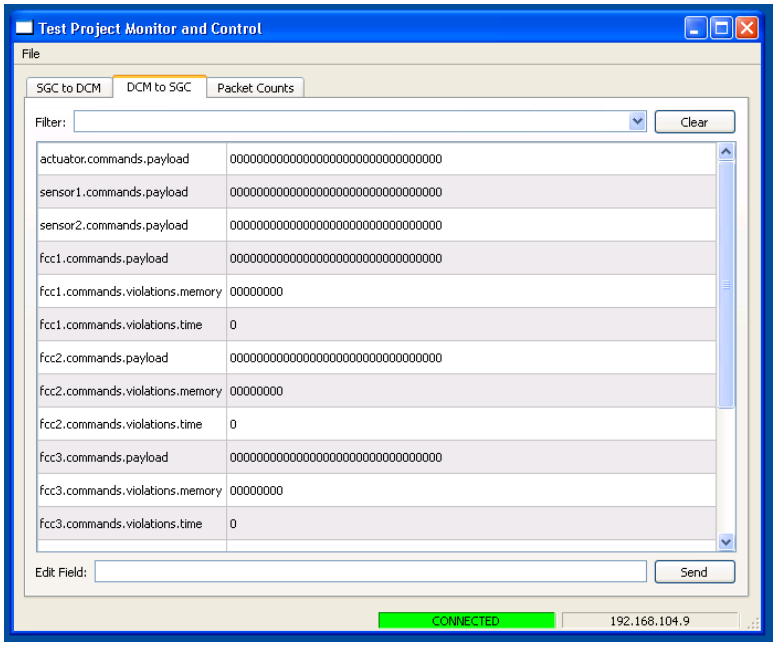


Figure 15: MonCon DCM to SGC Tab

To change a field, click on the field to change. The content of the field is placed in the 'Edit Field:' combo box. Modify the field, then click 'Send' to send it to the DCM.

The fields within MonCon update at a rate of approximately 2Hz.

5.4 Retrieving the Recorded Messages

The AECS Test Project records received messages to flash storage on the SGC-A (See Section 3.3.1). The recorded file size is limited to approximately 10MB. When the size limit is reached, recording stops. The size limit is reached approximately three minutes after power-up.

One file is recorded:

- `/sata9/recorder.pcap`: Packets received on the Avionics Bus.

The recorded file is stored in the packet capture (pcap) format. This is a common format used by many different network capture, monitoring, and analysis tools. Wireshark is provided on the DCM for analyzing the recorded files. The WVHTC Foundation modified the version of Wireshark on the DCM to support the AFDX protocol (see the *AECS User's Guide* for details on Wireshark).

The steps for retrieving and opening the AECS Test Project recorded files are:

- Start WinSCP.
- Open the 'sgc-a –sata9' stored session.
- Copy `recorder.pcap` to the DCM. Dragging to the DCM desktop should suffice.
- Close WinSCP.
- Double-click the pcap file you wish to view. You may also start Wireshark and use the 'File->Open' menu item to open the desired pcap file.
- Wireshark will start and load the file.
- View and analyze as required.

Instructions on access and using WinSCP and Wireshark are provided in the *AECS User's Guide*.

NOTE: That due to how packets are received by recorder, there is no header information available to record. Therefore, recorder puts an AECS Ethernet header on each packet which contains the virtual link encoded in a known MAC address. The WVHTC Foundation modified Wireshark will detect this header and display the packets correctly. To capture packet with header information, use the monitoring feature of the DCM and Wireshark (see the *AECS User's Guide*).

6 AECS Test Project Software Architecture

6.1 Module OS

The module OS is the core VxWorks 653 operating system. It is identical for both the FCC and SGC and only needs to be built once. The makefile for building the module OS are found in the `mos/` subdirectory. `mos/build.mk` sets the necessary variables to configure the mos build and includes `E:/aecs/build/mos.mk` to perform the actual build. `E:/aecs/build/mos.mk` is based upon the Makefile provided by WindRiver and documented in Section 22.4 of the *VxWorks 653 Configuration and Build Guide*. See the *AECS User's Guide* for additional information on `E:/aecs/build/mos.mk`.

The 'mos' rule can be used to build the module OS. Using the VxWorks 653 AECS Shell, change to `E:\projects\test` and execute:

```
make mos
```

The default rule will build the module OS if it is not already rebuilt so the above command is rarely needed. The module OS will only need to be rebuilt if the configuration options within `mos/Makefile` change or the files making up the core OS/BSP within `os/vx653` change.

6.2 Avionics Bus Configuration

6.2.1 Avionic Bus Messages

As introduced in Section 3.1, the AECS Test Project applications exchange five message types via the Avionics Bus. Table 8 lists the AECS Test Project message types and provides a brief description of each. The message ID is always the first 4 bytes of the message and allows for easy identification of the message type both programmatically and while monitoring the network traffic. The structure defining each message is found in the listed header file located in the directory of the application that is the source of the message.

Table 8: AECS Test Project Message Types

Message ID	Description	Header File
0x12345011	FCC to FCC CCDL Message	`fcc/fapp1/fccTOfccs.h`
0x12345022	FCC Response Message	`fcc/fapp1/fccTOsgc.h`
0x90000033	SGC Command Message	`sgc-a/commd/sgcTOall.h`
0xA0000044	Actuator Message	`sgc-a/actuator/actuatorTOfccs.h`
0xBC000044	Sensor Message	`sgc-b/sensor/sensorTOfccs.h`

6.2.1.1 FCC to FCC CCDL Message

Each FCC sends a FCC to FCC CCDL Message. The purpose of this message is to share received data between all FCCs for verification and processing. Table 9 lists the fields of the FCC to FCC CCDL Message.

Table 9: FCC to FCC CCDL Message Fields

Name	Type	Size (in bytes)

id	unsigned int	4
fcc.payload	unsigned char array	16
fcc.violations.time	unsigned int	4
fcc.violations.memory	void pointer	4
sensor1.count	unsigned int	4
sensor1.payload	unsigned char array	16
sensor2.count	unsigned int	4
sensor2.payload	unsigned char array	16
actuator.count	unsigned int	4
actuator.payload	unsigned char array	16

The FCC to FCC CCDL Message is defined in the `fcc/fapp1/fccTOfccs.h` header file.

6.2.1.2 FCC Response Message

Each FCC sends a FCC Response Message to commd and actuator on the SGC. The FCC Response Message contains all of the data received by the FCC during a frame. Actuator commands are represented by the FCC payload. Table 10 lists the fields of the FCC Response Message.

Table 10: FCC Response Message Fields

Name	Type	Size (in bytes)
id	unsigned int	4
local.payload	unsigned char array	16
local.violations.time	unsigned int	4
local.violations.memory	void pointer	4
peerX.payload	unsigned char array	16
peerX.violations.time	unsigned int	4
peerX.violations.memory	void pointer	4
peerY.payload	unsigned char array	16
peerY.violations.time	unsigned int	4
peerY.violations.memory	void pointer	4
sensor1.count	unsigned int	4
sensor1.payload	unsigned char array	16
sensor2.count	unsigned int	4
sensor2.payload	unsigned char array	16
actuator.count	unsigned int	4
actuator.payload	unsigned char array	16

The FCC Response Message is defined in the `fcc/fapp1/fccTOsgc.h` header file.

To allow for a common packet sent from each FCC, the terminology local, peerX, and peerY is used to designate the FCC data sent from a particular FCC. Table 11 shows the FCC peer mappings.

Table 11: FCC Peer Mappings

FCC	Peer X	Peer Y
FCC1	FCC2	FCC3
FCC2	FCC1	FCC3
FCC3	FCC1	FCC2

FCC4	FCC5	FCC6
FCC5	FCC4	FCC6
FCC6	FCC4	FCC6

6.2.1.3 SGC Command Message

The SGC sends a SGC Command Message to the FCCs, actuator, and sensors. The SGC Command Message contains the FCC payloads, sensor payloads, and actuator payloads as well as any violation commands. Table 12 lists the fields of the SGC Command Message.

Table 12: SGC Command Message Fields

Name	Type	Size (in bytes)
id	unsigned int	4
fcc1.payload	unsigned char array	16
fcc1.violations.time	unsigned int	4
fcc1.violations.memory	void pointer	4
fcc2.payload	unsigned char array	16
fcc2.violations.time	unsigned int	4
fcc2.violations.memory	void pointer	4
fcc3.payload	unsigned char array	16
fcc3.violations.time	unsigned int	4
fcc3.violations.memory	void pointer	4
fcc4.payload	unsigned char array	16
fcc4.violations.time	unsigned int	4
fcc4.violations.memory	void pointer	4
fcc5.payload	unsigned char array	16
fcc5.violations.time	unsigned int	4
fcc5.violations.memory	void pointer	4
fcc6.payload	unsigned char array	16
fcc6.violations.time	unsigned int	4
fcc6.violations.memory	void pointer	4
actuator.payload	unsigned char array	16
sensor1.payload	unsigned char array	16
sensor2.payload	unsigned char array	16

The SGC Command Message is defined in the `sgc-a/commd/sgcTOall.h` header file.

6.2.1.4 Sensor/Actuator Message

Each sensor/actuator sends a Sensor/Actuator Message to the FCCs. The Sensor/Actuator Message represents the data collected by the sensor or actuator. Table 13 lists the fields of the Sensor Message while Table 14 lists the fields of the Actuator Message.

Table 13: Sensor Message Fields

Name	Type	Size
id	unsigned int	4
sensor.count	unsigned int	4
sensor.payload	unsigned char array	16

The Sensor Message is defined in the `sgc-b/sensor/sensorTOfccs.h`.

Table 14: Actuator Message Fields

Name	Type	Size
id	unsigned int	4
actuator.count	unsigned int	4
fcc1Count	unsigned int	4
fcc2Count	unsigned int	4
fcc3Count	unsigned int	4
actuator.payload	unsigned char array	16

The Actuator Message is defined in the `sgc-a/actuator/actuatorTOfccs.h` header files. The actuator adds the FCC counts to its message. These are used to verify that the actuator can actually receive the FCC response messages.

6.2.2 AFDX/TTE Switch Connections

As discussed in the *AECS User's Guide*, each AFDX/TTE interface card (1 per FCC SBC, 2 per SGC SBC) connects to a port on an Avionics switch. Table 15 shows the AFDX/TTE connections used by the AECS Test Project.

Table 15: AECS Host to AFDX/TTE Switch Port Mapping

AECS Host	AFDX Switch Port	TTE Switch Port
FCC1	1	1
FCC2	2	2
FCC3	3	3
FCC4	4	4
FCC5	5	5
FCC6	6	6
SGC-A (Interface Card 1)	9	9
SGC-A (Interface Card 2)	10	10
SGC-B (Interface Card 3)	11	11
SGC-B (Interface Card 4)	12	12
DCM	16	Monitor

Note that we enable FCC4, FCC5, and FCC6 in the AFDX/TTE configuration. This allows FCC4, FCC5, and FCC6 requirements to be tested using the AECS Test Project.

6.2.3 Virtual Links

6.2.3.1 Virtual Link Conventions

The AECS Test Project uses a simple naming convention for virtual links. The hundreds place is the switch port of the sending host. The ones place identifies the message type from the host. Figure 16 graphically depicts the virtual link naming convention.

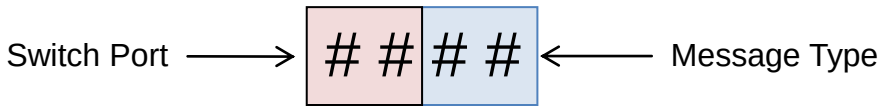


Figure 16: Virtual Link Naming Convention

For example, all messages sent from FCC1, connected to AFDX/TTE switch port 1, are 1## where ## is the message type. 101 is the CCDL message from FCC1 and 102 is the response message from FCC1. For FCC2, connected to AFDX/TTE switch port 2, the CCDL message is 201 and the response message is 202.

6.2.3.2 Virtual Link Notes

The AFDX/TTE configuration sends all AECS Test Project virtual links to port 9. Port 9 is read by the SGC router task. The router task forwards received messages to the DCM via the maintenance network and to the recorder for on-disk recording.

The AFDX (but not TTE) configuration sends all AECS Test Project virtual links to port 16. Port 16 is connected to the DCM and allows monitoring of AFDX traffic via the Wireshark application. Additional information on monitoring AFDX traffic via Wireshark is available in the *AECS User's Guide*.

The TTE switches have a built in monitoring port and thus do not need a dedicated port (and associated configuration) for monitoring. Additional information on monitoring TTE traffic via Wireshark is available in the *AECS User's Guide*.

6.2.3.3 VLs 101, 201, and 301

Figure 17 shows the virtual link diagram for VL 101, the FCC to FCC CCDL message. Note that ports 13-16 are only applicable for AFDX.

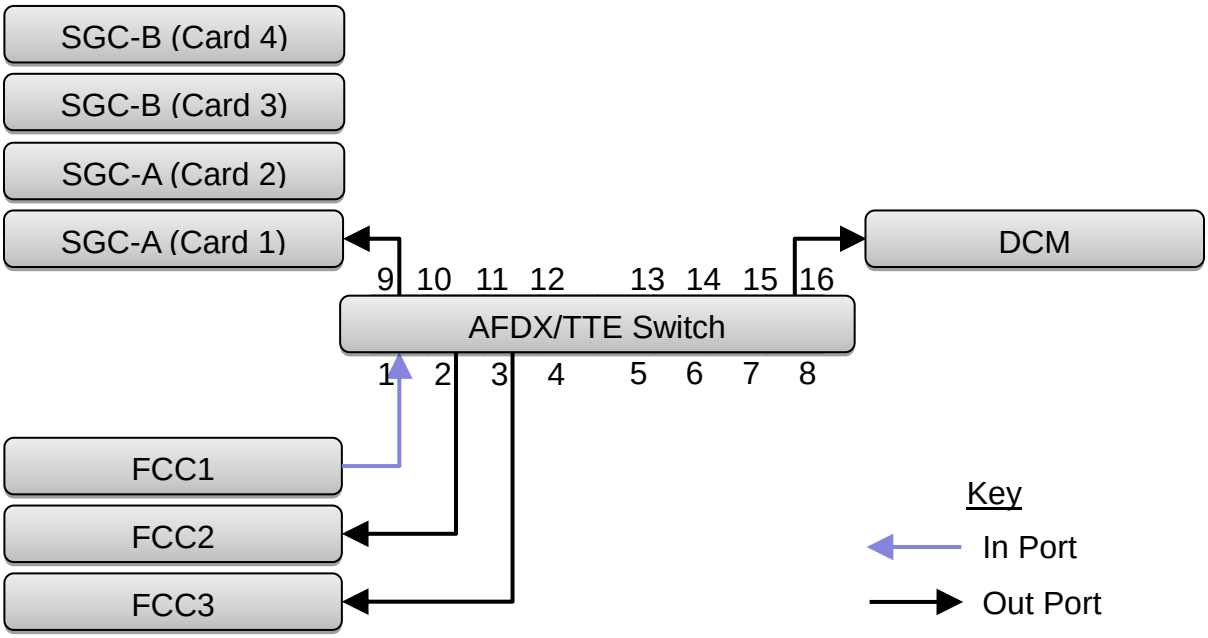


Figure 17: Virtual Link Diagram for VL 101

For VL 201, FCC2 is the in port and FCC1 is an out port. For VL 301, FCC3 is the in port and FCC1 is an out port.

6.2.3.4 VLs 102, 202, and 302

Figure 18 shows the virtual link diagram for VL 102, the FCC response message. Note that ports 13-16 are only applicable for AFDX.

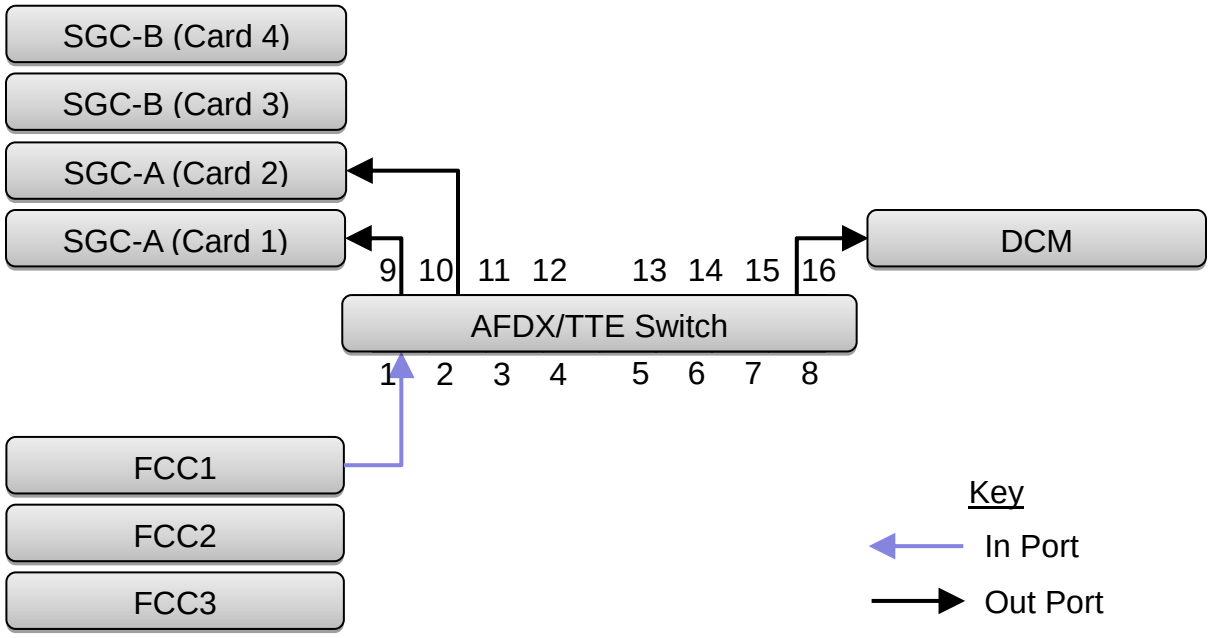


Figure 18: Virtual Link Diagram for VL 102

For VL 202, FCC2 is the in port. For VL 302, FCC3 is the in port.

6.2.3.5 VL 903

Figure 19 shows the virtual link diagram for VL 903, the SGC Command Message. Note that ports 13-16 and the DCM connection are only applicable for AFDX.

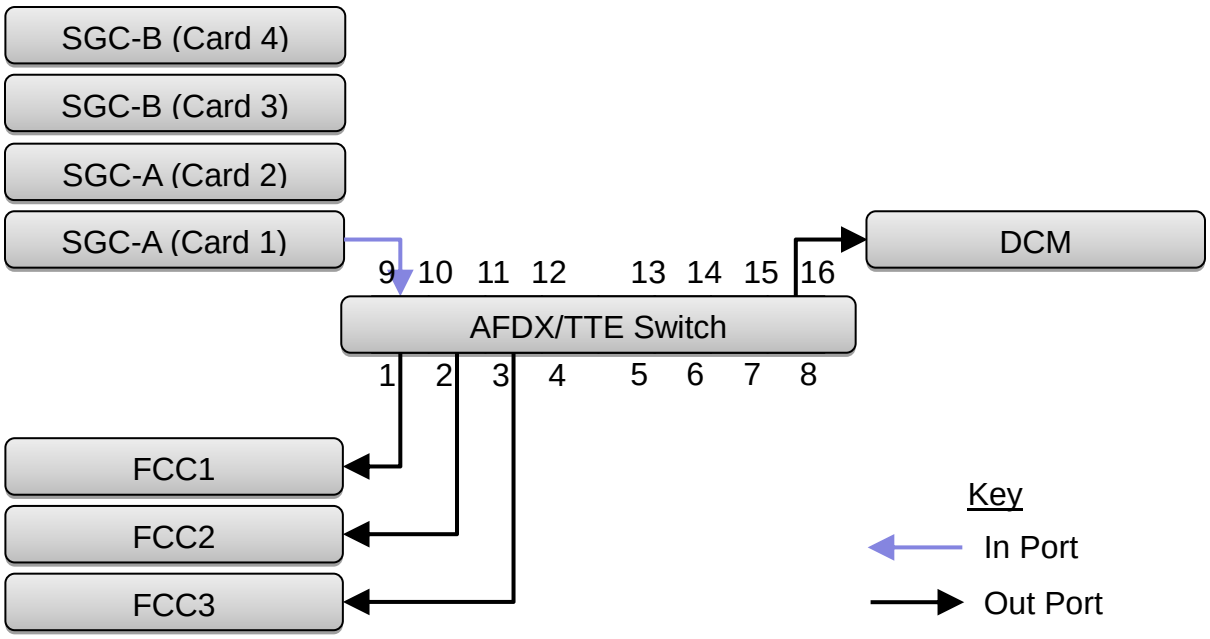


Figure 19: Virtual Link Diagram for VL 903

6.2.3.6 VL 1004, 1104, and 1204

Figure 20 shows the virtual link diagram for VL 1004, the Sensor/Actuator Message. Note that ports 13-16 are only applicable for AFDX.

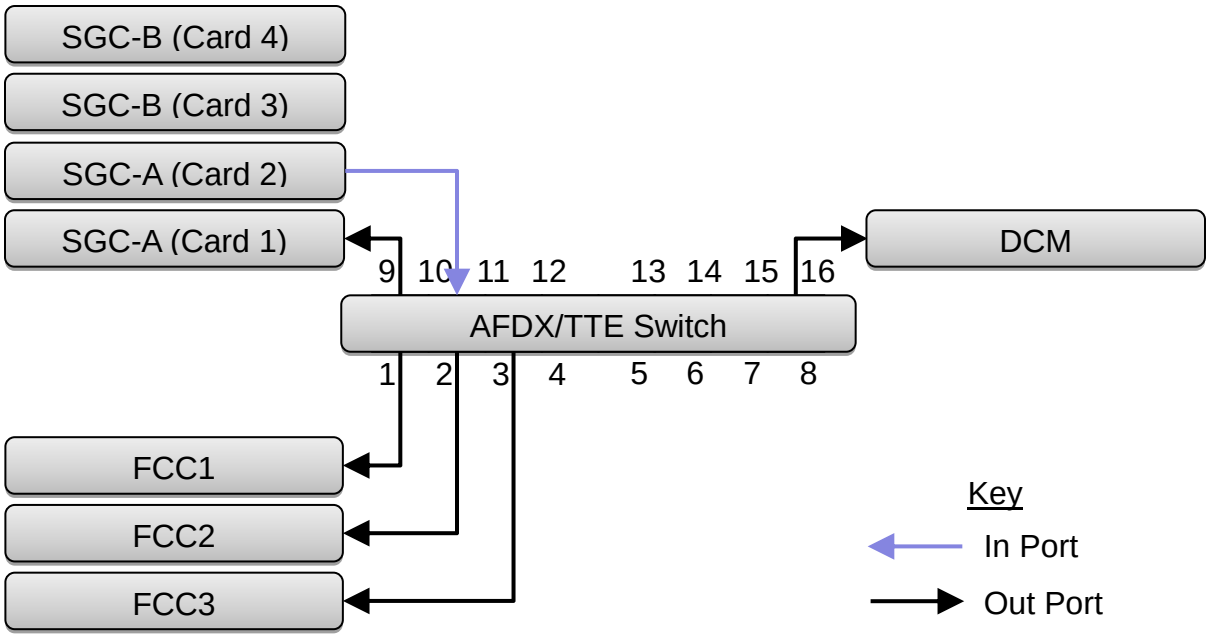


Figure 20: Virtual Link Diagram for VL 1004

For VL 1104, SGC-B (Card 3) is the in port. For VL 1204, SGC-B (Card 4) is the in port.

6.2.3.7 Virtual Link Summary

Table 16 summarizes the virtual links presented in Sections 6.2.3.3, 6.2.3.4, 6.2.3.5, and 6.2.3.6.

Table 16: Virtual Link In Ports and Out Ports

Virtual Link	Message ID	In Port	Out Port
101	0x12345011	1	2,3,9,16
102	0x12345022	1	9,10,16
201	0x12345011	2	1,3,9,16
202	0x12345022	2	9,10,16
301	0x12345011	3	1,2,9,16
302	0x12345022	3	9,10,16
401	0x12345011	4	5,6,9,16
402	0x12345022	4	9,10,16
501	0x12345011	5	4,6,9,16
502	0x12345022	5	9,10.16
601	0x12345011	6	4,5,9,16
602	0x12345022	6	9,10,16
903	0x90000033	9	1,2,3,4,5,6,16
1004	0xA0000044	10	1,2,3,4,5,6,9,16
1104	0xBC000044	11	1,2,3,4,5,6,9,16
1204	0xBC000044	12	1,2,3,4,5,6,9,16

(*) Port 16 only applicable when configuring the AFDX switch.

Note that FCC4, FCC5, and FCC6 are listed above. These FCCs were added to the AECS Test Project configuration to aid in testing the FCC4/FCC5/FCC6 related requirements.

6.2.4 AFDX Specific Configuration

This section describes the configuration and settings for the AECS Test Project specific to the AFDX Avionics Bus.

6.2.4.1 AFDX Switch Configuration

A single configuration file contains the AFDX switch configuration. This section first presents the detailed configuration information and describes the AFDX switch configuration file.

Each AFDX port has several associated configuration parameters within the switch configuration file. Table 17 lists the configuration of each AFDX port for the AECS Test Project. Omitted ports (6-8 and 13-15) are disabled in the AFDX configuration.

Table 17: AECS Test Project AFDX Port Configuration

Port	Host	Low Priority Buffer Size (pkts)	High Priority Buffer Size (pkts)	Port State (0 or 1)	Max Delay (ms)	Port Speed (10 or 100)
1	FCC1	128	128	1 (ENABLED)	10	100
2	FCC2	128	128	1 (ENABLED)	10	100
3	FCC3	128	128	1 (ENABLED)	10	100
4	FCC4	128	128	1 (ENABLED)	10	100
5	FCC5	128	128	1 (ENABLED)	10	100
6	FCC6	128	128	1 (ENABLED)	10	100
9	SGC-A (1)	128	128	1 (ENABLED)	10	100

10	SGC-A (2)	128	128	1 (ENABLED)	10	100
11	SGC-B (3)	128	128	1 (ENABLED)	10	100
12	SGC-B (4)	128	128	1 (ENABLED)	10	100
16	DCM	128	128	1 (ENABLED)	10	100

The listed values are the default values noted in the AFDX switch documentation. The defaults are adequate for the AECS Test Project. Future AECS systems, with additional and/or higher rate messages, may need to alter the default values to obtain the desired performance.

Each virtual link has several configuration parameters within the AFDX switch configuration file. These parameters define the flow of information on the AFDX bus. As the AECS Test Project has a very small number of messages, the default values for these parameters are acceptable. These parameters are standard AFDX parameters and described in *AFDX Lab-Switch 8/16/24 User Manual*. Table 18 lists the values of the virtual link parameters used in the AECS Test Project for AFDX

Table 18: Virtual Link Parameters for AFDX Configuration

Parameter	Value
Jitter	10 ms
Bag	8 ms
Minimum Packet Size	64 bytes
Maximum Packet Size	1518 bytes
Multicast Flag	1 (TRUE)
Account Sharing Group	0

If the AFDX switch detects a violation of one of the virtual link parameters listed in Table 18 or one of the port parameters listed in Table 17, a count is incremented. The violation counts can be retrieved from the AFDX switch console. See the *AECS User's Guide* for information on accessing the AFDX switch console.

Table 19 lists the virtual link priorities used in the AECS Test Project for AFDX.

Table 19: AFDX Virtual Link Priorities

Virtual Link	Priority
101	1 (HIGH)
102	0 (LOW)
201	1 (HIGH)
202	0 (LOW)
301	1 (HIGH)
302	0 (LOW)
401	1 (HIGH)
402	0 (LOW)
501	1 (HIGH)
502	0 (LOW)
601	1 (HIGH)
602	0 (LOW)
903	0 (LOW)

1004	0 (LOW)
1104	0 (LOW)
1204	0 (LOW)

The AFDX configuration listed in Table 16, Table 17, Table 18, and Table 19, is stored in an American Standard Code for Information Interchange (ASCII) text file. The file is located at `afdx/afdx.conf`. The file is divided into five sections with a comment preceding each section (comments start with the standard '#' character). The five sections are shown below. A brief description is provided followed by the associated content from `afdx.conf`. Line numbers may differ if the file has been edited by the user.

- PIN: Set to the AECS default of 16; see the *AECS User's Guide* for additional information.

```
1 #PIN
2 16
```

- Configuration Name: Set to 1_AECS_Test_Project_Configuration as this is the first and only configuration for the AECS Test Project. The switch supports multiple configurations; see the *AFDX Lab-Switch 8/16/24 User Manual* for additional information.

```
3 #CONFIG-NAME
4 1_AECS_Test_Configuration
```

- Port Settings: The port settings as described in Table 17.

```
 5 #Port-Settings
 6 #PortId;LowBufferSize;HighBufferSize;PortState;MaxDelay;PortSpeed
 7 1;128;128;1;10;100
 8 2;128;128;1;10;100
 9 3;128;128;1;10;100
10 4;128;128;1;10;100
11 5;128;128;1;10;100
12 6;128;128;1;10;100
13 7;128;128;0;10;100
14 8;128;128;0;10;100
15 9;128;128;1;10;100
16 10;128;128;1;10;100
17 11;128;128;1;10;100
18 12;128;128;1;10;100
19 13;128;128;0;10;100
20 14;128;128;0;10;100
21 15;128;128;0;10;100
22 16;128;128;1;10;100
```

- End System Settings: The end system capability is not used by the AECS Test Project; see the *AFDX Lab-Switch 8/16/24 User Manual* for information on this feature.

```
23 #EndSystem-Settings
24 #SrcMAC;SrcIP;TxVL;RxVL
25 02-00-00-0f-f0-20;10.15.252.1;4092;4090
```

- Routing/Virtual Link Settings: The routing settings as described in Table 16, the virtual link settings as described in Table 18, and the virtual link priorities as described in Table 19.

```
#Routing-VL-Settings
#InPort;OutPortList;VLID;Jitter;BAG;Lmin;Lmax;MulticastFlag;PriorityFlag;SharingID
1;2,3,9,16;101;10;8;64;1000;1;0
2;1,3,9,16;201;10;8;64;1000;1;0
3;1,2,9,16;301;10;8;64;1000;1;0
4;5,6,9,16;401;10;8;64;1000;1;0
5;4,6,9,16;501;10;8;64;1000;1;0
6;4,5,9,16;601;10;8;64;1000;1;0
1;9,10,16;102;10;8;64;1000;1;0
2;9,10,16;202;10;8;64;1000;1;0
3;9,10,16;302;10;8;64;1000;1;0
4;9,10,16;402;10;8;64;1000;1;0
5;9,10,16;502;10;8;64;1000;1;0
6;9,10,16;602;10;8;64;1000;1;0
9;1,2,3,4,5,6,9,10,11,12,16;903;10;8;64;1000;1;0
10;1,2,3,4,5,6,9,16;1004;10;8;64;1000;1;0
11;1,2,3,4,5,6,9,16;1104;10;8;64;1000;1;0
12;1,2,3,4,5,6,9,16;1204;10;8;64;1000;1;0
```

Section 5.1 describes installation of the AFDX switch configuration for the AECS Test Project. General information on configuring the AFDX switch is available in the AECS User's Guide and the *AFDX Lab-Switch 8/16/24 User Manual*.

6.2.4.2 AFDX FCC/SGC Configuration

As discussed in the *AECS User's Guide*, AFDX/TTE virtual links are created at initialization by the VxWorks 653 device drivers and remain throughout execution of the system. AFDX/TTE virtual links are configured via comma separated value (CSV) files. Each line in the CSV file represents one virtual link and the parameters required by the device driver to configure that virtual link. For AFDX, TX and RX virtual links require a different list of settings. See the *AECS User's Guide* for additional information and an overview of the AFDX/TTE device drivers.

For the AECS Test Project, most settings are identical across all virtual links and SBCs. Table 20 shows the common settings for AFDX TX virtual links.

Table 20: AFDX TX Virtual Link Common Settings

Parameter	Value
SubVLs	1
Bag	32
Maximum Frame Length	1000
Frame Buffer Size	0
MAC Source, Least Significant Word	0x00089AC0
MAC Source, Most Significant Word	0x00000200
Network Select	FDX_TX_FRAME_BOTH
Skew	100
Port Type	FDX_UDP_SAMPLING
UDP Source Port	23
UDP Destination Port	24
IP Source Address	10.1.33.1
IP Destination Address	224.224.0.33
UDP Number of Buffered Messages	1

Table 21 shows the common settings for AFDX RX virtual links.

Table 21: AFDX RX Virtual Link Common Settings

Parameter	Value
Virtual Link Range	1
Enable Mode	FDX_RX_VL_ENA_EXT
Payload Mode	FDX_PAYLOAD_FULL
TCB Index	0
Verification Mode	FDX_RX_FL_CHECK_ENA_DEFAULT
Bag	32
Jitter	100
Maximum Frame Length	1000
Maximum Skew	1000
Virtual Link Buffer Size	0x8000
Minimum Frame Length	1
Port Type	FDX_UDP_SAMPLING_PORT
UDP Number of Buffered Messages	1
UDP Source Port	23
UDP Destination Port	24
IP Source Address	10.1.33.1
IP Destination Address	224.224.0.33

For additional information on these parameters, see the *AFDX/ARINC 664 Interface Module Programmer's Guide* and *AFDX/ARINC 664 Interface Module Programmer's Reference*. Specifically, look at the parameters of the function calls *FdxCmdTxCreateVL()* and *FdxCmdTxUDPCreatePort()* for the transmit parameters and *FdxCmdRxVLControl()* and *FdxCmdRxUDPCreatePort()* for the receive parameters.

Table 22 and Table 23 show the AFDX device driver configuration for transmit and receive ports respectively for FCC1.

Table 22: AFDX TX Device Driver Configuration for FCC1

Port	Virtual Link ID	UDP Max Message Size	UDP Sampling Rate	Card Number
pseudo_fccTOfccs	101	124	50	0
pseudo_fcccTOsgc	102	200	50	0

Table 23: AFDX RX Device Driver Configuration for FCC1

Port	Virtual Link ID	UDP Max Message Size	Card Number
pseudo_sgcTOall	903	196	0
pseudo_sensor1TOfccs	1104	28	0
pseudo_sensor2TOfccs	1204	28	0
pseudo_actuatorTOfccs	1004	40	0
pseudo_fccTOfccs_peerX	201	124	0
pseudo_fccTOfccs_peerY	301	124	0

The other SBCs within the AECS Test Project have similar configuration. The configuration for each SBC can be found in the following files:

- FCC1: `fcc/afdx/fcc1/port_to_vls.csv`
- FCC2: `fcc/afdx/fcc2/port_to_vls.csv`
- FCC3: `fcc/afdx/fcc3/port_to_vls.csv`
- FCC4: `fcc/afdx/fcc4/port_to_vls.csv`
- FCC5: `fcc/afdx/fcc5/port_to_vls.csv`
- FCC6: `fcc/afdx/fcc6/port_to_vls.csv`
- SGC-A: `sgc-a/afdx/port_to_vls.csv`
- SGC-B: `sgc-b/afdx/port_to_vls.csv`

The content of `fcc/afdx/fcc1/port_to_vls.csv` is provided below for reference. Line numbers may differ if the file has been edited by the user. Other `port_to_vls.csv` are similar.

```
 1 [TX]
 2 #portName,VLId,SubVls,Bag,MaxFrameLength,FrameBufferSize,MACSourceLSLW,MACSourceMSLW,
      NetSelect,Skew,PortType,UdpSrc,UdpDst,IpSrc,IpDst,UdpMaxMessageSize,
      UdpNumBufMessages,UdpSamplingRate,card
 3 pseudo_fccTOfccs,101,1,32,1000,0,0x00089AC0,0x00000200,FDX_TX_FRAME_BOTH,100,
      FDX_UDP_SAMPLING,23,24,10.1.33.1,224.224.0.33,124,1,50,0
 4 pseudo_fccTOsgc,102,1,32,1000,0,0x00089AC0,0x00000200,FDX_TX_FRAME_BOTH,100,
      FDX_UDP_SAMPLING,23,24,10.1.33.1,224.224.0.33,200,1,50,0
 5
 6 [RX]
 7 #portName,VLId,VLRange,EnableMode,PayloadMode,TCBIndex,VerificationMode,Bag,Jitter,
       MaxFrameLength,MaxSkew,VLBufSize,MinFrameLength,PortType,UdpMaxMessageSize,
       UdpNumBufMessages,UdpSrc,UdpDst,IpSrc,IpDst,card
 8 pseudo_sgcTOall,903,1,FDX_RX_VL_ENA_EXT,FDX_PAYLOAD_FULL,0,FDX_RX_VL_CHECK_ENA_DEFAULT,
       32,100,1000,1000,0x8000,1,FDX_UDP_SAMPLING,196,1,23,24,10.1.33.1,224.224.0.33,0
 9 pseudo_sensor1TOfccs,1104,1,FDX_RX_VL_ENA_EXT,FDX_PAYLOAD_FULL,0,FDX_RX_VL_CHECK_ENA_DEFAULT,
       32,100,1000,1000,0x8000,1,FDX_UDP_SAMPLING,28,1,23,24,10.1.33.1,224.224.0.33,0
10 pseudo_sensor2TOfccs,1204,1,FDX_RX_VL_ENA_EXT,FDX_PAYLOAD_FULL,0,FDX_RX_VL_CHECK_ENA_DEFAULT,
       32,100,1000,1000,0x8000,1,FDX_UDP_SAMPLING,28,1,23,24,10.1.33.1,224.224.0.33,0
11 pseudo_actuatorTOfccs,1004,1,FDX_RX_VL_ENA_EXT,FDX_PAYLOAD_FULL,0,FDX_RX_VL_CHECK_ENA_DEFAULT,
       32,100,1000,1000,0x8000,1,FDX_UDP_SAMPLING,40,1,23,24,10.1.33.1,224.224.0.33,0
12 pseudo_fccTOfccs_peerX,201,1,FDX_RX_VL_ENA_EXT,FDX_PAYLOAD_FULL,0,FDX_RX_VL_CHECK_ENA_DEFAULT,
       32,100,1000,1000,0x8000,1,FDX_UDP_SAMPLING,124,1,23,24,10.1.33.1,224.224.0.33,0
13 pseudo_fccTOfccs_peerY,301,1,FDX_RX_VL_ENA_EXT,FDX_PAYLOAD_FULL,0,FDX_RX_VL_CHECK_ENA_DEFAULT,
       32,100,1000,1000,0x8000,1,FDX_UDP_SAMPLING,124,1,23,24,10.1.33.1,224.224.0.33,0
```

Due to the large number of columns within the AFDX CSV files, they are far easier to edit in an application that is column and CSV aware such as Excel.

6.2.5 TTE Specific Configuration

This section describes the configuration and settings specific to the TTE Avionics Bus.

The TTE network configuration resides in the `tte/network_description.xml` file within the AECS Test Project directory. This XML defines the properties of the TTE network, end-items, connections, and virtual links. In addition, there is one file per SBC to map the TTE virtual links to the VxWorks 653 queuing ports used to send/receive the TTE data within an application. These files are

named `port_to_vls.csv` and reside throughout the directory tree. Additional information on all of these configuration files is provided in the following sections.

For general information on TTE configuration see both the *AECS User's Guide* as well as the *TTE Plan*. A general understanding of TTE and TTE configuration is helpful prior to reading this section.

6.2.5.1 Network Description Attributes

The attributes for the NetworkDescription element within the TTE network description XML for the AECS Test Project define several key parameters of the network. These attributes are shown in Table 24.

Table 24: TTE Network Description Attributes

Attribute	Value
name	test
createUnknownDefaultRoutes	false
ctMarker	AB:AD:BA:BE
ctMask	FF:FF:FF:FF
enableDynamicRouting	false
redundancy	2
transmissionSpeed	1000Mbs

'name' is arbitrary and not referenced elsewhere. As all routes for the AECS Test Project are static, there is no need to create additional routes and 'createUnknownDefaultRoutes' and 'enableDynamicRouting' are both set to false. 'ctMarker' and 'ctMask' can be set to any value; the provided values are the default. Note that these values must be set correctly within the Wireshark preferences to correctly identify TTE traffic during monitoring. A 'redundancy' of 2 duplicates the switches and connections and provides for redundant bus operations. 'transmissionSpeed' is set to the highest available, as there is no reason to run at a slower speed.

The XML for the network attributes is shown below.

```
 1 <ns4:NetworkDescription
14    name                        = "test"
15    createUnknownDefaultRoutes = "false"
16    ctMarker                    = "AB:AD:BA:BE"
17    ctMask                      = "ff:ff:ff:ff"
18    enableDynamicRouting        = "false"
19    redundancy                  = "2"
20    transmissionSpeed           = "1000Mbps"
21 >
```

6.2.5.2 Periods

A single time critical period is defined within the TTE network description XML for the AECS Test Project. One 20 Hz period is sufficient to meet the AECS requirements. Table 25 lists the period attributes.

Table 25: TTE Periods for AECS Test Project

Name	Time
PERIOD_TT_20Hz	50,000,000 ns

The XML defining the single period is shown below.

```
861 <period
862    name="PERIOD_TT_20Hz"
863    time="50000000 ns"
864 ></period>
```

6.2.5.3 Synchronization Domain

A single synchronization domain is defined within the TTE network description XML for the AECS Test Project. A single domain is sufficient to meet the AECS requirements. Table 26 lists the synchronization domain attributes.

Table 26: TTE Synchronization Domain Attributes

Parameter	Value
name	syncDomain
refClusterPeriod	PERIOD_TT_20Hz
integrationCycleDuration	1,000,000 ns
falutTolerance	0FTSI_2SM
precision	5008 ns

A 'faultTolerance' of '0FTSI_2SM' allows synchronization with as few as two synchronization masters. Thus, you must have at least two SBCs running for TTE to synchronize and the network to work properly. 'integrationCycleDuration' and 'precision' are the values provided in the TTE Plan documentation; it was not necessary to change them for the AECS Test Project TTE configuration.

The XML defining the synchronization domain is shown below.

```
36 <syncDomain
37    name="syncDomain"
38    refClusterPeriod="#//@period[name='PERIOD_TT_20Hz']"
39    integrationCycleDuration="1000000 ns"
40    faultTolerance="0FTSI_2SM"
41    precision="5008 ns"
42    fullCBG="true"
43    value="0">
44    <syncPriority
45       name="syncPriority"
46       value="0"
47    ></syncPriority>
48 </syncDomain>
```

6.2.5.4 Devices

Eleven TTE devices are defined within the TTE network description XML for the AECS Test Project. One device is defined for each TTE card within the system (6 in the FCCs and 4 in the SGCs) and one device is defined for the switch. TTE Plan will create the second switch since a redundant network is defined. Table 27 lists the TTE devices.

Table 27: TTE Network Description Devices

Name	Type	Sync Role	Reference Sync Priority	Device

FCC1	EndSystem	syncMaster	syncPriority	TTE_PMC_ESys_1G
FCC2	EndSystem	syncMaster	syncPriority	TTE_PMC_ESys_1G
FCC3	EndSystem	syncMaster	syncPriority	TTE_PMC_ESys_1G
FCC4	EndSystem	syncMaster	syncPriority	TTE_PMC_ESys_1G
FCC5	EndSystem	syncMaster	syncPriority	TTE_PMC_ESys_1G
FCC6	EndSystem	syncMaster	syncPriority	TTE_PMC_ESys_1G
SGC_A1	EndSystem	syncMaster	syncPriority	TTE_PMC_ESys_1G
SGC_A2	EndSystem	syncClient	syncPriority	TTE_PMC_ESys_1G
SGC_B3	EndSystem	syncMaster	syncPriority	TTE_PMC_ESys_1G
SGC_B4	EndSystem	syncClient	syncPriority	TTE_PMC_ESys_1G
SW	Switch	syncCompressionMaster	syncPriority	TTE_Dev_Switch_12port_1G

The TTE card in each FCC and the first card in each SGC are designated as synchronization masters. The second card in each SGC is designated as a synchronization client. There was no need to have two cards connected to the same host both acting as synchronization masters. As there is only one synchronization domain (see Section 6.2.5.3), all devices share the same synchronization priority. Take care with the device name, the string must match exactly as this is how TTE Plan determines the hardware capabilities.

Each device also configures a series of ports. For an end-system, each card has three physical ports and an internal port representing the host. For a switch, there are twelve physical ports and two internal ports representing the synchronization component and the switch management component. The port names within the AECS Test Project all follow the same convention:

`PORT_<device_name>_<#|HOST|SYNC|MGMT>`

Where `<device_name>` is one of the names from Table 27. `<#>` is 1 or 2 for end systems or 1 through 12 (except for 7) for the switch. For example, the port name within the TTE network description for the host port for FCC2 is:

`PORT_FCC2_HOST`

Each end-item device must also define receive buffers for each virtual link it receives. The receive buffers are given a unique name and reference a particular virtual link. No receive buffers are defined for switches.

The XML defining end-item SGC_B4 is shown below.

```
622 <device
623    xsi:type        = "topo:EndSystem"
624    syncRole        = "syncClient"
625    refSyncPriority = "#//@syncDomain/@syncPriority[name='syncPriority']"
626    name            = "SGC_B4"
627    deviceTarget    = "TTE_PMC_ESys_1G">
628    <port
629       type = "P1"
630       name = "PORT_SGC_B4_1"
631    ></port>
632    <port
633       type = "P2"
```

```
634       name = "PORT_SGC_B4_2"
635    ></port>
636    <port
637       type = "PHOST"
638       name = "PORT_SGC_B4_HOST"
639    ></port>
640    <receiveBuffer
641       xsi:type       = "buf:CTBuffer"
642       semantic       = "sample"
643       name           = "RCVBUF_VL903_SENSOR2"
644       refVirtualLink = "#//@virtualLink[name='VL903']"
645       size           = "1"
646    ></receiveBuffer>
647 </device>
```

The switch device must correctly define the best effort routes to the management interface to allow loading of the switch outside of bootstrap mode. The XML required to correctly configure the switch is provided in the AECS User's Guide. While a different configuration is possible, the AECS environment assumes the provided configuration and the load rules will not work with a different configuration unless modified.

6.2.5.5 Physical Links

Ten physical connections are defined within the TTE network description XML for the AECS Test Project. These physical connections define the cabling between the different TTE end-items and switches. Note that physical links connect the ports of an end-item to the ports of a switch. The connections map directly to the connections described previously in Section 6.2.3.7. All connections have a 'mediaType' of 'copper' as all AECS cabling is CAT6 twisted pair.

The XML defining several physical connections is shown below.

```
812 <physicalLink
813    mediaType = "copper"
814    name      = "connection_1"
815    refPort   = "#//@device[name='FCC1']/@port[name='PORT_FCC1_1']
816                 #//@device[name='SW']/@port[name='PORT_SW_1']"
817 ></physicalLink>
818 <physicalLink
819    mediaType = "copper"
820    name      = "connection_2"
821    refPort   = "#//@device[name='FCC2']/@port[name='PORT_FCC2_1']
822                #//@device[name='SW']/@port[name='PORT_SW_2']"
823 ></physicalLink>

851 <physicalLink
852    mediaType = "copper"
853    name      = "connection_9"
854    refPort   = "#//@device[name='SGC_B3']/@port[name='PORT_SGC_B3_1']
855                 #//@device[name='SW']/@port[name='PORT_SW_11']"
856 ></physicalLink>
857 <physicalLink
858    mediaType = "copper"
859    name      = "connection_10"
860    refPort   = "#//@device[name='SGC_B4']/@port[name='PORT_SGC_B4_1']
861                 #//@device[name='SW']/@port[name='PORT_SW_12']"
862 ></physicalLink>
```

6.2.5.6 Virtual Links

Sixteen virtual links are defined within the TTE network description XML for the AECS Test Project. The attributes unique to the TTE virtual links are provided in Table 28.

Table 28: TTE Network Description Virtual Links

Name	Type	Virtual Link ID	Maximum Payload Size	Redundancy Management	Period
VL101	TTVirtualLink	101	1000	tt_redundancy	PERIOD_TT_20Hz
VL102	TTVirtualLink	102	1000	tt_redundancy	PERIOD_TT_20Hz
VL201	TTVirtualLink	201	1000	tt_redundancy	PERIOD_TT_20Hz
VL202	TTVirtualLink	202	1000	tt_redundancy	PERIOD_TT_20Hz
VL301	TTVirtualLink	301	1000	tt_redundancy	PERIOD_TT_20Hz
VL302	TTVirtualLink	302	1000	tt_redundancy	PERIOD_TT_20Hz
VL401	TTVirtualLink	401	1000	tt_redundancy	PERIOD_TT_20Hz
VL402	TTVirtualLink	402	1000	tt_redundancy	PERIOD_TT_20Hz
VL501	TTVirtualLink	501	1000	tt_redundancy	PERIOD_TT_20Hz
VL502	TTVirtualLink	502	1000	tt_redundancy	PERIOD_TT_20Hz
VL601	TTVirtualLink	601	1000	tt_redundancy	PERIOD_TT_20Hz
VL602	TTVirtualLink	602	1000	tt_redundancy	PERIOD_TT_20Hz
VL903	TTVirtualLink	903	1000	tt_redundancy	PERIOD_TT_20Hz
VL1004	TTVirtualLink	1004	1000	tt_redundancy	PERIOD_TT_20Hz
VL1104	TTVirtualLink	1104	1000	tt_redundancy	PERIOD_TT_20Hz
VL1204	TTVirtualLink	1204	1000	tt_redundancy	PERIOD_TT_20Hz

All of the virtual links are configured identically except for their sender/receiver attributes. All virtual links are time-triggered virtual links. They all have a period of 20Hz as that is the only period defined within the AECS Test Project.

Table 16 presented earlier details the sender/receivers associated with each virtual link. Note that when specifying senders and receivers, always use the host port for an end-item.

The XML defining virtual link 202 is shown below.

```
941 <virtualLink
942    name           = "VL202"
943    xsi:type       = "vl:TTVirtualLink"
944    vlid           = "202"
945    maxPayloadSize = "1000"
946    redundancyMgmt = "tt_redundancy"
947    refPeriod      = "#//@period[name='PERIOD_TT_20Hz']"
948    refSender      = "#//@device[name='FCC2']/@port[name='PORT_FCC2_HOST']"
949    refReceivers   = "#//@device[name='SGC_A1']/@port[name='PORT_SGC_A1_HOST']
950                      #//@device[name='SGC_A2']/@port[name='PORT_SGC_A2_HOST']"/>
```

6.2.5.7 Port to VLs Mappings

To send/receive TTE data from within VxWorks 653, a configuration file is placed on the flash drive which provides the TTE configuration to the TTE device driver. The TTE device driver reads the configuration file and configures the TTE hardware during OS initialization. See the *AECS User's Guide* for additional information on this topic.

The AECS build subsystem creates the TTE configuration when provided two pieces of information. First, a list of end-item names from the TTE network description file and second, a CSV file mapping virtual links to pseudo-port names.

The CSV file mapping virtual links to pseudo-port names is called `port_to_vls.csv` for each SBC. These files reside in the following locations for the AECS Test Project:

- FCC1: `fcc/tte/fcc1/port_to_vls.csv`
- FCC2: `fcc/tte/fcc2/port_to_vls.csv`
- FCC3: `fcc/tte/fcc3/port_to_vls.csv`
- FCC4: `fcc/tte/fcc4/port_to_vls.csv`
- FCC5: `fcc/tte/fcc5/port_to_vls.csv`
- FCC6: `fcc/tte/fcc6/port_to_vls.csv`
- SGC-A: `sgc-a/tte/port_to_vls.csv`
- SGC-B: `sgc-b/tte/port_to_vls.csv`

Each file lists each virtual link associated with the end-item (both TX and RX). Each virtual link is associated with a pseudo-port name, a size, and a card number (card number should always be 0 for FCCs, 0 or 1 for SGCs).

The port_to_vls.csv file for fcc4 is shown below.

```
 1 #
 2 # Defines the TTE pseudo-port to virtual link configuration for FCC4.
 3 #
 4 #portName,VLId,direction,size,card
 5 pseudo_fccTOfccs,401,TX,1000,0
 6 pseudo_fccTOsgc,402,TX,1000,0
 7 pseudo_fccTOfccs_peerX,501,RX,1000,0
 8 pseudo_fccTOfccs_peerY,601,RX,1000,0
 9 pseudo_sgcTOall,903,RX,1000,0
10 pseudo_actuatorTOfccs,1004,RX,1000,0
11 pseudo_sensor1TOfccs,1104,RX,1000,0
12 pseudo_sensor2TOfccs,1204,RX,1000,0
```

The `build.mk` files map a particular SBC to a series of end-items within the TTE network description XML. These files reside in the following locations for the AECS Test Project:

- FCC1: `fcc/tte/fcc1/build.mk`
- FCC2: `fcc/tte/fcc2/build.mk`
- FCC3: `fcc/tte/fcc3/build.mk`
- FCC4: `fcc/tte/fcc4/build.mk`
- FCC5: `fcc/tte/fcc5/build.mk`
- FCC6: `fcc/tte/fcc6/build.mk`
- SGC-A: `sgc-a/tte/build.mk`
- SGC-B: `sgc-b/tte/build.mk.csv`

The `build.mk` file for sgc-a is shown below.

```
34 #
35 # List of configured end-systems.
36 #
37 TTE_ES_LIST=tte/SGC_A1 tte/SGC_A2
38
39 #
40 # Mapping from port names to virtual links.
41 #
42 TTE_PORT_TO_VLS=port_to_vls.csv
43
44 include $(AECS)/build/ttevx.mk
```

Note the TTE_ES_LIST contains a list of tte devices. SGC_A1 and SGC_A2 must match device names from within `network_description.xml`.

6.3 Common Files

The `common` directory contains common files shared by multiple components of the AECS Test Project.

6.3.1 Header file ut.h

Unit testing infinite loops (i.e., while(1)s) is problematic as the code is designed to execute forever. To unit test a function, it must return so the unit test tool can verify its outputs. `common/ut.h` contains macros used within the AECS Test Project to support unit testing of infinite loops. The macros are documented in-line; see the file for additional information.

6.3.2 Header file log.h and source file hmLog.c

`common/log.h` and `common/hmLog.c` contain the definition and source code for logging error messages to the health monitoring subsystem. This code is used by all applications. On an error, the source code calls the LOG_HM_EVENT macro to raise an application error and print the error message to the console.

6.4 Flight Control Computers

6.4.1 FCC Application 1

Note that FCC Application 1 is abbreviated fapp1 throughout this section.

An overview of fapp1 is presented in Section 3.2.1.

6.4.1.1 Application Files

The fapp1 source directory contains the following files:

- `fcc/fapp1/build.mk`: Build configuration for fapp1.
- `fcc/fapp1/fapp1.c`: fapp1 source code.
- `fcc/fapp1/fcc.h`: Definition of common data structures.
- `fcc/fapp1/fapp1.xml`: fapp1 VxWorks 653 application description; see Section 6.4.1.3.

- `fcc/fapp1/fapp1TOfapp2.h`: Header file describing the data structure passed between fapp1 and fapp2 via a VxWorks 653 sampling port.
- `fcc/fapp1/fccTOfccs.h`: Header file describing the data structure passed between the FCCs via the Avionics Bus; see 6.2.1.1.
- `fcc/fapp1/fccTOsgc.h`: Header file describing the data structure passed between the FCCs and the SGC via the Avionics Bus; see Section 6.2.1.2.

6.4.1.2 Application Notes

Key points related to fapp1 include:

- Only uses the Application Executive (APEX) Application Programmer Interface (API) and select C APIs.
- Same source executes on all six FCCs; initialization code reads FCC number from shared data region.
- Uses the 653 deadline capability to execute at a rate of 20 Hz.
- Will transition the 653 partition to idle in the event of a serious error.
- Sends error messages to health monitor and standard out which is displayed on the target's console.

6.4.1.3 Application Description

The `fcc/fapp1/fapp1.xml` file contains the VxWorks 653 application description. The application is named 'fapp1' within this file. This name is referenced from other Extensible Markup Language (XML) files.

Table 29 shows the memory sizes selected for fapp1. The defaults suffice as fapp1 does not have significant memory usage.

Table 29: fapp1 Application Description Memory Allocation

Parameter	Value
Memory Size, BSS	0x10000
Memory Size, Text	0x10000
Memory Size, Data	0x10000
Memory Size, Read-only Data	0x10000

Table 30 shows the sampling ports allocation for fapp1. Sampling ports provide communications between fapp1 and fapp2.

Table 30: fapp1 Application Description Sampling Ports

Name	Direction	Message Size	Refresh Rate
fapp1TOfapp2	SOURCE	50	10000
fapp2TOfapp1	DESTINATION	50	10000

Table 31 shows the queuing port allocation for fapp1. Queuing ports provide communications over the Avionics bus to other FCCs and the SGC.

Table 31: fapp1 Application Description Queuing Ports

Name	Direction	Message Size	Queue Length
sgcTOall	DESTINATION	196	1
sensor1TOfccs	DESTINATION	28	1
sensor2TOfccs	DESTINATION	28	1
actuatorTOfccs	DESTINATION	40	1
fccTOfccs	SOURCE	124	1
fccTOfccs_peerX	DESTINATION	124	1
fccTOfccs_peerY	DESTINATION	124	1
fccTOsgc	SOURCE	200	1

6.4.2 FCC Application 2

Note that FCC Application 2 is abbreviated fapp2 throughout this section.

An overview of fapp2 is presented in Section 3.2.2.

6.4.2.1 Application Files

The fapp2 source directory contains the following files:

- `fcc/fapp2/build.mk`: Build configuration for fapp2.
- `fcc/fapp2/fapp2.c`: fapp2 source code.
- `fcc/fapp2/fapp2.xml`: fapp2 VxWorks 653 application description; see Section 6.4.2.3.
- `fcc/fapp2/fapp2TOfapp1.h`: Header file describing the data structure passed between fapp2 and fapp1 via a VxWorks 653 sampling port.

6.4.2.2 Application Notes

Key points related to fapp2 include:

- Only uses the APEX API and select C APIs.
- Uses the 653 deadline capability to execute at a rate of 40 Hz.
- Will transition the 653 partition to idle in the event of a serious error.
- Sends error messages to health monitor and standard out which is displayed on the target's console.

6.4.2.3 Application Description

The `fcc/fapp2/fapp2.xml` file contains the VxWorks 653 application description. The application is named 'fapp2' within this file. This name is referenced from other XML files. Table 32 shows the memory sizes selected for fapp2. The defaults suffice as fapp2 does not have significant memory usage.

Table 32: fapp2 Application Description Memory Allocation

Parameter	Value
Memory Size, BSS	0x10000
Memory Size, Text	0x10000
Memory Size, Data	0x10000

Memory Size, Read-only Data	0x10000

Table 33 shows the ports allocation for fapp2. Sampling ports provide communications between fapp1 and fapp2.

Table 33: fapp2 Application Description Sampling Ports

Name	Direction	Message Size	Refresh Rate
fapp1TOfapp2	DESTINATION	50	10000
fapp2TOfapp1	SOURCE	50	10000

6.4.3 FCC Partition OS

The `fcc/pos` directory contains the VxWorks 653 partition OS XML files. There are three files in this directory:

- `fcc/pos/build.mk`: Build configuration for the FCC partition OS.
- `fcc/pos/fccPos-api.xml`: Contains the `Shared_Library_API` element which includes the `Interface` element describing the API available to applications in the partition.
- `fcc/pos/fccPos-shlib.xml`: Contains the `SharedLibraryDescription` element which includes the `MemorySize` element describing the shared library size for the partition.

6.4.3.1 FCC Partition OS API

For the FCC partition, applications may access the APEX API and a select number of vThreads APIs (printf and memcpy for example, see `fccPos-api.xml` for the complete list). The full vThreads API is not available since a certified system would be restricted to the APEX API. A select number of vThreads APIs are used to ease development (memcpy) and provide feedback to the user (printf).

The shared library API is named 'fccPos'. This name is referenced in the PartitionDescription elements.

6.4.3.2 FCC Partition OS Shared Library

Table 34 shows the memory sizes selected for the FCC shared library.

Table 34: FCC Shared Library Description Memory Allocation

Parameter	Value
Memory Size, BSS	0x10000
Memory Size, Text	0x50000
Memory Size, Data	0x10000
Memory Size, Read-only Data	0x10000

6.4.4 FCC Image

The `fcc/image` directory contains the XML files needed to combine the previously discussed FCC components (applications and partition OSes) into an executable image. There are six files in this directory:

- `fcc/image/build.mk`: Build configuration for the FCC image.
- `fcc/image/fccImage-part1.xml`: Contains the PartitionDescription element describing the first FCC partition containing the fapp1 application.
- `fcc/image/fccImage-part2.xml`: Contains the PartitionDescripition element describing the second FCC partition containing the fapp2 application.
- `fcc/image/fccImage-pseudo.xml`: Contains the PseudoPartitionDescription element describing the pseudo-partition required for AFDX/TTE communications.
- `fcc/image/fccImage-module.xml`: Contains the Module element describing the FCC image.
- `fcc/image/image_startup`: Script executed when vxWorks 653 starts.

6.4.4.1 FCC Partition 1

`fcc/image/fccImage-part1.xml` contains the PartitionDescription element for FCC partition 1. The application is set to 'fapp1'. The shared library region is set to 'fccPos'. Table 35 lists the FCC partition 1 settings.

Table 35: FCC Partition 1 Settings

Parameter	Value
RequiredMemorySize	0x100000
PartitionHMTable	fccAppHm
maxGlobalFds	10
numFiles	0xFFFFFFFF
numDrivers	0xFFFFFFFF
isrStackSize	0xFFFFFFFF
syscallPermissions	0xFFFFFFFF
maxEventQStallDuration	INFINITE_TIME

See the *VxWorks 653 Configuration and Build Reference 2.2* for detailed information on the parameters.

FCC Partition 1 configuration also allows read-only access to the 'sd' shared data region. The shared data region allows fapp1 to retrieve the FCC number.

6.4.4.2 FCC Partition 2

`fcc/image/fccImage-part2.xml` contains the PartitionDescription element for FCC partition 2. The application is set to 'fapp2'. The shared library region is set to 'fccPos'. Table 36 lists the FCC partition 2 settings.

Table 36: FCC Partition 2 Settings

Parameter	Value
RequiredMemorySize	0x100000
PartitionHMTable	fccAppHm
numFiles	0xFFFFFFFF
numDrivers	0xFFFFFFFF
isrStackSize	0xFFFFFFFF

syscallPermissions	0xFFFFFFFF
maxEventQStallDuration	INFINITE_TIME

See the *VxWorks 653 Configuration and Build Reference 2.2* for detailed information on the parameters.

6.4.4.3 FCC Pseudo Partition

`fcc/image/fccImage-pseudo.xml` contains the PseudoPartitionDescription element for the FCC pseudo partition. This element provides the queuing port definitions used by the AFDX/TTE device driver for AFDX/TTE packets.

Table 31 shows the queuing port allocation for the pseudo partition. Note that the direction is the reverse of the direction in Table 37 and all have names have pseudo prepended as they must be unique within the XML.

Table 37: Pseudo Partition Description Queuing Ports

Name	Direction	Message Size	Queue Length
pseudo_sgcTOall	SOURCE	196	1
pseudo_sensor1TOfccs	SOURCE	28	1
pseudo_sensor2TOfccs	SOURCE	28	1
pseudo_actuatorTOfccs	SOURCE	40	1
pseudo_fccTOfccs	DESTINATION	124	1
pseudo_fccTOfccs_peerX	SOURCE	124	1
pseudo_fccTOfccs_peerY	SOURCE	124	1
pseudo_fccTOsgc	DESTINATION	200	1

6.4.4.4 FCC Scheduling

The FCCs are scheduled at a major frame rate of 20 Hz. Within the frame, the partitions execute as shown in Table 38.

Table 38: FCC Scheduling

Partition	Duration
fapp2	0.005 s
fapp1	0.030 s
fapp2	0.005 s
SPARE	0.010 s

This allows fapp2 to execute at an effective rate of 40 Hz (twice per frame) and fapp1 to execute at an effective rate of 20 Hz (once per frame). The SPARE cycles are provided to the core OS so it may execute the AFDX/TTE device driver and any other scheduled task.

6.4.4.5 FCC Module

`fcc/image/fccImage-module.xml` contains the Module element describing the FCC image. The Module element contains the following child elements:

- `CoreOS`: This defines the BSP for the module and should always be bsp.xml which is a copy of cwv183.xml.
- `Applications`: This lists the applications executed in the image. For the FCC image, these are 'fapp1' and 'fapp2'.
- `SharedDataRegions`: The FCC creates a single shared data region, 'sd'. The shared data region is used to communicate the FCC number between the core OS and fapp1. As the FCC number is stored in system memory, fapp1 is not able to access it directly.
- `SharedLibrary` Region: This lists the shared libraries used in the image. For the FCC image, there is a single shared library, 'fccPos'.
- `Partitions`: Lists the partitions for the image. For the FCC image, there are three partitions, one for each application and the pseudo partition. The partition names match the application names, 'fapp1' and 'fapp2'.
- `Schedules`: Defines the scheduling for the partitions; see Section 6.4.4.3 for additional details.
- `Connections`: Defines the port connections. For the FCC image, multiple connections are defined. One from 'fapp1' to 'fapp2', one from 'fapp2' to 'fapp1', and several connecting 'fapp1' and the pseudo-partition (these are the AFDX/TTE connections).
- `HealthMonitor`: Defines the health monitoring tables for the image.
- `Payloads`: Defines the payloads for the image.

6.4.5 FCC Startup File

The file `fcc/image/image_startup` executes within the VxWorks 653 core partition during boot-up. VxWorks 653 executes the script after OS initialization is complete. The startup file performs the following operations:

- Loads the FCC core OS code and executes fccCoreConfig().
- Loads the user partitions binaries.
- Waits for TTE synchronization (does nothing in the AFDX case).
- Starts both user partitions.

6.4.6 Core OS Components

The `fcc/core/` directory contains FCC source code that is built and loaded into the core OS. There are two source files within core:

- `fcc/core/config.c`: Contains an initialization function, fccCoreConfig(), that copies the target name to the shared data region. This is required so that fapp1 knows which FCC it is.
- `fcc/core/version.c`: Provides a function, aecsVersion(), to print a version string. Identifies the AECS Test Project software version. aecsVersion() can be called from the FCC console.

The core OS components are loaded by the FCC startup script.

6.5 Signal Generator Computer, SGC A

As noted previously, the communications daemon and recorder execute on SGC-A.

6.5.1 SGC Communications Daemon

Note that the SGC Communications Daemon is abbreviated commd throughout this section.

An overview of commd is presented in Section 3.3.1.

6.5.1.1 Application Files

The commd source directory contains the following files:

- `sgc-a/commd/build.mk`: Build configuration commd.
- `sgc-a/commd/commd.c`: commd source code; initializes the partition.
- `sgc-a/commd/commd.h`: Header file for shared commd data.
- `sgc-a/commd/router.c`: Source code for the commd router task.
- `sgc-a/commd/recorder.c`: Source code for the commd recorder task.
- `sgc-a/commd/commd.xml`: commd VxWorks 653 application description; see Section 6.5.1.3.
- `sgc-a/commd/sgcTOall.h`: Header file describing the data structure passed between commd and the other components via the Avionics Bus; see Section 6.2.1.3 for details.

6.5.1.2 Application Notes

Key points related to commd include:

- Uses the APEX API and vThreads API.
- Communications between router and recorder is via a vThreads message queue.
- recorder blocks on the vThreads message queue receive.
- router uses the 653 deadline capability to execute at a rate of 20 Hz.
- Will transition the 653 partition to idle in the event of a serious error (stops both router and recorder).
- Sends error messages to health monitor and standard out which is displayed on the target's console.

6.5.1.3 Application Description

The `sgc-a/commd/commd.xml` file contains the VxWorks 653 application description. Table 39 shows the memory sizes selected for commd. The defaults suffice as commd does not have significant memory usage.

Table 39: commd Application Description Memory Allocation

Parameter	Value
Memory Size, BSS	0x10000
Memory Size, Text	0x10000
Memory Size, Data	0x10000

Memory Size, Read-only Data	0x10000

Table 40 shows the queuing port allocation for commd.

Table 40: commd Application Description Queuing Ports

Name	Direction	Message Size	Queue Length
sgcTOall	SOURCE	196	1
fcc1TOfccs	DESTINATION	124	1
fcc2TOfccs	DESTINATION	124	1
fcc3TOfccs	DESTINATION	124	1
fcc4TOfccs	DESTINATION	124	1
fcc5TOfccs	DESTINATION	124	1
fcc6TOfccs	DESTINATION	124	1
fcc1TOsgc	DESTINATION	200	1
fcc2TOsgc	DESTINATION	200	1
fcc3TOsgc	DESTINATION	200	1
fcc4TOsgc	DESTINATION	200	1
fcc5TOsgc	DESTINATION	200	1
fcc6TOsgc	DESTINATION	200	1
sensor1TOfccs	DESTINATION	28	1
sensor2TOfccs	DESTINATION	28	1
actuatorTOfccs	DESTINATION	40	1

6.5.2 SGC Actuator

An overview of actuator is presented in Section 3.3.2.

6.5.2.1 Application Files

The actuator source directory contains the following files:

- `sgc-a/actuator/build.mk`: Build configuration actuator.
- `sgc-a/actuator/actuator.c`: actuator source code.
- `sgc-a/actuator/actuator.h`: Header file for shared actuator data.
- `sgc-a/actuator/actuator.xml`: actuator VxWorks 653 application description; see Section 6.5.2.3.
- `sgc-a/actuator/actuatorTOfccs.h`: Header file describing the data structure passed between actuator and the FCCs via the Avionics Bus; see Section 6.2.1.4 for details.

6.5.2.2 Application Notes

Key points related to actuator include:

- Only uses the APEX API.
- Uses the 653 deadline capability to execute at a rate of 20 Hz.
- Will transition the 653 partition to idle in the event of a serious error.

- Sends error messages to health monitor and standard out which is displayed on the target's console.

6.5.2.3 Application Description

The `sgc-a/actuator/actuator.xml` file contains the VxWorks 653 application description. Table 41 shows the memory sizes selected for actuator. The defaults suffice as actuator does not have significant memory usage.

Table 41: actuator Application Description Memory Allocation

Parameter	Value
Memory Size, BSS	0x10000
Memory Size, Text	0x10000
Memory Size, Data	0x10000
Memory Size, Read-only Data	0x10000

Table 42 shows the queuing port allocation for actuator.

Table 42: actuator Application Description Queuing Ports

Name	Direction	Message Size	Queue Length
actuatorTOfccs	SOURCE	196	1
fcc1TOsgc	DESTINATION	200	1
fcc2TOsgc	DESTINATION	200	1
fcc3TOsgc	DESTINATION	200	1
sgcTOall	DESTINATION	196	1

6.5.3 SGC-A Partition OS

The `sgc-a/pos` directory contains the VxWorks 653 partition OS XML files. There are three files in this directory:

- `sgc-a/pos/build.mk`: Build configuration for SGC-A partition OS.
- `sgc-a/pos/sgcPos-api.xml`: Contains the `Shared_Library_API` element which includes the `Interface` element describing the API available to applications in the partition.
- `sgc-a/pos/sgcPos-shlib.xml`: Contains the `SharedLibraryDescription` element which includes the `MemorySize` element describing the shared library size for the partition.

6.5.3.1 SGC-A Partition OS API

For the SGC-A partition, applications may access the APEX API and the vThreads API. The SGC-A applications require APEX for normal operations and commd requires the vThreads API for the vThreads message queues used to communicate between the router and recorder. recorder requires the vThreads API for I/O operations.

The shared library API is named 'sgcPos'. This name is referenced in the PartitionDescription elements.

6.5.3.2 SGC Partition OS Shared Library

Table 43 shows the memory sizes selected for the SGC shared library.

Table 43: SGC Shared Library Description Memory Allocation

Parameter	Value
Memory Size, BSS	0x10000
Memory Size, Text	0x50000
Memory Size, Data	0x10000
Memory Size, Read-only Data	0x10000

6.5.4 SGC Image

The `sgc-a/image` directory contains the XML files needed to combine the previously discussed SGC components (applications and partition OSes) into an executable image. There are six files in this directory:

- `sgc-a/image/build.mk`: Configures the SGC-A image build.
- `sgc-a/image/sgcImage-part1.xml`: Contains the PartitionDescription element describing SGC-A's first partition containing the commd application.
- `sgc-a/image/sgcImage-part2.xml`: Contains the PartitionDescripition element describing SGC-A's second partition containing the actuator application.
- `sgc-a/image/sgcImage-pseudo.xml`: Contains the PseudoPartitionDescripition element describing SGC-A's pseudo partition.
- `sgc-a/image/sgcImage-module.xml`: Contains the Module element describing the SGC image.
- `fcc/image/image_startup`: Script executed when vxWorks 653 starts.

6.5.4.1 SGC-A Partition 1

`sgc-a/image/sgcImage-part1.xml` contains the PartitionDescription element for SGC-A partition 1. The application is set to 'commd'. The shared library region is set to 'sgcPos'. Table 44 lists the SGC-A partition 1 settings.

Table 44: SGC-A Partition 1 Settings

Parameter	Value
RequiredMemorySize	0x100000
PartitionHMTable	sgcAppHm
maxGlobalFds	4
numFiles	0xFFFFFFFF
numDrivers	0xFFFFFFFF
isrStackSize	0xFFFFFFFF
syscallPermissions	0xFFFFFFFF
maxEventQStallDuration	INFINITE_TIME

maxGlobalFds must be set to at least 2 for the recorder flash open() and maintenance network open() to succeed in obtaining FDs. See the *VxWorks 653 Configuration and Build Reference 2.2* for detailed information on the parameters.

SGC-A Partition 1 configuration also allows read-write access to the 'sgcSd' shared data region. The shared data region allows router to exchange message data with the core OS. The DCM accesses the shared data region via the SGC console. This allows communications between MonCon and router.

6.5.4.2 SGC-A Partition 2

`sgc-a/image/sgcImage-part2.xml` contains the PartitionDescription element for SGC-A partition 2. The application is set to 'actuator. The shared library region is set to 'sgcPos'. Table 45 lists the SGC-A partition 2 settings.

Table 45: SGC-A Partition 2 Settings

Parameter	Value
RequiredMemorySize	0x100000
PartitionHMTable	sgcAppHm
maxGlobalFds	1
numFiles	0xFFFFFFFF
numDrivers	0xFFFFFFFF
isrStackSize	0xFFFFFFFF
syscallPermissions	0xFFFFFFFF
maxEventQStallDuration	INFINITE_TIME

See the *VxWorks 653 Configuration and Build Reference 2.2* for detailed information on the parameters.

6.5.4.3 SGC-A Pseudo-Partition

`sgc-a/image/sgcImage-pseudo.xml` contains the PseudoPartitionDescription element for the SGC-A pseudo partition. This element provides the queuing port definitions used by the AFDX/TTE device driver for AFDX/TTE packets.

Table 46 shows the queuing port allocation for the pseudo-partition. Note that the direction is the reverse of the direction in Table 40 and Table 42 and all have names have pseudo prepended and either commd/actuator appended as they must be unique within the XML.

Table 46: Pseudo Partition Description Queuing Ports

Name	Direction	Message Size	Queue Length
pseudo_sgcTOall_commd	DESTINATION	196	1
pseudo_fcc1TOfccs_commd	SOURCE	124	1
pseudo_fcc2TOfccs_commd	SOURCE	124	1
pseudo_fcc3TOfccs_commd	SOURCE	124	1
pseudo_fcc4TOfccs_commd	SOURCE	124	1
pseudo_fcc5TOfccs_commd	SOURCE	124	1
pseudo_fcc6TOfccs_commd	SOURCE	124	1
pseudo_fcc1TOsgc_commd	SOURCE	200	1
pseudo_fcc2TOsgc_commd	SOURCE	200	1

pseudo_fcc3TOsgc_commd	SOURCE	200	1
pseudo_fcc4TOsgc_commd	SOURCE	200	1
pseudo_fcc5TOsgc_commd	SOURCE	200	1
pseudo_fcc6TOsgc_commd	SOURCE	200	1
pseudo_sensor1TOfccs_commd	SOURCE	28	1
pseudo_sensor2TOfccs_commd	SOURCE	28	1
pseudo_actuatorTOfccs_commd	SOURCE	40	1
pseudo_actuatorTOfccs_actuator	DESTINATION	196	1
pseudo_fcc1TOsgc_actuator	SOURCE	200	1
pseudo_fcc2TOsgc_actuator	SOURCE	200	1
pseudo_fcc3TOsgc_actuator	SOURCE	200	1
pseudo_sgcTOall_actuator	SOURCE	196	1

6.5.4.4 SGC-A Scheduling

The SGC-A is scheduled at a major frame rate of 20 Hz. Within the frame, the partitions execute as shown in Table 47.

Table 47: SGC-A Scheduling

Partition	Duration
commd	0.020 s
actuator	0.020 s
SPARE	0.010

commd and actuator both execute at 20 Hz. The SPARE cycles are provided to the core OS so it may execute the AFDX/TTE device drivers and any other schedule task.

6.5.4.5 SGC-A Module

`sgc-a/image/sgcImage-module.xml` contains the Module element describing the SGC-A image. The Module element contains the following child elements:

- `CoreOS`: This defines the BSP for the module and should always be bsp.xml which is a copy of cwv183.xml.
- `Applications`: This lists the applications executed in the image. For the SGC-A image, these are 'commd' and 'actuator'.
- `SharedDataRegions`: The SGC-A creates a single shared data region, 'sgcSd'. The shared data region is used to communicate data between router and the core OS where it may be accessed by the DCM via the console.
- `SharedLibrary` Region: This lists the shared libraries used in the image. For the SGC-A image , there is a single shared library, 'sgcPos'.
- `Partitions`: Lists the partitions for the image. For the SGC-A image, there are three partitions, one for each application and the pseudo partition. The partition names match the application names, 'commd' and 'actuator'.
- `Schedules`: Defines the scheduling for the partitions. See Section 6.5.4.3 for additional details.

- `Connections`: Defines the port connections. For the SGC-A image, there are multiple connections between the 'commd' partition and the 'pseudo' partition and the 'actuator' partition and the 'pseudo' partition.
- `HealthMonitor`: Defines the health monitoring tables for the image.
- `Payloads`: Defines the payloads for the image.

6.5.5 SGC-A Startup File

The file `sgc-a/image/image_startup` executes within the VxWorks 653 core partition during boot-up. VxWorks 653 executes the script after OS initialization is complete. The startup file performs the following operations:

- Loads the SGC-A core OS code and executes sgcSdInit().
- Loads the user partitions binaries.
- Waits for TTE synchronization (does nothing in the AFDX case).
- Starts both user partitions.

6.5.6 Core OS Components

The `sgc-a/core`/ directory contains SGC-A source code that is built and loaded into the core OS. There are three source files within core:

- `sgc-a/core/dcmCli.c`: Contains the functions used by the DCM to query the current state of the system and set payload and violations. These functions set and read data from the shared data region.
- `sgc-a/core/sgcSd.c`: Contains an initialization function, sgcSdInit(), that initializes the shared data region used to copy data between router and the core OS.
- `sgc-a/core/version.c`: Provides a function, aecsVersion(), to print a version string. Identifies the AECS Test Project software version. aecsVersion() can be called from the SGC-A console.

The SGC-A core OS components are loaded by the SGC-A startup script.

The following functions are available from the VxWorks console prompt on the SGC-A:

- aecsVersion(): Prints the AECS Test Project version.
- sgcSdAddrs(): Prints the address in the shared data region for each FCC/SGC packet. The printed text contains the console command to dump the data region. This provides a low level debug routine for use during troubleshooting.
- dcmDump_sgcTOdcm(): Dumps the contents of all messages sent from the SGC to the DCM.
- dcmDump_dcmTOsgc(): Dumps the contents of all messages sent from the DCM to the SGC.
- dcmDump_counts(): Dumps the packet counts of all messages.
- dcmSet_fcc<#>Time(): Sets the time violation for an FCC; replace <#> with 1-6.
- dcmSet_fcc<#>Memory(): Sets the memory violation for an FCC; replace <#> with 1-6.
- dcmSet_fcc<#>Payload(): Sets the payload for an FCC; replace <#> with 1-6.

- dcmSet_actuatorPayload(): Sets the actuator payload.
- dcmSet_sensor1Payload(): Sets the sensor1 payload.
- dcmSet_sensor2Payload(): Sets the sensor2 payload.

6.6 Signal Generator Computer, SBC B

6.6.1 SGC Sensor

An overview of sensor is presented in Section 3.3.2.

6.6.1.1 Application Files

The sensor source directory contains the following files:

- `sgc-b/sensor/build.mk`: Build configuration for the sensor application.
- `sgc-b/sensor/sensor.c`: sensor source code.
- `sgc-b/sensor/sensor.h`: Header file for shared sensor1 data.
- `sgc-b/sensor/sensor.xml`: sensor VxWorks 653 application description; see Section 6.6.1.3.
- `sgc-b/sensor/sensorTOfccs.h`: Header file describing the data structure passed between sensor and the FCCs via the Avionics Bus, see Section 6.2.1.4 for details.

6.6.1.2 Application Notes

Key points related to sensor include:

- Only uses the APEX API.
- Uses the 653 deadline capability to execute at a rate of 20 Hz.
- Will transition the 653 partition to idle in the event of a serious error.
- Sends error messages to health monitor and standard out which is displayed on the target's console.

6.6.1.3 Application Description

The `sgc-b/sensor/sensor.xml` file contains the VxWorks 653 application description. Table 48 shows the memory sizes selected for sensor. The defaults suffice as sensor does not have significant memory usage.

Table 48: sensor Application Description Memory Allocation

Parameter	Value
Memory Size, BSS	0x10000
Memory Size, Text	0x10000
Memory Size, Data	0x10000
Memory Size, Read-only Data	0x10000

Table 49 shows the queuing ports allocation for the sensor partition.

Table 49: sensor Application Description Queuing Ports

Name	Direction	Message Size	Queue Length
sgcTOall	DESTINATION	196	1
sensorTOfccs	SOURCE	28	1

6.6.2 SGC-B Partition OS

The `sgc-b/pos` directory contains the VxWorks 653 partition OS XML files. There are three files in this directory:

- `sgc-b/pos/build.mk`: Build configuration for SGC-B partition OS.
- `sgc-b/pos/sgcPos-api.xml`: Contains the `Shared_Library_API` element which includes the `Interface` element describing the API available to applications in the partition.
- `sgc-b/pos/sgcPos-shlib.xml`: Contains the `SharedLibraryDescription` element which includes the `MemorySize` element describing the shared library size for the partition.

6.6.2.1 SGC-B Partition OS API

For the SGC partition, applications may access the APEX API and the full vThreads API.

The shared library API is named 'sgcPos'. This name is referenced in the PartitionDescription elements.

6.6.2.2 SGC-B Partition OS Shared Library

Table 50 shows the memory sizes selected for the SGC shared library.

Table 50: SGC-B Shared Library Description Memory Allocation

Parameter	Value
Memory Size, BSS	0x10000
Memory Size, Text	0x50000
Memory Size, Data	0x10000
Memory Size, Read-only Data	0x10000

6.6.3 SGC-B Image

The `sgc-b/image` directory contains the XML files needed to combine the previously discussed SGC components (applications and partition OSes) into an executable image. There are six files in this directory:

- `sgc-b/image/build.mk`: Configures the SGC-B image build.
- `sgc-b/image/sgcImage-part1.xml`: Contains the PartitionDescription element describing SGC-b's first partition containing the first sensor application.
- `sgc-b/image/sgcImage-part2.xml`: Contains the PartitionDescripition element describing SGC-b's second partition containing the second sensor application.
- `sgc-b/image/sgcImage-pseudo.xml`: Contains the PseudoPartitionDescripition element describing SGC-B's pseudo partition.

- `sgc-b/image/sgcImage-module.xml`: Contains the Module element describing the SGC image.
- `sgc-b/image/image_startup`: Script executed when vxWorks 653 starts.

6.6.3.1 SGC-B Partition 1

`sgc-b/image/sgcImage-part1.xml` contains the PartitionDescription element for SGC-B partition 1. The application is set to 'sensor'. The shared library region is set to 'sgcPos'. Table 51 lists the SGC-B partition 1 settings.

Table 51: SGC-B Partition 1 Settings

Parameter	Value
RequiredMemorySize	0x100000
PartitionHMTable	sgcAppHm
maxGlobalFds	4
numFiles	0xFFFFFFFF
numDrivers	0xFFFFFFFF
isrStackSize	0xFFFFFFFF
syscallPermissions	0xFFFFFFFF
maxEventQStallDuration	INFINITE_TIME

See the *VxWorks 653 Configuration and Build Reference 2.2* for detailed information on the parameters.

6.6.3.2 SGC-B Partition 2

`sgc-b/image/sgcImage-part2.xml` contains the PartitionDescription element for SGC-B partition 2. The application is set to 'sensor'. The shared library region is set to 'sgcPos'. Table 52 lists the SGC-B partition 2 settings.

Table 52: SGC-B Partition 2 Settings

Parameter	Value
RequiredMemorySize	0x100000
PartitionHMTable	sgcAppHm
maxGlobalFds	1
numFiles	0xFFFFFFFF
numDrivers	0xFFFFFFFF
isrStackSize	0xFFFFFFFF
syscallPermissions	0xFFFFFFFF
maxEventQStallDuration	INFINITE_TIME

See the *VxWorks 653 Configuration and Build Reference 2.2* for detailed information on the parameters.

6.6.3.3 SGC-B Pseudo-Partition

`sgc-b/image/sgcImage-pseudo.xml` contains the PseudoPartitionDescription element for the SGC-B pseudo partition. This element provides the queuing port definitions used by the AFDX/TTE device driver for AFDX/TTE packets.

Table 53 shows the queuing port allocation for the pseudo-partition. Note that the direction is the reverse of the direction in Table TBD and all have names have pseudo prepended and specify either sensor1/sensor2 as they must be unique within the XML.

Table 53: Pseudo Partition Description Queuing Ports

Name	Direction	Message Size	Queue Length
pseudo_sgcTOall_sensor1	SOURCE	196	1
pseudo_sensor1TOfccs	DESTINATION	28	1
pseudo_sgcTOall_sensor2	SOURCE	196	1
pseudo_sensor2TOfccs	DESTINATION	28	1

6.6.3.4 SGC-B Scheduling

The SGC-B is scheduled at a major frame rate of 20 Hz. Within the frame, the partitions execute as shown in Table 54.

Table 54: SGC-B Scheduling

Partition	Duration
commd	0.020 s
actuator	0.020 s
SPARE	0.010

commd and actuator both execute at 20 Hz. The SPARE cycles are provided to the core OS so it may execute the AFDX/TTE device drivers and any other scheduled task.

6.6.3.5 SGC-B Module

`sgc-b/image/sgcImage-module.xml` contains the Module element describing the SGC-B image. The Module element contains the following child elements:

- `CoreOS`: This defines the BSP for the module and should always be bsp.xml which is a copy of cwv183.xml.
- `Applications`: This lists the applications executed in the image. For the SGC-A image, these are ‘sensor1 and ‘sensor2.
- `SharedLibrary` Region: This lists the shared libraries used in the image. For the SGC-A image , there is a single shared library, ‘sgcPos’.
- `Partitions`: Lists the partitions for the image. For the SGC-B image, there are three partitions, one for each application and the pseudo partition. The partition names match the application names, ‘sensor1’ and ‘sensor2’.
- `Schedules`: Defines the scheduling for the partitions. See Section 6.5.4.3 for additional details.
- `Connections`: Defines the port connections. For the SGC-B image, there are multiple connections between the ‘sensor1 partition and the ‘pseudo’ partition and the ‘sensor2’ partition and the ‘pseudo’ partition.
- `HealthMonitor`: Defines the health monitoring tables for the image.

- `Payloads`: Defines the payloads for the image.

6.6.4 SGC-B Startup File

The file `sgc-b/image/image_startup` executes within the VxWorks 653 core partition during boot-up. VxWorks 653 executes the script after OS initialization is complete. The startup file performs the following operations:

- Loads the SGC-B core OS code.
- Loads the user partitions binaries.
- Waits for TTE synchronization (does nothing in the AFDX case).
- Starts both user partitions.

6.6.5 Core OS Components

The `sgc-b/core/` directory contains SGC-B source code that is built and loaded into the core OS. There are three source files within core:

- `sgc-b/core/version.c`: Provides a function, aecsVersion(), to print a version string. Identifies the AECS Test Project software version. aecsVersion() can be called from the SGC-B console.

The SGC-B core OS components are loaded by the SGC-B startup script.

6.7 Monitor and Control Application

6.7.1 Application Files

The MonCon source directory contains the following files:

- `dcm/moncon/moncon.pro`: Rules for building MonCon application.
- `dcm/moncon/moncon.cpp`: MonCon source file.
- `dcm/moncon/moncon.h`: MonCon header file.
- `dcm/moncon/moncon.qrc`: Qt resource file for Moncon.
- `dcm/moncon/moncon.ui`: Qt user interface definition file for MonCon.
- `dcm/moncon/moncon.png`: Application icon used by MonCon.

6.7.2 Application Notes

Key points related to MonCon incude:

- MonCon communicates with the SGC via the console port. Thus, when MonCon is running, the SGC console port is not available for use.
- MonCon executes the commands listed in Section 6.5.6 at the console prompt to retrieve its information.
- MonCon uses the Cygwin telnet to communicate with the SGC.
- MonCon updates the display at approximately 2 Hz.

6.7.3 Building MonCon

The following instructions detail building the MonCon application in the `dcm/moncon/` directory:

1. Start Qt Creator as described in the *AECS User's Guide*.
2. Click 'Open Project …'
3. Navigate to `dcm/moncon/`
4. Select `moncon.pro`
5. Select 'Build->Build All' from the menubar.

Note that the MonCon in `E:\projects\test\dcm\moncon` is executed when the user clicks MonCon.bat icon on the user's desktop. Modifying this MonCon will alter the behavior of MonCon on the desktop.

If you receive a 'Permission Denied' error when building MonCon, check to see if an instance of MonCon is running. You may not rebuild a Windows application while the application is running from the same location.

Appendix A Acronyms and Abbreviations

AE	Assessment Environment
AECS	Assessment Environment for Complex Systems
AFDX	Avionics Full-Duplex Switched Ethernet
APEX	Application Executive
API	Application Programmer's Interface
ARMD	Aeronautics Research Mission Directorate
ASCII	American Standard Code for Information Interchange
BSP	Board Support Package
CCDL	Cross Channel Data Link
commd	Communications Daemon
CSV	Comma Separated Values
DCM	Display Computer and Monitor
DFRC	Dryden Flight Research Center
EEPROM	Electrically Erasable Programmable Read-Only Memory
fapp1	FCC Application 1
fapp2	FCC Application 2
FCC	Flight Control Computer
HG	Hardware Guide
IMA	Integrated Modular Avionics Architecture
IPC	Inter-Process Communications
JPDO	Joint Planning and Development Office
MonCon	Monitor and Control User Interface
MSG	Mission Systems Group
NASA	National Aeronautics and Space Administration
NextGen	Next Generation Air Transport System
NIC	Network Interface Card
OS	Operating System
pcap	Protocol Capture
RTF	Rich Text Format
SATA	Serial Advanced Technology Attachment
SG	Software Guide
SGC	Signal Generator Computer
SUT	System Under Test
TTE	Time-Triggered Ethernet
TSP	Test Setup and Procedures
TR	Test Results
UDP	User Datagram Protocol
UG	User's Guide
URL	Uniform Resource Locator
V&VRTB	Verification and Validation Research Test Bench
VVFCS	Verification and Validation of Flight-Critical Systems
VL	Virtual Link
WVHTC	West Virginia High Technology Consortium
XML	Extensible Markup Language

Appendix B Minimal Projects

Appendix B.1 Minimal FCC Project

The Minimal FCC Project, referred to as minfcc from this point forward, implements a basic 'Hello World' type project for demonstrating AECS. The minfcc project executes on a single SBC and prints a message to the VxWorks 653 console at a rate of 0.5 Hz. The minfcc project requires a single SBC and can be run independent of the full AECS system.

The minfcc project is stored within the AECS subversion repository. Within the repository, it is located at:

```
/trunk/projects/minfcc
```

This directory is considered the 'root' of the minfcc project. It can be checked out on the DCM using the Uniform Resource Locator (URL):

```
https://localhost/svn/AECS/trunk/projects/minfcc
```

A copy of the minfcc project resides on the DCM in the directory:

```
E:\projects\minfcc
```

A copy of the subversion repository can also be found on the AECS Release DVD and from the AECS Collaborative Website. The minfcc project can be retrieved from both of these locations as well.

The root of the minfcc project directory tree contains the following files and subdirectories:

- `Makefile`: The top-level Makefile for the minfcc project.
- `app`: The application source code resides within this directory.
- `build.mk`: Defines the build for the minfcc project.
- `mos`: Subdirectory containing the module OS.
- `pos`: .Subdirectory containing the partition OS.
- `sbc.mk`: Contains the SBC information for installing the image.

To build the minfcc project from the command shell, start an AECS VxWorks 653 Command Shell as discussed in the *AECS User's Guide*. Note that this shell is slightly different than a standard VxWorks 653 Command Shell as the WIND_PATH environment variable must point to the AECS VxWorks kernel directory.

From the command shell, change to the minfcc project directory.

```
E:
cd \projects\minfcc
```

Once in the proper directory, execute the make command:

```
make
```

The build subsystem determines which components are out of date and builds them as needed. The build will stop if any errors are detected. Once the system is built, it must be installed to the targets.

Prior to installing the minfcc project, the SBC IP address and flash method must be set within `sbc.mk`. Edit `sbc.mk` and uncomment the SBC_FLASH_METHOD and SBC_IP_ADDR lines for the SBC to be used. If the IP address is not listed (i.e. it is not one of the delivered IP addresses), simply add a line with the desired IP address.

Once SBC_FLASH_METHOD and SBC_IP_ADDR are set correctly, install the minfcc project from the command shell (assumes the current directory is set as noted above) with the commands:

```
make flash
make upload
```

Restart the SBC to start the project. Open the console to the SBC. The following text should be displayed:

```
[DEBUG] In app while loop [n].
[DEBUG] In app while loop [n + 1]
[DEBUG] In app while loop [n + 2]
```

Where n is an integer value.

The following list provides some brief tips on modifying the minfcc project.

- To change the SBC the project is installed to, edit `sbc.mk`.
- To change the message, edit `app/app.c`, line 57. Additional code here will be executed at the application rate.
- To change the rate at which the task executes, edit the `PERIOD` constant in `usrAppInit()` in `app/app.c`. The `Schedule` element within `image/fccImage-module.xml` also has elements which affect the rate.
- To add a source file to the app partition, create the file in the `app` directory and add it to the list of source files variable, `APP_SRCS`, on line 64 of `app/build.mk`.
- To change the memory allocated to the application, edit the attributes of the `MemorySize` element within `app/app.xml`.

Adding a new application and partition requires quite a few steps (the VxWorks 653 documentation provides additional details on many of these steps):

- Create a new subdirectory to contain the application code (name it something other than app). Copy `app/app.xml` and `app/app.c` to the new directory changing their names to match the new directory name. Copy `app/build.mk` to the new directory (do not change its name).
- Update the new source file as desired.
- Update the new XML file if desired.
- As long as the new source file name and XML file name match the directory name, the new `build.mk` does not need to be modified as these are the default values.

- Edit the top level `build.mk` file and add the new subdirectory name to `SUBDIRS` on line 45. This will add the new application to the build.
- Make a copy of `image/fccImage-part1.xml` and change its name to something different. Edit this new file and change the `NameRef` attribute in the `Application` element on line 45 to match the name of the new application.
- Edit image/fccImage-module.xml. Within the `Applications` element, create a new `Application` element and change its name and included XML to match the application name used above. Within the `Partitions` element, create a new `Partition` element and change its name and included XML to match the application name and new partition XML created above. Within the `Schedules` element, add the new partition to the schedule. Within the `Payloads` element, add a new `PartitionPayload`. The `Base_Address` for the second partition should be 0x04C00000.
- Edit `image/build.mk` and add the new subdirectory name to `APP_DIRS` on line 71. Also add the partition XML file to the `INCLUDED_XML_FILES` on line 55. This will add the new application to the image.
- Update `image/image_startup` to load and start the partition. Make a copy of line 6 and update the name of the partition. Make a copy of line 9 and change the first argument to 2.

Appendix B.2 Minimal AFDX Project

The Minimal AFDX Project, referred to as minafdx from this point forward, implements a basic 'Hello World' type project for demonstrating AFDX traffic within AECS. The minafdx project executes on a single SBC and generates a single AFDX message at a rate of 0.5 Hz. The minafdx project requires a single SBC and can be run independent of the full AECS system. The AFDX packet can be received via a PC with a standard Ethernet port and the Wireshark utility.

The minafdx project is stored within the AECS subversion repository. Within the repository, it is located at:

```
/trunk/projects/minafdx
```

This directory is considered the 'root' of the minafdx project. It can be checked out on the DCM using the Uniform Resource Locator (URL):

```
https://localhost/svn/AECS/trunk/projects/minafdx
```

A copy of the minafdx project resides on the DCM in the directory:

```
E:\projects\minafdx
```

A copy of the subversion repository can also be found on the AECS Release DVD and from the AECS Collaborative Website. The minafdx project can be retrieved from both of these locations as well.

The root of the minafdx project directory tree contains the following files and subdirectories:

- `Makefile`: The top-level Makefile for the minafdx project.

- `afdx`: Contains `port_to_vls.csv` which maps AFDX VLs to VxWorks 653 pseudo-ports.
- `app`: The application source code resides within this directory.
- `build.mk`: Defines the build for the minafdx project.
- `mos`: Subdirectory containing the module OS.
- `pos`: .Subdirectory containing the partition OS.
- `sbc.mk`: Contains the SBC information for installing the image.

To build the minafdx project from the command shell, start an AECS VxWorks 653 Command Shell as discussed in the *AECS User's Guide*. Note that this shell is slightly different than a standard VxWorks 653 Command Shell as the WIND_PATH environment variable must point to the AECS VxWorks kernel directory.

From the command shell, change to the minafdx project directory.

```
E:
cd \projects\minafdx
```

Once in the proper directory, execute the make command:

```
make
```

The build subsystem determines which components are out of date and builds them as needed. The build will stop if any errors are detected. Once the system is built, it must be installed to the targets.

Prior to installing the minafdx project, the SBC IP address and flash method must be set within `sbc.mk`. Edit `sbc.mk` and uncomment the SBC_FLASH_METHOD and SBC_IP_ADDR lines for the SBC to be used. If the IP address is not listed (i.e. it is not one of the delivered IP addresses), simply add a line with the desired IP address.

Once SBC_FLASH_METHOD and SBC_IP_ADDR are set correctly, install the minafdx project from the command shell (assumes the current directory is set as noted above) with the commands:

```
make flash
make upload
```

Restart the SBC to start the project. Connect the first port on the AFDX card of the SBC to an Ethernet card on a standard PC which has the Wireshark utility installed (see the *AECS User's Guide* for detailed information on Wireshark). Monitor traffic on the interface where the SBC is connected and AFDX traffic should be visible.

The following list provides some brief tips on modifying the minafdx project (also see the tips on modifying the minfcc project as many of those tips are applicable as well).

- To change the message contents, edit the hex value assigned to `msg.pattern` on line 54 of `app/app.c`.

- To add an additional field to the message, add a field (or fields) to the structure on line 50 of `app/app.c` and set the values as desired. To change the value of a field each iteration, add code within the while() loop.
- To change the message rate, edit the `PERIOD` constant in `usrAppInit()` in `app/app.c`. The `Schedule` element within `image/fccImage-module.xml` also has elements which affect the rate.

Adding an additional message requires several steps:

- Copy line 3 within `afdx/port_to_vls.csv`. Change the portName and VLId to new values. This creates a mapping between the AFDX VL and a pseudo-port.
- Create a new `QueuingPort` element in `image/fccImage-pseudo.xml`. Copy lines 15 to 22 and change the `Name` attribute to match the new name from `afdx/port_to_vls.csv`.
- Create a new `QueuingPort` element in `app/app.xml`. Copy lines 56 to 62 and change the `Name` attribute. Name must be different from the name used in the pseudo partition.
- Create a new `Channel` element in `image/fccImage-module.xml`. Copy lines 75 to 78. Change the `Id` attribute and update the `PortNameRef` attributes to match the names used for the queuing ports above.
- Create the queuing port within the application. Copy lines 101-106 in `app/app.c` and change the name to match the name from within `app/app.xml`. Note that a new regional variable to store the ID is required.
- Add code to send the new message. Copy lines 66-71 in `app/app.c`.

Appendix B.3 Minimal TTE Project

The Minimal TTE Project, referred to as mintte from this point forward, implements a basic 'Hello World' type project for demonstrating AECS. The mintte project executes on FCC1 and FCC2, sending a TTE message between the SBCs. FCC1 sends the message. FCC2 receives the message. The mintte project requires two SBCs and two TTE switches for proper synchronization.

The mintte project is stored within the AECS subversion repository. Within the repository, it is located at:

```
/trunk/projects/mintte
```

This directory is considered the 'root' of the mintte project. It can be checked out on the DCM using the Uniform Resource Locator (URL):

```
https://localhost/svn/AECS/trunk/projects/mintte
```

A copy of the mintte project resides on the DCM in the directory:

```
E:\projects\mintte
```

A copy of the subversion repository can also be found on the AECS Release DVD and from the AECS Collaborative Website. The mintte project can be retrieved from both of these locations as well.

The root of the mintte project directory tree contains the following files and subdirectories:

- `Makefile`: The top-level Makefile for the mintte project.
- `build.mk`: Defines the build for the mintte project.
- `fcc1`: The application for FCC1.
- `fcc1.mk`: Contains the SBC information for installing the FCC1 image.
- `fcc2`: The application for FCC2.
- `fcc2.mk`: Contains the SBC information for installing the FCC2 image.
- `mos`: Subdirectory containing the module OS.
- `pos`: .Subdirectory containing the partition OS.
- `tte`: The TTE network description.

To build the mintte project from the command shell, start an AECS VxWorks 653 Command Shell as discussed in the *AECS User's Guide*. Note that this shell is slightly different than a standard VxWorks 653 Command Shell as the WIND_PATH environment variable must point to the AECS VxWorks kernel directory.

From the command shell, change to the mintte project directory.

```
E:
cd \projects\mintte
```

Once in the proper directory, execute the make command:

```
make
```

The build subsystem determines which components are out of date and builds them as needed. The build will stop if any errors are detected. Once the system is built, it must be installed to the targets.

By default, the mintte project installs to FCC1 and FCC2. This can be changed by changing the values of `SBC_FLASH_METHOD` and `SBC_IP_ADDR` in `fcc1.mk` and `fcc2.mk`.

Install the mintte project from the command shell (assumes the current directory is set as noted above) with the commands:

```
make flash
make upload
make tte/load
```

The last command above loads the configuration to the two TTE switches. Restart the SBCs to start the project. Open a console to both FCC1 and FCC2 to confirm that all is working correctly. It is also possible to monitor the TTE traffic using Wireshark and TTE Monitor. See the *AECS User's Guide* for additional information on monitoring TTE traffic.