

# Fault-Tolerance Verification of the Fluids and Combustion Facility of the International Space Station

Raquel Whittlesey-Harris and Mikhail Nesterenko  
Computer Science Department  
Kent State University  
Kent, OH 44242  
rwhittle@kent.edu, mikhail@cs.kent.edu

## Abstract

The Fluids and Combustion Facility (FCF) will be a permanent modular, multi-user facility used to accommodate microgravity science experiments in the U.S. Laboratory Module onboard the International Space Stations (ISS). The ability to withstand faults is vital for all ISS installations. Currently, the FCF safety specification requires a one-component fault-tolerance. In future versions, a more extensive fault-tolerance model may be required. In this paper, we describe the formal verification of fault-tolerance of the FCF Distributed State Model using SPIN. We program the FCF module state transitions in PROMELA (SPIN's internal language). We first verify the correctness of FCF without faults. We then simulate a single fault by moving one of the modules to an arbitrary state and verify correct recovery of the system. We extend our verification to the case of an extensive fault where the whole system has to recover from an arbitrary global state.

## 1 Introduction

**Automating Verification of Self-Stabilization.** One of the main difficulties in designing a self-stabling program is its lack of a limited set of initial states. The program is expected to start from an arbitrary state and eventually arrive at a legitimate state [1]. Traditionally, the correctness of a self-

stabilizing program is verified analytically. A classic approach is to find an invariant guaranteeing that a program starting from a state conforming to this invariant satisfies the specification. The correctness proof then proceeds by showing that regardless of the initial state the program eventually arrives at a state that satisfies the invariant.

However, in a practical distributed system the total number of states can be rather large. This makes analytical verification of stabilization a rather difficult task. Moreover, the presence of details and particulars of the system compound the problem: such details frequently result in special cases that have to be examined individually. Thus, the analytic proof of stabilization becomes tedious to construct and verify. As the size and complexity of such proof increases its validity becomes suspect.

In this paper we propose an alternative approach to verifying stabilization. We use model-checking techniques to automate the verification. We apply the techniques to the verification of fault tolerance of an experimental facility to be deployed at the International Space Station.

Model checkers are designed to find logical and functional design errors, such as deadlocks, livelocks, starvation, race conditions, priority problems, system bounds violations and specification incompleteness or redundancy. Because model checking is cost-effective and integrates well with conventional design methods, it is being adopted as a standard

procedure for the quality assurance of reactive systems [10].

**Addressing Safety of the Fluids and Combustion Facility.** We will be addressing the fault-tolerance of the Fluids and Combustion Facility (FCF), which is to become a permanent installation on board the International Space Station (ISS). The reliability of ISS facilities is critical. It is becoming even more important as budget demands increase while safety requirements become more central. Traditionally, space experiments have been conducted on the space shuttle. In this case, a failed experiment may be repeated during subsequent missions. However, ISS is a permanent facility where experiments operate continuously without down time. Therefore, opportunities for corrections are limited.

The adverse environment magnifies the reliability concerns. Non-space based experiments do not normally face the kind of environmental hardships to be endured by installations such as the FCF. The system must survive harsh acceleration forces including 3g peak during launch, 1.5g peak during re-entry, microgravity vibration and g-jitter (such as those caused by launch vibrations, motors, orbital maneuvers, astronaut activities, and experimental vibrations). There are a number of other factors that make the space environment particularly challenging. These may include: Galactic Cosmic Radiation, Trapped Belt Radiation, Solar Particle Events, Ionizing Radiation, the South Atlantic Anomaly, orbital debris, airborne contaminants, limited space craft volume, restricted physical access, satellite access, excess heat (both generated internally by experiments as well as charged particle) [6]. Protection of the space station environment presents another distinctive challenge. The ISS has strict requirements systems such as the FCF must adhere to in order to prevent contamination of the ISS environment. Moreover, protection of the crew is paramount: equipment failure should not

lead to the harm of the crew or the ISS. Much of the equipment to be used in space experiments is developed and tested on the ground. Hence, the reliability of such equipment in space is unproven.

Safety and reliability concerns are further increased by the limited access to the system. Crew time is limited in general. The FCF is currently expecting 1.5 hours per month of crew time. Maintaining a research installation in space, both the hardware and software components of it, is difficult. Replacement parts have to be transported to the ISS by spacecraft. Spacecraft have limited cargo space to accommodate such shipments. Once aboard the ISS, replacement parts have to be installed by crew members. The crew have limited experience and knowledge of the installation. Software upgrades may be uploaded via satellite. These kinds of uploads come with additional difficulties. Communications availability is currently at 30% coverage (the communication window is available less than 1/3 of the time.) In addition, operational challenges further complicate the scenario. The communication and accessibility limitations introduce a greater complexity to ascertain and repair the system when failures occur or upgrades are needed. Troubleshooting opportunities are limited. Ground operation teams are limited to the predefined telemetry downlinked during operation of the FCF. To upload new software care must be taken to insure installation of an error free replacement. The component responsible for managing the operation of the racks must be operational in order for this procedure to be implemented. This results in many opportunities for failure during implementation of this process. For example, a faulty kernel would not permit the component to boot or possibly open enough file descriptors to start up particular functionalities.

The current FCF system specification requires that the system must be able to handle a one-component failure [12].

*“The Flight Segment shall be designed to mitigate the consequences of Single Event Effects (SEE)”.*

Single Event Effects are Single Event Upsets, Single Event Latchups or Single Event Burnouts. Single Event Effects are radiation-induced errors in microelectronic circuits caused when charged particles (usually from the radiation belts or from cosmic rays) lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs. A Single Event Latchup is a condition that causes loss of device functionality due to a single-event induced current state. A Single Event Burnout is a condition that can cause device destruction due to a high current state in a power transistor. In addition to handling Single Event Upsets, the system is required to isolate failures to a particular component.

Greater robustness is expected to be achieved from the FCF in the future. It is also advantageous for the system to recover from more extensive failures (more than a one-component failure criteria). We anticipate that future specifications will require more extensive fault-tolerance.

**Our Approach.** We examine the behavior of the FCF as a collection of components. We assume that each component is capable of internally isolating the fault. External to the component, the failure manifests itself as a transition to an arbitrary state. We verify that this transition does not cause the system to violate the safety critical properties. We study more extensive faults.

We model the system using the SPIN [2], [3], [7], [8] tool. After modeling and verifying the operation of the system without faults, we systematically examine the operation of the FCF by simulating system faults using the SPIN simulator. We place the system in an arbitrary state and observe its behavior afterwards. The system is expected to arrive at a legitimate state.

We then formally verify the operation of FCF with a single component failure. We specify a number of safety critical predicates and verify that the system conforms to its specification. We model an extensive system failure by moving each component to an arbitrary state. We then verify the stabilization of the system to a state that conforms to the safety critical predicates.

**Related Literature.** The formal verification tool PRISM (Probabilistic Symbolic Model Checker) [5], [16] has been used to carry out probabilistic formal model checking of randomized distributed algorithms. Amongst the randomized distributed algorithms are self-stabilization algorithms. As a case study, PRISM was used to verify the self-stabilizing algorithm for token passing on a ring network.

PRISM differs from SPIN in that it can provide feedback on quantitative measures, such as reliability and performance, as well as the traditional functional correctness verification. In our experiment, only functional correctness was required. In addition, SPIN requires a deterministic, finite state model. PRISM on the other hand, does not require termination.

Another probabilistic modeling checking tool being utilized for distributed systems is the Approximate Probabilistic Model Checker (APMC) [17].

**Organization.** This paper is divided into the following sections. In section 1, we discuss the goal of our research. In section 2, we discuss the architecture of the Fluids and Combustion Facility. In section 3, we discuss the FCF model verified using the SPIN tool. In section 4, we discuss the experiments performed. We conclude this paper in section 5 with a discussion of the benefits of our experiments for the Fluids and Combustion Facility’s design team and future model verification plans.

## Terminology.

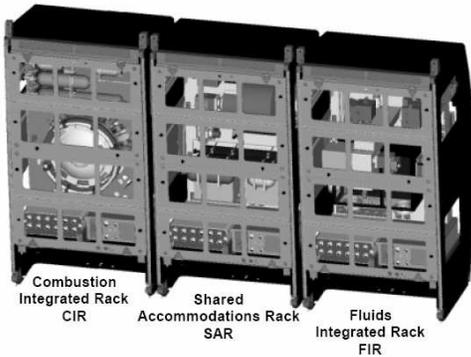


Figure 1. Fluids and Combustion Facility

Some of the terminology used in the paper is included below:

*Downlink* – data transmitted from the flight system to the ground system; *Flight Segment* - The FIR, CIR and SAR on the ISS; *Flight Segment Software* – The software component of the Flight Segment; *FSSS*- Flight Segment Support System - The GUI and Telescience support required to meet the objectives of the on-orbit mission; *Health and Status* - Data originating within the FCF Rack that is monitored by the Primary Processor to assure the safe and correct operation of the FCF and FCF Payloads, as well as assurance of ISS safety, as specified by safety guidelines; *Linear Temporal Logic Formulae* – technique for the specification of temporal rules; *Near Real Time* - The time the actual event occurs plus the time to process the data. Note, this time will vary with the situation to be performed. This time is usually in the order of seconds after the event occurred; *Uplink* – data transmitted from the ground system to the flight system.

## 2 Architecture

**Overview.** The FCF consists of the Combustion Integrated Rack (CIR) and the Fluids Integration Rack (FIR). See Figure 1. The Shared Accommodations Rack is currently not being developed and will not be further discussed in this paper. Due to

the unique operational environment, the FCF has been designed to be highly automated, modular, easily configured and maintained. In order to minimize cost, design complexity, and maintenance, it has been designed to maximize component reuse across the racks and the use of commercial off-the-shelf hardware and software. The CIR and FIR provide resources for Principal Investigators (PIs) to conduct scientific experiments in a microgravity environment.

**Common Components.** Common systems between the racks include: structural hardware, electrical (power control), environmental control (air and water thermal control, fire detection and suppression and gas interface), active rack isolation (payload isolation from mechanical disturbances onboard the ISS), device diagnostics (i.e., cameras, lenses, illumination, lasers), and command and data management (diagnostic control, image processing and communication hardware and software).

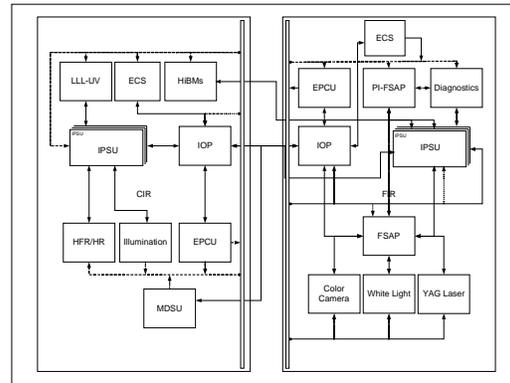


Figure 2. FCF Subsystems in a Potential Configuration

In our model we focus on the command and data management facilities of the FCF. We chose not to model communication with hardware, i.e., lasers. We assume that such communications is internal to the components.

The *Input/Output Processor* (IOP) is the component, which acts as the primary processor or rack/system controller. The IOP, amongst other things, is responsible for

processing and transmitting telemetry to and from ISS and monitoring as well as coordinating rack and inter-rack operations. Operations include health and status monitoring and time synchronization between components. The IOP consists of three separate processors, the IOP Main Processor (IOPMP), IOP Video Switch Processor (VSP) and the IOP High Rate Data Link (HRDL) Processor.

The *Input/Output Processor Main Processor* (IOPMP) provides all the data communication between the ISS and the FCF. This includes a MIL-STD-1553B bus interface to the ISS Low Rate Data Link (LRDL), IEEE 802.3 interface to the ISS Medium Rate Data Link (MRDL), and a fiber optic interface to the ISS High Rate Data Link (HRDL). IEEE 802.3 and a RS170 interfaces are also available to the ISS for a crew interface laptop. The IOPMP utilizes a CANBus interface between the IOP and components to monitor component health and status.

The *Input/Output Processor High Rate Data Link Processor* (IOPHP) is responsible for all HRDL communications between the ISS and the FCF. The link provides a 100-Mbps transmit interface.

The *Input/Output Processor Video Switch Processor* (IOPVSP) provides real-time switching of data and video from various sources in the FCF to various output devices/sources.

The *Image Processing and Storage Unit* (IPSU) is designed to accommodate image acquisition, processing and management typically required for fluids physics and combustion experiments. There are two types of IPSUs: one provides support for a wide range of digital cameras common to both the FIR and CIR. This IPSU stores video data in digital format and data acquired can be compressed if necessary to reduce memory and transfer bandwidth. Digital images can be processed to support closed loop control scenarios. The other

flavor of IPSU (IPSU-A) provides image acquisition from analog cameras. These images can be digitized and stored, processed and downlinked in much of the same manner as images directly from digital cameras. The CIR can accommodate up to six IPSUs while the FIR can accommodate up to two IPSUs. The FCF as a whole (FIR and CIR) has been designed to have the capability to utilize IPSUs located in the other rack (virtual IPSU) for extra processing power that may be required.

The *Diagnostic Control Module* (DCM) provides control, power, cooling and mechanical alignment interfaces between modules in a diagnostic package.

The *Image Acquisition Module* (IAM) is a device, which collects an image provided by a diagnostic package optical system and converts the image into a format that can be transferred via optical fiber to an IPSU. It integrates the camera with its power supply and serial data link in one unit and provides a four-flange mount to interface with other diagnostic modules.

The *Electrical Power Control Unit* (EPCU) performs power distribution, conversion, control, management and fault protection for the FCF racks. It is controlled by the IOP via a 1553 bus. On the front of the rack, there is a EPCU Shut-Off Switch Assembly (ESSA) to manually remove power from the rack.

The *Environmental Control System* (ECS) consists of four subsystems. These include the *Air Thermal Control Subsystem* (ATCS), the *Water Thermal Control Subsystem* (WTCS), the *Gas Interface Subsystem* (GIS) and the *Fire Detection and Suppression Subsystem* (FDSS).

The *Air Thermal Control Subsystem* (ATCS) provides an air-cooling system for the FCF avionics. It can remove up to 1650 W of facility generated waste thermal energy.

The *Air There Control Unit* (ATCU) uses a CANBus interface for communications with the IOP. It utilizes fans to draw warm air from the rear of the rack and send the air through a heat exchanger and filter returning cooled air into the system. Both the FIR and the CIR require a specific level of airflow for the system to safely operate. In the event that airflow inadvertently goes below this level, the system will be shut down.

The *Water Thermal Control Subsystem* (WTCS) provides cooling to the FCF equipment by removing FCF systems generated waste thermal energy and transferring it to the ISS *Internal Thermal Control System* (ITCS) *Moderate Temperature Loop* (MTL). The WTCS consists of a *Water Distribution Subsystem* to distribute water to primary and secondary loops; a Primary Loop Subsystem which provides cooling to non-science hardware; a Secondary Loop Subsystem which provides cooling to science hardware, a Control Subsystem located in the ATCU (EEU – ECS Electronic Unit) which provides electronic control between the IOP and WTCS hardware; and a Accumulator Subsystem which absorbs thermal control system pulsations due to temperature fluctuations during launch. If water flow is not maintained to achieve the required temperatures, the FCF system will be shut down.

The *Fire Detection and Suppression Subsystem* (FDSS) is responsible for detection and suppression of fire events. Each rack is independently monitored for smoke using an ISS smoke detector. Laser light attenuation and laser light scattering is used to detect smoke. Upon detection of a fire event, a red light emitting diode will provide a visual indication to the crew of the fire event location. Power to the rack will be removed by the IOP, which shuts down all airflow devices in the rack. Discharge and diffusion of CO<sub>2</sub> into the rack will depend on the charge pressure in the portable fire extinguisher.

The *Gas Interface Subsystem* (GIS) provides an interface for payloads to access ISS provided Gaseous Nitrogen (GN<sub>2</sub>), *Vacuum Exhaust System* (VES) and *Vacuum Resource System* (VRS) services. In the CIR, the *Fuel/Oxidizer Management Assembly* (FOMA) system provides the required controls for interfacing to the GIS. In the FIR, pressure regulation, flow control and exhaust gas processing functions are the responsibility of the Principal Investigator.

The *Active Rack Isolation System* (ARIS) isolates experiments from mechanical disturbances that can occur on the ISS. ARIS is essentially a shock absorber. ARIS has a sophisticated electronic sensing and control system that allows the racks to float within a 12.7 mm (0.5 inch) clearance in all directions thereby isolating payloads from motion disturbances. Attenuation of on-orbit low-frequency/large amplitude mechanical vibrations is achieved by utilization of accelerometers, actuators, a controller, drivers and sensors among other systems. Three tri-axial accelerometer packages measure vibration disturbances and transmit this data to a controller, which commands eight actuators to position the racks on three axes, six degrees of freedom to counteract the disturbances. ARIS has five system states, including, Idle – position and acceleration control is inactive however the controller reports health and status of the ARIS; Hold – the rack is centered and the position is held relative to the ISS without actively attenuating vibrations; Active – compensation for acceleration is achieved to maintain the microgravity environment; NOGO – essentially idle although entered on error; Power Off – emergency situations where ARIS power must be terminated immediately.

The *Space Acceleration Measurement System Triaxial Sensor Heads* (SAMS TSH) are used on both racks to expand the measurement capabilities.

The *Station Support Computer* (SSC) is a laptop computer, which is a shared ISS

resource. The primary purpose of the SSC is to provide crew support applications. Software necessary for command and control of the FCF using the SSC will be resident on the IOP. This software will be served up to the SSC for execution and will consist of applications and GUIs necessary to interact with the FCF.

**Combustion Integrated Rack.** The Combustion Integrated Rack (CIR), pictured in Figure 3, will provide sustained combustion physics research.

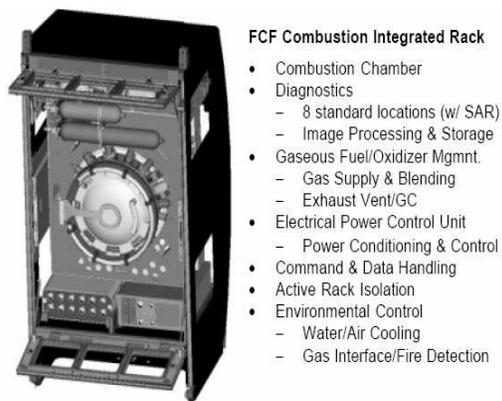


Figure 3. Combustion Integrated Rack

In addition to the common components discussed above, it includes, a Combustion Chamber, *Fuel Oxidizer and Management Assembly* (FOMA), additional diagnostic packages for imaging and other assemblies.

The *Fuel Oxidizer and Management Assembly* (FOMA) provides the capability to supply gaseous fuels, oxidizers and diluents into the combustion chamber. The FOMA also samples the environment of the combustion chamber via a *Gas Chromatograph* (GC) and controls the venting of chamber gases (to the ISS *Vacuum Exhaust System*) to acceptable concentration levels. Redundant valves are used to meet the safety requirements to protect bottled gas from being evacuated during venting. Pressure sensors and pressure switches ensure that an over

pressurization of the combustion chamber does not occur. Mass flow controllers are used to meter all gases. Pre-mixed gases may also be delivered to the combustion chamber via a nitrogen/high pressure manifold as an alternative. The *FOMA Control Unit* (FCU) is utilized as a software backup to these safety hazards. The software ensures that particular valve combinations are avoided to prevent such occurrences as over pressurization, contamination and improper venting to name a few. It performs command processing, control, data processing and health and status monitoring of the FOMA.

The two packages that make up the FOMA are the *Gas Delivery Package* (GDP) and the *Exhaust Vent Package* (EVP), which includes the *Gas Chromatograph* (GC). The *Gas Delivery Package* (GDP) consists of gas supply bottles and instrumentation to distribute and regulate gas delivery to the combustion chamber. Up to four consumable gases is permitted. The *Exhaust Vent Package* (EVP) is the interface between the combustion chamber and the ISS *Vacuum Exhaust System* (VES). It includes a PI-supplied adsorber cartridge and a recirculation loop. The adsorber cartridge is used to remove water, filter particulates, absorb trace amounts of unspent fuels, or chemically alter trace species (e.g., CO to CO<sub>2</sub>). The recirculation loop is used to convert post-combustion gases into acceptable species for venting and to improve the test gas environment for subsequent PI hardware tests.

The *Gas Chromatograph* (GC) includes a *Gas Chromatograph Instrumentation Package* (GCIP) and a *Gas Chromatograph Gas Supply Package* (GCGSP). It is used to sample gas from the combustion chamber and analyze exhaust vent gases for acceptability.

Diagnostic packages in the CIR include the *High Bit Depth/Multi-Spectral Package* (HiBMs), the *High Frame Rate/High Resolution Package* (HFR/HR), the *Color*

*Camera Package*, the *Low Light Level Ultraviolet Package (LLL-UV)*, the *Low Light Level Infrared Package (LLL-IR)* and the *Illumination Package*.

The *High Bit Depth/Multi-Spectral Package (HiBMs)* contains a spectral filter, a prism module, a telecentric optical system (provides pixel mapping through object space along parallel paths), a fixed mirror module, an optics housing module, a DCM, a high resolution 12-bit output digital camera in an IAM and a liquid crystal tunable filter.

The *High Frame Rate/High Resolution Package (HFR/HR)* consists of a telecentric (magnification does not change with focus position) optical system, a trombone prism assembly, a pointing mirror module, a filter compensator module, a high-resolution (1 mega-pixel) digital camera in the IAM, an optics housing module and a DCM.

The *Color Camera Package* contains two Objective Optics Modules, a Relay Optics Module, an IAM with a color camera, a Fixed Mirror Module, and Optics Housing Module and a DCM.

The *Low Light Level Packages (LLL)* produces images at a low radiance level. They each consist of a digital monochrome camera coupled with an IAM and fast numerical aperture optics with provision for spectral filtering of the transmitted illumination, a DCM, a Fixed Mirror Module and an Optics Housing Module.

The *Low Light Level Ultraviolet Package (LLL-UV)* package can be positioned on the optics bench to provide orthogonal views of an experiment. Combustion events can be recorded in matching or different spectral regions that are defined by investigator provided filters.

The *Illumination Package* contains a collimated optical system (Objective Optics Module and optics in the Illumination Source Module), a Fixed Mirror Module,

and an *Illumination Control Module (ICM)*. The illumination source is a laser diode array that can be used to provide monochromatic background illumination. This package can be used with the laser diode to provide a uniform illumination background for soot absorption measurements in soot volume fraction applications. The laser diode can be used as a non-coherent illumination source if operated below the lasing threshold. The laser diode can also be used as the background illumination source for shadowgraph measurements with the HiBMs Package or for droplet size measurement with the HFR/HR Package. Future growth considerations are feasible with the modular design of the system.

The Command and Data Management System (CDMS) for the CIR provides command, control, data acquisition, data processing, data management, health and status monitoring, interfaces and time synchronization between the IOP, FCU, IPSUs, DCMs, Diagnostic Packages and Science Payloads. The interfaces include the crew interface via the SSC and command, telemetry and video interfaces to the ISS Command and Data Handling System.

CDMS communications occur over Ethernet and CAN Bus. Health and Status is communicated via CAN Bus while interprocess communications occur over Ethernet.

**Fluids Integrated Rack.** The Fluids Integrated Rack (FIR), pictured in Figure 4, will provide sustained fluids physics research. The FIR provides common services (diagnostics) required by most fluid physics researchers to minimize the design and development for each experiment.

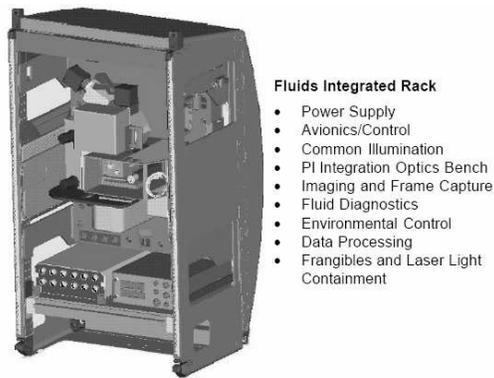


Figure 4. Fluids Integration Rack

In addition to the common subsystems discussed earlier, the FIR includes, a *Fluids Science Avionics Package* (FSAP), PI-FSAP, *Atmospheric Monitoring Assembly* (AMA), diagnostic packages and interfaces to accommodate PI provided diagnostics.

The *Fluids Science Avionics Package* (FSAP) is a multi-purpose data acquisition and control system that provides the capability to interact with a wide variety of fluids experiments. The FSAP provides a standard set of analog and digital Input/Output channels, motion controllers, analog video acquisition, data storage, and communication connectivity. An analog frame grabber is utilized to provide the capability to acquire images from an analog RS170A video source. The FSAP has the capability for two axes of motion control for stepper motors running in full, half, or microstep configurations, and two axes of motion control for servo motors and a resolution of 12 bits. Automated position and tracking is provided with the use of a CAN controller and DCM.

The *Principal Investigator Fluids Science and Avionics Package* (PI-FSAP) provides an enclosure with a microprocessor, communication interfaces, and card slots available for PI use. The PI has the ability to configure the PI-FSAP on the ground with science-specific circuit boards.

The *Atmospheric Monitoring Assembly* (AMA) provides temperature, pressure and relative humidity information for the rack volume.

Diagnostic packages in the FIR include the *Color Camera Assembly*, *White Light Assembly* and *Nd:YAG Laser Assembly*.

The *Color Camera Assembly* contains an analog 3-CCD (Charge-Coupled Device) color camera head and a *Color Camera Image Acquisition Module* (CCIAM). Real-time downlink of analog video can be provided to the ISS interface via the IOP. Health, status and control is provided through a CAN bus interface to the FSAP and CCIAM.

The *White Light Assembly* provides acquisition of color images while helping to prevent “ringing”. It contains two independently controllable lamp modules.

The *Nd:YAG Laser Assembly* converts 1064-nm output to 532 nm with the use of a non-linear, frequency-doubling crystal. The high quality beam is suitable for interferometry, velocimetry and monochromatic illumination of relatively large test cell areas.

The Command and Data Management System for the FIR is similar to the CIR. Time synchronization however, is between the IOP, FSAP, PI-FSAP, IPSUs, DCMs, Diagnostics and science payloads.

**The FCF Software System.** The FCF Flight Software System is a distributed real-time multitasking embedded system. Main components are running on the VxWorks [9] operating system. Communication between components is achieved through Ethernet, Fiber-Optic, CANBus, Analog, MIL-STD-1553 and Serial Data links.

All main component communication is done through the primary rack controller — the *IOP*. In addition, communication to the ISS is achieved through a Medium Rate Data

Link – an 802.3 interface running at 10 Mbps; a Low Rate Data Link – a MIL-STD-1553 interface running at 1 Mbps; and a High Rate Data Link – Fiber Optic Data Distributed Interface running at 100 Mbps. There is also analog video (RS170) and Ethernet (100BaseT) interfaces to the on board station computer for crew interface.

Communications between sub-components and main components may take place directly.

The IOP as the rack manager maintains the overall rack state and monitors component states as well as component health and status. The FCF defined rack states are depicted in Figure 5.

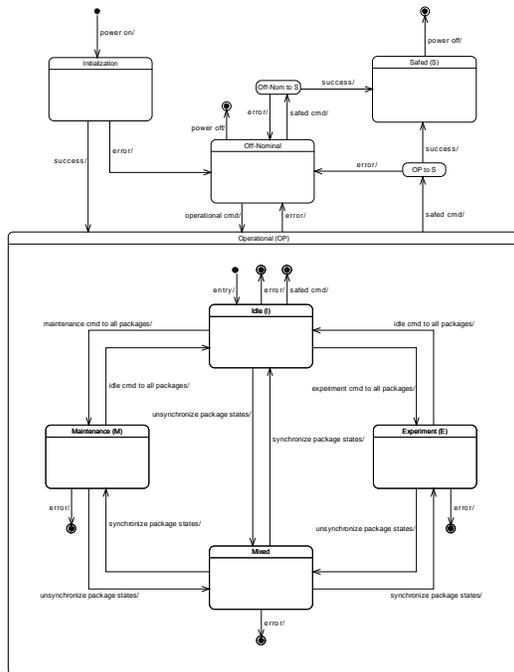


Figure 5. FCF Rack States

Component states (in general) include:

**good-off** — The component has either never been powered on or have been shut down due to a nominal circumstance, without any anomalies.

**bad-off** — The component is powered off due to an anomaly.

**initialization** — The component has been powered on and is performing system Power-On-Self-Test (POST) and initializing hardware and software components. The component is not yet ready to initiate communications with the Rack Manager and receive commands.

**operational-idle** — The component has completed initialization, is operating nominally and is ready to perform experiment operations. Generally only communications (telemetry and limited commanding) is available in **operational-idle**.

**operational-uplink/downlink** — The component is operating nominally and is ready to receive an uplink or transmit data to the ground (downlink).

**operational-maintenance** — The component has expanded commanding capability from **operational-idle**. May be used for troubleshooting or initiating non-sequenced events.

**operational-experiment** — The component is operating nominally and in a state to perform experiment operations. Commanding capability is expanded from **operational-idle**.

**off-nominal** — The component has encountered an anomaly that must be addressed before further operations may take place.

**safed** — The component is ready for power-down (all hardware and software components have been put in a state that will not damage the component or cause lost of data). Generally only the power down command is accepted in this state. Communications to the Rack Manager continues.

Rack states include:

**good-off** — All system components have either never been powered on or have been shut down due to a nominal circumstance, without any anomalies.

**bad-off** — All system components are powered off and at least one system component is in the powered off state due to an anomaly.

**initialization** — All powered system components are in their respective **initialization** states.

**operational-idle** — All powered system components are in their respective **operational-idle** states.

**operational-uplink/downlink** — All powered system components are in their respective **operational-uplink/downlink** states.

**operational-maintenance** — All powered system components are in their respective **operational-maintenance** states.

**operational-experiment** — All powered system components are in their respective **operational-experiment** states.

**operational-mixed** — Not all powered system components are in the same state.

**off-nominal** — At least one of the powered system components is in **off-nominal**.

**safed** — All powered system components are in their respective **safed** states.

The Rack Manager manages the rack state. The components communicate their current state to the Rack Manager with every telemetry packet sent. The Rack Manager in turn responds to changes in component states by possibly altering the rack state. The rack state is determined by the rules stated above. In addition, if a component

enters the **off-nominal** state, all powered on components will be commanded to their respective **operational-idle** states. The rack will remain in the **off-nominal** state until the **off-nominal** is first acknowledged from an operator and all **off-nominal** components are returned to an **operating** state or **safed/powered off**.

**Sets of States.** We clarify the states of the FCF into three sets of states. The operational states include all states where the FCF is capable of performing normal operations. The safe states include all states where the FCF is in off-nominal operations and is not violating the hazard specification. The unsafe states are states where the system goes during faults. The FCF handles faults by converging to a safe or operational state. A possible correct transition of the FCF Model between states is (see Figure 6),

1. The system initiates in a valid state,
2. The system moves to a fault span state,
3. The system converges to a safe state within a finite number of steps.

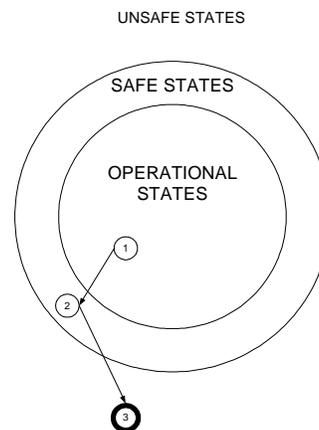


Figure 6. State Space

These three encompassing states represent the model self-stabilizing capability. With

an arbitrary initial system state, the system should always converge to the safe state.

In addition to the predefined states described above, the rack manager also monitors the health and status of components in the rack(s). The rack manager will take actions to ensure the safety of the system, the ISS, and the crew. This may include powering down a package, turning off hazardous items, even powering down the entire rack. There are seven predefined actions the IOP will take in response to a state of the system that is considered “Unsafe”. See Section 3 for a list of the seven actions. After executing one of these actions, the system will converge to a legitimate state. Legitimate states are considered “Safe” states and “Operational” states.

### 3 The FCF Model

**Overview.** The FCF Meta Model is written in the SPIN modeling language PROMELA. PROMELA, a non-deterministic, guarded command language. It enables the dynamic creation of concurrent processes and communication between processes via message channels.

There are limitations to modeling a distributed real-time system with SPIN. First, there is no concept of time in SPIN. Second, the distributed multi-tasking environment is difficult to simulate. Furthermore, the system modeled must be a finite-state model to guarantee decidability. Several of these limitations have been addressed in extensions to SPIN [14].

A model is an abstraction of the system. The FCF Model is an abstraction of the FCF state model and a subset of the safety specification. The FCF Meta Model includes several processes running on a single processor. The FCF Model includes a simplified communication protocol to simulate the interaction between components and processes. This includes, events, commands and state information.

### Component Model.

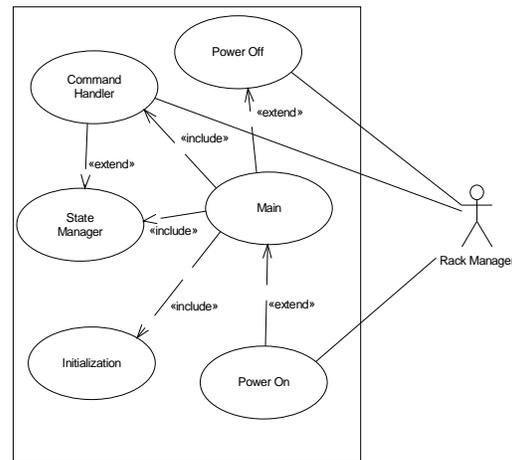


Figure 7. FCF Meta Model Processes

In the FCF Meta Model, each component consists of several processes. See Figure 7 (shown in UML notation [13]). Each process runs in parallel and implements the main functionality of a component. The main process handles initialization, communication direction, health and status checks, and nominal shutdown. The component command handler validates commands and initiates processing of a valid command. The component state manager manages the components state transitions. This includes the states described in Section 2 Architecture

**Component Power Up Example.** As an example, we describe the actions of the IPSU and Rack Manager by process while performing a power on, command processing and then power down sequence.

1. The Rack Manager initiates a power on of an IPSU. The *IOP* reads the configuration information for the component it wants to power on. The Power On task of the Rack Manager is executed to implement the power on of the component.

2. The Component Initialization. The component determines its own function. It determines if it is an IPSU (either 1 – 6). It initializes the appropriate state variables. The Power On task of the component is executed to power on any subcomponents. A Power-On Self Test (POST) is executed. This test simulates the health check of internal systems upon component power up. The component randomly selects whether it powers on and passes POST or fails. If the POST is successful, the component enters **operational-idle**. In this case, commanding and telemetry handlers are initiated.
3. Component Health Monitoring. The component now begins to monitor its own health and status, process commands, send regular communications to the *IOP*, and monitor the *IOP* status.
4. Command Processing. During operation, a command is sent from the *IOP* to the IPSU to transition to the IPSU to **operational-idle**. The *IOP* has determined a system component to be in **off-nominal**. All powered on components are therefore sent to **operational-idle**. A packet is transmitted from the *IOP* to the IPSU. The IPSU determines the packet to be a command and invokes the Command Handler to decipher it. The Command Handler determines the command is for the IPSU that has received it (IPSU 1-6). The Command Handler then determines the command to be a valid command for the IPSU. The command handler now determines the command is a state change request. In this case, the Command Handler forwards the command to the component's State Manager for processing.
5. State Request Processing. The component's State Manager receives a request to change the current state to **operational-idle**. The State Manager

determines which component the request is from and verifies it is a valid requestor; the rack manager. The State Manager determines if the transition is legal. If the transition is legal, the State Manager sets the component state to **operational-idle**.

6. Power Down Component. The Rack Manager determines that it needs to power down the IPSU. A command is sent to the IPSU to transition it to **safed**. The Rack Manager looks up the configuration information for the component and sends commands to the EPCU to power down any sub-components that are powered on. Following the power down of sub-components, the component is powered down via the EPCU.

### Rack Manager Model.

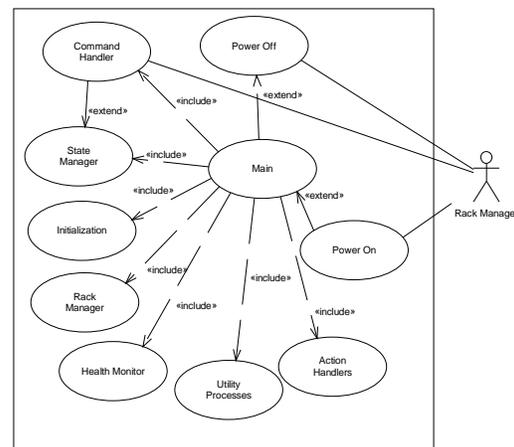


Figure 8. Rack Manager Processes

The *IOP*, being the racks and facility manager has similar processes to the component processes with additional functionality. See Figure 8 (shown in UML notation [13]). The *IOP* also has a rack manager, an action handler, health monitor,

processes for each action and several utility processes for jobs such as turning off and on components and determining what hazardous items are operating. In this model, there is one *IOP* managing all of the components in both racks.

Within the *IOP* Action Handler, the seven actions the *IOP* initiates in response to hundreds of safety events is implemented. These actions are described below:

Action 1 — Power off a component.

Action 2 — Power off a subcomponent.

Action 3 — Orderly power down of all components (including their subcomponents) except for the *IOP* and *ECS*.

Action 4 — Orderly power down of all components (including their subcomponents) along with the *IOP* and *ECS*.

Action 5 — Command the *FCU* to Off Nominal. If it fails to respond, power the *FCU* down.

Action 6 — Command the *IOP* to Off-Nominal.

Action 7 — Power off all hazardous items.

**Fault Model.** In every component, there is a fault simulation process that introduces faults into the system. This process randomly introduces state and safety faults.

Eight possible fault scenarios are introduced into the system by the fault simulator task. These faults are discussed below.

1. Set the state of solenoid valves SV9 and SV14 to open. — To prevent the inadvertent venting of out of limit chamber premixed gasses into ISS vent system (causes fire in vent system) the

*FCU* should never vent from valve nine and valve fourteen at the same time. The *IOP* acts as a watchdog. The health monitor shall monitor the status of solenoid valves nine and fourteen and command the *FCU* to **off-nominal** if such a state arises. The *FCU* automatically closes all energized solenoid valves when entering the off-nominal state. After commanding the *FCU* to **off-nominal**, if the solenoid valves remain energized, the *IOP* will power down the *FCU*. This in turn will deenergize the valves. — Predicate 7.

2. Set the state of the rack door to open. — To prevent astronaut exposure to hazardous items, all hazardous items (e.g., lasers) are powered off upon detection of an open rack door. — Predicate 1.
3. Set the state of the *IOP* communication received flag from a randomly selected package to false. This indicates that the *IOP* has not received communications from the package for a particular time frame. — Predicate 3.
4. Set the state of the *IOP* communication to a randomly selected package to false. This indicates that the package has not received communications from the *IOP* for a particular time frame. — The *IOP* as the rack manager monitors the health and status of all components powered on. If the *IOP* does not receive communications from a component for a specified period of time, the component must be powered down as to not cause a potential hazardous situation. — Predicate 5.
5. Set the state of the *ECS* CANBus communication flag to false. This indicates that the *IOP* has not received communication from the *ECS* over the *ECS* CANBus link for a particular time frame. The *ECS* is an essential component. The system cannot

safely operate without thermal (air and water) control. If the ECS component is not in communications with the Rack Manager, the system is powered down.— Predicate 2.

6. Set the state of the ECS component state flag to powered down. Again, the ECS must be functioning for the rack to be safely operated. — Predicate 6.
7. Set the state of solenoid valves SV14 and SV4 to open. — To prevent the inadvertent venting of out of limit chamber oxygen into ISS vent system (causes fire in vent system) the FCU should never vent from valve four and valve fourteen at the same time. The IOP acts as a watchdog. The health monitor shall monitor the status of solenoid valves nine and fourteen and command the FCU to **off-nominal** if such a state arises. The FCU automatically closes all energized solenoid valves when entering the off-nominal state. After commanding the FCU to **off-nominal**, if the solenoid valves remain energized, the IOP will power down the FCU. This in turn will deenergize the valves. — Predicate 8.
8. Set the state of solenoid valves SV12 and SV14 to open. — To prevent the over pressurization of the chamber, the FCU should never energize valves twelve and fourteen at the same time. The IOP acts as a watchdog. The health monitor shall monitor the status of solenoid valves twelve and fourteen and command the FCU to **off-nominal** if such a state arises. The FCU automatically closes all energized solenoid valves when entering the off-nominal state. After commanding the FCU to **off-nominal**, if the solenoid valves remain energized, the IOP will power down the FCU. This in turn will deenergize the valves. — Predicate 9.

**The Specification.** The FCF has hundreds of safety monitoring responsibilities, many of which have similar responses. We have therefore modeled a representation of the critical system responses. We express the safety properties in the form of predicates. We formally verify the system conforms to these predicates.

The FCF Model was formally verified against nine safety predicates. These predicates are described below.

When the rack door is open, the system will power off any hazardous items:

$$l \Rightarrow \Diamond m \quad (1)$$

where  $l$  represents the rack door is open and  $m$  represents all hazardous items are powered off.

When there is a loss of communication on the ECS CANBus, all packages must be shut down (are either in **good\_off** or **bad\_off**):

$$\begin{aligned}
 n \Rightarrow \Diamond( & \\
 (s\_IPSU1 \vee t\_IPSU1) \wedge & \\
 (s\_IPSU2 \vee t\_IPSU2) \wedge & \\
 (s\_IPSU3 \vee t\_IPSU3) \wedge & \\
 (s\_IPSU4 \vee t\_IPSU4) \wedge & \\
 (s\_IPSU5 \vee t\_IPSU5) \wedge & \\
 (s\_IPSU6 \vee t\_IPSU6) \wedge & \\
 (s\_FCU \vee t\_FCU) \wedge & \\
 (s\_FSAP \vee t\_FSAP) \wedge & \\
 (s\_PIP \vee t\_PIP) &
 \end{aligned} \quad (2)$$

where  $n$  represents IOP loss of communication on the ECS CANBus,  $s$  represents a package in **good\_off** and  $t$  represents a package in **bad\_off**.

When the *IOP* loses communication with a package, the package is sent to **safed**:

$$\begin{aligned}
&(o\_IPSU1 \Rightarrow \diamond r\_IPSU1) \wedge \\
&(o\_IPSU2 \Rightarrow \diamond r\_IPSU2) \wedge \\
&(o\_IPSU3 \Rightarrow \diamond r\_IPSU3) \wedge \\
&(o\_IPSU4 \Rightarrow \diamond r\_IPSU4) \wedge \\
&(o\_IPSU5 \Rightarrow \diamond r\_IPSU5) \wedge \quad (3) \\
&(o\_IPSU6 \Rightarrow \diamond r\_IPSU6) \wedge \\
&(o\_FCU \Rightarrow \diamond r\_FCU) \wedge \\
&(o\_FSAP \Rightarrow \diamond r\_FSAP) \wedge \\
&(o\_PIP \Rightarrow \diamond r\_PIP)
\end{aligned}$$

where *o* represents the *IOP* loses communication with the package and *r* represents the package is in **safed**.

When the *IOP* goes into **off-nominal**, all packages are sent to a state where the equipment and data is safe (**idle**, **safed**, **good\_off** or **bad\_off**):

$$\begin{aligned}
&p \Rightarrow \diamond( \\
&(q\_IPSU1 \vee r\_IPSU1 \vee \\
&s\_IPSU1 \vee t\_IPSU1 \vee \\
&p\_IPSU1) \wedge \\
&(q\_IPSU2 \vee r\_IPSU2 \vee \\
&s\_IPSU2 \vee t\_IPSU2 \vee \\
&p\_IPSU2) \wedge \\
&(q\_IPSU3 \vee r\_IPSU3 \vee \\
&s\_IPSU3 \vee t\_IPSU3 \vee \\
&p\_IPSU3) \wedge \\
&(q\_IPSU4 \vee r\_IPSU4 \vee \\
&s\_IPSU4 \vee t\_IPSU4 \vee \\
&p\_IPSU4) \wedge \\
&(q\_IPSU5 \vee r\_IPSU5 \vee \\
&s\_IPSU5 \vee t\_IPSU5 \vee \\
&p\_IPSU5) \wedge \\
&(q\_IPSU6 \vee r\_IPSU6 \vee \\
&s\_IPSU6 \vee t\_IPSU6 \vee \\
&p\_IPSU6) \wedge \\
&(q\_FCU \vee r\_FCU \vee \\
&s\_FCU \vee t\_FCU \vee \\
&p\_FCU) \wedge \quad (4) \\
&(q\_FSAP \vee r\_FSAP \vee \\
&s\_FSAP \vee t\_FSAP \vee \\
&p\_FSAP) \wedge \\
&(q\_PIP \vee r\_PIP \vee \\
&s\_PIP \vee t\_PIP \vee \\
&p\_PIP))
\end{aligned}$$

where *p* represents the package in **off-nominal**, *q* represents the package is in **idle**, *r* represents the package is in **safed**, *s* represents the package is in **good\_off**, *t* represents the package is in **bad\_off**.

When a package loses communication with the *IOP*, the package will go into **safed**:

$$\begin{aligned}
&(z\_IPSU1 \Rightarrow \diamond t\_IPSU1) \wedge \\
&]z\_IPSU2 \Rightarrow \diamond t\_IPSU2) \wedge \\
&(z\_IPSU3 \Rightarrow \diamond t\_IPSU3) \wedge \\
&(z\_IPSU4 \Rightarrow \diamond t\_IPSU4) \wedge \\
&(z\_IPSU5 \Rightarrow \diamond t\_IPSU5) \wedge \quad (5) \\
&(z\_IPSU6 \Rightarrow \diamond t\_IPSU6) \wedge \\
&(z\_FCU \Rightarrow \diamond t\_FCU) \wedge \\
&(z\_FSAP \Rightarrow \diamond t\_FSAP) \wedge \\
&(z\_PIP \Rightarrow \diamond t\_PIP)
\end{aligned}$$

where  $z$  represents the package has lost communication with the *IOP* and  $t$  represents the package is in **safed**. Note, this is an added capability added to the model and a suggestion for future FCF development.

When the *ECS* is powered off, the *IOP* will go into **bad\_off**:

$$\neg u \Rightarrow \diamond y \quad (6)$$

where  $u$  represents the *ECS* is powered on and  $y$  represents the *IOP* is in **bad\_off**.

When the FCU solenoid valve *S14* is open and FCU solenoid valve, *S9*, is open, the FCU will go into **off-nominal**:

$$(x\_S14 \wedge x\_S9) \Rightarrow \diamond p\_FCU \quad (7)$$

where  $x$  represents the solenoid valve is open and  $p$  represents the FCU is in **off-nominal**.

When the FCU solenoid valve, *S14*, is open and FCU solenoid valve, *S4*, is open, the FCU will go into the **off-nominal** state:

$$(x\_S14 \wedge x\_S4) \Rightarrow \diamond p\_FCU \quad (8)$$

where  $x$  represents the solenoid valve is open and  $p$  represents the FCU is in **off-nominal**.

When the FCU solenoid valve, *S14*, is open and FCU solenoid valve, *S12*, is open, the FCU will go into the **off-nominal**:

$$(x\_S14 \wedge x\_S12) \Rightarrow \diamond p\_FCU \quad (9)$$

where  $x$  represents the solenoid valve is open and  $p$  represents the FCU is in **off-nominal**.

## 4 Experiments

**Preamble.** Our experiments were carried out in two phases: a simulation phase and a verification phase. During the simulation phase, we ascertained the model was correctly implemented (exhibited the anticipated behavior of the FCF System). In the verification phase, we established the model was correct given the nine predicates as constraints. The simulator is a random, guided and interactive execution of the model. When initiating execution, the given seed provides determinism. This enables the user to execute the exact sequence precisely repeatedly. This is particularly useful for debugging. The simulator can repeatedly execute sequences with faults, such as transients, that are difficult to reproduce [4]. It does have the drawback of potentially missing an execution path and therefore possibly leaving behind implementation errors. The verifier differs from the simulator in that each execution of the model exposes every possible execution path. This is the only way to guarantee correctness of a model.

**Simulation.** We executed two simulation models to provide feedback on correct implementation. Initially, we created a FCF simulation model in order to exercise legitimate system states. We simulated in this matter to assist in debugging the implementation of the FCF in PROMELA.

During this phase we used the windows version of the SPIN simulator to take advantage of the graphical displays that could be monitored in real-time.

Three displays were invaluable in the debugging phase of the simulation. These include, the Simulation Output Display, The Message Passing Display and the Data Values Display. We show sample screen shots of the Simulation Output Display and the Message passing display in Figure 9 and Figure 10 below. The Data Values Display as well as other SPIN displays we utilized is included in Appendix A.

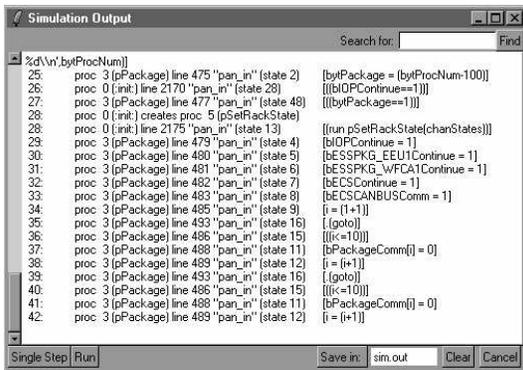


Figure 9. Simulation Output View

Depicts each step executed in order along with the text of the line executed and the value of data set or passed in the call. This display can be paused and/or resumed to assist in monitoring execution.

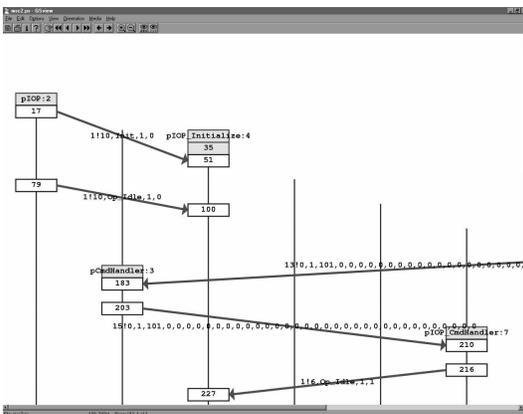


Figure 10. Message Passing View

Useful to monitor during run-time, but we found to be more useful for post-analysis. This provided a visual and expeditious way to determine message-passing sequences. In this display, each process that sends messages to another process is represented with a vertical line. The name of the process is highlighted in yellow. Red lines signifying messages are displayed going from the sending process to the receiving process. These messages are represented with a number, which can be related to a execution step. It is also possible to have the message text displayed.

After debugging and accepting the implementation of the FCF Model, we added the random fault generator process to the model. The random fault generator produces randomly introduces faults one after the other. This model was executed over 100 simulation runs. For each run we provided a different seed to provide a new random result. The random fault generator process is the process used in the final model to introduce faults into the system. However, during the simulation, the fault generator is cycled 50 times during each run. Since the simulator runs an execution of the system in a deterministic fashion based on the seed, this is the mechanism that permits the sequence of faults to be introduced one after the other in each execution.

This fault generator model was simulated in both the Windows environment as well as the Linux command line environment. The model was simulated in the Windows environment on a Pentium 4 with 256 MB RAM, 4GB of Virtual Memory and operating on Windows NT 4.0. The average run-time of the model in this environment with several other tasks running was 1 hour and 45 minutes. The model was simulated in the Linux environment on a machine with 4 Intel Zeon, 2.8 Gigahertz processors, 4 Gigabytes of RAM, 8 Gigabytes of swap space and operating on RedHat Linux Enterprise 3. The average run time of the model in this environment is negligible.

**Verification.** Initially, we ran the FCF Model in the verifier to verify there were no acceptance cycles or invalid end states. Following this, we ran nine verification models in the SPIN verifier against the predicates given above. Using SPIN, we compiled the FCF model with each predicate, separately. This process created nine verification models. We compiled each FCF verification model on an Intel Pentium P4 2.8 Gigahertz processor machine with 1 Gigabyte of RAM and running Linux 9.0. To accommodate compilation swap had to be extended to 4GB. The compile time for non-compound LTL-formulae (predicates 1 and 6 – 9) was negligible. Due to the time requirement to compile and run the complex LTL-formulae [3], these predicates were compiled in a simplified specification to provide correctness feedback in a more timely fashion. At the time of this publication, the complex LTL-formulae (predicates 2-5) is still in the process of completing compilation and execution.

The verification models generated are in the C programming language. We compiled these verification models using the GNU C compiler. We selected compiler options to reduce the complexity and memory requirements of the verification. This assisted in decreasing run-time and memory usage. We compiled the verification models on the machine used to compile the FCF models. The compile time for all verification models was negligible.

After compiling each verification model, we ran the models on machines with 4 Intel Zeon, 2.8 Gigahertz processors, 4 Gigabytes of RAM and operating on RedHat Linux Enterprise 3. To accommodate verification, the swap space had to be extended to 8 Gigabytes.

The following metrics is generated and supplied below for each verification run. The number of errors found in the model during verification is reported as *Errors*. The number of states revisited during verification is reported as *States Matched*.

The number of states stored during verification is reported as *States Stored*. The longest depth of the DFS State tree is reported as *Depth*. The number of state transitions during verification is reported as *Transitions*. The amount of memory used to store the states, in MB, is reported as *Memory*. A summary of the statistics for each execution is provided in Figure 11 below. The complete reported statistics is provided in Appendix B.

Verification model one executed in 6 hours without any errors reported. This model shows that the FCF system model will converge to the state of operating without power to any hazardous items in response to being thrown into a state with the rack door ajar.

Verification model two was first run with simplified LTL formulae in order to reduce run-time. The model executed in 18 hours without any errors reported. This model shows that the FCF system model will converge to the state of powered-off if communications is lost with the ECS.

Verification model three was first run with simplified LTL formulae in order to reduce run-time. The model executed in 14.5 hours without any errors reported. This model shows that the FCF system model will converge to the state of all packages are in **safed** state for which the rack manager has lost communication with.

Verification model four was first run with simplified LTL formulae in order to reduce run-time. The model executed in 18 hours without any errors reported. This model shows that the FCF system model will converge to the state of “safe off-nominal” when any package is determined to be in an **off-nominal** state.

Verification model five was first run with simplified LTL formulae in order to reduce time. The model executed in 18.5 hours without any errors reported. This model shows that the FCF system model will

converge to the state of all packages that have lost communications with the rack manager are in **safed** state. This verification shows packages other than the rack manager can stabilize the system.

Verification model six executed in 2.5 hours without any errors reported. This model shows that the FCF system model will converge to a state where the IOP is in **bad-off** in response to being put in a state where there is no communications with the ECS.

Verification model seven executed in 2.5 hours without any errors reported. This model shows that the FCF system model will converge to a state where the FCU is **off-nominal** in response to a state where solenoid valves *S14* and *S9* are found to be open

Verification model eight executed in 15.5 hours without any errors reported. This model shows that the FCF system model will converge to a state where the FCU is **off-nominal** in response to a state where solenoid valves *S14* and *S4* are found to be open.

Verification model nine executed in 3 hours without any errors reported. This model shows that the FCF system model will converge to a state where the FCU is **off-nominal** in response to a state where solenoid valves *S14* and *S12* are found to be open.

<b>Model</b>	<b>Run Time</b>	<b>States Stored</b>	<b>Sates Matched</b>	<b>Depth</b>	<b>Transitions</b>	<b>Memory</b>
Predicate 1	6 hours	2.86e+06	1.18e+07	7319	1.469e+08	393762.878
Predicate 2	18 hours	2.87e+06	1.19e+07	6305	1.472e+07	395904.356
Predicate 3	14.5 hours	2.55e+06	3.53e+06	6327	6.078e+06	352055.424
Predicate 4	18 hours	2.88e+06	1.19e+07	6337	1.477e+07	397429.996
Predicate 5	18.5 hours	2.93e+06	1.24e+07	6305	1.531e+07	403771.225
Predicate 6	2.5 hours	2.62e+06	3.68e+06	8055	6.301e+06	361863.855
Predicate 7	2.5 hours	2.53e+06	3.42e+06	8055	5.951e+06	34.9666.837
Predicate 8	15.5 hours	2.50e+06	5.32e+06	6311	7.818e+06	34411.991
Predicate 9	3 hours	2.50e+06	5.32e+06	8265	7.818e+06	34411.991

Figure 11. Model Statistics (Simplified LTL-formulae)

*States stored indicates the total states in the system. States Matched refers to the total states revisited during the verification. Depth refers to the longest path in the DFS tree. Transitions indicate the number of transitions from state to state to complete the verification. Memory refers to the memory used to store the states in MB.*

## 5 Conclusion

**Benefits.** We completed the implementation of the FCF System Model in PROMELA prior to the finalizing of the FCF system state design and verification. Therefore, the formal verification of the FCF System Model was particularly useful to the FCF design team at NASA Glenn Research Center for the purpose of verifying and debugging system state implementation. In addition to finding a few implementation errors, the simulation was executed over one hundred times. The FCF did not have the luxury of executing verifications to this degree. This provides added assurance of the soundness of the system state design.

The FCF design team was able to take advantage of the implementation of the FCF System Model. It provided the opportunity to clearly think through implementation details as the system was modeled, simulated, debugged and verified. A couple of implementation errors were found and corrected before formal verifications of the FCF were completed. An implementation error was found in the routine that handles sending the rack to the **off-nominal** state when the IOP is found to be off-nominal. This routine, `Action_6`, attempted to command packages in the **safed** state to **op-idle** state. Only packages in an operational state (**op-maintenance**, **op-experiment**, **op-uplink-downlink**) are to be commanded to **op-idle** state. The only transition allowed from **safed** state is to powered-off (**good-off** or **bad-off**). The correction was made to the FCF system implementation. An undesirable design feature was found in the routine that manages rack state changes. The rack state was being calculated for each package state calculated. It appears as if the rack state was in constant fluctuation. It is better if the rack state is set after a local copy of the rack state is calculated in entirety.

**Future Work.** There are opportunities for increased fault-tolerance of the FCF System. We propose incorporating crash failure tolerance. In the event the IOP “crashes”, the FCF is not functional. IOP fail-over capability between racks would improve system capabilities greatly and allow for better use of rack-to-rack capabilities.

We also propose to expand the state space of the model to introduce a more detailed modeling of the system components. There is a trade off that will have to be further investigated. The compile and run-time of a larger model is sure to become a more looming problem. We thus propose considering further optimization of code to enable efficient compilation and execution of the model.

One additional proposition is the addition of real-time constraints. Due to the fact that SPIN does not have real-time extensions, this would require the system be re-implemented in a tool such as RT-SPIN [14], UPPAAL [11], or a tool with discrete time extension to SPIN [15].

In the design of the FCF System Model, a few suggestions were made which may be incorporated into the design of the FCF in the future. These suggestions were implemented in the FCF System Model. First, the robustness of the FCF System would be increased if the capability for either IOP to control the power to all packages (i.e., an IOP in the CIR rack can control power to packages in the FIR rack and vice versa). Secondly, the packages should be “IOP aware”. Meaning, the a package can determine if the IOP has not communicated with it and enter it’s **safe** state upon detecting that communication has not taken place for a specified period of time. This will provide added system safety and self-stabilization.

## References

- [1] Arora, Anish, *Stabilization*. Department of Computer Sciences, Ohio State University, 2000.
- [2] Holzmann, J. Gerald and Peled Doron, *An Improvement in Formal Verification*. AT&T Bell Laboratories, 1994.
- [3] Holzmann, J. Gerald, The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23( 5):279-295, May 1997.
- [4] Chandra, Subhachandra, *An Evaluation of the Recovery-Related Properties of Software Faults*. Department of Computer Science and Engineering, University of Michigan, 2000.
- [5] Probabilistic Symbolic Model Checker: [www.cs.bham.ac.uk/~dxp/prism/index.html](http://www.cs.bham.ac.uk/~dxp/prism/index.html)
- [6] NASA, *Strategic Program Plan For Space Radiation Health Research*. National Aeronautics Space Administration, 1998.
- [7] Simple PROMELA Interpreter: [www.spinroot.com](http://www.spinroot.com)
- [8] Holzmann, J. Gerard, *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.
- [9] VxWorks: [www.wrs.com/products/html/vxworks.html](http://www.wrs.com/products/html/vxworks.html)
- [10] Merz, Stephan, *Model Checking: A Tutorial Overview*. Institut fur Informatik, Universitat Munchen, 2001.
- [11] UPPAAL: [www.uppaal.com/](http://www.uppaal.com/)
- [12] Whittlesey-Harris, S. Raquel, *Flight Software Requirements, Fluids and Combustion Facility*. National Aeronautics Space Administration, FCF-REQ-0063A.
- [13] Douglass, Bruce, *Real-Time UML: Developing Efficient Objects for Embedded Systems*. Addison Wesley Longman, Inc. 1998.
- [14] Tripakis, Stavros and Courcoubetis, Costas, *Extending PROMELA and Spin for Real-Time*. Department of Computer Science, University of Crete and Institute of Computer Science, FORTH.
- [15] Bosnacki, Dragan and Dams, Dennis *Integrating Real Time into Spin: A prototype Implementation*. Department of Math and Computer Science, Eindhoven University of Technology.
- [16] Kwiatkowska, Marta, Norman, Gethin and Parker, David, *Probabilistic Model Checking in Practice: Case Studies with PRISM*. School of Computer Science, University of Birmingham, 2005.
- [17] Approximate Probabilistic Model Checker: <http://apmc.berbiqui.org/>

## Acronyms

- APMC – Approximate Probabilistic Model Checker
- ARIS – Active Rack Isolation System
- ATCS – Air Thermal Control System
- ATCU – Air Thermal Control Unit
- CIR – Combustion Integrated Rack
- ECS – Environmental Control System
- FCU – FOMA Control Unit
- FIR – Fluids Integrated Rack
- FSAP – Fluids Science Avionics Package
- FOMA – Fuel Oxidizer and Management Assembly

IOP – Input/Output Processor

IPSU – Image Processing and Storage Unit

PI – Principal Investigator

PRISM – Probabilistic Symbolic Model  
Checker

*PROMELA* – PROcess Meta LAnguage

*RT* – Real Time

SPIN – Simple PROMELA Interpreter

UPPAAL – Uppsala and Aalborg  
Universities

UV – Ultra Violet

WTCS – Water Thermal Control System

YAG – Yttrium Aluminum Garnet

## APPENDIX A SPIN Simulation Displays

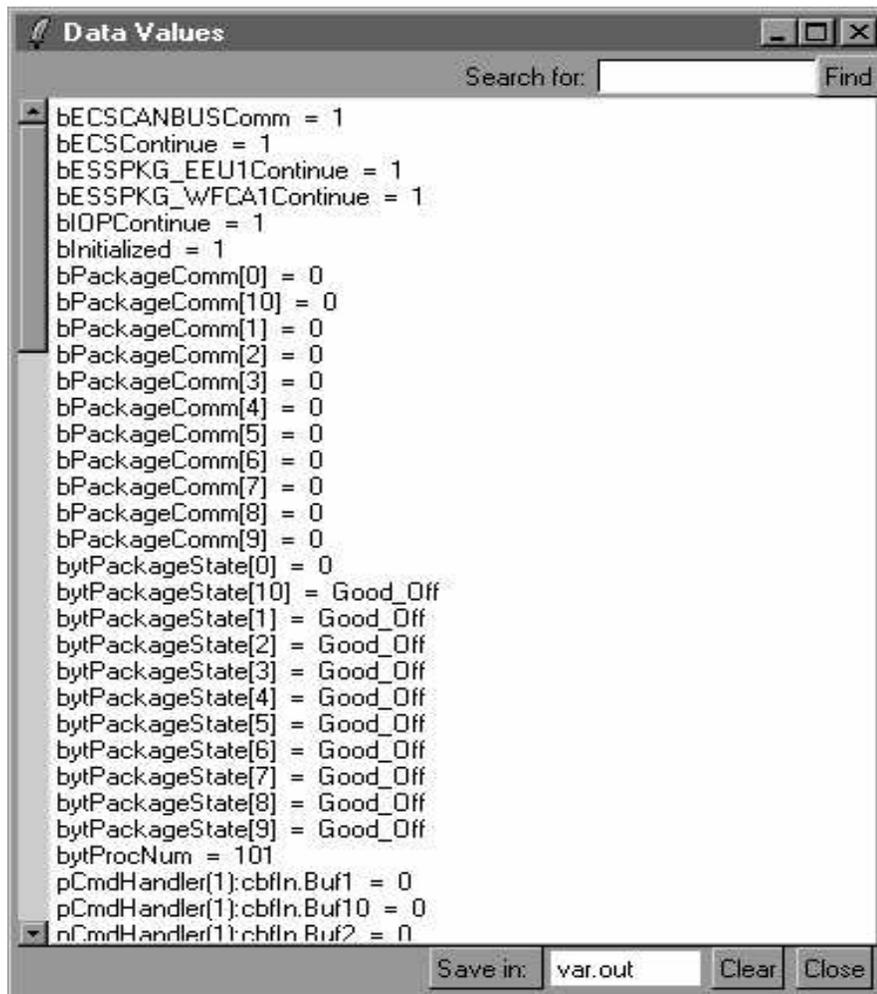


Figure 12. Variable Output View

*Particularly useful to monitor in run-time or to view post-run to get a snapshot of the final state of the system.*

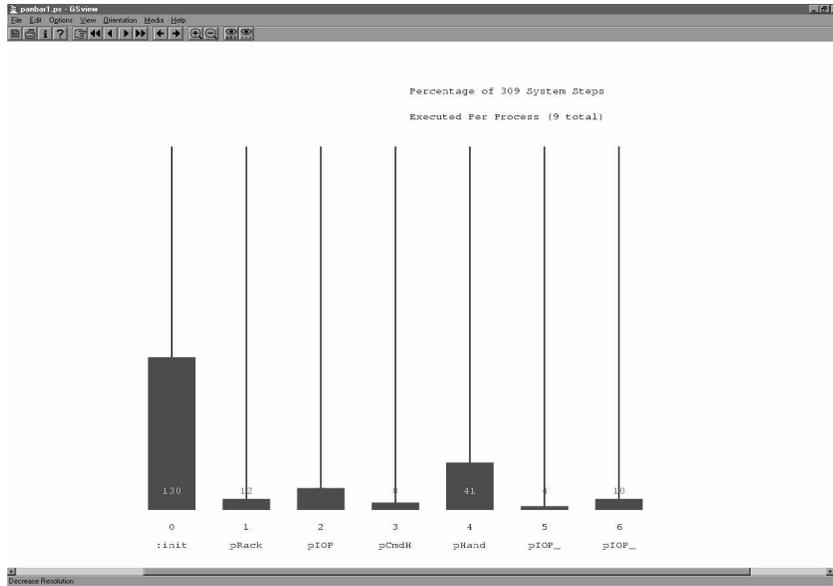


Figure 13. Process Run Percentage View

Captures the percentage of time each process runs. We found this display difficult to use since there are many processes in the FCF Model. The save feature on this display will only display the visible information (viewable). This presented a situation where all of the information could not be saved. Although not of particular importance to our research, it may be a nice metric to have.

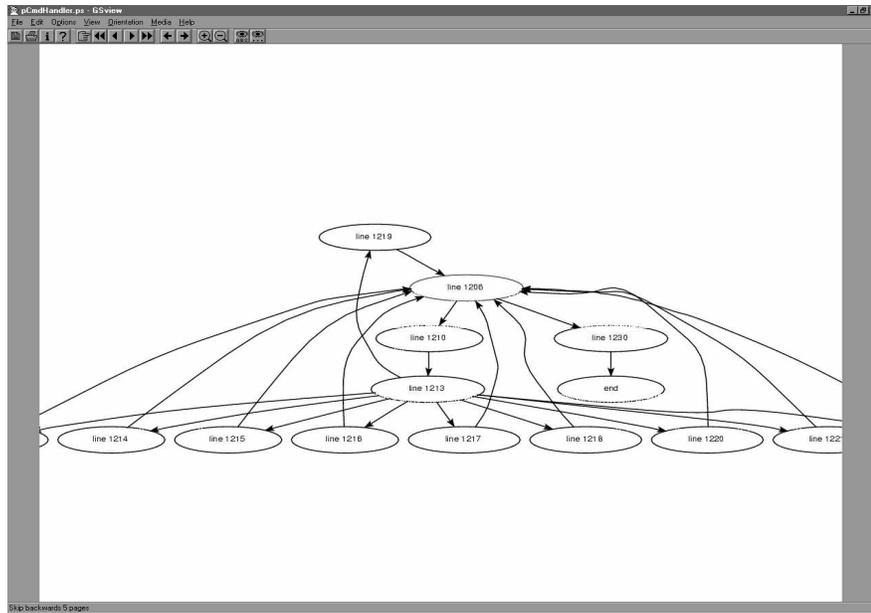


Figure 14. Process State View

Represents each state of a process and the paths taken to each state. In the Windows version, only the viewable display is savable. With Linux version of SPIN, this is not a problem.



Figure 15. Execution Sequence View

*This display provides a timeline of the execution. It shows in order each executable line and the cumulative time of execution in a time sequence.*

## APPENDIX B SPIN Verification Results

```
(Spin Version 4.1.2 -- 21 February 2004)
  + Partial Order Reduction

Bit statespace search for:
  never claim          +
  assertion violations + (if within scope of claim)
  cycle checks        - (disabled by -DSAFETY)
  invalid end states  - (disabled by never claim)

State-vector 137888 byte, depth reached 6305, errors: 0
2.85559e+06 states, stored
1.1831e+07 states, matched
1.46866e+07 transitions (= stored+matched)
  0 atomic steps
hash factor: 1.4688 (best coverage if >100)
(max size 2^22 states)

Stats on memory usage (in Megabytes):
393762.878  equivalent memory usage for states (stored*(State-vector + overhead))
1.049  memory used for hash array (-w22)
0.360  memory used for DFS stack (-m10000)
52.428 total actual memory usage
```

Figure 16. Predicate 1 Verification Results

(Spin Version 4.1.2 -- 21 February 2004)  
+ Partial Order Reduction

Bit statespace search for:

never claim	+
assertion violations	+ (if within scope of claim)
cycle checks	- (disabled by -DSAFETY)
invalid end states	- (disabled by never claim)

State-vector 137904 byte, depth reached 6305, errors: 0

2.87079e+06 states, stored

1.18518e+07 states, matched

1.47226e+07 transitions (= stored+matched)

0 atomic steps

hash factor: 1.46103 (best coverage if >100)

(max size 2<sup>22</sup> states)

Stats on memory usage (in Megabytes):

395904.356 equivalent memory usage for states (stored\*(State-vector + overhead))

1.049 memory used for hash array (-w22)

0.360 memory used for DFS stack (-m10000)

52.428 total actual memory usage

Figure 17. Predicate 2 Verification Results

(Spin Version 4.1.2 -- 21 February 2004)  
+ Partial Order Reduction

Bit statespace search for:

never claim	+
assertion violations	+ (if within scope of claim)
cycle checks	- (disabled by -DSAFETY)
invalid end states	- (disabled by never claim)

State-vector 137920 byte, depth reached 6327, errors: 0

2.55253e+06 states, stored

3.52539e+06 states, matched

6.07792e+06 transitions (= stored+matched)

0 atomic steps

hash factor: 1.64319 (best coverage if >100)

(max size  $2^{22}$  states)

Stats on memory usage (in Megabytes):

352055.424 equivalent memory usage for states (stored\*(State-vector + overhead))

1.049 memory used for hash array (-w22)

0.360 memory used for DFS stack (-m10000)

52.428 total actual memory usage

Figure 18. Predicate 3 Verification Results

(Spin Version 4.1.2 -- 21 February 2004)  
+ Partial Order Reduction

Bit statespace search for:

never claim +  
assertion violations + (if within scope of claim)  
cycle checks - (disabled by -DSAFETY)  
invalid end states - (disabled by never claim)

State-vector 137916 byte, depth reached 6337, errors: 0

2.8816e+06 states, stored

1.18932e+07 states, matched

1.47748e+07 transitions (= stored+matched)

0 atomic steps

hash factor: 1.45555 (best coverage if >100)

(max size 2<sup>22</sup> states)

Stats on memory usage (in Megabytes):

397429.996 equivalent memory usage for states (stored\*(State-vector + overhead))

1.049 memory used for hash array (-w22)

0.360 memory used for DFS stack (-m10000)

52.428 total actual memory usage

Figure 19. Predicate 4 Verification Results

(Spin Version 4.1.2 -- 21 February 2004)  
+ Partial Order Reduction

Bit statespace search for:

never claim	+
assertion violations	+ (if within scope of claim)
cycle checks	- (disabled by -DSAFETY)
invalid end states	- (disabled by never claim)

State-vector 137932 byte, depth reached 6305, errors: 0

2.92724e+06 states, stored

1.23792e+07 states, matched

1.53065e+07 transitions (= stored+matched)

0 atomic steps

hash factor: 1.43285 (best coverage if >100)

(max size  $2^{22}$  states)

Stats on memory usage (in Megabytes):

403771.225 equivalent memory usage for states (stored\*(State-vector + overhead))

1.049 memory used for hash array (-w22)

0.360 memory used for DFS stack (-m10000)

52.428 total actual memory usage

Figure 20. Predicate 5 Verification Results

(Spin Version 4.1.2 -- 21 February 2004)  
+ Partial Order Reduction

Bit statespace search for:

never claim            +  
assertion violations   + (if within scope of claim)  
cycle checks           - (disabled by -DSAFETY)  
invalid end states     - (disabled by never claim)

State-vector 137968 byte, depth reached 6313, errors: 0

2.62273e+06 states, stored

3.6787e+06 states, matched

6.30143e+06 transitions (= stored+matched)

0 atomic steps

hash factor: 1.59921 (best coverage if >100)

(max size 2<sup>22</sup> states)

Stats on memory usage (in Megabytes):

361863.855   equivalent memory usage for states (stored\*(State-vector + overhead))

1.049   memory used for hash array (-w22)

0.360   memory used for DFS stack (-m10000)

52.428 total actual memory usage

Figure 21. Predicate 6 Verification Results

(Spin Version 4.1.2 -- 21 February 2004)  
+ Partial Order Reduction

Bit statespace search for:

never claim	+
assertion violations	+ (if within scope of claim)
cycle checks	- (disabled by -DSAFETY)
invalid end states	- (disabled by never claim)

State-vector 137904 byte, depth reached 6313, errors: 0

2.53551e+06 states, stored

3.41563e+06 states, matched

5.95114e+06 transitions (= stored+matched)

0 atomic steps

hash factor: 1.65423 (best coverage if >100)

(max size  $2^{22}$  states)

Stats on memory usage (in Megabytes):

349666.837 equivalent memory usage for states (stored\*(State-vector + overhead))

1.049 memory used for hash array (-w22)

0.360 memory used for DFS stack (-m10000)

52.428 total actual memory usage

Figure 22. Predicate 7 Verification Results

(Spin Version 4.1.2 -- 21 February 2004)  
+ Partial Order Reduction

Bit statespace search for:

never claim	+
assertion violations	+ (if within scope of claim)
cycle checks	- (disabled by -DSAFETY)
invalid end states	- (disabled by never claim)

State-vector 137904 byte, depth reached 6311, errors: 0

2.4974e+06 states, stored

5.32071e+06 states, matched

7.81811e+06 transitions (= stored+matched)

0 atomic steps

hash factor: 1.67946 (best coverage if >100)

(max size 2<sup>22</sup> states)

Stats on memory usage (in Megabytes):

344411.991 equivalent memory usage for states (stored\*(State-vector + overhead))

1.049 memory used for hash array (-w22)

0.360 memory used for DFS stack (-m10000)

52.428 total actual memory usage

Figure 23. Predicate 8 Verification Results

(Spin Version 4.1.2 -- 21 February 2004)

+ Partial Order Reduction

Bit statespace search for:

never claim +  
assertion violations + (if within scope of claim)  
cycle checks - (disabled by -DSAFETY)  
invalid end states - (disabled by never claim)

State-vector 137904 byte, depth reached 6311, errors: 0

2.4974e+06 states, stored

5.32071e+06 states, matched

7.81811e+06 transitions (= stored+matched)

0 atomic steps

hash factor: 1.67946 (best coverage if >100)

(max size  $2^{22}$  states)

Stats on memory usage (in Megabytes):

344411.991 equivalent memory usage for states (stored\*(State-vector + overhead))

1.049 memory used for hash array (-w22)

0.360 memory used for DFS stack (-m10000)

52.428 total actual memory usage

Figure 24. Predicate 9 Verification Results