Kashif Javed | Elena Troubitsyna

Implementation and Verification of the Proposed Satellite System Model for Fault-Tolerance

Turku Centre *for* Computer Science

TUCS Technical Report
No 1021, October 2011

Implementation and Verification of the Proposed Satellite System Model for Fault-Tolerance

Kashif Javed
Åbo Akademi University, Department of Computer Science
kashif.javed@abo.fi
Elena Troubitsyna
Åbo Akademi University, Department of Computer Science
elena.troubitsyna@abo.fi

TUCS Technical Report
No 1021, October 2011

Abstract

The software employed to control processing and working of attitude and orbit of an satellite is extremely vital for ensuring timely completion of its planned tasks in an accurate manner. Many researchers have already done very useful work in this direction by managing various components, modes, unit states and controller mode transitions. The mechanism of error checking and correction is essentially required to avoid any disastrous behavior of the system. This research work has particularly focused on fault-tolerance procedures and their verification using SystemC that is easily translated in Promela for model checking through SPIN. Verification of the steps pertaining to forward mode transition, backward mode transition errors (both unit branch state transition errors and controller phase transition errors), and unit reconfiguration has been presented and discussed in order to ensure continuous monitoring and handling errors in accordance with the rules defined in fault-tolerance.

Keywords: Fault-Tolerant, Mode-Rich Systems, Design, Verification

TUCS Laboratory
Distributed Systems Laboratory

1 Introduction

Designing a system controlling a spacecraft is a challenging engineering task. The system should satisfy a large number of diverse functional and non-functional requirements. In particular, the designers should aim at building a fault tolerant system, i.e., the system that should cope with faults of various system components. Often behavior of satellite systems is structured using the notion of modes mutually exclusive sets of system behavior. Fault tolerance is achieved via putting the system to some degraded when an error occurs. In this report we consider an Attitude and Orbit Control System (AOCS) a generic subsystem of an spacecraft [1]. We demonstrate how to achieve fault tolerance via backward mode transitions. AOCS is a complex control system consisting of several components. To ensure correctness of mode transition we need to guarantee that all components reach a certain state. Moreover, when a component fails we need to guarantee that all other components make an appropriate backward transition. In this report we propose an approach to designing more rich system in System C. We propose an algorithm defining mode-transition scheme of AOCS. To verify correctness of our algorithm we translate it in Promela and verify using SPIN.

2 Architecture

The main purpose of AOCS (Attitude and Orbit Control System) is to control attitude and orbit [1] of a satellite. AOCS consists of a number of components – AOCS Manager, FDIR(Failure Detection, Isolation and Recovery) Manager, Mode Manager and Unit Manager. The AOCS manager plays key role while dealing with the processing of sensor data, managing actuator movements relating to the units of Reaction Wheel (RW) and Thruster (THR) and doing computation for various controls. The responsibility of FDIR is to deal timely with such tasks as failure detection, isolation and recovery. Mode transitions are handled by the Mode Manager whereas the Unit Manager deals with unit reconfigurations and unit level state transitions [2,3]. Mode and Unit Manager Architectures are further elaborated in the succeeding paragraphs.

2.1 Mode Manager

The responsibilities of mode include checking of mode transition preconditions, execution of mode transitions, management of controller phases and partially management of related units. There are six different types of controlled modes (i.e. Off, Standby, Safe, Nominal, Preparation and Science) in the mode manager and each mode has its own well-defined unique function. A brief summary of these modes is given below:

1. ***Off Mode.*** The satellite is immediately switched in the off mode as soon as the AOCS software booting is completed from the central data management unit.

2. ***Standby Mode.*** It is important to check and ensure successful separation of the spacecraft from the launcher and this work is continuously monitored and completed by the software process during the standby mode.

3. ***Safe Mode.*** Satellite enters this mode when the separation from the launcher is done. As soon as the system is in the safe mode, the relevant portions of Earth Sensor (ES), RW (Reaction Wheel) and Sun Sensor (SS) are switched to on state, the coarse pointing controller goes in the running phase and fine pointing controller is put in the idle phase. Initially the satellite acquires a stable attitude and then it achieves the coarse pointing.

4. ***Nominal Mode.*** When a mode transitions to nominal, the coarse pointing controller becomes idle and the fine pointing controller is set to the running phase. The selected branches of RW, Star Tracker (STR) and THR are switched to on state. In this mode, the satellite utilizes fine pointing control so that the Payload Instrument (PLI) in the AOCS is properly used for measurements.

5. ***Preparation Mode.*** The moment the mode is transitioned to the preparation, the concerned portion of Global Positioning System (GPS) is set to fine state, the relevant branch of PLI is switched to standby state and needed processes of RW, STR and THR go to on state. Thus, this mode ensures that the fine pointing control is reached and PLI gets ready for fulfilling its required tasks.

6. ***Science Mode.*** In science mode, the selected branch of GPS remains in the fine state, the concerned branch of PLI goes in the science state and the relevant parts of RW, STR and THR maintain their on state. Therefore, the PLI in this mode is ready to perform the tasks for which it has been designed. It stays in this mode till the completion of planned tasks.

2.2 Unit Manager

The AOCS consists of seven different units and internal state changes in these units are controlled by the unit manager. Mode manager controls the components of unit manager. Seven different controlled units are ES, SS, STR, GPS, RW, THR and PLI their brief description is as under:

1. ES is a device that measures the direction to the earth in the sensors field of view. ESs internal state is either on and off.
2. SS is a tool to measure the direction to the sun in the sensors field of view. It is also in the on or off state.
3. STR is an optical device that measures the position of stars in its field of view and performs pattern recognition on these stars in order to identify the portion of the sky at which it is looking. Two possible STRs operational states are on and off.
4. GPS is a sophisticated gadget that receives readings related to the satellite position and makes calculations to determine satellites attitude. Two possible states of GPS operation are coarse navigation and fine navigation.
5. RW is a rotating wheel which is essentially required in order to apply the required torque to the satellite. It is achieved by accelerating or breaking the wheel. RWs state can be either on or off.
6. THR is a position actuator that is used to force the satellite to change its position and its orbit by emitting gas. It can also be in either on or off state.
7. The PLI is an instrument which provides required measurements pertaining to the specific mission. It can operate in standby or science state.

3 Unit Branch States and State Transitions

Every unit is implemented as a pair of identical devices to maintain the nominal branch and the redundant branch. For each unit, one and only one branch is the selected at the time. Every selected branch is in on state and its status is locked. In other words, a branch in the off state is always allocated an unlocked status. In total, there are six states of unit components (i.e. on, off, coarse, fine, standby and science). Whenever an unit state goes from off to on, the powering takes place. Similarly, when the unit switches from on to off state, un-powering takes place. Powering and un-Powering are associated with the states and state transitions of a branch of ES, SS, STR, RW or THR. Occurrence of such states and state transitions is shown in Figure 1. For the GPS unit, unit state goes from off to coarse state and coarse to fine state, then powering and upgrading is carried out respectively. In case of fine to off state transition, first downgrading is performed then un-powering is done. States and Transitions of a Branch of GPS are depicted in Figure 2.

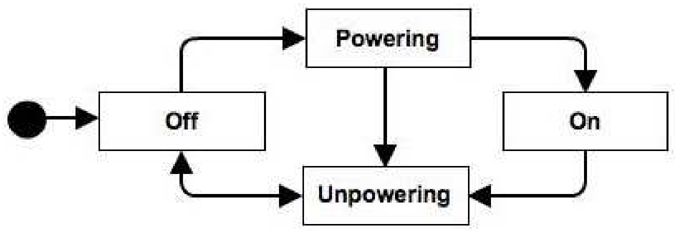

Figure 1: States and State Transitions of a Branch of ES, SS, RW, STR or THR [1]

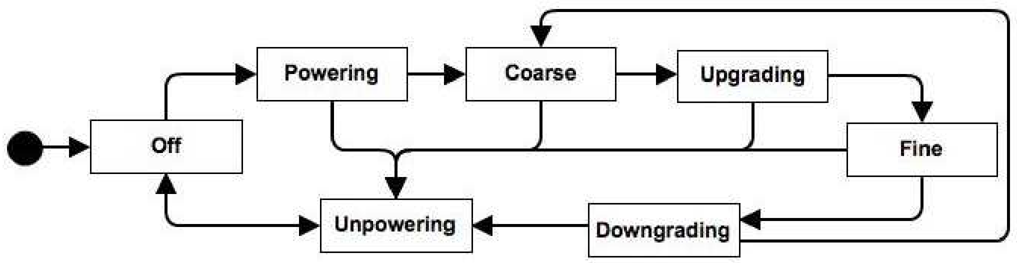

Figure 2: States and State Transitions of a Branch of GPS [1]

In case of PLI unit, when the unit state goes from off to standby and from standby to science state, then powering and upgrading is achieved respectively. In case of science to off state transition, first downgrading occurs and then un-powering takes place. Figure 3 demonstrates states and their transitions of a branch of PLI.

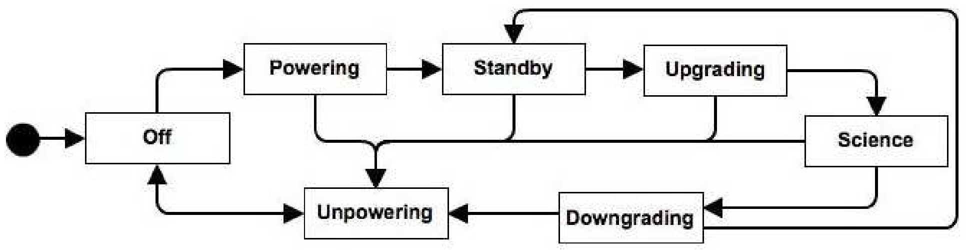

Figure 3: States and State Transitions of a Branch of PLI [1]

State transitions are very fast to accommodate time constrains for real-time satellite operations. Hence any state transition to powering, un-powering, upgrading or downgrading takes less than one AOCS cycle. However, every state transition to off takes minimum three and maximum four AOCS cycles. Any state transition to on, coarse, fine, standby or science has a success condition if the transition gets completed during the first AOCS cycle when the condition is observed to hold. However, any state transition to on, coarse, fine, standby or science is overridden if the associated success condition is not observed to hold within a predefined number of AOCS cycles from start of the transition.

4 Controller Phases and Phase Transitions

The AOCS has two controllers – Coarse Pointing Controller (CPC) and Fine Pointing Controller (FPC). The main objective of these two controllers is to direct the line of sight with a specified coarse accuracy and fine accuracy respectively. It is an essential requirement and must be met within given time limits. The following rules have to be observed during the controller phase transitions when a certain operational mode is reached:

1. Both controllers go to idle phase when the mode transition is set to off or standby state.

2. When the mode transition is switched to safe state, the CPC enters the running phase and the FPC remains in the idle phase.

3. The moment the mode transition shifts to nominal, preparation or science, the CPC goes in the idle phase and the FPC moves in the running phase.

Only one controller can be in non-idle phase at any point of time. When a controller phase has to switch from idle to running, first of all it is set to preparing. Then after predefined number of AOCS cycles, the controller is set to ready phase. Finally, the phase of controller is shifted to running as indicated in Figure 4. It can also be noticed that the controller can directly move to the idle phase from any of the other three phases (preparing, ready and running).

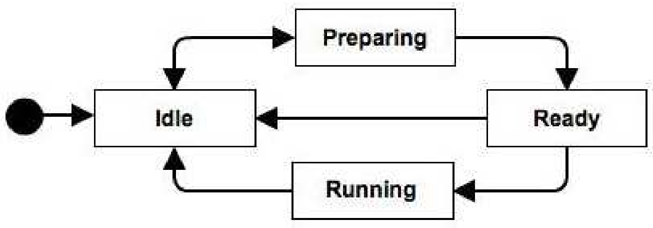

Figure 4: Phases and Phase Transitions of a Controller [1]

5 Mode Transitions

The following rules are imposed on mode transitions, to ensure correct satellite function in nominal (fault free) and faulty conditions:

1. When a mode transition to off or standby is completed, it is ensured that every branch in every unit is put in the off state.

2. On reaching to the safe mode, the selected branches of ES, RW and SS are set in the on state and all other branches pertaining to different units go to the off state.

3. In case of a transition to the nominal mode, the selected branch of GPS is turned in the coarse state, the concerned branches of RW, STR and THR are set to on state, and remaining every branch in every unit is put in the off state.

4. Completion of a mode transition to preparation ensures that the relevant branch of GPS is in the fine state, the chosen branch of PLI is in the standby state, the selected parts of RW, STR and THR are in the on state, and rest every branch in every unit is in the off state.

5. A mode transition to science requires that the needed branch of GPS is in the fine state, the selected branch of PLI is in the science state, the concerned branches of RW, STR and THR are in the on state, and all other branches pertaining to different units remain in the off state.

6 Fault Tolerance

Fault-tolerance should guarantee that the system continues to operate in predictable way even in case of failure of any of its components. Recovery from errors in fault-tolerant systems can be characterized as either roll forward or roll back. Forward error recovery aims at bringing the system to a new error-free state. Backward error recovery rolls back the system to some previous state before an error occurrence. In moderich systems the backward error recovery is achieved via backward mode transition, i.e., mode downgrading. The mode down-gradation depends on various errors, which are explained in below:

6.1 Branch State Transition Errors

A branch state transition error means that when some unit transitions to on state, the mode coarse, fine, standby or science gets overridden due to timeout condition. Because operation and state transition delays have to be avoided, we should time each mode transition. If a step of transition is not completed within a specified time limit, timeout signal is generated to get into a safe condition. The important error checks concerning to the branch state transitions are:

1. A branch state transition error on the redundant branch of ES, RW or SS causes a mode transition to off.

2. A mode transition to safe takes place when there is a branch state transition error on the redundant branch of GPS, STR or THR and there is no branch state transition error on the redundant branches of ES, RW and SS.

3. When a branch state transition error on the redundant branch of PLI occurs, it results into a mode transition to nominal provided that there is no branch state transition error on the redundant branches of ES, SS, GPS, RW, STR and THR.

6.2 Phase Transition Errors

A phase transition error or an attitude error may arise during the computations done by the selected controller. An attitude error is generated when there is a problem in the execution of an AOCS algorithm. It means that an error occurs only when one of the two controllers (i.e. CPC and FPC) is in the running phase. The key factors relating to the attitude errors are:

1. If the current mode is safe, then a non-ignored attitude error causes a transition to the off mode.

2. In case the existing mode is nominal and a nonignored attitude error occurs, a mode transition to safe takes place.

3. A mode transition to nominal takes place when the current mode is preparation and a non-ignored attitude error is generated.

4. The generation of a non-ignored attitude error moves the mode transition to preparation with the condition that the existing mode is science.

6.3 Unit Reconfiguration

Each logical unit consists of two hardware units known as nominal and redundant. Initially the nominal unit works in the active role and provides all the necessary support for normal operation of the system. The redundant unit serves as a backup resource. When an error is detected in the nominal unit, it becomes reconfigured. It means that the nominal unit is switched off and the redundant unit takes over the operational tasks. The important errors that take place during the unit reconfiguration are:

1. A branch state transition error on the nominal branch of ES, SS or RW causes a reconfiguration of the unit if there is no branch state transition error on the redundant branches of ES, SS and RW.

2. A branch state transition error on the nominal branch of GPS, STR, THR or PLI causes a reconfiguration of the unit if there is no branch state transition error on the redundant branches of ES, SS, GPS, RW, STR and THR.

7 Verification

We have implemented mode-transition algorithm shown in Figure 6 using SystemC language. The SystemC Verification Standard provides API for transaction based verification, constrained and weighted randomization, exception handling, and other verification tasks [4,5].It has got large language constructs which makes it easy to write the program with reduced efforts. System C supports the use of special data types which are often used by the hardware engineers. It comes with a strong simulation kernel to enable the designers to write good test benches for easy and speedy simulation. It is extremely important because the functional verification at the system level saves a lot of money and time.
The system architecture that is implemented in System C is verified in the SPIN model checker. SPIN [6,7,8] is often used to verify behavior of distributed and parallel systems. Its major impact has been in the area of formal verification of a number of software packages being used in distributed and parallel processing systems. SPIN major achievements have been in the areas of vital algorithms for space systems, verification of software routines for handling calls in the context of data communication for commercial purposes, inter-process communication based operating systems, verification of the mission-oriented control algorithms, measuring performance of parallel and distributed computing systems, checking performance of routing protocols [8] in the networked systems, verification of fault-tolerant application-specific schemes and implementation of a wide variety of switching methods. PROMELA (PROcess MEta LAnguage) is a high level language which is widely used to specify systems descriptions and is fully supported by SPIN for the purpose of verification of software-based applications. SPIN PROMELA is used to carry out detailed testing and verification of design and architecture of various systems.

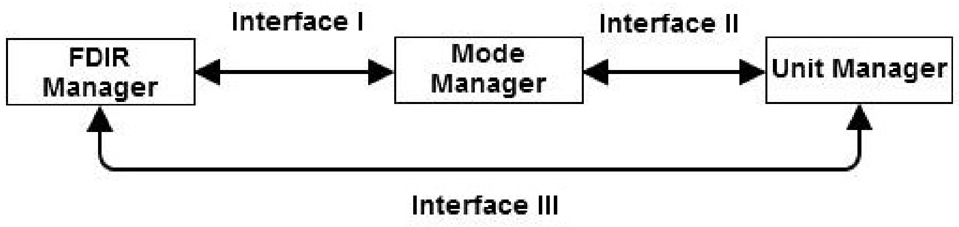

Figure 5: System Architecture [1]

The interfaces between the FDIR Manager, Mode Manager and Unit Manager indicated in Figure 5 are described in the sections below:

7.1 Interface I

When failure occurs in the system, FDIR detects the and issues the requests of mode transition, and then Mode Manager is responsible for mode transitions to the downgraded mode on the basis of error type. The following part of the code represents the Interface I scenario for Science Mode.

```
if (Mode==F) // Mode F: Science Mode
{ if (ES==off && SS==off && GPS==fine && STR==on && RW==on &&
THR==on && PLI==science && CPC==idle && FPC==run)
{/* The associated code describes that the conditions are valid
for Science Mode. The current mode is Science. */}
else if ((ES!=off || SS!=off || RW!=on) && STR==on && GPS==fine &&
THR==on && PLI==science && CPC==idle && FPC==run)
{/* The associated code describes that the conditions are not valid
for Science Mode as error occurs on the unit branch of ES, SS or RW.
It causes the mode transition to Off Mode. */}
else if ((GPS!=fine || STR!=on || THR!=on) && ES==off && SS==off &&
RW==on && PLI==science && CPC==idle && FPC==run)
{/* The associated code describes that the conditions are not valid
for Science Mode as error occurs on the unit branch of GPS, STR or
THR. It causes the mode transition to Safe Mode. */}
else if (ES==off && SS==off && GPS==fine && STR==on && RW==on &&
THR==on && PLI!=science && CPC==idle && FPC==run)
{/* The associated code describes that the conditions are not valid
for Science Mode as error occurs on the unit branch of PLI. It causes
the mode transition to Nominal Mode. */}
else if (ES==off && SS==off && GPS==fine && STR==on && RW==on &&
THR==on && PLI==science && (CPC!=idle || FPC!=run))
{/* The associated code describes that the conditions are not valid
for Science Mode as error occurs in the phase of Coarse or Fine
Pointing Controller. It causes the mode transition to Preparation Mode.
*/}
else
{/* The associated code describes that no transitions take place. */}}
else
{/* The associated code describes that invalid mode. Program terminated.*/}
```

7.2 Interface II

Unit Manager is responsible for internal state changes of the units. When all the unit states satisfy the condition (mentioned in section V) of the current mode then Mode Manager switches the mode to the next mode. The following part of the code represents the Interface II scenario for Preparation Mode.

```
if (Mode==E) // Mode E: Preparation Mode
{ if (ES==off && SS==off && GPS==fine && STR==on && RW==on &&
THR==on && PLI== standby)
{/* The associated code describes that the all unit states are
valid for Preparation Mode. The current mode is Preparation. Now
Mode Manager switches the mode to the next mode i.e. Science
Mode. */}
else if (ES!=off || SS!=off || GPS!=fine || STR!=on || RW!=on ||
THR!=on || PLI!=standby)
{/* The associated code describes that the one or more unit states
are not valid for Preparation Mode then Mode Manager cann't
switch the mode to the next mode. */}
else
{/* The associated code describes that no transitions take place.
*/}}
else
{/* The associated code describes that invalid mode. Program
terminated.*/}
```

7.3 Interface III

FDIR Manager detects the branch state errors on the nominal or redundant branches of the units. When error on the nominal branches occurs then unit reconfiguration is carried out and error on the redundant branches effects the mode transitions (mentioned in section VI). The following part of the code represents the Interface III scenario for Nominal Mode.

```
if (mode==D) // Mode D: Nominal Mode
{ /*N_ES, N_SS, N_RW, N_GPS, N_STR, N_THR and N_PLI are Nominal
Unit branches. R_ES, R_SS, R_RW, R_GPS, R_STR, R_THR and R_PLI
are Redundant Unit branches */
if (N_ES==off && N_SS==off && N_RW==on &&N_GPS==coarse &&
N_STR==on && N_THR==on &&N_PLI==off)
{/* The associated code describes that the all unit states are
valid for Nominal Mode. The current mode is Nominal. */}
else if ((N_ES!=off || N_SS!=off || N_RW!=on) && R_ES==off
&& R_SS==off && R_RW==on )
{/* The associated code describes that the error occurrs on
Nominal branch of ES, SS or RW in the Nominal Mode. Then Unit
Reconfiguration is carried out. It takes time to replace the
Nominal branch with Redundant branch. After that Nominal Mode
operation is continued. */}
else if((N_GPS!=coarse || N_STR!=on || N_THR!=on ||
N_PLI!=off) && R_GPS==coarse && R_STR==on &&
R_THR==on && R_PLI==off)
{/* The associated code describes that the error occurrs on
Nominal branch of GPS, STR, THR or PLI in the Nominal Mode.
Then Unit Reconfiguration is carried out. It takes time to
replace the Nominal branch with Redundant branch. After that
Nominal Mode operation is continued. */}
else
{/* The associated code describes that no transitions take place.
*/}}
else
{/* The associated code describes that invalid mode. Program
terminated.*/}
```

The SPINs verification model successfully checks all the global mode transitions and the fault tolerance of the system architecture. The following code shows part of the implemented SPIN PROMELA model. In this code, the operation conditions for Safe Mode are explained.

```
init
{ /*Declaration of Variables */
for (mode:1..6)
{if
::mode==A  //Associated code for Mode A: Off Mode
::mode==B  //Associated code for Mode B: Standby Mode
::mode==C  // Mode C: Safe Mode
if /* All units have two branches i.e. Nominal and Redundant */
::(N_ES==on && N_SS==on && N_RW==on && N_GPS==off && N_STR==off
&& N_THR==off && N_PLI==off && R_ES==on && R_SS==on && R_RW==on
&& R_GPS==off && R_STR==off && R_THR==off &&R_PLI==off &&
CPC ==run && FPC==idle)
{/* The associated code describes that all the conditions are
valid for Safe Mode. The current mode is Safe. Now Mode Manager
switches the mode to the next mode i.e. Nominal Mode. */}
::(N_ES==on && N_SS==on && N_RW==on && N_GPS==off && N_STR==off
&& N_THR==off && N_PLI==off && R_ES==on && R_SS==on && R_RW==on
&& R_GPS==off && R_STR==off && R_THR==off &&R_PLI==off off  &&
(CPC!=run || FPC!=idle))
{/* The associated code describes that the conditions are not
valid for Safe Mode as error occurs in the phase of Coarse or
FinePointing Controller. It causes the mode transition to Off
Mode. */}
::((R_ES!=on || R_SS==on || R_RW!=on ) && R_GPS==off &&
R_STR==off && R_THR==off && R_PLI==off && CPC==run && FPC==idle)
{/* The associated code describes that the conditions are not
valid for Safe Mode as error occurs on the redundant unit branch
of ES, SS or RW. It causes the mode transition to Off Mode. */}
::(R_ES==on && R_SS==on && R_RW==on && (R_GPS!=off || R_STR!=off
|| R_THR!=off ) && R_PLI==off&& CPC==run && FPC==idle)
{/* The associated code describes that the conditions are not
valid for Safe Mode as error occurs on the redundant unit branch
of GPS, STR or THR. It causes the mode transition again to Safe
Mode. */}
::((N_ES!=on || N_SS!=on || N_RW!=on) && R_ES==on && R_SS==on &&
R_RW==on )
{/* The associated code describes that the error occurrs on
Nominal branch of ES, SS or RW in the Safe Mode. Then Unit
Reconfiguration is carried out. It takes time to replace the
Nominal branch with Redundant branch. After completion of Safe
Mode operation, Mode Manager switches the mode to the next
mode i.e. Nominal Mode. */}
::((N_GPS!=off || N_STR!=off || N_THR!=off || N_PLI!=off) &&
R_GPS==off && R_STR==off && R_THR==off && R_PLI==off)
{/* The associated code describes that the error occurrs on
Nominal branch of GPS, STR, THR or PLI in the Safe Mode. Then
Unit Reconfiguration is carried out. It takes time to replace the
Nominal branch with Redundant branch. After completion of Safe
Mode operation, Mode Manager switches the mode to the next
mode i.e. Nominal Mode. */}
:: else
#ifdef FIX
 break
#endif
fi;
::mode==D //Associated code for Mode D: Nominal Mode
::mode==E //Associated code for Mode E: Preparation Mode
::mode==F //Associated code for Mode F: Science Mode
:: else
#ifdef FIX
 breakk
#endif
fi;} // end of for loop
} // end of init process
```

We have successfully verified forward and backward mode transitions and ensured correctness of global mode transitions with respect to component states.

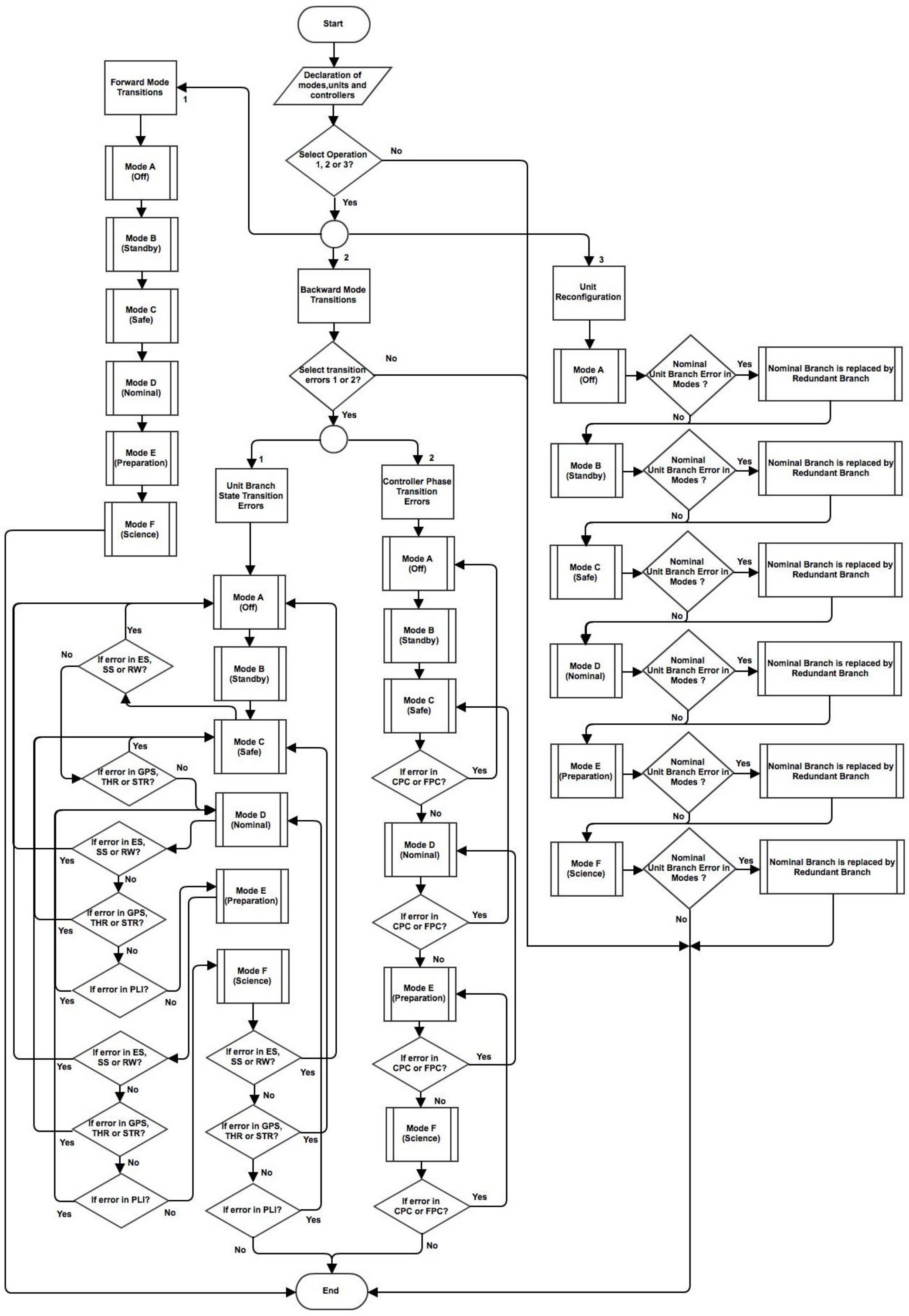
Start
Declaration of modes,units and controllers
Select Operation 1, 2 or 3?
No
Yes
1
2
3
Forward Mode Transitions
Mode A (Off)
Mode B (Standby)
Mode C (Safe)
Mode D (Nominal)
Mode E (Preparation)
Mode F (Science)
Backward Mode Transitions
Select transition errors 1 or 2?
Unit Branch State Transition Errors
Controller Phase Transition Errors
If error in ES, SS or RW?
If error in GPS, THR or STR?
If error in PLI?
If error in CPC or FPC?
Unit Reconfiguration
Nominal Unit Branch Error in Modes ?
Nominal Branch is replaced by Redundant Branch
End

Figure 6: System Flow Chart

8 Proposed System Model for Fault Tolerance

The proposed system model has been implemented using SystemC language. Because of its simulation environment, it is being used as a defacto standard in the Systems-on-a-Chip (SoC) industry. SystemC is very useful for the description of SOCs functional models at different levels of abstraction. Transaction Level Models (TLM) are easily developed in SystemC for embedded software in order to have a virtual prototype of the final chip. To validate correctness of the system model, SystemC code is translated to Promela which acts as a formal input language to the SPIN model checker. The system model comprises of six defined modes named as A (Off), B (Standby), C (Safe), D (Nominal), E (Preparation) and F (Science). Three different operational blocks have been implemented (i.e. forward mode transitions, backward mode transitions, and unit reconfiguration). The flow chart given in Figure 6 shows detailed design structure of the system. After necessary declarations of modes, units and controllers, the different verification procedures pertaining to the operational blocks are described in the following sections.

8.1 Verification of Forward Mode Transition

When all the unit states are in off state, controller phases are in idle phase, and no unit reconfiguration is in progress, then current mode is A and the error flag is set to low. After this, the mode moves forward to the next mode (i.e. Mode B). So in this way, when all conditions of unit states and controller phases within each mode fulfill their requirements, then the current mode switches to the next mode until it completes its operation after Mode F.

8.2 Verification of the Steps in the Backward Mode Transition

There are two types of backward mode transition errors: unit branch state transition errors, and controller phase transition errors. Fault-tolerance procedure for handling such type of errors is very challenging and is explained below.

1. ***Verification of the Steps in Unit Branch State Transition Errors.*** Initially, all the unit states are in the off state and the controller phases are in idle phase. No unit reconfiguration is in progress and the current mode is A. The error flag is set to low. The mode transition to the Mode B is autonomous for constantly monitoring and

ensuring of separation of the spacecraft from the launcher. The system automatically goes into the Mode C after completing the task of Mode B. If there is an error in ES, SS or RW, the system goes back to Mode A. In case of an error in GPS, THR or STR, it enters in Mode C. Otherwise, the satellite has attained secure attitude and required coarse pointing. Now it switches to Mode D. From Mode D, in case of an error in ES, SS or RW, the system moves back to Mode A. If there is an error in GPS, THR or STR, it enters in Mode C. If it encounters an error in PLI, it remains in Mode D. Otherwise, it goes to Mode E after fully utilizing FPC in order to prepare the PLI for measurements. If there is an error in ES, SS or RW at Mode E, the system again switches to Mode A. If an error occurs in GPS, THR or STR, it returns to Mode C. However, if PLI gets an error, it goes to Mode D. In case of no errors, it enters in Mode F. In Mode E, proper FPC is ensured so that the PLI becomes ready to complete the required measurements. While in Mode F, if it receives an error in ES, SS or RW, the system moves to A. In case of an error in GPS, THR or STR, it switches to Mode C. If there is an error in PLI, it returns to Mode D. If no error has been encountered during this mode, it means that the system has successfully completed its assigned tasks.

2. ***Verification of the Steps in Unit Controller Phase Transition Errors.*** When both the controllers of preparation mode are in the running phase, the error flag is set to high and the mode is downgraded to nominal mode. Likewise if error occurs in any controller phase or both controllers are in running phase in any mode, the mode is downgraded in accordance with the principles defined in fault-tolerance. It is evident from the Figure 6 that the Modes A, B and C are processed normally like forward mode transitions. After Mode C, if there is an error in CPC or FPC, the system goes back to Mode A. Otherwise, it moves to Mode D. At Mode D, in case of an error in CPC or FPC, it shifts back to Mode C, and goes to Mode E if no error is encountered. Similarly, the mode switches from E to D and F to E if there is an error at Modes E and F respectively. Likewise, the system moves from Mode E to Mode F if there is no error. Now it completes the given tasks as long as they are required

8.3 Verification of the Steps in Unit Reconfiguration

When nominal branch of ES unit in safe mode and nominal branch of THR unit in preparation mode are in the off state, an error has occurred in the nominal unit branch. So, unit reconfiguration is done so that the redundant unit branches of these units are taken into use to complete the remaining operation of the system. Similarly, if an error occurs in any nominal branch of any unit in any mode, the unit reconfiguration is done according to the principles that are defined in the fault-tolerance. Unit reconfiguration is, however, a burden on the system and takes some time while switching from nominal branch to redundant branch of the unit.

9 Conclusions and Future Work

Controlling attitude and orbit of the satellite is very demanding for the researchers in order to complete critical-mission-oriented tasks. The AOCS system contains a number of components to deal with processing of data, managing of movements and operation of its different units. The mode and unit managers play vital role in checking and managing mode and unit state transitions. The mode manager deals with six different types of controlled modes and the unit manager handles seven different types of units. All unit branch states and state transitions have been elaborated for required function of unit components. Rules to be observed during the controller phases and phase transitions have been highlighted to ensure that the CPC and FPC properly monitor the line of sight with desired accuracy.
A number of for mode transitions procedures and rules have also presented for correct functioning of the proposed system. The design of the system has laid special emphasis on the fault-tolerance measures so that error-free operation is maintained. Branch state transition errors, phase transition error and unit reconfiguration fault-tolerance strategies have been included in system design and are also accordingly explained for correcting and handling errors. The proposed system has been implemented in SystemC language as it is being used as a defacto verification standard in the SoC industry. SystemC is easily meshed with Promela which works as the input language to SPIN for model checking and verification. The design of the proposed system model has been presented in this report and verification steps pertaining to unit branch transition errors, controller phase transition errors and unit reconfiguration have been described The formal modeling undertaken in [9,10] aimed at enabling proof-based verification of mode-rich systems modeled in Event-B. In [11] the authors perform failure modes and effect analysis of each particular mode transition to systematically design mode transition scheme. Our work aims at building a gap between formal specification and code. This motivated our choice of System C as a design language and model-checking based verification. There is a need to investigate design and verification of decentralized mode rich systems. For future research work, an effort will be made to ensure the correctness of mode transitions when various mode

managers are interacting with each other in order to avoid transition errors.

References

[1] DEPLOY Work Package 3 - Attitude and Orbit Control System Software Requirements Document, Space Systems Finland, Ltd., December 2010.

[2] M. Heimdahl and N. Leveson, Completeness and Consistency in Hierarchical State-Based Requirements, IEEE Transactions on Software Engineering, Vol.22, No. 6, June 1996, pp. 363-377.

[3] M. Heimdahl and N. Leveson, Completeness and Consistency in Hierarchical State-Based Requirements, IEEE Transactions on Software Engineering, Vol.22, No. 6, June 1996, pp. 363-377.

[4] C. Ip and S. Swan, A tutorial introduction on the new SystemC verification standard, Technical report, www.systemc.org, 2003.

[5] L. Singh and L. Drucker, Advanced Verification Techniques : A SystemC Based Approach for Successful Tapeout, Springer, 2004.

[6] J. Katoen, Concepts, Algorithms and Tools for Model Checking", Lecture Notes, Chapter 1: System Validation, 1999.

[7] N. A. S. A. Larc, What is Formal Methods?", NASA Langley Methods, http://shemesh.larc.nasa.gov/fm/fmwhat. html, formal methods program, 2001.

[8] Kashif Javed, Asifa Kashif and Elena Troubitsyna, Implementation of SPIN Model Checker for Formal Verification of Distance Vector Routing Protocol, International Journal of Computer Science and Information Security (IJCSIS), Vol 8, No 3, June 2010, USA, ISSN 1947-5500, pp. 1-6.

[9] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Dubravka Ilic and Timo Latvala. Developing Mode-Rich Satellite Software by Refinement in Event B . In Proceedings of FMICS 2010, the 15th International Workshop on Formal Methods for Industrial Critical Systems, September 2010, LNCS 6371. Springer.

[10] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky and Kimmo Varpaaniemi, Pauli Visnen. Verifying Mode Consistency for On-Board Satellite Software, 2010, LNCS 6351, Computer Safety, Reliability, and Security, Pages 126-141, Springer

[11] Yuliya Prokhorova, Elena Troubitsyna, Linas Laibinis, Kimmo Varpaaniemi and Timo Latvala. Derivation and Formal Verification of a Mode Logic for Layered Control Systems. Asia-Pacific Software Engineering Conference. IEEE Computer, December 2011. To appear.

Appendix

System C

```
#include <iostream>
#include <windows.h>
#include <stdio.h>
using namespace std;
void mode_operation(int,int,int,int,int,int,int,int,int,int);
void controller_phase(int,int);
int main()
{char c;bool b=true;
// unit states
const int off=0;const int on=1;const int coarse=2;const int fine=3;
const int unit=0;const int Standby=4;const int Science=5;
const int idle=0;const int run=1;int FPC_phase,CPC_phase,in,scenario;
// Each unit has two branches i.e. Nominal and Redundant
int ES,SS,GPS,STR,RW,THR,PLI;
int R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI; //Redundant branch of units
int N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI; //Nominal branch of units
do
{cout<<"\t\t\tWelcome to the AOCS"<<"\n\n\n"
cout<<"Would you like to run this program"<<"\n";
cout<<"Press ’Y’ for Yes"<<"\n"<<"Press ’N’ for No"<<endl;
cin>>c;
cout<<endl;
if(c == ’y’ || c == ’Y’)b=true;
else if (c == ’n’ || c == ’N’)b=false;
else cout<<"Please enter either ’y’ or ’n’"<<endl;}
while ((c!=’y’) && (c!=’Y’) && (c!=’n’)&& (c!=’N’));
if (c == ’n’ || c == ’N’) cout<<"Program terminated"<<endl;
while(b)
{if(c == ’y’ || c == ’Y’)
cout<<"There are six types of Mode.\n\n";
cout<<"Mode A : Off\n\n";cout<<"Mode B : Standby\n\n";
cout<<"Mode C : Safe\n\n";cout<<"Mode D : Nominal\n\n";
cout<<"Mode E : Preparation\n\n";
cout<<"Mode F : Science\n\n";
cout<<"Select one of the following operation\n\n";
cout<<"1.\tForward Mode Transitions for AOCS\n"<<endl;
cout<<"2.\tBranch State Transition Errors Check\n"<<endl;
```

```
cout<<"3.\tAttitude Errors Check\n"<<endl;
cout<<"4.\tUnit Reconfiguration\n"<<endl;
cout<<"Please press '1' , '2' , '3' or '4'"<<endl;
cin>>in;
cout<<endl;
cout<<"Note:\n";
cout<<"'0'denotes 'Off state' & 'Idle phase', '1' denotes 'On state'";
cout<<"&'Running phase'\n"<<"'2' denotes 'Coarse state', '3' denotes";
coutt<<" 'Fine state', '4' denotes 'Standby state' & '5' denotes";
cout<<" 'Science state'\n\n";
if(in==1)   //  Normal mode transitions handled
{for (int mode = 1; mode < 7; mode++)
{if(mode==1){ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;
CPC_phase=idle;FPC_phase=idle;
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);}
else if(mode==2){ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;
CPC_phase=idle;FPC_phase=idle;
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);}
else if(mode==3){ES=on;SS=on;GPS=off;STR=off;RW=on;THR=off;PLI=off;
CPC_phase=run;FPC_phase=idle;
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);}
else if(mode==4){ES=off;SS=off;GPS=coarse;STR=on;RW=on;THR=on;PLI=off;
CPC_phase=idle;FPC_phase=run;
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);}
else if(mode==5){ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;
PLI=Standby;CPC_phase=idle;FPC_phase=run;
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);}
else if(mode==6){ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;
PLI=Science;CPC_phase=idle;FPC_phase=run;
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);}
else cout<<"Program Terminated\n";}
else if(in==2)
{for (int mode = 1; mode < 7; mode++)
{if (mode==1){R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;R_THR=off;
R_PLI=off;CPC_phase=idle;FPC_phase=idle;
mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,mode);}
else if (mode==2){R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;
R_THR=off;R_PLI=off;CPC_phase=idle;FPC_phase=idle;
mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,mode);}
else if (mode==3)
{R_ES=on;R_SS=on;R_RW=on;R_GPS=off;R_STR=off;R_THR=off;R_PLI=off;
```

```
CPC_phase=run;FPC_phase=idle;
if (R_ES==on && R_SS==on && R_RW==on && R_GPS==off && R_STR==off &&
R_THR==off && R_PLI==off){mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,
R_THR,R_PLI,CPC_phase,FPC_phase,mode);}
else if ((R_ES!=on || R_SS!=on || R_RW!=on) && R_GPS==off && R_STR==off
&& R_THR==off && R_PLI==off){R_ES=off;R_SS=off;R_GPS=off;R_STR=off;
R_RW=off;R_THR=off;R_PLI=off;CPC_phase=idle;FPC_phase=idle;scenario=7;
mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario);goto stop;}
else if (R_ES==on && R_SS==on && R_RW==on && (R_GPS!=off || R_STR!=off
|| R_THR!=off) && R_PLI==off){R_ES=on;R_SS=on;R_GPS=off;R_STR=off;
R_RW=on;R_THR=off;R_PLI=off;CPC_phase=run;FPC_phase=idle;scenario=8;
mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario);goto stop;}
else cout<<"\n\n";}
else if (mode==4)
{R_ES=off;R_SS=off;R_RW=on;R_GPS=coarse;R_STR=on;R_THR=on;R_PLI=off;
CPC_phase=idle;FPC_phase=run;
if (R_ES==off && R_SS==off && R_RW==on && R_GPS==coarse && R_STR==on
&& R_THR==on && R_PLI==off){mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,
R_THR,R_PLI,CPC_phase,FPC_phase,mode);}
else if ((R_ES!=off || R_SS!=off || R_RW!=on) && R_GPS==coarse &&
R_STR==on && R_THR==on && R_PLI==off)
{R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;R_THR=off;R_PLI=off;
CPC_phase=idle;FPC_phase=idle;scenario=9;
mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario);goto stop;}
else if (R_ES==off && R_SS==off && R_RW==on && (R_GPS!=coarse ||
R_STR!=on || R_THR!=on)&&R_PLI==off){R_ES=on;R_SS=on;R_GPS=off;
R_STR=off;R_RW=on;R_THR=off;R_PLI=off;CPC_phase=run;FPC_phase=idle;
scenario=10;mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,
CPC_phase,FPC_phase,scenario);goto stop;}
else if (R_ES==off && R_SS==off && R_RW==on && R_GPS==coarse &&
R_STR==on && R_THR==on && R_PLI!=off){R_ES=off;R_SS=off;R_GPS=coarse;
R_STR=on;R_RW=on;R_THR=on;R_PLI=off;CPC_phase=idle;FPC_phase=run;
scenario=11;mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,
CPC_phase,FPC_phase,scenario);goto stop;}else cout<<"\n\n";}
else if(mode==5){R_ES=off;R_SS=off;R_GPS=fine;R_STR=on;R_RW=on;
R_THR=on;R_PLI=Standby;CPC_phase=idle;FPC_phase=run;
if (R_ES==off && R_SS==off && R_RW==on && R_GPS==fine && R_STR==on
&& R_THR==on && R_PLI==Standby){mode_operation(R_ES,R_SS,R_GPS,R_STR,
R_RW,R_THR,R_PLI,CPC_phase,FPC_phase,mode);}
else if ((R_ES!=off || R_SS!=off || R_RW!=on) && R_GPS==fine &&
R_STR==on && R_THR==on && R_PLI==Standby)
{R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;R_THR=off;R_PLI=off;
CPC_phase=idle;FPC_phase=idle;scenario=12;mode_operation(R_ES,R_SS,
R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,FPC_phase,scenario);goto stop;}
else if (R_ES==off && R_SS==off && R_RW==on && (R_GPS!=fine || R_STR!=on
|| R_THR!=on) && R_PLI==Standby){R_ES=on;R_SS=on;R_GPS=off;R_STR=off;
R_RW=on;R_THR=off;R_PLI=off;CPC_phase=run;FPC_phase=idle;scenario=13;
mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario);goto stop;}
else if (R_ES==off && R_SS==off && R_RW==on && R_GPS==fine &&
R_STR==on && R_THR==on && R_PLI!=Standby){R_ES=off;R_SS=off;
R_GPS=coarse;R_STR=on;R_RW=on;R_THR=on;R_PLI=off;CPC_phase=idle;
FPC=run;scenario=14;mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,
R_PLI,CPC_phase,FPC_phase,scenario);goto stop;}else cout<<"\n\n";}
else if (mode==6)
{R_ES=off;R_SS=off;R_GPS=off;R_STR=on;R_RW=on;R_THR=on;R_PLI=Science;
CPC_phase=idle;FPC_phase=run;
if (R_ES==off && R_SS==off && R_RW==on && R_GPS==fine && R_STR==on &&
R_THR==on && R_PLI==Science){mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,
R_THR,R_PLI,CPC_phase,FPC_phase,mode);}
else if ((R_ES!=off || R_SS!=off || R_RW!=on) && R_GPS==fine &&
R_STR==on && R_THR==on && R_PLI==Science){R_ES=off;R_SS=off;R_GPS=off;
R_STR=off;R_RW=off;R_THR=off;R_PLI=off;CPC_phase=idle;FPC_phase=idle;
scenario=15;mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,
CPC_phase,FPC_phase,scenario);goto stop;}
else if (R_ES==off && R_SS==off && R_RW==on && (R_GPS!=fine ||
R_STR!=on|| R_THR!=on) && R_PLI==Science)
{R_ES=on;R_SS=on;R_GPS=off;R_STR=off;R_RW=on;R_THR=off;R_PLI=off;
CPC_phase=run;FPC_phase=idle;scenario=16;
mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario);goto stop;}
else if (R_ES==off && R_SS==off && R_RW==on && R_GPS==fine &&
R_STR==on && R_THR==on && R_PLI!=Science){R_ES=off;R_SS=off;
R_GPS=coarse;R_STR=on;R_RW=on;R_THR=on;R_PLI=off;CPC_phase=idle;
FPC_phase=run;scenario=17;mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,
R_THR,R_PLI,CPC_phase,FPC_phase,scenario);goto stop;}else cout<<"\n";}
else {cout<<"Program Terminated\n"; goto stop;}}}
else if (in==3)
{for (int mode = 1; mode < 7; mode++)
{if(mode==1){ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;
CPC_phase=idle;FPC_phase=idle;
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);}
else if(mode==2){ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;
```

```
CPC_phase=idle;FPC_phase=idle;
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);}
else if(mode==3){ES=on;SS=on;GPS=off;STR=off;RW=on;THR=off;PLI=off;
CPC_phase=run;FPC_phase=idle;
if (CPC_phase==run && FPC_phase==idle)
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);
else if (CPC_phase==run && FPC_phase==run)
{ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;CPC_phase=idle;
FPC_phase=idle;scenario=18;cout<<"To the Safe Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Safe Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}
else if (CPC_phase==idle && FPC_phase==idle)
{ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;CPC_phase=idle;
FPC_phase=idle;scenario=18;cout<<"To the Safe Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Safe Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}
else if (CPC_phase==idle && FPC_phase==run)
{ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;CPC_phase=idle;
FPC_phase=idle;scenario=18;cout<<"To the Safe Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Safe Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}else {cout<<"\n\n";}}
else if (mode==4)
{ES=off;SS=off;GPS=coarse;STR=on;RW=on;THR=on;PLI=off;CPC_phase=idle;
FPC_phase=run;
if (CPC_phase==idle && FPC_phase==run)
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);
else if (CPC_phase==run && FPC_phase==run)
{ES=on;SS=on;GPS=off;STR=off;RW=on;THR=off;PLI=off;CPC_phase=run;
FPC_phase=idle;scenario=35;cout<<"To the Nominal Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Nominal Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}
else if (CPC_phase==idle && FPC_phase==idle)
{ES=on;SS=on;GPS=off;STR=off;RW=on;THR=off;PLI=off;CPC_phase=run;
FPC_phase=idle;scenario=19;cout<<"To the Nominal Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Nominal Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}
else if (CPC_phase==run && FPC_phase==idle)
{ES=on;SS=on;GPS=off;STR=off;RW=on;THR=off;PLI=off;CPC_phase=run;
```

```
FPC_phase=idle;scenario=19;cout<<"To the Nominal Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Nominal Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}else {cout<<"\n\n";}}
else if (mode==5){ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;PLI=Standby;
CPC_phase=run;FPC_phase=run;
if (CPC_phase==idle && FPC_phase==run)
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);
else if (CPC_phase==run && FPC_phase==run){ES=off;SS=off;GPS=coarse;
STR=on;RW=on;THR=on;PLI=off;CPC_phase=idle;FPC_phase=run;scenario=20;
cout<<"To the Preparation Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Preparation Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}
else if (CPC_phase==idle && FPC_phase==idle){ES=off;SS=off;GPS=coarse;
STR=on;RW=on;THR=on;PLI=off;CPC_phase=idle;FPC_phase=run;scenario=20;
cout<<"To the Preparation Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Preparation Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}
else if (CPC_phase==run && FPC_phase==idle){ES=off;SS=off;GPS=coarse;
STR=on;RW=on;THR=on;PLI=off;CPC_phase=idle;FPC_phase=run;scenario=20;
cout<<"To the Preparation Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Preparation Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}else {cout<<"\n\n";}}
else if(mode==6){ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;PLI=Science;
CPC_phase=run;FPC_phase=run;
if (CPC_phase==idle && FPC_phase==run)
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode);
else if (CPC_phase==run && FPC_phase==run)
{ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;PLI=Standby;CPC_phase=idle;
FPC_phase=run;scenario=21;cout<<"To the Science Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Science Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}
else if (CPC_phase==idle && FPC_phase==idle){ES=off;SS=off;GPS=fine;
STR=on;RW=on;THR=on;PLI=Standby;CPC_phase=idle;FPC=run;scenario=21;
cout<<"To the Science Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Science Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}
else if (CPC_phase==run && FPC_phase==idle){ES=off;SS=off;GPS=fine;
```

```
STR=on;RW=on;THR=on;PLI=Standby;CPC_phase=idle;FPC_phase=run;
scenario=21;cout<<"To the Science Mode\n\n";
cout<<"Error occurrs in the Controllers phase in Science Mode\n\n";
mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,scenario);
goto stop;}else {cout<<"\n\n";}}else cout<<"Program Terminated\n";
goto stop;} } }
else if (in==4) //Reconfiguration handling
{for (int mode = 1; mode < 7; mode++)
{if (mode==1){N_ES=off;N_SS=off;N_RW=off;N_GPS=off;N_STR=off;N_THR=off;
N_PLI=off;CPC_phase=idle;FPC_phase=idle;mode_operation(N_ES,N_SS,N_GPS,
N_STR,N_RW,N_THR,N_PLI,CPC_phase,FPC_phase,mode);}
else if (mode==2){N_ES=off;N_SS=off;N_RW=off;N_GPS=off;N_STR=off;
N_THR=off;N_PLI=off;CPC_phase=idle;FPC_phase=idle;
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);}
else if (mode==3){N_ES=on;N_SS=off;N_RW=on;N_GPS=off;N_STR=off;
N_THR=off;N_PLI=off;R_ES=on;R_SS=on;R_GPS=off;R_STR=off;R_RW=on;
R_THR=off;R_PLI=off;CPC_phase=run;FPC_phase=idle;
if (N_ES==on && N_SS==on && N_RW==on && N_GPS==off && N_STR==off &&
N_THR==off && N_PLI==off)
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);
else if ((N_ES!=on || N_SS!=on || N_RW!=on) && R_ES==on && R_SS==on
&& R_RW==on )
{cout<<"To the Safe Mode\n"<<"Error occurrs in nominal unit branch\n";
cout<<"Please wait! Unit Reconfigured\n";Sleep(5000);
N_ES=R_ES;N_SS=R_SS;N_RW=R_RW;
cout<<"The redundant branch is taken into use\n\n";cout<<"Again ";
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);}
else if((N_GPS!=off || N_STR!=off || N_THR!=off || N_PLI!=off) &&
R_GPS==off && R_STR==off && R_THR==off && R_PLI==off)
{cout<<"To the Safe Mode \n"<<"Error occurrs in nominal unit branch";
cout<<"Please wait! Unit Reconfigured\n";Sleep(5000);
N_GPS=R_GPS;N_STR=R_STR;N_THR=R_THR;N_PLI=R_PLI;
cout<<"The redundant branch is taken into use\n\n";cout<<"Again ";
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);}else {goto stop;} }
else if (mode==4){N_ES=off;N_SS=off;N_RW=off;N_GPS=coarse;N_STR=on;
N_THR=on;N_PLI=off;R_ES=off;R_SS=off;R_RW=on;R_GPS=coarse;R_STR=on;
R_THR=on;R_PLI=off;CPC_phase=idle;FPC_phase=run;
if (N_ES==off && N_SS==off && N_RW==on && N_GPS==coarse && N_STR==on &&
N_THR==on && N_PLI==off)
```

```
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);
else if ((N_ES!=off || N_SS!=off || N_RW!=on) && R_ES==off && R_SS==off
&& R_RW==on){cout<<"To the Nominal Mode\n\n";
cout<<"Error occurrs in nominal branch of Unit\n";
cout<<"Please wait! Unit Reconfigured\n";Sleep(5000);
N_ES=R_ES;N_SS=R_SS;N_RW=R_RW;
cout<<"The redundant branch is taken into use\n\n";cout<<"Again ";
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);}
else if((N_GPS!=coarse || N_STR!=on || N_THR!=on || N_PLI!=off) &&
R_GPS==coarse && R_STR==on && R_THR==on && R_PLI==off)
{cout<<"To the Nominal Mode\n\n";
cout<<"Error occurrs in nominal branch of Unit\n";
cout<<"Please wait! Unit Reconfigured\n";Sleep(5000);
N_GPS=R_GPS;N_STR=R_STR;N_THR=R_THR;N_PLI=R_PLI;
cout<<"The redundant branch is taken into use\n\n";cout<<"Again ";
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);}else {goto stop;} }
else if (mode==5){N_ES=on;N_SS=off;N_RW=on;N_GPS=fine;N_STR=on;
N_THR=on;N_PLI=Standby;R_ES=off;R_SS=off;R_GPS=fine;R_STR=on;R_RW=on;
R_THR=on;R_PLI=Standby;CPC_phase=idle;FPC_phase=run;
if (N_ES==off && N_SS==off && N_RW==on && N_GPS==fine && N_STR==on
&& N_THR==on && N_PLI==Standby)
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);
else if ((N_ES!=off || N_SS!=off || N_RW!=on) && R_ES==off &&
R_SS==off && R_RW==on ){cout<<"To the Preparation Mode\n\n";
cout<<"Error occurrs in nominal branch of Unit\n";
cout<<"Please wait! Unit Reconfigured\n";Sleep(5000);
N_ES=R_ES;N_SS=R_SS;N_RW=R_RW;
cout<<"The redundant branch is taken into use\n\n";cout<<"Again ";
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);}
else if((N_GPS!=fine || N_STR!=on || N_THR!=on || N_PLI!=Standby) &&
R_GPS==fine && R_STR==on && R_THR==on && R_PLI==Standby)
{cout<<"To the Preparation Mode\n\n";
cout<<"Error occurrs in nominal branch of Unit\n";
cout<<"Please wait! Unit Reconfigured\n";Sleep(5000);
N_GPS=R_GPS;N_STR=R_STR;N_THR=R_THR;N_PLI=R_PLI;
cout<<"The redundant branch is taken into use\n\n";cout<<"Again ";
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);}else {goto stop;} }
```

```
else if (mode==6){N_ES=off;N_SS=off;N_RW=on;N_GPS=fine;N_STR=off;
N_THR=on;N_PLI=Science;R_ES=off;R_SS=off;R_GPS=fine;R_STR=on;R_RW=on;
R_THR=on;R_PLI=Science;CPC_phase=idle;FPC_phase=run;
if (N_ES==off && N_SS==off && N_RW==on && N_GPS==fine && N_STR==on &&
N_THR==on && N_PLI==Science)mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,
N_THR,N_PLI,CPC_phase,FPC_phase,mode);
else if ((N_ES!=off || N_SS!=off || N_RW!=on) && R_ES==off && R_SS==off
&& R_RW==on ){cout<<"To the Science Mode\n\n";
cout<<"Error occurrs in nominal branch of Unit\n";
cout<<"Please wait! Unit Reconfigured\n";Sleep(5000);
N_ES=R_ES;N_SS=R_SS;N_RW=R_RW;
cout<<"The redundant branch is taken into use\n\n";cout<<"Again ";
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode);}
else if((N_GPS!=fine || N_STR!=on || N_THR!=on || N_PLI!=Science) &&
R_GPS==fine && R_STR==on && R_THR==on && R_PLI==Science)
{cout<<"To the Science Mode"<<"Error occurrs in nominal unit branch";
cout<<"Please wait! Unit Reconfigured\n";Sleep(5000);
N_GPS=R_GPS;N_STR=R_STR;N_THR=R_THR;N_PLI=R_PLI;
cout<<"The redundant branch is taken into use\n\n";cout<<"Again ";
mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC,mode);}else {goto stop;}}else cout<<"Program Terminated\n";}}
else if (in<1 || in>4){cout<<"Invalid number entered\n";goto stop;}
else{cout<<"Program Terminated\n";goto stop;}
stop:cout<<"\n\n\n\t\t\tWelcome to the AOCS"<<"\n\n\n"<<"Do you want
to continue this program"<<"\n";
cout<<"Press 'Y' for Yes"<<"\n"<<"Press 'N' for No"<<endl;
    char bb;cin >> bb;cout<<endl;
    if (bb == 'y' || bb == 'Y') b = true;
    else{cout<<"Program terminated"<<endl;b = false;}
    if ( c == 'n') cout<<"Program terminated"<<endl;}
system("PAUSE");return 0; }
void mode_operation(int ES,int SS,int GPS,int STR,int RW,int THR,
int PLI,int CPC_phase,int FPC_phase, int scenario)
{bool low= false;bool high= true;bool error_flag;
if(scenario==1)
{cout<<"To the Off Mode\n\n";cout<<" ES = "<<ES<<", SS = "<<SS<<",
GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW<<", THR = "<<THR<<",
PLI = "<<PLI;cout<<"\nEach Unit status is unlocked in this mode";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration";
error_flag=low;cout<<"There is no error in a given mode.Error Flag is";
cout<<error_flag<<"\n";cout<<"Current Mode is A i.e. Off mode\n\n";}
```

```
else if (scenario==2)
{cout<<"To the Standby Mode\n\n";cout<<" ES = "<<ES<<", SS = "<<SS<<",
GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW<<", THR = "<<THR<<",
PLI = "<<PLI;cout<<"\nEach Unit status is unlocked in this mode";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration";
error_flag=low;cout<<"There is no error in a given mode.Error Flag is";
cout<<error_flag<<"\n";cout<<"Current Mode is B i.e.Standby mode\n\n";}
else if (scenario==3){cout<<"To the Safe Mode\n\n";
cout<<" GPS = "<<GPS<<", STR = "<<STR<<", THR = "<<THR<<", PLI = "<<PLI;
Sleep(2500);cout<<", ES = "<<ES<<", SS = "<<SS<<", RW = "<<RW;
cout<<"\nES, SS and RW have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration";
error_flag=low;cout<<"There is no error in a given mode.Error Flag is";
cout<<error_flag<<"\n";cout<<"Current Mode is C i.e.Safe mode\n\n";}
else if (scenario==4){cout<<"To the Nominal Mode\n\n";
cout<<" RW = "<<RW<<", PLI = "<<PLI;
Sleep(2500);cout<<", GPS = "<<GPS<<", STR = "<<STR<<", THR = "<<THR;
Sleep(12500);cout<<", ES = "<<ES<<", SS = "<<SS;
cout<<"\nGPS, STR, RW and THR have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration";
error_flag=low;cout<<"There is no error in a given mode.Error Flag is";
cout<<error_flag<<"\n";cout<<"Current Mode is D i.e.Nominal mode\n";}
else if (scenario==5)
{cout<<"To the Preparation Mode\n\n";
cout<<" ES = "<<ES<<", SS = "<<SS<<", STR = "<<STR<<", RW = "<<RW<<",
THR = "<<THR;Sleep(2500);cout<<", GPS = "<<GPS<<", PLI = "<<PLI;
cout<<"\nGPS, STR, RW, THR and PLI have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration";
error_flag=low;cout<<"\nNo error in a given mode.Error Flag is";
cout<<error_flag;cout<<"\nCurrent Mode is E i.e.Preparation mode";}
else if (scenario==6)
{cout<<"To the Science Mode\n\n";cout<<" ES = "<<ES<<", SS = "<<SS<<",
GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW<<", THR = "<<THR;
Sleep(2500);cout<<", PLI = "<<PLI;
cout<<"\nGPS, STR, RW, THR and PLI have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration";
error_flag=low;cout<<"\nNo error in a given mode. Error Flag is ";
cout<<error_flag;cout<<"\nCurrent Mode is F i.e. Science mode\n\n";}
```

```
else if (scenario==7){
// Safe to Off mode, when error occurrs in ES,SS or RW.
cout<<"To the Safe Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\nBack To the Off Mode"<<"All units are going to Off state\n";
if(ES==off)
{cout<<" ES = "<<ES<<", GPS = "<<GPS<<", STR = "<<STR<<", THR = "<<
THR<<", PLI = "<<PLI;Sleep(15000);cout<<", SS = "<<SS<<", RW = "<<RW;}
else if(SS==off)
{cout<<" SS = "<<SS<<", GPS = "<<GPS<<", STR = "<<STR<<", THR = "<<
THR<<", PLI = "<<PLI;Sleep(15000);cout<<", ES = "<<ES<<", RW = "<<RW;}
else if(RW==off)
{cout<<" RW = "<<RW<<", GPS = "<<GPS<<", STR = "<<STR<<", THR = "<<
THR<<", PLI = "<<PLI;Sleep(15000);cout<<", ES = "<<ES<<", SS = "<<SS;}
else {cout<<" ";}cout<<"\nEach Unit status is unlocked in this mode";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is A i.e. Off mode\n\n";}
else if (scenario==8){
// Safe to Safe mode, when error occurrs in GPS,STR or THR.
cout<<"To the Safe Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in given mode.Error Flag is"<<error_flag;
cout<<"\n\nBack To the Safe Mode\n\n"<<"All units are roll-backed\n";
if(GPS!=off)
{cout<<" ES = "<<ES<<", SS = "<<SS<<", STR = "<<STR<<", RW = "<<RW<<",
THR = "<<THR<<", PLI = "<<PLI;Sleep(15000);cout<<", GPS = "<<GPS;}
else if(STR!=off)
{cout<<" ES = "<<ES<<", SS = "<<SS<<", GPS = "<<GPS<<", RW = "<<RW<<",
THR = "<<THR<<", PLI = "<<PLI;Sleep(15000);cout<<", STR = "<<STR;}
else if(THR!=off)
{cout<<" ES = "<<ES<<", SS = "<<SS<<", GPS = "<<GPS<<", STR = "<<STR
<<", RW = "<<RW<<", PLI = "<<PLI;Sleep(15000);cout<<", THR = "<<THR;}
else {cout<<"\n\n";}cout<<"\nES, SS and RW have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is C i.e. Safe mode\n\n";}
else if (scenario==9){
// Nominal to Off mode, when error occurrs in ES,SS or RW.
cout<<"To the Nominal Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\nBack To the Off Mode\n"<<"All units are going to Off state\n";
if((ES!=off){cout<<" SS = "<<SS<<", PLI = "<<PLI;Sleep(15000);
cout<<", ES = "<<ES<<", GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW
<<", THR = "<<THR;}
else if(SS!=off){cout<<" ES = "<<ES<<", PLI = "<<PLI;Sleep(15000);
cout<<", SS = "<<SS<<", GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW
<<", THR = "<<THR;}
else if(RW!=on){cout<<" ES = "<<ES<<", SS = "<<SS<<", RW = "<<RW<<",
PLI = "<<PLI;Sleep(15000);cout<<", GPS = "<<GPS<<", STR = "<<STR<<",
THR = "<<THR;}else {cout<<"\n\n";}
cout<<"\nEach Unit status is unlocked in this mode";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is A i.e. Off mode\n\n";}
else if (scenario==10){
// Nominal to Safe mode, when error occurrs in GPS,STR or THR.
cout<<"To the Nominal Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\n\nBack To the Safe Mode\n\n"<<"All units are roll-backed\n";
if(GPS!=coarse){cout<<" GPS = "<<GPS<<", RW = "<<RW<<", PLI = "<<PLI;
Sleep(2500);cout<<", ES = "<<ES<<", SS = "<<SS;Sleep(12500);
cout<<", STR = "<<STR<<", THR = "<<THR;}
else if(STR!=on){cout<<" STR = "<<STR<<", RW = "<<RW<<", PLI = "<<PLI;
Sleep(2500);cout<<", ES = "<<ES<<", SS = "<<SS;Sleep(12500);
cout<<", GPS = "<<GPS<<", THR = "<<THR;}
else if(THR!=on){cout<<" THR = "<<THR<<", RW = "<<RW<<", PLI = "<<PLI;
Sleep(2500);cout<<", ES = "<<ES<<", SS = "<<SS;Sleep(12500);
cout<<", STR = "<<STR<<", GPS = "<<GPS;}
else {cout<<"\n\n";}cout<<"\nES, SS and RW have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is C i.e. Safe mode\n\n";}
else if (scenario==11){
// Nominal to Nominal mode, when error occurrs in PLI.
cout<<"To the Nominal Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"Back To the Nominal Mode\n\n";
cout<<" ES = "<<ES<<", SS = "<<SS<<", GPS = "<<GPS<<", STR = "<<STR<<",
RW = "<<RW<<", THR = "<<THR;Sleep(15000);cout<<", PLI = "<<PLI;
cout<<"\nGPS, STR, RW and THR have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is D i.e. Nominal mode\n\n";}
else if (scenario==12){
// Preparation to Off mode, when error occurrs in ES,SS or RW.
cout<<"To the Preparation Mode\n\n";error_flag=high;
```

```
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\nBack To the Off Mode\n"<<"All units are going to Off state";
if(ES!=off){cout<<" SS = "<<SS;Sleep(15000);
cout<<", ES = "<<ES<<", GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW
<<", THR = "<<THR<<", PLI = "<<PLI;}
else if(SS!=off){cout<<" ES = "<<ES;Sleep(15000);
cout<<", SS = "<<SS<<", GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW
<<", THR = "<<THR<<", PLI = "<<PLI;}
else if(RW!=on){cout<<" RW = "<<RW;Sleep(15000);
cout<<", ES = "<<ES<<", SS = "<<SS<<", GPS = "<<GPS<<", STR = "<<STR
<<", THR = "<<THR<<", PLI = "<<PLI;}
else {cout<<"\n\n";}cout<<"\nEach Unit status is unlocked in mode";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is A i.e. Off mode\n\n";}
else if (scenario==13){
// Preparation to Safe mode, when error occurrs in GPS,STR or THR.
cout<<"To the Preparation Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"Back To the Safe Mode\n\n"<<"All units are roll-backed\n";
if(GPS!=fine){cout<<" GPS = "<<GPS<<", RW = "<<RW;Sleep(2500);
cout<<", ES = "<<ES<<", SS = "<<SS;Sleep(12500);
cout<<", STR = "<<STR<<", THR = "<<THR<<", PLI = "<<PLI;}
else if(STR!=on){cout<<" STR = "<<STR<<", RW = "<<RW;
Sleep(2500);cout<<", ES = "<<ES<<", SS = "<<SS;Sleep(12500);
cout<<", GPS = "<<GPS<<", THR = "<<THR<<", PLI = "<<PLI;}
else if(THR!=on){cout<<" THR = "<<THR<<", RW = "<<RW;
Sleep(2500);cout<<", ES = "<<ES<<", SS = "<<SS;Sleep(12500);
cout<<", GPS = "<<GPS<<", STR = "<<STR<<", PLI = "<<PLI;}
else {cout<<"\n\n";}cout<<"\nES, SS and RW have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is C i.e. Safe mode\n\n";}
else if (scenario==14){
// Preparation to Nominal mode, when error occurrs in PLI.
cout<<"To the Preparation Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"Back To the Nominal Mode\n\n";
cout<<" ES = "<<ES<<", SS = "<<SS<<", STR = "<<STR<<", RW = "<<RW<<",
THR = "<<THR;Sleep(2500);cout<<", GPS = "<<GPS;Sleep(12500);
cout<<", PLI = "<<PLI<<"\nGPS, STR, RW and THR have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
```

```
cout<<"Now Current Mode is D i.e. Nominal mode\n\n";}
else if (scenario==15){
// Science to Off when error occurrs in ES,SS or RW.
cout<<"To the Science Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\nBack To the Off Mode\n"<<"All units are going to Off state\n";
if(ES!=off){cout<<" SS = "<<SS;Sleep(15000);
cout<<", ES = "<<ES<<", GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW
<<", THR = "<<THR<<", PLI = "<<PLI;}
else if(SS!=off){cout<<" ES = "<<ES;Sleep(15000);
cout<<", SS = "<<SS<<", GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW
<<", THR = "<<THR<<", PLI = "<<PLI;}
else if(RW!=on){cout<<" ES = "<<ES<<", SS = "<<SS<<" RW = "<<RW;
Sleep(15000);cout<<", GPS = "<<GPS<<", STR = "<<STR<<", THR = "<<THR
<<", PLI = "<<PLI;}
else {cout<<"\n\n";}cout<<"\nEach Unit status is unlocked in mode";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is A i.e. Off mode\n\n";}
else if (scenario==16){
// Science to Safe mode, when error occurrs in GPS,STR or THR.
cout<<"To the Science Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\nBack To the Safe Mode\n\n"<<"All units are roll-backed\n";
if(GPS!=fine){cout<<" GPS = "<<GPS<<", RW = "<<RW;Sleep(2500);
cout<<", ES = "<<ES<<", SS = "<<SS;Sleep(12500);
cout<<", STR = "<<STR<<", THR = "<<THR<<", PLI = "<<PLI;}
else if(STR!=on)
{cout<<" STR = "<<STR<<", RW = "<<RW;
Sleep(2500);cout<<", ES = "<<ES<<", SS = "<<SS;
Sleep(12500);cout<<", GPS = "<<GPS<<", THR = "<<THR<<", PLI = "<<PLI;}
else if(THR!=on)
{cout<<" THR = "<<THR<<", RW = "<<RW;
Sleep(2500);cout<<", ES = "<<ES<<", SS = "<<SS;
Sleep(12500);cout<<", GPS = "<<GPS<<", STR = "<<STR<<", PLI = "<<PLI;}
else {cout<<"\n\n";}cout<<"\nES, SS and RW have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is C i.e. Safe mode\n\n";}
else if (scenario==17){
// Science to Nominal mode, when error occurrs in PLI.
cout<<"To the Science Mode\n\n";error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
```

```
cout<<"\nBack To the Nominal Mode\n\n";cout<<" ES = "<<ES<<", SS = "
<<SS<<", STR = "<<STR<<", RW = "<<RW<<", THR = "<<THR;Sleep(2500);
cout<<", GPS = "<<GPS;Sleep(12500);cout<<", PLI = "<<PLI;
cout<<"\nGPS, STR, RW and THR have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Now Current Mode is D i.e. Nominal mode\n\n";}
else if(scenario==18) {error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\n\nBack To the Off Mode\n\n";cout<<" ES = "<<ES<<", SS = "<<SS
<<", GPS = "<<GPS<<", STR = "<<STR<<", RW = "<<RW<<", THR = "<<THR<<",
PLI = "<<PLI;cout<<"\nEach Unit status is unlocked in this mode";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations,No ongoing unit reconfiguration\n";
cout<<"Current Mode is A i.e. Off mode\n\n";}
else if (scenario==19) {error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\nBack To the Safe Mode\n\n";cout<<" GPS = "<<GPS<<", STR = "<<
STR<<", THR = "<<THR<<", PLI = "<<PLI;cout<<", ES = "<<ES<<", SS = "<<
SS<<", RW = "<<RW; cout<<"\nES, SS and RW have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Current Mode is C i.e. Safe mode\n\n";}
else if (scenario==20) {error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"Back To the Nominal Mode\n\n"<<" RW = "<<RW<<", PLI = "<<PLI;
cout<<", GPS = "<<GPS<<", STR = "<<STR<<", THR = "<<THR<<", ES = "<<ES
<<", SS = "<<SS;cout<<"\nGPS, STR, RW and THR have the status locked";
controller_phase(CPC_phase,FPC_phase);
cout<<"\nFrom these observations, No ongoing unit reconfiguration\n";
cout<<"Current Mode is D i.e. Nominal mode\n\n";}
else if (scenario==21) {error_flag=high;
cout<<"Unit error occurrs in the given mode.Error Flag is"<<error_flag;
cout<<"\nBack To the Preparation Mode\n\n";cout<<" ES = "<<ES<<", SS =
"<<SS<<", STR = "<<STR<<", RW = "<<RW<<", THR = "<<THR<<", GPS = "<<GPS
<<", PLI = "<<PLI<<"GPS, STR, RW, THR and PLI have the status locked";
controller_phase(CPC_phase,FPC_phase);cout<<"\nFrom these observations,
No ongoing unit reconfiguration\n"<<"Current Mode is E i.e. Preparation
mode\n\n";}else {cout<<"\n\n";} }
void controller_phase(int CPC_phase,int FPC_phase)
{int AOCS_cycles=10;AOCS_cycles=AOCS_cycles*1000; //time = 10sec
int idle=0;int run=1;
if(CPC_phase==idle && FPC_phase==idle)
```

```
{cout<<"\nCoarse pointing Controller phase is "<<CPC_phase;
cout<<"\nFine pointing Controller phase is "<<FPC_phase;}
else if (CPC_phase==run && FPC_phase==idle)
{cout<<"\nCoarse pointing Controller phase is preparing now";
Sleep(AOCS_cycles);
cout<<"\nCoarse pointing Controller phase is set to Ready";
cout<<"\nCoarse pointing Controller phase is set to "<<CPC_phase;
cout<<"\nFine pointing Controller phase is "<<FPC_phase;}
else if (CPC_phase==idle && FPC_phase==run)
{cout<<"\nCoarse pointing Controller phase is "<<CPC_phase;
cout<<"\nFine pointing Controller phase is preparing now";
Sleep(AOCS_cycles);
cout<<"\nFine pointing Controller phase is set to Ready";
cout<<"\nFine pointing Controller phase is set to "<<FPC_phase;}
else {cout<<"\n\n";} }
```

Spin Promela

```
active proctype controllerphase(int CPC_phase,FPC_phase)
{if
::(CPC_phase==0 && FPC_phase==0) ->
printf("\nCoarse pointing Controller phase is %d",CPC_phase);
printf("\nFine pointing Controller phase is %d",FPC_phase)
:: (CPC_phase==1 && FPC_phase==0) ->
printf("\nCoarse pointing Controller phase is preparing now");
printf("\nCoarse pointing Controller phase is set to Ready");
printf("\nCoarse pointing Controller phase is set to %d",CPC_phase);
printf("\nFine pointing Controller phase is %d",FPC_phase)
:: (CPC_phase==0 && FPC_phase==1) ->
printf("\nCoarse pointing Controller phase is %d",CPC_phase);
printf("\nFine pointing Controller phase is preparing now");
printf("\nFine pointing Controller phase is set to Ready");
printf("\nFine pointing Controller phase is set to %d",FPC_phase)
:: else
#ifdef FIX
-> break
#endif
fi }
active proctype mode_operation(int ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,
FPC_phase,scenario)
{#define low false
#define high true
bool error_flag;
if
:: scenario==1 ->
Off: printf("To the Off Mode\n\n");
printf(" ES = %d, SS = %d, GPS = %d, STR = %d, RW = %d, THR = %d, PLI
= %d",ES,SS,GPS,STR,RW,THR,PLI);
printf("\nEach Unit status is unlocked in this mode");
run controllerphase(CPC_phase,FPC_phase);
printf("\nFrom these observations, No ongoing unit reconfiguration");
error_flag=low;
printf("No error in a given mode.Error Flag is %d",error_flag);
printf("Current Mode is A i.e. Off mode\n\n")
:: scenario==2 ->
printf("To the Standby Mode\n\n");
printf(" ES = %d,SS = %d,GPS = %d,STR = %d,RW = %d,THR = %d,PLI = %d"
,ES,SS,GPS,STR,RW,THR,PLI);printf("\nEach Unit status is unlocked");
run controllerphase(CPC_phase,FPC_phase);
```

```
printf("\nFrom these observations, No ongoing unit reconfiguration");
error_flag=low;
printf("\nNo error in a given mode. Error Flag is %d",error_flag);
printf("Current Mode is B i.e. Standby mode\n\n")
:: scenario==3 ->
Safe: printf("To the Safe Mode\n\n");
printf(" GPS = %d,STR = %d,THR = %d,PLI = %d,ES = %d,SS = %d,RW = %d"
,GPS,STR,THR,PLI,ES,SS,RW);printf("\nES, SS and RW are locked");
run controllerphase(CPC_phase,FPC_phase);
printf("\nFrom these observations, No ongoing unit reconfiguration");
error_flag=low;
printf("\nNo error in a given mode. Error Flag is %d\n",error_flag);
printf("Current Mode is C i.e. Safe mode\n\n")
:: scenario==4 ->
Nominal: printf("To the Nominal Mode\n\n");
printf(" RW = %d,PLI = %d,GPS = %d,STR = %d,THR = %d,ES = %d,SS = %d"
,RW,PLI,GPS,STR,THR,ES,SS);printf("\nGPS, STR, RW and THR are locked");
run controllerphase(CPC_phase,FPC_phase);
printf("\nFrom these observations,No ongoing unit reconfiguration");
error_flag=low;
printf("\nNo error in a given mode. Error Flag is %d\n",error_flag);
printf("Current Mode is D i.e. Nominal mode\n\n")
:: scenario==5 ->
Preparation: printf("To the Preparation Mode\n\n");
printf(" ES = %d,SS = %d,STR = %d,RW = %d,THR = %d,GPS = %d,PLI = %d"
,ES,SS,STR,RW,THR,GPS,PLI);
printf("\nGPS, STR, RW, THR and PLI have the status locked");
run controllerphase(CPC_phase,FPC_phase); error_flag=low;
printf("\nFrom these observations, No ongoing unit reconfiguration");
printf("\nNo error in a given mode. Error Flag is %d\n",error_flag);
printf("Current Mode is E i.e. Preparation mode\n\n")
:: scenario==6 ->printf("To the Science Mode\n\n");
printf(" ES = %d,SS = %d,GPS = %d,STR = %d,RW = %d,THR = %d,PLI = %d"
,ES,SS,GPS,STR,RW,THR,PLI);
printf("\nGPS, STR, RW, THR and PLI have the status locked");
run controllerphase(CPC_phase,FPC_phase); error_flag=low;
printf("\nFrom these observations, No ongoing unit reconfiguration");
printf("\nNo error in a given mode. Error Flag is %d\n",error_flag);
printf("Current Mode is F i.e. Science mode\n\n")
:: (scenario==7) ->
/* Safe to Off mode, when error occurrs in ES,SS or RW.   */
printf("To the Safe Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Off
:: (scenario==8) ->
/* Safe to Safe mode, when error occurrs in GPS,STR or THR.    */
printf("To the Safe Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("Back");goto Safe
:: (scenario==9) ->
/* Nominal to Off mode, when error occurrs in ES,SS or RW.   */
printf("To the Nominal Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Off
:: (scenario==10) ->
/* Nominal to Safe mode, when error occurrs in GPS,STR or THR.   */
printf("To the Nominal Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Safe
:: scenario==11 ->
/* Nominal to Nominal mode, when error occurrs in PLI.    */
printf("To the Nominal Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("Back");goto Nominal
:: (scenario==12) ->
/* Preparation to Off mode, when error occurrs in ES,SS or RW.    */
printf("To the Preparation Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Off
:: (scenario==13) ->
/* Preparation to Safe mode, when error occurrs in GPS,STR or THR.*/
printf("To the Preparation Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Safe
:: (scenario==14) ->
/* Preparation to Nominal mode, when error occurrs in PLI. */
printf("To the Preparation Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Nominal
:: (scenario==15) ->
/* Science to Off when error occurrs in ES,SS or RW.  */
printf("To the Science Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are going to Off state\n");
printf("All units are roll-backed\n");printf("Back");goto Off
:: (scenario==16) ->
```

```
/* Science to Safe mode, when error occurrs in GPS,STR or THR.  */
printf("To the Science Mode\n\n");
error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Safe
:: (scenario==17) ->
/* Science to Nominal mode, when error occurrs in PLI.  */
printf("To the Science Mode\n\n");error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Nominal
:: scenario==18 ->error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Off
:: scenario==19 ->error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
pprintf("All units are roll-backed\n");printf("Back");goto Safe
:: scenario==20 ->error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Nominal
:: scenario==21 ->error_flag=high;
printf("Unit error occurrs in the mode. Error Flag is %d",error_flag);
printf("All units are roll-backed\n");printf("Back");goto Preparation
:: else -> skip
fi; }
init /*init Process for Normal Operation of AOCS */
{int ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode;
#define off 0
#define on 1
#define coarse 2
#define fine 3
#define Standby 4
#define Science 5
#define idle 0
#define running 1
for (mode : 1..6)
{if
::mode==1 -> ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;
CPC_phase=idle;FPC_phase=idle;mode=1;
run modeoperation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::mode==2 -> ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;
CPC_phase=idle;FPC_phase=idle;mode=2;
run modeoperation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::mode==3 -> ES=on;SS=on;GPS=off;STR=off;RW=on;THR=off;PLI=off;
```

```
CPC_phase=running;FPC_phase=idle;mode=3;
run modeoperation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::mode==4 -> ES=off;SS=off;GPS=coarse;STR=on;RW=on;THR=on;PLI=off;
CPC_phase=idle;FPC_phase=running;mode=4;
run modeoperation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::mode==5 -> ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;PLI=Standby;
CPC_phase=idle;FPC_phase=running;mode=5;
run modeoperation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::mode==6 -> ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;PLI=Science;
CPC_phase=idle;FPC_phase=running;mode=6;
run modeoperation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
:: else -> skip
fi; } }
init /* init Process for Branch State Transition Errors Check */
{int CPC_phase,FPC_phase,mode,scenario;
int R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI;//Redundant branch of units
/* unit states */
#define off 0
#define on 1
#define coarse 2
#define fine 3
#define Standby 4
#define Science 5
#define idle 0
#define running 1
for (mode : 1..6)
{if
:: mode==1 ->R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;
R_THR=off;R_PLI=off;CPC_phase=idle;FPC_phase=idle;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,mode)
:: mode==2 ->R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;
R_THR=off;R_PLI=off;CPC_phase=idle;FPC_phase=idle;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,mode)
:: mode==3 ->R_ES=on;R_SS=on;R_RW=on;R_GPS=off;R_STR=off;R_THR=off;
R_PLI=off;CPC_phase=running;FPC_phase=idle;
if
::(R_ES==on && R_SS==on && R_RW==on && R_GPS==off && R_STR==off &&
R_THR==off && R_PLI==off) ->
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,mode)
::((R_ES!=on || R_SS!=on || R_RW!=on)&& R_GPS==off && R_STR==off &&
R_THR==off && R_PLI==off) ->
R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;R_THR=off;R_PLI=off;
CPC_phase=idle;FPC_phase=idle;scenario=7;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
::(R_ES==on && R_SS==on && R_RW==on && (R_GPS!=off || R_STR!=off ||
R_THR!=off) && R_PLI==off) ->
R_ES=on;R_SS=on;R_GPS=off;R_STR=off;R_RW=on;R_THR=off;R_PLI=off;
CPC_phase=running;FPC_phase=idle;scenario=8;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
:: else -> skip
fi;
:: mode==4 ->R_ES=off;R_SS=off;R_RW=on;R_GPS=coarse;R_STR=on;
R_THR=on;R_PLI=off;CPC_phase=idle;FPC_phase=running;
if
::(R_ES==off && R_SS==off && R_RW==on && R_GPS==coarse && R_STR==on &&
R_THR==on && R_PLI==off) ->
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,mode)
::((R_ES!=off || R_SS!=off || R_RW!=on) && R_GPS==coarse && R_STR==on
&& R_THR==on && R_PLI==off) ->
R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;R_THR=off;R_PLI=off;
CPC_phase=idle;FPC_phase=idle;scenario=9;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
::(R_ES==off && R_SS==off && R_RW==on && (R_GPS!=coarse || R_STR!=on
|| R_THR!=on) && R_PLI==off) ->
R_ES=on;R_SS=on;R_GPS=off;R_STR=off;R_RW=on;R_THR=off;R_PLI=off;
CPC_phase=running;FPC_phase=idle;scenario=10;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
::(R_ES==off && R_SS==off && R_RW==on && R_GPS==coarse && R_STR==on &&
R_THR==on && R_PLI!=off) ->
R_ES=off;R_SS=off;R_GPS=coarse;R_STR=on;R_RW=on;R_THR=on;R_PLI=off;
CPC_phase=idle;FPC_phase=running;scenario=11;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
:: else -> skip
fi;
:: mode==5 ->R_ES=off;R_SS=off;R_GPS=fine;R_STR=on;R_RW=on;R_THR=on;
R_PLI=Standby;CPC_phase=idle;FPC_phase=running;
if
```

```
::(R_ES==off && R_SS==off && R_RW==on && R_GPS==fine && R_STR==on &&
R_THR==on && R_PLI==Standby) ->
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,mode)
::((R_ES!=off || R_SS!=off || R_RW!=on) && R_GPS==fine && R_STR==on
&& R_THR==on && R_PLI==Standby) ->
R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;R_THR=off;R_PLI=off;
CPC_phase=idle;FPC_phase=idle;scenario=12;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
  ::(R_ES==off && R_SS==off && R_RW==on && (R_GPS!=fine || R_STR!=on ||
R_THR!=on) && R_PLI==Standby) ->
R_ES=on;R_SS=on;R_GPS=off;R_STR=off;R_RW=on;R_THR=off;R_PLI=off;
CPC_phase=running;FPC_phase=idle;scenario=13;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
::(R_ES==off && R_SS==off && R_RW==on && R_GPS==fine && R_STR==on &&
R_THR==on && R_PLI!=Standby) ->
R_ES=off;R_SS=off;R_GPS=coarse;R_STR=on;R_RW=on;R_THR=on;R_PLI=off;
CPC_phase=idle;FPC_phase=running;scenario=14;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
::else -> skip
fi;
:: mode==6 ->R_ES=off;R_SS=off;R_GPS=off;R_STR=on;R_RW=on;R_THR=on;
R_PLI=Science;CPC_phase=idle;FPC_phase=running;
if
::(R_ES==off && R_SS==off && R_RW==on && R_GPS==fine && R_STR==on &&
R_THR==on && R_PLI==Science) ->
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,mode)
::((R_ES!=off || R_SS!=off || R_RW!=on) && R_GPS==fine && R_STR==on &&
R_THR==on && R_PLI==Science) ->
R_ES=off;R_SS=off;R_GPS=off;R_STR=off;R_RW=off;R_THR=off;R_PLI=off;
CPC_phase=idle;FPC_phase=idle;scenario=15;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
::(R_ES==off && R_SS==off && R_RW==on && (R_GPS!=fine || R_STR!=on ||
R_THR!=on) && R_PLI==Science) ->
R_ES=on;R_SS=on;R_GPS=off;R_STR=off;R_RW=on;R_THR=off;R_PLI=off;
CPC_phase=running;FPC_phase=idle;scenario=16;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
```

```
::(R_ES==off && R_SS==off && R_RW==on && R_GPS==fine && R_STR==on &&
R_THR==on && R_PLI!=Science) ->
R_ES=off;R_SS=off;R_GPS=coarse;R_STR=on;R_RW=on;R_THR=on;R_PLI=off;
CPC_phase=idle;FPC_phase=running;scenario=17;
run mode_operation(R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI,CPC_phase,
FPC_phase,scenario)
::else -> skip
fi;
::else -> skip
fi} }
init /* init Process for Attitude Errors Check */
{int ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode,scenario;
#define off 0
#define on 1
#define coarse 2
#define fine 3
#define Standby 4
#define Science 5
#define idle 0
#define running 1
for (mode : 1..6)
{if
:: mode==1 ->ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;
CPC_phase=idle;FPC_phase=idle;
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
:: mode==2 ->ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;
CPC_phase=idle;FPC_phase=idle;
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
:: mode==3 ->ES=on;SS=on;GPS=off;STR=off;RW=on;THR=off;PLI=off;
CPC_phase=running;FPC_phase=idle;
if
::(CPC_phase==running && FPC_phase==idle) ->
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::((CPC_phase==running && FPC_phase==running) || (CPC_phase==idle &&
FPC_phase==idle) || (CPC_phase==idle && FPC_phase==running)) ->
ES=off;SS=off;GPS=off;STR=off;RW=off;THR=off;PLI=off;CPC_phase=idle;
FPC_phase=idle;s=18;printf("To the Safe Mode\n\n");
printf("Error occurrs in the Controllers phase in Safe Mode\n\n");
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,s)
:: else -> skip
fi;
:: mode==4 ->ES=off;SS=off;GPS=coarse;STR=on;RW=on;THR=on;PLI=off;
CPC_phase=idle;FPC_phase=running;
```

```
if
::(CPC_phase==idle && FPC_phase==running) ->
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::((CPC_phase==running && FPC_phase==running) || (CPC_phase==idle &&
FPC_phase==idle) || (CPC_phase==running && FPC_phase==idle) )->
ES=on;SS=on;GPS=off;STR=off;RW=on;THR=off;PLI=off;CPC_phase=running;
FPC_phase=idle;s=19;printf("To the Nominal Mode\n\n");
printf("Error occurrs in the Controllers phase in Nominal Mode\n\n");
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,s)
:: else -> skip
fi;
:: mode==5 ->ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;PLI=Standby;
CPC_phase=running;FPC_phase=running;
if
::(CPC_phase==idle && FPC_phase==running) ->
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::((CPC_phase==running && FPC_phase==running) ||(CPC_phase==idle &&
FPC_phase==idle) || (CPC_phase==running && FPC_phase==idle))->
ES=off;SS=off;GPS=coarse;STR=on;RW=on;THR=on;PLI=off;CPC_phase=idle;
FPC_phase=running;s=20;printf("To the Preparation Mode\n\n");
printf("Error occurrs in the Controllers phase in Preparation Mode\n\n");
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,s)
:: else -> skip
fi;
:: mode==6 ->ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;PLI=Science;
CPC_phase=running;FPC_phase=running;
if
::(CPC_phase==idle && FPC_phase==running) ->
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,mode)
::((CPC_phase==running && FPC_phase==running) ||(CPC_phase==idle &&
FPC_phase==idle) || (CPC_phase==running && FPC_phase==idle))->
ES=off;SS=off;GPS=fine;STR=on;RW=on;THR=on;PLI=Standby;CPC_phase=idle;
FPC_phase=running;s=21;
printf("To the Science Mode\n\n");
printf("Error occurrs in the Controllers phase in Science Mode\n\n");
run mode_operation(ES,SS,GPS,STR,RW,THR,PLI,CPC_phase,FPC_phase,s)
:: else -> skip
fi;
:: else -> skip
fi } }
init /* init Process for Reconfiguration Check */
{int N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI; //Nominal branch of units
int R_ES,R_SS,R_GPS,R_STR,R_RW,R_THR,R_PLI;//Redundant branch of units
int CPC_phase,FPC_phase,mode;
#define off 0
#define on 1
#define coarse 2
#define fine 3
#define Standby 4
#define Science 5
#define idle 0
#define running 1
for (mode : 1..6)
{if
:: mode==1 ->N_ES=off;N_SS=off;N_RW=off;N_GPS=off;N_STR=off;
N_THR=off;N_PLI=off;CPC_phase=idle;FPC_phase=idle;
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
:: mode==2 ->N_ES=off;N_SS=off;N_RW=off;N_GPS=off;N_STR=off;
N_THR=off;N_PLI=off;CPC_phase=idle;FPC_phase=idle;
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
:: mode==3 ->N_ES=on;N_SS=off;N_RW=on;N_GPS=off;N_STR=off;N_THR=off;
N_PLI=off;R_ES=on;R_SS=on;R_GPS=off;R_STR=off;R_RW=on;R_THR=off;
R_PLI=off;CPC_phase=running;FPC_phase=idle;
if
::(N_ES==on && N_SS==on && N_RW==on && N_GPS==off && N_STR==off &&
N_THR==off && N_PLI==off) ->
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
::((N_ES!=on || N_SS!=on || N_RW!=on) && R_ES==on && R_SS==on &&
R_RW==on ) ->printf("To the Safe Mode\n\n");
printf("Error occurrs in nominal branch of Unit\n");
printf("Please wait! Unit Reconfigured\n");
N_ES=R_ES;N_SS=R_SS;N_RW=R_RW;
printf("The redundant branch is taken into use\n\n");printf("Again ");
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
::((N_GPS!=off || N_STR!=off || N_THR!=off || N_PLI!=off) && R_GPS==off
&& R_STR==off && R_THR==off && R_PLI==off) ->
printf("To the Safe Mode");
printf("Error occurrs in nominal unit branch");
printf("Please wait! Unit Reconfigured\n");
N_GPS=R_GPS;N_STR=R_STR;N_THR=R_THR;N_PLI=R_PLI;
printf("The redundant branch is taken into use\n\n");
printf("Again ");
```

```
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
:: else -> skip
fi;
:: mode==4 ->N_ES=off;N_SS=off;N_RW=off;N_GPS=coarse;N_STR=on;
N_THR=on;N_PLI=off;R_ES=off;R_SS=off;R_GPS=coarse;R_STR=on;R_RW=on;
R_THR=on;R_PLI=off;CPC_phase=idle;FPC_phase=running;
if
::(N_ES==off && N_SS==off && N_RW==on && N_GPS==coarse && N_STR==on &&
N_THR==on && N_PLI==off) ->
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
::((N_ES!=off || N_SS!=off || N_RW!=on) && R_ES==off && R_SS==off &&
R_RW==on ) ->printf("To the Nominal Mode\n\n");
printf("Error occurrs in nominal branch of Unit\n");
printf("Please wait! Unit Reconfigured\n");
N_ES=R_ES;N_SS=R_SS;N_RW=R_RW;
printf("The redundant branch is taken into use\n\n");printf("Again ");
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
::((N_GPS!=coarse || N_STR!=on || N_THR!=on || N_PLI!=off) &&
R_GPS==coarse && R_STR==on && R_THR==on && R_PLI==off) ->
printf("To the Nominal Mode\n\n");
printf("Error occurrs in nominal branch of Unit\n");
printf("Please wait! Unit Reconfigured\n");
N_GPS=R_GPS;N_STR=R_STR;N_THR=R_THR;N_PLI=R_PLI;
printf("The redundant branch is taken into use\n\n");
printf("Again ");
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
:: else -> skip
fi;
:: mode==5 ->N_ES=on;N_SS=off;N_RW=on;N_GPS=fine;N_STR=on;N_THR=on;
N_PLI=Standby;
R_ES=off;R_SS=off;R_GPS=fine;R_STR=on;R_RW=on;R_THR=on;R_PLI=Standby;
CPC_phase=idle;FPC_phase=running;
if
::(N_ES==off && N_SS==off && N_RW==on && N_GPS==fine && N_STR==on &&
N_THR==on && N_PLI==Standby) ->
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
::((N_ES!=off || N_SS!=off || N_RW!=on) && R_ES==off && R_SS==off &&
R_RW==on ) ->printf("To the Preparation Mode\n\n");
```

```
printf("Error occurrs in nominal branch of Unit\n");
printf("Please wait! Unit Reconfigured\n");
N_ES=R_ES;N_SS=R_SS;N_RW=R_RW;
printf("The redundant branch is taken into use\n\n");printf("Again ");
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
::((N_GPS!=fine || N_STR!=on || N_THR!=on || N_PLI!=Standby) &&
R_GPS==fine && R_STR==on && R_THR==on && R_PLI==Standby) ->
printf("To the Preparation Mode\n\n");
printf("Error occurrs in nominal branch of Unit\n");
printf("Please wait! Unit Reconfigured\n");
N_GPS=R_GPS;N_STR=R_STR;N_THR=R_THR;N_PLI=R_PLI;
printf("The redundant branch is taken into use\n\n");printf("Again ");
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
:: else -> skip
fi;
:: mode==6 ->N_ES=off;N_SS=off;N_RW=on;N_GPS=fine;N_STR=off;
N_THR=on;N_PLI=Science;
R_ES=off;R_SS=off;R_GPS=fine;R_STR=on;R_RW=on;R_THR=on;R_PLI=Science;
CPC_phase=idle;FPC_phase=running;
if
::(N_ES==off && N_SS==off && N_RW==on && N_GPS==fine && N_STR==on &&
N_THR==on && N_PLI==Science) ->
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
::((N_ES!=off || N_SS!=off || N_RW!=on) && R_ES==off && R_SS==off &&
R_RW==on ) ->
printf("To the Science Mode\n\n");
printf("Error occurrs in nominal branch of Unit\n");
printf("Please wait! Unit Reconfigured\n");
N_ES=R_ES;N_SS=R_SS;N_RW=R_RW;
printf("The redundant branch is taken into use\n\n");printf("Again ");
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
::((N_GPS!=fine || N_STR!=on || N_THR!=on || N_PLI!=Science) &&
R_GPS==fine && R_STR==on && R_THR==on && R_PLI==Science) ->
printf("To the Science Mode\n\n");
printf("Error occurrs in nominal branch of Unit\n");
printf("Please wait! Unit Reconfigured\n");
N_GPS=R_GPS;N_STR=R_STR;N_THR=R_THR;N_PLI=R_PLI;
printf("The redundant branch is taken into use\n\n");printf("Again ");
run mode_operation(N_ES,N_SS,N_GPS,N_STR,N_RW,N_THR,N_PLI,CPC_phase,
FPC_phase,mode)
:: else -> skip
fi;
:: else -> skip
fi } }
```

ISBN 978-952-12-2650-2
ISSN 1239-1891