Mission Critical Software Test Philosophy A SILS based approach In Indian Mars Orbiter Mission

Sumith Shankar S, Kiran Desai, Shuvo Dutta, Rajiv R Chetwani, M. Ravindra and K.M.Bharadwaj
Software Quality Assurance Division – Systems Reliability Group
ISRO Satellite Centre (ISAC)
Bangalore 560017, India
sumithss, kirandes, shuvo, , rajiv , ravix, kmb @isac.gov.in

***Abstract*— ISRO Satellite Centre (ISAC) is the lead centre of the Indian Space Research Organisation in the development and operationalisation of satellites for communication, navigation and remote sensing applications. In all these spacecrafts, highly advanced embedded systems carryout variety of mission critical functions. Each of these embedded systems house heterogeneous Processor and Embedded Software Combinations to carry out their functionalities. As per existing practices, testing of on board software to confirm its functioning takes place only when the software is integrated with its associated hardware. On the contrary, by a technique called the Software in Loop Simulation (SILS) test method, the on-board software can be fully tested in a software simulated dynamic environment without Hardware. This method of flight software validation is demonstrated in MARS ORBITER MISSION for AOCE, TCP, SSR, BDH software. The results very well demonstrate the effectiveness of the technique in early performance prediction and assessment of on-board software. This validation philosophy will be followed for all future spacecrafts. In a development environment where software requirements are too complex and requirement changes are to be incorporated even during final stages of development, this technique offers an excellent solution in fully validating on board software at source code level before it gets integrated with target hardware. This additional validation step not only improves software quality but also enhances productivity and reduces system turnaround time.**

Keywords—Automation, verification, validation, simulator, Software In loop simulation.

I. INTRODUCTION

ISRO Satellite Centre (ISAC) of Indian Space Research Organisation is responsible for developing Satellites for variety of applications such as Remote sensing, Communication and Scientific missions. A wide variety of mission critical functions of these spacecrafts are carried out by highly advanced processor based embedded systems. Based on System engineering life cycle followed as in Fig. 1, once the system requirements are finalised, the Hardware and Software development activities commence.

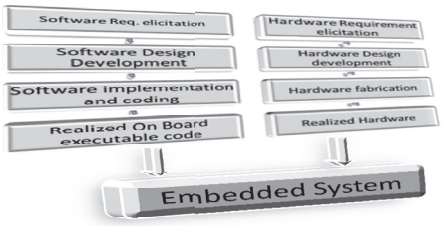

Fig. 1. Present Philosophy

Practically, it was observed that in this development time frame, software will be available much before hardware. Exhaustive testing of On –board Software was carried out only after its Integration with On Board Electronics at package level. This testing stage is referred as "Integrated Bench level test" (IBT) of On Board Computer.

Schedule impact due to dependency of hardware availability for software clearance was a serious concern to SQA engineers always. This became much prominent in MOM due to "opportunity based mission" management requirements. By introducing SILS inline to SDLC SQA could nullify the difficulties imposed by management.

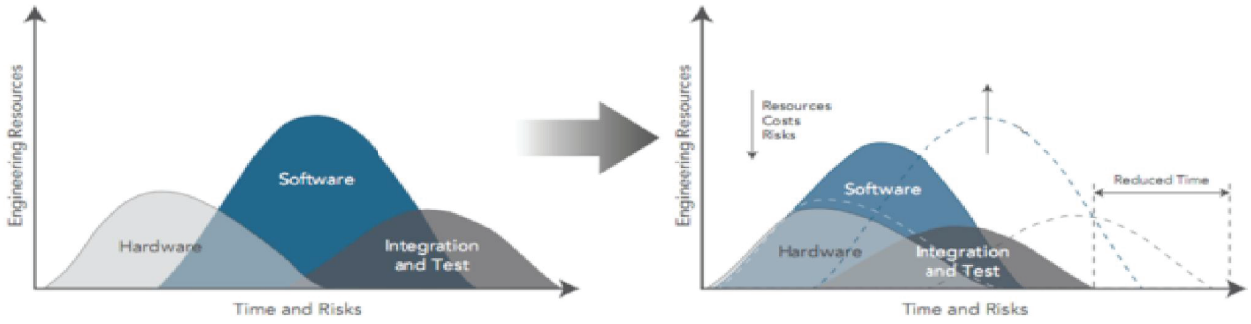

Fig.2 .

As shown in fig2 SILS could make a phase shift in traditional process sequence by co-extending software validation activities along with hardware realization time. Apart from process improvement achieved on turnaround time this paper gives insight to advantage gained over traditional validation approach supplemented with generalized SILS architecture and customization of the same for MOM on-board software.

II. Traditional Validation Strategy

Following are the set of V & V process steps carried out once source code is made available to Software QA team:

- Unit level testing
- Code Walkthrough
- Integration testing
- Initial bench level testing (IBT)
- Database verification.
- Hardware In Loop simulation
- Software PROM clearance.

Validation of software developed for spacecraft starts with unit level testing and ends with system testing. Presently software QA will audit the unit test results and directly participate in Initial bench level testing. QA will be invited for testing in IBT test setup once after designer gains sufficient confidence on the code developed. IBT test setup contains software integrated with flight / engineering model hardware. As far as SQA is concerned, IBT is the phase to exercise all those test cases generated to validate Integrated Software. Even though IBT is the key phase test case generation is the most intelligent work a QA person has to do without flaws.

Time management and forensic precision in finding bugs in such an environment, extremely depend on experience of SQA engineers. Various checklists and guidelines are generated based on the experience gained to minimize human dependent factors. Process optimization by identifying redundant activities and utility software development were the areas of focus to SQA engineers to achieve higher throughput. For example if we take CWT , strategies followed are;

- QA will perform complete CWT for the New and modified modules.
- CWT can be outsourced once it becomes routine.
- Audit CWT results by designers if the subsystem belongs to the class Non Operational Small satellite".
- Utility programs to check safe set violations etc.
- Similarly in case of testing strategies followed are
- Shift based test planning and execution.
- Verification utility program for test results
- from different test stations in different format.
- On-board software mimic model development etc

Till today strategies followed had found success in reaching towards zero defect and timely delivery.

But the following major limitations in technical and managerial aspects remained as a question mark on ISAC quality policy "Committed to Total Quality and Zero Defect in Space Systems and Services."

A. Technical limitations:

- Manual Coverage analysis
  - Statement coverage
  - Path coverage
  - MC/DC coverage
- Manual debugging with the help of logical analyser by probing pins in package.
- Lack of setup to probe software internals.
- Hardware imposed safety constraints for measuring and simulations.

B. Management limitations:

- Schedule issues
  - As there is a single FLIGHT Hardware, Test Engineers are driven to use the same Test Facility in shifts, increasing Testing time and thus delaying the Product delivery.
- Manpower issues
  - More manpower involvement in testing rather than creating test cases.
- Cost issues
  - Cost of replicating test stations
  - Cost of hardware anomalies caused by wrong/negative testing

Above mentioned constrains ultimately made SQA to develop a test bed platform for extreme testing and simulation by reducing human intervention as much as possible using Software In-loop simulation technique.

III. SILS A Concept

Software In Loop Simulation is a concept which is driven by simulated Microprocessor or Micro controller with non instrumented object code as input. Following are the major components of SILS architecture:

- A library of in house developed microprocessor or microcontroller simulators.
- A library of Subsystem Hardware - Software
- interaction simulators.
- Subsystem based generalized auto test case script
- language.
- User Interface (Command Line Interface or
- Graphical User Interface)

Layers of SILS is shown in Fig 3

A. Layer1: On Board Software Source Code -

This artifact is considered to be the starting point or prerequisite for SILS testing. The On Board Software Source Code developed can be High level language like Ada83, C, C++. On other hand there are subsystems which use assembly level language such as 8086, 8051 assembly language. The programming is done in host PC/workstation.

B. Layer2: Cross Compiler/ Target Assembler suite

In case of High level Source Code made available, "Standard Validated Cross Compilers" compile and link the On board Source Code and generate executable object code.

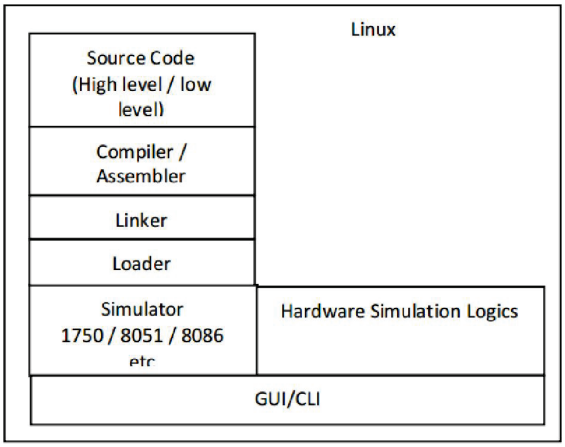

Fig. 3. SILS Layers

C. Layer3: Loader

Loader is a program or utility software which loads the program to be executed on to the Target platform or Simulated Target Platform (in this case). The Loader requires the executable code to be in one of the following formats. The utility Loader program is developed to support the following executable file formats

- Extended Tektronix Hex Format.
- Intel Hex 32 bit.
- Intel Hex 16 bit.
- Binary (bin) object file.

D. Layer4: Processor Simulator

The nucleus of the entire SILS architecture is the Processor simulator. This part has the simulated Processor architecture. The following are some of the important components simulated by the SILS

- Instruction Set Architecture with Timing specification
- Interrupt Servicing Scheme
- Timer Units
- Memory Management Unit
- RAM, PROM Simulation
- Loader Program
- I/O Management etc

As shown in the above fig.3 heterogeneous processors are used by ISAC for which Simulators are developed – MA31750, 8086 and 8051.

E. Layer5: Hardware Simulator Logics

A Generic Electronic Package Simulation also referred to as IO synchronizer is developed which acts as an interface and takes care of all the data transfer to and from the Processor simulator core. This Hardware Simulator in its broad continuum simulates all the Electronic Cards and its associated Control signals and memory requirements (if any). In general, across various subsystems features developed and configured are

- Data Acquisition Simulators
- Telecommand and Telemetry Simulators
- IO Port Processors
- EEPROM Simulation
- Power Electronics Simulation
- Communication protocol simulation etc...

IO synchronizer is designed by set of configurable data pools and process pools

1) Data pool :- Consists of a set of {Shared Memory, Semaphore} pairs with each one identified with a unique key. Used to mimic all analog and digital hardware interfaces.

2) Process pool:- It is a collection of threads that can be configured based of project requirements. The threads are mainly 1553 BC thread, Port IO thread, memory IO thread, register IO thread, Telemetry acquisition thread, telecommand dispatch thread, sensor simulation thread and actuator simulation thread.

F. Layer6: User Interface

A variety of options were explored for GUI design. Both GUI and CLI options were explored and used. But to begin with each Subsystem had its own GUI/CLI. Later a Generalized GUI which could interact with any Subsystem Software was developed and interfaced across multiple subsystems. This GUI was developed with QT4 Development toolkit. GUI is realized with widgets based working themes.

IV. MARS ORBITER MISSION (MOM) PROJECT

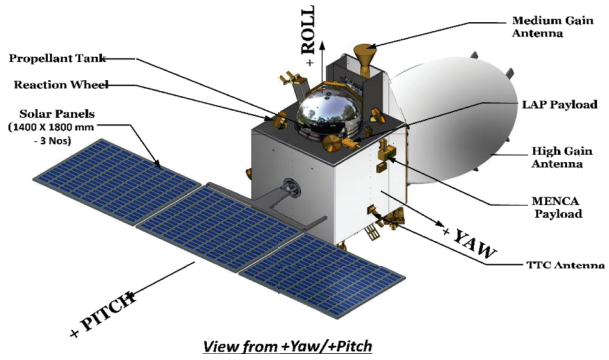

Fig. 4. MOM spacecraft (panel deployed View)

One of the main objectives of the Indian mission to Mars is to develop the technologies required for design, planning, management and operations of an interplanetary mission comprising the following major tasks.

- Orbit manoeuvres to transfer the s/c from Earth-centered orbit to heliocentric trajectory and finally capture into Martian orbit
- Development of force models and algorithms for orbit and attitude computations and analyses
- Navigation in all phases
- Maintain the S/C in all phases of the Mission meeting Power, Communications, Thermal and Payload operation requirements

- Incorporate autonomous features to handle contingency situations

Above mentioned tasks increased complexity in software requirements. Along with the difficulty in micromanaging the satellite called for building mission specific autonomy features in MOM spacecraft. Prime subsystems selected to implement this features are AOCE and TCP.

V. AOCE SILS TEST FACILTY ARCHITECTURE : A CASE STUDY

A. ADA TLD compiler suite on layer1 to layer 3

The AOCE On Board Software is implemented using Ada'83 Programming Language. This Source Code is compiled using TLD Ada Cross Compiler for Target system MA31750. The Compiled Executable object Code in the TEKHEX or Binary format is made available for Loader to load into the simulator. SILS is designed on layered architecture approach/

B. MA31750 Simulator on layer 4

MA31750 is the core layer of the framework. This Core layer is designed to have prominent and most exhaustive features to run On Board software. The following are the Components designed to have the following components;

- Configuration Register module
- Instruction Set Architecture module
- Input Output Port controller module
- Interrupt and Fault Handling module
- Timer modules
- Addressing mode modules
- RAM, PROM and other memory simulation
- Memory Management unit – MA31751
- Loader module – accepts binary and TEKHEX file formats.
- It is in this module that the AOCE Flight
- Software image is loaded for execution.

C. MOM specific hardware simulation logics at layer 5

1) Data pool configurations:

TABLE I. IO INTERFACE

Interface	Type	No of ports configured
Tacho	Digital	16
Sun Sensors	Analog	10
Solar Panel Sun Sensors	Analog	2
FDIR signals	Digital	4
Solar Panel Drive	Digital	2
EEPROM Signals	Digital	4

a. *Memory Interface*- It is used to Log RAM and PROM memory details of MA31750 and its associated memory peripherals

b. *Execution Log Interface*-It is use to Log the Instruction execution and Addresses accessed details. This will be used to generate path coverage information by mapping the address with Routines and Flight source code statements using Linker file.

c. *1553 BC-RT Interface*-This shared memory and semaphore pair is used to establish communication between 1553 BC simulator and 1553 RT Interfaces simulator. In MOM subsystems configured as RT are

- Star Sensors
- Gyro
- Accelerometer
- Telecommand interface
- Telemetry Interface and
- FDIR signals

2) Process Pool configurations: 1553 BC thread:-Bus controller requirements in MOM are for 111 messages aiding a combination of message transaction between 12 Remote terminal message transactions are configured.

a) *Port IO Thread:* According to data pool configurations this thread is configured

b) *Memory IO Thread:*- In unit level testing and integrated level this is used extensively to force software in defined states. The output of the same is in turn captured at GUI level to generate pass fail criteria. This helped in automating regression test result verification. Memory IO thread is always active in background unless not deactivated.

c) *Register IO Thread:* An always active thread helped in carrying out following analysis

TABLE II.

Activity	Remark
Statement coverage	100%
Dead Code analysis	All dead code identified are deleted
Decision coverage	100%
Timing Estimation	All time margins estimated matched with actual with resolution of 0.6ms

d) *Telecommand dispatch Thread:*-CCSDS based improvement in Telecommanding Scheme of MOM called for a change in conventional TC Despatch Thread Design. This new design has been used to send more than 10000 telecommands while testing.

All the necessary hardware parameters like;

- New Command arrival indication
- Command Execution status
- Valid command indication
- Command counter
- Checksum of the command etc

Are Simulated by the TC Simulation

e) *Telemetry acquisition thread:*-1553 based TM Scheme in MOM has called for a change in Conventional RMU based TM Acquisition Thread. The OnBoard House Keeping Data recorded by the AOCE Subsystem is dispatched as raw data to the GUI through LINUX IPC Mechanism. The GUI further decodes the raw data and displays the processed data.

f) simulation thread:-The On board Software requires data and control signals from Sensors to perform its controller operations. To aid this functionality, the Sensor stimuli are simulated for Analog Digital sensors and 1553interface data according to MOM Electrical Interface Definitions. This includes FDIR signal simulations required for Spacecraft Autonomy Testing. Whenever the user sets the data values through GUI/Scripts for sensors during testing, these data values along with control signals after conversion from Engineering units to Digital units is transmitted to On Board Software for further execution. The following Sensors are simulated as part of the MOM On Board Test facility;

- Star Sensor Head simulation
- Gyro Simulation
- Accelerometer simulation
- Tacho Simulation
- Battery simulation
- Sun Sensor Simulation
- Solar Panel Sun Vector Simulation

g) *Actuator Simulation thread:* Actuator signals of Control system are generated by AOCE Flight Software. These signals are processed by Actuator simulation thread. After processing the data/signals routed towards the GUI for suitable display. The parameters displayed consist of data such as;

- Thrusters Duty Cycle
- Wheel Torque Processing and Tacho Current Signal
- SPDM Motor Drive simulation

D. Graphical User Interface:- A Generalized approach

Developed to cater to all Subsystems like AOCE, TCP, BDH and SSR. The GUI communicates with IO Synchronizer through Shared Memory and Semaphores. In a broad spectrum it implements the following functionalities;

- Test Script Editing, Conversion And
- Execution
- TC Transmission
- Sensor Settings
- Actuator Data Acquisition
- TM Acquisition and Processing

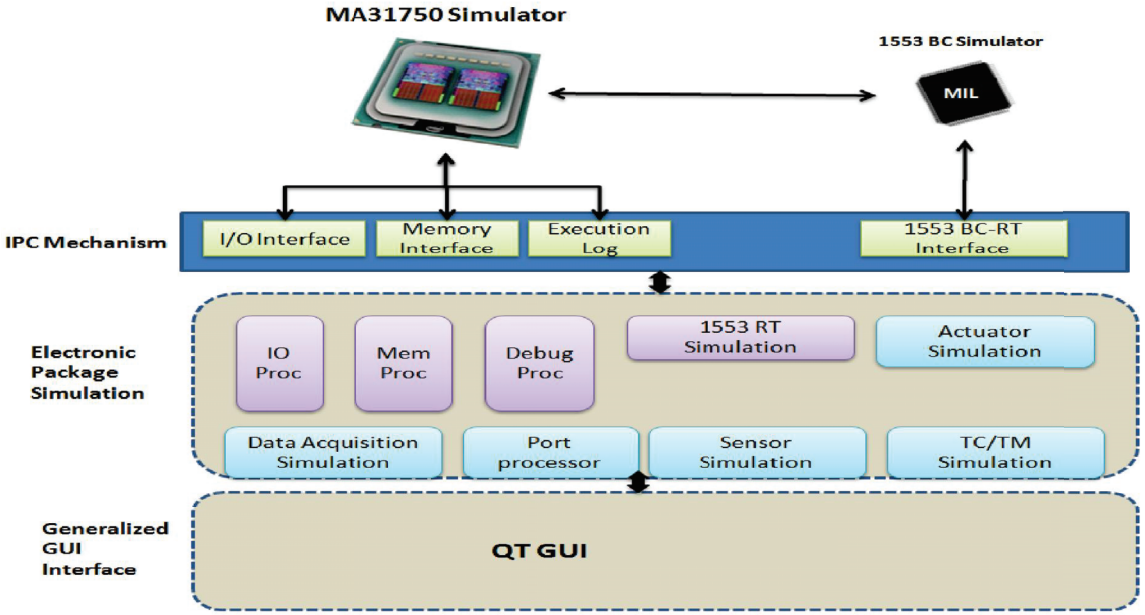

Fig.6. SILS mapping for MOM AOCE subsystem

- SILS CLI Extensions
- Configuration made easy
- Real Time Graph
- Offline Display

GUI is an optimized to achieve fast data rate monitoring.

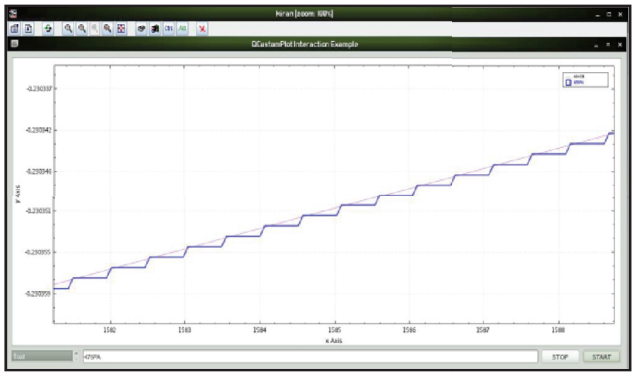

Fig 6. Sample plot

This help in to have a quick view of orbit profile . Compared to real time telemetry rate this 16 times faster. Fig. 6 is a sample plot generated in the GUI for MOM position information

Visual path coverage analysis is another highlight of GUI. A sample of visual path coverage for test case executed is shown in Fig7. The GUI capture the Instruction execution and Addressing details provided by IO Synchronizer through Generic interface and provide a visual display of one of the MOM On Board Software modules.

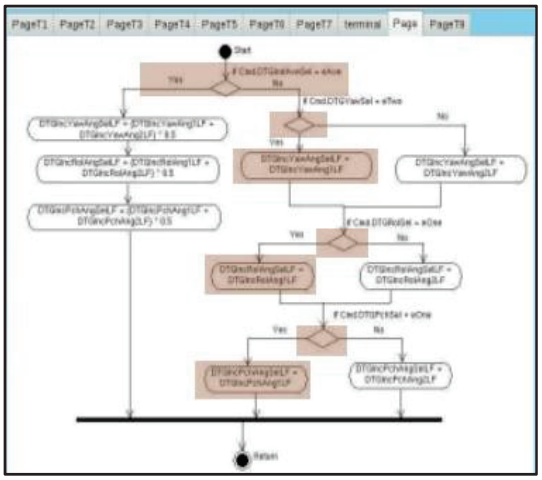

Fig.7 Path Coverage Analysis

VI. CONCLUSION

Implementation details of the technique along with the effectiveness of this test methodology in bringing out residual defects in software are highlighted in the paper. Moreover it clearly brings out the benefits of the test bench whereby testing of software in both open loop and closed loop forms are possible. A comparison with hardware-in-the-loop-tests shows how the software-in -the-loop-method is really capable of automating and comprehensively testing embedded systems early in the development cycle. The merits of the system in systematically eliminating possible malfunctions are worth mention. In a development environment where software requirements are too complex and frequent modifications are called for, this technique offers an excellent solution for developing quality software. Application of above explained SILS Technique has given flurry of Advantages and win over some of limitations faced earlier;

- Online debugging of On board software is made easier and user friendly.
- SILS will provide Faster than Real Time Testing Environment on a Workstation. It is normally observed that in worst case, simulators run from 5x-6x that of real time.
- All developers and T & E engineers have their software available on their Desktop Workstations.
- Simulation based Workstations provide better visibility into the high fidelity Software models as compared to when running on the Real Time Hardware devices.
- Since, C programming Language has been selected as the middle layer; Test Bench APIs can be linked with any of the Open Source analysis tools.
- Through the Fault injection on SILS, more extensive and detail test for FDIR (Fault Detection, Isolation and Recovery) capability is possible.

VII. ACKNOWLEDGMENT

Authors deeply acknowledge the support and encouragement extended by colleagues of SQAD and SRG for their help in the endeavor.

VIII. REFERENCES

[1] W Richard Stevens, Advanced *Programming in Unix Environment,* Addison – Wesley Professional Computing Series, Second Edition.

[2] M Beck, H Bohme, M Dziadzka, U Kunitz, R Magnus, D Verworner; Linux Kernel Internals; Addison Wesley Publications; Second Edition.

[3] James R Wertz, *Spacecraft Attitude Determination and Control***,** Kluwer Academic Publishers, Third Edition.

[4] Marcel J Sidi, Spacecraft Dynamics and Control: A Practical Engineering Approach, Cambridge University Press, 1997 Edition.

[5] Brian W. Kernighan and Dennis *M.* Ritchie, *The C Programming Language,* Prentice Hall Publications, Second Edition.

[6] David E Simon, *An Embedded Software Primer*, Pearson Education, 2007 Edition.

[7] *Enhanced Sμmmit Family Product Handbook*, UTMC Micro – electronic systems Inc., October 1999.

[8] *MA31750, MA31751 Datasheet, Dynex Semiconductors, January 2000 version.*