A Novel Approach

for

SLM GUI Design, Development

&

Test Suite Automation

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Information Security

Submitted By
Silvi Goel
(801333025)

Under the supervision of:
Dr. (Mrs.) Rinkle Rani
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July, 2015

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*A Novel Approach for SLM GUI Design, Development & Test Suite Automation*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Information Security* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Rinkle Rani,* refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Silvi Goel)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

02/07/15

(Dr. Rinkle Rani)
Assistant Professor
Computer Science and
Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. S. Bhatia)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose constant guidance and encouragement secured the success. First of all I wish to acknowledge the benevolence of omnipotent God who gave me strength and courage to overcome all obstacles and showed me the silver lining in the dark clouds. With the profound sense of gratitude and heartiest regards, I express my sincere feelings of indebtedness to my guide, **Dr. Rinkle Rani**, Assistant Professor, CSED, Thapar University, Patiala, for her positive attitude, excellent guidance, constant encouragement, keen interest, invaluable cooperation, generous attitude and above all, her blessings. She has been a source of inspiration for me. I would also like to thank **Mr. Kamlesh Kumar Pathak,** Senior Manager, SPG for allowing me to work as a part of their team. I want to express my sincere gratitude to my mentor **Mr. Krishna Kumar,** Technical Lead, SPG for all his guidance and time despite his busy schedule. He was with me throughout the term of the project and rendered me all the help required. My gratitude also extends to all those people, including team SPG, who met me during this endeavor and enriched me with knowledge and gave a helping hand for the project. Without their abiding inspiration, generous guidance and encouragement, I would not have been able to cope with project work and corporate environment.

I am grateful to **Dr. Deepak Garg**, Head of Department, and **Ms. Jhilik Bhattacharya**, P.G. Coordinator, CSED, Thapar University, for the motivation for the completion of this thesis. I will be failing in my duty if I do not express my gratitude to **Dr. S. S. Bhatia**, Dean of Academic Affairs in the University, for making provisions of excellent infrastructure immensely useful for learners to equip themselves with latest in the field.

Last but not the least I would like to express my heartfelt thanks to my parents and my friends, who with their thought provoking views, veracity, and whole hearted cooperation helped me in doing the thesis.

Silvi Goel

(801333025)

Abstract

Recent times have seen a vast change in trends ranging from mode of imparting education to transforming the world into a global village through 'Digital Era'. In this course of changes, the size of electronic devices, is a key factor around which electronic industry is advancing. Starting from a room-size computer, we now hold a vision to develop the device that must occupy minimum space, maximum features with optimal utilization of power.

'System on Chip' is the technology that backs up the feasibility to turn the vision true. It involves a very complex level of integration of various parts of a system onto a single chip. SoCs are rapidly replacing CPUs in the electronics market, and the pace of this change demands the design cycle time to be even shorter. The thesis document discuss briefly this crucial issue and addresses the most time-consuming phase of SoC design, i.e., Verification. Since industry research is primarily need oriented, we, at STMicroelectronics, take up the opportunity to develop solutions for the sections that pose significant challenge to the business activities of the company.

We present, SLM, the memory model that improves the reliability, and reduces efforts to test the functional accuracy of SoC design involving memory components. In the thesis, the solution for automated testing of SLM and its user interface have been presented. Focus primarily lies in minimizing the effort to test the memory model which therefore will ensure proper support to SoC Verification Design. The portability of model is supported through three simulators- Incisiv by Cadence Design Systems, vcs by Synopsys and Questa SIM by Mentor Graphics. Currently, we have tested it with Incisive only. Some work on extending System Verilog support by SLM libraries, is also been discussed as SV is one of the trending verification languages in recent times.

Table of Contents

List Of Figures

List Of Tables

Chapter-1

Introduction

Evolution of ‘Information Age’ has contributed indispensably towards advancement of a number of disciplines. One such domain, where IT is doing a commendable job for researchers is Electronics. The rapid advancement in semiconductor processing technology, is a major causeof steep rise in complexity of circuit design. This complexity include various parameters, such as gate-density, operational features, and power consumption. Development of HDTV, 3G and 4G devices are clear indicators of the need to incorporate traditional processors, memories and peripherals on a single silicon. The technology that caters to this need very well, is termed ‘System On Chip’.

1.1 System-On-Chip Design

1.1.1 Technology and Trend

‘System-on-chip’ system is an integrated system that constitutes all the system components such as microprocessors, I/O systems, memories, buses system and other intellectual property components. This technology support systems, those can be deployed in an environment which is small, and require high speed and efficiency.Research agency In-Stat predicts robust market growth for System-on-Chips (SoCs),estimating that volumes will increase an average of 31% a year, reaching 1.3 billion unitsin 2004 [1]. The growing demand for powerful handheld devices and green data-centers will continue to influence the next generation SoC architectures. Fig. 1.1 shows a basic set of components usually incorporated in SoC system.

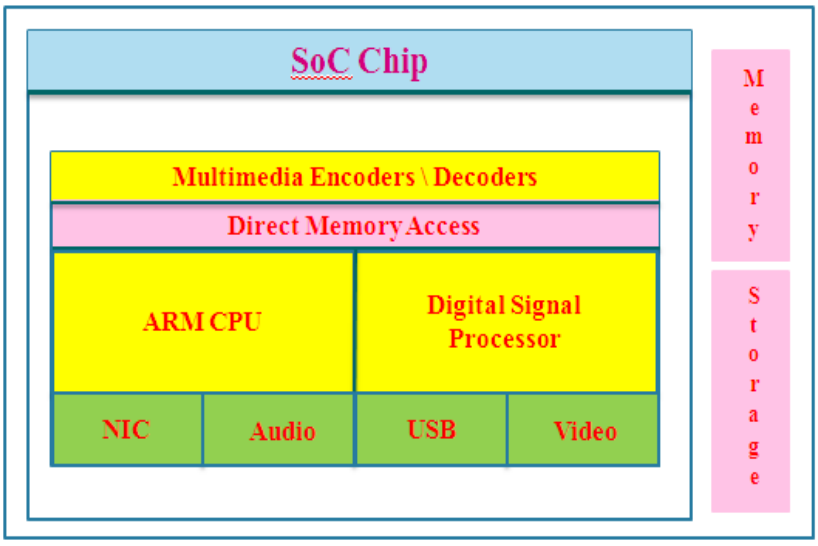

Fig. 1.1 Basic overview of components inside the SoC

1.1.2 Design Process

The basic design process of SoC system is divided into front-end design and back-end design.While front-end engineer creates a netlist with limited concept of physical world, the back-end engineer focus on flow creation strategy [1]. Overall cornerstones of SoC design process are described below :

a. **Architecture Design :** The architecture of processors and buses forming the interconnect for various IPs are a generic concern since they will ultimately control the efficiency of output from IP components. Depending upon the required processing capability expected, the system may need varying set of architectural configuration.

b. IP Selection : The IP components to be integrated on a system must be from trusted sources. Every IP must be properly validated through randomized tests.

c. Verification : It forms the most crucial phase of SoC Design. About 60% of SoC development time is spent on verification [1].

d. Integration : The key challenge in SoC Design is the interconnect design for various IPs installed on the chip. It is done with a certain set integretion test vectors which are genrally more directed in nature.

e. Validation : Validation of SoC system is a two level process. Firstly, it involves modular validation of each IP to be installed. Secondly, the integration should be thoroughly validated. The validation of IP and SoC differs in that, the test suite for IP generates randomized test cases, as against more directed test cases in the SoC validation test suite.

f. Physical Synthesis : It is the responsibility of back-end team to appropriately handle for physical factors becoming more prominent. Such factors may include Cross Talk, EMI effects, electromigration, 3D noise etc.

Since, the time-to-market is a major challenge in SoC manufacturing industry, the Hardware-Software Co-verification approach comes into play. In this methodology, the special

operating systems to be designed for the chips are simultaneously developed and tested against the software models of those chips. This results in release of product launch in lessertime frame. A diagrametic representation of the whole process and its utility is been depicted below :

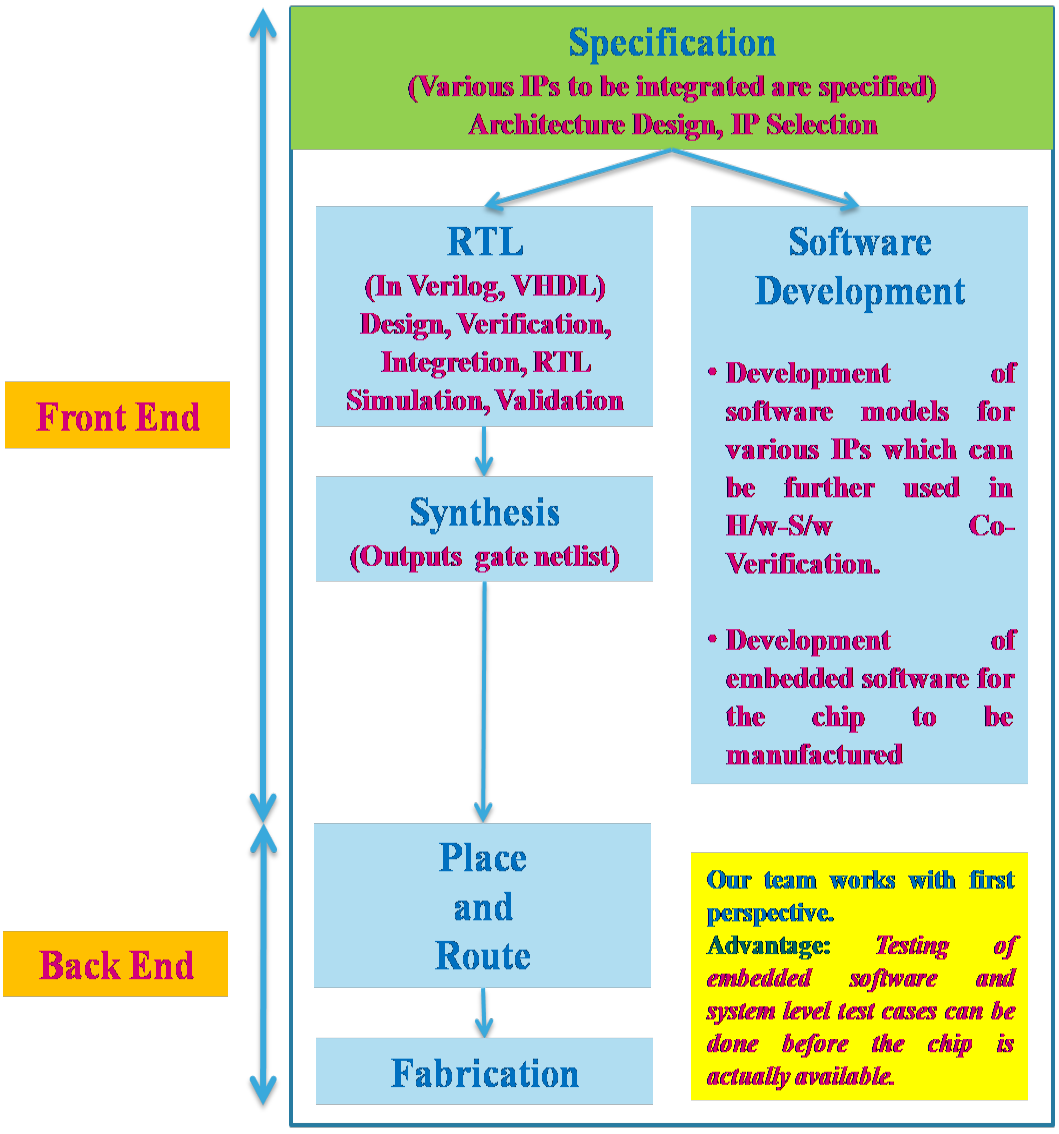

Fig. 1.2 SoC Design Process

As described in the Fig. 1.2, Software Development in parallel to RTl Design Development is beneficial in terms of overall design time. Software development is SoC Design is carried out with following perspectives :

- Developing software models for intellectual properties which can be further used in H/W-S/W Co-Verification.
- Developing embedded software for chip to be manufactured.

1.1.3 Opportunities and Challenges

Key Opportunities :

- Cost-effective.
- High level of IP integretion on a single piece of silicon.
- Significantly reduced overall design cycle time with retained/enhanced performance.
- Evolution of multi-core architecturesand network on chip result into complex designs for SoC. Appropriate software models can be advantageous to verify designs at early stages.

Key Challenges :

- Covering the design gap between increasing complexity of SoCs and way to verifying/validating it.
- Since complexity is increasing at rapid pace the manual efforts required are becoming a concern.

1.2 Role of Verification

In early times, the custom-crafted circuit designs were conveniently suited on the silicon area made available to designers. But, that changed altogether with need of complex chip design exhibiting an extreme level of integretion over the same silicon area being available. The growth of design reuse is resulting rise in the percentage ofsilicon occupied by IP blocks [2]. This trend significantly reduce complexity in design tasks but on the other hand raised concerns for functional verification of the system. As mentioned in[3], system-level verification can genuinely take up to 80% of overall design time.Since all the IP blocks must be integrated and verified within the context of each unique SoC, verification has become the dominant task in today's SoC and systems projects [2].Interconnection of various IPs on a single chip faces following challenges while verifying the system as a whole.

- Data, control and clock mismatch due to varying protocols of communication.
- Integrated and functionally verified sub-systems do not guarantee the same for complete system.
- System interconnect may be result into global execution order for processing unit or may be multi-threaded.
- Increased programmability of SoC result in large number of use cases to be verified.
- Design gap between evolving complexity of SoC and methods to validate them at an early stage.

After each step of refinement, the implementation is verified and validated. Verification refers to consistent operation at current level as validated at previous level of abstraction. Validation on the other hand tests for conformation of new development w.r.t the specification. Since, it is a crucial part of SoC design process, a lot of work is being carried to resolves the challenges mentioned above.

1.3 Our Team

Our team work with the development of software models for IPs, and specifically the thesis present work related to memory model. “System Design Solutions” is responsible for software modelling of different type of memories. These software models are then delivered to others teams within ST for incorporating with their simulators for verification purpose. Our team deals with one such memory model known as System Level Memory and its brief overview has been presented in the upcoming sections.

1.4 Complex Memory Operations

With a drastic change in computational and data miningtechniques,there is a significant change in the view for memory operations and CPU utilization. Development of GPUs, DMA Controllers and other dedicated processing units is a result of this change. The read/write memory operations in complex and computation intensive applications are a big overhead on the SoCs being developed for such applications. In-memory databases, memory scanners, and debuggers are few such applications which may violate TTM constraint, if relied on physical memory for verification. One such application where data processing capabilities are expected to be tackled by SoC design optimization and memory element being one of the crucial component is proposed in [4].

1.5 Verification Technologies in SoC Design

Commonly used verification techniques are:

1.5.1 Simulation Based : It can be transaction based, code coverage based, hardware-software co-verification, lint checking or other static technologies.

1.5.2 Formal Technologies : It can include theorem proving techniques, or formal model and equivalence checking techniques.

1.5.3 Physical Verification : It includes handling of various electrical issues such as IR drop, crosstalk, timing and signal integrity etc.

Our system, supports simulation based verification, with services related to ISS and Hardware-Software Co-Verification.

1.6 Verification Environment

Functional verification of a system is primarily to detect any bugs in the system before it is passed on to the consumer. It is not necessary that all bugs lead to failure but it is always an important metric for measuring quality of the system.In a true sense, it is a model of universe, as far as design is concerned [5]. Fig. 1.3 shows the components of a verification environment.

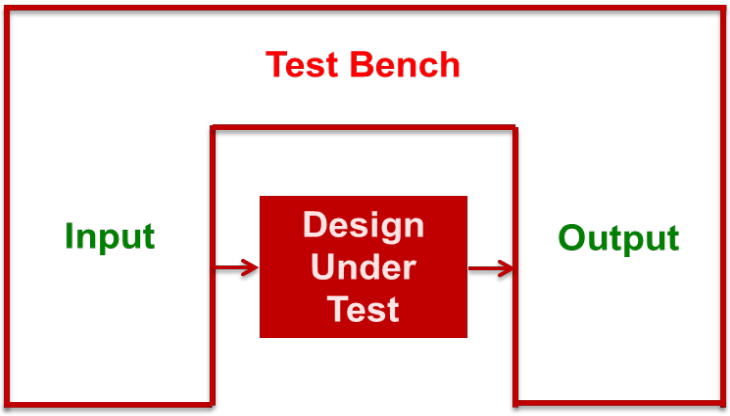

Fig. 1.3 Verification Environment

1.6.1 Key Terms

a. **Test Bench :** It is a virtual environment [6] that is used to verify the correctness or soundness of a design or model.

b. **Device Under Test :** DUT is the term commonly refers to the manufactured product under testing. It is also known as Unit Under Test or Equipment Under Test.

1.7 System Level Memory Model

SLM is proprietary software used at STMicroelectronics by verification teams to analyze various memory operations. SLM stands for 'System Level Memory'. It can be used to verify any functional design that require complex memory operations. It reduce the physical memory overhead by simply giving a 'C' memory view to the physical memory thereby enhancing performance by almost 50x to 1000x times. It allows to study the state transitions that a certain memory instance may undergo while being a part of a design simulated in HDLs such as Verilog and VHDL. SLM is a set of mixed HDL/C++ memory models, where

the models correspond to HDL functional architectures. Technically, replacement of memory array assignments with C/C++ subroutine calls turn HDL register/variable into ‘C Storage’.

1.7.1 What an SLM Model Looks Like ?

Fig. 1.4 shows a sample code which use SLM Services to register a memory instead of memory allocated through RTL.

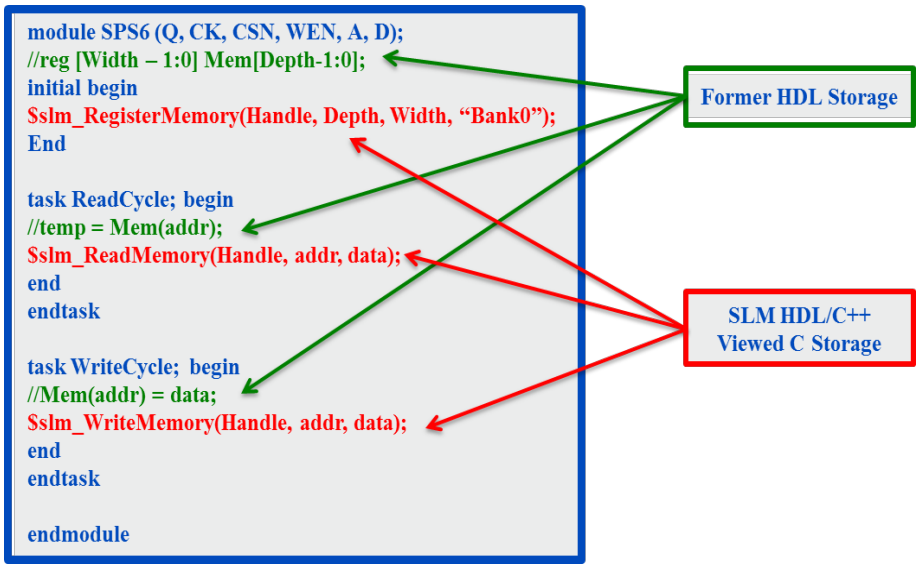

Fig. 1.4 Sample code depicting SLM Model

1.7.2 Calling C Procedures from HDL

Fig. 1.5 shows the invocation of C functions through HDL design, which is as simple as registering physical memory through HDL code.

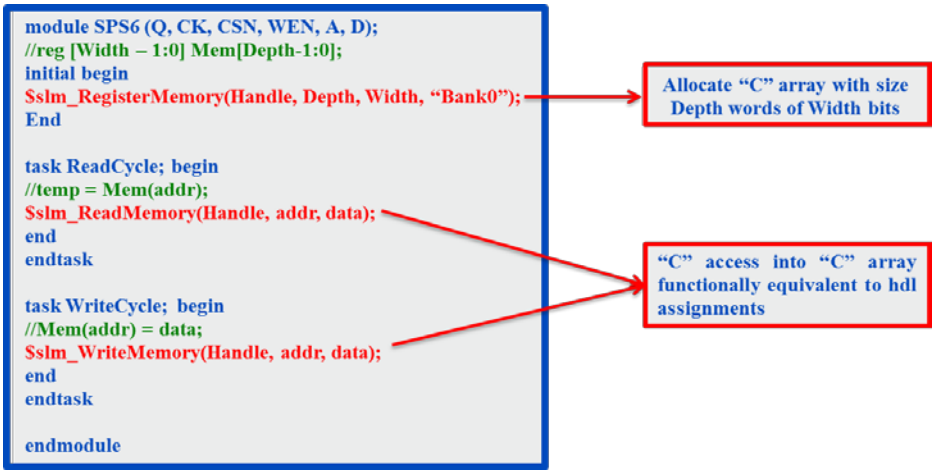

Fig. 1.5 C Procedure Calls from HDL

1.7.3 How It Works ?

SLM ‘C’ storage and services are delivered as pre-built ‘C’ dynamic libraries. In order to connect HDL logic together with the C array and related access functions, the simulator specific HDL/C interface are used.

1.7.4 SLM Advantages

- A set of verification, debug and recording services.
- A Memory viewer.
- Capability to record everything that actually happens on each SLM instance.
- Memory content is now in C/C++ which is a high level language that leverages the connection with 3d party tools: analysis tools, viewers...
- Memory content can be manipulated using a C interface, thus bypassing the standard HDL way called ‘Backdoor Access Capability’**.**
- SLM infrastructure can be used in a pure C/C++ environment (namely SystemC 2.0.1 TLM platforms)
- SLM models can expose their back-door interface to other C models that are involved in the HDL simulation (like Instruction Set Simulators ~ ISS)
- Common added-value services can be implemented in the C part and reused in different models as black-box “legacy code” (i.e. Fault injection, BIST testing)
- Same debug and verification services can be reused in different abstraction levels
  - HDL simulations
  - SystemC TLM simulations
- For different use
  - Verification&Debug
  - HW/SW Co-Verification
  - Memory Subsystem Analysis

1.7.5 Platforms, Simulators and Languages

It is well supported by Linux Operating System. In term of supporting simulators and languages the following depict a wide range of platforms that SLM is callable from. Fig. 1.6 shows the architecture for multi-platform support by SLM.

sin

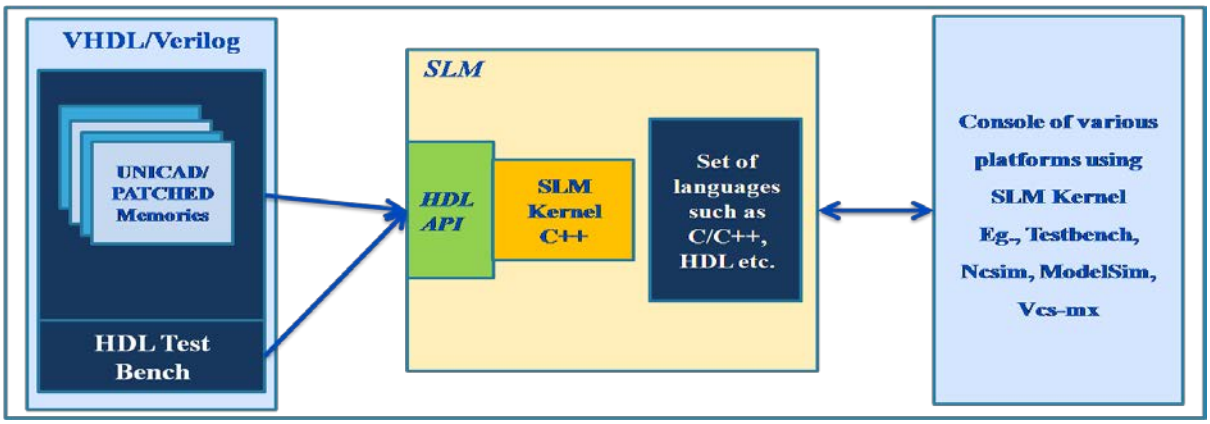

Fig. 1.6 Usability of SLM model with various simulators

Table 1.1 presents the list of SLM supporting simulators and designs they use to communicate with SLM.

Table 1.1 Supported simulators and languages

Simulator	SLM Memory model	Design
Cadence Ncsim	Mixed HDL	Mixed HDL
Mentor Modelsim	Mixed HDL	Mixed HDL
Synopsys VCS-MX	Mixed HDL	Mixed HDL

1.7.6 Need of SLM

- Calling SLM services is relevant to be used when any of enhanced debug and verification features are to be exercised.
- SLM services rely on the “C” storage of SLM memories.
- Services can be invoked from the beginning of the simulation because the “C” storage is registered before the start of the simulation.
- All services are 0-time consuming with respect to the logic simulation.

1.7.7 Applications

As depicted in Fig. 1.6, it is callable from a number of platforms such as ncsim, modelsim, vcs-mx console, C/C++ etc.This enhances its usability for a number of users including :

a. SLM Model providers. It include the internal services in the model which is not accessible by general users.
b. Verification, Integration and Design Engineers.
c. Users of external-services like memory sub-system analysis, standard functions like verification, debugging and recording etc., provided by SLM.

1.7.8 SLM Service Categories

a. Internal Services

They are used by model developers and are not visible to the users operating the model. Few internal services are :

- Registering the memory instances.
- Read and Write Access.
- Loading the memory instance.
- Reset the memory state.
- System memory based operations.

b. External Services

They are accessible to users operating the model, thereby, achieve easier interaction between user and software. Various external services can be categorized as :

- Utility Services
- Verification, Recording and Debug Services
- System Services
- HW/SW Co-Verification Services

Chapter-2

Literature Review

2.1 GUI Design &Test Automation

2.1.1 Recent Work

Number of approaches have been worked upon to automate the GUI test process, since it is one of the most robust solutions to pace-up the testing problem [7]. G. M. D. Gandhi et al. [7] proposed a solution, termed GUIRobo, a stress-testing based tool, that overcome a number of testability challenges in automation process for testing. But that may be subject to usability since the input file it takes as interface description, may miss out on dynamically changing interface.

An available choice to many testers, are test-script based automation and capture-replay methodology, which may seen viable solution for light-weight and simple GUIs, but as the complexity of GUI system scales up, they fail to meet the expectations. They require a lot of human intervention if the GUI is dynamically changing. Any state in the system, if required to be re-produced, is very-consuming and cumbersome.

Another alternative to these strategies is model based GUI testing, M. Blackburn, et al.[8] compared how model based testing can be a lot more efficient to handle the dynamic GUI systems. Model-based testing can also contribute to eliminating defects in requirement, and reduce typically manual design effort.

Two model-based test approaches that significantly provides for efficient coverage of GUI testing are Bayesian model based [9]and State-chart based [10] approaches. They offer viable alternatives to replace high overhead of script-based automation systems. Z. F. Yang et al. [9] very systemetically proposed the reliability of GUI as a measure of probabilities that covers both the structural and operational profile of the system. This creates a space for complex event interactions to be accomodated through ‘transition matrix’, which makes it very suitable for testing the GUI functionally as well as operationally, thereby giving solution for accuracy and reliability of system.

But the assumption it takes regarding freezed source code, again makes it non-adaptable to the dynamic perspective of interface layouts.

V. Santiago et.al. [10] proposed an efficient solution to present specification model in terms of Statechart diagrams, and further translating them to XML format. This can be effective in handling dynamic nature of layouts, and rich set of details can be acquired about complex systems. R. M. Sharma [11] presented a practical result in metric terms to show the difference automated testing can make. Z. Liu et.al. [12] proposed a systemetic approach to differentiate between script driven content, layout description and test inputs in data definition. H. Zhu et.al. [13] presented a compiled view of various aspects in software testing, which primarily include, security and test case generation. Z. Liu et.al. [14] presented a framework to support the concept of different definitions of test inputs, script driven variables and layout description. M. Sharma et.al. [15] described model based testing as an evolving technique for generating test cases from behavioral model of SUT.

2.1.2 Challenges

GUI testing can be manual or automated. Manual Testing is really a tedious task, that involve executing a number of test cases designed by engineer manually.It requires the tester to be creative, innovative, conjectural and open-minded. For manually testing to be effective, the tester must possess patience, good observations and skill at job.Despite being time consuming and effort demanding, reliability of manual testing [16], is questionable, when the complexity of prevailing projects in the domain, is observed. Thus, manual testing is not a viable alternative, when it cost too much without being reliable and may get error-prone.While manual testing have serious implications on system quality, automated testing offer a number of advantages. Automated Testing is faster and reliable as compared to manual option. It can facilitate improved regression testing of system, with considerable effort on one-time design of test suite automation [11]. Automating the GUI testing, if implemented properly, can save a lot of time, cost and effort.

Significant benefits of automated test suite are [7]:

A. Improved system quality and minimal testing effort.
B. Support for testing the GUI over a number of platforms.
C. Improved regression testing.
D. Repeated generation of certain states in system become significantly easy.
E. Significant contribution to prove the conclusions consistent with results, since replication of scenarios is no more cumbersome.

Number of challenges are inherent from the complexities in GUI design and test automation. Completeness, reliability and accuracy become the important parameters for software quality in such a case. Few constraints impact the choice very significantly.

- **Programming Language**

Programming language play a very important due to its dependency on compiler and third-party support. Different programming paradigms can make one approach easy in ccertain language and same approach extremely difficult in another. For instance, a good alternative for VB based GUI may not prove to be as good for Java based GUI. A change in language may simply need the developer to modify whole recording structure of test suite [11].

- **Platform**

The operating system come into play if GUI relates to some system application or if it is tightly coupled with the operating system.

- **Structural Profile**

Structural profile of GUI [9], constitutes the components planted on the interface and it becomes a concern when that is dynamically changing throughout the user interaction. In such case monitoring of changes and the complex event interactions may need to be published into reports for proper tracking.

Although a number of tools for automated testing are available, there is not much significance of them for many companies.It is so because many of them are based on test scripts which are either developed or they may have been created using recorded and replayed approach, which fails to counter the impact of changing layouts [7].

GUI Design had totally been worked upon w.r.t SLM structure. The inherent challenges in SLM Core were traced to plan some cornerstones and verticals for exploration. A major challenge was **‘A big ball of mud’ [17]**. Basically, we found the system as one of those whose structures are either missing right from the initial phases of development, or if they ever exist, might have got eroded with various factors. It reached a stage beyond repair, i.e., an un-maintainable solution, which can’t be further supported with new ideas to sustain new user requirements.

We found our system in a state as depicted by Fig. 2.1.

Fig. 2.1 Inherent problem traced from current system

So we had a responsibility while designing GUI, that we do not end up adding more muddy code segments to already existing jungle of codes. Thus, we planned certain cornerstones for the project.

- Should run in non-Co-Sim mode as well.
- Simple environment setup
- Seamless integretion of sub-systems
- Modular and Loosely coupled sub-systems
- Usage of Free tools
- Mechanism to self test the system

Keeping that in focus, we explored following verticals while designing GUI as described through Fig. 2.2.

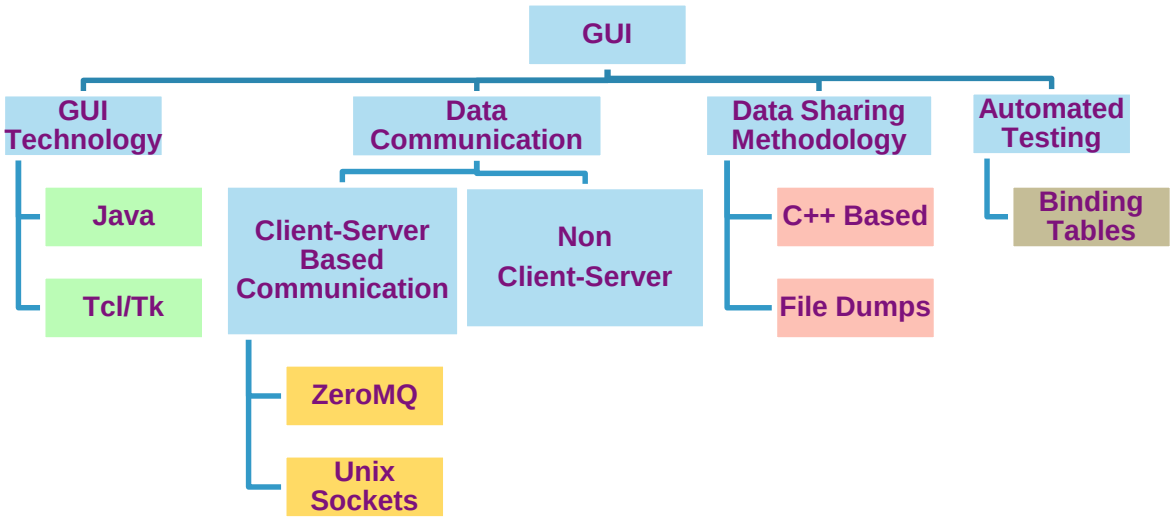

Fig. 2.2 Various verticals explored for SLM GUI Design

2.2 System Verilog Support

Presently, the system is based on mix C++/HDL models, where functions are developed in C++ and RTL design is Verilog and VHDL based. But Verilog and VHDL based designs are very complex and time consuming. A division between data-structures and code, makes it very difficult for Verilog users to understand the program functionality. System Verilog provides support for OOP features making RTL design much flexible and manageable withfeatures like inheritance, encapsulation, polymorphism, etc. Verilog and System Verilog both allow calling of C functions. While Verilog can do it using VPI or PLI, Systesm Verilog uses DPI for the same. A brief comparison of PLI, VPI and DPI is presented in the following section :

PLI offers two kind of libraries : TF and ACC Library. TF library can access the arguments of a system task, and therefore, simulators can predict only at compile time, about the information that shall be accessed by PLI application [18]. It was defined in a file 'veriuser.c'. To enhance its capabilities, ACC library was introduced, and was defined in a file 'acc_user.c'. It added up the facilities like power analysis, delay calculation and other types of analysis that involved cells that make up an ASIC netlist.Unlike TF library ACC library has the capability to access the structural level of design. But a major limitation was its inability to access RTL models, memory arrays, and many other objects that make up a large part in many Verilog based HDL models. Thus, a complete superset of these ACC and TF libraries, called VPI library, evolved.

VPI offers advantages of both the libraries and allow complete access to RTL models and behavioral code within a simulation data structure. Many inconsistent functions and redundancies were removed by replacing over 220 C functions with simpler 37 C functions [18].

A third alternative to interfacing HDL models with C/C++ functions is DPI-Direct Programming Interface. It originates from Synopsys VCS DirectC Interface and Co-Design Cblend Interface [19]. The DPI replaces the complex Verilog PLI with a simple and straight forward import/export methodology [20].

The PLI has many capabilities that are not required to interface Verilog with SystemC, but that add to the complexity of using the PLI. It act as a layer between Verilog simulation data structures and C code, and ensure type safety concerns, which adds overhead to simulation performance and complexity to the code. Interestingly, it allows inter-language function calls [21] as demonstrated in Fig. 2.3.

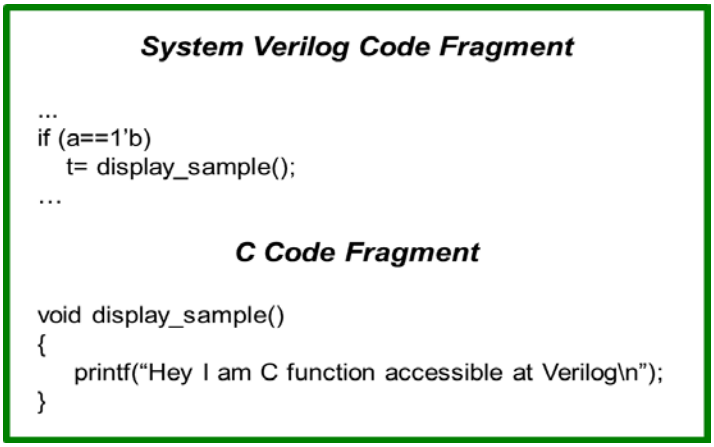

Fig. 2.3 Sample DPI-based program

2.3 System Upgradation

In corporate world, the softwares develop for internal use, are continuously evolving at design and development level, as per the requirement of users. The access to SLM facilities is now made available through GUI shell. Moreover, a test design strategy is being developed to support verification of new introduced features of SLM.

2.4 Test Suite Enhancement and Automation

2.4.1 Simulators

Immediate step after RTL coding is creation of test benches [22]. The test bench is used to inject a set of stimulus to the inputs of DUT, to check that the actual output conforms to the designers' expectation and system requirements. Once it is done, the simulators play the role to put together RTL code and testbench for simulation.

2.4.1.1 Incisive by Cadence

As referred from the spec manual by Cadence [23], key features and structure of Incisive are presented in this section. Incisive Enterprise Version of the Simulator combines a number of

components together, which includes, Simulator, Desktop Manager, eAnalyzer, Scenario Builder, SimVision, Verification Builder, and the Plan-to-Closure Methodology[23]. It allows to 'hot-swap' [23]the state of software based simulation for additional performance.
A more comprehensive view of the system is shown below in Fig. 2.4.

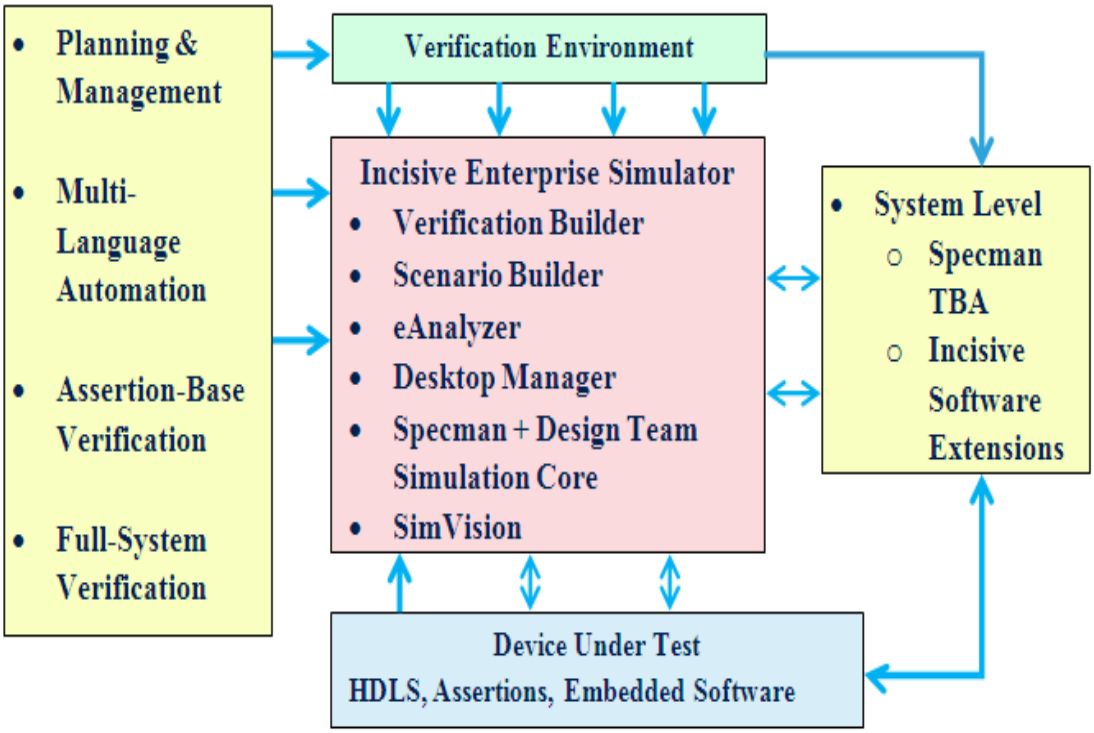

Fig. 2.4 Comprehensive View of Incisive [23]

2.4.1.2 Modelsim/Questasim by Mentor Graphics

ModelSim is a hardware simulation and debug environment primarily targeted at smaller ASIC and FPGA design. QuestaSim is a Simulator with additional Debug capabilities targeted at complex FPGA's and SoC's. QuestaSim can be used by users who have experience with ModelSim as it shares most of the common debug features and capabilities.

One of the main differences between QuestaSim and Modelsim (besides performance/capacity) is that QuestaSim is the simulation engine for the Questa Platform which includes integration of Verification Management, Formal based technologies, Questa Verification IP, Low Power Simulation and Accelerated Coverage Closure technologies. QuestaSim natively supports SystemVerilog for Testbench, UPF, UCIS, OVM/UVM where ModelSim does not.

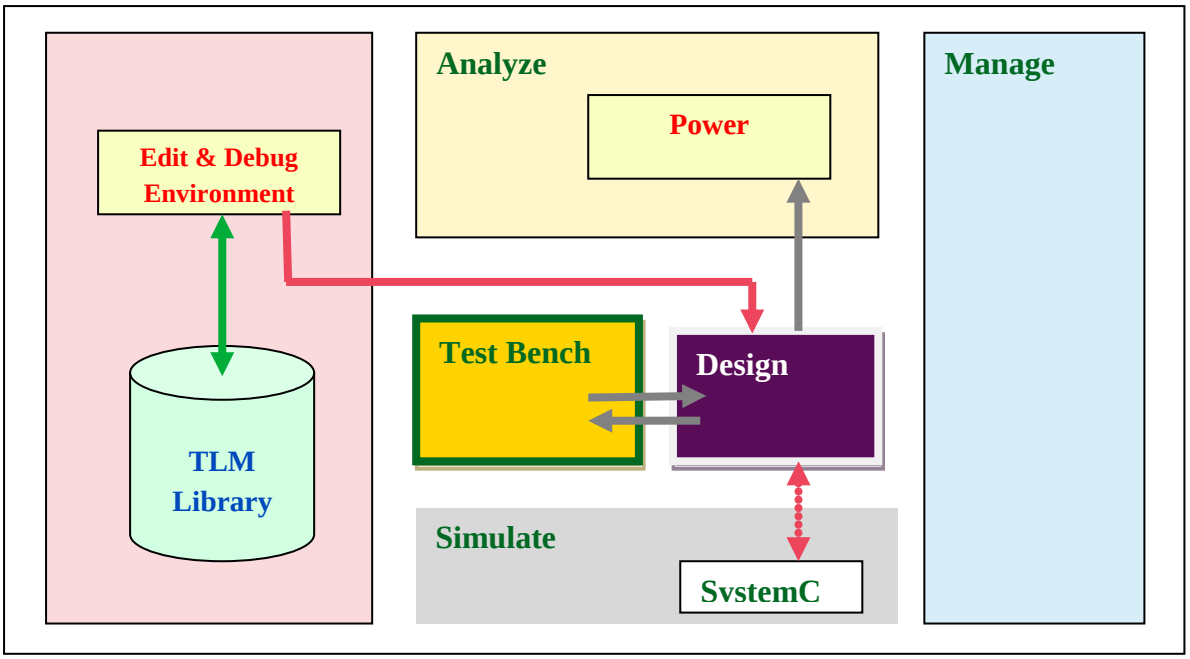

Fig. 2.5 Comprehensive View of ModelSim[24]

2.4.1.3 VCS by Synopsys

VCS leads the EDA market in terms of capacity and performance, with a set of advanced features such as bug-finding, constraint debugging, methodology-aware testbench, planning, coverage and assertion technologies. It delivers a 2x verification speed-up and cuts down verification time by using distributed architecture on multiple cores [25]. Easy setup and diagnostic reports by vcs improve productivity significantly.

Table 2.1 Modelling Languages and formats supported by VCS AMS[25]

Analog	Digital	Mixed-Signal
SPICE	Verilog	Verilog-AMS
Verilog-A	VHDL	Real Number Model
SPEF, DSPF, DPF	SystemVerilog	
	SystemC, Matlab	

VCS provide a high-performance, built-in coverage technology to measure verification completeness. Unified coverage aggregates all aspects of comprehensive coverage in single database, thereby allowing execution of powerful queries and generating useful unified

reports. The unified coverage database offers 2x to 5x improvement in merge times and up to 2x reduction in disk space usage, which is critical for large regression environments [25].

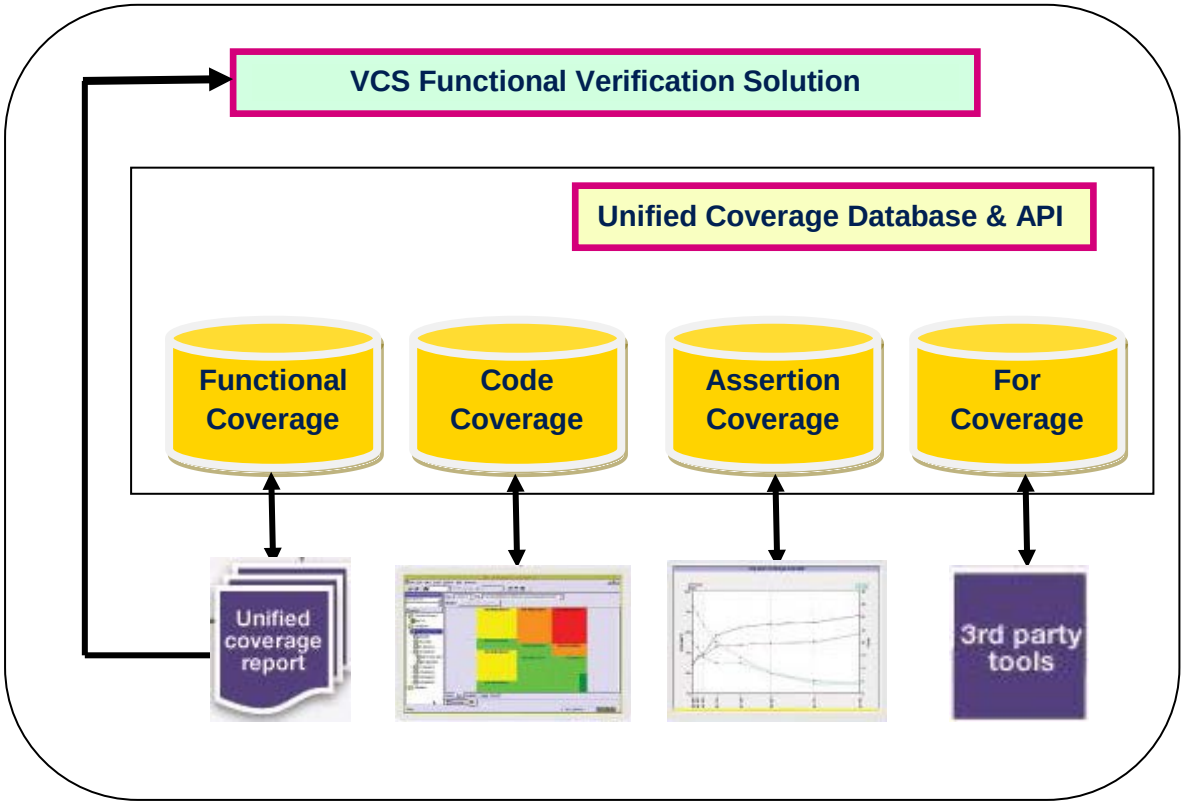

Fig. 2.6 Unified Coverage in VCS [25]

Chapter-3

Research Problem

Since nature of industrial research is significantly different than academic research, the problem statement is usually governed by those differences. Where academic research takes into account a broader range of parameters such as individual interests, trends in research, open problems, etc., industry focus on research based on market needs, business processes and resource planning in the industry. The thesis problem statement is formulated around the time-to-market constraint in semiconductor manufacturing industry. Moreover, the work in the thesis is done with objectives set from two different perspectives :

a. Project-Specific
b. Generic

3.1 Problem Statement

As discussed in previous sections, concept of system level memory and its features benefit a number of applications in SoC Design. But the challenge to ease its testing and maintainence pose significant delay to users. This directly hampers the usability of SLM and impact the time-to-market constraint due to delayed verification process. Scaling SLM to counter the rising gap between high complexity of specs and slower verification techniques is the driving force for upgrading various sections of the model. Broadly classified sections that impact the usability of SLM at STMicroelectronics are diagrametically presented in Fig. 3.1.

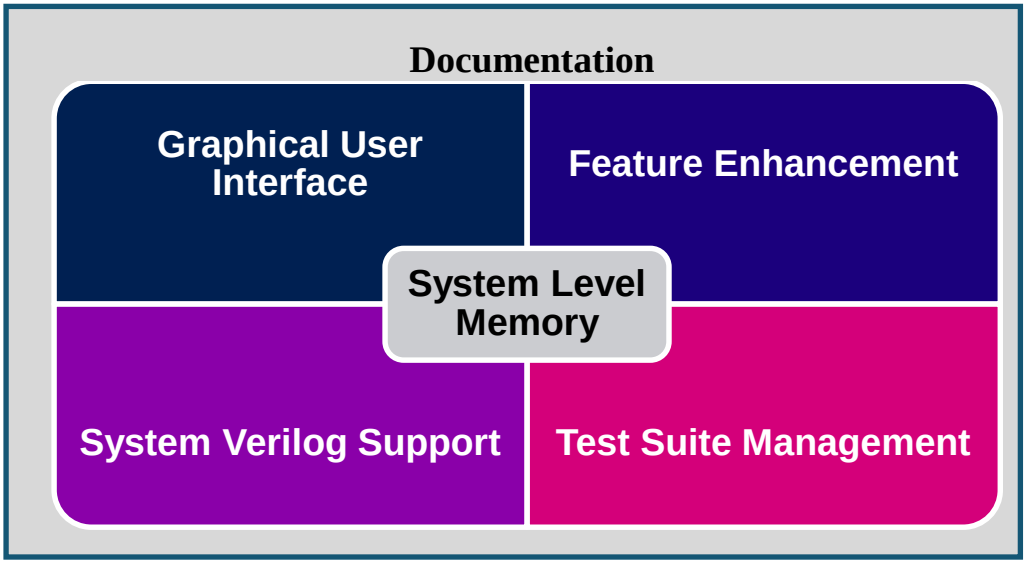

Fig. 3.1 Explored sub-sections for problem solving

A brief explanation about set of problems addressed is presented below:

a. **Graphical User Interface *:*** SLM is currently available in command line mode only. Thus, it is complex for designers and verification engineers to enjoy seamless integration of SLM, while carrying out hardware-software co-verification. Moreover,SLM is portable to a number of simulators, thus making it cumbersome to work with changing syntax, every now and then.

b. **Feature Enhancement :**SLM features must be enhanced to make it more user friendly and provide better accessability to HDL models through C function calls.

c. **System Verilog Support :** Extending support to System Verilog Based models is a crucial step to curb TTM constraint. Effort to code in System Verilog is much lesser than coding Verilog/VHDL based models.

d. **Test Suite Management :**SLM is a large-size software system that pose a number challenges to validation of complete system with incremental development of features. Thus an efficient automated mechanism is required fulfill this gap.

e. **Documentation :**Finally some work around for maintaining updates through a user guide should be provided for the designed system.

The above mentioned statement views our problem with a project specific view. From a generic view, some GUI design decisions and its testing methodology has been presented in the thesis. While designing GUI, we tried to address the problem of automated GUI testing and create a solution which, in a generic sense, can cater to need of GUI testing for in-house projects of organizations. The challenge there is, the proprietery systems of organizations are at security risk, while subjected to testing by free tools or third-party licensed tools.

3.2 Objectives

The main objectives of the research are :

a. Development of an automated testing strategy for the graphical interface SLM shell.
b. To enhance test-suite functionality by introducing more test-cases, for the system.
c. To generate and upgrade test-case for SLM in co-sim mode and TLM Mode.
d. Development of an automation strategy to automate tests.

e. Development of enhanced features for SLM graphical interface such as fault injection and function call mapping.

f. To extend System Verilog Support for following simulators :

- Ncsim
- Questa/Modelsim
- Vcs

3.3 Research Methodology

A pragmatic approach for research has been followed at STMicroelectronics. The research activity is initiated after problem specification. A number of methods and technologies were explored and after proper study, the choices have been made. A chart depicting flow of literature survey conducted has been presented below in Fig. 3.2.

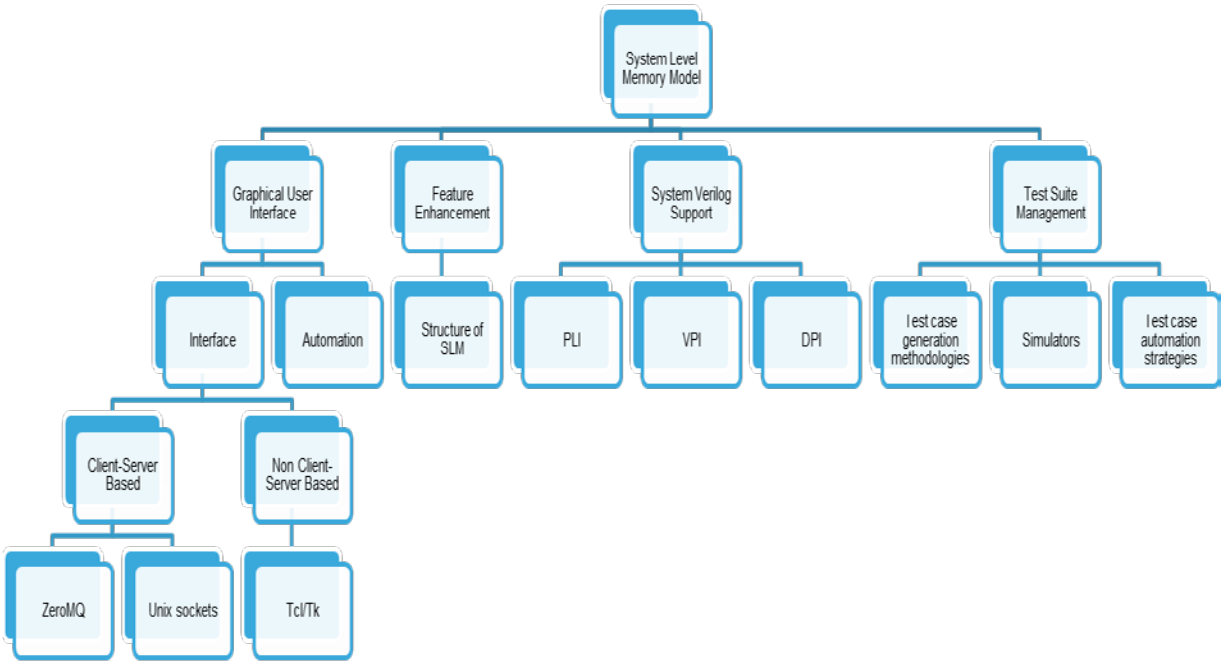

Fig. 3.2 Research Domain Tree

Chapter-4

SLM GUI Design

4.1 Design Decisions

In previous phase of explorations, communication architecture and implementation technology, were the center of focus. In current phase, some design decisions related to data sharing methodology and GUI testing methods, were taken. Given below is a brief account of explored methodologies for data sharing, our inferences, and the outcome of exploration.

4.1.1 Data Sharing Methodologies

A major challenge to be dealt-with is sharing the data between SLM Core and GUI. Since data is required to be consistent and accurate at every instant of simulation in Co-Sim and non-Co-Sim mode.Thus, we carefully studied various methods of data sharing and prototypes of viable options were created as will be presented in the implementation section. The prototypes established contribute to decide on the suitable ooptions in terms of setup complexity and run-time overheads.

a. Serialization based

- No scope for two-way communication
- Tightly coupled with legacy code of SLM
- Need Java expertise for C++ based SLM
- Not extensible

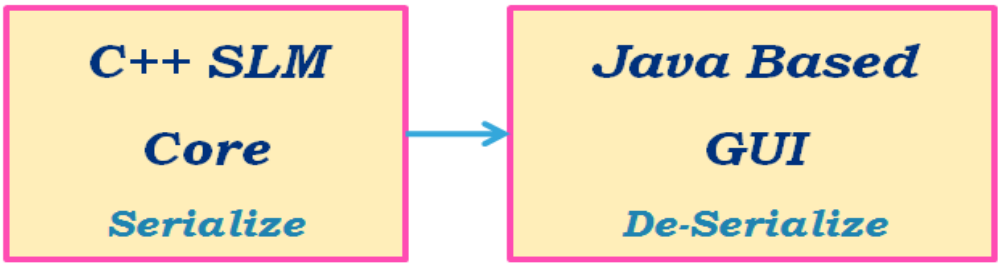

Fig. 4.1 Serialization based communication

b. C++ Object based

- No additional overhead on environment setup
- Seamless integretion
- Support two-way communication

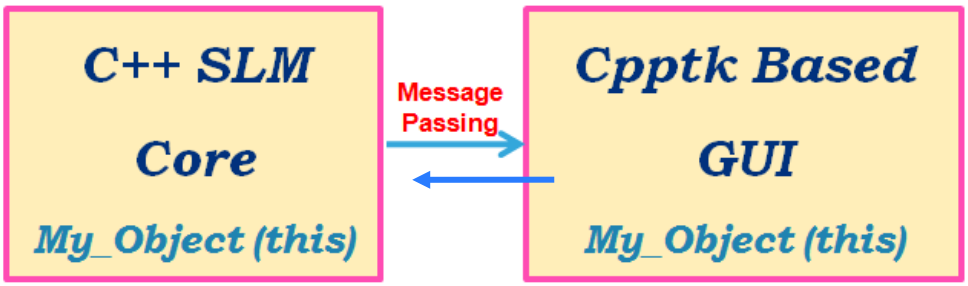

Fig. 4.2 C++ based communication

c. File based

- No additional overhead on environment setup.
- Intermediate tracing facility for not well-trained user of SLM.
- Contribute to offline independent launch of GUI

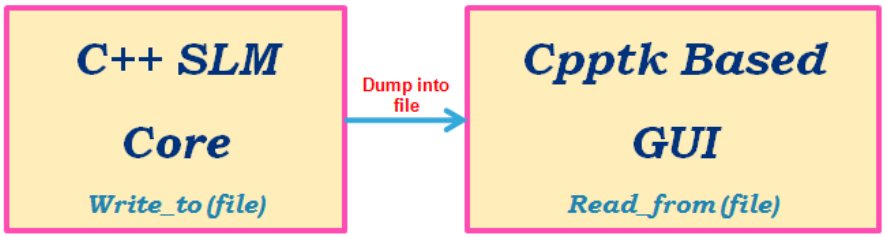

Fig. 4.3 File based communication

d. In-memory XML based

- A concrete C data-structure would allow easier data sharing.
- Overhead of XML maintainance.
- Complex environment setup.

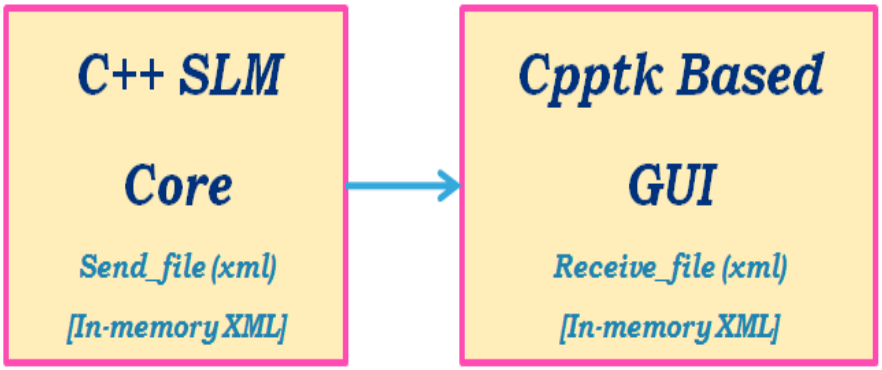

Fig. 4.4 In-memory XML based communication

Result of exploration : ***C++ data structures or file dumps to be used for sharing data between cpptk based GUI and SLM Core.***

4.2 System Design

4.2.1 System Architecture

In the basic architecture, an overview of sub-systems visible to SLM users, is presented. It do include the internal sub-systems of SLM. Once the SLM Core is initialized the SLM Scratchpad layer will prepare all the information required for GUI sub-system and push it towards SLM Data Exchanger. This Data Exchanger will only communicate with GUI thereby making GUI isolated from SLM Core.

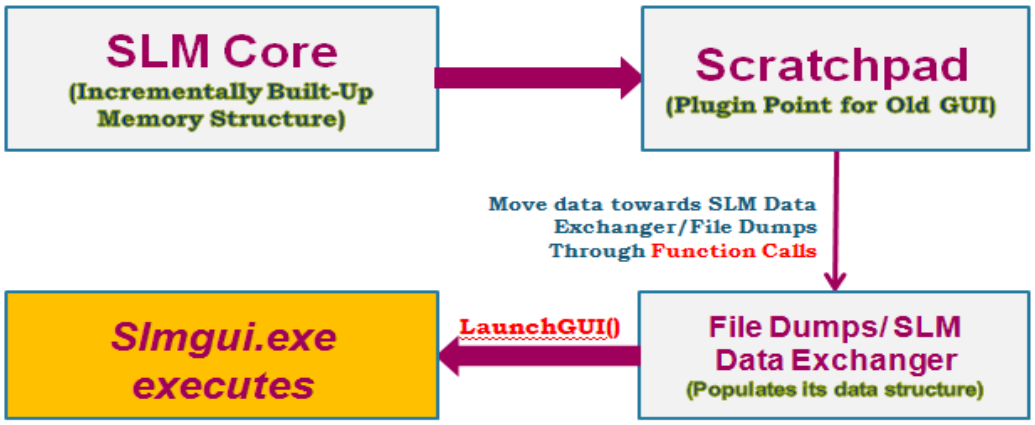

Fig. 4.5 SLM basic architecture of sub-systemss

4.2.2 Layered View of SLM

After a proper study of SLM design we located the layers which will help in re-construction of SLM Core. Fig. 4.6 shows the identified layers and the plugin points in the system.

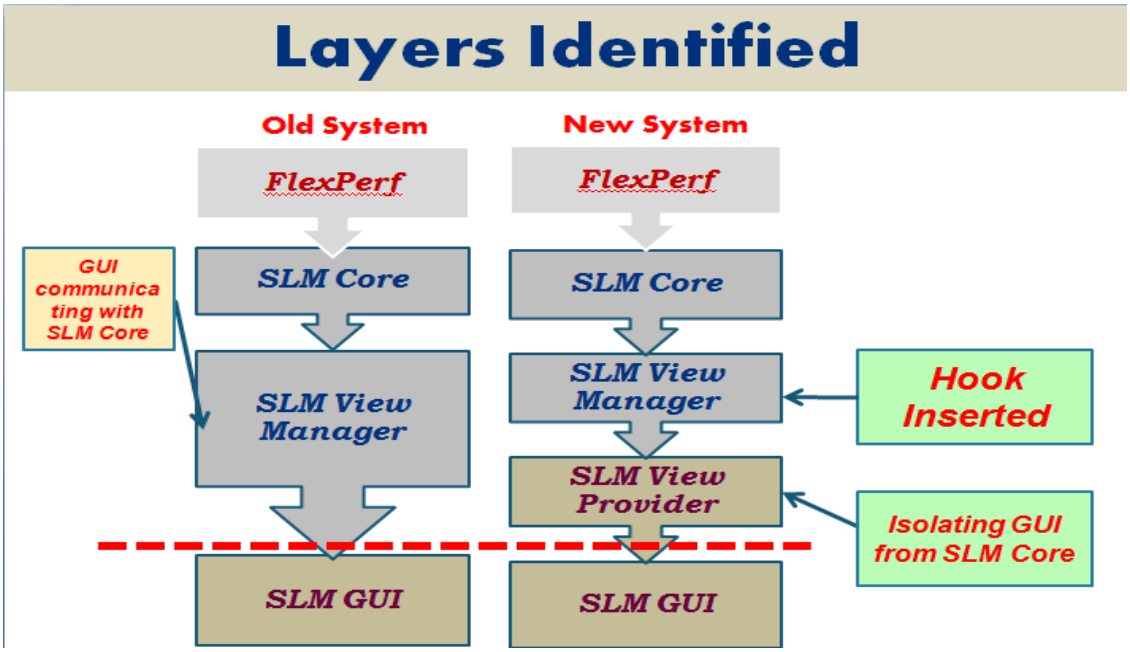

Fig. 4.6 Comparative Layered View of SLM

Basically we added up many small hooks within the core, a crucial hook inside the SLM View Manager and an isolating layer of SLM View Provider. A brief about crucial hook and isolating layer is been provided as follows :

a) SLM View Manager : Init function act as function making arrangement for all information required by SLM Exchanger based on function invoked. Function is differentiated through its parameter : function_differentiator.

- *Init*will make arrange for all the required data as per requirement.
- *Function_differentiator* will identify the functions where from the GUI is invoked. (eg: TbInfo, TbView etc.)

```
101 //By Silvi
102 //
103 //
104 //
105 //
106
107
108     /* Function which export data to exchanger
109      * It will keep the data structure with it, populate it in consolidated way.
110      * Exchanger will have a simpler buffer structure to store data.
111      * It will expose get/set functions to exchanger which are directly accessible to GUI
112      */
113
114     int init(int function_differentiator);
115 //
116 //
117 //
118 //By Silvi
```

Fig. 4.7 Hook in SLM View Manager

b) SLM View Provider : It acts as the layer responsible for isolating GUI from the SLM Core.

a) **Attributes :** Set of attributes used by exchanger for communication and self management are shown in Fig. 4.8.

```
class SlmDataExchanger
{
    private:


    int Sons_Count;
    static SlmDataExchanger *self;
    static bool instanceFlag;


    // it will contain complete set of properties retrieved from the SLM Core
    SLMCoreGuiDataExchangeInfoSet::MemoryData* dataset;


 public:

 SLMCoreGuiDataExchangeInfoSet::Parameters* param;
```

Fig. 4.8 Attributes of SLM View Provider

b) Container Data Structure

The subset of data shared with GUI is in form of C++ data structure. The handle to this data-structure is shared by SLM View Provider with GUI. SLM View Provider populates this data with the help of SLM View Manager, which pushes the set of information once its initialization is complete.

```
/*************************************Purpose*****************************************************
# Contains data structure shared between SLM Core and GUI through the DataExchanger
 ***************************************************************************************************/
#include <pthread.h>

namespace SLMCoreGuiDataExchangeInfoSet
{
    struct MemoryData
    {
        char* logical_name;
        char* alias;
        unsigned short mau;
        unsigned short address_bus_size;
        unsigned int nmau;
        unsigned int begin_addr;
        unsigned int end_addr;
        unsigned short id;
        unsigned int mem_type;
        unsigned int width;
        unsigned int depth;
    };

    struct Parameters
    {
        MemoryData* mem;
        int son_count;
    };

    struct NewThread
    {
        pthread_t pt;
        int thread_id;
    };
}
```

Fig. 4.9 Current data structure for retrieved information

c) Behaviour

Methods used to access and populate exchanger interface are described in Fig. 4.10.

```


    void setLogicalName(SLMCoreGuiDataExchangeInfoSet::MemoryData mem,char* logicalName);
    void setAlias(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, char* alias);
    void setMau(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned short mau);
    void setAddressBusSize(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned short address_bus_size);
    void setNmau(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, int nmau);
    void setBeginAddress(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int begin_addr);
    void setEndAddress(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int end_addr);
    void setId(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int id);
    void setMemType(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int mem_type);
    void setWidth(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int width);
    void setDepth(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int depth);


```

```

    //used by SLM View Manager to push data towards Provider and Provider will populate its own dataset through it.
    void moveData(char* name, char* alias, unsigned int id, unsigned int width, unsigned int depth, unsigned short mau, unsigned short address_bus_size,
unsigned int nmau, unsigned int begin_addr, unsigned int end_addr, unsigned int mem_type, int i);

    //used by SLM View Manager to instruct Provider that it should launch GUI which thus follows the instruction.
    int launchGUI(int function_identifier);

};

```

Fig. 4.10 Behavior of SLM View Provider

Fig. 4.11 shows the SLM Data Exchanger related constructors and destructors.

```
34
35      SlmDataExchanger(int NoSons)
36      {
37          Sons_Count = NoSons;
38          dataset = new SLMCoreGuiDataExchangeInfoSet::MemoryData[Sons_Count];
39      }
40
41
42      ~SlmDataExchanger()
43      {
44          instanceFlag=false;
45      }
46
```

Fig. 4.11 Constructors & Destructors for SLM View Provider

4.2.3 Overall Process Flow Description

- **Assemble Configurations**
  - Done by Makefile related to Core SLM
- **Prepare Core**
  - Initialize SLM Core in Co-Sim Mode
- **Init ()**
  - Prepare information required by GUI
  - Initialize SLM View Provider and push data to help populating its data-structure
  - Share the data-structure with GUI and let it “BUILD-SELF”.
- **Steps for GUI Building**
  - GUI Builds itself in three steps:
  - **Environment Checking :**Finds required environment variables.

```
1828 //=============================================================================
1829
1830 void *initDummySLMCore(void* p)
1831 {
1832
1833     std::ostringstream ss;
1834     try
1835     {
1836         extern char *__progname;
1837         printf("%s\n", __progname);
1838         init(__progname);
1839
1840         GUIGlobalInfoSet::status.open("Status.log",std::ofstream::out);
1841
1842         SLMCoreGuiDataExchangeInfoSet::Parameters *param = static_cast<SLMCoreGuiDataExchangeInfoSet::Parameters *>(p);
1843         GUIGlobalInfoSet::status << "------------------------Initializing data from SLM Core-------------------------\n";
1844
1845         ss.str("");
1846         for (int i=0; i<param->son_count; i++)
1847         {
1848             ss<< param->mem[i].logical_name << " ";
1849         }
1850
1851         GUIGlobalInfoSet::status << "Memory Structures initialized.\n\n";
1852
1853
1854         char* gui_enable = getenv ("GUI_ENABLER");
1855         if(gui_enable==NULL)
1856         {
1857             GUIGlobalInfoSet::status << "Please set GUI_ENABLER environment variable to 1 for enabling GUI.\n";
1858         }
1859         else if(strcmp(gui_enable,"1")==0)
1860         {
1861             guiStarter(ss.str(), param);
1862         }
1863         return NULL;
1864     }
1865     catch(...)
1866     {
1867         GUIGlobalInfoSet::status << "****Exception in initialization";
1868     }
1869
1870     return NULL;
1871 }
1872
```

Fig. 4.12 Environment Checking

- **GUI Starter :**Initialize GUI Core Information.

```
1795
1796 void guiStarter(std::string mem_list, SLMCoreGuiDataExchangeInfoSet::Parameters *param)
1797 {
1798     try
1799     {
1800         GUIGlobalInfoSet::status << "----------------------Extracting core information for GUI Builder----------------------\n";
1801         CoreInfo *GuiInfoHandle = CoreInfo::getInstance();
1802
1803         GUIGlobalInfoSet::status << __FILE__ << ": " << __LINE__ << ": " << __func__ << "() \n\n" ;
1804
1805         if(GuiInfoHandle!=NULL)
1806         {
1807             GUIGlobalInfoSet::status << "-------------------------Building GUI-------------------------\n";
1808             GuiBuilder builder;
1809             builder.build(mem_list, param , GuiInfoHandle);
1810         }
1811     }
1812     catch(...)
1813     {
1814         GUIGlobalInfoSet::status << "****Exception in starting GUI \n";
1815     }
1816 }
1817
```

Fig. 4.13 GUI Starter

- **GUI Builder :**Populate the GUI with SLM Core Information.

```
1763
1764 int GuiBuilder::build(std::string mem_list, SLMCoreGuiDataExchangeInfoSet::Parameters *param, CoreInfo *GuiInfoHandle)
1765 {
1766     try
1767     {
1768         GUIGlobalInfoSet::status << "Data recovered... " << mem_list << "\n\n";
1769         populateTree(mem_list,param);
1770         #define TEST_MODE 1;
1771 #ifdef TEST_MODE
1772         GUIGlobalInfoSet::status << "-------------------Test mode activated---------------\n\n";
1773         TestSuite t(EventBinding::bindHandler.getBindingTable(), EventBinding::bindHandler.BT_SIZE);
1774 #endif
1775         runEventLoop();
1776         GUIGlobalInfoSet::status.close();
1777     }
1778     catch(...)
1779     {
1780         GUIGlobalInfoSet::status << "****Exception in building GUI \n";
1781     }
1782     return 0;
1783 }
```

Fig. 4.14 GUI Builder

Chapter-5

SLM GUI Test Suite Design

5.1 Design Decisions

In previous phase of explorations, communication architecture and implementation technology, were the center of focus. In current phase, some design decisions related to data sharing methodology and GUI testing methods, were taken. Given below is a brief account of explored methodologies for GUI testing methods, our inferences, and the outcome of exploration.

5.1.1 GUI Testing Methodologies

In this phase of exploration, we tried to construct a generic approach that can provide effectively a solution to GUI testing for in-house projects. Thus, here are some approaches which we explored and observed certain pros and cons.

- **GUI Robo:** But that may be subject to usability since the input file it takes as interface description, may miss out on dynamically changing interface.

- **Test Script:** As the complexity of GUI system scales up, they fail to meet the expectations. They require a lot of human intervention if the GUI is dynamically changing.

- **Model Based Testing:** It can be a lot more efficient to handle the dynamic GUI systems. Model-based testing can also contribute to eliminating defects in requirement, and reduce typically manual design effort.

- **Bayesian Model based:** The assumption it takes regarding freezed source code, again makes it non-adaptable to the dynamic perspective of interface layouts.

- **Statechart Models:** Specification model in terms of Statechart diagrams, and further translating them to XML format. This can be effective in handling dynamic nature of layouts.

Result of exploration : ***We planned to introduce an approach able to handle GUI layout dynamicity and a generic concept extensible to number of in-house projects. Thus, wechose a combination of model tracking through 'Binding Tables' and generic test-script through Cpp code.***

5.1.2 Design Approach

The methodology revolve around the effort to allow the system to take important and responsiveparticipation of responsibility once it is out for testing phase. Thus a development strategy is being modelled that can facilitate easier testing automation. Basically, the thesis try to cover GUI testing in three perspectives as depicted in Fig. 5.1.

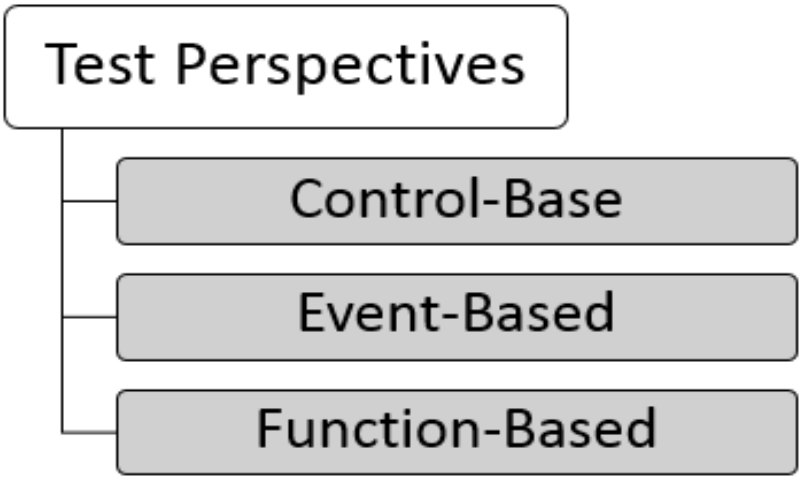

Fig. 5.1 Three perspectives for GUI Testing [16]

A. CONTROL AND EVENT BASED COVERAGE

The term "control" refers to the layout components that can dynamically change with unpredictable user-interactions and events. When any control is initialized, it get registered with the "Binding Table" with certain set of attributes. It also contribute to execute automated set of tests, and make troubleshooting while failures easier [16].

The "Binding Table" concept ensure that each dynamically added control, during interaction of user with GUI, is being in the list of testing program. Whenever an event is invoked on a control, certain function is performed. Is_update variable will act as indicator to the completion of event testing for a particular control. It require very less effort for implementation, as it adds is a single line call code to initializer function, which allocates

memory and registers the controls [16]. We just need to overload the initializer function to make it generic for any language. Pseudo code depicting Binding Table Structure is depicted in the Fig. 5.2.

```
class BindingTableStructure
{
public
       Table_Size;
       structure BindingTableEntry
       {
              control_name;
              registered_event_name;
              control_type;
              BindingTableEntry child_controls;
              is_update;
       };
       function registerEvents(std::string
control_name, std::string label);
       structure BindingTable
       {
              BindingTableEntry entry[1000];
       };
       BindingTable getBindingTable();
Private:
       BindingTable table;
}
```

Fig. 5.2 Binding Table Structure using pseudo-code [16]

Fig. 5.3 depicts brief log of Binding Table contents enabled during test mode operation.

```
--------------------Test mode activated----------------


Control - .mbar.fl registered for event - command

Control - .mbar.vi registered for event - command

Control - .mbar.vi registered for event - command

Control - .tab1.home.view registered for event -
EventBinding::gridLoader

Control - .tab1.home.move registered for
event - .tab1.home.result yview

Control - .tab1.home.movex registered for
event - .tab1.home.result xview

Control - .tab1.home.scrolltree registered for
event - .tab1.home.tree.yview

Control - .tab1.combar.go registered for event -
EventBinding::bye

Control - .tab1.combar.move registered for
event - .tab1.combar.result yview
```

Fig. 5.3 Sample log of Binding Table contents [16]

B. FUNCTIONAL COVERAGE

Functional coverage of system is ensured using a script of milestones for each function. Say 'n' functions are the features provided in the system, it will take approximately O(n) flags in the script to be coded individually [16]. It is always better for a developer to keep a check on tracking requirements while preparing a design so that, a reliable structure for test plan is prepared by the time design document is finalized. It can also contribute to automated execution of test-case generated.

Fig. 5.4 show the xml syntax in which trackers for every function and every branch of it can be placed into a milestone-holder file. This file will later act as input to the function coverage test module. This can also be maintained as an in-memory xml to save the complexity of file operations. Basically **dump_milestone*()*** would be responsible to generate this file.

```
<Signature_By_Function>
        <Tested> </Tested>
        <Specific Input> </Specific Input>
        <Updated> </Updated>
        <branch_signature>
                <Tested> </Tested>
                <Specific Input> </Specific Input>
                <Updated> </Updated>
        </branch_signature>
<Signature_By_Function>
```

Fig. 5.4 XML format for milestone representation used to dump branch info [16]

Brief description of the fields in the syntax are as follows [16]:

1. Signature_By_Function

It can take the value as any alias representing a milestone to a particular feature of the system. It must be ensured that each function should be tagged with a unique milestone.

2. Branch Signature

It will place the milestone at each branch of a function. Branch includes all the choice-based constructs of the function.

3. Tested

It can take the value as 0 or 1, indicating if the testing has been done after latest update or not. It will be set to 0 as soon as updated flag corresponding to it is set to 0.

4. Specific Input

Any special inputs to the branch if required for testing can be dumped into this field. It will ease the test case generation of specific scenarios.

5. Updated

It can take the value as 0 or 1, indicating if a repeated testing of a function is required after change in system state. It automatically set to 1 if the tested option is updated.

```
int Test_notify_milestone(string func, string branch)
int Update_notify_milestone(string func, string branch)
int create_milestone(string func, int test, <generic>)
int create_branch (string parent, string child_mile, int test, <generic>)
boolean is_mile()
dump_milestone()
```

Fig. 5.5 API model associated with Functional Coverage Mechanism [16]

If not dumped in a file, the mile_stone_xml can also be directed into the test case generator using xsd.

C. TEST CASE GENERATION

Test case generator is introduced to generate inputs automatically, and test all the branches of function automatically. It ensures that no control fails the system under exceptional set of inputs and no case-specific inputs are missed. It thoroughly checks each function with all type of inputs including specific cases for the function under test are taken from the milestone data-structure [16]. Good inputs can be supplied using boundary values and a random value generator functions for each data type [16]. Functions can be tested for exceptional inputs using out-of-bound values or garbage value generator to the function.

5.1.3 System Architecture

The basic architecture of system is inspired by introducing helping structures during develoment phase, so that systsem can be made more responsive in its testing phase. A diagrametical representation to depict how those helping structures complement automated testing process is given in Fig. 5.6.

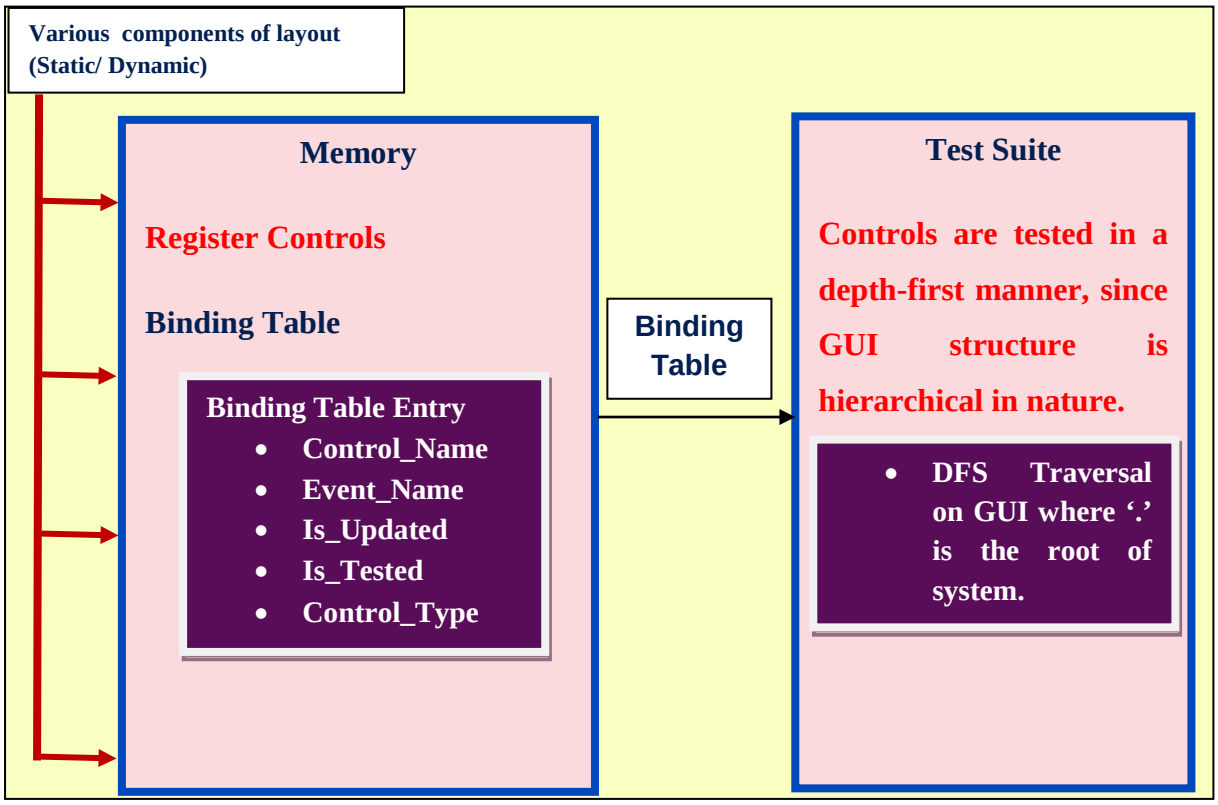

Fig. 5.6 System Architecture for GUI Test Suite

Chapter-6

SLM Test Suite Design

6.1 System Design

Any test case while running follows an order of execution. It takes RTL input from the Verilog, VHDL codes, sends in the layout in elaboration phase, where SLM libraries are then fetched in. Finally, a Co-Sim mode execution of design with software memory module is being done. Runtime order of execution is shown in Fig. 6.1.

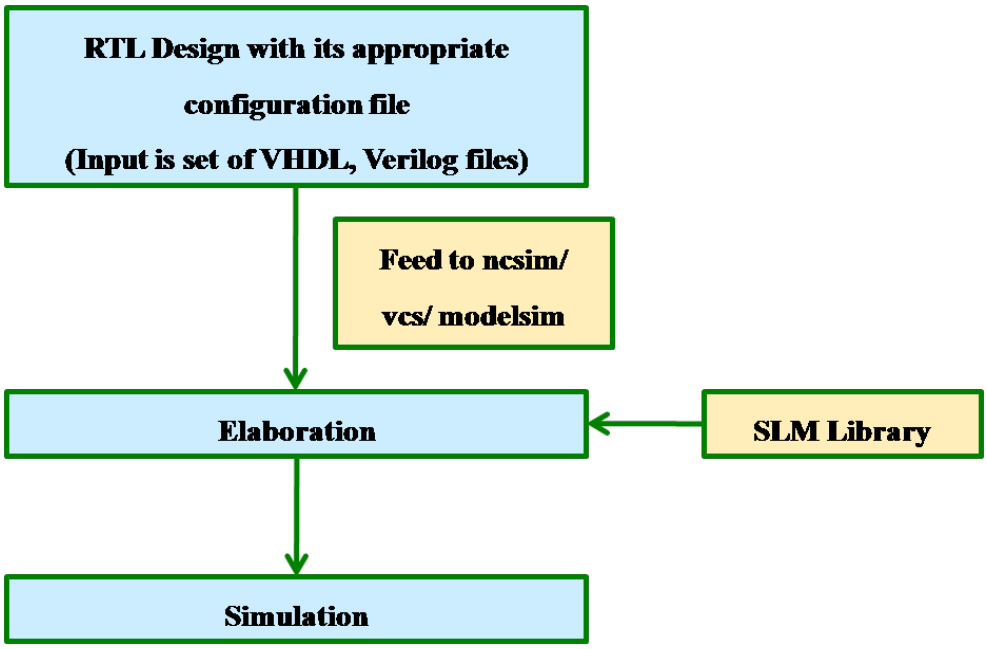

Fig. 6.1 Test Case Run-time

A number of test cases for SLM are being worked upon. A basic design view of SLM Test Suite is presented below :

Table. 6.1 Categories of test plans in the test suite

Categories of Test Plans
Access
Debug
Utility
Verification
Recording
Co-Verification

6.2 Test Suite Structure

A basic structure of validation suite directory is presented below with their brief description:

a) References: Expected results w.r.t which the decision of tests are based, are defined in this section.

b) Test Cases: A set of test cases for various type of memories in SLM, are defined in this section.

c) Test Plan: A strategy to verify the system and deciding on its compliance with design and much of regression testing strategy is defined in this section.

d) Platforms: The definitions of platform building files and their access is defined in this section.

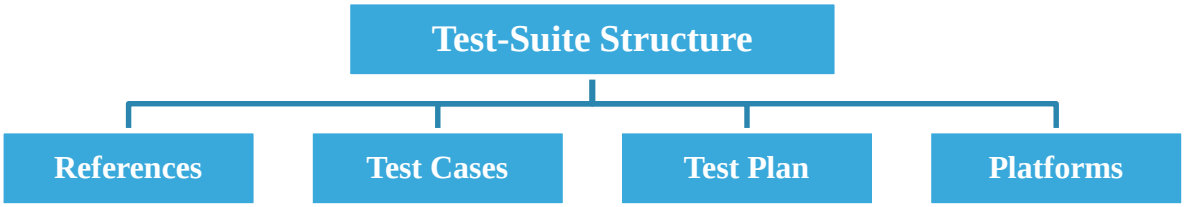

Fig. 6.2 Basic Test Suite Structure

Since SLM support a number of platforms, a few of them has been explored. SystemC and VHDL/Verilog based platforms are used for running the test cases. Test Plan are designed in C++ and HDL. Top level reference folder contain the benchmark for running logs, while internal References log the actual running tests. A tree view of test is presented as a sample for golden logs and actual running log.

```
References
|-- Adk
|   |-- GoldRun.log
|   `-- LOADFILES
|       |-- rom0.slm
|       |-- rom1.slm
|       |-- rom2.slm
|       `-- rom3.slm
|-- Assertion
|   |-- Assertion.dbi
|   |-- Assertion.dsn
|   |-- Assertion.trn
|   `-- GoldRun.log
|-- Basic
|   `-- GoldRun.log
|-- CadenceVHDLFormat
|   `-- LOADFILES
|       `-- TMP
|           |-- 1024x1.cdeIA
|           |-- 1024x2.cde
|           |-- 255x15.cde
|           |-- sed_cmd
|           `-- vloghex2bin
|-- PhySysRW
|   `-- GoldRun.log
|-- ReadWrite
|   `-- GoldRun.log
```

Fig. 6.3(a) References-Golden Log

```
dlhl2117{goels2}427:tree TestPlan/c++/access/AddrScram/Reference/
TestPlan/c++/access/AddrScram/Reference/
|-- AddrScram.log
|-- Phy0.sav
|-- Phy1.sav
|-- Sys0.sav
`-- Sys1.sav

0 directories, 5 files
```

Fig. 6.3(b) Reference-Runtime Log

The runtime log is compared with Golden log and results are written to Compare.rep file. A detailed view of test-suite, test cases and test plans is presented. Fig. 6.4. cover the major components which constitute to form the test-suite structure.

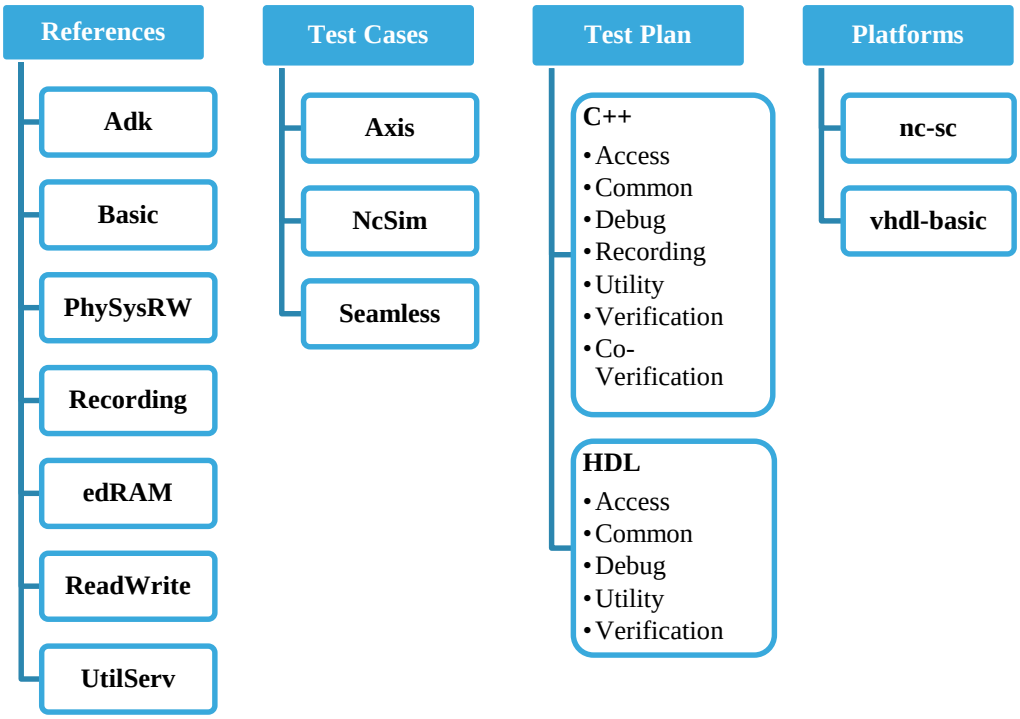

Fig. 6.4 Detailed Test Suite Structure

Fig. 6.5 covers the structure of Test Cases. It shows categories based on platforms over which various test cases are supported.

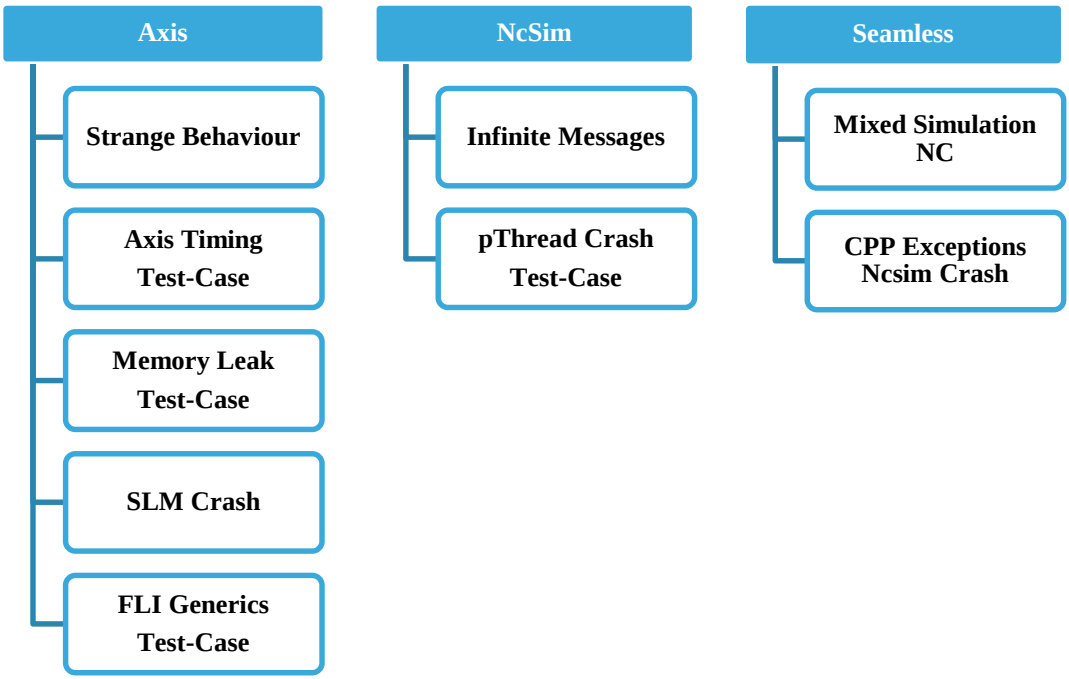

Fig. 6.5 Detailed Test Cases Structure

Fig. 6.6 cover the specific test plans designed to verify the internals of SLM Core. SLM is made testable in C++ and HDL environment. Basically C++ test plans are used in TLM mode and HDL test plans are used in Co-Sim mode.

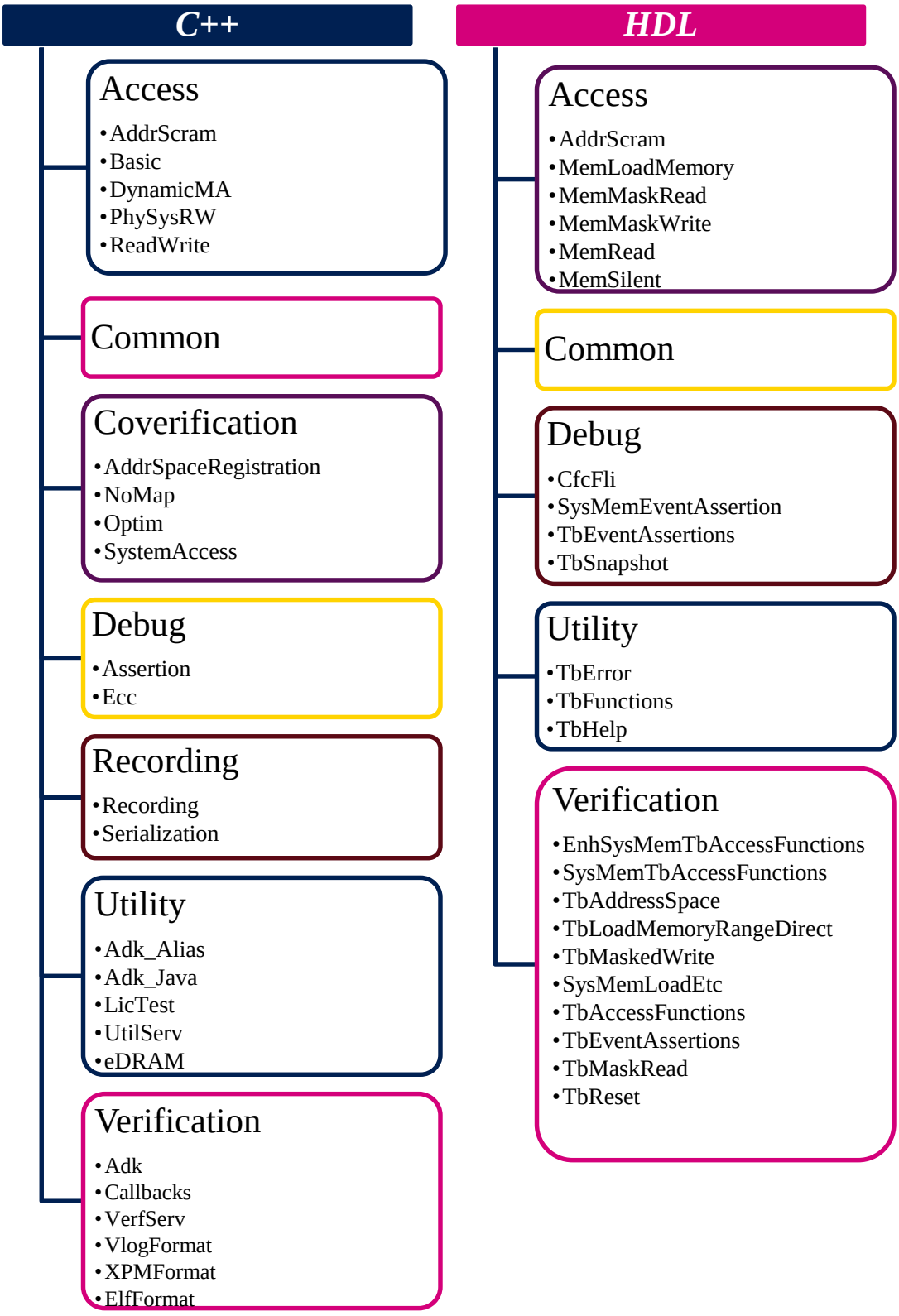

Fig. 6.6 Detailed Test Plan Structure

6.3 Test Case Description

In this section, some more test cases has been described briefly which have been a part of system. These test cases act as various scenarious under which SLM is used. These differ from others, in way that the test cases in test suite were testing functions, but these are test cases for various environments under which SLM can be invoked.

6.3.1 BIST

- **Test Case Description**

  This test case is used to depict SLM validating correct run of 14N March algorithm for BIST. For interactive debug we can use :

  a. Memory Viewer.
  b. Using TbRead and TbWrite at simulator prompt, these will have execute a back-door access into memory i.e. it does not have any side effect in terms of time or anything else but it will perform the read/write in memory.

6.3.2 eCallBackTest

- **Test Case Description**

  This test case aims at exhibiting usage of callback in SLM from "e" code. "e" code provides stimuli to do some writes in the memory and at the same time it waits on SlmEvent to take place that gets triggered whenever there is any read/write action on the SLM memory.

6.3.3 eReadWriteTest

- **Test Case Description**

  This test case aims at exhibiting backdoor read/write from/into SLM memory using SLM external services like TbRead and TbWrite. It also provides many scripts that show how slm can be linked with specman and ldv on 3 different ways:

  a. creating a static executable using e code written in eapi.e provided with SLM.
  b. dynamic loading of slm library into nc-specman executable using a dynamic shared library.
  c. loading both specman and slm dynamically into ldv.

6.3.4 Controller

- **Test Case Description**

  This test case shows the usage of SLM validating a System containing a Controller with some memories and capable of performing write and copy operations. Also it shows how to integrate C/C++ functions (external to SLM) with SLM. Indeed that features PLI (C/VERILOG) system tasks as well a CFC functions that do not belong to the SLM scope. Since there can only be one libpli.so and libcfc.so (implicit load), the different libraries would have collide in case implicit dynamic load is performed. For instance, such a thing happens when other CAD tools are involved in the logic simulation together with SLM components.

6.3.5 MemoryEvcTest

- **Test Case Description**

  It involves a BIST using 14N March Algorithm and single port memory. Along with verfication it also shows the usage of SLM in a Verilog and VHDL mixed environment. All BIST and Memory Models in this example are actual UNICAD models and the methodology is going to be used by the BIST team.

These test cases are basically supported on multiple communication interfaces including pli, fli, fmi and vhpi. Currently Bist and Controller are supported on ncsim, modelsim and vcs, but others are currently supported on ncsim only.

Chapter-7

Implementation and Results

7.1 SLM GUI & GUI Test Suite

7.1.1 Environment Setup

A systemetic view of setting up the environment has been adopted to make the system extensible and adaptable to various changes. The setup is categorized into four sections as depicted in Fig. 7.1.

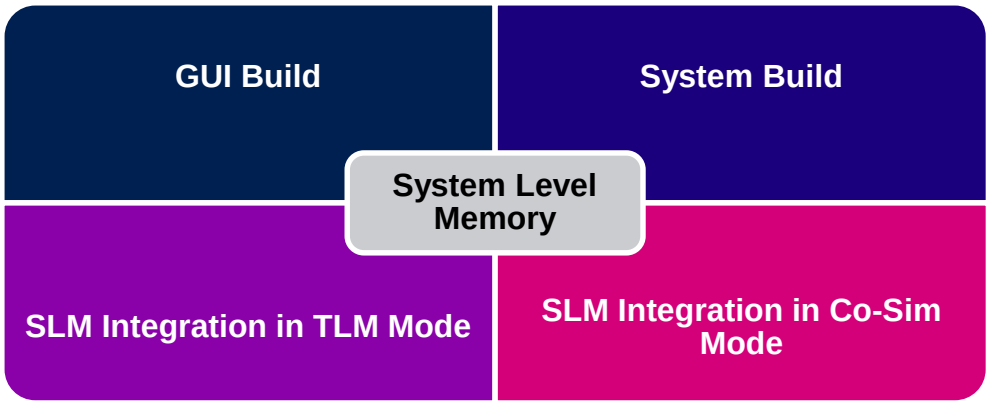

Fig. 7.1 Various sections in SLM Environment Setup

1. **GUI Build :** This section will be responsible for individual build of GUI and configuration related to integrate GUI with system level memory at run-time. Following section depicts the environment variable setup and makefile configuration of GUI.

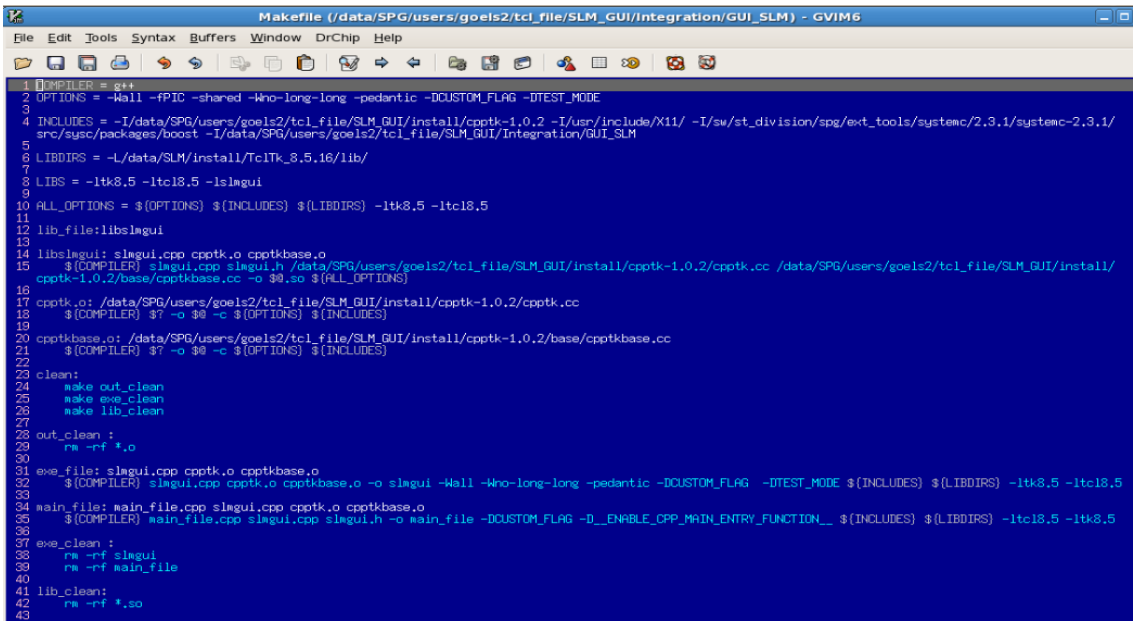

Fig. 7.2 Makefile for GUI Build

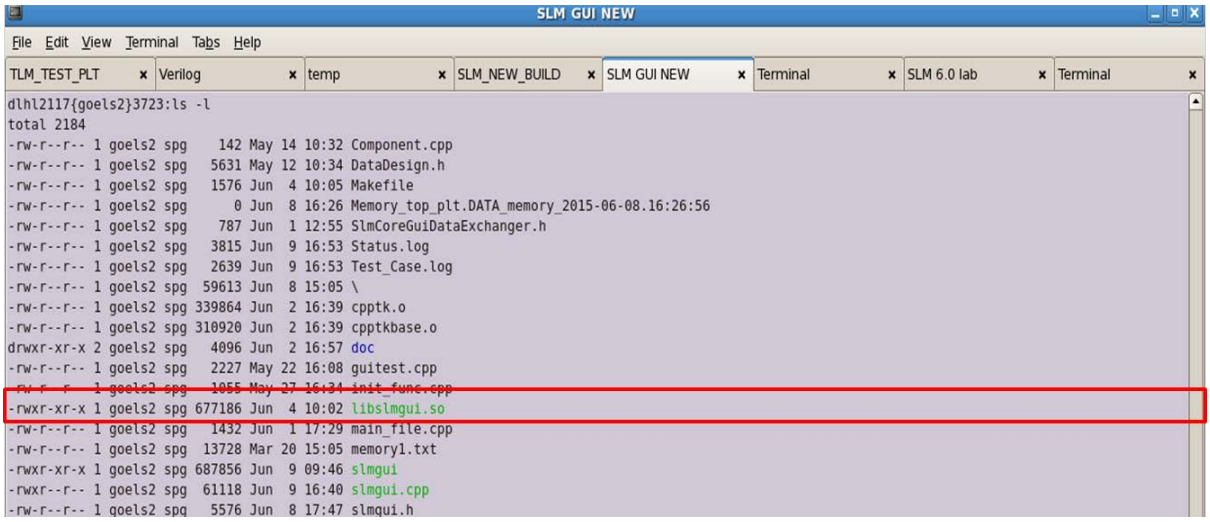

Fig. 7.3 Shared GUI Library

a) *GUI_ENABLER :* Set the value to 1 for launching GUI, else gui will not be lanuched.

```
1624
1625
1626        char* gui_enable = getenv ("GUI_ENABLER");
1627        if(gui_enable==NULL)
1628        {
1629            GUIGlobalInfoSet::status << "Please set GUI_ENABLER environment variable to 1 for enabling GUI.\n";
1630        }
1631        else if(strcmp(gui_enable,"1")==0)
1632        {
1633            guiStarter(ss.str(), param);
1634        }
```

Fig. 7.4 GUI_ENABLER Environment Variable

b) *TEST_MODE :*It is a macro definition based upon which the user mode and test mode are differentiated at runtime.

```
1536
1537 int GuiBuilder::build(std::string mem_list, SLMCoreGuiDataExchangeInfoSet::Parameters *param, CoreInfo *GuiInfoHandle)
1538 {
1539     try
1540     {
1541         GUIGlobalInfoSet::status << "Data recovered... " << mem_list << "\n\n";
1542         populateTree(mem_list,param);
1543 #ifdef TEST_MODE
1544         GUIGlobalInfoSet::status << "------------------Test mode activated--------------\n\n";
1545         TestSuite t(EventBinding::bindHandler.getBindingTable(), EventBinding::bindHandler.BT_SIZE);
1546 #endif
1547         runEventLoop();
1548         GUIGlobalInfoSet::status.close();
1549     }
1550     catch(...)
1551     {
1552         GUIGlobalInfoSet::status << "****Exception in building GUI \n";
1553     }
1554     return 0;
1555 }
```

Fig. 7.5 Test Mode Macro Controller

c) *NEWGUI_COMPILE_ENABLED :* If defined in slm_compile_generic python script, following configuration of variables is done in Makefile.

NEWGUI_INCLUDE_PATH

-I/data/SPG/users/goels2/tcl_file/ SLM_GUI/install/ cpptk-1.0.2/

-I/usr/include/X11/

-I/sw/st_division/spg/ext_tools/systemc/2.3.1/systemc2.3.1/src/sysc/packages/boost/

-I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM/

NEWGUI_LIBRARY_NAME

-ltcl8.5 –ltk8.5 –lslmgui

NEWGUI_LIBRARY_PATH

-L/data/SLM/install/TclTk8.5.16/lib/

-L/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM/

```
145 #BY_SILVI Introduced to integrate new gui compilation {{{
146     os.environ["NEWGUI_COMPILE_ENABLED"] = "true"
147     os.environ["NEWGUI_EXE_ENABLED"] = "true"
148     os.environ["TEST_MODE_ENABLED"]  = "true"
149 #}}} BY SILVI
```

Fig. 7.6 SLM Generic Compile Script for GUI Integration

➢ *NEWGUI_EXE_ENABLED*

- If defined, following configuration of variables is done in Makefile

  GUI_ENABLE = -DNEW_GUI_ENABLE

```
215 ifdef NEWGUI_COMPILE_ENABLED
216     NEWGUI_INCLUDE_PATH =  -I/data/SPG/users/goels2/tcl_file/SLM_GUI/install/cpptk-1.0.2 \
217                 -I/usr/include/X11/ \
218                 -I/sw/st_division/spg/ext_tools/systemc/2.3.1/systemc-2.3.1/src/sysc/packages/boost \
219                 -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM
220     NEWGUI_LIBRARY_NAME =  -ltcl8.5 -ltk8.5 -lslmgui
221     NEWGUI_LIBRARY_PATH = -L/data/SLM/install/TclTk_8.5.16/lib/ \
222                         -L/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM
223 else
224     NEWGUI_INCLUDE_PATH =
225     NEWGUI_LIBRARY_NAME =
226     NEWGUI_LIBRARY_PATH =
227     NEWGUI_EXE_ENABLED =
228 #SILVI
229 endif
230
231
232 ifdef NEWGUI_EXE_ENABLED
233 GUI_ENABLE = -DNEW_GUI_ENABLE
234 else
235 GUI_ENABLE =
236 endif
```

Fig. 7.7 SLM Makefile Script for GUI Integration

➢ *CUSTOM_FLAG*

- It is a work around to support certain non-available features in cpptk library. The change is made in cpptk.h file.

```
594 template <typename T1, typename T2>
595 details::Expr identify(T1 const &x, T2 const &y)
596 {
597 #ifdef CUSTOM_FLAG
598 //*************************************silvi:12-11-2014*****************************************
599 //*************************************Find element by row*************************************
600       std::string str("identify row ");
601 #else
602       std::string str("identify row ");
603 #endif
604       str += details::toString(x); str += " ";
605       str += details::toString(y);
606       return details::Expr(str);
607 }
```

Fig. 7.8 Modified header file cpptk.h

➢ *TEST_MODE_ENABLED*

- If defined, following configuration of variables is done in Makefile

  TEST_MODE = -DTEST_MODE

2. System Build : This section will be responsible for build of complete SLM, whose set of shared libraries can further be used further in tlm_mode and co-sim mode.

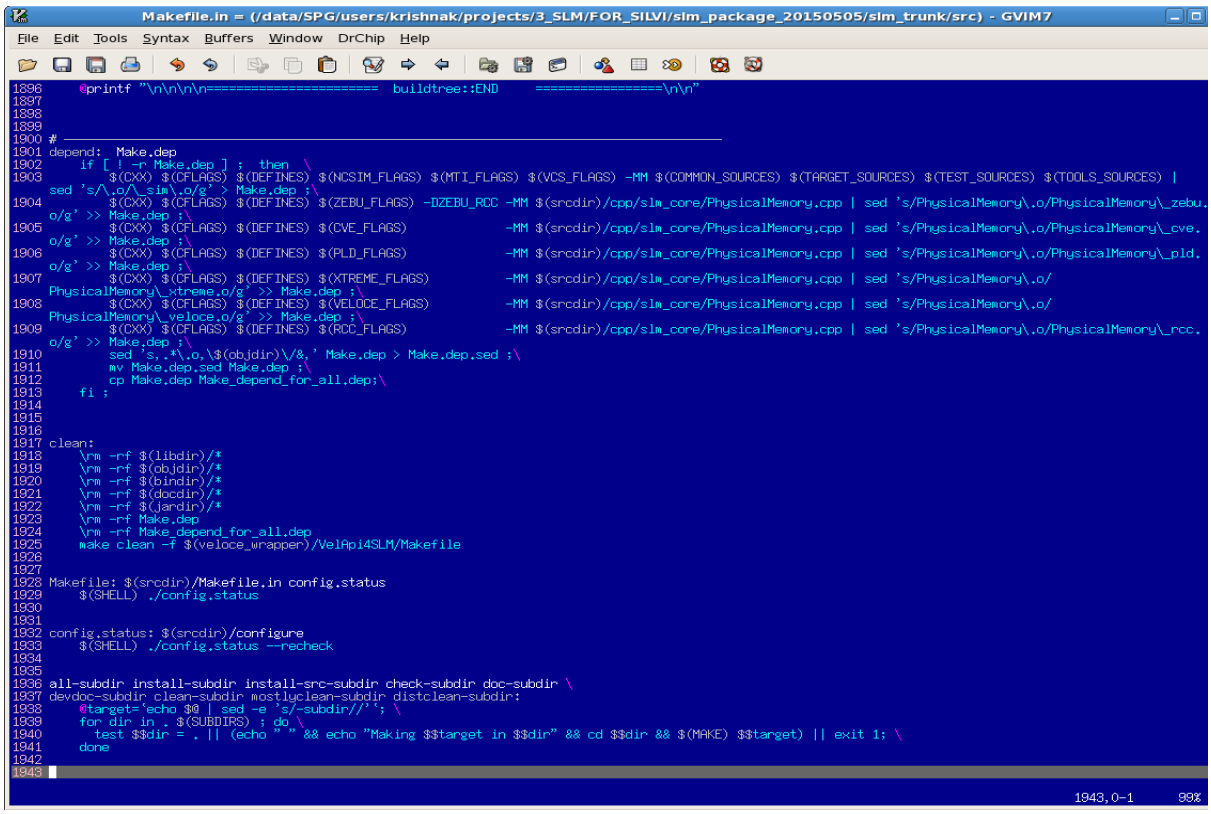

Fig. 7.9 SLM Build Makefile

3. System Integration in TLM Mode

```
tlm_config.pl -noktmp -pkg_tools=/sw/st_division/spg/spg_tools/SPG_SOS3/bld/pkg_tools/3.4.9 -key=/sw/st_division/spg/pkg_key3.txt -
key=/sw/st_division/spg/pkg_key_dlh_license.txt -key=/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SAND.SLM.TRUNK/setup/slm_key.txt -
obj=/sw/st_division/spg/spg_tools/SPG_AUTO_INSTALL -obj=/sw/st_division/spg/spg_tools/SPG_SOS3 -
obj=/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SAND.SLM.TRUNK -
obj=/data/SPG/users/krishnak/projects/3_SLM/FOR_SILVI/slm_package_20150505/INFRA_OBJECTS/ -
obj=/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SAND.SLM.TRUNK/setup -
dp=/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SAND.SLM.TRUNK/setup/slm_dp.txt -
prefix=/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SAND.SLM.TRUNK/setup/BUILD_SLM_TEST_PLT PLT slm_test_plt -with-USE_SLM=1 -with-SLM=1 -with-
SLM_BITS=64 -with-SLM_HOME-version=6.1.0 -with-
SLM_HOME=/data/SPG/users/krishnak/projects/3_SLM/FOR_SILVI/slm_package_20150505/slm_trunk/distrib_package/ddmmyyyy_optimized -with-CMP-tac_memory-
version=2.11.3 -with-BLD-pkg_tools-version=3.4.9 -with-DVK-command_line_manager-version=3.0 -with-EXT-syc-version=3.3.24 -with-EXT-tlm_standard-version=2.5
-with-EXT-cc-version=3.3.15 -with-CFLAGS=-DUSE_TAC_MEMORY -with-BLD-tlm_infra-version=3.5.10 -with-PRT-tlm_synchro-version=3.7.1 -v
```

Fig. 7.10 TLM Mode Configuration for SLM

4. System Integration in Co-Sim Mode

```
tlm_config.pl -noktmp -
pkg_tools=/sw/st_division/spg/spg_tools/SPG_SOS3/bld/pkg_tools/3
.4 -key=/data/SPG/users/goels2/lab2/key_plt.txt -
key=/sw/st_division/spg/pkg_key3.txt -
key=/sw/st_division/spg/pkg_key_dlh_license.txt -
obj=/sw/st_division/spg/st_tools/stbus_genkit/9.5.4 -
obj=/sw/st_division/spg/spg_tools/SPG_SOS_HOTFIX -
obj=/data/SPG/users/goels2/lab2/objects -
obj=/sw/st_division/spg/spg_tools/SPG_AUTO_INSTALL -
obj=/sw/st_division/spg/spg_tools/SPG_SOS3 -
obj=/data/SPG/users/krishnak/projects/3
_SLM/FOR_SILVI/slm_package_20150505/INFRA_OBJECTS -
dp=/data/SPG/users/goels2/lab2/DP_demo.txt -
prefix=/data/SPG/users/goels2/lab2/build_plt PLT top_plt -v -
prefix=build_plt
```

Fig. 7.11 Co-Sim Mode Configuration for SLM

7.1.2 Logging Facilities

1. **Test_Case.log ---** It log the pass and fail status of test cases.

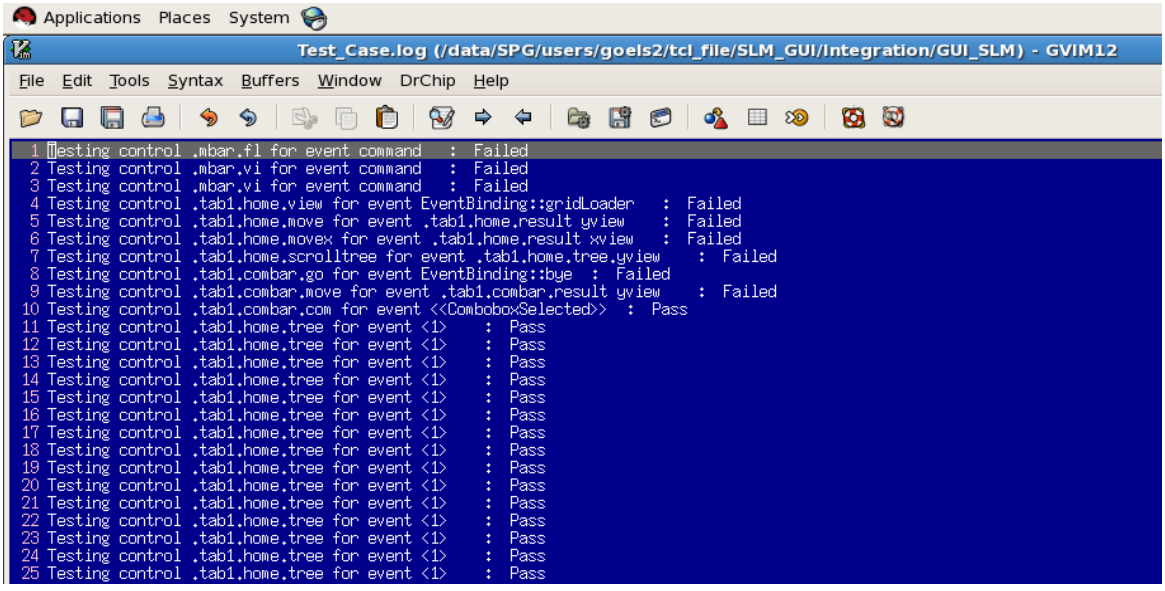

Fig. 7.12 Test_Case.log

2. **Status.log ---** It log the flow of program execution.

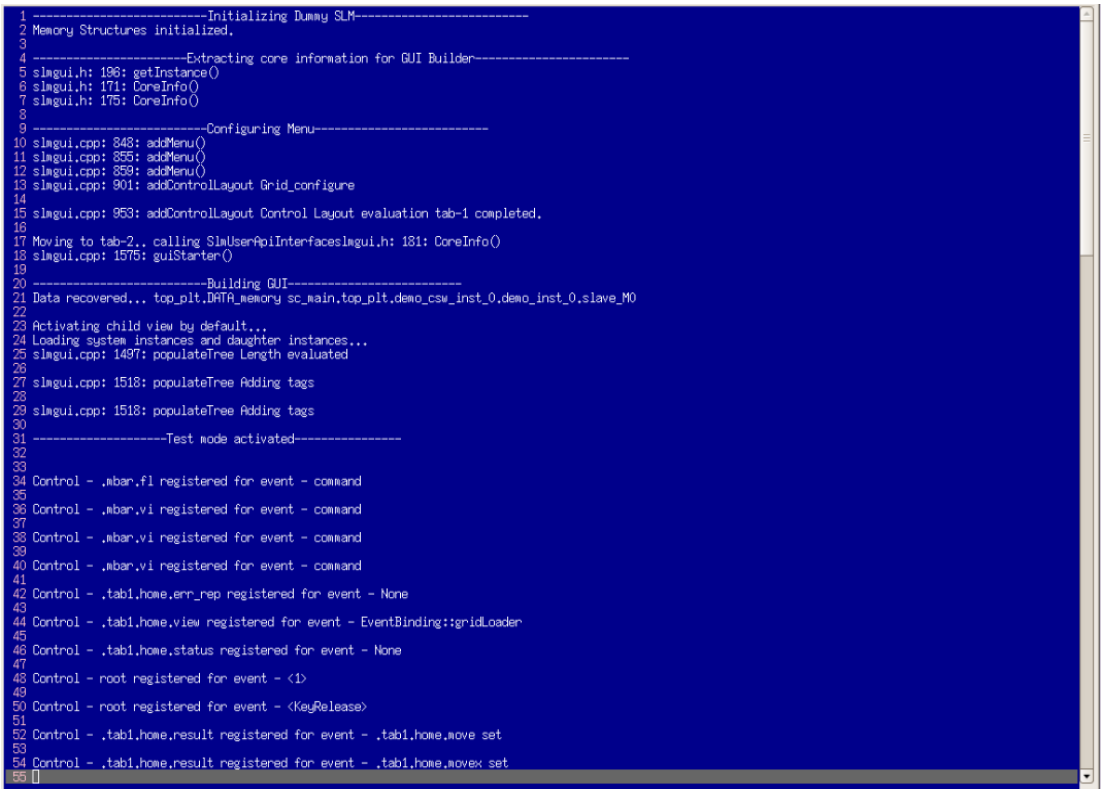

Fig. 7.13 Status.log

7.1.3 Execution Screenshots

1. **SLM Graphical User Interfaces**

Fig. 7.14 shows the main screen of GUI which shows the list of SLM Memory instances in memory.

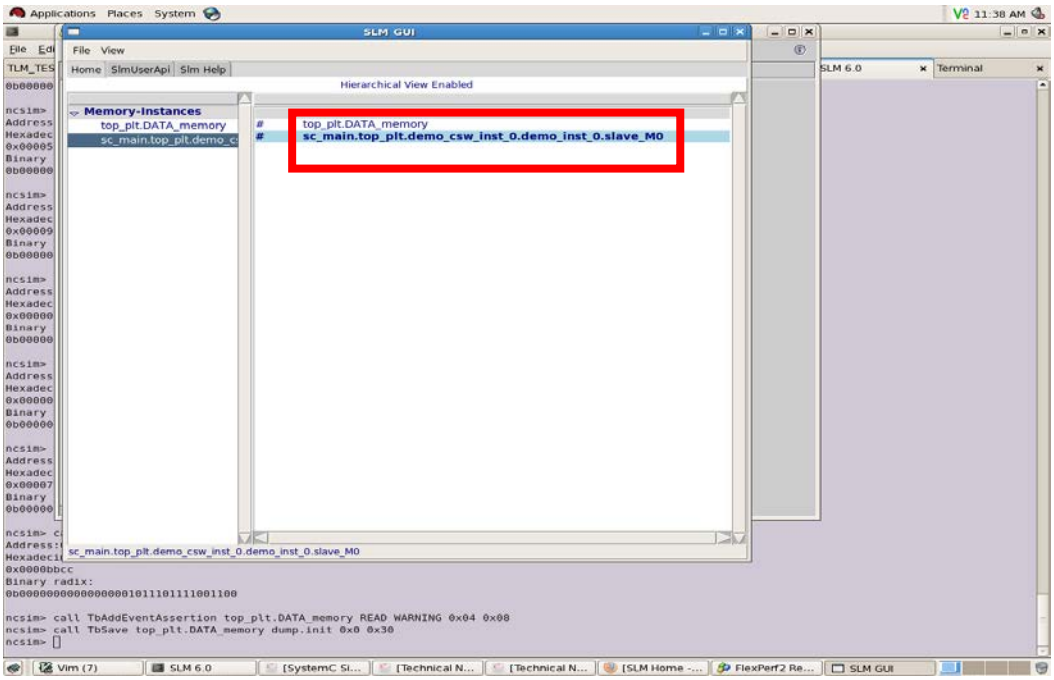

Fig. 7.14 Splash Screen

Fig. 7.15 shows the main screen of GUI which shows the properties of SLM Memory instance in memory.

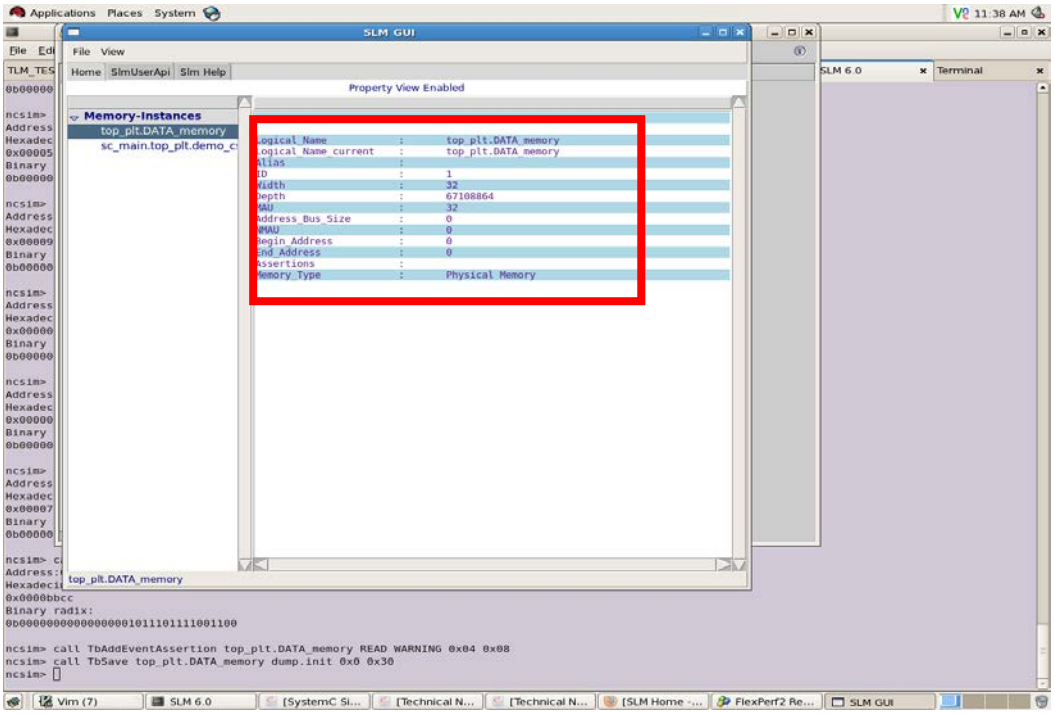

Fig. 7.15 TbInfo Screen

Fig. 7.16(a) shows the mapper interface of GUI which allow the SLM Internals to be directly invoked through GUI in Co-Sim & TLM Mode. In current snapshot, TbHelp is mapped.

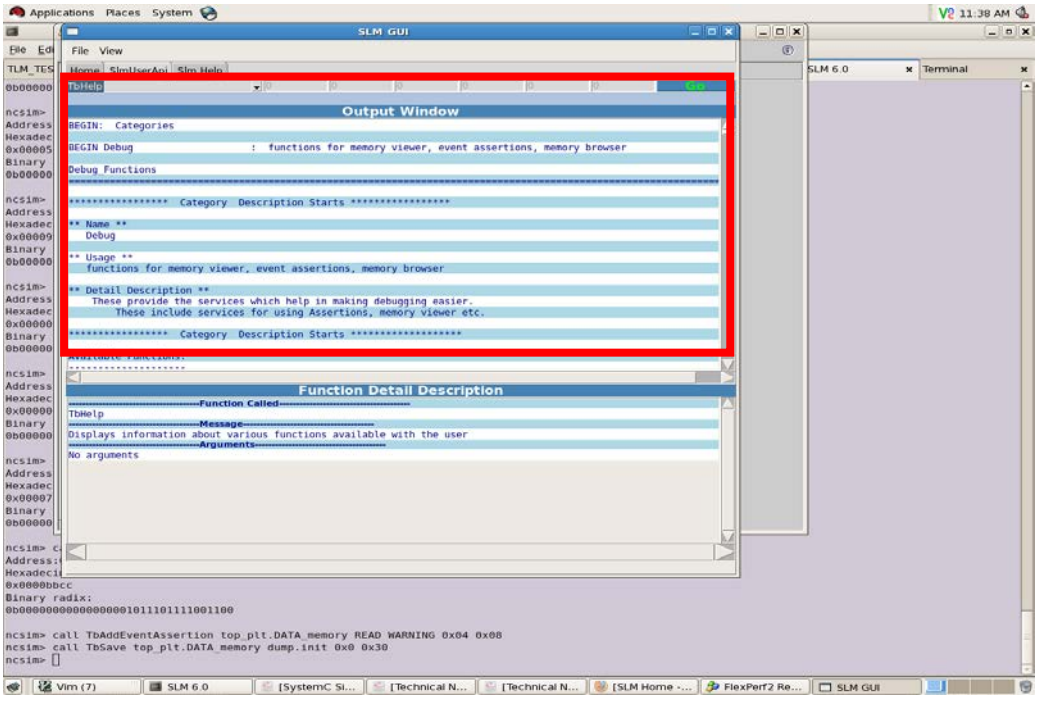

Fig. 7.16(a) Function Mapper Screen-1

Fig. 7.16(b) shows another mapping where record of parameter types is also maintained.

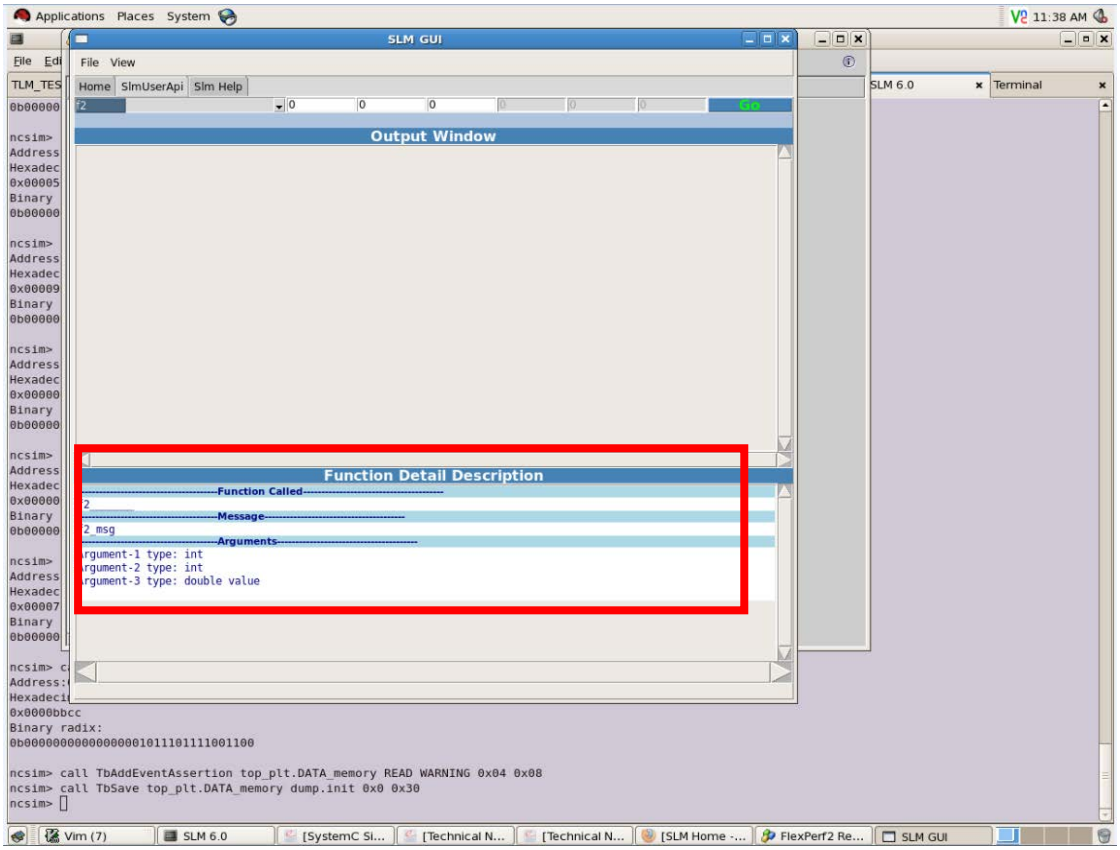

Fig. 7.16(b) Function Mapper Screen-2

Fig. 7.17 shows another feature of offline SLM Help which is still under development.

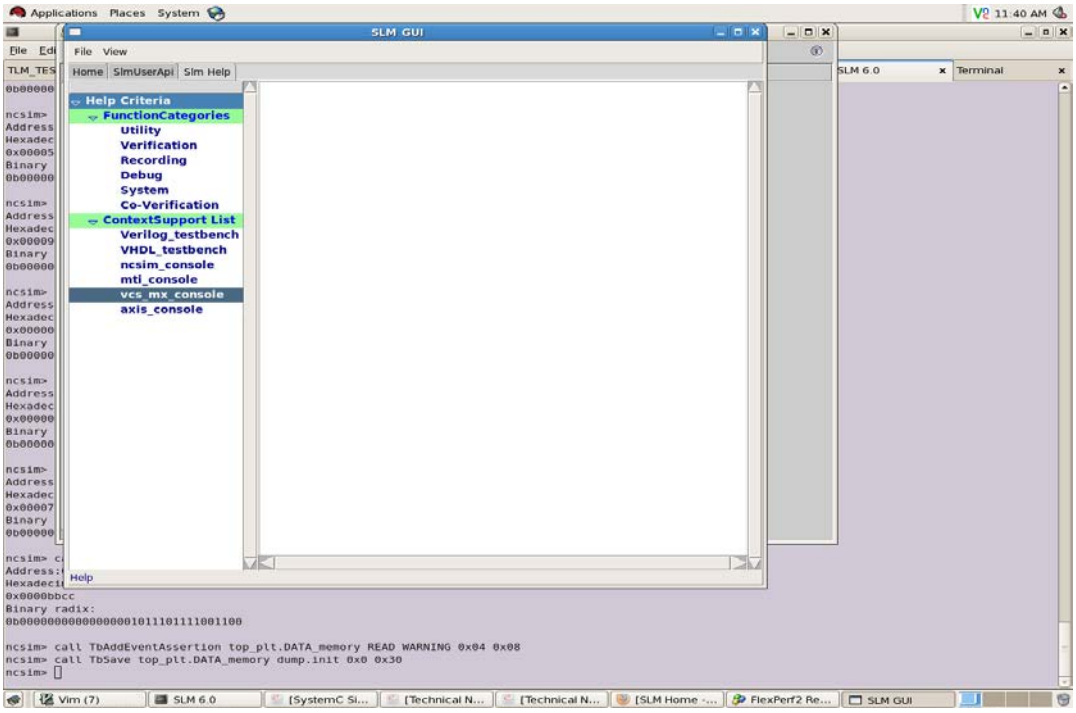

Fig. 7.17 Offline Help [In process of development]

Fig. 7.18 shows memory buffer of SLM instance. Here in display the buffer of memory ‘sc_main.top_plt.demo_csw_inst_0.demo_inst_0.slave_M0’ is displayed.

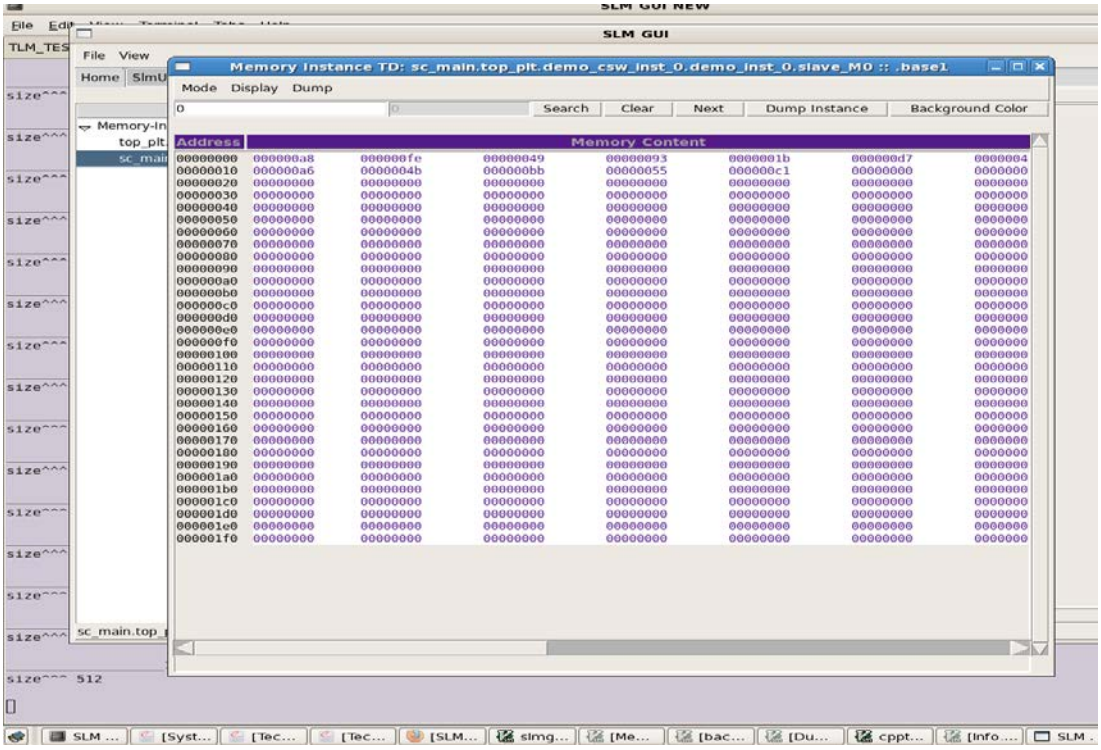

Fig. 7.18 Memory Grid

Fig. 7.19(a), 7.19(b) and 7.19(c) show the runtime logs of simulation in GUI mode. Fig. 7.19(a) shows various stages in GUI initialization.

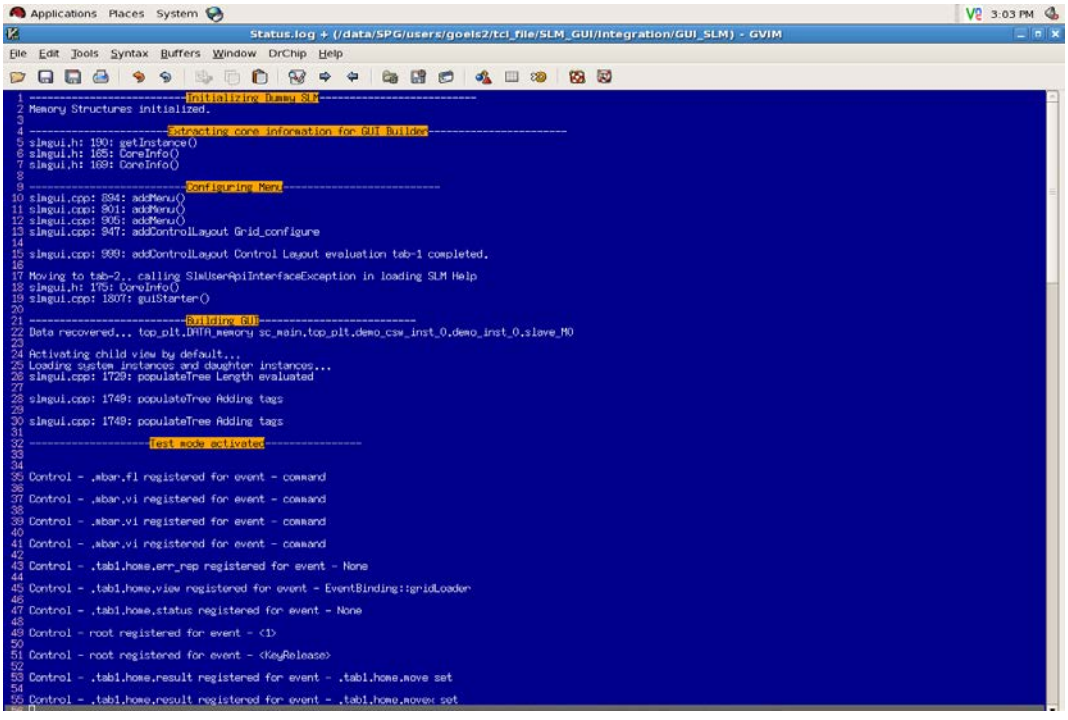

Fig. 7.19(a) GUI Running Log-1

Fig. 7.19(b) and Fig. 7.19(c) show the logs of Binding Table when operated GUI in testing mode.

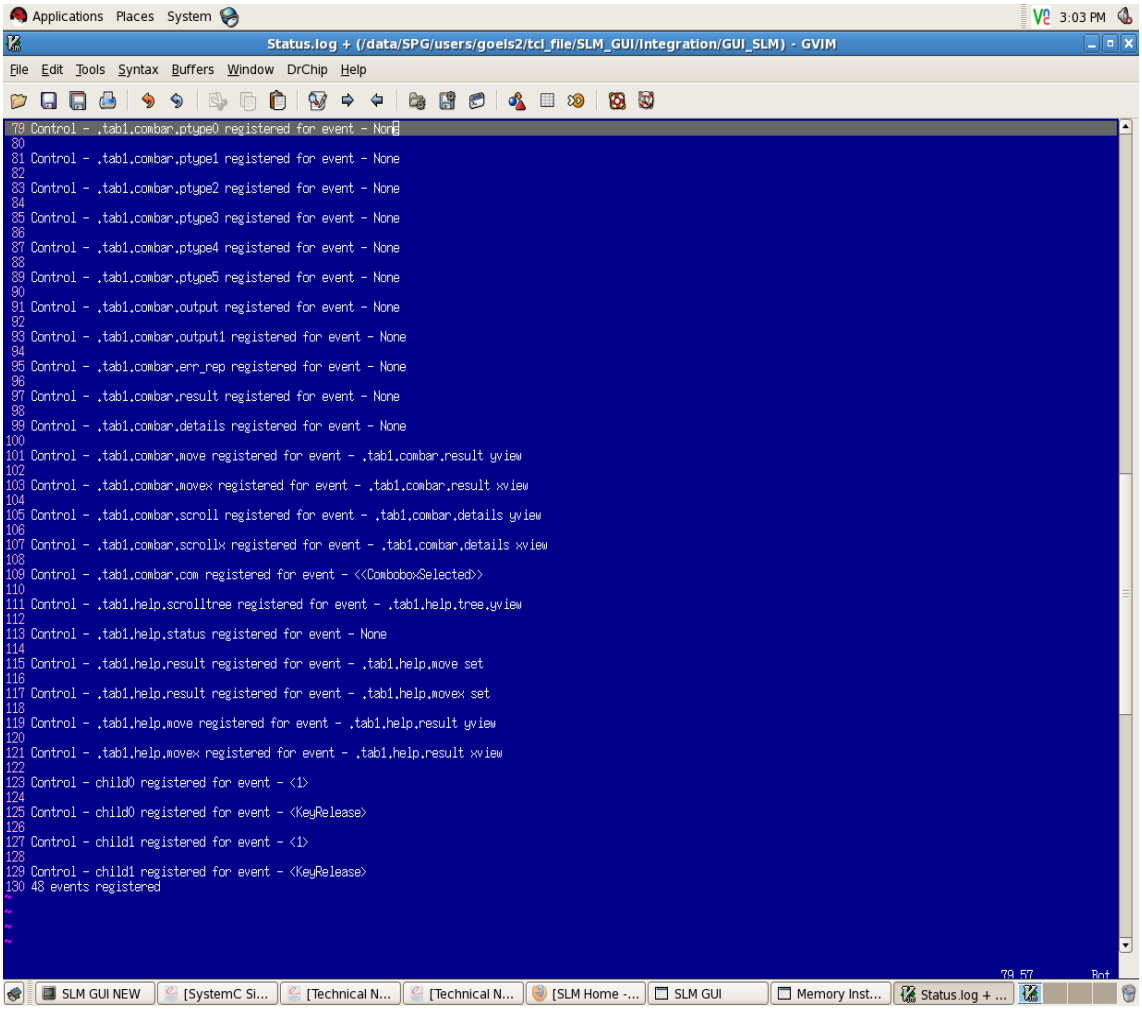

Fig. 7.19(b) GUI Running Log-2

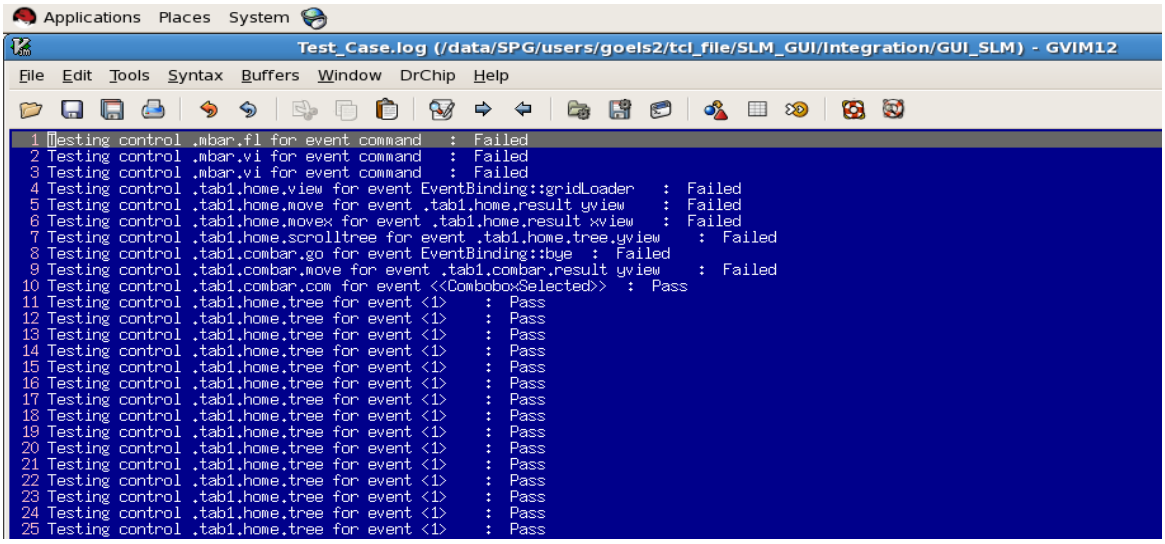

Fig. 7.19(c) GUI Running Log-3

2. SLM Build

Once GUI is build independent of SLM, SLM Core need to build too. The resulting libraries will thus be used during elaboration phase to access SLM Services. Fig. 7.20(a) and 7.20(b) shows build process terminal and resulting set of libraries.

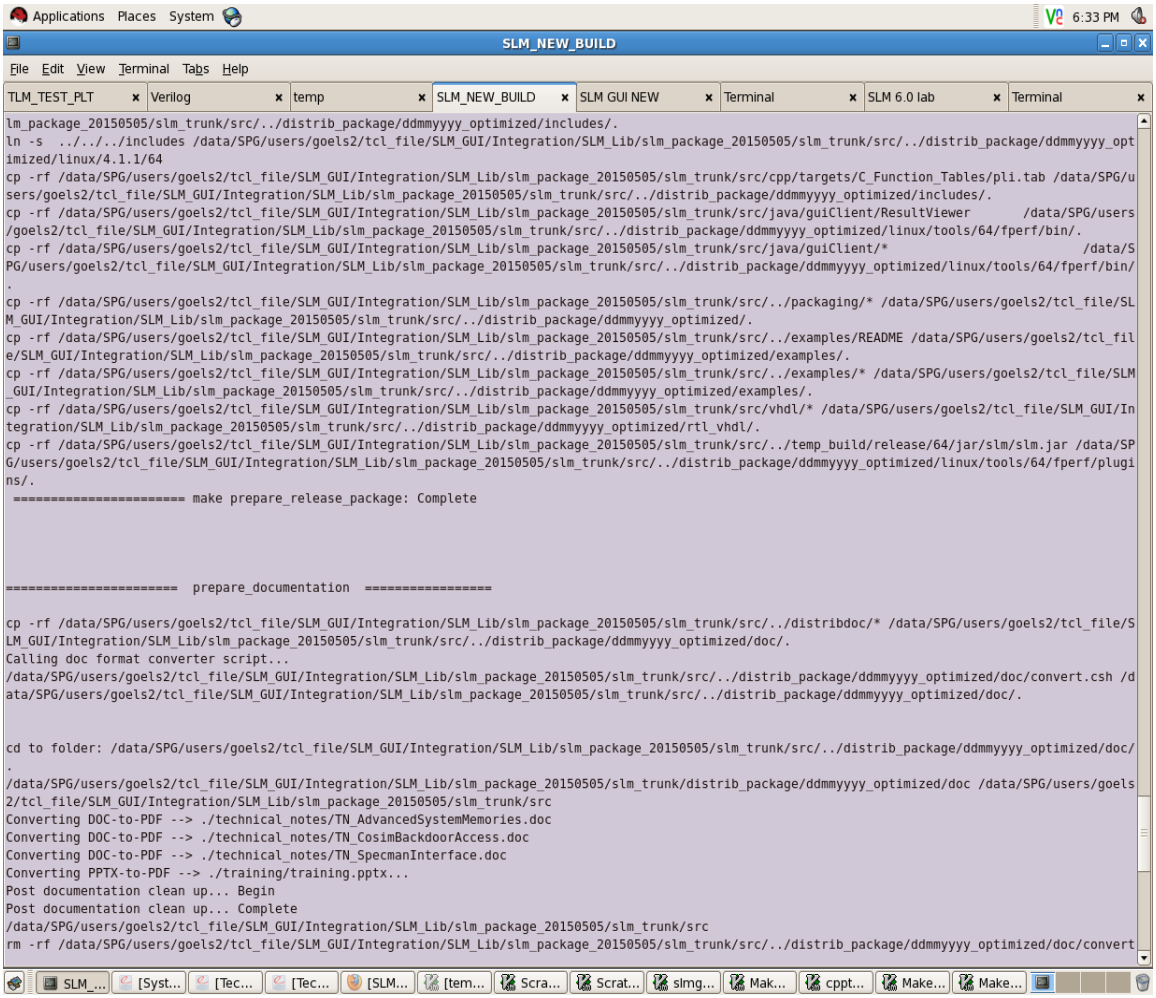

Fig. 7.20(a) SLM Build Terminal Output

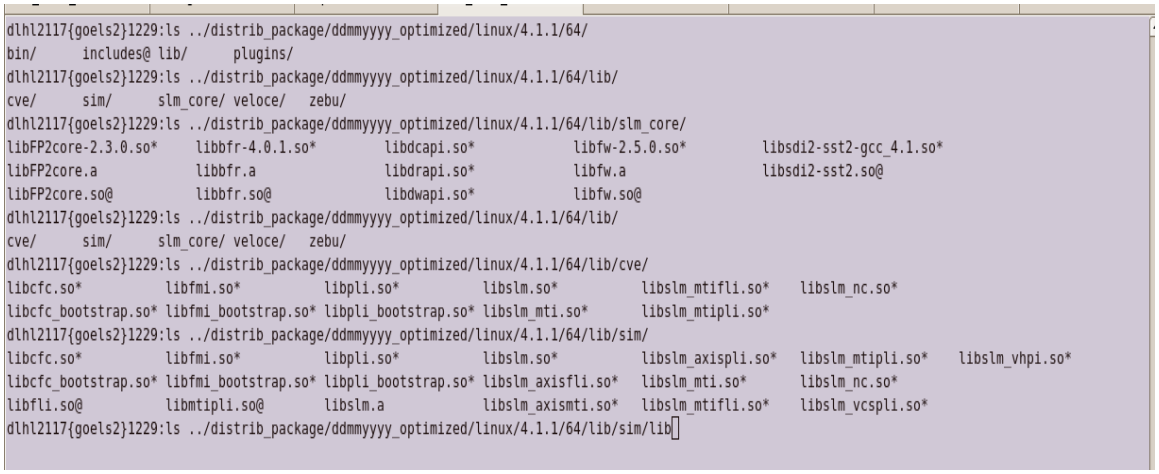

Fig. 7.20(b) Set of SLM shared libraries

3. TLM_Mode

Fig. 7.21(a) and 7.21(b) shows the SLM run in TLM Mode, i.e., purely C++ environment.

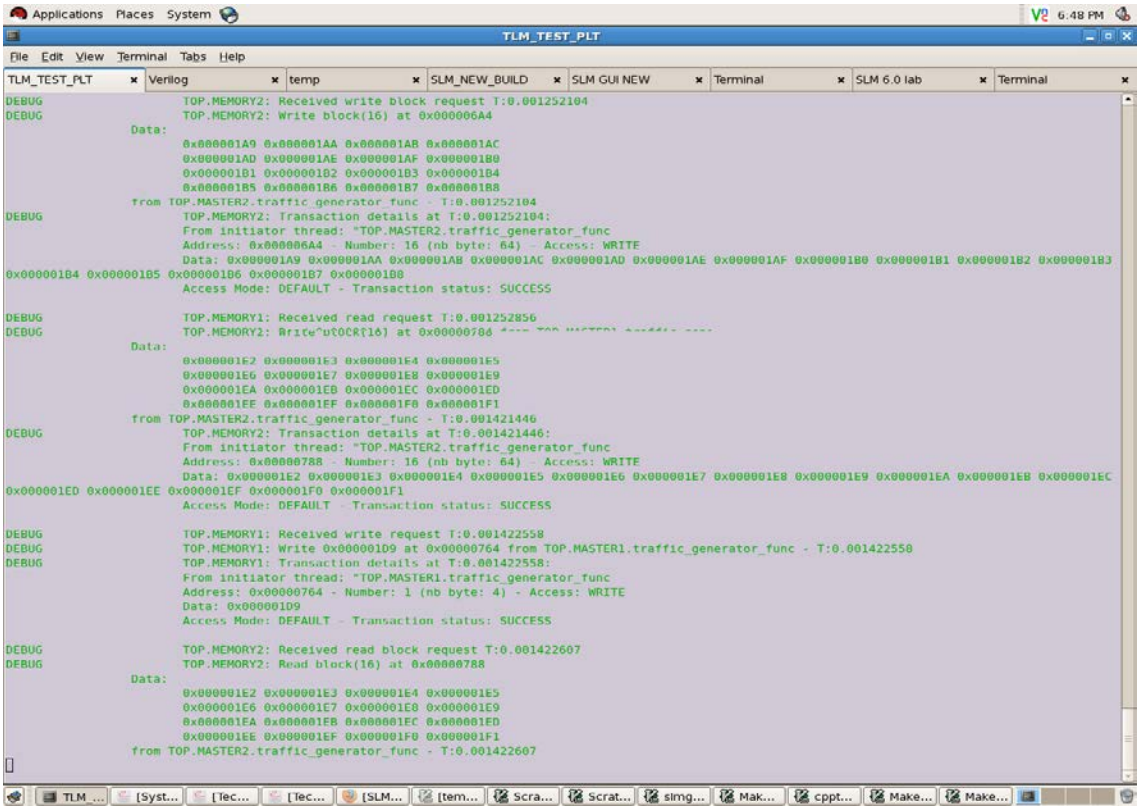

Fig. 21(a) TLM Run-1

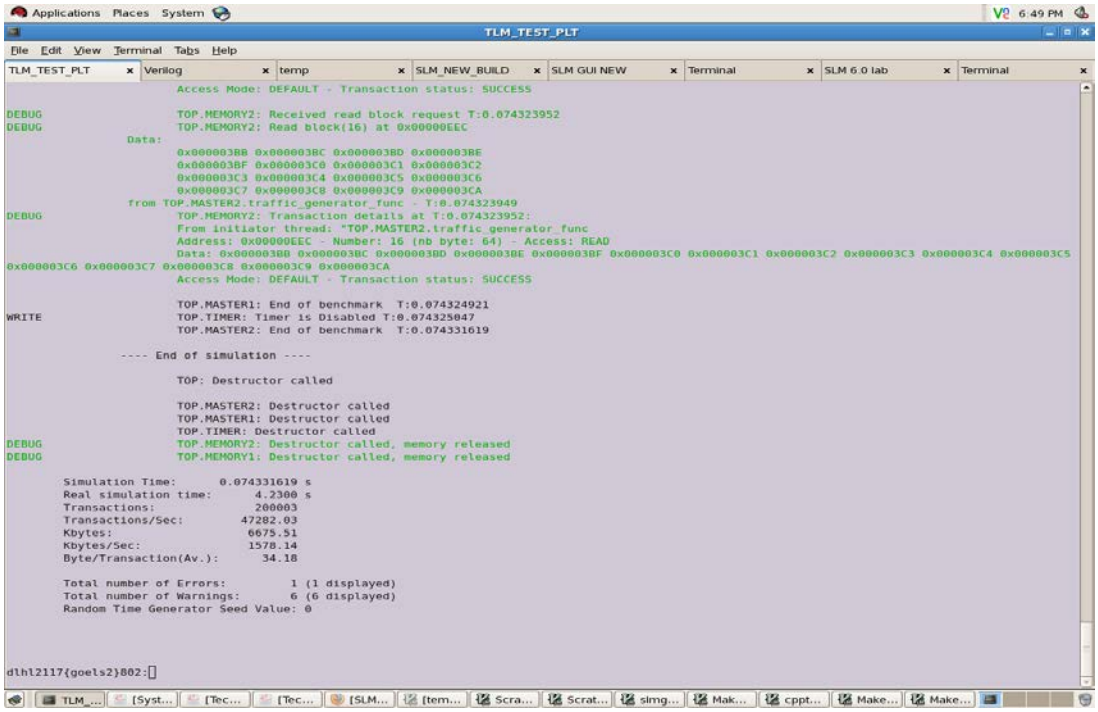

Fig. 7.21(b) TLM Run-2

4. Co-Sim Mode with ncsim

Fig. 7.22 shows the SLM run starting in Co-Sim Mode, i.e., purely mix HDL/C++ environment. In Fig. 7.22, the simulator used is ncsim, and it is started on remote server through lsf in GUI mode. Fig. 7.23 is the actual ncsim interface to be operated.

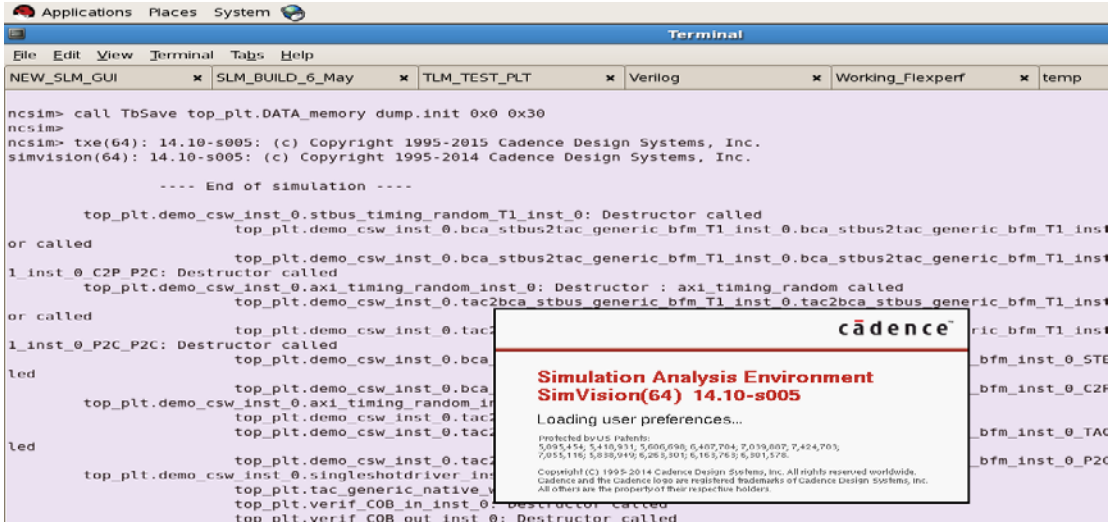

Fig. 7.22 Ncsim GUI Initialization

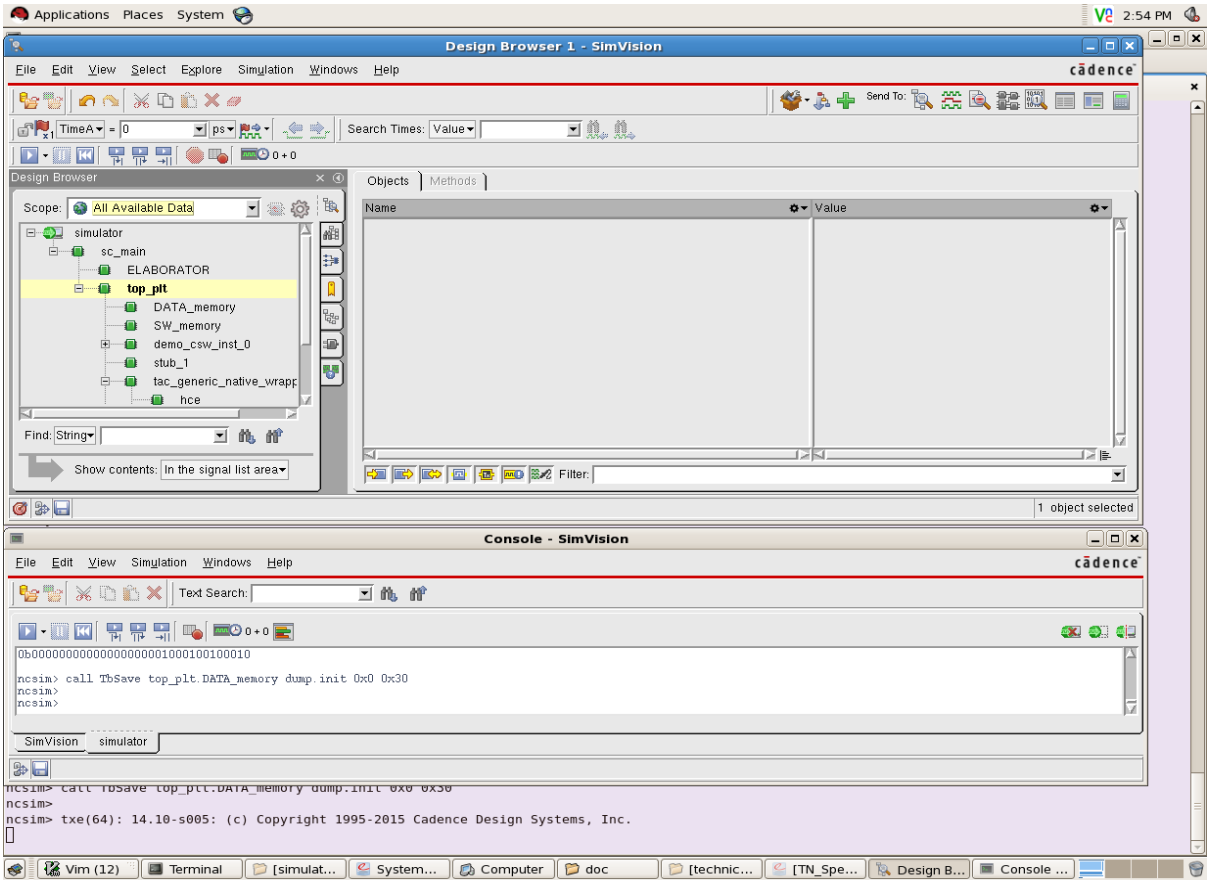

Fig. 7.23 Ncsim Design Browser

Fig. 7.24 shows the SimVision Console in ncsim, through which SLM Services are invoked from tcl prompt.

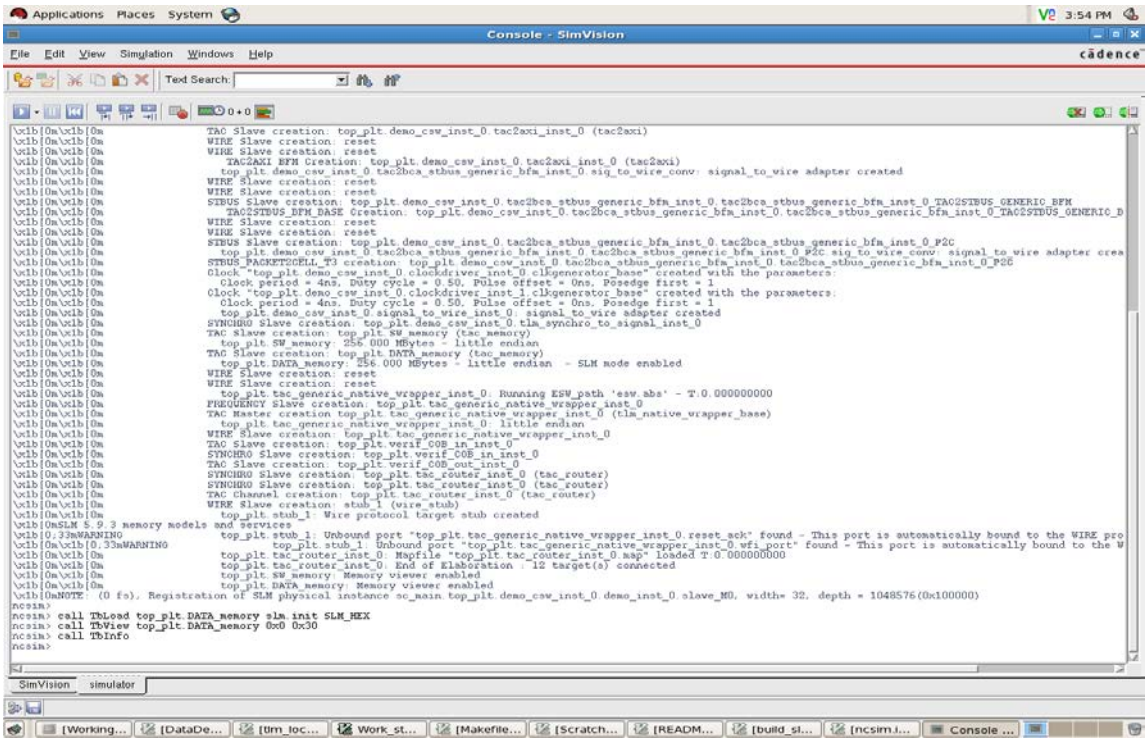

Fig. 7.24 SimVision Console

Fig. 7.25(a), Fig. 7.25(b), Fig. 7.25(c) shows memory registration, access to SLM Services, and GUI access in Co-Sim mode, respectively.

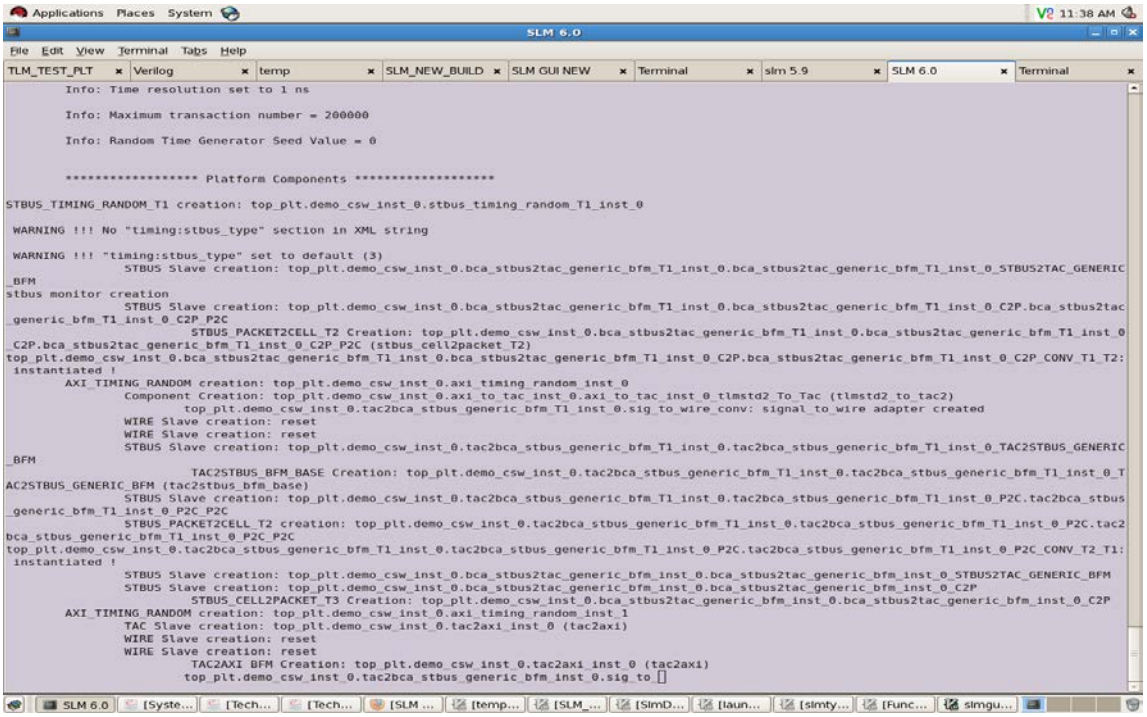

Fig. 7.25(a) Ncsim Simluation using SLM-View 1

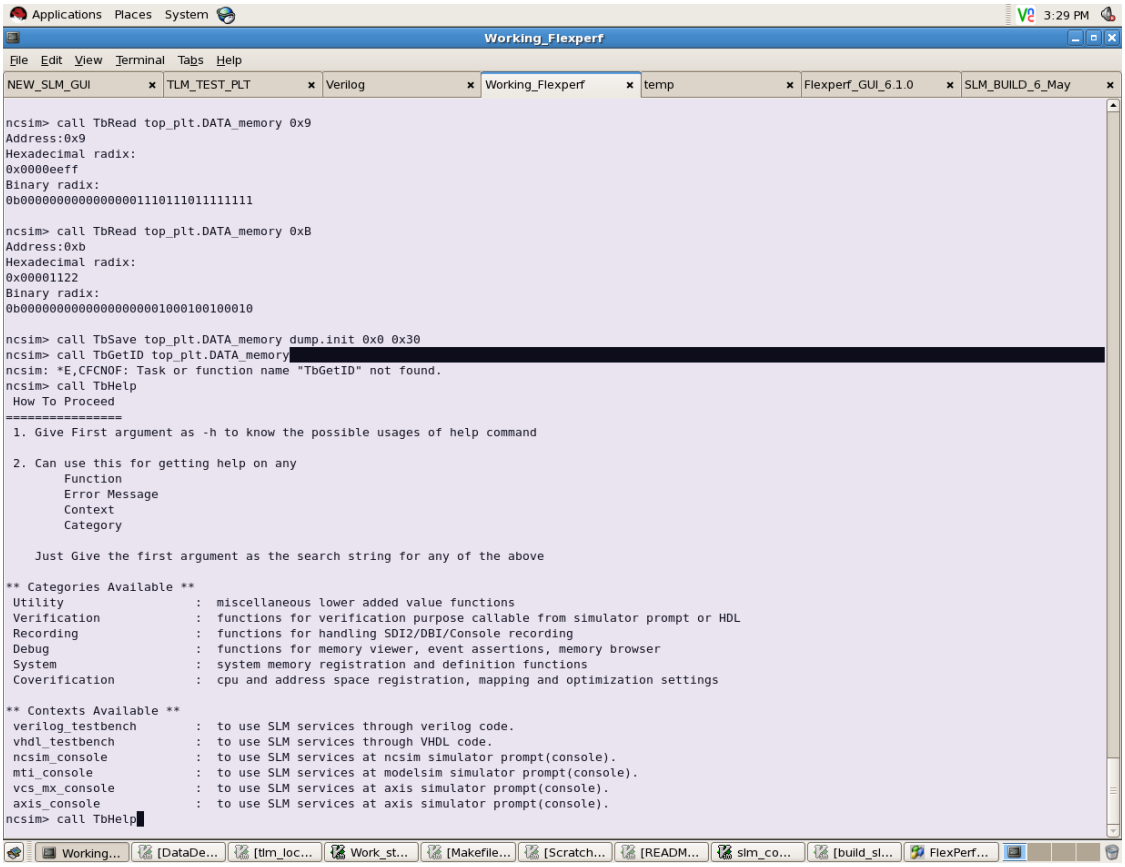

Fig. 7.25(b) Ncsim Simluation using SLM - View 2

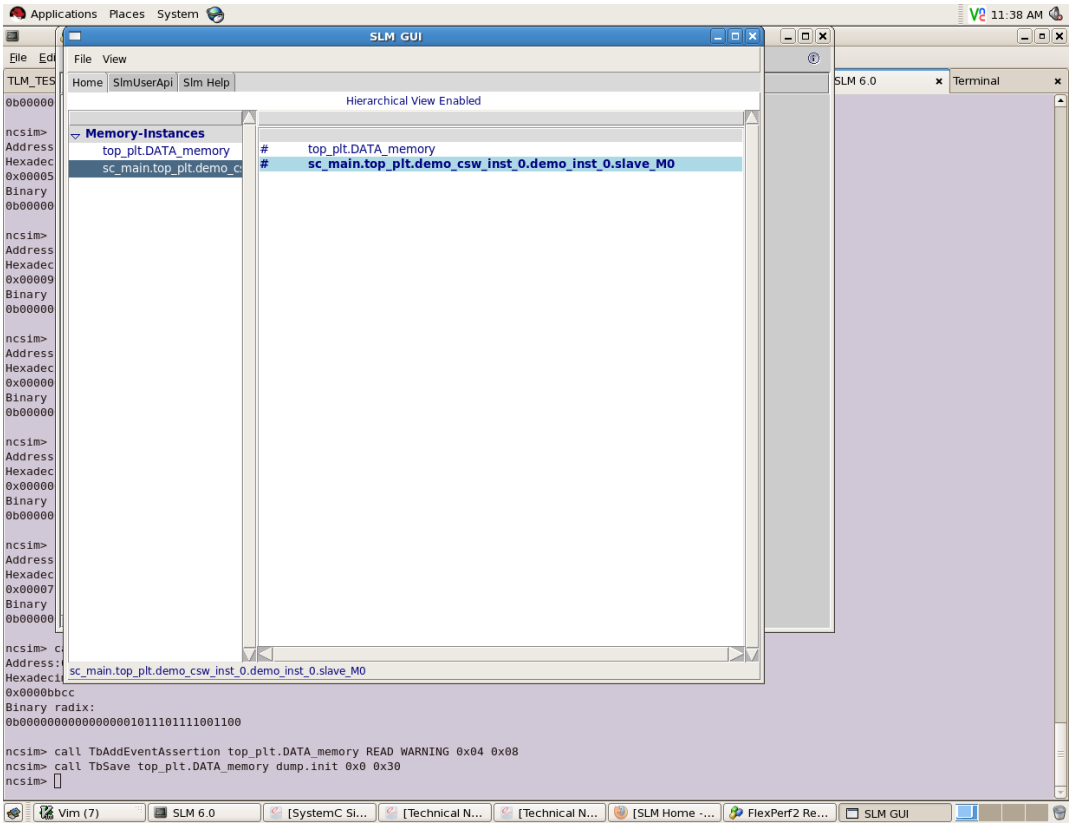

Fig. 7.25(c) Ncsim Simluation using SLM – View 3

7.1.4 Extracted Information

A subset of memory information extracted from SLM Core included memory buffer and its attributes. Fig. 7.26 shows the attributes of various SLM instances and Fig. 7.27 shows the memory buffer of one of those instances.

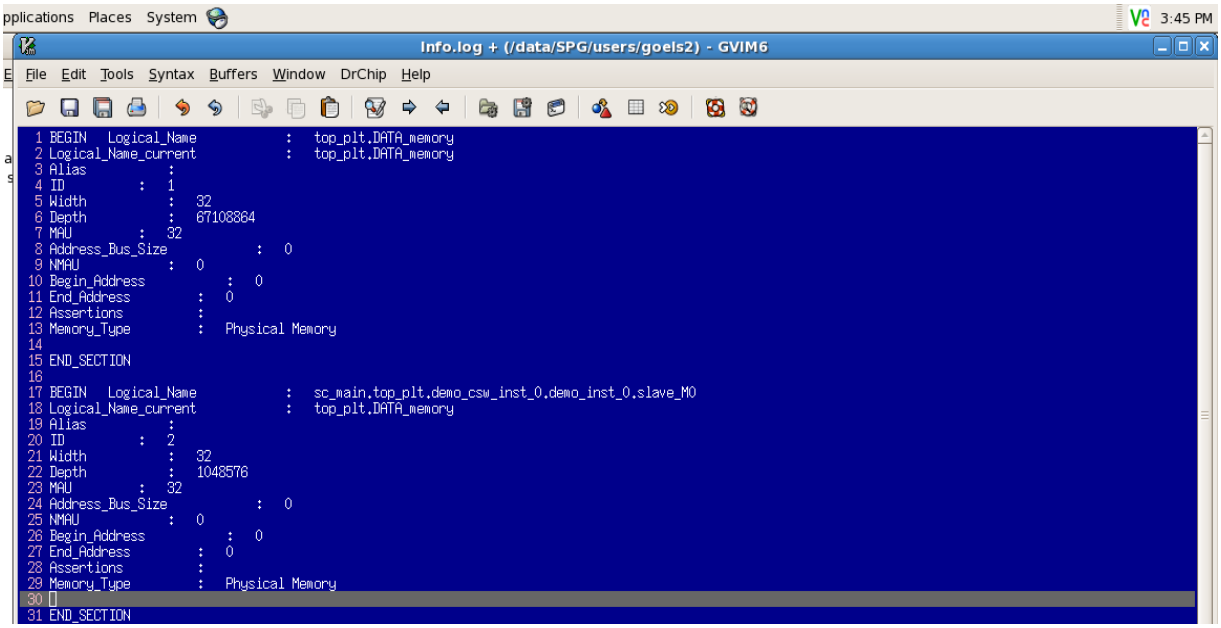

Fig. 7.26 Memory Attributes

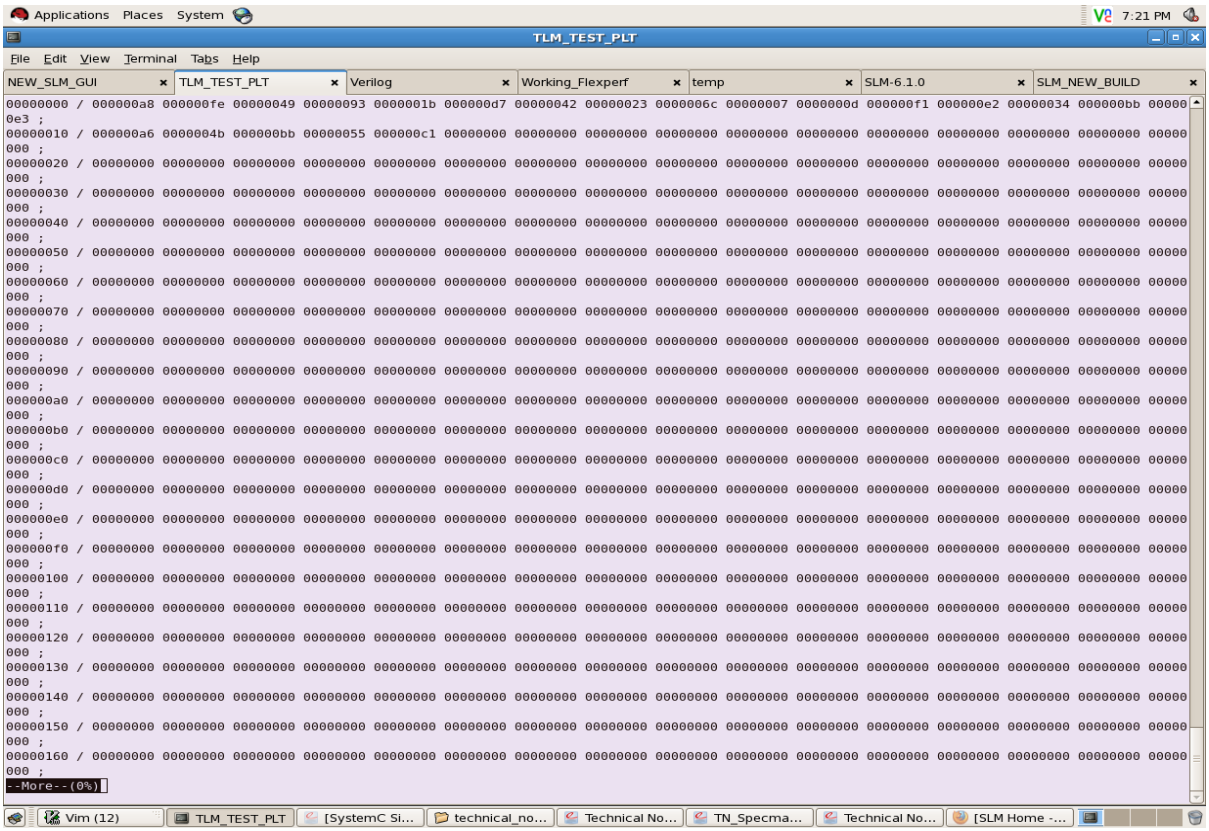

Fig. 7.27 Memory Buffer

7.1.5 GUI Testing Process

A brief view of <control, event> registerations, their logs, invocation and traversal by GUI test suite are presented below in Fig. 28(a), 28(b) , 28(c) and 28(d) respectively.

```
void BindingTablesTestInfoSet::registerEvents(std::string
control_name, std::string event)
{
      BindingTableEntry bt = {control_name,event};
      table.entry[BT_SIZE]=bt;
      BT_SIZE = BT_SIZE + 1;
```

Fig. 28(a) Event Registerations

```
void EventBinding::logRegisteredEvents()
{
      BindingTablesTestInfoSet::BindingTable table =
bindHandler.getBindingTable();
      for(int i=0; i<bindHandler.BT_SIZE ;i++)
      {
            GUIGlobalInfoSet::status << "\nControl - "<<
table.entry[i].control <<" registered for event - " <<
table.entry[i].event << "\n";
      }

      GUIGlobalInfoSet::status << bindHandler.BT_SIZE <<" events
registered\n";
}
```

Fig. 28(b) Binding Table Logger

```
EventBinding::bindHandler.registerEvents( ".tab1.home.result" ,
".tab1.home.move set");
           EventBinding::bindHandler.registerEvents(
".tab1.home.result" , ".tab1.home.movex set");
```

Fig. 28(c) Binding Function Call

```
void dfsControlNameTraversal(std::string root_name)
            {
                  std::vector<std::string> v2 =
eval(std::string("winfo children ")+root_name);
                  if(v2.size()==0)
                  {
                        std::cout << "Leaf "<< root_name <<"\n";
                        dfsEventNameTraversal(root_name);
                  }
                  for(unsigned int i=0;i<v2.size();i++)
                  {
                              std::cout << "Vector "<< i <<" : =
" <<v2[i] <<"\n";
                              dfsEventNameTraversal(root_name);
                  }
                  for(unsigned int i=0;i<v2.size();i++)
                  {
                        dfsControlNameTraversal(v2[i]);
                  }
            }
```

Fig. 28(d) DFS traversal

7.1.6 Directory Structure

Fig. 29(a) and 29(b) depicts the directory structure of GUI.

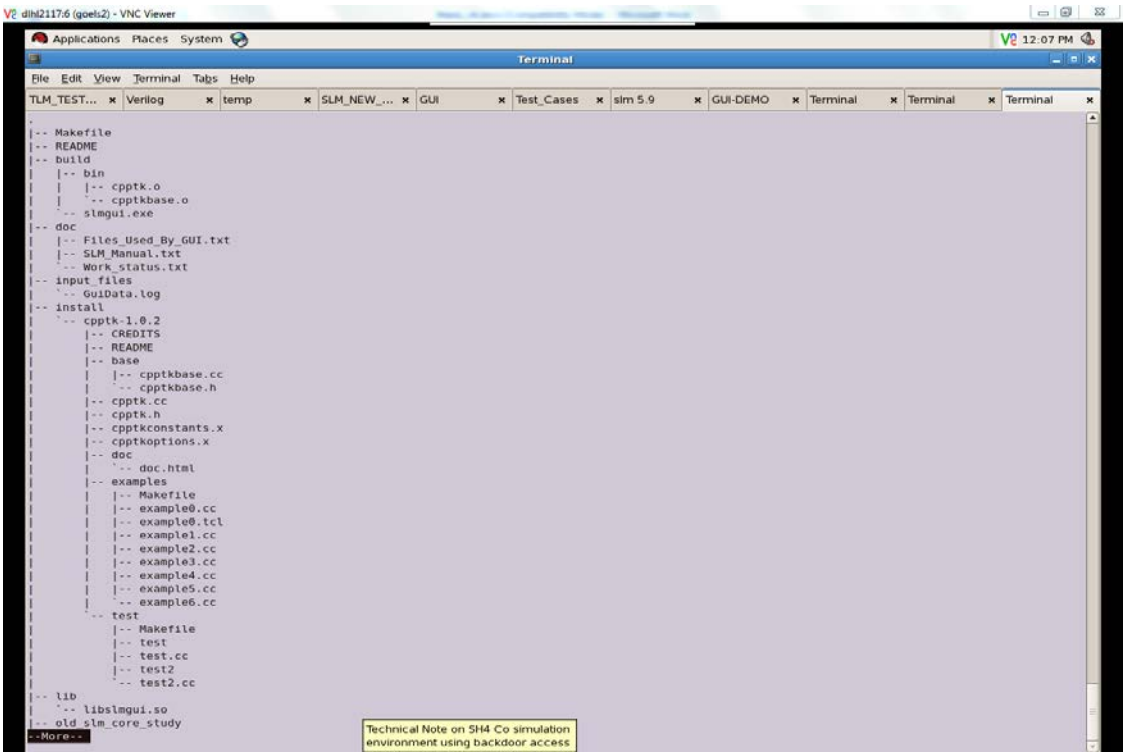

Fig. 7.29(a) Directory structure of GUI

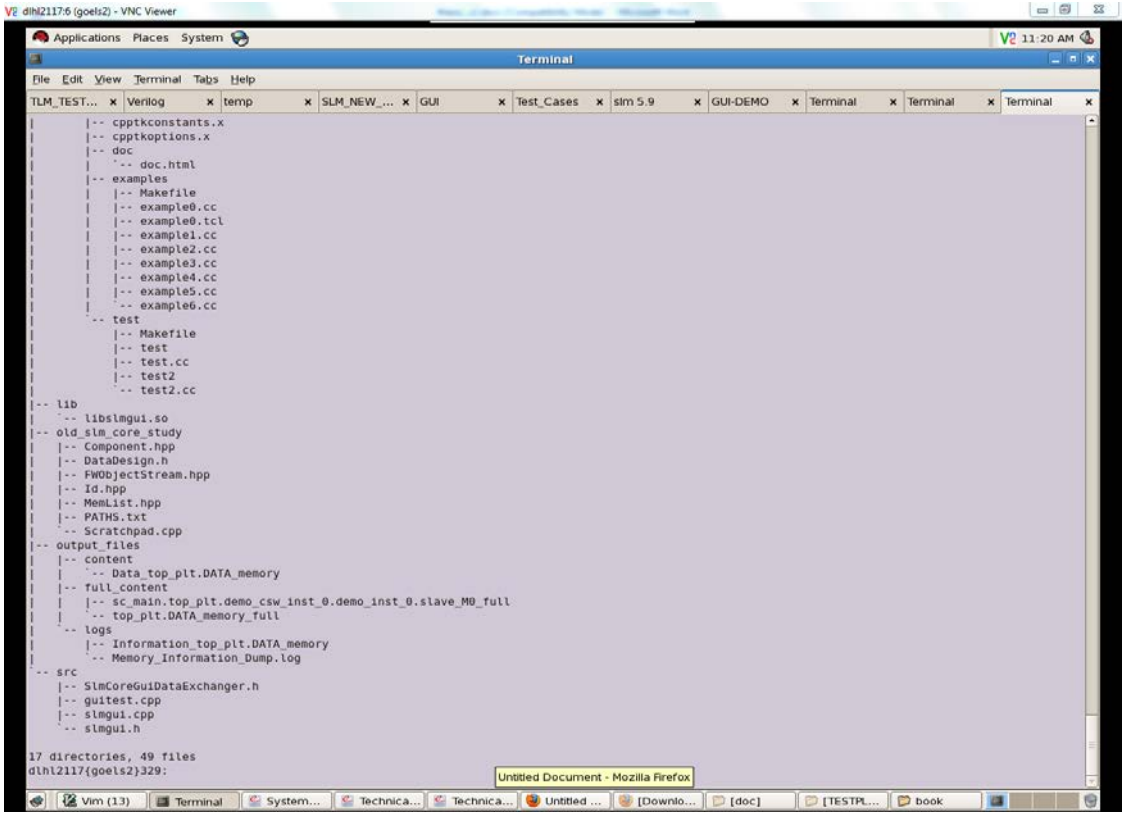

Fig. 7.29(b) Directory structure of GUI

7.2 SLM Test Suite

7.2.1 Environment Setup

It reqiures a set of configuration done through makefile. A compiled view of required setup is presented in Fig. 7.30(a), 7.30(b) and 7.30(c).

```
#!/bin/tcsh

        # setenv SLM_HOME to <LOCAL_PATH>/distrib/ddmmyyyy/
        # setenv GNUVERSION <VERSION>
        # setenv SLM_BITS    64
        # source $SLM_HOME/SLM.csh

```

Fig. 7.30(a) Required Environment Variables

```
source $SLM_HOME/SLM.csh

source /data/SPG/users/krishnak/projects/3_SLM/SLM_TRUNK/slm/examples/setup.csh

echo "------------------------------------------------------------------------------"
echo $SLM_HOME
echo $GNUVERSION
echo $SLM_BITS
echo "------------------------------------------------------------------------------"
```

Fig. 7.30(b) Required scripts for configuration setup

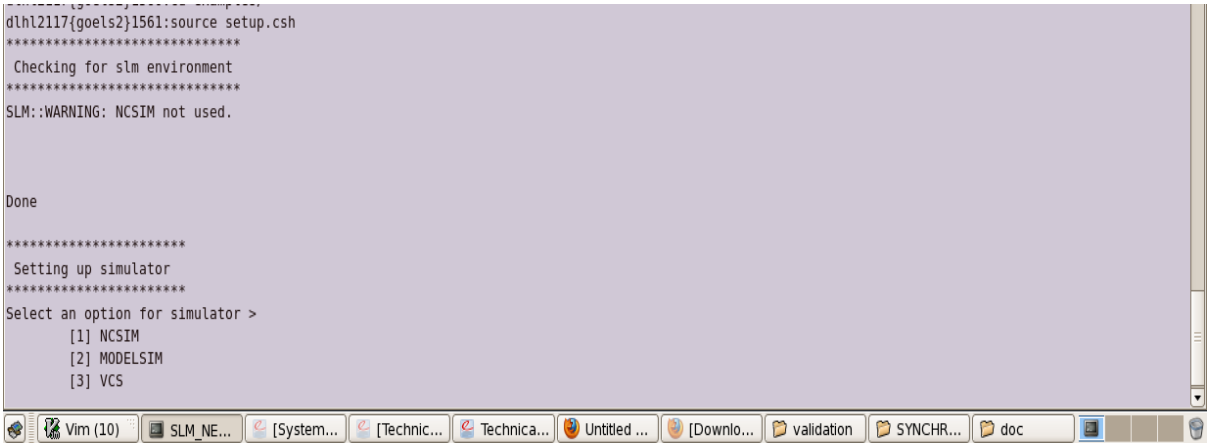

Fig. 7.30(c) SLM Setup

Fig. 31(a) and Fig. 31(b) depicts build process for tests.

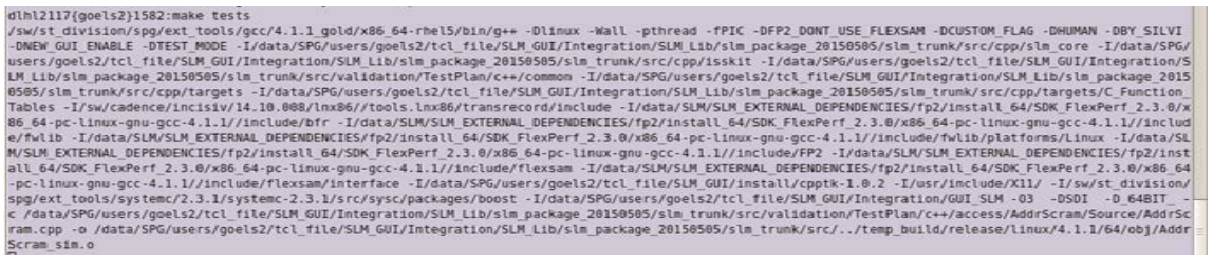

Fig. 7.31(a) Build Tests

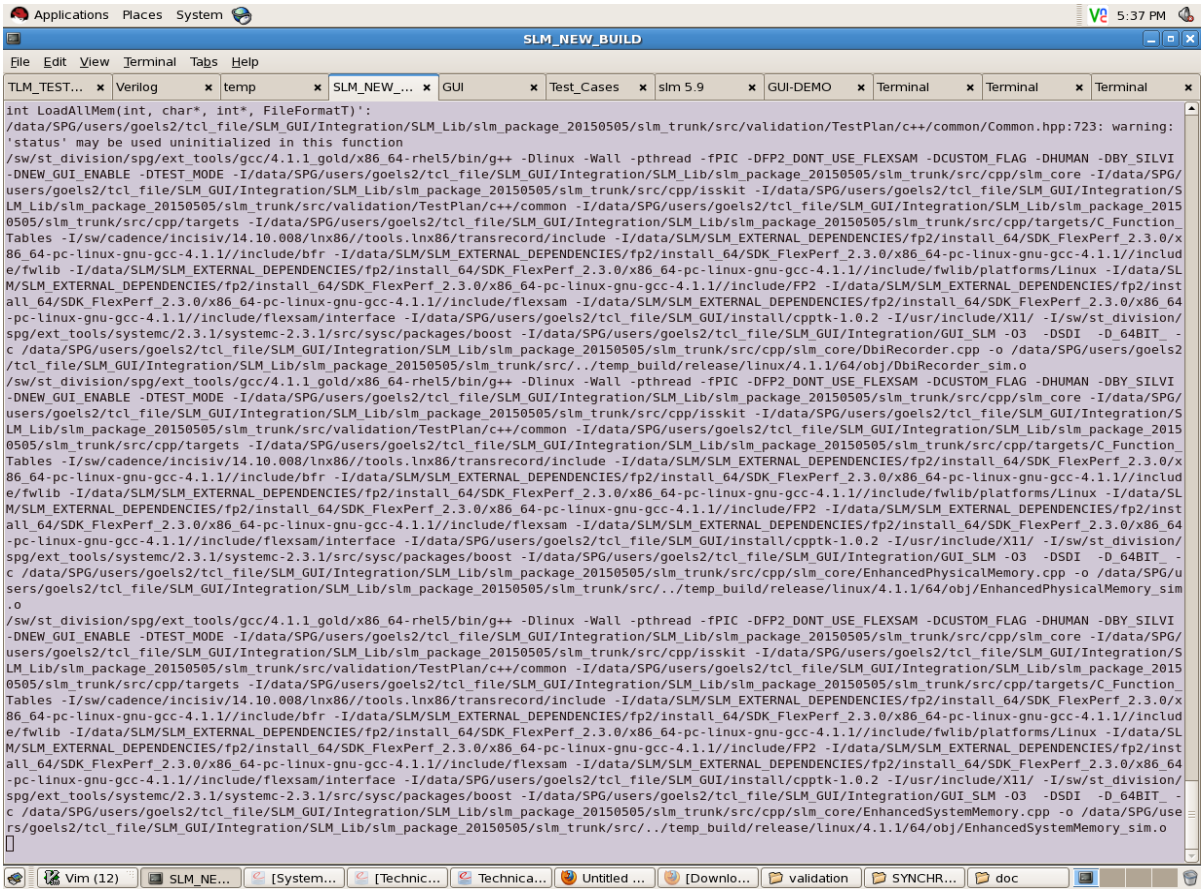

Fig. 7.31(b) Build Tests

7.2.2 Running Status of Test Cases

Fig. 7.32 shows the current running status of test cases. Many of them are working, but some are not currently working.

```
 82 ====================
 83 1.access
 84 ====================
 85
 86 AddrScram                    FAIL            FAIL            FAIL            FAIL
 87 MemRead                     PASS            PASS            PASS            PASS
 88 MemLoadMemory               PASS            PASS            PASS            PASS
 89 MemMaskRead                 PASS            PASS            PASS            PASS
 90 MemMaskWrite                PASS            PASS            PASS            PASS
 91 MemSilent                   PASS            PASS            PASS            PASS
 92
 93 ====================
 94 2.debug
 95 ====================
 96
 97 CfcFli                      FAIL            FAIL            FAIL            FAIL
 98 SysMemEventAssertion        PASS            PASS            PASS            PASS
 99 TbEventAssertions           PASS            PASS            PASS            PASS
100 TbSnapshot                  FAIL            FAIL            FAIL            FAIL
101
102 ====================
103 3.utility
104 ====================
105
106 TbError                     FAIL            PASS            FAIL            PASS
107 TbFunctions                 FAIL            ----            FAIL            PASS
108 TbHelp                      FAIL            FAIL            FAIL            FAIL
109
110 ====================
111 4.verification
112 ====================
113
114 EnhSysMemTbAccessFunctions  FAIL            FAIL            FAIL            FAIL
115 SysMemTbAccessFunctions     FAIL            FAIL            FAIL            FAIL
116 TbAddressSpace              FAIL            FAIL            FAIL            FAIL
117 TbLoadMemoryRangeDirect     PASS            PASS            PASS            PASS
118 TbMaskedWrite               PASS            PASS            PASS            PASS
119 SysMemLoadEtc               FAIL            FAIL            FAIL            FAIL
120 TbAccessFunctions           PASS            PASS            PASS            PASS
121 TbEventAssertions           FAIL            ----            ----            ----
```

Fig. 7.32 Status of Test Cases [A subset of test-suite]

7.2.3 Test Cases Runtime logs

Following is a set of logs recorded during test case execution. Testrun script is written to conduct automated execution of tests. Testrun.sum marks the beginning of test-automation script.

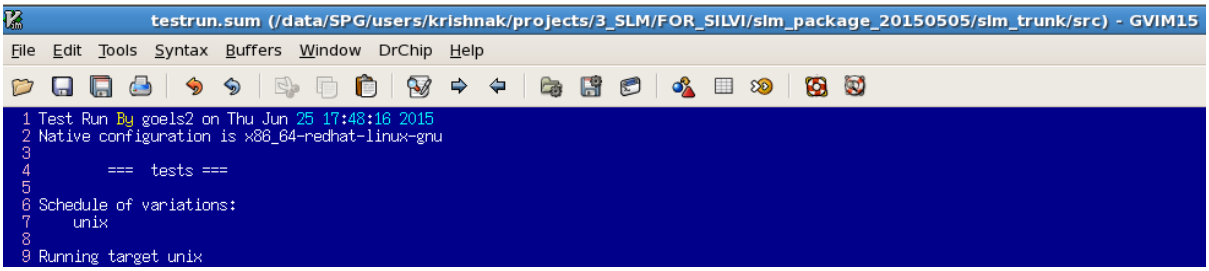

Fig. 7.33 Test Run Start

Fig. 7.34(a) and 7.34(b) depicts log and inputs of Adk test respectively.

Fig. 7.34(a) Adk Run

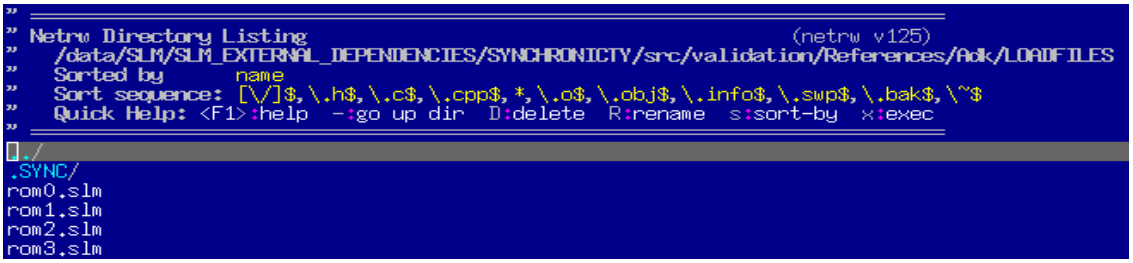

Fig. 7.34(b) Inputs to Adk Test

Fig. 7.35(a), 7.35(b), 7.35(c), 7.35(d) and 7.35(e) show the log of Assertions run. It checks various memories for different type of assertions such as Read, Write, etc.

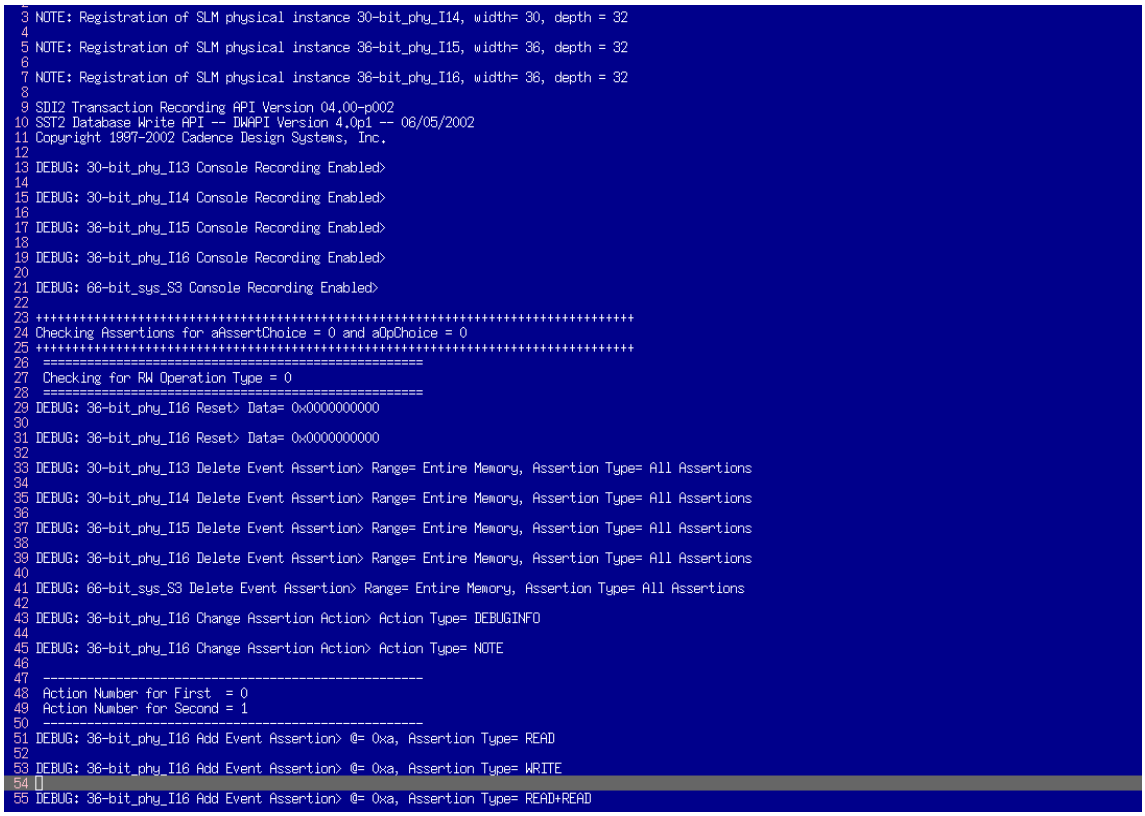

Fig. 7.35(a) Assertions Run

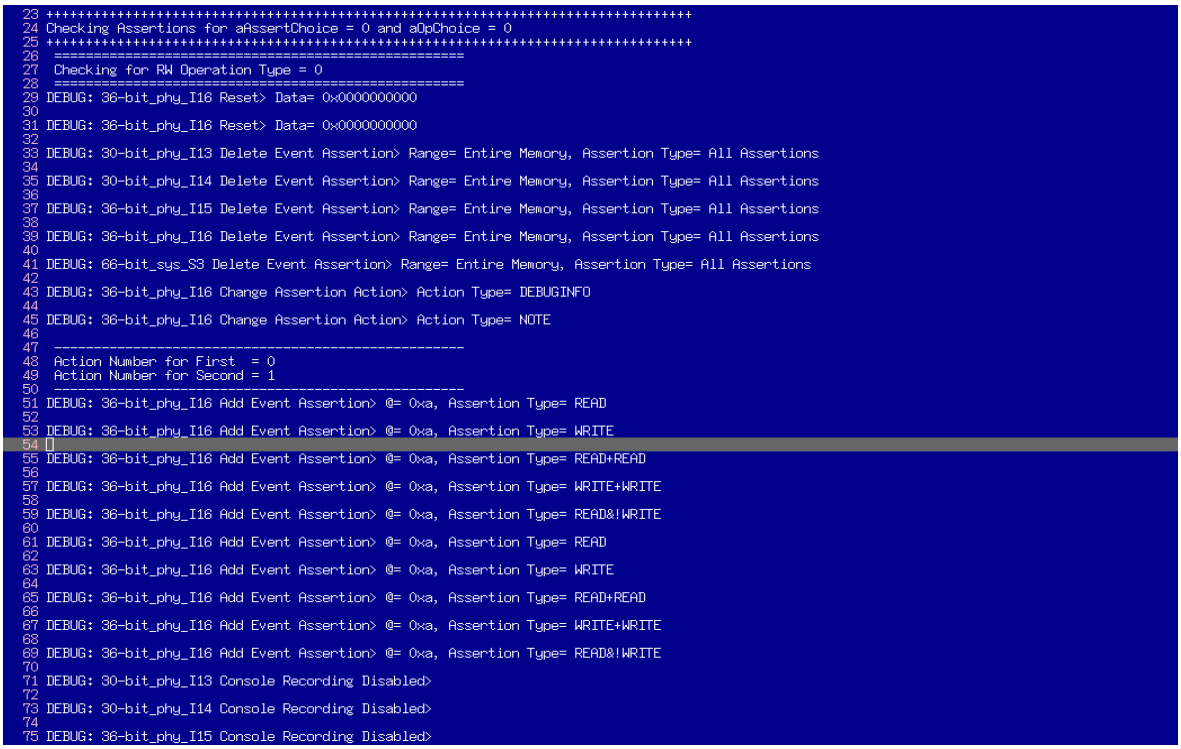

Fig. 7.35(b) Assertions Run

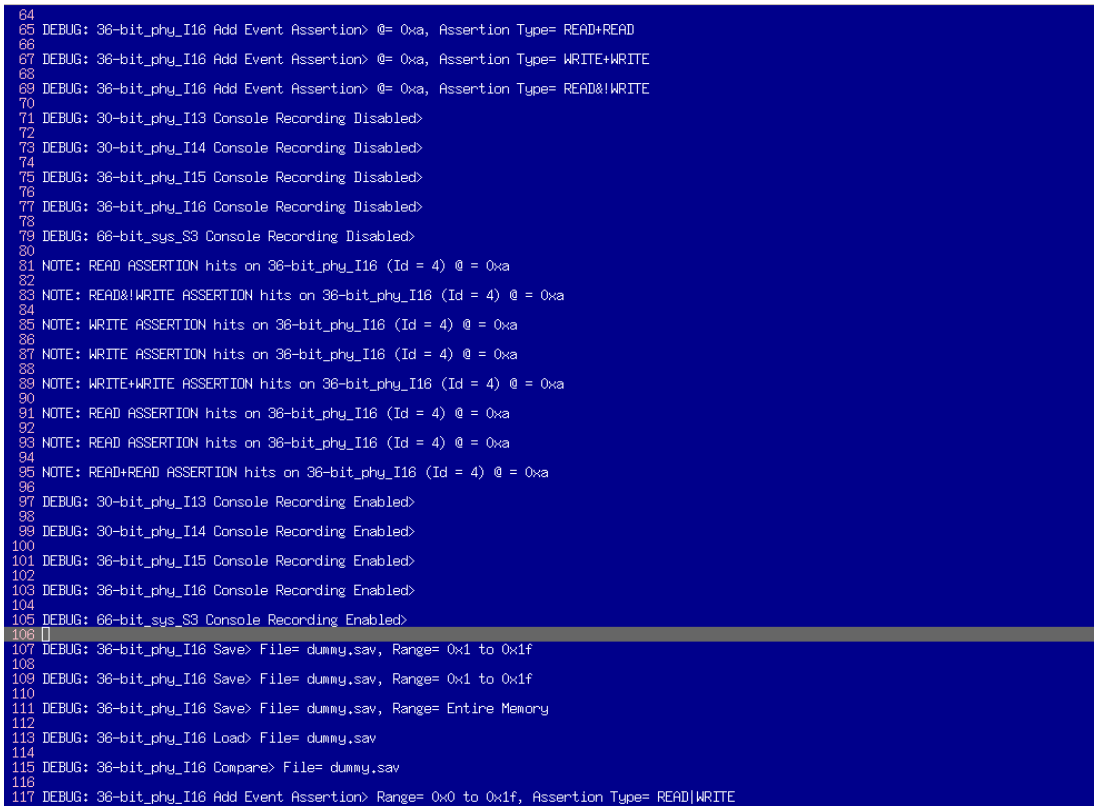

Fig. 7.35(c) Assertions Run

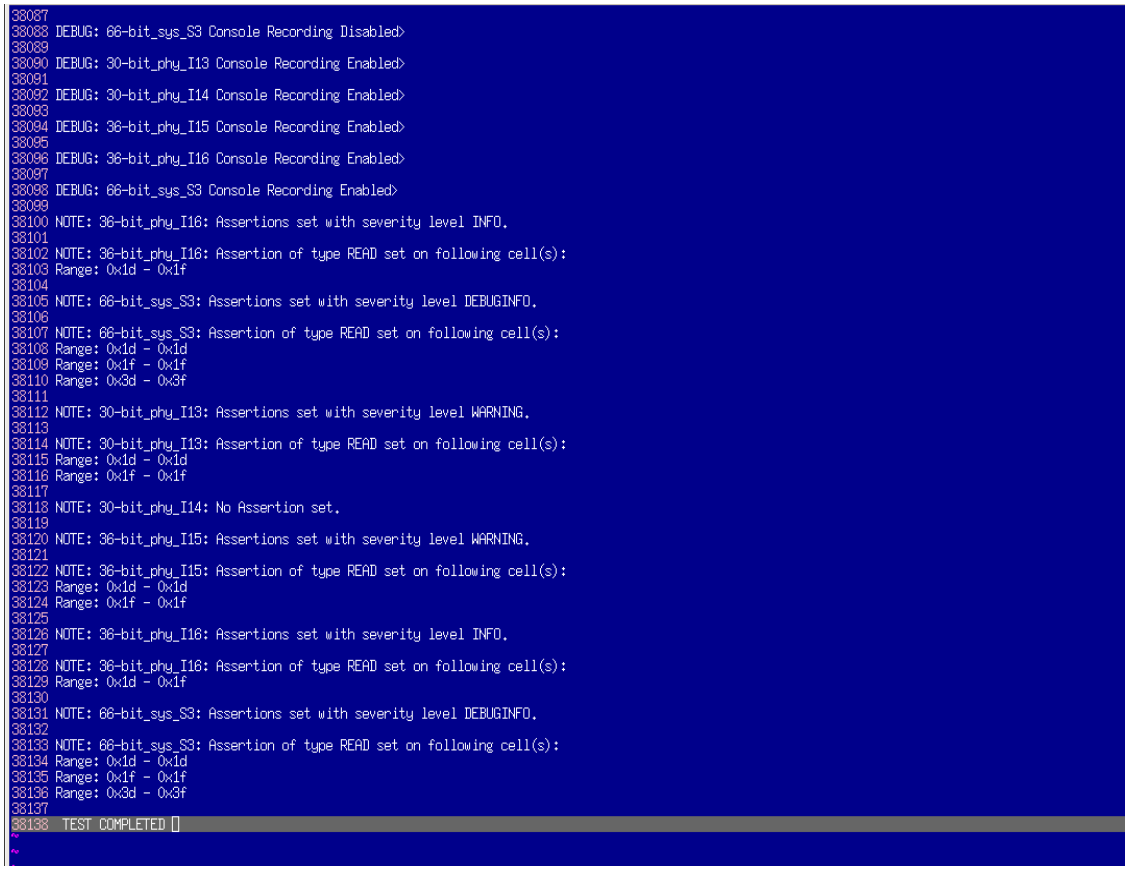

Fig. 7.35(d) Assertions Run

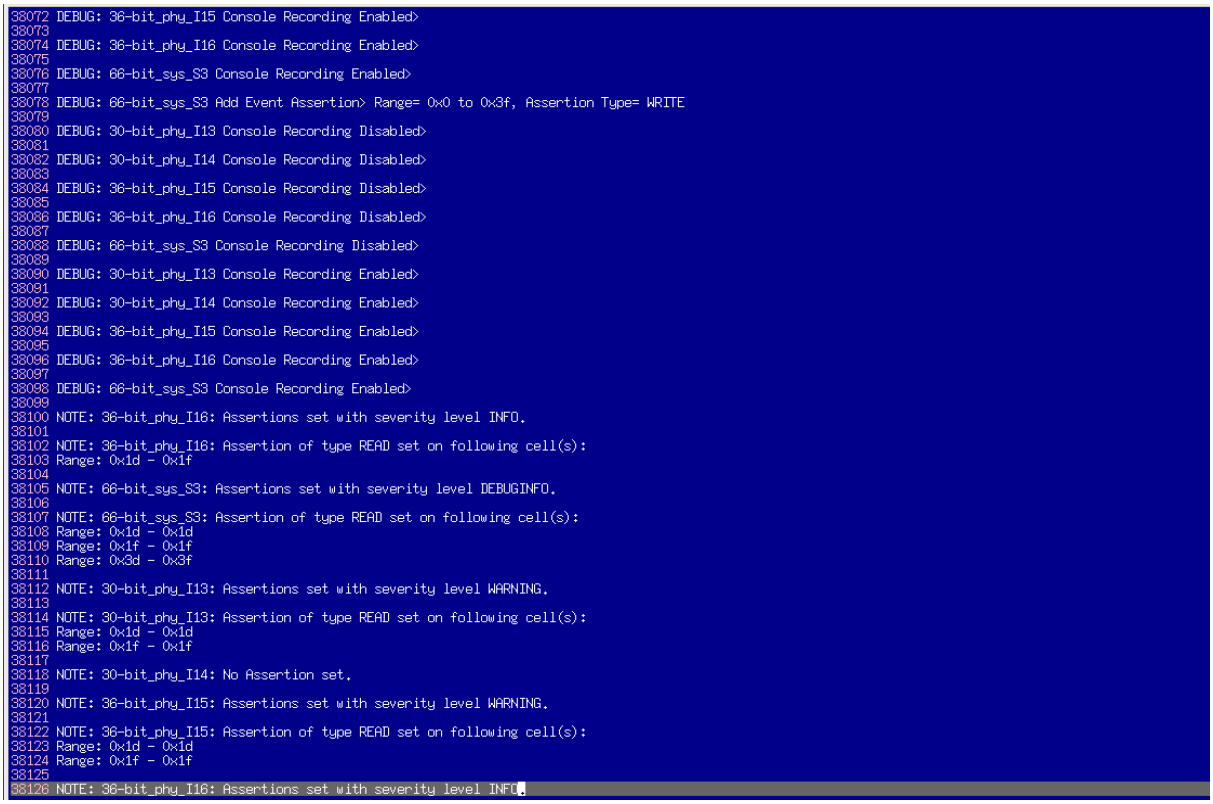

Fig. 7.35(e) Assertions Run

Fig. 7.36 depicts run-time log of Basic run test, where memory registration with varying types, widths and expansions are tested.

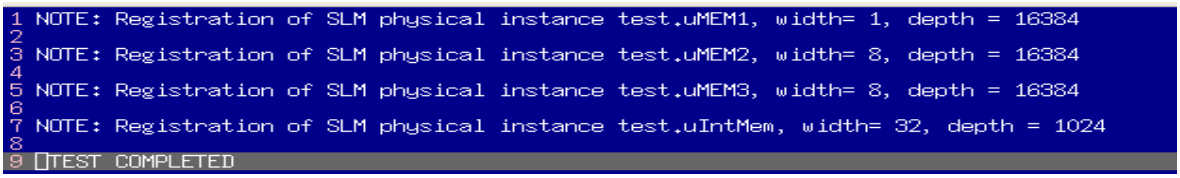

Fig. 7.36 Basic Run

Fig. 7.37 depicts run-time log of UtilServ run test, where memory registration with varying utilities, such as RAM, ROM, FIFO etc. are tested.

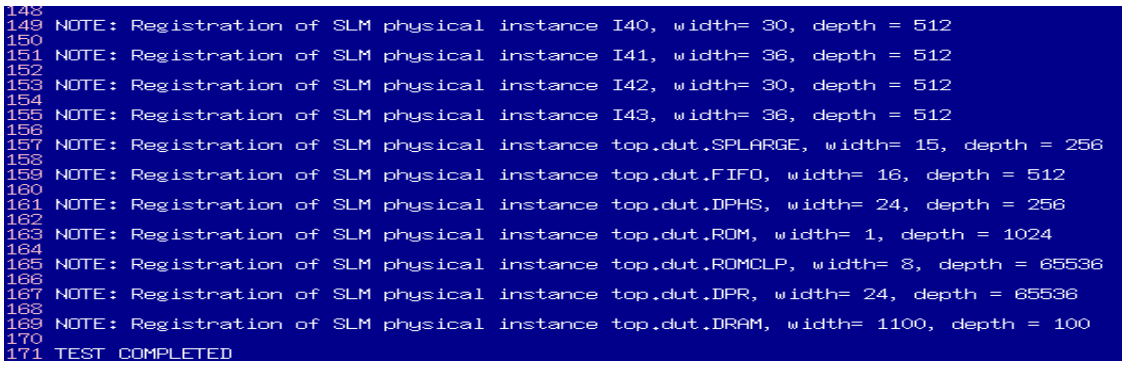

Fig. 7.37 UtilServ Test Run

Fig. 7.38(a) and 7.38(b) depicts run-time log of VerifServ where different sets of memory instances are tested for same verification services they provide.

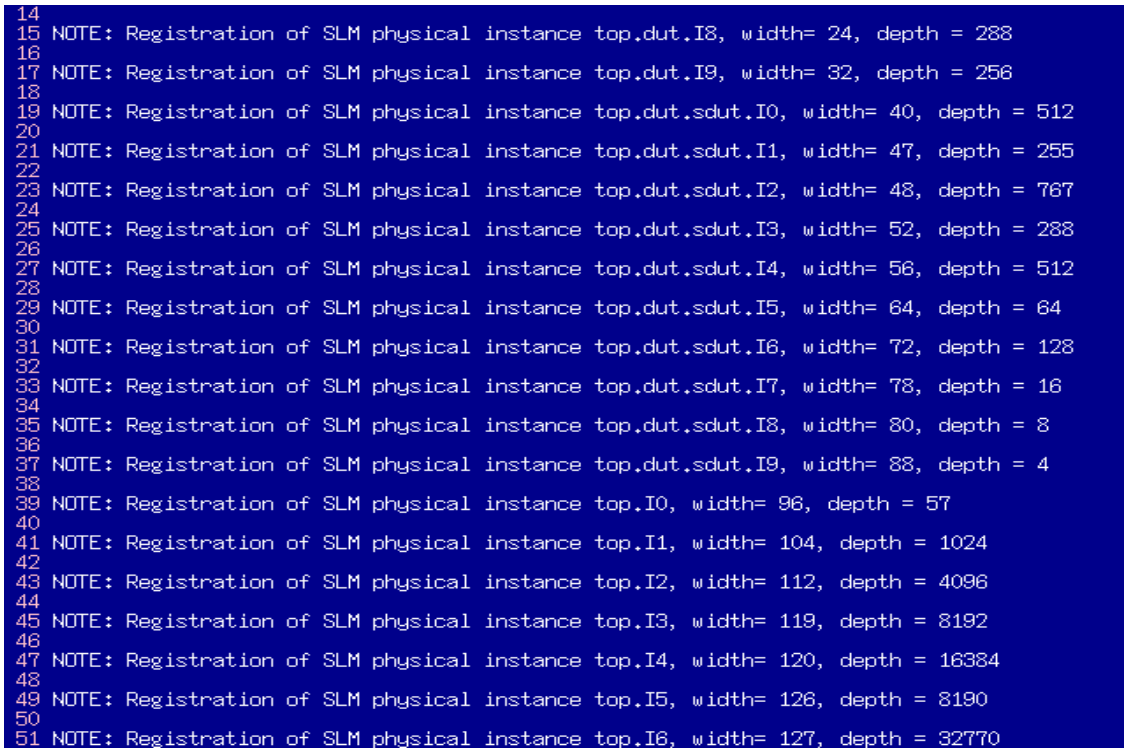

Fig. 7.38(a) VerifServ Test Run

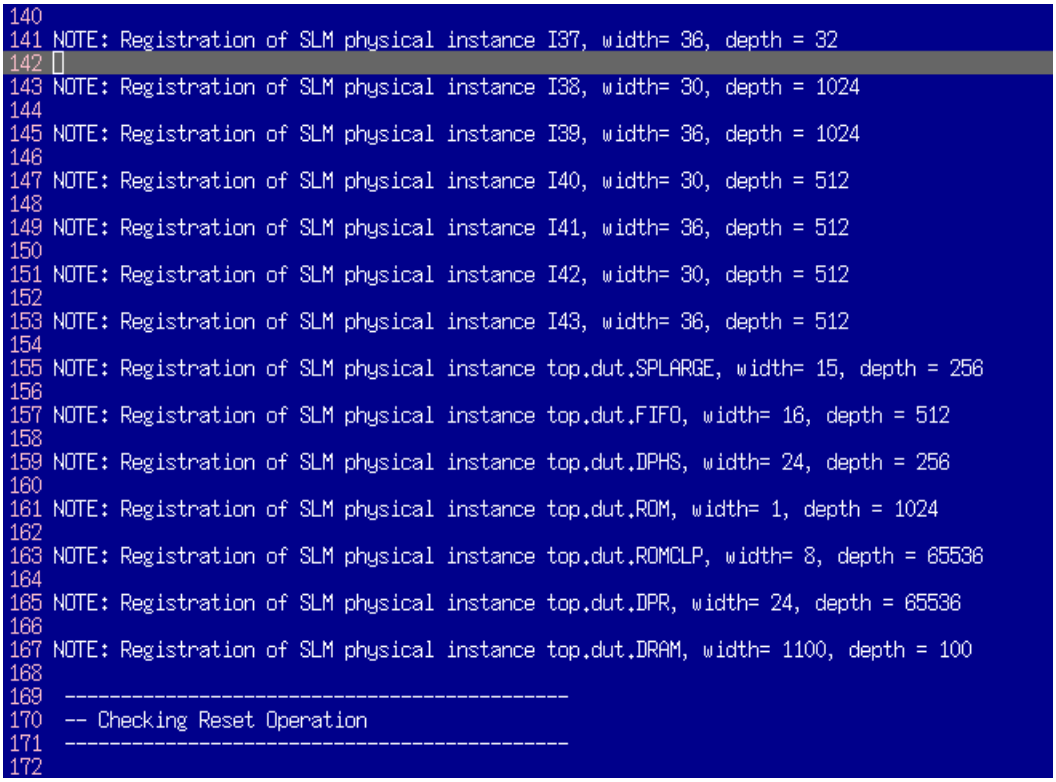

Fig. 7.38(b) VerifServ test log

7.2.4 SLM Inputs

Following files present a sample of inputs to SLM which acts as a common input for all types of simulators and platforms. They are later parsed into suitable formats by various users.

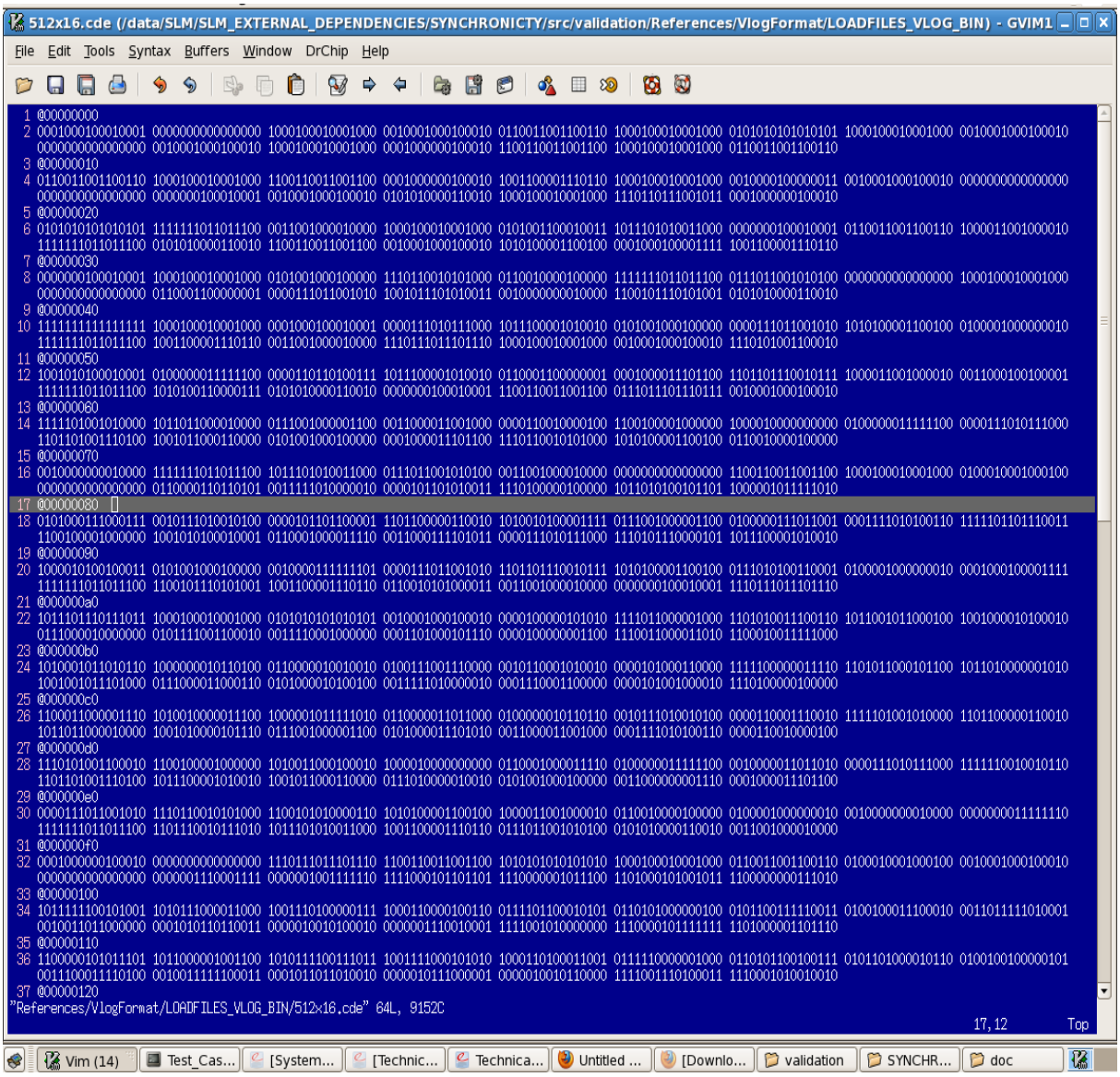

Fig. 7.39(a) 512x16.cde input

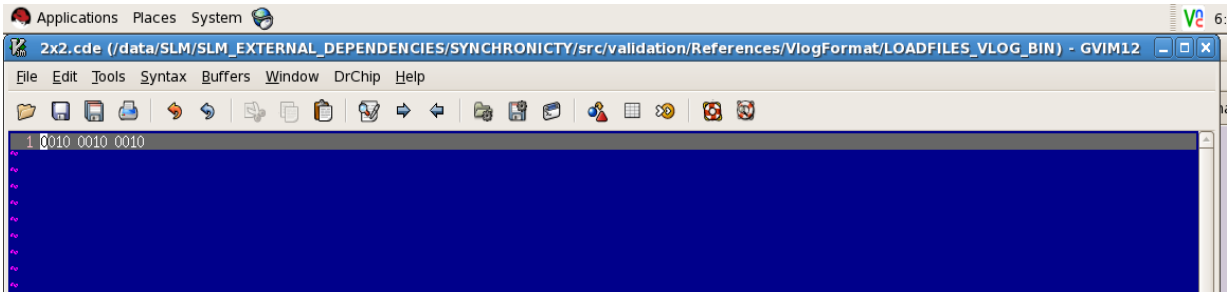

Fig. 7.39(b) 2x2.cde input

Compare.rep is used to log any differences in run_time log w.r.t the Goldrun log of test. Fig. 7.40 present a sample Compare.rep file.

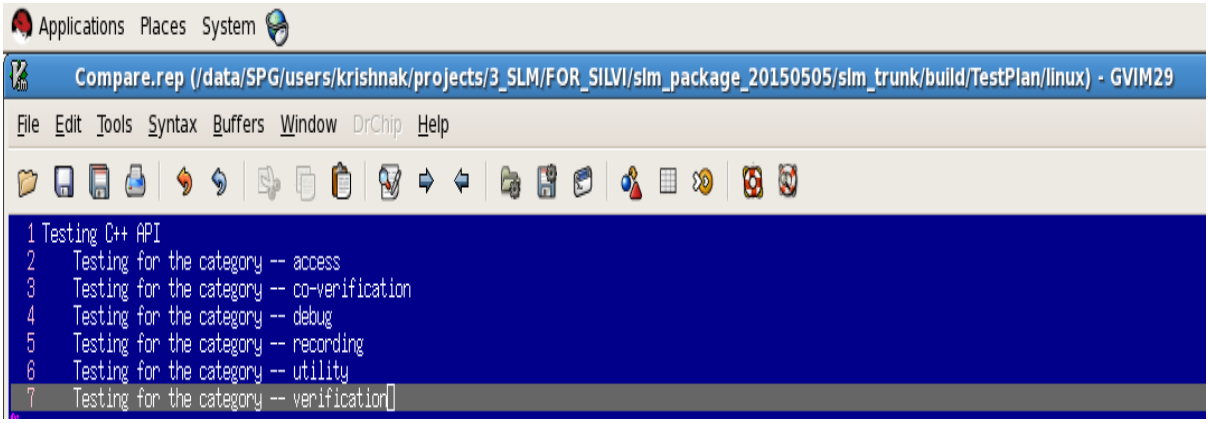

Fig. 7.40 Compare.rep file

Fig. 7.41 presents a list of Test Plans and Functions that are currently included in the automation scope of test cases.

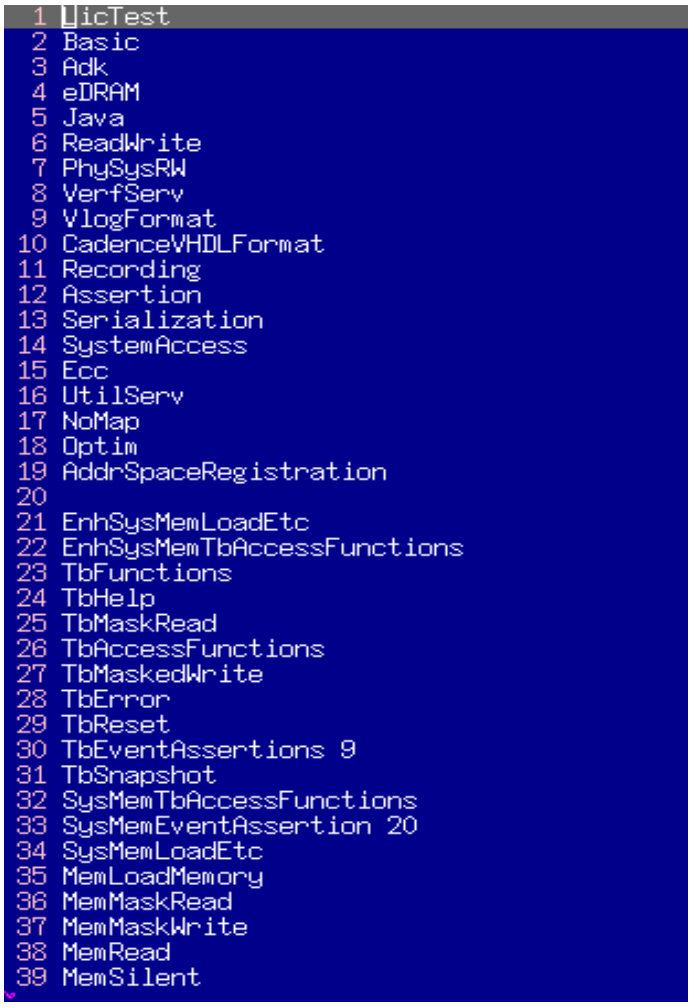

Fig. 7.41 List of Tests included in Automation

Fig. 7.42(a) and 7.42(b) shows two sample memory designs which act as caller to SLM Services in co-sim mode.

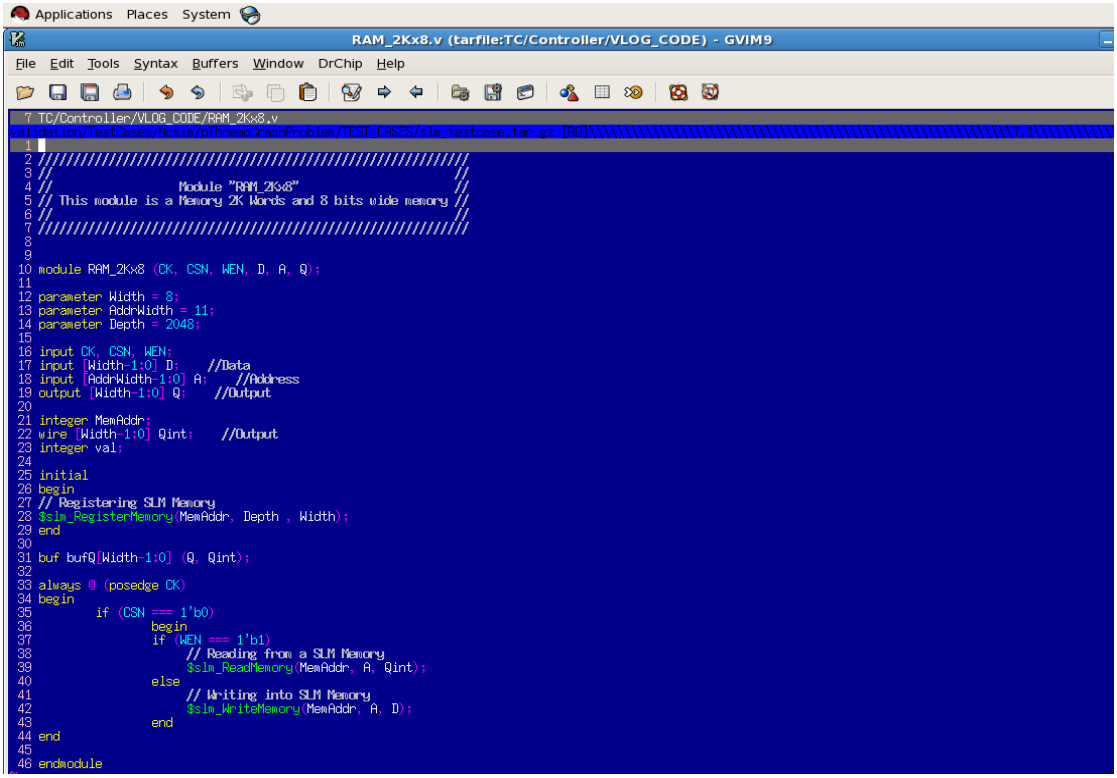

Fig. 7.42(a) Sample RTL Memory Design

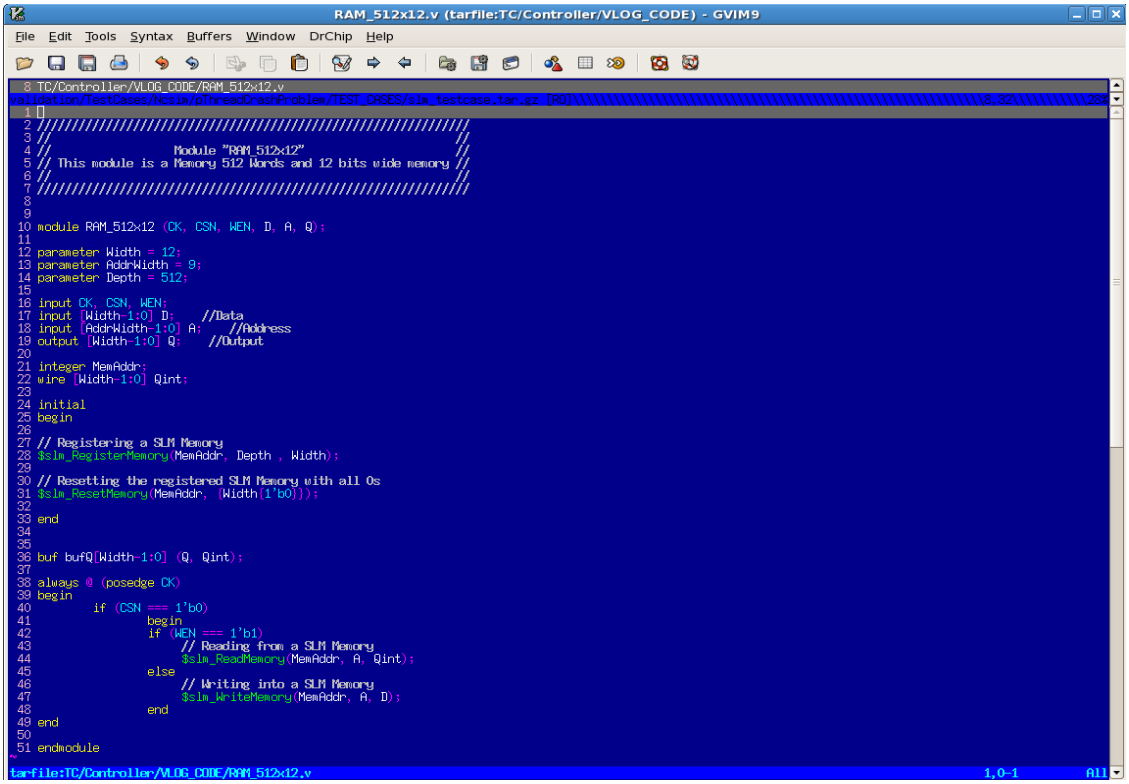

Fig. 7.42(b) Sample RTL Memory Design

7.2.5 Automation Scripts

Fig. 7.43 shows the help for automation test-script usage.

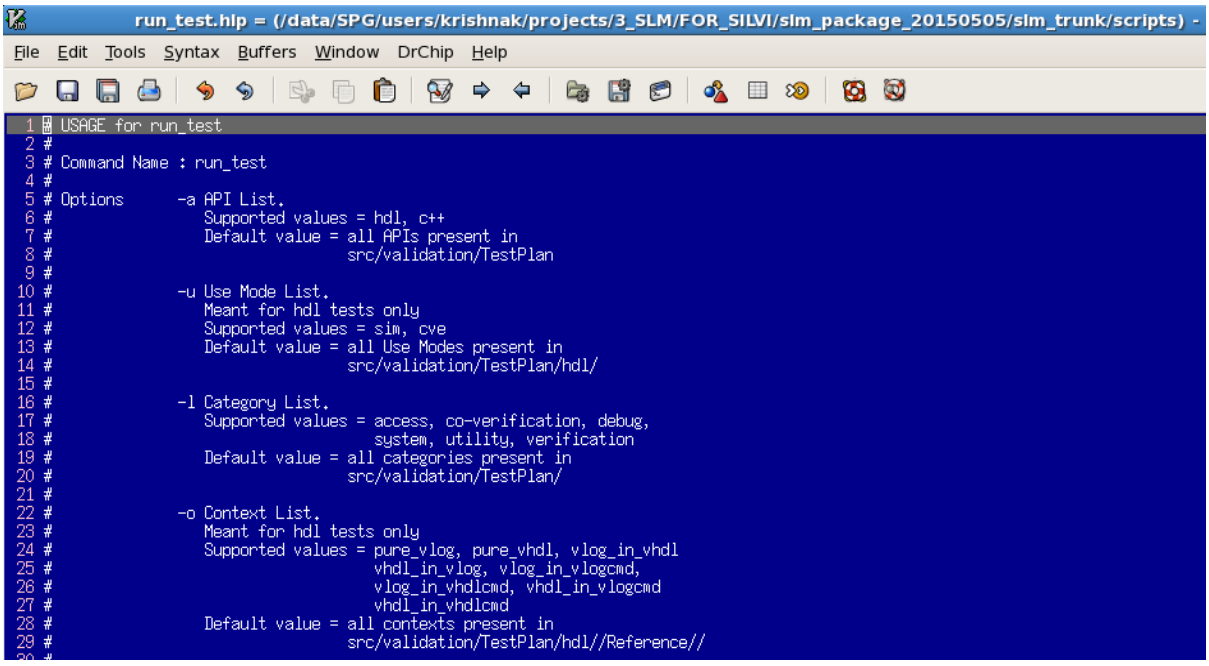

Fig. 7.43 Run_tests.hlp

Fig. 7.44 shows the call to C++ test plan execution in automation script.

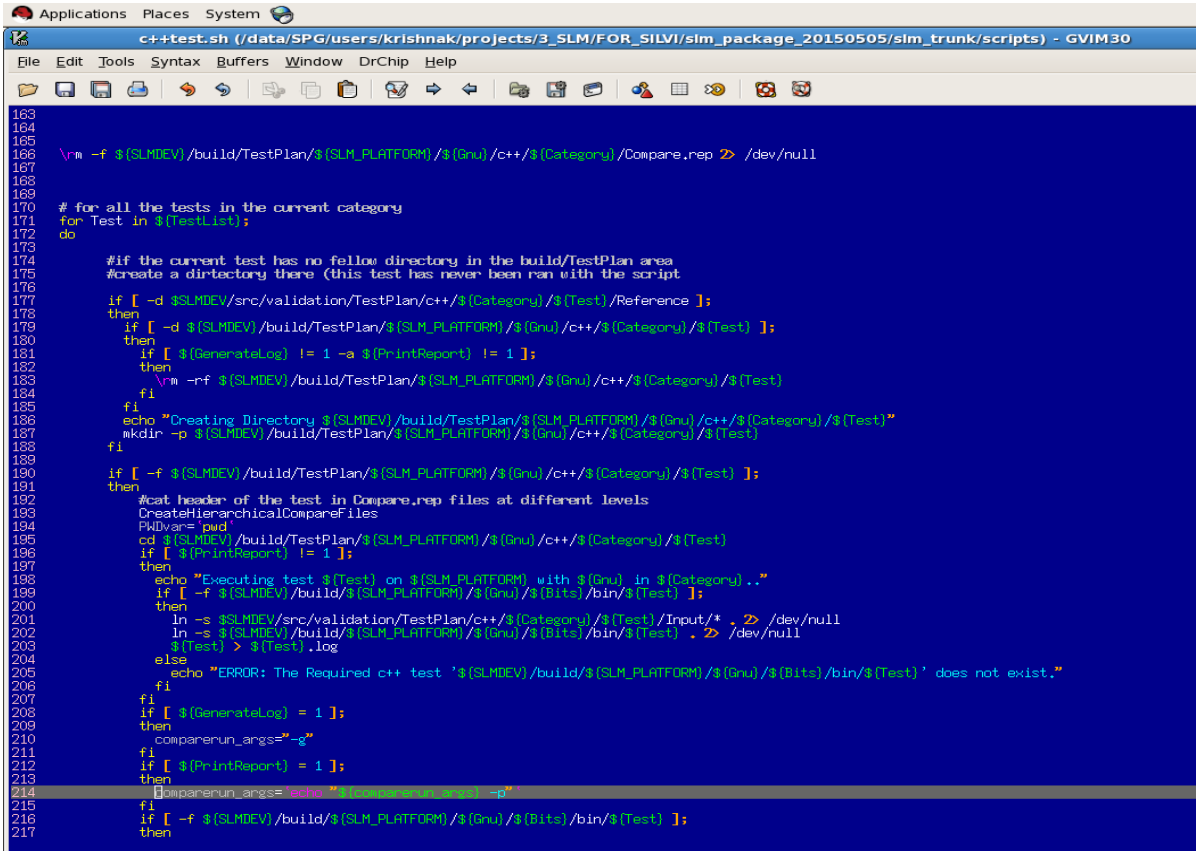

Fig. 7.44 Script for C++ Test Plan Invocation

Fig. 7.45(a), 7.45(b) and 7.45(c) show various sections of script including, checks on input, building output directories and logging results.

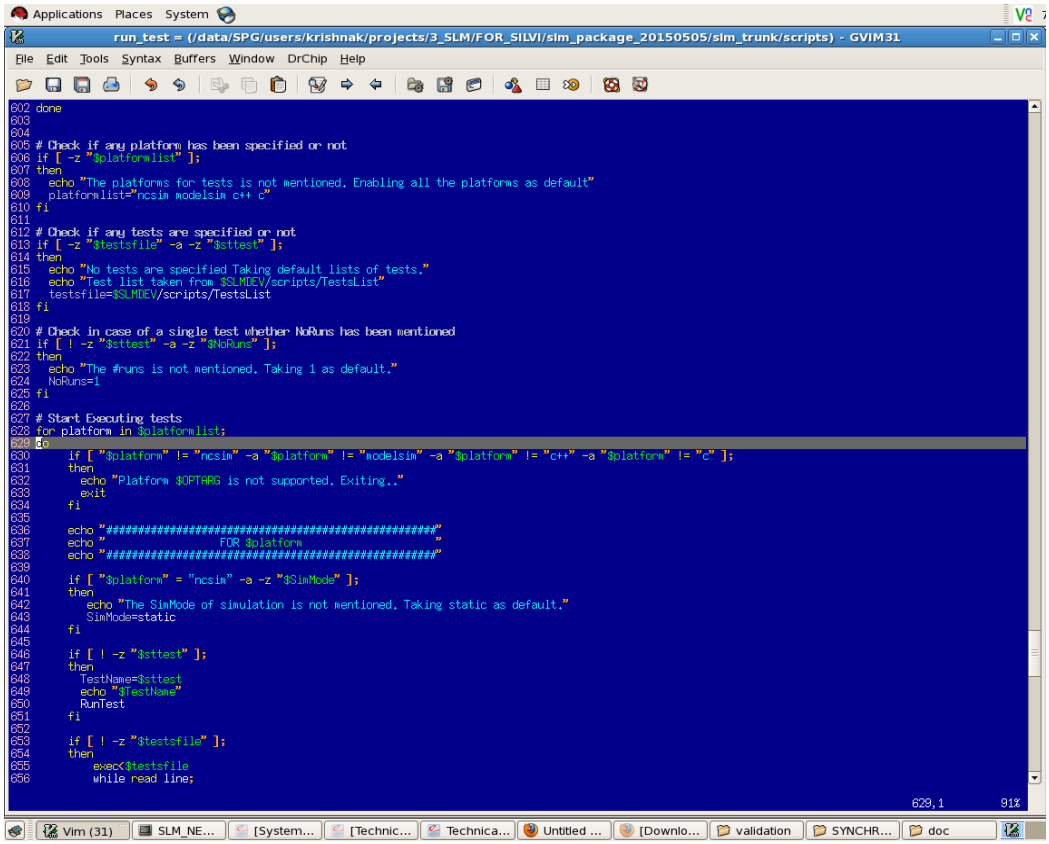

Fig. 7.45(a) Automation Script

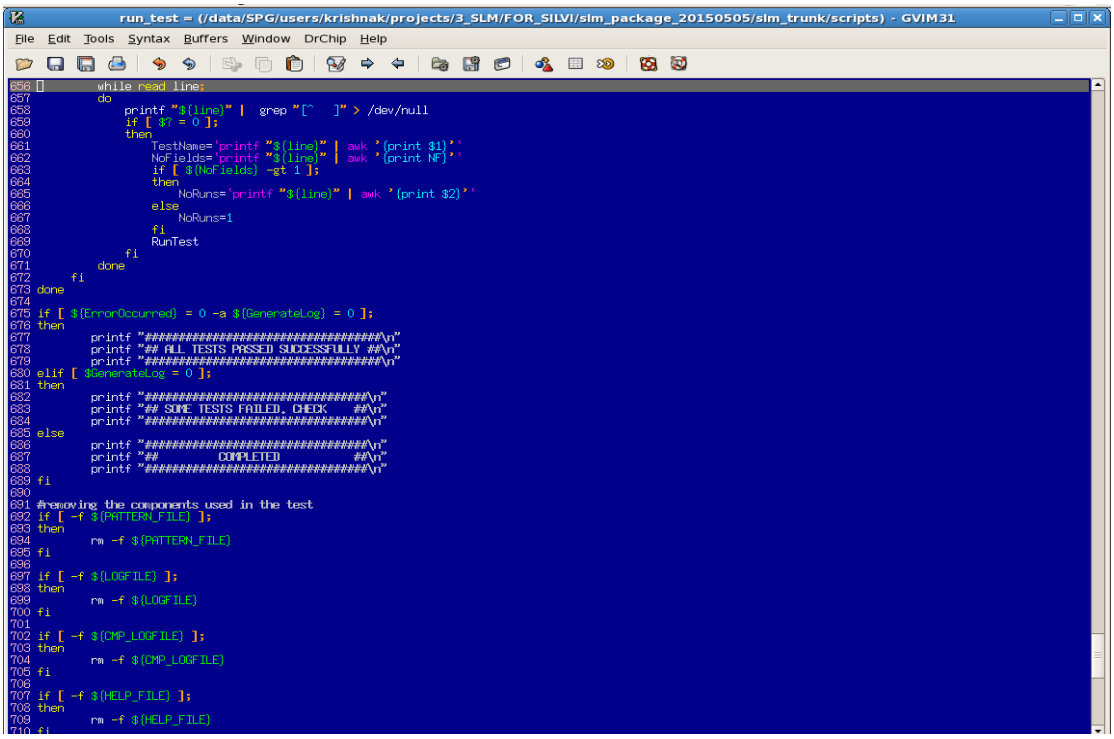

Fig. 7.45(b) Automation Script

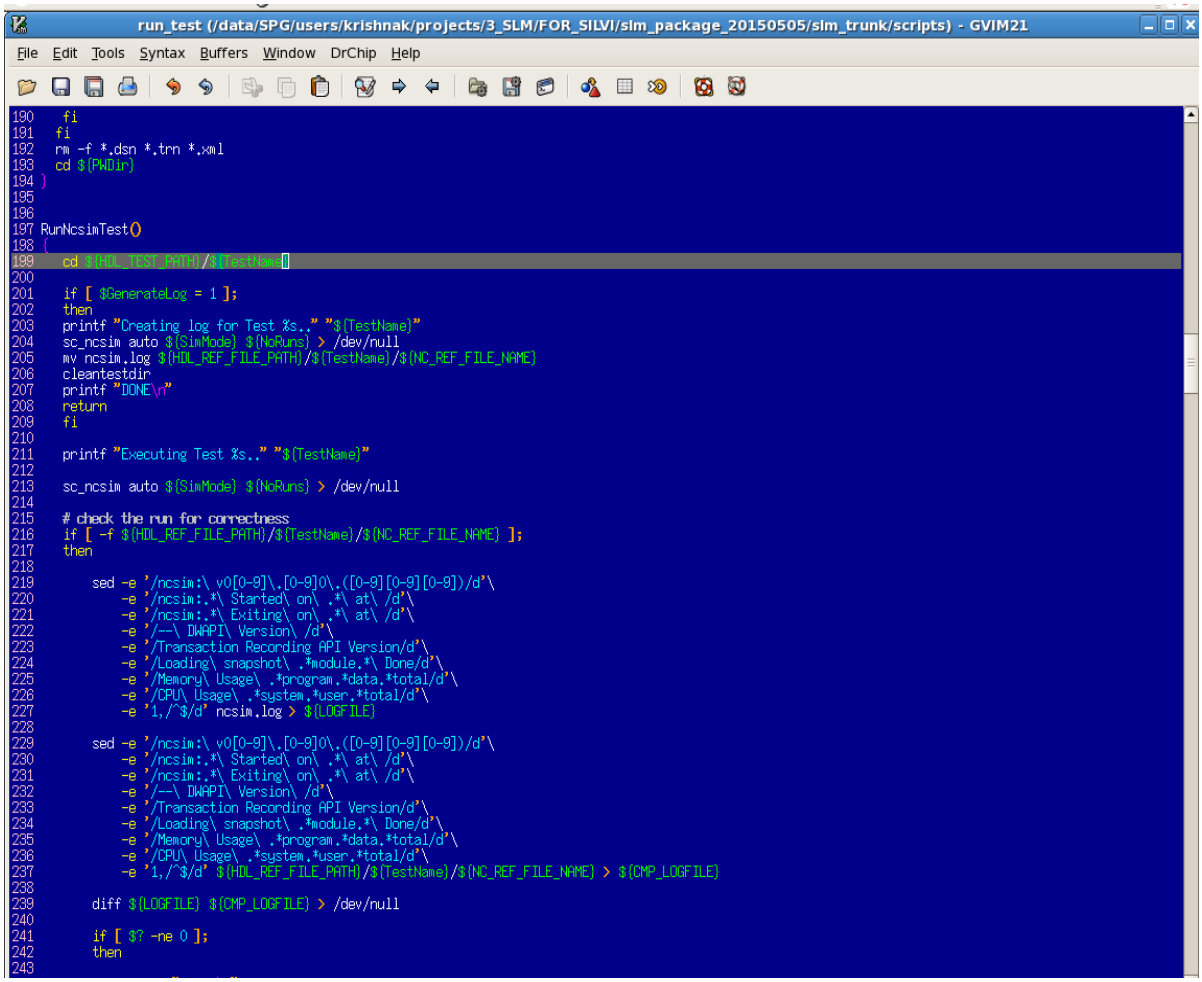

Fig. 7.45(c) Automation Script

Fig. 7.46 shows the log of script testing. It logs if any script is missing, or any resource required by any script is missing. A close look on environment setting is being placed through different scripts which in case of any discrepency , is logged during this run.

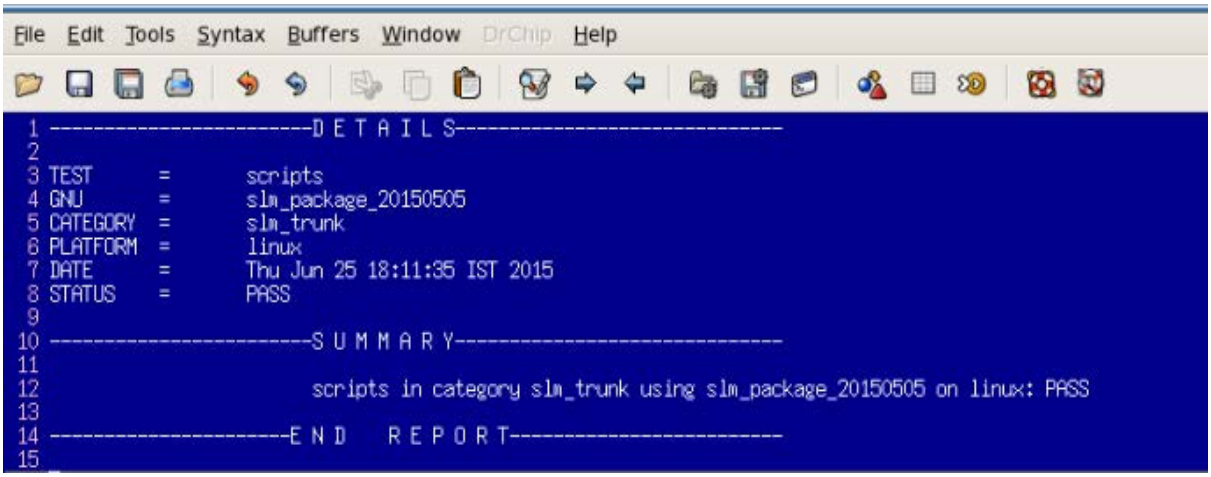

Fig. 7.46 Script Testing

Set of scripts involved in test suite development and automation are depicted in Fig. 7.47.

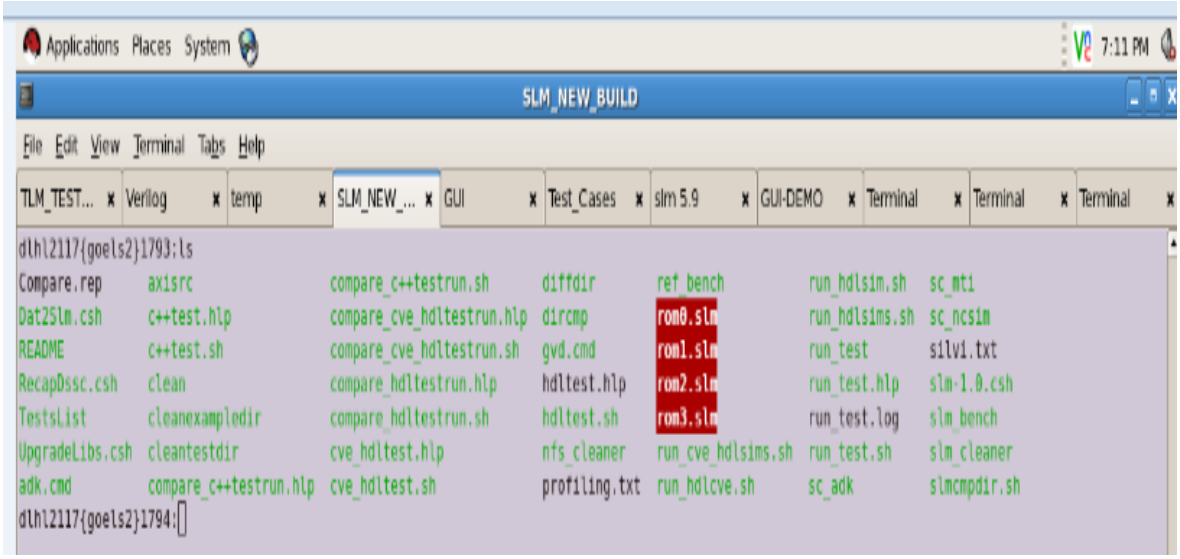

Fig. 7.47 Set of Scripts

7.2.6 Directory Structure

The directory structure for Axis, Ncsim and Seamless based test cases and c++ and HDL is presented in Fig. 48(a), 48(b), 48(c) and 48(d) respectively.

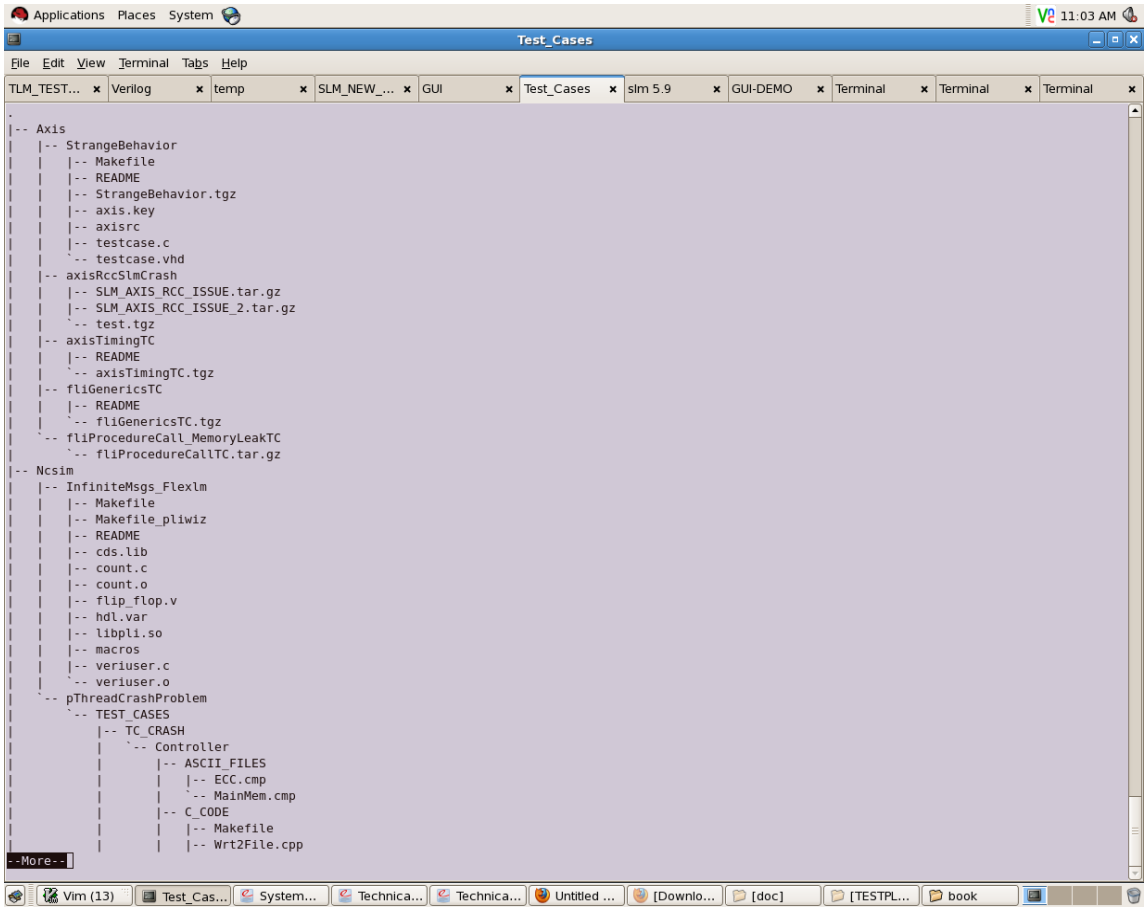

Fig. 7.48(a) Axis & Ncsim Directory Structure

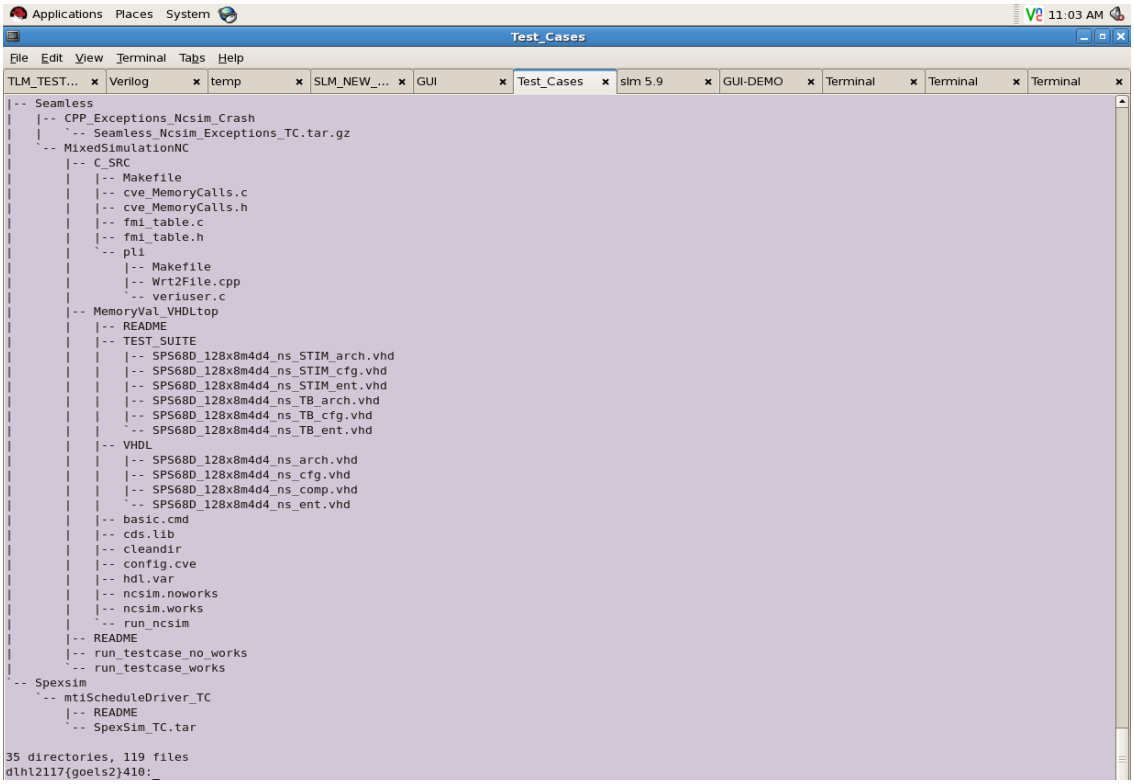

Fig. 7.48(b) Seamless Directory Structure

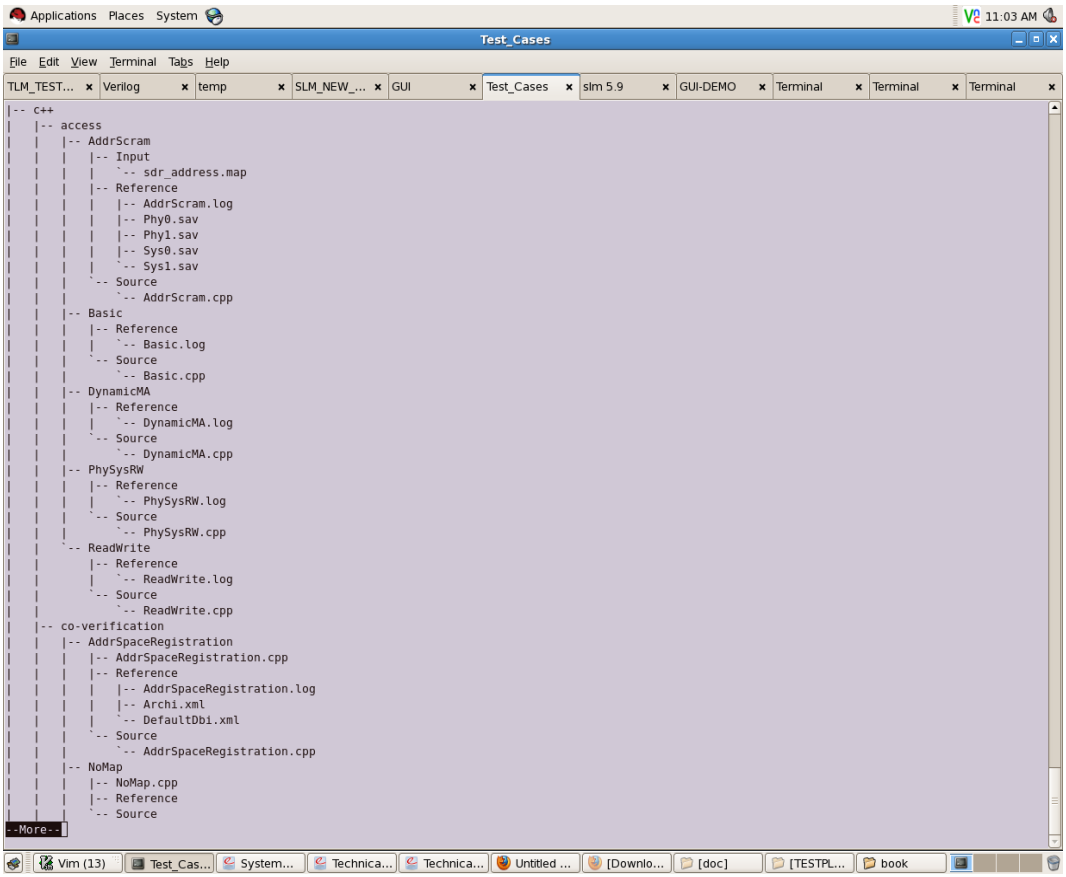

Fig. 7.48(c) C++ Test Plan Directory Structure

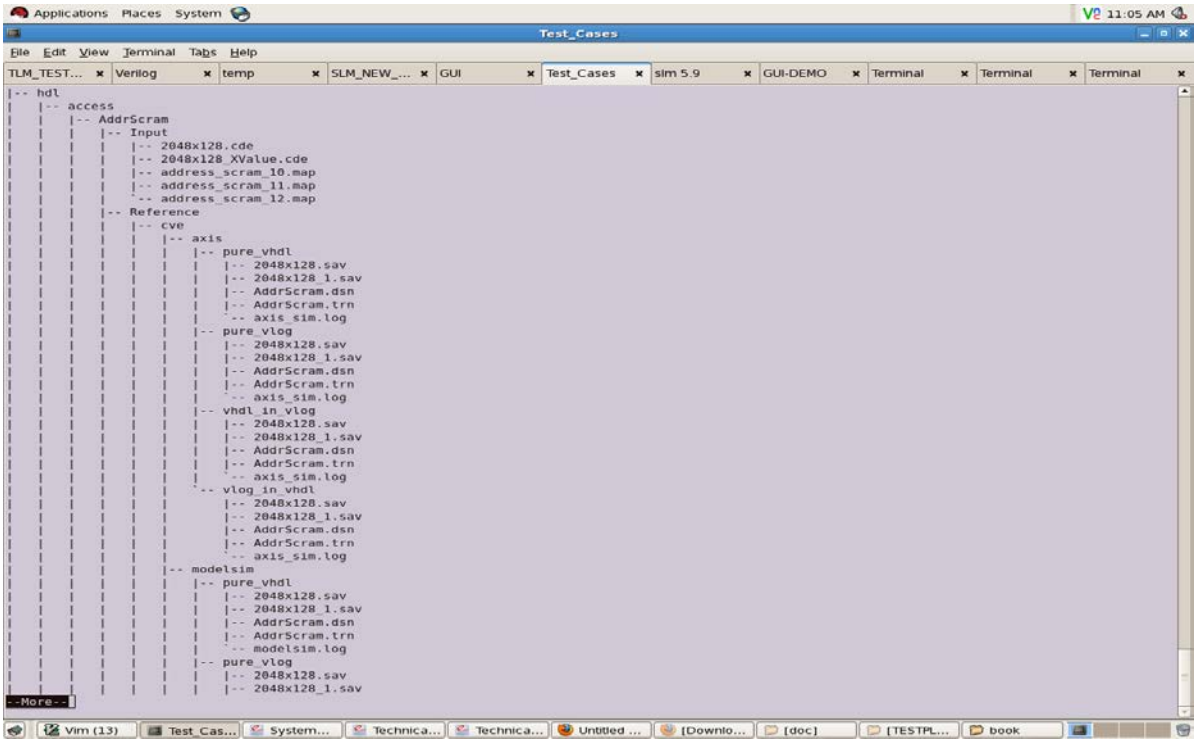

Fig. 7.48(d) HDL Test Plan Directory Structure

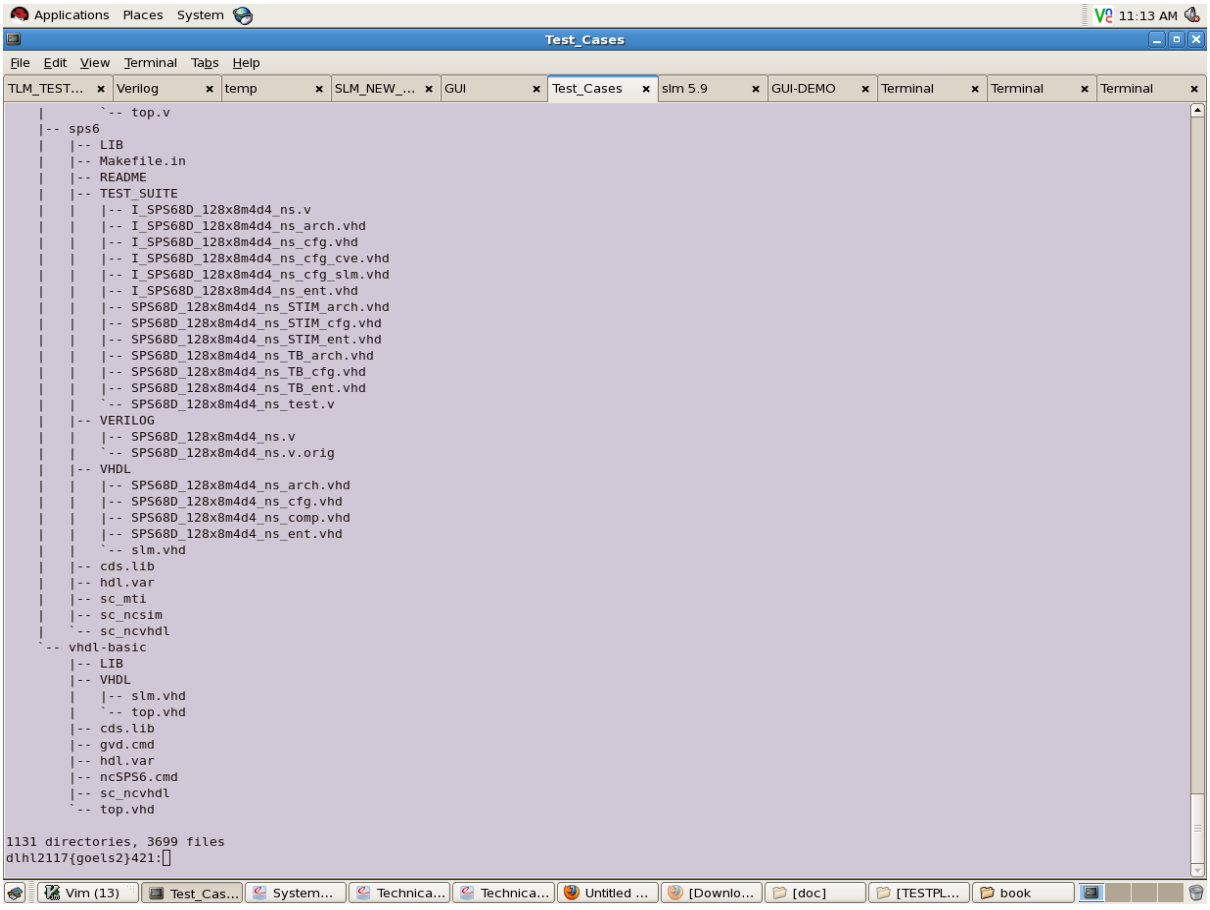

Fig. 7.48(e) Overall Test Suite Directory Structure

7.3 VPI Demonstration

7.3.1 Environment Setup

a. Work Area Path

/data/SPG/users/goels2/tcl_file/SLM_GUI/verilog

b. Environment Variables

setenv CDS_AUTO_64BIT INCLUDE:INCA

setenv INCA_64BIT

c. Instruction

bsub -I -q short ncverilog poc.v

7.3.2 Screenshots

VPI will use PLI and CFC Tables Configuration, which can be done through a set of files to be compiled inclusing veriuser.c and cfctable.c.

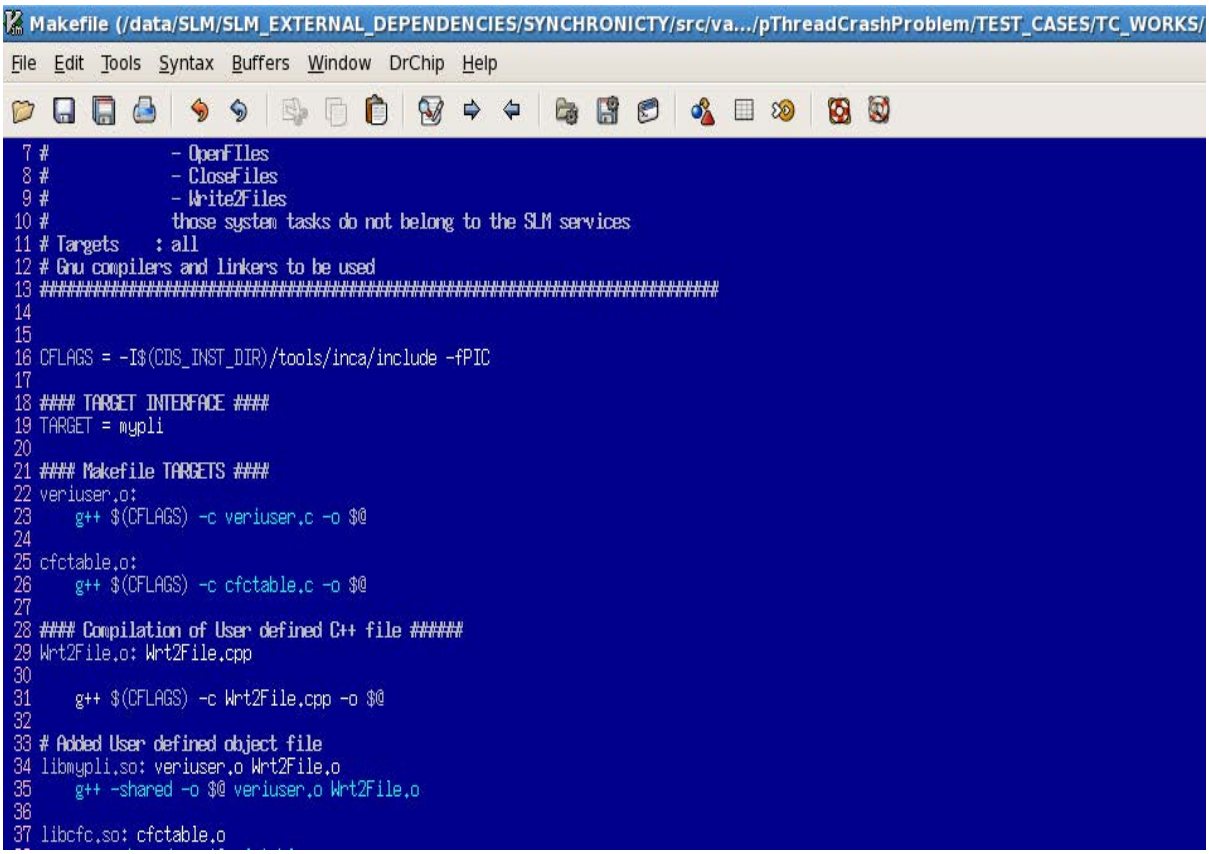

Fig. 7.49 Makefile for CFCTable & PLI Setup

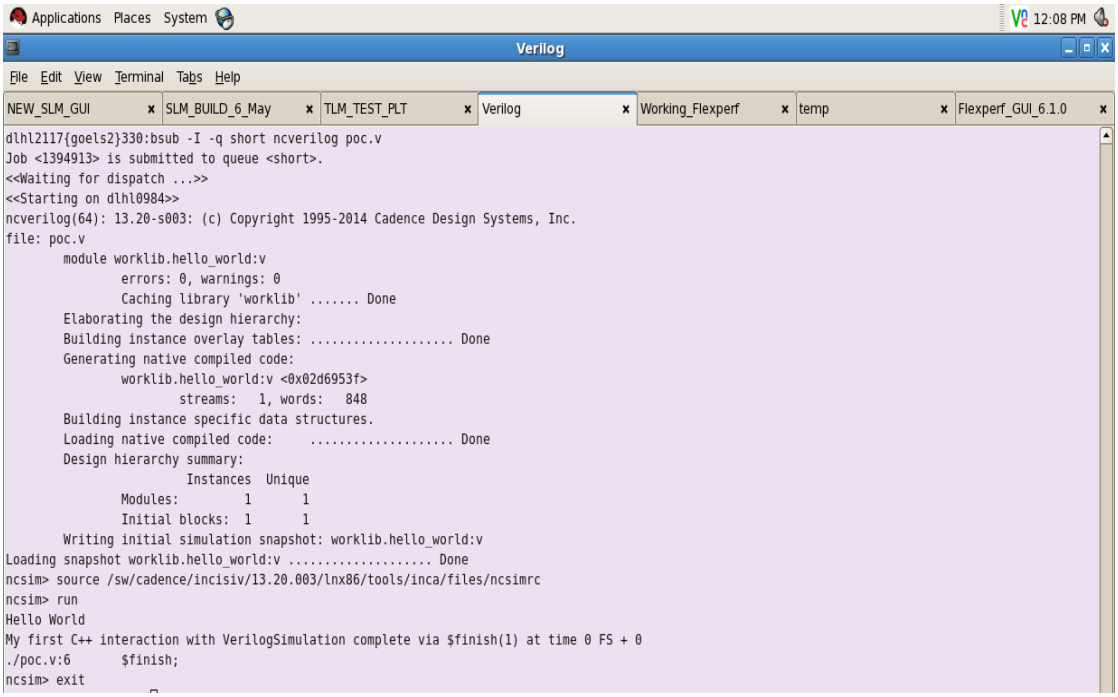

Fig. 7.50 SV Support POC

7.4 SLM Core

7.4.1 Observations & Results

A compiled view of R&D done over SLM Core is provided in a brief through the snapshots of file.

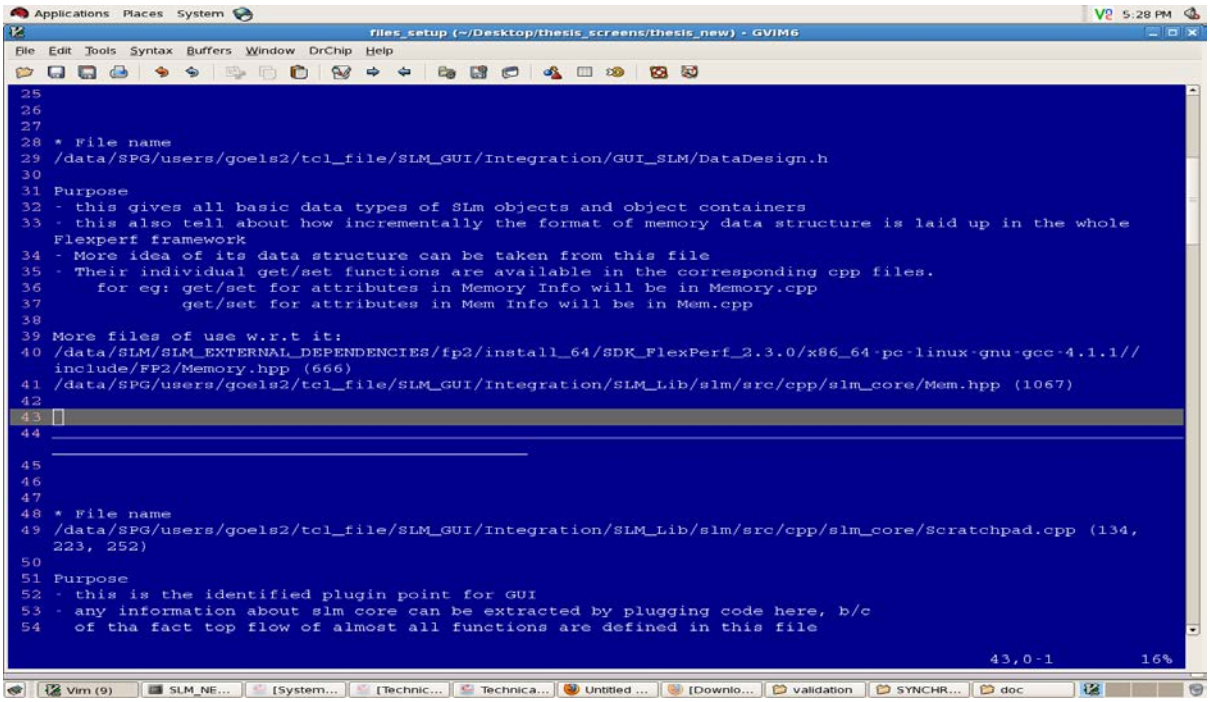

Fig. 7.51(a) R&D Over SLM Core

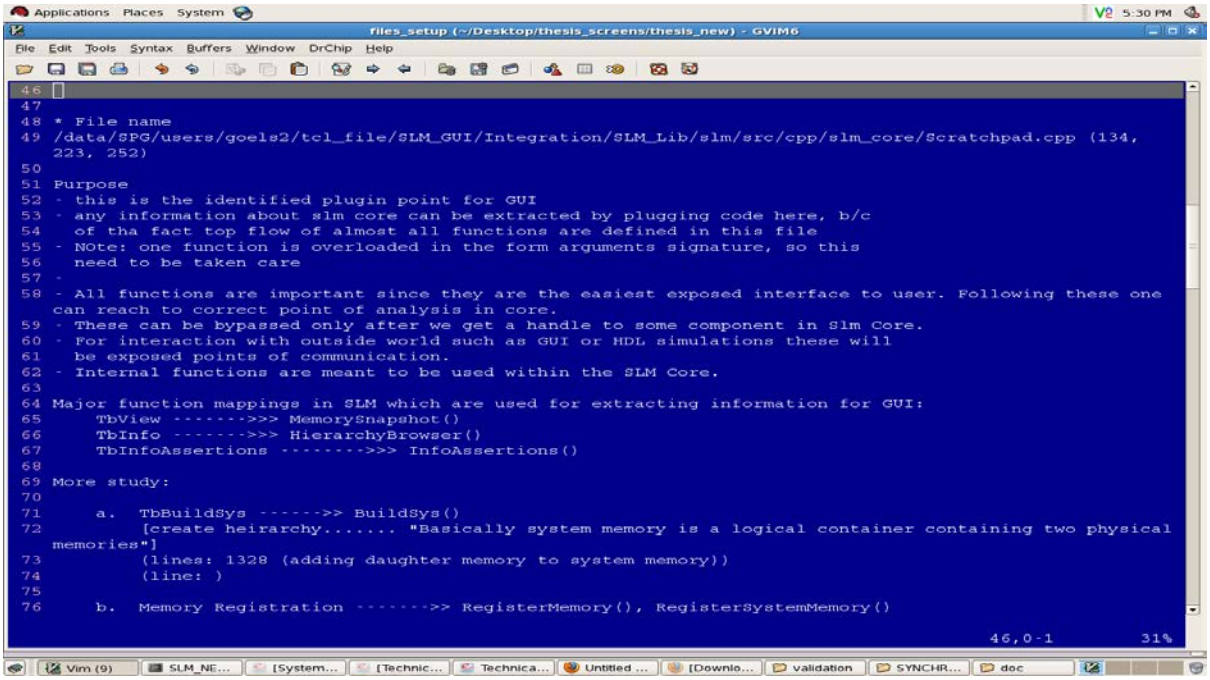

Fig. 7.51(b) R&D Over SLM Core

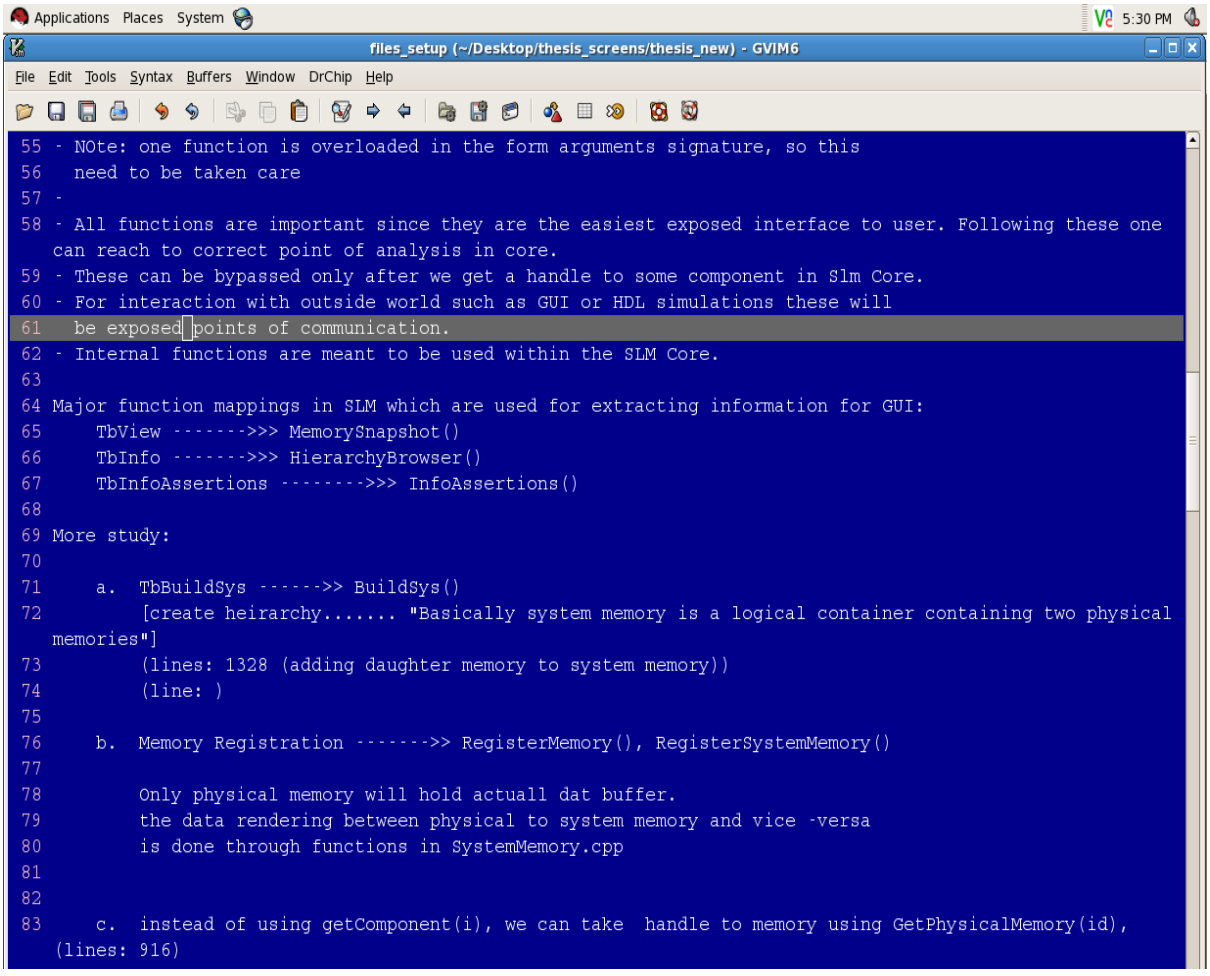

Fig. 7.51(c) R&D Over SLM Core

Chapter-8

Conclusion and Future Scope

8.1 Conclusion

A lot of efforts are being put towards providing user-friendly interfaces to complex system designs. The testing methodology and its automation is a center of focus because many of the systems that are proprietary systems of companies, are still in dire need of appropriate methods for testing. In such a scenario, we present a plan to incorporate helping structures during development phase, which can later make the system being more responsive in the testing phase, and would contribute to a scalable code for testing scripts.

8.2 Future Scope

Introduce testcases improves overall SoC design verification and the developed system can be a new face to SLM by incorporating following enhancements :

- To allow two-way communication (simulator <-> GUI)
- Should also run in Pure TLM mode (non-Co-Sim)
- Provide an extensible solution to the SLM library
- Fully testable without human intervention (for qualification)

References

[1] U. Kamath and R. Kaundin, "System-on-Chip Designs, Strategy for Success", *Wipro Technologies*, pp. 1-9, June 2001.

[2] "The importance of a verification strategy and verification IP in SoC design", *Tech Brief on Verification Strategy, Cadence Design Systems, Inc.*, pp. 1-5, 2005.

[3] J. A. Abraham and D. G. Saab, "Tutorial T4A: Formal Verification Techniques and Tools for Complex Designs", in proceedings of IEEE *20th International Conference on VLSI Design*, 2007.

[4] R. Rajesvari, G. Manoj and M. A. Ponrani, "System-on-Chip for Telecommand System Design", *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 2, No. 3, pp. 1580-1585, March 2013.

[5] J. Bergeron, "Writing TestBenches: Functional Verification of HDL Models", Springer US, pp. 1-24, 2003.

[6] P. Abinaya, S. B. Jeevitha, M. Janani and S. Prema, "Ethernet IP Core Verification Using System Verilog HDL", *International Journal of Advanced Information and Communication Technology*, Vol. 1, No. 11, pp. 865-869, March 2015.

[7] G. M. D. Gandhi and A. S. Pillai, "Challenges in GUI Automation", *International Journal of Computer Theory and Engineering*, Vol. 6, No. 2, pp. 192-195, 2014.

[8] M. Blackburn, R. Busser and A. Nauman, "Why Model-Based Test Automation Is Different and What You Should Know to Get Started", in proceedings of *International conference on practical software quality and testing*, Software Productivity Consortium, pp. 212-232, 2004.

[9] Z. F. Yang, Z. X. Yu, B. B. Yin, and C. G. Bai, "GUI Reliability Assessment based on Bayesian Network and Structural Profile", *International Journal of Signal Processing and Pattern Recognition*, Vol. 8, No. 1, pp. 225-240, 2015.

[10] V. Santiago, A. S. M. Amaral, N. L. Vijaykumar, M. D. Fatima, M. Fransisco, E. Martins and O. C. Lopes, "A Practical Approach for Automated Test Case Generation

using Statecharts", in proceedings of *IEEE Conference on Computer Software and Applications*, Vol. 2, pp. 183-188, 2006.

[11] R. M. Sharma, "Quantitative Analysis of Automation and Manual Testing", *International Journal of Engineering and Innovative Technology*, Vol. 4, No. 1, pp. 252-257, 2014.

[12] Z. Liu, Q. Chen and L. Cai, "Research on GUI-based Automation Test Technology driven by Separated Definition Data", *International Journal of Control and Automation*, Vol. 7, No. 6, pp. 421-432, 2014.

[13] H. Zhu, D. Hoffman, J. Hughes and D. Xu, "Toward a mature industrial practice of software test automation", Software Quality Control, Vol. 22, No. 2, pp.239-240, 2014.

[14] Z. Liu, Q. Chen and L. Cai, "An Automated Function Test Framework for Business Workflow Test Based on Data File", in the proceedings of *Advanced Science and Technology Letters*, Vol. 45, pp. 136-141, 2014.

[15] M. Sarma, P. V. R. Murthy, S. Jell and A. Ulrich, "Model-based testing in industry: a case study with two MBT tools", *in proceedings of the 5th Workshop on Automation of Software Test*, pp. 87-90, ACM, 2010.

[16] S. Goel and R. Aggarwal, "A Novel Approach for GUI Test Automation Using Conscious Development Constructs", *in proceedings of International Conference on Cloud Computing Computer Science And Advances in Information Technology*, pp. 1-5, 2015.

[17] B. Foote and J.Yoder, "Big Ball of Mud", in proceedings of *Fourth Conference on Patterns Languages of Programs*, pp. 654-692, June 1997.

[18] S. Sutherland, "Using PLI 2.0 (VPI) with VCS", in proceedings of Synopsys User Group Conference, San Jose, pp. 1- 15, 2002.

[19] S. Sutherland, "The Verilog PLI is Dead (may be) Long Live the SystemVerilog DPI!", in proceedings of Synopsys User Group Conference, San Jose, March 2004.

[20] S. Sutherland, “Integrating SystemC & Verilog using SystemVerilog’s DPI”, in proceedings of Synopsys User Group Conference, Europe, 2004.

[21] System Verilog DPI Tutorial, Online: [http://www.project-veripage.com/dpi_tutorial_1. Php.

[22] B. K. Madhavi and R. P. Rao, “A Novel Design Approach to Increase the speed Of VLSI Circuits in mixed Signal Environment”, *International Journal of VLSI and Enbedded Systems*, Vol.6, pp. 1441-1449, January 2015.

[23] “Incisive Enterprise Simulator, Multi-language simulation for low-power, metric-driven, mixed signal verification”, Spec Manual by Cadence, 2011.

[24] Online Reference: http://www.mentor.com

[25] Online Reference: http://www.synopsys.com/

List of Publications

1. S. Goel and R. Rani, “A Novel Approach for GUI Test Automation Using Conscious Development Constructs”, *in Proceedings of International Journal Of Advances In Cloud Computing And Computer Science*, Chandigarh, pp. 1-5, May 17, 2015.

2. Online: [https://www.youtube.com/watch?v=m6o9bRqIUW0]