

**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



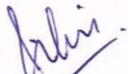
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT  
THAPAR UNIVERSITY  
PATIALA – 147004

**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.

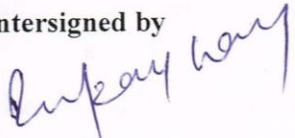
  
(Silvi Goel)


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

  
(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

---

---

Certificate.....	i
Acknowledgement .....	ii
Abstract.....	iii
Table of Contents .....	iv
List of Figures .....	ix
List of Tables .....	xiii
<b>1. Introduction.....</b>	<b>1</b>
<b>1.1 SYSTEM-ON-CHIP DESIGN .....</b>	<b>1</b>
1.1.1 TECHNOLOGY AND TREND .....	1
1.1.2 DESIGN PROCESS .....	2
1.1.3 OPPORTUNITIES AND CHALLENGES .....	4
<b>1.2 ROLE OF VERIFICATION .....</b>	<b>4</b>
<b>1.3 OUR TEAM .....</b>	<b>5</b>
<b>1.4 COMPLEX MEMORY OPERATIONS.....</b>	<b>5</b>
<b>1.5 VERIFICATION TECHNOLOGIES IN SOC DESIGN.....</b>	<b>5</b>
<b>1.6 VERIFICATION ENVIRONMENT .....</b>	<b>6</b>
1.6.1 KEY TERMS .....	6
<b>1.7 SYSTEM LEVEL MEMORY MODEL.....</b>	<b>6</b>
1.7.1 WHAT AN SLM MODEL LOOKS LIKE? .....	7
1.7.2 CALLING C PROCEDURES FROM HDL .....	7
1.7.3 HOW IT WORKS? .....	8

1.7.4	SLM ADVANTAGES?	8
1.7.5	PLATFORMS, SIMULATORS AND LANGUAGES	9
1.7.6	NEED OF SLM	9
1.7.7	APPLICATIONS	10
1.7.8	SLM SERVICE CATEGORIES	10
<b>2.</b>	<b>Literature Review</b>	<b>11</b>
<b>2.1</b>	<b>GUI DESIGN &amp; TEST AUTOMATION</b>	<b>11</b>
2.1.1	RECENT WORK	11
2.1.2	CHALLENGES	12
<b>2.2</b>	<b>SYSTEM VERILOG SUPPORT</b>	<b>15</b>
<b>2.3</b>	<b>SYSTEM UPGRADATION</b>	<b>16</b>
<b>2.4</b>	<b>TEST SUITE ENHANCEMENT AND AUTOMATION</b>	<b>16</b>
2.4.1	SIMULATORS	16
2.4.1.1	INCISIVE BY CADENCE	16
2.4.2.1	MODELSIM BY MENTOR GRAPHICS	17
2.4.3.1	VCS BY SYNOPSYS	18
<b>3.</b>	<b>Research Problem</b>	<b>20</b>
<b>3.1</b>	<b>PROBLEM STATEMENT</b>	<b>20</b>
<b>3.2</b>	<b>OBJECTIVES</b>	<b>21</b>
<b>3.3</b>	<b>RESEARCH METHODOLOGY</b>	<b>22</b>

<b>4. SLM GUI Design.....</b>	<b>23</b>
<b>4.1 DESIGN DECISIONS .....</b>	<b>23</b>
4.1.1 DATA SHARING METHODOLOGIES. ....	23
<b>4.2 SYSTEM DESIGN.....</b>	<b>25</b>
4.2.1 SYSTEM ARCHITECTURE. ....	25
4.2.2 LAYERED VIEW OF SLM. ....	25
4.2.3 OVERALL PROCESS FLOW DESCRIPTION. ....	28
<b>5. SLM GUI Test Suite Design .....</b>	<b>30</b>
<b>5.1 DESIGN DECISIONS .....</b>	<b>30</b>
5.1.1 GUI TESTING METHODOLOGIES. ....	30
<b>5.2 DESIGN APPROACH.....</b>	<b>31</b>
5.2 SYSTEM ARCHITECTURE .....	35
<b>6. SLM Test Suite Design .....</b>	<b>36</b>
<b>6.1 SYSTEM DESIGN.....</b>	<b>36</b>
<b>6.2 TEST SUITE STRUCTURE.....</b>	<b>37</b>
<b>6.3 TEST CASE DESCRIPTION.....</b>	<b>40</b>
6.3.1 BIST.....	40
6.3.2 ECALLBACKTEST .....	40
6.3.3 EREADWRITE TEST .....	40
6.3.4 CONTROLLER .....	40

6.3.5	MEMORYEVCTEST .....	41
<b>7.</b>	<b>Implementation and Results .....</b>	<b>42</b>
<b>7.1</b>	<b>SLM GUI &amp; GUI TEST SUITE .....</b>	<b>42</b>
7.1.1	ENVIRONMENT SETUP.....	42
7.1.2	LOGGING FACILITIES .....	46
7.1.3	EXECUTION SCREENSHOTS .....	47
7.1.4	EXTRACTED INFORMATION.....	57
7.1.5	GUI TESTING PROCESS.....	58
7.1.6	DIRECTORY STRUCTURE.....	59
<b>7.2</b>	<b>SLM TEST SUITE .....</b>	<b>60</b>
7.2.1	ENVIRONMENT SETUP.....	60
7.2.2	RUNNING STATUS OF TEST CASES .....	61
7.2.3	TEST CASES RUNTIME LOGS.....	62
7.2.4	SLM INPUTS .....	67
7.2.5	AUTOMATION SCRIPTS.....	70
7.2.6	DIRECTORY STRUCTURE.....	73
<b>7.3</b>	<b>VPI DEMONSTRATION .....</b>	<b>76</b>
7.3.1	ENVIRONMENT SETUP.....	76
7.3.2	SCREENSHOTS.....	76
<b>7.4</b>	<b>SLM CORE .....</b>	<b>77</b>
7.4.1	OBSERVATION & RESULTS .....	77

<b>8. Conclusion and Future Scope .....</b>	<b>79</b>
<b>8.1 CONCLUSION .....</b>	<b>79</b>
<b>8.2 FUTURE SCOPE.....</b>	<b>79</b>
<b>References .....</b>	<b>80</b>
<b>List of Publications .....</b>	<b>83</b>

## List Of Figures

<b>Figure No.</b>	<b>Description</b>	<b>Page No.</b>
Fig.1.1	Overview of basic components inside the SoC	1
Fig.1.2	SoC Design Process	3
Fig.1.3	Verification Environment	6
Fig.1.4	Sample code depicting SLM Model	7
Fig.1.5	C Procedure Calls from HDL	7
Fig.1.6	Usability of SLM model with various simulators	9
Fig. 2.1	Inherent problem traced from current system	14
Fig. 2.2	Various verticals explored for SLM GUI Design	14
Fig. 2.3	Sample DPI-based program	16
Fig. 2.4	Comprehensive View of Incisive	17
Fig. 2.5	Comprehensive View of ModelSim	18
Fig. 2.6	Unified Coverage in VCS	19
Fig. 3.1	Explored sub-sections for problem solving	20
Fig. 3.2	Research Domain Tree	22
Fig. 4.1	Serialization based communication	23
Fig. 4.2	C++ based communication	24
Fig. 4.3	File based communication	24
Fig. 4.4	In-memory XML based communication	24
Fig. 4.5	SLM basic architecture of sub-systems	25
Fig. 4.6	Comparative Layered View of SLM	25
Fig. 4.7	Hook in SLM View Manager	26
Fig. 4.8	Attributes of SLM View Provider	26
Fig. 4.9	Current data structure for retrieved information	27
Fig. 4.10	Behavior of SLM View Provider	27
Fig. 4.11	Constructors & Destructors for SLM View Provider	28
Fig. 4.12	Environment Checking	28
Fig. 4.13	GUI Starter	29
Fig. 4.14	GUI Builder	29
Fig. 5.1	Three perspectives for GUI Testing	31

Fig. 5.2	Binding Table Structure using pseudo-code	32
Fig. 5.3	Sample log of Binding Table contents	32
Fig. 5.4	XML format for milestone representation used to dump branch information	33
Fig. 5.5	API model associated with Functional Coverage Mechanism	34
Fig. 5.6	System Architecture for GUI Test Suite	35
Fig. 6.1	Test Case Run-time	36
Fig. 6.2	Basic Test Suite Structure	37
Fig. 6.3(a)	References-Golden Log	37
Fig. 6.3(b)	Reference-Runtime Log	37
Fig. 6.4	Detailed Test Suite Structure	38
Fig. 6.5	Detailed Test Cases Structure	38
Fig. 6.6	Detailed Test Plan Structure	39
Fig. 7.1	Various sections in SLM Environment Setup	42
Fig. 7.2	Makefile for GUI Build	42
Fig. 7.3	Shared GUI Library	43
Fig. 7.4	GUI_ENABLER Environment Variable	43
Fig. 7.5	Test Mode Macro Controller	43
Fig. 7.6	SLM Generic Compile Script for GUI Integration	44
Fig. 7.7	SLM Makefile Script for GUI Integration	44
Fig. 7.8	Modified header file cpptk.h	45
Fig. 7.9	SLM Build Makefile	45
Fig. 7.10	TLM Mode Configuration for SLM	46
Fig. 7.11	Co-Sim Mode Configuration for SLM	46
Fig. 7.12	Test_Case.log	46
Fig. 7.13	Status.log	47
Fig. 7.14	Splash Screen	47
Fig. 7.15	TbInfo Screen	48
Fig. 7.16(a)	Function Mapper Screen-1	48
Fig. 7.16(b)	Function Mapper Screen-2	49
Fig. 7.17	Offline Help [In process of development]	49

Fig. 7.18	Memory Grid	50
Fig. 7.19(a)	GUI Running Log-1	50
Fig. 7.19(b)	GUI Running Log-2	51
Fig. 7.19(c)	GUI Running Log-3	51
Fig. 7.20(a)	SLM Build Terminal Output	52
Fig. 7.20(b)	Set of SLM shared libraries	52
Fig. 7.21(a)	TLM Run-1	53
Fig. 7.21(b)	TLM Run-2	53
Fig. 7.22	Ncsim GUI Initialization	54
Fig. 7.23	Ncsim Design Browser	54
Fig. 7.24	SimVision Console	55
Fig. 7.25(a)	Ncsim Simluation using SLM-View 1	55
Fig. 7.25(b)	Ncsim Simluation using SLM - View 2	56
Fig. 7.25(c)	Ncsim Simluation using SLM – View 3	56
Fig. 7.26	Memory Attributes	57
Fig. 7.27	Memory Buffer	57
Fig. 7.28(a)	Event Registerations	58
Fig. 7.28(b)	Binding Table Logger	58
Fig. 7.28(c)	Binding Function Call	58
Fig. 7.28(d)	DFS traversal	58
Fig. 7.29(a)	Directory structure of GUI	59
Fig. 7.29(b)	Directory structure of GUI	59
Fig. 7.30(a)	Required Environment Variables	60
Fig. 7.30(b)	Required scripts for configuration setup	60
Fig. 7.30(c)	SLM Setup	60
Fig. 7.31(a)	Build Tests	60
Fig. 7.31(b)	Build Tests	61
Fig. 7.32	Status of Test Cases [A subset of test-suite]	61
Fig. 7.33	Test Run Start	62
Fig. 7.34(a)	Adk Run	62
Fig. 7.34(b)	Inputs to Adk Test	62
Fig. 7.35(a)	Assertions Run	63

Fig. 7.35(b)	Assertions Run	63
Fig. 7.35(c)	Assertions Run	64
Fig. 7.35(d)	Assertions Run	64
Fig. 7.35(e)	Assertions Run	65
Fig. 7.36	Basic Run	65
Fig. 7.37	UtilServ Test Run	65
Fig. 7.38(a)	VerifServ Test Run	66
Fig. 7.38(b)	VerifServ Test Run	66
Fig. 7.39(a)	512x16.cde input	67
Fig. 7.39(b)	2x2.cde input	67
Fig. 7.40	Compare.rep file	68
Fig. 7.41	List of Tests included in Automation	68
Fig. 7.42(a)	Sample RTL Memory Design	69
Fig. 7.42(b)	Sample RTL Memory Design	69
Fig. 7.43	Run_tests.hlp	70
Fig. 7.44	Script for C++ Test Plan Invocation	70
Fig. 7.45(a)	Automation Script	71
Fig. 7.45(b)	Automation Script	71
Fig. 7.45(c)	Automation Script	72
Fig. 7.46	Script Testing	72
Fig. 7.47	Set of Scripts	73
Fig. 7.48(a)	Axis & Ncsim Directory Structure	73
Fig. 7.48(b)	Seamless Directory Structure	74
Fig. 7.48(c)	C++ Test Plan Directory Structure	74
Fig. 7.48(d)	HDL Test Plan Directory Structure	75
Fig. 7.48(e)	Overall Test Suite Directory Structure	75
Fig. 7.49	Makefile for CFCTable & PLI Setup	76
Fig. 7.50	SV Support POC	77
Fig.7.51(a)	R&D Over SLM Core	77
Fig.7.51(b)	R&D Over SLM Core	78
Fig.7.51(c)	R&D Over SLM Core	78

## List Of Tables

<b>Table No.</b>	<b>Description</b>	<b>Page No.</b>
Table 1.1	Supported simulators and languages	9
Table 2.1	Modelling Languages and formats supported by VCS AMS	18
Table 6.1	Categories of test cases in the test suite	36

# 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 cause of steep rise in complexity of circuit design. This complexity includes 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 supports 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 units in 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.

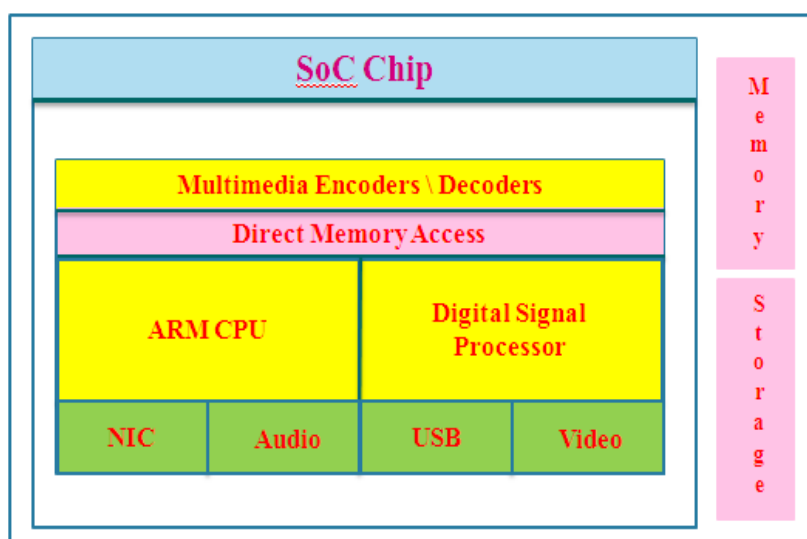


Fig. 1.1 Basic overview of components inside the SoC

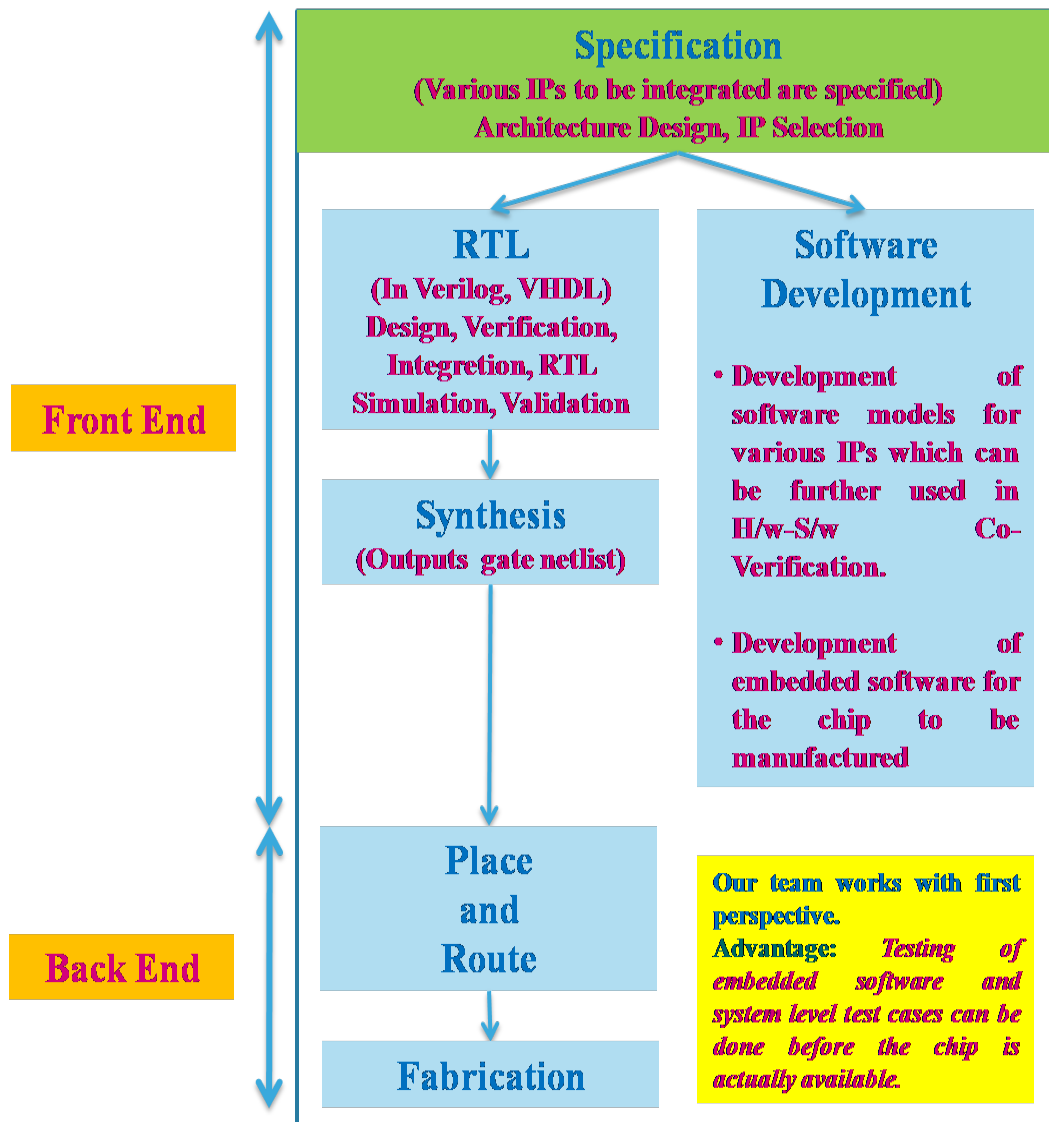
### 1.1.2 Design Process

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 integration test vectors which are generally 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 in 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 integration on a single piece of silicon.
- Significantly reduced overall design cycle time with retained/enhanced performance.
- Evolution of multi-core architectures and 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 integration over the same silicon area being available. The growth of design reuse is resulting rise in the percentage of silicon 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 mining techniques, 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.

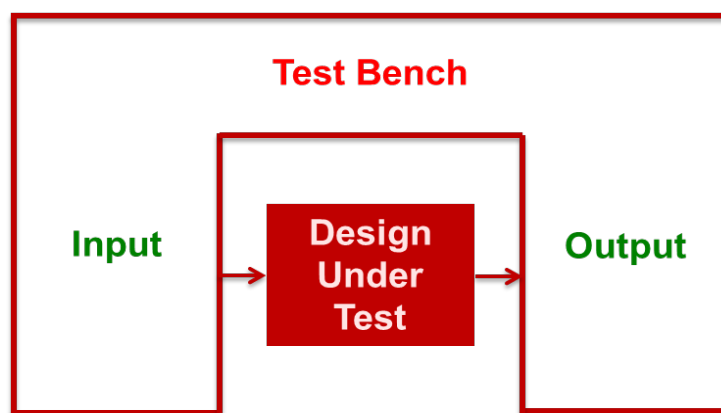
**1.5.2 Formal Technologies :** It can include theorem proving techniques, or formal model and equivalence checking techniques.

**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.

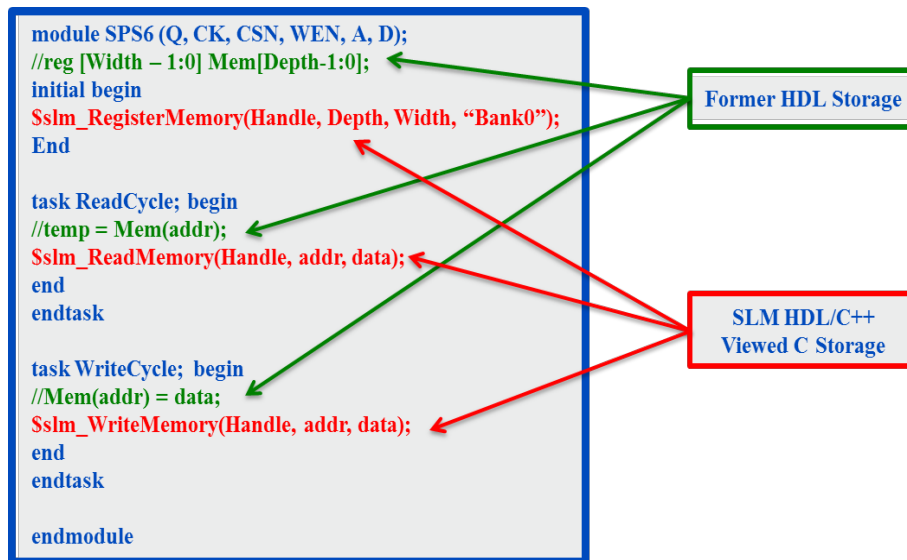


Fig. 1.4 Sample code depicting SLM Model

### 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.

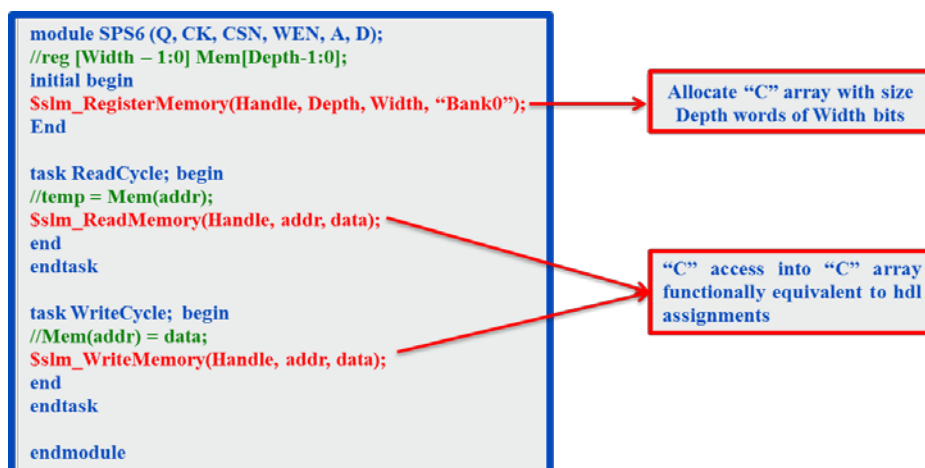


Fig. 1.5 C Procedure Calls from HDL

### 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.

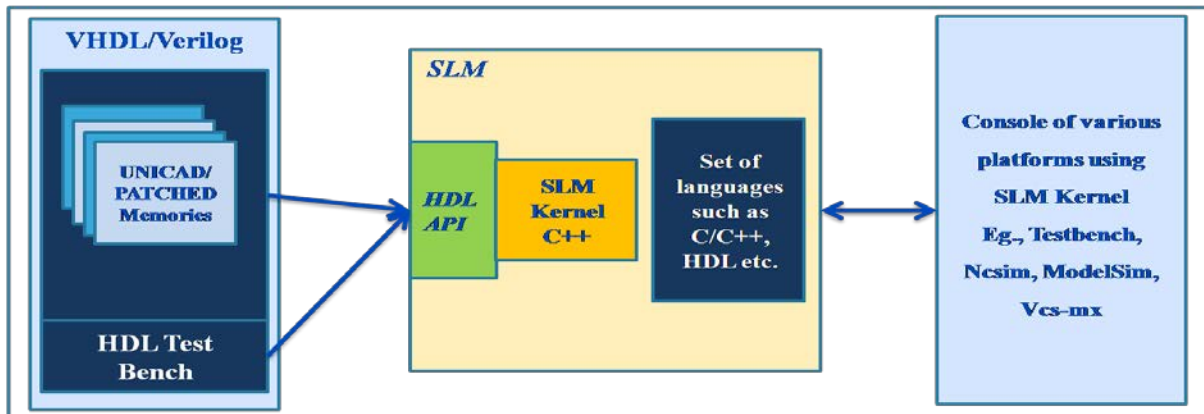
### 1.7.4 SLM Advantages

- 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 systematic 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.

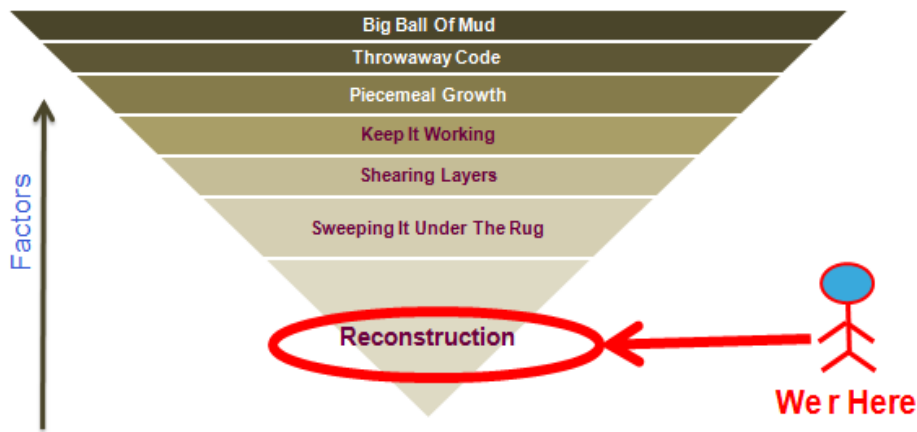
- **Structural Profile**

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.

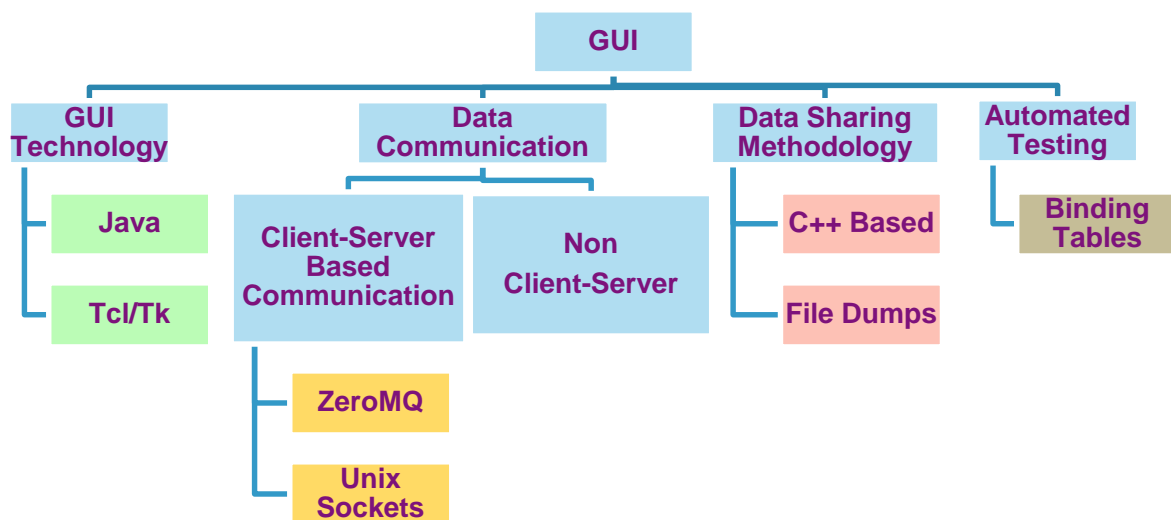


**Fig. 2.1 Inherent problem traced from current system**

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 integration 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 with features like inheritance, encapsulation, polymorphism, etc. Verilog and System Verilog both allow calling of C functions. While Verilog can do it using VPI or PLI, System 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 'veriusr.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.

```
System Verilog Code Fragment  
  
...  
if (a==1'b)  
    t= display_sample();  
...  
  
C Code Fragment  
  
void display_sample()  
{  
    printf("Hey I am C function accessible at Verilog\n");  
}
```

**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**

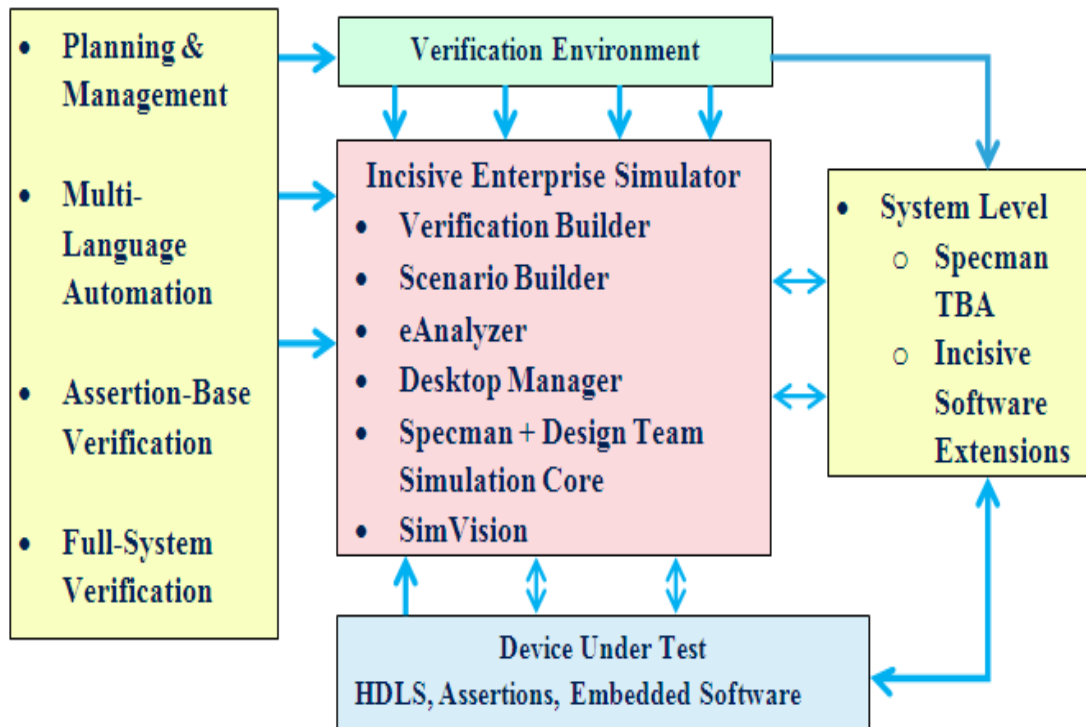
Immediate step after RTL coding is creation of test benches [22]. The test bench is used to inject a set of stimulus to the inputs of DUT, to check that the actual output conforms to the designers' expectation and system requirements. Once it is done, the simulators play the role to put together RTL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

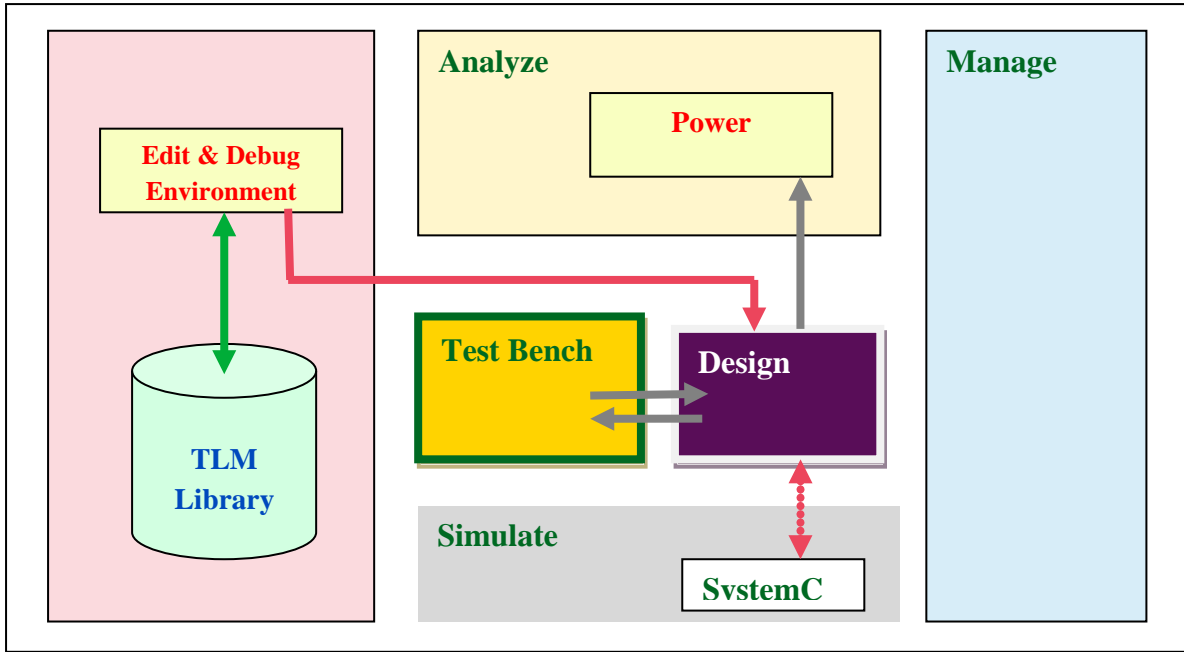


**Fig. 2.4 Comprehensive View of Incisive [23]**

### 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].

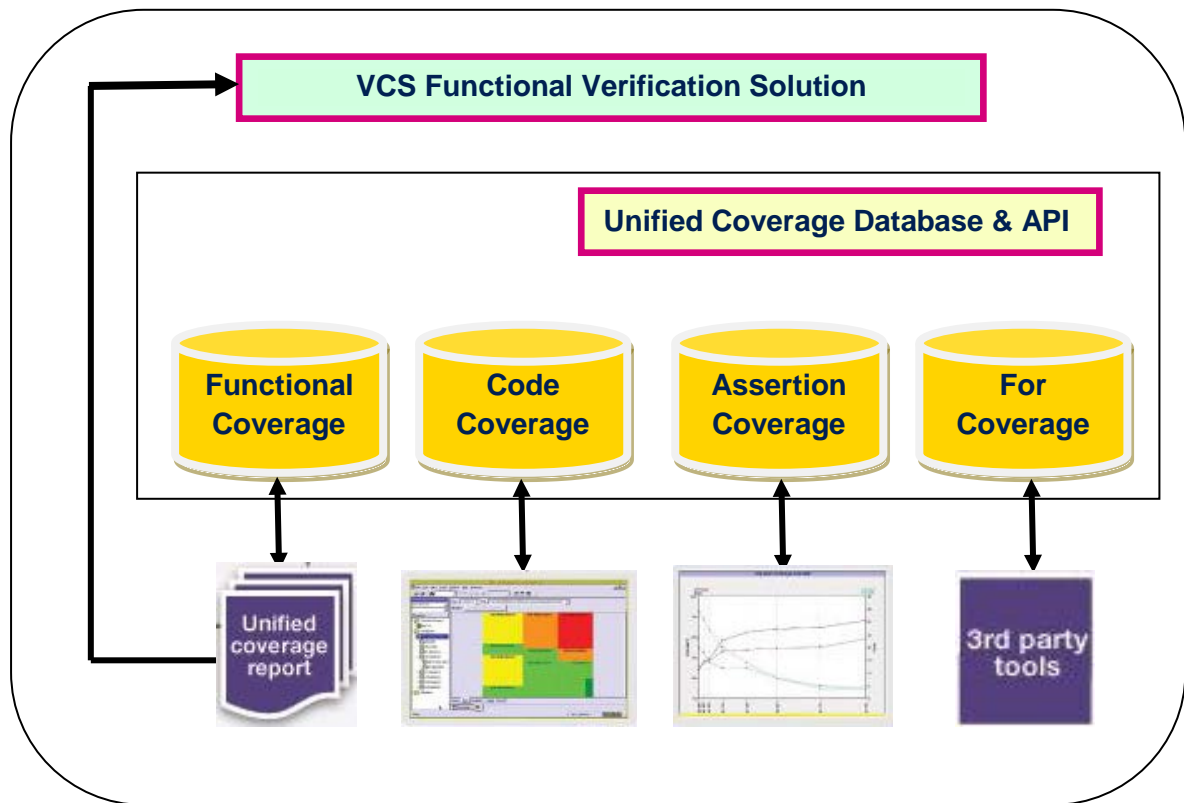


Fig. 2.6 Unified Coverage in VCS [25]

## Chapter-3

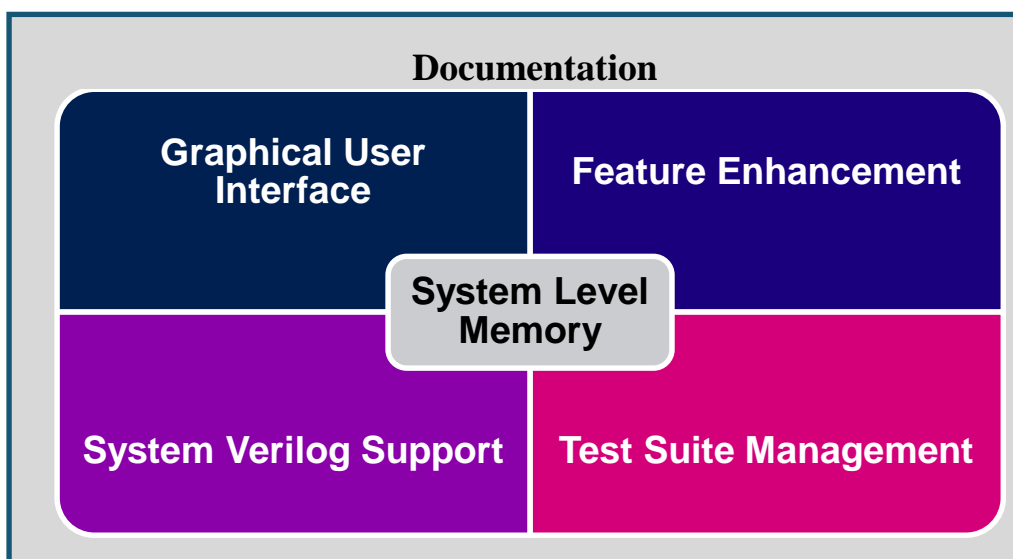
### Research Problem

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 maintenance 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 diagrammatically 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 accessibility 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 proprietary systems of organizations are at security risk, while subjected to testing by free tools or third-party licensed tools.

### 3.2 Objectives

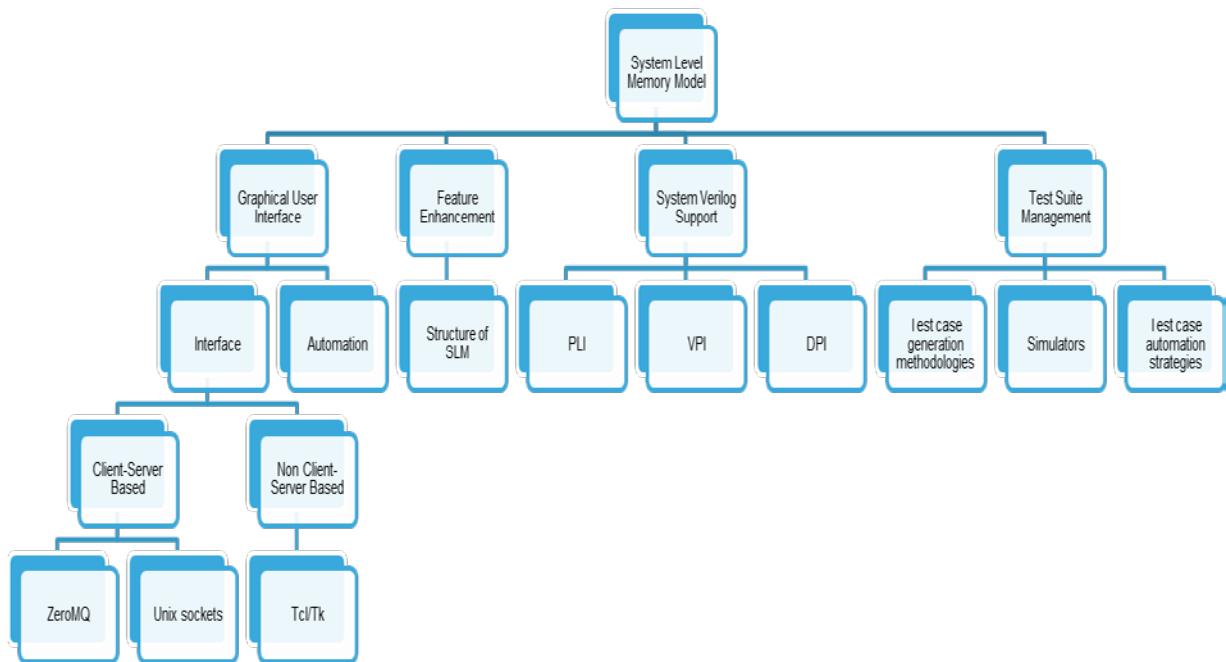
The main objectives of the research are :

- 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 options 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

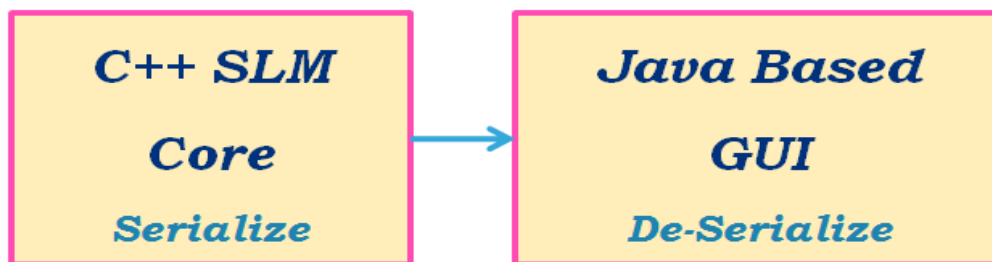


Fig. 4.1 Serialization based communication

##### b. C++ Object based

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

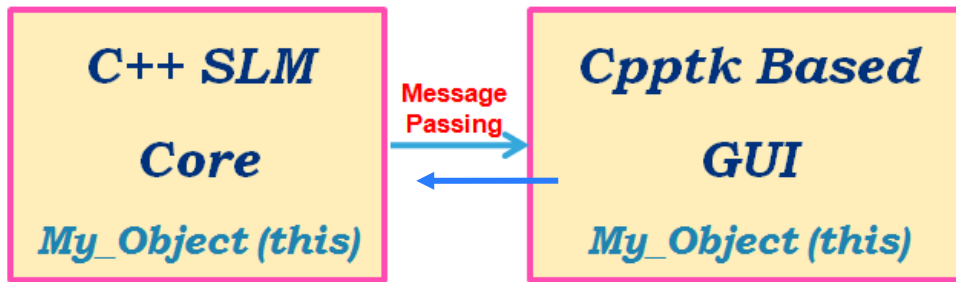


Fig. 4.2 C++ based communication

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

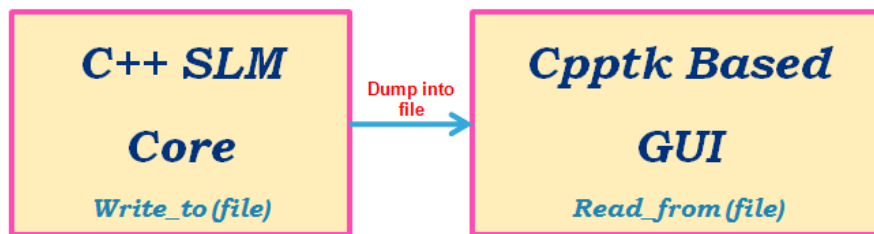


Fig. 4.3 File based communication

d. In-memory XML based

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

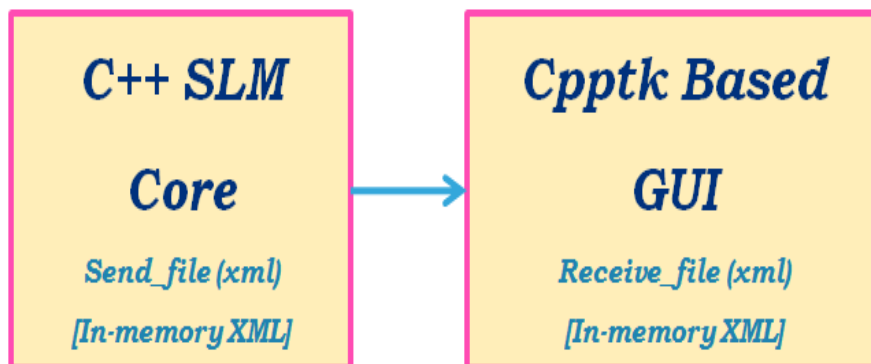


Fig. 4.4 In-memory XML based communication

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.

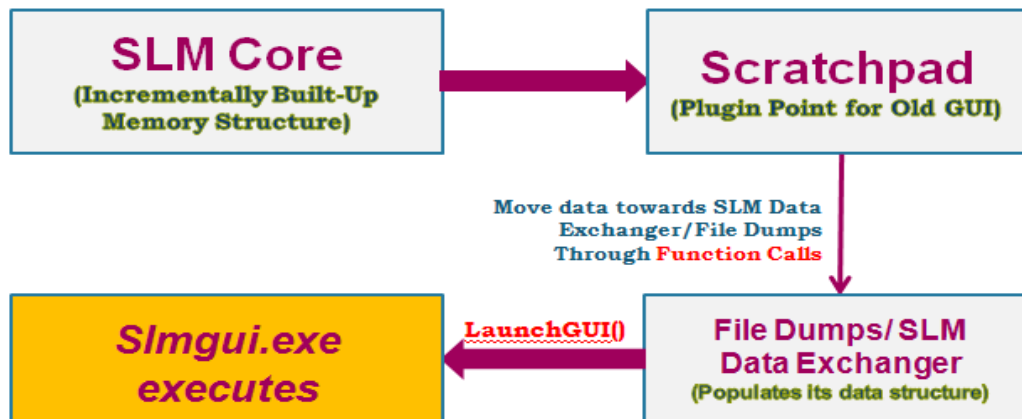


Fig. 4.5 SLM basic architecture of sub-systemss

### 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.

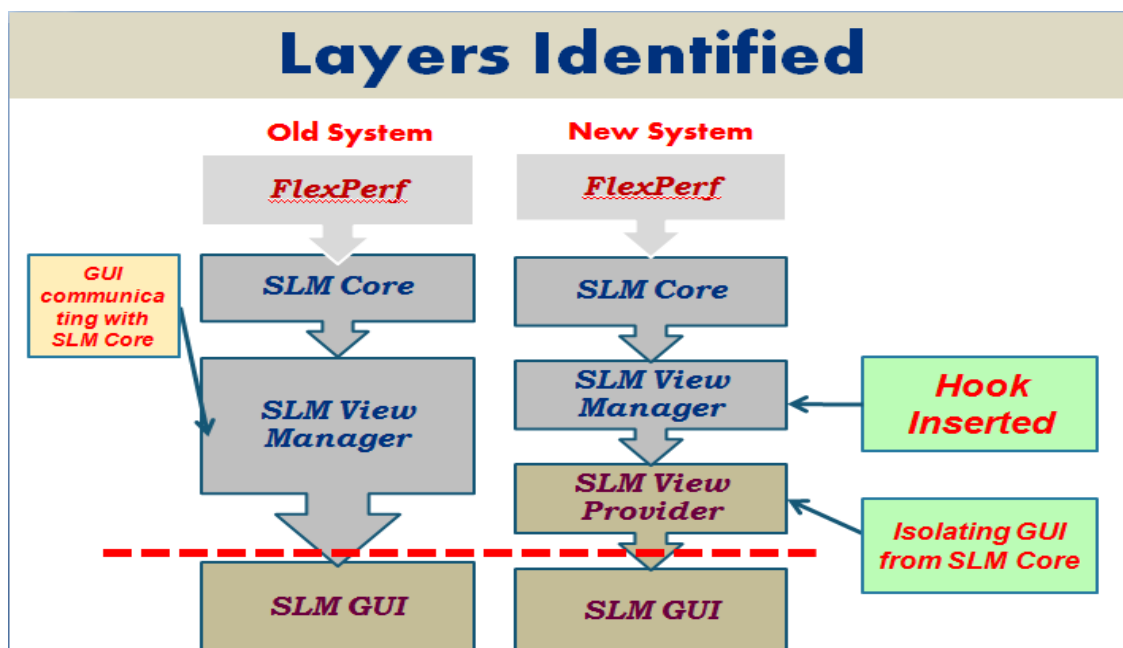


Fig. 4.6 Comparative Layered View of SLM

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.

```
1 /*****Purpose*****/
2 # Contains data structure shared between SLM Core and GUI through the DataExchanger
3 *****/
4 #include <pthread.h>
5
6 namespace SLMCoreGuiDataExchangeInfoSet
7 {
8     struct MemoryData
9     {
10         char* logical_name;
11         char* alias;
12         unsigned short mau;
13         unsigned short address_bus_size;
14         unsigned int nmau;
15         unsigned int begin_addr;
16         unsigned int end_addr;
17         unsigned short id;
18         unsigned int mem_type;
19         unsigned int width;
20         unsigned int depth;
21     };
22
23     struct Parameters
24     {
25         MemoryData* mem;
26         int son_count;
27     };
28
29     struct NewThread
30     {
31         pthread_t pt;
32         int thread_id;
33     };
34 }
```

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.

```
20
21
22 void setLogicalName(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, char* logicalName);
23 void setAlias(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, char* alias);
24 void setMau(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned short mau);
25 void setAddressBusSize(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned short address_bus_size);
26 void setNmau(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, int nmau);
27 void setBeginAddress(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int begin_addr);
28 void setEndAddress(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int end_addr);
29 void setId(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int id);
30 void setMemType(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int mem_type);
31 void setWidth(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int width);
32 void setDepth(SLMCoreGuiDataExchangeInfoSet::MemoryData mem, unsigned int depth);
33
34
69
70 //used by SLM View Manager to push data towards Provider and Provider will populate its own dataset through it.
71 void moveData(char* name, char* alias, unsigned int id, unsigned int width, unsigned int depth, unsigned short mau, unsigned short address_bus_size,
72             unsigned int nmau, unsigned int begin_addr, unsigned int end_addr, unsigned int mem_type, int i);
73
74 //used by SLM View Manager to instruct Provider that it should launch GUI which thus follows the instruction.
75 int launchGUI(int function_identifier);
76
77 };
78
```

Fig. 4.10 Behavior of SLM View Provider

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

```

34     SLMDataExchanger(int NoSons)
35     {
36         Sons_Count = NoSons;
37         dataset = new SLMCoreGuiDataExchangeInfoSet::MemoryData[Sons_Count];
38     }
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         GUIGlobalInfoSet::status << "Memory Structures initialized.\n\n";
1851
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 (strncap(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

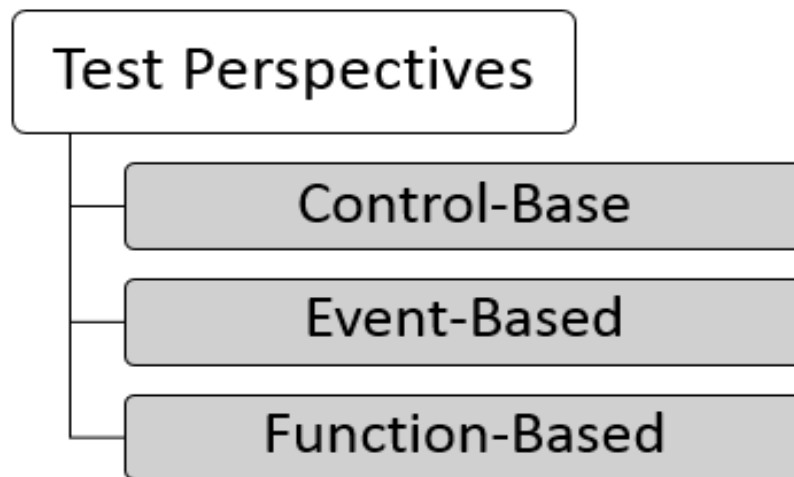
In this phase of exploration, we tried to construct a generic approach that can provide effectively a solution to GUI testing for in-house projects. Thus, here are some approaches which we explored and observed certain pros and cons.

- **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, we chose 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 responsive participation 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 development phase, so that system can be made more responsive in its testing phase. A diagrammatical representation to depict how those helping structures complement automated testing process is given in Fig. 5.6.

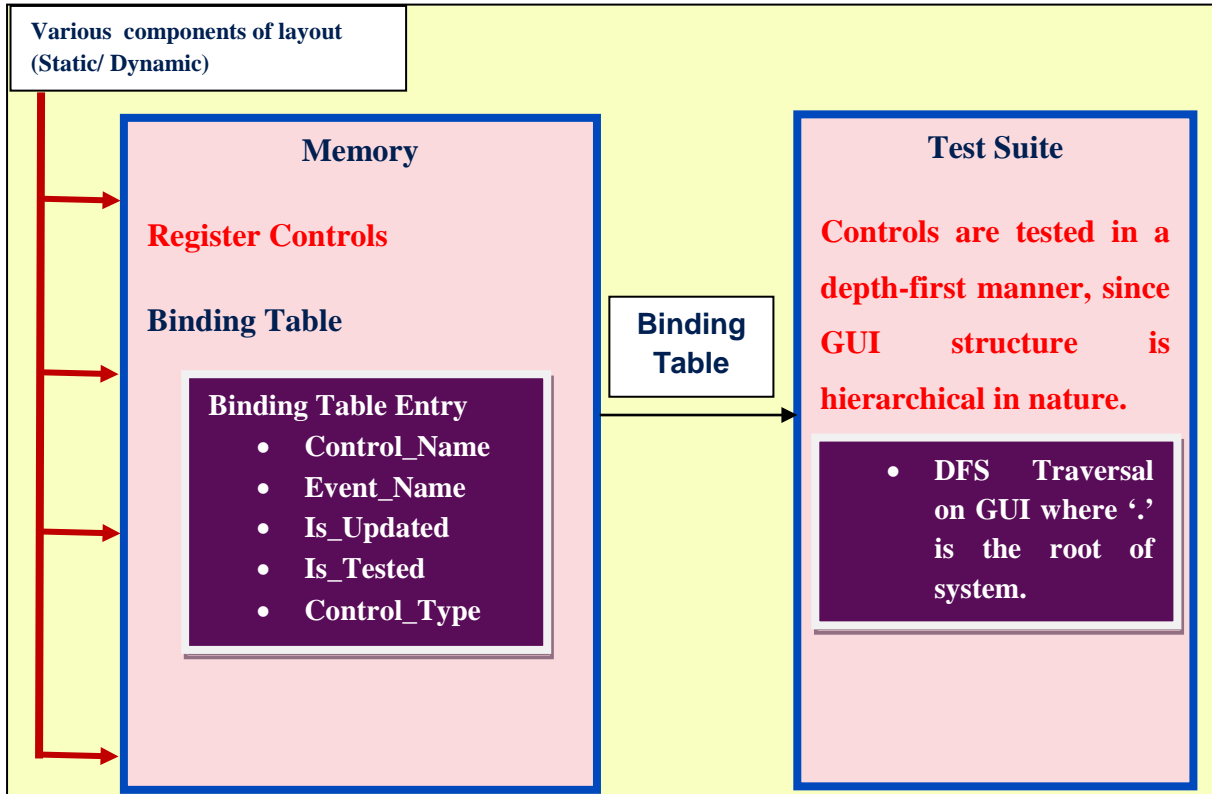


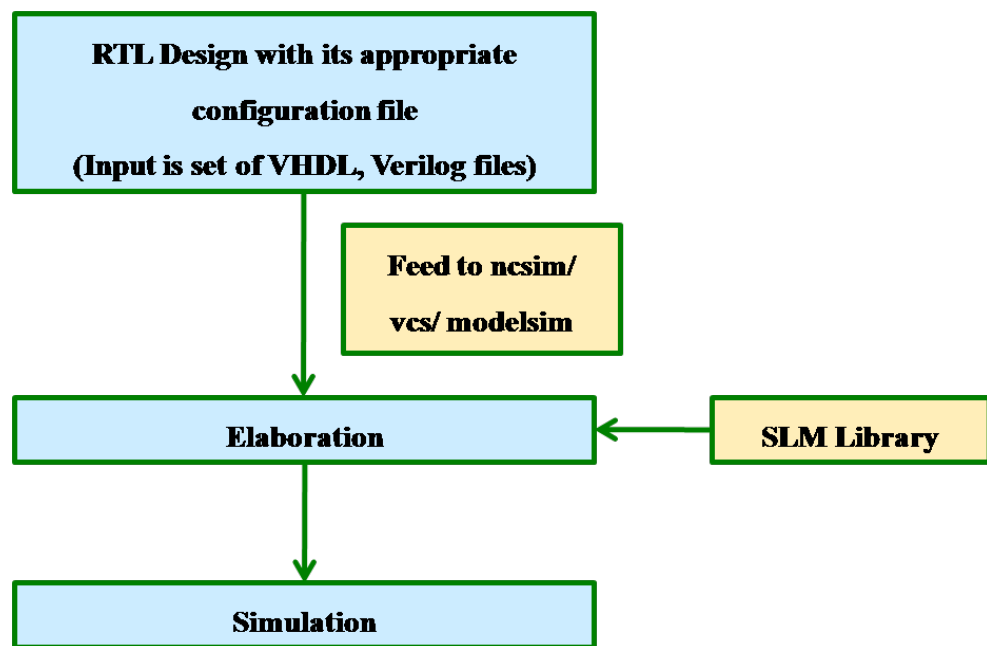
Fig. 5.6 System Architecture for GUI Test Suite

## 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 :

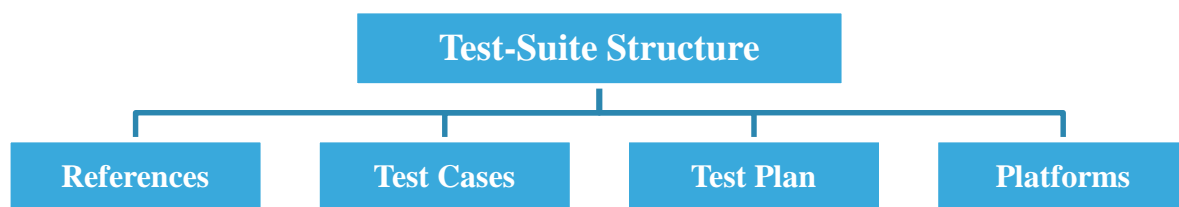
**Table. 6.1 Categories of test plans in the test suite**

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
  
```

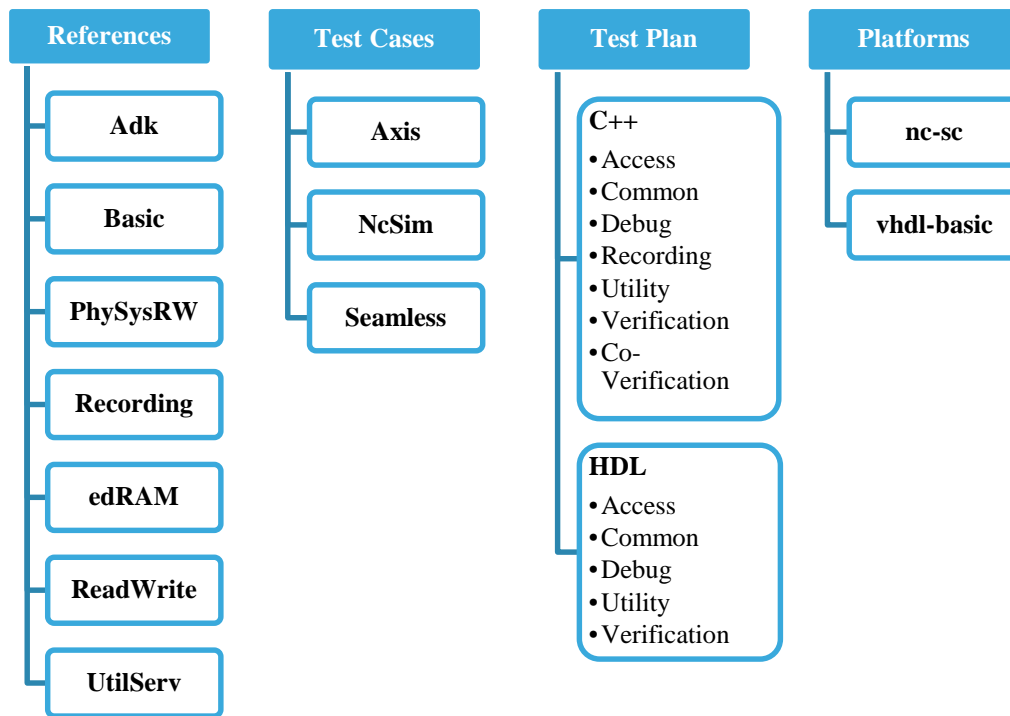
**Fig. 6.3(a) References-Golden Log**

```

dlh12117{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
  
```

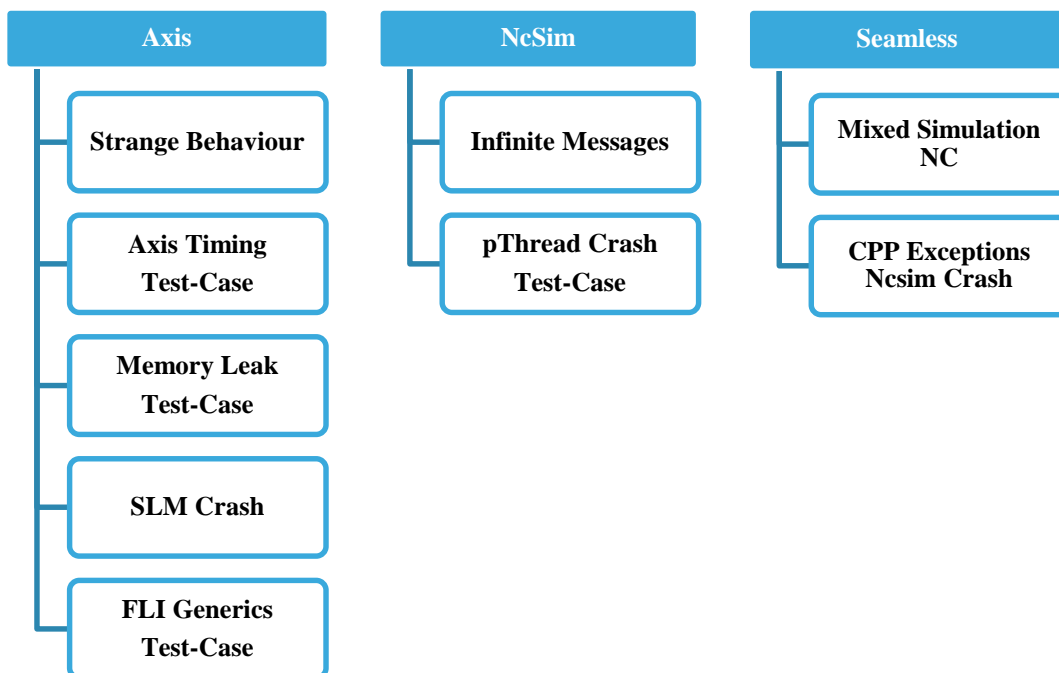
**Fig. 6.3(b) Reference-Runtime Log**

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.

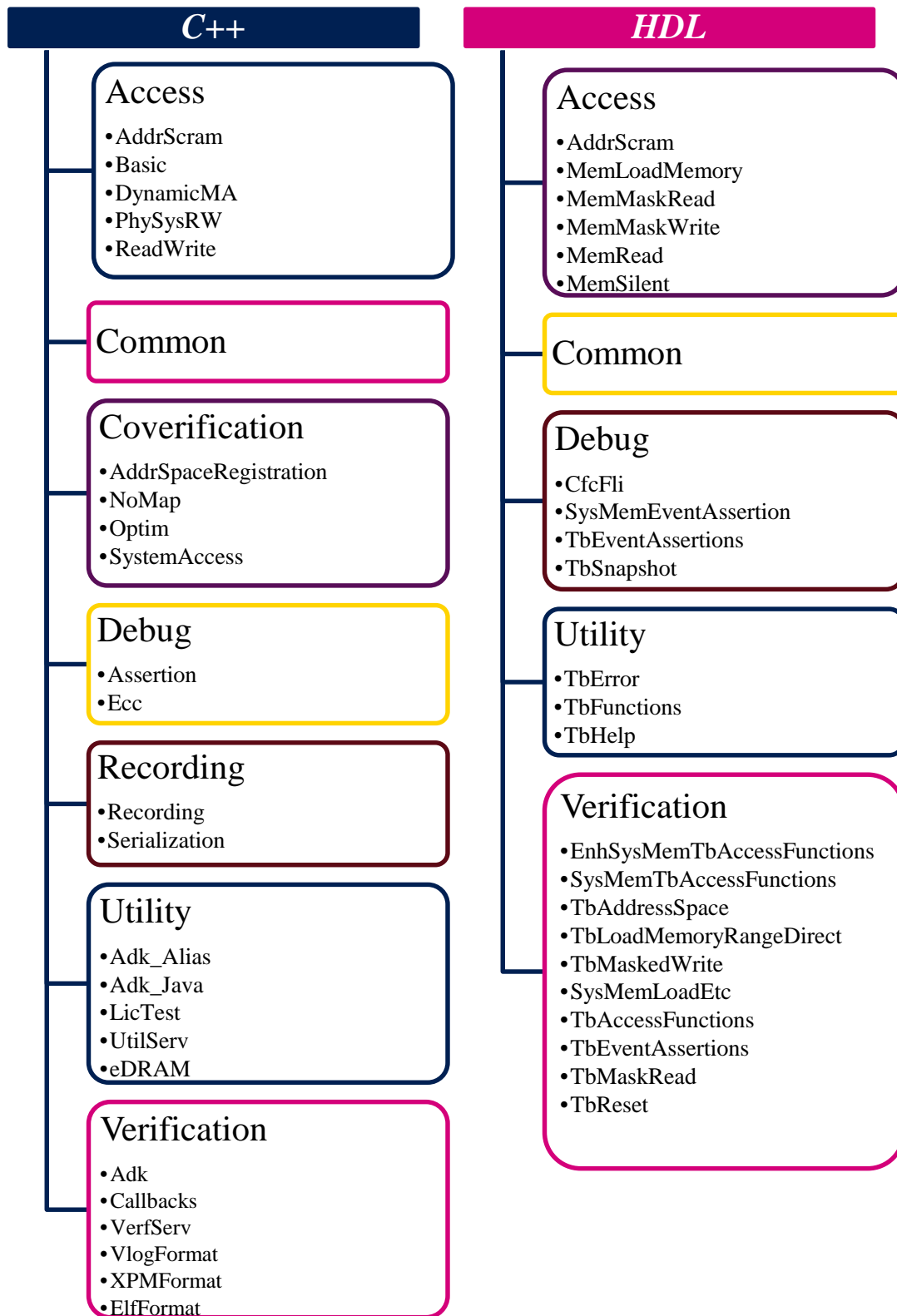


Fig. 6.6 Detailed Test Plan Structure

## 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 backdoor 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 verification 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 systematic 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.

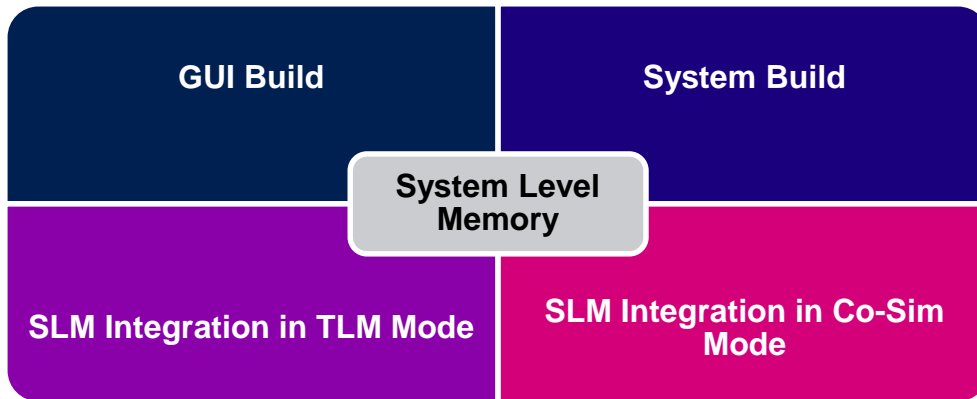


Fig. 7.1 Various sections in SLM Environment Setup

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.

```
Makefile (/data/SPG/users/goels2/tcl_file/SLM_GUI/integration/GUI_SLM) - GVIM6
1 | COMPILER = g++
2 | OPTIONS = -Wall -fPIC -shared -kno-long-long -pedantic -DCUSTOM_FLAG -DTEST_MODE
3
4 | INCLUDES = -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/systemc-2.3.1/
  src/sysc/packages/boost -I/data/SPG/users/goels2/tcl_file/SLM_GUI/integration/GUI_SLM
5
6 | LIBDIRS = -L/data/SLM/install/Tcltk_8.5.16/lib/
7
8 | LIBS = -ltk8.5 -ltcl8.5 -lslagui
9
10 | ALL_OPTIONS = $(OPTIONS) $(INCLUDES) $(LIBDIRS) -ltk8.5 -ltcl8.5
11
12 | lib_file:libslagui
13
14 | libslagui: slagui.cpp cpptk.o cpptkbase.o
  $(COMPILER) slagui.cpp slagui.h /data/SPG/users/goels2/tcl_file/SLM_GUI/install/cpptk-1.0.2/cpptk.cc /data/SPG/users/goels2/tcl_file/SLM_GUI/install/
  cpptk-1.0.2/base/cpptkbase.cc -o $@.so $(ALL_OPTIONS)
16
17 | cpptk.o: /data/SPG/users/goels2/tcl_file/SLM_GUI/install/cpptk-1.0.2/cpptk.cc
  $(COMPILER) $? -o $@ -c $(OPTIONS) $(INCLUDES)
19
20 | cpptkbase.o: /data/SPG/users/goels2/tcl_file/SLM_GUI/install/cpptk-1.0.2/base/cpptkbase.cc
  $(COMPILER) $? -o $@ -c $(OPTIONS) $(INCLUDES)
22
23 | clean:
24 |     make out_clean
25 |     make exe_clean
26 |     make lib_clean
27
28 | out_clean :
29 |     rm -rf *.o
30
31 | exe_file: slagui.cpp cpptk.o cpptkbase.o
  $(COMPILER) slagui.cpp cpptk.o cpptkbase.o -o slagui -Wall -kno-long-long -pedantic -DCUSTOM_FLAG -DTEST_MODE $(INCLUDES) $(LIBDIRS) -ltk8.5 -ltcl8.5
33
34 | main_file: main_file.cpp slagui.cpp cpptk.o cpptkbase.o
  $(COMPILER) main_file.cpp slagui.cpp slagui.h -o main_file -DCUSTOM_FLAG -D_ENABLE_CPP_MAIN_ENTRY_FUNCTION_ $(INCLUDES) $(LIBDIRS) -ltcl8.5 -ltk8.5
36
37 | exe_clean :
38 |     rm -rf slagui
39 |     rm -rf main_file
40
41 | lib_clean:
42 |     rm -rf *.so
43
```

Fig. 7.2 Makefile for GUI Build

```

SLM GUI NEW
File Edit View Terminal Tabs Help
TLM_TEST_PLT x Verilog x temp x SLM_NEW_BUILD x SLM GUI NEW x Terminal x SLM 6.0 lab x Terminal x
d\h12117\goels2\3723:ls -l
total 2184
-rw-r--r-- 1 goels2 spg 142 May 14 10:32 Component.cpp
-rw-r--r-- 1 goels2 spg 5631 May 12 10:34 DataDesign.h
-rw-r--r-- 1 goels2 spg 1576 Jun 4 10:05 Makefile
-rw-r--r-- 1 goels2 spg 0 Jun 8 16:26 Memory_top_plt.DATA_memory_2015-06-08.16:26:56
-rw-r--r-- 1 goels2 spg 787 Jun 1 12:55 SlmCoreGuiDataExchange.h
-rw-r--r-- 1 goels2 spg 3815 Jun 9 16:53 Status.log
-rw-r--r-- 1 goels2 spg 2639 Jun 9 16:53 Test_Case.log
-rw-r--r-- 1 goels2 spg 59613 Jun 8 15:05 \
-rw-r--r-- 1 goels2 spg 339864 Jun 2 16:39 cpptk.o
-rw-r--r-- 1 goels2 spg 310920 Jun 2 16:39 cpptkbase.o
drwxr-xr-x 2 goels2 spg 4096 Jun 2 16:57 doc
-rw-r--r-- 1 goels2 spg 2227 May 22 16:08 guitest.cpp
-rw-r--r-- 1 goels2 spg 1055 May 27 16:34 init_func.cpp
-rwxr-xr-x 1 goels2 spg 677186 Jun 4 10:02 libslmgui.so
-rw-r--r-- 1 goels2 spg 1432 Jun 1 17:29 main_file.cpp
-rw-r--r-- 1 goels2 spg 13728 Mar 20 15:05 memory1.txt
-rwxr-xr-x 1 goels2 spg 687856 Jun 9 09:46 slmgui
-rwxr--r-- 1 goels2 spg 61118 Jun 9 16:40 slmgui.cpp
-rw-r--r-- 1 goels2 spg 5576 Jun 8 17:47 slmgui.h

```

**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 #}}}
```

**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**

```
214
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**



### 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.

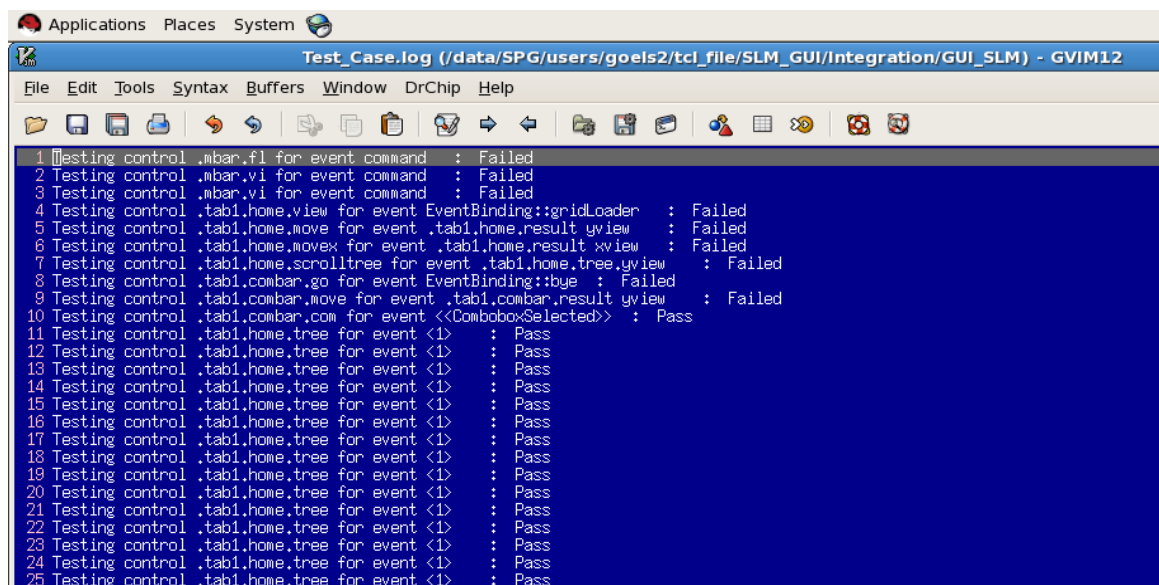


Fig. 7.12 Test\_Case.log

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

```
1-----Initializing Dummy SLM-----
2 Memory Structures initialized.
3
4-----Extracting core information for GUI Builder-----
5 slgui.h: 196: getInstance()
6 slgui.h: 171: CoreInfo()
7 slgui.h: 175: CoreInfo()
8
9-----Configuring Menu-----
10 slgui.cpp: 848: addMenu()
11 slgui.cpp: 853: addMenu()
12 slgui.cpp: 859: addMenu()
13 slgui.cpp: 901: addControllLayout Grid_configure
14
15 slgui.cpp: 953: addControllLayout Control Layout evaluation tab-1 completed.
16
17 Moving to tab-2.. calling SIMUserApiInterfacesIsgui.h: 181: CoreInfo()
18 slgui.cpp: 1578: guiStarter()
19
20-----Building GUI-----
21 Data recovered... top_plt.DATA_memory sc_main.top_plt.demo_csw_inst_0.demo_inst_0.slave_M0
22
23 Activating child view by default...
24 Loading system instances and daughter instances...
25 slgui.cpp: 1497: populateTree Length evaluated
26
27 slgui.cpp: 1518: populateTree Adding tags
28
29 slgui.cpp: 1518: populateTree Adding tags
30
31-----Test mode activated-----
32
33
34 Control - .mbar.fl registered for event - command
35
36 Control - .mbar.vi registered for event - command
37
38 Control - .mbar.vi registered for event - command
39
40 Control - .mbar.vi registered for event - command
41
42 Control - .tab1.home.err_rep registered for event - None
43
44 Control - .tab1.home.view registered for event - EventBinding::gridLoader
45
46 Control - .tab1.home.status registered for event - None
47
48 Control - root registered for event - <D>
49
50 Control - root registered for event - <KeyRelease>
51
52 Control - .tab1.home.result registered for event - .tab1.home.move set
53
54 Control - .tab1.home.result registered for event - .tab1.home.moveX set
55 []
```

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.

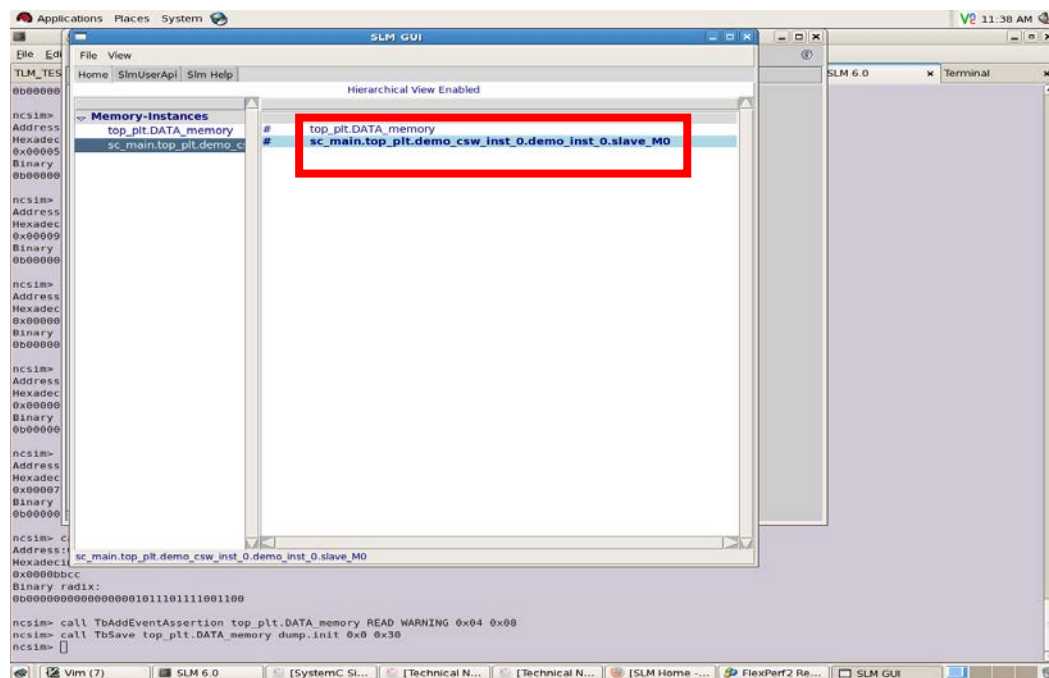
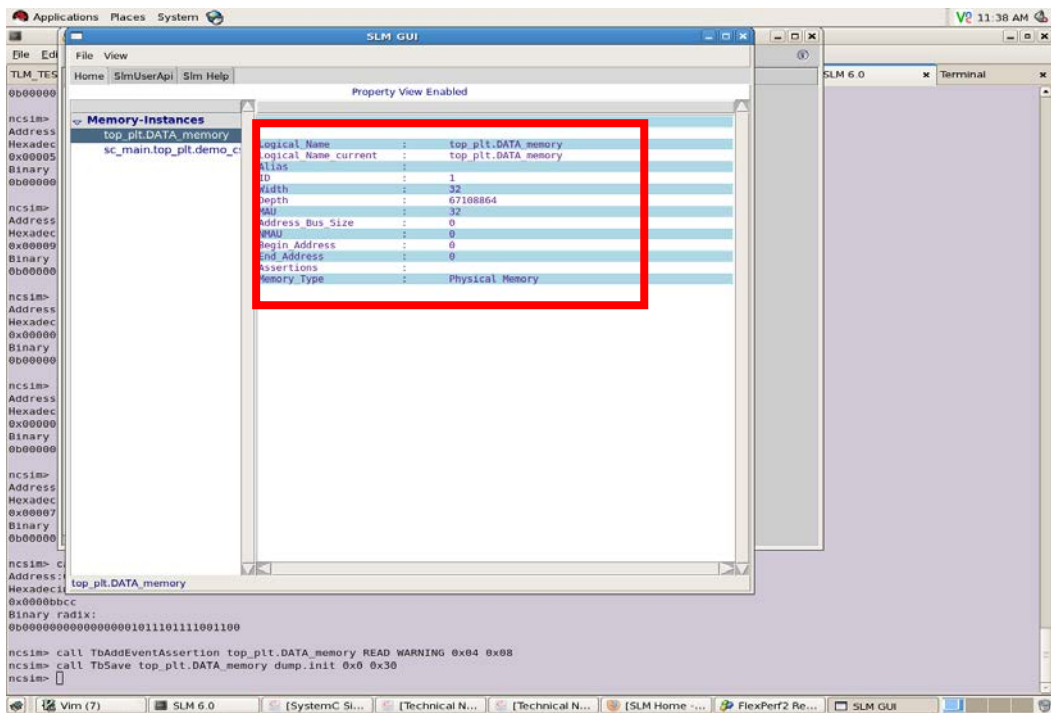


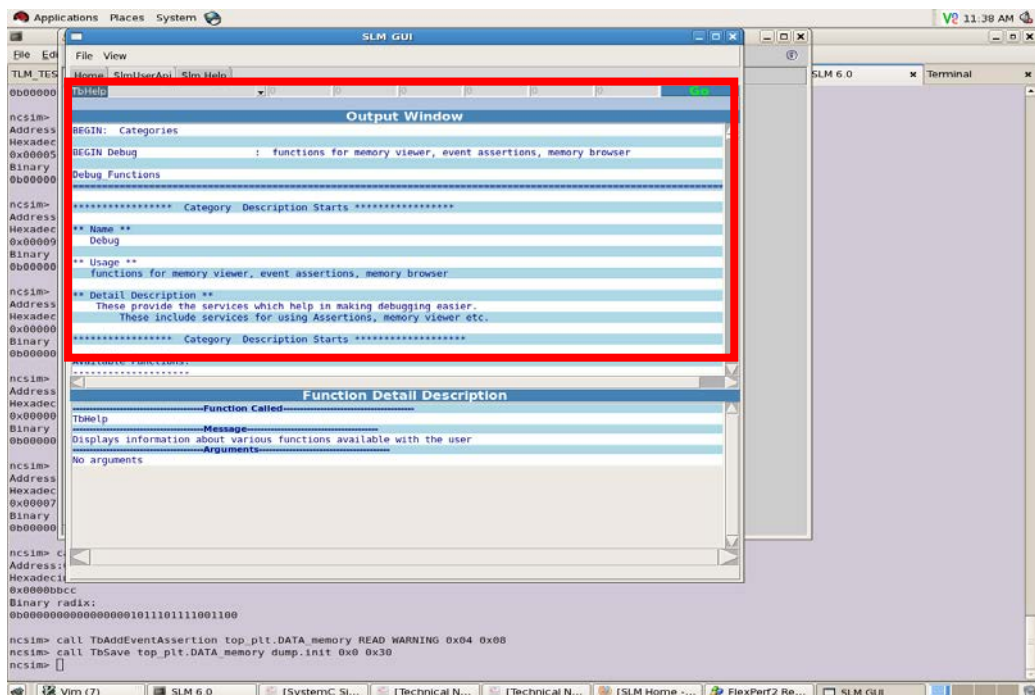
Fig. 7.14 Splash Screen

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



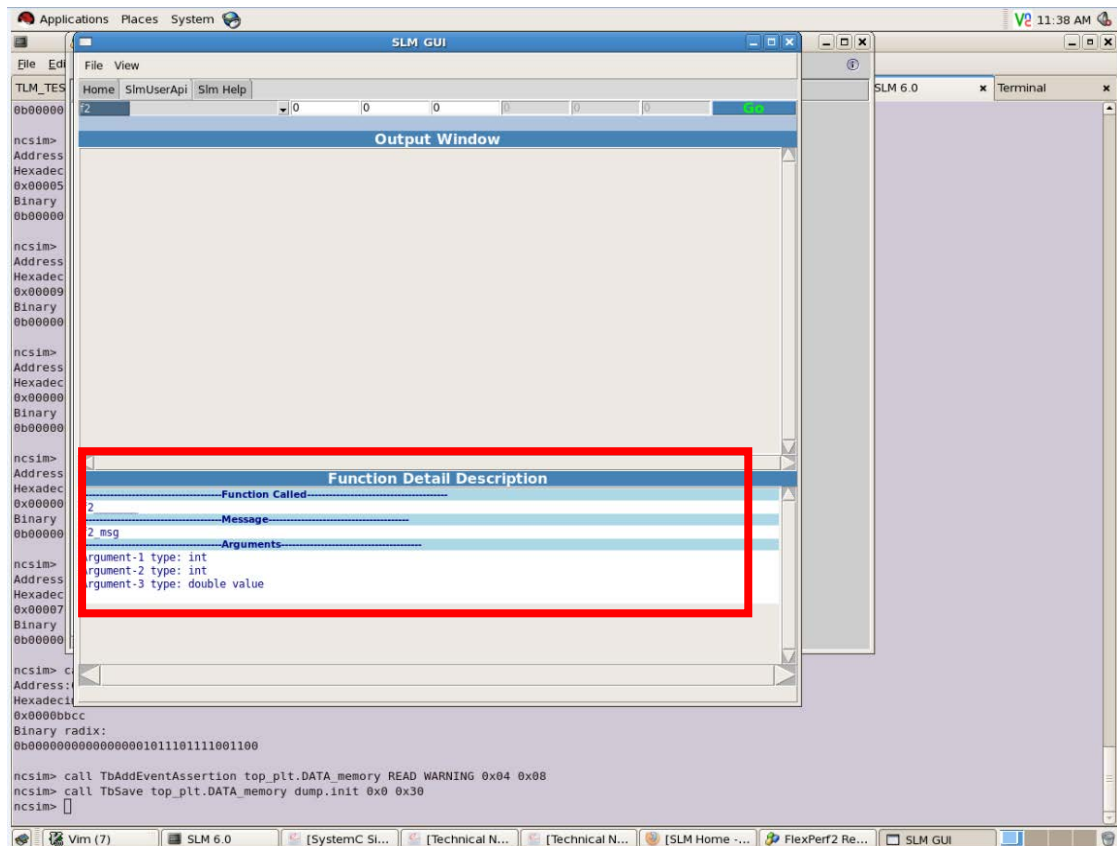
**Fig. 7.15 TbInfo Screen**

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.



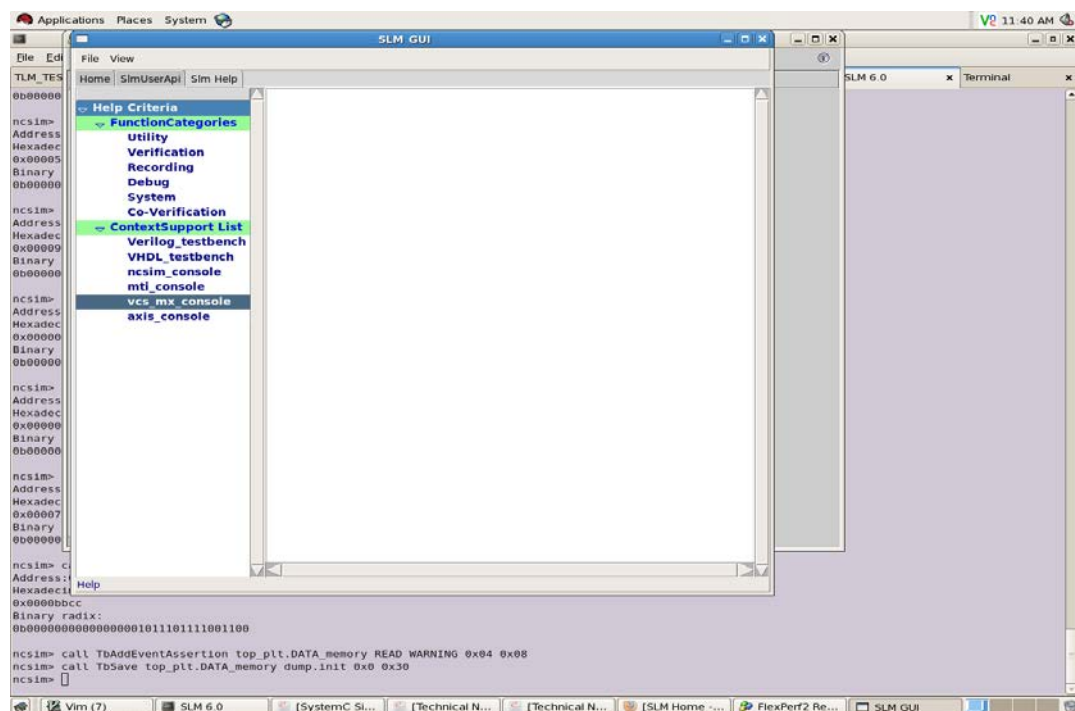
**Fig. 7.16(a) Function Mapper Screen-1**

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.

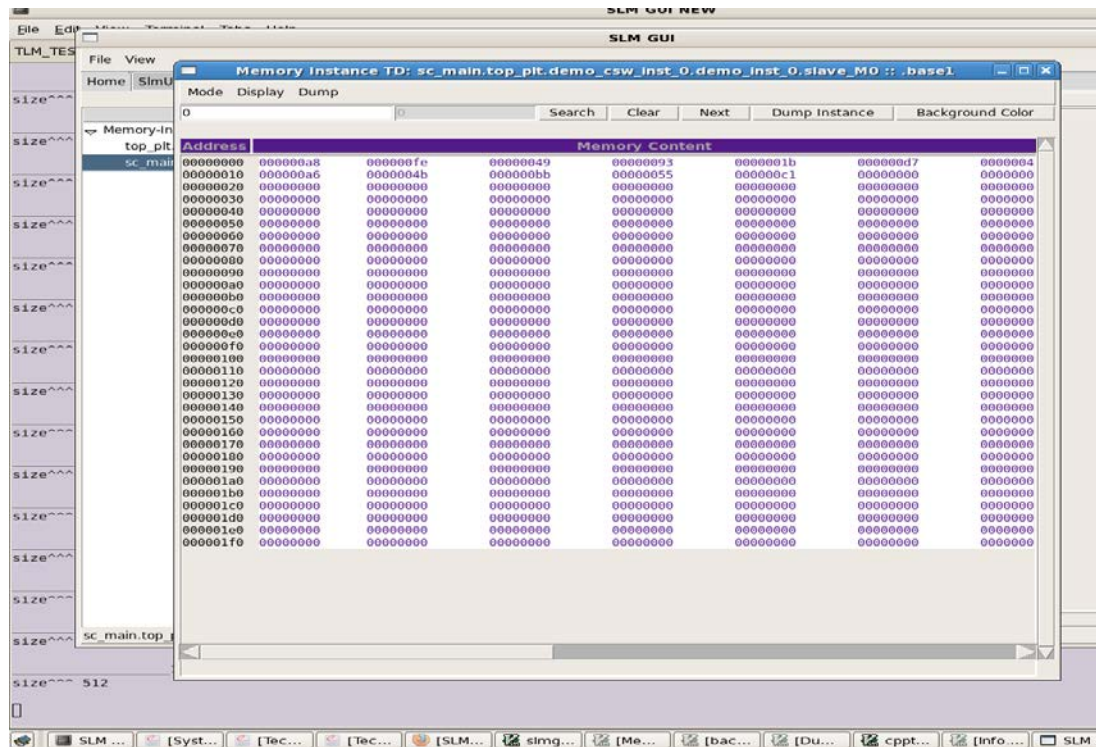


Fig. 7.18 Memory Grid

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.

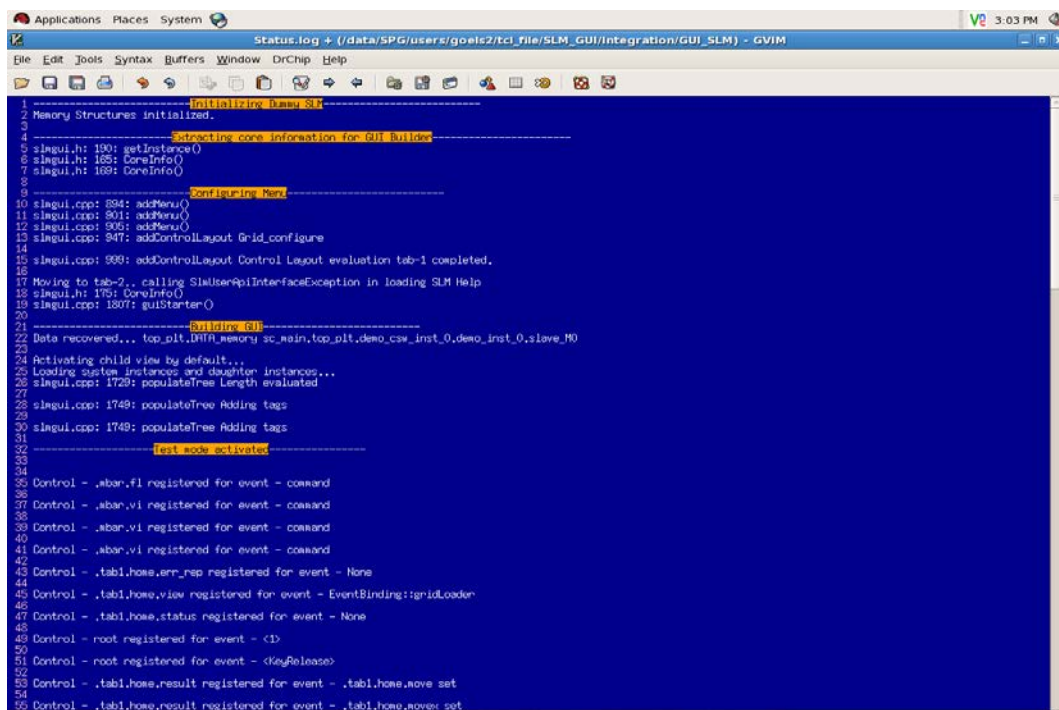


Fig. 7.19(a) GUI Running Log-1

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

```

Applications Places System
Status.log + (/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM) - GVIM
File Edit Tools Syntax Buffers Window DrChip Help
79 Control - .tab1.combar.ptype0 registered for event - None
80
81 Control - .tab1.combar.ptype1 registered for event - None
82
83 Control - .tab1.combar.ptype2 registered for event - None
84
85 Control - .tab1.combar.ptype3 registered for event - None
86
87 Control - .tab1.combar.ptype4 registered for event - None
88
89 Control - .tab1.combar.ptype5 registered for event - None
90
91 Control - .tab1.combar.output registered for event - None
92
93 Control - .tab1.combar.output1 registered for event - None
94
95 Control - .tab1.combar.err_rep registered for event - None
96
97 Control - .tab1.combar.result registered for event - None
98
99 Control - .tab1.combar.details registered for event - None
100
101 Control - .tab1.combar.move registered for event - .tab1.combar.result yview
102
103 Control - .tab1.combar.movex registered for event - .tab1.combar.result xview
104
105 Control - .tab1.combar.scroll registered for event - .tab1.combar.details yview
106
107 Control - .tab1.combar.scrollx registered for event - .tab1.combar.details xview
108
109 Control - .tab1.combar.com registered for event - <<ComboboxSelected>
110
111 Control - .tab1.help.scrolltree registered for event - .tab1.help.tree.yview
112
113 Control - .tab1.help.status registered for event - None
114
115 Control - .tab1.help.result registered for event - .tab1.help.move set
116
117 Control - .tab1.help.result registered for event - .tab1.help.movex set
118
119 Control - .tab1.help.move registered for event - .tab1.help.result yview
120
121 Control - .tab1.help.movex registered for event - .tab1.help.result xview
122
123 Control - child0 registered for event - <D>
124
125 Control - child0 registered for event - <KeyRelease>
126
127 Control - child1 registered for event - <D>
128
129 Control - child1 registered for event - <KeyRelease>
130 48 events registered
  
```

Fig. 7.19(b) GUI Running Log-2

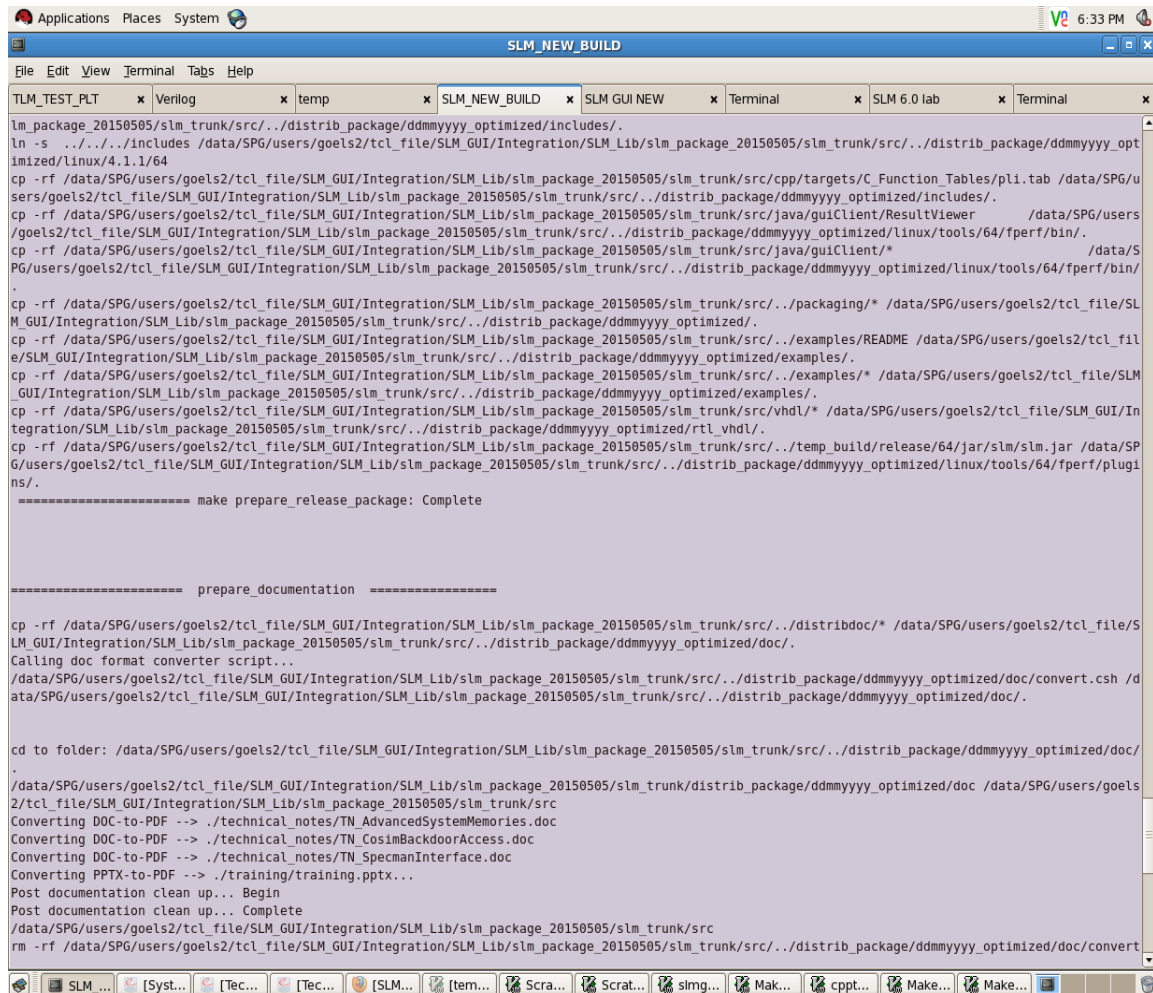
```

Applications Places System
Test_Case.log (/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM) - GVIM12
File Edit Tools Syntax Buffers Window DrChip Help
1 Testing control .mbar.fl for event command : Failed
2 Testing control .mbar.vi for event command : Failed
3 Testing control .mbar.vi for event command : Failed
4 Testing control .tab1.home.view for event EventBinding::grid_loader : Failed
5 Testing control .tab1.home.move for event .tab1.home.result yview : Failed
6 Testing control .tab1.home.movex for event .tab1.home.result xview : Failed
7 Testing control .tab1.home.scrolltree for event .tab1.home.tree.yview : Failed
8 Testing control .tab1.combar.go for event EventBinding::bye : Failed
9 Testing control .tab1.combar.move for event .tab1.combar.result yview : Failed
10 Testing control .tab1.combar.com for event <<ComboboxSelected> : Pass
11 Testing control .tab1.home.tree for event <D> : Pass
12 Testing control .tab1.home.tree for event <D> : Pass
13 Testing control .tab1.home.tree for event <D> : Pass
14 Testing control .tab1.home.tree for event <D> : Pass
15 Testing control .tab1.home.tree for event <D> : Pass
16 Testing control .tab1.home.tree for event <D> : Pass
17 Testing control .tab1.home.tree for event <D> : Pass
18 Testing control .tab1.home.tree for event <D> : Pass
19 Testing control .tab1.home.tree for event <D> : Pass
20 Testing control .tab1.home.tree for event <D> : Pass
21 Testing control .tab1.home.tree for event <D> : Pass
22 Testing control .tab1.home.tree for event <D> : Pass
23 Testing control .tab1.home.tree for event <D> : Pass
24 Testing control .tab1.home.tree for event <D> : Pass
25 Testing control .tab1.home.tree for event <D> : Pass
  
```

Fig. 7.19(c) GUI Running Log-3

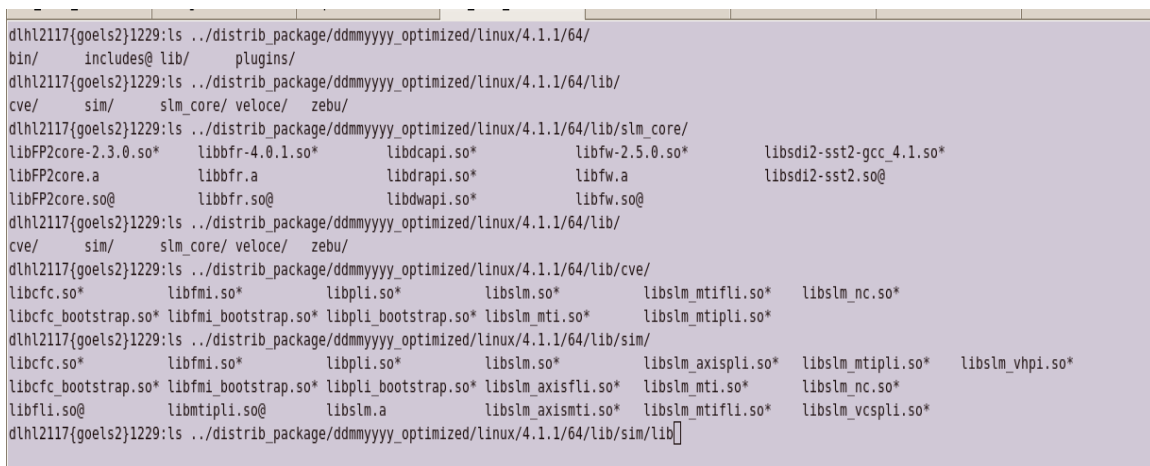
## 2. SLM Build

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.



```
Applications Places System 6:33 PM
SLM_NEW_BUILD
File Edit View Terminal Tabs Help
TLM_TEST_PLT x Verilog x temp x SLM_NEW_BUILD x SLM GUI NEW x Terminal x SLM 6.0 lab x Terminal x
lm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/includes/.
ln -s ../.././includes /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/linux/4.1.1/64
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/targets/C_Function_Tables/pli.tab /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/includes/.
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/java/guiClient/ResultViewer /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/linux/tools/64/fperf/bin/.
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/java/guiClient/* /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/linux/tools/64/fperf/bin/.
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./packaging/* /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/.
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./examples/README /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/examples/.
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./examples/* /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/examples/.
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/vhdl/* /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/rtl_vhdl/.
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./temp_build/release/64/jar/slm/slm.jar /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/linux/tools/64/fperf/plugins/.
===== make prepare_release_package: Complete
===== prepare_documentation =====
cp -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distribdoc/* /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/doc/.
Calling doc format converter script...
/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/doc/convert.csh /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/doc/.
cd to folder: /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/doc/.
/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/distrib_package/ddmmyyyy_optimized/doc /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src
Converting DOC-to-PDF -> ./technical_notes/TN_AdvancedSystemMemories.doc
Converting DOC-to-PDF -> ./technical_notes/TN_CosimBackdoorAccess.doc
Converting DOC-to-PDF -> ./technical_notes/TN_SpecmanInterface.doc
Converting PPTX-to-PDF -> ./training/training.pptx...
Post documentation clean up... Begin
Post documentation clean up... Complete
/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src
rm -rf /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./distrib_package/ddmmyyyy_optimized/doc/convert
```

Fig. 7.20(a) SLM Build Terminal Output



```
d\hl2117[goels2]1229:ls ../distrib_package/ddmmyyyy_optimized/linux/4.1.1/64/
bin/ includes@ lib/ plugins/
d\hl2117[goels2]1229:ls ../distrib_package/ddmmyyyy_optimized/linux/4.1.1/64/lib/
cve/ sim/ slm_core/ veloce/ zebu/
d\hl2117[goels2]1229:ls ../distrib_package/ddmmyyyy_optimized/linux/4.1.1/64/lib/slm_core/
libFP2core-2.3.0.so* libbfr-4.0.1.so* libdcapi.so* libfw-2.5.0.so* libsd2-sst2-gcc_4.1.so*
libFP2core.a libbfr.a libdrapi.so* libfw.a libsd2-sst2.so@
libFP2core.so@ libbfr.so@ libdwapi.so* libfw.so@
d\hl2117[goels2]1229:ls ../distrib_package/ddmmyyyy_optimized/linux/4.1.1/64/lib/
cve/ sim/ slm_core/ veloce/ zebu/
d\hl2117[goels2]1229:ls ../distrib_package/ddmmyyyy_optimized/linux/4.1.1/64/lib/cve/
libcfc.so* libfmi.so* libpli.so* libslm.so* libslm_mtifli.so* libslm_nc.so*
libcfc_bootstrap.so* libfmi_bootstrap.so* libpli_bootstrap.so* libslm_mti.so* libslm_mtipli.so*
d\hl2117[goels2]1229:ls ../distrib_package/ddmmyyyy_optimized/linux/4.1.1/64/lib/sim/
libcfc.so* libfmi.so* libpli.so* libslm.so* libslm_axispli.so* libslm_mtipli.so* libslm_vhpi.so*
libcfc_bootstrap.so* libfmi_bootstrap.so* libpli_bootstrap.so* libslm_axisfli.so* libslm_mti.so* libslm_nc.so*
libfli.so@ libmtipli.so@ libslm.a libslm_axismti.so* libslm_mtifli.so* libslm_vcspli.so*
d\hl2117[goels2]1229:ls ../distrib_package/ddmmyyyy_optimized/linux/4.1.1/64/lib/sim/lib/
```

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.

```

TLM_TEST_PLT
Verilog temp SLM_NEW_BUILD SLM GUI NEW Terminal SLM 6.0 lab Terminal
DEBUG TOP_MEMORY2: Received write block request T:0.001252104
DEBUG TOP_MEMORY2: Write block(16) at 0x000006A4
Data:
0x000001A0 0x000001AA 0x000001AB 0x000001AC
0x000001AD 0x000001AE 0x000001AF 0x000001B0
0x000001B1 0x000001B2 0x000001B3 0x000001B4
0x000001B5 0x000001B6 0x000001B7 0x000001B8
from TOP_MASTER2.traffic_generator_func - T:0.001252104
DEBUG TOP_MEMORY2: Transaction details at T:0.001252104:
From Initiator thread: *TOP_MASTER2.traffic_generator_func
Address: 0x000006A4 - Number: 16 (nb byte: 64) - Access: WRITE
Data: 0x000001A9 0x000001AA 0x000001AB 0x000001AC 0x000001AD 0x000001AE 0x000001AF 0x000001B0 0x000001B1 0x000001B2 0x000001B3
0x000001B4 0x000001B5 0x000001B6 0x000001B7 0x000001B8
Access Mode: DEFAULT - Transaction status: SUCCESS
DEBUG TOP_MEMORY1: Received read request T:0.001252856
DEBUG TOP_MEMORY2: Write block(16) at 0x00000788
Data:
0x000001E2 0x000001E3 0x000001E4 0x000001E5
0x000001E6 0x000001E7 0x000001E8 0x000001E9
0x000001EA 0x000001EB 0x000001EC 0x000001ED
0x000001EE 0x000001EF 0x000001F0 0x000001F1
from TOP_MASTER2.traffic_generator_func - T:0.001421446
DEBUG TOP_MEMORY2: Transaction details at T:0.001421446:
From Initiator thread: *TOP_MASTER2.traffic_generator_func
Address: 0x00000788 - Number: 16 (nb byte: 64) - Access: WRITE
Data: 0x000001E2 0x000001E3 0x000001E4 0x000001E5 0x000001E6 0x000001E7 0x000001E8 0x000001E9 0x000001EA 0x000001EB 0x000001EC
0x000001ED 0x000001EE 0x000001EF 0x000001F0 0x000001F1
Access Mode: DEFAULT - Transaction status: SUCCESS
DEBUG TOP_MEMORY1: Received write request T:0.001422558
DEBUG TOP_MEMORY1: Write 0x000001D9 at 0x00000764 from TOP_MASTER1.traffic_generator_func - T:0.001422558
DEBUG From Initiator thread: *TOP_MASTER1.traffic_generator_func
Address: 0x00000764 - Number: 1 (nb byte: 4) - Access: WRITE
Data: 0x000001D9
Access Mode: DEFAULT - Transaction status: SUCCESS
DEBUG TOP_MEMORY2: Received read block request T:0.001422607
DEBUG TOP_MEMORY2: Read block(16) at 0x00000788
Data:
0x000001E2 0x000001E3 0x000001E4 0x000001E5
0x000001E6 0x000001E7 0x000001E8 0x000001E9
0x000001EA 0x000001EB 0x000001EC 0x000001ED
0x000001EE 0x000001EF 0x000001F0 0x000001F1
from TOP_MASTER2.traffic_generator_func - T:0.001422607

```

Fig. 21(a) TLM Run-1

```

TLM_TEST_PLT
Verilog temp SLM_NEW_BUILD SLM GUI NEW Terminal SLM 6.0 lab Terminal
Access Mode: DEFAULT - Transaction status: SUCCESS
DEBUG TOP_MEMORY2: Received read block request T:0.074323952
DEBUG TOP_MEMORY2: Read block(16) at 0x00000EEC
Data:
0x000003B8 0x000003BC 0x000003BD 0x000003BE
0x000003BF 0x000003C0 0x000003C1 0x000003C2
0x000003C3 0x000003C4 0x000003C5 0x000003C6
0x000003C7 0x000003C8 0x000003C9 0x000003CA
from TOP_MASTER2.traffic_generator_func - T:0.074323949
DEBUG TOP_MEMORY2: Transaction details at T:0.074323952:
From Initiator thread: *TOP_MASTER2.traffic_generator_func
Address: 0x00000EEC - Number: 16 (nb byte: 64) - Access: READ
Data: 0x000003B8 0x000003BC 0x000003BD 0x000003BE 0x000003BF 0x000003C0 0x000003C1 0x000003C2 0x000003C3 0x000003C4 0x000003C5
0x000003C6 0x000003C7 0x000003C8 0x000003C9 0x000003CA
Access Mode: DEFAULT - Transaction status: SUCCESS
WRITE TOP_MASTER1: End of benchmark T:0.074324921
TOP_TIMER: Timer is Disabled T:0.074325047
TOP_MASTER2: End of benchmark T:0.074331619
---- End of simulation ----
TOP: Destructor called
TOP_MASTER2: Destructor called
TOP_MASTER1: Destructor called
TOP_TIMER: Destructor called
DEBUG TOP_MEMORY2: Destructor called, memory released
DEBUG TOP_MEMORY1: Destructor called, memory released
Simulation Time: 0.074331619 s
Real simulation time: 4.2300 s
Transactions: 260003
Transactions/Sec: 47282.03
Kbytes: 6675.51
Kbytes/Sec: 1570.14
Byte/Transaction(Av.): 34.18
Total number of Errors: 1 (1 displayed)
Total number of Warnings: 6 (6 displayed)
Random Time Generator Seed Value: 0
dlh12117(goets2)002:

```

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.

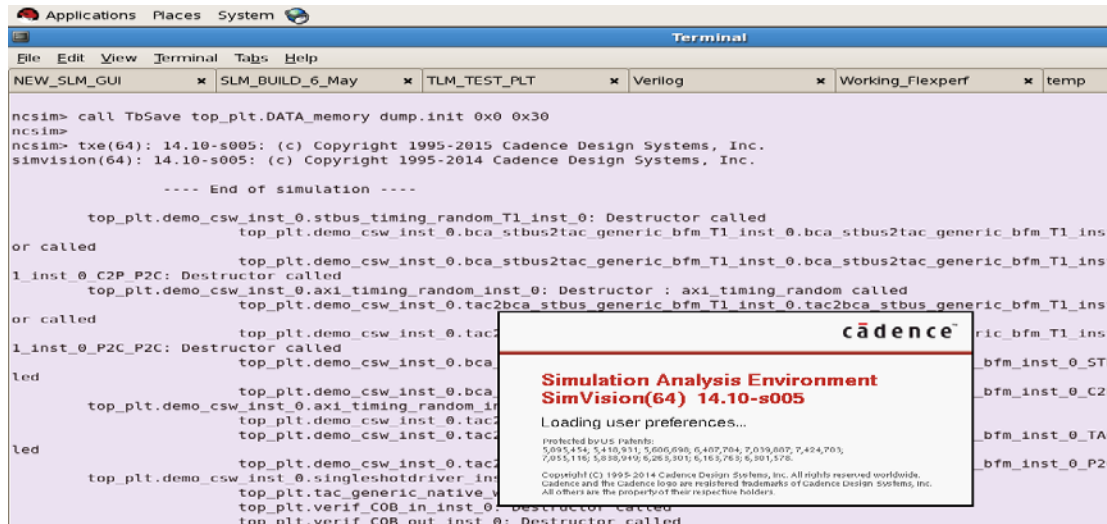


Fig. 7.22 Ncsim GUI Initialization

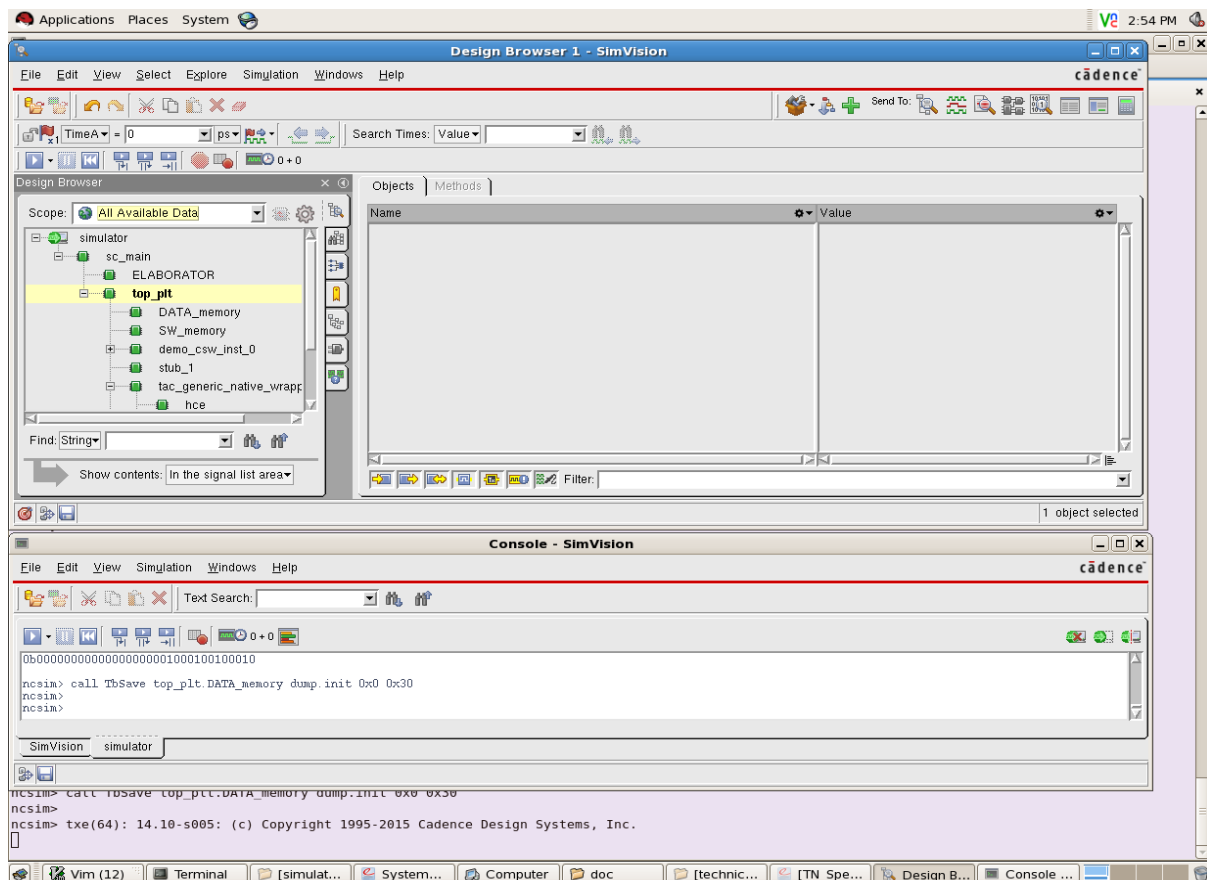


Fig. 7.23 Ncsim Design Browser

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

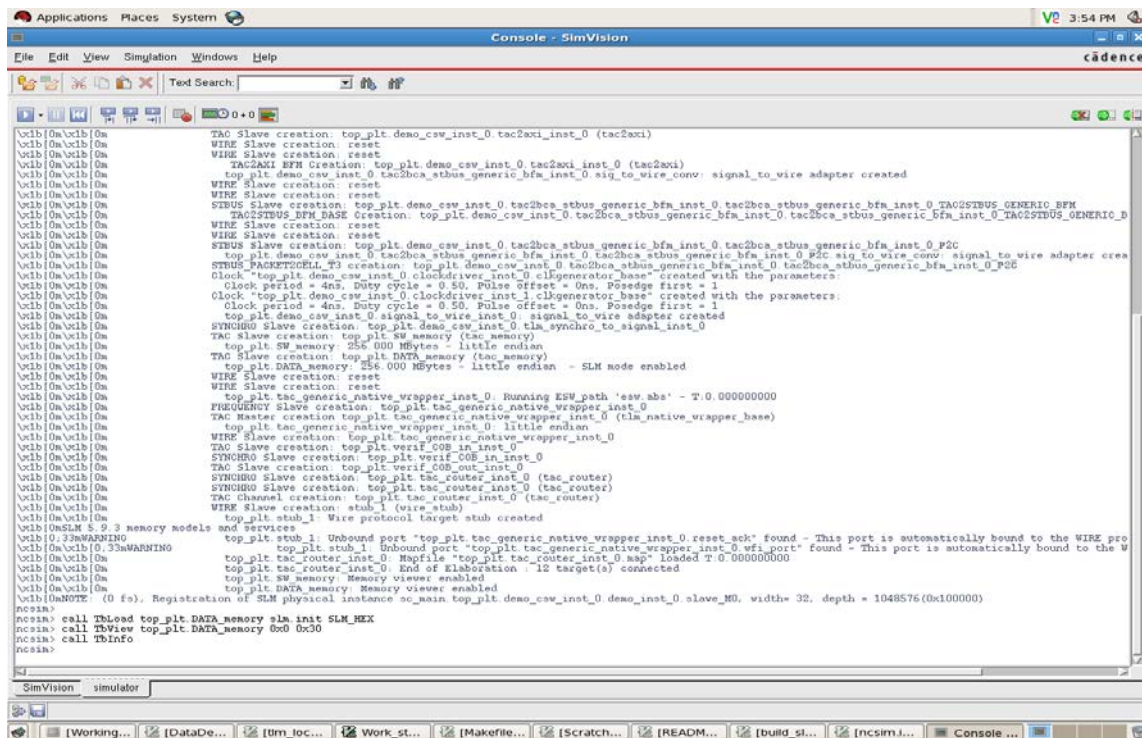


Fig. 7.24 SimVision Console

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.

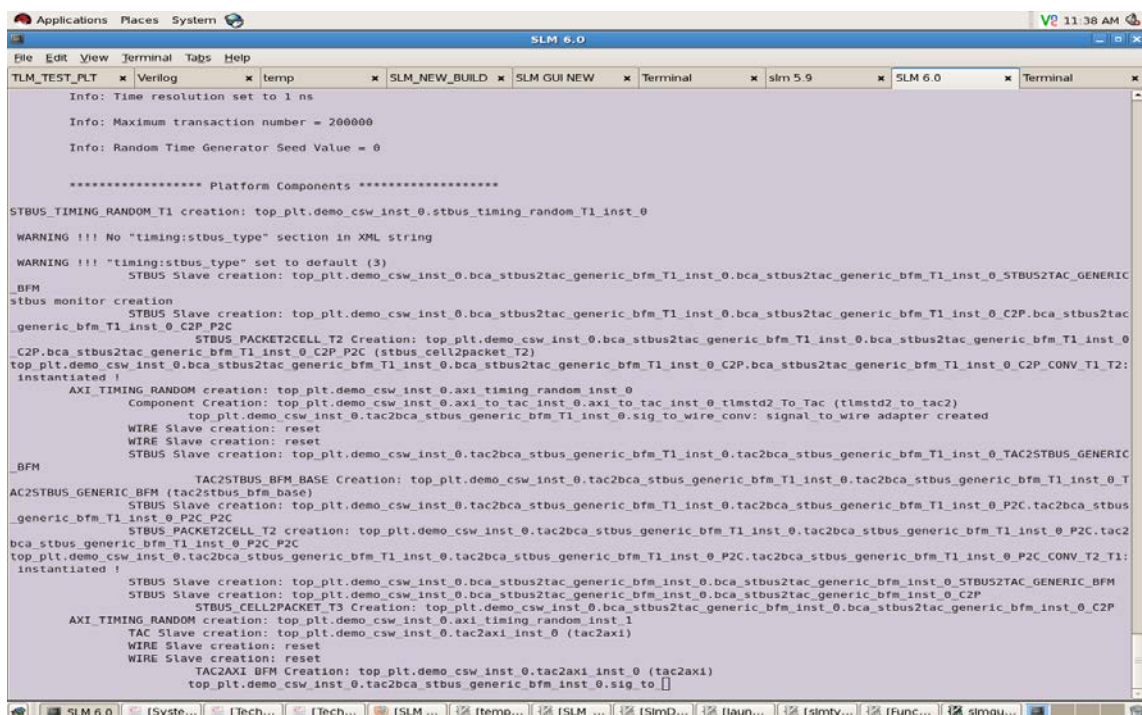


Fig. 7.25(a) Ncsim Simulation using SLM-View 1



### 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.

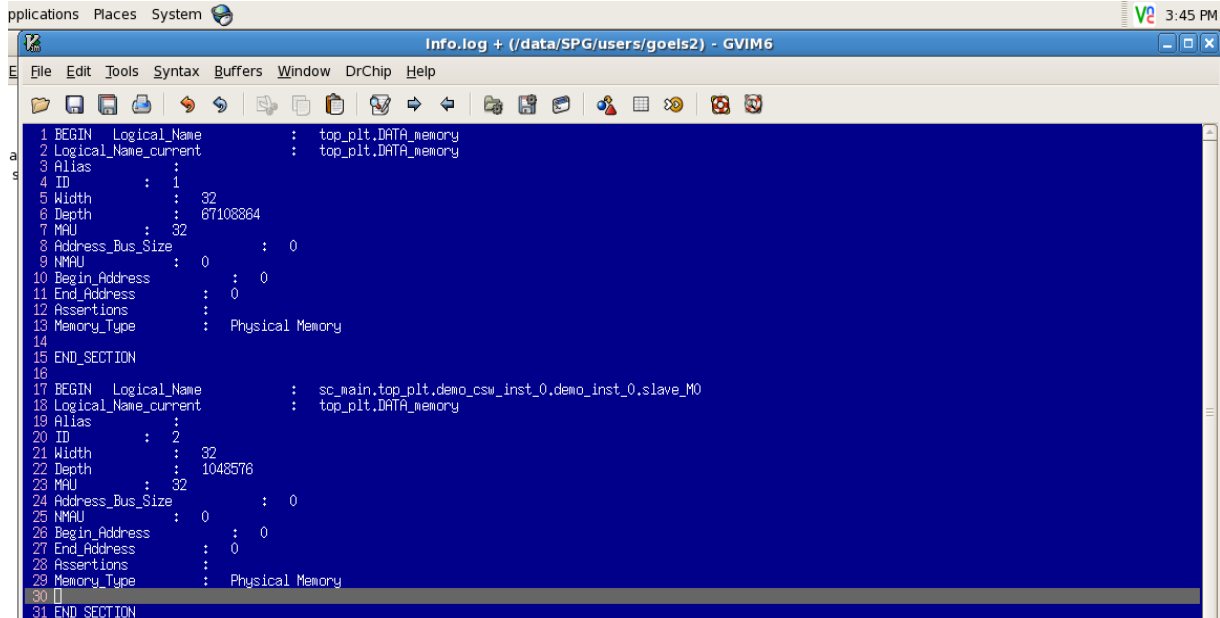


Fig. 7.26 Memory Attributes

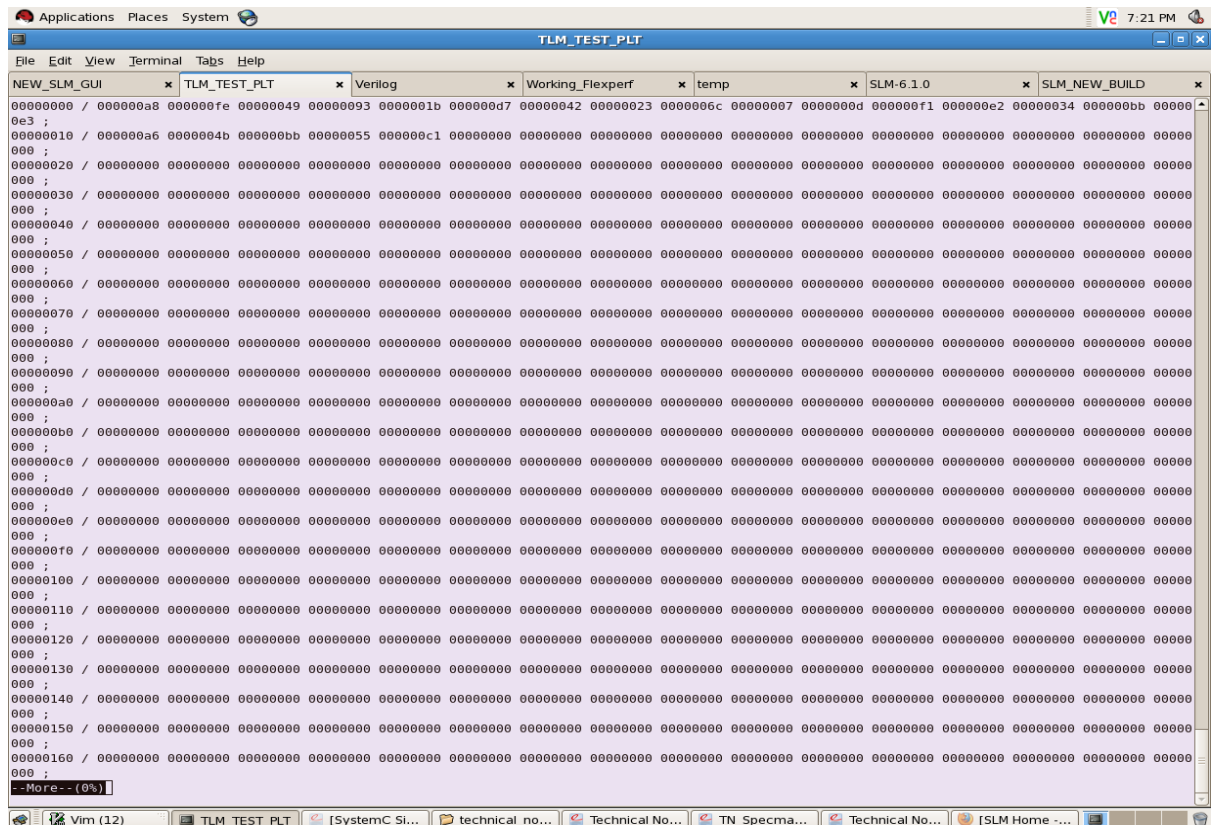


Fig. 7.27 Memory Buffer

### 7.1.5 GUI Testing Process

A brief view of <control, event> registrations, 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 Registrations**

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

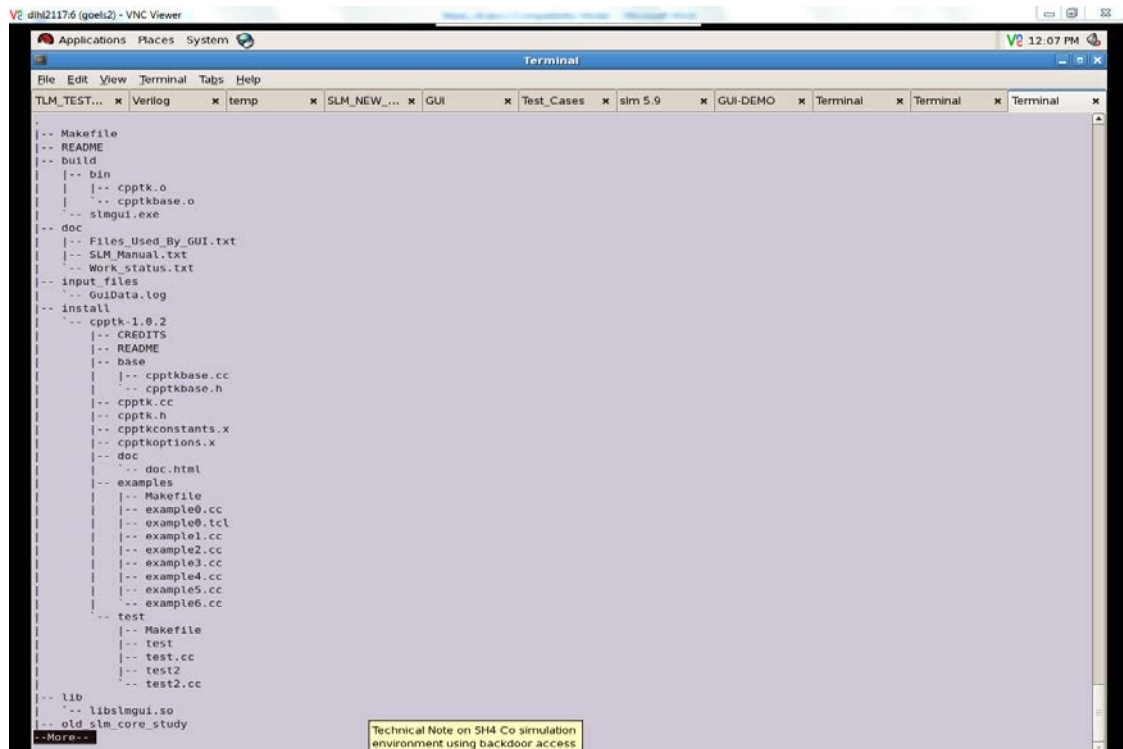
**Fig. 28(c) Binding Function Call**

```
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**

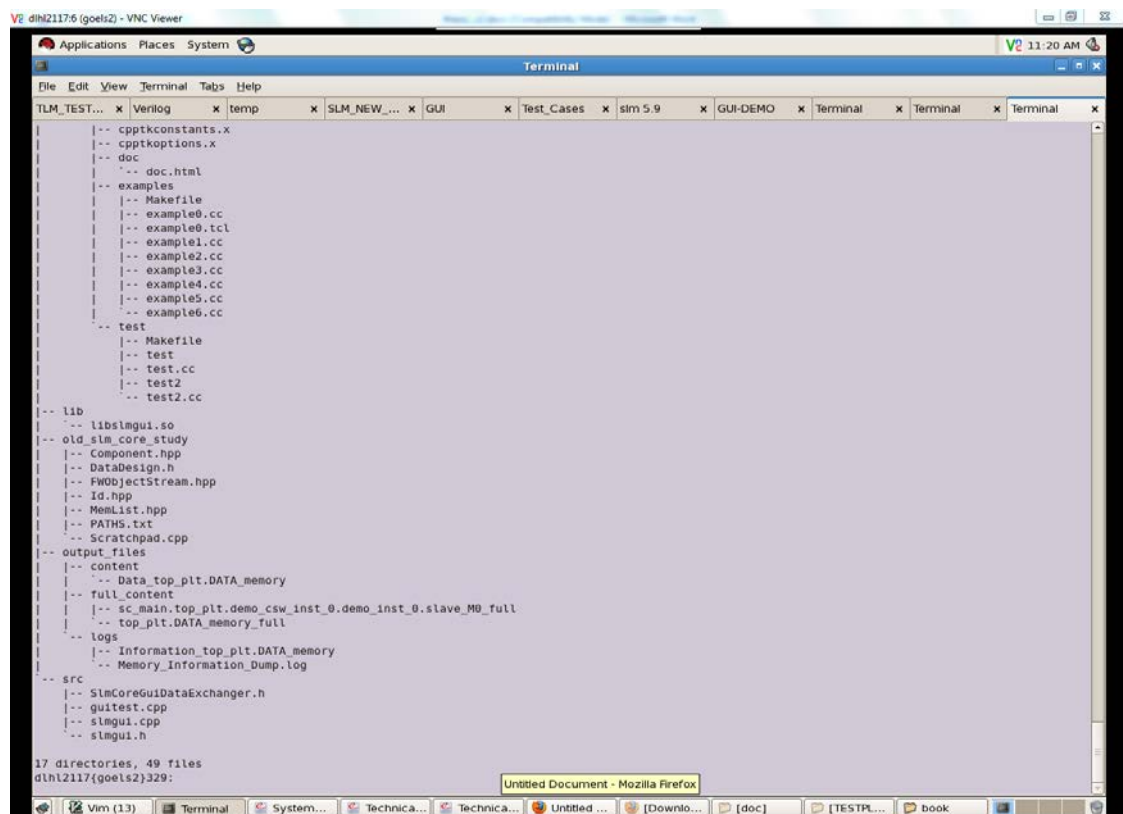
## 7.1.6 Directory Structure

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



```
dlh2117:6 (goels2) - VNC Viewer
Applications Places System
Terminal
TLM_TEST... x Verilog x temp x SLM_NEW... x GUI x Test_Cases x slm 5.9 x GUI-DEMO x Terminal x Terminal x Terminal x
-- Makefile
-- README
-- build
| -- bin
| | -- cpptk.o
| | -- cpptkbase.o
| -- stegui.exe
-- doc
| -- Files_Used_By_GUI.txt
| -- SLM_Manual.txt
| -- Work_status.txt
-- input_files
| -- GuiData.log
-- install
| -- cpptk-1.0.2
| -- CREDITS
| -- README
| -- base
| | -- cpptkbase.cc
| | -- cpptkbase.h
| -- cpptk.cc
| -- cpptk.h
| -- cpptkconstants.x
| -- cpptkoptions.x
| -- doc
| | -- doc.html
| -- examples
| | -- Makefile
| | | -- example0.cc
| | | -- example0.tcl
| | | -- example1.cc
| | | -- example2.cc
| | | -- example3.cc
| | | -- example4.cc
| | | -- example5.cc
| | | -- example6.cc
| -- test
| -- Makefile
| -- test
| -- test.cc
| -- test2
| -- test2.cc
-- lib
| -- libstmgui.so
-- old_slm_core_study
--More--
Technical Note on SH4 Co simulation
environment using backdoor access
```

Fig. 7.29(a) Directory structure of GUI



```
dlh2117:6 (goels2) - VNC Viewer
Applications Places System
Terminal
TLM_TEST... x Verilog x temp x SLM_NEW... x GUI x Test_Cases x slm 5.9 x GUI-DEMO x Terminal x Terminal x Terminal x
| -- cpptkconstants.x
| -- cpptkoptions.x
| -- doc
| | -- doc.html
| -- examples
| | -- Makefile
| | | -- example0.cc
| | | -- example0.tcl
| | | -- example1.cc
| | | -- example2.cc
| | | -- example3.cc
| | | -- example4.cc
| | | -- example5.cc
| | | -- example6.cc
| -- test
| -- Makefile
| -- test
| -- test.cc
| -- test2
| -- test2.cc
-- lib
| -- libstmgui.so
-- old_slm_core_study
| -- Component.hpp
| -- DataDesign.h
| -- FwObjectStream.hpp
| -- Id.hpp
| -- MemList.hpp
| -- PATHS.txt
| -- Scratchpad.cpp
-- output_files
| -- content
| | -- Data_top_plt.DATA_memory
| -- full_content
| | -- sc.main.top_plt.demo_csw_inst_0.demo_inst_0.slave_M0_full
| | -- top_plt.DATA_memory_full
| -- logs
| -- Information_top_plt.DATA_memory
| -- Memory_Information_Dump.log
-- src
| -- SlmCoreGuiDataExchange.h
| -- guitest.cpp
| -- stmgui.cpp
| -- stmgui.h
17 directories, 49 files
dlh2117(goels2)329:
Untitled Document - Mozilla Firefox
```

Fig. 7.29(b) Directory structure of GUI

## 7.2 SLM Test Suite

### 7.2.1 Environment Setup

It requires 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).

```
1  #!/bin/tcsh
2
3      # setenv SLM_HOME to <LOCAL_PATH>/distrib/dclm/uuuu/
4      # setenv GNUVERSION <VERSION>
5      # setenv SLM_BITS 64
6      # source $SLM_HOME/SLM.csh
7
```

Fig. 7.30(a) Required Environment Variables

```
17 source $SLM_HOME/SLM.csh
18
19
20 source /data/SPG/users/krishnak/projects/3_SLM/SLM_TRUNK/slm/examples/setup.csh
21
22
23 echo "-----"
24 echo $SLM_HOME
25 echo $GNUVERSION
26 echo $SLM_BITS
27 echo "-----"
```

Fig. 7.30(b) Required scripts for configuration setup

```
dhl2117@goels2:~$ source setup.csh
*****
Checking for slm environment
*****
SLM::WARNING: NCSIM not used.

Done

*****
Setting up simulator
*****
Select an option for simulator >
[1] NCSIM
[2] MODELSIM
[3] VCS
```

Fig. 7.30(c) SLM Setup

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

```
dhl2117@goels2:~$ make tests
./sw/st_division/spg/ext_tools/gcc/4.1.1/gold/x86_64-rhel5/bin/g++ -Dlinux -Wall -pthread -fPIK -DFP2_DONT_USE_FLEXSAM -DCUSTOM_FLAG -DHUMAN -DBY_SILVI
-DNEM_GUI_ENABLE -DTEST_MODE -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/slm_core -I/data/SPG/
users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/isskit -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/S
LM_Lib/slm_package_20150505/slm_trunk/src/validation/TestPlan/c++/common -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_2015
0505/slm_trunk/src/cpp/targets -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/targets/C_Function_
Tables -I/sw/cadence/incisiv/14.10.008/lnx86//tools.lnx86/transrecord/include -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-
pc-linux-gnu-gcc-4.1.1/include/bfr -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1/include
e/fwtlib -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1/include/twlib/platforms/Linux -I/data/SL
M/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1/include/FP2 -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/inst
all_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1/include/flexsam -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64
-pc-linux-gnu-gcc-4.1.1/include/flexsam/interface -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/systemc-2.3.1/src/sysc/packages/boost -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM_03 -DSDI -D64BIT -c
/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/validation/TestPlan/c++/access/AddrScram/Source/AddrSc
ram.cpp -o /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/./temp_build/release/Linux/4.1.1/64/obj/Addr
Scram.sin.o
```

Fig. 7.31(a) Build Tests

```

Applications Places System
SLM_NEW_BUILD
File Edit View Terminal Tabs Help
TLM_TEST... x Verilog x temp x SLM_NEW... x GUI x Test_Cases x slm 5.9 x GUI-DEMO x Terminal x Terminal x Terminal x
int LoadAllMem(int, char*, int*, FileFormatT)*:
/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/validation/TestPlan/c++/common/Common.hpp:723: warning:
'status' may be used uninitialized in this function
/sw/st_division/spg/ext_tools/gcc/4.1.1.gold/x86_64-rhel5/bin/g++ -Dlinux -Wall -pthread -fPIC -DFP2_DONT_USE_FLEXSAM -DCUSTOM_FLAG -DHUMAN -DBY_SILVI
-DNEW_GUI_ENABLE -DTEST_MODE -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/slm_core -I/data/SPG/
users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/isskit -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/S
LM_Lib/slm_package_20150505/slm_trunk/src/validation/TestPlan/c++/common -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_2015
0505/slm_trunk/src/cpp/targets -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/targets/C_Function
Tables -I/sw/cadence/incisiv/14.10.008/lnx86//tools.lnx86/transrecord/include -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x
86_64-pc-linux-gnu-gcc-4.1.1//include/bfr -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//includ
e/fwlib -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/fwlib/platforms/Linux -I/data/SL
M/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/FP2 -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/inst
all_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/flexsam -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64
-pc-linux-gnu-gcc-4.1.1//include/flexsam/interface -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/systemc-2.3.1/src/sysc/packages/boost -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM -O3 -DSDI -D_64BIT -c
/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/slm_core/DbiRecorder.cpp -o /data/SPG/users/goels2
/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/.temp_build/release/linux/4.1.1/64/obj/DbiRecorder.slm.o
/sw/st_division/spg/ext_tools/gcc/4.1.1.gold/x86_64-rhel5/bin/g++ -Dlinux -Wall -pthread -fPIC -DFP2_DONT_USE_FLEXSAM -DCUSTOM_FLAG -DHUMAN -DBY_SILVI
-DNEW_GUI_ENABLE -DTEST_MODE -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/slm_core -I/data/SPG/
users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/isskit -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/S
LM_Lib/slm_package_20150505/slm_trunk/src/validation/TestPlan/c++/common -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_2015
0505/slm_trunk/src/cpp/targets -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/targets/C_Function
Tables -I/sw/cadence/incisiv/14.10.008/lnx86//tools.lnx86/transrecord/include -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x
86_64-pc-linux-gnu-gcc-4.1.1//include/bfr -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//includ
e/fwlib -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/fwlib/platforms/Linux -I/data/SL
M/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/FP2 -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/inst
all_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/flexsam -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64
-pc-linux-gnu-gcc-4.1.1//include/flexsam/interface -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/systemc-2.3.1/src/sysc/packages/boost -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM -O3 -DSDI -D_64BIT -c
/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/slm_core/EnhancedPhysicalMemory.cpp -o /data/SPG/usa
ers/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/.temp_build/release/linux/4.1.1/64/obj/EnhancedPhysicalMemory.slm.o
/sw/st_division/spg/ext_tools/gcc/4.1.1.gold/x86_64-rhel5/bin/g++ -Dlinux -Wall -pthread -fPIC -DFP2_DONT_USE_FLEXSAM -DCUSTOM_FLAG -DHUMAN -DBY_SILVI
-DNEW_GUI_ENABLE -DTEST_MODE -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/slm_core -I/data/SPG/
users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/isskit -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/S
LM_Lib/slm_package_20150505/slm_trunk/src/validation/TestPlan/c++/common -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_2015
0505/slm_trunk/src/cpp/targets -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/targets/C_Function
Tables -I/sw/cadence/incisiv/14.10.008/lnx86//tools.lnx86/transrecord/include -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x
86_64-pc-linux-gnu-gcc-4.1.1//include/bfr -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//includ
e/fwlib -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/fwlib/platforms/Linux -I/data/SL
M/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/FP2 -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/inst
all_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//include/flexsam -I/data/SLM/SLM_EXTERNAL_DEPENDENCIES/fp2/install_64/SDK_FlexPerf_2.3.0/x86_64
-pc-linux-gnu-gcc-4.1.1//include/flexsam/interface -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/systemc-2.3.1/src/sysc/packages/boost -I/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM -O3 -DSDI -D_64BIT -c
/data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/cpp/slm_core/EnhancedSystemMemory.cpp -o /data/SPG/usa
ers/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm_package_20150505/slm_trunk/src/.temp_build/release/linux/4.1.1/64/obj/EnhancedSystemMemory.slm.o

```

Fig. 7.31(b) Build Tests

## 7.2.2 Running Status of Test Cases

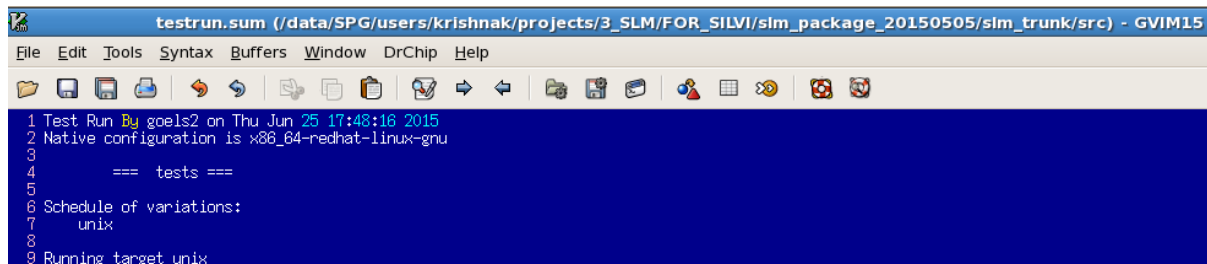
Fig. 7.32 shows the current running status of test cases. Many of them are working, but some are not currently working.

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

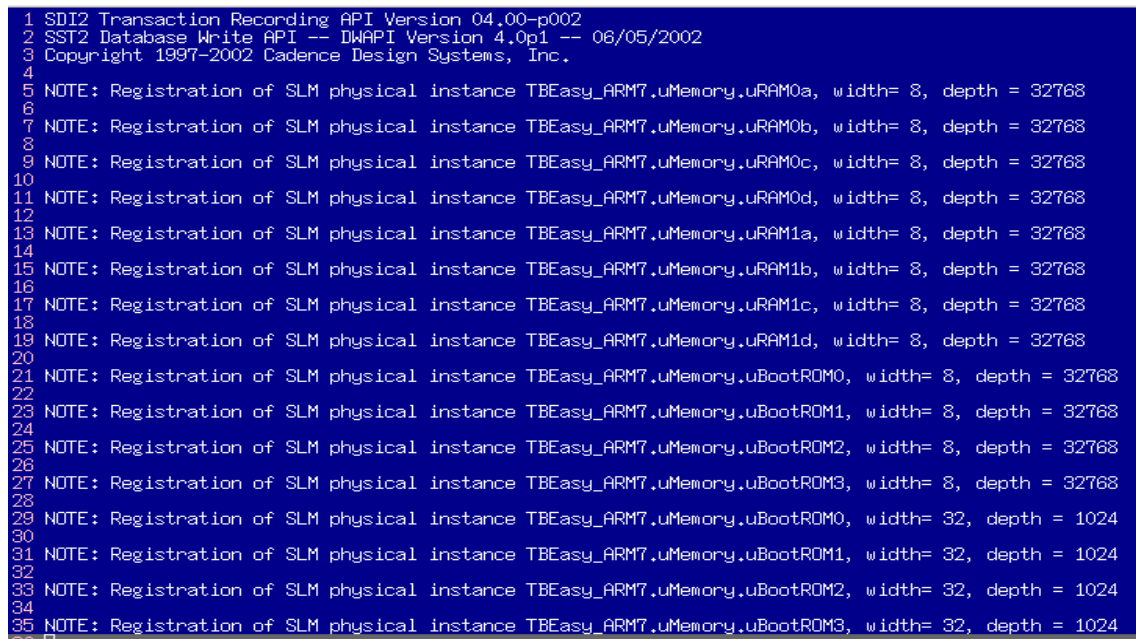
Following is a set of logs recorded during test case execution. Testrun script is written to conduct automated execution of tests. Testrun.sum marks the beginning of test-automation script.



```
testrun.sum (/data/SPG/users/krishnak/projects/3_SLM/FOR_SILVI/slm_package_20150505/slm_trunk/src) - GVIM15
File Edit Tools Syntax Buffers Window DrChip Help
1 Test Run By goels2 on Thu Jun 25 17:48:16 2015
2 Native configuration is x86_64-redhat-linux-gnu
3
4 === tests ===
5
6 Schedule of variations:
7 unix
8
9 Running target unix
```

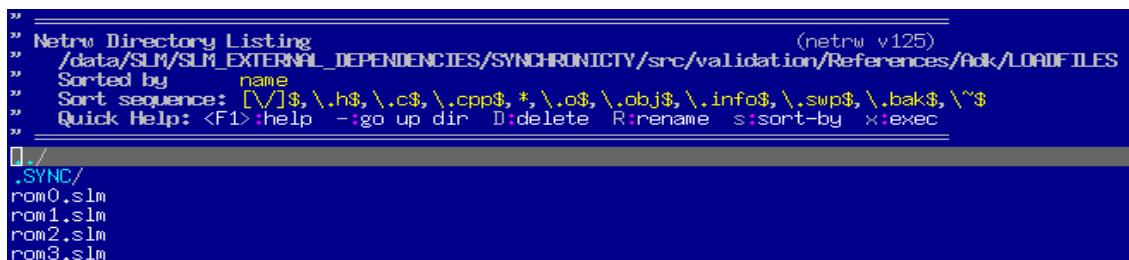
**Fig. 7.33 Test Run Start**

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



```
1 SDI2 Transaction Recording API Version 04.00-p002
2 SST2 Database Write API -- DWAPI Version 4.0p1 -- 06/05/2002
3 Copyright 1997-2002 Cadence Design Systems, Inc.
4
5 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uRAM0a, width= 8, depth = 32768
6
7 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uRAM0b, width= 8, depth = 32768
8
9 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uRAM0c, width= 8, depth = 32768
10
11 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uRAM0d, width= 8, depth = 32768
12
13 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uRAM1a, width= 8, depth = 32768
14
15 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uRAM1b, width= 8, depth = 32768
16
17 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uRAM1c, width= 8, depth = 32768
18
19 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uRAM1d, width= 8, depth = 32768
20
21 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uBootROM0, width= 8, depth = 32768
22
23 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uBootROM1, width= 8, depth = 32768
24
25 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uBootROM2, width= 8, depth = 32768
26
27 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uBootROM3, width= 8, depth = 32768
28
29 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uBootROM0, width= 32, depth = 1024
30
31 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uBootROM1, width= 32, depth = 1024
32
33 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uBootROM2, width= 32, depth = 1024
34
35 NOTE: Registration of SLM physical instance TBEasy_ARM7.uMemory.uBootROM3, width= 32, depth = 1024
```

**Fig. 7.34(a) Adk Run**



```
Netrw Directory Listing (netrw v125)
/data/SLM/SLM_EXTERNAL_DEPENDENCIES/SYNCHRONICTY/src/validation/References/Adk/LOADFILES
Sorted by name
Sort sequence: [\V]$, \.h$, \.c$, \.cpp$, *, \.o$, \.obj$, \.info$, \.swp$, \.bak$, \~$
Quick Help: <F1>:help -:go up dir D:delete R:rename s:sort-by x:exec

./
./SYNC/
rom0.slm
rom1.slm
rom2.slm
rom3.slm
```

**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.

```

3 NOTE: Registration of SLM physical instance 30-bit_phy_I14, width= 30, depth = 32
4
5 NOTE: Registration of SLM physical instance 30-bit_phy_I15, width= 30, depth = 32
6
7 NOTE: Registration of SLM physical instance 30-bit_phy_I16, width= 30, depth = 32
8
9 SUI2 Transaction Recording API Version 04.00-p002
10 SST2 Database Write API -- DWAPI Version 4.0p1 -- 06/05/2002
11 Copyright 1997-2002 Cadence Design Systems, Inc.
12
13 DEBUG: 30-bit_phy_I13 Console Recording Enabled>
14
15 DEBUG: 30-bit_phy_I14 Console Recording Enabled>
16
17 DEBUG: 30-bit_phy_I15 Console Recording Enabled>
18
19 DEBUG: 30-bit_phy_I16 Console Recording Enabled>
20
21 DEBUG: 60-bit_sys_S3 Console Recording Enabled>
22
23 ++++++
24 Checking Assertions for aAssertChoice = 0 and aOpChoice = 0
25 ++++++
26 =====
27 Checking for RW Operation Type = 0
28 =====
29 DEBUG: 30-bit_phy_I16 Reset> Data= 0x0000000000
30
31 DEBUG: 30-bit_phy_I16 Reset> Data= 0x0000000000
32
33 DEBUG: 30-bit_phy_I13 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
34
35 DEBUG: 30-bit_phy_I14 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
36
37 DEBUG: 30-bit_phy_I15 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
38
39 DEBUG: 30-bit_phy_I16 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
40
41 DEBUG: 60-bit_sys_S3 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
42
43 DEBUG: 30-bit_phy_I16 Change Assertion Action> Action Type= DEBUGINFO
44
45 DEBUG: 30-bit_phy_I16 Change Assertion Action> Action Type= NOTE
46
47 -----
48 Action Number for First = 0
49 Action Number for Second = 1
50 -----
51 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ
52
53 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= WRITE
54
55 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ+READ

```

**Fig. 7.35(a) Assertions Run**

```

23 ++++++
24 Checking Assertions for aAssertChoice = 0 and aOpChoice = 0
25 ++++++
26 =====
27 Checking for RW Operation Type = 0
28 =====
29 DEBUG: 30-bit_phy_I16 Reset> Data= 0x0000000000
30
31 DEBUG: 30-bit_phy_I16 Reset> Data= 0x0000000000
32
33 DEBUG: 30-bit_phy_I13 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
34
35 DEBUG: 30-bit_phy_I14 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
36
37 DEBUG: 30-bit_phy_I15 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
38
39 DEBUG: 30-bit_phy_I16 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
40
41 DEBUG: 60-bit_sys_S3 Delete Event Assertion> Range= Entire Memory, Assertion Type= All Assertions
42
43 DEBUG: 30-bit_phy_I16 Change Assertion Action> Action Type= DEBUGINFO
44
45 DEBUG: 30-bit_phy_I16 Change Assertion Action> Action Type= NOTE
46
47 -----
48 Action Number for First = 0
49 Action Number for Second = 1
50 -----
51 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ
52
53 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= WRITE
54
55 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ+READ
56
57 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= WRITE+WRITE
58
59 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ&WRITE
60
61 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ
62
63 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= WRITE
64
65 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ+READ
66
67 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= WRITE+WRITE
68
69 DEBUG: 30-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ&WRITE
70
71 DEBUG: 30-bit_phy_I13 Console Recording Disabled>
72
73 DEBUG: 30-bit_phy_I14 Console Recording Disabled>
74
75 DEBUG: 30-bit_phy_I15 Console Recording Disabled>

```

**Fig. 7.35(b) Assertions Run**

```

64
65 DEBUG: 36-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ+READ
66
67 DEBUG: 36-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= WRITE+WRITE
68
69 DEBUG: 36-bit_phy_I16 Add Event Assertion> @= 0xa, Assertion Type= READ&!WRITE
70
71 DEBUG: 30-bit_phy_I13 Console Recording Disabled>
72
73 DEBUG: 30-bit_phy_I14 Console Recording Disabled>
74
75 DEBUG: 36-bit_phy_I15 Console Recording Disabled>
76
77 DEBUG: 36-bit_phy_I16 Console Recording Disabled>
78
79 DEBUG: 66-bit_sys_S3 Console Recording Disabled>
80
81 NOTE: READ ASSERTION hits on 36-bit_phy_I16 (Id = 4) @ = 0xa
82
83 NOTE: READ&!WRITE ASSERTION hits on 36-bit_phy_I16 (Id = 4) @ = 0xa
84
85 NOTE: WRITE ASSERTION hits on 36-bit_phy_I16 (Id = 4) @ = 0xa
86
87 NOTE: WRITE ASSERTION hits on 36-bit_phy_I16 (Id = 4) @ = 0xa
88
89 NOTE: WRITE+WRITE ASSERTION hits on 36-bit_phy_I16 (Id = 4) @ = 0xa
90
91 NOTE: READ ASSERTION hits on 36-bit_phy_I16 (Id = 4) @ = 0xa
92
93 NOTE: READ ASSERTION hits on 36-bit_phy_I16 (Id = 4) @ = 0xa
94
95 NOTE: READ+READ ASSERTION hits on 36-bit_phy_I16 (Id = 4) @ = 0xa
96
97 DEBUG: 30-bit_phy_I13 Console Recording Enabled>
98
99 DEBUG: 30-bit_phy_I14 Console Recording Enabled>
100
101 DEBUG: 36-bit_phy_I15 Console Recording Enabled>
102
103 DEBUG: 36-bit_phy_I16 Console Recording Enabled>
104
105 DEBUG: 66-bit_sys_S3 Console Recording Enabled>
106 []
107 DEBUG: 36-bit_phy_I16 Save> File= dummy.sav, Range= 0x1 to 0x1f
108
109 DEBUG: 36-bit_phy_I16 Save> File= dummy.sav, Range= 0x1 to 0x1f
110
111 DEBUG: 36-bit_phy_I16 Save> File= dummy.sav, Range= Entire Memory
112
113 DEBUG: 36-bit_phy_I16 Load> File= dummy.sav
114
115 DEBUG: 36-bit_phy_I16 Compare> File= dummy.sav
116
117 DEBUG: 36-bit_phy_I16 Add Event Assertion> Range= 0x0 to 0x1f, Assertion Type= READ|WRITE

```

**Fig. 7.35(c) Assertions Run**

```

32087
32088 DEBUG: 66-bit_sys_S3 Console Recording Disabled>
32089
32090 DEBUG: 30-bit_phy_I13 Console Recording Enabled>
32091
32092 DEBUG: 30-bit_phy_I14 Console Recording Enabled>
32093
32094 DEBUG: 36-bit_phy_I15 Console Recording Enabled>
32095
32096 DEBUG: 36-bit_phy_I16 Console Recording Enabled>
32097
32098 DEBUG: 66-bit_sys_S3 Console Recording Enabled>
32099
32100 NOTE: 36-bit_phy_I16: Assertions set with severity level INFO.
32101
32102 NOTE: 36-bit_phy_I16: Assertion of type READ set on following cell(s):
32103 Range: 0x1d - 0x1f
32104
32105 NOTE: 66-bit_sys_S3: Assertions set with severity level DEBUGINFO.
32106
32107 NOTE: 66-bit_sys_S3: Assertion of type READ set on following cell(s):
32108 Range: 0x1d - 0x1d
32109 Range: 0x1f - 0x1f
32110 Range: 0x3d - 0x3f
32111
32112 NOTE: 30-bit_phy_I13: Assertions set with severity level WARNING.
32113
32114 NOTE: 30-bit_phy_I13: Assertion of type READ set on following cell(s):
32115 Range: 0x1d - 0x1d
32116 Range: 0x1f - 0x1f
32117
32118 NOTE: 30-bit_phy_I14: No Assertion set.
32119
32120 NOTE: 36-bit_phy_I15: Assertions set with severity level WARNING.
32121
32122 NOTE: 36-bit_phy_I15: Assertion of type READ set on following cell(s):
32123 Range: 0x1d - 0x1d
32124 Range: 0x1f - 0x1f
32125
32126 NOTE: 36-bit_phy_I16: Assertions set with severity level INFO.
32127
32128 NOTE: 36-bit_phy_I16: Assertion of type READ set on following cell(s):
32129 Range: 0x1d - 0x1f
32130
32131 NOTE: 66-bit_sys_S3: Assertions set with severity level DEBUGINFO.
32132
32133 NOTE: 66-bit_sys_S3: Assertion of type READ set on following cell(s):
32134 Range: 0x1d - 0x1d
32135 Range: 0x1f - 0x1f
32136 Range: 0x3d - 0x3f
32137
32138 TEST COMPLETED []

```

**Fig. 7.35(d) Assertions Run**

```

38072 DEBUG: 36-bit_phy_I15 Console Recording Enabled>
38073
38074 DEBUG: 36-bit_phy_I16 Console Recording Enabled>
38075
38076 DEBUG: 66-bit_sys_S3 Console Recording Enabled>
38077
38078 DEBUG: 66-bit_sys_S3 Add Event Assertion> Range= 0x0 to 0x3f, Assertion Type= WRITE
38079
38080 DEBUG: 30-bit_phy_I13 Console Recording Disabled>
38081
38082 DEBUG: 30-bit_phy_I14 Console Recording Disabled>
38083
38084 DEBUG: 36-bit_phy_I15 Console Recording Disabled>
38085
38086 DEBUG: 36-bit_phy_I16 Console Recording Disabled>
38087
38088 DEBUG: 66-bit_sys_S3 Console Recording Disabled>
38089
38090 DEBUG: 30-bit_phy_I13 Console Recording Enabled>
38091
38092 DEBUG: 30-bit_phy_I14 Console Recording Enabled>
38093
38094 DEBUG: 36-bit_phy_I15 Console Recording Enabled>
38095
38096 DEBUG: 36-bit_phy_I16 Console Recording Enabled>
38097
38098 DEBUG: 66-bit_sys_S3 Console Recording Enabled>
38099
38100 NOTE: 36-bit_phy_I16: Assertions set with severity level INFO.
38101
38102 NOTE: 36-bit_phy_I16: Assertion of type READ set on following cell(s):
38103 Range: 0x1d - 0x1f
38104
38105 NOTE: 66-bit_sys_S3: Assertions set with severity level DEBUGINFO.
38106
38107 NOTE: 66-bit_sys_S3: Assertion of type READ set on following cell(s):
38108 Range: 0x1d - 0x1d
38109 Range: 0x1f - 0x1f
38110 Range: 0x3d - 0x3f
38111
38112 NOTE: 30-bit_phy_I13: Assertions set with severity level WARNING.
38113
38114 NOTE: 30-bit_phy_I13: Assertion of type READ set on following cell(s):
38115 Range: 0x1d - 0x1d
38116 Range: 0x1f - 0x1f
38117
38118 NOTE: 30-bit_phy_I14: No Assertion set.
38119
38120 NOTE: 36-bit_phy_I15: Assertions set with severity level WARNING.
38121
38122 NOTE: 36-bit_phy_I15: Assertion of type READ set on following cell(s):
38123 Range: 0x1d - 0x1d
38124 Range: 0x1f - 0x1f
38125
38126 NOTE: 36-bit_phy_I16: Assertions set with severity level INFO.

```

**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.

```

1 NOTE: Registration of SLM physical instance test.uMEM1, width= 1, depth = 16384
2
3 NOTE: Registration of SLM physical instance test.uMEM2, width= 8, depth = 16384
4
5 NOTE: Registration of SLM physical instance test.uMEM3, width= 8, depth = 16384
6
7 NOTE: Registration of SLM physical instance test.uIntMem, width= 32, depth = 1024
8
9 [ ]TEST COMPLETED

```

**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.

```

143
149 NOTE: Registration of SLM physical instance I40, width= 30, depth = 512
150
151 NOTE: Registration of SLM physical instance I41, width= 36, depth = 512
152
153 NOTE: Registration of SLM physical instance I42, width= 30, depth = 512
154
155 NOTE: Registration of SLM physical instance I43, width= 36, depth = 512
156
157 NOTE: Registration of SLM physical instance top.dut.SPLARGE, width= 15, depth = 256
158
159 NOTE: Registration of SLM physical instance top.dut.FIFO, width= 16, depth = 512
160
161 NOTE: Registration of SLM physical instance top.dut.DPHS, width= 24, depth = 256
162
163 NOTE: Registration of SLM physical instance top.dut.ROM, width= 1, depth = 1024
164
165 NOTE: Registration of SLM physical instance top.dut.ROMCLP, width= 8, depth = 65536
166
167 NOTE: Registration of SLM physical instance top.dut.DPR, width= 24, depth = 65536
168
169 NOTE: Registration of SLM physical instance top.dut.DRAM, width= 1100, depth = 100
170
171 TEST COMPLETED

```

**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.

```

14
15 NOTE: Registration of SLM physical instance top.dut.I8, width= 24, depth = 288
16
17 NOTE: Registration of SLM physical instance top.dut.I9, width= 32, depth = 256
18
19 NOTE: Registration of SLM physical instance top.dut.sdut.I0, width= 40, depth = 512
20
21 NOTE: Registration of SLM physical instance top.dut.sdut.I1, width= 47, depth = 255
22
23 NOTE: Registration of SLM physical instance top.dut.sdut.I2, width= 48, depth = 767
24
25 NOTE: Registration of SLM physical instance top.dut.sdut.I3, width= 52, depth = 288
26
27 NOTE: Registration of SLM physical instance top.dut.sdut.I4, width= 56, depth = 512
28
29 NOTE: Registration of SLM physical instance top.dut.sdut.I5, width= 64, depth = 64
30
31 NOTE: Registration of SLM physical instance top.dut.sdut.I6, width= 72, depth = 128
32
33 NOTE: Registration of SLM physical instance top.dut.sdut.I7, width= 78, depth = 16
34
35 NOTE: Registration of SLM physical instance top.dut.sdut.I8, width= 80, depth = 8
36
37 NOTE: Registration of SLM physical instance top.dut.sdut.I9, width= 88, depth = 4
38
39 NOTE: Registration of SLM physical instance top.I0, width= 96, depth = 57
40
41 NOTE: Registration of SLM physical instance top.I1, width= 104, depth = 1024
42
43 NOTE: Registration of SLM physical instance top.I2, width= 112, depth = 4096
44
45 NOTE: Registration of SLM physical instance top.I3, width= 119, depth = 8192
46
47 NOTE: Registration of SLM physical instance top.I4, width= 120, depth = 16384
48
49 NOTE: Registration of SLM physical instance top.I5, width= 126, depth = 8190
50
51 NOTE: Registration of SLM physical instance top.I6, width= 127, depth = 32770

```

**Fig. 7.38(a) VerifServ Test Run**

```

140
141 NOTE: Registration of SLM physical instance I37, width= 36, depth = 32
142 []
143 NOTE: Registration of SLM physical instance I38, width= 30, depth = 1024
144
145 NOTE: Registration of SLM physical instance I39, width= 36, depth = 1024
146
147 NOTE: Registration of SLM physical instance I40, width= 30, depth = 512
148
149 NOTE: Registration of SLM physical instance I41, width= 36, depth = 512
150
151 NOTE: Registration of SLM physical instance I42, width= 30, depth = 512
152
153 NOTE: Registration of SLM physical instance I43, width= 36, depth = 512
154
155 NOTE: Registration of SLM physical instance top.dut.SPLARGE, width= 15, depth = 256
156
157 NOTE: Registration of SLM physical instance top.dut.FIFO, width= 16, depth = 512
158
159 NOTE: Registration of SLM physical instance top.dut.DPHS, width= 24, depth = 256
160
161 NOTE: Registration of SLM physical instance top.dut.ROM, width= 1, depth = 1024
162
163 NOTE: Registration of SLM physical instance top.dut.ROMCLP, width= 8, depth = 65536
164
165 NOTE: Registration of SLM physical instance top.dut.DPR, width= 24, depth = 65536
166
167 NOTE: Registration of SLM physical instance top.dut.IDRAM, width= 1100, depth = 100
168
169 -----
170 -- Checking Reset Operation
171 -----
172

```

**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.

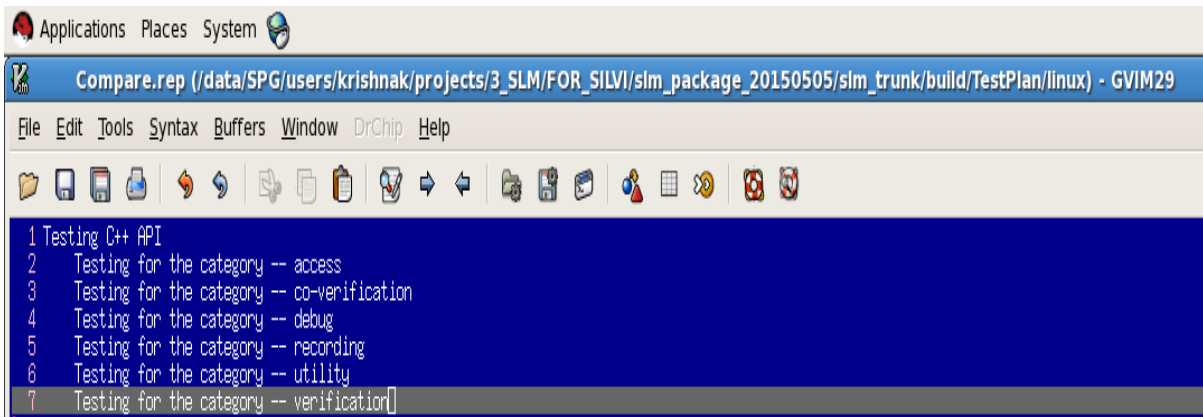
```
1 @00000000
2 0001000100010001 0000000000000000 1000100010001000 0010001000100010 0110011001100110 1000100010001000 0101010101010101 1000100010001000 0010001000100010
0000000000000000 0010001000100010 1000100010001000 0001000000100010 1100110011001100 1000100010001000 0110011001100110
3 @00000010
4 0110011001100110 1000100010001000 1100110011001100 0001000000100010 1001100001110110 1000100010001000 0010000100000011 0010001000100010 0000000000000000
0000000000000000 00000001000010001 0010001000100010 0101010000110010 1000100010001000 1110110111001011 0001000000100010
5 @00000020
6 0101010101010101 1111111011011100 0011001000010000 1000100010001000 01010011000010011 10111010100111000 0000000100010001 0110011001100110 1000011001000010
1111111011011100 0101010000110010 1100110011001100 0010001000100010 1010100001100100 0001000100001111 1001100001110110
7 @00000030
8 0000000100010001 1000100010001000 0101001000100000 1110110010101000 0110010000100000 1111111011011100 0111011001010100 0000000000000000 1000100010001000
0000000000000000 0110001100000001 0000111011001010 1001011101010011 0001000000010000 1100101110101001 0101010000110010
9 @00000040
10 1111111111111111 1000100010001000 0001000100010001 0000111010111000 1011100001010010 0101001000100000 0000111011001010 1010100001100100 0100001000000010
1111111011011100 1001100001110110 0011001000010000 1110111011101110 1000100010001000 0010001000100010 1110110011000010
11 @00000050
12 1001010100010001 0100000011111100 0000110110100111 1011100001010010 0110001100000001 0001000011101100 1101101110010111 1000011001000010 0011000100100001
1111111011011100 1010100110000111 0101010000110010 0000000100010001 1100110011001100 0111011101110111 0010001000100010
13 @00000060
14 1111101001010000 1011011000010000 0111001000001100 0011000011001000 0000110010000100 1100100001000000 1000010000000000 0100000011111100 0000111010111000
1101101001110100 1001011000110000 0101001000100000 0001000011101100 1110110010101000 1010100001100100 0110010000100000
15 @00000070
16 0010000000010000 1111111011011100 1011101010011000 0111011001010100 0011001000010000 0000000000000000 1100110011001100 1000100010001000 0100010001000100
0000000000000000 0110000110110101 0011111010000010 0000101101010011 111010000100000 1011010100101101 1000001011111010
17 @00000080
18 0101000111000111 0010111010010100 0000101101100001 1101100000110010 10100101000001111 0111001000001100 0100000111011001 0001111010100110 1111101101100110
1100100010000000 1001010100010001 0110001000011110 001100011101011 0000111010111000 1110101110000101 1011100001010010
19 @00000090
20 100001010001000011 0101001000010000 0010000111111101 0000111011001010 1101101110010111 1010100001100100 01110101000110001 0100001000000010 0001000100001111
1111111011011100 1100101110101001 1001100001110110 0110010101000011 0011001000010000 0000000100010001 1110111011101110
21 @000000a0
22 1011101110111011 1000100010001000 0101010101010101 0010001000100010 0000100000101010 1111011000001000 1101010011100110 1011001011000100 1001000010100010
0111000010000000 0101111001100010 0011110001000000 00011010000101110 0000100000001100 1110011000011010 1100010011111000
23 @000000b0
24 1010001011010110 1000000010110100 0110000010010010 0100111001110000 0010110001010010 0000101000110000 1111100000011110 1101011000101100 1011010000001010
1001001011101000 0111000011000110 0101000010100100 0011111010000010 0001110001100000 0000101001000010 1110100000100000
25 @000000c0
26 1100011000001110 1010010000011100 1000001011111010 0110000011011000 0100000010110110 0010111010010100 0000110001110010 1111101001010000 1101100000110010
1011011000010000 1001010000101110 0111001000001100 0101000011101010 0011000011001000 0001111010100110 0000110010000100
27 @000000d0
28 1110101001100010 1100100001000000 1010011000100010 1000010000000000 0110001000011110 0100000011111100 0010000011011010 0000111010111000 1111110010010110
1101101001110100 1011100001010010 1001011000110000 0111101000010010 0101001000100000 0011000000001110 0001000011101100
29 @000000e0
30 0000111011001010 1110110010101000 1100101010000110 1010100001100100 1000011001000010 0110010000100000 0100001000000010 001000000011111110
1111111011011100 1101110010111010 1011101010011000 1001100001110110 0111011001010100 0101010000110010 0011001000010000
31 @000000f0
32 0001000000100010 0000000000000000 1110111011101110 1100110011001100 1010101010101010 1000100010001000 0110011001100110 0100010001000100 0010001000100010
0000000000000000 0000001110001111 0000001001111110 1111000101101101 1110000001011100 1101000101001011 1100000000111010
33 @00000100
34 1011111100101001 1010111000011000 1001110100000111 1000110000100110 01111011000010101 0110101000000100 0101100111110011 0100100011100010 0011011111010001
0010011011000000 0001010110110011 0000010010100010 0000001110010001 1111001010000000 1110000101111111 1101000001101110
35 @00000110
36 1100000101011101 1011000001001100 1010111100111011 1001111000101010 1000110100011001 0111110000001000 0110110110010011 0101101000010110 0100100100000101
0011100011110100 0010011111100001 0001011011010010 0000010111000001 0000010010110000 1111001110100011 11100001010010010
37 @00000120
References/VlogFormat/LOADFILES_VLOG_BIN/512x16.cde" 64L, 9152C
```

Fig. 7.39(a) 512x16.cde input

```
1 0010 0010 0010
```

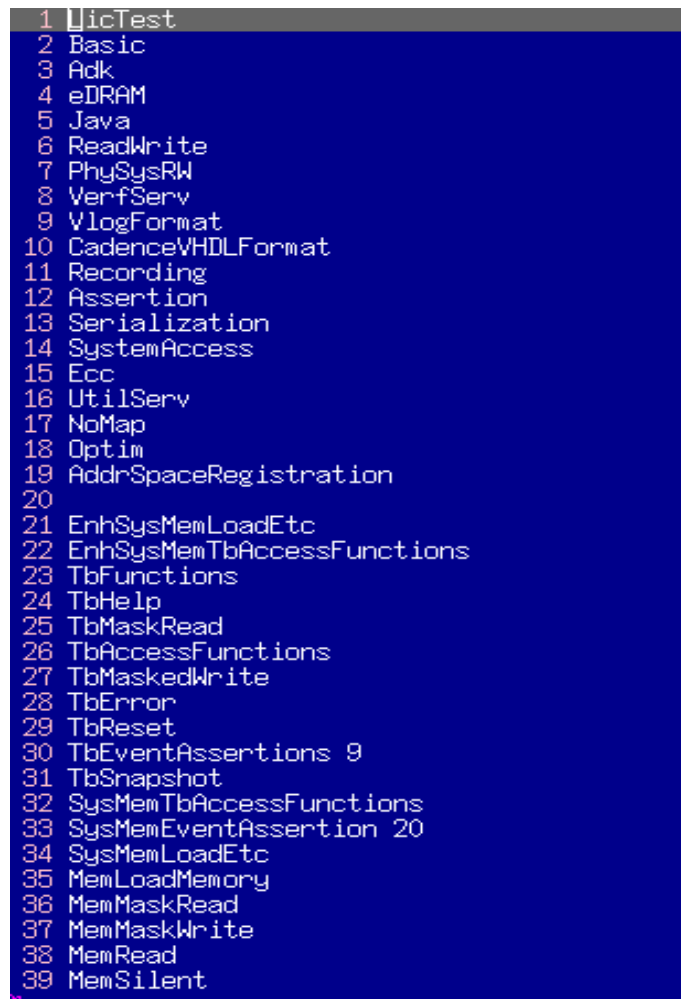
Fig. 7.39(b) 2x2.cde input

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.

```

RAM_2Kx8.v (tarfile:TC/Controller/VLOG_CODE) - GVIM9
File Edit Tools Syntax Buffers Window DrChip Help
7 TC/Controller/VLOG_CODE/RAM_2Kx8.v
1 ////////////////////////////////////////////////////////////////////
2 //
3 //
4 //          Module "RAM_2Kx8"
5 // This module is a Memory 2K Words and 8 bits wide memory
6 //
7 ////////////////////////////////////////////////////////////////////
8 //
9 //
10 module RAM_2Kx8 (CK, CSN, WEN, D, A, Q);
11
12 parameter Width = 8;
13 parameter AddrWidth = 11;
14 parameter Depth = 2048;
15
16 input CK, CSN, WEN;
17 input [Width-1:0] D; //Data
18 input [AddrWidth-1:0] A; //Address
19 output [Width-1:0] Q; //Output
20
21 integer MemAddr;
22 wire [Width-1:0] Qint; //Output
23 integer val;
24
25 initial
26 begin
27 // Registering SLM Memory
28 $slm_RegisterMemory(MemAddr, Depth, Width);
29 end
30
31 buf bufQ[Width-1:0] (Q, Qint);
32
33 always @ (posedge CK)
34 begin
35     if (CSN == 1'b0)
36     begin
37         if (WEN == 1'b1)
38             // Reading from a SLM Memory
39             $slm_ReadMemory(MemAddr, A, Qint);
40         else
41             // Writing into SLM Memory
42             $slm_WriteMemory(MemAddr, A, D);
43         end
44     end
45 end
46 endmodule

```

Fig. 7.42(a) Sample RTL Memory Design

```

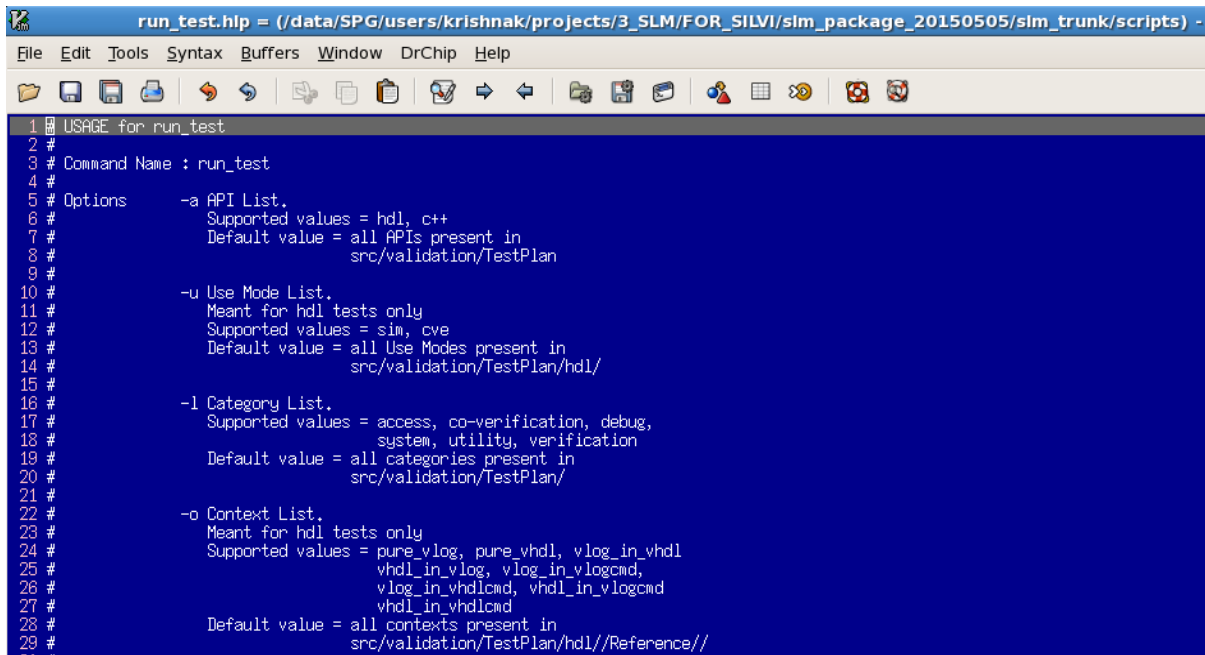
RAM_512x12.v (tarfile:TC/Controller/VLOG_CODE) - GVIM9
File Edit Tools Syntax Buffers Window DrChip Help
6 TC/Controller/VLOG_CODE/RAM_512x12.v
1 ////////////////////////////////////////////////////////////////////
2 //
3 //
4 //          Module "RAM_512x12"
5 // This module is a Memory 512 Words and 12 bits wide memory
6 //
7 ////////////////////////////////////////////////////////////////////
8 //
9 //
10 module RAM_512x12 (CK, CSN, WEN, D, A, Q);
11
12 parameter Width = 12;
13 parameter AddrWidth = 9;
14 parameter Depth = 512;
15
16 input CK, CSN, WEN;
17 input [Width-1:0] D; //Data
18 input [AddrWidth-1:0] A; //Address
19 output [Width-1:0] Q; //Output
20
21 integer MemAddr;
22 wire [Width-1:0] Qint;
23
24 initial
25 begin
26
27 // Registering a SLM Memory
28 $slm_RegisterMemory(MemAddr, Depth, Width);
29
30 // Resetting the registered SLM Memory with all 0s
31 $slm_ResetMemory(MemAddr, (Width{1'b0}));
32
33 end
34
35
36 buf bufQ[Width-1:0] (Q, Qint);
37
38 always @ (posedge CK)
39 begin
40     if (CSN == 1'b0)
41     begin
42         if (WEN == 1'b1)
43             // Reading from a SLM Memory
44             $slm_ReadMemory(MemAddr, A, Qint);
45         else
46             // Writing into a SLM Memory
47             $slm_WriteMemory(MemAddr, A, D);
48         end
49     end
50 end
51 endmodule

```

Fig. 7.42(b) Sample RTL Memory Design

## 7.2.5 Automation Scripts

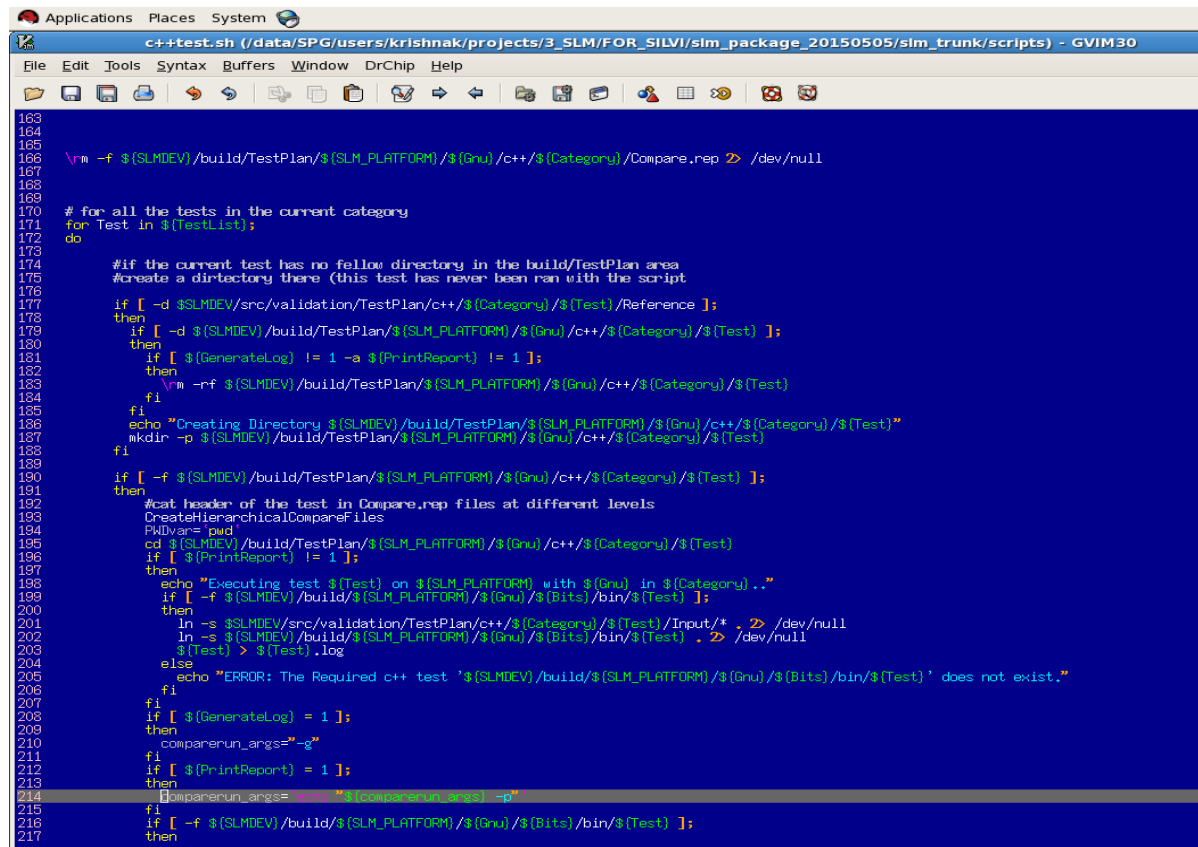
Fig. 7.43 shows the help for automation test-script usage.



```
run_test.hlp = (/data/SPG/users/krishnak/projects/3_SLM/FOR_SILVI/sim_package_20150505/slm_trunk/scripts) -
File Edit Tools Syntax Buffers Window DrChip Help
1 # USAGE for run_test
2 #
3 # Command Name : run_test
4 #
5 # Options
6 #   -a API List.
7 #     Supported values = hdl, c++
8 #     Default value = all APIs present in
9 #                   src/validation/TestPlan
10 #
11 #   -u Use Mode List.
12 #     Meant for hdl tests only
13 #     Supported values = sim, cve
14 #     Default value = all Use Modes present in
15 #                   src/validation/TestPlan/hdl/
16 #
17 #   -l Category List.
18 #     Supported values = access, co-verification, debug,
19 #                       system, utility, verification
20 #     Default value = all categories present in
21 #                   src/validation/TestPlan/
22 #
23 #   -o Context List.
24 #     Meant for hdl tests only
25 #     Supported values = pure_vlog, pure_vhdl, vlog_in_vhdl
26 #                       vhdl_in_vlog, vlog_in_vlogcmd,
27 #                       vlog_in_vhdlcmd, vhdl_in_vlogcmd
28 #     Default value = all contexts present in
29 #                   src/validation/TestPlan/hdl//Reference//
30 #
```

Fig. 7.43 Run\_tests.hlp

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



```
Applications Places System
c++test.sh (/data/SPG/users/krishnak/projects/3_SLM/FOR_SILVI/sim_package_20150505/slm_trunk/scripts) - GVIM30
File Edit Tools Syntax Buffers Window DrChip Help
163
164
165
166 \vrm -f $(SLMDEV)/build/TestPlan/$(SLM_PLATFORM)/$(Gnu)/c++/$(Category)/Compare.rep >> /dev/null
167
168
169
170 # for all the tests in the current category
171 for Test in $(TestList);
172 do
173
174 #if the current test has no fellow directory in the build/TestPlan area
175 #create a directory there (this test has never been ran with the script
176
177 if [ -d $(SLMDEV)/src/validation/TestPlan/c++/$(Category)/$(Test)/Reference ];
178 then
179 if [ -d $(SLMDEV)/build/TestPlan/$(SLM_PLATFORM)/$(Gnu)/c++/$(Category)/$(Test) ];
180 then
181 if [ $(GenerateLog) != 1 -a $(PrintReport) != 1 ];
182 then
183 \vrm -f $(SLMDEV)/build/TestPlan/$(SLM_PLATFORM)/$(Gnu)/c++/$(Category)/$(Test)
184 fi
185 fi
186 echo "Creating Directory $(SLMDEV)/build/TestPlan/$(SLM_PLATFORM)/$(Gnu)/c++/$(Category)/$(Test)"
187 mkdir -p $(SLMDEV)/build/TestPlan/$(SLM_PLATFORM)/$(Gnu)/c++/$(Category)/$(Test)
188 fi
189
190 if [ -f $(SLMDEV)/build/TestPlan/$(SLM_PLATFORM)/$(Gnu)/c++/$(Category)/$(Test) ];
191 then
192 #test header of the test in Compare.rep files at different levels
193 CreateHierarchicalCompareFiles
194 PKIDvars=paid
195 cd $(SLMDEV)/build/TestPlan/$(SLM_PLATFORM)/$(Gnu)/c++/$(Category)/$(Test)
196 if [ $(PrintReport) != 1 ];
197 then
198 echo "Executing test $(Test) on $(SLM_PLATFORM) with $(Gnu) in $(Category).."
199 if [ -f $(SLMDEV)/build/$(SLM_PLATFORM)/$(Gnu)/$(Bits)/bin/$(Test) ];
200 then
201 ln -s $(SLMDEV)/src/validation/TestPlan/c++/$(Category)/$(Test)/Input/* >> /dev/null
202 ln -s $(SLMDEV)/build/$(SLM_PLATFORM)/$(Gnu)/$(Bits)/bin/$(Test) >> /dev/null
203 $(Test) > $(Test).log
204 else
205 echo "ERROR: The Required c++ test '$(SLMDEV)/build/$(SLM_PLATFORM)/$(Gnu)/$(Bits)/bin/$(Test)' does not exist."
206 fi
207 fi
208 if [ $(GenerateLog) = 1 ];
209 then
210 comparerun_args="-g"
211 fi
212 if [ $(PrintReport) = 1 ];
213 then
214 comparerun_args="echo " $(Test) " -p"
215 fi
216 if [ -f $(SLMDEV)/build/$(SLM_PLATFORM)/$(Gnu)/$(Bits)/bin/$(Test) ];
217 then
```

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.

```

602 done
603
604
605 # Check if any platform has been specified or not
606 if [ -z "$platformlist" ];
607 then
608     echo "The platforms for tests is not mentioned. Enabling all the platforms as default"
609     platformlist="ncsim modelsim c++ c"
610 fi
611
612 # Check if any tests are specified or not
613 if [ -z "$testfile" -a -z "$tstest" ];
614 then
615     echo "No tests are specified Taking default lists of tests."
616     echo "Test list taken from $HOME/scripts/TestsList"
617     testfile=$HOME/scripts/TestsList
618 fi
619
620 # Check in case of a single test whether NoRuns has been mentioned
621 if [ ! -z "$tstest" -a -z "$NoRuns" ];
622 then
623     echo "The #runs is not mentioned. Taking 1 as default."
624     NoRuns=1
625 fi
626
627 # Start Executing tests
628 for platform in $platformlist;
629 do
630     if [ "$platform" != "ncsim" -a "$platform" != "modelsim" -a "$platform" != "c++" -a "$platform" != "c" ];
631     then
632         echo "Platform $PLATFORM is not supported. Exiting.."
633         exit
634     fi
635
636     echo "#####"
637     echo "          FOR $platform"
638     echo "#####"
639
640     if [ "$platform" = "ncsim" -a -z "$SimMode" ];
641     then
642         echo "The SimMode of simulation is not mentioned. Taking static as default."
643         SimMode=static
644     fi
645
646     if [ ! -z "$tstest" ];
647     then
648         TestName=$tstest
649         echo "TestNow"
650         RunTest
651     fi
652
653     if [ ! -z "$testfile" ];
654     then
655         exec $testfile
656         while read line;

```

Fig. 7.45(a) Automation Script

```

656     while read line;
657     do
658         printf "%(line)" | grep "[ ]" > /dev/null
659         if [ $? = 0 ];
660         then
661             TestNames=$(printf "%(line)" | awk '{print $1}');
662             NoFields=$(printf "%(line)" | awk '{print NF}');
663             if [ $(NoFields) -gt 1 ];
664             then
665                 NoRuns=$(printf "%(line)" | awk '{print $2}');
666             else
667                 NoRuns=1
668             fi
669             RunTest
670         fi
671     done
672 fi
673 done
674
675 if [ $(ErrorOccurred) = 0 -a $(GenerateLog) = 0 ];
676 then
677     printf "#####\n"
678     printf "## ALL TESTS PASSED SUCCESSFULLY ##\n"
679     printf "#####\n"
680 elif [ $(GenerateLog) = 0 ];
681 then
682     printf "#####\n"
683     printf "## SOME TESTS FAILED, CHECK ##\n"
684     printf "#####\n"
685 else
686     printf "#####\n"
687     printf "## COMPLETED ##\n"
688     printf "#####\n"
689 fi
690
691 #removing the components used in the test
692 if [ -f $(PATTERN_FILE) ];
693 then
694     rm -f $(PATTERN_FILE)
695 fi
696
697 if [ -f $(LOGFILE) ];
698 then
699     rm -f $(LOGFILE)
700 fi
701
702 if [ -f $(CMP_LOGFILE) ];
703 then
704     rm -f $(CMP_LOGFILE)
705 fi
706
707 if [ -f $(HELP_FILE) ];
708 then
709     rm -f $(HELP_FILE)
710 fi

```

Fig. 7.45(b) Automation Script

```

run_test (/data/SPG/users/krishnak/projects/3_SLM/FOR_SILVI/slm_package_20150505/sim_trunk/scripts) - GVM121
File Edit Tools Syntax Buffers Window DrChip Help
190 fi
191 fi
192 rm -f *.dsn *.trn *.xml
193 cd $(PWD)
194 }
195 }
196 }
197 RunNcsimTest()
198 {
199     cd /data/SPG/users/krishnak/projects/3_SLM/FOR_SILVI/slm_package_20150505/sim_trunk/scripts
200
201     if [ $GenerateLog = 1 ];
202     then
203         printf "Creating log for Test %s." "$TestName"
204         sc_ncsim auto $(SimMode) $(HbRuns) > /dev/null
205         mv ncsim.log $(HDL_REF_FILE_PATH)/$(TestName)/$(NC_REF_FILE_NAME)
206         cleantestdir
207         printf "DONE\n"
208         return
209     fi
210
211     printf "Executing Test %s." "$TestName"
212
213     sc_ncsim auto $(SimMode) $(HbRuns) > /dev/null
214
215     # check the run for correctness
216     if [ -f $(HDL_REF_FILE_PATH)/$(TestName)/$(NC_REF_FILE_NAME) ];
217     then
218
219         sed -e '/ncsim:\ v0[0-9]\.[0-9]0\.[0-9][0-9][0-9]/d'\
220             -e '/ncsim:.*\ Started\ on\ .* at\ /d'\
221             -e '/ncsim:.*\ Exiting\ on\ .* at\ /d'\
222             -e '/--\ DMAPI\ Version\ /d'\
223             -e '/Transaction Recording API Version/d'\
224             -e '/Loading\ snapshot\ .*module.*\ Done/d'\
225             -e '/Memory\ Usage\ .*program.*data.*total/d'\
226             -e '/DPU\ Usage\ .*system.*user.*total/d'\
227             -e '1,/^$/d' ncsim.log > $(LOGFILE)
228
229         sed -e '/ncsim:\ v0[0-9]\.[0-9]0\.[0-9][0-9][0-9]/d'\
230             -e '/ncsim:.*\ Started\ on\ .* at\ /d'\
231             -e '/ncsim:.*\ Exiting\ on\ .* at\ /d'\
232             -e '/--\ DMAPI\ Version\ /d'\
233             -e '/Transaction Recording API Version/d'\
234             -e '/Loading\ snapshot\ .*module.*\ Done/d'\
235             -e '/Memory\ Usage\ .*program.*data.*total/d'\
236             -e '/DPU\ Usage\ .*system.*user.*total/d'\
237             -e '1,/^$/d' $(HDL_REF_FILE_PATH)/$(TestName)/$(NC_REF_FILE_NAME) > $(CMP_LOGFILE)
238
239         diff $(LOGFILE) $(CMP_LOGFILE) > /dev/null
240
241         if [ $? -ne 0 ];
242         then
243

```

**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 discrepancy, is logged during this run.

```

File Edit Tools Syntax Buffers Window DrChip Help
1 -----D E T A I L S-----
2
3 TEST      =      scripts
4 GNU      =      slm_package_20150505
5 CATEGORY =      slm_trunk
6 PLATFORM =      linux
7 DATE     =      Thu Jun 25 18:11:35 IST 2015
8 STATUS   =      PASS
9
10 -----S U M M A R Y-----
11
12          scripts in category slm_trunk using slm_package_20150505 on linux: PASS
13
14 -----E N D   R E P O R T-----
15

```

**Fig. 7.46 Script Testing**

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

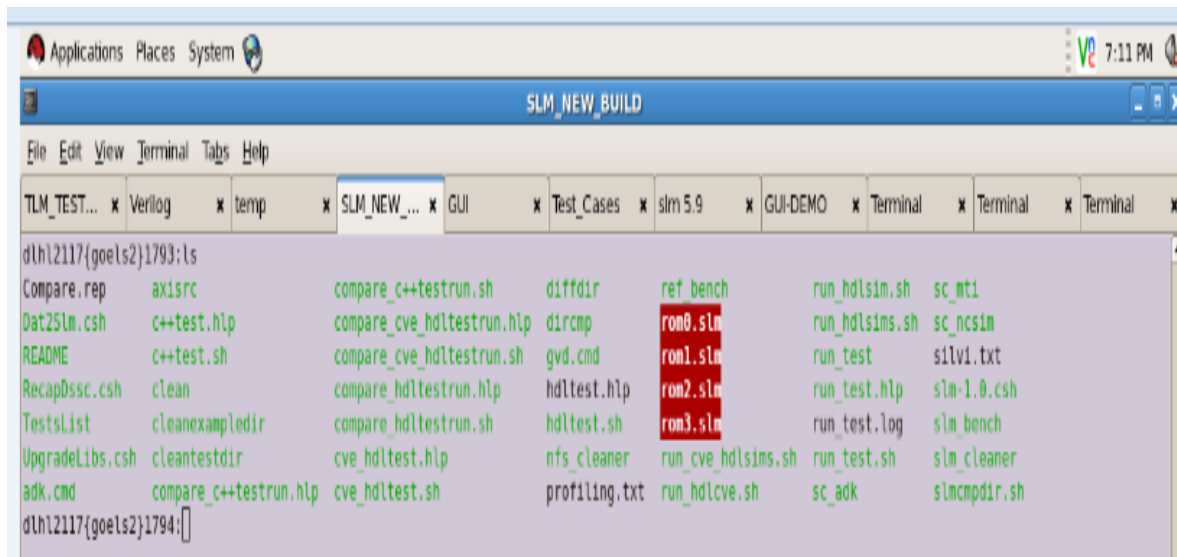


Fig. 7.47 Set of Scripts

## 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.

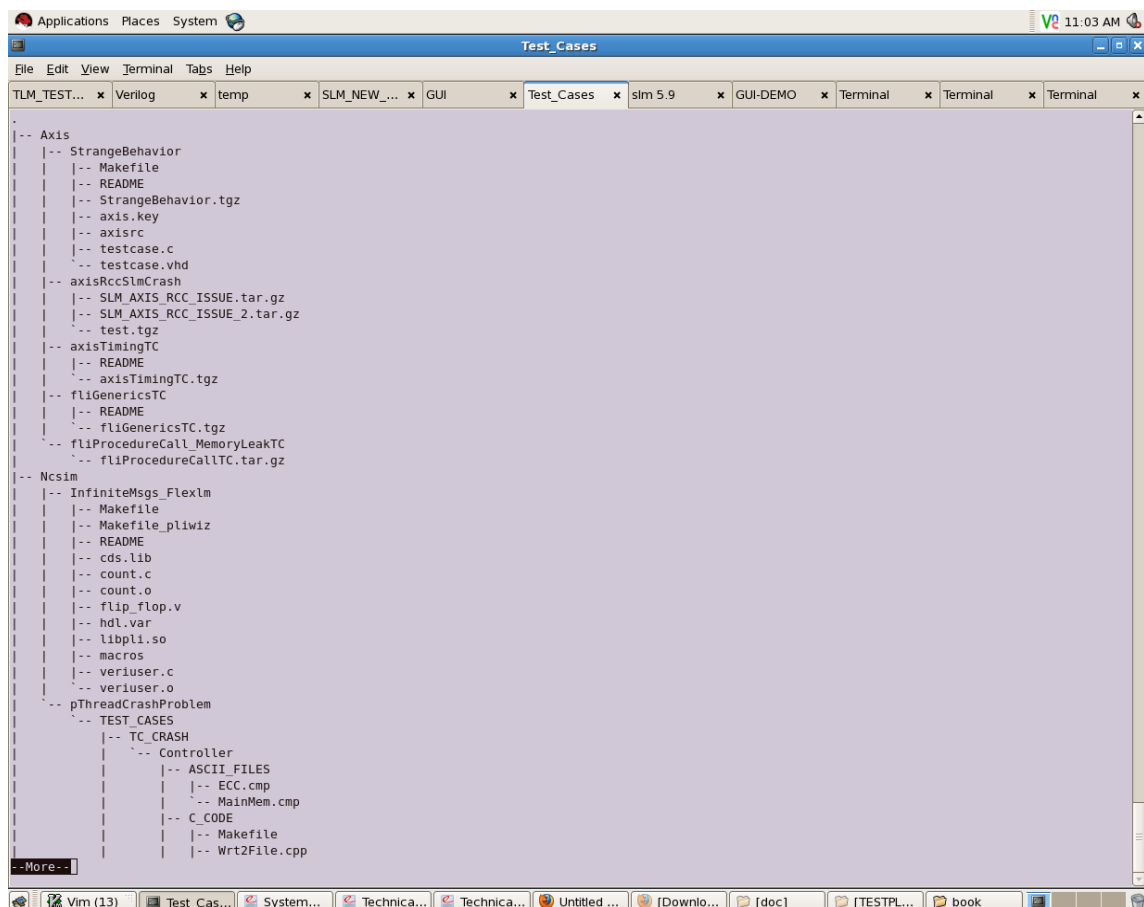


Fig. 7.48(a) Axis & Ncsim Directory Structure

```

Applications Places System 11:03 AM
Test_Cases
File Edit View Terminal Tabs Help
TLM_TEST... x Verilog x temp x SLM_NEW... x GUI x Test_Cases x sim 5.9 x GUI-DEMO x Terminal x Terminal x Terminal x

-- Seamless
|-- CPP_Exceptions_Ncsim_Crash
|   |-- Seamless_Ncsim_Exceptions_TC.tar.gz
|   |-- MixedSimulationNC
|   |-- C_SRC
|       |-- Makefile
|       |-- cve_MemoryCalls.c
|       |-- cve_MemoryCalls.h
|       |-- fmi_table.c
|       |-- fmi_table.h
|       |-- pli
|           |-- Makefile
|           |-- Wrt2File.cpp
|           |-- veriuser.c
|       |-- MemoryVal_VHDLtop
|           |-- README
|           |-- TEST_SUITE
|               |-- SP568D_128x8m4d4_ns_STIM_arch.vhd
|               |-- SP568D_128x8m4d4_ns_STIM_cfg.vhd
|               |-- SP568D_128x8m4d4_ns_STIM_ent.vhd
|               |-- SP568D_128x8m4d4_ns_TB_arch.vhd
|               |-- SP568D_128x8m4d4_ns_TB_cfg.vhd
|               |-- SP568D_128x8m4d4_ns_TB_ent.vhd
|           |-- VHDL
|               |-- SP568D_128x8m4d4_ns_arch.vhd
|               |-- SP568D_128x8m4d4_ns_cfg.vhd
|               |-- SP568D_128x8m4d4_ns_comp.vhd
|               |-- SP568D_128x8m4d4_ns_ent.vhd
|       |-- basic.cmd
|       |-- cds.lib
|       |-- cleandir
|       |-- config.cve
|       |-- hdl.var
|       |-- ncsim.noworks
|       |-- ncsim.works
|       |-- run_ncsim
|       |-- README
|       |-- run_testcase_no_works
|       |-- run_testcase_works
|-- Spexsim
    |-- mtIScheduleDriver_TC
    |   |-- README
    |   |-- SpexSim_TC.tar
35 directories, 119 files
dlhl2117(goels2)410:

```

**Fig. 7.48(b) Seamless Directory Structure**

```

Applications Places System 11:03 AM
Test_Cases
File Edit View Terminal Tabs Help
TLM_TEST... x Verilog x temp x SLM_NEW... x GUI x Test_Cases x sim 5.9 x GUI-DEMO x Terminal x Terminal x Terminal x

-- c++
|-- access
|   |-- AddrScram
|   |   |-- Input
|   |   |   |-- sdr_address.map
|   |   |-- Reference
|   |   |   |-- AddrScram.log
|   |   |-- Phy0.sav
|   |   |-- Phy1.sav
|   |   |-- Sys0.sav
|   |   |-- Sys1.sav
|   |-- Source
|   |   |-- AddrScram.cpp
|-- Basic
|   |-- Reference
|   |   |-- Basic.log
|   |-- Source
|   |   |-- Basic.cpp
|-- DynamicMA
|   |-- Reference
|   |   |-- DynamicMA.log
|   |-- Source
|   |   |-- DynamicMA.cpp
|-- PhySysRW
|   |-- Reference
|   |   |-- PhySysRW.log
|   |-- Source
|   |   |-- PhySysRW.cpp
|-- ReadWrite
|   |-- Reference
|   |   |-- ReadWrite.log
|   |-- Source
|   |   |-- ReadWrite.cpp
|-- co-verification
|   |-- AddrSpaceRegistration
|   |   |-- AddrSpaceRegistration.cpp
|   |   |-- Reference
|   |   |   |-- AddrSpaceRegistration.log
|   |   |   |-- Archi.xml
|   |   |   |-- DefaultDbi.xml
|   |   |-- Source
|   |   |   |-- AddrSpaceRegistration.cpp
|-- NoMap
|   |-- NoMap.cpp
|   |-- Reference
|   |-- Source
--More--

```

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

```

Applications Places System
Test_Cases
File Edit View Terminal Tabs Help
TLM_TEST... x Verilog x temp x SLM_NEW... x GUI x Test_Cases x sim 5.9 x GUI-DEMO x Terminal x Terminal x Terminal x
|-- hdl
|-- access
|-- AddrScram
|-- Input
|-- 2048x128.cde
|-- 2048x128_XValue.cde
|-- address_scram_10.map
|-- address_scram_11.map
|-- address_scram_12.map
|-- Reference
|-- cve
|-- axis
|-- pure_vhdl
|-- 2048x128.sav
|-- 2048x128_1.sav
|-- AddrScram.dsn
|-- AddrScram.trn
|-- axis_sim.log
|-- pure_vlog
|-- 2048x128.sav
|-- 2048x128_1.sav
|-- AddrScram.dsn
|-- AddrScram.trn
|-- axis_sim.log
|-- vhdl_in_vlog
|-- 2048x128.sav
|-- 2048x128_1.sav
|-- AddrScram.dsn
|-- AddrScram.trn
|-- axis_sim.log
|-- vlog_in_vhdl
|-- 2048x128.sav
|-- 2048x128_1.sav
|-- AddrScram.dsn
|-- AddrScram.trn
|-- axis_sim.log
|-- modelsim
|-- pure_vhdl
|-- 2048x128.sav
|-- 2048x128_1.sav
|-- AddrScram.dsn
|-- AddrScram.trn
|-- modelsim.log
|-- pure_vlog
|-- 2048x128.sav
|-- 2048x128_1.sav
--More--

```

Fig. 7.48(d) HDL Test Plan Directory Structure

```

Applications Places System
Test_Cases
File Edit View Terminal Tabs Help
TLM_TEST... x Verilog x temp x SLM_NEW... x GUI x Test_Cases x sim 5.9 x GUI-DEMO x Terminal x Terminal x Terminal x
|-- top.v
|-- sps6
|-- LIB
|-- Makefile.in
|-- README
|-- TEST_SUITE
|-- I_SPS68D_128x8m4d4_ns.v
|-- I_SPS68D_128x8m4d4_ns_arch.vhd
|-- I_SPS68D_128x8m4d4_ns_cfg.vhd
|-- I_SPS68D_128x8m4d4_ns_cfg_cve.vhd
|-- I_SPS68D_128x8m4d4_ns_cfg_slm.vhd
|-- I_SPS68D_128x8m4d4_ns_ent.vhd
|-- SPS68D_128x8m4d4_ns_STIM_arch.vhd
|-- SPS68D_128x8m4d4_ns_STIM_cfg.vhd
|-- SPS68D_128x8m4d4_ns_STIM_ent.vhd
|-- SPS68D_128x8m4d4_ns_TB_arch.vhd
|-- SPS68D_128x8m4d4_ns_TB_cfg.vhd
|-- SPS68D_128x8m4d4_ns_TB_ent.vhd
|-- SPS68D_128x8m4d4_ns_test.v
|-- VERILOG
|-- SPS68D_128x8m4d4_ns.v
|-- SPS68D_128x8m4d4_ns.v.orig
|-- VHDL
|-- SPS68D_128x8m4d4_ns_arch.vhd
|-- SPS68D_128x8m4d4_ns_cfg.vhd
|-- SPS68D_128x8m4d4_ns_comp.vhd
|-- SPS68D_128x8m4d4_ns_ent.vhd
|-- slm.vhd
|-- cds.lib
|-- hdl.var
|-- sc_mti
|-- sc_ncsim
|-- sc_ncvhdl
|-- vhdl-basic
|-- LIB
|-- VHDL
|-- slm.vhd
|-- top.vhd
|-- cds.lib
|-- gvd.cmd
|-- hdl.var
|-- ncSPS6.cmd
|-- sc_ncvhdl
|-- top.vhd
1131 directories, 3699 files
d:\hl2117\goels2\421:[]

```

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
```

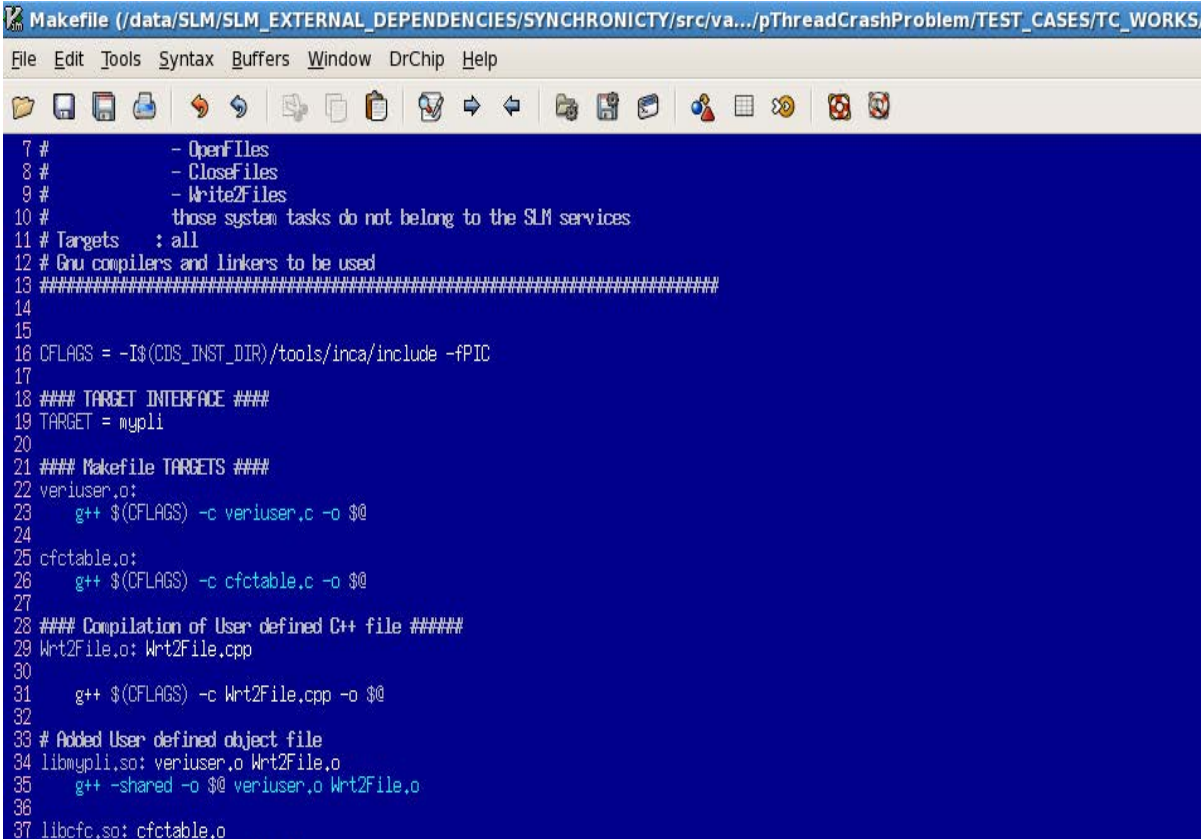
```
setenv INCA_64BIT
```

#### c. Instruction

```
bsub -I -q short ncoverilog 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 including veriuser.c and cfctable.c.



```
Makefile (/data/SLM/SLM_EXTERNAL_DEPENDENCIES/SYNCHRONICTY/src/va.../pThreadCrashProblem/TEST_CASES/TC_WORKS/
File Edit Tools Syntax Buffers Window DrChip Help
7 # - OpenFiles
8 # - CloseFiles
9 # - Write2Files
10 # those system tasks do not belong to the SLM services
11 # Targets : all
12 # Gnu compilers and linkers to be used
13 #####
14
15
16 CFLAGS = -I$(CDS_INST_DIR)/tools/inca/include -fPIC
17
18 #### TARGET INTERFACE ####
19 TARGET = mypli
20
21 #### Makefile TARGETS ####
22 veriuser.o:
23     g++ $(CFLAGS) -c veriuser.c -o $@
24
25 cfctable.o:
26     g++ $(CFLAGS) -c cfctable.c -o $@
27
28 ##### Compilation of User defined C++ file #####
29 Wrt2File.o: Wrt2File.cpp
30
31     g++ $(CFLAGS) -c Wrt2File.cpp -o $@
32
33 # Added User defined object file
34 libmypli.so: veriuser.o Wrt2File.o
35     g++ -shared -o $@ veriuser.o Wrt2File.o
36
37 libcfc.so: cfctable.o
```

Fig. 7.49 Makefile for CFCTable & PLI Setup

```

Applications Places System
Verilog
File Edit View Terminal Tabs Help
NEW_SLM_GUI x SLM_BUILD_6_May x TLM_TEST_PLT x Verilog x Working_Flexperf x temp x Flexperf_GUI_6.1.0 x
dlh2117(goels2)330:bsub -I -q short ncverilog poc.v
Job <1394913> is submitted to queue <short>.
<<Waiting for dispatch ...>>
<<Starting on dlh0984>>
ncverilog(64): 13.20-s003: (c) Copyright 1995-2014 Cadence Design Systems, Inc.
file: poc.v
  module worklib.hello_world:v
    errors: 0, warnings: 0
    Caching library 'worklib' ..... Done
  Elaborating the design hierarchy:
  Building instance overlay tables: ..... Done
  Generating native compiled code:
    worklib.hello_world:v <0x02d6953f>
      streams: 1, words: 848
  Building instance specific data structures.
  Loading native compiled code: ..... Done
  Design hierarchy summary:
    Instances Unique
    Modules:      1      1
    Initial blocks: 1      1
  Writing initial simulation snapshot: worklib.hello_world:v
  Loading snapshot worklib.hello_world:v ..... Done
ncsim> source /sw/cadence/incisiv/13.20.003/linux86/tools/inca/files/ncsimrc
ncsim> run
Hello World
My first C++ interaction with VerilogSimulation complete via $finish(1) at time 0 FS + 0
./poc.v:6 $finish;
ncsim> exit

```

**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.

```

Applications Places System
files_setup (~/Desktop/thesis_screens/thesis_new) - GVIM6
File Edit Tools Syntax Buffers Window DrChip Help
25
26
27
28 * File name
29 /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/GUI_SLM/DataDesign.h
30
31 Purpose
32 - this gives all basic data types of SLM objects and object containers
33 - this also tell about how incrementally the format of memory data structure is laid up in the whole
  Flexperf framework
34 - More idea of its data structure can be taken from this file
35 - Their individual get/set functions are available in the corresponding cpp files.
36   for eg: get/set for attributes in Memory Info will be in Memory.cpp
37   get/set for attributes in Mem Info will be in Mem.cpp
38
39 More files of use w.r.t it:
40 /data/SLM/SLM_EXTERNAL_DEPENDENCIES/tp2/install_64/SDK_FlexPerf_2.3.0/x86_64-pc-linux-gnu-gcc-4.1.1//
  include/FP2/Memory.hpp (666)
41 /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm/src/cpp/slm_core/Mem.hpp (1067)
42
43
44
45
46
47
48 * File name
49 /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm/src/cpp/slm_core/Scratchpad.cpp (134,
  223, 252)
50
51 Purpose
52 - this is the identified plugin point for GUI
53 - any information about slm core can be extracted by plugging code here, b/c
54   of the fact top flow of almost all functions are defined in this file
43,0-1 16%

```

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

```

46
47
48 * File name
49 /data/SPG/users/goels2/tcl_file/SLM_GUI/Integration/SLM_Lib/slm/src/cpp/slm_core/Scratchpad.cpp (134,
223, 252)
50
51 Purpose
52 - this is the identified plugin point for GUI
53 - any information about slm core can be extracted by plugging code here, b/c
54 of the fact top flow of almost all functions are defined in this file
55 - Note: one function is overloaded in the form arguments signature, so this
56 need to be taken care
57 -
58 - All functions are important since they are the easiest exposed interface to user. Following these one
can reach to correct point of analysis in core.
59 - These can be bypassed only after we get a handle to some component in Slm Core.
60 - For interaction with outside world such as GUI or HDL simulations these will
61 be exposed points of communication.
62 - Internal functions are meant to be used within the SLM Core.
63
64 Major function mappings in SLM which are used for extracting information for GUI:
65 TbView ----->>> MemorySnapshot()
66 TbInfo ----->>> HierarchyBrowser()
67 TbInfoAssertions ----->>> InfoAssertions()
68
69 More study:
70
71 a. TbBuildSys ----->>> BuildSys()
72 [create heirarchy..... "Basically system memory is a logical container containing two physical
memories"]
73 (lines: 1328 (adding daughter memory to system memory))
74 (line: )
75
76 b. Memory Registration ----->>> RegisterMemory(), RegisterSystemMemory()

```

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

```

55 - Note: one function is overloaded in the form arguments signature, so this
56 need to be taken care
57 -
58 - All functions are important since they are the easiest exposed interface to user. Following these one
can reach to correct point of analysis in core.
59 - These can be bypassed only after we get a handle to some component in Slm Core.
60 - For interaction with outside world such as GUI or HDL simulations these will
61 be exposed points of communication.
62 - Internal functions are meant to be used within the SLM Core.
63
64 Major function mappings in SLM which are used for extracting information for GUI:
65 TbView ----->>> MemorySnapshot()
66 TbInfo ----->>> HierarchyBrowser()
67 TbInfoAssertions ----->>> InfoAssertions()
68
69 More study:
70
71 a. TbBuildSys ----->>> BuildSys()
72 [create heirarchy..... "Basically system memory is a logical container containing two physical
memories"]
73 (lines: 1328 (adding daughter memory to system memory))
74 (line: )
75
76 b. Memory Registration ----->>> RegisterMemory(), RegisterSystemMemory()
77
78 Only physical memory will hold actual data buffer.
79 the data rendering between physical to system memory and vice-versa
80 is done through functions in SystemMemory.cpp
81
82
83 c. instead of using GetComponent(i), we can take handle to memory using GetPhysicalMemory(id),
(lines: 916)

```

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 20<sup>th</sup> 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 *5<sup>th</sup> 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](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 Embedded 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>]