Re-engineering of SLM with Automated Testing

Submitted By

Jayendra Vyas

14MCEC30

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2016

Re-engineering of SLM with Automated Testing

Major Project

Submitted in complete fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Jayendra Vyas

(14MCEC30)

Guided By

Prof. Jitali Patel

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2016

Certificate

This is to certify that the major project entitled **"Re-engineering of SLM with automated testing"** submitted by **Jayendra Vyas (Roll No: 14MCEC30)**, towards the complete fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part-II, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Jitali Patel
Guide & Associate Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Priyanka Sharma
Professor,
Coordinator M.Tech - CSE
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr P.N. Tekwani
Director,
Institute of Technology,
Nirma University, Ahmedabad

Certificate

This is to certify that the major project entitled **"Re-engineering of SLM with automated testing"** submitted by **Jayendra Vyas (Roll No: 14MCEC30)**, towards the fulfillment of the requirements for the award of degree of Master of Technology in Computer Science & Engineering (CSE) of Nirma University, Ahmedabad, is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination.

Date:

Project guide

Mr Krishna Kumar

CPD Department,

STMicroelectronics, India

Statement of Originality

I, **Jayendra Vyas**, Roll. No. **14MCEC30**, give undertaking that the Major Project entitled "**Re-engineering of SLM with automated testing**" submitted by me, towards the accomplishment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Prof. Jitali Patel

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Prof. Jitali Patel**, Associate Professor, Computer Science Department, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout this work. The appreciation and continual support he has imparted has been a great motivation to me in reaching a higher goal. Her guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Sanjay Garg**, Hon'ble Head of Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr P.N. Tekwani**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation he has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Jayendra Vyas

14MCEC30

Abstract

This project is about re-engineering of SLM model where SLM stands for system level memory. SLM is a memory model used internally in ST Microelectronics to simulate memory in verification phase of SoC design. Existing SLM is based on flexperf library which is obsolete now. So, We are re-engineering SLM with re-creating classes which are independent of flexperf library functions. Automation of existing testcases of SLM allows developers to test code regressively. SLM GUI is also an integral part of this project which provides different views for memory instances .

Abbreviations

SLM	System Level Memory
GUI	Graphical User Interface
HDL	Hardware Description language
VHDL	Verilog Hardware description language
SoC	System On Chip

Contents

List of Figures

Chapter 1

Introduction

1.1 Our Team

Our team work with the development of software models for IPs and specifically this report work is 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.

1.2 What is SLM Model?

In embedded system SoC is used to exhibit its functionalities. Fabrication of SoC is a very time consuming and costly process. So, it is designed very precisely and simulated to check its performance. SoC design process consists of following phases.

SoC Design Process Steps :-

1. Architecture Design
2. IP selection
3. Verification.
4. Integration
5. Validation

6. Physical Synthesis

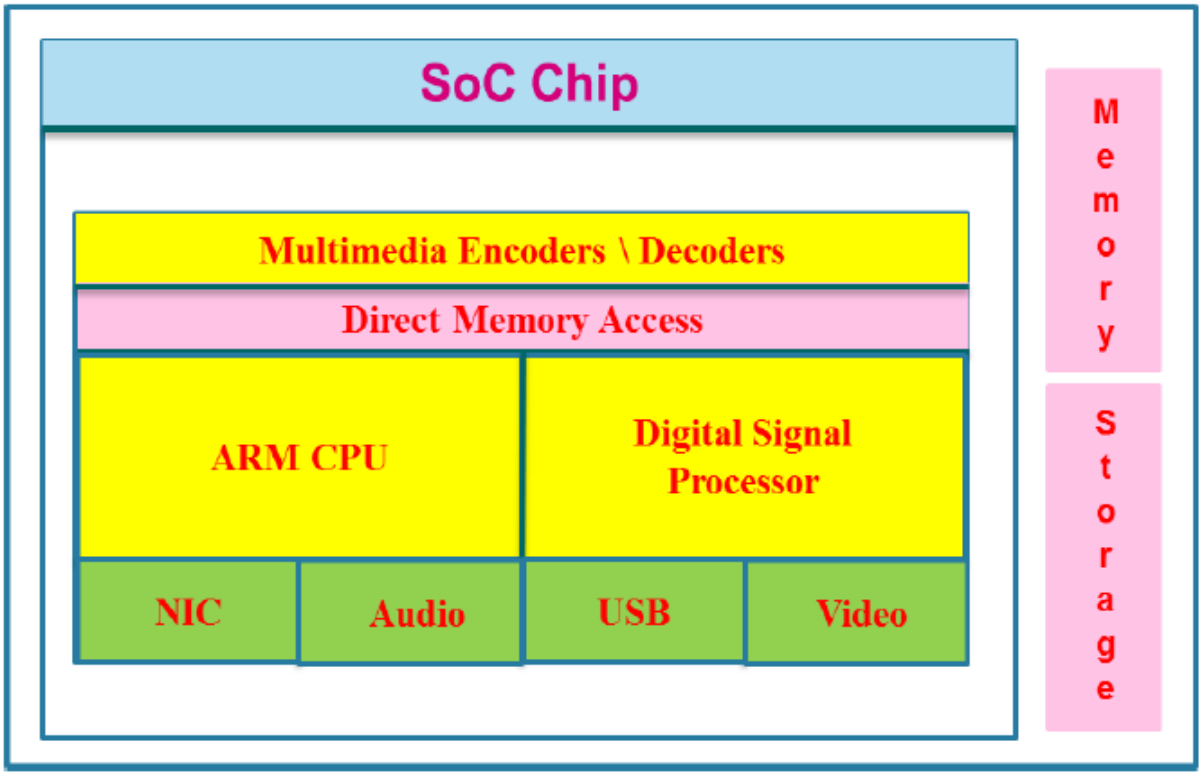

Figure 1.1: SoC design Components

Verification is the phase which takes maximum time in design process. In this phase various components are simulated on simulator like cadence, mentor graphics etc. Different component models are used in order to check the performance.
SLM model is used for simulating memory in ST Microelecronics. SLM model provides simulation of memory to add ease for developers in debugging memory in design process. Major functionalities of SLM are memory browsing and backdoor access to memory instances. Apart from that it provides wide range of functionality.
There are 3 major components in re-engineering of SLM

1. Classes of SLM

   They depict functionality of SLM in following categories.

(a) Utility

(b) Verification

(c) Recording

(d) Access

(e) Co-verification

(f) Debug

2. SLM GUI

   It provides memory browsing for different memory instances to add ease in debugging. It provides information about different memory instances. It has reach set of features like hierarchical view, Property view, searching data at specific address, range of data. Currently it is standalone software developed in C++/Tk.

3. Automated Testcases

   These testcases are automated to be executed when we build SLM. It regressively checks for any differences in expected output and obtained output. Every testcase generates log while running testcase. It has following 2 test suits.

   (a) HDL testsuit

   It contains testcases in verilog. They Check for HDL Code functionality in different simulators environment.

   (b) CPP testsuit

   It contains testcases in C++. They check for C++ code functionality for SLM core.

Chapter 2

Literature survey

2.1 Existing SLM:

As mentioned earlier, SLM is a memory model to simulate memory in verification process of SoC design.It is ST Microelctronics's proprietary software. It is used internally in projects to simulate memory on different simulators. It can be used to verify anu functional design that require complex memory operations.
SLM is a mixed HDL/C++ model. It reduce the physical memory overhead by simply giving a 'C' moemory view to the physical memory. It is heavily integrated with flexperf library functions. Flexperf library is again ST Microelecronics's internal tool. This library is very old and not compatible with modern GCC. So, there is a need to re-engineer SLM Classes which are independent of flexperf library functions.
SLM provides services in following categories.

1. Utility
2. Verification
3. Recording
4. Access
5. Co-verification

SLM GUI is developed with SLM to provide GUI to end users and developers to view and access instances. It facilitates developers in debugging.
There are two test suits in co-ordination of SLM to test code regressively. HDL test

suit checks HDL code and CPP test suite checks CPP code in order to check fro any differences between expected outcome and obtained outcome.

2.2 SLM GUI

It is used to communicate with SLM Core. It allows developers to browse memory and zero time access to memory instances.SLM GUI was initially implemented in JAVA. In current version it is implemented in CPP/Tk which is very light in comparison of JAVA version.

In the current version it provides services to display memory instances. It also provides different views like Property view which gives detail about properties of current memory instance, Hierarchical view which gives relation of daughter memory instance with other parent memory and Memory grid view which gives grid view for current memory instance. It also allows to dump memory instance in required format.

2.3 Testcases

Testcases for HDL and CPP check code for identifying any differences in obtained output after modification in given code. These testcases are automated to check code regressively and less time consuming.

Testcases are built with the same time when SLM is built. if they are not automated then it is very lengthy and complicated process.

Testcases contains test cases for different perspectives. For example , Access category from CPP test suit contains testcase which checks address scrambling in SLM. It yields results whether address scrambling has been affected or not after modification in code.

Chapter 3

Technological Review

3.1 Language and Tools

SLM models are mixed HDL/C++ memory models. They correspond to HDL functional architectures where assignments of the memory models are replaced by C/C++ subroutine calls. This way the storage becomes C storage in place of the former HDL register/variable . To connect HDL logic together with the C array and related access functions, the simulator-specific HDL/C interface are used. SLM C storage and services are delivered as prebuilt C dynamic libraries.

3.1.1 C++/Tk

SLM GUI is implemented in C++/Tk. It is an interface to the Tk GUI toolkit. C++/Tk uses templates, operator overloading and implicit conversions. Each C++/Tk expression ends up as equivalent Tcl/Tk command in the string format which is passed to underlying Tcl interppreter. It means that C++/Tk uses and depends on Tcl/Tk environment.[1]

3.2 Supported Operating Systems

1. Solaris
2. Linux

3.3 Supported Logic Simulators

1. Cadence NCsim
2. Mentor Modelsim

3. Synopsis vcs-mx

4. Verisity Xsim (Axis)

3.4 SLM functionalities

SLM provides functionalities in basically 2 categories.

1. Internal functions for SLM Model providers

   (a) Registration

   (b) Read and write access

   (c) Subset of verification features (Load, Reset system memory)

2. External functions for End users

   (a) Utility

   These are help and information functions.

   (b) Verification

   Verification functions are meant to be used in either in the HDL test bench or from logic simulator console. They rely on backdoor access. They are executed in 0-HW time. Accesses to SLM cuts are not simulated by the logic simulator hence reducing simulation time.

   (c) Recording

   These functions help in logging the memory events in different formats .

   (d) Debug

   These functions are used for debugging purpose.

   (e) System

   These functions are related to creation and building of system memories.

   (f) HW/SW co-verification

   Functions used in parallel of hardware design.

3.5 How SLM 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.

3.6 SLM GUI

SLM GUI can be divided in following components.

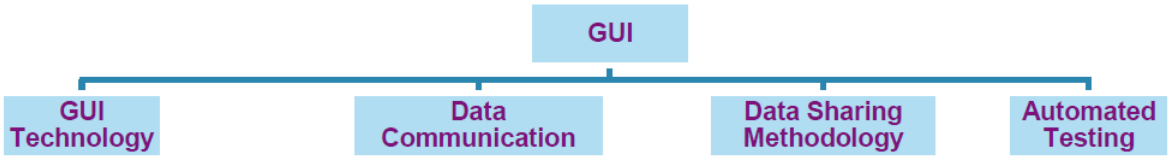

Figure 3.1: GUI Components

Chapter 4

Requirement Analysis

4.1 Objectives for SLM GUI

4.1.1 Functional Objectives

1. Debug existing SLM GUI code implemented in C++/Tk

   SLM GUI was a stand alone software earlier. Debugging was required to ensure functionalities which were supposed to be delivered.

2. Optimize code

   it includes identifying redundant functions, variables and memory leaks. As some of the functions may not be contributing in depicting functionality of SLM GUI. Some global variables may also not be used in code. SLM GUI takes memory instances as input file and generates various log and error files. So, It is also important to check traces of memory leaks.

3. Add Partial Search feature

   Existing SLM GUI was exhibiting search feature which gave an address of data we look for. But, it did not support partial search. In partial search, GUI finds all partial matches at binary level and displays.

4. Integrating SLM GUI into SLM Core

   Intially it was standalone application. Afterwards need arose to integrate it into SLM core. So that it can get invoked whenever we build SLM Core. All the files are directly passed from SLM core to SLM GUI to be displayed by it. Initially the

files which we wanted to display on SLM GUI were used to be dumped in a specific location and GUI was accessing those files from those locations.

4.1.2 Software Objective

1. Modular and extensible

2. Simple integration setup

4.2 Objectives for SLM core

4.2.1 Functional Objective

1. Abstract analysis of SLM core

   This objective is about to understand relations between existing files. It includes understanding flow of SLM Core.

2. Analysis of plugin point for SLM GUI

   Currently SLM GUI is stand alone software. It is not integrated with SLM. To allow SLM GUI communicate directly with SLM, we need to integrate it with SLM core. So that we can invoke SLM GUI from within SLM.

3. Analysis of classes to be re-engineered

   Before re-engineering the SLM Core classes, through analysis was required to carry out re-engineering.

4. Re-engineering of classes

   After thorough analysis of classes built with flexperf library functions, new classes were supposed to be developed in order to provide dependency on flexperf library.

5. Integration of re-engineered classes

   All re-engineered classes were supposed to be integrated to ensure functionalities. Separate testcases were also supposed to be developed to ensure individual functionalities.

4.3 Objective for Test cases

4.3.1 Functional objectives

1. Abstract analysis of existing test suits

   Initially there were two test suits

   (a) HDL Test suit

   It contains test cases to check whether they are carrying out functionality. Every functionality is checked with content of different simulators.

   (b) CPP test suit

   It contains test cases to check functionality of C++ code whether they are carrying out functionality.

2. Automate C++ testcases

   It is cumbersome to run testcases manually. There are 25 testcases implented in C++ checking different aspects of SLM. They need to be automated to check test results without manual intervention. We can set specific flags while building SLM which can also build test cases at same time.

   They generate log files and error reports to add ease in debugging. These log files and error reports are very helpful in case some errors are there in code. Automation of testcases saves plenty of time and makes debugging very simple.

Chapter 5

Implementation details

5.1 Snapshots of SLM GUI

5.1.1 SLM GUI architecture

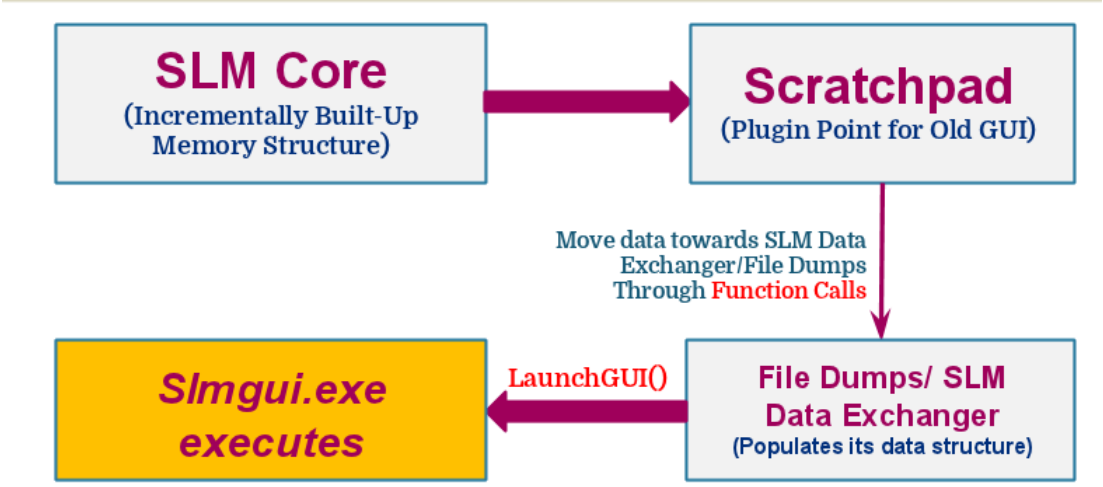

Figure 5.1: SLM GUI architecture

5.1.2 SLM GUI communication process

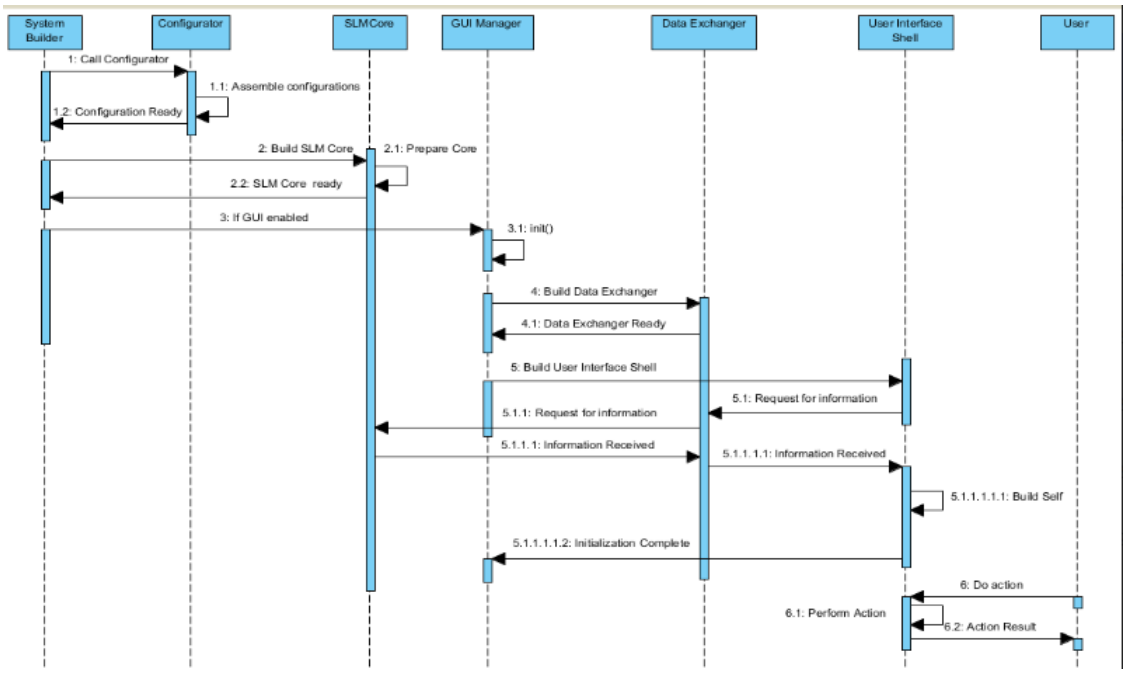

Figure 5.2: Overall Communication process

5.1.3 SLM GUI splash screen

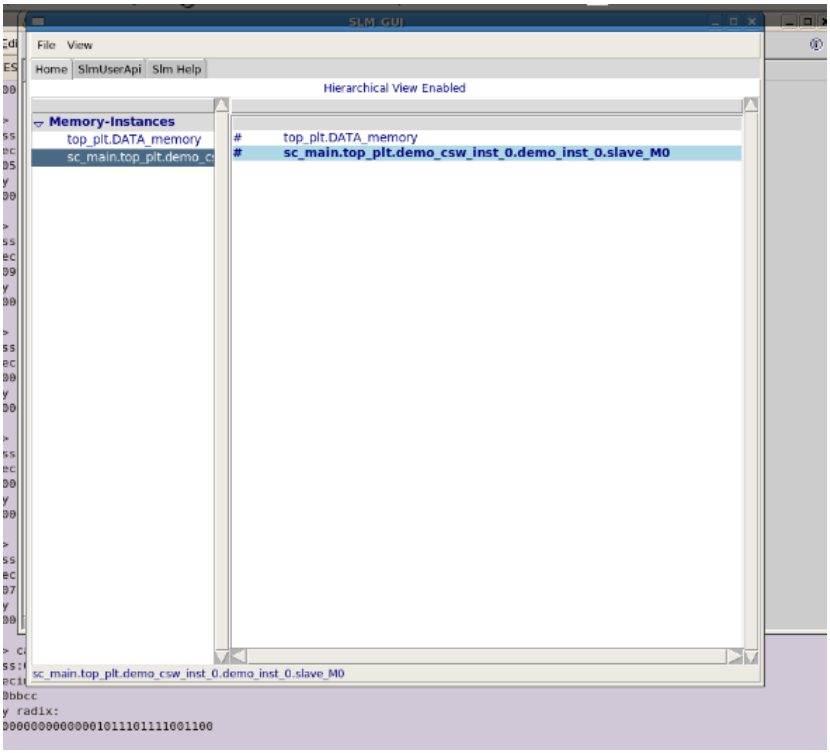

Figure 5.3: SLM GUI Home

5.1.4 SLM GUI property view window

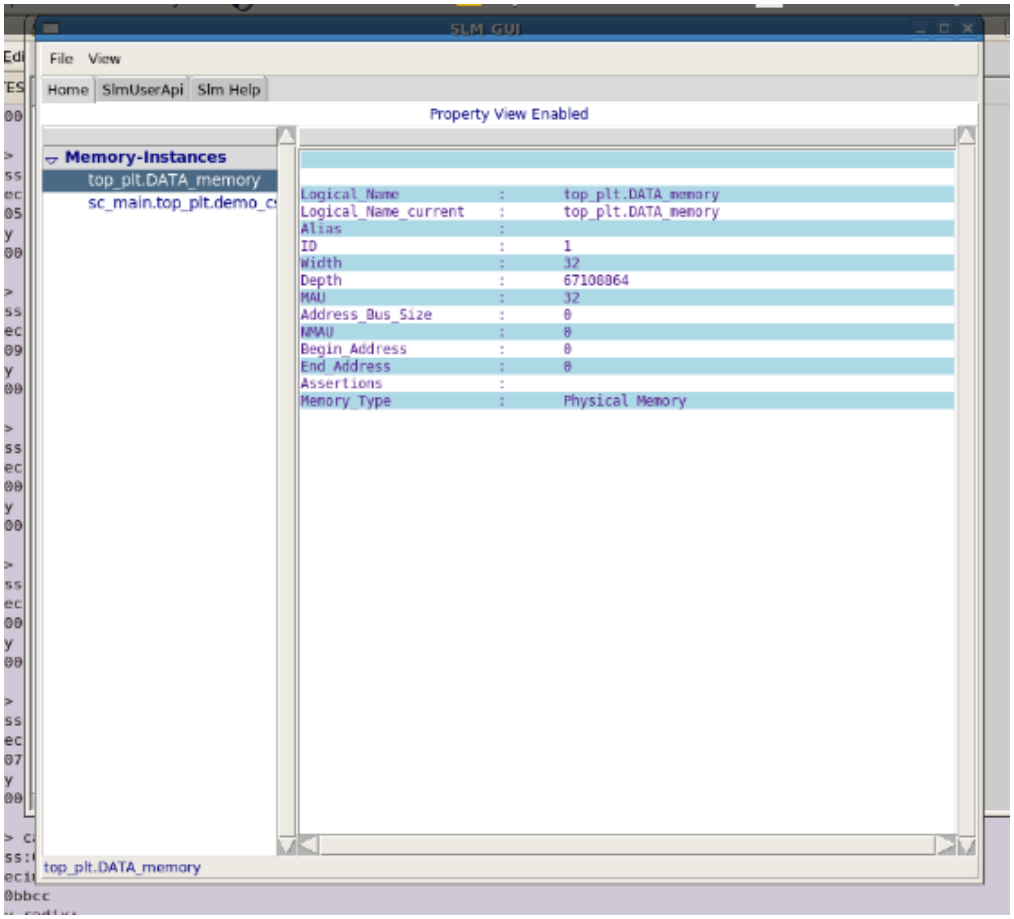

Figure 5.4: SLM GUI Property View

5.1.5 SLM GUI Output window

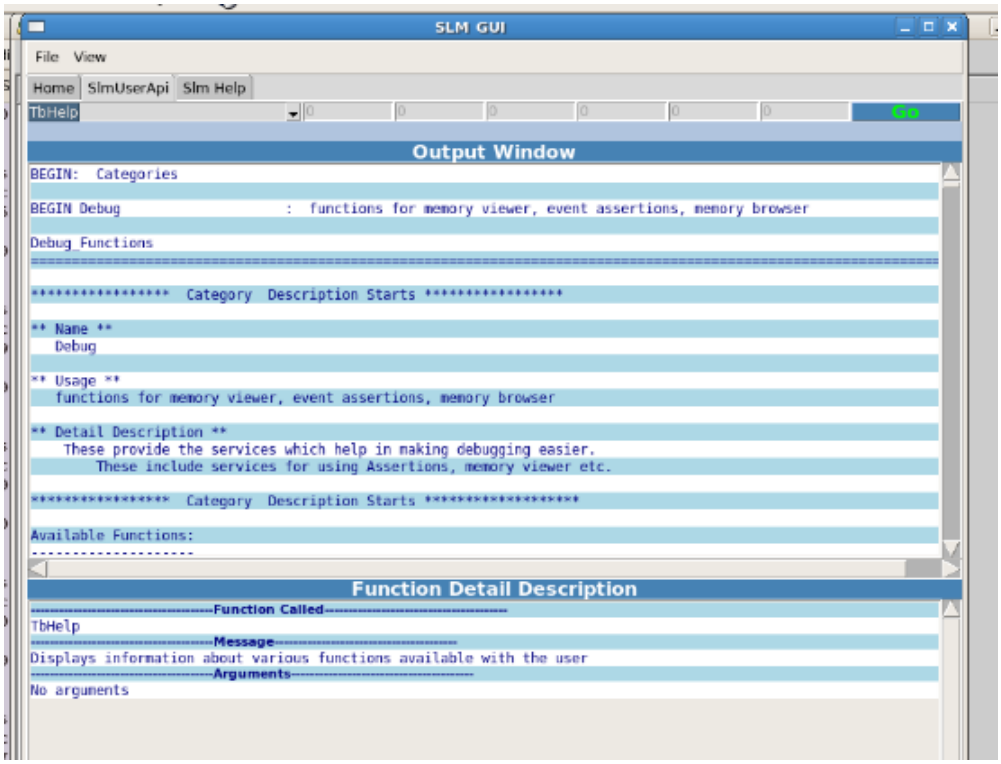

Figure 5.5: SLM GUI Output Window

5.1.6 SLM help command in Unix

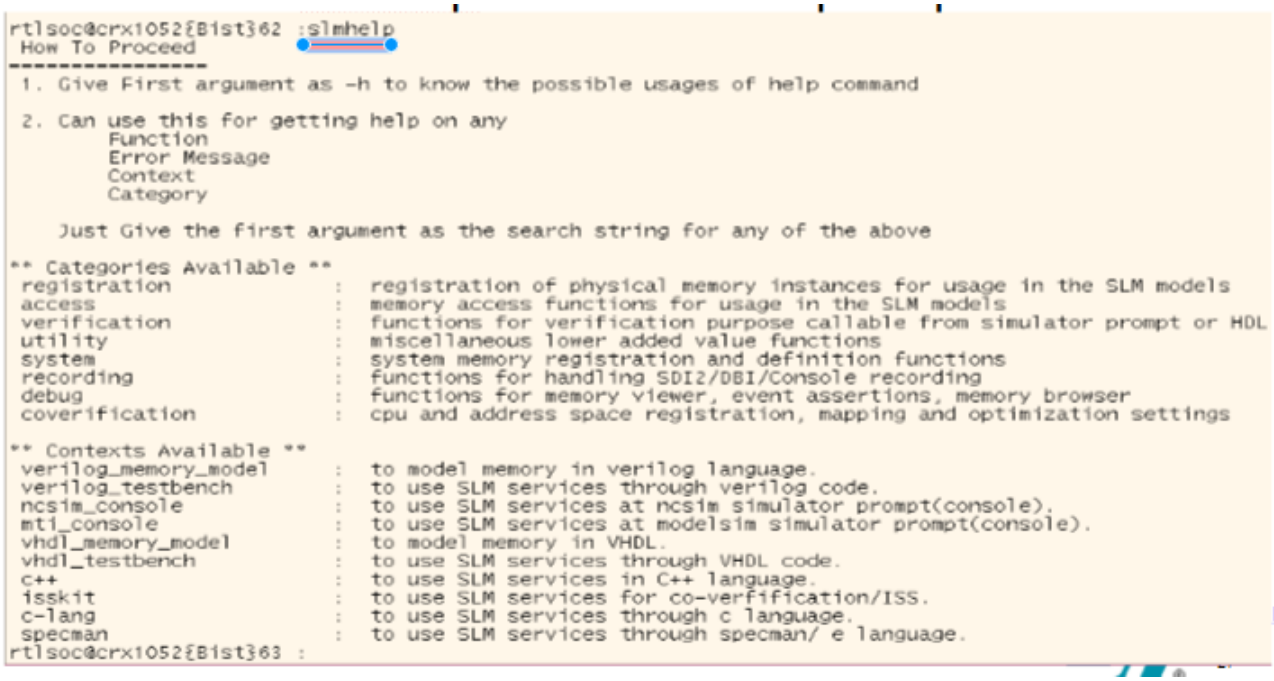

Figure 5.6: SLM help command in Unix

5.1.7 System Memory instance Properties

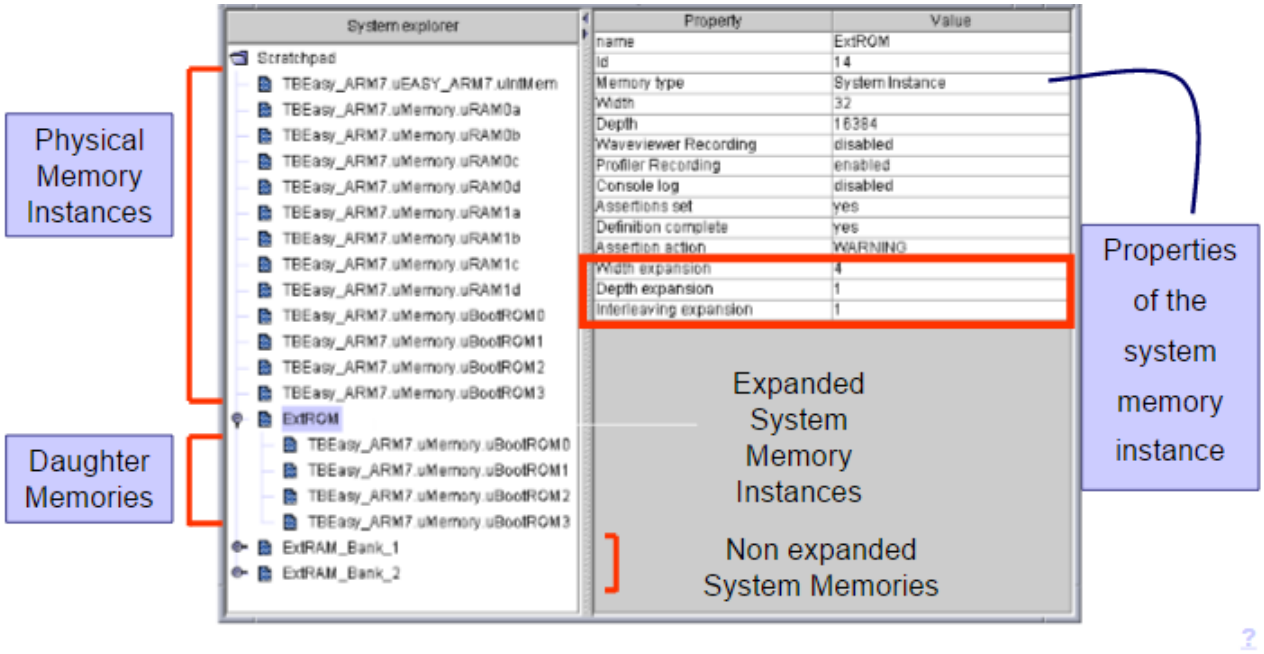

Figure 5.7: System Memory instance Properties

5.1.8 Daughter Memory instance Properties

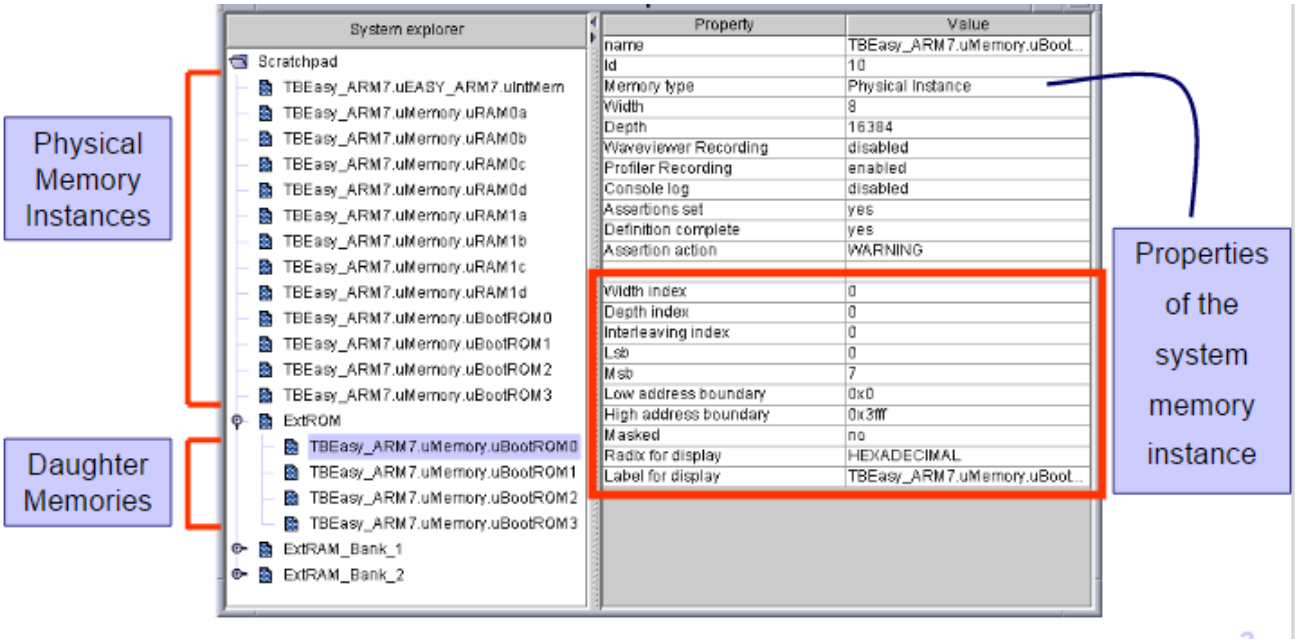

Figure 5.8: Daughter Memory instance Properties

5.1.9 Memory Layout

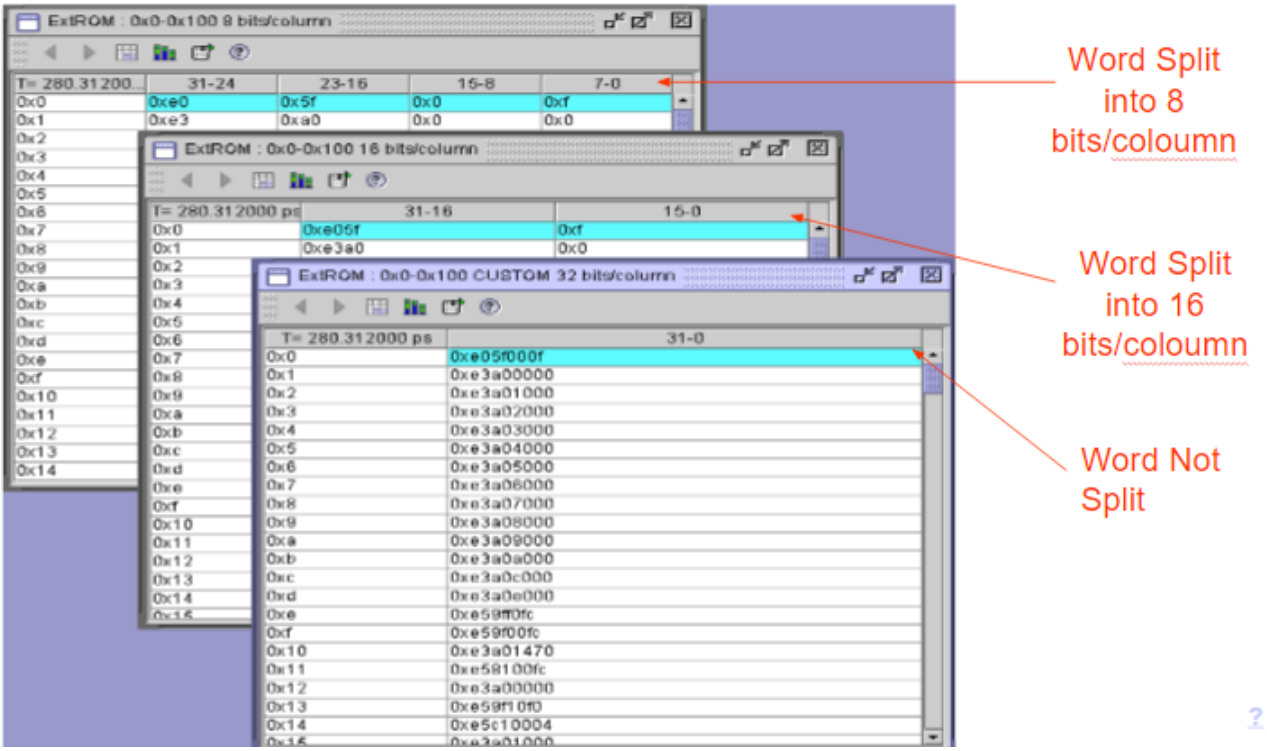

Figure 5.9: Memory Layout

5.2 Objectives achieved

We divided this project into 3 major components

1. SLM GUI
    (a) Analysis of SLM GUI
    (b) Optimize existing code
    (c) Partial search feature
    (d) Differentiate Hexadecimal and decimal mode in displaying memory content
    (e) Integrate Partial search feature into SLM GUI
    (f) Analysis for SLM GUI plugin oint into SLM Core
    (g) Integrate SLM GUI into SLM Core
    (h) Extending command line arguments set for SLM GUI
    (i) Internal release of SLM GUI
2. Testcases
    (a) Analysis of C++ testsuit
    (b) Optimize testcases
    (c) Automate C++ testcases
    (d) Integrate C++ testcases into SLM Core
    (e) Study of VHDL testcases
3. SLM Core
    (a) Analysis of existing code
    (b) Design of new classes
    (c) Re-engineer classes which are independent of flexperf library functions

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

This project is about Re-engineering. So, in the first phase project activities were largely dominated by analysis process. Initially SLM GUI was stand alone software. After adding features related to search it was integrated with SLM Core.
Existing SLM code is decade old and heavily integrated with flexperf library functions. This library is obsolote now. In the next phase major project activities were dominated by eleminating dependency of classes to flexperf library functions.

6.2 Future Scope

6.2.1 Objectives for SLM GUI

1. Add new features into SLM GUI

   Currently all the objectives related to SLM GUI is achieved. In future we can add new features if the requirement arises.

6.2.2 Objective for SLM Core

1. Analysis of SLM core classes which are dependent to flexperf library functions

   Thorough study of classes with dependency to flexperf library function will be required in next phase in order to re-engineering of them.

2. Re-engineering of remaining classes

   This is the central activity of this project. In this project activity, new classes which

are independent of flexperf library functions will be re-engineered.

6.2.3 Objectives for testcases

1. Automation of HDL testcases

   Currently HDL test suit is in debugging phase. After this phase they will be automated to facilitate developers to test their code.

2. Re-engineering of testcases for re-engineered classes

   It may be required to re-engineer test cases for both cpp and HDL test suit to test code for re-engineered classes.

Bibliography

[1] M. Sobczak, "C++/tk documentation." http://cpptk.sourceforge.net, 2015.