

Migration of MEM ACS to automated simlist generation flow

Major Project Report

*Submitted in fulfillment of the requirements
for the degree of*

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Amit Jangir

(17MECE07)



Electronics & Communication Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481
December-2018

Migration of MEM ACS to automated simlist generation flow

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

By

Amit Jangir
(17MECE07)

Under the guidance of

External Project Guide:

Vivek Agrawal

Principal Engineer

ARM Embedded Technologies Pvt. Ltd.,

Bengaluru.

Internal Project Guide:

Dr. Nagendra Gajjar

Prof. & PG Coordinator(Embedded Systems),

Institute of Technology,

Nirma University, Ahmedabad.



Electronics & Communication Engineering Department

Institute of Technology, Nirma University

Ahmedabad-382 481

December-2018

Declaration

This is to certify that

- a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.
- b. Due acknowledgment has been made in the text to all other material used.

- **Amit Jangir**
17MECE07

Disclaimer

“The content of this thesis does not represent the technology, opinions, beliefs, or positions of ARM Embedded Technologies Pvt. Ltd., its employees, vendors, customers, or associates.”



Certificate

This is to certify that the Major Project entitled “**Migration of MEM ACS to automated simlist generation flow**” submitted by **Amit Jangir (17MECE07)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date:

Place: Ahmedabad

Dr. N. P. Gajjar

Internal Guide

Dr. N. P. Gajjar

Program Coordinator

Dr. D. K. Kothari

Head, EC Department

Dr. Alka Mahajan

Director, ITNU

Certificate

This is to certify that the Major Project entitled “**Migration of MEM ACS to automated simlist generation flow**” submitted by **Amit Jangir (17MECE07)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance at **ARM Embedded Technologies Pvt. Ltd.** In my opinion, the submitted work has reached a level required for being accepted for examination.

Vivek Agrawal

Principal Engineer

ARM Embedded Technologies Pvt. Ltd.

Bengaluru.

Date:

Place: Bengaluru

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. Nagendra Gajjar**, PG Coordinator of M.Tech Embedded Systems for guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Dr. Nagendra Gajjar**, guide of my internship project for his exemplary guidance, monitoring and constant encouragement.

I would like to thank **Vivek Agrawal**, external guide of my internship project from **ARM Embedded Technologies Pvt. Ltd.**, for guidance, monitoring and encouragement regarding the project. I would also like to thank mentor **Pratik Bhattacharjee**, Staff Engineer for his encouragement and guidance.

I thank my Parents, faculty members and colleagues for their constant support and encouragement during this project work.

- **Amit Jangir**
17MECE07

Abstract

Verification and design analysis are major components of microprocessor design cycle time, any effort that improves verification effectiveness and design quality is crucial for meeting customer deadlines and requirements. It is well known to all IP creators and customers that function verification is a very big problem in semiconductor industry. As complexity of design increases, need of verification effort is more compare to design effort. For ARM CPU cores IP's which is a complex IP, it is difficult to detect desing errors and provide more validation coverage. Functional validation is one of the mostly known bottlenecks in System-on-Chip (SoC) design cycle. A mojority of engineering effort is spent on validating the SoC. According to Wilson Research Group, 57 percent time is spent of validation of a SoC project. Therefore optimization of validation flow is crucial for complex IPs such as ARM CPU Architecture. In this report a part of entire validation flow of ARM V8A architecture is optimized to reduce simulation time and complexity of system. In this work MEM suite of ACK kit is taken in to consideration for optimization, different MEM suites are migrated to a new validation flow that will directly link the simlist generation to the target configuration parameters.

Abbreviation Notation and Nomenclature

ARM	Advanced RISC Machines
IP	Intellectual Property
CPU	Central Processing Unit
ACK	Architecture Compliance Kit
ACS	Architecture Compliance Suite
SOC	System On Chip
MMU	Memory Management Unit
RISC	Reduced Instruction Set Computer
CISC	Complex Instruction Set Computer
ISA	Instruction Set Architecture
MEM	Memory
SIMD	Single Instruction Multiple Data
EL	Exception Level

List of Figures

1.1	Project Work-Flow	3
2.1	Project time spent in verification [3]	5
2.2	IP Blocks in a Typical System [3]	5
2.3	Validation Stages [3]	7
3.1	Development of ARM Architecture [5]	11
3.2	Exception Levels [5]	12
3.3	Compliance sign-off process	14
4.1	Types of Source list	17
4.2	Testdbv1 -TestlistGeneration Flow	17
4.3	Structure of a Suite	19
4.4	Structure of a target	19
4.5	Structure of Source Config Map	20
4.6	Testdbv2 - TestlistGeneration Flow	21
5.1	Testdb V1-V2 Comparison Tool	25
5.2	CSV Report	26
5.3	SimDiff Report	26
5.4	V1-V2 Mapping	27
5.5	Expanded Command Comparison tool	28
6.1	CPU time comparison of V2-V1	30

Contents

Declaration	iii
Disclaimer	iv
Certificate	v
Acknowledgements	vii
Abstract	viii
Abbreviation Notation and Nomenclature	ix
List of Figures	x
1 Introduction	1
1.1 Motivation	1
1.2 Objective	1
1.3 Requirements	2
1.4 Scope of Work	2
1.5 Gantt Charts	2
1.5.1 Project Work-Flow	2
1.6 Thesis Outline	3
2 Background Theory and Literature Survey	4
2.1 Background Theory and Literature Survey	4

2.2	Verification at ARM:	6
3	V8A Architecture and ACK	8
3.1	ARM V8A	8
3.1.1	Introduction	8
3.1.2	ARM ISA overview	9
3.1.3	ISAs in V8A	10
3.1.4	Exception levels	11
3.2	Architecture compliance Kit	12
3.2.1	Introduction	12
3.2.2	Compliance sign-off process	13
4	Automated Simlist Generation Flow(testdbV2)	16
4.1	TestdbV1	16
4.1.1	Types of source list	16
4.1.2	Testdbv1 -TestlistGeneration Flow	17
4.1.3	Disadvantages of testdbV1	18
4.2	Directory Structure	18
4.2.1	Structure of a Suite	18
4.2.2	Structure of a target/Implementation	18
4.2.3	Structure of Source Config Map	18
4.3	TestdbV2	20
4.3.1	Testdbv2 - TestlistGeneration Flow	20
5	Tools for testdbv2 migration process	23
5.1	Introduction	23
5.2	Tools	24
5.2.1	Testdb V1-V2 Comparison tool	24
5.2.2	Expanded Command Comparison tool	27

<i>CONTENTS</i>	xiii
6 Results and Conclusion	29
6.0.1 Results	29
6.0.2 Conclusion	30
References	32

Chapter 1

Introduction

1.1 Motivation

Verification and design analysis are major components of microprocessor design cycle time, any effort that improves verification effectiveness and design quality is crucial for meeting customer deadlines and requirements. It is well known to all IP creators and customers that function verification is a very big problem in semiconductor industry. As complexity of design increases, need of verification effort is more compare to design effort. For ARM CPU cores IP's which is a complex IP, it is difficult to detect design errors and provide more validation coverage. In this report a part of entire validation flow of ARM V8A architecture is optimized to reduce simulation time and complexity of system.

1.2 Objective

The main objectives of the project are as follows:

- To directly link the simlist with target configuration parameters
- To improve the performance of testlist/simlist generation
- To make a centralized place to define all the configs of a suite

- To improve the readability and reduce the complexity of testlist generation
- To achieve automated simlist generation flow

1.3 Requirements

The development and implementation of this project requires following:

- Good understanding of Linux environment.
- Knowledge of ACK flow.
- Understanding of ARM V8A architecture.
- Knowledge of Validation Tools.
- Shell scripting
- Knowledge of ARM Internal tools.

1.4 Scope of Work

This work mainly aims at migrating current MEM ACS of ARM V8A to a newly centralized place to get the automated simlist generation flow for finer level of control.

1.5 Gantt Charts

1.5.1 Project Work-Flow

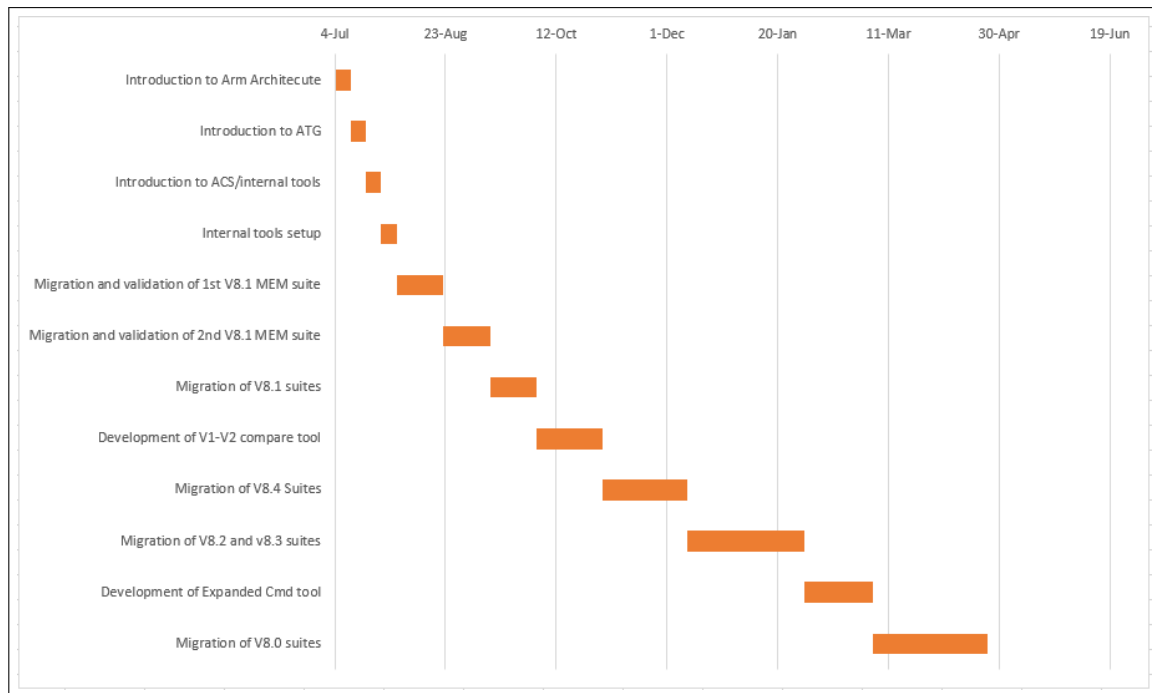


Figure 1.1: Project Work-Flow

1.6 Thesis Outline

- **Chapter-1** This chapter gives the brief information about motivation and objective of the project with the Gantt chart(time line) of project development work flow.
- **Chapter-2** This Chapter shows the literature survey and the overview of Functional Validation, Validation at ARM.
- **Chapter-3** discusses about ARMV8A Architecture and ACK.
- **Chapter-4** discusses the Testdbv1-testdbv2 Simlist generation flows.
- **Chapter-5** describes the tools developed and used in migration process.
- **Chapter-6** results and conclusion.

Chapter 2

Background Theory and Literature Survey

2.1 Background Theory and Literature Survey

The main obstructions in the system-on-chip (SoC) is the functional verification. An important part of the engineering time spent in certifying SoC goes in to verification. As per Wilson Research Group, validation spent more than 57 percent of a specific SoC development in 2015.

Despite this efforts, functional disasters are still Major risks for a first-time design from the beginning of Multiprocessor chips with odd design, The design complexity of SoCs has increased significantly. As can be seen in the diagram given below, the no of IPs The component in SoC is moving at solid rate.

SoCs have evolved into complex entities that integrate Many diverse units of intellectual property (IP) a Modern SoC may include many parts such as CPU, GPU, interconnect, memory controller, system MMU, interrupt controller etc. are IP Complex units of design that are personally verified. However, despite IP-level verification, it is not It is possible to detect all the errors - especially those who are Only activates when the IP interacts within a system.

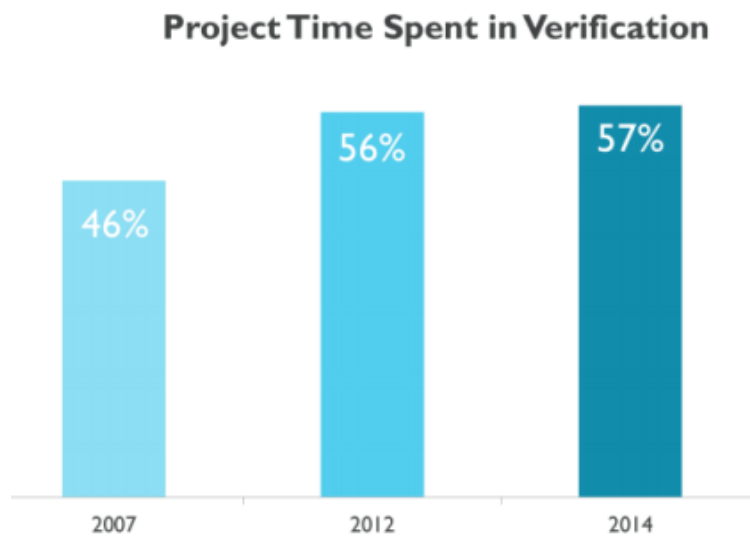


Figure 2.1: Project time spent in verification [3]

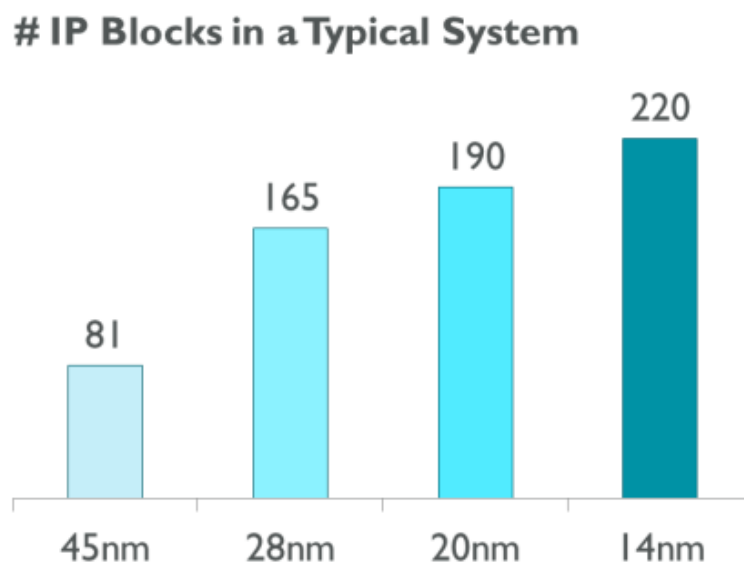


Figure 2.2: IP Blocks in a Typical System [3]

2.2 Verification at ARM:

Verification flow at ARM is comparatively Experienced in engineering. Verification of designs begins Initial and on the nuances of elements, which syndicate Create a stand-alone IP During full verification This process occurs at the unit level when engineers are the highest The amount of comprehension in the design. Separate signals Otherwise it may be deep inside the design Set to help investigate or verify the desired values. Once Unit level verification has touched the degree of maturity, Components are combined to create a full IP. Only IP-level verification of IP can be verified Start This is the first time the CPU is for the first time Testing of assembly program level may start. mostly Till this point the test is done separately. Wires / Signs Is written in tests at the IP level Assembly language. The processor raises the instructions From memory, decodes them etc. Once the IP-level verification fills some irritants Many IPs are combined in a system and system Verification attempt starts. During ARM's IP, many signals pass through Their design-verification cycle which they reproduce Functional fullness and accuracy Of these, Alpha and beta signals are internal quality Signal Limited access represents Milestone after which major partners get access to the IP. After that there is EAC, which indicates that point after which the IP is ready to be Designed to get engineering samples and tests The success of the release has gone through IP testing and is ready for mass production. In ARM, system verification of IP starts when they are Generally between alpha and beta quality. By this step Design cycle is still under IPs Significant quantities of test and lowest level errors Have already met. Excitement has to be done wisely Therefore it has been built that the internal state of the microarchitecture Each IP is stressful at extreme. Promotion is given either by assembly code or by Specially designed verification integrates using IP system. ARM uses a combination of both methods.

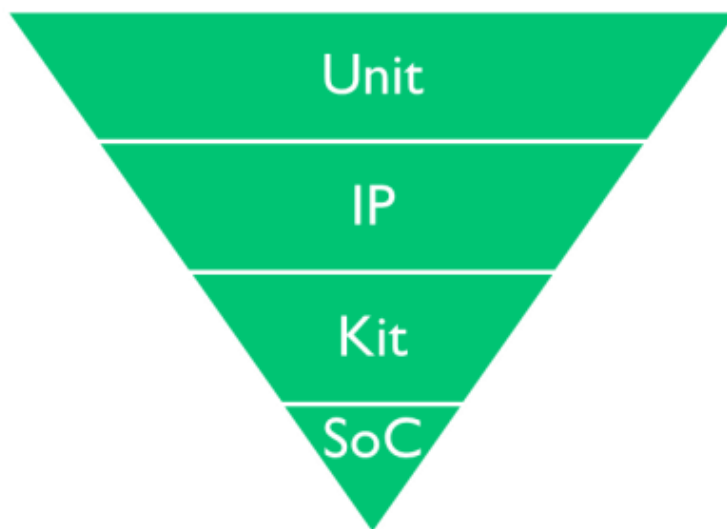


Figure 2.3: Validation Stages [3]

Chapter 3

V8A Architecture and ACK

3.1 ARM V8A

3.1.1 Introduction

ARM architecture is a Reduced instruction set computer (RISC) architecture, which originally stood for "Acorn RISC Machine" but stands for "Advanced RISC Machine" now. In the past years, ARM processors, with the spread of smartphones and tablets, are becoming very popular: mostly due to low cost, and more power efficiency than other architectures in the form of CISC.

ARM is a 64/32-bit RISC processor architecture which is currently being developed by ARM Corporation. The business model behind ARM is based on ARM architecture which wants to build ARM-based CPU or system-on-a-chip products. There are two main types of license implementation licenses and architecture licenses. The implementation license provides the complete information needed to design and manufacture an integrated circuit with the ARM processor core. ARM gives two types of core licenses: soft core and hard core. A hard core is optimized for a specific construction process, while a soft core can be used in any process, but less adaptation is done. Architectural license enables the licensee to develop with ARM ISA in line with its own processor. The ARM processor has a unique

combination of features that makes ARM the most popular embedded architecture today. First of all, the ARM core is much simpler than most other general-purpose processors, which means that they can be produced using a relatively small number of transistors, to which application-specific fi macro cells There is a lot of space left on the chip. An ARM chip can have some quantity of on-chip memory with many peripheral controllers, a digital signal processor and ARM core.

Second, both ARM ISA and Pipeline design aims to reduce energy consumption - an important requirement in mobile embedded systems. Third, ARM architecture is highly modular: the only compulsory component of the ARM processor is the integer pipeline; All other components, including Cash, MMU, Point Oting Point and other co-processors, are optional, which allows a lot of build in the creation of an application-speed ARM CARM-based processor.

3.1.2 ARM ISA overview

ARM is an RISC architecture. Like all RISC architectures, ARM ISA is a load-store one, that is, instructions which process data only work on registers and are different from the instructions that reach memory. All ARM instructions are 32-bit long and most of them have regular three-pronged encoding. Finally, ARM architecture has a large register with the 16general-purpose register. All the above features facilitate pipelining of ARM architecture. Whatever the design decisions were made to simplify the desire to keep the architecture and to make its implementation as simple as possible, it got distracted by the original RISC architecture. First of all, the original Berkeley RISC design used Register Win-Dos for speed invocation. ARM designers rejected this feat which would increase the processor size and complexity. In retrospect, it appears to be an intelligent decision, because register Windows does not prove a particularly useful feature and is not used in the majority of modern RISC processors.

Second, the execution phase of any instruction is required to complete the clas-

sical RISC approach in a cycle. It is necessary to bring an E3 client3-phase-decode-executed pipeline. While most ARM data processing instructions are completed in a cycle, data transfer instructions are an important exception. To complete a simple store or load instruction in one cycle, two memory access performance must be restarted in the same cycle: one - to bring the next instruction from memory, and second - to perform real data trans-show . Performing two memory access in one cycle, in its turn, requires a Harvard architecture with different instructions and data memories, which were considered very expensive by the designers of the ARM rst ARM processor. However, to achieve better utilization of pipelines during 2-cycle instruction execution, they started an auto-indexing addressing mode, where the value of an index register is increased or decreased, while a load or store is in progress. While all modern ARM implementations have different instructions and data caches and can complete memory transfers in one cycle, they still support auto-indexing mode which can improve the performance of ARM programs and code size Proved to be.

Third, ARM supports multiple-register-transfer instructions, which allows to load or store 16 registers at a time. In violation of a cycle of instruction theory, they sign-up, speed up performance-critical tasks such as process operation and bulk data transfers, and lead to more compact codes. In summary, ARM architecture omits all bene-ts of O FI RISC approach, such as pipeline-friendliness and simplicity, deviating from it in some aspects, which makes it even more attractive for embedded system developers.

3.1.3 ISAs in V8A

- AArch64 ARMv8-A 64-bit execution state, which uses 31 64-bit common-purpose registers (R0-R30), and 64-bit program counter (PC), stack pointers (SP) and Exception Link Register (ELR) . The SIMD vector and scalar provides 32 128-bit registers for floating-point support (V0-V31).

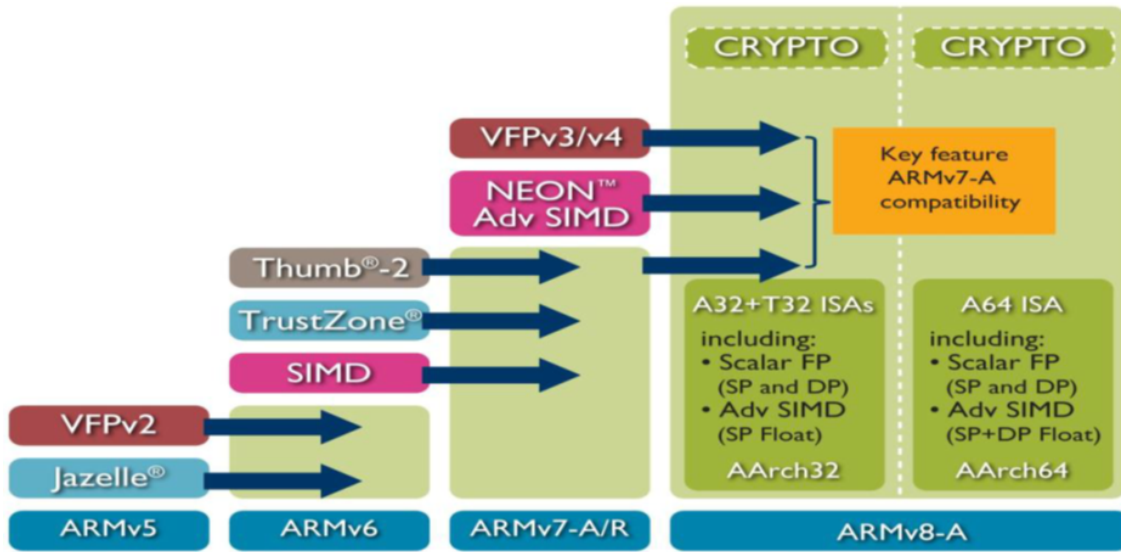


Figure 3.1: Development of ARM Architecture [5]

- The A64 instructions have a fixed length of 32 bits and are always small-endian. AArch32 ARMv8-A is a 32-bit execution state, which uses 13 32-bit general purpose registers (R0-R12), 32-bit program counters (PCs), stack pointer (SP), and link register (LR). Advanced SIMD vector and scalar provides 32 64-bit registers for floating-point support.
- The AArch32 execution state offers two instruction sets, A32 (ARM) and T32 (Thumb2) option. Operation in AArch32 state is compatible with ARMv7-A operation. T32: 16-bit instructions transparently disintegrate transparently to meet 32-bit ARM instructions in real time without performance loss. Thumb-2 technique set the thumb to a mixed (32- and 16-bit) length.

3.1.4 Exception levels

There are four exception levels, which replace 8 different processor modes, they work as a ring in Intel architecture, they are a form of privilege hierarchy:

- EL0 is the least privileged level, in fact it is called an unplugged executive. The apps are run here.

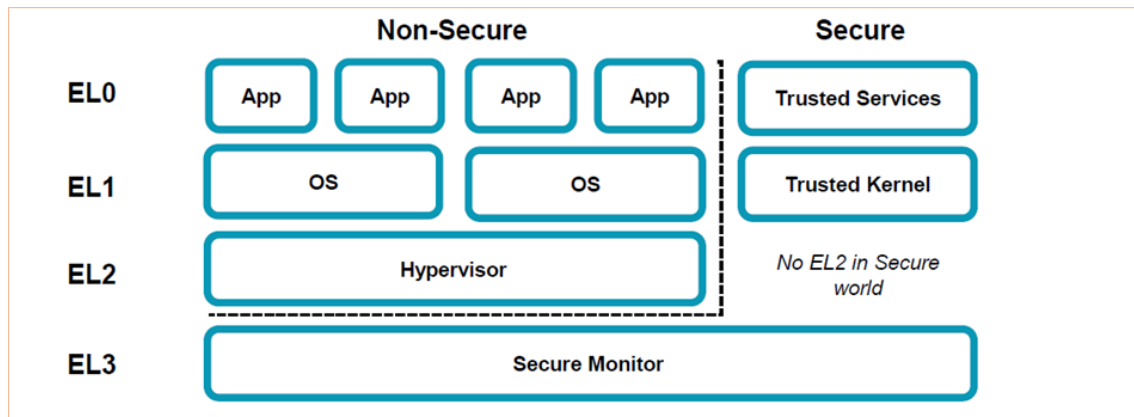


Figure 3.2: Exception Levels [5]

- EL1: OS kernel can be run here
- EL2: Provides support for virtualization of non-secure operations. Hypervisor can run here.
- The EL3 provides support for switch between two security states, secure state and non-secure state. A safe monitor can be run here.

When executing in AArch64 state, execution can occur only between exception levels, only with an exception or returning from exception. There are 3 private banked registers in each of the 4 privilege levels: Exception Link Register, Stack Pointer and Saved PSR.

3.2 Architecture compliance Kit

3.2.1 Introduction

ARMv8 ACK ARMv8 architecture enables licensees to verify that their implementation is Corresponding to architecture defined by ARM. All this can be achieved by running the application Tests which are provided in the ACK and report that all these tests have passed.

V8A AVK includes tests for the following ARM architecture definitions:

- Instruction set including instructions set and system level architecture (PE)
- External debug
- Generic timer, external memory mapped components of ARMv8 architecture
Interface to the Performance Monitors and External System Control Registers
- Generic Interrupt Controller Architecture
- Embedded Trace Macrocell Architecture
- RAS Architecture Extension.
- Statistical Profiling Extension.

While running the tests, implementation should include all the components applicable to them. Architecture allows implementation options, including non-inclusion of some of these Organ The kit can be configured to test the features and configurations implemented. The verification kit is standalone. Tests from AVS may be out-of-box at AEM supplied. The test equipment running on the VL library and AEM are included in the kit.

3.2.2 Compliance sign-off process

ARMv8 avk enables licensees to check for compliance with ARMv8 architecture Passing AVS. This verification should be done before release for each ARMv8 architectural device Production That is, licensees can run tests at any stage of development, but can not continue Production equipment in which AVS results are not cleaned or pardoned. Licensees should confirm that all Test runs in ARMv8 AVS with a clean pass. The following figure shows an overview of AVK compliance sign-off process.

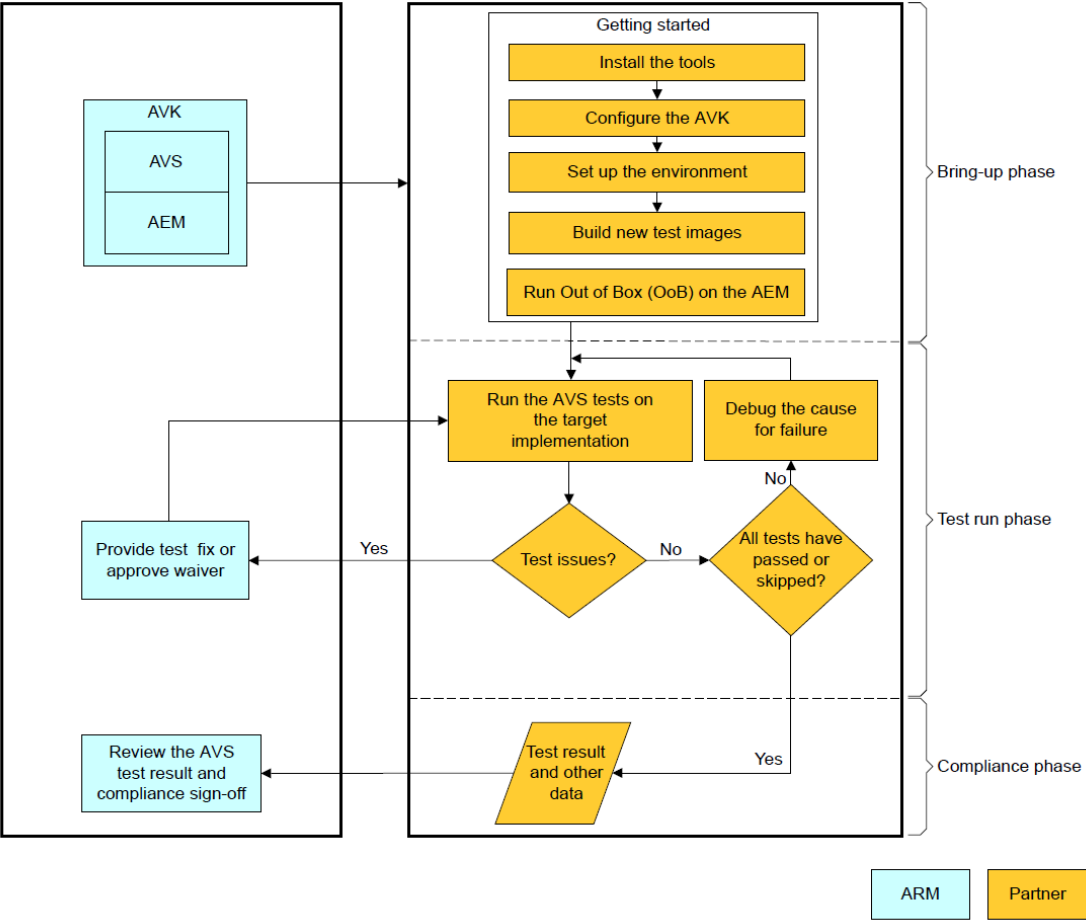


Figure 3.3: Compliance sign-off process

Any test that has been dropped has completely or partially failed, or should be replaced with the release version By ARM If tests are failing due to test issue, but ARMv8 architecture is not due to non-compliance After device, ARM can provide rebates or supply updated tests.

Chapter 4

Automated Simlist Generation

Flow(testdbV2)

This chapter describes the complete flow of simlist generation ,its components and directory structure of ACK kit.

4.1 TestdbV1

ACK contains various validations suites for ARM architecture and its extensions. Each suite is a collection of many test cases. A test can be run with multiple configurations which is called as a Sim or simulation. Test will generate multiple sims and that is decided by all the source list present in that suite.source list contains the list of test cases which will be going to run for that configs. All possible ways (with various configs) a test runs is called a sim or simulation. So basically simlist generation in testdbv1 depends on source list and config crosses defined in testdb file.

4.1.1 Types of source list

There are two ways to define the source list .

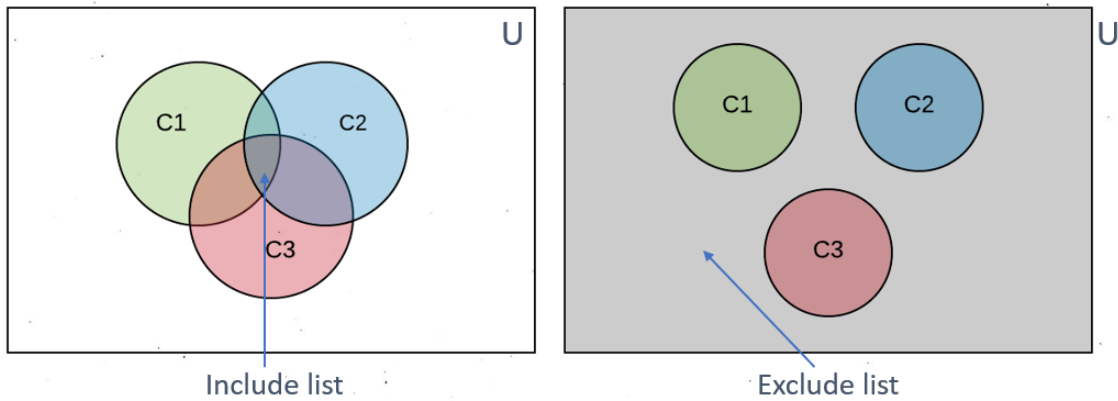


Figure 4.1: Types of Source list

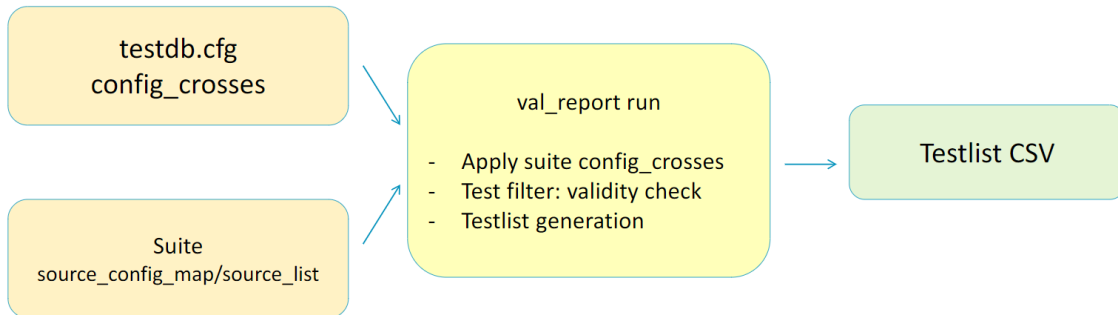


Figure 4.2: Testdbv1 -TestlistGeneration Flow

- Include list (C1.C2.C3): Example: Tests present in C1 list, will run with config C1.
- Exclude list (C1.C2.C3): Example: Tests present in C1 list, they will not run with config C1 rest all will run.

4.1.2 Testdbv1 -TestlistGeneration Flow

Testlist generation in testdbv1 dependent on testdb.cfg config crosses and source config map . The block diagram of flow is shown in fig 4.2.

4.1.3 Disadvantages of testdbV1

There are many disadvantages of testdbv1 listed below.

- The current version of MEM ACS simlist is not directly linked with target configuration parameters
- It is required to define different configs for a specific target.
- For each target a separate testdb file is to be maintained.
- There are multiple config files for a suite, so it is difficult to keep track of each config.

4.2 Directory Structure

4.2.1 Structure of a Suite

A suite contains various files including test cases , config files, source lists. the structure is shown in fig 4.3.

4.2.2 Structure of a target/Implementation

Each target contains many file like testdb, targetconfig etc. The structure of Target is shown in fig 4.4.

4.2.3 Structure of Source Config Map

Source config map is a new file which is present in the testdbv2 format of simlist generation flow. This file contains the all test cases of a suite, corresponding Boolean equation and 2nd level filter. The structure of this file is shown in fig 4.5.

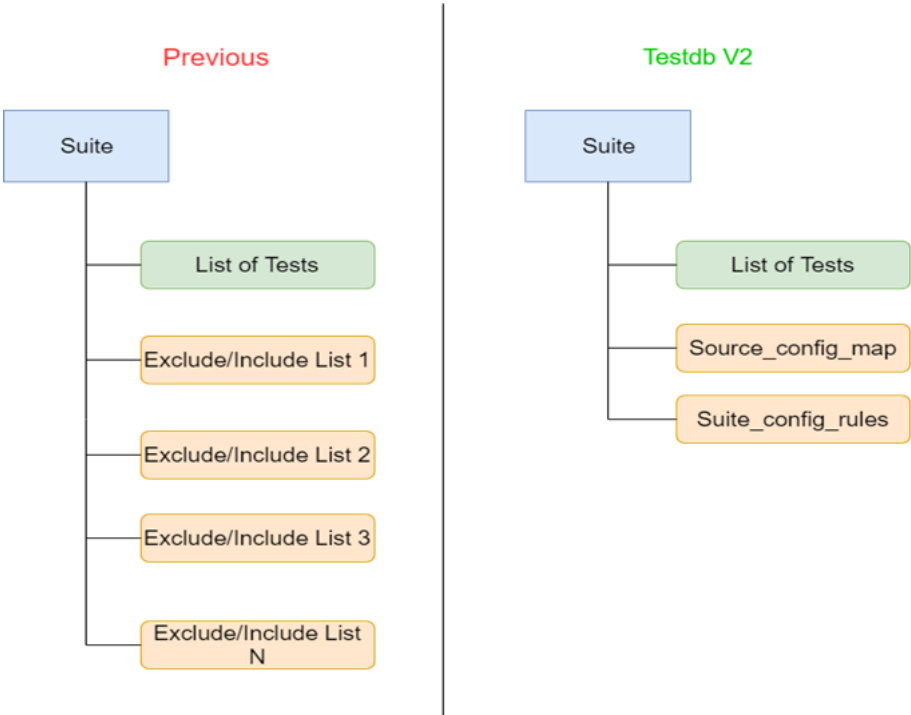


Figure 4.3: Structure of a Suite

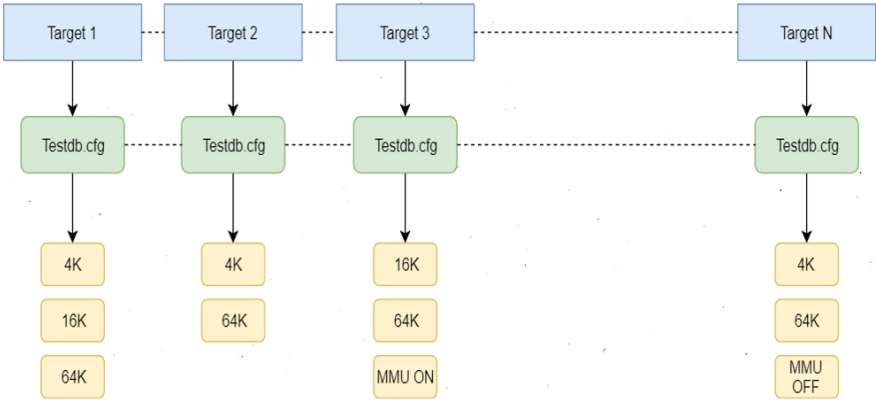


Figure 4.4: Structure of a target

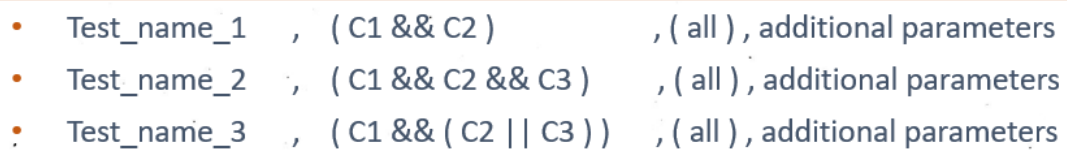
- 
- Test_name_1 , (C1 && C2) , (all) , additional parameters
 - Test_name_2 , (C1 && C2 && C3) , (all) , additional parameters
 - Test_name_3 , (C1 && (C2 || C3)) , (all) , additional parameters

Figure 4.5: Structure of Source Config Map

4.3 TestdbV2

TestdbV2 is the enhance version of testdbv1 format and which overcomes all the limitation which testdbv1 had. key points of testdbv2 is list below

- To directly link the simlist with target configuration parameters
- To improve the performance of testlist/simlist generation
- To make a centralized place to define all the configs of a suite
- To improve the readability and reduce the complexity of testlist generation
- To achieve automated simlist generation flow

4.3.1 Testdbv2 - TestlistGeneration Flow

In testdbv2 the automated simlist is generated based on source config map, suite config rule file , global config rules and feature config rules . The source config map file contains the list of test cases with their corresponding boolean equation to generate the sims. The suite config rule file contains the list of intermediate configs which are define in this file and used in SCM file. There are two global file which very important in automated sim generation. Global config rules and feature config rules . both the files are common for all the suite. The configs are defined in these

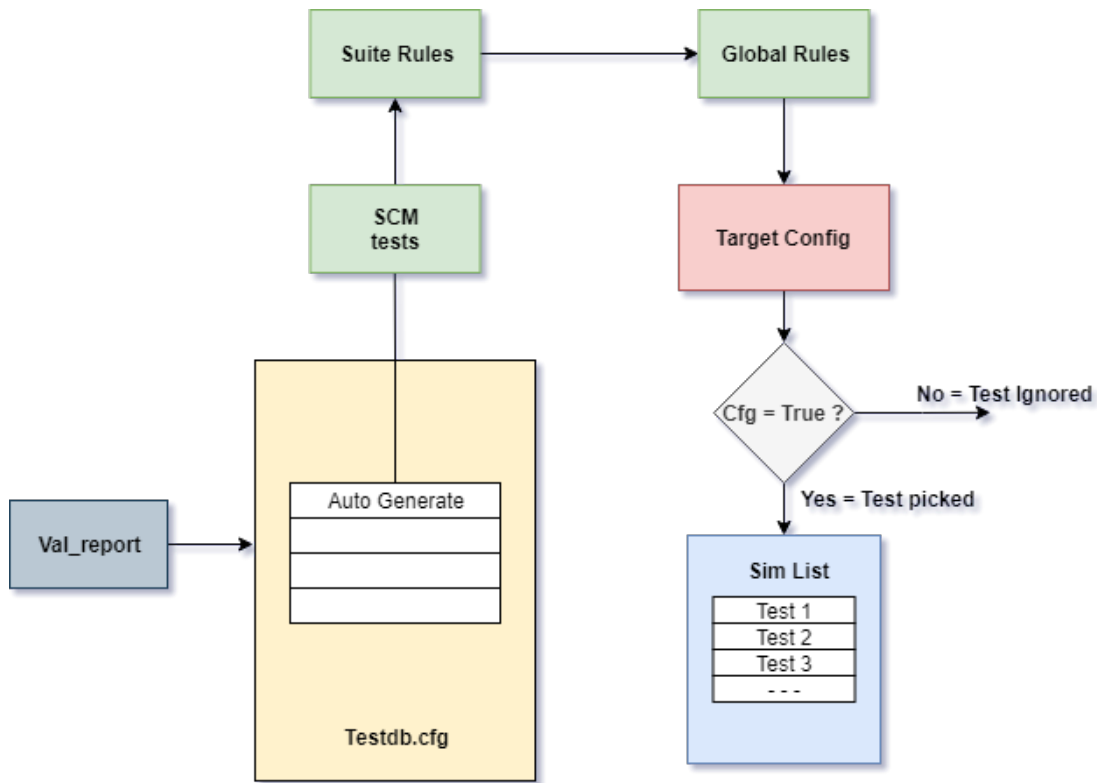


Figure 4.6: Testdbv2 - TestlistGeneration Flow

file with the boolean equation . if a config's value become true for particular target then that suite will run with that config for that target.

The block diagram for testdbV2 simlist genration flow is given fig 4.6.

TestdbV2 includes some new feature over the testdbv1 which are shown below.

- Fully automatic testlistgeneration without need for manually specified testdb cfg crosses
- Evaluation of suite, test and config validity as per target features
- Bottom-up simlist computation: valid config crosses computation for tests from source config map without suite level crosses
- In-memory computation no intermediate file generation

- C-style syntax for ease of editing/formatting with multi-line expression support
- Functional style and operator style syntax for ease of reading (with syntax highlighting)

Chapter 5

Tools for testdbv2 migration process

This chapter describes about the tools developed and used in entire migration process. Their working, inputs and outputs.

5.1 Introduction

Migration of a suite is a multi-step process which includes generating SCM and SCR files, Modifying SCM according to the requirement, generating simlist for single target with V2 flow and comparing it with simlist of V1. After that a comparison is required between V1 and V2.

An single suite can have hundreds of test cases and thousand's simulations. Manual comparison of thousand's simulations between V1 and V2 is not feasible. There is a strong requirement of a tool which can compare all simulations automatically and show any difference as an output.

For example lets assume that a suite generates 5000 simulations when it runs on a single target, So in this case we need to do 5000 comparison between V1 and V2. Now lets assume that the same suite is now running on 70 targets, So in this case we need to do $5000 * 70 = 3,50,000$, this amount of manual comparison is not

feasible to do hence there is a requirement of tools to be developed for the same purpose.

5.2 Tools

For this migration work two major tools are developed for the comparison purpose.

- Testdb V1-V2 Comparison tool
- Expanded Command Comparison tool

5.2.1 Testdb V1-V2 Comparison tool

This tool is mainly used for sim name comparison and to get the information about extra/missing sims. This will take the simlist of V1 and V2 as an input and in output it provides the different reports which will further gives the information about sim name diff or extra/missing sims.

Sometimes after migrating a suite we might have reordered configs due to this we may get diffence in sim name, this difference in intentionally done by the suite owner to maintain the readability of SCM. This kind of difference is automatically handled by this tool, tool will automatically ignores this difference.

Before running the tool we have to generate the simlist in V1 and V2 format on each and every target. After that generated simlists will go as an input to the tool and then it will various reports. A working block diagram of this tool is given in fig 5.1.

Steps to use the tool is given below.

- Run the qual.pl with Testdbv1 on all targets
- Run the qual.pl with Testdbv2 on all targets
- Generate Simlist on each target in V1 and V2

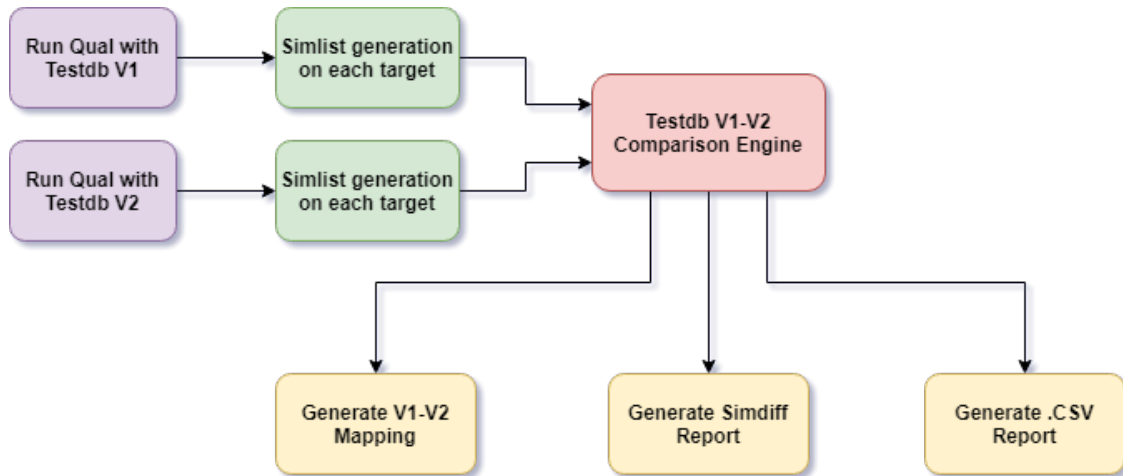


Figure 5.1: Testdb V1-V2 Comparison Tool

- give the path of v1/v2 working directory where all the targets are present and in each target corresponding sim list is present
- call testdb V1-V2 comparison engine
- analyze all the reports .csv , simdiff, V1-V2 mapping

.CSV Report

CSV report is one of the output of this tool. This is a excel sheet file which contain the total number of simulation of each target in both the cases V1 and V2. By analyzing this report one can figure-out that which targets are clean or have no difference. Example version of CSV report is shown in fig 5.2.

SimDiff Report

SimDiff is one of the output of this tool. This is .txt file which contains the extra/missing sim name for each target. In CSV report only sim count is given not the exact sim name which are differing in V1 and V2. This information can be get by this report.

Example version of SimDiff Report is shown in fig 5.3.

	A	B	C	D	E
1	Target_Name	V1	V2	Sim Diff	
2	target_01	77	98	21	
3	target_02	77	98	21	
4	target_03	77	98	21	
5	target_04	66	66	0	
6	target_05	8	8	0	
7	...	50	44	-6	
8	...	0	8	8	
9					
10					

Figure 5.2: CSV Report

```

682 =====
683 Target name: target_01
684 =====
685 No of Sims in V1 : 55                No of Sims in V2 : 56
686 -----
687 test_name : Test_01                  + extra_config: 4k.cfg1.cfg2
688 test_name : test_01                  + extra_config: 16k.cfg1.cfg2
689 test_name : test_02                  + extra_config: 64k.cfg1.cfg2
690 -----
691 test_name : test_03                  - missing_config: 4k.cfg4.cfg5
692 test_name : test_04                  - missing_config: 16k.cfg4.cfg5
693
694 =====
695 Target name: target_02]
696 =====
697 No of Sims in V1 : 55                No of Sims in V2 : 56
698 -----
699 test_name : Test_01                  + extra_config: 4k.cfg1.cfg2
700 test_name : test_01                  + extra_config: 16k.cfg1.cfg2
701 test_name : test_02                  + extra_config: 64k.cfg1.cfg2
702 -----
703 test_name : test_03                  - missing_config: 4k.cfg4.cfg5
704 test_name : test_04                  - missing_config: 16k.cfg4.cfg5

```

Figure 5.3: SimDiff Report

```

1 Test Name :          V1_ELF_NAME          V2_ELF_NAME
2 -----
3 Test_01             cfg1.4k.cfg2      ===== 4k.cfg1.cfg2
4 Test_02             cfg2.cfg3.cfg1    ===== cfg1.cfg2.cfg3
5 Test_03             cfg1.cfg2.64k     ===== 64k.cfg1.cfg2
6 Test_04             cfg3.16k.cfg5     ===== 16k.cfg13.cfg5
7
8

```

Figure 5.4: V1-V2 Mapping

V1-V2 Mapping

V1-V2 Mapping is one of the output of this tool. This is a .txt file which contains the sim name mapping of V1 and V2 if there is any reordering is done. This report helps to determine the sim name changes between V1 and V2.

Example version of V1-V2 Mapping is shown in fig 5.4.

5.2.2 Expanded Command Comparison tool

In process of porting a suite to testdbv2 flow, one have to deal with many config/sconfig files or direct defines. Some config files requires modification to make them suitable for testdbv2, some defines needs to be pass directly in SCM file. Due to all the above requirements we have to verify that Expanded command for each test must be matching with Expanded command in testdbv1.

Expanded Command Comparison tool will do the comparison for each simulation and any mismatching will be dumped to the output report file.

A working block diagram of this tool is given in fig 5.5.

Steps to use this tool is given below.

- Run all the tests of a suite on all targets in testdbv1
- Run all the tests of a suite on all targets in testdbv2
- Generate .log files of test simulation for both V1 and V2 format
- .log file will contain Expanded commands

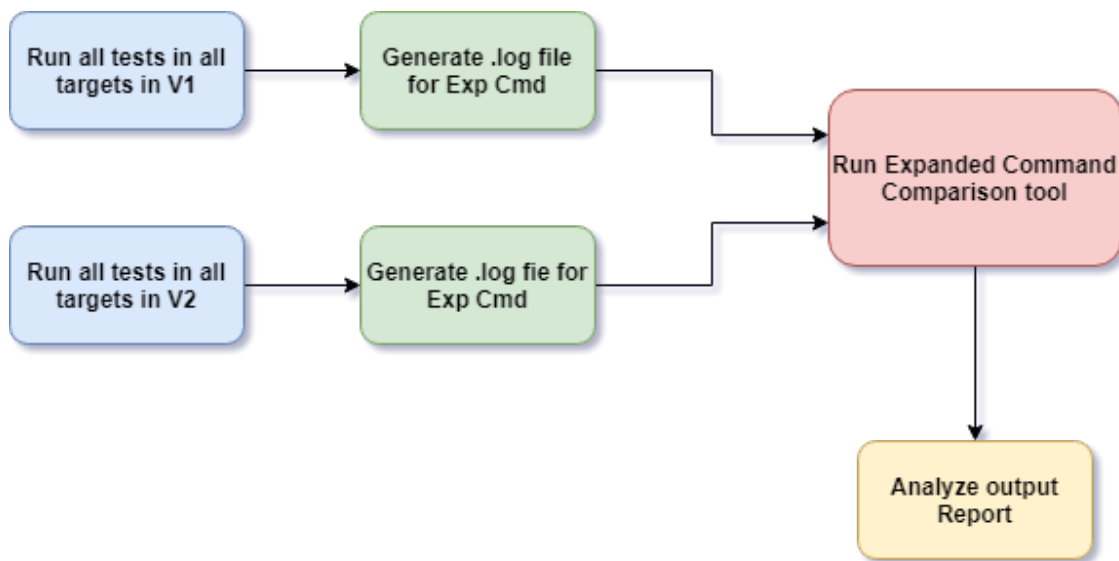


Figure 5.5: Expanded Command Comparison tool

- run the Expanded Command comparison tool
- Analyze the Expanded Command report for any differences

Chapter 6

Results and Conclusion

This chapter provides the details about obtained results and conclusion of this thesis.

6.0.1 Results

As mentioned earlier the main objective of this project is to directly link simlist to target configuration parameter and to improve the performance of testlist/simlist generation flow. After porting the suites to testdbv2, faster simlist generation is achieved.

To improve the performance of testlist generation, one must remove the redundancy or unwanted simulation. While porting suites to testdbV2, it is found that many unwanted tests were picking, due to this CPU taking longer time run all tests.

As we have linked the target parameter directly to simlist, these unwanted simulations were ignored and total CPU run time was significantly reduced.

For example one of the suite was taking 5.85 hours CPU time on a specific target when testdbv1 flow was used. Here total 775 simulations were running and 428 simlutions were skipping out total 775. These 428 simulations were increasing the total RTL Run time. When the same suite is ported to Testdbv2, only 347

*****			*****		
TOTAL TESTS	:	347	TOTAL TESTS	:	775
TOTAL PASSED	:	333 (96.0%)	TOTAL PASSED	:	333 (96.0%)
TOTAL FAILED	:	0	TOTAL FAILED	:	0
TOTAL ABANDONED	:	14 (4.0%)	TOTAL ABANDONED	:	14 (4.0%)
TOTAL SKIPPED	:	0	TOTAL SKIPPED	:	428 (55.2%)
TOTAL NOT RUN	:	0	TOTAL NOT RUN	:	0
TOTAL RES CHK FAIL	:	0	TOTAL RES CHK FAIL	:	0
TOTAL ELF GEN SKIP	:	0	TOTAL ELF GEN SKIP	:	0
TOTAL COMPILED OK	:	0	TOTAL COMPILED OK	:	0
TOTAL COMPILE FAIL	:	0	TOTAL COMPILE FAIL	:	0
TOTAL USR ANALYSIS	:	0	TOTAL USR ANALYSIS	:	0
TOTAL OUT-OF-DATE	:	0	TOTAL OUT-OF-DATE	:	0
TOTAL SIM TIME	:	0.93 hours	TOTAL SIM TIME	:	1.18 hours
TOTAL CPU TIME	:	3.07 hours	TOTAL CPU TIME	:	5.85 hours
TOTAL REAL TIME	:	4.14 hours	TOTAL REAL TIME	:	7.79 hours
TOTAL CYCLES	:	0k, 0 cps	TOTAL CYCLES	:	0k, 0 cps
BATCH REAL TIME	:	0.00	BATCH REAL TIME	:	0.00
PARALLELISM	:	0.00	PARALLELISM	:	0.00
*****			*****		

Figure 6.1: CPU time comparison of V2-V1

simulations were picking and all redundant/unwanted simulations were removed by testdbv2.

In the figure it can be seen that total CPU time is reduced by approx. 2.5 hours when testdbv2 flow is used.

In some targets such as ARM internal targets sim count in testdbv2 is increase as compared to testdbv1. This increment is positive increment as it increases the coverage.

6.0.2 Conclusion

In this project, Migration of 18 MEM suites successfully done and efficient automated simlist generation is achieved . In this thesis, Migration of MES ACS to automated simlist generation was carried out.

The experimental analysis of testdbv2 gives the improvemt in simlist generation time and memory uses.

In this report a part of entire validation flow of ARM V8A architecture is optimized to reduce simulation time and complexity of system.In this work MEM suite of ACK kit is taken in to consideration for optimization, different MEM suites are migrated to a new validation flow that will directly link the simlist generation

to the target configuration parameters.

References

- [1] Ho, Richard C., et al. “Architecture validation for processors.” ACM SIGARCH Computer Architecture News 23.2 (1995): 404-413
- [2] Fournier, Laurent, Anatoly Koyfman, and Moshe Levinger, “Developing an architecture validation suite: application to the PowerPC architecture” Proceedings of the 36th annual ACM/IEEE Design Automation Conference. ACM, 1999.
- [3] System Validation https://www.arm.com/files/pdf/System_Validation_at_ARM_Enabling_our_partners_to_build_better_systems.pdf
- [4] ARM Architecture Reference Manual https://static.docs.arm.com/ddi0487/da/DDI0487D_a_armv8_arm.pdf
- [5] ARM V8 Architecture https://www.arm.com/files/downloads/ARMv8_Architecture.pdf