TOP LEVEL CPU VERIFICATION

Submitted By

Unnikrishnan Patel

16MCEC27

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2018

TOP LEVEL CPU VERIFICATION

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Unnikrishnan Patel

(16MCEC27)

Guided By

Dr. Sanjay Garg

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2018

Certificate

This is to certify that the major project entitled **"TOP LEVEL CPU VERIFICATION"** submitted by **Unnikrishnan Patel (Roll No: 16MCEC27)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering (Specialization in title case, if applicable) 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-I, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. Sanjay Garg
Guide & Professor,
CE Department,
Institute of Technology,
Nirma University, Ahmedabad.

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

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

Dr Alka Mahajan
Director,
Institute of Technology,
Nirma University, Ahmedabad

Statement of Originality

I, **Unnikrishnan Patel**, **16MCEC27**, give undertaking that the Major Project entitled "**TOP LEVEL CPU VERIFICATION**"" submitted by me, towards the partial fulfillment 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

Dr. Sanjay Garg

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Dr. Sanjay Garg**, Head and Professor, Computer Engineering 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. His 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 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. Alka Mahajan**, 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.

- Unnikrishnan Patel

16MCEC27

Abstract

In application specific integrated circuit design , verification is a vital part in the processor development. Verification ensures that the product is designed is the same as it was intended, by applying the test signals to the design and check whether its matches with the output or not. Unfortunately, many design projects do not complete thorough design qualification resulting in products that does not meet customer expectations and require costly design modifications. More the complex is design , the verification of design is also more complex and verification time is also more.

Random stimulus generation or Random Instruction Sequence (RIS) is widely recognized as an effective approach for verifying corner cases that are hard to anticipate. While most of design bugs are flushed out by the deterministic approach, RIS tools are highly effective in hitting obscure cases. The work presented here uses the RIS tool to solve the problem. The tool uses template library that contains test-cases. A test file contains registers and memory values. Random test generator never generates a test which is invalid. The work presented here can be subdivided into two parts, one is to generate template (test-case) for specific conditions like instruction optimization, early forward and cryptography. These templates are developed to increase the hit-rate of corner cases to meet required targets, which are required for verifying arm cores. It helps to improve the randomization of RIS tools. These templates hit different conditions of early forward with various instruction sets like Arch32 and Arch64, and also with different subsets of instructions like branch, logical, arithmetic and load-store. The second part is to analyze the cache and branch behaviour with some random test suits. It is very tedious and time consuming task to find out behaviour for different types of templates. So, automation is required. These automation scripts comes together to form a regression framework, which handles everything from test preparation to test submission to clusters for execution. Framework is generic which can work with almost every cores developed by arm.

Abbreviations

ARMARM	ARM Architecture Reference Manual.
AVS	Architecture Validation Suites.
RAVEN	Random Architecture Verification Engine.
RFM	Regression Framework Manager
VIP	Virtual Intellectual Property
ISA	Instruction Set Architecture
UTB	Unified Test Bench
VAL	Validation Abstraction
DUT	Design Under Test
DVS	Design Validation Suites
ELs	Exception Levels
PE	Processing Element

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Random instruction stream (RIS) devices are generally utilized for arm processor check and approval. RIS instruments are principally used to discover bugs in RTL(outlines created utilizing equipment depiction dialects) plan. RIS apparatuses create circumstances which are difficult to ponder. RIS apparatuses are exceptionally helpful for hitting corner cases that are extremely hard to chronicle utilizing coordinated testing. On the off chance that bugs incorporates some particular succession of instructions in limit time hole than RIS devices which produce random arrangement of instructions are exceptionally powerful. Macros are valuable for giving controlled randomness to the test which is helpful for focusing on a particular territory in the processor engineering.

1.2 Objective

RIS is broadly perceived as a successful approach for confirming corner cases that are difficult to suspect. RIS is likewise very compelling in hitting dark cases. RIS apparatuses utilizes random test libraries to cover the greater part of corner cases. Changes in randomization of RIS devices enable us to builds the hit to rate of various corner cases. Current RIS instruments are great at focusing on pipelines for uniprocessor center and memory pecking orders in multiprocessor center. Since multiprocessor frameworks includes numerous corner instances of coherency, stops and dangers which are not all that simple to target physically.

1.3 Problem Statement

As part of CPU top level verification team (CPG), understanding the various aspects of Processor verification at the top level. This would include understanding the fundamentals of modern processor design, specifically understanding ARM architecture based designs done within ARM. Understand the system verilog based environment used for ARM CPU verification and related scripts. Develop the Regression Framework Manager for handling the regressions and smokes done weekly to verify the developing processors.

1.4 Thesis Organization

The report is planned into multiple chapters explaining the arm v8 architecture, random instruction stream tool and its working. Later the thesis explains the structure of Regression Framework Manager which exploits the RIS tools, concepts of machine learning to improve regressions which in turn improves the cluster's usage, failure managements and also reduce the users involvement with regression management(So that they can work on other issues).

CHAPTERS :-

- Engineering Specifications of ARM refrence CPU
- Porting from DVS to RIS
- RIS Tool
- Regression Management Framework
- Optimization of Random Test Constraints Using Machine learning Techniques
- Conclusion and Future Scope

Chapter 2

Engineering Specifications of ARM reference CPU

The core on which i am currently working on(Helios) is a power-efficient, mid-range, throughput computing oriented, simultaneously multithreaded (SMT) core that implements the Aarch64 execution state of the ARMv8-A architecture. It supports the v8.2 extension, the RAS extension, and the RCpc Load acquire (LDAPR) instructions introduced in the ARMv8.3 extension, and the Dot Product instructions introduced in the ARMv8.4 extension. it does not implement the AArch32 execution state of the ARMv8-A architecture. As an SMT core, Helios supports two execution threads on each core. Each thread is a separate architectural Processing Element (PE), and so has a complete copy of the architectural state. The core is used with cluster level logic to create a processor cluster. It is aimed at the networking data-plane market, ADAS, storage, image processing and HPC markets among others. While it is not expected to be used in a typical big.LITTLE configuration as seen in mobile devices, it will be used in heterogeneous systems and supports heterogeneous cluster configurations, and so has full ISA compatibility with Prometheus and Ares (other than AArch32 support), and aligns on key microarchitectural features visible to software.

2.1 Architecture and Configurable Options

This section details the features supported by Helios and the configurable options.

2.1.1 Supported Architectural options

- Implements the ARMv8-A architecture
  - AArch64 only no support for AArch32.
  - All exception levels, EL0-EL3 are supported
  - Advanced SIMD/Floating point support is optional.
  - Including optional support for the cryptographic instructions.
  - All translation granule sizes (4KB, 16KB, 64KB) are supported.
- Including the ARMv8.1 extensions
  - Hardware updates of page table access bits are supported.
  - 4 Limited Ordering Regions are supported.
- Including the ARMv8.2 extensions
  - The CnP page table bit is supported.
  - The 16-bit floating-point instructions are supported.
  - Statistical profiling is not supported.
- The VA size is 48-bit per page table.
- The PA size is 44-bit.
- GICv4 is implemented (and is backward-compatible with GICv3).
  - Memory-mapped accessed to the GIC registers is not supported.
- ETMv4.2 is implemented.
  - Only instruction trace is implemented, data trace is not supported.
  - System register access to the ETM is not supported.

2.1.2 Configurable Options

The Helios core implements the following build time configuration options:

- Optional Advanced-SIMD Floating-Point extensions (unlike Apollo, these are not separately licensable).
- Optional and separately licensable cryptographic extensions.
- Configurable L1 instruction cache size 32KB-64KB.
- Configurable L1 data cache size 32KB-64KB
- Optional per-core L2 unified cache with size range: 64KB-256KB
- Optional ECC/parity cache protection on core RAMs.
- Cores can be configured to either interface synchronously to the cluster, or to have an asynchronous bridge allowing independent clocking.The asynchronous bridges can be configured to use 2-flop synchronizers or 3-flop synchronizers.
- Optional support for integrating the ELA-500 Embedded Logic Analyser.

2.1.3 Core Configuration

Cores must have identical configurations, with the exception of the L2 cache size. That is, all cores have AdvancedSIMD/FP, or do not, have cryptography support, or do not, and all cores have the same size L1 caches as each other.
The core is designed so that it can be implemented once and then instantiated multiple times. Indeed, any multicore implementation is likely to use a hierarchical implementation approach.

2.2 Microarchitecture Overview

2.2.1 Multithreading

Helios supports two execution threads on each core. Each thread is a separate architectural Processing Element (PE), and so has a complete copy of the architectural state. Thus software sees the threads on a Helios core in the same way as multiple cores in a cluster on existing single-threaded processors, and so existing multicore software should

be able to run unmodified on Helios. This threading model matches that used on multithreaded processors from other architectures, and so the existing OS software support for multithreaded processors should require minimal porting to run on a Helios system

2.2.2 Cluster

Helios is delivered as a processor cluster. The cluster consists of between one and eight Helios cores, plus cluster level logic (referred to as Corinth) that contains the snoop control unit (SCU) and L3 cache, as well as clock, reset, power management, and other miscellaneous features. shows two example clusters that contain Helios.

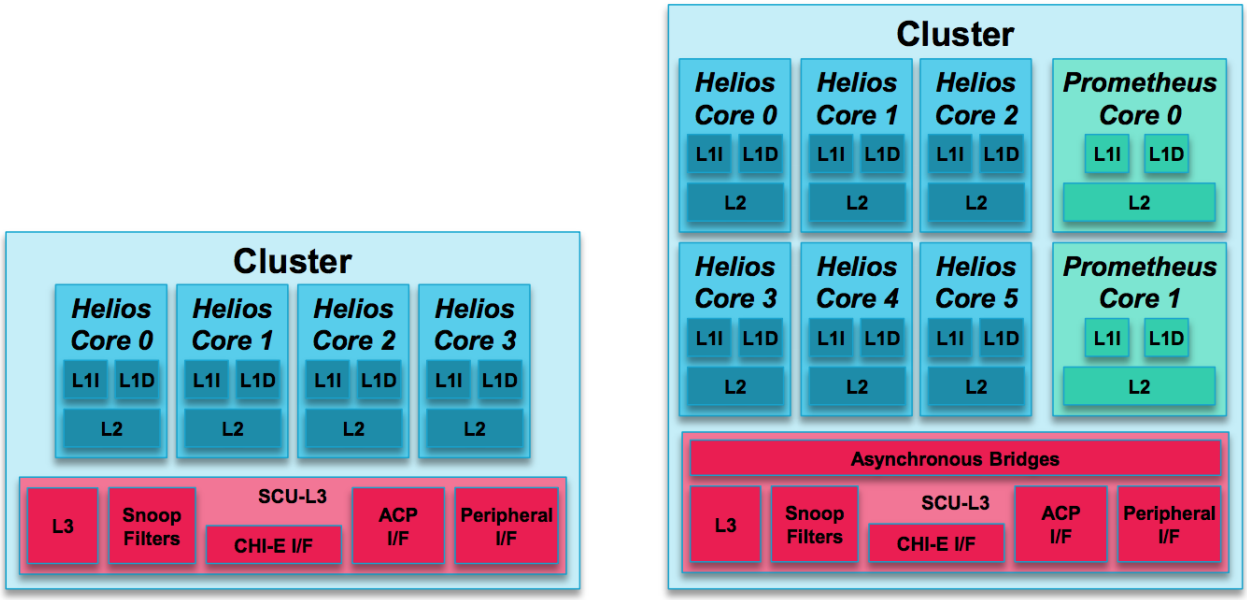

Figure 2.1: Example clusters, quad-core Helios and octa-core big.LITTLE (2 big and 6 LITTLE cores)

In addition to just Helios cores, if another compatible product such as Prometheus is also licenced then the two processors can be combined in the same cluster connected to a single Corinth SCU-L3, giving in-cluster big.LITTLE. When compared to the previous generation of big.LITTLE processors, this provides:

- Improved performance when sharing data between big and LITTLE cores
- More efficient use of area by sharing a large cache between more cores
- Simplified integration of the big and LITTLE cores together

- More opportunities for end product differentiation with different combinations of cores

The interface between the cores and Corinth supports an asynchronous interface which allows:

- Simplified implementation and timing closure across the cluster
- Different frequency, power, and area implementation points for different cores. In particular the big cores can run at a higher maximum frequency than the LITTLE Helios cores.
- Each core to be run at a different frequency and (optionally) at a different voltage, giving per-core DVFS capabilities.
- Support for running the SCU-L3 logic at a synchronous 1:1 or integer multiple of the system interconnect frequency, thus eliminating the need in many systems for an asynchronous bridge between the cluster and interconnect.
- Simplified system integration for other interfaces such as debug and trace which are already in the correct clock domain at the output of the cluster.

A key feature to support this change in topology is the introduction of private, per-core L2 caches. The latency of the L2 cache is important for processor performance, and placing an asynchronous bridge between L1 and L2 would impact this performance too much. Additionally, by making the L2 cache private, the latency can be reduced further, improving performance over previous generations of processors.

2.3 Micro-Architecture

2.3.1 Introduction

A Helios cluster consists of between one and eight cores coherently connected by a combined cluster L3 cache which includes a Snoop Control Unit (SCU). The cluster integrates CoreSight components such as ETM. Within a Helios core there are nine units:

- IFU (Instruction Fetch Unit)
- DPU (Data Processing Unit)

- DCU (Data Cache Unit)
- STB (Store Buffer)
- TLB (Translation Lookaside Buffer)
- BIU (Bus Interface Unit)
- ETM (Embedded Trace Macrocell)
- GIC (Generic Interrupt Controller CPU interface)
- L2 (Level 2 cache controller)
- CBU (CPU Bridge Unit between L2 and L3, and core level power management)

Within the cluster level logic:

- SCU (Snoop Control Unit, including L3 cache controller)
- SBU (SCU Bridge Unit, containing cluster level power management)

Outside the cluster:

- DebugBlock (containing debug-over-powerdown support, and other debug components such as CTI)

The constituent units and blocks of a dual-core Helios cluster are shown in Figure. Some of these blocks are optional, for example the L2 cache.

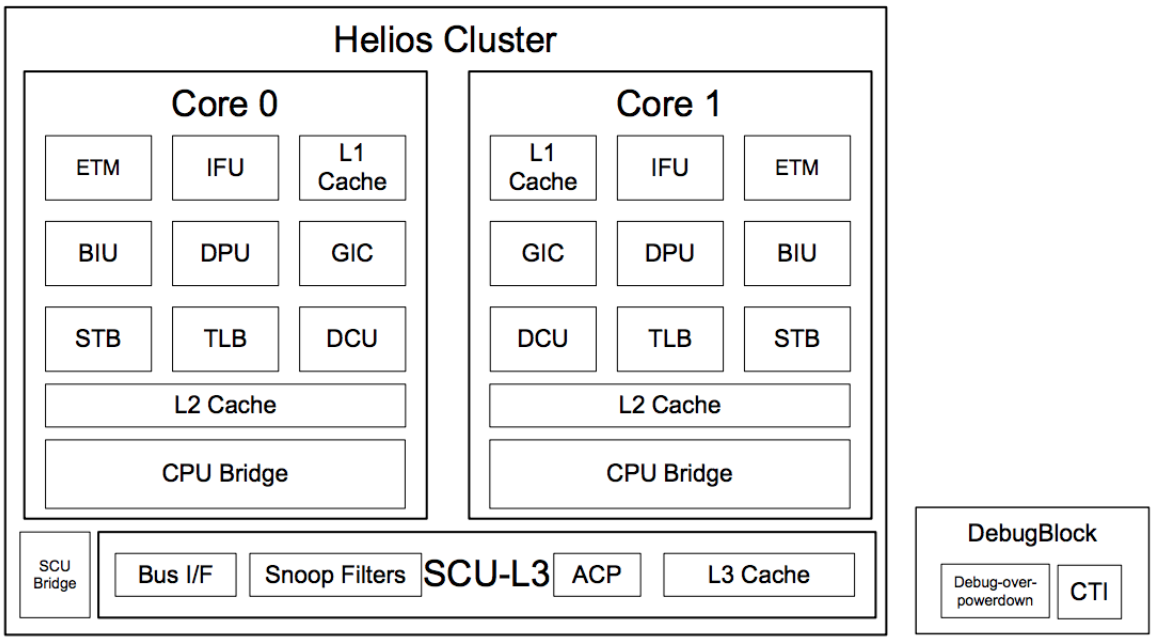

Figure 2.2: Helios Cluster (dual-core)

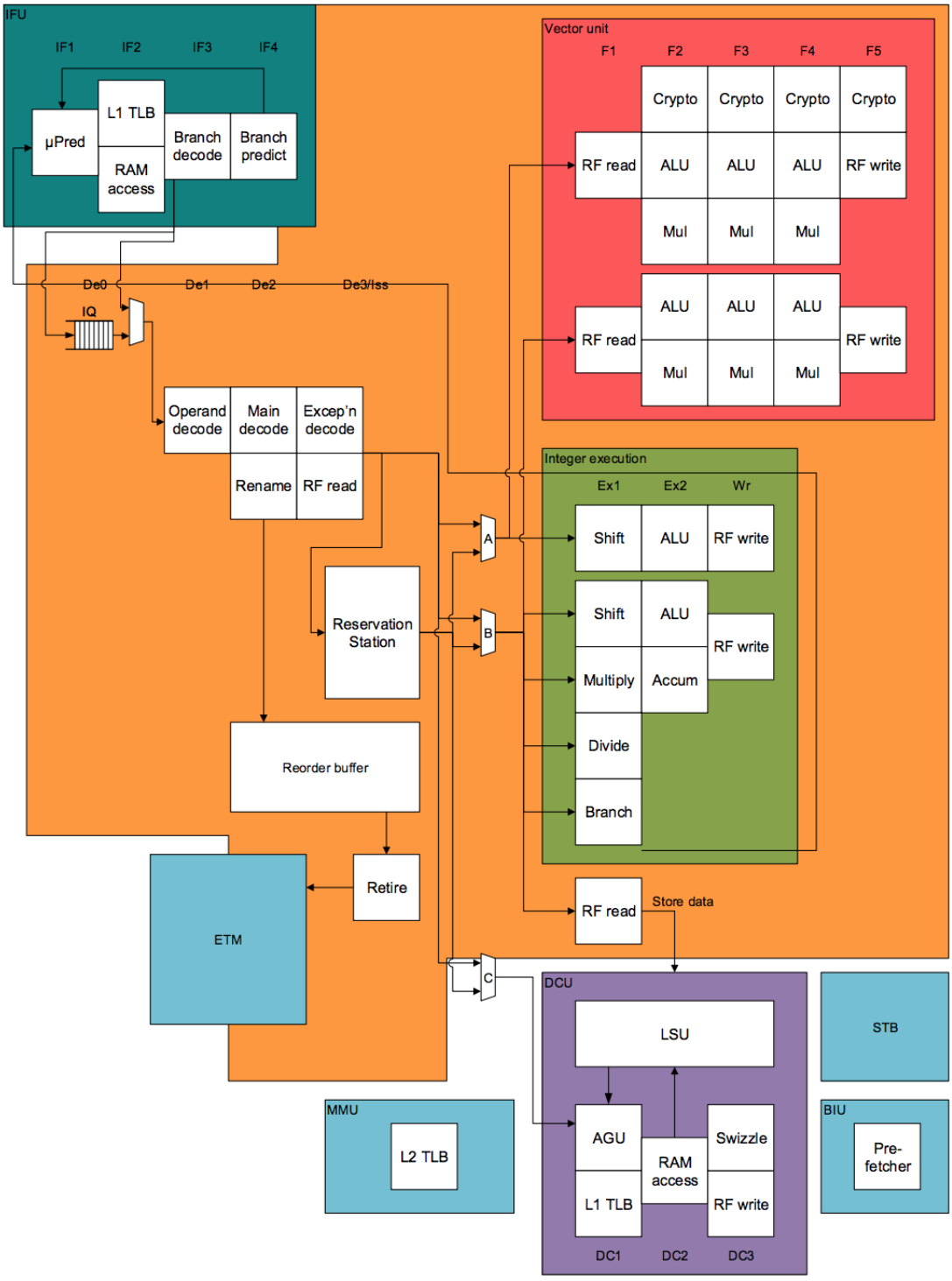

Figure 2.3: Pipeline Overview

Chapter 3

Porting from DVS to RIS

The chapter discusses about different types of tests that are used in verification process and trade-offs between them. This chapter includes advantages of porting of tests from DVS to RIS.

3.1 Verification

A critical and most important aspect of any VIP (Verification Intellectual Property) is the test suite. To verify design optimally, selection of appropriate type of tests are important. The verifier must ensure that tests cover all important areas of design that needs to be verified.[?]

Tests are fall into mainly three categories:

- Architecture Validation Suites (AVS)
- Device Validation Suites (DVS)
- Random Instruction Stream (RIS)

All Architecture Validation Suites check engineering usefulness, for example, the ISA (32-bit and 16-bit Thumb), the troubleshoot display, and the exemption design. Gadget Validation Suites center around the conduct of particular centers and confirm corner cases emerging from the specific execution. The scope accomplished by this immediate grouping is high, however not finish. In this way, we need to take after other approach to fill scope gaps. Irregular boost age or Random Instruction Sequence (RIS) is broadly perceived as a powerful approach for checking corner cases that are difficult to envision.

While the vast majority of configuration bugs are flushed out by the deterministic approach, RIS are likewise exceedingly viable in hitting dark cases.

3.1.1 Architecture Validation Suites

AVS refers to Architecture Validation Suites. These suites are comprised of tests that are intended to cover the architecture features for ARM. In general, AVS tests are self-checking; an AVS test provides both the stimulus and the expected results within the test itself. Any mismatch between the behaviour of the RTL and the behaviour expected by the test will produce a failure message.[?]

3.1.2 Device Validation Suites

DVS refers to Device Validation Suites. These suites are comprised of tests that are intended to cover the features which are difficult to cover using RIS either due to limitation of RIS tool ability to generate the stimulus or due to limitation in checking.[?]

3.2 Porting Tests from DVS to RIS

DVS tests are very simple test, where particular scenario is implemented for a known feature, whereas RIS tests cover more numbers of scenarios with different configuration. It is possible because of scenario is generated with random constraints with restricting parameters to values. There is always trade-off between direct and random test. With more directed tests, it is possible that we miss out some important permutations whereas with more constraint random tests, the complexity is greatly increases. Directed tests are used to verify whether feature have been implemented correctly or not. For each unique feature there is one or more dedicated directed test. It helps to getting clear picture of various functionalities. Directed tests are convenient during starting stage of development, where each feature should be tested separately to verify correctness of feature. Random test are important to ensure that whether design behaviour is follows design specification or not.

Chapter 4

Random Instruction Stream Tool

RIS test generator is for approval of Multi-preparing or Cluster frameworks through RTL simulations,and focusing on high guideline age rate 1000IPS.

ARMs front line MP RIS check mechanical assembly focusing on memory sub-structure operations and cross-PE coherency trades in Multi-Processor/Cluster systems.Its a server-class static RIS generator that finishes high rule age rates (more essential that 100 IPS) and proposed to allow deduced test groupings to quickly fulfill their pined for point, while moreover allowing most outrageous re-use of made circumstances for snappier over-age conclusion. It offers full help of ARMv8-An AArch64 execution state. The AArch32 A32 (ARM) ISA is to some degree maintained, and no assistance is open for AArch32 T32 (Thumb) ISA. Rule groupings can be described to target specific operations and little scale configuration features. The going with are additional irregular state features centered by ARM's new MP RIS Tool:

- Multiprocessor memory consistency.
- Constraints
- Cache and TLB stressing
- Message passing (Exclusive operations, Load Acquire or Store Release, Atomics).
- Load-store dependencies and hazards.
- Stress use of load-store pipeline and evictions.

- Used in Server class consistency verification.
- 1-32 processing elements (Cores), multiple cache hierarchies, sharing domains and interfaces.
- Efficient run time stimulus reaching test intent in minimum iterations.
- Max reuse of generated instances through code re-execution.

4.1 Random Instruction Sequence (RIS) Generation

RIS apparatuses are generally utilized over the business for processor confirmation and underwriting. These instruments are as frequently as conceivable used to discover configuration bugs in generally reliable however not yet make RTL plan. RIS devices are exceptionally persuading in making test conditions that are difficult to imagine. In any case, routinely completely erratic run courses of action are of little test a force for uncovering corner cases in the plan, particularly if the bug fuses a social event of occasions occurring in a tight organizing window. Macros can help update the test thought of the made govern movements by giving controlled affirmation around a particular game-plan of direction

4.2 Random Vs Deterministic stimulus generation

RIS-GEN is by and large seen as a capable approach for affirming corner cases that are hard to imagine. We found that, while an expansive bit of design bugs are used out by the deterministic approach, sporadic rule progressions are furthermore exceedingly capable in hitting dim cases, much of the time finding bugs that may lay undetected for an impressive period of time, everything considered, applications. ARM has an inside mechanical assembly that can deliver concentrated on self-assertive code groupings known as RIS. With RIS, we pre-make self-checking tests using an ISS as the reference diagram. This framework won't get design botches that are accessible in both the ISS and the HDL show, yet for all intents and purposes this condition is phenomenal and these groupings are most likely going to demonstrate diagram bumbles in either exhibit when enough progressions have been copied. The column method that we have used since the start of the key ARM CPU setup is deterministic amusement. This is a common and without a doubt knew framework that offers different purposes of enthusiasm, regardless

of the way that it's limited by the measure of effort required to deliver test cases and the execution of reenactment instruments. At ARM, we make test cases as self-checking developing specialist progressions. We by then replay these code groupings on an essential reenactment testbench containing the ARM CPU, a fundamental memory model, and some direct memory-mapped peripherals. Our tests fall into two classes, AVS (Architecture Validation Suites) and DVS (Device Validation Suites). ARM's all AVS class tests check building value, for instance, the rule set outline (32-bit and 16-bit Thumb), the exceptional case appear, and the investigate outline. Our DVS tests focus on the direct of specific focuses and check corner cases rising up out of the particular use. Inclination of this sort of test is that tests are autonomous and adaptable from ISS (Instruction Set Simulator) conditions to Verilog or VHDL test situate circumstances, or to FPGA models and at last to silicon. Along these lines, our customers and we can check the utilitarian proportionality of all these diagram sees. These suites of tests are effectively the ARM building consistence suites.

Chapter 5

Regression Management Framework

5.1 Introduction

Regression management framework ensure all possible hardware builds are regressed.It ensure all possible configurations and simulation options for a particular hardware build are regressed. It remove manual errors eg. Not passing correct simulation options/configuration for the particular hardware build, etc. It is able to run the complete regression with a single regression command.

5.1.1 Challenges

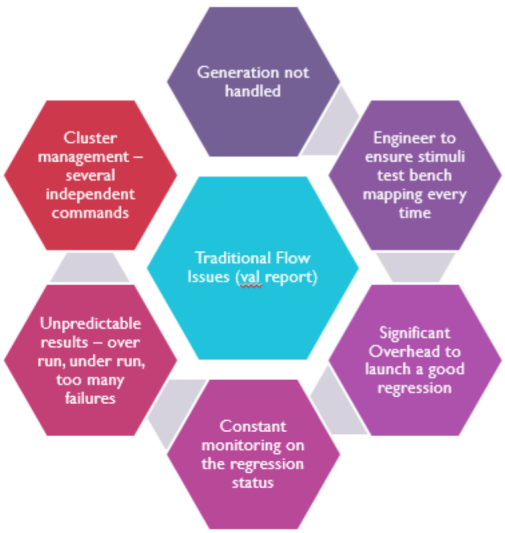

Figure 5.1: Traditional Flow Issues

5.1.2 Objectives

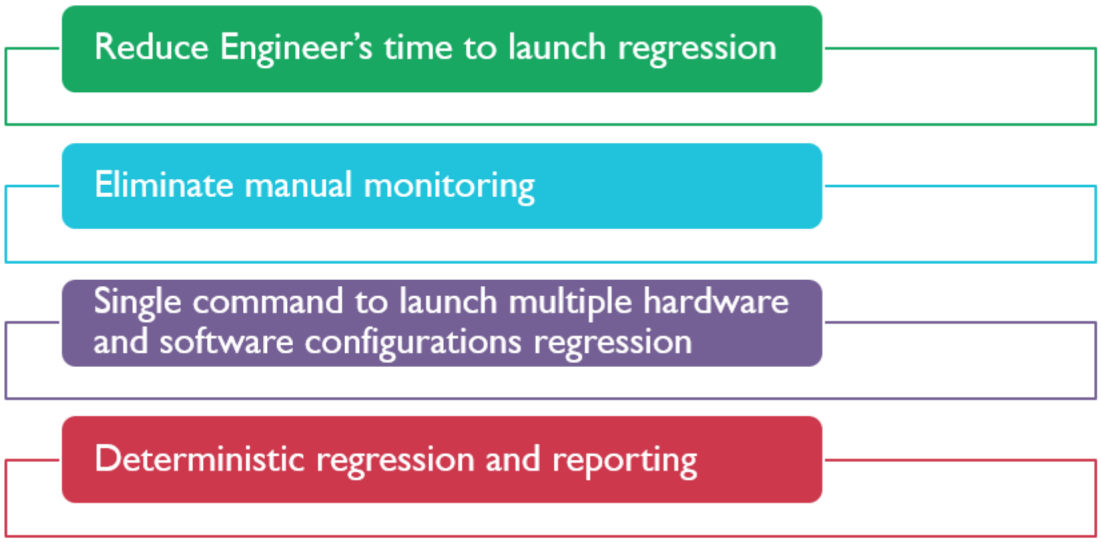

Figure 5.2: Objectives

5.2 Why it is needed?

- To ensure all possible hardware builds are regressed.
- To ensure all possible configs and simopts for a particular hardware build are regressed.
- To ensure all TB features are exercised efficiently for a feature.
- To remove manual errors eg. Not passing correct simulation options/configurations for the particular hardware build, etc.
- To be able to run the complete regression with a single regression command.

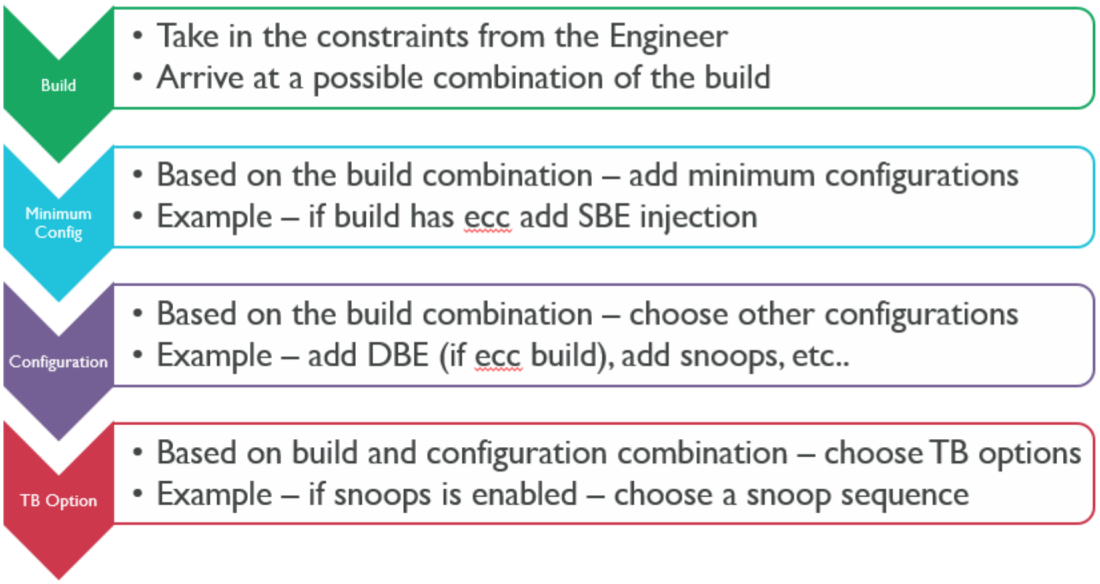

Figure 5.3: Effectively Resolving the Random Space

5.3 Framework Flow

Figure 5.4: Framework flow

5.3.1 Cluster Usage

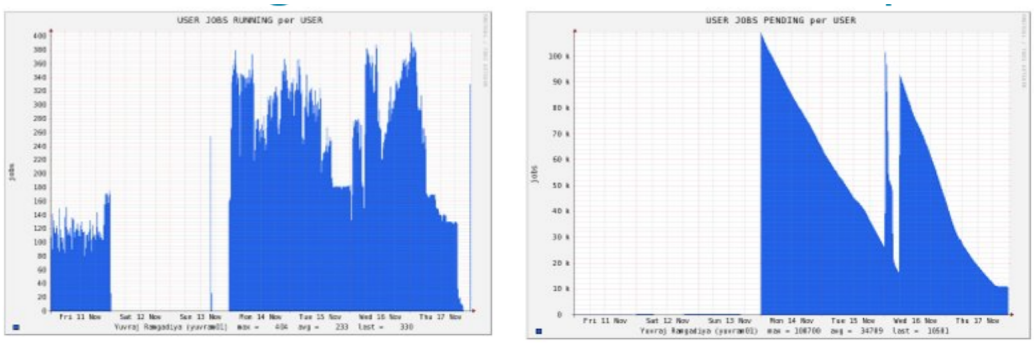

Figure 5.5: Cluster usage before the Flow was Adapted

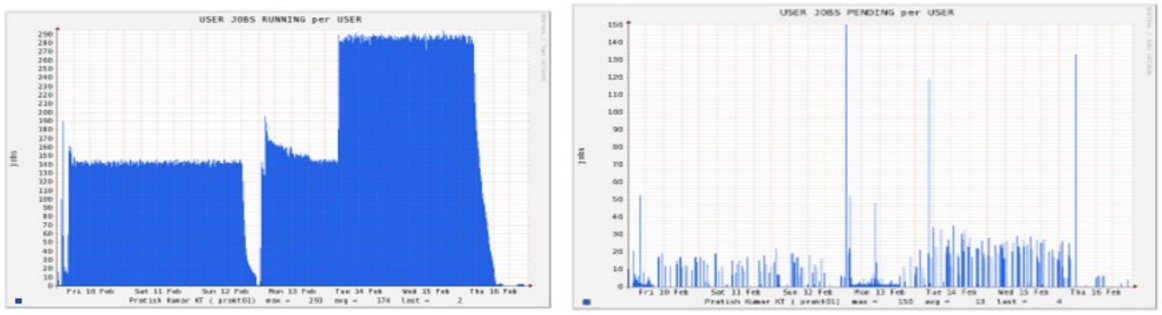

Figure 5.6: Cluster Usage After the Flow was Adapted

5.3.2 Flow In Detail

Usually regressions are very costly to manage as it takes a lot of human overhead and computation too. Idea behind regression management framework is to eliminate such factor. Since engineer can spend a lot more time on writing better tests than managing the regressions.

Regressions management framework takes few parameters(area, feature, cores, tool, project) as an input. Based on the inputs it will generate the rtls and fetch the respective test configurations files. After fetching the test configuration files, it sorts out tests based on builds, then it passes the files to the tools(ris tools) to generate .s files which are further converted to lower level target form called .elfs which are compatible to be executed on testbenches. Regression management framework has different modes for test generation:

- Batch mode
- Single mode

Once the tests are generated they are passed on cluster for execution. Regression management framework monitors the clusters as well as the framework's health and calculates certain parameters from the result of regressions and health reports. Then those parameters are given as a feedback to the framework to train its regression model, which inturns improves the consequent regressions.

5.3.3 Flow Chart

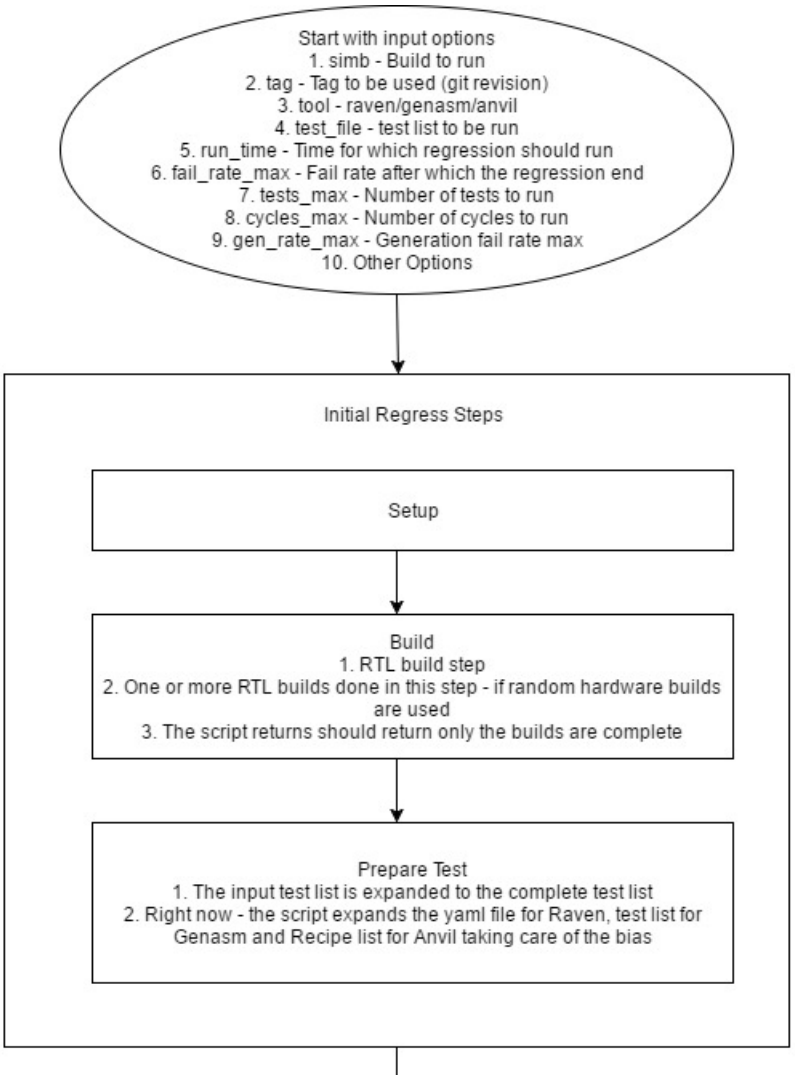

Figure 5.7: Flow Chart part1

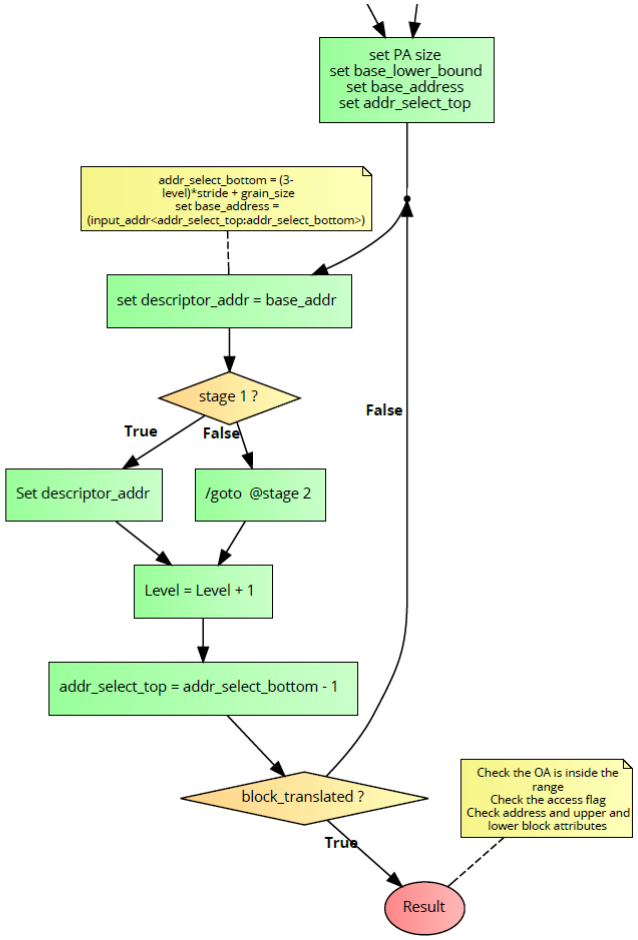

Figure 5.8: Flow Chart part2

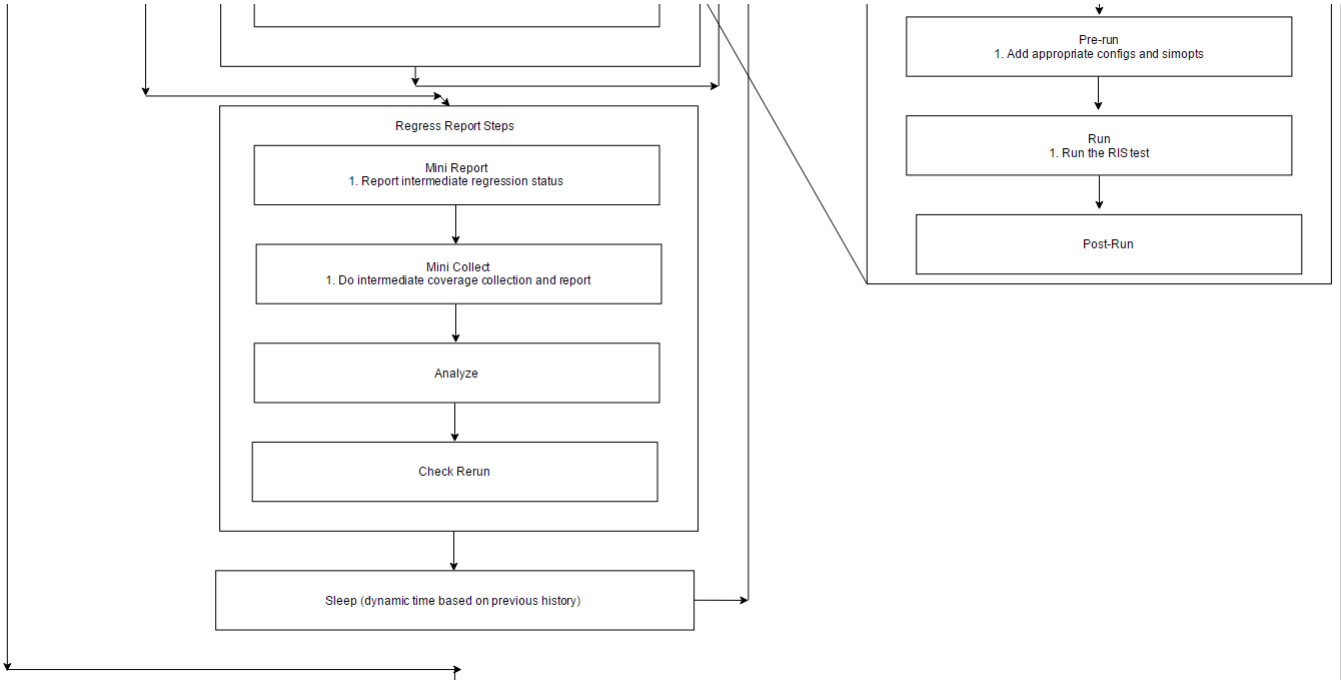

Figure 5.9: Flow Chart part3

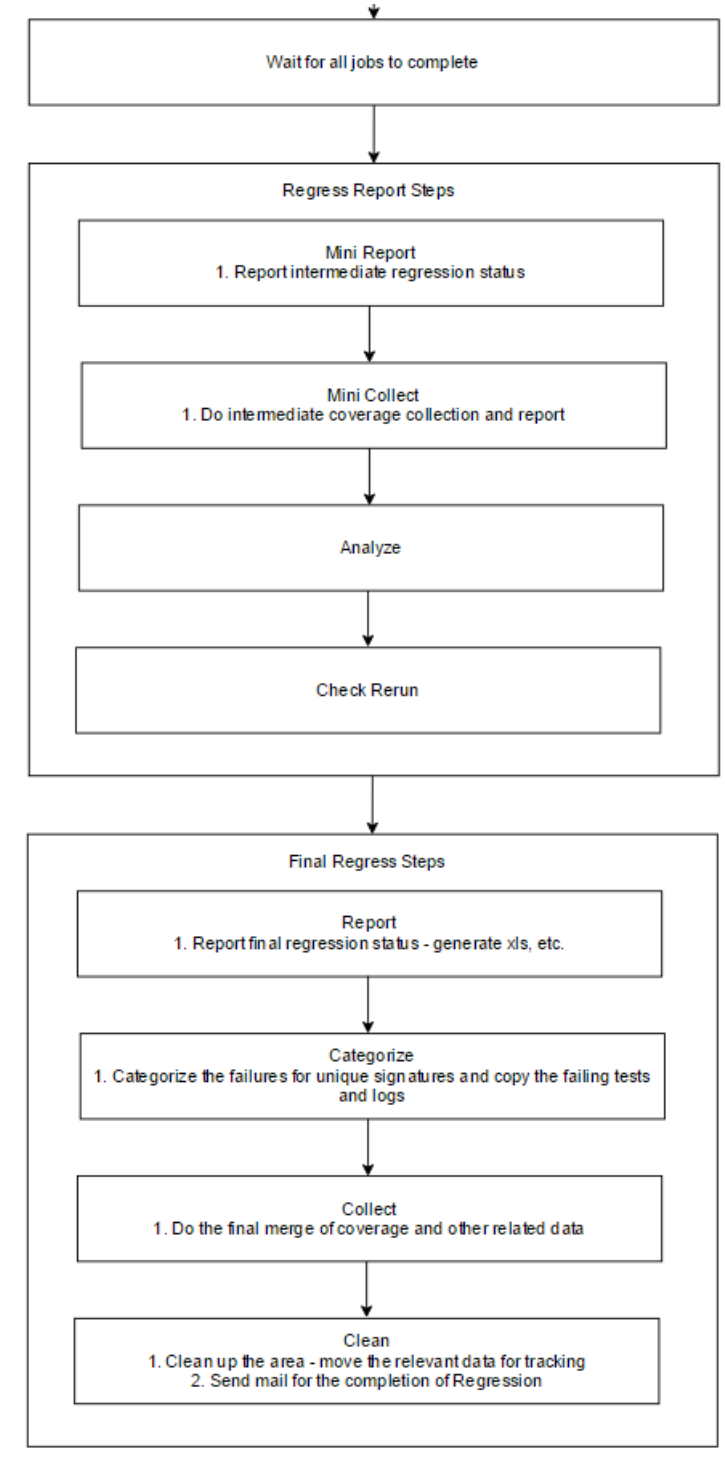

Figure 5.10: Flow Chart part4

Chapter 6

Optimization of Random Test Constraints Using Machine Learning Techniques

6.1 Introduction

- Modern designs are extremely convoluted
  - Impossible to come up with every viable combination of stimulus
- Constrained random simulation is a staple of verification
  - Generation of random instruction streams controlled through a set of tuned constraints
  - Great at hitting many common and uncommon design corners
- However, random testing is also inefficient and expensive!
- Random distributions hit most common cases most often, spending majority of the time testing the same things over and over
  - Hard to find bugs take a long time to find!

6.2 The Testing Loop

A new type of coverage is a way to extract information about a single test, to provide feedback on its quality. Testing loop is a way to use this feedback in machine learning

algorithms by optimization of designed to find hard to find bugs quicker.

6.2.1 Finding hard-to-find bugs

- Non-trivial bugs require a combination of events and state changes to occur in close proximity
- Most bugs arent particularly deep, it takes a couple of things to line up that we usually havent thought to line up
- Verification engineers bias stimulus towards areas that are likely to cause bugs
  - Great use of experience and knowledge to find most bugs
  - However, we cant just keep running the same things
- Need an objective way to evaluate test variety and coverage
  - Objective is the key we must eliminate the bias from hand-written functional coverage to find the hard-to-find corner cases

6.2.2 Exploring the state space

- One objective view of design coverage is its state space
  - State space of the design is represented by all of its flops
  - The total space size is 2flops, which is not practical to track
- The interesting things happen when state changes
  - Flop toggle coverage good start, but too simple, like CCOV

6.2.3 Lining things up

- Approximation for events lining up that takes design state into account: Two flops toggling in close proximity in time
- Still fairly simple to track (state space is flops**2), but much more interesting than single flop toggle
- Very objective requires no understanding of the design

6.3 Machine Learning through a Genetic Algorithm

- A type of reinforced learning algorithm
  - Select a random population of tests, and evaluate each
  - Create the next generation of tests by:
    - mutating (slightly adjusting constraints) current tests
    - mating (take an average of two tests) current tests
  - The evaluation score dictates the chance of a test participating in the next generation
  - Toggle pair coverage score used to select tests

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

Arbitrary guideline grouping apparatuses are generally utilized over the business for processor check and approval. RIS instruments are extremely compelling to create test situations that are difficult to imagine. Change in randomization of RIS apparatus causes us to builds hit rate of various corner cases. A few formats are composed in such way that it powers the test system to create settled situations to hit corner cases like direction improvement, crypto, right on time forward. Further we are planning to integrate tensorflow in the framework in order to make it intelligent. That will indirectly help the regressions to get better after every sessions

7.2 Future Scope

- Generate testcases from RIS tool that is targeted to memory area and debug the failed testcases.
- Analysis of different features of memory management unit. Analysis of Cache hierarchy and interpretation from virtual address to physical address and page table replacement policy.
- Understanding of micro-architecture and various components of it.
- Develop a framework that will analyse the quality of tests written by verification enginners.

7.3 Machine learning work conclusion

- This work is in early stages, and there are many ideas and trials to go through!
- Try other projects and designs
- Use meta-learning to learn the best GA parameters
- Continue to experiment with other ML algortihms

Bibliography

[1] ARM Architecture Reference Manual for ARMv8-A. Available: *http://infocenter.arm.com/help/topic/com.arm.doc.ddi0487b.b/DDI0487B$_{ba}$rmv8$_{a}$rm.pdf*.

[2] Jhon L. Hennessy, David A. Patterson, *A Quantitative Approach, 5th ed., MA: Morgan Kaufmann, 2012.*.

[3] Bruce Wile
`Comprehensive Functional Verification: The Complete Industry Cycle`