B. TECH. PROJECT REPORT

On

Infrastructure for Automated Regression Launch on Emulators

BY

Manish Shetty

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

Nov 2019

Infrastructure for Automated Regression Launch on Emulators

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in

ELECTRICAL ENGINEERING

Submitted by:
Manish Shetty

Guided by:
Dr. Santosh Kumar Vishvakarma
Associate Professor
Discipline of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY INDORE
Nov 2019

CANDIDATE'S DECLARATION

We hereby declare that the project entitled **"Infrastructure for Automated Regression Launch on Emulators"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Electrical Engineering' completed under the supervision of **Dr. Santosh Kumar Vishvakarma, Associate Professor, Discipline of Electrical Engineering,** IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student(s) with date

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

Signature of BTP Guide(s) with dates and their designation

Table of Contents

Preface

This report on “Infrastructure for Automated Regression Launch on Emulators" is prepared under the guidance of Dr. Santosh Kumar Vishvakarma.

In this internship report I will describe my experiences during my internship period. The internship report contains an overview of the internship company and the activities, tasks and projects that I have worked on during my internship. Writing this report, I will describe and reflect my learning objects and personal goals that I have set during my internship period. I have tried to discover the relationship between theoretical and practical type of knowledge and to bridge the gap between theoretical assumptions and practical necessities. With these objectives, I have made all possible efforts and the necessary investigations to submit this paper in an enlightened form in a very short time. I have tried my level best to eliminate errors from the paper. As I had to complete my internship within a short period of time so the study admits its limitations.

The report first shall give an overview of the tasks completed during the period of internship with technical details. Then the results obtained shall be discussed and analyzed.The report shall also elaborate on the future works which can be persuaded as an advancement of the current work. I have tried my best to keep report simple yet technically correct. I hope I succeed in my attempt.

I have gained new knowledge and skills. I achieved several of my learning goals. I got an insight into professional practice. I learned the different facets of working within a company.

Acknowledgements

This report has been prepared for the internship that has been done at Mentor Graphics Private Ltd. in order to study the practical aspect of the course and implementation of the theory in the real field with the purpose of fulfilling the requirements of the course of Electrical Engineering.

The aim of this internship is to be familiar with the practical aspects and uses of theoretical knowledge and clarifying my career goals, so I have successfully completed the internship and devised this report as the summary and the conclusion that have drawn from the internship experience.

I would like to express my sincere gratitude to my internship coordinator and mentor Dr. Santosh Kumar Vishwakarma who have given their valuable time and given me a chance to learn something. And I would like to thank him for supporting me during the completion and documentation of this internship

I would like to thank Mr. Chandra Jain for providing this opportunity to work in Mentor Graphics Pvt. Ltd. who assigned me this project and helped me to understand the importance of this project. I am also grateful to all the members of Mentor Emulator Division QA team for providing several documents, papers as well as sharing their experience with me and teaching me different techniques to build my knowledge. Thus, the time in Mentor Graphics was very audacious and supportive to my career through which I have gained valuable work experience that will help definitely makes a favorable impression on me as a prospective future employer. All information provided for Veloce is proprietary to Mentor Graphics.

Manish Shetty
B.Tech. IV Year
Discipline of Electrical Engineering
IIT Indore

Abstract

In VLSI industries verification is used to ensure that the manufactured products meet their specifications before fabrication. Faults often show their presence through abnormal outputs. EDA tools are used to find and then analyze these faults. Emulator is one such tool used for System-on-chip and embedded design verification. It uses a comodelling environment of hardware and software for it's working. Manual regression testing is done for such applications owing to its dependency on hardware and this makes this process a very time intensive task. The acquired bugs after the regression testing can also be prone to human errors. This is an iterative process where the after the fix the software is again sent for regression testing before clearing it for release. We propose herein an Automatic approach to launch regression testing for emulator which will create report and preserve the logs generated for further debugging. This solution enables fast results and more reliable and informed testing. The infrastructure can be launched again and again with ease to take care of all iterations of regression. Bugs found are further classified into hardware and software faults to send these bugs to their respective developers. So far, the developed system has been applied to the testing environment for Emulators. The preliminary results demonstrate that our system is valuable in this application scenario in that it can make the final report of the regression testing with all the required statistics which shortens the overall testing time.

List of Figures

Chapter 1
Introduction

Software has become an integral part of day-to-day modern life. The software affects almost every thing we use from digital alarm clocks to smartphones. To facilitate the growth of software there has to constant innovation in the existing software. Microsoft Windows Operating system can be seen as an example of software growth, it started from command line interface MS DOS and now it has a fancy GUI with Windows 10.

1.1 Regression

In every software industry there is a requirement to check that with every new release of software, it will be able to perform all the tasks that it's previous versions were capable of performing. The features introduced in the new version of software should not cause any hindrance in the old features of the software. This is tested by creating a suite of test cases for various scenarios that the software may have to go through during normal use. This suite also includes negative scenarios where the software must produce appropriate warnings and errors. There is a procedure of running this suite on software with every new version to validate it for release. This is called regression testing and the suite used for this process is called regression suite.

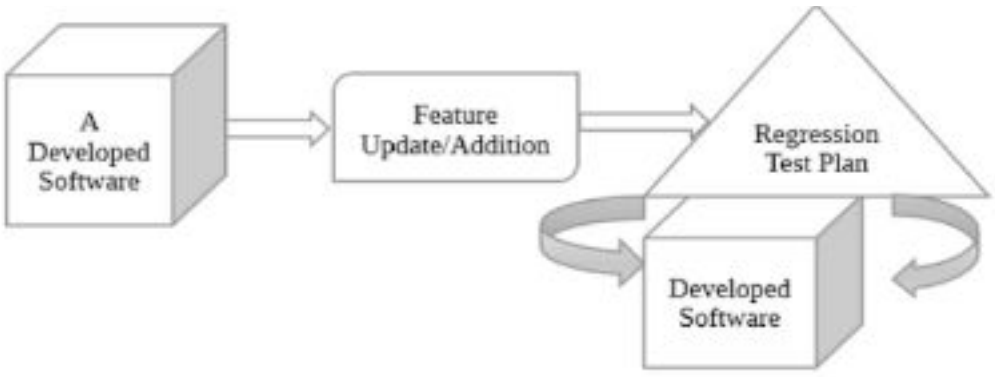

Figure 1-1 Regression Development cycle

Development and regression testing of software usually go hand in hand to facilitate improvement and advancement in already existing applications.

If regression testing is not done properly, there can be unexpected side effects in the software that reaches the customer leading to poor performance and also loss of its integrity among customers.

1.2 Problem Statement

The time required for regression had to be decreased, so that there won't be any delay in the release of the software. The goal was to automate this process which would give a report as output. This report generated could be sent to the developer for further debugging.

1.3 Motivation for work

For an emerging technology such as emulator, the hindrance caused due to regression can be removed by making this process happen automatically and using the time saved for further improvement in this technology.

As emulator is a hardware limited in number and available for a small amount of time. All the regression testing is usually done for pure software easily. It was a challenging task to make such a codependent combination of hardware and software automatic.

A new version of Veloce, an emulator created by Mentor Graphics is released every quarter of the year. The time and labour required to make testcases and sending them to emulators can be compensated by making an infrastructure that requires no manual intervention. The infrastructure once launched would send the provided list of test cases to the emulator after the compilation and check that the output obtained is correct. It would ease this regular procedure of validating every quarterly release of Veloce. The time saved from automatic regression testing can be used to make the emulation process more easy to use for customers with additional features. The testcases in the regression suite keeps on increasing as the software becomes bigger and more features are added. Eventually making it impossible to do regression handling manually. So automation is a viable step to take for regression.

Chapter 2
Background

Every hardware present today such as computers, audio recorders, smartphones and cameras use digital circuits to realize their function. These digital circuits are getting more and more complex as technology advances. In a micron of area, more than a million gates can be present. These gates are connected with each other to give outputs for certain inputs as desired by the designer. Doing this manually will be a very tedious task, leading to the requirement of a lot of manpower and time. So these circuits are written in a Hardware description language(HDL). A hardware description language is a computer programming language specialized to describe the structure and behavior of digital circuits. A textual description of the circuit to be designed is given using this language. Using this language RTL designing is done which is the first part of every ASIC design flow. RTL design is register transfer logic level design which describes a circuit in the form of registers that perform various register operations such as shift, arithmetic and logic operations. An automated analysis of this description is done to get the desired logic circuit.

2.1 Testbench Environment

The designer has to make sure that the hdl code written works according to the design specification. Chip fabrication is an expensive process so it is economically beneficial to ensure that the design works properly before fabrication. This is done using verification. The circuit is tested by applying signal vectors to the various input ports and checking the output corresponding to it. This is called direct testing. This can be done using hdl as well where signals are applied in the code itself called testbench. The output is observed after simulation of the hdl code is done. Simulation is the replication of the behaviour of the system using computer software by associating mathematical model with the said system. Here, the logic circuit derived from the hdl code is taken as the system for which the computer creates numerous mathematical models describing each net. So processes run to calculate the value at each net of the logical circuit. This is a very

time consuming task. So a testbench model is built which can be used again and again and will generate input signal vector on it's own.

A basic testbench model can be seen in Figure 2-1 where the model can be used for different DUT's. The purpose of the testbench is to generate stimulus , apply stimulus to the DUT , capture the response and check for correctness of the output.

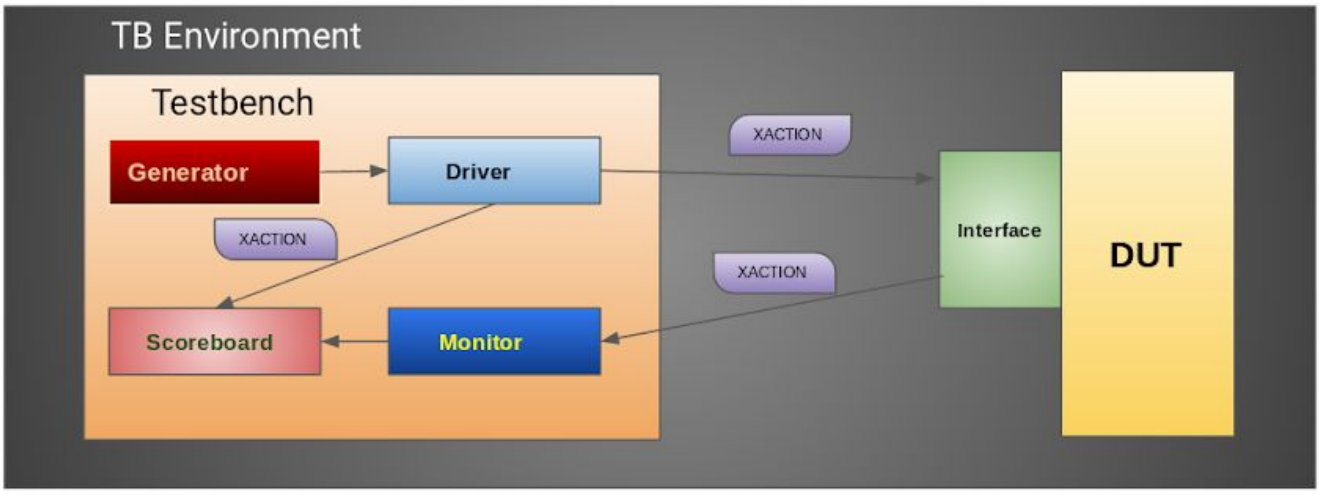

Figure 2-1 Testbench Environment

The verification plan is made using the hardware specification and contains a description of the features of the design that needs to be exercised. The testbench is made to execute this verification plan. Hardware verification language is used to create this testbench environment. The testbench tries to mimic the real world scenarios the design would face when various transactions are randomly sent towards it. Transactions are the data sent to the design which processes it to give output according to its specification. This testbench can be simulated for the design verification. A closer verification of the design is possible using emulators.

2.2 Emulation of the Testbench Environment

Emulator executes the SoC design for verification by synthesizing it on actual hardware which runs at the same speed at which the actual design after fabrication would have run. This increases the speed of verification as there is no requirement of simulating output to every transaction sent to the design like in simulators. Emulation provides a more thorough verification by completely mimicking the testbench model.

This is achieved by using Standard CoEmulation Modelling interface(SCEMI) in emulators. This specification describes a modeling interface which provides multiple channels of communication that allow software models describing system behavior to connect to structural models describing implementation of a device under test (DUT). Each communication channel is designed to transport un-timed messages of arbitrary abstraction between its two end points or "ports" of a channel.

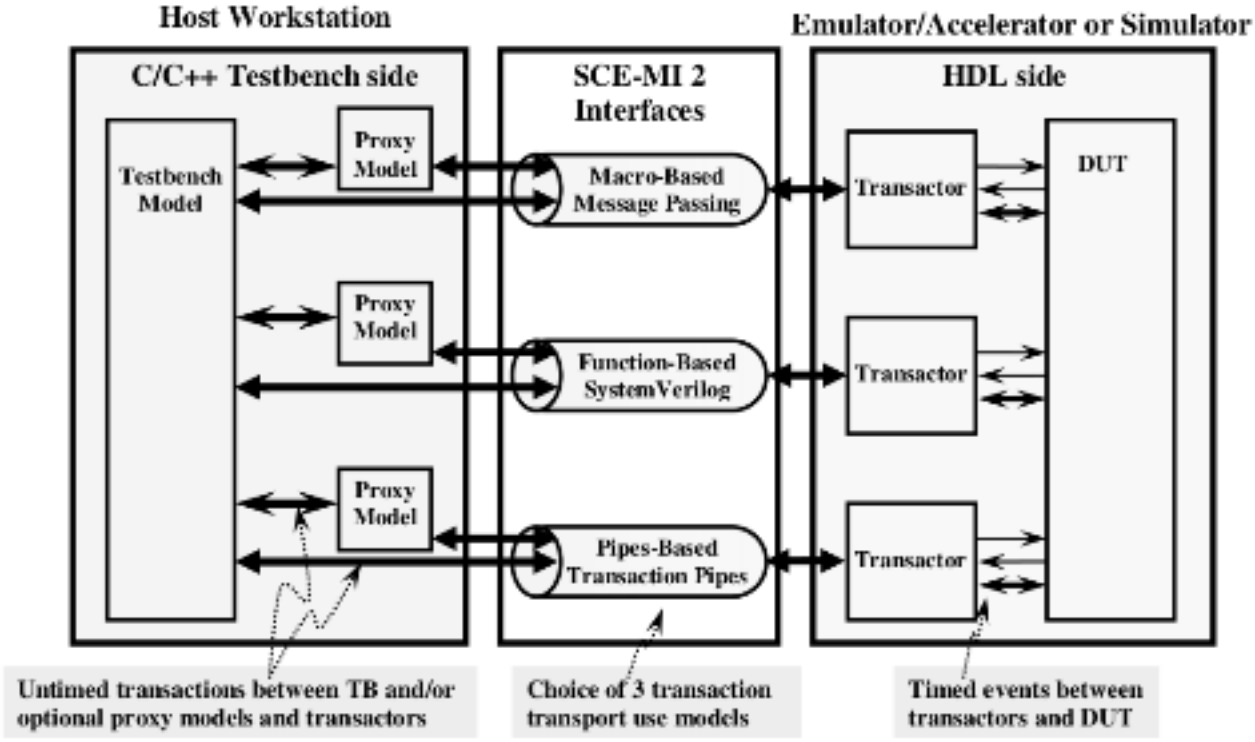

Figure 2-2 SCEMI model for Emulators

The division of testbench into software and hardware parts can be seen in Figure 2-2. The transactions are converted into timed signals by the transactor before sending it to the DUT. The software part is a host workstation on which C executables are run and the hardware part is the emulator which generates output

transactions for the transactions received from the software side.

Chapter 3
Emulators

3.1 Introduction To Emulators

Veloce emulators are hardware-assisted verification systems for System-on-Chip (SoC) and embedded system designs. Veloce provides transaction-based simulation acceleration, in-circuit emulation (ICE), VirtuaLab emulation, and targetless emulation to accelerate concurrent hardware and software validation and embedded software validation.

3.2 Emulator Working Environment

Veloce emulators operate in both a local hardware environment and a data center hardware environment.

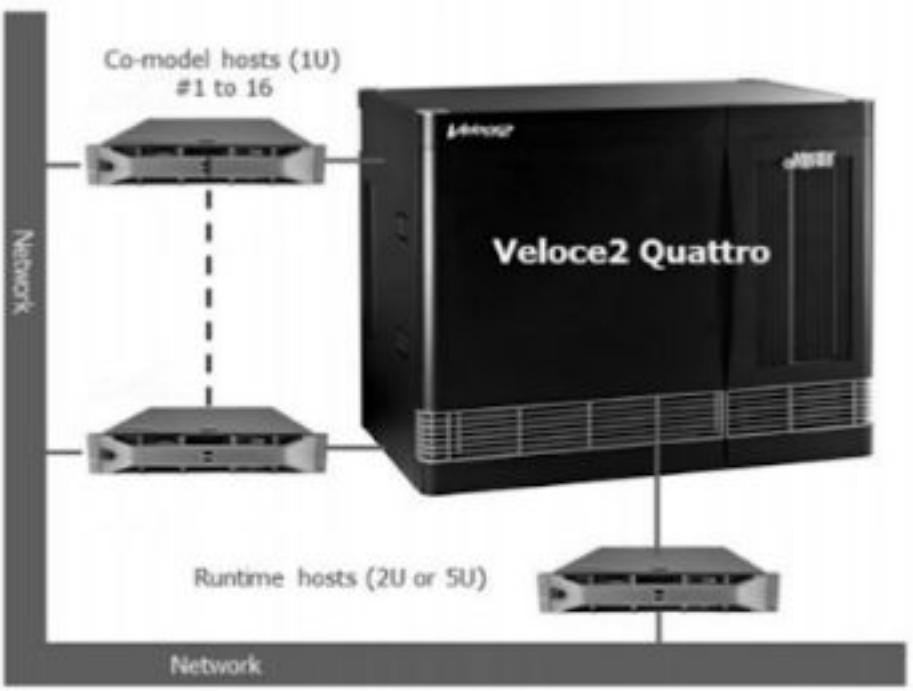

Figure 3-1 Veloce Emulation Environment

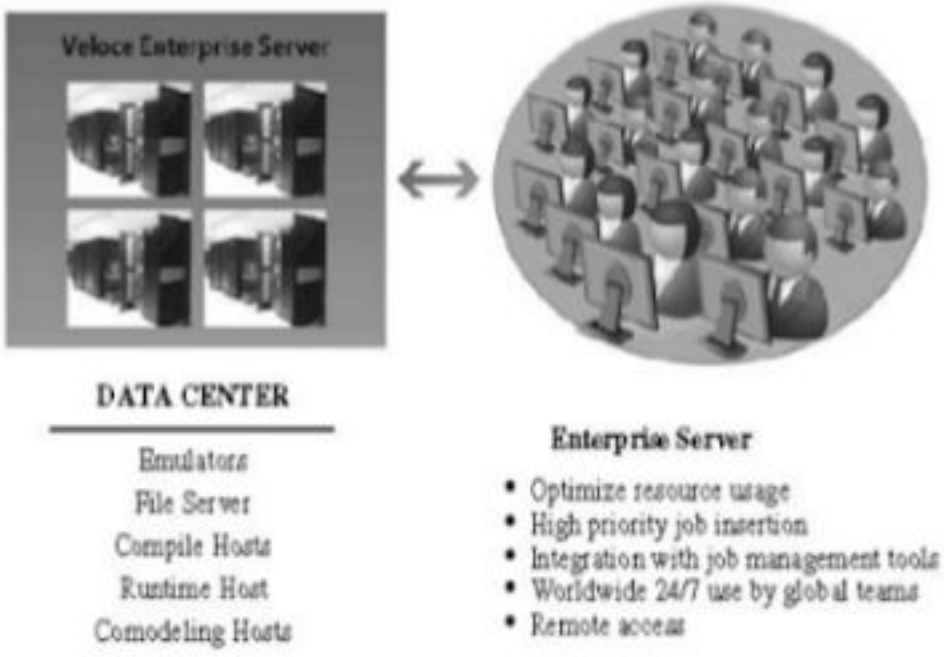

Figure 3-2 Veloce Enterprise Server Data Center Environment

Figure 3-1 shows a **local hardware environment**. All hardware is in one room, and users run the emulation flow from the local hardware.

Figure 3-2 shows a **data center hardware environment**. The compile, runtime, and comodel hosts reside in a network, perhaps on rack servers, and are accessible to anyone on your network—even across the world.

Veloce Emulator accelerates block and full SoC RTL simulations during all phases of the design process. Veloce can be used for transaction-level modelling of designs using it's Testbench Xpress (TBX) feature which establishes a transaction-level communication between testbenches running on a host system and SoC modeled in the Veloce emulator.

3.3 Types of Emulators

On the basis of their size,Veloce Emulators are further divided into three parts:

1. Veloce2 Quattro
2. Veloce2 Maximus
3. Veloce2 DoubleMaximus
4. Veloce StratoT
5. Veloce StratoM

Veloce2 Quattro contains 16 AVBs (advanced verification boards), 6 SXBs(switch matrix boards) ,4 IO Partitions, 4 Comodel channels and upto 4 users can use it at one time.

Veloce2 Maximus contains 64 AVBs, 60 SXBs ,8 IO Partitions, 16 Comodel Channels and upto 16 users can use it at one time.

Veloce Strato contains 16 AVBs, 8 SXBs, 8 IO Partitions, 4 Comodel Channels and upto 4 users can use it at one time.

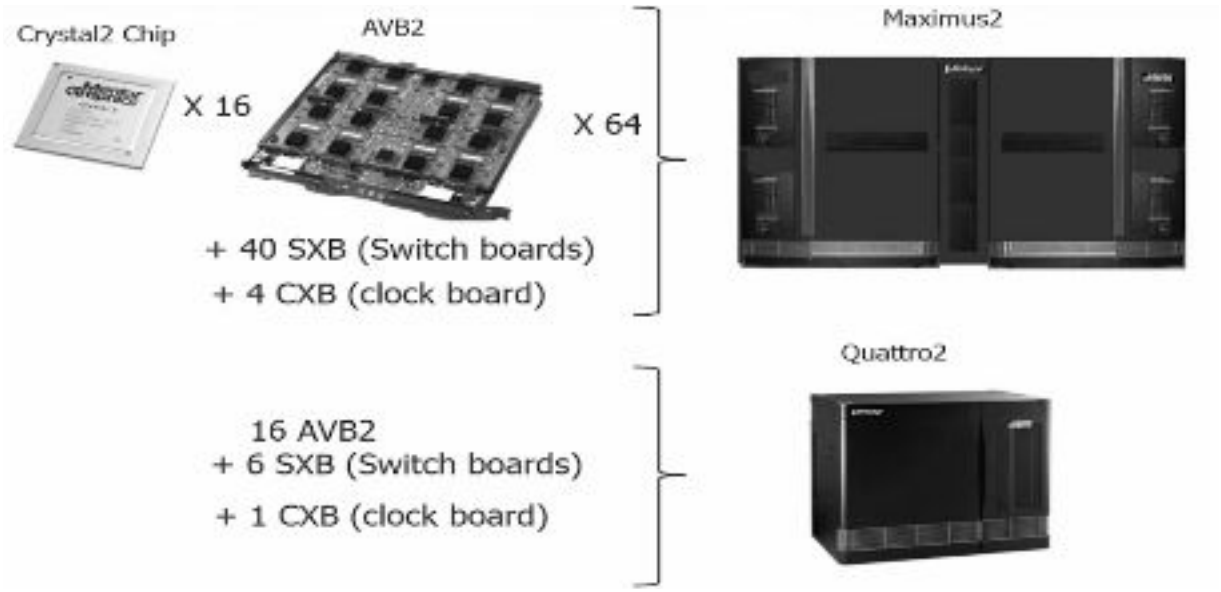

Figure 3-3 Veloce Internals

3.4 Internals of Emulator

3.4.1 AVB (Advanced Verification Board)

Advanced Verification Board are the smallest unit in emulator which helps in emulation. These are the boards on which the design is loaded for emulation. It contains 16 crystal chips per AVB and there can be multiple AVBs in an emulator. After compilation number of AVBs required is decided and get reserved for the particular design for emulation.

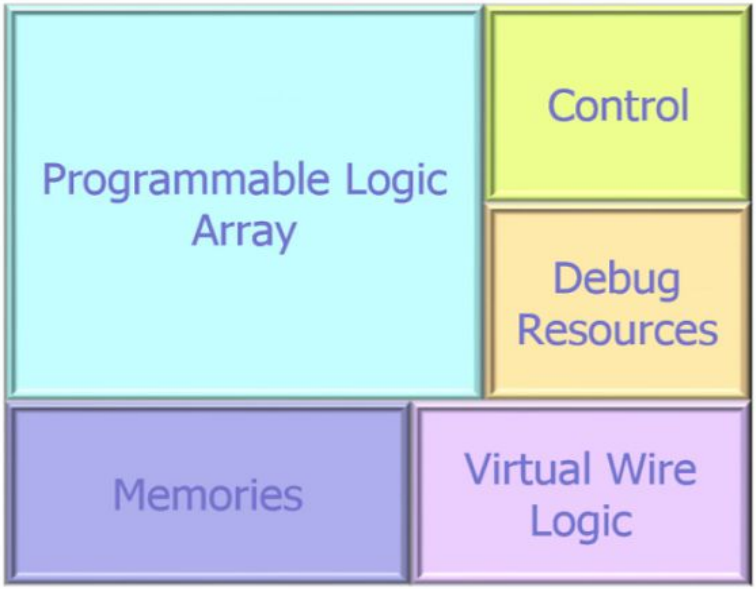

Figure 3-4 Basic Layout of a Crystal chip

3.4.2 Crystal Chip

Crystal chips are similar to LUT's present in FPGA.These are programmable logic which helps in emulation of 400K to 500K gates. It contains 16 CMEM (Chip user MEMory). It have read and write access to all the states (Flipflops and CMEMs) when clocks are stopped. It contains Trace controller to capture data for waveform visualization for SSR (software state replay process) or FT visibility which is used for debugging the results obtained after emulating the design. It also contains trigger reduction logic to support arbitrary trigger on any user state which can be used for debugging.

3.5 Comodel Channels on an Emulator

On a Veloce Strato, every AVB3 is connected to a comodel host, so it has no mapping limitations of logical-to-physical AVB3s based on presence or absence of a comodel host. Every module always has at least two comodel hosts, and every AVB3 has access to a comodel channel.

- A module can be connected to two, four, or eight PHB3s.
- Each PHB3 connects to multiple AVB3s: 16 per PHB3
- At runtime, a logical AVB3 with a comodel channel can map to any physical AVB3.

3.6 Use of Multi Comodel links in Emulators

There can be multiple comodel links between software and hardware so each comodel link has a dedicated role during emulation. A single comodel interface can become a bottleneck for testbench environments involving too much communication between hardware and software, and thereby degrading runtime performance. Increasing bandwidth to enhance the runtime performance becomes apparent for cases crossing the threshold of single-channel bandwidth. Veloce emulators have multiple physical channels (commonly referred to as comodel links) for communication with comodel hosts employed for multi-user mode. The multiple comodel features raise the bandwidth, because they use multiple physical channels for communication to hardware from the software side for single-design emulation. The performance gain can be many folds depending on how much parallelism is exploited at the communication level and may be at testbench level by concurrent execution of multiple HVL-side testbench processes.

3.7 Emulator IO Partitions

Veloce Strato has eight IO partitions per module.Veloce Strato IOs are organized in two rows of four. The Veloce emulation system can serve multiple users simultaneously using these IO partitions. The system can support multiple users at runtime, the runtime system being a collection of multiple clients and servers connected through a messaging system.

3.8 Memories supported by Emulators

There are four types of memories used in veloce based on their size for mapping memories present in the design. There are definite number of instances of these memory available per crystal .

They are:

1. **XMEM**: This is the smallest memory available for mapping memories of design. This memory has a size of 1 KB (1K x 8 bits) and there are 560 instances per crystal of this memory. This memory is only available in Veloce Strato Emulators.
2. **CMEM**: This is Chip Memory .This memory has a size 256 KB(64 K x 32bits) for veloce strato emulators and a size of 128 KB(32K x 32 bits) for veloce2 emulators. There are 64 instances per crystal of this memory in veloce strato emulators and 16 instances per crystal in veloce2 emulators. This memory has 4 read and 4 write ports.
3. **LCMEM**: This memory is second largest memory available for mapping design memories. This memory has a size of 4 MB (1 Mx 32 bits) and there are 4 instances per crystal of this memory. This memory has 4 read and 4 write ports and this memory is only available in Veloce Strato Emulators.
4. **SBMEM**: This is the largest memory available for mapping design memories. This memory has a size of 1 GB (64 M x 128 bits) and there are 4 instances per SXB of this memory.this memory has 1 read and 1 write port and is only available in Veloce Strato Emulators. SBMEMs are a new type of resource for large memory. A SBMEM is a system resource located outside the AVB3 rather than being an AVB3 Resource.SBMEMs are divided into eight SBMEM partitions. You can use one for each of eight multi-board users. SBMEM requirements are determined during compilation, and the number of SBMEMs can be anywhere from zero to eight for a single-module design.

Chapter 4
Emulation Flow

There are two stages in using Veloce first is compilation and then running emulation. Compilation involves checking the syntax of RTL code provided which is then send for elaboration that converts the RTL code to a circuit made of logic gates.

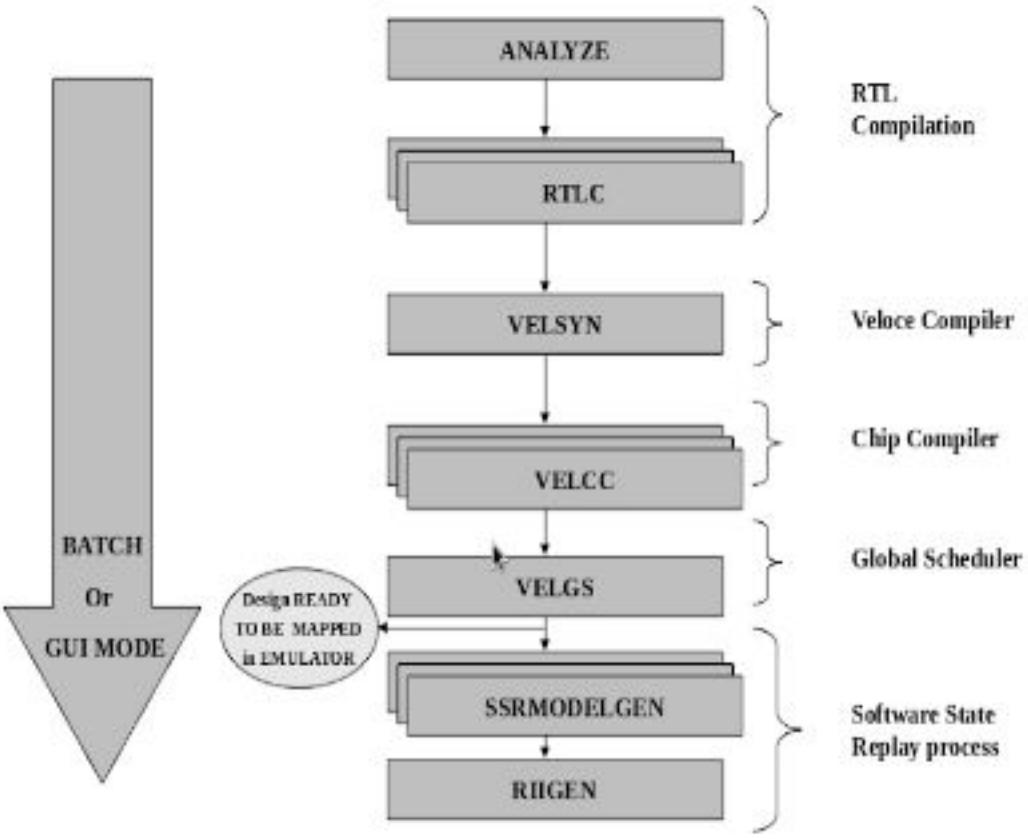

Figure 4-1 Compile Flow

Optimization is done to remove dead logic from the circuit whose output is not used anywhere during the verification. After elaboration the logic gates are mapped onto crystal chips present in AVB's of the emulator, thus synthesizing the design. During synthesis, the code is partitioned into RTL and XRTL divisions. RTL part is the conventionally synthesizable part of the code and XRTL is part of the code which is made synthesizable by Veloce. For example, $display is seen as non-synthesizable construct in Verilog but

in Veloce it gets mapped onto a memory which can then be accessed by the host server during emulation. Placement and routing of the crystal chips takes place after synthesis. This decides the frequency at which the design will run on emulator.

4.1 Stages in Emulation

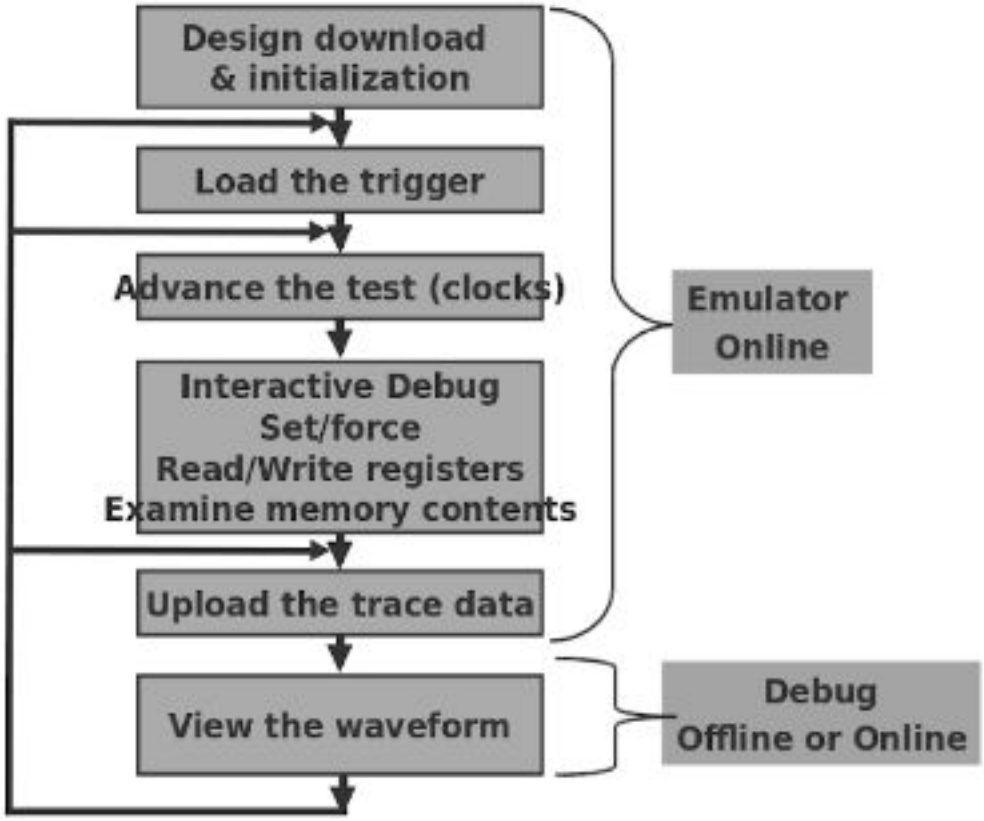

Figure 4-2 Runtime Flow

- **Analyse:** This command does syntax checking of the given source RTL design files.
  The RTL source files or gate level netlist are analyzed before sending it to compilation. This command ensures that no syntax errors are passed on to the elaboration process.
- **RTLC:** This is RTL Compiler. This command is used to convert RTL source files or structural netlist or gate level netlist into structural verilog netlist for veloce primitives i.e. veloce specific netlist. The velcomp utility invokes the XRTL and RTL compilers to compile XRTL and RTL modules in the design. This process takes options from veloce.config file.

- **Veloce.config file**: A veloce.config file is required to run Veloce emulation. It contains the options used during analysis,compile, and runtime.You must specify all required parameters for compilation in the configuration file before compilation. You cannot modify or specify parameters during compilation.

- **Velsyn:** This is Veloce Synthesiser.This command in a configuration file causes velcomp to pass specific options for the Veloce compile phase to the velsyn compiler. Using the velsyn command, you can pass the required velsyn options for the Veloce compile phase.
- **Velcc:** The Veloce crystal compiler (velcc) generates timing databases and configures binary files for the emulation chips.This command helps in placing and routing of crystal chip.
- **Velgs:** This command generates the global scheduling database (GSDB). This database contains comprehensive scheduling information.this command generates timing information for emulator events and resource access.
- **SSRMODELGEN:** This is Software State Replay MODEL GENeration. This is used to generate models for replay states for each chip. This command follows parallelisation (one per chip). This is necessary only for waveform visualization i.e. waveform analysis.
- **RIIGEN:** This is Recon Input Index GENeration . this command is used to create databases visibility system.

For Compiling HVL models Veloce provides a command Velhvl. This command compiles HVL models as an alternative to the velcomp -task hvl option. The compiled HVL model (also referred to as a comodel executable), communicates with the design in Veloce through the comodel transaction interface. It generates a library with the name you specify, includes the object code, and then takes the compiled SystemC work library source code and links the design. For Running Emulation Veloce provides a command Velrun ,which automatically connects, downloads the design, launches comodel applications, runs emulation, and exits when finished. For Questa testbench environments, use vsim instead of velrun. After Emulation , the next task is to debug, open and view the results obtained from emulation ,for this veloce provides a command Velview. This command is used to debug, open and view captured trace data.This command is also used to see the current design state, suspend or halt the design, and perform a waveform or memory upload on a batch job.

4.2 Use Modes for the Emulation

The primary use modes of emulation are:

• **Stand-alone emulation** — The Design Under Test (DUT) and testbenches exist within the emulator. There are no interfaces to systems outside the emulator, so emulation performance is not limited by external systems. An example of this use mode would occur with a completely compilable testbench, or test patterns being loaded into emulator memory.

• **In-Circuit Emulation (ICE)** — The DUT exists within the emulator. The rest of the verification system is connected to an ICE target system. You supply the target system that models the real-world environment where the final chip will exist.

• **Co-simulation and Co-modeling** — The DUT within the emulator communicates with a program or a simulator running on a co-modeling workstation.

1. **Co-simulation** refers to running an emulation design with a Verilog or VHDL testbench on the workstation. This is often a performance-limited emulation due to the speeds of HDL software simulators.
2. **Co-modeling** refers to running an emulation design with a C model or testbench on the workstation. C models generally run at much faster speeds as they represent models at higher abstraction levels. The C-to-emulation interface is brought to the transaction level, thus eliminating the potential problem of a bottleneck within the interface. This is made possible through Mentor Graphics transaction-based emulation technology.

To take advantage of the co-modelling environment of Veloce, the testbench is divided into HDL and HVL(hardware verification language). The HVL part made in C++ is simulated on a comodel host which sends transactions to the DUT present on emulator. Thus, creating a real-time environment for the design to be verified. The HDL part includes the actual RTL design and transactors such as PCI, USB and Memory collocated inside the emulator in Veloce. Transactors do the job of converting the transactions received from the HVL part into stimuli to drive the DUT. The HVL part uses SystemVerilog DPI(Direct programming interface) tasks and functions to create transactions. SystemVerilog DPI is basically an interface between SystemVerilog and a foreign language such as C,C++ and System C.

A proper verification environment for the design in Veloce can be created using both the HDL and HVL. UVM(Universal Verification Methodology) can also be used to verify designs in Veloce. To take full advantage of Veloce for verification, you need to know the proper pragmas that should be added in each part of the code. Pragmas are normal comments for all other system verilog compilers but have special meaning that only Veloce compilers are designed to understand. For example, "//tbx clkgen" will be treated as comment by other compilers but Veloce's TBX feature will treat it as a command to specify that the following sequential block should be synthesized as clock generator. Thus, to create test cases for Veloce, additional time is required for adding these pragmas to the testbench code**.**

Chapter 5
Proposed System

5.1 Infrastructure Model

There are various emulators with different number of AVB's and different OS version such as Strato,D2 in it, so the name of emulator which should be specified during the execution of infrastructure. The settings file include various environment settings that should be set while testcase is running on emulator and the version of Veloce that will be tested. The regression suite contains more than 4000 testcases so keeping a track of the status of each testcase is a difficult job. So, the idea was to make this infrastructure automatically generate report of passing and failing along with the reason for failure.

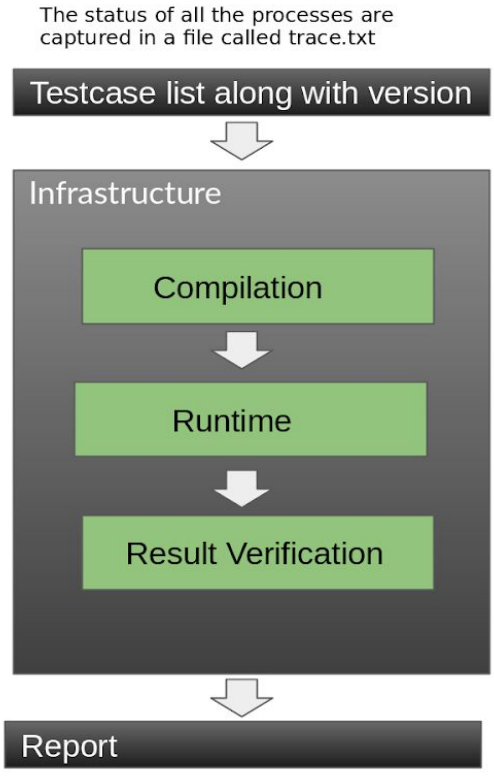

Figure 5-1 Infrastructure Model

Compilation:

A list of testcases is provided which is pulled from a git repository which contains all the hdl files and commands required to perform the compilation and run commands for the emulator.All testcases from the list are launched on the **grid** parallely for compilation.

Emulation:

All the passing testcases(compiled testcases) are queued to run on emulator. The passing and failing testcase lists are created.The emulator is powered on with the specific version and the testcases are picked from the queue and run commands are executed in Veloce. The output generated is captured in a file. And the status is continuously captured in a file until all the queued testcases are run on emulator.

Result Verification:

The output generated after running the design on Emulator is verified. This is done by comparing the generated output with expected output which is considered as golden file.A finalised report is generated indicating all the statistics of the launched regression to create a list of failures which needs to be corrected in high priority in software and hardware part of Veloce.

5.2 Challenges Faced

A huge number of failures appeared in Result Verification phase as the testcase developers weren't updating their golden files.The report generation script kept a database of old diff failures (which weren't updated by testcase owners of previous version)and report them differently from new diff failures. Some testcases got hanged during Runtime which caused the infrastructure to hang, thus halting the entire process.Time out was added to overcome this which killed the testcase running on emulator after some time.A number of new errors popped up which weren't related to Veloce software such as machines in the grid not responding and emulator getting powered off by someone in the middle of launch. These were called left testcases as they couldn't be categorized into passed or failed list. An error differentiating algorithm was

added which kept a track of all the phases during the regression and made a decision in which category the error belonged. Regressions had to be launched a number of times to take care of left and failing testcases(which were resolved afterwards) and all the results had to be accumulated to make the final report with no left and failing testcases. A rerun infrastructure was created to take care of left testcases after first run completes. This rerun infra launches automatically only the failing and left testcases on the same dump area till the left testcases become zero.

Chapter 6
Contribution and Results

6.1 Contribution towards Infrastructure

I was part of the team that looked into reporting bugs found in regression. My task was to make a smooth functioning regression launch infrastructure and see that this infrastructure will work on all corner cases.

The list provided to the infrastructure first send for compilation. The compilation of all testcases present in regression suite run parallelly on a grid of available host machines. A problem was produced due to this where multiple testcases after compilation tried to write their status in the same compile passed or failed list simultaneously leading to race condition. This caused the testcase names to get concatenated or get skipped altogether from the list creating a problem while making the final report for regression.

A file locking script was made by me in perl to overcome this issue. The script creates an exclusive lock for the file and if two processes try to write to the file at the same time, one which arrives even a nanosecond early will receive the lock and can write to the file whereas the other file has to wait till the first process releases the lock.

After compilation all the passed testcases are sent to emulator in a queue. Testcases are picked one by one depending on the resources available on the emulator by a daemon. Daemon is a process that runs in the background and performs a specified operation in response to certain events. After emuation the report is generated on the dump area where all the status of the testcases ran by infrastructure are kept.

I made the reporting script such that along with number of passing and failing testcases, it also specifies the reason for failure and whether that failure was present in older release of Veloce or a never seen before failure which should be sent for analysis. The output generated from the emulator is compared with the expected output called golden file. I made this script robust enough to also report and relaunch testcases which were failed or left from emulation due to certain circumstances such as the emulator getting powered off. The rerun script is launched after the regression gets completed.

6.2 Results

Figure 6-1 Report output from the infrastructure

Figure 6-1 shows the report generated from the infrastructure. The main goal of this report is to give the proper statistics of the status of all the testcases in the regression suite. Dump area is the directory in which all the files generated during the compilation and emulation process is preserved for use during debugging. It also stores the status of each of the process as the infrastructure takes the testcase through the emulation flow. The error differentiation can be seen as well. This divisions are done into known failures, diff failures, golden diff failures, timed out and velrun failures.

Known failures are failures that weren't resolved and are still present in the software. Diff failures are failures produced during result verification phase. Golden diff failures are diff failures which have been

present in the previous version regression and were. Timed out failures are testcases which were killed by the infrastructure as it exceeded the default time limit. Velrun failures are produced if some hardware issue occurs while the testcase is running on emulator. The lists of respective failures are created and sent to their respective owners for correction.

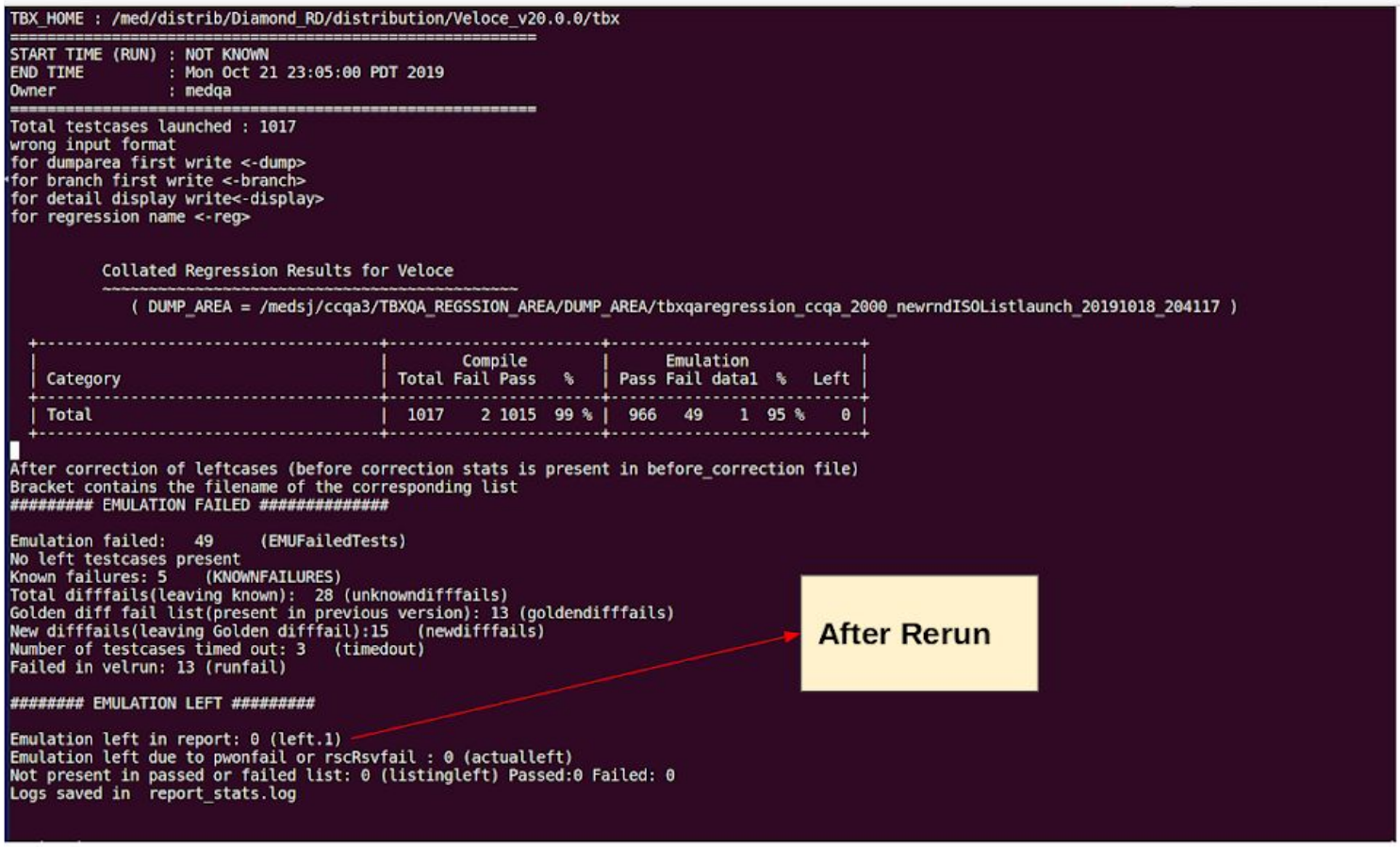

Figure 6-2 Report output after rerun

The left testcases produced are divided into two parts, one which denotes power on fail of the emulator and the other is testcase left which got executed but didn't appear in any of the lists. The number for the second part is zero as internal scripts work to remove these false errors. The first part of the left testcases are made zero by rerunning the infrastructure automatically. The new report generated after rerun can be seen in Figure 6-2. A decrease in timed out testcases can also be seen. This leads to change in the overall statistics of other failures as well. Thus, showing the importance of rerun.

6.3 Other Contributions

My job also involved making test scenarios for new features released in Veloce. I have made more than 20 testbenches in SystemVerilog for this purpose. Designs for testbench included fifo, controllers to test all the newly included commands in Veloce. A generic design having specified number of custom sized memories was also made to check memory shadow capability of Veloce. The memory shadow capability, below a certain amount of memory size, will use registers instead of using actual memories to preserve scarce memory resources available in emulator. I have also made algorithms to check whether correct number of flip flops and latches were synthesized for the designs. The testbenches were created with a proper environment containing generator,driver, monitor and scoreboard. So that the testbench can be reused for different transactions

I also built a system which will be used for performance testing of Veloce's capability of generating power report for designs running on it. The benchmarking will be done on the time taken to generate the report and accuracy with respect to a third party software commonly used for power estimation. The plan was to use some large SoC design from customer semiconductor industries along with their testbench for this benchmarking. There are two ways of getting switching activity from Veloce which will be used for calculating power. Online version creates power reports while the testbench is running on the emulator using the trace as they are generated. Trace is the value of the signals at specific timestamps. The offline version uses the FSDB file made after the complete trace is generated and does not require emulator. These methods of using Veloce for power generation were compared in terms of peak memory usage of host machine and time taken to generate report.

.

Chapter 7
Conclusion

This project was made with the purpose of achieving time efficiency for regression handing of Veloce software. The period for regression went from 15 days to 3 days. The number of testcases that can be handled at once went from 1000 to 6000 This infrastructure can be seen as an automation that uses an amalgamation of software and hardware.

7.1 Future Work

The work done by the infrastructure is done within the time constraints for the testing part. The same can be brought into the debugging process. This system can be enhanced to classify never seen errors produced and suggest ways to resolve it using machine learning. The patterns seen from previous errors can be used to make decisions for the new errors.

References

[1] Mentor Graphics Website: https://www.mentor.com/products/fv/emulation-systems/

[2] SystemVerilog for Verification: A Guide to Learning the Testbench Language Features, Chris Spears.

[3] Veloce User Guide, proprietary to Mentor Graphics

[4] Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual, Accelera