International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

VERIFICATION OF RISC-V PROCESSOR USING UVM TESTBENCH

Chevella Anilkumar[1], K Venkateswarlu[2]

[1.2] *ECE Department, JNTU HYDERABAD(INDIA)*

ABSTRACT

RISC-V (pronounced "risk-five") is a new, open, and completely free general-purpose instruction set architecture (ISA) developed at UC Berkeley. It is designed to be useful in modern computerized devices such as warehouse-scale cloud computers, high-end mobile phones and the smallest systems. The RISC-V instruction set is for practical computers. It have features to increase a computer's speed yet reduce its cost and power use. These include a load-store design, bit patterns to simplify the multiplexers in a CPU. The instruction set is designed for a wide range of uses. The instruction set is variable-width and extensible, so that more encoding bits can always be added.

The test bench is implemented using System Verilog, the Universal Verification Methodology and C++. System Verilog is Verilog with extensions to support object-oriented programming, improved synchronization and functional coverage. UVM is a joint effort between Mentor and Cadence to develop an SV library of common blocks and features to expedite the creation of SV test benches. UVM also defines a standard implementation methodology to follow.

The test bench determines generation of the stimulus, applies it to the DUT and Reference Model, collects it and scores it to determine the functional coverage. The test bench makes extensive use of the predefined classes in UVM. The Reference Model runs with the RS64 RTL to provide on-the-fly checking during simulation. The random constraint solver is used with sequences to create complex test scenarios controlling multiple interfaces.

KeyWords :C++DUT,ISA,RISC,RS64 RTL,UVM,VERILOG

I. INTRODUCTION

Today's devices are highly integrated, enabling a single chip to perform many functions such as networking, wireless etc. Each function is done by an Intellectual Property (IP). Grouping all the IPs and allowing them to communicate on one chip is called System on Chip (SoC).

The increasing demand for high-performance portable SoC's in communication and computing has added the power consumption to the traditional constraints, such as area, performance, cost, and reliability digital designs.

The main agenda of this project is to verify the RISC-V processor.

The VLSI design cycle starts with a formal specification of a VLSI chip, follows a series of steps, and eventually produces a packaged chip. VLSI design Flow is generally divided into two phases – Front End and Back end

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

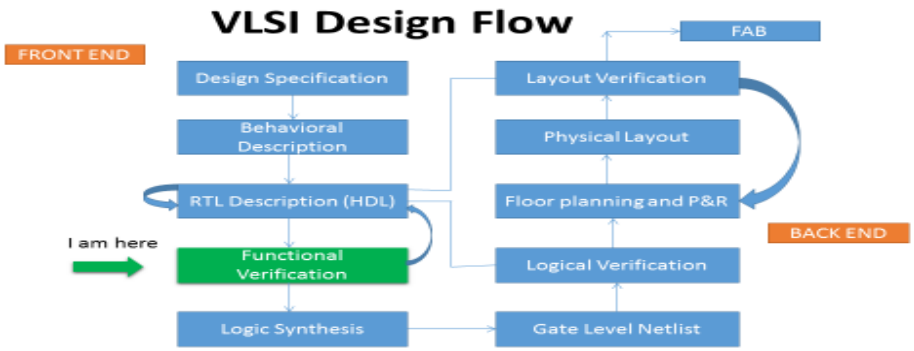

Fig.1 VLSI design flow

1.1 VERIFICATION FLOW

The verification process comprises of the following three steps: Verification Setup, Iterative Execution, and Quality Assessment & Termination. Each step has unique resource requirements.

The Verification setup includes setting up the verification environment and inputs to the verification system. In the most commonly deployed simulation based methodology, this requires setting up the run scripts and test bench. The test bench is made up of Error Checkers and Stimulus Generators. Typically, this is considered the responsibility of verification engineers. An initial setup starts the verification process and the setup itself may be iteratively refined as verification progresses.

Iterative execution requires running the verification flow, flagging errors, debugging errors and fixing the design to start the new iteration. Typically, verification engineers support running of the verification flow and initial debugging of the flagged errors to ensure test bench correctness. The design engineers debug the flagged errors and make design fixes. Once the design starts maturing and the bug rate falls down, the debugging burden on the verification engineers reduces and the effort is focused on Quality Assessment (Coverage, etc.). This may require test bench refinements, which may lead to more iteration. Upon reaching the quality targets, the process can be terminated.

Major verification costs include costs for setup (test bench development), debugging (verification and design engineering effort) and quality assessment (coverage). In addition, running the flows also has computation costs. The engineering costs provide a measure of Usability of the flow.

1.2 Universal Verification Methodology (UVM) for Verification

UVM provides the best framework to achieve coverage-driven verification (CDV). CDV combines automatic test generation, self-checking test benches, and coverage metrics to significantly reduce the time spent verifying a design. The purpose of CDV is to:

— Eliminate the effort and time spent creating hundreds of tests.

— Ensure thorough verification using up-front goal setting.

— Receive early error notifications and deploy run-time checking and error analysis to simplify debugging.

UVM helps setting verification goals using an organized planning process. Firstly a smart test bench is created that generates legal stimuli and sends it to the DUT. Coverage monitors are added to the environment to measure progress and identify non-exercised functionality. Checkers are added to identify undesired DUT behavior. Simulations are launched after both the coverage model and test bench have been implemented. Verification then can be achieved

1.3 INTRODUCTION TO RISC-V

RISC-V (pronounced “risk-five") is a new instruction set architecture (ISA) that was originally designed to support computer architecture research and education, but which we now hope will also become a standard free and open architecture for industry implementations. RISC-V include:

i. A completely open ISA that is freely available to academia and industry.
ii. A real ISA suitable for direct native hardware implementation, not just simulation or binary translation.
iii. An ISA that avoids “over-architecting" for a particular microarchitecture style (e.g., micro coded, in-order, decoupled, out-of-order) or implementation technology (e.g., full-custom, ASIC, FPGA), but which allows efficient implementation in any of these.
iv. An ISA separated into a small base integer ISA, usable by itself as a base for customized accelerators or for educational purposes, and optional standard extensions, to support general purpose software development.
v. Support for the revised 2008 IEEE-754 floating-point standard.
vi. An ISA supporting extensive user-level ISA extensions and specialized variants.
vii. Both 32-bit and 64-bit address space variants for applications, operating system kernels, and hardware implementations.
viii. An ISA with support for highly-parallel multicore or many core implementations, including heterogeneous multiprocessors.
ix. Optional variable-length instructions to both expand available instruction encoding space and to support an optional dense instruction encoding for improved performance, static code size, and energy efficiency.
x. A fully virtualizable ISA to ease hypervisor development.
xi. An ISA that simplifies experiments with new supervisor-level and hypervisor-level ISA designs.

The UVM library and methodology provides all necessary features the technologies for constructing reusable, constrained-random and coverage-driven test-benches. It provides the TLM-based modelling for building modular and reusable verification components which communicate through transaction-level interfaces.

The UVM class library allows us creating sequential constrained-random stimulus which helps to collect and analyze the functional coverage and include assertions configured as members of those test-bench environments.

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

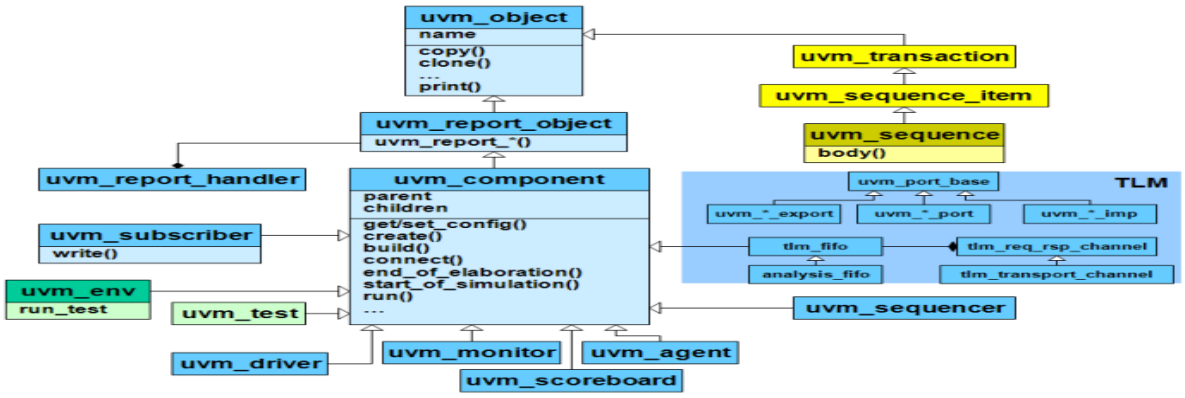

Fig.2 UVM Class Hierarchies

II. IMPLEMENTATION

2.1 Introduction

This chapter covers top level RISCV Verification Environment will be shown and each block of the environment will be explained. It also covers the working and execution of environment at block level and IP level. It will also give Introduction about Coverage reports and their importance.

2.2 RISC-V Verification

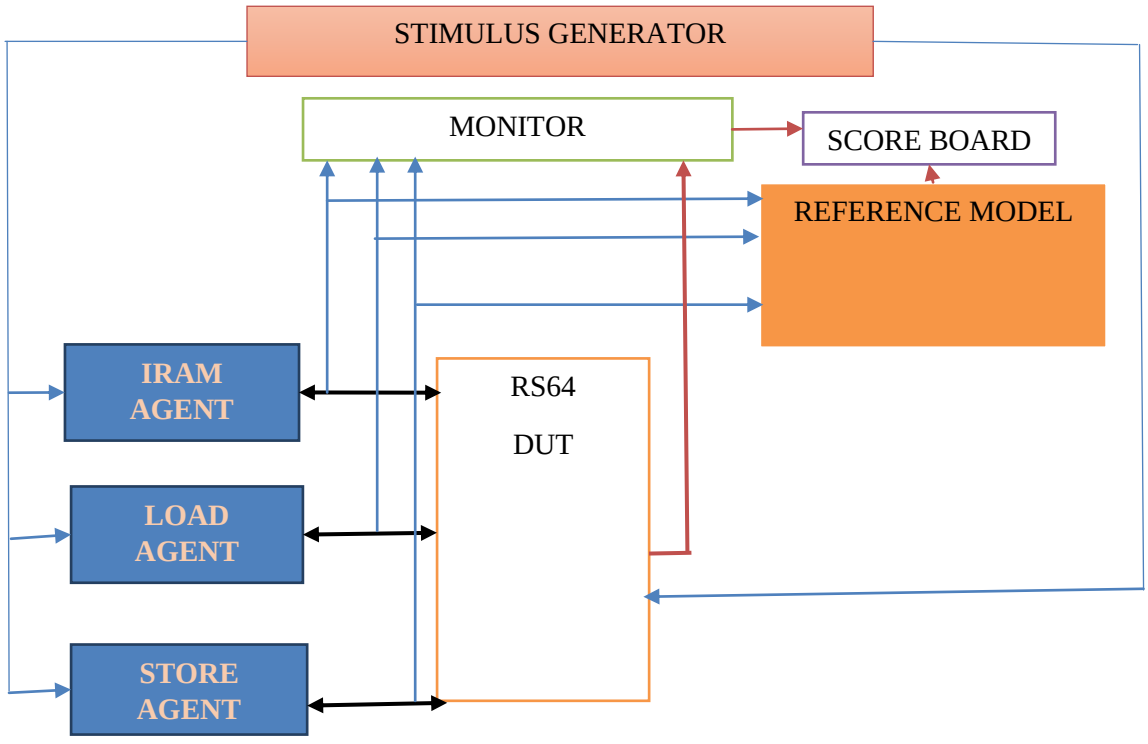

Fig.3 Verification Environment

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

The verification of rs64 will be in different steps. In first step functional coverage model is created which is based on RS64 specifications and instruction set and sequence of instructions to verify. This functional coverage model is used to define testbench, sequence items, sequences, tests and functional coverage. In second step the actual testbench is architected to enable the stimulus to verify RS64 response. In the architecture sequence items, agents, sequencers, monitors, reference model are included. In third step DUT, DUT drivers are used to test whether the environment is working properly or not. Once the environment is working then attach reference model. When RTL and reference model are working properly then checkers were added to compare results.

2.3 Working and Execution

The below flow diagram shows the sequence in which test case is executed step-by-step.

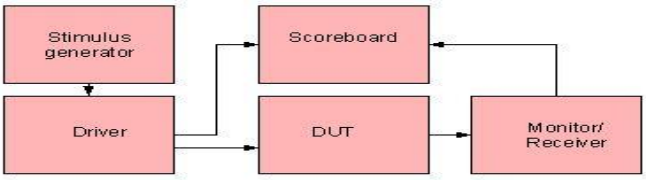

Fig.4 Flow diagram for Test case Execution

III. RESULTS

3.1 Introduction

This chapter will gives some of the Test case results. In this chapter Some Tests like Verification of alu instructions, load-store dependency verification, all instruction verification and conformal tests verification and their waveforms/log files are also provided.

3.2 Description

There are total of 30 arithmetic and logical instructions which include all R type instructions and some I type instructions. This test sequence verifies all instructions without redundancy. The tests pass fail status depends on BFM results in simulator. Running number of times in a loop will verifies different register combinations and different data of an instructions. The register contents can be seen in dump file and in waveforms also. The emulator dump contains instructions which is executed on core, simulator dump contains instructions executed on BFM.

3.3 Dump File For Alu Cyclic Random Testcase

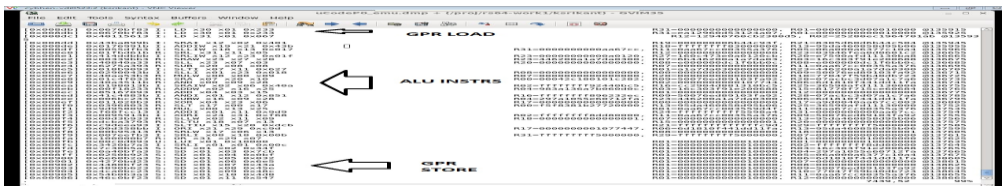

Fig.5 BFM dump file of alu cyclic random Sequence

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

Description

This test case main interest is verifying load store dependencies .The pattern followed here performing R type instructions and then load store operations and store-load operations. Now performing I type instructions and then load-store operations and store- load operations. In performing load -store operations verifying as address dependency load store and register dependency load-store operations.

3.4 DUT File For Load Store Dependency

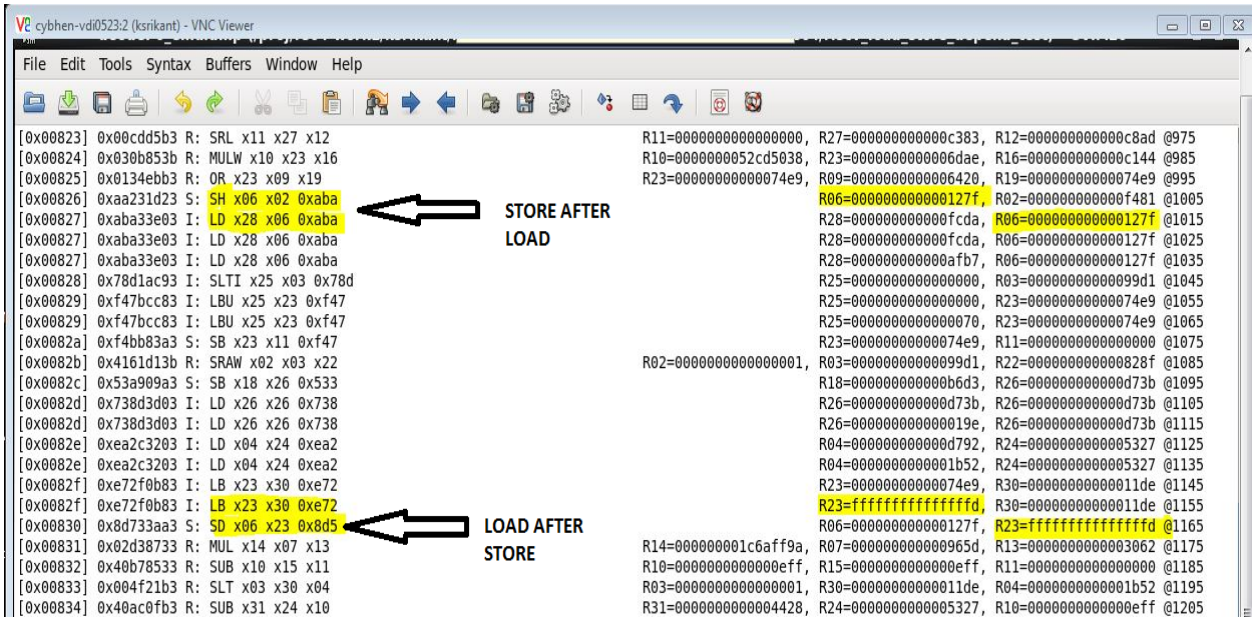

Fig.6 dump file of load store dependent Sequence

Description:

This test is to verify conformal test of ADD instruction.in conformal test input is known and expected output is also known. Hence memory is backdoor filled with input data and expected output data .Now performing the required operation by using data in memory. Conform the results by performing expected result and actual results SUB operation will conform the pass status.as subtraction result contains all zeros.

3.5 DUT dump file for conformal test

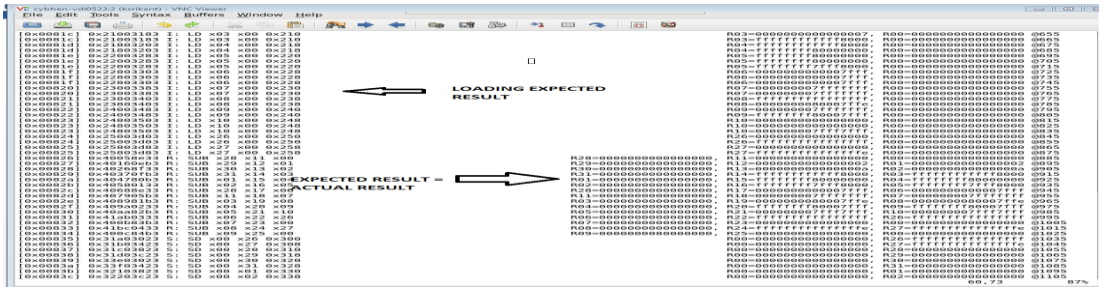

Fig.7 DUT dump file of conformal test

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

By this sequence total 51 instructions are verified directly. The pc change when JAL and JALR instructions are performed is shown in logfile.The sb type instruction are also verified by giving equal and unequal data, greater and lesser data. The pc change after performing SB type instructions also verified.

Description

This is conformal test verification. Here different combinations of data with expected output is given .after running this code on core if the result is expected data then the test is said to be passed and to know pass status and fail status used memory locations of 20000 and 30000 respectively.

3.6 DVE waveform of add.s pass

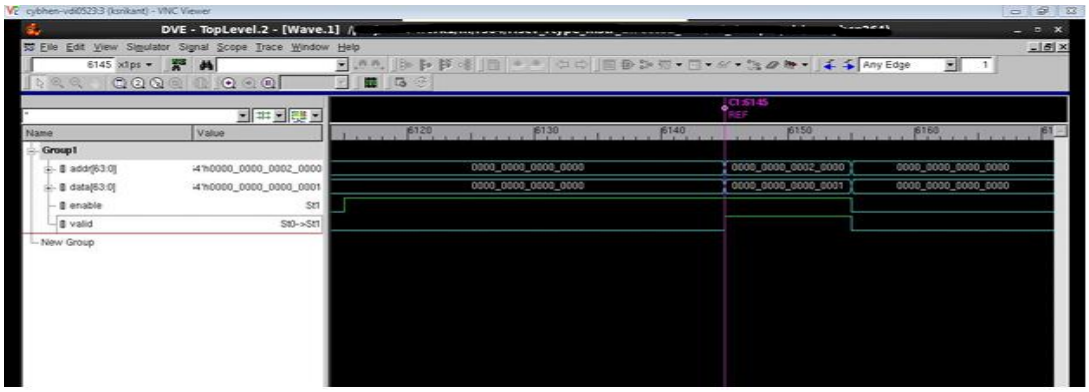

Fig.8 Dve of add.s pass

3.7 DUT file of add.s pass:

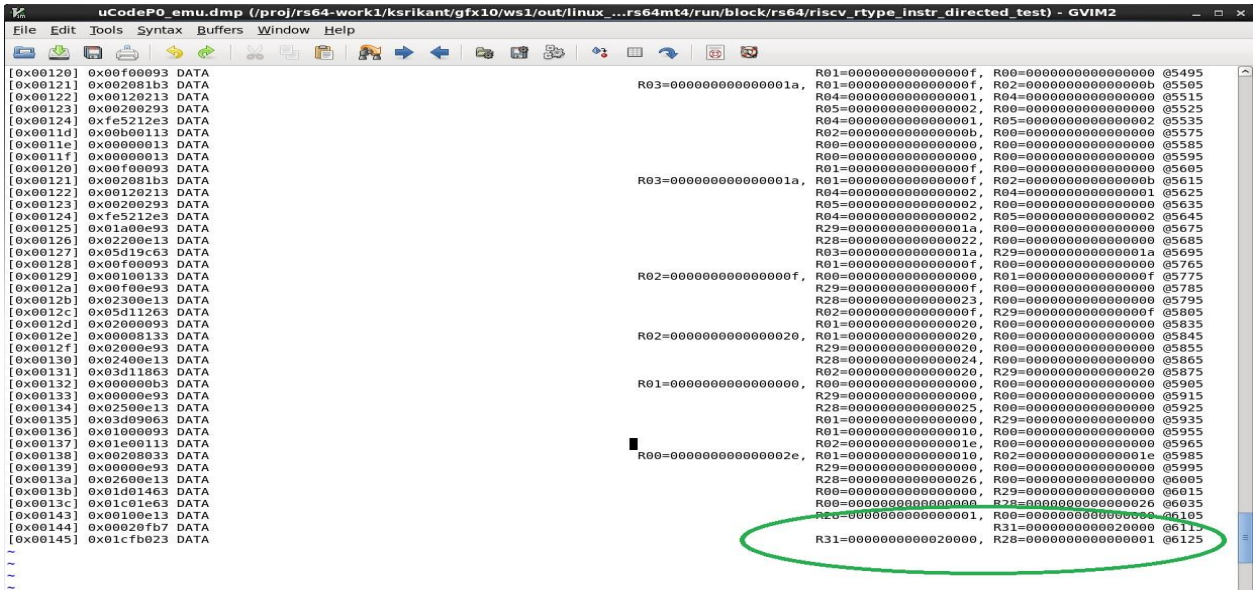

Fig.9 Sim file of add.s pass

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

3.8 DUT waveform of add.s fail

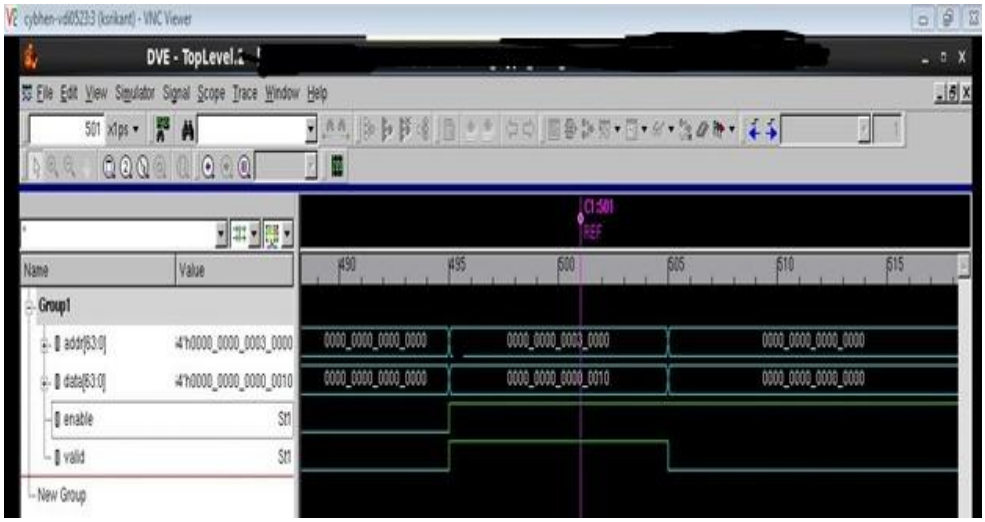

Fig.10 Dve of add.s fail

Summary:

In this chapter Different Test cases and their sequence of execution are shown. Results of some real time scenarios are also shown in this chapter. Waveforms and Log files are provided in this chapter for the respective tests and given their description also.

IV.CONCLUSION AND FUTURE SCOPE

4.1 Conclusion

RISCV is a completely open ISA that is freely available to academia and industry. A real ISA suitable for direct native hardware implementation, not just simulation or binary translation. Riscv that avoids over-architecting for a particular microarchitecture style (e.g., micro coded, in-order, decoupled, out-of-order) or implementation technology (e.g., full-custom, ASIC, FPGA), but which allows efficient implementation in any of these. In this project verified all instructions of riscv which supports integer type in UVM environment. Got knowledge on UVM verification environment and instructions of riscv.

The UVM Verification Methodology is used to verify the RTL behavior. Environment related to this is developed and verified it for different kinds of scenarios. The Verification environment required for the Riscv features verification has been developed and generated the required sequences. The waveforms had been analyzed to see if the RTL is working correctly or not.

4.2 Future Scope

We can verify the master DUT by adding additional blocks.

- This project can be further extended by adding instruction cache and data cache.

➢ This project can be further extended by adding branch prediction algorithm to achieve effective performance.

REFERENCES

[1.] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovi_c. The RISC-V instruction set manual, Volume I: Base user-level ISA version 2.0. Technical Report UCB/EECS- 2014-54, EECS Department, University of California, Berkeley, May 2014.

[2.] K. Diefendorff, P.K. Dubey, R. Hochsprung, and H. Scale. AltiVec extension to PowerPC accelerates media processing. IEEE Micro, 20(2):85 {95, 2000.

[3.] Waterman, Andrew; Lee, Yunsup; Patterson, David A.; Asanović, Krste. "The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA version 2.1". University of California, Berkeley. EECS-2016-118. Retrieved 22 July 2016.

[4.] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovi_c. The RISC-V instruction set manual, Volume I: Base user-level ISA. Technical Report UCB/EECS-2011-62,EECS Department, University of California, Berkeley, May 2011.

[5.] Marc Tremblay, Je_rey Chan, Shailender Chaudhry, Andrew W. Conigliaro, and Shing Sheung Tse. The MAJC architecture: A synthesis of parallelism and scalability. IEEE Micro, 20(6): 2000.