Design and Verification of a 16-bit RISC Processor using Universal Verification Methodology (UVM)

[1]Ajit Shridhar Gangad, [2]Prof.C.Prayline Rajabai
[1]M.Tech-VLSI Design, [2]Assistant Professor
School of Electronics Engineering
VIT University, Vellore, India

Abstract **- This paper presents Design of a 16-Bit RISC Processor supporting Arithmetic ,Logical, Data transfer, Branch instructions such as ADD , MUL , SUB , AND , OR , EXOR , EXNOR , RD , WR , BR , BRZ , NOT , NOP.RISC Processor supporting High Speed , Low Area, Low Power Uniform Carry Select Adder (UCSLA), High Speed 16-bit Hybrid Wallace Tree Multiplier .The Design is synthesized with 45nm library. Physical Design flow is performed with Cadence SoC Encounter. Verification environment is prepared by using Universal verification methodology(UVM) is most widely used methodology by verification industry word wide . Verification environment created in UVM which is reusable, efficient and well structured. The 16-bit RISC processor is a Design under test (DUT).The environment created in UVM is completely wrap a DUT. Assertion coverage is found to be 100% , Code Coverage consists of statement, Branch, Toggle, Expression coverage which is found to be 98.30%.Functional Coverage is obtained by writing cover-groups is found to be 99.87%.Overall coverage found to be 99.39%.**

Key Words **- System Verilog ,DUT, UVM, Functional Coverage , Assertion Based Verification ,RISC**

I.INTRODUCTION

Among all kinds of CPU in use today, the Reduced Instruction Set or RISC CPU has the majority market share. It is most commonly used in embedded systems, which are in almost every consumer products on the market. RISC CPUs are basic in nature and offers low-power consumption and small size. They are sometimes referred to as load-store processor because of the basic mechanics upon which it operates. The idea of RISC CPU is to reduce the complexity of the system and increase the speed. Any complex operation can be split into smaller chucks that can be calculated simultaneously in most cases. Other important features of the RISC CPU include uniform instruction coding, which allows faster coding. A good example is that the op-code is always in the same bit position in each instruction, which is always one word long. Another advantage is a homogeneous register set, which allows any register to be used in any context and simplify compiler design. Lastly, complex addressing modes are replaced by sequences of simple arithmetic instructions. The convenience of the RISC processor is a direct explanation why it dominates the CPU market.

II.ARCHITECTURE

Reduced instruction-Set computers (RISC) are designed to
have small set of instruction that executes in short clock cycles with small number of cycle per instruction. Machine consists of three functional units: processor ,controller ,memory as shown in Figure 1.
Following steps are performed by this architectures-
[1]fetching instruction from the SRAM
[2]Decoding
[3]Execution

The 16-bit RlSC processor having features such as 16-bit Uniform carry select adder (UCSLA) is designed by using single carry skip adder (CSKA) and binary to excess-1 converter (BEC) instead of using one set of ripple carry adder (RCA) to reduce area, power and delay. 16-bit High Speed Hybrid Wallace tree multiplier is configured with the help of novel compressor techniques such as 3:2 compressor, 4:2 compressor, 5:2 compressor.

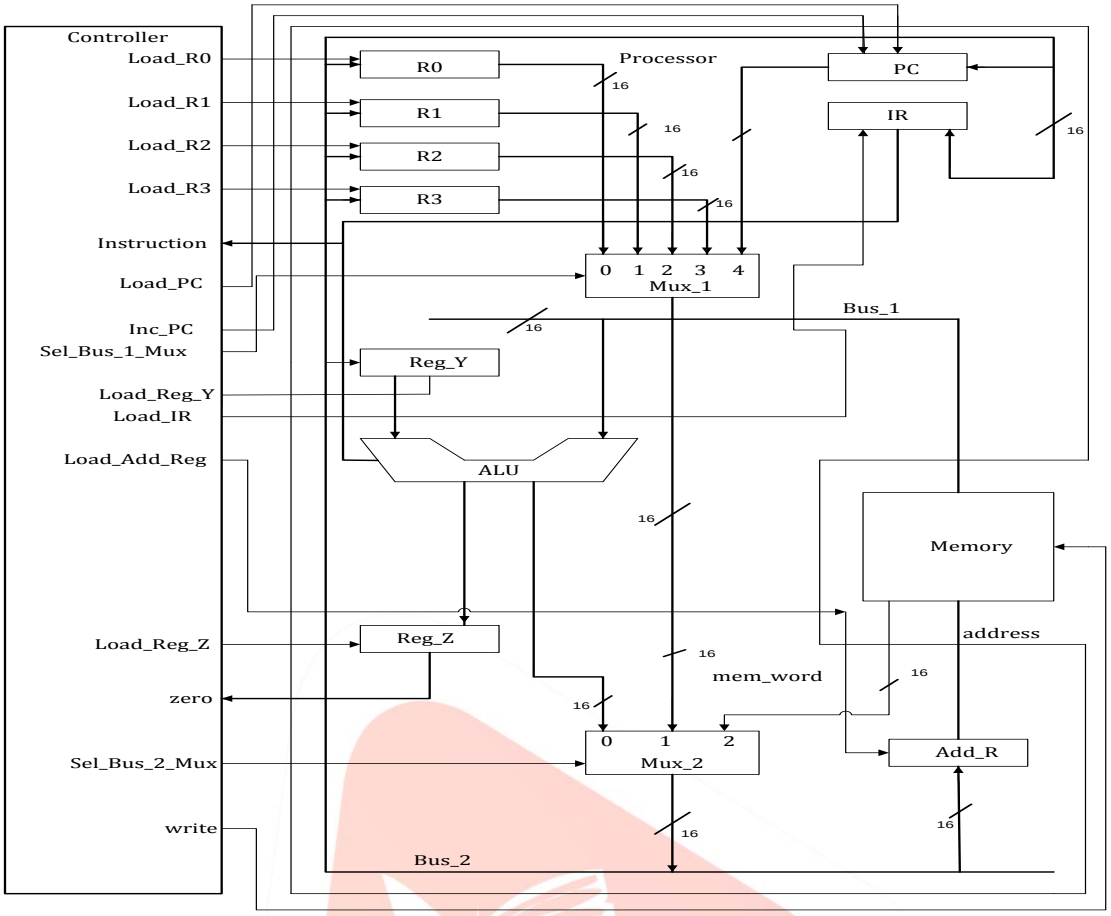

Figure 1. RISC Processor Architecture

A. processor

Processor includes registers, data paths ,control lines and ALU capable of performing arithmetic and logical operations on its operands ,subject to op-codes are hold in instruction register. A multiplexer Mux_1 determine source of data bound on Bus_1, second Mux_2 determine source of data for Bus_2.inputs to the Mux_1 are R0, R1, R2, R3, PC as shown in architecture .The contents of Bus_1 can be steered to the ALU, Mux_1 and memory unit .Thus, an instruction can be fetched from memory ,placed on Bus_2, and loaded into instruction register .A word of data can be fetched from memory ,and steered to GPIO or to the operand register (Reg_Y) prior to the operation of ALU .Result of ALU placed in Bus_2,loaded into a register and subsequently placed in the memory unit. A dedicated register (Reg_Z) holds a flag indicating that result of ALU operation is 0.

B. Controller

Following functions are performed by the control unit-

1. Specifying when to load specific register
2. Choose multiplexer among the two, depending upon the requirement of specific bus
3. Specifying when data to be loaded into the SRAM
4. Controlling all buses according to the requirement.

Timing of all activities are determined by controller .Following Table 1 shows the control signals and action corresponding to it.

Table 1 Control Signals and Actions

Control Signal	Action
Load_Add_Reg	Load the address register
Load_PC	Load Bus_2 to the PC
Load_IR	Load Bus_2 to the IR
Inc_PC	Increment PC
Sel_Bus_1_Mux	Select mux for Bus_1
Sel_Bus_2_Mux	Select mux for Bus_2
Load_R0	Load GPIO R0

Load_R1	Load GPIO R1
Load_R2	Load GPIO R2
Load_R3	Load GPIO R3
Load_Reg_Y	Load Bus_2 to Reg_Y
Load_Reg_Z	Stores the output of ALU to Reg_Z
Write	Memory write

C. ASM Charts

ASM charts in Figure 2 and Figure 3 shows the activities to be performed during execution of each instruction.

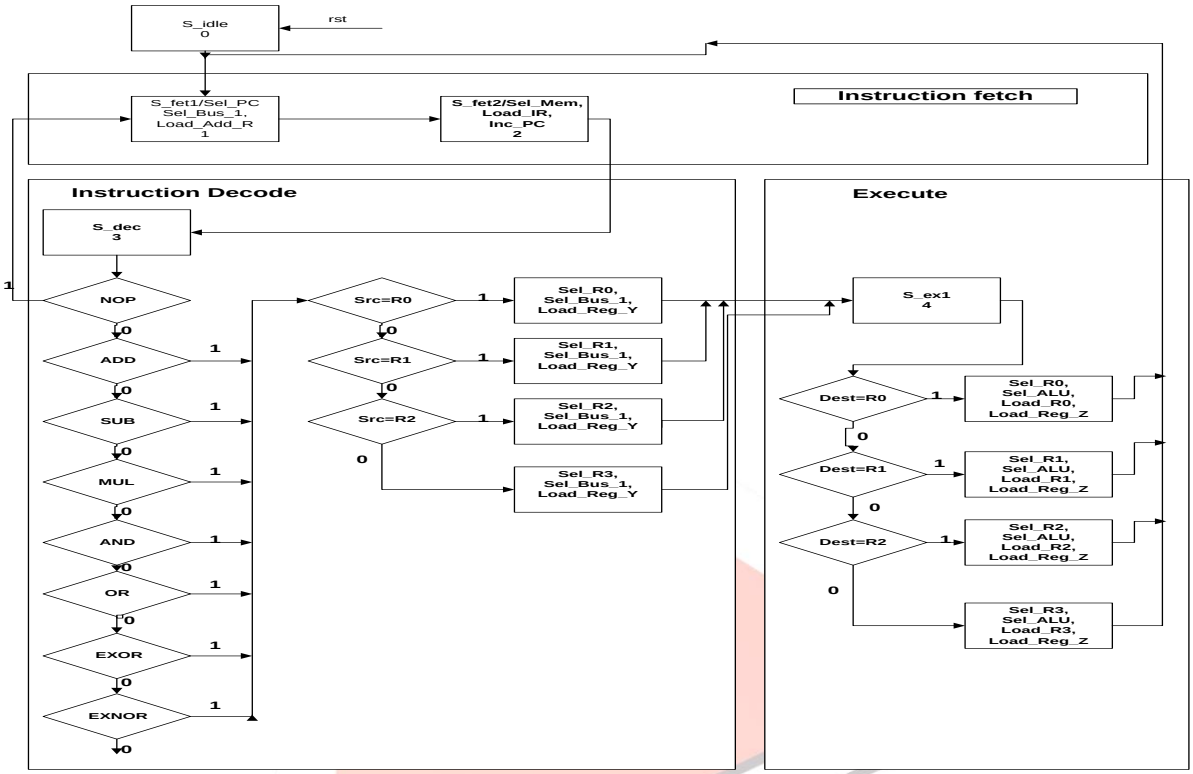

Figure 2. ASM Chart for Arithmatic and logical Instruction

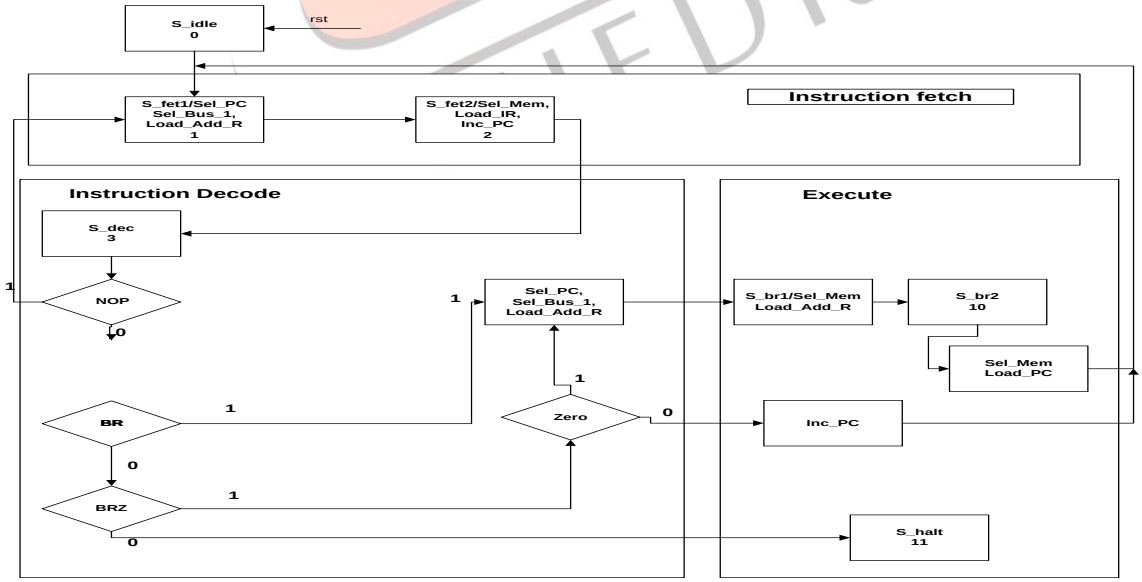

Figure 3. ASM Chart for Branch Instructions

D. INSTRUCTION SET

Instruction set of the RISC processor is divided into two categories short instruction and the long instruction .This can be explained as follows

1) Short instruction (1 byte instruction):: This consisting of most significant 4 bits as op-code and last 4 bit are used for specifying register as shown in Table 2

Table 2. short Instruction

Op-code				Source		Destination	
0	0	1	0	0	1	1	0

Instructions are

1. NOP-No any operation
2. ADD-Addition of source and destination register and storing data in to the destination register
3. SUB- Subtract of source register from destination register and storing data in to the destination register
4. AND/OR/EXOR/EXNOR-bitwise operation of source and destination register and stores data in destination register
5. NOT-bitwise negation performed

2) Long instruction(2 byte instruction):: This consisting of most significant 4 bits as opcode and afterward 4 bit are used for specifying register and last 8 bits specifies address of memory location as shown in Table 3

Table 3. long Instructions

Opcode				Source		destination	
0	1	1	0	0	1	Don't care	Don't care
Address							
0	0	0	1	1	1	0	1

Instructions are

1. RD-loads destination register with the address specified by second byte of instruction.
2. WR-write source register data into the address specified by second byte of instruction.
3. BR-branching occurs by addressing program counter to the SRAM memory
4. BRZ- branching occurs by addressing program counter to the SRAM memory if zero flag register asserted list of instruction to be executed as shown in Table 4.

Table 4. All Instructions details

Instruction	Opcode	src	dest	Action
NOP	0000	-	-	None
ADD	0001	src	dest	dest<=src+dest
SUB	0010	src	dest	dest<=src-dest
MUL	1001	src	dest	dest<=src*dest
AND	0011	src	dest	dest<=src&dest
NOT	0100	src	dest	dest<=~src
OR	1010	src	dest	dest<= src|dest
EXOR	1011	src	dest	dest<= src^dest
EXNOR	1100	src	dest	dest<=~(src^dest)
RD	0101	-	dest	deat<=memory[Add_R]
WR	0110	src	-	memory[Add_R]<=src
BR	0111	-	-	PC<=memory[Add_R]
BRZ	1000	-	-	PC<=memory[Add_R]
HALT	1111	-	-	Halt execution until reset

Table 5-Synthsis Results

Area of Design	34833 um2
Number of Gates in Design	9243
Leakage power with low power constraints	1098.874 nW
Dynamics power with low power constraints	866654.321 nW
Total power with low power constraints	867753.195 nW
Operating Frequency	200Mhz

III.PHYSICAL DESIGN

The Designed RISC Processor is synthesized with Cadence RTL Compiler. Synthesis Results are shown in Table 5 .

With help of SoC Encounter Physical Design Steps such as Floor-planning, Placement, Pin assignment, Routing are performed. Figure 4 and Figure 5 shows synthesized hardware and Physical design output respectively.

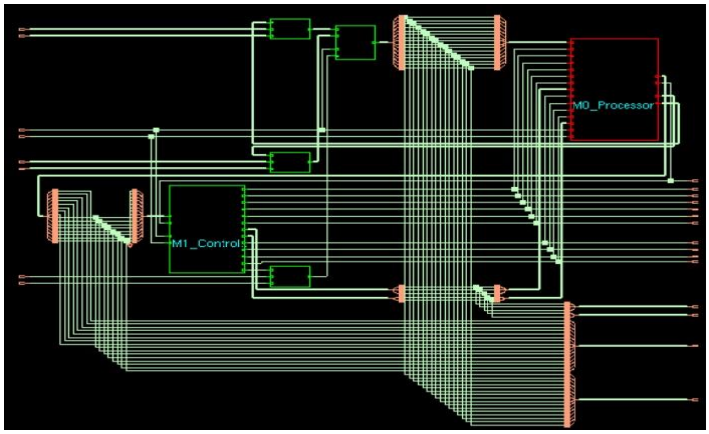

Figure 4. Synthesized Hardware

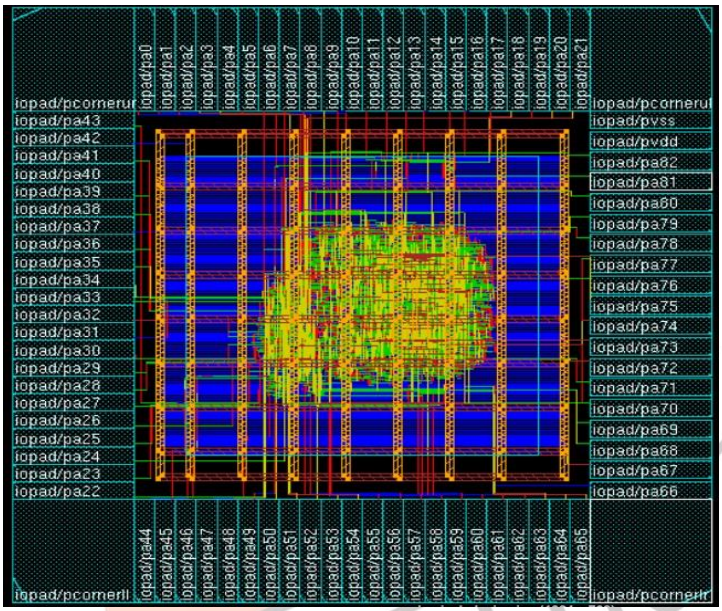

Figure 5. Physical Design Output

IV. UVM METHODOLOGY ARCHITECTURE

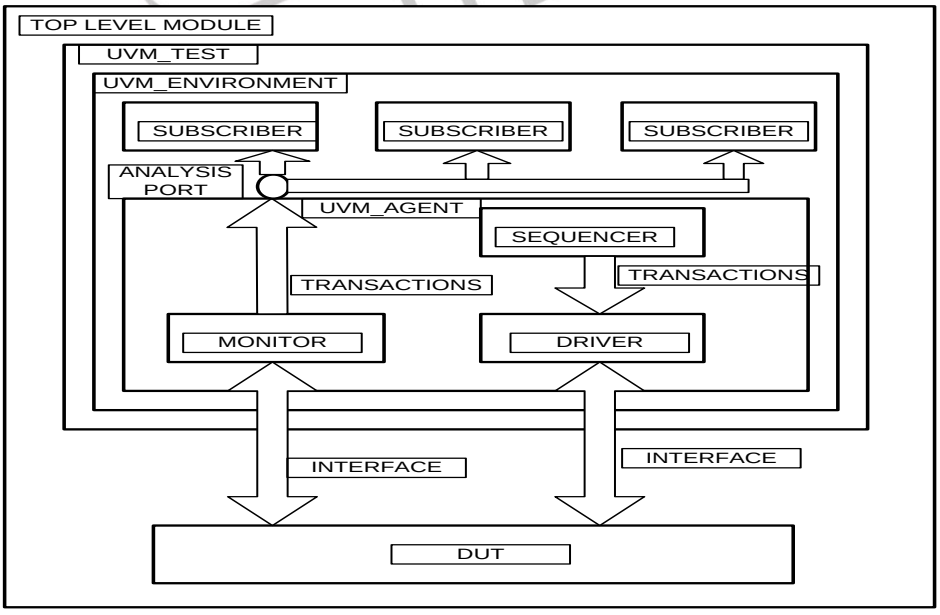

Figure 6. UVM Architecture

UVM Test Bench :
Test bench of UVM consisting of verification components which can be used to force stimulus to the Design under test(DUT) to check the 16-bit RISC Processor architecture. An UVM architecture is as shown in figure 6.

UVM Verification Components :

1]Design Under Test(DUT)-Design can be described with the help of hardware description language such as Verilog, VHDL which is intended to be verified. DUT describes whole implementation features and functionality of design.

2]Interface- Interface is a connection between DUT and verification environment (monitor and driver). There can be multiple interfaces depending upon number of agents. It is written in System Verilog. Interface is like a bus of wire which converts transactions to pin level data and vice-versa.

3]Virtual Interfaces- It separates abstract models from actual signals of the design. Virtual interface instance can be defined in multiple subprograms of verification environment. It dynamically controls the set of signals associated with subprogram which will pass same set of data to all verification components.

3]Transactions- uvm_transaction is derived from uvm_sequence_item class. In test many data items need to be generated and send to the DUT via driver. Data item fields are randomized using System Verilog constraints. With many number of different constraints different test cases can be created. Communication between sequencer to driver , monitor to subscriber is through transactions

4]Agents –uvm_agent class is extended from a uvm_components base class .UVM agent consists of Monitor , sequencer ,Driver. Communication between driver to DUT, DUT to monitor is through pin level interface. Generally DUTs have number of interfaces, each of which has their own protocol. UVM agent provides designer to generate and monitor signal level transactions.

5] Sequencer And Sequence – A sequence class is derived from uvm_sequence class.uvm_sequencer class is extended from uvm_components. uvm_sequencer controls the flow of transaction generation. uvm_sequencer does the generation of this sequence of transaction .For different test cases we can write different sequences with the help of constraint randomization. Each sequence is called by sequencer which takes each sequence and converts it into transaction. Further, driver which drives transaction to DUT.

6]Driver – Driver class is formed by extending uvm_driver base class. Driver takes the transactions from the sequencer by using seq_item_port . These transactions will be driven to DUT through interface. Timing information can be written in run phase with the help of different tasks such as resetting of DUT and setting configuration of DUT.. An instance of the driver class is created in The agent class and the sequencer is connected to it.

7]Monitor- Monitor class is derived from the uvm_monitor base class. Instance of monitor is created in agent and connected to the DUT through interface. Basically its functionality is to collects data from interface and convert it into transactions .Converted transactions can be provided to subscriber such as scoreboard through Analysis port.

8]Scoreboard –Scoreboard class is derived from uvm_scoreboard base class. Scoreboard has 2 analysis imports. One is used to for getting the packets from reference model and other from the monitor analysis port transactions. Packets comparison is carried out with compare function of transaction class and if doesn't match error signal is asserted. Covergroup is written with different coverpoints according to design features. Covergroup is used to define Functional coverage of design.

9]Environment –Environment class is formed with the extension of uvm_env class which is used to implement verification environments of UVM. Simulation steps such as building components ,connection between the components , starting the components need to be specified, which is possible through uvm_env base class. It has methods to formalize phases of simulation. All methods are virtual which are extended from environment. virtual sequencer that is used to run sequences . environment is the top level of the class based part of the testbench.

10]Testcases – Different test case scenarios for design are defined in the uvm_test class. Instance of environment is defined in Test case which defines environment object and defines specific functionality. Physical interface in top module is connected to virtual interface of environment in Test cases.

11]Top Module –Physical interface is instantiated in this top module. Connection between virtual interface and physical interface is defined with configuration database settings. DUT is instantiated and hooked up with interface .Clock generator is also defined here .With the help of run_test() method we can call all possible test cases . The test name can be passed as a command line argument during simulation which is having more precedence.

V.UVM CLASS HIERARCHY

Different components in UVM are derived from either uvm_object or uvm_components as shown in Figure 7.

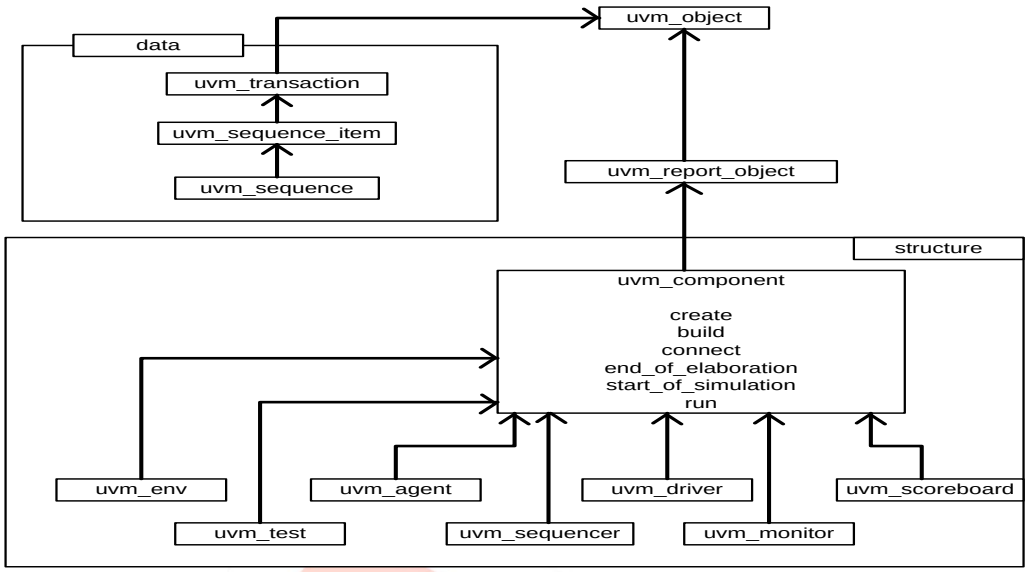

Figure 7. UVM Hierarchy

1]Build Phase: This phase is used to build various ports/exports/ class objects by using constructor.
2]Connect Phase: This phase connects all port/exports of various objects/components.
3]End of Elaboration: Configuration of different components is done in this phase.
4]Start of Simulation : To print various messages and topology of environments.
5]Run Phase:This phase is executed by calling task where multiple threads are concurrently called. Timing delays can be inserted in this phase only.

VI.VERICATION PLAN

The verification plan defines what exactly needs to be verified. The 16-bit RISC processor is divided into three sub modules such as Control Unit, Memory Unit, Processing Unit. Different test case scenarios for these three sub modules are written in different sequences and are as follows:

A] Test case scenarios for Control Unit:
1]Reset Condition-it should enter to S_idle state
2]Instruction fetch cycle check -It should enter to S_fet1 & S_fet2 state respectively
3]Instruction fetch cycle check -Whether appropriate Control Signal asserted or not
4]Instruction Decode Cycle Check -It should enter to S_dec state
5]Instruction Decode Cycle Check -Selecting mentioned source register
6]Instruction Decode Cycle Whether appropriate Control Signal asserted or not
7]Instruction Execute Cycle Check -it should enter to S_ex1 state
8]Instruction Execute Cycle Check -Selecting mentioned Destination register
9]Instruction Execute Cycle Check -Whether appropriate Control Signal asserted or not
10]Instruction fetch cycle check,Instruction decode cycle check,Instrction execute cycle check for ADD,SUB,MUL,AND,NOT,OR,XOR,XNOR,RD,WR,BR, RZ,NOP instructions

B] Test case scenarios for Processing Unit:
1]Loading data from Bus to R0,R1,R2,R3 ,PC,IR ,Reg_Y , Add_R,Reg_Z after Load_R0, Load_R1, Load_R2 , Load_R3, Load_PC, Load_IR, Load_Y,Load_Add_R , Load_Reg_Z signals assertions
2]Mux1 should change the channels according to Sel_Bus_Mux1 assertions
3]Mux2 should change the channels according to Sel_Bus_Mux2 assertions
4]ALU- Perform functionality according to opcode given to it for instructions (ADD,MUL,SUB,AND,OR,EXOR ,EXNOR ,EXNOR,NOT,NOP)
5]ALU-Alu zero flag condition check
6]PC Branching check –instructions BR,BRZ(Branch if alu output is zero)

C] Test case scenarios for Memory Unit:
1]A simple testcase to perform one write and read
2] Testcase to write 256 successive locations
3]Testcase is to demonstrate the simple random
4]Single Write Condition
5]Single Read operation
6] Multiple Write operation with delay at different conditions
7] Multiple Write operation without delay at different conditions
8]Multiple read Write operation with delay at different conditions
9] Multiple read Write operation without delay at different conditions.

VII.RESULTS AND DISCUSSIONS

For 16-bit RISC Processor, Instructions and data to be loaded as operand is specified in SRAM memory. One by one instruction is called from the memory and executed in fetch1,fetch2, decode, execute1, execute2 cycles depending upon the type of instruction. Following Figure 8 ,9 ,10 shows the Modelsim Simulation output for MUL, RD, BR instructions respectively.

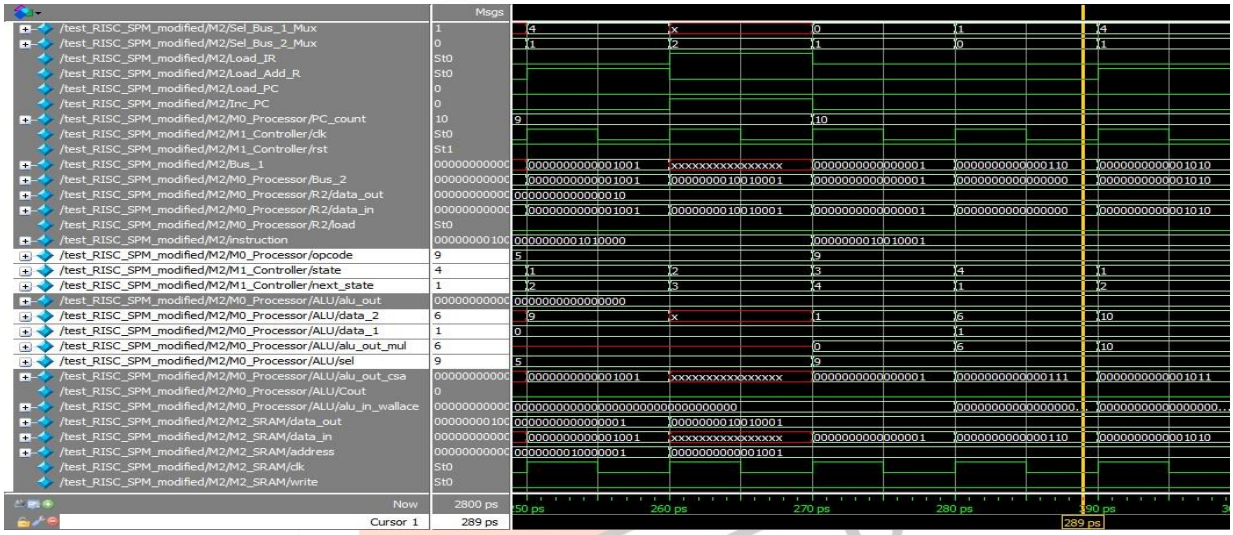

Figure 7. MUL Instruction Output

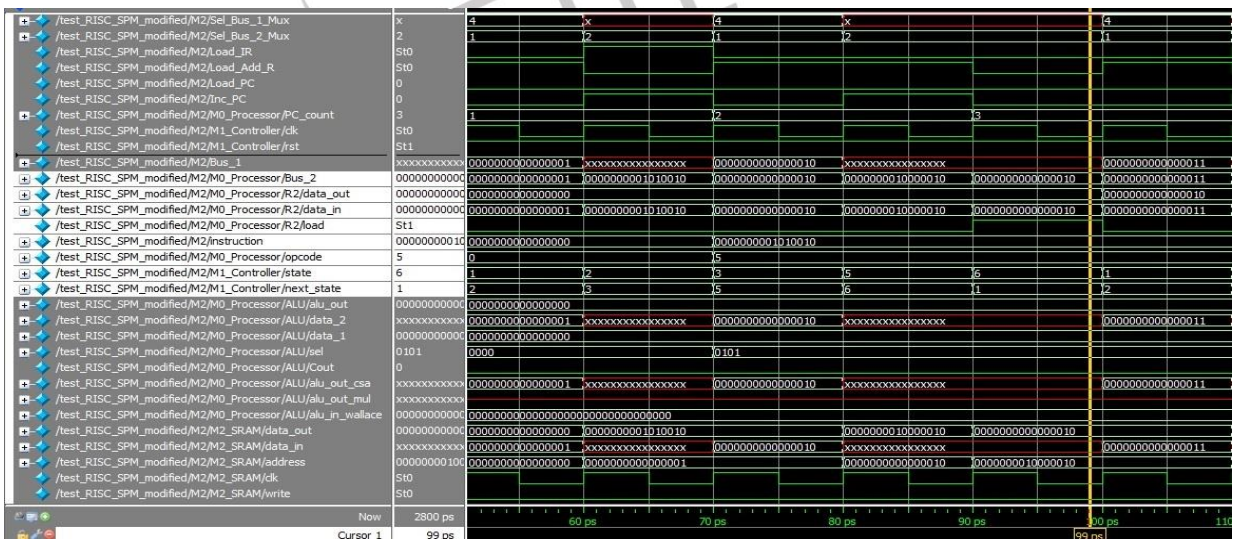

Figure 8.RD Instruction Output

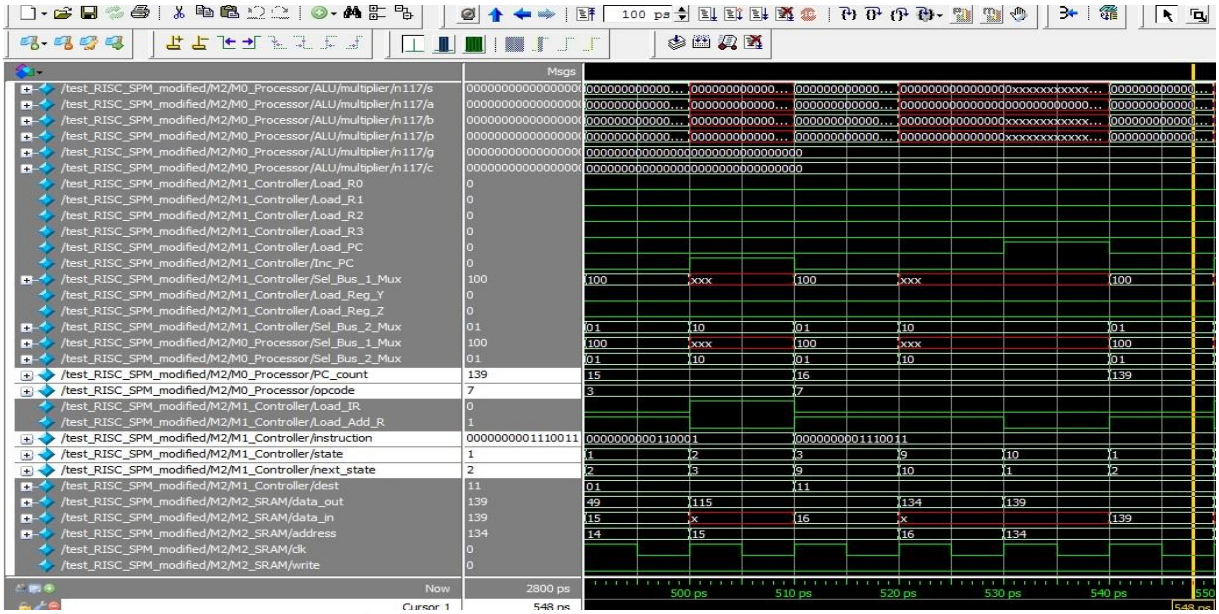

Figure 9.BR Instruction Output

Figure 10-256 Location Read Write Operation

For verification point of view, design is divided into three blocks Memory Unit, Control Unit and Processing Unit. Covergroup is written with all possible cover points to get functional coverage. Cross coverage also introduced to cover shared coverpoints. Code coverage consisting of Block coverage, Expression coverage, Toggle coverage and FSM coverage is maximized with various test case scenarios.

1]Memory Unit-First of all for Memory unit earlier mentioned test cases passed. Figure 10 shows questasim simulation output for 256 location write and read operations.

Questasim report shown in Figure 11 shows that, for Memory Unit 100% assertion passed,96 % Code coverage achieved and 100% functional coverage achieved with cover group and cross cover points.

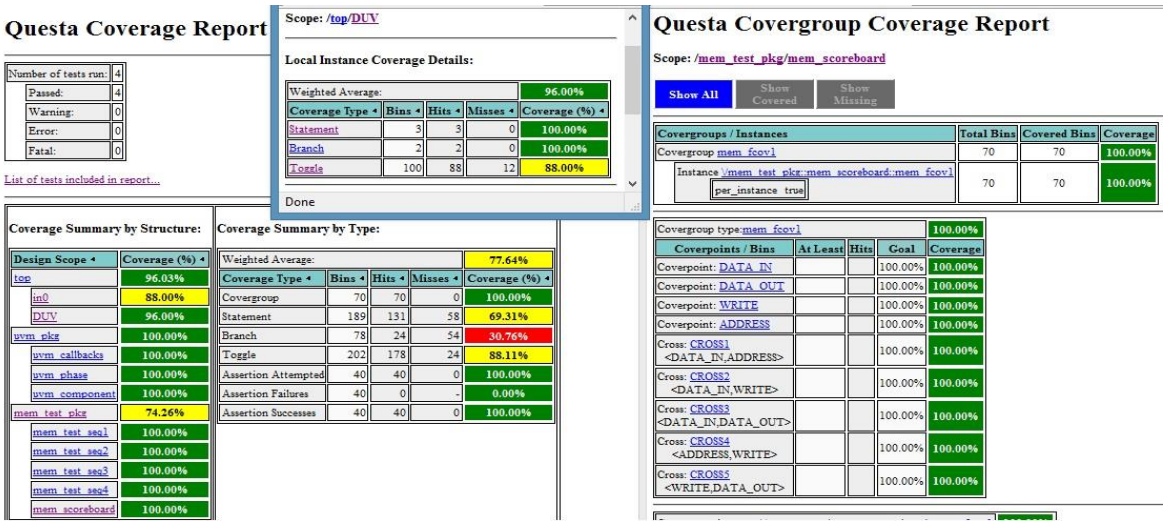

Figure 11-Coverage Report for Memory Unit

2]Control Unit-

Testcase scenarios mentioned in the verification plan of Control Unit are passed. Questasim Simulation Output for ADD Instruction fetch,decode ,execute cycle is as shown in Figure 12.

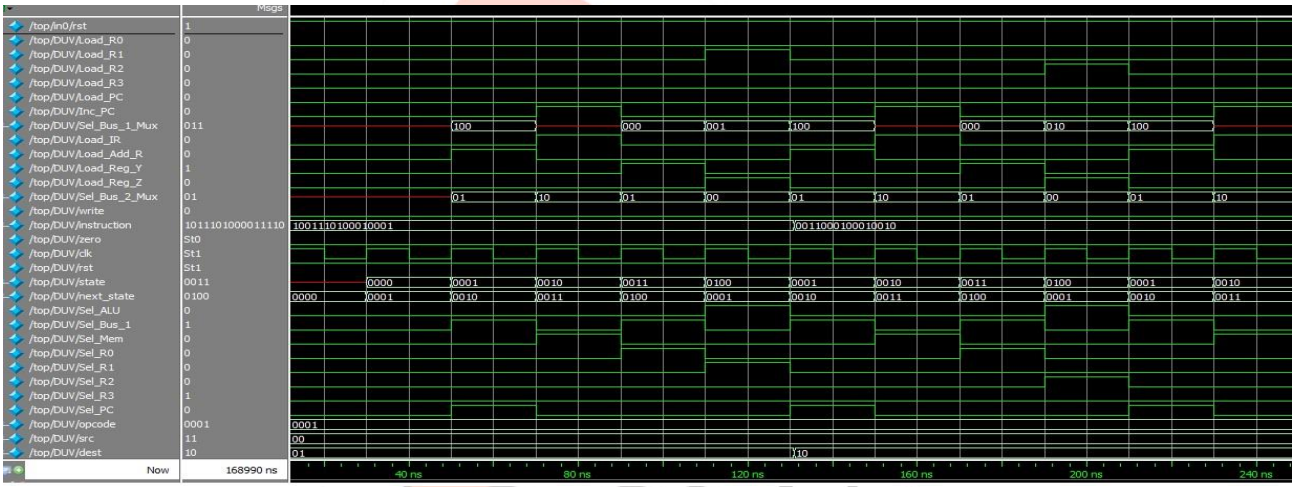

Figure 12-ADD Instruction testcase Output

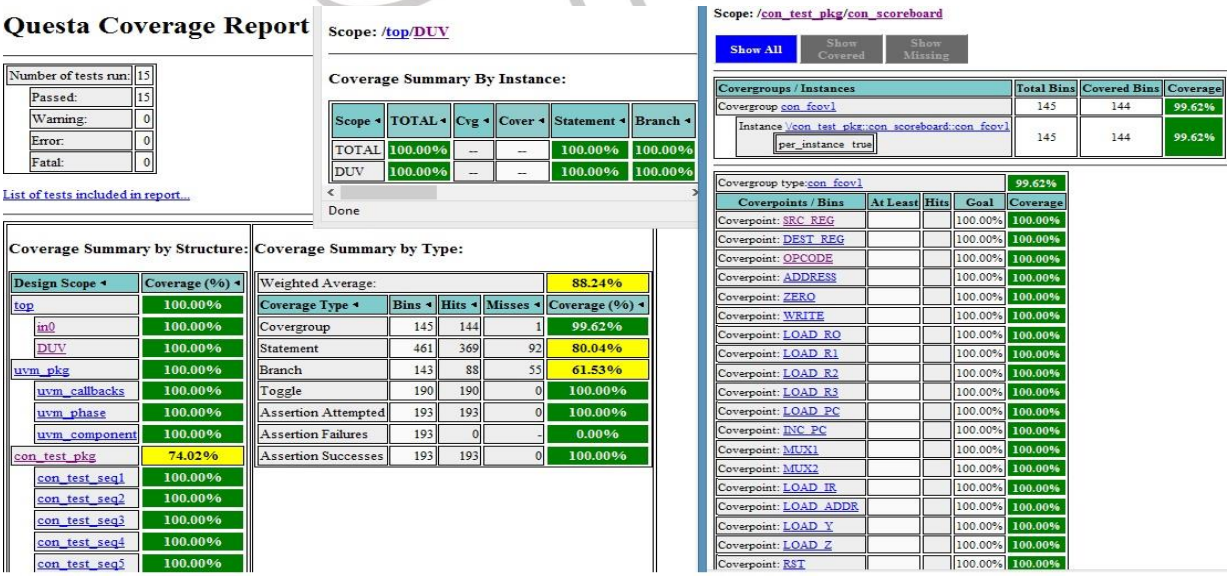

Figure 13- Coverage Report for Control Unit

Questasim report shown in Figure 13 shows that, for Control Unit 100% assertion passed,100 % Code coverage achieved and 99.62% functional coverage achieved with cover group and cross cover points.

3]Processing Unit-

Testcase scenarios mentioned in the verification plan of Processing Unit are passed. Questasim Simulation Output for AND instruction is as shown in Figure 14

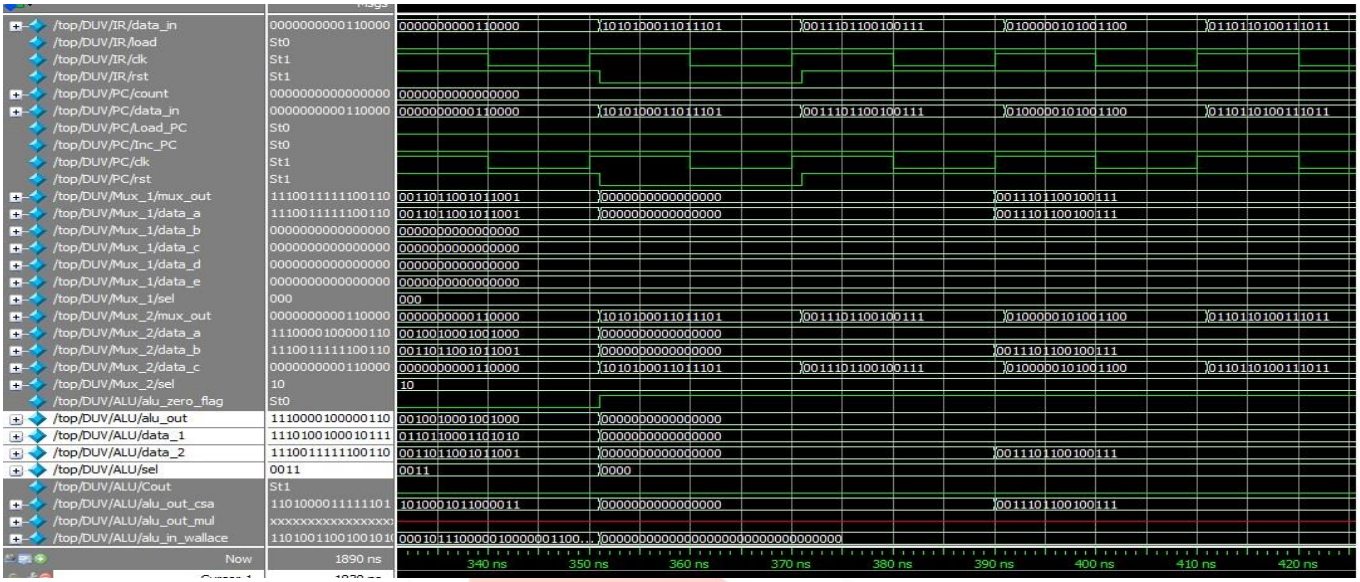

Figure 14-AND Instruction Testcase Output

Questasim report shown on Figure 15 shows that, for Processing Unit 100% assertion passed,98.90 % Code coverage achieved and 100% functional coverage achieved with cover group and cross cover points.

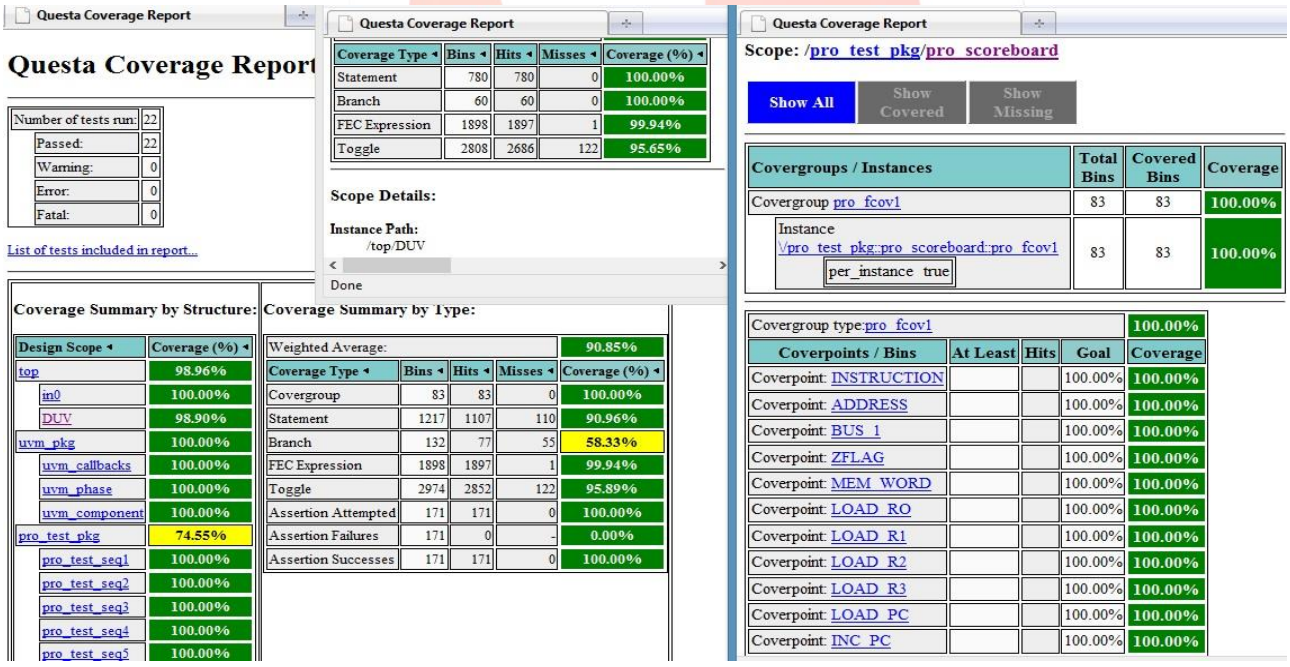

Figure 15- Coverage Report for Processing Unit

Thus for 16-bit RISC processor on average 100 % assertions are completed, 98.3% Code Coverage achieved and 99.87 % Functional Coverage achieved.

VIII.ACKNOWLEDGEMENT

I would like to thank Prof.Prayline Rajabai for her precious and valuable guidance and also VLSI lab, VIT University for Cadence RTL compiler, SoC Encounter, NC Launch software support.

IX.REFERENCES

[1] Seung Pyo Jung, Jingzhe Xu, Donghoon Lee, Kang-joo Kim, Koon-shik Cho,Ju Sung Park,"Design & Verification of 16 Bit RISC Processor",2008,IEEE.
[2] Pravin S. Mane, Indra Gupta, M. K. Vasantha,"Implementation of RISC Processor on FPGA",2006,IEEE
[3] Mamun Bin Ibne Reaz, Md. Shabiul Islam, Mohd. S. Sulaiman," A Single Clock Cycle MIPS RISC Processor Design using VHDL".
[4] HE Jing-yu, LI Li-li, ZHU Yan-chao, YANG Wen-tao, and YANG Jian-hong, "Multiply-Accumulator Using Modified Booth Encoders Designed for Application in 16-bit RISC Processor" ,2013 ,IMSNA.
[5] Jinde Vijay Kumar, Boya Nagaraju, Chinthakunta Swapna and Thogata Ramanjappa ,"Design and Development of FPGA Based Low Power Pipelined 64-Bit RISe Processor with Double Precision Floating Point Unit",2014.
[6] C.Vinoth, V. S. Kanchana Bhaaskaran, B. Brindha, S. Sakthikumaran, V. Kavinilavu, B. Bhaskar , M. Kanagasabapathy and B. Sharath"A Novel Low Power and High Speed Wallace Tree Multiplier for RISC Processor", 2011,IEEE
[7] P.Sadhasivam, Dr.M.Manikandan,"Low Area and High Speed Confined Multiplier using Multiplexer based Full Adder", IEEE,2014
[8] Haihua Shen, Heng Zhang, Tong Xu, "Verification of a Configurable Processor Core for System-on-a-Chip Designs"
[9] K. Jancy Bery , A. Beno,"Design of 64 Bit UCSLA for Low Power VLSI Application", International Journal of Scientific & Engineering Research, Volume 4, Issue 5, May-2013
[10] System Verilog for Verification by Chris Spear
[11] Learning UVM methodology - www.testbench.in
[12]Verification Academy Cookbooks for UVM and System Verilog -www.verificationacademy.com

Ajit Gangad was born in Maharashtra,India, in 1990. He received the B.E.Degree in Electronics and Telecommunication Engineering from Vidya Pratishthan's College of Engineering Baramati, Pune University, Maharashtra, India in 2012. He is currently pursuing M.Tech in VLSI Design in Vellore Institute of Technology (VIT) University, Tamil Nadu, India. His area of interest includes Modeling and Simulation of Nanoscale Devices-Multigate FETs, Digital system design, RTL Design with Verilog HDL,RTL verification using UVM.

Prof.C.Prayline Rajabai received M.E. degree in the Faculty of Information and Communication Engineering from Anna University, Chennai, India in 2014. She received B.E. degree in Electronics and Communication Engineering from Madurai Kamaraj University, Madurai, India in 2003. Currently, she is working as Assistant Professor in the School of Electronics, VIT University, Vellore, India. Her research interests include FPGA/ASIC Implementation of video compression and encryption algorithms and video processing.