AN INTRODUCTION TO VLSI
ECE429, SPRING 2014

PROJECT PROGRESS REPORT

DESIGN AND SYNTHESIS OF CENTRAL PROCESSING UNITS

NAME: VENKATA SAI SARATH MAREPALLI
CWID: A20319735
ECE 429 – SECTION 03

SUBMISSION DATE: 04-14-2014
TEACHING ASSISTANT: Mr. NAVEL GUPTE

OBJECTIVE:

The main aim of this project is to introduce the concepts of VLSI design including combinational and sequential circuit design, standard cell based design flow, and design validation and verification. This is done by constructing a 32-bit central processing unit(CPU) in verilog.

INTRODUCTION:

The project is divided into two parts first a 8bit CPU design is provided along with the project guidance file and then the design is then developed to 32-bit level. In this initial project progress report details about the 8-bit CPU are specified and the work completed till now.

The 8-bit CPU model is designed using Verilog (Hardware Descriptive Language), the implementation of this design is tested using a test program given in the guide with the specified inputs.

After the verification of the 8-bit CPU function we then perform a design synthesis by following standard cell based design flow by using Synopsys Design Compiler present in Cadence Virtuoso tool.

THEORY:

a) Verilog:

Verilog is the first modern hardware descriptive language, standardized as IEEE 1364, is a Hardware Descriptive Language used to model electronic systems. Verilog is case sensitive and has a basic preprocessor, and uses similar control flow words like while, if/else, for etc. like C-language. There are two assignment operators '=' (blocking assignment) and '<='(non-blocking assignment). The non blocking assignment is used to describe a state-machine update without a need of declaration. We come across various concepts in the Verilog in the coding of the 8-bit CPU.

b) Cadence Virtuoso Platform

Cadence Virtuoso is one of the EDA(Electronic Design Automation) tool used for designing integrated circuits. This platform offers all the parameters required for designing the Integrated Circuits in a real time scenario. The library we use in cadence virtuoso is FREEPDK45 which has all the basic building blocks required for designing a integrated circuit. We can even export the design done to a code format which can be used for analysis of the circuit.

c) Standard Cell ASIC Process:

A standard cell is logic gate of fabrication process and it contains different kinds of logic gates of different sizes. As the layout of standard cell is known we do not have to draw the whole layout by hand but rather use a Hardware description of the layout and the synthesis automatically designs the layout. This process includes two steps they are logic synthesis and physical design. In logical synthesis the code is written n hardware descriptive languages like Verilog or VHDL and then a net list is created with the logic in it. Then in the next step the physical design of the net list is converted into layout by choosing the surfaces and sizes of the transistors from the things available in the library.

d) Central Processing Unit(CPU):

The CPU is the heart of the modern day computer it contains various modules some of them are cache,memory,ALU,control unit etc which are used to perform any kind of operation that a CPU is designed to make.

I) Arithmetic Logic Unit(ALU):

The ALU is the brain of the CPU which performs all the logical functions and

controlling the data flow, in this design the ALU takes a 8-bit data and the computes the action which is intended to perform by following certain principles and detecting the type of operation that it should perform, and sends the 8-bit result out.

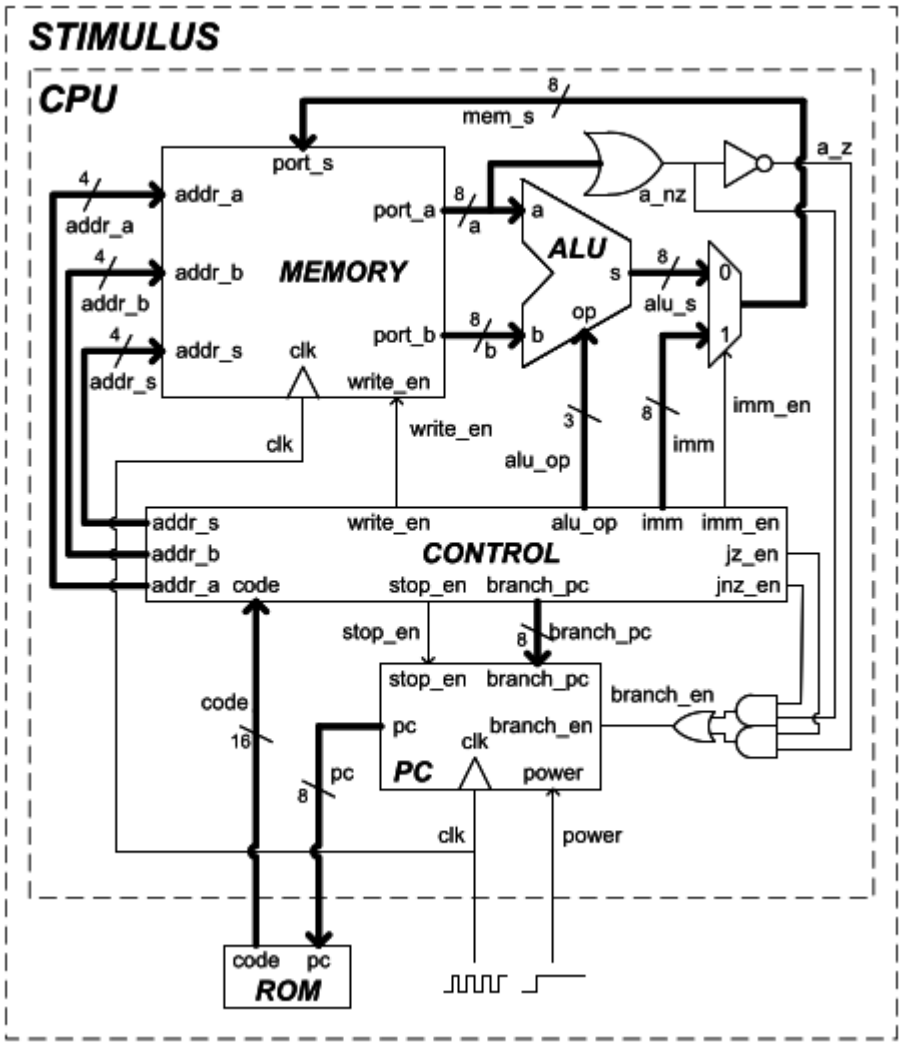


Fig: 8-bit CPU Implementation

II) Memory:

The memory in this project is very confined when compared to the original memory module, it contains 16 words with 8bits each. The value of each word is stored in 8 flipflops, and the memory is named as @0,@1,@2,@3,...@15 in our memory implementation. The memory words can be accessed by using 0000,0001,0010,.....,1111 for @0,@1,@2,.....,@15 respectively, which are the addresses for the memory implemented in this design. This is obtained by designing a 8bit 16 to 1 multiplexer. Which is shown in the appendix section.

III) Program Counter:

CPU needs to follow some order so that in fetches the next instruction after the first instruction is completed so to get this functionality we use a program counter, we use a 8bit program counter so the at most number of instructions accessed is 256.

IV) Instruction Set:

In this design we are implementing a small subset of the actual instruction set which contains like 8 to 10 type of instruction the instruction set implemented in this design is shown in a table in the appendix section below.

V) Program Storage:

The program that is to be implemented in this design is converted to binary code and then converted to HEX code and stores in a file called code.hex where all the instructions that are needed to be executed are written here and is placed in the same folder as the main and the test bench file, In the test bench file we provide the information about the whereabouts of this HEX file so when the program searched for the instructions it just goes to the hex file and fetches the instructions that are to be executed.

VI) Control Unit:

In this design we control the design flow by using a set of controls like clock (clk) which is used for the timely processing of instructions, and some other controls like write_en,imm_en,jz_en,jnz_en etc. so for example for the ALU output to be stored the addr_s and imm sets both write_en and imm_en to 1.

VII) Test Bench:

Test bench is a code written in verilog to test the design implemented, this looks similar to the regular verilog modules but the testbench code doesnot have any ports that is the main difference between regular modules and a test bench, in this design the test bench is cpu8_test.v which is provided along with the guide file provided.

IMPLEMENTATION:

The project guide provides the cpu8.v and cpu8_test.v files we take that files and compile it in Modelsim software, and the functional validation of this particular 8-bit CPU design is done by implementing the following model.

The project guide states that we need to multiply the last two digits of the CWID (in this case it is 35) and 2 where 35 is placed in memory location @2 and 2 is placed in location @3 and the result is stored in @6, later the product is subtracted from @6 and returns to zero, for this logic implementation we follow the below code.

```
0:  loadi @0 , 0 ;
1:  loadi @1 , 1 ;
2:  loadi @2 , 49 ;
3:  loadi @3 , 5 ;
4:  loadi @6 , 0 ;
5:  add @4 , @2 , @0 ;
```

```
6:  add @5 , @3 , @0 ;
7:  and @7 , @4 , @1 ;
8:  jz @7 , 10 ;
9:  add @6 , @6 , @5 ;
10:  srl @4 , @4 , @1 ;
11:  sll @5 , @5 , @1 ;
12:  jnz @4 , 7 ;
13:  add @5 , @5 , @0 ;
14:  add @6 , @6 , @0 ;
15:  add @4 , @3 , @0 ;
16:  add @5 , @2 , @0 ;
17:  and @7 , @4 , @1 ;
18:  jz @7 , 20 ;
19:  sub @6 , @6 ,@5 ;
20:  srl @4 , @4 , @1 ;
21:  sll @5, @5, @1;
22:  jnz @4, 17;
23:  add @5, @5,@0;
24:  add @6, @6, @0;
25:  stop
```

I converted the above program into machine code as directed by the guide, and saved in a HEX file which is used by the test bench and perform the execution. After verifying the result we migrate the verilog design files along with the program code (HEX) file to the linux environment to perform the design verification by following the Tutorial IV. First we check the verilog file by compiling it in the gcc compiler and then the design verification process is done and also the formal verification of the design is done all the screenshots are shown in the appendix section below.

INTERPRETATION:

1) A processor must provide all the arithmetic logical and control functions and only when the processor would be widely accepted by the end-user. When a processor is all setup to function the clk is set to start and enable input to the processor is set to '1' at the rising edge of the clock and then the servicing of the instruction starts by initializing the PC(Program Counter) to the first location of the instruction. Each instruction contains a opcode in the first 4 bits which decides the type of operation that needs to be performed by the CPU. For example when a CPU gets the opcode of a load that is “1010” it then gets the address to which it has to load the Imm(Immediate) value and then stores the value in that particular memory location, the particular memory location is selected by the multiplexer and decoder combination given in the code and after executing one instruction the PC value is incremented so that it points the next instruction in the code and now this is called as NPC(new PC) . Similarly when the code detects the opcode of an ADD function then it will take the address locations where the numbers are present and then the numbers in the source locations are taken to the adder, the

addition is performed and the result is again stored in the destination memory address.

2) For the evaluation of the design here in this case we execute a logic that is multiplying two numbers and subtracting the product so that the final result is zero. When the program is completed running, we check the final memory address where the result should be zero and then by verification we come to know that the design is perfect. When the two numbers are being multiplied we see that the instructions are repeated because of the presence of a conditional loop in the code and once the the loop is passed we get to see the product and later we can see the product being subtracted from the result and making it zero.

3) To evaluate the functional correctness of the design we now take the design to a linux environment and we run a verilog simulation on it as initial setup and check the results with the simulated results this is done by the following code.

Verilog cpu8test.v cpu8.v

after this we open simivison tool present in the Cadence Virtuoso from the terminal and add all the inputs and outputs to the graph and check the results plotted are true.
Then we do the design synthesis by importing libraries and required data by

cp /import/scripts/FreePDK45_2011/FreePDK45/osu_soc/flow_ece429/* .

then the verification is done at three stages to verify the perfectness of the design and in each case the result is compared to the previous one to make sure that the results obtained are true. And as a final step we do the equivalence check with the Formal ESP tool . The screenshots of each case are in the appendix section below.

CONCLUSION:

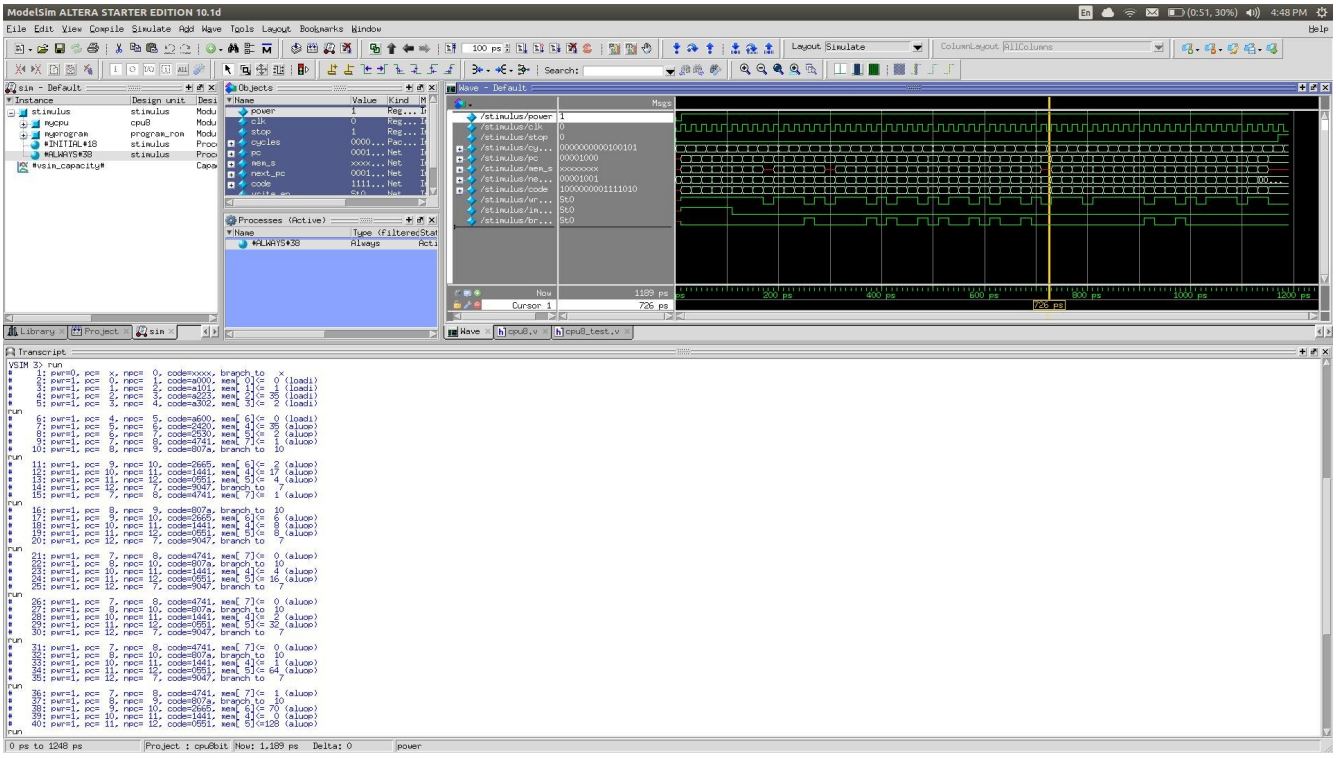
Up until now we have simulated the 8-bit CPU code provided to us along with the guidance material and verified its behavioral and functional exactness, in the next step we develop the code of the 8 bit CPU to 32-bit level and run the similar tests to verify the perfectness of the design, the main aim of this project is to introduce the concepts of VLSI design which is satisfied by this project. The screen shots for the Machine Code, Modelsim Simulation and functional verification are in the appendix section below.

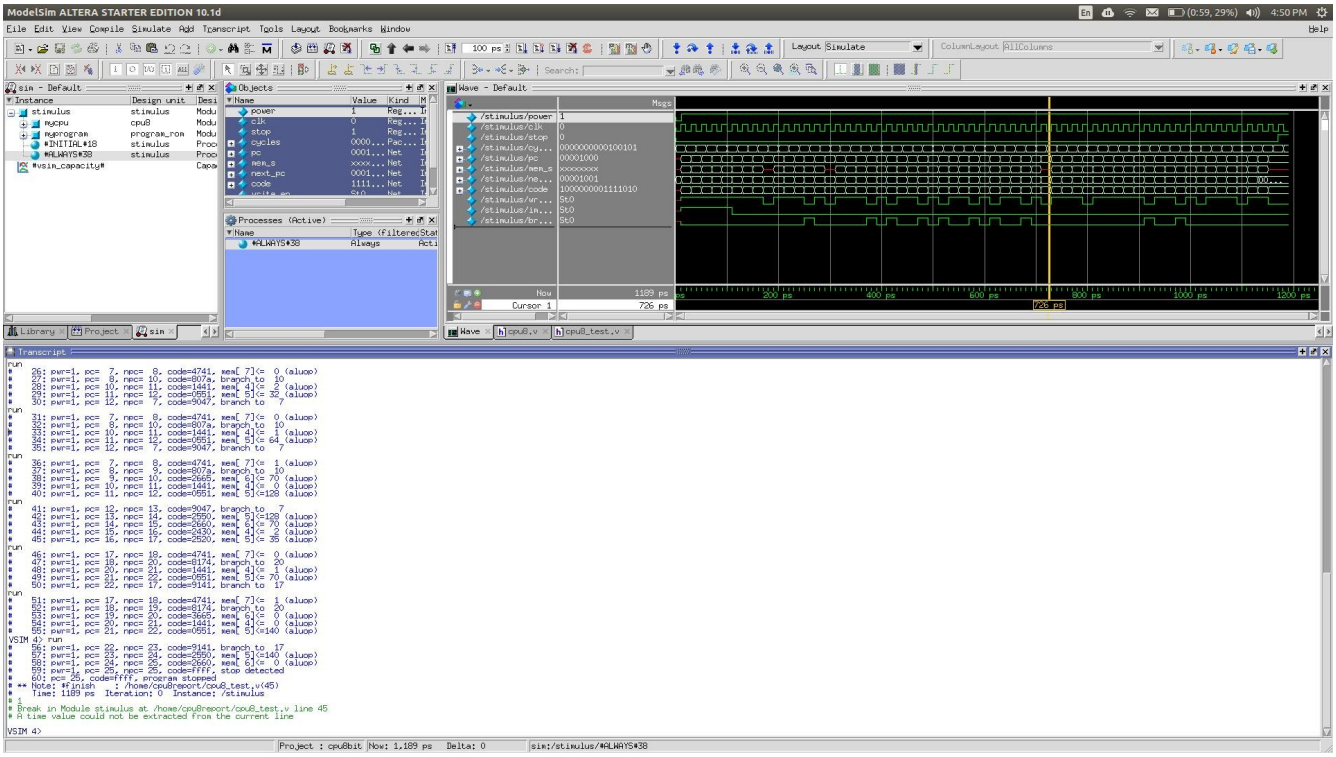
APPENDIX:

1. Machine Code:

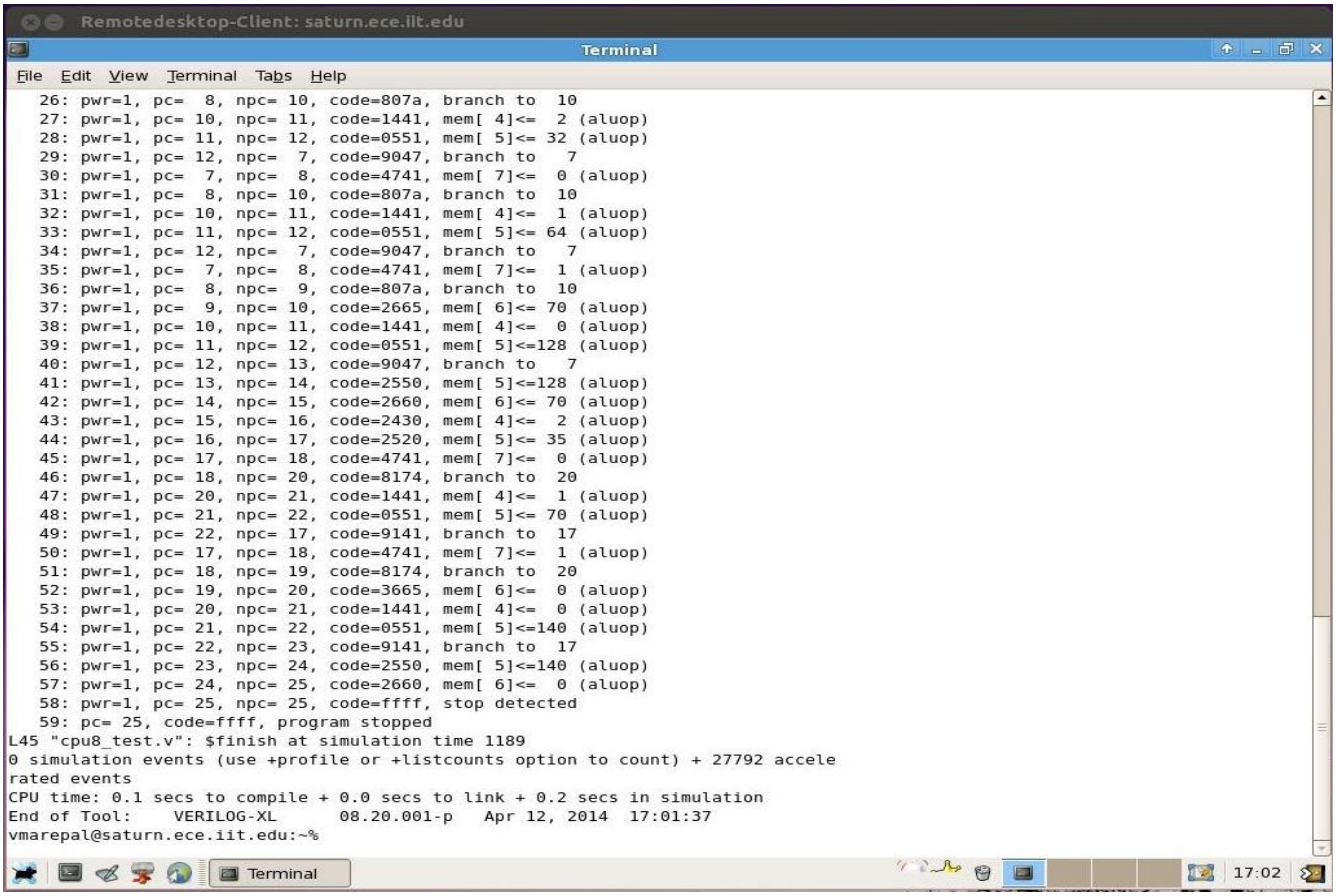
Code	Binary Representation	HEX representation
Loadi @0 , 0 ;	1010 0000 0000 0000	a000
Loadi @1, 1 ;	1010 0001 0000 0001	a101
Loadi @2 , 35 ;	1010 0010 0010 0011	a223
Load @3 , 2 ;	1010 0011 0000 0010	a302
Load @6 , 0 ;	1010 0110 0000 0000	a600
Add @4 , @2 , @0 ;	0010 0100 0010 0000	2420
Add @5 , @3 , @0 ;	0010 0101 0011 0000	2530
And @7 , @4 , @1 ;	0100 0111 0100 0001	4741
Jz @7 , 10 ;	1000 0000 0111 1010	807a
Add @6 , @6 , @5 ;	0010 0110 0110 0101	2665
Srl @4 , @4 , @1 ;	0001 0100 0100 0001	1441
Sll @5 , @5 , @1 ;	0000 0101 0101 0001	0551
Jnz @4 , 7 ;	1001 0000 0100 0111	9047
Add @5 , @5 , @0 ;	0010 0101 0101 0000	2550
Add @6 , @6 , @0 ;	0010 0110 0110 0000	2660
Add @4 , @3 , @0 ;	0010 0100 0011 0000	2430
Add @5 , @2 , @0 ;	0010 0101 0010 0000	2520
And @4 , @7 , @1 ;	0100 0111 0100 0001	4741
Jz @7 , 20 ;	1000 0001 0111 0100	8174
Sub @6 , @6 , @5 ;	0011 0110 0110 0101	3665
Srl @4 , @4 , @1 ;	0001 0100 0100 0001	1441
Sll @5 , @5 , @1 ;	0000 0101 0101 0001	0551
Jnz @4 , 17 ;	1001 0001 0100 0001	9141
Add @5 , @5 , @0 ;	0010 0101 0101 0000	2550
Add @6 , @6 , @0 ;	0010 0110 0110 0000	2660
Stop	1111 1111 1111 1111	ffff

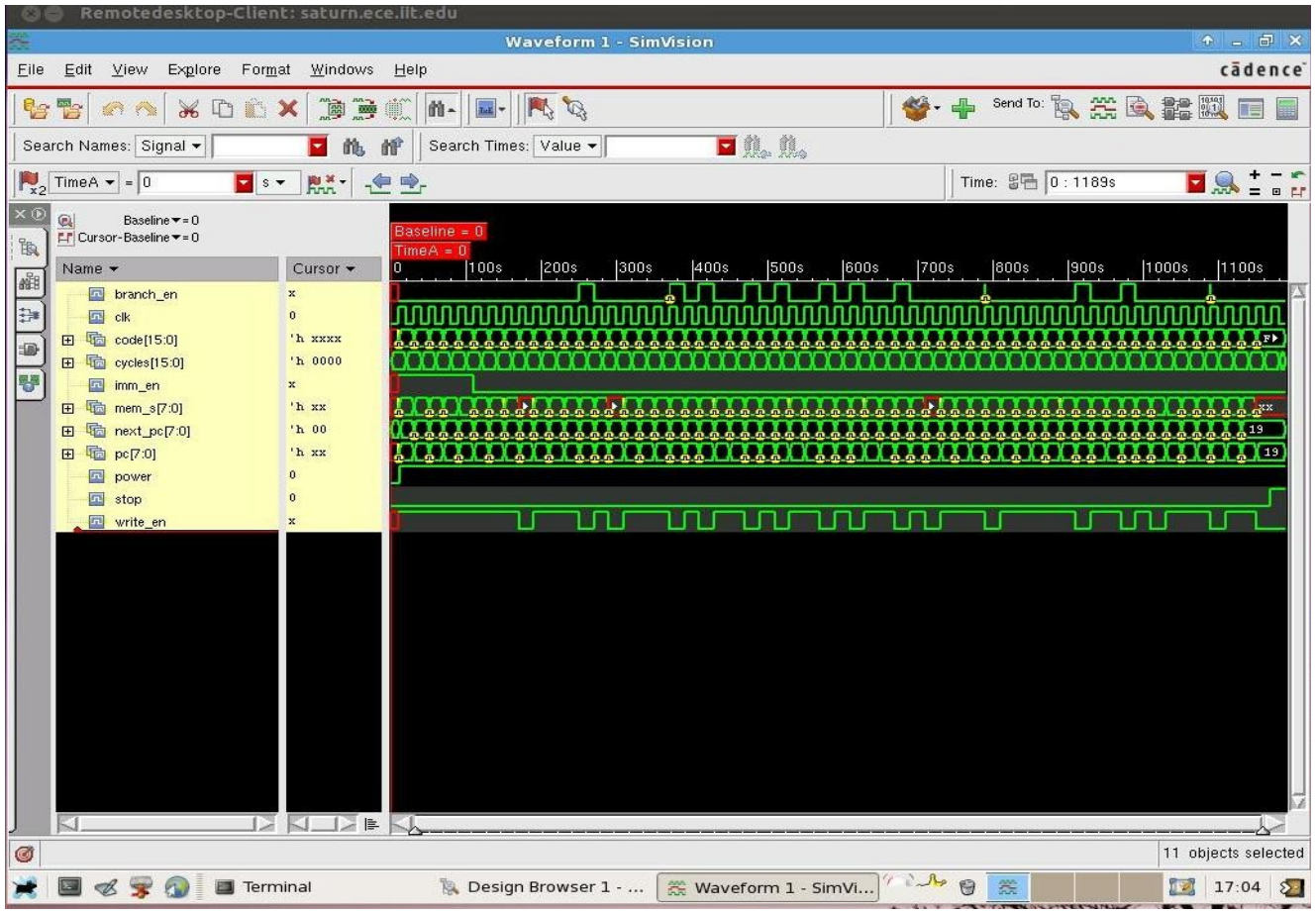
2. Screenshots showing the Modelsim Simulation:



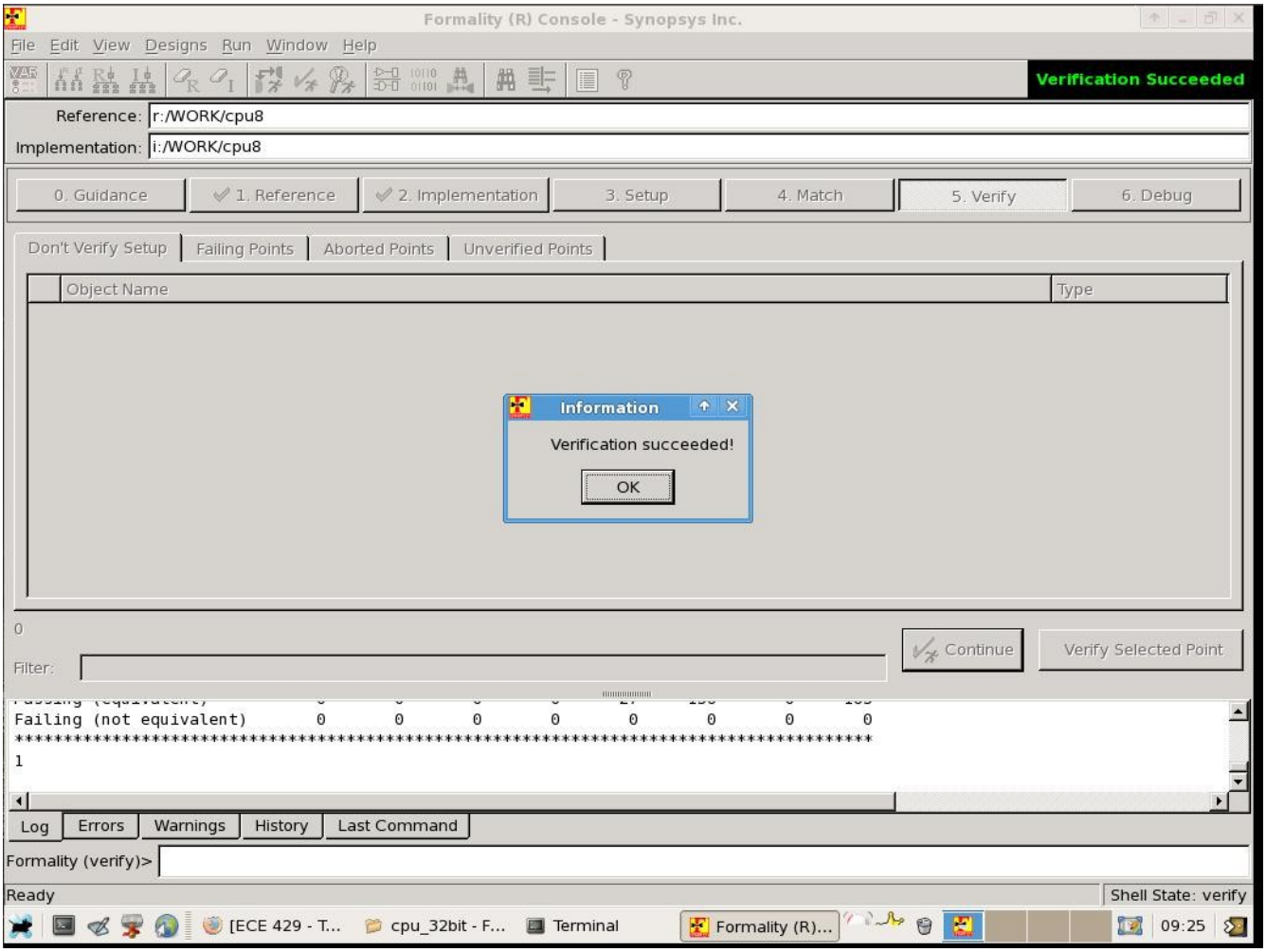


3) Design Verification Process using gcc compiler and Simivision:





4) Equivalence Check by Formal ESP:



REFERENCES:

1.http://en.wikipedia.org/wiki/Verilog