# Chapter-1

# Introduction

RISC or Reduced Instruction Set Computer is a design philosophy that has become a mainstream in Scientific and engineering applications. RISC is a microprocessor CPU design philosophy that favours a smaller and simpler set of instructions that all take about the same amount of time to execute.

## 1.1 RISC PHILOSOPHY

The idea was inspired by the discovery that many of the features that were included in traditional CPU designs to facilitate coding were being ignored by the programs that were running on them. Also these more complex features took several processor cycles to be performed. Additionally, the performance gap between the processor and main memory was increasing. This led to a number of techniques to streamline processing within the CPU, while at the same time attempting to reduce the total number of memory accesses. When the controller design become more complex in CISC and the performance was also not up to expectations, people started looking on some other alternatives. It had been found that when a processor talks to the memory the speed gets killed.

So the one improvement on CPI was to keep the instruction set very simple. Simple in not the way it works but the way it looks. That's why we have very few instructions in any typical RISC architecture where processor asks data from memory probably not other than Load and Store. We avoid keeping such addressing modes. The complexity of controller design has been overcome with the help of operands and Opcode bits fixed in instruction register.

The goal is to create an instruction set containing instructions that execute quickly; most of the RISC instructions are executed in a single machine cycle (after fetched and decoded). RISC instructions, being simple, are hard-wired, while CISC architectures have to use microprogramming in order to implement complex instructions.

Having only simple instructions results in reduced complexity of the control unit and the data path, as a consequence, the processor can work at a high clock frequency. Complex operations

on RISCs are executed as a sequence of simple RISC instructions. In the case of CISCs they are executed as one single or a few complex instruction.

Today RISC CPUs represent the vast majority of all CPUs in use. The RISC design technique offers power in even small sizes, and thus has come to completely dominate the market for low-power "embedded" CPUs. Embedded CPUs are by far the largest market for processors. RISC had also completely taken over the market for larger workstations for much of the 90s. After the release of the Sun SPARCstation the other vendors rushed to compete with RISC based solutions of their own. Even the mainframe world is now completely RISC based.

## 1.2 RISC vs. CISC

CISC stands for complex instruction set computer. An overriding characteristic of CISC machines is an approach to instruction set architecture that emphasizes doing more with each instruction. As a result, CISC machines have a wide variety of addressing modes. CISC machines take a “have it your way” approach to the location and number of operands in various instructions. As a result instructions are of widely varying length and execution times.

The capabilities of CISC allowed more operations to be performed into the same program size. During that period, program and data storage were given more importance since cost of memory was high. A research conducted by David Patterson and Donald Knuth showed that 85% of a program’s statements were assignments, conditional or procedure calls. Nearly 80% of the assignment statements were MOVE instructions with no arithmetic operations. As more and more capabilities were added to the processors, it was found increasingly difficult to support higher clock speeds that would otherwise have been possible.

Complex instructions and addressing modes worked against higher clock speeds, because of the greater number of microscopic actions that had to be performed per instruction. Moreover, RAM prices dropped sufficiently so that the pressure on system designers was less to design instructions that did more that it was to design systems that were faster. It was also becoming cost-effective to employ small amounts of higher-speed cache memory to reduce memory latency i.e. the writing time between when a memory is made and when it has been satisfied.

## 1.3 ADVANTAGES OF RISC

Various attempts have been made to increase the instruction execution rates by overlapping the execution of more than one instruction since the earliest day of computing. The most common ways of overlapping are pre-fetching, pipelining and superscalar operation.

1) Pre-fetching: The process of fetching next instruction or instructions into an event queue before the current instruction is complete is called pre-fetching. The earliest 16-bit microprocessor, the Intel 8086/8, pre-fetches into a non-board queue up to six bytes following the byte currently being executed thereby making them immediately available for decoding and execution, without latency.

2) Pipelining: Pipelining instructions means starting or issuing an instruction prior to the completion of the currently executing one. The current generation of machines carries this to a considerable extent. The PowerPC 601 has 20 separate pipeline stages in which various portions of various instructions are executing simultaneously.

3) Superscalar operation: Superscalar operation refers to a processor that can issue more than one instruction simultaneously. The PPC 601 has independent integer, floating-point and branch units, each of which can be executing an instruction simultaneously.

## 1.4 COMPARISON RISC AND CISC

CISC machine designers incorporated pre-fetching, pipelining and superscalar operation in their designs but with instructions that were long and complex and operand access depending on complex address arithmetic, it was difficult to make efficient use of these new speed-up techniques. Furthermore, complex instructions and addressing modes hold down clock speed compared to simple instructions. RISC machines were designed to efficiently exploit the caching, pre-fetching, pipelining and superscalar methods that were invented in the days of CISC machines. The simpler instruction set of RISC processors results in a larger memory requirement compared to the similar program compiled for a CISC architecture.

# Chapter-2

# Literature Survey

The problem of designing the processor and minimizing the power dissipation by various power minimization techniques is addressed by many authors and a brief overview of their work is mentioned below.

## 2.1 ARCHITECTURE

The RISC processor architecture consists of Arithmetic Logic Unit (ALU), Control Unit (CU), Barrel Shifter, Booth's Multiplier, Register File and Accumulator. RISC processor is designed with load/store (Von Neumann) architecture, meaning that all operations are performed on operands held in the processor registers and the main memory can only be accessed through the load and store instructions. One shared memory for instructions (program) and data with one data bus and one address bus between processor and memory. Instruction and data are fetched in sequential order so that the latency incurred between the machine cycles can be reduced. For increasing the speed of operation.

RISC processor is designed with five stage pipelining. The pipelining stages are Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Data Memory (MEM), and Write Back (WB).

The function of the instruction fetch unit is to obtain an instruction from the instruction memory using the current value of the PC and increment the PC value for the next instruction.

The main function of the instruction decode unit is to use the 32-bit instruction provided from the previous instruction fetch unit to index the register file and obtain the register data values.

The instructions opcode field bits [31-26] are sent to a control unit to determine the type of instruction to execute. The type of instruction then determines which control signals are to be asserted and what function the ALU is to perform, thus decoding the instruction. The register file reads in the requested addresses and outputs the data values contained in these registers.

These data values can then be operated on by the ALU whose operation is determined by the control unit to either compute a memory address (e.g. load or store), compute an arithmetic result (e.g. add, and or slt), or perform a compare (e.g. branch).

The control unit has two instruction decoders that decode the instruction bits and the decoded output of the control unit is fed as control signal either into Arithmetic logic unit (ALU) or Barrel shifter or Booth's Multiplier. If the instruction decoded is arithmetic, the ALU result must be written to a register. If the instruction decoded is a load or a store, the ALU result is then used to address the data memory. The final step writes the ALU result or memory value back to the register file.

RISC processor architecture consists of Arithmetic Logic Unit (ALU), Control Unit (CU), Barrel Shifter, Booth's Multiplier, Register File and Accumulator. RISC processor is designed with load

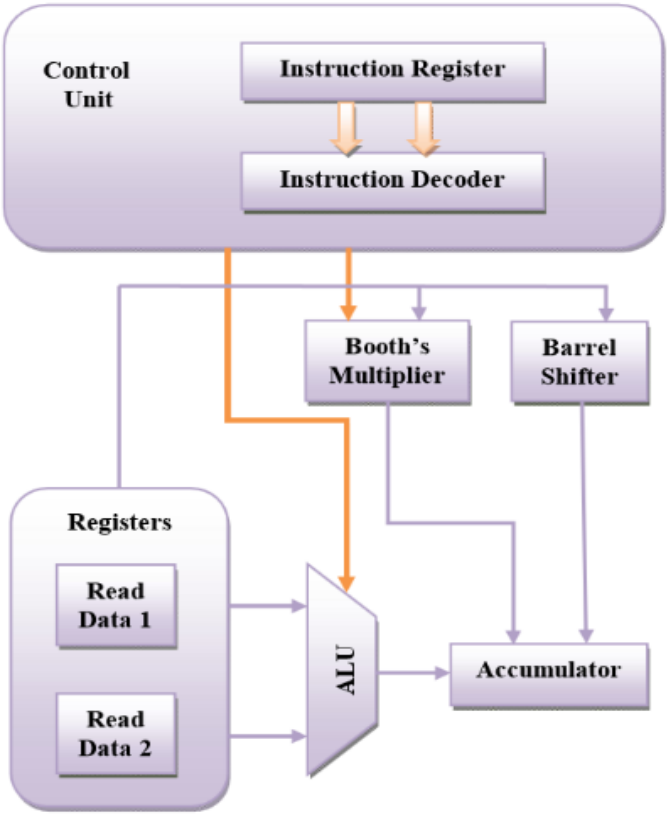


***Figure 2.1*** SYSTEM ARCHITECTUE OF A 32-BIT RISC PROCESSOR

**Figure 2.1** shows the layout of a RISC processor. It has 4 components Control Unit, Registers, ALU and accumulator.

## 2.2 ARITHMETIC LOGIC UNIT

The arithmetic/logic unit (ALU) executes all arithmetic and logical operations. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental block of many types of computing circuits, including the central processing unit (CPU) of computers. The arithmetic/logic unit can perform arithmetic operations or mathematical calculations like addition, and subtraction. As its name implies, the arithmetic/logic unit also performs logical operations include Boolean comparisons, such as AND, OR, XOR, NAND, NOR and NOT operations.

| Select Lines Op (2:0) | | | Function Performed |
|---|---|---|---|
| Op(2) | Op(1) | Op(0) | Operation |
| 0 | 0 | 0 | ADD |
| 0 | 0 | 1 | SUB |
| 0 | 1 | 0 | NOT |
| 0 | 1 | 1 | NAND |
| 1 | 0 | 0 | NOR |
| 1 | 0 | 1 | AND |
| 1 | 1 | 0 | OR |
| 1 | 1 | 1 | XOR |

***Table 2.1*** ALU TABLE

**TABLE 2.1** shows the ALU functions that will be performed according to the operational codes. ALU takes three input. The first input is the 3-bit OP code and the second and third inputs are the numbers on which the operation is to be performed. The OP code is a 3-bit number that decides what ALU function is to be implemented on the inputs.

## 2.3 BARREL SHIFTER

A Barrel Shifter is a digital circuit that can shift a data word by a specified number of bits without the use of any sequential logic, only pure combinational logic. The design consists of a total of eight 8x1 multiplexers. The output of one multiplexer is connected as input to the next multiplexer in such a way that the input data gets shifted in each multiplexer thus performing the rotation operation. Depending on the select lines the number of rotation varies. With select lines low there is no output. If select line s0 is high 1-bit rotation takes place, if s1 is high 3-bit rotation.

| Select Lines | | | Output of Rotator | | | | | | | | Function Performed |
|---|---|---|---|---|---|---|---|---|---|---|---|
| S2 | S1 | S0 | Q 7 | Q 6 | Q 5 | Q 4 | Q 3 | Q 2 | Q 1 | Q 0 | |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1- Bit rotate |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3- Bit rotate |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 5- Bit rotate |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 7- Bit rotate |

***Table 2.2*** BARREL SHIFTER TABLE

**TABLE 2.2** shows the BARREL SHIFTER table. Barrel shifter takes 2 inputs. The first input is the select lines which determines the bits to be shifted. The second input is the 8-bit number which is to be shifted. For e.g. if select input = 5, the number will get left shifted by 5 bits. In Barrel Shifter, no bits are lost.

## 2.4 BOOTH MULTIPLIER

The Multiplier is implemented using the Booth algorithm. The two main advantages of this algorithm are speed and the ability to do signed multiplication (using two's complement) without any extra conversions. Booth's algorithm uses an extra bit on the right of the least significant bit in the product register. This booth-bit starts out as 0.

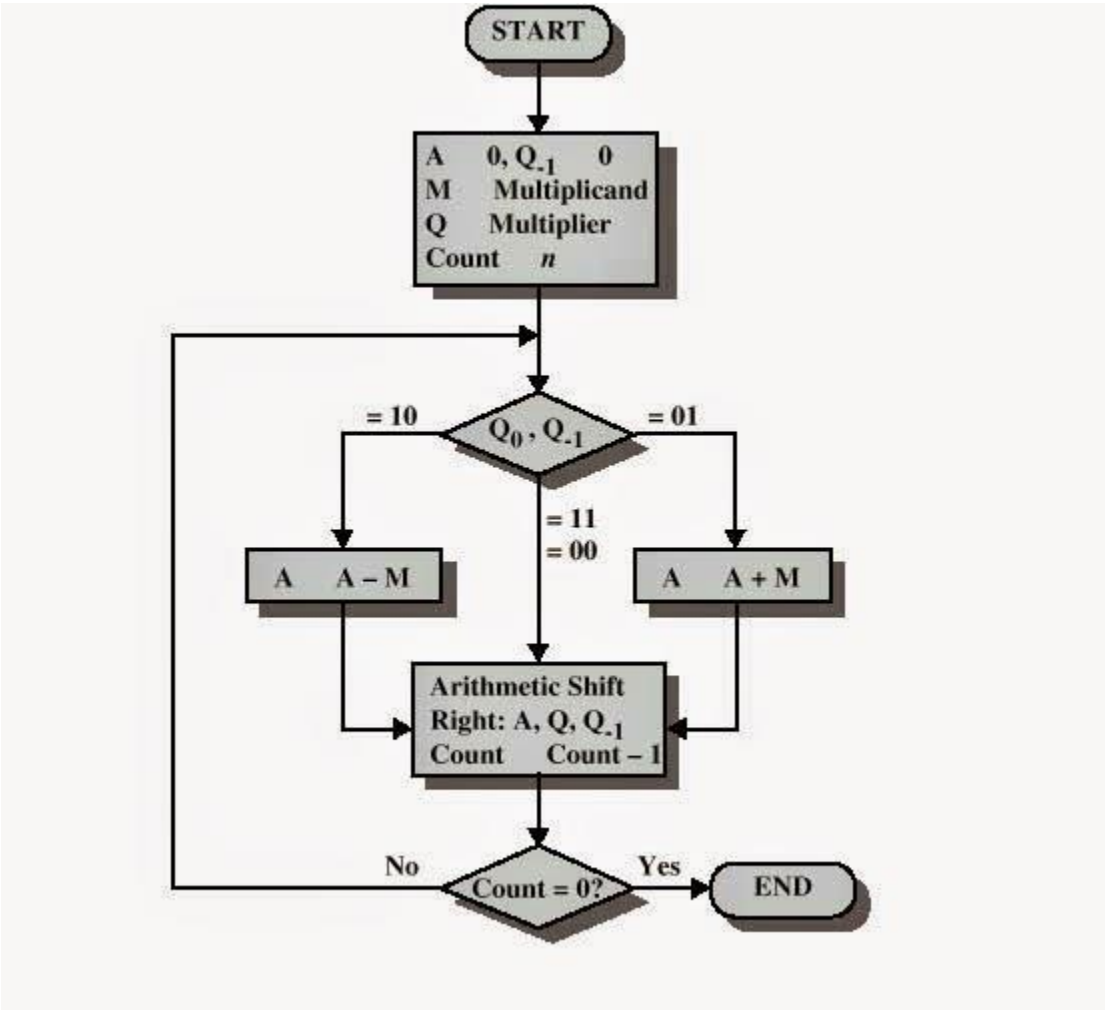


***Figure 2.2*** FLOWCHART OF BOOTH ALGORITHM

**Figure 2.2** shows flowchart of BOOTH ALGORITHM. The multiplier is scanned sequentially from right to left. In this case, last two adjacent bits are examined during each step of the procedure.

## 2.5 SOFTWARE USED

Xilinx Tools is a suite of software tools used for the design of digital circuits implemented using Xilinx Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD).

The design procedure consists of

(a) Design entry

(b) Synthesis and implementation of the design

(c) Functional simulation

(d) Testing and verification.

Digital designs can be entered in various ways using the above CAD tools: using a schematic entry tool, using a hardware description language (HDL) – Verilog or VHDL or a combination of both. In this lab we will only use the design flow that involves the use of Verilog HDL. The design is implemented using XILINX VIRTEX4 Tool for embedded and portable applications.

The results are analysed using ISim_Tool. ISim provides a complete, full-featured HDL simulator integrated within ISE. HDL simulation now can be an even more fundamental step within your design flow with the tight integration of the ISim within your design environment.

# Chapter-3

## Results and Discussions

We have implemented 3 modules of RISC processor using Verilog in XILINX software. 3 modules are 32-bit ALU, 8-bit BARREL SHIFTER and 32-bit BOOTH MULTIPLIER. We can see RTL schematic of these modules. We have also analysed the timing diagrams of these modules in ISIM.

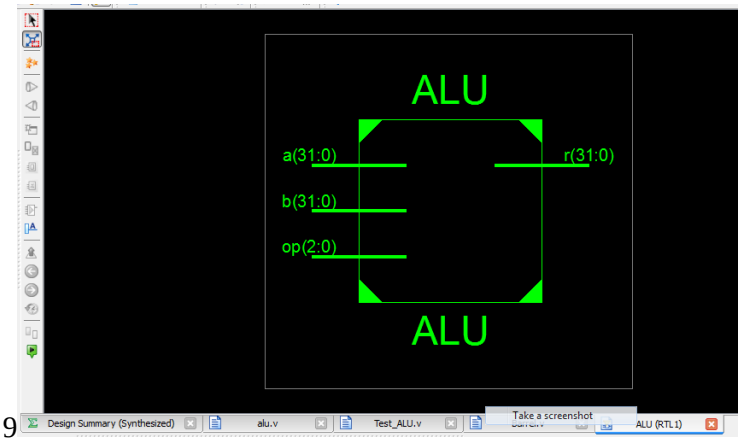


***Figure 3.1*** RTL SCHEMATIC OF ALU MODULE

**Figure 3.1** shows RTL Schematic of ALU Module. a & b are two 32-bit inputs, op is a 3-bit operational code and r is the 32-bit output.

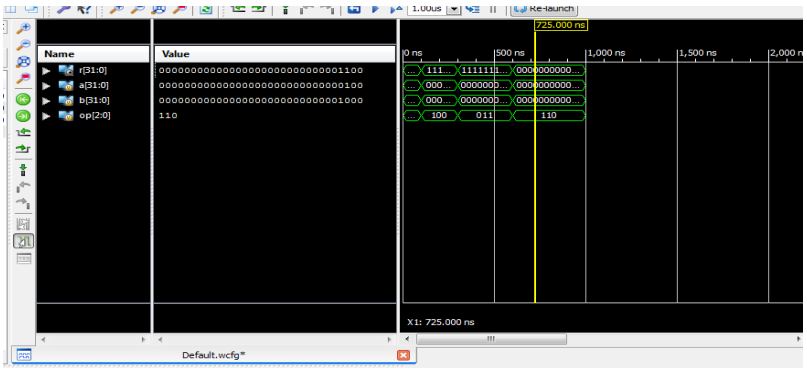


***Figure 3.2*** ISIM SIMULATION OF ALU MODULE

**Figure 3.2** shows ISIM simulation of ALU Module. Here a = 100 (4), b = 1000(8), OP code = 110 (6) (OR operation). The result r comes out to be 1100 (12).

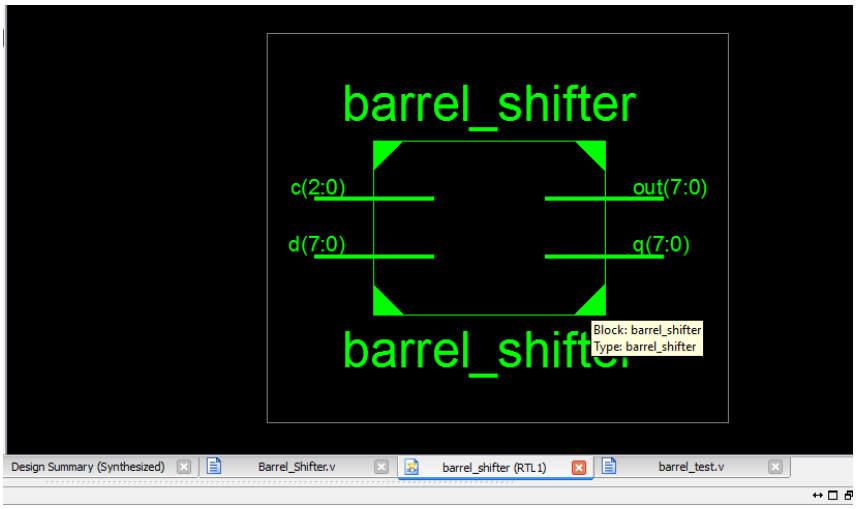


***Figure 3.3*** RTL SCHEMATIC OF BARREL SHIFTER

**Figure 3.3** shows RTL Schematic of BARREL SHIFTER. c is a 3-bit select input, d is an 8-bit number that is to shifted and out is the 8-bit output.

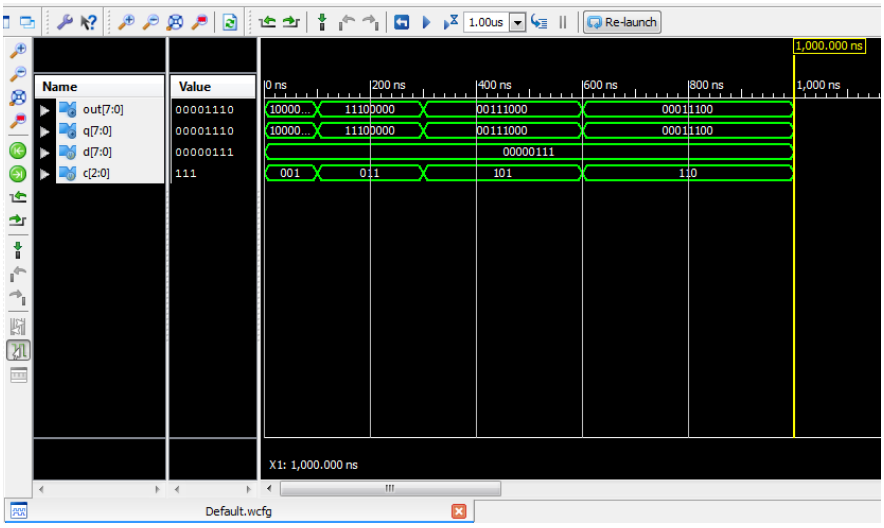


***Figure 3.4*** ISIM SIMULATION OF BARREL SHIFTER

**Figure 3.4** shows ISIM simulation of BARREL SHIFTER. Here c = 111 (7-bit shift), d = 00000111 (7). The result q comes out to be 00001110 (14).

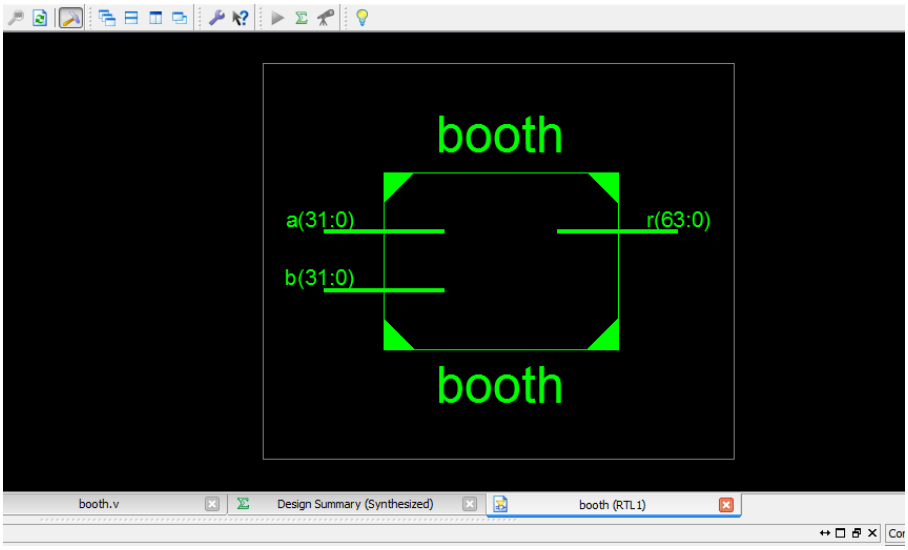


***Figure 3.5*** RTL SCHEMATIC OF BOOTH MULTIPLIER

**Figure 3.5** shows RTL Schematic of BOOTH MULTIPLIER. a & b are two 32-bit inputs and r is the 64-bit output.

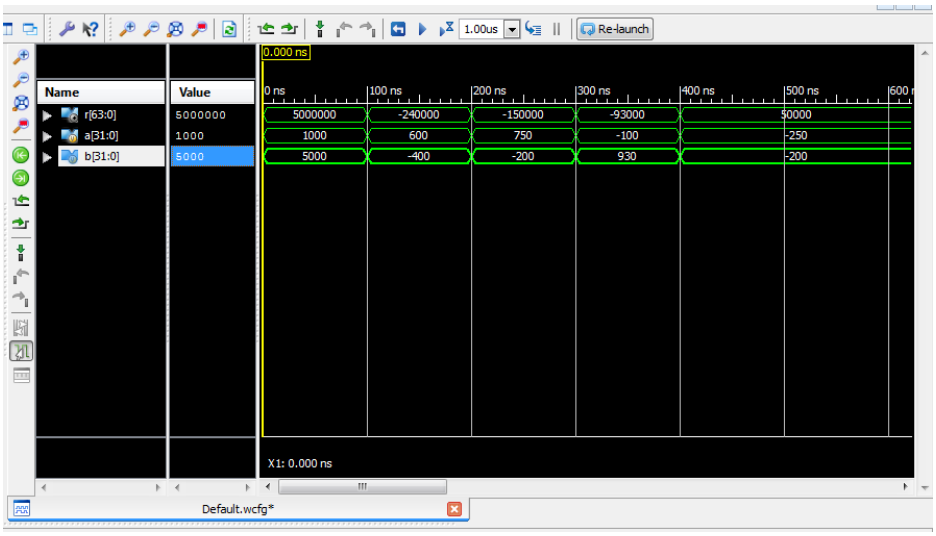


***Figure 3.6*** ISIM SIMULATION OF BOOTH MULTIPLIER

**Figure 3.6** shows ISIM simulation of BOOTH MULTIPLIER. Here a = 1000, b = 5000. The result r comes out 5000000.

# Chapter-4

# Conclusion and Future Enhancement

The research that led to the development of RISC architectures represented an important shift in computer science, with emphasis moving from hardware to software. This project is completed using Verilog HDL.

## 4.1 CONCLUSION

Reduced Instruction Set Computing is an evolution in computer architectures that emphasizes speed and cost-effectiveness over ease of assembly-language programming and the conservation of memory. In addition, RISC-based designs will continue to grow in speed and ability much more rapidly than comparable CISC designs over the next several years. The design of ALU, Barrel Shifter and Booth Multiplier is successfully implemented.

## 4.2 FUTURE ENHANCEMENTS

This is not the complete RISC processor, further modules to be implemented are CONTROL UNIT and GENERAL PURPOSE REGISTER. After implanting all the modules of the RISC processor we have to design a 32-bit processor with 16 instruction set. Furthermore we have to verify this design using simulation and compare the performance of our processor with other processors in terms of delay, power dissipation etc.

# REFERENCES


1) Galani Tina G., Riya Saini and R.D.Daruwala “Design and Implementation of 32 – bit RISC Processor using Xilinx”, International Journal of Emerging Trends in Electrical and Electronics *(IJETEE – ISSN: 2320-9569) Vol. 5, Issue. 1, July-2013*
2) Paul Gigliotti “Implementing Barrel Shifters Using Multipliers”
3) www.xilinx.com/literature
4) www.asic-world.com/verilog
5) www.fpga4student.com
6) www.stackoverflow.com
7) vlsi.pro/category/front-end/hdl/verilog/