IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 7, July 2016

Floating Point Parallel Processing Multiplier Based RISC (MIPS) Processor

Omkar A. Shastri[1], Asst. Prof. Shubhangini Ugale[2], Asst. Prof. Vipin Bhure[3]
G.H.R.A.E.T, Nagpur[1, 2, 3]

Abstract: This paper proposes a design of high speed 32 bit RISC processor. The processor consists of blocks namely Instruction Fetch block, Instruction Decode block and Execution block. The ALU in the execution block comprises of a single precision floating point multiplier designed in a parallel architecture thus improving the speed and accuracy of the execution. Furthermore the power gating technique is used which switch off the power at the time when processor execution is not required. All the blocks are designed using VHDL hardware description language.

Keywords: RISC, Floating point multiplier, Power gating, VHDL.

I. INTRODUCTION

Today microprocessors can be found in almost every digital system. The decision to include a microprocessor in a design is often very clear because it transforms the design effort from a logic design into a software design. Microprocessors are used in variety of electronic gadget such as computers, laptops, cell phones etc. In conventional approach speed of the processor is less. So there is a need of designing a high speed and high accuracy processors. John Cocke originated the RISC concept in 1974 by proving that about 20% of the instructions in a computer did 80% of the work. This paper describes a 32-bit RISC processor designed for embedded and portable application. The features of this processor are it consumes less power and it operates in high speed. Other features of RISC are Uniform instruction format, identical general purpose registers, and Simple addressing modes. RISC stands for Reduced Instruction Set Computer. Today RISC is considered to be the basis for designing high performance processors. The RISC processor have reduced number of Instructions, fixed instruction length, more general purpose register which are organized into register file, load-store architecture and simplified addressing modes which makes individual instruction execute faster, achieve a net gain in performance. The RISC processor requires less number of transistors hence the area required on the chip is less as compared to CISC. Only the load-store instruction access memory, no arithmetic or logic or IO instruction operates directly on memory content which is the key to single clock execution of instructions. It is easier to produces powerful optimized compilers since there are fewer instructions in the instruction set.

II. INSTRUCTION SET ARCHITECTURE

A. RISC (MIPS) Instruction Set

As proposed Processor is 32Bit MIPS RISC Processor, all the instructions are 32Bit in length which uses the four different types of formats. Instruction formats vary from instruction to instruction. The design has 32 general purpose registers each 32Bit. It supports addressing modes such as the Register addressing mode, immediate addressing mode, Register indirect addressing, Implicit/Inherent/Implied addressing and direct addressing mode.

B. RISC (MIPS) Instruction Format

The RISC (MIPS) processor consists of different instruction set format for certain instructions which are accessed using a control unit which consists of different control signal that decides the type of instruction. The format below shows the standard format for the instructions.

Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Figure 1 Instruction set Format

The different instruction formats are as shown below:

a) R-Type (Register Format)

	(31 to 26)	(25 to 21)	(20 to16)	(15 to11)	(10 to 6)	(5 to 0)
R-Format	Opcode	Rs	Rt	Rd	shift	Function

Figure 2 R-Type Instructions Format

Figure 3 shows R-Type instruction format. The last 6 bits represents the op-code. Next 15 bits represents 3 registers Rs, Rt and Rd. Rs and Rt are source registers, Rd is the destination registers. The next 5 bits represents shift amount which points to the number of bits to be shifted. Last 6 bits is function field that points to the function or the operation to be performed on the two source register value.

b) I-Type (Immediate Format)

	(31 to 26)	(25 to 21)	(20 to 16)	(15 to 0)
I- Format	Opcode	Rs	Rt	Address/Immediate value

Figure 3.I-Type Instructions Format

Figure 4 shows I-Type instruction format. Similar to R type, first 6 bits represents the op-code and next 10 bits represents Rs, Rt respectively were Rs is source register and Rt is source register for store and destination register for load operation. The remaining 16Bits represents field called as Address Value field of used for immediate data which must be sign extended from 16 to 32 bits simply by adding the sign-bit 16 times to the original value.

c) J-Type (Jump Format)

	(31 to 26)	(25 to 0)
J- Format	Opcode	Branch target address

Figure 4 J-Type Instructions Format

Figure 5 shows J-Type instruction format. First 5 bits of this instruction format represent the type of operation i.e. jump address operation to be performed. The remaining 26 bits represents the branch offset in 2's complement format. These 26 bits are added to the value of the PC to obtain the jump target address.

III. DATAFLOW

Data flow is achieved using data path of the hardware, which defines data flow. There is no clear difference between control and data. Operation code, operand, memory address, memory value, register address, register value, jump destination address and content are usually included in data, but control part consist of control signal of unit, time control signal and interrupt control signal, and these signals are not defined clearly

a) R-Format Data Path

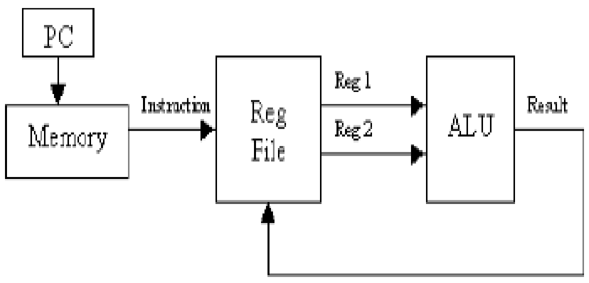


Figure 5.R Format Instruction Data Path

In R-Format data path, Instruction is fetched from the memory in accordance with the PC value. The instruction is then decoded to obtain the memory address of the two registers consisting of the values on which arithmetic operation is to be performed. The Arithmetic operation is executed with the help of ALU. The result of the operation is stored back into one of the registers.

b) RI-Format Data Path

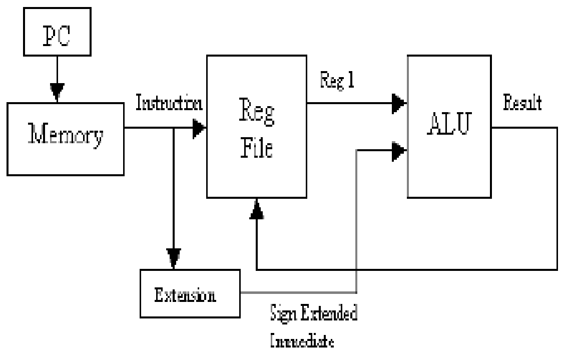


Figure 6.RI-Format Instruction Data Path

RI-Format data path is shown in Figure 7. It is similar to R-Format data path. The only difference is that the target register of R format instruction is replaced by immediate value of RI-Format data path. The immediate is 32-bit sign extended value which is fed to ALU as the second operand. Finally, the result of the ALU is written back to the register file.

c) Load Word Data Path

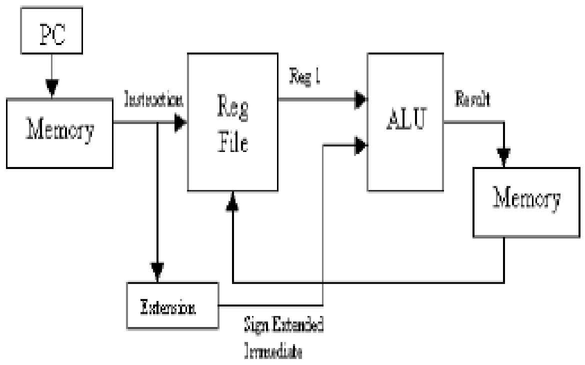


Figure 7.Load Word Data Path (lw)

The load word data path is shown in figure8. It is similar to the I type data path only the difference is that the ALU result which is an memory address is loaded into memory and the value stored at that address is fetched and stored in one of the register whose address or value is specified in the instruction.

d) Store word data path

Figure9 shows the Store Word data path. The load word and store word has a similar data path as they both have to access the data memory. In store word the ALU result is the memory address in the data memory where the specified register value is to be store through second output port of the register file.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 7, July 2016

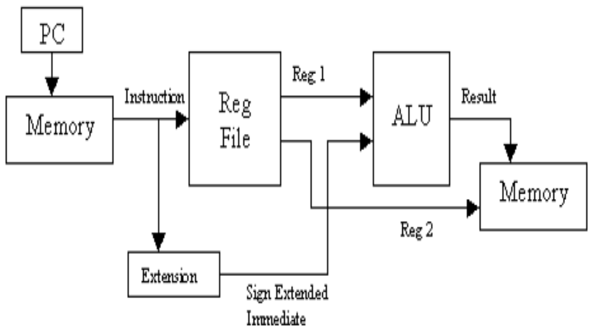


Figure 8.Store Word Data Path (sw)

IV. ARCHITECTURE OF RISC

Figure10 shows the architecture of RISC (MISC) processor. The processor mainly consists of the instruction fetch module, instruction decode module, execution module, memory module and write back module. The proposed architecture consists of 3stage: Instruction fetch stage, instruction decode stage and the execute stage. The execution stage consists of a FPU in which the floating point multiplier is designed using a parallel processing architecture thus increasing the speed of most time consuming element. The description of processors logic blocks are as follows:

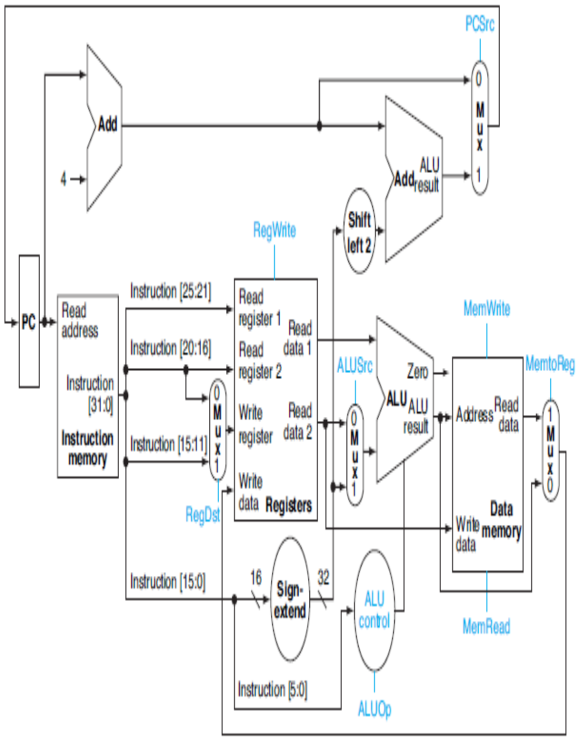


Figure 9.Architecture of RISC processor

a) Instruction fetch block
Program Counter (PC) is used to fetch the instruction from the Instruction Memory and is stored in the Instruction Register (IF/ID) at the next positive clock. This stage has various modules like Instruction Memory, which holds the instructions needed. PC holds the address of the current instruction, which is used as address to the Instruction Memory. The instructions read out from the Instruction memory are stored in the Instruction Register.

b) Instruction Decode:
In this stage, decodes the instructions sent from Instruction register. Based on the instructions, it reads the operands required for register file. Out of 32-bits, 16 go to sign extend, where those 16 bits are extended to 32-bits. The register file module gives out the value of 2 registers, which are sent to ALU in the EX stage.

c) Execution:
All the instructions are executed in this stage. All ALU operations like arithmetic and logical operations, take place in this stage. It performs operations on the data sent from ID stage. This stage also has left shift by 2 and an adder, for beq operation. The result from ALU is sent to register.

E. Floating Point Unit
A floating point (FPU), also known as a math co-processor or numeric processor is a specialized co-processor that manipulates numbers more quickly than the basic microprocessor circuitry. The FPU does this by means of instructions that focus entirely on large mathematical operations. Floating point computational logic has long been a mandatory component of high performance computer systems as well as embedded systems and mobile applications. The advantage of floating point representation over fixed point and integer representation is that it can support a much wider range of values. In the present work 32-bit FPU is used, which supports single precision IEEE-754 format. The IEEE-754 standard defines a single as 1 bit for sign, 8 bits for exponent and 23 bits for mantissa. The 32 bit single precision floating point unit has been proposed in this paper which performs certain operations like addition, subtraction, multiplication and division.

FP Add: In the module FP Add, the inputs operands are separated into their mantissa and exponent components. Then the exponents are compared and the smaller number is shifted right until it matches the larger one. Then the two mantissas are added and then the number is normalized by appending the sign bit, exponent and the obtained mantissa.

FP Sub: The input variables are separated into two components namely mantissa and exponent. Subtraction is similar to that of addition such that the mantissa of the smaller exponent is shifted to the right before performing the subtraction.

FP Mul: Firstly the two operands are converted into floating point representation number. Check if one of the operand is zero. If not then sign is computed by EX-ORING both sign bit. The two operands mantissa are multiplied and the exponents are added which are then subtracted from the biased exponent value.

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

FP Div: Division process is similar to the multiplication only the difference is that the mantissas are divided and the sum of the two exponents is added to the biased exponent value.

D) Memory Access:
In this stage, memory access stage's purpose is to read from and write to the data memory. The control signals passed determines which of the operations to do. The output of the memory is written into the register using the WB control.

V. RESULTS AND COMPARISON

A) RTL view

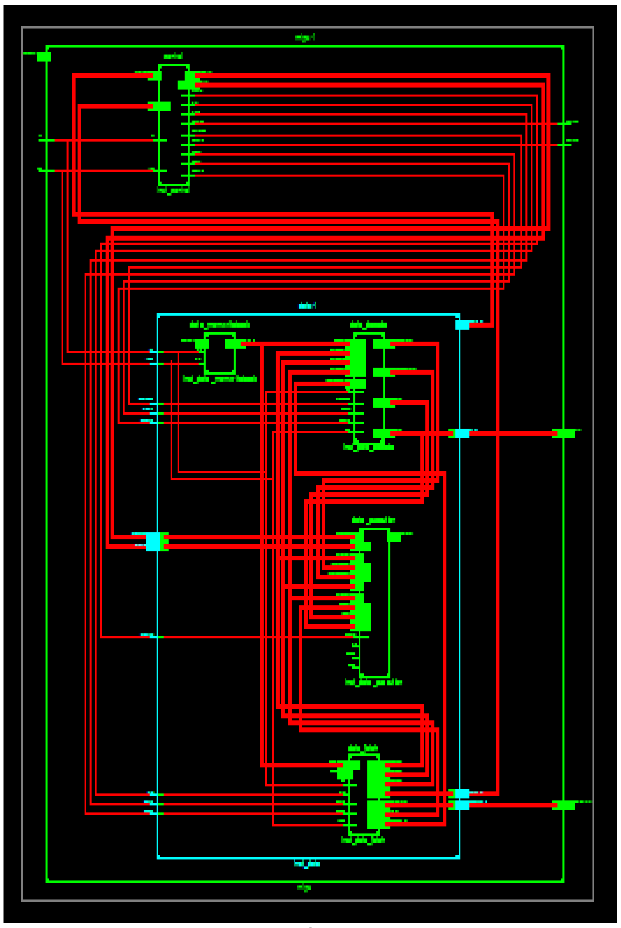
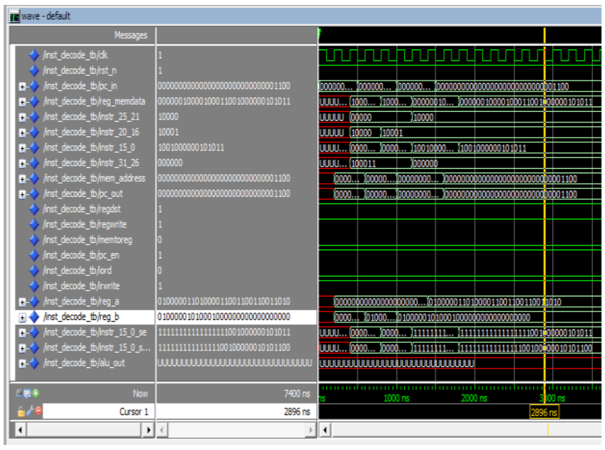


Figure 10.RTL View of MIPS RISC Processor

Figure 16 shows the RTL (Register Transfer Logic) view of 32Bit MIPS RISC Processor. It comprises of Instruction fetch unit, Instruction decoder unit, execution unit and a control unit. Function of instruction fetch unit is to fetch opcode from memory using PC and give it to instruction decoder unit. Instruction decoder unit receive the opcode and depending on the opcode, instruction format is selected and then it is fed to execution unit for execution of the instruction. Function of execution unit is to perform the specific operation according to the opcode specified in ALU control unit. It consists of Floating point ALU.

B) Simulation Result



```
wave - default
Messages
/inst_decode_tb/clk            1
/inst_decode_tb/rst_n          1
/inst_decode_tb/pc_in          00000000000000000000000000001100
/inst_decode_tb/reg_memdata    00000010000100011001000000101011
/inst_decode_tb/instr_25_21    10000
/inst_decode_tb/instr_20_16    10001
/inst_decode_tb/instr_15_0     1001000000101011
/inst_decode_tb/instr_31_26    000000
/inst_decode_tb/mem_address    00000000000000000000000000001100
/inst_decode_tb/pc_out         00000000000000000000000000001100
/inst_decode_tb/regdst         1
/inst_decode_tb/regwrite       1
/inst_decode_tb/memtoreg       0
/inst_decode_tb/pc_en          1
/inst_decode_tb/iord           0
/inst_decode_tb/irwrite        1
/inst_decode_tb/reg_a          01000001101000011001100110011010
/inst_decode_tb/reg_b          01000001010001000000000000000000
/inst_decode_tb/instr_15_0_se  11111111111111111001000000101011
/inst_decode_tb/instr_15_0_s...11111111111111100100000010101100
/inst_decode_tb/alu_out        UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
Now       7400 ns
Cursor 1  2896 ns
```

Figure11. Instruction Fetch and Decode Simulation result

The figure12 shows the simulation result of instruction fetch and instruction decode unit. Firstly the instruction is fetched and stored into instruction register and then it is decoded to get values of the two register to perform the arithmetic operation.

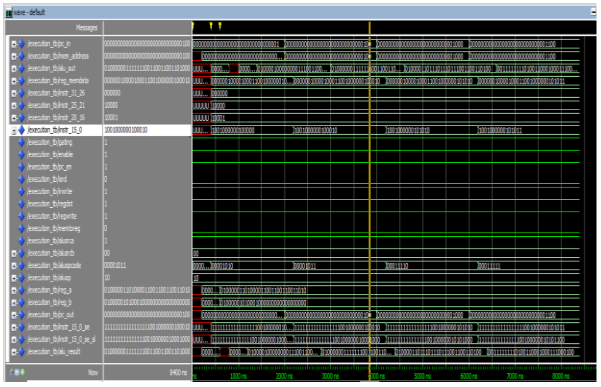


Figure12. Simulation result of R type instruction format

The figure13 shows the simulation result of R type instruction. The two registers reg_a and reg_b are obtained through the instruction fetch and decoding unit and then ALU performs operation specified by the opcodes.

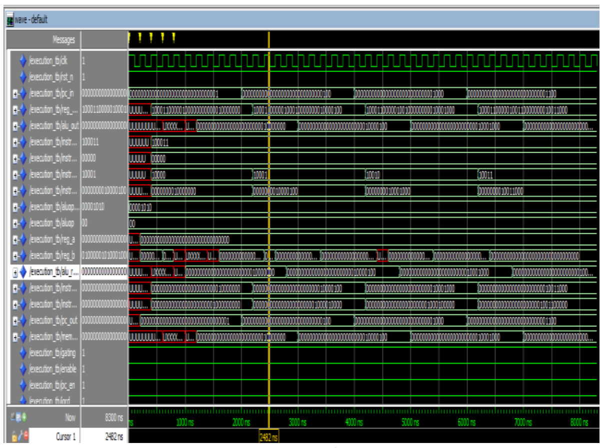


Figure 13 RI type instruction format simulation result

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 7, July 2016

The figure14 shows the RI type simulation result. The immidiate data from the instruction is sign extended and given as the second operand to the ALU. The first operand is the register value whose default value is zero thus addition gives the address at the ALU output.

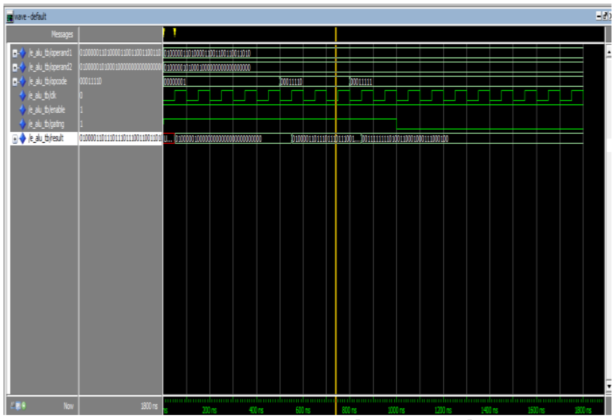
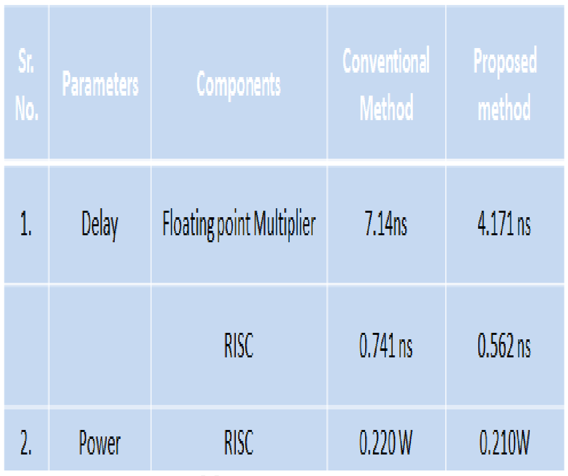


Figure 14. FPU simulation result

The figure15 shows the simulation result of Floating point unit (FPU). The simulation shows the addition, subtraction, multiplication and division of the two operands

c) Comparison Table



Sr. No.	Parameters	Components	Conventional Method	Proposed method
1.	Delay	Floating point Multiplier	7.14ns	4.171 ns
		RISC	0.741 ns	0.562 ns
2.	Power	RISC	0.220 W	0.210W

Table 1 Comparison

VI. CONCLUSION

In this research, we use VHDL to describe the system and uses top-down design method in which initially we design Instruction Fetch unit, then Decoder unit, then Execution unit, finally write back unit. The hierarchy of the design is very clear. It is easy to edit and debug. The modules synthesized and simulated are MIPS Instruction format, Floating point ALU, MIPS Instructions Set, MIPS Registers, Operation select. The design has been synthesized and simulated using Xilinx 14.7 ISE Simulator. All the goals were achieved and simulation shows that processor is working perfectly. Proposed 32Bit MIPS RISC Processor synthesis values are compared with the conventional one. So, the proposed processor can be called as a high performance processor.

REFERENCES

[1] Mr. S. P. Ritpurkar, Prof. M. N. Thakare, Prof. G. D. Korde “Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL” IEEE ICAETR – 2014

[2] Samiappa Sakthikumaran1, S. Salivahanan, V. S. Kanchana Bhaaskaran “16-Bit RISC Processor Design for Convolution Application”. IEEE-ICRTIT 2011

[3] Neenu Joseph, Sabarinath.S, Sankarapandiammal K “FPGA based Implementation of High Performance Architectural level Low Power 32-bit RISC Core” IEEE(2009 International Conference on Advances in Recent Technologies in Communication and Computing)

[4] Pravin S. Mane, Indra Gupta, M. K. Vasantha “Implementation of RISC Processor on FPGA” 2006 IEEE

[5] Naresh Grover, M.K.Soni “Design of FPGA based 32-bit Floating Point Arithmetic Unit and verification of its VHDL code using MATLAB” I.J. Information Engineering and Electronic Business, 2014, 1, 1-14

[6] Preethi Sudha Gollamudi, M. Kamaraju “Design Of High Performance IEEE- 754 Single Precision (32 bit) Floating Point Adder Using VHDL” International Journal of Engineering Research & Technology (IJERT) Vol. 2 Issue 7, July – 2013

[7] Jinde Vijay Kumar, Boya Nagaraju, Chinthakunta Swapna and Thogata Ramanjappa “Design and Development of FPGA Based Low Power Pipelined 64-Bit RISC Processor with Double Precision Floating Point Unit” International Conference on Communication and Signal Processing, April 3-5, 2014, India

[8] Mrs. Rupali S. Balpande, Mrs. Rashmi S. Keote “Design of FPGA based InstructionFetch & Decode Module of 32-bit RISC (MIPS) Processor” 2011 International Conference on Communication Systems and Network Technologies

[9] Mohit N. Topiwala, N. Saraswathi “Implementation of a 32-bit MIPS Based RISC Processor using Cadence” 2014 IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)

[10] Priyanka Trivedi, Rajan Prasad Tripathi “Design & Analysis of 16 bit RISC Processor Using low Power Pipelining” International Conference on Computing, Communication and Automation (ICCCA2015).