Design & Implementation of MIPS Processor

Jitendra Sen[1] Anil Khandelwal[2]
[1]M.Tech Scholar
[1,2]VNS Group of Institution, Bhopal, India

Abstract— The implementation of 16 bit RISC processor with microprocessor without interlocked pipeline stages MIPS) is presented. It was implemented in VHDL so as to reduce the instruction set present in the programmable memory. As the result the processor will contain the necessary logics for the implementation that requires fewer gates to be synthesized in the programmable matrix and has the capability to increase the speed of the target processor.

Key words: MIPS, VHDL, Instruction Set, ASIC

I. Introduction

Computer organization and design is a common engineering course where students learn concepts of modern computer architecture. Students often learn computer design by implementing individual sections of a computer microprocessor using a simulation-only approach that limits a student's experience to software design. As a result the students are not given the chance to implement and run their designs in real hardware, thus missing a good opportunity to gain a complete hands-on experience involves hardware-software integration. This project targets the computer architecture courses and presents an FPGA (Field Programmable Gate Array) implementation design of a MIPS (Microprocessor without Interlocked Pipeline Stages) RISC (Reduced Instruction Set Computer) Processor using VHDL (Very high speed integrated circuit Hardware Description Language). Furthermore, the goal of this work is to enhance the simulator-based approach by integrating some hardware design to help the computer architecture students gain a better understanding of both the MIPS single cycle and pipelined processor as described in the widely used book, Computer Organization and Design The Hardware/Software Interface by David A. Patterson and John L. Hennessy [11][14]. The MIPS single-cycle processor performs the tasks of instruction fetch, instruction decode, execution, memory access and write-back all in one clock cycle. First the PC value is used as an address to index the instruction memory which supplies a 32-bit value of the next instruction to be executed. This instruction is then divided into the different fields. The instructions opcode field bits [12-8] are sent to a control unit to determine the type of instruction to execute. The type of instruction then determines which control signals are 18 to be asserted and what function the ALU is to perform, thus decoding the instruction. The instruction register address field's bits [9-10], The register file supports two independent register reads and one register write in one clock cycle. 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). 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 [6][7][8][4][7]

II. The MIPS Instruction Set Architecture

As mentioned before MIPS is a RISC microprocessor architecture. The MIPS Architecture defines thirty-two, 32-bit general purpose registers (GPRs). Register $r0 is hardwired and always contains the value zero. The CPU uses byte addressing for word accesses and must be aligned on a byte boundary divisible by four (0, 8,).MIPS only has three instruction types: I-type is used for the Load and Stores instructions, R-type is used for Arithmetic instructions, and J-type is used for the Jump instructions as shown in MIPS is a load/store 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 (e.g lw, sw). A load instruction loads a value from memory into a register. A store instruction stores a value from a register to memory. The load and store instructions use 16 the sum of the offset value in the address/immediate field ALU Immediate (e.g. addi), three-operand (e.g. add, and, slt), and shift instructions (e.g. sll, srl). The J-type instructions are used for jump instructions (e.g. j). Branch instructions (e.g. beq, bne) are I-type instructions which use the addition of an offset value from the current address in the Address/immediate field along with the program counter (PC) to compute the branch target address

III. Instruction-Fetch-Unit

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 since this design uses an 8-bit data width we had to implement byte addressing to access the registers and word address to access the instruction memory. The instruction fetch component contains the following logic elements that are implemented invade: 8-bit program counter (PC) register, an adder to increment the PC by four, the instruction memory, a multiplexor, and an AND gate used to select the value of the next PC. [1][5]

IV. The-Control Unit

The control unit of the MIPS single-cycle processor examines the instruction opcode bits and decodes the instruction to generate nine control signals to be used in the additional modules as shown in Figure 3.12. The Register DST control signal determines which register is written to the register file. The Jump control signal selects the jump address to be sent to the PC. The Branch control signal is used to select the branch address to be sent to the PC. The Memory Read control signal is asserted during a load instruction when the data Memory is read to load a register with its memory contents. The Memory to Reg. control signal determines if the ALU result or the data memory output is written to the register file. The ALU Op control signals determine the function the ALU performs. (e.g. and, or, add, SBU, SLT) The Memory Write control signal is asserted when during a store instruction when a registers value is stored in the data memory. The ALU SRC control signal determines if the ALU second operand comes from the register file or the sign extend. The Register Write control signal is asserted when the register file needs to be written.

V. DATA-MEMORY-UNIT

The data memory unit is only accessed by the load and store instructions. The load instruction asserts the Memory Read signal and uses the ALU Result value as an address to index the data memory. The read output data is then subsequently written into the register file. A store instruction asserts the Memory Write signal and writes the data value previously read from a register into the computer memory address. The VHDL implementation of the data memory was described earlier. Figure 3.14 shows the signals used by the memory unit to access the data memory. The MIPS implementation as with all processors, consists of two main types of logic elements: combinational and sequential elements. Combinational elements are elements that operate on data values, meaning that their outputs depend on the current inputs. Such elements in the MIPS implementation include the arithmetic logic unit (ALU) and adder. Sequential elements are elements that contain and hold a state. Each state element has at least two inputs and one output. The two inputs are the data value to be written and a clock signal. The output signal provides the data values that were written in an earlier clock cycle. State elements in the MIPS implementation include the Register File, Instruction Memory, and Data Memory

VI. EXPERIMENTAL RESULT

With our new design, the register file is implemented to hold thirty-two, 8-bit general purpose registers amounting to 32 bytes of memory space. This easily fits into one 256 x 8 EAB within the FPGA. The full 32-bit version of MIPS will require combining four 256 x 8 EABs to implement the register file. The register file has two read and one write input ports, meaning that during one clock cycle, the processor must be able to read two independent data values and write a separate value into the register file. Figure 3.4 shows the MIPS register file. The register file was implemented in VHDL by declaring it as a one-dimensional array of 32 elements or registers each 8-bits wide. (e.g. TYPE register file IS ARRAY (0 TO 31) OF STD_LOGIC_VECTOR (7 DOWNTO 0)) By declaring the register file as a one-dimensional array, the requested register address would need to be converted into an integer to index the register file.(e.g. Read_Data_1 <= register file (CONV_INTEGER (read_register_address1 (4 DOWNTO 0)))) Finally, to save from having to load each register with a value, the registers get initialized to their respective register number when the

VII. MIPS-PROCESSOR

The initial task of this project was to implement in VHDL the MIPS single-cycle processor using Xilinx. Also helped in the overall design of the VHDL implementation. The first part of the design was to analyze the single-cycle data path and take note of the major function units and their respective connections. The MIPS implementation as with all processors, consists of two main types of logic elements combinational and sequential elements. Combinational elements are elements that operate on data values, meaning that their outputs depend on the current inputs. Such elements in the MIPS implementation include the arithmetic logic unit (ALU) and adder. Sequential elements are elements that contain and hold a state. Each state element has at least two inputs and one output. The two inputs are the data value to be written and a clock signal. The output signal provides the data values that were written in an earlier clock cycle. State elements in the MIPS implementation include the Register File, Instruction Memory, and Data Memory .While many of logic units are straightforward to design and implement in VHDL, considerable effort was needed to implement the state elements. It was determined that the full 32-bit version of the MIPS architecture would not fit on to the chosen .includes nine embedded array blocks(EABs) each providing only 2,048 bits of memory for a total of 2 KB memory space. The full 32-bit version of MIPS requires no less than twelve.

Number of Slice Registers	51	126,800	1%
Number used as Flip Flops	51		
Number of Slice LUTs	360	63,400	
Number used as Memory	64	19,000	
Number of occupied Slices	126	15,850	1%
Number of occupied Slices	126	15,850	1%
Number of LUT Flip Flop pairs used	371		
Number with an unused Flip Flop	320	371	86%
Number with an unused LUT	11	371	2%
Number of fully used LUT-FF pairs	40	371	10%
Number of unique control sets	5		

Table 1: Timing Summary

Timing Constraints	Proposed Design	Previous Design
Minimum period	5.533ns (Maximum Frequency: 180.726MHz	4.202ns (Maximum Frequency: 70.312MHz)
Minimum input arrival time before clock	5.021ns	3.939ns
Maximum output required time after clock	4.492ns	6.722ns
Maximum combinational path delay:	3.992ns	

Table 2: Utilization Summary

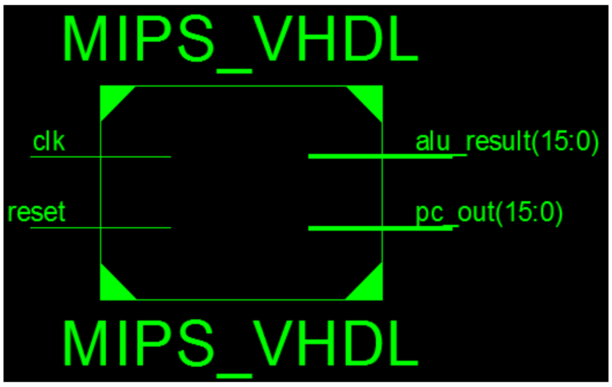

Fig. 1: MIPS Block

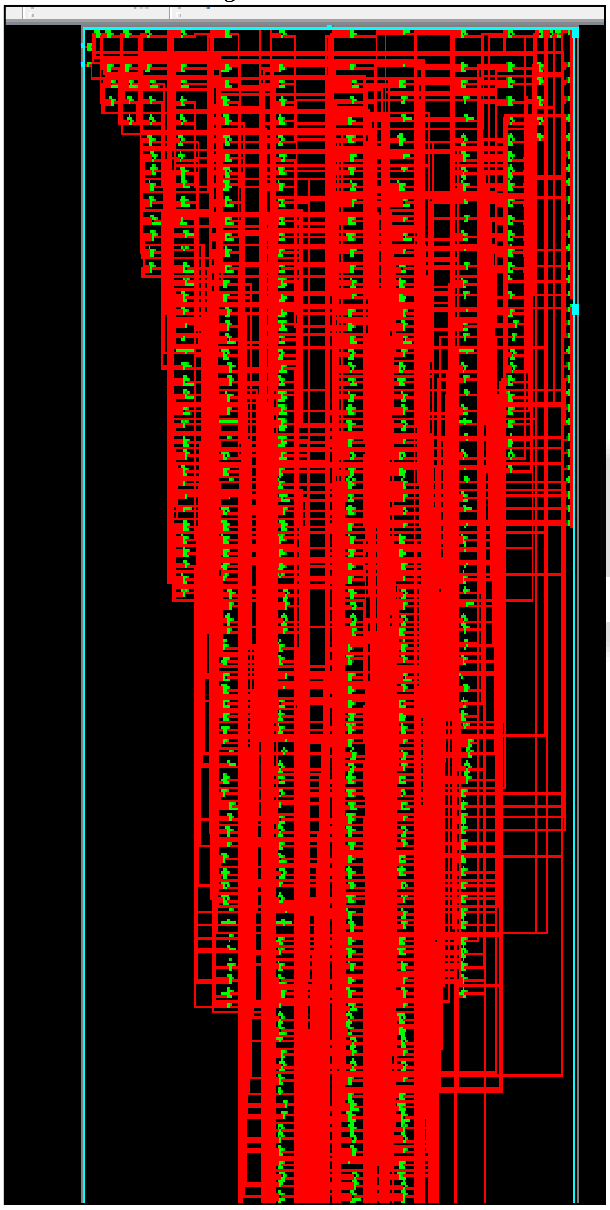

Fig. 2: RTL View of MIPS Block

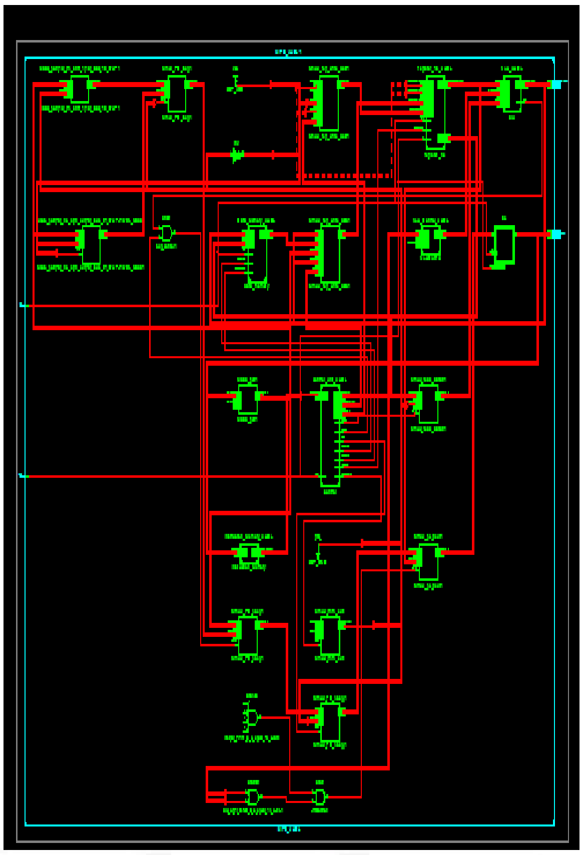

Fig. 3: RTL View OF MIPS Block

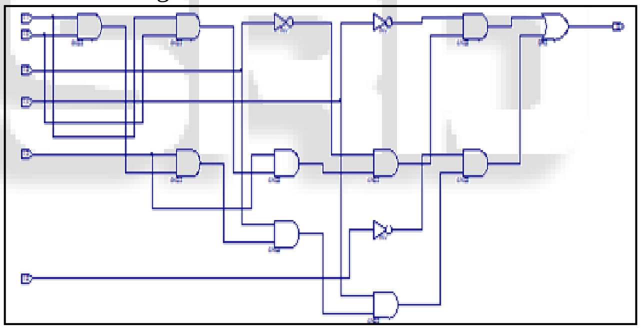

Fig. 4: Logic Diagram of MIPS

A. *Timing Constraint*

Default period analysis for Clock 'clk'Clock period: 5.533ns (frequency180.726MHz)Total number of paths / destination ports: 355555 / 229Delay:5.533ns (Levels, of, Logic=13)Source To get the accurate results of total power consumption, .ncd, .xpa and physical constraint file (.pcf file)need to be loaded into XPA which is generated while implementation.

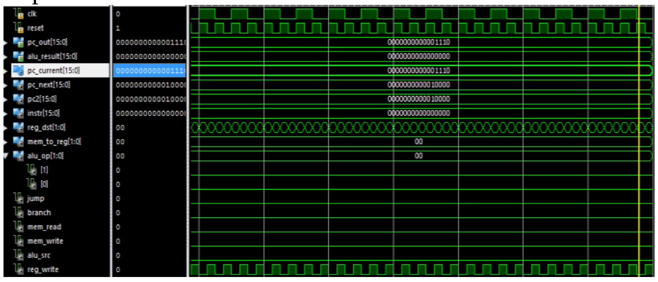

Fig. 2: Simulation of MIPS

On Chip Power Consumption	Previous Power	Proposed Power
Clock	5.80	0
Logic	3.39	0
Signals	7.99	0
IOs(dynamic power)	37.74	0.013
Quiescent	99.28	0.017
Total	155.03	0.042

Table 3: Summary of Power Optimization

VIII. CONCLUSION

In the paper we have been able to design and simulate a 16-bit RISC processor for a particular instruction set by modifying the data path of the single-cycle MIPS by using Xilinx 14.7 ISE design suite. The processor executes more number of instructions with less critical path delay. The reduction in critical path delay also led to the reduction in the execution time thus leading to a better processor. The future scope of this work includes: We can increase the number of instructions and additional pipelined stages, which will improve the performance of CPU design. And can enhance the total throughput of the design.

REFERENCES

[1] Agineti Ashok ; V. Ravi ASIC design of MIPS based RISC processor for high performance,2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2),16 October 2017,10.1109/ICNETS2.2017.8067945,

[2] Neha Dwivedi Pradeep Chhawcharia Power mitigation in high-performance 32-bit MIPS-based CPU on Xilinx FPGAs, INSPEC Accession Number: 17258563, India 2017 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia),10.1109/ICCE,ASIA.2017.8307850 ,Bangalore, India

[3] Mohammad Zaid, Prof. Pervez Design And Application Of Riscprocessor Aligarh Muslim University, India ,978-01-5090-6674-2-2017-IEEE,2017 International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT) DOI: 10.1109/MSPCT.2017.8364013 Aligarh, India 2017 8th 978-1-5090-6332-1/17/$31.00

[4] Safaa S. Omran Design Of Multithreading SHA-1 & SHA-2 MIPS Processor Using FPGA, Computer Engineering Techniques ,Baghdad, Iraq, May 2017, DOI: 10.1109/ICITECH.2017.8079918,Conference Location: Amman, Jordan

[5] Marko Mišić et.al. Extending valgrind framework with the MIPS MSA support Published in: 2017 Zooming Innovation in Consumer Electronics International Conference (ZINC)1 June INSPEC Accession, Number: 17026260DOI: 10.1109/ZINC.2017.7968660, Novi Sad, Serbia

[6] Bai-ZhongYing, Computer Organization, Science Press, 2000. 11.

[7] Wang-AiYing, Organization and Structure of Computer Tsinghua University Press, 2006.

[8] Wang-YuanZhen, IBM-PC Macro Asm Program, Huazhong

[9] University of Science and Technology Press, 1996.9.

[10] MIPS Technologies, Inc. MIPS32™ Architecture for Programmers Volume II: The MIPS32™ Instruction Set , June 9, 2003. Zheng-WeiMin, Tang-ZhiZhong. Computer System

[11] Structure (The second edition), Tsinghua University Press 2006.

[12] Pan-Song, Huang-JiYe, SOPC Technology Utility Tutorial, Tsinghua University Press, 2006.

[13] MIPS32 4KTMProcessor Core Family Software User' Manual, MIPS Technologies Inc.

[14] Mo-JianKun, Gao-JianSheng, Computer Organization, Huazhong University of Science and Technology Press, 1996.