Design and Implementation of a 32-bit RISC Processor on Xilinx FPGA

***Wael M El-Medany*[1], *Khalid A Al-Kooheji*[2]**

[1] Department of Communications and Electrical Engineering, Faculty of Engineering, Fayoum University, Egypt, Computer Engineering Department, Information Technology College, University Of Bahrain, 32038 Bahrain, Email: wmelmedany@itc.uob.bh, waelelmedany@gmail.com, Tel No: +973-39764964

[2]Alba, Manama, Bahrain Po Box 570

Abstract

This paper concerned with the design and implementation of a 32-bit Reduced Instruction Set Computer (RISC) processor on a Field Programmable Gate Arrays (FPGAs). The processor has been designed with VHDL, synthesized using Xilinx ISE 9.1i Webpack, simulated using ModelSim simulator, and then implemented on Xilinx Spartan 2E FPGA that has 143 available Input/Output pins and 50MHz clock oscillator. The test bench waveforms for the different parts of the processor are presented and the system architecture is demonstrated.

Keywords: *FPGA, CPU, VHDL, RISC, Processor.*

1. Introduction

Computer Engineering and Computer Design are very much concerned with the cost and performance of components in the implementation domain. Reduced Instruction Set Computer (RISC) focuses on reducing the number and complexity of instructions in the machine [1, 2]. Field Programmable Gate Arrays (FPGAs) are growing fast with cost reduction compared to ASIC design [3]. In this paper a low cost 32-bit RISC Processor has been designed and synthesized, the design has been described using VHDL, and some components have been implemented and tested on Xilinx FPGA [4, 5, 6, and 7]. Spartan 2E development board, DIO1, and DIO2 extension boards from Digilent have been used for the hardware implementation. The Webpack from Xilinx and ModelSim have been used for synthesis and simulation.

The text in this article is organized as follows; the introduction is given in section I; section II is talking about the system architecture; the design of the Control Unit is given in section III; in section IV we define the main structure of the DataPath; the design of the ROM is given in section V; section VI will presents the simulation results for the different parts of the processor; the conclusion and future work will be given at the end in section VII.

2. System Architecture

The RISC processor presented in this paper consists of three components as shown in Figure .1, these components are, the Control Unit (CU), the DataPath, and the ROM. The Central Processing Unit (CPU) has 17 instructions. In the following sections we will describe the design of the three main components of the processor.

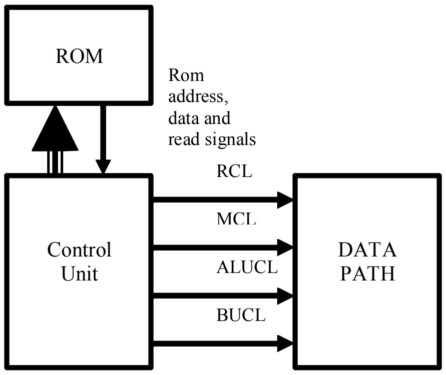

Figure (1) System Architecture

3. Design of the Control Unit

The control unit design is based on using FSM (Finite State Machine) and we designed it in a way that allows each state to run at one clock cycle, the first state is the reset which is initializes the CPU internal registers and variables. The machine goes to the reset state by enabling the reset signal for a certain number of clocks. Following the reset state would be the instruction fetching and decoding states which will enable the appropriate signals for reading instruction data from the ROM then decoding the parts of the instruction. The decoding state will also select the next state depending on the instruction, since every instruction has its own set of states, the control unit will jump to the correct state based on the instruction given. After all states of a running instruction are finished, the last one will return to the fetch state which will allow us to process the next instruction in the program. Figure .2 shows the state diagram for the control unit.

4. Design of the DataPath

The Data Path consists of subunits that are necessary for performing all of arithmetic and logic operations. A Datapath is a hardware that performs data processing operations [8, 9, 10, and 11]. It is one of two types of modules used to represent a digital system, the other being a control unit. The Datapath model we designed consists of the units necessary to perform all the operations on the data selected by the control unit. The components include a Register File, Arithmetic/Logic Unit, Memory Interface and Branching Unit as shown in figure .3. The *Register File* holds the table of the 32 general purpose registers available to the CPU, it has two output ports (output1,outpu2) and one input port, also it has a 16 bit bus connected directly to the Control Unit to pass immediate data. The *ALU* design consists of two input ports and one output port which mainly performs operations on two operands. It has a design similar to the control unit which selects an operation based on a code given by the ALUCL. The *Memory Interface* was designed to accommodate simple load/store operations with the 16x32 memory. The effective address is calculated by adding the content of the address register and the immediate data. The *Branch Unit* calculates a given condition by the control unit and raises a branch flag whether the condition is met or not, and if the flag is raised, it sends the branch address back to the control unit in order to replace the program counter. The control lines coming from the control unit operate all the units in the datapath. The path starts from the register file that has two output ports which are connected to all the other units, after that the processing is done by one of the other units then finally returned back to the register files input port using the multiplexer. The signals used in the datapath are forwarded from the control unit to each subcomponent as needed.

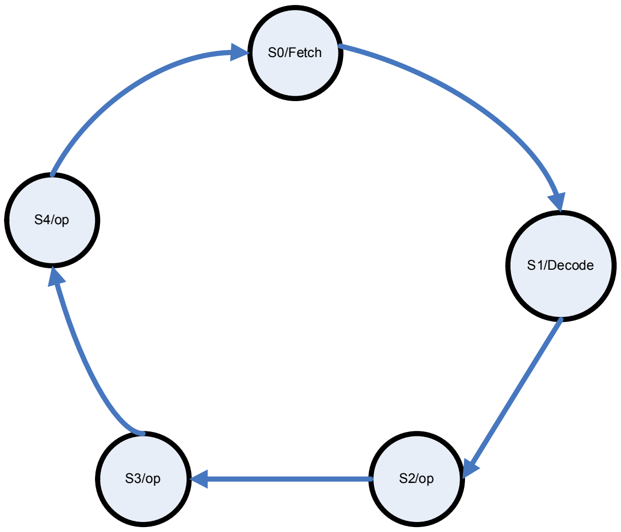

Figure (2) State Diagram for the CU Design

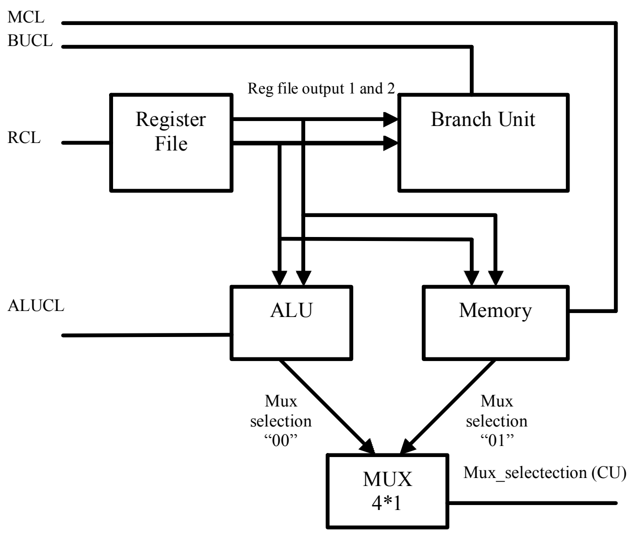

Figure (3) the Sub-components of the DataPath

5. Design of the ROM

The CPU has a built in ROM which enables us to program simple code and execute it. It is a basic 16x32 ROM and it is 32bit aligned. The List of signals in the ROM are:

- address: address sent by the control unit
- data_out: the data that is contained the given address
- read: the signal to enable reading from the ROM
- ready: the signal to indicate when the ROM is ready for reading
- clk: the main clock signal
- reset: the initial reset signal

5. Simulation Results

In this section we are going to show some test bench waveforms that will verify the working operation of our RISC Processor. In Figure .4, the following simple program was hard coded and simulated on the CPU:

LI r2,#0000, SHW r2, r1, LI r2,#0030, LI r1,#0000, SHW r1,r1, LI r1,#0001, LI r3,#0000
SHW r3,r3, LI r3,#0008, LI r4,#0000, SHW r4,r4, LI r4,#0004, ADD r4,r4,r1, BRAG r2,r3,r4

There are 5 main signals that are viewed in throughout the simulation. The sim_clock signal is the clock generated for the simulation and runs at 50Mhz, instruction_fetch signal shows when the control unit requests data from the ROM, the instruction_address 32bit bus is the address of the instruction being fetched, the instruction_data 32bit bus is the data sent out from the ROM, and the reset state is enabled for 3.5 cycle to give enough time for all units to reset and initialize, after that we can see the first instruction beginning at address 0 is executed followed by all the proceeding instructions until the instruction at address 40 Which is the shift half word "SHW". Figure .5 shows the simulation results of the ALU, in which we are attempting to use the addition operation to add the values (44 + 22), we send the opcode of the add operation (33) through the ALUCL signal and the result (66) will be in the alu_output_32bit bus. In Figure .6, the testbench waveform for the Memory Unit is shown, in which we demonstrate how the unit can be used to read some data from the RAM, our RAM is a 16 slot array of 32bit vectors, the following data are stored in the RAM for simulation purposes; "0x12345678" is stored in location "00", and "0x11133344" is stored in location "04". Figure .7 shows the simulation results of the Register File Unit. In this simulation, we demonstrate how to send data to a specific register in the file and store it, we tried to store the value '48' into register 2, the reg_test signal shows any data written to the input port of the register file.

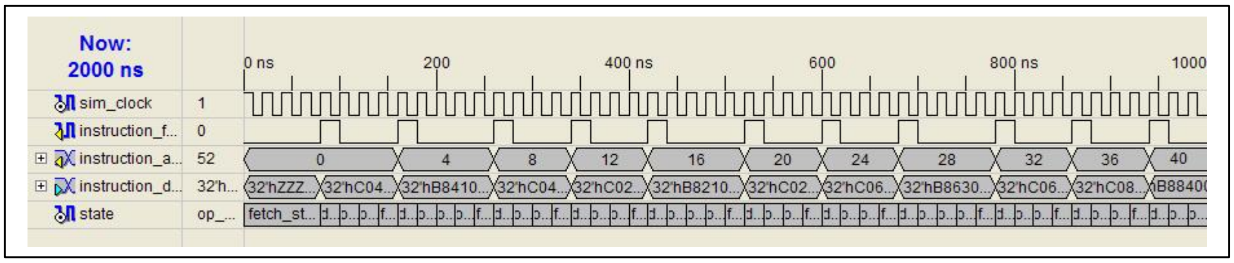

Figure (4) Simulation Results of a Simple Program Execution

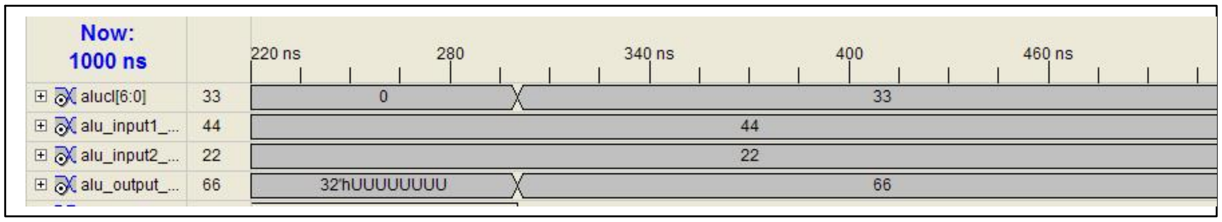

Figure (5) Simulation Results for the ALU

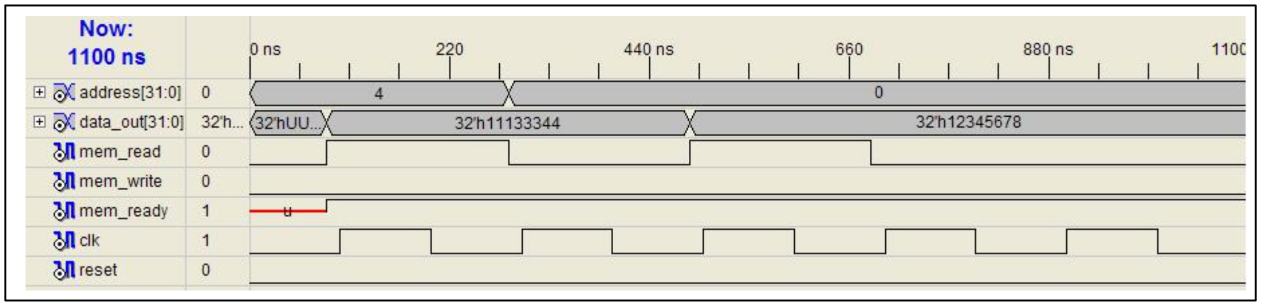

Figure (6) Simulation Results for the Memory Unit

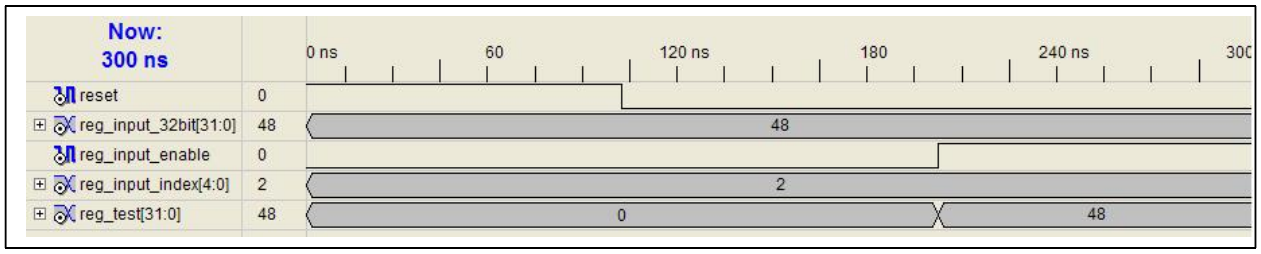

Figure (7) Simulation Results for the Register File Unit

4. Conclusion and Future Work

32-bit RISC Process has been design and implemented in hardware on Xilinx Spartan 2E FPGA. The design has been achieved using VHDL and simulated with ModelSim. Digilent Spartan 2E development board has been used for the hardware part. Most of the goals were achieved and simulation shows that the processor is working perfectly, but the Spartan 2E FPGA was not sufficient for implementing the whole design into a real hardware, since the total available logic gate in Spartan 2E is 200K Logic Gate, which was not enough for implementing the whole processor, but parts of the processor have been implemented and test in a real hardware. Future work will be added by increasing the number of instructions and make a pipelined design with less clock cycles per instruction and more improvement can be added in the future work.

References

1. John L. Hennessy, and David A. Patterson, "Computer Architecture A Quantitative Approach", 4th Edition; 2006.
2. Vincent P. Heuring, and Harry F. Jordan, "Computer Systems Design and Architecture", 2nd Edition, 2003.
3. Wayne Wolf, FPGA-Based System Design, Prentice Hall, 2005.
4. Dal Poz, Marco Antonio Simon, Cobo, Jose Edinson Aedo, Van Noije, Wilhelmus Adrianus Maria, Zuffo, Marcelo Knorich, "Simple Risc microprocessor core designed for digital set-top-box applications", Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors, 2000, p 35-44.
5. Brunelli Claudio, Cinelli Federico, Rossi Davide, Nurmi Jari, "A VHDL model and implementation of a coarse-grain reconfigurable coprocessor for a RISC core", 2nd Conference on Ph.D. Research in MicroElectronics and Electronics - Proceedings, PRIME, 2006, p 229-232.
6. Rainer Ohlendorf, Thomas Wild, Michael Meitinger, Holm Rauchfuss, Andreas Herkersdorf, "Simulated and measured performance evaluation of RISC-based SoC platforms in network processing applications", Journal of Systems Architecture 53 (2007) 703–718.
7. Luker, Jarrod D., Prasad, Vinod B., "RISC system design in an FPGA", MWSCAS 2001, v2,2001,p532-536.
8. Jiang, Hongtu; "FPGA implementation of controller-datapath pair in custom image processor design"; IEEE International Symposium on Circuits and Systems - Proceedings; 2004, p V-141-V-144.
9. K. Vlachos, T. Orphanoudakis, Y. Papaeftathiou, N. Nikolaou, D. Pnevmatikatos, G. Konstantoulakis, J.A. Sanchez-P., "Design and performance evaluation of a Programmable Packet Processing Engine (PPE) suitable for high-speed network processors units", Microprocessors and Microsystems 31, 2007, p 188–199.
10. Lou Dongjun, Yuan Jingkun, Li Daguang, Jacobs Chris, "Datapath verification with SystemC reference model", *ASICON 2005, 6th* International Conference on ASIC, 2005, Proceedings, v 2, p 906-909.
[11] Jiang Hongtu, Owall Viktor, "FPGA implementation of controller-datapath pair in custom image processor design", IEEE International Symposium on Circuits and Systems, Proceedings v 5, p V-141-V-144.