Single core hardware modeling of 32-bit MIPS RISC processor with a single clock

Mamun Bin Ibne Reaz
Universiti Kebangssan Malaysia
394 PUBLICATIONS 7,059 CITATIONS
SEE PROFILE

Jubayer Jalil
Universiti Kebangsaan Malaysia
18 PUBLICATIONS 321 CITATIONS
SEE PROFILE

Labonnah Farzana Rahman
Universiti Kebangsaan Malaysia
69 PUBLICATIONS 415 CITATIONS
SEE PROFILE

Research Journal of Applied Sciences, Engineering and Technology 4(7): 825-832, 2012
ISSN: 2040-7467
© Maxwell Scientific Organization, 2012
Submitted: December 06, 2011 Accepted: January 10, 2012 Published: April 01, 2012

Single Core Hardware Modeling of 32-bit MIPS RISC Processor with A Single Clock

M.B.I. Reaz, J. Jalil and L.F. Rahman
Department of Electrical, Electronic and Systems Engineering,
Universiti Kebangsaan Malaysia, 43600, UKM, Bangi, Selangor, Malaysia

Abstract: This study describes a design methodology of a single clock cycle MIPS RISC Processor using very high speed hardware description language (VHDL) to ease the description, verification, simulation and hardware realization. The RISC processor has fixed-length of 32-bit instructions based on three different format R-format, I-format and J-format, and 32-bit general-purpose registers with memory word of 32-bit. The MIPS processor is separated into five stages: instruction fetch, instruction decode, execution, data memory and write back. The control unit controls the operations performed in these stages. All the modules in the design are coded in VHDL, as it is very useful tool with its concept of concurrency to cope with the parallelism of digital hardware. The top-level module connects all the stages into a higher level. Once detecting the particular approaches for input, output, main block and different modules, the VHDL descriptions are run through a VHDL simulator, followed by the timing analysis for the validation, functionality and performance of the designated design that demonstrate the effectiveness of the design.

Key words: MIPS RISC processor, register, VHDL

INTRODUCTION

Increasing performance and gate capacity of recent FPGA devices permits complex logic systems to be implemented on a single programmable device. Such a growing complexity demands design approaches, which can cope with designs containing hundreds of thousands of logic gates, memories, high-speed interfaces, and other high-performance components. One category of such design approaches is design methodologies based on language VHDL (Sharma *et al*., 2009; Mane *et al*., 2006). It allows designers to develop hardware systems at high level (Sulik *et al*., 2000).

RISC or Reduced Instruction Set Computer is a design philosophy that become mainstream in the last few years, as the quest for raw speed has dominated the highly competitive computer industry. There is a desire to enhance the processor's speed and simplify the hardware for reasons of cost. RISC design resulted in computers that execute instructions faster than other computers built of the same technology (Charles and Veljko, 1989).

In a RISC machine, the instruction set is based upon a load/store approach. Only load and store instructions access memory. No arithmetic, logic or I/O instruction operates directly on memory contents. This is the key to single-cycle execution of instructions. The simplification results in an instruction decoder that is small, fast and relatively easy to design (Charles and Veljko, 1989). Due to the robust performance, simple instruction formats, breadth of products and depth of support, the MIPS RISC processor architecture becomes the market leader in high performance embedded 32-bit and 64-bit RISC processors (Sharma *et al*., 2009; White paper, 2011).

The use of VHDL for modeling is especially appealing since it provides a formal description of the system and allows the use of specific description styles to cover the different abstraction levels (architectural, register transfer and logic level) employed in the design (Yasin *et al*., 2004; Reaz *et al*., 2007; Chen *et al*., 1993). In the computation of method, the problem is first divided into small pieces; each can be seen as a submodule in VHDL. Following the software verification of each submodule, the synthesis is then activated. It performs the translations of hardware description language code into an equivalent netlist of digital cells. The synthesis helps integrate the design work and provides a higher feasibility to explore a far wider range of architectural alternative (Akter *et al*., 2008; Reaz *et al*., 2003; Alvarez *et al*., 2011).

In this study a design study of a single clock cycle MIPS RISC processor using VHDL has been presented. The goal of this work was to evaluate the feasibility of using VHDL for rapid design and prototyping of microprocessors.

Corresponding Author: M.B.I. Reaz, Department of Electrical, Electronic and Systems Engineering, Universiti Kebangsaan Malaysia, 43600, UKM, Bangi, Selangor, Malaysia, Tel.: +603-89216316; Fax: +603-89216146

MATERIALS AND METHODS

The architecture of the MIPS RISC processor is designed based on three MIPS 32-bit instruction formats R-format, I-format and J-format illustrated in Table 1, which was conducted at the System Design lab, Universiti Kebangsaan Malaysia in the middle of 2010. The design of this project consists of 32-bit instructions and 32-bit datapath.

The implementation of RISC performs fetch, decode, and execute in one clock cycle. The single clock cycle MIPS RISC architecture is separated into five stages: instruction fetch, instruction decode, execute, data memory and write back.

Instruction fetch stage: The first stage of a MIPS RISC is Instruction Fetch (IF) stage. The IF stage obtain the requested instruction from memory. The operation of the IF stage starts when the Program Counter (PC) a 32-bit register is sent out to fetch the instruction from memory into the Instruction Register (IR) and the PC is incremented by 4 through an adder to address the next sequential instruction. The IR is used to hold the instruction needed on subsequent clock cycles. A block diagram of IF stage is shown in Fig. 1.

The branch instruction has three operands, two registers that compare the equality and a 16-bit offset to compute the branch target address relative to the branch instructon address. The branch target address is computed by adding the instruction's sign-extended offset field to the PC.

The AND gate in the instruction fetch cycle is set to 1 when branch and zero are both true and the input Addresult from the ALU unit fed into the PC as the next target address else the normal incremented PC replace the current.

Instruction decode stage: When the instruction is fetched from the IF stage, the instruction's opcode is sent to the control unit and the function code is sent to the ALU control unit. The instruction's register address fields are used to address the two-port register file. The two-port register performs two independent reads and one write in one clock cycle. Hence implements the decode operation.

In this stage, the instructions are decoded and the register file is accessed to read the registers. The outputs of the general-purpose registers are read into two temporary registers, register 1 and register 2, for use in later clock cycles. The lower 16 bits of the IR are sign-extended and stored into the temporary register IMM, for use in the next cycle. The processor's 32 registers are stored in 'register file' that contains the register state of the machine.

For R-format instructions, there are three register operands. Two data words are read from the register file

Table 1: MIPS 32-bit instruction formats

Field size	6-bits	5-bits	5-bits	5-bits	5-bits	6-bits
R-Format	Opcode	Rs	Rt	Rd	Shift	Function
I-Format	Opcode	Rs	Rt	Address/immediate value		
J-Format	Opcode	Branch target address				

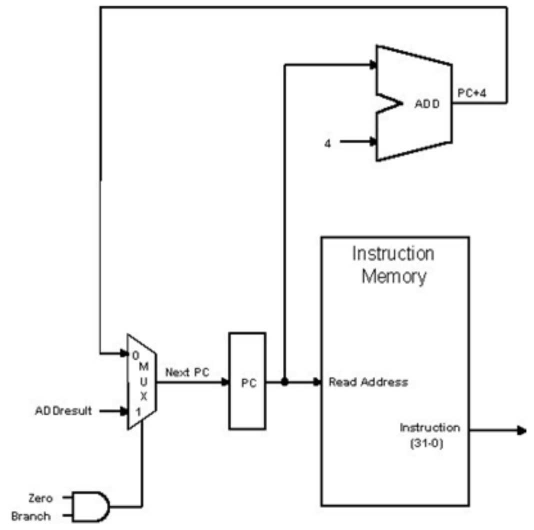

Fig. 1: Instruction Fetch Stage

Table 2: The three instruction formats

a. R-type instruction						
Field	0	rs	rt	rd	Shamt	Funct
Bit positions	31-26	25-21	20-16	15-11	10-6	5-0
b. Load or store instruction						
Field	35 or 43	rs	rt	address		
Bit positions	31-26	25-21	20-16	15-0		
c. Branch instruction						
Field	4	rs	rt	address		
Bit positions	31-26	25-21	20-16	15-0		

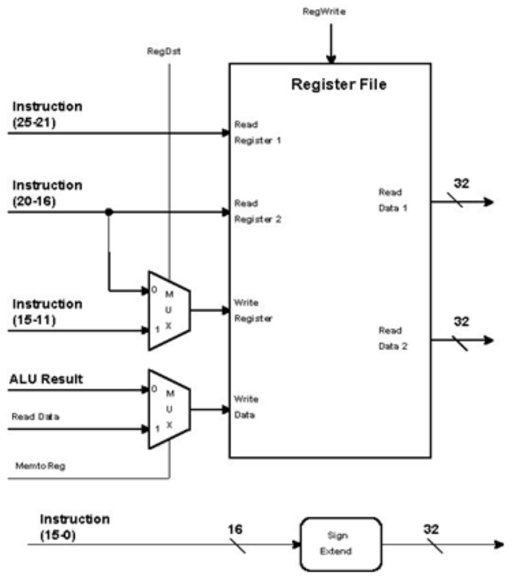

Fig. 2: Architecture of instruction decode stage

Table 3: The values of three ALU control lines and corresponding ALU operations

ALU control inputs	Function
000	AND
001	OR
010	Add
110	subtract
111	Set on less than

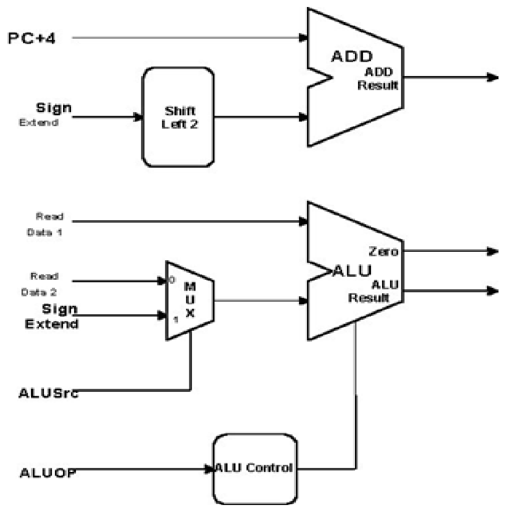

Fig. 3: Architecture of Execution Stage

and one data word is written into the register file for each instruction. Four inputs: three for register numbers and one for data and two outputs (both for data) are used. The register number inputs are 5 bites wide to specify one of the 32 registers, whereas the data input and two data output buses are each 32 bits wide. The block diagram of Instruction decode is shown in Fig. 2 and the format of the three instruction classes of MIPS: the R-type, branch and load/store instructions are tabulated in Table 2.

In the VHDL coding, D flip-flop with enable is used to solve the problem of unstable transients so that it is sensitive to its excitation inputs only during rising or falling of transitions of the clock. In this design, positive edge triggered is used.

Execution stage: After the instruction decode stage, the function code is sent to the execution stage. The execution stage performs calculations. The components in this stage are a data ALU and a branch address adder. The ALU is made up of arithmetic, logic and shifting capabilities and the branch address adder is used for PC-relative branch instructions.

To compute the branch target address, the branch datapath in the execution stage includes a sign extension unit and an adder. To perform the comparison, two register operands are fed into the ALU unit with a control set to do a subtraction operation. A block diagram of execution stage shown in Fig. 3.

The ALU control: The ALU control unit has 5-bit instruction function field and a 2-bit control field as input ALUOp. The output of the ALU control unit is a 3-bit signal that directly controls the ALU by generating one of the five 3-bit combinations as shown in Table 3.

Table 4 shows the setup of the ALU control inputs based on the 2-bit ALUOp control and the 6-bit function code for the R-type instruction and Table 5 shows the partial truth table for the three ALU control bits.

The Karnaugh maps (K-maps) are used to further simply the truth table in designing the logic. A partial VHDL code that generate the ALU control bits are shown below:

```
ALUctl(0) <= (Func_op(0) OR Func_op(3)) AND
ALUOp1

ALUctl(1) <= (NOT Func_op(2)) OR (NOT
ALUOp1)

ALUctl(2) <= (Func_op(1) AND ALU Op1) OR
ALUOp0
```

ALU operations: To compute the branch target address, the sign extend bits are shifted left two bits. This process is done by concatenating the lower 6 bits of the sign extend bits with $00_{2.}$ Then, these shifted bits are added with the PC+4 from the Instruction Fetch stage.

Data memory stage: The data memory stage stores and loads values to and from memory. This stage is active during the branch instruction. Data Memory takes two inputs for the address and the write data, and one output for the read. Two separate read and write controls either MemWrite or MemRead asserted on any given clock.

Memory read: To load an instruction the read memory control MemRead is asserted to enable the data to be read from the data memory. The data memory takes an address from one of the inputs. The data is then sent out from the data memory to register file through its output Read Data.

While developing the VHDL codes for Data Memory stage, the memory is initialized to some value to ease the simulation process and error checking. A partial code is shown below.

```
address <= readadd(2 downto 0);

mem_out <= mem0 WHEN address = To
_stdlogicvector (B"000") ELSE
mem1 WHEN address=To_stdlogicvector(B"001")
ELSE
mem2 WHEN address = To_stdlogicvector(B"010")
ELSE
To_stdlogicvector(X"FF");
```

Table 4: Setup of ALU control inputs

Instruction opcode	ALU Op	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	010
SW	00	store word	XXXXXX	add	010
Branch equal	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	and	100100	and	000
R-type	10	or	100101	or	001
R-type	10	set on less than	101010	set on less than	111

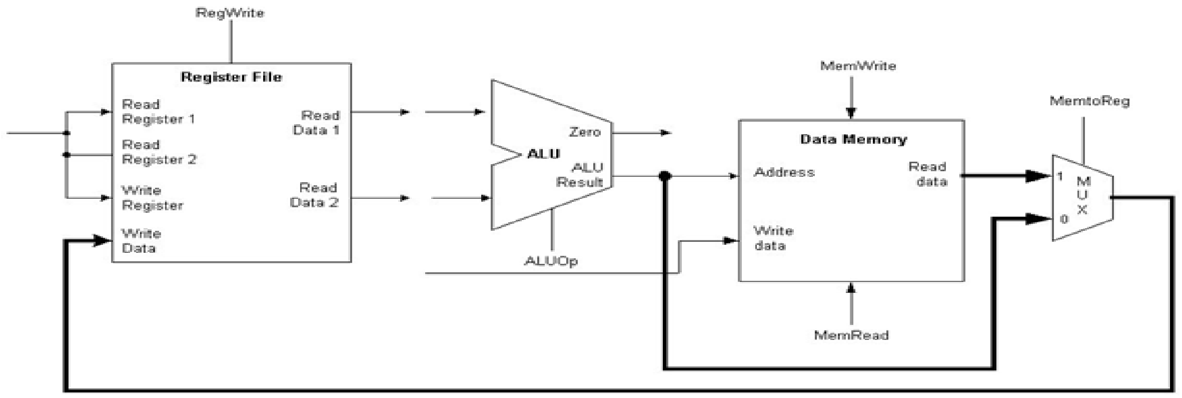

Fig. 4: Architecture of the Write Back Stage

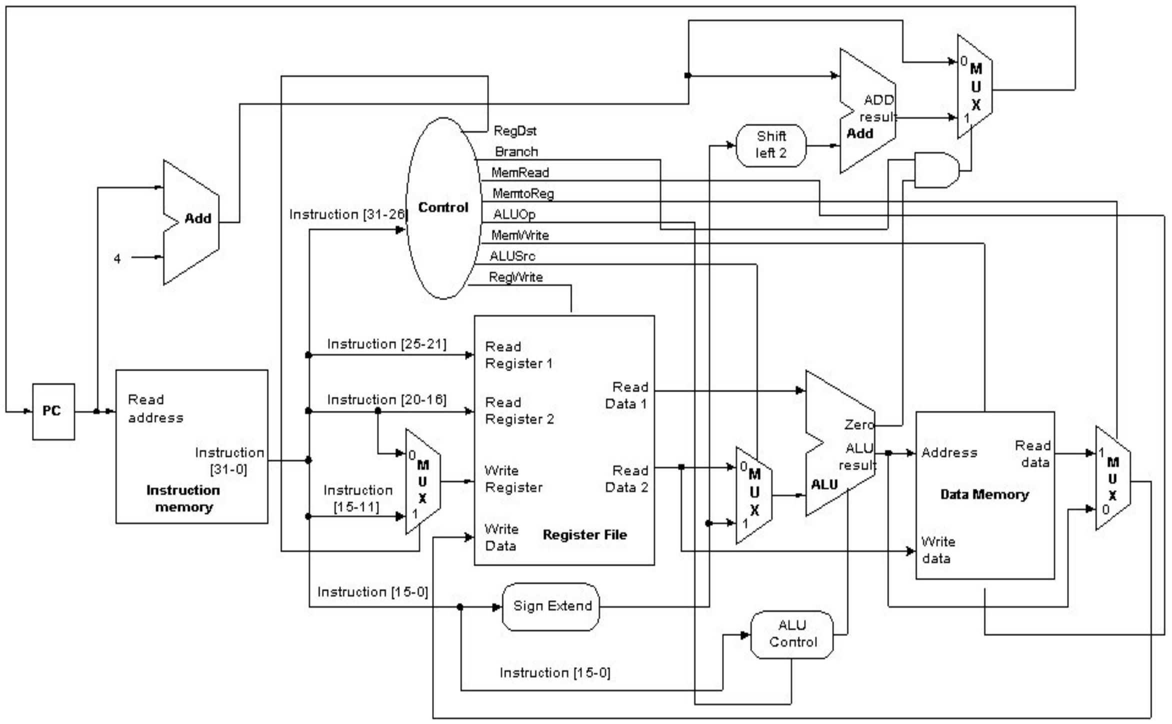

Fig. 5: The Simple Datapath with the Control Unit

to written into the data memory. The data memory takes an address from one of the inputs. The data to be written into he data memory is sent into it through the input Write Data.

In VHDL coding, the write control signal of the corresponding memory asserted in order to begin the writing process into the memory. This is done by inserting an AND gate between the memWrite control signal and

Table 5: Truth table of ALU control bits as function of ALUOp and function code field

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Table 6: Control signals in each RISC stage

MIPS RISC stage	Control signals
Instruction decode	RegDst
	RegWrite
Execution	ALUSrc
Data memory	MemWrite
	MemRead
Write back	MemtoReg

Memory write: To store an instruction the write memory control MemWrite is asserted to enable the data to be the address that indicates the memory is to be written. Then, the data on the write data bus is written into the corresponding memory location.

Write back stage: The Write Back stage writes the result of a calculation, memory access or input into the register file. The component multiplexor determines whether the data from data memory or the ALU result from the ALU to be written back into the register file. The operation of this multiplexor is controlled by signal MemtoReg.

The operations involved in this stage are handled by instructions Register-Register ALU instruction and Load instruction. For Register-Register ALU instruction, the MemtoReg is deasserted. For Load instruction, the MemtoReg is asserted. When it is asserted, the value, which comes out from the Read Data of data memory is selected that becomes the input to the write data of the register file. The block diagram of write back stage is shown in Fig. 4.

Control unit: In every stage of MIPS RISC, there are some control signals that controls the operations of each of the stages that illustrated in Table 6. Table 7 shows the effect of each of the seven control signals. The control unit is able to take inputs and generate a write signal for each state element, the selector control for each multiplexor, and the ALU control.
The different stages of MIPS RISC are combined to make a datapath for MIPS architecture. Figure 5 shows the datapath obtained by composing the separate stages.

Since the setting of the control lines depends only on the opcode, each control signal is either 0, 1 or don't care

Table 7: Effect of each of the seven control signals

Sinal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20-16).	The register destination number for the Write register comes from the rd field (bits 15-11)
Reg Write	None	The register on the Write register input is written with the value on the Write data input
ALUSrc	The second ALU operand comes from the second register file output (Read data 2)	The second ALU operand is the sign-extended ; lower 16 bits of the instruction
PCSrc	The PC is replaced by the output of the adder that computes the value of PC+4.	The PC is replaced by the output of the adder that computes the branch target
MemRead	None	Data memory contents designed by the address input are put on the Read data output
MemWrite	None	Data memory contents designated by the address input are replaced by the value on the Write data input
MemtoReg	The value fed to the register Write data input comes from ALU	The value fed to the register Write data input comes from the data memory

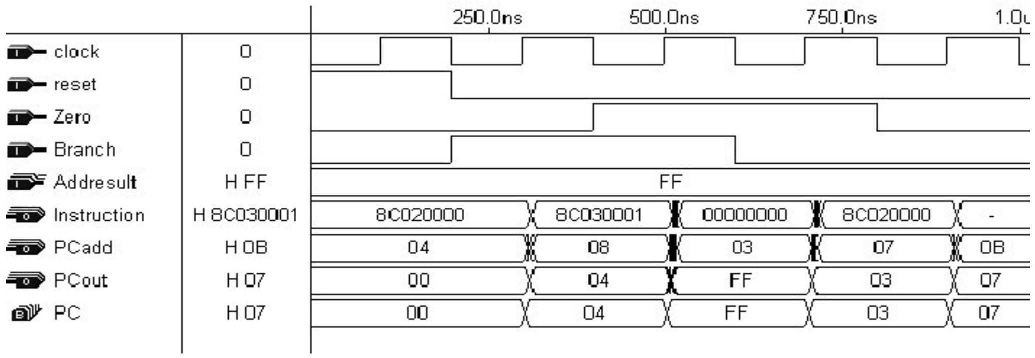

Fig. 6: Simulation of instruction fetch stage

Table 8: Setting of the control lines

Instruction	Reg dst	ALUSrc	Memto-reg	Reg write	Mem read	Mem write	Branch	ALU Op1	ALU Op0
R-format	1	0	0	1	0	0	0	1	0
Lw	0	1	1	1	1	0	0	0	0
Sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

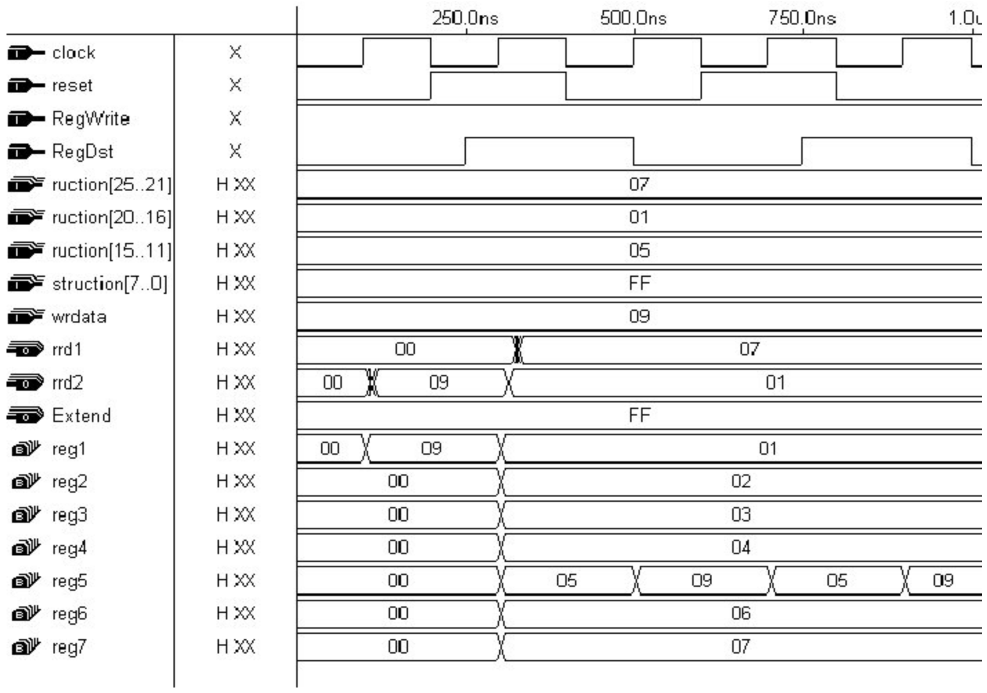

Fig. 7: Simulation of instruction decode stage

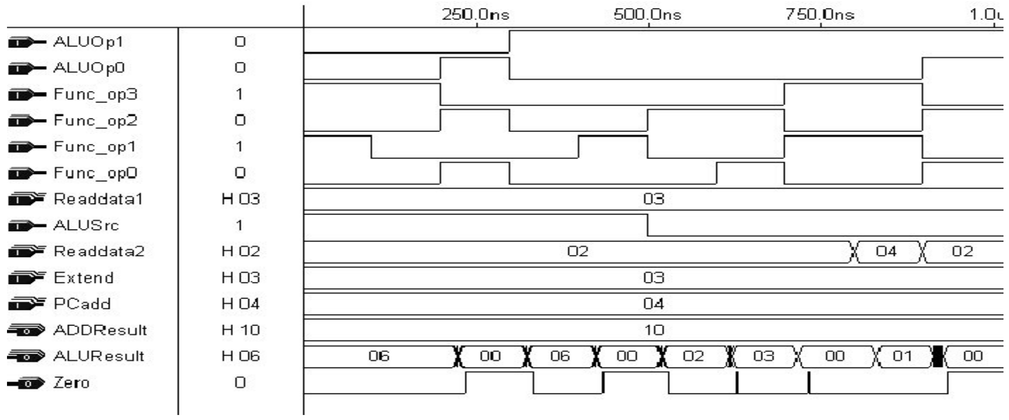

Fig. 8: Simulation waveform of execution stage

(X), for each of the opcode values. Table 8 defines the control signals that set for each opcode.

While developing the codes for control unit , the LCELL function is used to isolate logic blocks in the MIPS design. The LCELL component is a buffer that allocates a logic cell for all the files that are associated with the RISC design. The LCELL buffer produces the true and complement of a logic function.

RESULTS AND DISCUSSION

In this project the compilation and simulation of the model is run using MAX+PLUS II version 9.23. Simulation of design is done in a bottom-up fashion to verify the correctness of design. Small modules are simulated in separate testbenches before they are integrated. There is one testbench for each small module

Table 9: Results of the top-level module simulation

PC	Out_nst	Op	Inst	Branch out	RegWrite out	MemWrite out	Rs	rrld bus	Rt	rr2d bus	Rd	ALU result	Zero out	Wrd _bus
00	8C020000	35	Lw	0	1	0	Reg0	00H	Reg2	02H	-	00H	1	55H
04	8C030001	35	Lw	0	1	0	Reg0	00H	Reg3	03H	-	01H	0	AAH
08	00430820	00	R	0	1	0	Reg2	55H	Reg3	AAH	Reg1	FFH	0	FFH
0C	AC010003	43	Sw	0	0	0	Reg0	00H	Reg1	FFH	-	03H	0	FFH
10	1022FFFF	4	Beq	1	0	0	Reg1	FFH	Reg2	55H	-	AAH	0	AAH
14	1021FFFA	4	beq	1	0	0	Reg1	FFH	Reg1	FFH	-	00H	1	00H

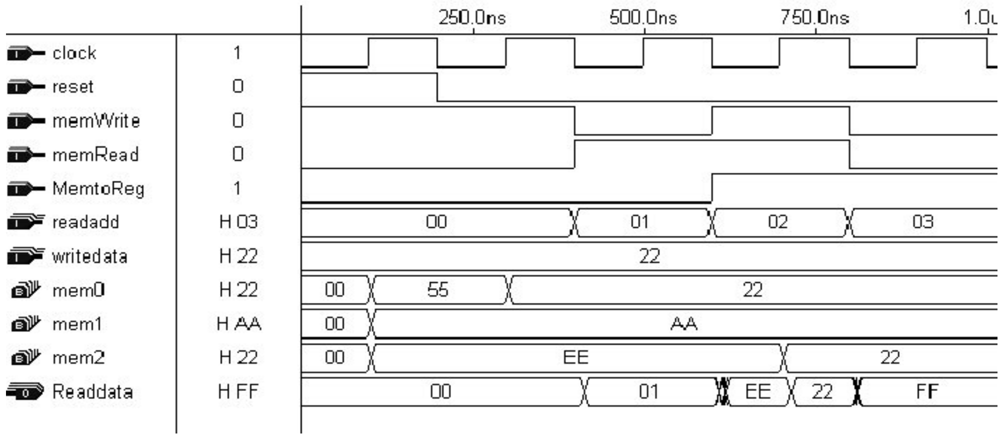

Fig. 9: Simulation of data memory and write back stage

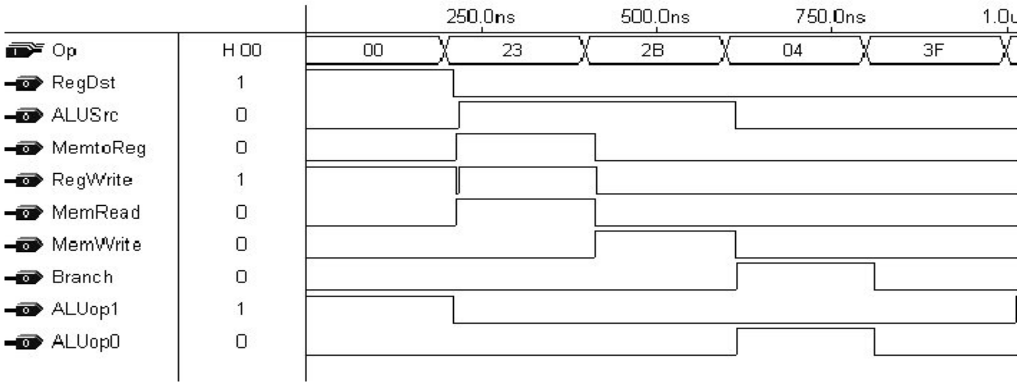

Fig. 10: Simulation waveform of control unit

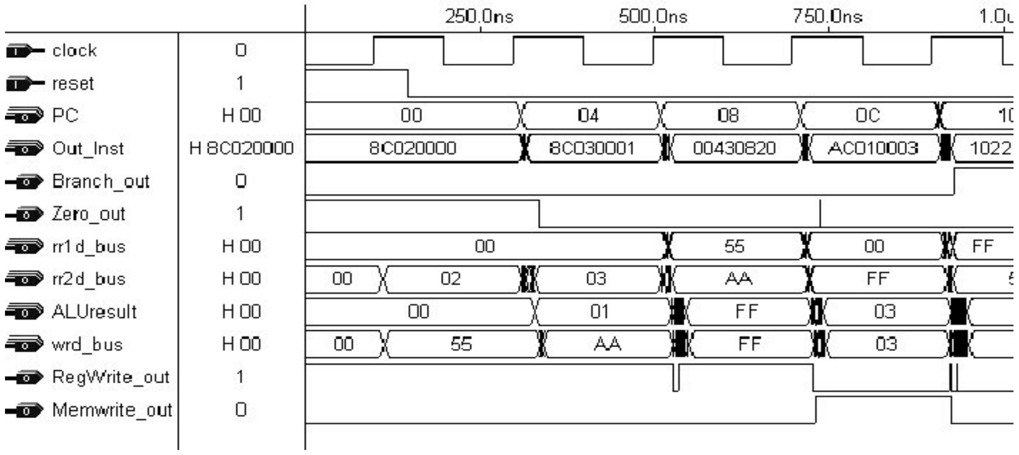

Fig. 11: Simulation waveform of top-level module

and a testbench for the complete device. Simulated waveforms for each stages are shown in Fig. 6 to 11.

The simulated results of the top-level module are tabulated in Table 9 that shows the correctness of the model.

CONCLUSION

The objective of this project was to hardware prototyping of a MIPS RISC processor. By simulating with various test vectors a single clock cycle 32-bit MIPS RISC processor using VHDL is successfully designed, implemented and tested. An 8-bit datapath is used with the intention of hardware realization onto Altera FLEX 10K FPGA chip. The modules were successfully compiled, simulated and synthesized. The hardware modeling and simulation demonstrated complete, correct functionality and met all the initial system requirements.

REFERENCES

Alvarez, J., O. Lopez, F.D. Freijedo and J. Doval-Gandoy, 2011. Digital Parameterizable VHDL Module for Multilevel Multiphase Space Vector PWM. IEEE Trans. Indus. Electron., 58(9): 3946-3957.

Akter, M., M.B.I. Reaz, F.M. Yasin and F. Choong, 2008. Hardware implementations of image compressor for mobile communications. J. Communi. Technol. Electron., 53(8): 899-910.

Charles, E.G. and M.M. Veljko, 1989. RISC Principles, Architecture and Design. Computer Science Press Inc.

Chen, S., B. Mulgrew and P.M. Grant, 1993. A clustering technique for digital communications channel equalization using radial basis function networks. IEEE Trans. Neural Networks, 4: 570-578.

Mane, P.S., I. Gupta and M.K. Vasantha, 2006. Implementation of RISC Processor on FPGA. In the IEEE International Conference on Industrial Technology, ICIT 2006, pp: 2096-2100.

Reaz, M.B.I., F. Choong, M.S. Sulaiman and F. Mohd-Yasin, 2007. Prototyping of Wavelet Transform Artificial Neural Network and Fuzzy Logic for Power Quality Disturbance Classifier. J. Electric Power Comp. Syst., 35(1): 1-17.

Reaz, M.B.I., M.T. Islam, M.S. Sulaiman, M.A.M. Ali, H. Sarwar and S. Rafique, 2003. FPGA Realization of Multipurpose FIR Filter, Parallel and Distributed Computing, Applications and Technologies, PDCAT Proceedings, pp: 912-915.

Sharma, R., V.K. Sehgal, N. Nitin, P.Bhasker and I. Verma, 2009. Design and Implementation of a 64-bit RISC Processor Using VHDL. In the 11th International Conference on Computer Modelling and Simulation, UKSIM '09, Cambridge, pp: 568-573.

Sulik, D., M. Vasilko, D. Durackova and P. Fuchs, 2000. Design of a RISC Microcontroller Core in 48 Hours. In Embedded Systems Show, London Olympia. Retrieved from: http: //wenku.baidu.com/view/b22c84fc910ef12d2af9e7b4, (Accessed on: Nov. 15, 2011).

White paper, 2011. The MIPS32-24KE Core Family: High Performance RISC Cores with DSP Enhancements. MIPS Technologies Inc. retrieved from: http://www.mips.com/media/files/white-papers/24ke.pdf, (Accessed on: Nov. 07, 2011).

Yasin, F.M., A.L. Tan and M.I. Reaz, 2004. The FPGA Prototyping of Iris Recognition for Biometric Identification Employing Neural Network. Proceedings of the 16th IEEE International Conference on Microelectronics, pp: 458-461.