A FPGA Implementation of a RISC Processor for Computer Architecture

Vijay R. Wadhankar
Mtech(VLSI-IV Sem)
G.H.Raisoni College of Engg
Nagpur

Vaishali Tehre
Asst. Professor
Dept of Electronics & Communication
G.H.Raisoni College of Engg
Nagpur

ABSTRACT
This paper is concerned with the design and implementation of a 32bit Reduced Instruction Set Computer (RISC) processor on a Field Programmable Gate Arrays (FPGAs). We are designing the processor with VHDL and the simulation using Altera Quartus Plus2, and we will implement on Altera cyclone II in FPGA.The test bench waveforms for the different parts of the processor are presented and the system architecture is demonstrated.

1. INTRODUCTION
Computer organization and design is a common engineering course where students learn concepts of modern computer architecture. This project targets the computer architecture courses and presents an FPGA (Field Programmable Gate Array) implementation design of a 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 gain a better understanding of both the RISC single-cycle and pipelined processor..

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 we are designing a low cost 32bit RISC Processor , the design has been described using VHDL, and some components have been implemented and tested on Altera FPGA [4, 5, 6, and 7]. CycloneII development board, extension boards from Digilent will be used for the hardware implementation.

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 the instruction fetch unit is given in section V Risc Instruction Set Architecture is given in section VI Data Memory is given and in section VII 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. PROPOSED SYSTEM ARCHITECTURE
The RISC processor presented in this paper consists of following components as shown in Figure .1, These components are, the Control Unit (CU), the DataMemory, and the Register Unit. The Central Processing Unit (CPU) has 17 instructions

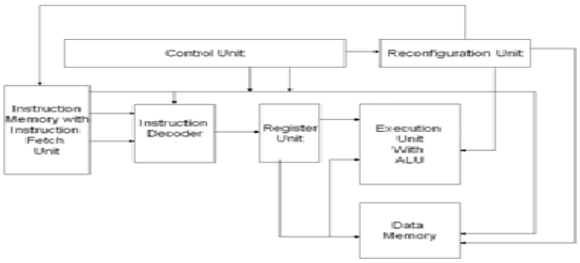
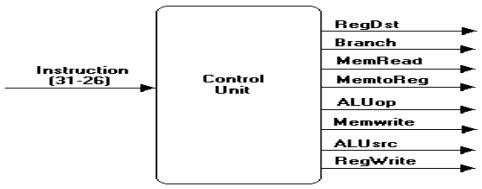


Fig1 Block Diagram of Proposed Architecture

3. DESIGN OF THE CONTROL UNIT
The control unit of the RISC single-cycle processor examines the instruction opcode bits [31 – 26] and decodes the instruction to generate nine control signals to be used in the additional modules as shown in Figure2. The RegDst 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 MemRead control signal is asserted during a load instruction when the data memory is read to load a register with its memory contents. The MemtoReg control signal determines if the ALU result or the data memory output is written to the register file. The ALUOp control signals determine the function the ALU performs. (e.g. and, or, add, sbu, slt) The MemWrite control signal is asserted when during a store instruction when a registers value is stored in the data memory. The ALUSrc control signal determines if the ALU second operand comes from the register file or the sign extend. The RegWrite control signal is asserted when the register file needs to be written. Table shows the control signal values from the instruction decoded.



4. 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 as shown in Figure3. 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 in VHDL: 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.

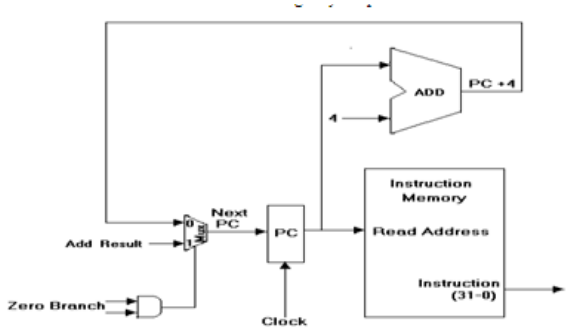


Fig3 RISC Instruction Fetch Unit

5. THE RISC INSTRUCTION SET ARCHITECTURE

The RISC Architecture defines thirty-two, 32-bit general purpose registers (GPRs). Register $r0 is hard-wired 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, 4, 8, …). RISC 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 . Table2 provides a description of each of the fields used in the three different instruction types.

RISC 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 and the base register in the $rs field to address the memory. Arithmetic instructions or R-type include: 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; this is considered PC-relative addressing.

Table shows a summary of the core RISC instructions.

Field	Description
opcode	6-bit primary operation code
rd	5-bit specifier for the destination register
rs	5-bit specifier for the source register
rt	5 bit specifier for the targetsource/destination)register or used to specify functions within the primary opcode REGIMM
Address/immediate	16 bit signed immediate used for logical operands,airthematic signed operands,load/store address byte offsets,and PC – relative branch signed instruction displacement.
sa	5 bit shift amount
function	6-bit function field used to specify functions within the primary opcode SPECIAL

Table1. RISC Instruction Fields

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				

TABLE2 RISC Instruction Types

Mnemonic	**Format**	**Opcode Field**	**Function Field**	**Instruction**
Add	R	0	32	Add
Addi	I	8	-	Add Immediate
Addu	R	0	33	Add Unsigned
Sub	R	0	34	Subtract
Subu	R	0	35	Subtract Unsigned
And	R	0	36	Bitwise And
Or	R	0	37	Bitwise OR
Sll	R	0	0	Shift Left Logical
Srl	R	0	2	Shift Right Logical
Slt	R	0	42	Set if Less Than
Lui	I	15	-	Load Upper Immediate
Lw	I	35	-	Load Word
Sw	I	43	-	Store Word
Beq	I	4	-	Branch on Equal
Bne	I	5	-	Branch on Not Equal
J	J	2	-	Jump
Jal	J	3	-	Jump and Link (used for Call)
Jr	R	0	8	Jump Register (used for Return)

Table3. RISC Processor Core Instructions.

6. RISC SINGLE-CYCLE PROCESSOR VHDL IMPLEMENTATION

The initial task of this project was to implement in VHDL the RISC single-cycle processor using Altera Quartus plus2 Text Editor to model the processor. The IEEE Standard VHDL Language Reference Manual [13], also helped in the overall design of the VHDL implementation. The first part of the design was to analyze the single-cycle datapath and take note of the major function units and their respective connections.

The RISC 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 RISC 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 RISC implementation include the Register File, Instruction Memory, and Data Memory as seen in Figure5.

It was determined that the full 32-bit version of the RISC architecture would not fit onto the chosen FLEX10K70 FPGA. The FLEX10K70 device 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 RISC requires no less than twelve EABs to support the processor's register file, instruction memory, and data memory. In order for our design to model that in, the data 20 width was reduced to 8-bit while still maintaining a full 32-bit instruction. This new design allows us to implement all of the processor's state elements using six EABs, which can be handled by the FLEX10K70 FPGA device. Even though the data width was reduced, the design has minimal VHDL source modifications from the full 32-bit version, thus not impacting the instructional value of the RISC VHDL model.

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 RISC 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. Figure4 shows the RISC 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 Reset signal is asserted. (e.g. $r1 = 1, $r2 = 2, etc.)

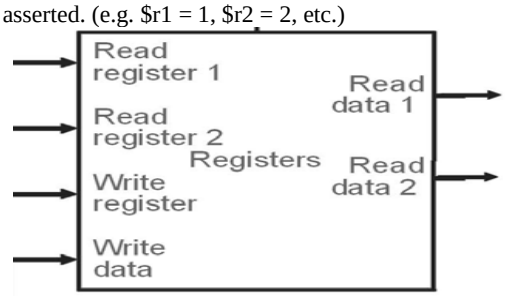


Fig4. RISC Register File

Altera Quartus Plus2 is packaged with a Library of Parameterized Modules (LPM) that allow one to implement RAM and ROM memory in Altera supported PLD devices. With our design this library was used to declare the instruction memory as a read only memory (ROM) and the data memory as a random access memory (RAM). Using the lpm_rom component from the LPM Library, the Instruction memory is declared as a ROM and the following parameters are set: the width of the output data port parameter lpm_width is set to 32-bits, the width of the address port parameter lpm_widthad is set to 8-bits, and the parameter lpm_file is used to declare a memory initialization file (.mif) that contains ROM initialization data. This allows us to set the indexed address data width to 8-bits, the instruction output to 32-bits wide, and enables us to initialize the ROM with the desired RISC program to test the RISC processor implementation. With these settings, four 256 x 8 EABs are required to implement the instruction memory. An example of the RISC instruction memory can be seen in Figure5 and the VHDL code implementation can be seen in Figure6.

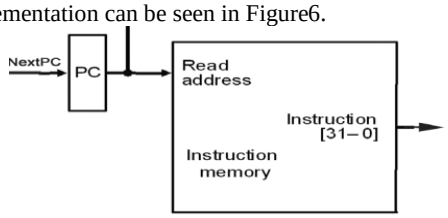


Fig 5 RISC Instruction Memory

```
Instr_Memory: LPM_ROM
GENERIC MAP(
LPM_WIDTH => 32,
LPM_WIDTHAD => 8,
LPM_FILE => "instruction_memory.mif",
LPM_OUTDATA =>
"UNREGISTERED",
LPM_ADDRESS_CONTROL =>
"UNREGISTERED")
PORT MAP (
```

Fig 6 VHDL – RISC Instruction Memory

The data memory is declared using the lpm_ram_dq component of the LPM library. This component is chosen because it requires that the memory address to stabilize before allowing the write enable to be asserted high. The input Address width (lpm_widthad) and the Read Data output width (lpm_width) are both declared as 8-bit wide, in lieu of our altered design. Using these settings allows us to use one 256 x 8 EAB instead of the 4 combined EABs required for the full 32-bit version of RISC. An example of the RISC data memory can be seen in Figure7 and the VHDL code implementation can be seen in Figure8.

Fig 7 RISC Data Memory

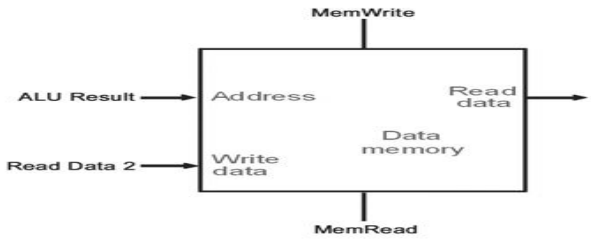
```
Data_Memory : LPM_RAM_DQ
GENERIC MAP(
LPM_WIDTH => 8,
LPM_WIDTHAD => 8,
LPM_FILE => "data_memory.mif",
LPM_INDATA => "REGISTERED",
LPM_ADDRESS_CONTROL =>
"UNREGISTERED",
LPM_OUTDATA => "UNREGISTERED")
PORT MAP(
inclock => Clock,
data => Write_Data,
address => Address,
we => LPM_WRITE,
q => Read_Data);
```

Table 4. VHDL – RISC Data Memory

7. DATA MEMORY UNIT

The data memory unit is only accessed by the load and store instructions. The load instruction asserts the MemRead 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 MemWrite signal and writes the data value previously read from a register into the computed memory address. Figure8 shows the signals used by the memory unit to access the data memory.



8. 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 Fig9.1 , the following simple program was hard coded and simulated on the CPU:

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 9.2 shows the simulation results of the cpu depending upon the opcode given.In which we are attempting to use the different opcodes depending upon the program written. And depending upon the opcode we get the result.In Figure 9.3 we get the simulation result of the instruction fetch.
In Fig9.4 the initialization of the instruction memory location is shown.

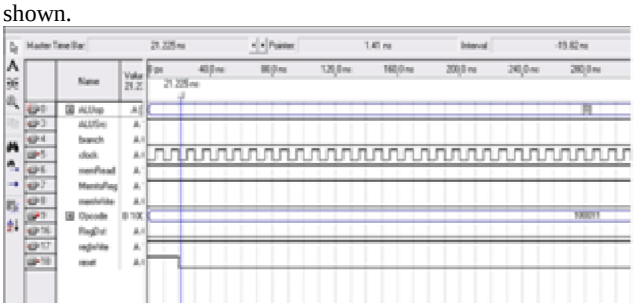


Fig9.1 Simulation Results for Control Unit Depending on opcode 100011

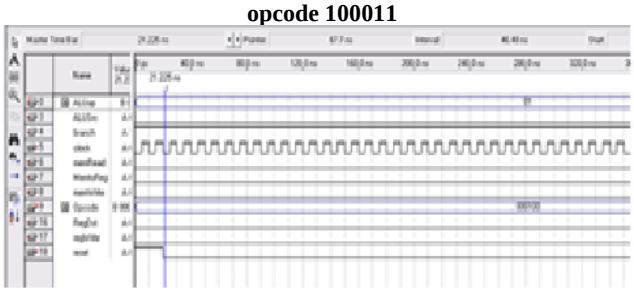


Fig9.2Simulation Results for Control Unit Depending on opcode 000100

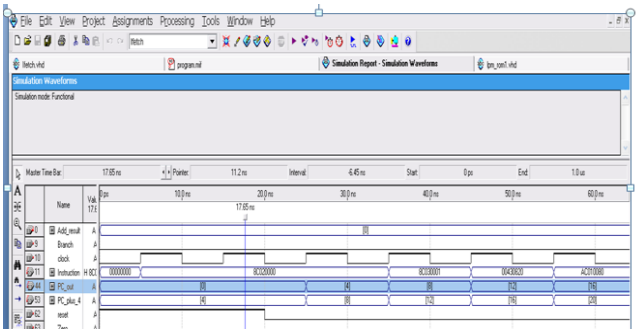


Fig9.3Simulation Results for Ifetch

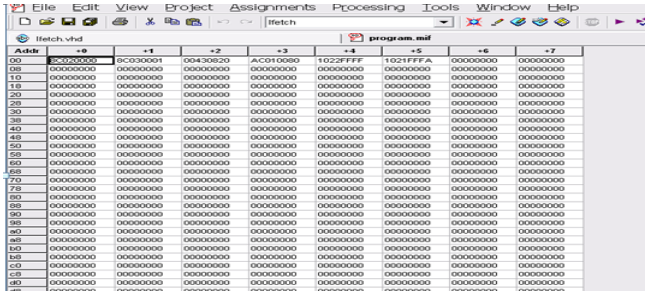


Fig9.4 Results for Instruction Memory

9. CONCLUSION AND FUTURE WORK

We are designing 32bit RISC Processor and implementing in hardware on Altera CycloneII in FPGA. The design has been achieved using VHDL and simulated with Altera Quartus Plus 2 development board has been used for the hardware part. Most of the goals were achieved and simulation shows that the processor is working perfectly, implemented and test in a real hardware..

In the Future we are trying to implement the reconfigurable unit which can reconfigure the data memory,instruction memory and may be in future to reconfigure ALU according to application requirement. Extra 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.In this project in the future we are going to reconfigure the data memory and instruction memory

10. 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, FPGABased 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 settopbox applications", Proceedings of the International Conference on Application Specific Systems, Architectures and Processors, 2000, p 3544.

[5] Brunelli Claudio, Cinelli Federico, Rossi Davide, Nurmi Jari, "A VHDL model and implementation of a coarsegrain reconfigurable coprocessor for a RISC core", 2nd Conference on Ph.D. Research in MicroElectronics and Electronics Proceedings,2006,

[6] Rainer Ohlendorf, Thomas Wild, Michael Meitinger, Holm Rauchfuss, Andreas Herkersdorf, "Simulated and measured performance evaluation of RISCbased 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,

[8] Jiang, Hongtu; "FPGA implementation of controllerdatapath pair in custom image processor design"; IEEE International Symposium on Circuits and Systems Proceedings;2004, p V141V144.

[9] Jiang Hongtu, Owall Viktor, "FPGA implementation of controllerdatapath pair in custom image processor design", IEEE International Symposium on Circuits and Systems, Proceedings v 5, p V141V144.