FPGA Implementation of A MIPS RISC Processor

V.N.Sireesha
Regd No:10331DS709, Dept. of ECE,
MVGR College of Engg, Vizianagaram India
sireeshavadlamani1989@gmail.com

D.Hari Hara Santosh
Asst.Prof., Dept. of ECE
MVGR College of Engg, Vizianagaram India
santosh_ece2005@gmail.com

Abstract—This project targets the 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). In this paper MIPS instruction format, instruction data path, decoder modules are analyzed. Furthermore, we design instruction fetch (IF) module of a CPU based on RISC CPU instruction set. Function of IF module mainly includes fetch instruction and latch module address arithmetic module check validity of instruction module and synchronous control module.

Keywords- XILINX 11.1; RISC; LUT; CPU; MIPS.

INTRODUCTION

Great disparity among instructions and low universal property of CISC (Complex Instruction Set Computer-CISC) result in instruction realization difficulty and long running-time cost. Comparing to CISC, RISC CPU have more advantages, such as faster speed, simplified structure easier implementation. RISC CPU is extensive use in embedded system. Developing CPU with RISC structure is necessary choice. MIPS (Microprocessor without Interlocked Pipeline Stages) is a RISC microprocessor architecture. The MIPS Architecture defines thirty-two, 32-bit general purpose registers (GPRs). Instruction Set Architecture (ISA) of processor is composed of instruction set and corresponding registers. Program based on same ISA can run on the same instruction set. MIPS instruction has been developed from 32-bit MIPS I to 64-bit MIPS III and MIPS IV since it was created. To assure downward compatibility, every generation production of MIPS instruction directly extends new instruction based on old instruction but not abnegates any old instruction, so MIPS processor of 64-bit instruction set can execute 32-bit instruction.

I. IMPLEMENTATION OF CONVOLUTION PROCESSOR

A. Instructions

There are three formats of MIPS instructions:

(i) Register Format (R-type)

<table>
<thead>
<tr>
<th>OPCODE (31 to 26)</th>
<th>RS (25 to 21)</th>
<th>RT (20 to 16)</th>
<th>RD (15 to 11)</th>
<th>Shift (10 to 6)</th>
<th>FUN (5 to 0)</th>
</tr>
</thead>
</table>

The first two 5-bit register specifications are the two read registers and the last 5-bit register specification is the destination register, that the instruction will use. The last 6-bits in the instruction are used as function bits.

(ii) Immediate Format (I-type)

<table>
<thead>
<tr>
<th>OPCODE (31 to 26)</th>
<th>RS (25 to 21)</th>
<th>RT (20 to 16)</th>
<th>Address/Immediate Value (20 to 16)</th>
</tr>
</thead>
</table>

The I-type is similar to the R-type except the second read register and the 5 function bits are replaced by a 16-bit immediate value. Each I-type opcode can have only one instruction because it has no function bits like the R-type.

(iii) Jump Type Format (J-type)

The J-type format consists of a 6-bit opcode and remaining bit indicates branching address.
B. Architecture

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 [31-26] are sent to a control unit to determine the type of instruction to execute. The type of instruction then determines which control signals are to be asserted and what function the ALU is to perform, thus decoding the instruction. The instruction register address fields $rs bits [25 - 21], $rt bits [20 - 16], and $rd bits[15-11] are used to address the register file. 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. Data memory and Instruction memory are picked from Xilinx library.

II. RESULTS

Implementation is done using XILINX 11.1. RTL schematic and Floor plan view are shown in Fig.3 and Fig.4. Simulation snap is shown in Fig.2.
Hardware utilization summary is reported in Table I.

TABLE. I DEVICE UTILIZATION SUMMERY

<table>
<thead>
<tr>
<th>S.NO.</th>
<th>LOGIC UTILIZATION</th>
<th>UTILIZED HARDWARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>NUMBER OF SLICES</td>
<td>354</td>
</tr>
<tr>
<td>2.</td>
<td>NUMBER OF SLICE FLIP FLOPS</td>
<td>256</td>
</tr>
<tr>
<td>3.</td>
<td>NUMBER OF 4INPUT LUTS</td>
<td>458</td>
</tr>
<tr>
<td>4.</td>
<td>NUMBER OF BONDED IOBs</td>
<td>85</td>
</tr>
</tbody>
</table>

III. CONCLUSION

In this paper, top-down design method adopted and VHDL is used to describe system. The structure and hierarchical of design is very clear. It is easy to edit and debug. Design of instruction fetch (IF) stage simulates, integrates and routes on XILINX 11.1. Data Memory and Instruction Memory are generated using XILINX COREGEN.

References