FPGA Implementation of 32-bit MIPS Processor with CISC Multiplication Operation

Anu Mariam John
Assistant Professor,
Dept. of Electronics and Communication Engineering
Mar Baselios College of Engineering and Technology,
Thiruvananthapuram, Kerala, India

Shilpi Varshney
Senior Design Engineer,
AMD Research & Development Ctr.
India Private Limited
Mindspace – Cyberabad, Madhapur, Hyderabad

***Abstract*—MIPS architecture is one of the first commercially available RISC processor. MIPS stands for 'Microprocessor without Interlocked Pipeline Stages'. In a normal MIPS RISC architecture, for 32-bit multiply operation it can hold the processor for more than 32 clock cycles, which affects the processor performance. In order to avoid this problem, here we have implemented 32-bit MIPS processor with one CISC operation for multiplication which is realized using a Booth multiplier. Processor is tested in Xilinx Nexsys Spartan3 board, using a 177MHz clock frequency.**

***Keywords*—MIPS, ISA, Pipeline, booth multiplier**

I. INTRODUCTION

The Arithmetic and Logic Unit is the most important part of a processor, which executes all the arithmetic and logical operations. To make a processor with lighter hardware, the Arithmetic and Logic Unit(ALU) should be simple. But the instructions like multiply and divide takes many clock cycles with normal ALU. So it is better to implement those instructions separately with a dedicated hardware.

This paper implements a 32-bit 5-stage RISC Processor, with one CISC operation, which is implemented using Booth Multiplication Algorithm. This algorithm uses less number of cycles for execution than normal multipliers. A dedicated ALU is designed, for realizing the booth multiplier. The normal ALU does all other operations except multiplication.

Processor is designed using Verilog (IEEE 1364) Hardware Description Language (HDL) language. The design is synthesized in Xilinx ISE design suite 12.4. For implementing the processor, Xilinx Nexsys Spartan-3E board with FG320 package has been used.

The paper is organized as follows: Section II gives an Introduction to MIPS processor; Section III explains MIPS pipeline architecture and Instruction Set Architecture. In Section IV CISC Operation-multiplication algorithm is given. Section V deals with the FPGA Implementation of the proposed design. Section VI gives Simulation results and Section VII: conclusion and future scope.

II. MIPS PROCESSOR

MIPS processor, designed in 1984 by researchers at Stanford University, is typically a Reduced Instruction Set Computer (RISC) with Harvard architecture. Here we have implemented a 32-bit processor which follows MIPS Instruction Set architecture (ISA). Along with this 32-bit RISC processor, a CISC operation - multiplication have been incorporated, with normal ALU operation.

RISC processors typically support fewer and much simpler instructions. RISC architecture has a simpler hardware compared to CISC. The CISC operation adds more hardware to the design. Even if it increases hardware complexity, it reduces number of cycles required to execute a multiplication operation.

The ISA has 32-bit instructions, with 5-bit opcode, 5-bit each for registers and rest of the bits are for shift amount and function value in the case of R-type instructions, 16-bit data in the case of I-type instructions, and Jump address for J-type instructions.

The instruction execution in a processor can be split into a number of stages. As shown in Fig.1, in a MIPS processor there are 5 stages:

1. The Instruction Fetch stage fetches the next instruction from memory using the address in the Program Counter (PC) register and stores this instruction in the Instruction Register (IR).
2. The Instruction Decode stage decodes the instruction in the IR, calculates the next PC, and reads any operands required from the register file.
3. The Execute stage executes the above decoded instruction. ALU units are there in the execute stage.
4. The Memory Access stage performs any memory access required by the current instruction. For load instruction, it is to load an operand from the memory. For store instruction, it is to store an operand into memory.
5. For instructions that have a result which have to be written into a register, the Write Back writes this result back to the register file.

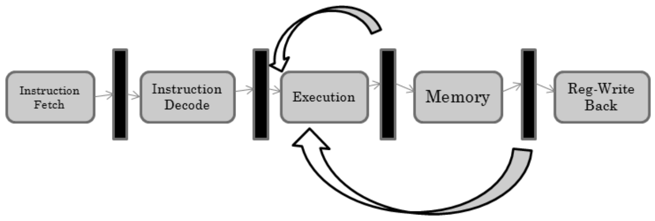

Fig. 1. 5-stage MIPS Processor

III. MIPS Pipeline architecture and Instruction Set Architecture

Pipelining is a method used to improve processor performance. Pipeline reduces the number of processor cycles needed to execute a set of instructions. Pipelining is incorporated with 5-stage MIPS processor architecture to improve its performance.

A. MIPS Pipelined Architecture

Instructions are first stored in the instruction memory. Based on the PC value, processor selects the instruction from the memory and passes it on to Decode Issue environment, which will decode the instruction into Operation code, Operand register and Destination register. Next, it will take the value from corresponding registers, and gives to the ALU for execution.

There are 2 ALUs in our proposed processor:

- First one is 'Dedicated ALU' for Multiplication Operation. When ALU is executing multiplication operation, it will hold all other instruction for 4 processor clock cycles. This multiplication operation is implemented using Booth Algorithm. This is for the CISC operation.
- Second ALU is for all other instructions (ADD, SUB, AND, OR , NOR, NAND, XOR, Shift Left and Shift Right).

After this it will *Write-Back* to the respective destination Registers, as given in the instruction.

For special Instructions, it does the following:

- For J-type instructions, this will take the 26-bit address and left shift it by 2, to make it 28 bit (this is done to make it into a word format).
- For *Load* and *Store* instructions, address is calculated by the ALU and is given to the PC.

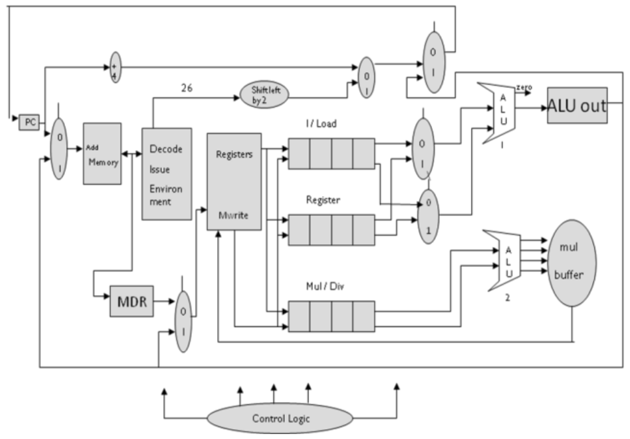

Fig. 2. Implementation Approach

B. Instruction Set Architecture(ISA)

Here we have implemented R-type, I-type and J-type instructions.

1) R-type Instruction: Register type instructions are one which takes the operands from registers and write the result back to a register. Format of the register instruction is as shown in Fig. 3.

In Fig. 3, *Opcode* stands for operation code of the instruction. *Rs* and *Rt* are source registers. *Shamt* is the shift amount and *Functval* stands for function value.

2) I-type and J-type Instruction: I-type is immediate instruction and J-type is Jump instruction. Format of the I-type instruction is shown in Fig. 4. and J-type is in Fig. 5. The instruction set is given in Table II and III for I-type and J-type respectively.

Opcode	Rs	Rt	Rd	Shamt	Functval
31 26	25 21	20 16	15 11	10 6	5 0

Fig. 3. R-type Instruction Format

Opcode	Rs	Rd	I-data
31 26	25 21	20 16	15 0

Fig. 4. I-type Instruction Format

Opcode	J-address of 26 bits
31 26	25 0

Fig. 5. J-type Instruction Format

TABLE I. R-TYPE INSTRUCTIONS

Sl.No.	Instruction	Action	Opcode	Functval
1	ADD $s1,$s2,$s3	$s3←$s1+$s2	000000	000001
2	SUB $s1,$s2,$s3	$s3←$s1-$s2	000000	000010
3	MUL $s1,$s2,$s3	$s3←$s1*$s2	000000	000011
4	AND $s1,$s2,$s3	$s3←$s1 (AND)$s2	000000	000100
5	OR $s1,$s2,$s3	$s3←$s1 (OR) $s2	000000	000101
6	NOR $s1,$s2,$s3	$s3←$s1 (NOR) $s2	000000	000110
7	NAND $s1,$s2,$s3	$s3←$s1 (NAND) $s2	000000	000111
8	XOR $s1,$s2,$s3	$s3←$s1 (XOR) $s2	000000	001000
9	DIV $s1,$s2,$s3	$s3←$s1 / $s2	000000	001001
10	SLT $s1,$s2,$s3	Set s3 if s1<s2	000000	001010

TABLE II. I-TYPE INSTRUCTIONS

Sl.No.	Instruction	Action	Opcode
1.	ADDI $s1,$s2,100	$s2←$s1+100	000001
2.	SUBI $s1,$s2,100	$s2←$s1-100	000010
3.	MULI $s1,$s2,100	$s2←$s1*100	0000 11
4.	ANDI $s1,$s2,100	$s2←$s1 (AND) 100	000100
5.	ORI $s1,$s2,100	$s2←$s1 (OR) 100	000101
6.	NORI $s1,$s2,100	$s2←$s1 (NOR) 100	000110
7.	NANDI $s1,$s2,100	$s2←$s1 (NAND) 100	000111
8.	XORI $s1,$s2,100	$s2←$s1 (XOR) 100	001000
9.	DIVI $s1,$s2,100	$s2←$s1 / 100	001001
10.	SLTI $s1,$s2,100	Set s2 if s1<100	001010
13.	BEQ $s1,$s2,25	Go to location $s2 if s1=25	001101
14.	BNE $s1,$s2,25	Go to location $s2 if s1!=25	001110
15.	BGT $s1,$s2,25	Go to location $s2 if s1>25	001111
16.	BLT $s1,$s2,25	Go to location $s2 if s1<25	010000
17.	SLL $s1,$s2,03	Shift Left Logical $s2←$s1<<3	010001
18.	SRL $s1,$s2,03	Shift Right Logical $s2←$s1>>3	010010

TABLE III. J-TYPE INSTRUCTIONS

Sl.No.	Instruction	Action	Opcode
1.	J 2500	Jump to PC <= 2500	010011
2.	JAL 2591	Jump to PC <= 2591, $ra = $ra +4 , SP [$ra] = Old PC value ; { $ra is address of stack pointer }	010100
3.	JR $ra	PC <= SP[$ra] ; $ra = $ra - 4	010101

IV. CISC OPERATION- MULTIPLICATION

Here the CISC operation- multiplication is implemented using Booth multiplication algorithm. Booth's multiplication algorithm implements 32-bit signed binary number multiplication using 2's compliment method. Booth multiplier architecture is shown in Fig. 6.

Booth's algorithm is implemented by repeatedly adding values A and S to a product P and then performing a shift right operation. Let m_1 and m_2 be the multiplicand and multiplier, respectively; and let n_1 and n_2 represent the number of bits in m_1 and m_2.

Step 1: First determine the value of A, S and P. Length of each of them is $n_1 + n_2 + 1$.

A: Fill the most significant bits with value of m1 and the remaining ($n_2 + 1$) bits with zeros.

S: Fill the most significant bits with 2's compliment of m_1 and remaining bits with zeros.

P: Fill the most significant n_1 bits with zeros, append the value of m_2 to the right and fill the last bit with zero.

Step 2: Following operations are performed based on the two least significant bits of P:

01: Find P + A. Ignore any overflow.

10: Find P + S. Ignore any overflow.

00: Use P directly in the next step.

11: Use P directly in the next step

Step 3: Right shift one place, the value obtained in the 2nd step and assign the value to P.

Step 4: Repeat steps 2 and 3 for n_2 times.

Step 5: Drop the least significant bit from P to obtain the final product of $m_1 * m_2$.

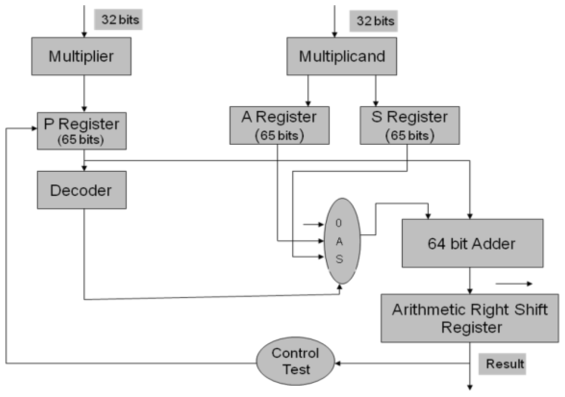

Fig. 6. Multiplier Architecture

V. FPGA IMPLEMENTATION

The above proposed 32-bit MIPS processor with one CISC operation, is implemented in Verilog language and finally emulated in Xilinx Nexsys Spartan 3E series FPGA. The different steps involved in mapping the Verilog source code are Synthesis, Translate, Map, Place & Route and finally generating the *Bit file*. These steps are carried out using Xilinx ISE Design Suite 12.4. Finally Functional verification is done and the waveforms are obtained in Xilinx ISIM.

The processor is tested at a maximum clock frequency of 177.336MHz (5.639ns time period) and verified the ISA. Fig. 7 shows how the proposed processor is connected to input-output pins of FPGA.

In FPGA implementation, the output of Processor is written into a Block RAM. Then from the Block RAM output is read using an External Clock, and given to an on-board 7-segment display.

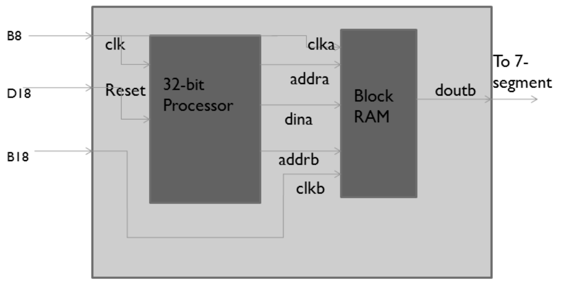

Fig. 7. Procssor connected with Spartan3 FPGA

VI. SIMULATION RESULTS

The proposed design is simulated Xilinx ISE and ISIM. Design emulation is done after obtaining the Bit-file from Xilinx ISE and dumping that to the target FPGA. The outputs are connected to the on-board 7-segment display of the target FPGA (Spartan3E).

Here we are demonstrating the simulation result for a Fibonacci series test. Fibonacci series is obtained by writing code using instructions in ISA. Following are the instructions that are placed in the Instruction memory:

```
0: data   = 32'd1;
4: data   = {6'd0, `Reg1, `Reg0, `Reg3, 5'd0, 6'd1};
8: data   = {6'd0, `Reg3, `Reg1, `Reg4, 5'd0, 6'd1};
12: data = {6'd0, `Reg3, `Reg4, `Reg5, 5'd0, 6'd1};
16: data = {6'd0, `Reg5, `Reg4, `Reg6, 5'd0, 6'd1};
20: data = {6'd0, `Reg5, `Reg6, `Reg7, 5'd0, 6'd1};
24: data = {6'd0, `Reg6, `Reg7, `Reg5, 5'd0, 6'd1};
28: data = {6'd0, `Reg5, `Reg7, `Reg1, 5'd0, 6'd1};
32: data = {6'd0, `Reg1, `Reg5, `Reg4, 5'd0, 6'd1};
36: data = {6'd0, `Reg1, `Reg4, `Reg5, 5'd0, 6'd1};
```

In the above code 0, 4, 8… 36 are PC values. The output obtained will be 1, 2, 3, 5, 8, 13, 21, 34, and 55. The initial values that are loaded into the Reg0 and Reg1 are 0 and 1 respectively. Fig. 8 shows the simulation waveform obtained for Fibonacci series test. Fig. 9 shows a multiplication instruction output which was implemented using a Booth multiplier.

The 32-bit MIPS processor with one CISC operation is verified using appropriate test cases. For this we did functional verification and then FPGA emulation, for each testcases.

The FPGA hardware utilized by the processor design are given in Table.IV.

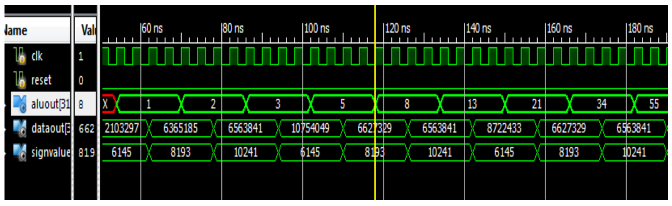

Fig. 8. Output waveform for Fibanocci Series test

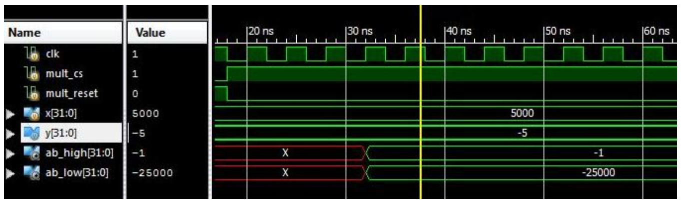

Fig. 9. Output waveform for a multiplication operation performed using a Booth multiplier

TABLE IV. FPGA HARDWARE UTILIZED BY THE DESIGN

Hardware units in FPGA	Number occupied	Total number of units	Percentage Utilized
Slices	403	4656	8%
Slice Flip Flops	494	9312	5%
4 input LUT	502	9312	5%
Bounded IOBs	98	232	42%

VII. CONCLUSION AND FUTURE SCOPE

In this paper we presented a 32-bit MIPS Processor with a CISC operation- multiplication. Typically a MIPS processor follows RISC architecture. Here we implemented the Multiplication operation using a 32-bit booth multiplier, which completes multiplication in 4 processor cycles. This greatly enhances the processor speed, whenever we need to execute a multiply instruction. But the disadvantage with this CISC operation is it increases the hardware complexity of the processor.

We implemented the processor design in Verilog Hardware Description Language (HDL) and verified the results in Xilinx Nexsys Spartan3E board. The design is verified at a maximum clock speed of 177MHz.

Usually in a processor design, the scope of parallelism is limited due to some data and control hazards. These data hazards are Read after Write (RAW), Write after Read (WAR) and Write after Write (WAW). These data hazards can be solved using Dynamic Scheduling algorithm. So it is always better to incorporate Dynamic Scheduling with a 32-bit processor, so that overall speed of the processor will be enhanced.

REFERENCES

[1] David A. Patterson John L. Hennessey ,Computer architecture: a quantitative approach, 3rd edition, Morgan Kaufmann Publishers.

[2] Balaji valli, A. Uday Kumar, B.Vijay Bhaskar, “FPGA implementation and functional verification of a pipelined MIPS processor” , International Journal Of Computational Engineering Research (ijceronline.com) Vol. 2 Issue. 5.

[3] Paresh Kumar Pasayat, Manoranjan Pradhan, Bhupesh Kumar Pasayat “ FPGA based implementation of 8-bit ALU of a RISC processor using Booth”, International Journal of Engineering Research & Technology (IJERT),Vol. 2 Issue 8, August – 2013.

[4] Marri Mounika,, Aleti Shankar **“**Design & Implementation Of 32-Bit Risc (MIPS) Processor**”** International Journal of Engineering Trends and Technology (IJETT) – Volume 4 Issue 10 - Oct 2013.

[5] MIPS® Architecture for Programmers Volume II-B: The microMIPS32™ Instruction Set, Revision 3.05 April 04, 2011.