SSRG International Journal of VLSI and Signal Processing
ISSN: 2394–2584 / https://doi.org/10.14445/23942584/IJVSP-V10I2P103

Volume 10 Issue 2, 17-21, May-Aug 2023

Original Article

Design and Implementation of RISC-V ISA (RV32IM) on FPGA

Anmol Singh[1], Arpit Kumar[2], Abhishek Singh[3], R. Anirudh Reddy[4], K. N. Pushpalatha[5]

[1,2,3,4,5]*Department of Electronics & Communication Engineering, Dayananda Sagar College of Engineering, Karnataka, India.*

Received: 29 April 2023 Revised: 05 June 2023 Accepted: 20 June 2023 Published: 06 July 2023

Abstract *- RISC-V, an open-source Instruction Set Architecture, originated from the collaborative efforts of researchers at the University of California, Berkeley, in 2010. It is a basic Load and Store type architecture based on traditional principles of RISC whilst providing flexibility in terms of extensions to the base Integer Set such as multiply, floating point and atomic instructions. This paper details the Design and Implementation of 5 stages pipelined RV32IM (base integer set with multiply extension). The design also incorporates a 2-bit branch predictor for increased throughput. Analysis and Verification have been performed for proper decoding, pipelined operation, branch prediction, stalling, memory access, and overall functionality. Verilog HDL on Intel QuestaSim has been used to design the core and simulation. DE 10 Lite board with Max 10 family of FPGA has been used for hardware synthesis and analysis of the design.*

Keywords *- RISC-V, Instruction Set Architecture, RV32IM, 5-stage pipeline, DE10 Lite FPGA.*

1. Introduction

A processor forms one of the major and important parts of any modern computer system. An Instruction Set Architecture (ISA) distinguishes one type of architecture from another. RISC V offers a viable alternative to proprietary ISAs such as ARM and x86. One of the key features of RISC-V is its modularity, which allows users to customize the ISA to fit their specific needs [1]. This flexibility is achieved using a small base ISA, which provides minimal instructions, and a series of standard extension modules that can be added to the base ISA as needed. Another advantage of RISC-V is its open-source nature, which allows for collaborative development and reduces the dependence on a single vendor or manufacturer. This has led to a growing ecosystem of RISC-V tools and platforms, including development boards, compilers, and operating systems.

A five-stage pipeline is a fundamental concept in processor design, which aims to improve the efficiency and performance of instruction execution. It involves breaking down the instruction execution process into five distinct stages, each handling a specific operation. The five stages typically include instruction fetch, instruction decode, execution, memory access, and writeback [2].

Branch prediction is a crucial technique employed in modern processors to mitigate the performance impact of conditional branch instructions. Branches occur when the processor encounters instructions like conditional branches, loops, or function calls that can alter the sequential flow of instructions. The purpose of branch prediction is to anticipate the outcome of a branch instruction before it is resolved and to speculatively fetch and execute the predicted instructions. By doing so, the processor can avoid pipeline stalls and maintain a high instruction throughput.

The DE10 Lite board with Max 10 FPGA combines the DE10 Lite hardware platform with the Intel Max 10 FPGA family. The Max 10 FPGA devices are low-power, non-volatile programmable logic devices that offer a range of resources and capabilities suitable for various applications.

The further sections detail the implementation process, results and review carried out.

2. Reference Study

A. Singh et al. [3] introduce a hardware design framework aimed at implementing the RV32I base integer instruction set in RISC-V for 32-bit address space. The implemented architecture discussed in this paper is a single-core, in-order, non-bus-based, single-cycle design that fully supports the RV32I base integer instruction set. This architecture finds application in various domains, including acoustic signal processing, real-time embedded systems, sensor technology and myriad other domains. A suite of tools and test frameworks around RISC-V was created targeted at 32-bit architectures. RV32I was specifically designed to serve as a comprehensive compiler target and provide support for modern operating systems. Additionally, its design aims to minimize the hardware resources needed for a basic implementation.

The fetch, decode and control logic block is responsible for fetching the instruction from the instruction memory, decoding the instruction, and generating the control signals. It is also responsible for resolving jump and branch target addresses. The RV32I architecture includes various components such as the program counter, target address selection logic, instruction memory controller, instruction decoder, and a control unit. Furthermore, it incorporates a dedicated adder responsible for incrementing the program counter in each cycle. The control unit is purely combinatorial, with all the control signals generated in the same cycle.

A comparison of synthesis and implementation has been conducted on two Virtex family boards, considering utilization reports and power reports as the evaluation criteria [3].

3. Materials and Methods

Once we have designed and optimized each sub-component, we must integrate them in a way that enables them to collaborate effectively and deliver the expected output for each instruction. Achieving this often requires making specific modifications to the design of each sub-component to ensure smooth integration with the others.
Pipeline stages are made by adding registers to store the result from each stage so that another instruction can be processed after the first instruction. SDC file is added to provide constraints for the design so that we can find the maximum operating frequency.

Before uploading the design to the FPGA, we convert the desired C code that we want to execute on the RISC-V core we covert into a hex file. This is because our digital design cannot understand high-level code. Therefore, we need to convert these instructions to hex values that can be read by our design. The path of this hex file is then provided to the design via Platform Designer (formerly known as Qsys). The design is then compiled and uploaded to the FPGA. While it's possible to use our own design for the register file during simulations, when implementing it on the FPGA, we'll need to rely on the internal memory blocks of the FPGA instead.

The different functions of the pipeline stages are mentioned below [11]:

3.1. Instruction Fetch

The instructions reside in memory that takes one cycle to read. This memory can be dedicated to SRAM or an Instruction Cache. During the Instruction Fetch stage, a 32-bit instruction is fetched from the instruction memory.

3.2. Instruction Decode

During the instruction decodes stage, the core receives the instruction from the previous fetch stage. The instruction is then processed and broken down into its constituent parts to determine the operation to be performed, the operands involved, and any additional information needed for execution.

3.3. Execute Stage

The Execute stage is responsible for carrying out the actual computations in a processor. Usually, this stage incorporates an Arithmetic Logic Unit (ALU). In addition to the ALU, it may also include a multiple-cycle multiplier and divider for more complex mathematical operations. The ALU is responsible for performing Boolean operations (and, or, not, nand, nor, xor, xnor) and also for performing integer addition and subtraction. The bit shifter is responsible for shifts and rotations.

3.4. Memory Stage

If data memory needs to be accessed, it is done in this stage. It is responsible for performing memory-related operations, such as loading data from memory or storing data in memory.

3.5. Writeback

During this stage, both single-cycle and two-cycle instructions store their results in the register file. It is important to note that two different stages access the register file simultaneously. The decode stage reads two source registers, while the writeback stage writes the destination register of a previous instruction. This situation can lead to a hazard on actual silicon (more on hazards below). A hazard arises when there is a conflict between the source registers being read in the decode stage and the destination register being written in the writeback stage. In such cases, the same memory cells in the register file are read and written simultaneously. The block diagram of the design is shown below.

3.6. Block Diagram

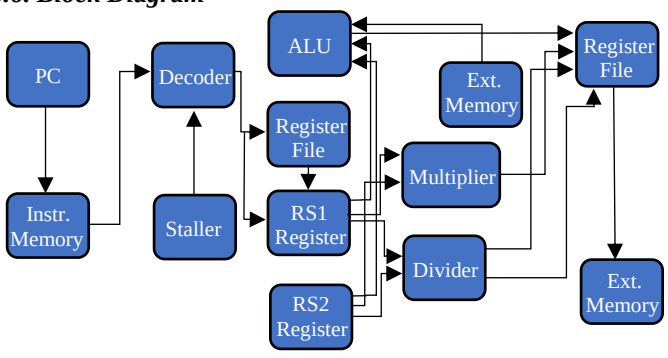

Fig. 1 Block Diagram

4. Results

The following results are from the simulation of the RV32IM core on Intel QuestaSim. The results shown are for basic operations taking place within the core.

4.1. Add Instruction

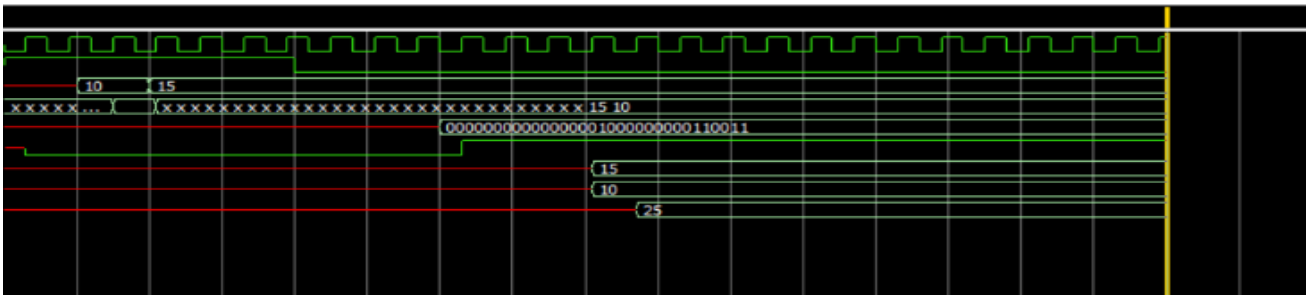

Fig. 2 Add instruction

4.2. Multiplication Instruction

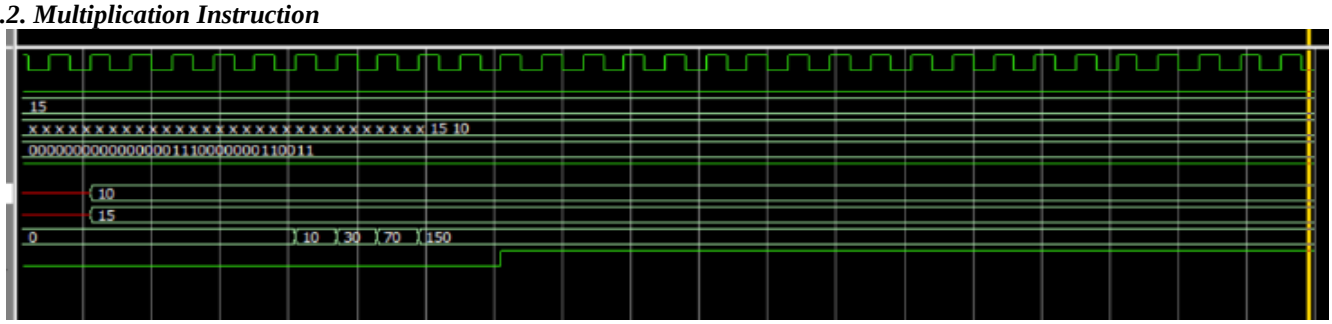

Fig. 3 Multiplication instruction

4.3. Subtraction Instruction

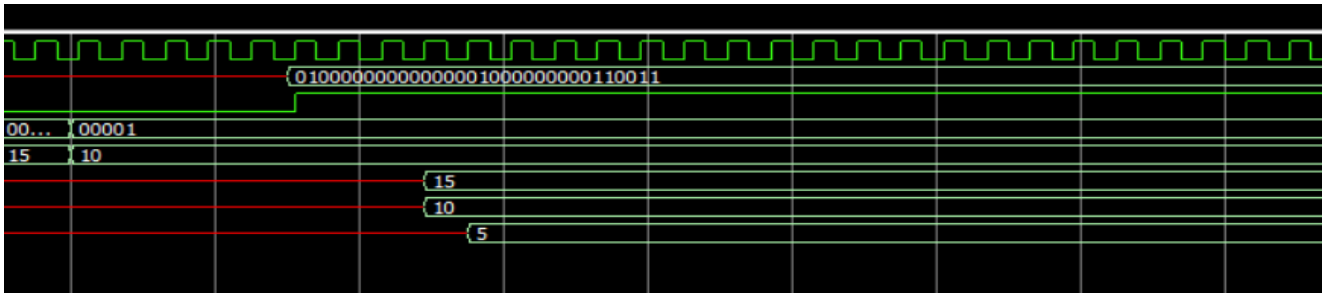

Fig. 4 Subtraction instruction

4.4. XOR Instruction

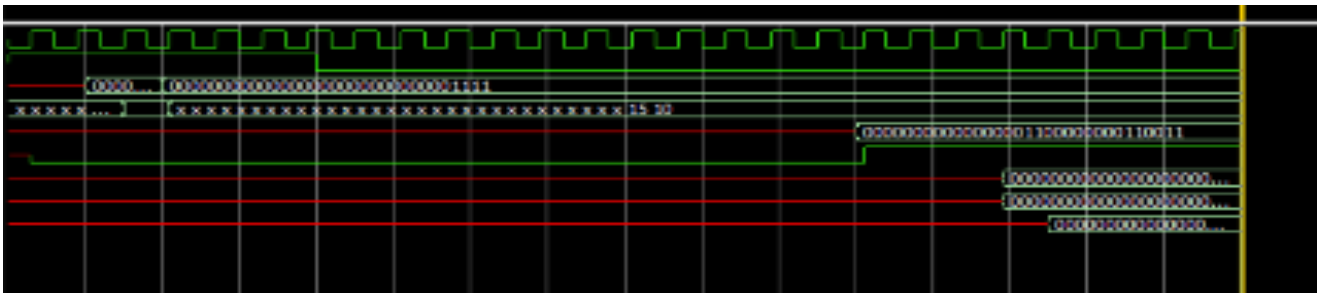

Fig. 5 XOR instruction

4.5. Quartus Synthesis Summary

Flow Summary

<<Filter>>

Flow Status	Successful - Wed May 31 13:18:42 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	Core
Top-level Entity Name	top_trial
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	558
Total registers	390
Total pins	135
Total virtual pins	0
Total memory bits	256
Embedded Multiplier 9-bit elements	0
Total PLLs	0
UFM blocks	0
ADC blocks	0

Fig. 6 Hardware utilization

4.6. Synthesized RTL

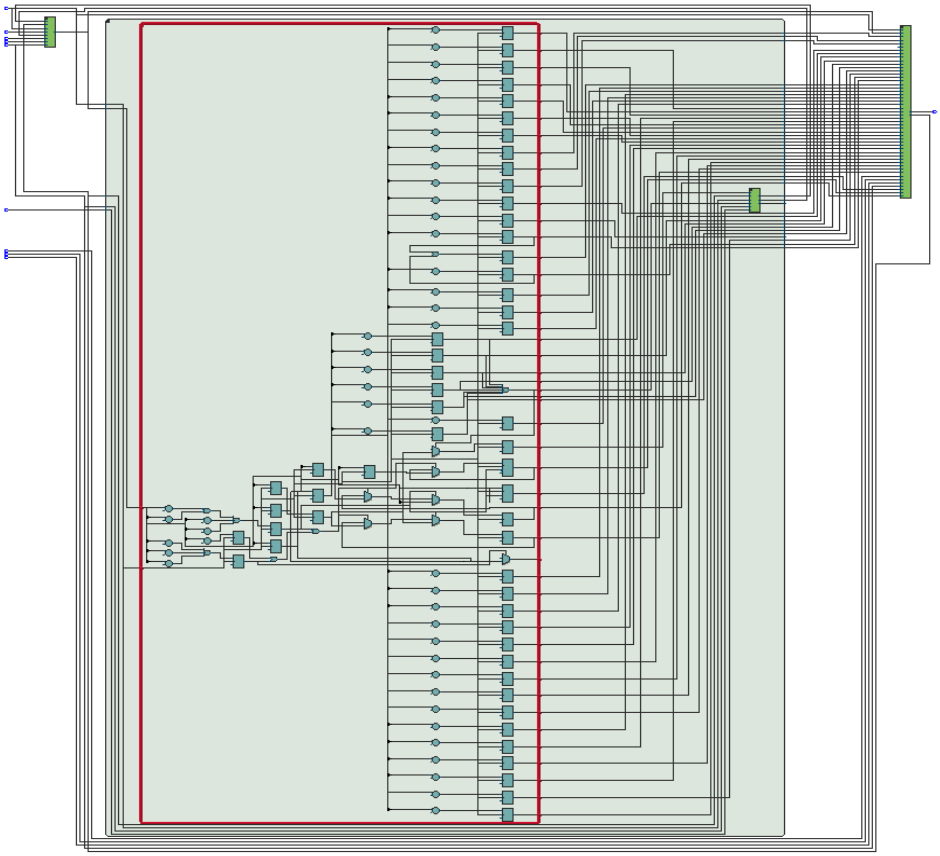

Fig. 7 Synthesized RTL

5. Conclusion

In this paper, we have presented our work, Design and Implementation of RISC V ISA on FPGA and have illustrated and experienced the findings and the specific features of our design.

The paper also demonstrates the design's 5-stage pipeline and branch prediction capabilities whilst illustrating its functionality through simulation and hardware results. Further, the paper also aims to showcase the development of the architecture core on the DE10 Lite board.

In conclusion, the objective of this paper is to provide hardware designers and developers with an overview of the RISC-V architecture, enabling them to gain a deeper understanding of its prominent implementations and versions.

Additionally, the paper aims to explore advanced features and their integration within the RISC-V framework, like the 5-stage pipeline and branch prediction, all combined, forming a large hardware and software ecosystem.

References

[1] Akshay Birari et al., "A RISC-V ISA Compatible Processor IP," *24th International Symposium on VLSI Design and Test*, pp. 1-6, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[2] Aneesh Raveendran et al., "A RISC-V Instruction Set Processor Microarchitecture Design and Analysis," *International Conference on VLSI Systems, Architectures, Technology and Applications*, pp. 1-7, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[3] Aslesa Singh et al., "Design and Implementation of a 32-bit ISA RISC-V Processor Core using Virtex-7 and Virtex-UltraScale," *IEEE, 5th International Conference on Computing Communication and Automation*, pp. 126-130, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[4] Sudhir Dagar, and Geeta Nijhawan, "Area Efficient Moving Object Detection using Spatial and Temporal Method in FPGA," *International Journal of Engineering Trends and Technology*, vol. 70, no. 9, pp. 138-147, 2022. [CrossRef] [Publisher Link]

[5] I. Kuroda et al., "A 16-bit Parallel MAC Architecture for a Multimedia RISC Processor," *IEEE, Workshop on Signal Processing Systems, SIPS 98, Design and Implementation*, pp. 103-112, 1998. [CrossRef] [Google Scholar] [Publisher Link]

[6] Shofiqul Islam et al., "Design of High-Speed-Pipelined Execution Unit of 32-bit RISC Processor," *Annual IEEE India Conference*, pp. 1-5, 2006. [CrossRef] [Google Scholar] [Publisher Link]

[7] Adrian Oleksiak et al., "Design and Verification Environment for RISC-V Processor Cores," *MIXDES - 26th International Conference 'Mixed Design of Integrated Circuits and Systems*', pp. 206-209, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[8] Don Kurian Dennis et al., "Single Cycle RISC-V Micro Architecture Processor and its FPGA Prototype," *7th International Symposium on Embedded Computing and System Design*, pp. 1-5, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[9] Yang Jing et al., "Electrical Diagnosis of Temperature-Dependent Global Clock Failures Using Probeless Isolation and Pattern Commonality Analysis," *19th IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits*, pp. 1-6, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[10] Ludovico Poli et al., "Design and Implementation of a RISC V Processor on FPGA," *17th International Conference on Mobility, Sensing and Networking*, pp. 161-166, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[11] David A. Patterson, and John L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, Morgan Kaufmann Publisher, Elsevier, 2017. [Google Scholar] [Publisher Link]

[12] Andrew Waterman et al., *The RISC-V Instruction Set Manual*, EECS Department, University of California, 2016. [Google Scholar] [Publisher Link]

[13] Reinhold P. Weicker, "Dhrystone: a Synthetic Systems Programming Benchmark," *Communications of the ACM*, vol. 27, no. 10, pp. 1013-1030, 1984. [CrossRef] [Google Scholar] [Publisher Link]

[14] Jan Andersson, "Development of a NOEL-V RISC-V SoC Targeting Space Applications," *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, pp. 66-67, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[15] Jaewon Lee et al., "RISC-V FPGA Platform toward ROS-Based Robotics Application," *30th International Conference on Field-Programmable Logic and Applications*, pp. 370-370, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[16] RISC-V Specifications, 2021. [Online]. Available: https://riscv.org/technical/specifications

[17] Pierre Maillard et al., "Test Methodology & Neutron Characterization of Xilinx 16nm Zynq® UltraScale+™ Multi-Processor System-on-Chip (MPSoC)," *IEEE Radiation Effects Data Workshop (REDW)*, pp. 1-4, 2018. [CrossRef] [Google Scholar] [Publisher Link]