Grenze International Journal of Engineering and Technology, June Issue

Design and Physical Implementation of Custom RISC-V Processor from RTL to GDSII

Prajwal N[1] and Nakul Shetty K[2]
[1-2]Department of ECE, Manipal Institute of Technology, Manipal, India
Emai: prajwalnmysore2002@gmail.com, nakul.shetty@manipal.edu

***Abstract*— This paper describes the details of a custom RISC-V processor design from the Register Transfer Level (RTL) stage to the final GDSII layout. It is capable of supporting single-precision R-type (Register) and F-type (floating-point) instructions. On the R-type side, the processor can perform the following operations: ADD (addition), SUB (subtraction), AND (bitwise AND), OR (bitwise OR), XOR (bitwise XOR), SLL (shift left logical), SLT (set less than), SLTU (set less than unsigned), SRL (shift right logical), and SRA (shift right arithmetic). In addition to these F-type operations: ADD, SUB, MUL (multiplication), DIV (division), MAX (maximum), MIN (minimum), LEQ (less than or equal), EQ (equal), and LT (less than). Major blocks such as an Arithmetic Logic Unit (ALU), decoder, and memory are wired together, with a data selector multiplexer regulating ALU operations and memory bank. The floating-point instructions extend the range of numerical calculations, thus providing a faster solution for the scientific simulations and graphic processing applications. The implementation is a full custom design and the stages of the development have been documented as follows: The design has been verified with the help of Cadence SimVision, synthesized using Cadence Genus, Physical design accomplished by using Cadence Innovus.**

***Index Terms*— coprocessor, floating point arithmetic, RISC-V instructions, single precision.**

I. INTRODUCTION

RISC-V is an open-source instruction set architecture (ISA) that offers great flexibility and scalability for the latest computing systems. In this paper, we describe our journey from the Register Transfer Level (RTL) to the final GDSII layout of a custom RISC-V processor. Our processor design includes support for single- precision R-type and F-type (floating-point) instructions, covering basic arithmetic, logical, and comparison operations. We have incorporated the main elements, for example, the Arithmetic Logic Unit (ALU), decoder, and memory; a data selector multiplexer is employed to enable smooth operation selection and data storage operations. This design aims to produce a processor that optimally balances performance, area, and power, therefore, making it ideal for applications such as scientific computing, graphics processing, and embedded systems. The processor's functionality was verified using Cadence SimVision, synthesized with Cadence Genus, and realized as a full custom design. Here we also describe the entire ASIC design flow, including verification, synthesis results, power and timing analysis, the physical layout generation, and more.
This work is carried out and confirmed through industry-standard EDA tools from Cadence specifically Genus for logic synthesis, Innovus for physical design, and Tempus for timing analysis right from RTL to GDSII stage.

Grenze ID: 01.GIJET.12.2.123_23

Along with having both Register and floating-point computing features, this single-chip processor can act as a core unit for efficient embedded systems and hardware accelerators. The paper outlines the architectural strategy, design methodology, and functional analysis of the processor which, in turn, supports the research in the domain of open-source hardware and ASIC-based RISC-V implementations.

II. METHODOLOGY

A. Basic System Architecture

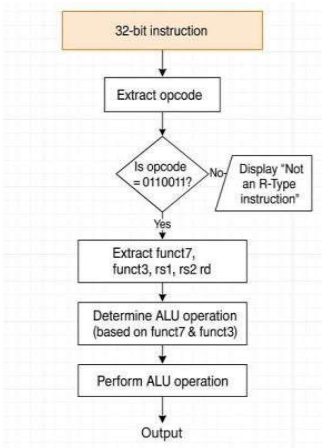

Figure 1. Basic system Architecture

Figure 1 shows the decoding and execution of an R-type instruction in a RISCV processor. The process starts with fetching a 32-bit instruction, which is then checked for the opcode of 0110011. If the opcode is a match, the instruction is recognized as an R-type one, and the Function7 field is retrieved. The RS1 and RS2 fields, which contain the source register numbers, are also extracted. The ALU performs the specified operation in the Function7 field on the values fetched from the registers indicated by RS1 and RS2. The ALU operation outcome is written back to the destination register given by the RD field, thus the execution of the R-type instruction is finished.

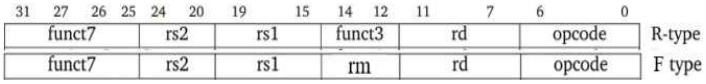

Figure 2. Instruction format of Register and Floating point [2]

Figure 2 shows the 32-bit instruction word is divided into fixed fields to represent different parts of an instruction. The opcode occupies the least significant 7 bits and identifies the instruction type. The destination register (rd) uses 5 bits. The source registers (rs1 and rs2) also use 5 bits each. The remaining higher-order bits are divided into funct3 or rm (3 bits) and funct7 (7 bits), which further define the operation or rounding mode. The rounding mode controls how fractional results are handled, for example in an FADD instruction, a result like 3.748 can be rounded to 3.75 using round-to-nearest.

B. Data Path

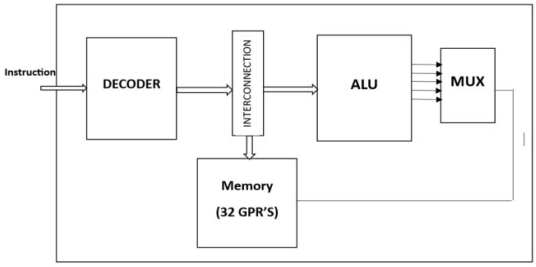

Figure 3. Top module [1]

Figure 3 shows the data path of the processor that has been laid out with the decoder, memory, and ALU blocks as the main components. The decoder block generates the control signals for carrying out the operations and also the memory unit is given the address of the data that is to be fetched. Aside from this, the decoder block generates control signals for the ALU block to perform the operations. The memory unit provides the data to the ALU block where the ALU takes the instructions to operate and the final result will get back to the Memory unit, to store the result. Thus, the RISC-V R-Type data path has been completed.
A register-type unit has been combined with the RISC-V ISA and is presented in the table below for ADD, SUB, OR, XOR, AND, SLL, SLT, SLTU, SRL, SRA. The floating-point unit is equipped with the following operations: ADD, SUB, MUL, DIV, MAX, MIN, EQ, LT, LEQ. Hence, designed a data path with the following components a Decoder, 1KB Memory, 32-bit ALU, and 19:1 Mux for data selection output which is written back to memory.

TABLE I: RISC-V REGISTER-TYPE FORMAT [2]

FUNC7	RS2	RS1	FUNC3	RD	OPCODE	INSTRUCTION
0000000	rs2	rs1	000	rd	0110011	RADD.S
0100000	rs2	rs1	000	rd	0110011	RSUB.S
0000000	rs2	rs1	111	rd	0110011	RAND.S
0000000	rs2	rs1	110	rd	0110011	ROR.S
0000000	rs2	rs1	100	rd	0110011	RXOR.S
0010100	rs2	rs1	001	rd	0110011	RSLT.S
0010100	rs2	rs1	000	rd	0110011	RSLTU.S
1010000	rs2	rs1	010	rd	0110011	RSLL.S
1010000	rs2	rs1	001	rd	0110011	RSRL.S
1010000	rs2	rs1	000	rd	0110011	RSRA.S

TABLE II: RISC-V FLOATING-TYPE FORMAT [2]

FUNC7	RS2	RS1	RM3	RD	OPCODE	INSTRUCTION
0000000	rs2	rs1	rm	rd	1010011	FADD.S
0000100	rs2	rs1	rm	rd	1010011	FSUB.S
0001000	rs2	rs1	rm	rd	1010011	FMUL.S
0001100	rs2	rs1	rm	rd	1010011	FDIV.S
0010100	rs2	rs1	000	rd	1010011	FMAX.S
0010100	rs2	rs1	001	rd	1010011	FMIN.S
1010000	rs2	rs1	000	rd	1010011	FLE.S
1010000	rs2	rs1	001	rd	1010011	FLT.S
1010000	rs2	rs1	010	rd	1010011	FEQ.S

The opcode in the above tables is essentially a code that helps the processor identify the type of instruction (R-type, F-type, etc.) while FUNC3 and FUNC7 tell the processor the exact operation to be executed, i.e. addition, subtraction, multiplication, or logical operations. In these instruction formats, RS1 and RS2 are the source registers that contain the operands of the operation, while RD is the destination register where the result of the operation is saved. These fields allow data transfer from one register to another within the processor, which is the core of computation in the RISC-V architecture.

III. RESULTS AND DISCUSSION

A. Decoder

The main function of a decoder is to separate the instruction word for major elements such as opcode, registers, and immediate values and going to respective execution units. In R-type & F-type instructions decoder plays main role extracting the opcode from bits [6:0] which is used for identifying type of instruction and its operation. The decoder next isolates the register rs1, rs2 and destination register rd, in addition to the function code (func3, func7) which also changes the operation.

Figure 4. Decoder output

The figure 4 explains how an instruction is decoded using opcode, func3 or rm3, and func7 fields. The opcode first decides the type of instruction. Then func3 or rm3 and func7 further refine the operation. Based on their combination, the corresponding instruction enable signal is activated, ensuring correct and controlled execution.

B. Memory

In the RISC-V architecture, memory is an essential component that helps in the efficient execution of instructions through the use of General-Purpose Registers (GPRs) and, possibly, other memory components. For the R-type & F-type instructions, memory access is mainly concerned with getting the data from the source registers rs1 and rs2 and then storing the result in the register rd.

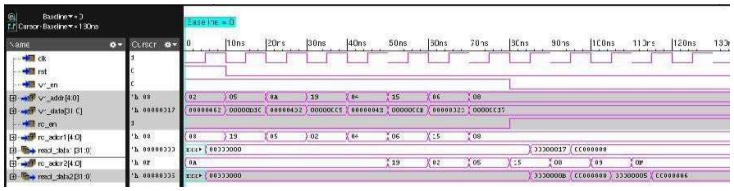

Figure 5. Memory output

Figure 5 explains when the write enable signal goes high, the value present on the write data bus is written into the memory location pointed to by the address. Later, during a read operation, the memory outputs the stored value on the read data lines for the selected address. This behavior confirms that the memory is correctly performing both write and read operations

C. ALU

For R-type the available set of ALU operations includes arithmetic operations like ADD (RADD.S) which add the values in RS1 and RS2, and SUB (RSUB.S) which takes the value in RS2 away from RS1, etc. Logical operations comprise AND (RAND.S) for bitwise AND, OR (ROR.S) for bitwise OR, and XOR (RXOR.S) for bitwise XOR. The processor also supports comparison instructions SLT (RSLT.S) for signed less-than and SLTU (RSLTU.S) for unsigned less-than. The shift instructions are SLL (RSLL.S) for logical left shift, SRL (RSRL.S) for logical right shift, and SRA (RSRA.S) for arithmetic right shift.

The F-type arithmetic instructions [6] that are available include FADD.S for floating-point addition, FSUB.S for subtraction, FMUL.S for multiplication, and FDIV.S for division. Besides, there are the comparison operations like FMAX.S and FMIN.S for choosing the maximum or minimum of two numbers and relational comparison operations [5] like FEQ.S (equal), FLT.S (less than), and FLE.S (less than or equal).

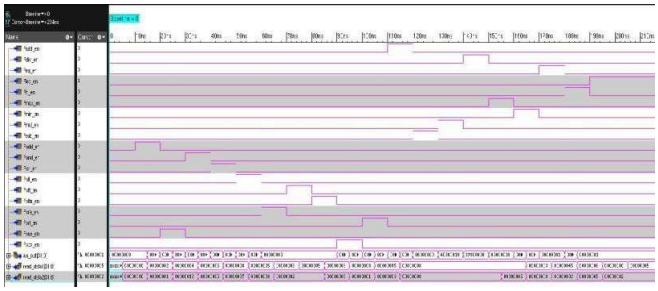

Figure 6. ALU output

Figure 6 explains that the operation to be performed is selected using control signals and opcode values. As the input operands change, the ALU performs the selected arithmetic or logical operation and produces the corresponding result. The output updates correctly with each operation, showing that the ALU is functioning as expected.

D. Datapath

The data path of the custom RISC-V processor is a key component of instruction execution, which allows the flow of data from one functional unit to another. It is made up of various blocks that are interconnected, such as the decoder, memory, Arithmetic Logic Unit (ALU), and multiplexers. The instruction decoder is the first to understand the execution of a 32-bit instruction and generates control signals that guide the flow of data and operation selection. The register file is accessed to retrieve the source operands, which are then sent to the ALU or FPU, the choice depending on the type of the operation, i.e., whether it is an R-type or F-type operation. The ALU executes the integer arithmetic, logical, comparison, and shift operations, the FPU executes the single-precision floating-point operations conforming to the IEEE-754 standard. A 19:1 multiplexer is needed to choose the right output from the different functional units. The result of the computation in the register is updated through the memory block. This well-organized data path allows for quick execution of both Register and floating-point instructions and at the same time, it keeps synchronization through the control logic.

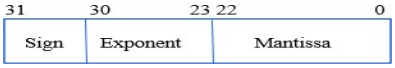

Figure 7. IEEE-754 format [2] [11]

As per IEEE-754 Floating Point Standard, which is specified for a single precision of 32 bits. Figure 7 illustrates how the 32 bits are divided into three sections: Mantissa (0–22) has 23 bits, Exponent (23–30) has 8 bits, and the sign bit will be the 31st bit. In this design, floating-point exception handling is simplified by representing NaN as 32'bx. When an invalid or undefined operation occurs, such as division by zero or an unsupported floating-point operation, the output is driven to 32'bx to indicate an indeterminate result. This method avoids complex exception control logic while clearly signaling an invalid computation, with minimal impact on overall latency

Figure 8. Datapath output

Figure 8 illustrates the simulation of the top module, integrating all major blocks of the processor. It shows the interaction between the decoder, memory & ALU under clock control. Changes in input signals propagate through the datapath, and the corresponding outputs update correctly, indicating proper coordination and functional correctness of the complete processor design.

E. Synthesis

Synthesis is essentially turning a detailed specification of a digital system at a high-level hardware description language such as Verilog, VHDL into a gate level netlist. This netlist essentially contains the basic logic gates and the interconnections among them that implement the digital circuit. The ultimate goal of synthesis is to provide a hardware design that can be implemented with the given technology, while also satisfying certain performance parameters such as timing, area, and power consumption.

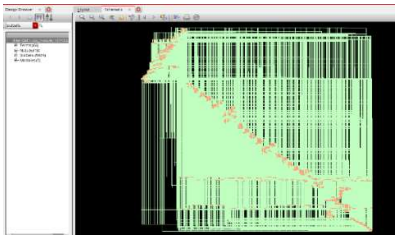

Figure 9. Gate Level Netlist

The synthesis output is checked with the lec auto tool from Cadence, using the slow library of the 90nm technology node, in order to verify that the RTL design and the synthesized netlist are functionally equivalent which is shown in figure 9.

F. Physical Design

Physical Design (PD) means taking a synthesized RTL netlist and changing it into a layout that can be put on a silicon chip. So, it is about doing work on the floorplan, placing, routing, etc., and then it is checked if the chip needed the specifications in terms of performance, power, and area (PPA) requirements and also if the design rules of the foundry have been respected.

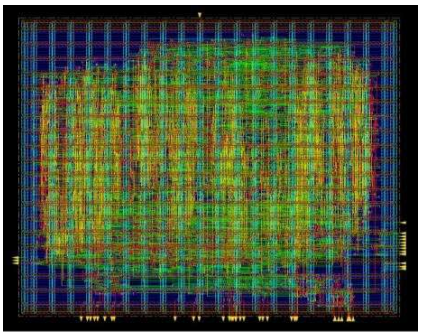

Figure 10. Final layout

Figure 10 shows the final physical layout of the processor after place and route, where standard cells, routing layers, and power grids are arranged to meet area, timing, and connectivity requirements before GDSII generation.

TABLE III: COMPARISON WITH PREVIOUS WORK

	This work	[4]	[7]
Frequency	1GHz is maximum operating frequency	100MHz	240 Mhz
Power	32.74 mW (All operations)	15.53 mW (only Divider operation)	-
Area	77192.446 µm²	469561.50 µm²	-

G. GDSII

The GDSII (Graphic Design System II) layout is essentially the final physical representation of a circuit design that is ready to be sent for manufacturing. It contains the extremely detailed geometric information of the chip including shapes, paths and layer data that specify different areas and materials on the chip. The GDSII file comes out after placement, routing, and physical verification stages and it is the standard format that is passed to the semiconductor foundry for mask making and production. Furthermore, this layout preserves the entire hierarchical structure of the design and thus is able to guarantee that all design rules and connectivity are followed, which makes it a very crucial output of the physical design flow

Figure 11. GDSII Format

IV. CONCLUSION

The main goal of the project was to design basic R-type and F-type (floating point) instructions of the RISC V architecture using Verilog and to complete the physical design (PD) flow up to GDSII. Addition, subtraction, AND, OR, exclusive OR, arithmetic shift left, logical shift right, and arithmetic shift right instructions were

implemented using logic simulation to verify their correctness. Floating-point arithmetic was implemented using ADD, SUB, MUL, DIV, MAX, MIN, EQ, LT, and EQ instructions, following the Verilog coding and simulation steps. The design was synthesized successfully and subsequently went through the backend flow consisting of placement, CTS (clock tree synthesis), routing, and signoff stages, resulting in a GDSII layout.

REFERENCES

[1] Harshitha, K., Hithen, A. S., Madappa, T. J., Karthik, B. G., and Bhat, N. S., "Microarchitecture design and verification of co-processor for floating point operation," In Recent Trends in Computational Sciences pp. 84-89, CRC Press, 2023.

[2] A. Waterman, K. Asanovi´c, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA," RISC V Foundation, 2014.

[3] Cherian, R., Thomas, N., & Shyju, Y. (2013, March). Implementation of binary to floating point converter using HDL. In 2013 International Mutli-Conference on Automation, Computing, Communication, Control and Compressed Sensing (iMac4s) (pp. 461-464). IEEE.

[4] Oh, Hyun Woo, "RF2P: A Lightweight RISC Processor Optimized for IEEE from Migration Rapid- Posit." to 754 2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). IEEE, 2023.

[5] Samuel, A. T., & Senthilkumar, J. (2015, March). A novel floating point comparator using parallel tree structure. In 2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015] (pp. 1-6). IEEE.

[6] Haijun, S., Zhibiao, S., Gang, Z., & Ning, Z. (2005, May). The research on optimization techniques of 32-bit floating-point RISC microprocessor. In Proceedings of 2005 IEEE International Workshop on VLSI Design and Video Technology, 2005. (pp. 63-66). IEEE..

[7] V. Patil, A. Raveendran, P. M. Sobha, A. David Selvakumar and D. Vivian, "Out of order floating point coprocessor for RISC V ISA," 2015.19th International Symposium on VLSI Design and Test, Ahmedabad, India, 2015, pp. 1-7, doi: 10.1109/ISVDAT.2015.7208116.

[8] Sandoval-Hernández, M. A., Velez-López, G. C., Vázquez-Leal, H., Filobello-Nino, U. A., Morales-Alarcón, G. J., De-Leo-Baquero, E., ... & Cuellar-Hernández, L. (2023). Basic Implementation of Fixed-Point Arithmetic in Numerical Analysis. International Journal of Engineering Research y Technology, 12(01), 313-318.

[9] Ozbilen, Metin Mete, and Mustafa Gok. "A multi-precision floating-point adder." 2008 Ph. D. Research in Microelectronics and Electronics. IEEE, 2008.

[10] Thakkar, Anuja J., and Abdel Ejnioui. "Design and implementation of double precision floating point division and square root on FPGAs." 2006 IEEE Aerospace Conference. IEEE, 2006.

[11] Rao, R. P., Rao, N. D., Naveen, K., & Ramya, P. (2018, February). Implementation of standard floating point MAC using IEEE 754 floating point adder. In 2018 Second International Conference on Computing Methodologies and Communication (ICCMC) (pp. 717-722). IEEE.