International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 12 Issue: 05 | May 2025 www.irjet.net p-ISSN: 2395-0072

PipelineAR: Bridging RTL and Layout in a 5-Stage 32-bit MIPS Processor Design: Verilog RTL, GTKWave Simulation, and Tanner EDA Implementation.

Tummala Deepak[1], D Veeraswamy [2], B Ravi Kumar[3], Basavaraj Aland[4]

[1]BTECH 3rd year, Dept. of ECE, Institute of Aeronautical Engineering, Telangana, India
[2] Professor, Dept. of ECE, Institute of Aeronautical Engineering, Telangana, India
[3] Professor, Dept. of ECE, Institute of Aeronautical Engineering, Telangana, India
[4]BTECH 3rd year, Dept. of ECE, Institute of Aeronautical Engineering, Telangana, India

---

Abstract - *In today's digital era, processor design remains a core focus within computer architecture, with RISC (Reduced Instruction Set Computer) architectures gaining prominence for their simplicity, speed, and energy efficiency. A major factor behind modern processor performance is pipelining, which divides instruction execution into stages to allow overlapping operations. The commonly adopted 5-stage pipeline—comprising Instruction Fetch, Decode, Execute, Memory Access, and Write Back—significantly enhances throughput. Additionally, the Harvard architecture, which uses separate instruction and data memory units, further improves performance by enabling parallel access.*

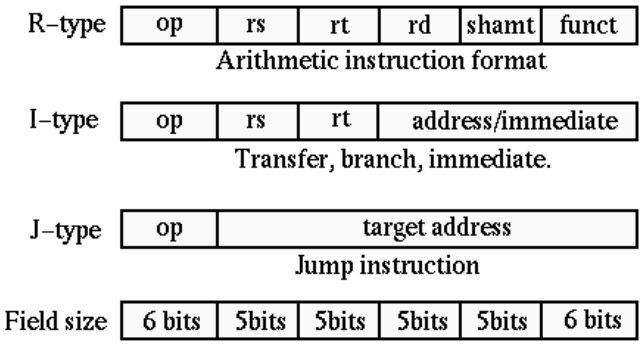
This paper details the design of a 32-bit pipelined MIPS processor using Harvard architecture, modeled in Verilog HDL with each stage implemented as an individual module. Simulation and functional validation were conducted using GTKWave, and schematic designs were developed in Tanner EDA for all pipeline stages. GDSII layout files were generated to analyze placement and routing in 250nm technology. While full physical metrics like power and area were not measured, approximate layout estimations were made from schematic observations. The project highlights the complete RTL-to layout design flow, offering both educational and practical insights into pipelined RISC processor development.

Key Words: **RISC (Reduced Instruction Set Computer), MIPS (Microprocessor without Interlocked Pipeline Stages), pipeline hazards, GDSII (Graphical Data Stream Information Interchange), Verilog.**

1.INTRODUCTION

There are numerous types of processors that exist now. All of them are built using hardware descriptive language, i.e., VHDL or Verilog. There are two processor types: RISC and CISC-based. RISC processors are said to be the most effective CPU design, a substitute for CISC. It is composed of optimized instructions doing fewer operations but at a faster rate. The use of thread-level parallelism and instruction-level parallelism by RISC processors to enhance register set and internal parallelism is its key feature. It also causes the CPU to execute instructions at a faster rate. Smartphones, tablets, computers, and numerous other smart devices employ RISC architecture. RISC processor units not only function more efficiently but are also more cost-effective to design and manufacture than CISC, using fewer transistors. Because of the aforementioned reasons, system-on-chip (SoC) is the optimal use of the RISC architecture. MIPS architecture design is always consistent but may vary depending on whether the pipeline is single-cycle or multi-cycle implemented [2]. Routers and other such small computing devices utilize MIPS.

The MIPS design has three kinds of instruction sets: Register type , Immediate type and jump type



Where,

op = opcode
rs = identifier of first source register
rt = identifier of second source register
rd = identifier of destination register
shamt = shift amount indicating how many bits the contents of a register must be shifted left or right (only used in shift instructions – NOT used here)
funct = distinguishes among R-type instructions as all R-type instructions have op = 0.

1.1 Register type (R type) :

Here the opcode is indicated by the last 6 bits. The other next 15 bits stand for the three registers Rs, Rt, and Rd, which are used for operations. The source registers are Rs, Rt, and the destination registers are

Rd, respectively. The other five bits indicate the shift amount, which indicates the number of bits to be shifted. The function field in the last six bits indicates the operation to be performed on the registers.

1.2 Type Immediate (I type):

If an instruction has to operate on a register value and an immediate value, an I instruction is employed. The source and destination register operands, rs and rt, are 5 bits each. The immediate value is 16 bits (0 to 15). This value is represented in two's complement depending on the instruction and is typically employed as the offset value in most instructions.

1.3 Jump Type (J type):

The first 5 bits of the J-type Instruction format determine the kind of jump operation to be executed. The branch offset is kept in 2's complement in the next 26 bits.

2. Tools Used:

a) Iverilog

b) GTKWave

c) Qflow, Graywolf, Magic, and other workflow software.

d) Tanner EDA

e) Visual Studio Code.

3. Methodology

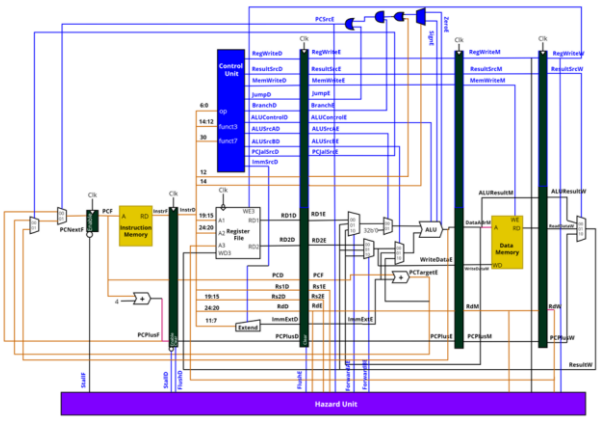


Fig: Proposed 5-stage Pipelined MIPS Processor

The 5-stage pipelined 32-bit MIPS processor implementation based on Harvard architecture is attained based on a properly structured RTL-to-layout flow. The primary goal is to implement a simplified MIPS processor that demonstrates the pipelining concept using Verilog HDL, functionally simulate it using GTKWave, and further develop its schematic and layout in Tanner EDA for a 250nm technology node.

A minimal subset of the MIPS32 instruction set was selected to implement the basic functionality required for testing pipeline behavior. The standard 5 pipeline stages — Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB) — were used to model instruction flow. Each stage was carefully designed to operate concurrently while handling pipeline hazards using data forwarding and stalling mechanisms (nops).

The various pipelining phases are given below:

3.1 Instruction Fetch Stage:

Here, the CPU loads instructions from memory at the address contained in the value of the Instruction Pointer (IP) or Program Counter (PC). The Opcode of the instruction is loaded from the Instruction Memory with all of the instructions required for the process.

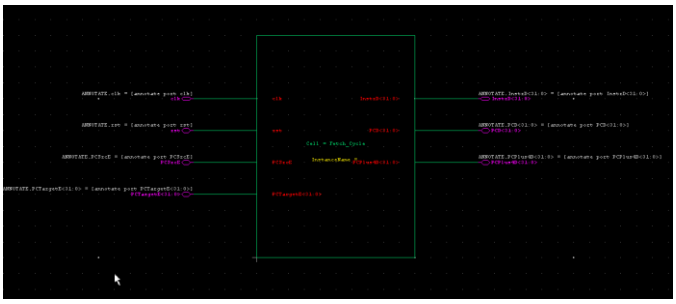


Fig: Symbol generation using the Fetch Cycle Verilog HDL.

3.2 Instruction Decode Stage:

The role of the decoder stage is to decode and receive the loaded instructions and send them to the control unit. The ID stage processes the data received from the IF stage. The Opcode is sent to the decoder unit via the fetch stage. ID stage controls different components and modules of the design as a function of the Opcode and its allied functions. Four instruction types are: Register type, Immediate-type, Jump-type, and I/O type instructions. Operations of the control unit are performed based on the Opcodes of these instructions. One of the mandatory components as well as the brain of the processor and ID stage, is a controller that takes as input instruction's Opcode of 6-bits length and produces the Execute instruction as output and which is forwarded to the Arithmetic and Logical Unit. The Execute Command defines a specific operation that ALU has to execute. Once a threat is detected, the controller halts all write instructions to the register file / memory. Instruction decode stage contains the Sign Extend unit. This unit translates the last 16 bits of the immediate instruction—the immediate data—into 32-bits. The MSB bit value of Immediate Instruction is added to the rest of the bits to execute sign extension.

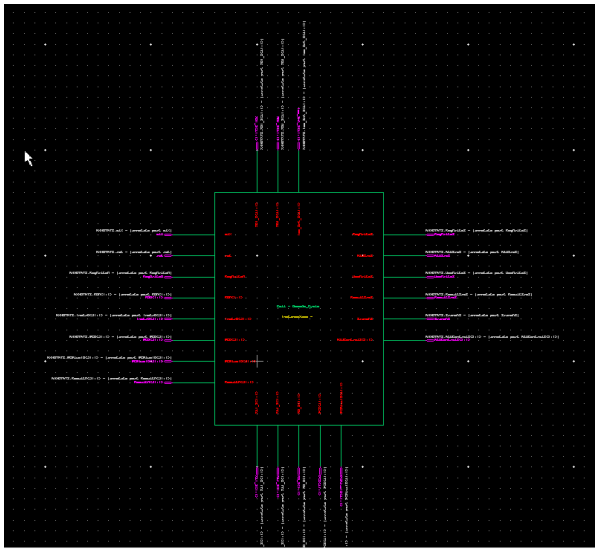


Fig: Symbol generation using the Decode Cycle Verilog HDL

3.3 Instruction Execute Stage:

Computation is performed in the EX-stage. The arithmetic logic unit is the central part of a processor. It will execute all operations to be performed. ALU processes the operations based on the Execute Command input. The ALU performs shift operations, logical and arithmetic, and logical operations like ADD, SUB, AND, OR, NOR, and XOR.

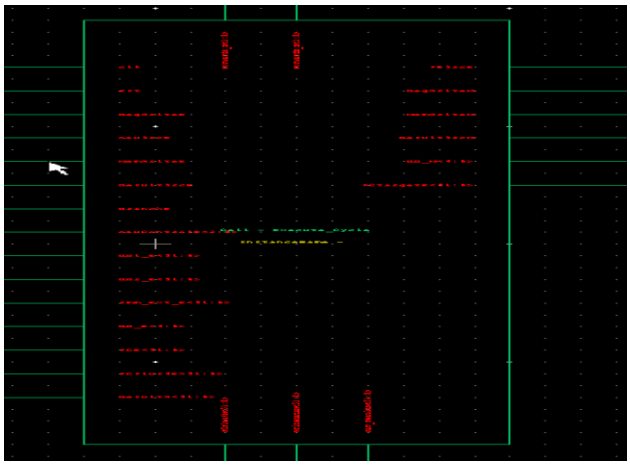


Fig: Symbol generation using the Execute Cycle Verilog HDL.

3.4 Memory Access Stage:

The loading and unloading of values into and from the registers is the main work and activity at the memory stage. They also fetch and forward the data from the memory module. The main work of the data value is to get stored in the appropriate destination registers according to the instruction.

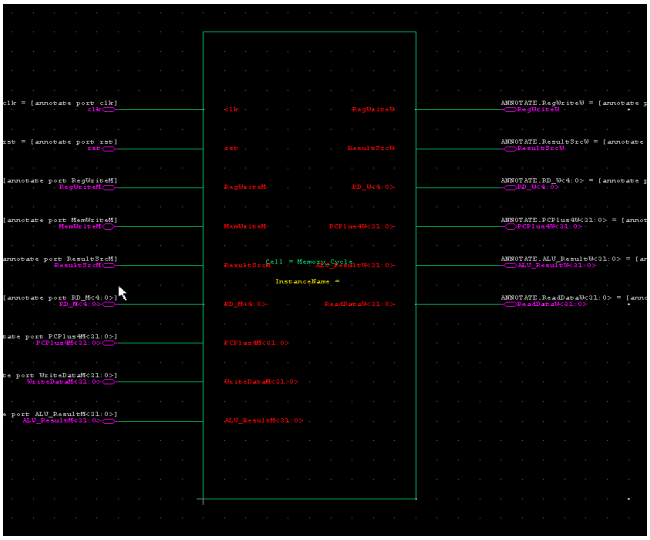


Fig: Symbol generation using the Memory Cycle Verilog HDL.

3.5 Write Back Stage:

During this stage computed or loaded value will be written in the specified register as per instructions. Observe that two distinct stages are accessing the register file at the same time: the decode stage is reading the two source registers, and the WB stage is writing the destination register of a previous instruction.

This stage is accountable for the result, information storage, and input and output of data to and from the register. Pipeline Registers or Buffers—different types of memory—separate these pipelined stages. These kinds of buffers are utilized to split the stages so that issues do not emerge when different instructions are processed at the same time without any data conflict. Since the function of the WB stage is to writing data to the destination register. For instance, the R1 register is produced by the memory of the ADD R1, R2, and R3 instruction to speed up programs.

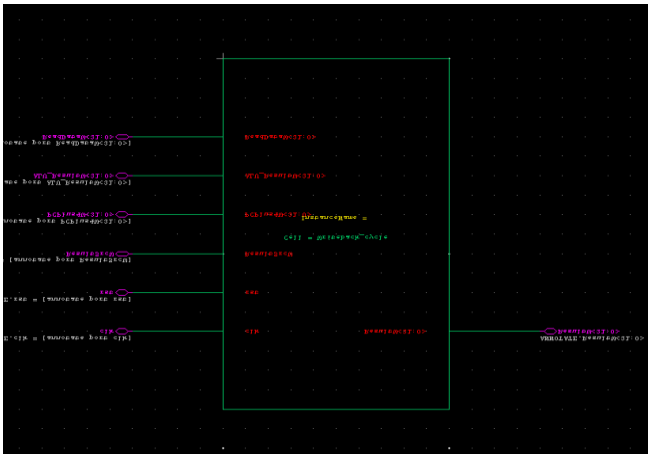


Fig: Symbol generation using the Writeback cycle Verilog HDL.

4. Simulation and Verification:

The entire Verilog design was compiled and simulated using GTKWave. Testbenches were written to provide a sequence of instructions to the processor and observe the signal transitions across all five stages. Functional correctness was verified by checking the output waveforms and register values at each clock cycle.

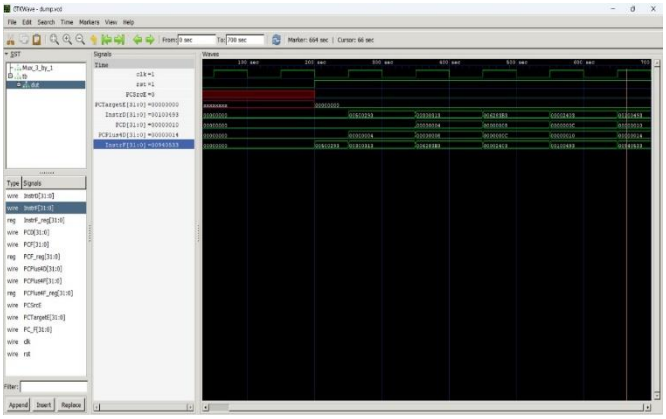


Fig: Simulation of Fetch_Cycle in GTKWave software

I created a pipeline top module that includes the hazard detection unit. All five pipeline stages were implemented and instantiated within this top module. Additionally, I developed a comprehensive testbench to verify the overall functionality and flow of the processor. And the simulation is performed in the GTKwave software. The result of the simulation is shown below. We can see the instruction flow from fetch, decode, and execute stages for each clock cycle overlappingly.

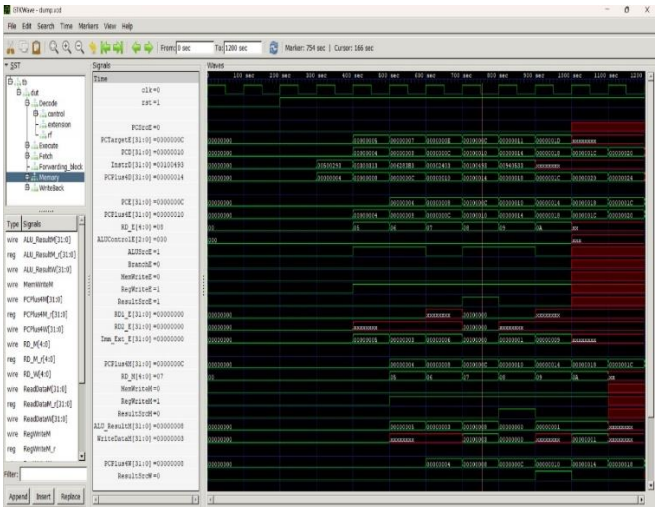


Figure 1

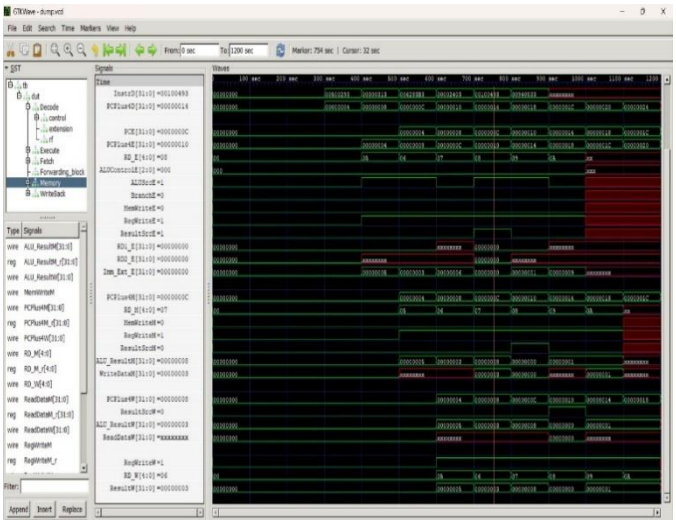


Figure 2

Figures 1 and 2: Final Simulation result of all combined cycles.

5. Schematic and Layout Design:

Using Tanner EDA tools, schematic designs were created for each pipeline stage. Components such as multiplexers, ALUs, registers, and control logic were designed at the transistor level. Once verified, layouts were generated from the schematics for each stage.

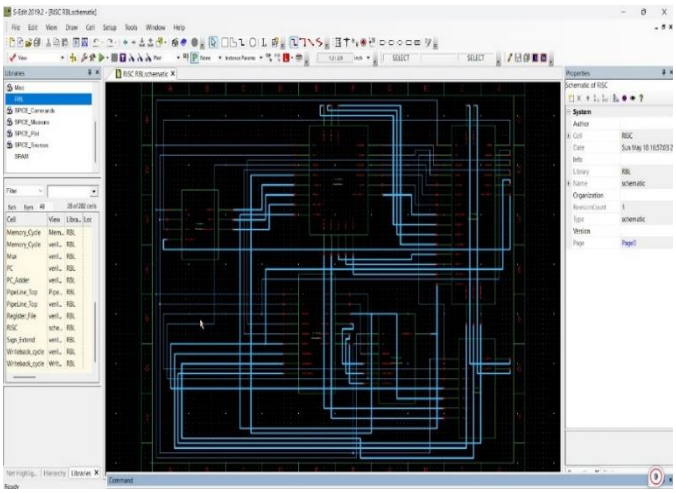
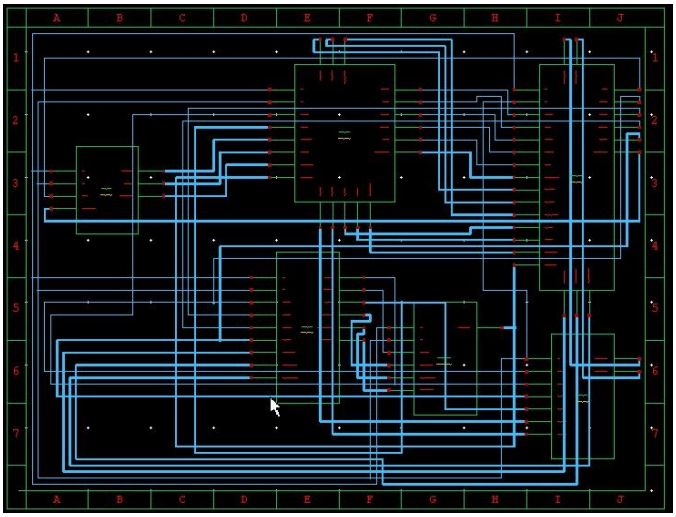


Fig Schematic design of RISC Processor by combining all the corresponding cycle symbols.



5.1 GDSII Generation:

The individual pipeline stage layouts were exported to GDSII format, which is a standard for IC layout representation. The GDSII files were generated using Qflow, an open-source digital synthesis and backend tool, to support layout generation and visualization. Although full physical verification (such as power, timing, and area analysis) was not performed, an estimated layout evaluation was carried out to assess placement and routing feasibility at the 250nm technology node. This methodology ensures a comprehensive processor design flow, from high-level modeling in Verilog to low-level physical design using Tanner EDA and Qflow, showcasing the complete lifecycle of a pipelined MIPS processor architecture.

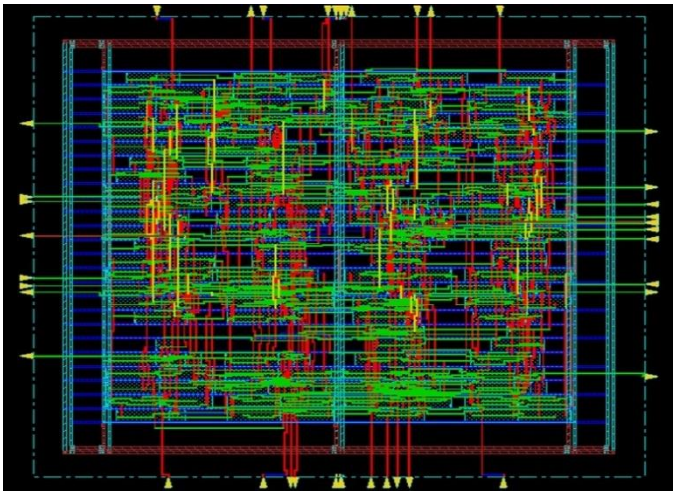


Fig: Layout design of the proposed processor

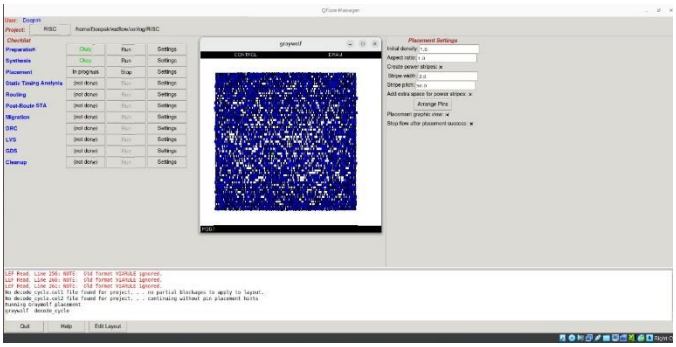


Fig: Placement of cells using Graywolf software.

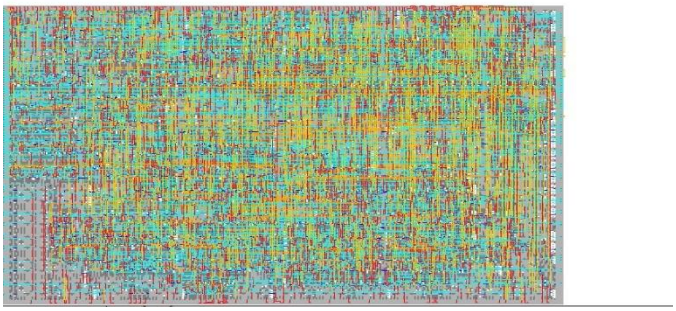


Fig: GDSII Image of proposed processor using Qflow and magic.

6. Estimation:

Theoretically, the use of the Harvard architecture contributes to improved speed and reduced power consumption in processor design. By maintaining separate memory units for instructions and data, the processor achieves parallel access, allowing instruction fetch and data access to occur simultaneously. This increases pipeline throughput and minimizes stalls caused by memory conflicts, thereby enhancing performance.

Additionally, clock gating is a key low-power design technique used to further improve power efficiency. By disabling the clock signal to inactive pipeline stages or modules, unnecessary switching activity is prevented, leading to a significant reduction in dynamic power consumption. When combined with Harvard architecture, clock gating allows the processor to maintain high operational throughput while minimizing power usage, especially in idle or low-activity conditions. These design strategies are theoretically beneficial in embedded or battery-powered systems, making the processor both performance-efficient and power-aware.

7. Conclusions:

This work successfully demonstrates the complete design flow of a 5-stage pipelined 32-bit MIPS processor based on the Harvard architecture. The processor was modeled in Verilog HDL, incorporating modules such as hazard

detection, data forwarding, and pipeline control logic to handle instruction-level parallelism efficiently. The design was verified through waveform simulation using GTKWave, ensuring functional correctness across pipeline stages.

Furthermore, the project extended into the physical design domain, where each pipeline stage was translated into schematic layouts using Tanner EDA and exported into GDSII format using QFlow for placement and routing visualization at the 250nm technology node. While full physical characterization was not performed, a theoretical estimation of performance and power benefits was made based on architectural choices and layout structure.

The adoption of the Harvard architecture enabled parallel access to instruction and data memory, reducing bottlenecks and improving throughput. In addition, the integration of clock gating techniques in the design flow further emphasized the importance of low-power operation, contributing to energy-efficient processor behavior. Overall, this case study provides a comprehensive educational framework for understanding both RTL design and backend physical implementation of pipelined RISC processors.

REFERENCES

[1] Ankita Yadav and Varsha Bendre, "Design and Verification of 16-bit RISC Processor Using Vedic Mathematics," Pimpri Chinchwad College of Engineering, Pune, India, March 2021.

[2] Chandran Venkatesan, Thabsera Sulthana M, Sumithra M.G, Suriya M," Design of a 16-Bit Harvard Structure RISC Processor in Cadence 45nm Technology ", KPR Institute of Engineering and Technology Coimbatore, India https://orcid.org/0000-0003-2412-9493. 2019

[3] Shraddha M. Bhagat and Sheetal U. Bhandari, "Design and Analysis of 16-bit RISC ProcessorPimpri Chinchwad College of Engg, Pune, India.

BIOGRAPHIES:

Tummala Deepak studying 3rd year, Department of Communication Engineering at Institute of Aeronautical Engineering, Dundigal.

Email: **deepaktummala20@gmail.com**

Dr. B Ravi Kumar professor in the Department of Electronics and Communication Engineering at the Institute of Aeronautical Engineering (IARE). He holds a Ph.D. degree in Electronics and Communication Engineering with a specialization in Signal and Image Processing. He has 7 years of experience in teaching field.

Email: **b.ravikumar@iare.ac.in**

Mr. D Veeraswamy

Professor in Department of Electronics and Communication Engineering at the Institute of Aeronautical Engineering (IARE).He holds a Master's degree in Electronics and Communication Engineering with a specialization in Communication. He has 10 years of experience in the teaching field.

Email: d.veeraswamy@iare.ac.in

Basavaraj Aland, studying 3rd year, Department of Communication Engineering at the Institute of Aeronautical Engineering, Dundigal.

Email: basavarajaland31@gmail.com