SRFP-2024

Design and Development of an RV-32 IM Processor Board with UART Integration

Guide:
Name: Dr. Sujay Deb
Institution: IIIT-D

Submitted by:
Name: Vadali Susmitha Sri Ramani
Application no: ENGS764
Institution: SVECW
Date: July 17, 2024

Abstract

This report details the design and development of an RV32IM processor during an eight-week summer internship. The RV32IM is a 32-bit RISC-V processor that includes integer multiplication and division instructions, enhancing its computational capabilities. The project focused on several key aspects of processor design, including the creation and implementation of the immediate generator module, development of pipelined stages, management of hazard detection and forwarding mechanisms, stalling and overall system integration. Throughout the internship, significant emphasis was placed on ensuring efficient pipeline performance and accurate hazard resolution to optimize processor functionality. Although testing was not completed within the internship period, the groundwork laid provides a robust foundation for future verification and validation. This report presents a comprehensive overview of the methodologies employed, the challenges encountered, and the solutions devised, providing valuable insights into the intricate process of processor design and development.

Acknowledgment

I would like to extend my sincere gratitude to all those who have supported and guided me throughout my internship on the design and development of the RV32IM processor. First and foremost, I would like to thank **Dr.Sujay Deb**, my internship supervisor, for their invaluable guidance, constant encouragement, and insightful feedback. Their expertise and support have been instrumental in the successful completion of this project. I am deeply grateful to **Tarun Sharma**, my mentor, for their patience, support, and willingness to share their knowledge and experience. Their mentorship has significantly contributed to my understanding of processor design and development. I would also like to express my appreciation to the entire team at **IIIT-D** for creating a collaborative and motivating environment. Their cooperation and assistance have been crucial in overcoming the challenges faced during the project. A special thanks to the **Indian Academy of Sciences** for organizing this internship program and for their continuous support throughout my tenure. This internship has been a profoundly enriching experience, and I am grateful for the opportunity to work on such a challenging and rewarding project.

Contents

Chapter 1

Introduction

1.1 Objectives of the Internship

During my 8-week summer internship at **III-D**, I worked as an intern in the **Embedded Systems**, focusing on the design and development of the RV32IM processor. The primary objective of my internship was to contribute to the implementation and testing of this open-source RISC-V architecture processor.

I was actively involved in the design and implementation of various critical modules, including the Single Cycle Processor, Pipelined Processor and handling Hazards . My role also involved integrating these components to ensure the smooth functioning of the processor. Additionally, I documented the design process and prepared technical reports and presentations.

Throughout the internship, I enhanced my technical skills in Verilog programming and digital design, faced and overcame numerous challenges, and improved my teamwork and communication skills. One of my notable achievements was successfully implementing the pipelined stages, which contributed to the processor's overall efficiency.

Based on my experience, I suggested potential improvements for future work, such as optimizing specific modules and conducting more comprehensive integration tests. Overall, this internship was a highly valuable experience, significantly contributing to my academic and professional growth. I am grateful to my mentors and team members for their guidance and support.

1.2 Importance of RISC-V Architecture

A Reduced Instruction Set Computer is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions rather than the highly-specialized set of instructions typically found in other architectures. RISC is an alternative to the Complex Instruction Set Computing (CISC) architecture and is often considered the most efficient CPU architecture technology available today.

The RISC-V architecture is built upon a set of key design principles that contribute to its performance, efficiency, and adaptability. key design principle of the RISC-V architecture is its modularity and extensibility. Modularity refers to the organization of the ISA into separate, independent components that can be combined in various ways to create a customized processor. Extensibility, on the other hand, refers to the ability to add new instructions, features, or extensions to the ISA without disrupting existing functionality. The standard extensions as follows:

M - For Integer Multiplication and Division.

A - For Atomic Instructions.

F- For Single-Precision Floating-Point.

D - For Double-Precision Floating-Point.

C - For Compressed Instructions.

B - For Bit Manipulation.

V - For Vector Operations.

It is an open-source instruction set architecture used to develop custom processors for a variety of applications, from small embedded systems to powerful supercomputers. It simplifies hardware implementation. This can lead to faster execution times and lower power consumption. And it executes one action per instruction, means each instruction typically completes in one clock cycle.

For chip designers, RISC processors simplify the design and deployment process and provide a lower per-chip cost due to the smaller components required. Because of the reduced instruction set and simple decoding logic, less chip space is used, fewer transistors are required, and more general-purpose registers can fit into the central processing unit(CPU).

Chapter 2

Project Description

Problem Statement

The objective of this project is to design, implement, and validate a 32-bit RISC-V processor with the integration of IM that meets the following criteria:

Functionality: The processor must accurately implement the RV32IM instruction set, including all standard integer instructions and the additional multiplication and division instructions defined by the 'M' extension.

Efficiency: The design should optimize for performance, minimizing latency and maximizing throughput through effective pipelining and hazard management techniques.

Scalability and Modularity: The processor should be designed in a modular fashion, allowing for easy integration of additional features and extensions in the future.

2.1 Objectives and Methodology

Objectives The primary objectives of my internship were to design and develop a functional RV32IM processor, integrating integer multiplication and division instructions as per the RISC-V ISA. Additionally, the goals included incorporating TileLink Protocol for communication and efficient data transfer, implementing pipelining techniques to enhance processor performance, and conducting rigorous testing to ensure reliability and efficiency. Documentation and presentation of the project's progress and outcomes were also essential objectives.

Methodology To achieve these objectives, I adopted a systematic approach beginning with an extensive literature review on the RISC-V architecture and RV32IM instruction set. This was followed by defining architecture specifications and designing the processor components using Verilog in the Xilinx Vivado. The implementation phase involved developing core components, integrating TileLink, and applying pipelining techniques. Comprehensive testing and validation were conducted using a detailed testbench, followed by debugging and optimization.

2.2 Design and Implementation

The design and implementation of the RV32IM processor involved several critical steps to ensure a robust and efficient architecture. The design phase began with defining the processor's architecture and its key components, including the ALU, control unit, registers, and memory interfaces. Verilog was chosen as the hardware description language for its flexibility and widespread use in digital design. The implementation phase focused on developing these components and integrating them into a cohesive processor unit. Special attention was given to the implementation of integer multiplication and division instructions, as well as the incorporation of UART for serial communication. Pipelining techniques were applied to allow multiple instructions to be processed simultaneously, thereby improving overall performance. Rigorous testing was conducted using a comprehensive testbench to verify functionality and performance. Debugging and optimization were carried out to resolve any issues and enhance the processor's efficiency. The final design was thoroughly documented, highlighting the implementation details, challenges faced, and solutions devised.

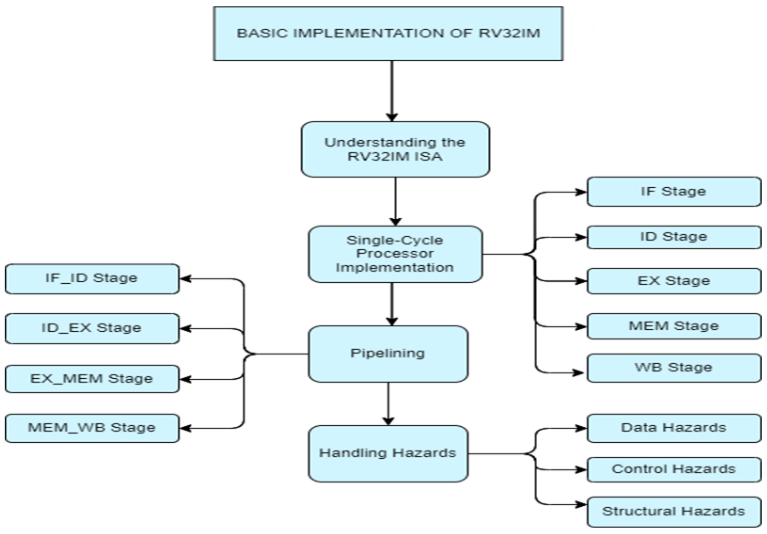


Figure 2.1: Flow Diagram for Implentation of RV32IM

2.2.1 Single Cycle Processor

The single cycle processor design focuses on executing each instruction within a single clock cycle, ensuring simplicity and straightforward control.The sequential execution model ensures that each instruction completes its execution in a predictable and fixed amount of time, simplifying the control logic and facilitating easier debugging and validation of the processor design. This architecture involves a centralized control unit that coordinates the execution of instructions, managing the fetch, decode, execute, memory access, and write-back stages. Elaboration of each stage as follows:

1. Instruction Fetch (IF) Stage In this stage, the processor retrieves the instruction from memory. The Program Counter (PC) holds the address of the current instruction, which is sent to the instruction memory. The instruction is then fetched and stored in the Instruction Register (IR). Simultaneously, the PC is incremented to point to the next instruction.

2. Instruction Decode (ID) Stage The fetched instruction is decoded in this stage. The opcode is extracted to determine the type of instruction and the necessary control signals are generated. The processor also reads the source operands from the register file. For R-type instructions, the operands are read directly from the registers specified in the instruction. For I-type and other instructions, immediate values are extracted from the instruction.

3. Execution (EX) Stage In the execution stage, the actual computation takes place. The ALU performs the arithmetic or logical operations based on the decoded instruction. For branch instructions, the target address is computed in this stage. Load and store instructions also calculate the effective memory address using the ALU.

4. Memory Access (MEM) Stage This stage is involved only for load and store instructions. For load instructions, the data is read from the memory location computed in the EX stage and placed into a register. For store instructions, the data from a register is written to the computed memory address. Other types of instructions bypass this stage.

5. Write-Back (WB) Stage In the final stage, the result of the computation or the data read from memory is written back to the register file. This completes the instruction cycle, with the output being stored in the destination register specified in the instruction.

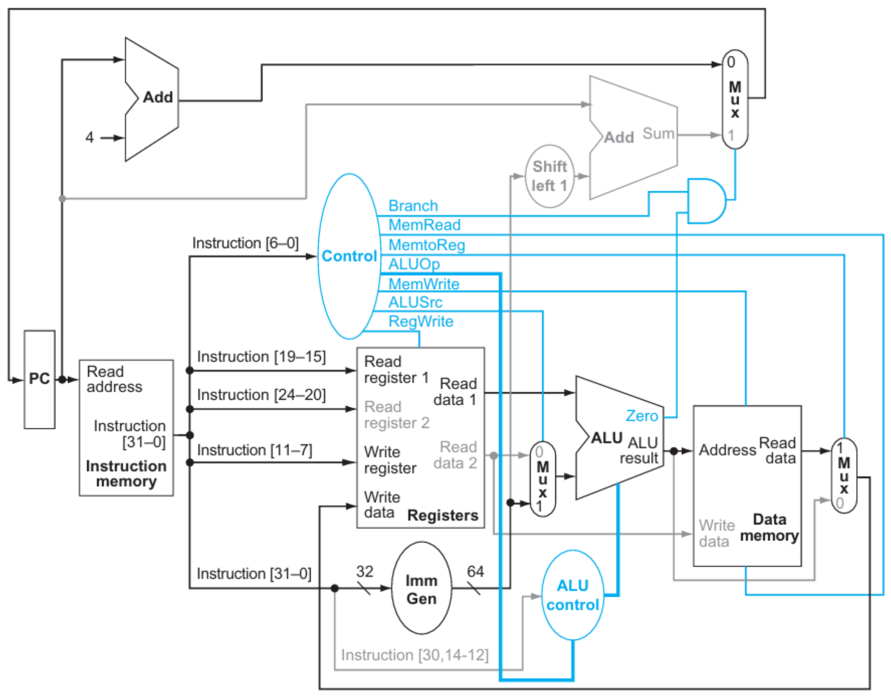


Figure 2.2: Block diagram of Single Cycled non-pipeline Processor

2.2.2 Single Cycle Pipelined Processor

A pipelined processor is a type of microprocessor design that improves performance by overlapping the execution of multiple instructions. This is achieved by dividing the processor's work into separate stages, each performing a different part of the instruction execution process. These stages work simultaneously, allowing multiple instructions to be processed at different stages of completion at the same time.

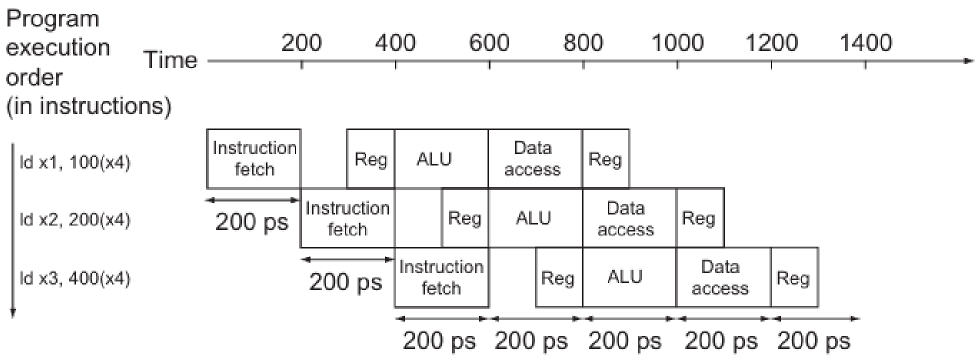


Figure 2.3: Single-cycle pipelined execution

The datapath consists of five stages as follows:

1. **Instruction fetch:** The instruction being read from memory using the address in the PC and then being placed in the IF/ID pipeline register. The PC address is incremented by 4 and then written back into the PC to be ready for the next clock cycle. This PC is also saved in the IF/ID pipeline register in case it is needed later for an instruction, such as beq. The computer cannot know which type of instruction is being fetched, so it must prepare for any instruction, passing potentially needed information down the pipeline.

2. **Instruction decode and register file read:**The instruction portion of the IF/ID pipeline register supplying the immediate field, which is sign-extended to 64 bits, and the register numbers to read the two registers. All three values are stored in the ID/EX pipeline register, along with the PC address. We again transfer everything that might be needed by any instruction during a later clock cycle.

3. **Execute or address calculation:** The load instruction reads the contents of a register and the sign-extended immediate from the ID/EX pipeline register and adds them using the ALU. That sum is placed in the EX/MEM pipeline register.

4. **Memory access:** The load instruction reading the data memory using the address from the EX/MEM pipeline register and loading the data into the MEM/WB pipeline register.

5. **Write-back:** Reading the data from the MEM/WB pipeline register and writing it into the register.

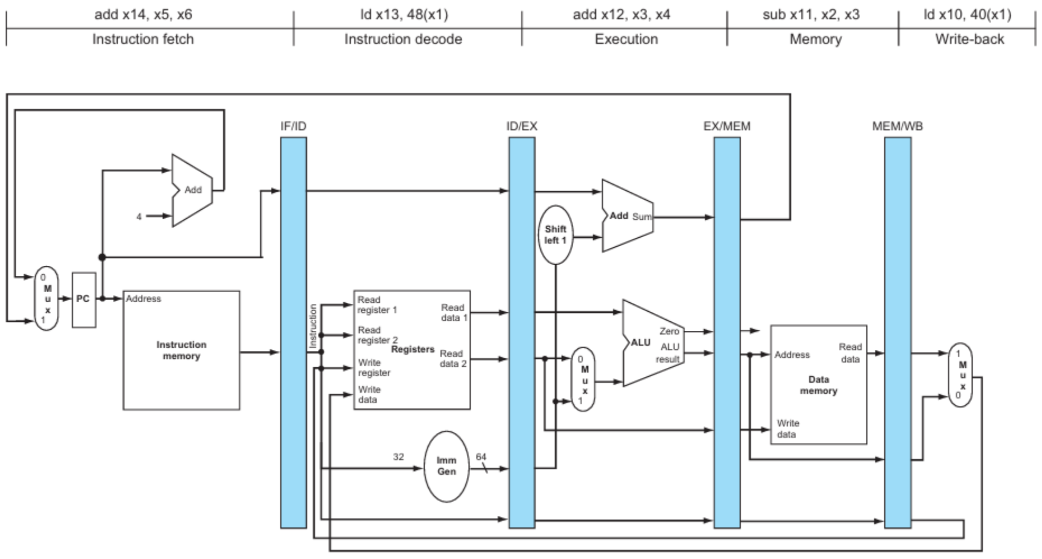


Figure 2.4: Datapath for the Single-cycle Pipelined Processor

Control Path: To specify control for the pipeline, we need the control values during each pipeline stage. Because each control line is associated with a component active in only a single pipeline stage, we can divide the control lines into five groups according to the pipeline stage.

1. Instruction fetch: The control signals to read instruction memory and to write the PC are always asserted, so there is nothing special to control in this pipeline stage.

2. Instruction decode/register file read: The two source registers are always in the same location in the RISC-V instruction formats, so there is nothing special to control in this pipeline stage.

3. Execution/address calculation: The signals to be set are ALUOp and ALUSrc. The signals select the ALU operation and either Read data 2 or a sign-extended immediate as inputs to the ALU.

4. Memory access: The control lines set in this stage are Branch, MemRead, and MemWrite. The branch if equal, load, and store instructions set these signals, respectively. Recall that PCSrc selects the next sequential address unless control asserts Branch and the ALU result was 0.

5. Write-back: The two control lines are MemtoReg, which decides between sending the ALU result or the memory value to the register file, and RegWrite, which writes the chosen value.

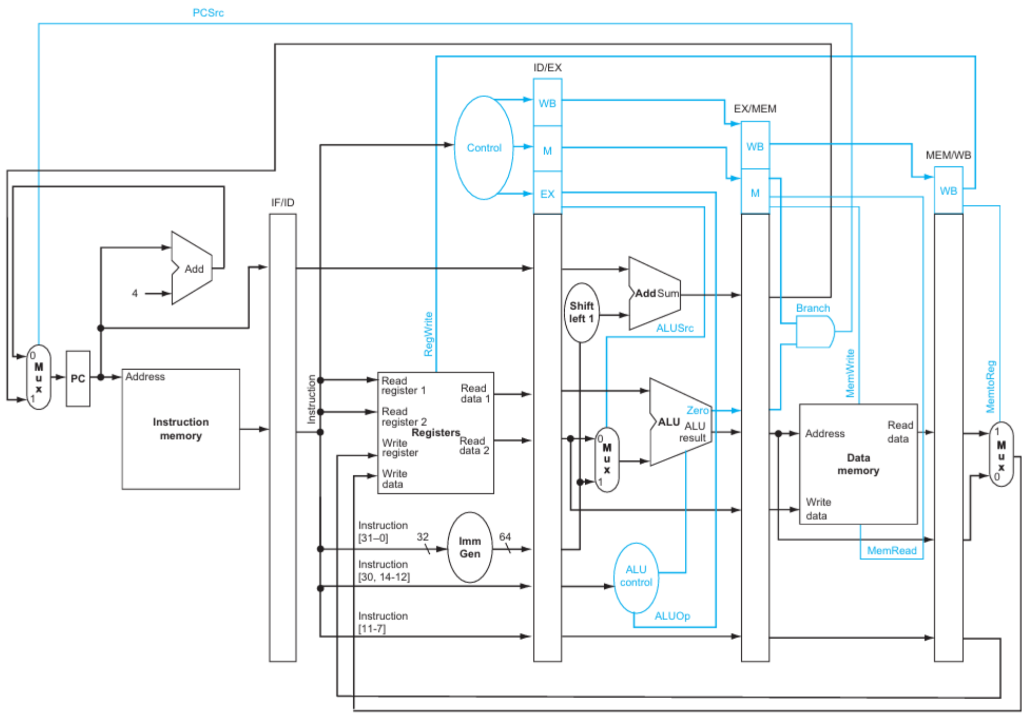


Figure 2.5: Control Path for the Single-cycle Pipelined Processor

2.2.3 Handling Hazards

Introduction to Hazards: In the design of the RV32IM pipelined processor, hazards pose significant challenges to achieving smooth and efficient instruction flow. Hazards are situations that prevent the next instruction from executing in its designated clock cycle, leading to potential delays and performance degradation. The primary types of hazards encountered are data hazards, control hazards, and structural hazards.

Data Hazard:s Data hazards occur when instructions that exhibit data dependence modify data at different stages of the pipeline. These hazards can manifest as:

(i)Read After Write (RAW): Occurs when an instruction depends on the result of a previous instruction that has not yet completed.

(ii)Write After Read (WAR): Occurs when an instruction writes to a destination before a previous instruction reads it.

(iii)Write After Write (WAW): Occurs when multiple instructions write to the same destination in different stages. To mitigate data hazards, we implemented techniques such as:

Forwarding (Bypassing): Allows data to be sent directly from one pipeline stage to another to resolve RAW hazards.

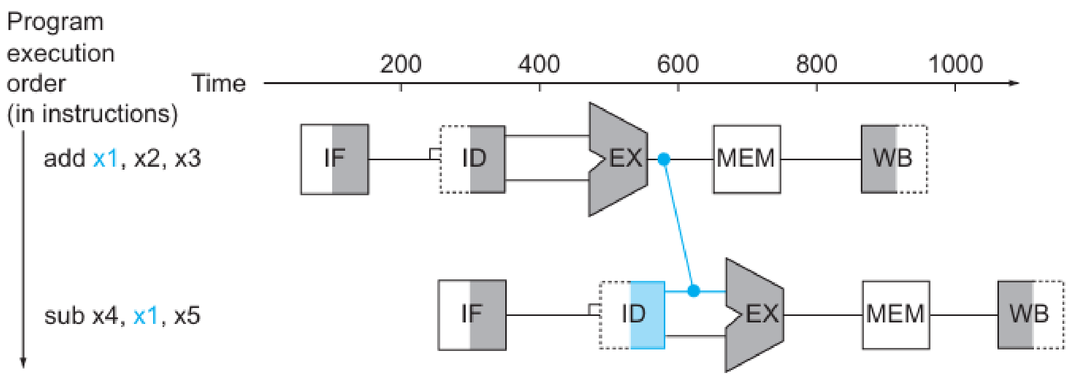


Figure 2.6: Graphical representation of forwarding

Pipeline Stalling: Temporarily halts the pipeline until the hazard is resolved.

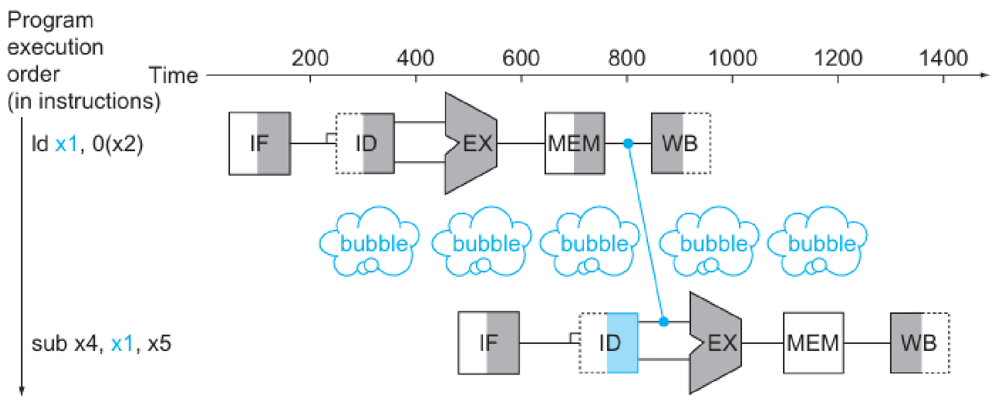


Figure 2.7: Graphical representation of stalling

Control Hazards: Control hazards arise from branch instructions that alter the flow of execution.Also known as branch hazard. To address control hazards in our RV32IM processor, we employed branch prediction techniques:

(i)Branch Prediction: Static and dynamic branch prediction methods that guess the outcome of a branch to minimize stalls.

(ii)Delayed Branch: Defers the execution of branch instructions to avoid control hazards.

(iii)Branch Target Buffer (BTB): Stores the destinations of recently executed branches to predict future ones.

Structural Hazards: Structural hazards occur when the hardware resources required for simultaneous operations are insufficient. In the RV32IM design, structural hazards were mitigated through:

(i)Resource Duplication: Duplicating resources, such as ALU's or memory ports, to resolve structural hazards.

(ii)Pipeline Scheduling: Careful scheduling of operations to ensure that resources are available when needed. Practical Implementation and Challenges.

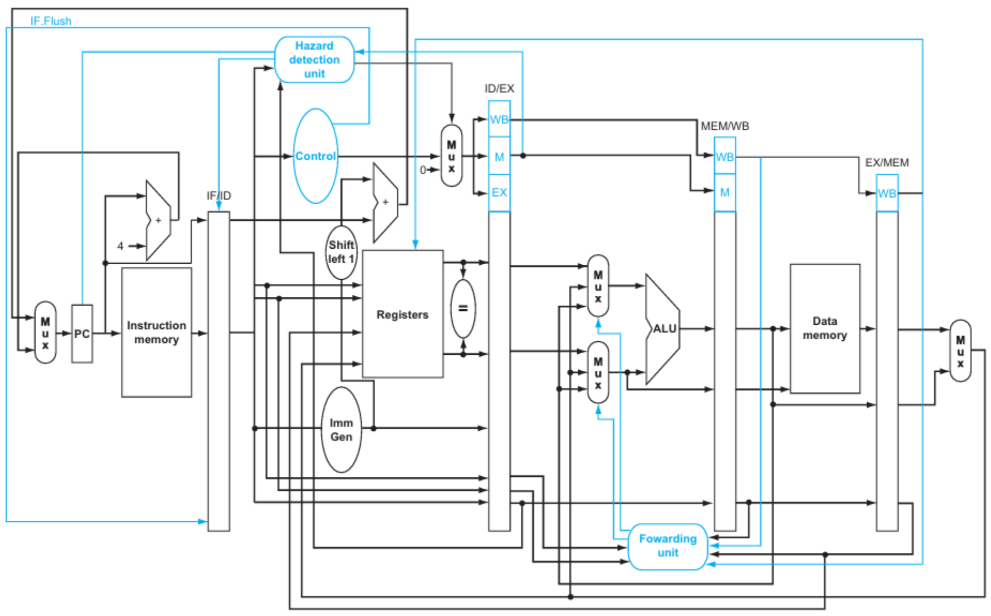


Figure 2.8: Block Diagram of Hazard Detection

2.2.4 TileLink

Communications Outside the Core Complex (TileLink) In FE31-G0002, communication outside the Core Complex is managed using the TileLink protocol. To understand its role in the Freedom 310, we must explore TileLink's functionality. TileLink is a chip-scale interconnect standard providing multiple masters with coherent memory-mapped access to memory and other slave devices.

TileLink features a physically addressed, shared-memory system implemented over scalable, hierarchically composable, point-to-point networks. It offers coherent access for various caching and non-caching masters and can scale from simple slave devices to high-throughput slaves. Key features include cache-coherent shared memory supporting a MOESI-equivalent protocol, out-of-order completion for improved throughput, decoupled interfaces for easier register-stage insertion, stateless bus-width adaptation, and power-aware signal encoding.

TileLink has three conformance levels: TL-UL (TileLink-Uncached Lightweight), TL-UH (TileLink Uncached Heavyweight), and TL-C (TileLink-Cached). TL-UL is the simplest, handling only Get (Read) and Put (Write) operations related to memory access. TL-UH adds atomic operations, Hint operations, and Burst operations. TL-C incorporates cache block transfers on top of TL-UH operations.

TileLink operates through five channels: A, B, C, D, and E, each serving a specific function. TL-UL and TL-UH do not utilize channels B, C, and E, making them optional for designs not using the TL-C conformance level. Channels A and D are fundamental for memory access operations. Channel A transmits requests for operations on specified address ranges, while Channel D returns data responses or acknowledgment messages.

The TL-C conformance level adds three channels to manage permissions on cached data blocks. Channel B handles requests for operations at addresses cached by a master agent, Channel C responds to Channel B requests, and Channel E provides final acknowledgment for cache block transfers.

The operations supported by TL-UL include Get, AccessAckData, PutFullData, PutPartialData, and AccessAck. TL-UH adds Intent and HintAck operations. Arithmetic and LogicalData operations are atomic, with responses being AccessAckData. The basic channels present in all TileLink operations, Channel A and Channel D, facilitate the primary memory access functions.

Channel A, flowing from master to slave interface, carries request messages to specific addresses. It includes signals like operation code (a$_{o}pcode), parametercode(a_{p}aram), address(a_{a}ddress)$

The TileLink protocol is crucial for handling cache misses, allowing data to be loaded from memory outside the core-complex. While a complete exploration of TileLink is beyond this report's scope, understanding TL-UL and TL-UH without atomic instructions is sufficient for working with an RV32IMC processor.

The GPIO and other peripherals in the FE310-G0002 include UART, SPI, PWM, and I2C, connecting to the Core Complex via the TileLink bus. These peripherals manage digital signals, sensor data, and external device communications, enhancing the system's versatility for various applications.

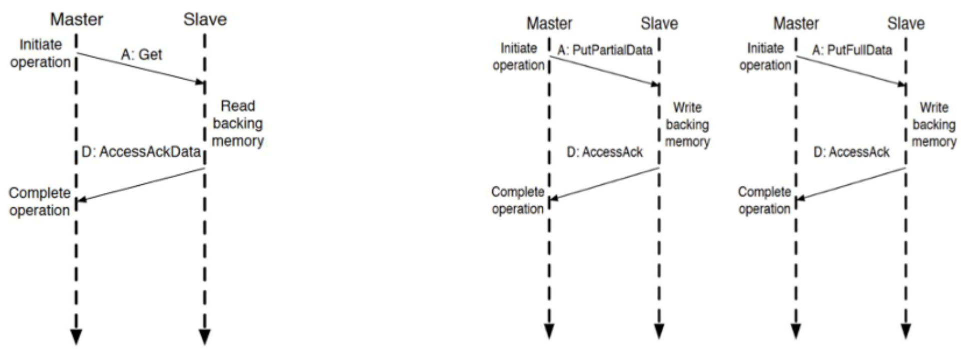


Figure 2.9: TileLink- Uncached Lightweight operations

2.2.5 Integration of RV32I with M

The RV32IM processor architecture combines the foundational principles of the RV32I base ISA with advanced integer multiplication and division capabilities provided by the M extension. This integration not only extends the processor's computational capabilities but also enhances its suitability for a wide range of applications. As a pivotal part of the RISC-V ecosystem, the RV32IM architecture continues to drive innovation in embedded systems and computational devices, setting standards for efficiency, performance, and versatility in modern processor designs.

Multiplier

The multiplier in the RV32IM architecture is a critical component that significantly enhances computational efficiency by supporting 32-bit integer multiplication. And we use Dadda Multiplier.In a 32 x 32 bit multiplication the final product would be 64 bit long and based on the instruction the higher 32 bits are stored in one register and the lower 32 bits are stored in 32 bit another register.

The **Dadda Multiplier** is a type of hardware multiplier, invented by the "Luigi Dadda" in 1965.And this is optimized for speed and area.The main idea behind the Dadda Multiplier is to use a combination of partial product reduction and the parallel reduction stages to minimize the no of partial products. It is a parallel multiplier based on the Dadda tree, a type of Wallace tree multiplier that reduces the partial products more efficiently.

Working of Dadda Multiplier

The Dadda multiplier operates in three main stages:

(i)Generation of Partial Products: This stage involves the generation of all possible partial products from the two multiplicands. For two n-bit numbers, this will generate n×n partial products.

(ii)Reduction of Partial Products: The Dadda tree reduces the partial products to two rows using a sequence of reduction steps. This is done using full adders and half adders to minimize the number of rows of partial products step-by-step, ensuring that each step maintains a specific count of bits in each column.

The progression of reduction is controlled by maximum height sequence dj, defined

d1 = 2 and dj+1 = floor(1.5*dj) which leads to the sequence 2,3,4,6,9,13....

(iii)Final Addition: The last two rows of partial products are added together using a fast adder (like a carry-lookahead adder) to produce the final product.

Given the depth is 9 we take the largest value in the series which is lesser or equal to the depth.We find that that d4 = 6 which is lesser than 8. Half-Adders and Full Adders are to be used in the columns where the depth is more than 6. The carry must also be taken into consideration when deciding to use the Half-Adder.This continues till we reach a depth of 2. At this point a simple addition would give the required product.

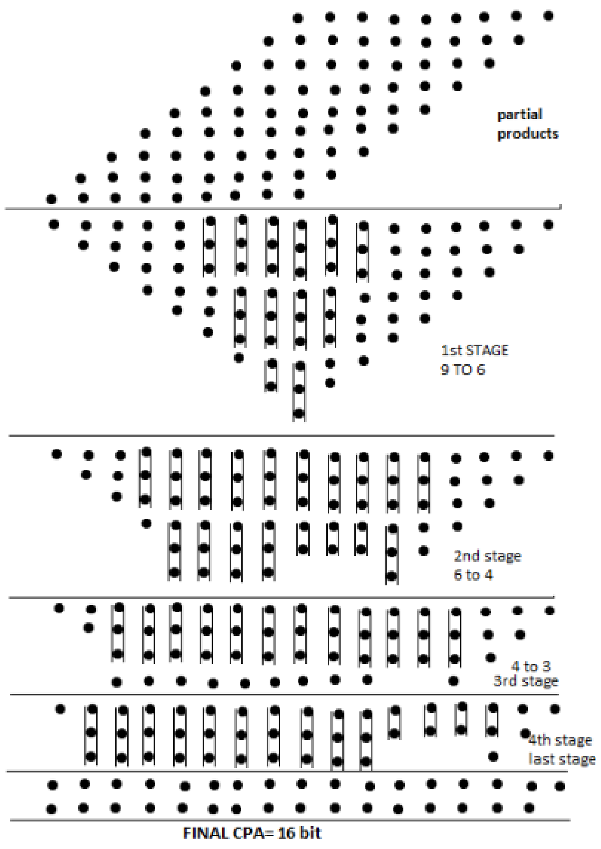


Figure 2.10: Multiplier Reduction Scheme

The main advantages of this Dadda Multiplier are:

(i)High Speed: Dadda multipliers are designed to reduce the number of partial products as quickly as possible.

(ii)Reduced Complexity: The algorithm simplifies the reduction of partial products by focusing on minimizing the height of the matrix, which can lead to less complex circuitry.

(iii)Lower Propagation Delay: The logarithmic nature of the reduction process means that the delay increases logarithmically with the number of bits, which is much faster than a linear increase. This makes Dadda multipliers particularly advantageous for large bit-width multiplications.

(iv)Parallelism: The architecture allows for parallel processing, which significantly enhances the speed of multiplication operations.

Divider

The divider module within the RV32IM microprocessor represents a critical component of its arithmetic logic unit (ALU), essential for handling division operations between 32-bit integers. Its design intricacies underscore its role in enabling complex mathematical computations fundamental to modern processors.

Synchronous Operation and Clock Synchronization: Central to the divider module's functionality is its synchronous operation, synchronized with the system clock (clk). This design choice ensures that all internal operations, including data loading, shifting, comparison, and result generation, occur in harmony with the processor's timing signals. By adhering strictly to clock cycles, the module optimizes performance and maintains consistency in computation, crucial for real-time applications and deterministic processing.

Initialization and Data Handling: Upon receiving a start signal, the divider module initializes by loading the dividend and divisor values into dedicated registers. These registers serve as temporary storage for operands during the division process.This structured approach to data handling ensures that each component of the division algorithm operates efficiently within the module's architecture.

Quotient and Remainder Calculation: During each iteration, the module compares the current remainder with the divisor. If the remainder is greater than or equal to the divisor, subtraction occurs, and the corresponding bit is set in the quotient. This iterative approach continues until all bits of the dividend have been processed (bits equals zero), at which point the module indicates completion by setting done signal. This meticulous process ensures accurate computation of both the quotient and remainder, critical for maintaining precision in mathematical operations performed by the microprocessor.

Integration and Performance Considerations: Integrating seamlessly into the broader RV32IM microprocessor architecture, the divider module contributes to overall computational efficiency and performance. Its design not only supports standard division operations but also addresses edge cases and exceptional conditions, ensuring robust handling of diverse input scenarios. By managing internal data paths and control signals effectively, such as start and done, the module orchestrates division operations with minimal latency, enhancing overall processor throughput.

In conclusion, the divider module exemplifies the intersection of advanced algorithm design, synchronous processing principles, and integration within a complex microprocessor architecture. Its role in enabling efficient division operations underscores its significance in modern processor design, contributing to the performance and reliability of computing systems across diverse applications.

Chapter 3

Weekly Progress

The progress made during my 8-week internship on the design and development of the RV32IM processor.

WEEK 1:

During the first week, we focused on comprehending the RISC-V Instruction Set Architecture (ISA). This involved understanding the data flow and identifying various hazards, including structural, data, and control hazards. The concept of pipelining was also explored, which is crucial for enhancing processor performance. Additionally, I reviewed computer architecture and Verilog to prepare for the implementation of the RV32 processor on Vivado. This foundational knowledge was essential for the upcoming tasks and helped in setting the stage for the project.

WEEK 2:

In the second week, I concentrated on mastering the IP protocols, specifically Tile-Link UL (Uncached Lightweight) and UH (Uncached Heavyweight). This understanding was pivotal as it allowed for the effective integration of peripherals into the system-on-chip (SoC) design. By comprehending the nuances of Tile-Link UL, I ensured that lightweight, low-latency communication with simple peripherals was streamlined. The protocol's uncached memory access model was particularly beneficial for peripherals that do not demand complex cache coherency, simplifying the integration process and boosting system efficiency for specific low-bandwidth components. Concurrently, I delved into the Tile-Link UH protocol for high-bandwidth peripherals, ensuring coherent memory-mapped access and preventing data corruption.

WEEK 3:

The third week marked significant progress in the implementation phase. I developed Verilog code for essential components of the processor, including the register file, immediate generator, control unit, ALU control unit, and ALU. This step was critical as these components form the core of the processor's functionality. The process involved detailed design and coding to ensure each module performed its designated tasks correctly and efficiently. Despite some challenges in ensuring the correctness and efficiency of the Verilog code, the week ended with the successful development of these fundamental components.

WEEK 4:

In the fourth week, the focus shifted to verification and testing. I designed comprehensive test benches for each implemented component to ensure functionality and correctness. Rigorous testing was conducted to validate the behavior and integration of the register file, immediate generator, control unit, ALU control unit, and ALU modules.This process involved running various test cases and debugging any issues that arose.The extensive testing ensured that each module operated as expected, laying a solid foundation for the subsequent stages of the project.

WEEK 5:

During the fifth week, I implemented the pipelining stages and hazard management mechanisms. I also developed hazard detection and forwarding mechanisms to manage data and control hazards. These mechanisms were crucial in ensuring smooth data flow between pipeline stages and minimizing pipeline stalls. The successful implementation of these stages and mechanisms was a significant milestone in the project, setting the stage for system integration.

WEEK 6:

The sixth week was dedicated to system integration and initial testing. I integrated the individual modules, including the register file, ALU, control unit, and pipelined stages, into a cohesive RV32IM processor system. This involved ensuring proper data and control signal propagation between the modules.Once integrated, I created test programs to evaluate the functionality of the system and ran simulations to verify the correct execution of basic RISC-V instructions. Initial testing helped identify and resolve integration issues, achieving a stable system ready for more comprehensive testing.

WEEK 7:

In the seventh week, we focused on the integration of RV32I with M-extension. I conducted the testing and debugging of RV32IM. And debugging was performed using behavioural analysis and debug tools in Xilinx Vivado to trace and resolve issues. Several bugs related to control flow and data dependencies were identified and fixed. Additionally, I optimized the design for performance and reduced latency where possible, ensuring the processor operated efficiently.And we concentrate on the cache controller to integrate with the processor.

WEEK 8:

The final week was dedicated to final verification and Documentation.Extensive testing was performed to confirm the processor's compliance with the RISC-V ISA specifications, ensuring the correct operation of all implemented instructions and features. A comprehensive report summarizing the project's objectives, methodologies, results, and conclusions was also created. Additionally, I prepared presentation materials to showcase the project work and findings.

Chapter 4

Challenges and Solutions

Designing an RV32IM processor presents challenges in understanding its complex instruction set, integrating various components effectively, and optimizing performance. This section outlines these challenges and practical solutions to ensure a successful processor design.

(i)Instruction Memory Creation and Module Connections:

During our RV32IM project, we encountered challenges in creating the instruction memory and establishing connections between modules.

Solution: Ensure that the instruction memory is correctly initialized with the program instructions. Double-check the addressing and data width requirements to match the processor specifications. For module connections, verify the signal paths and ensure proper interfacing between modules using appropriate protocols (such as TileLink).

(ii)Hazards Handling and I with M Integration:

Additionally, we faced difficulties in handling hazards and integrating the I (Instruction Fetch) stage with the M (Memory Access) stage.

Solution: Implement forwarding paths and stall mechanisms to handle hazards effectively. Ensure that the I (Instruction Fetch) stage does not fetch incorrect or stale instructions by synchronizing with the M (Memory Access) stage properly. Use hazard detection and resolution techniques such as data forwarding and pipeline stalls.

(iii)TileLink Protocol Channel Issues:

Furthermore, in implementing the TileLink protocol, we noted discrepancies where one channel (A) produced output while another channel (D) consistently returned zero.

Solution: Investigate the TileLink protocol specifications and ensure that channel A and channel D configurations align with expected behaviors. Check for any misconfigurations or data path issues that might be causing the discrepancy. Verify that all necessary signals are properly connected and synchronized between modules.

Chapter 5

Results and Testing

In this section, we present the comprehensive results and testing outcomes of the RV32IM processor design. We detail the functional verification against the RISC-V ISA specifications, performance metrics including clock speed and execution times, and discuss simulation results and RTL Designs.

Functional Testing:

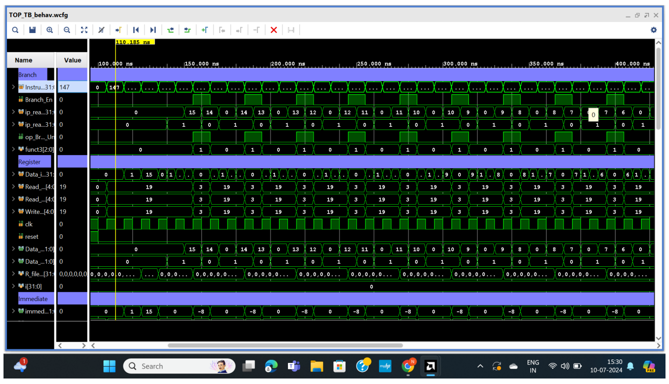
Functional testing of your RV32IM processor involves meticulous verification to ensure precise implementation of the RISC-V ISA functionalities. This process encompasses rigorous testing of integer arithmetic, multiplication, division, load/store operations, and control transfers. Additionally, it includes validation of register file operations, accurate instruction decoding, proper memory access handling, and robust exception management. Through systematic testing methodologies and comprehensive test suites, we validate the processor's ability to execute instructions accurately and reliably across diverse scenarios and edge cases.

Verification Against Specifications:

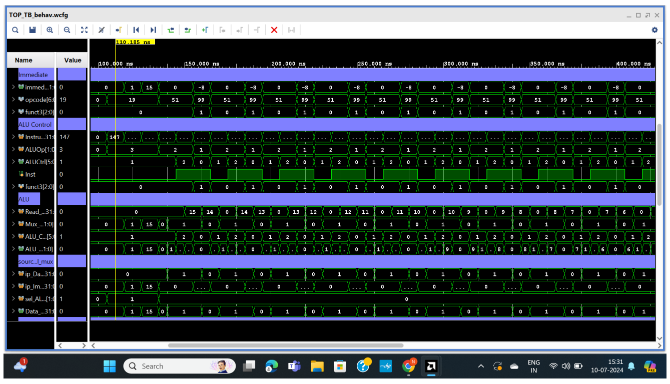
Verification against the RV32IM ISA specifications was rigorously conducted across key areas. We validated precise instruction decoding for integer arithmetic, multiplication, division, and control transfers to ensure accurate execution. Register operations were scrutinized for correct data handling, adhering closely to ISA guidelines. Memory access tests rigorously verified load/store operations for accurate data management and address handling. Thorough testing of exception handling mechanisms confirmed robust responses to interrupts and errors, ensuring reliable operation in varied scenarios. Our systematic testing protocols and simulations affirmed that our RV32IM processor design meets all ISA requirements, ensuring consistent and dependable performance.

Simulation Results:

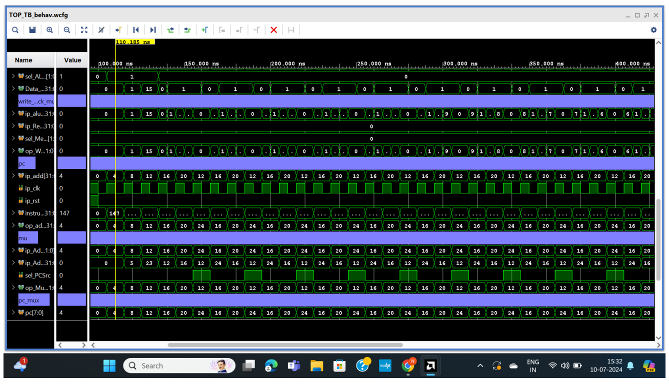
Our simulation tests yielded insightful observations, supported by waveform captures illustrating operational dynamics. We identified occasional timing issues and data hazards during complex instruction sequences, prompting adjustments in pipeline stages and hazard detection logic.



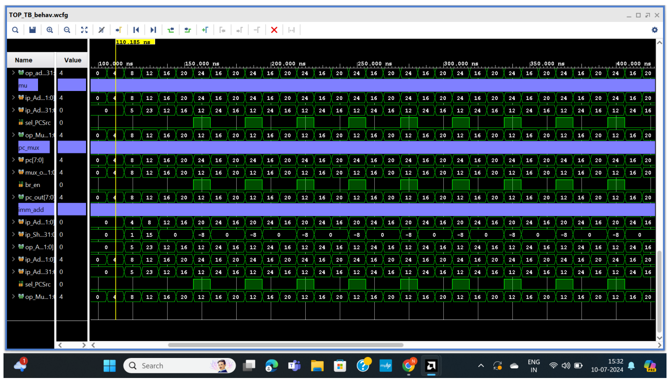
(a)



(b)



(c)



(d)

Figure 5.1: Behavioural model of Single Cycle Non-Pipelined Processor

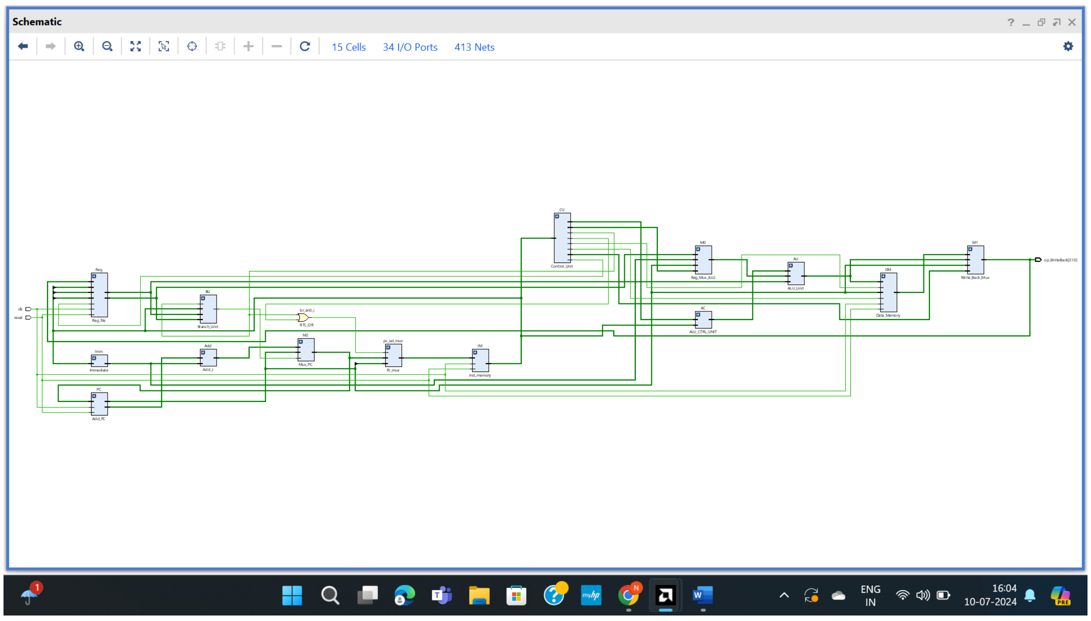
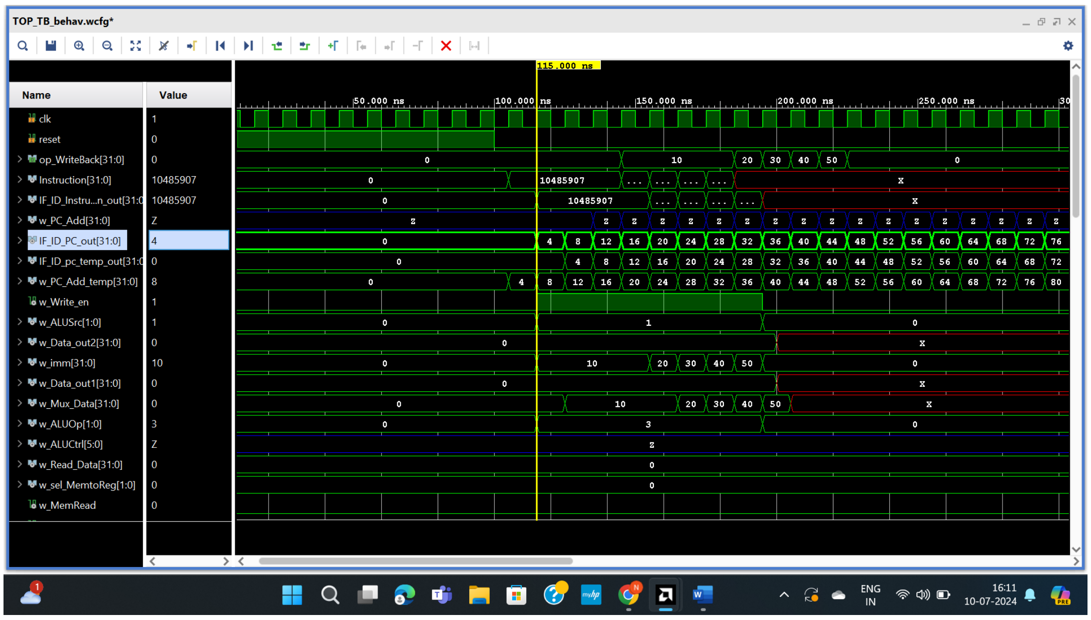


Figure 5.2: RTL Design for Single Cycle Non-Pipelined Processor



Name	Value
clk	1
reset	0
op_WriteBack[31:0]	0
Instruction[31:0]	10485907
IF_ID_Instru...n_out[31:0	10485907
w_PC_Add[31:0]	Z
IF_ID_PC_out[31:0]	4
IF_ID_pc_temp_out[31:0	0
w_PC_Add_temp[31:0]	8
w_Write_en	1
w_ALUSrc[1:0]	1
w_Data_out2[31:0]	0
w_imm[31:0]	10
w_Data_out1[31:0]	0
w_Mux_Data[31:0]	0
w_ALUOp[1:0]	3
w_ALUCtrl[5:0]	Z
w_Read_Data[31:0]	0
w_sel_MemtoReg[1:0]	0
w_MemRead	0

Figure 5.3: Behavioural model of Single Cycle Pipelined Processor

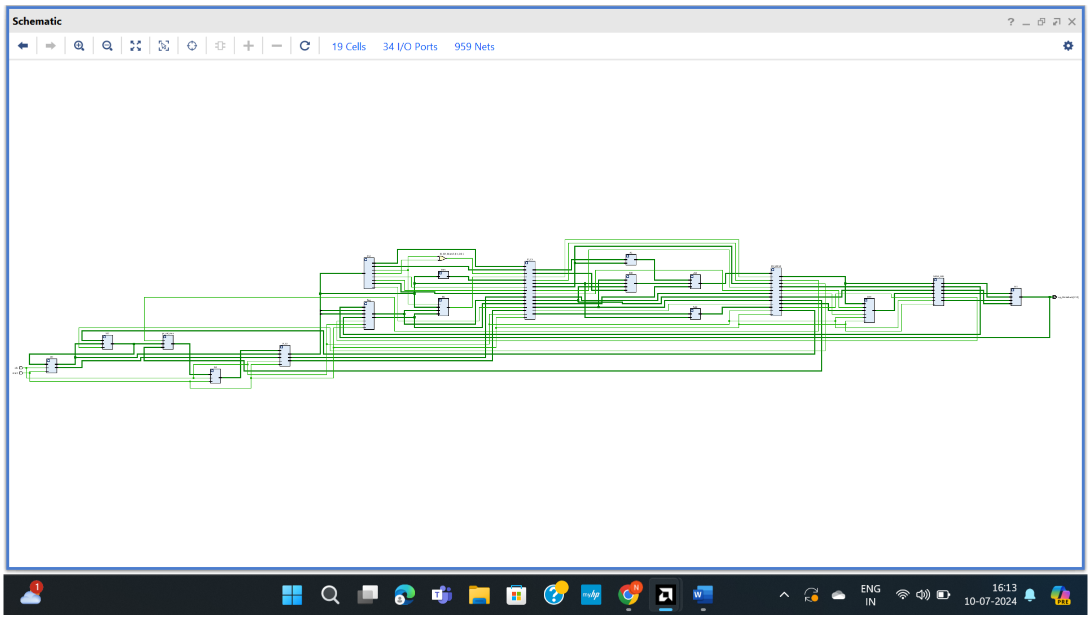
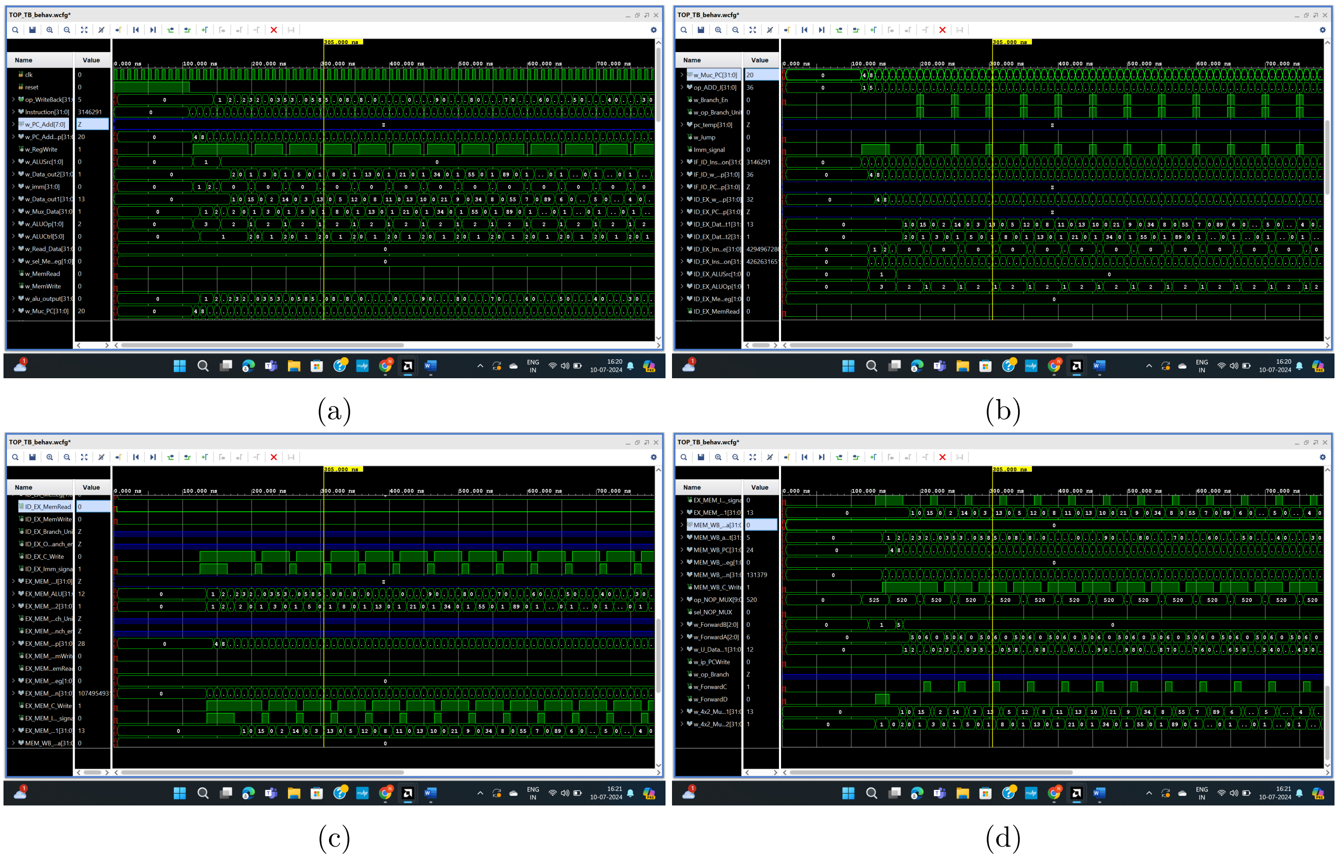


Figure 5.4: RTL Design for Single Cycle Pipelined Processor



(a) (b) (c) (d)

Figure 5.5: Behavioural model of Handling Hazards

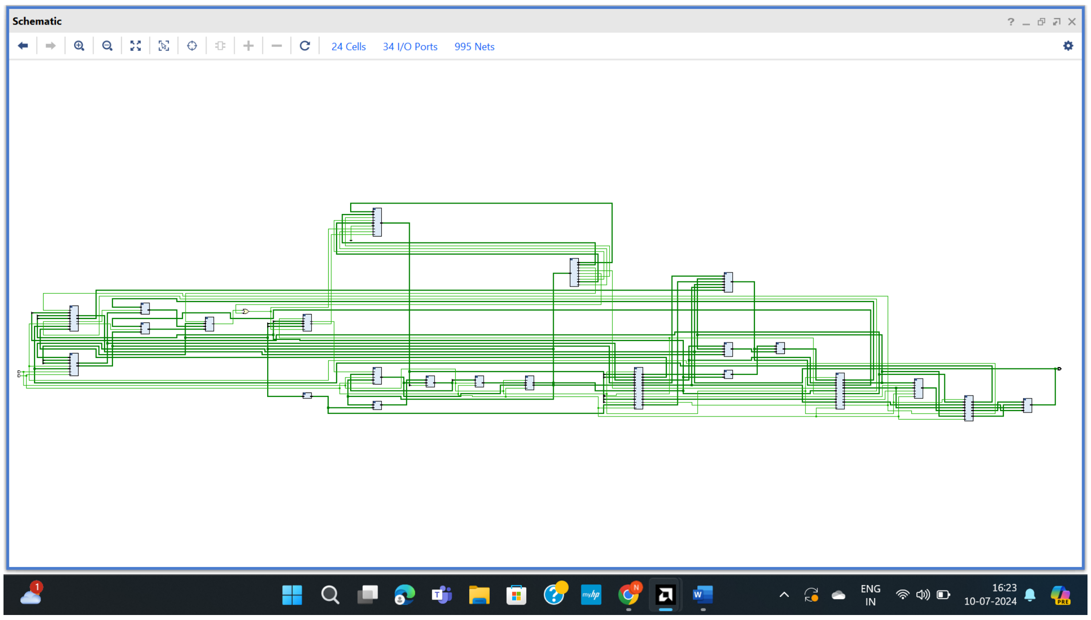


Figure 5.6: RTL Design for Handling Hazards

Chapter 6

Conclusion

Over the past eight weeks, I have had the opportunity to engage deeply with the design and development of the RV32IM processor. This internship has been an immensely valuable experience, providing me with both practical and theoretical knowledge in the field of microprocessor design.

During the initial weeks, I focused on understanding the architecture and specifications of the RV32IM processor, followed by the implementation of various modules including the pipelined stages, hazard detection, and forwarding mechanisms. The process of integrating these components into a cohesive system was both challenging and rewarding, enhancing my problem-solving and critical-thinking skills.

In the latter part of the internship, I concentrated on testing and debugging the processor to ensure its functionality and efficiency. This phase was crucial in identifying and rectifying potential issues, thereby reinforcing the importance of thorough validation in hardware design.

Overall, this internship has significantly broadened my understanding of processor architecture and design. It has also provided me with hands-on experience in using industry-standard tools and methodologies. The guidance and support from my supervisor, mentor, and colleagues were instrumental in my progress, and I am grateful for their contributions.

Looking ahead, I am excited to apply the knowledge and skills gained from this internship to future projects. This experience has not only deepened my interest in microprocessor design but also inspired me to pursue further studies and research in this area.

Chapter 7

Future Scope

The RV32IM processor project has laid a strong foundation for further research and development. Several potential areas for future work have been identified:

Power Efficiency Improvements: Power consumption is a critical factor in processor design, especially for embedded systems. Research can be directed towards developing low-power techniques and implementing power management features. Dynamic voltage and frequency scaling (DVFS) and clock gating are potential areas for exploration.

Integration of Advanced Features: The inclusion of advanced features such as vector processing, hardware accelerators, and enhanced security mechanisms can be considered. These features would expand the applicability of the processor in various domains, including artificial intelligence, machine learning, and cryptography.

In conclusion, the RV32IM processor project has significant potential for future advancements. Continued research and development in the areas mentioned above will contribute to the evolution of this processor, making it more powerful, efficient, and versatile.

References

- FE310-G002 manual
- E31- RISC-V Core IP manual
- TileLink specifications
- The RISC-V Instruction Set Manual v2.2
- RISC-V Assembly Language Programming (Draft v0.18.3-0-g8a08bae) - John Winans
- A 32-bit Integer Division Algorithm Based on Priority Encoder - Firas Hassan, Ahmed Ammar, and Hayden Drennen
- Dadda Multiplier - Dogo Rangsang Research Journal