ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

DESIGN AND IMPLEMENTATION OF HIGH-PERFORMANCE DUAL ISSUE RISC-V CORE

SENIOR DESIGN PROJECT
INTERIM REPORT

Abdullah Aykut KILIÇ
Mustafa Kerem ERTEM

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

May, 2024

ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

DESIGN AND IMPLEMENTATION OF HIGH-PERFORMANCE DUAL ISSUE RISC-V CORE

SENIOR DESIGN PROJECT
INTERIM REPORT

Abdullah Aykut KILIÇ
(040190207)

Mustafa Kerem ERTEM
(040200218)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

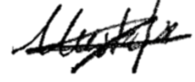
Project Advisor: Prof. Dr. Sıddıka Berna ÖRS YALÇIN

May, 2024

We are submitting the Senior Design Project Interim Report entitled as “Design and Implementation of High-Performance Dual Issue RISC-V Core”. The Senior Design Project Interim Report has been prepared as to fulfill the relevant regulations of the Electronics and Communication Engineering Department of Istanbul Technical University. We hereby confirm that we have realized all stages of the Senior Design Project Interim Report by ourselves, and we have abided by the ethical rules with respect to academic and professional integrity .

Abdullah Aykut KILIÇ
(040190207)

Mustafa Kerem ERTEM
(040200218)



FOREWORD

We offer our sincere gratitude to our academic advisor Prof. Dr. Sıddıka Berna ÖRS YALÇIN, who has been with us with her goodwill and vast knowledge throughout the whole process, besides, we are appreciated to Yasin YILMAZ who is the designer of the Hornet core and who always helped us whenever we needed.

We are thankful to our precious families and friends who always support us both in life and education.

May 2024

Abdullah Aykut KILIÇ
Mustafa Kerem ERTEM

TABLE OF CONTENTS

Page

FOREWORD .... **iv**
TABLE OF CONTENTS .... **vi**
ABBREVIATIONS .... **vii**
SYMBOLS .... **viii**
LIST OF TABLES .... **ix**
LIST OF FIGURES .... **x**
SUMMARY .... **xi**
1. INTRODUCTION **1**
2. PROCESSORS AND CORES **2**
2.1 General Core Design 2
2.1.1 Single-Issue Cores 3
2.1.2 Dual-Issue Cores 3
2.1.2.1 Static Multiple Issue 4
2.1.2.2 Dual-Issue Cores 4
2.2 RISC-V 5
2.3 Literature Review 7
2.4 Hornet Core and Dual issue 9
3. PREPARING THE ENVIRONMENT **9**
3.1 Setting Up the Linux Environment 10
3.2 GNU Toolchain Installation 10
3.3 Compiling and Running a Test Code 10
3.3.1 Simulation and Validation 10
4. DESIGNING THE DUAL-ISSUE CORE **13**
4.1 Extracting the Register File From Hornet 13
4.2 Separation of Data Memory and Instruction Memory 16
4.3 Fetching Instructions 16
4.4 Instruction Memory 17
4.5 Dual Issue Hazards 18
4.6 The Prototype of The Dual Issue Core 18
4.7 The Finalized Design 19
4.7.1 Modification of the Register File 20
4.7.2 Implementation of the Issue Unit 21
4.7.3 Implementation of the PC Logic Unit 23
4.7.4 Implementation of the Dual Hazard Unit 24
5. REALISTIC CONSTRAINTS AND CONCLUSIONS **26**
5.1 Practical Application of this Project 27
5.2 Realistic Constraints 27
5.2.1 Social, Environmental and Economic Impact 27
5.2.2 Cost Analysis 27
5.2.3 Standards 28
5.2.4 Health and safety concerns 28
5.3 Future Work and Recommendations 28
REFERENCES **29**
APPENDICES **30**
Appendix A 31
CURRICULUM VITAE **33**

ABBREVIATIONS

AIC : Akaike Information Criteria
ALU : Arithmetic Logic Unit
CSR : Control-Status Registers
CPU : Central Processing Unit
EX : Execute
FPGA : Field Programmable Gate Arrays
FPU : Floating Point Unit
GNU : Gnu's Not Unix
ID : Instruction Decode
IEEE : The Institute of Electrical and Electronics Engineers
IF : Instruction Fetch
IP : Intellectual Property
ISA : Instruction Set Architecture
ITU : Istanbul Technical University
M-extension : Multiplt Extension
NOP : No-operation
PULP : Parallel Ultra-Low Power
PC : Program Counter
RISC-V : Reduced Instruction Set Architecture
SoC : System On Chip
TRY : Turkish Liras

SYMBOLS

MHz **:** Megahertz

LIST OF TABLES

Page

Table 2.1 : Comparison Between Cortex and RISC-V processors. **8**
Table 4.1 : Instruction Issue Logic Table.. **22**
Table 4.2 : PC Logic Truth Table **24**

LIST OF FIGURES

Page

Figure 2.1 : Architecture Comparison. **2**
Figure 2.2 : Dynamic Multiple Issue Structure. **5**
Figure 2.3 : Comparison Between Needed Number of Cycles to Run Test Programs with Single-Issue and Dual-Issue RISC-V processors. **7**
Figure 2.4 : Comparison Between Power Consumption of Single-Issue and Dual-Issue RISC-V processors. **8**
Figure 3.1 : Bubble Sort Input and Correct Output Values **11**
Figure 3.2 : All Possible Arithmetic and Logic Operations. **12**
Figure 3.3 : Simulation of Bubble Sort Algorithm Execution. **12**
Figure 3.4 : The Test Function for the Hazard Detection **13**
Figure 3.5 : It is seen that the hazard_stall signal is set to 1 on the red line **13**
Figure 4.1 : The Input and Output Port Connections of the Register Bank. **14**
Figure 4.2 : The General Hierarchy **14**
Figure 4.3 : The Completely Separated Register Bank Module **15**
Figure 4.4 : The Bubble Sort Simulation with the Original Core. **15**
Figure 4.5 : The Bubble Sort Simulation with the Modified Core. **15**
Figure 4.6 : RTL Schematic of Combined Instruction Memory and Data Memory. **16**
Figure 4.7 : RTL Schematic of Separated Instruction Memory and Data Memory. **16**
Figure 4.8 : Verilog Code for Separated Instruction Memory. **17**
Figure 4.9 : Block Diagram of The First Prototype **18**
Figure 4.10 : The Block Diagram of the Overall Design **20**
Figure 4.11 : Schematic View of the Initial Register Bank and the Modified Register Bank **21**
Figure 4.12 : Schematic View of the Issue Unit. **23**
Figure 4.13 : The Schematic View of the PC_logic Unit **23**
Figure 4.14 : PC Logic Unit Block Diagram. **24**
Figure 4.15 : The Schematic View of Dual Hazard Unit **25**
Figure 4.16 : Verilog Codes for the Dual Hazard Unit **26**
Figure 4.17: Verilog Codes for the Dual Hazard Unit – Continued **27**
Figure A.1: RTL Schematic of the Top Module in Vivado **31**
Figure A.2: RTL Schematic of the Core_wb Module in Vivado **32**

DESIGN AND IMPLEMENTATION OF HIGH-PERFORMANCE DUAL ISSUE RISC-V CORE

SUMMARY

According to Cheng, the technical marketing director of Qualcomm, RISC-V has gone a significant way since its establishment by University of California, Berkeley in 2010 due to its resilience and cost-saving advantages when it is compared to proprietary ISAs [1]. It is crucial to understand how the RISC-V ISA is important. The main benefits coming from the open-source and simple environment. This has motivated people to design numerous cores and make it publicly available on many platforms. Thus, people have been able to inspire from each other's design and develop further improved cores and SoCs.

The purpose of this paper is to advance the pre-designed single-issue Hornet core by Istanbul Technical University students to a dual-issue architecture and clarify every component of the design. The dual-issue architectures mainly benefit from approximately double instruction throughput. This design approach may confuse the designers in terms of potential hazards which may occur during the operation of the core. In this paper, we strived to find intelligent solutions for complicated structure of the dual-issue architecture.

The general methodology which has been followed throughout this research consists of three main parts: Setting-up the RISC-V toolchain environment and running the existing design, understanding it and developing the new architecture. There are both pros and cons of improving an existing design. The advantageous part is to simply focus on what can be further enhanced; however, the unfavorable part is to strive to make sense of the components of the existing design. Therefore, it may take a long time and may necessitate consulting to the designers. Although, it has been more straightforward the stage of dealing with the existing design compared to the stage of developing the dual-issue architecture. The design of dual-issue primarily requires issuing two instructions at every clock cycle to the pipelines. This job entails a tedious brainstorming to figure out the details and complications which may occur. Therefore, in this study, the design process has been divided into stages and first, has been started from the overall picture then gone into details.

In essence, RISC-V carries significant importance in terms of its flexibility and low-cost nature. This allows many organizations to develop their own cores and SoCs in order to realize their projects. In this paper, it is aimed to advance a previous work to have better performance and higher instruction throughput. While doing this, the dual-issue architecture has been selected as the main attitude. This is because, it had been turned out to be an optimum solution when the many research papers had been analyzed where the performance improvements are broadly close to %80 percent [2].

ÇİFT İŞLEMLİ RISC-V ÇEKİRDEĞİNİN TASARIMI VE İMPLEMENTASYONU

ÖZET

2010 yılında Kaliforniya Üniversitesi, Berkeley tarafından geliştirilen RISC-V mimarisi Qualcomm'un teknik pazarlama direktörü Cheng'e göre, özel ISA'lara kıyasla sahip olduğu dayanıklılık ve maliyet avantajları sayesinde kuruluşundan bu yana kayda değer bir yol kat etmiştir [1]. RISC-V ISA'nın önemini kavramak kritik öneme sahiptir. Bu önemin temel faydaları açık kaynaklı ve kolay kullanılabilir yapısından gelmektedir. Bu sayede birçok çekirdek tasarımı yapılmış ve bu tasarımlar çeşitli platformlarda herkese açık hale getirilmiştir. Böylelikle tasarımcılar birbirlerinden ilham alarak daha gelişmiş çekirdekler ve SoC'ler üretebilmişlerdir.

Bu çalışmasının amacı, İstanbul Teknik Üniversitesi öğrencilerinin tasarladığı Hornet çekirdeğini çift işlemli bir mimariye yükseltmek ve tasarımın her bir bileşenini açıklığa kavuşturmaktır. Çift işlemli çekirdek mimarileri, birim zamanda yaklaşık olarak iki kat daha fazla sayıda işlem gerçekleştirebilmesiyle öne çıkmaktadır. Ancak bu tasarım yaklaşımı, çekirdeğin çalışma esnasında ortaya çıkabilecek potansiyel problemler açısından tasarımcıları zorlayabilmektedir. Bu çalışmada, çift işlemli çekirdek mimarisinin karmaşık yapısı için verimli çözümler üretilmeye çalışılmıştır.

Araştırma boyunca izlenen genel metodoloji üç ana bölümden oluşmaktadır: RISC-V toolchain'in kurulması ve mevcut tasarımın çalıştırılması, mevcut tasarımın anlaşılması ve yeni mimarinin geliştirilmesi. Halihazırda varolan bir tasarımı geliştirmenin hem avantajları hem de dezavantajları vardır. Olumlu yanı, tasarımın temel kısmından ziyade üzerine koyulabiliecek gelişmiş özelliklere odaklanabilmektir. Dezavantajı ise, bir başkası tarafından tasarlanmış komplike bir sistemi anlamaya çalışmaktır. Bu yönüyle yoğun bir çalışma gerektirebilir ve tasarımcılarla görüşüp onların bakış açısını anlamak gerekmektedir. Varolan tasarımı anlamdırma aşaması, çift işlemli mimariyi geliştirme aşamasına kıyasla daha kolay olsa da; çift işlemli tasarımın asıl amacı her saat döngüsünde işlem hatlarına iki komut göndermektir. Bu özellik, ortaya çıkabilecek problemli ve karmaşık durumları çözmek için detaylı bir inceleme gerektirmektedir, dolayısıyla bu çalışmada tasarım süreci aşamalara ayrılmış olup, öncelikle genel resimden başlanarak daha sonra detaylara inilmiştir.

Özet olarak, RISC-V mimarisi esnekliği ve düşük maliyeti açısından önemli bir yere sahiptir. Bu sayede birçok kuruluş kendi çekirdeklerini ve SoC'lerini geliştirebilmektedir. Bu çalışmada, öncesinde İstanbul Teknik Üniversitesi öğrencileri tarafından geliştirilmiş olan Hornet çekirdeğini daha iyi performans ve daha yüksek işlem verimi elde edecek şekilde geliştirmek hedeflenmektedir. Bu amaçla, yapılan literatür taramasında performans artışının genellikle %80'e yakın olduğu görüldüğünden çift işlemli çekirdek mimarisinin geliştirilmesi kararlaştırılmıştır [2].

1. INTRODUCTION

In the rapidly evolving environment of embedded systems and high-performance computing, the demand for efficient and powerful processor architectures is continuously growing. Traditional single-issue processors face limitations in meeting the increasing computational requirements of modern applications like artificial intelligence, image processing etc. The RISC-V instruction set architecture provides an excellent solution for processor design. But unfortunately, there is not enough high-performance design for RISC-V architecture.

The main objective of this project is to improve the performance of the Hornet core developed by ITU students. This performance increase is aimed to be achieve by converting the current single issue core into a dual issue structure. First of all, various performance tests were performed to observe that the core, which is the only issue, is working properly. Then, the structure of this single issue core was analysed and a new architecture was developed to make this core as dual issue core with higher performance. This new architecture has been finalised through various revisions and has been applied to the current core with various changes in the code. Basically, a dual-core architecture with more performance than the existing single-core architecture has been created.

As a result of this project, a dual-issue core with RISC-V architecture, which has not many equivalents both in the market and in the academic field, will be obtained. The higher performance of this core, which can fetch two instructions at once, will enable it to come to the forefront in applications that require more processing power but not so complicated applications such as image processing, artificial intelligence and deep learning in basic manner. In addition, the fact that this core has an open source architecture such as RISC-V will make the design more accessible to all parties.

2. PROCESSORS AND CORES

2.1 General Core Design

The processors in use today are based on two different architectures, Neumann and Hardvard. The main difference between these two architectures lies in their access to memory. In the Harvard architecture, memory is divided into two for data and instrumentation, while in the Neumann architecture, memory is shared. Data and instruction are carried at the same time. The remaining parts are common for both architectures. These parts are basically arithmetic and logic unit (ALU), control unit, programme counter, registers and memory.

The ALU is responsible for performing basic operations. These operations consist of arithmetic operations such as addition and subtraction and logic operations such as and or. Registers are memory elements that store the necessary information while the ALU is running. The control unit controls the rest of the processor by generating various signals during operations. Lastly, memory stores the necessary registers, data and instructions [3].

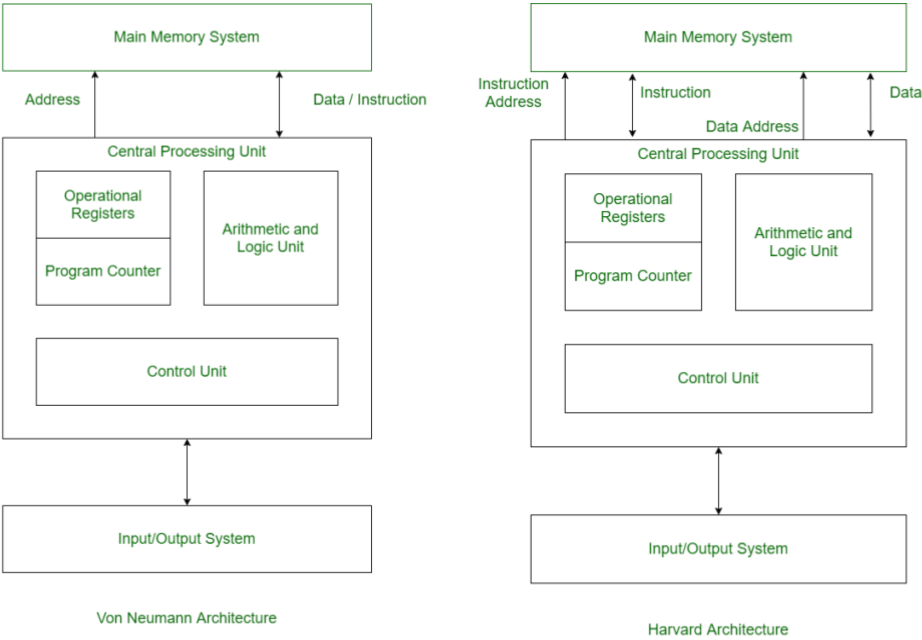


Figure 2.1 : Architecture Comparison [4]

Cores are classified according to the number of instructions they can execute in a single cloclk cycle. These are single issue and multiple issue cores. Multiple issue cores are specially called Dual issue if they can execute two instructions in a single cycle.

2.1.1 Single-issue cores

A microprocessor type known as a single-issue processor is limited to executing a single instruction each clock cycle. The ability of the processor to send instructions to the execution units for processing is referred to as "issue" in the context of computer architecture. A single instruction can only be executed per a clock cycle in a single-issue processor; multiple instructions cannot be executed at the same time by the processor. In contrast, several instructions can be issued and carried out concurrently during a single clock cycle in superscalar architectures.

Single-issue processors are easier to design and build because of their simplicity, but they might not be able to process instructions as quickly as more complicated superscalar processors. With instruction-level parallelism, superscalar processors—which can execute many instructions at once—may be able to attain even better performance. They do, however, also typically have more intricate designs and demand more hardware. Single-issue processors are frequently found in embedded systems, simpler computing devices, or applications with lower computational demands that can be met with a simpler design while still achieving the necessary performance.

2.1.2 Dual-issue cores

A microprocessor that has the ability to issue and carry out two instructions each clock cycle is known as a dual-issue processor. The ability of the processor to send instructions to the execution units for processing is referred to as the "issue" in computer architecture. The processor can benefit from instruction-level parallelism, in which two instructions are carried out concurrently throughout each clock cycle, thanks to the dual-issue capabilities. Comparing this to single-issue processors, more instruction throughput and better performance may result.

A dual-issue processor features multiple continuously operating execution units and a pipeline intended to accommodate the simultaneous processing of two instructions. Because this architecture can take use of instruction-level parallelism, it makes better

use of the resources that are available. As a subset of superscalar architecture, dual-issue processors are capable of carrying out multiple instructions at once. Processors with numerous execution units that can issue and execute several instructions each clock cycle are referred to as "superscalar" processors. A particular class of superscalar processors known as dual-issue processors is dedicated to processing two instructions at once.

While dual-issue processors may perform better than single-issue processors due to their additional complexity, more complex design and control logic are also needed. In more complex and high-performance computing systems, when utilizing instruction-level parallelism is essential to improving overall performance, dual-issue processors are frequently found. Multiple issue core can be classified in two different ways: static and dynamic.

2.1.2.1 Static Multiple Issue

In a static multiple issue, the programmer or the compiler must determine which independent instructions can run simultaneously. The processor schedules instructions statically onto a number of pipelines or execution units. To find parallelism, this method necessitates the use of a complex compiler or programmer, which is not always practical or effective. Its capability to use parallelism across a large range of instructions is one advantage; nevertheless, it may be restricted by the compiler's ability to detect parallelism and requires a significant amount of compiler support.

2.1.2.2 Dynamic Multiple Issue

Dynamically recognizing separate instructions at runtime and allocating them to different execution units for parallel execution is known as dynamic multiple issue, or dynamic instruction scheduling. Instead of relying on compiler instructions or human intervention, dynamic multiple issues use hardware techniques to detect parallelism. This is in contrast to static multiple issues. Unlike static multiscalar scheduling, which is based on availability and dependencies, the dispatching of instructions to execution units allows for more adaptable and flexible scheduling. Because dynamic multiple issues can adjust to changes in program behavior as it is being executed, they may be able to achieve higher performance in scenarios when the compiler may not be able to identify all available parallelism. “Static multiple issue” architecture was chosen for this core design. In the design, the first pipeline can execute only integer and branch

instructions. In addition to that, second pipeline can execute only integer and memory instructions. A simple representation of this structure is as in the **Figure 2.2**.

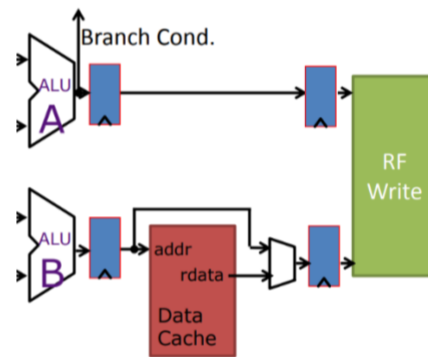


Figure 2.2 : Dynamic Multiple Issue Structure [5]

The CPU receives various commands from the outside world in order to perform the necessary operations, and these commands are called instructions. These instructions have a binary structure for machines to understand and they consist of a certain number of bits. The structure formed by these instructions coming together, containing instruction properties and names, is called Instruction set arctirecture (ISA). Depending on this structure preference, cores based on different architectures can be designed. In RISC-V this is one of the ISAs [6].

2.2 RISC-V

RISC-V is an instruction set architecture (ISA) developed and maintained by RISC-V International (formerly the RISC-V Foundation). It is used to create custom processors for a range of uses, including supercomputers and embedded systems. In contrast to proprietary processor designs, custom processors targeting a range of end applications can be developed using the open-source RISC-V instruction set architecture (ISA). RISC-V is designed around a simple, modular ISA that can be extended with optional features. The base ISA includes a minimal set of instructions for basic computational needs, while additional standard extensions cover floating-point operations, atomic operations, and vector processing, among others. This modularity is a key strength of RISC-V, allowing it to be tailored for a wide range of applications from microcontrollers to high-performance computing. The architecture is also notable for its simplicity and efficiency [7].

The RISC-V ISA, which was first created at the University of California, Berkeley, is regarded as the fifth generation of processors based on the reduced instruction set computer (RISC) architecture. It has grown a lot in popularity recently because of its openness and technical advantages. With over 3,000 members, RISC-V International is currently in charge of overseeing the standard. As of the end of 2022, the organization had shipped over 10 billion chips with RISC-V cores. Many implementations of RISC-V are available, both as open-source cores and as commercial IP products.

For example, the SiFive Freedom Platform, based on RISC-V, offers a family of SoCs that cater to various market needs from low-power to high-performance applications. Additionally, the PULP (Parallel Ultra-Low Power) platform from ETH Zurich and the University of Bologna showcases RISC-V's potential in ultra-low-power computing for IoT applications [8][9]. Also, major tech companies like NVIDIA, Western Digital, and Google have shown interest in RISC-V, indicating its potential to disrupt the processor market.

Because of the architecture's ability to give the processor more straightforward instructions to complete a variety of jobs, RISC-V has grown in popularity. It also speeds up time to market by allowing designers to generate thousands of possible bespoke processors. Additionally, the shared processor IP reduces the time needed for program development. Some advantages of RISC-V are as follows:

- Because of its open-standard design, the industry may collaborate and innovate together.
- Common ISA: Since all processors may utilize the same architecture, this facilitates software development. From small embedded systems to the biggest supercomputers, designers can customize their creations to meet market demands by using the same basic ISA.
- RISC-V ISAs are different from earlier ISAs in that they can be tailored to meet specific needs. The availability of more compact, modular, and energy-efficient solutions
- Open-source reference designs, software composition analysis tools, and security extensions are sources of security features. Furthermore, because the RISC-V architecture is open-source, all of its components can be thoroughly

examined in the public domain, removing any potential for back doors or hidden channels.

2.3 Literature Review

This project is based on the paper "Design and Implementation of a 32-bit RISC-V Core". [10] In this paper, the single-issue RISC-V Core which is designed by ITU students is revealed. It was seen that there could be further improvements on this core. To figure out what could be done, various studies are examined. There are many perspectives including application specific accelerators, dual-core architectures, dual-issue and multiple-issue architectures. Besides, it turned out that there is no previous research about dual-issue RISC-V Core in Turkey.

Thus, different approaches about dual-issue cores are examined. A researcher Hongsheng designed and tested a dual-issue RISC-V processor [2]. In that study, first a single-issue processor is designed then it is extended to a dual-issue processor. The fetched data in Instruction Fetch (IF) stage is doubled and Instruction Decode (ID) stage is modified accordingly. Execution (EX) stage unit is doubled and a data hazard unit is added to prevent complications. The main idea is to use the same modules to extend the single-issue processor to dual-issue core which reduces the design complexity. Consequently, the final design was subjected to performance analysis and it turned out that the dual-issue processor was 80% faster than the single-issue processor (Figure 2.3) while maintaining just a slight difference between power consumption (Figure 2.4) [2].

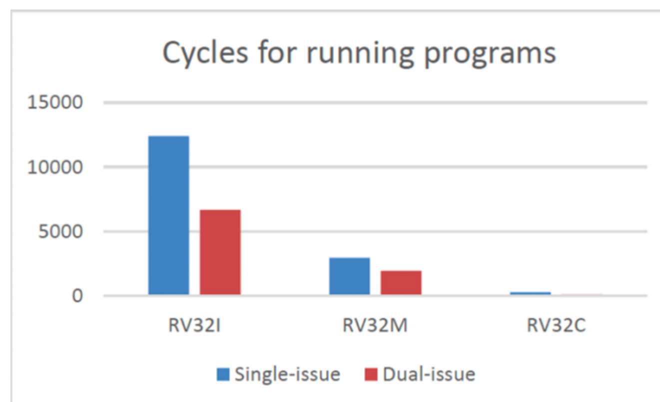


Figure 2.3 : Comparison Between Needed Number of Cycles to Run Test Programs with Single-Issue and Dual-Issue RISC-V processors [2]

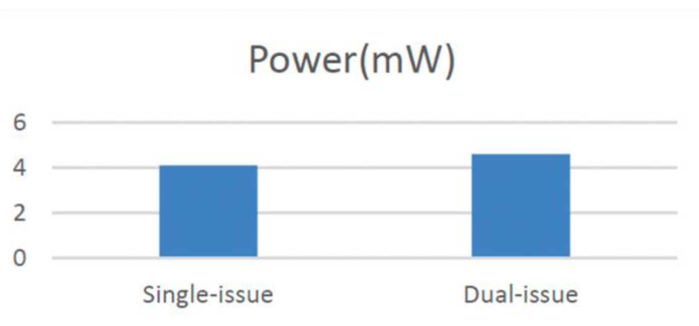


Figure 2.4 : Comparison Between Power Consumption of Single-Issue and Dual-Issue RISC-V processors [2]

A group of researchers from the Indian Institute of Science designed a 32-bit dual pipeline superscalar RISC-V processor on an FPGA. The architecture was proposing fetching two sequential instructions in the IF stage from Instruction Cache (I-Cache). And one or two of the instructions are issued by Instruction Issuing Unit (IIU) depending on dependencies in between. Because the architecture is just capable of executing two integer instructions or one integer and one floating point instructions. Besides the two instruction fetching, branch prediction unit is also implemented. This unit is used in the EX stage in order to predict the next Program Counter (PC) value. Thus, the processor becomes faster. In conclusion, on Virtex-7 FPGA, the processor is tested via CoreMark benchmark and the dual-issue RISC-V processor was 13% faster than Arm Cortex-M4 Processor (Table 2.1) [11].

Processor	CoreMark/MHz
Dual-Issue RISC-V	3.84
Cortex-M4 (NXP Kinetis K70)	3.40
Cortex-M3 (STM32L152)	3.34
Cortex-M0+ (Atmel SAM D20)	2.46
Cortex-M0 (STM32F051C8	2.33

Table 2.1 : Comparison Between Cortex and RISC-V processors [11]

When the mentioned designs are examined it is seen that dual-issue RISC-V processors can provide faster processing compared to well-known processors and they consume less power compared to single-issue RISC-V processors. Also extending single-issue processors to dual-issue processors improves the performance significantly. Thus, these properties convinced us to extend the single-issue Hornet core to dual-issue core.

2.4 Hornet Core and Dual issue

Hornet is a single-issue core that implements the M extension of RISC-V developed by ITU students. This core is designed to run more efficiently in applications requiring less power like microcontroller systems. The core is designed in a modular way to make it easier to add, remove and modify, and all steps in the design of the core are documented. Because of these features, Hornet is a core that is open to be made dual issue. The fact that it has detailed documentation and the designers of the core are more easily accessible minimises the disadvantages and incomprehensions that may arise during the design of the new core. For these reasons, it was decided to design our dual issue core based on the Hornet core. In addition, instead of designing a dual issue core from scratch, it is aimed to dual issue an existing core in the market. The main motivation here is to create a more efficient core with less labour and research time. Considering the real life conditions, it is seen how important this is for product development. As a result, it was decided to make the Hornet core dual issue, which required first understanding the structure of the Hornet core and also running the core smoothly.

3. PREPARING THE ENVIRONMENT

This was basically the first physical step that we have taken in order to develop a further core. Validating the current core and ensuring the installation steps was quite important to create a solid project. The following steps mostly originated from the "Design and Implementation of a 32-bit RISC-V Core" paper [1], however some problems are fixed and documented in our Github [12]. The main stages are setting up

the Linux environment, RISC-V GNU toolchain installation, test code compilation and running, and the last step is simulation.

3.1 Setting up the linux environment

RISC-V GNU Toolchain can only be utilized on a Linux environment, this is why it is used. The Linux operating system is set on a UTM virtual machine which makes it easier to set up the Linux environment. The installation steps can be found in many sources, after setting up the Linux operating system, toolchain should be installed.

3.2 GNU toolchain installation

First and foremost, the main and the additional prerequisites are installed. After cloning the toolchain, it is configured with related prefixes and multilib generators. Then, "make" command is executed. In this step if python cannot be found, the following code can be executed in order to have a compatible version of python and execute "make" again.

- ```
  # Install python 2
  sudo apt install python
  # Make python refer to python3
  sudo apt install python-is-python3
  # Prepare for a clean build
  make distclean
  sudo rm -rf <prefix-dir-specified-at-configure-time>
  # Configure and make
  ./configure --prefix=... --with-multilib-generator="..."
  make 2>&1 | tee build.log
  ```

After toolchain installation is done successfully, its directory should be added to path in order to compile codes.

3.3 Compiling and running a test code

Before running a code it should be compiled. For this purpose ".elf", ".bin" and ".data" files were created respectively. Then a script is executed which utilizes a "c wrapper file" to make the original test code with ".c" extension compatible with the simulator.

3.3.1 Simulation and validation

To simulate the core, Verilator and GTKWave are installed. Verilator is actually used to simulate the core and GTKWave is used to observe the signal waves. To verify the core a test code is written including all the combinations of arithmetic and logic
  ```

operations (Figure 3.2) and it is observed that except from floating point and 38 bit shift operation everything works properly. The shift operation can be done maximum with 32 bits and in order to utilize floating point operations FPU (Floating Point Unit) should be added to the design.

```
1   #include "string.h"
2   #define DEBUG_IF_ADDR 0x00002010
3
4   void bubble_sort(int* arr, int len)
5   {
6       int sort_num;
7       do
8       {
9           sort_num = 0;
10          for(int i=0;i<len-1;i++)
11          {
12              if(*(arr+i) > *(arr+i+1))
13              {
14                  int tmp = *(arr+i);
15                  *(arr+i) = *(arr+i+1);
16                  *(arr+i+1) = tmp;
17                  sort_num++;
18              }
19          }
20      }
21      while(sort_num!=0);
22  }
23
24  int main()
25  {
26      int unsorted_arr[] = {195,14,176,103,54,32,128};
27      int sorted_arr[] = {14,32,54,103,128,176,195};
28      bubble_sort(unsorted_arr,7);
29
30      int *addr_ptr = DEBUG_IF_ADDR;
31      if(memcmp((char*) sorted_arr, (char*) unsorted_arr, 28) == 0)
32      {
33          //success
34          *addr_ptr = 1;
35      }
36      else
37      {
38          //failure
39          *addr_ptr = 0;
40      }
41      return 0;
```

Figure 3.1 : Bubble Sort Input and Correct Output Values

```
r_sum = a + b;
r_sum1 = r_sum + 10;
r_mul = a * b;
r_div = a/b;
r_sum2 = b/d;
r_mul2 = f * d;
r_and = c & d;
r_or = c | d;
r_sftr = c >> 12;
r_sftl = c << 12;
r_sftm = c >> 38;
r_xor = c ^ d;
r_xnor = c ~^ d;
r_neg = !d
```

Figure 3.2 : All Possible Arithmetic and Logic Operations

And also the bubble sort algorithm is executed successfully (Figure 3.3) in the core and the source code can be seen in Figure 3.1. The outputs are given in data_wb_dat_i signal.

Figure 3.3 : Simulation of Bubble Sort Algorithm Execution

Besides, it is tested how the core behaves when there is a dependency between consecutive instructions. To do that the code of the hazard detection module was observed and it is seen that the hazard detection module sets “stall_IF” signal to 1 and the pipeline is stalled at that duration. In the below simulation figure the behavior of the core is simulated when there is a hazard. When the “r_sum” is attempted to be summed with “10” the hazard_stall signal is set to “1” and pipeline is stalled.

```c
void calculator(int a, int b, int c, int d)
{


        r_sum = a + b;
        r_sum1 = r_sum + 10;
        r_mul = c * d;
        r_div = a/b;
        r_sum2 = a + d;
        r_mul2 = a * b * c * d;
}

int main()
{

    calculator(8,4,7,9);
```

Figure 3.4 : The Test Function for the Hazard Detection

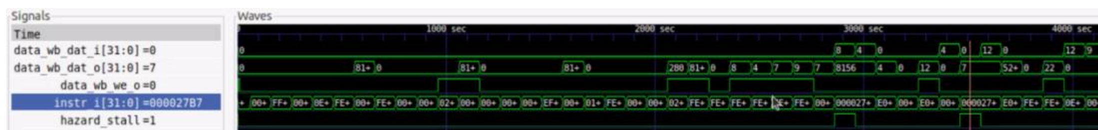


Figure 3.5 : It is seen that the hazard_stall signal is set to 1 on the red line.

4. DESIGNING THE DUAL-ISSUE CORE

4.1 Extracting the Register File From Hornet

The register file is used to assign registers that are used in compiler to physical registers that essentially make the real arithmetic and logical operations. In the previous single-issue core the register file was embedded to core. However, in order to develop a dual-issue processor it is needed to separate the register file from the core and use it as a common block for both pipelines. To realize this design all the code lines related to the register file are isolated from the core and defined into a new module called "register bank" (Figure 4.3). Then all the necessary input and output ports are defined and connected to the core (Figure 4.1). While extracting the register file from the core

all the registers and wires that are defined in the new modules should be defined with the exact same bit number.

```
116  reg_bank REG_BANK( .clk_i(clk_i),
117                    .reset_i (reset_i),
118                    .rf_wen_WB (rf_wen_WB),
119                    .rd_WB (rd_WB),
120                    .mux_o_WB (mux_o_WB),
121                    .take_branch (take_branch),
122                    .csr_id_flush (csr_id_flush),
123                    .stall_EX (stall_EX),
124                    .misaligned_access (misaligned_access),
125                    .IDEX_preg_rs1 (IDEX_preg_rs1),
126                    .IDEX_preg_rs2 (IDEX_preg_rs2),
127                    .stall_ID (stall_ID),
128                    .rs1_ID (rs1_ID),
129                    .rs2_ID (rs2_ID),
130                    .IDEX_preg_data1_w (IDEX_preg_data1),
131                    .IDEX_preg_data2_w (IDEX_preg_data2) );
132
```

Figure 4.1 : The Input and Output Port Connections of the Register Bank

The general hierarchy is aimed to be separate instruction memory, data memory and register bank. Those three common modules are connected to both pipelines. In the Figure 4.2 a single pipeline (core.v) and a separate register bank (reg_bank) are given and their instances are created then connected with each other under “core_wb.v”.

Figure 4.2 : The General Hierarchy

```verilog
module reg_bank(input clk_i,
                input reset_i,
                input rf_wen_WB,
                input [4:0] rd_WB,
                input [31:0] mux_o_WB,
                input take_branch,
                input csr_id_flush,
                input stall_EX,
                input misaligned_access,
                input [4:0] IDEX_preg_rs1,
                input [4:0] IDEX_preg_rs2,
                input stall_ID,
                input [4:0] rs1_ID,
                input [4:0] rs2_ID,
                output  [31:0] IDEX_preg_data1_w,
                output  [31:0] IDEX_preg_data2_w);

reg [31:0] IDEX_preg_data2, IDEX_preg_data1;
reg [31:0] register_bank [31:0];

assign   IDEX_preg_data1_w  = IDEX_preg_data1;
assign   IDEX_preg_data2_w  = IDEX_preg_data2;

//write to register file
integer i;
always @(negedge clk_i or negedge reset_i)
begin
    if(!reset_i)
    begin
        for(i=0; i < 32; i = i+1)
            register_bank[i] <= 32'b0; //reset all
    end

    else if(!rf_wen_WB)
        register_bank[rd_WB] <= mux_o_WB;
end

always @(posedge clk_i or negedge reset_i)
begin
    if(!reset_i)
    begin
        {IDEX_preg_data1, IDEX_preg_data2} <= 64'b0;
    end

    else if(take_branch || csr_id_flush) //flush the pipe
    begin
        {IDEX_preg_data1, IDEX_preg_data2} <= 64'b0;
    end

    else if(stall_EX || misaligned_access)
    begin
        if(IDEX_preg_rs1 == 5'b0)
            IDEX_preg_data1 <= 32'b0;
        else
            IDEX_preg_data1 <= register_bank[IDEX_preg_rs1];

        if(IDEX_preg_rs2 == 5'b0)
            IDEX_preg_data2 <= 32'b0;
        else
            IDEX_preg_data2 <= register_bank[IDEX_preg_rs2];
    end

    else
    begin
        if(!stall_ID)
        begin

            if(rs1_ID == 5'b0)
                IDEX_preg_data1 <= 32'b0;
            else
                IDEX_preg_data1 <= register_bank[rs1_ID];

            if(rs2_ID == 5'b0)
                IDEX_preg_data2 <= 32'b0;
            else
                IDEX_preg_data2 <= register_bank[rs2_ID];
        end
    end
end


endmodule
```

Figure 4.3 : The Completely Separated Register Bank Module

The new configuration with extracted register bank is tested with bubble sort algorithm via Vivado simulations and compared with the original core and the result is turned out to be that the modified core is working properly. The simulation with original core and modified core can be seen in Figure 4.4 and Figure 4.5 respectively.

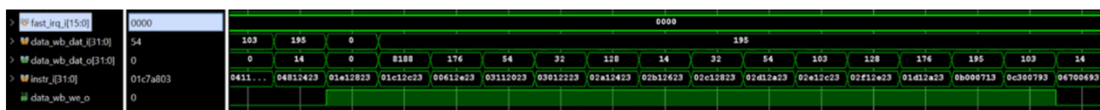


Figure 4.4 : The Bubble Sort Simulation with the Original Core

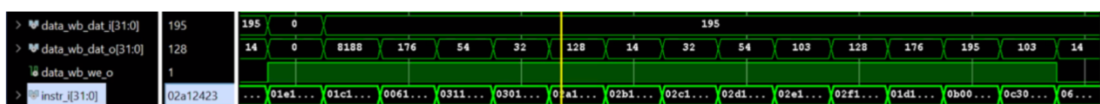


Figure 4.5 : The Bubble Sort Simulation with the Modified Core

4.2 Separation of Data Memory and Instruction Memory

The memory designed for Hornet Core is designed as a single module including both data and instruction memory. Since the new core to be designed will have a dual issue structure, these memories must be separate and can be manipulated independently of each other. Therefore, data memory and instruction memory, which are normally in the same module, have been redesigned as separate modules. In the Figure 4.7, it can be seen that the new instruction memory and data memory. The Figure 4.6 shows the memory in the combined state.

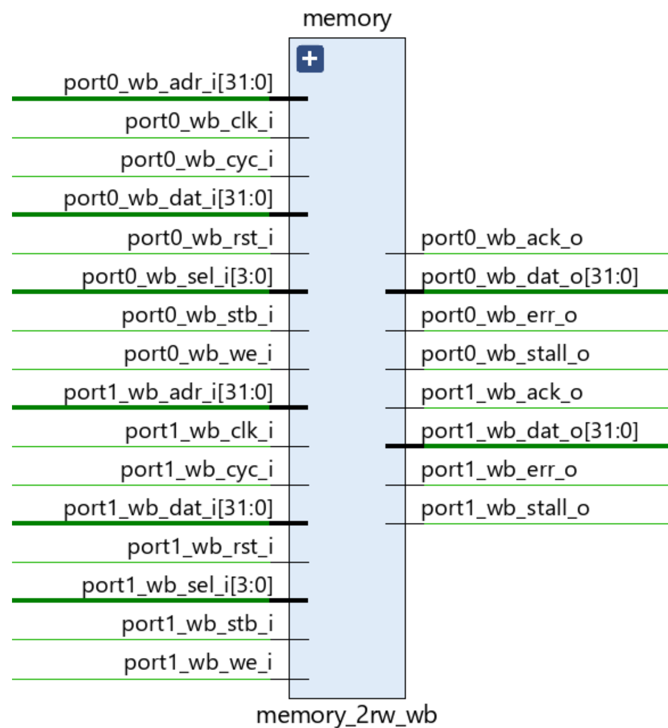


Figure 4.6 : RTL Schematic of Combined Instruction Memory and Data Memory

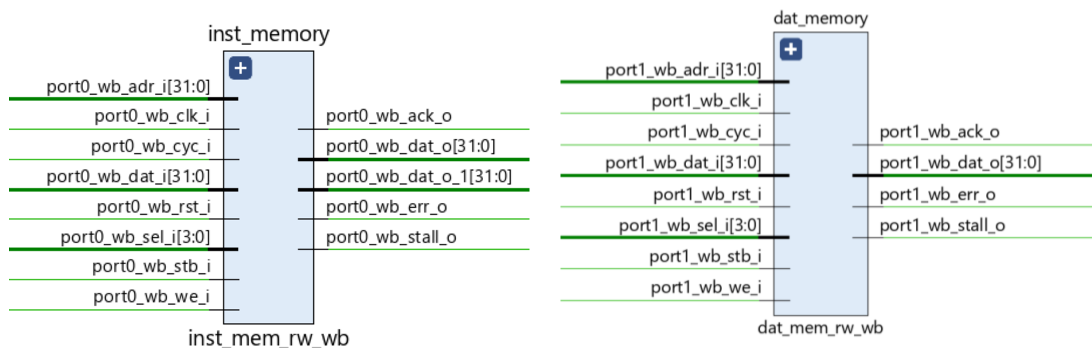


Figure 4.7 : RTL Schematic of Separated Instruction Memory and Data Memory

4.3 Fetching Instructions

In the stage of developing the initial single-issue processor to create a dual-issue processor, it had first planned to fetch instructions one by one to each core in each clock cycle then fetching two instructions simultaneously to both pipes in order to make the developing procedure easier. However, this step is skipped because it turned out to be logically more sensible to transform single port instruction memory to dual port instruction memory and fetch two instructions simultaneously. The process of modifying instruction memory is given in the next paragraph.

4.4 Instruction Memory

Instruction memory has been redesigned for the new core with dual issue core structure. The design, which previously could only send one instruction output in one clock cycle, has been made capable of sending two instructions in one clock cycle. Each instruction sends an instruction to a pipeline. Thanks to this change, the instructions sent in a single clock cycle are executed simultaneously.

```
1  module inst_mem_rw_wb (
2  input           port0_wb_cyc_i,
3  input           port0_wb_stb_i,
4  input           port0_wb_we_i,
5  input [31:0]    port0_wb_adr_i,
6  input [31:0]    port0_wb_dat_i,
7  input [3:0]     port0_wb_sel_i,
8  output          port0_wb_stall_o,
9  output          port0_wb_ack_o,
10 output reg [31:0] port0_wb_dat_o,
11 output reg [31:0] port0_wb_dat_o_1,
12 output          port0_wb_err_o,
13 input           port0_wb_rst_i,
14 input           port0_wb_clk_i);
15
16
17 parameter NUM_WMASKS = 4 ;
18 parameter DATA_WIDTH = 32 ;
19 parameter ADDR_WIDTH = 9 ;
20 parameter RAM_DEPTH = 1 << ADDR_WIDTH;
21
22 wire clk0; // clock
23 wire cs0; // active low chip select
24 wire we0; // active low write control
25 wire [NUM_WMASKS-1:0] wmask0; // write mask
26 wire [ADDR_WIDTH-1:0] addr0;
27 wire [DATA_WIDTH-1:0] din0;
28 wire [DATA_WIDTH-1:0] dout0;
29
30
31 assign clk0 = port0_wb_clk_i;
32 assign cs0 = ~port0_wb_stb_i;
33 assign we0 = ~port0_wb_we_i;
34 assign wmask0 = port0_wb_sel_i;
35 assign addr0 = port0_wb_adr_i[ADDR_WIDTH+1 : 2];
36 assign din0 = port0_wb_dat_i;
37 assign port0_wb_stall_o = 1'b0;
38 reg port0_ack;
39 always @(posedge port0_wb_clk_i or posedge port0_
40 begin
41     if(port0_wb_rst_i)
42         port0_ack <= 1'b0;
43     else if(port0_wb_cyc_i)
44         port0_ack <= port0_wb_stb_i;
45 end
46 assign port0_wb_ack_o = port0_ack;
47 assign port0_wb_err_o = 1'b0;
48
49 reg [DATA_WIDTH-1:0] mem [0:RAM_DEPTH-1] /*verilator
50
51 `ifdef FPGA_READMEM
52 initial $readmemh("reset_handler.mem",mem,7424,7487);
53 initial $readmemh("bootloader.mem",mem,7488,8191);
54 `endif
55
56   // Memory Write Block Port 0
57   // Write Operation : When we0 = 0, cs0 = 0
58 always @ (posedge clk0)
59 begin
60     if ( !cs0 && !we0 )
61     begin
62         if (wmask0[0])
63             mem[addr0][7:0] = din0[7:0];
64         if (wmask0[1])
65             mem[addr0][15:8] = din0[15:8];
66         if (wmask0[2])
67             mem[addr0][23:16] = din0[23:16];
68         if (wmask0[3])
69             mem[addr0][31:24] = din0[31:24];
70     end
71 end
72
73   // Memory Read Block Port 0
74   // Read Operation : When we0 = 1, cs0 = 0
75 always @ (posedge clk0)
76 begin
77     if (!cs0 && we0)
78         port0_wb_dat_o <= mem[addr0];
79         port0_wb_dat_o_1 <= mem[addr0 + 1];
80 end
81
82 endmodule
```

Figure 4.8 : Verilog Code for Separated Instruction Memory

4.5 Dual Issue Hazards

The most compelling part of designing a dual-issue processor is undoubtedly the presence of hazards. Hazards can be divided into two: Structural hazards and data hazards. The case in which the two instructions cannot be executed simultaneously can be named as structural hazard where the data hazard implies the data dependency between two consecutive instructions means that the result of current instruction is dependent on the result of prior instruction [13]. Resolution of hazards comprises two main parts: Detection of the hazard and stall of the pipeline. Detection is done by hazard units which check the source registers of the current instruction and the destination register of the previous instruction and the functional types of the instructions as well. On the other hand, stall operation is simply implies feeding the same PC to the processor and placing no-operation instruction to the pipeline stages. That basically completes the hazard algorithm.

4.6 The Prototype of The Dual Issue Core

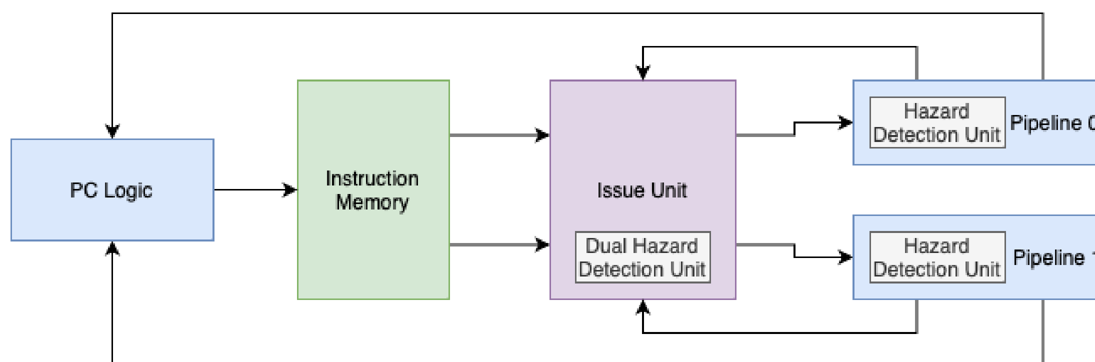


Figure 4.9 : Block Diagram of The First Prototype

After separating instruction memory and data memory, the output port of instruction memory is doubled. This was the first step to fetch two instructions at a time to the core. However, this little step necessitates a very-well planned design. In order to make the core simpler and tidier it is decided that one of the pipelines will execute only branch and ALU operations, on the other side the second pipeline will just execute memory and ALU operations. Thus, the first design component is that this architecture necessitates an issue unit that forwards incoming instructions to the pipelines according to their instruction types.

In a regular single pipeline processor, there is just a single hazard detection unit that detects hazards such as structural hazards, data hazards etc. between sequential instructions that are fetched to the pipeline. However, in the dual-issue case, there are two pipelines which are operating simultaneously. That phenomena generates a second type of hazard that is the dependencies or the structural hazards between instructions that are fetched simultaneously to the pipelines. For this reason, a second type of hazard unit which is responsible for detecting the data dependencies or structural hazards between the parallel instructions that are going to be fetched to the pipelines is created in the issue unit.

In order to update the PC regularly, the PC logic unit is designed. This unit is in charge of updating PCs depending on the hazard signals which are generated from the hazard detection units within the pipeline and also in the issue unit as well. The PC is desired to be increased by eight regularly, because in each clock cycle, two instructions are needed to be read. However, in case of hazard, the PC may need to be partially or fully stalled. That is to say, if just one of the hazard units sends a signal to the PC logic unit, then the PC needs to be partially stalled which means it needs to be increased by four. However, if there is a hazard in two of the hazard units then the PC should be stalled completely.

Furthermore, the dual hazard unit is desired to detect hazards between parallel instructions before they are fetched, and update PC according to the hazard situations. The problem with this design is timing. In the base core design, hazard detection is made on the decode stage, the decoded instruction information is sent to the hazard unit then it decides whether there is a hazard or not and according to that, it stalls the pipe and PC. On the contrary, the proposed design comprises a third hazard unit in issue unit, in other words, just before the fetch operation. Consequently, it is decided that there is a timing issue and all of the hazard units should be in the same stage, in order to control the PC properly.

4.7 The Finalized Design

The major part of designing a dual-issue core based on a single-issue core is the instruction issuing and hazard handling. Fetching two instructions at a time may cause numerous problems. Thus, the design has been made considering those cases. In this

section, mostly the instruction and hazard signal flow are discussed. In this perspective, there are three essential units: Instruction memory, PC logic unit, issue unit and hazard units.

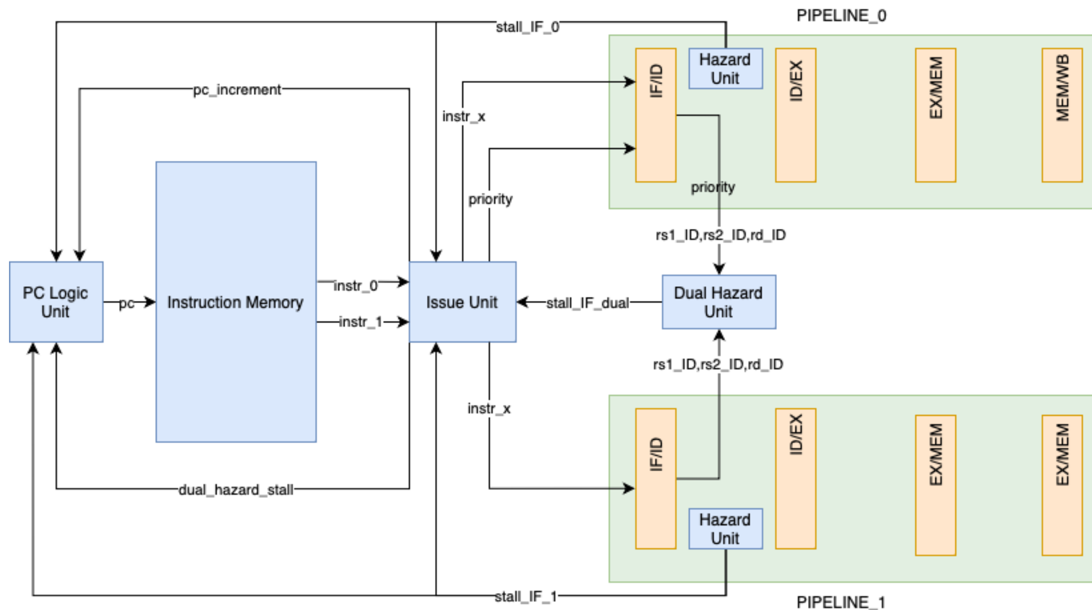


Figure 4.10 : The Block Diagram of the Overall Design

To begin with the instruction memory, it outputs two instructions at a single clock cycle to the issue unit. Issue unit checks the instructions' types and the current hazard situations in order to fetch the instructions to each pipeline since one of the pipelines executes memory and ALU operations while the other one executes branch and ALU operations. Moreover, it sends a "pc_increment" signal to the PC logic unit in order it to make a decision on the amount of pc increment. The fetched instructions are decoded in each pipeline, and thereafter hazard checks are made by two regular hazard units and one dual hazard unit. The generated stall signals from all three units are sent to the issue unit and the PC logic unit, in order to control the instruction flow. Eventually, the PC logic unit determines the next PC value according to the stall, branch, CSR and PC increment signals. This PC is plugged into the instruction memory for the next instructions to be obtained. This structure can be seen in Figure 4.10. Besides, RTL schematics are given in Figure A.1 and Figure A.2.

4.7.1 Modification of the Register File

Register file has 32-bit 32 registers. In the initial case for the Hornet core there was just a single pipeline which utilizes the register file from two read and one write port. However, the developed dual-issue core necessitates two separate access to the register file from each pipeline since both pipelines have ability to handle register operations. In order to satisfy this condition the register file has been modified such that there are four read ports and two write ports. Besides, the internal structure, other control and status signals are modified according to the dual-issue structure.

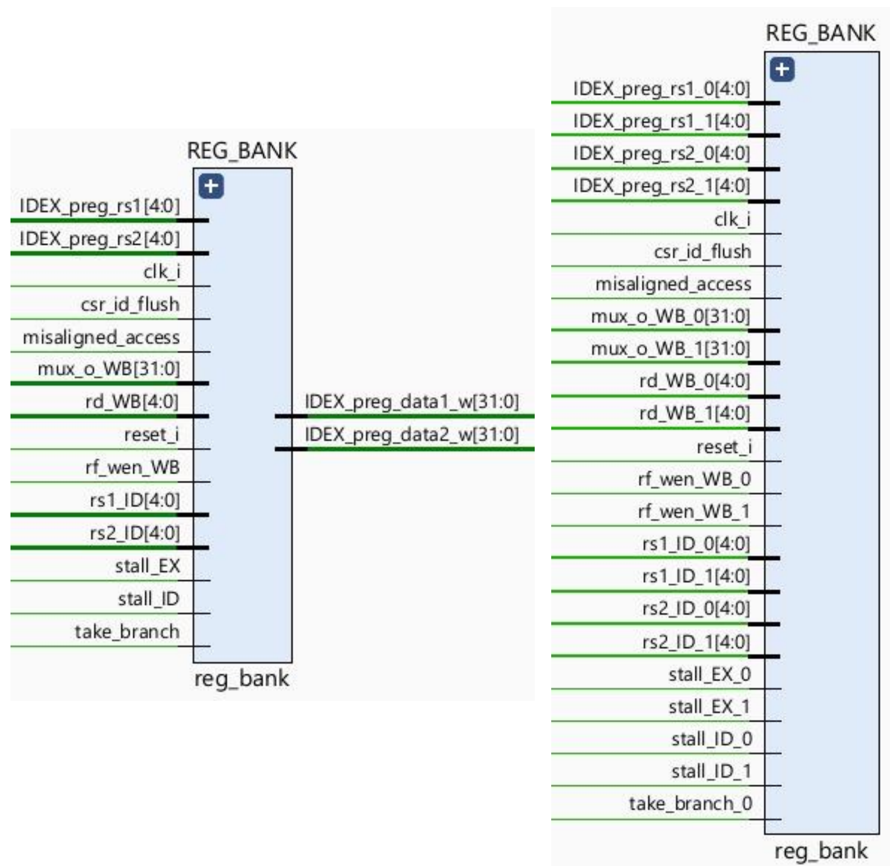


Figure 4.11 : Schematic View of the Initial Register Bank and the Modified Register Bank

4.7.2 Implementation of the Issue Unit

The issue unit simply covers the functionality of issuing each instruction to appropriate pipelines according to their opcodes. This module has two main parts: First is to decode the instructions and assess the instruction type depending on the opcodes. Second is to evaluate two consecutive instructions and determine whether both of them or just the first one or none of them are going to be sent according to the Table 4.1. In

addition, the issue unit also determines which instruction is going to which pipeline since the first pipeline executes branch and ALU operations while the second pipeline executes memory and ALU operations. Besides, it is important to emphasize that no second instruction is issued if the first instruction is a branch operation. Because if the condition of the branch operation is satisfied then jump operation would be held and the executed second instruction would be a hazard.

Incoming Instructions		**Issued Instructions**		**Status Signals**
1. Instruction Type	2. Instruction Type	1. Pipeline	2. Pipeline	
Memory-1	Memory-2	NOP	Memory-1	issue_stall_0 = 1, issue_stall_1 = 0 priority = 1, pc_increment = 4
Memory	Branch	Branch	Mem	issue_stall_0 = 0, issue_stall_1 = 0 priority = 1, pc_increment = 8
Branch	Mem	Branch	NOP	issue_stall_0 = 0, issue_stall_1 = 1 priority = 0, pc_increment = 4
Mem	Alu	Alu	Mem	issue_stall_0 = 0, issue_stall_1 = 0 priority = 1, pc_increment = 8
Alu	Mem	Alu	Mem	issue_stall_0 = 0, issue_stall_1 = 0 priority = 0, pc_increment = 8
Alu	Branch	Branch	Alu	issue_stall_0 = 0, issue_stall_1 = 0 priority = 1, pc_increment = 8
Branch	Alu	Branch	NOP	issue_stall_0 = 0, issue_stall_1 = 1 priority = 0, pc_increment = 4
Branch-1	Branch-2	Branch-1	NOP	issue_stall_0 = 0, issue_stall_1 = 1 priority = 0, pc_increment = 4
Alu-1	Alu-2	Alu-1	Alu-2	issue_stall_0 = 0, issue_stall_1 = 0 priority = 0, pc_increment = 8

Table 4.1 : Instruction Issue Logic Table

The issue unit checks some status signals and generates some status signals as well. Firstly, the stall signals which are generated from the hazard units are checked in order to decide whether to forward instructions or not. If there is a hazard signal then the issue unit just stalls the instruction to the next clock cycle. If there is no hazard signal then it runs the given logic in the Table 4.1. The status signals are "issue_stall, pc_increment, priority". The first, 1-bit "issue_stall" signal is utilized in the pipeline in order to decide stalling the pipeline and this signal is generated if the pipeline is expected not to execute any instruction, in other words to execute no-operation (NOP). The second, 32-bit "pc_increment" signal is utilized in the "pc_logic" unit in order to decide how much the PC needs to be increased depending on whether there is a stalled operation or not. And the third, 1-bit "priority" signal is designed for the "dual hazard unit". This signal specifies which pipeline is fetched by the primary instruction. The

functionality of the “priority” signal is going to be further detailed in the “implementation of the dual hazard unit” section.

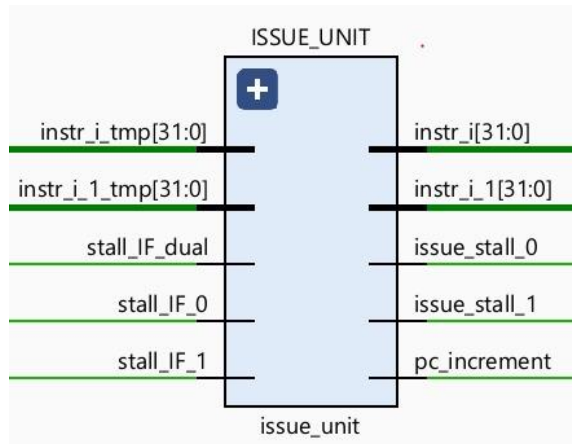


Figure 4.12 : Schematic View of the Issue Unit

4.7.3 Implementation of the PC Logic Unit

In the original single-issue design the PC used to be assigned in the same module with the rest of the core. However, in the dual-issue design there are two separate modules for the pipeline stages and the PC is assigned under another module named “PC_logic”. The assigned PC goes through a couple of modules then arrives at the pipelines as it is mentioned in the finalized design.

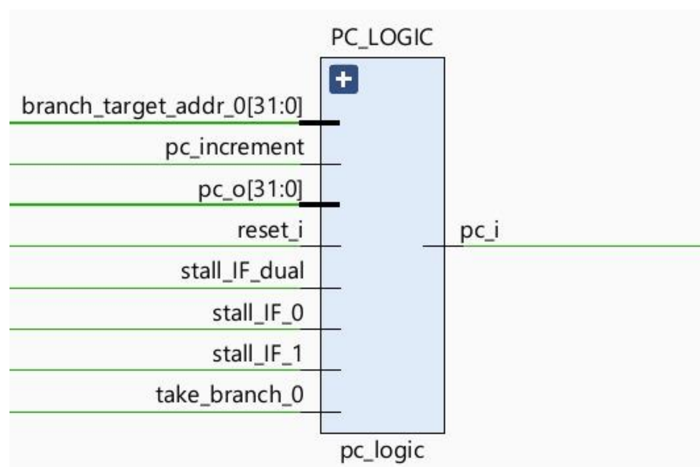


Figure 4.13 : The Schematic View of the PC_logic Unit

This unit mainly calculates the next value of the PC considering several factors such as reset, branch and stall signals. The stall signals are generated from three separate

hazard units. Two of them are from within pipelines and the third one is from the dual hazard unit. If any one of the stall signals is generated then the PC is directly stalled, otherwise increased by “pc_increment” value from the issue unit. The PC logic unit might be able to handle jump operations as well. Thus, “take_branch” signal controls whether to jump to “take_branch_addr” or just use the new PC value. There are also CSR and reset controls which are compulsory for the core to work properly.

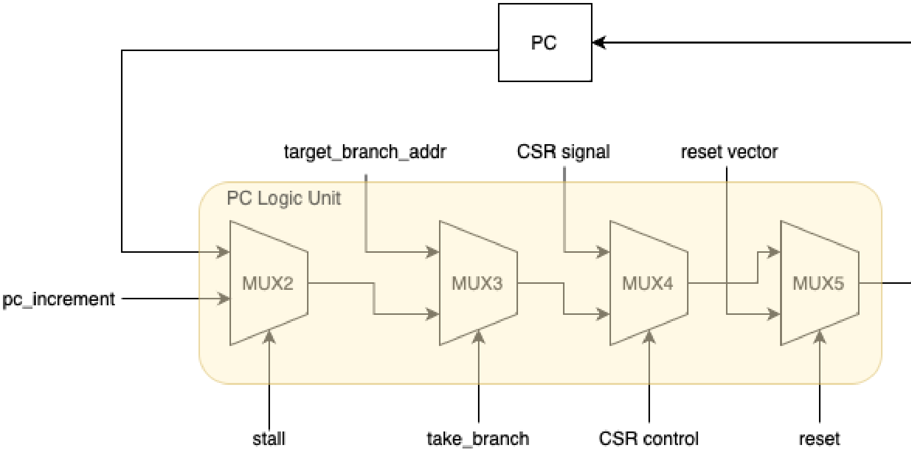


Figure 4.14 : PC Logic Unit Block Diagram

stall_0	stall_1	stall_dual	PC
0	0	0	+pc_increment
0	0	1	+0
0	1	0	+0
0	1	1	+0
1	0	0	+0
1	0	1	+0
1	1	0	+0
1	1	1	+0

Table 4.2 : PC Logic Truth Table

4.7.4 Implementation of the Dual Hazard Unit

In the single-issue processor, the hazard detection unit checks the source registers 1 and 2 from the decode stage and destination register from the execute stage. However, in the dual-issue processor design to detect the hazard between parallel instructions, the dual hazard unit utilizes source registers and destination register all from decode

stage as it is seen in Figure 4.15, since instructions to be compared need to be ones which are fetched in the same clock cycle.

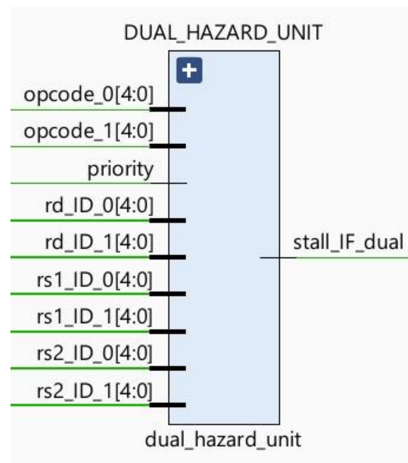


Figure 4.15 : The Schematic View of Dual Hazard Unit

In order to implement the dual hazard unit, first the opcodes of the decoded instructions from both pipes are obtained, besides, the source registers and the destination registers from both pipes are accepted as an input. The most tricky part about the dual hazard unit is to evaluate these parameters. It is crucial to determine which instruction between pipes is the primary in terms of the execution order. This is accomplished by checking the priority bit which tells the core which instruction has the priority in the instruction memory. To make it clear, if it is assumed that the instruction which belongs to the first pipe is the primary, then the source registers of the instruction going through the first pipe and the destination register of the instruction going through the second pipe along with the opcode are analyzed and a dual hazard situation is determined.

```verilog
module dual_hazard_unit (
    input priority,
    input[4:0] rs1_ID_0,
    input[4:0] rs2_ID_0,
    input[4:0] rd_ID_0,
    input[4:0] opcode_0,
    input funct3_0,
    input[4:0] rs1_ID_1,
    input[4:0] rs2_ID_1,
    input[4:0] rd_ID_1,
    input[4:0] opcode_1,
    input funct3_1,
    output reg stall_IF_dual
);

wire uses_rs1_0, uses_rs2_0;

assign uses_rs1_0 = opcode_0[4:1] == 4'b1100 || //JALR and branch instructions
                    opcode_0[4:0] == 5'b00100 || //register-immediate arithmetic
                    opcode_0[4:0] == 5'b01100 || //register-register arithmetic
                    (opcode_0[4:0] == 5'b11100 && funct3_0 == 1'b0); //CSR instructions

assign uses_rs2_0 = opcode_0[4:0] == 5'b11000 || //branch instructions
                    opcode_0[4:0] == 5'b01100; //register-register arithmetic

wire uses_rs1_1, uses_rs2_1;

assign uses_rs1_1 = opcode_1[4:0] == 5'b00000 || //load instructions
                    opcode_1[4:0] == 5'b01000 || //store instructions
                    opcode_1[4:0] == 5'b00100 || //register-immediate arithmetic
                    opcode_1[4:0] == 5'b01100 || //register-register arithmetic
                    (opcode_1[4:0] == 5'b11100 && funct3_1 == 1'b0); //CSR instructions

```

Figure 4.16 : Verilog Codes for the Dual Hazard Unit

```verilog
assign uses_rs2_1 = opcode_1[4:0] == 5'b01000 || //store instructions
                    opcode_1[4:0] == 5'b01100; //register-register arithmetic

always @(*)
begin
    if(priority==0)
    begin
            if((rs1_ID_1 == rd_ID_0 && uses_rs1_1) || (rs2_ID_1 == rd_ID_0 && uses_rs2_1))
                stall_IF_dual = 1'b1;

            else
                stall_IF_dual = 1'b0;
    end
    else if(priority==1)
    begin
            if((rs1_ID_0 == rd_ID_1 && uses_rs1_0) || (rs2_ID_0 == rd_ID_1 && uses_rs2_0))
                stall_IF_dual = 1'b1;

            else
                stall_IF_dual = 1'b0;
    end
end
endmodule

```

Figure 4.17 : Verilog Codes for the Dual Hazard Unit (Continued)

5. REALISTIC CONSTRAINTS AND CONCLUSIONS

Open-source softwares, which all parties can use without any restrictions, has gained importance and popularity in recent years. In this study, it is aimed to increase performance by turning the previously designed open-source single issue (hornet) into a core dual issue. This design is completely open to everyone's contribution and update. In addition, this design is planned to be used in various low-power, high-performance applications and in the field of education.

5.1 Practical Application of this Project

This work can be used in a variety of applications that require low power and are not very complex. With artificial intelligence becoming popular again, computing units have begun to be needed in edge applications. This design may be a suitable solution for those computational units.

5.2 Realistic Constraints

5.2.1 Social, environmental and economic impact

More optimized operation of the system indirectly causes power consumption to be less than alternatives. In this way, its negative impact on the environment will be reduced. Additionally, there is no need to pay any fee to produce this design. If the design is turned into a product, this product will have positive economic impacts on the country. In addition, in case of encountering various embargo dangers, this product will constitute an alternative and will be able to undertake critical tasks in important areas like the defense industry.

5.2.2 Cost analysis

Many things to be done within the scope of this project will be carried out through software, so the costs will be relatively less. Programs such as Xilinx Vivado and Github repo that will be used during the development of the project are free for students. In addition, the FPGA we will use costs approximately 10,000 TRY and is provided free of charge by the school's lab. Finally, two engineers allocate a total of 16 hours a week for the project. Considering the 9-month working period and the salary of a newly graduated engineer, the cost of this project in terms of labor is 144,000 TRY.

5.2.3 Standards

Since project is not working on a subject that relies on protocols such as communication systems, the number of standards we use in the project will be relatively less. The standards we will use are as follows: The RISC-V Instruction Set Manual IEEE Standards which is relevant with the project

5.2.4 Health and safety concerns

This project has no visible or invisible health and safety concerns.

5.3 Future Work and Recommendations

This work is completely open source so that it can be developed further in the future. By making various additions to the dual issue design architecture, a higher performance core can be obtained by turning it into a multiple issue. Or, this design can become more performant by using different performance-enhancing methods such as branch prediction with fewer changes. Or the current design can be optimized to obtain a more efficient output. Finally, the design chip tape-out can be created and a product can be created.

REFERENCES

[1] **Url-1** *<https://www.qualcomm.com/news/onq/2023/09/what-is-risc-v-and-why-were-unlocking-its- potential>,* date of access 26.05.2024.

[2] **Zhang Hongsheng et al 2020** J. Phys.: Conf. Ser. 1693 012192

[3] **Patterson D., Hennessy J.** (2018). *Computer Organization and Design (RISC-V Edition*). Elsevier Inc.

[4] **Url-2** *<https://www.geeksforgeeks.org/difference-between-von-neumann-and-harvard-architecture/>,* date of access 26.05.2024.

[5] **Wentzlaff** (2013). Session 4 presentation: Computer Architecture ELE 475 / COS 475 Superscalar.Princeton University. https://eleclass.princeton.edu/classes/ele475/spring_2018/lib/exe/fetch.php?media=sd4.pdf

[6] **Fasiku, A. I., Olawale, J. B., & Jinadu, O. T**. (2012). A Review of Architectures - Intel Single Core, Intel Dual Core and AMD Dual Core Processors and the Benefits. *International Journal of Engineering and Technology*, *2*.

[7] **Waterman A., Lee Y., Patterson D. A., Asanovic K.** (2016). The RISC-V Instruction Set Manual, Volume I: User- Level ISA, Version 2.1 (Publication No. UCB/EECS-2016-118). Electrical Engineering and Computer Sciences University of California at Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html

[8] **Url-3** *<https://www.sifive.com>, date of access 26.05.2024.*

[9] **Url-4** *<https://pulp-platform.org/implementation.html>, date of access 26.05.2024.*

[10] **Tozlu, Y., Yılmaz, Y.** (2021). Design And Implementation of A 32-Bit RISC-V Core

[11] **T, G., Muraleedharan, A., & Varghese, K.** (2020). Design of a 32-bit, dual pipeline superscalar RISC-V processor on FPGA (thesis).

[12] **A. A. Kılıç, M. K. Ertem,** “Hornet RISC-V Core,” GitHub, Sep. 26, 2023. https://github.com/mskertem/RISC-V (accessed Jan. 2024)

[13] **Parthasarathi, D. R.** (2018, July 24). *Computer architecture*. Pipeline Hazards – Computer Architecture. https://www.cs.umd.edu/~meesh/411/CA-online/chapter/pipeline-hazards/index.html

APPENDICES

APPENDIX A: RTL Schematics

APPENDIX A

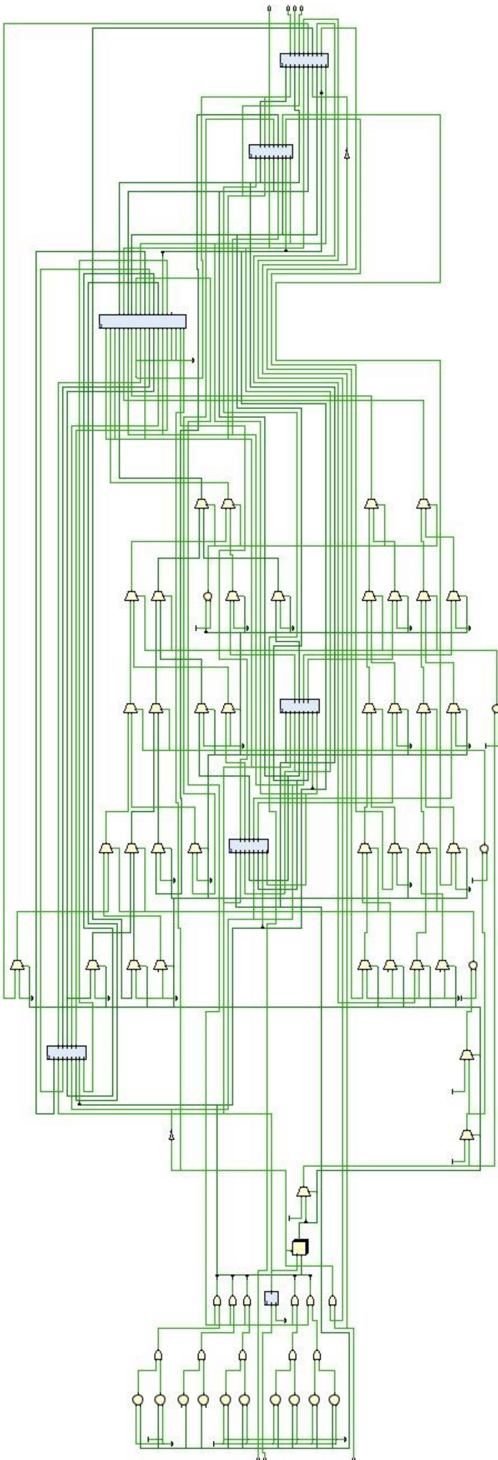


Figure A.1 : RTL Schematic of the Top Module in Vivado

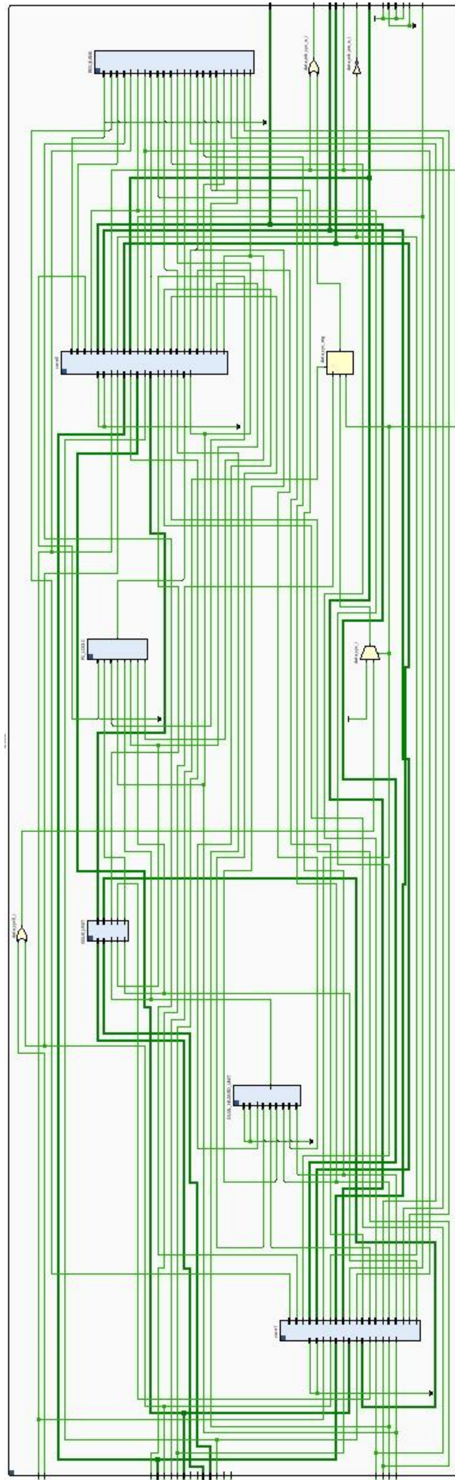


Figure A.2 : RTL Schematic of the Core_wb Module in Vivado

CURRICULUM VITAE

Name Surname : Abdullah Aykut KILIÇ

Place and Date of Birth : Kayseri, 20.05.2001

E-Mail : kabdullahaykut@gmail.com

CURRICULUM VITAE

Name Surname	**:** Mustafa Kerem ERTEM
Place and Date of Birth	**:** Ankara, 18.04.2002
E-Mail	**:** msk.ertem@gmail.com

Experience

- TUBITAK BILGEM TUTEL – Kocaeli, Turkey June – July 2023
- ITU Aerospace Research Center – Istanbul, Turkey June – July 2022