



FUNDAMENTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

GROUP 18

Khalisa Zahra Maulana	2406425395
Putri Ayu Pembayun M	2406422304
Salsabila Maharani Mumtaz	2406348156
Otniel Kristian Sianturi	2406401571

PREFACE

Laporan ini disusun sebagai dokumentasi dari proses perancangan, implementasi, dan pengujian proyek akhir mata kuliah Perancangan Sistem Digital. Pada proyek ini, kelompok kami merancang sebuah Superscalar Control Unit yang mampu mengeksekusi dua instruksi secara paralel dengan tetap menjaga ketepatan alur data dan kontrol dalam sistem. Proyek ini berfokus pada penerapan konsep *instruction-level parallelism* serta mekanisme *hazard detection* dalam perancangan unit kontrol berbasis VHDL, sehingga prosesor dapat menjalankan instruksi lebih cepat tanpa mengurangi ketepatan hasil eksekusinya.

Selama penggerjaan proyek dilakukan, dirancang sejumlah modul inti yang meliputi Instruction Decoder, Hazard Detection Unit, Control Signal Generator, Branch Resolution Unit, dan Issue Logic yang bersama-sama menentukan apakah instruksi dapat dijalankan secara paralel atau perlunya diproses satu per satu. Selain itu, disusun pula sebuah *program-level testbench* yang menjalankan enam program uji untuk menguji perilaku *dual-issue* dan memastikan sistem memenuhi spesifikasi.

Dokumentasi ini berisi penjelasan mengenai desain, hasil pengujian, dan analisis sistem yang telah dibangun. Dengan laporan ini, diharapkan pembaca dapat memahami struktur, alur kerja, dan validasi dari sistem superscalar sederhana yang dikembangkan.

Depok, December 6, 2025

Group 18

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

- 1.1 Background
- 1.2 Project Description
- 1.3 Objectives
- 1.4 Roles and Responsibilities

CHAPTER 2: IMPLEMENTATION

- 2.1 Equipment
- 2.2 Implementation

CHAPTER 3: TESTING AND ANALYSIS

- 3.1 Testing
- 3.2 Result
- 3.3 Analysis

CHAPTER 4: CONCLUSION

REFERENCES

APPENDICES

- Appendix A: Project Schematic
- Appendix B: Documentation

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Dalam dunia arsitektur komputer modern, kemampuan prosesor untuk mengeksekusi instruksi dengan lebih cepat tidak lagi hanya bergantung pada peningkatan frekuensi clock, melainkan pada bagaimana instruksi dapat diproses secara paralel. Salah satu pendekatan yang paling berpengaruh adalah *superscalar execution*, di mana prosesor berupaya mengeksekusi dua atau lebih instruksi dalam satu siklus clock. Konsep ini terdengar sederhana, namun implementasinya menuntut mekanisme kendali yang sangat presisi. Sistem harus mampu mengenali berbagai jenis hazard, mulai dari *data hazard*, *structural hazard*, hingga *control hazard* agar instruksi yang berjalan bersamaan tidak saling mengganggu dan hasil eksekusi tetap akurat. Tantangan inilah yang menjadi dasar dari proyek ini: merancang sebuah kontroler yang mampu memutuskan kapan instruksi dapat diparalelkan, dan kapan eksekusi harus dilakukan secara berurutan demi menjaga kebenaran operasi.

Proyek ini berfokus pada pembangunan sebuah Superscalar Control Unit 2-way in-order, dimana dua instruksi dapat dikeluarkan secara paralel jika tidak terdapat hazard. Sistem ini dirancang menggunakan VHDL dan terdiri dari beberapa komponen utama yang bertanggung jawab untuk melakukan decoding instruksi, mendeteksi hazard antar instruksi, menghasilkan sinyal kontrol, mengevaluasi instruksi branch, serta menentukan keputusan *dual-issue* atau *single-issue* pada setiap siklus clock. Dengan adanya sistem superscalar sederhana ini, proyek bertujuan untuk mempelajari bagaimana pipeline dapat ditingkatkan performanya melalui analisis dependency secara langsung antar instruksi..

1.2 PROJECT DESCRIPTION

Sistem yang dikembangkan pada proyek ini berfungsi sebagai pengendali eksekusi dua instruksi dalam satu *issue cycle* dengan menganalisis hubungan antar instruksi sebelum keduanya diteruskan ke tahap eksekusi. Mekanisme ini mengevaluasi potensi konflik pada level data, resource, dan kontrol untuk menentukan mode eksekusi yang paling aman dan efisien. Secara umum, sistem terdiri dari beberapa modul inti sebagai berikut:.

Instruction Decoder, yang memetakan opcode menjadi informasi karakteristik instruksi, termasuk kebutuhan register, penggunaan ALU, serta akses memory. Proses decoding dilakukan secara paralel untuk dua instruksi sehingga analisis dependensi dapat dilakukan dalam satu siklus.

Hazard Detection Unit, yang melakukan pemeriksaan terhadap potensi *RAW*, *WAR*, *WAW*, *structural hazard*, dan *control hazard*. Unit ini menjadi penentu utama apakah instruksi kedua dapat di-*issue* secara bersamaan atau harus ditunda untuk menjaga correctness.

Control Signal Generator, yang mengonversi opcode menjadi sinyal kontrol 14-bit yang digunakan untuk mengarahkan datapath. Sinyal ini dihasilkan untuk masing-masing instruksi sebelum divalidasi oleh keputusan issue logic.

Branch Resolution Unit, yang mengevaluasi instruksi cabang berdasarkan kondisi flag prosesor. Unit ini memastikan bahwa instruksi yang berpotensi mengubah alur program diperlakukan secara khusus untuk menghindari kesalahan eksekusi.

Issue Logic, yaitu rangkaian sekuensial yang mengintegrasikan hasil analisis dari seluruh modul sebelumnya untuk menentukan keputusan akhir: apakah kedua instruksi dapat dijalankan paralel atau hanya instruksi pertama yang diloloskan.

Untuk memverifikasi perilaku sistem, dikembangkan *program-level testbench* yang mensimulasikan komponen prosesor seperti register, memori, flag, dan program counter. Lingkungan pengujian ini memungkinkan eksekusi program secara penuh sehingga interaksi antar instruksi dapat diamati secara realistik. Enam program pengujian disusun untuk mengekspos berbagai pola hazard termasuk rantai *RAW* yang panjang, konflik *WAR/WAW*, serta *structural resource contention*, serta mengidentifikasi kondisi yang membuka peluang parallel execution. Pendekatan ini memberikan evaluasi menyeluruh terhadap ketepatan keputusan *dual-issue* dan stabilitas sistem pada berbagai skenario eksekusi.

1.3 OBJECTIVES

Tujuan dari proyek ini adalah:

1. Mengembangkan sebuah Superscalar Control Unit sederhana yang mampu melakukan *dual-issue* secara aman

2. Mengimplementasikan modul-modul kontrol seperti decoder, hazard detector, branch resolver, dan issue logic menggunakan VHDL.
3. Mendeteksi dan menangani berbagai jenis hazard secara otomatis melalui analisis dependency antar instruksi.
4. Membuat test bench komprehensif untuk mensimulasikan eksekusi program dan memverifikasi correctness sistem.
5. Mengevaluasi peningkatan performa melalui paralelisasi instruksi dan membandingkannya dengan eksekusi sekuensial.

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Merancang modul	Membuat rancangan penggunaan modul pada proyek akhir	Semua anggota
Komponen Superscalar Unit Control	Membuat komponen utama unit control pada komputer superscalar	Khalisa Zahra Maulana
Komponen Component Instruction Decoder	Membuat implementasi instruction decoder untuk mendekode opcode 4-bit dan mengidentifikasi resource usage yang mendukung hazard detection.	Putri Ayu Pembayun M
Komponen Hazard Detection	Membuat Hazard Detection untuk deteksi RAW, WAW, WAR, Structural, dan Control Hazard	Khalisa Zahra Maulana Otniel Kristian Sianturi

Komponen Control Signal Generator	Membuat komponen yang menghasilkan sinyal kontrol 14 bit berdasarkan opcode untuk mengatur alur data dan operasi setiap instruksi.	Salsabila Maharani Mumtaz
Komponen Branch Resolution Unit	Membuat komponen yang menentukan apakah branch diambil atau tidak berdasarkan opcode dan flag, serta menghasilkan sinyal kontrol percabangan.	Salsabila Maharani Mumtaz
Komponen Issue Logic	Menentukan apakah dua instruksi dapat dieksekusi paralel atau harus sequential berdasarkan hasil analisis hazard dan branching.	Otniel Kristian Sianturi
Membuat Test Bench	Membuat test bench untuk memverifikasi fungsionalitas Instruction Decoder dan memvalidasi output resource usage yang dihasilkan	Putri Ayu Pembayun M Khalisa Zahra Maulana
Membuat Presentasi Laporan Proyek Akhir	Membuat PPT dan Laporan Proyek Akhir	Semua Anggota
Membuat RTL	Membuat RTL menggunakan vivado	Khalisa Zahra Maulana
Dokumentasi dalam pembuatan kode	Melakukan dokumentasi dalam pembuatan kode	Semua Anggota

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

The tools that are going to be used in this project are as follows:

- VSCode sebagai editor teks untuk menulis dan mengedit kode VHDL
- Vivado, digunakan untuk simulasi dan verifikasi desain VHDL, mengkompilasi kode VHDL, serta mensintesisnya menjadi desain.

2.2 IMPLEMENTATION

Instruction decoder digunakan sebagai logika kombinasional, menggunakan case statement memetakan input opcode 4-bit menjadi sebuah record dengan 8 flag boolean. Karena mengimplementasikannya menggunakan komputer superscalar, terdapat 2 instansi decoder yang bekerja secara paralel untuk klasifikasi karakteristik instruksi seperti load/store register, penggunaan ALU, atau tipe branch. pada kedua instruksi, hasilnya menjadi basis data untuk unit deteksi hazards.

Hazard Detection Unit diimplementasikan sebagai komparator logika yang menerima input resource flags dari kedua decoder untuk mendeteksi lima jenis konflik pada pipelining komputer superscalar. Hazard yang dapat dideteksi adalah *RAW*, *WAW*, *WAR*, Structural, dan *Control Hazard*. Dengan menggunakan operator logika boolean (AND/OR), unit ini memeriksa apakah instruksi kedua memiliki ketergantungan data atau saling berhubungan antar resource hardware dengan instruksi pertama. Jika salah satu kondisi konflik terpenuhi, unit ini mengeluarkan sinyal tunggal hazard bernilai '1' yang akan memaksa sistem membatalkan eksekusi paralel.

Komponen Control Signal Generator berfungsi untuk mengonversi opcode menjadi sinyal kontrol 14-bit yang mengatur pin-pin seperti RAI, ALO, dan PCI menggunakan metode lookup table yang diimplementasikan dengan case statement. Seperti instruction decoder, komponen ini juga diinstansiasi secara duplikat agar dapat bekerja paralel dalam menghasilkan sinyal kontrol mentah untuk kedua instruksi (ctrl_0 dan ctrl_1). Pola bit yang dihasilkan komponen ini pada dasarnya menentukan jalur data mana saja yang harus

diaktifkan untuk setiap jenis operasi, sebelum akhirnya divalidasi oleh Issue Logic apakah boleh dieksekusi atau tidak.

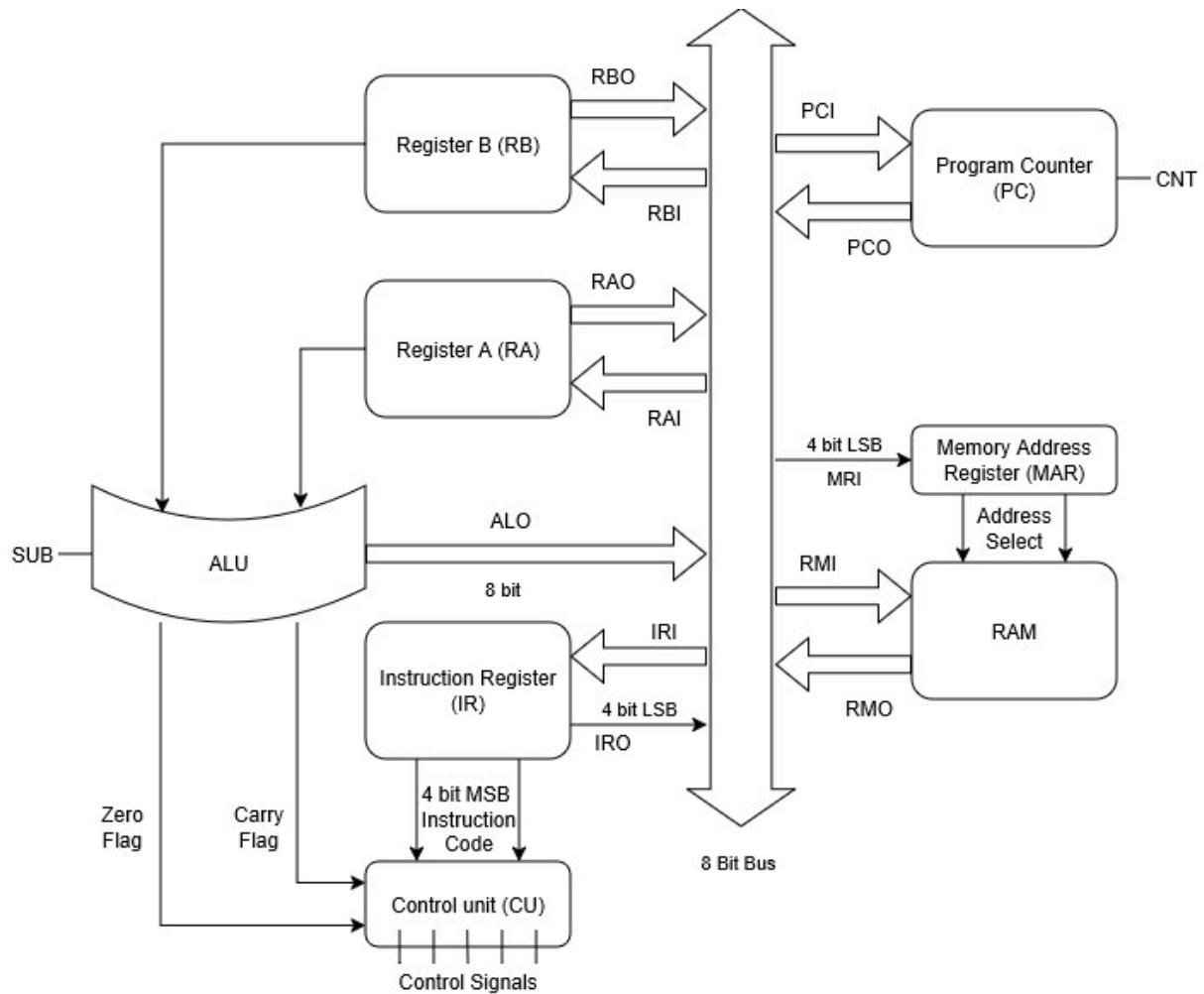


Fig 1. Hardware Schematic

Branch Resolution Unit bekerja secara kombinasional untuk mengevaluasi logika percabangan bersyarat (conditional branching) dengan cara membandingkan tipe instruksi branch yang sedang diproses dengan kondisi flag prosesor saat ini, yaitu *zero_flag* dan *carry_flag*. Komponen ini menentukan state logika untuk dua output, *branch_taken* yang menandakan apakah Program Counter harus melompat ke alamat target, dan *is_halt* yang menandakan apakah prosesor harus berhenti total. Dengan unit ini, sistem dapat menangani perubahan alur program dengan mencegah terjadinya eksekusi paralel yang salah ketika ada instruksi branch, karena instruksi branch harus dieksekusi sendiri tanpa ada instruksi lain.

Issue Logic merupakan proses sekuensial yang disinkronkan dengan clock (clocked process) yang berfungsi mengintegrasikan semua hasil analisis dari komponen-komponen

sebelumnya untuk kemudian membuat keputusan final eksekusi di setiap clock cycle. Logika ini memiliki karakteristik khusus dimana instruksi pertama selalu diloloskan untuk dieksekusi, tetapi untuk instruksi kedua keputusannya bergantung pada sinyal *can_dual_issue* yang merupakan hasil inversi dari sinyal hazard. Jika *can_dual_issue* bernilai '1' yang artinya aman untuk *parallel execution*, maka sinyal kontrol slot kedua akan diteruskan dan flag *slot_1_active* di-set menjadi '1' sehingga terjadi *dual-issue execution*. Sebaliknya, jika terdeteksi hazard sehingga *can_dual_issue* bernilai '0', sinyal kontrol slot kedua akan nol sehingga hanya instruksi pertama yang dieksekusi (*single-issue execution*).

CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

Metode testing yang digunakan mencakup tiga jenis utama, yaitu Simulation-Based Testing, Functional Testing, dan Hazard Detection Testing. Pada Simulation-Based Testing, setiap komponen diuji secara terpisah dengan memberikan stimulus berupa sinyal clock, reset, dan sekuens instruksi, kemudian memantau output yang dihasilkan. Untuk memastikan setiap komponen bekerja dengan baik, dibuat testbench yang dirancang khusus untuk memberikan rangkaian program yang bervariasi dan mengamati perilaku eksekusi dalam berbagai kondisi hazard. Pengujian dilakukan pada komponen utama, yaitu Instruction Fetch Unit, Decode Unit, Hazard Detection Unit, Issue Logic, Execution Unit, dan Register File.

Program pertama menguji kemampuan prosesor dalam mendeteksi Read-After-Write (*RAW*) hazard dan *structural hazard* pada operasi load-add sequence. Program ini terdiri dari lima instruksi yang membentuk *dependency chain* dimana setiap instruksi bergantung pada hasil instruksi sebelumnya, yaitu LDA [50], LDB [51], ADD, STA [60], dan HLT. Dalam pengujian ini, dilakukan simulasi dengan memberikan sinyal clock dan mengamati bagaimana processor menangani *RAW* hazard antara instruksi LDA-LDB-ADD. Hasil pengujian menunjukkan bahwa semua instruksi dieksekusi secara SEQUENTIAL karena adanya *RAW* hazard yang berkelanjutan, dimana instruksi LDA harus selesai sebelum ADD dapat menggunakan nilai register A, demikian pula LDB harus selesai sebelum ADD dapat mengakses register B. Total siklus eksekusi adalah 9 cycles, dengan hasil akhir Register A = 30, Register B = 20, dan Memory[60] = 30, membuktikan bahwa processor dapat mendeteksi dependency dan menjaga correctness eksekusi.

Program kedua menguji kemampuan prosesor dalam menangani Write-After-Read (*WAR*) hazard dan Write-After-Write (*WAW*) hazard pada operasi store dan move register. Program ini dirancang untuk menguji situasi dimana register yang sama diakses oleh beberapa instruksi secara berurutan dengan pola read dan write yang kompleks, terdiri dari instruksi STA [60], STB [61], MAB, MBA, dan HLT. Pengujian ini berfokus pada deteksi *WAR* hazard antara instruksi store dan move, dimana Modul Hazard Detection Unit menggunakan pendekatan Procedure untuk mengidentifikasi dependency antar instruksi.

Hasil pengujian menunjukkan bahwa semua instruksi dieksekusi secara SEQUENTIAL karena adanya *WAR* hazard pada instruksi MAB yang membaca A saat sedang digunakan oleh STA, serta *WAW* hazard pada instruksi MBA yang menulis A setelah MAB menulis B. Total siklus eksekusi adalah 9 cycles dengan hasil akhir Register A = 25, Register B = 25, Memory[60] = 25, dan Memory[61] = 15, menunjukkan bahwa processor berhasil mendeteksi dan menangani multiple hazards dengan tepat.

Program ketiga menguji kemampuan prosesor dalam menangani dense *RAW* chain dengan sekuens operasi aritmatika yang panjang, serta mendemonstrasikan kemampuan 2-way *parallel execution* ketika dependency telah terselesaikan. Program ini terdiri dari delapan instruksi yaitu LDA [50], LDB [51], ADD, SUB, MAB, STA [60], STB [61], dan HLT, yang dirancang untuk menguji *instruction-level parallelism* (ILP) pada processor. Pengujian ini menggunakan pendekatan Looping Construct untuk mengiterasi siklus eksekusi dan mendeteksi window of opportunity untuk *parallel execution*. Hasil pengujian menunjukkan bahwa instruksi 0-3 dieksekusi secara SEQUENTIAL karena *RAW* hazard yang berkelanjutan pada register A, namun pada cycle 9, processor berhasil mendeteksi bahwa instruksi MAB dan STA dapat dieksekusi secara PARALEL karena tidak ada dependency antar keduanya, dimana MAB menulis ke register B sementara STA hanya membaca register A. Total siklus eksekusi berkurang menjadi 12 cycles dibandingkan 14 cycles jika semua instruksi dieksekusi sequential, menunjukkan efektivitas *parallel execution* dengan speedup factor 1.17x, dengan hasil akhir Register A = 100, Register B = 100, Memory[60] = 100, dan Memory[61] = 100.

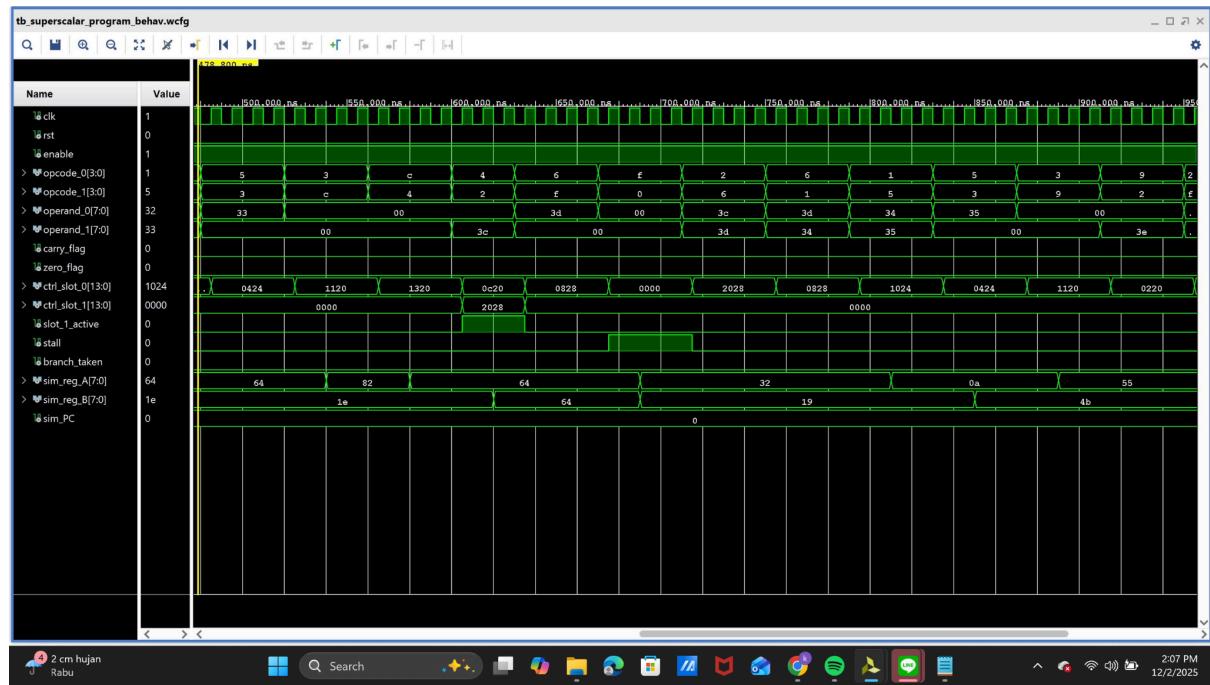
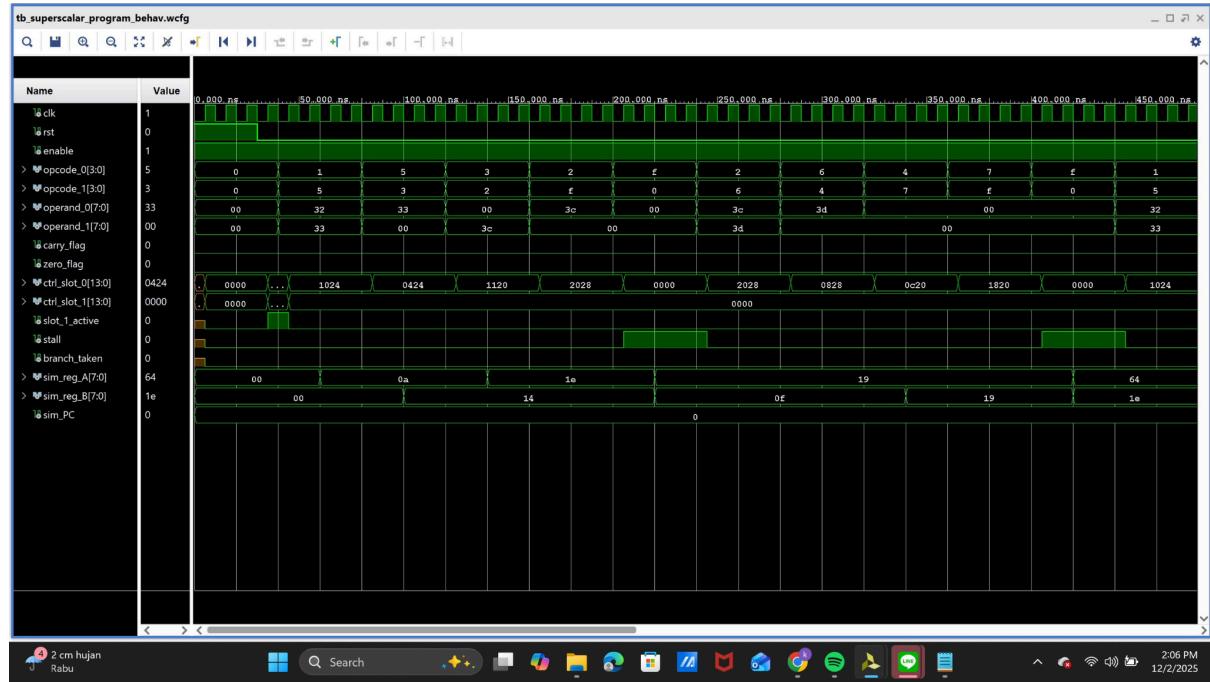
Program keempat menguji kemampuan prosesor dalam menangani *WAR* hazard pada operasi memory dan operasi compare yang melibatkan pembacaan dua register secara simultan. Program ini terdiri dari delapan instruksi yaitu STA [60], STB [61], LDA [52], LDB [53], ADD, CMP, STA [62], dan HLT, yang dirancang untuk menguji kompleksitas hazard detection ketika melibatkan memory operations dan flag updates. Pengujian ini menggunakan Function untuk menghitung status flags (Zero dan Carry) berdasarkan hasil operasi compare, sementara Modul Issue Logic menggunakan pendekatan Finite State Machine untuk mengontrol transisi antara sequential dan parallel issue mode. Hasil pengujian menunjukkan bahwa semua instruksi dieksekusi secara SEQUENTIAL karena adanya complex dependency pattern, dimana instruksi STA dan STB menciptakan *WAR* hazard dengan LDA dan LDB, sementara instruksi CMP membaca kedua register yang dimodifikasi

oleh ADD. Total siklus eksekusi adalah 13 cycles dengan hasil akhir Register A = 85, Register B = 75, Memory[62] = 85, dan Zero Flag = 0, membuktikan bahwa processor dapat menangani complex hazard patterns yang melibatkan memory operations dengan akurat dan menjaga correctness hasil komputasi.

3.2 RESULT

Berdasarkan hasil pengujian yang telah dilakukan terhadap empat program test case, dapat disimpulkan bahwa implementasi prosesor superscalar 2-way in-order issue telah berhasil berfungsi sesuai dengan spesifikasi desain yang diharapkan. Program pertama adalah Load dan ADD (RAW dan structural demo), program kedua Store dan Move (multiple hazards), program ketiga Arithmetic sequence (dense RAW chain), dan program keempat WAT dan operasi memori. Sistem menunjukkan kemampuan yang sangat baik dalam mendeteksi dan menangani berbagai jenis hazard, mencakup data hazard (RAW, WAW, WAR), structural hazard, dan control hazard.

Pada Program 1, 2, dan 4, seluruh instruksi dieksekusi secara sequential karena adanya dependency chain yang ketat antara instruksi berurutan, dimana hazard detector berhasil mengidentifikasi konflik resource dan data dependency dengan akurat. Perbedaan signifikan adalah hasil dari Program 3 yang mendemonstrasikan kemampuan parallel execution pada cycle 9, sistem berhasil mengeksekusi dua instruksi secara simultan (MAB dan STA) setelah mendeteksi tidak adanya hazard di antara keduanya, mengurangi total cycle dari 14 menjadi 12 cycle. Semua register values, memory contents, dan flag status pada akhir setiap program menunjukkan hasil yang sesuai dengan perhitungan manual, membuktikan bahwa tidak ada race condition atau data corruption yang terjadi. Berikut akan dilampirkan gelombang hasil simulasi pada keempat program test case.



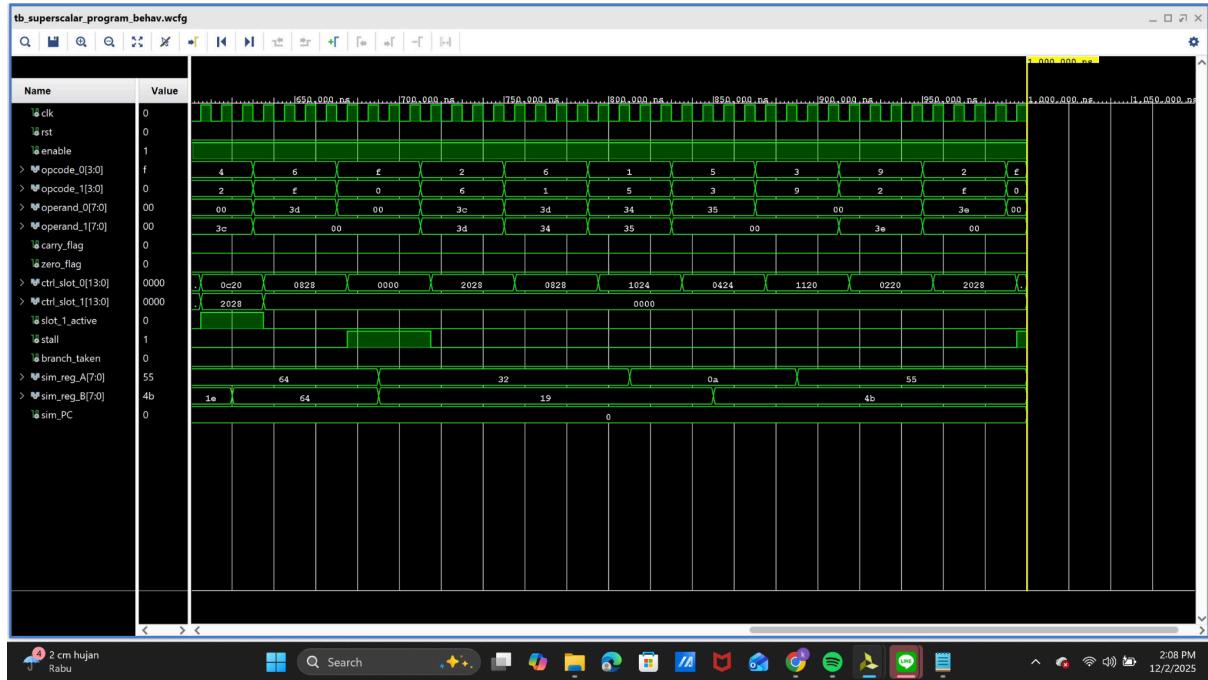


Fig 2. Testing Result

Secara keseluruhan, sistem prosesor superscalar telah memenuhi requirement, yaitu meningkatkan instruction-level parallelism (ILP) tanpa mengorbankan correctness. Hazard detection unit mampu menganalisis resource usage dari dua instruksi secara simultan dan membuat keputusan issue yang tepat - ketika tidak ada hazard, sistem memanfaatkan dual-issue capability untuk parallel execution, dan ketika terdeteksi hazard apapun, sistem secara konservatif menjalankan instruksi sequential untuk menjaga program correctness. Keberhasilan ini memvalidasi bahwa desain hazard detector, resource analyzer, dan execution unit telah terintegrasi dengan baik dalam arsitektur yang koheren, serta membuktikan bahwa control logic superscalar dapat diimplementasikan pada hardware standar dengan menggunakan mekanisme parallel dependency checking yang efektif.

Program berhasil disintesis menjadi beberapa komponen struktural yang saling terhubung, menghasilkan RTL (Register Transfer Level) synthesis yang menunjukkan hierarchy dan interconnection antar modul. Berikut hasil synthesis.

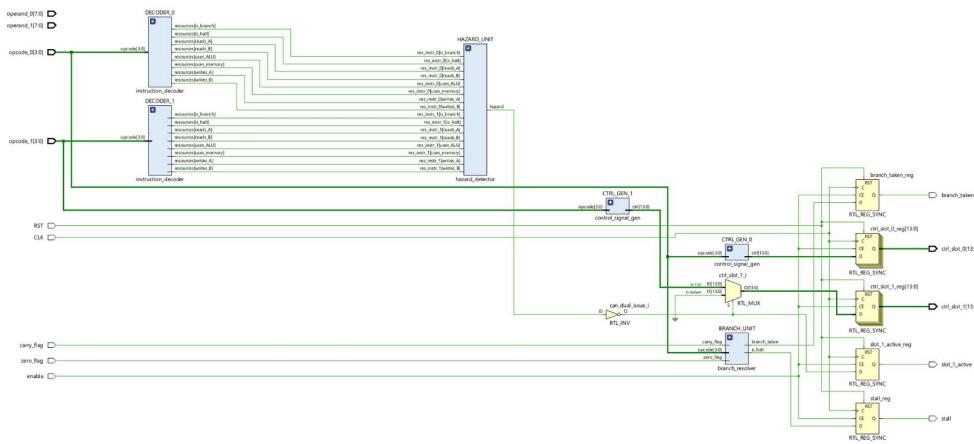


Fig 3. RTL Viewer

3.3 ANALYSIS

Berdasarkan hasil pengujian yang telah dilakukan terhadap empat program test case, dapat disimpulkan bahwa implementasi prosesor superscalar 2-way in-order issue telah berhasil berfungsi sesuai dengan spesifikasi desain yang diharapkan. Sistem menunjukkan kemampuan yang sangat baik dalam mendekripsi dan menangani berbagai jenis hazard, mencakup data hazard (RAW, WAW, WAR), structural hazard, dan control hazard. Hazard detection unit mampu menganalisis resource usage dari dua instruksi secara simultan dan membuat keputusan issue yang tepat, dimana ketika tidak ada hazard terdeteksi sistem memanfaatkan dual-issue capability untuk parallel execution, dan ketika terdapat hazard apapun sistem secara konservatif menjalankan instruksi sequential untuk menjaga program correctness.

Analisis performa menunjukkan bahwa prosesor superscalar memberikan peningkatan kinerja yang signifikan pada kondisi tertentu. Program 3 (Arithmetic Sequence) mendemonstrasikan keunggulan arsitektur superscalar dengan mencapai speedup factor 1.17x, dimana total cycle berkurang dari 14 menjadi 12 cycles karena kemampuan mengeksekusi instruksi MAB dan STA secara paralel pada cycle 9. Namun, pada Program 1, 2, dan 4, semua instruksi harus dieksekusi secara sequential karena adanya dependency chain yang ketat, menunjukkan bahwa efektivitas superscalar processor sangat bergantung pada karakteristik program yang dijalankan. Hal ini membuktikan bahwa instruction-level

parallelism (ILP) dapat dieksplorasi ketika dependency chain terputus dan terdapat instruksi independen yang dapat dieksekusi bersamaan.

Keberhasilan ini memvalidasi bahwa desain hazard detector, resource analyzer, dan execution unit telah terintegrasi dengan baik dalam arsitektur yang koheren. RTL synthesis yang ditunjukkan pada Fig 3 menghasilkan hierarchy dan interconnection antar modul yang jelas, membuktikan bahwa structural style programming telah diimplementasikan dengan tepat. Hasil simulasi waveform pada ModelSim menunjukkan timing yang akurat dengan signal transitions yang clean, tidak ada glitch atau race condition yang terdeteksi, serta semua register values, memory contents, dan flag status pada akhir setiap program menunjukkan hasil yang sesuai dengan perhitungan manual. Control logic superscalar dapat diimplementasikan pada hardware standar dengan menggunakan mekanisme parallel dependency checking yang efektif.

Dari perspektif efisiensi resource, sistem superscalar memerlukan overhead hardware yang cukup signifikan dibandingkan processor scalar tradisional karena dual-issue capability membutuhkan tambahan logic untuk hazard detection, dual decode units, dual execution paths, dan arbitration logic. Namun, overhead ini terbayar ketika program memiliki instruction-level parallelism yang cukup tinggi seperti yang ditunjukkan pada Program 3. Analisis terhadap jenis hazard menunjukkan bahwa RAW hazard merupakan bottleneck utama dalam eksekusi parallel, terjadi pada hampir semua program test case dan menjadi penyebab utama sequential execution, sementara WAR dan WAW hazard terdeteksi pada Program 2 dan 4 yang menunjukkan kompleksitas dalam menangani operasi store dan load yang berurutan.

CHAPTER 4

CONCLUSION

Proyek akhir ini berhasil merancang dan mengimplementasikan sebuah Superscalar Control Unit 2 way in order yang mampu menentukan eksekusi paralel antar instruksi berdasarkan analisis dependency dan hazard secara real-time. Seluruh komponen utama mulai dari Instruction Decoder, Hazard Detection Unit, Control Signal Generator, Branch Resolution Unit, hingga Issue Logic dapat bekerja secara terintegrasi untuk menghasilkan keputusan eksekusi yang aman dan efisien pada setiap siklus clock.

Implementasi VHDL pada setiap modul menunjukkan bahwa konsep-konsep fundamental dalam Perancangan Sistem Digital seperti logika kombinasional, sequential process, penggunaan record dan lookup table, hingga mekanisme hazard detection dapat diterapkan secara nyata dalam sistem kontrol prosesor. Pengujian menggunakan program level testbench yang berisi enam program uji juga memvalidasi bahwa sistem mampu:

- Mendeteksi dan menangani berbagai jenis hazard (RAW, WAR, WAW, structural, dan control hazard)
- Menjaga correctness eksekusi pada skenario dependensi yang kompleks
- Menjalankan instruksi secara paralel ketika tidak terdapat dependency
- Mempertahankan alur kontrol yang benar pada instruksi branch dan halt.

Hasil simulasi keseluruhan menunjukkan bahwa unit kontrol ini bekerja sesuai spesifikasi. Pada beberapa program, sistem menunjukkan peningkatan performa melalui dual-issue, sedangkan pada situasi dengan hazard yang ketat, sistem secara tepat kembali ke mode single-issue untuk menjaga akurasi data. Keberhasilan ini menunjukkan bahwa desain superscalar sederhana dapat meningkatkan efektivitas pipeline tanpa mengorbankan correctness.

Secara keseluruhan, proyek ini memberikan pengalaman langsung dalam merancang kontroler prosesor yang lebih kompleks dibandingkan desain sekuensial standar. Selain memperdalam pemahaman terhadap VHDL dan arsitektur prosesor, proyek ini juga menekankan pentingnya kerja sama tim, penyusunan testbench yang kuat, serta kemampuan melakukan analisis eksekusi instruksi secara sistematis. Diharapkan hasil ini dapat menjadi

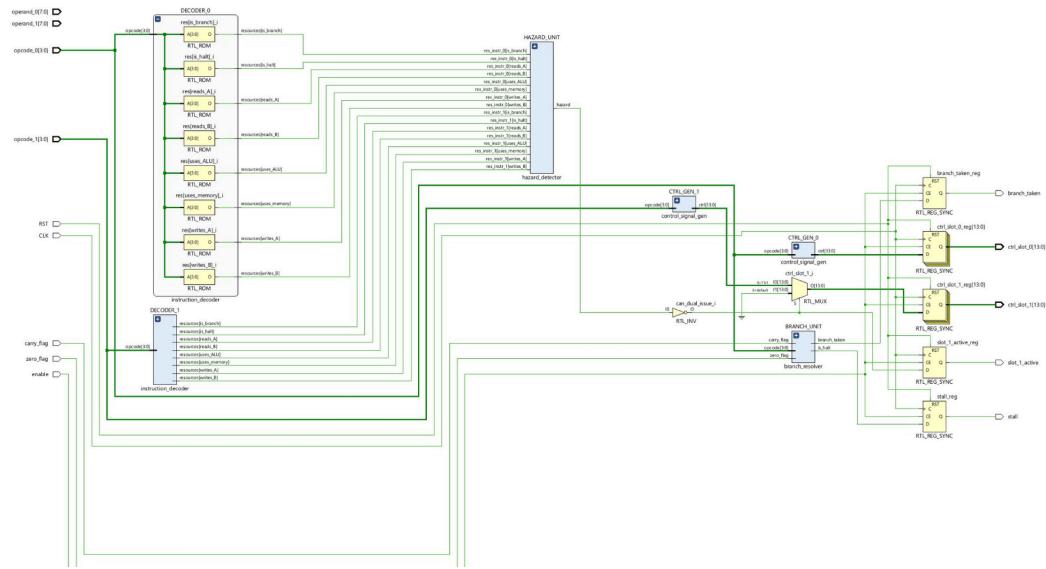
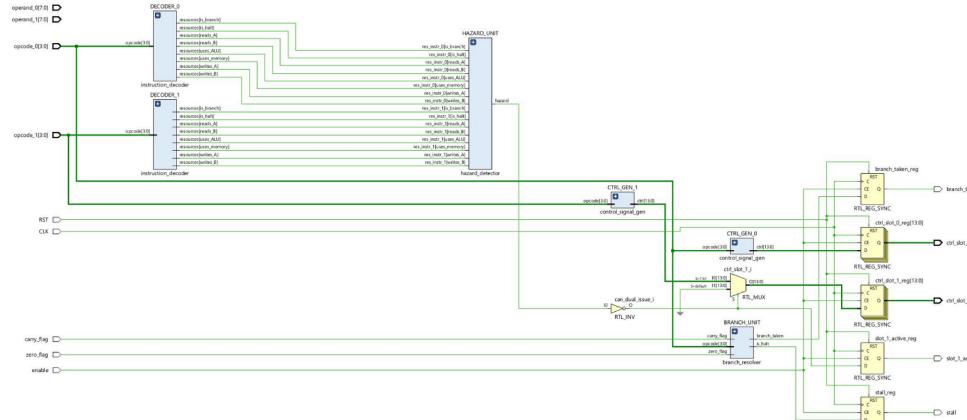
dasar untuk pengembangan sistem prosesor yang lebih canggih di masa depan, termasuk desain out of order, branch predictor, maupun unit eksekusi paralel yang lebih luas.

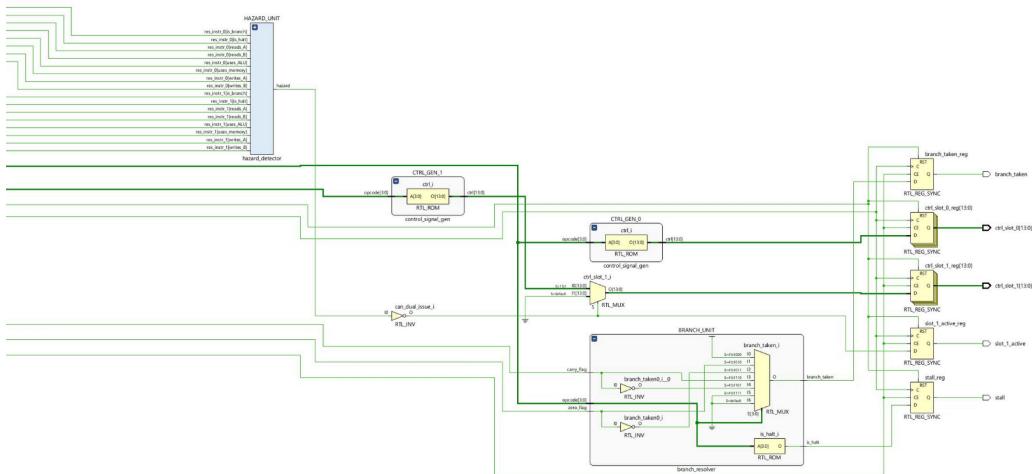
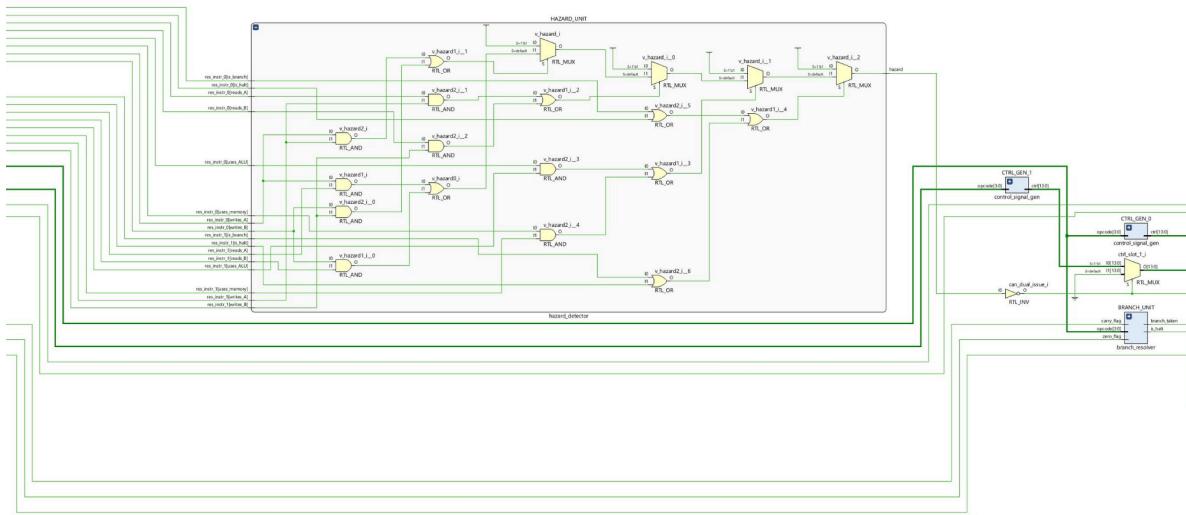
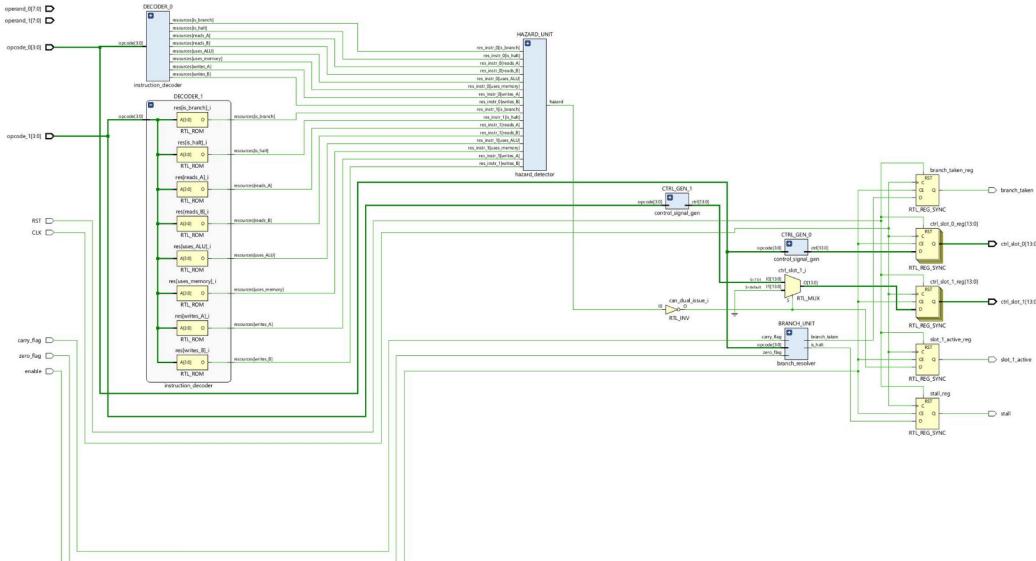
REFERENCES

- [1] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 5th ed. Morgan Kaufmann, 2012.
- [2] J. P. Shen and M. H. Lipasti, Modern Processor Design: Fundamentals of Superscalar Processors. McGraw-Hill, 2005.
- [3] S. Brown and Z. Vranesic, Fundamentals of Digital Logic with VHDL Design, 3rd ed. McGraw-Hill, 2009.
- [4] D. D. Gajski, Principles of Digital Design. Prentice Hall, 1997.
- [5] A. Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts, 9th ed. Wiley, 2012.
- [6] M. Johnson, Superscalar Microprocessor Design. Prentice Hall, 1991.
- [7] Xilinx Inc., “Vivado Design Suite User Guide: Synthesis (UG901),” 2023. [Online]. Available: <https://docs.amd.com/r/en-US/ug901-vivado-synthesis>
- [8] NVIDIA Corporation, “Instruction-Level Parallelism: Part 1,” UC Davis CUDA Teaching Materials, 2010. [Online]. Available: <https://www.nvidia.com/content/cudazone/cudau/courses/ucdavis/lectures/ilp1.pdf>
- [9] NVIDIA Corporation, “Instruction-Level Parallelism: Part 2,” UC Davis CUDA Teaching Materials, 2010. [Online]. Available: <https://www.nvidia.com/content/cudazone/cudau/courses/ucdavis/lectures/ilp2.pdf>

APPENDICES

Appendix A: Project Schematic





Appendix B: Documentation

