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

MINI PROCESSOR

GROUP A3

ALDRIAN RAFI WICAKSONO	**2106653256**
ENRICCO VERINDRA PUTRA	**2106651793**
MUHAMMAD NAJIH AFLAH	**2106653880**
NAJWA FATHIADISA	**2106654391**

PREFACE

Assalamualaikum wr.wb. Puji syukur atas rahmat Allah SWT, berkat rahmat serta karunia-Nya sehingga Laporan Proyek Akhir kami yang berjudul “Mini-Microprocessor” dapat selesai.

Laporan Proyek Akhir ini kami buat dengan tujuan memenuhi tugas akhir Praktikum Perancangan Sistem Digital dengan dosen pengampu Bapak Ruki Harwahyu, S.T., M.Sc., M.T., Ph.D. dan Bapak Yan Maraden, S.T., M.T. Selain itu, penyusunan laporan ini juga bertujuan untuk memperluas wawasan pembaca mengenai penerapan mini processor dalam kode VHDL.

Kelompok kami juga ingin menyampaikan ucapan terima kasih kepada Bapak Ruki Harwahyu, S.T., M.Sc., M.T., Ph.D , Bapak Yan Maraden, S.T., M.T. dan Kakak Asisten Laboratorium Digital yang telah membimbing penyusunan proyek akhir ini. Berkat proyek akhir yang diberikan ini, dapat menambah wawasan kami mengenai implementasi microprocessor dengan menggunakan VHDL. Penulis juga mengucapkan terima kasih yang sebesarnya kepada semua pihak yang membantu dalam proses penyusunan makalah ini.

Penulis menyadari bahwa dalam penyusunan dan penulisan masih terdapat banyak kesalahan. Oleh karena itu penulis memohon maaf atas kesalahan dan ketidaksempurnaan terdapat dalam laporan ini. Penulis juga berharap adanya kritik serta saran dari pembaca apabila menemukan kesalahan dalam laporan ini.

Depok, December 10, 2022

Group A3

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Pada kesempatan pengerjaan proyek ini, kelompok A3 menyadari bahwa dengan perkembangan zaman yang dianalisis dari segi teknologi dan juga pola pikir manusia yang semakin maju. Dengan begitu, perkembangan ini dinilai dapat mempermudah masyarakat untuk menyelesaikan yang sebelumnya hanya manusia yang bisa selesaikan, sekarang selain manusia dengan bantuan bahasa pemrograman juga dapat menyelesaikan dengan efisien. bukti yang dapat diperoleh yaitu ada teknologi sistem digital yang digunakan manusia untuk menghasilkan output yang diinginkan dalam berbagai hal.

pada sistem digital yang kami design disini, kamu membuat sebuah microcontroller yagn mengimitasi sebuah controller yang lebih kecil dengan performa yang lebih cepat dari controller pada umumnya namun memiliki sisi positif dan juga negatifnya sendiri yang ikut beriringan dengannya. dalam makalah ini menjelaskan tentang peranfcangan sistem digital itu sendiri dengan studi perancangan mini controller ini sendiri dengan memanfaatkan hash.

brangkat dari keseluruhan itu sendiri maka kelomok kami memilih untuk menajdi salah satu untuk menjadi solusi dari permasalahan dan pengefisiensian segala kebutuhan manusia itu sendiri

1.2 PROJECT DESCRIPTION

Pada kesempatan pengerjaan proyek ini, kelompok A3 memanfaatkan ilmu yang sudah didapatkan dari modul-modul sebelumnya. kelompok kami membuat mini processor yang dimana didalamnya terdapat register yang berukuran 16x16 bit. selain itu juga kami juga menggunakan fitur ada operasi ADD, SUB, AND, OR, XOR, NOT, dan MOVE dengan nilai opcodenya masing-masing.

Pada program ini kami juga mengimplementasikan shifter dimana memiliki fitur dengan menggunakan bantuan ROR8, ROR4 dan juga SLL8. Tentunya dari ketiga fitur tersebut memiliki nilai opcodenya masing-masing. Kemudian, Hasher disini juga menggunakan mekanisme non linear looku table sehingga hashing disini akan jauh lebih cepat dibandingkan lainnya yang disebabkan oleh berjalannya hashing tersebut secara paralel

dan pada nantinya akan selesai pada clock yang sama. Dengan kata lain tidak membutuhkan state yang berbeda. Hanya saja pada microcontroller ini memiliki kekurangan dari segi keamanan. hal ini disebabkan oleh apabila key nya tersebar, maka siapapun dapat mengetahui isi data sebenarnya.

1.3 OBJECTIVES

Tujuan yang diharapkan dengan melakukan proyek ini:

1. Menyelesaikan permasalahan keamanan dengan menerapkan algoritma dalam Bahasa Pemrograman VHDL
2. Mengimplementasikan ROR8, ROR4 dan SLL8 dengan baik dan benar
3. mengimlementasikan operasi-operasi yang tersedia pada VHDL dalam perancangan sistem digital

1.4 ROLES AND RESPONSIBILITIES

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

Roles	Responsibilities	Person
Membuat laporan	Membuat laporan	Ryan, Najwa,Enricco, Aflah
Membuat presentasi	Membuat PPT	Ryan, Najwa,Enricco, Aflah
State diagram	Membuat state diagram sesuai rangkaian	Ryan
Melakukan simulasi Kode	Menjalankan simulasi pada Modelsim dan Quartus	Aflah
membuat code state	membuat kode VHDL didalam sebuah proyek ini	Ryan, Najwa,Enricco, Aflah
Membuat Block Diagram	Membuat block diagram sesuai rangkaian	Najwa

membuat read.me	membuat read.me markdown	Enricco

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

Software yang kami gunakan dalam percobaan ini antara lain:

- Visual Studio Code
- ModelSim
- Quartus Prime 21.1

2.2 IMPLEMENTATION

Pada percobaan kali ini, kelompok kami merangcang sebuah mini processor melalui kode VHDL. Dalam mini processor yang telah kami buat terdapat kode mini_proc yang merupakan Top Level dari seluruh rangkaian yang akan memanggil komponen lainnya. Pada perancangan mini processor kali ini kami juga menusun structural.vhd yang di dalamnya terdapat implementasi dan proses dari input register, register file yang berukuran 16x16 bit, serta logika kombinasional yang digambarkan pada diagram block dalam fig.1.

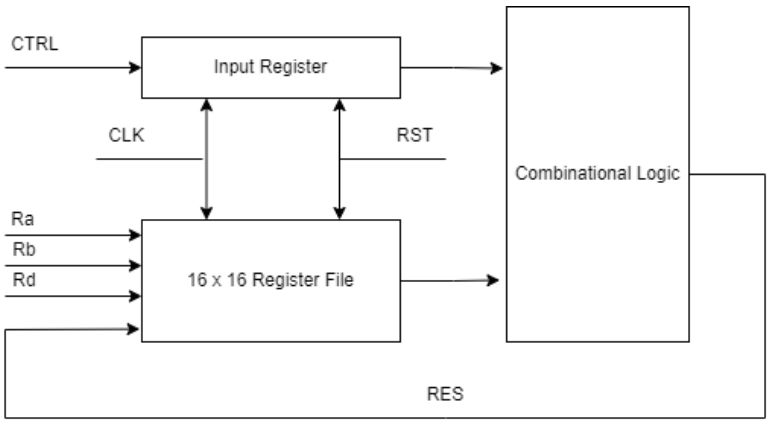

Fig 1. Block Diagram of structural

Dalam pemaparan block diagram diatas dapat diketahui bahwa terdapat input Ra, Rb, dan Rd yang akan diproses dalam 16x16 bit register bersamaan dengan hasil yang didapatkan pada proses dari combinational logic. Selanjutnya, input dari register juga terdapat CLK yang

berfungsi sebagai clock serta CTRL yang berfungsi sebagai control. Pada register ini juga dapat melakukan reset.

Dalam combinational logic sendiri, terdapat beberapa komponen yaitu non linear lookup unit, ALU, Shifter, MUX, dan juga Control logic. Hal ini dapat dilihat pada fig.2. Dimana terdapat input yaitu A_BUS dan B_BUS yang akan diproses pada non linear lookup, ALU, dan juga Shifter yang kemudian hasilnya akan di MUX kan. ALI memiliki fitur dengan operasi ADD, SUB, AND, OR, XOR, NOT, dan MOVE dengan nilai opcodenya masing-masing Shifter memiliki fitur dengan operasi ROR8, ROR4 serta SLL8 dengan nilai opcode nya masing-masing Hasher disini menggunakan mekanisme non linear lookup table sehingga hashingnya akan jauh lebih cepat karena dapat berjalan secara paralel dan selesai pada clock yang sama. Maka dari itu tidak membutuhkan state yang berbeda. Namun, hal ini memiliki kekurangan dari segi keamanan, karena apabila key nya tersebar, maka siapapun dapat mengetahui isi data sebenarnya.

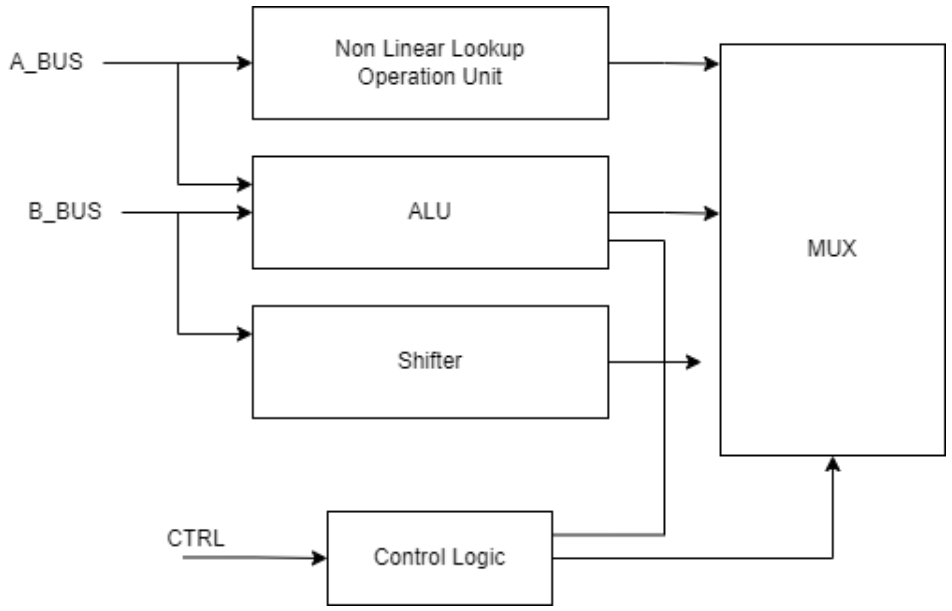

Fig 2. Block Diagram of the combinational logic

Dengan mengimplementasikan seluruh komponen yang telah kami buat. Selanjutnya kami melakukan simulasi melalui modelsim dan quartus prime. Dimana pada simulasi ini kami dapat mengamati simulasi yang dilakukan pada top level entity dan juga rangkaian ALU. Kami memilih simulasi pada kedua komponen tersebut dikarenakan keduanya sudah mencakup keseluruhan rangkaian kami. Untuk lebih lanjutnya hasil simulasi akan dijelaskan pada Chapter 3.

CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

Uji coba dilakukan terhadap sejumlah program VHDL untuk memeriksa apakah cara kerjanya telah sesuai dengan yang diinginkan. Pengujian pertama dilakukan menggunakan aplikasi Modelsim untuk melihat diagram waktu yang menunjukkan perubahan nilai input dan output dari masing-masing port dan signal seiring clock berdetak. Untuk mempermudah simulasi diagram waktu kami menggunakan file testbench dari masing-masing komponen yang ingin kami uji. Komponen-komponen VHDL yang kami uji menggunakan aplikasi Modelsim antara lain,

1. mini_proc.vhd (**Top Level**)
2. alu.vhd

Kami memilih untuk melakukan pengujian testbench pada komponen-komponen tersebut dengan beberapa alasan. Pertama, mini_proc sebagai Top Level dari seluruh komponen perlu di cek apakah sudah dapat memanggil komponen lain dengan baik. Kedua yaitu alu.vhd diuji karena memanggil 3 komponen lain yaitu adder, subtractor dan non_linear_lookup, sehingga menurut kami cukup banyak posibilitas nilai yang dapat dikeluarkan apabila diuji secara manual. Untuk komponen structural tidak kami lakukan pengujian testbench karena komponen ini memanggil komponen alu, yang dimana juga memanggil komponen lainnya, sehingga kami memutuskan untuk diuji secara langsung pada mini_proc saja. Seluruh komponen tersebut diuji dengan bantuan file testbench yang masing-masing memiliki nama sebagai berikut,

1. tb_mini_proc.vhd (**Top Level**)
2. tb_alu.vhd

Selain aplikasi Modelsim, pengujian juga dilakukan dengan bantuan aplikasi Quartus Prime dari intel yang membantu kami untuk melakukan sintesis terhadap program ke dalam skema rangkaian komponen juga *state machine*. Sehingga kami bisa membandingkan model state machine yang kami rancang di awal dengan hasil sintesis.

3.2 RESULT

Pada pengujian terhadap komponen alu didapatkan hasil sesuai dengan yang kami harapkan pada proses perancangan. Pada time diagram kami telah mencoba segala macam operasi yang kami sediakan di dalam komponen alu dengan masing-masing *opcode* nya yaitu, ADD (0000), SUB (0001), AND (0010), OR (0011), XOR (0100), NOT (0101), MOVE (0110). Time Diagram, Transcript dan Source Code disajikan pada Fig.3, Fig.4 dan Fig.5.

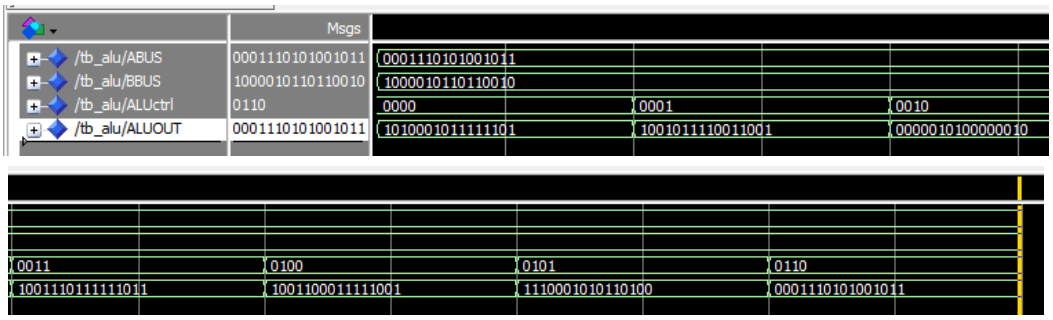

Fig 3. Time Diagram tb_alu.vhd

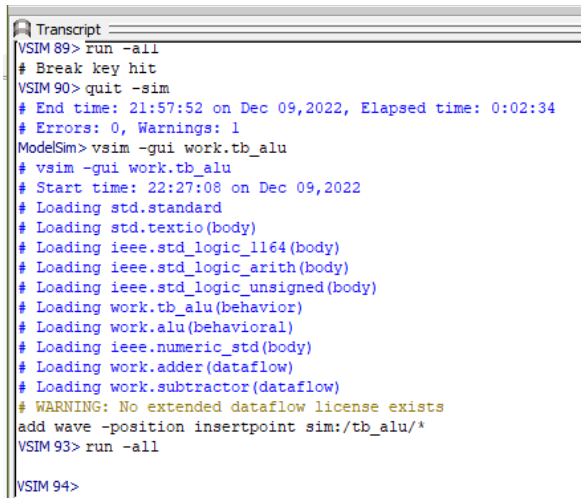

Fig 4. Transcript tb_alu.vhd

```
tb: process
begin
    ABUS <= x"1d4b";
    BBUS <= x"85b2";

    ALUctrl <= "0000";
    wait for 100 ns;
    assert (ALUOUT = "1010001011111101")
        report "Error at 0 Operation"
        severity Error;
```

```
    ALUctrl <= "0001";
    wait for 100 ns;
    assert (ALUOUT = "1001011110011001")
        report "Error at 1 Operation"
        severity Error;

    ALUctrl <= "0010";
    wait for 100 ns;
    assert (ALUOUT = "0000010100000010")
            report "Error at 2 Operation"
            severity Error;

    ALUctrl <= "0011";
    wait for 100 ns;
    assert (ALUOUT = "1001110111111011")
        report "Error at 3 Operation"
        severity Error;

    ALUctrl <= "0100";
    wait for 100 ns;
    assert (ALUOUT = "1001100011111001")
        report "Error at 4 Operation"
        severity Error;

    ALUctrl <= "0101";
    wait for 100 ns;
    assert (ALUOUT = "1110001010110100")
        report "Error at 5 Operation"
        severity Error;

    ALUctrl <= "0110";
    wait for 100 ns;
    assert (ALUOUT = "0001110101001011")
        report "Error at 6 Operation"
        severity Error;

    wait;
end process;
```

Fig 4. Source Code testbench process tb_alu.vhd

Pengujian terhadap komponen mini_proc menunjukkan hasil output sesuai dengan yang kami harapkan pada proses perancangan. Pada time diagram kami telah mencoba segala macam operasi yang kami sediakan di dalam komponen mini_proc termasuk juga yang terdapat pada komponen alu dengan masing-masing *opcode* nya yaitu,

- ADD (0000),
- SUB (0001),
- AND (0010),
- OR (0011),
- XOR (0100),
- NOT (0101),

- MOVE (0110),
- ROR8 (1000),
- ROR4(1001),
- SLL8 (1010),
- LUT (1011).

Seluruh operasi telah menghasilkan output sesuai dengan yang direncanakan. Time Diagram, Transcript dan Source Code disajikan pada Fig.5, Fig.6 dan Fig.7.

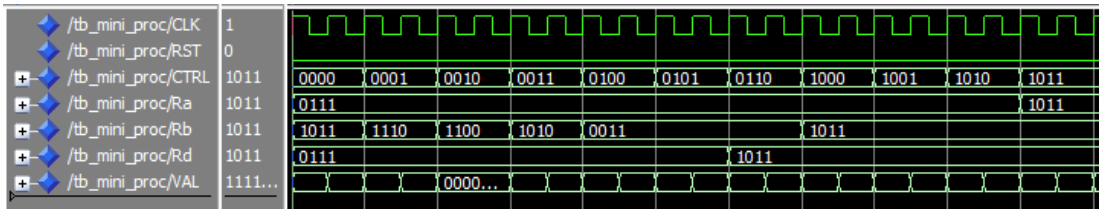

Fig 5. Time Diagram tb_mini_proc.vhd

```
ModelSim> vsim -gui work.tb_mini_proc
# vsim -gui work.tb_mini_proc
# Start time: 22:51:52 on Dec 09,2022
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.tb_mini_proc(behavior)
# Loading work.mini_proc(behavioral)
# Loading ieee.numeric_std(body)
# Loading work.register_file(behavioral)
# Loading work.structural(behavioral)
# Loading work.alu(behavioral)
# Loading work.adder(dataflow)
# Loading work.subtractor(dataflow)
# Loading work.shifter(behavioral)
# Loading work.non_linear_lookup(behavioral)
# WARNING: No extended dataflow license exists
add wave -position insertpoint sim:/tb_mini_proc/*
force -freeze sim:/tb_mini_proc/CLK 1 0, 0 {50 ps} -r 100
VSIM 98> run -all
```

Fig 6. Transcript tb_mini_proc.vhd

```
tb: process
begin

    RST <= '0'; -- Set State
    CTRL <= "0000";-- ADD
    Ra <= "0111"; -- R7
    Rb <= "1011"; -- R11
    Rd <= "0111"; -- R7
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "0001001000000000")
        report "Error at ADD Operation"
        severity Error;
```

```
    wait for clock_half_period;

    CTRL <= "0001";-- SUB
    Ra <= "0111"; -- R7
    Rb <= "1110"; -- R14
    Rd <= "0111"; -- R7
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "0001000100100000")
        report "Error at SUB Operation"
        severity Error;
    wait for clock_half_period;

    CTRL <= "0010";-- AND
    Ra <= "0111"; -- R7
    Rb <= "1100"; -- R12
    Rd <= "0111"; -- R7
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "0000000000000000")
       report "Error at AND Operation"
       severity Error;
    wait for clock_half_period;

    CTRL <= "0011";-- OR
    Ra <= "0111"; -- R7
    Rb <= "1010"; -- R10
    Rd <= "0111"; -- R7
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "0000000010100000")
        report "Error at OR Operation"
        severity Error;
    wait for clock_half_period;

    CTRL <= "0100";-- XOR
    Ra <= "0111"; -- R7
    Rb <= "0011"; -- R3
    Rd <= "0111"; -- R7
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "0000001110100000")
        report "Error at XOR Operation"
        severity Error;
    wait for clock_half_period;

    CTRL <= "0101";-- NOT
    Ra <= "0111"; -- R7
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "1111110001011111")
        report "Error at NOT Operation"
        severity Error;
    wait for clock_half_period;

    CTRL <= "0110";-- MOVE
    Ra <= "0111"; -- R7
    Rd <= "1011"; -- R11
    wait for clock_period;
    wait for clock_half_period;
```

```
    assert (VAL = "1111110001011111")
        report "Error at MOVE Operation"
        severity Error;
    wait for clock_half_period;

    CTRL <= "1000"; --ROR8
    Rb <= "1011"; --R11
    Rd <= "1011"; --R11
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "0101111111111100")
        report "Error at ROR8 Operation"
        severity Error;
    wait for clock_half_period;

    CTRL <= "1001"; --ROR4
    Rb <= "1011"; --R11
    Rd <= "1011"; --R11
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "1100010111111111")
        report "Error at ROR4 Operation"
        severity Error;
    wait for clock_half_period;

    CTRL <= "1010"; --SLL8
    Rb <= "1011"; --R11
    Rd <= "1011"; --R11
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "1111111100000000")
        report "Error at SLL8 Operation"
        severity Error;
    wait for clock_half_period;

    CTRL <= "1011";  --LUT
    Ra <= "1011"; --R11
    Rd <= "1011"; --R11
    wait for clock_period;
    wait for clock_half_period;
    assert (VAL = "1111111110101001")
        report "Error at LUT Operation"
        severity Error;
    wait for clock_half_period;
    wait;
end process;
```

Fig.7 Source Code testbench process tb_mini_proc.vhd

Terakhir, ujicoba dilakukan menggunakan aplikasi Quartus untuk melihat hasil sintesis skematik rangkaian dan juga *state machine*. Hasil kedua sintesis dan *source code process* dalam mini_proc dapat dilihat dalam Fig. 8, Fig.9 dan Fig.10.

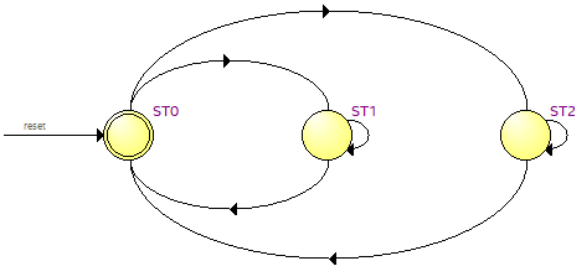

Fig.8 State Machine Synthesis mini_proc.vhd

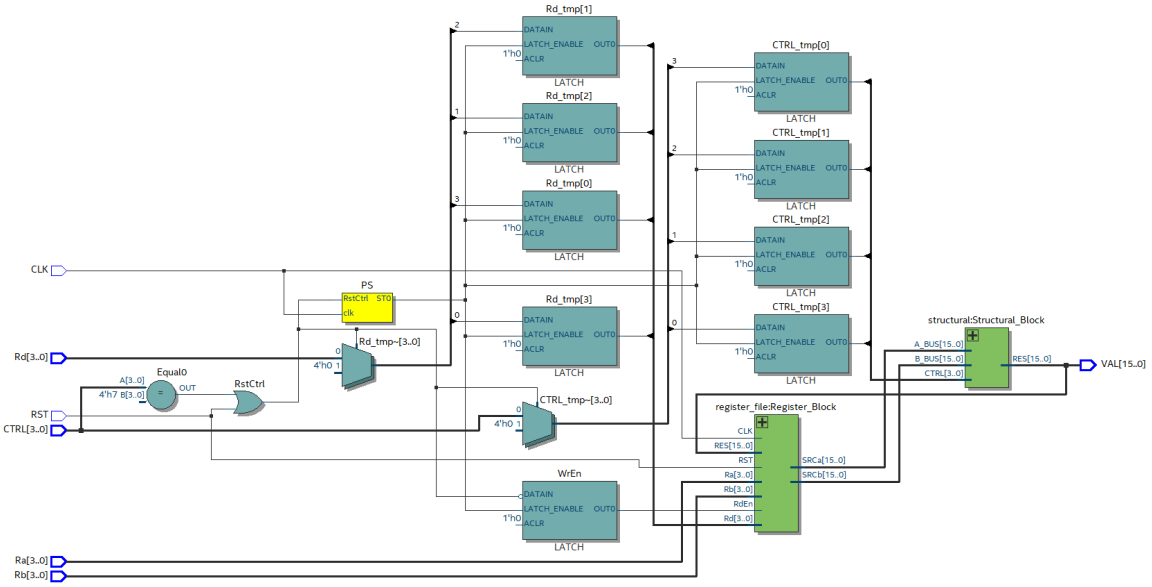

Fig.9 RTL Viewer mini_proc.vhd

```
sync_proc : process(CLK)
begin
    if (rising_edge(CLK)) then
        PS <= NS;
    end if;
end process;

comb_proc : process(PS, RST, RstCtrl, CTRL)
begin
    if (RST = '1' OR CTRL = "0111") then -- If either of them TRUE,
mini_proc is disabled
        RstCtrl <= '1';
    else
        RstCtrl <= '0';
    end if;

    case PS is
    when ST0 =>
        if (RstCtrl = '1') then
            NS <= ST2;
            CTRL_tmp <= x"0"; -- Zero filled
```

```
            Rd_tmp <= x"0"; -- Zero filled
            WrEn <= '0'; -- Register CANT write to Rd
        else
            NS <= ST1;
            CTRL_tmp <= CTRL;
            Rd_tmp <= Rd;
            WrEn <= '1'; -- Register CAN write to Rd
        end if;
    when ST1 =>
        if (RstCtrl = '0') then
            NS <= ST0;
        else
            NS <= ST1;
        end if;
    when ST2 =>
        if (RstCtrl = '1') then
            NS <= ST0;
        else
            NS <= ST2;
        end if;
    end case;
end process;
```

Fig.10 Source Code Process mini_proc.vhd

3.3 ANALYSIS

Terlihat pada time diagram milik tb_alu.vhd, hasil komputasi dapat segera muncul secara paralel. Penggunaan FPGA disini terlihat membantu meningkatkan performa kecepatan dalam komputasi dibandingkan harus menggunakan sistem berbasis pemrograman bukan rangkaian gerbang. Dengan karakteristik FPGA dan VHDL yang melakukan komputasi secara paralel, hasil output dapat keluar jauh lebih cepat.

Jika dibandingkan dengan milik tb_alu.vhd, hasil komputasi pada tb_mini_proc muncul pada *clock pulse* berikutnya, tidak serta merta langsung menghasilkan output yang sesuai sesaat setelah penginputan. Hal ini terjadi karena mini_proc bekerja dengan karakteristik FSM atau *Finite State Machine*. Mini_proc baru melakukan komputasi dan penulisan pada register ketika pindah ke *state* berikutnya. Sehingga hasil output yang sesuai baru dapat diterima pada *clock pulse* berikutnya.

Jika berkaca pada hasil pengujian menggunakan modelsim, terlihat bahwa tidak ada error pada kedua transcript. hal tersebut mengindikasikan bahwa proses komputasi telah berjalan sesuai harapan dan mengeluarkan output dengan seperti yang diekspektasikan.

CHAPTER 4

CONCLUSION

Kelompok kami merancang sebuah Mini processor yang memiliki mode komputasi dan menulis serta komputasi tanpa menulis. Didalamnya terdapat proses dari input register, 16x16 register file, dan logika kombinasional. Dalam logika kombinasional sendiri terdapat beberapa komponen seperti ALU, Shifter, Hasher, MUX, dan juga Control logic. ALU memiliki fitur operasi ADD, SUB, AND, OR, XOR, NOT, dan MOVE dengan nilai opcodenya masing-masing, Shifter memiliki fitur operasi ROR8, ROR4, dan SLL8 dengan nilai opcodenya masing-masing. Hasher menggunakan mekanisme non linear lookup table.

Kode VHDL kita uji dengan simulasi menggunakan modelsim dan quartus prime, dengan menguji top level entity dan juga ALU. Simulasi untuk testbench top level dan juga ALU mendapatkan hasil output sesuai yang kami inginkan.

Dapat disimpulkan bahwa pada ALU, hasil komputasi dapat muncul secara paralel yang menyebabkan output dapat keluar jauh lebih cepat, tetapi memiliki kekurangan di segi kemanan dimana apabila key tersebar maka siapapun dapat mengetahui isi datanya. Pada top level, hasil komputasi muncul pada *clock pulse* berikutnya sehingga output didapatkan lebih lama. Hal ini terjadi karena top level bekerja dengan karakteristik *Finite State Machine.*

REFERENCES

[1] C. IBM, "Hash lookup - IBM Documentation," Ibm, Apr. 4, 2022. https://www.ibm.com/docs/en/iotdm/11.3?topic=lf-hash-lookup-function(accessed Jan. 1, 1970).

[2] Ronald J. Hayne. Hayne, "AN INSTRUCTIONAL PROCESSOR DESIGN USING VHDL AND AN FPGA," .

APPENDICES

Appendix A: Project Schematic

Put your final project latest schematic here

Appendix B: Documentation

Put the documentation (photos) during the making of the project