SYNTHESIZABLE RISC-V (RV32I) MICROPROCESSOR BAGIAN I: INSTRUCTION SET, REGISTER, DAN MEMORY

William Anthony (13223048)
Asisten : Abraham Pratomo (23223071)
Tanggal Percobaan : 05/11/2025
EL3011 Arsitektur Sistem Komputer
Laboratorium Sinyal dan Sistem – Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

***Abstrak*—Praktikum ini berfokus pada pemahaman dan implementasi arsitektur mikroprosesor RISC-V (RV32I) dalam konfigurasi single-cycle. Percobaan mencakup eksplorasi datapath eksekusi instruksi, pengenalan instruction set inti RV32I, penulisan, dan simulasi program assembly sederhana menggunakan simulator/assembler RISC-V. Perancangan instruction memory, data memory, dan register file RISC-V menggunakan Verilog HDL. Seluruh tugas percobaan berhasil disintesis dan disimulasikan menggunakan Intel® Quartus® Prime, GTKWave dan ModelSim, sehingga menunjukkan fungsionalitas sesuai spesifikasi dan performa timing yang dapat diterima. Analisis mencakup perbandingan implementasi memori manual dengan penggunaan Altera® MegaFunction ALTSYNCRAM yang efisien, serta validasi disiplin clocking untuk menghindari konflik baca/tulis. Nilai register x0 yang selalu bernilai nol terverifikasi.**

***Kata Kunci*—RISC-V, RV32I, Microprosesor, Memori, Verilog HDL**

I. Pendahuluan

Praktikum ini berfokus pada pengenalan dan implementasi dasar mikroprosesor RISC-V RV32I. Tujuannya adalah agar praktikan memahami konsep arsitektur komputer, khususnya arsitektur RISC-V yang terbuka dan modular. Praktikan akan mempelajari cara kerja datapath single-cycle, memahami berbagai format instruksi (R, I, S, B, U, J), dan mempraktikkan penulisan program assembly sederhana. Melalui simulasi dengan Venus, praktikan dapat mengamati eksekusi instruksi dan memantau perubahan pada register serta memori. Bagian inti adalah perancangan komponen-komponen mikroprosesor seperti instruction memory, data memory, dan register file menggunakan Verilog HDL. Hasil yang diharapkan adalah implementasi fungsional dari komponen-komponen tersebut yang telah disimulasikan secara fungsional dan timing menggunakan Intel® Quartus® Prime dan ModelSim.

II. Landasan Teoretis

A. Bahasa Verilog HDL

Verilog HDL adalah bahasa deskripsi perangkat keras (hardware description language) yang digunakan untuk mendesain rangkaian digital, baik untuk FPGA maupun IC. Dalam Verilog, sinyal dapat direpresentasikan sebagai wire untuk koneksi kombinasional atau reg untuk penyimpanan atau variabel yang di-assign dalam blok sekuensial atau kombinasional. Penting untuk dicatat bahwa Verilog bersifat case-sensitive. Simbol-simbol seperti 0 (Low), 1 (High), x (Unknown/Undefined), dan z (High-impedance) digunakan untuk merepresentasikan kondisi sinyal.

Verilog HDL adalah bahasa deskripsi perangkat keras (hardware description language) untuk mendesain rangkaian digital pada FPGA maupun IC. Dalam Verilog, sinyal direpresentasikan sebagai wire (koneksi kombinatorial) dan reg (penyimpanan/variabel yang di-assign di blok sekuensial atau kombinatorial). Verilog bersifat case-sensitive (huruf besar/kecil dibedakan). Implementasi fungsi dapat dilakukan dengan continuous assignment (assign) untuk logika kombinasional atau blok always untuk logika kombinasional atau sekuensial.

Setiap entitas desain disimpan dalam file Verilog HDL terpisah dengan nama yang sesuai. Desain ditulis sebagai module yang dapat diinstansiasi oleh module lain untuk membentuk rangkaian yang lebih besar. Setiap desain pada Verilog ditulis sebagai module yang dapat diinstansiasi oleh module lain untuk membentuk rangkaian yang lebih besar (top-level module). Praktiknya, rangkaian besar dipecah menjadi komponen-komponen kecil (misalnya multiplexer, adder, flip-flop), masing-masing diuji (simulasi fungsional/timing), lalu digabung kembali melalui instansiasi

modul dan pengkabelan port input/output.

Terdapat tiga jenis concurrent signal assignment (CSA):

I. Simple CSA. Assignment sinyal dilakukan dengan ekspresi logika biasa. Hasil implementasi Simple CSA akan berupa gerbang logika biasa.

II. Conditional CSA. Assignment sinyal dilakukan dengan construct WHEN-ELSE. Hasil implementasi Conditional CSA akan berupa kumpulan 2-to-1 multiplexer yang disusun secara bertahap dengan boolean_expr sebagai selektor dan value_expr sebagai nilai sinyal yang dapat dipilih.

III. Selected CSA. Assignment sinyal dilakukan dengan construct WITH-SELECT. Hasil implementasi Selected CSA akan berupa satu buah n-to-1 multiplexer dengan select_expression sebagai selektor dan value_expr_3 sebagai nilai sinyal yang dapat dipilih.

Dalam Verilog, elemen memori direalisasikan juga dengan blok always. D Latch bersifat level-sensitive (keluaran mengikuti masukan saat enable aktif) dan tidak direkomendasikan pada FPGA karena menyebabkan latch inference yang merugikan analisis timing. D Flip-Flop (DFF) bersifat edge-triggered dan menjadi dasar implementasi register. Register adalah sekumpulan DFF yang menyimpan suatu vektor data dan banyak digunakan untuk rangkaian sekuensial seperti (FSM) finite state machine maupun register file. Untuk DFF, gunakan non-blocking assignment (<=) di blok always @(posedge clk); untuk logika kombinatorial gunakan always @(*) dengan blocking assignment (=).

Contoh Implementasi Mux dalam Verilog adalah pada gambar 1 dalam lampiran. Akan membuat seperti Fig. 1.

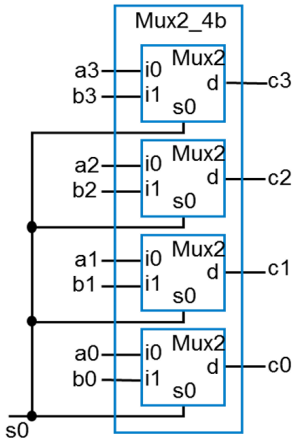


Fig. 1. Mux2_4b di Verilog

B. *Altera® Quartus® II*

Intel® Quartus® Prime (sebelumnya Altera® Quartus® II) adalah perangkat lunak untuk desain FPGA. Proses desain dimulai dengan pembuatan proyek, pemilihan device (meskipun untuk praktikum ini hanya simulasi), dan kompilasi. Kompilasi dapat berupa full compilation (termasuk Analysis & Synthesis, Fitter, dan Assembler) untuk analisis timing, atau hanya Analysis & Synthesis untuk simulasi fungsional. Untuk simulasi, terutama versi Quartus yang lebih baru, disarankan menggunakan Mentor Graphics® ModelSim®.

Fig. 2. Altera® Quartus® II

Untuk menggunakan Altera® Quartus® II, pertama membuat project terlebih dahulu. Untuk membuat project, gunakan new project wizard kemudian ikuti petunjuk-petunjuk yang ada. Beri lokasi dan nama project yang diinginkan. Pilih dokumen-dokumen yang akan dimasukkan ke dalam project (kita dapat melewati langkah ini terlebih dahulu). Kemudian pilih device yang akan digunakan. Pilih FPGA dengan spesifikasi tertinggi baik untuk Altera® Cyclone™ maupun Altera® Stratix™. Setelah project dibuat, kita dapat mulai bekerja di dalamnya. Untuk melakukan simulasi, harus dilakukan kompilasi terhadap project yang kita buat. Kompilasi bisa dilakukan secara kompilasi penuh atau Analysis & Synthesis saja. Kompilasi penuh akan memakan waktu yang lebih lama karena semua proses meliputi Analysis & Synthesis, Fitter, dan Assembler akan dilakukan. Kompilasi penuh ini akan memberi kita gambaran terutama dari sisi timing analysis. Sedangkan dengan Analysis & Synthesis, kita telah mendapat rangkaian yang kita buat dan dapat dilakukan simulasi fungsional.

C. *RISC-V*

RISC-V merupakan arsitektur set instruksi (instruction set architecture, ISA) terbuka yang dikelola oleh RISC V International. RISC-V yang digunakan adalah basis 32-bit (RV32I, base integer ISA).
RISC-V banyak dipakai untuk pembelajaran arsitektur komputer karena spesifikasi intinya ringkas, konsisten, dan mudah diimplementasikan; di dunia nyata ia hadir dari mikrokontroler berdaya rendah hingga SoC tertanam.

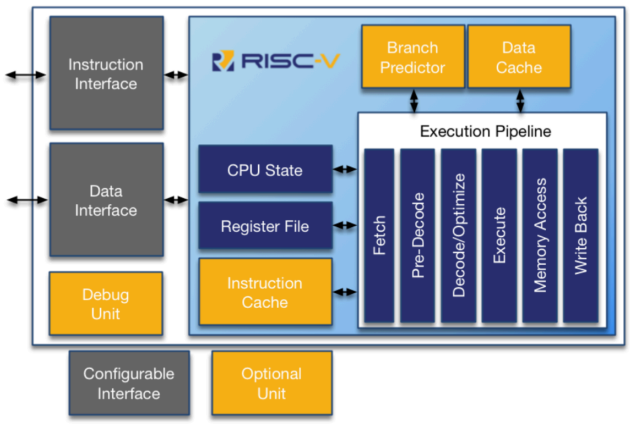


Fig. 3. RISC-V Diagram.

Terdapat lima tahap yang dilakukan ketika mikroprosesor RISC-V melakukan eksekusi suatu instruksi. Kelima tahap tersebut adalah sebagai berikut.

Pertama, Instruction Fetch (IF). Tahap instruction fetch berfungsi mengatur aliran instruksi yang akan diolah pada tahap berikutnya. Instruksi yang sedang dijalankan merupakan instruksi yang berasal dan disimpan dari memory. Pada arsitektur ini, memory dipisahkan menjadi dua bagian yaitu instruction memory yang berfungsi menyimpan instruksi-instruksi yang akan dieksekusi dan data memory yang berfungsi untuk menyimpan data untuk menghindari structural hazard. Dengan demikian, arsitektur ini menganut Harvard Architecture.

Kedua, Instruction Decode (ID). Tahap berikutnya, instruksi yang telah diambil (fetched) dari instruction memory berpindah ke tahap instruction decode. Pada tahap ini, instruksi dengan lebar 32-bit akan dipecah sesuai format instruksi yang digunakan. Penjelasan mengenai decoding instruksi ini dapat dilihat pada bagian selanjutnya, tetapi secara umum akan dipecah menjadi bagian alamat register destinasi (rd), alamat register sumber/source (rs1 dan rs2), dan opcode atau kode yang menandakan operasi yang akan dijalankan (misalkan, instruksi add dan sub memiliki opcode yang sama karena keduanya instruksi tipe-R, dijelaskan berikutnya).

Ketiga, Execute / Address Calculation (EX). Tahap ini merupakan tahap sebagian besar operasi aritmatika dan logika pada arithmetic and logical unit (ALU) dilakukan, misalkan operasi penjumlahan untuk instruksi add. Pada tahap ini juga terdapat tempat untuk meneruskan alamat register kembali ke tahap instruction decode sebagai deteksi hazard.

Keempat, Data Memory (MEM). Pada tahap ini, data disimpan dan/atau diambil dari data memory. Data memory hanya dapat disimpan atau dibaca jika ada sinyal MemRead dan/atau MemWrite yang sesuai sehingga operasi baca dan/atau tulis pada data memory dapat dilakukan.

Kelima, Write Back (WB). Tahap terakhir ini digunakan untuk mengalirkan data dari data memory atau hasil perhitungan arithmetic and logical unit (ALU) ke register untuk disimpan, guna dipakai untuk instruksi-instruksi berikutnya. Dalam praktikum ini kita akan mengimplementasikan mikroprosesor RISC-V (RV32I) sederhana.

Desain yang dibuat tanpa pipeline (non-pipelined), sehingga setiap instruksi selesai dalam satu siklus clock. Dengan demikian, kita membangun Single-Cycle RISC-V menggunakan Verilog HDL yang dapat disintesis. Diagram arsitektur prosesor single-cycle RV32I yang digunakan pada praktikum ditunjukkan pada gambar berikut.

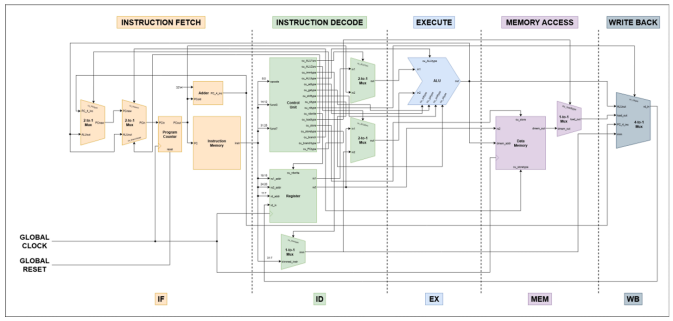


Fig. 4. Tahap-tahap Mikroprosesor RISC-V ketika melakukan eksekusi suatu instruksi.

D. Harvard Architecture

Arsitektur komputer Harvard adalah model arsitektur komputer dengan memori instruksi dan memori data disimpan pada dua memori fisik terpisah, yang masing-masing memiliki jalur komunikasi sendiri dengan unit pemrosesan pusat (CPU). Pada arsitektur komputer Harvard, terdapat dua memori fisik terpisah, yaitu memori instruksi dan memori data. Memori instruksi digunakan untuk menyimpan program dan instruksi, sedangkan memori data digunakan untuk menyimpan data yang sedang diproses.

Arsitektur komputer Harvard terapkan instruksi-instruksi pada saat yang sama, karena unit pemrosesan pusat (CPU) dapat mengakses memori instruksi dan memori data secara bersamaan. Hal ini membuat komputer dengan arsitektur Harvard untuk melakukan pemrosesan data dengan kecepatan yang lebih tinggi.

Arsitektur komputer Harvard lebih aman dari serangan seperti buffer overflow. Hal ini karena, bila serangan berhasil memodifikasi data pada memori data, serangan tersebut tidak dapat mengubah instruksi pada memori instruksi.Arsitektur komputer Harvard memiliki keterbatasan dalam kapasitas memori, karena terdapat pembatasan dalam jumlah memori instruksi dan data yang dapat digunakan. Contoh penggunaan arsitektur komputer Harvard meliputi prosesor Harvard yang banyak digunakan pada mikrokontroler dan beberapa sistem komputer tertentu seperti DEC PDP-8 dan PDP-11. Kelebihan arsitektur komputer Harvard adalah kecepatan pemrosesan data yang lebih tinggi, sementara kekurangannya adalah keterbatasan dalam kapasitas memori dan biaya yang lebih

mahal dalam membangun sistem komputer dengan arsitektur ini.

E. Von Neumann Architecture

Arsitektur komputer Von Neumann adalah model arsitektur komputer yang menggunakan satu memori fisik untuk menyimpan program, instruksi, dan data yang sedang diproses. Pada arsitektur komputer Von Neumann, terdapat satu memori fisik yang digunakan untuk menyimpan program, instruksi, dan data yang sedang diproses. Hal ini membuat pengunaan memori yang lebih efisien dan lebih mudah diimplementasikan.

Arsitektur komputer Von Neumann melakukan pemrosesan instruksi secara sekuensial, yang berarti satu instruksi diproses pada satu waktu.Arsitektur komputer Von Neumann memiliki kecepatan pemrosesan data yang relatif lebih lambat daripada arsitektur Harvard, karena unit pemrosesan pusat (CPU) hanya dapat mengakses satu memori pada suatu waktu. Arsitektur komputer Von Neumann rentan terhadap serangan seperti buffer overflow, yang dapat memodifikasi instruksi pada memori dan menyebabkan komputer melakukan operasi yang tidak diinginkan. Sebagian besar sistem komputer modern menggunakan arsitektur komputer Von Neumann, termasuk komputer pribadi, server, dan sistem komputer lainnya.

Kelebihan arsitektur komputer Von Neumann adalah penggunaan memori yang lebih efisien dan biaya yang lebih murah dalam membangun sistem komputer. Kekurangan arsitektur ini adalah kecepatan pemrosesan data yang lebih lambat dan rentan terhadap serangan keamanan.

Arsitektur komputer Von Neumann berbeda dengan arsitektur Harvard dalam hal penggunaan memori dan kecepatan pemrosesan data. Arsitektur Von Neumann menggunakan satu memori fisik untuk semua data dan instruksi, sementara arsitektur Harvard menggunakan dua memori fisik terpisah. Ini membuat arsitektur Harvard lebih cepat dalam pemrosesan data, tetapi membatasi kapasitas memori dan memerlukan biaya yang lebih mahal dalam membangun sistem komputer.

F. Instruction Set dan Register Mikroprosesor RISC-V RV32I

RISC-V RV32I memiliki lebar instruksi 32-bit. Format atau tipe dasarnya ada enam: R, I, S, B, U, J. Komponen dari format dasar instruksi tersebut dijelaskan pada tabel selanjutnya.

TABLE I RISC-V DATA

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	funct7							rs2					rs1					funct3			rd					opcode						
I	imm												rs1					funct3			rd					opcode						
S	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
B	imm[12] imm[10:5]							rs2					rs1					funct3			imm[4:1] imm[11]					opcode						
U	imm[31:25]							imm[24:20]					imm[19:15]					imm[14:12]			rd					opcode						
J	imm[20] imm[10:5]							imm[4:1] imm[11]					imm[19:15]					imm[14:12]			rd					opcode						

Untuk keterangan masing-masing *field*, ada di sini :

Komponen	Keterangan
`opcode [6:0]`	Menentukan kelas/format instruksi dan keluarga operasinya.
`rd/rs1/rs2`	Indeks register tujuan (rd) dan sumber (rs1, rs2).
`funct3/funct7`	Sub-opcode untuk membedakan varian operasi dalam satu opcode.
`immediate`	Konstanta bertanda; tata letak tergantung format (PC-relative untuk B/J, upper-immediate untuk U).

III. Hasil dan Analisis

Dari melaksanakan tugas 1, tugas 2, tugas 3, dan tugas 4 dengan lengkap (tanpa iterasi berlebihan, memanfaatkan Timing Analyzer dan Fitter Analyzer pada Quartus II). dapat diamati bahwa dengan prosesor single-cycle. Semua tahap eksekusi instruksi dalam satu siklus clock tunggal. Lima tahap dasar pipeline (Instruction Fetch, Instruction Decode, Execute, Memory Access, Write Back) terjadi. Dalam single-cycle, semua logika untuk setiap tahap harus diselesaikan dalam durasi satu clock period. Instruction Fetch mengambil instruksi. Instruction Decode menguraikan instruksi dan membaca register. Execute melakukan operasi ALU atau perhitungan alamat. Memory Access membaca atau menulis data. Write Back menyimpan hasil ke register. Semua ini harus terjadi secara berurutan dalam satu tick clock. Hasilnya, ini berbeda dengan prosesor pipelined di mana setiap tahap memiliki latensi satu siklus clock sendiri dan instruksi yang berbeda berada di tahap pipeline yang berbeda secara bersamaan.

Dari melakukan semua percobaan tugas pada praktikum modul ini, dapat diambil insight bahwa perbedaan fundamental terletak pada bagaimana instruksi diproses. Umumnya, prosesor RISC-V generik yang berpipa 5-tahap membagi eksekusi instruksi menjadi lima tahapan yang berjalan secara bersamaan (concurrently) untuk instruksi-instruksi yang berbeda. Setiap tahap mengambil satu siklus clock. Maka, banyak instruksi dapat "dalam penerbangan" secara bersamaan, sehingga menghasilkan throughput yang lebih tinggi. Sebaliknya, implementasi Single-Cycle RV32I pada praktikum ini menyelesaikan seluruh instruksi dari awal (fetch) hingga akhir (write back) dalam satu siklus clock penuh. Konsekuensinya, periode clock harus cukup panjang untuk mengakomodasi jalur terpanjang di antara semua operasi yang bisa dibuat, seringkali jalur load dari memori. Sehingga ini membatasi frekuensi operasi dan menyebabkan throughput yang lebih rendah dibandingkan desain pipelined. Keuntungannya adalah kesederhanaan desain dan tidak adanya hazard pipeline yang perlu dikelola.

Dari simulasi, proses decode instruksi RISC-V RV32I melibatkan pemisahan instruksi 32-bit ke dalam berbagai field berdasarkan formatnya (R, I, S, B, U, J) untuk mengidentifikasi jenis operasi dan operand yang terlibat.

Opcode[6:0] adalah field 7-bit yang selalu berada pada bit 0-6. Ini merupakan penentu utama format instruksi serta keluarga operasi, misalnya aritmetika register-register, aritmetika immediate, load, store, branch, atau jump. Rd[11:7] adalah register destination, field 5-bit yang menunjukkan register tujuan tempat hasil operasi akan disimpan. Rs1[19:15] dan rs2[24:20] adalah register source 1 dan register source 2, masing-masing field 5-bit yang menunjukkan register sumber. Funct3[14:12] adalah function 3, field 3-bit yang bersama dengan opcode dan atau funct7 membantu membedakan varian operasi dalam satu keluarga opcode. Funct7[31:25] adalah function 7, field 7-bit yang digunakan terutama untuk membedakan operasi tipe-R yang berbagi opcode dan funct3. Immediate adalah nilai konstanta yang tertanam langsung dalam instruksi.

Tata letaknya bervariasi tergantung format. Tipe I menggunakan immediate 12-bit (imm[31:20]) untuk operasi aritmatika immediate, load, dan JALR. Tipe S menggunakan immediate 12-bit yang dibagi menjadi imm[11:5] pada bit 31:25 dan imm[4:0] pada bit 11:7 untuk store. Tipe B menggunakan immediate 13-bit untuk instruksi branch. Bit-bitnya tersebar: imm[12] pada bit 31, imm[10:5] pada bit 30:25, imm[4:1] pada bit 11:8, dan imm[11] pada bit 7, membentuk nilai offset relatif PC. Tipe U menggunakan immediate 20-bit (imm[31:12]) untuk operasi upper immediate seperti LUI dan AUIPC. Tipe J menggunakan immediate 21-bit yang digunakan untuk instruksi jump. Bit-bitnya juga tersebar: imm[20] pada bit 31, imm[10:1] pada bit 30:21 yang disusun ulang, imm[11] pada bit 20, dan imm[19:12] pada bit 19:12, membentuk nilai offset relatif PC yang lebih besar.

A. Tugas I : Perancangan Instruction Memory (RV32I)

Instruction memory (instr_rom_rv32i.v) dirancang sebagai ROM berukuran 32x32-bit yang digunakan untuk menyimpan instruksi program RISC-V. Memori ini menerima alamat berupa Program Counter (PC) 32-bit, Pengindeksan dilakukan berdasarkan word address menggunakan potongan bit PC[6:2], karena setiap instruksi RISC-V memiliki panjang tetap 32-bit dan selalu word-aligned. Dengan demikian, dua bit terbawah (PC[1:0]) diabaikan. Pada kondisi reset, keluaran memori diatur ke instruksi NOP (No Operation) dengan nilai 0x00000013 untuk memastikan sistem tetap stabil walaupun belum ada instruksi aktif yang dijalankan.

Implementasi instruction memory dilakukan menggunakan blok initial di Verilog untuk inisialisasi konten memori secara internal. Kode tersebut mengisi 32 word memori dengan instruksi NOP sebagai nilai default, kemudian menimpa empat lokasi pertama (mem[0] hingga mem[3]) dengan machine code dari instruksi addi x1,x0,5, addi x2,x0,7, add x3,x1,x2, dan sw x3,0(x0). Pembacaan memori dilakukan secara sinkron pada posedge clock, dan saat sinyal reset aktif, keluaran INSTR langsung diatur ke 32'h00000013 (NOP RV32I).

Hasil kompilasi menggunakan Intel® Quartus® Prime (dengan asumsi FPGA Stratix II) menunjukkan status Full Compilation Successful dengan penggunaan sumber daya FPGA yang sangat kecil, kurang dari 1% dari total ALUT dan register yang tersedia. Berdasarkan laporan Fitter dan Timing Analyzer, tidak terdapat jalur timing yang gagal. Ini mengindikasikan bahwa desain memenuhi semua batasan timing yang diberikan dan dapat beroperasi dengan stabil pada frekuensi clock yang ditentukan. Simulasi fungsional menunjukkan bahwa setiap perubahan alamat (ADDR) pada input memori menghasilkan keluaran instruksi (INSTR) yang sesuai dengan isi memori yang telah diinisialisasi. Misalnya, alamat 0x00000000 hingga 0x0000002C menampilkan instruksi hasil kompilasi dari assembler RISC-V. Perubahan nilai INSTR terjadi tepat setelah tepi naik clock berikutnya, yang mengonfirmasi bahwa pembacaan sinkron telah berjalan dengan benar.

Terlebih dari pengamatan pada timing, analisis timing menunjukkan bahwa waktu tCO (clock-to-output delay) untuk komponen ini adalah 6.486 ns, yang menggambarkan performa baik untuk operasi pembacaan instruksi. Delay read dari perubahan alamat hingga data instruksi yang valid muncul pada keluaran mencapai 16.671 ns, tunjukkan waktu propagasi internal pada blok logika yang membentuk memori instruksi.

B. Tugas 2 : Perancangan Instruction Memory dengan Altera® MegaFunction ALTSYNCRAM

Implementasi instruction memory pada tugas ini memanfaatkan Altera® MegaFunction ALTSYNCRAM (instr_rom_rv32i.v) yang menerapkan perancangan memori sinkron. Hal ini diterapkan dengan inisialisasi file eksternal .mif (memory initialization file). Memori ini dikonfigurasi sebagai ROM berukuran 32x32-bit, menerima alamat Program Counter (PC) 32-bit, dan menggunakan PC[6:2] sebagai word index untuk pengaksesan memori, mirip dengan Tugas 1. Konfigurasi ALTSYNCRAM mencakup operation_mode("ROM"), width_a(32), widthad_a(5) untuk 32 word, dan init_file("imemory_rv32i.mif") untuk memuat konten program.

Penggunaan ALTSYNCRAM ini memberikan keuntungan signifikan dalam kemudahan desain dan fleksibilitas. Dengan inisialisasi melalui file .mif, konten program dapat diubah atau diperbarui tanpa memerlukan kompilasi ulang seluruh kode Verilog, yang mempercepat proses iterasi pengembangan. File imemory_rv32i.mif berisi machine code instruksi addi x1,x0,5, addi x2,x0,7, add x3,x1,x2, dan sw x3,0(x0) pada alamat 00 hingga 03, dengan sisa alamat diinisialisasi sebagai NOP (0x00000013).

Hasil kompilasi desain menggunakan ALTSYNCRAM (seperti yang ditunjukkan pada "Flow Status" yang Successful

dan "Logic utilization" 0%) menunjukkan optimasi yang sangat baik. Penggunaan block RAM khusus (Total block memory bits 1,024 / 419,328 < 1%) pada FPGA Stratix II oleh MegaFunction ini secara signifikan lebih efisien dibandingkan implementasi memori "manual" berbasis array reg (RTL). Kondisi timing requirements yang Met dan tidak adanya failed paths (0) pada "Timing Analyzer Summary" sehingga desain bekerja dengan stabil.

Dalam simulasi fungsional (ditunjukkan oleh waveform), keluaran INSTR berubah sesuai dengan alamat (PC) yang dimasukkan dan konten yang ada di imemory_rv32i.mif. Misalnya, pada alamat 0x00000000 hingga 0x0000002C, instruksi yang dimuat dari .mif terlihat pada sinyal INSTR. Perubahan instruksi pada keluaran juga teramati terjadi secara sinkron setelah rising edge clock, menunjukkan bahwa pembacaan memori berjalan dengan akurat dan sesuai dengan instruksi yang dimuat dari file .mif.

Analisis timing dari desain ini menunjukkan waktu delay penting. Worst-case tco (waktu dari clock hingga keluaran) adalah 8.983 ns, dan delay read dari input alamat hingga data instruksi yang valid muncul pada keluaran adalah 18.254 ns. Delay ini mencakup waktu propagasi internal pada block RAM dan logika wrapper. Dari timing analysis terdapat delay, performa timing yang tertera (Met timing requirements: Yes) menunjukkan bahwa desain ini memenuhi batasan waktu yang ditetapkan dan berfungsi dengan baik dalam konteks arsitektur sinkron FPGA.

C. Tugas 3 : Perancangan Data Memory dengan Altera® MegaFunction ALTSYNCRAM

Pada percobaan ini, perancangan data memory dilakukan menggunakan Altera® MegaFunction ALTSYNCRAM, yang berfungsi sebagai memori sinkron dengan kemampuan baca dan tulis (RAM). Modul data_mem_rv32i diimplementasikan dengan ukuran 32x32-bit dan mendukung operasi load (baca) serta store (tulis) terhadap data berukuran word, half-word, maupun byte. Penggunaan parameter operation_mode("SINGLE_PORT"), width_a(32), dan widthad_a(5) membuat memori dapat diakses dengan alamat 5-bit, sementara pengaturan outdata_reg_a("UNREGISTERED") sehingga output data berubah secara langsung setelah alamat berubah tanpa menunggu clock edge berikutnya.

Pada level desain, sinyal be (byte enable) dan wr_data_aligned berperan penting untuk memastikan data ditulis ke bagian word yang tepat, terutama untuk instruksi store byte (SB) dan store half-word (SH). Mekanisme tersebut membuat saat operasi tulis tidak word-aligned, data tetap tersimpan pada posisi yang benar dalam word 32-bit. Sementara itu, sinyal clk_n = ~clock diterapkan untuk membuat penulisan data terjadi pada negative edge clock, sesuai dengan implementasi write @negedge yang digunakan dalam praktikum. Dengan demikian, proses baca dan tulis memiliki domain waktu yang terpisah sehingga mengurangi possibilitas ada terjadinya konflik memori.

Hasil kompilasi, seperti yang ditunjukkan pada "Fitter Summary" di gambar lampiran, berhasil dengan penggunaan logika yang sangat rendah (<1% Logic utilization) dan pemanfaatan block memory bits 8,192 / 419,328 (<1%), menunjukkan efisiensi tinggi dalam penggunaan hardware resource FPGA. Laporan Timing Analyzer Summary juga menunjukkan Total number of failed paths adalah 0, menandakan bahwa desain ini memenuhi semua batasan timing yang ditentukan dan akan beroperasi dengan andal.

Hasil simulasi fungsional dan timing pada Quartus II, ditunjukkan oleh waveform pada gambar, memperlihatkan bahwa operasi read memiliki delay sebesar 17.934 ns, sedangkan operasi write memiliki delay sebesar 11.475 ns. Perbedaan ini dapat dijelaskan oleh sifat dasar dari implementasi memori sinkron. Jalur baca (read path) melibatkan propagasi sinyal alamat hingga ke blok RAM internal dan keluaran data (q_ram), sehingga membutuhkan waktu lebih lama. Sebaliknya, jalur tulis lebih pendek karena proses penulisan terjadi secara sinkron terhadap clock edge dan langsung memodifikasi isi RAM internal.

Dari percobaan tugas 3 ini, penggunaan ALTSYNCRAM MegaFunction memberikan efisiensi dan prediktabilitas waktu yang lebih baik dibandingkan implementasi manual menggunakan array reg. MegaFunction ini secara otomatis memanfaatkan block RAM internal FPGA untuk mempercepat akses memori dan mengoptimalkan pemakaian sumber daya logika. Hasil simulasi yang diperoleh menunjukkan perilaku compiling yang sesuai dengan teori, di mana delay read sedikit lebih besar daripada delay write, menandakan bahwa desain telah bekerja dengan benar dan konsisten terhadap spesifikasi praktikum, serta pengunaan clocking yang digunakan efektif dalam mencegah konflik baca/tulis

D. Tugas 4: Perancangan Register

Register file (reg_file_rv32i.v) diimplementasikan sebagai komponen memori krusial pada mikroprosesor Single-Cycle RISC-V (RV32I), terdiri dari 32 general-purpose registers berlebar 32-bit. Desain ini memiliki dua read ports (rs1_addr, rs2_addr) dan satu write port (rd_addr).

Dari GTKwave dapat diamati setelah 5 siklus clock, tidak bisa diganti lagi nilai yang diberikan. Hal ini karena hasil dari Single Cycle. Demikian terlampir 0000000 untuk seluruh nilai meskipun diganti setelah 0x000000DD. Dapat dilihat register x0 (register nol), yang nilainya selalu dijaga pada 0 dan setiap upaya penulisan ke x0 diabaikan, sesuai dengan spesifikasi arsitektur RISC-V.

Implementasi verilog mengkonfirmasi inisialisasi semua

register ke 32'b0 pada awal simulasi. Disiplin clocking yang digunakan adalah penulisan data pada rising edge clock (@posedge clock) dan pembacaan data pada falling edge clock (@negedge clock). Logika penulisan mencakup kondisi if (cu_rdwrite && (rd_addr != 5'd0)) rf[rd_addr] <= rd_in; yang mengabaikan penulisan ke x0. Secara eksplisit, rf[0] <= 32'b0; juga ditambahkan di blok always @(posedge clock) untuk menjaga nilai x0 tetap nol. Sementara itu, logika pembacaan (rs1_addr == 5'd0) ? 32'b0 : rf[rs1_addr] menjamin bahwa jika alamat baca adalah x0, keluaran yang diberikan adalah 32'b0.

Hasil kompilasi, seperti yang ditunjukkan pada "Fitter Summary" di gambar, berhasil dengan penggunaan logic utilization sebesar 12% (Combinational ALUTs 5%, Dedicated logic registers 9%). Total registers yang digunakan adalah 1067, yang menunjukkan bahwa register file ini diimplementasikan menggunakan banyak dedicated flip-flop di FPGA, sesuai dengan desainnya.

Simulasi fungsional menggunakan GTKWave (ditampilkan dalam waveform yang diberikan) memvalidasi sifat register file. Terlihat bahwa data berhasil ditulis ke register tertentu pada rising edge clock ketika cu_rdwrite aktif dan rd_addr bukan x0. Kemudian, data tersebut dapat dibaca dari read ports pada falling edge clock. Perilaku x0 yang selalu 0 dan pengabaian penulisan ke dalamnya juga terkonfirmasi, menunjukkan fungsionalitas yang benar.

Analisis timing dari "Timing Analyzer Summary" menunjukkan nilai-nilai kunci: Worst-case tsu (setup time) adalah 8.179 ns, dan Worst-Case TCO (clock-to-output delay) adalah 7.134 ns. Delay ini merepresentasikan waktu yang dibutuhkan dari clock edge hingga data valid tersedia di keluaran register file. Disiplin clocking yang memisahkan write pada posedge dan read pada negedge secara efektif mencegah konflik data pada port yang bertabrakan, memastikan integritas data dan operasi yang andal dalam lingkungan single-cycle RISC-V ini.

IV. Simpulan

- Praktikan telah memahami arsitektur mikroprosesor RISC-V (RV32I) secara umum, termasuk datapath eksekusi single-cycle. Konsep Harvard Architecture dengan pemisahan instruction memory dan data memory juga telah dipahami untuk menghindari structural hazard. Teramati pada percobaan tugas 1,2,3.
- Praktikan telah memahami instruction set inti RV32I, termasuk format instruksi R, I, S, B, U, J, serta fungsi opcode, funct3, funct7, dan immediate. Melalui penggunaan simulator/assembler RISC-V seperti Venus, praktikan mampu menulis dan mensimulasikan program assembly sederhana, serta memahami alur eksekusi setiap instruksi. Hal ini ditinjau dari percobaan tugas 1,2,3,4.
- Praktikan berhasil membuat implementasi instruction memory, data memory, dan register file RISC-V dalam kode Verilog. Komponen-komponen ini telah disintesis dan disimulasikan menggunakan Intel® Quartus® Prime dan ModelSim, dengan waveform simulasi yang menunjukkan kesesuaian dengan alamat yang dimasukkan dan operasi yang diharapkan. Hal ini ditinjau dari percobaan tugas 1,2,3,4.
- Penggunaan Altera® MegaFunction ALTSYNCRAM terbukti lebih efisien dalam implementasi memori dibandingkan dengan desain manual, terutama dalam hal pemanfaatan block RAM, timing performance, dan kemudahan inisialisasi melalui file .mif. Hal ini ditinjau dari percobaan tugas 1,2,3,4.
- Disiplin clocking yang digunakan, yaitu penulisan pada rising edge clock dan pembacaan pada falling edge clock untuk register file, serta kombinasi write @negedge dan read asinkron untuk data memory, berhasil memastikan operasi baca/tulis berjalan tanpa konflik pada port yang bertabrakan. Hal ini ditinjau dari percobaan tugas 1,2,3,4.
- Fungsi register x0 yang selalu bernilai 0 pada RISC-V telah dipahami oleh praktikan dan diimplementasikan secara tepat dalam register file. Hal ini teramati pada percobaan tugas 4, di mana penulisan ke x0 diabaikan dan nilainya selalu dipertahankan nol.

Referensi

[1] D. A. Patterson & J. L. Hennessy, Computer Organization and Design RISC-V Edition.

[2] D. Harris & S. Harris, Digital Design and Computer Architecture: RISC-V Edition.

[3] S. Sutandi, dkk., "Modul Praktikum EL3011 Arsitektur Sistem Komputer Semester I Tahun Akademik 2025 / 2026,". Ms Teams (Diakses 6 November 2025).

[4] Venus. (n.d.). [Online]. Available: https://github.com/61c-teach/venus/wiki (Diakses 6 November 2025).

[5] AMD Inc. Xilinx, "Choosing between Distributed RAM and Dedicated Block RAM," UG901, 2018. [Online]. Available: https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Choosing-Between-Distributed-RAM-and-Dedicated-Block-RAM (Diakses 6 November 2025).

[6] "Teknologi pada FPGA (Field Programmable Gate Array)," Binus Bandung, 2019. [Online]. Available: https://binus.ac.id/bandung/2019/12/teknologi-pada-fpga-field-programmable-gate-array (Diakses 6 November 2025).

[7] "Implementasi Sistem Kendali ON-OFF pada Field Programmable Gate Array (FPGA)." [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwijo-bk39eQAxWq1TgGHV-SHX8QFnoECBgQAQ&url=https%3A%2F%2Fjtera.polteksmi.ac.id%2Findex.php%2Fjtera%2Farticle%2Fdownload%2F548%2F256&usg=AOvVaw3HRMjsHSvpioU64D1ru2xw&opi=89978449 (Diakses 6 November 2025).

[8] "All About the RISC-V Processors," Stromasys, 2025. [Online]. Available: https://www.stromasys.com/resources/all-about-the-risc-v-processors (Diakses 6 November 2025).

[9] "Arsitektur dan Organisasi Komputer - Pertemuan 6," Narin Laboratory, 2023. [Online]. Available: https://narin.co.id/artikel/arsitektur-dan-organisasi-komputer-pertemuan-6.htm (Diakses 6 November 2025)

Lampiran

1. Source code untuk studi pustaka Bahasa Verilog HDL

```
module Mux2_4b (
    input  wire [3:0] A_IN,
    input  wire [3:0] B_IN,
    input  wire       S_IN,
    output wire [3:0] C_OUT
);
genvar i;
generate
    for (i=0; i<4; i=i+1) begin : G
    Mux2 u(.A(A_IN[i]), .B(B_IN[i]), .S(S_IN), .D(C_OUT[i]));
    end
    endgenerate
endmodule
```

2. Source code untuk tugas I

```
//
================================================================================
// Praktikum EL3011 Arsitektur Sistem Komputer
// Modul        : 1
// Percobaan    : 1
// Tanggal      : 5 November 2025
// Kelompok     :
// Rombongan    : Rabu
// Nama (NIM) 1 : William Anthony (13223048)
// Nama (NIM) 2 : Darren Johan (13223032)
// Nama File    : instr_rom_rv32i.v
// Deskripsi    : Instruction ROM RV32I, 32x32, baca sinkron @posedge clock
//
================================================================================


module instr_rom_rv32i (
    input  wire [31:0] ADDR,   // byte address dari PC
    input  wire        clock,
    input  wire        reset,
    output reg  [31:0] INSTR   // instruksi 32-bit
);
  // 32 word x 32-bit
  reg [31:0] mem [0:31];
  // Word index = PC[6:2] (bit [1:0] = 2'b00)
  wire [4:0] waddr = ADDR[6:2];

  integer i;
  initial begin
    // Default: semua NOP (addi x0,x0,0)
    for (i = 0; i < 32; i = i + 1) mem[i] = 32'h00000013;

     // Isi program (ganti sesuai dengan "Machine Code" dari Venus pada Tugas
Pendahuluan nomor 3!):
    mem[0] = 32'h00500093; // addi x1,x0,5
    mem[1] = 32'h00700113; // addi x2,x0,7
    mem[2] = 32'h002081B3; // add  x3,x1,x2
    mem[3] = 32'h00302023; // sw   x3,0(x0)

    // Alternatif: load dari file heksa
    //$readmemh("imemory_rv32i.hex", mem);
  end
```

```
  // Baca sinkron; reset keluarkan NOP
  always @(posedge clock or posedge reset) begin
    if (reset) INSTR <= 32'h00000013;   // NOP RV32I
    else       INSTR <= mem[waddr];
  end
endmodule
```

3. Screenshot hasil tugas 1

Fitter Status	Successful - Tue Nov 04 21:00:37 2025
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	instr_rom_rv32i
Top-level Entity Name	instr_rom_rv32i
Family	Stratix II
Device	EP2S15F484C3
Timing Models	Final
Logic utilization	< 1 %
Combinational ALUTs	8 / 12,480 (< 1 %)
Dedicated logic registers	9 / 12,480 (< 1 %)
Total registers	9
Total pins	66 / 343 (19 %)
Total virtual pins	0
Total block memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)

Analysis Fitter Tugas 1

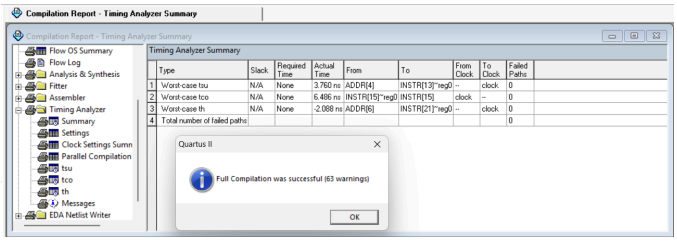
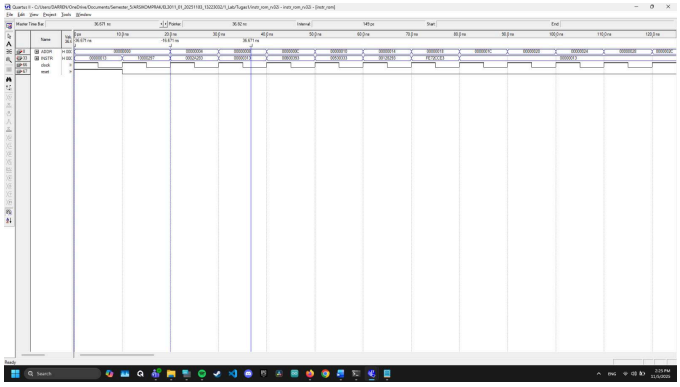
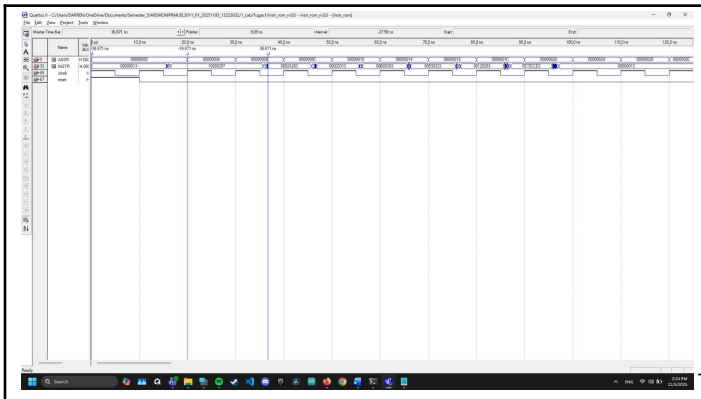


Fig. 6 Timing Analysis Tugas 1



Fungsional Run Tugas 1


Timing Run Tugas 1

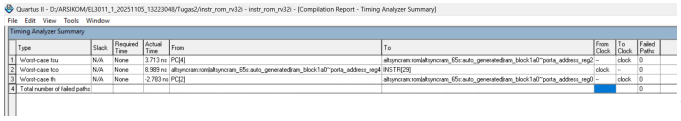
4. Source code untuk tugas 2

```
//
=================================================================================
// Praktikum EL3011 Arsitektur Sistem Komputer
// Modul         : 1
// Percobaan     : 2
// Tanggal       : 5 November 2025
// Kelompok      :
// Rombongan     : Rabu
// Nama (NIM) 1 : William Anthony (13223048)
// Nama (NIM) 2 : Darren Johan (13223032)
// Nama File     : instr_rom_rv32i.v
// Deskripsi     : Instruction ROM 32x32 (RV32I) via ALTSYNCRAM + .mif
//
=================================================================================


`timescale 1ns/1ps
module instr_rom_rv32i (
    input  wire        clock,
    input  wire [31:0] PC,      // byte address
    output wire [31:0] INSTR
);
  // Word index untuk 32 word
  wire [4:0] waddr = PC[6:2];

  altsyncram #(
    .operation_mode("ROM"),
    .width_a(32),
    .widthad_a(5),                  // 32 word
    .init_file("imemory_rv32i.mif"),
    .outdata_reg_a("UNREGISTERED")
  ) rom (
    .clock0    (clock),
    .address_a(waddr),
    .q_a       (INSTR),
    .wren_a    (1'b0),
    .data_a    (32'b0)
  );
endmodule
```

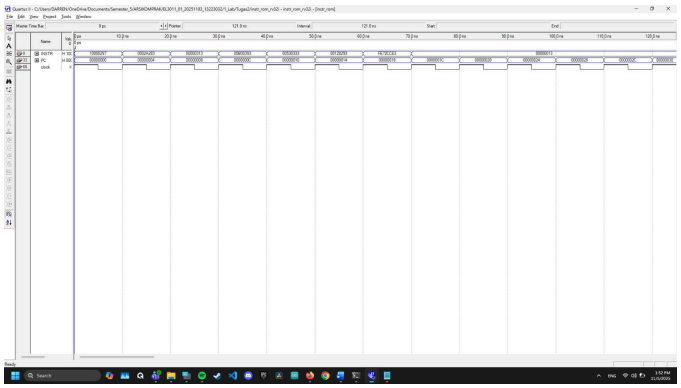
5. Screenshot hasil tugas 2

Flow Status	Successful - Tue Nov 04 21:50:16 2025
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	instr_rom_rv32i
Top-level Entity Name	instr_rom_rv32i
Family	Stratix II
Met timing requirements	Yes
Logic utilization	0 %
Combinational ALUTs	0 / 12,480 (0 %)
Dedicated logic registers	0 / 12,480 (0 %)
Total registers	0
Total pins	65 / 343 (19 %)
Total virtual pins	0
Total block memory bits	1,024 / 419,328 (< 1 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

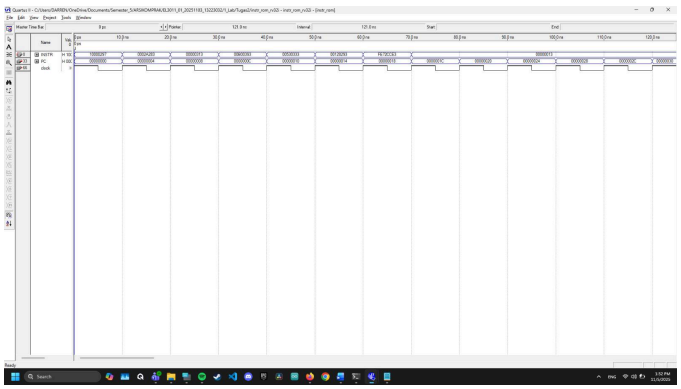
Fitter Analysis Tugas 2



Timing Analysis



Fungsional



Timing

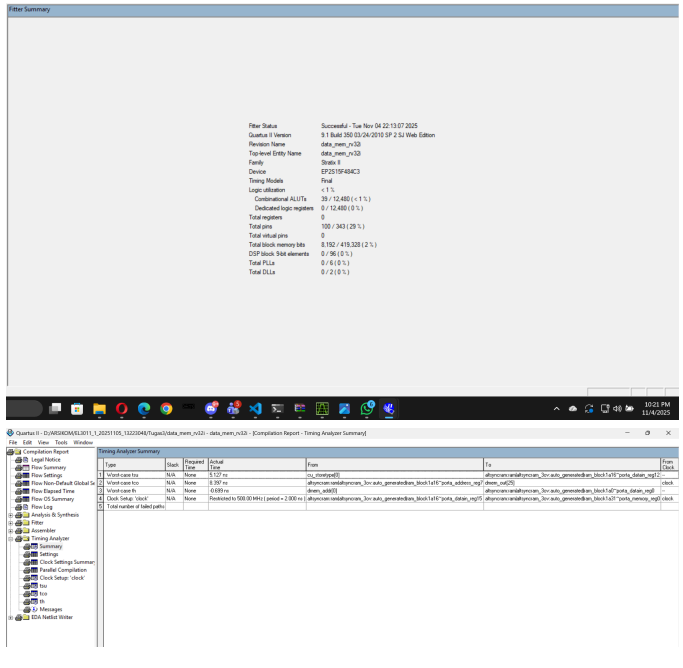
6. Source code untuk tugas 3

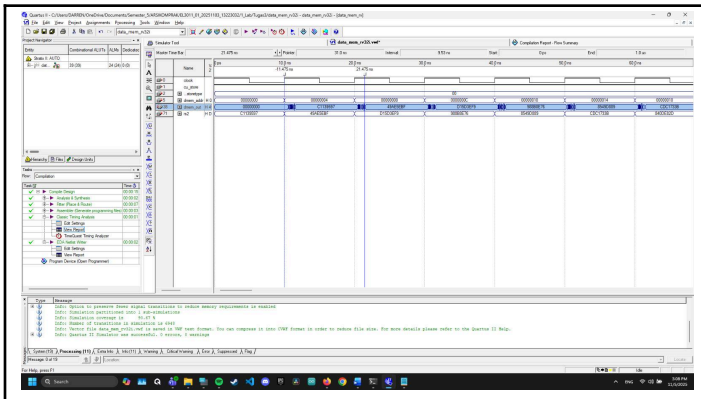
```
//
================================================================================
// Praktikum EL3011 Arsitektur Sistem Komputer
// Modul         : 1
// Percobaan     : 3
// Tanggal       : 5 November 2025
// Kelompok      :
// Rombongan     : Rabu
// Nama (NIM) 1  : William Anthony (13223048)
// Nama (NIM) 2  : Darren Johan (13223032)
// Nama File     : data_mem_rv32i.v
// Deskripsi     :
// Modul ini merepresentasikan Data Memory (DMEM) untuk prosesor RV32I.
```

```
// Akses tulis dilakukan pada tepi jatuh clock (Write @negedge via clock terbalik),
// sedangkan pembacaan dilakukan secara asinkron (keluaran mengikuti alamat).
// Hindari konflik tulis/baca pada alamat yang sama jika membahayakan timing.
//
================================================================================

`timescale 1ns/1ps
module data_mem_rv32i (
    input  wire        clock,
    input  wire        cu_store,        // WE dari Control Unit
    input  wire [1:0]  cu_storetype,    // 00=SW, 01=SH, 10=SB
    input  wire [31:0] dmem_addr,       // alamat byte
    input  wire [31:0] rs2,             // data yang akan ditulis
    output wire [31:0] dmem_out         // data yang dibaca (asinkron)
);

    wire [7:0] waddr = dmem_addr[9:2];
    reg [3:0] be;
    always @* begin
        case (cu_storetype)
             2'b00: be = 4'b1111;                                          // SW
(Word)
             2'b01: be = (dmem_addr[1]) ? 4'b1100 : 4'b0011;               // SH
(Halfword)
            2'b10: case (dmem_addr[1:0])                                   // SB
(Byte)
                       2'b00: be = 4'b0001;
                       2'b01: be = 4'b0010;
                       2'b10: be = 4'b0100;
                       default: be = 4'b1000;
                   endcase
            default: be = 4'b0000;
        endcase
    end

    reg [31:0] wr_data_aligned;
    always @* begin
        case (cu_storetype)
            2'b00: wr_data_aligned = rs2;                                  // SW
            2'b01: wr_data_aligned = (dmem_addr[1]) ?
                                     {rs2[15:0], 16'b0} :                  // SH
                                     {16'b0, rs2[15:0]};
            2'b10: case (dmem_addr[1:0])                                   // SB
                       2'b00: wr_data_aligned = {24'b0, rs2[7:0]};
                       2'b01: wr_data_aligned = {16'b0, rs2[7:0], 8'b0};
                       2'b10: wr_data_aligned = {8'b0, rs2[7:0], 16'b0};
                       default: wr_data_aligned = {rs2[7:0], 24'b0};
                   endcase
            default: wr_data_aligned = 32'b0;
        endcase
    end

    // Clock negatif ya untuk menghindari konflik tulis-baca
    wire clk_n = ~clock;

    // RAM internal (altsyncram)
```

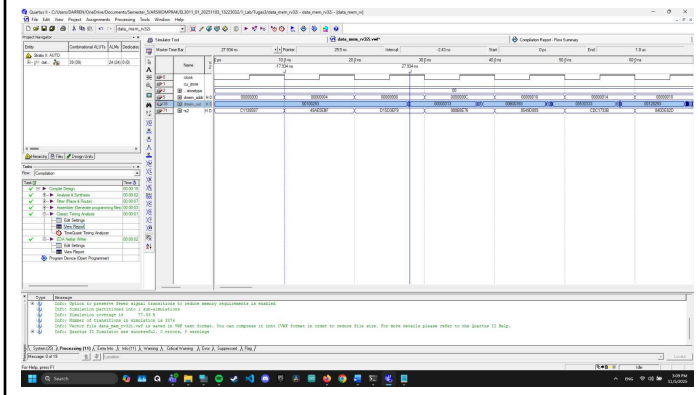
```
    // - Write dilakukan pada falling edge
    // - Read bersifat asinkron (keluaran langsung mengikuti alamat)
    wire [31:0] q_ram;
    assign dmem_out = q_ram;

    altsyncram #(
        .operation_mode    ("SINGLE_PORT"),
        .width_a           (32),
        .widthad_a         (8),
        .outdata_reg_a     ("UNREGISTERED"),    // read async
        .init_file         ("dmemory.mif"),
        .width_byteena_a   (4)
    ) ram (
        .clock0     (clk_n),                    // tulis pada tepi jatuh clock
        .wren_a     (cu_store),
        .address_a  (waddr),
        .data_a     (wr_data_aligned),
        .q_a        (q_ram),
        .byteena_a  (be)
    );

endmodule

/*
cd D:/ARSIKOM/EL3011_1_20251105_13223048/Tugas3; vlib work; vmap work work;
vlog data_mem_rv32i.v data_mem_rv32i_tb.v; vsim data_mem_rv32i_tb; run 500ns
*/
```

7. Screenshot hasil tugas 3





Delay Write



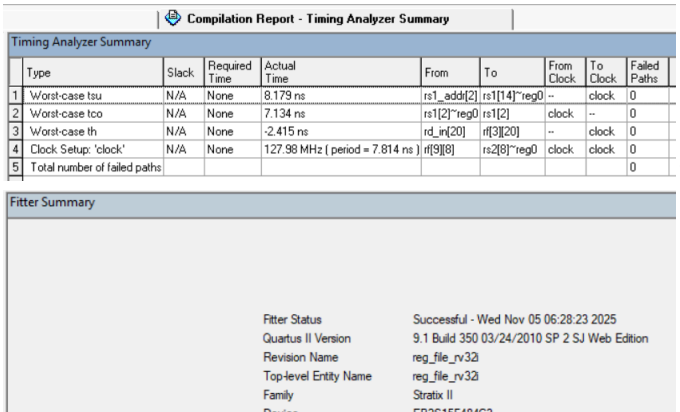
Delay Read

8. Source code untuk tugas 4 (Verilog)

```
//
================================================================================
// Praktikum EL3011 Arsitektur Sistem Komputer
// Modul          : 1
// Percobaan      : 4
// Tanggal        : 5 November 2025
// Kelompok       :
// Rombongan      : Rabu
// Nama (NIM) 1   : William Anthony (13223048)
// Nama (NIM) 2   : Darren Johan (13223032)
// Nama File      : reg_file_rv32i.v
// Deskripsi      : Register file RV32I, 32x32, 2 read ports, 1 write port.
//                  x0 selalu 0; write @posedge, read @negedge

`timescale 1ns/1ps
module reg_file_rv32i (
input  wire        clock,          // global clock
input  wire        cu_rdwrite,     // write enable dari CU
input  wire [4:0]  rs1_addr,       // alamat baca port 1
input  wire [4:0]  rs2_addr,       // alamat baca port 2
input  wire [4:0]  rd_addr,        // alamat tulis (write-back)
input  wire [31:0] rd_in,          // data tulis (write-back data)
output reg  [31:0] rs1,            // data baca port 1
output reg  [31:0] rs2             // data baca port 2
);
// 32 register x 32-bit
reg [31:0] rf [0:31];
integer i;
```

```
initial begin
for (i = 0; i < 32; i = i + 1) rf[i] = 32'b0;  // untuk simulasi
end
// Tulis @posedge; tulis ke x0 diabaikan. Jaga x0 selalu nol.
always @(posedge clock) begin
if (cu_rdwrite && (rd_addr != 5'd0))
rf[rd_addr] <= rd_in;
rf[0] <= 32'b0;
end
// Baca @negedge (sinkron, sesuai instruksi praktikum)
always @(negedge clock) begin
rs1 <= (rs1_addr == 5'd0) ? 32'b0 : rf[rs1_addr];
rs2 <= (rs2_addr == 5'd0) ? 32'b0 : rf[rs2_addr];
end
endmodule

/*
cd D:/ARSIKOM/EL3011_1_20251105_13223048/Tugas4
vlib work
vmap work work
vlog "C:/altera/91sp2/quartus/eda/sim_lib/altera_mf.v"
vlog reg_file_rv32i.v reg_file_rv32i_tb.v
vsim reg_file_rv32i_tb
add wave *
run 1000ns
*/
```

9. Screenshot hasil tugas 4



Compilation Report - Timing Analyzer Summary

Timing Analyzer Summary

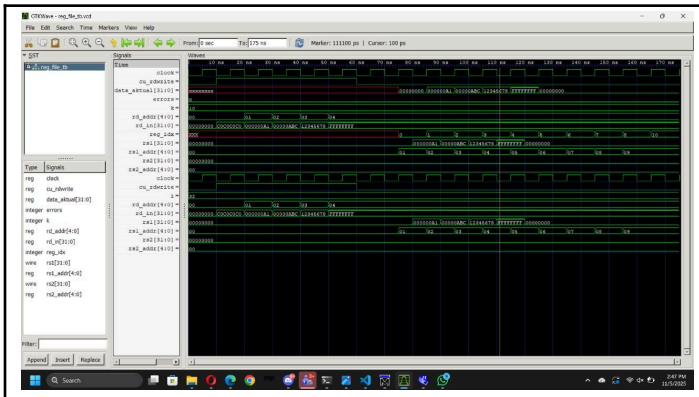
	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	8.179 ns	rs1_addr[2]	rs1[14]~reg0	--	clock	0
2	Worst-case tco	N/A	None	7.134 ns	rs1[2]~reg0	rs1[2]	clock	--	0
3	Worst-case th	N/A	None	-2.415 ns	rd_in[20]	rf[3][20]	--	clock	0
4	Clock Setup: 'clock'	N/A	None	127.98 MHz (period = 7.814 ns)	rf[9][8]	rs2[8]~reg0	clock	clock	0
5	Total number of failed paths								0

Timing Analyzer

Fitter Summary

Fitter Status	Successful - Wed Nov 05 06:28:23 2025
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	reg_file_rv32i
Top-level Entity Name	reg_file_rv32i
Family	Stratix II
Device	EP2S15F484C3
Timing Models	Final
Logic utilization	12 %
Combinational ALUTs	681 / 12,480 (5 %)
Dedicated logic registers	1,067 / 12,480 (9 %)
Total registers	1067
Total pins	113 / 343 (33 %)
Total virtual pins	0
Total block memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)

Fitter Analysis



GTKWave fungsional hasil