GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x7 Deney Raporu

Lojik Devreler ve Tasarım Laboratuvarı Dersi

Hazırlayanlar
1) 171024086-Berat KIZILARMUT
2) 1801022037-Ömer Emre POLAT

1. Giriş

Deneyimizde istenilen protokollere göre farklı seçimler yapabilen bir devre oluşturacağız.

Bu deneyde iki farklı devre tasarlayıp, bu iki devreyi kullanarak üçüncü bir devre oluşturacağız. İlk devremiz gelen komutlara göre davranışını değiştirerek çıktılar verecek. İkinci devremiz ise bir hafıza ünitesi olacak. Ardından bu iki devre birleştirilecek.

2. Problemler

2.1. Problem I – Komut Ayırıcı

İlk problemimizde gönderilen 32-bit'lik bir komutu istenilen yönergelere uyarak parçalayacak ve istenilen işlemleri gerçekleştirecek bir devre oluşturacağız.

A. HDL Yazılımı

İstenilen işlevleri gerçekleştiren SystemVerilog yazılımımız, Switch Case kullanılarak opcode'ların tanım çerçevesi oluşturuldu. Gelen komut sinyalleri belirli parçalara bölünerek, çıkış sinyali olarak verildi.

```
/* lab7_g41_p1.sv
* Hazırlayanlar: Berat Kızılarmut,
                 Ömer Emre Polat
* Notlar: Komut Ayırıcı */
module lab7_g41_p1 (
input logic clk, reset,
input logic [31:0] komut,
output logic [6:0] opcode,
output logic [3:0] aluop,
output logic [4:0] rs1,
output logic [4:0] rs2,
output logic [31:0] rs1_data,
output logic [31:0] rs2_data,
output logic [4:0] rd,
output logic [31:0] imm,
output logic hata
);
assign opcode = komut[6:0];
assign rs1_data = 32'd0;
assign rs2_data = 32'd0;
always_comb
begin
    case(opcode)
        7'b0000001: /R/
        begin
            aluop = {komut[30], komut[14:12]};
            rs1 = komut[19:15];
            rs2 = komut[24:20];
            rd = komut[11:7];
            imm = 32'd0;
            hata = 1'b0;
        end
        7'b0000011: /I/
        begin
            aluop = {1'b0, komut[14:12]};
            rs1 = komut[19:15];
            rs2 = 5'd0;
            rd = komut[11:7];
            imm = {20'd0, komut[31:20]};
            hata = 1'b0;
    end
        7'b0000111: /U/
    begin
            aluop = 4'd0;
            rs1 = 5'd0;
            rs2 = 5'd0;
            rd = komut[11:7];
            imm = {12'd0, komut[31:12]};
            hata = 1'b0;
     end
        7'b0001111: /B/
     begin
            aluop = {1'b0, komut[14:12]};
            rs1 = komut[19:15];
            rs2 = komut[24:20];
            rd = 5'd0;
            imm = {19'd0, komut[31:25], komut[11:7], 1'b0};
            hata = 1'b0;
      end
      default:
      begin
            hata = 1'b1;
            aluop = 4'd0;
            rs1 = 5'd0;
            rs2 = 5'd0;
            rd = 4'd0;
            imm = 32'd0;
      end
    endcase
end
endmodule
```

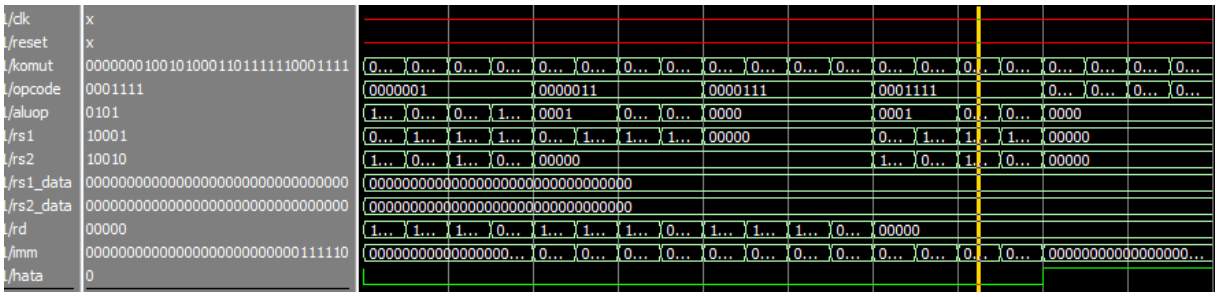
B. Testbench Kurulumu

Kurulan devreyi sınamak için duruma ve devreye uygun bir testbench devresi kuruldu. Devre explicit instantiation şeklinde çağırıldı. Verilen 4 farklı komut seti için ayrı ayrı denemeler yapıldı. Hata kodu da sınandı.

```
/* tb_lab7_g41_p1.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Komut ayırıcı testbenchi */
`timescale 1ns/1ps
module tb_lab7_g41_p1();
logic clk, reset;
logic [31:0] komut;
logic [6:0] opcode;
logic [3:0] aluop;
logic [4:0] rs1;
logic [4:0] rs2;
logic [31:0] rs1_data;
logic [31:0] rs2_data;
logic [4:0] rd;
logic [31:0] imm;
logic hata;
lab7_g41_p1 dut1(.clk(clk), .reset(reset), .komut(komut), .opcode(opcode),
.aluop(aluop), .rs1(rs1), .rs2(rs2), .rs1_data(rs1_data), .rs2_data(rs2_dat
a), .rd(rd), .imm(imm), .hata(hata));
initial
begin
    /////////////////R//////////////////////
    komut = 32'b0010000_10100_00110_001_10011_0000001; #10;
    komut = 32'b0000000_01011_11001_001_10111_0000001; #10;
    komut = 32'b0000000_10010_10001_101_11111_0000001; #10;
    komut = 32'b0010000_01110_10101_100_01100_0000001; #10;
    /////////////////I//////////////////////
    komut = 32'b001000010100_00110_001_10011_0000011; #10;
    komut = 32'b000000001011_11001_001_10111_0000011; #10;
    komut = 32'b000000010010_10001_101_11111_0000011; #10;
    komut = 32'b001000001110_10101_100_01100_0000011; #10;
    /////////////////U//////////////////////
    komut = 32'b00100001010000110001_10011_0000111; #10;
    komut = 32'b00000000101111001001_10111_0000111; #10;
    komut = 32'b00000001001010001101_11111_0000111; #10;
    komut = 32'b00100000111010101100_01100_0000111; #10;
    /////////////////B//////////////////////
    komut = 32'b0010000_10100_00110_001_10011_0001111; #10;
    komut = 32'b0000000_01011_11001_001_10111_0001111; #10;
    komut = 32'b0000000_10010_10001_101_11111_0001111; #10;
    komut = 32'b0010000_01110_10101_100_01100_0001111; #10;
    /////////////////HATA//////////////////////
    komut = 32'b0010000_10100_00110_001_10011_0010001; #10;
    komut = 32'b0000000_01011_11001_001_10111_0100001; #10;
    komut = 32'b0000000_10010_10001_101_11111_0001001; #10;
    komut = 32'b0010000_01110_10101_100_01100_0101101; #10;
    $stop;
end
endmodule
```

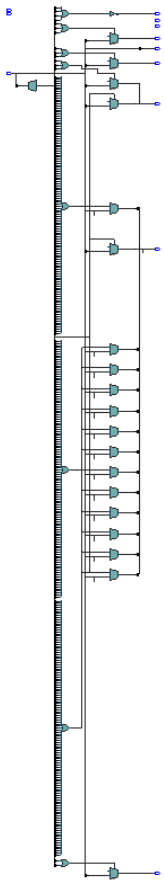
C. Waveshape Analizi

Testbench sonuçlarını gösteren dalga formu elde edildi. Devremiz verilen opcode'lara göre doğru işlemleri gerçekleştirmiştir. Opcode girişinde bir hata bulunduğunda ise hata sinyalini vermiştir. Devre, istediğimiz ve beklediğimiz gibi çalışmıştır.

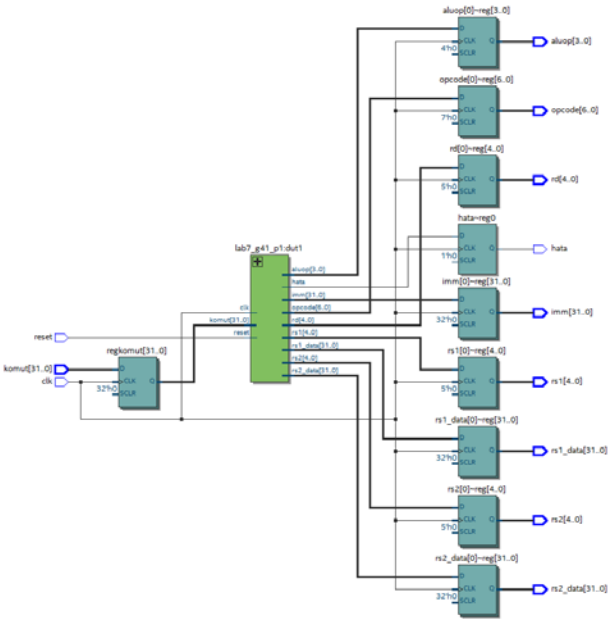


D. Problem I Analiz ve Yorumlar

SystemVerilog yazılımımız beklenildiği gibi çalışmıştır. Bu devrenin kurulumunda 64 Combinational ALUT ünitesi kullanılmıştır. Hiç register kullanılmıştır. Devremizin Fmax'i 328 MHz olarak hesaplanmıştır. Devre Switch Case ile oluşturulduğu için equal ünitesi kullanılmamıştır. Bundan dolayı RTL Şeması çok büyük bir boyutta çıkmıştır.



RTL Şeması



Timing Analysis RTL

Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
|lab7_g41_p1	64 (64)	0 (0)	0	0	0	0	0	157

Fmax	Restricted Fmax	Clock Name	Note
328.08 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

2.2 Problem II – Hafıza oluşturma ve okuma

Bu problemde 32 adet 32-bit register kullanarak bir basit 32-bit hafıza modülü tasarlayacağız.

A. HDL Yazılımı

SystemVerilog HDL dilinde, 32-bitlik verileri saklayan, 32 bit registerlardan oluşan bir memory tasarlandı. Memory verilen adres ve veriyi kullanarak, yazma veya okuma işlemlerini gerçekleştirebilmektedir.

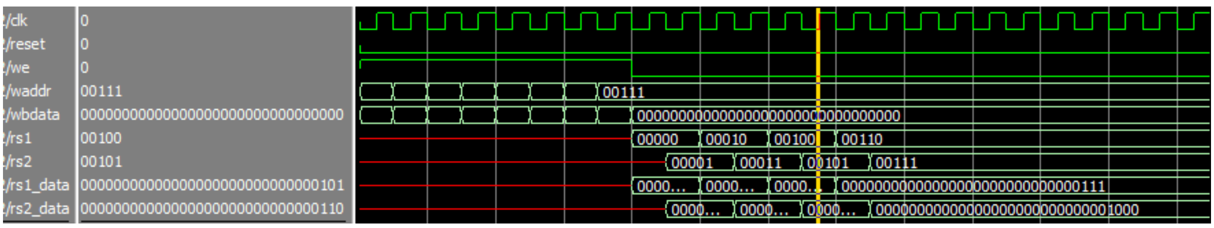
```
/* lab7_g41_p2.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Hafıza oluşturma ve okuma */
module lab7_g41_p2 (
input logic clk, reset,
input logic we,
input logic [4:0] waddr,
input logic [31:0] wbdata,
input logic [4:0] rs1,
input logic [4:0] rs2,
output logic [31:0] rs1_data,
output logic [31:0] rs2_data
);
logic [31:0] mem [0:31];
always_comb
begin
    rs1_data = mem[rs1];
    rs2_data = mem[rs2];
end

always_ff @(posedge clk)
begin
    if(we)
        mem[waddr] <= wbdata;
end
endmodule
```

B. Testbench Kurulumu

Memory'nin ilk sekiz verisi, 1'den 8'e kadar sayılarla doldurulduru. Doldurma işleminin ardından rs1 ve rs2 adresleri değiştirilerek registerlar okundu ve kontrol edildi. Memory istenildiği gibi yazma ve okuma işlemlerini gerçekleştirdi.

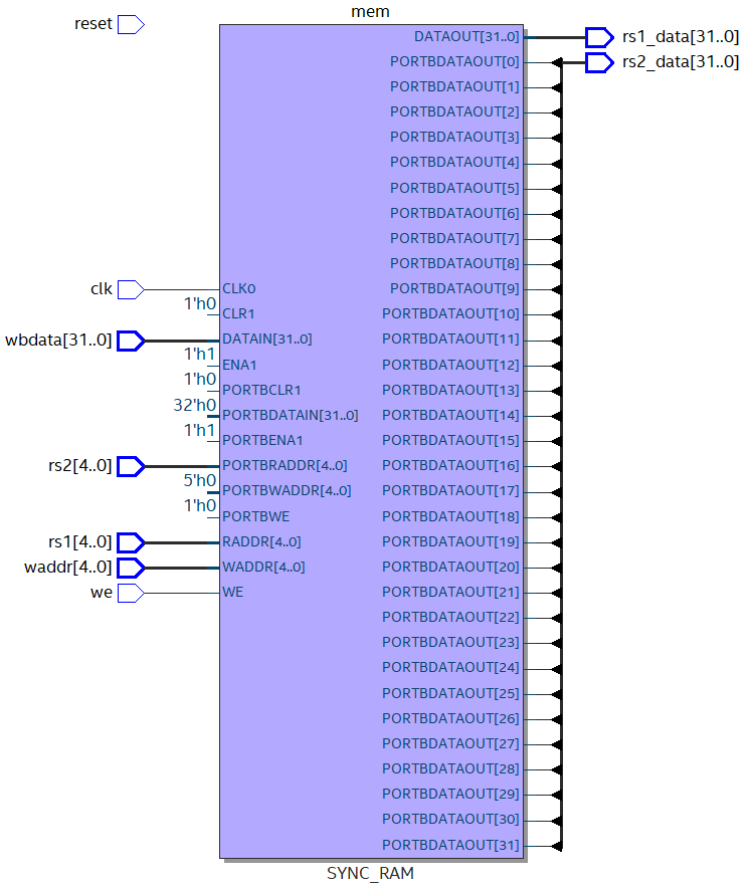
```
/* tb_lab7_g41_p2.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Basit hafıza testbenchi */
`timescale 1ns/1ps
module tb_lab7_g41_p2();
    logic clk, reset;
    logic we;
    logic [4:0] waddr;
    logic [31:0] wbdata;
    logic [4:0] rs1;
    logic [4:0] rs2;
    logic [31:0] rs1_data;
    logic [31:0] rs2_data;
    lab7_g41_p2 dut0(.clk(clk), .reset(reset), .we(we), .waddr(waddr), .wbdata(wbdata), .
rs1(rs1), .rs2(rs2), .rs1_data(rs1_data), .rs2_data(rs2_data));
    always begin
        clk <= 0; #5; clk <= 1; #5;
    end
    initial
    begin
        we = 1; reset = 0;
        waddr = 5'b00000; wbdata = 32'd1; #10;
        waddr = 5'b00001; wbdata = 32'd2; #10;
        waddr = 5'b00010; wbdata = 32'd3; #10;
        waddr = 5'b00011; wbdata = 32'd4; #10;
        waddr = 5'b00100; wbdata = 32'd5; #10;
        waddr = 5'b00101; wbdata = 32'd6; #10;
        waddr = 5'b00110; wbdata = 32'd7; #10;
        waddr = 5'b00111; wbdata = 32'd8; #10;
        we = 0; reset = 0; wbdata = 32'b0;
        rs1 = 5'b00000; # 10;
        rs2 = 5'b00001; # 10;
        rs1 = 5'b00010; # 10;
        rs2 = 5'b00011; # 10;
        rs1 = 5'b00100; # 10;
        rs2 = 5'b00101; # 10;
        rs1 = 5'b00110; # 10;
        rs2 = 5'b00111; # 100;
        $stop;
    end
endmodule
```

C. Waveshape Analizi



İşlemin başında memory'ye veri yazılırken rs1 ve rs2 sinyali bulunmadığı için data çıkışları don't care olmuştur. Okunma moduna geçildiğinde ve rs değerleri tanımlı hale geldiğinde istenilen adreslerdeki veriler doğru şekilde okunmuştur.

D. Problem II Analiz ve Yorumlar



Memory'miz istediğimiz gibi çalışmıştır. Devremizde toplam 1392 Combinational ALUT ünitesi kullanılmıştır. Ayrıca devremizde 1024 Logic Register kullanılmıştır. Devrenin RTL Şemasında da görüldüğü üzere, Quartus Prime otomatik olarak registerlardan oluşan bir Sync Ram kullanmıştır.

Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
|lab7_g41_p2	1392 (1392)	1024 (1024)	0	0	0	0	0	114

2.3 Problem III – Modül Birleştirme

Önceki iki adımda tasarlanan Komut ayrıcı devre ile hafıza devresi birbiriyle çalışacak hale getirip, beraber kullanılacaktır.

A. HDL Yazılımı

Devre explicit instantiation şeklinde çağırıldı. Internal node'lar tanımlandı. Bu node'ların tanımlanmasının ardından gerekli olan şekilde devreler birbirine bağlandı.

```
/* lab7_g41_p3.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Modül birleştirme */
module lab7_g41_p3 (
    input logic clk, reset,
    input logic [31:0] komut,
    output logic [6:0] opcode,
    output logic [3:0] aluop,
    output logic [31:0] rs1_data,
    output logic [31:0] rs2_data,
    output logic [31:0] imm,
    output logic hata,
    input logic we,
    input logic [31:0] rd_data
);
logic [4:0] rdin, rs1in, rs2in;
logic [31:0] rs1_datain, rs2_datain;
lab7_g41_p1 dut1(.clk(clk), .reset(reset), .komut(komut), .opcode(o
pcode), .aluop(aluop), .rs1(rs1in), .rs2(rs2in), .rs1_data(rs1_data
in), .rs2_data(rs2_datain), .rd(rdin), .imm(imm), .hata(hata));

lab7_g41_p2 dut2(.clk(clk), .reset(reset), .we(we), .waddr(rdin), .
wbdata(rd_data), .rs1(rs1in), .rs2(rs2in), .rs1_data(rs1_data), .rs
2_data(rs2_data));
endmodule
```

B. Testbench Kurulumu

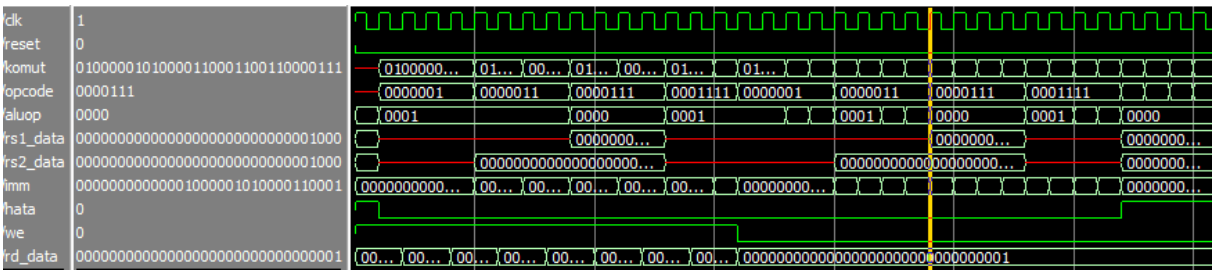
Bu problemdeki asıl zorluk çıkartan kısım burası olmuştur. Bunun sebebi iki devreyi birden sağlıklı bir şekilde sınayacak bir testbench yazılması gerekmekte olmasıdır. Çeşitli işlemlerin komut sinyalini değiştirerek test edilmesi gerekmesi ve hafızaya yazılacak veri girişinin değiştirilerek test edilmesi gerekmesidir.

```
/* tb_lab7_g41_p3.sv
* Hazırlayanlar: Berat Kızılarmut, Ömer Emre Polat
* Notlar: Modül birleştirme testbench */
`timescale 1ns/1ps
module tb_lab7_g41_p3();
logic clk, reset;
logic [31:0] komut;
logic [6:0] opcode;
logic [3:0] aluop;
logic [31:0] rs1_data;
logic [31:0] rs2_data;
logic [31:0] imm;
logic hata;
logic we;
logic [31:0] rd_data;

lab7_g41_p3 dut0(.clk(clk), .reset(reset), .komut(komut), .opcode(opcod
e), .aluop(aluop), .rs1_data(rs1_data), .rs2_data(rs2_data), .imm(imm),
 .hata(hata), .we(we), .rd_data(rd_data));
always
begin
    clk = 1; #5;
    clk = 0; #5;
end
initial
begin
    we = 1; reset = 0;
```

```
    rd_data = 32'd1; #10;
    komut = 32'b0100000_10100_00110_001_10011_0000001; #10;
    rd_data = 32'd2; #10;
    komut = 32'b0100000_10100_00110_001_10011_0000001; #10;
    rd_data = 32'd3; #10;
    komut = 32'b0100000_10100_00110_001_10011_0000011; #10;
    rd_data = 32'd4; #10;
    komut = 32'b0000000_01011_11001_001_10111_0000011; #10;
    rd_data = 32'd5; #10;
    komut = 32'b0100000_10100_00110_001_10011_0000111; #10;
    rd_data = 32'd6; #10;
    komut = 32'b0000000_01011_11001_001_10111_0000111; #10;
    rd_data = 32'd7; #10;
    komut = 32'b0100000_10100_00110_001_10011_0001111; #10;
    rd_data = 32'd8; #10;
    komut = 32'b0000000_01011_11001_001_10111_0001111; #10;
    we = 0; reset = 0; rd_data = 32'd1;
    ////////////////R///////////////////////
    komut = 32'b0100000_10100_00110_001_10011_0000001; #10;
    komut = 32'b0100000_10100_00110_001_10011_0000001; #10;
    komut = 32'b0000000_10010_10001_101_11111_0000001; #10;
    komut = 32'b0100000_01110_10101_100_01100_0000001; #10;
    ////////////////I///////////////////////
    komut = 32'b0100000_10100_00110_001_10011_0000011; #10;
    komut = 32'b0000000_01011_11001_001_10111_0000011; #10;
    komut = 32'b0000000_10010_10001_101_11111_0000011; #10;
    komut = 32'b0100000_01110_10101_100_01100_0000011; #10;
    ////////////////U///////////////////////
    komut = 32'b0100000_10100_00110_001_10011_0000111; #10;
    komut = 32'b0000000_01011_11001_001_10111_0000111; #10;
    komut = 32'b0000000_10010_10001_101_11111_0000111; #10;
    komut = 32'b0100000_01110_10101_100_01100_0000111; #10;
    ////////////////B///////////////////////
    komut = 32'b0100000_10100_00110_001_10011_0001111; #10;
    komut = 32'b0000000_01011_11001_001_10111_0001111; #10;
    komut = 32'b0000000_10010_10001_101_11111_0001111; #10;
    komut = 32'b0100000_01110_10101_100_01100_0001111; #10;
    ////////////////HATA//////////////////////
    komut = 32'b0100000_10100_00110_001_10011_0010001; #10;
    komut = 32'b0000000_01011_11001_001_10111_0100001; #10;
    komut = 32'b0000000_10010_10001_101_11111_0001001; #10;
    komut = 32'b0100000_01110_10101_100_01100_0101101; #10;
    $stop;
end
endmodule
```

C. Waveshape Analizi

Testbench 1'de olduğu gibi farklı farklı aluop sinyalleri komutun içinde yollanarak farklı girişler ile test edildi. Aynı zamanda testbench 2'de yapıldığı gibi hafızaya yazma ve okuma işlemleri farklı girişlerde yapılarak sınandı.



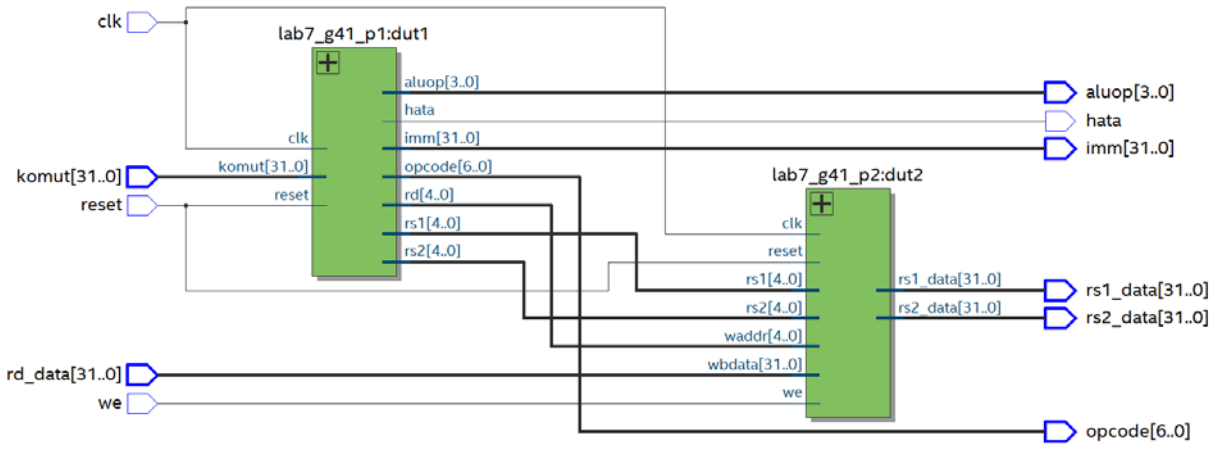
D. Problem III Analiz ve Yorumlar

Resource	Usage
Estimated Total logic elements	2,481
Total combinational functions	1457
Logic element usage by number of LUT inputs	
-- 4 input functions	1333
-- 3 input functions	91
-- <=2 input functions	33
Logic elements by mode	
-- normal mode	1457
-- arithmetic mode	0
Total registers	1024
-- Dedicated logic registers	1024
-- I/O registers	0
I/O pins	175
Embedded Multiplier 9-bit elements	0
Maximum fan-out node	clk~input
Maximum fan-out	1024
Total fan-out	9014
Average fan-out	3.18

Devrede toplam 1457 Combinational ALUT ünitesi kullanıldı. Bu ünitelerden 1333 tanesi dört girişli, 91 tanesi üç girişli ve 33 tanesi iki veya daha az girişli olmaktadır. Devrede toplam 1024 Register kullanılmaktadır.

Bu devreye ALU eklemek için kendi belirleyeceğimiz encoding protokolüne göre yolladığımız komut sinyaline aritmetik işlem seçeneği, işlemin sonucunun yazılacağı hafıza adresi ve işlemde kullanılacak verilerin hafıza adresleri bulunacaktır. P3 Devremiz bu bilgiler kapsamında bir aluop sinyali çıktısı verecektir, bu sinyali ALU'da kendi oluşturduğumuz bir encoding'e göre decode edip gelen veri işlenip, belirlenen adrese hafızaya yazılacaktır.

Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
|lab7_g41_p3	1457 (0)	1024 (0)	0	0	0	0	0	175
|lab7_g41_p1:dut1|	63 (63)	0 (0)	0	0	0	0	0	0
|lab7_g41_p2:dut2|	1394 (1394)	1024 (1024)	0	0	0	0	0	0



3. Genel Yorumlar ve Kazanımlar

Bu deneyde bir komuta ünitesi ile hafıza ünitesinin oluşturulması ve bu iki ünitenin birbiri ile çalışabilecek hale getirilmesini öğrendik. Komuta ünitesi tasarlanmasında çeşitli bit slicingler kullanılmıştır. Bu bit slicing işlemleri ile giriş olarak gelen 32-bitlik komut verisi istenilen şekle sokulmuş ve gereken işlemler yapılmıştır. Hafıza ünitesinde ise register'lar ile hafıza oluşturulması, oluşturulan bu hafızanın giriş ve çıkış sinyalleri ile okunup yazılma işlemleri gerçekleştirmesi sağlanmıştır. Son olarak bu iki devrenin birleştirilmesi aşamasında ise internal node'lar kullanılarak iki modülün arasındaki bilgi alışverişi sağlanmıştır.

4. Kaynaklar

1. Harris and D. Harris, Digital Design and Computer Architecture: ARM Edition, 1st edition. Morgan Kaufmann, 2015