GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x6 Deney Raporu

Komut Ayırıcı

Hazırlayanlar
1) 1801022035 – Ruveyda Dilara Günal
2) 200102002087 – Alican Bayındır

Bu labın amacı

● Komut parçakalama devresi tasarlama

● Tasarlanan devreleri gerçekleyip test edebilmek.

Problem 1 - Komut ayırıcı

Bu problemde 32-bit gönderilen bir komutu, aşağıda verilen isterlere uygun olarak parçalayıp, gerekli çıkışları üreteceksiniz. Bütün devre kombinasyonel lojik olarak çalışacaktır. (clk ve reset pinleri kullanılmayacaktır). Gelen komutlar dört farklı tipte olabilir.

Sizin yapmanız gereken, öncelikle opcode bölümünü kontrol edeceksiniz. Tablo 6 te verilen değerlerden bir tanesi gelmiş ise ona göre Tablo 1,2,3 veya 4 e göre geri kalan bitleri ayıracaksınız. Eğer Tablo 6 te verilen değerlerden farklı bir değer gelirse hata biti 1 olacak (default olarak 0)

● Eğer R tipi bir komut geldiyse rs1, rs2 ve rd değerlerine direkt atama yapacaksınız, aluop çıkışına da fonksiyonda gösterilen formüle göre oluşturup atama yapacaksınız. (bit30, bit14, bit13, bit12). imm çıkışı 0 olacak.

● Eğer I tipi bir komut geldiyse rs1 ve rd değerlerine direkt atama yapacaksınız, aluop çıkışına da fonksiyon bitlerini soldan 0 ekleyerek atama yapacaksınız. (0, bit14, bit13, bit12). imm çıkışı imm12 olacak. diğer çıkışlar 0 olacak.

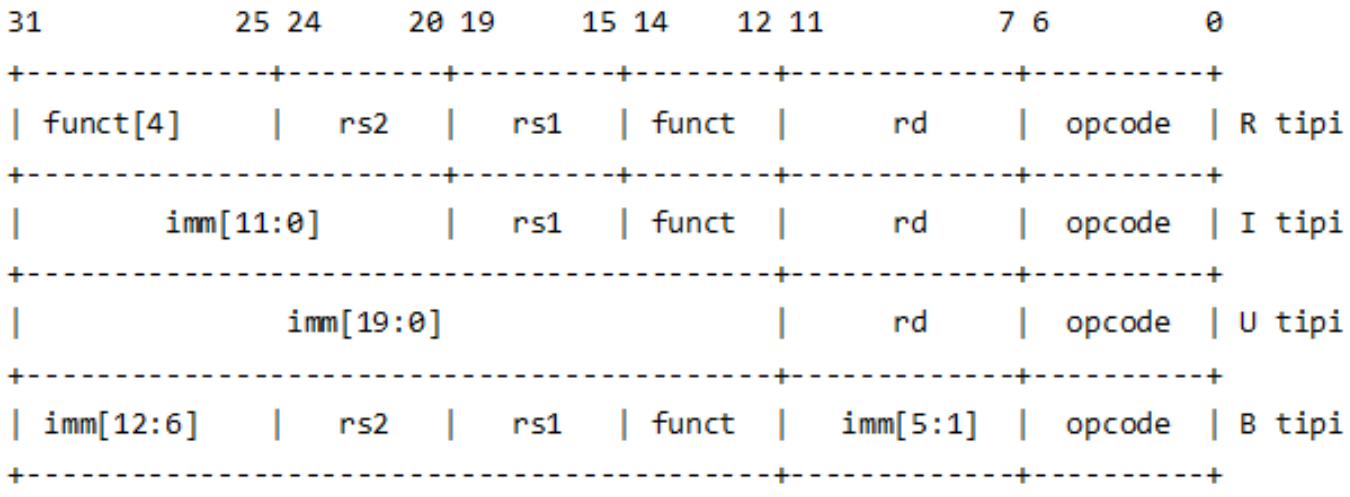
● Eğer U tipi bir komut geldiyse rs1 ve rd değerlerine direkt atama yapacaksınız. imm çıkışı imm20 olacak. diğer çıkışlar 0 olacak.

● Eğer B tipi bir komut geldiyse rs1, rs2 değerlerine direkt atama yapacaksınız, aluop çıkışına da fonksiyonda gösterilen formüle göre oluşturup atama yapacaksınız. (0, bit14, bit13, bit12). imm çıkışı LSB si 0 olacak şekilde 13 bitlik imm13 olacak.

Not: rs1_data ve rs2_data değerlerine her zaman 0 atayın.

A. Bu problemin testbench inde, her bir komut tipi için birkaç bit vektörü oluşturup test edin.

B. Kapsamlı bir test vektörü oluşturun ve bu testvektörünü kullanarak test edin



Şekil 1 Komut tiplerinin kapsamlı gösterimi.

İstenen komut ayırıcı devrenin kodları aşağıdaki gibidir;

```
module lab6_g29_p1 (
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'b0;
assign rs2_data = 32'b0;
always_comb
begin

hata = 0;



    if(opcode == 7'b0000001)
            begin
            rs1 = komut[19:15];
            rs2 = komut[24:20];
```

```
        rd =  komut[11:7] ;
        aluop = {komut[30],komut[14],komut[13],komut[12]};
        imm = 32'b0;
 end

else if(opcode == 7'b0000011)
        begin
        rs1 = komut[19:15];
        rd =  komut[11:7] ;
        aluop = {1'b0,komut[14],komut[13],komut[12] };
        imm = {20'b0 , komut[31:20] };
        rs2 = 5'b0;
 end

else if(opcode == 7'b0000111)
        begin
        rd =  komut[11:7] ;
        imm = {12'b0 , komut[31:12] };
        rs1 = 5'b0;
        rs2 = 5'b0;
        aluop = 4'b0;
 end

else if(opcode == 7'b0001111)
        begin
        rs1 = komut[19:15];
        rs2 = komut[24:20];
```

```
            aluop = {1'b0,komut[14],komut[13],komut[12]};
            imm = {19'b0,komut[31:25],komut[11:7],1'b0};
            rd = 5'b0;

        end

            else
            begin

            hata = 1;
            rs1 = 5'b0;
            rs2 = 5'b0;
            aluop = 4'b0;
            imm = 32'b0;
            rd = 5'b0;

            end
end
endmodule
```

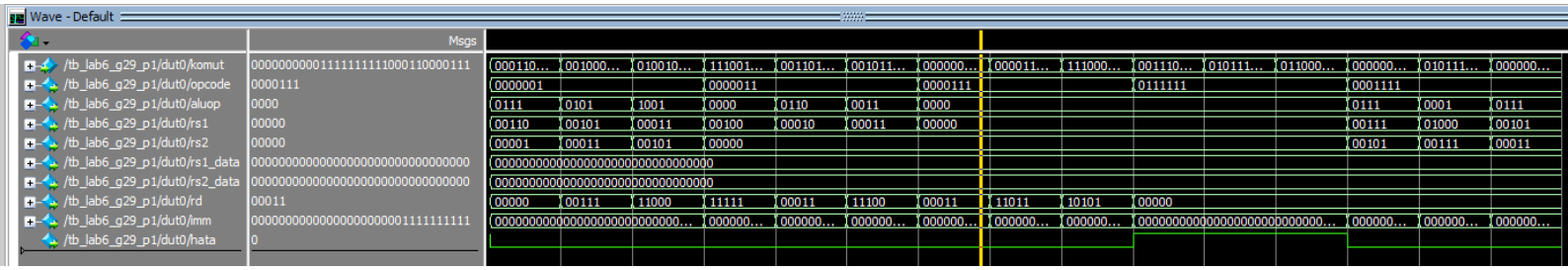
Yazılan kodun testbench kodları ise aşağıdaki tablodan incelenebilir.

```
`timescale 1ns/1ps


module tb_lab6_g29_p1 ();
  logic [4:0] rs1   ;
  logic [4:0] rs2   ;
  logic [31:0] komut;
```

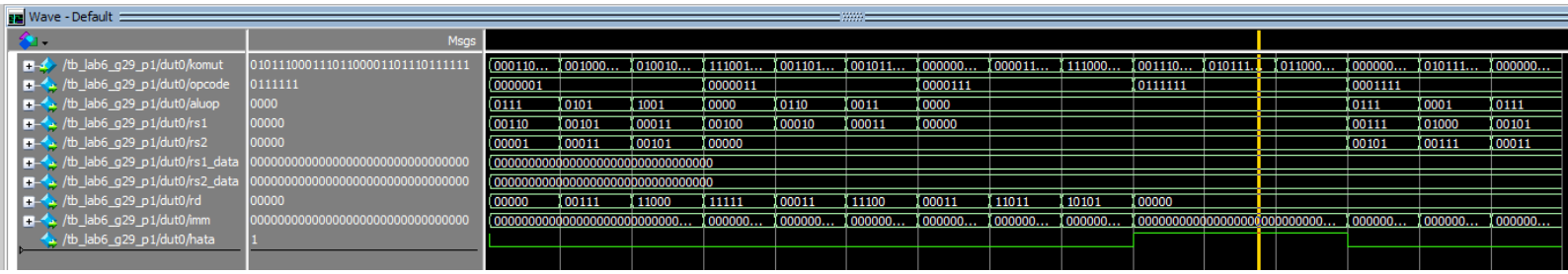
```
  logic [6:0] opcode;
  logic [3:0] aluop ;
  logic [4:0] rd    ;
  logic [31:0] imm  ;
  logic [31:0] rs1_data;
  logic [31:0] rs2_data;
  logic hata;

lab6_g29_p1 dut0(
 .rs1(rs1), .rs2(rs2), .komut(komut), .opcode(opcode), .aluop(aluop), .rd(rd), .imm(imm),
.rs1_data(rs1_data), .rs2_data(rs2_data), .hata(hata)
);
initial begin
komut = 32'b0_0_1101_00001_00110_111_00000_0000001;#10;//R Tipi
komut = 32'b0_1_0001_00011_00101_101_00111_0000001;#10;//R Tipi
komut = 32'b1_0_0100_00101_00011_001_11000_0000001;#10;//R Tipi

komut = 32'b111001111000_00100_000_11111_0000011;#10;//I Tipi
komut = 32'b001101111100_00010_110_00011_0000011;#10;//I Tipi
komut = 32'b001011001110_00011_011_11100_0000011;#10;//I Tipi

komut = 32'b00000000001111111111_00011_0000111;#10;//U Tipi
komut = 32'b00001110001111110001_11011_0000111;#10;//U Tipi
komut = 32'b11100000001100100011_10101_0000111;#10;//U Tipi

komut = 32'b0011100_01101_10111_111_10011_0111111 ;#10;// Hata
komut = 32'b0101110_00111_01100_001_10111_0111111 ;#10;// Hata
```

```
komut = 32'b0110001_10011_01101_111_01011_0111111 ;#10;// Hata


komut = 32'b0000000_00101_00111_111_00011_0001111 ;#10;// B Tipi

komut = 32'b0101110_00111_01000_001_10111_0001111 ;#10;// B Tipi

komut = 32'b0000000_00011_00101_111_01011_0001111 ;#10;// B Tipi

$stop;

end


endmodule
```

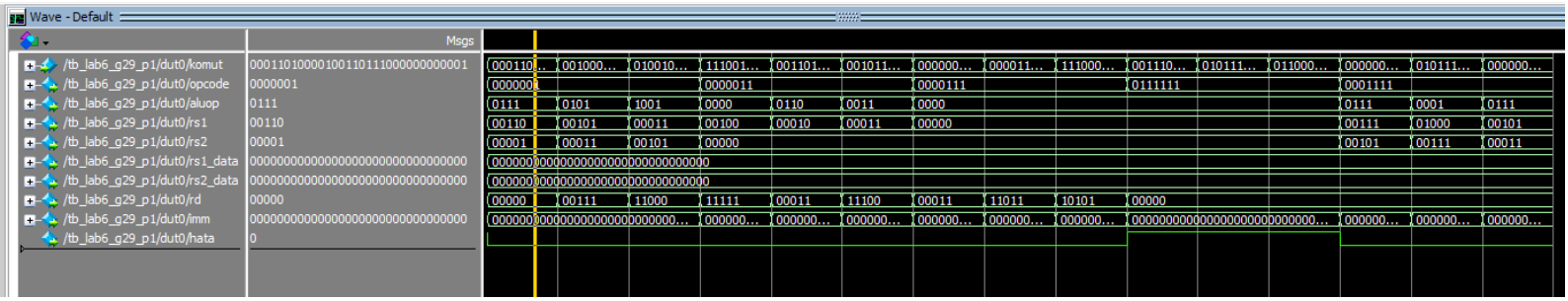
Yukarıdaki testbench dosyasında yazılan kodlar da sırasıyla tüm işlemler test edilmiştir. Sırasıyla R tipi, I tipi, U tipi, Hata ve B tipi komutları test edilmiştir. Bu testbench simüle edildikten sonra çıkan dalga formundan yorumlanacağı üzere yazılan kod amacını başarılı bir şekilde gerçekleştirmiştir. İlgili sinyal dalga formu aşağıda görüldüğü gibidir;



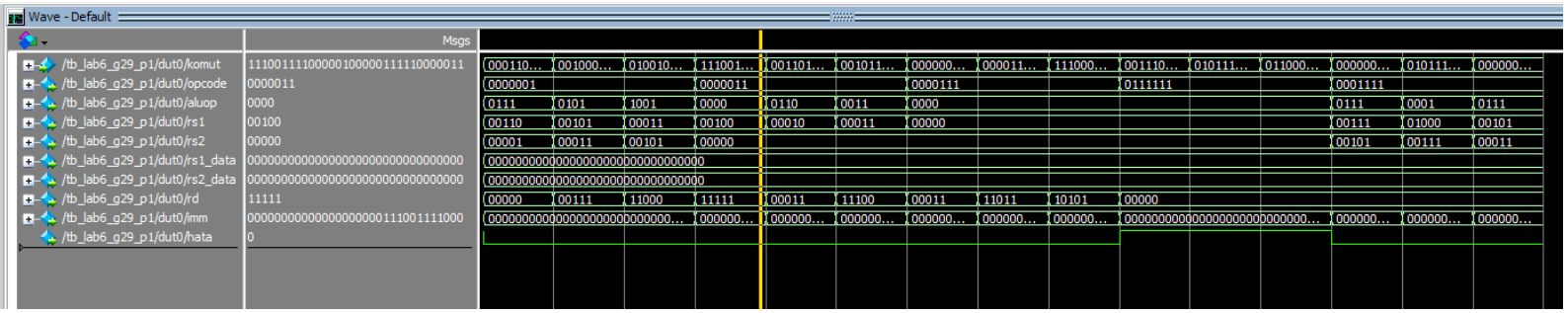
Şekil 2 U tipi operasyon için dalga formunda alınan değerler.



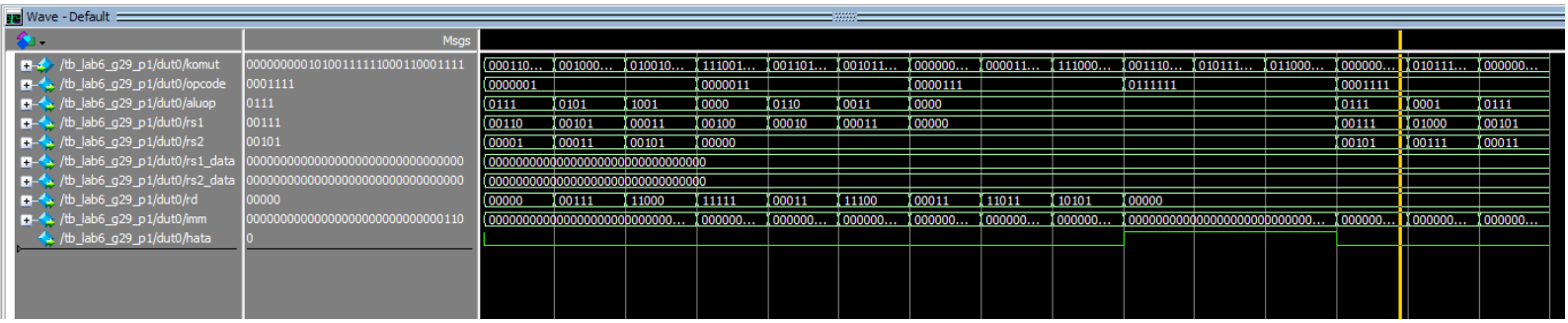
Şekil 3 Hata bloğunun çalıştığını gösteren dalga formu.



Şekil 3 R tipi komut için dalga formu ekran görüntüsü.

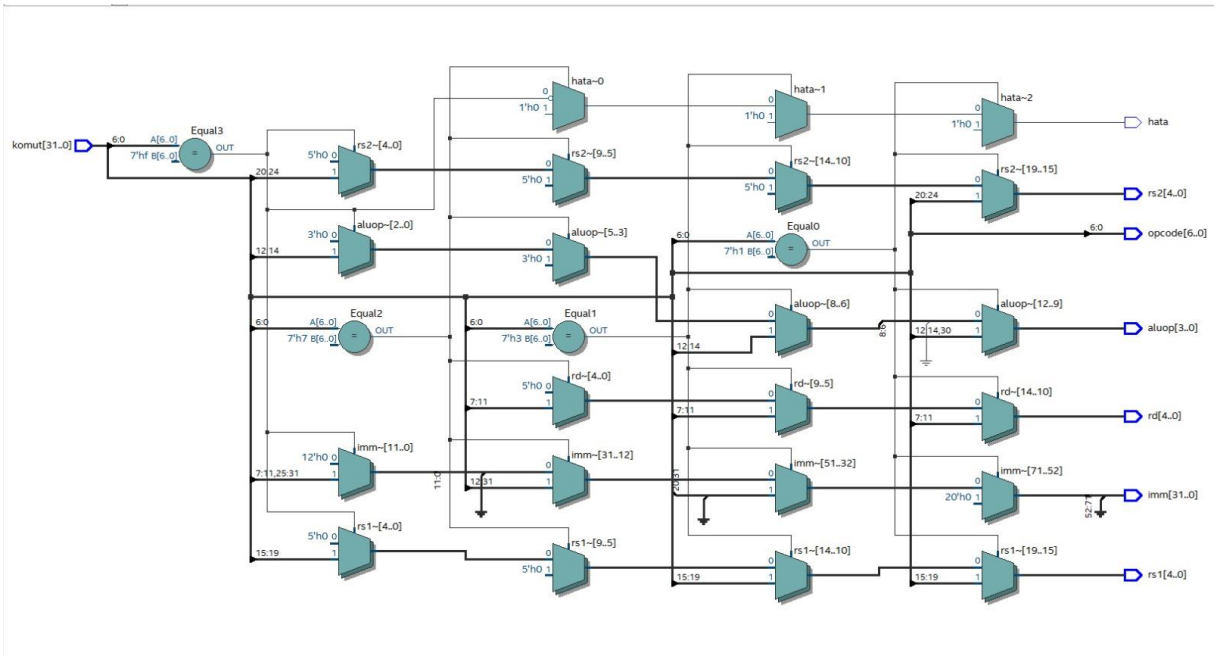


Şekil 4 I tipi komut için dalga formu ekran görüntüsü.

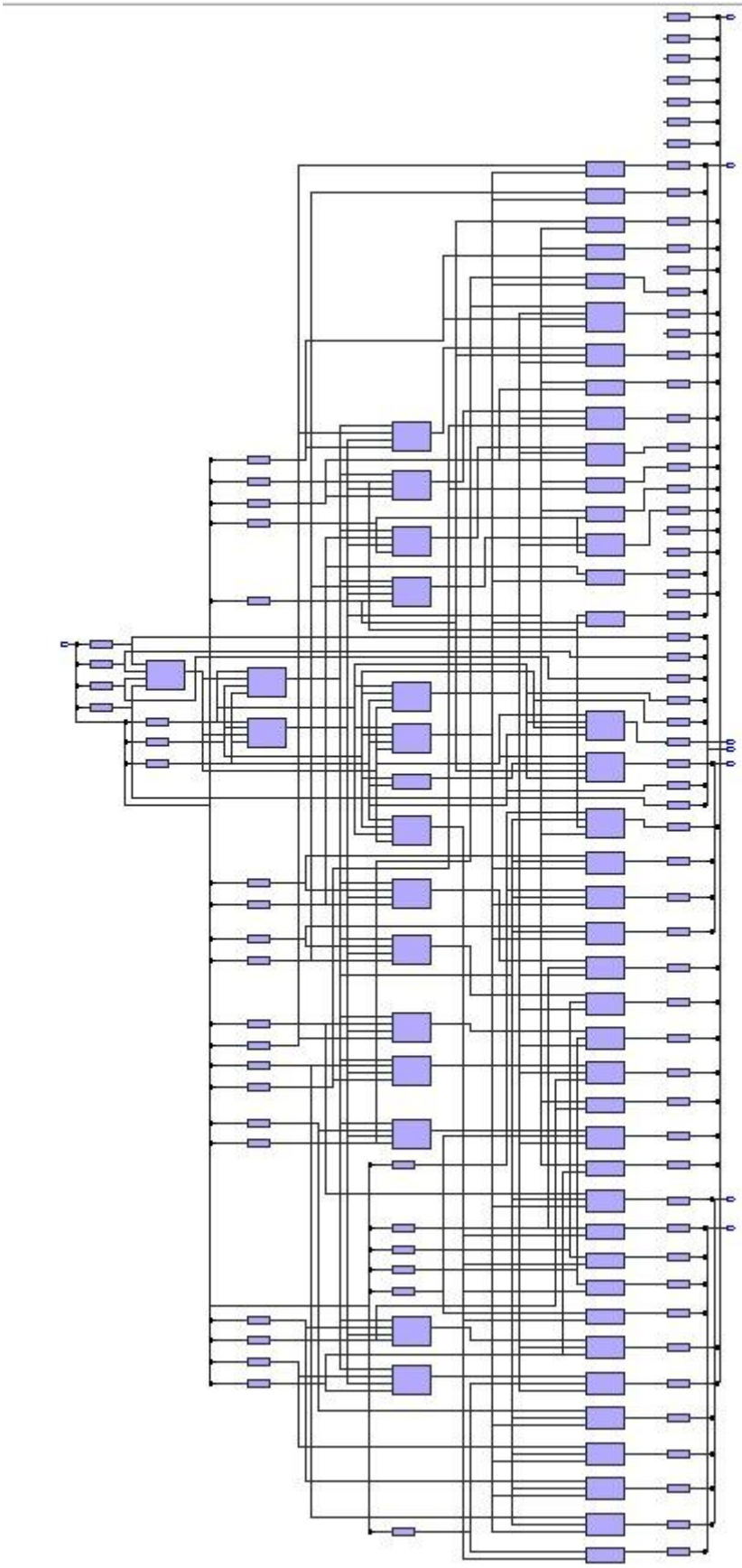


Şekil 5 B tipi komut için dalga formu ekran görüntüsü.

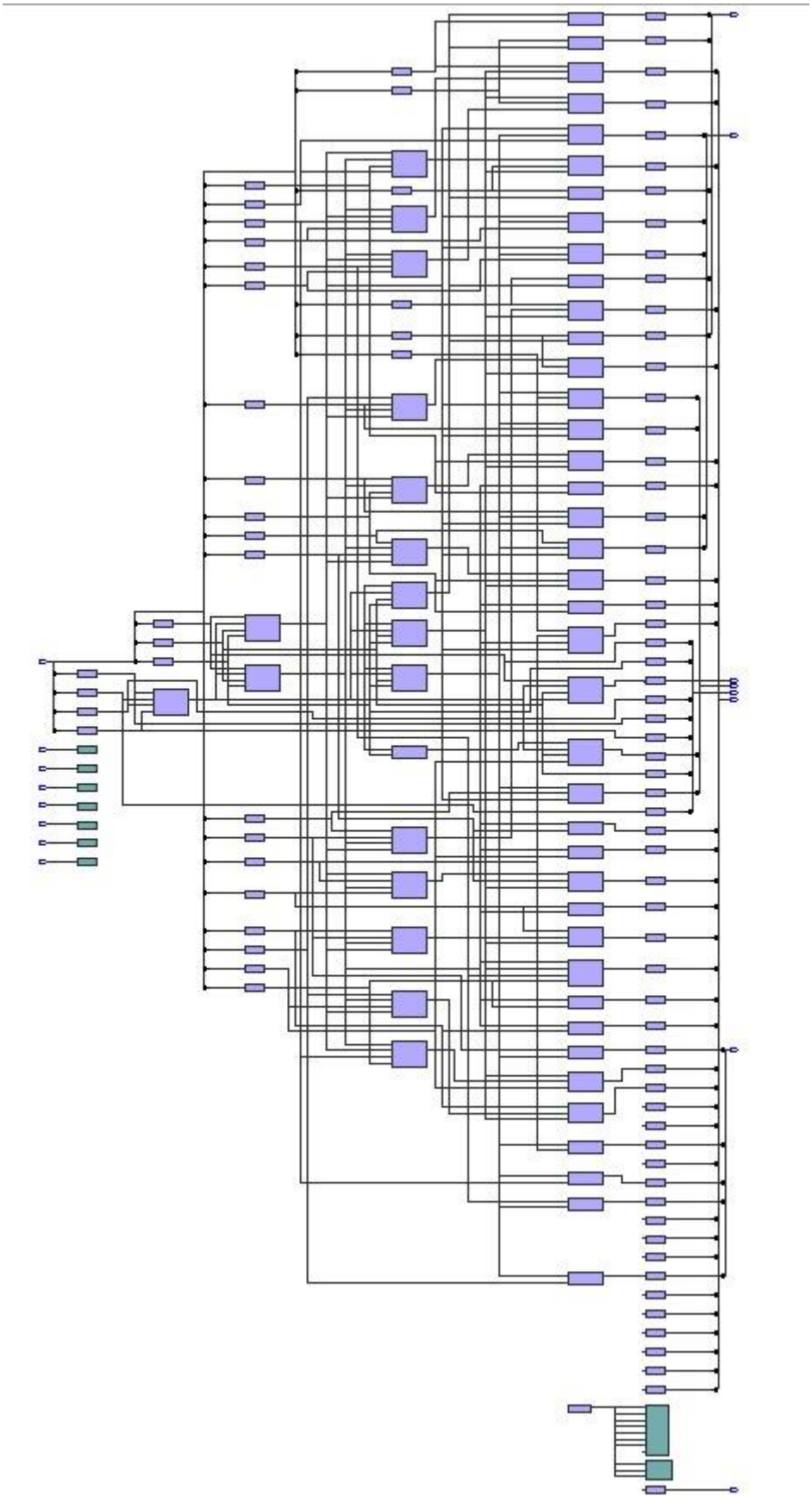
Yukarıdaki sinyal dalga formu ekran görüntülerinden yorumlanabileceği üzere yazılan kod ve test vektörleri kapsamlı bir şekilde bütün komut tiplerini kendi içlerinde ayırmış ve belirtilen opcode'lar harici bir komut geldiğinde ise hata sinyalini gösterip geriye kalan bütün istenen sinyalleri 0 olarak düzenlemiştir.



Şekil 5 Devrenin RTL şeması



Şekil 6 Devrenin post mapping şeması



Şekil 7 Devrenin Post fitting şeması

	Resource	Usage
1	Estimated Total logic elements	58
2		
3	Total combinational functions	58
4	⌄ Logic element usage by number of LUT inputs	
1	-- 4 input functions	21
2	-- 3 input functions	19
3	-- <=2 input functions	18
5		
6	⌄ Logic elements by mode	
1	-- normal mode	58
2	-- arithmetic mode	0
7		
8	⌄ Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0
9		
10	I/O pins	91
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	Equal1~0
15	Maximum fan-out	20
16	Total fan-out	315
17	Average fan-out	1.31

Şekil 8 Resource Usage Summary

Devrede toplamda 58 adet lojik eleman ve 91 adet pin kullanılmıştır.

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements
1	|lab6_g29_p1	58 (58)	0 (0)	0	0	0

Şekil 9 Resource utilization report.

DSP 9x9	DSP 18x18	Pins	Virtual Pins	ADC blocks	Full Hierarchy Name	Entity Name	Library Name
0	0	91	0	0	|lab6_g29_p1	lab6_g29_p1	work

Şekil 10 Resource Utility by Entity.

Genel Sonuç ve Yorumlar;

Bu laboratuvar föyünde tasarlanması istenen komut ayırıcı devre bütün isterleri başarılı bir biçimde yerine getirebilecek halde tasarlanmıştır. Tasarlanan komut ayırıcı birimi gelen bütün fonksiyonları başarılı bir biçimde sınıflandırabilmiş ve sinyal olarak dalga formunda gözlemlenmesi sağlanmıştır. Bu deney sonucunda komut ayırıcı devreler ve komutların detayları öğrenilmiştir.

Referanslar;

[1] https://github.com/fcayci/sv-digital-design

[2]https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/hdl/vlog/vlog_pro_ram_inferred.htm

[3] Logic Gates Classes by Furkan Cayci [2020-2021] Lecture 10

[4] Harris D.M., Harris S.L. - Digital Design and Computer Architecture (2016).pdf