



GEBZE TEKNİK ÜNİVERSİTESİ
ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x2 Deney Raporu

Lojik Devreler ve Tasarım Laboratuvarı Dersine Hazırlık

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

1. Problemler

1.1. Problem I – Hafıza Elemanları Karşılaştırması

HDL ifadelerini kullanarak Latch, Rising-edge triggered Flip-Flop ve Falling-edge triggered Flip-Flop oluşturuldu. Her birinin girişine aynı sinyal gelecek şekilde tasarlandı ve çıkışları ayrı sinyallere bağlandı.

```
/*lab2_g29_p1.sv
 *Hazırlayanlar:
 *  Ruveyda Dilara Günal
 *  Alican Bayındır
 */
module lab2_g29_p1(
    input logic clk, D,
    output logic [2:0] q
);

    always_latch
    begin
        if(clk) q[0] <= D;
    end

    always @(posedge clk)
    begin
        q[1] <= D;
    end

    always @(negedge clk)
    begin
        q[2] <= D;
    end

endmodule
```

Testbench oluşturularak, devre test edilmiştir.

```
// tb_lab2_g29_p1

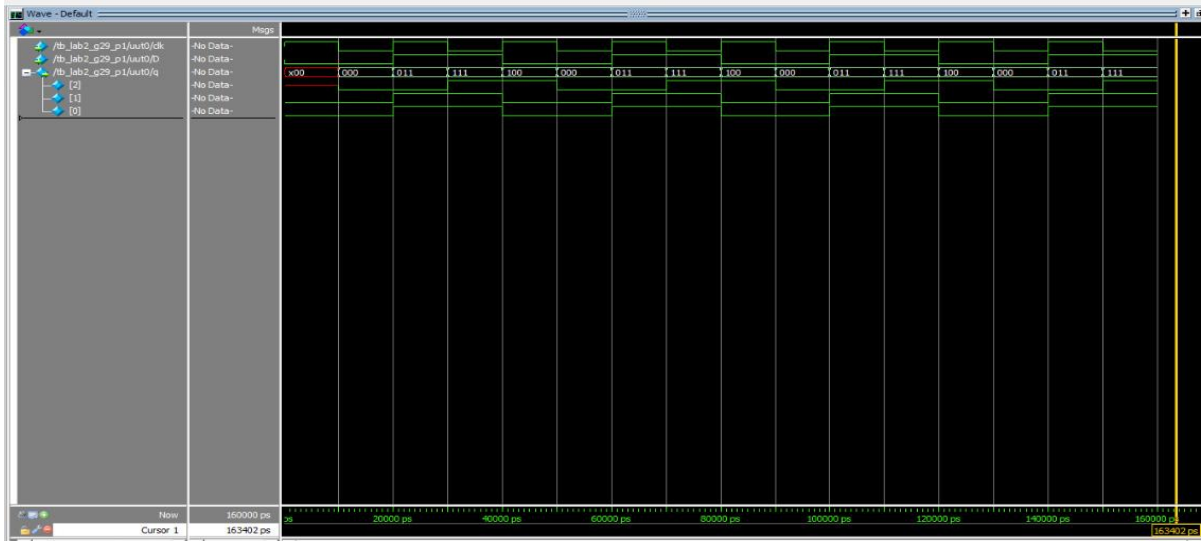
`timescale 1ns/1ps
module tb_lab2_g29_p1 ();
    logic D;
    logic [2:0] q;
    logic clk;

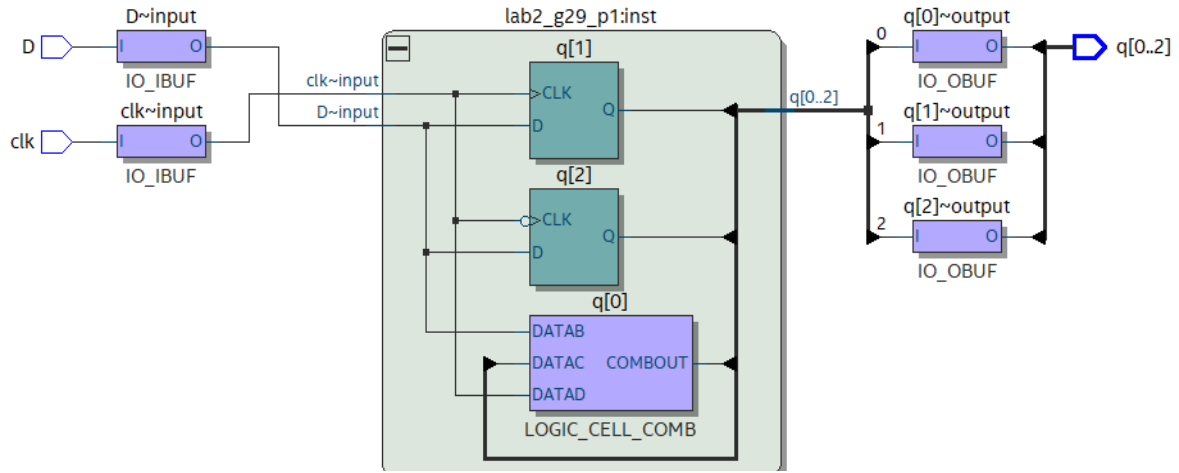
    lab2_g29_p1 uut0(.D(D), .q(q), .clk(clk));
    always begin
        clk = 1; #10;
        clk = 0; #10;
    end
end
```

```

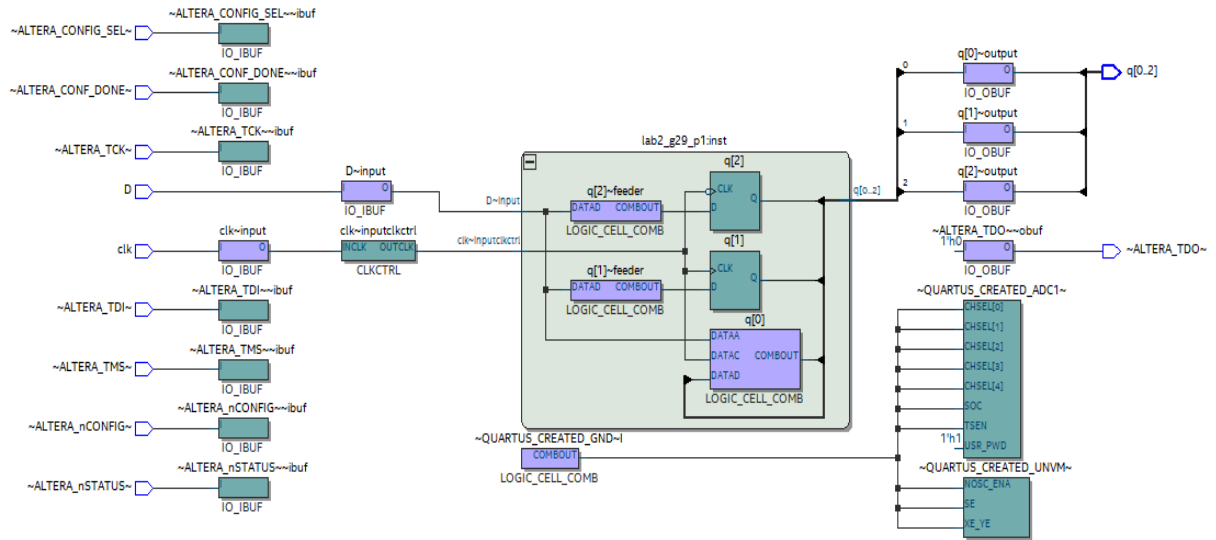
initial begin
    D = 0; #20;
    D = 1; #20;
    D = 0; #20;
    D = 1; #20;
    D = 0; #20;
    D = 1; #20;
    D = 0; #20;
    D = 1; #20;
    $stop;
end
endmodule

```





Şekil 3: Quartus Çıktısı - Post Mapping



Şekil 4: Quartus Çıktısı - Post fitting

Yukarıdaki tasarlanan devrede her bir eleman(latch, flip floplar) tek bir D sinyaline bağlandı. Sonrasında bu sinyallerin her birisi istenen 3 adet kombinyonel lojiklere bağlanarak çıkış için q[0,1,2] çıkışlarına iletildi. Bütün bu işlemlere sonucunda

Q[0] -> D latch,

Q[1] -> posedge flip flop,

Q[2] -> negedge flip flop'un

Çıkışlarını temsil etmektedir.

1.2. Problem II – ALU Tasarımı

```
/*lab2_g29_p2.sv
 *Hazırlayanlar:
 * Ruveyda Dilara Günal
 * Alican Bayındır
 */

module lab2_g29_p2 (
    input logic [31:0] a, b,
    input logic [2:0] op,
    output logic [31:0] res,
    output logic n, z, v, c, cout,
    output logic hata
);
    reg [32:0] o;

    always_comb
    begin
        if (op == 3'b000)
            {o,res} = a + b;

        else if (op==3'b001)
            {o,res} = a - b;

        else if (op==3'b010)
            res = a ^ b;

        else if (op==3'b011)
            res = a | b;

        else if (op==3'b100)
            res = a << ((b[4]*2*2*2*2) + (b[3]*2*2*2) + (b[2]*2*2)
+ (b[1]*2) + (b[0]*1));

        else if (op==3'b101)
            res = a >> ((b[4]*2*2*2*2) + (b[3]*2*2*2) + (b[2]*2*2)
+ (b[1]*2) + (b[0]*1));

        else if (op==3'b110)
            res = a>>> (b[0] + 2*b[1] + 4*b[2] + 8*b[3] + 16*b[4]);

        else if (op==3'b111)
            res = a & b;
        else
            hata = 1; //3 bit olarak tüm sayılar temsil edildiği
ve bu satırdaki koddan dolayı hata biti hep 1 olacaktır.
        end

    always_comb
    begin
```

```

        if(a[31] == 0 && b[31] == 0 && res [31] == 1 || a[31] == 1
        && b[31] == 1 && res[31] == 0)
            v = 1 ;
        else
            v = 0;

        if(res == 32'd0)
            z = 1;
        else
            z = 0;

        if (res [31] == 1)
            n = 1 ;
        else
            n = 0;

        if (cout == 1 && op ==3'b000 || cout == 1 && op == 3'b001
        || cout == 1 && op == 3'b100)
            c = 1;
        else
            c = 0;
        end
    endmodule

```

```

// tb_lab2_g29_p2

`timescale 1ns/1ps

module tb_lab2_g29_p2 ();
    logic [31:0] a,b ;
    logic [31:0] res ;
    logic [2:0] op ;
    logic v , z , n , c;

    lab2_g29_p2 uut0(.a(a), .b(b), .op(op), .res(res),
    .hata(hata), .v(v), .z(z), .n(n), .c(c));

    initial begin
        a      = 32'b10011100110000110101000000111011;    b      =
        32'b11100111000110111100001001001001; op = 3'b000; #10 ;
        a      = 32'b10011100110000110101000000111011;    b      =
        32'b11100111000110111100001001001001; op = 3'b001; #10 ;
        a      = 32'b10011100110000110101000000111011;    b      =
        32'b11100111000110111100001001001001; op = 3'b010; #10 ;
        a      = 32'b10011100110000110101000000111011;    b      =
        32'b11100111000110111100001001001001; op = 3'b011; #10 ;
        a      = 32'b10011100110000110101000000111011;    b      =
        32'b11100111000110111100001001001001; op = 3'b100; #10 ;
        a      = 32'b10011100110000110101000000111011;    b      =
        32'b11100111000110111100001001001001; op = 3'b101; #10 ;
    end

```

```

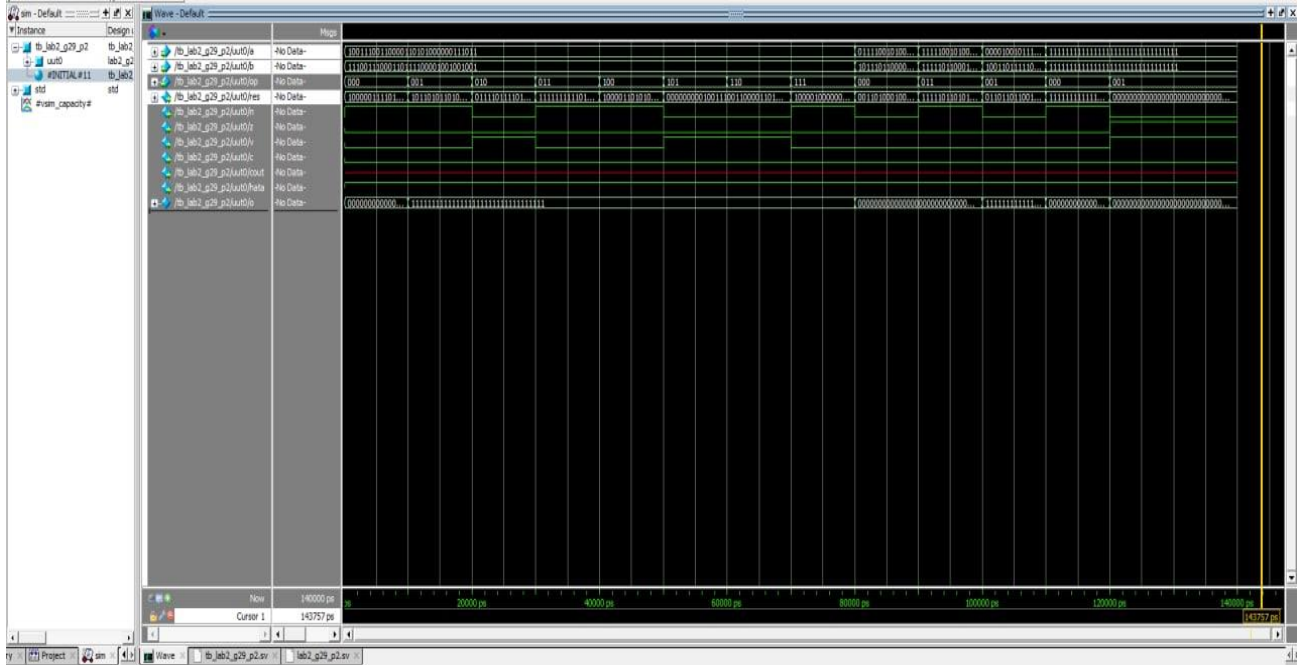
a      =      32'b10011100110000110101000000111011;      b      =
32'b11100111000110111100001001001001; op = 3'b110; #10 ;
a      =      32'b10011100110000110101000000111011;      b      =
32'b11100111000110111100001001001001; op = 3'b111; #10 ;

a      =      32'b01111001010000011000001010011010;      b      =
32'b101110110000000011001101000010011; op = 3'b000; #10 ;
a      =      32'b11111001010000011001101010011010;      b      =
32'b11111011000110011001101010010111; op = 3'b011; #10 ;
a      =      32'b00001001011110011000001010011010;      b      =
32'b10011011111000011001101001100000; op = 3'b001; #10 ;

a      =      32'b11111111111111111111111111111111;      b      =
32'b11111111111111111111111111111111; op = 3'b000; #10 ;
a      =      32'b11111111111111111111111111111111;      b      =
32'b11111111111111111111111111111111; op = 3'b001; #10 ;

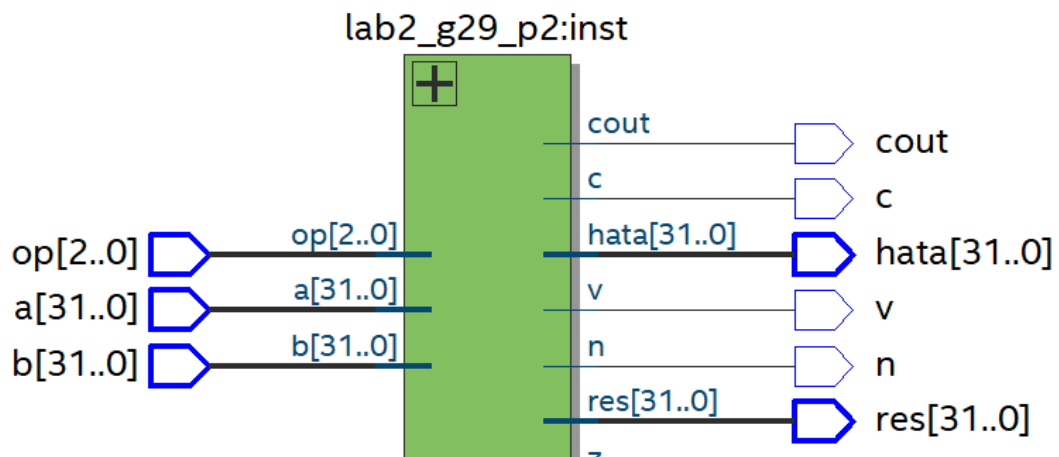
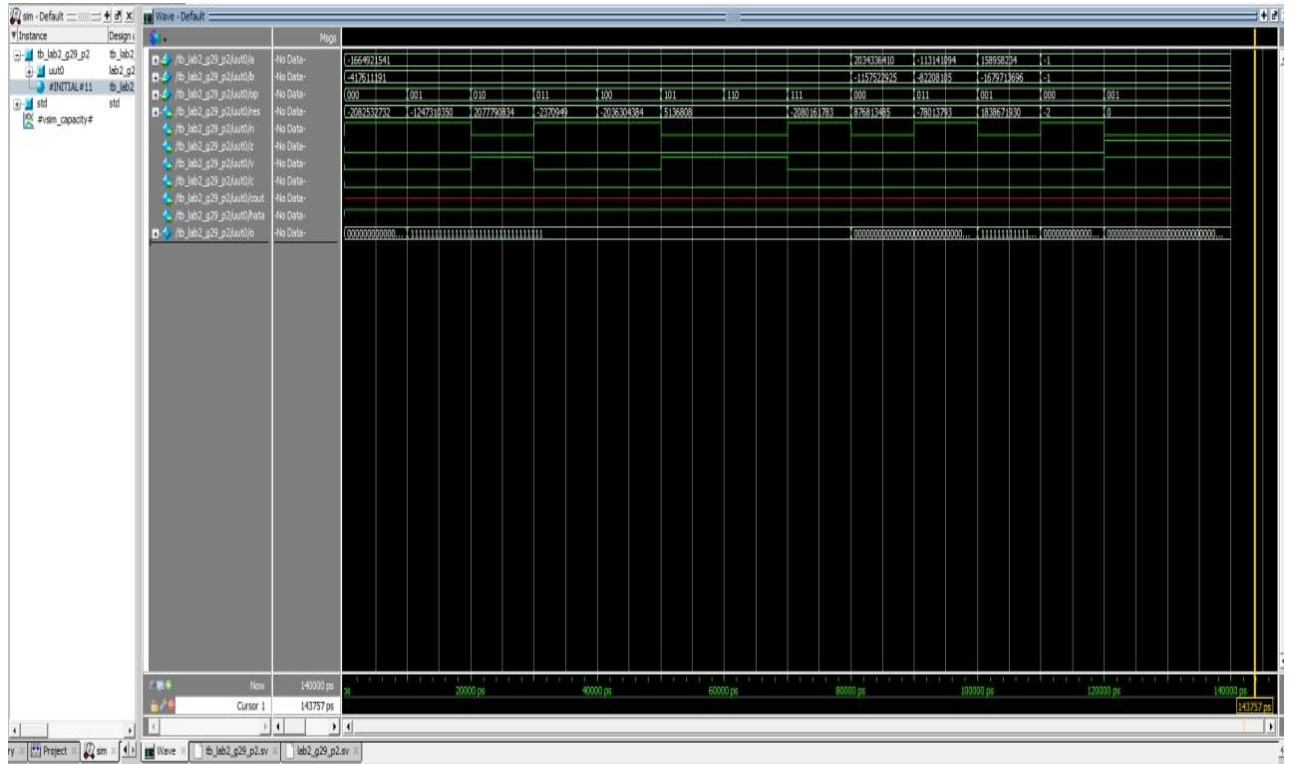
a      =      32'b11111111111111111111111111111111;      b      =
32'b11111111111111111111111111111111; op = 9; #10 ;
$stop ;
end
endmodule

```

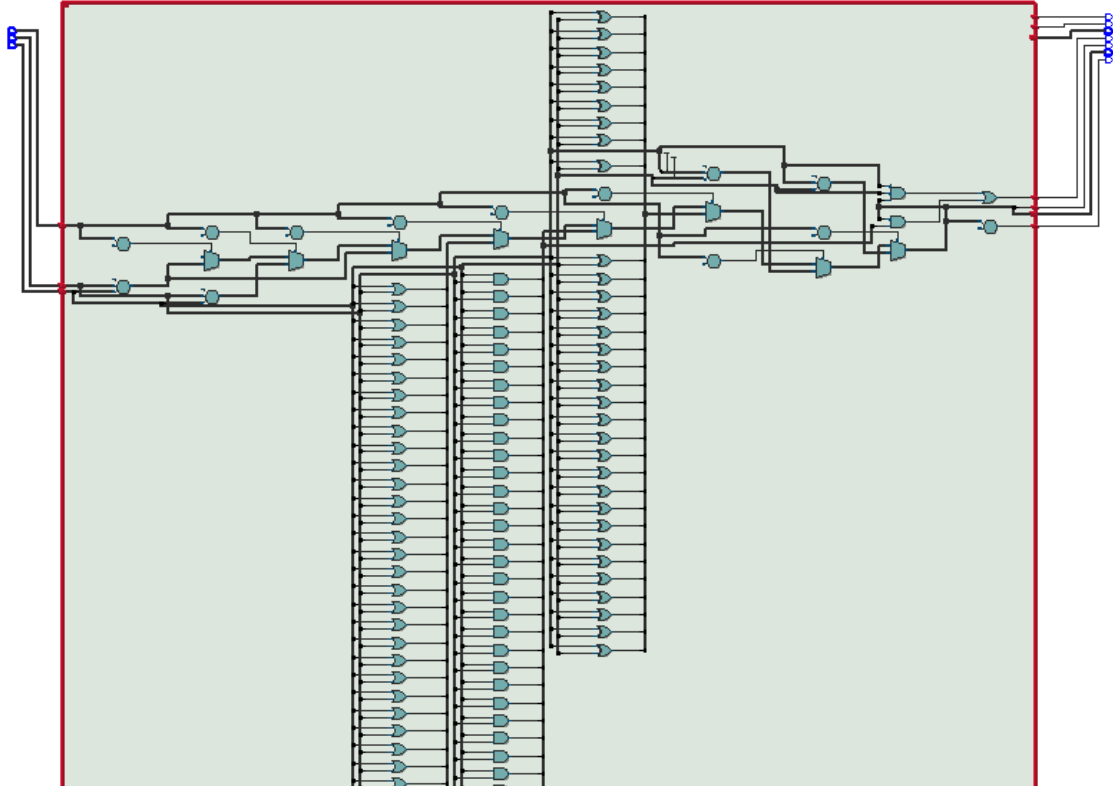


Şekil 5: Modelsim Çıktısı Binary

Sonuçlar daha kolay kontrol için binary'den decimal'e çevrildiğinde aşağıdaki sonuçlar elde edilmiştir.



Şekil 6: Modelsim Çıktısı Decimal



Şekil 8: Quartus Çıktısı

ALU devresinin tasarlanması için ilk olarak ödev föyünde verilen modül tanımı ve değişken isimleri alınmıştır. Sonrasında ise oluşturulan combinational logic bloğu sayesinde tablo 1 de verilen op-code'lara uygun şekilde devrenin çalışması sağlanmıştır. 2. Combinational logic bloğunda ise flag koşulları belirlenmiş, tanımlanmış ve devreye uygulanmıştır. Son olarak ise testbench yazılmış ve devrenin çalışır durumda olduğu teyit edilmiştir (şekil 5-6). Bu testbenchte ilk önce aynı sayılar üzerinde farklı işlemler denenmiş daha sonra farklı sayılar üzerinde farklı işlemler denenmiş ve son olarakta hata bitinin 1 olması için uğraşılmıştır. Hata bitinin hep 1 olma sebebi ise 2^3 kadar olasılığın hepsini kullanmamızdan dolayıdır. Belirlediğimiz op-code'lar 3 bitlik olduğu için her zaman 3 bitten fazla gelen sayıyı overflow edip 3 bite düşürecek ve ona göre işlemi yapacaktır. Bundan dolayı koddaki else bloğu hata bitini hep 1 yapmaktadır ancak bu bir yanlış sinyaldir. Çözümü için o satırı yorum satırı yapmak yeterlidir veya 4 bitlik op-code kullanarak kullanılmayan $2^4 - 2^3$ kadar ihtimalde "hata" bitini aktif konuma getirmek yeterlidir (n bit için $2^n - 2^3$).

Tablo 1

op ²	operasyon	açıklama
000	addition	A + B
001	subtraction	A - B
100	shift left logical	A sinyalini B kadar sola kaydır
010	xor	A XOR B
101	shift right logical	A sinyalini B kadar sağa kaydır1 (0 padded)
110	shift right arithmetic	A sinyalini B kadar sağa kaydır1 (sign padded)

011	or	A OR B
111	and	A AND B

1.3. Problem III – Senkron Tasarım ve Zamanlama

```

/*lab2_g29_p3.sv
*Hazırlayanlar:
* Ruveyda Dilara Günal
* Alican Bayındır
*/

module lab2_g29_p3 (
    input logic [31:0] a,b,
    input logic clk ,
    input logic [2:0] op,
    output logic [31:0] result, hata,
    output logic v, z, n, carry, cout
);
    logic [31:0] a_reg, b_reg, hata_reg;
    logic [31:0] result_reg;
    logic [2:0] op_reg;
    logic v_reg, z_reg, n_reg, carry_reg, cout_reg;

    always_comb
    begin
        if(op_reg == 3'b000)
            {cout_reg,result_reg} = a_reg + b_reg;

        else if (op_reg == 3'b001)
            {cout_reg,result_reg} = a_reg + (~b_reg+1);

        else if (op_reg == 3'b010)
            {cout_reg,result_reg} = (a_reg
<<((1)*b_reg[0]+(2)*b_reg[1]+(4)*b_reg[2]+(8)*b_reg[3]+(16)*
b_reg[4]));

        else if (op_reg == 3'b011)
            result_reg = a_reg ^ b_reg;

        else if (op_reg == 3'b101)
            {result_reg,cout_reg} = a_reg >>
((1)*b_reg[0]+(2)*b_reg[1]+(4)*b_reg[2]+(8)*b_reg[3]+(16)*b_
reg[4]);

        else if (op_reg == 3'b110)
            result_reg = a_reg >>>
((1)*b_reg[0]+(2)*b_reg[1]+(4)*b_reg[2]+(8)*b_reg[3]+(16)*b_
reg[4]);

        else if (op_reg == 3'b011)
            result_reg = a_reg | b_reg;
    end

```

```

else if (op_reg == 3'b111)
    result_reg = a_reg & b_reg;

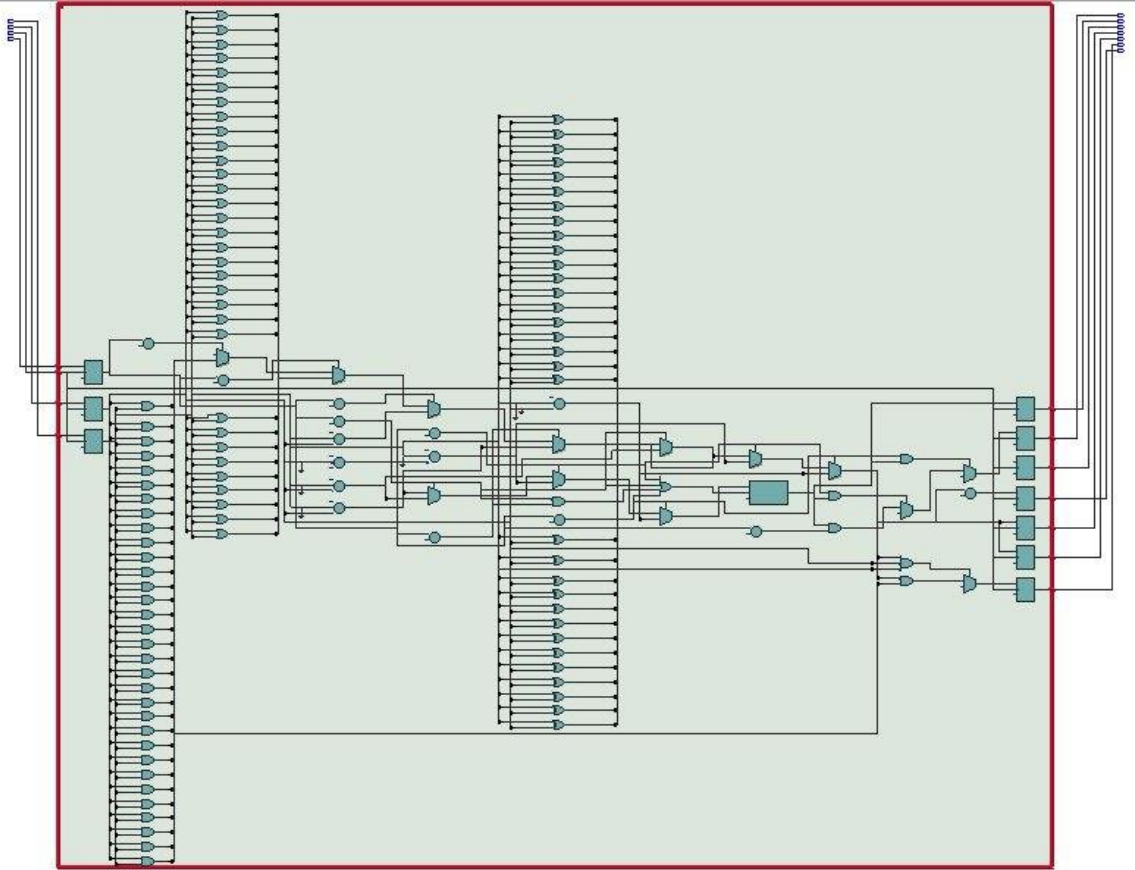
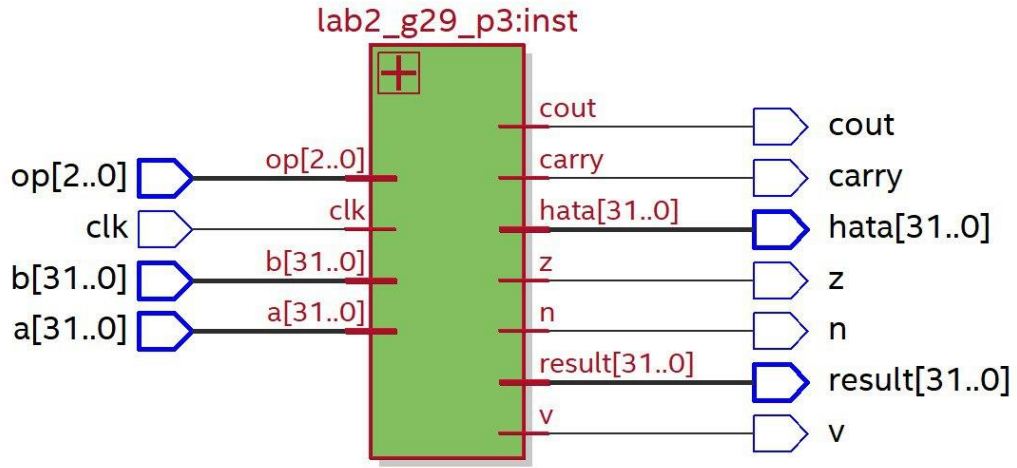
else
    result_reg = hata_reg;

if(a_reg[31] == 0 & b_reg[31] == 0 & result_reg[31] == 1)
    v_reg = 1;
else if (a_reg[31] == 1 & b_reg[31] == 1 & result_reg[31]
== 0)
    v_reg = 1;
else
    v_reg = 0;
if(result_reg == 32'd0)
    z_reg = 1;
else
    z_reg = 0;
if (result_reg[31] == 1)
    n_reg = 1;
else
    n_reg = 0;


if (cout_reg==1 & op_reg == 3'b000)
    carry_reg = 1;
else if (cout_reg==1 & op_reg == 3'b001)
    carry_reg = 1;
else if (cout_reg==1 & op_reg == 3'b100)
    carry_reg = 1;
else
    carry_reg = 0;
end

always_ff @ (posedge clk)
begin
    a_reg <= a ;
    b_reg <= b;
    op_reg <= op;
    result <= result_reg;
    hata <= hata_reg;
    v <= v_reg;
    z <= z_reg;
    n<= n_reg;
    carry<=carry_reg;
    cout<=cout_reg;
end
endmodule

```




Şekil 9: Quartus Çıktısı

Slow 1200mV 85C Model Fmax Summary				
 <<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	80.27 MHz	80.27 MHz	clk	
2	299.58 MHz	299.58 MHz	lab2_g29_p3:inst op_reg[0]	

Şekil 10: Fmax

Slow 1200mV 85C Model Setup Summary			
 <<Filter>>			
	Clock	Slack	End Point TNS
1	clk	-11.458	-325.644
2	lab2_g29_p3:inst op_reg[0]	-8.917	-8.917

Şekil 11: Setup

Slow 1200mV 85C Model Hold Summary			
 <<Filter>>			
	Clock	Slack	End Point TNS
1	clk	0.139	0.000
2	lab2_g29_p3:inst op_reg[0]	0.141	0.000

Şekil 12: Hold

2. Sonuçlar ve Genel Yorumlar

3. problemde oluşturulan ALU devresine bir sonraki problemde giriş ve çıkışlarına birer register bağlanmıştır. Devre quartus prime programında test edilmiştir. Devrenin max. Frekansı 80.27 MHz olarak gözlemlenmiştir. Bu soruda nedeni bilinmeyen bir hatadan dolayı istenilen timing analiz gözlenememiştir. Fmax, Hold time ve Setup time gözlemlenmiş. Fmax Şekil 10'da, Setup Şekil 11'de ve Hold Şekil 12'de gösterilmiştir.

Deneyde half adder, full adder, ripple carry adder, latch, Rising-edge triggered Flip-Flop, Falling-edge triggered Flip-Flop, ALU ve pipelined ALU devrelerinin kullanımı öğrenilmiştir. Sorularda istenilen devreler istenilen sıra ile tasarlanmış ve testbench dosyaları ile test edilerek devrelerin çalışırılığı doğrulanmıştır. ALU probleminde verilen auto - testbench dosyası kullanılmaya çalışılmış ancak verilen testbench dosyası gerekli değişiklikleri yapamadığımızdan dolayı yazdığımız koda uyarlanılamamıştır (testbench yorumlarında "videoda yapıldığı gibi" diye cümleler var ancak elimizde video yok). Deneyde genel olarak Donanım tanıma dillerini (HDL) kullanarak devre tasarımı, Aritmetik devreleri ve structural tasarımı ve Kombinasyonel devreleri senkronize edebilme öğrenilmiştir.

Referanslar

- [1] Intel® Quartus® Prime Software Suite. URL: <https://www.intel.com.tr/content/www/tr/tr/software/programmable/quartus-prime/overview.html> Accessed: 21.02.2020
- [2] Quartus Prime Introduction Using Schematic Designs. URL: ftp://ftp.intel.com/Pub/fpgaup/pub/Intel_Material/16.0/Tutorials/Schematic/Quartus_II_Introduction.pdf Accessed: 21.02.2020
- [3]https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/msgs/msgs.htm#msgs/egdfx_pins_overlap_error.htm
- [4] <https://stackoverflow.com/questions/40503093/alu-design-error>
- [5]<https://upload.wikimedia.org/wikipedia/commons/thumb/c/c0/74181aluschematic.png/400px-74181aluschematic.png>
- [6] <https://www.nandland.com/verilog/examples/example-shift-operator-verilog.html>
- [7] <https://core.ac.uk/download/pdf/48532494.pdf>
- [8] <http://web.mit.edu/6.111/www/f2016/handouts/L09.pdf>
- [9] Harris, S. and Harris, D., 2015. Digital Design and Computer Architecture: ARM Edition. Morgan Kaufmann.