
GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

# LAB 0x4 Deney Raporu

Lojik Devreler ve Tasarım Laboratuvarı Dersi


| Hazırlayanlar |
| --- |
| 1) 171024086-Berat KIZILARMUT |
| 2) |

# 1. Giriş

Bu deneyde HDL kullanımından faydalanarak aritmetik devreler ve structural design yapacağız. Kombinasyonel lojik devrelere giriş yapacağız.

# 2. Problemler

## 2.1. Problem I – Hafıza elemanları karşılaştırması

### A. HDL Yazılışı

SystemVerilog dili ile ModelSim programında ilk devrenin dosyası hazırlandı. Latch, Rising Edge FF ve Falling Edge FF hepsi bir devrede tanımlandı.

```
/* lab_4_g41_p1.sv
 * Hazırlayanlar: Berat Kızılarmut
 * Notlar: Hafıza elemanlarının karşılaştırılması */
module lab4_g41_p1(
	input logic  d,clk,
	output logic latch,reff,feff);
	always_latch
	begin
		if(clk) latch<=d;
	end
	always_ff @(posedge clk)
	begin
		reff<=d;
	end
	always_ff @(negedge clk)
	begin
		feff<=d;
	end
endmodule
```

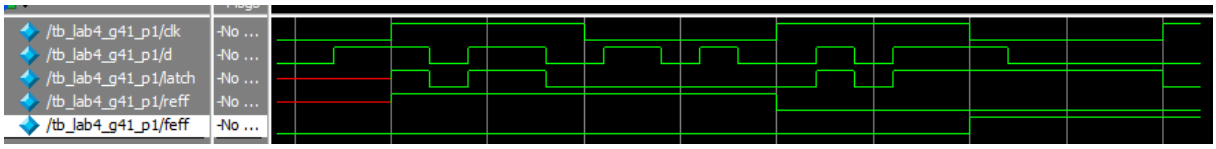
### B. Testbench Kurulumu

Kurulan devreyi sınamak clock belirlendi. D girişi rastgele şekillerde değiştirildi.

```
/* tb_lab_4_g41_p1.sv
 * Hazırlayanlar: Berat Kızılarmut
 * Notlar: Hafıza elemanlarının testbenchi */
module tb_lab4_g41_p1 ();
	logic d, clk;    logic latch, reff, feff;
	lab4_g41_p1 uut0(d,clk,latch,reff,feff);
	always
	 begin
	  clk = 0; #10; clk = 1; #10;
	 end
	initial begin
		d = 0; #7; d = 1; #5; d = 0; #2;
		d = 1; #4; d = 0; #3; d = 1; #3;
		d = 0; #2; d = 1; #2; d = 0; #4;
		d = 1; #2; d = 0; #2; d = 1; #6;
		d = 0; #10;        $stop;
	end
endmodule
```

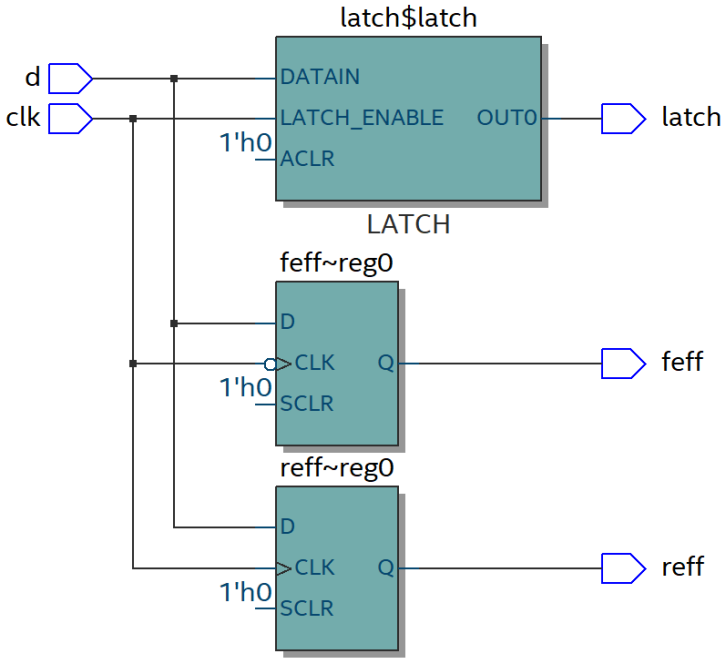
## C. Waveshape

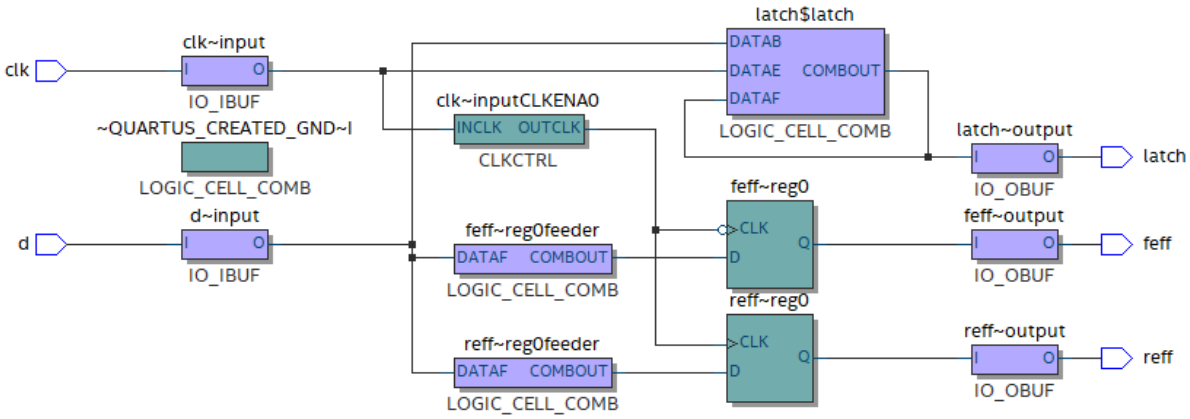
Testbench sonuçlarını gösteren dalga formu. Devre beklenildiği sonuçları vermiştir. Latch, clock aktif iken girişleri direkt hafızaya çekmiştir. Rising Edge FF, clock artışta iken hafızaya çekmiştir. Falling Edge FF, clock azalışta iken hafızaya çekmiştir.




## D. RTL ve Tech Map Şemaları

Devre Quartus'a aktarıldığında RTL ve Teknoloji Haritası görünüşleri aşağıdaki gibi olmaktadır.



### E. Problem I Yorumlar

Systemverilog yazılımımız beklenildiği gibi çalışmıştır. Dalga formunda da gözlenildiği gibi, değerler doğru zamanlarda hafızaya çekilmiştir. Quartus şemalarını incelediğimizde ise bu üç farklı hafızalama işlemi için de kendilerine özel olan paketlerin kullanıldığını görmekteyiz.

## 2.2 Problem II – Adder Tasarımı

### A. Half-adder Tasarımı

Half-adder'ın doğruluk tablosuna baktığımızda bir XOR kapısı gibi davrandığını görürüz. Bundan faydalanarak basit half adder devresi bu şekilde kuruldu.

```
/* lab_4_g41_p2_halfadder.sv
 * Hazırlayanlar: Berat Kızılarmut
 * Notlar: Half adder tasarımı*/
module lab4_g41_p2_halfadder(
      input logic a,b,
      output logic s,cout);
      assign s=a^b;
      assign cout=a&b;
endmodule
```

### B. Half-adder ile Full-adder Tasarımı

İlk adımdaki half-adder kullanılarak, 3 internal node yardımı ile full-adder tasarlandı.

```
/* lab_4_g41_p2_fulladder.sv
 * Hazırlayanlar: Berat Kızılarmut
 * Notlar: Half adder yardımı ile fulladder tasarımı
*/

module lab4_g41_p2_fulladder(
    input logic a, b, cin,
    output logic  s,cout);

    logic s1,c1,c2;
    lab4_g41_p2_halfadder ha1(a,b,s1,c1);
    lab4_g41_p2_halfadder ha2(s1,cin,s,c2);
    assign cout = c1 | c2;

endmodule
```

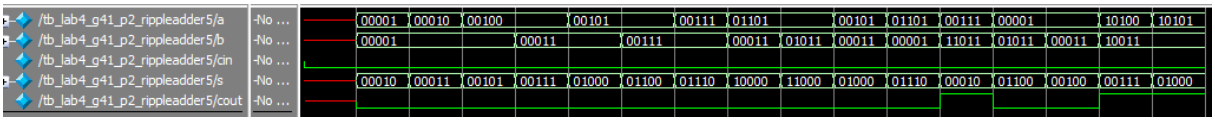
## C. 5-Bit Ripple Adder

Full-adderlar uçuca eklenerek, least significant bit'ten most significant'a doğru sıralı şekilde sıralı bir Ripple Carry Adder yapıldı.

```
        /* lab_4_g41_p2_rippleadder5.sv
         * Hazırlayanlar: Berat Kızılarmut
         * Notlar: Fulladder kullanımı ile ripple carry adder tasarımı */

module lab4_g41_p2_rippleadder5(
        input logic [4:0] a,b,
        input logic cin,
        output logic [4:0] s,
        output logic cout);
          logic [3:0] n;
lab4_g41_p2_fulladder fa1(a[0],b[0],cin,s[0],n[0]);
lab4_g41_p2_fulladder fa2(a[1],b[1],n[0],s[1],n[1]);
lab4_g41_p2_fulladder fa3(a[2],b[2],n[1],s[2],n[2]);
lab4_g41_p2_fulladder fa4(a[3],b[3],n[2],s[3],n[3]);
lab4_g41_p2_fulladder fa5(a[4],b[4],n[3],s[4],cout);
endmodule
```

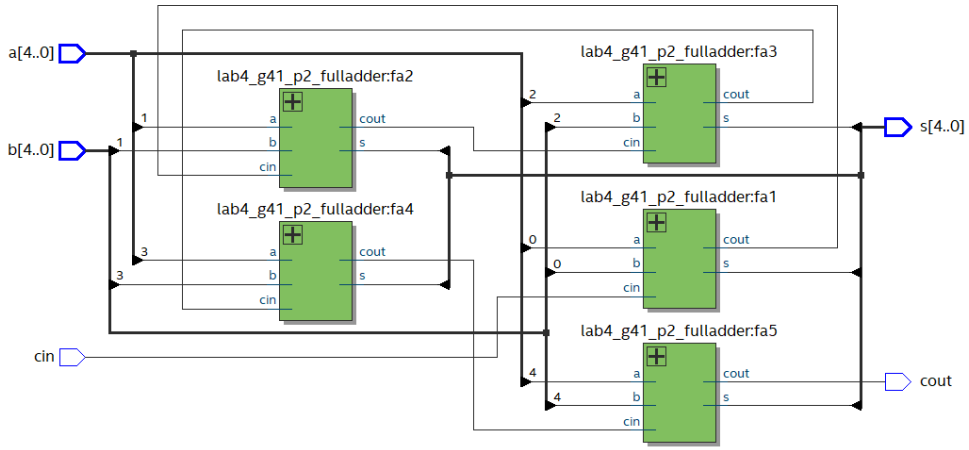
## D. Testbench

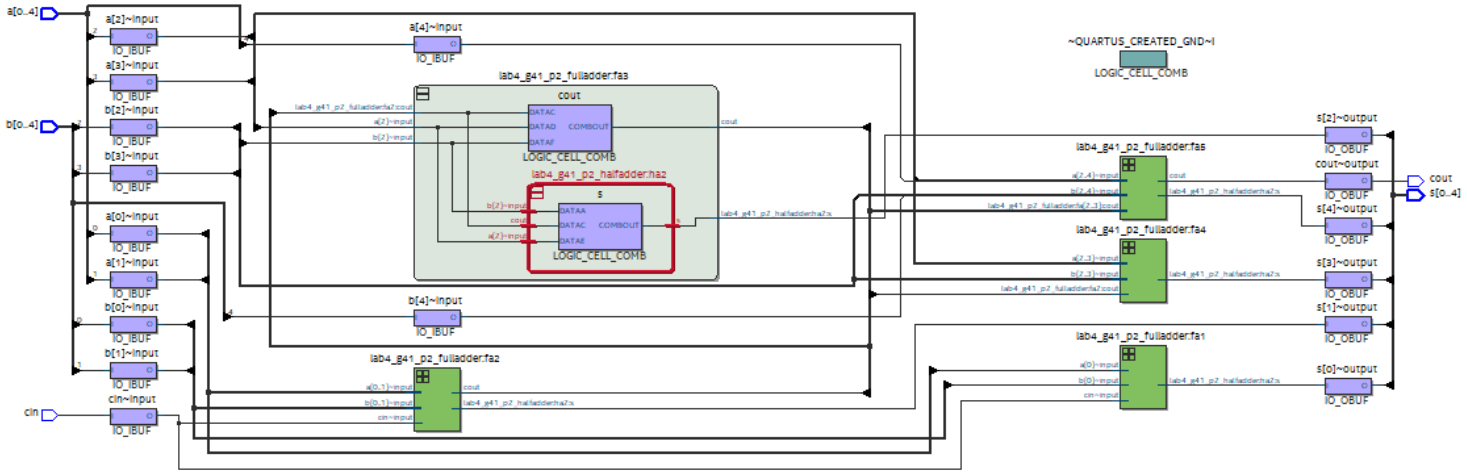



```
        /* tb_lab_4_g41_p2_rippleadder5.sv
         * Hazırlayanlar: Berat Kızılarmut
         * Notlar: Ripple Carry Adder testbenchi */

        module tb_lab4_g41_p2_rippleadder5();
            logic [4:0] a, b, s;
            logic cin, cout;
            lab4_g41_p2_rippleadder5 adder5(.a(a), .b(b), .s(s), .cin(cin), .cout(cout));
            initial begin
            cin = 0; #10;
                a = 5'b00001; b = 5'b00001; #10;
                a = 5'b00010; b = 5'b00001; #10;
                a = 5'b00100; b = 5'b00001; #10;
                a = 5'b00100; b = 5'b00011; #10;
                a = 5'b00101; b = 5'b00011; #10;
                a = 5'b00101; b = 5'b00111; #10;
                a = 5'b00111; b = 5'b00111; #10;
                a = 5'b01101; b = 5'b00011; #10;
                a = 5'b01101; b = 5'b01011; #10;
                a = 5'b00101; b = 5'b00011; #10;
                a = 5'b01101; b = 5'b00001; #10;
                a = 5'b00111; b = 5'b11011; #10;
                a = 5'b00001; b = 5'b01011; #10;
                a = 5'b00001; b = 5'b00011; #10;
                a = 5'b10100; b = 5'b10011; #10;
                a = 5'b10101; b = 5'b10011; #10;    $stop;
            end
endmodule
```

## E. Şemalar ve Yorumlar




| | Resource | Usage |
|---|---|---|
| 1 | Estimate of Logic utilization (ALMs needed) | 5 |
| 2 | | |
| 3 | Combinational ALUT usage for logic | 9 |
| 1 | -- 7 input functions | 0 |
| 2 | -- 6 input functions | 1 |
| 3 | -- 5 input functions | 4 |
| 4 | -- 4 input functions | 0 |
| 5 | -- <=3 input functions | 4 |
| 4 | | |
| 5 | Dedicated logic registers | 0 |
| 6 | | |
| 7 | I/O pins | 17 |
| 8 | | |
| 9 | Total DSP Blocks | 0 |
| 10 | | |
| 11 | Maximum fan-out node | lab4...cout |
| 12 | Maximum fan-out | 4 |
| 13 | Total fan-out | 60 |
| 14 | Average fan-out | 1.40 |

Devre toplama işlemini sıra ile least significant bit'den most significant bit'e doğru yapmaktadır.

Devre beklenildiği gibi çalışmaktadır. Şemalarda herhangi bir anormallik bulunmamakta. Post-Fitting şemasında gösterildiği gibi Full-adder'ın içinde half adderlar bulunmaktadır.

Devre toplam 9 ALUT paketi kullanmıştır. Bunların biri 6 girişli, dörder adeti 5 ve 3 girişlidir. Toplam 17 giriş ve çıkış pini kullanılmıştır.

## 2.3 Problem III – ALU Tasarımı

### A. 32-Bit NZVC ALU

Deney föyü değişikliği öncesi yazılmış olan ALU kodu aşağıda ektedir. Deney talimatları değiştiğinde yapılan değişiklikler ise C şıkkında gösterilecektir.

```
       /* lab_4_g41_p3_alu.sv
       * Hazırlayanlar: Berat Kızılarmut
       * Notlar: Arithmetic Logic Unit tasarımı */
module lab4_g41_p3_alu(
input logic [31:0] a,b,
input logic [3:0] op,
output logic  [31:0] s,
output logic n,z,v,c,hata);
always @(op or a or b)
begin
hata = 1'b0;
if (op==4'b0000) //addition
begin
s = a + b;
c = (a & b == 32'b0) ? 1'b0 : 1'b1;
v = ((a[31] ~^ b[31])) ? ( (s[31] ^ a[31]) ? 1'b1 : 1'b0) : 1'b0;
end
else if (op==4'b1000) //subtraction
begin
s = a - b;
c = (a & b == 32'b0) ? 1'b0 : 1'b1;
v = ((a[31] ~^ b[31])) ? ( (s[31] ^ a[31]) ? 1'b1 : 1'b0) : 1'b0;
end
else if (op==4'b0001) //shiftleftlogic
s = a << b;
else if (op==4'b0010) //signedcomp
s = $signed(a) > $signed(b) ? 32'd1:32'd0;
else if (op==4'b0011) //unsignedcomp
s = a > b ? 32'd1:32'd0;
else if (op==4'b0100) //xor
s = a ^ b;
else if (op==4'b0101) //shiftrightlogic
s = a >> b;
else if (op==4'b1101) //shiftrightarithmetic
s = a >>> b;
else if (op==4'b0110) //or
s = a | b;
else if (op==4'b0111) //and
s = a & b;
else
begin
hata = 1'b1;
s = 32'bz;
end
if(hata)
begin
n = 1'b0;
z = 1'b0;
end
else
begin
n = s[31];
z = ~|s;
end
end
endmodule
```

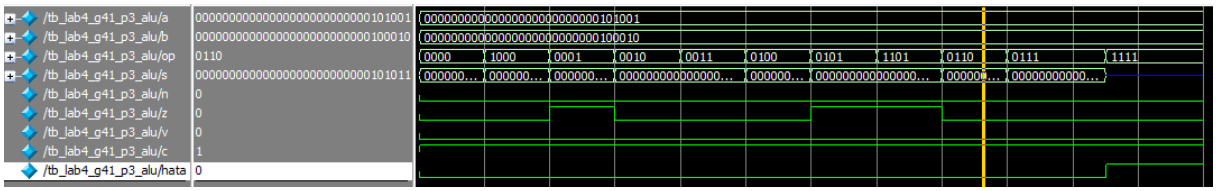
## B-C. Deney Föyü Güncellemeleri

Deneyin bu aşamaları geçildikten sonra yeni talimatlar geldiği için bu iki kısım ayrıldı. Yeni talimatlara göre eski devredeki hatalara düzeltmeler yapıldı. Carry işlemindeki hatalar düzeltildi. Shift işlemlerine carry eklendi. Always @(op or a or b) formatında, sensitivity listli block'tan, always_comb'a geçiş yapıldı.

Fakat biz deneyin bu kısımlarını geçip, değişiklik öncesi Problem 3-Yeni Testbench kısımlarına geldiğimiz için A kısmında verilen devreye değişiklikler zaten yapılmıştı. Tekrar geri dönülüp yeni yönergelere göre değişiklikler de yapıldı. Yeni devre F kısmında bütün bu değişiklikler ile birlikte verilecek.

## D. Waveshape

Eski Devre (föy güncellenmesi öncesi) Quartus Prime'da sentezlendi ver ModelSim'de sınandı. Rastgele iki değer verilip bütün fonksiyon işlemleri gerçekleştirildi. Devre bu testbenche göre doğru davranış gösterdi. Fakat sınanma yeterli olmadı. E Şıkkında yeni hatalar keşfedildi.




## E. Yeni Testbench

Yeni testbench ile devre sınandığında, öncelikli olarak V ve C fonksiyonlarının düzgün çalışmadığı gözlenildi. Bu fonksiyonlarda değişiklikler yapıldı. Shift right arithmetic operatöründe, a değişkeni unsigned olduğunda problemler yaşandığı için signed olarak değiştirildi. Always_comb'un içine n, z, v, c, ve hata değişkenlerini sıfırlayacak bir kural eklendi. Shift işlemlerine carry eklendi.

Kanıtlanmış ve doğru bir testbench kullandığımızda, doğru çalıştığını düşündüğümüz devredeki hata noktalarını keşfetmiş olduk ve hatalarımızı düzelttik. Kendi yazdığımız testbench devreyi sınamada yeterli olmamıştır. Öngöremediğimiz hataları, doğru çalışan bir devrenin sonuçları ile karşılaştırdığımızda kendi hatalarımızı görmemiz mümkün oldu.

## F. Revize Edilmiş Devre Testler

Bütün düzeltmelerden sonra A kısmındaki devre aşağıdaki hali aldı;

```
        /* lab_4_g41_p3_alu.sv
        * Hazırlayanlar: Berat Kızılarmut
        * Notlar: Revize edilmiş ALU */
module lab4_g41_p3_alu(
        input logic [31:0] a,b,
        input logic [3:0] op,
        output logic  [31:0] s,
        output logic n,z,v,c,hata);
        logic [31:0] twoscompb;
        logic [32:0] carrys;
        always_comb
        begin
                hata = 1'b0;n = 1'b0;z = 1'b0;v = 1'b0;c = 1'b0;twoscompb = 32'b0;carrys = 33'b0;
                if (op==4'b0000) //addition
                        begin
                        carrys = a + b;
                        s=carrys[31:0];
                                c = (carrys[32]) ? 1'b1 : 1'b0;
                                v = ((a[31] ~^ b[31])) ? ( (s[31] ^ a[31]) ? 1'b1 : 1'b0) : 1'b0;
                        end

                else if (op==4'b1000) //subtraction
                        begin
                        carrys = a - b;
                                s = carrys[31:0];
                                c = (carrys[32]) ? 1'b0 : 1'b1;
                                twoscompb = (~b) + 32'd1;
                                v = ((a[31] ~^ twoscompb[31])) ? ( (s[31] ^ a[31]) ? 1'b1 : 1'b0) : 1'b0;
                        end

                else if (op==4'b0001) //shiftleftlogic
                        begin
                        carrys = a << b[4:0];
                        c = carrys[32];
                        s = carrys[31:0];
                        end

                else if (op==4'b0010) //signedcomp
                        s = ($signed(a) > $signed(b)) ? 32'd1:32'd0;

                else if (op==4'b0011) //unsignedcomp
                        s = (a > b) ? 32'd1:32'd0;

                else if (op==4'b0100) //xorZ
                        s = a ^ b;

                else if (op==4'b0101) //shiftrightlogic
                        begin
                        if(|b[4:0])
                                begin
                        s = a >>> (b[4:0]-5'd1);
                        c = s[0];
                        s = s >>> (5'd1);
                                end
                        else
                                begin
                                s=a;
                                c=1'd0;
                                end
                        end

                else if (op==4'b1101) //shiftrightarithmetic
                        begin
                        if(|b[4:0])
                        begin
                        s = $signed(a) >>> (b[4:0]-5'd1);
                        c = s[0];
                        s = $signed(s) >>> (5'd1);
                        end
```

```
                        else
                                begin
                                s=a;
                                c=1'd0;
                                end
                        end

                else if (op==4'b0110) //or
                        s = a | b;

                else if (op==4'b0111) //and
                        s = a & b;

                else
                        begin
                        hata = 1'b1;
                        s = 32'bz;
                        end
                if(hata)
                        begin
                        n = 1'b0;
                        z = 1'b0;
                        end
                else
                        begin
                        n = s[31];
                        z = ~|s;
                        end
        end
endmodule
```

| | Resource | Usage |
|---|---|---|
| 1 | Estimate of Logic utilization (ALMs needed) | 355 |
| 2 | | |
| 3 | Combinational ALUT usage for logic | 502 |
| 1 | -- 7 input functions | 7 |
| 2 | -- 6 input functions | 201 |
| 3 | -- 5 input functions | 82 |
| 4 | -- 4 input functions | 61 |
| 5 | -- <=3 input functions | 151 |
| 4 | | |
| 5 | Dedicated logic registers | 0 |
| 6 | | |
| 7 | I/O pins | 105 |
| 8 | | |
| 9 | Total DSP Blocks | 0 |
| 10 | | |
| 11 | Maximum fan-out node | b[0]~input |
| 12 | Maximum fan-out | 86 |
| 13 | Total fan-out | 2480 |
| 14 | Average fan-out | 3.48 |

Devre toplam 355 ALM kullanmıştır. Toplam kullandığı 502 ALUT unit'den, 7'si 7 input, 201'i 6 input, 82'si 5 input, 61'i 4 input ve 151'i 3 veya daha az inputa sahiptir.

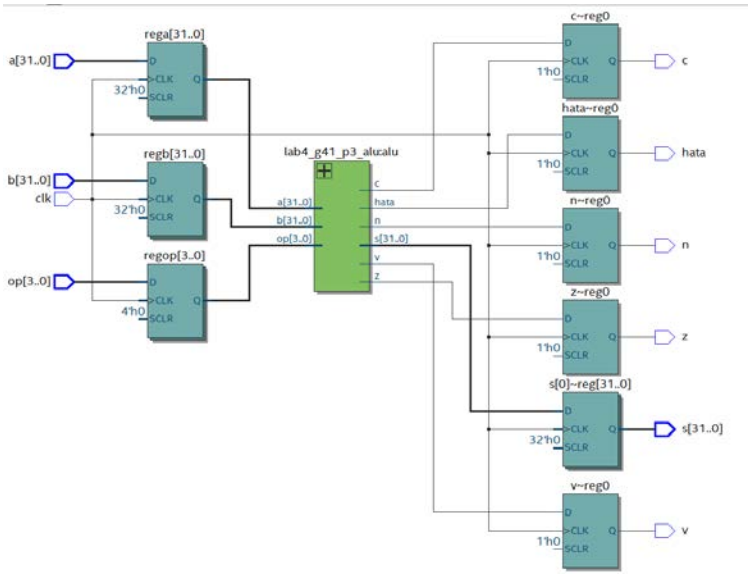
Şu ana kadar tasarladığımız en büyük mantıksal paket olmuştur.

RTL Şeması 2 sayfa, Post-Fitting Şeması 5 sayfa yer kaplamaktadır. Okunamayacağı derecede büyük oldukları için rapora eklenmemiştir.

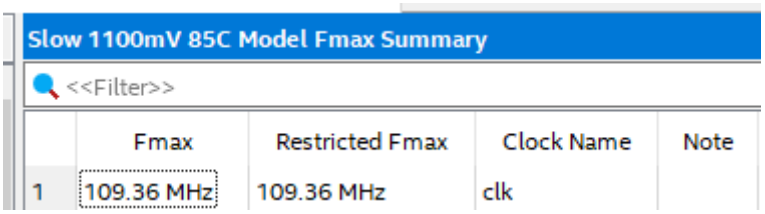
## 2.4 Problem IV – Senkron Tasarım ve Zamanlama

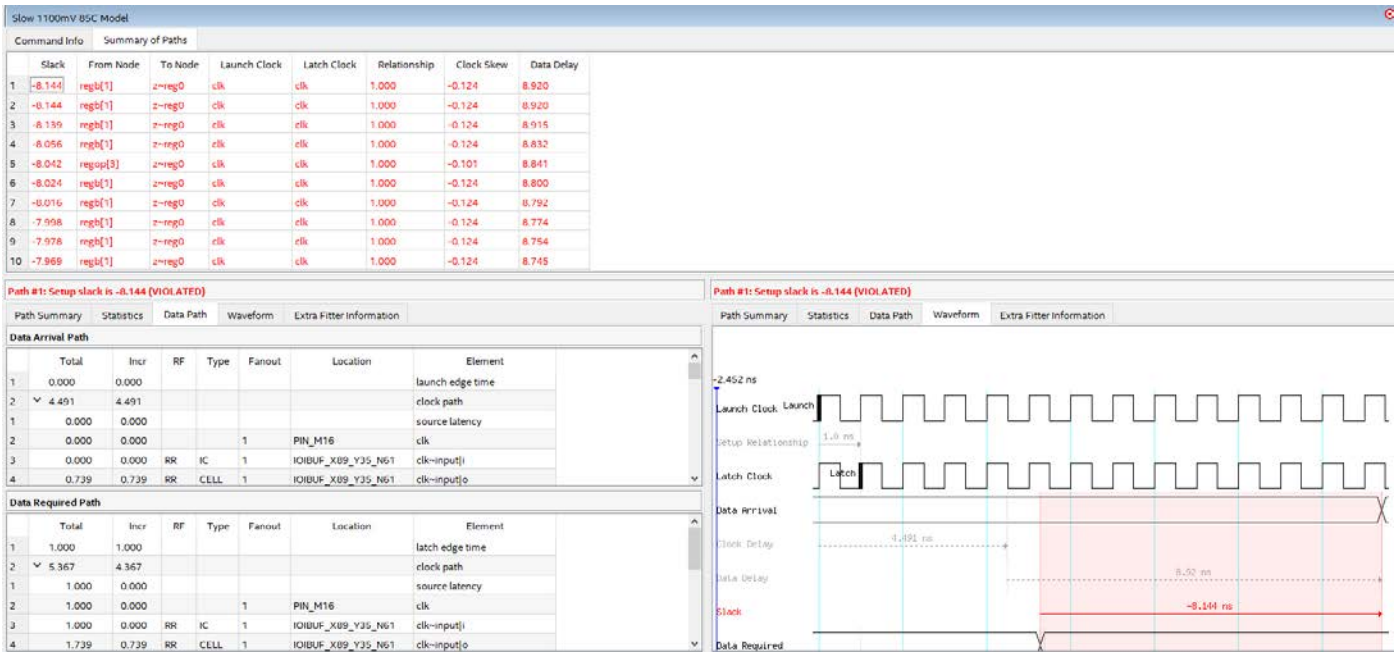
Problem 3'te tasarlanan ALU'ya register girişleri bağlandı.

```
        /* lab_4_g41_p4_alu.sv
        * Hazırlayanlar: Berat Kızılarmut
        * Notlar: Registry bağlantıları yapılmış ALU */
module lab4_g41_p4_alu(
        input logic [31:0] a,b,
        input logic [3:0] op,
        output logic  [31:0] s,
        output logic n,z,v,c,hata,
    input logic clk
);
logic [31:0] rega, regb, regs;
logic [3:0] regop;
logic regn, regz, regv, regc, reghata;
lab4_g41_p3_alu
alu(.a(rega),.b(regb),.op(regop),.s(regs),.n(regn),.z(regz),.v(regv),.c(regc),.hat
a(reghata));
always_ff @(posedge clk)
begin
    rega <= a;
    regb <= b;
    regop <= op;
    s <= regs;
    n <= regn;
    z <= regz;
    v <= regv;
    c <= regc;
    hata <= reghata;
end
endmodule
```



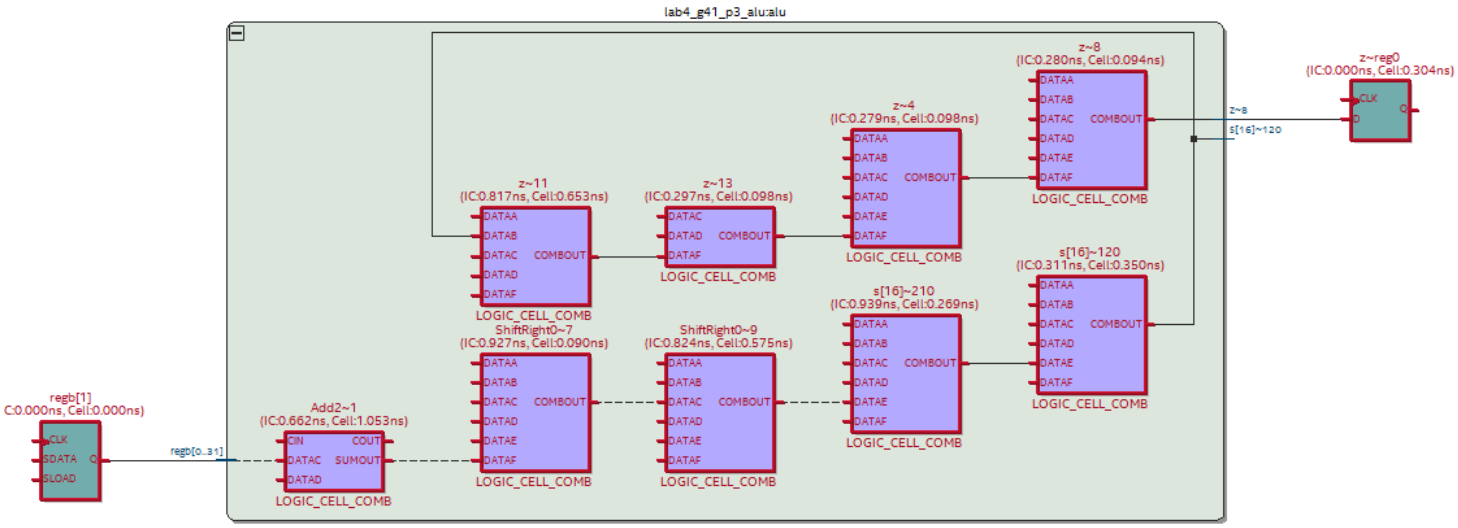

Ardından yukarıdaki devre Quartus'ta sentezlendi. Zamanlama analizi sonucu devrenin maksimum frekansının 109.36 MHz olduğu tespit edildi. Devremiz çok kalabalık bir devre olduğu için bu çalışma frekansının iyi olduğunu düşünmekteyiz.






Zamanlama analizini yaptığımızda ise devremizde toplam 17,739 setup path bulmuştur. Bunların hepsi kırmızı olarak gözükmüştür. Bunun sebebi ise timing analizinde kullandığımız testbenchin frekansının 1 GHz olmasından kaynaklıdır. Test bench'in clock hızını daha makul bir hıza çektiğimizde ise bu kırmızı sayı miktarı dolayısıyla değişmektedir.

Devremizin kritik hattı regb[1] registrysinin izlediği bir hattır.
Bu hat 0.856 Slack, -0.124 Clock Skew ve 8.920 ns Data delay'e sahiptir.
Bahsi geçen bu kritik hat aşağıda gösterilmiştir.



## 3. Genel Yorumlar ve Kazanımlar

Bu deneyde combinational circuitlere bir giriş yapmış bulunmaktayız. Deney boyunca always block'unu kullanarak combinational circuitler oluşturduk. Daha önceki deneylerde de temellerini attığımız structural design'dan da faydalanarak, lojik elementlerini birer yapıp bloğu olarak kullanıp, üstlerine eklemeler yaparak fonksiyonlarını geliştirdik.

Deneyde kazandığımız önemli bilgilerden biri ise bir devreyi tasarlarken kullandığımız test benchin önemi oldu. Gerekli sınamaları yapmayan bir test benchte, hatalı bir devreyi doğru sanmamız mümkün olmakta.

## 4. Kaynaklar


1. Harris and D. Harris, Digital Design and Computer Architecture: ARM Edition, 1st edition. Morgan Kaufmann, 2015
2. Half Adder and Full Adder Circuit with Truth Tables. Retrieved from https://www.elprocus.com/half-adder-and-full-adder/
3. Shift Operator – Verilog Example. Retrieved from https://www.nandland.com/verilog/examples/example-shift-operator-verilog.html