GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x4 Deney Raporu

Senkron Tasarım ve Aritmetik Mantık Devreleri

Hazırlayanlar
1) 1801022091 - Ogün Uygar YILDIRIM
2) 1801022022 - Alperen KARATAŞ

1. Giriş

Bu deney kapsamında ; senkron tasarım ve asimetrik mantık devreleri hakkında gerekli araştırmalar yapılmıştır. Belirli araştırmalar sonrasında deney için gerekli olan bilgiler sağlanılmıştır.

2. Problemler

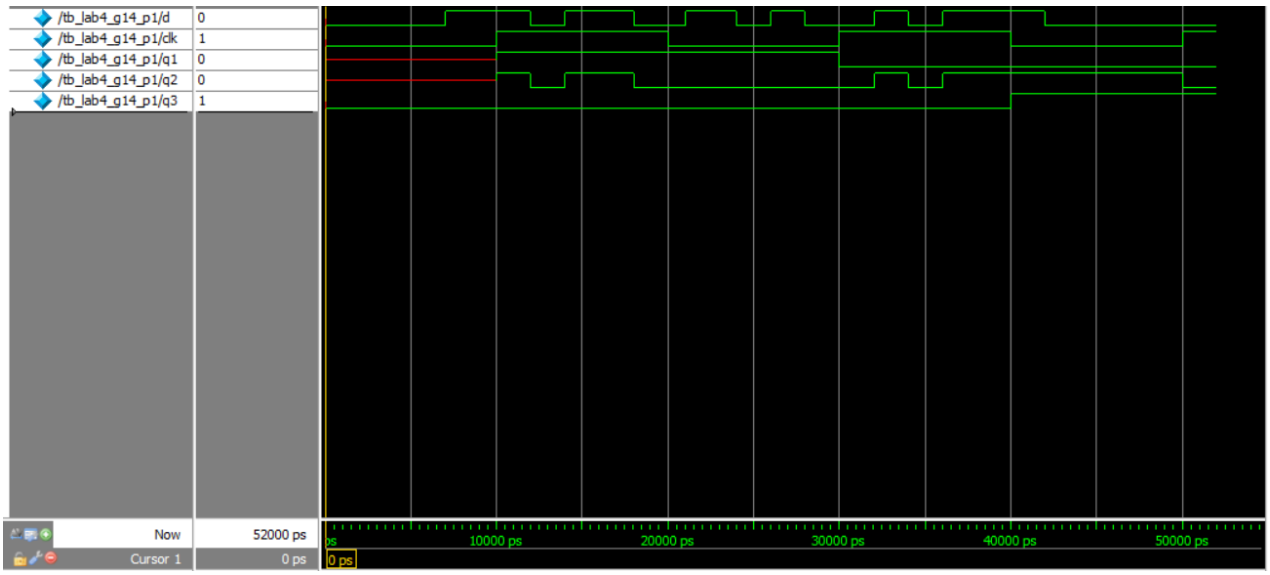
2.1. Problem I - - Hafıza elemanları karşılaştırması

2.1.1. Teorik Araştırma

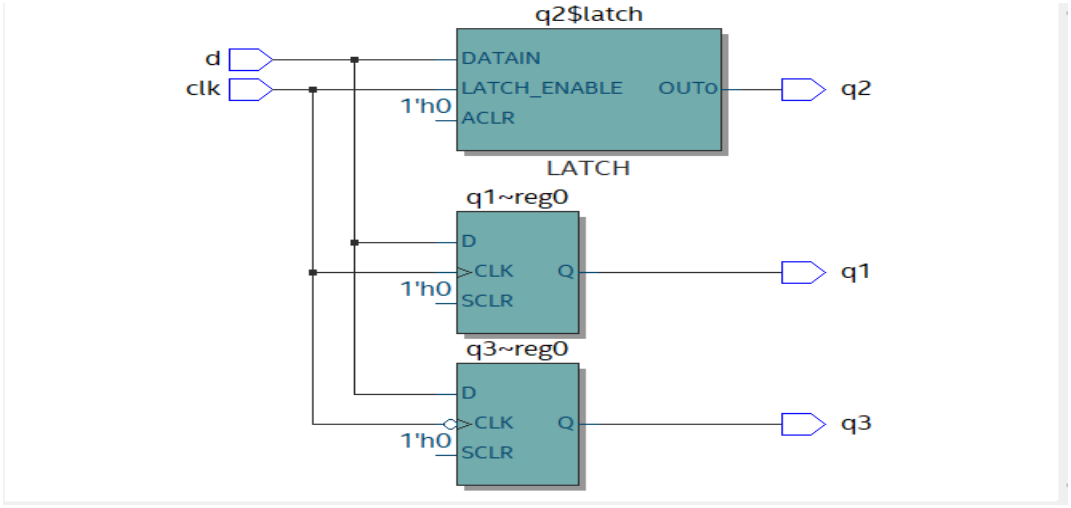
Bu problemin teorik araştırmasında lojik hafıza elemanları olan Latch, Rising-edge triggered Flip-Flop ve Falling-edge triggered Flip-Flop hakkında araştırmalar yapılmıştır ve bu elemanların systemverilog dili ile nasıl kullanılacağı öğrenilmiştir.

2.1.2. Deneyin Yapılışı

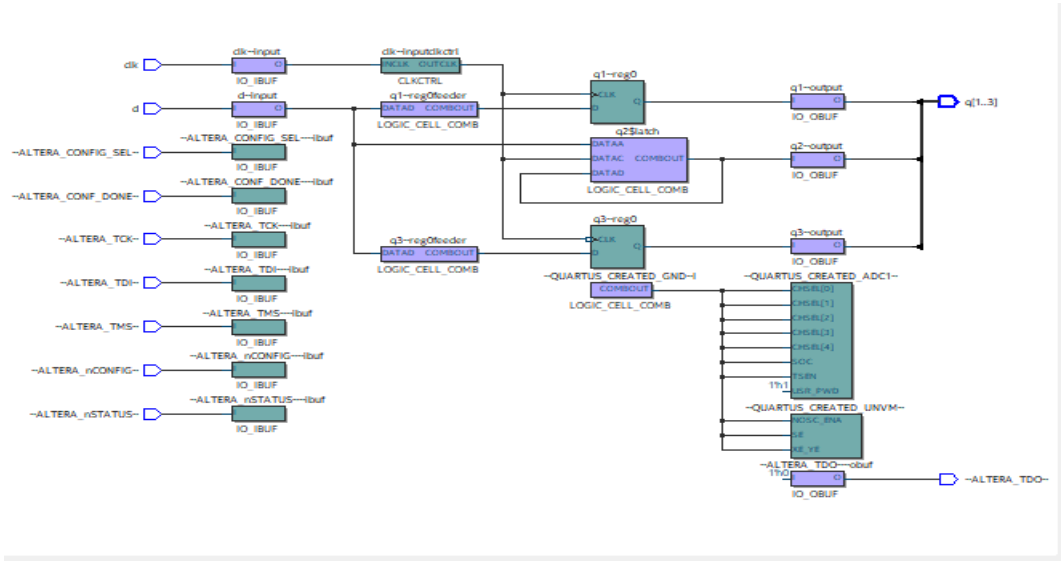
Deneye başlamadan önce gerekli SystemVerilog dosyaları açılmıştır. Problemde istenilen Latch , Rising-edge triggered Flip-Flop ve Falling-edge triggered Flip-Flop hafıza elemanları girişleri tek sinyale , çıkışları ise ayrı sinyallere bağlanmıştır.



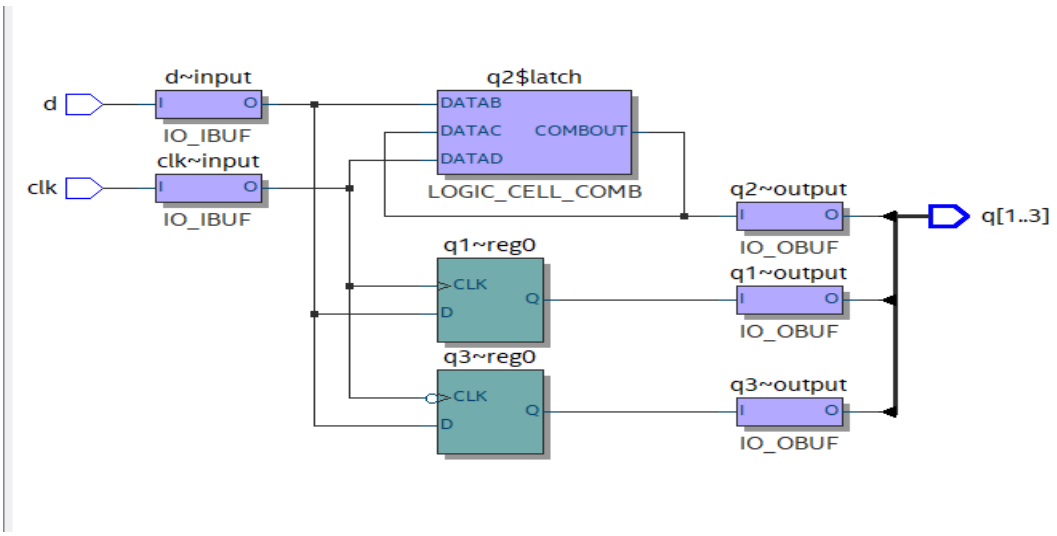
Şekil 1



Şekil 2



Şekil 3



Şekil 4

2.1.3. Sonuçların Yorumu

Şekil 2'de görüldüğü üzere devrede giriş aynı sinyale bağlanmıştır, çıkış ise ayrı ayrı sinyallere bağlanmıştır. Latch, Rising-edge triggered Flip-Flop ve Falling-edge triggered Flip-Flop arasındaki simülasyon sonuçları farkı şekil 1'de görülüyor.Bu simülasyon farkı hafıza devreleri arasındaki yapı farkları sonucunda oluşmuştur. Latchlarde d'nin q'ya bağlı değişiminin belli bir süreli sinyal çizgisinde gerçekleşirken, flip-flop'larda bu değişimin rising veya falling edge'lerde değiştiğini görüyoruz.

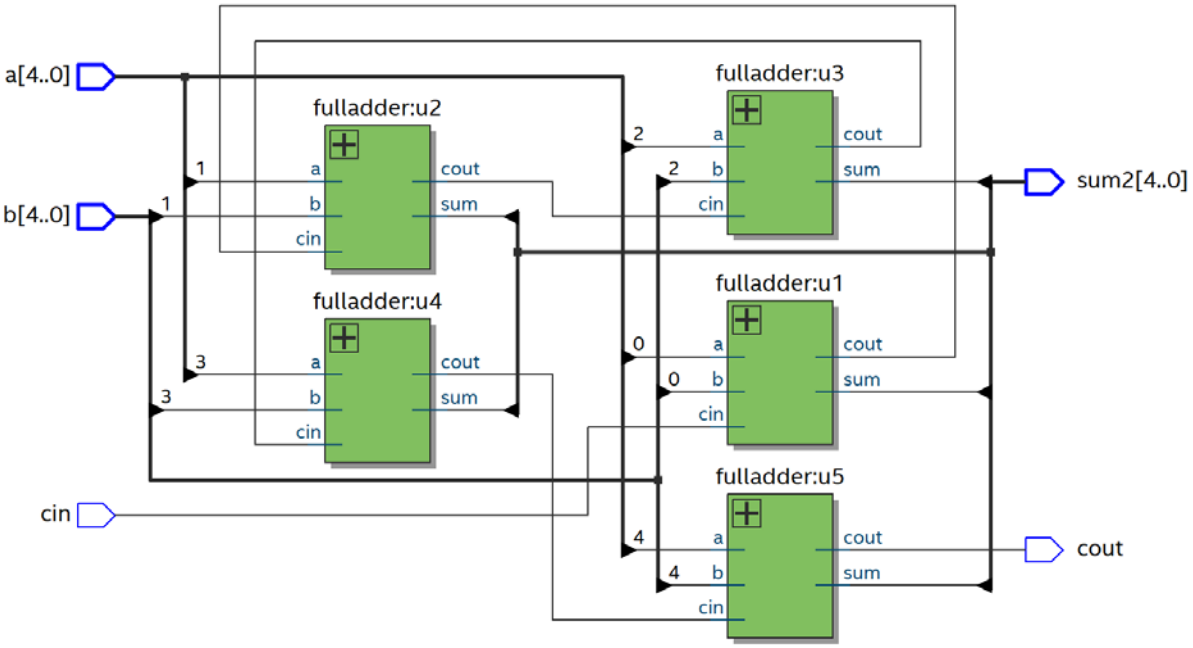
2.2. Problem II – Adder Tasarımı

2.2.1. Teorik Araştırma

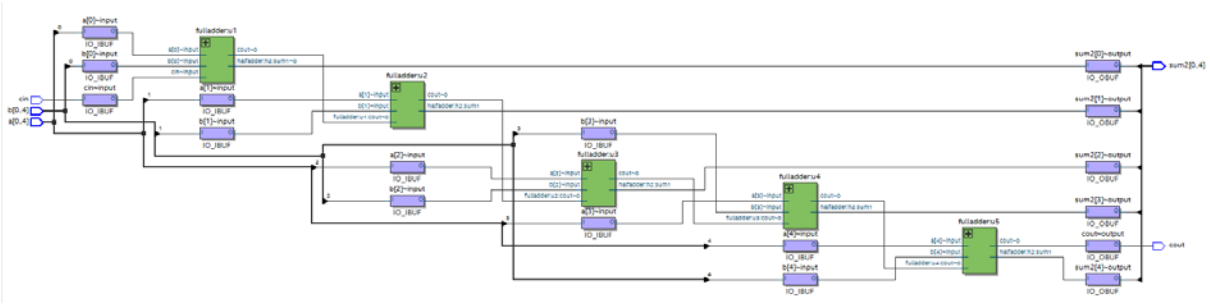
Deneyde kullanılacak olan adder araştırılıp hakkında bilgi sahibi olunmuştur. Half-adder kullanarak full-adder, full-adder kullanarak ripple carry adder tasarlanması araştırılmıştır.

2.2.2. Deneyin Yapılışı

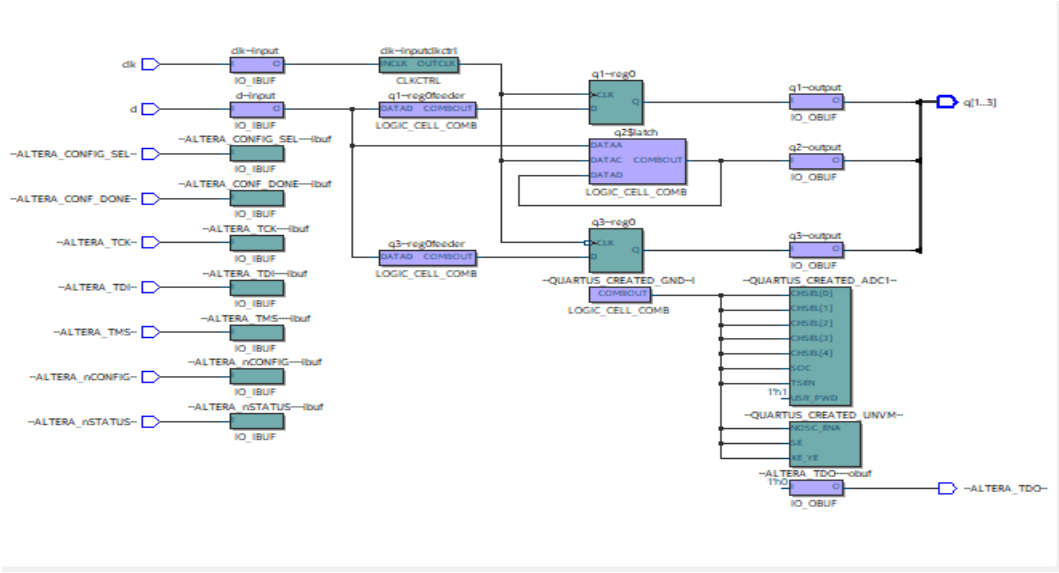
Deney başlarken öncelikle half-adder ve half-adder kullanılarak full-adder tasarlanmıştır. 5 bitlik ripple carry adder tasarlanması için sırayla 5 adet full-adder kullanılmıştır ve herhangi bir full-adder girişi bir sonraki full-adder çıkışına bağlanmıştır.



Şekil 5



Şekil 6



Şekil 7

Flow Status	Successful - Sun Apr 19 00:51:27 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	lab4_g14_p2c
Top-level Entity Name	lab4_g14_p2c
Family	MAX 10
Device	10M08DAF484C8G
Timing Models	Final
Total logic elements	11 / 8,064 (< 1 %)
Total registers	0
Total pins	17 / 250 (7 %)
Total virtual pins	0
Total memory bits	0 / 387,072 (0 %)
Embedded Multiplier 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 2 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 1 (0 %)

Şekil 8

2.2.3. Sonuçların Yorumu

Bu problem kodlarında 5 adet full-adder kullanılarak 5 bitlik ripple carry adder tasarlanmıştır.Bu durum şekil5'de de belirtilmiştir.Toplam 5 adet full-adder vardır ve her bir adder girişi bir sonraki adder çıkışına bağlanmıştır. Özellikle structural design yapısı etkin bir biçimde kullanıldı. Bu sayede daha senkron bir çalışma elde ettik.

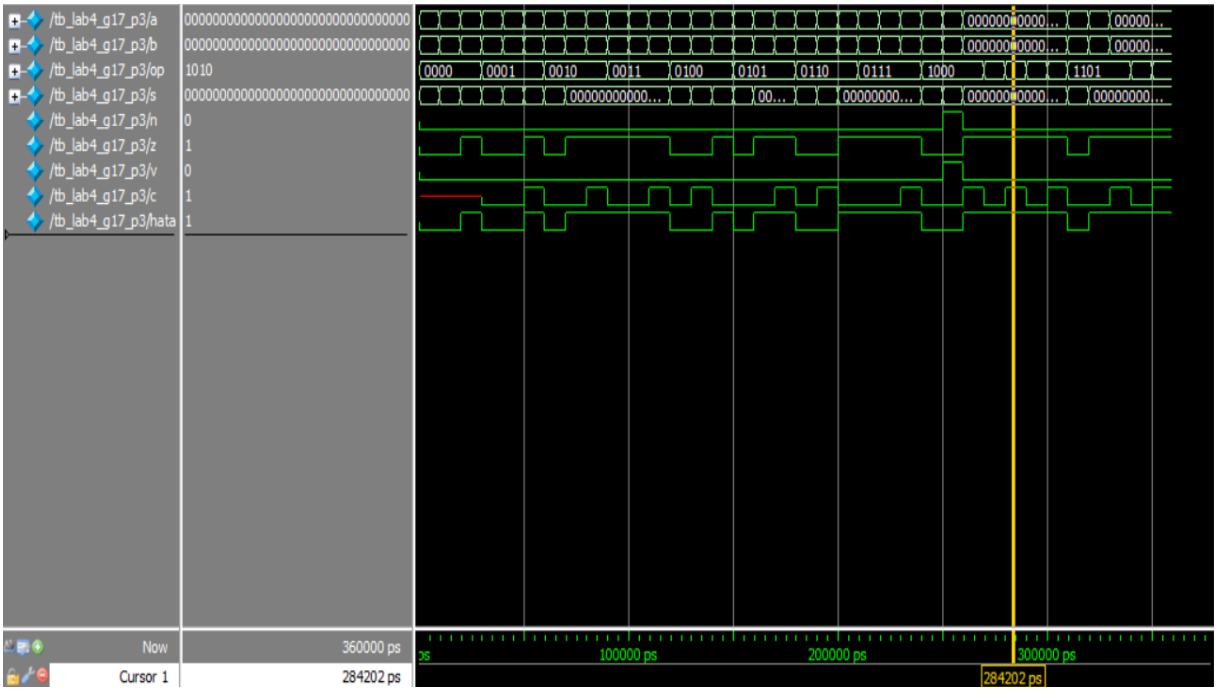
2.3. Problem III -- ALU Tasarımı

2.3.1. Teorik Araştırma

Deneye başlamadan önce aritmetik devre tasarımları hakkında araştırma yapılmıştır ve yeterli seviyede bilgi sahibi olunmuştur.

2.3.2. Deneyin Yapılışı

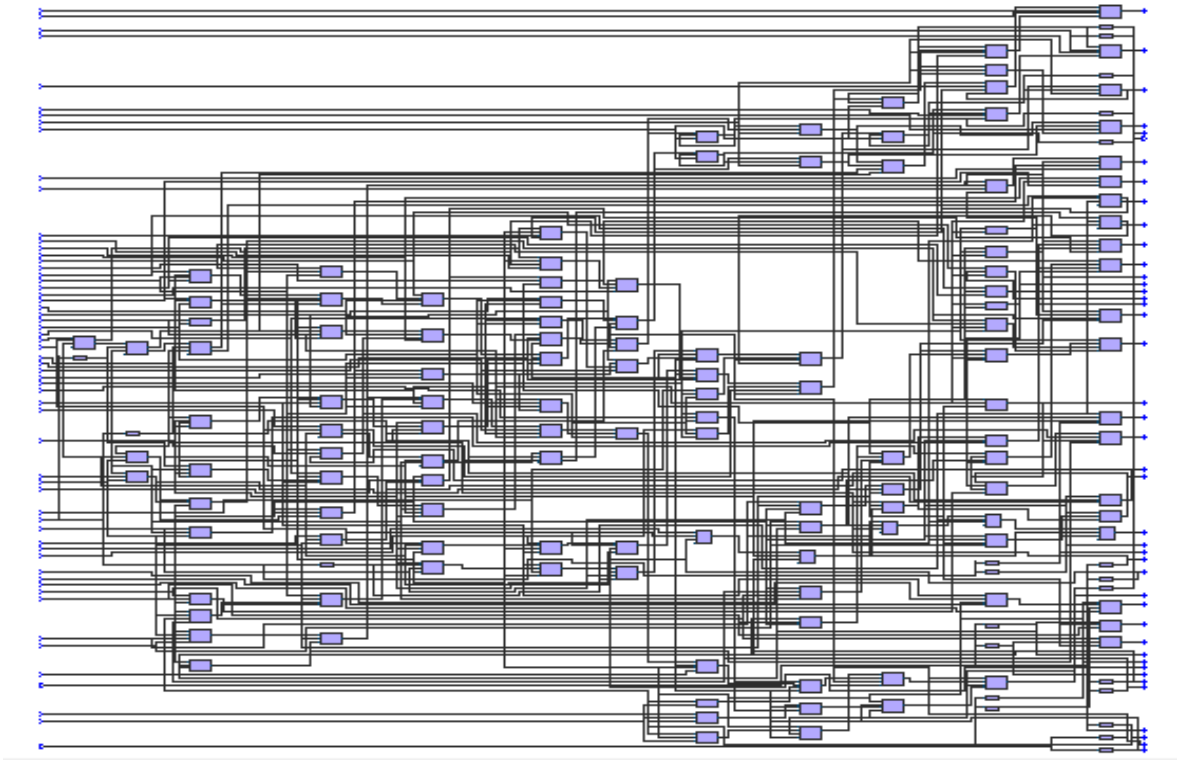
Deneye başlamadan önce tasarlanacak olan devre , şemalar halinde kağıda çizilmiştir deney yapımı şema şema ilerlenmiştir. 32-bitlik NZVC destekli ALU tasarlanmıştır. ALU nun opcodeları ve gerçeklemeniz gereken fonksiyonlar testbench dosyasında birkaç girişle kontrol edilmiştir.



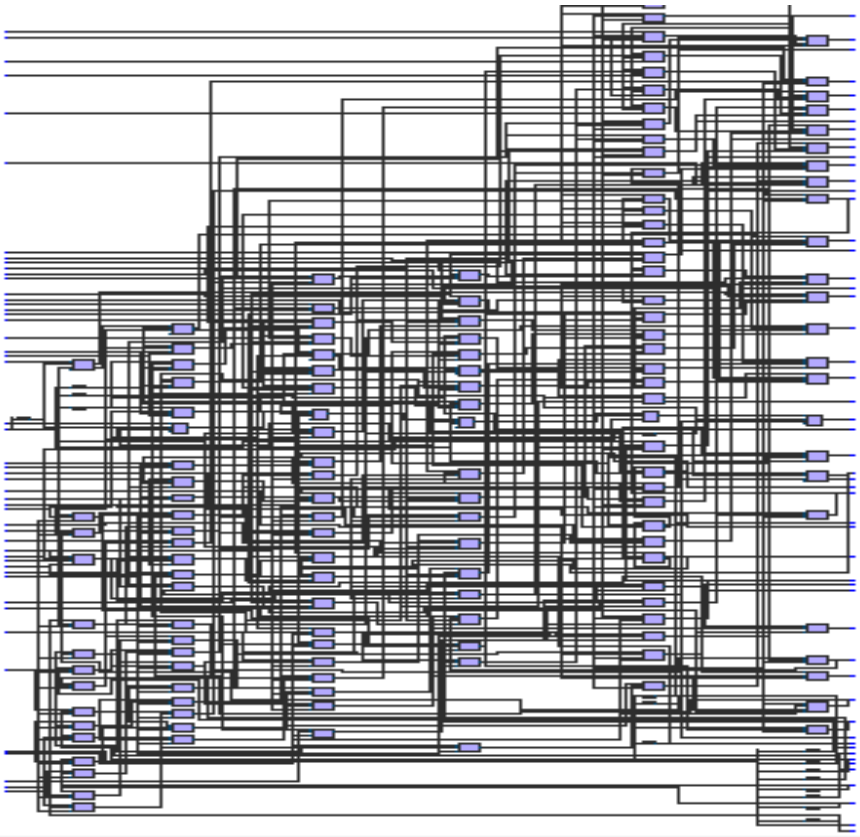
Şekil 9

Flow Status	Successful - Mon Apr 20 00:08:32 2020
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	lab04_g17_p3
Top-level Entity Name	lab04_g17_p3
Family	MAX 10
Device	10M08DAF484C8G
Timing Models	Final
Total logic elements	700 / 8,064 (9 %)
Total registers	0
Total pins	105 / 250 (42 %)
Total virtual pins	0
Total memory bits	0 / 387,072 (0 %)
Embedded Multiplier 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 2 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 1 (0 %)

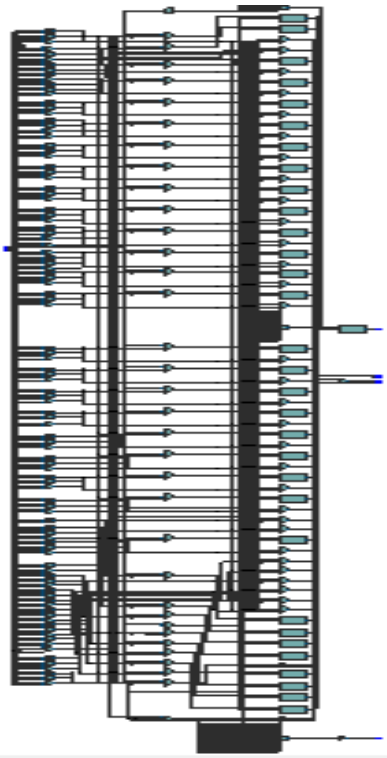
Şekil 10



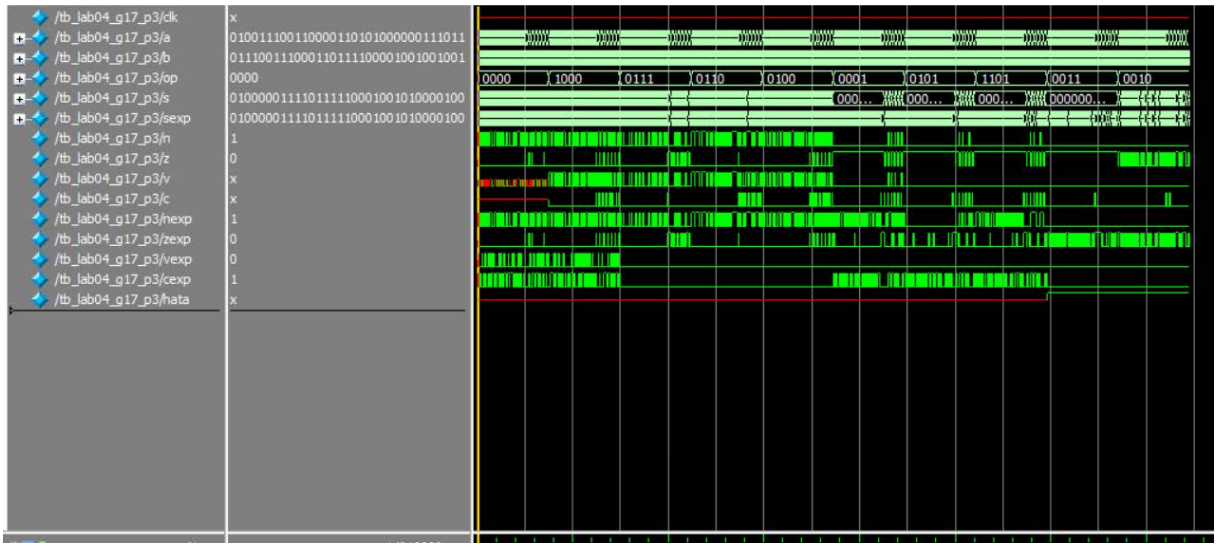
Şekil 11



Sekil 12



Sekil 13



Şeki1 14

2.3.3. Sonuçların Yorumu

Deneyde anlaşıldığı üzere her bir operatöre bir fonksiyon atanmıştır ve NVZC bayrakları kullanılmıştır. Devrede atanan operatörler haricinde herhangi bir operatör kulanılırsa hata biti devreye girmiştir. Şekil 9 ve Şekil 14' de belirtilen simülasyon sonuçları arasında belirli farklar gözlenmektedir. Bu farkların sebebi iki testbench arasındaki kod farkları ve kontrol mekanizması olmuştur. Bizden istenen 10 farklı kombinansyonda istenen sonuçlar elde edilirken, kalan 6 kombinasyonda simülasyon ilerleyişinin daha farklı sonuçlar verdiğini görebiliriz.

2.4.Problem IV-- Senkron Tasarım ve Zamanlama

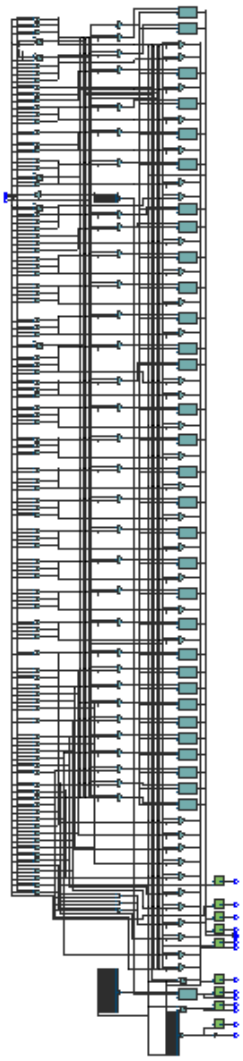
2.3.4. Teorik Araştırma

Bu deney öncesi registerın işlevi hatırlanıp ALU-register senkron çalışması üzerinde araştırmalar yapılmıştır.

2.3.5. Deneyin Yapılışı

Bu deneyde ilk aşama olarak, bir önceki deneyde tasarlanan ALU'nun tüm giriş ve çıkışlarına registerın bağlanması üzerine, nasıl uygun bir devre dizaynı oluşacağı yönünde düşünülmüş ve tasarlanan devre bir kağıda çizilerek giriş-çıkışlara bağlanan registerların yerleri öngörülmüştür.

İlk aşama tamamlandıktan sonra, structural design yapısı kullanılarak devrenin kodu yazılmıştır.



Şekil 15

Şekil 15'te bahsi geçen devrenin RTL şeması görülmektedir

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-0.225	op[0]	out[0]	op[0]	op[0]	0.000	5.047	4.822
2	0.102	op[0]	out[11]	op[0]	op[0]	0.000	5.034	5.136
3	0.182	op[0]	out[12]	op[0]	op[0]	0.000	5.044	5.226
4	0.220	op[0]	out[0]	op[0]	op[0]	-0.500	5.047	4.787
5	0.455	op[0]	out[19]	op[0]	op[0]	0.000	5.043	5.498
6	0.547	op[0]	out[11]	op[0]	op[0]	-0.500	5.034	5.101
7	0.749	op[0]	out[12]	op[0]	op[0]	-0.500	5.044	5.313

Şekil 16

Şekil 16 pathler hakkında bize bilgi vermektedir.

	Fmax	Restricted Fmax	Clock Name	Note
1	97.13 MHz	97.13 MHz	op[0]	

Şekil 17

Şekil 17 frekanslar hakkında bize bilgi vermektedir.

2.3.6. Sonuçların Yorumu

Şekil 16'daki 1.satır bize critical pathi verir. Sonuçların bize söylediği değere göre data delayi 4.822 ns'dir. 1.satırın critical path olduğunu göz önünde bulundurup diğer satırlarla karşılaştırdığımızda bunun bize doğru sonucunu verdiğinden emin değiliz.

Şekil 17 bize max frekansı söylemektedir. Görüldüğü gibi max frekans 97.13 MHz'dir. Bu bilgiye göre devrenin max 97.13 MHz'lik frekanslarla yeni datalar ürettiğini düşünüyoruz.

3. Sonuçlar ve Genel Yorumlar

Deneyler/problemler genel olarak zorlayıcıydı. Yoğun araştırma ve bilgi gerektiren problemlerle uğraştık. İstenmeyen latch oluşumları bizi oldukça yordu diyebiliriz. Kimi zaman bu sorunun üstesinden gelebilirken, kimi zaman da gelemedik. Pozitif yönlere baktığımızda; özellikle iki veya daha fazla kondisyonların birbirleriyle karşılaştırılması açısından, koşul komutlarının koddaki işleyişi açısından ve de structural design yapısının daha etkin bir şekilde kullanımı bakımından oldukça faydalı bir laborauvardı.

4. Referanslar

[1] Ders slaytları

[2]stackoverflow.com

[3] fpga4student.com

[4] www.youtube.com

/* Bir sonraki sayfalarda KODLAR bulunmaktadır*/

KODLAR

Problem 1-sv

```
/* lab4_g14_p1.sv
* Hazırlayanlar:
* Alperen Karataş - Ogün Uygar Yıldırım
* Notlar:
* ELM235 2020 Bahar Lab4 - Problem 1
* FLIP-FLOP ve LATCH Tasarlama
*/

module lab4_g14_p1(
input logic clk,
input logic d,
output logic q1,q2,q3
);

always_ff @(posedge clk)
      q1<=d;

always_latch
if(clk) q2<=d;

always_ff @(negedge clk)
      q3<=d;
endmodule
```

Problem 1-tb

```
/* tb_lab4_g14_p1.sv
* Hazırlayanlar:
* Alperen Karataş - Ogün Uygar Yıldırım
* Notlar:
* ELM235 2020 Bahar Lab4 - Problem 1
* FLIP-FLOP ve LATCH Tasarlama – Testbench
*/
`timescale 1ns/1ps
module tb_lab4_g14_p1 ();
logic d;
logic clk;
logic q1, q2, q3;
lab4_g14_p1
uut0(.q1(q1),.q2(q2),.q3(q3),.d(d),.clk(cl
k));
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
```

Problem 2.a-sv

```
/* lab4_g17_p2a.sv
* Hazırlayanlar:
* Alperen Karataş - Ogün Uygar Yıldırım
* Notlar:
* ELM235 2020 Bahar Lab4 - Problem 2.a
* Half-adder  tasarlama
*/

module lab4_g17_p2a (
  input logic  a    ,
  input logic  b    ,
  output logic sum,
  output logic carry
          );
assign sum=a^b;
assign carry=a&b;

endmodule
```

Problem 2.b-sv

```
/* lab4_g17_p2b.sv
* Hazırlayanlar:
* Alperen Karataş - Ogün Uygar Yıldırım
* Notlar:
* ELM235 2020 Bahar Lab4 - Problem 2.b
* Half-adder kullanarak full-adder tasarlama
*/
module halfadder(
  input logic  a    ,
  input logic  b    ,
  output logic sum,
  output logic carry
          );
assign sum=a^b;
assign carry=a&b;
endmodule
module lab4_g17_p2b(
 input logic a,
 input logic b,
 input logic cin,
 output logic sum,
 output logic cout
);
 logic s1,n1,n2,c1,c2;
halfadder h1 (.a(a), .b(b), .sum(n1),
.carry(c1));
halfadder h2(.a(n1), .b(cin), .sum(sum),
.carry(c2));
assign cout = c1 | c2;
endmodule
```

Problem 2.c-sv

```
*
* Hazırlayanlar:
* Alperen Karataş - Ogün Uygar Yıldırım
*
* Notlar:
* ELM235 2020 Bahar Lab4 - Problem 2.c
* Full-adder kullanarak ripple carry adder
tasarlama
*
*/

module halfadder(
  input logic  a ,
  input logic  b ,
  output logic sum1,
  output logic carry
          );
assign sum1=a^b;
assign carry=a&b;
endmodule

module fulladder(
 input logic a,
 input logic b,
 input logic cin,
 output logic sum,
 output logic cout
);
 logic s1,n1,n2,c1,c2;
halfadder h1 (.a(a), .b(b), .sum1(n1),
.carry(c1));
halfadder h2(.a(n1), .b(cin), .sum1(sum),
.carry(c2));
assign cout = c1 | c2;
endmodule


module lab4_g17_p2c(
input logic [4:0] a,b,
input logic cin,
output logic cout,
output logic [4:0]sum2
);
```

```
logic w1,w2,w3,w4,w5;
fulladder
u1(.a(a[0]),.b(b[0]),.cin(cin),.sum(sum2[0
]),.cout(w1));
fulladder
u2(.a(a[1]),.b(b[1]),.cin(w1),.sum(sum2[1]
),.cout(w2));
fulladder
u3(.a(a[2]),.b(b[2]),.cin(w2),.sum(sum2[2]
),.cout(w3));
fulladder
u4(.a(a[3]),.b(b[3]),.cin(w3),.sum(sum2[3]
),.cout(w4));
fulladder
u5(.a(a[4]),.b(b[4]),.cin(w4),.sum(sum2[4]
),.cout(cout));
endmodule
```

Problem 3-sv

```
/* lab4_g17_p3.sv
* Hazırlayanlar:
* Alperen Karataş - Ogün Uygar Yıldırım
* Notlar:
* ELM235 2020 Bahar Lab4 - Problem 3
* ALU Tasarlama
*/
module lab04_g17_p3 (
input logic [31:0] a, b,
input logic [ 3:0] op,
output logic [31:0] s,
output logic n, z, v, c,
output logic hata
);
reg [31:0] out;
assign s = out;
always @(*)
begin
    case(op)
    4'b0000:
    out = a+b;
    4'b1000:
  out = a-b;
    4'b0001:
    out = a << b;
    4'b0010:
    out = ($signed(a)>$signed(b)) ? 32'd1
: 32'd0 ;
    4'b0001:
    out = (a>b) ? 32'd1 : 32'd0 ;
    4'b0100:
    out= a ^ b;
    4'b0101:
    out = a >> b;
    4'b0110:
    out = a | b;
    4'b0111:
    out =  a & b;
     default:
     hata=out?0:1;
    endcase

 n = out[31];
 z = ~(|out);
 c = {c,out} == a+b;
 v = ({c,out[31]} == 2'b01);
end
endmodule
```

Problem 3-tb

```
/* tb_lab4_g17_p3.sv
* Hazırlayanlar:
* Alperen Karataş - Ogün Uygar Yıldırım
* Notlar:
* ELM235 2020 Bahar Lab4 - Problem 3
* ALU Tasarlama - Testbench
*/

`timescale 1ns / 1ps
module tb_lab4_g17_p3 ();
logic [31:0] a, b;
logic [3:0] op;
logic [31:0] s;
logic n, z, v, c;
logic hata;
lab4_g17_p3 dut0(.a(a), .b(b), .op(op),
.s(s), .n(n), .z(z), .v(v), .c(c),
.hata(hata));
```

```
initial begin
    op = 4'b0000;  //0
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b0001;  //1
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b0010;  //2
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b0011;  //3
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b0100;  //4
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
```

```
    op = 4'b0101;  //5
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b0110;  //6
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b0111;  //7
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b1000;  //8
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b1001; #10  //9
    op = 4'b1010; #10  //10
    op = 4'b1011; #10  //11
    op = 4'b1100; #10  //12
    op = 4'b1101;  //13
    a = 8'd5; b=8'd2; #10
    a = 8'd2; b=8'd4; #10
    a = 8'd0; b=8'd0; #10
    op = 4'b1110; #10  //14
    op = 4'b1111; #10  //15
    $stop;
end
endmodule
```

Problem 4-sv

```
 /* lab4_g14_p4.sv
*
* Hazırlayanlar:
* Alperen Karataş - OgÃ¼n Uygar Yıldırım
*
* Notlar:
* ELM235 2020 Bahar Lab4 - Problem 4
* Senkron Tasarım ve Zamanlama
*/


module reg32 (
    input logic clk,
    input logic [31:0] d,
    output logic [31:0] q
    );

    always_ff @(posedge clk)
    q <= d;

    endmodule

module reg4 (
    input logic clk,
    input logic [31:0] d,
    output logic [31:0] q
    );

    always_ff @(posedge clk)
    q <= d;

    endmodule

module reg0 (
    input logic clk,
    input logic d,
    output logic q
    );

    always_ff @(posedge clk)
    q <= d;

    endmodule


module lab4_g14_p4 (
input logic [31:0] a, b,
input logic [ 3:0] op,
output logic [ 3:0] q1,
output logic [31:0] s,q3,q4,q5,
output logic n, z, v, c,q2,q6,q7,q8,q9,
output logic hata
);


reg [31:0] out;
assign s = out;
```

```
always @(*)
begin
    case(op)
    4'b0000:
    out = a+b;
    4'b1000:
    out = a-b;
    4'b0001:
    out = a << b;
    4'b0010:
    out = ($signed(a)>$signed(b)) ? 32'd1 :
32'd0 ;
    4'b0001:
    out = (a>b) ? 32'd1 : 32'd0 ;
    4'b0100:
    out= a ^ b;
    4'b0101:
    out = a >> b;
    4'b1101:
    out = a >>> b;
    4'b0110:
    out = a | b;
    4'b0111:
    out =  a & b;
     default:
     hata=out?0:1;
    endcase

 n = out[31];
 z = ~(|out);
 c = {c,out} == a+b;
 v = ({c,out[31]} == 2'b01);
end

reg32 r32_1 (.d(a), .clk(clk), .q(q3));
reg32 r32_2 (.d(b), .clk(clk), .q(q4));
reg32 r32_3 (.d(s), .clk(clk), .q(q5));
reg4 r4 (.d(op), .clk(clk), .q(q1));
reg0 r_1 (.d(n), .clk(clk), .q(q2));
reg0 r_2 (.d(z), .clk(clk), .q(q6));
reg0 r_3 (.d(v), .clk(clk), .q(q7));
reg0 r_4 (.d(c), .clk(clk), .q(q8));
reg0 r_5 (.d(hata), .clk(clk), .q(q9));


endmodule
```