GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 4 Deney Raporu

Senkron Lojik Tasarım ve Aritmetik Lojik Devreler

Hazırlayanlar
1) 200102002025 – Umut Mehmet ERDEM
2) 200102002066 – Emre TANER

İçindekiler

1. Giriş

Bu deneyde aşağıda verilen maddeler amaçlanmaktadır:

1. Donanım tanıma dillerini (DTD) kullanarak devre tasarımı yapmak.

2. Aritmetik devreleri yapısal formda hiyerarşik olarak tasarlamak.

3. Sentezleyici araçları kullanarak, DTD ile tanımlanan aritmetik devreleri FGPA için sentezlemek.

4. Simülasyon araçları kullanarak, otomatik simülasyon yaptırmak.

5. Birleşik lojik devreleri senkronize etmek.

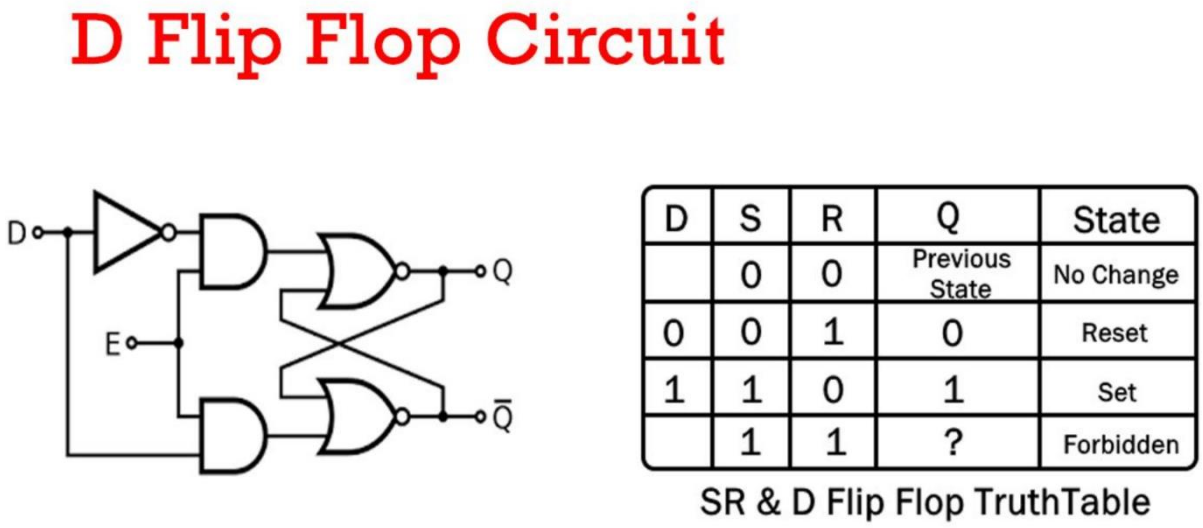
6. Tasarımların fonksiyonel testlerini dosyadan okuma yaparak otomatik şekilde gerçekleştirmek ve sonuçları otomatik şekilde karşılaştırarak raporlamak.

2. Problemler

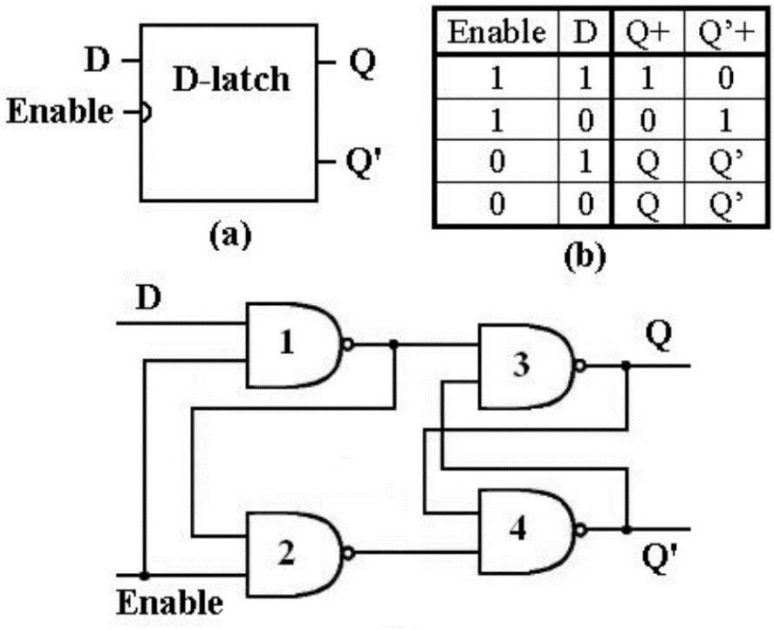
2.1. Problem I – Bellek Elemanlarının Karşılaştırılması

2.1.1. Teorik Araştırma

Elektronik sistemler açısından büyük bir öneme sahip olan ve yaygın kullanılan D-FlipFlop, belli başlı görevleri üzerinden devrelerde değerlendirilmektedir. Özellikle girişlerde uygulanmış olan sinyallerin değiştirilmediği süre içerisinde çıkış durumunu korurlar ve böylece 1 bitlik bilgiyi saklama imkânı verirler.



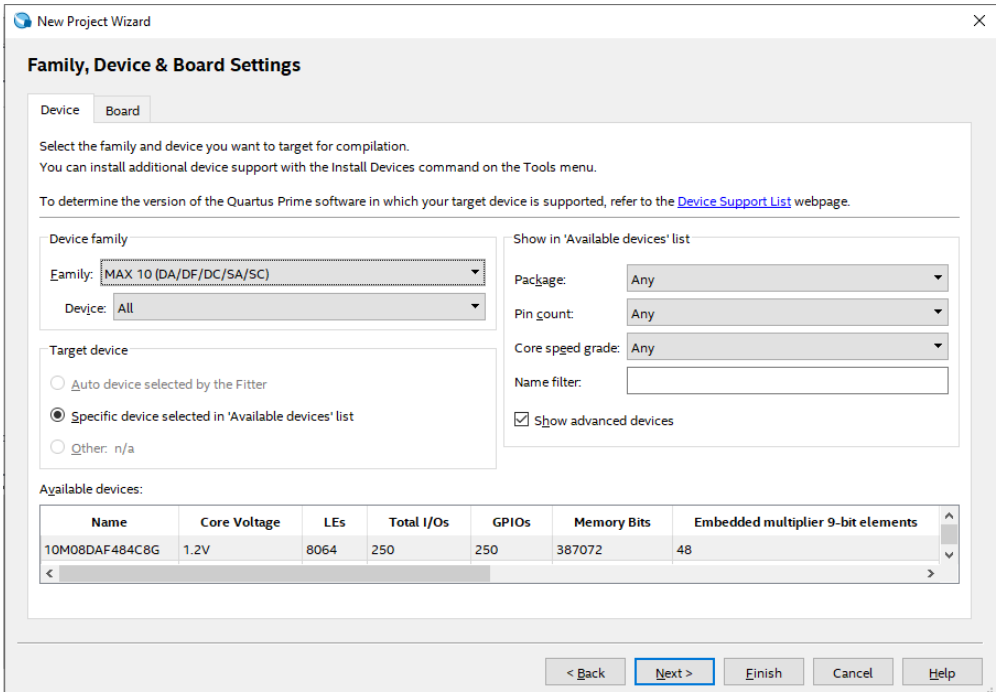
Şekil 1. D-FlipFlop Bellek Elemanının Lojik Tasarımı ve Doğruluk Tablosu



Şekil 2. D-Latch Bellek Elemanının Lojik Tasarımı ve Doğruluk Tablosu

2.1.2 Deneyin Yapılışı

Quartus Prime üzerinden yeni bir proje açılmış açılan projenin üst düzey tasarım varlığının adı (name of top-level design entity) Modelsim programı üzerinden oluşturulan System Verilog (.sv) dosyası ile aynı şekilde isimlendirilmiştir. İleriki adımlarda ".sv" dosyası ve kullanılacak board Şekil 3' teki gibi seçilmiştir.



Şekil 3.Yeni Proje Oluştururken Board Seçimi

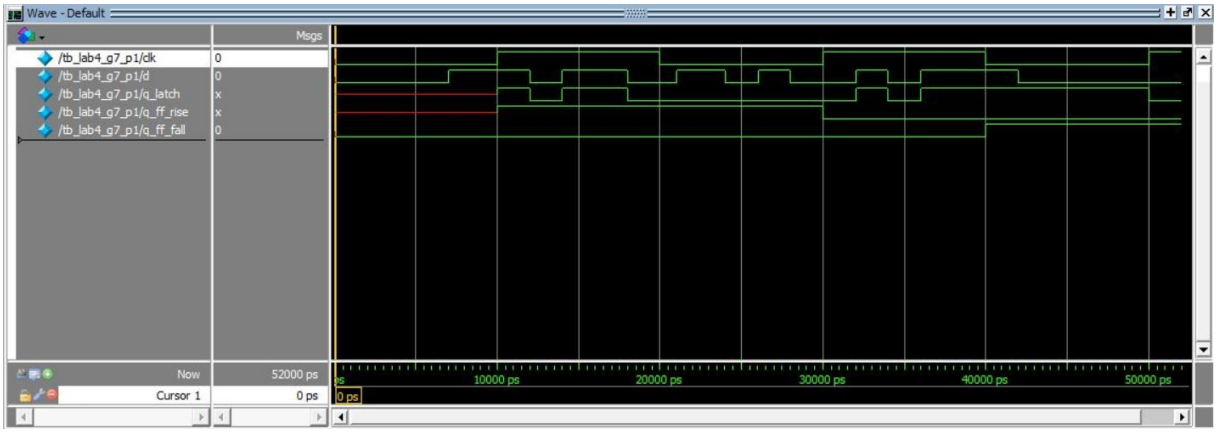
a) Bellek Elemanlarının Karşılaştırılması

Latch, yükselen kenar tetiklemeli D-FlipFlop ve düşen kenar tetiklemeli D-FlipFlop bellek elemanları DTD kullanarak tasarlandı. Tasarlanılan DTD modelini çıkış sinyali gözlemlenecek şekilde simüle edilmiştir. Yazılan kodlar ve zamanlama diyagramı aşağıdaki gibidir:

```
/* lab4_g7_p1.sv
*  Hazırlayanlar: Umut Mehmet ERDEM – Emre TANER
*  Notlar: ELM235 2023 Bahar Lab4 – Problem 1
*  Bellek Elemanlarının Karşılaştırılması
*/
`timescale 1ns/1ps
module lab4_g7_p1(
	input logic clk, d,
	output logic q_latch, q_ff_rise, q_ff_fall
);
	always_ff @(posedge clk) // rising edge
		q_ff_rise <= d;

	always_ff @(negedge clk) // falling edge
		q_ff_fall <= d;

	always_latch
		if(clk) q_latch<=d;
endmodule
```

Şekil 2' de verilen D-Latch diyagramından elde edilen doğruluk tablosu kullanılarak "always_latch" bloğu içersinde clock değeri 1 olduğunda q_latch portu d' ye non-blocking şekilde atanmıştır. Benzer şekilde Şekil 1'deki D-FlipFlop diyagramından elde edilen doğruluk tablosu kullanılarak FlipFlop devreleri için kullanılan "always_ff" bloğu içerisinde yükselen (posedge clk) ve düşen kenar (negedge clk) için clock değerlerinde tetiklenecek şekilde "q_ff_rise" ve "q_ff_fall" değişkenlerine d değerleri atanmıştır.

```
/* tb_lab4_g7_p1.sv
*  Hazırlayanlar: Umut Mehmet ERDEM – Emre TANER
*  Notlar: ELM235 2023 Bahar Lab4 – Problem 1 Testbench
*  Bellek Elemanlarının Karşılaştırılması
*/

// Zaman birimi ve simülasyon çozunurlugu
`timescale 1ns/1ps
module tb_lab4_g7_p1();

// Test tezgahlarinda port bulunmaz
	logic clk, d;
	logic q_latch, q_ff_rise, q_ff_fall;

// Test edilecek modulun yaratimi ve port baglantilarinin yapilmasi
// dut = device under test
	lab4_g7_p1 dut0(clk, d, q_latch, q_ff_rise, q_ff_fall);

// always blogu surekli calistirilacak
// clock oluşturmak için clk sinyalini bir lojik
// seviyeye çekip belli bir süre bekleyip, evirerek geri çekiyoruz
// aşağıdaki haliyle 20ns lik peryotlu 50MHzlik bir clk oluşur

always
begin
	clk = 0; #10; clk = 1; #10; //Duty cycle 50% yarim peryot 10ns
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; // simülasyonu durdur ve beklet. Bitirmek için
$finish;
end
endmodule
```



Şekil 4. Problem 1'in Zamanlama Diyagramı

b) Analiz, Sentez, Kaynak Kullanım Özeti ve RTL, Eşleştirme Sonrası Teknoloji Şeması

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated Total logic elements	3
2		
3	Total combinational functions	1
4	⌄ Logic element usage by number of LUT inputs	
1	-- 4 input functions	0
2	-- 3 input functions	1
3	-- <=2 input functions	0
5		
6	⌄ Logic elements by mode	
1	-- normal mode	1
2	-- arithmetic mode	0
7		
8	⌄ Total registers	2
1	-- Dedicated logic registers	2
2	-- I/O registers	0
9		
10	I/O pins	5
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	d~input
15	Maximum fan-out	3
16	Total fan-out	15
17	Average fan-out	1.15

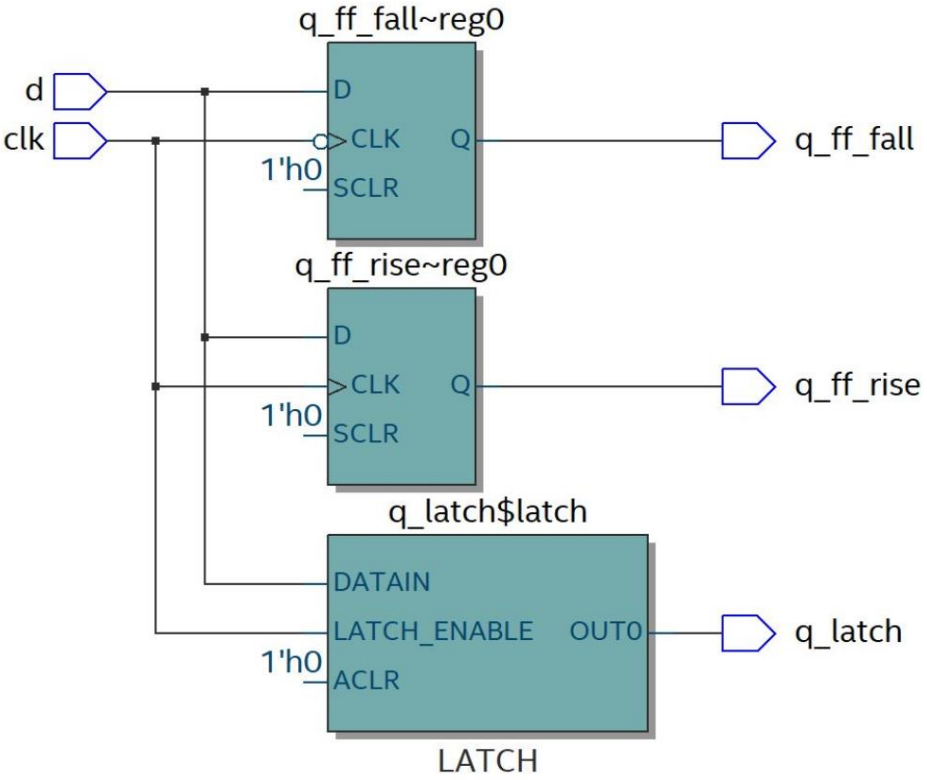
Şekil 5. Problem 1'in Analiz ve Sentez Kaynak Kullanımı Özeti

Flow Summary

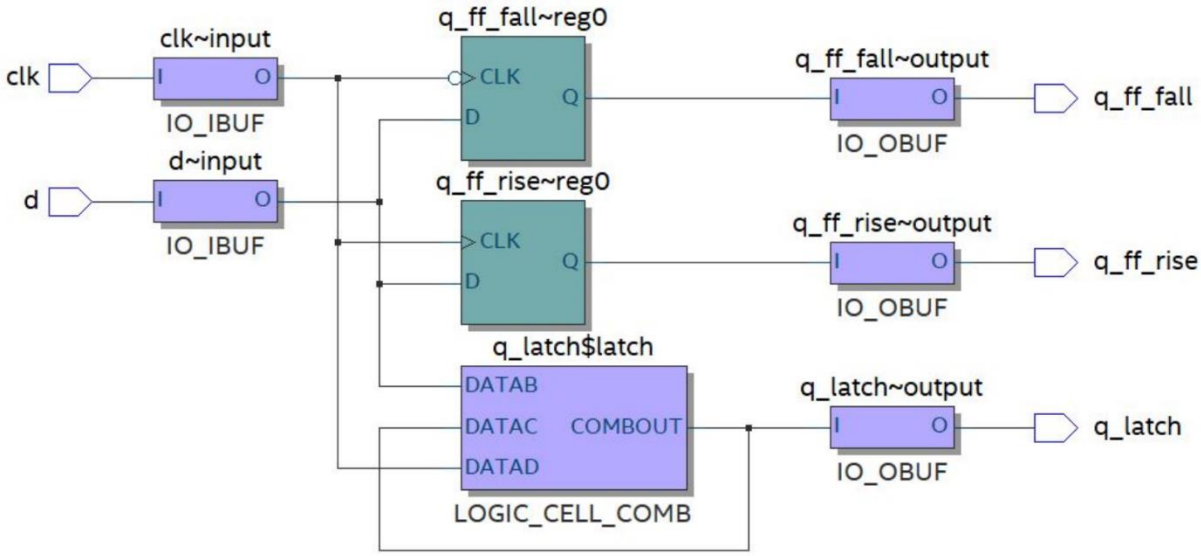
<<Filter>>

Flow Status	Successful - Thu May 25 22:52:00 2023
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	lab4_g7_p1
Top-level Entity Name	lab4_g7_p1
Family	MAX 10
Device	10M08DAF484C8G
Timing Models	Final
Total logic elements	4 / 8,064 (< 1 %)
Total registers	2
Total pins	5 / 250 (2 %)
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 6. Problem 1'in Analiz ve Sentez Özeti



Şekil 7. Problem 1'in RTL Şeması



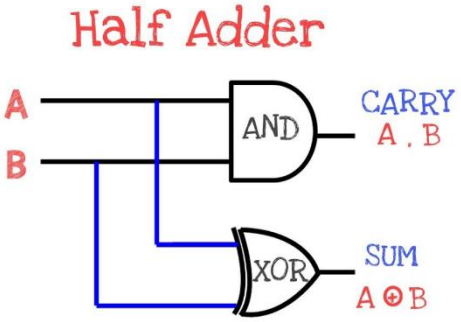
Şekil 8. Problem 1'in Eşleştirme Sonrası Teknoloji Şeması

2.1.3. Sonuçların Yorum

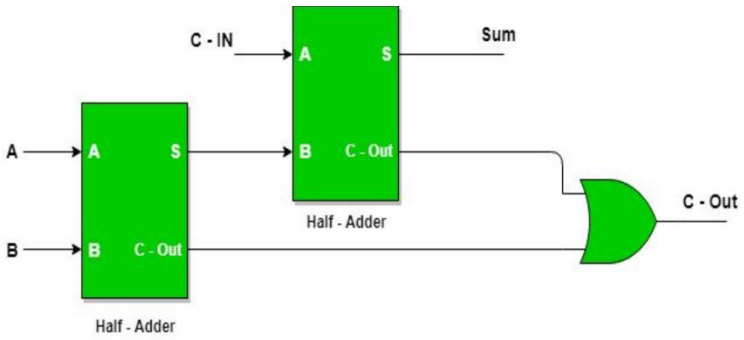
Problem 1'de istenilen Latch, yükselen kenar tetiklemeli D-FlipFlop ve düşen kenar tetiklemeli D-FlipFlop başarılı bir şekilde tasarlandı ve rapora eklendi. Şekil 4' te verilen zamanlama diyagramında "q_latch" portu, D-Latch' in çalışma mekanizmasına uygun olarak "clk" portunun 1 olduğu durumda "d" portunun çıkışını direkt aynısını çıkartır. "q_ff_rise" portu, D-FlipFlop' un çalışmasına uygun olarak "clk" portunun yükselen kenarından önce "d" portunun değerini hafızada tutar ve yükselen kenarda bu değeri çıkartır. "q_ff_fall" portu, D-FlipFlop' un çalışmasına uygun olarak "clk" portunun düşen kenarından önce "d" portunun değerini hafızada tutar ve yükselen kenarda bu değeri çıkartır. Başlangıçta "clk" portu 0 olduğundan dolayı "q_ff_rise" ve "q_latch" sinyal oluşmamıştır. Fakat başlangıçta "clk" portunun düşen kenarında başladığı için değer alıp "q_ff_fall" portundan çıkış sağlanmaktadır. Buna göre, Şekil 4' te verilen zamanlama diyagramının doğru olduğu teyit edilmiştir. Şekil 5 ve Şekil 6' da verilen register miktarı, Şekil 7 ve Şekil 8' de verilen D-FlipFlop'ları olduğu görülmektedir. Latch'ler asenkron çalışır ve giriş sinyalleri tam olarak senkronize olmadığından zamanlama problemleri ve veri bütünlüğü kaybına neden olabilir. Latch'ler tasarım karmaşıklığını artırır. Daha fazla kontrol sinyali gerektirirler. Bu da tasarımın hatalı olma olasılığını artırır ve tasarım sürecini karmaşıklaştırır.

2.2. Problem II- Aritmetik Devrelerin Tasarımı

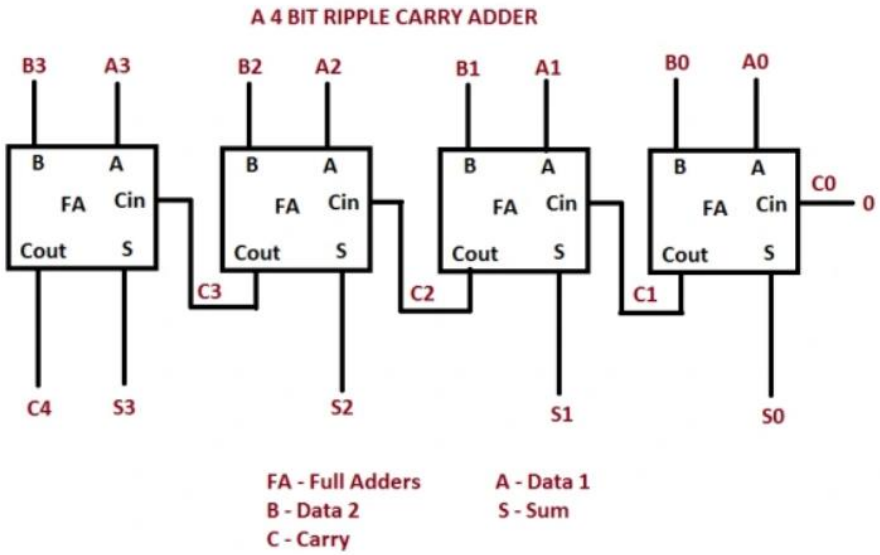
2.2.1. Teorik Araştırma



Şekil 9. Half-Adder Lojik Tasarımı



Şekil 10. Full-Adder Lojik Tasarımı



Şekil 11. Ripple-Carry Adder Lojik Tasarımı

2.2.2. Deneyin Yapılışı

a) Half-Adder ile Donanım Tanımlama Dili Kullanarak Full-Adder Yapımı, Oluşturulan Full-Adder ile DTD Kullanarak Ripple-Carry Adder Yapımı

Half-Adder kullanarak Full-Adder tasarlanmıştır. Daha sonra tasarlanılan Full-Adder ile 5-bitlik bir Ripple-Carry Adder hiyeraşik olarak tasarlanmıştır. Yazılan kod aşağıdaki gibidir:

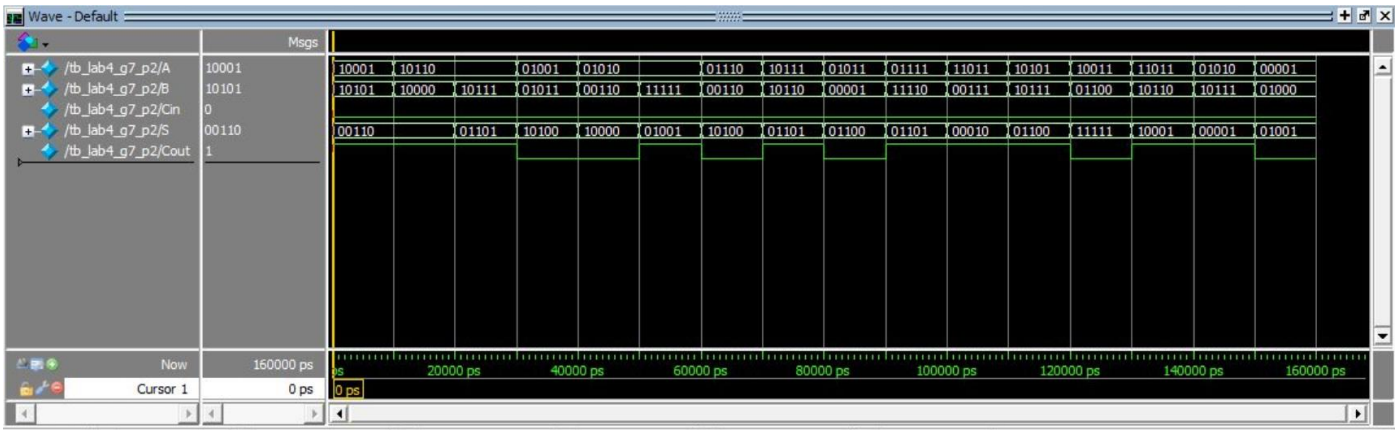
```
/* lab4_g7_p2.sv
*  Hazırlayanlar: Umut Mehmet ERDEM – Emre TANER
*  Notlar: ELM235 2023 Bahar Lab4 – Problem 2
*  Half-Adder ile Donanım Tanımlama Dili Kullanarak Full-Adder Yapımı
*  Oluşturulan Full-Adder ile DTD Kullanarak Ripple-Carry Adder Yapımı
*/
`timescale 1ns/1ps
module lab4_g7_p2(
      input logic [4:0] A, B,
      input logic Cin,
      output logic [4:0] S,
      output logic Cout
);
ripple_carry_adder ripple_carry_adder(A[4:0], B[4:0], Cin, Cout, S[4:0]);
endmodule
module ripple_carry_adder
(
      input logic [4:0] A, B,
      input logic Cin,
      output Cout,
      output logic [4:0] S
);
```

```
logic cout1, cout2, cout3, cout4;
      full_adder full_adder1(A[0], B[0], Cin, cout1, S[0]);
      full_adder full_adder2(A[1], B[1], cout1, cout2, S[1]);
      full_adder full_adder3(A[2], B[2], cout2, cout3, S[2]);
      full_adder full_adder4(A[3], B[3], cout3, cout4, S[3]);
      full_adder full_adder5(A[4], B[4], cout4, Cout, S[4]);
endmodule

module full_adder(
      input logic A, B, Cin,
      output logic Cout, S
);

logic Cout1, Cout2, S1;

half_adder half_adder_1(A, B, Cout1, S1);
half_adder half_adder_2(Cin, S1, Cout2, S);

assign Cout = Cout1 | Cout2;
endmodule

module half_adder(
      input logic A, B,
      output logic Cout, S
);
assign Cout = A&B;
assign S = A^B;
endmodule
```

2.problemde; "half_adder" modülü içerisinde A, B giriş portları ve Cout, S çıkış portları tanımlanmıştır. Çıkış portlarına Şekil 9' da verilen boole denklemleri atanmıştır.Full adder'ın çalışma mekanizmasına göre half adder' da bulunmayan Cin giriş portu "full_adder" modülü içerisinde ek olarak tanımlanmış ve Şekil 10' da verilen diyagrama göre oluşturulmuştur. "ripple_carry_adder" modülünde 5 bitlik olacak şekilde 5 girişlik A ve B portları, tek bitlik Cin giriş ve Cout çıkış portu, 5 bitlik "S" değişkeni ile ifade edilen sum portu tanımlanmıştır. Şekil 11' de verilen RCA diyagramına göre 5 bitlik çıkış alacağımızdan dolayı 5 tane "full_adder" modülü tanımlanmış ve LSB'lerden başlanarak MSB'lerin toplamına gidecek şekilde S bitleri bulunmuştur. Cout portları baştaki full_adder haricinde Cout portları Cin portlarına girecek şekilde parametre olarak "full_adder" modülleri içerisinde verilmiştir. Ana modül içerisinde A, B, C'in girişleri ve Cout, S çıkış çıkış bitleri tanımlanmış ve oluşturulan "ripple_carry_adder" modülü içerisinde parametre verilmiş ve TestBench dosyası içerisinde giriş parametreleri verilerek S ve Cout çıkış portlarının değerleri Şekil 12' de görülmektedir.

b) Ripple-Carry Adder İçin Tasarlanılan DTD Modelinin Simüle Edilmesi

```
/* tb_lab4_g7_p2.sv
*  Hazırlayanlar: Umut Mehmet ERDEM – Emre TANER
*  Notlar: ELM235 2023 Bahar Lab4 – Problem 2 Testbench
*  Half-Adder ile Donanım Tanımlama Dili Kullanarak Full-Adder Yapımı
*  Oluşturulan Full-Adder ile DTD Kullanarak Ripple-Carry Adder Yapımı
*/
// Zaman birimi ve simülasyon çozunurlugu
`timescale 1ns/1ps
module tb_lab4_g7_p2();
// Test tezgahlarinda port bulunmaz
       logic [4:0] A, B;
       logic Cin;
       logic [4:0] S;
       logic Cout;

// Test edilecek modulun yaratimi ve port baglantilarinin yapilmasi
// dut = device under test
       lab4_g7_p2 dut0(A, B, Cin, S, Cout);
// Bu kisimda sinyaller test edilen devreye sirali olarak uygulanir.
// Sonuclar test edilen devre cikislarinda gozlenebilir.

initial begin
       A=5'b10001; B=5'b10101; Cin=0;  #10
       A=5'b10110; B=5'b10000;         #10
       A=5'b10110; B=5'b10111;         #10
       A=5'b01001; B=5'b01011;         #10
       A=5'b01010; B=5'b00110;         #10
       A=5'b01010; B=5'b11111;         #10
       A=5'b01110; B=5'b00110;         #10
       A=5'b10111; B=5'b10110;         #10
       A=5'b01011; B=5'b00001;         #10
       A=5'b01111; B=5'b11110;         #10
       A=5'b11011; B=5'b00111;         #10
       A=5'b10101; B=5'b10111;         #10
       A=5'b10011; B=5'b01100;         #10
       A=5'b11011; B=5'b10110;         #10
       A=5'b01010; B=5'b10111;         #10
       A=5'b00001; B=5'b01000;         #10
       $stop;
end
endmodule
```



Şekil 12. Ripple-Carry Adder Zamanlama Diyagramı

Şekil 12' de bulunan zamanlama diyagramı incelendiğinde sinyallerin doğru geldiği görülmektedir.

c) Analiz, Sentez, Kaynak Kullanım Özeti ve RTL, Eşleştirme Sonrası Teknoloji Şeması

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated Total logic elements	10
2		
3	Total combinational functions	10
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	0
2	-- 3 input functions	10
3	-- <=2 input functions	0
5		
6	Logic elements by mode	
1	-- normal mode	10
2	-- arithmetic mode	0
7		
8	Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0
9		
10	I/O pins	17
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	rip...t~0
15	Maximum fan-out	2
16	Total fan-out	53
17	Average fan-out	1.20

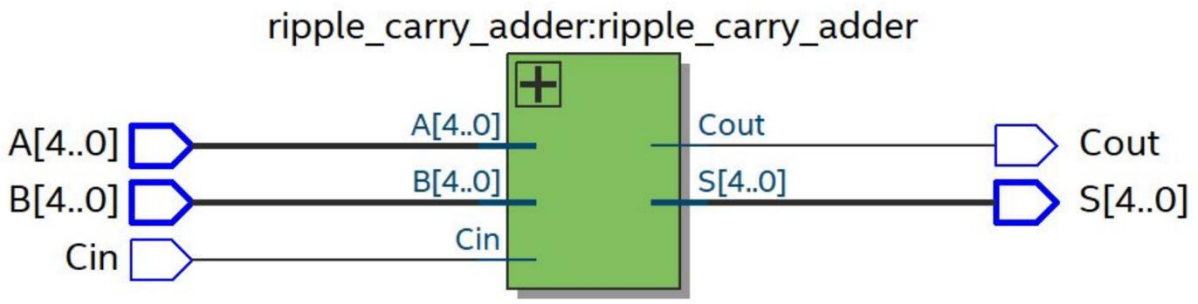
Şekil 13. Problem 2'nin Analiz ve Sentez Kaynak Kullanımı Özeti

Flow Summary

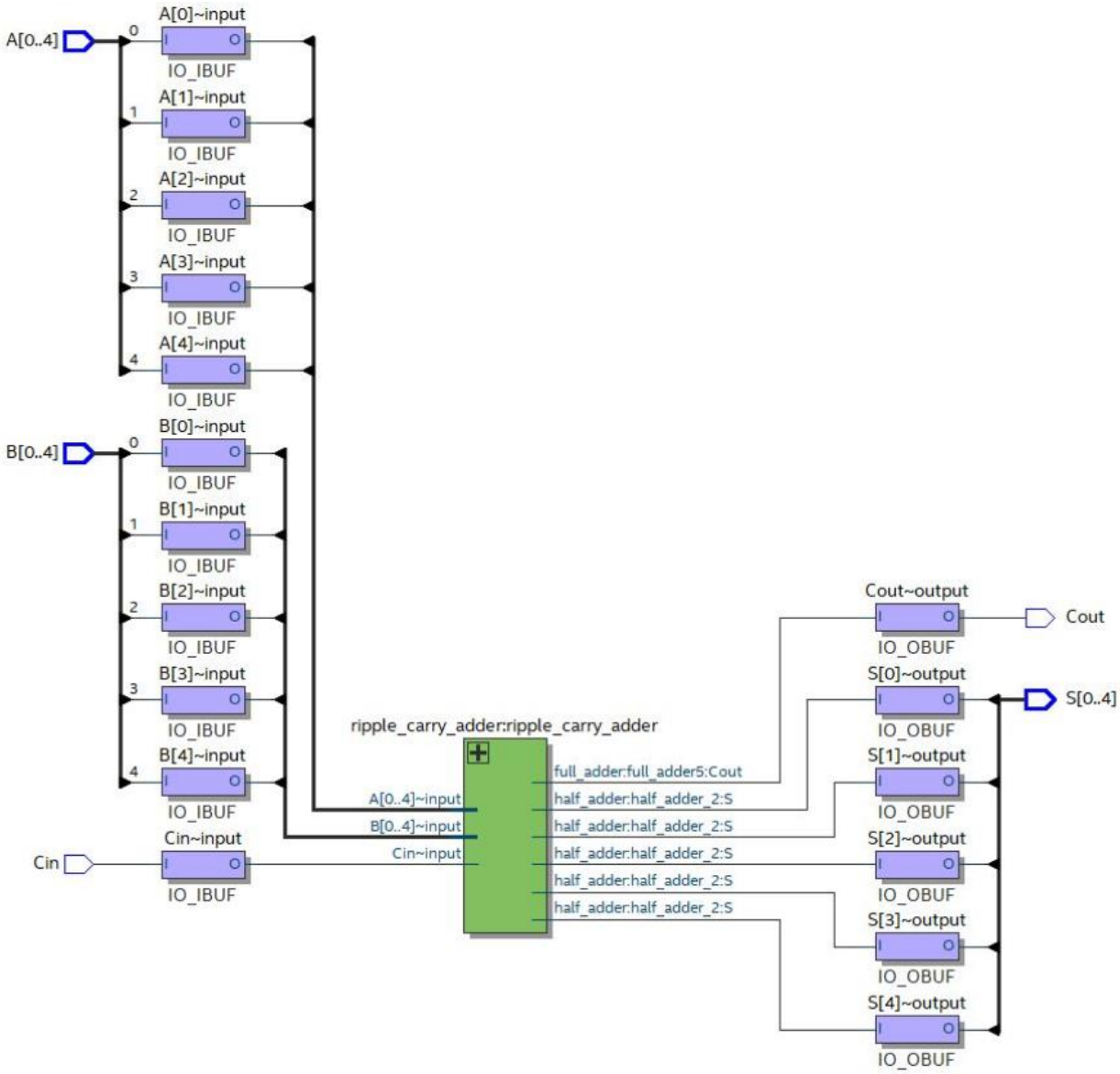
<<Filter>>

Flow Status	Successful - Thu May 25 22:43:35 2023
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	lab4_g7_p2
Top-level Entity Name	lab4_g7_p2
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 14. Problem 2'nin Analiz ve Sentez Özeti



Şekil 15. Problem 2'nin RTL Şeması



Şekil 16. Problem 2'nin Eşleştirme Sonrası Teknoloji Şeması

2.2.3. Sonuçların Yorumu

Problem 2'de istenilen Ripple-Carry Adder başarılı bir şekilde tasarlandı ve rapora eklendi. Modern senkron ardışık lojik devre tasarımlarında latch elemanlarından kaçınılmaktadır. Flip-flop'lar gibi senkronize elemanlar tercih edilir. Flip-flop'lar, senkronize giriş ve çıkış sinyalleriyle çalışır ve metastabilite sorunlarını önlemek için zamanlama sinyalleri kullanılır. Bu sayede daha güvenilir ve öngörülebilir bir çalışma elde edilir. Şekil 14'teki total pin sayısına baktığımızda Şekil 15 ve Şekil 16'daki toplam giriş ve çıkış sayılarını göstermektedir. Şekil 16'da gösterilen teknoloji şemasında verilen Ripple-Carry Adder içerisinde total logic element sayısı 10'dur ve A ve B girişleri 5'er bitlik sayıları ifade etmektedir.

2.3. Problem III- 32-Bitlik Bir Aritmetik Lojik Birimin (ALU) Tasarımı

2.3.1. Deneyin Yapılışı

a) ALU Tasarımını Davranışsal Donanım Tanımlama Dili Kullanarak Gerçeklenmesi

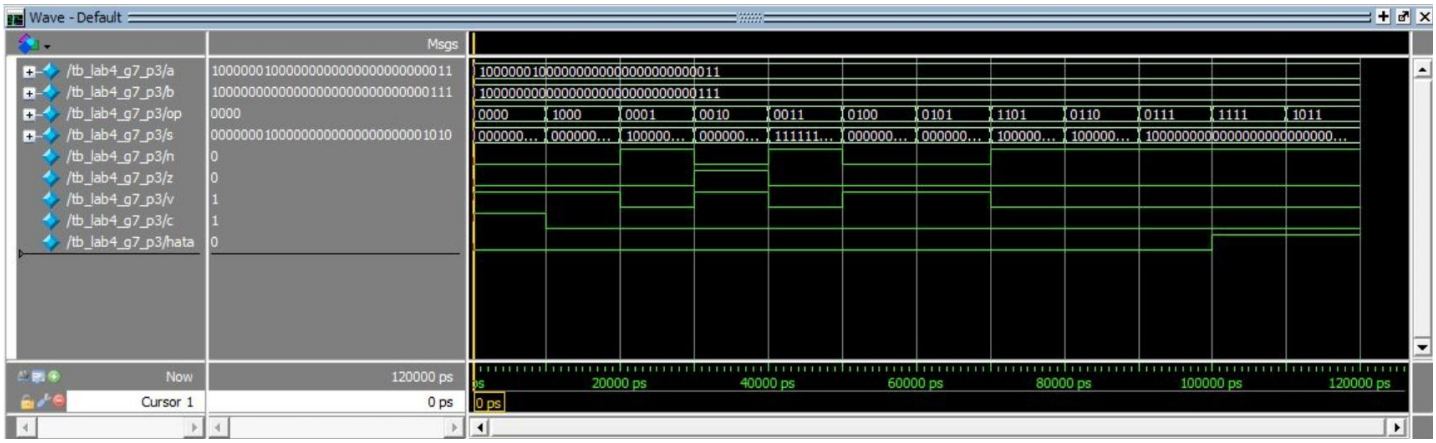
OPCODE detayları verilen 32-bitlik ve NZVC bayraklarını destekleyen bir ALU tasarımını davranışsal DTD ifadeleri kullanarak gerçeklendi. Gerçeklenen DTD modelini çıkış sinyali gözlemlenecek şekilde simüle edilmiştir. Yazılan kodlar ve zamanlama diyagramı aşağıdaki gibidir:

```
/* lab4_g7_p3.sv
*  Hazırlayanlar: Umut Mehmet ERDEM – Emre TANER
*  Notlar: ELM235 2023 Bahar Lab4 – Problem 3
*  ALU Tasarımını DTD Kullanarak Tasarlanması
*/
`timescale 1ns/1ps
module lab4_g7_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
);
always_comb begin
      hata = 0;
      casex (op[3:0])
            4'b0000: {c, s} = a + b;
            4'b1000: {c, s} = a + ~b + 1;
            4'b0001: {c, s} = {1'b0, a << b[4:0]};
            4'b0011: begin c<=1'b0; s <= (b<a)? ~32'b0 : 32'b0; end
```

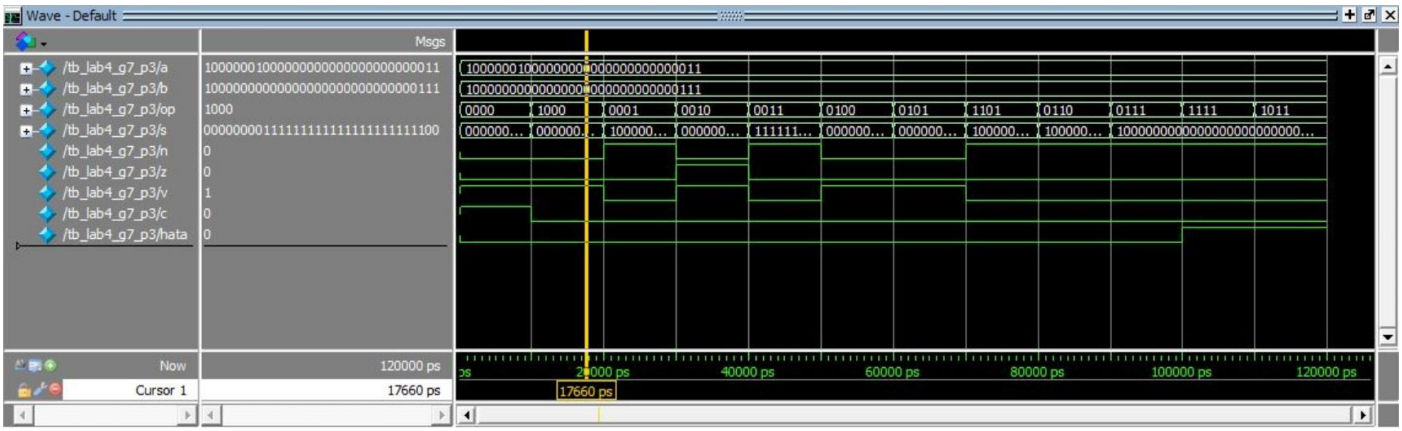
```
            4'b0010: begin
                   c<=1'b0; s <= (a[31]==0 && b[31]==1) ? ~32'b0:
                          ((a[31]==1 && b[31]==0) ? 32'b0:
                          ((a[31]==0 && a[31]==b[31] && b<a)? ~32'b0:
                          ((a[31]==1 && a[31]==b[31] && a<b)? ~32'b0: 32'b0)));
                   end
            4'b0101: {s, c} = {1'b0, a >> b[4:0]};
            4'b1101: {s, c} <= {a[31], a >>> b[4:0]};
            4'b0100: {c, s} = a^b;
            4'b0110: {c, s} = a|b;
            4'b0111: {c, s} = a&b;
            default: hata = 1;
      endcase
      assign n = (s[31]==1'b1)? 1 : 0;
      assign z = (s==32'b0)? 1 : 0;
      assign v = (a[31]==b[31] && s[31]!=a[31])? 1 : 0;
end
endmodule
```

Problem 3' de verilen Tablo 1' e göre, OPCODE değerlerine göre "always_comb" bloğu içerisinde casex-default bloğu oluşturulmuş ve TestBench içerisinde 4 bitlik "op" giriş parametresine giriş değerleri atanmıştır. Buna göre casex bloğunda "0000" için "a" ve "b" 32 bitlik giriş portları toplanarak 32 bitlik "s" çıkış portuna, oluşabilecek carry out değeri "c" portuna atanmıştır. "1000" için "a" ve "b" 32 bitlik giriş portlarını çıkarmak için "b" portunun 2's complement i alınarak "a" portu ile toplanmış ve 32 bitlik "s" çıkış portuna, oluşabilecek carry out değeri "c" portuna atanmıştır. "0001" için "a" portu, "b" portunun LSB den başlayarak 5 bitlik kısmının ondalık sayısı kadar sola kaydırılmış ve "s" çıkış portuna, carry out pini olarak "c" portuna 0 atanmıştır. "0011" için unsigned olduğundan direkt karşılaştırma yapılmış ve şart operatörü kullanılarak "s" parametresine "a" nın büyük olduğu durumda 1 bitleri, olmaması durumda 0 bitleri atanmıştır. "0010" için signed olduğundan şart operatörleri iç içe kullanılarak bir bakıma iç içe if-else blokları oluşturulmuştur. a ve b giriş portlarının MSB değerlerinin farklı olması ilk olarak karşılaştırılmış, olmaması durumda MSB'lerin 0 ve 1 olmasına göre karşılaştırma işlemleri yapılmıştır ve bunun sonucunda "s" çıkış portlarının 1 veya 0 atanmıştır. "0101" için "a" portu, "b" portunun LSB den başlayarak 5 bitlik kısmının ondalık sayısı kadar sağa kaydırılmış ve "s" çıkış portuna, carry out pini olarak "c" portuna değerler atanmıştır. "1101" için aritmetik kaydırma operatörü kullanılmış ve "a" portu, "b" portunun LSB den başlayarak 5 bitlik kısmının ondalık sayısı kadar sağa kaydırılmış, MSB değeri kaydırma işlemi yapıldıktan sonra tekrar MSB bitine tekrar atanmıştır. "s" çıkış portuna, carry out pini olarak "c" portuna değerler atanmıştır. "0100", "0110" ve "0111" için sırasıyla "s" çıkış portuna "a" ve "b" portlarının XOR, OR ve AND işlemlerinin sonucu atanmıştır. Tablo da verilen girdiler haricinde bir OPCODE girdisi verilirse default kısmında hata portu 1' e atanmıştır. casex bloğundan çıkıldıktan sonra n, z ve v bayraklarına koşul operatörüne göre atama yapılmıştır.

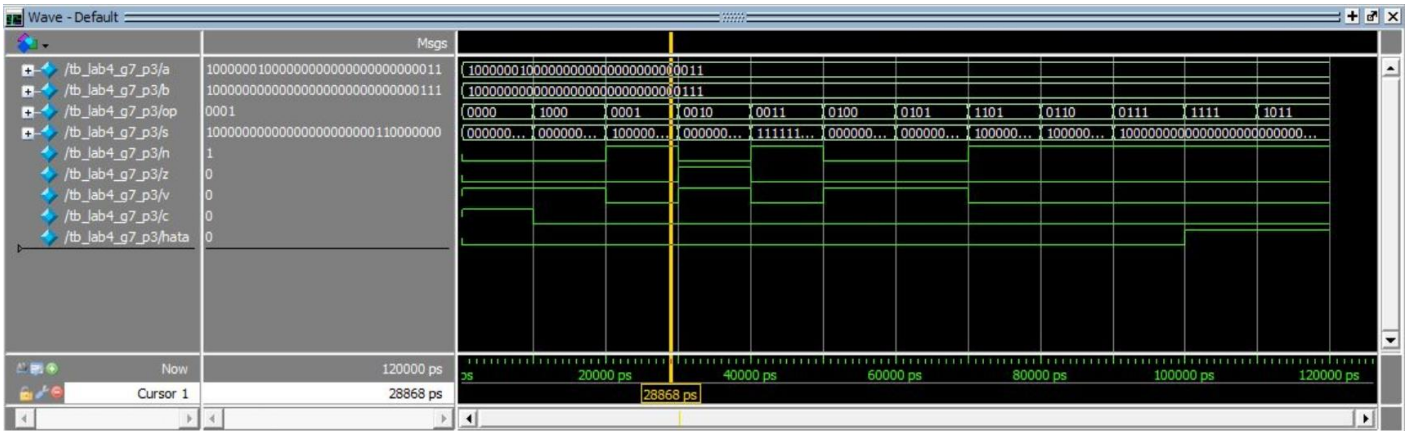
```
/* tb_lab4_g7_p3.sv
*  Hazırlayanlar: Umut Mehmet ERDEM – Emre TANER
*  Notlar: ELM235 2023 Bahar Lab4 – Problem 3 Testbench
*  ALU Tasarımını DTD Kullanarak Tasarlanması
*/
// Zaman birimi ve simülasyon çozunurlugu
`timescale 1ns/1ps
module tb_lab4_g7_p3();
// Test tezgahlarinda port bulunmaz
       logic [31:0] a, b;
       logic [3:0] op;
       logic [31:0] s;
       logic n, z, v, c, hata;
// Test edilecek modulun yaratimi ve port baglantilarinin yapilmasi
// dut = device under test
       lab4_g7_p3 dut0(a, b, op, s, n, z, v, c, hata);
// Sonuclar test edilen devre cikislarinda gozlenebilir.
initial begin
       a=32'h81000003; b=32'h80000007; op=4'b0000; #10
       a=32'h81000003; b=32'h80000007; op=4'b1000; #10
       a=32'h81000003; b=32'h80000007; op=4'b0001; #10
       a=32'h81000003; b=32'h80000007; op=4'b0010; #10
       a=32'h81000003; b=32'h80000007; op=4'b0011; #10
       a=32'h81000003; b=32'h80000007; op=4'b0100; #10
       a=32'h81000003; b=32'h80000007; op=4'b0101; #10
       a=32'h81000003; b=32'h80000007; op=4'b1101; #10
       a=32'h81000003; b=32'h80000007; op=4'b0110; #10
       a=32'h81000003; b=32'h80000007; op=4'b0111; #10
       a=32'h81000003; b=32'h80000007; op=4'b1111; #10
       a=32'h81000003; b=32'h80000007; op=4'b1011; #10
       $stop;
end
endmodule
```



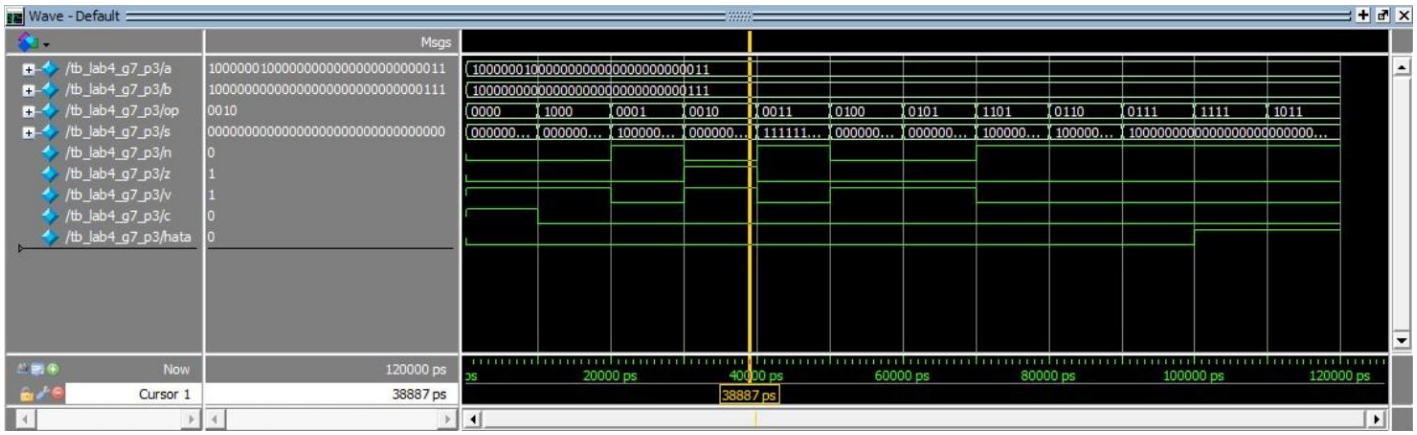
Şekil 17. Problem 3'ün İlk Girdisi İçin Zamanlama Diyagramı



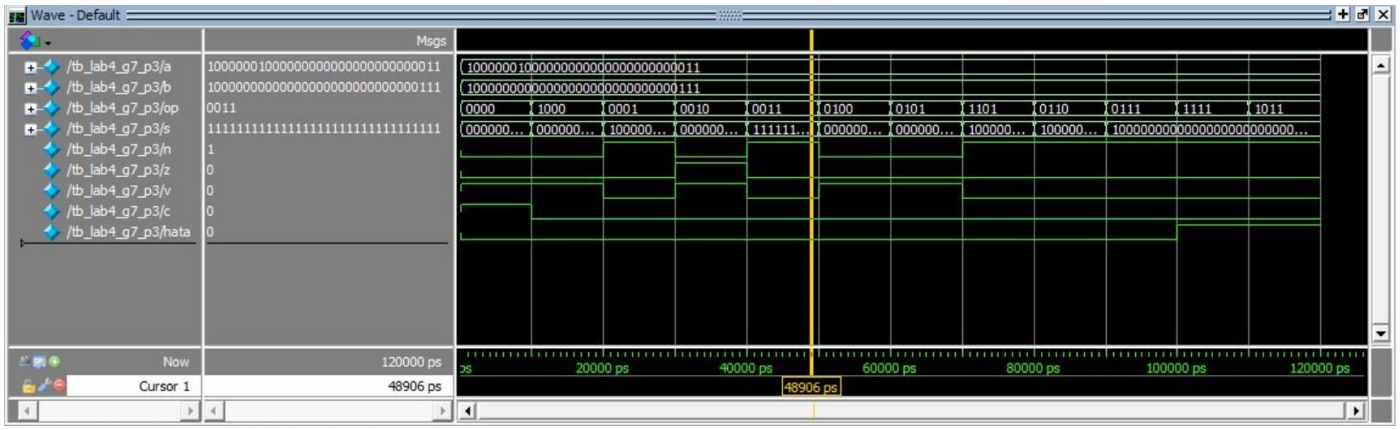
Şekil 18. Problem 3'ün İkinci Girdisi İçin Zamanlama Diyagramı



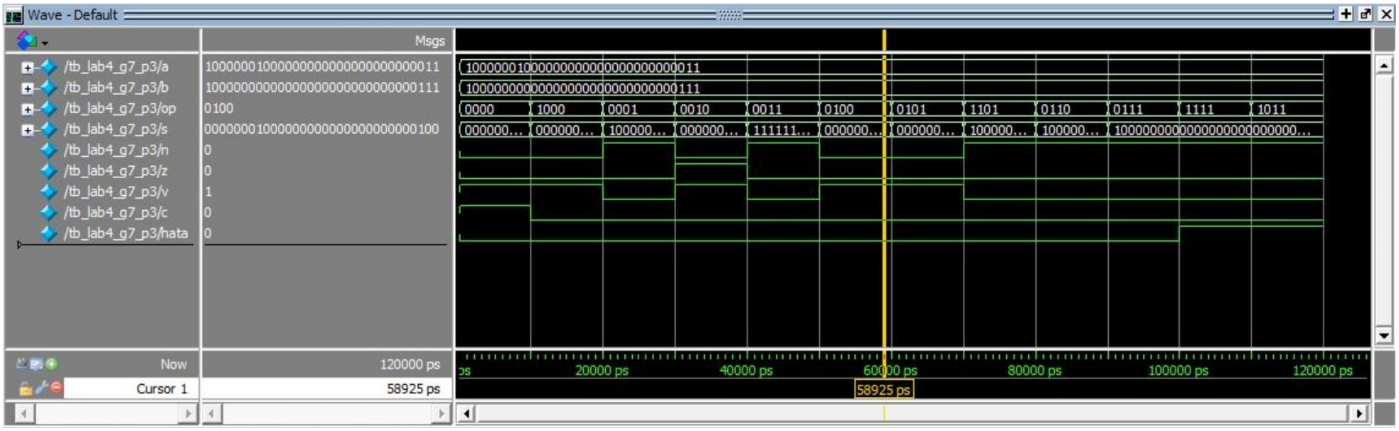
Şekil 19. Problem 3'ün Üçüncü Girdisi İçin Zamanlama Diyagramı



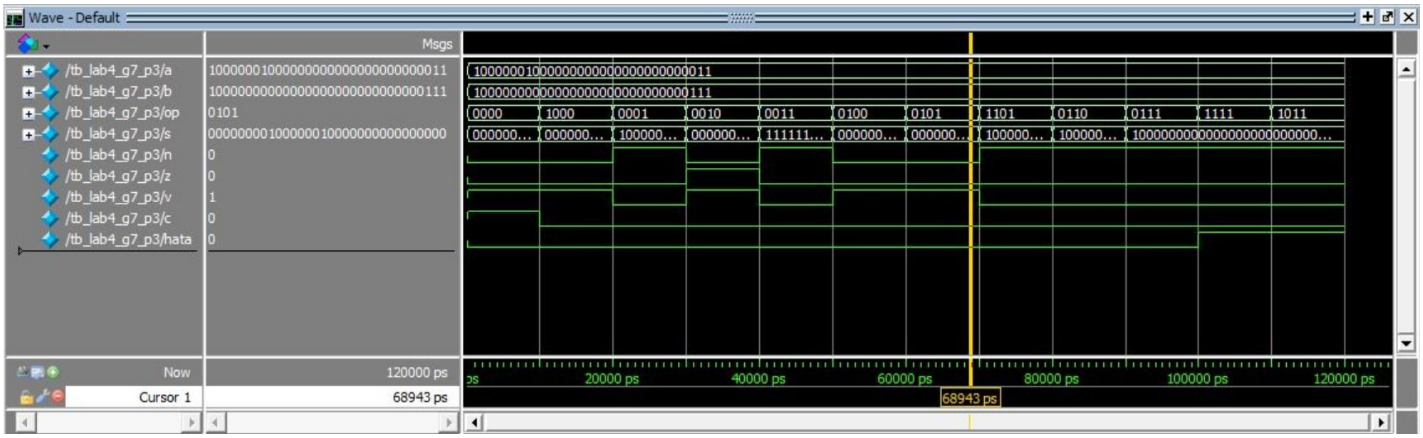
Şekil 20. Problem 3'ün Dördüncü Girdisi İçin Zamanlama Diyagramı



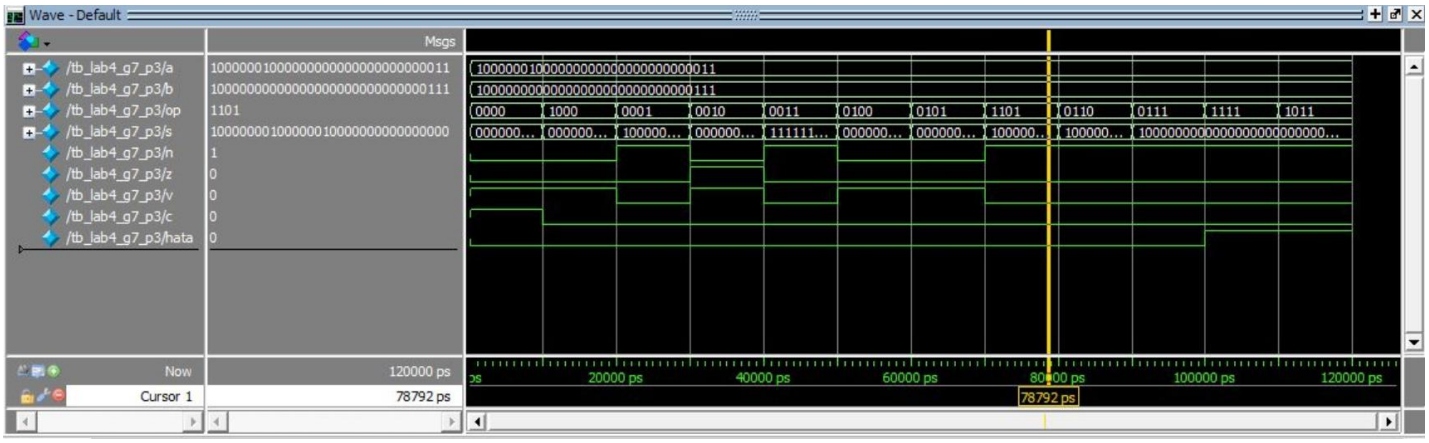
Şekil 21. Problem 3'ün Beşinci Girdisi İçin Zamanlama Diyagramı



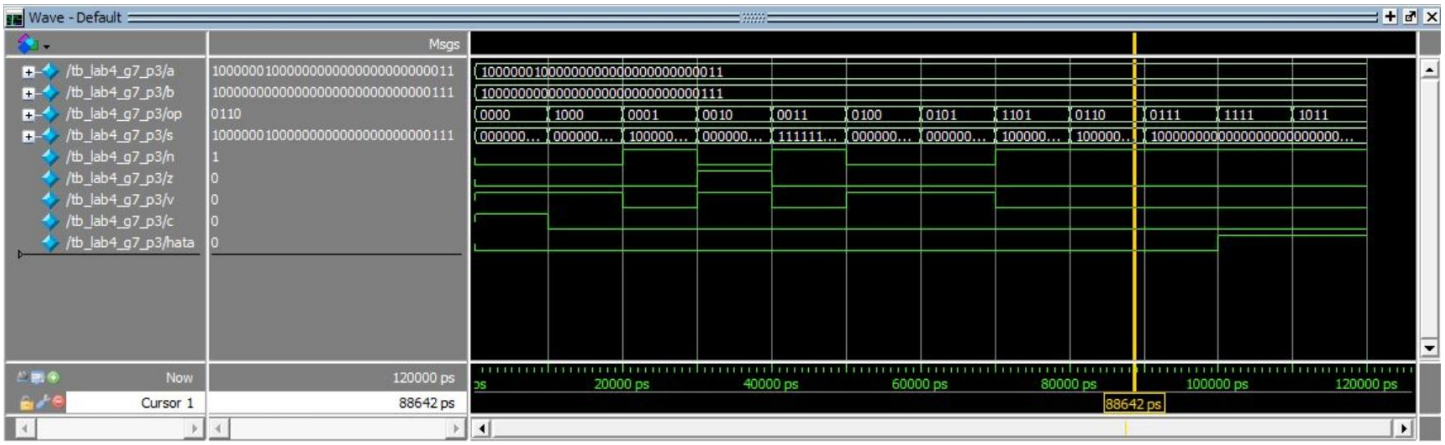
Şekil 22. Problem 3'ün Altıncı Girdisi İçin Zamanlama Diyagramı



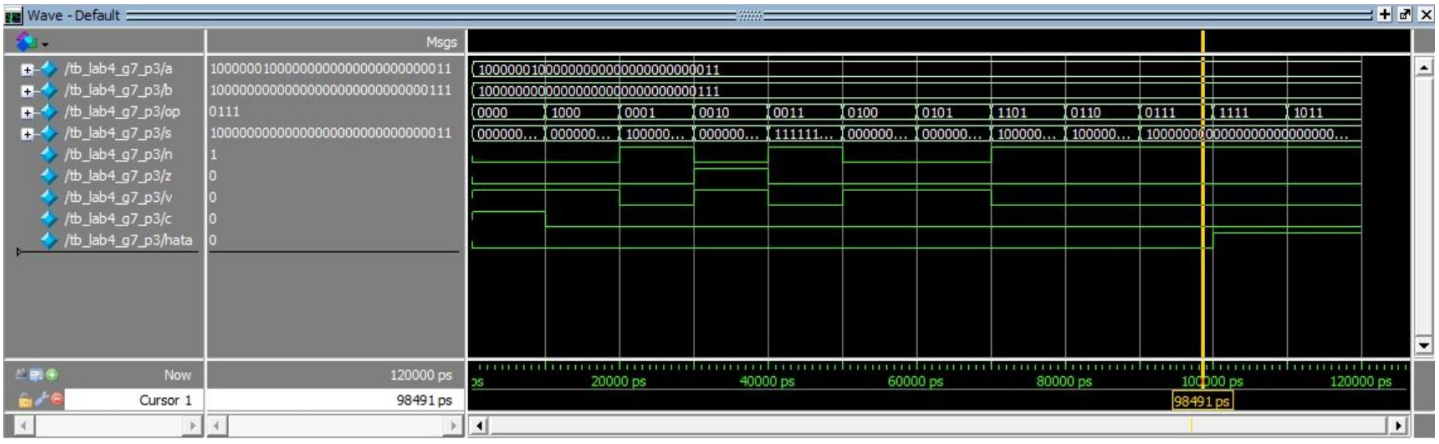
Şekil 23. Problem 3'ün Yedinci Girdisi İçin Zamanlama Diyagramı



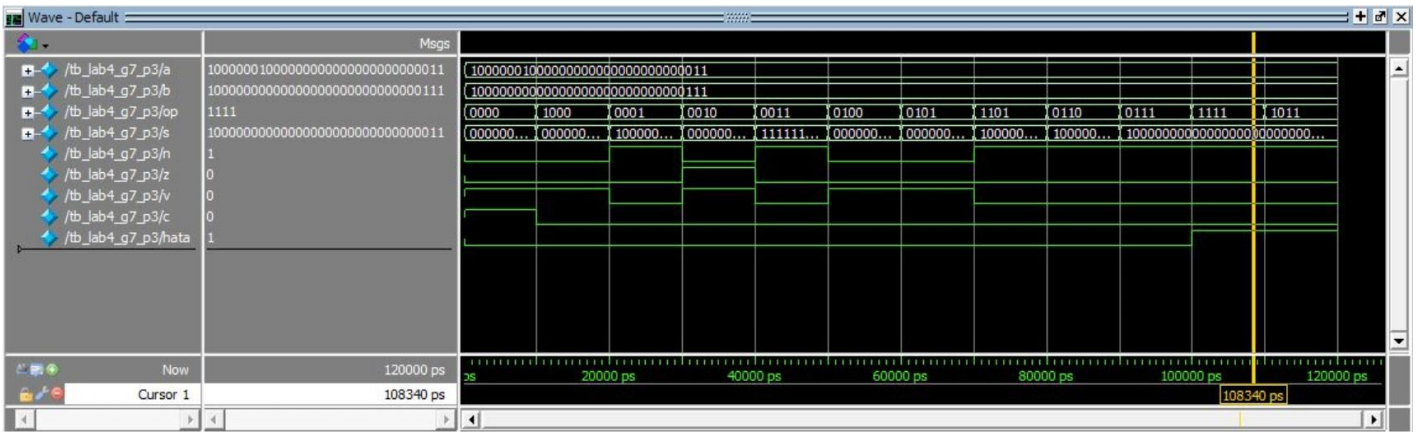
Şekil 24. Problem 3'ün Sekizinci Girdisi İçin Zamanlama Diyagramı



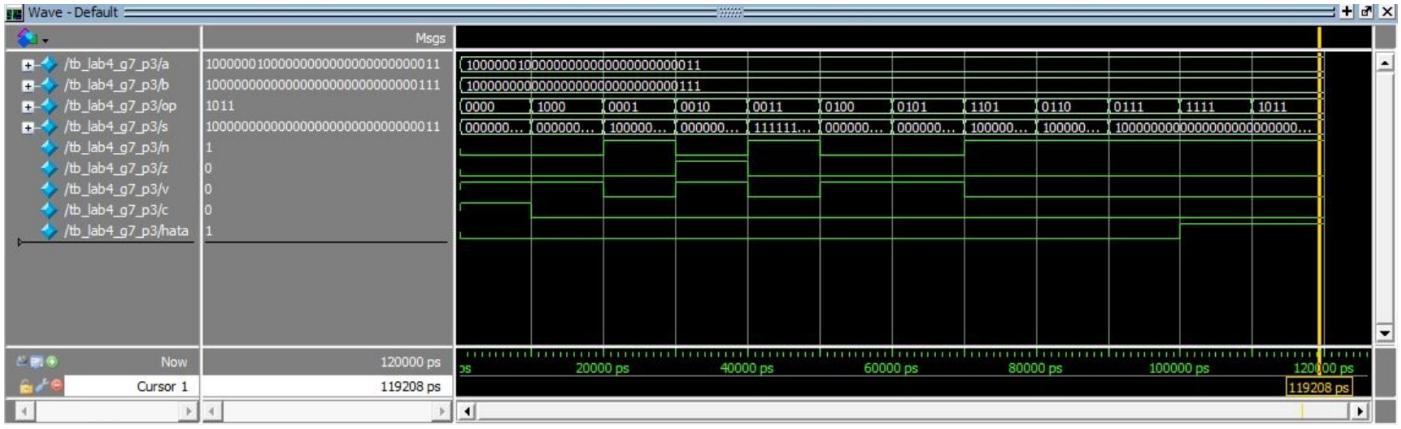
Şekil 25. Problem 3'ün Dokuzuncu Girdisi İçin Zamanlama Diyagramı



Şekil 26. Problem 3'ün Onuncu Girdisi İçin Zamanlama Diyagramı



Şekil 27. Problem 3'ün On Birinci Girdisi İçin Zamanlama Diyagramı



Şekil 28. Problem 3'ün On İkinci Girdisi İçin Zamanlama Diyagramı

Problem 3 için bulunan zamanlama diyagramlarındaki sinyaller her bir "op" port giriş değeri için verilen işlemleri yapmış ve sonuçlar incelendiğinde doğruluğu teyit edilmiştir. Port girişinin farklı girdisi alması durumunda hata portu 1 çıkmıştır.

2.3.2. Sonuçların Yorumu

ALU devrelerinin aynı bit uzunluklarında 2 giriş verilerek yapılacak işlem seçimi sonrasında ona göre çıktı değeri vermesi ve çıkan değere göre bayrak atamaları yapması gerektiğini, bu bayraklara bağlı olarak verilen girdi değerlerinin karşılaştırması yapıldığı öğrenilmiş ve buna göre ALU devresinin kodu yazılmıştır.

3. Sonuçlar ve Genel Yorumlar

Yapılan laboratuvar çalışmasında, D-Latch kullanılarak D-FlipFlop yapılması, clock sinyali kullanılarak verilerin alınması anlaşılmıştır. D-FlipFlop ve clock sinyalinin birlikte kullanılması ile register oluştuğu öğrenilmiştir. Half adder kullanılarak full adder, full adder kullanarak ripple carry adder yapılmış ve nasıl kullanıldıkları ve oluşturuldukları anlaşılmıştır. Verilen föydeki tablodaki giriş değerlerine uygun ALU tasarlanmıştır.

4. Referanslar

[1]https://www.electronicsforu.com/technology-trends/latch-not-bad-latch-vs-flip-flop

[2]https://jamboard.google.com/d/1oYoeZvuaOXVXE6HK3oYmVCPwA0ZGdTQo7So9H8Bo2Qc/edit?usp=meet_whiteboard

[3] https://allthingsvlsi.wordpress.com/2013/04/30/4-bit-ripple-carry-adder/

[4] https://www.hurriyet.com.tr/egitim/d-flip-flop-nedir-ne-ise-yarar-ve-nasil-calisir-d-flip-flop-ozellikleri-ve-kullanim-alanlari-41719245