GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x3 Deney Raporu

Lojik Devreler ve Tasarım Laboratuvarı Dersi

Hazırlayanlar
1) 171024086-Berat KIZILARMUT
2)

1. Giriş

Bu deneyde HDL kullanımına giriş yapıp ilk defa HDL kullanarak mantıksal devreler oluşturacağız. Bunun için normalde kullandığımız Quartus Prime programına ek olarak ModelSim programı da kullanılacak.

2. Problemler

2.1. Problem I - Basit bir devre tasarımı ve simülasyonu

A. HDL Yazılışı

SystemVerilog dili ile ModelSim programında ilk devrenin dosyası hazırlandı.

```
module devre1(
input logic a,b,
      output logic y
 );
      assign y = ~a & b;
endmodule
```

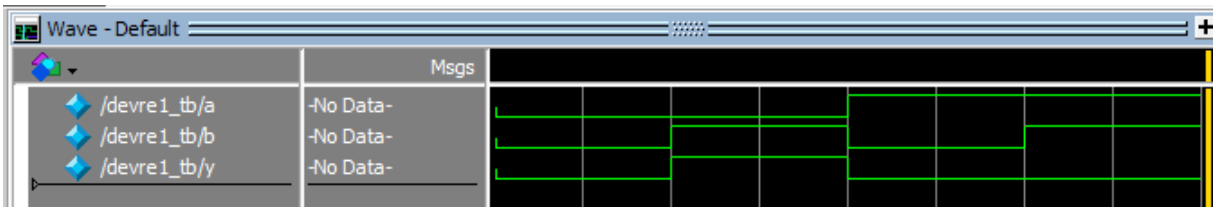
B. Testbench Kuru

Kurulan devreyi sınamak için, devrenin giriş ve çıkış sayısı göz önünde bulundurularak bir testbench oluşturuldu.

```
`timescale  1ns/1ps
module devre1_tb ();
      logic a,b;
      logic y;
      devre1 dut0(a,b,y);
      initial begin
            a=0;b=0;#10;
            b=1;    #10;
            a=1;b=0;#10;
            a=1;b=1;#10;
            $stop;
      end
endmodule
```

C. Waveshape

Testbench'te verilen giriş ve çıkışlara göre waveform sonucu.



2.2 Problem II – Basit bir devrede gecikme simülasyonu

A. Delay

I.Problemdeki devreye gecikmeler eklendi.

```
`timescale  1ns/1ps
module devre2(
input logic a,b,
output logic y
);
assign #3 n1 = ~a;
assign #5 y = n1 & b;
endmodule
```

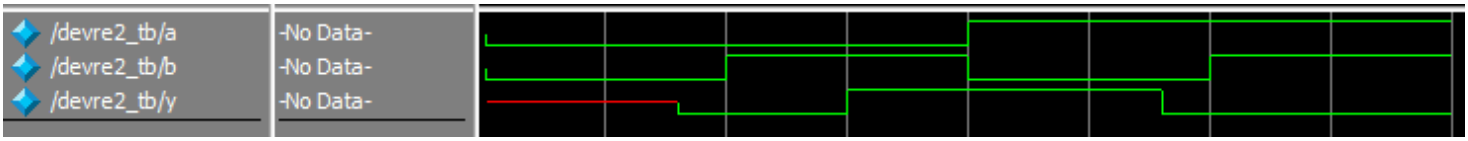
B. Testbench

Değiştirilen devre için testbenchlerde de değişiklik yapıldı. Devre2'nin internal node'u olan n1 testbenchte çağırılmak zorunda kalınmadı.

```
`timescale  1ns/1ps
module devre2_tb ();
      logic a,b;
      logic y;
      devre2 dut0(a,b,y);
      initial begin
            a=0;b=0;#10;
            b=1;    #10;
            a=1;b=0;#10;
            a=1;b=1;#10;
            $stop;
      end
endmodule
```

C. Waveshape

Değişikliklerden sonra waveshape sonucu.



2.3 Problem III – Glitch Simülasyonu

A. HDL Yazılışı

Her bir lojik elemana 2 ns gecikme eklenilerek denklem 2 SystemVerilog dili ile yazıldı.

```
`timescale  1ns/1ps
module devre3(
      input logic a,b,c,d,
      output logic x
 );
      assign #2 n1 = ~b;
      assign #2 n2 = ~c;
      assign #2 n3 = a & n1 & c;
      assign #2 n4 = n2 & d;
      assign #2 x = n3 | n4;
endmodule
```

B. Testbench

Devreye uygun testbench yazıldı. Input sayısı arttığı için yeniden bir giriş değişikliği haritası yazıldı.

```
`timescale  1ns/1ps
module devre3_tb ();
      logic a,b,c,d;
      logic x;
      devre3 dut0(a,b,c,d,x);
      initial begin
            a = 0; b = 0; c = 0; d = 0; #10;
            d = 1;                      #10;
            c = 1; d = 0;               #10;
            d = 1;                      #10;
            b = 1; c = 0; d = 0;        #10;
            b = 1; d = 1;               #10;
            b = 1; c = 1; d = 0;          #10;
            b = 1; d = 1;                 #10;
            a = 1; b = 0; c = 0; d = 0; #10;
            b = 0; d = 1;                 #10;
            c = 1; d = 0;                 #10;
            d = 1;                        #10;
            b = 1; c = 0; d = 0;          #10;
            d = 1;                        #10;
            c = 1; d = 0;                 #10;
            d = 1;                        #10;
      $stop;
      end
endmodule
```

C. Glitch Analizi

X	CD				
AB		00	01	11	10
	00	0	1	0	0
	01	0	1	0	0
	11	0	1	0	0
	10	0	1	1	1

Tablo 1 Devrenin K haritası

X	CD				
AB		00	01	11	10
	00	0	1	0	0
	01	0	1	0	0
	11	0	1	0	0
	10	0	1	1	1

Tablo 2 Glitchli K haritası

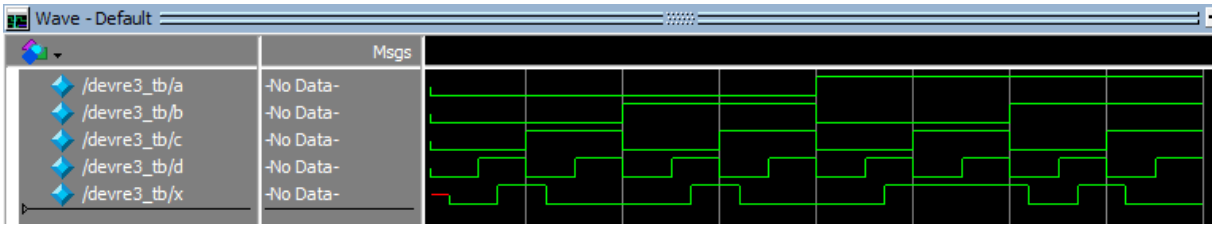
Devrenin asıl KMap'inde dörtlü şekilde alınan kısmı, ikili iki ayrı bölmeye dönüştürerek glitch elde etme şansımızı arttırmış oluruz.

$$X = \bar{A}\&\bar{C}\&D \mid A\&\bar{C}\&D \mid A\&\bar{B}\&C$$

Bunun haricinde başka bir glitch sağlayacak durum bulunmadı.

D. Waveshape

Dört girişli, internal nodelar önemsenmemiş waveform grafiği.



2.4 Problem IV – Çözücü Tasarım

A. 8 Bit Input, 4 Bit Output Encoder Devre Kurulumu

Inputumuz için 8 elemandan oluşan bir array, outputumuz için 4 elemandan oluşan bir array oluşturuldu. 8 bitlik ve 4 bitlik buslar elde etmiş olduk. Arrayler her zaman most significant bit'den least significant bit'e gider şekilde kuruldu.

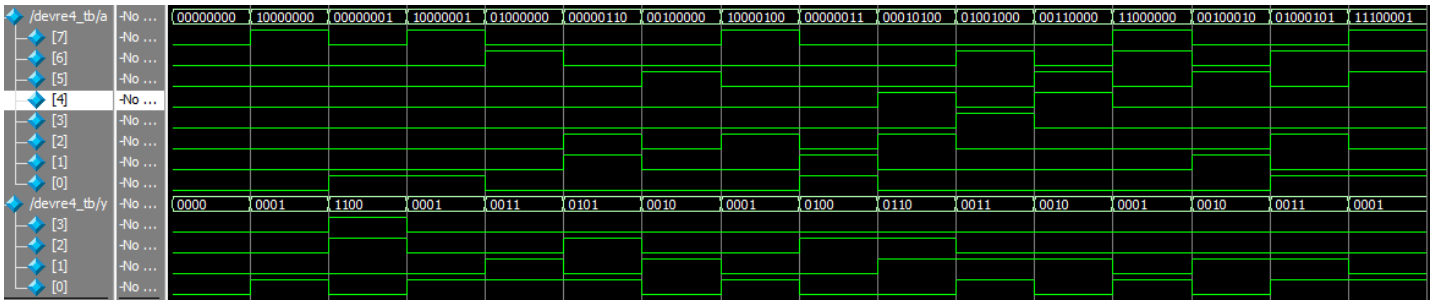
Ternary operator nested formunda kullanılarak, array most significant bit'den, least significant bit'e kadar sorgulandı.

```
module devre4(
      input logic [7:0] a,
      output logic [3:0] y
 );
      assign y = a[7] ? 4'b0001:
                 a[6] ? 4'b0011:
                 a[5] ? 4'b0010:
                 a[4] ? 4'b0110:
                 a[3] ? 4'b0111:
                 a[2] ? 4'b0101:
                 a[1] ? 4'b0100:
                 a[0] ? 4'b1100: 4'b0000;
endmodule
```

B. Testbench ve Waveform

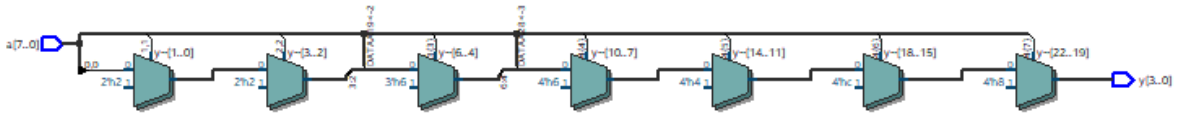
Testbench'te, inputumuzu array olarak vermemizin faydasından yararlanarak, dalga formundaki değişikler, arraye binary sayı girişi olarak verildi. Bu sayede simülasyona giriş yapmak ve simülasyonu okumak oldukça kolaylaştı.

```
             `timescale  1ns/1ps
             module devre4_tb ();
                    logic [7:0]a;
                    logic [3:0]y;
                    devre4 dut0(a,y);
                    initial begin
                                 a = 8'b00000000; #10;
                                 a = 8'b10000000; #10;
                                 a = 8'b00000001; #10;
                                 a = 8'b10000001; #10;
                                 a = 8'b01000000; #10;
                                 a = 8'b00000110; #10;
                                 a = 8'b00100000; #10;
                                 a = 8'b10000100; #10;
                                 a = 8'b00000011; #10;
                                 a = 8'b00010100; #10;
                                 a = 8'b01001000; #10;
                                 a = 8'b00110000; #10;
                                 a = 8'b11000000; #10;
                                 a = 8'b00100010; #10;
                                 a = 8'b01000101; #10;
                                 a = 8'b11100001; #10;
                    $stop;
                    end
             endmodule
```

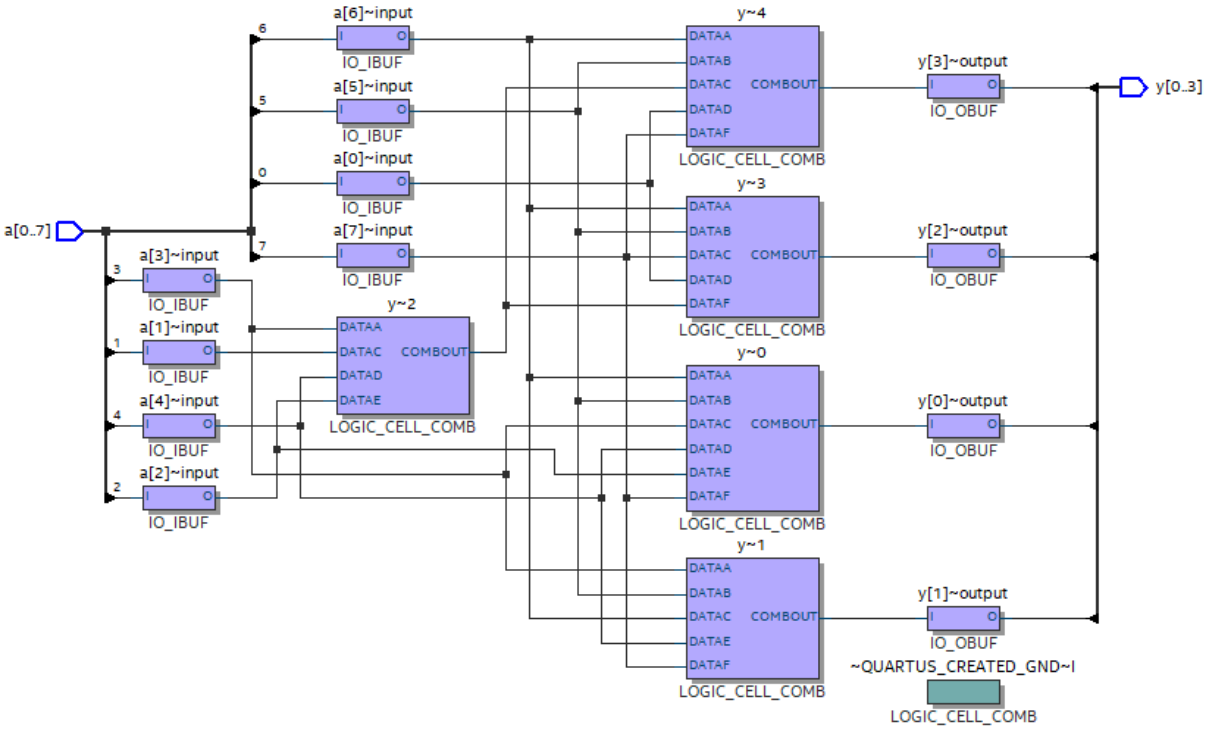


C. RTL ve Post Fitting Şemaları

ModelSim'de yazılan SystemVerilog dosyası, Quartus Prime'a import edildi ve compile edildi.

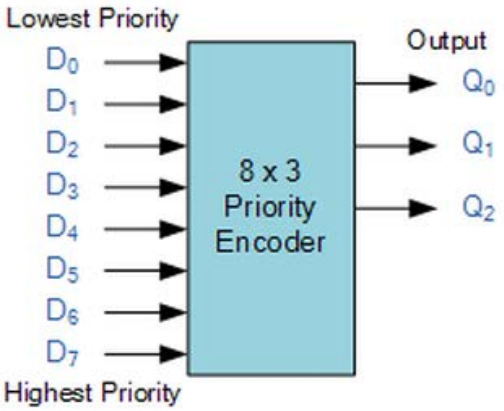


Şekil 1 RTL Şeması



Şekil 2 Post-Fitting Haritası

D. Alternatif Yollar



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = don't care

Bu işlevi görecek bir devre Priority Encoder ile de yapılabilir. Girişleri ve çıkışları binary 8 ve 4 bitlik sayıları verecek şekilde değiştirebiliriz. Sonuç olarak daha fazla lojik kapı kullanmış olacağız, fakat paketlerin soyutlanması ile bu devre priority encoder ile daha sade bir görünüş elde edebilir.

2.5 Problem V – Yapısal Tasarım

A. 8-Bit 2to1 Multiplexer

Basit bir multiplexer tasarımı yapıldı. Parametlerin değişimi ile giriş ve çıkış bitleri arttırılıp azaltılabilir.

```
module devre5(
      input logic [7:0] a,b,
      input logic s,
      output logic [7:0] y
 );
      logic n1;
      logic [7:0] n2,n3;
      assign n1 = ~s;
      assign n2 = {8{n1}} & a;
      assign n3 = {8{s}} & b;
      assign y = n2 | n3;
endmodule
```

B. 8-Bit 4to1 Multiplexer

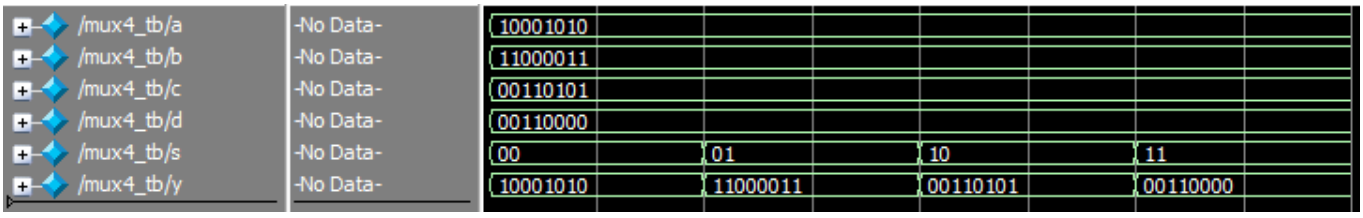
8 bitlik 2to1 Muxı kullanıp, paralel bağlayarak 4to1 Mux yapabiliriz. Bunu yapınca S değerimiz de 2 elemanlı bir array olacak.

```
module mux4(
      input logic  [7:0] a,b,c,d,
      input logic  [1:0]s,
      output logic [7:0] y
 );
      logic [7:0] n1,n2;
      devre5 mux1(a,b,s[0],n1);
      devre5 mux2(c,d,s[0],n2);
      devre5 mux3(n1,n2,s[1],y);
endmodule
```

C. Testbench ve Waveform

Devre beklenildiği gibi çalıştı ve doğru 8 bitlik girişleri seçti.

```
`timescale  1ns/1ps
module mux4_tb ();
      logic [7:0]a,b,c,d;
      logic [1:0]s;
      logic [7:0] y;
      mux4 dut0(a,b,c,d,s,y);
      initial begin
            a = 8'b10001010; b = 8'b11000011;
      c = 8'b00110101; d = 8'b00110000;
      s[0]=0;s[1]=0;  #10;
      s[0]=1;         #10;
      s[0]=0;s[1]=1;     #10;
      s[0]=1;            #10;
      $stop;
      end
endmodule
```

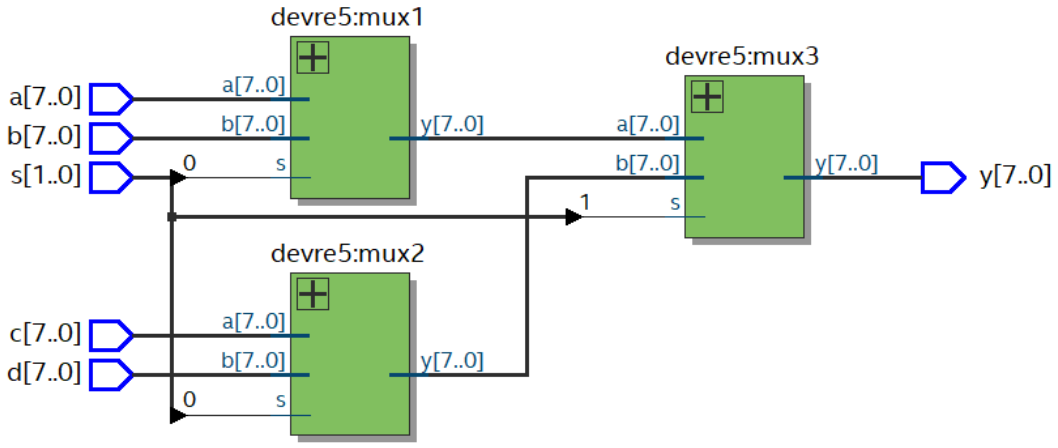


D. RTL ve Post-Fitting Şemaları

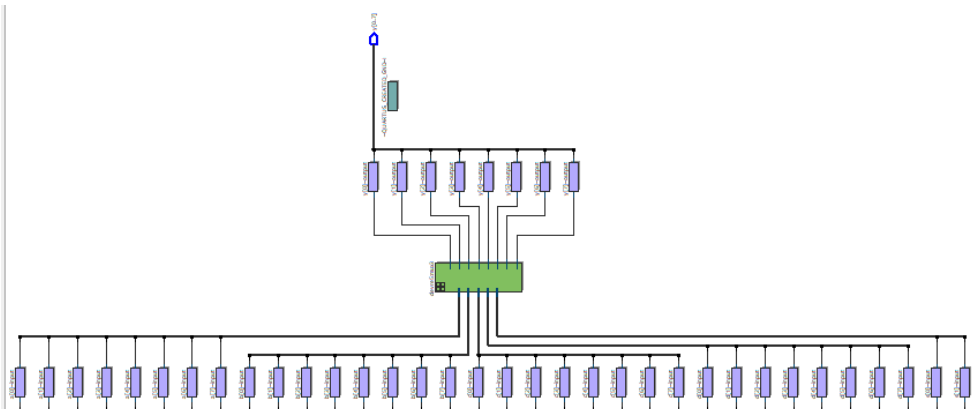
	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	8
2		
3	Combinational ALUT usage for logic	8
1	-- 7 input functions	0
2	-- 6 input functions	8
3	-- 5 input functions	0
4	-- 4 input functions	0
5	-- <=3 input functions	0
4		
5	Dedicated logic registers	0
6		
7	I/O pins	42
8		
9	Total DSP Blocks	0
10		
11	Maximum fan-out node	s[0]~input
12	Maximum fan-out	8
13	Total fan-out	98
14	Average fan-out	1.07

Kod, Quartus Prime'a aktarıldı ve compilelanıp analizlendi.

Toplam olarak bütün devrede 6 giriş fonksiyonlardan 8 adet kullanıldı.



Şekil 4 RTL Şeması



Şekil 3 Post-Fitting Haritası

E. Yapısal Tasarım

Yapısal tasarımın amaçları, bir projede birden fazla yerde kullanılacak veya farklı varyasyonları kullanılacak paketler hazırlayarak, lojik devre tasarımında çok ciddi zamanlar kazanılabilir.

Yeni bir fonksiyon ve işlev yazımındaki oluşabilecek hatalardan, doğru ve çalışan bir fonksiyonu tekrar çağırıp kullanarak kurtulabiliriz.

Projemizde bize gerekecek şeyleri mantıksal temel taşları gibi birbirinin üstüne ekleyerek geliştirerek çok daha büyük ve fazla işlevli mantıksal paketler elde ederiz.

3. Genel Yorumlar ve Kazanımlar

Bu deneyde ModelSim yazılımının kullanımı öğrenildi. SystemVerilog dili ilk defa kullanılarak, C++'dan SV'ye geçişte alışma süreci yaşandı. Şematik devre tasarımına göre VHDL dilleri ile devre kurulumu çoğu durumda hız kazandırdı. Bir problem olduğunda tespit edilmesi daha kolaylaştı.

Yapısal tasarımın prensipleri ve faydaları pratik şekilde yaşanıldı ve öğrenildi. Büyük devreleri şematik şekilde kurulmasına nazaran VHDL dilleri yardımıyla çok kısa bir sürede devreler tasarlanabilir oldu.

4. Kaynaklar

1. Gray code. (2020, April 2). Retrieved from https://en.wikipedia.org/wiki/Gray_code
2. Harris and D. Harris, Digital Design and Computer Architecture: ARM Edition, 1st edition. Morgan Kaufmann, 2015
3. Priority Encoder and Digital Encoder Tutorial. (2018, September 10). Retrieved from https://www.electronics-tutorials.ws/combination/comb_4.html