GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 4 Deney Raporu

Lojik Devreler ve Tasarım Laboratuvarı Dersine Hazırlık

Hazırlayanlar
1) 210102002053 – Şule Nur Demirdaş

1. GİRİŞ

2. Problemler

2.1. Problem 1 - Aritmetik Devrelerin Tasarımı

2.1.1. Teorik Araştırma

Half Adder (Yarım Ekleyici):
Yarım ekleyici, iki tek bitlik girişi (A ve B) alır ve bu girişlerin toplamını (sum) ve taşıma değerini (carry) hesaplar. Yarım ekleyici, sadece iki girişli bir XOR (eşit değilse 1) ve bir AND (her ikisi de 1 ise 1) kapısı kullanarak çalışır. Yarım ekleyici, sadece tek bir basamağı toplama işlemi yapabilen basit bir birimdir.

Full Adder (Tam Ekleyici):
Tam ekleyici, iki tek bitlik giriş (A ve B) ve bir taşıma (carry-in) değeri alır. Bu girişlerin toplamını (sum) ve taşıma değerini (carry) hesaplar. Tam ekleyici, yarım ekleyicilerin kullanılmasıyla oluşturulur. İki yarım ekleyici kullanılarak toplama işlemi gerçekleştirilirken, bir diğer yarım ekleyici ise taşıma değerini hesaplamak için kullanılır. Full adder, bir basamağı toplama işlemi yapabilen bir birimdir ve ardışık ekleyicilerle daha büyük sayıların toplamını hesaplamak için kullanılabilir.

Ripple Carry Adder (Serbest Akışlı Ekleyici):
Serbest akışlı ekleyici, birden çok tam ekleyiciyi bağlayarak daha büyük sayıları toplayabilen bir ekleyici türüdür. Ripple carry adder, basamakları sırayla toplar ve taşıma değerini bir basamağın çıkışından bir sonraki basamağın girişine aktarır. Taşıma değeri bir basamağın hesaplanması için bir önceki basamağın taşıma değerine bağlı olduğu için taşıma değerinin geçiş süresi (ripple) oluşur. Ripple carry adder, basit ve kolay uygulanabilir olmasına rağmen, büyük sayıları toplarken gecikmelere ve yavaşlamalara neden olabilir. Bu nedenle, daha hızlı ve gelişmiş ekleyici tasarımları da kullanılmaktadır.

2.1.2. Deneyin Yapılışı

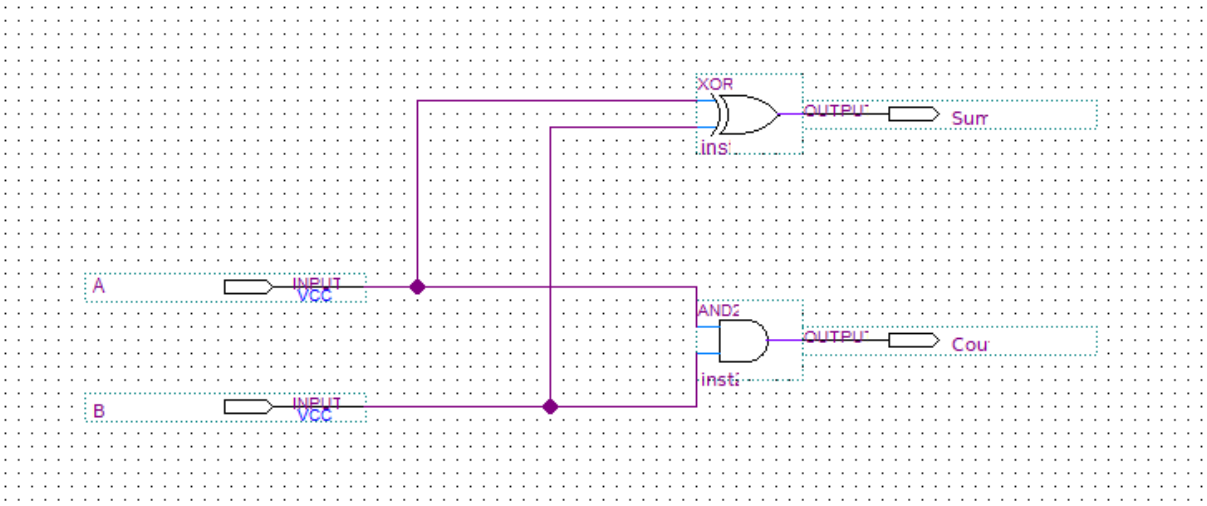
İlk olarak bir half-adder devresi tasarlandı.

a) Half-Adder Devresi Doğruluk Tablosu

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tablo 1: Half-Adder Doğruluk Tablosu

b) Half-Adder Devre Şeması



Şema 1: Half-Adder Devre Şeması

$$Sum = A\ XOR\ B$$

$$C_{OUT} = A\ AND\ B$$

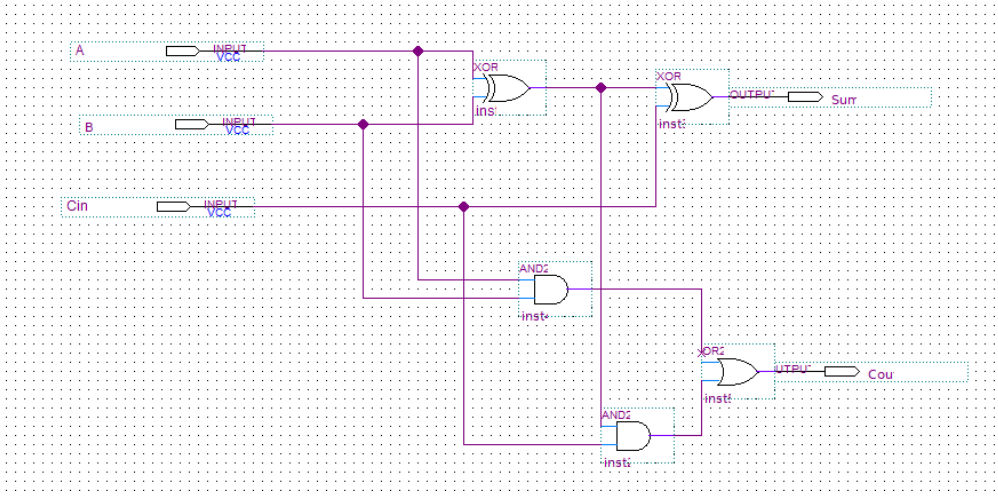
Bu oluşturulan half-adder devresi kullanılarak bir full-adder tasarlandı.

c) Full-Adder Doğruluk Tablosu

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tablo 2: Full-Adder Doğruluk Tablosu

d) Full-Adder Devre Şeması

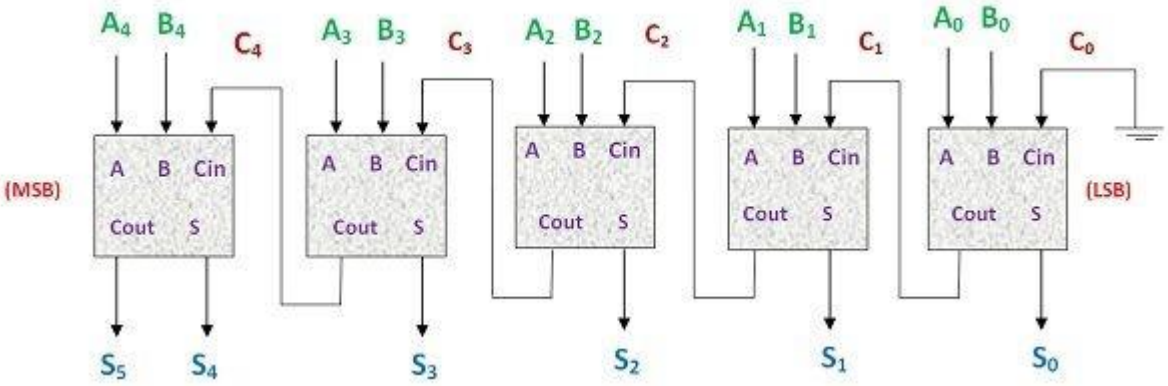


Tablo 3: Full-Adder Devre Şeması

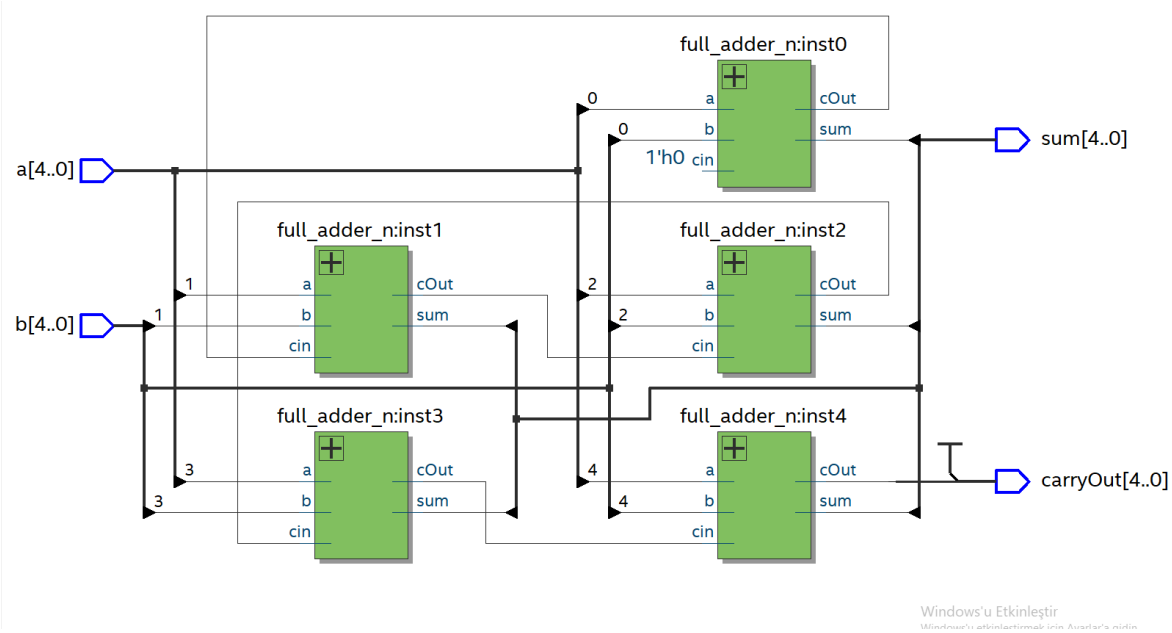
$$Sum = A\ XOR\ B\ XOR\ C$$

$$C_{OUT} = AB + AC_{IN} + BC_{IN}$$

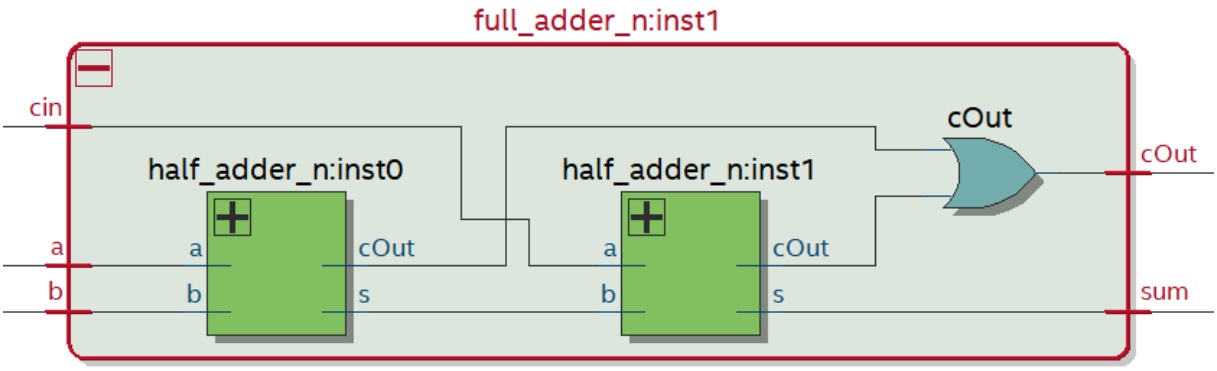
Oluşturulan Full-Adder devresinin 5 kopyası kullanılarak her birinin Cout değeri diğerinin Cin değeri birbirine bağlanmak üzere 5-bitlik bir ripple-carry-adder tasarlandı.



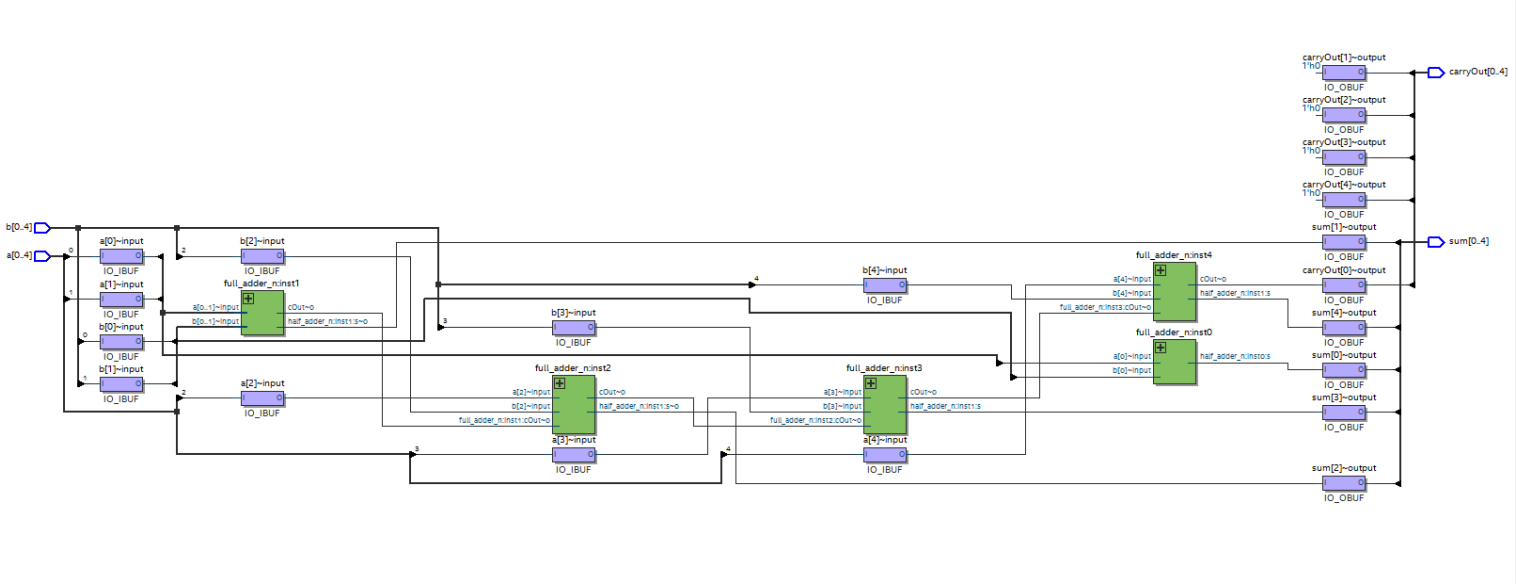
e) Ripple Carry Adder RTL Şeması



Şema 2: Ripple Carry Adder RTL Şeması



f) Eşleştirme Sonrası Devre Şeması



Şema 3:RippleCarryAdder Eşleştirme Sonrası Devre Şeması

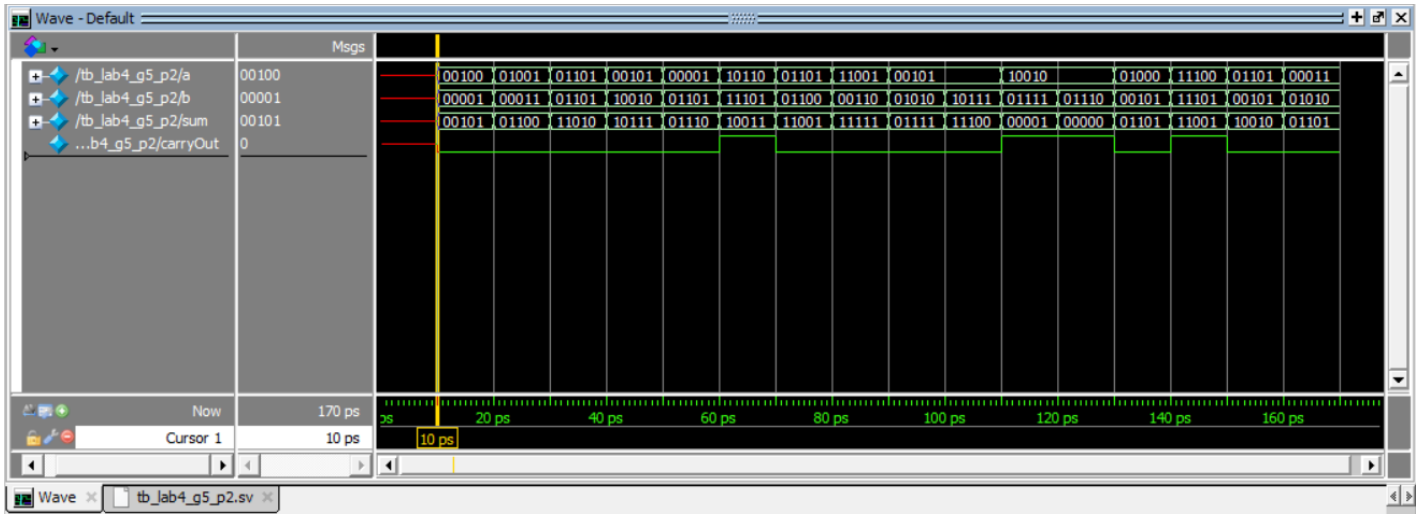
g) Analiz ve Sentez Kaynak Kullanım Özeti

Analysis & Synthesis Resource Usage Summary

<<Filter>>

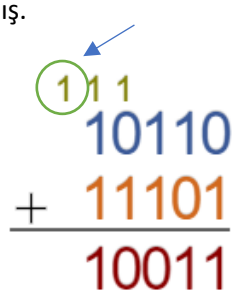
	Resource	Usage
1	Estimated Total logic elements	9
2		
3	Total combinational functions	9
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	2
2	-- 3 input functions	6
3	-- <=2 input functions	1
5		
6	Logic elements by mode	
1	-- normal mode	9
2	-- arithmetic mode	0
7		
8	Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0
9		
10	I/O pins	20
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	a[0...put
15	Maximum fan-out	3
16	Total fan-out	54
17	Average fan-out	1.10

h) Simülasyon



Teyit etmek amaçlı örneğin simülasyondaki 60 ps, 70 ps arasındaki random atanan değeri yorumlayalım:

$A = 10110 \quad B = 11101 \quad A + B = 10011$ şeklinde toplanmış ve carryOut değeri yani toplamdan gelen elde değeri de 1 olarak hesaplanmış.



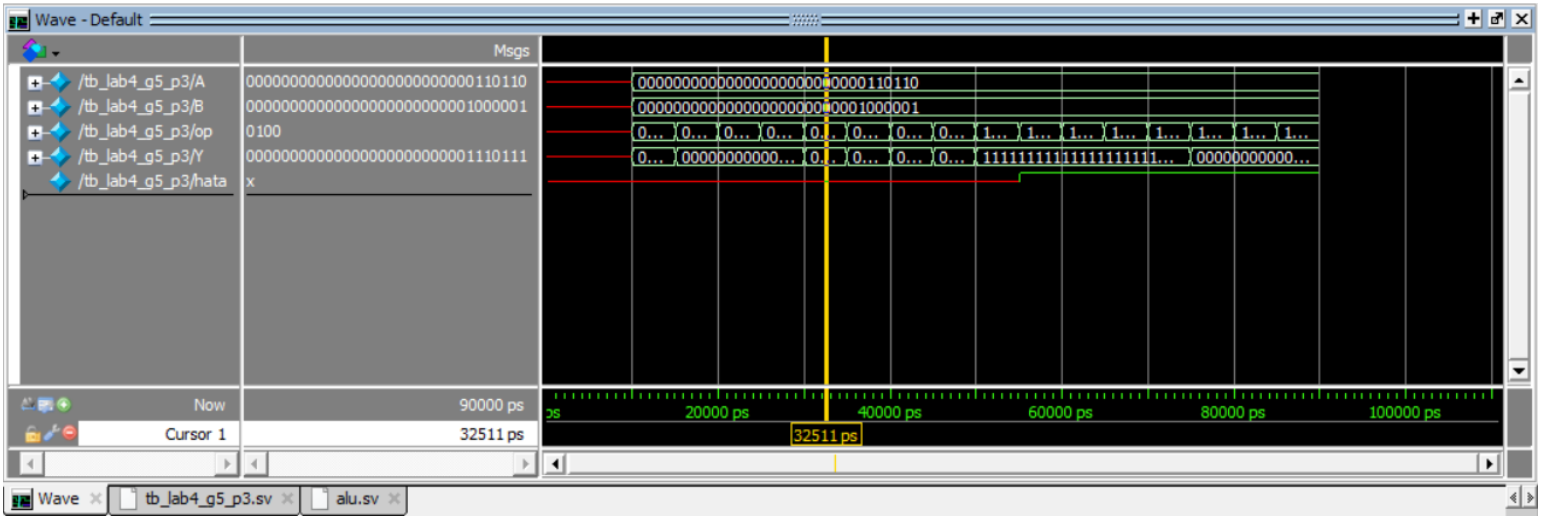
2.2. Problem 3 - 32-bitlik Bir Aritmetik Lojik Birimin (ALU) Tasarımı

2.2.1. Teorik Araştırma

2.2.1.1. ALU Nedir?

ALU, aritmetiksel ve mantıksal işlemleri gerçekleştiren bir donanım birimidir. ALU'nun temel amacı, matematiksel işlemler ve mantıksal operasyonlar gibi temel hesaplamaları gerçekleştirmektir. Aritmetik işlemler, toplama, çıkarma, çarpma ve bölme gibi matematiksel işlemleri içerirken, mantıksal işlemler, AND, OR, XOR gibi mantıksal operasyonları içerir

2.2.2. Simülasyon



EKLER 1. Problem 2

Half Adder

```
module half_adder_n (
    input logic a,b,
    output logic s,cOut
);

assign s = a ^ b;
assign cOut = a & b;

endmodule
```

Full Adder

```
//Full adder using half adder
module full_adder_n(
     input logic a,b,cin,
     output logic sum,cOut
);

logic cOut2,c1,s;

half_adder_n inst0(a,b,s,cOut2);
half_adder_n inst1(cin,s,sum,c1);

assign cOut = cOut2 | c1 ;

endmodule
```

Ripple Carry Adder

```
module ripple_carry_adder_5(
      input logic [4:0] a, b,
      output logic [4:0] sum, carryOut
);

logic cOut0,cOut1,cOut2,cOut3,cOut4,s0,s1,s2,s3,s4;

full_adder_n inst0(a[0],b[0],0,sum[0],cOut0);

full_adder_n inst1(a[1],b[1],cOut0,sum[1],cOut1);

full_adder_n inst2(a[2],b[2],cOut1,sum[2],cOut2);

full_adder_n inst3(a[3],b[3],cOut2,sum[3],cOut3);

full_adder_n inst4(a[4],b[4],cOut3,sum[4],carryOut);

endmodule
```

Ripple Carry Adder Testbench

```
module tb_lab4_g5_p2();
      logic [4:0] a;
      logic [4:0] b;
      logic [4:0] sum;
      logic carryOut;

ripple_carry_adder_5 dut0(a,b,sum,carryOut);
```

```
initial begin

      #10;
      repeat(16) begin
      a = $random;
      b = $random;
      #10;
      end
      $stop;
end
endmodule
```

EKLER 2. Problem 3

```
module alu (
   input  logic [31:0] A,
   input  logic [31:0] B,
   input  logic [3:0]   op,
   output logic [31:0] Y,
   output logic hata
);

   always_comb begin
      case(op)
         0:    Y = A + B;
         1:   Y = A << B;
         2:    Y = A > B ? 1:0;
         3:    Y = A > B ? 1:0;
         4:    Y = A ^ B;
         5:    Y = A >> B;
```

```
         6:    Y = A | B;
         7:    Y = A & B;
       8:   Y = A - B;
      9:    hata = 1;
      10:   hata=1;
      11:   hata =1;
      12:   hata = 1;
      13:   Y = A<<<B;
      14:   hata = 1;
      15:   hata = 1;
       endcase
   end

endmodule
```

Problem 3 Testbench

```
`timescale 1ns / 1ps
module tb_lab4_g5_p3();

logic [31:0] A;
logic [31:0] B;
logic [3:0]   op;
logic [31:0] Y;
logic hata;



alu dut0(.A(A),.B(B),.op(op),.Y(Y),.hata(hata));

initial begin
   #10;
```

```
    A = 54;
    B = 65;
    for (int i = 0; i < 16; i++) begin
        op = i;
        #5;
    end
end
endmodule
```