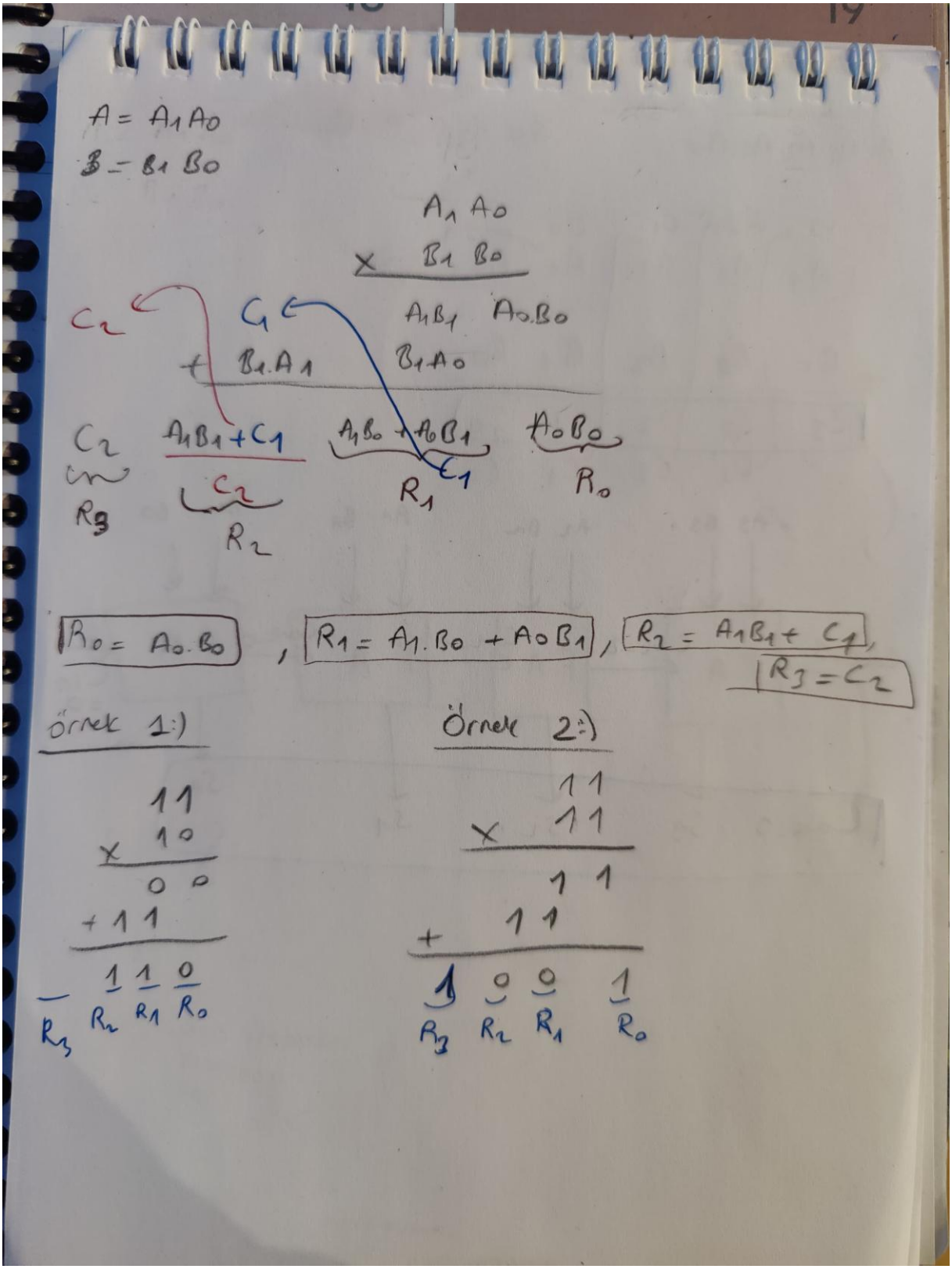
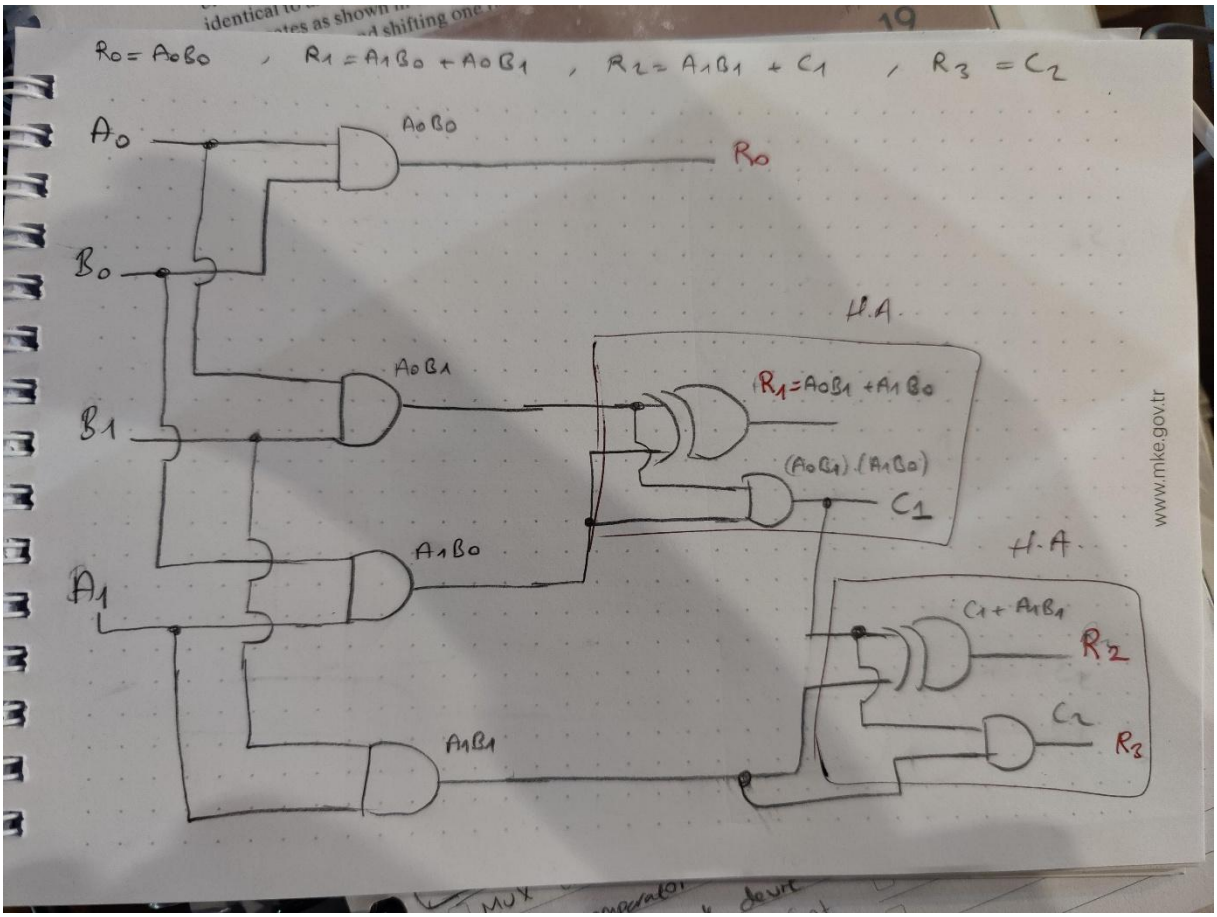
**ÖRNEK 1:** Half adder kullanarak 2-bitlik bir çarpıcı tasarlayınız ve tüm olası çarpımları gösteriniz.

Öncelikle 2-bitlik bir çarpıcı tasarlayalım:



Çarpıcı tasarımımız bu şekildedir. Şimdi devre şemasını çıkaralım:



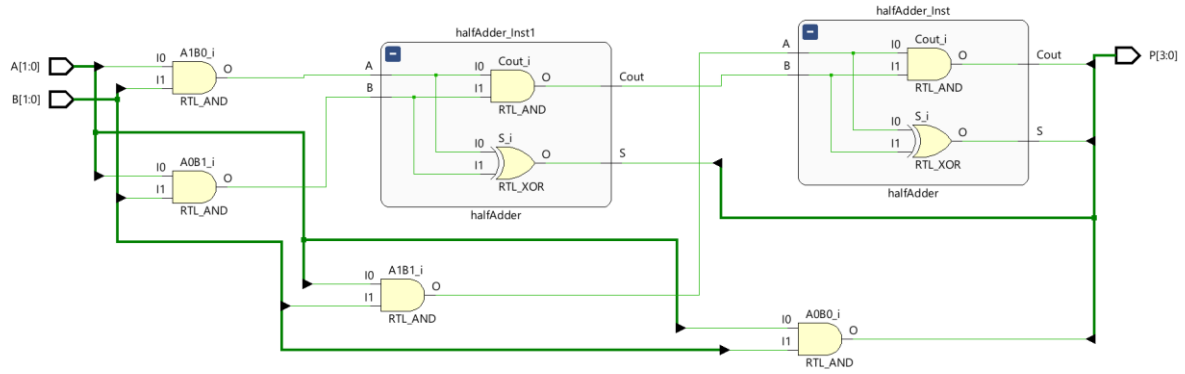

Artık design kodunu yazmaya başlayabiliriz:

```
module two_bit_multiplier(
    input logic [1:0] A,
    input logic [1:0] B,
    output logic [3:0] P
  );

  // Ara sinyaller
  logic A0B0, A0B1, A1B0, A1B1; // Ara sonuçlar
  logic R0, R1, R2, C1, C2; // Toplam ve elde bitleri

  assign A0B0 = A[0] && B[0];
  assign A0B1 = A[0] && B[1];
  assign A1B0 = A[1] && B[0];
  assign A1B1 = A[1] && B[1];

  assign R0 = A0B0;

  // R1 = A1B0 + A0B1 , ilk half adder
```

```
  halfAdder halfAdder_Inst1(
              .A(A1B0),
              .B(A0B1),
              .S(R1),
              .Cout(C1)
            );

  // R2 = A1B1 + C1 , ikinci half adder

  halfAdder halfAdder_Inst(
              .A(A1B1),
              .B(C1),
              .S(R2),
              .Cout(C2)
            );

  assign P[0] = R0;
  assign P[1] = R1;
  assign P[2] = R2;
  assign P[3] = C2; // R3 = C2



endmodule
```

(NOT : Half adder kullanacağımız için önceden yazdığımız half adder dosyasını projemizin içine include etmemiz gerekir. Bunun için yine "**sources -> + -> design sources -> add or Create design sources -> add files**" diyerek önceden hazırlamış olduğunuz half adder dosyanızı seçebilirsiniz ya da aynı proje içerisinde yeni bir half adder dosyası oluşturabilirsiniz.

Design çıktımız şu şekilde olacaktır:



Şimdi de simülasyon kodumuzu yazalım:

```
module tb_two_bit_multiplier();
    logic [1:0] A;
    logic [1:0] B;
    logic [3:0] P;

    two_bit_multiplier two_bit_multiplier_Inst(.*);

    initial begin
        A = 2'b00;
        B = 2'b00;
        #10;

        for (int i = 0; i < 4; i++)
        begin
            for (int j = 0; j < 4; j++)
            begin
                A = i;
                B = j;
                #10;
                $display("A = %b . B = %b = P = %b", A, B, P);
            end
        end

        $stop;
    end
endmodule
```

Ardından kodumuzu simüle edelim ve sonuçları inceleyelim:

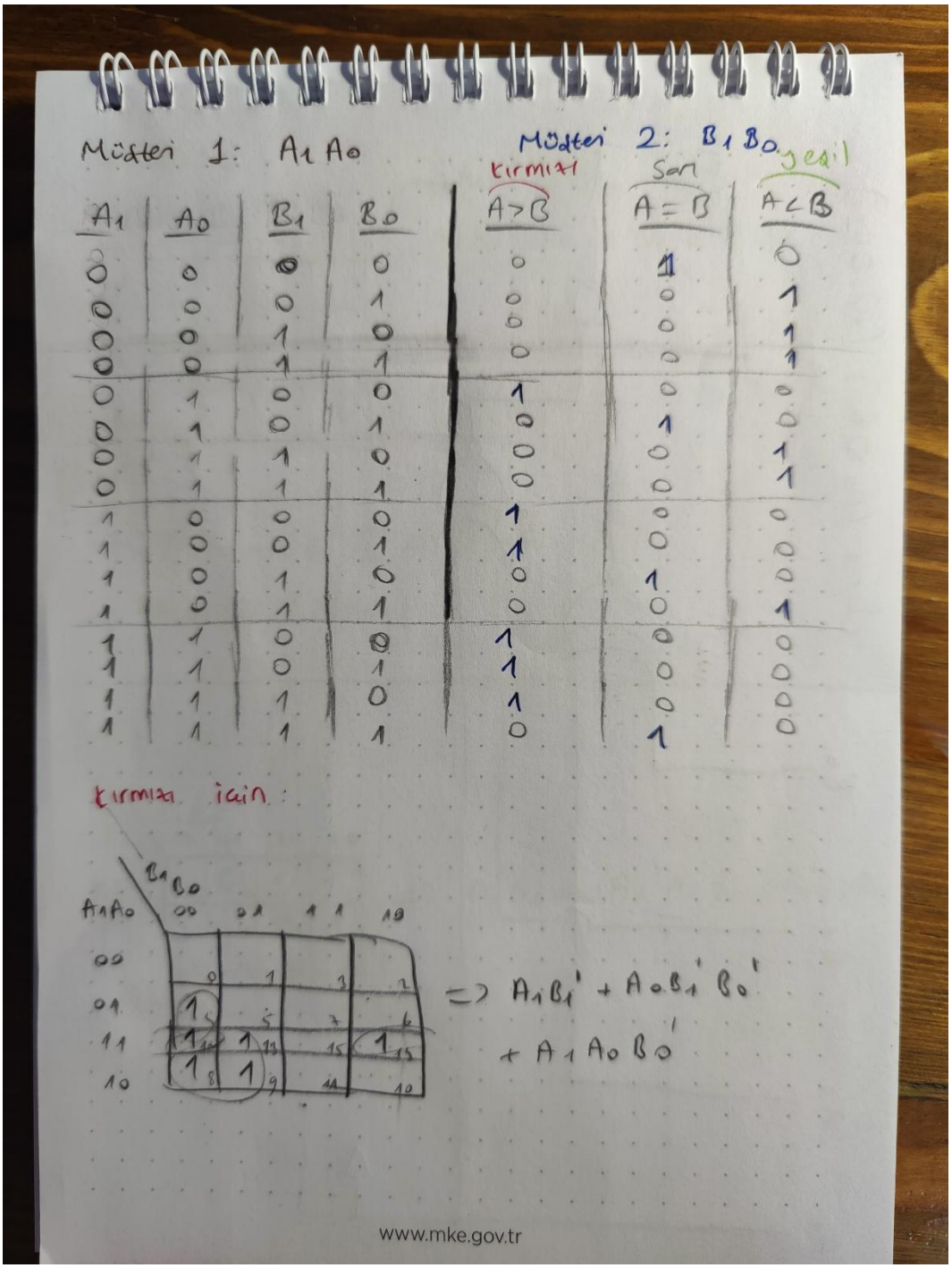

Simülasyon sonuçlarımız doğru gözüküyor. Tcl console üzerinden de sonuçlarımıza bakalım.

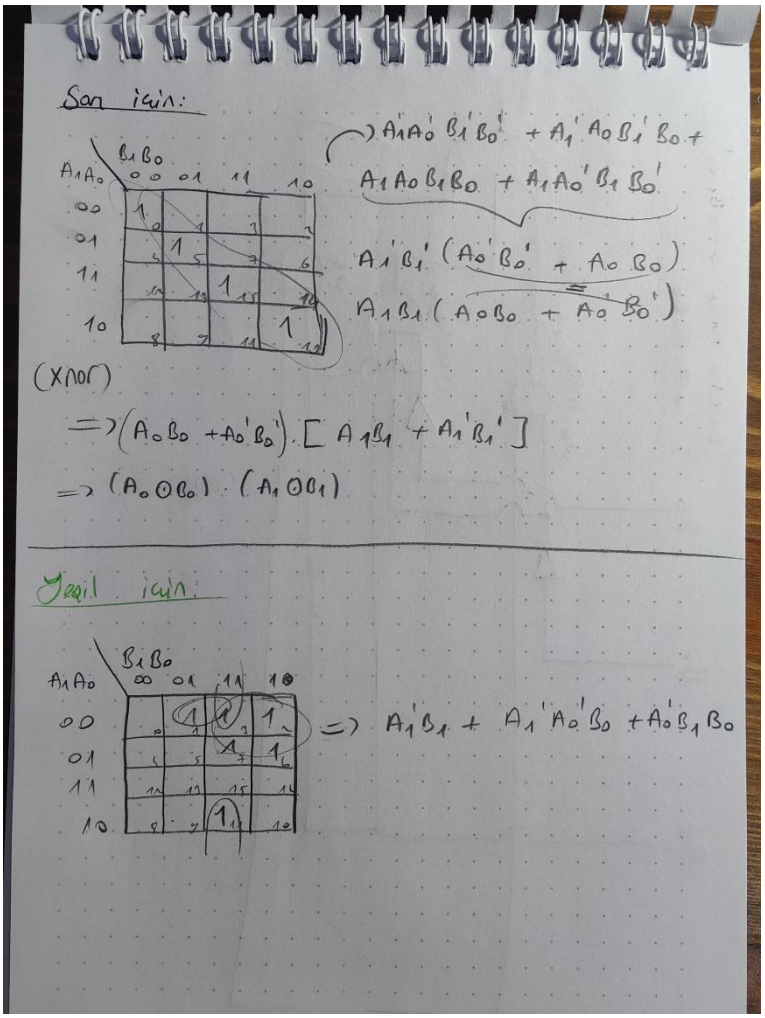
```
# run 1000ns
A = 00 . B = 00 = P = 0000
A = 00 . B = 01 = P = 0000
A = 00 . B = 10 = P = 0000
A = 00 . B = 11 = P = 0000
A = 01 . B = 00 = P = 0000
A = 01 . B = 01 = P = 0001
A = 01 . B = 10 = P = 0010
A = 01 . B = 11 = P = 0011
A = 10 . B = 00 = P = 0000
A = 10 . B = 01 = P = 0010
A = 10 . B = 10 = P = 0100
A = 10 . B = 11 = P = 0110
A = 11 . B = 00 = P = 0000
A = 11 . B = 01 = P = 0011
A = 11 . B = 10 = P = 0110
A = 11 . B = 11 = P = 1001
$stop called at time : 170 ns
```

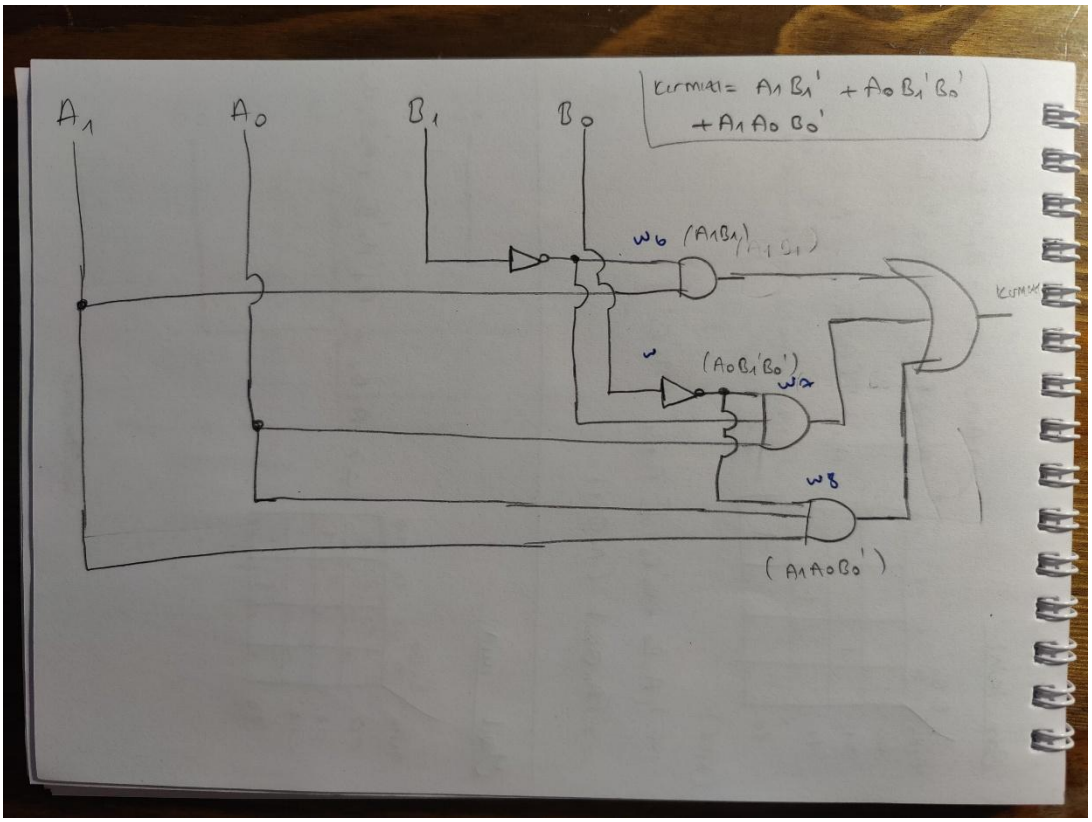
Görüldüğü gibi devremiz doğru bir şekilde çalışıyor.

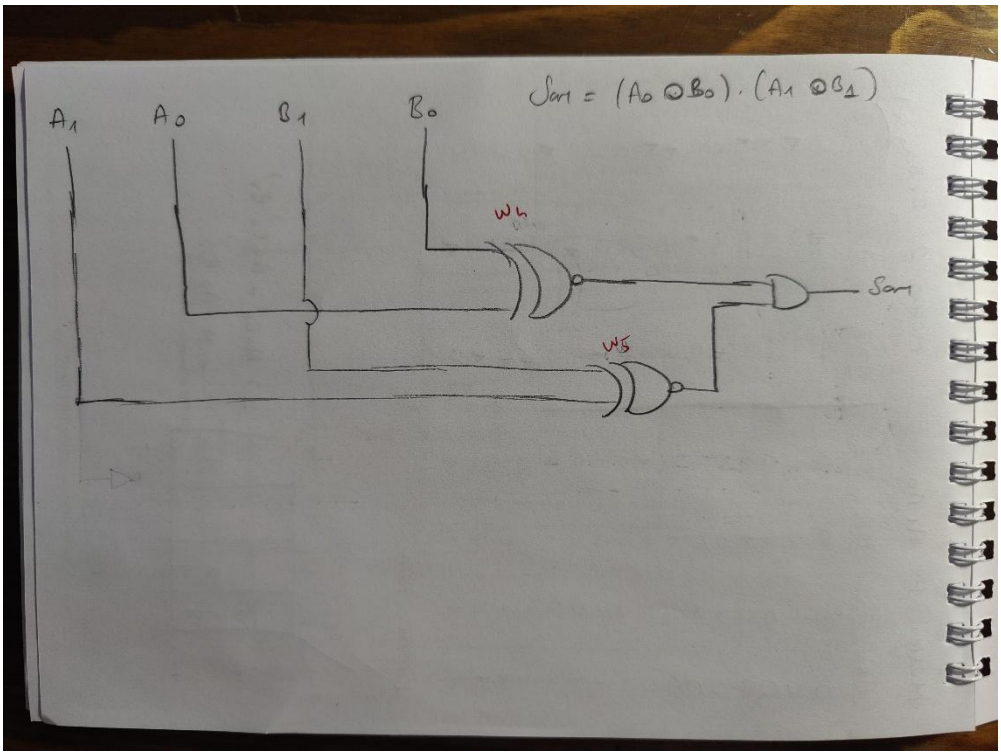
**ÖRNEK 2:** Bir markette en fazla 3 ürünün ödemesi yapılabilen bir “jet kasa” olsun. Müşteriler, kasayı kullanmadan önce ellerindeki ürün adedini gerekli ekrana girmek zorundalar. Sıra olması takdirinde, jet kasa en az ürüne sahip olan müşteriye öncelik verecek şekilde programlanmıştır. Aynı anda sıraya giren iki müşteri ürün adetlerini makineye girmişlerdir. Eğer “Müşteri 1” in ürün sayısı daha az ise jet kasada yeşil ışık, “Müşteri 2”nin daha az ise kırmızı ışık yanacaktır. Eşit olduğu takdirde sarı ışık gösterilecektir. Bunu gerçekleştiren bir devre tasarımı yapınız.

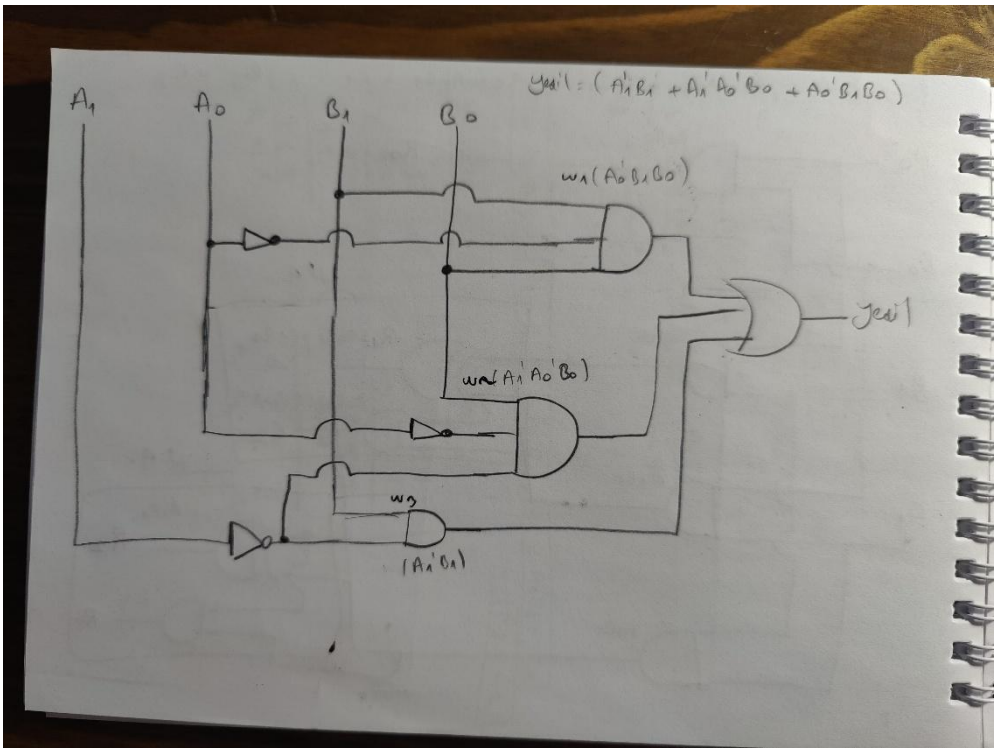
Sorudan da anlaşılacağı üzere bu soru bit “2 – bit karşılaştırıcı” sorusudur. Öncelikle 2 bit bir karşılaştırıcı tasarımı yapalım:




Son için:
A1A0 B1'B0' + A1' A0 B1' B0 +
A1 A0 B1 B0 + A1 A0' B1 B0'
A1'B1' (A0'B0' + A0 B0)
A1 B1 (A0 B0 + A0' B0')
(XNOR)
=> (A0B0 + A0'B0') . [A1B1 + A1'B1']
=> (A0 ⊙ B0) . (A1 ⊙ B1)
Yesil için:
=> A1'B1 + A1'A0'B0 + A0'B1 B0


A1
A0
B1
B0
KIRMIZI = A1 B1' + A0 B1' B0'
+ A1 A0 B0'
w6 (A1B1)
(A0B1'B0')
w8
(A1A0B0')



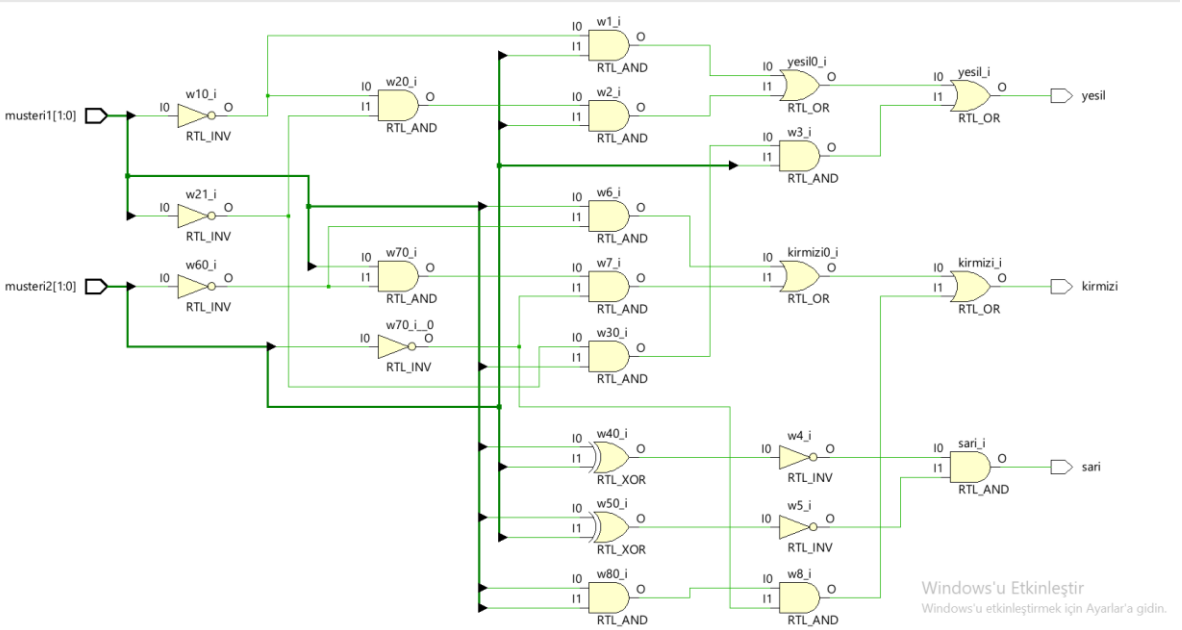




Tasarımımız böyledir. Şimdi de design kodumuzu yazalım:

```
module two_bit_comparator(
    input logic [1:0] musteri1, // A1A0
    input logic [1:0] musteri2, // B1B0
    output logic kirmizi,
    output logic yesil,
    output logic sari
    );
```

```
    //Ara sinyaller
    logic w1, w2, w3; //müşteri 1 < müşteri 2 / yeşil
    logic w4, w5; //müşteri 1 = müşteri 2 / sarı
    logic w6, w7, w8; //müşteri 1 > müşteri 2 / kırmızı

    //Kırmızı
    assign w6 = (musteri1[1] && !musteri2[1]);
    assign w7 = (musteri1[0] && !musteri2[1] && !musteri2[0]);
    assign w8 = (musteri1[1] && musteri1[0] && !musteri2[0]);
    assign kirmizi = w6 || w7 || w8;

    // Sarı
    assign w4 = !(musteri1[0] ^ musteri2[0]);
    assign w5 = !(musteri1[1] ^ musteri2[1]);
    assign sari = w4 && w5;

    //Yeşil
    assign w1 = (!musteri1[1] && musteri2[1]);
    assign w2 = (!musteri1[1] && !musteri1[0] && musteri2[0]);
    assign w3 = (!musteri1[0] && musteri1[1] && musteri2[0]);
    assign yesil = w1 || w2 || w3;
endmodule
```

Design çıktımız şu şekil olacaktır:

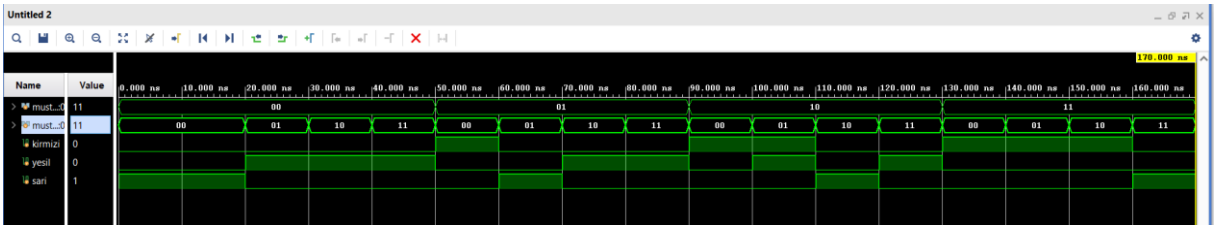



(Not: RTL analizde en fazla iki girişli kapılar kullanıldığı için fonksiyonumuz bu şekilde modellenmiştir.)

Şimdi de testbanch kodunu yazalım:

```
module tb_two_bit_comparator();
logic [1:0] musteri1;
logic [1:0] musteri2;
logic kirmizi;
logic yesil;
logic sari;


two_bit_comparator two_bit_comparator_Inst(.*);

initial begin
    musteri1 = 2'b00;
    musteri2 = 2'b00;
    #10;

    for (int i = 0; i < 4; i++)
    begin
        for (int j = 0; j < 4; j++)
        begin
            musteri1 = i;
            musteri2 = j;
            #10;
            $display("Musteri 1: %b, Musteri 2: %b, Kirmizi: %b, Yesil: %b,
Sari: %b",musteri1, musteri2, kirmizi, yesil, sari);
        end
    end

    $stop;
end
endmodule
```

Run Simulation dediğimizde çıktımız şu şekil olacaktır:



```
# run 1000ns
Musteri 1: 00, Musteri 2: 00, Kirmizi: 0, Yesil: 0, Sari: 1
Musteri 1: 00, Musteri 2: 01, Kirmizi: 0, Yesil: 1, Sari: 0
Musteri 1: 00, Musteri 2: 10, Kirmizi: 0, Yesil: 1, Sari: 0
Musteri 1: 00, Musteri 2: 11, Kirmizi: 0, Yesil: 1, Sari: 0
Musteri 1: 01, Musteri 2: 00, Kirmizi: 1, Yesil: 0, Sari: 0
Musteri 1: 01, Musteri 2: 01, Kirmizi: 0, Yesil: 0, Sari: 1
Musteri 1: 01, Musteri 2: 10, Kirmizi: 0, Yesil: 1, Sari: 0
Musteri 1: 01, Musteri 2: 11, Kirmizi: 0, Yesil: 1, Sari: 0
Musteri 1: 10, Musteri 2: 00, Kirmizi: 1, Yesil: 0, Sari: 0
Musteri 1: 10, Musteri 2: 01, Kirmizi: 1, Yesil: 1, Sari: 0
Musteri 1: 10, Musteri 2: 10, Kirmizi: 0, Yesil: 0, Sari: 1
Musteri 1: 10, Musteri 2: 11, Kirmizi: 0, Yesil: 1, Sari: 0
Musteri 1: 11, Musteri 2: 00, Kirmizi: 1, Yesil: 0, Sari: 0
Musteri 1: 11, Musteri 2: 01, Kirmizi: 1, Yesil: 0, Sari: 0
Musteri 1: 11, Musteri 2: 10, Kirmizi: 1, Yesil: 0, Sari: 0
Musteri 1: 11, Musteri 2: 11, Kirmizi: 0, Yesil: 0, Sari: 1
```

Görüldüğü gibi, kodumuz sorunsuz bir şekilde çalışmaktadır.