Half Adder – Full Adder

ÖRNEK 1: a) Sadece NAND kapıları ve b) Sadece NOR kapıları kullanarak bir half – adder tasarlayın.

Öncelikle bir half – adder doğruluk tablosu çıkaralım:

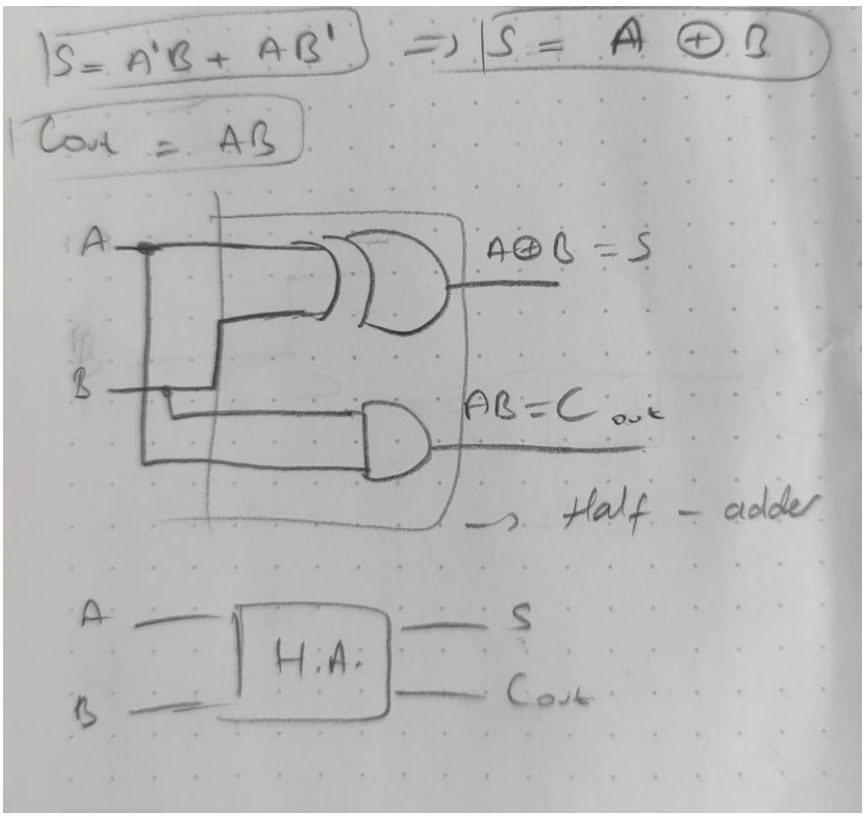
A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(NOT: Geçen hafta yaptığımız alarm – sensör örneğiyle aynı doğruluk tablosuna sahiptir. – ve buna bağlı olarak aynı devre şemasına sahip olacaktır.)

Buradan da açık bir şekilde görüldüğü üzere;

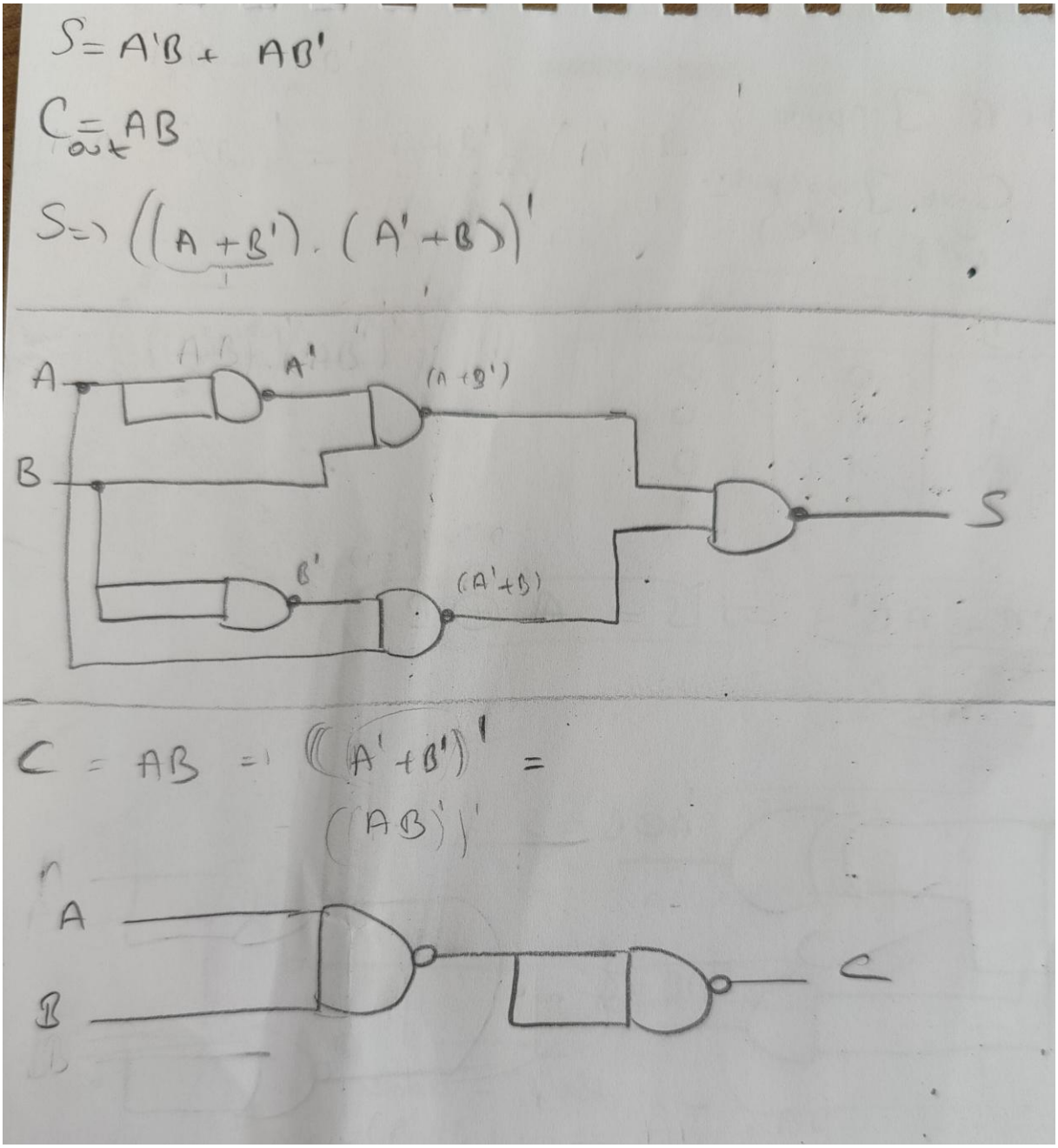
S (sum) = AB' + A'B = A xor B,
Cout (carry – elde) = AB

Devre tasarımımız şu şekilde gözükecektir:



a) Şimdi yalnız NAND kapıları kullanarak çizmeye başlayalım:

Devre şemamız aşağıdaki gibi olacaktır:

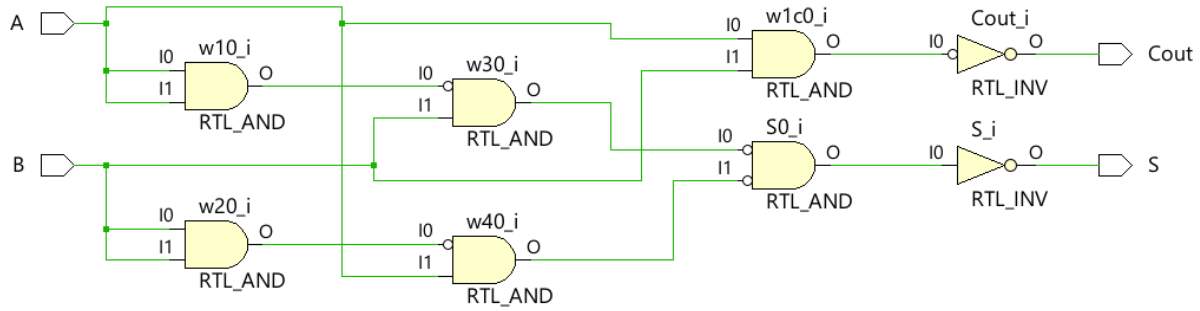


Artık design kodunu yazmaya başlayabiliriz:

```
module halfAdderNand(
    input logic A,
    input logic B,
    output logic S,
    output logic Cout
  );
  // Ara Sinyaller
  logic w1, w2, w3, w4, w1c;
  nand(w1,A,A); // A'
```

```
  nand(w2,B,B); // B'
  nand(w3,w1,B); // (A + B')
  nand(w4,w2,A); // (B + A')
  nand(S,w3,w4); // S = ((A + B') . (B + A'))'

  nand(w1c,A,B);
  nand(Cout,w1c);

endmodule
```

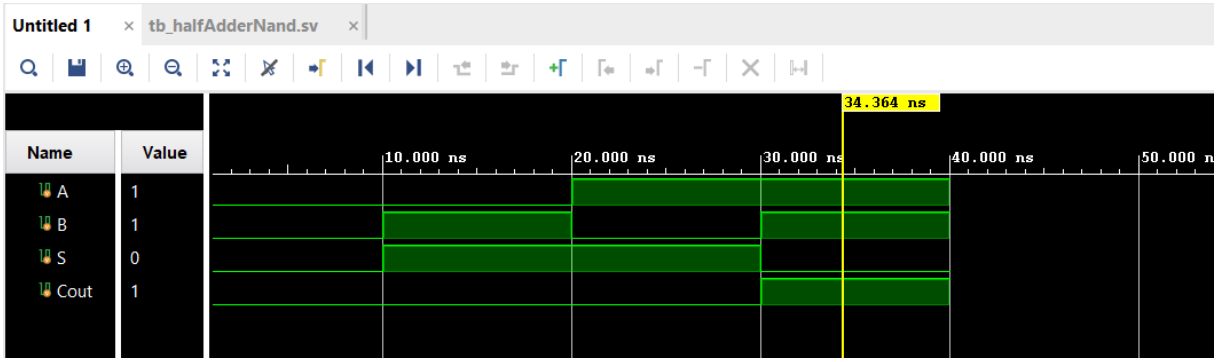
Design çıktımız şu şekil olacaktır:



Şimdi de testbanch kodumuzu yazmaya başlayalım:

```
module tb_halfAdderNand();
    logic A;
    logic B;
    logic S;
    logic Cout;

    halfAdderNand halfAdderNand_Inst(.*);
    initial begin
        A = 1'b0; B = 1'b0; #10;
        A = 1'b0; B = 1'b1; #10;
        A = 1'b1; B = 1'b0; #10;
        A = 1'b1; B = 1'b1; #10;

        $stop;
    end
endmodule
```

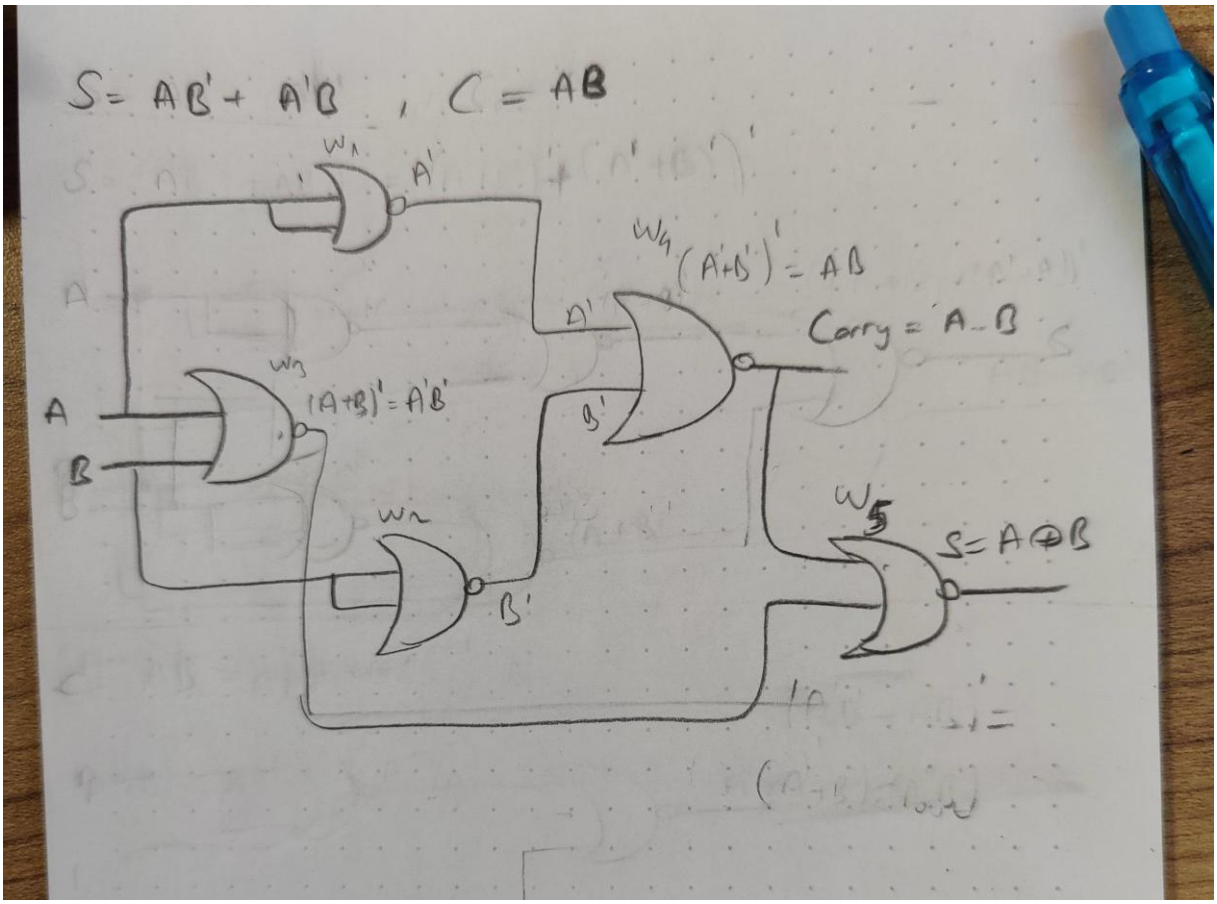
Ardından kodumuzu simüle edelim ve sonuçları inceleyelim:



Görüldüğü gibi, devremiz doğru bir şekilde çalışıyor.

b) Sorunun ikinci kısmı olan NOR kapıları ile çizime başlayalım:

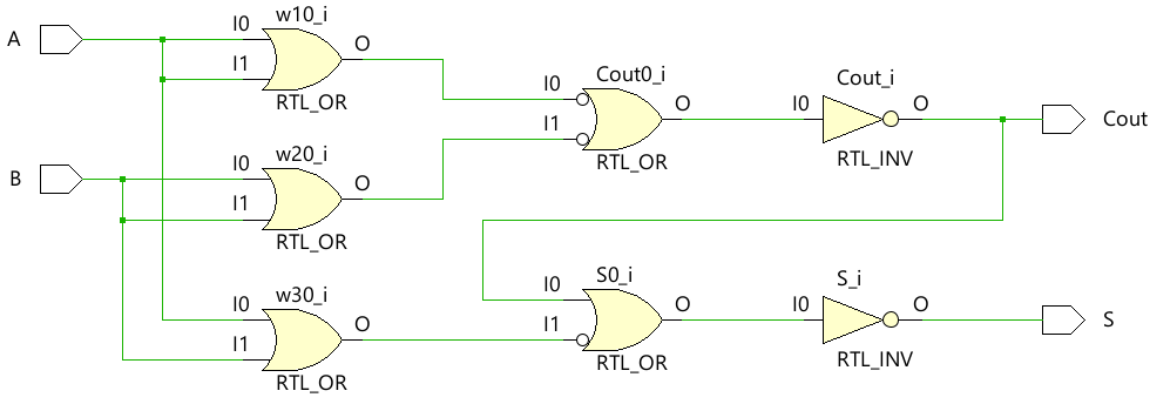
Devre şemamız aşağıdaki gibi olacaktır:



Design kodumuzu yazalım:

```
module halfAdderNor(
    input logic A,
    input logic B,
    output logic S,
    output logic Cout
  );

  // Ara sinyaller
  logic w1, w2, w3;

  //Cout için:
  nor (w1, A, A); // A'
  nor (w2, B, B); // B'
  nor (Cout, w1, w2);

  // S için:
  nor (w3, A, B);
  nor (S, Cout, w3);

endmodule
```

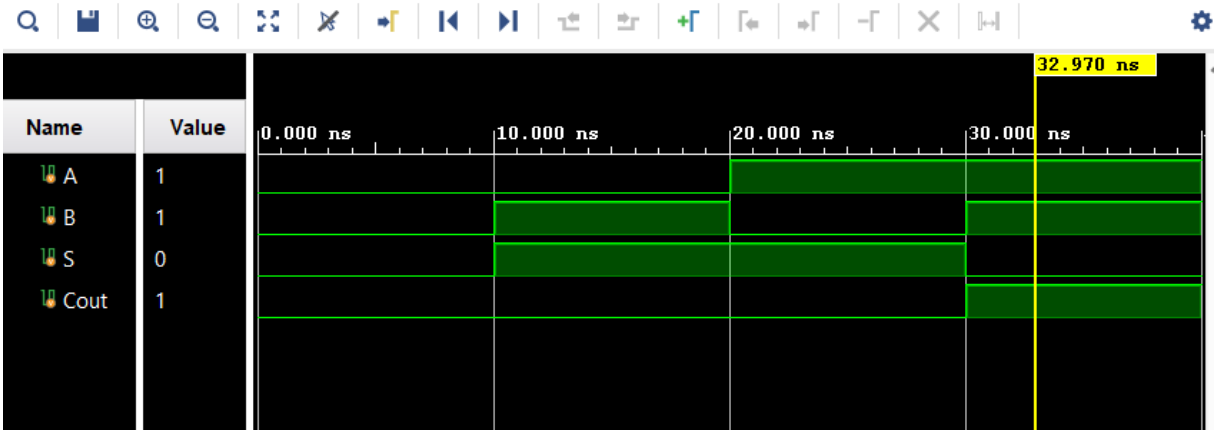
Design çıktımız aşağıdaki gibi olacaktır:



Şimdi de testbanch kodumuzu yazmaya başlayalım:

```
module tb_halfAdderNor();
    logic A;
    logic B;
    logic S;
    logic Cout;

    halfAdderNor halfAdderNor_Inst(.*);
    initial begin
        A = 1'b0; B = 1'b0; #10;
        A = 1'b0; B = 1'b1; #10;
        A = 1'b1; B = 1'b0; #10;
        A = 1'b1; B = 1'b1; #10;

        $stop;
    end
endmodule
```

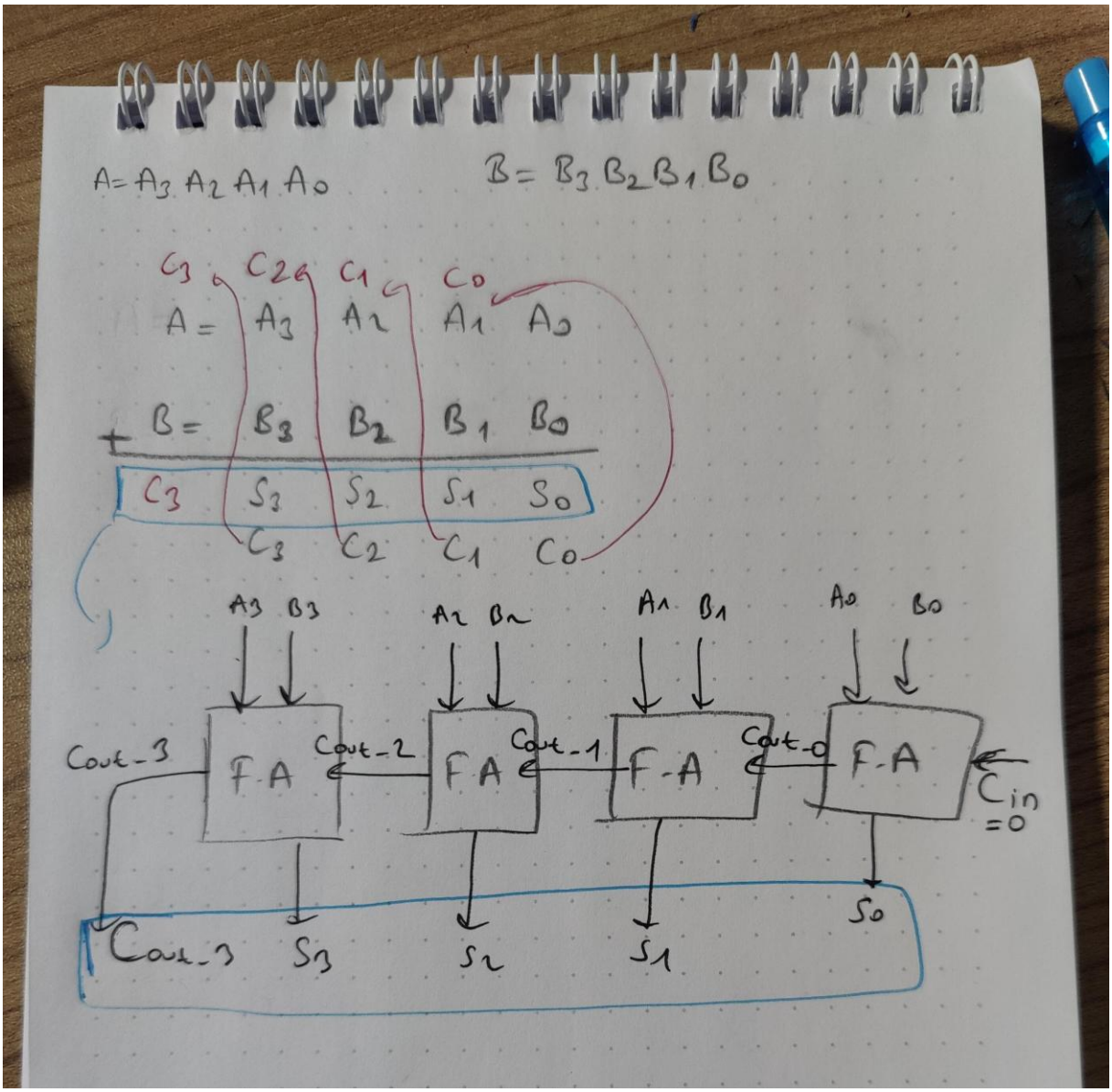
Simülasyon çıktımız da şu şekildedir:



ÖRNEK 2: Full – Adder kullanarak 4 bitlik bir toplayıcı tasarımı yapın ve aşağıdaki sayıları toplayın:

- **a)** A = 0000 , B = 0000
- **b)** A = 1010 , B = 0101
- **c)** A = 1111 , B = 1111
- **d)** A = 1001 , B = 0110

Öncelikle Full – Adder ile 4 bitlik bir toplayıcının şemasını çizelim, genel mantığını çıkaralım:



Şimdi kod aşamasına geçelim:

Öncelikle belirtmemiz gerekir ki full adder ile 4 bitlik bir toplayıcı yapacaksak yapacağımız modülde full adder'ı oluşturup çağırmamız, bunu da half adder'lar yardımıyla oluşturmamız gerekir. Yani, öncelikle çalışacağımız dosyanın içinde bir tane half adder design dosyası oluşturalım ve bu dosyamıza half adder kodumuzu yazalım:

```
module halfAdder(
    input logic A,
    input logic B,
    output logic S,
    output logic Cout
    );

    assign S = A ^ B;
    assign Cout = A && B;
endmodule
```

Half Adder kodumuzu bu şekilde yazdıktan sonra aynı projenin içinde yeni bir full adder dosyası açalım ve önceden yazmış olduğumuz half adder modülümüzü çağırarak full adder'ımızı oluşturalım:

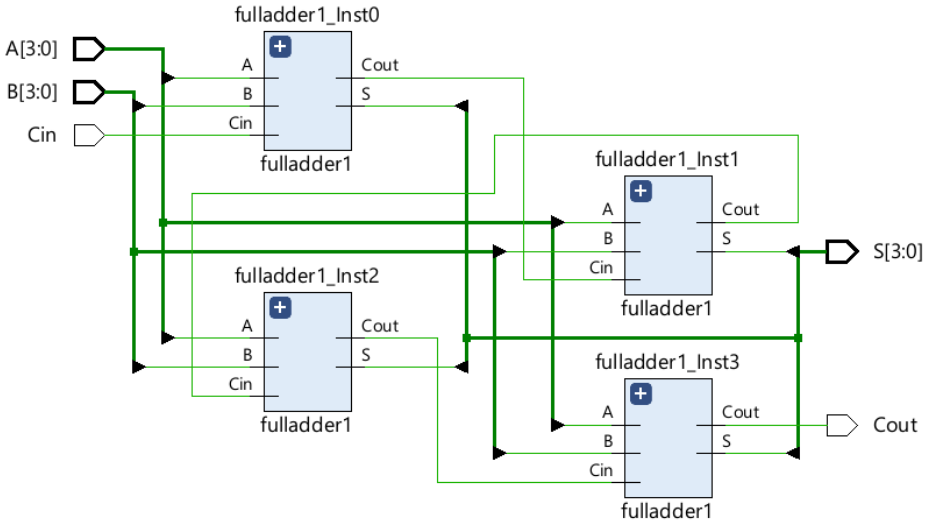
```
module fulladder1(
    input logic A,
    input logic B,
    input logic Cin,
    output logic S,
    output logic Cout
    );

    logic S1, C1, C2;

    // İlk Half Adder
    halfAdder halfAdder_Inst0 (
        .A(A),
        .B(B),
        .S(S1),
        .Cout(C1)
    );

    // İkinci Half Adder
    halfAdder halfAdder_Inst1 (
        .A(S1),
        .B(Cin),
        .S(S),
        .Cout(C2)
    );

    // Toplam elde
    assign Cout = C1 || C2; // Carry-out = Carry1 OR Carry2
endmodule
```

Artık Full adder'ımızı da oluşturduğumuza göre 4 bitlik toplayıcımızı yazmaya başlayabiliriz! Design kodumuzu oluşturalım:

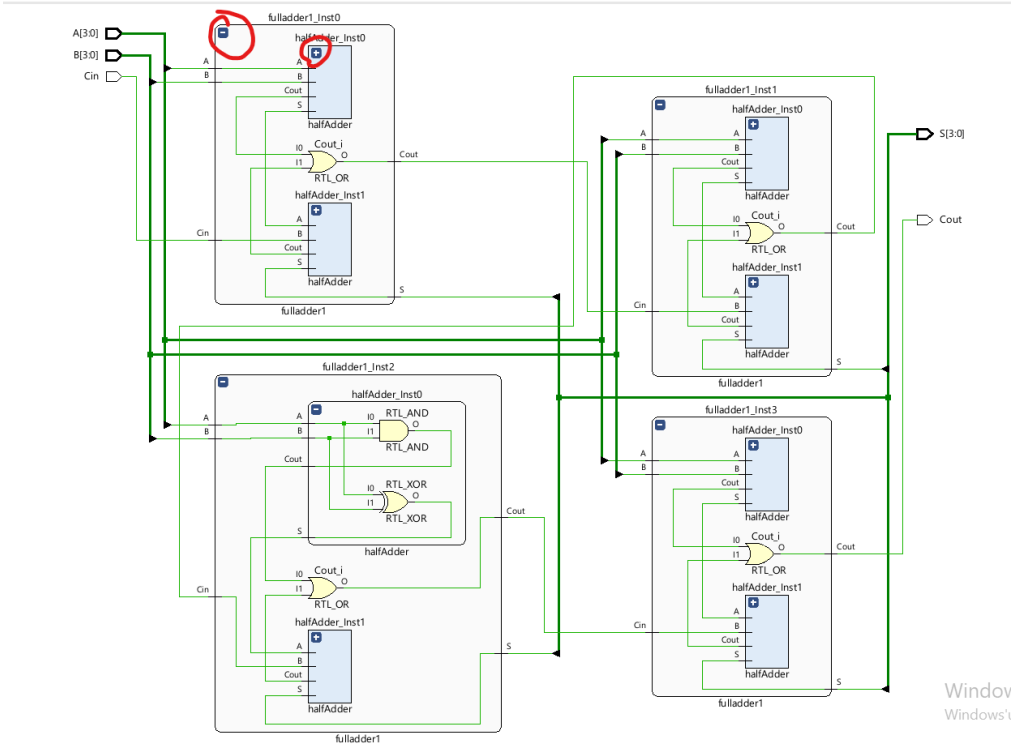
```
module fourBitFullAdder(
    input logic [3:0] A,
    input logic [3:0] B,
    input logic Cin, // Elde girişi
    output logic [3:0] S,
    output logic Cout // final elde çıkışı
  );

  /* bugüne kadar hep 1 bitlik ifadelerle çalıştığımız için [x:0] şeklinde bit
sayısını ifade eden gösterim kullanmadık.
  Çok bitli ifadelerle çalışmak istiyorsak [x:0] şeklinde bit sayısını ifade
eden gösterim kullanmamız gerekmektedir.
  */

  logic [3:0] C; // Ara elde sinyali

  // LSB
  fulladder1 fulladder1_Inst0
            (
              .A(A[0]), //A bitinin LSB bitini çağırdık.
              .B(B[0]),
              .Cin(Cin),
              .S(S[0]),
              .Cout(C[0])
            );

  fulladder1 fulladder1_Inst1
            (
              .A(A[1]),
              .B(B[1]),
              .Cin(C[0]),
              .S(S[1]),
              .Cout(C[1])
            );

  fulladder1 fulladder1_Inst2
            (
              .A(A[2]),
              .B(B[2]),
              .Cin(C[1]),
              .S(S[2]),
              .Cout(C[2])
            );

  //MSB
  fulladder1 fulladder1_Inst3
            (
              .A(A[3]),
              .B(B[3]),
```

```
            .Cin(C[2]),
            .S(S[3]),
            .Cout(Cout)
          );


endmodule
```

Design çıktımız aşağıdaki şekilde olacaktır:



Portların üzerindeki + / - ifadelerine basarak yapıları açık bir şekilde görebilirsiniz:

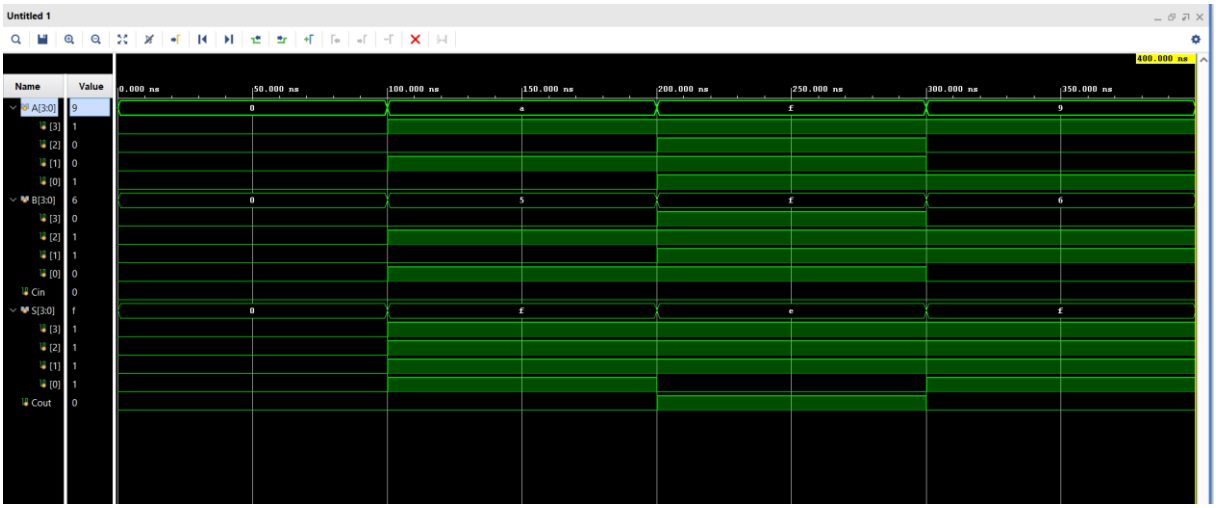


Şimdi de simülasyon kodumuzu yazalım:

```
module tb_fourBitFullAdder();
    logic [3:0] A;
    logic [3:0] B;
    logic Cin;
    logic [3:0] S;
    logic Cout;

    fourBitFullAdder fourBitFullAdder_Inst(.*);

    initial begin

        A = 4'b0000;
        B = 4'b0000;
        Cin = 1'b0;

        /*
        Soruda istenilen toplam ifadeleri:
        a)  A = 0000 , B = 0000
        b)  A = 1010 , B = 0101
        c)  A = 1111  , B = 1111
        d)  A = 1001 , B = 0110
        */

        A = 4'b0000;
        B = 4'b0000;
        Cin = 1'b0;
        #100;
        $display("A = %b, B = %b, Cin = %b, S = %b, Cout = %b", A, B, Cin, S,
Cout);
        // display ifadesi ile Tcl console aracılığıyla istenen çıktıları
ekrana yazdırma işlemi gerçekleştirilir.

        A = 4'b1010;
        B = 4'b0101;
        Cin = 1'b0;
        #100;
        $display("A = %b, B = %b, Cin = %b, S = %b, Cout = %b", A, B, Cin, S,
Cout);

        A = 4'b1111;
        B = 4'b1111;
        Cin = 1'b0;
        #100;
        $display("A = %b, B = %b, Cin = %b, S = %b, Cout = %b", A, B, Cin, S,
Cout);

        A = 4'b1001;
```

```
        B = 4'b0110;
        Cin = 1'b0;
        #100;
        $display("A = %b, B = %b, Cin = %b, S = %b, Cout = %b", A, B, Cin, S,
Cout);

        $stop;
    end
endmodule
```

Simülasyon çıktılarımız şu şekilde olacaktır:




```
# run 1000ns
A = 0000, B = 0000, Cin = 0, S = 0000, Cout = 0
A = 1010, B = 0101, Cin = 0, S = 1111, Cout = 0
A = 1111, B = 1111, Cin = 0, S = 1110, Cout = 1
A = 1001, B = 0110, Cin = 0, S = 1111, Cout = 0
$stop called at time : 400 ns : File "C:/Users/zulal/OneDrive/Desktop/lc
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_fourBitFullAdder
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```