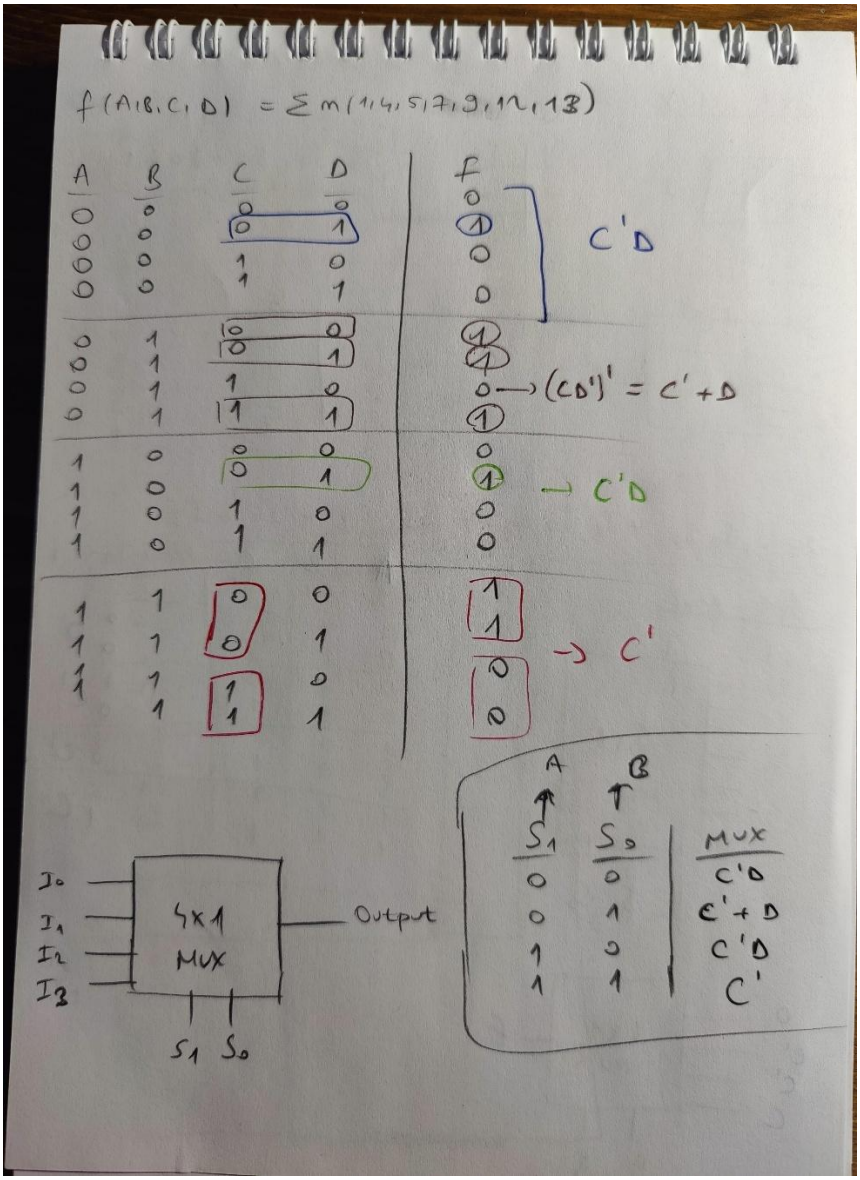
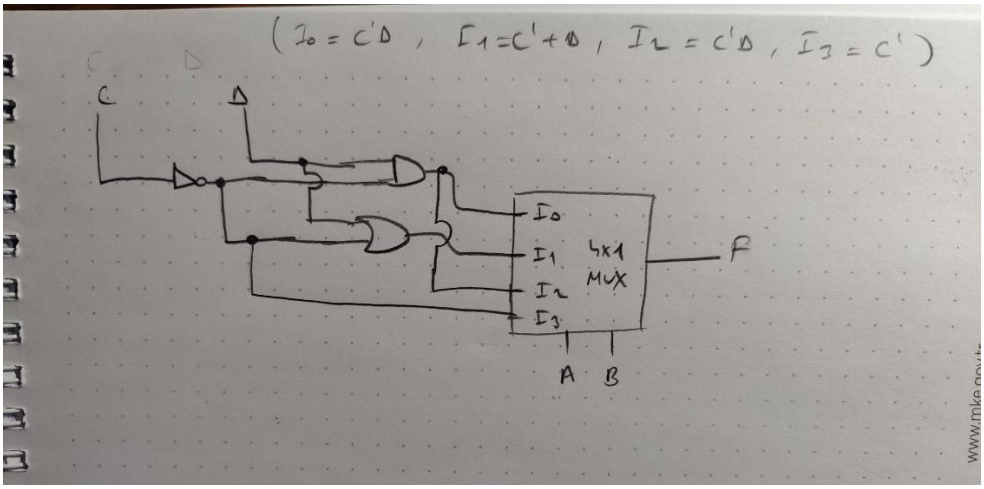
ÖRNEK 1: MUX kullanarak f(A, B, C, D) = M(1, 4, 5, 7, 9, 12, 13) devre tasarımını yapınız.

Önce doğruluk tablomuzu çıkaralım ve devremizi tasarlayalım:



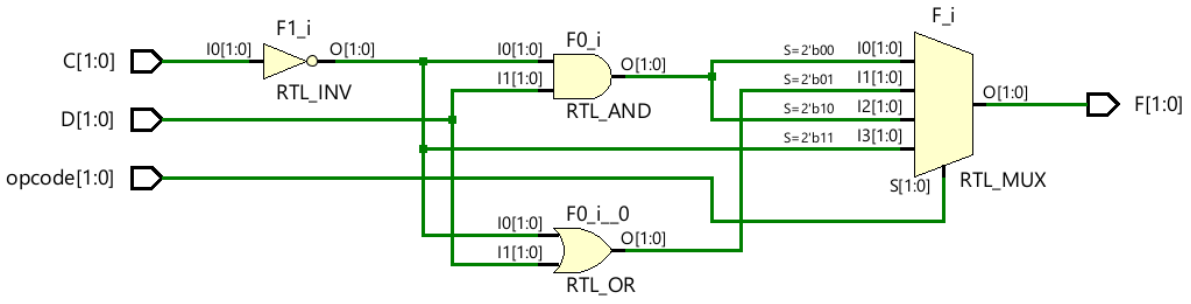


Şimdi design kodumuzu tasarlamaya başlayabiliriz:

```
module devre_tasarimi_mux
  #(
     parameter N = 2
   )
   (
     input logic [N-1:0] C,
     input logic [N-1:0] D,
     input logic [1:0] opcode,
     output logic [N-1:0] F

   );

    always_comb
    begin : mux
        case(opcode)
            0: F = ~C & D;
            1: F = ~C | D;
            2: F = ~C & D;
            3: F = ~C;
            default: F = '0;
        endcase
    end
endmodule
```

NOT : Fark edilebileceği üzere, ALU örneğimizdeki gibi yine bir case yapısı kullandık. Bir sistemde MUX oluşturmak istersek case yapısı yardımıyla oluşturabiliriz.

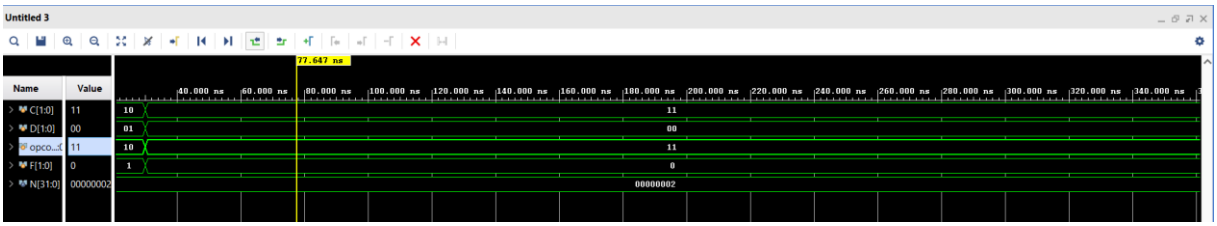
Design çıktımız şu şekilde olacaktır:



Şimdi de testbanch kodunu yazmaya başlayabiliriz:

```
module tb_devre_tasarimi_mux
#(
   parameter N = 2
 );

logic [N-1:0] C;
logic [N-1:0] D;
logic [1:0] opcode;
logic [N-1:0] F;

devre_tasarimi_mux
  #(
    .N(N)
  )
  devre_tasarimi_mux_Inst(.*);

initial begin
  for (int i = 0; i < 4; i++) begin // opcode için döngü (00, 01, 10, 11)
    opcode = i;
    case (opcode)
      2'b00: begin
        C = 2'b00; // I0
        D = 2'b11;
      end
      2'b01: begin
        C = 2'b01; // I1
        D = 2'b10;
      end
      2'b10: begin
        C = 2'b10; // I2
        D = 2'b01;
      end
      2'b11: begin
        C = 2'b11; // I3
        D = 2'b00;
      end
    endcase
    #10;
    $display("opcode : %b, C : %b, D : %b --> F : %b", opcode, C, D, F);
  end
end
endmodule
```

Design çıktımızın sonuçları aşağıdaki gibidir:



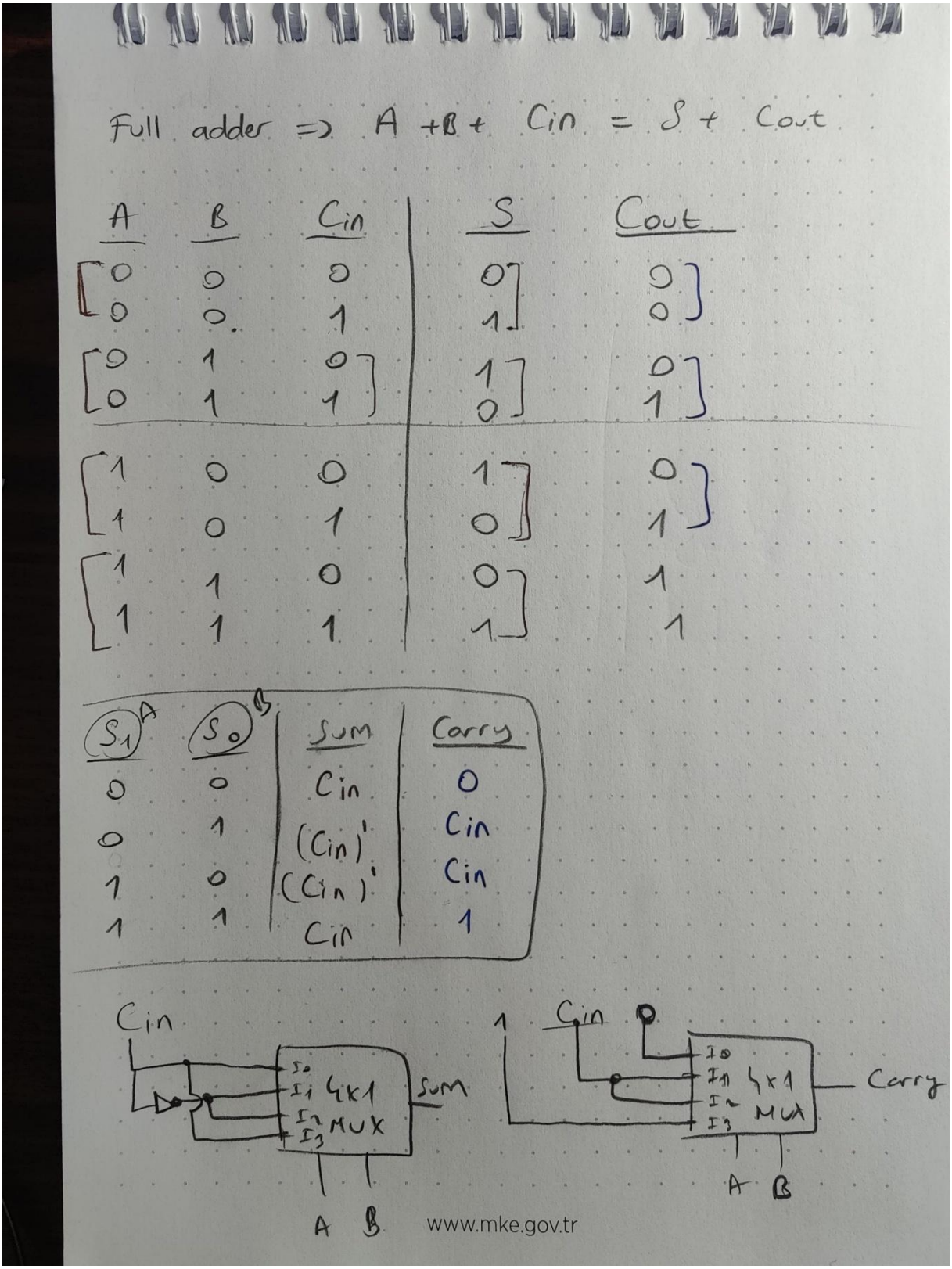

```
# run 1000ns
opcode : 00, C : 00, D : 11 --> F : 11
opcode : 01, C : 01, D : 10 --> F : 10
opcode : 10, C : 10, D : 01 --> F : 01
opcode : 11, C : 11, D : 00 --> F : 00
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Görüldüğü gibi, kodumuz sorunsuz bir şekilde çalışıyor.

(NOT: Simülasyon sonucunun dalga kısmında neden opcode'un 32 bit olarak görüldüğü kafa karıştırabilir. Bunun sebebi her işlem sırasında #10 (10 nanosaniye) bekleme süresi koyduğumuz içindir, simülasyon aracının her durumda sinyal değişimlerini göstermesidir. Ancak bu, testbench'in yanlış olduğu anlamına gelmez. Testbench her zaman 4 durum için "$display" komutuyla doğru çıktıları konsolda gösterecektir. Sonucu değiştirmez. Tcl Console çıkışı doğruysa sorun yok demektir.)

ÖRNEK 2: 4X1 MUX'lar kullanarak bir full adder tasarlayınız.

Öncelikle devremizi tasarlayalım:

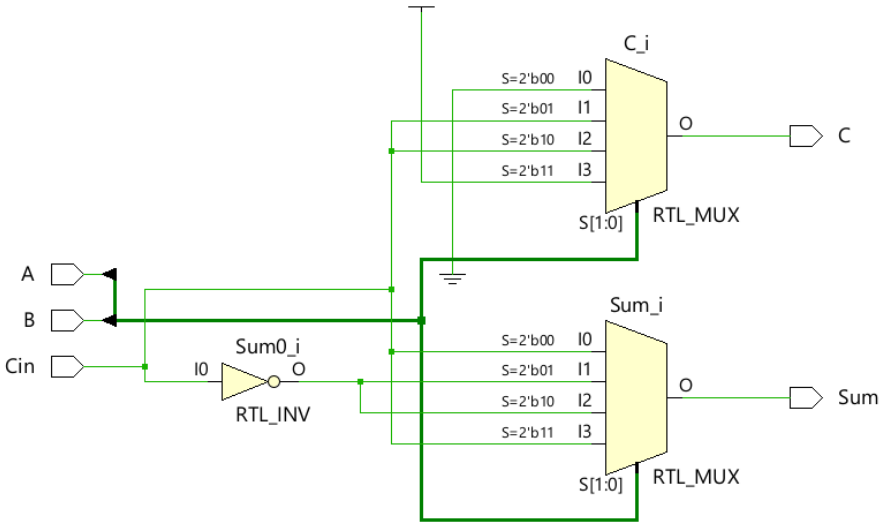


Artık design kodumuzu yazmaya başlayabiliriz:

```
module mux_fullAdder

  (
    input logic A,B,
    input logic Cin,
    output logic Sum,
    output logic C
  );

  logic [1:0] opcode;
  assign opcode = {A,B}; //opcode [1] = A, opcode [0] = B

  always_comb
  begin : toplam
     case (opcode)
        2'b00: Sum = Cin;
        2'b01: Sum = ~Cin;
        2'b10: Sum = ~Cin;
        2'b11: Sum = Cin;
        default: Sum = '0;
     endcase
  end

  always_comb
  begin : elde
        case(opcode)
            2'b00: C = 0;
            2'b01: C = Cin;
            2'b10: C = Cin;
            2'b11: C = 1;
            default: C = 0;
        endcase
  end
endmodule
```

Burada fark edileceği üzere önceki örnekten farklı olarak opcode'u bir ara sinyal ve A,B inputlarına bağlı olarak tanımladık. Çünkü opcode'u bir "ara bilgi" olarak kullandık. Parametre yardımıyla sabit olarak tanımlayıp dışarıdan vermek yerine başka girişlerden türettik. Bu tasarım için donanım açısından daha karmaşık olabilir fakat yapıya esneklik sağlar. (çünkü ara sinyal duruma göre değişebilir.)

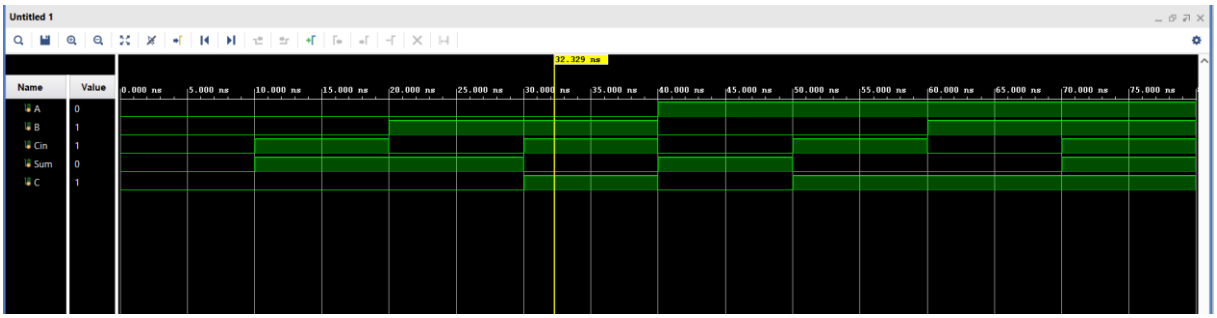
Design çıktımız şu şekilde olacaktır:



Testbanch kodunu yazalım:

```
module tb_mux_fullAdder();
    logic A;
    logic B;
    logic Cin;
    logic Sum;
    logic C;

    mux_fullAdder mux_fullAdder_Inst(.*);

    initial begin
        for(int i = 0; i < 8; i++)
        begin
            {A,B,Cin} = i; // toplu yazımlar için "{}"
            #10;
            $display("A = %b, B = %b, Cin = %b --> Sum = %b, Carry out = %b",
A, B, Cin, Sum, C);
        end

        $finish;
    end
endmodule
```

Simüle edelim ve sonucun doğru olup olmadığını kararlaştıralım:




```
# run 1000ns
A = 0, B = 0, Cin = 0 --> Sum = 0, Carry out = 0
A = 0, B = 0, Cin = 1 --> Sum = 1, Carry out = 0
A = 0, B = 1, Cin = 0 --> Sum = 1, Carry out = 0
A = 0, B = 1, Cin = 1 --> Sum = 0, Carry out = 1
A = 1, B = 0, Cin = 0 --> Sum = 1, Carry out = 0
A = 1, B = 0, Cin = 1 --> Sum = 0, Carry out = 1
A = 1, B = 1, Cin = 0 --> Sum = 0, Carry out = 1
A = 1, B = 1, Cin = 1 --> Sum = 1, Carry out = 1
$finish called at time : 80 ns : File "C:/Users/zulal/O
```

Görüldüğü üzere, kodumuz sorunsuz bir şekilde çalışmaktadır.