ÖRNEK 1: 4 girişli bir priority encoder tasarlayınız.

Tasarıma geçmeden önce priority encoder nedir bunu öğrenelim:

Bir **priority encoder (öncelikli kodlayıcı)**, girişlerin birden fazla aktif olduğu durumlarda en yüksek öncelikli aktif girişe karşılık gelen kodlanmış bir çıkış üreten dijital bir devredir.

Temel Özellikleri:

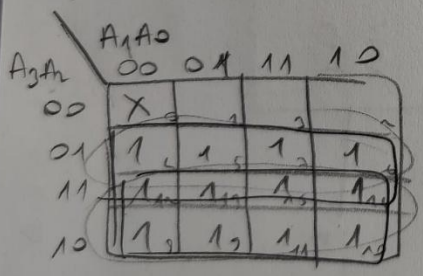
1. **Birden Fazla Aktif Giriş:** Eğer birden fazla giriş aktif (1) olursa, encoder, en yüksek öncelikli olan girişe göre çıkış üretir.
2. **Kodlanmış Çıkış:** Giriş sayısına bağlı olarak, çıkış bir **ikili (binary)** kod şeklinde olur.
3. **Ek Çıkışlar:** Bazı encoder'lar, girişlerin hiçbirinin aktif olmadığını belirtmek için bir **geçersizlik (validity)** sinyali sağlar.

Priority Encoder Nerelerde Kullanılır?

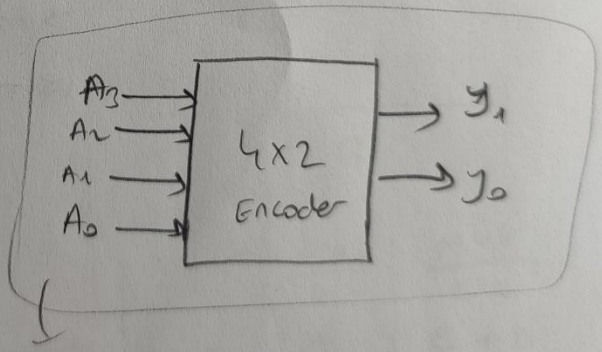
1. **İşlemciler ve Bellek Sistemleri:**
    - Kesme sinyallerinde hangi kesmenin önce işleneceğini belirler.
2. **Dijital Sistemler:**
    - Durum kontrolü ve karar mekanizmalarında.
3. **Donanım Tasarımı:**
    - Seçici devrelerde (örneğin, birden fazla kaynağın hangi birine izin verileceğini seçmek için).
4. **Hata Tespiti ve Yönetimi:**
    - Öncelikli hata algılama sistemlerinde.

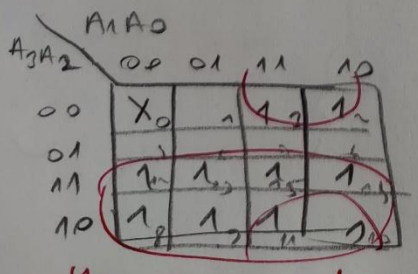
Yani, anlaşılacağı üzere bizim devremizde sayının MSB’si hangi basamaktaysa onu verecek bir sistem tasarlayacağız. İşe doğruluk tablosunu ve sadeleştirmesini çıkarmakla başlayalım:

A_3	A_2	A_1	A_0	y_1	y_0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

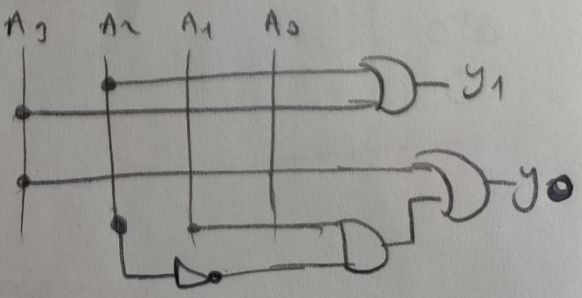


$y_1 = A_2 + A_3$





$y_0 = A_3 + A_2' A_1$

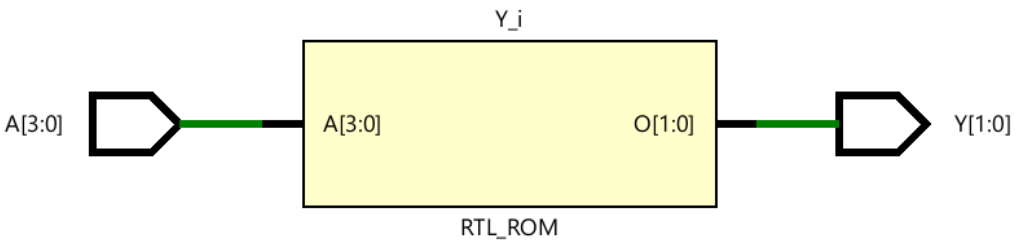


Design kodumuzu tasarlamaya başlayabiliriz:

```
module priority_encoder
#(
    parameter N = 4
)
(
    input logic [N-1:0] A,
    output logic [$clog2(N)-1:0] Y //bildiğimiz 2 tabanlı logaritmadır.
Özellikle büyük bitlerle çalışırken sayıyı hesaplamamıza gerek kalmaz.
    );

    always_comb
    begin
        priority case (A)
            4'b1000 : Y = 2'b11; // A[3]
            4'b0100 : Y = 2'b10; // A[2]
            4'b0010 : Y = 2'b01; // A[1]
            4'b0001 : Y = 2'b00; // A[0]
            default : Y = 2'b00;
        endcase
    end
endmodule
```

(NOT: priority case yapısı sadece System verilog'a özeldir. Verilog ya da başka dillerde oluşturmak için direkt priority case modülünü kullanamazsınız.)

Design çıktımız şu şekilde olacaktır:

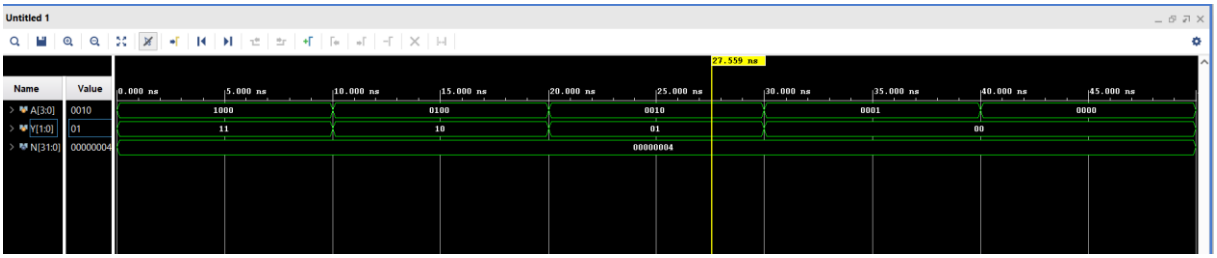


Artık simülasyona geçebiliriz:

```
module tb_prority_encoder
  #(
     parameter N = 4
   );


  logic [N-1:0] A;
```

```
  logic [$clog2(N)-1:0] Y;

    prority_encoder
    #(
        .N(N)
    )

    priority_encoder_Inst
    (
        .A(A),
        .Y(Y)
    );
  initial begin

    A = 4'b1000; #10; $display("A : %b , Y : %b", A, Y); // Y = 2'b11
    A = 4'b0100; #10; $display("A : %b , Y : %b", A, Y); // Y = 2'b10
    A = 4'b0010; #10; $display("A : %b , Y : %b", A, Y); // Y = 2'b01
    A = 4'b0001; #10; $display("A : %b , Y : %b", A, Y); // Y = 2'b00
    A = 4'b0000; #10; $display("A : %b , Y : %b", A, Y); // Y = 2'b00
(default)

    $finish;
  end

endmodule
```

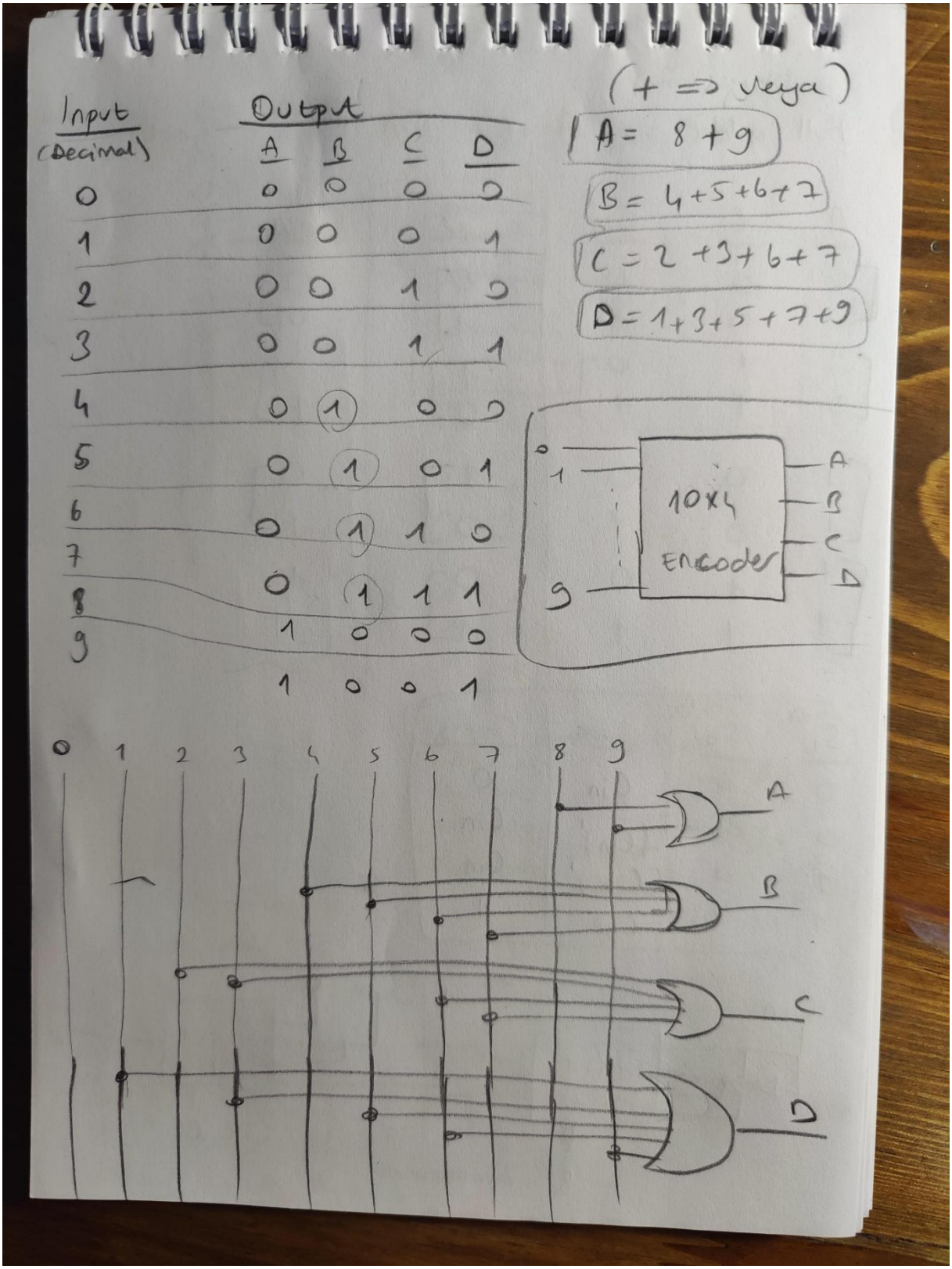
Çalışmamızı simüle ettiğimizde sonuçlarımız şöyle çıkacaktır:




```
# run 1000ns
A : 1000 , Y : 11
A : 0100 , Y : 10
A : 0010 , Y : 01
A : 0001 , Y : 00
A : 0000 , Y : 00
$finish called at time : 50 ns : File "C:/Users/zulal/OneDriv
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_prori
```

ÖRNEK 2: Decimal sistemden binary sisteme dönüşüm yapan bir encoder tasarlayınız.

Öncelikle doğruluk tablosunu çıkartmakla başlayalım:

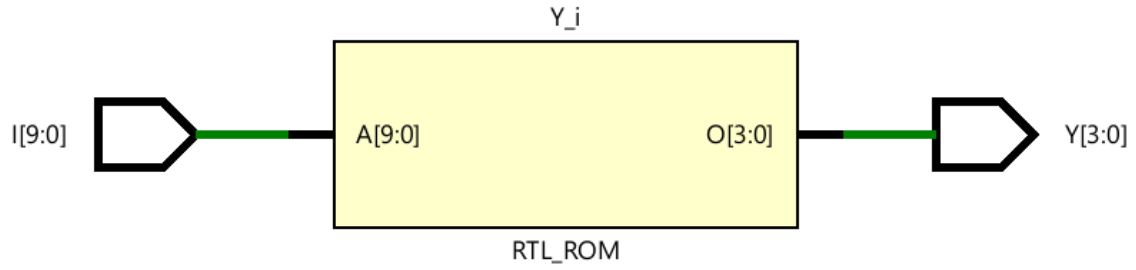


Design kodumuzu yazmaya başlayabiliriz:

```
module decimal_to_binary (
    input  logic [9:0] I,
    output logic [3:0] Y
);

    always_comb begin
        case (I)
            10'b0000000001: Y = 4'd0; //d = decimal
            10'b0000000010: Y = 4'd1;
            10'b0000000100: Y = 4'd2;
            10'b0000001000: Y = 4'd3;
            10'b0000010000: Y = 4'd4;
            10'b0000100000: Y = 4'd5;
            10'b0001000000: Y = 4'd6;
            10'b0010000000: Y = 4'd7;
            10'b0100000000: Y = 4'd8;
            10'b1000000000: Y = 4'd9;
            default: Y = 4'd0;
        endcase
    end

endmodule
```

(NOT: Hem kullanılan kaynak kitapta hem de internette kendiniz araştırdığınızda case yapısı için "casez" veya "casex" yapılarını görebilirsiniz fakat bu yapılar artık "obsolote" olarak geçerler. Kodunuzda oluşabilecek mevcut hataları (özellikle "?" gibi joker ifadeleri) normalmiş gibi görerek davranabilirler ve simülasyonda da sıkıntı çıkarmadıkları için anlayamayabilirsiniz. İş, herhangi bir FPGA kartına gömmeye gelince hatanın kendini göstereceğinden ötürü hem maliyet hem de zamandan kayıp sağlayabilirsiniz. Bu yapılar Verilog dilinde hala mevcut. System Verilog'taki "priority case" ve "unique case" yapıları Verilog'ta mevcut olmadığı için bu yapılar hala kullanılıyor fakat riskli oldukları gerçeğini değiştirmiyor. O yüzden kullanmamaya dikkat etmekte fayda var.)

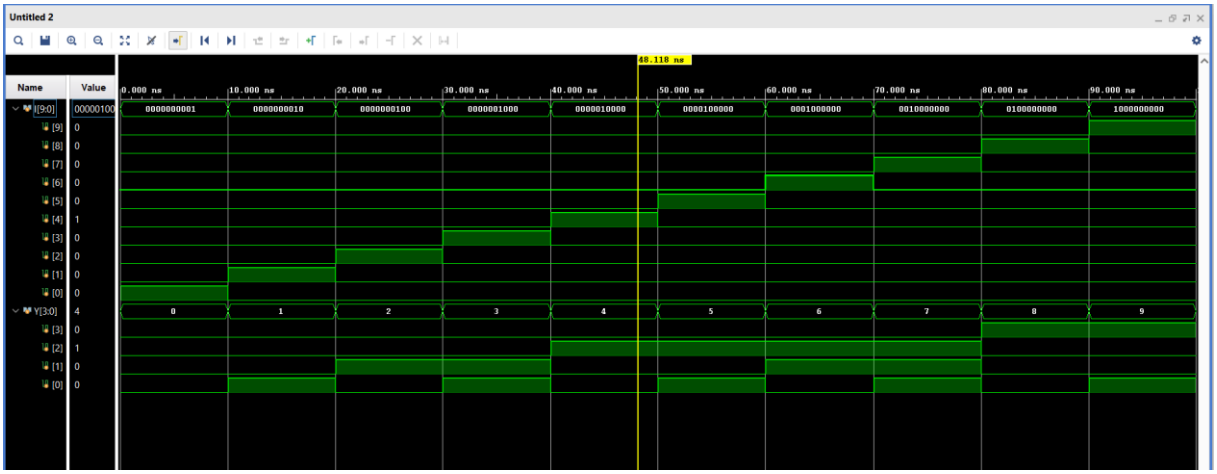
Design çıktımız şu şekil olacaktır:



Şimdi de testbanch kodumuzu yazalım:

```
module tb_decimal_to_binary();
    logic [9:0] I;
    logic [3:0] Y;

    decimal_to_binary decimal_to_binary(.*);

    initial begin
        for (int i = 0; i < 10; i++) begin
            I = 1 << i;  // Sadece bir biti 1 yap
            #10;
            $display("I = %b --> Y = %d", I, Y);
        end
        $finish;
    end

endmodule
```

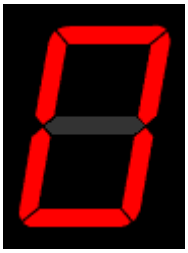
Sonuçlarımız şu şekil görünecektir:

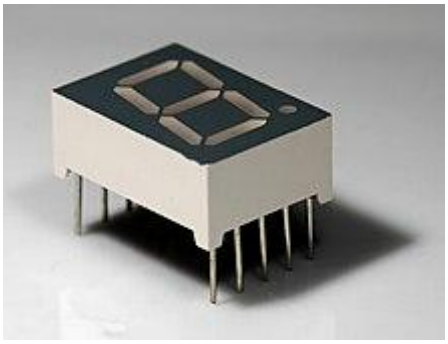



```
# run 1000ns
I = 0000000001 --> Y =  0
I = 0000000010 --> Y =  1
I = 0000000100 --> Y =  2
I = 0000001000 --> Y =  3
I = 0000010000 --> Y =  4
I = 0000100000 --> Y =  5
I = 0001000000 --> Y =  6
I = 0010000000 --> Y =  7
I = 0100000000 --> Y =  8
I = 1000000000 --> Y =  9
$finish called at time : 100 ns : File "C:/Users
INFO: [USF-XSim-96] XSim completed. Design snaps
INFO: [USF-XSim-97] XSim simulation ran for 1000
```

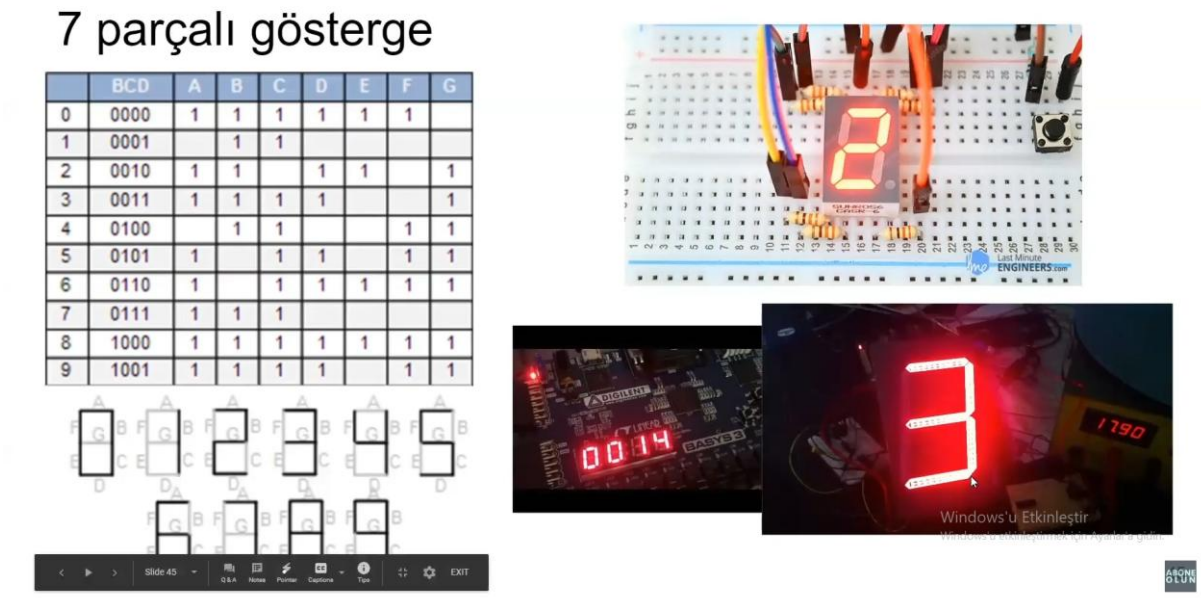
ÖRNEK 3: Bir “BCD to 7 segment decoder”ı System Verilog ile tasarlayınız.

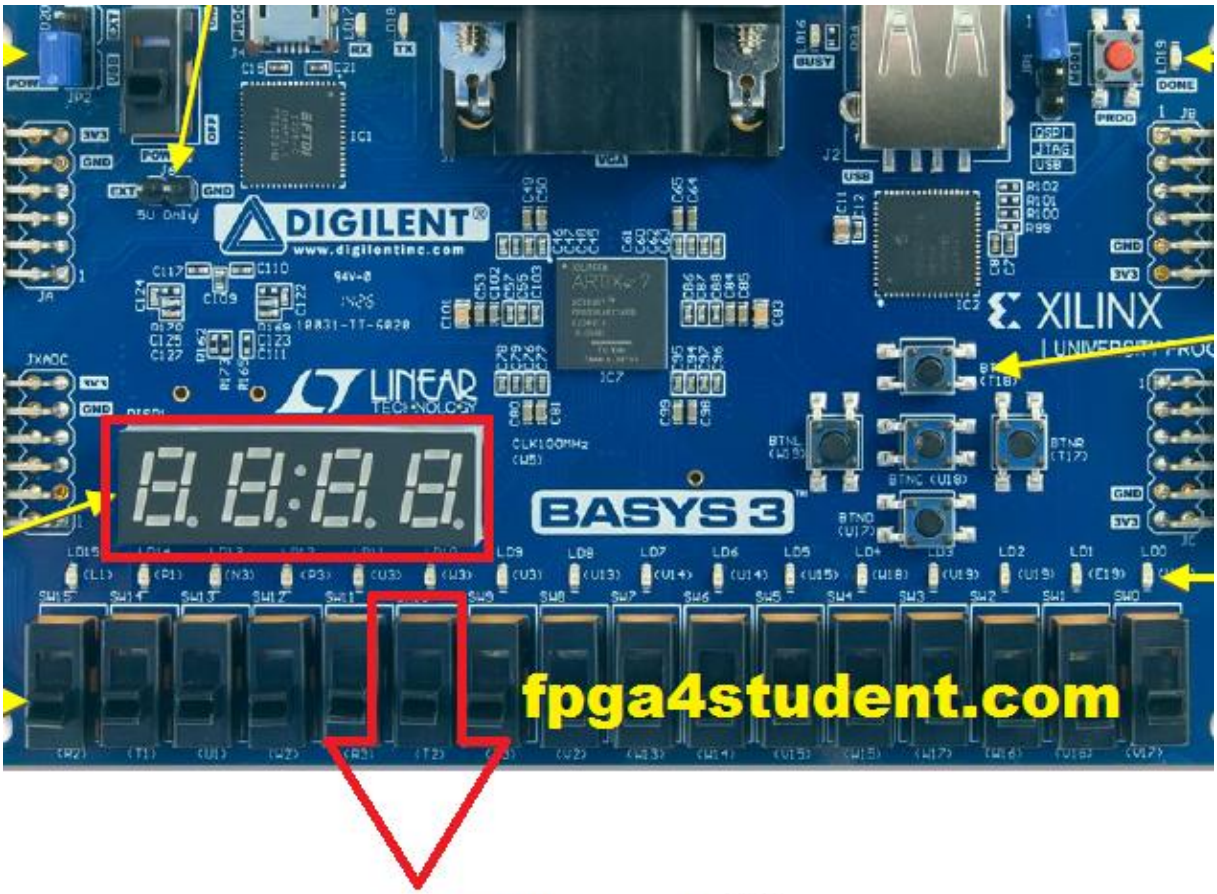
Öncelikle yedi segmentli LED görüntü modülü nedir bunu öğrenelim. Yedi segment LED görüntü modülü ondalık sayıların gösteriminde kullanılan elektronik bir görüntülüme cihazdır. Yedi segment göstericiler dijital saatlerde, elektronik sayaçlarda ve diğer birçok elektronik cihazda sayısal bilgi görüntülemek için yaygın olarak kullanılmaktadır. Çizgi seklindeki LED ya da LCD parçaların sekiz rakamının görünüşüne benzer şekilde bir modül üzerine dizilmesi ile oluşturulur. Her bir parçanın uçları modülün alt kısmında bulunan ayrı bir bacaklara bağlıdır. Görüntü belirli bacaklar bir devreye bağlanıp LED ya da LCD parçalar üzerinden akım geçmesi sağlanarak üretilir.

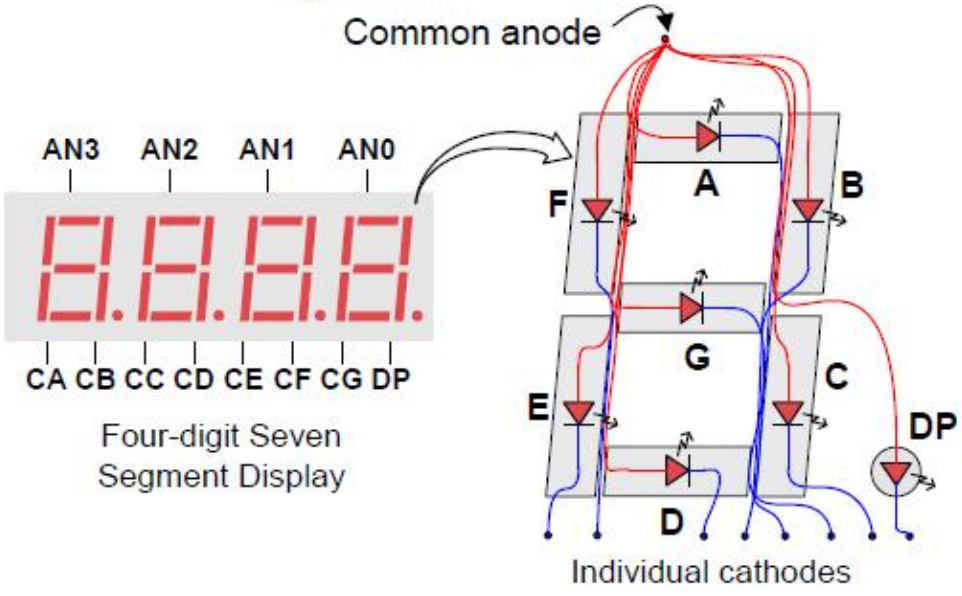




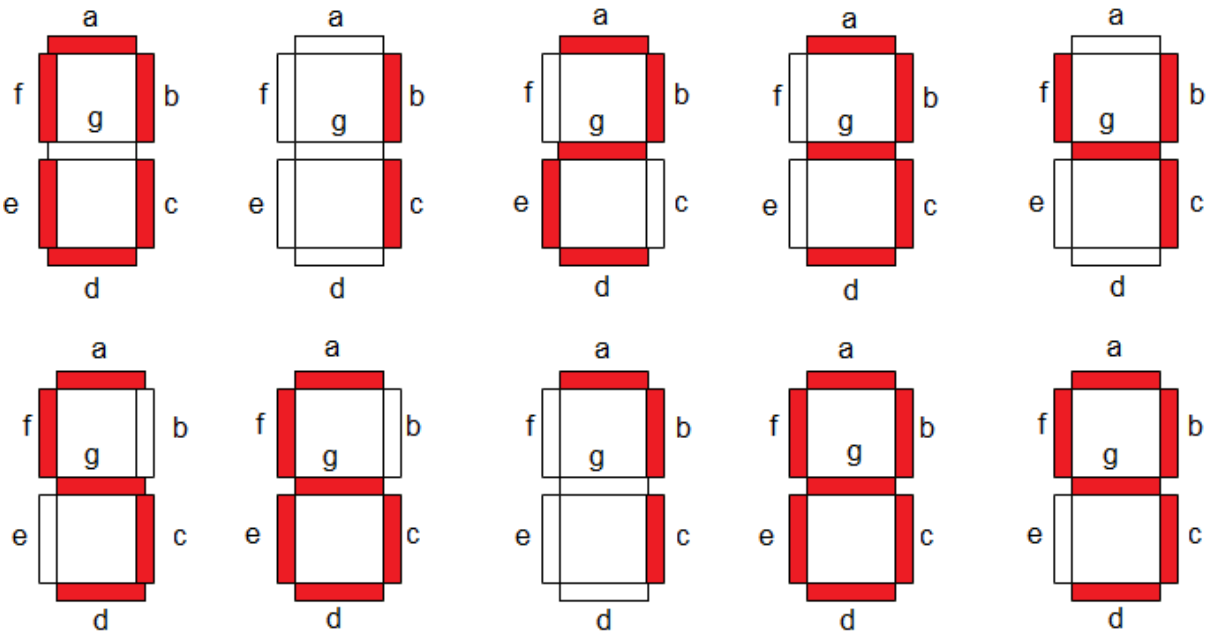
Günlük hayatta asansörler, dijital saatler gibi kullanım yerlerinin yanı sıra FPGA kartlarının üzerinde ve daha bir çok alanda kullanılır.




DIGILENT
www.digilentinc.com
XILINX
LINEAR TECHNOLOGY
BASYS 3
fpga4student.com


Common anode
AN3
AN2
AN1
AN0
CA CB CC CD CE CF CG DP
Four-digit Seven
Segment Display
A
B
C
D
E
F
G
DP
Individual cathodes

Devre şemamızın mantığına geçecek olursak:



Sayılar bu şekilde görünecektir. Örneğin “0” sayısı için **a,b,c,d,e ve f** ledlerinin yanması gerekir ve tasarlayacağımız devrede bunun binary olarak karşılığı “1111110” olacaktır.

Tüm sayılar için durum şöyle gözükecektir:

Decimal `abcdefg` (7 segment output)

0 → `1111110`

1 → `0110000`

2 → `1101101`

3 → `1111001`

4 → `0110011`

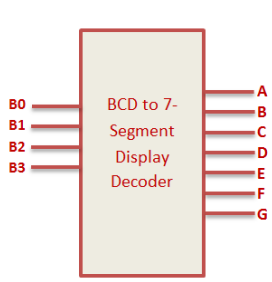
5 → `1011011`

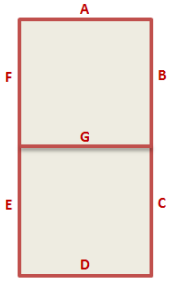
6 → `1011111`

7 → `1110000`

8 → `1111111`

9 → `1111011`

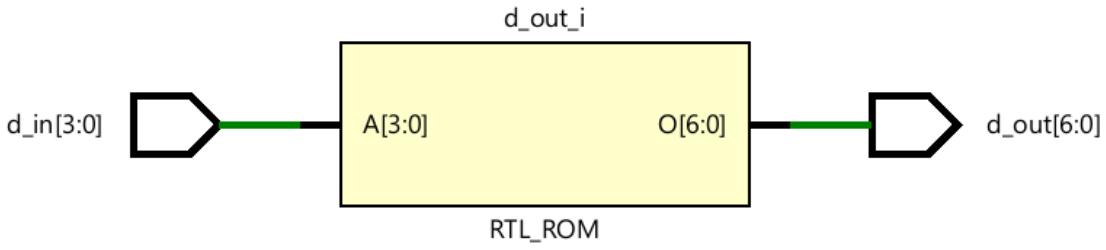




Design kodumuzu yazmaya başlayabiliriz:

```
module bcd_to_7_segment_decoder(
    input logic [3:0] d_in,
    output logic [6:0] d_out
    );

    always_comb
    begin
        case(d_in)
            4'b0000: d_out = 7'b1111110; //0
            4'b0001: d_out = 7'b0110000; //1
            4'b0010: d_out = 7'b1101101; //2
            4'b0011: d_out = 7'b1111001; //3
            4'b0100: d_out = 7'b0110011; //4
            4'b0101: d_out = 7'b1011011; //5
            4'b0110: d_out = 7'b1011111; //6
            4'b0111: d_out = 7'b1110000; //7
            4'b1000: d_out = 7'b1111111; //8
            4'b1001: d_out = 7'b1111011; //9
            default: d_out = 7'b0000000;
        endcase
    end
endmodule
```

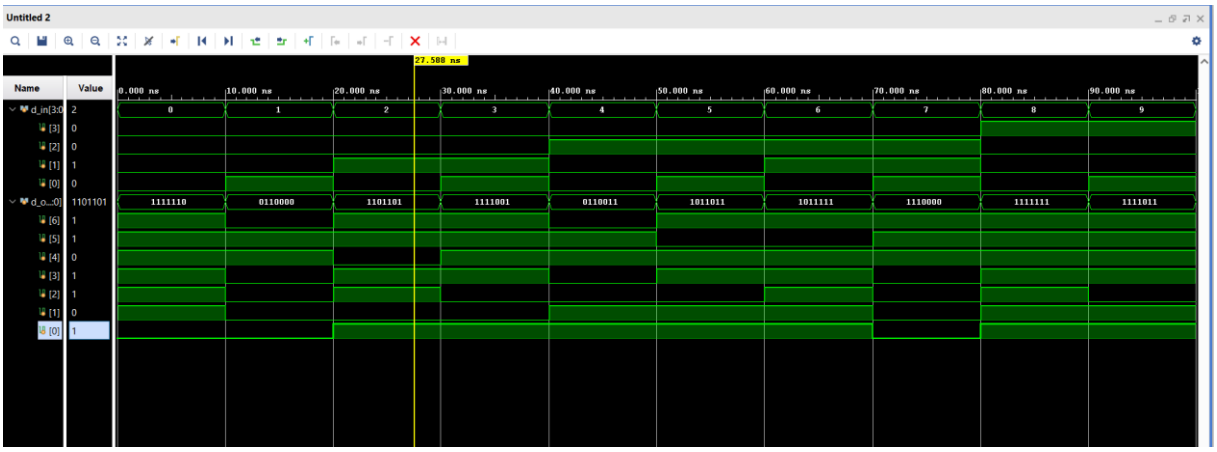
Design çıktımız şöyledir:



Artık testbanch kodumuzu yazmaya başlayabiliriz:

```
module tb_bcd_to_7_segment_decoder();
    logic [3:0] d_in;
    logic [6:0] d_out;

    bcd_to_7_segment_decoder bcd_to_7_segment_decoder(.*);

    initial begin
        for(int i = 0; i<10; i++)
        begin
            d_in = i;
            #10;
            $display("d_in: %d --> d_out: %b",d_in, d_out); // inputları
decimal olarak görelim
        end
        $finish;
    end

endmodule
```

Simülasyon çıktılarımız şu şekilde olacaktır:




```
# run 1000ns
d_in:  0 --> d_out: 1111110
d_in:  1 --> d_out: 0110000
d_in:  2 --> d_out: 1101101
d_in:  3 --> d_out: 1111001
d_in:  4 --> d_out: 0110011
d_in:  5 --> d_out: 1011011
d_in:  6 --> d_out: 1011111
d_in:  7 --> d_out: 1110000
d_in:  8 --> d_out: 1111111
d_in:  9 --> d_out: 1111011
$finish called at time : 100 ns : File "C:/Use
INFO: [USF-XSim-96] XSim completed. Design sna
INFO: [USF-XSim-97] XSim simulation ran for 10
```