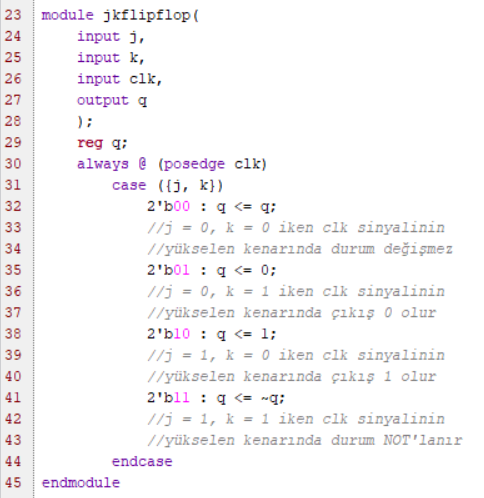
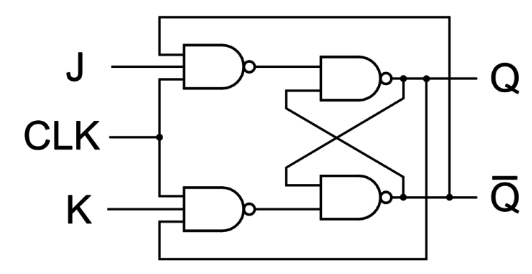
**JK Flip Flop ve test bench:**

metin, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

J = 0, K = 1 iken saat sinyalinin ilk yükselen kenarında çıkışımız 0 değerini aldı.

J = 0, K = 0 iken çıkışın son değeri ne ise o halde kalmaya devam etti.

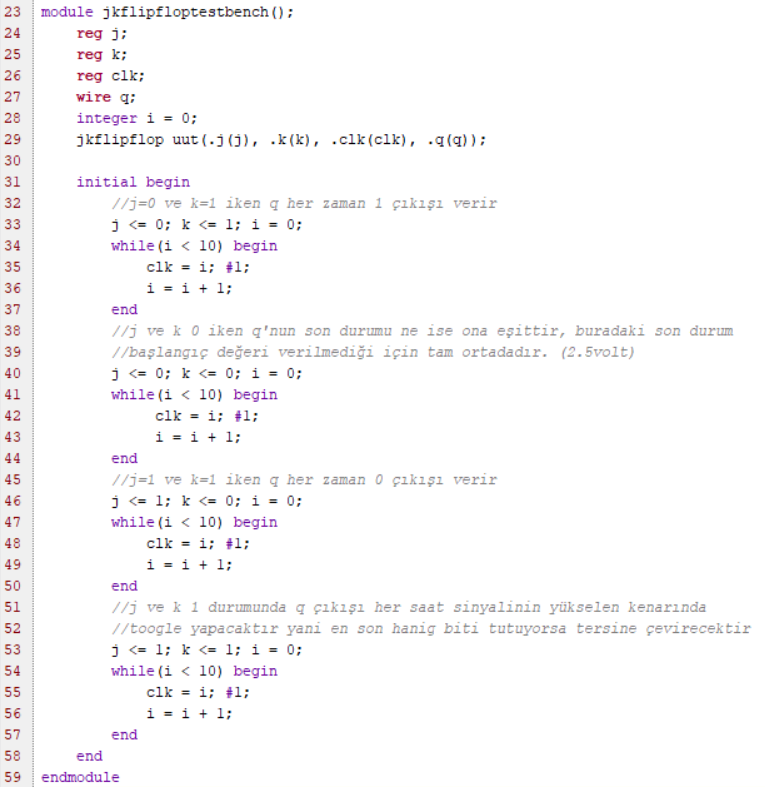
J = 1, K = 0 iken saat sinyalinin ilk yükselen kenarında çıkışımız 1 değerini aldı.

J = 1, K = 1 iken saat sinyalinin her yükselen kenarında çıkışımız durum değiştirdi

Alt kısımda yukarıdaki şemanın kodlarını paylaştım.

While döngüleri 1er nanosaniyelik sürede bir saat sinyali üretiyor ve buna göre o anki j ve k durumlarına göre q çıkışına değer atanıyor.

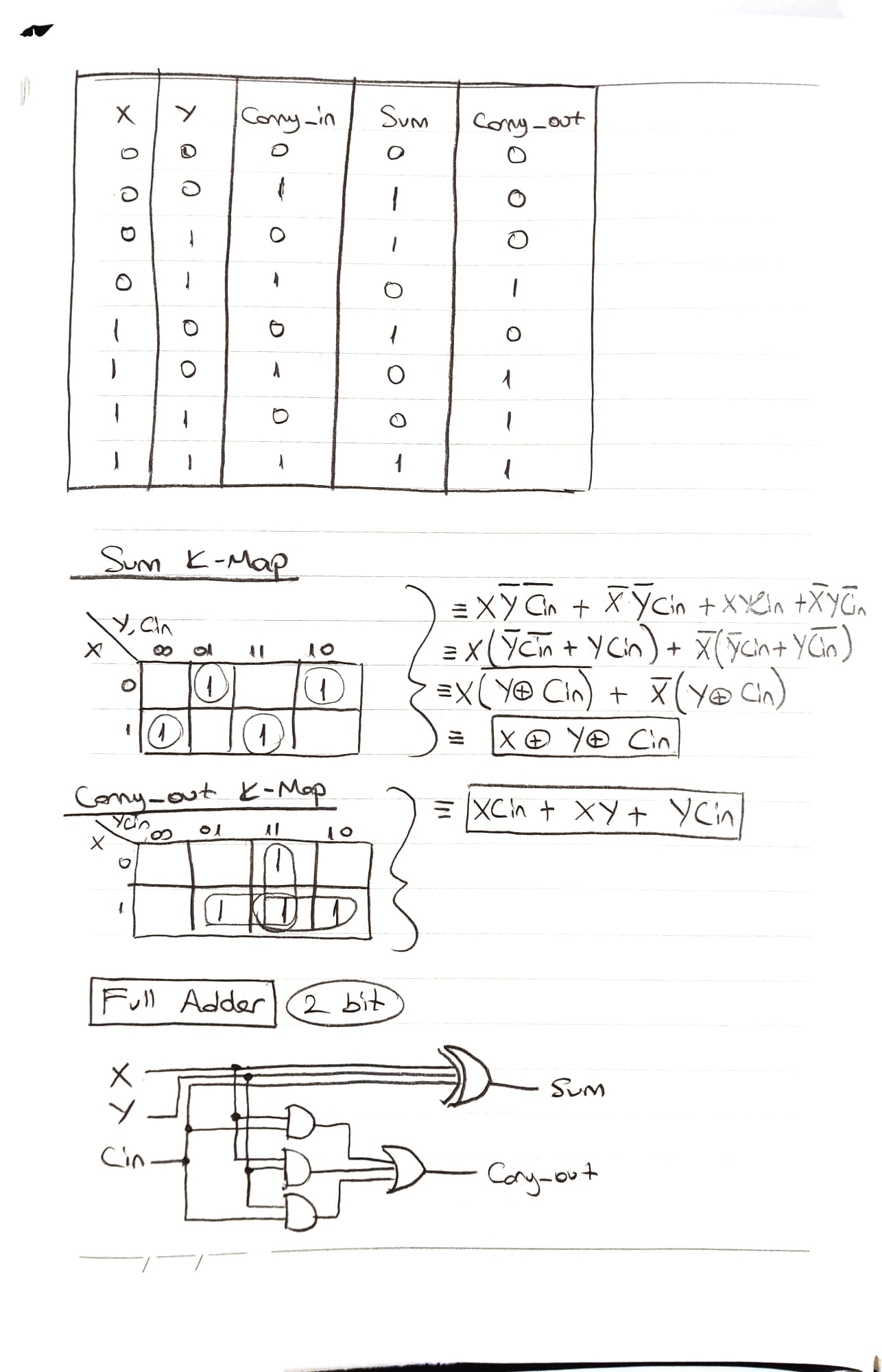
Aslında clk = ~clk olarak atama yapmam gerekirdi. Clk = i şeklinde yaptığım atamanın çalışma nedeni, i değerinin LSB bitinin her artışında değişiyor olması, yani …00001, ….00010, ….00011, …00100

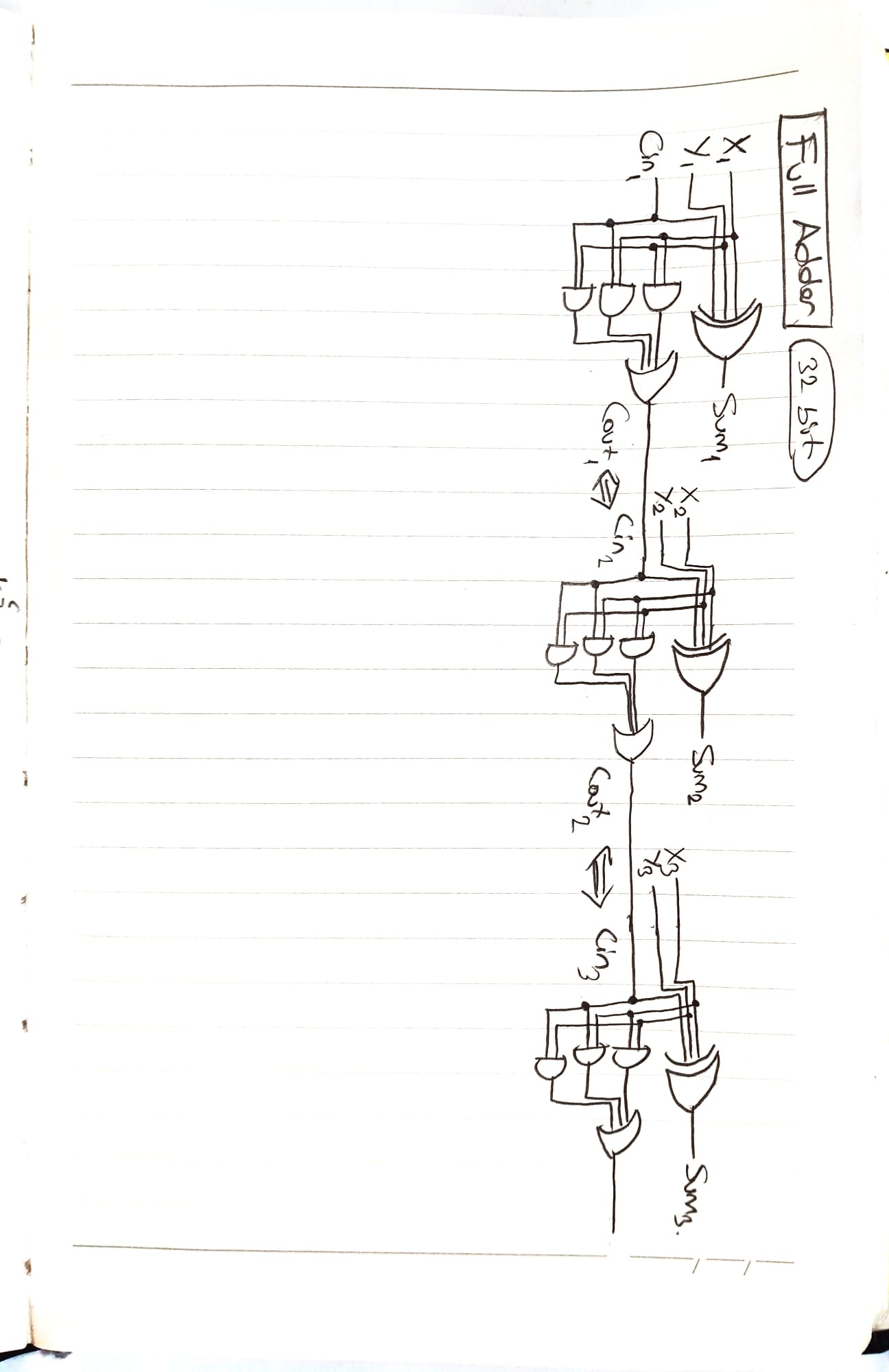


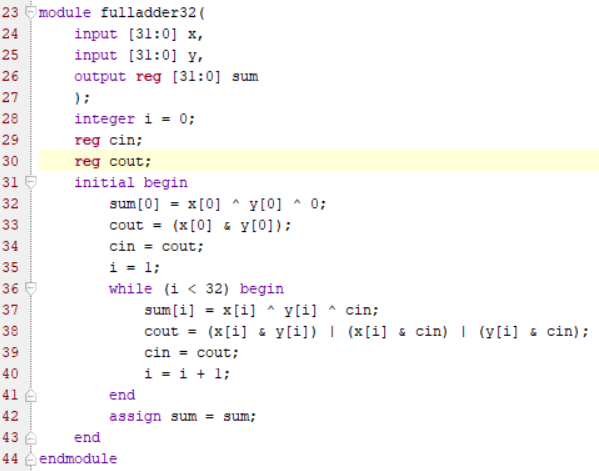
**32 bit toplayıcı modül:**

Önce 2 biti toplayan tam toplayıcı devrenin karnough haritasını ve mantıksal devresini çizdim.

Ardından 32 bitlik olması için bu devreleri 32 tane uç uca ekledim. Eklemeyi yaparken bir önceki bitten çıkan elde (carry\_out) bitini bir sonraki devrenin elde girişine (carry\_in) uzattım.

****

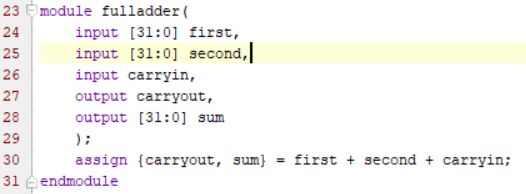
Bu şekilde 32 adet devam ediyor. İlk carry\_in değerini 0 varsayıyoruz çünkü ilk bit’in eldesi ikinci bit’e aktarılıyor.



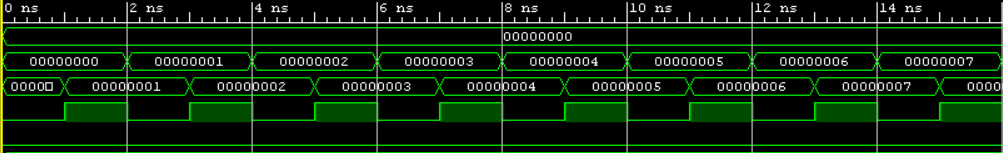
Çözemediğim bir hata alıyorum, fakat yukarıdaki devrenin modül kodu bu şekilde yazılmalıydı. Neden çalışmadığını anlayamadım.

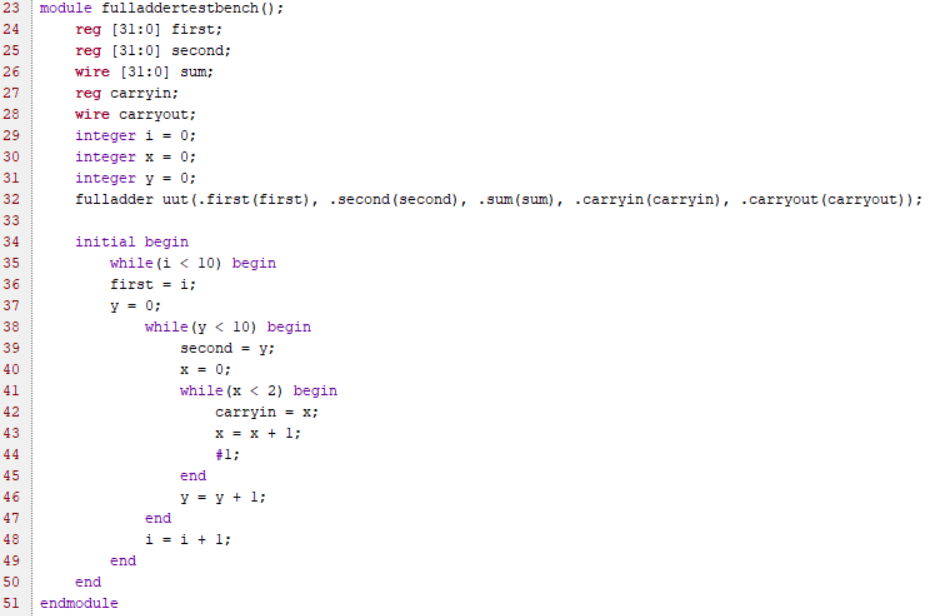
Bir aşağıda anlamadığım ama çalışan bir kod bırakıyorum.

-------------------------------------------------------------------------------------------------------------------------------------------



Çalışan modül burada!





0 + 0’dan başlayarak her bir toplamayı elde biti 1 ile birlikte topladığımızda karşımıza çıkan tablo.

İlk satır birinci toplanacak sayıyı,

İkinci satır ikinci toplanacak sayısı,

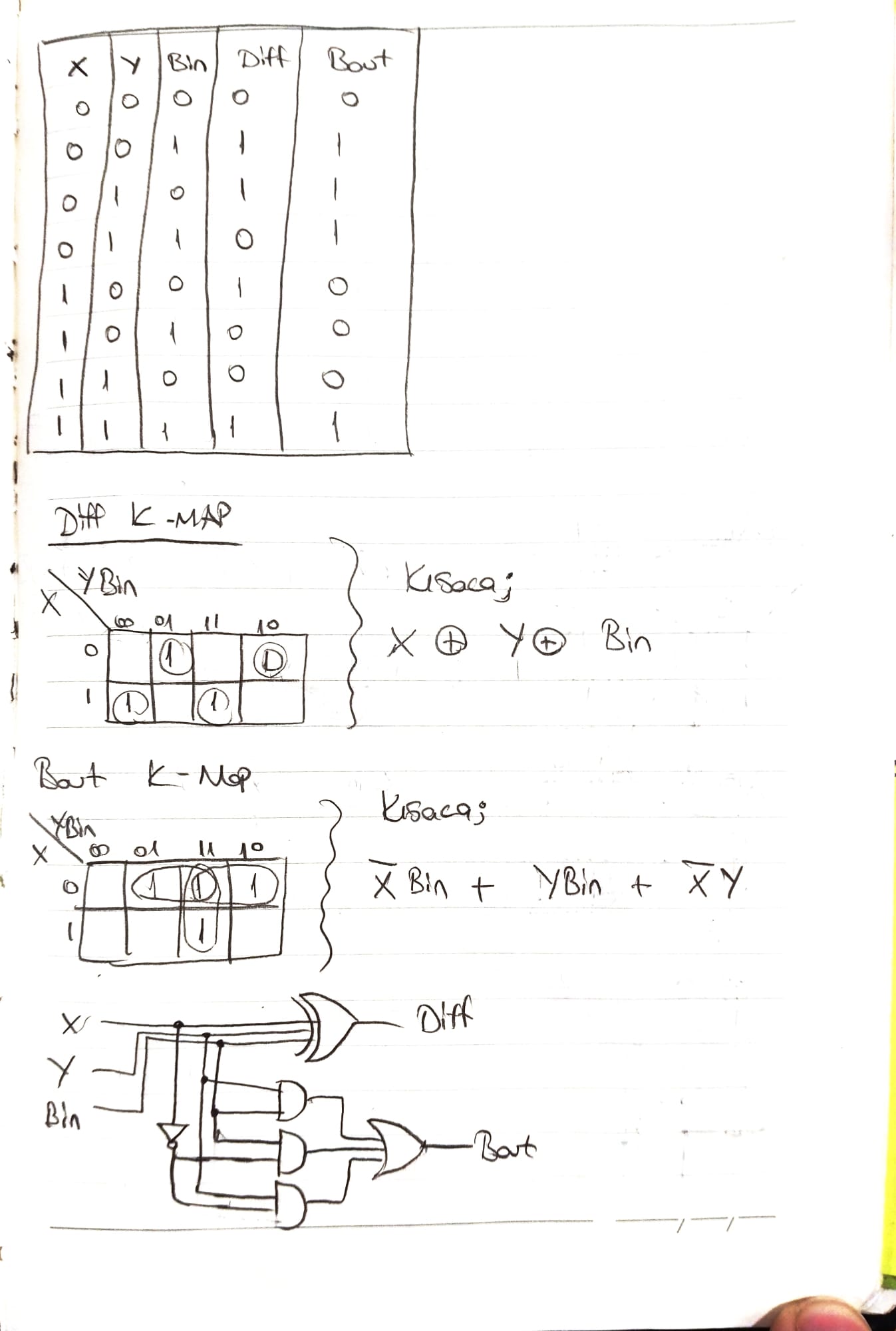
Üçüncü satır toplamı,

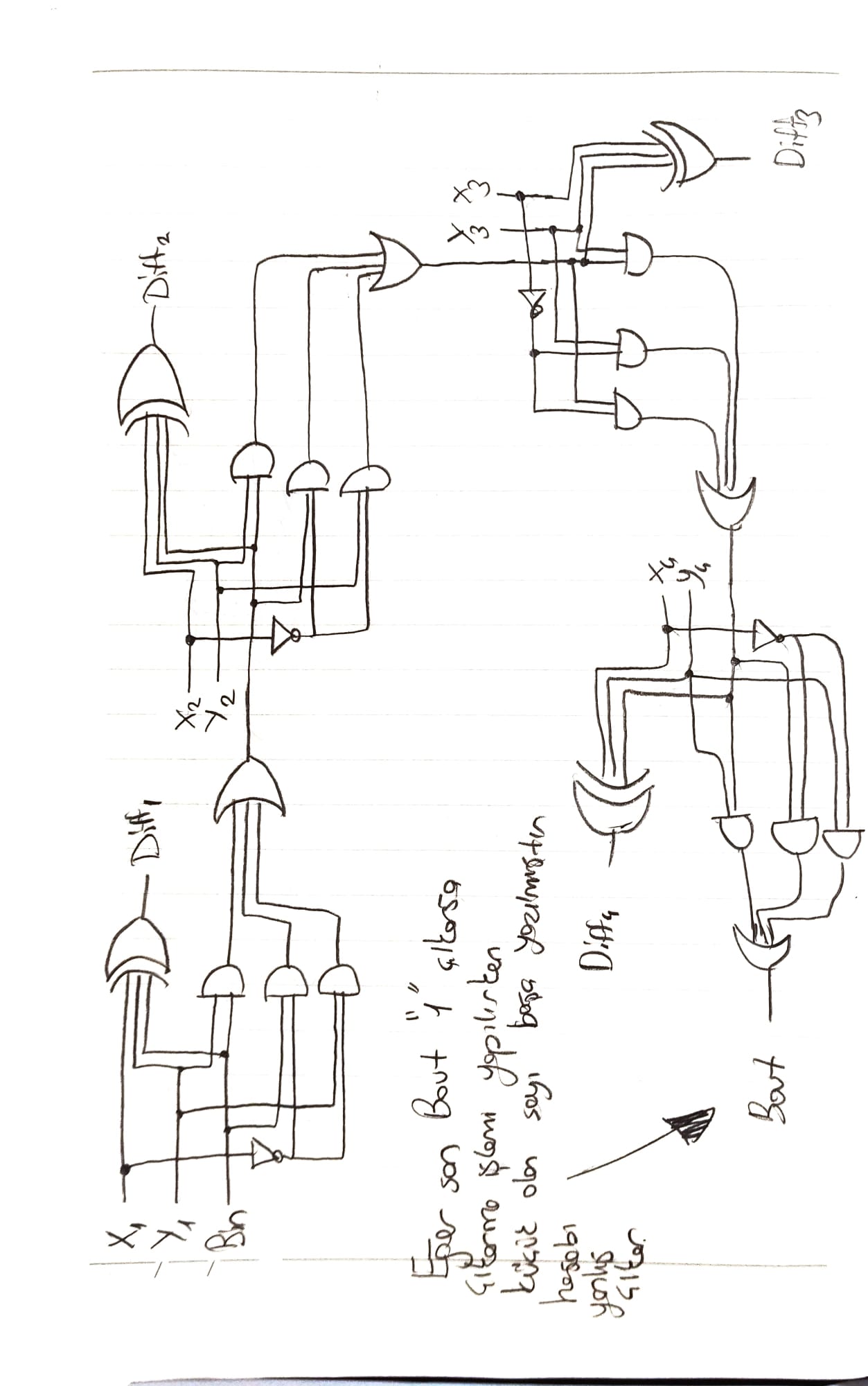
Dördüncü satır elde bitini

Beşinci satır ödünç bitini temsil ediyor

**4 bit çıkarıcı:**

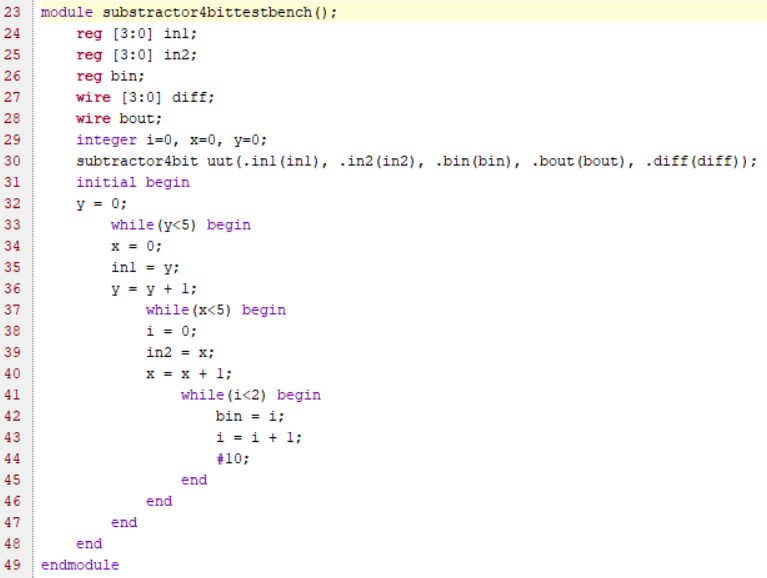
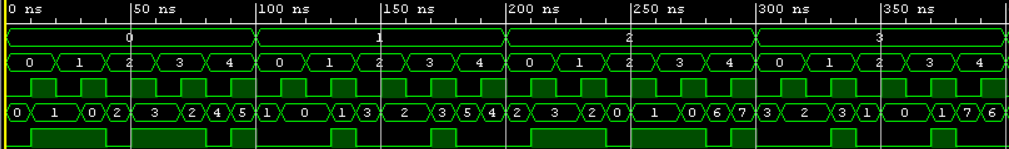
Tekrar tam çıkarıcı devresini çizerek başladım.





metin içeren bir resim

Açıklama otomatik olarak oluşturulduMantık devresindeki yapıyı koda geçirdiğimizde şu şekilde oluyor:



İlk satır çıkan sayı,

İkinci satır çıkarılan sayı,

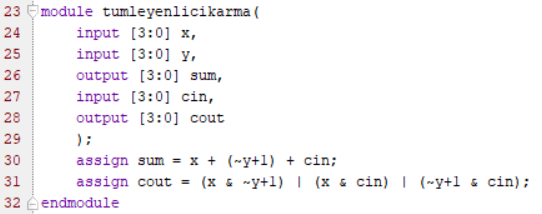
Üçüncü satır ödünç varsa veya yoksa farkı görmek için,

Dördüncü kısım sonuç

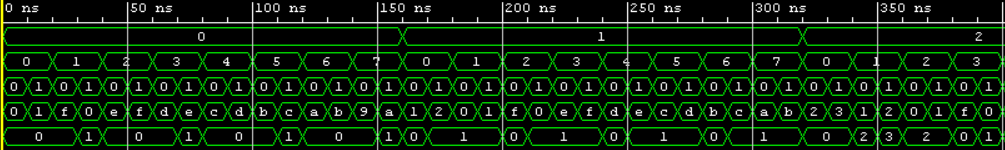
Beşinci kısım da ödünç alınma durumu

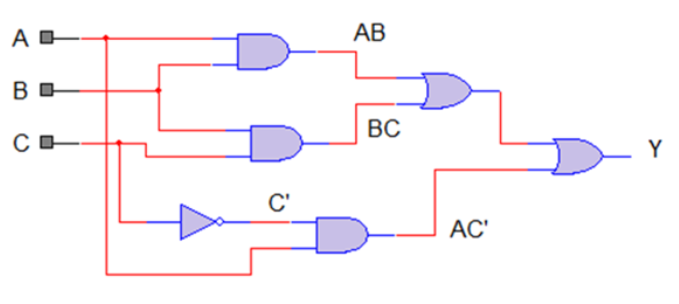
Tasarımıma göre sadece büyükten küçüğü çıkarırken doğru cevabı elde edebiliyorum. Sistem işaretli sayılar için uygun olmadığından farklı bir yöntem deneyeceğim

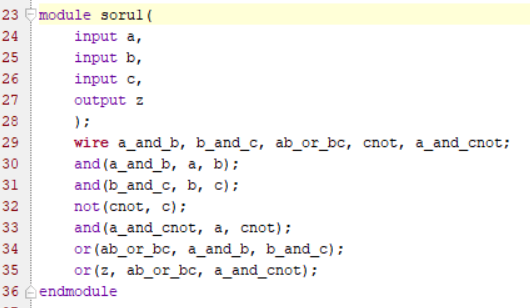
Tam çıkarma yapmak için 2. tümleyeni ile toplama yöntemini kullanabiliriz. İkinci sayının tüm bitlerinin tersini alıp 1 ekleyerek 2. tümleyeni elde ediyoruz. Bu iki sayıyı topladığımızda en sonda bir elde biti kalıyorsa sonuç pozitiftir ve doğrudur, elde biti kalmıyorsa sonuç negatiftir ve tekrar 2. tümleyeni alınır. Başına negatif işareti konulur.

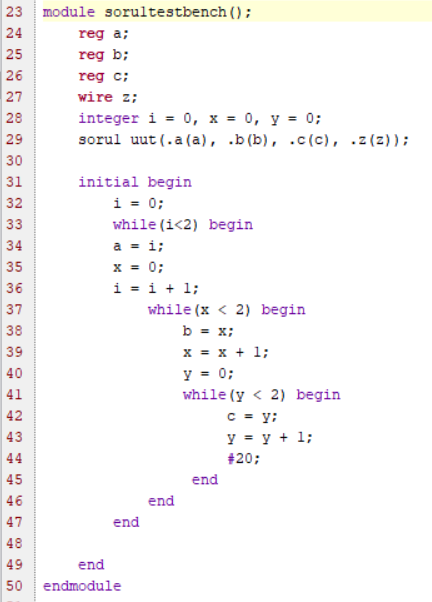


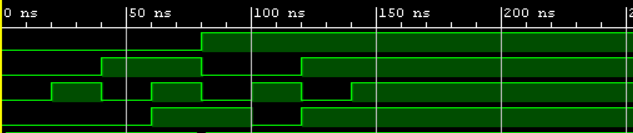
Full toplama işleminin aynısını ikinci sayının 2 ikinci tümleyenini alarak yaptım. Simulasyona bakınca; örneğin 0000’dan 0001 çıkardığımızda sonuç olarak F değerini yani 1111 değerini dönmüş. Aslında kod düzgün çalışıyor, sadece tekrar ikinci tümleyenini almak gerekiyor. 1111 in ikinci tümleyeni 0001 ve 0 – 1 de -1 yapıyor.

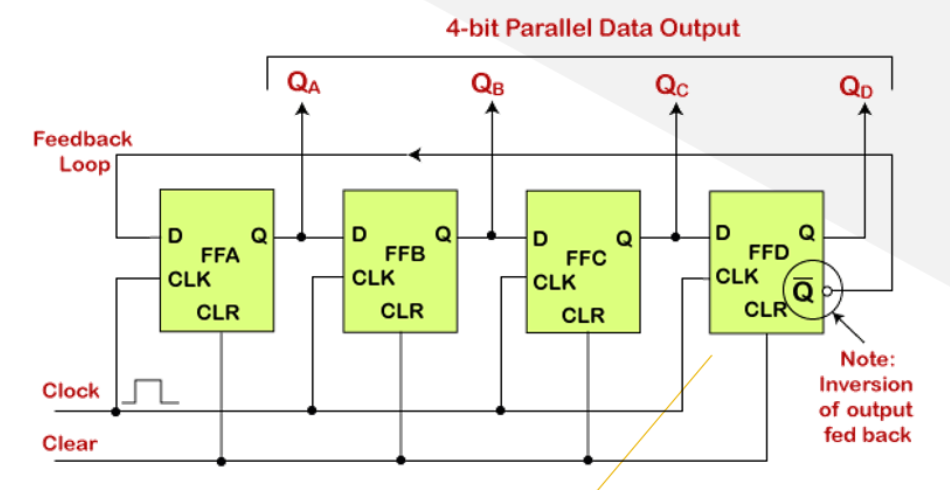


**Soru 1:**







**Johnson Sayıcı devresi:**

Başlangıç : 1000

İlk FF’un D1 girişine 1 sinyali vererek başlanır. Saat darbesinin yükselen kenarı gelir gelmez Q1 çıkışında 1 değerini alırız, bu sayede D2 girişi de 1 sinyali gelmiş olur.

Sonuç : 1100

Sistem bir sonraki saat sinyalinin yükselen kenarını bekler ve bitler bir defa daha kayar.

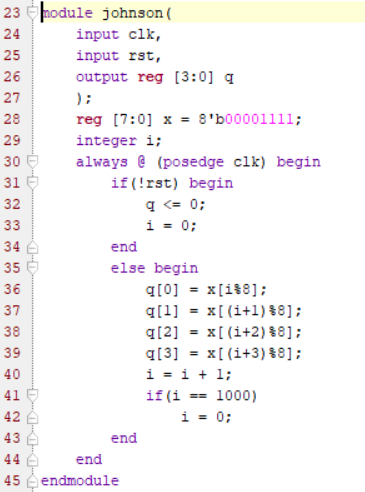
Sonuç : 1110

Bir sonraki aşamada tekrar saat sinyali beklenir ve bitler bir defa daha kayar.

Sonuç : 1111

Şimdi ise son FF’un yani Q4 çıkışının değeri 1 olduğu için, bu değerin tersi ilk FF’un girişine yansıtılır. D1 girişi 0 olduğunda ilk saat sinyalinde çıkışı da 0 yapacaktır ve döngü böylece devam eder.

Sonuç : 0111 – 0011 – 0001 – 0000 – 1000 – 1100 – 1110 -----------



Burada bit kaydırma işini kolaylaştırmak için 8 bitlik register tanımladım. 4 bitlik çıkışı yukarıdaki sayma sırası oluşacak şekilde bir döngü içine aldım. Reset gelmediği sürece kaldığı yerden saymaya devam eder. Reset geldiğinde ise sayma sıfırlanır ve baştan başlar.

