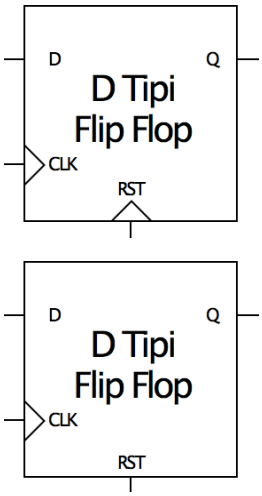
Verilog HDL ile Davranışsal Modelleme 5
(Sıralı Mantık Devrelerinin Tasarımı)

Önceki lablarda gerek kapı seviyesinde gerekse davranışsal olarak modellediğimiz devrelerin herhangi birinde saat vuruşu veya resetleme gibi kavramlardan bahsetmedik. Aslında şu ana kadar tasarlanan tüm devreler hafızası olmayan eski bir zaman dilimine ait veri barındırmayan, o anki girişlerine göre çıkışı değiştiren devrelerdi.

Oysa ki bugün en basit bir makinenin bile tasarımında makinenin o anki durumu ve geçkmişteki durumları hakkında bilgi veren hafıza birimlerine ihtiyaç duyuyoruz. Verilog HDL ile bu devre parçalarının nasıl tasarlandığı bu dokümanda anlatılmıştır.

D tipi flip flop



Sağdaki şekilde gösterilen D Flip Flop, her saat vuruşunun yükselen kenarında D telinden okuduğu değeri Q teline yazar. Eğer herhangi bir anda RST sinyalinin yükselen kenarı algılanırsa Q çıkışı sıfırlanır. Bu flip flopta reset sinyali asenkron olarak çalışır. Saat veya reset sinyallerinden herhangi birinde yükselen kenar algılanmazsa, D girişi ne kadar değişse de Q çıkışının değeri değişmez. Dolayısıyla Q çıkışı hafızalı bir yapıdır. D, CLK ve RST sinyallerinin ise hafızalı olup olmadığı konusunda bir şey söylenemez.

Sağdaki flip flop ise bir öncekinin aksine senkron bir RST sinyaline sahiptir. Dolayısıyla Q çıkışı yalnızca saatin yükselen kenarında değişir. Saatin yükselen kenarı geldiği anda RST = 1 ise Q çıkışı sıfırlanır, RST = 0 ise Q çıkışına D girişinden okunan değer yazılır. Burada da bir önceki gibi Q çıkışının hafızalı olduğunu söyleyebilir, diğer sinyaller için yorum yapamayız.

```
module d_ff (
  input wire clk, rst, D,
  output reg Q
);
```

Her iki flip flop için de Q çıkışı hafızalı bir yapı gerektirir. Verilog HDL bize 2 çeşit yapı sunuyordu. Bunlardan hafızalı olanı reg idi. Dolayısıyla Q çıkışı reg tanımlanmalıdır. Bu durumda modül şu şekilde tanımlanabilir:

```
always @ (posedge clk)
begin
  . . .
  . . .
  . . .
end
```

Q sinyali reg olarak tanımlandığı için atamasının da always bloğu içinde yapılması gerekir. Bir always bloğu yazıldığında hassasiyet listesi de eklenmelidir. Eğer 2. flip flopu modelliyorsak, Q sinyali yalnızca saatin yükselen kenarında değiştiği için hassasiyet listesine **posedge clk** yazmak zorundayız.

```
always@(posedge clk, posedge rst)
begin
  . . .
  . . .
  . . .
end
```

Eğer ilk flip flop devresini modelliyor olsaydık Q çıkışı saatten bağımsız olarak reset sinyalinin yükselen darbesinde de değişeceğinden hassasiyet listesi şöyle olmalıydı:

posedge clk , posedge rst

Burada **posedge** yeni gördüğümüz bir anahtar kelime olup yükselen kenar anlamına gelir. Düşen kenar için kullanmamız gerekseydi **negedge** kullanabilirdik.

```
always@(posedge clk, posedge rst)
begin
  if (rst) begin
      Q = 0;
  end
  else begin
      Q = D;
  end
end
```

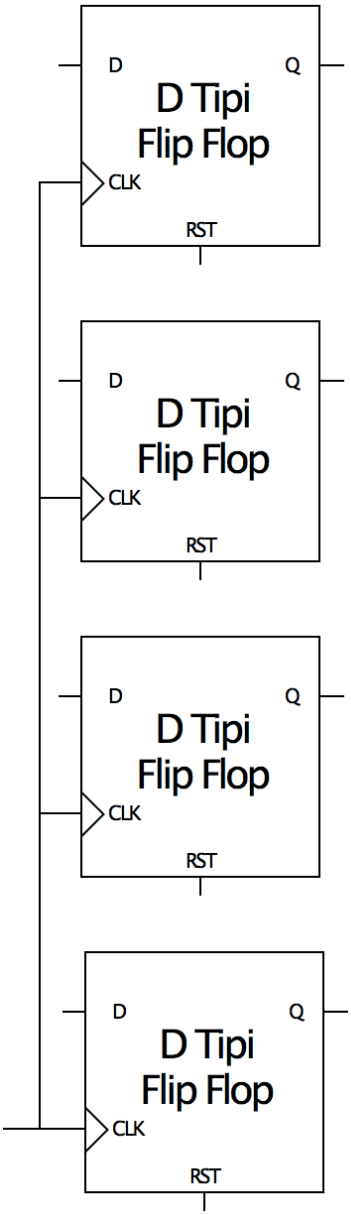
Always bloğunun içini yazabilmek için D tipi flip flopun nasıl davranması gerektiğini sorgulamamız lazım. D tipi flip flop, çıkışını değiştirmesi gereken bir durum oluşmuşsa (saat yükselen darbesinin gelmesi gibi), reset sinyaline bakar, reset = 1 ise Q = 0, reser = 0 ise Q = D işlemini gerçekleştirir. Bunu sentez aracının anlayabileceği şekilde Verilog HDL'de soldaki gibi yazabiliriz.

DİKKAT!

Daha önceden **always** içine yazılan kod satırlarının ard arda çalıştığını öğrenmiştik. Yalnız flip flopların ideal durumda 0 zaman harcayarak tamamen paralel bir şekilde bu işi yürütmesi gerekmektedir. Örneğin sağdaki şekilde tek saat sinyaline bağlanmış 4 farklı flip flop'un beklenen çalışma şekli saatin yükselen kenarı geldiğinde bütün Q değerlerini aynı anda değiştirmesidir. Burada herhangi bir Q değerinin başka bir Q değerini beklemesi gibi bir lüks yoktur.

Dolayısıyla hafızalı birim özelliğini veren reg yapısı ve always bloğu paralelliği sağlamadığı için yetersiz kalmaktadır.

Bu paralelliği sağlamak adına Verilog HDL dilinde always bloğu içindeki atamalar **blocking** ve **non-blocking assignment** olarak ikiye ayrılmıştır. Daha önce gösterildiği şekilde eğer always bloğu içindeki atamalar **=** sembolü ile yapılırsa tüm satırlar üstündeki satırları bekleyerek çalışır. Ancak atamalarda **<=** sembolü kullanılırsa burada paralellikten söz edebiliriz. always bloğu içindeki atamalar **<=** sembolü ile yapıldığı zaman tüm satırlar **paralel** olarak **aynı anda** çalışır.



KURAL 1: Bir always bloğu içindeki atamaların ya hepsi blocking ya da hepsi unblocking olmak zorundadır. Bazıları blocking bazıları unblocking yapılamaz.

KURAL 2: Yaygın kullanım olarak saatten bağımsız olan bileşik (combinational) devrelerde her zaman blocking (=), flip flop gibi bazı sinyallerin yükselen veya düşen darbelerinde çalışan devrelerde ise non-blocking (<=) atamalar kullanılır.

Sonuç olarak senkron resetli ve asenkron resetli iki farklı D flip flop Verilog HDL ile şu şekilde modellenmiş olur.

```
//Senkron reset kullanilan durum
always@(posedge clk) begin
  if (rst) begin
      Q <= 0;
  end
  else begin
      Q <= D;
  end
end
```

```
//Asenkron reset kullanilan durum
always@(posedge clk, posedge rst) begin
  if (rst) begin
      Q <= 0;
  end
  else begin
      Q <= D;
  end
end
```

Yazmaç modelleme

Yazmaç adı verilen donanımlar flip flop dizileri ile oluştururlar. Bu sebeple Verilog HDL ile modelleme yapılırken herhangi bir yazmacı modellemek için kulanılan kod D tipi flip flop koduna oldukça yakındır. Örnek bir yazmaç modellemesini şu şekilde yapabiliriz:

Soru: 8 bit genişliğinde bir yazmacı Verilog HDL kullanarak modelleyin. wrEn girişi 1 ise yazmacın içine paralel olarak 8 bitlik data_in girişinde bulunan değer saat sinyalinin yükselen kenarında yazılmalıdır. wrEn girişi 0 ise yazmacın değeri değişmemelidir. Yazmacın değeri sürekli olarak 8 bitlik data_out çıkışından okunabilmelidir.

Tasarım:

Devrenin giriş ve çıkışları 8 bitlik data_in, data_out, 1 bitlik wrEn, saat ve reset olacaktır.

```
module yazmac (
  input  wire saat, reset, wrEn,
  input  wire [7:0] data_in,
  output wire [7:0] data_out
);
```

Yazmaç devrenin hafızalı bir elemanı olduğundan ne input ne de output olmadığından modülün içine reg olarak tanımlanmalıdır.

```
reg [7:0] yazmac;
```

Yazmacın değeri yalnızca saat vuruşunun gelmesi ile değil, **wrEn** sinyaline bağlı olarak değişecektir. Bu durumda **data_in** telini doğrudan always bloğunun yazmacın sonraki alacağı tutan bir başka tel tanımlamam gerekir. Bu telin değerini assign ile de **always** ile de değiştirebilirim, **assign** ile değiştireceğimi varsayarak **wire** olarak tanımlıyorum.

```
wire [7:0] sonraki_yazmac;
```

Flip flop dizisini tanımlayabiliriz.

```
always @ (posedge saat) begin
  if (rst) begin
    yazmac <= 8'd0;
  end
  else begin
    yazmac <= sonraki_yazmac;
  end
end
```

Mevcut kod, her saatin yükselen darbesinde yazmaca, yazmacın sonraki değerini atayan bir devreyi modeller. Ancak yazmacın sonraki değerinin nasıl hesaplanacağı tanımlanmamıştır. İstediğimiz davranış **wrEn** 1 ise **data_in'deki** değerin alınması, **wrEn** 0 ise kendi değerinin korunmasıydı. Bu durumda yazmacın sonraki değeri şu şekilde hesaplanabilir:

```
assign sonraki_yazmac = wrEn ? data_in : yazmac;
```

Böylece yazmaca yazma işlemi tamamlanmış olur. Tasarımda son istenen **data_out** çıkışından sürekli olarak yazmacın tuttuğu değerin verilmesiydi. Bu işlev de şu şekilde modellenebilir:

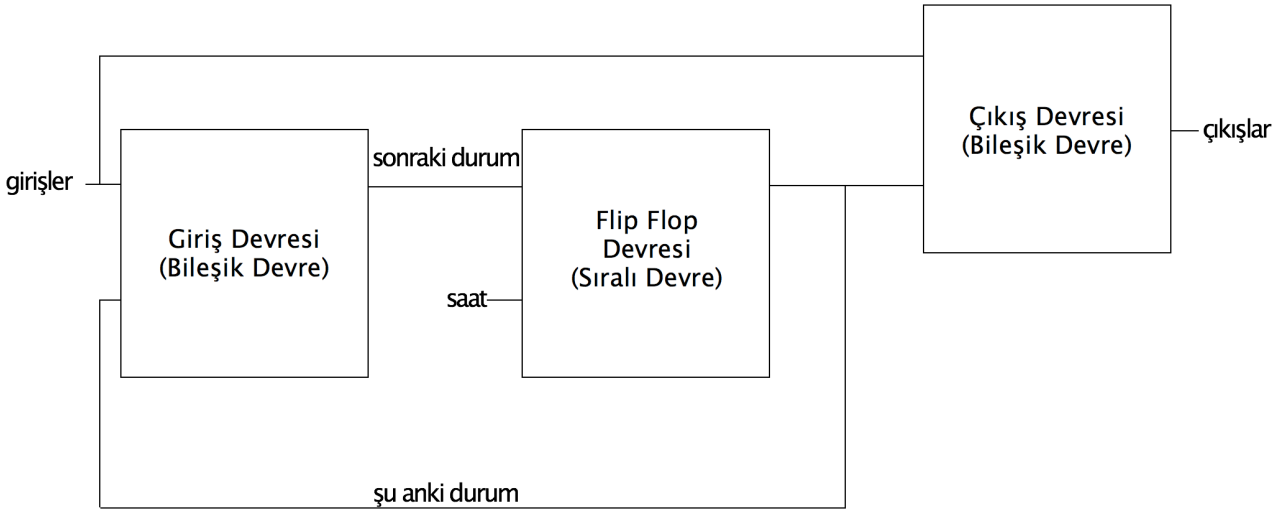
```
assign data_out = yazmac;
```

Bu şekilde istenen bir yazmaç modellenmiş olur.

DİKKAT! Senkron tasarımda Verilog HDL ile modellemede önemli bir nokta devrenin senkron çalışan kısmının olabildiğince yalın ve hesaplamadan uzak olması gerekliliğidir. Bu tasarımda da yazmacın sonraki değeri **posedge** ile tetiklenen always bloğunun **dışında** hesaplanmıştır.

Genel geçer tasarım mimarisi

Yazmaç devresi tasarlanırken yapıldığı gibi saat ile çalışacak devrelerin Verilog HDL modelleri çıkarılırken yaygın olarak aşağıdaki mimari çizimine uyulur. Bu ders kapsamında tasarladığınız bütün devrelerde de bu mimariye uymanız beklenmektedir.



Yazmaç tasarımı örneğinde verilen kod parçaları mimarideki bloklar ile şu şekilde eşleşir:

Giriş Devresi

```
assign sonraki_yazmac = wrEn ? data_in : yazmac;
```

Flip Flop Devresi

```
always @ (posedge saat) begin
  if (rst) begin
    yazmac <= 8'd0;
  end
  else begin
    yazmac <= sonraki_yazmac;
  end
end
```

Çıkış Devresi

```
assign data_out = yazmac;
```