Arkadaşlar merhaba,

Bu yazıyı yazmamın sebebi bir sınav sorusunu çözerken nasıl düşündüğümü, nelere dikkat ettiğimi sizlerle paylaşmak ve işinize yarayabilecek bazı bilgileri sunmak istememdir.

Soru ve Soru Hakkında Açıklamalar

Çözümünü yapacağım soru eski bir sınav sorusunun üzerine eklemeler yapılmış halidir. Söz konusu sınav 2016 yazının final sınavıdır ve aşağıdaki bağlantıdan ulaşılabilir:

https://drive.google.com/open?id=0B4pwZQIDTjQ4WVNQODVQcXZDREk

Bu soruda başlangıçta kaç tane kum tanesi olacağı ve kum tanelerinin ne taraftan ne tarafa akmaya başlayacağı belirtilmemiş. Ayrıca her saniyede bir kum tanesinin yer değiştirdiği söylenmiş.

Bu yazıda daha çok şey anlatabilmek için bu soruya bazı eklemeler yapıyorum. Bunları şu şekilde listeleyebiliriz:

1. Başlangıç konumunda kaç tane kum tanesi olacağını belirleyebilelim. (Bu değerin en fazla 8 olacağını varsayabiliriz. Çünkü toplam 16 LED var. Bu durumda bir yarıda en fazla 8 LED olduğundan başlangıçta en fazla 8 kum tanesi olabilir.)
2. Başlangıç akış yönünü belirleyebilelim. (Sağdan sola mı? Soldan sağa mı?)
3. Bir kum tanesinin kaç saniyede bir yer değiştirdiğini belirleyebilelim. (Bu değerin 1-15 arasında pozitif tam sayı olacağını varsayalım.)
4. Kum saatinin kaç saniye sonra boşalacağını 7 Segment Display aygıtları üzerinde gözlemleyebilelim.

   Not: Yazının geri kalan kısmında 7 Segment Display yerine kısaca 7SD yazacağım.

Çözüm Aşaması

“kumsaati” isimli bir modül oluşturup ve girdilerin ve çıktıların nasıl olması gerektiğine karar vermeye başlıyorum.

Bu modülde saat kullanacağımız çok açık. Kum saatinin akış yönünü değiştirmek için orta butonu kullanacağız. O halde birer bitlik 2 adet girdimiz var.

Kum saatini gösterebilmek için LED’lerin tamamını kullanacağız. Bunun için 16 bitlik tek bir çıktı yerine 8’er bitlik 2 çıktı kullanmayı tercih ediyorum sağ ve sol taraf üzerinde rahatça işlem yapabileyim diye. Kum saatinin ömrünü 7SD üzerinde göstereceğiz. 7SD üzerindeki nokta şeklindeki ışığı yakmayacağız. 7SD üzerindeki aygıtlar üzerinde aynı anda farklı değerler görmemiz gerekeceğinden (aslında teknik olarak mümkün değil ancak insan gözü için aynı anda diyebiliriz) 4 bitlik bir değere ihtiyacımız var.

Girdi ve çıktıların son hali aşağıdaki gibidir:

```
module kumsaati(
    input btn, clk,
    output [3:0] an,
    output dp,
    output [7:0] led_left, led_right,
    output [6:0] seg
    );
endmodule
```

Soruya yaptığımız eklemelerden sonra tasarlayacağımız modülün parametrik olması gerekmektedir. 3 adet parametreye ihtiyacımız var. Parametrelerin isimleri ve görevleri aşağıdaki gibidir:

1. **C:** Başlangıçtaki kum tanesi sayısı
2. **A:** Başlangıç akış yönü (0: sağdan sola, 1: soldan sağa)
3. **N:** Bir kum tanesinin yer değiştirmesi için geçen saniye

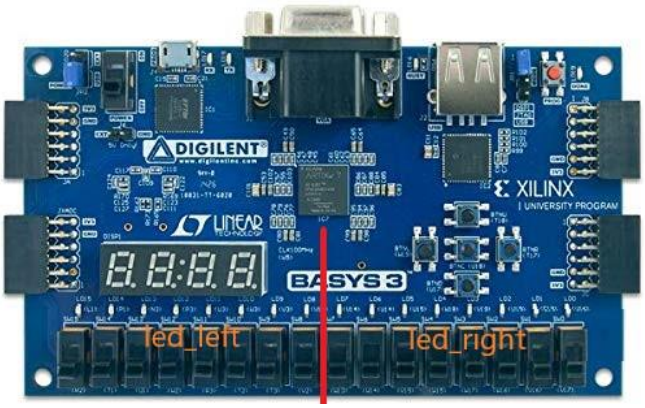
```
module kumsaati #(parameter C=4, A=0, N=2)(
    input btn, clk,
    output [3:0] an,
    output dp,
    output [7:0] led_left, led_right,
    output [6:0] seg
    );
endmodule
```

Kodumu 2 adet “always” bloğu kullanarak tamamlayacağım. Bu bloklardan birinde hafızalarımın (reg) bir sonraki saat darbesinde almaları gereken değerleri (next_value) hesaplarken saatin yükselen kenarında çalışacak olan blokta ise hesaplananmış değerleri ilgili hafızalara atayacağım. Şimdilik sağdaki ve soldaki LED’lerin değerlerini ve sonraki saat darbesinde alacakları değerleri tutmak için hafızalarımı oluşturuyorum.

```
reg [7:0] r_left, r_left_next;
reg [7:0] r_right, r_right_next;

always @* begin
    r_left_next = r_left;
    r_right_next = r_right;
end

always @(posedge clk) begin
    r_left <= r_left_next;
    r_right <= r_right_next;
end
```



Burada “r_left” ve “r_right” isimli hafızalar sırasıyla “led_left” ve “led_right” isimli çıktılarımızın alacağı değerleri tutuyor. “assign” anahtar kelimesi ile ilgili atamaları gerçekleştiriyor ve hafızalarımın ilk değerlerini hesaplıyorum. Bu hesaplamaları yaparken ‘C’ ve ‘A’ isimli parametreleri dikkate almam gerekiyor.

Önce “r_left” hafızasının ilk değerini hesaplayalım.

Eğer ‘A’ parametresi ‘0’ ise başlangıçta sağdan sola akış olacak demektir. Bu durumda sol taraftaki LED’lerin yanmasına gerek yoktur. “r_left” değeri ‘0’ olur. Eğer soldan sağa akış olacaksa, sol taraftaki ‘C’ parametresi kadar LED’in yanması gerekir (ortaya dayalı bir şekilde). Yani “C=6” ve “A=1” değerleri için “led_left” başlangıçta “8’b0011_1111” değerini almalı. Genel olarak bunu “concatenation” operatörünü kullanarak “{C{1’b1}}” olarak ifade edebiliriz.

Şimdi sırada “r_right” hafızasının ilk değerini hesaplamak var.

“r_left” in tersine ‘A’ parametresi ‘1’ değerini aldığında başlangıç akış yönü soldan sağa olacağından “r_right” hafızasının ilk değeri bu durumda ‘0’ olur. Çünkü sağ taraf boş olmalı. Eğer sağdan sola akış olacaksa sağ tarafta ortaya dayalı olarak ‘C’ parametresinin değeri kadar LED yanmalı. Yani “C=3” ve “A=0” için “r_right” başlangıçta “8’b1110_0000” olmalı. Genel olarak bunu “{8{1’b1} << (8 – C)}” olarak ifade edebiliriz.

LED'lerin ilk değer atamalarını yaptıktan sonra kodumuz şöyle oluyor:

```
reg [7:0] r_left = A == 0 ? 8'd0 : {C{1'b1}}, r_left_next;
reg [7:0] r_right = A == 0 ? {{8{1'b1}} << (8-C)} : 8'd0, r_right_next;

always @* begin
    r_left_next = r_left;
    r_right_next = r_right;
end

always @(posedge clk) begin
    r_left <= r_left_next;
    r_right <= r_right_next;
end

assign led_left = r_left;
assign led_right = r_right;
```

Kendime yeni bir hedef belirliyorum. Kum tanelerini hareket ettirmeye çalışıyorum. 'N' saniyede bir hareket etmeleri lazım. Bunun için saat yavaşlatmak gerekiyor. Bir sayaç oluşturuyorum.

Bunların yanında, programın akış yönü sabit olmayacak. Akış yönünü tutan bir hafıza oluşturuyorum. Artık '0' ve '1'lerle kum saatinin sağa ya da sola akacağını belirtmek yerine yerel parametreler (localparam) tanımlıyorum.

```
localparam LEFT = 1'b0;
localparam RIGHT = 1'b1;

reg [31:0] r_counter = 32'd0, r_counter_next;
reg r_flow = A, r_flow_next;
reg [7:0] r_left = A == LEFT ? 8'd0 : {C{1'b1}}, r_left_next;
reg [7:0] r_right = A == LEFT ? {{8{1'b1}} << (8-C)} : 8'd0, r_right_next;

always @* begin
    r_counter_next = r_counter + 1'b1;
    r_flow_next = r_flow;
    r_left_next = r_left;
    r_right_next = r_right;

    if (r_counter == 100_000_000 * N - 1) begin
        // N saniyede bir calisacak
    end
end

always @(posedge clk) begin
    r_counter <= r_counter_next;
    r_flow <= r_flow_next;
    r_left <= r_left_next;
    r_right <= r_right_next;
end
```

Yukarıdaki kodu biraz açıklayalım. Her saat darbesinin ardından hafızaların sonraki değerini hesaplamak için ilk "always" bloğu çalışıyor. FPGA'in saat sıklığı 100 MHz (saniyede 100 milyon tekrar) olduğundan ilk "always" bloğu saniyede 100 milyon kere tekrar edecektir. O halde "r_counter" isimli hafızam 0'dan başlayıp 99.999.999 olduğunda anlıyorum ki bir sonraki tekrarla birlikte toplam 1 saniyelik zaman geçmiş.

'N' parametresi '1' değerini aldığında "if (r_counter == 100_000_000 * N – 1)" değerini "if (r_counter == 99_999_999)" olarak düşünebiliriz. Bu "if" bloğunun içerisinde N=1 saniyede gerçekleşmesini istediğimiz işlemleri yapabiliriz.

Not: “-1” kafanızı karıştırmasın. İlk “always” bloğunda hafızaların sonraki değerlerini hesapladığımıza dikkat edin. Aslında “r_counter” hafızamız “100_000_000 * N - 1” değerine eşit olduğunda bir sonraki değerleri belirliyoruz ve sonraki saat darbesinde hafızalarımız güncellenmiş oluyor. Yani “100_000_000 * N” tekrarda bir gerçek hafızalarımız güncelleniyor.

Kum tanelerini hareket ettirme

Akış yönü soldan sağaysa LED’lerimizin sonraki değerlerini şöyle bulabiliriz.

Sol LED’imizin sonraki değerini sağ tarafa 1 kere kaydırarak bulabiliriz. Sağ LED’imizin sonraki değerini ise mevcut değerin en sağına (en anlamsız bit) ‘1’ ekleyerek (concatenation) bulabiliriz.

Akış ters yönde ise benzer durum söz konusu.

Bu sefer sağ LED’imizin mevcut değerini sola 1 kere kaydırarak sonraki değeri hesaplayabiliriz. Sol LED’in sonraki değeri ise mevcut değeri sağa 1 kere kaydırdıktan sonra en anlamlı biti ‘1’ olarak ayarladıktan (or operatörü ile) sonra elde edilebilir.

Eğer daha fazla kum tanesi hareket edemeyecekse LED’lerin eski değerlerini koruması lazım. Bunun için kontroller yapıyorum.

```
if (r_counter == 100_000_000 * N - 1) begin
    r_counter_next = 32'd0;
    if (r_flow == RIGHT) begin
        if (r_right[C-1] != 1'b1) begin
            r_left_next = r_left >> 1'b1;
            r_right_next = {r_right, 1'b1};
        end
    end
    else begin
        if (r_left[8-C] != 1'b1) begin
            r_left_next = (r_left >> 1'b1) | 8'b1000_0000;
            r_right_next = r_right << 1'b1;
        end
    end
end
```

Kaç saniye kaldı?

Bu zamana kadar başlangıç kum tanesi sayısı, akış yönü ve bir kum tanesinin hareketi için kaç saniye geçeceğini ayarlayabildiğimiz bir modülümüz var. Ayrıca, bu modül kum tanelerinin hareketini de sağlıyor. Ancak orta butona basıldığında tanelerin akış yönünü değiştirecek seviyede değil. Butonu en sona bırakıyorum ve 7SD üzerinde kalan saniyeleri göstermeyi kendime yeni hedef olarak belirliyorum.

Öncelikle “dp” çıktısına doğrudan ‘1’ atıyorum. Çünkü nokta şeklindeki ışığın yanmasını istemiyorum.

2,5 milisaniye boyunca 7SD aygıtlarından birisi yanarken diğerleri sönecek ve aygıtlar sırasıyla yanacak. Bu sayede insan gözü aygıtların hepsinin aynı anda yandığını ve farklı değerler gösterdiğini algılayacak.

Saniyede bir kere aygıt üzerinde gösterilen kalan saniye değerini 1 azaltmak gerekli.

Toplamda 2 sayaç kullanıyorum ki bu zamanlamaları kontrol edebileyim.

Ek olarak, kalan saniye değerini tutabilmek için hafıza oluşturuyorum ve ilk değerini “C * N” olarak belirliyorum. Ayrıca, “an” çıkışını belirlemek için de ilk değeri “1110” olan bir hafıza oluşturuyorum ve

bu hafızayı 2.5 milisaniyede bir kere dairesel olarak sola kaydırıyorum. Son güncellemelerle birlikte kod aşağıdaki gibi gelişiyor.

```
reg [3:0] r_an = 4'b1110, r_an_next;
reg [31:0] r_counter = 32'd0, r_counter_next;
reg [17:0] r_counter_an = 18'd0, r_counter_an_next;
reg [26:0] r_counter_sec = 27'd0, r_counter_sec_next;
reg r_flow = A, r_flow_next;
reg [7:0] r_left = A == LEFT ? 8'd0 : {C{1'b1}}, r_left_next;
reg [7:0] r_right = A == LEFT ? {{8{1'b1}} << (8-C)} : 8'd0, r_right_next;
reg [6:0] r_second = C * N, r_second_next;

always @* begin
    r_an_next = r_an;
    r_counter_next = r_counter + 1'b1;
    r_counter_an_next = r_counter_an + 1'b1;
    r_counter_sec_next = r_counter_sec + 1'b1;
    r_flow_next = r_flow;
    r_left_next = r_left;
    r_right_next = r_right;
    r_second_next = r_second;

    if (r_counter_an == 18'd249_999) begin
        r_counter_an_next = 18'd0;
        r_an_next = {r_an[2:0], r_an[3]};
    end

    if (r_counter_sec_next == 27'd99_999_999) begin
        r_counter_sec_next = 27'd0;
        if (r_second != 7'd0) begin
            r_second_next = r_second - 1'b1;
        end
    end
```

```
always @(posedge clk) begin
    r_an <= r_an_next;
    r_counter <= r_counter_next;
    r_counter_an <= r_counter_an_next;
    r_counter_sec <= r_counter_sec_next;
    r_flow <= r_flow_next;
    r_left <= r_left_next;
    r_right <= r_right_next;
    r_second <= r_second_next;
end

assign an = r_an;
assign dp = 1'b1;
assign led_left = r_left;
assign led_right = r_right;
```

Şimdi ise “r_an” hafızasına göre “seg” çıkışının değerini belirlemeye çalışacağım. Bunun için “BCDto7SD” isimli hazır modülden yararlanıyorum. Modül girdi olarak BCD (Binary Coded Decimal) sayı alıyor ve çıktı olarak 7 bitlik bir değer veriyor. Bu değer BCD sayının 7SD üzerindeki gösterimine karşılık geliyor.

İlk değeri ‘0’ olan “r_bcd” adında bir hafıza söz konusu modülün girişine bağlanmak üzere oluşturuluyor. Modülün çıkışına ise doğrudan “seg” çıktısı bağlanıyor.

```
BCDto7SD (r_bcd, seg);
```

Hangi aygıtın yanacağını kontrol ettikten sonra “r_bcd” değerini ilk “always” bloğu içerisinde güncelliyoruz. “r_an” hafızası “4’b1110” değerine sahipken en sağdaki aygıt yanacaktır. Bu durumda kalan saniye değerinin (r_second) birler basamağı yanmalı. “r_second” değeri en fazla 3 basamaklı olacağından (çünkü 8*15=120, c parametresi en fazla 8, n parametresi en fazla 15) en soldaki aygıt yanacağı zaman doğrudan ‘0’ değerini sürüyorum.

```
    case (r_an)
        4'b1110: r_bcd = r_second % 10;
        4'b1101: r_bcd = (r_second / 10) % 10;
        4'b1011: r_bcd = r_second / 100;
        4'b0111: r_bcd = 4'd0;
    endcase
```

Kum saatini çevirme

Şu aşamaya kadar orta butona basarak kum saatinin akış yönünü değiştirme dışında her şeyi yapan bir modül var.

Öncelikle butonumuzun ideal olması lazım. Bunun için Piazza'da paylaşılmış olan "buttonResponse" isimli modülden faydalanıyorum. Bu modülün çıktısını bir kabloya bağlayarak butona basıldı mı diye kontrol edebilirim.

```
wire flag_btn;

BCDto7SD (r_bcd, seg);
buttonResponse(btn, clk, flag_btn);
```

Butona basıldığında kalan saniyeyi ve LED'leri güncellemem gerek. Şu aşamada kalan saniyeyi elimdeki hafızaları kullanarak güncelleyemeyeceğimi fark ettim. Başlangıç konumundan itibaren belirli bir süre geçtikten sonra butona basıldığında bu değeri güncelleyebilmem için toplam kaç kum tanesinin hareket ettiği bilgisini biliyor olmam gerek. Bunun için bir çözüm geliştiriyorum ve sağ taraftaki kum tanelerinin sayısını tutmak için bir hafıza oluşturuyorum. Bu hafızanın ilk değerini belirledikten sonra tanelerin hareket etmesiyle artacak ya da azalacak şekilde kodun gerekli yerlerinde düzenlemeler yapıyorum.

```
reg [3:0] r_counter_right = A == RIGHT ? 4'd0 : C, r_counter_right_next;

always @* begin
    r_an_next = r_an;
    r_counter_next = r_counter + 1'b1;
    r_counter_an_next = r_counter_an + 1'b1;
    r_counter_sec_next = r_counter_sec + 1'b1;
    r_counter_right_next = r_counter_right;
    r_flow_next = r_flow;
    r_left_next = r_left;
    r_right_next = r_right;
    r_second_next = r_second;

    if (r_counter == 100_000_000 * N - 1) begin
        r_counter_next = 32'd0;
        if (r_flow == RIGHT) begin
            if (r_right[C-1] != 1'b1) begin
                r_counter_right_next = r_counter_right + 1'b1;
                r_left_next = r_left >> 1'b1;
                r_right_next = {r_right, 1'b1};
            end
        end
        else begin
            if (r_left[8-C] != 1'b1) begin
                r_counter_right_next = r_counter_right - 1'b1;
                r_left_next = (r_left >> 1'b1) | 8'b1000_0000;
                r_right_next = r_right << 1'b1;
            end
        end
    end
```

```
always @(posedge clk) begin
    r_an <= r_an_next;
    r_counter <= r_counter_next;
    r_counter_an <= r_counter_an_next;
    r_counter_sec <= r_counter_sec_next;
    r_counter_right <= r_counter_right_next;
    r_flow <= r_flow_next;
    r_left <= r_left_next;
    r_right <= r_right_next;
    r_second <= r_second_next;
end
```

Artık butona basıldığı anda sağ taraftaki kum tanesi sayısına sahibim. O halde butona basıldığında eğer akış sağ tarafa doğruysa, yeni akış tarafı sola doğru olacaktır ki bu durumda sağ taraftaki kum tanelerinin yer değiştirmesi için gereken saniye kalan saniyeye eşit olur. Akış aksi yönde olsa dahi aynı bilgi kullanılarak kalan saniye hesaplanabilir.

İlk “always” bloğuna yazıyorum:

```
if (flag_btn) begin
    r_counter_next = 32'd0;
    r_counter_sec_next = 27'd0;
    r_flow_next = ~r_flow;
    if (r_flow == RIGHT) begin
        r_second_next = r_counter_right * N;
    end
    else begin
        r_second_next = (C - r_counter_right) * N;
    end
end
```

Tek bir şey eksik. LED’leri kontrol eden hafızaların bitlerin ayna görüntülerini almak. Örneğin akışın sol tarafa olduğunu düşünelim.

“r_left” hafızası “1110_0000” iken ve “r_right” hafızası “1100_0000” iken butona basıldığında yeni değerler sırasıyla “0000_0111” ve “0000_0011” olacaktır.

Sağ taraftaki kum tanesi sayısını (r_counter_right) kullanarak yeni değerleri hesaplayabiliriz. Ancak derste gösterilmeyen yeni bir şey göstermek istiyorum.

“yeni” isminde bir hafızamız olsun ve değeri “001101” olsun. “yeni = {<<{yeni}}” ifadesinin ardından hafızamızın yeni değeri “101100” olur.

LED’lerimizin yeni değerlerini bu şekilde hesaplıyorum. Detaya girmeyeceğim. Sınavda böyle yapmak zorunda değilsiniz. Ek bilgi olması için çözümü bu şekilde yapıyorum. Ayrıntılı bilgi için aşağıdaki bağlantıyı takip edebilirsiniz:

https://www.amiq.com/consulting/2017/05/29/how-to-pack-data-using-systemverilog-streaming-operators/

```
if (flag_btn) begin
    r_counter_next = 32'd0;
    r_counter_sec_next = 27'd0;
    r_flow_next = ~r_flow;
    if (r_flow == RIGHT) begin
        r_second_next = r_counter_right * N;
    end
    else begin
        r_second_next = (C - r_counter_right) * N;
    end
    r_left_next = {<<{r_left}};
    r_right_next = {<<{r_right}};
end
```

Bu hamleyle birlikte parametrik modülün tasarlanmasını tamamlıyorum.

Gösterim

Parametrik “kumsaati” modülünü “C=6, A=1, N=2” parametreleriyle kullanıyorum.

```
module FPGAkumsaati(
    input btn, clk,
    output [3:0] an,
    output dp,
    output [15:0] led,
    output [6:0] seg
    );

    kumsaati #(.C(6), .A(1), .N(2)) saat(
    .btn(btn),
    .clk(clk),
    .an(an),
    .dp(dp),
    .led_left(led[15:8]),
    .led_right(led[7:0]),
    .seg(seg)
    );
```

“Constraint” dosyası hazırayarak FPGA kartı üzerinde başlangıç kum tanesi sayısı 6, başlangıçta soldan sağa akan ve kum tanesi değişim hızı 2 saniye olan bir kum tanesi modelini gözlemliyorum.