VERİLOG İLE PARAMETRİK TARASARIM

1. PARAMETRİK TASARIM

Verilog modüllerini parametrik tasarlayarak farklı bit genişlikleri için aynı işi yapan farklı modüller tasarlama zahmetinden kurtulabiliriz. Örneğin; 3 bitlik sayılar yerine 4 bitlik sayıları toplayan bir toplayıcı modüle ihtiyaç duymamız durumunda, 4 bitlik sayıları toplayan yeni bir modül gerçekleştirmemiz gerekir. Fakat parametrik bit genişliğine sahip bir toplayıcı modül gerçekleştirerek, ihtiyaç duyulan bit genişliğindeki sayıları toplayan toplayıcılar oluşturabiliriz (bir modülün başka bir modül içinde nasıl kullanılacağını hatırlamak için 3. laboratuvar ön çalışmasına bakabilirsiniz).

Parametrik toplayıcı modül Verilog ile davranışsal modelleme ile aşağıdaki gibi kodlanabilir.

```
module AdderNBit #(parameter N = 8) (
        input[N - 1:0] num1,
        input[N - 1:0] num2,
        output reg[N:0] sum);

        always@(num1 or num2) begin
                sum = num1 + num2;
        end
endmodule
```

Parametrik modüller gerçekleştirmek için, kullanacağınız parametreleri modül isminden sonra "#(...)" şeklinde tanımlamanız gerekir. Yukarıdaki örnekte "N" isminde bir parametre tanımlanmış olup bunun varsayılan (default) değeri 8 olarak belirlenmiş.tir Birden fazla parametre tanımlarken "," ile yeni bir parametre ismi ve varsayılan değerini yazabilirsiniz ("parameter N = 8, M = 6" gibi). Her parametre için **varsayılan değer** atamanız zorunludur. "AdderNBit" modülünün örnekleri (instance) oluşturulurken "N" parametresi değiştirilerek farklı bit genişliklerindeki sayıları toplayan toplayıcılar elde edilebilir.

Daha önce gerçekleştirdiğiniz "AdderLeds_behavioral" modülünü parametrik şekilde aşağıdaki gibi kodlayabilirsiniz.

```
module AdderLeds_param #(parameter ADDER_WIDTH = 10) (
        input[ADDER_WIDTH - 1:0] val1,
        input[ADDER_WIDTH - 1:0] val2,
        output reg[LEDS_WIDTH - 1:0] leds);

        localparam LEDS_WIDTH = (1 << (ADDER_WIDTH + 1)) – 2;

        wire[ADDER_WIDTH:0] sum;
        AdderNBit #(.N(ADDER_WIDTH)) adder(
                .num1(val1),
                .num2(val2),
                .sum(sum)
        );
        always@* begin
                leds = (1 << sum) – 1;
        end
endmodule
```

"AdderLeds_param" modülünün tek parametresine "ADDER_WIDTH" ismini verdik. Bu parametre, toplanacak sayıların bit genişliği olarak kullanılıyor. Çıkış ("leds") için gerekli bit genişliği "LEDS_WIDTH" yerel parametresi ("**localparam**") ile gösterilmiştir. "localparam" ile sadece o modül içinde kullanılacak parametreleri tanımlıyoruz. Bir modülü içerisinde birden fazla kullanacağınız sayı değerlerinizi localparam ile tanımlayarak okunması daha kolay bir kod yazabilirsiniz. Modülün örneğini (instance) oluştururken "localparam" olan değerleri değiştiremezsiniz (Fakat "localparam" değerler parametrelere bağlı olarak farklı şekilde hesaplanabilir. Örneğin; yukarıdaki örnekte "ADDER_WIDTH" 5 olduğunda "LEDS_WIDTH" 62, "ADDER_WIDTH" 3 olduğunda ise 14 olur).

Girişlerden gelen "ADDER_WIDTH" bitlik sayıları toplayabilmek için, "ADDER_WIDTH" genişliğinde sayıları toplayan toplayıcıya ihtiyacımız var. Bu toplayıcıyı parametrik olarak gerçekleştirdiğimiz "AdderNBit" modülünü kullanarak oluşturuyoruz. Parametrik modüllerin parametrelerinin değerlerini belirlemek için modül isminden önce "*#(.<PARAMETRE_ISMI1>(<DEGER2>), .<PARAMETRE_ISMI2>(<DEGER1>))*" şeklinde bir kod parçası yazıyoruz. Yukarıdaki örnekte, "AdderNBit" modülünün örneğini oluştururken, bu modülün "N" parametresine "ADDER_WIDTH" değeri verilmiş.

"leds" çıkışı artık sabit bir uzunlukta olmadığından *if-else* veya *case* yapıları kullanarak herhangi bir bit genişliğindeki "sum" sinyali için "leds" çıkışının ne olması gerektiğini kodlayamıyoruz. Bunun yerine, *if-else* ve *case* yapılarının gerçekleştirdiği işlemi formülize etmemiz gerekiyor. "*(1 << sum) – 1*" şeklindeki kod parçası "sum" değeri kadar biti "leds" çıkışının en sağından başlayarak mantık-1 yapmaktadır. Örneğin; "sum = 5" için,

leds = (1 << 5) – 1 = 100000 – 1 = 011111 olmaktadır.