Always em Verilog

Always

O bloco `always` em Verilog é uma das estruturas mais importantes para descrever o comportamento de circuitos digitais. Ele permite que você defina um bloco de código que será executado sempre que ocorrer uma mudança em seus sinais de entrada ou quando uma condição específica for atendida. Um bloco `always` pode ser usado de forma combinacional ou sequencial, dependendo de como você o define.

Always Combinacional

```
always @(entrada) begin
    saida = entrada + 1; // sempre que a entrada mudar, a
        sa da ser  a entrada mais 1
end
```

Geralmente, para lógica combinacional, você utiliza o bloco `always` com `@(*)`. Isso garante que o bloco seja reavaliado sempre que qualquer sinal que está sendo atribuído dentro do bloco mudar. A sintaxe básica é:

```
always @(*) begin
    saida = entrada1 + entrada2; // sempre que qualquer
        sinal de entrada mudar, a sa da ser  atualizada
end
```

Ao utilizar dessa forma, qualquer sinal que esteja dentro do bloco `always` e do lado *direito* da atribuição será considerado no `always`, e o bloco irá rodar sempre que qualquer um desses sinais mudar. No exemplo acima, a saída será atualizada sempre que `entrada1` ou `entrada2` mudar.

Always Sequencial

```
always @(posedge clk) begin
    saida <= entrada; // sempre que houver uma borda de
        subida no clock, a sa da ser  atualizada com o
        valor da entrada
end
```

Neste caso, o bloco `always` é sensível à borda de subida do sinal de clock (`clk`), e a saída (`saida`) será atualizada com o valor da entrada (`entrada`) sempre que ocorrer uma transição de subida no clock.

Atribuições Não Bloqueantes

O uso de `<=` quer dizer que a atribuição é feita de forma **não bloqueante**, o que significa que todas as atribuições dentro do bloco serão feitas ao final do ciclo de clock, permitindo que outras operações ocorram antes que a saída seja atualizada:

```
always @(posedge clk) begin
    a <= b; // atribui  o  n  o  bloqueante
    b <= a; // outra  atribui  o  n  o  bloqueante
end
```

Neste exemplo, `a` e `b` serão atualizados ao final do ciclo de clock, permitindo que ambas as atribuições sejam feitas sem interferência mútua. `a` recebe o valor antigo de `b` e `b` recebe o valor antigo de `a`. A **ordem** da atribuição não importa, pois ambas são feitas ao final do ciclo de clock.

Como Verilog é uma linguagem de descrição de hardware, em circuitos sequenciais é recomendado utilizar sempre atribuições não bloqueantes para evitar problemas de sincronização.