Alvarado Lomeli Gael Ramses

Mini reporte: 04 Participación - [MUXO] Modulos OP(x6) & Mux

En esta ocasión, este reporte tratara de lo aprendido en la clase, la clase fue acerca de la ALU con bloqueo y sin bloqueo, primero explicare lo que entendí sobre la ALU con bloqueo, que básicamente las instrucciones que da la ALU se procesan de forma secuencial, esto significa que espera a que termine una instrucción para poder continuar con la siguiente instrucción. Ahora, el ALU sin bloque es lo contrario a lo que es el ALU con bloqueo, ya que aquí las instrucciones se procesan al mismo tiempo, esto hace que sea más rápido.

Este es el código del ALU con bloqueo:

```
1
2   module AluB(
3       input [31:0] A,
4       input [31:0] B,
5       input CLK,
6       input [2:0] op,
7       output reg [31:0] Res,
8       output reg Zflag
9   );
10
11  always @(posedge CLK) begin
12      case(op)
13          3'b000: Res = A + B;      // Suma
14          3'b001: Res = A & B;      // AND
15          3'b010: Res = A | B;      // OR
16          3'b011: Res = A - B;      // Resta
17          3'b100: Res = A * B;      // Multiplicación
18          3'b111: Res = (A < B) ? 32'd1 : 32'd0;  // Ternaria
19
20          default: Res = 32'd0;
21      endcase
22
23      Zflag = (Res == 32'd0) ? 1'b1 : 1'b0;
24  end
25
26  endmodule
```

Para este código, tenemos las siguientes entradas: A, B, CLK, para el reloj, y la op. Y para las salidas definimos: Res, Zflag y solo eso. El Zflag nos avisara cuando Res este en 0. Para este código usamos un bloque always, y utilizamos posedge, esto es para una curva ascendente del reloj, despues definimos los casos, y no olvidar el default.

```
module AluNB(
    input [31:0] A,
    input [31:0] B,
    input CLK,
    input [2:0] op,
    output reg [31:0] Res,
    output reg Zflag
);

initial begin
    Res <= 32'd0;
    Zflag <= 1'b0;
end

always @(posedge CLK or op) begin
    case (op)
        3'b000: Res <= A + B;
        3'b001: Res <= A & B;
        3'b010: Res <= A | B;
        3'b011: Res <= A - B;
        3'b100: Res <= A * B;
        3'b101: Res <= (A < B) ? 32'd1 : 32'd0;
        default: Res <= 32'd0;
    endcase
```

```

     if (Res == 32'd0)
         Zflag <= 1'b1;
     else
         Zflag <= 1'b0;
end

endmodule
```

Para el código sin bloqueo, las variables son las mismas, la diferencia más grande es el utilizar <= en vez de =, esto para que las instrucciones no esperen a la anterior.

Y para terminar usamos un if para checar que Res sea 0 y modificar los valores de Zflag.

Ahora sigue el testbench para probar ambos códigos:

```
module ALU_TB();
    reg [31:0] ATB, BTB;
    reg CLKTB;
    reg [2:0] SELTB;
    wire [31:0] RTB_NB, RTB_B;
    wire ZFLAGTB_NB, ZFLAGTB_B;


    // Instanciamos la ALU con bloqueo (ALUB)
    AluB alu_b ( .A(ATB), .B(BTB), .op(SELTB), .Res(RTB_B), .CLK(CLKTB), .Zflag(ZFLAGTB_B));

    // Instanciamos la ALU sin bloqueo (ALUNB)
    AluNB alu_nb ( .A(ATB), .B(BTB), .op(SELTB), .Res(RTB_NB), .CLK(CLKTB), .Zflag(ZFLAGTB_NB));


    always #50 CLKTB = ~CLKTB;

    initial begin
        CLKTB = 0; // Inicializa el reloj
        ATB = 32'd300;
        BTB = 32'd100;

        SELTB = 3'b000; #100;  // Suma
        SELTB = 3'b001; #100;  // AND
        SELTB = 3'b010; #100;  // OR
        SELTB = 3'b011; #100;  // Resta
        SELTB = 3'b100; #100;  // Multiplicacion
        SELTB = 3'b101; #100;  //Ternaria
```

```
        $stop;
    end
endmodule
```

Para el testbench hicimos los registros necesarios, los wire e instanciamos ambos módulos. Ahora le indicamos un tiempo específico a el reloj para que cambie de estado y agregamos initial para agregar los valores.

Estos fueron los resultados: