Participación 5

DIEGO JARED JIMENEZ SILVA

En esta actividad vimos el tema de la ALU sin bloqueo y con bloqueo, revisamos, investigamos y comprendimos la diferenciación de cada uno para posteriormente hacer la implementación dentro de MATLAB. La ALU con bloqueo hace referencia a que las instrucciones que se le dan a la ALU se procesan de forma secuencial esperando a que la instrucción anterior se termine, de esta forma se va haciendo operación por operación en orden. La ALU sin bloqueo es todo lo contrario, aquí se procesan las instrucciones al mismo tiempo o en simultaneo, aplicando el pipeling que antes habíamos visto, de esta forma es más rápido y eficiente.

A continuación, empezamos con la ALU bloqueo:

```
//ALU con bloqueo
module ALUB(
    input [31:0] A,
    input [31:0] B,
    input CLK,
    input [2:0] op,
    output reg [31:0] Res,
    output reg Zflag
);

// Utilizamos posedge que se refiere a la curva ascendente del reloj
always @(posedge CLK) begin
    case(op)
        3'b000: Res = A + B;      // Suma
        3'b001: Res = A & B;      // AND
        3'b010: Res = A | B;      // OR
        3'b011: Res = A - B;      // Resta
        3'b100: Res = A * B;      // Multiplicación
        3'b101: Res = A < B ? 32'd1 : 32'd0;  // Ternaria

        default: Res = 32'd0; // Caso por defecto
    endcase


    Zflag = (Res == 32'd0) ? 1'b1 : 1'b0;
end

endmodule
```

Definimos las entradas A, B y el reloj CLK, además de la op. Como salidas tenemos Res y Zflag, este último nos indicará cuando Res valga cero. Usamos un bloque donde utilizamos posedge que se refiere a la curva ascendente del reloj y definimos todos los casos, además de agregar un default.

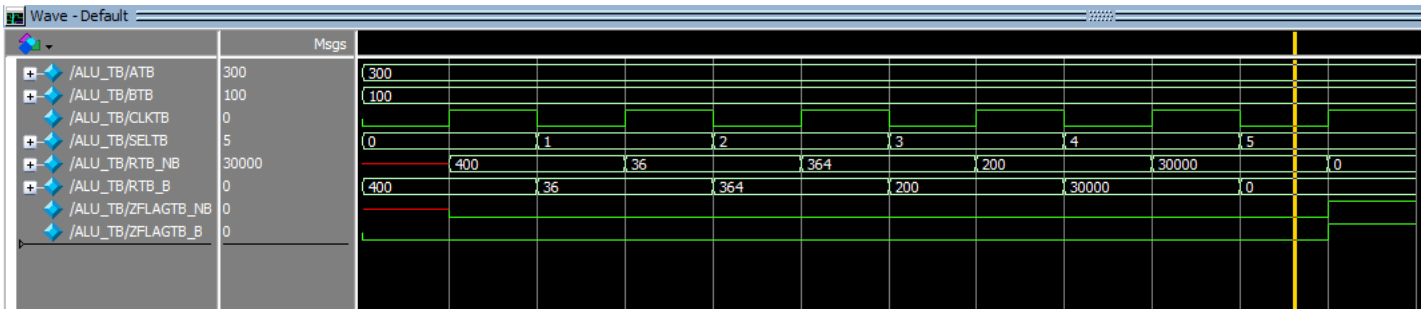
```
//Alu sin bloqueo
module ALUNB(
    input [31:0] A,
    input [31:0] B,
    input CLK,
    input [2:0] op,
    output reg [31:0] Res,
    output reg Zflag
);

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
```

De una manera similar hacemos el modulo sin bloqueo, definimos las mimas variables, inicializamos valores, utilizamos op y CLK en este caso dentro de los parámetros de always y definimos todos los casos con la diferencia de utiliza <= lo cual describe que la instrucción no debe esperar a que finalice la anterior. Finalmente usamos un if else para revisar si Res es cero y cambiar los valores de Zflag.

Probamos todo dentro de un testbench:

```
// Probamos ambas ALU
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;

        // Pruebas con diferentes operaciones
        SELTB = 3'b000; #100;  // Suma
        SELTB = 3'b001; #100;  // AND
        SELTB = 3'b010; #100;  // OR
        SELTB = 3'b011; #100;  // Resta
        SELTB = 3'b100; #100;  // Multiplicación
        SELTB = 3'b101; #100;  //Ternaria


        $stop;
    end
endmodule
```

Aquí utilizamos los registros correspondientes, los cables. Posteriormente instanciamos ambos módulos. Definimos un tiempo para que el reloj cambie de estado y utilizamos el bloque initial para inicializar los valores de entrada en cada caso.



Vemos que los resultados son los mismo para las dos ALU pero sabemos que la ejecución de instrucciones será más rápida con la ALU sin bloqueo.