



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Arquitectura de Computadoras

“Diseño de una ALU”

Alumno:

Malagón Baeza Alan Adrian

Profesor:

Alemán Arce Miguel Ángel

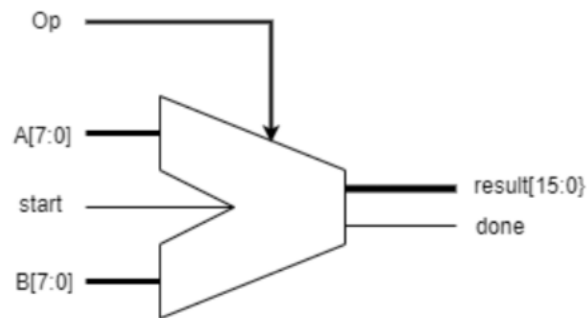
Grupo: 5CV1

Introducción

Objetivo: Diseñar una ALU con verilog que realice 6 operaciones en total, 3 aritméticas (deberá incluir una multiplicación) y 3 Lógicas.

Proponga su código de operación por función. Incluya operación de RESET y operación NOP.

Considerar una señal de entrada de start (inicio de operación) y una señal de done (de que se llevó a cabo una operación).



Desarrollo

Código Verilog

```
ALU.v x ALU_tb.v x ALU_tb_behav.wcfg x
C:/Users/alanm/Documents/GitHub/ArquitecturaDeComputadoras/Vivado/DiseñoALU/DiseñoALU.srcs/sources_1/new/ALU.v

1 | `timescale 1ns / 1ps
2 |
3 | module ALU(
4 |     input [7:0] A,
5 |     input [7:0] B,
6 |     input clk,
7 |     input [2:0] op,
8 |     input reset,
9 |     input start,
10 |    output reg done,
11 |    output reg [15:0] result
12 | );

14 | always @(posedge clk)
15 | begin
16 |     if (reset) begin
17 |         result <= 16'b0;
18 |         done <= 0;
19 |     end
20 |     else if (start) begin
21 |         case (op)
22 |             3'b000: begin // Operación lógica AND
23 |                 result <= A & B;
24 |             end
25 |             3'b001: begin // Operación lógica OR
26 |                 result <= A | B;
27 |             end
28 |             3'b010: begin // Operación lógica XOR
29 |                 result <= A ^ B;
30 |             end
31 |             3'b011: begin // Operación aritmética suma
32 |                 result <= A + B;
33 |             end
34 |             3'b100: begin // Operación aritmética resta
35 |                 result <= A - B;
36 |             end
37 |             3'b101: begin // Operación aritmética multiplicación
38 |                 result <= A * B;
39 |             end
40 |             default: begin // Operación NOP
41 |                 result <= 16'bX;
42 |             end
43 |         endcase
44 |         done <= 1;
45 |     end
46 |     else begin
47 |         done <= 0;
48 |     end
49 | end
50 | endmodule
```

Este código describe un módulo Verilog llamado "ALU" que representa una Unidad Aritmético-Lógica (ALU, por sus siglas en inglés). Una ALU es un componente en un procesador que se encarga de realizar operaciones aritméticas y lógicas en datos.

El módulo "ALU" tiene las siguientes entradas:

- `A` y `B`: Son buses de 8 bits que representan los operandos de las operaciones que se realizarán.
- `clk`: Es la señal de reloj utilizada para sincronizar las operaciones en el módulo.
- `op`: Es un bus de 3 bits que indica el tipo de operación que se debe realizar. Los valores posibles están definidos en el código.
- `reset`: Es una señal de reinicio que, cuando se activa, restablece el resultado a cero y establece el indicador de finalización en cero.
- `start`: Es una señal de inicio que, cuando se activa, inicia la operación de la ALU.

El módulo tiene dos salidas:

- `done`: Es un indicador de finalización que se establece en 1 cuando la operación ha terminado y se establece en 0 en otros momentos.
- `result`: Es un bus de 16 bits que representa el resultado de la operación realizada por la ALU.

El comportamiento del módulo está definido por un bloque `always` sensibilizado al flanco de subida del reloj (`posedge clk`). Dentro de este bloque, se utilizan declaraciones condicionales (`if-else`) para determinar el comportamiento en diferentes situaciones.

- Si `reset` es activado, se restablece el resultado a cero (`16'b0`) y se establece `done` en cero.
- Si `start` es activado, se realiza una operación dependiendo del valor de `op`. El resultado se asigna a `result` y se establece `done` en 1.

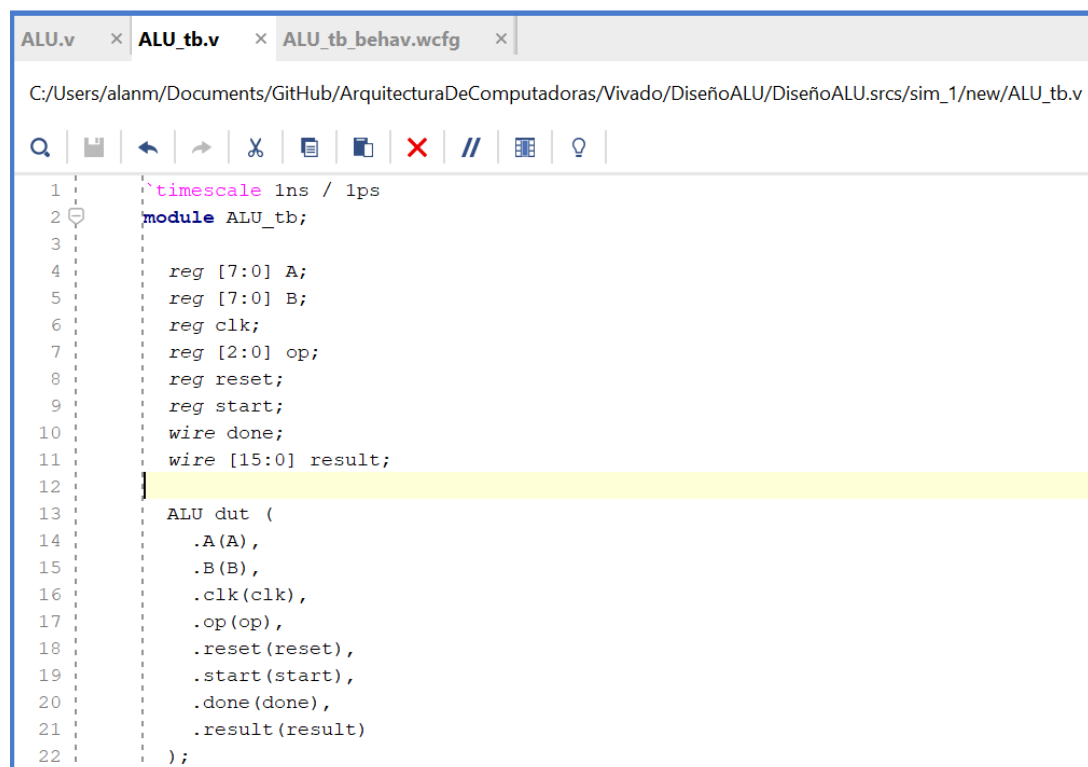
- Si no se cumple ninguna de las condiciones anteriores, se establece `done` en cero.

Las operaciones realizadas en la ALU dependen del valor de `op`. Los comentarios dentro del código indican qué operación se realiza en cada caso. Por ejemplo, si `op` es `3'b011`, se realiza la operación de suma (`result <= A + B`).

El caso `default` se utiliza cuando `op` no coincide con ninguna de las opciones anteriores, y en ese caso se establece `result` en `16'bX`, lo que indica un valor desconocido.

En resumen, este código describe una ALU en Verilog que realiza operaciones aritméticas y lógicas según las entradas proporcionadas. El resultado de la operación se almacena en `result` y se indica la finalización de la operación a través de `done`.

Código Verilog Testbench



```
1 | `timescale 1ns / 1ps
2 | module ALU_tb;
3 |
4 |     reg [7:0] A;
5 |     reg [7:0] B;
6 |     reg clk;
7 |     reg [2:0] op;
8 |     reg reset;
9 |     reg start;
10 |     wire done;
11 |     wire [15:0] result;
12 |
13 |     ALU dut (
14 |         .A(A),
15 |         .B(B),
16 |         .clk(clk),
17 |         .op(op),
18 |         .reset(reset),
19 |         .start(start),
20 |         .done(done),
21 |         .result(result)
22 |     );
```

```
ALU.v x ALU_tb.v x ALU_tb_behav.wcfg x
C:/Users/alanm/Documents/GitHub/ArquitecturaDeComputadoras/Vivado/DiseñoALU/DiseñoALU.srscs/sim_1/new/ALU_tb.v

22 | );
23 |
24 | initial begin
25 |     $monitor("Time=%t, A=%h, B=%h, op=%b, result=%h, done=%b", $time, A, B, op, result, done);
26 |
27 |     // Test case 1: Suma
28 |     A = 8'h0A;
29 |     B = 8'h05;
30 |     op = 3'b000; // Suma
31 |     reset = 0;
32 |     start = 1;
33 |     #10;
34 |     start = 0;
35 |     #20;
36 |
37 |     // Test case 2: Resta
38 |     A = 8'h0A;
39 |     B = 8'h05;
40 |     op = 3'b001; // Resta
41 |     reset = 0;
42 |     start = 1;
43 |     #10;
44 |     start = 0;
45 |     #20;
46 |
47 |     // Test case 3: Multiplicación
48 |     A = 8'h0A;
49 |     B = 8'h05;
50 |     op = 3'b010; // Multiplicación
51 |     reset = 0;
52 |     start = 1;
53 |     #10;
54 |     start = 0;
55 |     #20;
56 |
57 |     // Test case 4: Operación lógica AND
58 |     A = 8'h0A;
59 |     B = 8'h05;
60 |     op = 3'b011; // Operación lógica AND
61 |     reset = 0;
62 |     start = 1;
63 |     #10;
64 |     start = 0;
65 |     #20;
66 |
```

```
ALU.v x ALU_tb.v x ALU_tb_behav.wcfg x
C:/Users/alanm/Documents/GitHub/ArquitecturaDeComputadoras/Vivado/DiseñoALU/DiseñoALU.srscs/sim_1/new/ALU_tb.v

66
67 // Test case 5: Operación lógica OR
68 A = 8'h0A;
69 B = 8'h05;
70 op = 3'b100; // Operación lógica OR
71 reset = 0;
72 start = 1;
73 #10;
74 start = 0;
75 #20;
76
77 // Test case 6: Operación lógica XOR
78 A = 8'h0A;
79 B = 8'h05;
80 op = 3'b101; // Operación lógica XOR
81 reset = 0;
82 start = 1;
83 #10;
84 start = 0;
85 #20;
86
87 // Test case 7: Operación NOP
88 A = 8'h0A;
89 B = 8'h05;
90 op = 3'b110; // Operación NOP (no existe)
91 reset = 0;
92 start = 1;
93 #10;
94 start = 0;
95 #20;
96
97 // Test case 7: Operación NOP
98 A = 8'h0A;
99 B = 8'h05;
100 op = 3'b110; // Operación NOP (no existe)
101 reset = 1;
102 start = 1;
103 #10;
104 start = 0;
105 #20;
106
107 $finish;
108 end
109
```

```

110 always begin
111     #5 clk = ~clk;
112 end
113
114 initial begin
115     clk = 0;
116     #5;
117     forever #10 clk = ~clk;
118 end
119
120 endmodule
121

```

```

Time=          0, A=0a, B=05, op=000, result=xxxx, done=x
Time=        5000, A=0a, B=05, op=000, result=0000, done=1
Time=       15000, A=0a, B=05, op=000, result=0000, done=0
Time=       30000, A=0a, B=05, op=001, result=0000, done=0
Time=       35000, A=0a, B=05, op=001, result=000f, done=1
Time=       40000, A=0a, B=05, op=001, result=000f, done=0
Time=       60000, A=0a, B=05, op=010, result=000f, done=1
Time=       75000, A=0a, B=05, op=010, result=000f, done=0
Time=       90000, A=0a, B=05, op=011, result=000f, done=0
Time=       95000, A=0a, B=05, op=011, result=000f, done=1
Time=      100000, A=0a, B=05, op=011, result=000f, done=0
Time=      120000, A=0a, B=05, op=100, result=0005, done=1
Time=      135000, A=0a, B=05, op=100, result=0005, done=0
Time=      150000, A=0a, B=05, op=101, result=0005, done=0
Time=      155000, A=0a, B=05, op=101, result=0032, done=1
Time=      160000, A=0a, B=05, op=101, result=0032, done=0
Time=      180000, A=0a, B=05, op=110, result=xxxx, done=1
Time=      195000, A=0a, B=05, op=110, result=xxxx, done=0
Time=      215000, A=0a, B=05, op=110, result=0000, done=0

```

Este código describe un banco de pruebas (testbench) en Verilog para el módulo `ALU` que se mencionó anteriormente. Un testbench se utiliza para verificar el comportamiento y la funcionalidad de un módulo.

El testbench tiene las siguientes declaraciones y conexiones:

- Se declaran varias señales `reg` para representar las entradas del módulo `ALU`, incluyendo `A`, `B`, `clk`, `op`, `reset` y `start`.
- Se declaran señales `wire` para representar las salidas `done` y `result` del módulo `ALU`.
- Se instancia el módulo `ALU` con las conexiones adecuadas entre las señales declaradas y los puertos del módulo.

- Se utiliza la directiva ``$monitor`` para imprimir los valores de las señales en el tiempo. Esta línea imprimirá el tiempo de simulación, los valores de ``A``, ``B``, ``op``, ``result``, y ``done`` en formato hexadecimal y binario.

A continuación, se definen varios casos de prueba utilizando la sintaxis ``initial begin``. Cada caso de prueba se compone de las siguientes acciones:

- Se establecen los valores de ``A``, ``B``, ``op``, ``reset`` y ``start`` para configurar el caso de prueba.
- Se esperan algunos ciclos de reloj (``#10``) antes de cambiar el valor de ``start`` para activar la operación en el módulo ``ALU``.
- Después de desactivar ``start``, se esperan más ciclos de reloj (``#20``) para permitir que la operación se complete en el módulo ``ALU``.

Los casos de prueba incluyen diferentes operaciones como suma, resta, multiplicación y operaciones lógicas (AND, OR, XOR). También se incluye un caso de prueba para la operación NOP, que no existe en el módulo ``ALU`` y se utiliza como un caso de error.

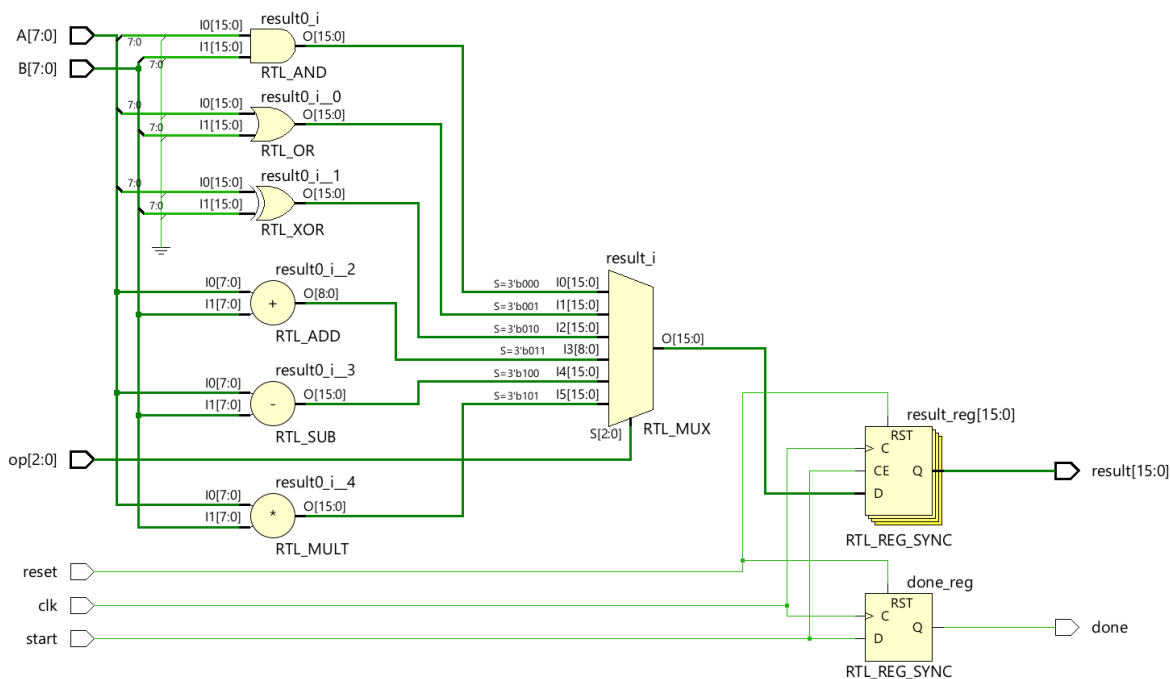
Finalmente, se utiliza la directiva ``$finish`` para finalizar la simulación después de que se hayan completado todos los casos de prueba.

Además del bloque ``initial`` para configurar y ejecutar los casos de prueba, hay otro bloque ``always`` que genera la señal de reloj ``clk``. Esta señal se invierte cada 5 unidades de tiempo, generando un reloj con una frecuencia de 10 unidades de tiempo.

En resumen, este testbench verifica el funcionamiento del módulo ``ALU`` al simular varios casos de prueba con diferentes operaciones y configuraciones. Los resultados y las señales se imprimen durante la simulación para verificar la correcta operación del módulo. Este testbench es útil para verificar el

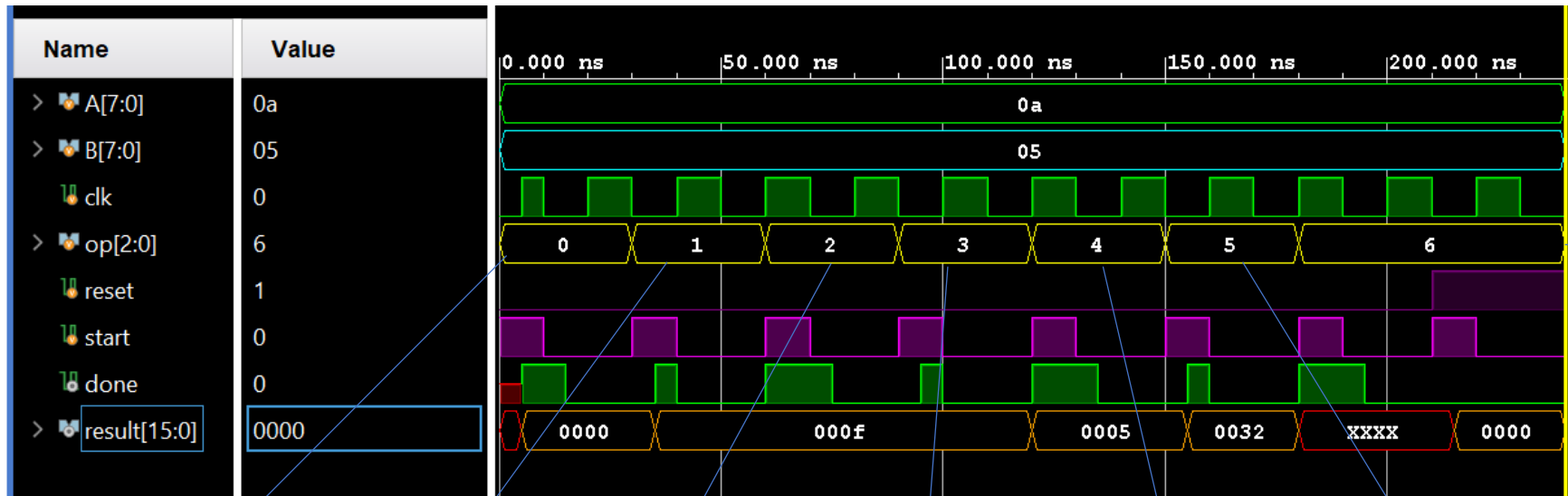
comportamiento y la funcionalidad del módulo 'ALU' antes de integrarlo en un sistema más grande.

Implementación RTL en Vivado 2022.2



Podemos observar cómo utilizó una compuerta AND, OR y XOR, operadores de suma, resta y multiplicación, un multiplexor y FlipFlops para la implementación del medio sumador.

Resultado de la simulación:



Op = 0 = AND
A = 0a=1010
B = 05=0101
result = 0000 ✓

Op = 1 = OR
A = 0a=1010
B = 05=0101
result = f = 1111 ✓

Op = 2 = XOR
A = 0a=1010
B = 05=0101
result = f = 1111 ✓

Op = 3 = SUMA
A = 0a= 10
B = 05= 5
result = f = 15 ✓

Op = 4= RESTA
A = 0a= 10
B = 05= 5
result = 32 = 5 ✓

Op = 6 = NOP
A = 0a= 10
B = 05= 5
result = XXXX ✓

Reset = 1
done = 0
result = 0 ✓

Conclusión

El código del módulo ALU describe una Unidad Aritmético-Lógica (ALU) que es un componente clave en un procesador. La ALU es responsable de realizar operaciones aritméticas y lógicas en datos. El módulo ALU tiene entradas para los operandos (A y B), la señal de reloj (clk), el tipo de operación (op), así como señales de control como reset y start. Las salidas del módulo son el resultado de la operación (result) y una señal de finalización (done).

El código del testbench proporciona casos de prueba para verificar el funcionamiento del módulo ALU. Cada caso de prueba configura las entradas del módulo y verifica las salidas esperadas. El testbench simula diferentes operaciones aritméticas y lógicas, como suma, resta, multiplicación y operaciones lógicas (AND, OR, XOR). También incluye un caso de prueba para verificar la respuesta cuando se utiliza una operación inexistente (NOP). Durante la simulación, se monitorean las señales de entrada y salida para verificar la correcta operación del módulo ALU.

En conclusión, el código del módulo ALU describe la funcionalidad del componente, mientras que el testbench proporciona casos de prueba para verificar su comportamiento. Estos códigos demuestran la capacidad de realizar operaciones aritméticas y lógicas en Verilog, lo cual es fundamental en el diseño de procesadores y sistemas digitales. Al utilizar un testbench para verificar el módulo, se asegura que la implementación de la ALU sea correcta y se reduce el riesgo de errores en la etapa de diseño.

Referencia

1. Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog, Springer, 1st Edition, USA, 2017.
2. Harris, D., & Harris, S. (2013). Digital design and computer architecture (2nd ed.). Morgan Kaufmann Publishers.
3. Palnitkar, S. (2003). Verilog HDL: A guide to digital design and synthesis (2nd ed.). Prentice Hall.
4. Weste, N. H. E., & Harris, D. (2011). CMOS VLSI design: A circuits and systems perspective (4th ed.). Addison-Wesley.