Reporte Actividad 03: ALU

Estrada Rivera Gustavo de Jesús 220746114 | 07/09/2025

INTRODUCCIÓN

El presente reporte se centra en una **Unidad Aritmético-Lógica (ALU)**, el cual es uno de los componentes más cruciales en el diseño de un procesador. Es el bloque fundamental encargado de realizar todos los cálculos numéricos (como sumas y restas) y operaciones lógicas (AND, OR, XOR) que un programa necesita para ejecutarse. En el corazón de las operaciones aritméticas se encuentra el sumador, un circuito digital cuya eficiencia impacta directamente el rendimiento general del procesador.

Para implementar una ALU en hardware digital (como en FPGA), se nos solicitó agregar sumadores, compuertas lógicas y desplazadores.

Existen múltiples arquitecturas para implementar sumadores, cada una con un compromiso diferente entre velocidad, área de silicio y complejidad. A continuación, se exploran algunas de las más relevantes:

Sumador de Acarreo Rizado (Ripple-Carry Adder): Es el diseño más simple, formado por sumadores completos conectados en serie, donde el acarreo de uno se usa como entrada del siguiente. Su principal desventaja es la lentitud, debido a que el acarreo debe propagarse desde el bit menos significativo hasta el más significativo.

Sumador con Anticipación de Acarreo (Carry-Lookahead Adder): Esta arquitectura acelera el cálculo de acarreos al anticiparlos usando señales de Propagación (P) y Generación (G), trabajando en paralelo. Aunque es más rápida, necesita más hardware y es más compleja.

Sumador con Selección de Acarreo (Carry-Select Adder): Este diseño equilibra Ripple-Carry y Carry-Lookahead, dividiendo los bits en bloques. Para cada bloque, se hacen dos sumas paralelas con acarreo de entrada “0” y “1”. Un multiplexor elige el resultado correcto según el acarreo real del bloque anterior. Es más rápido que Ripple-Carry y menos complejo que Carry-Lookahead completo.

Sumador con Almacenamiento de Acarreo (Carry-Save Adder): Se usa principalmente para sumar varios números. En vez de propagar el acarreo en cada suma, produce una suma parcial y una serie de "acarreos guardados". El resultado final se obtiene sumando ambos con un sumador rápido, como el Carry-Lookahead.

OBJETIVOS

Como objetivos podemos describir los siguientes:

- Diseñar y simular una **ALU extendida de 4 bits** que implemente operaciones aritméticas, lógicas, comparaciones y desplazamientos.
- Verificar el correcto funcionamiento de la ALU para operaciones aritméticas, lógicas, de comparación y de desplazamiento mediante un banco de pruebas (testbench).
- Documentar la arquitectura, el conjunto de operaciones y la importancia de la ALU como componente central de una Unidad Central de Procesamiento (CPU).
- Comprender la relevancia de la ALU en la arquitectura de un procesador.

DESARROLLO

Para realizar esta práctica se realizado un diseño de ALU el cual consta de la siguiente descripción:

Entradas:

- ***A[3:0] (Entrada de 4 bits)***
- ***B[3:0] (Entrada de 4 bits)***
- ***Sel[3:0] (Entrada de 4 bits)***

Salida:

- ***C[3:0] (Salida de 4 bits)***

Internamente, un multiplexor controlado por la señal sel elige cuál de los resultados de las operaciones (suma, resta, AND, etc.) se enrutará hacia la salida C. Todas las operaciones se calculan en paralelo, pero solo una se selecciona.

A continuación, se muestra su diagrama:

A su vez, se realizó un programa en pseudocódigo que utilice esas operaciones según su caso:

```
// Suma (sel= 0000)
int a = 5;
int b = 3;
int c = a + b; // c será 8

// Comparación (igualad)(sel= 0101)
if (A == B) { // sel = 0101
    // hacer algo
}

// Desplazamiento (a la Izquierda) (sel=0111)
// Equivalente a multiplicar por 2^n.
int a = 3; // 0b0011
int c = a << 2; // Desplaza 2 posiciones. c será 12 (0b1100)

// Compuerta AND (sel=0010)
int a = 0b1010;
int b = 0b1100;
int c = a & b; // c será 0b1000
```

Además de las operaciones básicas, esta ALU realiza ***comparaciones (A == B, A > B)***, que devuelven 4'b0001 para "verdadero" y 4'b0000 para "falso", facilitando decisiones en el procesador. También incluye **desplazamientos lógicos**: a la *izquierda* (<<), que multiplica por potencias de 2, y a la *derecha* (>>), que divide enteros por potencias de 2, rellenando con ceros los espacios vacíos.

Podemos resumir esto con la siguiente tabla, en donde la ALU se define por la señal de control ***sel***:

sel [3:0]	Operación	Descripción	Resultado (C)
0000	A + B	Suma	A + B
0001	A – B	Resta	A – B
0010	A & B	AND	A & B
0011	A | B	OR	A | B
0100	A ^ B	XOR	A ^ B
0101	A == B	Igualdad	1 si A=B, 0 si no
0110	A > B	Mayor que	1 si A>B, 0 si no
0111	A << B	Desplazamiento a la izquierda	A << B

1000	A >> B	Desplazamiento a la derecha	A >> B
default	N/A	Valor por defecto	C= 0

Una vez obtenido el diseño de nuestra ALU y descrito la lógica de sus operaciones, a continuación, se pasará el código y su simulación:

Código:

ALU Extendida:

Para esta sección, rescatamos el código de la Pre-ALU (en donde solo estaba implementado la suma y la compuerta AND), y agregamos las operaciones restantes que se nos piden:

```
timescale 1ns/1ns
module alu_extendida (
        input  [3:0] A,
        input  [3:0] B,
        input  [3:0] sel,
        output reg [3:0] C
);

always @*
        begin
                case(sel)
                        // Operaciones Simples
                        4'b0000: C = A + B;        // Suma
                        4'b0001: C = A - B;        // Resta

                        // Compuertas Logicas
                        4'b0010: C = A & B;        // AND
                        4'b0011: C = A | B;        // OR
                        4'b0100: C = A ^ B;        // XOR

                        // Comparaciones
                        4'b0101: C = (A == B) ? 4'b0001 : 4'b0000; //
Igualdad (A == B)
                        4'b0110: C = (A > B)  ? 4'b0001 : 4'b0000; //
Mayor que (A > B)

                        // Desplazamientos lÃ³gicos
                        4'b0111: C = A << B;       // Desplazamiento a la
izquierda
                        4'b1000: C = A >> B;       // Desplazamiento a la
derecha

                        default: C = 4'b0000;
                endcase
        end
```

```
endmodule
```

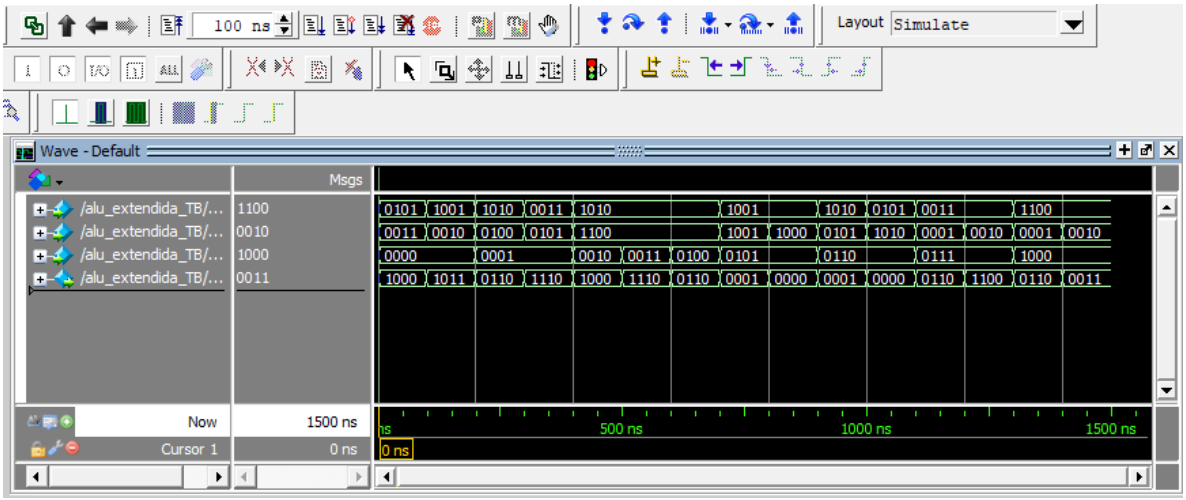
TB ALU Extendida:
Con la misma base del código del TB de la Pre-ALU, la complementamos con las operaciones restantes:

```
`timescale 1ns/1ns
module alu_extendida_TB();

reg  [3:0] A_tb;
reg  [3:0] B_tb;
reg  [3:0] sel_tb;
wire [3:0] C_tb;

alu_extendida DUV (
        .A(A_tb),
        .B(B_tb),
        .sel(sel_tb),
        .C(C_tb)
);

initial
begin
        $monitor("Tiempo=%0t | A=%d, B=%d, sel=%b | Resultado C = %d
(%b)",
                        $time, A_tb, B_tb, sel_tb, C_tb, C_tb);
        sel_tb = 4'b0000;
        A_tb = 4'd5; B_tb = 4'd3;
        #100;
        A_tb = 4'd9; B_tb = 4'd2;
        #100;
        sel_tb = 4'b0001;
        A_tb = 4'd10; B_tb = 4'd4;
        #100;
        A_tb = 4'd3; B_tb = 4'd5;
        #100;
        sel_tb = 4'b0010;
        A_tb = 4'b1010; B_tb = 4'b1100;
        #100;
        sel_tb = 4'b0011;
        A_tb = 4'b1010; B_tb = 4'b1100;
        #100;
        sel_tb = 4'b0100;
        A_tb = 4'b1010; B_tb = 4'b1100;
        #100;
        sel_tb = 4'b0101;
        A_tb = 4'd9; B_tb = 4'd9;
        #100;
        A_tb = 4'd9; B_tb = 4'd8;
        #100;
        sel_tb = 4'b0110;
        A_tb = 4'd10; B_tb = 4'd5;
        #100;
```

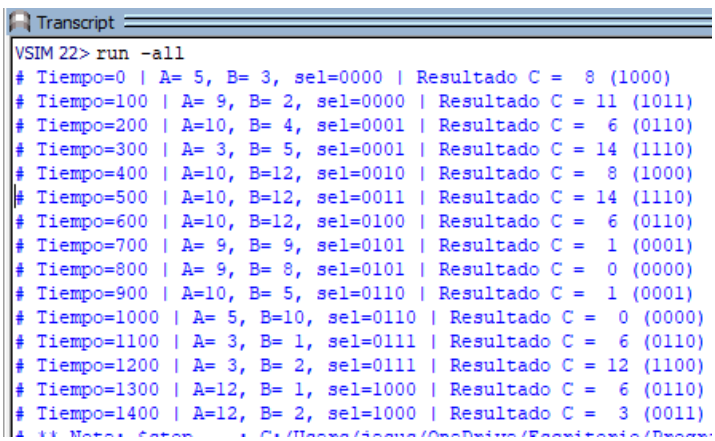
```
        A_tb = 4'd5; B_tb = 4'd10;
        #100;
        sel_tb = 4'b0111;
        A_tb = 4'b0011; B_tb = 4'd1;
        #100;
        A_tb = 4'b0011; B_tb = 4'd2;
        #100;
        sel_tb = 4'b1000;
        A_tb = 4'b1100; B_tb = 4'd1;
        #100;
        A_tb = 4'b1100; B_tb = 4'd2;
        #100;
        $stop;
end

endmodule
```

Simulación:

Para la simulación solo debemos ejecutar el testbench de la ALU, ya que es en donde hemos definido las entradas para realizar la simulación, simulación que se vería de la siguiente forma:



Para complementar lo anterior y describir mejor los resultados de la simulación, la salida del monitor (*$monitor*) nos permite observar cómo cambia el resultado **C** en función de las entradas **A**, **B** y ***sel*** en cada intervalo de tiempo:



Reflexión:

¿Por qué la ALU es fundamental en un procesador?

La ALU es fundamental ya que, en términos generales es el corazón computacional del procesador. Sin ella, un CPU no podría ejecutar las tareas más básicas que definen la computación. Cada instrucción de un programa, ya sea sumar dos números, comparar valores para un if, o manipular bits para controlar hardware, es ejecutada por la ALU. Es la unidad que transforma los datos siguiendo las directrices del programa. Un procesador puede tener registros para almacenar datos y una unidad de control para dirigir el flujo, pero es la ALU la que realiza el trabajo de cálculo real.

CONCLUSIONES

El diseño y la implementación de la ALU extendida de 4 bits se completaron de manera satisfactoria. El testbench confirmó que todas las operaciones, incluyendo las aritméticas, lógicas, de comparación y de desplazamiento, funcionan como se esperaba.

Esta actividad demuestra la posibilidad de construir un componente de cálculo versátil utilizando Verilog. Se ha fortalecido la comprensión de cómo la señal de control (sel) permite que una sola unidad de hardware realice una amplia variedad de funciones, un principio fundamental en el diseño de procesadores.

Finalmente, el análisis de su integración con un decodificador de instrucciones clarifica el papel esencial de la ALU como unidad central de ejecución, que opera bajo la dirección de la unidad de control para dar vida al software.

REFERENCIAS

How does an Adder work? | Types & Functions explained. (2023, May 28). https://www.lenovo.com/gb/en/glossary/adder/?orgRef=https%253A%252F%252Fwww.google.com%252F

Agarwal, T. (2020, October 28). Half Adder and Full Adder Circuit with Truth Tables. ElProCus - Electronic Projects for Engineering Students. https://www.elprocus.com/half-adder-and-full-adder/

Unidad aritmética lógica (ALU) - definición | Distribuidor de componentes electrónicos. Tienda en línea: Transfer Multisort Elektronik México. (n.d.). TME. https://www.tme.com/mx/es/news/library-articles/glossary/page/61914/unidad-aritmetica-logica-alu-definicion/

Repositorio en GitHub:
https://github.com/GustavoEstrada4611/Arquitectura_de_Computadoras/tree/main/ACT03_ALUEXT