Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Ingeniería en Computación

ARQUITECTURA DE COMPUTADORAS

TRABAJO PRÁCTICO Nº1

DISEÑO Y TESTING DE UNA ALU

Integrantes:

Tissot, Esteban *35276396*

Manero, Matías *35250112*

Profesor:

Ing. Santiago Rodriguez

Índice

Introducción

Empezaremos definiendo uno de los circuitos digitales más usados es estos días, la ALU, la cual podemos definirla como una unidad aritmética lógica o unidad aritmético-lógica (siglas en inglés de arithmetic logic unit), que calcula operaciones aritméticas (comunmente suma, resta, multiplicación, etc.) y operaciones lógicas (sí, y, o, no), entre valores (generalmente dos).

Muchos tipos de circuitos electrónicos necesitan realizar algún tipo de operación aritmética y podemos mencionar que los circuitos electrónicos más complejos son los que están construidos dentro de los chips de microprocesadores modernos. Por lo tanto, estos procesadores tienen dentro de ellos una ALU muy compleja y potente. De hecho, un microprocesador moderno puede tener múltiples núcleos, cada núcleo con múltiples unidades de ejecución, cada una de ellas con múltiples ALU.

Una herramienta usada con gran frecuencia son las placas FPGA, la misma la podemos definir como dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada en el momento mediante un lenguaje de descripción especializado. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip.

Diseño

EL diseño seguido fue el propuesto por el profesor, siendo este el siguiente

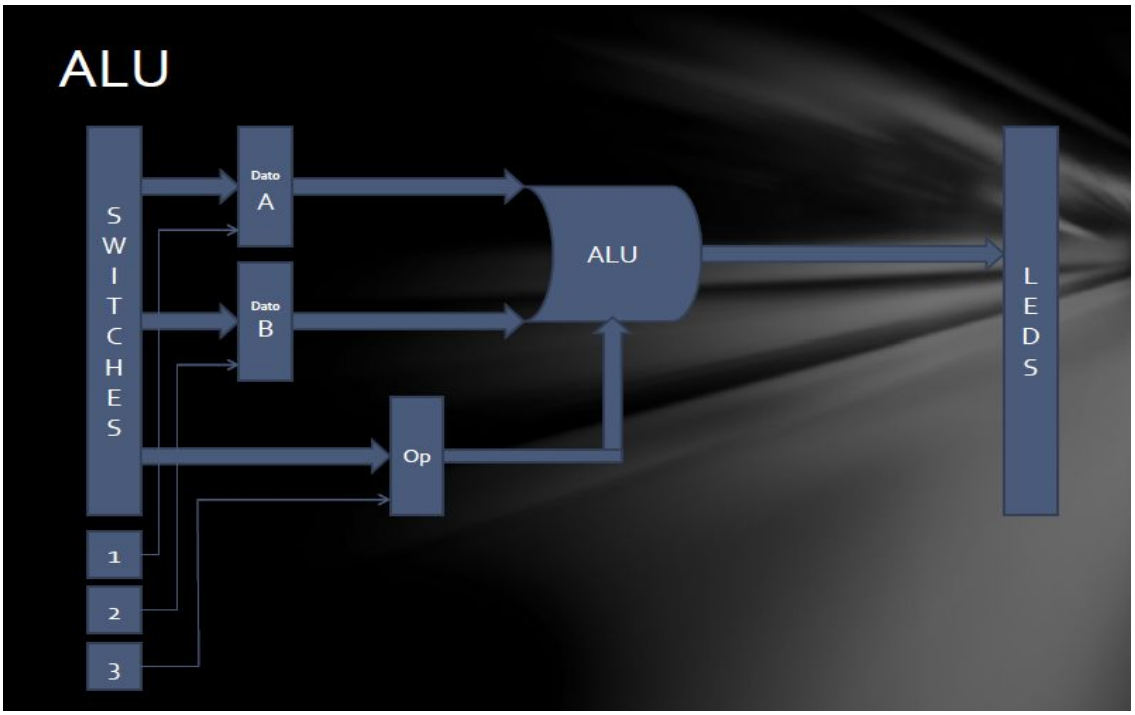


Figura 1

Los códigos de las operaciones son los que se presentan a continuación

Operación	Código
ADD	100000
SUB	100010
AND	100100
OR	100101
XOR	100110
SRA	000011
SRL	000010
NOR	100111

Figura 2

Implementación

Antes de empezar con la implantación de la ALU explicaremos brevemente su funcionamiento. Podemos decir que la misma necesita datos de entrada sobre los que se harán las operaciones y un código indicando que operación realizar, la salida es el resultado del cómputo de la operación. Para resumir la ALU toma datos de los registros de entrada, estos datos son procesados y los resultados de esta operación se almacenan en los registros de salida.

En nuestro caso la implementación de la ALU se llevó a cabo usando el software de desarrollo Xilinx y el lenguaje de programación Verilog. Para esto se realizó la creación de dos módulos que se comunicaban entre ellos. El primer módulo es el encargado de almacenar los registros de entrada y el código de operación, mientras que el segundo en hacer los cálculos correspondientes dependendiendo de dicho código.

Este código fue cargado, después del respectivo mapeo de pines, a una FPGA de la familia Spartan3E. El dispositivo usado de esa familia es el XC3S100E(Figura 3).

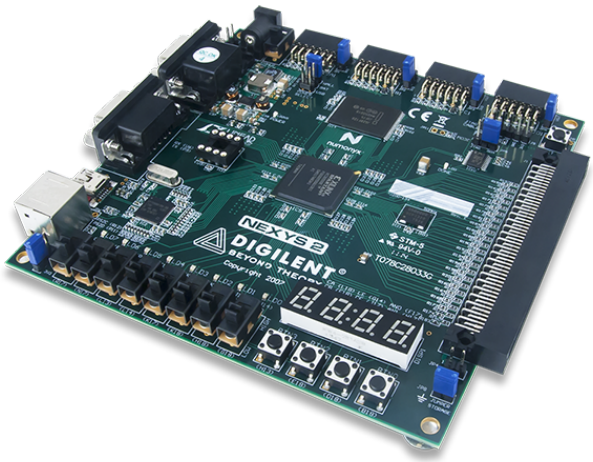


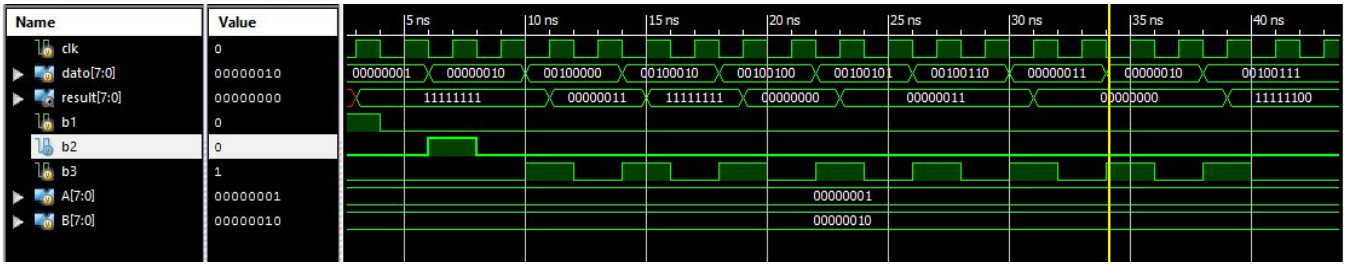
FIGURA 3

Se usaron los swiches que se ven en la figura para introducir los respectivos valores de entrada y el código de operación. Se utilizan diferentes botones para introducirlos en memoria, siguiendo el diseño de la Figura 2.

TestBenchs

Luego del desarrollo del software y antes de cargarlo a la placa se realizó un test para comprobar el correcto funcionamiento del sistema desarrollado, en el ANEXO se encuentra el código de dicho test.

Como resultado hemos obtenido la siguiente gráfica, la cual ilustra que el circuito diseñado funciona correctamente



Conclusión

Hemos aprendido a diseñar circuitos descriptos por código Verilog, cargarlos en una placa FPGA y simular su comportamiento. De esta manera se puede desarrollar circuitos combinacionales sincronizados por un clock por medio de código de descripción de hardware, lo que nos permite simular y estudiar su comportamiento de una manera más simple, rápida y productiva.

Cabe aclarar que si bien al compilar el proyecto se obtiene un warning, no es de gran importancia en la implementación. Es decir no introduce un error, solo informa que se produce un loop combinacional cuando se hace un desplazamiento aritmético más grande que la cantidad de bits que el dato posee. La solución sencilla es acotar los bits del corrimiento para que sean igual a la cantidad de bits que se pueden desplazar, esto se podría hacer con mediante el paso de un parámetro para que el proyecto siga siendo escalable.

Anexo

```
module Test_Alu;

	// INPUTS
	reg clk;
	reg [7:0] dato;
	reg b1;
	reg b2;
	reg b3;

	//REGISTERS
	reg [7:0]A;
	reg [7:0]B;

	reg [5:0] ADD = 6'b100000;
	reg [5:0] SUB = 6'b100010;
	reg [5:0] AND = 6'b100100;
	reg [5:0] OR  = 6'b100101;
	reg [5:0] XOR = 6'b100110;
	reg [5:0] SRA = 6'b000011;
	reg [5:0] SRL = 6'b000010;
	reg [5:0] NOR = 6'b100111;


	// OUTPUTS
	wire [7:0] result;

	// Instantiate the Unit Under Test (UUT)
	Top_Alu uut (
		.clk(clk),
		.dato(dato),
		.b1(b1),
		.b2(b2),
		.b3(b3),
		.result(result)
	);
	always begin
		#1 clk =~clk;
	end

	initial begin
		//Numero y codigo de operacion
		A= 8'd1;
		B= 8'd2;

		// Initialize Inputs
		clk = 0;
```

```
dato = 0;
b1 = 0;
b2 = 0;
b3 = 0;

#2;
b1=1'b1;
dato=A;

#2;
b1=1'b0;

#2;
b2=1'b1;
dato=B;

#2;
b2=1'b0;

#2;
b3=1'b1;
dato=ADD;

#2;
b3=1'b0;

#2;
b3=1'b1;
dato=SUB;

#2;
b3=1'b0;

#2;
b3=1'b1;
dato=AND;

#2;
b3=1'b0;

#2;
b3=1'b1;
dato=OR;

#2;
b3=1'b0;

#2;
b3=1'b1;
dato=XOR;

#2;
b3=1'b0;
```

```
            #2;
            b3=1'b1;
            dato=SRA;

            #2;
            b3=1'b0;

            #2;
            b3=1'b1;
            dato=SRL;

            #2;
            b3=1'b0;

            #2;
            b3=1'b1;
            dato=NOR;

            #2;
            b3=1'b0;

        end

endmodule
```