Trabajo Páctico Nro. 1

Implementación de una ALU en verilog sobre FPGA Spartan3E

Morales Esteban, Salamandri Santiago, Uboldi Marino
FCEFyN, UNC, Argentina

ABSTRACT

En computación, la unidad aritmético lógica, también conocida como ALU (siglas en inglés de arithmetic logic unit), es un circuito digital que calcula operaciones aritméticas (como suma, resta, multiplicación, etc.) y operaciones lógicas (si, y, o, no), entre dos números.

CONSIGNA IMPLEMENTADA

- Implementar una ALU sobre FPGA.
- Utilizar las placas de desarrollo Basys II.
- La ALU debe ser parametrizable (bus de datos) para poder ser utilizada posteriormente en el trabajo final.
- Validar el desarrollo por medio de Test Bench.
- La ALU deberá realizar las siguientes operaciones:

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

DESARROLLO

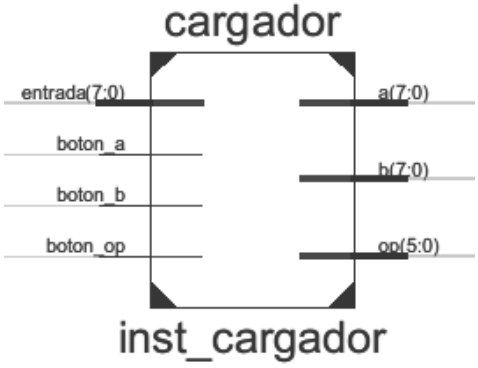
Se implementa una ALU de 8 bits (bus de datos parametrizable) en FPGA, la cual se divide en 3 módulos:

- Calculadora: Módulo encargado de instanciar los otros módulos.
- Cargador: Módulo encargado de inicializar los valores de los registros.
- ALU: Módulo encargado de realizar las operaciones tabuladas en los requerimientos.

DESCRIPCIÓN

CARGADOR

```
cargador(entrada,boton_a,boton_b,
boton_op,a,b,op);
```



Entradas

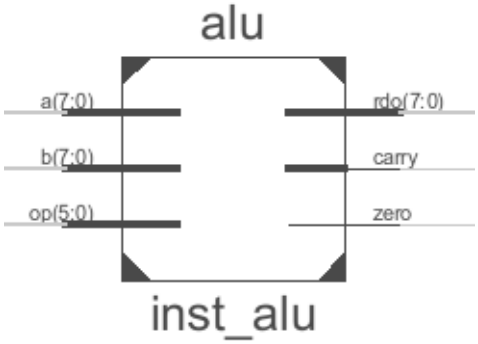
- entrada: Es el registro de entrada de ancho de bus parametrizable.
- boton_a: Cuando esta entrada está en ’1’ carga el valor de la entrada en la salida a;
- boton_b: Cuando esta entrada está en ’1’ carga el valor de la entrada en la salida b;
- boton_op: Cuando esta entrada está en ’1’ carga el valor de la entrada en la salida op;

Salidas

- a: Este registro de salida será un operando de la ALU.
- b: Este registro de salida será un operando de la ALU.
- op: Este registro de salida será el código de operación de la ALU.

ALU

`alu(a,b,op,rdo,carry,zero);`



Entradas

- a: Este registro de entrada es un operando de la ALU.
- b: Este registro de entrada es un operando de la ALU.
- op: Este registro de entrada es el código de operación de la ALU.

Variables Internas

- `wire [‘BUS_DAT_MSB+1:0] resultado;`

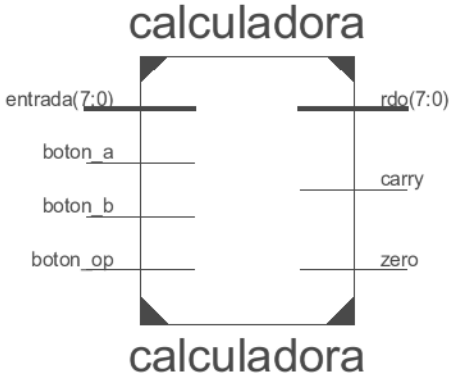
  Se utiliza una sola variable interna de tipo ”wire”para almacenar el resultado de la función ”funcion_alu”

Salidas

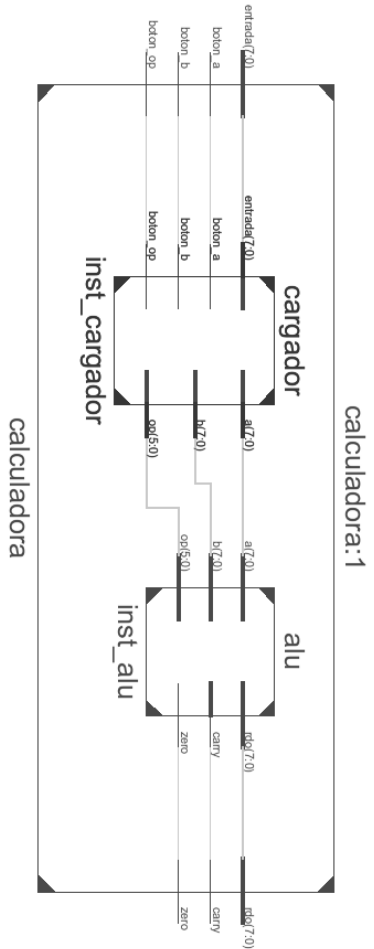
- rdo: Este registro almacena el resultado de la operación de la ALU.
- carry: Este registro de un bit representa el carry de la operación.
- zero: Este registro de un bit es la bandera de zero de la operación.

CALCULADORA

`module calculadora(entrada,boton_a,boton_b, boton_op,rdo,carry,zero);`



Como la calculadora es el módulo que instancia a los otros dos módulos las entradas serán iguales a las entradas de uno y las salidas serán iguales a las salidas del otro.



VALIDACIÓN

A continuación se muestran los resultados de los testbench realizados sobre todas las operaciones implementadas, el código es el siguiente:

```
`include "definiciones.vh"
module ALU_tb;

        // Inputs
        reg [7:0] a;
        reg [7:0] b;
        reg [5:0] op;

        // Outputs
        wire [7:0] rdo;
        wire carry;
        wire zero;

        alu #(8,6) uut(
                .a(a),
                .b(b),
                .op(op),
                .rdo(rdo),
                .carry(carry),
                .zero(zero)
        );

        initial begin
                a = 0;
                b = 0;
                op = 0 ;
                #20;
//-------ADD----------------------------//
                a = 250;
                b = 20;
                op = `ADD ;
                #20;
//-------SUB----------------------------//
                a = 250;
                b = 20;
                op = `SUB ;
                #20;
//-------SUB_zero-----------------------//
                a = 250;
                b = 250;
                op = `SUB ;
                #20;
//-------AND----------------------------//
                a = 8'b10101010;
                b = 8'b11110000;
                op = `AND ;
                #20;
//-------OR-----------------------------//
                a = 8'b10101010;
                b = 8'b11110000;
                op = `OR ;
                #20;
//-------XOR----------------------------//
                a = 8'b10101010;
                b = 8'b11110000;
                op = `XOR ;
                #20;
//-------NOR----------------------------//
                a = 8'b10101010;
                b = 8'b11110000;
                op = `NOR ;
                #20;
//-------SRA----------------------------//
                a = 8'b11000000;
                b = 8'b00000011;
                op = `SRA ;
                #20;
//-------SRL----------------------------//
                a = 8'b11000000;
                b = 8'b00000011;
                op = `SRL ;
                #20;

        end

endmodule
```

Para las pruebas se fueron cambiando los valores de “entrada”, para cargar los registros con los valores que prueben mejor cada una de las operaciones.

ADD (binario 100000)

Variable	Valor
a	250
b	20
op	100000
resultado	14 + (carry)

Teniendo en cuenta el carry (256 por ser bit 9) mas 14 se tiene 270.

SUB (binario 100010)

Variable	Valor
a	250
b	20
op	100010
resultado	230

Se agrega una resta que active la bandera de zero.

Variable	Valor
a	250
b	250
op	100010
resultado	0 + (zero)

AND (binario 100100)

Variable	Valor
a	10101010
b	11110000
op	100100
resultado	10100000

OR (binario 100101)

Variable	Valor
a	10101010
b	11110000
op	100101
resultado	11111010

XOR (binario 100110)

Variable	Valor
a	10101010
b	11110000
op	100110
resultado	01011010

NOR (binario 100111)

Variable	Valor
a	10101010
b	11110000
op	100111
resultado	01011010

SRA (binario 000011)

Variable	Valor
a	11000000
b	00000011
op	000011
resultado	11111000

SRL (binario 000010)

Variable	Valor
a	11000000
b	00000011
op	000010
resultado	00011000

SIMULACIÓN A continuación se muestran los resultados obtenidos de la simulación del testbench en ISim.