UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS, QUÍMICAS Y NATURALES

Arquitectura de Computadoras

TRABAJO PRÁCTICO N° 1 : Arithmetic Logic Unit (ALU)

ALUMNOS:

- BOSACK,Federico
- OLIVA, Nahuel

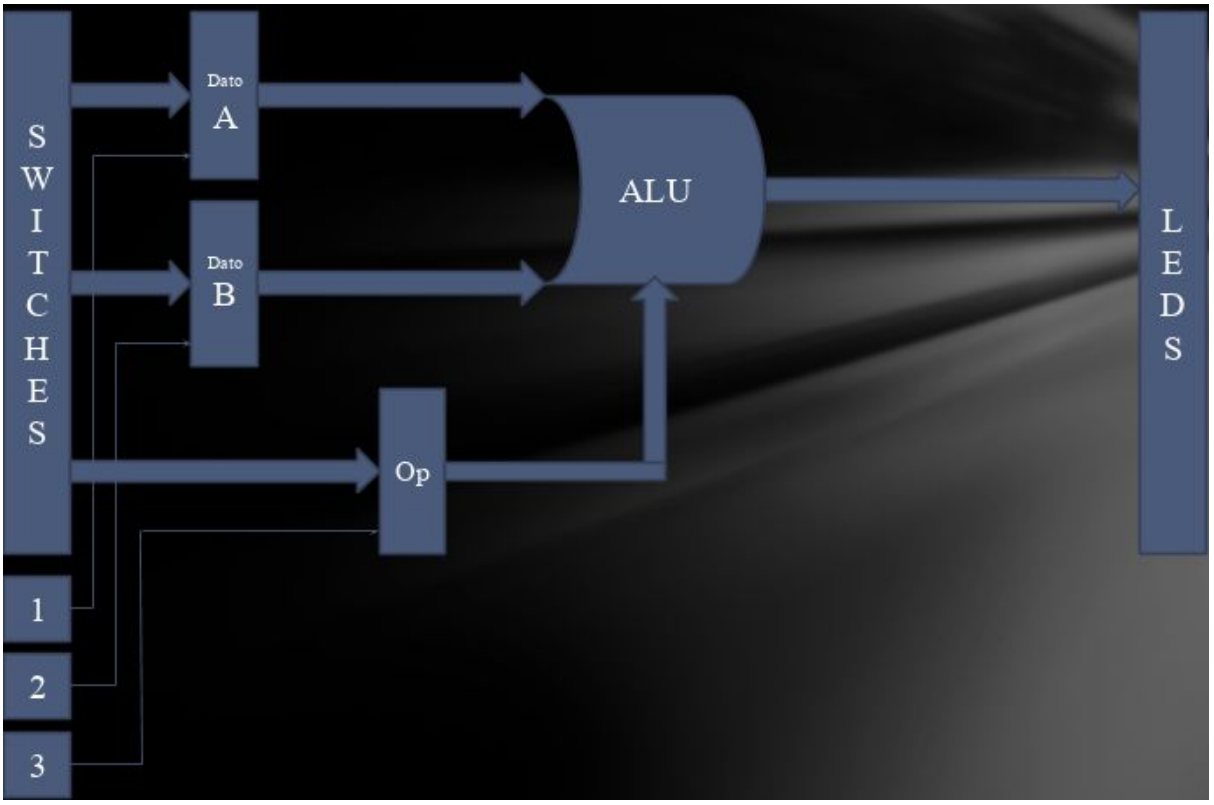
Fecha: 02/10/2019

INTRODUCCIÓN

El presente informe se realiza con el objetivo de describir en detalle la implementación de una unidad de lógica aritmética ,con bus de datos parametrizable, mediante lenguaje de descripción de hardware,más precisamente en lenguaje verilog, para luego poder ser utilizado en la placa FPGA,Basys 2 o Nexys 3 a preferencia.

Además de la ALU ,el sistema deberá implementarse de tal manera que permita validar el funcionamiento de la misma, desde la FPGA,mediante el uso de los switches y botones (1,2,3) para el ingreso de datos y de diodos leds para la salida del resultado,junto con los respectivos testbenches que validen el sistema completo .

En la siguiente figura se muestra la arquitectura del sistema a implementar.

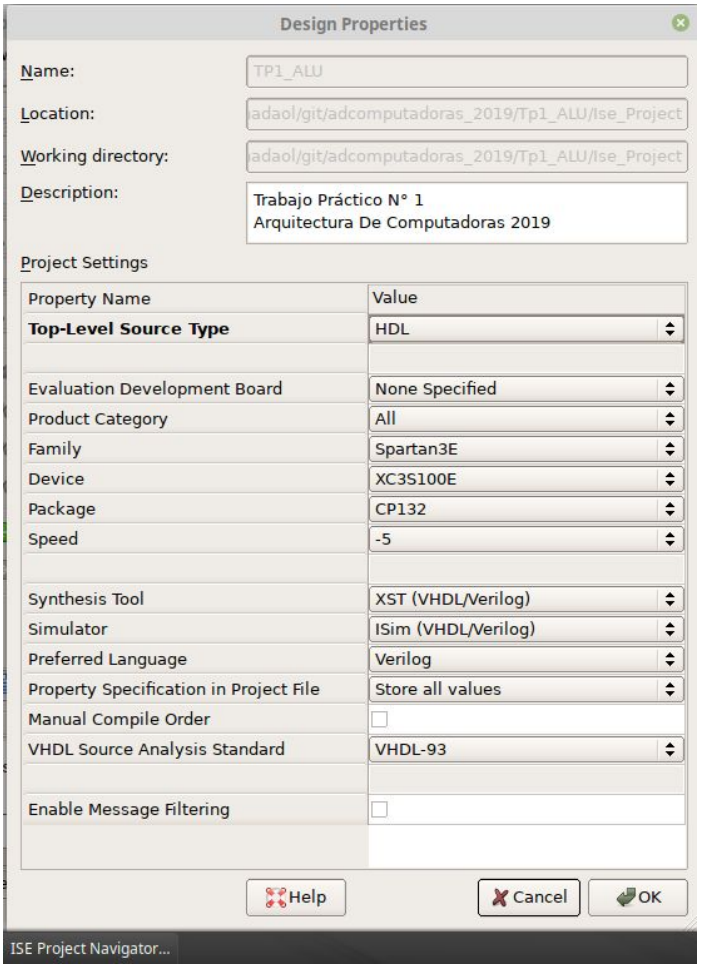


La ALU deberá soportar las operaciones de suma,resta,and,or,or exclusiva,or negado,desplazamiento lógico izquierdo y desplazamiento aritmético derecho, según los siguientes códigos de operación.

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

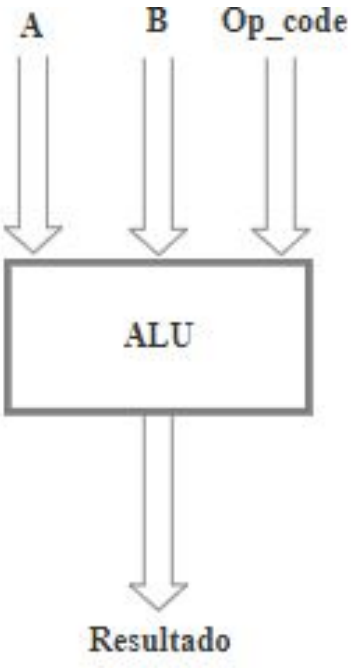
DESARROLLO

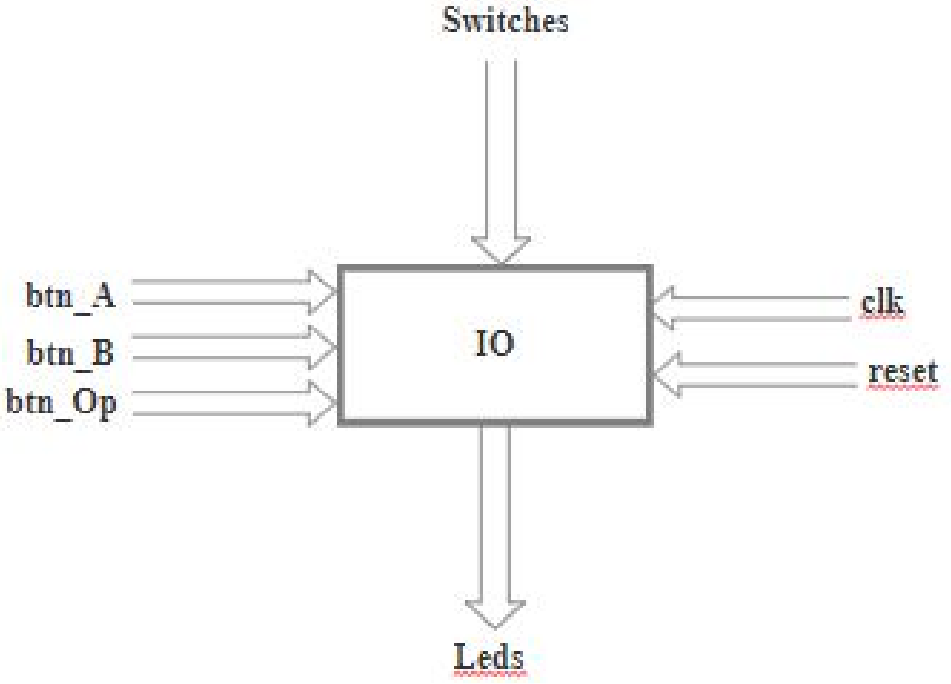
Para la implementación del sistema se utilizó el entorno de desarrollo IDE Design con las siguientes configuraciones.



El sistema consta de dos módulos principales,el módulo ALU para la implementación de la unidad de aritmética lógica,y el módulo IO el cual opera como una interface entre el operador y la ALU.

A continuación se muestran los diagramas de bloque de los módulos mencionados





En la siguiente imagen se observa la lógica combinacional implementada mediante un bloque always y una sentencia case para el código de operación del módulo de la unidad aritmética,el cual consta con bus de datos parametrizado por DATA_LENGTH ,el cual por default se define con 1 byte.

```
module Alu#(parameter DATA_LENGTH=8)
                        (
                        input wire signed [(DATA_LENGTH-1):0] A,
                        input wire signed [(DATA_LENGTH-1):0] B,
                         input wire [5:0] Op_code,
                         output reg signed [(DATA_LENGTH-1):0] Resultado
                         );

        always @(*)
        begin
                case(Op_code)
                        6'b100000: Resultado=A+B;//ADD
                        6'b100010: Resultado=A-B;//SUB
                        6'b100100: Resultado=A&B;//AND
                        6'b100101: Resultado=A|B;//OR
                        6'b100110: Resultado=A^B;//XOR
                        6'b000011: Resultado=A>>>B;//SRA
                        6'b000010: Resultado=A>>B;//SRL
                        6'b100111: Resultado=~(A|B);//NOR
                        default: Resultado={(DATA_LENGTH){1'b0}};
                endcase
        end

endmodule
```

A continuación se observa la implementación del ‘Top module’ IO el cual tiene como objetivo sincronizar las entradas con el clock de la Fpga y proveer de una lógica que permita el cargado de datos a la unidad aritmética, para así luego poder presentar el resultado mediante los leds de la placa.

Este módulo consta de tres variables las cuales luego se mapearán a los botones de la placa para el cargado de datos,al mantenerlas presionadas el dato indicado por el posicionamiento de los switches se redirigirá bien al operando 1,2 o al código de operación de la unidad aritmética según corresponda.

```
module IO#(parameter DATA_LENGTH=8)
                    (input wire [(DATA_LENGTH-1):0] switches,
                              input wire btn_A, btn_B, btn_Op, clk, reset,
                              output wire [(DATA_LENGTH-1):0] Leds
    );

        reg [(DATA_LENGTH-1):0] A;
        reg [(DATA_LENGTH-1):0] B;
        reg [5:0] Op_code;


        Alu #(.DATA_LENGTH(DATA_LENGTH)) Alu(.A(A),.B(B),.Op_code(Op_code),.Resultado(Leds));

        always @(posedge clk)

        begin
                if(reset==1)
                begin
                A<=0;
                B<=0;
                Op_code<=6'b000000;
                end

case({btn_A,btn_B,btn_Op})
                        3'b100: A<=switches;//Cargo Operando A
                        3'b010: B<=switches;//Cargo Operando B
                        3'b001: Op_code<=switches[5:0];//Cargo codigo de operacion
        endcase
        end
endmodule
```

Configuración para el mapeo de pines de la Basys 2 con el módulo IO

```
NET "switches[0]" LOC = P11;
NET "switches[1]" LOC = L3;
NET "switches[2]" LOC = K3;
NET "switches[3]" LOC = B4;
NET "switches[4]" LOC = G3;
NET "switches[5]" LOC = F3;
NET "switches[6]" LOC = E2;
NET "switches[7]" LOC = N3;
NET "Leds[0]" LOC = M5;
NET "Leds[1]" LOC = M11;
NET "Leds[2]" LOC = P7;
NET "Leds[3]" LOC = P6;
NET "Leds[4]" LOC = N5;
NET "Leds[5]" LOC = N4;
NET "Leds[6]" LOC = P4;
NET "Leds[7]" LOC = G1;
NET "btn_A" LOC = A7;
NET "btn_B" LOC = M4;
NET "btn_Op" LOC = C11;
NET "reset" LOC = G12;
NET "clk" LOC = B8;
```

TESTBENCHES

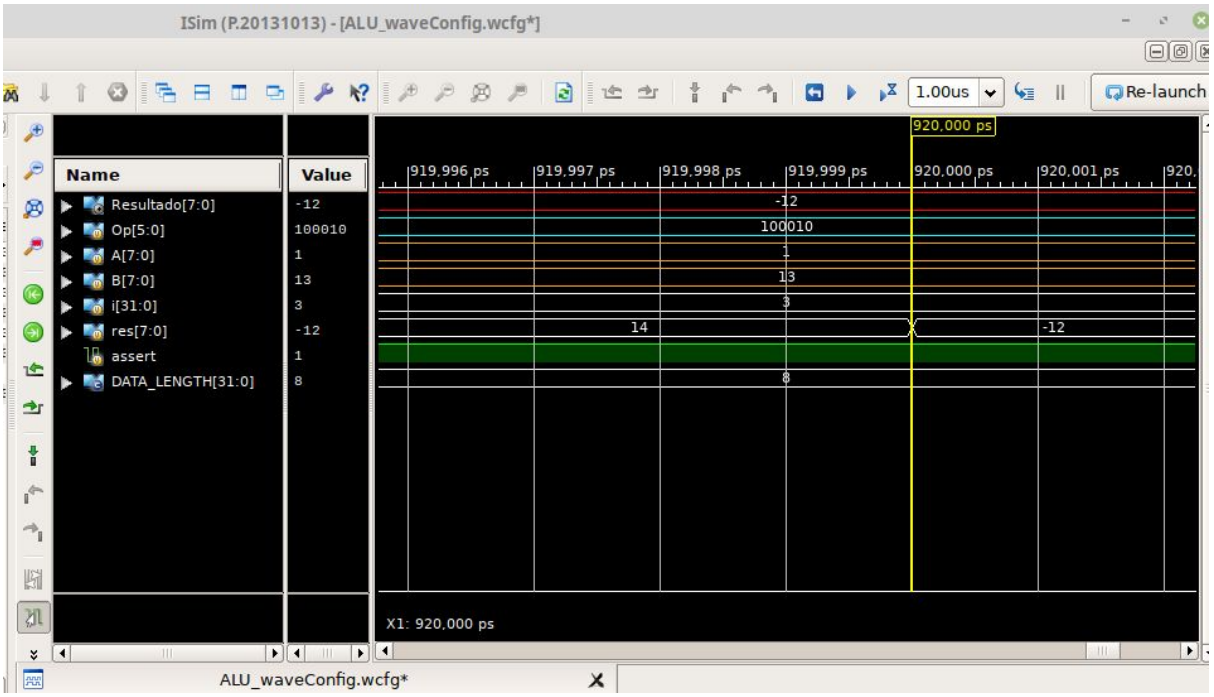
Para la validación del funcionamiento del sistema se realizó un testbench para cada módulo . En el siguiente código se observa el testbench que testea el módulo de unidad aritmética lógica, el cual se realiza mediante un loop, estableciendo aleatoriamente un valor para cada operando y posteriormente cambiando el código para probar todas operaciones soportadas.

Para una validación más sencilla de observar en la simulación, se creó un registro de control ‘res’ el cual se utiliza para comparar el resultado obtenido de la Alu con el obtenido en el testbench , para luego asignar al registro ‘assert’ si efectivamente coincide o no.

```
module Alu_sim1;

//local parameters
localparam DATA_LENGTH = 8;
// Inputs
        reg [5:0] Op;
        reg signed [(DATA_LENGTH-1):0] A;
        reg signed [(DATA_LENGTH-1):0] B;
// Outputs
        wire [(DATA_LENGTH-1):0] Resultado;

// Instancio el módulo a testear
        Alu #(.DATA_LENGTH(DATA_LENGTH))
        uut (
                .Op_code(Op),
                .A(A),
                .B(B),
                .Resultado(Resultado)
        );
integer i=0;
        reg signed [(DATA_LENGTH-1):0]res;
        reg assert;

        initial begin
// Inicializacion de registros
                Op = 0;
                A = 0;
                B = 0;
                i=0;
                res=0;
                assert=0;
end
```

```
always
begin
for(i=0;i<3;i=i+1)

//Cargo valores random a A y B
        #10;
                A=$random;
                #2;
                B=$random;
                #2;

                Op=6'b100000; //ADD
                #10;
                res=A+B;if(res==Resultado)assert=1;else assert=0;
                #10;
                Op=6'b100010; //SUB
                #10;
                res=A-B;if(res==Resultado)assert=1;else assert=0;
                #10;
                Op=6'b100100; //AND
                #10;
                res=A&B;if(res==Resultado)assert=1;else assert=0;
                #10;
                Op=6'b100101; //OR
                #10;
                res=A|B;if(res==Resultado)assert=1;else assert=0;
                #10;
                Op=6'b100110; //XOR
                #10;
                res=A^B;if(res==Resultado)assert=1;else assert=0;
                #10;
                Op=6'b000011; //SRA
                #10;
                res=A>>>B;if(res==Resultado)assert=1;else assert=0;
                #10;
                Op=6'b000010; //SRL
                #10;
                res=A>>B;if(res==Resultado)assert=1;else assert=0;
                #10;
                Op=6'b100111; //NOR
                #10;
                res=~(A|B);if(res==Resultado)assert=1;else assert=0;
                #10;
                Op=6'b111111; //UNDEFINED
                #10;
                res=0;if(res==Resultado)assert=1;else assert=0;
                #10;

        end

endmodule
```

En la siguiente imagen se observa en la simulación del ISim,el instante en el que se valida el resultado de resta, entre los operandos A (1) y B (13), obtenido de la Alu con la variable de control res,dando como resultado ambos -12 y un assert positivo.



Finalmente se puede observar a continuación el testbench del módulo principal ‘IO’.

Al igual que el testbench anterior se realiza mediante un ciclo for cargando los valores de los operandos y posteriormente cambiando todos los códigos de operación,con la salvedad de que ahora se genera un clock para la sincronización y los datos se cargan sobre una única entrada de 1 byte (datos->Switches) mediante los botones correspondientes.

```
module IO_sim1;

localparam DATA_LENGTH = 8;
// Inputs
        reg btn_A;
        reg btn_B;
        reg btn_Op;
        reg clk;
        reg signed [(DATA_LENGTH-1):0] dato;
// Outputs
        wire [(DATA_LENGTH-1):0] Resultado;

// Instancio el módulo a testear
        IO uut (
                .clk(clk),
                .btn_A(btn_A),
                .btn_B(btn_B),
                .btn_Op(btn_Op),
                .switches(dato),
                .Leds(Resultado)
        );

integer i = 0;
        initial
         begin
                dato = 0;
                btn_A=0;
                btn_B = 0;
                btn_Op= 0;
                clk = 0;
                end
always
begin
                #1 clk = !clk;
end
```

```
always
begin
for(i=0;i<3;i=i+1)
//Cargo valores random a A y B
        #10
        dato=$random;
                #2;
                btn_A=1;
                #2
                btn_A=0;
                #2
                dato=$random;
                #2;
                btn_B=1;
                #2
                btn_B=0;

//Realizo todas las Operaciones
                #10  dato = 6'h20;//ADD
                #10  btn_Op= 1;
                #10  btn_Op = 0;

                #10  dato = 6'h22;//SUB
                #10  btn_Op = 1;
                #10  btn_Op = 0;

                #10  dato = 6'h24;//AND
                #10  btn_Op = 1;
                #10  btn_Op = 0;

                #10  dato = 6'h25;//OR
                #10  btn_Op = 1;
                #10  btn_Op = 0;

                #10  dato = 6'h26;//XOR
                #10  btn_Op = 1;
                #10  btn_Op = 0;

                #10  dato = 6'h27;//NOR
                #10  btn_Op = 1;
                #10  btn_Op = 0;

                #10  dato = 6'h02;//SRA
                #10  btn_Op = 1;
                #10  btn_Op = 0;

                #10  dato = 6'h03;//SRL
                #10  btn_Op = 1;
                #10  btn_Op = 0;

        end

endmodule
```

En la siguiente imagen se observa mediante el debugger del ISim, el momento en el que se carga el código de operación de resta (100010) al presionar el botón btn_Op, anteriormente habiendo cargado el operando A en 9 y el operando B en 99,dando como resultado -90.