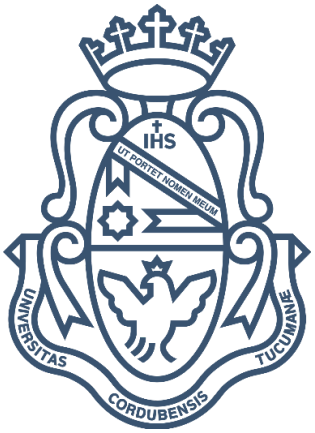


Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales;

Arquitectura de Computadoras

Profesores :

Pereyra, Martín

Alonso, Martín

Ciordia Cantarella, Francisco

Alumnos (por órden alfabético):

Badariotti, Juan miguel

Erlicher, Ezequiel

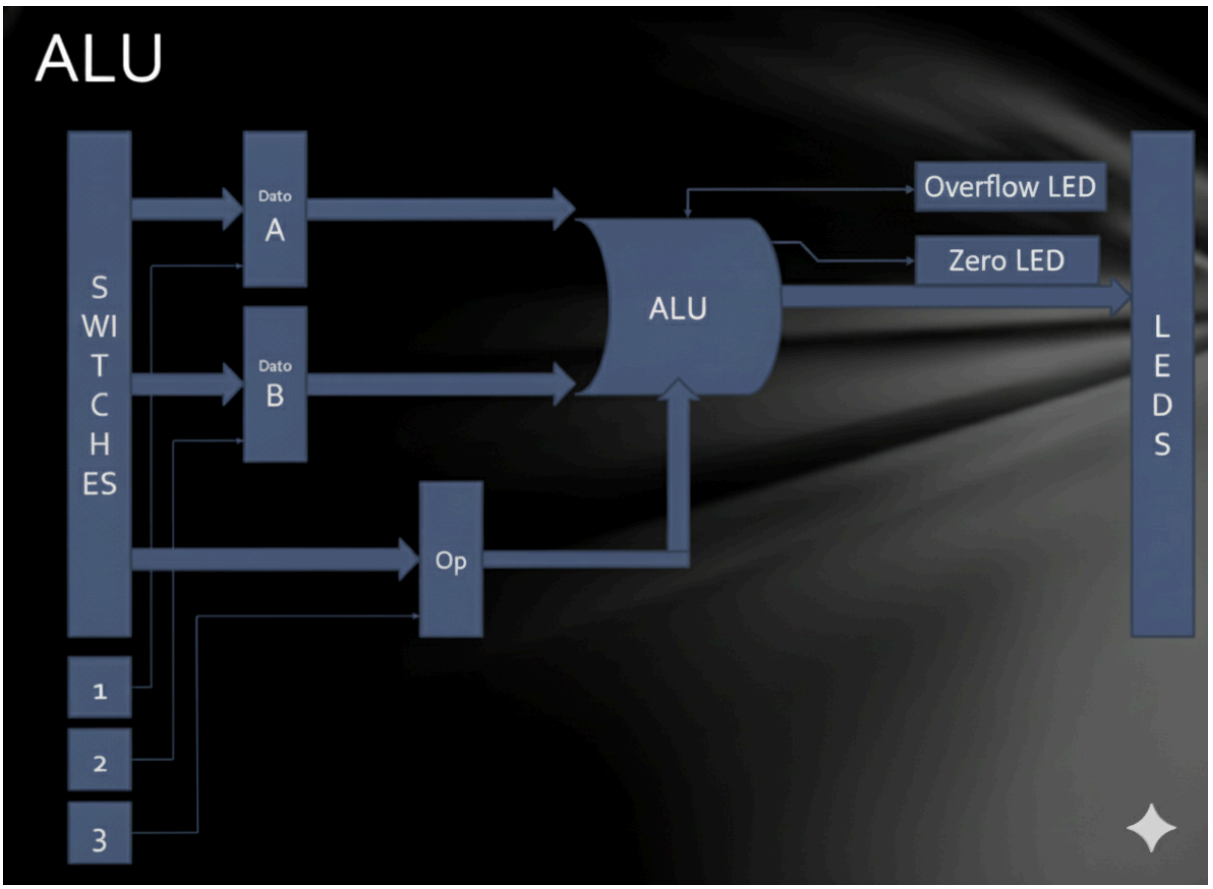
TRABAJO PRÁCTICO NÚMERO I : ALU

Descripción

Este trabajó consistió en diseñar e implementar una Arithmetic Logic Unit (ALU) en Verilog para ser sintetizada sobre una FPGA Basys3. La ALU que se implementó, permite realizar un total de 8 operaciones cada con su correspondiente código de 6 bits:

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

Los operandos ingresados son **números signados** (es decir, el primer bit indica el signo del número: 1 si es negativo y 0 en caso contrario) de 8 bits. Se configurarón en total un conjunto de 8 switches y 3 botones para ingresar el valor de los operandos y el código de operación (en el caso de este último, se usan los 6 switches asociados a los bits menos significativos). Por último, el resultado se muestra en un conjunto de 8 LEDS consecutivo más 2 LEDS extra para overflow y cero. El esquema lógico resultante es el siguiente:



Estructura del código

a) alu.v

Este archivo implementa el módulo de la alu. El tamaño de los operandos y el código de operación son parametrizables (con valor por defecto 8 y 6 bits respectivamente) mediante `NB_DATA` y `NB_OP`. Las señales de entrada y salida se definen como:

Entradas:

`i_data_a` y `i_data_b`: operandos de entrada, tratados como números con signo (signed).

`i_operation_code`: indica qué operación debe ejecutar.

Salidas:

`o_result`: resultado de la operación.

`o_overflow`: bit de overflow en suma o resta.

`o_zero`: bit cero.

```
1   module alu #(
2       parameter NB_DATA=8,
3       parameter NB_OP=6
4   )
5   (
6       input wire signed  [NB_DATA-1:0] i_data_a,
7       input wire signed  [NB_DATA-1:0] i_data_b,
8       input wire         [NB_OP-1:0]   i_operation_code,
9       output wire signed [NB_DATA-1:0] o_result,
10      output wire o_overflow,
11      output wire o_zero
12  );
```

Los códigos de cada operación se definen como constantes (`localparam`):

```
14      localparam ADD_OP = 6'b100000;
15      localparam SUB_OP = 6'b100010;
16      localparam AND_OP = 6'b100100;
17      localparam OR_OP  = 6'b100101;
18      localparam XOR_OP = 6'b100110;
19      localparam SRA_OP = 6'b000011;
20      localparam SRL_OP = 6'b000010;
21      localparam NOR_OP = 6'b100111;
```

Luego, se definen registros internos para ser usados dentro de un bloque `always` y actualizar tanto el resultado como los bits de overflow y zero. El bloque se ejecuta frente a cualquier cambio en la señales de entrada y realiza la operación correspondiente según el código de operación ingresado.

```
24      reg signed [NB_DATA-1:0] result;
25      reg ovflw;
26      reg z;
```

```
29    always @(*) begin
30
31        result  = {NB_DATA{1'b0}};
32        ovflw   = 1'b0;
33
34        case (i_operation_code)
35
36            ADD_OP: // ADD
37                begin
38                    result = i_data_a + i_data_b;
39                    ovflw = (i_data_a[NB_DATA-1]== i_data_b[NB_DATA-1]) & (i_data_a[NB_DATA-1]!=result[NB_DATA-1]);
40                end
41
42            SUB_OP: // SUB
43                begin
44                    result = i_data_a - i_data_b;
45                    ovflw = (i_data_a[NB_DATA-1]!= i_data_b[NB_DATA-1]) & (i_data_a[NB_DATA-1]!=result[NB_DATA-1]);
46                end
47
48            AND_OP: result = i_data_a & i_data_b; // AND
49
50            OR_OP: result = i_data_a | i_data_b; // OR
51
52            XOR_OP: result = i_data_a ^ i_data_b; //XOR
53
54            SRA_OP: result = i_data_a >>> i_data_b; //SRA
55
56            SRL_OP: result = i_data_a >> i_data_b; //SRL
57
58            NOR_OP: result = ~(i_data_a | i_data_b); //NOR
59
60            default: result = {NB_DATA{1'b0}};
61
62        endcase
63
64        z = (result == {NB_DATA{1'b0}});
65
66    end
```

Finalmente, se asignan los registros internos a las salidas definidas anteriormente

```
68      assign o_result = result;
69      assign o_overflow = ovflw;
70      assign o_zero = z;
71
72      endmodule
```

b) alu_tb.v

La función de este archivo es generar una serie de tests, que se corren sobre una simulación en el entorno de Vivado,para verificar el correcto funcionamiento del módulo **alu.v**

En primera instancia se definen los mismos párametros, códigos de operación y señales internas con los que la ALU trabaja:

```
7         parameter NB_DATA = 8;
8         parameter NB_OP   = 6;
9
10        // Códigos de operación
11        localparam OP_ADD = 6'b100000;
12        localparam OP_SUB = 6'b100010;
13        localparam OP_AND = 6'b100100;
14        localparam OP_OR  = 6'b100101;
15        localparam OP_XOR = 6'b100110;
16        localparam OP_SRA = 6'b000011;
17        localparam OP_SRL = 6'b000010;
18        localparam OP_NOR = 6'b100111;
19
20        reg [NB_DATA-1:0] data_a, data_b;
21        reg  [NB_OP-1:0]   op_code;
22        wire [NB_DATA-1:0] result;
23        wire overflow, zero;
```

Luego, se instancia un módulo de la ALU como UUT (Unit Under Test)

```
25        alu #(
26            .NB_DATA(NB_DATA),
27            .NB_OP(NB_OP)
28        ) uut (
29            .i_data_a(data_a),
30            .i_data_b(data_b),
31            .i_operation_code(op_code),
32            .o_result(result),
33            .o_overflow(overflow),
34            .o_zero(zero)
35        );
```

A continuación, se inicializan las señales y se espera 10 ns antes de empezar las pruebas. Se generan valores aleatorios para A y B para entonces testear todas las operaciones una por una. Para cada operación, se calcula el resultado esperado llamando a la tarea `run_test` para verificar si el resultado de la ALU coincide.

```
41          initial begin
42              // Inicialización
43              data_a = 0;
44              data_b = 0;
45              op_code = 0;
46              #10;
47
48              repeat (10) begin
49                  A = $random;
50                  B = $random;
51
52                  // Suma
53                  expected = A + B;
54                  run_test(A, B, OP_ADD, expected, "ADD");
55
56                  // Resta
57                  expected = A - B;
58                  run_test(A, B, OP_SUB, expected, "SUB");
59
60                  // AND
61                  expected = A & B;
62                  run_test(A, B, OP_AND, expected, "AND");
63
64                  // OR
65                  expected = A | B;
66                  run_test(A, B, OP_OR, expected, "OR");
67
68                  // XOR
69                  expected = A ^ B;
70                  run_test(A, B, OP_XOR, expected, "XOR");
71
72                  // SRA
73                  expected = A >>> B;
74                  run_test(A, B, OP_SRA, expected, "SRA");
75
76                  // SLA
77                  expected = A <<< B;
78                  run_test(A, B, OP_SRL, expected, "SRL");
79
80                  // NOR
81                  expected = ~(A | B);
82                  run_test(A, B, OP_NOR, expected, "NOR");
83              end
```

Por último, el testbench evalúa casos específicos donde ocurre overflow y/o operaciones con resultado igual a 0

```
// Casos específicos para overflow y zero
// Overflow positivo: 255 + 1 (para 8 bits)
run_test(255, 1, OP_ADD, 0, "ADD Overflow Positivo");
// Overflow negativo: 100 - 128 (para 8 bits)
run_test(100, -128, OP_SUB, 228, "SUB Overflow Negativo");
// Zero: 5 - 5
run_test(5, 5, OP_SUB, 0, "SUB Zero");
// Zero: 0 & 0
run_test(0, 0, OP_AND, 0, "AND Zero");

$finish;
```

c) top.v

Es el módulo superior del sistema. Se encarga de instanciar el módulo de la ALU asi como también de manejar los valores de las entradas y salidas (de acuerdo a un clock) definidas según el esquema lógico de la Imágen 1

Parámetros:

`N_SWITCHES`: cantidad de switches de entrada (por defecto 8).

`N_BUTTONS`: cantidad de botones (3 por defecto).

`NB_OPERATIONS`: cantidad de bits del código de operación (6 por defecto).

`N_LEDS`: cantidad de LEDs de salida (8 por defecto).

Entradas:

`i_clk`: reloj del sistema.

`i_sw`: conjunto de switches que ingresan datos u operaciones.

`i_button`: 3 botones usados para decidir qué estamos cargando (A, B u operación).

`reset_button`: reinicia todo a cero.

Salidas:

`o_led:` resultado de la operación de la ALU (mostrado en LEDs).

`o_overflow:` Led de overflow.

`o_zero:` Led de resultado cero.

```
1   module top #(
2       parameter N_SWITCHES=8,
3       parameter N_BUTTONS=3,
4       parameter NB_OPERATIONS=6,
5       parameter N_LEDS=8
6   )
7   (
8       input wire                  i_clk,
9       input wire  [N_SWITCHES-1:0] i_sw,
10      input wire  [N_BUTTONS-1:0]  i_button,
11      input wire                  reset_button,
12      output wire [N_LEDS-1:0]    o_led,
13      output wire                 o_overflow,
14      output wire                 o_zero
15  );
```

Luego se definen las señales internas:

`i_data_a` y `i_data_b`: operandos internos que se conectan a la ALU.

`i_operation_code`: operación que debe ejecutar la ALU.

```
17  //Internal signals
18      reg signed [N_SWITCHES-1:0] i_data_a, i_data_b;
19      reg [NB_OPERATIONS-1:0]i_operation_code;
```

Una vez hecho esto, se procede a instanciar el modulo de la ALU. Se asocian las entradas del módulo Top a las entradas de la ALU y el resultado de la operación, así como los bits de overflow y zero, a los Leds de salida correspondientes.

```
21          alu #(
22              .NB_DATA(N_SWITCHES),
23              .NB_OP(NB_OPERATIONS)
24          ) alu_unit (
25              .i_data_a(i_data_a),
26              .i_data_b(i_data_b),
27              .i_operation_code(i_operation_code),
28              .o_result(o_led),
29              .o_overflow(o_overflow),
30              .o_zero(o_zero)
31          );
```

Finalmente, se define un bloque secuencial `always` que actualiza los valores de los operandos y el código de operación en cada flanco ascendente del reloj según el botón que esté pulsado.

```
33          always @(posedge i_clk) begin
34
35              if (reset_button) begin
36                  i_data_a <= {(N_SWITCHES) {1'b0}};
37                  i_data_b <= {(N_SWITCHES) {1'b0}};
38                  i_operation_code <= {(NB_OPERATIONS) {1'b0}};
39              end
40
41              else begin
42                  if (i_button[0]) begin //Pulsador 1 = Data A
43                      i_data_a <= i_sw;
44                  end
45                  if (i_button[1]) begin //Pulsador 2 = Data B
46                      i_data_b <= i_sw;
47                  end
48                  if (i_button[2]) begin //Pulsador 3 = Operation
49                      i_operation_code <= i_sw[NB_OPERATIONS-1:0];
50                  end
51              end
52
53          end
```

d) tp1_constraints.xdc

Define los botones,switches,leds y el valor del clock que se utilizan en la placa

```
## Constraints for basys 3 FPGA

## Clock
set_property -dict { PACKAGE_PIN W5 IOSTANDARD LVCMOS33 } [get_ports { I_clk }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { I_clk }];

## Reset
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { reset_button }];

## Leds
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { O_led[0] }];
set_property -dict { PACKAGE_PIN E19 IOSTANDARD LVCMOS33 } [get_ports { O_led[1] }];
set_property -dict { PACKAGE_PIN U19 IOSTANDARD LVCMOS33 } [get_ports { O_led[2] }];
set_property -dict { PACKAGE_PIN V19 IOSTANDARD LVCMOS33 } [get_ports { O_led[3] }];
set_property -dict { PACKAGE_PIN W18 IOSTANDARD LVCMOS33 } [get_ports { O_led[4] }];
set_property -dict { PACKAGE_PIN U15 IOSTANDARD LVCMOS33 } [get_ports { O_led[5] }];
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { O_led[6] }];
set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { O_led[7] }];

## Overflow and Zero Leds
set_property -dict { PACKAGE_PIN W3 IOSTANDARD LVCMOS33 } [get_ports { O_overflow }];
set_property -dict { PACKAGE_PIN U3 IOSTANDARD LVCMOS33 } [get_ports { O_zero }];

## Buttons
set_property -dict { PACKAGE_PIN W19 IOSTANDARD LVCMOS33 } [get_ports { I_button[0] }];
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { I_button[1] }];
set_property -dict { PACKAGE_PIN T17 IOSTANDARD LVCMOS33 } [get_ports { I_button[2] }];

## Switches
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { I_sw[0] }];
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { I_sw[1] }];
set_property -dict { PACKAGE_PIN W16 IOSTANDARD LVCMOS33 } [get_ports { I_sw[2] }];
set_property -dict { PACKAGE_PIN W17 IOSTANDARD LVCMOS33 } [get_ports { I_sw[3] }];
set_property -dict { PACKAGE_PIN W15 IOSTANDARD LVCMOS33 } [get_ports { I_sw[4] }];
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { I_sw[5] }];
set_property -dict { PACKAGE_PIN W14 IOSTANDARD LVCMOS33 } [get_ports { I_sw[6] }];
set_property -dict { PACKAGE_PIN W13 IOSTANDARD LVCMOS33 } [get_ports { I_sw[7] }];
```