Arquitectura de Computadoras

Trabajo Práctico 1

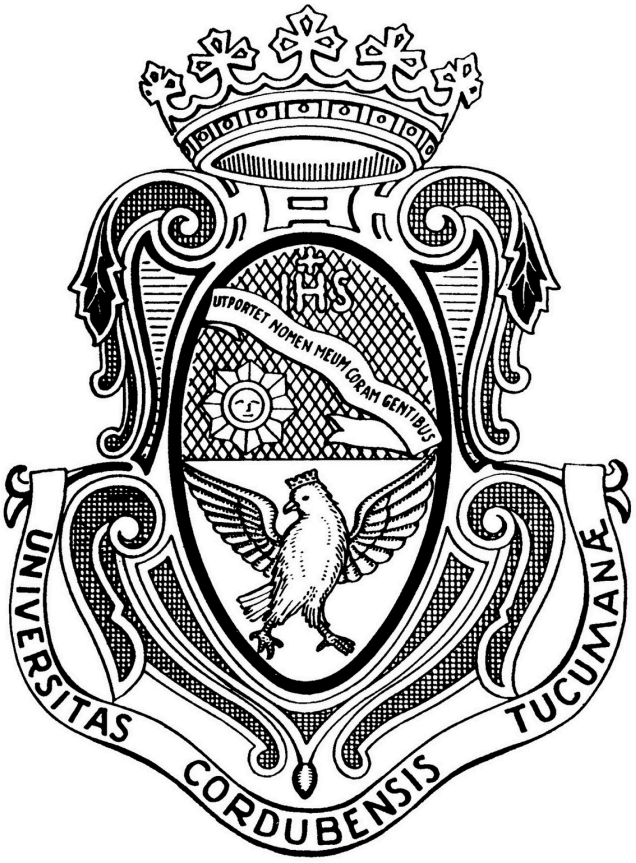
ALU

Autores:

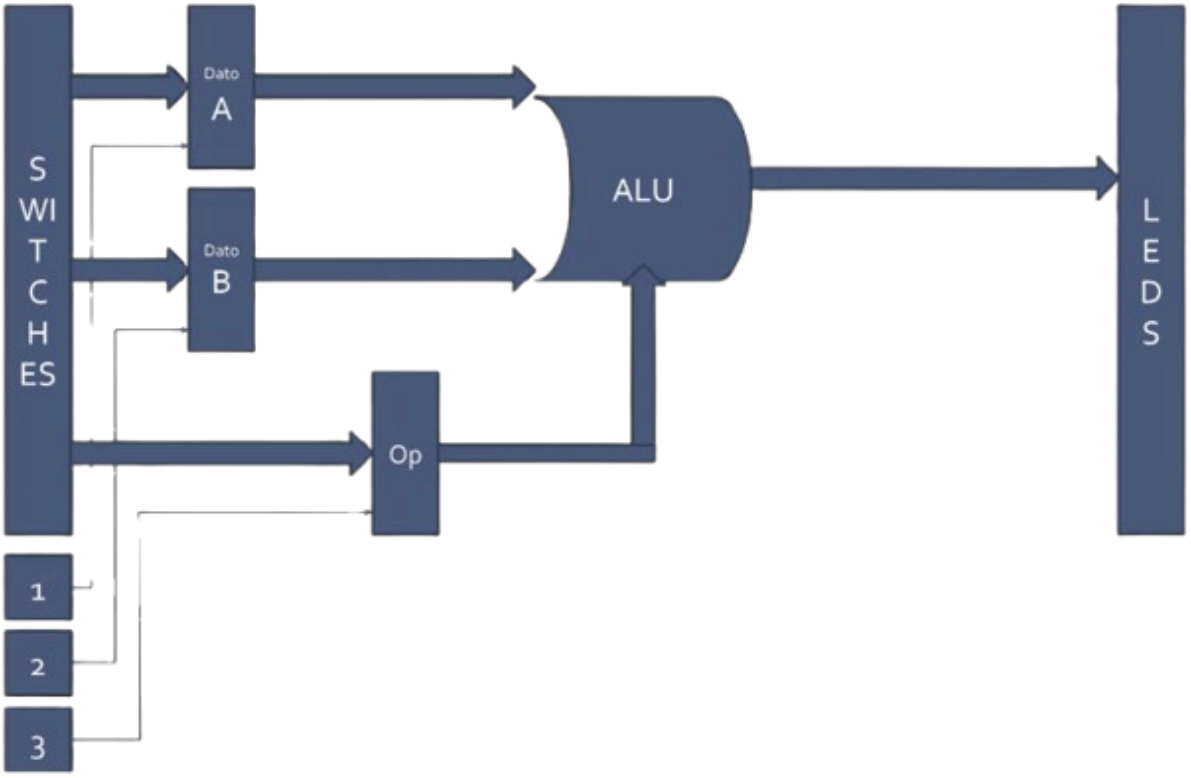
BORGATELLO, Ignacio

DALLARI LARROSA, Gian Franco

2025



1. CONSIGNA



Implementar en FPGA una ALU.

1. La ALU debe ser parametrizable (bus de datos) para poder ser utilizada posteriormente en el trabajo final.
2. Validar el desarrollo por medio de Test Bench. El testbench debe incluir generación de entradas aleatorias y código de chequeo automático.
3. Simular el diseño usando las herramientas de simulación de vivado.

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

2. DESARROLLO

2.1 Módulo Top

El módulo Top recibe entradas de botones e interruptores, los cuales son procesados por el módulo Latch para generar operandos y un código de operación. Estos se pasan a la ALU, que realiza una operación aritmética o lógica y envía el resultado a la salida.

```
`timescale 1ns / 1ps
module Top(
    input wire i_boton1, i_boton2, i_boton3, i_boton4,
    input wire [5:0] i_switches,
    input wire i_clk,
    output wire [3:0] o_alu_output
);

     wire [3:0] o_operandoA;
     wire [3:0] o_operandoB;
     wire [5:0] o_codigoOperacion;

    // Instancia el módulo de control de registros
    Latch instanciaLatch (
        .i_btn1(i_boton1),
        .i_btn2(i_boton2),
        .i_btn3(i_boton3),
        .i_btn4(i_boton4),
        .i_sw(i_switches),
        .i_clk(i_clk),
        .o_opA(o_operandoA),
        .o_opB(o_operandoB),
        .o_opcode(o_codigoOperacion)
    );

    // Instancia el módulo de la ALU
    ALU instanciaALU (
        .i_operandoA(o_operandoA),
        .i_operandoB(o_operandoB),
        .i_operacion(o_codigoOperacion),
        .o_resultado(o_alu_output)
    );
endmodule
```

2.2 Módulo Latch

El módulo Latch almacena valores en registros controlados por botones y una señal de reloj. Los valores de entrada provienen de i_sw y se guardan en tres registros: uno para el operando A, otro para el operando B, y otro para el código de operación. Estos registros se actualizan en un flanco positivo del reloj dependiendo de qué botón se presione. Por ejemplo, al presionar el botón 1, el valor de los interruptores se guarda en el operando A, y al presionar el botón 2, el valor se guarda en el operando B. El botón 3 almacena el valor en el registro del código de operación. Si se presiona el botón 4, todos los registros se reinician a cero. Las salidas del módulo corresponden a los valores almacenados en estos registros.

```
`timescale 1ns / 1ps

module Latch(
    input i_btn1, i_btn2, i_btn3, i_btn4,
    input [5:0] i_sw,
    input i_clk,
    output [3:0] o_opA,
    output [3:0] o_opB,
    output [5:0] o_opcode
);

    reg [3:0] reg_operandoA;
    reg [3:0] reg_operandoB;
    reg [5:0] reg_opcode;

    always @(posedge i_clk) begin
            if (i_btn1) begin
                reg_operandoA <= i_sw;
            end
            if (i_btn2) begin
                reg_operandoB <= i_sw;
            end
            if (i_btn3) begin
                reg_opcode <= i_sw;
            end
            if (i_btn4) begin
                reg_operandoA <= 4'b0;
                reg_operandoB <= 4'b0;
                reg_opcode <= 6'b0;
        end
    end

    assign o_opA = reg_operandoA;
    assign o_opB = reg_operandoB;
    assign o_opcode = reg_opcode;
endmodule
```

2.3 Módulo ALU

La ALU realiza diferentes operaciones entre dos operandos de 4 bits dependiendo del código de operación de 6 bits seleccionado. El resultado luego se muestra en los LEDS de la placa.

Las operaciones que realiza incluyen suma, resta, AND, OR, XOR, desplazamiento a la izquierda, desplazamiento a la derecha y NOR. Si el código de operación no coincide con ninguna operación definida, el resultado es cero. Las operaciones se seleccionan mediante una estructura case, que determina el comportamiento de la ALU según el valor del código de operación.

```
`timescale 1ns / 1ps

module ALU(i_operandoA,i_operandoB,i_operacion,o_resultado);

  parameter tamanioEntrada=4;
  parameter tamanioSalida=4;
  parameter tamanioOperacion=6;


  input wire [tamanioEntrada-1:0] i_operandoA;
  input wire [tamanioEntrada-1:0] i_operandoB;
  input wire [tamanioOperacion-1:0] i_operacion;
  output wire [tamanioSalida-1:0] o_resultado;

  reg [tamanioSalida-1:0] temp;

  always @(*)
  begin
    case(i_operacion)
      6'b100000 : temp = i_operandoA + i_operandoB;
      6'b100010 : temp = i_operandoA - i_operandoB;
      6'b100100 : temp = i_operandoA & i_operandoB;
      6'b100101 : temp = i_operandoA | i_operandoB;
      6'b100110 : temp = i_operandoA ^ i_operandoB;
      6'b000011 : temp = i_operandoA << i_operandoB;
      6'b000010 : temp = i_operandoA >> i_operandoB;
      6'b100111 : temp = ~(i_operandoA | i_operandoB);
      default   : temp = {tamanioSalida{1'b0}};
    endcase
  end

  assign o_resultado = temp;

endmodule
```

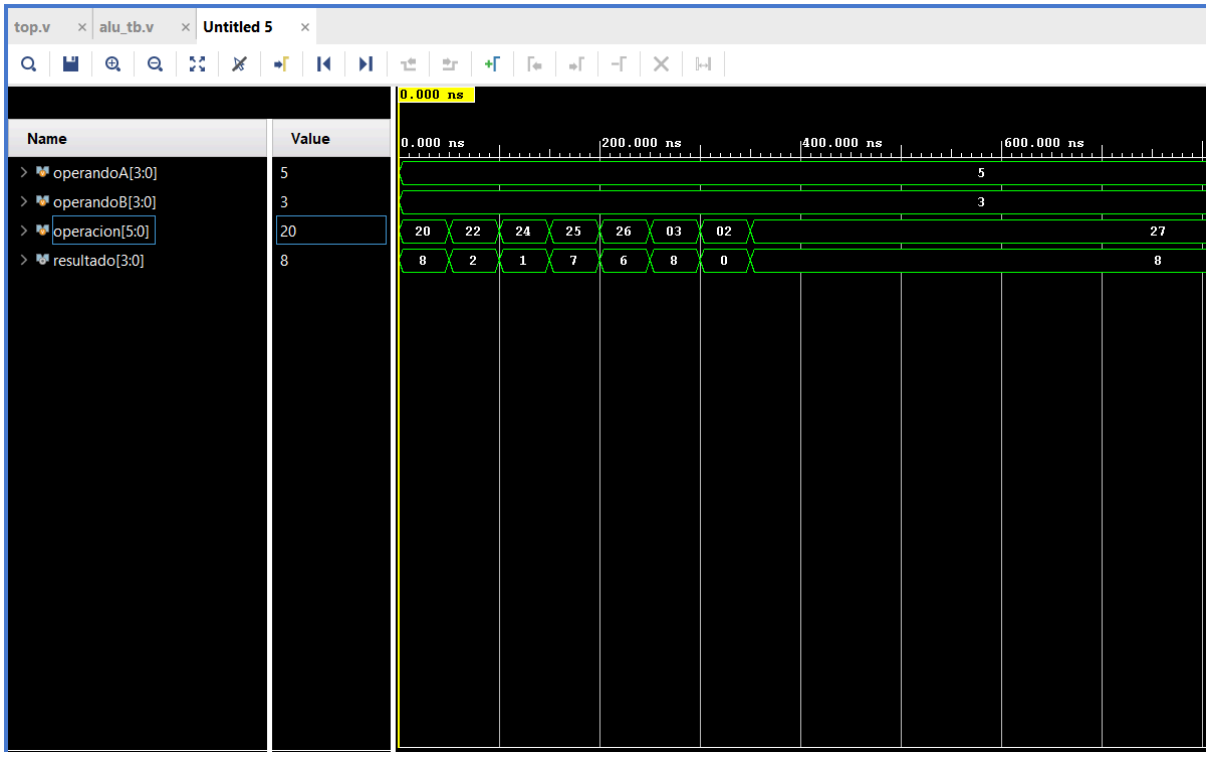
2.4 Archivo ‘Constraints’

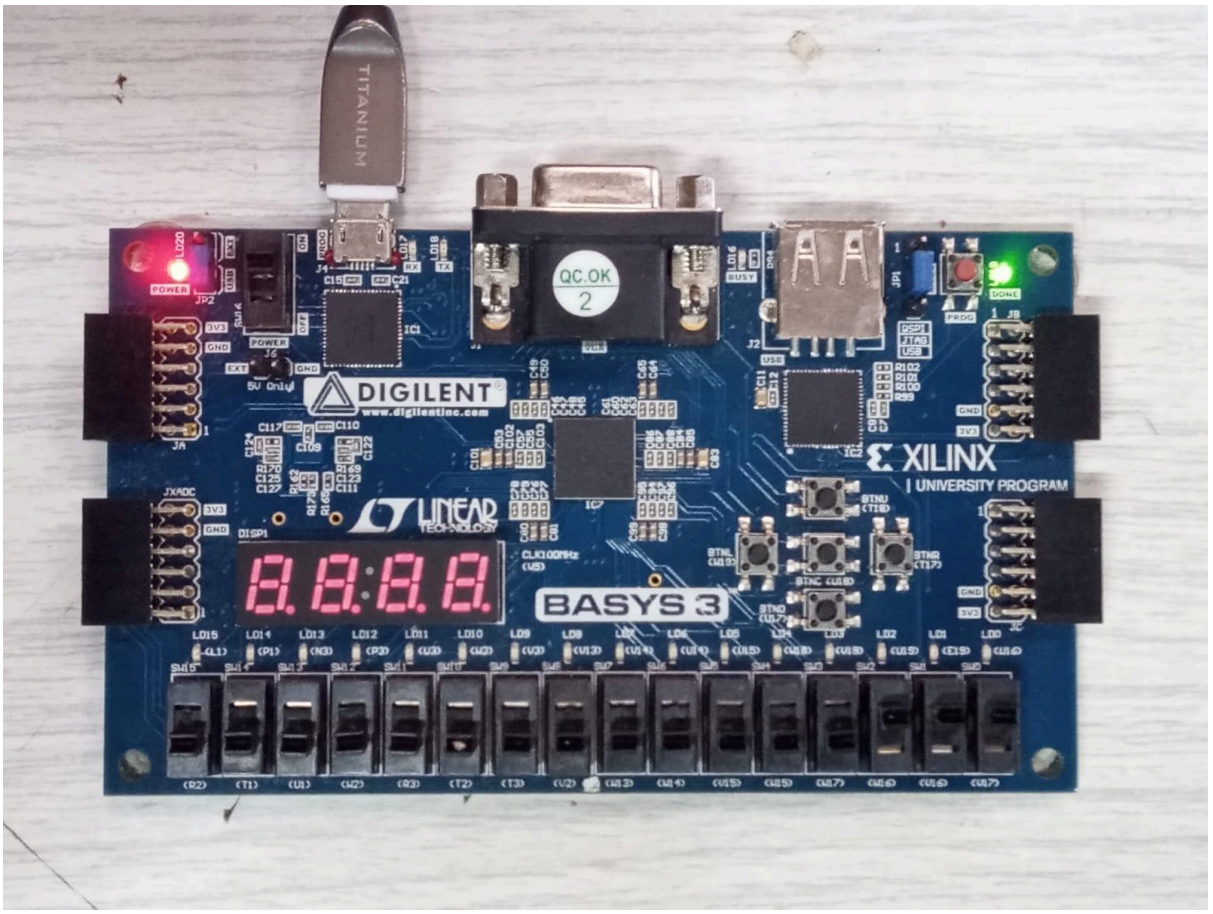
El propósito del archivo Constraints es asignar los pines físicos de la FPGA a señales lógicas de nuestro diseño, especificar la configuración del reloj y definir las opciones de configuración del sistema.

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level
signal names in the project

## Clock signal
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]

## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {i_switches[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {i_switches[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {i_switches[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {i_switches[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {i_switches[4]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {i_switches[5]}]

## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {o_alu_output[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {o_alu_output[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {o_alu_output[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {o_alu_output[3]}]

##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports i_boton4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports i_boton3]
set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports i_boton1]
set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports i_boton2]


## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

2.5 Pruebas y resultados

```
`timescale 1ns / 1ps

module alu_tb;

    reg  [3:0] operandoA;
    reg  [3:0] operandoB;
    reg  [5:0] operacion;
    wire [3:0] resultado;

    // Instancia de la ALU
    alu DUT (
        .i_operandoA(operandoA),
        .i_operandoB(operandoB),
        .i_operacion(operacion),
        .o_resultado(resultado)
    );

    initial begin
        // Inicialización
        operandoA = 4'b0101; // 5
        operandoB = 4'b0011; // 3

        // Pruebas de operaciones
        operacion = 6'b100000; #50; // ADD
        operacion = 6'b100010; #50; // SUB
        operacion = 6'b100100; #50; // AND
        operacion = 6'b100101; #50; // OR
        operacion = 6'b100110; #50; // XOR
        operacion = 6'b000011; #50; // SLL
        operacion = 6'b000010; #50; // SRL
        operacion = 6'b100111; #50; // NOR

        // Terminar simulación
        #10 $finish;
    end

    // Monitoreo en consola
    initial begin
        $monitor("t=%0t | A=%b | B=%b | op=%b | R=%b",
                 $time, operandoA, operandoB, operacion, resultado);
    end

endmodule
```





2.7 Repositorio de GitHub

https://github.com/Cabra98/TP1-Arquitectura