# 

# **Facultad de Ingeniería**

### **Ingeniería Informática**

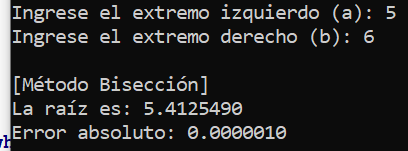
Guia N°1

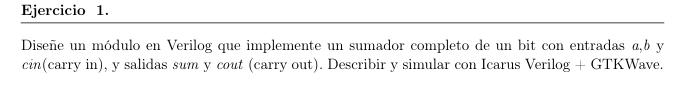
### **Arquitectura de Computadoras I**

### **Año Académico 2025**

# **Autor:**

# Quinteros Marcos





module ejercicio1 ( input a , input b , input cin , output sum , output co ) ;

assign sum = a ^ b ^ cin ;

assign co = ( a & b ) | cin & ( a ^ b ) ;

endmodule

TESTBENCH

`timescale 1ns / 1ps // 1 Nanosegundo / 1 Picosegundo

`include "ejercicio1.v"

// Testbench

module ejercicio1\_tb;

// Declaración de las señales de entrada y de salida

reg a; // Entrada A

reg b; // Entrada B

reg cin;

wire sum; // Salida sum

wire co;

// Instanciación del módulo or\_gate

ejercicio1 uut (

.a(a), // Conecta A al módulo

.b(b), // Conecta B al módulo

.cin(cin), // Conecta cin al módulo

.sum(sum),

.co(co)

);

// Generaciòn de combinaciones para las entradas

initial begin

$dumpfile("ejercicio1\_tb.vcd");

$dumpvars(0, ejercicio1\_tb);

// Mostrar Cabecera

$display ("Time | A | B | Cin | Sum | Co");

$display ("------------------------------");

// Aplicación de las combinaciones

a = 0; b = 0; cin = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b | %b ", $time, a, b, cin, sum, co);

a = 0; b = 1; cin = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b | %b ", $time, a, b, cin, sum, co);

a = 1; b = 0; cin = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b | %b ", $time, a, b, cin, sum, co);

a = 1; b = 1; cin = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b | %b ", $time, a, b, cin, sum, co);

a = 0; b = 0; cin = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b | %b ", $time, a, b, cin, sum, co);

a = 0; b = 1; cin = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b | %b ", $time, a, b, cin, sum, co);

a = 1; b = 0; cin = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b | %b ", $time, a, b, cin, sum, co);

a = 1; b = 1; cin = 1; #10; // Espera 10 unidades de tiempo

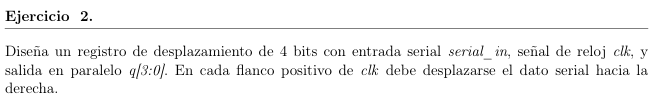
$display ("%0t | %b | %b | %b | %b | %b ", $time, a, b, cin, sum, co);

$stop; // Detener la simulación

end

endmodule





module ejercicio2(input wire clk,input wire serial\_in,output reg [3:0] q);

always @(posedge clk) begin

q <= {serial\_in, q[3:1]}; // desplaza a la derecha, entra serial\_in en MSB

end

endmodule

`timescale 1ns / 1ps // 1 Nanosegundo / 1 Picosegundo

`include "ejercicio2.v"

// Testbench

module ejercicio2\_tb;

// Declaración de las señales de entrada y de salida

reg clk; // Entrada A

reg serial\_in; // Entrada B

wire [3:0] q; // Salida

// Instanciación del módulo Shift\_register

ejercicio2 uut (

.clk(clk), // Conecta clk al módulo

.serial\_in(serial\_in), // Conecta serial\_in al módulo

.q(q) // Conecta q al módulo

);

// Generaciòn de combinaciones para las entradas

initial begin

$dumpfile("ejercicio2\_tb.vcd");

$dumpvars(0, ejercicio2\_tb);

// Mostrar Cabecera

$display ("Time | Serial\_in | clk | Q");

$display ("------------------------------");

// Aplicación de las combinaciones

serial\_in = 0; clk = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

serial\_in = 0; clk = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

serial\_in = 1; clk = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

serial\_in = 1; clk = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

serial\_in = 0; clk = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

serial\_in = 0; clk = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

serial\_in = 1; clk = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

serial\_in = 1; clk = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

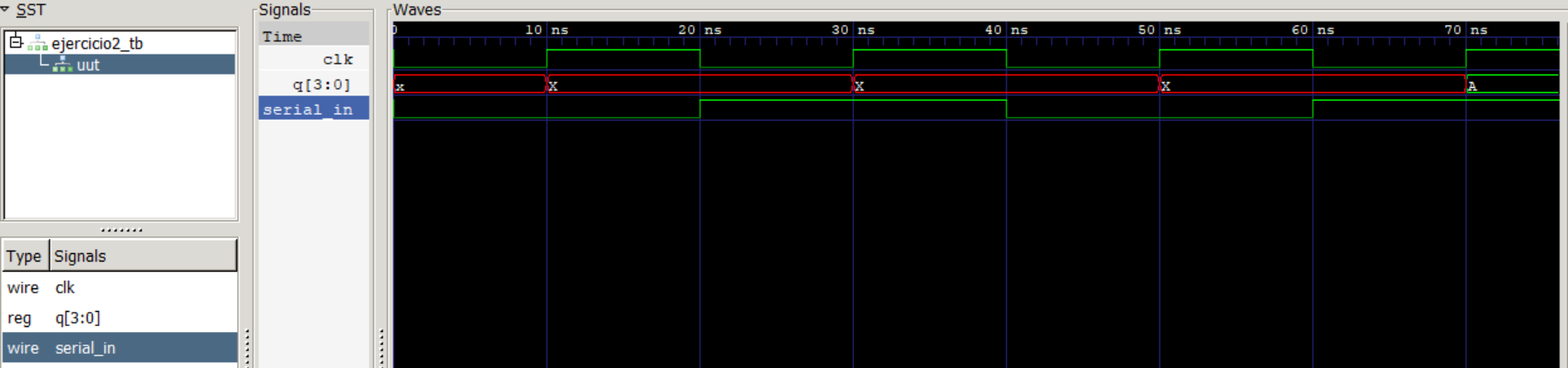
serial\_in = 0; clk = 0; #10; // Espera 10 unidades de tiempo

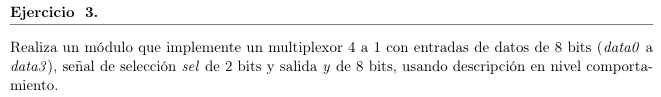
$display ("%0t | %b | %b | %b ", $time, serial\_in, clk, q);

$stop; // Detener la simulación

end

endmodule





module ejercicio3 (

input wire [7:0] data0,

input wire [7:0] data1,

input wire [7:0] data2,

input wire [7:0] data3,

input wire [1:0] sel,

output reg [7:0] y

);

always @(\*) begin

case (sel)

2'b00: y = data0;

2'b01: y = data1;

2'b10: y = data2;

2'b11: y = data3;

default: y = 8'b0;

endcase

end

endmodule

TESTBENCH

`timescale 1ns / 1ps // 1 Nanosegundo / 1 Picosegundo

`include "ejercicio3.v"

// Testbench

module ejercicio3\_tb;

// Declaración de las señales de entrada y de salida

reg [7:0] data0;

reg [7:0] data1;

reg [7:0] data2;

reg [7:0] data3;

reg [1:0] sel; // Entrada de selección

wire [7:0] y; // Salida

// Instanciación del módulo Shift\_register

ejercicio3 uut (

.data0(data0),

.data1(data1),

.data2(data2),

.data3(data3),

.sel(sel),

.y(y)

);

// Generaciòn de combinaciones para las entradas

initial begin

$dumpfile("ejercicio3\_tb.vcd");

$dumpvars(0, ejercicio3\_tb);

//Mostrar Cabecera

$display ("Time | Data0 | Data1 | Data2 | Data3 | Sel | Y");

$display ("---------------------------------------------");

// Aplicación de las combinaciones

// Caso 1

data0 = 8'b00000001; data1 = 8'b00000010; data2 = 8'b00000100; data3 = 8'b00001000; sel = 2'b00; #10;

$display ("%0t | %b | %b | %b | %b | %b | %b", $time, data0, data1, data2, data3, sel, y);

// Caso 2

sel = 2'b01; #10;

$display ("%0t | %b | %b | %b | %b | %b | %b", $time, data0, data1, data2, data3, sel, y);

// Caso 3

sel = 2'b10; #10;

$display ("%0t | %b | %b | %b | %b | %b | %b", $time, data0, data1, data2, data3, sel, y);

// Caso 4

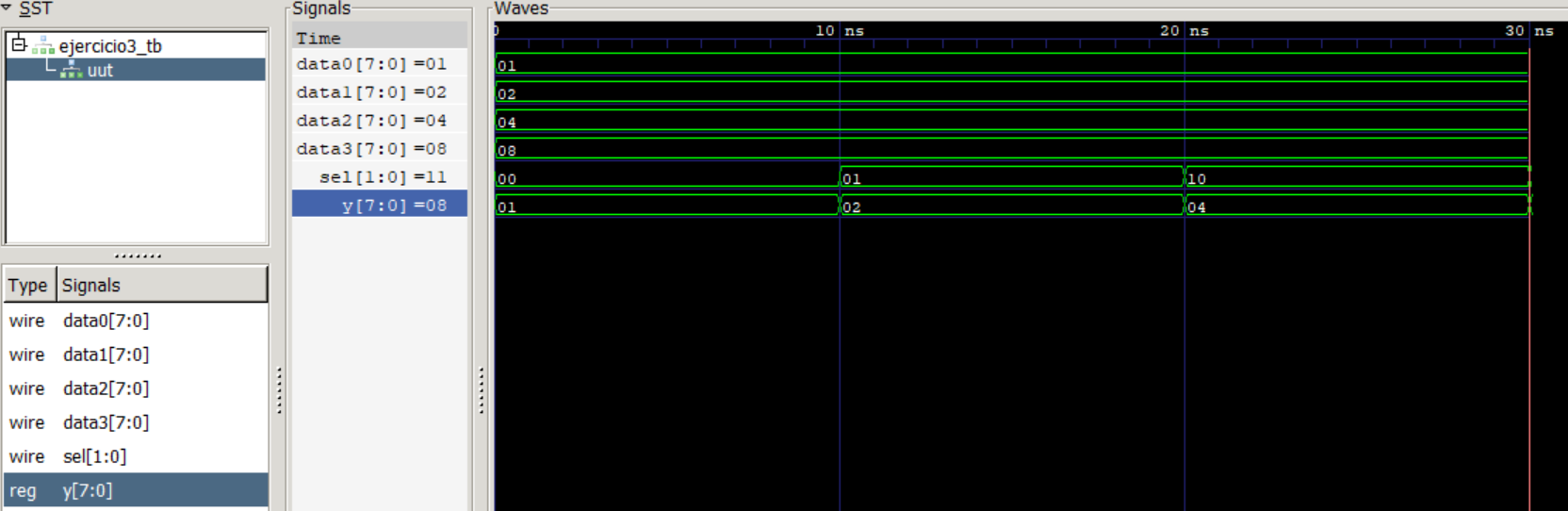
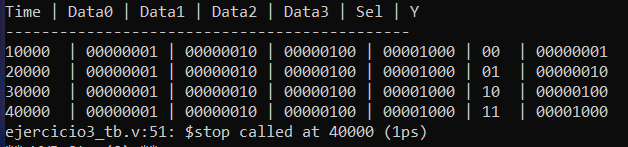
sel = 2'b11; #10;

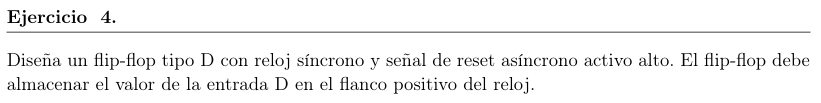
$display ("%0t | %b | %b | %b | %b | %b | %b", $time, data0, data1, data2, data3, sel, y);

$stop; // Detener la simulación

end

endmodule



module ejercicio4(input wire clk, input wire d, input wire reset, output reg q);

always @(posedge clk or posedge reset) begin

if (reset)

begin

q <= 1'b0; // reset asíncrono: Q -> 0

end

else begin

q <= d; // en flanco positivo de clk: capturar D

end

end

endmodule

TESTBENCH

`timescale 1ns / 1ps // 1 Nanosegundo / 1 Picosegundo

`include "ejercicio4.v"

// Testbench

module ejercicio4\_tb;

// Declaración de las señales de entrada y de salida

reg clk;

reg d;

reg reset;

wire q;

// Instanciación del Multiplexor

ejercicio4 uut (

.clk(clk), // Entrada de reloj

.d(d), // Entrada d

.reset(reset), // Entrada de reseteo

.q(q) // Salida

);

// Generador de reloj

initial begin

clk = 0;

forever #5 clk = ~clk; // Periodo de 10ns

end

// Generaciòn de combinaciones para las entradas

initial begin

$dumpfile("ejercicio4\_tb.vcd");

$dumpvars(0, ejercicio4\_tb);

// Mostrar Cabecera

$display ("Time | clk | d | reset | q");

$display ("--------------------------------");

// Aplicación de las combinaciones

//Caso 1

clk = 0; d = 0; reset = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b", $time, clk, d, reset, q);

//Caso 2

clk = 0; d = 0; reset = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b", $time, clk, d, reset, q);

//Caso 3

clk = 0; d =1; reset = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b", $time, clk, d, reset, q);

//Caso 4

clk = 0; d = 1; reset = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b", $time, clk, d, reset, q);

//Caso 5

clk = 1; d = 0; reset = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b", $time, clk, d, reset, q);

//Caso 6

clk = 1; d = 0; reset = 1; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b", $time, clk, d, reset, q);

//Caso 7

clk = 1; d = 1; reset = 0; #10; // Espera 10 unidades de tiempo

$display ("%0t | %b | %b | %b | %b", $time, clk, d, reset, q);

//Caso 8

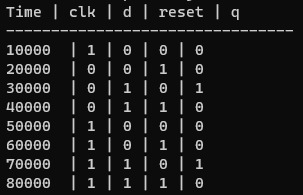
clk = 1; d = 1; reset = 1; #10; // Espera 10 unidades de tiempo

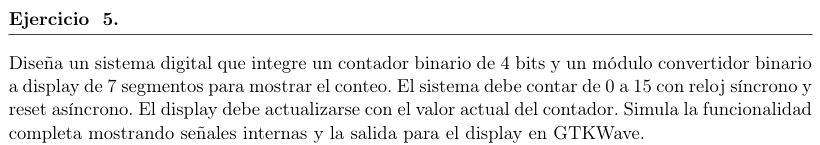
$display ("%0t | %b | %b | %b | %b", $time, clk, d, reset, q);

$stop; // Detener la simulación

end

endmodule



module ejercicio6 (input wire [3:0] a, input wire [3:0] b, input wire [1:0] opcode, output reg[3:0]result, output reg zero);

always @(\*)begin

case (opcode)

2'b00: result = a + b; // Suma

2'b01: result = a - b; // Resta

2'b10: result = a & b; // AND

2'b11: result = a | b; // OR

default: result = 4'b0000; // Valor por defecto

endcase

// Actualizar el flag de cero

zero = (result == 4'b0000); //si result es cero -> zero = 1, si result != 0 -> zero = 0

end

endmodule

TESTBENCH

`timescale 1ns / 1ps // 1 Nanosegundo / 1 Picosegundo

`include "ejercicio6.v"

// Testbench

module ejercicio6\_tb;

// Declaración de las señales de entrada y de salida

reg [3:0] a;

reg [3:0] b;

reg [1:0] opcode;

wire [3:0] result; // Salida

wire zero; // Flag de cero

// Instanciación del módulo ejercicio6

ejercicio6 uut (

.a(a),

.b(b),

.opcode(opcode),

.result(result),

.zero(zero)

);

// Generaciòn de combinaciones para las entradas

initial begin

$dumpfile("ejercicio6\_tb.vcd");

$dumpvars(0, ejercicio6\_tb);

$display("Time | a | b | opcode | result | zero");

$display("-----------------------------------------------");

// Aplicación de combinaciones

a = 4'b0000; b = 4'b0000; opcode = 2'b00; #10; $display("%4dns | %b | %b | %b | %b | %b", $time, a, b, opcode, result, zero);

a = 4'b0011; b = 4'b0011; opcode = 2'b01; #10; $display("%4dns | %b | %b | %b | %b | %b", $time, a, b, opcode, result, zero);

a = 4'b0101; b = 4'b0011; opcode = 2'b00; #10; $display("%4dns | %b | %b | %b | %b | %b", $time, a, b, opcode, result, zero);

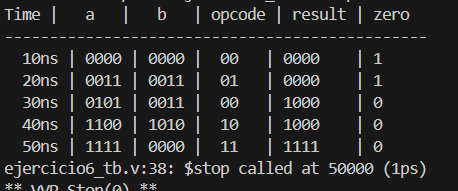
a = 4'b1100; b = 4'b1010; opcode = 2'b10; #10; $display("%4dns | %b | %b | %b | %b | %b", $time, a, b, opcode, result, zero);

a = 4'b1111; b = 4'b0000; opcode = 2'b11; #10; $display("%4dns | %b | %b | %b | %b | %b", $time, a, b, opcode, result, zero);

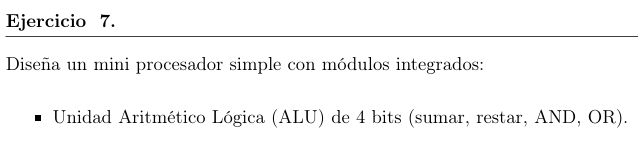
$stop; // Detener la simulación

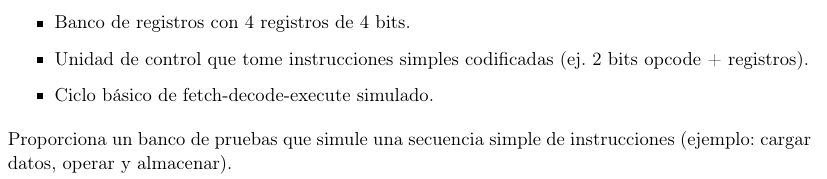
end

endmodule



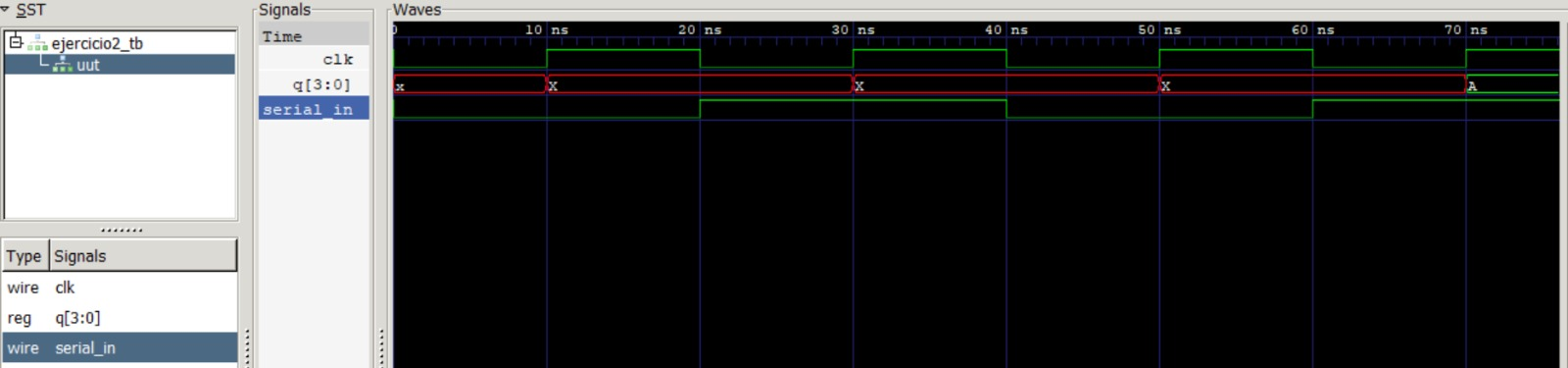






CONSULTA

ejercicio 2



como se lee esto, y q se lee con el q[3:1]

Ejercicio 6.

Diseña una ALU de 4 bits que realice las operaciones básicas: suma, resta, AND, OR entre dos

operandos de 4 bits (a y b), seleccionando la operación mediante una señal de 2 bits opcode. Salida

result y flag de cero zero.

ejercicio 6 grafico  


Ejercicio 7 (como hacer) y Ejercicio 5 (como hacer)(porque no me ejecuta el vvp, como hago cuando son varios módulos en un mismo archivo verilog)