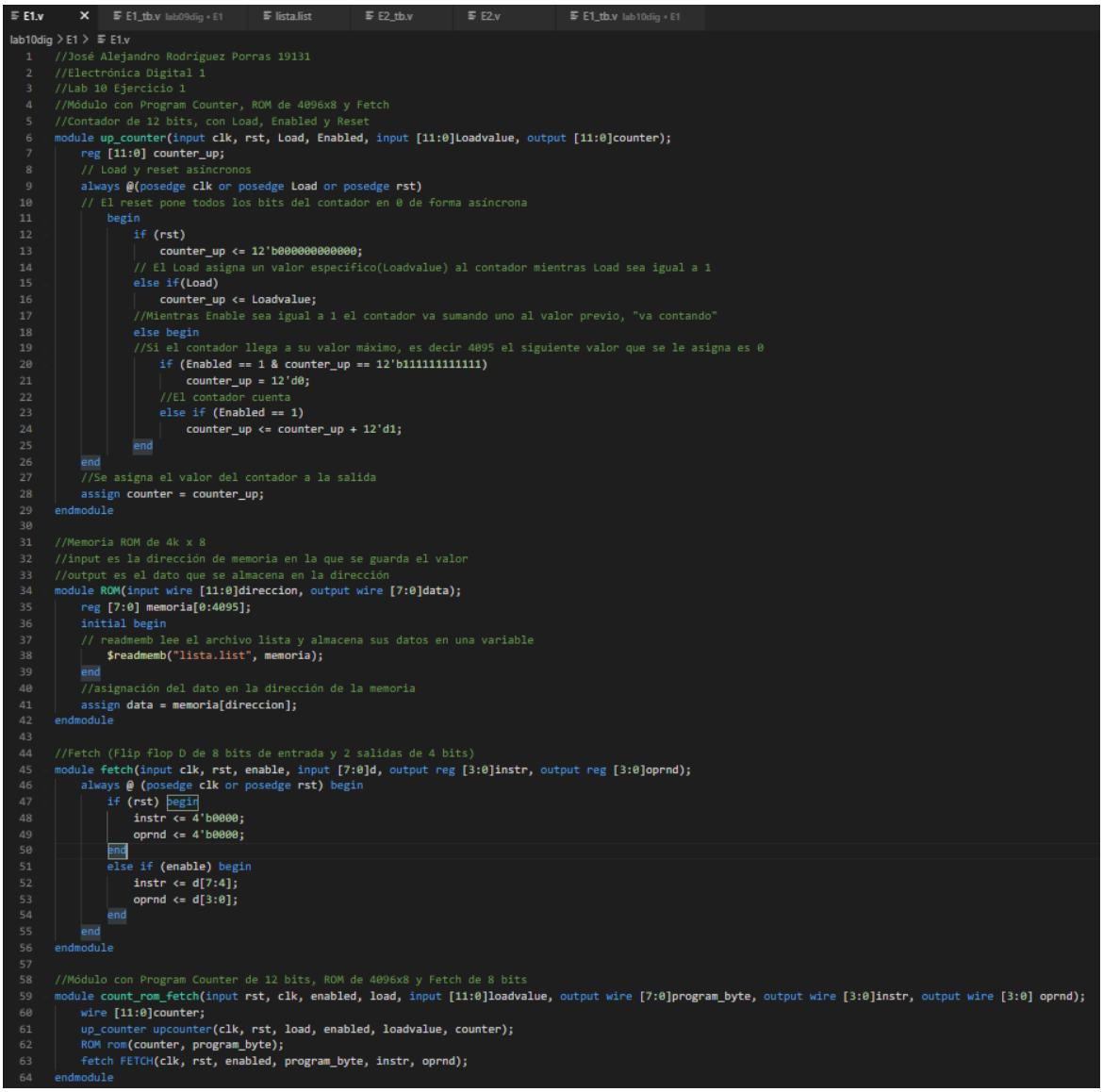
Nombre: José Alejandro Rodríguez Porras Carné: 19131 Fecha: 7/11/2020

Laboratorio 10

Ejercicio 1

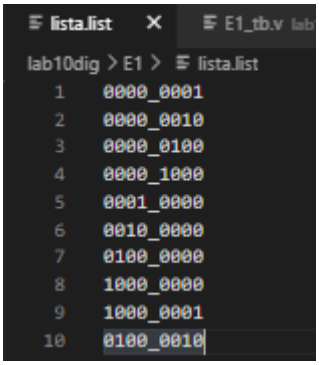


```
//José Alejandro Rodriguez Porras 19131
//Electrónica Digital 1
//Lab 10 Ejercicio 1
//Módulo con Program Counter, ROM de 4096x8 y Fetch
//Contador de 12 bits, con Load, Enabled y Reset
module up_counter(input clk, rst, Load, Enabled, input [11:0]Loadvalue, output [11:0]counter);
    reg [11:0] counter_up;
    // Load y reset asincronos
    always @(posedge clk or posedge Load or posedge rst)
    // El reset pone todos los bits del contador en 0 de forma asincrona
        begin
            if (rst)
                counter_up <= 12'b000000000000;
            // El Load asigna un valor especifico(Loadvalue) al contador mientras Load sea igual a 1
            else if(Load)
                counter_up <= Loadvalue;
            //Mientras Enable sea igual a 1 el contador va sumando uno al valor previo, "va contando"
            else begin
            //Si el contador llega a su valor máximo, es decir 4095 el siguiente valor que se le asigna es 0
                if (Enabled == 1 & counter_up == 12'b111111111111)
                    counter_up = 12'd0;
                //El contador cuenta
                else if (Enabled == 1)
                    counter_up <= counter_up + 12'd1;
            end
    end
    //Se asigna el valor del contador a la salida
    assign counter = counter_up;
endmodule

//Memoria ROM de 4k x 8
//input es la dirección de memoria en la que se guarda el valor
//output es el dato que se almacena en la dirección
module ROM(input wire [11:0]direccion, output wire [7:0]data);
    reg [7:0] memoria[0:4095];
    initial begin
    // readmemb lee el archivo lista y almacena sus datos en una variable
        $readmemb("lista.list", memoria);
    end
    //asignación del dato en la dirección de la memoria
    assign data = memoria[direccion];
endmodule

//Fetch (Flip flop D de 8 bits de entrada y 2 salidas de 4 bits)
module fetch(input clk, rst, enable, input [7:0]d, output reg [3:0]instr, output reg [3:0]oprnd);
    always @ (posedge clk or posedge rst) begin
        if (rst) begin
            instr <= 4'b0000;
            oprnd <= 4'b0000;
        end
        else if (enable) begin
            instr <= d[7:4];
            oprnd <= d[3:0];
        end
    end
endmodule

//Módulo con Program Counter de 12 bits, ROM de 4096x8 y Fetch de 8 bits
module count_rom_fetch(input rst, clk, enabled, load, input [11:0]loadvalue, output wire [7:0]program_byte, output wire [3:0]instr, output wire [3:0] oprnd);
    wire [11:0]counter;
    up_counter upcounter(clk, rst, load, enabled, loadvalue, counter);
    ROM rom(counter, program_byte);
    fetch FETCH(clk, rst, enabled, program_byte, instr, oprnd);
endmodule
```

Código del módulo count_rom_fetch

En esta imagen se puede observar la implementación de un módulo que contiene los módulos hechos en los laboratorios previos. El módulo principal tenía como función implementar los submódulos de program counter, la memoria ROM de 4k x 8, y el fetch de 8 bits. Los inputs de módulo fueron el rst, clk, enabled, load y los 12 bits de program counter para hacer loads. Las salidas fueron los 8 bits del program_byte, 4 bits de instrucción y 4 bits de operando. La salida del program counter se implementó de forma que fuera la entrada del program ROM, y la salida de este, es decir los 8 bits del program_byte era los inputs del fetch, que se

implementó como un flip flop tipo D con 2 salidas, 4 bits de instrucción y 4 de operando.



lista.list

lab10dig > E1 > lista.list

```
0000_0001
0000_0010
0000_0100
0000_1000
0001_0000
0010_0000
0100_0000
1000_0000
1000_0001
0100_0010
```

Archivo lista para la ROM.

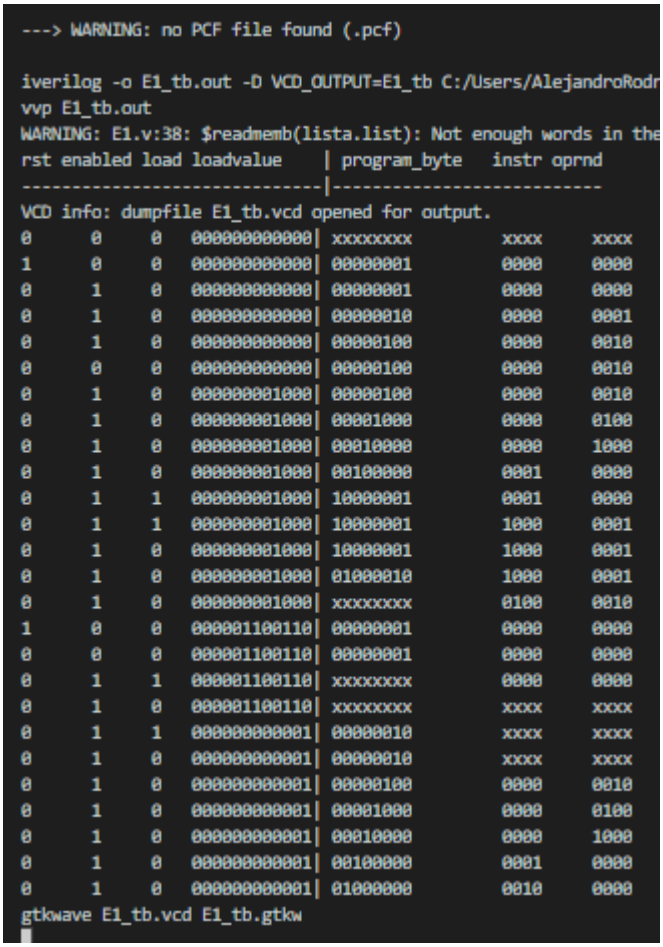
E1_tb.v lab10dig • E1 | E1.v | E1_tb.v lab09dig • E1 | lista.list | E2_tb.v | E2.v

lab10dig > E1 > E1_tb.v

```
//José Alejandro Rodríguez Porras 19131
//Electrónica Digital 1
//Lab 10 Ejercicio 1 Testbench
//Módulo con Program Counter, ROM de 4096x8 y Fetch
module testbench();
    //Declaración de variables
    reg clk, rst, enabled, load;
    reg [11:0]loadvalue;
    wire [7:0]program_byte;
    wire [3:0]instr;
    wire [3:0]oprnd;
    //Asignación de módulos
    count_rom_fetch COUNT_ROM_FETCHU(rst, clk, enabled, load, loadvalue, program_byte, instr, oprnd);
    //Prueba de funcionamiento del Módulo con counter, rom y fecth
    initial begin
        $display("rst enabled load loadvalue    | program_byte   instr oprnd");
        $display("------------------------------|--------------------------");
        $monitor("%b      %b     %b   %b| %b         %b     %b", rst, enabled, load, loadvalue, program_byte, instr, oprnd);
        clk = 0; rst = 0; enabled = 0; load = 0; loadvalue = 12'b000000000000;
        //Prueba del reset
        #1   rst = 1; enabled = 0; load = 0; loadvalue = 12'b000000000000;
        #1   rst = 0; enabled = 1; load = 0; loadvalue = 12'b000000000000;
        //Prueba del enable en 0
        #20  rst = 0; enabled = 0; load = 0; loadvalue = 12'b000000000000;
        #20  rst = 0; enabled = 0; load = 0; loadvalue = 12'b000000000000;
        //Prueba del enable en 1, comienza a contar, y loadvalue no se carga si load es 0
        #20  rst = 0; enabled = 1; load = 0; loadvalue = 12'b000000001000;
        //Prueba de la carga de un valor loadvalue con load igual a 1
        #30  rst = 0; enabled = 1; load = 1; loadvalue = 12'b000000001000;
        //Prueba de que comienza a contar desde la dirección que se cargó
        #30  rst = 0; enabled = 1; load = 0; loadvalue = 12'b000000001000;
        //Prueba del reset
        #20  rst = 1; enabled = 0; load = 0; loadvalue = 12'b000001100110;
        #1   rst = 0; enabled = 0; load = 0; loadvalue = 12'b000001100110;
        //Prueba de la carga de un valor de una dirección de la ROM que no tiene ningún valor guardado
        #1   rst = 0; enabled = 1; load = 1; loadvalue = 12'b000001100110;
        #1   rst = 0; enabled = 1; load = 0; loadvalue = 12'b000001100110;
        //Prueba de la carga de un valor cualquiera de una dirección de la ROM que si tiene un valor guardado
        #1   rst = 0; enabled = 1; load = 1; loadvalue = 12'b000000000001;
        #1   rst = 0; enabled = 1; load = 0; loadvalue = 12'b000000000001;
    end
    //Fin de la simulación
    initial
    #200 $finish;
    //Clock
    always
        #5 clk = ~clk;
    //Creación del diagrama de timing
    initial begin
        $dumpfile("E1_tb.vcd");
        $dumpvars(0,testbench);
    end
endmodule
```

Testbench del Ejercicio 1

El testbench fue diseñado para probar el enabled, el reset, el loadvalue, el load y su efecto en las salidas de program_byte, instr y oprnd.



```
---> WARNING: no PCF file found (.pcf)

iverilog -o E1_tb.out -D VCD_OUTPUT=E1_tb C:/Users/AlejandroRodr
vvp E1_tb.out
WARNING: E1.v:38: $readmemb(lista.list): Not enough words in the
rst enabled load loadvalue    | program_byte   instr oprnd
------------------------------|-------------------------
VCD info: dumpfile E1_tb.vcd opened for output.
0     0     0   000000000000| xxxxxxxx         xxxx     xxxx
1     0     0   000000000000| 00000001         0000     0000
0     1     0   000000000000| 00000001         0000     0000
0     1     0   000000000000| 00000010         0000     0001
0     1     0   000000000000| 00000100         0000     0010
0     0     0   000000000000| 00000100         0000     0010
0     1     0   000000001000| 00000100         0000     0010
0     1     0   000000001000| 00001000         0000     0100
0     1     0   000000001000| 00010000         0000     1000
0     1     0   000000001000| 00100000         0001     0000
0     1     1   000000001000| 10000001         0001     0000
0     1     1   000000001000| 10000001         1000     0001
0     1     0   000000001000| 10000001         1000     0001
0     1     0   000000001000| 01000010         1000     0001
0     1     0   000000001000| xxxxxxxx         0100     0010
1     0     0   000001100110| 00000001         0000     0000
0     0     0   000001100110| 00000001         0000     0000
0     1     1   000001100110| xxxxxxxx         0000     0000
0     1     0   000001100110| xxxxxxxx         xxxx     xxxx
0     1     1   000000000001| 00000010         xxxx     xxxx
0     1     0   000000000001| 00000010         xxxx     xxxx
0     1     0   000000000001| 00000100         0000     0010
0     1     0   000000000001| 00001000         0000     0100
0     1     0   000000000001| 00010000         0000     1000
0     1     0   000000000001| 00100000         0001     0000
0     1     0   000000000001| 01000000         0010     0000
gtkwave E1_tb.vcd E1_tb.gtkw
```

Output del programa en la terminal.

En esta tabla se pueden observar como los cambios en los inputs se reflejan en los outputs. Cabe resaltar que si no hay un valor guardado en la localidad de memoria que se está llamando en la ROM, las salidas de program_byte, instr y oprnd se vuelve x, ya que no hay un valor definido en esa localidad.

Diagrama de timing del E1.

En este diagrama se pueden observar cada una de las pruebas realizadas en el testbench. Se comprobó que efectivamente funcionaba el reset asíncrono. También se verificó que el loadvalue solo se cargara cuando load fuera igual a 1, así como que el counter contaba cuando el enabled estaba en 1. Se puede observar que las salidas se volvieron x en la instrucción y el operando cuando se llamó en la ROM a una localidad que no tenían ningún valor guardado.

Ejercicio 2

E2.v

lab10dig > E2 > E2.v

```
//Jose Alejandro Rodriguez Porras 19131
//Electrónica Digital 1
//Laboratorio 10
//Ejercicio 2 ALU
//Flip Flop de 4 bits
//Flip Flop tipo D de 4 bits con Enable (acumulador)
module accu(input clk, rst, enable, input [3:0]d, output reg [3:0]q);
    always @ (posedge clk or posedge rst) begin
        if (rst) begin
            q <= 4'b0000;
        end
        else if (enable) begin
            q <= d;
        end
    end
endmodule

//Buffer Triestado de 4 bits
module tribuf4(input enable, input [3:0]in, output wire [3:0]out);
    assign out = (enable) ? in:4'bz;

endmodule
//ALU de 4 bits
module ALU(input [3:0]A, input [3:0]B, input [2:0]opcode, output reg [3:0]resultado, output reg Carry, output reg Zero);

        reg [4:0]q; //Variable interna

        always @ (A or B or opcode) begin
            q = 5'b00000;
            // Depende de los casos del opcode se ejecuta una operación especifica
            case(opcode)
                3'b000: begin
                            // Dejar pasar A
                            q = A;
                            Carry = 1'b0;
                        end

                3'b001: begin
                            // Comp
                            q = A - B;
                            Carry = (q[4] == 5'b00000) ? 1:0;

                        end

                3'b010: begin
                            // Dejar pasar B
                            q = B;
                            Carry = 1'b0;
                        end

                3'b011: begin
                            // Suma
                            q = A + B;
                            Carry = (q[4] == 5'b00000) ? 1:0;
                        end

                3'b100: begin
                            // NAND
                            q = ~(A & B);
                            Carry = 1'b0;
                        end
            endcase
            Zero = (q == 5'b00000) ? 1:0;
            resultado = q[3:0];
        end
endmodule
```

Código del módulo de la ALU con buses y acumulador parte 1

En esta imagen se puede observar la primera parte del código del ejercicio 2, que consistió en implementar los módulos hechos en laboratorios pasados para luego implementarlos en un módulo unificador. Los submódulos usados fueron los buffers

triestado de 4 bits, un acumulador (flip flop tipo D de 4 bits) y 1 ALU pero modificada para orientarse al proyecto. La ALU tiene 5 posibles operaciones: Dejar pasar A, Comparar, Dejar pasar B, Sumar A y B, NAND entre A y B.

```
67  //Módulo completo con los demás submodulos, es decir, el buffer triestado de 4 bits, el accu y la ALU
68  module alu_driver_accu(input [3:0]in, input clk, rst, en_accu, en1, en2, input [2:0]opcode, output wire Carry, Zero, output wire [3:0]out);
69          wire [3:0]A;
70          wire [3:0]B;
71          wire [3:0]alu_out;
72          //Entrada
73          tribuf4 BT1(en1, in, B);
74          //Acumulador
75          accu DFF4(clk, rst, en_accu, alu_out, A);
76          //ALU
77          ALU alu(A, B, opcode, alu_out, Carry, Zero);
78          //Salida
79          tribuf4 BT2(en2, alu_out, out);
80  endmodule
```

Código del módulo de la ALU con buses y acumulador parte 2

En esta imagen se puede observar el módulo principal en el que se implementaron todos los demás submódulos.

```
E2_tb.v   E1_tb.v   E1.v lab09dig • E1   E3.v   E1.v lab10dig • E1   E4.v
lab10dig > E2 > E2_tb.v
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital 1
3   //Laboratorio 10
4   //Ejercicio 2 Testbench ALU
5   module testbench();
6       //Declaración de variables
7       reg en_accu, en1, en2, clk, rst;
8       reg [3:0]in;
9       reg [2:0]opcode;
10      wire Carry, Zero;
11      wire [3:0]out;
12      //Asignación de módulos
13      alu_driver_accu ADA(in, clk, rst, en_accu, en1, en2, opcode, Carry, Zero, out);
14      //Prueba de funcionamiento de la ALU con acumulador y buses
15      initial begin
16          $display("rst en_accu en1 en2 opcode in  | Carry Zero Out");
17          $display("-------------------------------|-------------------------");
18          $monitor("%b   %b       %b   %b   %b    %b| %b     %b    %b ", rst, en_accu, en1, en2, opcode, in, Carry, Zero, out);
19          rst = 0; en_accu = 0; en1 = 0; en2 = 0; opcode = 3'b000; in = 4'b0000; clk = 0;
20          //Prueba del reset
21          #1  rst = 1; en_accu = 0; en1 = 0; en2 = 0; opcode = 3'b000; in = 4'b0000;
22          //Prueba de que cuando todos los enables estan apagados la salida se pone en alta impedancia
23          #1  rst = 0; en_accu = 0; en1 = 0; en2 = 0; opcode = 3'b000; in = 4'b0111;
24          //Prueba de dejar pasar A y poniendo un input cualquiera
25          #10 rst = 0; en_accu = 1; en1 = 1; en2 = 1; opcode = 3'b000; in = 4'b0111;
26          //Prueba de dejar pasar B y poniendo un input cualquiera
27          #10 rst = 0; en_accu = 1; en1 = 1; en2 = 1; opcode = 3'b010; in = 4'b0001;
28          //Prueba de comparación entre A y B
29          #10 rst = 0; en_accu = 1; en1 = 1; en2 = 1; opcode = 3'b001; in = 4'b1001;
30          //Prueba de la suma entre A y B
31          #10 rst = 0; en_accu = 1; en1 = 1; en2 = 1; opcode = 3'b011; in = 4'b1011;
32          //Prueba de el NAND entre A y B
33          #10 rst = 0; en_accu = 1; en1 = 1; en2 = 1; opcode = 3'b100; in = 4'b1010;
34          //Prueba de apagar todos los enables
35          #10 rst = 0; en_accu = 0; en1 = 0; en2 = 0; opcode = 3'b100; in = 4'b1100;
36          //Prueba del reset asincrono
37          #10 rst = 1; en_accu = 1; en1 = 1; en2 = 1; opcode = 3'b000; in = 4'b1010;
38          #1  rst = 0; en_accu = 1; en1 = 1; en2 = 1; opcode = 3'b000; in = 4'b0000;
39          #200 $finish;
40      end
41      //Clock
42      always
43          #5 clk = ~clk;
44      //Creación del diagrama de timing
45      initial begin
46          $dumpfile("E2_tb.vcd");
47          $dumpvars(0,testbench);
48      end
49  endmodule
```

Código del Testbench del módulo de la ALU con buses y acumulador

En esta imagen se puede observar el módulo de testbench para probar la ALU con los buses y el acumulador. Se realizó una prueba de reset así como cada una de las operaciones que puede realizar la ALU. También se probaron los enables.

```
iverilog -o E2_tb.out -D VCD_OUTPUT=E2_tb C:/Users/AlejandroRodriguez/.apio
vvp E2_tb.out
rst en_accu en1 en2 opcode in  | Carry Zero Out
-------------------------------|------------------------
VCD info: dumpfile E2_tb.vcd opened for output.
0   0       0   0   000    0000| 0     x    zzzz
1   0       0   0   000    0000| 0     1    zzzz
0   0       0   0   000    0111| 0     1    zzzz
0   1       1   1   000    0111| 0     1    0000
0   1       1   1   010    0001| 0     0    0001
0   1       1   1   001    1001| 0     0    1000
0   1       1   1   001    1001| 0     0    1111
0   1       1   1   011    1011| 0     0    0011
0   1       1   1   011    1011| 1     0    1110
0   1       1   1   100    1010| 0     0    1101
0   1       1   1   100    1010| 0     0    0111
0   0       0   0   100    1100| 0     0    zzzz
1   1       1   1   000    1010| 0     1    0000
0   1       1   1   000    0000| 0     1    0000
gtkwave E2_tb.vcd E2_tb.gtkw
```

Output del programa del Ejercicio 2 en la terminal

En esta figura se puede observar la tabla de inputs vs outputs que sale cuando se ejecuta el programa. Se puede observar que cuando todos los enables están en 0 la salida Out del programa se pone en alta impedancia.

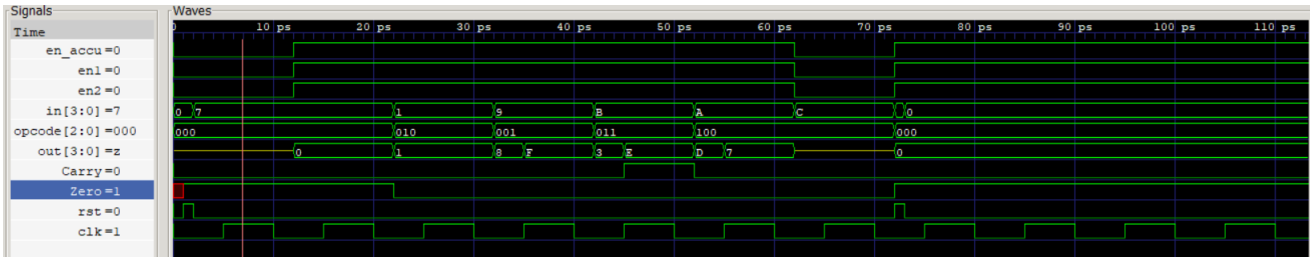


Diagrama de timing del ejercicio 2

En esta figura se puede observar el diagrama de timing en el que se puede visualizar cada una de las pruebas de una mejor manera. Se puede observar, por ejemplo, que cuando los enables están en 0, la salida se pone en alta impedancia z, en el diagrama representado por una línea amarilla. También se puede observar que las operaciones funcionan, como se puede observar en la primera operación que se ejecuta, es la 000, que quiere decir dejar pasar A, y en efecto se deja pasar A. También se puede observar que el Carry y el Zero funcionan bien ya que se activan cuando se da el caso en el que la salida es 000, para el Zero; mientras que el carry se activa cuando en la operación realizada existe overflow. También se puede observar que el reset funciona de manera asíncrona.