Nombre: Brayan José Castillo Alvarado
Carné: 19700
Sección: 10

Laboratorio 09

Ejercicio 01

El primer modulo es el Program Counter de 12 bit, con las entradas clock, reset, enabled, y load, con una unica salida. El segundo modulo es la memoria ROM de 12 bits, que leera el archivo ROMM.list. El tercer modulo es un Flip Flop tipo D ded 1 bit que se utilizo para el siguiente modulo para hacer uno de bits. El ultimo modulo es el de las conexiones donde se unen los modulos anteriores. Por ultimo, el testbench que utiliza las combinaciones de los 2 enableds y el load.

```
//Modulo del ProgramCounter
module ProgramCounter(input wire clock, reset, enable, load,
  input wire [11:0]entrada, output [11:0]salida);

//Definir el contador
  reg[11:0]contador;
//Condiciones del contador
  always @ (posedge clock or posedge reset or posedge load) begin
  if(reset)
  contador<=12'd0;
  else if(load)
  contador<=entrada;
  else begin
  //Si el contador esta lleno y enable sigue activado vuelva a 0
  if(contador == 12'b111111111111 & enable == 1)
  contador<=12'd0;
  else if(enable == 1)
  contador <= contador + 12'd1;
  end
  end
  assign salida = contador;
  endmodule

//Modulo de la memoria ROM
module ROM(input wire [11:0]address, output wire[7:0]val);

reg[7:0] M[0:4095];
initial begin
  $readmemh("ROMM.list",M);
  end
  assign val = M[address];
endmodule

// Flip Flop tipo D de 1 bit
module FFD1(input clk , input reset, E, input d,  output reg q);

  always@(posedge clk, posedge reset) begin
    if(E == 1) begin
    q <= d;
    end
    if(reset) begin
    q <= 0;
    end
  end
endmodule
```

Nombre: Brayan José Castillo Alvarado
Carné: 19700
Sección: 10

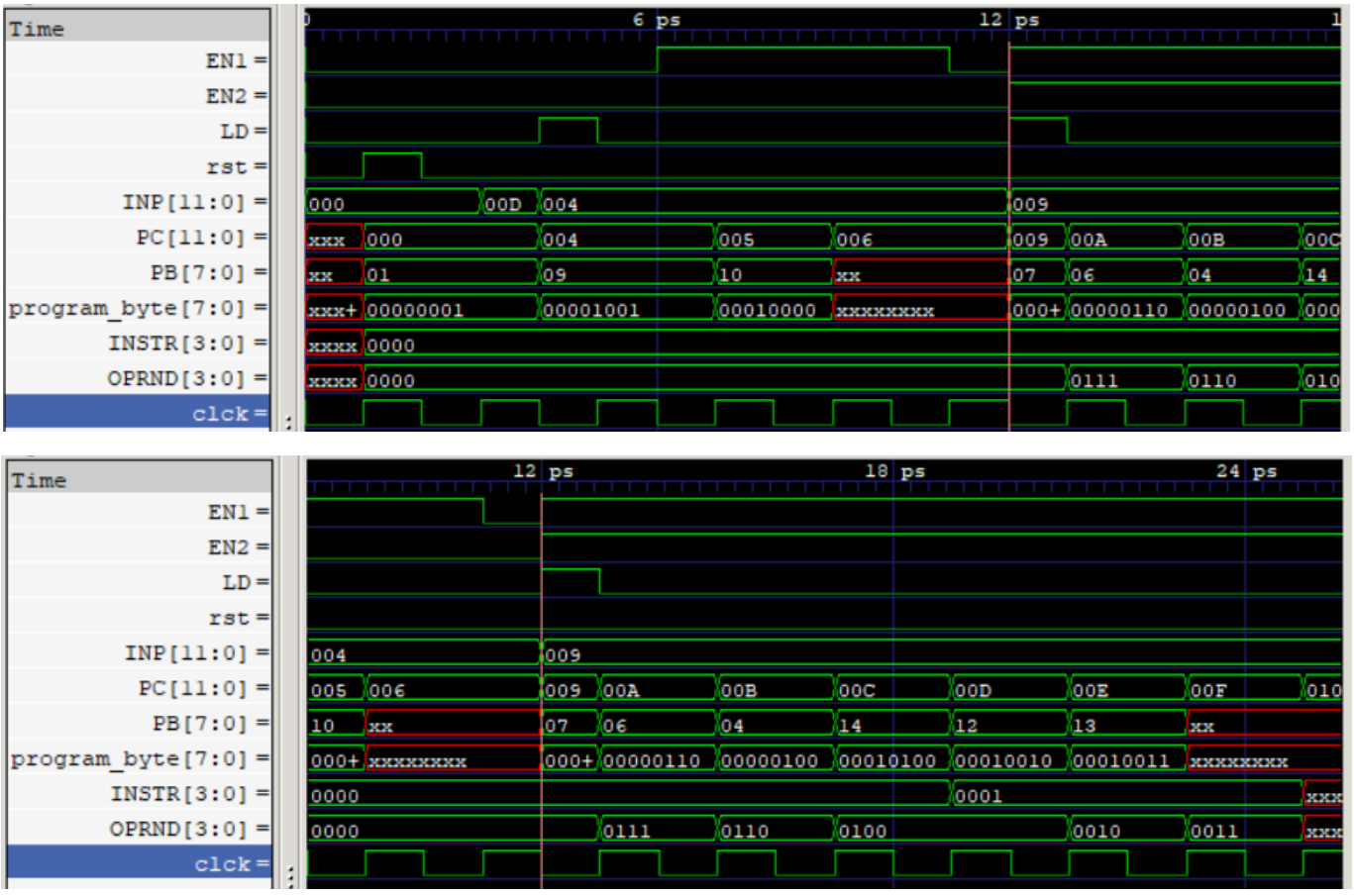
```
46
47  //Flip Flop tipo D de 8 bits
48  module FFD8(input clk , input reset, E, input [7:0]d, output wire [3:0]in, op);
49
50    FFD1 G1(clk, reset, E, d[0], op[0]);
51    FFD1 G2(clk, reset, E, d[1], op[1]);
52    FFD1 G3(clk, reset, E, d[2], op[2]);
53    FFD1 G4(clk, reset, E, d[3], op[3]);
54    FFD1 G5(clk, reset, E, d[4], in[0]);
55    FFD1 G6(clk, reset, E, d[5], in[1]);
56    FFD1 G7(clk, reset, E, d[6], in[2]);
57    FFD1 G8(clk, reset, E, d[7], in[3]);
58  endmodule
59
60  //Modulo de conexiones
61  module EJ01(input clk, reset, enable1, enable2, load, input [11:0]entrada, output [7:0]program_byte, output [3:0]instr, oprnd);
62
63    wire [11:0]PC;
64    wire [7:0]programb;
65
66    ProgramCounter U1(clk, reset, enable1, load, entrada, PC);
67    ROM U2(PC,programb);
68    FFD8 U3(clk, reset, enable2, programb, instr, oprnd);
69    assign program_byte = programb;
70  endmodule
```

```
1   module testbench();
2   reg clck, rst, EN1, EN2, LD;
3   reg [11:0]INP;
4   wire [7:0]PB;
5   wire [3:0]INSTR, OPRND;
6   EJ01 M1(clck, rst, EN1, EN2, LD, INP, PB, INSTR, OPRND);
7
8   initial begin
9   clck=0; rst=0; EN1=0; EN2=0; LD=0; INP=12'b0;
10  #1 rst=1;
11  #1 rst=0;
12  #1 INP=13;
13  #1 INP=4; LD=1;
14  #1 LD=0;
15  #1 EN1=1;
16  #5 EN1=0;
17  #1 INP=9; EN2=1; LD=1; EN1=1;
18  #1 LD=0;
19  end
20  always
21  #1 clck = ~clck;
22  initial
23  #40 $finish;
24
25  initial begin
26
27      $dumpfile("EJ01_tb.vcd");
28      $dumpvars(0,testbench);
29    end
30  endmodule
```

Nombre: Brayan José Castillo Alvarado
Carné: 19700
Sección: 10

Diagrama de Timing

En el diagrama podemos observar que en INP ingresamos los valores al ProgramCounter el cual tiene como salida el wire PC, donde el valor se cargara unicamente cuando el load sea 1 que es cuando carga 004, que la ROM ira a busca que en este caso en ese espacio esta el valor 09. Luego de eso desactivamos el load y activamos el enabled 1, que hace que las salidas de PC empiecen a contar, ahora estando en 005 el valor sera 10 que es lo que esta en nuestra imagen del archivo.list, y en el caso de 006 PC es x debido a que nosotros hicimos un salto en el archivo y por lo tanto esta vacio ese espacio. En el testbench ahora revisaremos el espacio 009 que es donde realizamos el salto y como podemos ver si tiene la salida 07, ademas de esto justo en ese momento activamos el enabled 2 para las salidas INSTR y OPRND, que obtendran los valores en cada flanco como se observa en la segunda imagen del diagrama de timing, en esta podemos ver que INSTR toma los primeros 4 digitos de la salida program_byte y OPRND los ultimos 4 digitos en cada flanco. En este caso volvimos a activar el enabled 1 para que volviera a contar y cuando llega a 00F, las salidas empiezan a quedar en x porque nuestra lista no tiene mas valores.



ROMM.list

```
1  01 05 08 02 09 10 @9
2  07 06 04 14 12 13
```

Nombre: Brayan José Castillo Alvarado
Carné: 19700
Sección: 10

Ejercicio 02

El primer modulo es un Buffer Tri-estado de 4 bits utilizado en laboratorios anteriores. El siguiente es un Flip Flop tipo D el cual se utilizo para generar un Flip Flop tipo D de 4 bits. El siguiente modulo es la ALU la cual realiza, la cual realiza las respectivas operaciones de dejar pasar A, dejar pasar B, Sumar, Comparar y un NAND. Por último, se tiene el modulo de conexiones donde se interconectaron 2 Buffer tri-estado (Bus Driver), el Flip Flop de 4 bits (Acumulador) y la ALU. En el testbench se realizaron diferentes combinaciones para probara cada una de las opciones y demostrar el funcionamiento del Carry y el Zero.

```
// Buffer Tri-estado de 4 bits
module BTE(input E, input [3:0]D,  output [3:0]Q);

    assign Q = E==1?D:4'bzzzz;

endmodule

//Flip Flop tipo D de 1 bit

module FFD1(input clk , input reset, E, input d,  output reg q);

  always@(posedge clk, posedge reset) begin
    if(E == 1) begin
    q <= d;
    end
    if(reset) begin
    q <= 0;
    end
  end
endmodule

//Flip Flop tipo D de 4 bits

module FFD4(input clk , input reset, E, input [3:0]d, output wire [3:0]q);

  FFD1 G1(clk, reset, E, d[0], q[0]);
  FFD1 G2(clk, reset, E, d[1], q[1]);
  FFD1 G3(clk, reset, E, d[2], q[2]);
  FFD1 G4(clk, reset, E, d[3], q[3]);
endmodule
```

```
//Modulo de la ALU

module ALU (input [2:0]F,input [3:0]A, B, output reg[3:0]salida, output reg carry, zero);
reg [4:0]CS,CR;
always @(F,A,B) begin
    case(F)
    3'b000: begin
    salida = A;
    carry = 0;
    end
    3'b001: begin
    CR = A-B;
    carry = CR[4];
    salida = A;
    end
    3'b010: begin
    salida = B;
    carry = 0;
    end
    3'b011: begin
    CS = A + B;
    carry = CS[4];
    salida = CS[3:0];
    end
    3'b100: begin
    salida = ~(A & B);
    carry = 0;
    end
    default: salida = A;
  endcase
  end
  always@(salida) begin
  if(salida == 0) begin
```

Nombre: Brayan José Castillo Alvarado
Carné: 19700
Sección: 10

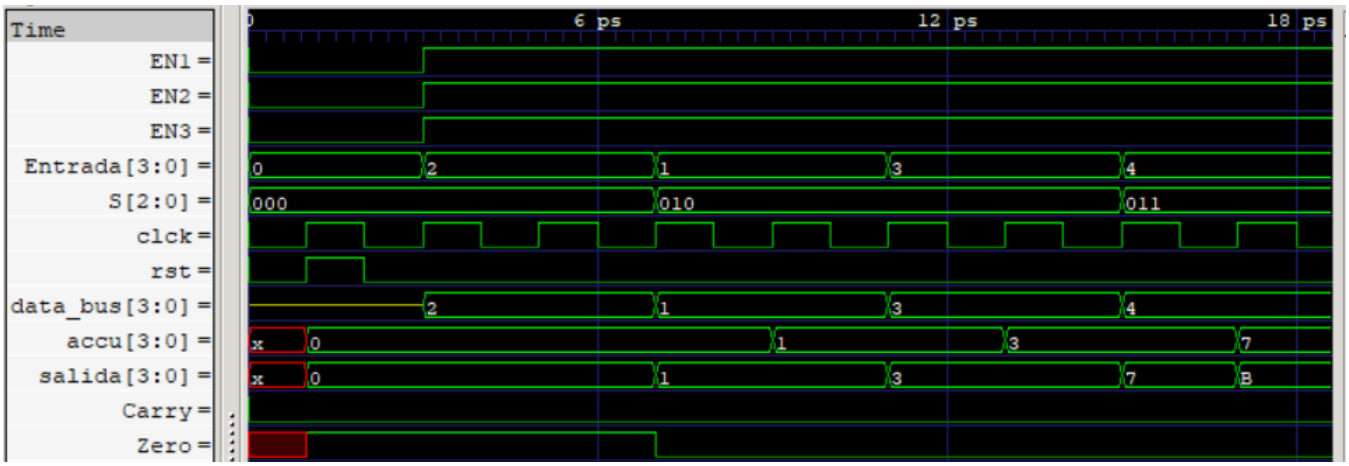
```
  always@(salida) begin
  if(salida == 0) begin
  zero = 1;
  end
  else
  zero = 0;
  end

endmodule

//Modulo de conexiones

module EJ02(input clk, reset, enable1, enable2, enable3, input [3:0]entrada, input [2:0]S, output Carry, Zero, output [3:0]salida);

  wire [3:0]data_bus, accu;
  wire [3:0]aluc;

  BTE U1(enable1, entrada,data_bus);
  FFD4 U2(clk, reset, enable2, aluc, accu);
  ALU U3(S, accu, data_bus, aluc, Carry, Zero);
  assign salida = aluc;
endmodule
```

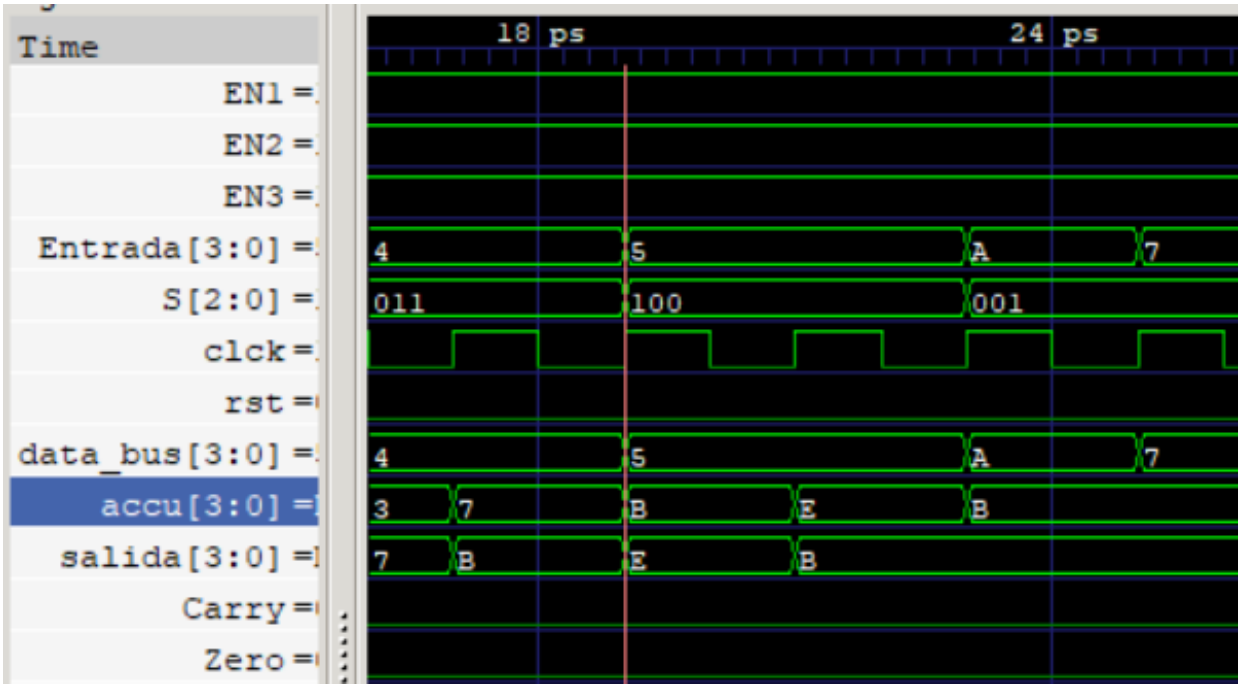
```
module testbench();
reg clck, rst, EN1, EN2, EN3;
reg [3:0]Entrada;
reg [2:0]S;
wire Carry, Zero;
wire [3:0]salida;
EJ02 M1(clck, rst, EN1, EN2, EN3, Entrada, S, Carry, Zero, salida);

initial begin
clck=0; rst=0; EN1=0; EN2=0; EN3=0; Entrada=0; S=0;
#1 rst=1;
#1 rst=0;
#1 Entrada= 4'b0010; EN1=1; EN2=1; EN3=1;S=3'b000;
#4 Entrada= 4'b0001; S=3'b010;
#4 Entrada= 4'b0011;
#4 Entrada= 4'b0100; S=3'b011;
#4 Entrada= 4'b0101; S=3'b100;
#4 Entrada= 4'b0111; S=3'b011;
#4 Entrada= 4'b1010; S=3'b001;
#2 Entrada= 4'b0111;

end
always
#1 clck = ~clck;
initial
#40 $finish;

initial begin

    $dumpfile("EJ02_tb.vcd");
    $dumpvars(0,testbench);
  end
endmodule
```

Nombre: Brayan José Castillo Alvarado
Carné: 19700
Sección: 10

Diagrama de Timing

En el diagrama de timing podemos observar que en el valor de A es cero y realizamos la instrucción de dejar pasar A (Instrucción 000) por lo que la salida es cero haciendo que la salida Zero se vuelva 1. Luego cuando cambiamos la instrucción 010, que es la que nos permite dejar pasar B que en este caso es el data_bus podemos ver que la salida es la misma en el flanco de reloj. En la siguiente instrucción 011 se realizan las sumas en este caso teníamos los valores 4 y 3 que dieron en la salida el valor 7, se realizaron algunas sumas más como se observa en la segunda imagen donde luego de 7 se sumó 3 y dio como resultado 11, luego se realizó la instrucción del NAND, donde los valores de la salida estaban dados conforme a si ambas entradas eran 1 el valor seria 0 y en cualquier otro caso 1, como en la imagen 3. Luego se realizo la instrucción 001 que es la comparación que únicamente resta A y B pero copia el valor de A en la salida como se observa en la ultima imagen. Luego se volvió a ejecutar la instrucción de suma 011 para mostrar el funcionamiento del Carry donde se realizaron sumas en las cuales hubo un overflow como se ve en la última imagen.





Nombre: Brayan José Castillo Alvarado
Carné: 19700
Sección: 10