Valerie Valdez
Universidad del Valle de Guatemala

Carné: 19659
Sección: 20

Electrónica digital
Laboratorio No. 10

Ejercicio No. 1

Código en Verilog

```
// Flip Flop 4 bits tipo D de 4 bits con reset síncrono y asíncrono
module FFD4(input wire clk, reset, EN,
  input wire [3:0]C,
  output reg [3:0]D);

// Con always una sección del código estará siempre activado
  always @ (posedge clk, posedge reset) begin // Count on the rising edge of the clk

      if(reset) begin // En caso de reset colocar todas las salidas en 0
        D <= 4'b0;
      end

      else if (EN) begin // Entrada igual a la salida
        D <= C;
      end
  end
endmodule


// Flip Flop tipo D de 8 bits
module FETCH(input wire clk, reset, EN, input wire [7:0]D, output wire[3:0]instr, oprnd);

  FFD4 A1(clk, reset, EN, D[7:4], instr); // bits más significativos
  FFD4 B1(clk, reset, EN, D[3:0], oprnd); // bits menos significativos
endmodule



// Implementar un contador de 12 bits con un enable y capaz de hacer un load
module contador(input wire clk, rst, EN, blo,
  input wire [11:0]load,
  output reg [11:0]val);
```

```
// Non-blocking assigment
    always @(posedge clk or posedge rst or posedge blo) begin

      if(rst == 1) // Colocar todos los valores en 0
        val <= 12'b0;

      else if(EN == 1) //Enabled encendido cuenta en cada flanco de reloj
        val <= val + 1;

      else if(blo == 1) // Blo encendido entonces Load comienza a contar
        val <= load;

        end
endmodule


// Implementación de la memoria ROM de 4Kx8
module Mrom(input wire [11:0]address,
  output wire [7:0]D); // Tamaño del dato de 8 bits

  reg[7:0] Mrom[0:4095]; // Ancho y localidades de memoria


// Inicializar la memoria
  initial begin
    $readmemb("Mrom.list", Mrom); // Leer los valores de la lista en binario
  end

  assign D = Mrom[address]; // Asignarle el valor en la dirección dada
endmodule
```

```
/ Integrando todo
odule integrador(input wire clk, reset, ENPC, ENF, blo, input wire [11:0]load, output wire [7:0]P_B, output wire [3:0]instr, output wire [3:0]oprnd);
 wire [11:0]PC;
 contador C1(clk, reset, ENPC, blo, load, PC);
 Mrom C2(PC, P_B);
 FETCH C3(clk, reset, ENF, P_B, instr, oprnd);

ndmodule
```

Testbench

```
// Testbench del circuito mostrado en el diagrama

module testbench();
  reg clk, reset, ENPC, ENF, blo;
  reg [11:0]load;
  wire [7:0]P_B;
  wire[3:0]instr;
  wire [3:0]oprnd;


  // Reloj de una unidad de tiempo con cambio en el flanco
  always
      #1 clk = ~clk;

  // Iteración del módulo
  integrador U(clk, reset, ENPC, ENF, blo, load, P_B, instr, oprnd);


  // Prueba para el Flip Flop tipo D de 8 bits
  initial begin
    #1
    $display("\n");
    $display(" Programa ");
    $display("\n");
    $display("-----------------------------|--------------------------|");
    $display("Clk reset ENPC ENF blo load  |   P_B     instr    oprnd |");
    $display("-----------------------------|--------------------------|");
    $monitor("%b  %b  %b  %b  %b  %b  | %b   %b    %b  |", clk, reset, ENPC, ENF, blo, load, P_B, instr, oprnd);

    clk = 0; reset = 1; ENF = 0; ENPC = 0; blo = 0; load = 12'b000000000000;
    #2 reset = 0; ENF = 1; ENPC = 1; blo = 0;
    #2 reset = 0; ENF = 1; ENPC = 1; blo = 0;
    #2 reset = 0; load = 12'b000000000100;
    #2 reset = 0; ENF = 1; ENPC = 1; blo = 1;
    #3 reset = 0; ENF = 0; ENPC = 0; blo = 0;

  end
```

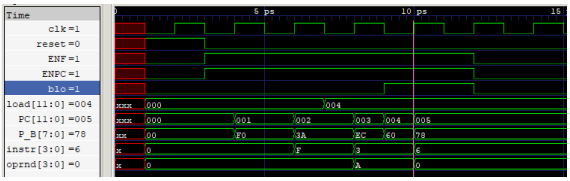
```
  initial
    #35 $finish;

// GTK wave
  initial begin
    $dumpfile("Lab10_tb.vcd");
    $dumpvars(0, testbench);
  end
endmodule
```

Prueba realizada

```
-----------------------------|--------------------------|
Clk reset ENPC ENF blo load  |   P_B     instr    oprnd |
-----------------------------|--------------------------|
0  1  0  0  0  000000000000  | 00000000  0000    0000   |
1  1  0  0  0  000000000000  | 00000000  0000    0000   |
0  0  1  1  0  000000000000  | 00000000  0000    0000   |
1  0  1  1  0  000000000000  | 11110000  0000    0000   |
0  0  1  1  0  000000000000  | 11110000  0000    0000   |
1  0  1  1  0  000000000000  | 00111010  1111    0000   |
0  0  1  1  0  000000000100  | 00111010  1111    0000   |
1  0  1  1  0  000000000100  | 11101100  0011    1010   |
0  0  1  1  1  000000000100  | 01100000  0011    1010   |
1  0  1  1  1  000000000100  | 01111000  0110    0000   |
0  0  1  1  1  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
1  0  0  0  0  000000000100  | 01111000  0110    0000   |
0  0  0  0  0  000000000100  | 01111000  0110    0000   |
gtkwave Lab10_tb.vcd Lab10_tb.gtkw
```

Diagrama de Timing



Para realizar implementar el program counter, la program rom y el fetch en un solo módulo se crearon los módulos de cada uno. El FETCH es un FF de 8 bits con dos salidas las cuales se dividen en instrucciones y operaciones. El program counter es un contador de 12 bits con un enable y capaz de hacer un load por lo que, a este, se le incluyó un bit para controlar el load. Por último, el program rom esta conformado por una memoria de 4Kx8 y se utilizó el comando $readmemb para leer información de un documento externo.

Para el testbench se comprobó que el load funcionara apropiadamente, así como ambos enables permitieran pasar el valor establecido y para que el load comience a contar.

Ejercicio No. 2

```
// Módulo acumulador: Flip Flop tipo D de 4 bits
module ACCUMULATOR(input wire clk, reset, EN,
  input wire [3:0]ALU,
  output reg [3:0]ACCU);

// Con always una sección del código estará siempre activado
  always @ (posedge clk, posedge reset) begin // Count on the rising edge of the clk

      if(reset) begin // En caso de reset colocar todas las salidas en 0
        ACCU <= 4'b0;
      end

      else if (EN) begin // Entrada igual a la salida
        ACCU <= ALU;
      end
  end
endmodule



// Módulo Buffer triestado
module BT(input wire EN,
  input wire [3:0]I,
  output wire [3:0]Y); // Salida de 4 bits del buffer

  assign Y = (EN) ? I : 4'bz; // Cuando EN = 0 la salida está en alta impedancia
endmodule
```

```
// Módulo ALU
module ALUU(input wire [3:0]ACCU,
  input wire [3:0]BUFFER,
  input wire [2:0]S,
  output wire [3:0]SAL,
  output wire C, ZERO);

  reg [4:0]OF; // el 5to bit es por si hay un overflow

  always @ (ACCU, BUFFER, S) begin
    case(S)
      3'b000: OF <= ACCU;
      3'b001: OF <= ACCU - BUFFER;
      3'b010: OF <= BUFFER;
      3'b011: OF <= ACCU + BUFFER;
      3'b100: OF <= (ACCU ~| BUFFER);
      default: OF <= 5'b00000; // Si no es alguna operación la salida es igual a 0
    endcase
  end
  assign SAL = OF[3:0];
  assign C = OF[4];
  assign ZERO = (OF == 5'B00000) ? 1:0;
endmodule



// Módulo para juntar la ALU, los buffers y el acumulador
module operacional(input wire clk, reset, ENA, ENB, ENC,
  input wire [2:0]S,
  input wire [3:0]OPRND,
  output wire [3:0]SAL,
  output wire [3:0]DATA,
  output wire [3:0]ALU_SAL,
  output wire [3:0]ACCUMULATOR,
  output wire C, ZERO); //Enable de los buffers y del acumulador
```

```
// Módulo para juntar la ALU, los buffers y el acumulador
module operacional(input wire clk, reset, ENA, ENB, ENC,
  input wire [2:0]S,
  input wire [3:0]OPRND,
  output wire [3:0]SAL,
  output wire [3:0]DATA,
  output wire [3:0]ALU_SAL,
  output wire [3:0]ACCUMULATOR,
  output wire C, ZERO); //Enable de los buffers y del acumulador

  wire [3:0]BUFFER;
  wire [3:0]ACCU;
  wire [3:0]ALU;

  // Instanciación de los módulos
  BT R2(ENB, OPRND, BUFFER);
  BT R3(ENC,ALU, SAL);
  ACCUMULATOR R4(clk, reset, ENA, ALU, ACCU);
  ALUU R1(ACCU, BUFFER, S, ALU, C, ZERO);

  assign DATA = BUFFER;
  assign ALU_SAL = ALU;
  assign ACCUMULATOR = ACCU;

endmodule
```

Testbench

```
// Testbench
module testbench();
  reg clk, reset, ENA, ENB, ENC;
  reg [2:0]S;
  reg [3:0]OPRND;
  wire [3:0]DATA;
  wire [3:0]SAL;
  wire [3:0]ALU_SAL;
  wire [3:0]ACCU;
  wire C, ZERO;


  // Reloj de una unidad de tiempo con cambio en el flanco
  always
    #1 clk = ~clk;

// Instanciación del módulo
operacional A1(clk, reset, ENA, ENB, ENC, S, OPRND, SAL, DATA, ALU_SAL, ACCU, C, ZERO);


initial begin
  #1
  $display("\n");
  $display(" Ejercicio No. 2 ");
  $display("\n");
  $display("-----------------------------------|-------------------------------|");
  $display("Clk reset ENA ENB ENC  S   OPRND  | SAL   DATA  SALIDA  ACCU  C  ZERO |");
  $display("-----------------------------------|-------------------------------|");
  $monitor(" %b     %b   %b   %b   %b  %b   %b  | %b   %b    %b    %b   %b   %b  |", clk, reset, ENA, ENB, ENC, S, OPRND, SAL, DATA, SAL, ACCU, C, ZERO);
```

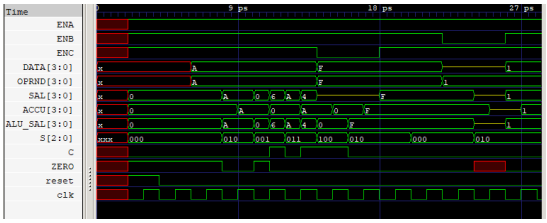
```
#1 clk = 0; reset = 1; ENA = 1; ENB = 1; ENC = 1; S = 3'b000;
#2 reset = 0;
#2 OPRND = 4'b1010;
#2 S = 3'b010;
#2 S = 3'b001;
#2 S = 3'b011;
#2 S = 3'b100; ENC = 0; OPRND = 4'B1111;
#2 S = 3'b010;
#2 ENC = 1;
#2 S = 3'b000;
#2 ENB = 0; OPRND = 4'b0001;
#2 S = 3'b010;
#2 ENB = 1;

end

initial
  #30 $finish;

// GTK wave
initial begin
  $dumpfile("Ejercicio_2_tb.vcd");
  $dumpvars(0, testbench);
end
endmodule
```

Diagrama de Timing




```
Ejercicio No. 2

--------------------------------|-----------------------------------|
Clk reset ENA ENB ENC  S   OPRND | SAL   DATA  SALIDA  ACCU  C  ZERO |
--------------------------------|-----------------------------------|
x     x    x   x   x  xxx  xxxx | xxxx  xxxx  xxxx    xxxx  x  x    |
0     1    1   1   1  000  xxxx | 0000  xxxx  0000    0000  0  1    |
1     1    1   1   1  000  xxxx | 0000  xxxx  0000    0000  0  1    |
0     0    1   1   1  000  xxxx | 0000  xxxx  0000    0000  0  1    |
1     0    1   1   1  000  xxxx | 0000  xxxx  0000    0000  0  1    |
0     0    1   1   1  000  1010 | 0000  1010  0000    0000  0  1    |
1     0    1   1   1  000  1010 | 0000  1010  0000    0000  0  1    |
0     0    1   1   1  010  1010 | 1010  1010  1010    0000  0  0    |
1     0    1   1   1  010  1010 | 1010  1010  1010    1010  0  0    |
0     0    1   1   1  001  1010 | 0000  1010  0000    1010  0  1    |
1     0    1   1   1  001  1010 | 0110  1010  0110    0000  1  0    |
0     0    1   1   1  011  1010 | 1010  1010  1010    0000  0  0    |
1     0    1   1   1  011  1010 | 0100  1010  0100    1010  1  0    |
0     0    1   1   0  100  1111 | zzzz  1111  zzzz    1010  1  0    |
1     0    1   1   0  100  1111 | zzzz  1111  zzzz    0000  1  0    |
0     0    1   1   0  010  1111 | zzzz  1111  zzzz    0000  0  0    |
1     0    1   1   0  010  1111 | zzzz  1111  zzzz    1111  0  0    |
0     0    1   1   1  010  1111 | 1111  1111  1111    1111  0  0    |
1     0    1   1   1  010  1111 | 1111  1111  1111    1111  0  0    |
0     0    1   1   1  000  1111 | 1111  1111  1111    1111  0  0    |
1     0    1   1   1  000  1111 | 1111  1111  1111    1111  0  0    |
0     0    1   0   1  000  0001 | 1111  zzzz  1111    1111  0  0    |
1     0    1   0   1  000  0001 | 1111  zzzz  1111    1111  0  0    |
0     0    1   0   1  010  0001 | zzzz  zzzz  zzzz    1111  0  x    |
1     0    1   0   1  010  0001 | zzzz  zzzz  zzzz    zzzz  0  x    |
0     0    1   1   1  010  0001 | 0001  0001  0001    zzzz  0  0    |
1     0    1   1   1  010  0001 | 0001  0001  0001    0001  0  0    |
0     0    1   1   1  010  0001 | 0001  0001  0001    0001  0  0    |
1     0    1   1   1  010  0001 | 0001  0001  0001    0001  0  0    |
0     0    1   1   1  010  0001 | 0001  0001  0001    0001  0  0    |
gtkwave Ejercicio_2_tb.vcd Ejercicio_2_tb.gtkw
```

Para implementar el acumulador, la ALU, y ambos bus drivers se utilizó un FF de 4 bits para crear el acumulador. Asimismo, para los bus drivers se utilizó un buffer tri estado que fue instanciado para obtener dos. Su comportamiento se basa en que al estar Enable en 0 la salida estará en altaimpedancia. Por otro lado, para la Alu cabe mencionar que se utilizó un bus de 5 bits para que, al momento de haber un overflow, en la bandera del carry se evidencie este comportamiento.

Para finalizar, en el módulo del testbench se probaron diferentes operaciones para comprobar que el funcionamiento era el correcto.

Link al repositorio:
https://github.com/valeelorraine/Laboratorios_Digital