Fernando Arribas

Carné: 19030

Lab 08 Digital 1

Ejercicio 1

```
1   //Ejercicio 1 Laboratorio 8 Electronica Digital 1
2
3   module counter #(parameter N = 12) //Definimos vaeiables
4                   (input wire clk, reset, en, load,
5                    input wire [N-1:0] val,
6                    output reg [N-1:0] q);
7
8     always @ (posedge clk or posedge load or posedge reset) begin
9       if (reset == 1) //flanco de reloj se colocara la salida en 0
10        q <= 12'b0;
11
12      else if (load == 1)//flando de reloj se cargara un valor al
13        q <= val;        //contador
14
15      else if (en == 1)//cuando enable sea igual a 1
16        q <= q + 1;   //comenzara a contar
17
18    end
19
20
21  endmodule
```

Primero definimos las variables a utilizar las cuales son un clock, un reset, un enable (en), un load, un variable de 12 bits llamada val, y la salida de 12 bits; adicionalmente definimos un parámetro, el cual nos indicará de cuantos bits será nuestro contador, en este caso 12 bits. Luego todo eso se ingresa a un always, en donde el reloj, el reset y load trabajaran con flancos de reloj positivos. Dentro existen un total de 3 condicionales, el primero indica si hay un cambio de reloj para reset la salida se colocara en 0. Luego si existe un flanco de reloj en load la salida pasara a ser los que tengamos en la variable val. Por último, si enable es igual a 1 la salida del contador será el valor pasado del mismo más uno. Hay que mencionar que todas estas operaciones están realizadas con non-blocking assignment, para que el contador pueda saltar entre las distintas opciones sin ninguna restricción.

```
module testbench();

  reg clk, reset, en, load;
  reg [11:0] val;
  wire [11:0] q;

  counter e1(clk, reset, en, load, val, q);

  always
   #1 clk = ~clk;

  initial begin

  clk = 0;

  $display("Ejercicio 1: Contador de 12 bits");
  $display("---Cuenta--- | ---Carga----");
  $monitor("%b | %b", q, val);

  reset = 0; en = 0; load = 0; val = 12'b0;
  #10 reset = 1;
  #2 reset = 0; val = 12'b000001100100;
  #2 load = 1;
  #20 load = 0; en = 1;


  #60 $finish;
  end

  initial begin
    $dumpfile("Contador_tb.vcd");
    $dumpvars(0, testbench);
  end

endmodule
```

Testbench contador

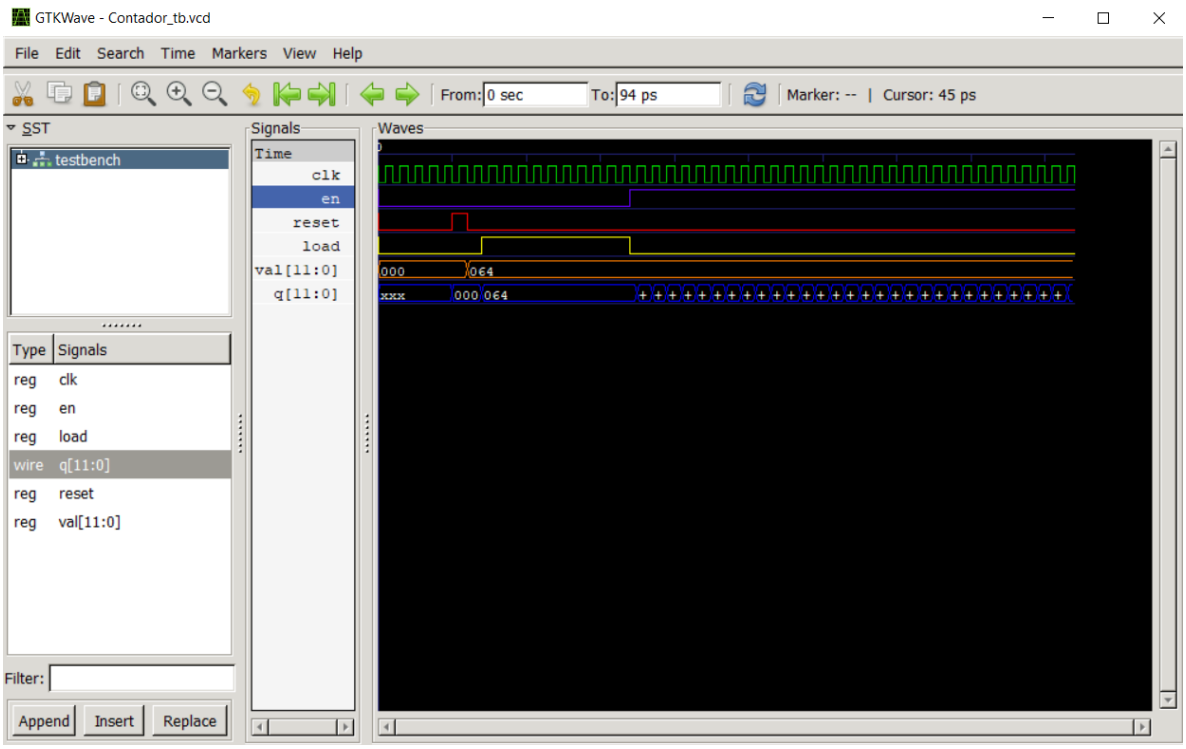

Diagrama de Timing Contador

Ejercicio 2

```
/Ejercicio 2 Laboratortio 8 Electronica Digital 1

odule memoria(input wire [11:0] l, output wire [7:0] y);
    //Definimos las variables de entrada y salida.

    reg [7:0] m[0:4095]; //Asignamos el tamaño de la memoria
                         //la cual es de 4k con 8 bits de ancho

    initial begin
        $readmemb("memoria.ejercicio_2", m); //guardamos un valor
            //binario en la memoria
    end

    assign y = m[l];

ndmodule
```

Primero declaramos las variables de entrada, siendo una entrada L de 12 bits, la cual nos servirá para buscar en las 4096 localidades de nuestra memoria y una salida Y que nos desplegara lo que esta guardado en cierta localidad de nuestra memoria. Lo primero a realizar fue un array de memoria, el cual será de 4k localidades y 8 bits de ancho, la cual será una variable M. Luego por medio de la instrucción $readmemb leeremos el archivo externo que guardaremos en nuestra memoria, la cual será guardada en el arreglo m. Por ultimo asignamos que el valor de la salida será la localidad que seleccionemos con L.

```
module testbench();

    reg [11:0] l;
    wire [7:0] y;

    memoria a(l, y);

    initial begin
        $display("Ejercicio 2: Memoria ROM");
        $display("-Localidad- |----Dato----");
        $monitor("%b | %b", l, y);

        #1 l = 12'b000000000000;
        #1 l = 12'b000000000001;
        #1 l = 12'b000000000010;
        #1 l = 12'b000000000011;
        #1 l = 12'b000000000100;
        #1 l = 12'b000000000101;
        #1 l = 12'b000000000110;
        #1 l = 12'b000000000111;
        #1 l = 12'b000000001000;
        #1 l = 12'b000000001001;

    end

    initial
    #100 $finish;

    initial begin
        $dumpfile("memory_tb.vcd");
        $dumpvars(0, testbench);
    end


endmodule
```

Testbench Memoria

```
1    0000_0000
2    0001_0001
3    0011_0000
4    1111_1111
5    1010_0101
6    1100_0011
7    0010_0000
8    1110_1110
9    1001_1111
10   1010_0000
```

Datos Guardados en la Memoria

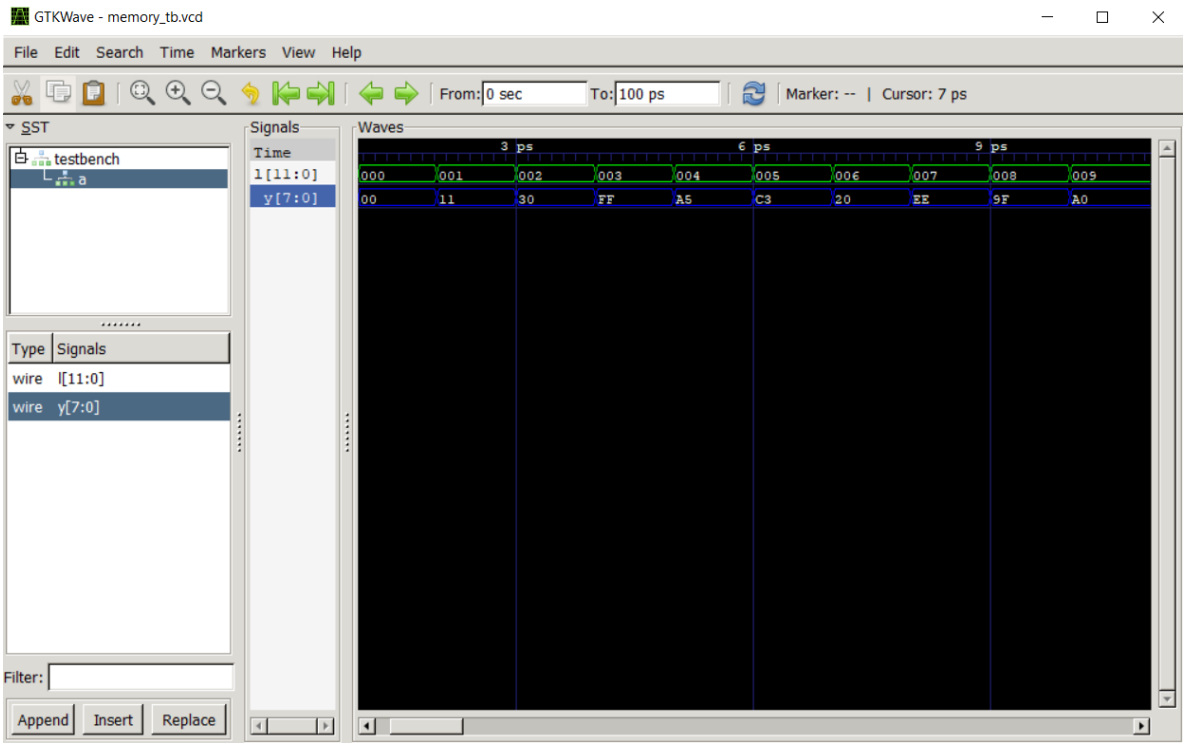

Diagrama de Timing Memoria

Array: se declara un reg, el cual poseerá cierta cantidad de bits, luego dentro del reg se colocará una variable a la cual en su derecha se le colocará otra cierta cantidad de variables.
$reg[7:0]\ m[0:4095]$ la cual será un array de 4096X8, o 4096 localidades con un ancho de ocho bits.

$readmemb se utiliza para leer entradas en binario, mientras que $readmemh se utiliza para leer entradas en hexadecimal.

Ejercicio 3

```
1   module ALU(input wire [3:0] a, b,
2              input wire [2:0] f,
3              output reg [3:0] y);
4
5       always @ (a or b or f) begin
6
7           case(f)
8               3'b000: y = (a & b); //AND
9               3'b001: y = (a | b); //OR
10              3'b010: y = (a + b); //Suma
11              3'b100: y = (a & ~b); //AND, con NOT B
12              3'b101: y = (a | ~b); //OR, con NOT B
13              3'b110: y = (a - b); //Resta
14              3'b111: y = (a < b) ? 1:0; //Comparacion entre A y B
15              default: y = 3'b0; //Valor por defecto por si la
16                                 //opción no se encuentra en la ALU
17          endcase
18
19      end
20
21
22  endmodule
```

Lo primero es definir las variables a utilizar, las cuales son dos entradas de 4 bits llamadas A y B, el controlador de 3 bits F y la salida de 4 bits Y, Para describir el funcionamiento de la ALU se utilizo casos basados en F, en donde cada combinación de F dará paso a una acción, exceptuando la combinación 011 la cual no realizará nada. La primera acción (000) será un AND, luego (001) un OR, luego (010) una Suma, la combinación 011 no hace nada, la 100 es un AND con B negada, 101 es un OR con B negada, 110 es una resta y 111 es una comparación entre A y B para ver cuál es mayor. Por último, se colocará una salida ‘por defecto’ la cual dará paso si la instrucción ingresada no existe.

```
1   module testbench();
2
3       reg [3:0] a, b;
4       reg [2:0] f;
5       wire [3:0] y;
6
7       ALU m(a, b, f, y);
8
9
10      initial begin
11      $display("Ejerccio 3: ALU");
12      $display("A | B | F | Salida");
13      $monitor("%b | %b | %b | %b", a, b, f, y);
14
15          a = 4'b0000; b = 4'b0000; f = 3'b000;
16      #1 a = 4'b0001; b = 4'b0000; f = 3'b000;
17      #1 a = 4'b0001; b = 4'b0001; f = 3'b000;
18      #1 a = 4'b0000; b = 4'b0000; f = 3'b001;
19      #1 a = 4'b0000; b = 4'b0001; f = 3'b001;
20      #1 a = 4'b0001; b = 4'b0001; f = 3'b001;
21      #1 a = 4'b1000; b = 4'b0100; f = 3'b010;
22      #1 a = 4'b0010; b = 4'b0000; f = 3'b100;
23      #1 a = 4'b0000; b = 4'b0000; f = 3'b101;
24      #1 a = 4'b0110; b = 4'b0001; f = 3'b110;
25      #1 a = 4'b0100; b = 4'b0000; f = 3'b111;
26      #1 a = 4'b0000; b = 4'b0100; f = 3'b111;
27
28      end
29
30      initial
31          #30 $finish;
32
33      initial begin
34          $dumpfile("ALU_tb.vcd");
35          $dumpvars(0, testbench);
36      end
37
38
39  endmodule
```

Testbench ALU

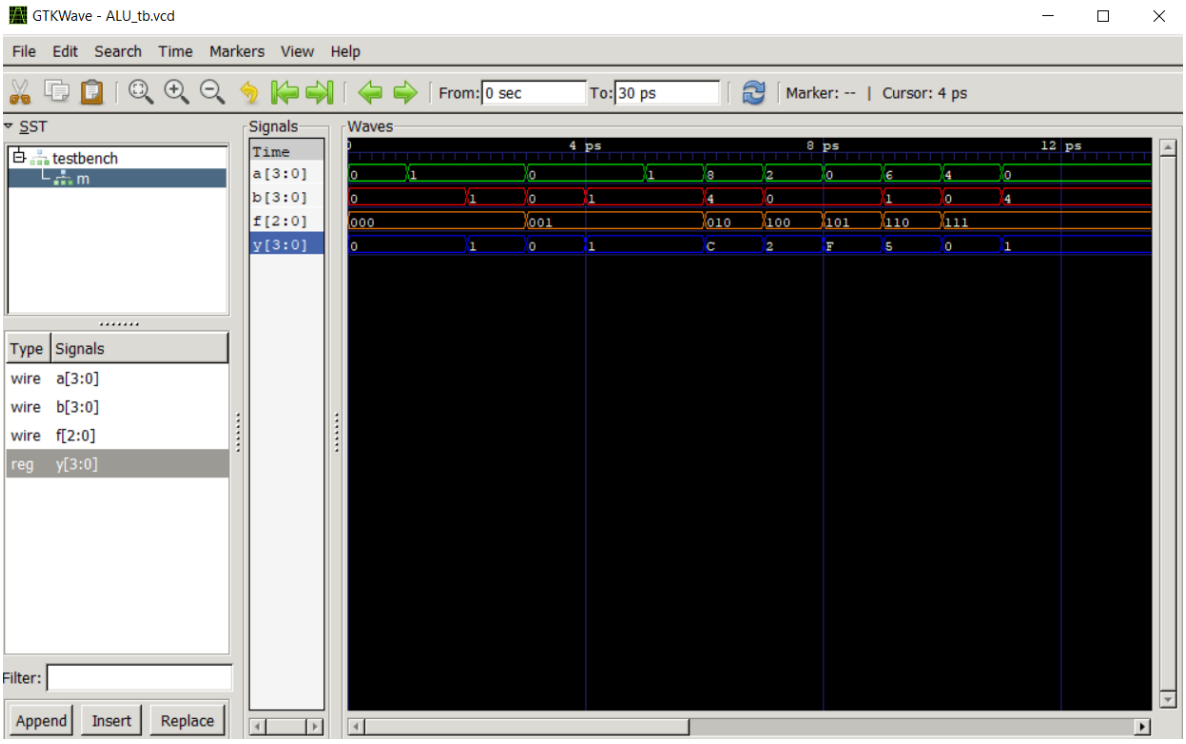

Diagrama de Timing ALU

Link Repositorio de Github: https://github.com/fernando19030/Laboratorios-Electronica_Digital_1-19030.git