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

Laboratorio 08

Ejercicio 01

Codigo del modulo del contrador, donde se ingresan las entradas clock, reset, enable, load, entrada y la salida, estas dos ultimas de 12 bits. Agregamos el reg de contador y usamos el always para que nuestro contador este sincronizado con clock, reset y load. Si reset es 1 el contador sera igual a 0, si load esta en 1 el contador sera igual a la entrada de 12 bits, si enable esta en 1 el contador empezara a sumar 1, en caso llegue al maximo numero de bits este se reiniciara a 0. Al final asignar a la salida el valor de contador.

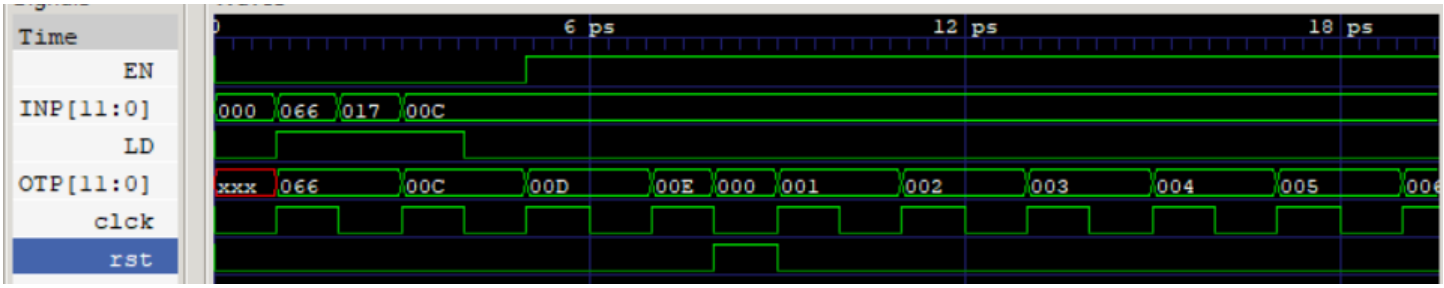
```
//Creación del modulo
module EJ1Contador(input wire clock, reset, enable, load,
  input wire [11:0]entrada, output [11:0]salida);
//Definir el contador
  reg[11:0]contador;
//Condiciones del contador
  always @ (posedge clock or posedge reset or posedge load) begin
  if(reset)
  contador<=12'd0;
  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
```

```
module testbench();

reg clck, rst, EN, LD;
reg [11:0]INP;
wire [11:0]OTP;
EJ1Contador U1(clck, rst, EN, LD, INP, OTP);

initial begin
clck=0; rst=0; EN=0; LD=0; INP=0;
#1 INP=102; LD=1;
#1 INP=23;
#1 INP=12;
#1 LD=0;
#1 EN = 1;
#3 rst=1;
#1 rst=0;
end
always
#1 clck = ~clck;
initial
#40 $finish;

initial begin

    $dumpfile("EJ1_tb.vcd");
    $dumpvars(0,testbench);
  end
endmodule
```

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

Diagrama de Timing

En el código del testbench podemos ver que de primero dejamos load en 1 por lo que en la salida OTP se encuentran los valores de la entrada INP, luego desactivamos el load y activamos en enable por lo que el contador empieza a sumar 1 con el ultimo valor que obtuvo del load. Luego se realiza un pulso del reset, por lo que la salida se vuelve 0 y como el enable sigue activado seguimos aumentando de 1 en 1 con cada flanco de reloj.



Ejercicio 02

Creamos el modulo de la ROM el cual tendrá como entrada address de 12 bits y como salida val de 8 bits, declaramos reg para tener las 4096 localidades. Utilizamos readmemh para que lea los valores en hexadecimal y que llame a la lista de numero que creamos en nuestro archivo. Por último, que a la salida val se le asigne lo que se encuentra en la memoria.

```
1  module ROM(input wire [11:0]address, output wire[7:0]val);
2  reg[7:0] M[0:4095];
3  initial begin
4      $readmemh("ROMM.list",M);
5      end
6      assign val = M[address];
7      endmodule
```

Archivo.list

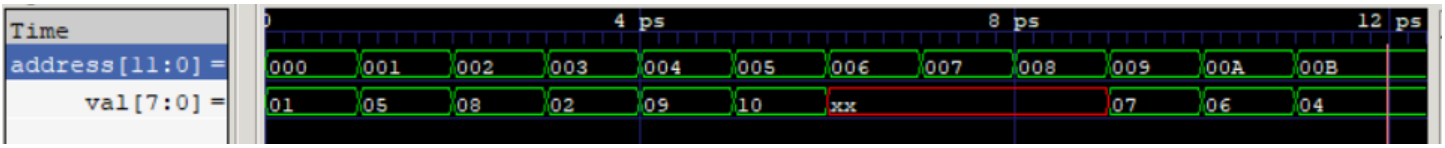
```
01
05
08
02
09
@62
07
06
04
14
```

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

```
module testbench();

reg [11:0]address;
wire [7:0]val;

ROM U1(address, val);

initial begin
$display("Memoria ROM");
$display("    Address     |  Data");
$display("------------|---------");
$monitor("%b | %b", address, val);

address=12'b0;
#1 address=12'b000000000001;
#1 address=12'b000000000010;
#1 address=12'b000000000011;
#1 address=12'b000000000100;
#1 address=12'b000000000101;
#1 address=12'b000000000110;
#1 address=12'b000000000111;
#1 address=12'b000000001000;
#1 address=12'b000000001001;
#1 address=12'b000000001010;
#1 address=12'b000000001011;
end
initial
#15 $finish;

initial begin

    $dumpfile("EJ2_tb.vcd");
    $dumpvars(0,testbench);
  end
endmodule
```

Diagrama de Timing

En el diagrama de timing escribimos las direcciones para obtener los valores que se encuentran en la memoria como se observa con los 6 primeros valores, del valor 006 al 008 se realizo un salto en el .list el cual es la @9 por lo que los valores anteriores no están declarados y por eso en el diagrama de timing son x. Luego del salto sigue obteniendo los valores que hay en la memoria.



Array, $readmemb y $readmemh

Para crear un array se escribe “reg” y el tamaño de bits al principio de la variable en este caso fue de 8 bits por lo que se escribió [0:7], luego de la variable se escribe el tamaño de las localidades en este caso era de 4096 por lo que se escribe [0:4095]. Los system task $readmemb y $readmemh, se utilizan para leer archivos en este caso .list que tendrán guardados los valores en cada una de sus localidades, donde se llama al archivo y se escribe junto a este el array que se utilizara. La diferencia entre estos es que el $readmemb es usado cuando se leen valores en binario y el $readmemh en hexadecimal.

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

Ejercicio 03

Se declaran las entradas A y B de 4 bits para las operaciones, la entrada F de 3 bits para elegir qué operación realizar y la salida de 4 bits. Un reg de 4 bits, el cual se usará en el case. Se usa el always para cada vez que cambien las entradas, donde el case elegirá la opción que se realizara entre A y B dependiendo de la entrada F, para al final asignarle a salida los valores de val.

```
module ALU (input wire [2:0]F,input wire[3:0]A, B, output wire[3:0]salida);
reg [3:0]val;
always @(F,A,B) begin
case(F)
3'b000: val <= A & B;
3'b001: val <= A | B;
3'b010: val <= A + B;
3'b011: val <= 4'b0;
3'b100: val <= A & ~B;
3'b101: val <= A | ~B;
3'b110: val <= A - B;
3'b111: val <= (A<B) ? 1:0;
default: val <= 4'b0;
endcase
end
assign salida = val;

endmodule
```

```
module testbench();

reg [2:0]F;
reg [3:0]A,B;
wire [3:0]salida;
ALU U1(F,A,B,salida);

initial begin
F=3'b011; A=4'b0; B=4'b0;
#1 A=4'b0101; B=4'b0100; F=3'b0;
#1 F=3'b001;
#1 F=3'b010;
#1 F=3'b011;
#1 F=3'b100;
#1 F=3'b101;
#1 F=3'b110;
#1 F=3'b111;
#1 A=4'b0100; B=4'b0101;

end

initial
#15 $finish;

initial begin

    $dumpfile("EJ3_tb.vcd");
    $dumpvars(0,testbench);
  end
endmodule
```

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

Diagrama de Timing

En las condiciones iniciales se utilizó la opción cuatro para tener los valores en cero en la primera opción se hace un AND entre A y B como se observa en la tabla, realizando cada una de las demás opciones. En la última opción de SLT se implemento que cuando A sea menor a B la salida sea 0 y en cualquier otro caso sea 1, justo como en la ultima salida del diagrama.

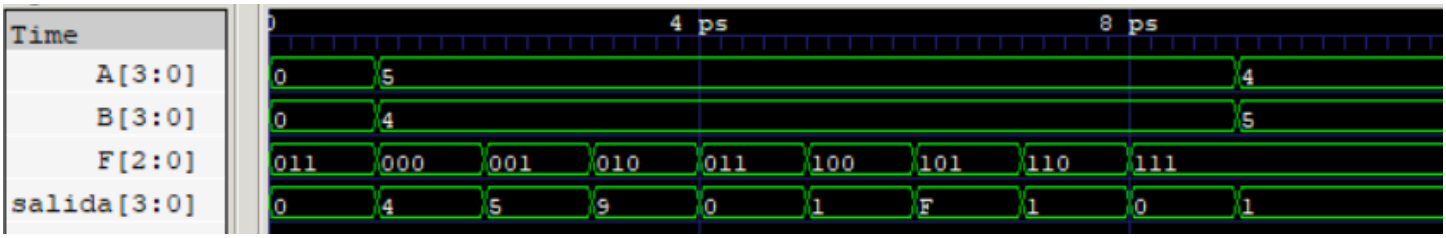


Table 5.1 ALU operations

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT