

Curso de Sistemas Digitales.

Instructor: Dr.Ing. Sergio A. Abreo C.

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Universidad Industrial de Santander



Semana: 9

Connectivity and Signal Processing Research Group.

info@cps.uis.edu.co <http://cps.uis.edu.co/index.php>

Agenda

- 1 Sesión 9
 - Divisores de Frecuencia
 - Memorias
- 2 Consulta
- 3 Agradecimientos
- 4 Referencias

Divisores de Frecuencia

Divisor 01

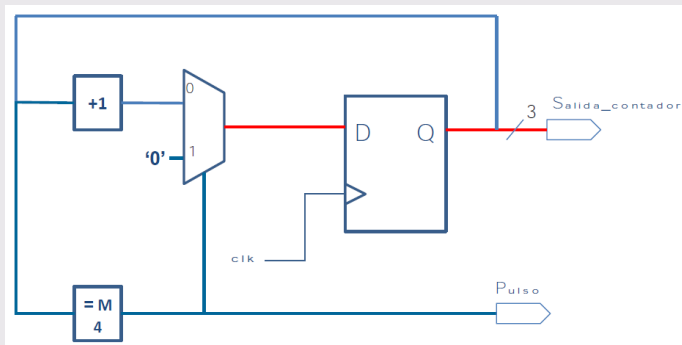
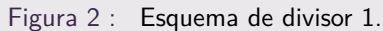


Figura 1 : Esquema de divisor 1.

Divisor 01



Divisores de Frecuencia

Divisor 01

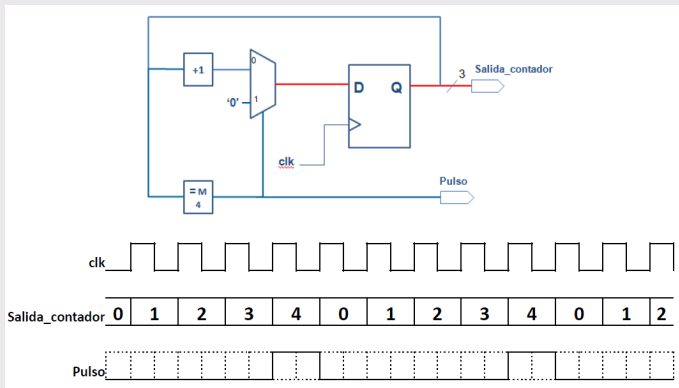


Figura 3 : Esquema de divisor 1.

Divisores de Frecuencia

Divisor 02

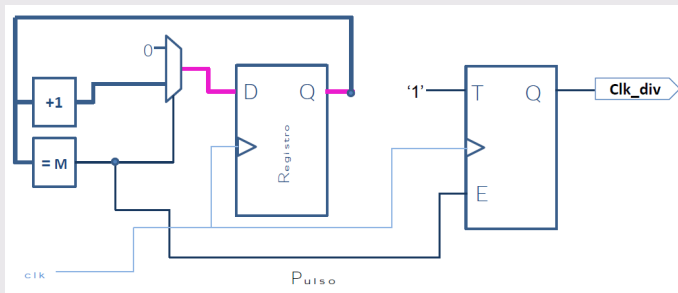


Figura 4 : Esquema de divisor 2.

Divisores de Frecuencia

Divisor 02

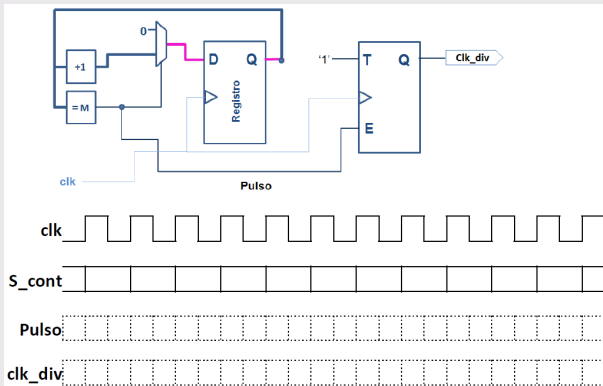


Figura 5 : Esquema de divisor 2.

Divisores de Frecuencia

Divisor 02

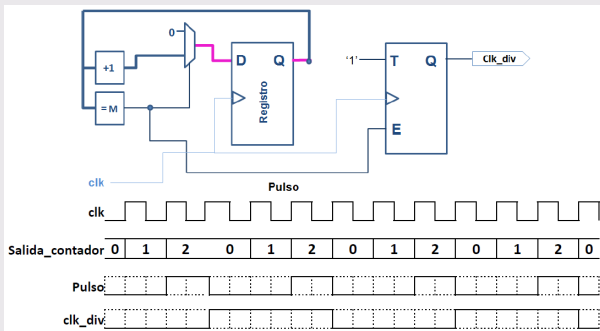


Figura 6 : Esquema de divisor 2.

Divisores de Frecuencia

Cuando cambia?

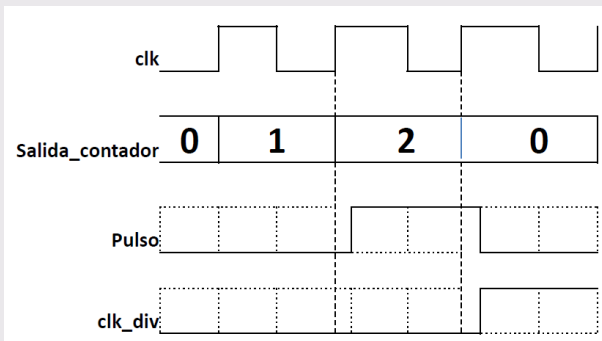


Figura 7 : Transición en la salida.

Divisores de Frecuencia

Cuando cambia?

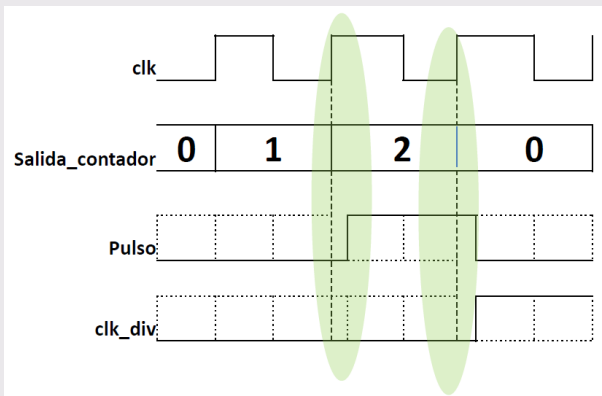


Figura 8 : Transición en la salida.

Memorias

Definición

- Las memorias son dispositivos que permiten almacenar bits de información.
- La información se puede guardar de manera permanente (ROM) o temporal (RAM).
- Se utilizan para guardar datos (Datos para ser procesados o programas).

Memorias

Diagrama General

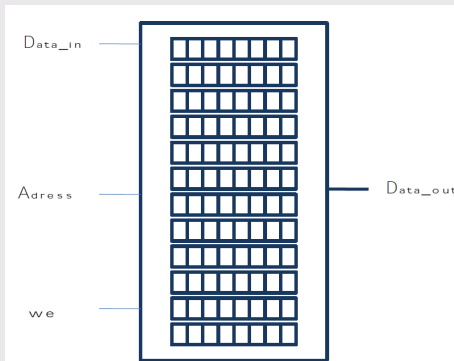


Figura 9 : Puertos.

Memorias

Diagrama General

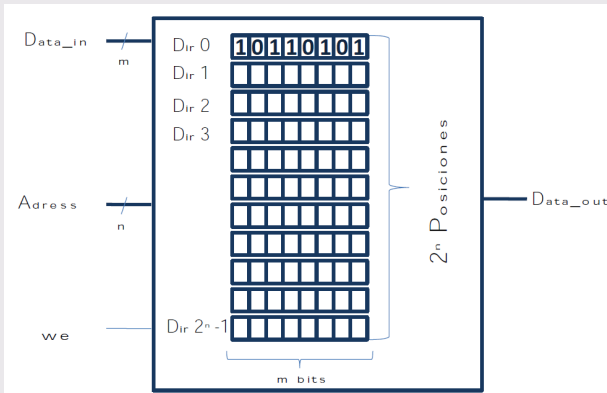


Figura 10 : Interior.

Memorias

Diagrama General

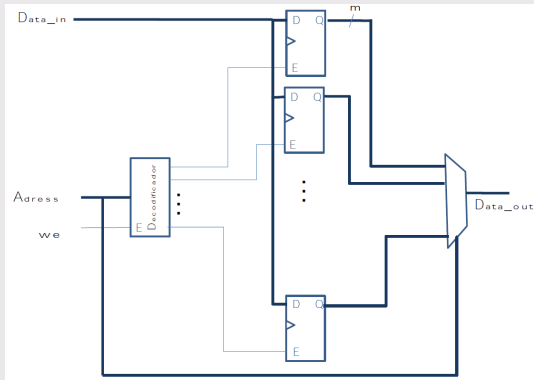


Figura 11 : Interior.

Memorias

Procedimiento

- 1 Se establece la dirección (Address) en la cual queremos guardar el dato.

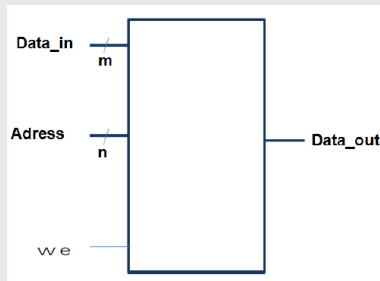


Figura 12 : Escritura.

Memorias

Procedimiento

- 2 El dato que queremos guardar se fija en la entrada de los datos (Data_in).

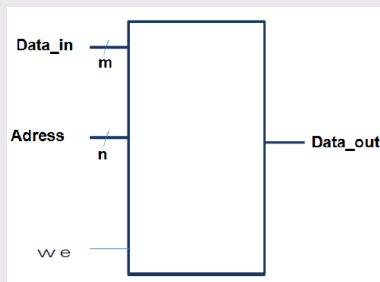


Figura 13 : Escritura.

Memorias

Procedimiento

- 3 Se establece un '1' en la entrada we.

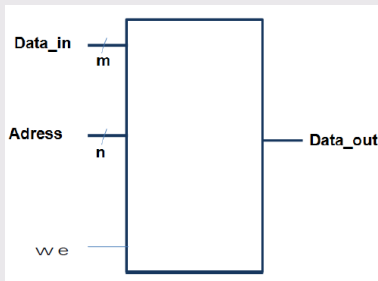


Figura 14 : Escritura.

Memorias

Procedimiento

- ❶ Se establece la dirección (Address) que se quiere leer.

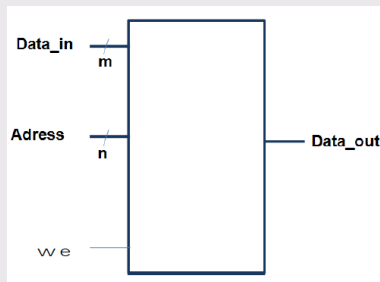


Figura 15 : Lectura.

Memorias

Procedimiento

- 2 Se establece un '0' en la entrada we.

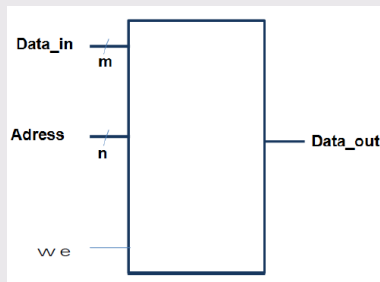


Figura 16 : Lectura.

Memorias

Procedimiento

- El contenido de la dirección seleccionada se lee en la salida de datos (Data_out).

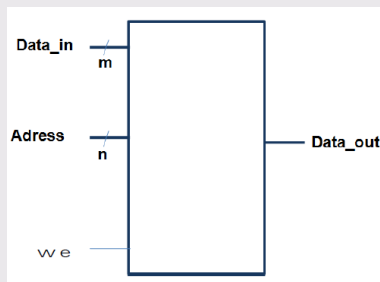


Figura 17 : Lectura.

Memorias

Tipos en las FPGAs

- Memoria Distribuida: se implementa usando los recursos lógicos de la FPGA.
- Bloques de Memoria RAM: se hace uso de los bloques de memoria que vienen dentro de la FPGA.
- Se debe usar la plantilla dada por el fabricante.

Verilog

Memoria

```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////
3  ///////////////////////////////////////////////////
4
5  module My_Memory
6      # (parameter N = 10, M = 32)
7      // N= Número de bits de la dirección, tamaño de la memoria.
8      // M = Al ancho de cada palabra.
9      (
10         input clk,
11         input we,
12         input [N-1:0] dir,
13         input [M-1:0] data_in,
14         output [M-1:0] data_out
15     );
16
17     reg [M-1:0] my_mem [2**N-1:0]; // Creo una variable tipo matriz de tamaño (M x 2**N)
18
19     always @ (posedge clk)
20     if (we) my_mem[dir] <= data_in;
21
22     assign data_out = my_mem[dir];
23
24 endmodule
25
```

Figura 18 : Distribuida.

Verilog

Memoria

```

1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////
3  ///////////////////////////////////////////////////
4  ///////////////////////////////////////////////////
5  module My_Memory
6      # (parameter N = 10, M = 32)
7      // N= Número de bits de la dirección, tamaño de la memoria.
8      // M = Al ancho de cada palabra.
9      (
10         input clk,
11         input we,
12         input [N-1:0] dir,
13         input [M-1:0] data_in,
14         output [M-1:0] data_out
15     );
16         Ancho      Profundidad
17     reg [M-1:0] my_mem [2**N-1:0]; // Creo una variable tipo matriz de tamaño (N x 2**N)
18
19     always @ (posedge clk)
20     if (we) my_mem [dir] <= data_in;
21
22     assign data_out = my_mem[dir];
23
24 endmodule
25

```

Figura 19 : Distribuida.

Memorias

Tipos según info en la salida

- Write First RAM: La salida muestra el dato que se está escribiendo en la posición de memoria.
- Read First RAM: La salida muestra el dato previo almacenado en la posición de memoria que se está escribiendo.
- No Change RAM: La salida se mantiene sin cambiar.

Verilog

Memoria

```

1 `timescale 1ns / 1ps
2 module ram_write_first #(parameter BITS_DATA = 4, // data width
3                           parameter BITS_ADDR = 3 // address width
4                           ) (
5     input CLK,                // Clock
6     input WE,                 // Write enable
7     input [BITS_ADDR-1 : 0] ADDR, // Address bus
8     input [BITS_DATA-1:0] D_in, // input data
9     output reg [BITS_DATA-1:0] D_out // output data
10 );
11
12 // Memory array (depth x width): 2^3=8_rows(depth) x 4_bits(width)
13 reg [BITS_DATA-1 : 0] memoria [0 : (2**BITS_ADDR)-1];
14 // data_width name memory_depth (big-endian:
15 // the most significant byte is at the lowest address)
16
17 // actualización memoria * * * * *
18 always @(posedge CLK)
19     if (WE)
20         memoria[ADDR] <= D_in;
21
22 // actualización salida: * * * * *
23 // Write First RAM: The output reflects the same data being written to the memory location when
24 // new data is written to the memory.
25 always @(posedge CLK)
26     if (WE)
27         D_out <= D_in;
28     else
29         D_out <= memoria[ADDR];
30
31 // The following code is only necessary if you wish to initialize the memory values to all zeros
32 generate
33     integer ram_index;
34     initial
35         for (ram_index = 0; ram_index < 2**BITS_ADDR; ram_index = ram_index + 1)
36             memoria[ram_index] = {BITS_DATA{1'b0}};
37 endgenerate
38 endmodule

```

Figura 20 : Write First.

Verilog

Memoria

```

1  `timescale 1ns / 1ps
2  module ram_read_first #(parameter BITS_DATA = 4, // data width
3                          parameter BITS_ADDR = 3 // address width
4                          ) (
5      input CLK,                // Clock
6      input WE,                 // Write enable
7      input [BITS_ADDR-1 : 0] ADDR, // Address bus
8      input [BITS_DATA-1:0] D_in, // Input data
9      output reg [BITS_DATA-1:0] D_out // output data
10 );
11
12 // Memory array (depth x width): 2^3=8_rows(depth) x 4_bits(width)
13 reg [BITS_DATA-1 : 0] memoria [0 : (2**BITS_ADDR)-1];
14 // data_width name memory_depth (big-endian:
15 // the most significant byte is at the lowest address)
16
17 // actualización memoria * * * * *
18 always @(posedge CLK)
19     if (WE)
20         memoria[ADDR] <= D_in;
21
22
23 // actualización salida: * * * * *
24 // Read First RAM: The output reflects the prior contents of the memory location when new data
25 // is written to the memory.
26 always @(posedge CLK )
27     D_out <= memoria[ADDR];
28
29
30 // The following code is only necessary if you wish to initialize the memory values to all zeros
31 generate
32     integer ram_index;
33     initial
34         for (ram_index = 0; ram_index < 2**BITS_ADDR; ram_index = ram_index + 1)
35             memoria[ram_index] = {BITS_DATA{1'b0}};
36 endgenerate
37
38 endmodule

```

Figura 21 : Read First.

Verilog

Memoria

```

1  `timescale 1ns / 1ps
2  module ram_no_change #(parameter BITS_DATA = 4, // data width
3                          parameter BITS_ADDR = 3 // address width
4                          ) (
5      input CLK,                // Clock
6      input WE,                // Write enable
7      input [BITS_ADDR-1 : 0] ADDR, // Address bus
8      input [BITS_DATA-1:0] D_in, // input data
9      output reg [BITS_DATA-1:0] D_out // output data
10 );
11
12 // Memory array (depth x width): 2^3=8_rows(depth) x 4_bits(width)
13 reg [BITS_DATA-1 : 0] memoria [0 : (2**BITS_ADDR)-1];
14 // data_width name memory_depth (big-endian:
15 // the most significant byte is at the lowest address)
16
17 // actualización memoria * * * * *
18 always @(posedge CLK)
19     if (WE)
20         memoria[ADDR] <= D_in;
21
22 // actualización salida: * * * * *
23 // No Change RAM: The output remains unchanged when new data is written to the memory.
24 always @(posedge CLK )
25     if (!WE)
26         D_out <= memoria[ADDR];
27
28
29 // The following code is only necessary if you wish to initialize the memory values to all zeros
30 generate
31     integer ram_index;
32     initial
33         for (ram_index = 0; ram_index < 2**BITS_ADDR; ram_index = ram_index + 1)
34             memoria[ram_index] = {BITS_DATA{1'b0}};
35 endgenerate
36
37
38 endmodule

```

Figura 22 : No change.

Verilog

Memoria

```

1 `timescale 1ns / 1ps
2 module TEST_RAM();
3   parameter PERIODO_CLK = 100;
4   parameter bits_a = 3;
5   parameter bits_d = 4;
6
7   reg reloj, escribir;
8   reg [bits_a-1:0] direccion;
9   reg [bits_d-1:0] dato_in;
10  wire [bits_d-1:0] D_OUT_WF, D_OUT_RF, D_OUT_NC;
11
12  ram_write_first #(bits_d,bits_a) unit_under_test_1(
13    .CLK(reloj),
14    .WE(escribir),
15    .ADDR(direccion),
16    .D_in(dato_in),
17    .D_out(D_OUT_WF)
18  );
19
20  ram_read_first #(bits_d,bits_a) unit_under_test_2(
21    .CLK(reloj),
22    .WE(escribir),
23    .ADDR(direccion),
24    .D_in(dato_in),
25    .D_out(D_OUT_RF)
26  );
27
28  ram_no_change #(bits_d,bits_a) unit_under_test_3(
29    .CLK(reloj),
30    .WE(escribir),
31    .ADDR(direccion),
32    .D_in(dato_in),
33    .D_out(D_OUT_NC)
34  );
35
36  always begin
37    reloj = 1'b0;    #(0.4*PERIODO_CLK);
38    reloj = 1'b1;    #(0.6*PERIODO_CLK);
39  end
40
41  initial begin
42    //número aleatorio entre (max,min)
43    dato_in = $random_range(15,1);
44
45    direccion = {bits_a{1'd1}};
46    escribir = 1'd0;
47    #(3*PERIODO_CLK);
48
49    forever begin
50      dato_in = $random_range(15,1);
51      direccion = direccion +1;
52      escribir = 1'd1;
53      #(1*PERIODO_CLK);
54
55      escribir = 1'd0;
56      #(2*PERIODO_CLK);
57    end
58  end
59
60 endmodule

```

Figura 23 : Test Bench para comparar los tres tipos.

Verilog

Memoria



En los tres tipos de memoria el dato de entrada (que en este momento es 3) es guardado en la **dirección 2** cuando se habilita la escritura.

- **write first:** D_OUT = 3 porque es el dato que se acaba de guardar en la **dirección 2**.
- **read first:** D_OUT = 0 porque es el dato que había la **dirección 2** antes de habilitar la escritura.
- **no change:** D_OUT = 1 porque es el dato que se leyó en el flanco de reloj antes de habilitar la escritura.

Figura 24 : Resultados de la simulación.

¿Donde Puedo Aprender Más?

Textos de Referencia.

- [Tocci and Widmer, 2003].
- [Harris and Harris, 2010].

Agradecimientos

Grupo CPS: Línea Sistemas Digitales.

La información presentada en estas diapositivas intenta recopilar los elementos pedagógicos desarrollados por los profesores Carlos Fajardo y Carlos Angulo en sus cursos de Sistemas Digitales I durante los últimos años de trabajo en esta línea.

Referencias I



Harris, D. and Harris, S. (2010).
Digital design and computer architecture.
Morgan Kaufmann.



Tocci, R. J. and Widmer, N. S. (2003).
Sistemas digitales: principios y aplicaciones.
Pearson Educación.