



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Arquitectura de Computadoras

“Implementación de Memoria ROM en Verilog”

Alumno:

Malagón Baeza Alan Adrian

Profesor:

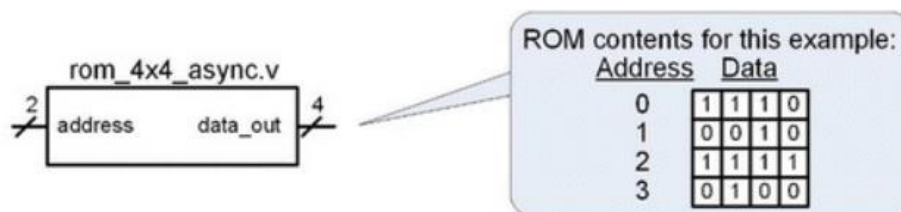
Alemán Arce Miguel Ángel

Grupo: 5CV1

Introducción

Una memoria ROM en Verilog se puede definir de dos maneras. La primera es simplemente usar una sentencia case para definir el contenido de cada ubicación en la memoria en función de la dirección entrante. Un segundo enfoque es declarar una matriz y luego inicializar su contenido. Cuando se usa una matriz, un bloque de procedimiento separado maneja la asignación del contenido de la matriz a la salida en función de la dirección entrante. La matriz puede ser inicializada utilizando un bloque inicial o a través de las tareas del sistema de E/S de archivos `$readmemb()` o `$readmemh()` (Opción 3). El siguiente ejemplo muestra dos enfoques para modelar una Memoria ROM 4x4. En este ejemplo, la memoria es asíncrona, lo que significa que como tan pronto como cambie la dirección, los datos de la ROM aparecerán inmediatamente. Para modelar este comportamiento asíncrono, los bloques de procedimiento son sensibles a la entrada entrante.

Ejemplo: Modelos con descripción de comportamiento de una memoria ROM asíncrona 4x4 en Verilog



Opción 1

Un enfoque simple para una rom es implementarla como una sentencia case

```
module rom_4x4_async
(output_reg [3:0] data_out,
input wire [1:0] address);

always @ (address)
case (address)
0      : data_out = 4'b1110;
1      : data_out = 4'b0010;
2      : data_out = 4'b1111;
3      : data_out = 4'b0100;
default : data_out = 4'bXXXX;
endcase

endmodule
```

Opción 2

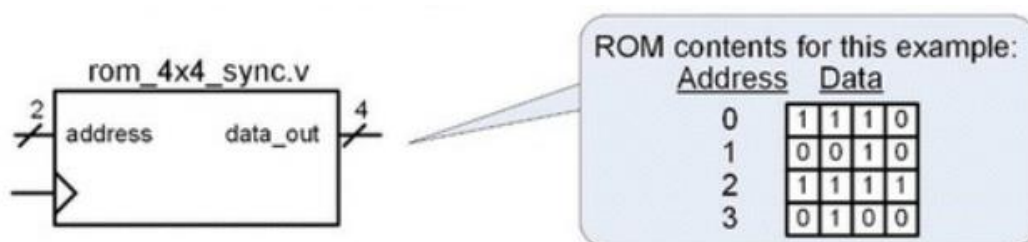
Un enfoque diferente es declarar una matriz y usar un bloque "inicial" para definir su contenido. Luego se usa un bloque always para asignar el vector direccionado a data_out.

```
module rom_4x4_async
(output reg [3:0] data_out,
input wire [1:0] address);
reg[3:0] ROM[0:3];
initial
begin
ROM[0] = 4'b1110;
ROM[1] = 4'b0010;
ROM[2] = 4'b1111;
ROM[3] = 4'b0100;
end
always @ (address)
data_out = ROM[address];
endmodule
```

Se declara una matriz MxN

Una ROM síncrona se puede crear de manera similar al enfoque asíncrono. La única diferencia es que, en una ROM síncrona, se usa un borde de reloj para activar el bloque de procedimiento que actualiza data_out. Se utiliza una lista de sensibilidad que contiene el reloj para activar la asignación. El siguiente ejemplo muestra dos modelos de Verilog para una ROM síncrona.

Ejemplo: Modelos con descripción de comportamiento de una memoria ROM síncrona 4x4 en Verilog



Opción 1

```
module rom_4x4_sync
  (output_reg [3:0] data_out,
   input wire [1:0] address,
   input wire Clock);

  always @ (posedge Clock) ←
  case (address)
    0      : data_out = 4'b1110;
    1      : data_out = 4'b0010;
    2      : data_out = 4'b1111;
    3      : data_out = 4'b0100;
    default : data_out = 4'bXXXX;
  endcase
endmodule
```

Opción 2

```
module rom_4x4_sync
  (output_reg [3:0] data_out,
   input wire [1:0] address,
   input wire Clock);

  reg[3:0] ROM[0:3];

  initial
  begin
    ROM[0] = 4'b1110;
    ROM[1] = 4'b0010;
    ROM[2] = 4'b1111;
    ROM[3] = 4'b0100;
  end

  always @ (posedge Clock) ←
    data_out = ROM[address];
endmodule
```

El comportamiento síncrono de estos modelos de ROM se logra haciendo que el bloque de procedimiento que actualiza data_out sea sensible al flanco ascendente del reloj.

Desarrollo

Opción 1

Código Verilog

```
rom.v x tb_rom.v x Schematic x
C:/Users/alanm/Desktop/Arqui/MemoriaROM/MemoriaROM.srscs/sources_1/new/rom.v

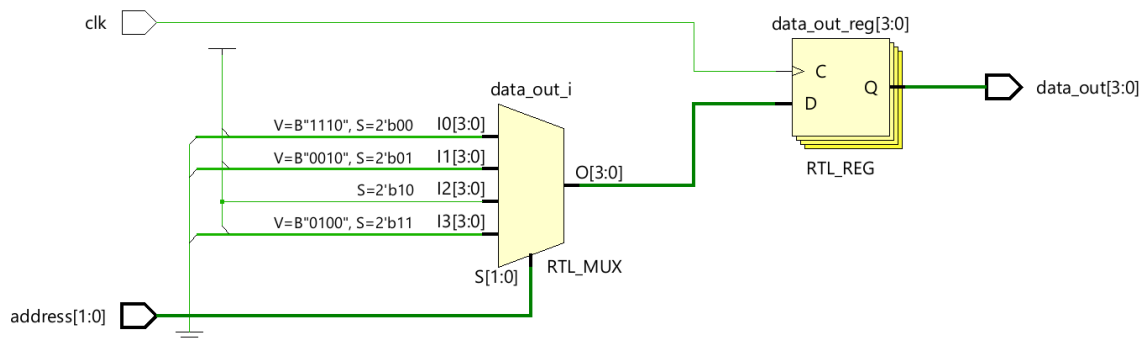
1 | timescale 1ns / 1ps
2 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 6CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Create Date: 25.03.2023 20:00:34
9 | // Design Name: rom
10 | // Module Name: rom
11 | // Project Name: MemoriaROM
12 | // Target Devices: Artix
13 | // Description: Version 1
14 | // Una declaración de caso para definir el contenido de cada ubicación en la
15 | // memoria en función de la dirección entrante
16 | //
17 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
18 |
19 | // Example: Behavioral Model of a 4x4 Synchronous Read Only Memory in Verilog
20 | // Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog,
21 | // Springer, 1st Edition, USA, 2017. pp 353
22 |
23 | module rom (
24 |     input          clk,
25 |     input  wire [1:0] address,
26 |     output reg  [3:0] data_out
27 | );
28 |
29 | always @(posedge clk)
30 |     case (address)
31 |         //      0          e
32 |         2'b00 : data_out = 4'b1110;
33 |         //      1          3
34 |         2'b01 : data_out = 4'b0010;
35 |         //      2          f
36 |         2'b10 : data_out = 4'b1111;
37 |         //      3          4
38 |         2'b11 : data_out = 4'b0100;
39 |         default : data_out = 4'bxxxx;
40 |     endcase
41 |
42 | endmodule
```

Código Verilog Testbench

```
rom.v x tb_rom.v x Schematic x
C:/Users/alanm/Desktop/Arqui/MemoriaROM/MemoriaROM.srscs/sim_1/new/tb_rom.v

1  `timescale 1ns / 1ps
2  //////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 6CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Create Date: 25.03.2023 20:00:34
9  // Design Name: rom
10 // Module Name: rom
11 // Project Name: MemoriaROM
12 // Target Devices: Artix
13 // Description: Version 1
14 // Una declaración de caso para definir el contenido de cada ubicación en la
15 // memoria en función de la dirección entrante
16 //
17 //////////////////////////////////////
18
19 //Modulo de Estimulo
20 module tb_rom ();
21
22     reg        clk;
23     reg  [1:0] address;
24
25     wire [3:0] data_out;
26
27     rom uut(clk, address, data_out);
28
29     initial clk = 1'b0;
30
31     always #10 clk = ~clk; //periodo de 20 unidades de tiempo
32
33     initial
34     begin
35         address = 2'b00; // 0
36         #20
37         address = 2'b11; // 3
38         #20
39         address = 2'b10; // 2
40         #20
41         address = 2'b01; // 1
42     end
43
44     initial #80 $finish;
45
46 endmodule
```

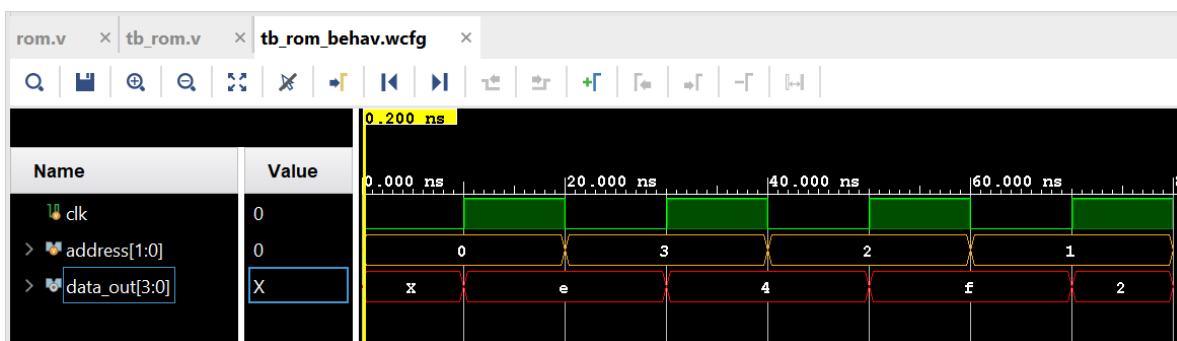
Implementación RTL en Vivado 2022.2



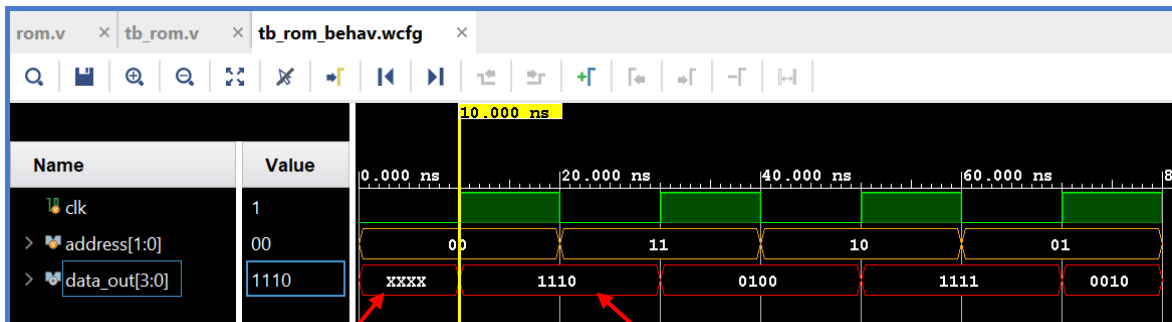
Podemos observar cómo utilizo FlipFlops y un multiplexor para hacer la implementación con un Case.

Resultado de la simulación:

En la simulación, se proporciona cada dirección posible (es decir, "00", "01", "10", y "11") para verificar que la ROM se inicializó correctamente. Antes del primer borde del reloj, el simulador no sabe qué asignar a data_out, por lo que enumera el valor como desconocido (X).



Hexadecimal



Binario

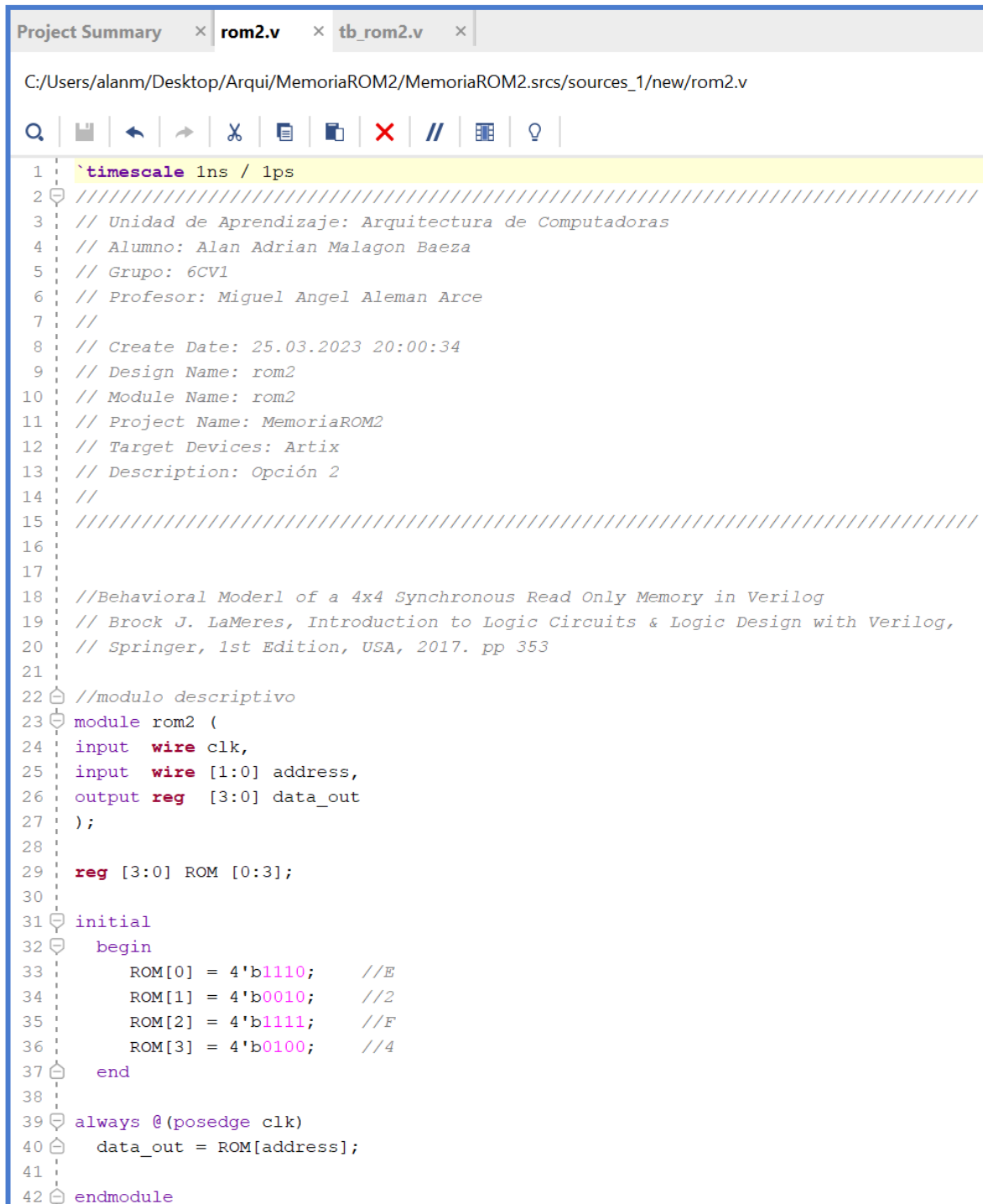
Antes del primer borde del reloj, el valor de data_out es desconocido (X).

Los datos no aparecen en la salida hasta un flanco ascendente del reloj.

Como se observa en la simulación, los resultados son los almacenados en la memoria de acuerdo con la localidad.

Opción 2

Código Verilog



```
Project Summary x rom2.v x tb_rom2.v x
C:/Users/alanm/Desktop/Arqui/MemoriaROM2/MemoriaROM2.srscs/sources_1/new/rom2.v

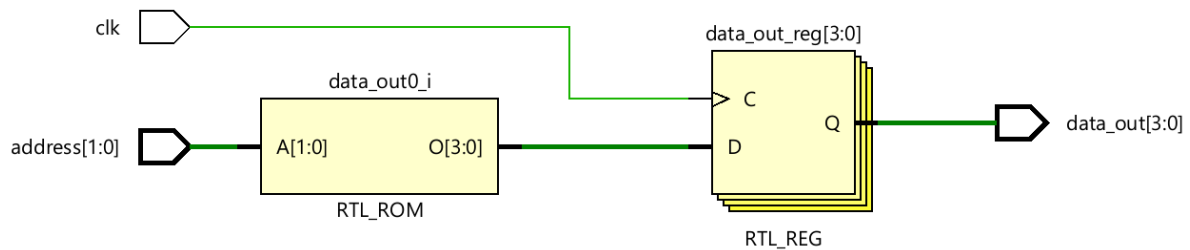
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 6CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Create Date: 25.03.2023 20:00:34
9  // Design Name: rom2
10 // Module Name: rom2
11 // Project Name: MemoriaROM2
12 // Target Devices: Artix
13 // Description: Opción 2
14 //
15 //////////////////////////////////////////////////
16
17
18 //Behavioral Model of a 4x4 Synchronous Read Only Memory in Verilog
19 // Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog,
20 // Springer, 1st Edition, USA, 2017. pp 353
21
22 //modulo descriptivo
23 module rom2 (
24     input wire clk,
25     input wire [1:0] address,
26     output reg [3:0] data_out
27 );
28
29     reg [3:0] ROM [0:3];
30
31     initial
32     begin
33         ROM[0] = 4'b1110;    //E
34         ROM[1] = 4'b0010;    //2
35         ROM[2] = 4'b1111;    //F
36         ROM[3] = 4'b0100;    //4
37     end
38
39     always @(posedge clk)
40     data_out = ROM[address];
41
42 endmodule
```

Código Verilog Testbench

```
tb_rom2.v x rom2.v x
C:/Users/alanm/Desktop/Arqui/MemoriaROM2/MemoriaROM2.srscs/sim_1/new/tb_rom2.v

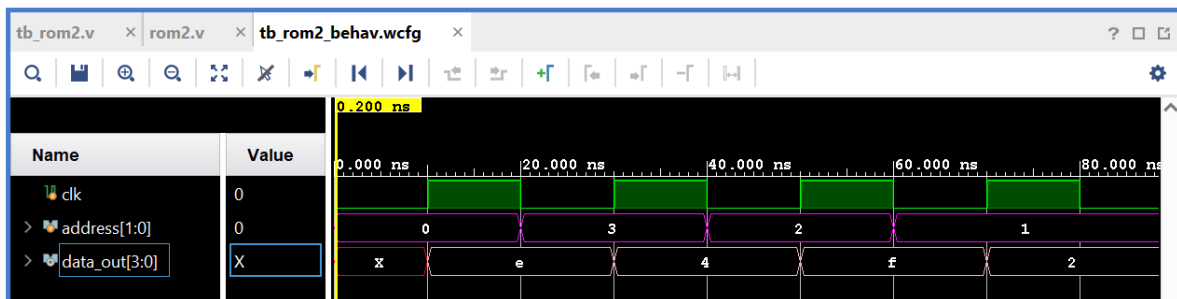
1 `timescale 1ns / 1ps
2 ///////////////////////////////////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 6CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Create Date: 25.03.2023 20:00:34
9 // Design Name: rom2
10 // Module Name: rom2
11 // Project Name: MemoriaROM2
12 // Target Devices: Artix
13 // Description: Opción 2
14 //
15 ///////////////////////////////////////////////////////////////////
16
17 //Modulo de estimulo
18 module tb_rom2 ();
19
20 //Inputs
21 reg clk;
22 reg [1:0] address;
23
24 //Outputs
25 wire [3:0] data_out;
26
27 //Instantiation of Unit Under Test
28 rom2 uut(clk, address, data_out);
29
30 initial clk = 1'b0;
31
32 always #10 clk = ~clk;
33
34 initial
35 begin
36     address = 2'b00;
37     #20
38     address = 2'b11;
39     #20
40     address = 2'b10;
41     #20
42     address = 2'b01;
43 end
44
45 endmodule
```

Descripción RTL en Vivado 2022.2

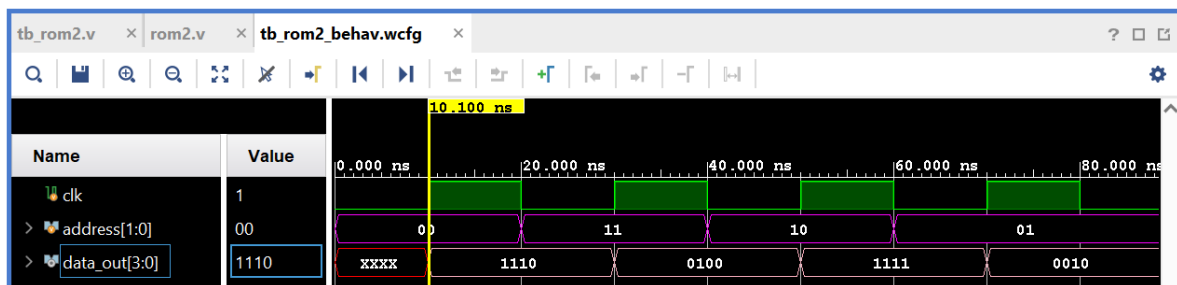


Como se puede observar se utilizó un bloque de memoria ROM del FPGA seleccionado, y para el control del dato a la salida, se implementó mediante flipflops con su terminal de clk.

Resultado de la simulación:



Hexadecimal



Binario

Como se observa en la simulación, los resultados son los almacenados en la memoria de acuerdo con la localidad.

Opción 3

Código Verilog

```
1 | `timescale 1ns / 1ps
2 | ///////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Create Date: 25.03.2023 20:00:34
9 | // Design Name: rom3
10 | // Module Name: rom3
11 | // Project Name: MemoriaROM3
12 | // Target Devices: Artix
13 | // Description: Opción 3: Inicializar arreglo por medio de archivo
14 | //
15 | ///////////////////////////////////////////////////////////////////
16 |
17 | //modulo descriptivo
18 | module rom3
19 | # (
20 |     parameter WIDTH = 4,
21 |     parameter DEPTH = 4,
22 |     parameter DEPTH_LOG = $clog2(DEPTH)
23 | )
24 | (
25 |     input clk,
26 |     input [DEPTH_LOG-1:0] addr_rd,
27 |     output reg [WIDTH-1:0] data_out
28 | );
29 | // Declaramos el arreglo de la ROM
30 | reg [WIDTH-1:0] rom [0:DEPTH-1];
31 |
32 | // Cargamos la rom con datos del archivo data.mem
33 | // La ubicación relativa depende de la ubicación del proyecto de Vivado
34 | initial
35 |     begin
36 |         $readmemb("data.mem",rom,0,DEPTH-1);
37 |     end
38 |
39 | // La lectura es síncrona
40 | always @(posedge clk)
41 |     data_out <= rom[addr_rd];
42 |
43 | endmodule
```

Código Verilog Testbench

read_file.v

C:/Users/alanm/Desktop/Arqui/Vivado/MemoriaROM2/MemoriaROM2.srscs/sim_1/new/read_file.v

```
1 | `timescale 1ns / 1ps
2 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 6CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Create Date: 25.03.2023 20:00:34
9 | // Design Name: rom3
10 | // Module Name: rom3
11 | // Project Name: MemoriaROM3
12 | // Target Devices: Artix
13 | // Description: Opción 3
14 | //
15 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
16 |
17 | //Modulo de estimulo
18 | module read_file ();
19 |
20 | localparam WIDTH= 4;
21 | localparam DEPTH = 4;
22 | localparam DEPTH_LOG = $clog2(DEPTH);
23 |
24 | //Inputs
25 | reg clk;
26 | reg [DEPTH_LOG-1:0] addr_rd;
27 | wire [WIDTH-1:0] data_rd;
28 |
29 | integer i;
30 |
31 | // Instanciamos el modulo
32 | rom3 #(.WIDTH(WIDTH),
33 |       .DEPTH(DEPTH)
34 |       ) ROM0
35 |     (
36 |       .clk (clk),
37 |       .addr_rd (addr_rd),
38 |       .data_out(data_rd)
39 |     );
40 |
41 | // Creamos la señal del reloj
42 | initial clk = 1'b0;
43 |
44 | always #10 clk = ~clk;
```

```

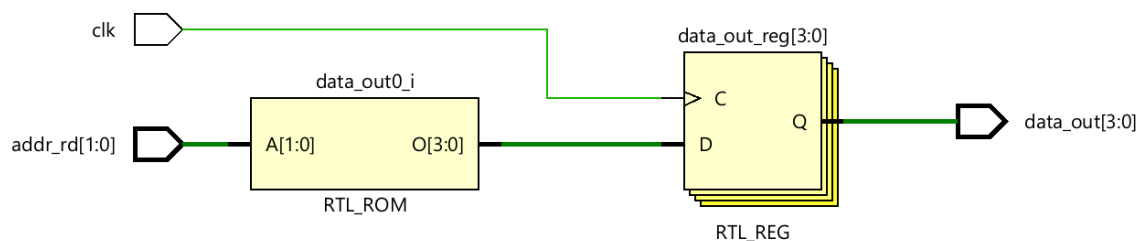
46 initial
47 begin
48     #1;
49     $display($time,"Contenido de la ROM:");
50     for(i=0;i<DEPTH;i=i+1) begin
51         read_data(i);
52     end
53     #40 $stop;
54 end
55
56 // Leemos los datos de forma asíncrona
57 task read_data(input[DEPTH_LOG-1:0] address_in);
58 begin
59     @(posedge clk);
60     addr_rd = address_in;
61     @(posedge clk);
62     #0.1; // Esperamos a que los datos se propaguen
63     $display($time,"address=%2d, data_rd=%x",addr_rd,data_rd);
64 end
65 endtask
66 endmodule

```

Archivo de memoria

data.mem	
C:/Users/alanm/[
Q	📁 ↩
1	1110
2	0010
3	1111
4	0100

Descripción RTL en Vivado 2022.2



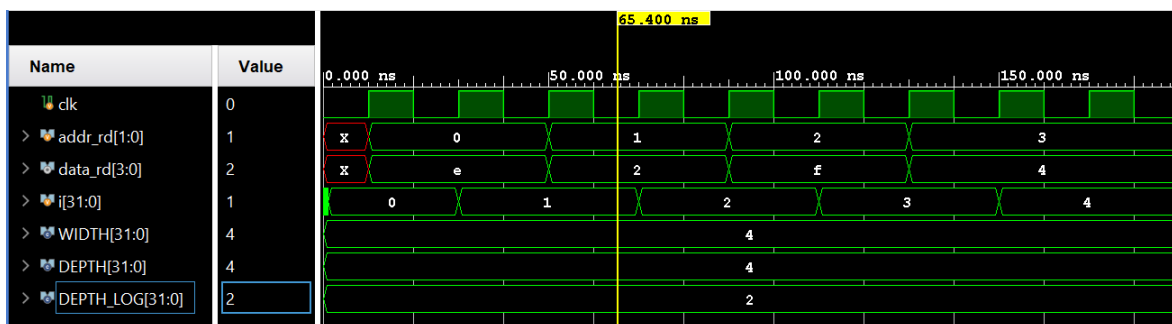
Como se puede observar se utilizó un bloque de memoria ROM del FPGA seleccionado, y para el control del dato a la salida, se implementó mediante flipflops con su terminal de clk.

Resultado de la consola:

Tiempo, dirección de la rom (address), contenido de la rom (data_rd)

```
1Contenido de la ROM:
30address= 0, data_rd=e
70address= 1, data_rd=2
110address= 2, data_rd=f
150address= 3, data_rd=4
```

Resultado de la simulación:



Como se observa en la simulación, los resultados son los almacenados en la memoria dados por el archivo data.mem de acuerdo con la localidad.

Conclusión

Las ventajas de modelar la memoria en Verilog son que permite abstraer los detalles de la celda de almacenamiento del funcionamiento funcional del sistema de memoria, es demasiado difícil modelar el comportamiento analógico de la celda de almacenamiento y hay demasiadas celdas para modelar, por lo que la simulación tardaría demasiado.

El término memoria se refiere a grandes conjuntos de almacenamiento digital. La tecnología utilizada en la memoria normalmente se optimiza para la densidad de almacenamiento a expensas de la capacidad de control. Esto es diferente de un D-flip-flop, que está optimizado para un control completo a nivel de bit.

Un dispositivo de memoria siempre contiene una entrada de bus de direcciones. El número de bits en el bus de direcciones determina a cuántas ubicaciones de almacenamiento se puede acceder. Un bus de direcciones de n-bits puede acceder a 2^n (o M) ubicaciones de almacenamiento.

El ancho de cada ubicación de almacenamiento (N) permite aumentar la densidad de la matriz de memoria al leer y escribir vectores de datos en lugar de bits individuales.

Un mapa de memoria es una representación gráfica de una matriz de memoria. Un mapa de memoria es útil para dar una visión general de la capacidad de la matriz y cómo se utilizan los diferentes rangos de direcciones de la matriz.

Una lectura es una operación en la que se recuperan datos de la memoria.

Una matriz de memoria asíncrona responde inmediatamente a sus entradas de control. Una matriz de memoria síncrona solo responde en el borde de activación del reloj.

La memoria ROM es un tipo de memoria en la que no se puede escribir durante el funcionamiento normal.

La arquitectura básica de una ROM consta de líneas de bits (verticales) y líneas de palabras (horizontales) que se cruzan y contienen celdas de almacenamiento en sus puntos de cruce. Los datos se leen de la matriz ROM utilizando las líneas de bits.

La memoria se puede modelar en Verilog utilizando un tipo de datos de matriz que consta de elementos de tipo reg.

Referencia

1. Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog, Springer, 1st Edition, USA, 2017.