Instituto Politécnico Nacional
Escuela Superior de Cómputo
“ESCOM”

Unidad de Aprendizaje:
Diseño de sistemas digitales

Entregable No. 12
Memoria RAM

Integrantes:
Álvarez Hernández Gabriel Alexander
García Quiroz Gustavo Iván
Huesca Laureano Josué Alejandro
Muñoz Valdivia Irving Omar
Pedroza Villagómez Emir

Nombre del profesor: Flores Escobar José Antonio

Índice de contenido

Introducción

En el ámbito de los sistemas digitales, las memorias RAM (Random Access Memory) juegan un papel fundamental como componentes esenciales para el almacenamiento temporal de datos. Esta práctica se centra en el diseño, implementación y simulación de una memoria RAM parametrizable utilizando el lenguaje de descripción de hardware Verilog. El proyecto no solo sirve como una valiosa herramienta educativa para comprender los principios de diseño de memorias digitales, sino que también proporciona una base sólida para futuras aplicaciones en sistemas más complejos.

La memoria RAM diseñada en esta práctica se caracteriza por su flexibilidad y adaptabilidad, gracias a su naturaleza parametrizable. Los parámetros N y M permiten ajustar el tamaño de la dirección y el ancho de los datos, respectivamente, lo que hace que el diseño sea versátil y aplicable a una variedad de escenarios. Además, la implementación incluye puertos separados para lectura y escritura, lo que mejora la eficiencia en operaciones simultáneas y refleja las arquitecturas comúnmente utilizadas en sistemas de memoria modernos.

Un aspecto destacado de este diseño es la incorporación de un vector de validación (VALID), que añade una capa adicional de control sobre los datos almacenados. Esta característica mejora la integridad y seguridad de la memoria, permitiendo un manejo más preciso de qué direcciones contienen datos válidos. Tal enfoque es particularmente relevante en aplicaciones donde la gestión de la validez de los datos es crucial.

El proceso de desarrollo de esta práctica abarca varias etapas clave del diseño de hardware digital. Comienza con la implementación del módulo principal de la memoria RAM en Verilog, seguido por la creación de un testbench exhaustivo para la verificación del diseño. El testbench simula diversas condiciones de operación, incluyendo la inicialización de la memoria, operaciones de lectura y escritura, y el manejo del reset, proporcionando así una validación completa del funcionamiento del módulo.

La simulación juega un papel crucial en este proyecto, permitiendo la observación detallada del comportamiento de la memoria bajo diferentes escenarios. Mediante el uso de herramientas de simulación, se pueden visualizar las señales clave y verificar la correcta operación de la memoria en tiempo real. Este proceso no solo valida el diseño, sino que también proporciona insights valiosos sobre el funcionamiento interno de la memoria RAM.

Esta práctica no solo se enfoca en la implementación técnica, sino que también resalta la importancia de la documentación y el análisis en el proceso de diseño de hardware. A través de la elaboración de reportes detallados, incluyendo la descripción del desarrollo, la implementación del código, los resultados de la simulación y las conclusiones, se fomenta una comprensión profunda de cada aspecto del proyecto.

Marco teórico

Memoria

Memoria es la sección de un sistema digital encargada del almacenamiento permanente o temporal de la información del sistema.

Sobre la memoria se pueden hacer dos operaciones:

- Lectura: la memoria presenta en sus terminales de salida el contenido de la localidad seleccionada. La lectura se puede hacer en todos los tipos de memoria.
- Escritura: se escribe el dato presente en los terminales de entrada en la localidad seleccionada. No todos los tipos de memoria permiten la escritura.

La ejecución de una operación de lectura o escritura se denomina acceso a memoria.

Memorias de acceso aleatorio (RAM)

Características comunes de las RAM

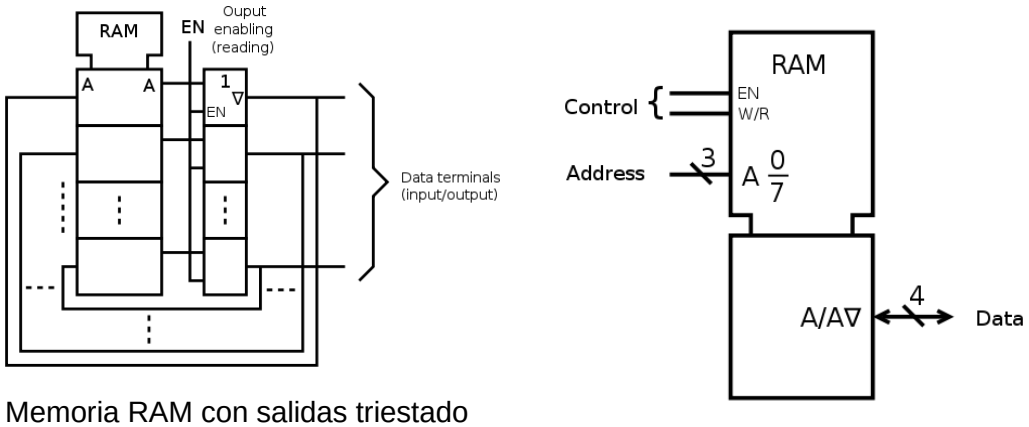
El *tiempo de acceso* (lectura o escritura) es constante e independiente de la posición de la localidad sobre la que se opere.

Los terminales de una RAM se dividen en tres grupos:

Terminales de dirección: Indican la localidad sobre la que se quiere operar.

Terminales de datos: Presentan del dato de una operación de lectura o proporcionan el dato de una operación de escritura.

Terminales de control: Indican a la memoria la operación a realizar (lectura o escritura). También incluyen un terminal que inhibe las salidas forzándolas al tercer estado. En los sistemas electrónicos, las memorias se conectan con un bus de datos, por lo que deben estar dotadas de tercer estado.



Memoria RAM con salidas triestado

Bloque funcional de una RAM 8x4
(8 localidades de 4 bits cada una)

Memorias RAM volátiles

Las memorias volátiles pierden la información cuando no tienen alimentación eléctrica.

Las RAM volátiles se pueden clasificar por el tiempo que permanece la información aunque se mantenga alimentadas eléctricamente en:

- Estáticas (SRAM, Static RAM)

Mantienen la información siempre y cuando estén alimentadas eléctricamente. *Funcionalmente* las celdas de estas memorias son biestables D.

- Dinámicas (DRAM; Dinamic RAM)

Aunque se mantenga el suministro eléctrio pierden la información, por lo que se reescriben constantemente (refresco). Las celdas son condensadores MOS.

Otro criterio clasificatorio es por la simultaneidad de las operaciones de lectura y escritura (L/E):

- L/E no simultáneas

En un determinado instante sólo se pueden leer o escribir. Disponen de los siguientes terminales de control:

- Chip enable (CE o CS): inhibe (0) o habilita (1) la memoria
- Out Enable (OE): inhibe (0) las salidas al tercer estado o las habilita (1) si CE=1
- Write Enable (WE): inhibe (0) o habilita (1) la escritura si CE=1 y OE=0
- L/E simultáneas

Se puede leer en una localidad y simultáneamente escribir en otra. La memoria debe disponer de terminales independientes de dirección de escritura, dirección de lectura, líneas de entrada (escritura) y líneas de salida (lectura).

Desarrollo

Implementación del código Verilog

Módulo MemoriaRAM

```
/*
   Proyecto: MemoriaRAM
   Archivo: MemoriaRAM.v
   Descripcion: Descripcion de una MemoriaRAM
                parametrizable con puertos separados
                para lectura y escritura
   Asignatura: DSD
   Prof: Flores Escobar Jose Antonio
   Equipo: Coloque a los integrantes...
            Álvarez Hernández Gabriel Alexander
            Bueno Aguilar Alexis Haziel
            García Quiroz Gustavo Iván
            Huesca Laureano Josué Alejandro
            Muñoz Valdivia Irving Omar
            Pedroza Villagómez Emir
*/
```

```
module MemoriaRAM #(
   parameter N = 4,
   parameter M = 4
)
(
   //Señales generales
   input             clk_i,
   input             rst_i,
   input    [N-1:0]  addr_i,
   //Señales del puerto de lectura
   input             rden_i,
   output   [M-1:0]  dato_read_o,
   //Señales del puerto de escritura
   input    [M-1:0]  dato_write_i,
   input             wren_i
);

   //Definicion de la memoria
   reg   [M-1:0]  RAM   [0:2**N-1];
```

```
   //Definición de vector de validación
   reg   [0:2**N-1]  VALID;

   //Inicializacion de la memoria
   initial
   begin
      $readmemh("ram_init.hex", RAM);
   end

   //Puerto de escritura
   always @(posedge clk_i, posedge rst_i)
   begin
      if(rst_i)
         VALID <= 16'b0011111100011101;
      else
         if(wren_i)
            RAM[addr_i] <= dato_write_i;
   end

   //Puerto de lectura
   assign dato_read_o = (rden_i && VALID[addr_i]) ? RAM[addr_i] : {M{1'b0}};

endmodule
```

Figura 1 Módulo MemoriaRAM

El módulo MemoriaRAM implementa una memoria RAM parametrizable con puertos separados para lectura y escritura. Utiliza una matriz de registros para almacenar los datos y un vector de validación para controlar qué direcciones contienen datos válidos. La memoria se inicializa desde un archivo hexadecimal externo. La escritura se realiza de forma síncrona en el flanco de subida del reloj cuando la señal de escritura está activa, mientras que la lectura es asíncrona y solo devuelve datos si la señal de lectura está activa y la dirección es válida.

Módulo MemoriaRAM_fpga

```
/*
   Proyecto: MemoriaRAM
   Archivo: MemoriaRAM_fpga.v
   Descripcion: Descripcion de una MemoriaRAM
                parametrizable con puertos separados para lectura y escritura
   Asignatura: DSD
   Prof: Flores Escobar Jose Antonio
   Equipo: Coloque a los integrantes...
            Álvarez Hernández Gabriel Alexander
            Bueno Aguilar Alexis Haziel
            García Quiroz Gustavo Iván
            Huesca Laureano Josué Alejandro
            Muñoz Valdivia Irving Omar
            Pedroza Villagómez Emir*/
module MemoriaRAM_fpga #(
    parameter N = 4,
    parameter M = 4
)
(
    // Señales generales
    input wire          clk_i,
    input wire          rst_i,
    input wire [N-1:0]  addr_i,
    // Señales del puerto de lectura
    input wire          rden_i,
    output wire [M-1:0] dato_read_o,
    // Señales del puerto de escritura
    input wire [M-1:0]  dato_write_i,
    input wire          wren_i,
    // Salidas del display
    output wire [0:6]   display_o,
    output wire         display_enable
);
```

```
    // Seccion de definicion de señales
    wire        slow_clk;

    // Instancia de la memoria RAM
     MemoriaRAM #(
         .N(N),
         .M(M)
     )RAM_U0(
         .clk_i(clk_i),
         .rst_i(rst_i),
         .addr_i(addr_i),
         .rden_i(rden_i),
         .dato_read_o(dato_read_o),
         .dato_write_i(dato_write_i),
         .wren_i(wren_i)
     );

     // Instancia del divisor de frecuencia
     DivFreq #(
         .freqdev(10_000_000), // Frecuencia de desarrollo de 10MHz
         .freqfinal(4)         // Frecuencia final dividida en 4
     )div_freq_instance(
         .clk_i(clk_i),
         .rst_ni(rst_i),
         .clk_o(slow_clk)      // Salida de reloj dividido
     );

     // Instancia del display
     display display_instance (
         .cuenta_i    (dato_read_o),
         .enable_i    (1'b1),        // Habilitamos el display siempre
         .display_o   (display_o),
         .daenable_o  (display_enable)
     );
endmodule
```

Figura 2 Módulo MemoriaRAM_fpga

Este módulo actúa como un wrapper para la implementación en FPGA, integrando la memoria RAM con componentes adicionales para su visualización y control. Instancia el módulo MemoriaRAM principal, un divisor de frecuencia para generar un reloj más lento, y un módulo de display de 7 segmentos para mostrar el dato leído. Este diseño facilita la integración de la memoria RAM en un sistema FPGA más amplio, proporcionando interfaces para la visualización de datos y el control de frecuencia.

Módulo DivFreq

```
/* Proyecto: MemoriaRAM
   Archivo: DivFreq.v
   Descripcion: Descripcion de un divisor de frecuencia
   Asignatura: DSD
   Prof: Flores Escobar Jose Antonio
   Equipo: Coloque a los integrantes...
            Álvarez Hernández Gabriel Alexander
            Bueno Aguilar Alexis Haziel
            García Quiroz Gustavo Iván
            Huesca Laureano Josué Alejandro
            Muñoz Valdivia Irving Omar
            Pedroza Villagómez Emir*/
module DivFreq #(
    parameter freqdev = 10_000_000, // 10MHz
    parameter freqfinal = 4 // freqdev Dividido en 4
)(
    input clk_i,
    input rst_ni,
    output reg clk_o);
    reg [31:0] counter_r;
    always @(posedge clk_i or posedge rst_ni) begin
        if (rst_ni) begin
            counter_r <= 32'b0;
            clk_o <= 1'b1; // Comenzamos con el reloj en bajo
        end else begin
        if (counter_r == (32'd10_000)) begin
            counter_r <= 32'b0;
            clk_o <= ~clk_o;
        end else begin
            counter_r <= counter_r + 32'b1;
        end
        end
    end
endmodule
```

Figura 3 DivFreq

El módulo DivFreq implementa un divisor de frecuencia configurable mediante parámetros. Utiliza un contador interno para dividir la frecuencia de entrada del reloj. El módulo genera una señal de reloj de salida con una frecuencia reducida, que se determina por los parámetros de frecuencia de desarrollo y frecuencia final deseada. Este diseño permite adaptar fácilmente la frecuencia de operación del sistema a diferentes requisitos sin necesidad de modificar el código del módulo.

Módulo display

```
/*
   Proyecto: MemoriaRAM
   Archivo: display.v
   Descripcion: Descripcion de un display

   Asignatura: DSD
   Prof: Flores Escobar Jose Antonio
   Equipo: Coloque a los integrantes...
            Álvarez Hernández Gabriel Alexander
            Bueno Aguilar Alexis Haziel
            García Quiroz Gustavo Iván
            Huesca Laureano Josué Alejandro
            Muñoz Valdivia Irving Omar
            Pedroza Villagómez Emir

*/
module display (
   input    [3:0] cuenta_i,
   input          enable_i,
   output   [0:6] display_o,
   output         daenable_o
);

   reg [0:6] display_w;
   assign daenable_o = 1'b1;
   always @(*)
   begin
      case (cuenta_i)
         4'b0000:
            display_w = 7'b0111111;
         4'b0001:
            display_w = 7'b0000110;
         4'b0010:
            display_w = 7'b1011011;
         4'b0100:
            display_w = 7'b1100110;
         4'b0101:
            display_w = 7'b1101101;
         4'b0110:
            display_w = 7'b1111101;
         4'b0111:
            display_w = 7'b0000111; //7
         4'b1000:
            display_w = 7'b1111111; //8
         4'b1001:
            display_w = 7'b1101111; //9
         4'b1010:
            display_w = 7'b1110111; //A
         4'b1011:
            display_w = 7'b1111100; //B
         4'b1100:
            display_w = 7'b0111001; //C
         4'b1101:
            display_w = 7'b1011110; //D
         4'b1110:
            display_w = 7'b1111001; //E
         4'b1111:
            display_w = 7'b1110001; //F
         default:
            display_w = 7'b0000000;
      endcase
   end
   assign display_o = (enable_i) ? display_w : ~display_w;

endmodule
```

Figura 4 display

El módulo display controla un display de 7 segmentos para mostrar dígitos hexadecimales. Utiliza una estructura case para decodificar el valor de entrada de 4 bits en las señales correspondientes para los 7 segmentos del display. El módulo puede mostrar dígitos del 0 al 9 y letras de la A a la F, cubriendo así todo el rango hexadecimal. Incluye una señal de

habilitación que permite controlar cuándo se activa el display, lo que facilita su integración en sistemas más complejos con múltiples displays o modos de visualización.

Módulo ram_init.hex

```
F
E
D
C
B
A
9
8
7
6
5
4
3
2
1
0
```

Figura 5 ram_init.hex

El archivo ram_init.hex contiene 16 valores hexadecimales, cada uno en una línea separada, que se utilizan para inicializar la memoria RAM en el módulo MemoriaRAM. Los valores están en orden descendente desde F hasta 0, lo que corresponde a los números hexadecimales del 15 al 0. Esta inicialización se realiza mediante la siguiente línea en el módulo MemoriaRAM.

Módulo MemoriaRAM_tb

El módulo MemoriaRAM_tb es un testbench diseñado para verificar el funcionamiento del módulo MemoriaRAM. Utiliza una escala de tiempo de 100ps y define señales de prueba que corresponden a las entradas y salidas del módulo de memoria. El testbench genera un reloj con un período de 2ns y ejecuta una secuencia de pruebas que incluye la inicialización de la memoria, un reset, lectura de valores iniciales, escritura de nuevos valores, y lectura de estos nuevos valores. Utiliza bucles para acceder a todas las 16 direcciones de memoria, tanto para lectura como para escritura. Además, emplea las funciones $display para mostrar los resultados de las lecturas y $monitor para realizar un seguimiento continuo de las señales clave durante la simulación. Esta estructura permite una verificación completa y sistemática del comportamiento de la memoria RAM bajo diferentes condiciones de operación.

```
`timescale 100ps / 1ps

module MemoriaRAM_tb();

// Parámetros
parameter N = 4;
parameter M = 4;

// Señales de prueba
reg clk_i;
reg rst_i;
reg [N-1:0] addr_i;
reg rden_i;
wire [M-1:0] dato_read_o;
reg [M-1:0] dato_write_i;
reg wren_i;

integer i; // Variable para bucles

// Instancia del módulo bajo prueba
MemoriaRAM #(
    .N(N),
    .M(M)
) uut (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .addr_i(addr_i),
    .rden_i(rden_i),
    .dato_read_o(dato_read_o),
    .dato_write_i(dato_write_i),
    .wren_i(wren_i)
);

// Generación de reloj
initial begin
    clk_i = 0;
    forever #1 clk_i = ~clk_i; // Período de reloj de 2ns
end

// Procedimiento de prueba
initial begin
    // Inicialización
    rst_i = 1;
    addr_i = 0;
    rden_i = 0;
    dato_write_i = 0;
    wren_i = 0;
```

```
    // Liberación del reset
    #10 rst_i = 0;

    // Lectura de valores iniciales
    #2;
    for (i = 0; i < 16; i = i + 1) begin
        addr_i = i;
        rden_i = 1;
        #2;
        $display("Dirección %0d: %h", addr_i, dato_read_o);
    end

    // Escritura de nuevos valores
    #2;
    wren_i = 1;
    for (i = 0; i < 16; i = i + 1) begin
        addr_i = i;
        dato_write_i = i + 1;
        #2;
    end
    wren_i = 0;

    // Lectura de nuevos valores
    #2;
    for (i = 0; i < 16; i = i + 1) begin
    for (i = 0; i < 16; i = i + 1) begin
        addr_i = i;
        rden_i = 1;
        #2;
        $display("Dirección %0d después de escritura: %h", addr_i, dato_read_o);
    end

    // Fin de la simulación
    #20 $finish;
end

// Monitoreo de señales
initial begin
    $monitor("Tiempo=%0t, Addr=%h, Read=%b, Write=%b, DataIn=%h, DataOut=%h",
             $time, addr_i, rden_i, wren_i, dato_write_i, dato_read_o);
end

endmodule
```

Figura 6 MemoriaRAM_tb

Análisis del circuito RTL

El circuito RTL generado por Quartus II para este proyecto de memoria RAM muestra la estructura lógica del diseño a nivel de registros y lógica combinacional. En el centro del diseño se encuentra el bloque de memoria RAM, representado como una matriz de elementos de almacenamiento (flip-flops) organizados en filas y columnas correspondientes a las direcciones y los bits de datos respectivamente.

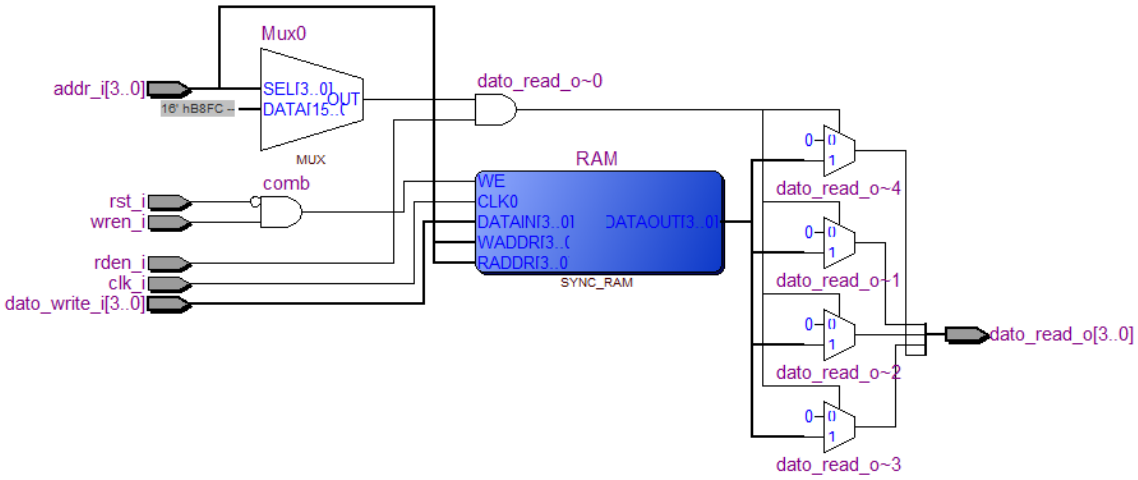


Figura 7 Análisis del circuito RTL

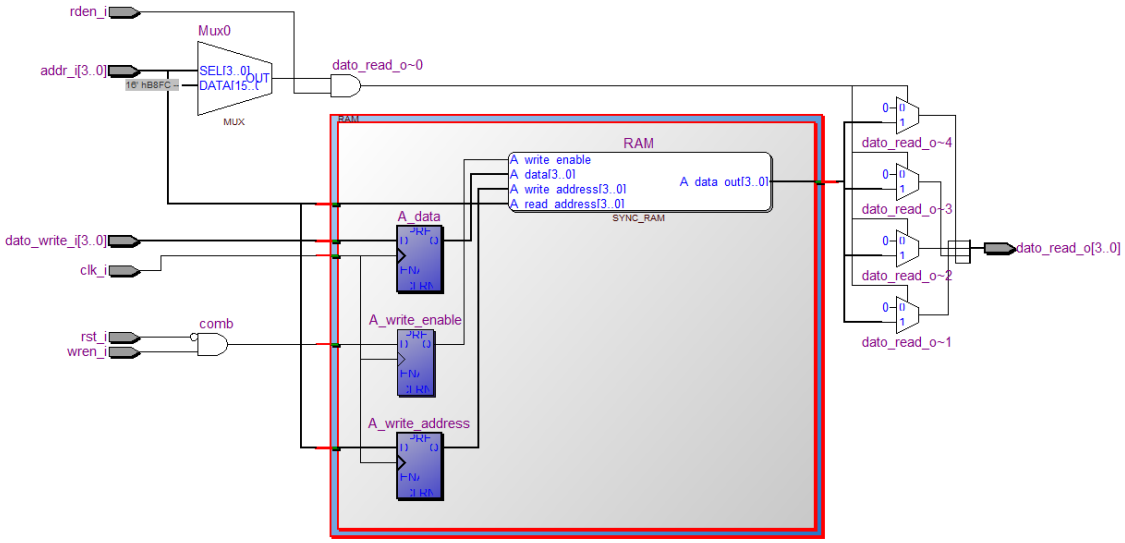


Figura 8 Análisis de la RAM extendida del circuito RTL

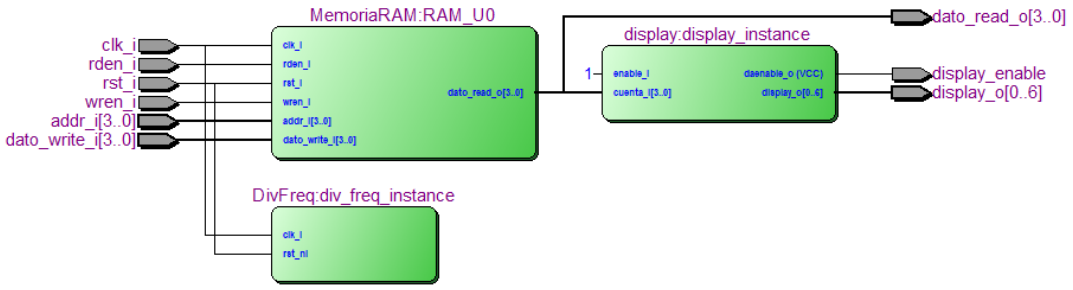


Figura 9 Análisis del circuito RTL para la implementación en FPGA.

Conectado a la memoria RAM, se observa la lógica de decodificación de direcciones. Esta lógica utiliza el bus de direcciones (addr_i) para seleccionar la fila específica de la memoria que se va a acceder. La decodificación se implementa típicamente como una serie de multiplexores o como una estructura de árbol de decodificación, dependiendo de la optimización realizada por el sintetizador.

Para el puerto de escritura, el RTL muestra una red de puertas AND que combinan la señal de habilitación de escritura (wren_i) con la señal de reloj (clk_i). Estas puertas controlan la activación de los flip-flops en la fila seleccionada de la memoria, permitiendo que los nuevos datos (dato_write_i) se almacenen solo cuando la escritura está habilitada y en el flanco de subida del reloj.

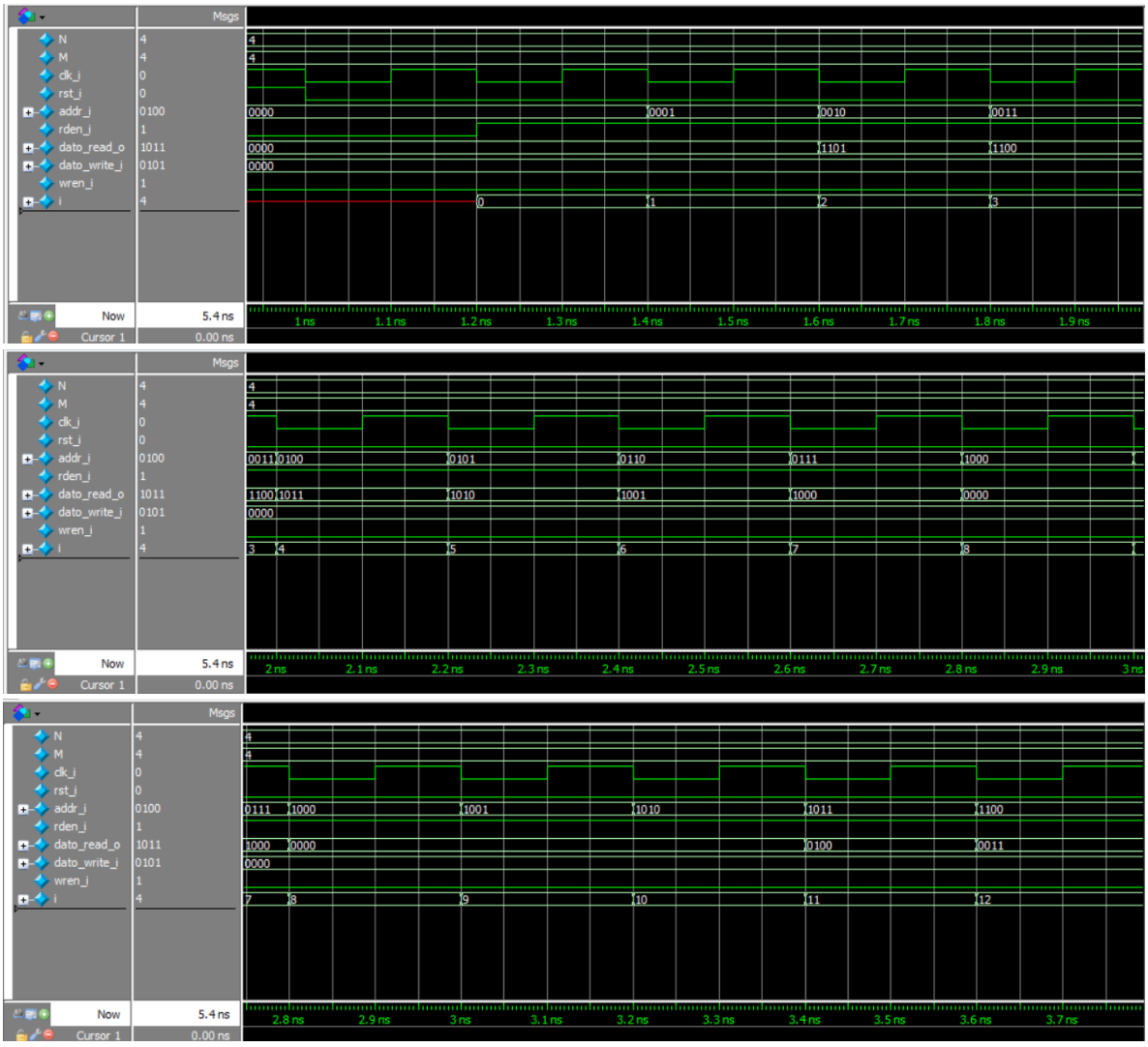
El puerto de lectura se implementa como un conjunto de multiplexores que seleccionan los datos de la fila de memoria correspondiente a la dirección de lectura. La salida de estos multiplexores se combina con la señal de habilitación de lectura (rden_i) y el vector de validación (VALID) a través de puertas AND, asegurando que solo se lean datos válidos cuando se solicite.

El circuito RTL también muestra el divisor de frecuencia como un contador con lógica de comparación y un flip-flop para generar la señal de reloj dividida. Este contador se incrementa con cada ciclo del reloj principal y reinicia cuando alcanza el valor de división especificado.

Para el módulo de display, el RTL presenta una estructura de decodificación, típicamente implementada como una tabla de consulta (LUT) o una serie de puertas lógicas, que convierte el valor de 4 bits de entrada en las señales apropiadas para los 7 segmentos del display.

Simulaciónes

En la simulación de este código, se debería observar el siguiente comportamiento: Inicialmente, la memoria RAM se carga con valores del archivo "ram_init.hex". Tras el reset, se lee el contenido inicial de las 16 direcciones de memoria, mostrando los valores cargados. Luego, se realiza una operación de escritura en todas las direcciones, asignando valores incrementales (1 a 16). Finalmente, se vuelven a leer todas las direcciones para verificar los nuevos valores escritos. Durante todo el proceso, se pueden observar las señales de reloj, dirección, lectura/escritura y datos en la ventana de simulación, permitiendo verificar el correcto funcionamiento de la memoria RAM, incluyendo las operaciones de lectura y escritura, así como el manejo del vector de validación VALID.



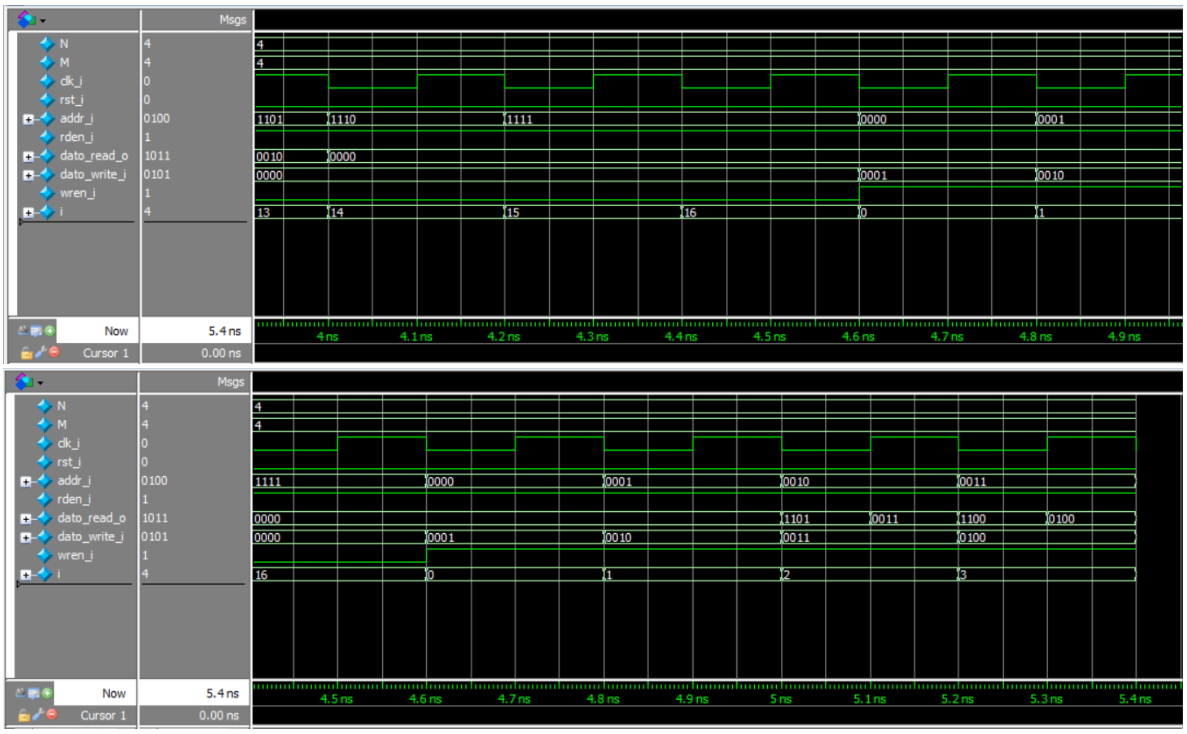


Figura 10 Simulaciónes

Implementación en hardware

Al implementar el diseño en la FPGA MAX V 5M240Z, se debió observar el funcionamiento práctico de la memoria RAM a través del display de 7 segmentos. Inicialmente, el display mostraría los valores hexadecimales preconfigurados en la memoria, comenzando con 'F' y descendiendo hasta '0', correspondientes a las 16 posiciones de memoria inicializadas. Esto permitiría verificar visualmente que la carga inicial de la memoria desde el archivo ram_init.hex se realizó correctamente.

Durante la operación, se podrían utilizar los interruptores o botones de la placa para controlar las operaciones de lectura y escritura. Al realizar una lectura, el display mostraría el valor almacenado en la dirección seleccionada. En el caso de una escritura, se podría ver cómo el nuevo valor se almacena y luego se muestra al leer esa misma dirección. El funcionamiento del divisor de frecuencia no sería directamente visible, pero su efecto se podría notar en la velocidad de actualización del display si se implementara un modo de escaneo automático de las direcciones de memoria.

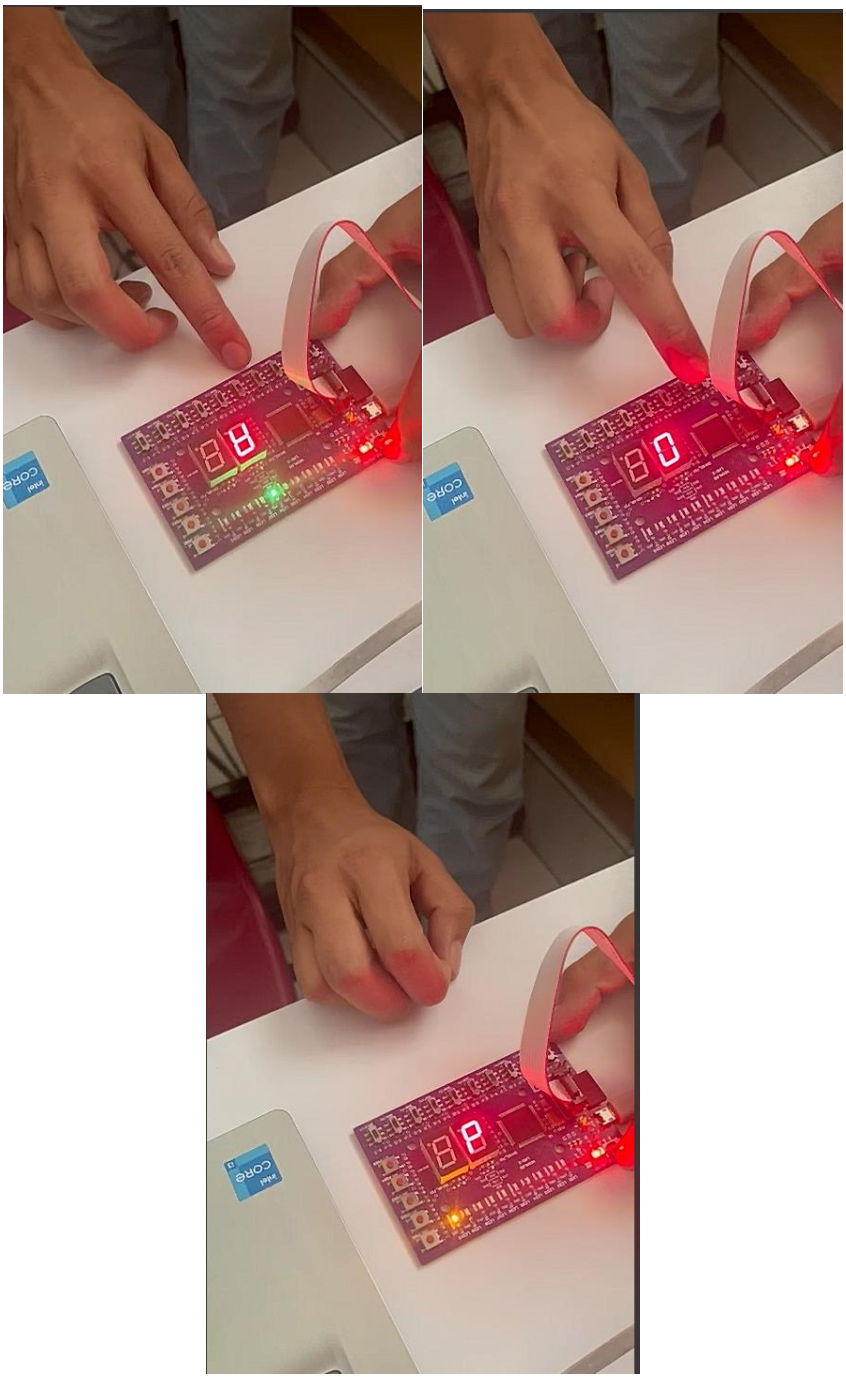


Figura 11 Implementación en hardware MAX V 5M240Z

Conclusiones

El proyecto de implementación de una Memoria RAM parametrizable en Verilog demuestra la versatilidad y potencia de los lenguajes de descripción de hardware para el diseño de sistemas digitales complejos. A través de la creación de un módulo de memoria con puertos separados para lectura y escritura, se logró una estructura flexible y fácilmente adaptable a diferentes tamaños y configuraciones. La inclusión de un vector de validación añade una capa adicional de control sobre los datos almacenados, mejorando la integridad y seguridad de la memoria. El proceso de simulación, que abarca desde la inicialización de la memoria hasta las operaciones de lectura y escritura, permitió verificar el correcto funcionamiento del diseño bajo diversas condiciones. Este proyecto no solo sirve como una valiosa herramienta educativa para comprender los principios de diseño de memorias digitales, sino que también proporciona una base sólida para futuras expansiones y aplicaciones en sistemas más complejos, destacando la importancia de la modularidad y parametrización en el diseño de hardware digital.

Referencias

[1] “Diseño de circuitos digitales y tecnología de computadores/Memorias”, Wikibooks.org. [En línea]. Disponible en: https://es.wikibooks.org/wiki/Dise%C3%B1o_de_circuitos_digitales_y_tecnolog%C3%ADa_de_computadores/Memorias. [Consultado: 27-jun-2024].