15 de Septiembre de 2020

Tecnológico de Monterrey

Memoria de instrucciones y memoria de datos

Laboratorio de arquitectura de Computadoras

ITESM Campus Monterrey

Profesor
Diego Fernando Valencia Martínez

Miguel Morales de la Vega
Javier Mondragón Martín del Campo

A00821541
A01365137

Introducción

La memoria es uno de los componentes más importantes en el funcionamiento de los equipos de computo. La memoria guarda la información que requiere el equipo de cómputo para llevar tareas como la configuración inicial, llamada a instrucciones, direcciones de registros así como datos que utilizan programas, aplicaciones y periféricos. Dentro de la práctica pondremos a prueba nuestros conocimientos en la programación de descripción de hardware para la creación de una memoria de datos y otra memoria la cual es usada para la alocación de instrucciones.
Se usará la memoria ROM la cual tiene la siguiente definición:
"La **memoria ROM** es el medio de almacenamiento de programas o datos que permiten el buen funcionamiento de los ordenadores o dispositivos electrónicos a través de la lectura de la información sin que pueda ser destruida o reprogramable."

Se usará la memoria RAM la cual tiene la siguiente definición:
"La memoria RAM es conocida como **memoria volátil** lo cual quiere decir que **los datos no se guardan de manera permanente**, es por ello, que cuando deja de existir una fuente de energía en el dispositivo la información se pierde."

La práctica requiere la creación de dos tipos de memoria no solo es su tipo de uso, sino en la forma en cómo se categorizan a nivel de hardware. La memoria RAM será usada para la alocación de datos y esta podrá ser escrita y leída múltiples veces. La memoria ROM será usada para la alocación de instrucciones y solo será usada para la lectura de su contenido.

Procedimiento

Memoria de instrucciones

Este módulo tiene como propósito el guardado de datos que podrían ser interpretados por el procesador como instrucciones a ejecutarse. Es importante mencionar que la memoria de instrucciones es normalmente una ROM (read only memory) esto quiere decir que su acceso es únicamente como lectura. La ROM en anteriores equipos de computo tenía el propósito de guardar la información del BIOS, esta nos permite la revisión de nuestro equipo de cómputo para así poder iniciar y tener todos sus componentes en una correcta conexión. Actualmente la BIOS es guardada en una RAM, ya que esta nos permite actualizar los parámetros y configuración de la BIOS.

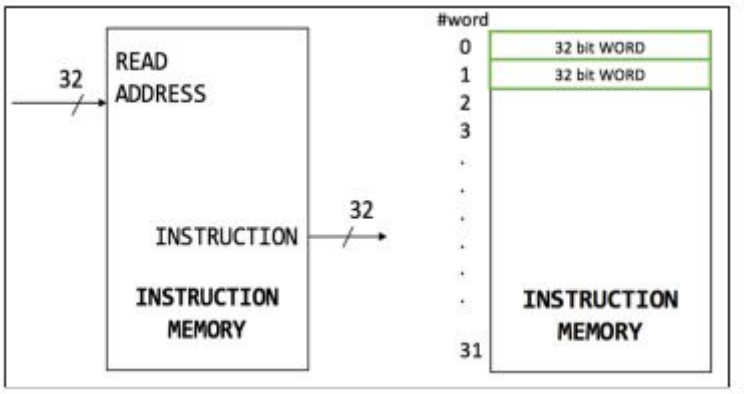

Figura 1.1. Memoria de instrucciones para 32 bits.

La memoria de instrucciones de la práctica debe contar con una entrada de 32 bits para apuntar a la dirección de memoria a la cual se quiere hacer la lectura. También debe contar con una salida de 32 bits, la cual corresponde a las instrucciones que están alocadas dentro de la memoria, esta descripción es observable en la Figura 1.1.
La construcción del módulo corresponde a un arreglo de 32 x 32, este arreglo hace referencia a 32 lugares de memoria y cada uno tiene un espacio de 32 bits. En términos de cómputo la memoria de instrucciones debe de estar compuesta por 32 espacios de memoria con la capacidad de almacenar palabras de 32 bits, esta construcción se muestra en la Figura 1.2.

```
type ROM_type is array (size-1 downto 0) of STD_LOGIC_VECTOR(word-1 downto 0);
        constant ROM : ROM_type := (
                x"00000001", x"00000005", x"00000009", x"0000000d", x"00000011", x"00000015", x"00000019", x"0000001d",
                x"00000002", x"00000006", x"0000000a", x"0000000e", x"00000012", x"00000016", x"0000001a", x"0000001e",
                x"00000003", x"00000007", x"0000000b", x"0000000f", x"00000013", x"00000017", x"0000001b", x"0000001f",
                x"00000004", x"00000008", x"0000000c", x"00000000", x"00000014", x"00000018", x"0000001c", x"00000010");
```

Figura 1.2. Arreglo de memoria de instrucciones.

En la Figura 1.2 podemos observar como la memoria es llenada con datos y estos no pueden ser modificados por medio de un pin de entrada. Para el acceso del arreglo se convierte el valor de 32 bits de *“READ_ADDRESS”* a un número entero el cual nos permite acceder a la posición del arreglo.

Memoria de datos

La memoria de datos es aquella que almacena los datos que produce el equipo de cómputo, sus módulos o los periféricos conectados. Un ejemplo de los datos que se almacenan en la memoria sería los datos de un documento de word, el programa en python que se desea compilar o bien los datos de la tarjeta de red.

Al ver estos ejemplos sabemos que un documento puede ser editado cientos de veces o bien los datos de la tarjeta de red son actualizados constantemente, esto convierte a nuestra memoria de datos en una RAM. La memoria RAM (Random Access Memory), puede ser escrita muchas veces así como leída, en nuestro equipo de cómputo las memorias RAM se encuentran como tarjetas verticales y su espacio de almacenamiento es mayor a la ROM, esto es por el volumen de datos y que casi todos los

componentes/periféricos tiene acceso a editar los datos que la RAM guarda en su interior y la ROM solo guarda información como la BIOS o los parámetros de componentes básicos para que el computará pueda encender.

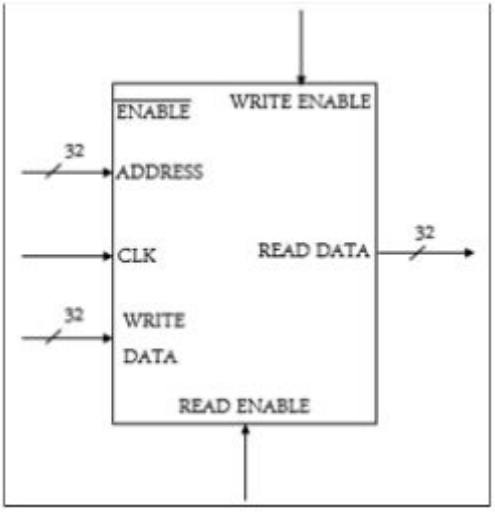

Figura 2.1. Memoria de datos.

Para la construcción de nuestro módulo se hará uso del siguiente listado de entradas y salidas.

Entradas	Salidas
ENABLE WRITE ENABLE READ ENABLE CLK WRITE DATA ADDRESS	READ DATA

La entrada de WRITE ENABLE y READ ENABLE, como lo dicen sus nombre nos servirán para indicar en qué estado funcionará la RAM. Si el pin de WRITE está encendido haremos una escritura de datos a la dirección deseada, si el pin de READ está encendido haremos una lectura a la dirección deseada. Es importante mencionar que ambos pines no pueden estar encendidos al mismo tiempo.

La entrada de WRITE ADDRESS nos permite escribir el dato en la dirección especificada por ADDRESS, el dato a guardar solo puede tener una extensión de 32 bits por espacio de memoria a guardar. La descripción del tamaño de las entradas y salidas así como sus nombre son indicados en la FIgura 2.1.

```
type ram_type is array(size-1 downto 0) of STD_LOGIC_VECTOR(word-1 downto 0);
signal RAM: ram_type;

begin

        process (ENABLE, CLK)
        begin
                if (ENABLE = '1' and falling_edge(CLK)) then
                        if (WRITE_ENABLE = '1') then
                                        RAM(to_integer(unsigned(ADDRESS))) <= WRITE_DATA;
                                        READ_DATA <= (others => '0');
                        elsif (READ_ENABLE = '1') then
                                READ_DATA <= RAM(to_integer(unsigned(ADDRESS)));
                        end if;
                end if;
        end process;
```

Figura 2.2. Funcionamiento de la RAM.

En el código de la Figura 2.2. podemos observar 2 componentes esenciales para la creación de la RAM.
El primer componente es un arreglo muy parecido al de la memoria ROM de 32 espacios de memoria y la capacidad de guardar una palabra de 32 bits en cada espacio, pero este arreglo se encuentra vacío, ya que la RAM se piensa para guardar datos y sobre escribirlos.
El segundo componente son los condicionales para el funcionamiento de la RAM donde este válida que enable es encendido para identificar la operación a realizar. Este funcionamiento también valida que la RAM esté energizada y su funcionamiento siga los pulsos de reloj.

Resultados

En la memoria ROM se almacenaron los siguientes números: x"00000001", x"00000005", x"00000009", x"0000000d", x"00000011", x"00000015", x"00000019", x"0000001d", x"00000002", x"00000006", x"0000000a", x"0000000e", x"00000012", x"00000016", x"0000001a", x"0000001e", x"00000003", x"00000007", x"0000000b", x"0000000f", x"00000013", x"00000017", x"0000001b", x"0000001f", x"00000004", x"00000008", x"0000000c", x"00000000", x"00000014", x"00000018", x"0000001c", x"00000010"
Considerando que VHDL almacena la última dirección (en este caso x"00000010") como la primera. Se intentó acceder a las siguientes direcciones: 0, 1, 2, 14 y 15. Los resultados son mostrados en la imagen 3.1 (los resultados de la salida instruction están en hexadecimal).

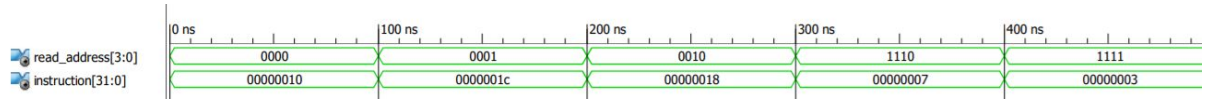

Imagen 3.1
Resultados de el ROM Testbench

Podemos observar que los resultados fueron acertados, las salidas fueron las mismas registradas.

En la memoria RAM se realizaron distintas operaciones registrando 3 números en 3 registros primero y después leerlos en el mismo orden en el que se escribieron, también se hicieron pruebas intentando registrar números a pesar que está en disabled y estos fueron los resultados los podemos observar en la imagen 3.2.

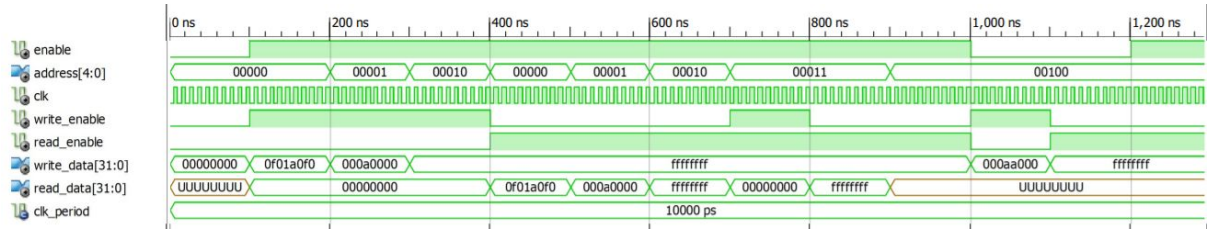

Imagen 3.2
Resultados de el RAM Testbench

Podemos observar que las operaciones fueron realizadas con el comportamiento adecuado de una RAM y esta registra los datos cuando debe y se pueden acceder en cualquier momento siempre y cuando el Enable esté habilitado.

Conclusión

Como lo mencionamos al momento de la introducción, durante el desarrollo de la práctica observamos las diferentes memorias que existen en un equipo de cómputo así como el uso específico para cada una. En la investigación más a fondo sobre los usos de la ROM se vio que la BIOS, comúnmente encontrada en la ROM, se encuentra en una memoria tipo flash ya que esta es actualizada en ciertas ocasiones.

Al crear los módulos pudimos conectar algunos componentes previamente creados, esto nos permitió tener una vista más amplia de cómo funciona la arquitectura computacional para el cómputo de PC, laptops o servidores. Las descripciones de cada módulo son simples manteniendo un funcionamiento ordenado y el menor número de instrucciones para su correcta ejecución.

Reflexion Individual

Miguel Morales A00821541:

> Durante la práctica puede recapitular los conceptos vistos en Sistemas Digitales sobre el funcionamiento de la ROM en VHDL. La práctica tuvo una complejidad simple en cuanto entender el direccionamiento, así como el acceso al arreglo en cada una de las memoria. Esta práctica ha sido una de mis favoritas ya que la memoria de datos e instrucciones son unos de los componentes más importantes y el como el almacenamiento de datos ha cambiado en el transcurso de los años.

Javier Mondragon A01365137:

Durante esta práctica pudimos observar que aquí si se puede utilizar un process ya que VHDL lo transforma a una memoria RAM en el esquematico por medio de la sintaxis y no se ejecuta de manera secuencial ni nada por el estilo, es importante tener la referencia de VHDL para porder obtener este codigo y entender como funciona VHDL y de qué maneras se puede utilizar para generar codigo que se traduzca a hardware. La ROM es muy sencilla y vemos lo importante de guardar los códigos realizados en cursos anteriores ya que este ya se había realizado en Sistemas Digitales Avanzados, con pocas modificaciones se puede llegar a cumplir con requisitos de otras prácticas.

Referencias

- https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/vhdl/vhd-single-port-rom.html
- https://www.estadofinito.com/rom/
- https://vhdl.es/memoria-ram-en-fpga/
- https://www.significados.com/memoria-ram/
- https://www.significados.com/memoria-rom/