CENTRO REGIONAL
UNIVERSITARIO
CÓRDOBA IUA

Facultad de Ingeniería

Centro Regional Universitario Córdoba - IUA

Arquitectura de Computadoras I

informe técnico

Trabajo Práctico 1

Memoria de Programa (ROM)

Autores:
Antún, Lucas
Fernández, Federico
Gauna, Gabriel
Mollani Gambarotta, Guido
Quinteros, Marcos

Profesor/a:
Ing. Leonardo Gabriel Gamboa

27 de agosto de 2025

Índice

1. Objetivos

El presente trabajo práctico tuvo como objetivos principales:

- Diseñar una memoria ROM de 256 × 8 bits que almacene instrucciones de programa.
- Implementar la carga de un programa de instrucciones hardcodeado en la memoria ROM.
- Realizar simulaciones de accesos de lectura para validar el contenido almacenado en la memoria.
- Analizar el comportamiento y las características de las memorias ROM en sistemas digitales.

2. Marco Teórico

2.1. Memoria de Programa en Sistemas Digitales

En los sistemas digitales y procesadores, la Memoria de Programa constituye el componente encargado de almacenar las instrucciones que fueron ejecutadas por el procesador. Estas instrucciones forman el software básico que define el comportamiento del sistema. El acceso a estas instrucciones es usualmente de solo lectura (ROM - Read Only Memory), lo que significa que el contenido está predefinido y no puede ser modificado durante la ejecución normal, garantizando una operación confiable y estable.

2.2. Memoria ROM (Read Only Memory)

La ROM es un dispositivo de almacenamiento digital no volátil que contiene datos permanentes, en este caso, instrucciones del programa. Las características principales de la ROM incluyen:

- El contenido se carga durante la fabricación o al inicializar el dispositivo (hardcodeo).
- No permite escritura durante la operación normal (solo lectura).
- Permite accesos rápidos y directos mediante la selección de direcciones.
- Es ideal para almacenar programas fijos y código de arranque.

2.3. Arquitectura de una ROM 256 × 8 bits

La especificación 256 × 8 bits indica que la memoria puede almacenar 256 palabras de 8 bits cada una. Esto significa:

- 256 posiciones de memoria direccionables con un bus de direcciones de 8 bits ($2^8 = 256$)

- Cada posición almacena 8 bits de datos
- Capacidad total de almacenamiento: $256 \times 8 = 2048$ bits

La estructura básica se representa mediante la ecuación:

$$\text{Dirección} = A[7:0] \rightarrow \text{Datos} = D[7:0] \tag{1}$$

donde $A[7:0]$ representa la entrada de dirección de 8 bits y $D[7:0]$ la salida de datos de 8 bits.

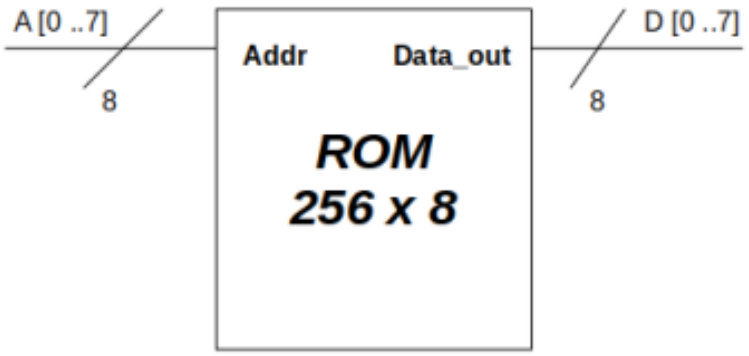

Figura 1: Circuito integrado (chip) de una ROM.

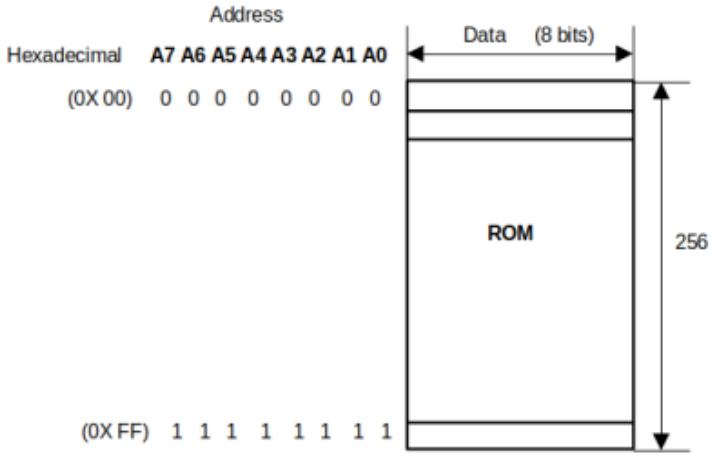

Figura 2: Mapa de memoria de una ROM 256×8 bits

3. Instrumentos Utilizados

Para la realización del presente trabajo práctico fueron utilizadas las siguientes herramientas:

3.1. Software

- Icarus Verilog para simulación
- GTKWave para visualización de formas de onda
- Visual Studio Code para desarrollo de código Verilog

4. Procedimiento Experimental

4.1. Diseño del Módulo ROM

Se diseñó un módulo ROM en lenguaje Verilog HDL con las siguientes especificaciones:

- Entrada de dirección: `addr[7:0]` (8 bits)
- Salida de datos: `data_out[7:0]` (8 bits)
- Memoria interna de 256 posiciones de 8 bits cada una

4.2. Implementación en Verilog

El código Verilog implementado fue el siguiente:

```
module rom(input wire [7:0] addr,output reg [7:0] data_out);

    reg [7:0] rom_mem [0:255];
    integer i;

    initial begin
        rom_mem[0] = 8'hAA;
        rom_mem[1] = 8'hFF;
        rom_mem[2] = 8'hF4;
        rom_mem[3] = 8'hB2;
        rom_mem[4] = 8'hE0;
        rom_mem[5] = 8'hD3;
        rom_mem[6] = 8'hF1;
        rom_mem[7] = 8'hB6;
        rom_mem[8] = 8'hA3;
        for (i = 9; i < 256; i = i + 1) begin
            rom_mem[i] = 8'h00;
        end
    end

```

```
    always @(*) begin
        data_out = rom_mem[addr];
    end

endmodule
```

Listing 1: Módulo ROM 256×8 bits

4.3. Desarrollo del Testbench

Se desarrolló un testbench para verificar el funcionamiento correcto de la ROM:

```
`timescale 1ns/1ps

`timescale 1ns/1ps
`include "rom.v"

module tb_rom;

    reg [7:0] addr;
    wire [7:0] data_out;

    rom rom_inst (
        .addr(addr),
        .data_out(data_out)
    );

    initial begin
        $dumpfile("rom.vcd");
        $dumpvars(0, tb_rom);

        addr = 0;
        #10;
        $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
            data_out);

        addr = 1;
        #10;
        $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
            data_out);

        addr = 2;
        #5;
        $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
            data_out);

        addr = 3;
        #5;
        $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
            data_out);
```

```
35
36          addr = 4;
37          #10;
38          $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
                data_out);
39
40          addr = 5;
41          #5;
42          $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
                data_out);
43
44          addr = 6;
45          #5;
46          $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
                data_out);
47
48          addr = 7;
49          #5;
50          $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
                data_out);
51
52          addr = 8;
53          #5;
54          $display("Tiempo:%0t|Direccion:%0d|Dato_leido:%h",$time,addr,
                data_out);
55
56          #10;
57          $finish;
58      end
59
60  endmodule
```

Listing 2: Testbench para ROM 256×8 bits

5. Resultados

5.1. Simulación

La simulación del diseño fue realizada exitosamente utilizando Icarus Verilog y visualizada en GTKWave. Los resultados obtenidos fueron:

Tiempo	Dirección	Dato leído
10000	0	aa
20000	1	ff
25000	2	f4
30000	3	b2
40000	4	e0
45000	5	d3
50000	6	f1
55000	7	b6
60000	8	a3

Tabla 1: Lecturas de memoria en distintos tiempos

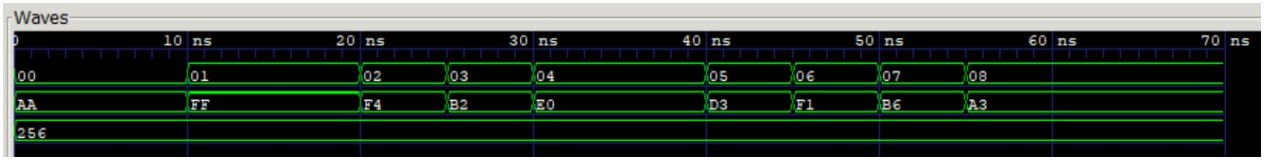

Figura 3: Simulación en GTKWave

6. Discusión y Conclusiones

6.1. Análisis de Resultados

Los resultados obtenidos demuestran que el diseño de la memoria ROM 256×8 bits funciona correctamente según las especificaciones. La simulación verificó que las instrucciones hardcodeadas pueden ser accedidas de manera precisa mediante el direccionamiento apropiado.

El tiempo de acceso es prácticamente instantáneo debido a la naturaleza asíncrona del diseño, lo que lo hace ideal para aplicaciones donde se requiere acceso rápido a las instrucciones del programa.

6.2. Respuestas a las Preguntas Planteadas

¿Por qué se utiliza una memoria de solo lectura para almacenar las instrucciones del programa?

Se utiliza una memoria ROM para almacenar instrucciones del programa por las siguientes razones:

- **Estabilidad**: El contenido no puede ser modificado accidentalmente durante la ejecución, garantizando la integridad del programa.

- **Confiabilidad**: No se ve afectada por fallos de alimentación, manteniendo el programa intacto.
- **Seguridad**: Previene la modificación maliciosa o accidental del código del programa.
- **Costo**: Es más económica que las memorias de lectura/escritura para aplicaciones donde no se requiere modificación del programa.
- **Velocidad**: Ofrece acceso rápido y predecible a las instrucciones.

¿Qué ventajas y limitaciones tiene el uso de un programa hardcodeado en la ROM?

Ventajas:

- Velocidad de acceso constante y predecible
- Inmunidad a errores de escritura accidental
- Menor consumo de energía comparado con memorias volátiles
- Costo reducido en producción masiva
- Alta confiabilidad y estabilidad del sistema

Limitaciones:

- Imposibilidad de actualización o modificación del programa sin reprogramar
- Falta de flexibilidad para adaptarse a diferentes aplicaciones dinámicamente
- Dificultad para corregir errores una vez sintetizada en hardware
- Tiempo de desarrollo más largo debido a la necesidad de validación exhaustiva
- Limitación en el tamaño del programa por la capacidad fija de la ROM

¿Qué tipos de errores o problemas podrían presentarse al codificar la ROM y cómo podrían detectarse mediante simulación?

Los errores más comunes incluyen:

- **Errores de inicialización**: Datos incorrectos en posiciones específicas. Se detectan comparando sistemáticamente los valores leídos con los esperados en el testbench.
- **Errores de direccionamiento**: Lecturas de posiciones incorrectas o desbordamiento de direcciones. Se identifican mediante pruebas exhaustivas de todas las direcciones válidas.
- **Errores de timing**: Problemas de sincronización en la lectura. Se detectan analizando las formas de onda y verificando que los datos se estabilicen correctamente.

- **Errores de codificación**: Instrucciones mal codificadas o formato incorrecto. Se previenen mediante verificación del conjunto de instrucciones y validación del testbench.
- **Errores de síntesis**: Diferencias entre simulación y hardware real. Se detectan mediante comparación entre resultados de simulación y pruebas en hardware.

¿En qué situaciones reales es útil emplear memorias ROM para almacenar el programa en sistemas digitales?

Las memorias ROM son especialmente útiles en:

- **Firmware de sistemas embebidos**: Microcontroladores en electrodomésticos, automóviles, dispositivos médicos donde se requiere código estable.
- **BIOS/UEFI**: Código de arranque en computadoras personales que debe ejecutarse inmediatamente al encender el sistema.
- **Sistemas de control industrial**: Controladores de procesos que requieren alta confiabilidad y operación en entornos críticos.
- **Dispositivos de comunicación**: Routers, switches y otros equipos de red que necesitan firmware estable.
- **Sistemas de seguridad**: Donde la integridad del código es crítica y no se puede permitir modificación externa.
- **Consolas de videojuegos**: Almacenamiento de juegos en cartuchos con acceso rápido.

¿Cómo verifica la simulación que las instrucciones almacenadas en la ROM se recuperan correctamente en función de la dirección?

La simulación verifica la correcta recuperación mediante:

- **Pruebas secuenciales**: Lectura sistemática de direcciones consecutivas para verificar que cada posición contiene el valor esperado.
- **Pruebas aleatorias**: Acceso a direcciones no secuenciales para verificar que el direccionamiento funciona correctamente en cualquier orden.
- **Comparación automática**: Uso de aserciones y monitores que comparan automáticamente valores leídos con los esperados.
- **Análisis temporal**: Verificación de que los datos se presentan en el tiempo correcto después del cambio de dirección.

Si tuviera que escalar la memoria para aumentar capacidad de almacenamiento ¿Qué criterios tendría en cuenta al momento de hacer el rediseño?

Los criterios principales para el rediseño serían:

- **Ancho del bus de direcciones**: Aumentar los bits de dirección para acceder a más posiciones (n bits = 2^n posiciones). Por ejemplo, para 1024 posiciones se necesitarían 10 bits de dirección.
- **Ancho de palabra**: Considerar si aumentar los bits por palabra para manejar instrucciones más complejas (16, 32 o 64 bits por instrucción).
- **Tiempo de acceso**: Considerar el impacto en la frecuencia máxima de operación del sistema al aumentar el tamaño.
- **Particionamiento**: Dividir la memoria en bancos o módulos para optimizar el acceso y permitir paralelismo.
- **Consumo de energía**: Evaluar el impacto energético de memorias más grandes.
- **Costo**: Balance entre capacidad de almacenamiento y recursos utilizados de la FPGA.

6.3. Conclusiones Finales

El trabajo práctico permitió comprender los fundamentos del diseño de memorias ROM en sistemas digitales. Se logró implementar exitosamente una memoria ROM de 256×8 bits con las siguientes características:

- Acceso asíncrono rápido a las instrucciones almacenadas
- Inicialización correcta con un programa de prueba hardcodeado
- Verificación exhaustiva mediante simulación

Los objetivos planteados fueron alcanzados satisfactoriamente, demostrando la viabilidad del diseño propuesto para aplicaciones de memoria de programa en sistemas embebidos. La experiencia adquirida en el uso de herramientas de simulación (Icarus Verilog, GTKWave) constituye una base sólida para futuros desarrollos en arquitectura de computadoras. El diseño implementado muestra las ventajas de utilizar memorias ROM para almacenar código de programa en sistemas donde la estabilidad y confiabilidad son prioritarias sobre la flexibilidad de modificación dinámica.

Referencias

[1] Patterson, D.A. & Hennessy, J.L. *Arquitectura de Computadores*. Morgan Kaufmann Publishers, 5ta Edición, 2013.

[2] Morris Mano. *Ingeniería computacional: diseño del hardware*. Prentice Hall, 1991.