Práctica 7. Modelado de Memorias

ROM

7.1. Introducción.

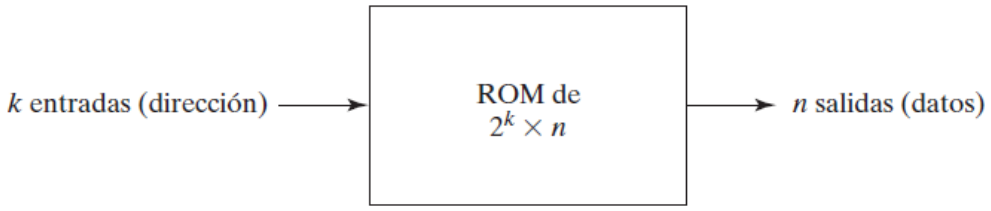
Una unidad de memoria es un dispositivo al que se transfiere información binaria que desea almacenarse y del que se puede obtener información que es necesario procesar. Cuando se efectúa procesamiento de datos, la información de la memoria se transfiere a registros selectos de la unidad de procesamiento. Los resultados intermedios y finales obtenidos en la unidad de procesamiento se transfieren de vuelta a la memoria para guardarse. La información binaria recibida de un dispositivo de entrada se almacena en la memoria, y la información transferida a un dispositivo de salida se toma de la memoria. Una unidad de memoria es una colección de celdas que permite almacenar una gran cantidad de información binaria.

Hay dos tipos de memorias que se usan en los sistemas digitales: *memoria de acceso aleatorio*

(RAM, *random-access memory*) y *memoria de sólo lectura* (ROM, *read-only memory*).

La memoria de sólo lectura es un *dispositivo lógico programable*. La información binaria que se almacena en un dispositivo lógico programable se especifica de alguna manera y luego se incorpora al hardware. Llamamos a este proceso *programar* el dispositivo. La palabra "programación" en este caso se refiere a un procedimiento de hardware que especifica los bits que se insertan en la configuración de hardware del dispositivo.

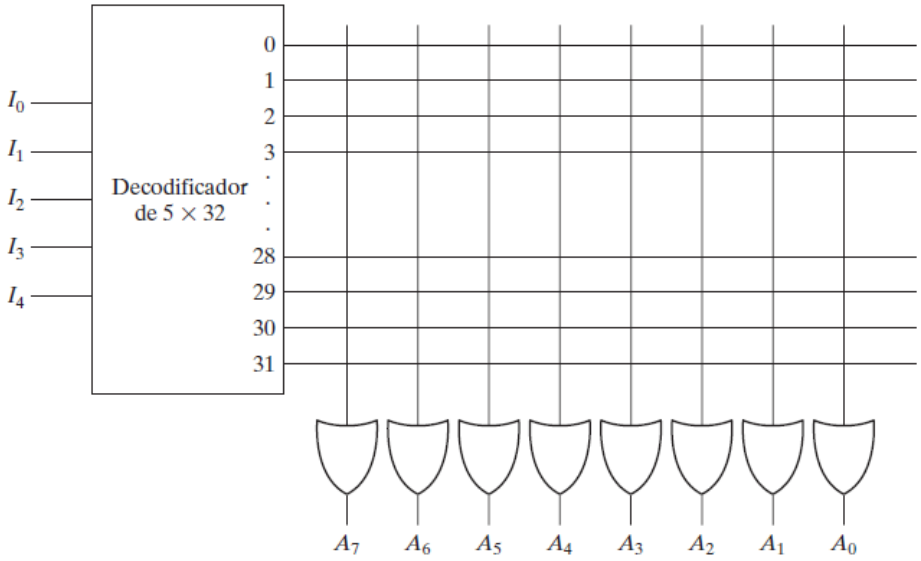
Una memoria de sólo lectura (ROM) es, en esencia, un dispositivo de memoria en el que se almacena información binaria permanente. El diseñador debe especificar la información, que entonces se incorpora a la unidad para formar el patrón de interconexión requerido. Una vez establecido el patrón, permanece en la unidad aunque se apague y se vuelva a encender.



En la figura anterior se reproduce un diagrama de bloques de una ROM. Tiene *k* entradas y *n* salidas.

Las entradas proporcionan la dirección de memoria y las salidas suministran los bits de datos de la palabra almacenada seleccionada por la dirección. El número de palabras de una ROM está determinado por el hecho de que se necesitan *k* líneas de dirección para especificar 2*k* palabras. La ROM no tiene entradas de datos porque no efectúa la operación de escritura.

Consideremos, por ejemplo, una ROM de 32*8. La unidad consiste en 32 palabras de 8 bits cada una. Hay cinco líneas de entrada que forman los números binarios del 0 al 31 para la dirección. La figura siguiente muestra la construcción lógica interna de la ROM. Las cinco entradas se decodifican a 32 salidas distintas con un decodificador de 5*32. Cada salida del decodificador representa una dirección de memoria. Las 32 salidas del decodificador se conectan a cada una de las ocho compuertas OR. El diagrama muestra la convención de arreglos lógicos que se emplea en circuitos complejos. Debe considerarse que cada compuerta OR tiene 32 entradas. Cada salida del decodificador se conecta a una de las entradas de cada compuerta OR. Puesto que cada compuerta OR tiene 32 conexiones de entrada y hay ocho compuertas OR, la ROM contiene 32*8=256 conexiones internas. En general, una ROM de 2k*n tiene un decodificador interno de k*2k y n compuertas OR. Cada compuerta OR tiene 2k entradas, que se conectan a cada una de las salidas del decodificador.



Las 256 interconexiones de la figura anterior son programables. Una conexión programable entre dos líneas equivale lógicamente a un interruptor que se puede alterar de modo que esté cerrado (o sea, que las dos líneas están conectadas) o abierto (o sea, que las dos líneas están desconectadas). La intersección programable entre dos líneas se conoce como punto de cruce.

Se usan diversos dispositivos físicos para implementar interruptores de punto de cruce. Una de las tecnologías más sencillas utiliza un fusible que normalmente conecta los dos puntos, pero que se abre o "quema" con un pulso de alto voltaje.

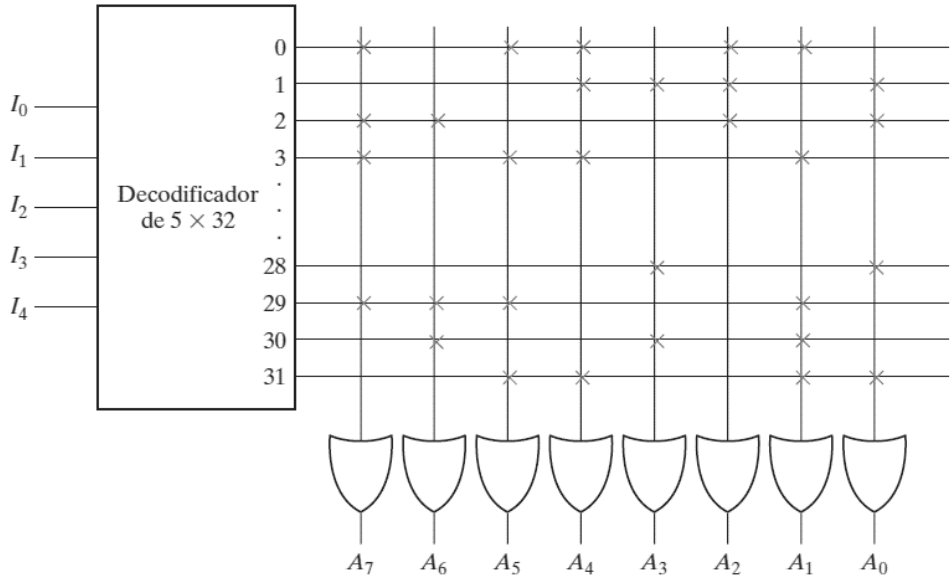
El almacenamiento binario interno de una ROM se especifica con una tabla de verdad que indica el contenido de palabra de cada dirección. Por ejemplo, el contenido de una ROM de

32*8 se especifica con una tabla de verdad similar a la que se observa en la siguiente tabla. Esa tabla presenta las cinco entradas bajo las que se da una lista de las 32 direcciones. En cada dirección, está almacenada una palabra de ocho bits, que se da bajo las columnas de salida. La tabla sólo muestra las primeras cuatro y las últimas cuatro palabras de la ROM. La tabla completa debe incluir la lista de las 32 palabras.

Tabla de verdad de ROM

Entradas					Salidas							
I4	I3	I2	I1	I0	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮					⋮					
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

El procedimiento en hardware que programa la ROM hace que se quemen fusibles según una tabla de verdad dada. Por ejemplo, la programación de la ROM según la tabla de verdad de la tabla produce la configuración que se aprecia en la siguiente figura. Cada 0 de la tabla de verdad especifica una ausencia de conexión, y cada 1, especifica una trayectoria que se obtiene con una conexión. Por ejemplo, la tabla especifica la palabra de ocho bits 10110010 que se almacenará permanentemente en la dirección 3. Los cuatro ceros de la palabra se programan quemando los fusibles entre la salida 3 del decodificador y las entradas de las compuertas OR asociadas a las salidas *A*6, *A*3, *A*2 y *A*0. Los cuatro unos de la palabra se han marcado en el diagrama con X para denotar una conexión, en lugar del punto que se usa para indicar una conexión permanente en los diagramas lógicos. Cuando la entrada de la ROM es 00011, todas las salidas del decodificador son 0 excepto la salida 3, que es 1 lógico. La señal equivalente a 1 lógico en la salida 3 del decodificador se propaga por las conexiones hasta las salidas de compuerta OR *A*7, *A*5, *A*4 y *A*1. Las otras cuatro salidas siguen en 0. El resultado es que la palabra almacenada 10110010 se aplica a las ocho salidas de datos.



7.2. Previo Práctica 1.

1) Defina el principio de funcionamiento de una memoria de solo lectura.

2) ¿Qué son los **array** en programación?

3) ¿Por qué el Flip – Flop tipo D es el mas utilizado para el diseño de memorias?

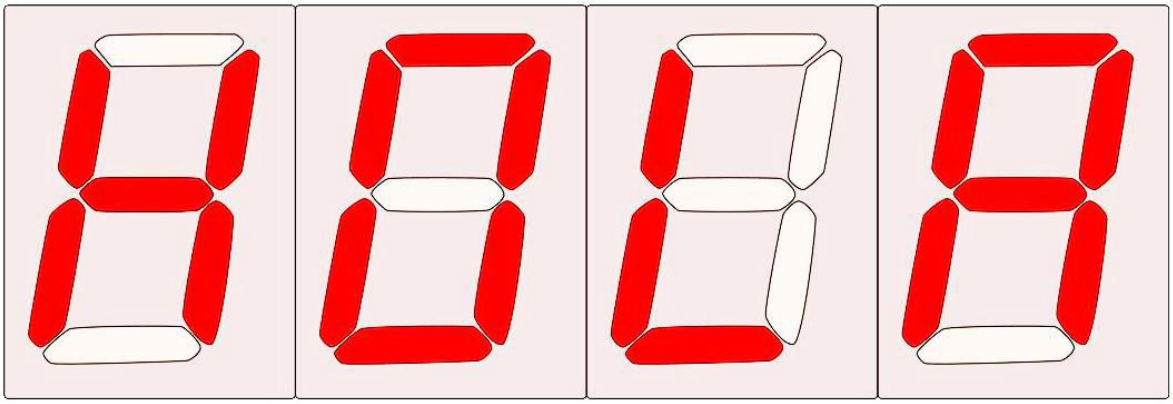
7.3. Ejemplo de implementación de un módulo de memoria

```
module memoria (clk,salida);
   input clk;
   output [7:0] salida;

   reg [7:0] memory [3:0]; //guarda 4 datos de 8 bits cada uno

   reg [7:0] memory_Q;
   reg [7:0] memory_D;

   reg [1:0] contador_D; //para contar los 4 datos
   reg [1:0] contador_Q;

   assign salida = memory_Q;

   initial
   begin
         $readmemh("datos.txt",memory); //almacenar datos de un archivo
   end

   always @ (posedge clk)
   begin
         memory_Q <= memory_D;
         contador_Q <= contador_D; //como cualquier contador D=Q
   end

   always @ (*)
   begin
         memory_D = memory[contador_Q];
         //sacamos la informacion de D, y asignandola a Q
         contador_D = contador_Q + 2'd1; // 2'b0000_0000
   end
endmodule
```

7.4. Trabajo de laboratorio

1) Realizar una unidad de memoria que cargue la cuenta de 0 – F para un display de 7 segmentos.

2) Realizar una unidad de memoria que cargue la palabra **HOLA** en **4** display de 7 segmentos y se mantenga fija en las salidas.



Display 1	Display 2	Display 3	Display 4

7.5. Conclusiones