

# Contadores, Shifters y Memorias

Curso de Diseño Lógico EL3307

Material basado en [1]

Docente: Ó. Caravaca M.

## Índice

<b>1. Contador</b>	<b>3</b>
1.1. Contador binario de N bits . . . . .	3
1.2. División de frecuencia . . . . .	3
1.3. Oscilador controlado digitalmente (DCO) . . . . .	4
1.4. Resumen . . . . .	4
<b>2. Registros de Desplazamiento y Cadenas de Escaneo</b>	<b>5</b>
2.1. Introducción . . . . .	5
2.2. Estructura básica . . . . .	5
2.3. Tipos de operación . . . . .	5
2.4. Cadenas de escaneo (Scan Chains) . . . . .	6
2.5. Flip-flop escaneable . . . . .	6
2.6. Ejemplo práctico . . . . .	7
2.7. Aplicaciones . . . . .	7
2.8. Preguntas de repaso . . . . .	7
2.9. Conclusión . . . . .	7
<b>3. Memorias</b>	<b>8</b>
3.1. Introducción . . . . .	8
3.2. Organización básica . . . . .	8
3.3. Tipos de acceso . . . . .	10
3.4. Tipos de memoria RAM . . . . .	10
3.5. Memorias no volátiles . . . . .	11
3.6. Señales y tiempos de operación . . . . .	12
3.7. Jerarquía de memoria . . . . .	12
3.8. Aplicaciones en diseño digital . . . . .	12
3.9. Resumen comparativo . . . . .	12
3.10. Ejemplo de memoria RAM en verilog . . . . .	12

<b>4. Conclusión</b>	<b>13</b>
4.1. Trabajo práctico: Diseño de Contador de Programa, Registro de Desplazamiento y Memoria RAM . . . . .	14
4.2. Objetivo general . . . . .	14
4.3. Fundamento teórico . . . . .	14
4.4. Parte 1. Contador de Programa (PC) de 4 bits — 30 puntos . . . . .	15
4.4.1. Descripción . . . . .	15
4.4.2. Entradas y salidas . . . . .	15
4.4.3. Actividades . . . . .	15
4.4.4. Investigación teórica . . . . .	15
4.5. Parte 2. Registro de Desplazamiento Aritmético (4 bits) — 25 puntos . . . . .	16
4.5.1. Descripción . . . . .	16
4.5.2. Entradas y salidas . . . . .	16
4.5.3. Actividades . . . . .	16
4.6. Parte 3. Memoria RAM de $16 \times 8$ bits con acceso mediante el PC — 35 puntos . . . . .	17
4.6.1. Descripción . . . . .	17
4.6.2. Entradas y salidas . . . . .	17
4.6.3. Actividades . . . . .	17
4.6.4. Ejemplo de tabla de simulación . . . . .	17
4.6.5. Evaluación final: Análisis y Reflexión — 10 puntos . . . . .	18

# 1. Contador

Un contador es un circuito secuencial que avanza automáticamente su valor binario con cada pulso de reloj. Se utiliza ampliamente en sistemas digitales para contar eventos, generar retardos, dividir frecuencias o secuenciar operaciones.

## 1.1. Contador binario de N bits

Un contador binario de N bits tiene una salida  $Q$  compuesta por N bits, que representan un número binario. Cada vez que llega un pulso de reloj, el contador incrementa su valor en 1, y cuando alcanza su valor máximo ( $2^N - 1$ ), vuelve a cero. Un reset o reinicio permite establecer el contador en cero en cualquier momento (ver Figuras 1-2).

El siguiente proceso ocurre en cada ciclo de reloj:

1. El circuito lee el valor actual almacenado en un registro.
2. Suma 1 a ese valor utilizando un sumador.
3. Guarda el nuevo valor nuevamente en el registro.

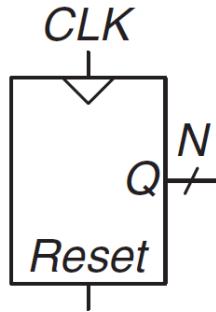


Figura 1: Símbolo del contador.

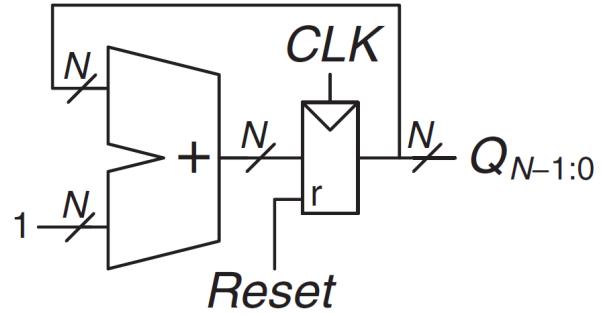


Figura 2: Contador N-bits.

## 1.2. División de frecuencia

El bit más significativo (MSB) del contador cambia de estado cada  $2^N$  ciclos, lo que significa que la frecuencia de ese bit es  $f_{clk}/2^N$ . Por ello, un contador puede funcionar como un divisor de frecuencia o divide-by- $2^N$ , útil para generar señales más lentas.

**Ejemplo:** Si se tiene un reloj de 50 MHz y un contador de 24 bits, la frecuencia de salida del bit más alto será:

$$f_{out} = \frac{50 \times 10^6}{2^{24}} \approx 2,98 \text{ Hz}$$

Este tipo de señal se puede usar, por ejemplo, para hacer parpadear un LED a una velocidad visible.

### 1.3. Oscilador controlado digitalmente (DCO)

Una extensión del contador binario es el oscilador controlado digitalmente (DCO). En lugar de sumar siempre 1 en cada ciclo, el contador suma un valor  $p$ . De esta manera, la frecuencia de salida depende de  $p$  según la relación:

$$f_{out} = f_{clk} \times \frac{p}{2^N}$$

Al ajustar  $p$ , es posible producir distintas frecuencias de salida con alta precisión. Sin embargo, esto requiere más hardware y un diseño más complejo.

### 1.4. Resumen

Los contadores son bloques fundamentales en el diseño lógico porque permiten:

- Contar ciclos o eventos.
- Dividir frecuencias de reloj.
- Generar señales periódicas.
- Sincronizar y secuenciar operaciones digitales.

Dominar su funcionamiento es esencial para diseñar sistemas digitales eficientes y comprender el comportamiento temporal de los circuitos secuenciales.

## 2. Registros de Desplazamiento y Cadenas de Escaneo

### 2.1. Introducción

Los *registros de desplazamiento* (Shift Registers) son circuitos secuenciales que permiten mover o desplazar bits a través de una serie de flip-flops. Son ampliamente usados para convertir datos entre formatos **serie** y **paralelo**, implementar retardos, almacenar información temporal o realizar operaciones de prueba en circuitos digitales.

### 2.2. Estructura básica

Un registro de desplazamiento está formado por  $N$  flip-flops conectados en serie. En cada flanco ascendente del reloj:

- Un nuevo bit entra por la entrada serial  $S_{in}$ .
- Cada flip-flop pasa su contenido al siguiente.
- El último bit desplazado aparece en la salida serial  $S_{out}$ .

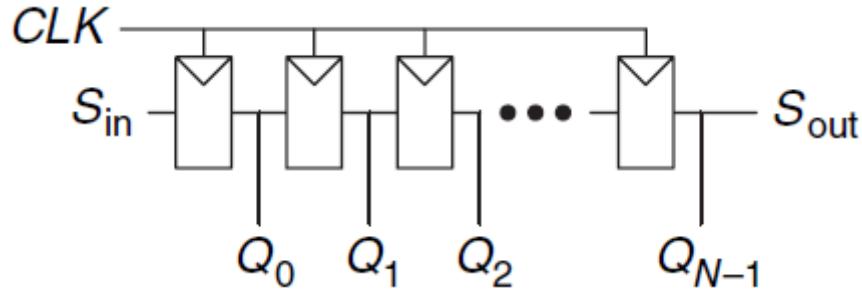


Figura 3: Registro de desplazamiento esquema.

Las salidas intermedias  $Q_0, Q_1, \dots, Q_{N-1}$  representan los valores almacenados en paralelo.

### 2.3. Tipos de operación

#### Serie a paralelo

El registro recibe los bits uno a uno (entrada serial) y, tras  $N$  ciclos de reloj, los pone disponibles en paralelo en las salidas  $Q_{N-1:0}$ .

#### Paralelo a serie

En este caso, el registro se carga con  $N$  bits en paralelo (entradas  $D_{N-1:0}$ ) y luego los envía uno a uno por la salida serial  $S_{out}$ . Este modo se activa con una señal de control Load:

- Load = 1: Carga los datos en paralelo.
- Load = 0: Desplaza los bits en serie.

```

SystemVerilog
module shiftreg #(parameter N = 8) (input logic clk,
    input logic reset, load,
    input logic sin,
    input logic [N-1:0] d,
    output logic [N-1:0] q,
    output logic sout);
    always_ff @(posedge clk, posedge reset)
        if (reset) q <= 0;
        else if (load) q <= d;
        else q <= {q[N-2:0], sin};
        assign sout = q [N-1];
endmodule

```

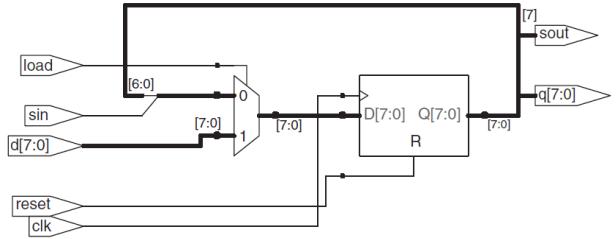


Figura 5: Shiftreg sintetizado.

Figura 4: Código en SystemVerilog de un registro de desplazamiento con carga paralela.

## 2.4. Cadenas de escaneo (Scan Chains)

En sistemas grandes, probar cada flip-flop de forma individual sería muy complejo. Por ello, los diseñadores usan **cadenas de escaneo**, donde todos los flip-flops se conectan en serie para formar un registro de desplazamiento. Así es posible:

- Cargar directamente un estado interno del circuito.
- Leer el contenido de todos los flip-flops.

**Modo normal:** los flip-flops operan con su entrada  $D$ . **Modo de prueba:** los flip-flops se conectan en cadena y desplazan datos con  $S_{in}$  y  $S_{out}$ .

## 2.5. Flip-flop escaneable

El flip-flop escaneable incluye un multiplexor que selecciona entre:

- La entrada de datos normal  $D$ .
- La entrada serial  $S_{in}$  usada en modo de prueba.

Cuando la señal de control Test está activa, el flip-flop se comporta como parte de una cadena de escaneo.

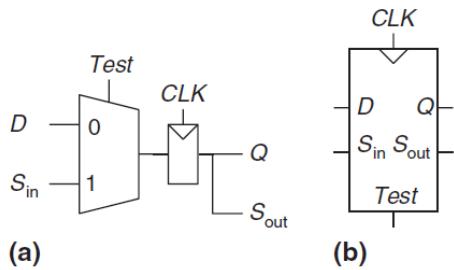


Figura 6: Scannable Flip flop: (a) Esquemático (b) Símbolo.

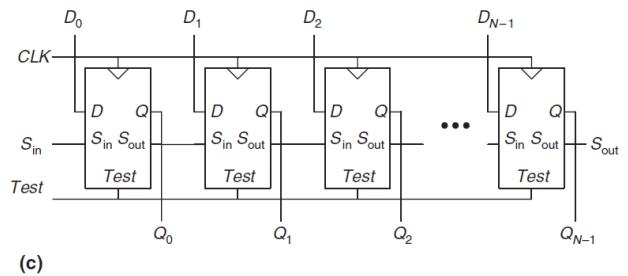


Figura 7: Scannable Flip flop: N-bit scannable register.

## **2.6. Ejemplo práctico**

Para verificar un contador de 32 bits:

1. En modo de prueba, se cargan los bits: 011111..,111.
2. Se ejecuta un ciclo de reloj en modo normal.
3. Se cambia nuevamente a modo de prueba y se leen los bits de salida.

El resultado esperado es 100000..,000, lo que confirma el funcionamiento del contador. Este método requiere solo  $32 + 1 + 32 = 65$  ciclos, en lugar de los más de dos mil millones de ciclos que implicaría una prueba convencional.

## **2.7. Aplicaciones**

- Conversión serie-paralelo y paralelo-serie.
- Retardos y temporización digital. Cada FF agrega un retardo que puede ser útil para sincronizar información con otros datos.
- Almacenamiento temporal de datos. Filtros y Buffers.
- Prueba y depuración de circuitos secuenciales.

## **2.8. Preguntas de repaso**

1. ¿Qué diferencia existe entre un registro de desplazamiento y un contador?
2. ¿Qué función cumple la señal Load?
3. ¿Qué ventajas ofrecen las cadenas de escaneo en el diseño digital?
4. ¿Por qué un flip-flop escaneable incluye un multiplexor?
5. Explica con tus palabras cómo probarías un contador usando una cadena de escaneo.

## **2.9. Conclusión**

Los registros de desplazamiento son bloques esenciales del diseño lógico. Su comprensión permite analizar y construir circuitos más complejos, además de entender cómo se prueban y diagnostican los sistemas digitales modernos.

## 3. Memorias

### 3.1. Introducción

Las memorias son circuitos digitales que almacenan información binaria y permiten recuperarla cuando se necesita. Cada posición de memoria tiene una *dirección* única y contiene una *palabra* de datos. Las memorias pueden clasificarse según su forma de acceso, su persistencia y su tecnología de implementación.

### 3.2. Organización básica

Una memoria está compuesta por:

- Celdas de almacenamiento: cada una guarda un bit.
- Líneas de dirección: seleccionan qué palabra se accede.
- Buses de datos: permiten leer o escribir información.
- Señales de control: indican si se desea leer o escribir.

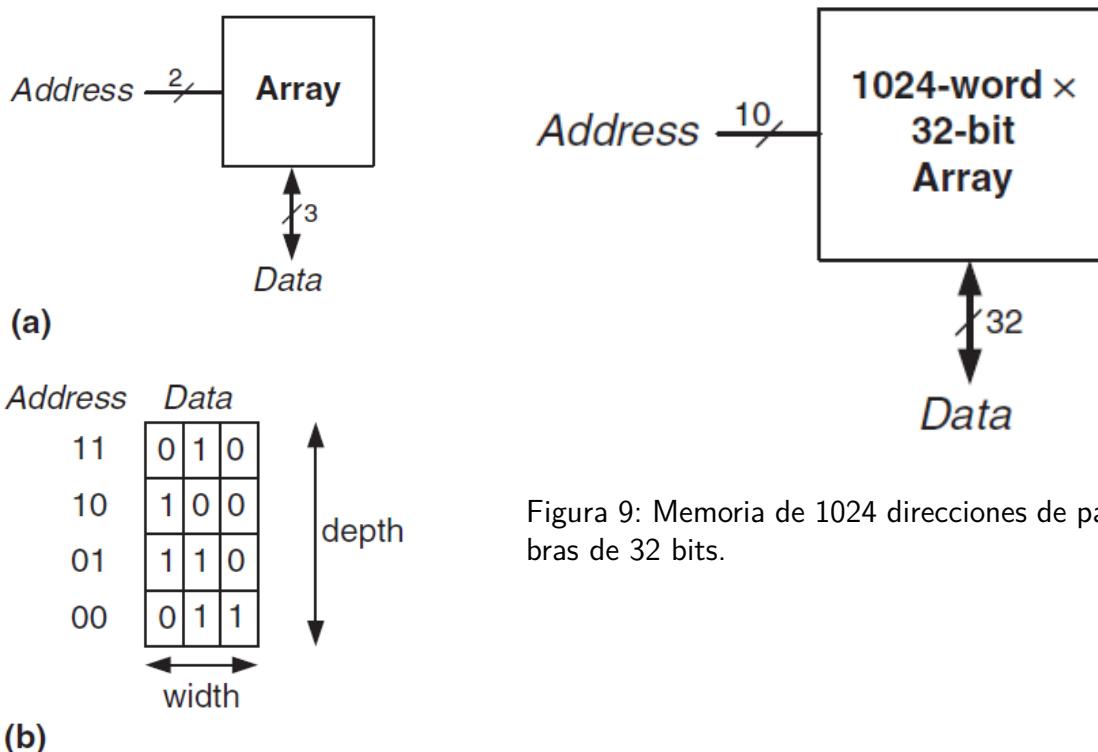


Figura 9: Memoria de 1024 direcciones de palabras de 32 bits.

Figura 8: Estructura básica de una memoria: dirección, matriz de celdas y bus de datos.

Una memoria de  $1024 \times 8$  tiene 1024 direcciones de 8 bits cada una, es decir, puede almacenar 1024 bytes.

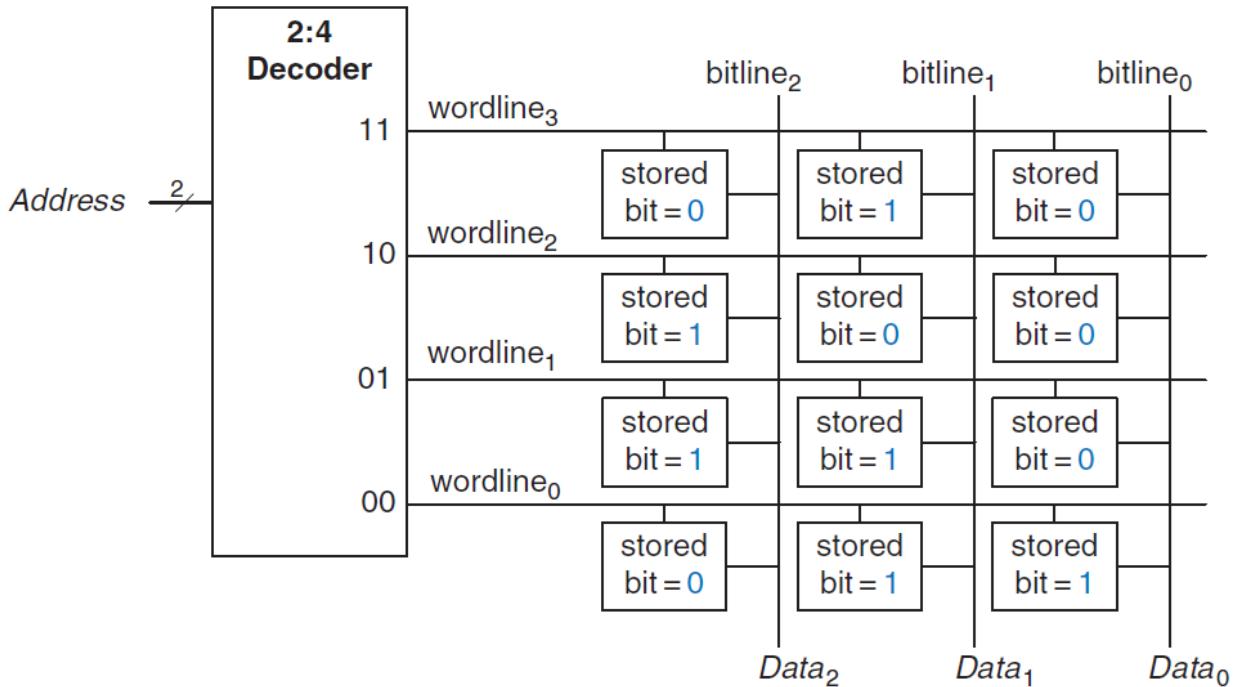


Figura 10: Organización interna de una matriz de memoria  $4 \times 3$ . La dirección activa un wordline que permite el almacenamiento o lectura de un bit en cada celda de memoria.

La Figura 10 muestra la organización interna de una matriz de memoria de  $4 \times 3$ . Aunque en la práctica las memorias reales son mucho más grandes, su funcionamiento puede extrapolarse a partir de este arreglo reducido. Durante una **lectura**, el decodificador activa una línea de palabra (**wordline**) específica, y las celdas de esa fila colocan sus valores (0 o 1) en las líneas de bits (**bitlines**). Durante una **escritura**, primero se fijan los niveles de las bitlines (HIGH o LOW) con los nuevos datos, y luego se activa la **wordline** correspondiente para almacenar dichos valores en las celdas seleccionadas.

Por ejemplo:

- Para leer la dirección 10, las bitlines se mantienen en estado flotante, el decodificador activa la **wordline<sub>2</sub>**, y los bits almacenados en esa fila (100) se leen en las líneas de datos.
- Para escribir el valor 001 en la dirección 11, se establecen las bitlines con el valor 001 y luego se activa la **wordline<sub>3</sub>**, almacenando el nuevo valor en las celdas correspondientes.

En resumen, las operaciones de lectura y escritura en la memoria se controlan mediante la activación coordinada de las **bitlines** y **wordlines**, determinando así qué celdas son leídas o modificadas dentro de la matriz.

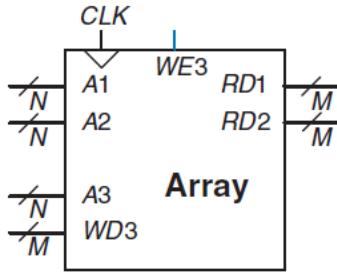


Figura 11: Memoria con más de un puerto de lectura-escritura.

Las memorias pueden tener uno o varios *puertos*, y cada puerto permite realizar operaciones de lectura y/o escritura sobre una dirección específica de memoria. Las memorias de un solo puerto sólo podían ejecutar una operación a la vez (lectura o escritura). En cambio, las **memorias multi-puerto** permiten acceder simultáneamente a varias direcciones, mejorando el rendimiento y la flexibilidad del sistema (Figura 11).

Por ejemplo, una memoria de tres puertos puede operar de la siguiente forma:

- **Puerto 1:** realiza una lectura desde la dirección  $A_1$  y coloca los datos leídos en la salida  $RD_1$ .
- **Puerto 2:** realiza otra lectura desde la dirección  $A_2$ , entregando los datos en la salida  $RD_2$ .
- **Puerto 3:** escribe el dato de entrada  $WD_3$  en la dirección  $A_3$  durante el flanco ascendente del reloj, siempre que la señal de escritura  $WE_3$  esté activa.

### 3.3. Tipos de acceso

- **RAM (Random Access Memory):** acceso aleatorio; cualquier posición se lee o escribe en el mismo tiempo.
- **ROM (Read-Only Memory):** sólo lectura; el contenido se graba una vez y no se puede modificar en operación normal.

### 3.4. Tipos de memoria RAM

#### SRAM (Static RAM)

Cada bit se almacena usando un biestable formado por seis transistores. No requiere refresco, lo que la hace rápida, aunque costosa y de baja densidad. Se usa en cachés, buffers o registros de alta velocidad.

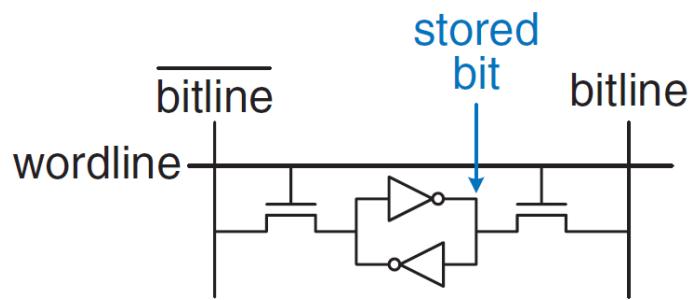


Figura 12: Celda básica de SRAM: dos inversores cruzados con transistores de acceso controlados por la línea de palabra.

### DRAM (Dynamic RAM)

Cada bit se almacena como una carga eléctrica en un capacitor. Requiere un ciclo de *refresco* periódico para mantener los datos. Es más densa y económica que la SRAM, pero más lenta. Es el tipo de memoria usada como memoria principal en la mayoría de los sistemas.

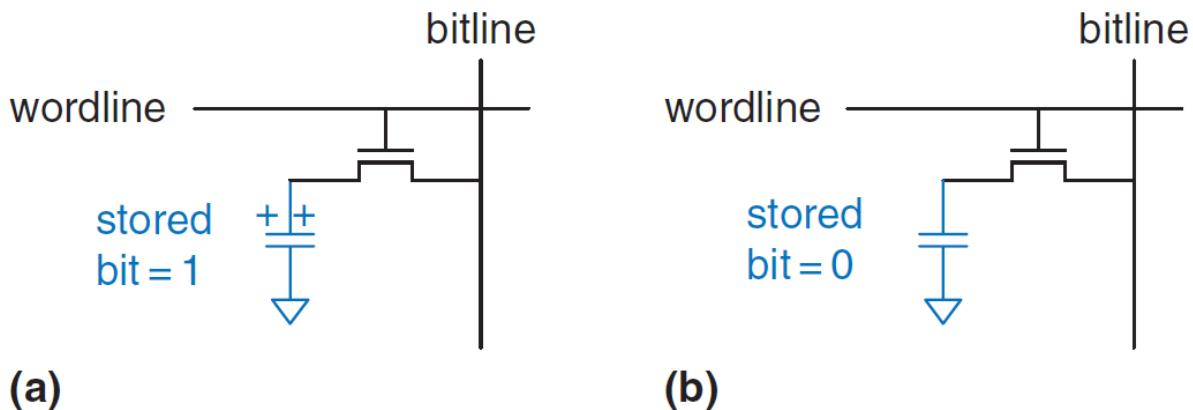


Figura 13: Celda de DRAM: un transistor y un capacitor almacenan cada bit.

### 3.5. Memorias no volátiles

- **ROM:** grabada en fábrica, contiene datos permanentes (firmware).
- **PROM:** programable una sola vez por el usuario.
- **EPROM:** puede borrarse con luz ultravioleta.
- **EEPROM y Flash:** pueden reescribirse eléctricamente. La memoria Flash es muy utilizada en memorias USB y almacenamiento embebido.

### **3.6. Señales y tiempos de operación**

Durante una lectura:

1. Se aplica una dirección.
2. Se activa la línea de lectura.
3. Los datos aparecen en el bus de salida tras el *tiempo de acceso*.

Durante una escritura:

1. Se aplica la dirección y el dato de entrada.
2. Se activa la señal de escritura.
3. Los datos se almacenan en la celda seleccionada.

### **3.7. Jerarquía de memoria**

Los sistemas modernos emplean una jerarquía de memoria para equilibrar velocidad, costo y capacidad:

Registros → Caché (SRAM) → Memoria principal (DRAM) → Almacenamiento secundario (Flash, disco, etc.)

### **3.8. Aplicaciones en diseño digital**

Las memorias se emplean para:

- Implementar tablas de verdad grandes (ROM).
- Almacenar programas o datos en sistemas embebidos.
- Crear retardos o buffers en comunicaciones digitales.
- Guardar configuraciones o constantes.
- Sincronizar y retener información temporal en procesadores.

### **3.9. Resumen comparativo**

Tipo	Ejemplo	Volátil	Reescribible	Velocidad
SRAM	Caché, registros	Sí	Sí	Muy alta
DRAM	Memoria principal	Sí	Sí	Alta
ROM	Firmware	No	No	Alta
Flash / EEPROM	BIOS, USB	No	Sí	Media

### **3.10. Ejemplo de memoria RAM en verilog**

```

module ram #(parameter N = 6, M = 32)
  (input logic clk,
   input logic we,
   input logic [N-1:0] adr,
   input logic [M-1:0] din,
   output logic [M-1:0] dout);

  logic [M-1:0] mem [2**N-1:0];
  always_ff @(posedge clk)
    if (we) mem [adr] <= din;
  assign dout = mem[adr];

endmodule

```

Figura 14: Enter Caption

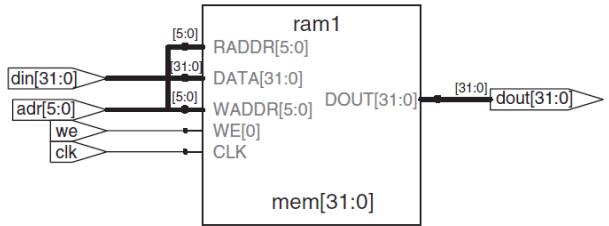


Figura 15: Enter Caption

## 4. Conclusión

Las memorias son componentes esenciales de todo sistema digital. Su comprensión permite diseñar arquitecturas eficientes equilibrando velocidad, costo, capacidad y persistencia de datos. Desde las LUTs en FPGAs hasta los sistemas de caché de los procesadores modernos, las memorias constituyen la base del almacenamiento y procesamiento de información digital.

## 4.1. Trabajo práctico: Diseño de Contador de Programa, Registro de Desplazamiento y Memoria RAM

**Aplicación: Algoritmo de Booth y Acceso Secuencial a Memoria**

**Puntaje total: 100 puntos**

### 4.2. Objetivo general

Diseñar, simular y analizar tres módulos secuenciales fundamentales en sistemas digitales:

1. Un contador de programa (Program Counter, PC) de 4 bits.
2. Un registro de desplazamiento aritmético (ASR) de 4 bits aplicado al algoritmo de Booth.
3. Una memoria RAM de  $16 \times 8$  bits, leída secuencialmente utilizando el contador de programa.

Además, se realizará una breve investigación teórica sobre el **control microprogramado** y el papel del contador de programa en dicho contexto.

### 4.3. Fundamento teórico

En los sistemas digitales secuenciales, el flujo de datos y el control dependen de circuitos que operan sincronizados con una señal de reloj:

- El **contador de programa (PC)** genera direcciones secuenciales, indicando la posición de la instrucción o dato a procesar.
- El **registro de desplazamiento** permite mover bits de forma controlada, y en el algoritmo de Booth se emplea para realizar multiplicaciones binarias con signo.
- La **memoria RAM** almacena temporalmente los datos o instrucciones y permite acceso de lectura y escritura mediante direcciones binarias.

Comprender la interacción entre estos bloques es esencial para el diseño de procesadores y sistemas embebidos.

## **4.4. Parte 1. Contador de Programa (PC) de 4 bits — 30 puntos**

### **4.4.1. Descripción**

El contador de programa (PC) es un registro secuencial que mantiene la dirección de la siguiente instrucción o palabra de datos que será accedida en un sistema digital. En este ejercicio, el PC tendrá 4 bits, lo que permite direccionar hasta  $2^4 = 16$  posiciones de memoria.

### **4.4.2. Entradas y salidas**

- Entradas: clk, rst, load, next\_pc[3:0].
- Salida: pc[3:0].

### **4.4.3. Actividades**

1. Diseñe un contador de programa de 4 bits con incremento, carga y reinicio. **(10 pts)**
2. Simule el comportamiento durante al menos 10 ciclos de reloj. **(10 pts)**
3. Verifique que al activar la señal load, el contador adopte el valor de next\_pc. **(5 pts)**
4. Redacte una breve explicación sobre la función del contador de programa en una arquitectura secuencial. **(5 pts)**

### **4.4.4. Investigación teórica**

Responda brevemente:

1. ¿Qué es el control microprogramado y cómo se diferencia del control cableado? **(3 pts)**
2. ¿Qué papel cumple el contador de programa dentro de una unidad de control microprogramada? **(3 pts)**
3. Ejemplo de cómo el contador recorre microinstrucciones almacenadas. **(4 pts)**

## **4.5. Parte 2. Registro de Desplazamiento Aritmético (4 bits) — 25 puntos**

### **4.5.1. Descripción**

El registro de desplazamiento aritmético se utiliza en el algoritmo de Booth para almacenar el multiplicador y un bit adicional  $Q_{-1}$ . Durante cada iteración, el registro realiza un desplazamiento aritmético a la derecha del conjunto  $(A, Q, Q_{-1})$ , conservando el bit de signo.

### **4.5.2. Entradas y salidas**

- Entradas: `clk`, `rst`, `load`, `data_in[3:0]`.
- Salidas:  $Q[3:0]$ ,  $Q_{-1}$ .

### **4.5.3. Actividades**

1. Diseñe un registro de desplazamiento aritmético de 4 bits. **(10 pts)**
2. Simule con valores positivos y negativos (por ejemplo, 0110 y 1010) y observe la conservación del bit de signo. **(7 pts)**
3. Describa el efecto del desplazamiento sobre  $(A, Q, Q_{-1})$ . **(4 pts)**
4. Explique la diferencia entre desplazamiento lógico y aritmético. **(4 pts)**

## **4.6. Parte 3. Memoria RAM de $16 \times 8$ bits con acceso mediante el PC — 35 puntos**

### **4.6.1. Descripción**

Se diseñará una memoria RAM de 16 direcciones (4 bits) y 8 bits por palabra, que será leída secuencialmente utilizando el contador de programa de 4 bits como dirección de acceso.

### **4.6.2. Entradas y salidas**

- Entradas: clk, we, addr[3:0], data\_in[7:0].
- Salida: data\_out[7:0].

### **4.6.3. Actividades**

1. Diseñe la memoria RAM de  $16 \times 8$  bits e inicialícela con valores de ejemplo. **(10 pts)**
2. Conecte la salida del PC (pc) a la entrada de dirección (addr) de la RAM y simule la lectura secuencial. **(10 pts)**
3. Documente los resultados en una tabla con los valores de pc, addr y data\_out por ciclo. **(5 pts)**
4. Analice qué ocurre si se activa la escritura (we = 1) en un ciclo determinado. **(5 pts)**
5. Explique el comportamiento cuando el PC alcanza su valor máximo (1111). **(5 pts)**

### **4.6.4. Ejemplo de tabla de simulación**

Ciclo	PC (4 bits)	Dirección usada	Dato leído (data_out)
1	0000	0000	01000001 (A)
2	0001	0001	01000010 (B)
3	0010	0010	01000011 (C)
4	0011	0011	01000100 (D)

#### **4.6.5. Evaluación final: Análisis y Reflexión — 10 puntos**

Responda en sus propias palabras:

1. ¿Cómo se relaciona el contador de programa con la dirección de lectura de la memoria RAM? **(3 pts)**
2. ¿Qué ventajas tiene utilizar un contador de programa para recorrer la memoria frente al direccionamiento manual? **(3 pts)**
3. ¿Qué relación existe entre el registro de desplazamiento y la memoria en un sistema secuencial? **(2 pts)**
4. ¿Cómo podría este sistema evolucionar hacia una unidad de control simple o microprogramada? **(2 pts)**

#### **Resumen de Evaluación**

<b>Parte</b>	<b>Puntos</b>
Parte 1: Contador de Programa	30
Parte 2: Registro de Desplazamiento	25
Parte 3: Memoria RAM con acceso secuencial	35
Evaluación final: Análisis y Reflexión	10
<b>Total</b>	<b>100 pts</b>

#### **Referencias**

- [1] Harris, D. M., & Harris, S. L. (2021). *Digital Design and Computer Architecture: RISC-V Edition*. Morgan Kaufmann, Cambridge, MA.
- [2] Wakerly, J. F. (2018). *Digital Design: Principles and Practices* (5th ed.). Pearson, New York.
- [3] Patel, D. (2023). *Booth Multiplication Algorithm*. Disponible en: <https://dhruvpatel004.github.io/Booth-Multiplication-Algorithm/#hero> (Consultado el 21 de octubre de 2025).