Tema 8 — Registros

Electrónica Digital · Grado en Ing. Electrónica, Robótica y Mecatrónica (GIERM) · Universidad de Sevilla

> Apuntes propios orientados a examen. Estilo *worked-example primero*: ves el resultado (cronograma / esquema) y luego la teoría que lo explica.

Índice

Idea central

> Un **registro** almacena **n bits** de información. Se construye con **n biestables** que comparten el reloj. Un registro de **desplazamiento** añade la capacidad de mover los bits de una posición a la siguiente en cada flanco de reloj.

La diferencia clave entre tipos de registro está en **cómo entran** y **cómo salen** los datos:

	Entrada	Salida
Paralelo	los n bits a la vez (un cable por bit)	los n bits a la vez
Serie	un bit por ciclo de reloj (un solo cable)	un bit por ciclo de reloj

0. Utilidades de dibujo

Definimos dos ayudantes que reutilizaremos en todo el notebook:

- `cronograma(...)` — dibuja un **diagrama de tiempos** (reloj + señales) con `matplotlib`.
- schemdraw para los **esquemas** de los registros.

Solo se necesitan `numpy`, `matplotlib` y `schemdraw`.

```
Utilidades cargadas.
```

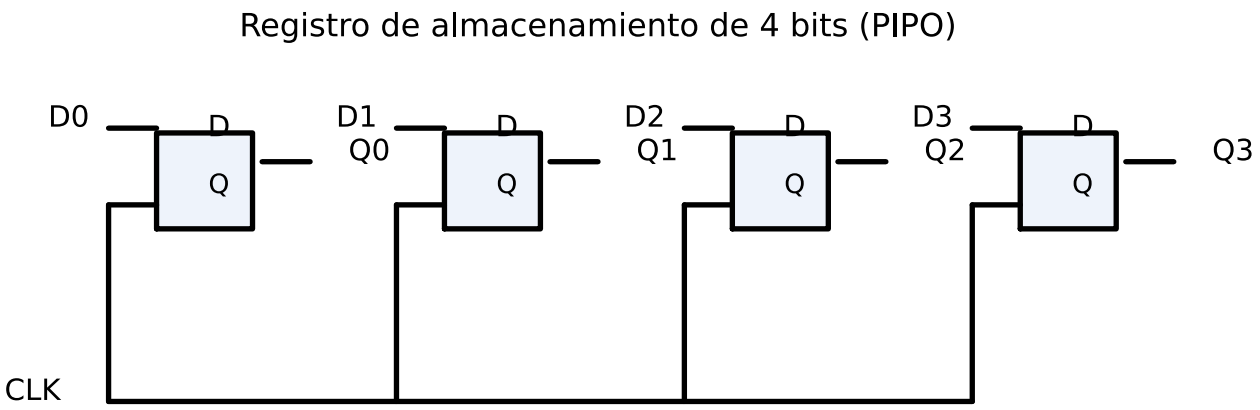
1. Registro básico de almacenamiento

- Almacena **n bits** y se **actualiza en cada flanco** de reloj.
- **Carga y salida en paralelo** (PIPO sin enable).
- Cada bit es un biestable **D**: en cada flanco .

$$n+1$$

Es simplemente un *latch* de palabra: lo que pongas en las entradas 0 $n-1$ aparece en 0 $n-1$ tras el siguiente flanco.

Para que los esquemas sean **robustos y autocontenidos** (la API de pines de `schemdraw` cambia entre versiones), dibujamos los biestables con una función propia basada en cajas (`Rect`).



2. Registro con habilitación de carga (*load enable*)

Igual que el anterior, pero **solo se actualiza si la señal de habilitación `E = 1`** en el flanco de reloj.

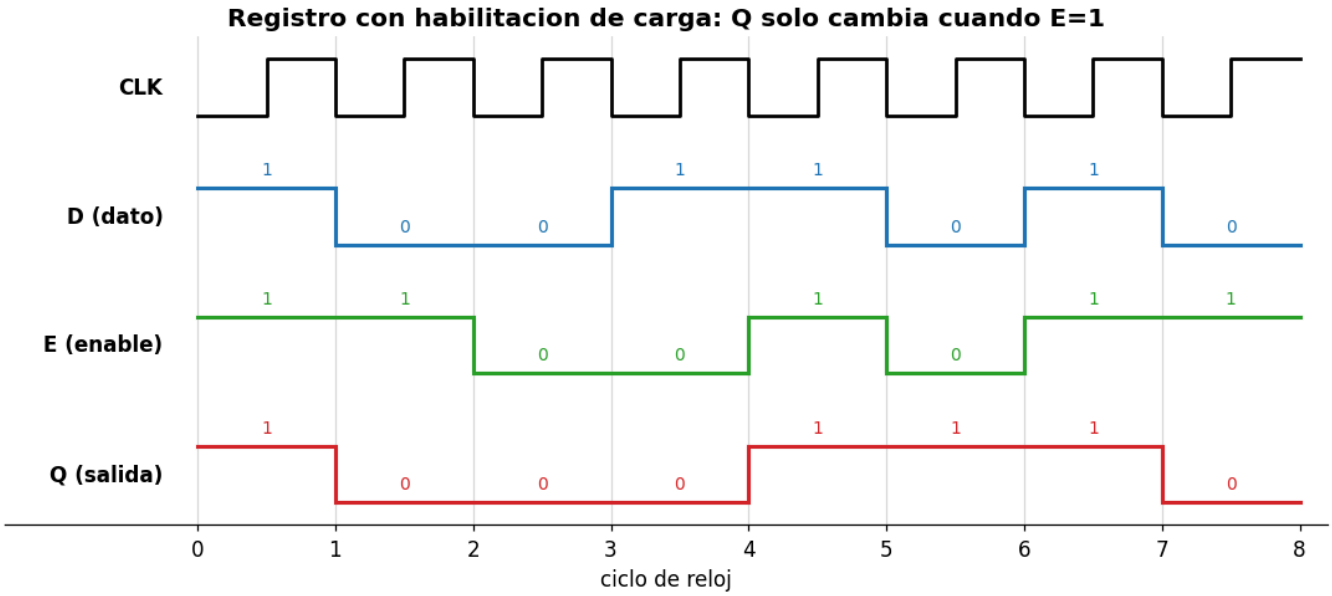
$$n+1$$

Se implementa poniendo a la entrada de cada biestable D un **multiplexor 2:1** controlado por `E`:

- `E=1` → pasa el dato externo `D`.
- `E=0` → realimenta la salida `Q` (mantiene el valor).

Cronograma del comportamiento del bit menos significativo:

```
D = [1, 0, 0, 1, 1, 0, 1, 0]
E = [1, 1, 0, 0, 1, 0, 1, 1]
Q = [1, 0, 0, 0, 1, 1, 1, 0]
```



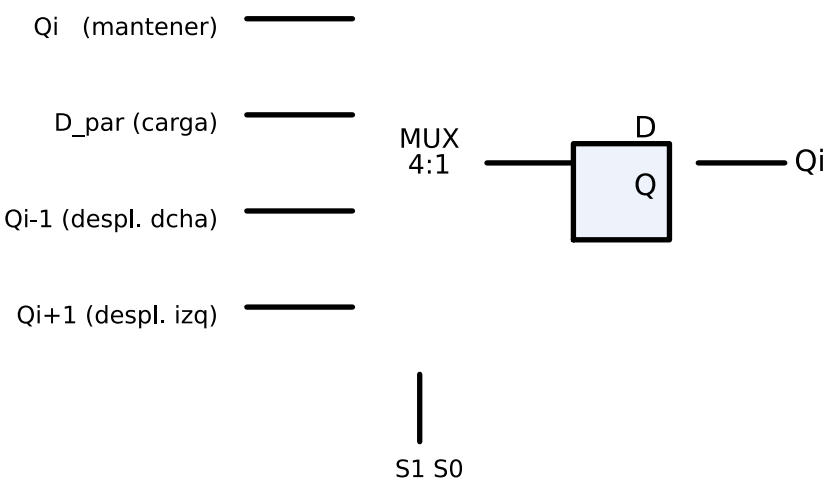
Observa: cuando `E=0` (ciclos 3 y 4, y ciclo 6) la salida **congela** su valor anterior aunque `D` cambie. Esa es exactamente la funcionalidad de "load" de un registro real.

3. Estructura general por celda

- Cada biestable tiene **varias entradas posibles** y unas **señales de control** que eligen cuál se aplica en cada caso.
- Normalmente se implementa con **multiplexores** delante de cada D, aunque a veces hay formas más sencillas.

Esta es la idea que unifica TODOS los registros: *almacenamiento, desplazamiento, carga paralela, bidireccional...* se diferencian solo en **qué se conecta a la entrada D de cada celda** mediante el MUX de control.

Celda i de un registro universal



4. Registros de desplazamiento

Un **registro de desplazamiento** mueve los bits una posición en cada flanco. Con biestables D en cadena ($_{i}$ $_{i-1}$), en cada flanco:

$$^{(t+1)}_{i} \quad ^{(t)}_{i-1}$$

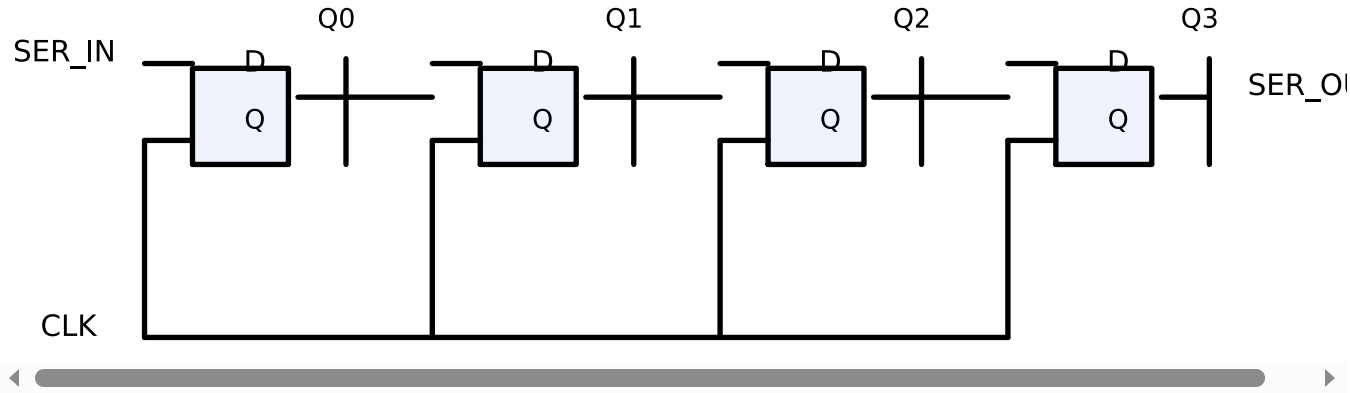
y por la primera celda entra el bit serie de entrada.

Hay **4 combinaciones** según entrada/salida (la nomenclatura clásica de examen):

Sigla	Entrada	Salida	Uso típico
SISO	Serie	Serie	línea de retardo, *delay* de n ciclos
SIPO	Serie	Paralelo	conversión **serie→paralelo** (recepción)
PISO	Paralelo	Serie	conversión **paralelo→serie** (transmisión)
PIPO	Paralelo	Paralelo	almacenamiento (el básico del apartado 1)

(*S=Serial In/Out, P=Parallel In/Out*)

Registro de desplazamiento a la derecha (D0->D1->D2->D3)



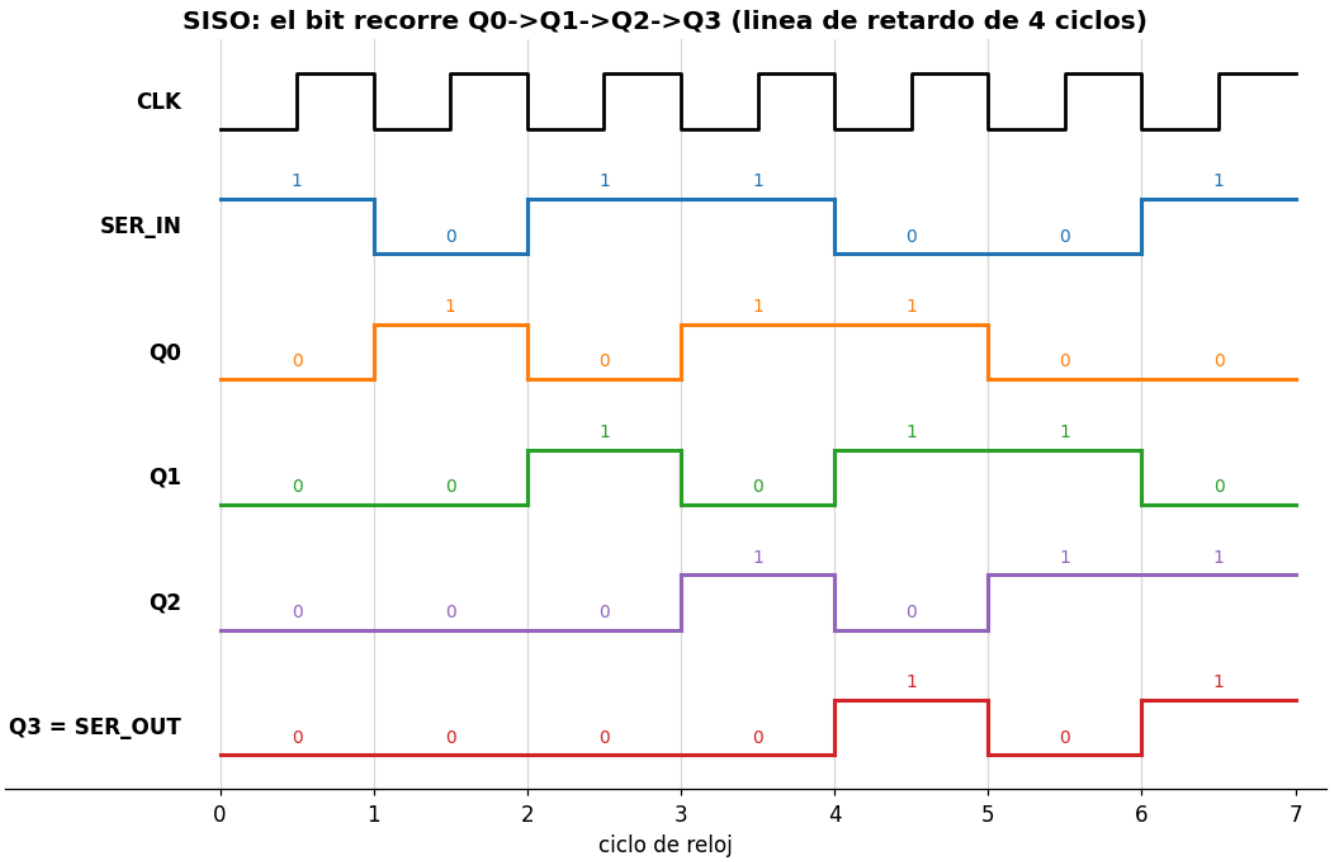
4.1 SISO — Serie In / Serie Out (worked example)

Es una **línea de retardo**: el bit que entra sale **n ciclos después** (n = nº de biestables).

Metemos la secuencia `1 0 1 1 0 0 1` por la entrada serie de un registro de **4 bits** inicializado a 0 y observamos la salida serie (= $_3$).

```
Entrada serie : [1, 0, 1, 1, 0, 0, 1]
Salida serie  : [0, 0, 0, 0, 1, 0, 1]   (retardo de 4 ciclos)

t  | SER_IN | Q0 Q1 Q2 Q3 | SER_OUT
0  |   1    | 0  0  0  0  |   0
1  |   0    | 1  0  0  0  |   0
2  |   1    | 0  1  0  0  |   0
3  |   1    | 1  0  1  0  |   0
4  |   0    | 1  1  0  1  |   1
5  |   0    | 0  1  1  0  |   0
6  |   1    | 0  0  1  1  |   1
```



Fíjate cómo el "1" del ciclo 0 va bajando una fila por cada ciclo de reloj: aparece en `Q0` en t=1, `Q1` en t=2... Eso es el **desplazamiento**.

4.2 SIPO — Serie In / Parallel Out

Mismo hardware que SISO, pero ahora **leemos las 4 salidas** $_0$ $_3$ **en paralelo**. Sirve para **recibir** un dato que llega bit a bit por un único cable y reconstruir la palabra completa.

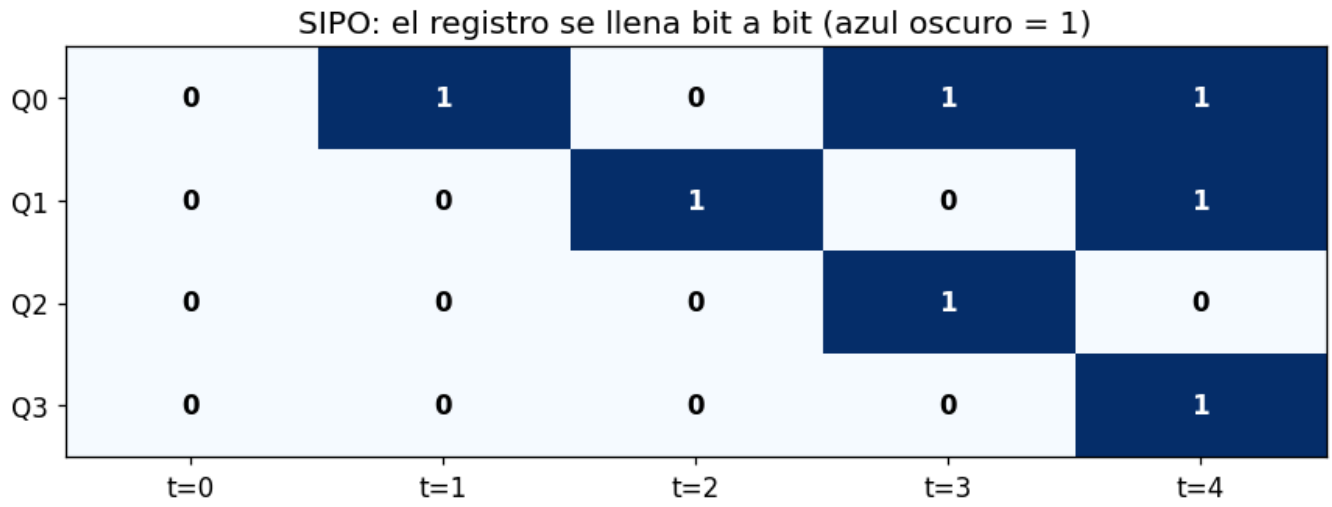
Es el principio del chip **74HC164** (registro SIPO de 8 bits).

Ejemplo: recibimos en serie el byte `1011` (MSB primero) y tras 4 ciclos lo tenemos completo en paralelo.

```
Llegada serie por ciclo: [1, 0, 1, 1]

t=0: registro (Q0..Q3) = [0, 0, 0, 0]
t=1: registro (Q0..Q3) = [1, 0, 0, 0]
t=2: registro (Q0..Q3) = [0, 1, 0, 0]
t=3: registro (Q0..Q3) = [1, 0, 1, 0]
t=4: registro (Q0..Q3) = [1, 1, 0, 1]

Tras 4 ciclos, salida paralela = [1, 1, 0, 1] <- palabra reconstruida
```



4.3 PISO — Parallel In / Serial Out

Carga en paralelo los n bits de golpe (con una señal `LOAD`) y luego los **saca en serie**, uno por ciclo. Es el inverso de SIPO: sirve para **transmitir** una palabra por un único cable.

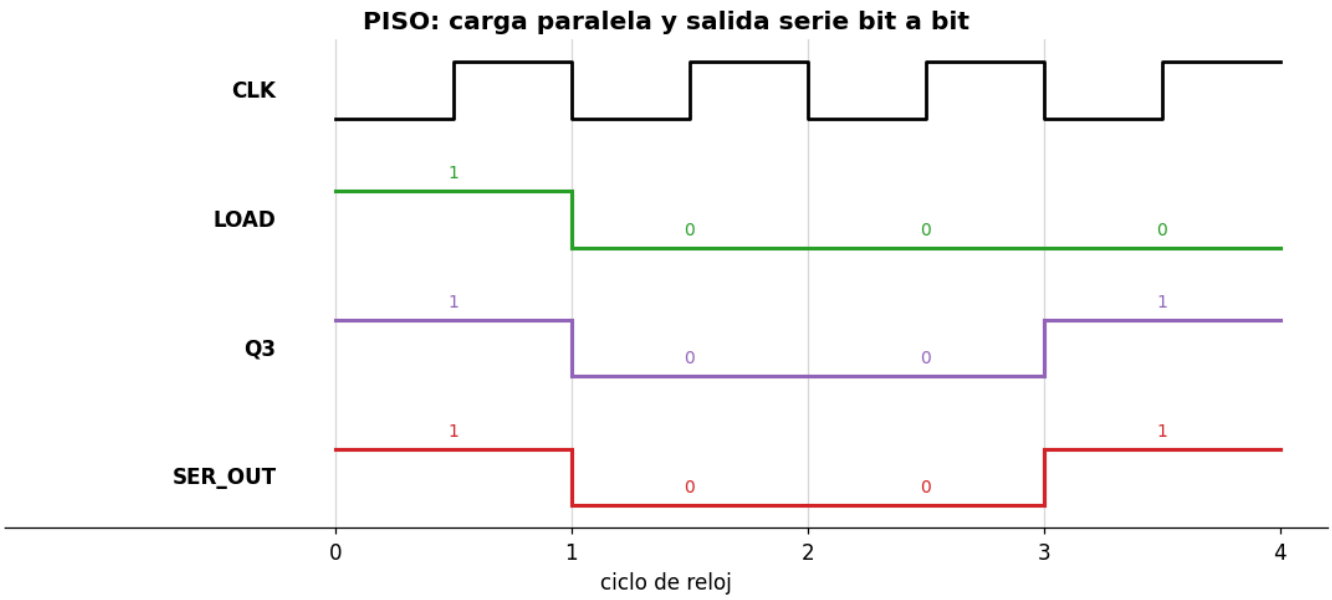
Es el principio del chip **74HC165**.

Funcionamiento:

1. `LOAD=1`: el registro se carga con `data_in` en paralelo.
2. `LOAD=0`: en cada flanco desplaza y va sacando un bit por `SER_OUT`.

```
Carga paralela (LOAD): [1, 0, 0, 1]
Salida serie         : [1, 0, 0, 1]

ciclo | Q0 Q1 Q2 Q3 | SER_OUT
  0   | 1  0  0  1  |   1
  1   | 0  1  0  0  |   0
  2   | 0  0  1  0  |   0
  3   | 0  0  0  1  |   1
```



4.4 PIPO — Parallel In / Parallel Out

Entra y sale todo en paralelo: es exactamente el **registro de almacenamiento** del apartado 1 (con o sin `enable`). No hay desplazamiento; cada celda guarda su bit. Ya lo vimos, así que no repetimos cronograma.

5. Desplazamiento bidireccional

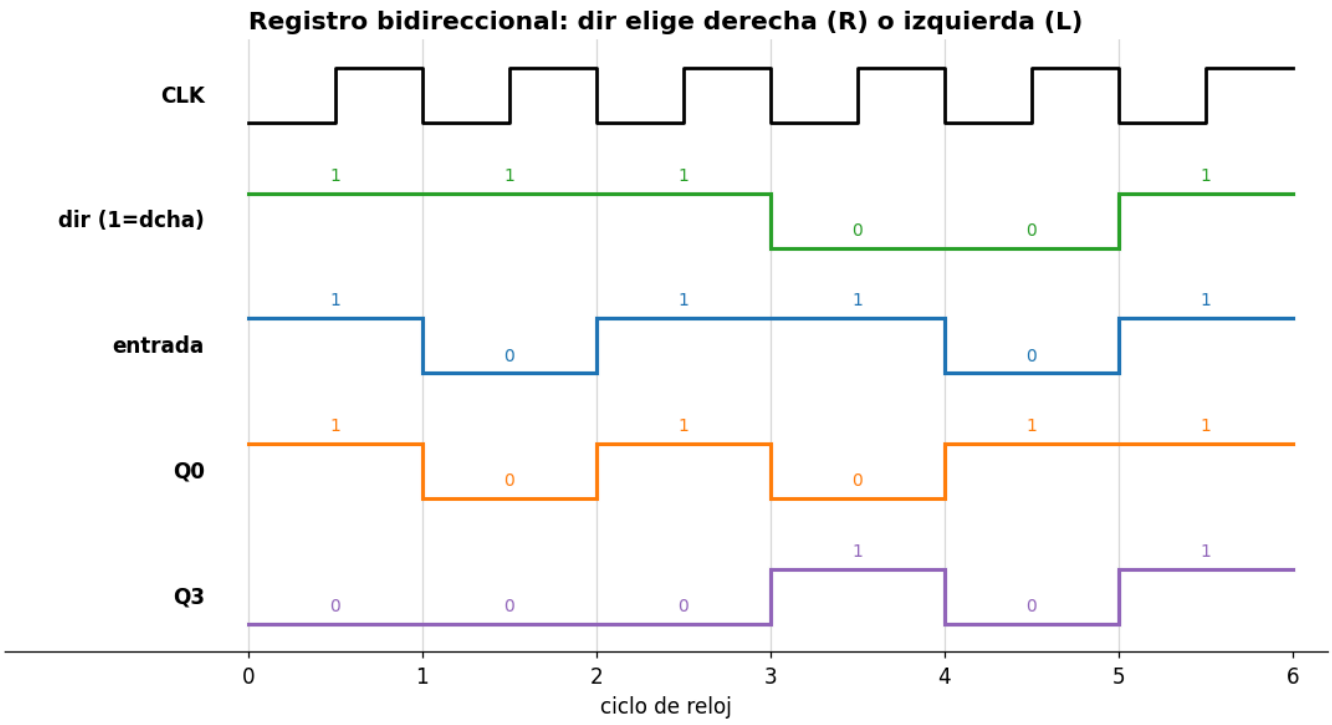
El registro puede desplazar a **derecha** o a **izquierda** según una señal de control `dir` (o `Modo`). La entrada de cada celda se elige con un MUX:

$$\overline{\text{dir}} \quad i \quad i-1 \qquad \text{dir} \quad i \quad i+1$$

- Desplazar **a la derecha**: $\overline{\text{dir}}$ i $i-1$ (entra por la izquierda, `sr_in`).
- Desplazar **a la izquierda**: dir i $i+1$ (entra por la derecha, `sr_in2`).

```
dir   : ['R', 'R', 'R', 'L', 'L', 'R']
entra : [1, 0, 1, 1, 0, 1]

t  | dir | Q0 Q1 Q2 Q3
0  |  R  | 1  0  0  0
1  |  R  | 0  1  0  0
2  |  R  | 1  0  1  0
3  |  L  | 0  1  0  1
4  |  L  | 1  0  1  0
5  |  R  | 1  1  0  1
```



6. Registro universal (74HC194)

Combina **todo** lo anterior. Tiene 2 bits de control `S1 S0` que seleccionan el **modo de operación** en cada flanco:

S1	S0	Operación
0	0	**Mantener** (no hace nada)
0	1	**Desplazar a la derecha** (entra `SR`)
1	0	**Desplazar a la izquierda** (entra `SL`)
1	1	**Carga paralela** (toma `data_in`)

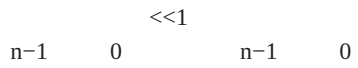
Es justo la "celda con MUX" del apartado 3 replicada n veces. El **74HC194** es un registro universal de 4 bits con estas 4 operaciones.

```
S1 S0 | operacion     | Q0 Q1 Q2 Q3
 1  1  | carga par.    | 1  0  1  0
 0  0  | mantener      | 1  0  1  0
 0  1  | despl. dcha   | 1  1  0  1
 1  0  | despl. izda   | 1  0  1  0
 0  1  | despl. dcha   | 0  1  0  1
```

7. Aplicaciones

7.1 Aritmética: multiplicar / dividir por 2

- **Desplazar a la izquierda** (con un 0 entrando por la derecha) = **multiplicar por 2** (sin signo).
- **Desplazar a la derecha** (replicando el bit de signo, *desplazamiento aritmético*) = **dividir por 2** (con signo).




```
x        = 00010011 = 19
x << 1   = 00100110 = 38   (x2)

y        = 11110110 = -10 (con signo)
y >>a 1  = 11111011 = -5   (/2 con signo, redondeo a -inf)
```

7.2 Conversión serie ↔ paralelo

- **SIPO** = conversión **serie→paralelo** (recepción: llega bit a bit, se reconstruye la palabra). Ya vista en 4.2.
- **PISO** = conversión **paralelo→serie** (transmisión). Ya vista en 4.3.

Es la base de cualquier interfaz serie (UART, SPI…): el emisor usa un PISO, el receptor un SIPO.

7.3 Retardo de tiempo

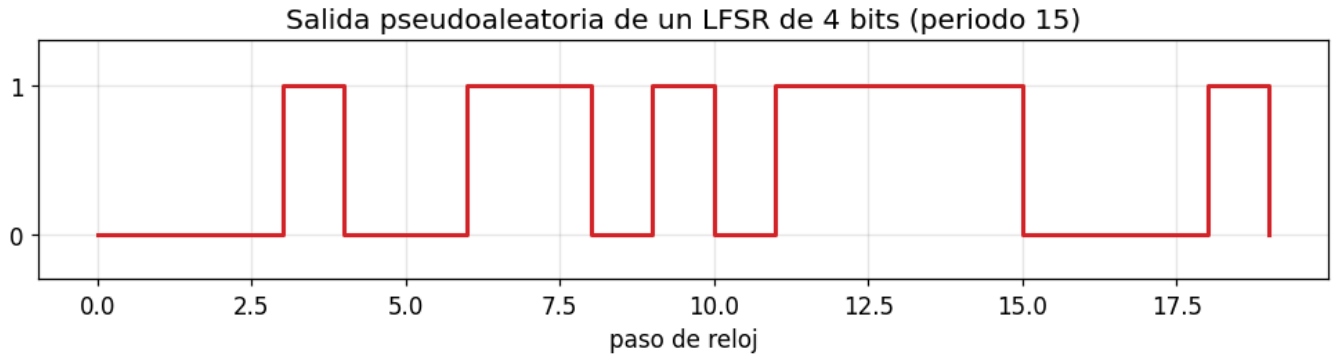
Un SISO de n biestables retrasa la señal **n ciclos** de reloj. Útil para sincronizar caminos de datos. (Es lo que vimos en el cronograma de 4.1.)

7.4 Generador de secuencias / números pseudoaleatorios (LFSR)

Un **LFSR** (*Linear Feedback Shift Register*) es un registro de desplazamiento cuya entrada serie se calcula con **XOR de algunas salidas** (los *taps*). Genera secuencias largas y deterministas que parecen aleatorias.

Con un registro de **m bits** y taps bien elegidos se obtienen 2^m estados antes de repetir (secuencia de longitud máxima, *m-sequence*). Es el "Generador de números aleatorios" de las diapositivas (9 bits → 29 estados en el ejemplo… en realidad 2^9 con taps maximales).

```
Bit de salida (serie): [0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0]
Valores del registro : [4, 2, 9, 12, 6, 11, 5, 10, 13, 14, 15, 7, 3, 1, 8]
Periodo              : 15 (esperado 15 = 2^4-1)
```



8. Resumen / chuleta de examen

- **Registro** = n biestables con el **mismo reloj**; almacena n bits. $_{n+1}$.
- **Habilitación de carga (E)**: actualiza solo si `E=1`; se hace con un MUX 2:1 que realimenta `Q` cuando `E=0`.
- **Estructura por celda**: MUX delante de cada D elige la entrada → unifica todos los tipos.
- **Desplazamiento**: $_{i}$ $_{i-1}$ (derecha) en cada flanco; entra el bit serie por el extremo.
- **4 tipos** según E/S: **SISO** (retardo), **SIPO** (serie→paralelo, 74HC164), **PISO** (paralelo→serie, 74HC165), **PIPO** (almacenamiento).
- **Bidireccional**: $_{i}$ $_{i-1}$ $_{i+1}$.
- **Universal (74HC194)**: `S1 S0` → {mantener, dcha, izda, carga paralela}.
- **Aplicaciones**: ×2 (despl. izda) / ÷2 (despl. dcha aritmético), conversión serie↔paralelo, retardo, **LFSR** (secuencias pseudoaleatorias, ^{m} estados con taps maximales).

> **Truco mnemotécnico**: la primera letra dice cómo **entra** el dato, la tercera cómo **sale**. SI**P**O = Serie In, Parallel Out.

Apuntes propios — Tema 8, Electrónica Digital (GIERM, US). Generado a partir de las diapositivas oficiales ED8-Registros.