Circuitos Secuenciales: Biestables

Electrónica Digital — 2º GIERM (Mecatrónica) Apuntes propios · Tema 7 · Enfoque examen (ejemplo resuelto primero)

> Cada bloque combina la teoría mínima de examen + tablas + **cronogramas** (timing diagrams) y **esquemáticos** generados con Python. Autocontenido: solo `numpy`, `matplotlib`, `schemdraw`.

Mapa del tema

1. Combinacional vs **secuencial** (qué es el "estado")
2. Principio de **biestabilidad** (realimentación positiva)
3. **Latch SR asíncrono** (NOR y NAND)
4. **Latch SR / D activo por nivel** (entrada de habilitación `E`)
5. Entradas **asíncronas** `Preset` y `Clear`
6. **Maestro-esclavo** y disparo por **flanco**
7. **Flip-flops** disparados por flanco: **D, SR, JK, T**
8. Tablas de **verdad** y de **excitación**; equivalencias entre biestables
9. **Cronogramas** completos (clock + entradas + salidas Q)

0. Preparación: utilidades de dibujo

Definimos una función auxiliar para dibujar **cronogramas digitales** de forma consistente en todo el notebook (señales como escalones 0/1 apiladas, con marcas opcionales de flanco).

```
Utilidades listas. schemdraw 0.22
```

1. Combinacional vs Secuencial

En un **circuito combinacional** la salida depende *solo* de las entradas en ese instante:

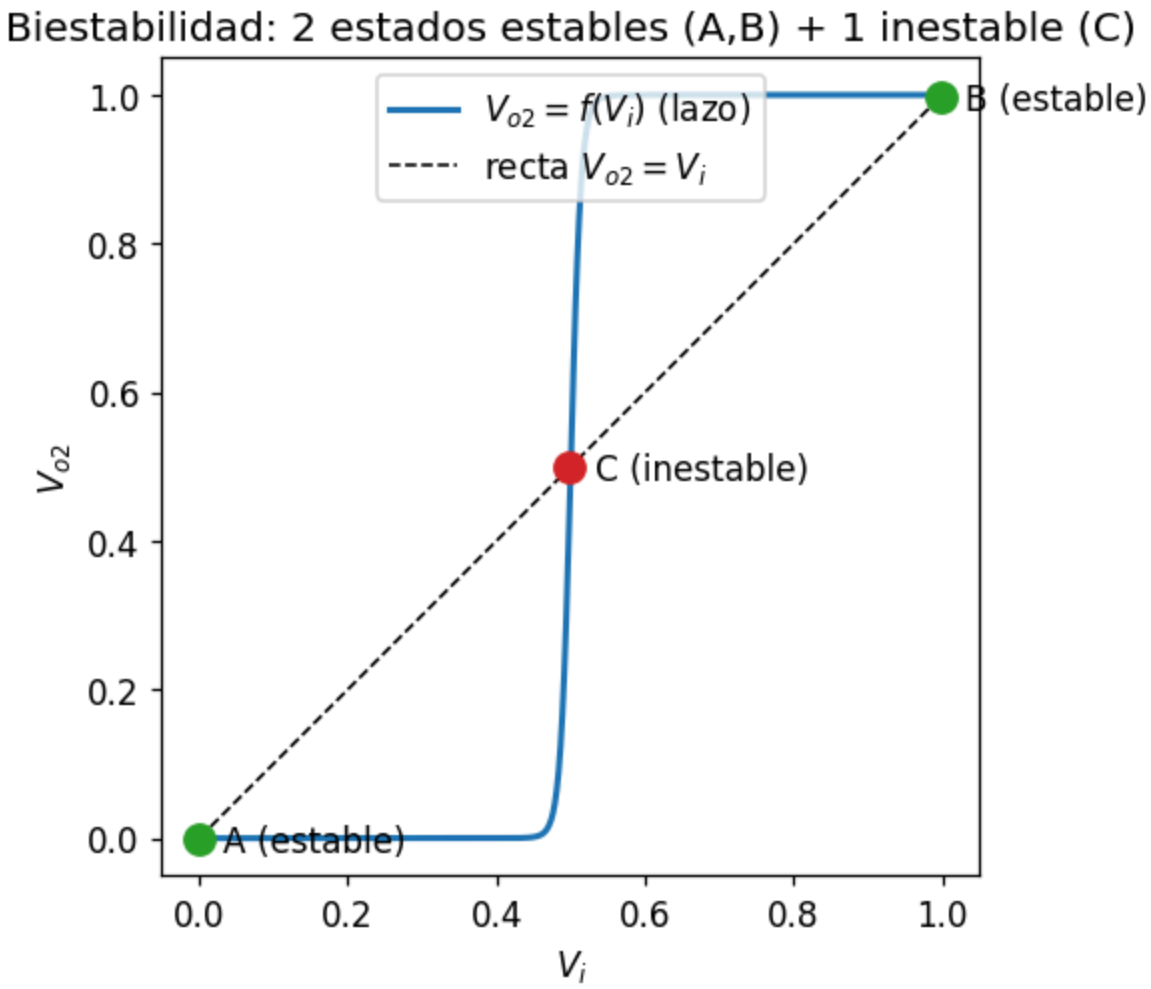
$$S_i = f(E_i)$$

En un **circuito secuencial** la salida depende de las entradas **y del estado** Q, que es la *memoria* de la historia pasada:

$$S_i = f(E_i,\ Q) \qquad \text{con} \qquad Q = g(E_i, E_{i-1}, \ldots, E_0)$$

	Combinacional	Secuencial
Salida depende de	solo entradas actuales	entradas + estado (historia)
Memoria	no	**sí** (variable de estado Q)
Ejemplo	multiplexor, sumador	latch, flip-flop, registro, contador
Realimentación	no	**sí** (positiva)

Principio de biestabilidad: dos inversores en serie realimentados tienen **dos puntos estables** (`A` =0, `B` =1) y uno inestable (`C`). Con ganancia alta ($|m| \gg 1$) cualquier perturbación lleva la salida a uno de los estados estables → *memoria de 1 bit*. El problema es que no se puede cambiar el estado sin cortocircuito, por eso se añaden entradas `S` / `R` .



2. Latch SR asíncrono (R-S con puertas NOR)

Es la celda de memoria más básica. **Asíncrono** = no hay reloj; reacciona en cuanto cambian las entradas. Construido con **dos puertas NOR** realimentadas, con `S` (Set) y `R` (Reset) **activas a nivel alto**.

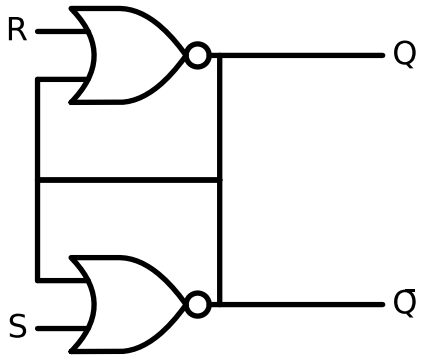
Tabla de verdad (NOR, entradas activas en alto)

S	R	Q_{n+1}	Comentario
0	0	Q_n	**Memoria** (mantiene)
0	1	0	**Reset**
1	0	1	**Set**
1	1	—	**Prohibido** ($Q = \bar{Q} = 0$)

> Con puertas **NAND** se obtiene el latch $\bar{S}\,\bar{R}$ con entradas **activas a nivel bajo**.
> Aplicación típica: **eliminación de rebotes** (debounce) de un pulsador.

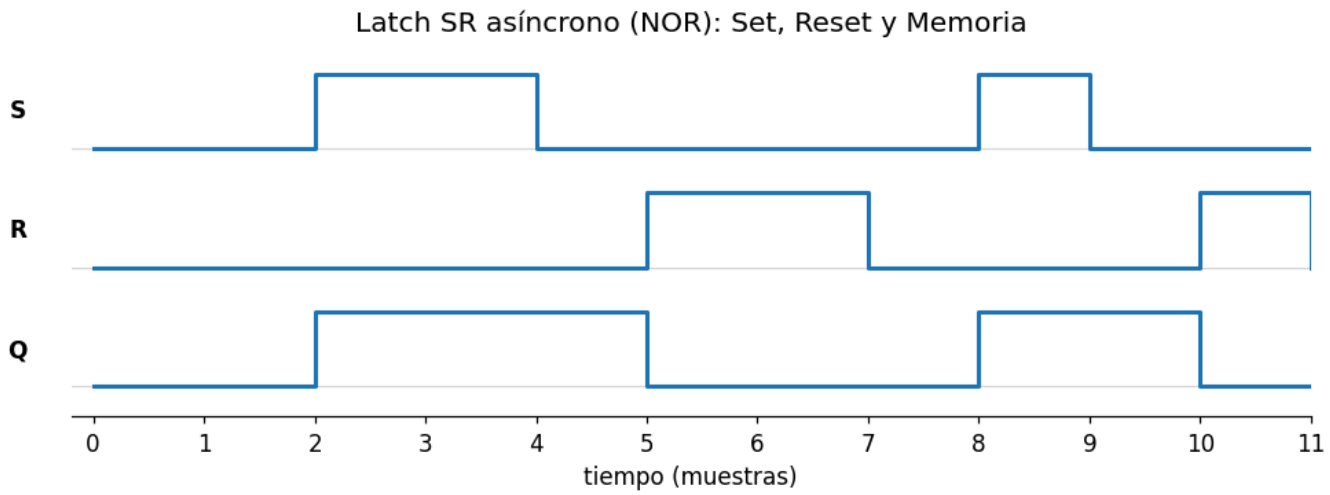
Esquemático (puertas NOR cruzadas)

```
Latch SR: dos NOR con realimentacion cruzada. S=set, R=reset.
```



Cronograma del latch SR asíncrono (NOR)

Simulamos el comportamiento muestra a muestra aplicando la tabla de verdad. Observa cómo en `S=R=0` la salida **recuerda** el último valor (memoria).



3. Latch activo por nivel (SR y D)

Añadimos una **entrada de habilitación** `E` (enable / *gate*). El latch solo "escucha" a `S` / `R` cuando `E=1`; si `E=0` **memoriza**. Esto es **activo por nivel** (level-triggered): es *transparente* mientras el nivel de `E` esté activo.

Latch SR por nivel — tabla

E	S	R	Q_{n+1}
0	x	x	Q_n (memoria)
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	— prohibido

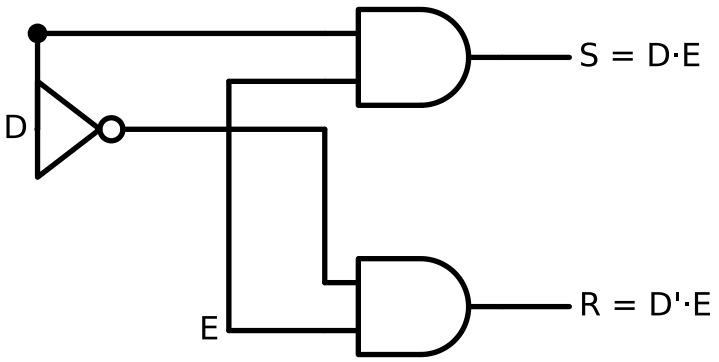
Latch D por nivel — el truco que elimina el estado prohibido

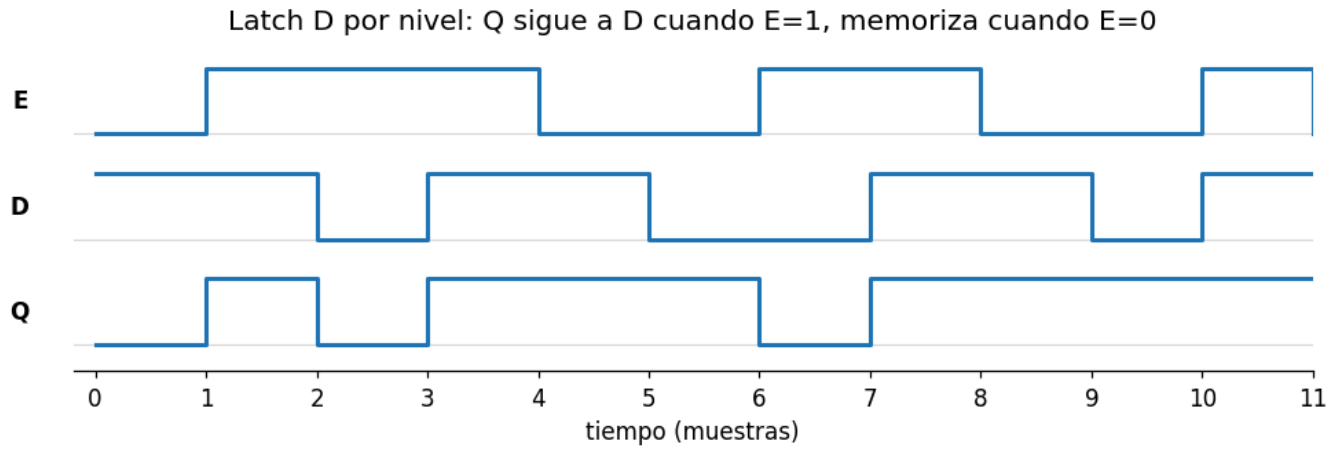
El **latch D** hace $R = \bar{S}$ (es decir `S=D`, `R=NOT D`), eliminando las combinaciones prohibidas. Con `E=1` la salida **sigue** a la entrada (`Q=D`, *transparente*); con `E=0` **retiene** el último valor.

E	D	Q_{n+1}
1	0	0
1	1	1
0	x	Q_n (memoria)

Esquemático: generación de S,R a partir de D y E

`Latch D: S=D.E, R=D-negada.E -> alimentan un latch SR por nivel (sin estado prohibido).`





4. Entradas asíncronas: Preset (Pr) y Clear (Cl)

Son entradas que **no dependen del reloj** y tienen **prioridad absoluta**: fuerzan la salida sin importar `D` / `J` / `K` ni el flanco.

- **Preset** ($\overline{Pr}$, activa en bajo) → fuerza `Q=1`.
- **Clear** ($\overline{Cl}$, activa en bajo) → fuerza `Q=0`.
- $\overline{Pr} = \overline{Cl} = 1$ → operación **normal** (síncrona).
- $\overline{Pr} = \overline{Cl} = 0$ → **prohibido**.

Sirven para **inicializar** el biestable a un estado conocido al arrancar el sistema (reset global). Las vemos en acción combinadas con un flip-flop D en §8.

5. Del nivel al flanco: Maestro-Esclavo y disparo por flanco

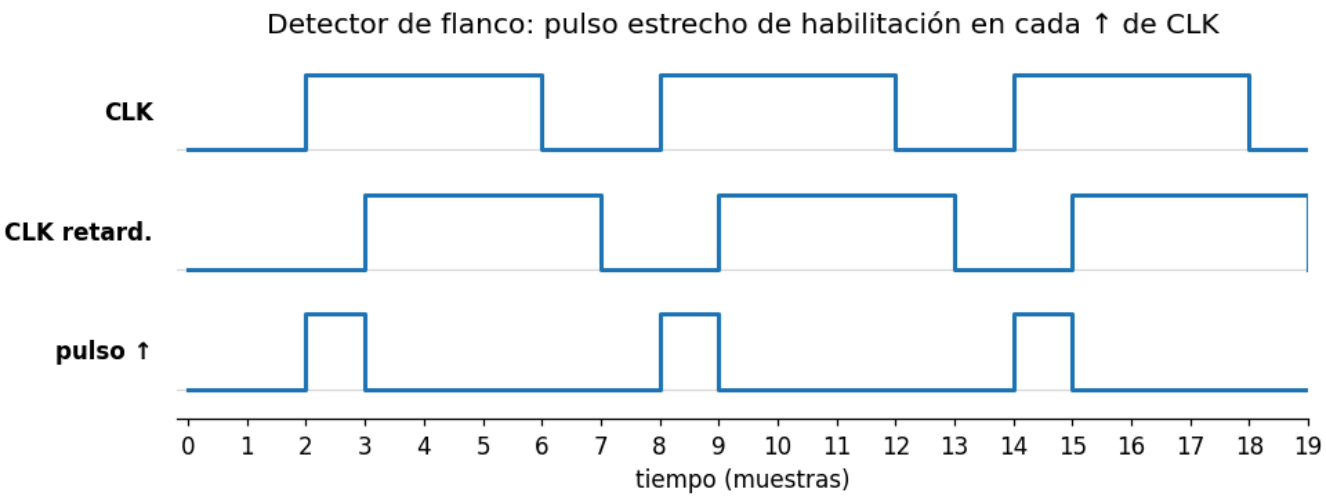
Problema del latch por nivel: mientras `E=1` es *transparente*, así que si la entrada cambia durante ese intervalo la salida también cambia → propenso a glitches y a *race-around* (en JK).

Solución 1 — Maestro-Esclavo (MS): dos latches en cascada con relojes **opuestos**.

- Con `CLK=1` : el **maestro** captura la entrada (esclavo bloqueado).
- Con `CLK=0` : el **esclavo** copia al maestro (maestro bloqueado).
- Resultado: la salida solo se actualiza una vez por ciclo → se comporta como **disparo por flanco** (de bajada, en este montaje).

Solución 2 — Disparo por flanco (edge-triggered): un **detector de flanco** genera un pulso estrechísimo (del orden de t_{pd}, ns) que habilita la captura **solo en el instante del flanco** (subida ↑ o bajada ↓). Es lo que usan los flip-flops D/JK/T modernos.

> Notación: triángulo `>` en la entrada de reloj del símbolo = disparo por flanco.



6. Flip-flops disparados por flanco: D, SR, JK, T

A partir de aquí todos los biestables capturan **en el flanco activo** del reloj (usaremos **flanco de subida** ↑). Estas tablas son las de memorizar para el examen.

Tabla de verdad (lo que hace en el flanco)

Tipo	Entradas	Q_{n+1}	Regla
D	D	D	copia la entrada
SR	S, R	set/reset/mem; `11` prohibido	poner / borrar
JK	J, K	ver tabla	como SR pero `11` = **toggle**
T	T	$Q_n \oplus T$	`T=1` conmuta, `T=0` mantiene

Tabla del JK (la estrella: no tiene estado prohibido)

J	K	Q_{n+1}	
0	0	Q_n	memoria
0	1	0	reset
1	0	1	set
1	1	$\bar{Q}_n$	**toggle**

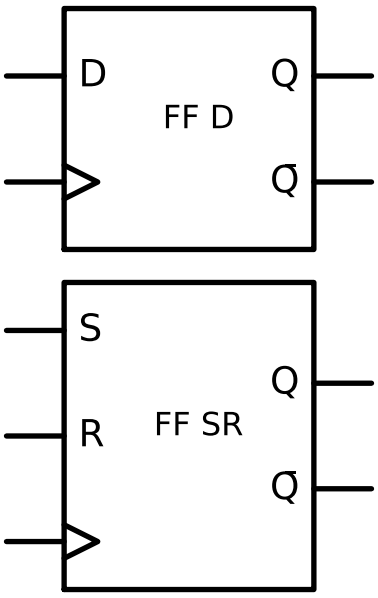
Tablas de **excitación** (qué entradas necesito para una transición $Q_n \to Q_{n+1}$)

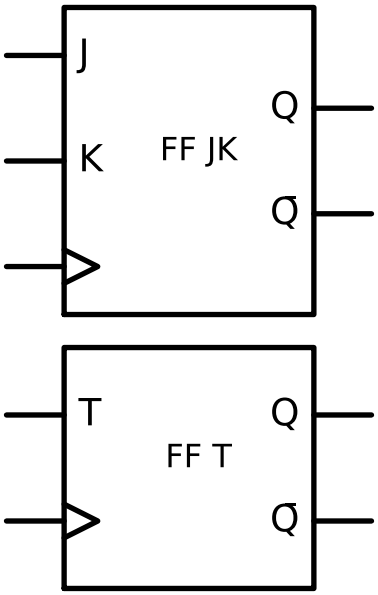
Imprescindibles para **diseñar** contadores/registros. `x` = indiferente.

$Q_n \to Q_{n+1}$	D	S R	J K	T
0 → 0	0	0 x	0 x	0
0 → 1	1	1 0	1 x	1
1 → 0	0	0 1	x 1	1
1 → 1	1	x 0	x 0	0

Símbolos de los flip-flops (schemdraw)

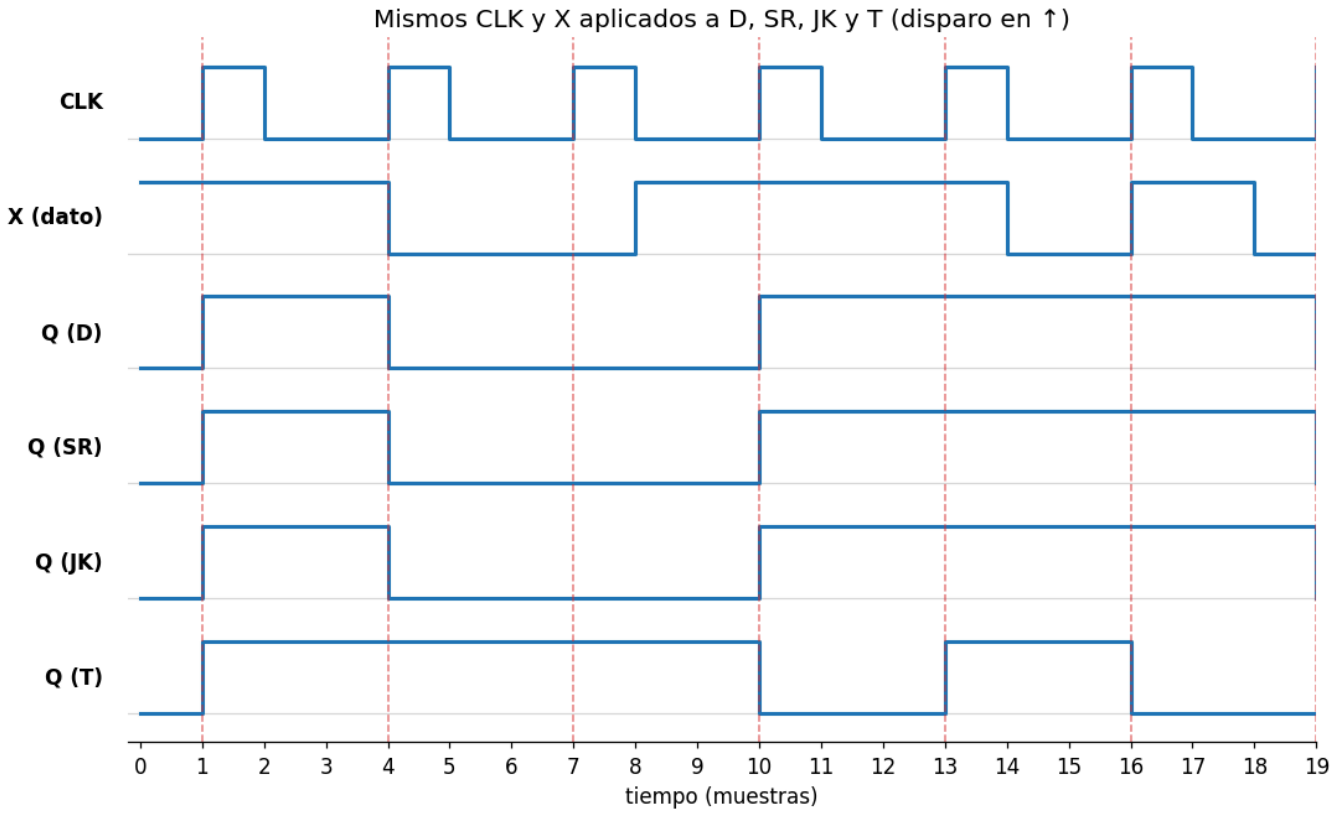
```
Simbolos D, SR, JK, T (el triangulo > marca disparo por flanco).
```





Cronograma de los 4 flip-flops con la MISMA entrada

Aplicamos la misma señal de datos y el mismo reloj a los cuatro tipos y comparamos sus salidas. Todos disparan en **flanco de subida** (líneas rojas discontinuas).



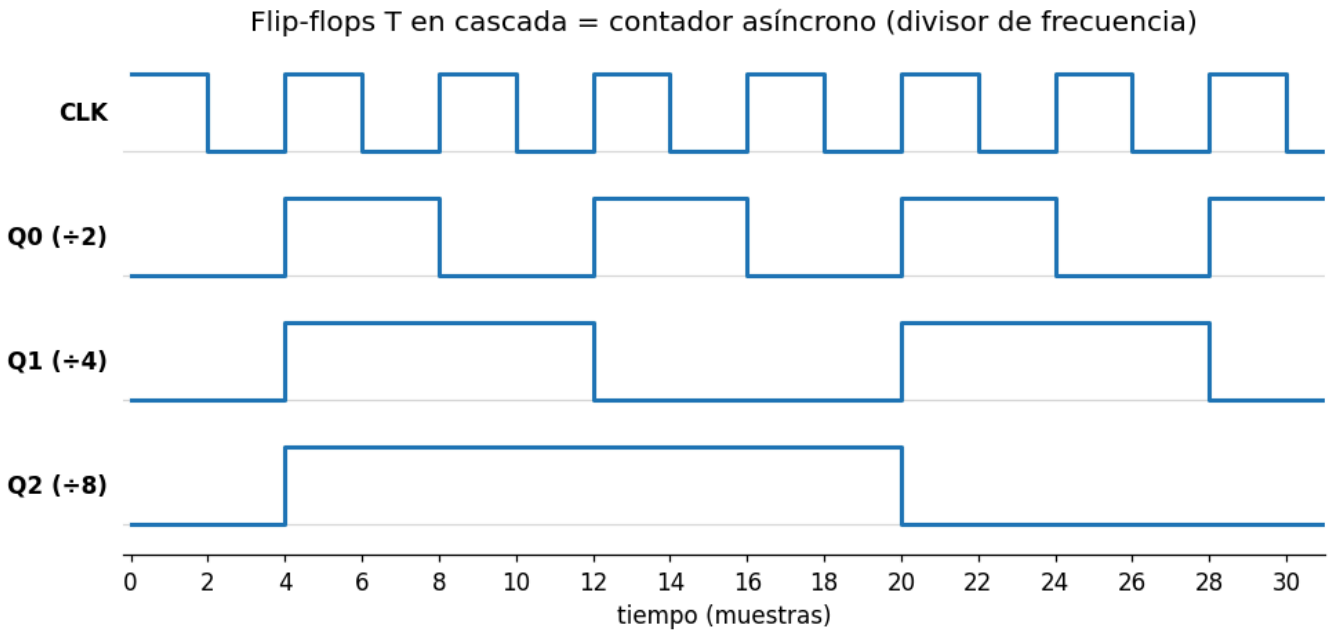
Lectura del cronograma:

- **D, SR (con R=$\bar{S}$), JK (con K=$\bar{J}$):** todos copian `X` en el flanco → salidas idénticas. Esto confirma que `D ≡ JK(J=D,K=¬D) ≡ SR(S=D,R=¬D)`.

- **T:** con `T=X`, cada flanco con `X=1` **conmuta** la salida; con `X=0` la mantiene → forma de onda distinta.

7. El flip-flop T como divisor de frecuencia

Caso clásico de examen: **JK con J=K=1** (o **T con T=1**) → **toggle** en cada flanco → la salida `Q` tiene **la mitad de la frecuencia** del reloj. Encadenando varios se obtiene un **contador asíncrono** (ripple counter).



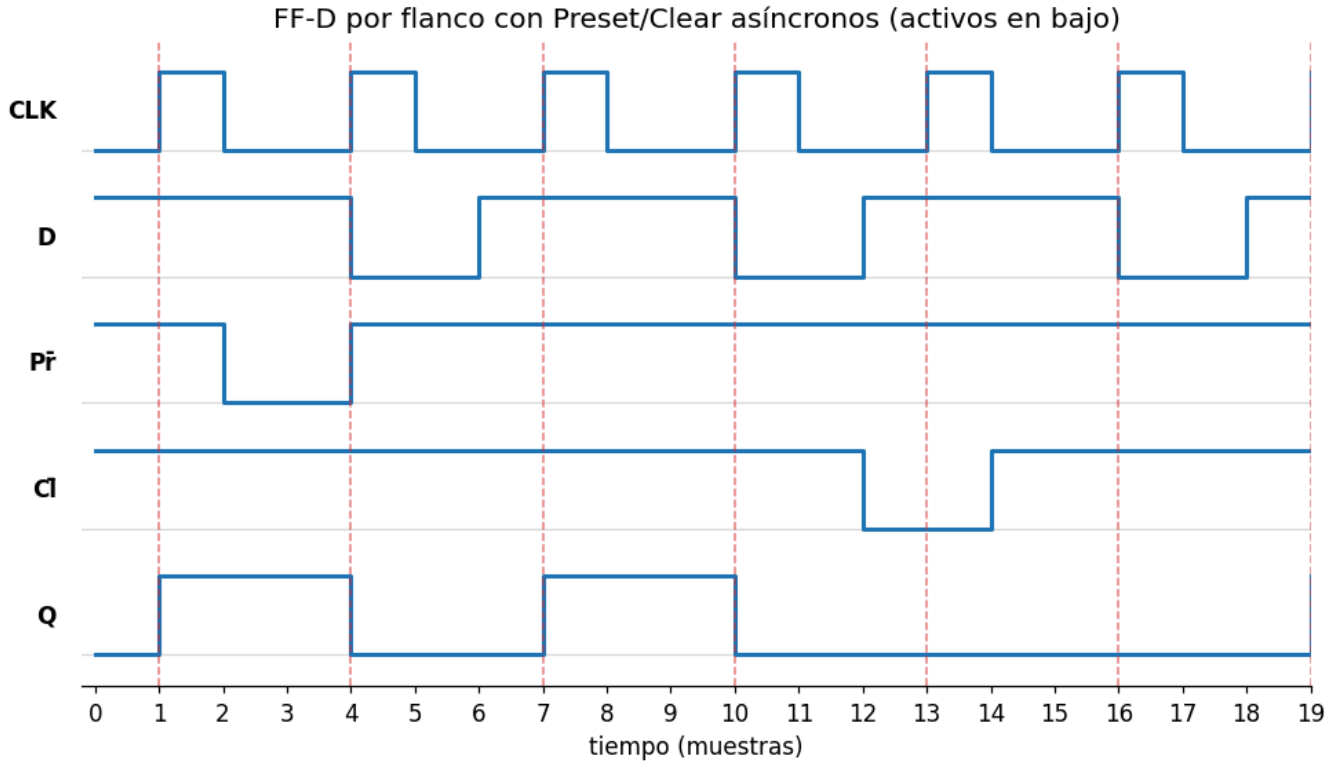

```
Cuenta binaria Q2Q1Q0 en cada muestra:
[0 0 0 0 7 7 7 7 6 6 6 6 5 5 5 5 4 4 4 4 3 3 3 3 2 2 2 2 1 1 1 1]
```

8. Flip-flop D con Preset y Clear asíncronos (worked example)

Juntamos todo: un flip-flop **D** disparado por flanco con entradas **asíncronas** $\overline{Pr}$ y $\overline{Cl}$ (activas en bajo, prioridad sobre el reloj).

Reglas (en orden de prioridad):

1. Si $\overline{Cl} = 0$ → `Q=0` (inmediato, ignora reloj).
2. Si $\overline{Pr} = 0$ → `Q=1` (inmediato).
3. Si ambas en 1 → en el **flanco** ↑: `Q=D`. Entre flancos: memoria.



Observa:

- En muestra 2 baja $\overline{Pr}$ → `Q` salta a 1 **sin esperar reloj**.
- En muestra 12 baja $\overline{Cl}$ → `Q` cae a 0 **inmediatamente**, ignorando `D` y el flanco.
- En el resto, `Q` solo cambia en los flancos ↑, copiando `D`.

9. Equivalencias entre biestables (resumen de examen)

Cualquier biestable puede construirse a partir de otro añadiendo lógica combinacional a sus entradas, usando la **tabla de excitación**:

Quiero…	A partir de…	Conexión
D	JK	$J = D,$ $K = \bar{D}$
D	SR	$S = D,$ $R = \bar{D}$
T	JK	$J = K$ $= T$
T	D	$D = Q$ $\oplus T$
JK	SR	

Quiero…	A partir de…	Conexión
		$S = J\bar{Q}$, $R = K$ Q

Lo comprobamos numéricamente en §6 (D ≡ JK con $K = \bar{J}$ ≡ SR con $R = \bar{S}$).

Resumen final (chuleta)

- **Combinacional** = sin memoria; **secuencial** = con estado Q (realimentación positiva).
- **Latch** = activo por **nivel** (transparente mientras `E` / `CLK` activo).
- **Flip-flop** = activo por **flanco** (captura solo en ↑ o ↓). Maestro-esclavo o detector de flanco.
- **SR**: tiene estado prohibido (`11`). **D**: copia, sin prohibido. **JK**: `11` =toggle. **T**: `1` =toggle.
- **Pr/Cl asíncronos**: prioridad absoluta sobre el reloj, para inicializar.
- **T/JK toggle** → divisor de frecuencia → contadores.
- Para **diseñar**: usa la **tabla de excitación** de cada biestable.

Apuntes generados a partir de `ED7-Circuitos Secuenciales. Biestables_2022.pdf` (Universidad de Sevilla). Cronogramas con matplotlib, esquemáticos con schemdraw; la lógica de cada biestable se simula en Python a partir de su tabla de verdad.