Logica e Reti Logiche
(Episodio 14: Introduzione ai circuiti sequenziali)

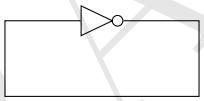
Francesco Pasquale

30 novembre 2023

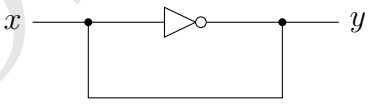
Tutti i circuiti che abbiamo incontrato negli episodi precedenti sono "aciclici": non c'è mai un "percorso" che dall'output di una porta logica P torna in input alla stessa porta. Questi circuiti si chiamano *circuiti combinatori*. Pensate che siano gli unici circuiti possibili? In questo episodio vediamo che non è assolutamente così, e in realtà possiamo costruire dei circuiti molto interesanti che contengono dei cicli, ma bisogna farlo con attenzione...

1 Antipasto

Secondo voi che succede se costruiamo un circuito fatto così?

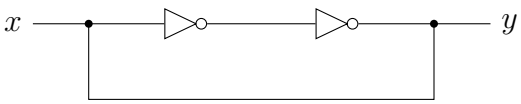


Sembra una specie di "paradosso logico": se l'input della porta NOT è 1 deve essere anche 0 e viceversa. Quindi se proviamo a costruirlo così

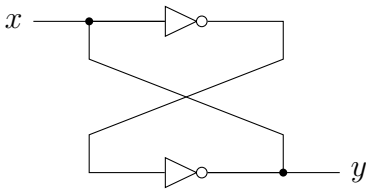


e diamo corrente all'input x, non è difficile immaginare che qualcosa di brutto succederà alla povera porta logica NOT lì in mezzo...

Tuttavia, costruire un circuito che contiene un ciclo non è sempre qualcosa che crea un *cortocircuito*. Prendete per esempio il circuito qui sotto

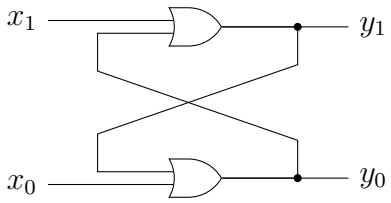


Anche se contiene un ciclo, in questo caso le due porte NOT non hanno niente da temere. Certamente è un circuito che fa ben poco: l'output y sarà uguale all'input x. Lo chiamiamo elemento *bistabile* e osservate che possiamo anche disegnarlo in quest'altro modo e descrivere l'output in funzione dell'input con una tabella di verità, come per ogni circuito visto finora



x	y
0	0
1	1

Cosa succede se proviamo a usare lo stesso schema utilizzando porte a due ingressi, per esempio porte OR in cui l'output di una delle porte entra in input nell'altra porta



Siccome l'output di una porta OR è 1 quando almeno uno dei due ingressi è 1, non c'è nessun dubbio su quali siano i valori degli output y_0 e y_1 quando almeno uno dei due input è 1: per esempio, se $x_0 = 1$ allora y_0 deve essere 1, e quindi anche y_1 sarà 1. Ma cosa possiamo dire sui bit in output (y_0, y_1) quando i bit in input sono entrambi zero, $(x_0, x_1) = (0, 0)$?

x_0	x_1	y_0	y_1
0	0	?	?
0	1	1	1
1	0	1	1
1	1	1	1

Se immaginiamo che il circuito "parta" da una configurazione iniziale in cui sia gli input che gli output sono 0, allora non appena poniamo a 1 uno dei due bit in input entrambi gli output andranno a 1 e da lì in poi rimarranno sempre $(y_0, y_1) = (1, 1)$.

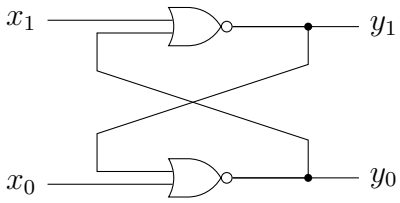
Come per l'elemento bistabile quindi anche in questo circuito, nonostante ci siano dei cicli, non creiamo nessun "paradosso logico" assegnando dei valori in input. Anche questo circuito però non fa nulla di interessante: manovrando gli input possiamo soltanto spostare una volta i valori di output da 0 a 1, che poi rimarranno fissi a 1 indipendentemente da come riassegnamo gli input.

Esercizio 1. Verificare che cambia ben poco se invece di porte OR usiamo porte AND.

Nella prossima sezione invece vediamo che se costruiamo un circuito in quel modo usando porte NOR (o NAND), non solo non ci sono paradossi logici, ma succede anche qualcosa di molto interessante.

2 SR-Latch e D-Latch

Consideriamo il circuito seguente, costruito a partire da porte NOR



e richiamiamo per comodità la tabella di verità di una porta NOR

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

Ricordate che quando abbiamo studiato la logica proposizionale abbiamo chiamato *joint denial* il connettivo corrispondente. È la negazione dell'OR, cioè vale 1 se e solo se entrambi i due valori in input sono 0.

Quindi come è fatta la tabella di verità del nostro circuito costruito con porte NOR?

Esercizio 2. Verificare che quando gli input (x_0, x_1) sono $(0,1)$, $(1,0)$ o $(1,1)$ gli output sono determinati

x_0	x_1	y_0	y_1
0	0	?	?
0	1	1	0
1	0	0	1
1	1	0	0

Cosa succede quando abbiamo in input $(x_0, x_1) = (0,0)$?

Esercizio 3. Verificare che quando $(x_0, x_1) = (0,0)$, i valori di y_0 e y_1 rimangono uguali a come erano "in precedenza", se in precedenza erano $(1,0)$ oppure $(0,1)$: $y_0 = y_0^{(\text{precedente})}$ e $y_1 = y_1^{(\text{precedente})}$, se $(y_0^{(\text{precedente})}, y_1^{(\text{precedente})}) \in \{(0,1),(1,0)\}$

x_0	x_1	y_0	y_1
0	0	$y_0^{(\text{prec})}$	$y_1^{(\text{prec})}$
0	1	1	0
1	0	0	1
1	1	0	0

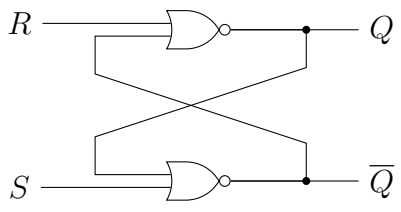
Con questo circuito quindi possiamo, per esempio, impostare per un attimo gli input a $(x_0, x_1) = (1,0)$ e così *settare* il bit in output $y_1 = 1$ (e il bit $y_0 = 0$), riportare gli input a $(x_0, x_1) = (0,0)$ e y_1 rimarrà a 1 (e y_0 a 0). Inoltre, impostando per un attimo gli input a $(x_0, x_1) = (0,1)$ possiamo *resettare* il bit $y_1 = 0$.

Osservate che abbiamo appena costruito una *memoria*: un circuito in grado di fissare un bit in output y_1 a 0 o 1, a seconda dei valori assegnati agli input (x_0, x_1), e *ricordare* il bit assegnato a y_1 quando entrambi gli input tornano ad avere valore 0.

Il circuito che abbiamo costruito si chiama $SR - Latch$: gli input si chiamano S e R (Set e Reset) e gli output Q e $\overline{Q}$, e il simbolo con cui si indica il circuito è quello in Figura 1.

Esercizio 4. Costruire un SR-Latch usando porte NAND invece che NOR, facendo attenzione a qual è l'input di *set* e quale quello di *reset*.

Osservate che chiamare gli output di un SR-Latch Q e $\overline{Q}$ non è propriamente corretto, perché se gli input R e S sono entrambi 1 gli output sono entrambi zero. Inoltre impostare entrambi gli input a 1 crea anche un altro problema:



S	R	Q	$\overline{Q}$
0	0	$Q^{(\text{prec})}$	$\overline{Q}^{(\text{prec})}$
0	1	0	1
1	0	1	0
1	1	0	0

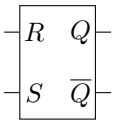
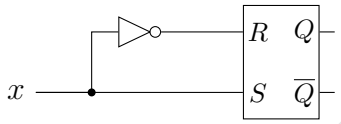


Figura 1: Schema, tabella e simbolo di un SR-Latch costruito con porte NOR

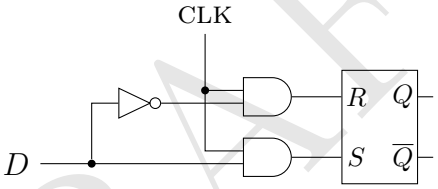
Esercizio 5. Se in un determinato istante in un SR-Latch diamo in input $(R, S) = (1, 1)$ e poi passiamo direttamente a $(R, S) = (0, 0)$ cosa succede agli output?

Dobbiamo assolutamente eliminare la possibilità che gli input possano essere entrambi 1.

Esercizio 6. Spiegare perché una soluzione come quella in figura qui sotto non va bene



Esercizio 7. Si consideri il circuito seguente con due input, D e CLK. Osservare che quando CLK $= 1$ il valore dell'output Q è uguale al valore dell'input D mentre quando CLK $= 0$ il valore di Q rimane fisso.



Il circuito costruito nell'esercizio precedente, oltre a garantire che i due output Q e $\overline{Q}$ saranno sempre uno il negato dell'altro, consente di separare *quale* bit viene assegnato all'output Q (il bit in input D) da *quando* viene assegnato (ossia, quando CLK $= 1$).

Il circuito così costruito si chiama D-Latch e il simbolo usato è quello in Figura 2. Quando CLK $= 1$ si dice che il latch è *trasparente* (fa passare il valore dell'input D all'output Q), quando invece CLK $= 0$ si dice che il latch è *opaco* (anche se l'input D cambia il valore di Q rimane lo stesso).

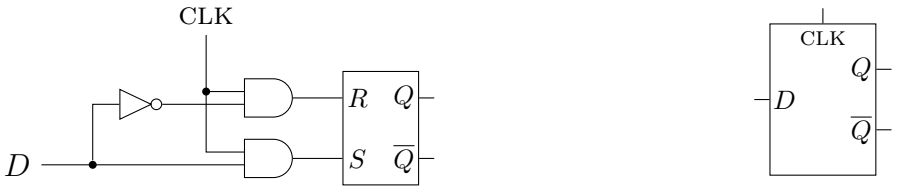
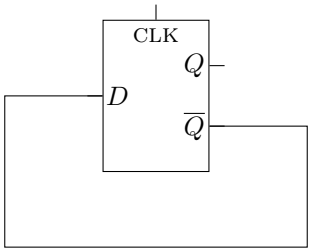


Figura 2: Schema e simbolo di un D-Latch

3 Flip-Flop e registri

Un D-Latch ci consente di *settare* il valore di Q quando CLK $= 1$ e *tenerlo in memoria* per tutto il tempo in cui CLK $= 0$. Tuttavia il fatto che il latch è "trasparente" quando CLK $= 1$ può essere uno svantaggio in alcuni casi. Considerate, per esempio, un circuito fatto così



Quando CLK = 0 non c'è nessun problema: il valore in input D sarà uguale al negato dell'output Q, ma il valore di Q non cambia perché il latch è opaco. Nel momento in cui poniamo CLK = 1 però creiamo un "paradosso logico".

Esercizio 8. Ricostruire il circuito nella figura qui sopra esplicitando le porte elementari (NOR, AND e NOT) da cui è formato e osservare il caso CLK = 1.

Il problema si può risolvere mettendo in sequenza due D-latch collegati ad un unica variable CLK che entra negata nel primo latch e asserita nel secondo. Il circuito che otteniamo si chiama *Flip-Flop*: lo schema e il simbolo sono in Figura 3

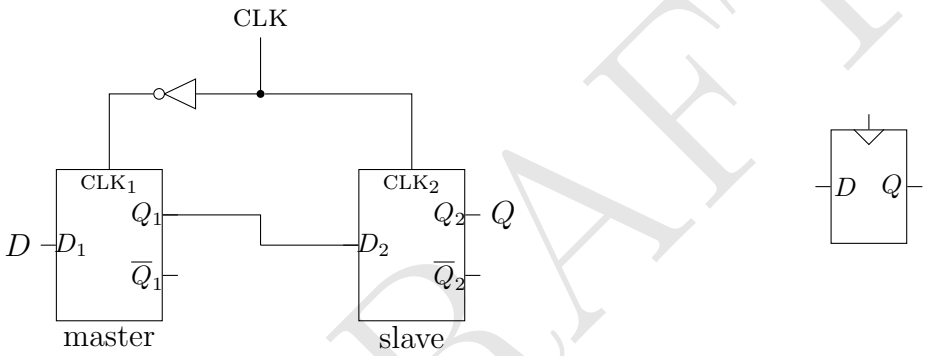


Figura 3: Schema e simbolo di un D-FlipFlop

In un D-FlipFlop:
- Quando CLK = 0 il latch *master* è trasparente (il valore dell'input D passa a Q_1 e quindi a D_2) ma il latch *slave* è opaco (il valore di D_2 non passa a Q)
- Quando CLK = 1 il latch *slave* è trasparente (il valore di D_2 passa a Q) ma il latch *master* è opaco (il valore dell'input D non passa a Q_1)

In altre parole, il FlipFlop fa passare il valore dell'input D all'output Q solo "nel momento" in cui il valore di CLK passa da 0 a 1.

Se pensiamo all'input CLK come a una variabile che periodicamente cambia valore da 0 a 1 e viceversa[1] l'output Q rappresenta lo *stato attuale* del FlipFlop e l'input D il suo *stato futuro* (ossia quello che diventerà lo stato attuale la volta successiva che CLK passerà da 0 a 1).

Esercizio 9. Un *T-FlipFlop* ha un solo input, CLK, e un output Q. Ogni volta che CLK passa da 0 a 1, il valore di Q cambia (se è 0 diventa 1, se è 1 diventa 0). Progettare un *T-FlipFlop* utilizzando un *D-FlipFlop* e una porta `not`.

[1] Questo è quello che succede effettivamente in una *CPU*: una *frequenza di clock* di 3GHz, per esempio, significa che la variabile CLK del processore oscilla fra 0 e 1 circa tre miliardi di volte al secondo.

Esercizio 10. Un *JK-FlipFlop* ha un output, Q, e tre input: CLK, J e K. Ogni volta che CLK passa da 0 a 1, il valore di Q si aggiorna in questo modo:
- Se $J = K = 0$ allora Q mantiene il suo valore precedente;
- Se $J = K = 1$ allora Q cambia valore (se era 0 diventa 1, se era 1 diventa 0);
- Se $J = 1$ e $K = 0$ allora Q assume valore 1;
- Se $J = 0$ e $K = 1$ allora Q assume valore 0.

1. Progettare un *JK-FlipFlop* usando un *D-FlipFlop* e le opportune porte logiche;
2. Progettare un *D-FlipFlop* usando un *JK-FlipFlop* e le opportune porte logiche;
3. Progettare un *T-FlipFlop* usando un *JK-FlipFlop*.

Una serie di FlipFlop con i clock sincronizzati costituiscono collegati un *registro*

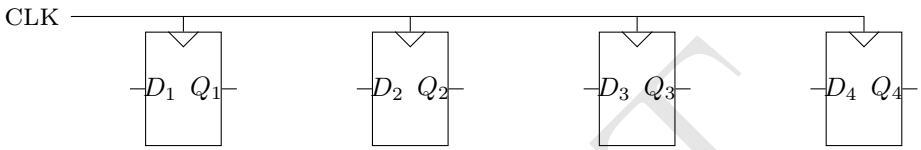
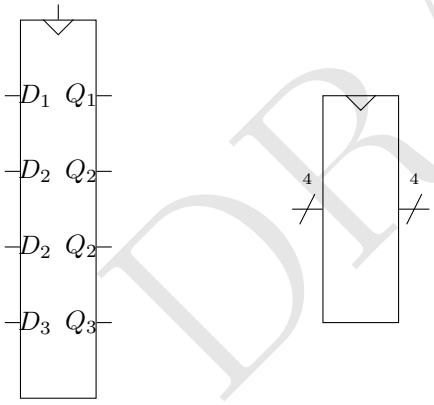


Figura 4: Un registro a quattro bit

Possiamo disegnare un registro a 4 bit anche in questo modo



Esercizio 11. In quanti stati distinti può trovarsi un registro a n bit?

I FlipFlop, come gli altri blocchi funzionali, possono avere ulteriori input. Per esempio per *abilitarli* (*enabling*: il FlipFlop deve comportarsi normalmente quando EN = 1, mentre quando EN = 0 il FlipFlop non deve fare niente) o per *resettarli* (il FlipFlop deve impostare l'output a 0 quando l'input di reset è 1 mentre deve comportarsi normalmente quando è 0).

Esercizio 12. Mostrare come si possono costruire FlipFlop con input di abilitazione e di reset.