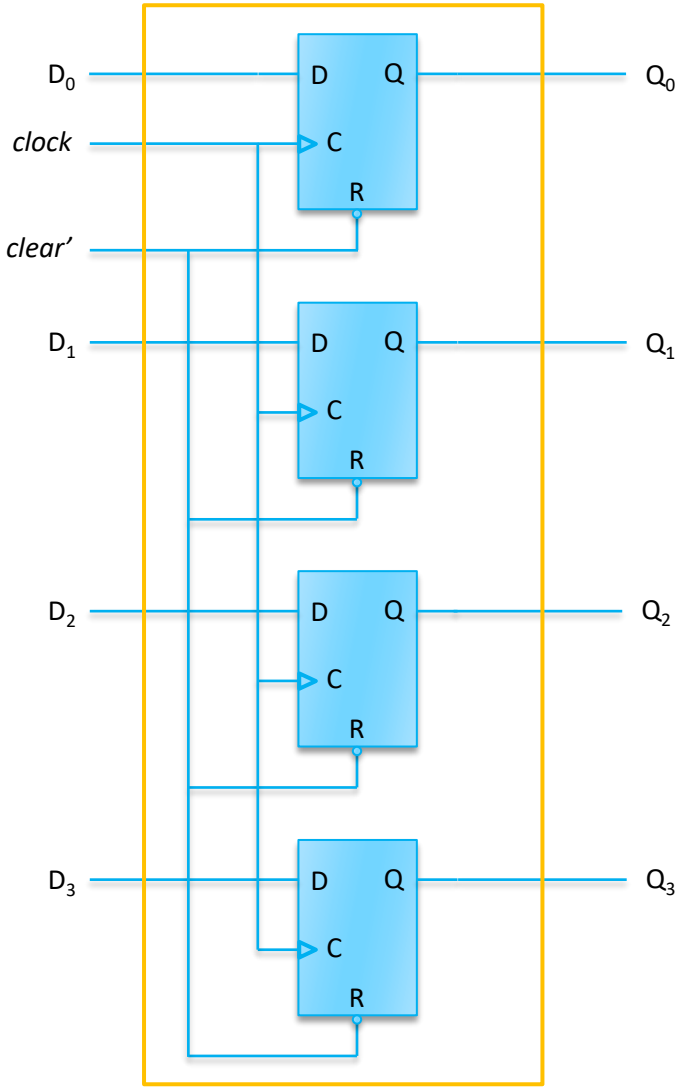
Circuiti sequenziali di base

Registri, contatori, et similia

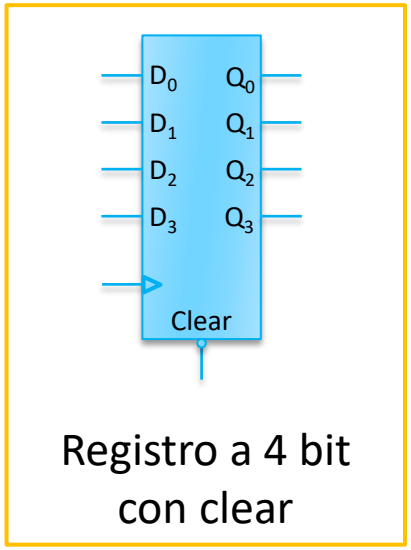
I registri

- **Il registro è l'elemento sequenziale di base**
  - E' usato più o meno come una variabile e memorizza un valore
  - Nella sua realizzazione più semplice è solamente un flip flop di tipo D che memorizza un valore a un bit
  - In generale il valore potrà essere di molti bit, nel qual caso si usano flip flop in parallelo
- **La differenza tra un registro ed un gruppo di flip flop sta nel fatto che i valori nel registro sono logicamente correlati**
  - Per esempio sono i bit di un numero binario
  - Oppure esprimono condizioni di funzionamento del circuito
- **I registri sono usati ovunque**
  - Per esempio i registri di un microprocessore
  - Gli accumulatori di circuiti aritmetici
  - Valori di programmazione e stato di periferiche

Registro parallelo con Clear

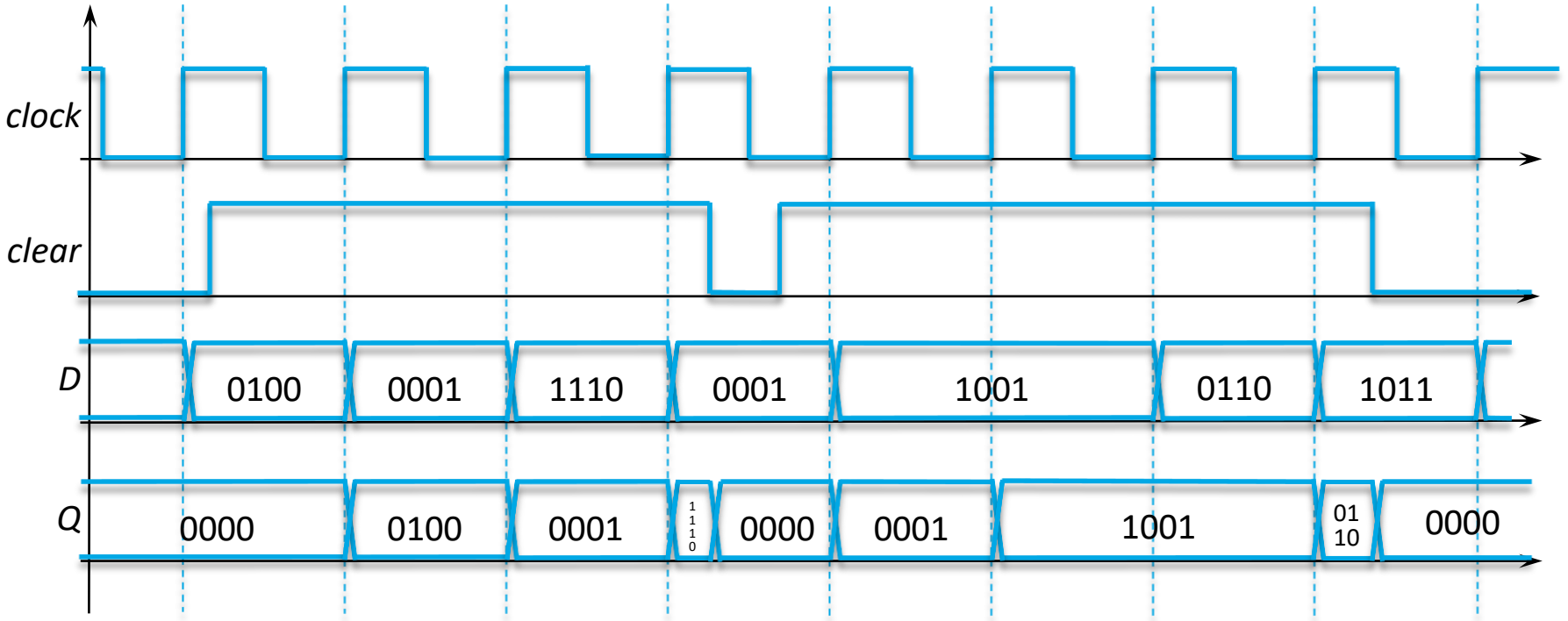


- **Il registro è un semplice flip flop D**
  - Carica un nuovo valore ad ogni fronte del clock
  - Azzerato con il comando Clear asincrono, per esempio attivo basso
  - Multi-bit con configurazione parallela di registri identici



Registro a 4 bit con clear

Diagramma temporale



- **Il registro carica ad ogni fronte attivo un nuovo valore**
  - L'uscita Q appare quindi come una versione ritardata dell'ingresso D
- **Clear attivo basso asincrono**
  - Impone immediatamente il valore 0 fino al primo fronte attivo del clock in cui il comando non è più presente

Registro parallelo con load enable

- **Il registro carica un nuovo valore ad ogni fronte del clock**
  - Talvolta può essere conveniente controllare per quali colpi di clock caricare un nuovo valore
  - Si aggiunge un segnale di load enable
- **Clock gating**
  - Un metodo per realizzare il dispositivo è quello di mascherare il segnale di clock
  - Si fa arrivare il clock solo per i cicli in cui si vuole effettivamente caricare un nuovo valore e lo si tiene costante ad un valore per il resto del tempo

Problemi con il clock gating

- **In un circuito sincrono, tutti i registri devono commutare nello stesso istante**
  - Questo è necessario per evitare comportamenti errati
  - Se un registro commuta troppo presto, gli effetti delle sue uscite possono raggiungere gli ingressi degli altri registri *prima* del segnale di clock, e quindi fornire un valore errato
  - Analogamente se un registro commuta troppo tardi, può ricevere dati errati dagli altri registri
- **La rete di distribuzione del clock è complessa**
  - Immaginate portare lo stesso segnale a migliaia di elementi diversi
  - Si usano normalmente alberi di distribuzione per bilanciare la lunghezza dei collegamenti
- **Morale: il segnale di clock non si tocca**
  - E' già abbastanza problematico, meglio lasciarlo stare
  - Oggi lo si fa, ma solo per risparmiare energia

Realizzazione sincrona

- **Si sceglie il valore da caricare**
  - Se *load* è attivo, si carica il valore dell'ingresso
  - Se *load* non è attivo, si carica il valore già memorizzato
  - La scelta può essere fatta agevolmente tramite un multiplexer
  - La connessione parallela realizza il registro multi-bit
- **Circuito più complesso del precedente**
  - Ma più sicuro dal punto di vista del clock
  - Si può aggiungere il segnale di *clear*

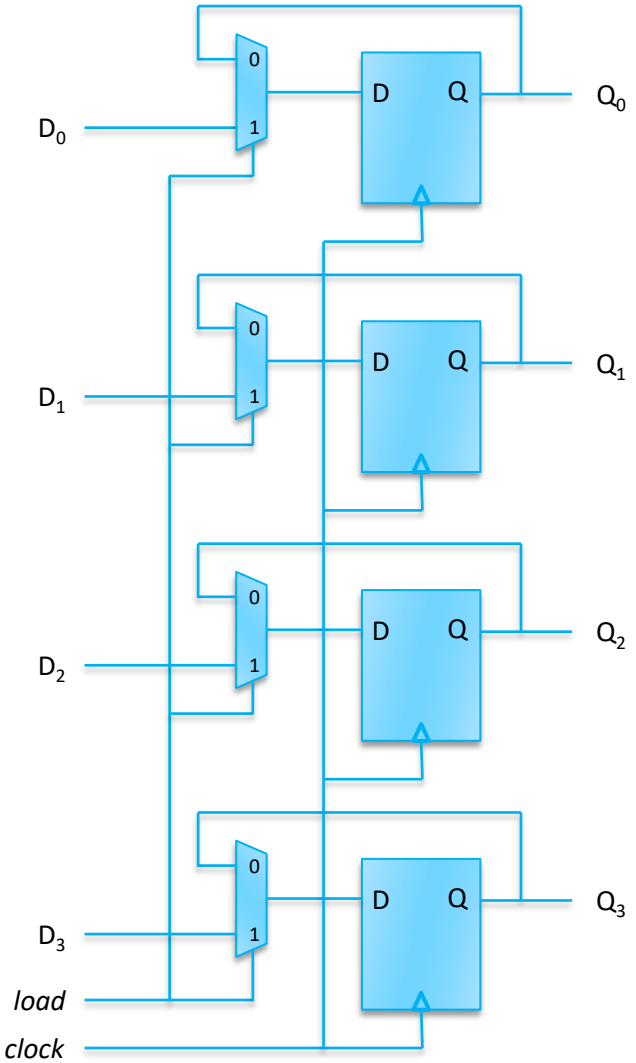
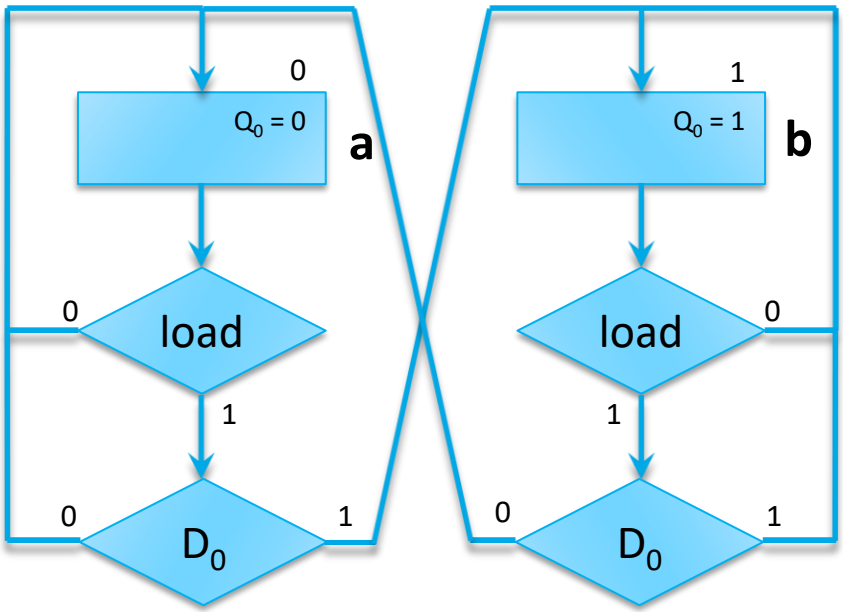
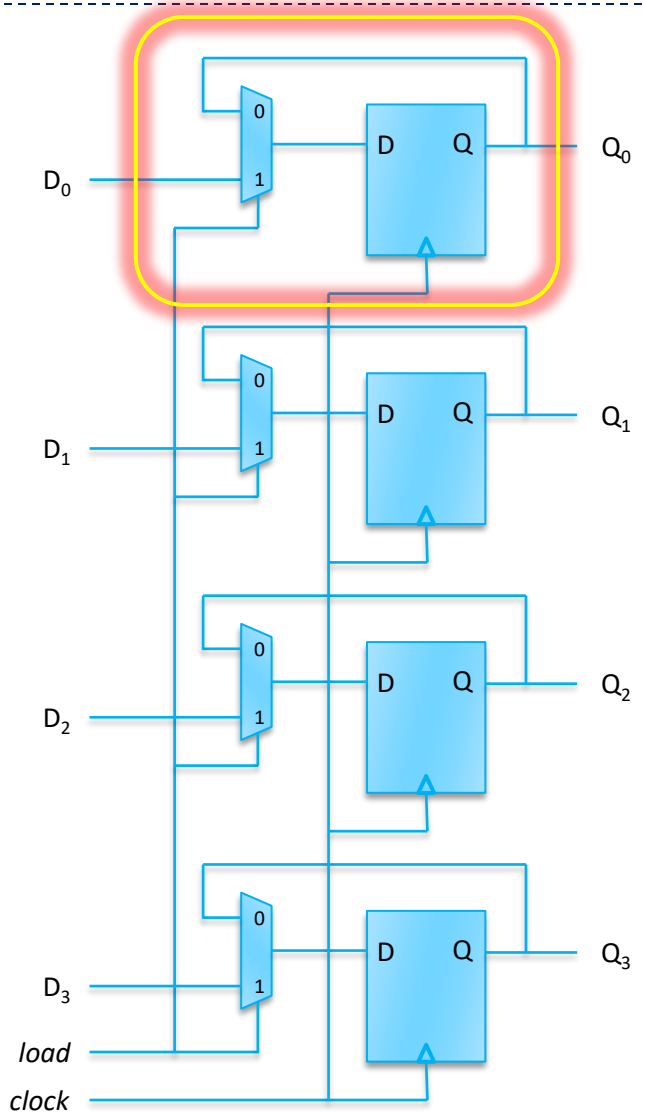


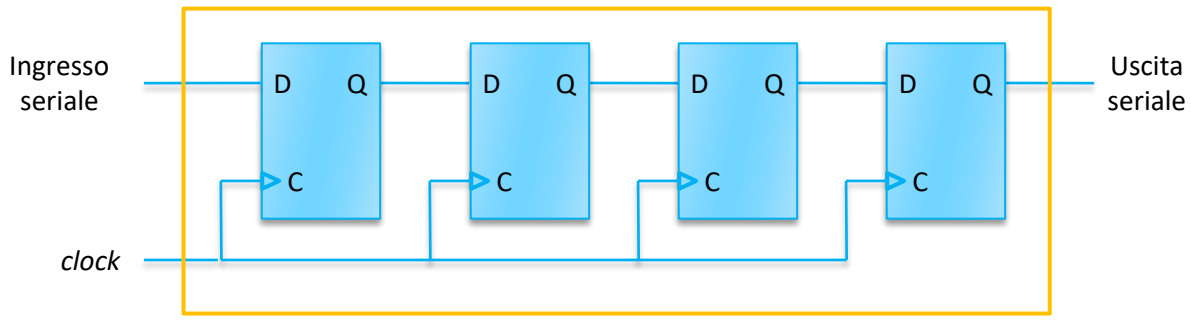
Diagramma a stati





Registro a scorrimento (shift register)

- **Un tipo di registro in cui i bit scorrono da un flip flop al successivo**
  - Al contrario di quello parallelo, dove un dato parallelo viene caricato in un colpo solo
  - Il collegamento dei flip flop avviene quindi in serie
  - Ad ogni colpo di clock il contenuto si trasferisce verso destra



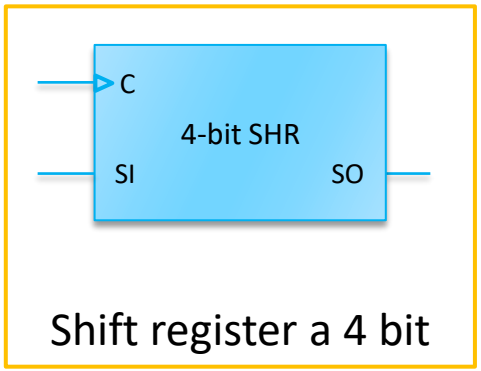
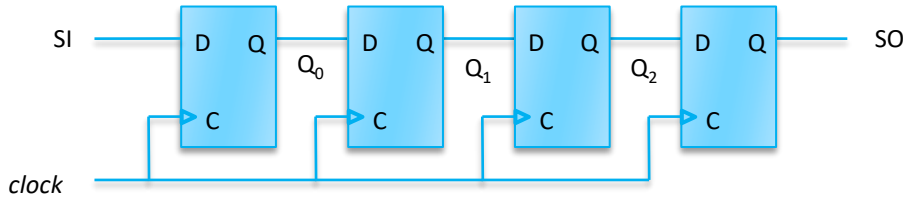


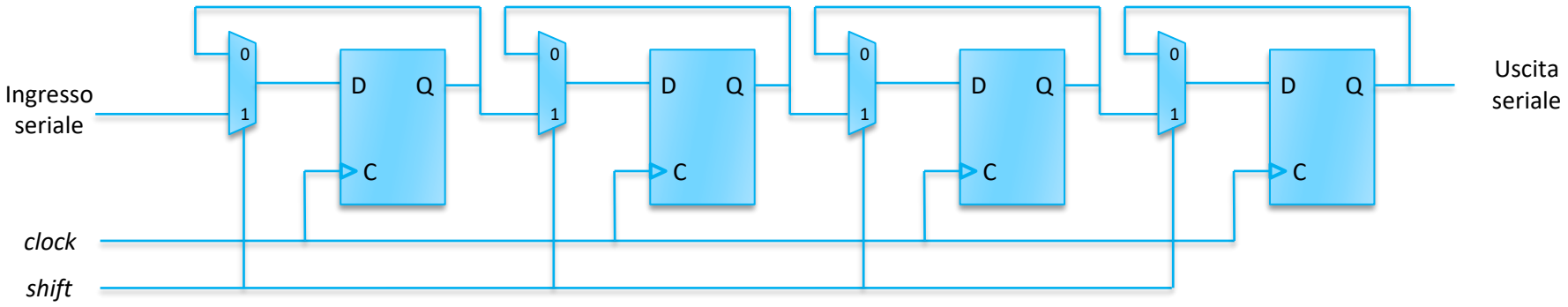
Diagramma temporale



- **Il dato entra e si trasferisce in forma seriale**

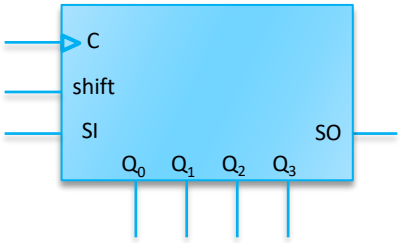
Shift register con shift enable

- **Come per il registro parallelo, si può abilitare o disabilitare lo scorrimento al colpo di clock**
  - Mascherando il segnale di clock
    - Attenzione che la rete combinatoria tra un registro e il successivo è solo un filo, quindi è velocissima
  - Interponendo un multiplexer all'ingresso di ogni flip flop

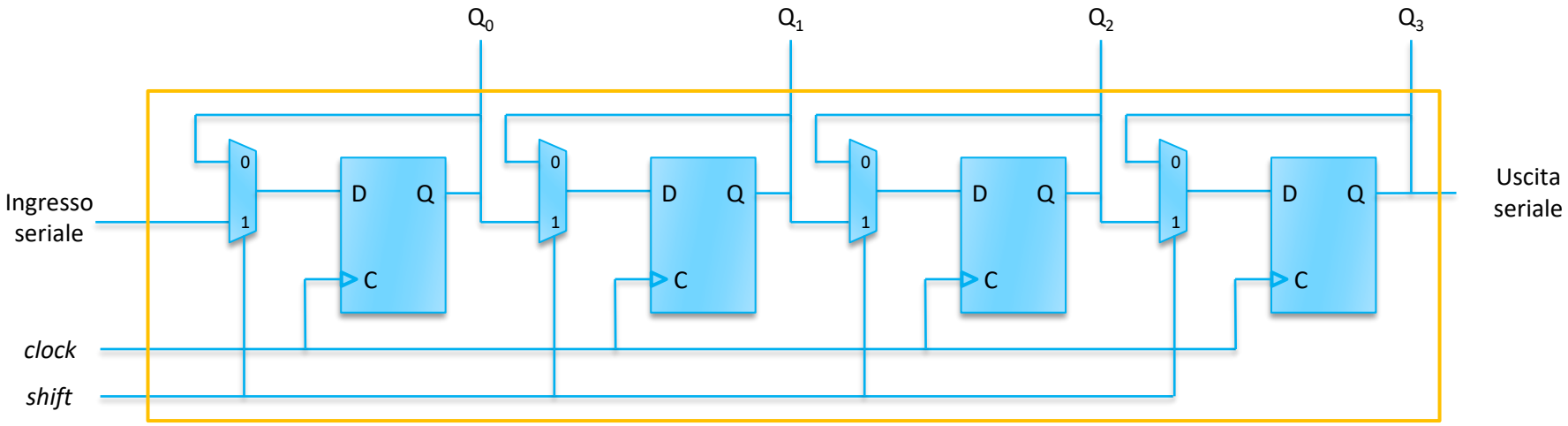


Shift register con uscita parallela

- **I registri a scorrimento offrono banalmente l'uscita parallela**
  - Basta tirare fuori le singole uscite di tutti i flip flop, invece della sola uscita dell'ultimo flip flop



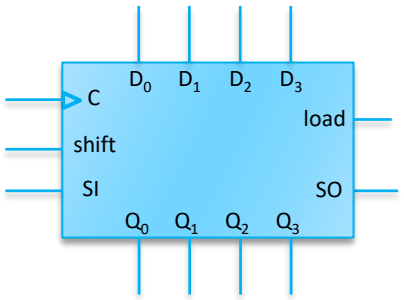
Shift register a 4 bit con uscita parallela



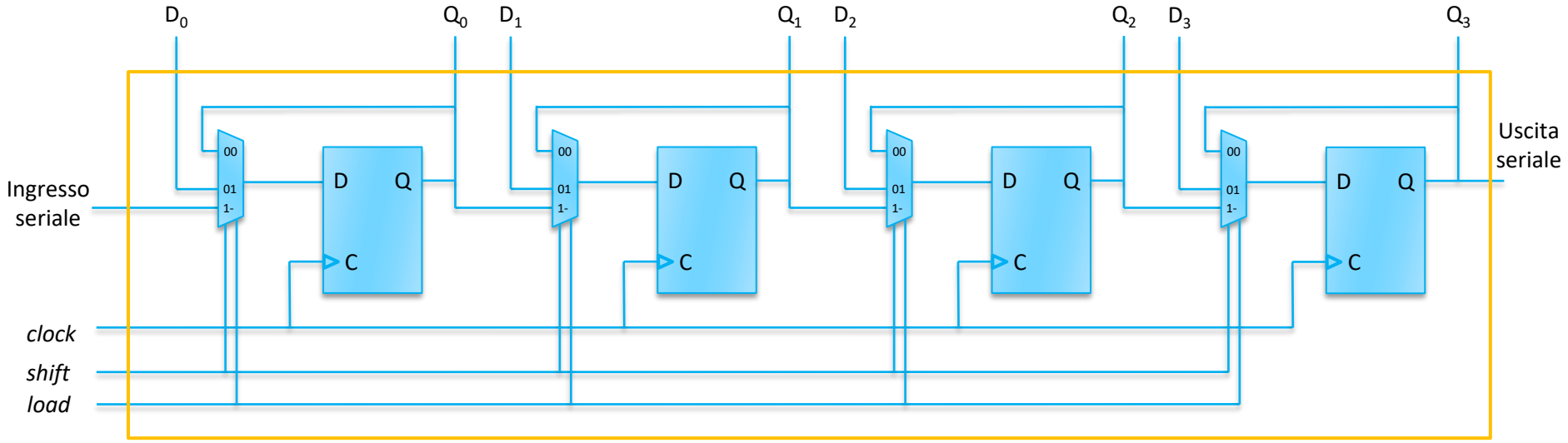
Shift register con ingresso parallelo

- **Facile realizzare un registro a scorrimento con un ingresso parallelo**
  - Basta aggiungere un ingresso al multiplexer
  - Un segnale di *load* indica quando eseguire il caricamento

sh	ld	Modo
0	0	Nessun cambiamento
0	1	Caricamento parallelo
1	-	Scorrimento in avanti



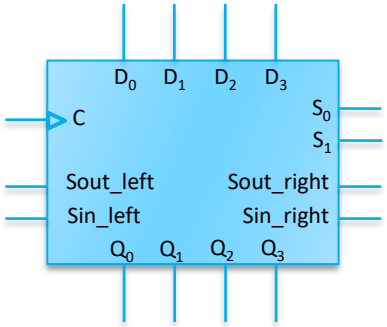
Shift register a 4 bit con entrata parallela



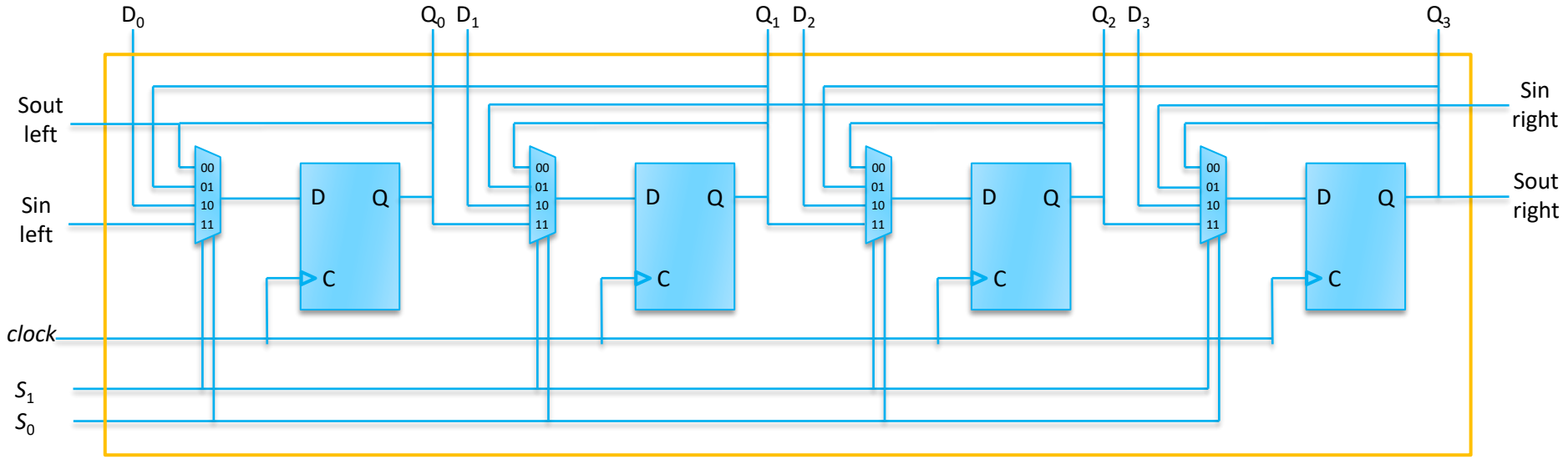
Shift register bidirezionale

- **Semplice gestire le due direzioni tramite un comando di selezione**
  - E' sufficiente usare un quarto ingresso del multiplexer

S_1	S_0	Modo
0	0	Nessun cambiamento
0	1	Scorrimento indietro
1	0	Caricamento parallelo
1	1	Scorrimento avanti



Shift register a 4 bit bidirezionale



Uso dei registri a scorrimento

- **Utili per comunicazione seriale**
  - Si carica un dato in parallelo e lo si trasmette in forma seriale su un solo filo
  - Dall'altra parte si riceve dal singolo filo, e lo si ritrasforma in parallelo
  - Esempio: RS232, USB, SPI, I2C, e molti altri

Uso dei registri a scorrimento

- **Utili per comunicazione seriale**
  - Si carica un dato in parallelo e lo si trasmette in forma seriale su un solo filo
  - Dall'altra parte si riceve dal singolo filo, e lo si ritrasforma in parallelo
  - Esempio: RS232, USB, SPI, I2C, e molti altri
- **Utili per serializzare le operazioni**
  - Operazioni aritmetiche in forma seriale bit a bit
  - Calcolo di parità o di altri codici
- **Indispensabili nel testing**
  - I registri formano una lunga catena a shift per poter imporre valori o leggerli in sistemi complessi
  - Esempio: JTAG

Il contatore

- **E' un registro che segue una sequenza prescritta di stati all'applicazione di una sequenza di impulsi**
  - Per esempio gli impulsi di clock
  - La sequenza di stati può essere quella dei numeri binari, nel qual caso si parla di contatore binario
- **Ve ne sono molte varianti diverse**
  - Ripple counter asincrono
  - Contatore sincrono
  - Contatore in avanti e all'indietro
  - Sequenze normali o strane
  - Con diversi fondo scala
- **Hanno molteplici utilizzi**
  - Orologi, indirizzamento di memoria, conta eventi, effetti luminosi

Contatore sincrono modulo 16

- **Realizzare una scatola con 4 uscite che conti in binario da 0 a 15 incrementando di 1 ad ogni colpo di clock**
  - Avremo bisogno di 16 stati diversi
  - Codifichiamo ogni stato con il valore del conteggio
  - Scelta naturale, l'uscita sarà uguale allo stato
- **Non ci sono ingressi**
  - A differenza di un circuito combinatorio, uno sequenziale può evolvere anche senza ingressi
  - Le transizioni da uno stato ad un altro saranno incondizionate

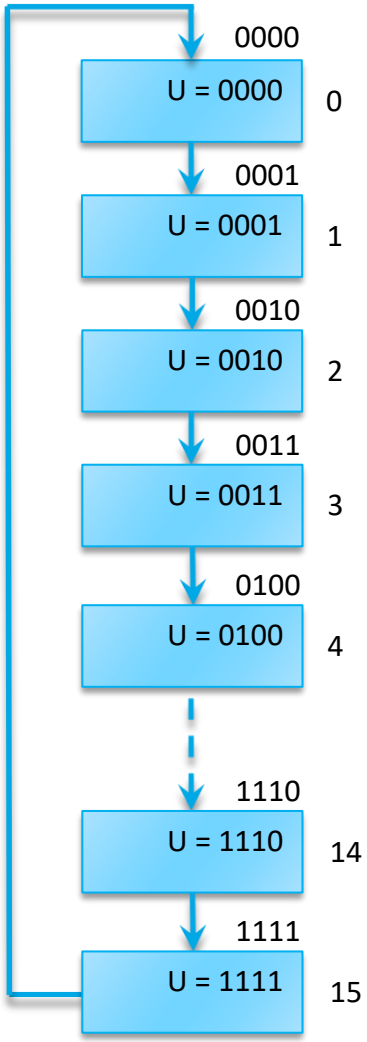


Tabella degli stati

- **Flip flop D**
  - Indichiamo con *abcd* le variabili di stato
  - Ad ogni stato sommiamo 1
  - E' un po' come fare un sommatore
  - Che somma sempre 1

Stato presente				*Stato futuro*			
a	*b*	*c*	*d*	*a*	*b*	*c*	*d*
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

Mappe di Karnaugh

ab \ *cd*	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	1	1	0	1
10	1	1	1	1

$D_a = ab' + ac' + ad' + a'bcd$

ab \ *cd*	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$D_c = c'd + cd'$

ab \ *cd*	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	1	1	0	1
10	0	0	1	0

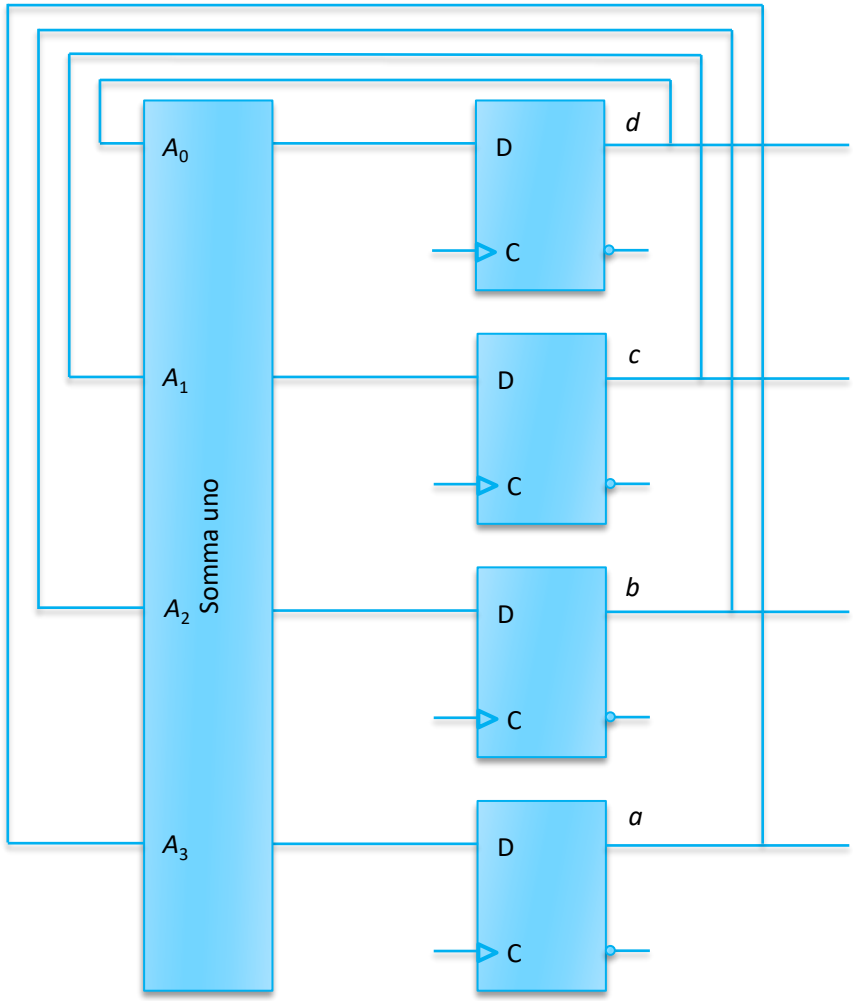
$D_b = bc' + bd' + b'cd$

ab \ *cd*	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

$D_d = d'$

Circuito

- **Possiamo implementare le espressioni che abbiamo derivato**
  - Otteniamo circuiti a due livelli per ciascun ingresso dei flip flop
- **Oppure si può più semplicemente usare un sommatore**
  - Si può usare un sommatore apposta che somma 1 al numero binario in ingresso



Circuito

- **Possiamo implementare le espressioni che abbiamo derivato**
  - Otteniamo circuiti a due livelli per ciascun ingresso dei flip flop
- **Oppure si può più semplicemente usare un sommatore**
  - Si può usare un sommatore apposta che somma 1 al numero binario in ingresso
  - O si può usare un sommatore normale, al quale si mette il valore binario 1 su un ingresso, e l'uscita dei flip flop sull'altro
  - Si può usare un sommatore ripple carry o un carry look-ahead a seconda della velocità richiesta e dimensione del contatore

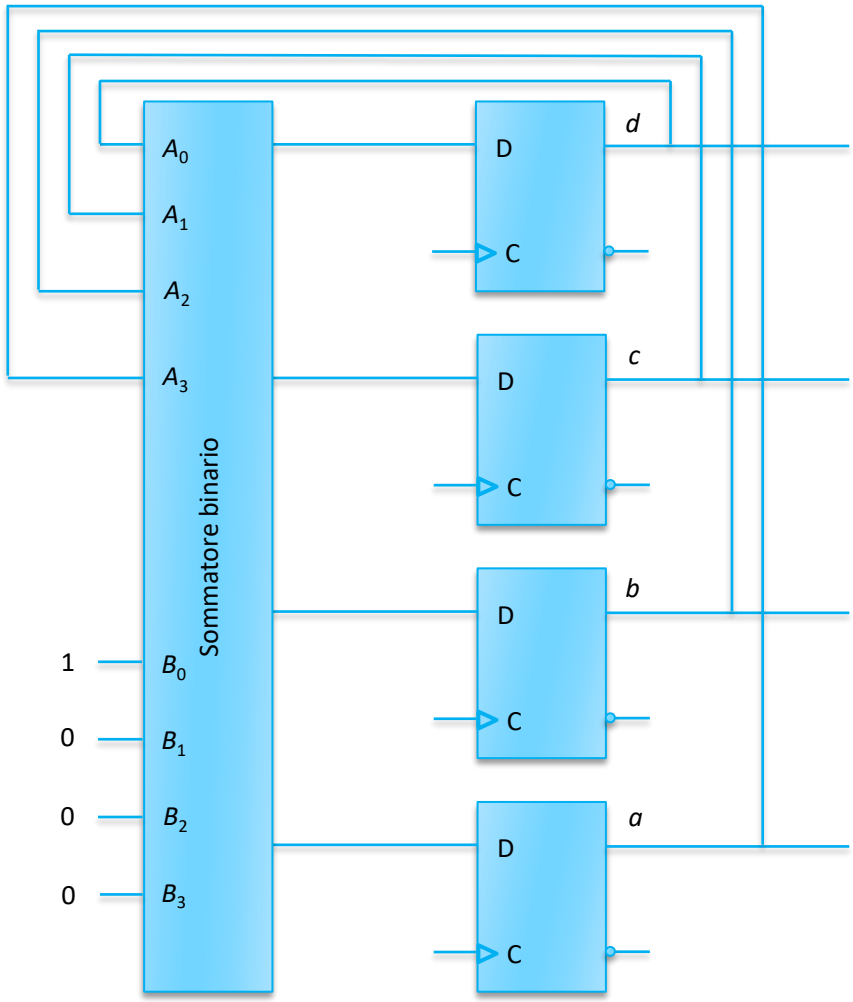
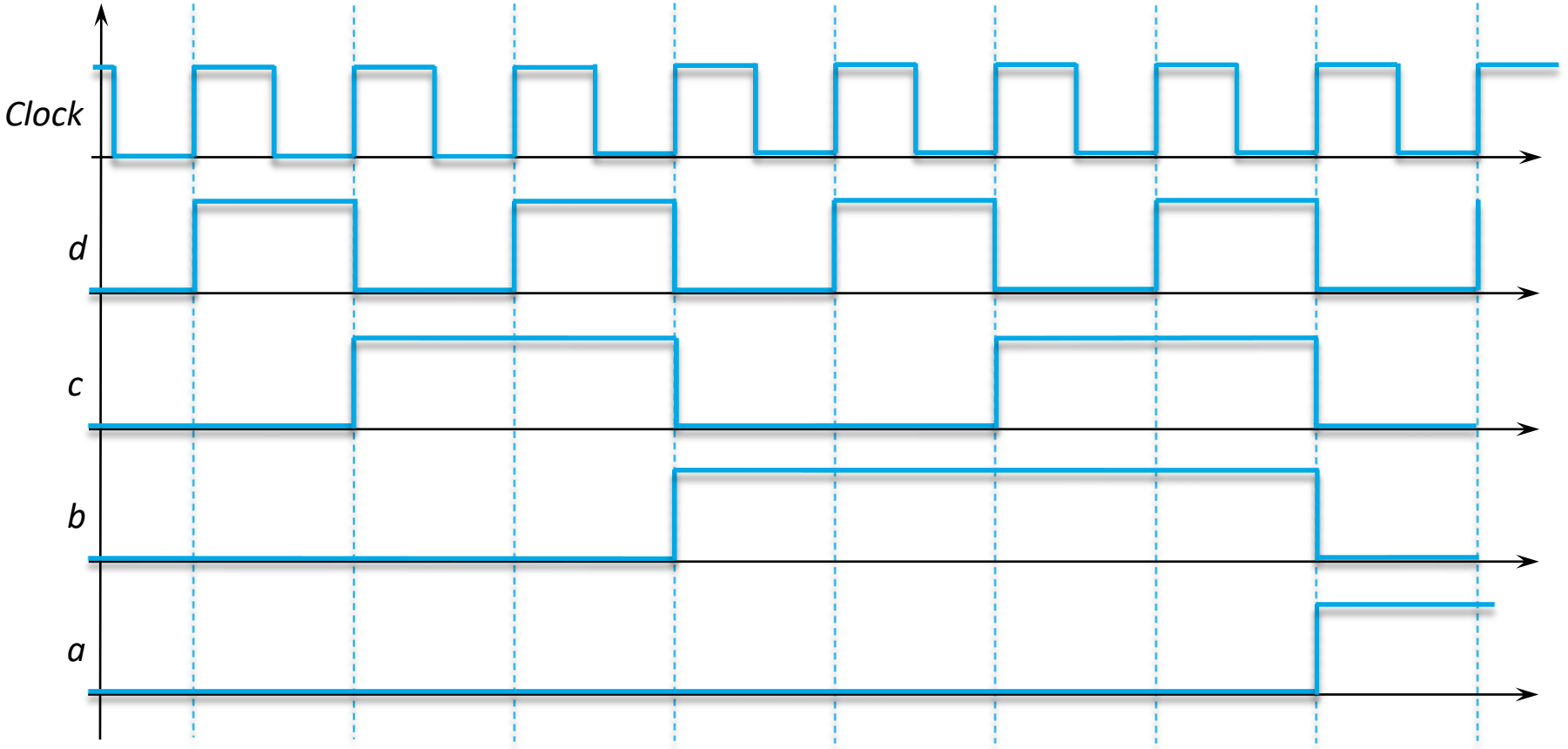


Diagramma temporale



- **Funziona anche da divisore di frequenza**
  - Se il clock è a frequenza *f*, allora *d* ha frequenza *f* /2, *c* ha frequenza *f* /4, *b* ha frequenza *f* /8, etc.

Contatore modulo 10 (BCD)

- **Realizzare una scatola che conti in binario da 0 a 9 incrementando di 1 ad ogni colpo di clock**
  - Come il precedente ma occorrono solo 10 stati
  - Come prima codifichiamo lo stato con il valore del conteggio, per semplificare l'espressione dell'uscita
- **Variabili di stato**
  - Occorrono comunque 4 bit per codificare 10 configurazioni diverse
  - Alcuni degli stati non compaiono nel diagramma, quindi non sono utilizzati

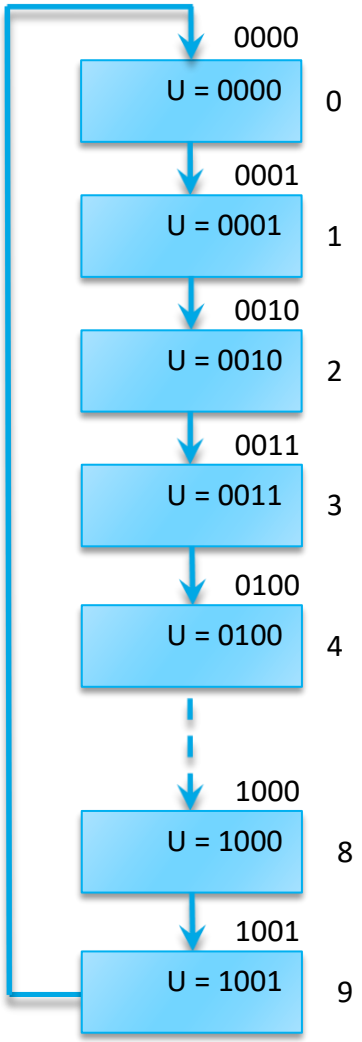


Tabella degli stati

- **Molti don't care in tabella**
  - Per gli stati non utilizzati non ci interessa lo stato futuro
  - E neanche le uscite, che però in questo caso corrispondono con lo stato presente

Stato presente				*Stato futuro*			
a	*b*	*c*	*d*	*a*	*b*	*c*	*d*
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	-	-	-	-
1	0	1	1	-	-	-	-
1	1	0	0	-	-	-	-
1	1	0	1	-	-	-	-
1	1	1	0	-	-	-	-
1	1	1	1	-	-	-	-

Mappe di Karnaugh

ab \ cd	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	-	-	-	-
10	1	0	-	-

$D_a = ad' + bcd$

ab \ cd	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	-	-	-	-
10	0	0	-	-

$D_c = a'c'd + cd'$

ab \ cd	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	-	-	-	-
10	0	0	-	-

$D_b = bc' + bd' + b'cd$

ab \ cd	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	-	-	-	-
10	1	0	-	-

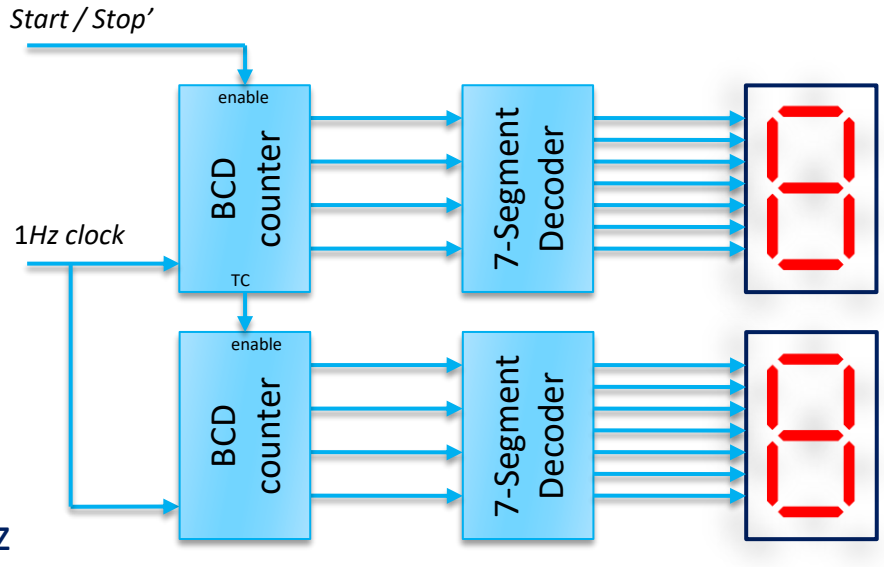
$D_d = d'$

Per casa: disegnare il circuito

Uso del contatore BCD

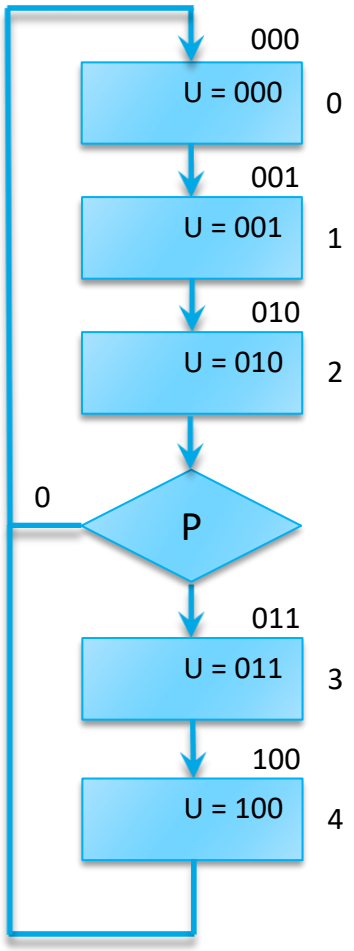
- **Il contatore BCD è utile per contare direttamente in decimale**
  - La codifica BCD è nata per trattare numeri decimali, codificando in binario ogni cifra decimale
- **Aggiungendo una uscita di Terminal Count ed un ingresso di Enable si possono mettere contatori BCD in cascata**
  - E' allora semplice fare dei contasecondi

- **Per casa**
  - Provate a progettare un orologio con ore, minuti e secondi
  - Usate eventualmente contatori con caricamento parallelo e attenzione a quando si commuta
    - Per esempio i secondi vanno da 59 a 00
    - Lo stesso per i minuti
    - Per le ore si passa da 23:59:59 a 00:00:00
  - Ipotizzate di avere un clock a 16.384 kHz
  - Che succede se uso contatori non BCD?



Contatore binario modulo 3 e 5

- **Realizziamo un contatore che conti**
  - modulo 3 se l'ingresso P è a 0
  - modulo 5 se l'ingresso P è a 1
- **Abbiamo bisogno di 5 stati**
  - Usiamo una codifica a 3 bit
  - Codifichiamo lo stato con le variabili *abc*
- **Possiamo realizzarlo con il metodo tradizionale**
  - Costruiamo la tabella degli stati, che include sia stato presente sia ingressi
  - Costruiamo le mappe di Karnaugh
- **Un metodo alternativo ci produce mappe a variabili riportate**
  - Riportiamo tutti gli ingressi e teniamo solo lo stato



Altre varianti

- **Contatore binario up-down**
  - Si include un comando che consente di scegliere se contare in avanti o all'indietro
  - Si può realizzare con il sommatore sottrattore visto discutendo dei circuiti aritmetici
- **Contatore binario con ingresso parallelo**
  - Un segnale di *load* consente di inserire un valore arbitrario nel contatore
  - Si realizza facilmente mettendo un multiplexer tra le uscite del sommatore ed i flip flop, come visto per i registri
  - Occorre disabilitare il TC durante la fase di caricamento
- **Contatore binario con ingresso offset**
  - Invece di sommare sempre 1, si somma un valore di offset quando un segnale abilita l'operazione
  - In pratica basta usare un sommatore tradizionale
  - Provate a realizzarne lo schema a blocchi
- **Contatori a sequenza arbitraria**
  - Si specifica la sequenza sul diagramma degli stati

Take away

- **Registri e contatori sono la base dei circuiti sequenziali**
  - Li usiamo come blocchi già fatti per costruire circuiti più complessi
  - Formano le strutture di dati alla base dei circuiti
- **Varianti infinite**
  - Potete invertavene di nuove combinando le varie funzioni (abilitazioni, up/down, carico parallelo e seriale, etc.)
- **La parte di intelligenza invece non può essere codificata**
  - Occorre sviluppare un diagramma degli stati specifico
  - Forma la logica di controllo o diagramma di flusso
  - Lo utilizzeremo per controllare il funzionamento dei blocchi base sequenziali e combinatori

Pattern sequenziali

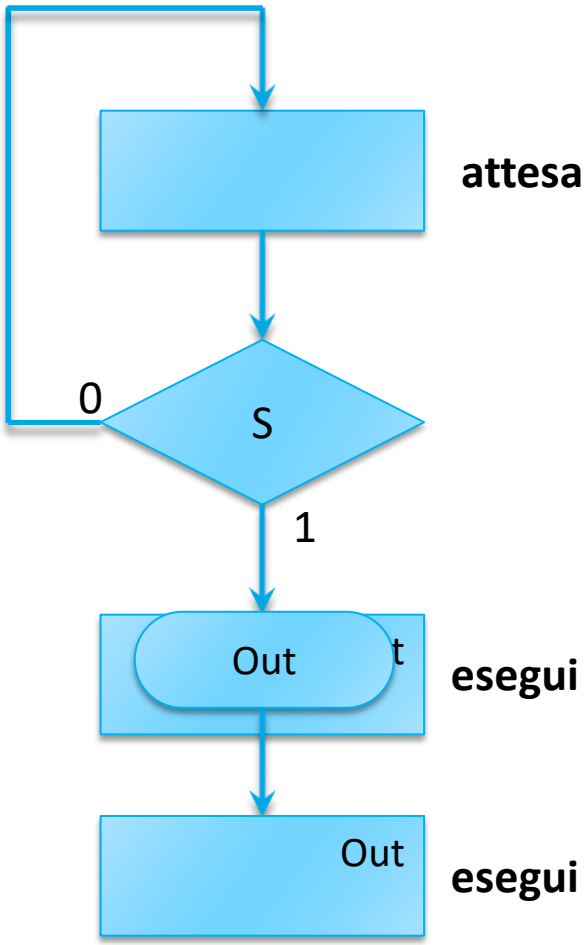
Le cose che capitano di solito

Pattern

- **Alcune configurazioni di macchine a stati capitano ripetutamente**
  - E’ utile conoscerle ed utilizzarle quando servono
  - Sapete già come funzionano, non c’è bisogno di pensarci troppo

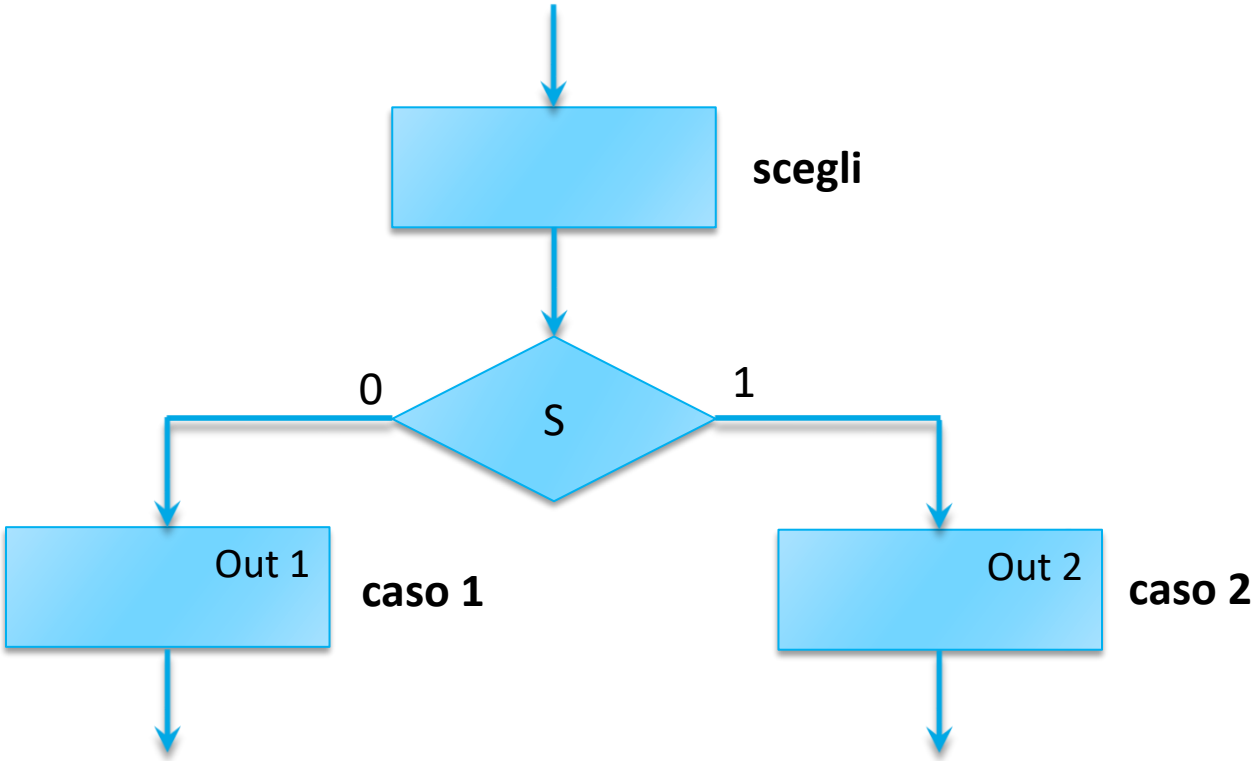
Busy waiting

- **Si aspetta che un certo segnale S esegua una transizione, per esempio da 0 a 1**
  - Notare che si passa nel nuovo stato durante il ciclo di clock *successivo* a quello in cui S esegue la transizione
  - La reazione ha quindi un ritardo
  - Si può eventualmente reagire subito tramite una uscita condizionata

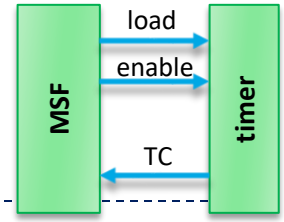


Choice

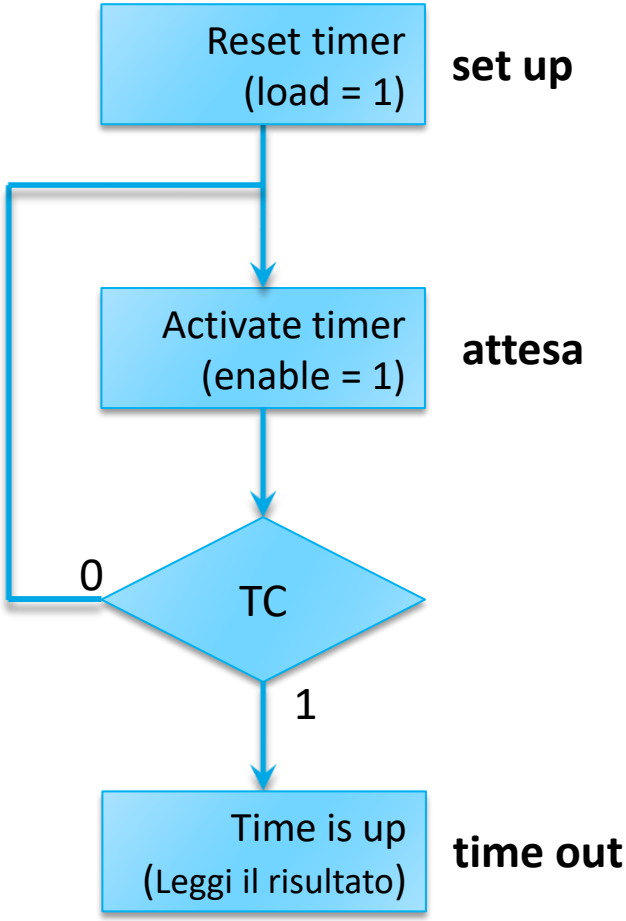
- **Si segue un percorso piuttosto che un altro a seconda del valore di un segnale S**



Fire and check

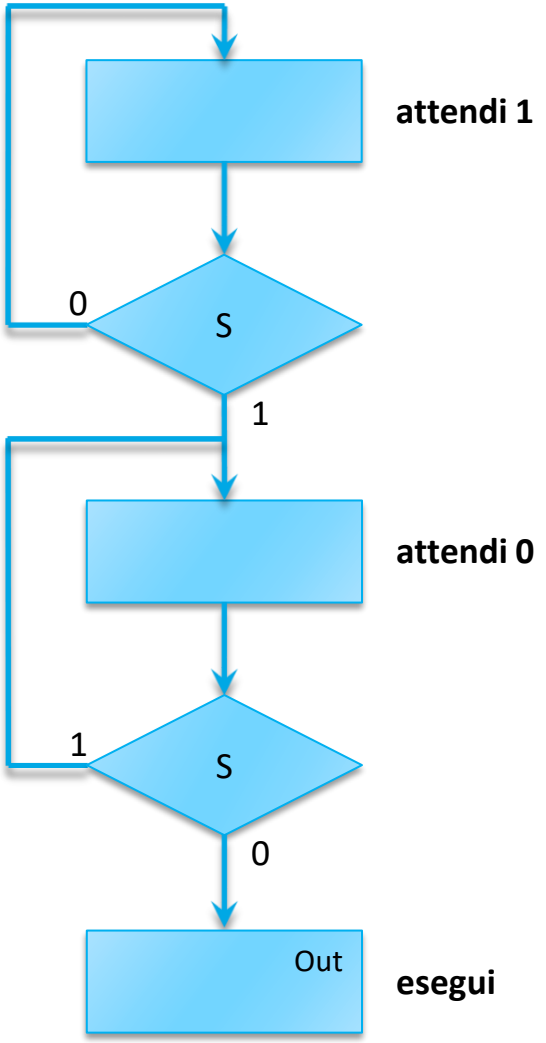


- **Si fa partire un calcolo (per esempio un contatore) e poi si attende che questo finisca**
  - In questo modo si evita ad esempio di ricostruire contatori esplicitamente ogni volta che servono
  - Con 3 stati si risolve il problema, qualunque sia il tipo di conteggio
  - Utile anche per operazioni aritmetiche che richiedono tempo, come ad esempio il floating point



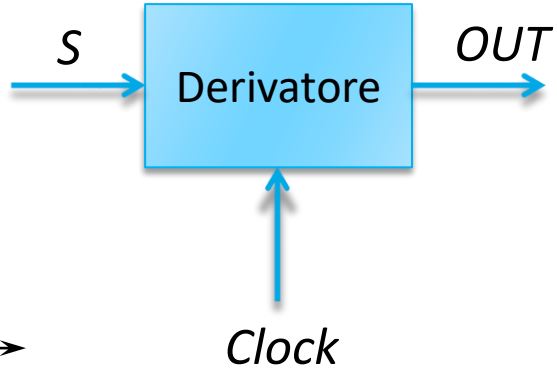
Edge detector

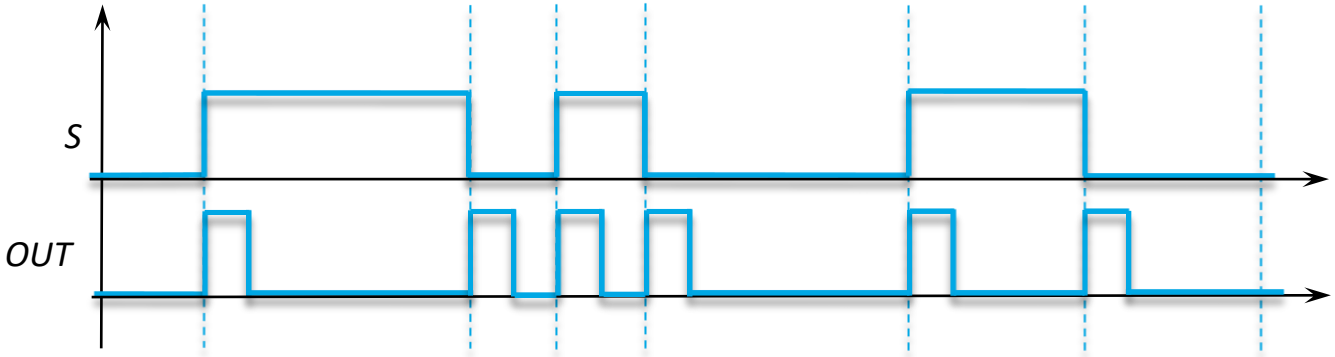
- **Identifica i fronti di discesa (o di salita) di un segnale S di ingresso**
  - Occorre prima aspettare che S vada a 1
  - Quando S torna a 0 si segnala la condizione
- **Due attese in sequenza**
  - La prima attende sullo 0, la seconda attende sull'1
  - Uscite condizionate possono anticipare la reazione



Derivatore

- **Si segnalino con un valore 1 in uscita tutti i fronti dell'ingresso**
  - Ingresso S ed uscita OUT
  - Supponiamo che l'ingresso sia inizialmente a 0
  - Circuito sincrono





Derivatore

- **Manteniamo l'uscita a 1 per un solo ciclo di clock ad ogni transizione dell'ingresso**
  - Altrimenti, se non la riportiamo a 0, rimarrebbe sempre a 1
- **Aspettiamo le transizioni con un busy wait ciascuna**
  - Separate con uno stato in cui l'uscita è attiva
  - Totale 4 stati che richiedono due variabili di stato

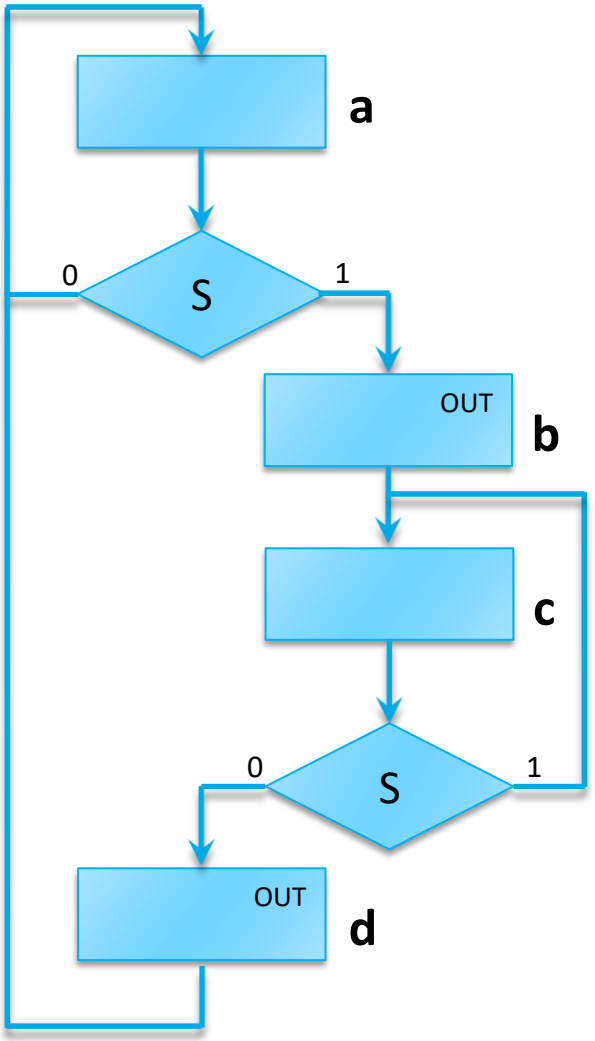
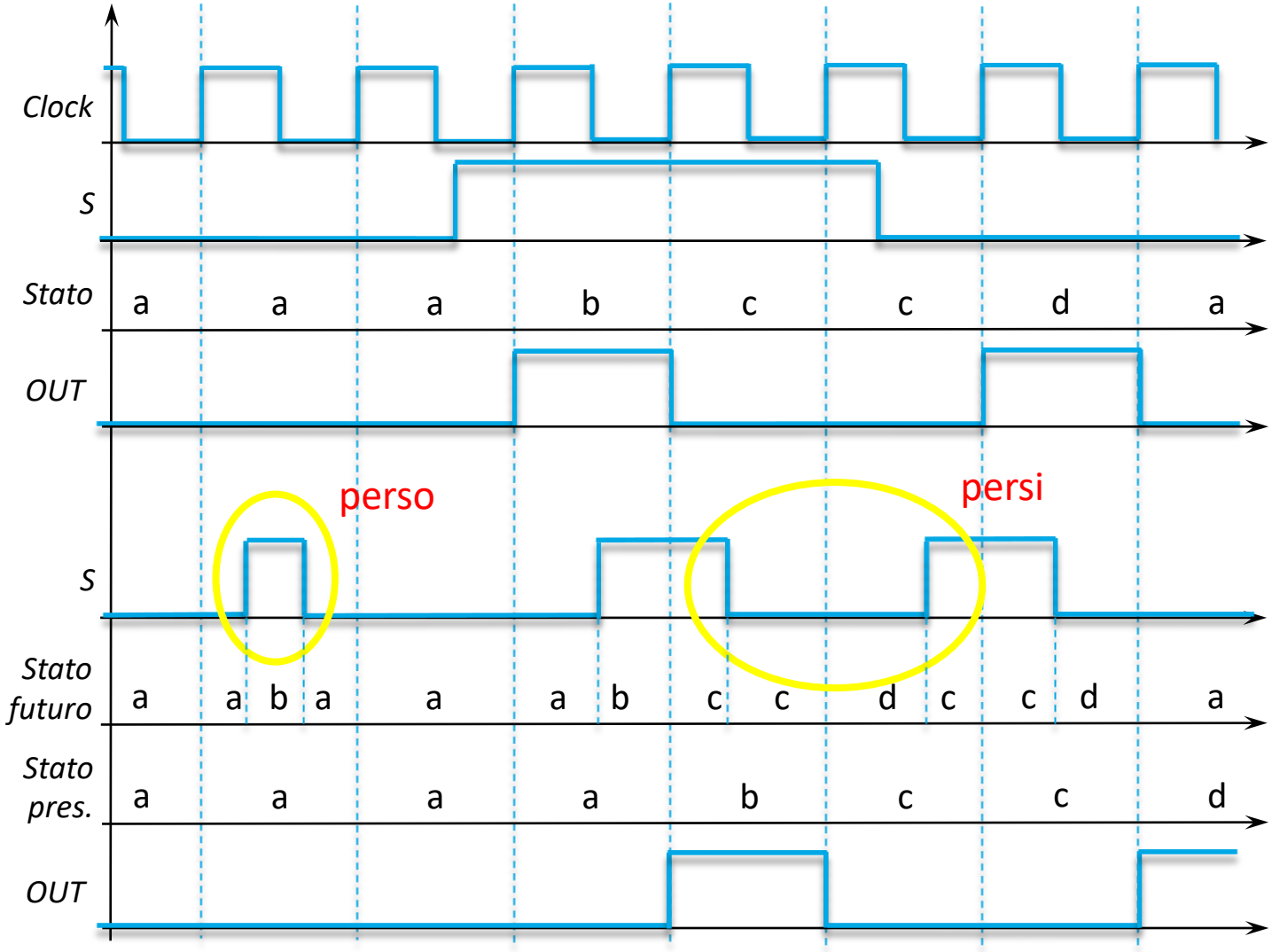
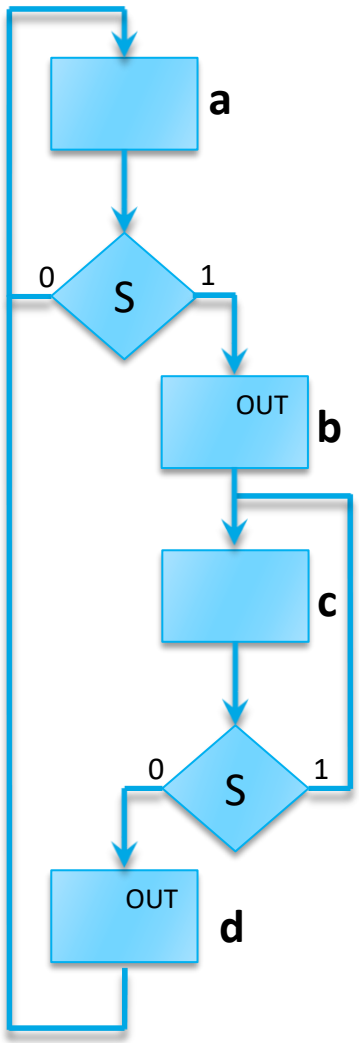


Diagramma temporale

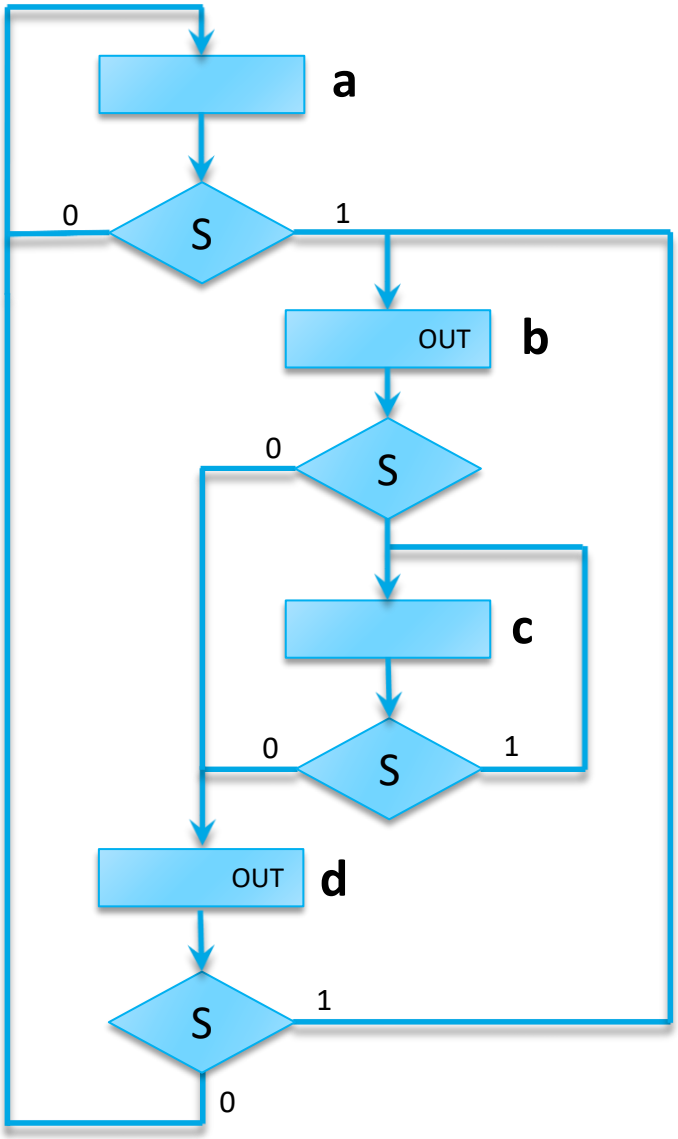




Caratteristiche del derivatore

- **Lo abbiamo realizzato come una macchina di Moore**
  - Semplice gestire le uscite, che dipendono solo dallo stato
- **La macchina di Moore reagisce con una certa lentezza**
  - Al limite può anche perdere delle transizioni
  - Occorre che il periodo del clock sia molto più piccolo del minimo tempo di permanenza dell'ingresso S ad uno dei due valori logici
  - In questo modo abbiamo un ritardo di al più un colpo di clock
- **Alcune transizioni vanno perse**
  - Se l'impulso dura meno di un colpo di clock la macchina potrebbe non accorgersene
  - Gli stati **b** e **d** non sono sensibili alle transizioni

Una soluzione



- **Guardiamo il valore di S anche dagli stati b e d**
  - Provate a fare il diagramma temporale

Derivatore con uscite condizionate

- **Poniamo l'uscita subito a 1 non appena rileviamo la transizione in ingresso**
  - Occorre una uscita condizionata
  - Quindi entriamo direttamente nel ciclo di attesa del fronte successivo
  - In questo modo non sprechiamo tempo
  - Nota: l'uscita condizionata NON è uno stato!!!
- **Cambio di specifiche**
  - Adesso l'uscita non è detto che rimanga ad 1 per almeno un colpo di clock

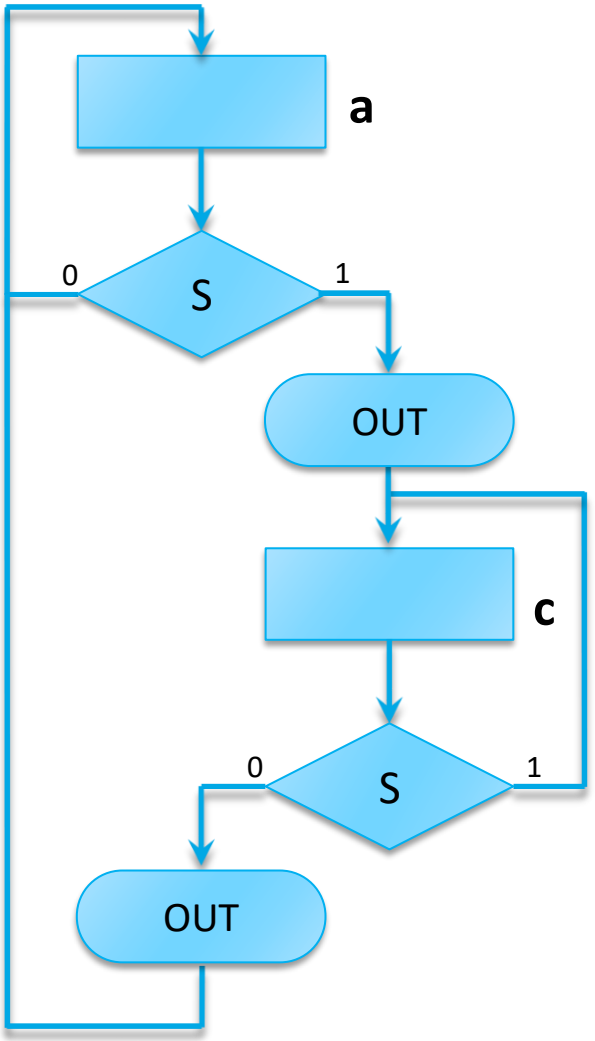
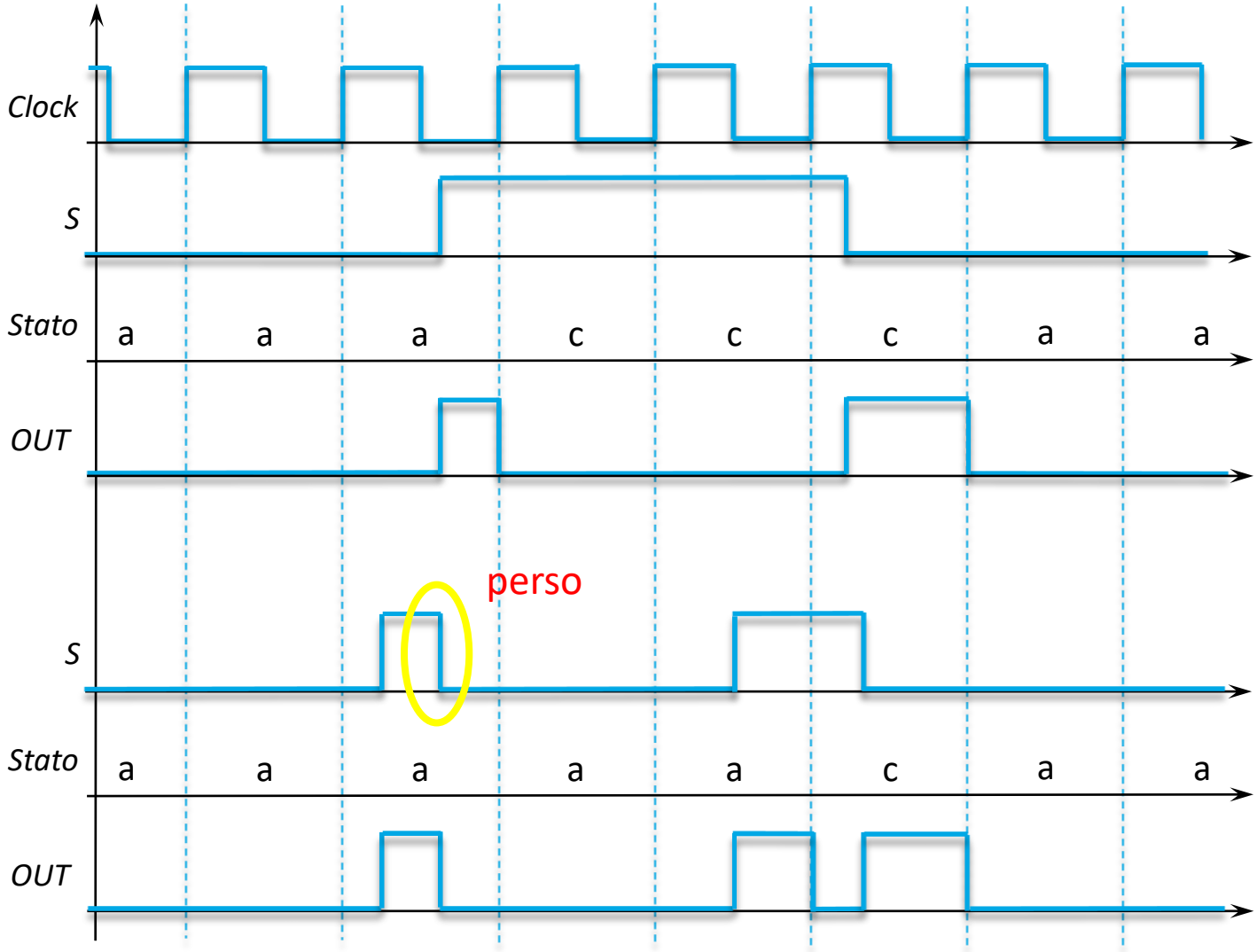
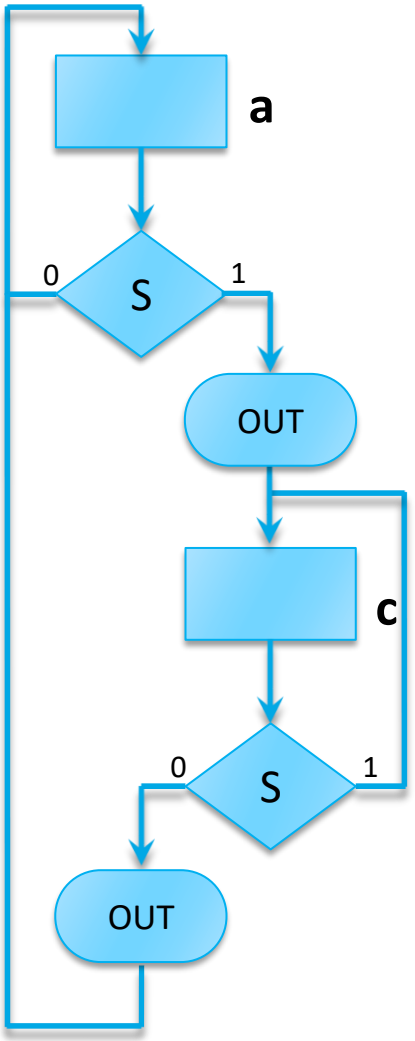


Diagramma temporale





Realizzazione versione di Moore

y \ x	0	1
0	a	d
1	b	c

Codifica degli stati

y \ x	0	1
0	0	1
1	1	0

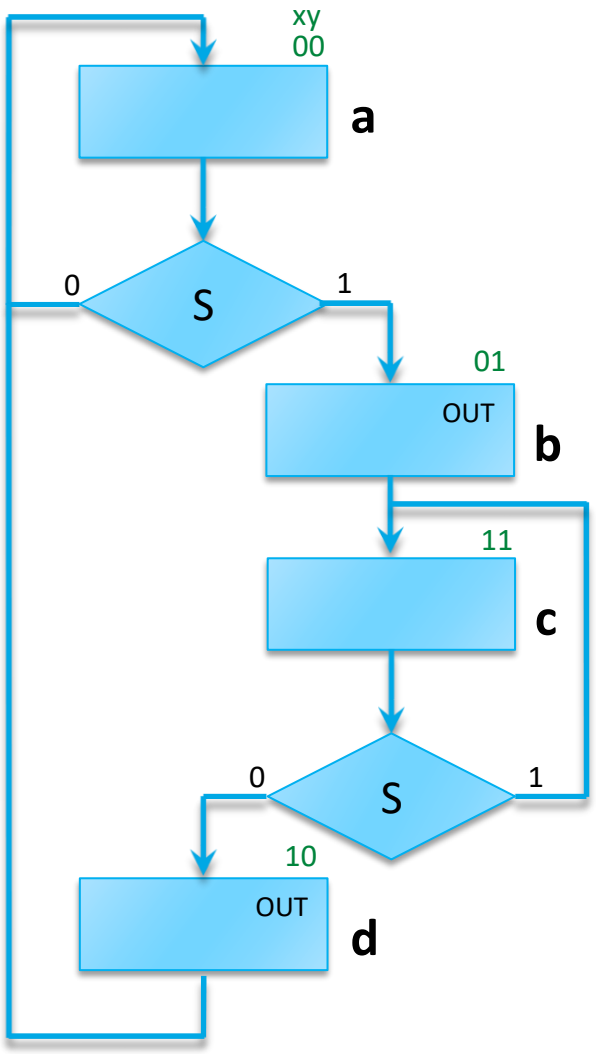
OUT = x ⊕ y

y\Sx	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$D_x = y$

y\Sx	00	01	11	10
0	0	0	0	1
1	1	0	1	1

$D_y = x'y + S(x' + y)$



Realizzazione versione di Mealy

x	0	1
	a	c

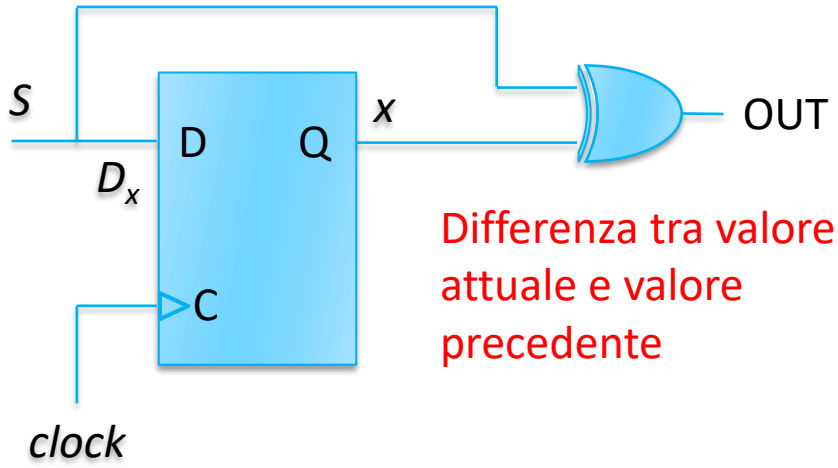
Codifica degli stati

S/x	0	1
0	0	0
1	1	1

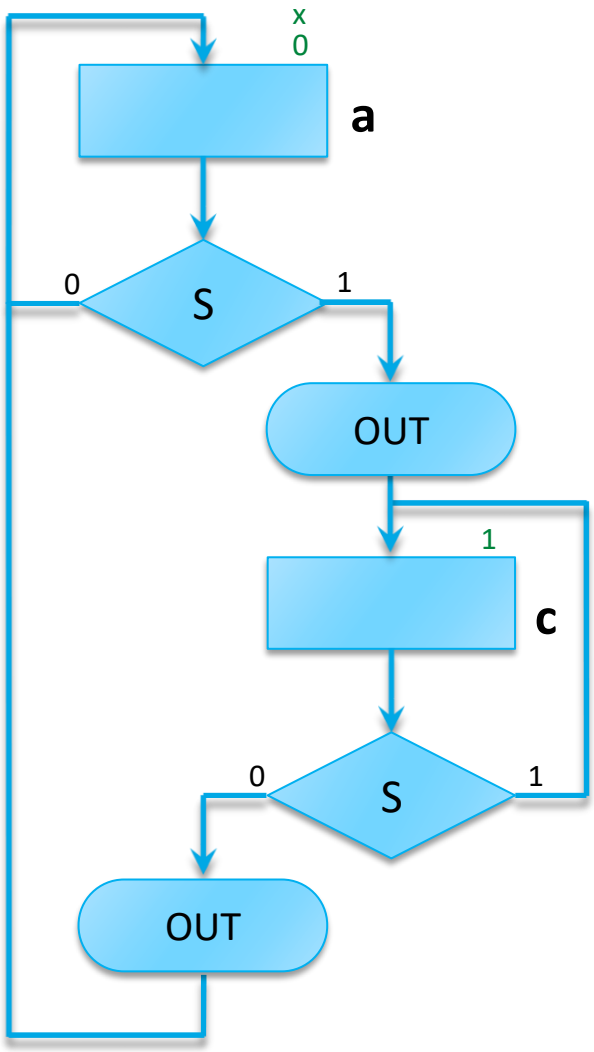
$D_x = S$

S/x	0	1
0	0	1
1	1	0

$OUT = x \oplus S$



Differenza tra valore attuale e valore precedente



Take away

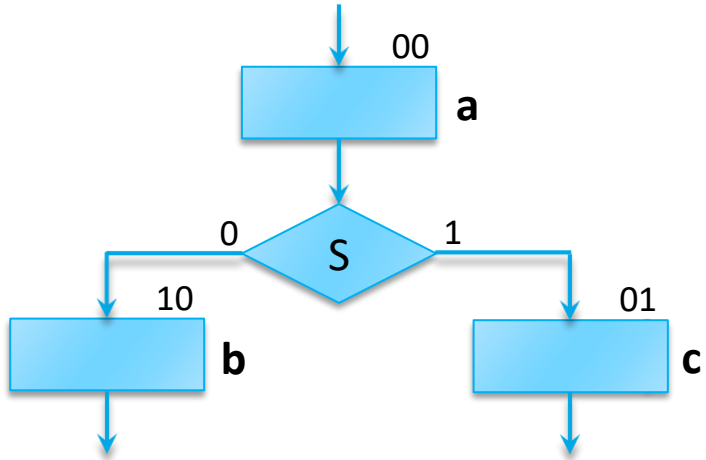
- **Spesso le cose si ripetono**
  - Vari pattern e configurazioni si utilizzano in molte occasioni
  - Utile combinarli per ottenere comportamenti più complessi
- **Pensare in maniera algoritmica**
  - Immaginate di sequenzializzare le operazioni
  - Più o meno come scrivere software
  - Però con la possibilità di operare in parallelo

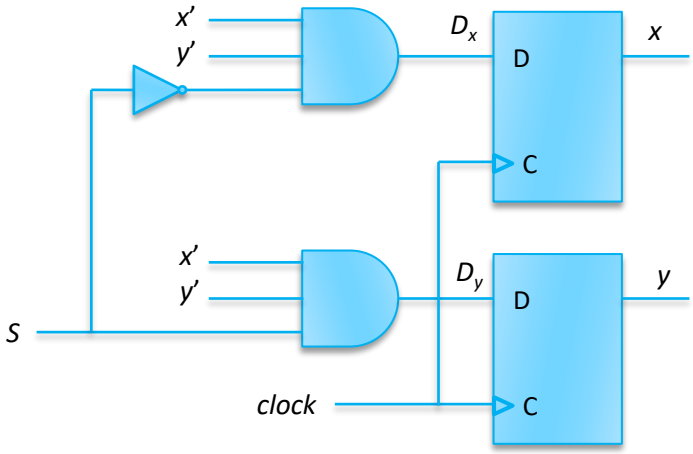
Corse critiche sugli ingressi

- **I circuiti sincroni non hanno corse sugli stati**
  - Perché tutto è bloccato dal clock
- **Gli ingressi primari però non sono necessariamente sincronizzati con il clock**
  - Un bottone potrebbe essere premuto in ogni momento
  - L'effetto delle transizioni sugli ingressi potrebbe allora farsi sentire agli ingressi dei flip flop proprio nei pressi del fronte attivo del clock
  - Per flip flop diversi, il tempo di arrivo potrebbe essere differente

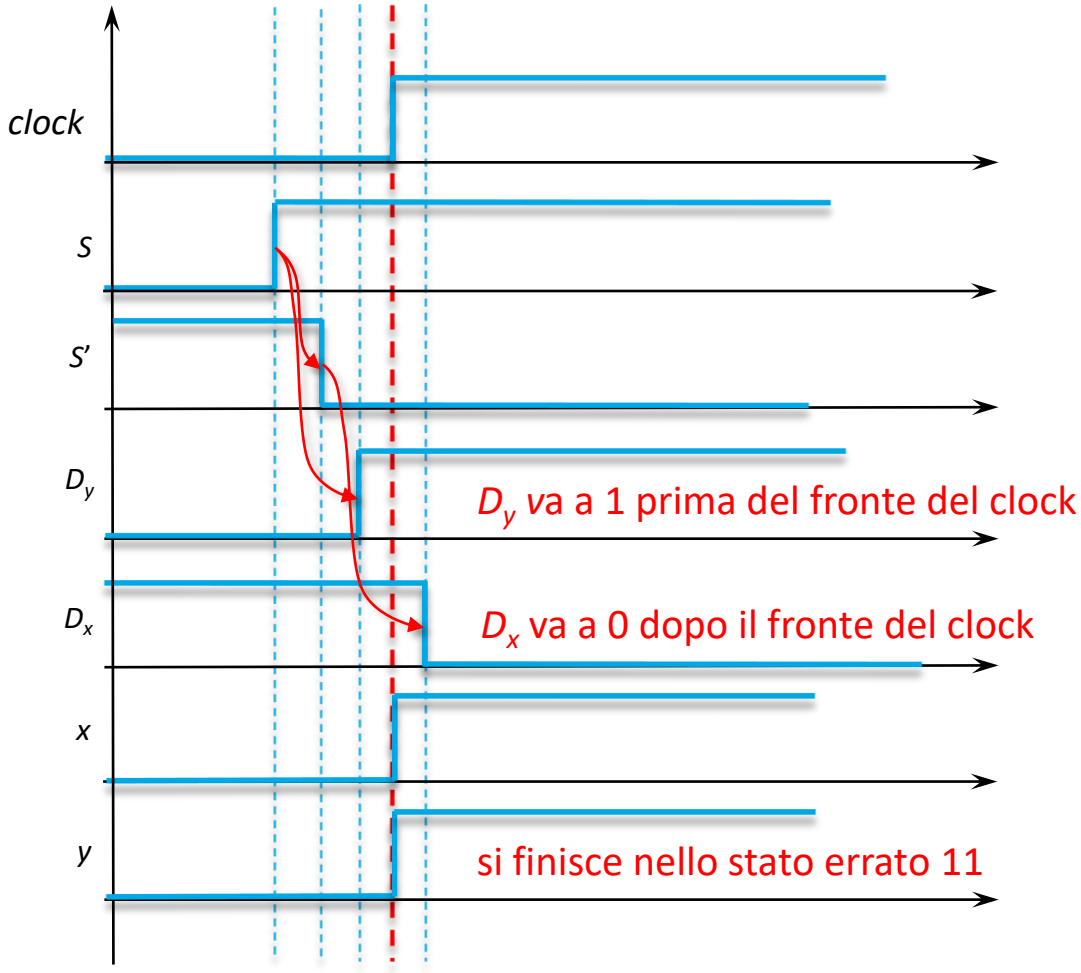
Esempio di corsa sugli ingressi

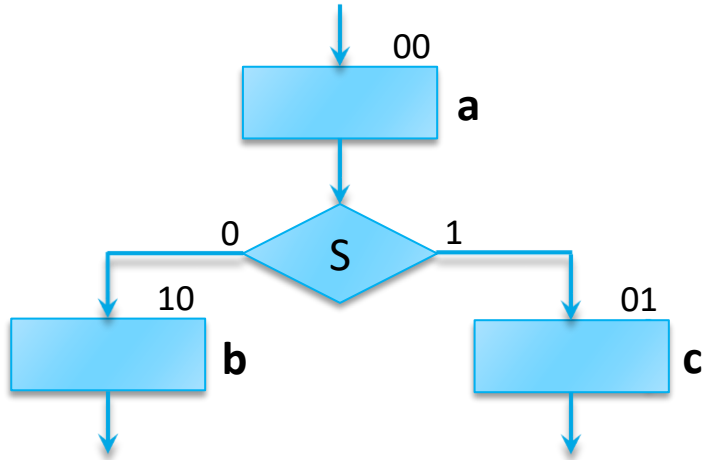
- **Nella configurazione a fianco il segnale S sceglie se far cambiare la variabile *x* o la *y***
  - Facile realizzare il circuito
- **Supponiamo che *S* commuti nelle vicinanze del fronte del clock**
  - Il suo effetto può arrivare ad *x* dopo il clock, e ad *y* prima del clock

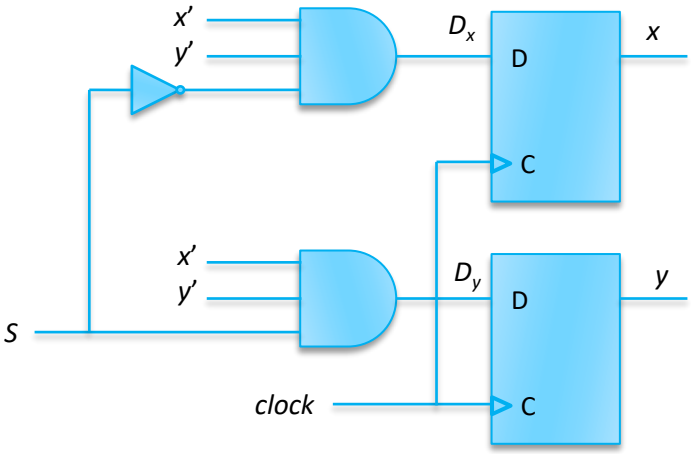




Esempio di corsa sugli ingressi

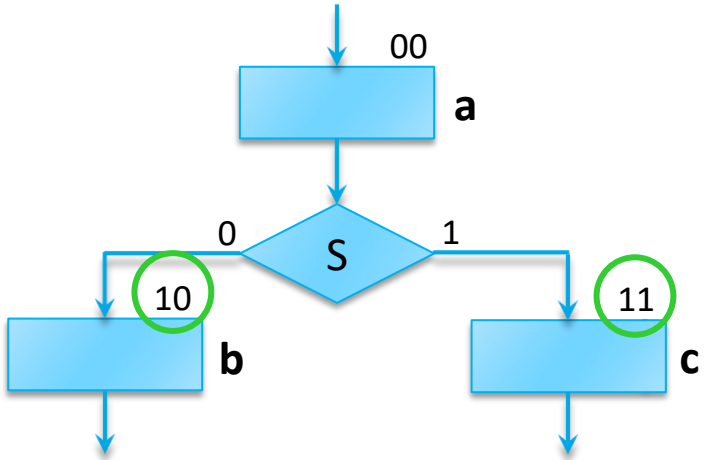






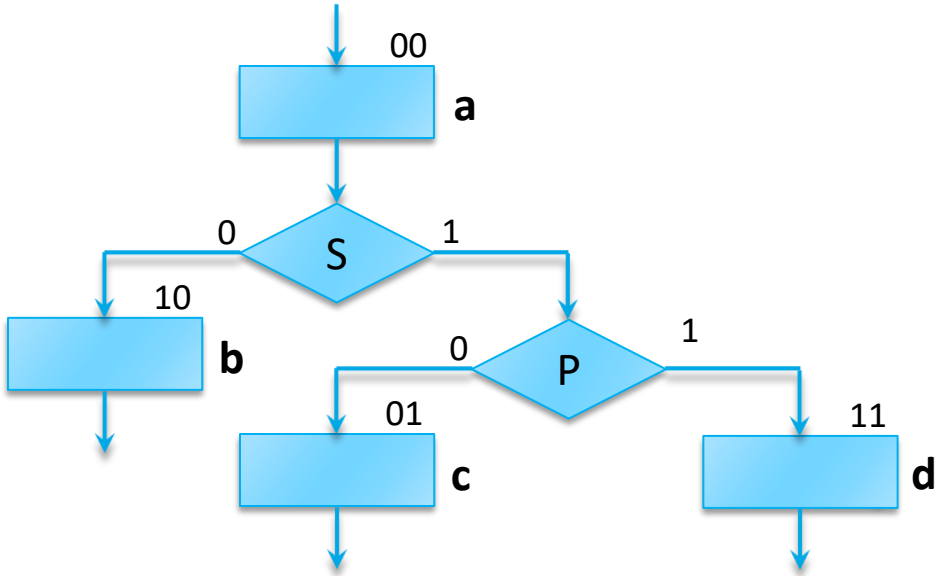
Soluzioni

- **E' sufficiente assegnare a tutti gli stati che possono essere raggiunti da a dei codici che tra loro differiscano di una sola variabile**
  - Non c'è bisogno che il codice di **a** differisca di una sola variabile da quelli destinazione
  - Nel nostro caso possiamo codificare **b** con 10 e **c** con 11
  - Così la D_x viene messa a 1 subito all'inizio del ciclo, perché non dipende più da S, e si finisce in **c**
  - Se l'ingresso è molto vicino al clock, al limite si finisce in **b**
  - Ma non si finisce mai in uno stato che non esiste



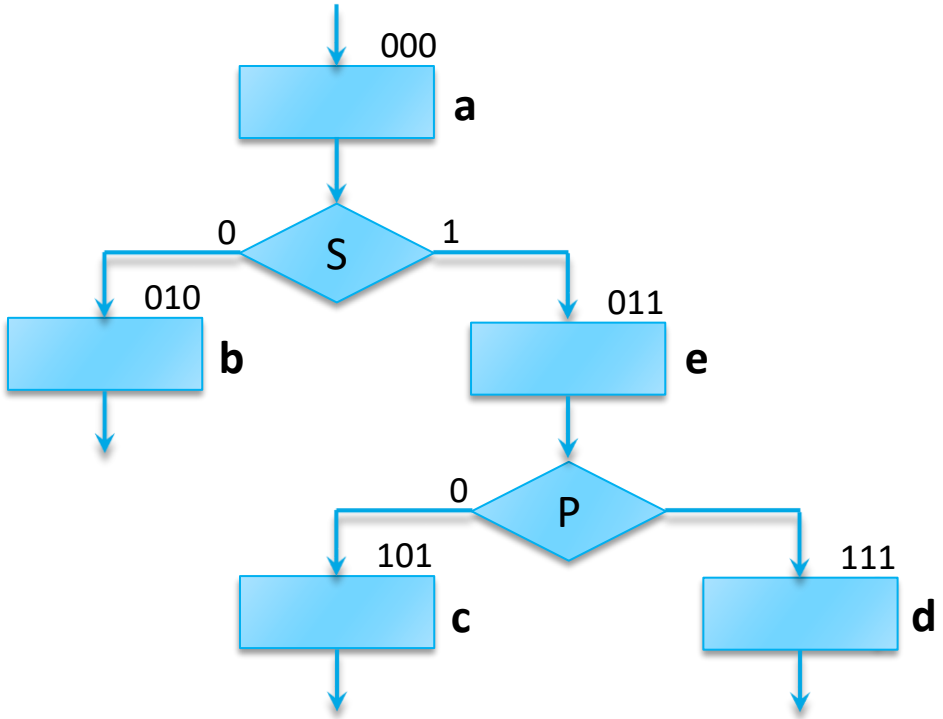
Soluzioni

- **Alle volte non è possibile eseguire la codifica**
  - In questo caso, gli stati **b**, **c** e **d** dovrebbero tutti differire per una sola variabile



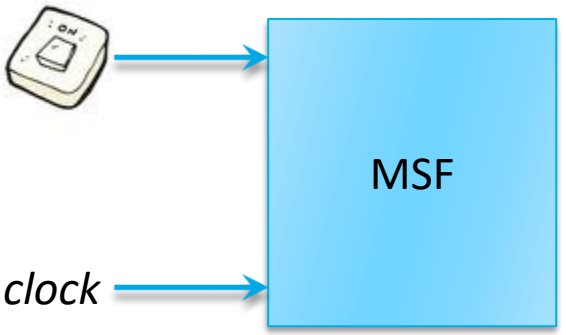
Soluzioni

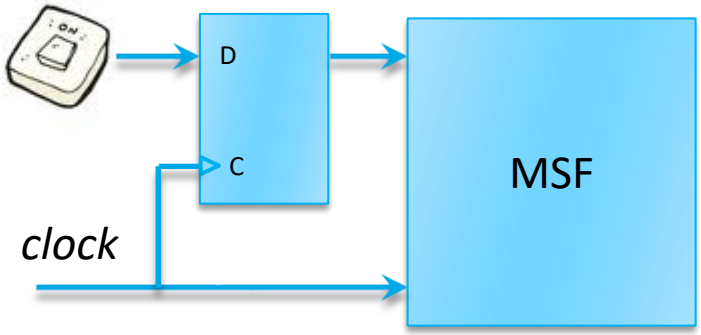
- **Alle volte non è possibile eseguire la codifica**
  - In questo caso, gli stati **b**, **c** e **d** dovrebbero tutti differire per una sola variabile
- **Occorre aggiungere un nuovo stato**
  - Ora **b** ed **e** sono a distanza 1
  - Anche **c** e **d** sono a distanza 1
  - Gli stati sono però ora 5
  - Devo allora aggiungere una variabile di stato
- **Lo stato aggiuntivo introduce un ritardo**
  - Occorre verificare che la funzione sia ancora accettabile



Soluzioni

- **Un altro metodo è quello di eliminare il problema alla radice**
  - Il fenomeno è causato dall'avere ingressi asincroni
  - E' sufficiente allora sincronizzare gli ingressi
  - Basta interporre un flip flop di tipo D tra l'ingresso e la macchina a stati
  - Si introduce comunque un ritardo, ma si risolve il problema delle corse
- **Problemi reali**
  - Queste cose accadono veramente
  - Nei progetti degli anni scorsi, vari problemi sono stati risolti sincronizzando gli ingressi della scheda





Codifica degli stati

- **Vi sono vari obiettivi nel codificare lo stato**
  - Minimizzare il costo della rete combinatoria
  - Minimizzare il numero di transizioni per ridurre i consumi
  - Eliminare le corse critiche
- **I vari criteri possono essere contrastanti**
  - Occorre quindi valutare quale sia l'aspetto più importante
  - Il problema è comunque molto complesso, e lo si affronta solitamente con strumenti automatici di progetto