14 Aprile 2023

ARCHITETTURE DI CALCOLO LEZIONE 9

Reti sequenziali sincrone

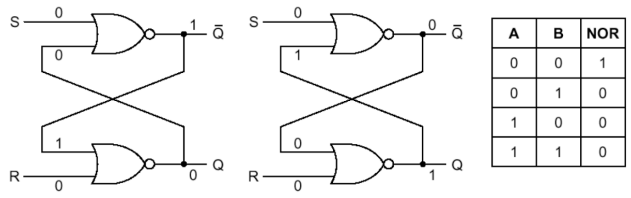
Reti sequenziali

Mentre le reti combinatorie prevedono un output che dipende esclusivamente dall'input, senza ricorrere ad alcun tipo di dato in memoria, le reti sequenziali sono in grado di calcolare funzioni che dipendono anche dallo stato precedente. Tale stato in qualche modo risulta memorizzato nel sistema (elemento di memoria) bit a bit tramite l'uso di una serie di circuiti detti latch e flip-flop.

Come avevo preannunciato dopo aver visto i circuiti combinatori, sostanzialmente avete capito che si tratta di Reading in cui l'output viene sempre soltanto prodotto in quel momento, quindi non c'è modo che circuiti combinatori di legare l'output a uno stato.
Questo perché ?
Perché nei circuiti combinatori non c'è un modo per memorizzare lo stato, non c'è il chip di memoria.
Le reti sequenziali, a differenza delle reti combinatorie, sono in grado di calcolare dei valori di output che dipendono anche da uno stato, in altri termini l'output dipende dall'input e dallo stato precedente, ovviamente lo stato in qualche modo dobbiamo memorizzarlo, quindi dobbiamo introdurre il concetto di unità di memoria, noi abbiamo già visto la struttura dei calcolatori e che internamente ci sono le memorie, le memorie si possono realizzare utilizzando porte logiche combinate tra loro in modo da permettere di memorizzare i singoli bit.
Vedremo come si memorizza un singolo bit e ovviamente le memorie, le memorie che conosciamo sono tabelle di tanti bit, non sono altro che l'unione di tanti singoli bit; quindi se noi capiamo come memorizzare un bit sapremo anche realizzare delle tabelle più complesse.
Capito come memorizzare un bit con dei circuiti, che si chiamano latch e Flip-flop, andremo a capire come un circuito combinatorio, attraverso questi elementi di memoria, sia in grado di definire, tramite delle sequenze temporali, l'output come funzione dell'input della memoria, quindi sostanzialmente dovremo capire due cose che sono molto semplici: • capire come memorizzare i dati, • sfruttare le nostre conoscenze pregresse per sviluppare i circuiti sequenziali (circuiti combinatori, mappe di karnaugh e automi a stati finiti).

Latch S-R

In figura vediamo un latch S-R (Set-Reset), un circuito composto da 2 porte NOR concatenate, che costituisce l'elemento base per costruire elementi di memoria più complessi (memorizzare singoli bit a formare un elemento a più bit).

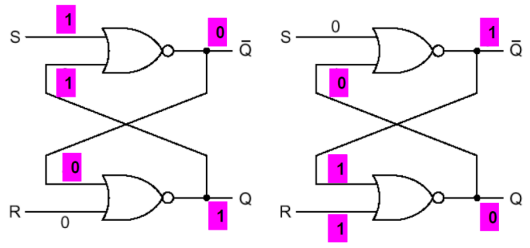


A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

- L'S-R Latch è un circuito, composto da 2 porte NOR concatenate, che costituisce l'elemento base per costruire elementi di memoria più complessi
  - S sta per Set e R sta per Reset

Tale circuito permette essenzialmente di memorizzare il valore 1 o 0 di Q a patto che R ed S siano entrambi 0.
Le possibili combinazioni S,R sono 4 ma quelle utilizzabili sono solo 3; infatti, la combinazione (1,1) non deve mai essere presentata al latch in quanto porterebbe in primis ad una

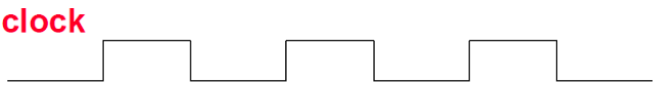
nullità di entrambi i valori e, successivamente, si avrebbe una fase d'imprevedibilità dovuta ai ritardi dei gate.



A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

•Set=1 memorizza (set) il valore di Q ad 1, ed il valore viene mantenuto anche quando S viene riportata a 0.
•Reset=1 memorizza il valore di Q a 0.

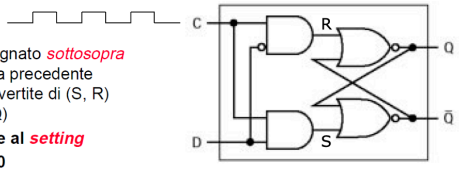
I segnali S e R devono essere stabili, e valere (1,0) o (0,1) per poter memorizzare il valore corretto; tuttavia, essendo tali segnali il risultato di un circuito combinatorio legato a dei ritardi, l'output diventa stabile dopo un certo intervallo di tempo. È possibile calcolare questo intervallo di tempo, che dipende dal numero di porte attraversate e dal ritardo delle porte. Bisogna evitare che durante questo intervallo gli output intermedi del circuito vengano presentati al latch per la memorizzazione (altrimenti possono essere memorizzati valori errati).
La soluzione a tale problema è il clock, un segnale a gradino (immagine a destra) il cui periodo, ovvero l'intervallo di tempo, viene scelto abbastanza grande da assicurare la stabilità degli output del circuito. Il clock è, quindi, usato per abilitare la scrittura nei latch e determina il ritmo dei calcolatori e delle relative operazioni di memorizzazione.



Il primo elemento di studio è il castello del latch, questo circuito che vedete qui dentro è lo stesso circuito con dei valori di input differenti, questo è il più semplice circuito che possiamo utilizzare per memorizzare dei bit, questo in particolare è il latch SR la S è la iniziale di set mentre la R di reset, quindi si capisce che questi due terminali essere S ed R sono utilizzati per settare e resettare il valore.
Questo circuito è un pò particolare, concentratevi su questo sinistra noterete che l'output del della porta logica in alto, finisce in relazione come input della porta logica in basso, così come si noterà che l'output della porta logica in basso finisce come input della porta logica in alto.
Sostanzialmente questi due output che sono chiamati Q negato e Q, chiamati cosi perché questi due output sono sempre uno l'opposto dell'altro, questi output servono anche come input per la porta logica opposta, adesso quello che dobbiamo cercare di capire è come fa il circuito a memorizzare un valore, anzitutto questa è una porta logica NOR la quale il risultato della porta logica OR, quindi sostanzialmente restituisce l'opposto del risultato di una porta OR (restituisce 1 solo se entrambi gli input sono pari a 0),quindi queste porte che vedete qua funzionano secondo la tabella verità qui riportata.
Adesso proviamo a vedere questo schema sopra, ipotizziamo in questo schema sopra che noi stiamo dando come input 0 e 0, cioè S pari a zero ed R pari a 0, ipotizziamo inoltre che nello stato iniziale in alto ci sia il valore 1 in uscita e il valore 0 in uscita in basso.
Che cosa succede ?
Quest'uscita 1, la riportiamo qui all'ingresso della porta logica di sotto e fa sì che la porta logica di sotto abbia 0 ed 1 come input in ingresso, ciò significa che l'uscita è 0, allo stesso modo questo 0 qui portandolo in ingresso sopra fa sì che 0 e 0 in ingresso ci diano 1, ciò sostanzialmente vuol dire che finché noi manteniamo qui in ingresso il valore 0 0 questo stato è consistente, cioè è in grado di mantenere il valore 1 nella porta sopra e 0 in quella sotto.
Passando alla parte di destra possiamo vedere che se noi siamo in grado anche di memorizzare il valore opposto, cioè 0 ed 1, vediamo il perché.
Ipotizziamo che qui sopra ci sia 0, questo finisce in input qui sotto e quindi abbiamo come input 0 0, che genera in uscita 1, questo 1 mandato in input sopra diventa parte della coppia 01 che da come output 0, e quindi lasciando in input i valori 0 e 0 siamo in grado di memorizzare il valore 1.
Se consideriamo come output complessivo del circuito il valore di Q allora possiamo dire che questo circuito latch è in grado di mantenere il valore 1 memorizzato precedentemente, così come lo stesso circuito in grado di mantenere il valore 0; ciò

vuol dire in sostanza che questo circuito in presenza dei due valori di input S ed R pari a 0 è in grado di mantenere memorizzato il valore precedentemente inserito, sia che il valore sia pari a 0 che pari a 1.
Questo è l'elemento circuitale più semplice in grado di memorizzare 0 oppure 1, a patto che in input manteniamo i valori di ingresso 0, cioè il valore basso, quindi se teniamo S ed R pari a 0 siamo capaci di mantenere il valore che c'era prima.
Il valore di Q prima come era stato messo ?
Ci dovrà essere un modo per cambiare questo valore?
Questo valore di Q può essere accettato o cambiato usando valori di S ed R diversi da 0, allora per i valori di S ed R quante combinazioni abbiamo ?
Quattro, ma in realtà di queste quattro combinazioni solamente tre sono combinazioni utilizzabili, la combinazione 0 0 che abbiamo visto già, la quale è in grado di mantenere memorizzato il valore precedente; poi abbiamo altre due combinazioni valide che sono S pari a 1 ed R pari a 0 oppure S pari a 0 ed R pari a 1.
L'unica combinazione che non si può usare è quella nella quale sia S sia R sono pari ad 1, quella è una condizione non ammessa perché può portare ad una situazione di crisi e da dato memorizzato non valido.
A questo ipotizziamo di cambiare i valori di ingresso di S e di R, cominciamo con guardare lo schema sinistra qui stiamo mettendo S pari ad uno ed R pari a 0, noi grazie alla tabella di verità sappiamo che basta che uno dei due valori in input sia pari ad 1 nella NOR per far si che l'output valga zero; d'altra parte noi abbiamo il valore di output della NOR di sopra pari a 0 ma questo valore finisce in quella di sotto, per cui qui sotto abbiamo come input 0 e 0 che ci dara come output 1.
Da qui possiamo dire che essendo che la S sta per set, impostando 1 come input nella s noi stiamo effettivamente sestando il valore in memoria pari a 1; una volta che ho impostato il valore di output pari ad, posso tornare nello stato di S pari a 0 per lasciare questo valore memorizzato nel circuito.
Guardiamo il caso opposto in cui abbiamo S pari a 0 ed R pari a 1, possiamo intuire che dato che la R sta per rese questa combinazione serve proprio per resettare il valore, ma vediamo cosa accade, se noi impostiamo qui il valore pari ad 1 guardando la tabella di verità possiamo notare che in qualsiasi caso ci dara come valore in uscita 0, e quindi cosi siamo capaci di impostare il valore del circuito pari a 0, fatto ciò torniamo allo stato di R pari a 0 e memorizziamo quindi il valore Q pari a 0.
Quindi questo è circuito nel quale in condizioni di non modifica del valore mi basta alimentare R ed S a tensione bassa, mentre quando voglio cambiare il contenuto mando un valore alto sulla S e quindi settario mentre quando lo voglio resettare mando un valore di alta tensione sulla R.
L'unica combinazione che non si deve utilizzare è S ed R pari a 1 perché si può verificare che sia il valore di Q che quello di Q negato si annulla, per cui successivamente qualsiasi in input io darò alla porta produrrà una risultato non prevedibile, data questa imprevedibilità la combinazione S ed R pari ad 1 deve essere negata.
Qual è un limite di questo tipo di circuito latch ?
Il circuito latch diciamo in linea di principio funziona, e noi lo enunciamo soprattutto per capire come si può memorizzare un dato in un circuito combinatorio, ma in realtà ci sono altri problemi, intanto i valori di S e R devono essere stabili ciò vuol dire che non devono presentare variazioni di valore nel tempo, salvo quelle previste, altrimenti si può dare una situazione di errore; tuttavia noi abbiamo già visto visto che S e R sono sicuramente gli output di un circuito combinatorio, ma noi avevamo anche visto che, gli output dei circuiti combinatori essendo soggetti a alee statiche, ovvero variabilità legate ai ritardi, e delle volte non diventano stabili istantaneamente ma si stabilizzano dopo un certo intervallo di tempo.

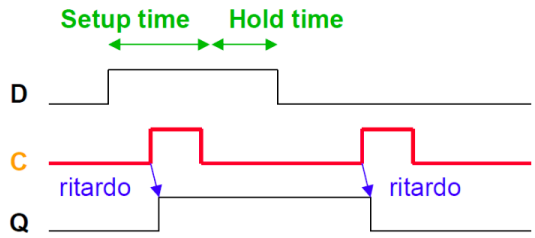
Clocked latch (D-latch)



- NOTA: latch disegnato *sottosopra* rispetto alla figura precedente
  => posizioni invertite di (S, R) e di (Q, ~Q)
- D=1 corrisponde al *setting*
  - S=1 e R=0
- D=0 corrisponde al *resetting*
  - S=0 e R=1
- La combinazione S=1 e R=1 non può mai verificarsi
- A causa delle 2 porte AND, quando il clock è *deasserted* abbiamo che nel latch non viene memorizzato alcun valore:
  - S=0 e R=0
- Viene memorizzato un valore (in dipendenza del valore di D) solo quando il clock è *asserted*
- Se il segnale C fosse invertito (porta NOT aggiuntiva), il latch memorizzerebbe sul valore *basso* del clock (cioè quando il clock è *deasserted*)

Il clock, dunque, genera dei segnali a gradino usati per abilitare i circuiti solo quando il clock stesso si trova in uno stato specifico (condizione: valore uguale a 1 o valore uguale a 0 oppure quando si ha il fronte di salita o discesa).
Si dice che il clock ha un valore abilitativo in quanto l'output del latch viene memorizzato se il clock è abilitato.

Quanto deve essere lungo il periodo abilitante del clock?
Viene scelto dal programmatore, il quale deve fare in modo che il valore di D sia maggiore del valore del clock.



In particolare, il valore dell'input è formato da:

•**Tempo di Setup**, che è il tempo necessario affinché il segnale in input al circuito del latch dia il segnale desiderato in output;

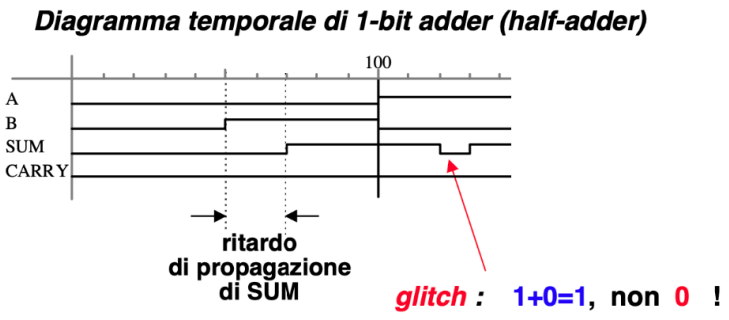
•**Tempo di Hold** (Mantenimento), si aggiunge per evitare mal funzionamenti.

La somma dell'Hold time e del Setup time deve essere maggiore del tempo del clock e deve diventare positivo poco prima che il clock lo sia e, d'altra parte, deve essere negativo poco dopo che il clock è diventato negativo.
Da quando il clock diventa positivo, il valore di Q (output) diventa positivo con un ritardo rispetto al clock. Quando il clock ritorna abilitato, si ha, sempre con un lieve ritardo, il ritorno di Q a 0.

Cosa succede prima che si stabilizzi l'output ?
Il risultato potrebbe essere scorretto, abbiamo visto per esempio che ci potrebbe essere un piccolo intervallo di tempo in cui il risultato scorretto, pois alvo situazioni stranissime i risultati diventano più corretti, ma noi dobbiamo gestire anche gli output transitori, per evitare che questi output ci portino a memorizzare dati errati si usano dei segnali di abilitazione nell'input, che prendono il nome di clock, il clock è semplicemente un segnale di un orologio che è presente nel computer e che dà la tempistica dell'esecuzione delle istruzioni, segnali che possiamo immaginare come dei segnali a gradino, in cui si alternano i valori in cui valgono 0 ed 1, per il momento possiamo dire, poi vedremo che non è sempre così, che questi segnali di clock sono usati per amplificare l'esecuzione di operazioni soltanto quando il valore del Clock è pari ad 1; quindi se abbiamo un circuito con un clock, possiamo decidere di abilitare l'inserimento dei valori solo se il clock vale 1 mentre se vale 0 non si possono inserire.
In realtà ci sono anche altre interpretazioni, si può effettuare il contrario, cioè si può inserire il valore quando il Clock è pari a 0, si può anche impostare l'abilitazione all'inserimento quando c'è la transizione del clock da 0 ad 1, ovvero in fronte di salita, ma ciò si può impostare ovviamente anche con il fronte di discesa ovvero quando il Clock passa da 1 a 0; in ogni caso usiamo un segnale il gradino, ovvero il clock, in cui periodo di tempo che intercorre tra i segnali è definito, ed il periodo però deve essere abbastanza grande in modo che il circuito si sia stabilizzato prima di dare l'input un segnale di ingresso al latch, ciò in sostanza serve a dare l'altra tempistica con cui questo latch si può modificare.
Se noi prendiamo il latch di prima, come vedete fatto da due porte NOR, anziché alimentarlo direttamente con R ed S, facciamo in modo che R ed S siano prodotti a loro volta da due porte AND.
Queste porte AND come vedete ricevono in input un segnale che è il clock, cosi in realtà state facendo ?
Sappiamo la AND tira fuori come output 1 se e solo se entrambi i valori di input sono 1, quindi se io metto qui il Clock che può valere 0 o 1, solamente quando vale 1 viene generato un output.
Quindi se io prendo una variabile e la metto diretta in una porta e negata in quest'altra porta, l'uscita di queste due porte And sarà sempre opposta, e in particolare se D è pari ad 1 e il clock anche sarà pari a 1, qui il risultato sarà 1 mentre qui uscirà 0, ma se il clock è pari a zero indipendentemente dal valore di D l'uscita sarà sempre zero, quindi noi stiamo dicendo che questo segnale di sopra il clock, un segnale abilitativo, cioè se vale zero il valore di D può essere un valore qualsiasi ma esce sempre 0, ma se C è pari ad uno allora il valore di D viene considerato, e in particolare se D è pari a 1 esce 1 e quella sopra esce 0, viceversa se D è pari a 0 e C è pari a 1 i risultati si invertiranno.
In pratica abbiamo costruito con un circuito che è in grado di generare gli input R ed S di questo latch, a partire da un unico valore D unitamente a un clock di abilitazione.
Quindi in sostanza se D è pari ad 1 è come se stessimo facendo il setting di questo latch, mentre se D è pari a 0 stiamo effettuando il resetting.
Chiaramente S ed R pari a 1 qui non si può verificare mai, perché per come sono generati vengono sempre di di valore opposto, quindi abbiamo anche risolto il problema dell' eventuale errore del doppio input pari a 1.
La cosa importante è che a causa delle due porte AND i valori R ed S si possono produrre solamente quando il clock è pari ad 1, quando il Clock è pari a 0 R ed S sono sempre pari a 0, quindi il valore resta memorizzato e noi possiamo modificare il valore di questo circuito soltanto quando il clock è pari a 1.
Come detto prima nulla vieta di ragionare al contrario, potremmo anche decidere di abilitare la scrittura quando il clock è pari a 0, in quel caso ci basta aggiungere una porta NOT e funziona esattamente il contrario, cioè quando il clock vale zero si può settare il circuito.
Qual è il motivo per cui viene introdotto questo clock ?
Per ridurre i casi di errore in quanto questi possono avvenire solamente quando il valore di clock è pari a 1.
Quanto deve essere lungo il periodo del clock ?
Questa è una scelta di progettazione molto importante, noi dobbiamo impostare un valore costituito da due componenti: • il tempo di setup e • il tempo di mantenimento; quello che è importante è il tempo di setup, che rappresenta il tempo necessario affinché il segnale in input al circuito del latch sia in grado di assegnare il valore desiderato, più una somma di

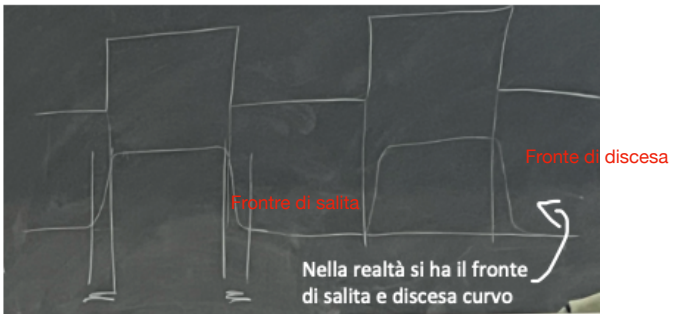
mantenimento che si aggiunge per intero, la somma di queste due deve essere maggiore del tempo del clock e soprattutto l'input di D deve diventare positivo prima che il clock diventi positivo e di conseguenza deve diventare negativo dopo che il clock è diventato negativo.
Bisogna quindi generare il segnale D in modo che diventi positivo prima che il clock diventi positivo e negativo dopo che il clock diventi negativo, in altri termini il tempo di clock deve racchiudere il tempo di generazione dell'output (ovvero il valore memorizzato dal latch).
Questo perché il circuito ha un ritardo, indicato con questa freccia, il valore di Q diventa positivo un pochino dopo di quando diventa positivo il valore del clock, analogamente avviene con il caso opposto e quindi si può vedere il fronte di discesa.
Ricordiamo che quando in un circuito combinatorio si fanno entrare degli input che presentano un ritardo non nullo, come nella realtà, si possono verificare dei cosiddetti glitch ovvero un errore tecnico che consiste nel trovare un valore di uscita non coerente con quello che ci aspetteremo di trovare, ma la caratteristica fondamentale dei glitch è che sono errori tecnici transitori.
In questo esempio che mostra un Half-adder, circuito che calcola la somma tra bit, si evince che mutando il valore dei bit la somma non si genera istantaneamente ma dopo un po di tempo, questo tempo è dovuto da un ritardo di propagazione della soglia, se cambio contemporaneamente i due valori si può verificare un errore, ricordiamo che come abbiamo già detto nei circuiti di AND e OR due valori non vanno cambiati contemporaneamente, questo perché ci sarebbe un istante di tempo in cui il risultato potrebbe essere completamente sbagliato.
Dobbiamo in qualche modo tenere conto di questo problema dei circuiti combinatori nel definire il timing corretto del nostro clock.

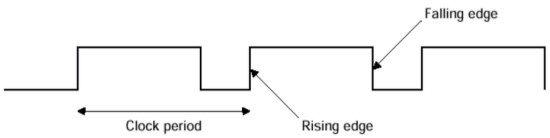
Metodologie di timing

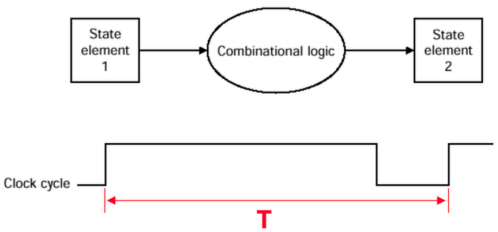


Un glitch è un errore tecnico transitorio dell'output prodotto.
Per prevenire tale errore è necessario fissare un corretto timing del clock.
Esistono due metodologie di timing:

- **Level-triggered methodology**: avviene sul livello alto (o basso) del clock. Presenta un ritardo considerevole dovuto al fatto che il fronte di salita/discesa occupa un intervallo di tempo non trascurabile.
- **Edge-triggered methodology**: avviene sul fronte di salita (o discesa) del clock. È quasi istantaneo.







In questa seconda tipologia di timing si può immaginare che la memorizzazione avvenga istantaneamente, e che l'eventuale segnale sporco, proveniente dal circuito combinatorio,

non faccia in tempo ad arrivare a causa dell'istantaneità della memorizzazione.

Quali sono le metodologie di timing che si possono utilizzare ?
Sostanzialmente il timing può essere suddiviso in due categorie:
- Level-triggered methodology
- Edge-triggered methodology.

Quali sono le differenze ?
Nel modello Level, ovvero basato sui livelli, la memorizzazione del valore avviene o sul valore alto o sul valore del basso, in particolare in questo esempio fatto qua è un modello level triggered settato sul valore alto; nel modello Edge triggered la memorizzazione avviene o sul fronte di salita o sul fronte di discesa.
Si intuisce che i due approcci presentano una differenza logica fondamentale: mentre nei modelli level la memorizzazione avviene in un tempo che non è trascurabile, nei modelli edge possiamo immaginare che la memorizzazione avviene istantaneamente,.
Il segnale sul fronte di salita in realtà è una cosa tangibile (in quanto una curva e non una retta) in quel contesto sembrerebbe avere una durata nulla, lo stesso vale per il fronte di discesa; anche se essi sono quasi istantanei, possono essere più o meno ripidi, più sono ripidi e meglio è ma non sono sicuramente nulli.
Questo è il motivo per cui noi preferiamo in molte circostanze lavorare sui fronti di salita o di discesa, proprio perchè riducendosi di molto il tempo durante il quale si modifica lo stato del del latch, impedisce al segnale di output del latch di rientrare indietro e bloccare gli eventuali segnali di ritorno sporco.
Per segnali sporchi si intendono segnali di valore scorretto che si potrebbero ripresentare l'ingresso nel circuito alterandone in modo errato il valore.

Periodo del ciclo di clock

Il periodo T deve essere scelto abbastanza lungo affinché l'output del circuito combinatorio si stabilizzi.

Il periodo T deve essere scelto abbastanza lungo affinché l'output del circuito combinatorio si stabilizzi (deve essere stabile un po' prima del periodo di apertura del latch)

Se T è espresso in *sec*, la frequenza del clock è dato da Freq = 1/T Hz
Se T = 10 nsec, qual è la frequenza del clock?
- **1 nsec = 10^{-9} sec 1 μsec = 10^{-6} sec 1 msec = 10^{-3} sec**
- **Freq = 1/T = 1 / (10 * 10^{-9}) = 10^{8} Hz = 10^{8} / 10^{6} MHz = 10^{2} MHz = 100 MHz**

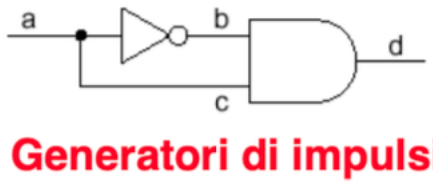
Dobbiamo ridurre il più possibile la lunghezza del clock positivo per fare in modo che il latch non rimanga aperto troppo a lungo modificando più volte l'output (glitch).
Ciò è possibile tramite un generatore di impulsi∫, ossia un clock con dei periodi positivi molto brevi.

Il periodo naturalmente si esprime in secondi mentre la frequenza si esprime in hertz.
La scelta del della frequenza importante poiché determina per l'appunto la frequenza con cui possiamo cambiare lo stato di un circuito.
Se io ho un circuito che cambia di Stato nel tempo maggiore la frequenza e maggiore è la possibilità che quest'ultimo cambi il suo stato nel tempo.
L'esempio che vedremo è un circuito sequenziale che gestisce un semaforo, quindi noi impareremo a progettare un semplice circuito che fa variare il valore del semaforo da verde a rosso in base alle macchine.
Dovremmo stabilire qual è la frequenza con cui facciamo questo aggiornamento, ciò determinerà ogni quanto il semaforo cambia il suo stato, quindi calcoleremo la frequenza del clock, noi non la vediamo ma in tutti i sistemi sequenziali c'è un clock che determina ogni quanto si aggiorna lo stato.
Supponiamo che il periodo sia 10 nanosecondi, cioè io voglio che lo stato del mio circuito si possa aggiornare ogni 10 nanosecondi, un nanosecondo sarebbe un miliardesimo di secondo, la frequenza di clock deve essere 1/t cioè 1/10 miliardesimi (10^-9) ovvero 10^8 Hz, volendo usare un multiplo, quindi 100 MHz.
quindi per poter modificare lo stato ogni 10 nanosecondi dobbiamo avere un clock che va alla frequenza di 100 MHz.
Abbiamo capito che per formare un circuito sequenziale ci serve una unità di memoria che potrebbe essere il latch che abbiamo visto prima, poi ci serve un circuito combinatorio che sulla base dei valori di ingresso e del risultato memorizzato nel latch o in altri circuiti equivalenti, sia in grado di produrre un output e ovviamente anche di alimentare di nuovo lo stato.

Quanti bit memorizziamo qui dentro ?
Uno solo, e ovviamente questo significa che se io volessi memorizzare 10 bit si dovrebbero usare 10 circuiti come questi, tutti messi uno dopo l'altro, quindi i famosi registi di cui abbiamo parlato in precedenza non sono altro che tanti di questi messi uno dopo l'altro.
In questo schema qui l'elemento di Stato non è altro che un latch o un'insieme di latch, quindi se mi serve memorizzare uno stato a solo due valori mi basta un latch, se dovessi memorizzare quattro valori avrei bisogno di due latch, se dovessi memorizzare 8 valori avrei bisogno di tre latch e così via, più alternative di Stato dobbiamo memorizzare maggiore è il numero di latch che dobbiamo introdurre.
Il circuito combinatore che viene mostrato qui con forma ovale, calcola una funzione sulla base del vecchio valore dello Stato, e poi produce un nuovo valore che va a finire di nuovo nello stato, questo schema qui si chiama schema di retroazione o feedback, è tipico e parte integrante di tutti i sistemi sequenziali.
Se io dovessi in uno stesso ciclo di clock calcolare un output a partire dal valore immagazzinato e poi contemporaneamente memorizzare il nuovo valore, posso usare un D-latch che contemporaneamente da un lato mi restituisce un valore e dall'altro è in grado di memorizzare il nuovo valore ?
Purtroppo no, non è in grado in un'unico periodo di generare un output e contemporaneamente memorizzare il nuovo valore, queste due azioni devono essere effettuate in due clock separati, dovrei usare un clock per generare il valore ed un clock successivo per inserire il valore nel registro; di conseguenza bisogna in qualche modo modificare questo circuito per renderlo più adatto allo scopo, sostanzialmente voglio evitare che l'output vada a modificare lo stato e di conseguenza l'output che genera il circuito (nel caso in cui il fronte di salita o di discesa dovesse essere molto lungo si potrebbe modificare lo stato prima di quando previsto [prima della fine del periodo del clock in cui è abilitata la modifica] ed il circuito produrrebbe due output, magri diversi per cui noi andremo a leggere quello sbagliato).

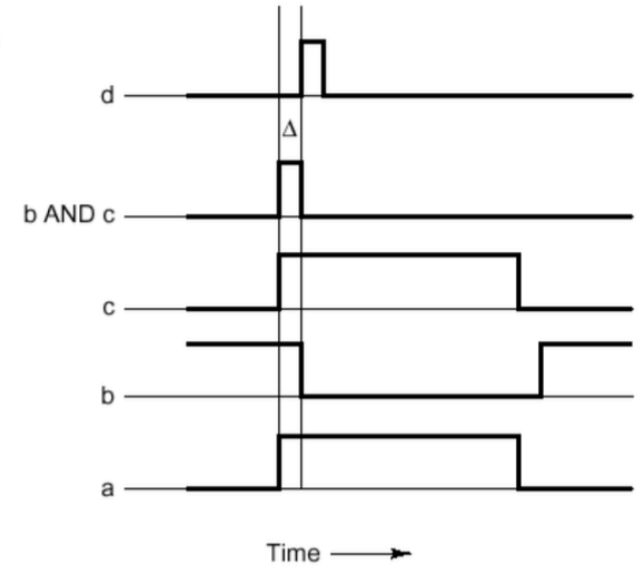
Generatore di impulsi

Il generatore di impulsi permette appunto di generare impulsi brevissimi in corrispondenza del fronte di salita di un segnale a gradino.
Il circuito del generatore d'impulsi è:



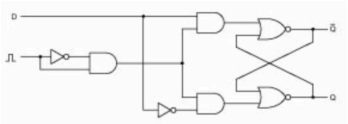
a=clock a gradino con periodo lungo
b= Set
c=Reset
d=output

Consideriamo un ritardo (Δ) nella porta NOT del circuito; ciò implica che, mentre c cambia in modo quasi istantaneo, la b presenta un ritardo. Mettendo in AND b e c si ha, se b e c sono 1, un valore di output d pari ad 1, anch'esso con un ritardo breve. Si ha così in output un impulso.
Il valore ottenuto viene memorizzato "istantaneamente" tramite un circuito detto flip-flop.

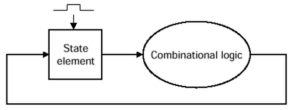


Quindi noi cosa dobbiamo cercare di fare ?
Dobbiamo ridurre la durata del fronte il più possibile per fare in modo che l'output generato, una volta fuori dal circuito combinatorio non fa in tempo a rientrare indietro perché nel frattempo il valore del clock è diventato zero e cosi la modifica dello stato si blocca.
Qual è il modo migliore, che si usa nella realtà, per ridurre il più possibile il tempo in cui il latch rimane aperto in un sistema retro-azionato ?
La soluzione che si usa, è quella di usare un generatore di impulsi, cioè un oggetto in grado di generare dei clock con dei periodi di asserzione molto brevi (Clock con dei periodi positivi molto brevi), che sono sufficienti per dargli un valore ma fanno a tempo anche a chiudere il latch, in modo che l'eventuale valore generato dal circuito combinatore a valle non faccia in tempo ad essere memorizzato dal latch stesso.
Ciò che volevo dire prima è che quando viene generato un output dal circuito combinatorio, questo finisce di nuovo in ingresso nel latch, e ciò questo è indispensabile perché altrimenti non sarebbe un circuito retro-azionato, ma ciò che deve essere evitato è che questo valore arrivi quando il latch è ancora aperto, quindi quando il clock è ancora positivo (nello stesso istante di clock) e finisce per modificare lo statiche potrebbe modificare a sua volta il valore di output uscente; come già detto la soluzione è quella di ridurre il tempo di clock, cosi da evitare che il circuito del latch risulti ancora aperto nello stesso istante/ciclo di clock ed evitare cosi a sua volta che l'output si possa modificare molteplici volte durante lo stesso ciclo.
Il principio generale dei circuiti sequenziali è che l'output deve cambiare una solva volta per ogni ciclo di clock.
Un generatore di impulsi è un circuito semplicissimo in grado di generare dei fronti positivi molto brevi, è un circuito in cui diamo in input due impulsi in una porta AND entrambi ottenuti a partire da un clock; il clock è caratterizzato da un periodo abbastanza lungo, per ridurre il periodo sfruttiamo il tipico ritardo (il tempo che intercorre fra quanto diamo l'ingresso alla porta e quando arriva l'uscita) presente in qualsiasi porta, in particolare in una porta NOT, come possiamo vedere il valore C dipende solo da A e quindi non è caratterizzato da ritardo, mentre B dipende dall'uscita di A dalla porta NOT ed avrà dunque un ritardo, ciò vuol dire che i due valori cambieranno in momenti diversi, se adesso consideriamo ciò che avviene a C e B in AND noteremo che l'impulso in uscita sarà dipendente dalla durata del ritardo Delta della porta NOT, ciò produce delle rapide generazioni di impulso dalla porta AND, perché per ogni cambio di stato di A in realtà ci sarà un valore che si formerà per lo stato precedente di B (dato che cambia in ritardo) ed uno che si formerà per quando B raggiungerà lo stato di A negato.
Sfruttiamo questo generatore di impulsi per generare un circuito basato sul latch, in grado di memorizzare il valore istantaneamente quindi durante questo breve impulso, e di chiudersi subito in modo che l'output del circuito combinatorio non faccia in tempo a rientrare dentro il circuito stesso, questo circuito prendi nota di flip-flop.

Flip-flop



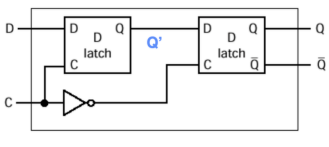
- Il D flip - flop memorizza "istantaneamente" il valore di D sul fronte di salita del clock, ovvero in corrispondenza dell'impulso
  - metodologia edge-triggered di tipo rising triggered

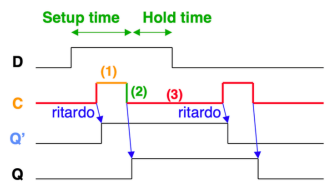


- Il segnale memorizzato nel flip-flop comincia a fluire subito fuori dal flip-flop
- A causa della brevità dell'impulso, questo segnale "non fa in tempo" a entrare nel circuito combinatorio a valle, e a modificare l'input del flip-flop

Il flip flop semplice appartiene alle metodologie edge-triggered di tipo rising triggered, ossia è come se stessimo considerando il clock positivo nei fronti di salita. Grazie alla breve durata del clock, non si ha più la problematica legata al cambiamento multiplo dell'output dovuto alla sua uscita e rientro nel circuito.

Si può ottenere un flip flop più complesso di tipo D formato da due latch in cascata ed un clock.
Il clock è generato da un generatore di impulsi, la D contiene i valori di S ed R. Quando il clock ha un valore alto il primo latch è aperto mentre il secondo è chiuso e viceversa, quando il clock è basso il primo è chiuso ed il secondo aperto (logica inversa).
Quando il clock è positivo il primo latch si apre e restituisce un valore Q' al





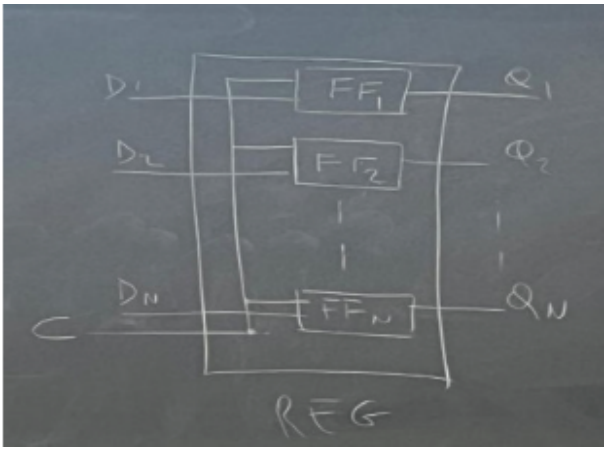
- (1)
  - Il primo latch è aperto e pronto per memorizzare D. Il valore memorizzato Q' fluisce fuori, ma il secondo latch è chiuso
    - => nel circuito combinatorio a valle entra ancora il vecchio valore di Q
- (2)
  - Il segnale del clock scende, e in questo istante il secondo latch viene aperto per memorizzare il valore di Q'
- (3)
  - Il secondo latch è aperto, memorizza D (Q'), e fa fluire il nuovo valore Q nel circuito a valle. Il primo latch è invece chiuso, e non memorizza niente

secondo latch, il quale è però chiuso.
Quando il clock diventa negativo, per
la presenza di una porta NOT, attiva il secondo latch che memorizza il valore di Q' e rilascia in output il valore Q. Grazie alla presenza del clock, che chiude in modo "istantaneo" il circuito, non si ha il ritorno dell'output nel primo latch, riuscendo a separare l'input dall'output.
Un eventuale nuovo valore di input non modifica il valore precedentemente in output.
È una tecnica sempre edge-triggered ma di tipo falling triggered, ossia la memorizzazione è istantanea rispetto al fronte di discesa.
Riassumendo: un flip-flop di tipo C-D è in grado di memorizzare il valore precedente nel primo latch senza influenzare l'output corrente del secondo latch in serie.

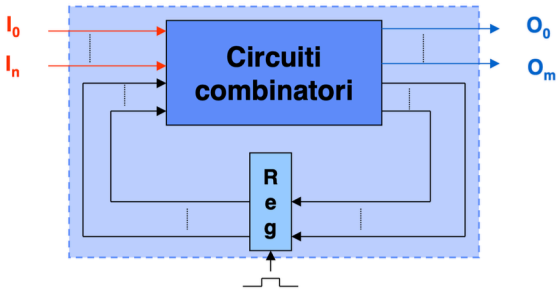
Un Flip-flop non è nient'altro che un latch di tipo d in cui sostanzialmente usiamo un clock ad impulsi, quindi utilizziamo questo generatore di impulsi appena descritto, in grado però di memorizzare il valore del risultato su quello che chiamiamo fronte di salita del clock, ma che di fatto è questo brevissimo impulso che il generatore riesce a creare.
Questa tecnica per memorizzare il valore appartiene tecnicamente al metodologia edge-triggered, cioè memorizziamo come se fossimo sul fronte di salita, ovvero siccome la durata del periodo è molto breve è come se stessimo memorizzando sul fronte di salita, per questo motivo la definiamo una tecnica di memorizzazione basata sul fronte di salire in particolare appartiene alla metodologia di tipo rising.
Che cosa succede in questo circuito ?
Lo state Element manda un impulso al circuito combinatorio, il quale calcola un output, questo torna indietro ma quando torna si è già abbassato il valore del clock, conseguentemente il circuito continuerà a produrre un risultato stabile per tutta la durata del fronte nullo.
Rimane però un problema poiché noi abbiamo la necessità in uno stesso ciclo di clock da un lato di usare il valore dell'output dall'altro di modificare lo stato senza che si incorra nel problema di prima.
La soluzione si ottiene mettendo in cascata due latch ottenendo un flip-flop che è più complesso ma che risolve il problema, questo prende il nome di Flip-flop falling edge triggered.
Questo tipo di flip-flop viene di solito sostituito con questo schema più articolato in cui si usano in cascata due latch, come al solito c'è un clock generato da un generatore di impulsi e poi c'è un valore D che è quello che internamente produce i valori S ed R, il clock come vedete alimenta sia il primo latch sia il secondo, ma li alimenta in modo inverso, quando è alto in uno arriva un valore alto e nell'altro basso e viceversa, questo fa sì sostanzialmente che si crei un alternanza tra le aperture dei latch.
Abbiamo realizzato un componente che ha due unità di memoria, che lavorano sempre in modo indipendente, quindi lavorano sempre con una logica invertita.
Quando il clock è aperto, ovvero vale 1 accetta modifiche quindi il segnale D memorizza un valore, poi chiude e avremo un valore intermedio che chiameremo Q' , primo per dire che il valore è diverso da Q finale che uscirà, questo andrà in input al secondo latch il quale lo memorizzerà durante il fronte nullo.
È chiaramente una metodologia e Edge-triggered, perché si basa sull'uso di un generatore di impulsi, ma è tipo FALLING Edge-triggered perché il valore finale viene memorizzato sul fronte di discesa.
In questo schema sono riportati i valori di D,C e Q primo, D sarebbe il valore che do in ingresso all'intero circuito, Q è il valore complessivo finale, Q' è il valore intermedio e poi chiaramente la C è il clock.
Ci sono sostanzialmente 3 fasi: • il clock diventa 1 ed il primo latch è aperto e chiaramente il secondo è chiuso, quindi il primo legge il valore di D ed esce fuori un output, ovvero Q' , tuttavia questo non entra nel secondo perché è ancora chiuso e quindi vede e lavora ancora sul vecchio valore; • scende il valore del clock; • il secondo latch viene aperto Q' viene memorizzato e il valore di Q viene fatto uscire fuori dal circuito.
In questo modo il valore che esce non riesce a rientrare e quindi siamo riusciti a separare l'output dall'input, e ciò che ci interessa è che questo dall'esterno viene come un unico circuito, di fatto Q' non si vede fuori, e noi abbiamo trovato un modo che permette di memorizzare il valore in uscita e far uscire il valore precedente.
Il flip-flop di tipo CD che noi siamo internamente si realizza tra due latch CD, alimentato da un generatore di impulsi e che ha proprietà di riuscire in uno stesso ciclo di clock di memorizzare un valore nuovo e di il valore per generare l'output senza che questo finisca per generare cambiamenti di stato non voluti.

Circuito sequenziale sincrono

Un registro è banalmente una serie di flip-flop uno dopo l'altro, in particolare abbiamo bisogno di tanti flip flop per quanti sono i bit che devono essere memorizzati.
Un registro è caratterizzato dall'avere un unico valore di clock in ingresso che viene passato in parallelo a tutti flip flop.
Nella foto a destra abbiamo un esempio di circuito sequenziale sincrono con il clock, ossia un circuito in cui le modifiche dello stato si hanno soltanto in corrispondenza dei valori di clock positivi. *Q*1, *Q*2...,*Q*N sono i valori di output e il valore C indica il clock.

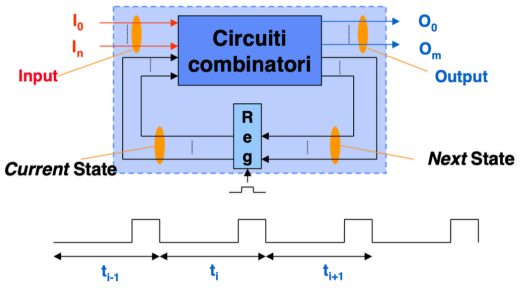


Tale circuito è composto da due parti fondamentali:



•Circuiti combinatori
•Elementi di memoria clockati (registri realizzati tramite flip-flop), che mantengono lo stato del circuito e che possono essere letti/scritti nello stesso periodo di clock.

Il circuito combinatorio deve sostanzialmente prendere i valori di input da due componenti:



Consideriamo che il Registro di stato è realizzato con flip-flop che impiegano una metodologia *falling-edge triggered*
– **durante il tempo t_i, il prossimo stato viene calcolato (ovvero lo stato al tempo t_{i+1}), ma viene memorizzato *solo* in corrispondenza del fronte di discesa del clock**

•Input prodotti dall'esterno (*I*0 , *I*N)
•Valori del registro.
Successivamente elabora un duplice risultato: l'output che va all'esterno (senza andare in memoria) ed i nuovi valori che vanno messi nel registro.

Il registro di stato è realizzato con flip-flop che impiegano una metodologia falling-edge triggered, ossia secondo cui lo stato successivo viene memorizzato solo in corrispondenza del fronte di discesa del clock. In pratica, durante il tempo *t*& viene

calcolato il prossimo stato al tempo (ti+1), ma tale valore viene memorizzato solo se il valore del clock corrisponde al fronte di discesa.
Il circuito combinatorio prende l'input e genera l'output sulla base dello stato corrente mentre produce lo stato successivo (Next State) che va in registro.
I circuiti sequenziali possono essere modellati secondo due approcci:

- Mealy
- Moore

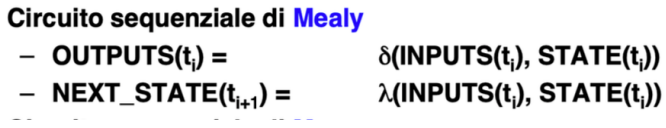
Definiamo:

- INPUTS(ti) e OUTPUTS(ti) come i valori presenti, rispettivamente, sugli input e gli output dei circuiti combinatori a tempo ti.
- STATE(ti) come i valori presenti nei registri di stato al tempo ti.

Una volta che abbiamo questo componente possiamo costruire quello che qui viene indicato con la parola registro, un registro cos'è ?
Una serie di Flip-flop, provvisto di tanti di questi oggettini qui per quanti sono i bit che devono essere memorizzati, chiaramente il registro è caratterizzato da un avere un unico valore di clock di ingresso, che viene passato in parallelo a tutti i Flip-flop.
Quindi un registro non è altro che un componente che al suo interno ha tanti flip-flop quanti sono i bit da memorizzare, in output ha altrettanti tanti bit e chiaramente ha un unico clock che viene distribuito in ingresso a tutti i flip-flop, per memorizzare i valori chiaramente bisogna farlo quando il clock è alto, quando il clock si abbassa il valore precedente rimane memorizzato.
D'ora in poi possiamo immaginare che qui dove c'è indicato REG ci sia una cosa di questo tipo, dove quel segnetto ("_-_") rappresenta il segnale c, queste linee che entrano qui dentro sono i segnali D.
Questo schema in gergo chiamato circuito sequenziale sincrono, dove sincrono sta per sincronizzato con il clock, cioè le modifiche di stato in tale circuito vengono soltanto in corrispondenza dei clock positivi, è composto da due parti fondamentali il circuito combinatore e il registro, quindi l'insieme dei Flip-flop che mantengono lo stato, riceve in input dei bit e genera degli output; gli input vanno in ingresso al circuito combinatore e gli output sono generati sempre dal circuito combinatore.
Cosa deve fare il circuito combinatorio ?
Il circuito combinatorio deve sostanzialmente prendere il valore di input e prendere i valori del registro, quindi riceve in input due componenti per questo vedete queste doppie frecce, ed elaborare un duplice risultato, un risultato è quello che va all'esterno e un altro sono i nuovi valori che vanno messi nel registro.
Ora capite bene perché il tipo di registro che subiamo qua è basato sul flip-flop dell'ultimo tipo che abbiamo visto, perché deve essere in grado di usare da un lato i vecchi valori del registro e dall'altro di produrre dei valori che finiscono come nuovi valori del registro, è un po come se stessimo usando una memoria di appoggio.
Ciò avviene anche quando voi programmando scrivete in codice, quando fate x=x+1 in pratica qui state prendendo il vecchio valore di x lo state cambiando e lo state rimettendo di nuovo in x, questa operazione viene fatta con una variabile di appoggio ed anche nel caso del registro state usando una memoria di appoggio che si ottiene duplicando la memoria (2 latch).
Essendo quindi sul principio di falling-edge-triggered il valore viene memorizzato su fronte di discesa, ciò viene calcolato nell'istante t con i, ma viene internalizzato nel fronte di discesa.
Il circuito combinatorio prende un input e sulla base dello stato corrente genera un output, producendo cosi lo stato successivo (NEXT STATE).
I circuiti sequenziali possono essere modellati secondo due approcci: quello di Mealy e quello di Moore, questi hanno tra loro sottili differenze ma per comprenderle dobbiamo prima definire tre formalismi: gli input all'istante T con i, gli output ad un certo istante T con i, ed infine i valori di registro nell'istante T con i.

Circuito sequenziale di Mealy

Il circuito combinatorio calcola lo stato e l'input successivo considerando il valore degli input e dello stato momentaneo.
Secondo il modello Mealy sia l'output sia il prossimo stato dipendono entrambi dall'input e dallo stato corrente.



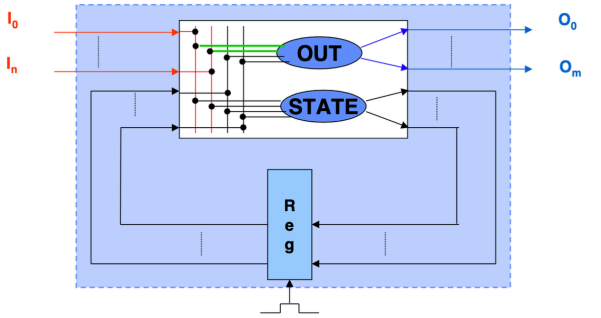
Circuito sequenziale di Moore

L'output dipende esclusivamente dallo stato attuale del registro, mentre il prossimo stato dipende sia dallo stato che dagli input del momento.
Da notare che nel circuito di Moore il valore dell'output al tempo ti dipende solo dal valore dei registri di stato, mentre nel circuito di Mealy dipende anche dall'input. Nei circuiti di Moore, i registri di stato sono modificati dal ciclo di clock precedente (ti-1) sulla base degli input a quel tempo presenti in ingresso al circuito.

Circuito sequenziale di Moore

- $OUTPUTS(t_i) = \delta(STATE(t_i))$
- $NEXT\_STATE(t_{i+1}) = \lambda(INPUTS(t_i), STATE(t_i))$

Abbiamo quindi bisogno di 2 circuiti combinatori, che in principio sono distinti
I collegamenti in verde verso il circuito OUT non sono necessari per realizzare circuiti sequenziali di Moore



In questa figura si osserva la differenza tra il modello di Mealy e il modello Moore. Quando realizziamo un circuito sequenziale alla fine abbiamo bisogno di due circuiti combinatori, un circuito di output e un circuito di calcolo dello stato.
Il circuito di output prende l'input a secondo che siamo nel caso di Mealy o Moore.

Ipotizziamo di essere in un circuito di Moore, l'output dipende esclusivamente dallo stato del momento e quindi non considererà l'input corrente. Nel caso di circuito di Mealy, l'output prende in considerazione anche gli input correnti.

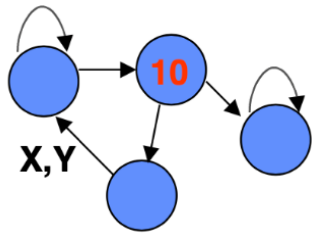
Se noi abbiamo gli input ad un certo istante di tempo T con i, gli output e gli stati ad allo stesso istante di tempo, la differenza tra Mealy e Moore è: • nei circuiti sequenziali di Mealy abbiamo che l'output ad un certo istante di tempo viene determinata da una funzione chiamata Delta che dipende sia dagli input che dallo stato in quell'istante di tempo, il prossimo stato invece dipende da una funzione Lambda applicata agli input e agli stati nel determinato istante di tempo, in altre parole sia Delta che Lambda dipendono entrambi dall'input e dallo stato corrente; • secondo invece il circuito sequenziale di Moore l'output dipende esclusivamente dallo stato attuale del circuito mentre il prossimo stato dipende dall'input e dallo stato nell'istante di tempo corrente.
Possiamo notare quindi che l'unica differenza si trova nella restituzione dell'output perché per Moore dipende solo dallo stato (t-1 produce da solo l'output in t), mentre per Mealy dipende anche dagli input.
Quando noi realizziamo un circuito sequenziale, alla fine ci ritroveremo a realizzare due circuiti combinatori, un circuito di output ed un circuito di calcolo dello stato, ed ovviamente li realizzeremo usando le tecniche delle mappe di Karnaough.
Il circuito di Moore abbiamo detto che calcola l'output cioè questi solamente in funzione del valore del registro, quindi nel circuito queste due linee entrano salgono e finisce in output, quindi circuito combinatorio out calcola gli output solamente tenendo conto nel caso di Moore del valore contenuto nei registri.

Nota: i collegamenti in verde verso il circuito OUT non sono necessari per realizzare circuiti sequenziali di Moore, ci sono solamente in caso di Mealy. Invece la funzione di cambiamento di stato, NEXT STATE (la funzione che calcola il prossimo stato), prevede lo stesso circuito combinatorio in entrambi i casi, sia in caso di Mealy che di Moore, che prende come input sia l'input esterno che dei registri.

Dunque, la differenza tra Moore e Mealy è che la funzione di output nel primo caso prende come input solo l'input del registro, mentre nell'altro caso prende gli input del registro e gli input correnti. Per sintetizzare il circuito sequenziale in maniera diretta basta conoscere le tabelle di verita delle funzioni OUTPUTS e NEXT-STATE.
Dalle tabelle siamo poi in grado di determinare le equazioni booleane e i corrispondenti circuiti.

Nel caso invece di Mealy l'output non dipende solamente da questi valori di registro ma dipende anche dagli input, questi indicati in verde (presenti solo nel caso di Mealy); invece la funzione di next state, cioè la funzione di calcolo al prossimo stato in entrambi i casi prende come input sia S0 e S1 sia I0 e In, chiaramente l'uscita del next state finisce di nuovo nel registro.
La sintesi di circuiti sequenziali si basa sulla definizione delle tabelle di verità sulla successiva minimizzazione tramite le mappe di Karnaugh per le due funzioni Outputs e next state.

Ma come si fanno a specificare e derivare le tabelle di verità?
Vedremo in seguito come è possibile specificare il comportamento di un circuito sequenziale tramite un particolare programma, espresso graficamente come un automa a stati finiti, formato da:



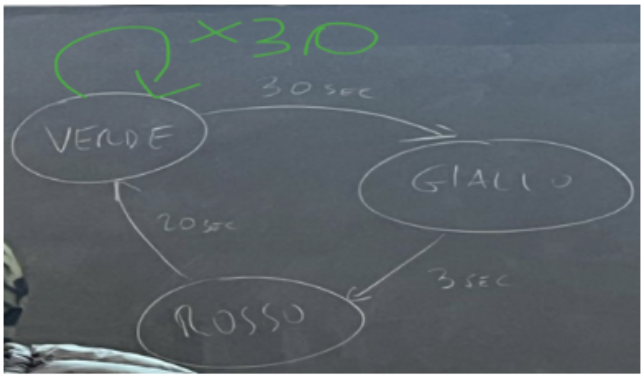
•Grafo diretto
•Nodi (stati) + archi (transizioni di stato)
•Etichette sui nodi esugli archi (input/output)

Grafo: struttura dati costituita da un insieme di nodi collegati tra loro da un insieme di archi (esempio: collegamenti stradali (archi) tra città (nodi)) che rappresenta le transizioni di stato (nodi=stato)

Esempio: considerando i tre stati di un semaforo (verde, giallo, rosso) si ha un passaggio da uno stato all'altro tramite un clock.

Come si fa a costruire queste tabelle di verità ?
Attraverso il formalismo degli automi a Stati finiti.
Cos'è l'automa a stati finiti?
È un semplicissimo formalismo che si usa continuamente, si usa un approccio predefinito, il grafo, una struttura dati costituita da un insieme di nodi e di archi, i nodi rappresentano gli stati o le entità mentre gli archi rappresentano il collegamento tra i nodi e variano di significato a seconda dello scenario e della natura del collegamento che viene valutata; ad esempio se vogliamo rappresentare i collegamenti stradali tra le città di una certa nazione, i nodi quindi rappresentano le città mentre gli archi le strade che collegano queste ultime, sugli archi come rappresentiamo la distanza tra le due città; la scelta delle strade e quindi del percorso minimo pero potrebbe essere influenzata anche da altri fattori (come il traffico) per cui la scelta dell'arco per eseguire la transizione (della strada per spostarsi da una città all'altra) potrebbe cambiare a seconda

dell'evenienza, di fatto tutti gli algoritmi che si utilizzano per calcolare i percorsi minimi fra due città si basano su una rappresentazione simile del tessuto stradale.
Gli archi possono essere bidirezionali (non diretti perché è come se non fossero direzionati) oppure possono avere una sola direzione (diretti).
A noi pero in realtà tra questi interessano pochi concetti, ovvero il concetto di nodo che rappresenta lo stato o un entità, il concetto di arco che rappresenta il collegamento e quindi la relazione tra i nodi, e l'etichetta che nei nodi definisce l'output di quello stato mentre sugli archi l'input che determina la transizione.
Quindi cos'è un automa a stati finiti ?
È di fatto un modo per modellare e rappresentare il funzionamento di un programma, che non fa uso dei formalismi del codice, ma fa so del grafo e rappresenta tramite questo le transizioni di stato.
Un esempio molto semplice per rappresentare praticamente questo concetto è il semaforo, esso ha normalmente 3 nodi ovvero i suoi stati: verde, giallo e rosso; ha altrettanti archi quindi delle frecce (DIREZIONATO poiché si può passare solo da uno stato ad un altro determinato stato in maniera ordinata): verde—> giallo, giallo —> rosso e rosso —> verde.
Come si passa da uno stato all'altro ?
Tutto dipende da come è stato progettato, ovvero da come sono definite le condizioni di transizione, ad esempio si passa da verde a giallo dopo 30 secondi che è stato verde, si passa da giallo a rosso dopo 3 secondi per esempio, si passa dal rosso al verde diciamo dopo altri i 20 secondi.
Come vedete abbiamo definito un semplicissimo funzionamento di un semaforo in cui i tre stati sono questi che abbiamo definito, e si può passare solo da verde a giallo, da giallo a rosso e da rosso a verde, e sono state definite le condizioni che causano le transizioni di Stato.
Cos'è fondamentale in un automa ?
Definire le condizioni di transizione di stato, cioè di passaggio con uno stato allo stato successivo, ora questo è un automa estremamente semplice e volendo si potrebbero ipotizzare altre condizioni e stati (fa esempi con semafori lampeggianti o con più colori accesi a definire una condizione di avvertimento per la transizione che sta per avvenire, ad esempio sta per diventare verde o rosso).



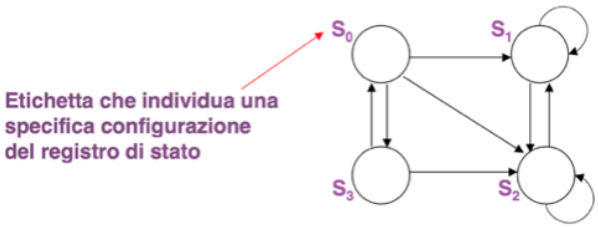
Ci possono essere anche delle frecce che vanno da un nodo a sé stesso. Ciò vuol dire che lo stato non cambia (c'è una condizione che si rimane nello stesso lato).
Il clock è in grado di valutare la situazione una volta al secondo, cosicché per lo stato verde valuterà la situazione per 30 volte prima di passare al giallo.

Ciò vale per tutti gli stati.
Notiamo che il seguente grafo presenta solo archi diretti, cioè unidirezionali.

Quindi qual'è il concetto importante ?
Questa cosa sarebbe possibile scriverla con un piccolo codice in Python, con alcuni if indentati, quindi potete capire che descrivere con degli automi dei sistemi di questo tipo è abbastanza intuitivo, per cui useremo questo formalismo per modellare i circuiti sequenziali.
In questo esempio vedete qui ci possono essere anche delle frecce che vanno dal nodo a se stesso, ciò vuol dire che lo stato non cambia cioè c'è una condizione per la quale si rimane nello stesso stato, ciò può essere utile se si vuole rivalutare la situazione di un sistema ogni istante di tempo, è chiaro che se io rivaluto la condizione molto spesso, per esempio ogni secondo valuto la condizione del semaforo ciò non vuol dire che io ogni secondo voglio cambiare lo stato del semaforo, magari valuto lo stato durante il tempo in cui è verde per cui ritornerò nello stato verde per 30 volte, finito il lasso di tempo in cui deve essere verde poi lo manderò nello stato giallo.
Questo è il motivo per cui spesso noi abbiamo degli stati che tornano in loro stessi, sono semplicemente dei modi con cui valuto la situazione e decido se rimanere su quello stesso stato.
Perché si chiamano automi a stati finiti ?
Si chiamano stati finiti perché gli stati sono appunto di numero finito, esistono e non ne parliamo qui anche automi che hanno un numero di stati non determinati, un'altra cosa molto importante da dire è che i collegamenti però possono essere diversi da uno stesso arco, possono esserci più uscite, e il loro comportamento può essere definito dal valore di un etichetta.

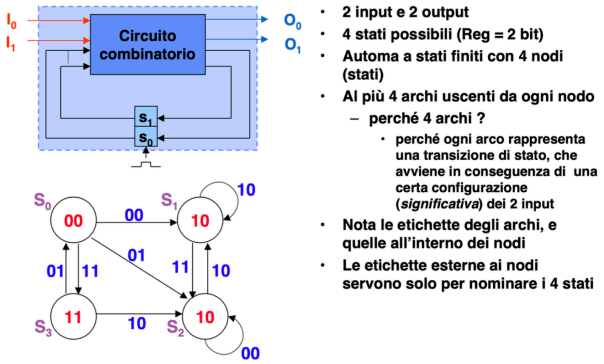
Ad esempio se voi avete una città, e da questa potrebbero esserci più strade allora voi potreste mettere una condizione per cui se una certa strada è bloccata prenderete un'altra strada; in effetti tutti i sistemi di routing stradale ormai mi fanno uso anche di queste informazioni perché la transizione potrebbe essere determinata da condizioni esterne.
Quello che noi vogliamo ottenere è un circuito combinatorio sincrono e guidato da un clock, in cui vogliamo determinare le funzioni output e Next State tenendo conto dei valori che stanno nel registro e dei valori di input.
Dobbiamo quindi costruire delle tabelle di verità in cui mettiamo in input le variabili che rappresentano i registri le variabili di input vere e proprie e come output mettiamo questi output qui e anche i successivi valori che vanno nei registri; quindi abbiamo due coppie di input due coppie di output.

Tornando ai circuiti sequenziali, un circuito di Moore presenta il seguente tipo di grafo:



•I **nodi** (in numero finito) corrispondono ai vari stati (configurazioni) assunti dagli elementi di memoria del circuito;
•Gli **archi** (in numero finito) corrispondono alle varie transizioni di stato, prodotte dallo stato di partenza, e all'output prodotto.

È importante ricordare che le etichette corrispondono a delle specifiche configurazioni (00,01, ecc.).

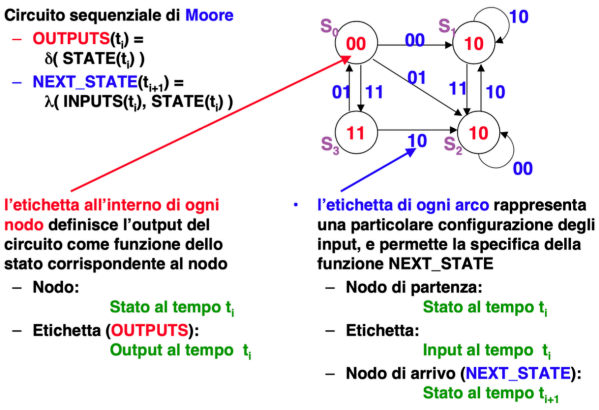


- 2 input e 2 output
- 4 stati possibili (Reg = 2 bit)
- Automa a stati finiti con 4 nodi (stati)
- Al più 4 archi uscenti da ogni nodo
  - perché 4 archi ?
    - perché ogni arco rappresenta una transizione di stato, che avviene in conseguenza di una certa configurazione (*significativa*) dei 2 input
- Nota le etichette degli archi, e quelle all'interno dei nodi
- Le etichette esterne ai nodi servono solo per nominare i 4 stati

Un circuito di Moore prevede la costruzione di un grafo con n nodi quanti sono gli stati (non per forza potenza di 2, ad esempio se sono 6 si usano sempre 8 stati ma due sono indifferenti); non c'è correlazione tra il numero di bit dello stato ed il nome dell'etichetta.

Circuiti sequenziali di Moore sono fatti così: noi abbiamo che l'output dipende esclusivamente dallo stato, il prossimo stato dipende dall'input attuale e dallo stato precedente, usiamo un automa stati finiti rappresentato un grafo diretto (quindi direzionato) nel quale gli archi hanno una freccia quindi un'unica direzione, nel quale i nodi del grafo rappresentano le possibili configurazioni dei registri (per esempio abbiamo un registro a 2 bit gli Stati saranno 4) e gli archi del grafo rappresentano la transizione di Stato cioè la condizione per cui si passa da uno stato all'altro; questa etichetta (nome sul nodo) è un'etichetta simbolica l'importante è che da qualche parte ci andiamo a segnare a cosa corrisponde, quindi questi nomi simbolici corrispondono a configurazione dei registi, di solito comunque si scelgono dei nomi significativi.
Allora immaginiamo di avere quattro stati ma non sappiamo cosa rappresentano, però diciamo che dentro i tondini degli stati vengono rappresentati i valori di output che devono essere prodotti in corrispondenza di questi stati, quindi per esempio in corrispondenza dello stato S0 l'output sarà 00, in corrispondenza di S1 10, in corrispondenza di S2 10 e in corrispondenza di S3 11, quindi questi valori esprimono gli output e non i valori immagazzinati nel registro; ciò che viene salvato nel registro deve essere salvato da un'apposita tabella in cui S0 corrisponde a 00, S1 a 01, S2 a 10 e S3 a 11; gli archi con questi etichette rappresentano il valore del prossimo stato cioè il valore che va memorizzato che registro allo stato successivo ovvero il Next State.
Cosa ci viene detto qui dall'automa ?
Assodando che gli archi rappresentano le transizioni di stato quindi servono a specificare come da uno stato si passano ad un altro in base alle condizioni di input; se noi ci troviamo nello stato S0 e in input arriva 00 e poi ci sposteremo verso lo stato S1 memorizzando nel next state il valore 10, se invece ci troviamo nello stato S0 e arriva in input il valore 01 noi ci sposteremo nello stato S2 e quindi salveremo in next state il valore 01, se invece in S0 dovesse arrivare in input il valore 11 allora ci sposteremo verso S3 e salveremo in Next State il valore 10; se invece siamo nello stato S1 e l'input è 10 rimaniamo in S1.

Le transizioni di Stato ci dicono verso quale stato ci spostiamo o se eventualmente rimaniamo sullo stesso in funzione dei valori di input.
Il numero di stati sono al più 2^n dove n rappresenta il numero di bit (2 bit al massimo 4 stati), questo lo dico perché in qualche caso ci potremo trovare in una situazione in cui abbiamo 6 stati, dunque ci servirebbero comunque 3 bit per poterli salvare ma ovviamente ci saranno delle combinazioni di registro che non servono.



I nodi sono collegati da archi. I numeri in rosso indicano l'output prodotto, i numeri in blu indicano l'input esterno, le etichette indicano la configurazione del registro di stato.
Gli archi rappresentano transizioni di stato, le quali nel modello Moore vengono prodotte sostanzialmente da due ingredienti: lo stato di partenza e l'input in un certo momento.
Questo esempio ci dice che se noi stiamo nello stato S0 con input in arrivo 00 ci spostiamo nello stato S1. Se siamo nello stato S0 con l'input di valore 01 ci spostiamo in uno stato diverso che in questo caso è S2.

Ad esempio, se sono in S0 registrerò in memoria il valore 00 e imposterò il nuovo stato di S1 (NEXT_STATE) al valore 10.
Il numero degli archi non può essere maggiore delle diverse combinazioni degli input. Il numero degli archi è compreso fra N (numero dei nodi) e le diverse combinazioni dell'input.

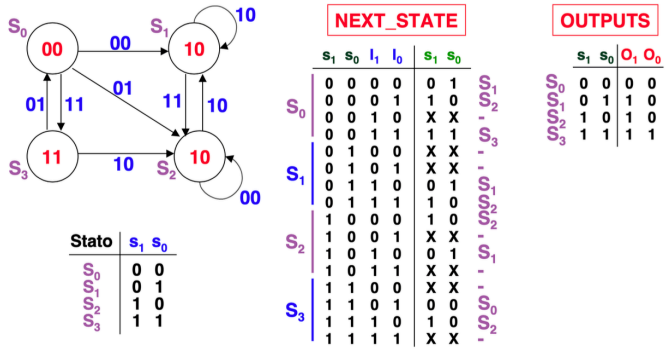
Stato	Registro a 2 bit
S_0	00
S_1	11
S_2	01
S_3	10

Dunque la prima cosa da definire è quanti stati ci sono, come memorizzarli nei registri, specificare l'output che il circuito produce in corrispondenza di quel particolare stato (l'output non per forza corrisponde con il numero di bit degli stati), specificare gli archi quindi le possibili transizioni di stato che possono avvenire in corrispondenza di una particolare sequenza di input
Come facciamo a sapere dove ci troviamo?
Essendo un circuito con memoria in uno stato a seconda dei valori dei registri, quindi se nel registro c'è il valore 00 che rappresenta ad esempio lo stato S0 vuol dire che noi ci troviamo all'interno dello stato S0 e cosi via.
Come faccio a capire dove si trova il circuito in questo momento ?
Banalmente vado a vedere cosa si trova all'interno del registro, se c'è scritto 01 significa che siamo in S2.
Questo è lo stesso automa di prima con alcuni commenti, è un esempio legato a questo circuito nel quale abbiamo due input I0 e I1 e due output O0 e O1.
Come si capisce che abbiamo due ingressi e due uscite?
Si capisce guardando l'automa.
Quanti sono i possibili archi presenti nel sistema ?
Se abbiamo 4 stati da ogni stato si può andare al più in quattro stati, perché si può andare verso se stesso e verso gli altri tre, quindi il numero degli archi è pari al più al numero dei nodi, quindi in questo caso abbiamo quattro archi.
Qui si ribadisce il concetto che le etichette esterne servono solamente per chiamare quattro stati (S0,S1,S2,S3), ma poi separatamente da qualche parte dobbiamo segnarci a cosa corrispondono questi stati rispetto a dei valori del registro.
In questo caso casualmente abbiamo un numero di stati che corrisponde casualmente a 2 elevato il numero degli input chiaramente il numero di archi massimi è 4, ma il numero degli archi non può essere maggiore al numero delle combinazioni degli input, faccio un esempio io avessi 16 stati e solamente quattro possibili valori di input il numero di archi in uscita era massimo 4, quello che si può dire del numero degli archi è il massimo fra n (numero dei nodi) e le diverse combinazioni di input.

Esempio completo

I valori dello stato in output non specificati nella tabella NEXT- STATE (X X) corrispondono a configurazioni degli input che non dovrebbero mai verificarsi.
È necessario generare due tabelle di verità:

- NEXT-STATE (quella tabella che ci dice come passare da uno stato al successivo)
- OUTPUTS (ci dice qual è il risultato prodotto in uscita dal circuito).



NEXT_STATE

s1	s0	I1	I0	s1	s0	
0	0	0	0	0	1	S1
0	0	0	1	1	0	S2
0	0	1	0	X	X	-
0	0	1	1	1	1	S3
0	1	0	0	X	X	-
0	1	0	1	X	X	-
0	1	1	0	0	1	S1
0	1	1	1	1	0	S2
1	0	0	0	1	0	S2
1	0	0	1	X	X	-
1	0	1	0	0	1	S1
1	0	1	1	X	X	-
1	1	0	0	X	X	-
1	1	0	1	0	0	S0
1	1	1	0	1	0	S2
1	1	1	1	X	X	-

OUTPUTS

	s1	s0	O1	O0
S0	0	0	0	0
S1	0	1	1	0
S2	1	0	1	0
S3	1	1	1	1

Il prossimo stato dipende, in questo caso, da 4 variabili (S1,S0, I1,I2). Per comodità, si tende a costruire la tabella in modo che 4 righe in successione indichino uno stesso stato.
Ad esempio, se si è nello stato S0 (cui configurazione è S1=0, S0=0) e si ha in input i valori I1=0, I2=0 si ha 01 che corrisponde al nuovo stato S1.

L'output prodotto sarà così 10 (valore in rosso nel nodo S1).
La tabella OUTPUT sostanzialmente ci deve mappare il contenuto dei nodi sull'output.
Tali tabelle di verità possono essere trasformate in mappe di Karnaugh per costruire successivamente il circuito.

I valori dello stato in output non specificati nella tabella NEXT_STATE (s_1 s_0 = X X) corrispondono a transizioni che non dovrebbero mai verificarsi

- **ovvero, le corrispondenti configurazioni degli input non dovrebbe mai verificarsi**

Facciamo l'esempio completo, supponiamo di avere questo automa in cui ci sono quattro stati, nei tondini ci sono i valori di output e sugli archi i possibili valori di input che determinano il passaggio dopo sta tra l'altro.
Come facciamo a rappresentare tutta sta roba sotto forma di tabelle di verità ?
Perché voglio fare le tabelle di verità ?
Perché delle tabelle di verità potrò costruirci il circuito che abbiamo già imparato a fare; allora noi dobbiamo generare due tabelle di verità: next state e outputs, next state è quella tabella che ci dice come passare da uno stato al successivo, l'output ci quale risultato viene prodotto in uscita dal circuito.
La prima cosa che dobbiamo fare è: devo segnare in una apposita tabella la corrispondenza tra l'etichetta e i valori memorizzati nel registro, ad esempio posso stabilire lo stato S0 rappresenta nel registro 00, S1 come 01, S2 come 10 ed S3 come 11, questa è una scelta assolutamente libera.
Dopo dovrò costruire la tabella di next state, il prossimo stato dipende in questo esempio dal quattro valor, due sono i valori di registro e due che sono i valori di ingresso, quindi dipende da S0 e S1 e da I0 e I1, S0 e S1 sono i valori successivi di next state (i valori di registro del prossimo stato espressi in bit separati).
Il modo più comodo con cui si rappresentare la bella di verità in questi casi è enumerare prima tutti i valori di S0 e S1, quindi le prime due colonne riportano tutti i possibili valori a partire da uno stato (quattro righe per volta: S0,S1,S2,S3), poi andiamo ad elencare i possibili valori di ingresso, ed infine i valori di uscita ma i valori di uscita che possiamo dedurre dall'automa (esempio: la prima riga va letta nello stato S0 [con valore S0 = 0 e S1 = 0 che rappresentano i bit riportati in

registro di questo stato] quando arriva l'input 00 [quindi I0 = 0 e I1 = 1] andrà poi nello stato S1 [con valore di registro S0 = 0 e S1 = 1], per non confonderci riportiamo anche l'etichetta dello stato).
Sto rappresentando questo automa sotto forma di tabella, gli output sono rappresentati da S1 e S0 perché questa è la tabella next state, quindi l'output che sto calcolando qui non è altro che il prossimo valore che deve assumere il registro, ci sarà poi un'altra tabella che si chiama Outputs che invece memorizza l'output complessivo del circuito combinatorio.
Gli input non previsti in un determinato stato vengono poi riportati sotto forma indeterminata di output (indicata con la X o - nella tabella).
Vediamo meglio la tabella outputs, sostanzialmente ci deve mappare il contenuto dei nodi sull'output, ma in questo caso è banale perché noi abbiamo un mapping uno ad uno fra uno stato e il corrispondente output, possono anche esserci due Stati che producono lo stesso output, in questo caso il registro corrisponde quasi sempre con il valore di output ma non è assolutamente una regola di fatto si può osservare l'incongruenza nello stato S1.

Deriviamo mappe di Karnaugh che generano NEXT_STATE

- **due mappe, una per ognuna delle 2 variabili s_0 e s_1**
- **equazione logica in forma SP ottimizzata, e circuito logico (STATE) a 2 livelli**

Deriviamo le mappe di Karnaugh che generano OUTPUTS

- **due mappe, una per ognuna delle 2 variabili O_0 e O_1**
- **equazione logica in forma SP ottimizzata, e circuito logico (OUT) a 2 livelli**

Una volta che abbiamo derivato queste tabelle dobbiamo costruire le mappe di Karnaugh, facciamo una mappatura per next state ed una per Outputs, ricordate che necessitiamo di una mappa per ogni output; quindi per il circuito corrispondente al next state dobbiamo fare due mappe di Karnaugh, la prima che c'ha quattro variabili ed anche la seconda e quindi ci facciamo i circuiti per S0 ed S1, per outputs il discorso è lo stesso anche se sono delle mappe più piccoline, anche se in realtà potremmo evitare di farle e scrivere direttamente la forma canonica.
Sintetizzate le mappe non ci resta che formare i due circuiti, Outputs riceve in ingresso S1 e S0 mentre Next state prende in ingresso S0,S1,I0,I1.