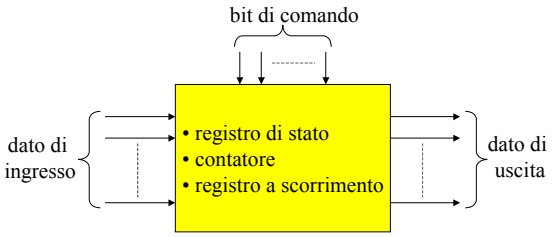
7.3 – Registri e Contatori

Per fare macchine complesse occorrono componenti complessi.

Nelle famiglie logiche, all'aumentare del livello di integrazione, sono stati di conseguenza resi disponibili circuiti sequenziali sincroni di sempre maggiore complessità interna.

Caratteristica comune è la **flessibilità d'impiego**. A tal fine ognuno di questi moduli ha diversi possibili comportamenti, tra cui il progettista sceglie di volta in volta quello di suo interesse attribuendo valori appropriati ad un certo numero di **bit di comando** resi disponibili all'esterno.

Moduli sequenziali programmabili



Registri, **contatori** e **registri a scorrimento** sono stati i primi componenti con queste caratteristiche. Oggi sono a loro volta inseriti all'interno di componenti ancora più complessi, ma ciò non elimina la necessità di saperne gestire bene il comportamento a livello di progetto logico.

Il registro di stato ed i comandi WE, RES

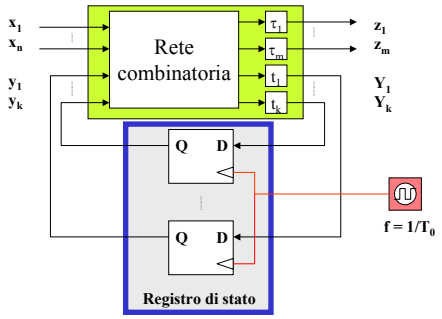
Per chiudere le retroazioni di una rete sincrona occorrono in generale diversi flip-flop, tutti azionati dallo stesso clock.

Un componente complesso decisamente utile per semplificare la struttura di ogni progetto è dunque un modulo sequenziale, detto **registro**, che contiene già al suo interno un certo numero di flip-flop disposti in parallelo.

ESEMPIO – I registri resi disponibili nelle famiglie logiche hanno racchiuso dapprima 2 flip-flop e poi, in rapida sequenza, 4, 8, 16 e 32.

La struttura degli attuali processori integrati su larga scala contiene registri di 64 bit.

Il registro per lo stato interno



Un registro è ancora più utile se contiene anche una parte combinatoria in grado di fargli assumere, tramite comandi esterni, comportamenti diversi ad istanti diversi.

Per individuare quali comportamenti sono in generale richiesti ad un registro, consideriamo una situazione tipica per il progetto architettonico: l'uso di una rete sincrona allocata sul **Data Path** per compiere, a partire da un certo istante, una certa elaborazione sulla sequenza dei dati d'ingresso.

ESEMPI- Conversione di un dato da forma seriale a forma parallela. Addizione di 100 dati alloggiati in una memoria.

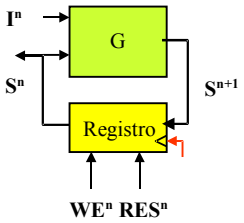
Il **Controller** deve imporre a tale rete tre modalità di funzionamento:

- **fase di inizializzazione** - all'inizio dei calcoli il registro di stato interno deve contenere un valore appropriato (spesso è utile la configurazione di tutti "zeri");
- **fase di aggiornamento -** al termine di ogni successivo intervallo elementare di elaborazione il registro di stato interno deve ricevere il valore calcolato dalla rete combinatoria;
- **fase di memorizzazione -** al termine dell'intera elaborazione il registro di stato interno deve mantenere l'ultimo valore ricevuto e questo fino a quando tale risultato finale non sarà prelevato dall'unità che ha commissionato il calcolo.

La gestione delle tre fasi diventa molto semplice se il registro di stato è dotato di due bit di comando, che nel seguito indicheremo con WE (*Write Enable)* e con RES(*RESet*).

La tabella di figura si riferisce al caso usuale di RES **prioritario** rispetto a WE: quando RES è attivo (fase di inizializzazione), il contenuto del registro è fissato e poi mantenuto a "zero". Quando RES è disattivo e WE è attivo (fase di aggiornamento), il registro incamera lo stato futuro calcolato dalla rete combinatoria. Quando infine RES e WE sono disattivi (fase di memorizzazione) il registro mantiene il risultato finale dei calcoli.

I comandi WE e RES



WE^n	RES^n	*fase*	*comportamento*
-	1	inizializzazione	$S^{n+1} = 0$
1	0	aggiornamento	$S^{n+1} = G(S^n, I^n)$
0	0	memorizzazione	$S^{n+1} = S^n$

La realizzazione dei comandi è semplice se, come è ragionevole pensare, la loro azione è sincrona. Non si pensi cioè che nell'istante in cui si attiva il comando RES, ad esempio, le uscite del registro debbano immediatamente azzerarsi, ma piuttosto che in un dato intervallo di clock il valore assunto da WE, RES e dai dati ingresso determina il valore dello stato futuro, che le uscite assumeranno al battito del clock e manterranno per il periodo successivo.

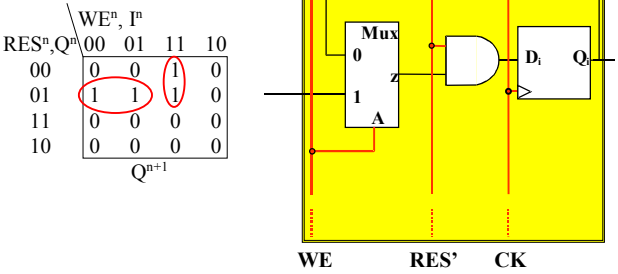
CASO DI STUDIO - Consideriamo la **i-esima cella** di un registro con flip-flop D e vediamo cosa occorre per disporre all'esterno dei comandi WE, RES. Nella parte sinistra di figura è riportata la tabella delle transizioni; impiegandola come mappa si ottiene:

$$D_i^n = Q_i^{n+1} = (RES'.WE.I_i + RES'.WE'.Q_i)^n$$
$$= (RES'.(WE.I_i + WE'.Q_i))^n$$

La parte combinatoria è dunque formata da un **selettore a due vie** seguito da un **and**.

Il **selettore** è azionato dal comando WE: quando **WE = 0**, in uscita si ha $\mathbf{Q_i}$ (fase di memorizzazione); quando **WE = 1**, in uscita si ha $\mathbf{I_i}$ (fase di aggiornamento). L'**and** è azionato da RES: **RES = 1** genera **0** sull'ingresso D del flip-flop (fase di inizializzazione) per qualsiasi valore di WE; in caso contrario D riceve il valore d'uscita del Mux.

I comandi WE e RES (Registro con ff D)



CASO DI STUDIO - Consideriamo ora la **i-esima cella** di un registro con flip-flop JK.

Modificando la mappa della figura precedente dapprima con le regole del comando J e poi con quelle del comando K (v. pag. 135) si ottengono le due mappe di figura.

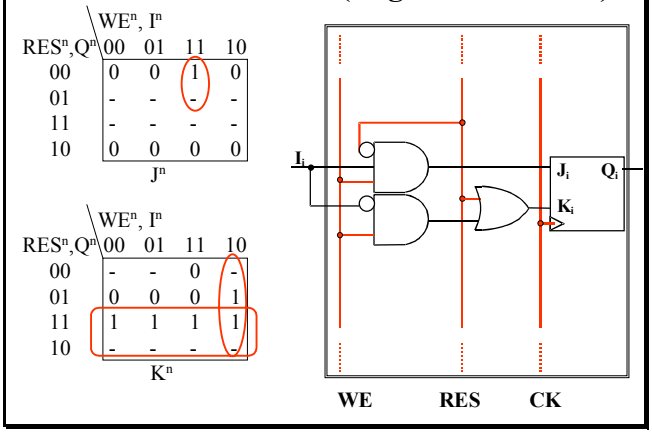
La copertura minima degli "uni" porta ad individuare le seguenti espressioni:

$$J_i^n = RES'.WE.I_i$$
$$K_i = RES + WE.I_i'$$

La realizzazione è dunque ancora più semplice di quella richiesta dal flip-flop D.

A questo proposito si prenda nota che per mantenere un bit in un flip-flop D occorre un Mux ed una retroazione; nel "più potente" flip-flop JK è invece sufficiente imporre 0 su entrambi gli ingressi.

I comandi WE e RES (Registro con ff JK)



APPROFONDIMENTO - Un caso particolarmente importante di registro di stato è il cosiddetto **accumulatore**, un registro presente nell'unità di elaborazione di tutti i calcolatori.

Ogni **istruzione di elaborazione** del linguaggio di macchina, una volta in esecuzione, inserisce nell'accumulatore il risultato di una **operazione**, o aritmetica o logica, con uno o con due operandi.

Le operazioni "**ad un operando**" consentono ad esempio o di incrementare/decrementare il numero contenuto nell'accumulatore, o di sostituirlo con una certa costante, o di complementare tutti i bit, o di farli scorrere verso destra/sinistra.

Le operazioni "**a due operandi**" elaborano il contenuto dell'accumulatore con un dato proveniente o da un registro del processore, o da una cella della memoria principale, o da un'interfaccia di I/O.

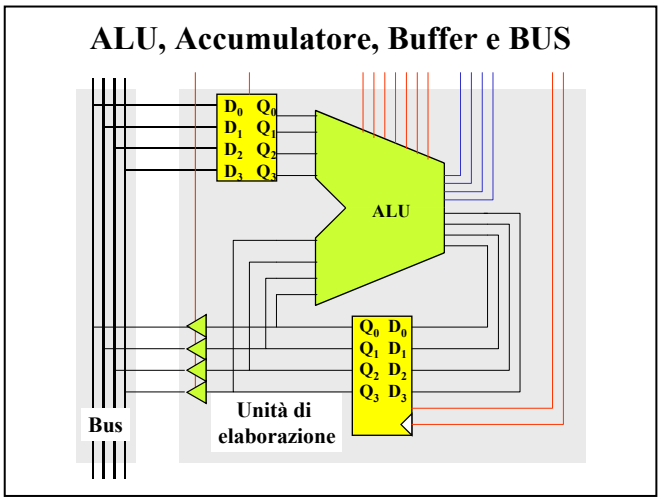
Operazioni aritmetiche tipiche sono l'addizione e la sottrazione; operazioni logiche tipiche sono l'AND, l'OR, l'EX-OR tra i bit di eguale posizione nei due operandi.

Quanto detto richiede che il registro accumulatore sia posto in retroazione ad una rete combinatoria programmabile, la **ALU** (*Arithmetic and Logic Unit*), a cui l'unità di controllo, tramite un certo numero di bit, impone di volta in volta di eseguire l'operazione specificata dall'istruzione in esecuzione; l'unità di controllo invia anche il clock ed il WE all'accumulatore, per stabilire quando e se deve avvenire la scrittura del risultato dell'operazione.

A titolo d'esempio, in figura è riportata la struttura di un'unità che elabora e memorizza dati di 4 bit, avvalendosi di un **bus** per scambiarli con l'esterno.

Si noti il particolare simbolo logico usualmente impiegato per rappresentare una ALU. Di norma tale rete combinatoria fornisce sia i bit del risultato da alloggiare nell'accumulatore (in figura i quattro più in basso), sia un certo numero di bit, detti **flag** (in figura i quattro più in alto), che consentono all'Elaborazione di **notificare** singolarmente al Controllo la presenza o meno di una certa proprietà del risultato (tutti i bit a zero, segno positivo, risultato più grande di "15", numero pari di bit con valore uno).

Si noti anche che la ALU di figura, ricevendo dal Controllo 7 bit di comando, ha potenzialmente la capacità di eseguire al suo interno 128 operazioni diverse; di norma sono però meno.



Si notino infine i due comandi ed i componenti primitivi impiegati dall'unità di controllo per gestire il trasferimento bi-direzionale dei dati. L'attivazione del comando di scrittura del **registro buffer** consente al Controllo di memorizzare all'interno dell'unità di elaborazione l'operando che ha prelevato, tramite il bus, da un'altra unità del Data Path. L'attivazione del comando degli **amplificatori a tre stati** disposti a valle dell'accumulatore consente, quando è richiesto, di inoltrare sul bus il risultato conseguito fino a quel momento dall'unità di elaborazione.

Il contatore binario

Sono dette **contatori** le reti sequenziali sincrone il cui grafo presenta un **ciclo** contenente tutti, o quasi tutti, gli stati interni.

Aspetti importanti per la struttura ed il comportamento di un contatore sono il **n° di stati** appartenenti al ciclo, detto **base di conteggio**, e la **codifica degli stati**.

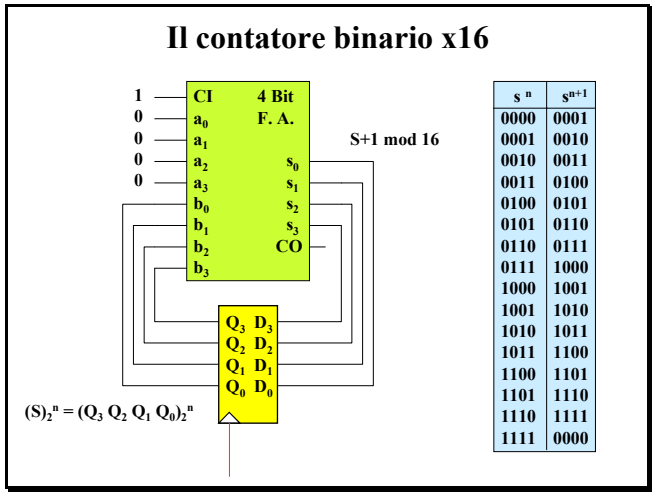
Il caso di maggior interesse è quello del **contatore binario**, in cui tutti gli stati appartengono al ciclo (un contatore binario con **k** flip-flop ha dunque base $\mathbf{2^k}$) ed in cui stati successivi sono codificati da numeri in base 2 che differiscono di un'unità.

In figura è riproposta la struttura vista nel cap.3: per contare per 16 ci si può avvalere di un sommatore per numeri di 4 bit retroazionato da un registro contenente 4 flip-flop D.

A lato è indicata la tabella delle transizioni. Una volta denotato con S il numero binario che costituisce lo stato interno, il comportamento può anche essere descritto con una formula:

$$S^{n+1} = (S+1 \bmod 16)^n$$

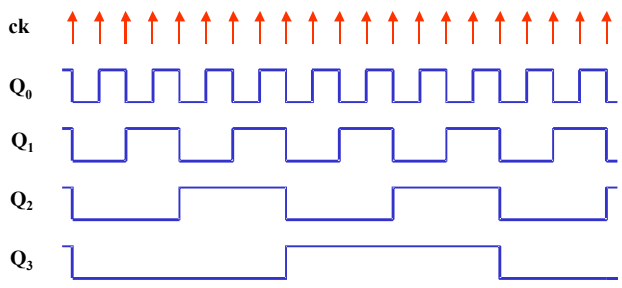
Si noti che l'incremento di S è fatto modulo 16: ogniqualvolta si raggiunge 15 (configurazione 1111) si riparte infatti da 0 (configurazione 0000).



L'uso del sistema binario per numerare i successivi stati interni del ciclo di conteggio attribuisce proprietà notevoli alle forme d'onda dei segnali di stato:

- tutte le uscite dei flip-flop sono onde quadre,
- l'uscita Q_i del flip-flop che memorizza il bit di peso 2^i ha un **periodo doppio** di quella presente su Q_{i-1}, l'uscita del flip-flop ove è memorizzato il bit di peso 2^{i-1},
- l'uscita Q_0 del flip-flop che memorizza il bit di peso 2^0 divide per 2 la frequenza del clock.

Forme d'onda



Naturalmente sono possibili diverse strutture interne; ad esempio può sorgere il dubbio che l'uso di un sommatore sia una complicazione inutile, dato che uno degli operandi è sempre "zero" (l'incremento unitario dello stato ad ogni clock è garantito da CI = 1).

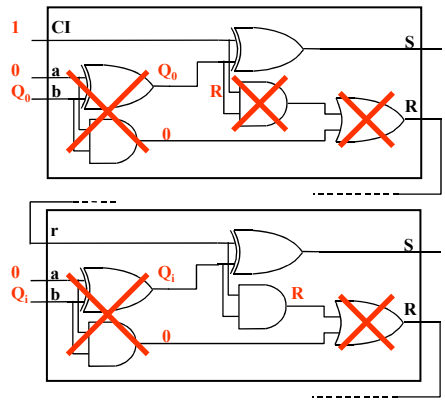
In figura è dimostrato graficamente che il dubbio è più che lecito.

Per ottenere Q_0^{n+1} da Q_0^n è sufficiente un EX-OR; per ottenere il riporto R_1 basta collegarsi a Q_0.

Anche tutte le altre celle possono essere semplificate: per ottenere Q_i^{n+1} è infatti sufficiente un EX-OR, questa volta tra Q_i^n e R_i^n; il calcolo di R_{i+1} richiede inoltre solo l'AND tra R_i e Q_i.

Una volta arrivati a queste conclusioni sulla parte combinatoria, si deve scegliere il tipo di flip-flop.

Una rete più semplice per l'incremento

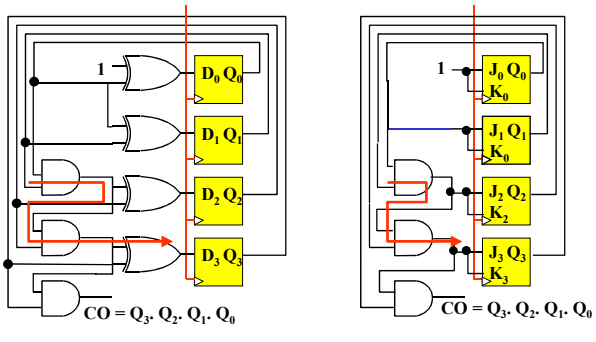


Con il flip-flop D la rete combinatoria necessaria è esattamente quella che abbiamo appena individuato (in realtà si potrebbe eliminare dalla prima cella l'EX-OR, collegando all'ingresso del primo flip-flop la sua uscita in forma complimentata).

Il percorso di elaborazione più lungo è formato da 3 gate e di ciò bisogna tener conto quando si stabilisce la frequenza di clock.

Con il flip-flop JK (o anche con il flip-flop T) si ha un duplice vantaggio: scompaiono tutti gli EX-OR ed il contatore può essere impiegato con una frequenza di clock più alta.

Periodo del clock e complessità della rete



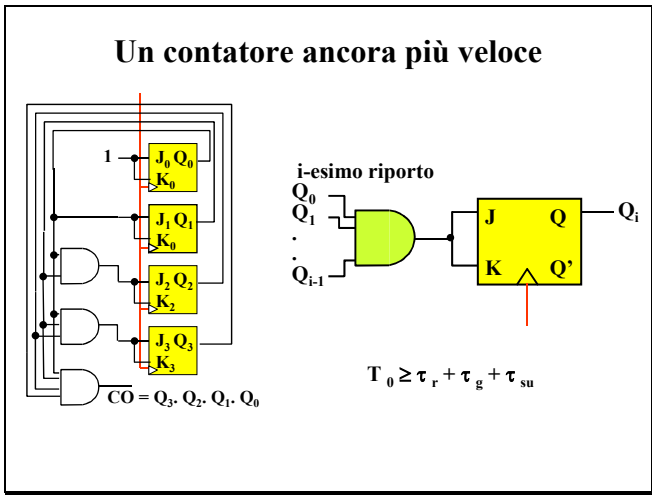
$T_0 \geq \tau_r + 3.\tau_g + \tau_{su}$ $T_0 \geq \tau_r + 2.\tau_g + \tau_{su}$

In entrambe le strutture è stato previsto anche il calcolo del prodotto logico (CO) che identifica la presenza nel registro della configurazione di tutti "uni": tra poco ne vedremo l'utilità.La discussione sulla struttura interna di un contatore non è infatti ancora finita.

Per la proprietà associativa del prodotto logico è possibile eliminare la disposizione in cascata degli AND prevista nelle precedenti strutture. Un modo diverso per dire la stessa cosa si basa sull'aritmetica binaria: nel caso di incremento unitario, ogni bit del numero da incrementare modifica il suo valore **se e solo se** tutti i bit di minor peso hanno valore 1.

L'i-esimo flip-flop JK di un contatore binario può dunque essere preceduto da un solo AND, su cui entrano tutti i Q_j con $j < i$.

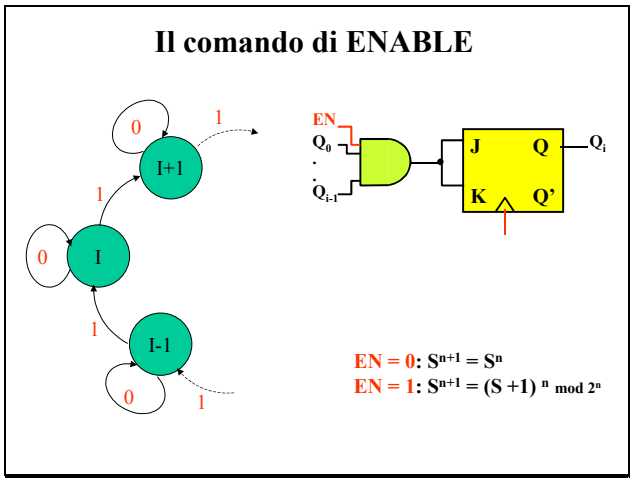
Questo nuovo schema è più veloce dei precedenti (un gate per percorso). La contro-partita è che il fan-in degli AND cresce con i.

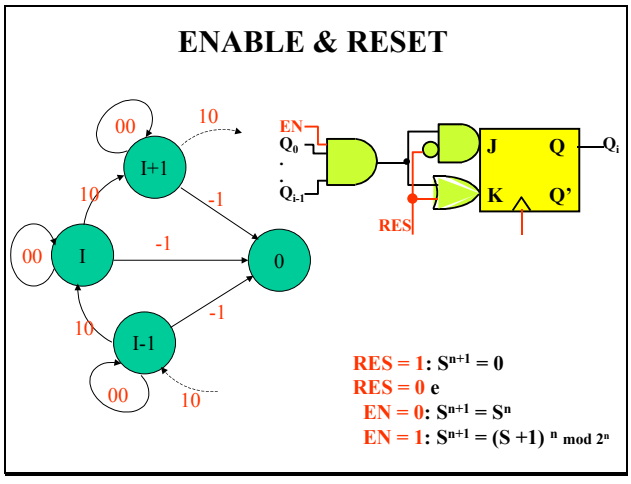


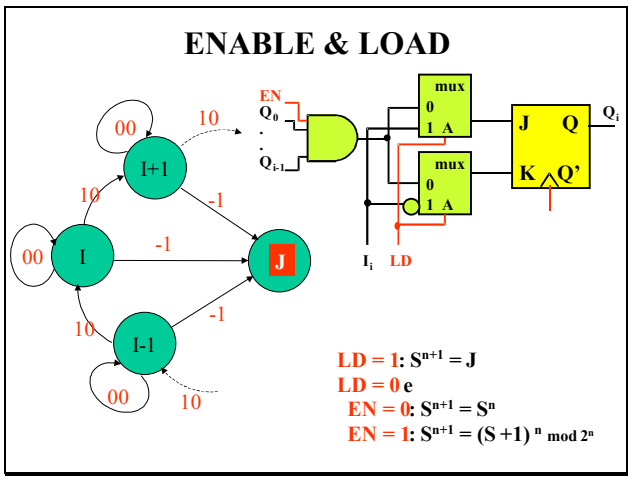
I comandi di un contatore

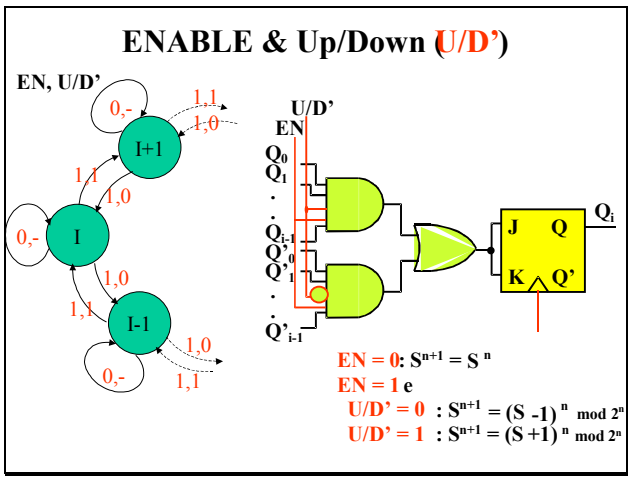
Per offrire flessibilità d'impiego, un circuito di conteggio è di solito dotato di comandi **sincroni** esterni. I comandi più frequentemente messi a disposizione del progettista logico, anche se non tutti contemporaneamente, sono l'ENABLE (che consente di abilitare o meno la modifica dello stato interno), il RESET (che consente di imporre la configurazione iniziale di tutti "zeri"), il LOAD (che consente di imporre dall'esterno una qualsiasi configurazione iniziale) e l'UP/DOWN (che consente di percorrere il ciclo di conteggio in un verso o nell'altro).

Nelle sottostanti figure sono evidenziate le celle di contatori binari in grado di interpretare il solo EN, EN & RES (RES prioritario), EN & LD (LD prioritario), EN & U/D' (EN prioritario).









Per i moduli di conteggio vale un'interessante regola di composizione: "**la disposizione in cascata di n moduli di conteggio (CO di ogni modulo connesso al EN del modulo successivo), rispettivamente con base B1, B2, .., BN, fornisce un contatore con base B = B1 x B2 x … x BN**"

CASO DI STUDIO – Si vuole ottenere un contatore x256 a partire da due contatori x16 dotati di comando di ENABLE e di uscita CO.

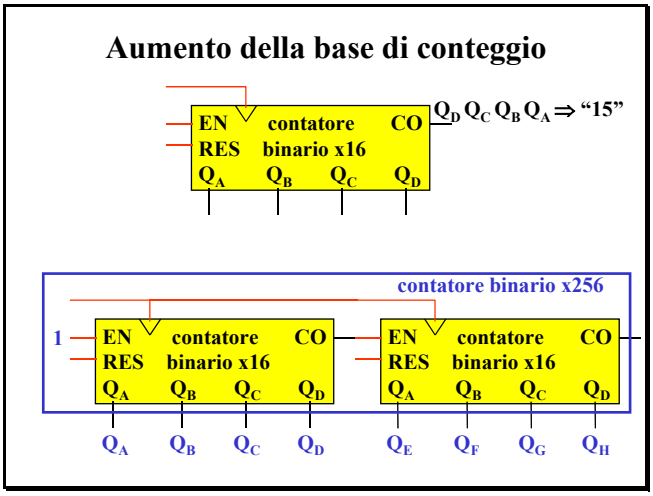
La soluzione è semplicissima: si azionano i due moduli di conteggio con lo stesso clock e si connette l'uscita CO di uno con l'ingresso di ENABLE dell'altro.

Il modulo di conteggio a sinistra contiene i 4 bit meno significativi, il modulo a destra quelli più significativi.

Se si vuole contare sempre si deve imporre 1 sull'ingresso EN del modulo a sinistra.

Se invece si vuole che anche il contatore complessivo abbia l'ENABLE, si deve prima controllare il data sheet dei moduli a disposizione:

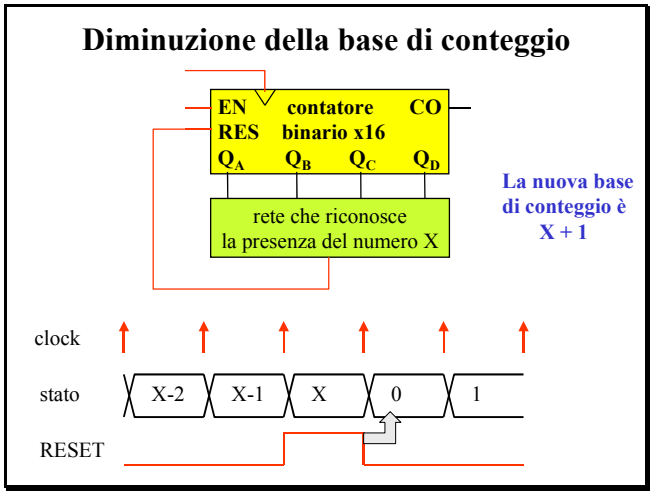
- se il calcolo interno del CO contiene già l'ENABLE lo schema va bene così,
- se non lo contiene, occorre aggiungerlo all'esterno predisponendo un AND tra l'uscita CO del modulo a sinistra e l'ingresso EN del modulo a destra.



Se si vuole disporre anche di un RESET complessivo, occorre collegare il segnale corrispondente agli ingressi RES di entrambi i moduli.

Dato un modulo di conteggio binario con base $\mathbf{2^k}$, dotato di EN e di RES, è possibile ottenere un contatore binario con base $\mathbf{B < 2^k}$.

CASO DI STUDIO – Consideriamo un contatore binario x16. Per ottenere la base X+1, con X<15, occorre realizzare una rete combinatoria che generi il valore 1 quando lo stato è **X** ed il valore 0 per ogni stato "più piccolo" (v. la forma d'onda indicata in figura); l'uscita di questa rete dovrà poi essere connessa all'ingresso RES del contatore. In questo modo, al termine dell'intervallo corrispondente allo stato **X** tutti i flip-flop vengono azzerati, e quindi viene continuamente ripetuto un ciclo di **X+1** stati. Se occorre un'abilitazione al conteggio, il corrispondente segnale deve essere sia connesso a EN, sia inviato alla rete di riconoscimento di X per generare un RESET **solo in presenza di abilitazione**. È ovvio che in caso contrario sarebbe impossibile bloccare il contatore sull'ultimo stato.



Il progetto della rete di riconoscimento può essere fatto in tre modi diversi.

Il primo prevede di impiegare un AND per realizzare il mintermine corrispondente a X.

Il secondo si basa sull'osservazione che tutti gli stati "più grandi" di X non potranno più appartenere al ciclo di conteggio e costituiranno quindi condizioni di indifferenza per la funzione di RESET. Nel caso di figura il comando può dunque essere fornito da un AND con soli due ingressi (rete di costo minimo per X = 10).

Si noti che il risultato può essere agevolmente esteso a qualsiasi X senza l'ausilio delle mappe: **il prodotto logico dei soli bit a "uno" in X riconosce o X o numeri più grandi.**

Il terzo modo prevede di progettare una rete che fornisca valore 1 in corrispondenza di X e di ogni altro numero più grande: ciò consente l'autoinizializzazione del contatore in un solo intervallo elementare.