

ELETTRONICA APPLICATA

1) CIRCUITI COMBINATORI

Ricordiamo che un **SEGNALE DIGITALE** è discreto nel tempo (campionato) e in ampiezza (quantizzato); qui il degrado legato al rumore è recuperabile proprio per i livelli (stati) logici di quantizzazione (rappresentati con tensioni).

I **CIRCUITI LOGICI** sono collegati ad una tensione positiva di **ALIMENTAZIONE** (V_{AL} oppure GND) e hanno dei **segnali binari** di ingresso e uscita (in forma seriale o parallela). Gli **STATI LOGICI** sono rappresentati da delle tensioni tale che (in questo corso):

- STATO 1 LOGICO (H) → livello elettrico alto, ovvero tensione $V_O = V_H = V_{AL}$;
- STATO 0 LOGICO (L) → livello elettrico basso, ovvero tensione $V_O = V_L = GND = 0V$.

Questo però non garantisce la **TOLLERANZA AL RUMORE**; per farlo, pongo una tensione di soglia (threshold) V_T tale che ho:

- STATO ALTO (H) se $V_{IN} > V_T$;
- STATO BASSO (L) se $V_{IN} < V_T$.

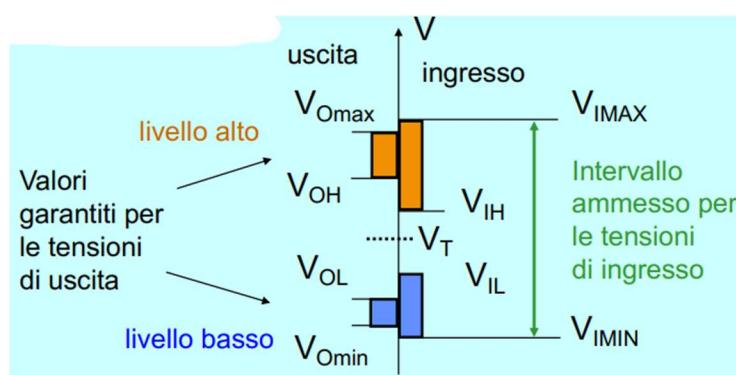
Dalle soglie si passa ai livelli:

- INGRESSO della porta:
 - STATO ALTO (H) se $V_O > V_{OH}$;
 - STATO BASSO (L) se $V_O < V_{OL}$.
- USCITA della porta:
 - STATO ALTO(H) se $V_{IN} > V_{IH}$;
 - STATO BASSO (L) se $V_{IN} < V_{IL}$.

Quindi abbiamo dei **RANGE DI USCITA** e dei **RANGE DI INGRESSO** (DINAMICA) per garantire il collegamento tra porte (COMPATIBILITÀ), che messi insieme mi danno come **CONDIZIONI DI COMPATIBILITÀ**:

$$V_{OL} < V_{IL} \wedge V_{OH} > V_{IH}$$

Inoltre per evitare di danneggiare il dispositivo, dovremo avere anche dei **limiti** sulla V_O ovvero V_{Omax} e V_{Omin} :

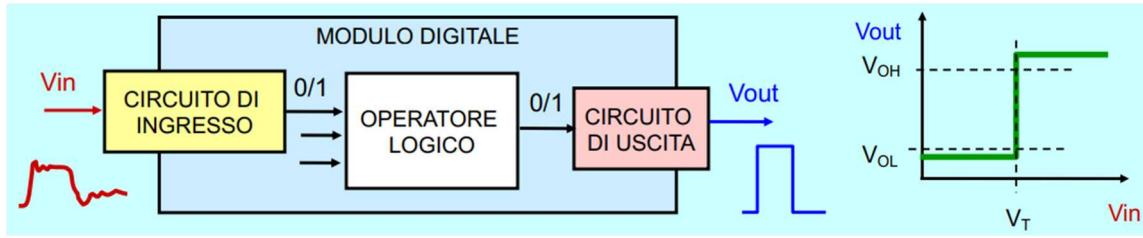


La differenza tra V_{OH} e V_{IH} , e la differenza tra V_{OL} e V_{IL} garantiscono che lo stato logico venga interpretato in modo corretto anche con rumore/disturbo; infatti questa differenza è il **MARGINE DI RUMORE** ("noise margin"):

$$NM_H = V_{OH} - V_{IH}$$

$$NM_L = V_{IL} - V_{OL}$$

Quindi ogni ingresso digitale si comporta come un **COMPARATORE**:



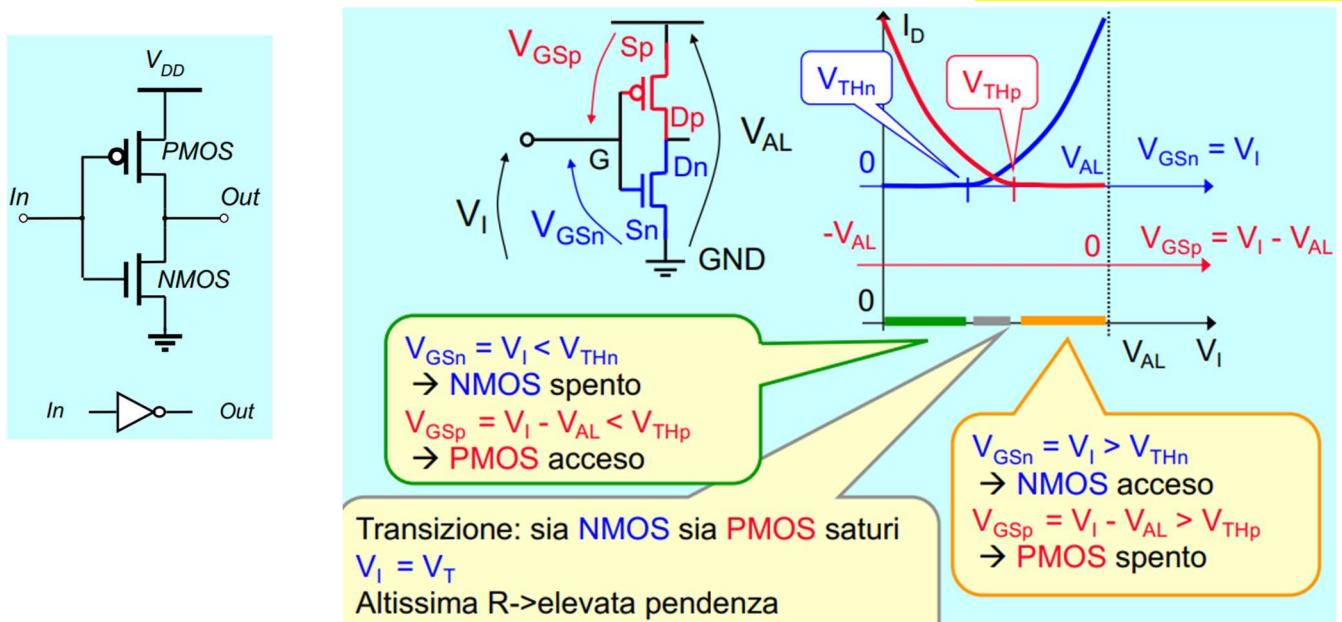
⚠ L'uscita generata ricorda che deve essere $V_{OUT} > V_{OH}$ e $V_{OUT} < V_{OL}$ (deve essere esterna, guarda grafico su)!

Nei circuiti logici uso dei **TRANSISTOR** per creare **PORTE LOGICHE** combinatorie; un transistor MOS è un interruttore non ideale con resistenza R_{ON} (nella regione di TRIONDO [ON]); quando sono nella regione di OFF ($V_{GS} < V_{TH}$) non sono proprio in un circuito aperto (passa della corrente molto piccola dipendente da V_{TH}).

Il circuito **INVERTITORE CMOS** (il più semplice) è un esempio di porta logica realizzata con 2 transistor: **1 PMOS** (che fa da interruttore verso V_{DD}) e **1 NMOS** (che fa da interruttore verso GND); in questo modo quando:

- $V_{IN} = 0$ (stato L), NMOS spento e PMOS lineare, quindi $V_{OUT} = V_{DD}$;
- $V_{IN} = V_{DD}$ (stato H), PMOS spento e NMOS lineare, quindi $V_{OUT} = 0$.

La transizione tra stato L e H avviene però ad una tensione di soglia; vediamo la **caratteristica dell'INVERTITORE**:

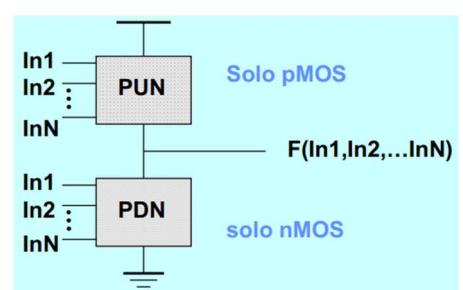


Nella zona di **TRANSIZIONE** (ovvero a metà tra uscita H e L), il risultato logico dipende da chi "tira di più" tra NMOS e PMOS. Posso definire questo così:

- $V_I > V_{IH} \rightarrow$ STATO LOGICO H
- $V_I < V_{IL} \rightarrow$ STATO LOGICO L
- $V_{IL} < V_I < V_{IH} \rightarrow$ TRANSIZIONE (STATO NON DEFINITO)

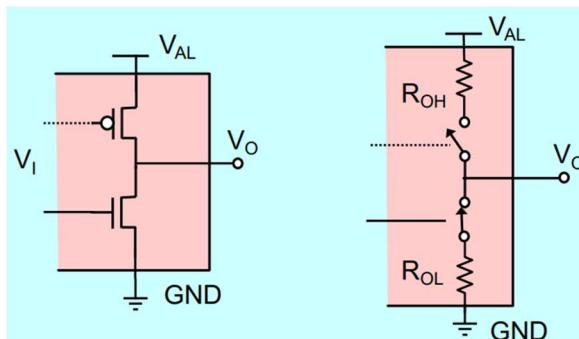
⚠ Sulle curve di questi grafici, se faccio le tangenti a 45°, trovo V_{IL} e V_{OH} sulla curva più a sinistra, e trovo V_{IH} e V_{OL} sulla curva più a destra!

Un'estensione dell'invertitore sono le **PORTE LOGICHE COMPLESSE CMOS**, dove ho una rete di **PULL-UP** (composta da n PMOS) e una rete di **PULL-DOWN** (composta da m NMOS) al posto dei singoli PMOS e NMOS; un esempio è la **PORTA NAND CMOS** (dove ho tanti NMOS in serie verso GND e tanti PMOS in parallelo verso V_{AL} [questo perché se almeno un ingresso è a 0, il NAND "agisce", mentre il risultato degli ingressi a 1 è indifferente]) e la **PORTA NOR CMOS** (dove ho tanti NMOS in parallelo verso GND e tanti PMOS in serie verso V_{AL} [questo perché se almeno un ingresso è a 1, il NOR "agisce", mentre il risultato degli ingressi a 0 è indifferente]).



⚠ Quindi il procedimento è realizzare una rete di interruttori, partendo dal pull-down e facendo l'operazione di AND/OR (serie/parallelo), da cui poi faccio il pull-up come complementare!

Vediamo ora i **PARAMETRI ELETTRICI STATICI E INTERFACCIAMENTO** di queste porte CMOS; i MOS di uscita hanno **resistenza equivalente R_O** che può essere diversa verso il GND (ovvero R_{OL}) e verso V_{AL} (ovvero R_{OH}). Ci servono le 2 resistenze per valutare i ritardi di propagazione e per capire qual è il valore massimo della corrente di uscita che il transistor può erogare (perché se ci fosse un'altra resistenza sul ramo di uscita, si aggiungerebbe un'altra tensione su V_O , quindi utile se V_O dipende da I_O):

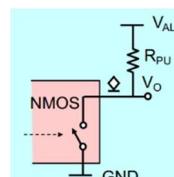
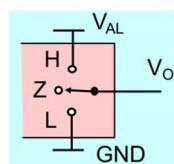
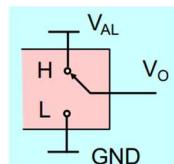


Alle condizioni di compatibilità sulle tensioni delle porte si aggiunge la **COMPATIBILITÀ SULLE CORRENTI** [⚠ ricorda che le correnti di uscita da una porta è definita sempre come entrante!]. Quindi riassumento, le condizioni su una V_O che dipende da I_O sono:

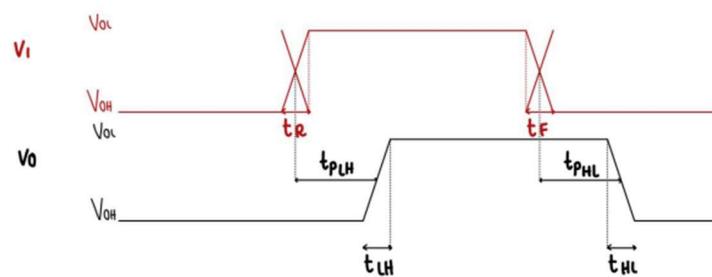
$$\begin{cases} V_O > V_{OH} \Leftrightarrow |I_O| < |I_{OH}| \\ V_O < V_{OL} \Leftrightarrow |I_O| < |I_{OL}| \\ V_{OL} < V_{IL} \\ V_{OH} > V_{IH} \end{cases}$$

Gli **STADI DI USCITA** delle porte logiche possono essere:

- **Totem Pole** = uscita di un circuito logico binario in cui la tensione di uscita dello stato alto (V_{OH}) rappresenta V_{AL} , mentre la tensione di uscita dello stato basso (V_{OL}) rappresenta il GND. Quindi interruttore con unico comando. Tempi di scarico e carico sono simili;
- **3-State** = analogo al Totem Pole, ma si aggiunge un'uscita Z, ovvero lo stato di alta impedenza (lo stato che dipende dalle altre parti del circuito). Questo modello prevede 2 switch: uno per lo stato logico di uscita e uno che abilita/disabilita l'uscita. Questo modulo può essere usato per creare un modulo di controllo per le abilitazioni esclusive (es. multiplexer). Quindi 2 comandi indipendenti. Tempi di scarico e carico sono simili;
- **Open Drain (Open Collector)** = realizzato con 1 solo NMOS verso GND (se NMOS chiuso \rightarrow uscita verso GND [$V_O = 0$]; se NMOS aperto \rightarrow uscita di alta impedenza portata a livello alto da una resistenza esterna detta di "pullup" R_{PU} [$V_O = V_{AL}$]). L'open drain si può usare per realizzare funzioni logiche cablate (es. OR cablato), collegando più uscite OD in parallelo [es. bus seriali a basso costo]. Quindi 1 solo interruttore. I tempi di salita sono più lunghi dei tempi di discesa.



Per quanto riguarda i **PARAMETRI DINAMICI**, vediamo che nei segnali reali la commutazione da uno stato logico ad un altro non è mai istantanea ma avviene con un ritardo; inoltre le variazioni di tensione e corrente in ogni modulo generano un altro ritardo. Questi ritardi combinati inducono **RITARDI DI PROPAGAZIONE** e **LIMITI ALLA VELOCITÀ OPERATIVA**. Si hanno quindi i **TEMPI DI TRANSIZIONE** (tempo di salita t_R e di discesa t_F) e i **RITARDI DI PROPAGAZIONE** (che invece indicano la propagazione tra ingresso alto-uscita bassa t_{PHL} e tra ingresso basso-uscita alta t_{PLH}).



Quindi legato a ciò vediamo i **RITARDI** (partendo dal presupposto che i MOS di ingresso si modellano con una capacità equivalente): il **RITARDO DI PROPAGAZIONE** è l'intervallo di tempo tra quando l'ingresso raggiunge il 50% e l'uscita raggiunge il 50%; il punto raggiunto al 50% vale:

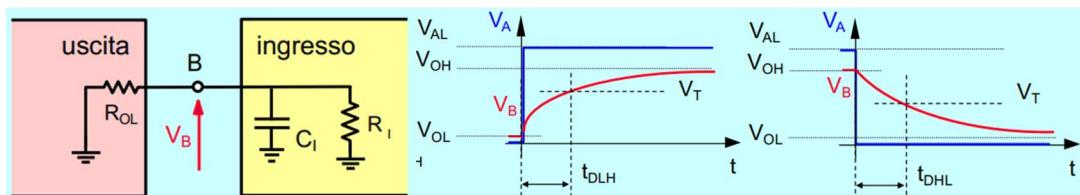
$$V_{50\%} = \frac{V_{OH} + V_{OL}}{2}$$

Il ritardo di propagazione basso-alto t_{PLH} e alto-basso t_{PHL} non sono necessariamente uguali e il **ritardo di propagazione complessivo** si calcola con il loro valor medio:

$$t_P = \frac{t_{PLH} + t_{PHL}}{2}$$

La formula sotto invece vale solo quando V_T è il valore di tensione trovato sia in salita che in discesa:

$$t_{PHL} = 0.69 R_{OL} C_I$$



Ora parliamo degli **EFFETTI DI CARICO**: oltre alla resistenza, anche il carico incide sulla costante di tempo τ . Aumentando il numero di carichi, aumenta la capacità totale e questo comporta un aumento di ritardo e tempi di transizione che causano a loro volta delle variazioni di stato multiple (a causa del rumore). Il massimo numero di ingressi collegabili ad un'uscita (**FANOUT**) è limitato dal carico capacitivo; il fanout dipende dalla compatibilità statica (correnti di ingresso e uscita, carichi) e dalla compatibilità dinamica (ritardi, t_R e t_F) [nei circuiti CMOS, essendo la corrente di ingresso quasi nulla, il fanout dipende solo dal carico capacitivo (compatibilità statica trascurabile)].

Vediamo degli **ESERCIZI** riguardo ciò:

ANALISI SEPARATAMENTE STATO ALTO e BASSO:

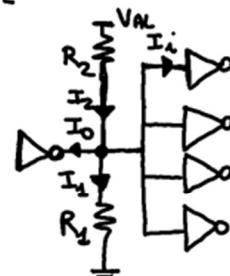
- ① **VERIFICA COMPATIBILITÀ TENSIONE** $L: V_{OL} < V_{IL}$; $H: V_{OH} > V_{IH}$
- ② **CALCOLA DUE CORRENTI RICHIESTE DALL'USCITA IN H e L**
- ③ **VERIFICA COMPATIBILITÀ CORRENTI** $L: I_{OL} < I_{IL}$; $H: I_{OH} > I_{IH}$

- ④ OUT: $V_{OL} = 0.4V$; $V_{OH} = 3V$; $I_{OL} = 4mA$; $I_{OH} = -1mA$
IN: $V_{IL} = 0.8V$; $V_{IH} = 2V$; $I_{IL} = -0.8mA$; $I_{IH} = 0.2mA$
→ Si vede l'effetto del carico $V_{DD} = 5V$ con:

- ⑤ i soli 4 ingressi (SENZA CARICO);
- ⑥ $R_1 = 10k\Omega$;
- ⑦ $R_2 = 1k\Omega$.

Le condizioni sono verificate: $\begin{cases} V_{OL} < V_{IL} \\ V_{OH} > V_{IH} \end{cases}$

- ⑧ $L: I_{OL} + 4I_L = 0.8mA > 0$ [USCITA PUÒ ASSORBIRE LA CORRENTE DEI 4 INGRESSI]
 $H: I_{OH} + 4I_H = -0.2mA < 0$ [USCITA PUÒ EROGARE LA CORRENTE AI 4 INGRESSI]
- Dunque il FANOUT MASSIMO È 5



⑨ $I_1 + 4I_L = \frac{V_{IL}}{R_1} + 4I_L = 0.08 - 3.2 = -3.12mA \rightarrow I_{OL} + I_1 + 4I_{IL} = 0.88mA$
(perciò R_1 aiuta lo stato logico L, assorbiendo corrente)

$I_1 + 4I_H = \frac{V_{IH}}{R_1} + 4I_H = 0.2 + 0.6 = 1mA \rightarrow I_{OH} + I_1 + 4I_{IH} = 0mA$
MARGINE MINIMO PER IL FUNZIONAMENTO

(perciò R_1 assorbe corrente, rendendo più difficile a I_{OH} il fornire corrente)

⑩ $I_2 - 4I_L = \frac{V_{AL} - V_{IL}}{R_2} - 4I_L = 4.2 + 3.2 = 7.4mA$; DATO CHE $I_{OL} = 4mA$,
NON PUESCHE AD ASSORBIRE LA CORRENTE ENOGATA

$I_{OH} - I_2 + 4I_L = -1 - 3 + 0.8 = -3.2mA$
(perciò c'è corrente erogata dal resistore aggiuntivo → aiuta l'uscita)

2) TROVARE t_{DLH} CON:

$$V_{OL} = 1V; V_{OH} = 3V; V_T = 2V; R_O = 200\Omega; C = 50\text{ pF}$$

$$V_{DD} = 4V; R_I = 1M\Omega$$

$$\rightarrow \tau = (R_I \parallel R_O) C \approx R_O C = 10\text{ ns}$$

$$\begin{cases} V_O < V_{OL} = 1V \rightarrow 0 < V_O < 1 \\ V_O > V_{OH} = 3V \rightarrow 3 < V_O < 4 \end{cases} \xrightarrow{V_{DD}}$$

DUNQUE NEI 4 CASI:

- $V(0) = 0V; V(\infty) = 3V$
- $V(0) = 0V; V(\infty) = 4V$
- $V(0) = 1V; V(\infty) = 3V$
- $V(0) = 1V; V(\infty) = 4V$

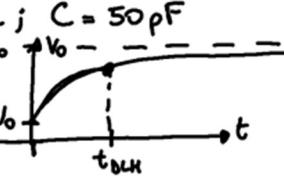
SAPENDO CHE $V(t) = (V_{\infty} - V_0)(1 - e^{-\frac{t}{\tau}}) + V_0$ [CONDENSATORE] E CHE:

$$V(t_{DLH}) = V_T \Leftrightarrow t_{DLH} = \tau \log \left(\frac{V_{\infty} - V_0}{V_{\infty} - V_T} \right)$$

ANORA NEI 4 CASI:

- | | |
|---|-------|
| • $V(0) = 0V; V(\infty) = 3V \rightarrow t_{DLH} = 10.98\text{ ns}$ | 66.6% |
| • $V(0) = 0V; V(\infty) = 4V \rightarrow t_{DLH} = 6.98\text{ ns}$ | 30% |
| • $V(0) = 1V; V(\infty) = 3V \rightarrow t_{DLH} = 6.93\text{ ns}$ | 30% |
| • $V(0) = 1V; V(\infty) = 4V \rightarrow t_{DLH} = 4.05\text{ ns}$ | 33.3% |

QUINDI t_{DLH} VARIA DA 4 ns A 11 ns



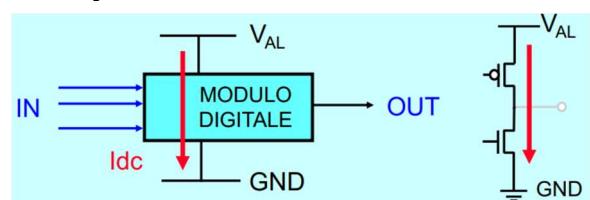
⚠ Per quanto riguarda i **SEGNALI DIGITALI DIFFERENZIALI**, questi non sono "single-ended", ma sono composti da una coppia di circuiti complementari: in questo modo è meno affetto da rumori/disturbi esterni.

Quindi riassumendo:

- **TENSIONI ALL'INTERFACCIA:**
 - decise dalle porte che pilotano (ovvero dagli stadi di uscita);
 - devono essere maggiori (a livello H) o minori (a livello L) della soglia delle porte pilotate.
- **CORRENTI NELLE INTERFACCIE:**
 - decise dalle porte pilotate (ovvero dagli stadi di ingresso) e dagli eventuali carichi (es. R_{PU});
 - sommandole in modulo, devono essere minori delle correnti massime in uscita (I_{OL}, I_{OH});
 - nel caso del CMOS, I_{IH} e I_{IL} sono trascurabili, quindi il fanout è limitato dai ritardi.

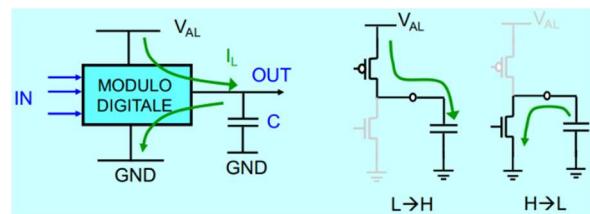
Ora guardiamo il **CONSUMO DI POTENZA**: ogni modulo richiede energia, usata per il funzionamento del modulo stesso, per i segnali esterni ed una parte viene trasformata in calore. L'energia richiesta viene **fornita attraverso la tensione di alimentazione** (V_{AL}), solitamente fornita in maniera costante (V_{AL} = costante). L'indicatore del consumo di energia è la **corrente assorbita** dall'alimentazione. Definiamo quindi:

- **POTENZA STATICÀ** = potenza assorbita in assenza di commutazione. L'assorbimento è quasi costante. Questa potenza varia in base alla temperatura e alla V_{AL} ; è visibile come una corrente continua (I_{DC}) tra V_{AL} e GND. Quindi dipende dalla tecnologia (dipende da V_{AL});



- **POTENZA DINAMICA** = potenza assorbita per eseguire una commutazione ($H \rightarrow L$ oppure $L \rightarrow H$). Dipende dalla tensione e dalla corrente (che carica o scarica la capacità in uscita, quindi che dipende dalla capacità del condensatore):

$$P_D = F \cdot C \cdot V^2$$



⚠ Da notare che F è la frequenza di un ciclo "carica-scarica"; quindi se mi danno la frequenza di clock, allora la frequenza di interesse equivale a 2 periodi di clock (1 clock = carica, 1 clock = scarica).

Quindi P_D dipende dalla tecnologia, ma soprattutto dal carico capacitivo.

Per ridurre la potenza dinamica, posso operare:

- riducendo la frequenza di commutazione F (usando algoritmi che richiedono meno commutazione per esempio);
- riducendo la capacità C (migliore tecnologia);
- riducendo l'escursione di tensione $V_H - V_L$.

Un parametro che rappresenta la potenza in funzione della tecnologia è il **PRODOTTO POTENZA-RITARDO**:

$$P_D \cdot T_P = k$$

In un circuito logico ideale potenza dissipata $P_D = 0$ e ritardo $T_P = 0$. Graficamente, le curve generate dal prodotto potenza-ritardo sono delle **iperboli** che rappresentano la proporzionalità inversa tra potenza e ritardo (moduli con ritardi più bassi consumano più energia e viceversa).

⚠ Nelle strutture CMOS:

- condizioni **statiche** (senza carico) $\rightarrow P_S = I_{OFF}V_{AL}$
- condizioni **dinamiche** $\rightarrow P_D = F C V_{AL}^2$

⚠ In prima approssimazione c'è solo un consumo dinamico, tuttavia nei circuiti recenti diventa significativo il consumo statico.

2) CIRCUITI BISTABILI

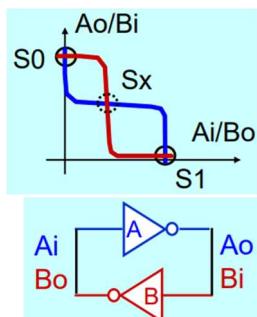
Ricordiamo che un circuito è definito combinatorio se le uscite in t_0 sono funzioni solo degli ingressi applicati nell'istante t_0 , mentre è **SEQUENZIALE** se le uscite sono funzioni degli ingressi correnti ma anche della storia precedente (contengono quindi elementi di memoria, ovvero i **FLIP-FLOP** [o **CIRCUITI BISTABILI**]).

Ci sono 2 modalità con cui un flip-flop può commutare lo stato:

- SINCRONA** \rightarrow con un clock che regola la temporizzazione;
- ASINCRONA** \rightarrow tramite singoli comandi.

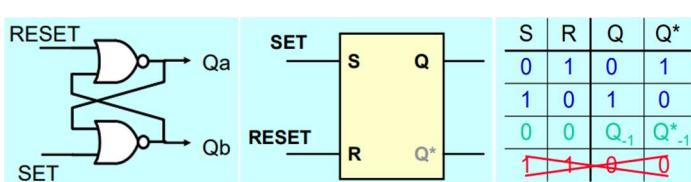
⚠ Il punto di funzionamento è l'incrocio delle transcaratteristiche, ovvero se ho i 2 stati **S0** (dove $Ao = H$, $Bo = L$) e **S1** (dove $Ao = L$, $Bo = H$), il loro punto di incrocio è un terzo stato **Sx** dove lo stato è instabile ("metastabile").

L'elemento alla base del flip-flop è l'**ANELLO INVERTENTE** (o anello di inverter), dove l'uscita di A è l'ingresso di B e viceversa.



Il flip-flop più semplice dal punto di vista funzionale è il **FLIP-FLOP SET RESET (FF-SR)**, che ha 2 ingressi [S = set, R = reset (o clear)], 2 porte NOR (potrebbe però essere realizzato anche con 2 porte NAND) e 2 uscite [Qa e Qb, complementari tra loro]. L'ingresso "set" porta l'uscita a 1, "reset" a 0; si possono verificare 4 stati:

- SET** $\rightarrow S = 1, R = 0$; $Q_a = 1, Q_b = 0$;
- RESET** $\rightarrow S = 0, R = 1$; $Q_a = 0, Q_b = 1$;
- MEMORIA** $\rightarrow S = 0, R = 0$; dipende dallo stato precedente;
- CONDIZIONE PROIBITA** $\rightarrow S = 1, R = 1$; uscite non complementari (non permesso).

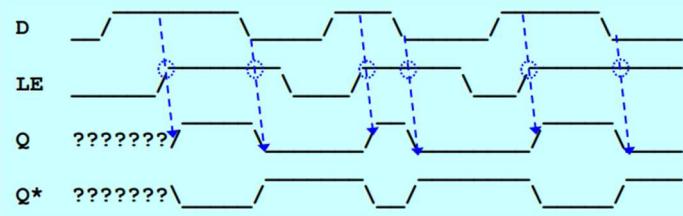


⚠ Il FF-SR si può fare anche con 2 porte NAND, ma gli stati ottenuti (set, reset, memoria e condizione proibita) sono ottenuti dagli ingressi inversi (set se S=0, R=1, etc...) [quindi la condizione proibita si verifica con l'ingresso S=0, R=0 e non con gli 1].

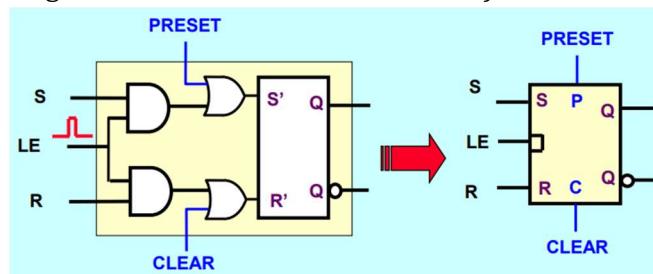
Il **LATCH D** è un dispositivo dove viene eliminata la condizione instabile del FF Set Reset: gli ingressi S e R vengono cortocircuitati con un inverter così da non avere 2 ingressi uguali; è un circuito asincrono, ovvero commuta tramite un comando (il comando **D**, che produce un set e un reset obbligatoriamente opposti). Il Latch D prevede un comando chiamato **LE (latch enable)** [un segnale di **clock**], il quale attiva e disattiva la memorizzazione dei dati:

- LE = attivo \rightarrow **TRASPARENZA** $[Q(t) = D(t)]$;
- LE = disattivo \rightarrow **MEMORIA** $[Q(t) = D(t-1)]$.

Qui l'uscita commuta sempre dopo la variazione dell'ingresso:

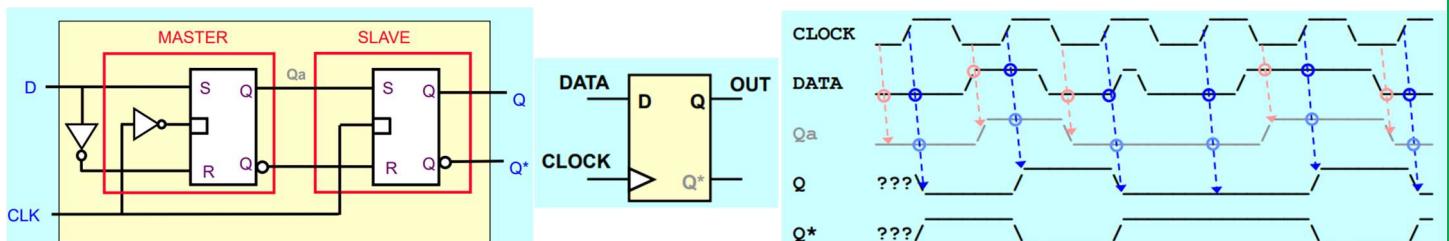


Esiste una versione del Latch D con 2 segnali asincroni chiamati **PRESET** e **CLEAR** (che **agiscono in maniera indipendente dal LE**, sono collegati direttamente all'alimentazione).



Poi ci sono i **FF Master-Slave** (o **D Master-Slave**) formati da una cascata di FF Latch (uscite del FF precedente = ingressi del FF successivo) con **clock [CLK]** in opposizione di fase (per evitare che il master [il 1° FF] e lo slave [il 2° FF] non siano mai nello stesso stato) ed un unico ingresso $D = S = R^*$. Se:

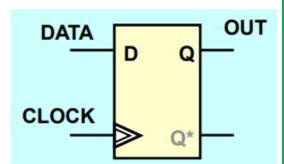
- **CLK = 0** \rightarrow abilita il 1° latch, blocca il 2°; master = trasparente; slave = memoria
- **CLK = 1** \rightarrow blocca il 1° latch, abilita il 2°; master = memoria; slave = trasparente



Infatti se ho una transizione del CLK basso-alto ($0 \rightarrow 1$), blocco il 1° latch (ovvero il master, in quanto prende il controllo del dato e lo immagazzina), mentre il 2° FF (slave) è trasparente (per questo detto FF "edge triggered").

⚠ È la porta sequenziale più usata!

Esiste anche un 2° tipo di D Master-Slave, differenziato da quello appena descritto per la condizione di memoria: si chiama **D dual edge DDR**, dove la memorizzazione avviene al doppio della frequenza in quanto non memorizza solo a valori di clock alti ($0 \rightarrow 1$), ma ad ogni fronte di clock (ad ogni transizione sia $0 \rightarrow 1$ sia $1 \rightarrow 0$).

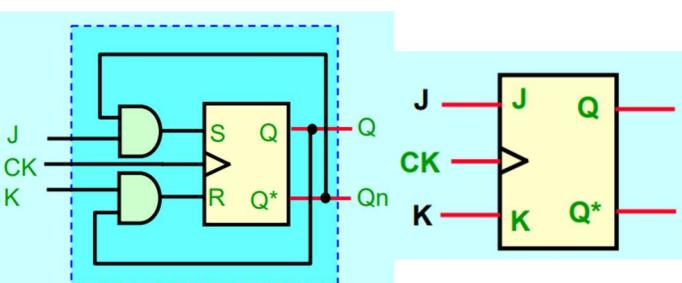


⚠ Quindi riassumendo:

- **Latch** \rightarrow trasparenza $LE = 1$; memoria $LE = 0$; memorizza D se $LE 1 \rightarrow 0$;
- **FF D (master-slave)** \rightarrow uscita commuta solo sulle transizioni del clock; memorizza D se $CLK 0 \rightarrow 1$.

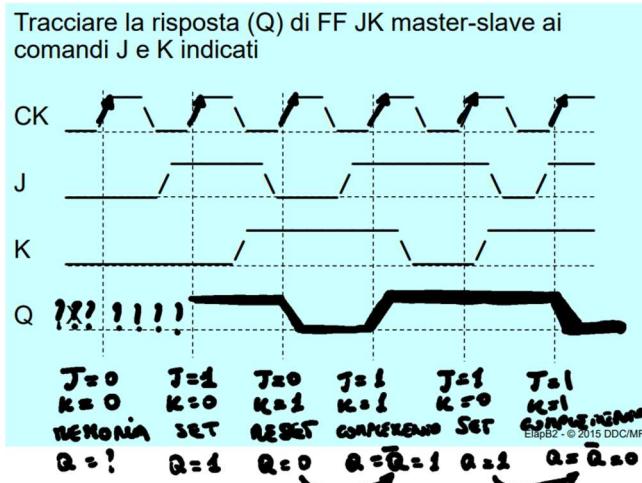
Altro tipo è il **FF JK**, ovvero un **FF SR master-slave con reazione incrociata** (le uscite sono le stesse del FF SR **eccetto la condizione proibita**, che diventa il “complemento dell’uscita”, cioè con **J=K=1** l’uscita commuta ad ogni colpo di clock); gli ingressi sono J e K, mentre con CK indichiamo il clock.

J	K	S	R	Q	Q^*
0	0	0	0	Q_{-1}	Q_{-1}^*
0	1	0	Q_{-1}	0	1
1	0	Q_{-1}^*	0	1	0
1	1	Q_{-1}	Q_{-1}	Q_{-1}^*	Q_{-1}



J	K	Q	memoria
0	0	Q_{n-1}	reset
0	1	0	set
1	0	1	
1	1	Q_{n-1}	complemento

Un **ESEMPIO** di esercizio legato a questi FF:



Parliamo ora della **TEMPORIZZAZIONE** dei FF. Le porte logiche introducono dei ritardi: l’uscita infatti commuta sempre dopo il comando all’ingresso con un ritardo t_p (tempo di propagazione). Oltre a ciò vanno rispettati dei parametri affinchè si passi dall’ingresso all’uscita senza errori; questi parametri sono:

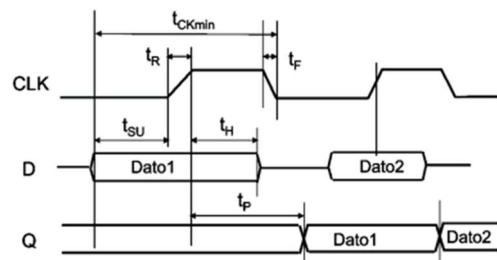
- t_p = tempo di propagazione (da ingresso a uscita);
- t_{SU} = tempo di setup (tempo prima della salita del clock in cui il dato deve rimanere stabile);
- t_H = tempo di hold (tempo dopo il fronte di salita del clock in cui il dato deve rimanere stabile).

A cui si aggiungono anche i t_R (tempo di salita/rise del clock) e t_F (tempo di discesa/fall del clock). Si definisce “resolving time” (t_{res}) il ritardo con cui l’uscita si porta in uno stato corretto (prevedibile in modo statistico e non deterministico).

I comandi non devono essere troppo brevi tali che non si verifica nessuna commutazione (quindi situazione transitoria o di “**metastabilità**”); affinchè il FF commuti in modo stabile, la **variazione di stato deve propagarsi lungo tutto l’anello** (per esempio nei FF SR e nei latch, i comandi S e R devono avere durata $> 2t_p$; oltre a ciò, nei master-slave di tipo D ci vuole anche una durata minima del clock).

Quindi il clock deve restare per un tempo minimo negli stati H e L secondo la formula (**periodo minimo di clock** e quindi **massima frequenza operativa**):

$$T_{CKmin} = t_{SU} + t_R + t_H + t_F \rightarrow f_{CKmin} = \frac{1}{T_{CKmin}}$$



3) CIRCUITI SEQUENZIALI

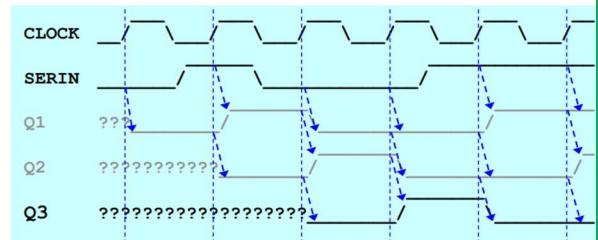
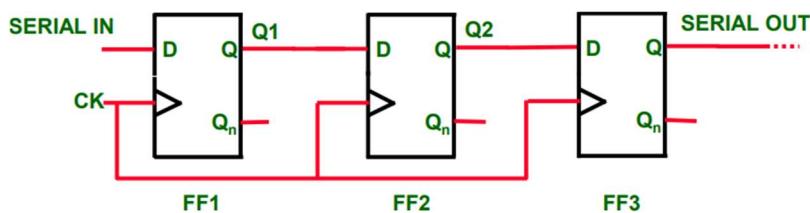
I segnali numerici (digitali) possono essere trattati in forma **SERIALE** (bit successivi su 1 bus; 1 bit per ogni ciclo di clock; meno velocità, meno consumi e meno costi [più usata su lunghe distanze]) o **PARALLELA** (bit contemporanei su diversi bus; N bit contemporaneamente per ogni ciclo di clock; più velocità, più consumi e più costi). Oggi si usa più la comunicazione seriale ad alta velocità.

Come abbiamo già detto, ci sono **4 tipi di segnali di clock** (quando non diciamo nulla, il default è il CK):

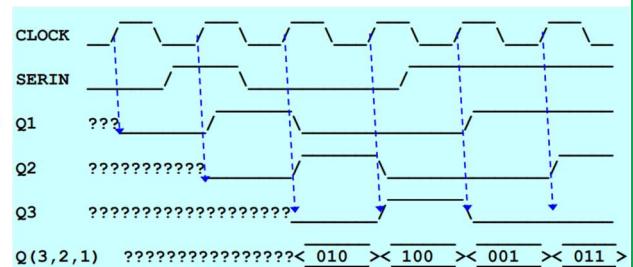
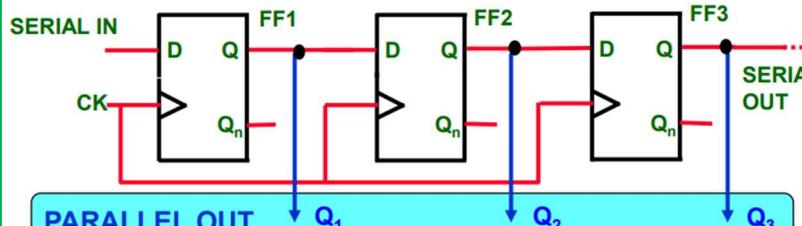
- **Latch** = sensibile al livello:
 - o Positivo = LE
 - o Negativo = \overline{LE}
- **Edge-triggered (clock)** = sensibile al fronte:
 - o Positivo = CK
 - o Negativo = \overline{CK}

Ricordato questo, parliamo di **REGISTRO**, ovvero un insieme di FF con comandi comuni. Possono avere ingresso e uscita parallela o seriale (PIPO, PISO, SIPO o SISO [S = seriale; P = parallelo; I = input; O = output]). Anche i registri sono classificati in base ai 4 segnali di clock sopra citati (Latch e Edge-triggered), ovvero il segnale di comando può essere un **CK** (o \overline{CK}) oppure un **EN** (o \overline{EN} , enable).

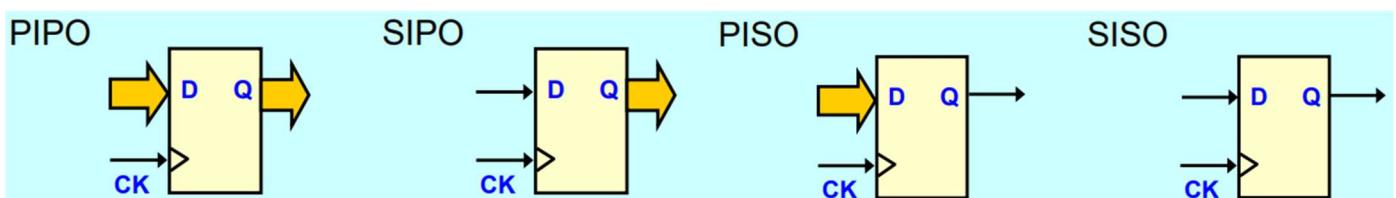
Un tipo è lo **SHIFT-REGISTER** (convertitori seriale-parallelo), ovvero un insieme di **FF-D** in cascata con un **clock comune**, soggetti ad un meccanismo a scorrimento (**shift**) dove il **dato viene scalato nella catena** (si vede questo comportamento di scalamento dal grafico di timing); sotto un **SR SISO** ↓:



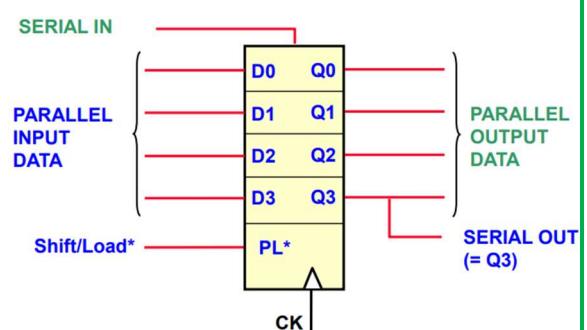
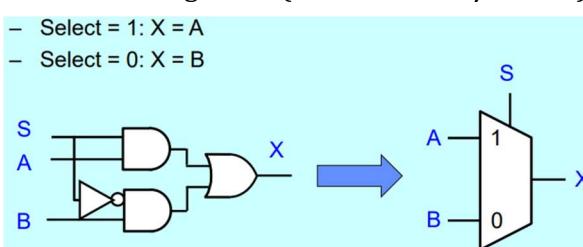
Analogamente qui sotto un **SR SIPO** con il suo grafico di timing:



Analogamente abbiamo anche i **SR PISO** e **PIPO**; quindi riassumendo:

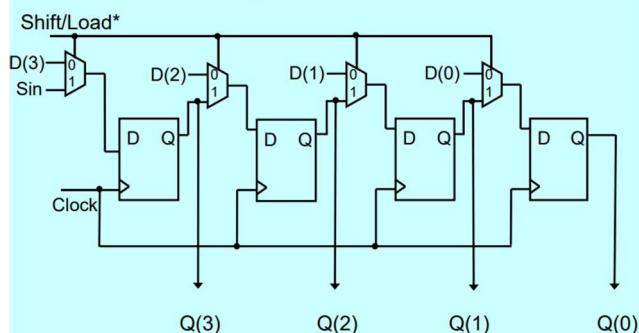


Uno **SHIFT-REGISTER COMPLETO** ha ingressi e uscite sia seriali sia paralleli (ibrido tra tutti). Ogni FF ha in ingresso un **MUX** (multiplexer) che seleziona 1 fra 2 dati in ingresso (parallelo o seriale) mediante un segnale **S** (select o **SHIFT/LOAD***):



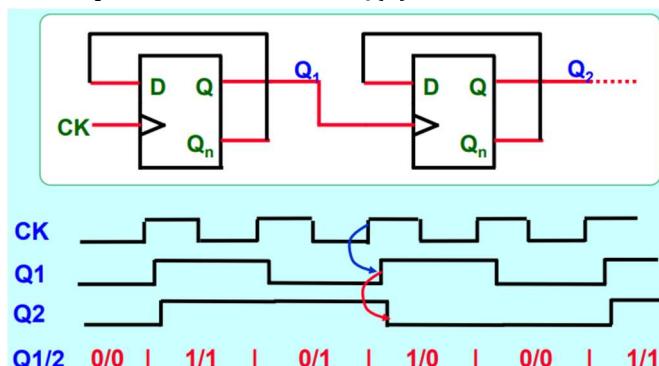
Nella pratica, con un ESEMPIO con 4 ingressi D(0)-D(1)-D(2)D(3), dove il comando SHIFT/LOAD* carica gli ingressi in parallelo se basso, mentre esegue lo scorrimento se alto.

- Shift/Load*=0 carica ingresso parallelo D(3:0)
 - Shift/Load*=1 esegue lo scorrimento



Ora vediamo il **CONTATORE** (circuito logico che genera sulle uscite una sequenza di conteggio binario, incrementata ad ogni ciclo di Clock) e il **DIVISORE** (contatore di cui prendo solo 1 uscita $Q(n)$, dove il modulo M del divisore è il rapporto $f_{clock}/f_{o(n)}$).

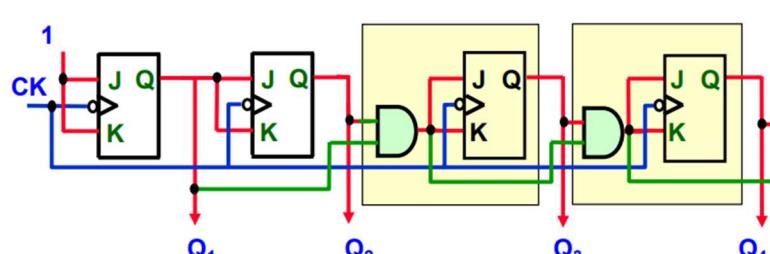
Un **DIVISORE modulo 2/4** a riporto è mostrato a lato(dove ogni stadio divide la frequenza di $CLK \% 2$). Quindi il campo di applicazione di un divisore è la **DIVISIONE IN FREQUENZA**, ovvero, in base a quanti FF ha il contatore, avverrà una divisione in frequenza di ordine 2^{nFF} (es. se ho **clock con $f = 50MHz$** in contatore con 1 FF [cioè ad 1 bit], l'uscita avrà $f = 25MHz$).



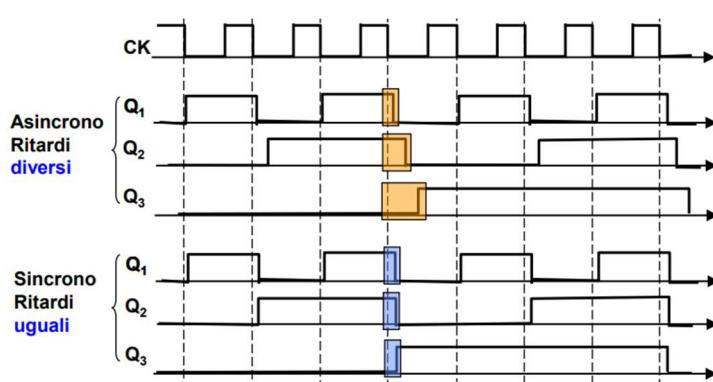
C'è il **CONTATORE ASINCRONO**, dove il clock è collegato solo al 1° FF della cascata. I successivi clock sono collegati a catena (“**ripple counter**”), generando un ritardo di commutazione comulativo (se t_p = ritardo di propagazione del singolo FF, allora ritardo totale sarà Nt_p). Le uscite commutano con ritardi differenti.

⚠️ Un **CONTATORE JK-FF** (ovvero realizzato con FF JK) è un contatore asincrono, in quanto il FF cambia stato (commuta)ognivolta che $J \equiv K \equiv 1$.

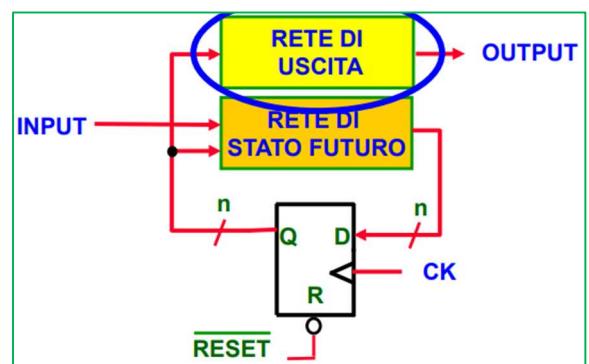
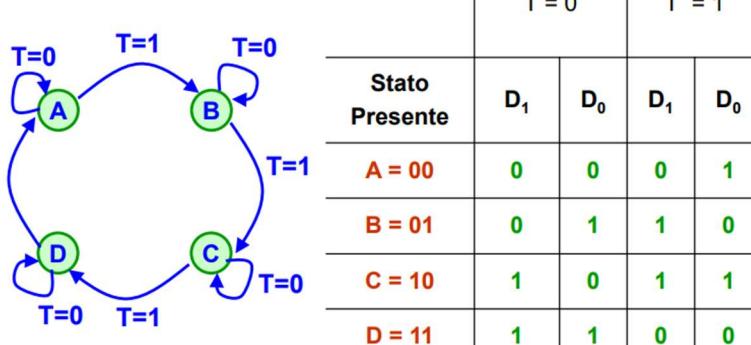
C'è poi il **CONTATORE SINCRONO**, dove tutti i FF ricevono lo stesso clock (tutti ricevono il clock direttamente) e quindi le uscite commutano tutte con lo stesso ritardo (che non viene accumulato da FF a FF).



Quindi visivamente:



I contatori visti finora sono delle **FSM** (Finite State Machine), dove ogni FF (elemento di memoria) ha una condizione (bit = 0 o bit = 1) che identifica lo **STATO** del sistema; i passaggi tra stati sono rappresentati da archi e ad ogni stato corrisponde una **combinazione delle uscite** (mutuamente esclusive). Questa rappresentazione è il **DIAGRAMMA DEGLI STATI** (dove lo stato presente è quello corrente, lo stato futuro è quello a cui si passerà):



A destra la rappresentazione a blocchi di una FSM. Se l'uscita è indipendente dagli ingressi correnti (quindi dipende solo dalle uscite precedenti [gli stati]), ho una FSM di **Moore**; se invece dipende sia dagli stati sia dagli ingressi correnti, ho una FSM di **Mealy**.

Parlando delle **TEMPORIZZAZIONI CON LC** (**LOGICA COMBINATORIA**) abbiamo già detto che la frequenza massima di funzionamento è legata al soddisfacimento del tempo di setup dei FF t_{SU} (oltre che legata ai t_H , t_R e t_F), ma la frequenza massima è anche influenzata dai **ritardi legati alla logica combinatoria** dei circuiti. Introduciamo quindi un nuovo ritardo da garantire:

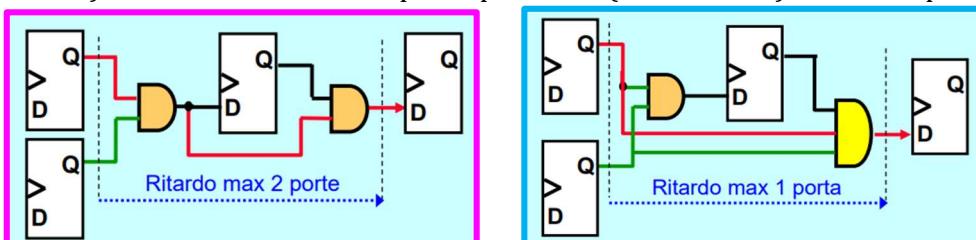
- $t_{CK-Q} \rightarrow$ ritardo transizione “clock-uscita” = dopo il fronte di clock, l'uscita si stabilizza dopo t_{CK-} ;
- $t_{LC,min}$ e $t_{LC,max}$ → minimo e massimo ritardo di propagazione della LC.

Se uniamo quindi le temporizzazioni viste prima a queste, avremo come **CONDIZIONI** da rispettare nel calcolo del periodo minimo di clock (e quindi reciprocamente massima frequenza di clock):

$$\begin{cases} T_{CK} > t_{CK-Q} + t_{LC,max} + t_{SU} \\ t_{CK-Q} + t_{LC,min} > t_H \end{cases}$$

Ovvero la **frequenza massima** (cioè il periodo minimo) di clock è legata a t_{CK-Q} , $t_{LC,max}$ e t_{SU} (in queste disequazioni andrebbe aggiunto nella 1^a e sottratto nella 2^a il tempo di skew, ovvero la differenza di quando arriva il clock ad un FF rispetto ad un altro FF [perché non viviamo nell'idealità]).

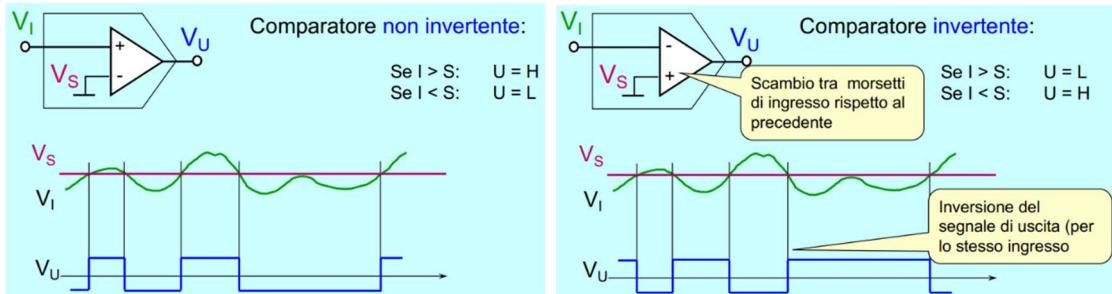
Le strutture a catene di porte (“**ripple**”), come il contatore asincrono, seppur siano più facili da implementare, portano ad un accumulo di ritardo. Una soluzione per **ridurre i ritardi** è usare al posto delle “**ripple**”, strutture dirette (“**look-ahead**”), ovvero 1 solo livello di porte, più veloci (meno ritardi), ma anche più complesse.



⚠ Da notare che nel “look-ahead” è max 1 ritardo perché il segnale passa massimo per 1 porta (o la AND sopra oppure parte dal FF e finisce nell'AND sotto)!

4) COMPARATORI e OSCILLATORI

Il **COMPARATORE** è un circuito usato per tradurre un ingresso analogico in un'uscita digitale (binaria, L/H \rightarrow 0/1). Se l'ingresso è maggiore di una soglia, in uscita ho un stato H (non-invertente; se invertente, l'opposto).



Dato che però il segnale è soggetto a **RUMORE**, si potrebbero verificare commutazioni spurie (non volute); per evitare attraversamenti di soglia non voluti, si usa il **COMPARATORE CON ISTERESI** (2 soglie, **isteresi** = differenza delle 2 soglie), ovvero, a differenza del COMPARATORE SEMPLICE (che è ad anello aperto), l'isteresi è dovuta da retroazione positiva (a differenza dell'amplificatore operazionale che ha retroazione negativa).

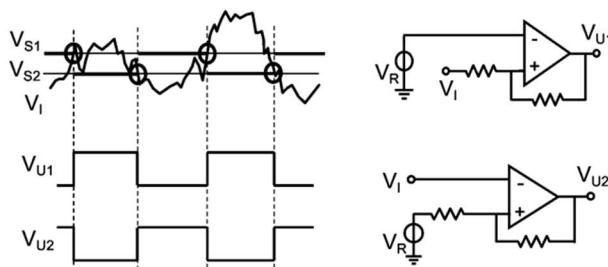


Figure 26: Comparatore con isteresi *invertente* e *non invertente*

Il **comparatore con isteresi** è anche detto "**TRIGGER DI SCHMITT**" e può usato come **generatore di onda quadra** se messo in collegamento con una rete RC (il condensatore si carica e scarica tra le soglie S1 e S2 del comparatore).

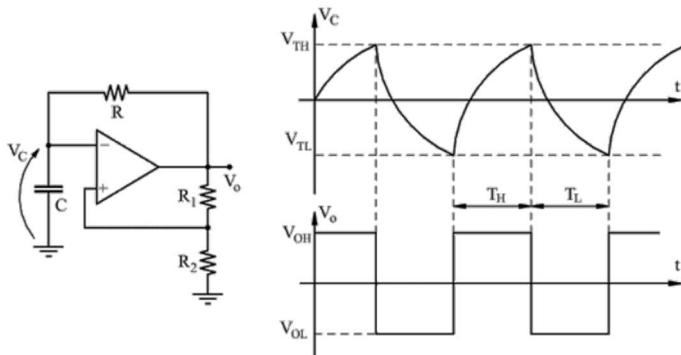
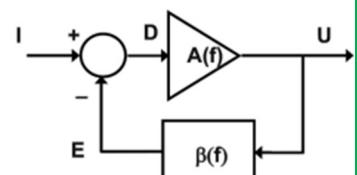


Figure 27: Generatore di onda quadra con trigger di Schmitt

L'OSCILLATORE è un circuito che genera onde di varia frequenza, ampiezza e forma **senza un segnale in ingresso**.

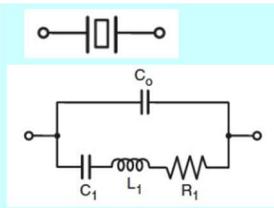
L'OSCILLATORE AD ANELLO è formato da un **numero dispari di inverter** ed è **senza uno stato stabile**; il suo **periodo** non è preciso in quanto **dipende dai ritardi di propagazione degli inverter** $T = N(t_{pLH} + t_{pHL})$. Perciò si preferiscono altre soluzioni con frequenza più stabile.

Nei **SISTEMI CON REAZIONE**, l'uscita U può oscillare anche senza alcun ingresso specifico se $A(f)\beta(f) = -1$ (**Criterio di Barkhausen**); a prescindere dall'ingresso, le oscillazioni avvengono proprio per $A(f)\beta(f) = -1$. La struttura è qui a lato (**OSCILLATORE CON RETROAZIONE**). Solitamente **$A(f)$ è costante** (perché è un amplificatore), mentre il **guadagno della rete di retroazione β** dipende dalla frequenza, implementata con reti RC, circuito risonante LC oppure con il **QUARZO** (cristallo piezoelettrico).

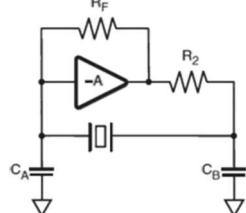


Il **QUARZO** è importante perché [simbolo circuitale a lato, con conseguente modello elettrico che lo rappresenta]:

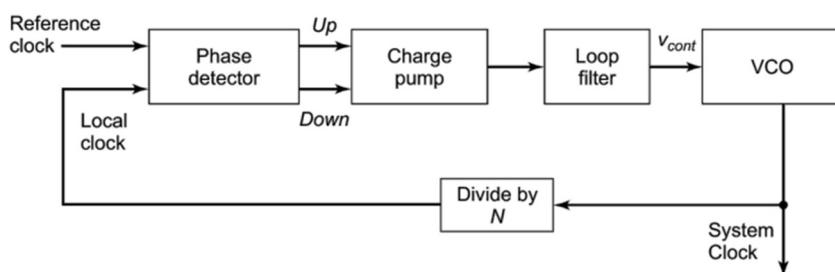
- **Si deforma in un campo elettrico;**
- **Genera tensione se sottoposto a sollecitazioni meccaniche;**
- **È un elemento risonante con basse perdite e alta precisione.**



Un esempio di utilizzo del quarzo è l'**OSCILLATORE DI PIERCE**, dove l'amplificatore può essere sostituito da una porta NOT, la resistenza in parallelo R_F forza l'inverter a lavorare intorno a $V_{in} = V_{out}$ e si ha una transcaratteristica con alto guadagno.

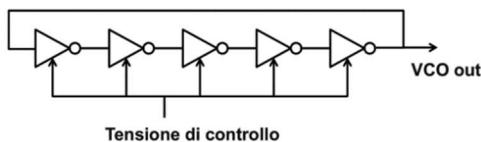
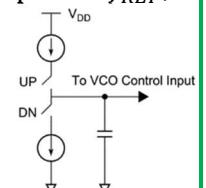


La massima frequenza di un oscillatore al quarzo è $f = 200\text{MHz}$; perciò, per frequenze nell'ordine dei GHz usate processori attuali, serve il **PLL** (**Phase Locked Loop**), ovvero un sistema di controllo automatico che genera un segnale periodico che ha fase in relazione fissa con il segnale in ingresso di riferimento. Il suo utilizzo primario è **sintetizzare una frequenza**: si fa agganciando un **VCO** (voltage controlled oscillator), caratterizzato da un'alta frequenza ma una bassa precisione, ad un oscillatore al quarzo, che ha bassa frequenza e alta precisione; in questo modo si sintetizza un **oscillatore ad ALTA FREQUENZA e PRECISIONE**.



Il PLL sopra mostrato è **composto da**:

- **PHASE DETECTOR** → rilevatore di fase. Identifica quando l'oscillatore è troppo lento o veloce rispetto a f_{REF} ;
- **CHARGE PUMP** → generatore di tensione controllato da 2 interruttori, azionati se l'oscillatore è troppo lento (Up) o troppo veloce (Down) [immagine a lato];
- **VCO** (voltage controller oscillator) → oscillatore controllato in tensione realizzato con un oscillatore ad anello che può raggiungere alte frequenze (GHz), stabilizzato come citato sopra dall'anello di reazione che lo aggancia all'oscillatore al quarzo di riferimento. I singoli stadi hanno un ritardo dipendente dalla tensione di controllo.

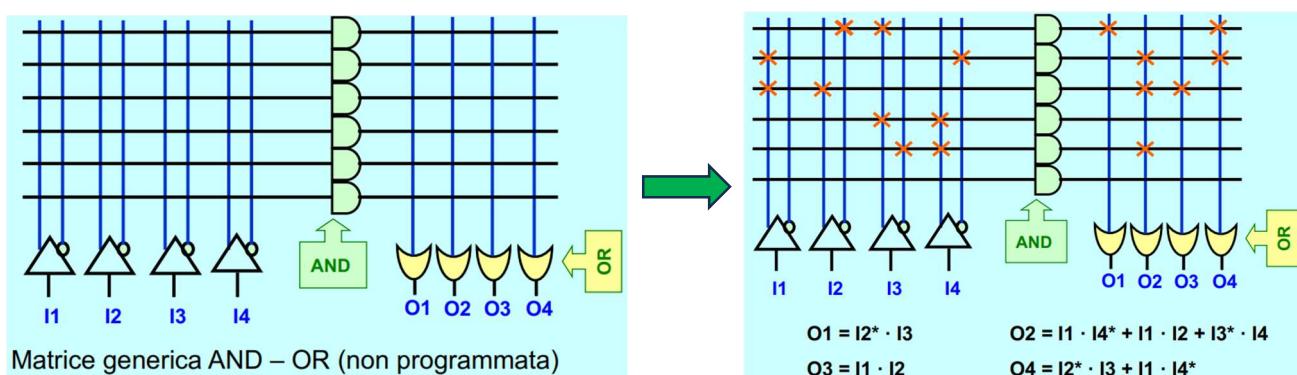


5) LOGICHE PROGRAMMABILI

SOLO DA LEGGERE [Ricordiamo la **LEGGE DI MOORE** (legge economica), che afferma che il n° di transistor su un microprocessore raddoppierà circa ogni 2 anni. Definiamo NRE = costo di progetto e di allestimento della produzione (indipendente dal numero di oggetti prodotti), e CU = costo unitario di produzione. Il tempo di progetto ritarda l'ingresso sul mercato con un effetto quadratico sui ricavi. **CIRCUITI FULL CUSTOM [ASIC]** = sono necessari tutti i passi di progetto (fino a livello transistor) e fabbricazione, partendo dal silicio non lavorato (più ottimizzazione di progetto, ma altissimi costi); **CIRCUITI SEMICUSTOM** = sono necessari alcuni passi di progetto, ma tutti i passi di programmazione (volumi elevati, alte prestazioni).]

Definiamo **DISPOSITIVO LOGICO PROGRAMMABILE** (**PLD, FPGA, PSOC**) un circuito integrato programmabile che alla fabbricazione non ha una funzione logica definita, ma va programmato dall'utente prima dell'uso (maggiore flessibilità, minori prestazioni, maggiore costo [basso costo di progetto NRE, alto costo unitario]). Vediamo:

- **PLA (Programmable Logic Array)** → dato che **ogni funzione logica si può esprimere come somma logica**, nasce il PLA, che prevede una **matrice di porte AND programmabili collegata con una serie di porte OR programmabili** (sotto un esempio di logica programmabile):



- **FPGA (Field Programmable Gate Array)** → dispositivo HW elettronico composto da un **circuito integrato con logica programmabile e modificabile** (usando linguaggi di descrizione HW). Usato al posto di una CPU in un sistema, questo richiede maggiori costi e tempi di programmazione, ma meno costi di progetto e maggiore flessibilità alle modifiche [sotto a dx la struttura di una FPGA, dove 4-LUT è una look-up table a 4 ingressi].

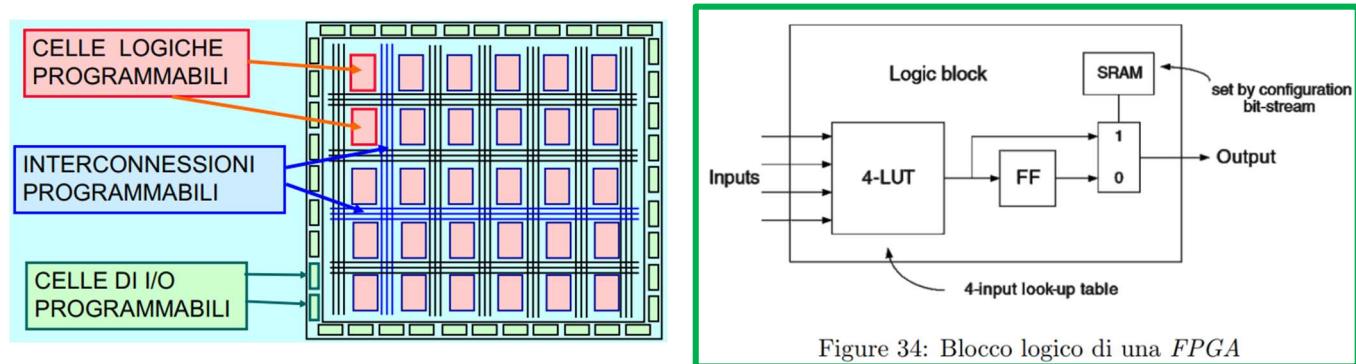
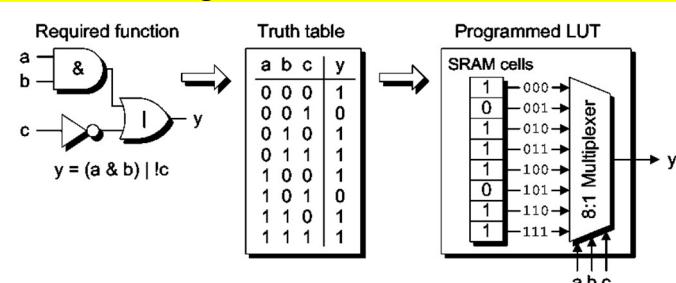


Figure 34: Blocco logico di una *FPGA*

Legato agli FPGA, c'è la **LOOK-UP TABLE (LUT)**, ovvero una struttura dati (composta da celle **SRAM** e da **MUX**) dove i risultati della tavola di verità (ovvero i risultati ottenuti dalla combinazione degli ingressi) vengono salvati nella SRAM (e quindi posso avere una tavola di verità "custom" fatta da me per svolgere la funzione logica che voglio). Quindi **posso accedere alle soluzioni dei circuiti logici mediante la tavola di verità, senza dover fare effettivamente il calcolo runtime** (quindi prendo la soluzione della mia funzione logica cercata, senza dover effettivamente implementarla e senza farle fare i calcoli sul momento).



6) LINGUAGGIO VERILOG

Verilog è un **linguaggio parallelo** utile per modellare il parallelismo dell'HW; ci sono 2 tipi di modellazione: a livello di porte logiche (strutturale, tutte le istruzioni sono simultanee [ordine delle istruzioni non importante]) e comportamentale (istruzioni eseguire in sequenza [ordine importante]). In Verilog troviamo i dati di tipo:

- **net** = connettono le parti del progetto. Il tipo **wire** è un tipo di dati net: **non memorizza** un valore/stato (deve essere continuamente pilotato altrimenti perde il valore) **ma trasferisce** valore;
- **variabile** = **memorizzano** dati. Il tipo **reg** è un tipo di dati variabile che memorizza l'ultimo valore assegnatogli (non corrisponde necessariamente ad un registro).

Si possono usare **vettori di bit** (detti **bus**) che si scrivono con **tipo [MSB:LSB] nome_vettore**. Se voglio prendere un singolo elemento del vettore faccio **nome_vettore[indice]** (oppure posso prendere range di valori con **nome_vettore[inizio:fine]**).

Si possono anche usare le **matrici di bit**: se definisco per esempio **reg mat[15:0]** sto definendo una matrice di 16 elementi da 1 bit ciascuno (scalare), ma se definisco **wire [7:0] mat[1023:0]** sto definendo una matrice di 1024 elementi da 8 bit ciascuno (vettore). Con **mat[i]** prendo un byte specifico della matrice, mentre con **mat[i][j]** prendo un bit di un byte della matrice (j indica la posizione del bit nel byte).

Verilog usa **4 valori logici di base**:

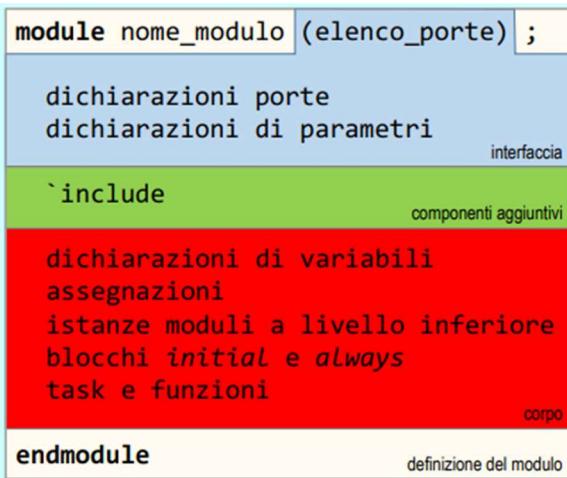
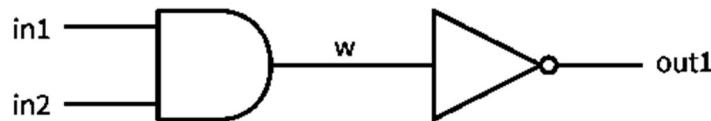
- **0** = zero logico (false);
- **1** = uno logico (true);
- **Z (o z)** = alta impedenza (uscita disabilitata per esempio);
- **X (o x)** = valore logico sconosciuto.

In Verilog si definisce una **costante** con **[dimensione][‘base]valore** con dimensione = numero di bit (default = 32 bit), e base = {‘b = binario, ‘o = ottale, ‘d = decimale (default), ‘h = esadecimale} (es. 16’o56377 → 56377 scritto in base ottale su 16 bit). Legate alle costanti, abbiamo le **macro** definite con **`define nome_macro valore**, e richiamate nel codice con **`nome_macro**.

Un **MODULO** è un componente riutilizzabile che ha struttura riportata a destra (elenco_porte indica i pin [porte in e out], mentre gli include vanno fatti come in C ma con ` davanti). I **commenti** si fanno come in C con **//**. Le **porte** possono essere dichiarate con tipo **input**, **output** o **inout** (input + output, che dovrebbero essere dunque pilotate da **porte tri-state**).

Nella modellazione a livello di porte logiche, le **porte logiche** sono definite con “tipo logico” **and**, **or**, **not**, **buf** (mantiene il valore - buffer), **nand**, **nor**, **xor**, **xnor**, **bufif1**, **bufif0**, **notif1**, **notif0** mettendo come **primo parametro** l’uscita e **come seguenti parametri gli ingressi**. Es (codice fa riferimento a circuito a dx):

```
module my_gate (out1, in1, in2);
    output out1;
    input in1, in2;
    wire w;
    and (w, in1, in2);
    not (out1, w);
endmodule
```



I **RITARDI** delle porte logiche sono espressi come **#(tp)** o **#(tPLH, tPHL)** e sono scritti tra il tipo di funzione logica e i parametri di uscita/ingresso; si scrivono con 1 solo valore (per indicare il ritardo massimo) o con 3 valori (min:medio:max) [es. **xor #(2:3:4,5) (o,i1,i2)**]. L’unità di misura dei ritardi è definita con **`timescale unità_tempo/precisione**.

Se ho circuiti che implementano pezzi di circuiti già definiti in altri moduli, **posso richiamare il modulo e riusarlo**: se per esempio ho definito un “half-adder” nel modulo **hadd** e sto ora definendo un “full-adder” che sono più half-adder collegati tra loro, posso richiamare il modulo dell’half-adder nel modulo del full-adder con la dichiarazione

hadd M1 (w1, w2, a, b), dove **hadd** = nome_modulo, **M1** = nome_istanza (ogni richiamo ad un altro modulo ha un suo identificativo che sarebbe l'**istanza**).

Nella **modellazione a livello di registri di trasferimento (RTL, register transfer level)** [usata per descrivere la logica combinatoria], avremo **assign [#ritardo] nome_rete = espressione** (es. **assign out = i1 & i2 | i3;** oppure anche es. **assign eq = (a+b == c)** [sommatore e comparatore]). Gli operatori supportati in Verilog sono:

Operatori aritmetici		Operatori bit a bit		Operatori di spostamento	
a + b	Somma	~a	NOT bit a bit	a << n	Shift logico a sinistra
a - b	Differenza	a & b	AND bit a bit	a >> n	Shift logico a destra
-a	Cambio segno	a b	OR bit a bit	a <<< n	Shift aritmetico a sinistra
a * b	Moltiplicazione	a ^ b	XOR bit a bit	a >>> n	Shift aritmetico a destra
a / b	Divisione	a ~^ b	XNOR bit a bit	{a, b}	Concatenare
a % b	Resto	a ^~ b	XNOR bit a bit	~^a	XNOR tutti i bit
Operatori relazionali		Operatori di riduzione		Operatori logici	
a == b	Uguale	&a	AND tutti i bit	!a	Negazione logica
a != b	Diverso	a	OR tutti i bit	a && b	AND logico
a < b	Minore	^a	XOR tutti i bit	a b	OR logico
a > b	Maggiore	~&a	NAND tutti i bit	sel?a:b	Condizionale
a <= b	Minore o uguale	~ a	NOR tutti i bit		
a >= b	Maggiore o uguale	~^a	XNOR tutti i bit		

⚠ Ricorda che le assegnazioni di una variabile ad una **somma**, se ho **c = a + b** (con a = N bit, b = N bit), allora c = N+1 bit. Se invece ho **c = espressione logica** (es. **c = a > b** oppure **c = a | | b**), allora c = 1 bit (risultato di un'operazione logica).

⚠ Se devo usare dei valori con segno, devo usare **signed** prima del valore (es. **output signed [3:0] s;**).

Se voglio usare dei **parametri al posto di valori (più flessibilità)**, posso usare **parameter nome_parametro = valore_parametro** prima del codice e posso riusarlo chiamando il suo nome (es. **parameter width = 2;** → nel codice potrò scrivere **input [width-1:0] a;**) [usati come costanti]. Quando chiamo un modulo contenente dei **parametri**, posso **sostituire il valore che voglio nella mia chiamata** usando **nome_modulo #(valore_parametro) nome_istanza (...argomenti).**

C'è poi il blocco **generate**, che posso inserire dentro un modulo per fargli **eseguire del codice condizionale con for e if-else**; ha struttura:

```
generate for (i = 0; i < 4; i = i + 1) begin
    if (i==0)      fadd (co[0], s[0], a[0], b[0], ci);
    else if (i==3) fadd (cout, s[3], a[3], b[3], co[2]);
    else          fadd (co[i], s[i], a[i], b[i], co[i-1]);
end endgenerate
```

Se voglio invece attuare una modellazione comportamentale, devo usare i blocchi **ALWAYS** (descrive un processo che viene eseguito ripetutamente per sempre), con cui posso modellare sia circuiti combinatori sia sequenziali. Ha sintassi come riportata a dx; la **sensitivity_list** definisce quando un **always @ (sensitivity_list) begin** blocco always viene eseguito 1 volta da cima a fondo, ovvero può essere:

- ATTIVO SUL **LIVELLO (LATCH)** → **always @(espressione logica);**
- ATTIVO SUL **FRONTE (FF con reset)** → **always @(posedge clock)** oppure **always @(negedge clock or reset)**.

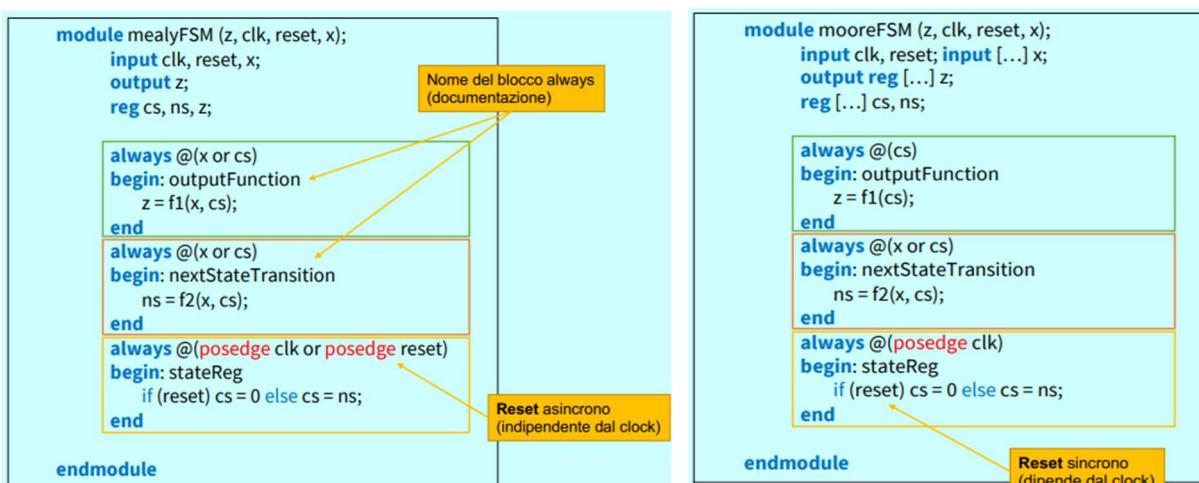
⚠ Tutte le uscite di un blocco always devono essere di tipo **reg** (wire sono le uscite delle istruzioni assign).

Nei blocchi always ci sono le **ASSEGNAZIONI BLOCCANTI** (le variabili vengono assegnate **subito**, ovvero hanno un effetto immediato) [con sintassi normale **a = b**] e le **ASSEGNAZIONI NON BLOCCANTI** (le variabili vengono assegnate **dopo un po' di tempo** e quindi l'assegnazione assegna un valore futuro) [con sintassi **a <= b**]. **Noi in questo corso useremo solo le BLOCCANTI.**

Altra istruzione utile è la **if-[else if]-else-[begin-end]** (analogia al C, ma il begin-end opzionale può racchiudere delle istruzioni da eseguire in una delle condizioni). Poi c'è il costrutto **case** che sarebbe l'equivalente dello switch del C (**case(i1)i1: ...; i2: ...; default: ...; endcase**). Esempio (ALU):

```
module ALU (y, a, b, sel);
    parameter width = 3;
    input [width-1:0] a, b;
    input [1:0] sel;
    output [width-1:0] y;
    reg [width-1:0] y;
    always @(a or b or sel) begin
        case (sel)
            2'b00: y = a + b;
            2'b01: y = a - b;
            2'b10: y = a + 1;
            2'b11: y = a - 1;
        endcase
    end
endmodule
```

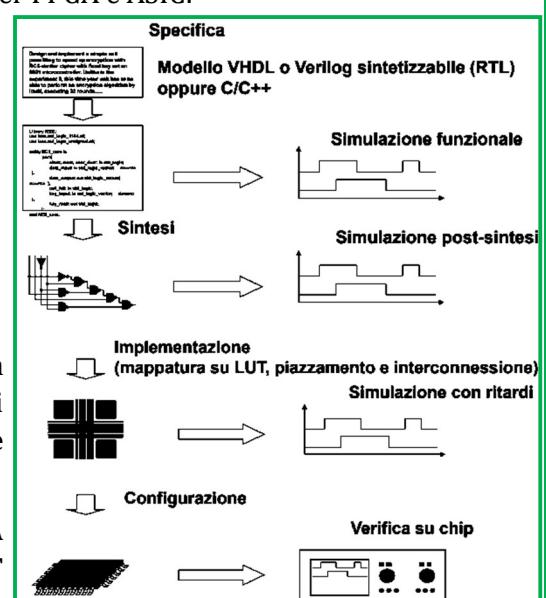
Possiamo usare Verilog anche per simulare una FSM di Mealy (sx) e FSM di Moore (dx):



Per quanto riguarda il **FLUSSO DI PROGETTAZIONE** vediamo quello per FPGA e ASIC:

- **FPGA:**

- **SINTESI** = compila codice RTL (es. Verilog o VHDL) in registri e logica combinatoria, ottimizza la logica combinatoria (con l'algebra booleana), genera un'interconnessione di porte (netlist), fornisce stime iniziali dei tempi/ritardi (ignorando la mappatura LUT, piazzamento e interconnessione);
- **REALIZZAZIONE (IMPLEMENTAZIONE)** = PLACING (sceglie la posizione sul chip per LUT e FF, cercando di minimizzare la distanza per minimizzare i ritardi) + ROUTING (sceglie la sequenza di segmenti di interconnessione tra uscite e ingressi di LUT e FF, cercando di minimizzare la lunghezza delle interconnessioni) [nei FPGA interconnessioni programmabili];
- **CONFIGURAZIONE** = viene creato un file che configura la FPGA con i valori da assegnare ai bit di memoria che configurano LUT e interconnessioni ("bitstream"). Per FPGA basate su:
 - ❖ RAM → caricato in una ROM che FPGA legge (o direttamente sull'FPGA solo per debug);
 - ❖ EEPROM (o fusibili [CPLD]) → caricato direttamente sull'FPGA.
- **VERIFICA E DEBUG SUL CHIP** = la simulazione non è adatta a tutti i tipi di verifica perché lenta rispetto all'HW reale, ma al tempo stesso ci garantisce accesso a tutti i segnali interni. L'applicazione viene eseguita direttamente sull'FPGA (PC comunica con FPGA con collegamento JTAG).



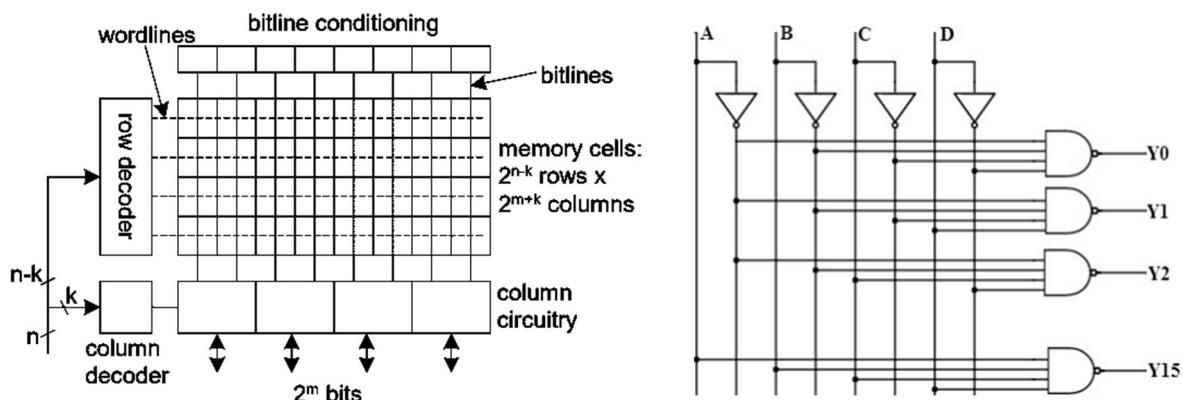
- **ASIC** (rispetto all'FPGA, richiede molte altre verifiche [molto più costoso] e non si possono fare errori):
 - **SINTESI**;
 - **REALIZZAZIONE (IMPLEMENTAZIONE)** = PLACING → porte logiche organizzate in righe (non in una matrice come gli FPGA) cercando di fare un posizionamento adeguato per ridurre la congestione e migliorare i costi, i ritardi e la potenza. ROUTING → interconnessioni locali (sceglie le geometrie della regione locale) + interconnessioni globali (sceglie le regioni locali da usare e lavora sull'intero circuito integrato);
 - **CONFIGURAZIONE**;
 - **VERIFICA E DEBUG SUL CHIP** (molto più dell'FPGA).

7) MEMORIE A SEMICONDUTTORE

Le **MEMORIE A SEMICONDUTTORE** sono le componenti a semiconduttore (transistor, chip, ...) del calcolatore in grado di mantenere per un certo tempo dati/istruzioni. Le memorie possono essere **VOLATILI** (staccata l'alimentazione, non tiene più il dato) o **NON-VOLATILI** (tiene il dato anche in assenza della V_{AL}). Le celle di memoria sono indirizzabili singolarmente o in gruppi di bits/bytes. Le memorie, in base al livello di astrazione, possono avere come unità base:

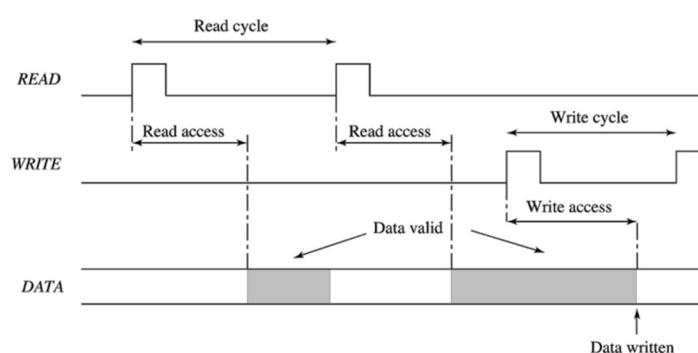
- **bit** → livello di circuito;
- **byte** → livello di chip;
- **word** (pagine) → livello di sistema.

Per indirizzare le singole word, si usa un **decodificatore di indirizzo** (riduce il numero di segnali necessari a $K = \log_2 N$ con N = numero di word da indirizzare [SOTTO A DX un decoder base]). Se ho una matrice di memoria composta da 2^n word (le righe) da 2^m bit ciascuna (le colonne), avrò un'architettura del tipo:



Dove avrò il decoder di indirizzo diviso in **DECODER DI RIGA** ("row decoder") e **DECODER DI COLONNA** ("column decoder"). Avrò 2 fasi:

- **LETTURA**:
 - row decoder attiva 1 **wordline (WL)**;
 - 2^k word selezionate dalla wordline raggiungono il circuito di colonna attraverso le **bitline (BL)**;
 - column decoder seleziona 1 word da leggere delle 2^k selezionate.
- **SCRITTURA**:
 - word da scrivere viene trasferita con le bitline;
 - solo la word selezionata dalla wordline viene scritta.



Partiamo dalle MEMORIE VOLATILI:

- **DRAM** (RAM dinamica, "dynamic RAM") → immagazzina i bit di memoria nel condensatore di "storage". Viene detta dinamica perché il condensatore si scarica e quindi ha bisogno di cicli di **REFRESH** per tenere il dato. Una cella di DRAM è composta da 1 transistor e 1 condensatore: il transistor deve caricare/scaricare il condensatore, il quale memorizza il bit sotto forma di carica. La **BL (bitline)** ha una grande capacità parassita a causa delle moltissime celle connesse ad essa ($C_S \ll C_{BL}$; la variazione di tensione è di $250mV$ sulla BL durante la lettura). Quindi su una DRAM si può fare:

- **SCRITTURA** = la tensione sulla BL è alta (H, 1) o bassa (L, 0). Poi alla riga selezionata dalla WL viene trasferita la tensione sul condensatore della cella (C_S).
- **LETTURA** = ci sono più step:
 - BL viene portata a $\frac{V_{DD}}{2}$;
 - WL viene portata a V_{DD} (per far entrare il transistor in conduzione);
 - i 2 condensatori C_S e C_{BL} sono in parallelo;
 - Si ha una variazione di tensione ai capi di C_{BL} : se C_S sta memorizzando 1, allora la tensione v_{CS} sarà $\approx V_{DD}$ che quindi farà caricare il condensatore C_{BL} aumentando la tensione su di esso (variazione positiva); viceversa se C_S sta memorizzando 0 (variazione negativa di tensione su C_{BL}). In formula:

$$\Delta V_{BL} = \frac{C_S}{C_S + C_{BL}} \left(v_{CS} - \frac{V_{DD}}{2} \right)$$

- Questa variazione di tensione viene rilevata dal "sense amplifier" della colonna (questo amplifica il segnale per farlo tornare al valore iniziale e riportare il condensatore di storage C_S al suo valore iniziale).

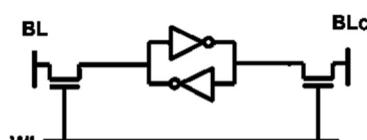
⚠ Per accedere ad una cella di DRAM, accedo con il segnale di riga (RAS, Row Address Strobe) e di colonna (CAS, Column Address Strobe); inoltre convenzionalmente la DRAM è organizzata in **super celle** composte da più bit ciascuna. Se voglio accedere sequenzialmente posso farlo con la "**fast page mode**" (RAS, poi CAS, ma poi solo CAS sequenzialmente per bit/celle vicini e non di nuovo RAS).

- **REFRESH** = a causa delle correnti di perdita bisogna periodicamente rinfrescare il contenuto di una riga con delle finte letture delle righe, tuttavia `e un processo lento.

Distinguiamo poi tra DRAM:

- **ASINCRONA** → regolata da segnali di controlli come RAS (seleziona la riga), CAS (seleziona la colonna), WE (write enable, ovvero setta il pin come ingresso) e OE (output enable, ovvero setta il pin come uscita);
- **SINCRONA (SDRAM)** → lettura e scrittura avvengono ai fronti di salita del clock (se anche sui fronti di discesa si dice DDR [double data rate] [es. RAM DDR5]).

- **SRAM** (RAM statica, "static RAM") → è una memoria volatile che **non ha bisogno di refresh continuo**. Una sua cella è composta da 6 transistor: 2 inverter (ciascuno di 2 transistor, quindi 4) + 2 pass-transistor di selezione (effettuata tramite la WL) che permettono di fare lettura/scrittura (effettuata tramite la BL, 1 normale e 1 complementata BL).



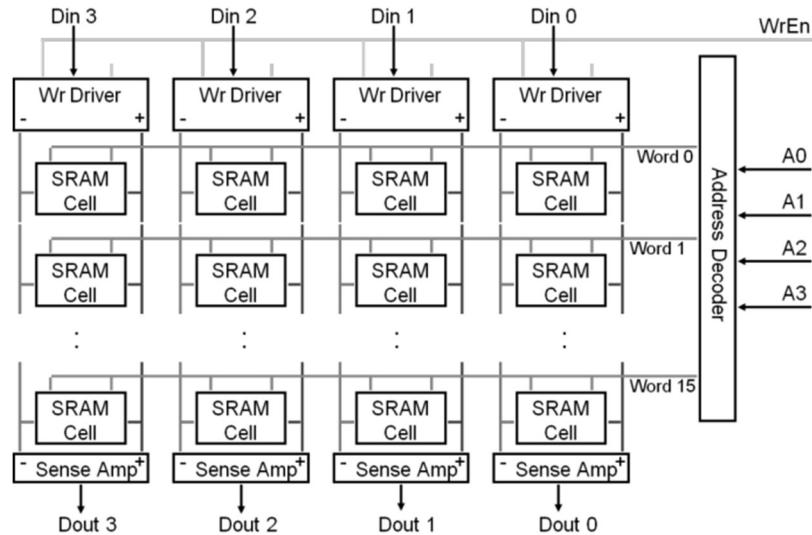
Con una SRAM si può fare:

- **SCRITTURA** → BL (bitline) viene forzata al valore desiderato (e quindi la BL al complementare) e poi viene selezionata la riga tramite WL = 1.

➤ **LETTURA** → 4 step:

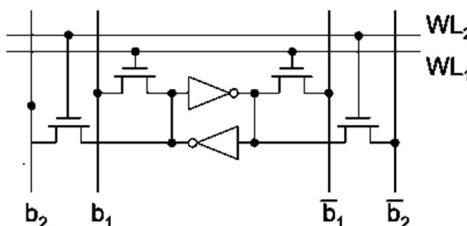
- BL e \overline{BL} caricate a V_{DD} (circuito di pre-carica);
- WL caricata a V_{DD} ;
- BL e \overline{BL} variano in base al valore memorizzato nella cella;
- La variazione viene amplificata dal “sense amplifier” quando la tensione differenziale tra BL e \overline{BL} è sufficientemente grande, e quindi viene trasmessa.

Qui riportata l'architettura di **SRAM da 16 Word x 4 bit**:

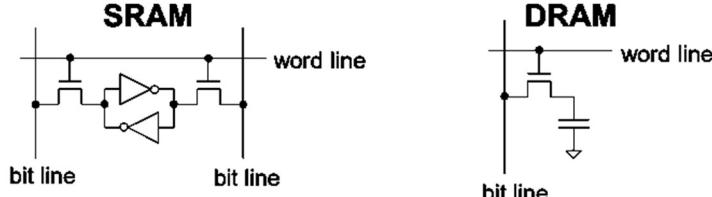


⚠ Nelle **SRAM asincrone** Din e Dout sono combinati insieme per risparmiare pin (OE definisce se in o out) e spesso sono SRAM “standalone”; invece se ho una **SRAM “embedded”** questa spesso è **sincrona**.

⚠ Abbiamo anche le **SRAM Dual-Port**, ovvero con read e write simultanee; questo viene usata per implementare code FIFO (a dx lo schema).



Riassumendo:

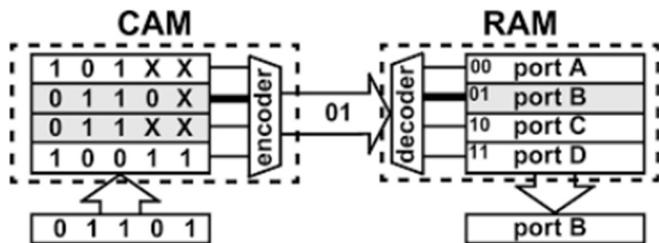


- | | |
|--|---|
| <ul style="list-style-type: none"> • Cella + grande ⇒ densità inferiore, maggiore costo/bit • Consumo statico molto basso • Lettura non-distruttiva • No refresh • Lettura semplice ⇒ accesso + veloce • Processo di fabbricazione standard ⇒ scelta naturale per integrazione con i circuiti logici | <ul style="list-style-type: none"> • Cella + piccola ⇒ densità maggiore, minore costo/bit • Richiede refresh periodico e refresh dopo ogni lettura • Lettura complessa ⇒ accesso + lento • Processo di fabbricazione dedicato ⇒ difficile integrazione con i circuiti logici • La densità elevata impone schemi di indirizzamento non banali |
|--|---|

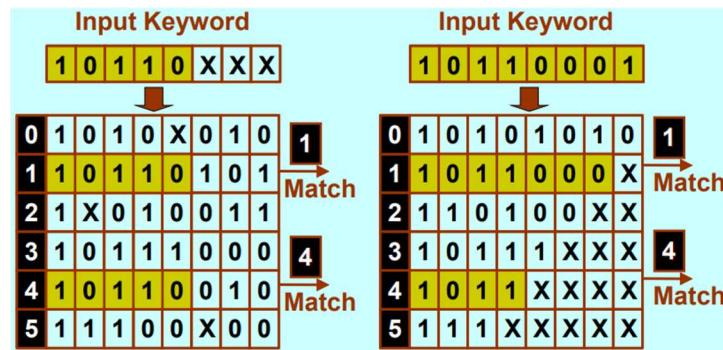
⚠ Nelle SRAM, dato che sono soggette a difetti, abbiamo anche le **CELLE RIDONDANTI**, ovvero celle di memoria aggiuntive integrate nel chip che possono essere usate per sostituire le celle difettose (evitando errori). Queste, **combinate ai codici di correzione errore (ECC)** aumentano la resa di fabbricazione (“yield”).

- **CAM** (“Content Addressable Memory”, o **MEMORIA ASSOCIAUTA**) → sono usate per trovare l'indirizzo di un dato da una tabella di dati memorizzati o usate per trovare il dato ad un certo indirizzo:

Address In → Data Out (oppure) Data In → Address Out

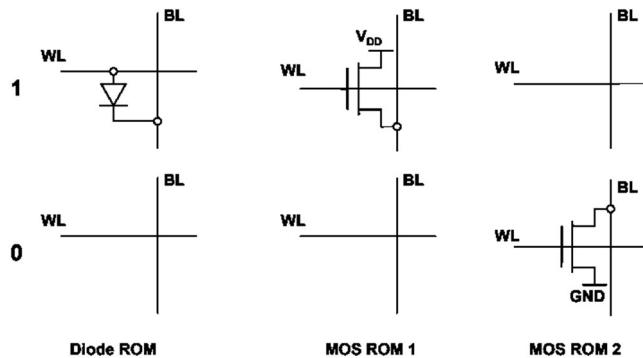


La CAM è **molto veloce**, infatti viene usata nelle routing table e in particolari tipi di cache. Un particolare tipo sono le **TCAM** ("Ternary CAM"), dove dato un input ci dà 2 output possibili (trova 2 dati con una particolare sequenza di bit perché questi dati sono incompleti, ovvero presentano dei "DON'T CARE" [X] al posto di alcuni bit tali che entrambe le sequenze rispettano la ricerca).



Ora vediamo le **MEMORIE NON VOLATILI** (tengono dato anche senza alimentazione):

- **Mask-Programmed ROM** (Mask-Programmed Read Only Memory) → si ha solamente accesso di lettura. Una volta memorizzati i dati necessari, viene realizzata con i connettori direttamente sulla **maschera di fabbricazione**, che viene applicata sul circuito in modo che non sia modificabile. **Quando la sto realizzando, faccio in modo di assegnare già i valori 0/1 alle celle realizzandole in maniera diversa** (oggi si usano le MOS ROM; si usano spesso le MOS ROM 1 con le NAND ROM e le MOS ROM 2 con le NOR ROM):



- **EPROM (Erasable Programmable ROM)** → sola lettura, ma rispetto alla ROM, è modificabile tramite un trattamento a **raggi UV**, ma per un numero limitato di volte. Sono composte dal "**floating gate**" (un gate sospeso, non attaccato a nulla che mantiene le cariche) dove vengono iniettate le cariche tramite "**effetto valanga**" e, dopo che vengono tolte le tensioni elevate, le cariche restano intrappolate nel gate flottante.
- **EEPROM (Elettrically Erasable Programmable ROM)** → sono memorie **modificabili elettricamente** che presentano il **FLOTOX** (floating gate con tunneling oxide).
- **FLASH** → sono l'estensione delle memorie EEPROM (cancellabili elettricamente "on system" [senza usare attrezzatura esterna]), chiamate FLASH perché **la cancellazione avviene a blocchi di dimensione variabile "in un flash"** (rapida cancellazione a blocchi). Sono quindi memorie non volatili che permettono lettura e scrittura. Sono composte da dei **floating gate MOSFET** che mantengono le cariche elettriche (bit). 2 tipi:
 - **Flash NOR** = accesso **casuale**, lettura veloce, **scrittura e cancellazione lente**, usata soprattutto per il **codice** (accesso casuale);
 - **Flash NAND** = accesso a **pagine**, **alta densità e costo inferiore**, **scrittura e cancellazione veloce** (meno la lettura), usata soprattutto per i **dati con accesso sequenziale**.

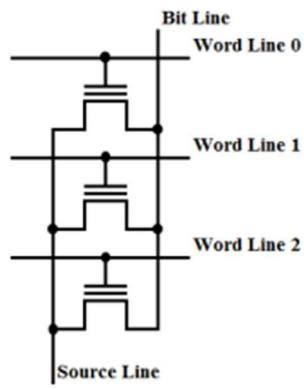


Figure 42: NOR flash

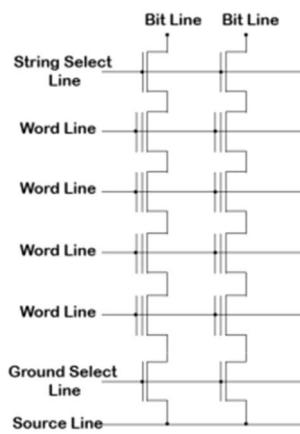


Figure 43: NAND flash

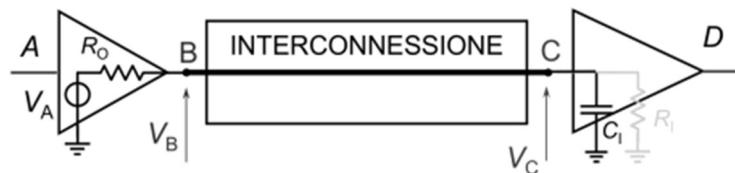
⚠ Dopo molti cicli di cancellazione (~ 10000), la Flash avrà subito una **degradazione** che la danneggia!

⚠ Abbiamo poi le “**serial flash**” (flash seriali), piccole flash a basso consumo che accede ai dati serialmente!

8) BUS e INTERCONNESSIONI

Finora abbiamo considerato un “clock ideale” (senza variazioni su T_{CK}), ma in realtà è presente un “rumore temporale” sul periodo di clock; questo è detto **JITTER** T_J (**variazione di T_{CK}**). Questo incide anche sul calcolo della frequenza di clock. Oltre a ciò, **ogni interconnessione introduce un ritardo di propagazione t_p** (anche questo si aggiunge a T_{CK} e incide quindi su f_{CK}). Oltre a ciò bisogna considerare anche lo **SKEW t_K** , che sarebbe la differenza tra ritardi dei segnali in ingresso nel circuito; sia skew sia jitter modificano la relazioni temporali. Definendo il **tempo di trasmissione** t_{TX} come il ritardo con cui viene rilevata una variazione di stato logico [$t_{TX} \neq t_p$ perché t_{TX} viene riconosciuto solo quando viene attraversata la tensione di soglia V_{TH} , mentre t_p non deve necessariamente attraversare la **soglia**], allora definiamo lo **SKEW t_K** come la differenza $t_K = t_{TXmax} - t_{TXmin}$. Lo SKEW riduce il tempo di setup t_{SU} .

Un’interconnessione **ideale** è vista come un **singolo nodo equipotenziale**, ovvero una connessione nella quale l’uscita (**DRIVER**) e l’ingresso (**RECEIVER**) sono equivalenti per tensioni e correnti (quindi tensione/corrente di input = tensione/corrente di output). Un’interconnessione **reale** però non è equipotenziale, il segnale di uscita non è un’onda quadra perfetta (in quanto subisce **rumore** e **ritardi**). L’interfaccia di connessione è del tipo:



Definiamo **LINEA DI TRASMISSIONE** il più accurato dei modelli **RLC**, necessario per collegamenti lunghi e per segnali con transizioni veloci (veloci non in frequenza, ma con fronti di clock ripidi). Per descrivere una linea di trasmissione abbiamo parametri:

- **Elettrici:**
 - Impedenza caratteristica Z_∞
 - Lunghezza L
 - Velocità di propagazione v_p
 - Tempo di propagazione t_p
- **Fisici:**
 - Induttanza unitaria L_U
 - Capacità unitaria C_U

Con relazioni:

$$\begin{cases} Z_\infty = \sqrt{\frac{L_U}{C_U}} \\ v_p = \frac{1}{\sqrt{L_U C_U}} \\ t_p = \frac{L}{v_p} \end{cases}$$

Parliamo ora delle **RIFLESSIONI**: il segnale arriva in forma digitale; quindi si forma un gradino di tensione da 0V a V_A (tensione del driver). La tensione che esce dal driver è:

$$V_B(0) = \frac{Z_\infty}{R_O + Z_\infty} V_A$$

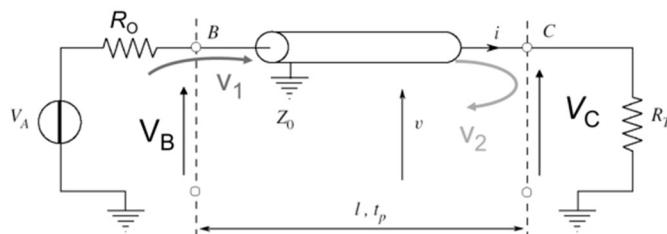
dove R_O = resistenza del driver. Il segnale si sposta quindi tra i 2 punti senza subire distorsioni in $t = t_p$. Una volta che raggiunge la terminazione (lato remoto), dove si trova una **resistenza di terminazione** R_T , se la resistenza è uguale all'impedenza, allora la terminazione assorbe tutta l'energia, altrimenti si genera un'**onda riflessa** che si muove verso il driver. Da qui introduciamo il **COEFFICIENTE DI RIFLESSIONE** Γ_T dove:

$$\Gamma_T = \frac{R_T - Z_\infty}{R_T + Z_\infty}$$

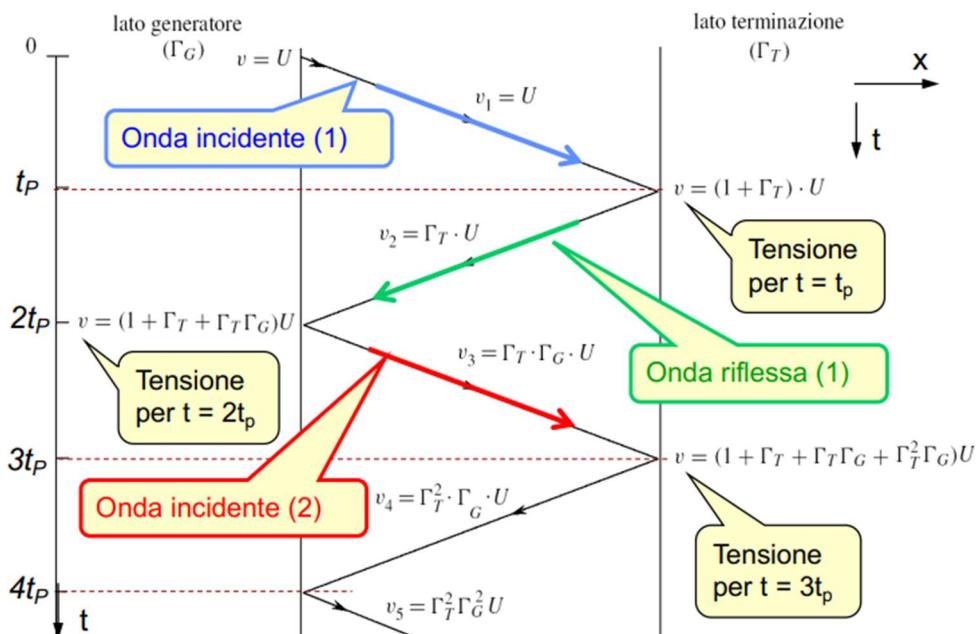
Si parla di **LINEA**:

- **CHIUSA** ($R_T = Z_\infty, \Gamma_T = 0$) → tutta l'energia incidente viene dissipata, senza generare un'onda riflessa;
- **APERTA** ($\Gamma_T = 1$) → si genera un'onda riflessa di ampiezza = a quella incidente con la tensione totale alla terminazione raddoppiata;
- **IN CORTO** ($R_T = 0, \Gamma_T = -1$) → la tensione totale alla terminazione è nulla con tensione dell'onda riflessa uguale e opposta a quella incidente.

Sotto un **ESEMPIO** di circuito di linea di trasmissione, dove $V_B(0) = \frac{Z_\infty}{R_O + Z_\infty} V_A = V_1, V_2 = \Gamma_T V_1$ e $V_C = V_1 + V_2$:



In questo esempio V_1 = **ONDA INCIDENTE** e V_2 = **ONDA RIFLESSA** (che si propaga verso il driver); si possono anche studiare graficamente in diagrammi $V(x, t)$:



⚠ Definiamo "fronti lenti" i fronti confrontabili con t_{TX} .

Parliamo ora di **IWS** e **RWS**. Abbiamo i collegamenti punto-punto (dove le condizioni e i ritardi di propagazione sono ben definiti [noi vediamo questi per IWS e RWS]) e i collegamenti a bus (dove i parametri di propagazione sono variabili). Differenziamo **skew complessivo** (o **globale**) t_{KC} = skew di tutti i ricevitori che comprende la dispersione dei parametri e le differenti posizioni dei receiver, e lo **skew locale** t_K = skew di solo 1 ricevitore di interesse che comprende solo la dispersione dei parametri.

La transizione impressa dal **driver** (il “**near-end**”, ovvero l'estremo sinistro del collegamento) prende il nome di **“PRIMO GRADINO”**, il quale determina un'onda incidente che si propaga lungo la linea. L'ampiezza di questo gradino dipende da R_O e da Z_∞ :

- $R_O < Z_\infty$ [bus veloci] → **primo gradino AMPIO (DRIVER AD ALTA CORRENTE)**: rende possibile attraversare la soglia con il primo gradino (quindi basso tempo di trasmissione t_{TX}). Avviene **IWS**, o commutazione sull'onda **INCIDENTE (Incident Wave Switching)**. Inoltre $\Gamma_D < 0$ (a causa di R_O bassa) quindi avrà oscillazioni e per questo è necessaria una terminazione adattata all'altro estremo (“**far end**”, ovvero l'estremo destro).
- $R_O = Z_\infty$ [buon compromesso: bus più lenti di IWS, ma meno consumi] → **DRIVER ADATTATO**: primo gradino = metà della tensione a regime. Avviene **RWS**, o commutazione sull'onda **RIFLESSA (Reflected Wave Switching)**. Inoltre $\Gamma_D = 0$ quindi non avremo riflessione lato driver.
- $R_O > Z_\infty$ [bus lenti] → **primo gradino BASSO (DRIVER A BASSA CORRENTE)**: non attraverserà la soglia del ricevitore. Le commutazioni (attraversamenti di soglia) avvengono su riflessioni multiple (quindi alto tempo di trasmissione t_{TX}).

In generale valgono le relazioni:

Riflessione	t_{TXmax}	t_K
Incident	t_P	t_P
Reflected	$2t_P$	$2t_P$
Multiple	$\sim N \cdot 2t_P$	-

⚠ Per le terminazioni distinguiamo tra **TERMINAZIONE**:

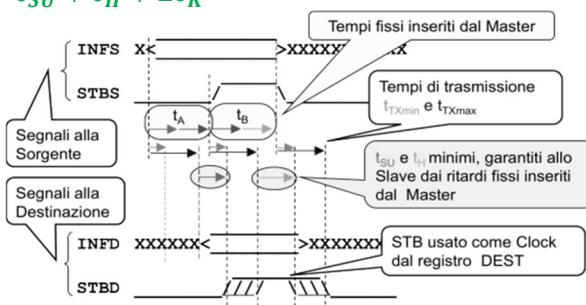
- **PARALLELO** = basso ritardo, elevato consumo;
- **SERIE** = alto ritardo, basso consumo.

Legato a ciò abbiamo i **PROTOCOLLI A LIVELLO CICLO** (o **CICLI DI TRASFERIMENTO**), i quali devono garantire il corretto trasferimento delle informazioni nonostante le variazioni temporali dovute allo skew. Le principali operazioni di trasferimento di informazioni sono **SCRITTURA** (attivato dalla sorgente) e **LETTURA** (richiesto dal ricevitore); per essere eseguite correttamente dobbiamo rispettare t_K , t_H e t_{SU} . Ci sono 2 tecniche per effettuare dei cicli di trasferimento:

- **TEMPORIZZAZIONE FISSA** → ciclo **SINCRONO** che opera sul “**worst case scenario**” (ritardi fissi che possano sempre garantire la corretta ricezione dei dati); nelle FSM parliamo di protocollo CADENZATI (**CLOCKED**):

- **CICLO SCRITTURA SINCRONO**: la sorgente (master) deve conoscere le temporizzazioni della destinazione; il trasferimento avviene così:
 1. Sorgente invia informazione
 2. Aspetta un tempo $t_1 = t_{SU} + 1 \cdot t_K$ per garantire la corretta ricezione al FF
 3. Invia il segnale **STB** (“strobe”) che funziona da clock per il FF
 4. Aspetta un tempo $t_2 = t_H + 1 \cdot t_K$ per garantire il tempo di hold del FF
 5. Rimuove segnale di informazione e STB

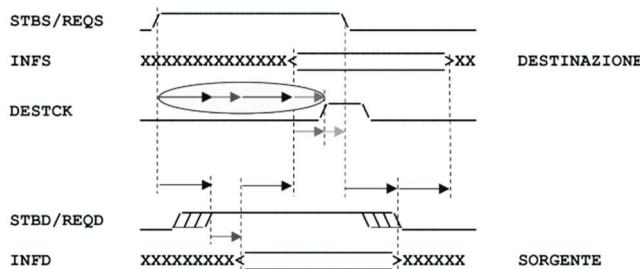
Tempo totale: $t_{WR} = t_{SU} + t_H + 2t_K$



- **CICLO LETTURA SINCRONO**: la destinazione (ovvero il FF, master) aggiunge i ritardi necessari per la richiesta allo slave delle informazioni; il ciclo avviene così:

1. Destinazione invia segnale di richiesta alla sorgente
2. Dopo t_{TX} , è necessario t_A (tempo di accesso) per reperire il dato
3. Sorgente invia dato che arriva a destinazione con ritardo = t_{TX}
4. Dopo aver ricevuto dato, destinazione aspetta t_{SU} richiesto dal FF
5. FF memorizza il dato al fronte di salita
6. Destinazione attende t_H e rimuove il segnale di richiesta
7. Dopo t_{TX} , sorgente rimuove il segnale di dato
8. Dopo t_{TX} , la rimozione del segnale di dato arriva alla destinazione

Tempo totale: $t_{RD} = t_A + t_{SU} + t_H + 4t_{TX,max}$

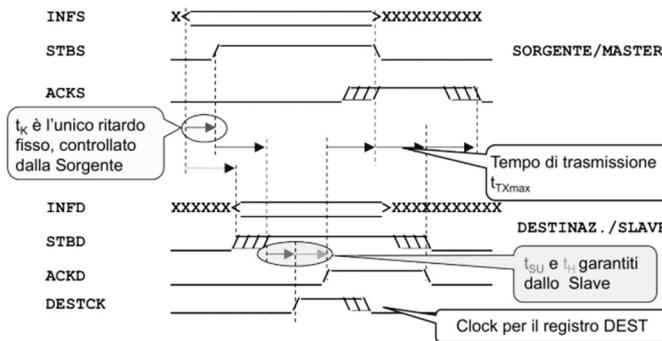


- **TEMPORIZZAZIONE ADATTIVA** → ciclo **ASINCRONO** che opera con segnali di “acknowledge” (conferma, **ACK** [quindi “**handshaking**”]) che devono essere regolati da entrambi trasmettitore e ricevitore:

- **CICLO SCRITTURA ASINCRONO**:

1. Sorgente invia informazione
2. Sorgente aspetta t_K (skew) per garantire che l'ACK (ovvero lo strobe) arrivi a destinazione sempre dopo il dato
3. Invia quindi l'ACK
4. Dopo t_{TX} , la logica di controllo aspetta t_{SU} richiesto dai FF
5. Invia il segnale di clock al FF che memorizza l'informazione
6. Invia ACK alla sorgente per informarla che può rimuovere i segnali di informazione e strobe
7. Dopo t_{TX} , sorgente rimuove i segnali di dato e strobe
8. Dopo t_{TX} , logica rimuove l'ACK
9. Dopo t_{TX} , la rimozione dell'ACK arriva alla sorgente

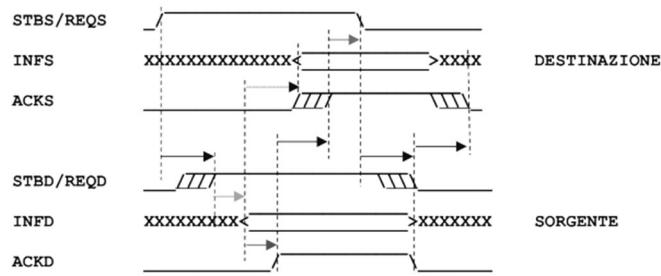
Tempo totale: $t_{WR} = t_K + t_{SU} + t_H + 4t_{TX,max}$ (con t_{SU} e t_H garantiti dallo **slave** [cioè destinazione])



- **CICLO LETTURA ASINCRONO**: si aggiunge 1 segnale di ACK che la destinazione deve inviare alla sorgente per dire che il segnale di informazione è pronto; passaggi:

1. Destinazione invia segnale di richiesta alla sorgente
2. Dopo t_{TX} , serve t_A (tempo di accesso) prima che l'informazione richiesta sia pronta
3. Sorgente invia informazione
4. Dopo t_K (skew), sorgente invia ACK alla destinazione
5. Dopo t_{TX} , FF memorizza informazione al fronte di salita del segnale di ACK
6. Dopo $t = t_{SU} + t_H$ (richiesti dal FF), destinazione rimuove segnale di richiesta
7. Dopo t_{TX} , sorgente rimuove segnali di informazione di ACK
8. Dopo t_{TX} , la rimozione dei segnali arriva alla destinazione

Tempo totale: $t_{RD} = t_A + t_K + t_{SU} + t_H + 4t_{TX,max}$



⚠️ Un nuovo ciclo può iniziare solo quando il precedente è terminato e i segnali di controllo sono tornati "non attivi" (**CICLO CHIUSO** o **COMPLETO**). 2 parametri usati nella temporizzazione sono **latenza dell'informazione** (tempo atteso per ottenere informazione; **dipende da t_K e da t_{TX}**) e **durata del ciclo** (inverso della cadenza [frequenza]; **dipende da t_K e non da t_{TX}**).

Parliamo dei **BUS** (ricordiamo che **MASTER** = attiva le operazioni, **SLAVE** = risponde ai comandi del master): i **protocolli di transazione** (nei sistemi multi-punto o a bus, ovvero con più master e più slave connessi al bus) hanno come fasi **ALLOCAZIONE** (o **ARBITRAGGIO**, selezione del master; se ho 1 solo master, skippata), **INDIRIZZAMENTO** (selezione dello slave; a questo punto master e slave formano un canale punto-punto) e **TRASFERIMENTO** (trasmissione dell'informazione da master a slave).

Le **PRESTAZIONI DI UN BUS** si valutano in base alla quantità di informazione trasferita nell'unità di tempo (ovvero **THROUGHPUT**):

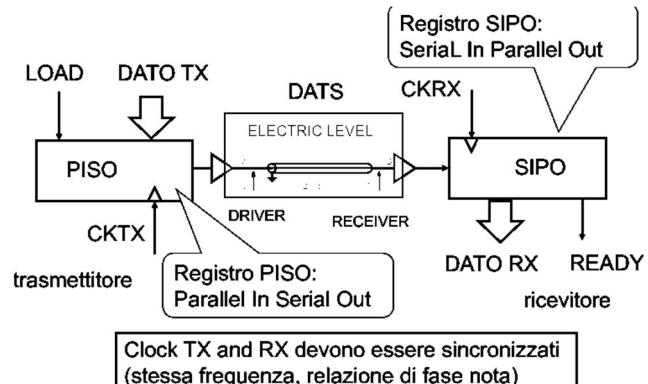
$$T = W \cdot S \rightarrow \begin{cases} W = \text{larghezza del bus} \\ S = \text{velocità del bus} = \left[\frac{\text{cicli}}{s} \right] = \frac{1}{t_c} \quad \text{con } t_c = \text{durata ciclo} \end{cases}$$

Per **migliorare le prestazioni a livello di transazione**, possiamo usare il protocollo **SOURCE SYNCHRONOUS**, dove i segnali di controllo sono gestiti da entrambe sorgente e destinazione, così da poter avviare un nuovo trasferimento prima che sia concluso quello prima (**parallelismo, pipeline**; per ridurre il n° di conduttori, possiamo usare lo stesso conduttore per più segnali [**BUS MULTIPLEXED**]).

Per **migliore le prestazioni a livello di ciclo**, il **più efficiente** è il **DDR** (*double data rate*), dove i segnali di comando (STB/ACK) sono commutati su entrambe le transizioni ($L \rightarrow H$ e $H \rightarrow L$) così da avere minor consumo e maggiore velocità.

Il **BURST** (trasferimento a blocchi) è il trasferimento di 1 segnale di indirizzo seguito da soli segnali di informazione (usato con pacchetti lunghi, dove i dati sono memorizzati in sequenza rispetto all'indirizzo del 1° dato); **si aumenta quindi il throughput**, evitando di inviare per ogni dato il suo indirizzo.

Parliamo di **COLLEGAMENTI SERIALI** (unico conduttore che trasporta bit in tempi diversi): per trasferire N bit dovrò fare N cicli (alta latenza); nei seriali ho pochi segnali (riduco interferenza elettromagnetica). Se trasporto dati e clock sullo stesso conduttore (**clock embedding**) non ho disallineamento dovuto allo skew [si usa **MODEM** per unire e separare dati e clock]. Un segnale è una sequenza di simboli (**simbolo = 1 o più bit**; **bit rate** = bit/s, **baud rate** = simboli/s). Nei collegamenti seriali non ci sono interferenze tra segnali vicini (1 solo segnale per volta), ma tra simboli vicini dello stesso segnale (**ISI, inter symbolic interference**). Vediamo la **struttura di un collegamento seriale**:



Il **SINCRONISMO** avviene a livello di bit (garantisce corretto campionamento del singolo bit) e a livello di carattere (garantisce corretto riconoscimento di MSB e LSB). Ma il corretto campionamento dipende anche da spostamenti di fase e rumore: devo quindi fare una risincronizzazione che può essere periodica o su ogni bit (embedded clock); l'intervallo di risincronizzazione è legato a $t_{CK}/\Delta t_{CK}$. Un collegamento seriale può essere **asincrono** (bit organizzati in caratteri, trasmissione discontinua, sincronizzazione CK ad inizio carattere [periodica]) o **sincrono** (bit organizzati in pacchetti, trasmissione continua, sincronizzazione CK su ogni bit [continua] con estrazione del clock dai dati e sincronizzazione nei dati con bit/byte stuffing).

Parlando invece di **DIAFONIA** (o **CROSSTALK**), ovvero il passaggio di segnale tra 2 canali, abbiamo come casi:

- Tra CONDUTTORI DIVERSI = **accoppiamenti induttivo** (induttanza mutua L_M) e **capacitivo** (capacità mutua C_M), con L_M e C_M distribuiti su tutta la lunghezza del conduttore. Ricordiamo che definiamo DRIVER l'estremo vicino, RECEIVER (o terminazione) l'estremo remoto, FORWARD (onda incidente, diretta), BACKWARD (onda riflessa). Entrambi gli accoppiamenti generano disturbi tra le 2 linee (ovvero la diafonia, crosstalk).

Per ridurre il crosstalk si può rallentare i fronti dei driver della linea disturbante, ridurre C_M e L_M (evitando spire ampie e concatenate [usando collegamenti a massa separati per ogni segnale]) o usare segnali differenziali (immunità al rumore di modo comune e corrente totale costante [riduzione interferenze]); per ridurre gli effetti del crosstalk, si possono filtrare i receiver della linea disturbata e usare "error detection & correction".

Inoltre per distribuire il clock in una connessione multipunto, bisogna usare dispositivi specifici per pilotare **clock multipli** (clock driver), i quali garantiscono ritardi controllati, adattamento lato driver (near end) e stessa lunghezza per tutti i percorsi.

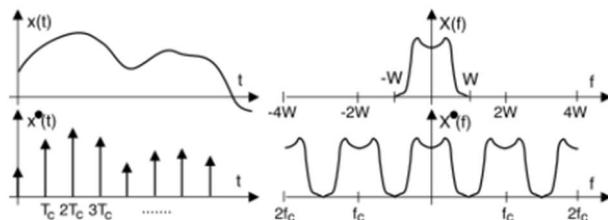
- Sullo STESO CONDUTTORE = **accoppiamento per maglie comuni**, dove le correnti impulsive di alimentazione disturbano i segnali (ci possono essere correnti di parti diverse del circuito con percorsi condivisi). Qui si verifica il **RIMBALZO DI MASSA** ("ground bounce", le correnti impulsive spostano il riferimento; può trasformare 0 in 1 [analogo a **POWER BOUNCE**, che può trasformare 1 in 0]) e **RUMORE DI COMMUTAZIONE SIMULTANEA** (per commutazione simultanea di più uscite), legati a pendenza delle transizioni, quantità di cariche da spostare e induttanza dei collegamenti GND-alimentazione. Per ridurre il rumore di commutazione si possono usare condensatori di disaccoppiamento ("bypass").

9) SISTEMI DI ACQUISIZIONE DATI

I **SISTEMI DI ACQUISIZIONE DATI** sono dispositivi elettronici in grado di rilevare e memorizzare grandezze analogiche/digitali per poi farle elaborare da un microprocessore; dato che i segnali digitali (non esistenti direttamente in natura) sono più facili da analizzare, usiamo la CONVERSIONE A/D e D/A.

Nei **SISTEMI DI CONVERSIONE** (A/D e D/A) abbiamo 2 passaggi: **CAMPIONAMENTO** (discretizzazione nel tempo) e **QUANTIZZAZIONE** (discretizzazione in ampiezza); la conversione analogico-digitale però genera una perdita di informazione (a causa della discretizzazione) da non sottovalutare. Vediamo quindi:

- **CAMPIONAMENTO** → il campionamento è la moltiplicazione del segnale analogico per un treno di impulsi:



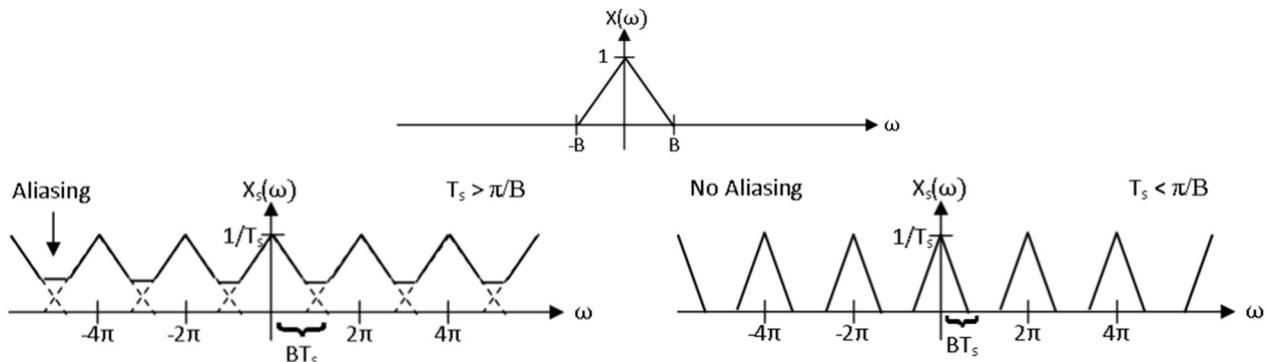
A sinistra un campionamento nel dominio del tempo, a destra nel dominio della frequenza

Dall'immagine si vede che, nello spettro dell'analisi in frequenza del segnale campionato, la **banda fondamentale** si ripete periodicamente per multipli della **frequenza** (o cadenza) di campionamento. Per ricostruire il segnale campionato serve un **FILTRO PASSA-BASSO** (anti-aliasing) che elimina le bande

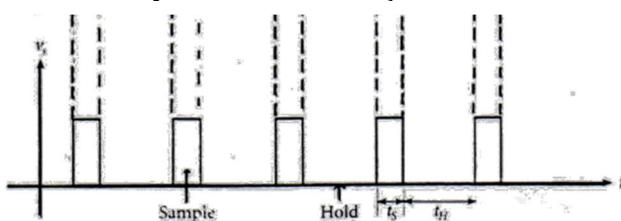
secondarie; per fare in modo che non si perdano informazioni nella ricostruzione, è necessario che banda fondamentale e secondarie non si sovrappongano, ovvero che non avvenga **ALIASING**. L'aliasing si può evitare se si può applicare il **teorema di Nyquist** (o **teorema del campionamento**; se vale Nyquist, segnale può essere sempre ricostruito):

$$F_S > 2F_M \rightarrow \begin{cases} F_S = \text{frequenza di campionamento} \\ F_M = \text{frequenza della banda fondamentale} \end{cases}$$

Nella **realità**, non ci può essere attenuazione infinita, quindi **sarà sempre presente un rumore di aliasing**:



Il campionamento avviene tramite il **SAMPLE/HOLD**, il quale campiona il segnale in ingresso e lo mantiene stabile durante tutta la conversione fino al campione successivo; il mantenimento del sample però modifica lo spettro del segnale (trasforma impulsi in gradini di lunghezza T_H), quindi nella ricostruzione del segnale originario bisogna considerare anche questa distorsione (attenuata da una correzione spettrale).



- **QUANTIZZAZIONE** → la quantizzazione introduce delle **fasce di tensione** per discretizzare gli infiniti valori **di tensione del segnale analogico** (perché il segnale digitale ha **risoluzione finita**, ovvero con N bit posso esprimere 2^N fasce di tensione). La quantizzazione però introduce un **ERRORE DI QUANTIZZAZIONE** (ϵ_q), che è la differenza tra il segnale analogico e il segnale digitale quantizzato da esso:

$$|\epsilon_q| \leq \frac{s}{2^{N+1}}$$

Con una **quantizzazione uniforme** (intervalli di tensione tutti uguali) si può affermare che:

$$|\epsilon_q| \leq \frac{1}{2} LSB$$

Con LSB = ampiezza di ciascun intervallo.

La qualità del segnale quantizzato è rappresentata dal rapporto segnale-rumore SNR_q (signal-noise ratio), ovvero il rapporto tra la potenza di segnale σ_A^2 e la potenza del rumore di quantizzazione σ_{eq}^2 :

$$SNR_q = \frac{\sigma_A^2}{\sigma_{eq}^2} \quad \text{con } \sigma_{eq}^2 = \frac{s^2}{12 \cdot 2^{2N}}$$

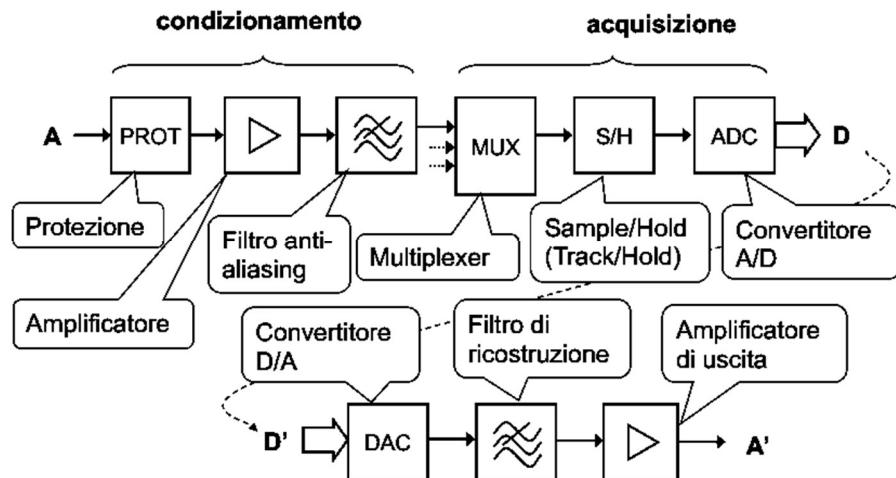
Per i segnali che arrivano dal fondo scala (ovvero ampiezza di segnale = fondo scala), valgono:

- Segnale sinusoidale → $SNR_q = (6N + 1.76)dB$
- Segnale triangolare → $SNR_q = 6N dB$
- Voce → $SNR_q = (6N - 4.77)dB$

La **precisione effettiva** si rappresenta come rapporto tra segnale e rumore totale (SNR_{tot}); da questo deriva il concetto di **ENOB** (effective number of bits), ovvero il numero di bit significativi per il convertitore):

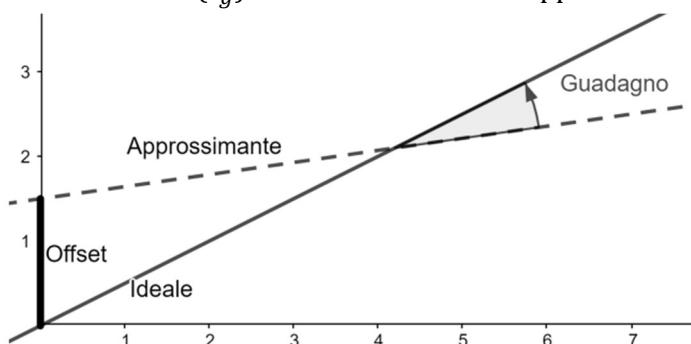
$$N = ENOB = \frac{SNR_{tot} - 1.76}{6}$$

Il sistema di conversione $A \rightarrow D \rightarrow A$ completo ha struttura:



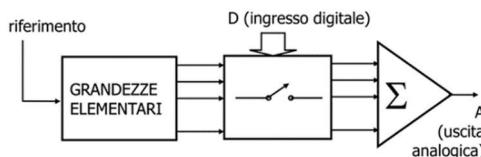
Parliamo ora dei **CONVERTITORI D/A** (digitale-analogico) [DAC], i quali hanno caratteristica ideale lineare ed è soggetto a 2 tipi di errori:

- **ERRORI STATICI**: riguardano il comportamento a regime (ovvero con segnale ad ingresso costante):
 - Errori di **NON-LINEARITÀ** = riguardano il confronto tra la caratteristica reale e la retta approssimante:
 - **INTEGRALE** (ε_{nli}) → max scostamento totale (fascia di ampiezza) tra la caratteristica reale e la retta approssimante;
 - **DIFFERENZIALE** (ε_{nld}) → preso 1 LSB, è lo scostamento locale della caratteristica reale dalla retta approssimante [$\varepsilon_{nld} = A_D - A'_D$];
 - **NON MONOTONICITÀ** → se la non-linearità differenziale è maggiore di 1 LSB.
 - Errori di **LINEARITÀ** = riguardano il confronto tra retta approssimante e retta ideale, ma si possono compensare con una taratura del circuito:
 - **ERRORE DI OFFSET** (ε_0) → valore dell'intercetta dell'asse y , ovvero una tensione V_{off} ;
 - **ERRORE DI GUADAGNO** (ε_g) → differenza tra retta approssimante e retta ideale.



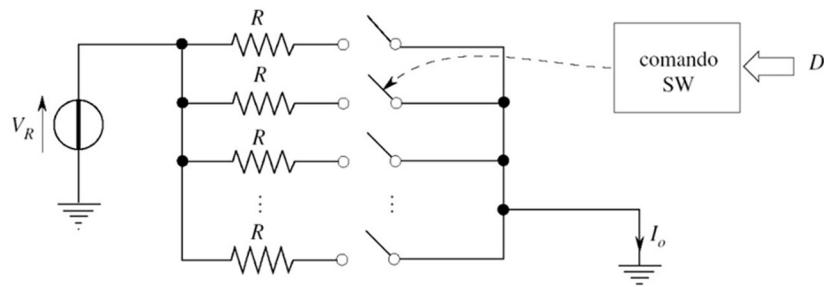
- **ERRORI DINAMICI**: riguardano il comportamento a transitorio (ovvero con un segnale ad ingresso variabile):
 - **TEMPO DI ASSETTO** = Δt impiegato dall'uscita del DAC per portarsi al nuovo valore. Il transitorio si considera esaurito quando l'uscita inizia ad oscillare intorno al nuovo valore, restando nella fascia ± 1 LSB legata all'errore di quantizzazione intrinseco;
 - **GLITCH** = variazione sostanziale dell'uscita in un breve tempo (dovuto alle differenze nei ritardi di commutazione).

Oltre agli errori, ora vediamo la **struttura del DAC**. L'uscita analogica A è ottenuta dalla somma delle grandezze elementari in cui la grandezza elettrica di riferimento è stata scomposta:

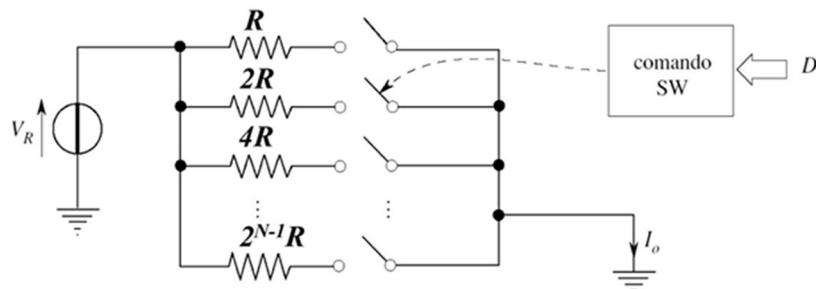


Le tecniche di conversione di base sono:

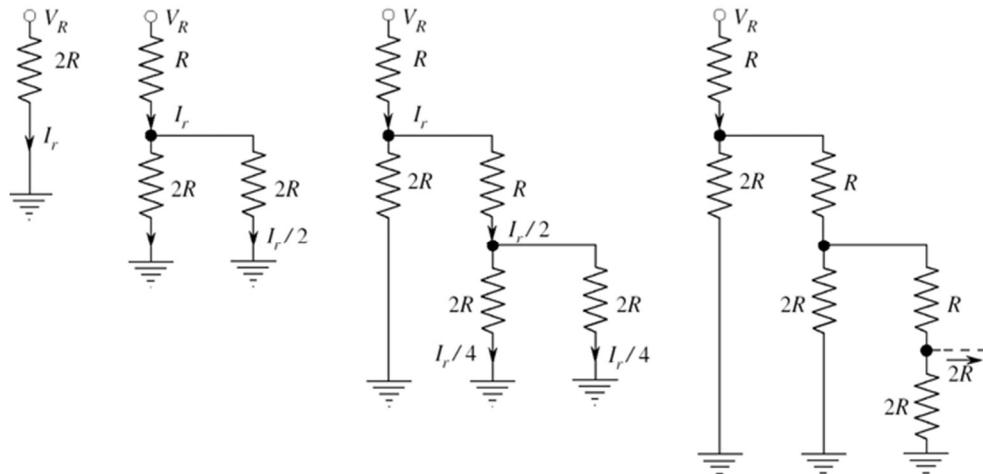
- **A GRANDEZZE UNIFORMI** → basata sulla **somma di variabili di peso uguale** (ovvero 1), inserite in quantità corrispondente al valore di ingresso digitale. L'output sarà $out = D \cdot LSB$. Il circuito è composto da **resistenze tutte uguali in parallelo**, il cui ramo può essere aperto o chiuso in base al valore digitale D . L'uscita è una corrente ottenuta come somma di correnti uguali.



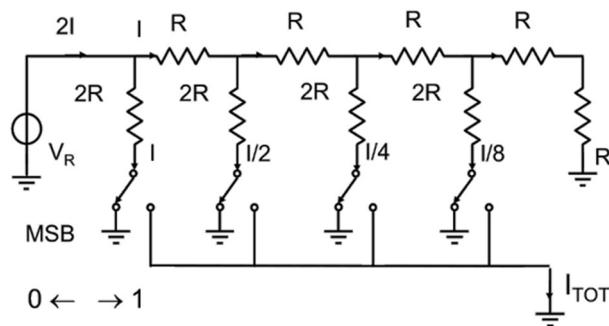
- **A GRANDEZZE PESATE** → basata sulla **somma di variabili di peso corrispondente alle potenze di 2**, ovvero $out = 2^i \cdot D_i$. Il circuito è come quello delle uniformi, ma le **resistenze sono pesate e hanno la funzione di "emulare" le potenze di 2** (es. voglio 9, uso la resistenza in posizione 0 e in posizione 3 perché $9 = 2^3 + 2^0$). Questa configurazione ha il **difetto** di avere una forte dinamica dei valori delle resistenze (da 0 a 2^{N-1}).



Questo difetto può essere risolto grazie ad una configurazione chiamata **RETE A SCALA**:



La creazione della rete a scala è facile: basta **dividere in 2 la corrente che circola nel ramo più a destra**. Questa configurazione, oltre a diminuire la dinamica, **usa solamente resistenze di valore R e 2R** (senza dover usare quindi grosse resistenze), perciò può essere espansa senza vincoli sui valori delle resistenze. Le **uscite in tensione** si possono ricavare usando la conversione Norton/Thevenin. Sotto un'altra rappresentazione della rete a scala:



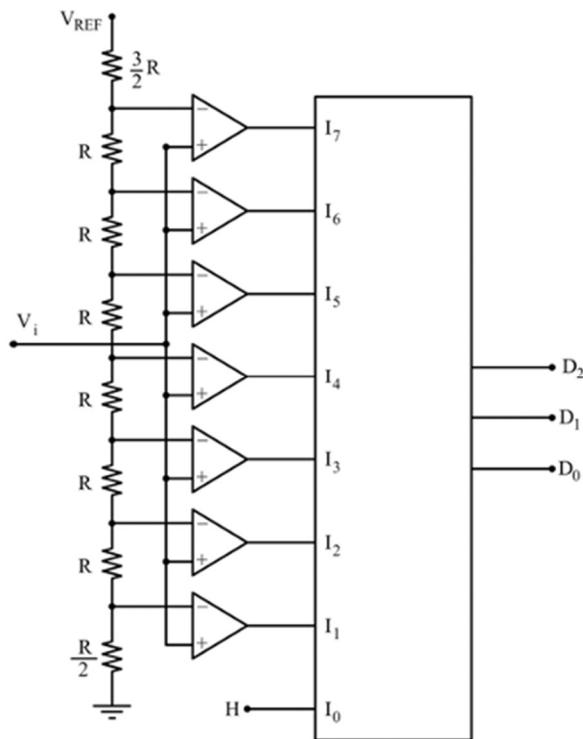
La corrente totale di questa rete avrà forma del tipo $I_{TOT} = I + \frac{I}{4} + \frac{I}{8}$, che ha rappresentazione binaria 1011.

Inoltre le grandezze da sommare possono essere delle **cariche elettriche** (in questo caso usare condensatori al posto delle resistenze).

Il parametro più critico che può portare ad errori di non-linearietà è la **precisione dei rami MSB**, in quanto maggiore è il peso di un ramo, maggiore è l'errore in uscita (quindi **diversi rami pesati determinano errori differenti in uscita**).

Parliamo ora dei **CONVERTITORI A/D** (analogico-digitale) [ADC], ovvero un circuito elettrico in grado di convertire un segnale continuo (come una tensione) in un segnale discreto. Esistono diversi ADC classificabili in base a **velocità** (legata al tempo di conversione T_C [**n° conversioni al secondo**]) e **complessità** (legata al **n° di comparatori usati**). I 2 parametri sono **inversamente proporzionali**. Vediamo i tipi di ADC:

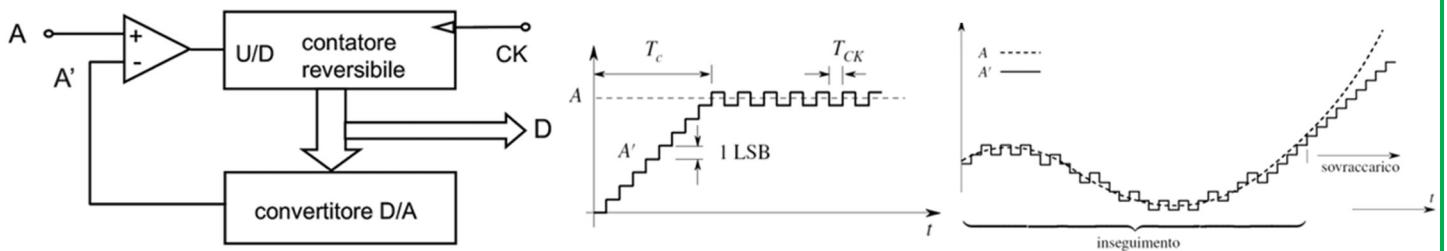
- **ADC Flash Parallelo** → **velocissimo**. Prende il segnale analogico in ingresso e produce un'uscita lineare che poi sarà convertita in valori binari da un **encoder**. È formato da 2^{N-1} **comparatori** (quindi **complesso**) che in uscita producono 0/1 in base alla comparazione con il valore di soglia. Opera in parallelo, ovvero **tutti i comparatori decidono contemporaneamente il valore in uscita comparato alla soglia; basta 1 solo ciclo di confronto per convertire N bit**.



- **Convertitore ad Inseguimento** → è un circuito che **converte l'uscita D con un DAC in reazione**, così da poter confrontare il segnale approssimato A' con il segnale in ingresso A . Ad ogni colpo di clock, il contatore viene incrementato in base al rapporto $\frac{A}{A'}$. In particolare:

- Se $A' < A$, il contatore viene incrementato di 1 LSB
- Se $A' > A$, il contatore viene decrementato di 1 LSB

Si procede così fino a che il segnale di reazione A' non raggiunge la migliore approssimazione di A . È **molto lento** perché nel caso peggiore servono **2^N cicli di confronto per convertire N bit**, ma è **semplice** perché serve solo **1 comparatore**. L'inseguimento può essere costante o variabile.



- **Convertitore ad Approssimazioni Successive** → è un circuito con reazione che **ad ogni colpo di clock** determina il valore di 1 bit consecutivamente a partire dal MSB fino al LSB: l'ingresso **A** viene **ogni volta confrontato con la metà del campo rimasto** (inizialmente il campo è pari al fondo scala **S**):

 - Se **A** si trova nella metà inferiore del campo rimasto, bit = 0 e il campo corrente è la metà inferiore;
 - Se **A** si trova nella metà superiore del campo rimasto, bit = 1 e il campo corrente è la metà superiore.

È **lento** in quanto nel caso peggiore servono **N cicli di confronto per convertire N bit**, ma è **semplice** in quanto occorre solamente **1 comparatore**.

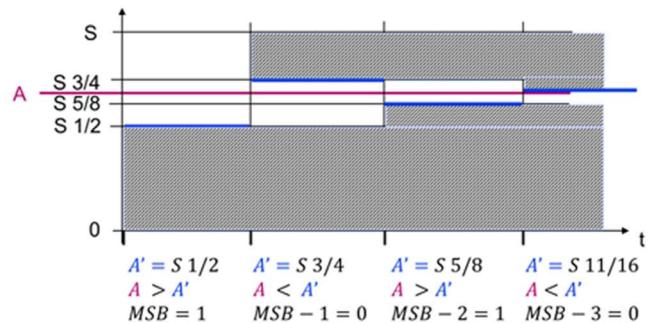
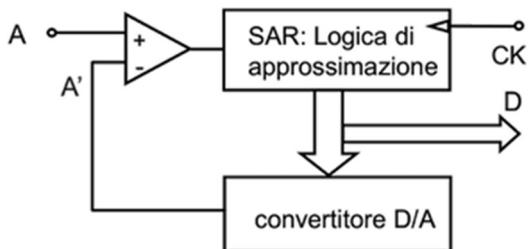
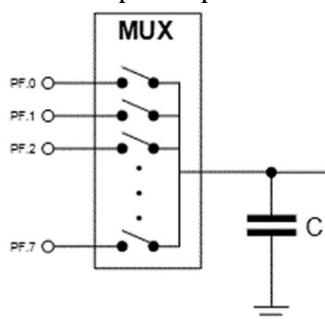


Figure 66: Andamento del campo corrente

- ⚠ Esiste anche il **Convertitore Δ** (convertitore **differenziale** [viene quantizzata la differenza tra valore attuale e precedente] a integratore [segnale ricostruito con integrale]) e **Convertitore $\Sigma\Delta$** (si aggiunge un integratore all'ingresso e un derivatore all'uscita per limitare lo slew rate, e quindi ampliare la dinamica).

Parliamo ora di **MULTIPLEXER** e **SAMPLE/HOLD**:

- **MULTIPLEXER (MUX)** → banco di N interruttori realizzato con transistori MOS, dove tramite un comando di selezione viene chiuso solamente 1 interruttore per selezionare 1 canale di ingresso tra gli N canali (ogni canale deve avere un proprio filtro). Gli elementi che producono **effetti di non-idealità** sono:
 - Resistenza equivalente R_{ON} per l'interruttore chiuso
 - Corrente di perdita I_{OFF} per gli interruttori aperti
 - Capacità parassita C_P (costituita dalle capacità parassite del MUX e del carico)

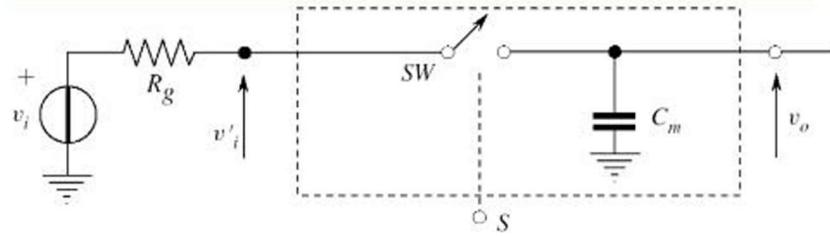


- **SAMPLE/HOLD (S&H)** → campionatore usato come interfaccia tra un segnale analogico (che varia velocemente nel tempo) e un dispositivo successivo (solitamente un ADC). Il suo scopo è **mantenere il valore analogico costante per il tempo necessario all'ADC**. Le fasi operative di questo modulo sono:

 - **TRACK** (acquisizione + inseguimento): **uscita è uguale all'ingresso** (ovvero S&H = amplificatore con guadagno unitario). Abbiamo **errori statici** (guadagno, offset e non-linearietà) e **dinamici** (tempo di settling, limiti di banda).
 - **SAMPLE** (campionamento): **viene campionato il valore dell'ingresso da mantenere costante**. I 2 parametri da considerare sono **tempo di apertura** (Δt tra comando di hold e completa apertura dell'interruttore) e **tempo di assetto** (Δt in cui l'uscita scende ad un valore più basso di quello campionato, con una variazione di ampiezza chiamata "piedistallo").

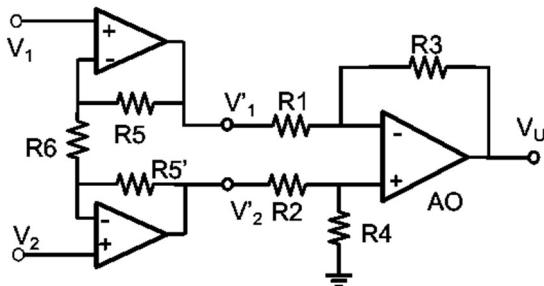
Il tempo di assetto è affetto da rumore dovuto al jitter di campionamento, che produce un errore in ampiezza del segnale. Possiamo valutare il **SNRj** (signal-to-noise ratio del jitter) calcolando il rapporto tra segnale e rumore.

- **HOLD** (mantenimento): valore campionato viene mantenuto costante durante la conversione. Abbiamo **errori di decadimento** (valore dell'uscita diminuisce nel tempo) e **di feedthrough** (una minima parte delle variazioni dell'ingresso viene riportata sull'uscita a causa dell'imperfetto isolamento).
- **NUOVA ACQUISIZIONE**: uscita passa dal valore mantenuto al valore corrente dell'ingresso.



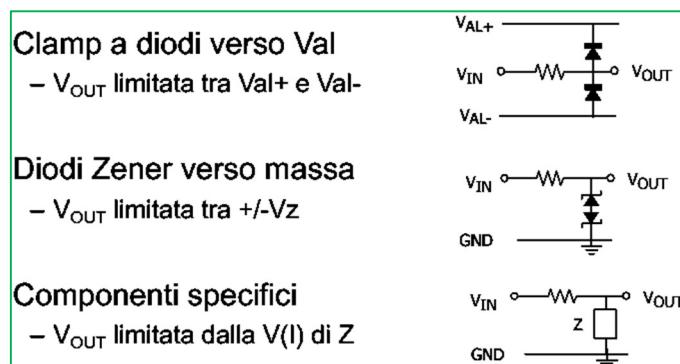
Parliamo ora di **CONDIZIONAMENTO DEL SEGNALE**: il convertitore A/D può avere come ingresso **tensione** o **corrente**, riferito a massa o differenziale. Per adattare il segnale di ingresso all'A/D, come dinamica e come tipo di segnale, si usa un **amplificatore di condizionamento** (ce ne sono di diversi tipi).

⚠ Noi in questo corso useremo un **AMPLIFICATORE DIFFERENZIALE** (amplifica i segnali differenziali [A_d alto] e attenua i segnali di modo comune [A_{CM} basso $\rightarrow 0$], avendo un **CMRR** [rapporto di reiezione, ovvero A_d/A_{CM}] richiesto). Il problema però è che l'asimmetria del differenziale classico trasforma il segnale di modo comune in segnale differenziale, peggiorando CMRR; per questo vengono messi 2 voltage follower sui 2 ingressi (1 ciascuno) per fare una “simmetrizzazione” (in modo da avere un'impedenza Z_i alta e uguale su entrambi gli ingressi); su questi voltage follower si inserisce un guadagno, ottenendo l'**AMPLIFICATORE DA STRUMENTAZIONE**:



$$V_u = (V_2 - V_1) \left(2 \frac{R5}{R6} + 1 \right) \left(\frac{R3}{R1} \right)$$

Il segnale proveniente dall'esterno è soggetto a cariche statiche, disturbi elettromagnetici, rumore, contatti accidentali etc... Per non danneggiare il circuito, bisogna **limitare la tensione di ingresso**. Per **limitare la tensione** di uscita tra valori specifici, si usano dei **CIRCUITI DI PROTEZIONE** per proteggere le componenti del circuito; questi circuiti di protezione sono **CLAMP A DIODI** verso massa/alimentazione, **DIODI ZENER** e altri **COMPONENTI SPECIALI**.



Oltre ai circuiti di protezione, abbiamo il **FILTRO ANTI-ALIASING**, un filtro analogico usato prima del campionamento di un segnale, al fine di restringere la banda del segnale per soddisfare approssimativamente il **teorema di Nyquist** (o del campionamento); sapendo che f_B = banda del filtro ricostruttivo, f_S = frequenza di campionamento, il filtro anti-aliasing:

- non deve attenuare fino a f_B
- deve attenuare di un fattore SNR_A a $(f_S - f_B)$
- da f_B a $(f_S - f_B)$, la dinamica è $\frac{f_S - f_B}{f_B}$. Inoltre, in questo intervallo, ogni polo attenua di un fattore:

$$A_{P_{dB}} = 20 \log_{10} \frac{f_S - f_B}{f_B}$$

Inoltre, il numero di poli necessari per progettare il filtro si calcola con:

$$P = \frac{SNR_A}{A_{P_{dB}}}$$

Per quanto riguarda l'**ERRORE TOTALE** e **SNR**: il rapporto segnale-rumore totale di un sistema di conversione A/D (SNR_{TOT}) dipende da vari errori, tra cui:

- **ERRORE DI QUANTIZZAZIONE**: legato al n° di bit del convertitore A/D
- **RUMORE DI ALIASING**: dovuto alle sovrapposizioni degli spettri del segnale campionato
- **ERRORE DI JITTER**: introdotto dall'S&H in fase di campionamento
- **ERRORI DI CONDIZIONAMENTO** (es. errore di guadagno degli amplificatori)

Avremo quindi:

$$SNR_{TOT} = 10 \log_{10} \frac{P_S}{\sum_i P_{N_i}} = 10 \log_{10} \frac{1}{\sum_i A_i} \text{ con } A_i = \frac{P_{N_i}}{P_S} = 10^{\frac{-SNR_i}{10}}$$

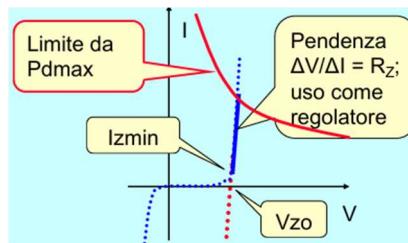
Ma l'errore totale è espresso dall'**ENOB** (Effective Number Of Bits), ricavato dall' SNR_{TOT} :

$$ENOB = \frac{SNR_{TOT}}{6} - 0.3$$

⚠ Parlando dei **FILTRI** usati per i filtri-antialiasing, questi sono dei filtri passa-basso. Per realizzare la f.d.t desiderata del filtro (cosa che noi non faremo, **noi calcoleremo solo il numero di poli necessari**), useremo una f.d.t ideale approssimata con rapporti di polinomi, usando approssimazione di Bessel (fase lineare), Butterworth (no ondulazione in banda passante) e Chebicheff (ondulazione in banda passante).

10) CIRCUITI DI POTENZA

Ogni giunzione di un diodo ha una tensione di "breakdown", oltre cui si generano danni permanenti al circuito; nel **diodo Zener**, il breakdown è voluto e controllato, in quanto sono fatti apposta per lavorare in breakdown [a differenza di un normale diodo che lavora in polarizzazione diretta/inversa, il diodo Zener lavora solo in polarizzazione inversa] (vengono pertanto usati nei circuiti di protezione, nei regolatori di tensione e per generare tensione di riferimento pari alla tensione di breakdown). Nell'immagine vediamo che sotto la curva rossa ho la "safe zone" a livello di potenza, mentre la retta blu è la **zona in cui esso può funzionare da generatore di tensione**; è comunque necessario che passi corrente affinché il diodo funzioni correttamente, compresa in un range (oltre questo il diodo si brucia):



Abbiamo poi i **TRANSISTOR BJT** (dispositivi bipolari di potenza) usati per amplificare (**amplificatori**) o commutare (**interruttori**) i segnali elettrici. Un BJT è costituito da 3 strati di materiale conduttore; si basa sul **controllo della corrente tra 2 terminali (collettore e emittitore)** tramite una piccola corrente applicata al **3° terminale (base)**, con relazione $I_{collettore} = \beta I_{base}$ (con β = guadagno di corrente del transistor). Ha 3 modalità operative (regioni):

- Regione di **SATURAZIONE** → transistor acceso, usato come **interruttore chiuso**;
- Regione **ATTIVA** → transistor come **amplificatore** (quindi qui $I_{collettore} = \beta I_{base}$);
- Regione di **INTERDIZIONE** → transistor **spento** (correnti ≈ 0).

Ci sono poi i **TRANSISTOR MOSFET**, dispositivi a semiconduttore usati per amplificare o commutare segnali elettronici. Sono meno complessi e più resistenti alla temperatura dei BJT. Sono basati su **4 terminali** principali:

- **GATE (G)** = controlla la corrente tra D e S, mediante una tensione applicata al G
- **DRAIN (D)** = esce la corrente dal MOS
- **SOURCE (S)** = entra la corrente nel MOS

Le modalità operative del MOSFET sono:

- Regione di **ON** → $V_{GS} > V_{TH}$ → **interruttore chiuso** (passa corrente tra D e S)
- Regione di **OFF** → $V_{GS} < V_{TH}$ → **interruttore aperto** (non passa corrente)
- Regione di **Triodo (Lineare)** → V_{DS} piccolo → **corrente di drain proporzionale a V_{DS}**
- Regione di **Saturazione** → V_{DS} grande → **corrente di drain dipende solo da V_{GS}**

Parliamo ora di **SOA (SAFE OPERATING AREA)**, una regione che definisce i limiti di tensione, corrente e potenza entro cui il dispositivo elettronico può lavorare in sicurezza senza subire danni:

- **TENSIONE** → $V < V_{BREAKDOWN}$ → evita la rottura della giunzione
- **CORRENTE** → $I < I_{MAX}$ → evita il surriscaldamento dei conduttori
- **POTENZA** → $VI < P_{d_{MAX}}$ (potenza dissipabile massima) → evita surriscaldamento eccessivo

Legata alla SOA, abbiamo il **MODELLO TERMICO**, rappresentabile con un **circuito equivalente termico** dove:

- **Potenza dissipata** = rappresentata da generatore di corrente
- **Temperatura** = rappresentata da tensione
- **Conduzione calore** = rappresentata da una resistenza termica (θ) espressa in Celsius/Watt

Da questo modello si trova che la potenza dissipabile massima deve essere:

$$P_D < \frac{T_{j_{max}} - T_A}{\theta_{JA}} \quad \text{con} \quad \begin{cases} T_A = \text{temperatura ambiente} \\ T_j = \text{temperatura di giunzione massima} \\ \theta_{JA} = \text{resistenza termica dalla giunzione all'ambiente} \end{cases}$$

Oltre a ciò abbiamo il **DERATING DELLA POTENZA**, ovvero la **potenza dissipabile (P_D) diminuisce con l'aumentare della temperatura ambiente (T_A); se T_A raggiunge $T_{j_{max}}$, $P_D = 0$.**

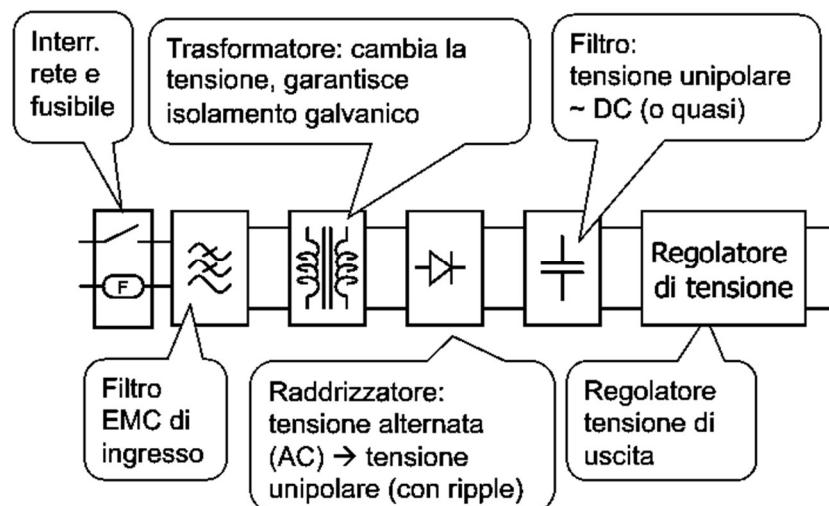
Oltre a ciò, c'è il **PERCORSO TERMICO** dalla giunzione all'ambiente, composto da:

- Giunzione a Case (θ_{JC}) = resistenza termica legata al contenitore del dispositivo
- Case a Dissipatore (θ_{CS}) = bloccaggio termico tra case e dissipatore
- Dissipatore a Ambiente (θ_{SA}) = resistenza termica legata al dissipatore e alle condizioni operative

⚠ Nelle regioni di ON e OFF, la potenza dissipata è minima/nulla, ma è massima negli stati intermedi (regione attiva); per minimizzare la potenza dissipata, bisogna usare una **commutazione rapida**, ma questo genera disturbi elettromagnetici (EMI) [bisogna cercare un compromesso].

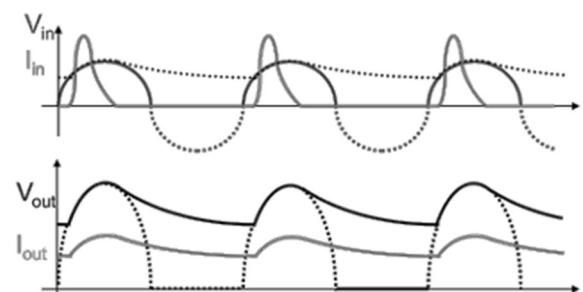
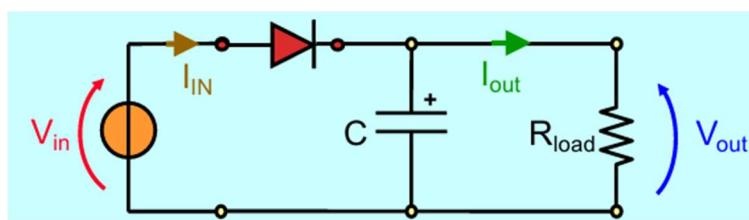
⚠ I carichi induttivi possono generare sovratensioni significative all'apertura del circuito: bisogna prevedere un percorso quindi per la corrente induttiva nel passaggio da ON a OFF, e bisogna limitare la tensione sul Drain per proteggere il dispositivo.

In tutti i **MODULI DI POTENZA**, vogliamo **alto rendimento** e **basse perdite**. Vediamo lo **SCHEMA CLASSICO di una PSU** (unità di alimentazione, generatore di tensione):



Il **RADDRIZZATORE** è un dispositivo usato per trasformare la corrente elettrica da alternata a continua; l'energia elettrica viene **DISTRIBUITA IN AC**, ma **UTILIZZATA IN DC**, quindi c'è **necessità di conversione**. Ci sono raddrizzatori:

- **A SINGOLA SEMIONDA** → il segnale di ingresso (sinusoidale) viene applicato ad un **diodo** [diodo ideale fa passare corrente solo in una sola direzione, mantenendo ai suoi capi una tensione $\cong 0$; quindi nella direzione voluta si comporta come cortocircuito, nell'altra direzione come circuito aperto]. Il diodo è in serie alla **resistenza di carico**. **Se il catodo è rivolto verso il carico, il diodo consente il passaggio delle sole semionde positive**, lasciando a zero il valore della tensione in corrispondenza delle semionde negative.

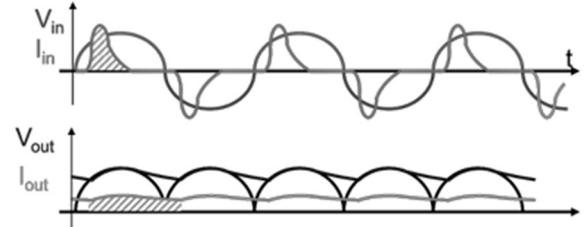
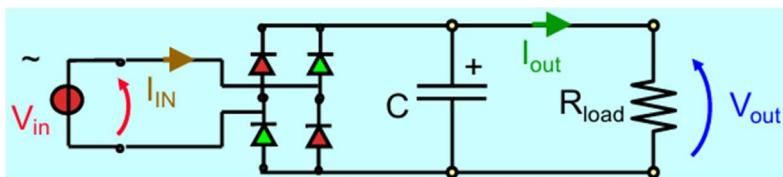


I parametri di questo raddrizzatore sono:

- **TENSIONE DI RIPPLE** → $V_{ri} = \frac{I_o}{fC}$
- **TENSIONE DI PICCO** → $V_{pi} = V_{in} - V_d$ con V_d = caduta di tensione singola su un diodo

- VALORE MEDIO $\rightarrow V_m = \frac{V_{pi}}{\pi}$
- VALORE EFFICACE $\rightarrow V_{eff} = \frac{V_{pi}}{2}$

- **A SEMIONDA INTERA** → è formato da 2 diodi che trasformano un'onda alternata in un'onda sempre positiva, "capovolgendo" la semionda negativa (tipo modulo, valore assoluto).



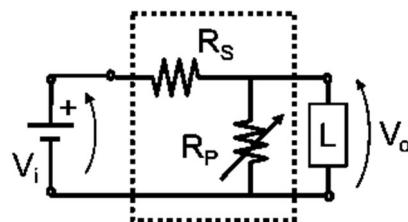
I parametri di questo raddrizzatore sono:

- TENSIONE DI RIPPLE $\rightarrow V_{ri} = \frac{I_o}{2fC}$
- TENSIONE DI PICCO $\rightarrow V_{pi} = V_{in} - 2V_d$ con V_d = caduta di tensione doppia sui diodi
- VALORE MEDIO $\rightarrow V_m = \frac{2V_{pi}}{\pi}$
- VALORE EFFICACE $\rightarrow V_{eff} = \frac{V_{pi}}{\sqrt{2}}$

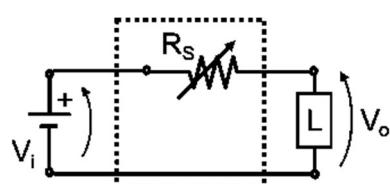
⚠️ Quindi: a singola semionda → elimina parti negative dell'onda; a onda intera → capovolge parti negative dell'onda.

Il **REGOLATORE DI TENSIONE** quindi mantiene una tensione di uscita costante (V_{out}) anche in presenza di variazioni della corrente di carico (I_L) [in questo caso parliamo di resistenza equivalente di uscita $R_{out} = \frac{\Delta V_{out}}{\Delta I_L}$] o della tensione di ingresso (V_I) [regolazione ingresso/uscita $S_i = \frac{\Delta V_{out}}{\Delta V_I}$]. I REGOLATORI ATTIVI (LINEARI) possono essere di 3 tipi (i regolatori integrati sono inclusi negli altri, come variazioni):

- **REGOLATORI PARALLELO (SHUNT)** = usano diodi Zener e riferimenti di tensione. Per ottenere una V_{out} costante, si usa il **PARTITORE DI CORRENTE** (vario il rapporto di partizione agendo sul ramo parallelo R_P). Il regolatore parallelo **con diodo Zener** è semplice e adatto per basse potenze, ma ha bassa efficienza; qui i riferimenti di tensione usano correnti tra $I_{z_{min}} = 5mA < I < I_{z_{max}}$ per limitare la potenza dissipata:



- **REGOLATORI SERIE** = usano transistor e amplificatori operazionali per permettere la limitazione di corrente. Funziona come una resistenza variabile (BJT o MOS), controllata da un controllore che confronta l'uscita V_{out} con una tensione di riferimento V_r (retroazione negativa). Per mantenere una V_{out} costante, si usa il **PARTITORE DI TENSIONE** (cambio il rapporto di partizione agendo sul ramo serie R_S). I regolatori serie base usano resistenze e Zener per ripartire la corrente tra Zener e carico, con bassa efficienza e correnti gestibili limitate; l'uso della tensione sullo Zener V_Z e di un amplificatore di corrente con Emitter follower, migliora la regolazione.

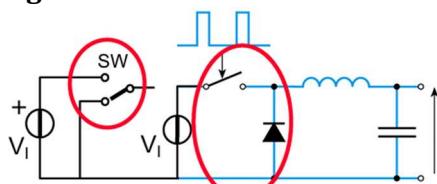


Il **rendimento** dei regolatori serie è influenzato dalle **perdite dovute alla caduta di tensione e alle correnti di perdita**:

$$\eta = \frac{V_{out}}{V_{in}}$$

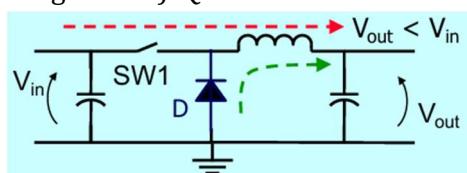
Per migliorare il rendimento posso ridurre la caduta di tensione (usando Low DropOut) e le correnti di perdita, o ancora meglio posso passare ai **regolatori a commutazione**.

- **REGOLATORI INTEGRATI** = 3 terminali e bassa caduta di tensione (Low DropOut [LDO], quindi più efficienza).
- **REGOLATORI A COMMUTAZIONE** = il controllo della potenza viene fatto variando il **duty cycle** (ciclo di lavoro) di un interruttore (usato per controllare l'energia trasmessa al carico) Il **PSU** (Power Supply Unit, ovvero il sistema di alimentazione) a commutazione è molto simile a quello lineare, ma ci sono alcune differenze:
 - o Il **regolatore è a commutazione**, ovvero l'**elemento base è l'interruttore**;
 - o Il **trasformatore è posto dopo il regolatore** (per poter lavorare a frequenze più alte);
 - o Nella parte di controllo in retroazione, il **segnale ad onda quadra che commuta l'interruttore è fornito da un oscillatore digitale**.



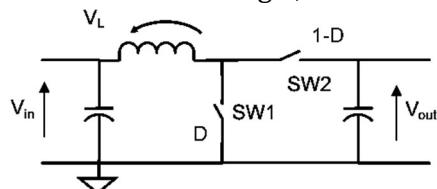
I regolatori a commutazione hanno **rendimento $0.8V < \eta < 0.9V$** , più alto dei regolatori lineari grazie alle basse perdite dell'interruttore e della cella LC (induttore-condensatore; l'induttanza L mantiene costante la corrente). Ci sono **vari tipi** di regolatori a commutazione (ricorda che $D = \text{duty cycle}$):

- **BUCK ($\eta = D$)** → riduce la tensione d'uscita rispetto a quella d'entrata ($V_{out} < V_{in}$). Alterna stati ON e OFF per controllare la corrente e la tensione di uscita, usando un induttore per tenere costante la corrente. Ha lo svantaggio di avere ondulazione in uscita e generazione di EMI (interferenze elettromagnetiche). Quindi:



→ **SW1 chiuso**: diodo polarizzato inversamente, corrente nulla (SW2 OFF)
→ **SW1 aperto**: induttanza mantiene corrente costante, che circola solo attraverso il diodo (SW2 ON)

- **BOOST ($\eta = \frac{1}{1-D}$)** → aumenta la tensione d'uscita rispetto a quella d'entrata ($V_{out} > V_{in}$). Simile al Buck, ma con configurazione diversa degli interruttori e delle'induttore per aumentare la V_{out} . Durante lo stato **ON**, l'induttore accumula energia; durante lo stato **OFF**, l'energia viene trasferita al carico.



- **BUCK-BOOST ($\eta = -\frac{D}{1-D}$)** → **polarità invertita**. Combina Buck e Boost per generare una tensione di polarità inversa rispetto all'ingresso.
- **FLYBACK** → fornisce **isolamento galvanico** (isolare elettricamente l'entrata dall'uscita) usando un trasformatore. È come un Buck-Boost, con un trasformatore al posto dell'induttanza; per la regolazione della V_{out} occorre una reazione isolata.

Quindi come dicevo su, la **PSU a commutazione (ALIMENTAZIONE A COMMUTAZIONE)** è del tipo: A = filtro EMF; B = AC → DC; C = regolatore a commutazione (DC → AC); D = AC → AC; E = AC → DC.

