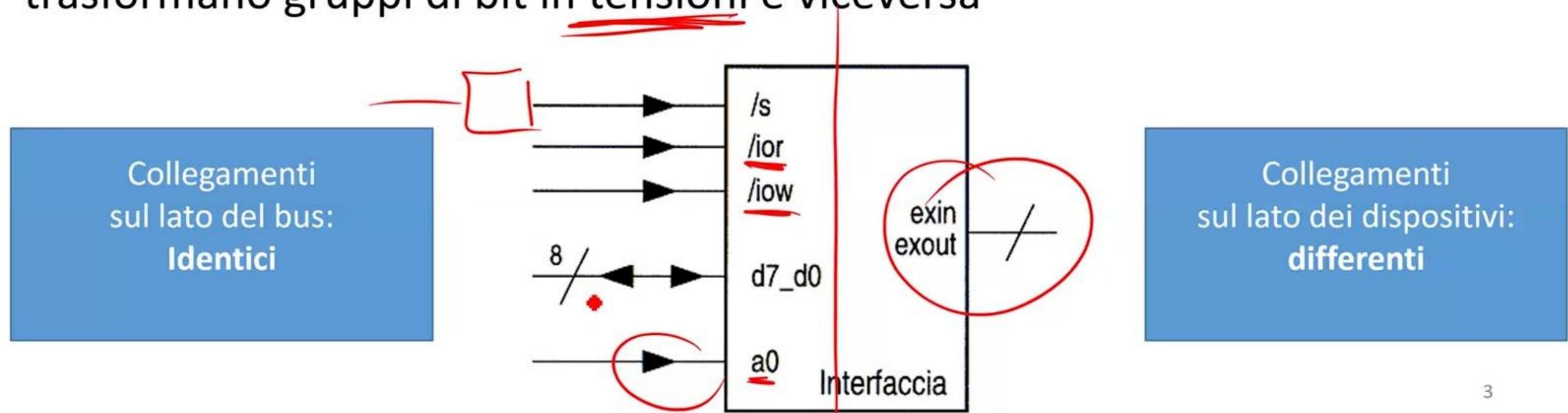


Interfacce

- Descriveremo le varie interfacce che completano il calcolatore
- **Parallele**, che sono in grado di colloquiare con dispositivi ai quali inviano (o dai quali prelevano) **un byte alla volta**
- **Seriali**, che colloquiano con dispositivi con i quali scambiano **un bit alla volta**.
- quelle per la **conversione analogico/digitale e digitale/analogica**, che trasformano gruppi di bit in tensioni e viceversa

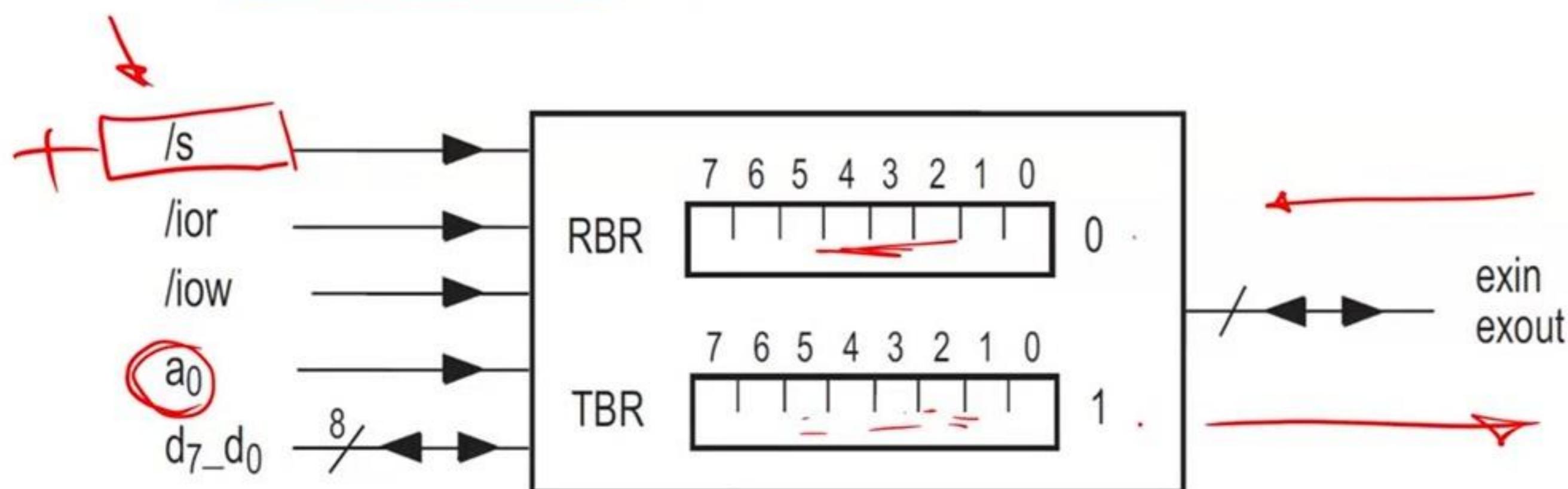
Interfacce

- Descriveremo le varie interfacce che completano il calcolatore
- **Parallele**, che sono in grado di colloquiare con dispositivi ai quali inviano (o dai quali prelevano) un byte alla volta
- Seriali, che colloquiano con dispositivi con i quali scambiano un bit alla volta.
- quelle per la **conversione analogico/digitale e digitale/analogica**, che trasformano gruppi di bit in tensioni e viceversa



Visione funzionale di un'interfaccia

- **Funzionale:** dal punto di vista di chi ci deve interagire (un sistemista, per il montaggio, o un programmatore)
 - Collegamenti e registri



Lettura di RBR
IN offset_RBR, %AL

Scrittura di TBR
OUT %AL, offset_TBR

- Receive Buffer Register, Transmit Buffer Register
 - Interfaccia di ingresso/uscita

Sinronizzazione processore-dispositivi

- Supponiamo che un programma contenga le seguenti istruzioni:

```
IN offset_RBR, %AL •  
...  
IN offset_RBR, %AL • ←
```

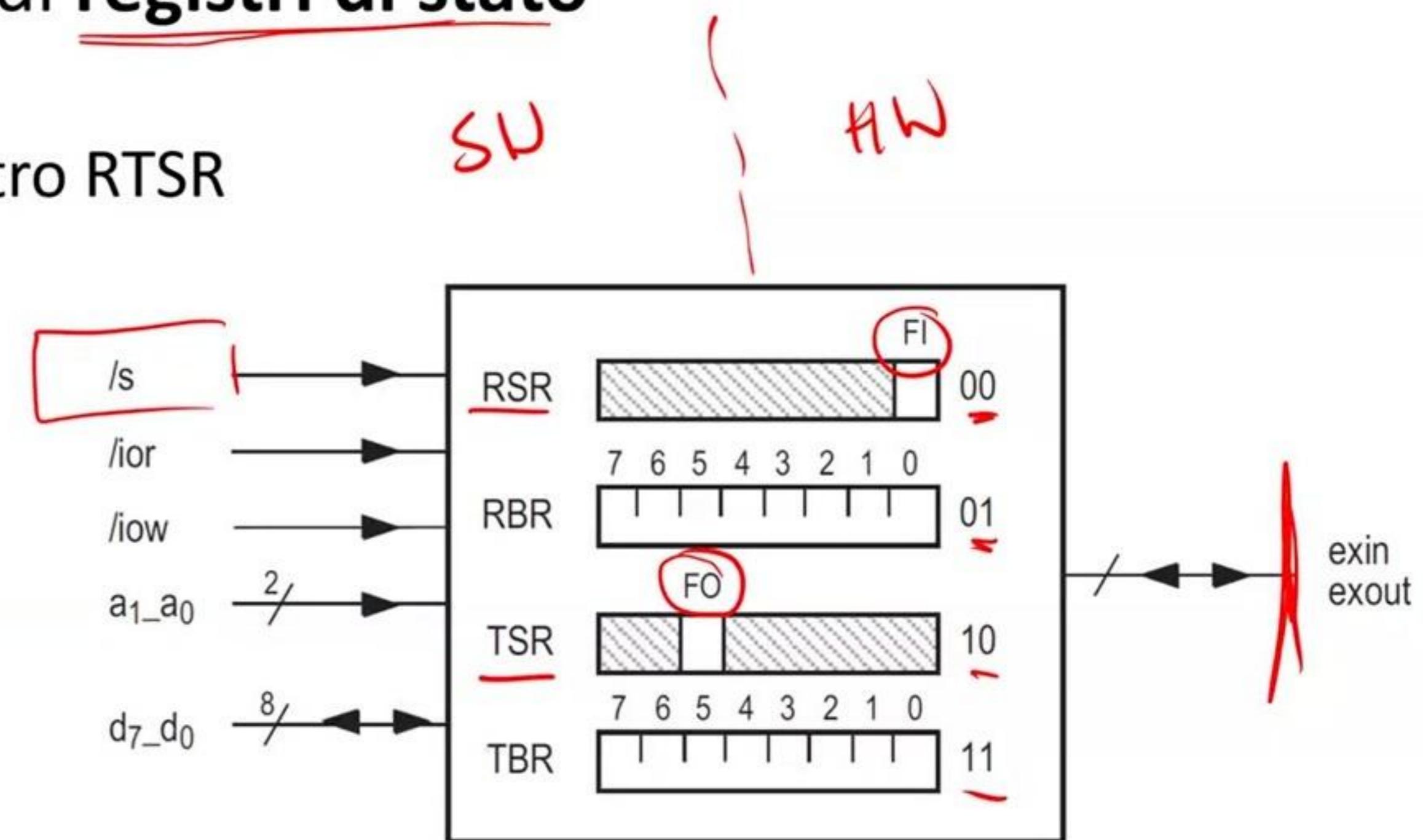
```
• OUT %AL, offset_TBR  
...  
OUT %AL, offset_TBR ←
```

- Il processore non ha modo di sapere se tra le due IN il dispositivo è stato in grado di produrre un dato nuovo
 - la seconda IN potrebbe avere come esito l'ingresso di un dato non significativo.
- Allo stesso modo, il processore non può sapere se tra le due OUT il dispositivo ha processato il primo dato inviato

Sinronizzazione processore-dispositivi

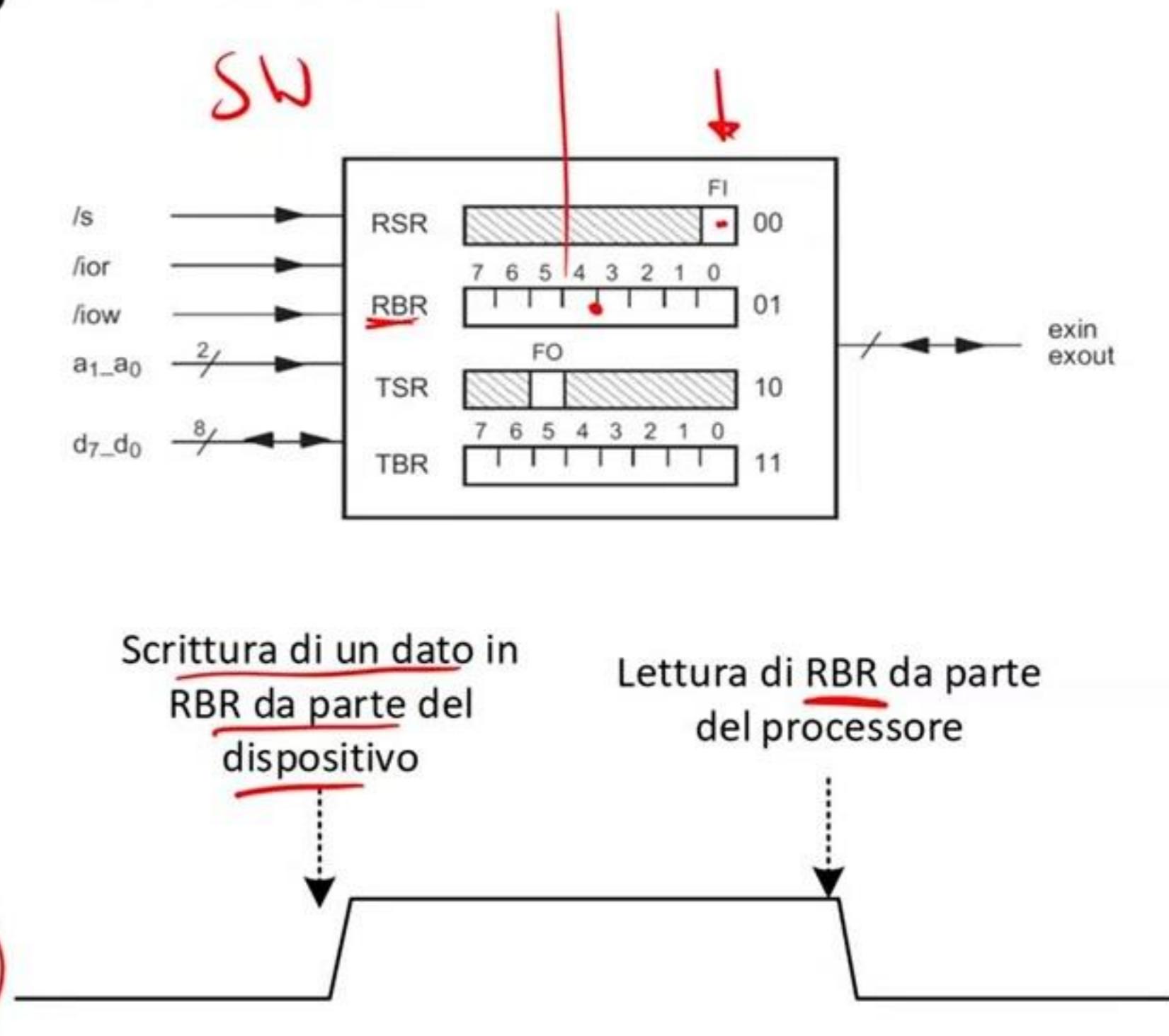
- Ci vuole un handshake tra processore e dispositivi
- Si realizza dotando le interfacce di registri di stato
 - Receive/Transmit Status Register
 - Spesso collassati in un unico registro RTSR
- Contengono **flag**
 - FI: buffer di ingresso pieno
 - FO: buffer di uscita **vuoto**

↓ : dispositivo ha fatto il
suo dovere



Ingresso dati a controllo di programma

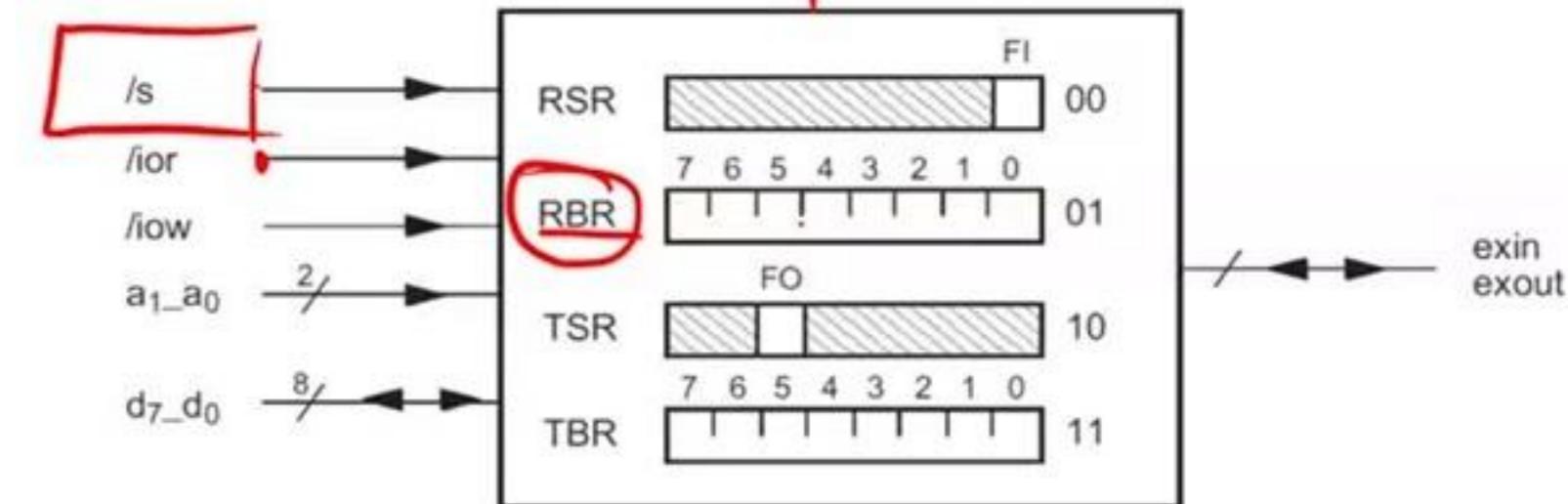
- Il flag FI è inizialmente a 0.
- L'interfaccia lo mette ad 1 quando il dispositivo scrive un dato in RBR
 - per segnalare che il dato in RBR è nuovo.
- Quando il processore, tramite un'istruzione di IN, accede in lettura al registro **RBR**, l'interfaccia porta a 0 il flag FI.



```
testFI:  
    IN RSR_offset,%AL      # Copia in AL il contenuto di RSR  
    AND $0x01,%AL          # Evidenzia in AL il contenuto di FI  
    JZ testFI               # cicla finché FI vale 0  
    IN RBR_offset,%AL      # Copia in AL il contenuto di RBR  
    RET                     # Ritorna al chiamante
```

Uscita dati a controllo di programma

- Il flag FO è inizialmente a 1.
- L'interfaccia lo mette a 0 quando il **processore** scrive un dato in TBR (tramite una OUT)
 - per segnalare che il dispositivo non lo ha ancora processato.
- Quando il dispositivo accede al registro TBR per leggere il dato, l'interfaccia riporta a 1 FO.



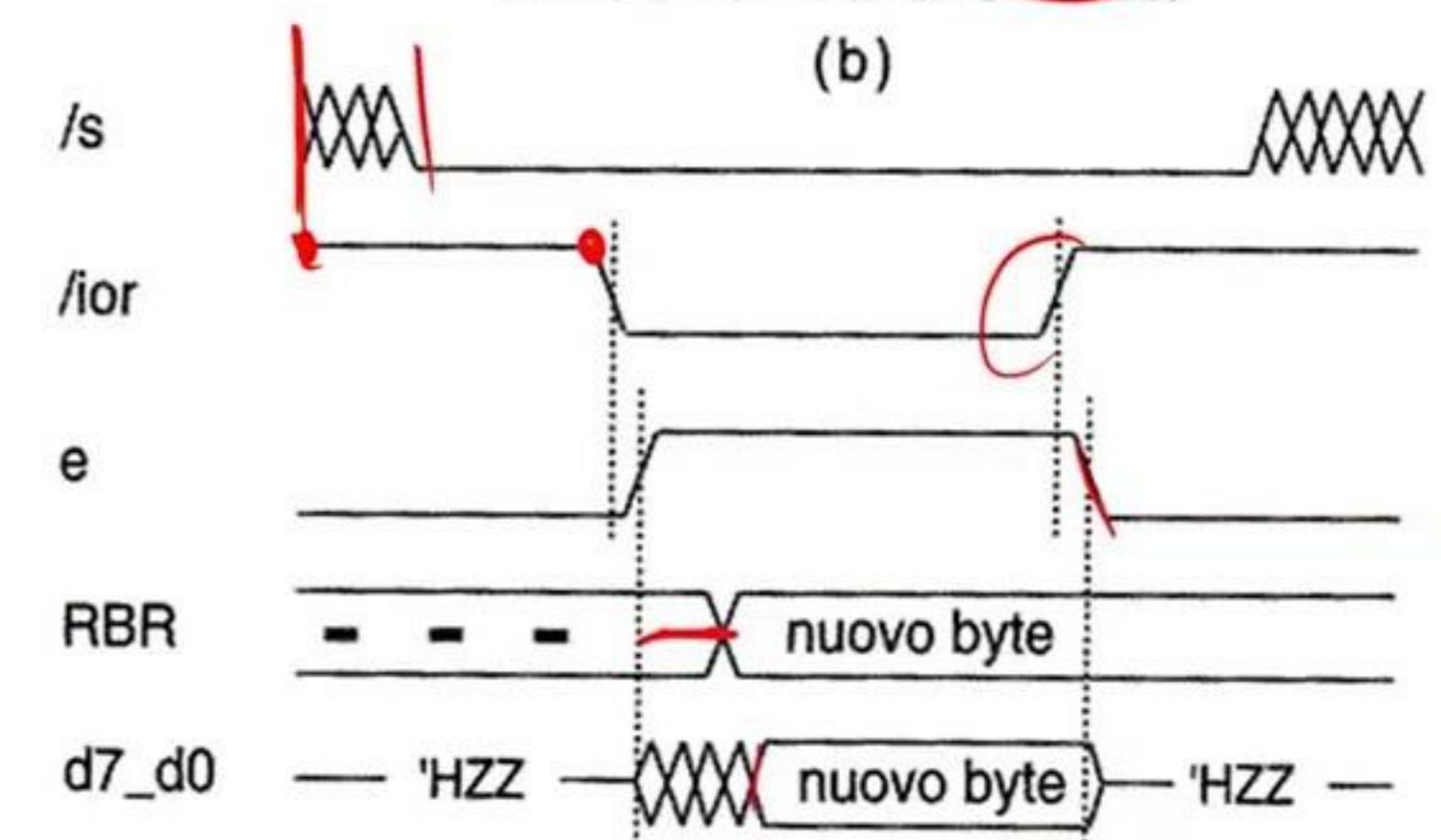
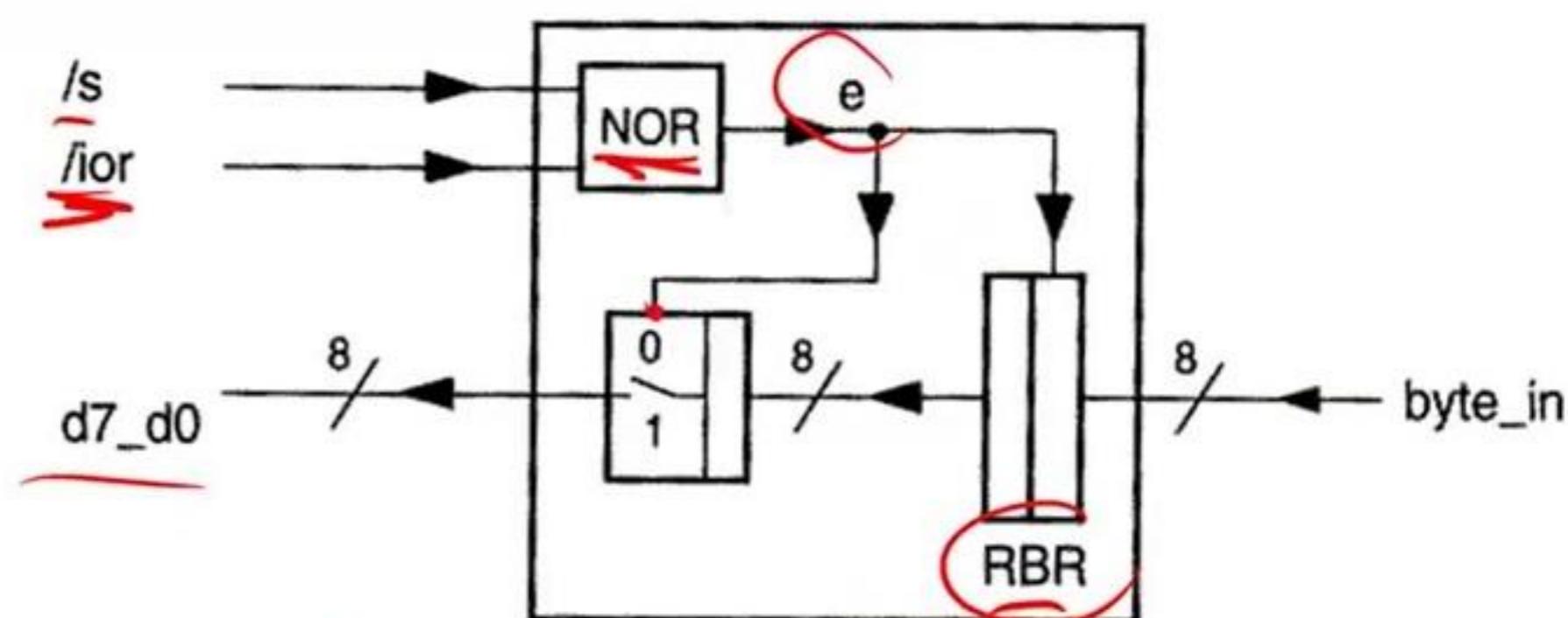
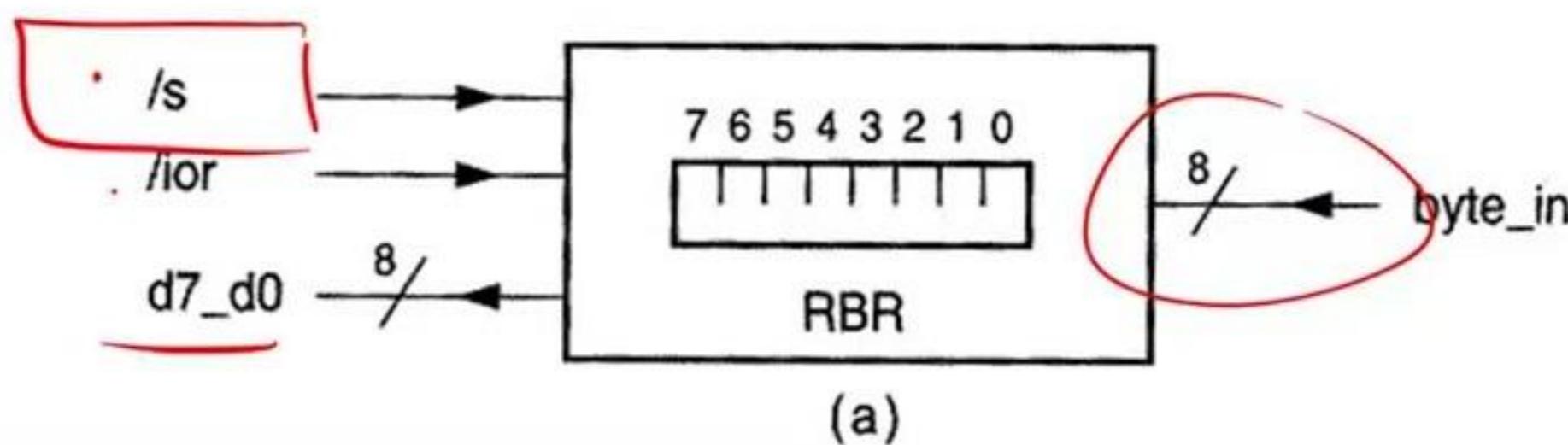
(lettura)
→ A₂₃-A₀ ← - - - ; *IOR-L=φ*

Accesso a controllo di programma

- Prevede che il processore resti in **attesa attiva**
 - cicla (all'interno del sottoprogramma) in attesa che il dispositivo esterno sia pronto.
- Particolarmente inefficiente
 - fa perdere tempo inutilmente al processore.
- Sarebbe molto meglio se
 - Il processore potesse **andare avanti** per conto proprio
 - Le interfacce segnalassero al processore quando sono pronte
 - “interrompendo” il lavoro del processore.
- **Accesso ad interruzione di programma,**
 - tecnica che vedrete durante il corso di Calcolatori Elettronici
- **Direct memory access (DMA)**
 - Il processore demanda ad un'altra unità (detta *DMA controller*) il compito di trasferire dati tra la memoria e le interfacce, mentre va avanti con le sue elaborazioni

Interfacce parallele

- Cominciamo da quelle senza handshake
- **Interfaccia parallela di ingresso**
- Visione funzionale:
 - un solo registro
 - Nessun filo di indirizzo (vanno tutti alla maschera)
- Struttura interna
 - Quando **/s ed /ior** vanno a zero, RBR **campiona** il dato del dispositivo
 - Le tri-state vanno in conduzione, e dopo poco il dato appare sul bus
 - **/ior** va giù **dopo** gli indirizzi

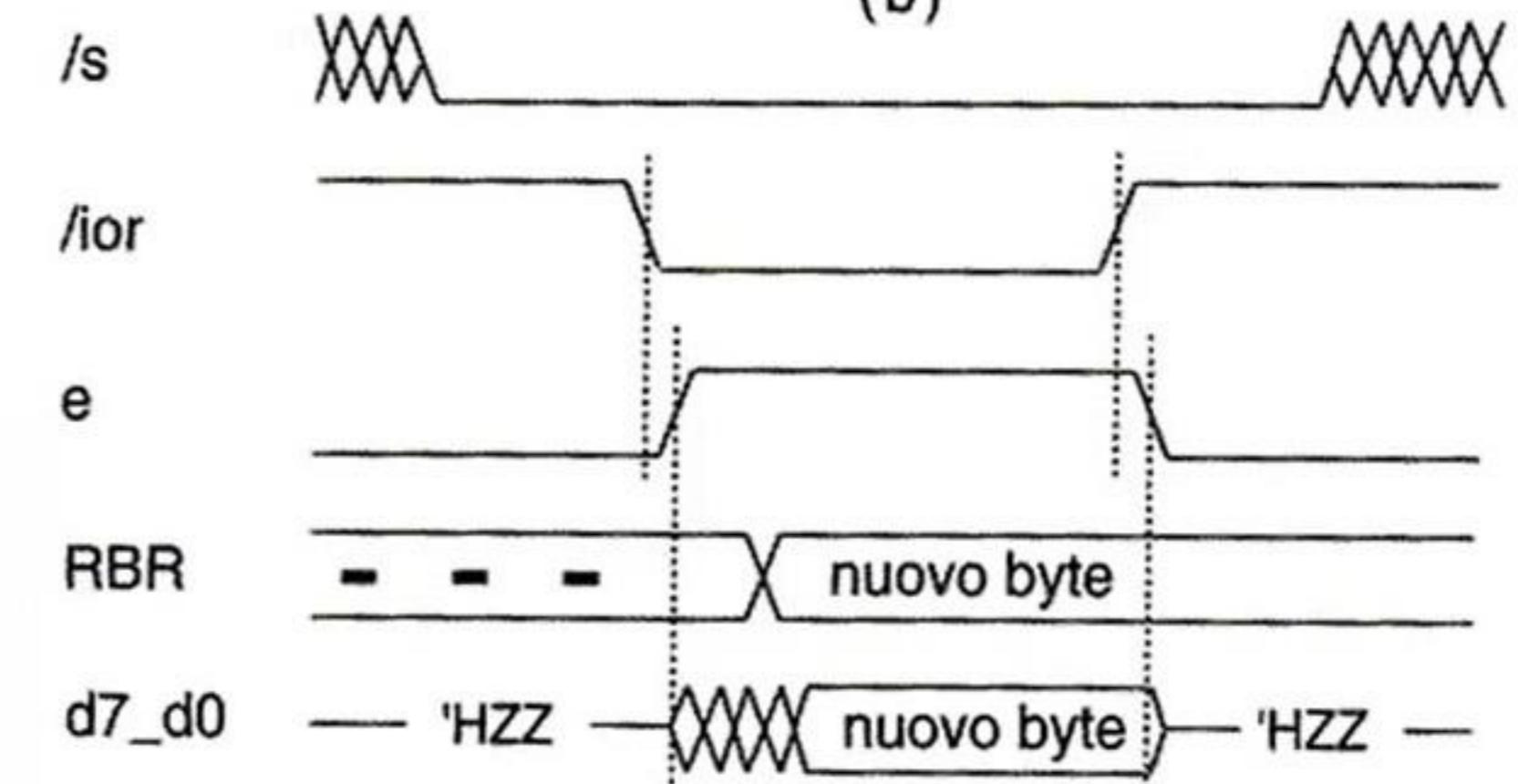
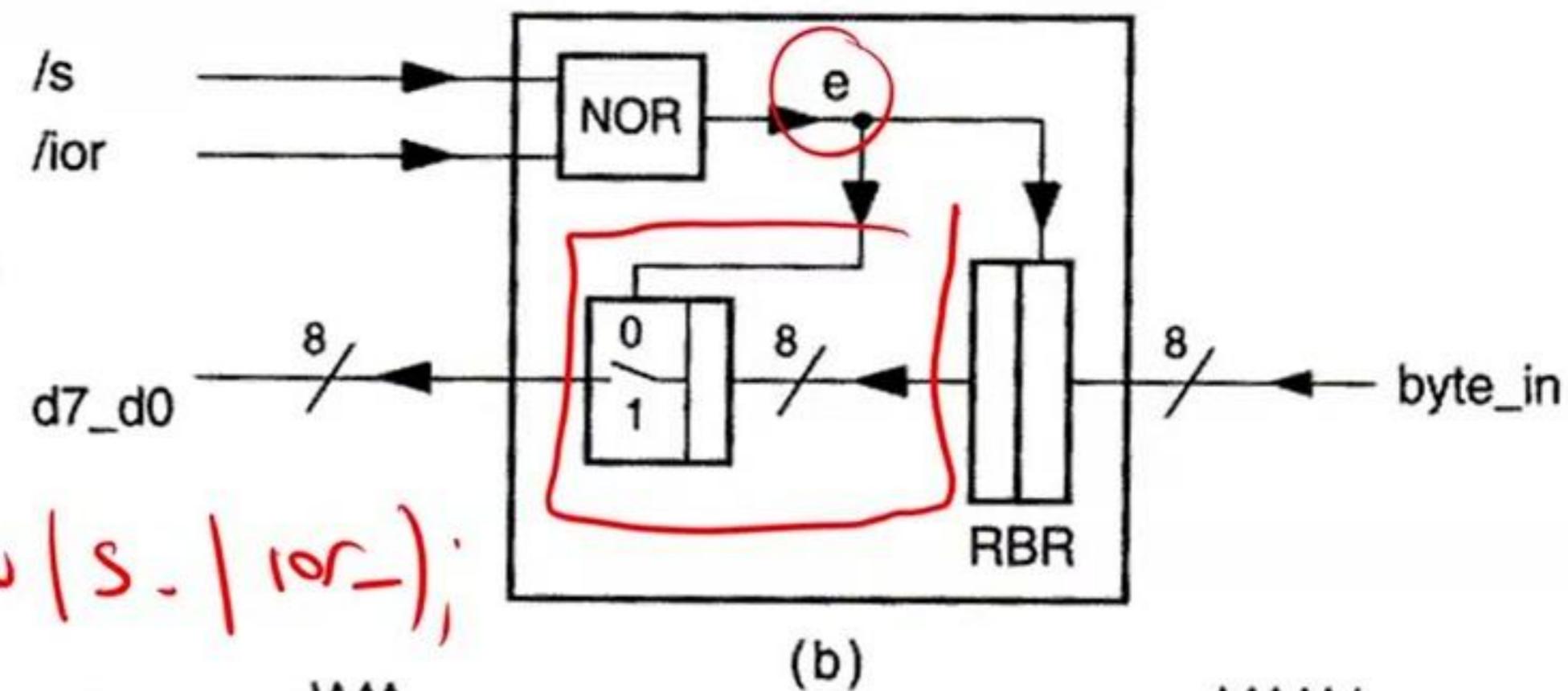
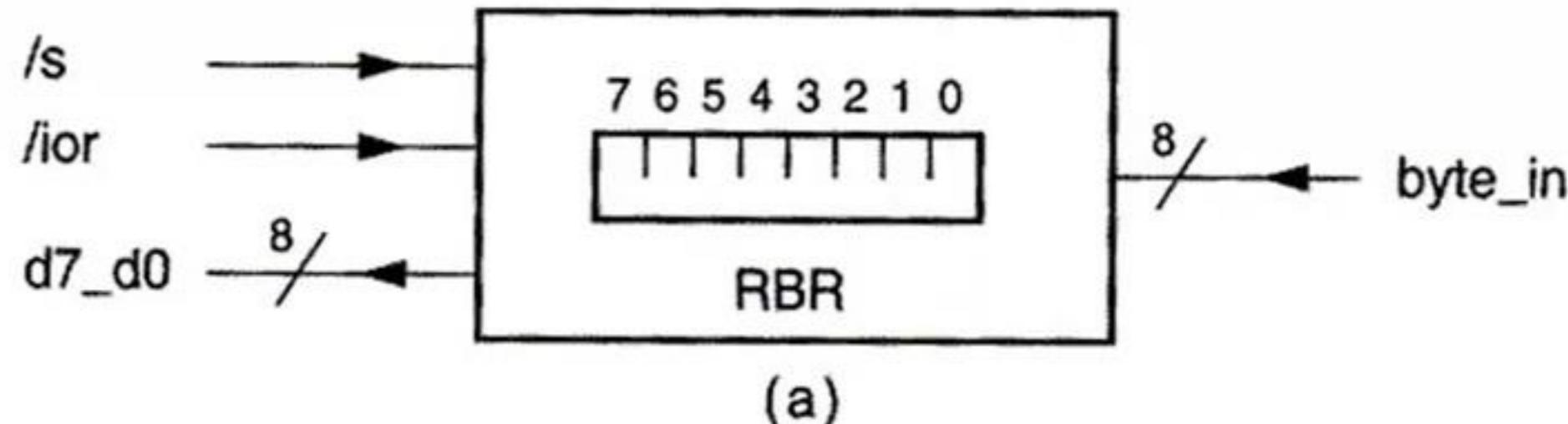


Interfacce parallele (cont.)

```

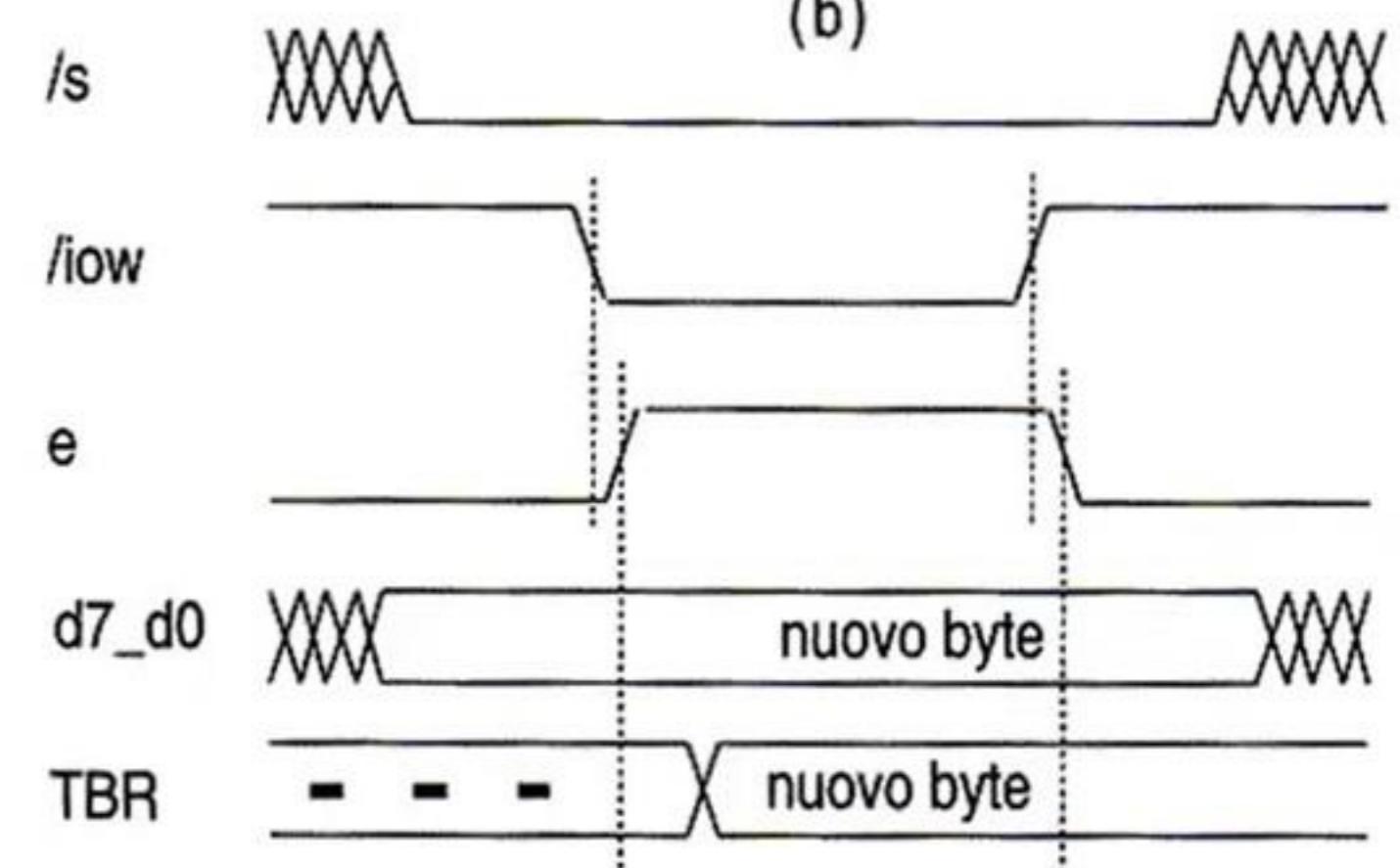
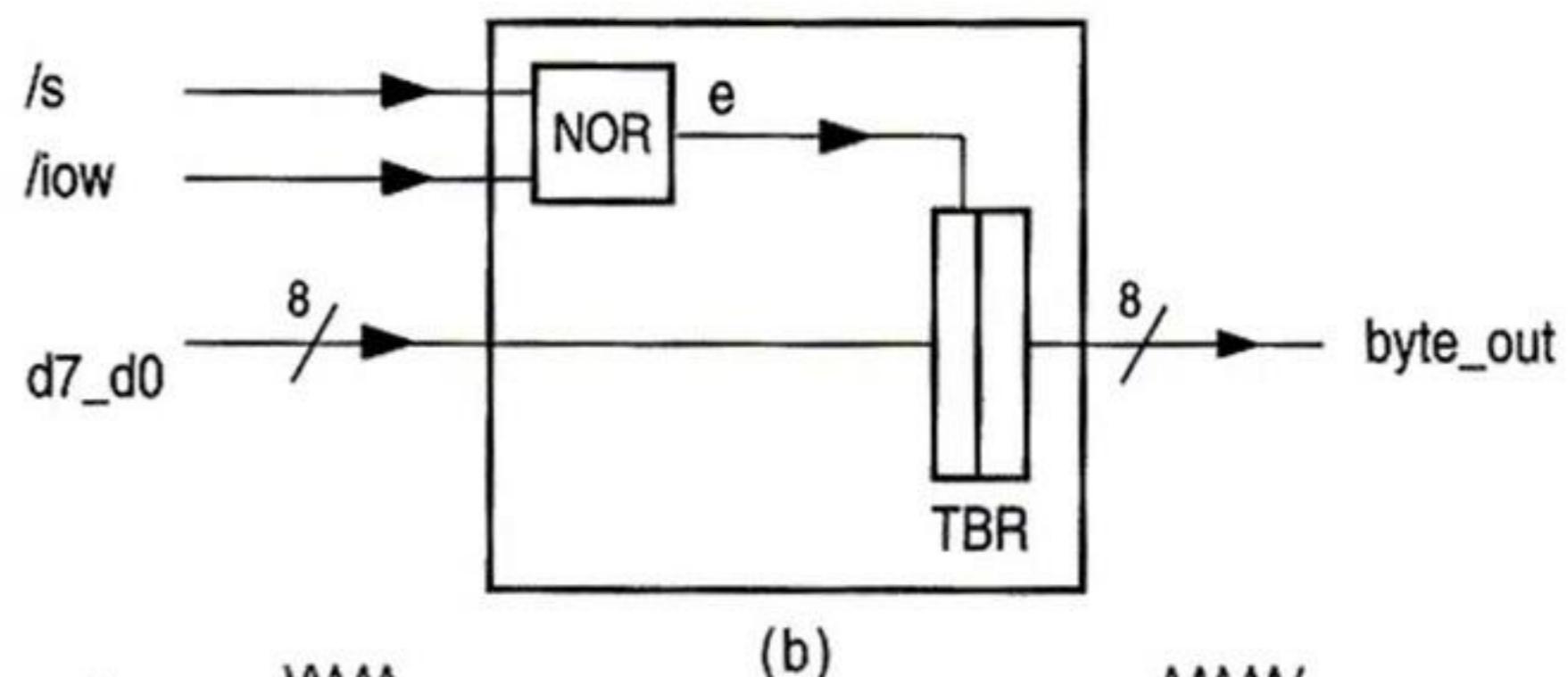
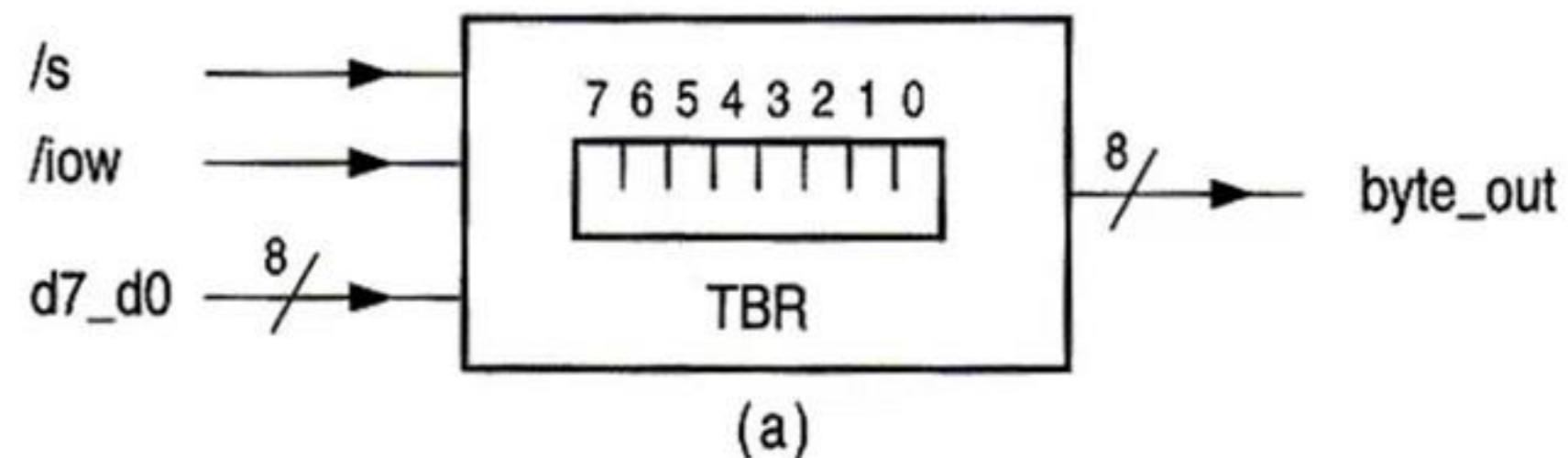
module Interfaccia_Parallela_di_Ingresso
    (d7_d0, s_, ior_, byte_in);
    input s_, ior_;
    output[7:0] d7_d0;
    input[7:0] byte_in;
    reg[7:0] RBR;
    wire e; assign e=({s_, ior_}=='B00)?1:0;
    [assign d7_d0=(e==1)?RBR:'HZZ;
    always @(posedge e) #3 RBR<=byte_in;
endmodule

```



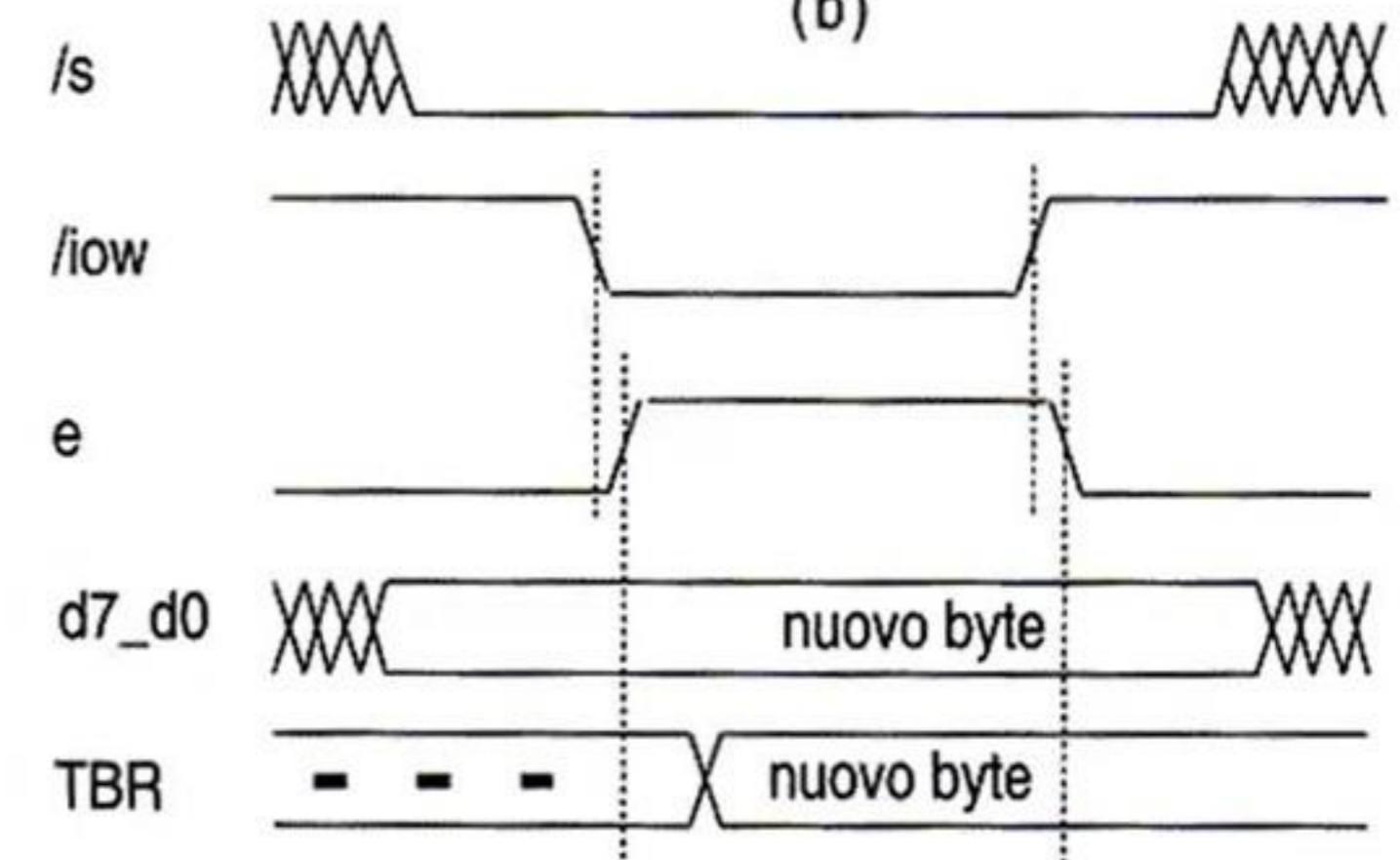
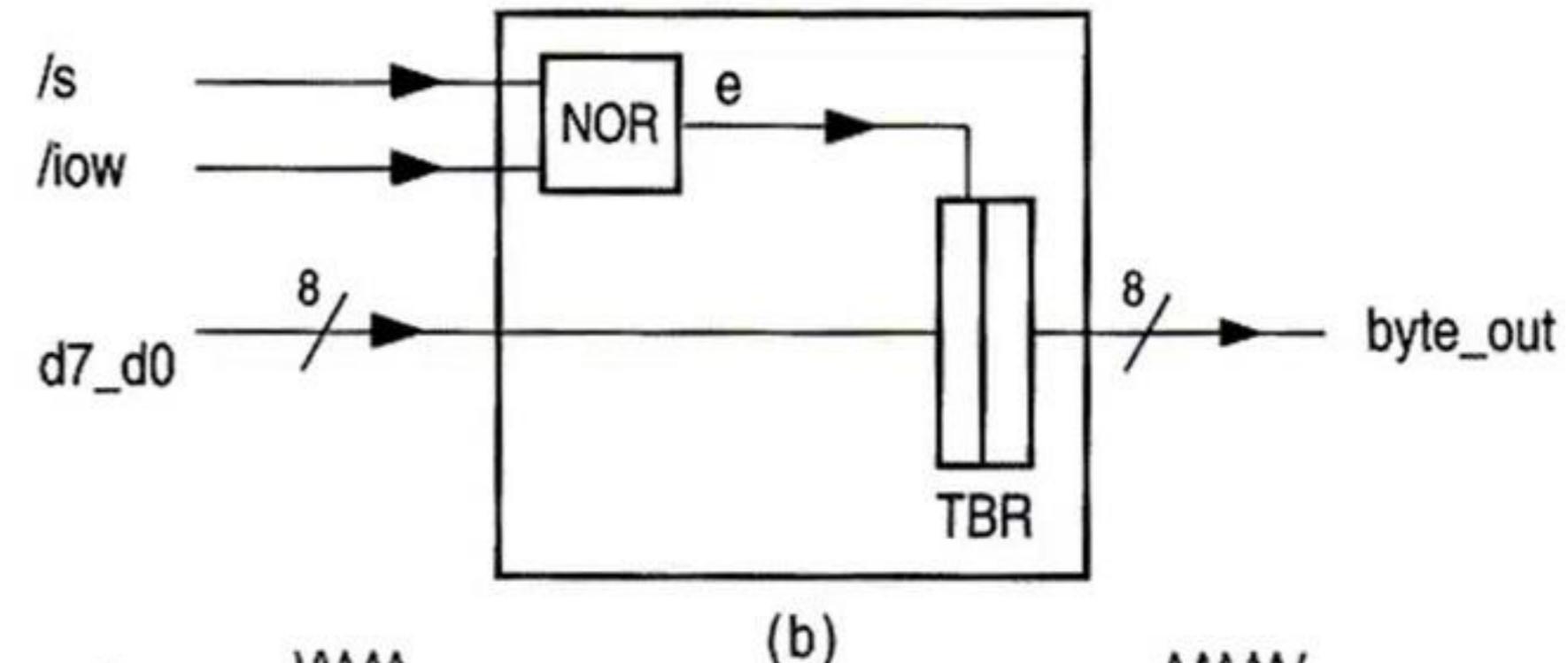
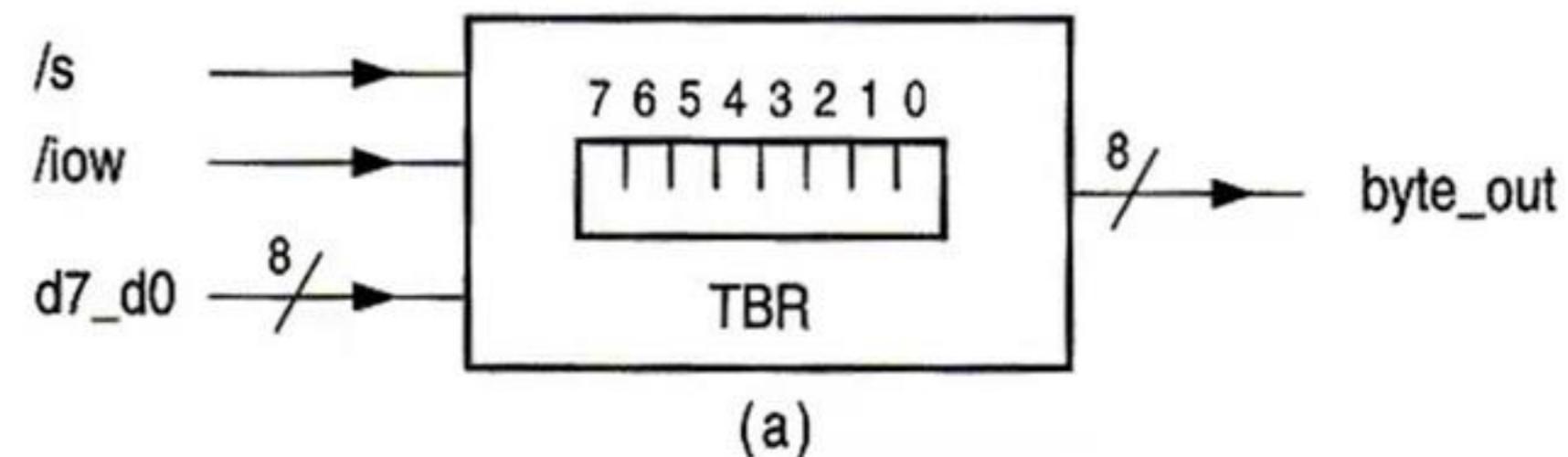
Interfacce parallele (cont.)

- Interfaccia **parallela di uscita**
- Visione funzionale:
 - un solo registro
 - Nessun filo di indirizzo (vanno tutti alla maschera)
- Struttura interna
 - Quando **/s ed /iow vanno a zero**, TBR **campiona il dato dal bus**
 - TBR campiona sul fronte di **discesa** di /iow
 - **/iow deve andare giù dopo gli indirizzi**



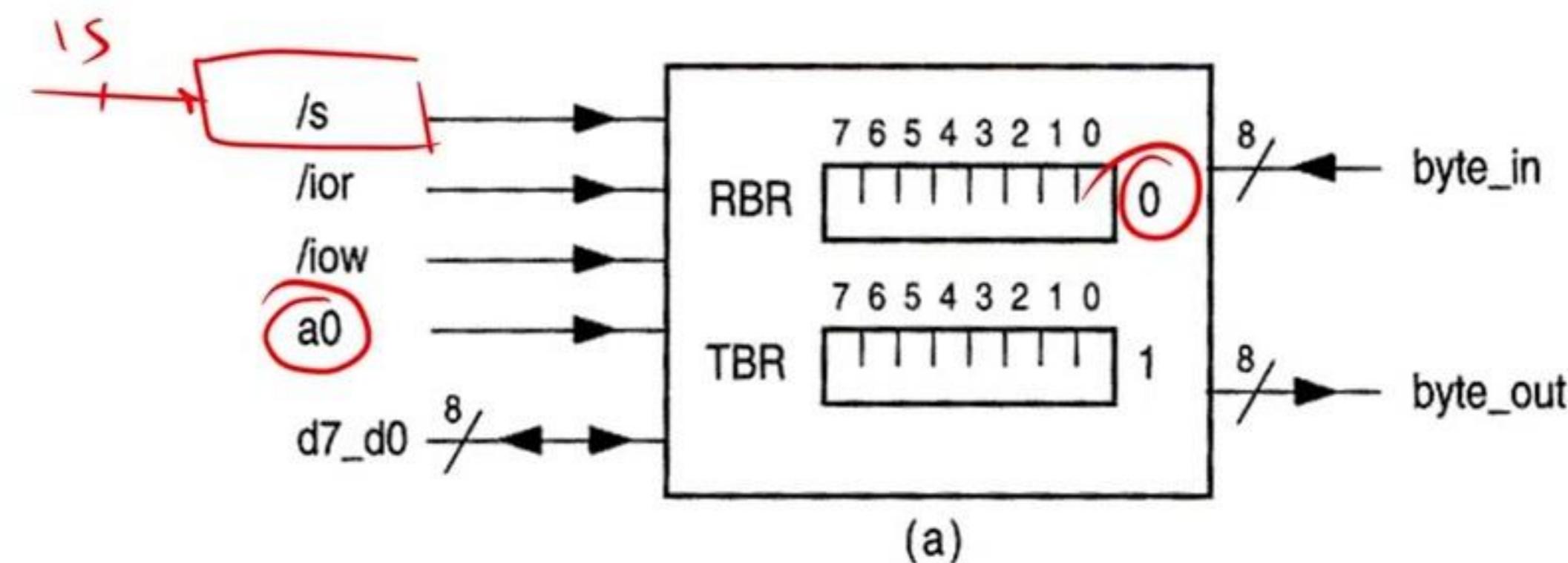
Interfacce parallele (cont.)

```
module Interfaccia_Parallela_di_Uscita
    (d7_d0,s_,iow_,byte_out);
    input s_,iow_;
    input[7:0] d7_d0;
    output[7:0] byte_out;
    reg[7:0] TBR; assign byte_out=TBR;
    wire e; assign e=({s_,iow_}=='B00)?1:0;
    always @ (posedge e) #3 TBR<=d7_d0;
endmodule
```

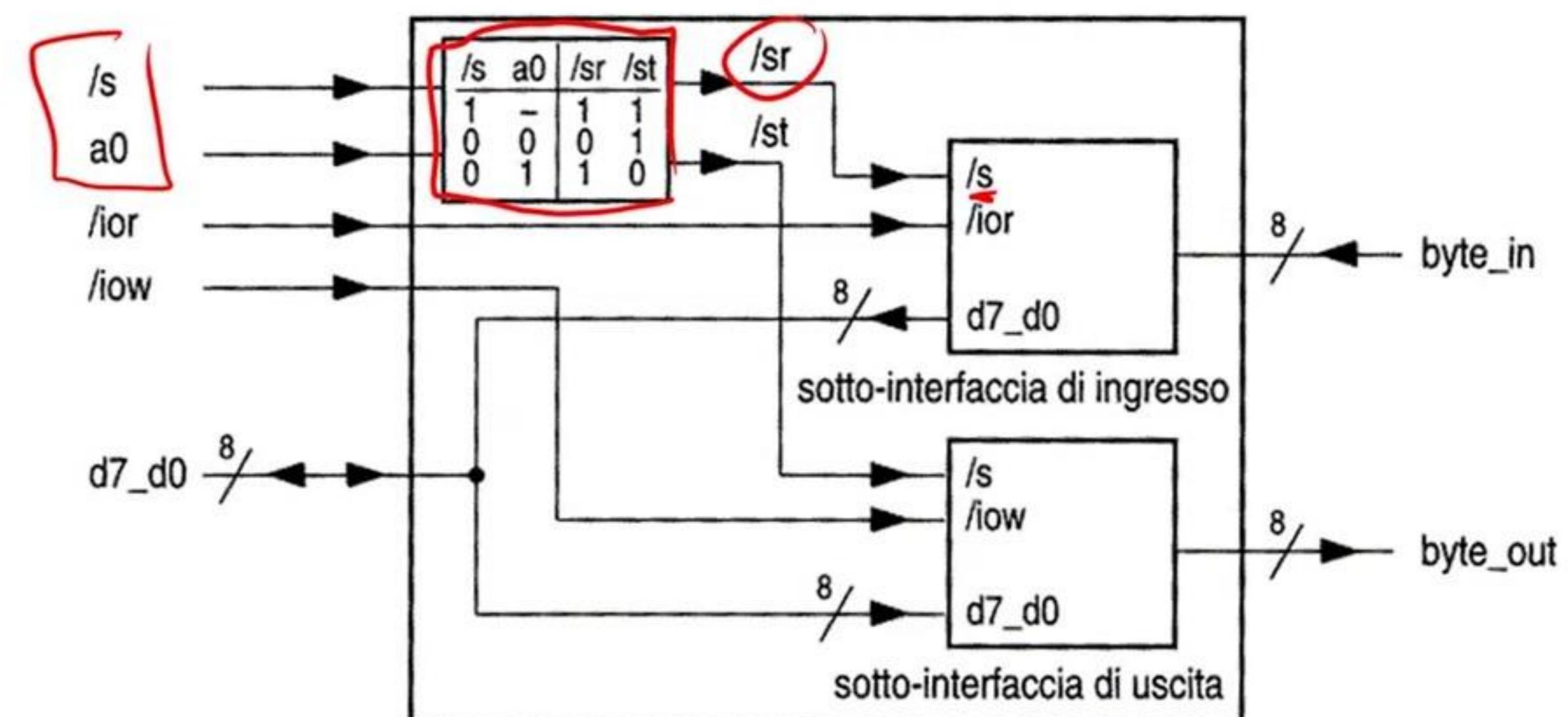


Interfacce parallele (cont.)

- Porta di indirizzo **pari**: ingresso
- Porta di indirizzo **dispari**: uscita
- Logica combinatoria (minima) per produrre i due select

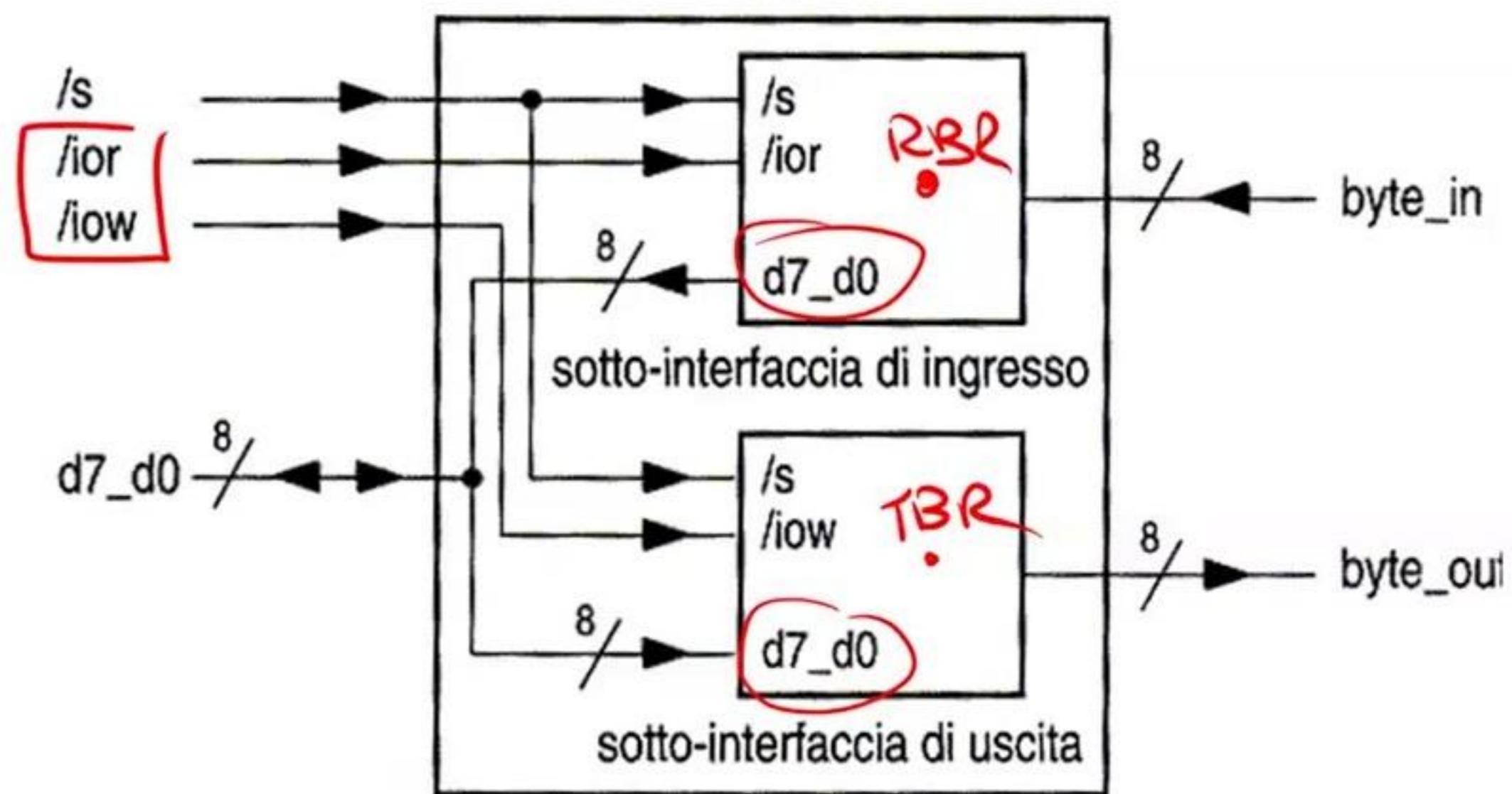
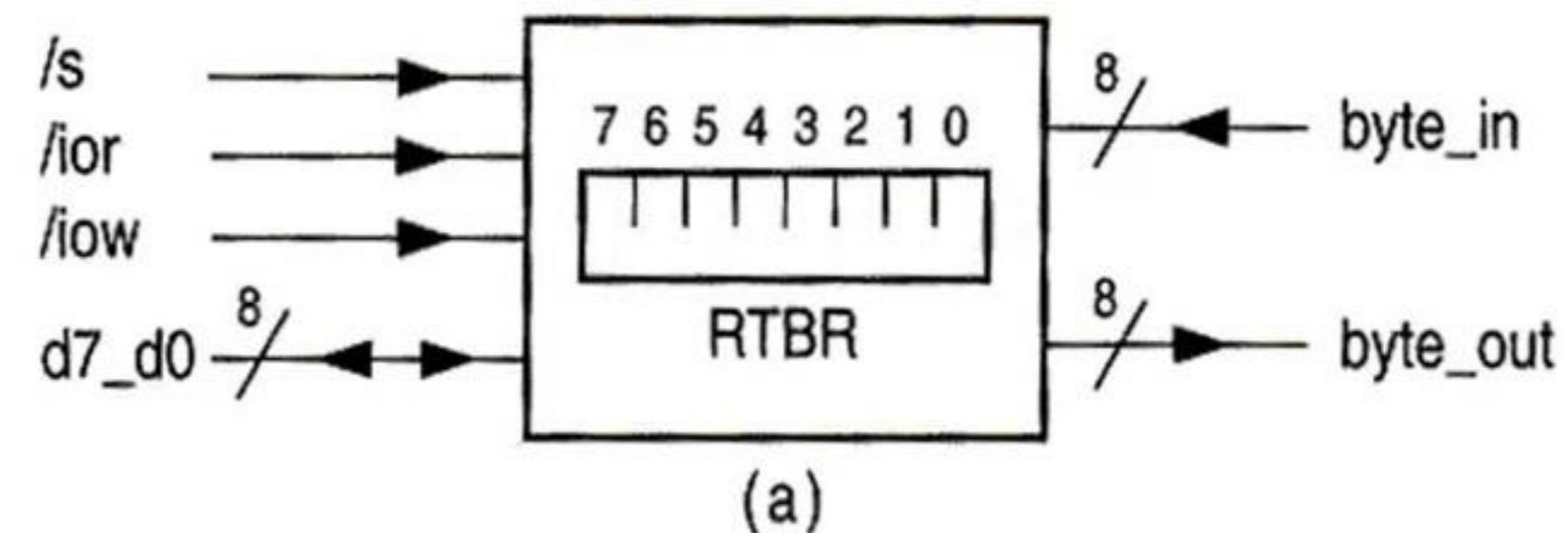


(a)



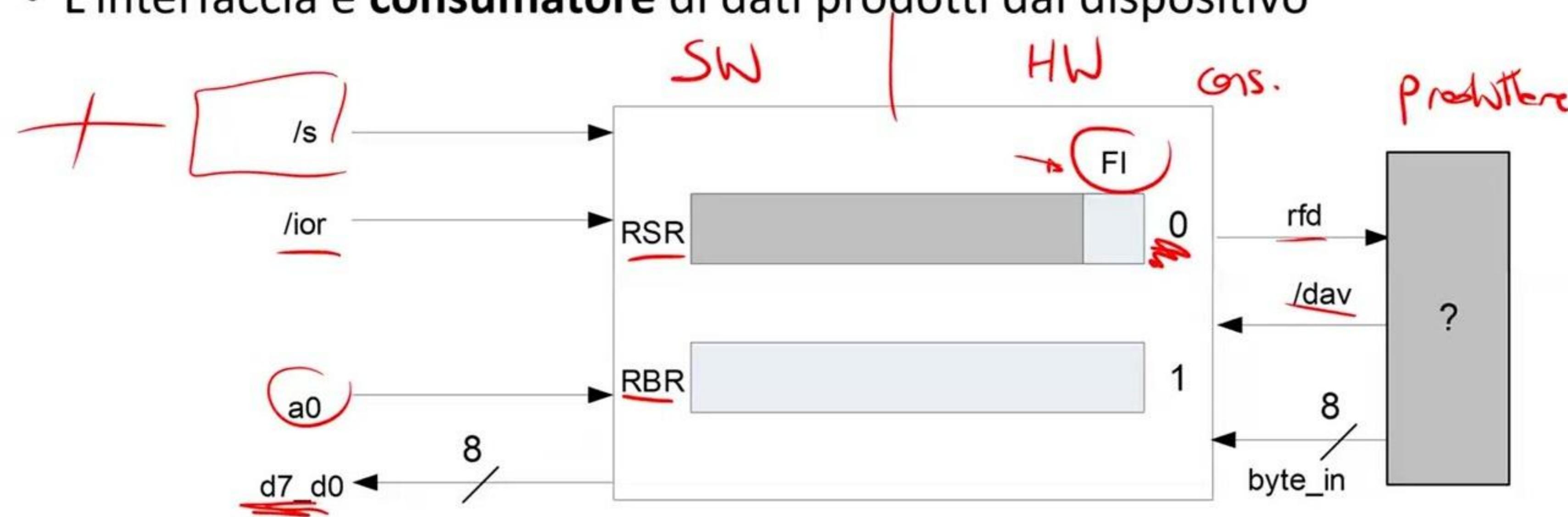
Interfacce parallele (cont.)

- Montaggio alternativo:
 - Le due porte sono allo **stesso offset**
 - Il programmatore le riferisce con lo stesso indirizzo nello spazio di I/O
 - La distinzione avviene in base al tipo di accesso



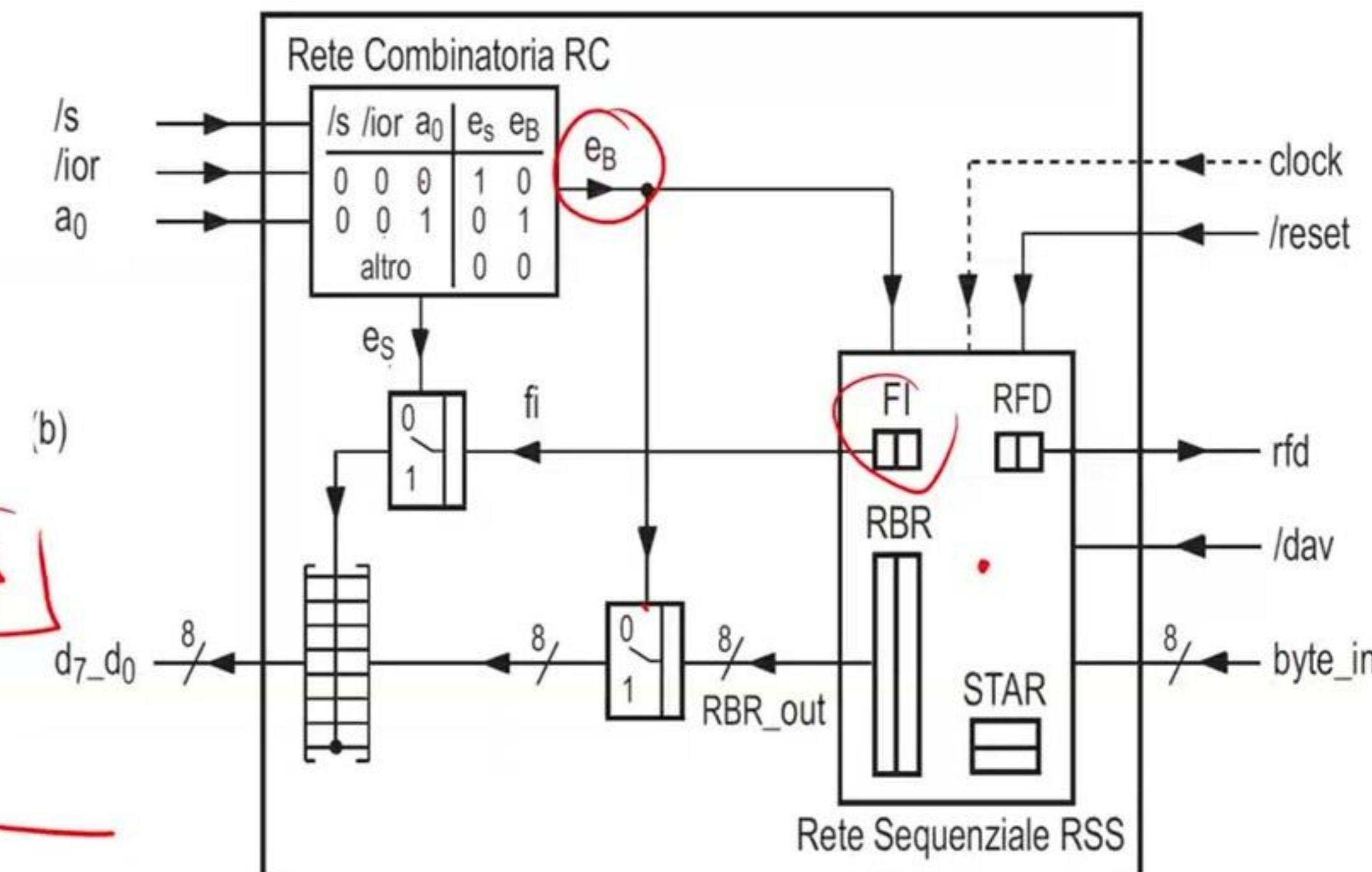
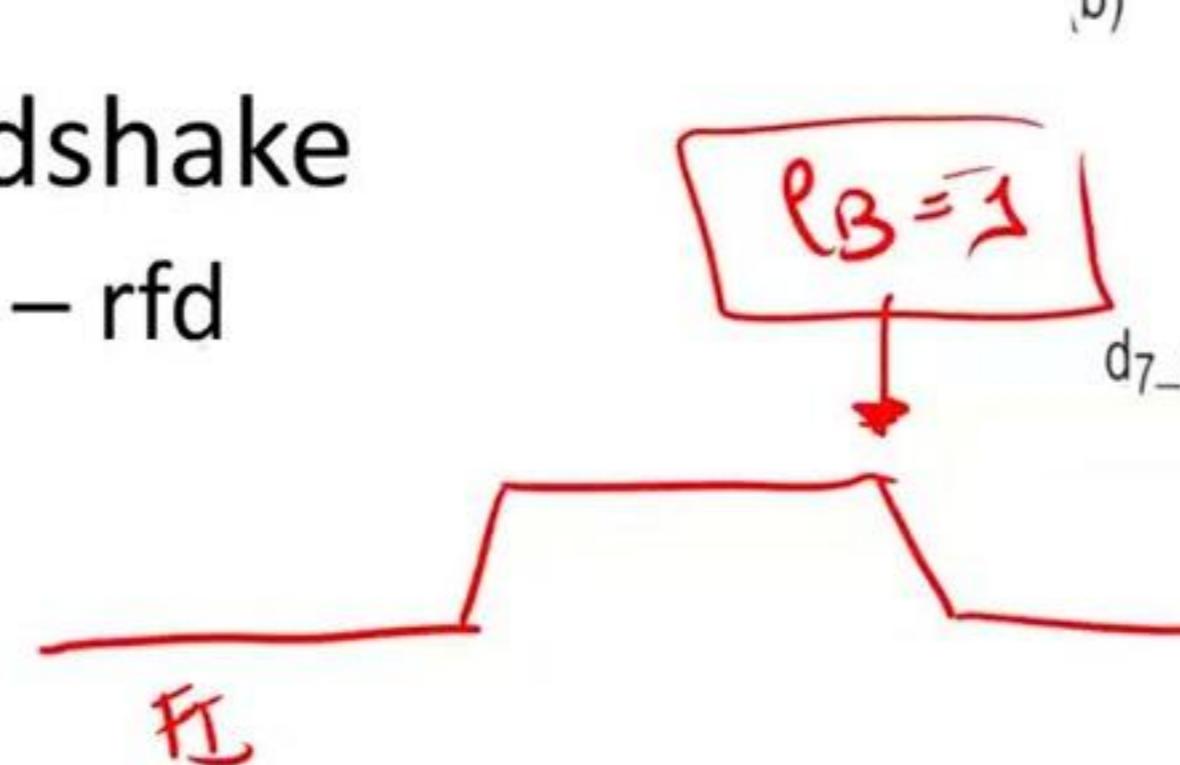
Interfaccia parallela di ingresso con handshake

- Visione **funzionale**
- Lato processore: **RSR** con flag **FI**
 - Ci sono **due registri**, ci vuole un filo di indirizzo per distinguerli
- Lato dispositivo: fili **/dav** ed **rfd**
 - L'interfaccia è **consumatore** di dati prodotti dal dispositivo

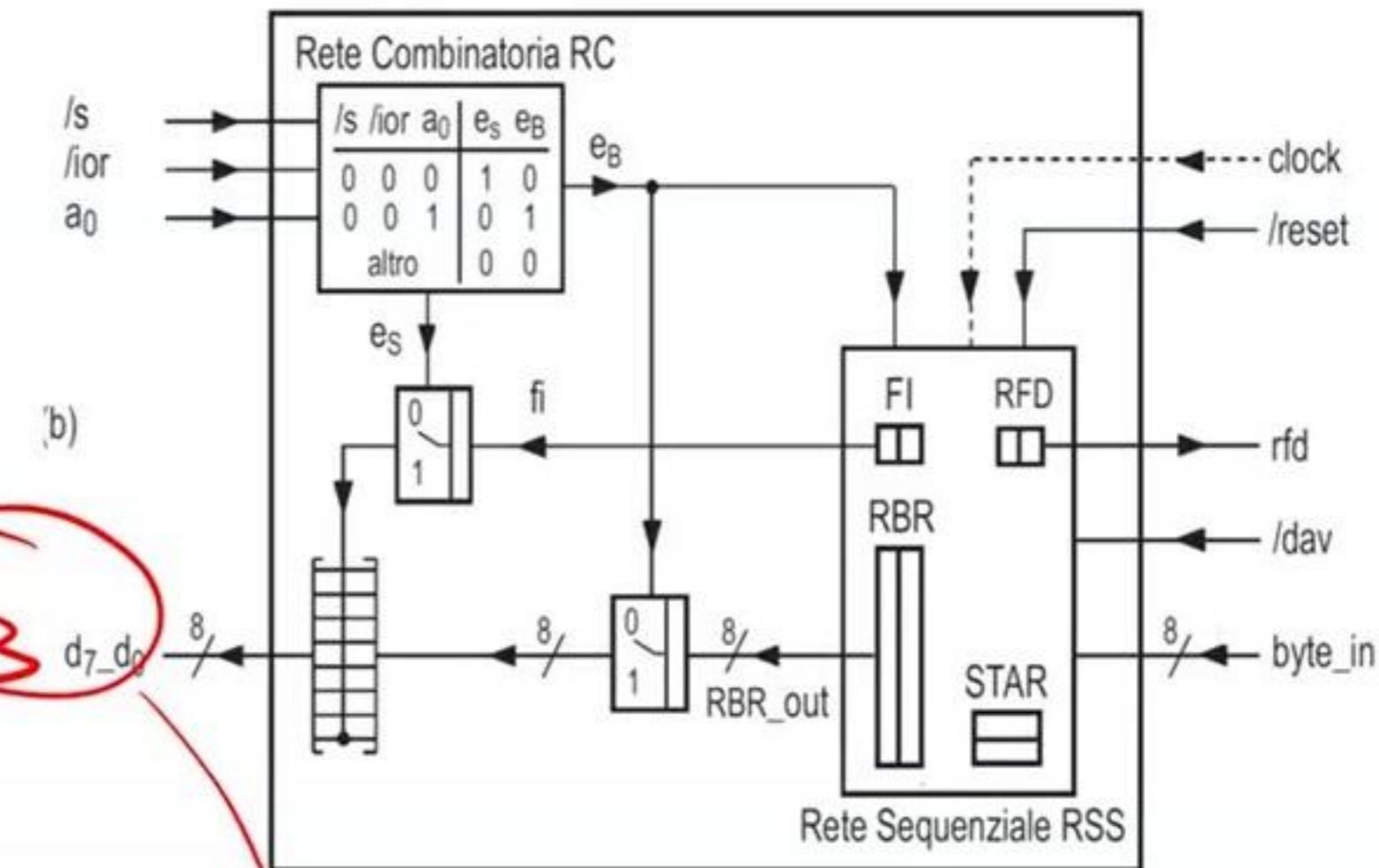
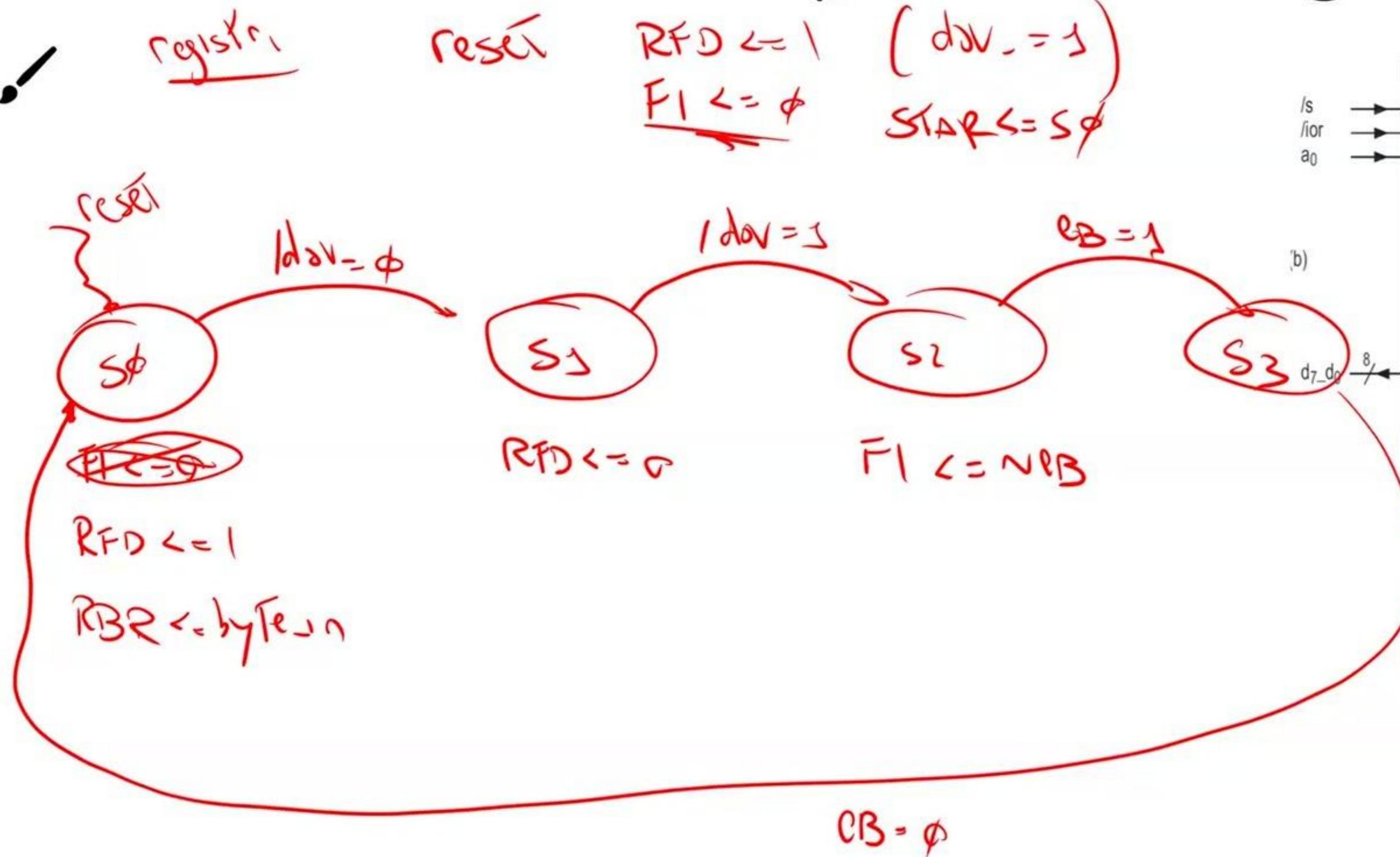


Interfaccia parallela di ingresso con handshake

- **Struttura interna**
- RC genera i segnali di abilitazione per le tri-state quando il processore accede in lettura a RBR o RSR
 - Le 2 tri-state non sono mai abilitate contemporaneamente
- La RSS gestisce gli handshake
 - con il dispositivo: /dav – rfd
 - Con il processore: FI



RSS dell'interfaccia parallela di ingresso con HS



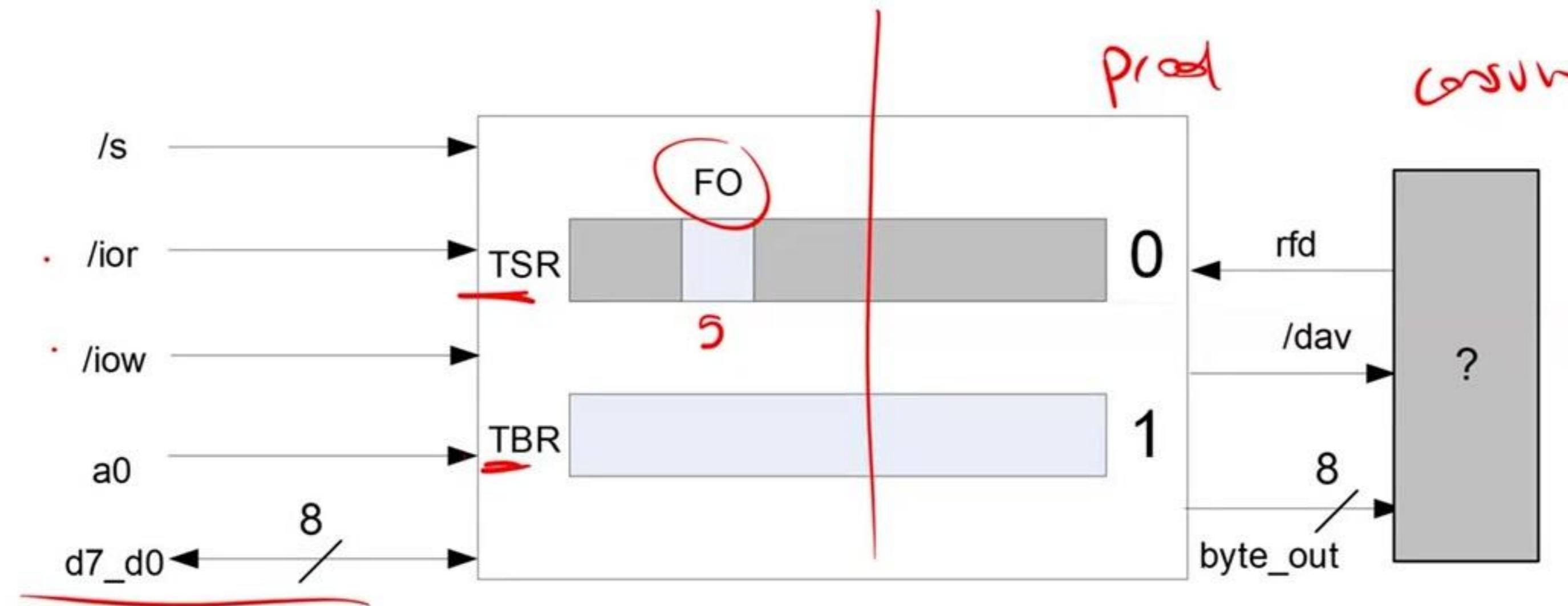
Interfaccia parallela di ingresso con handshake

```
module RSS(dav_, rfd, byte_in, fi, RBR_out, eB, clock, reset_);  
    input clock, reset_;  
    wire clock_RSS; assign #5 clock_RSS=clock;  
    input      dav_, eB;  
    output     rfd, fi;  
    input[7:0]  byte_in;  
    output[7:0] RBR_out;  
  
    reg   RFD;    assign rfd=RFD;  
    reg   FI;     assign fi=FI;  
    reg[7:0] RBR;  assign RBR_out=RBR;  
    reg[1:0] STAR; parameter S0=0, S1=1, S2=2, S3=3;  
  
    always @ (reset_==0) #1 begin RFD<=1; FI<=0; STAR<=S0; end  
    always @ (posedge clock_RSS) if (reset_==1) #3  
        casex (STAR)  
            S0: begin RFD<=1; RBR<=byte_in; STAR<=(dav_==1)?S0:S1; end  
            S1: begin RFD<=0; STAR<=(dav_==0)?S1:S2; end  
            S2: begin FI<=(eB==0)?1:0; STAR<=(eB==0)?S2:S3; end  
            S3: begin STAR<=(eB==1)?S3:S0; end  
        endcase  
    endmodule
```

Pagina 20 non esiste

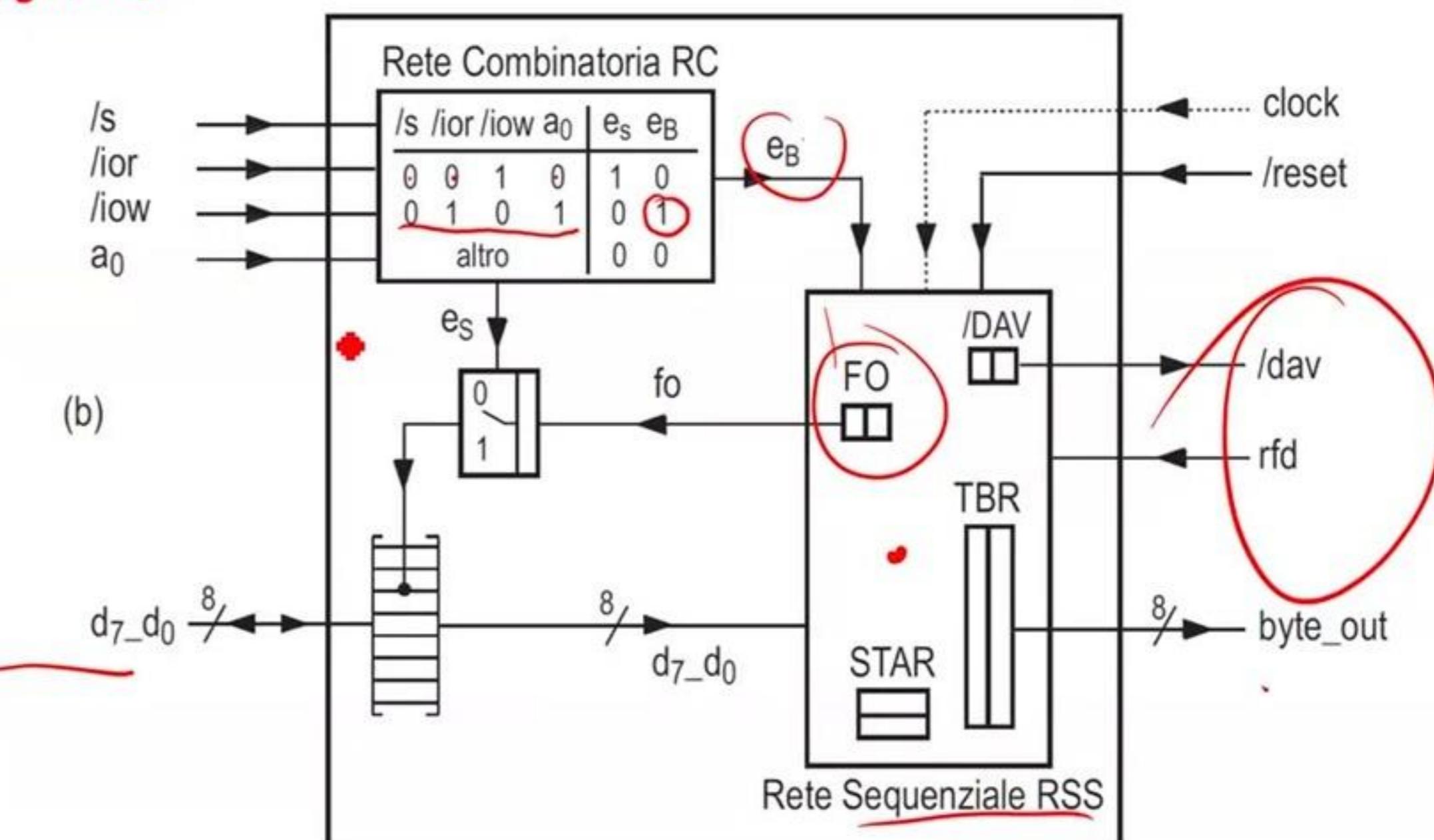
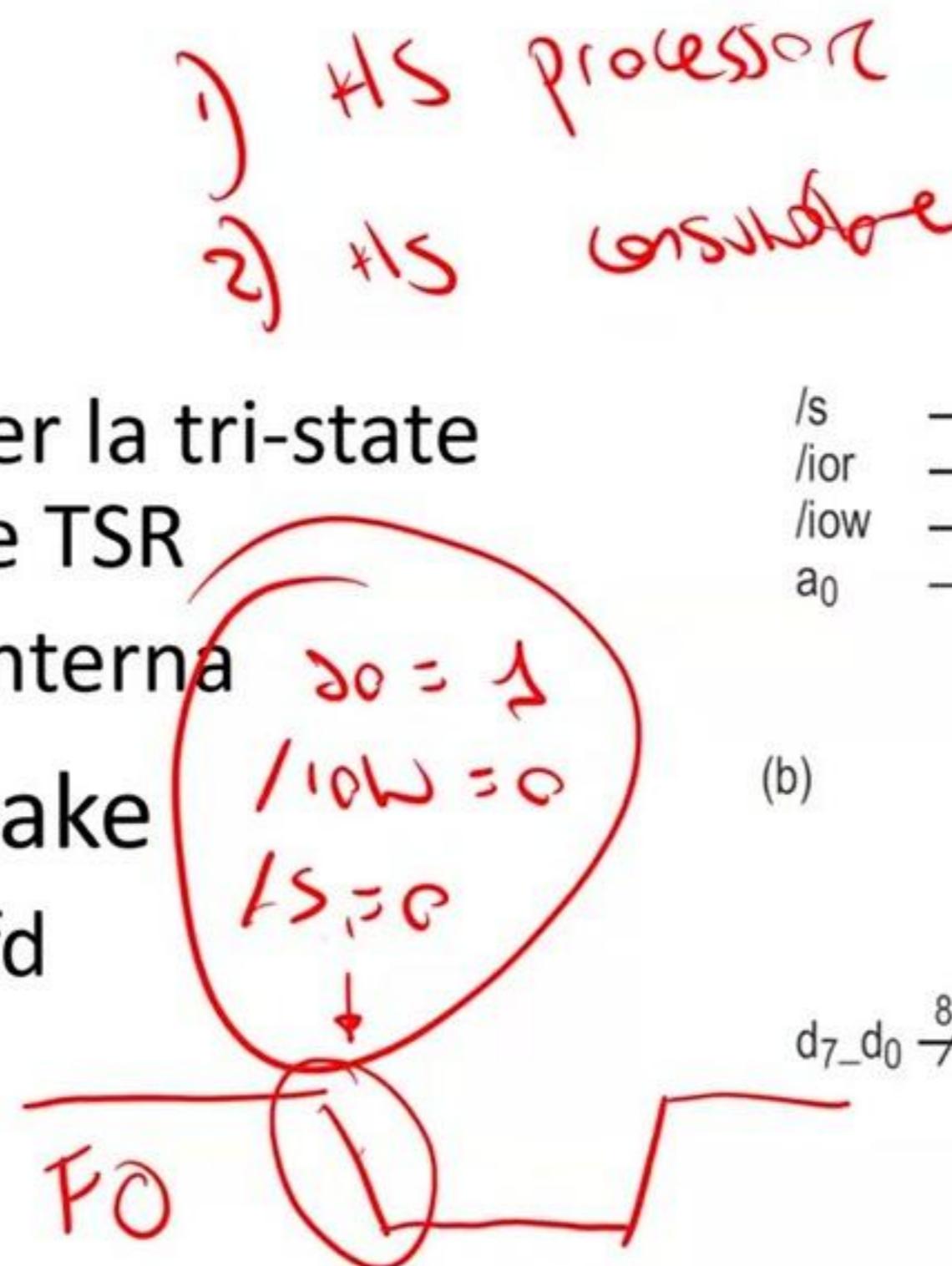
Interfaccia parallela di uscita con handshake

- Visione **funzionale**
- Lato processore: **TSR** con flag **FO**
 - Ci sono **due registri**, ci vuole un filo di indirizzo per distinguerli
- Lato dispositivo: fili **/dav** ed **rfd**
 - L'interfaccia è **produttore** di dati consumati dal dispositivo



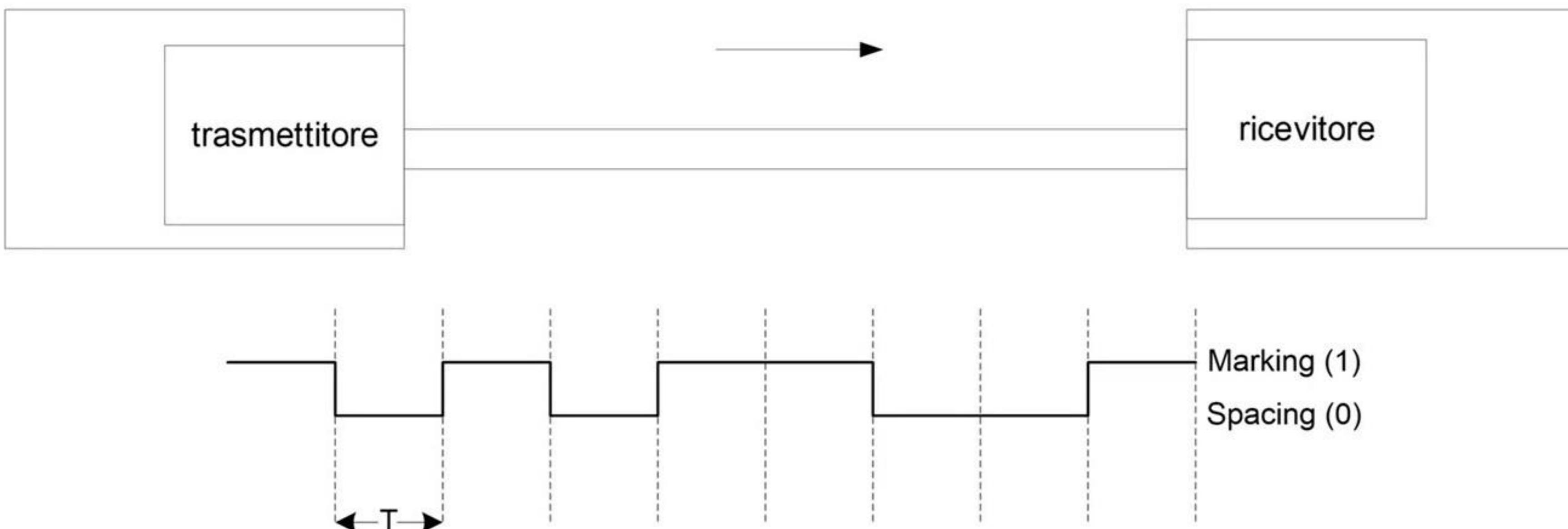
Interfaccia parallela di uscita con handshake

- **Struttura interna**
- RC genera
 - Il segnale di abilitazione per la tri-state quando il processore legge TSR
 - Un segnale eB per la RSS interna
- La RSS gestisce gli handshake
 - con il dispositivo: /dav – rfd
 - Con il processore: FO



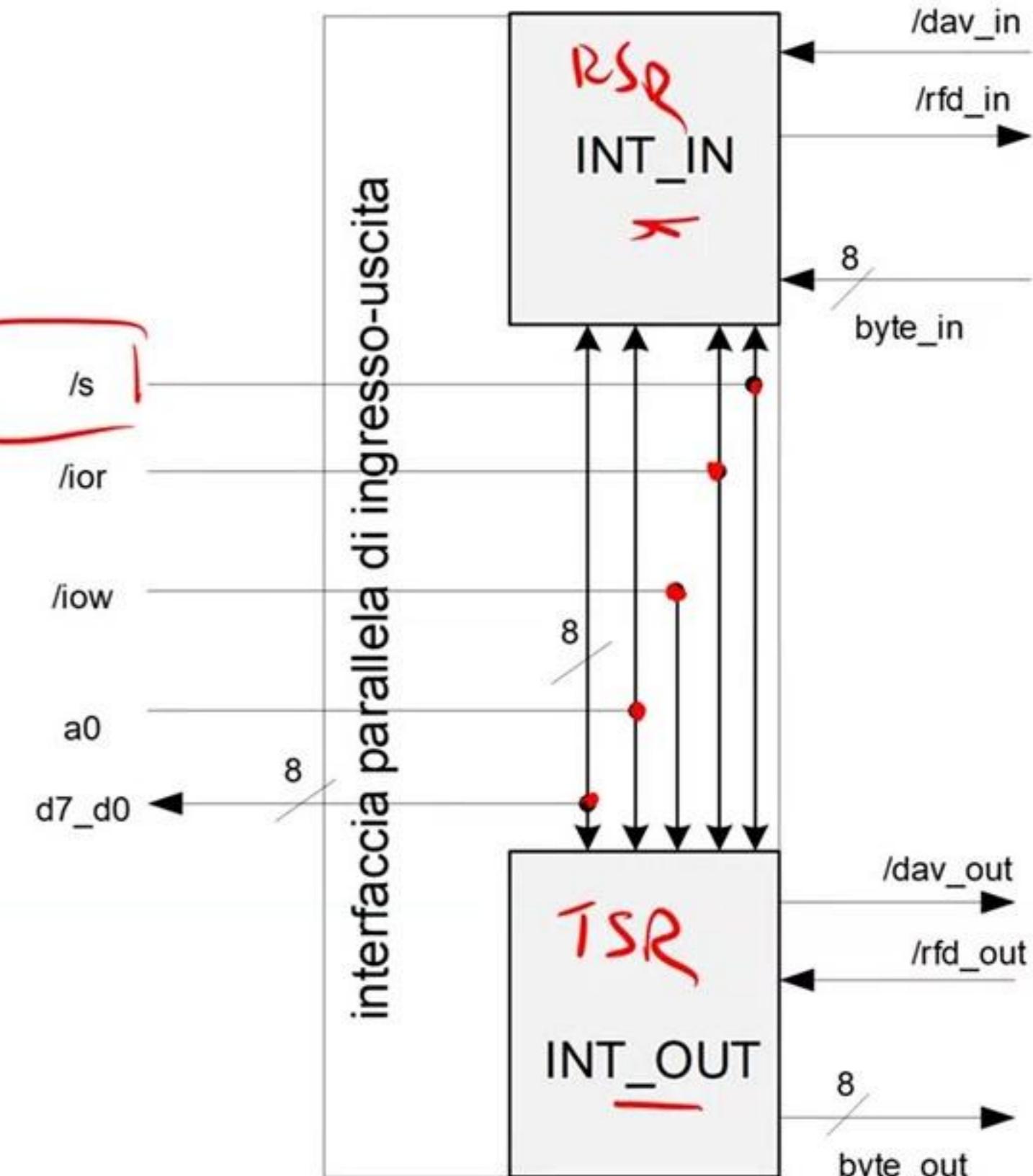
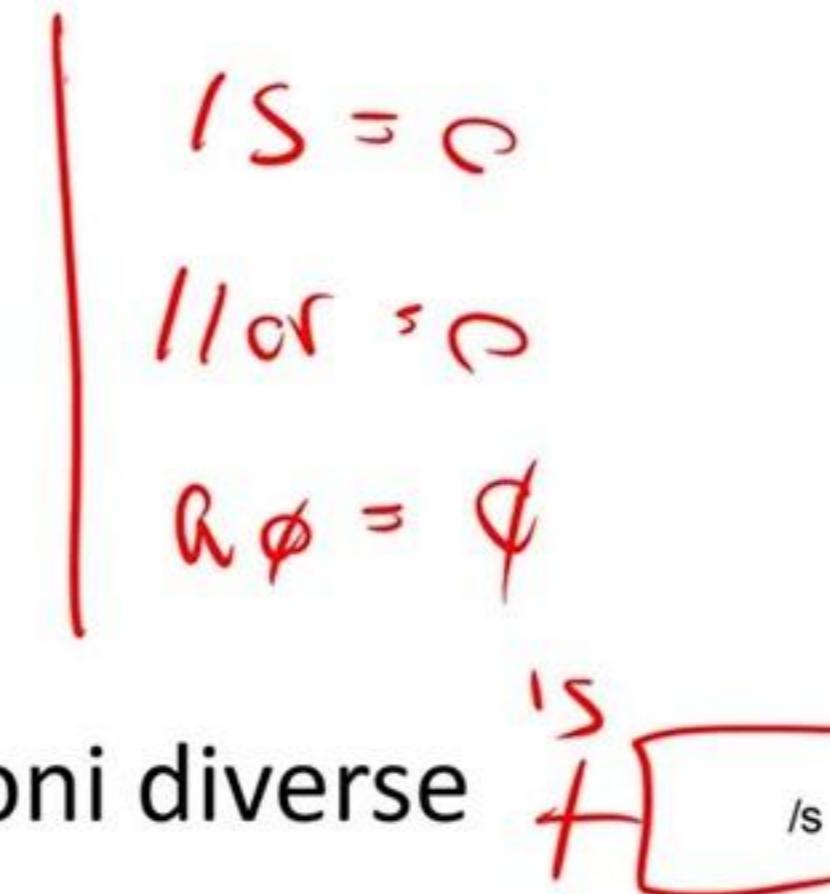
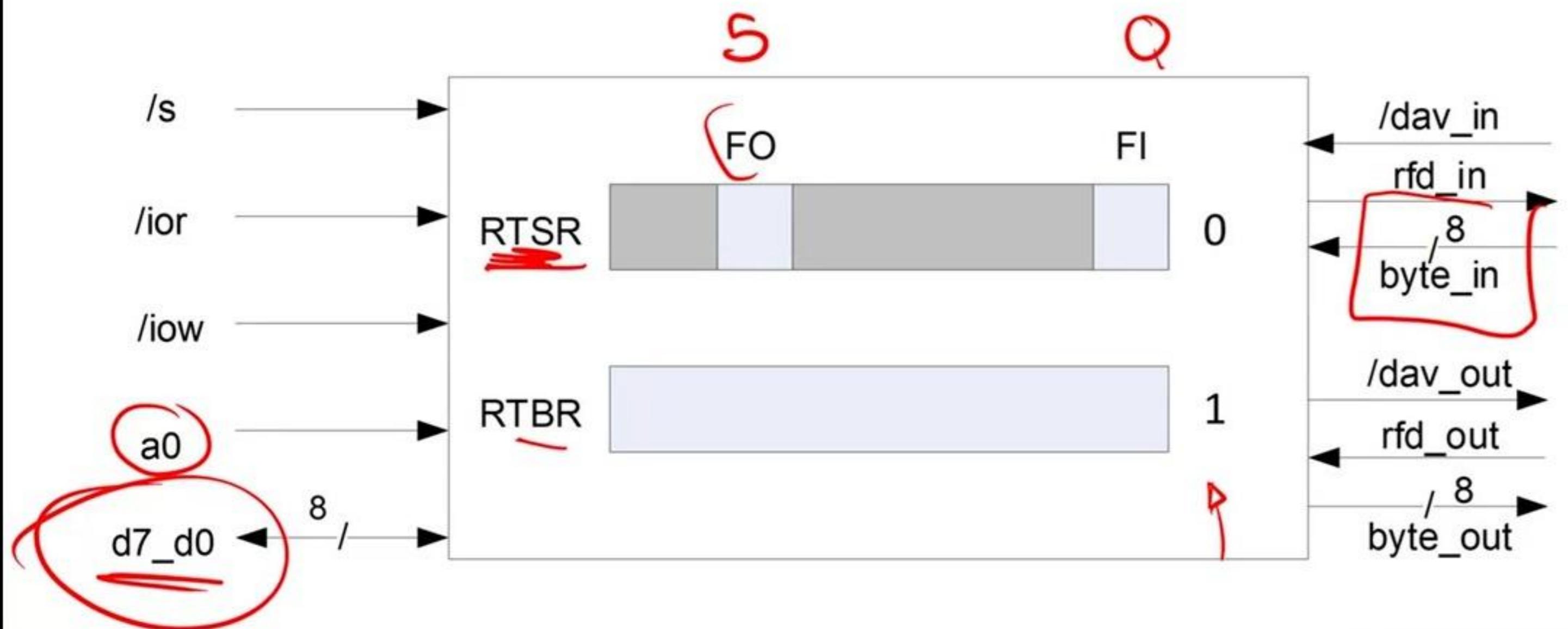
Comunicazione seriale (cont.)

- Come si fa a **sincronizzare** trasmettitore e ricevitore?
 - **No:** condividere un clock
 - **No:** condividere linee di handshake per trasmettere un singolo bit

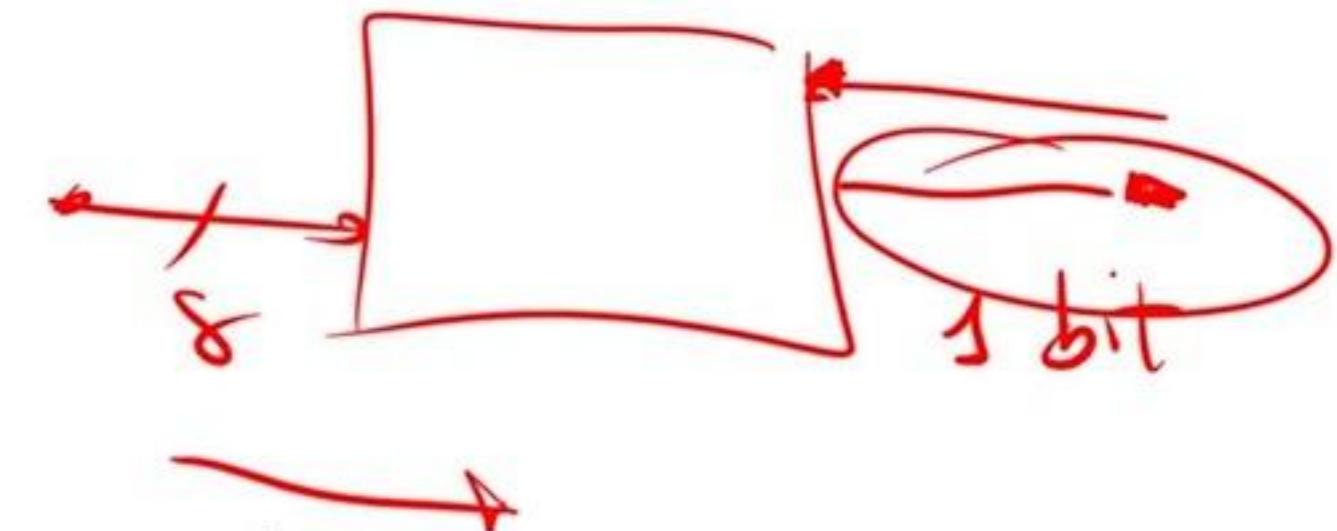


Interfaccia parallela di ingresso/uscita con handshake

- RBR e TBR mappati su un unico registro
 - Accesso distinto in base al tipo
- RSR e TSR mappati su un unico registro
 - I due bit significativi FI ed FO hanno posizioni diverse



Interfaccia seriale start/stop



- La trasmissione dei **singoli bit** avviene in modo seriale.
- Un byte viene trasmesso “un bit alla volta”, partendo (ad esempio) dal bit meno significativo.
 - Riceve dal bus **byte** (scritti dal processore in un opportuno registro di I/O) e trasmette all'esterno sequenze di **bit**
 - Riceve dall'esterno sequenze di **bit** e presenta al processore **byte** componendo quelle sequenze di bit in un registro che si possa leggere.
- Un'interfaccia *parallela* riceve invece **byte** dal processore e trasmette **byte** all'esterno, o riceve byte dall'esterno e presenta byte al processore.

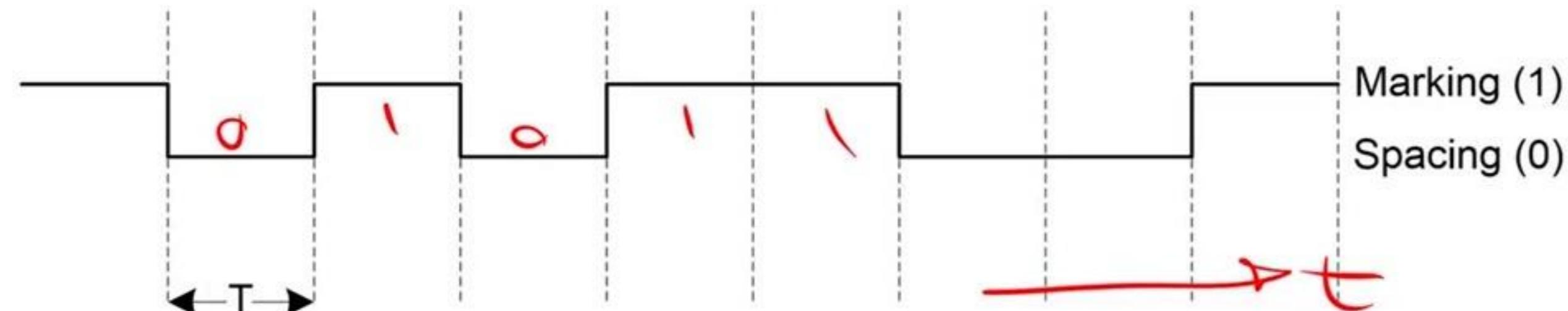
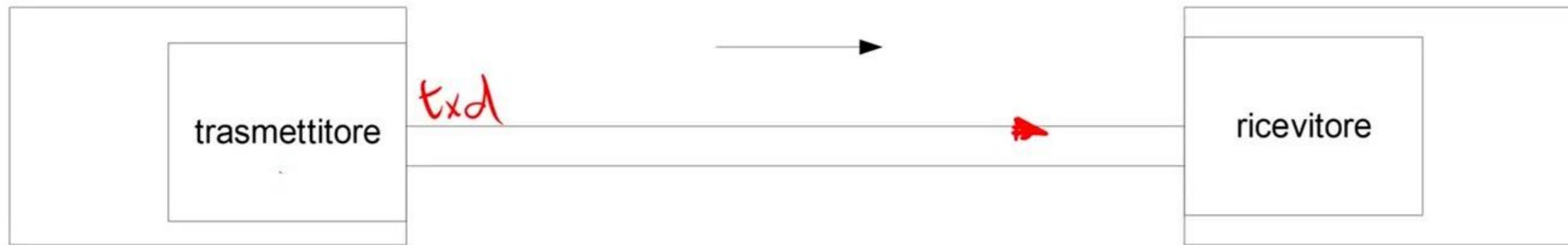
Interfacce seriali nel PC

- Un PC ha di norma più di una interfaccia (**porta**) seriale.
- Un tempo usate per modem (esterni) e mouse.
- Parecchi dispositivi che hanno del **firmware** configurabile sono, appunto, configurabili tramite un'interfaccia **seriale**
 - **router**, cioè quei dispositivi che inoltrano il traffico di rete.
- Un tempo, i calcolatori erano grossi elaboratori centrali (**mainframe**) che venivano connessi a terminali tramite, appunto, linee **seriali** (è infatti per questo che sono state inventate).
- Le interfacce seriali che sono sul PC sono piuttosto complesse
 - Ne vediamo una versione **semplificata**



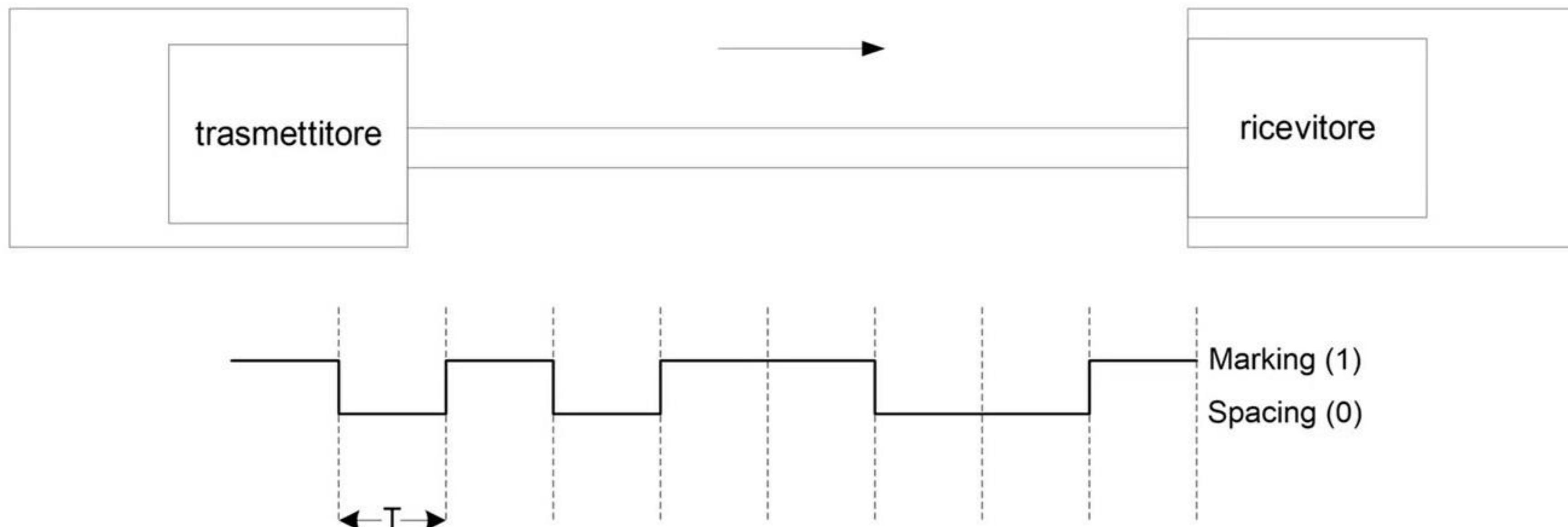
Comunicazione seriale

- Il mezzo fisico è dato da **due linee**
 - Una di massa, che funge da riferimento
 - Una che porta il segnale riferito a massa
- Marking: 1 logico
- Spacing: 0 logico
- T: tempo di bit



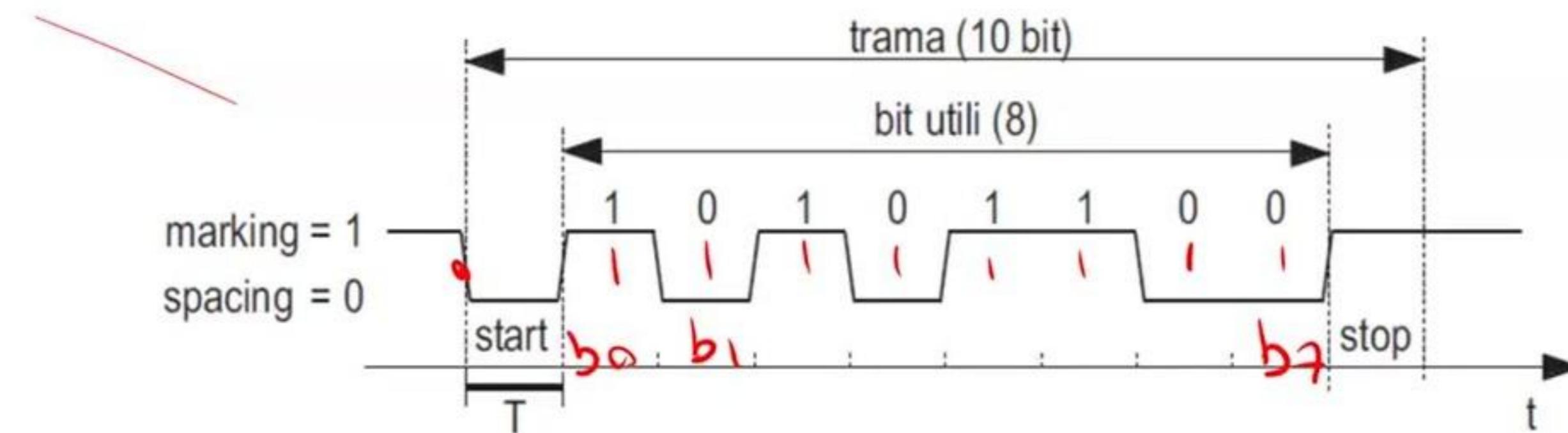
Comunicazione seriale (cont.)

- Come si fa a **sincronizzare** trasmettitore e ricevitore?
 - **No:** condividere un clock
 - **No:** condividere linee di handshake per trasmettere un singolo bit
 - **Sì:** concordare **tempo di bit T** e **formato della trama**



Formato della trama

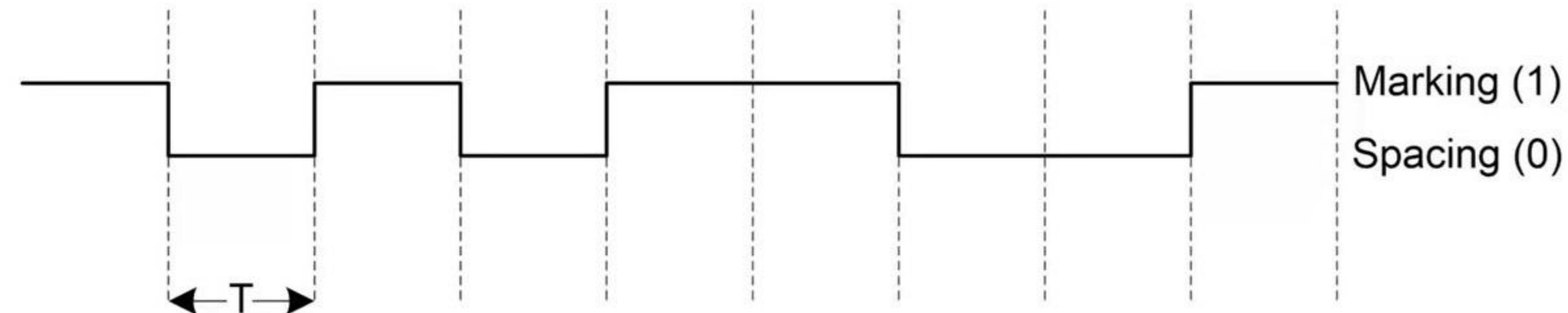
- La linea sta in stato di **marking** a riposo
 - Transizione a **spacing**: inizio della trama
 - Primo bit vale sempre 0, e non è un bit di informazione (**bit di start**)
 - Seguono da ~~5 a~~ 8 bit «utili» (LSB first)
 - La trama finisce con (almeno) un **bit di stop (marking)**



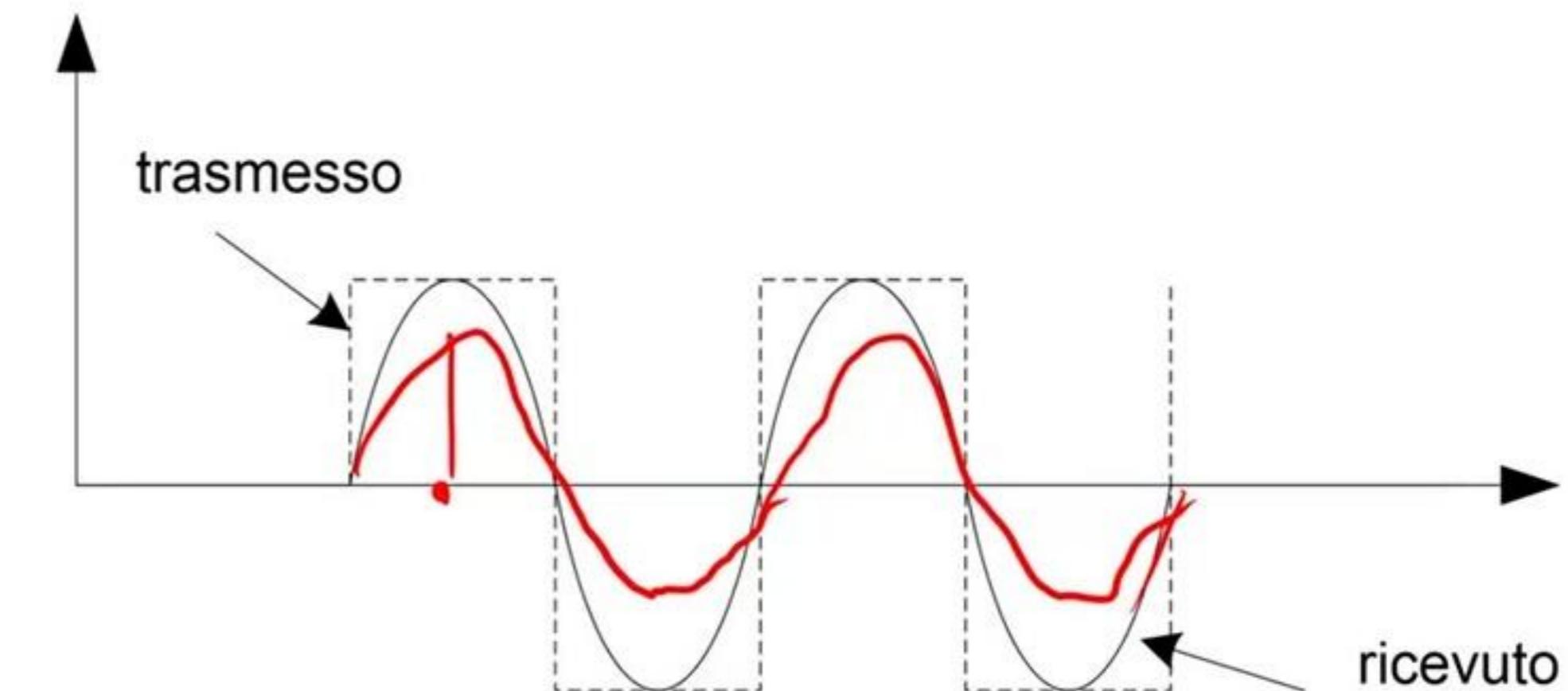
- Il bit di start ed il bit di stop sono **overhead**
 - Una trama di n bit utili è lunga almeno $n + 2$ bit
 - La velocità di trasmissione netta è $\frac{n}{n+2}$ della velocità della linea

Formato della trama (cont.)

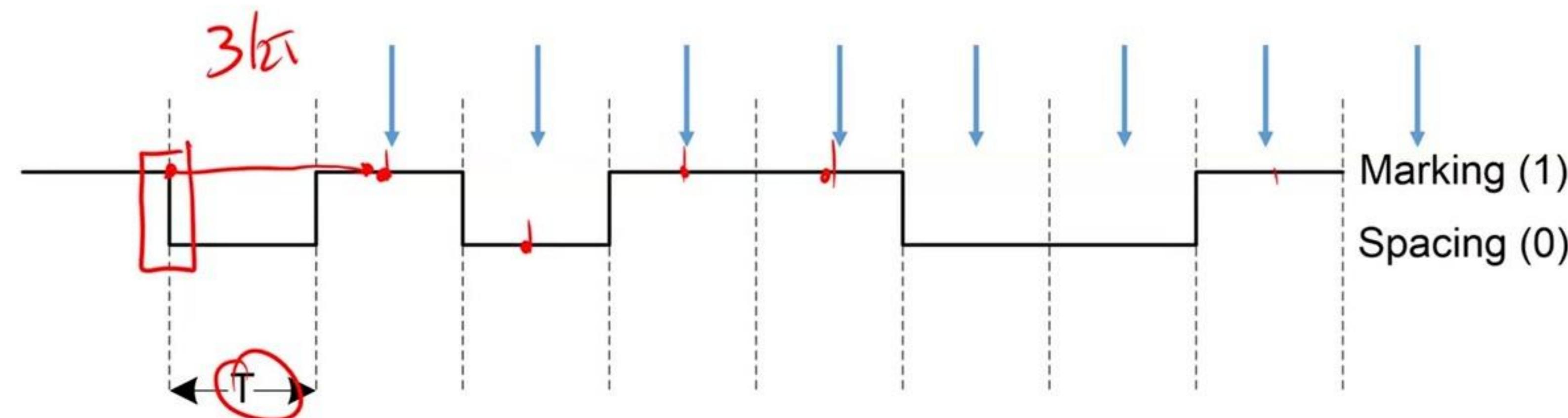
- La velocità di trasmissione netta è $\frac{n}{n+2}$ della velocità della linea
 - Converrebbe fare n molto grande per guadagnare efficienza
 - Ma non si può, per ragioni fisiche
- I clock del trasmettitore e del ricevitore **non sono identici**
 - La precisione dei clock non è molto elevata
 - Se le trame sono lunghe, gli errori si accumulano ed i bit della trama si disallineano



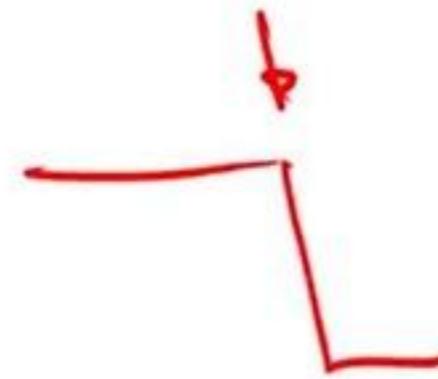
Formato della trama (cont.)



- Per leggere una trama, il ricevitore deve:
 - Riconoscere l'inizio di una trama (transizione marking/spacing)
 - Aspettare $3/2$ bit (per essere sicuro di «centrare» il bit)
 - Campionare n bit, uno ogni T



Formato della trama (cont.)



~~formato della trama~~

- La sincronizzazione tra trasmettitore e ricevitore avviene con la transizione marking/spacing di inizio trama
 - Da lì alla fine della trama, ci dobbiamo fidare della precisione dei clock
- Se il clock del ricevitore misura un tempo $T \pm \Delta T$, per poter decodificare correttamente n bit è necessario che sia $n \cdot \Delta T \leq \frac{T}{2}$

$$\frac{\Delta T}{T} \leq \frac{1}{2n}$$

~~se~~ ~~è~~

- Questo è il motivo per cui n non può essere arbitrariamente grande.

Comunicazione seriale



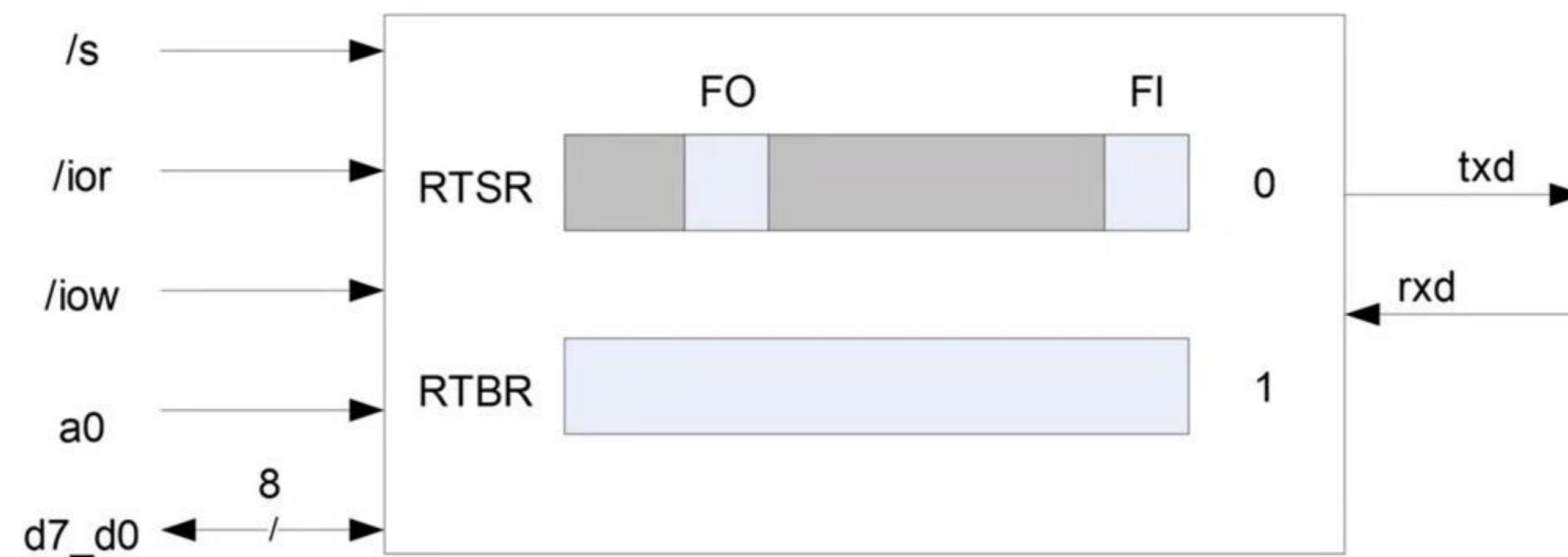
- Standard **EIA-RS232C**

- EIA = Electronic Industries Association, ente di standardizzazione)
- sviluppato all'inizio degli anni '60
- Uno standard fissa delle regole uguali per tutti per eseguire un certo compito, con ovvi benefici (interoperabilità, garanzia di correttezza, etc.)
- Include voltaggi elettrici dei segnali, temporizzazione, funzione dei segnali, formato e piedinatura dei connettori, formato delle trame, protocollo di comunicazione
- Ad esempio:
 - 0 logico: tensione positiva [+3;+25]V
 - 1 logico: tensione negativa [-25;-3]V



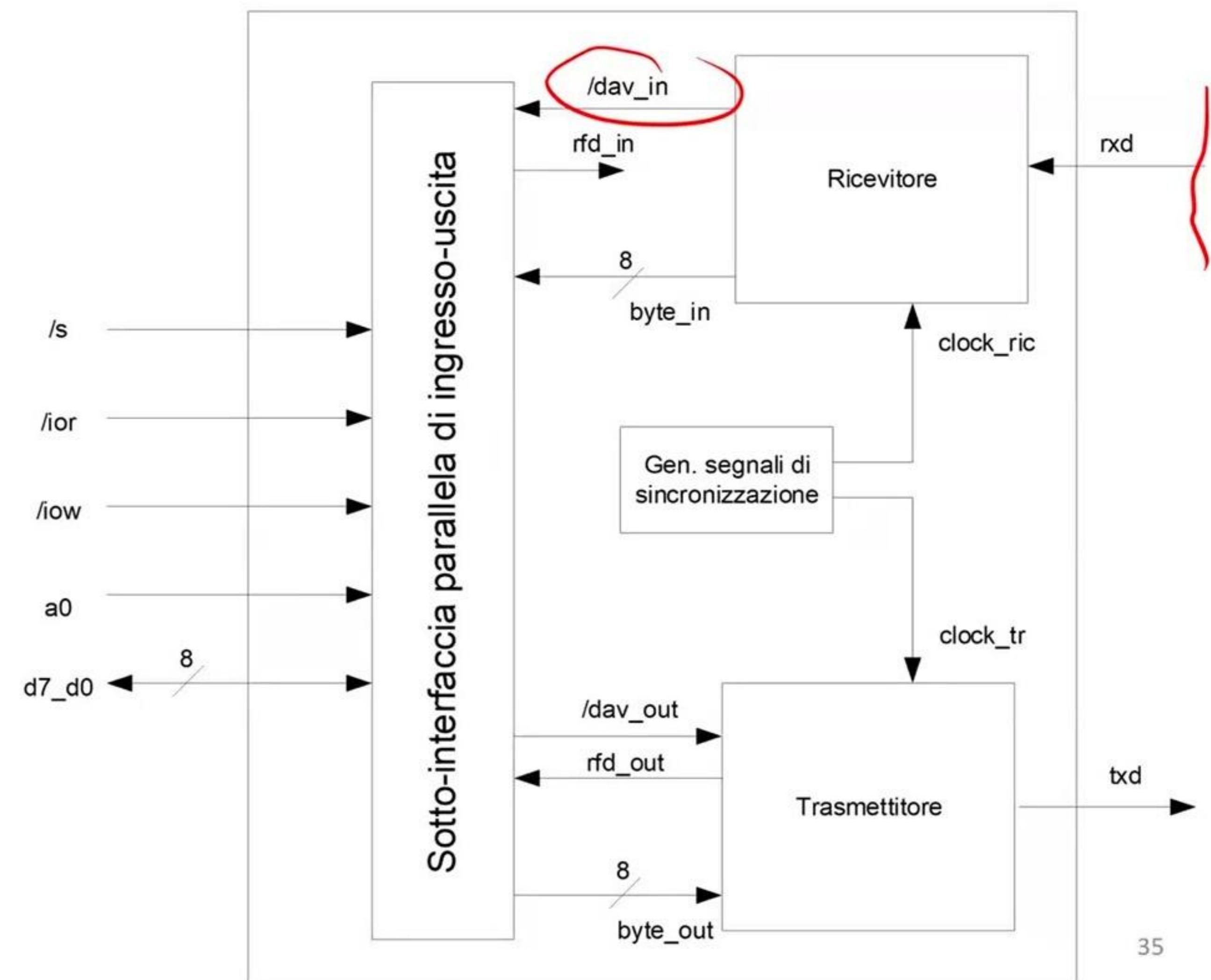
Interfacce seriali – visione funzionale

- L'interfaccia seriale è una interfaccia di ingresso/uscita con handshake



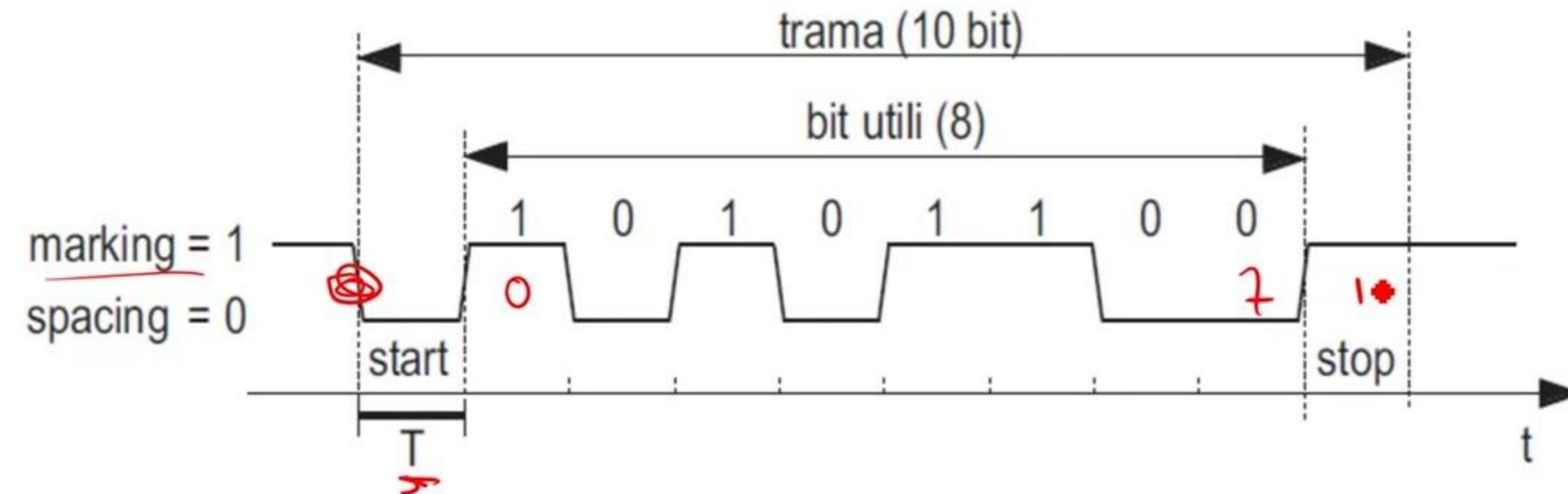
Interfacce seriali – struttura interna

- L'handshake con il ricevitore non è completo
 - Non è possibile fermare il trasmettitore dall'altra parte
 - Quando arriva una nuova trama, sovrascrivo la vecchia anche se non è stata letta



Formato della trama

- La linea sta in stato di **marking** a riposo
 - Transizione a **spacing**: inizio della trama
 - Primo bit vale sempre 0, e non è un bit di informazione (**bit di start**)
 - Seguono da 5 a 8 bit «utili» (LSB first)
 - La trama finisce con (almeno) un **bit di stop** (**marking**)



Trasmettitore seriale

RFID, TXD 1bit

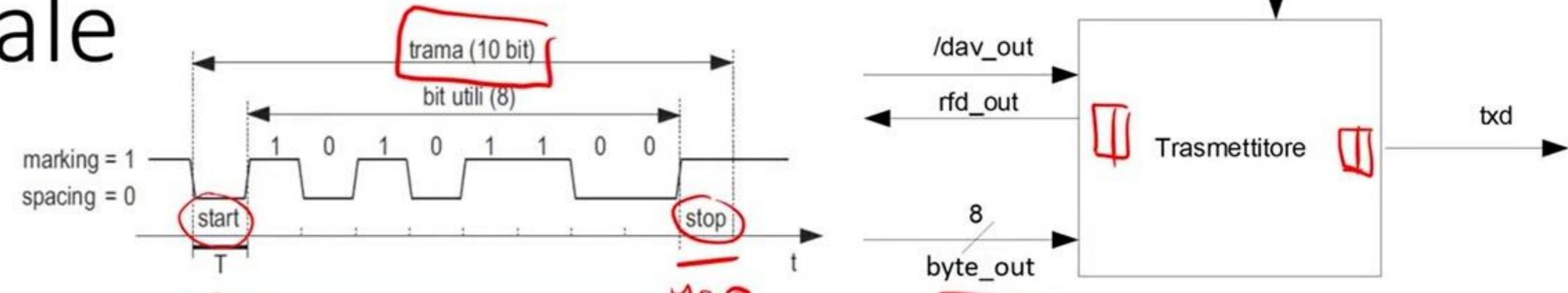
STAR

BUFFER 10 bit

COUNT

4 bit

reset



LSB

/dav_out = 1

COUNT = 1

S2

S0

RFID <= 1

TxD <= marking

BUFFER <= {marking, byteout, spacing}

COUNT <= 10 ;

reset: /dav_out = 1

txd = marking

rfd_out = 1

STAR <= S0

TxD <= BUFFER[0]

BUFFER <= {marking, BUFFER[9:1]};

RFID <= φ;

COUNT <= COUNT - 1;

```

module Trasmettitore (dav_out_, rfd_out, byte_out, txd, clock, reset_);

input clock, reset_;
input dav_out_;
input [7:0] byte_out;
output rfd_out, txd;

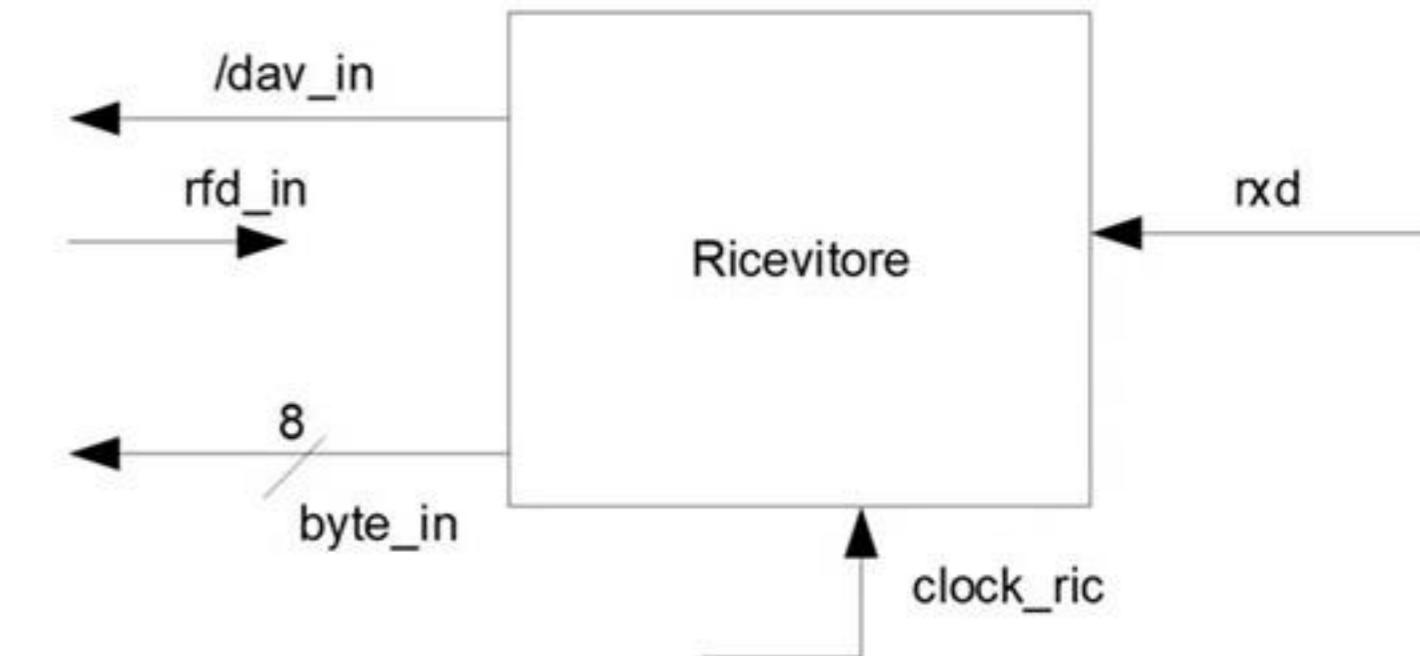
reg [3:0] COUNT;
reg [9:0] BUFFER;
reg RFD, TXD; assign rfd_out=RFD; assign txd=TXD;

reg [1:0] STAR; parameter S0=0, S1=1, S2=2, S3=3;
parameter mark=1'B1, start_bit=1'B0, stop_bit=1'B1;

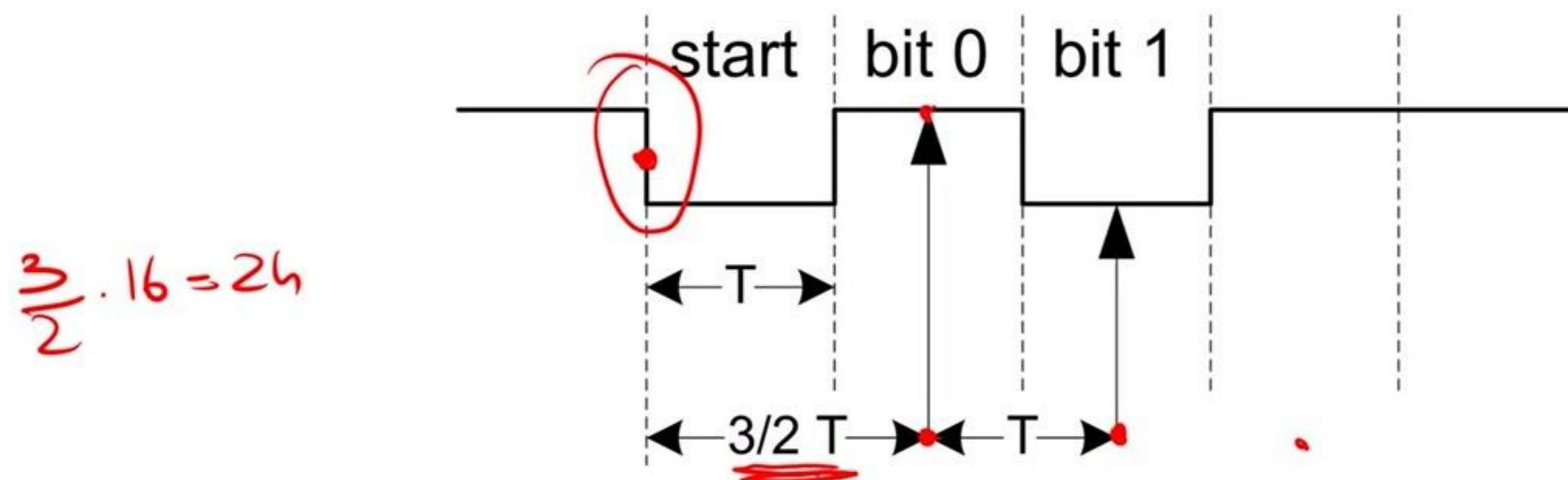
always @ (reset_==0) #1 begin RFD<=1; TXD<=mark; STAR<=S0; end
always @ (posedge clock) if (reset_==1) #3
  casex (STAR)
    S0: begin RFD<=1; COUNT<=10; TXD<=mark; BUFFER<={stop_bit, byte_out, start_bit};
      STAR<=(dav_out==1)?S0:S1; end
    S1: begin RFD<=0; TXD<=BUFFER[0]; BUFFER<={mark, BUFFER[9:1]};
      COUNT<=COUNT-1; STAR<=(COUNT==1)?S2:S1; end
    S2: begin STAR<=(dav_out==0)?S2:S0; end
  endcase
endmodule

```

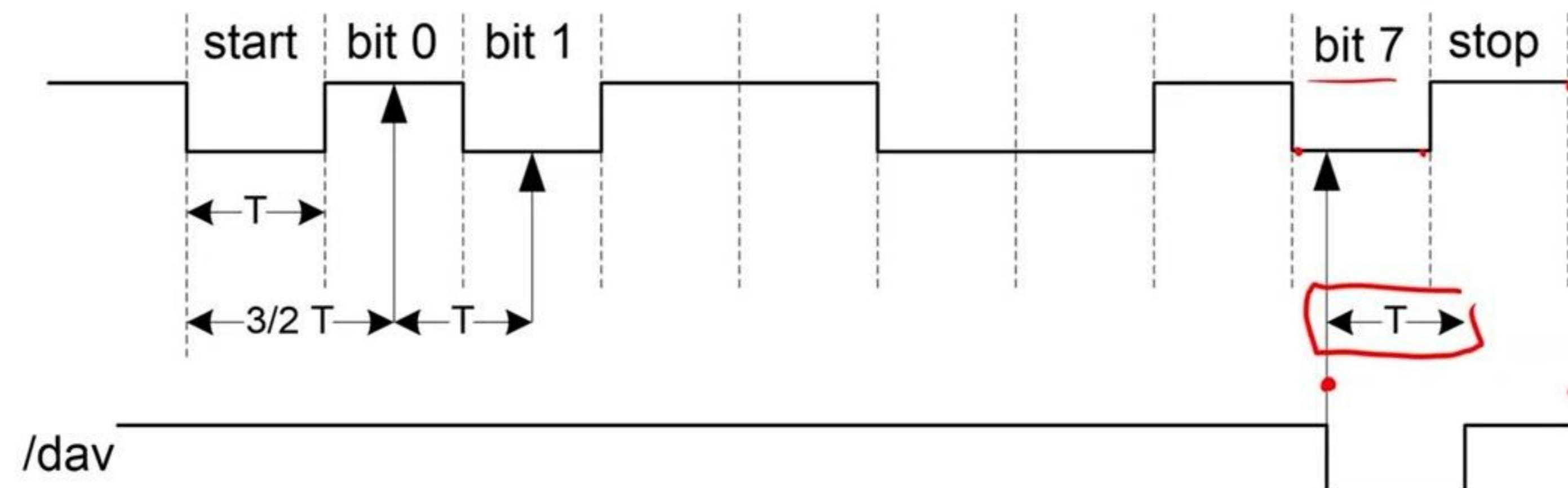
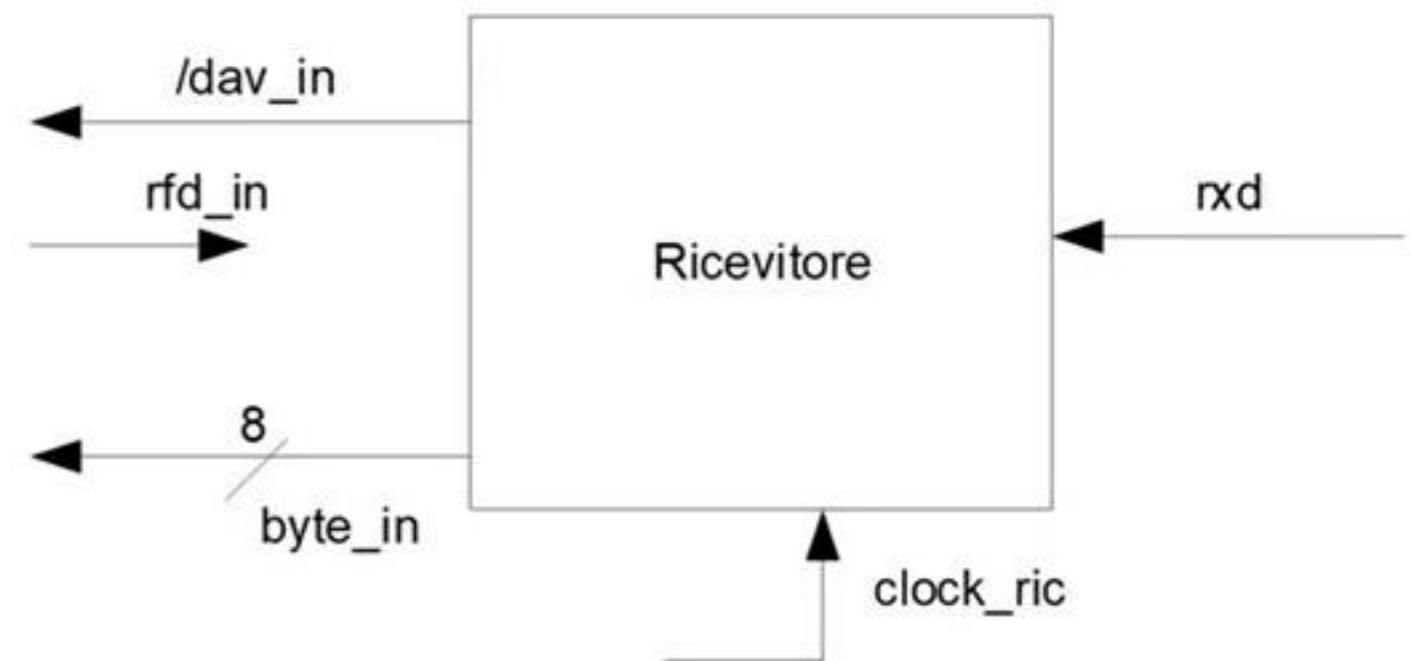
Ricevitore seriale



- Non può avere un clock uguale al tempo di bit
 - Deve poter aspettare $\frac{3}{2} T$
 - Frequenza di clock **almeno** doppia rispetto a quella di trasmissione dei bit
 - In realtà supporremo che il suo clock abbia frequenza **16x** (migliora l'accuratezza)



Ricevitore seriale



- L'handshake rimane a metà
 - Se l'interfaccia non legge il dato, questo verrà sovrascritto
 - Non c'è modo di fermare il trasmettitore dall'altro lato

Ricevitore seriale

DIV. 1 bit BUFFER 8 bit
 COUNT 4bit (?) WAIT 5bit
 STAR

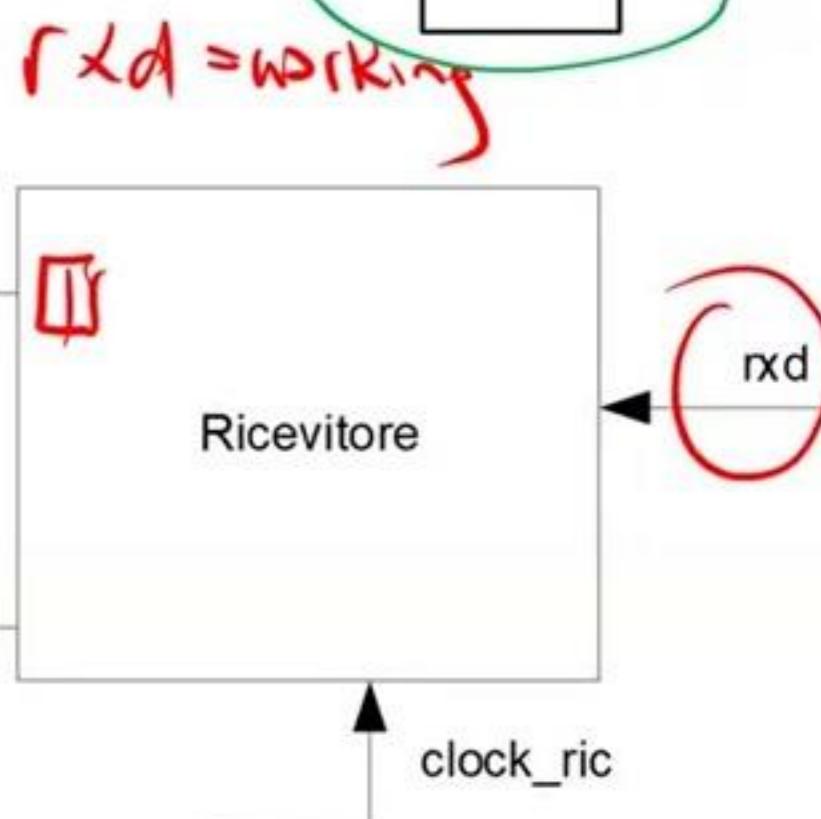
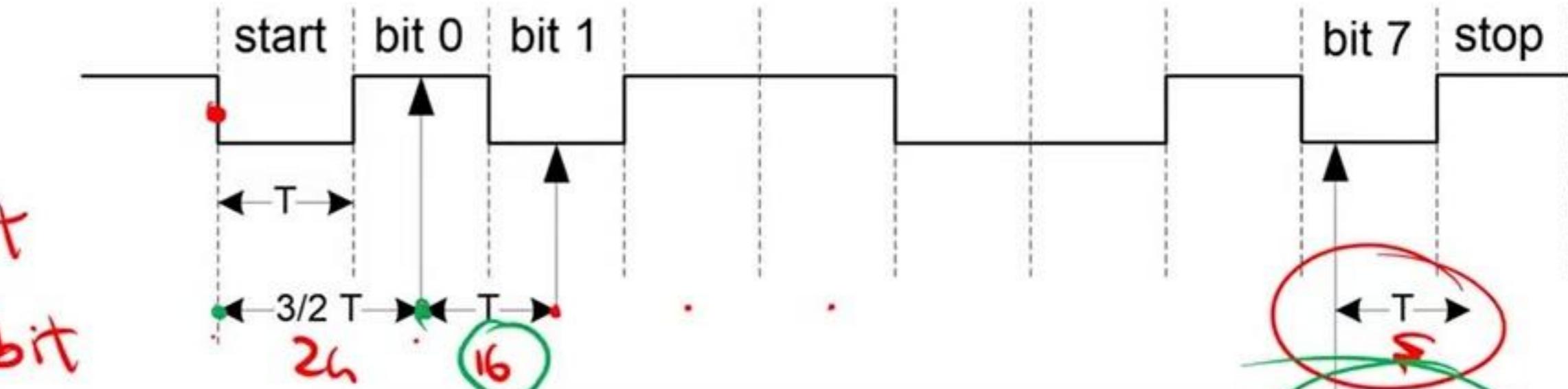
reset
 rxd
 DIV. = 1
 WAIT <= 23
 COUNT <= 8;

rxd = 0

Wbit
 COUNT > 1
 WAIT <= WAIT - 1

/dav

reset:
 DIV-in 1
 STAR <= S0
 /dav_in
 rfd_in
 byte_in 8
 COUNT <= COUNT - 1;
 WAIT <= 15;



• DIV-<=0 ;

WAIT <= WAIT - 1;

Ricevitore seriale



```

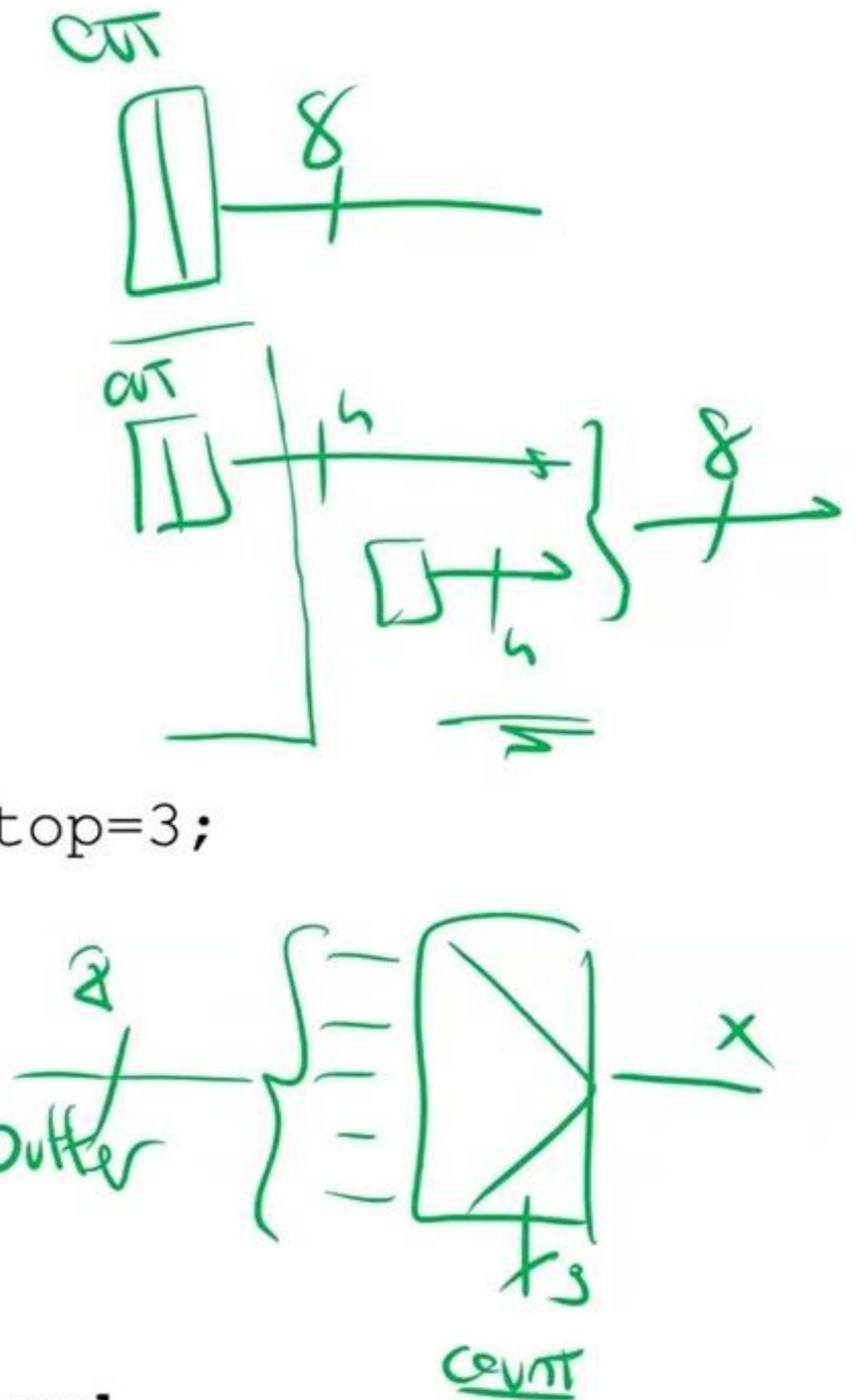
module Ricevitore (dav_in, clock, rxd, reset_, byte_in);
input clock, rxd, reset_;
output dav_in;
output [7:0] byte_in;

reg DAV_;
reg [3:0] COUNT;
reg [4:0] WAIT;
reg [7:0] BUFFER;
reg [1:0] STAR;
assign dav_in=DAV_;
assign byte_in=BUFFER;
parameter S0=0, S1=1, Wbit=2, Wstop=3;
parameter start_bit=1'B0;

always @ (reset_==0) #1 begin DAV_<=1; STAR<=S0; end
always @ (posedge clock) if (reset_==1) #3
  casex (STAR)
    S0: begin DAV_<=1; COUNT<=8; WAIT<=23;
          STAR<=(rxd==start_bit)?Wbit:S0; end
    Wbit: begin WAIT<=WAIT-1; STAR<=(WAIT==1)?S1:Wbit; end
    S1: begin COUNT<=COUNT-1; WAIT<=15; BUFFER<={rxd,BUFFER[7:1]};
          STAR<=(COUNT==1)?WStop:Wbit; end
    WStop: begin DAV_<=0; WAIT<=WAIT-1; STAR<=(WAIT==1)?S0:WStop; end
  endcase
endmodule

```

BUFFER [cont]



Conversione dig./analog. e analog./dig.

- Finora abbiamo visto soltanto interfacce che consentono a **due calcolatori** di dialogare tra loro.
- Se in un calcolatore devono entrare/uscire informazioni da/verso il resto del mondo, è necessario che queste vengano **convertite** dalla forma in cui si trovano ad una forma comprensibile per il calcolatore.
- nel mondo fisico, l'informazione è di norma associata a grandezze **analogiche**, che variano “con continuità”
- All'interno del computer, le informazioni sono associate a **stringhe di bit**, cioè a grandezze **digitali**, che variano in modo discreto.
- Sono necessari dispositivi di **conversione**
 - Ingresso: **analogico-digitale**
 - Uscita: **digitale-analogico**

Tensione

Conversione dig./analog. e analog./dig. (cont.)

- La grandezza analogica che consideriamo nel nostro caso è una **tensione**. Convertiremo questa tensione in un **numero (naturale o intero) in base 2**, e viceversa.
- Tensione v : scala di FSR volts (**Full-scale Range**)
 - Tipicamente, $FSR \in [5; 30]$
- Numero x nel quale sarà convertita: rappresentato su N bit
 - $N \in \{8, 16\}$
- A seconda dell'interpretazione del numero e della tensione, posso distinguere:
 - conversione unipolare: $v \in [0, FSR]$, $x \in [0, 2^N - 1]$
 - conversione bipolare: $v \in \left[-\frac{FSR}{2}, +\frac{FSR}{2}\right]$, $x \in [-2^{N-1}, +2^{N-1} - 1]$

Errori di conversione

- Definiamo $K = \frac{FSR}{2^N}$
- In una conversione **ideale**, dovrebbe essere $v = K \cdot x$.
- In realtà, dovremo accontentarci di $|v - K \cdot x| \leq \underline{\text{err}}$, con err detto **errore di conversione**, dovuto a:
 - Imprecisioni circuitali. Errore di non linearità, presente sia nella conversione D/A che in quella A/D
 - Arrotondamento. Errore di quantizzazione. Nella conversione A/D devo convertire una grandezza **continua** in una **discreta**. Facendo questo si perde dell'informazione

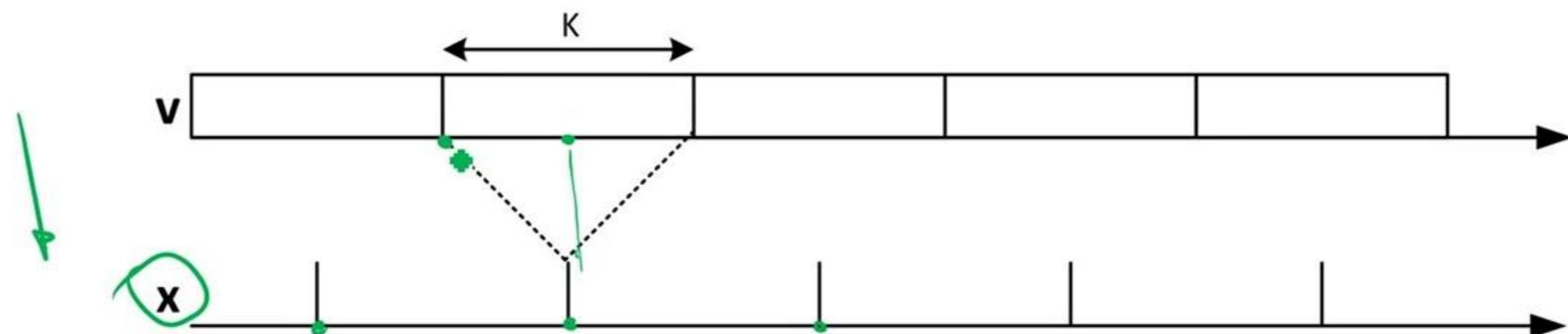
A/D

$$v = K \cdot x.$$

Errori di conversione (cont.)

$$v \in [K \cdot x - err, K \cdot x + err]$$

- L'errore di non linearità deve essere più piccolo di $\frac{K}{2}$
- Altrimenti, nella conversione **da digitale ad analogico**, gli intervalli centrati in due numeri consecutivi sarebbero **parzialmente sovrapposti**
 - Potrei convertire un numero più grande in una tensione più piccola e viceversa.
- Errore di quantizzazione: sempre pari a $\frac{K}{2}$



Errori di conversione (cont.)

- Riassumendo, abbiamo:
- conversione D/A: $err < \frac{K}{2}$ (non linearità)
- conversione A/D: $err < \frac{K}{2} + \frac{K}{2} = K$ (non linearità + quantizzazione)

Esempio

$$K = \frac{FSR}{2^N} = \frac{10.24}{256}$$

- conversione **bipolare**, con $N = 8$, $FSR = 10.24v$.
- K vale quindi **$40mv$** . Errore di non linearità minore di **$20mv$** .
- Il numero x varia tra -128 e $+127$.
 $\left[-\frac{FSR}{2}; +\frac{FSR}{2} \right]$
- Conversione **D/A ideale**:
 - $x = +127, \rightarrow v = 127 \cdot 40mv = +5.08v$.
 - $x = -128, \rightarrow v = -128 \cdot 40mv = -5.12v$.
- Visto che la conversione non è ideale, avremo invece:
 - $x = +127, \rightarrow v = 127 \cdot 40mv \pm 20mv \rightarrow v \in (+5.06, +5.10)v$
 - $x = -128, \rightarrow v = -128 \cdot 40mv \pm 20mv \rightarrow v \in (-5.14, -5.10)v$

Esempio (cont.)

- conversione **bipolare**, con $N = 8$, $FSR = 10.24v$.
- K vale quindi **40mv**. Errore di non linearità minore di **20mv**.
- Il numero x varia tra -128 e $+127$.
- Per una conversione **A/D ideale**: Quantizzazione
 - $v \in (-0.02, +0.02)v \rightarrow x = 0$
 - $v \in (-5.14, -5.10)v \rightarrow x = -\underline{128}$
 - $v \in (+5.06, +5.10)v \rightarrow x = +\underline{127}$
- Se la conversione **non è ideale**, affetta cioè anche da errori di non linearità, gli intervalli avranno una grandezza variabile, compresa tra **20mv e 60mv**.

Tempi di risposta dei convertitori

- Quelli D/A, che sono circuiti “combinatori”, sono estremamente veloci (pochi ns)

Sequenziali

- Quelli A/D hanno tempi di risposta variabili, perché sono circuiti sequenziali che possono avere architetture diverse.
 - Noi vedremo quelli ad approssimazioni successive (SAR), che hanno tempi di risposta di qualche centinaio di ns .

n.bit

Convertitori bipolari

- I convertitori **bipolari** lavorano rappresentando i numeri interi con **rappresentazione in traslazione**

- detta anche, appunto, **binaria bipolare**

- Il numero intero x è rappresentato dal naturale $X = x + 2^{N-1}$

- La tensione negativa di fondo scala, che corrisponde al numero intero negativo più piccolo, verrà convertita nel numero naturale 0

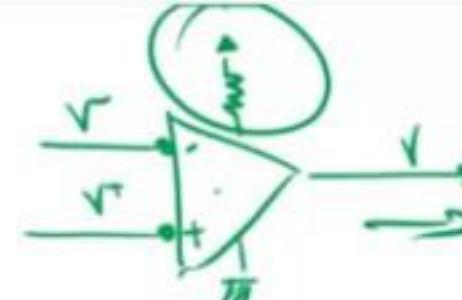
$$x = -128$$



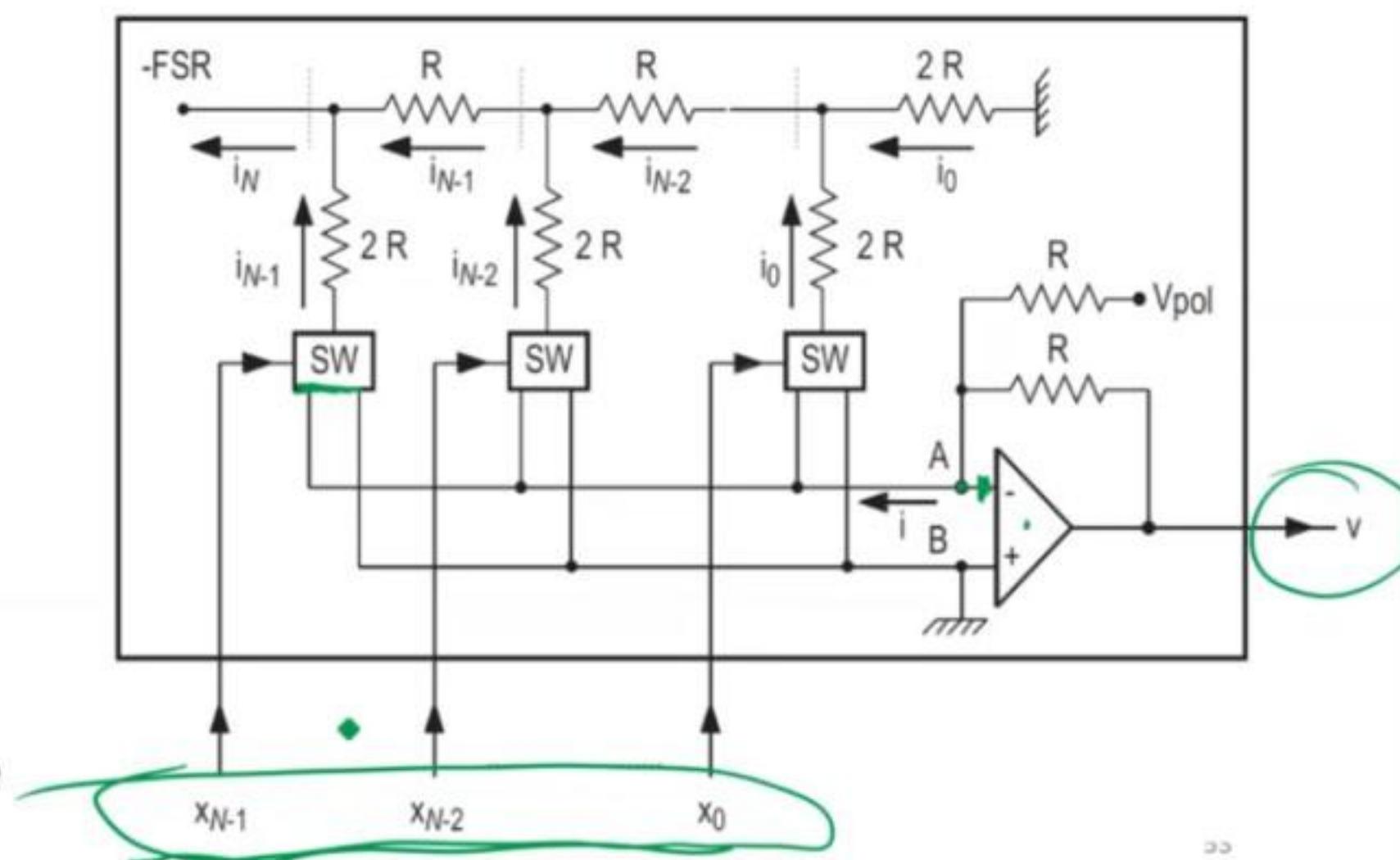
- Per convertire tra rappresentazione in traslazione e complemento a 2, basta complementare l'MSB



Convertitore D/A



- Ingresso: rappresentazione del numero x
 - Se $x \in \mathbb{Z}$, **in traslazione**
- Uscita: tensione v
- **Switch**, guidati dalla rappresentazione binaria del numero da convertire
- **Resistenze**
- **Amplificatore operazionale**



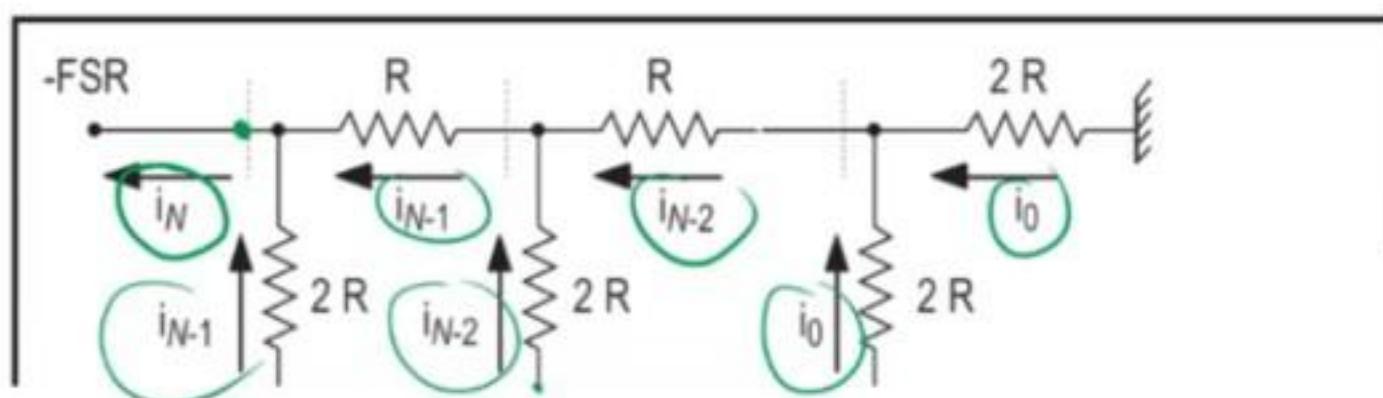
Convertitore D/A

- Assumiamo che tutte le linee verticali siano connesse a massa
- $i_N = 2^N \cdot i_0$
- $i_N = \frac{FSR}{R}$
- $i_0 = \frac{FSR}{2^N \cdot R} = \frac{K}{R}$

$$i_1 = 2 \cdot i_0$$

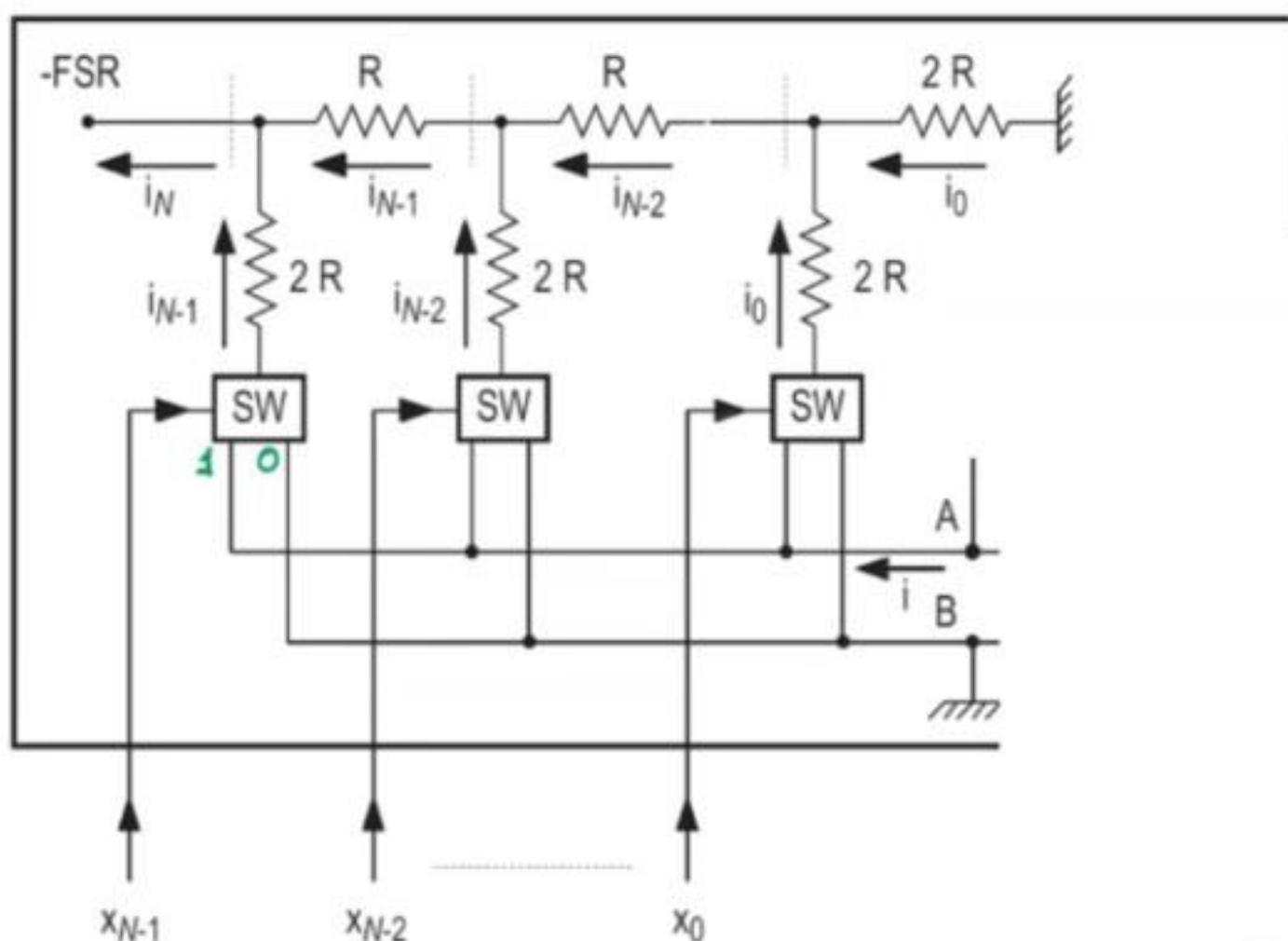
$$i_N = 2^N \cdot i_0$$

$$i_N = \frac{FSR}{R}$$



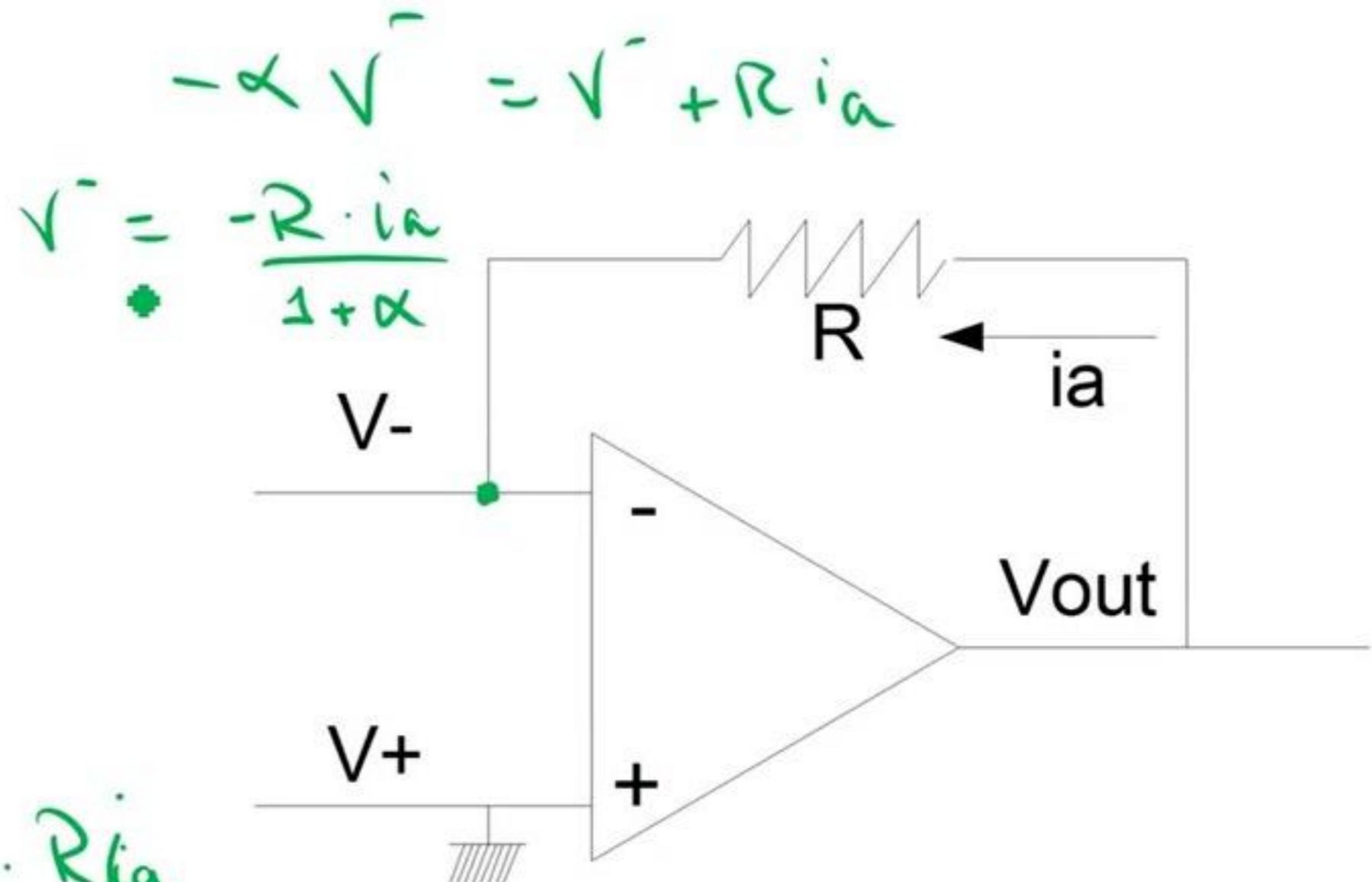
Convertitore D/A (cont.)

- Gli switch sono guidati dai **bit della rappresentazione** del numero da convertire.
- Quando $x_j = 0$, la linea è connessa a B
 - e quindi a massa
- Quando $x_j = 1$, la linea è connessa ad A



Convertitore D/A (cont.)

- Amplificatore operazionale
- Non fa passare corrente al suo interno
- $V^{out} = \alpha \cdot (V^+ - V^-)$,
- $\alpha \gg 1$ (importante!)
- $V^{out} = \alpha \cdot (V^+ - V^-) = -\alpha \cdot V^-$
- • $V^{out} - R \cdot i_a = V^-$
- $V^{out} = -\alpha \cdot V^- + \alpha \cdot R \cdot i_a$



$$V_{out} = \frac{\alpha}{1 + \alpha} \cdot R i_a$$

$$V^{out} = \frac{\alpha}{1 + \alpha} \cdot R \cdot i_a \cong R \cdot i_a$$

→ $V^- \cong 0$

Convertitore D/A (cont.)

- La corrente che esce da A e va verso sinistra vale:

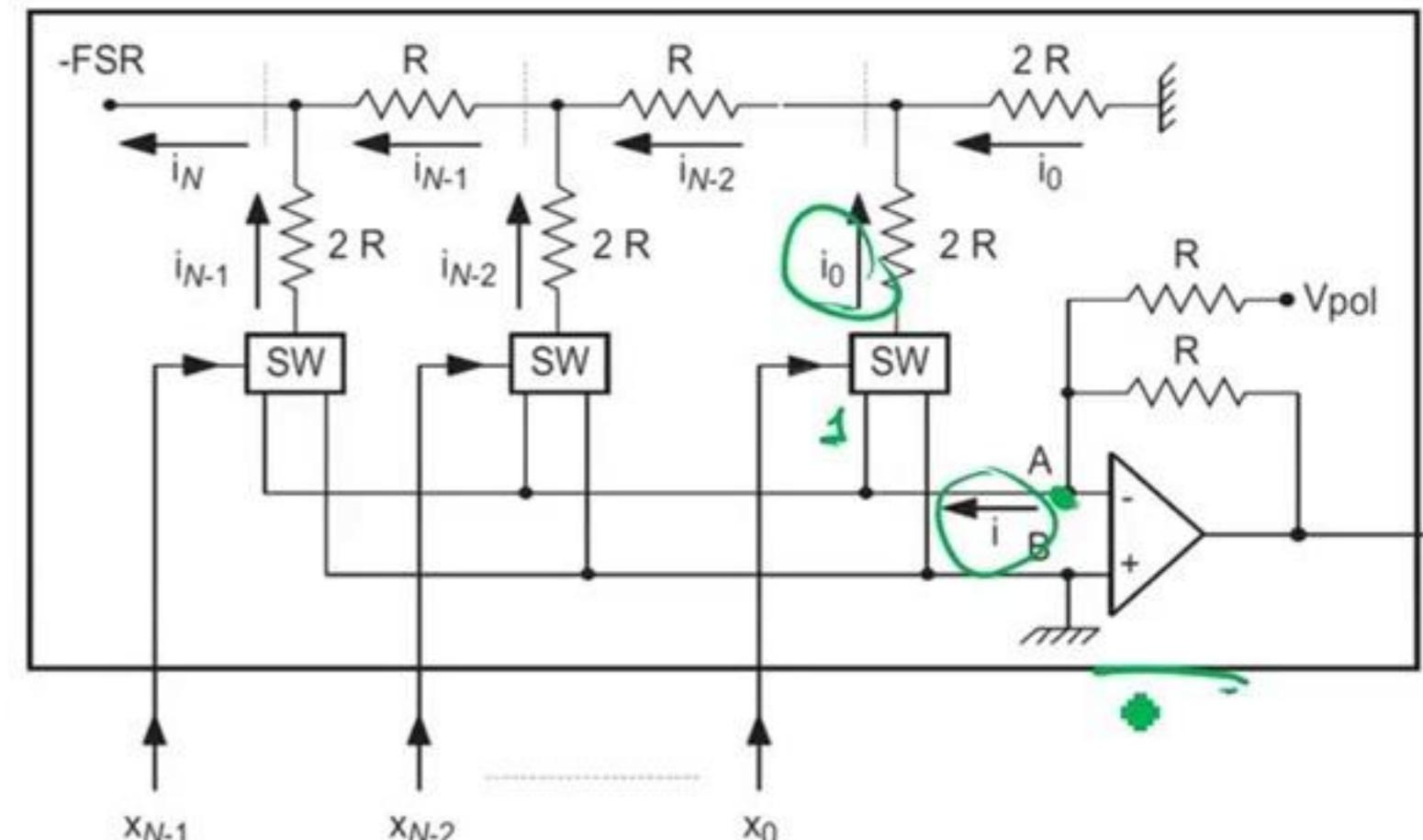
$$i = x_0 \cdot i_0 + x_1 \cdot i_1 + \dots + x_{N-1} \cdot i_{N-1}$$

$$= i_0 \cdot x_0 + (2 \cdot i_0) \cdot x_1 + \dots + (2^{N-1} \cdot i_0) \cdot x_{N-1} = i_0 \cdot$$

$$i_0 = \frac{FSR}{2^N} \cdot \frac{1}{R} = \frac{K}{R}$$

Quindi

$$i = \frac{K}{R} \cdot X$$



$$i = \frac{K}{R} \cdot X = \frac{FSR}{2^N} \cdot \frac{1}{R} \cdot \sum_{i=0}^{N-1} 2^i \cdot x_i$$

Convertitore D/A (cont.)

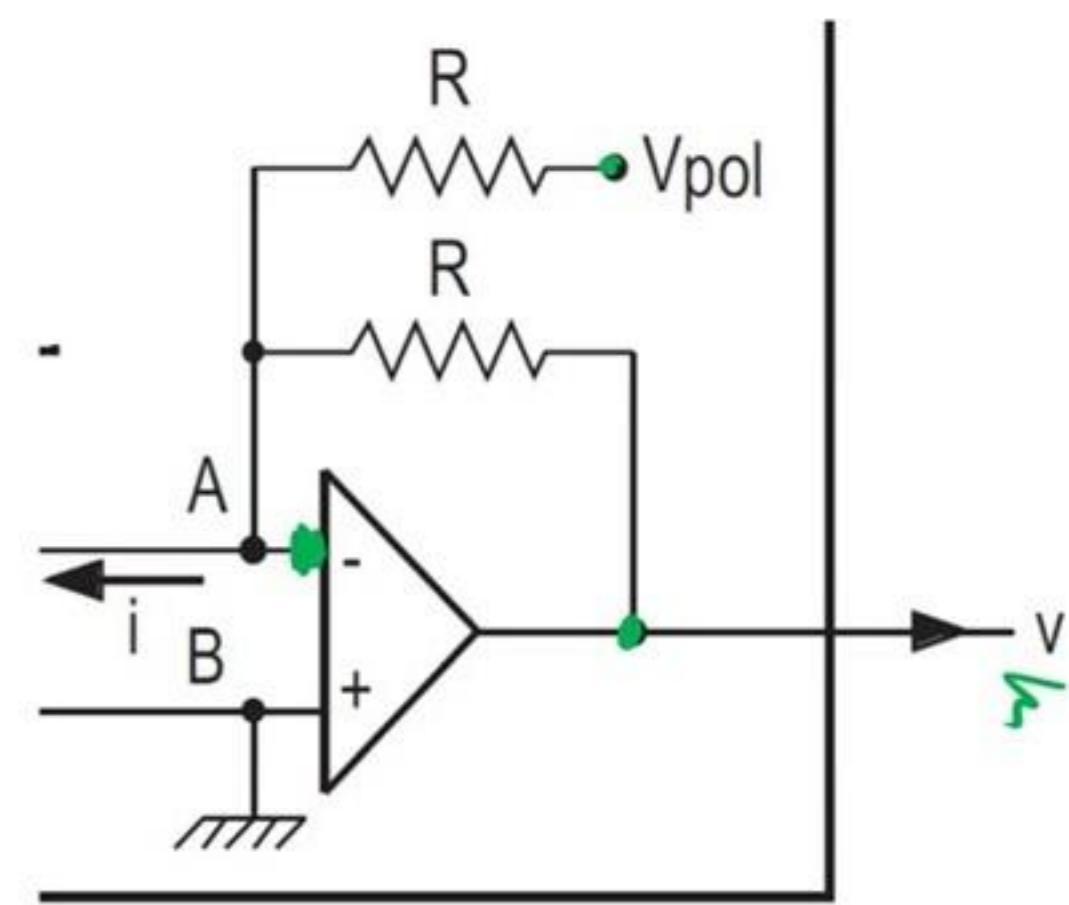
- Equazioni di bilancio della corrente al nodo A:

$$\frac{K}{R} \cdot X = \frac{V_{pol} + V}{R}$$

• cioè:

$$V = K \cdot X - V_{pol}$$

$$V = K \cdot \left(X - V_{pol} \cdot \frac{2^N}{FSR} \right)$$



Convertitore D/A (cont.)

$$V = K \cdot \left(X - V_{pol} \cdot \frac{2^N}{FSR} \right)$$

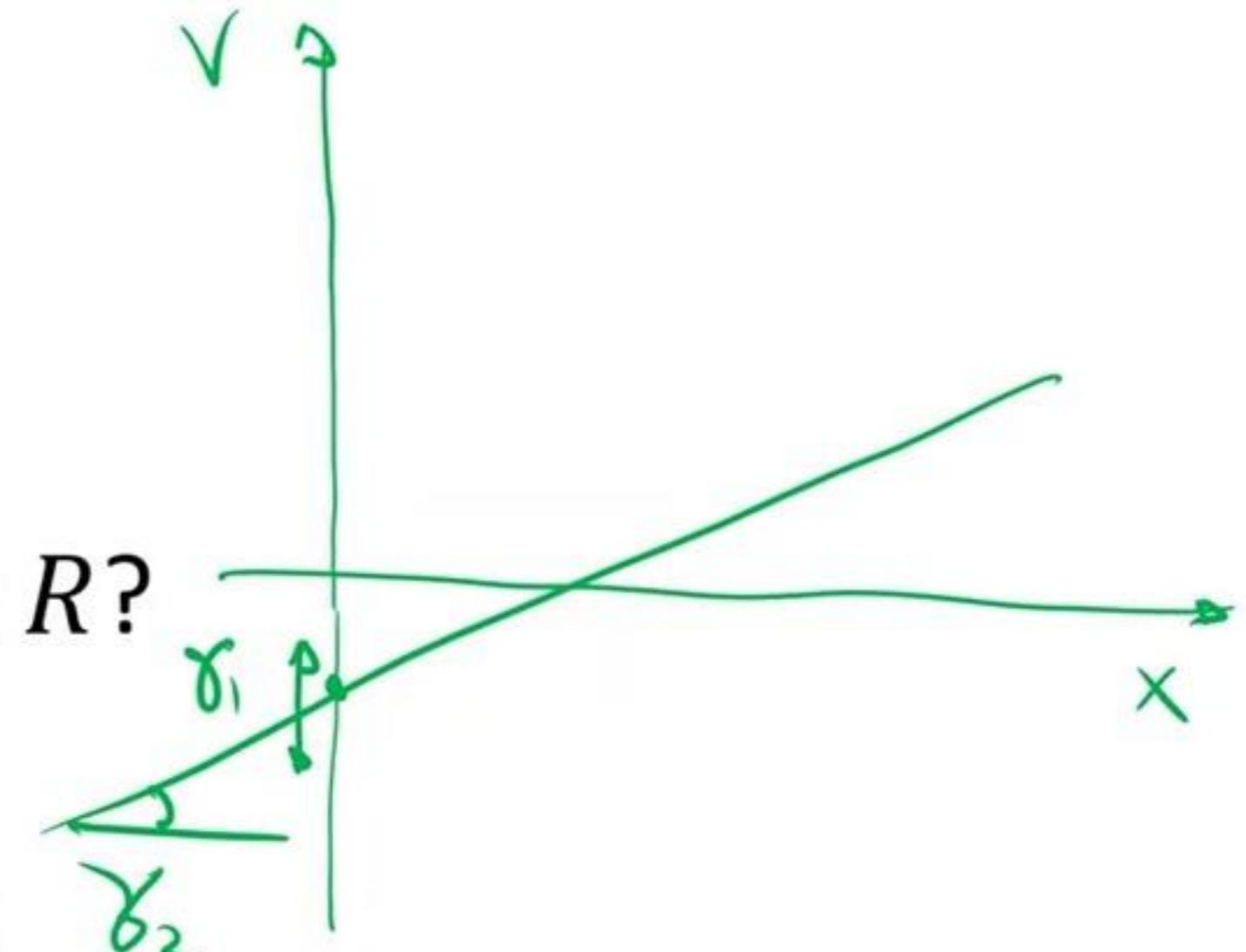
- $V_{pol} = 0 \rightarrow$ convertitore **unipolare** $V = K \cdot X$
- $V_{pol} = \frac{FSR}{2} \rightarrow$ convertitore **bipolare**, con $V = K \cdot (X - 2^{N-1})$



Imprecisioni sulle resistenze

- Che succede se le due resistenze sono diverse da R ?

$$\frac{K}{R} \cdot X = \frac{V_{pol}}{R_1} + \frac{V}{R_2} = \frac{V_{pol} \cdot \gamma_1 + V \cdot \gamma_2}{R}$$

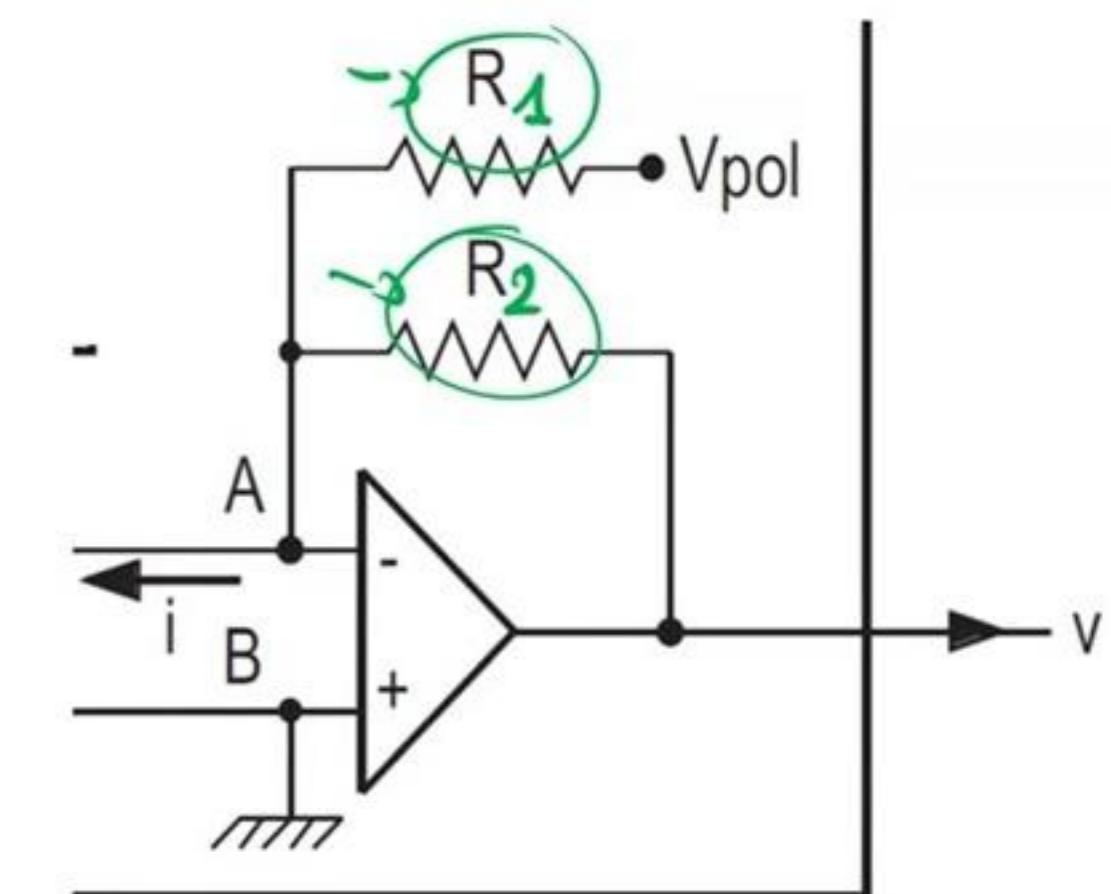


- Con $\gamma_i = \frac{R}{R_i}$

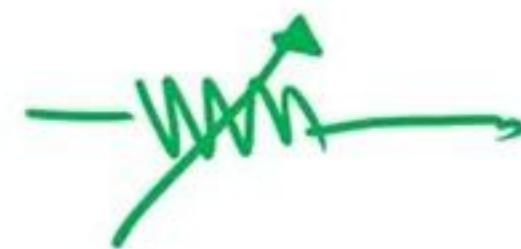
$$V = \frac{K}{\gamma_2} \cdot \left(X - V_{pol} \cdot \gamma_1 \cdot \frac{2^N}{FSR} \right)$$

errore guadagno

$$-\frac{V_{pol} \cdot 2^N}{FSR}$$



Imprecisioni sulle resistenze



$$V = \frac{K}{\gamma_2} \cdot \left(X - V_{pol} \cdot \gamma_1 \cdot \frac{2^N}{FSR} \right)$$

- Le due resistenze vanno tarate, in modo che sia $\gamma_1 = \gamma_2 \cong 1$
- Solo a valle della taratura abbiamo un errore di non linearità minore di $\frac{K}{2}$

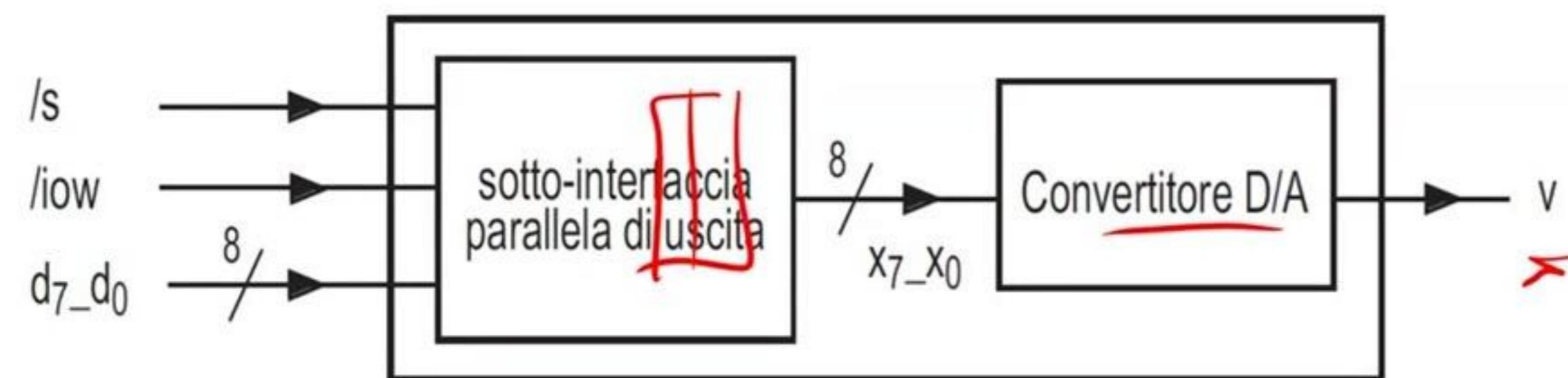
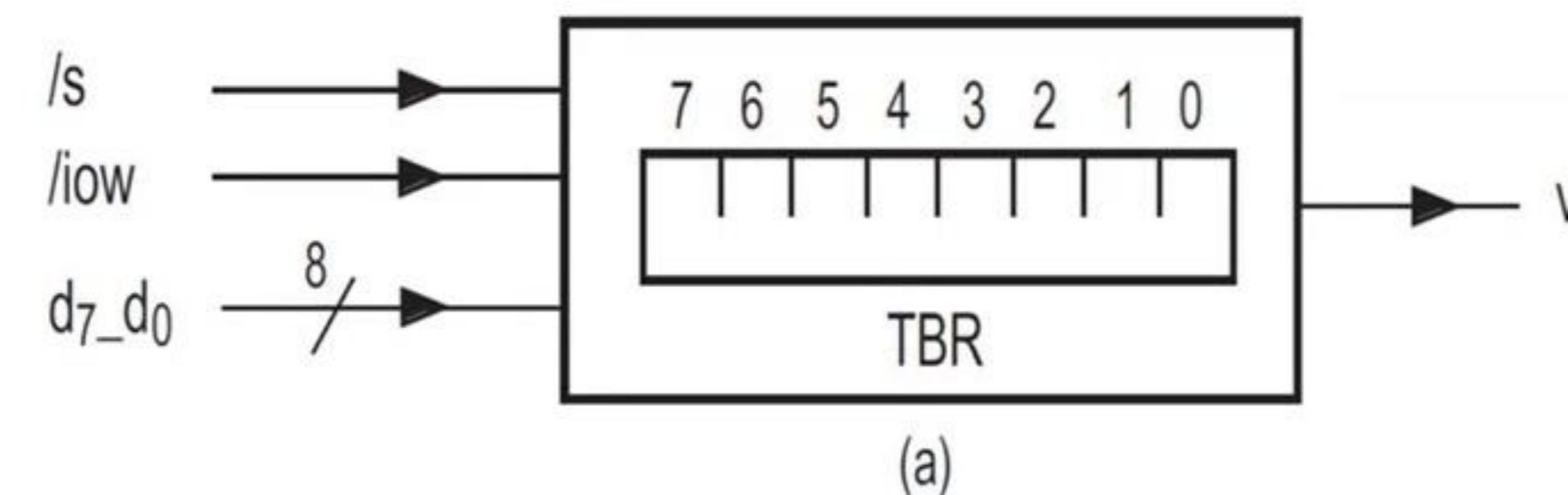
Convertitore D/A: tempi di risposta e tensioni

- Di fatto, circuito «combinatorio»: **molto veloce**
- Problemi di transizioni multiple dello stato di uscita.
- $X(i) = 01111111$
- $X(i + 1) = 10000000$
- Si mette a valle un filtro passa-basso
 - taglia le variazioni a frequenza troppo elevata

(\ \ \ \ \ \ \ \)

Interfaccia per la conversione D/A

- Semplice interfaccia parallela senza handshake



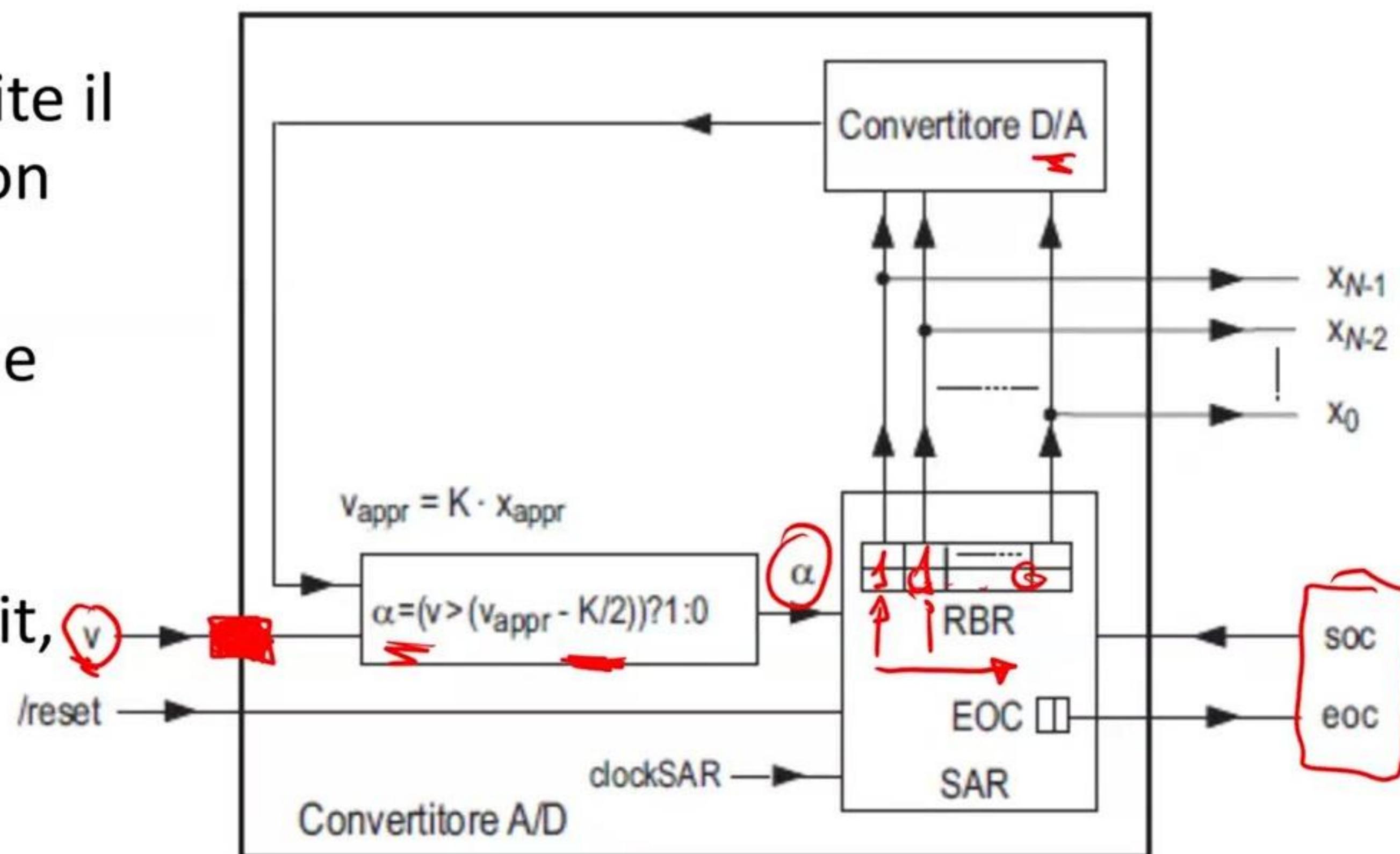
Convertitore Analogico/Digitale

- Descriviamo un **convertitore A/D ad approssimazioni successive ad 8 bit.**
- La sua parte centrale è una ~~RSS~~, con un proprio clock, detta **SAR** (Successive Approximation Register).
- Al suo interno ha un **convertitore D/A (dello stesso tipo del convertitore A/D)**
 - Bipolare se bipolare, unipolare se unipolare

Convertitore A/D (cont.)

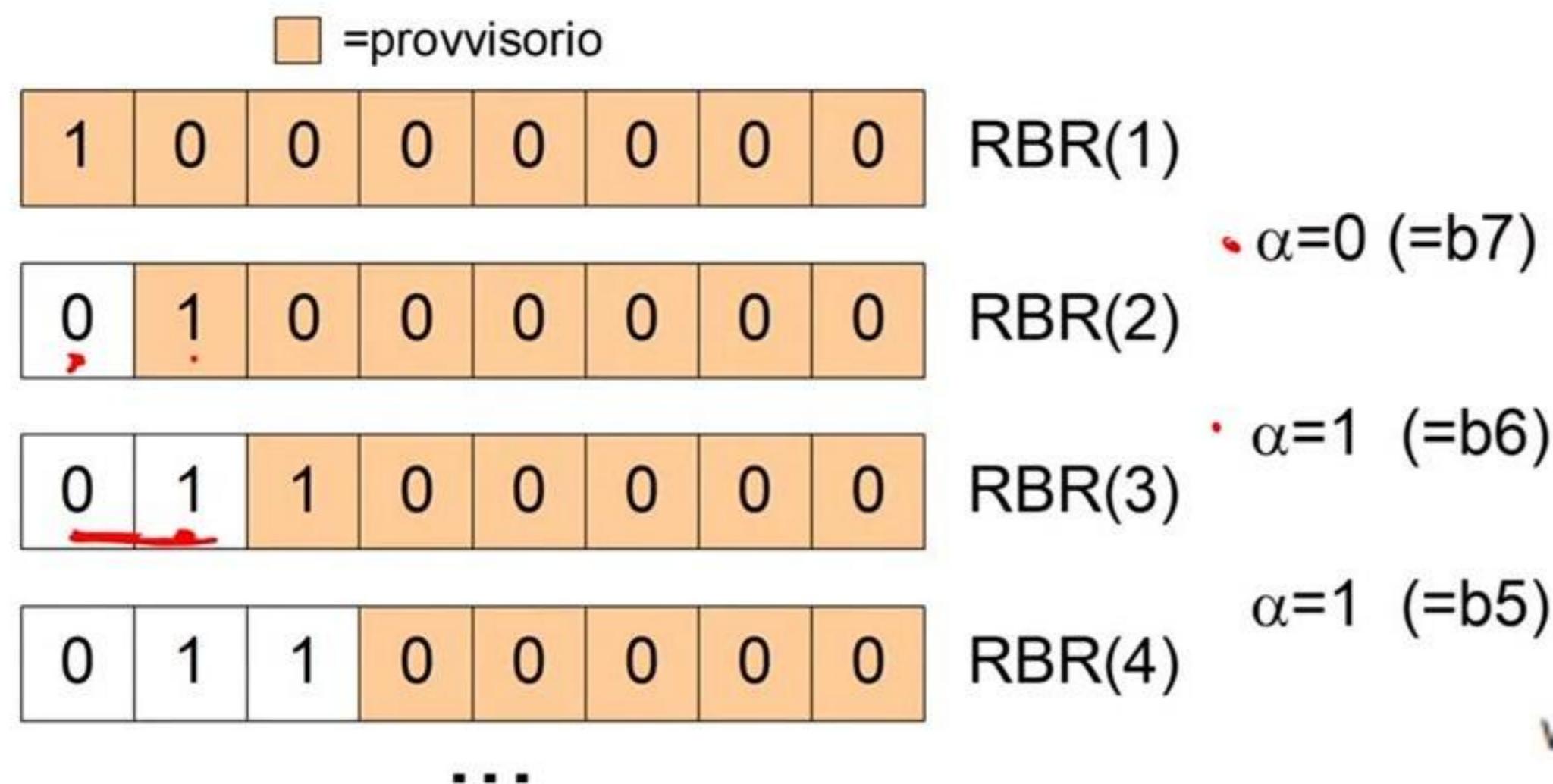
bisezione

- Esegue ricerca logaritmica
- Converte numeri in tensioni tramite il convertitore D/A e le confronta con quelle date in ingresso
- Cicla finché non genera la tensione «corretta»
 - La più vicina a quella di ingresso
- Ad ogni passo mette a posto un bit, partendo dal più significativo
 - N. passi = n. bit

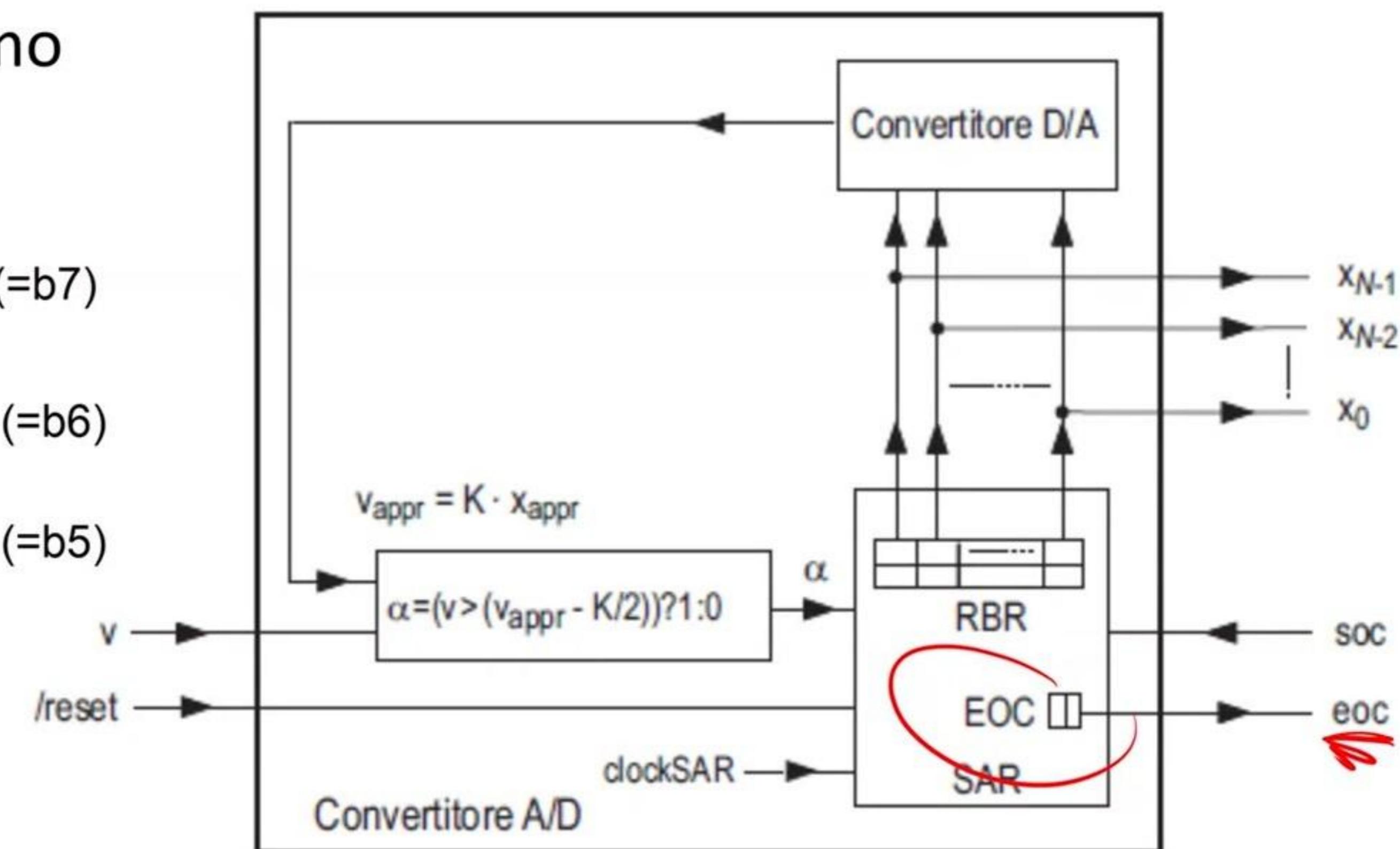


Convertitore A/D (cont.)

- **Ricerca logaritmica**
- Al passo j , α dà il bit $(n - j)$ -simo

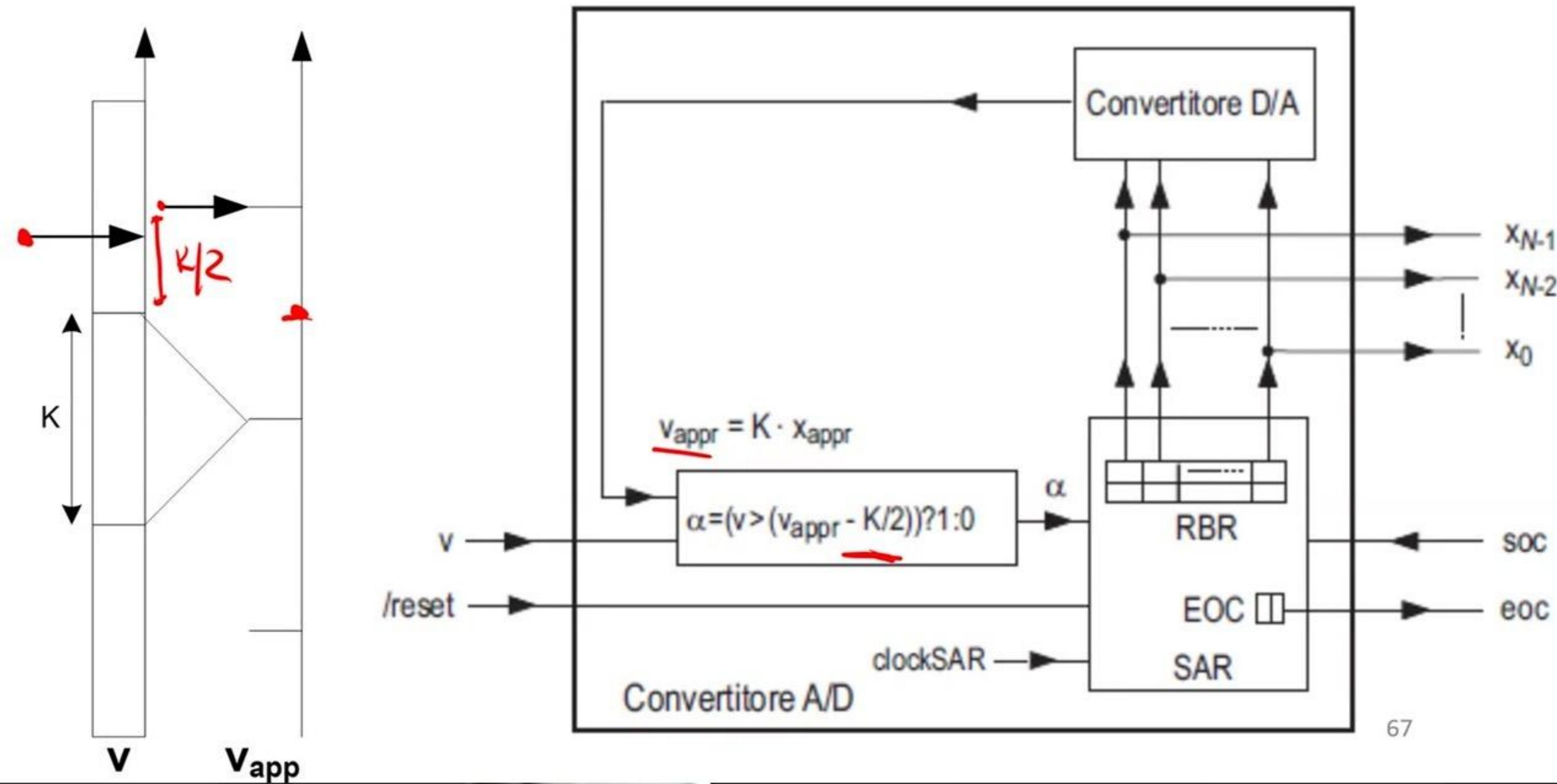


- v deve rimanere costante
 - Campionata con latch analogico



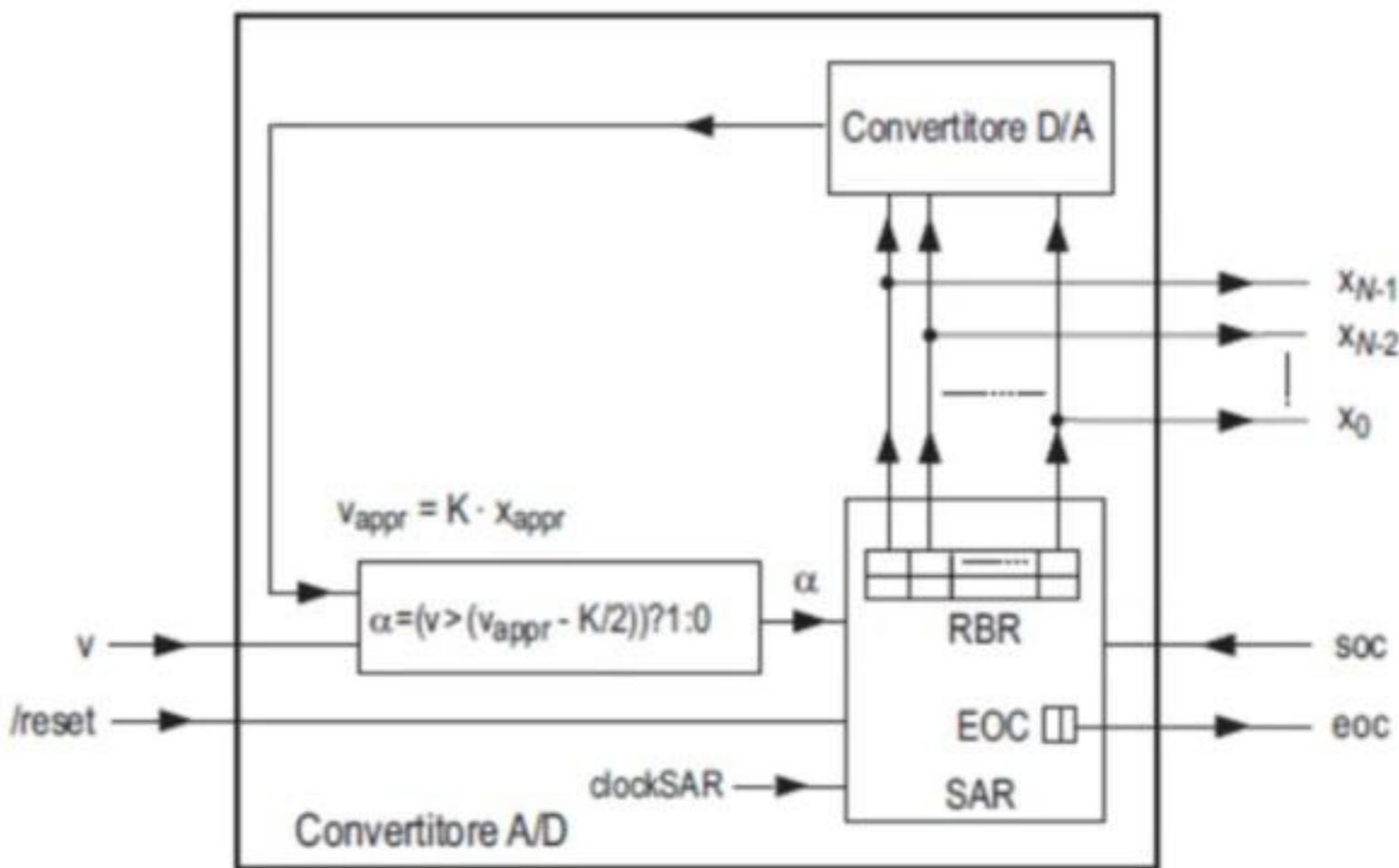
Convertitore A/D (cont.)

- Senza il termine $K/2$ nel confronto, l'errore di quantizzazione sarebbe pari a K , invece che $K/2$



Convertitore A/D (cont.)

- Handshake soc/eoc con l'interfaccia di conversione

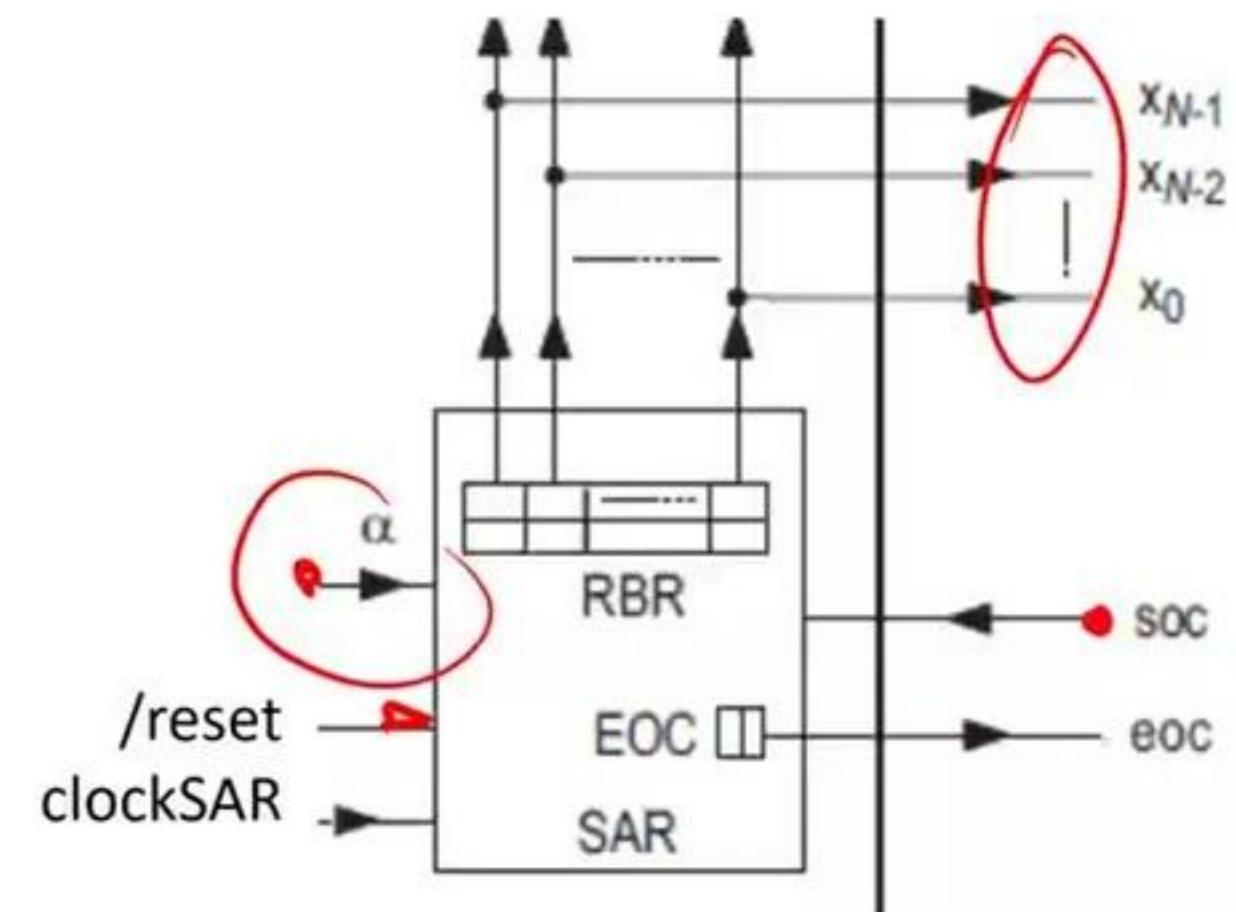


```

module SAR(eoc,x7_x0,soc,alpha,clockSAR,reset_);
  input clockSAR,reset_;
  input soc,alpha;
  output eoc;
  output[7:0] x7_x0;
  reg EOC; assign eoc=EOC;
  reg[7:0] RBR; assign x7_x0=RBR;
  reg[3:0] STAR;
  parameter S0=0,S1=1,S2=2,S3=3,S4=4,[...],S9=9,S10=10;
  always @(reset_==0) #1 begin EOC<=1; STAR<=S0; end
  always @(posedge clockSAR) if (reset_==1) #3
    casex(STAR)
      S0: begin EOC<=1; STAR<=(soc==0)?S0:S1; end
      S1: begin RBR<='B10000000; EOC<=0; STAR<=S2; end
      S2: begin RBR<={alpha,'B1000000}; STAR<=S3; end
      S3: begin RBR<={RBR[7],alpha,'B100000}; STAR<=S4; end
      S4: begin RBR<={RBR[7:6],alpha,'B10000}; STAR<=S5; end
      S5: begin RBR<={RBR[7:5],alpha,'B1000}; STAR<=S6; end
      [...]
      S9: begin RBR<={RBR[7:1],alpha}; STAR<=S10; end
      S10: begin EOC<=(soc==1)?0:1; STAR<=(soc==1)?S10:S0; end
    endcase
  endmodule

```

Count ↑ ->



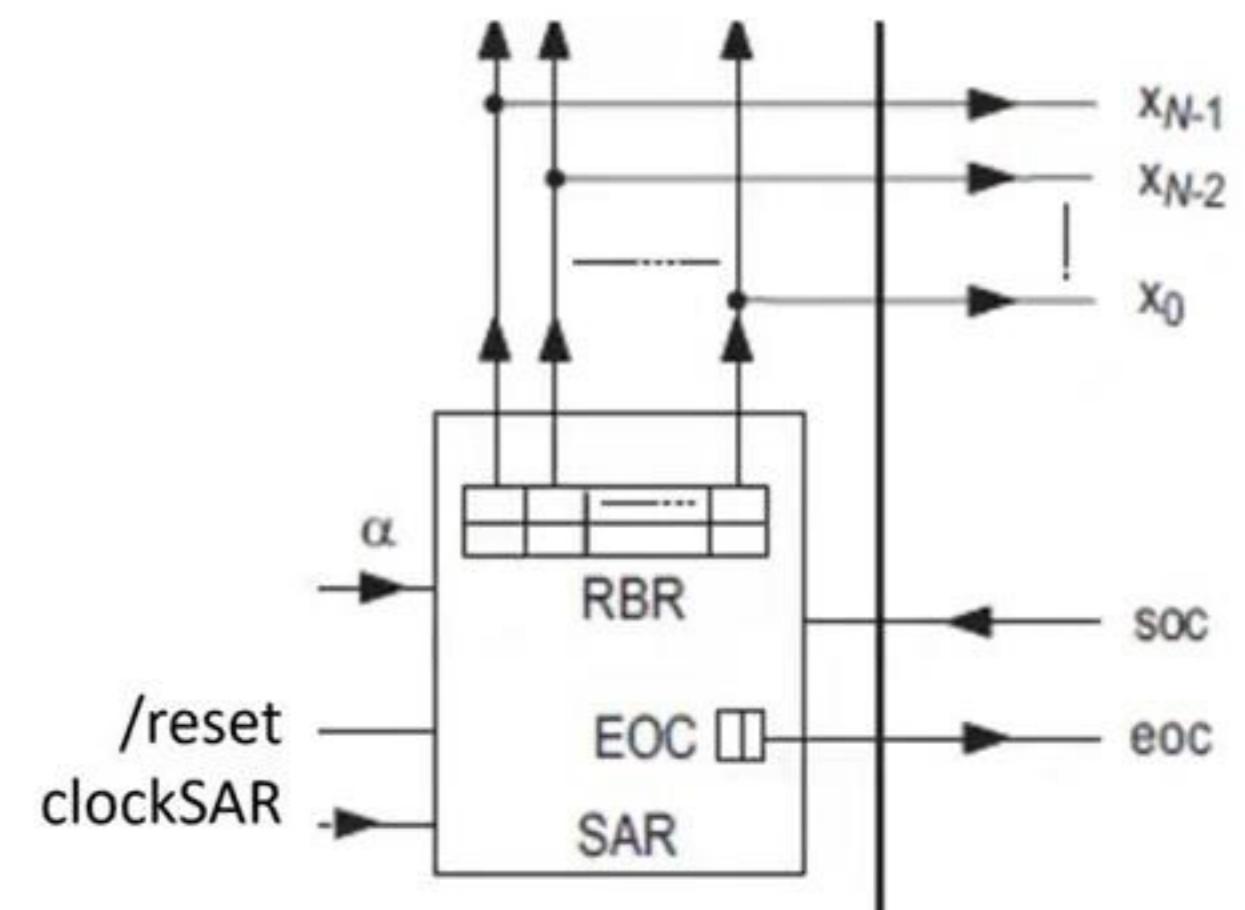
RBR, α, COUNT

```

module SAR(eoc, x7_x0, soc, alpha, clockSAR, reset_);
  input clockSAR, reset_;
  input soc, alpha;
  output eoc;
  output[7:0] x7_x0;
  reg EOC;   assign eoc=EOC;
  reg[7:0] RBR; assign x7_x0=RBR;
  reg[3:0] STAR;
  reg[2:0] COUNT;
  parameter S0=0, S1=1, S2=2, S3=3;
  always @ (reset_==0) #1 begin EOC<=1; COUNT<=7; STAR<=S0; end
  always @ (posedge clockSAR) if (reset_==1) #3
    casex (STAR)
      S0: begin EOC<=1; STAR<=(soc==0)?S0:S1; end
      S1: begin RBR<='B10000000; EOC<=0; STAR<=S2; end
      → S2: begin RBR<=nuovobyte(RBR, alpha, COUNT); COUNT<=COUNT-1;
            STAR<=(COUNT==0)?S3:S2; end
      S3: begin EOC<=(soc==1)?0:1; STAR<=(soc==1)?S3:S0; end
    endcase

    function [7:0] nuovobyte;
      [...]
    endfunction
  endmodule

```



RBR

cos

loc

nv

```
S2: begin RBR<=nuovobyte (RBR, alpha, COUNT) ; COUNT<=COUNT-1;  
      STAR<=(COUNT==0) ?S3:S2; end
```

```
function [7:0] nuovobyte;  
  input [7:0] vecchiobyte;  
  input alpha;  
  input [2:0] posizione;  
casex (posizione)  
  7: nuovobyte={alpha,'B1000000};  
  6: nuovobyte={vecchiobyte[7], alpha,'B100000};  
  5: nuovobyte={vecchiobyte[7:6],alpha,'B10000};  
  4: nuovobyte={vecchiobyte[7:5],alpha,'B1000};  
  3: nuovobyte={vecchiobyte[7:4],alpha,'B100};  
  2: nuovobyte={vecchiobyte[7:3],alpha,'B10};  
  1: nuovobyte={vecchiobyte[7:2],alpha,'B1};  
  0: nuovobyte={vecchiobyte[7:1],alpha };  
endcase  
endfunction
```

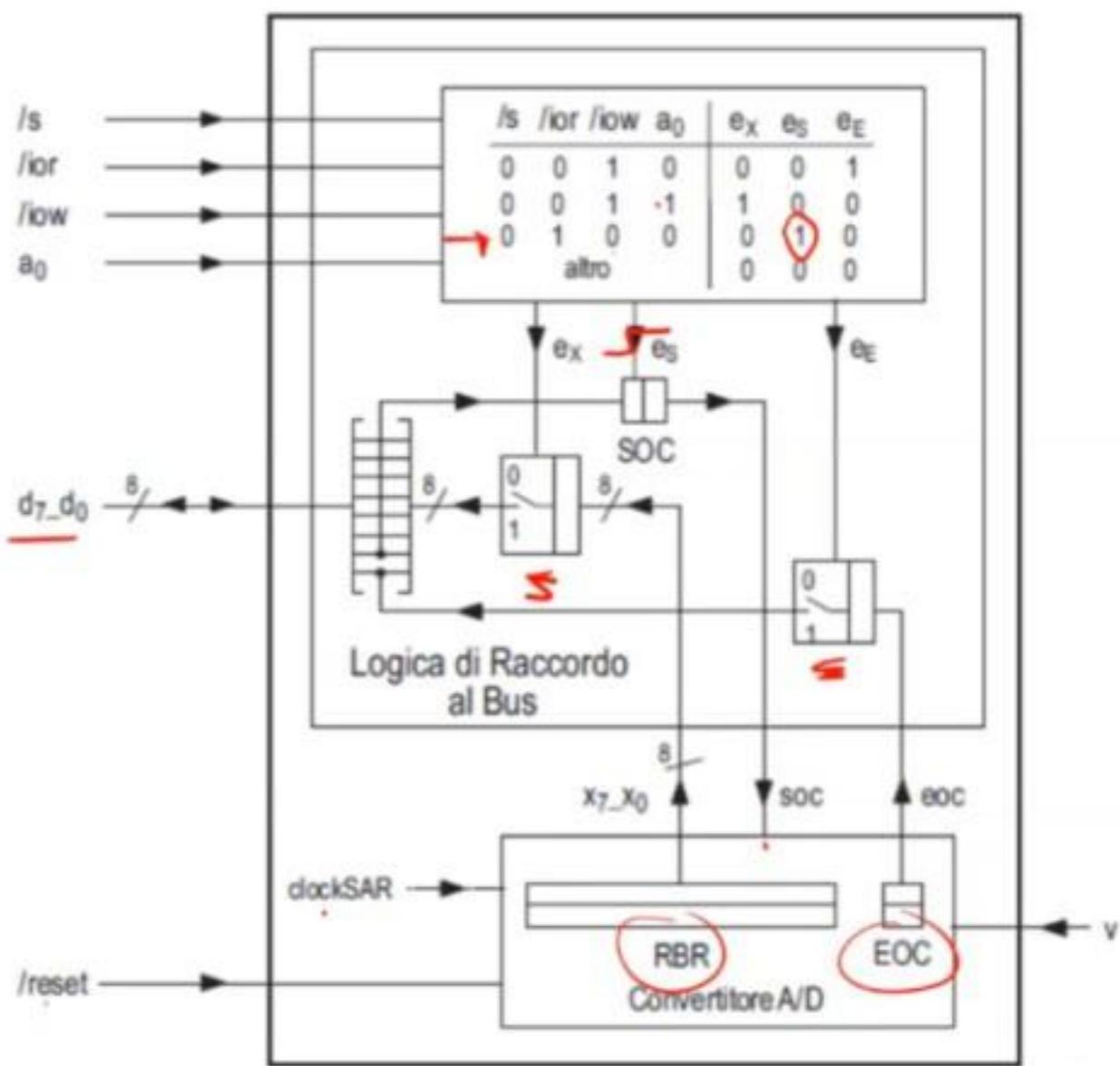
Interfaccia di conversione A/D

- Interfaccia parallela con handshake
- Il processore deve poter
 - impostare il bit SOC (accesso in scrittura a RCR)
 - Leggere il flag EOC (accesso in lettura a RSR)
- RSR e RCR mappati sullo stesso indirizzo interno



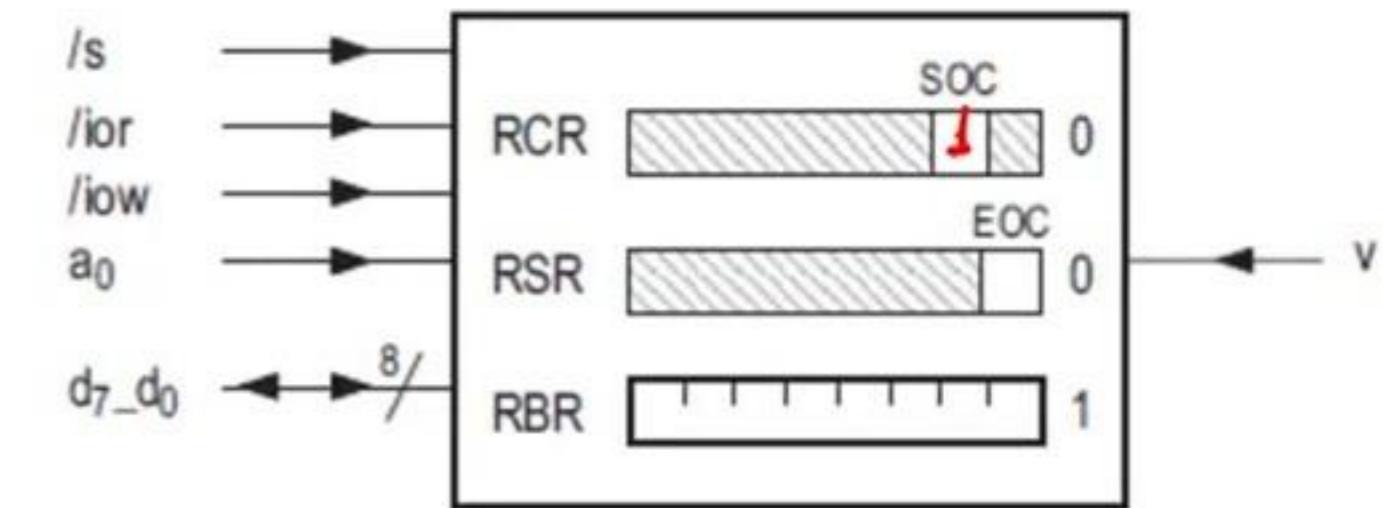
Interfaccia di conversione A/D (cont.)

- Logica combinatoria
 - per attivare le tri-state
 - Per memorizzare SOC
- Convertitore A/D di tipo SAR



Sottoprogramma per la conversione A/D

```
MOV $0x01, %AL  
OUT %AL, RCR_offset # SOC=1  
test1: [IN RSR_offset, %AL  
        AND $0x01, %AL  
        JNZ test1] # attendi EOC=0  
           [MOV $0x00, %AL  
            OUT %AL, RCR_offset] # SOC=0  
test2: [IN RSR_offset, %AL  
        AND $0x01, %AL  
        JZ test2] # attendi EOC=1  
           [IN RBR_offset, %AL] # prelievo dato convertito  
RET
```



Sottoprogramma per la conversione A/D

```
byte acquisizione( ) {  
    #define RCR_offset ...  
    #define RSR_offset RCR_offset  
    #define RBR_offset ...  
    byte tmp;  
  
    //Attiva la conversione immettendo 1 in SOC  
    outport(RCR_offset, 0x02);  
  
    //Attende che il contenuto di EOC vada a 0 e quindi immette 0 in SOC  
    do {tmp=import(RSR_offset)&0x01;} while (tmp!=0x00);  
    outport(RCR_offset, 0x00);  
  
    //Attende che il contenuto di EOC vada a 1  
    do {tmp=import(RSR_offset)&0x01;} while (tmp==0x00);  
  
    //Ritorna il risultato della conversione  
    return import(RBR_offset);  
}
```

SW

HW

