DISPENSA DOMANDE ORALI RETI LOGICHE

(aggiornata al 2015)

Questa dispensa nasce con lo scopo di aiutare tutti gli studenti a preparare al meglio l'orale di reti logiche. In questa dispensa sono contenute quasi tutte le domande orali più frequenti (ogni tanto il professore cambia domande) e non è intesa come sostituto del materiale fornito dai docenti bensì come supplemento allo studio.
Auguro a tutti di passarlo e se doveste riscontrare errori o mancanze di argomenti sentitevi pure liberi di modificarla e succsivamente di caricarla su
www.notestack.it

Un consiglio per l'orale: cercate di memorizzare meglio i circuiti perché sono fondamentali il professore guarda molto quello, più schematici siete e meglio è. Preferisce persone sintetiche a persone discorsive, quindi più disegni e meno discorsi.

Definire correttamente: Mintermine,Implicanti, implicanti principali, implicanti essenziali, implicante assolutamente eliminabile,implicante semplicemente eliminabile:

Data una legge F che caratterizza una rete combinatoria con una variabile di uscita, si consideri la sua forma canonica di tipo SP, si definiscono:

Mintermine: il prodotto di tutte le variabili d'ingresso dirette o negate che compare nella forma canonica SP, un mintermine riconosce uno stato d'ingresso

Implicante: è il prodotto di alcune variabili d'ingresso dirette o negate che compare in una forma di tipo SP, ottenuta per fusione di mintermini o implicanti che differiscono per una variabile d'ingresso diretta o negata. Cioè applicando le due identità:

$\alpha x + \alpha \overline{x} = \alpha x + \alpha\overline{x} + \alpha$	Espansione per fusione
$\beta+\beta = \beta$	Eliminazione dei doppioni

Implicante principale: è un implicante che non contribuisce a generare per fusione nessun altro implicante, dopo aver applicato la seguente semplificazione nella forma SP ottenuta nella lista degli implicanti:

$$\alpha x + \alpha = \alpha$$

Implicante principale essenziale: è un implicante principale che è l'unico a riconoscere uno stato d'ingresso riconosciuto dalla rete

Implicante principale assolutamente eliminabile: è un implicante principale che riconosce soltanto stati d'ingresso che sono già riconosciuti da implicanti principali essenziali

Implicante principale semplicemente eliminabile: è un implicante che riconosce soltanto stati d'ingresso riconosciuti da altri implicanti ma almeno uno non è riconosciuto da implicanti principali essenziali

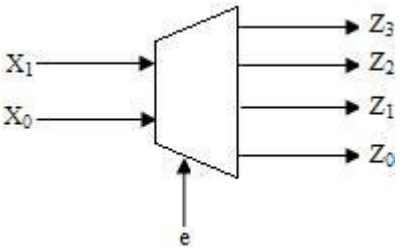
Lista di copertura: ogni lista di implicanti che sommati logicamente costituiscono una forma di tipo SP per F

Lista di copertura irridondante: è detta irridondante se non è più una lista di copertura qualora venga privata anche di uno solo dei suoi elementi.

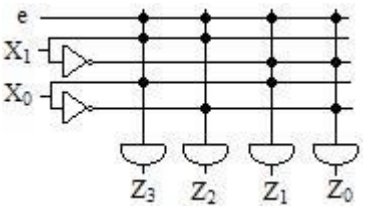
Forma canonica di una funzione booleana a N ingressi/variabili: Prendendo in considerazione una rete combinatoria con N variabili di ingresso e una sola variabile di uscita caratterizzata dalla legge F, si definisce forma canonica l'espressione algebrica ottenuta a partire dall'espansione di Shannon e applicando in modo esaustivo le seguenti identità:

- $0*\alpha = 0$
- $1*\alpha = \alpha$
- $0+\alpha=\alpha$

Struttura del decoder 2 to 4 con abilitazione



e	x_0	x_1	Z_3	Z_2	Z_1	Z_0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	-	-	0	0	0	0



$Z_3 = X_1 X_0\ e$ — assign $Z_3 = e\ \&\ (X_1\ \&\ X_0)$;

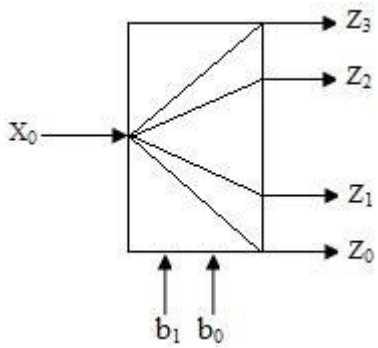
$Z_2 = \overline{X_0} X_1\ e$ — assign $Z_2 = e\ \&\ (X_1\ \&\ \sim X_0)$;

$Z_1 = \overline{X_1} X_0\ e$ — assign $Z_1 = e\ \&\ (\sim X_1\ \&\ X_0)$;

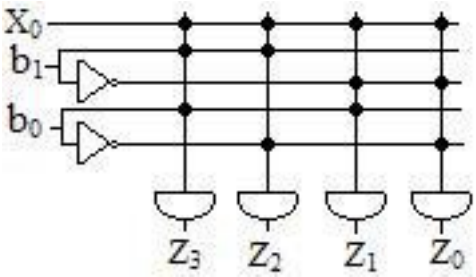
$Z_0 = \overline{X_1}\,\overline{X_0}\ e$ — assign $Z_0 = e\ \&\ (\sim X_1\ \&\ \sim X_0)$;

Quando l'enabler è a 0 tutte le uscite sono nulle, quando l'enabler è a 1 abilita il decoder a funzionare, cioè abilita un uscita e le altre le tiene disattive.

Demultiplexer 1 to 4: descrizione e sintesi:

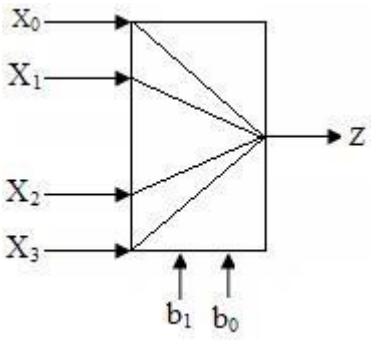



```
assign Z3 = ( [b1,b0] == 'B11) ? X0 : 0 ;
assign Z2 = ( [b1,b0] == 'B10) ? X0 : 0 ;
assign Z1 = ( [b1,b0] == 'B01) ? X0 : 0 ;
assign Z0 = ( [b1,b0] == 'B00) ? X0 : 0 ;
```



Giustamente la sintesi è uguale al decoder in quanto sono lo stesso componente, ma rovesciato

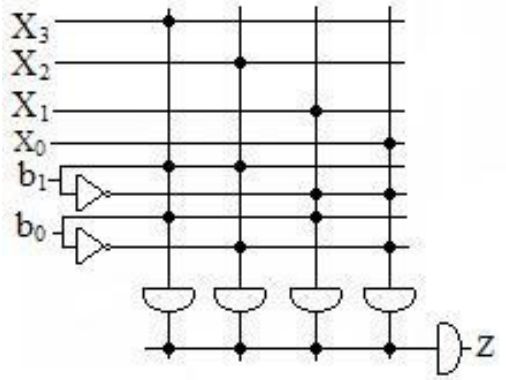
Multiplexer 4 to 1, simbolo descrizione della legge che lo caratterizza e struttura a porte:



assign: Z = ([b 1,b0] == 'B00) ? X 0: (
[b1,b0] == 'B01) ? X 1: (
[b1,b0] == 'B10) ? X2:
X3 ;

$$Z = \overline{b_1}\,\overline{b_0}\,X_0 + \overline{b_1}\,b_0\,X_1 + b_1\overline{b_0}\,X_2 + b_1 b_0\,X_3$$

Quindi la sintesi:



Sintetizzare a porte NOR

X_1X_0 \ X_3X_2	00	01	11	10
00	0	0	1	1
01	1	-	-	0
11	1	-	-	0
10	1	1	0	1

Z

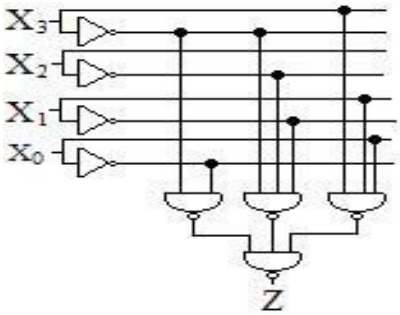
Passo a $\bar{Z}$

X_1X_0 \ X_3X_2	00	01	11	10
00	1	1	0	0
01	0	-	-	1
11	0	-	-	1
10	0	0	1	0

$\bar{Z}$

$$\bar{Z} = X_3X_0 + X_3X_2X_1 + \bar{X}_3\bar{X}_0\bar{X}_1$$

$$Z = \overline{\overline{X_3X_0} + \overline{X_3X_2X_1} + \overline{\bar{X}_3\bar{X}_0\bar{X}_1}} = \overline{\overline{X_3X_0} + \overline{X_3X_2X_1} + \overline{X_3X_0X_1}}$$



Leggi di De Mogan per N variabili e dimostrazione

1) $\overline{X_{n-1} + X_{n-2} + \ldots\ldots X_0} = \overline{X_{n-1}} \cdot \overline{X_{n-2}} \cdot \ldots\ldots \overline{X_0}$

Dimostrazione:

caso base, mediante tabella di verità (N = 2)

X1	X0	$\overline{X1 + X0}$	$\overline{X1} \cdot \overline{X0}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

ora supponiamo che valga per N-1 variabili e vediamo cosa succede

$$\overline{X_{n-1}} \cdot \overline{X_{n-2}} \cdot \ldots\ldots \cdot \overline{X_0} = \overline{X_{n-1}} \cdot (\overline{X_{n-2}} \cdot \ldots\ldots \cdot \overline{X_0}) = \overline{X_{n-1}} \cdot (\overline{X_{n-2} + \ldots + X_0}) = \overline{X_{n-1}} \cdot \overline{\alpha} = \overline{X_{n-1} + \alpha} = \overline{X_{n-1} + X_{n-2} + \ldots + X_0}$$

quindi vale per un qualsiasi N arbitrario

2) $\overline{X_{n-1} \cdot X_{n-2} \cdot \ldots\ldots \cdot X_0} = \overline{X}_{n-1} + \overline{X}_{n-2} + \ldots\ldots \overline{X}_0$

Dimostrazione:

caso base, mediante tabella di verità (N = 2)

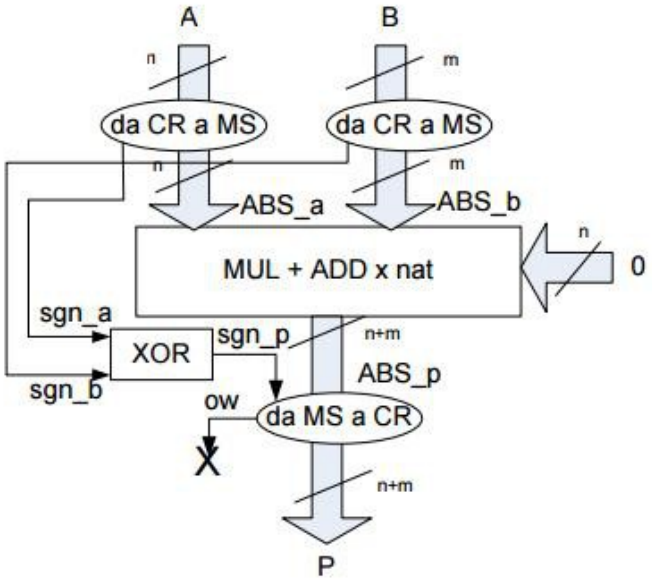
X1	X0	X1 + X0	X1 X0
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

ora supponiamo che valga per N-1 variabili e vediamo cosa succede

$$\overline{X}_{n-1} + \overline{X}_{n-2} + \ldots + \overline{X}_0 = \overline{X_{n-1}} + (\overline{X_{n-2}} + \ldots + \overline{X_0}) = \overline{X_{n-1}} + (\overline{X_{n-2} \cdot \ldots \cdot X_0}) = \overline{X_{n-1}} + \overline{\alpha} = \overline{X_{n-1} \cdot \alpha} = \overline{X_{n-1} \cdot X_{n-2} \cdot \ldots \cdot X_0}$$

Si lavori in base β in complemento. L'intero *a* è rappresentato dal naturale A su N cifre, L'intero *b* è rappresentato dal naturale B su M cifre. L'intero *p=a*b* è rappresentato dal naturale P.
Scrivere e giustificare l'algoritmo per trovare P su … cifre.

$$p = \begin{cases} ABS(a)\cdot ABS(b) & a,b \text{ concordi} \\ -ABS(a)\cdot ABS(b) & a,b \text{ discordi} \end{cases}$$

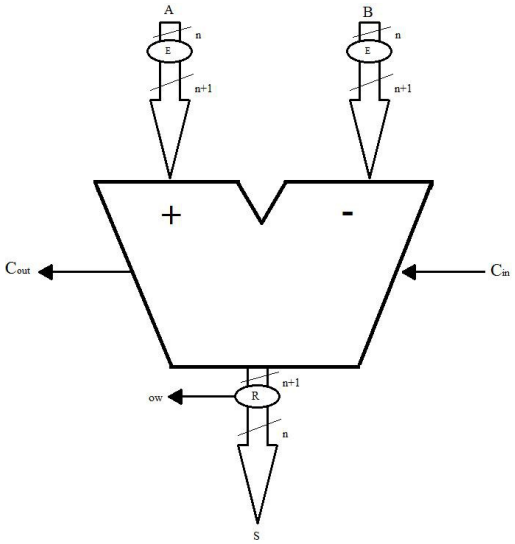


L'intero *a* è rappresentato dal naturale A in base β su N cifre in complemento. Algoritmo e dimostrazione per trovare X=-*a* precisando in quale passaggio la dimostrazione (non altro) NON è valida se *a=-β^n/2*.

$$B \overset{(*)}{=} |-a|_{\beta^n} \quad \text{<----NON valida se a = -(B\^n)/2}$$
$$= \left| |-1|_{\beta^n} \cdot |a|_{\beta^n} \right|_{\beta^n} = \left| (\beta^n - 1)\cdot A \right|_{\beta^n}$$
$$= \left| \beta^n \cdot A - A \right|_{\beta^n} = |-A|_{\beta^n}$$
$$= \left| -\beta^n + 1 + \overline{A} \right|_{\beta^n} = \left| 1 + \overline{A} \right|_{\beta^n}$$

Uso il complemento: $\overline{A} = \beta^n - 1 - A$

Circuito che trova S↔*s*=*a*+*b* su n cifre con *a*,*b* interi su n cifre in base β



Circuito di *overflow* per interi

ow= $c_n \oplus c_{n-1}$

Dove c_n e c_{n-1} sono i riporti uscenti degli ultimi due full adder.

Dato un intero *a*, circuito che passa da MS→CR e viceversa (non trascurare i circuiti per l'overflow)

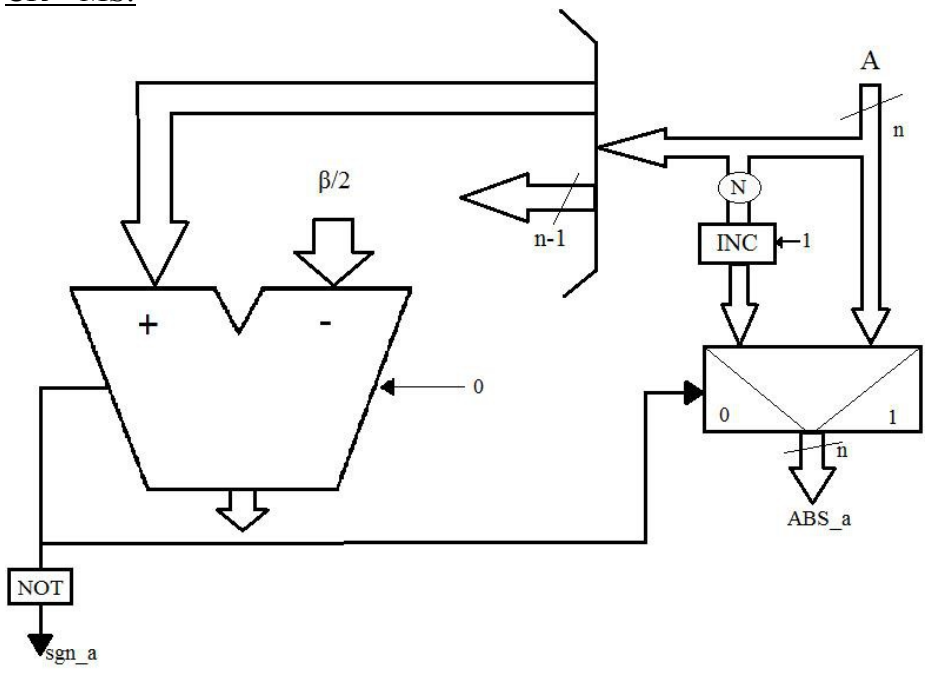
MS→CR:

$$A = |a|_{\beta^n} = \begin{cases} ABS\_a & \text{sgn}\_a = 0 \\ \left|\overline{ABS\_a} + 1\right|_{\beta^n} & \text{sgn}\_a = 1 \end{cases}$$

Per quanto riguarda l'overflow=1 esistono due condizioni:

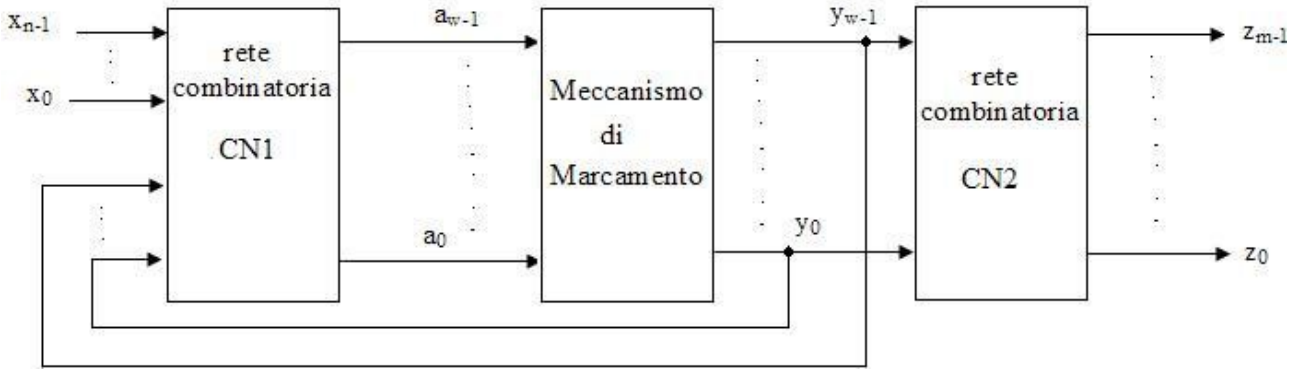
- ABS_a>$\beta^n/2$
- ABS_a=$\beta^n/2$ **and** sgn_a=0 (questa si fa con un comparatore)

CR→MS:



Modello per le Reti Sequenziali Asincrone (RSA)

Modello Strutturale:



1. Un insieme di **n** variabili logiche di ingresso.
2. Un insieme di **m** variabili logiche di uscita.
3. Un **meccanismo di marcatura** che ad ogni istante marca uno stato interno presente $S \equiv \{S_0,\ldots,S_{w-1}\}$.
4. Una legge **A:X×S→S** che aggiorna lo stato interno.
5. Una legge **B:S→Z** che decide lo stato di uscita in base allo stato interno.
6. La rete si evolve **esclusivamente** al variare degli ingressi.

Regole per un corretto funzionamento:

- La rete CN1 deve essere pilotata in modo fondamentale, cioè il suo stato d'ingresso può essere cambiato solo quando essa è a regime, e senza transizioni multiple in ingresso a livello delle variabili $X_{n-1},\ldots,X_0$.
- La rete CN1 non deve presentare alee statiche del I° ordine.
- Deve essere fatta una codifica degli stati interni che non preveda corse, cioè due stati interni temporalmente successivi devono essere adiacenti ovvero non devono differire per più di un bit.
- Lo stato interno successivo deve essere marcato solo quando CN1 è a regime. Detto T_{CN1} il tempo di attraversamento di CN1 si ha che il meccanismo di marcatura deve adeguare lo stato interno presente allo stato interno successivo solo dopo un tempo $Tmark \geq T_{CN1}$. Tale limitazione può essere rimossa se non esistono alee essenziali, ovvero se quando viene presentato il nuovo stato interno, invitando il meccanismo di marcatura a variare una variabile di stato, allora CN1 è a regime o sono a regime le porzioni di CN1 interessate alla variazione di quella variabile di stato. Possiamo inoltre legare il tempo di permanenza dello stato di ingresso T ai due tempi T_{CN1} e T_{mark}

$$\mathbf{T \geq T_{CN1} + i(T_{mark} + T_{CN1})}$$

dove i = numero di marcature necessarie prima di giungere in una situazione stabile. In conclusione per **pilotaggio fondamentale si intende, cambiare lo stato di ingresso solo quando si è in una situazione di stabilità.**

REGOLE DI PROGETTO	REGOLE DI PILOTAGGIO
CN1 priva di alee del I° ordine	Pilotaggio in modo fondamentale.
Inserire un meccanismo di marcatura tale che: • $T_{mark}=0$ se la tabella di flusso è normale e priva di alee essenziali. • $T_{mark}=T_{CN1}$ se la tabella di flusso è normale ma ha alee essenziali. • $T_{mark}\geq T_{CN1}$ altrimenti	NO transizioni multiple in ingresso.
Codificare gli stati interni in maniera da evitare corse delle variabili di stato (stati consecutivi hanno codifiche adiacenti).	Mantenere stabile uno stato d'ingresso per un tempo $T \geq T_{CN1} + i(T_{mark}+T_{CN1})$.

Definizione di Rete Normale

Una rete è <u>normale</u> se partendo da uno stato stabile e variando lo stato d'ingresso, **si raggiunge sempre uno stato stabile in al più un passo**.

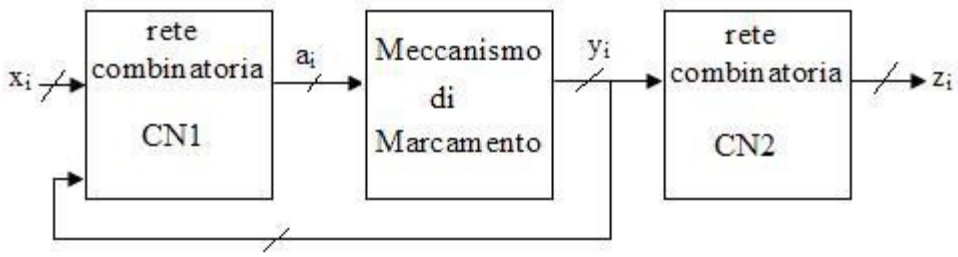
Il non problema delle transizioni multiple in ingresso per il latch SR.

Basta osservare la tabella di verità per capirlo. Se, per esempio, l'ingresso attuale è 01 avrò Q=0. Lo stato successivo supponiamo che sia 10. La contemporaneità perfetta non esiste quindi nel breve intervallo in cui avviene il transitorio il latch può ricevere o 00 oppure 11.
Nel primo caso non cambia nulla in quanto per 00 conserva. Nel secondo caso l'uscita potrebbe valere 0 per un brevissimo istante.

S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	-

Risoluzione della rete sequenziale asincrona descritta mediante la sua tabella di flusso

Modello strutturale:



Il meccanismo di marcatura può essere costituito da:

W ritardi , dove W è il numero di variabili di stato (*la barriera di ritardi serve nel caso in cui siano presenti alee essenziali, qualora non sussistesse il problema, si potrebbe eliminare, almeno per quelli relativi alle variabili per cui non si verifica questo fenomeno*.)

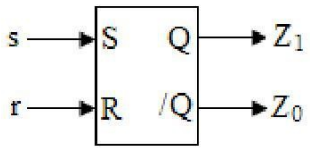
W Flip-Flop SR (*se il ritardo introdotto dal Flip-Flop non è sufficiente, si devono aggiungere dei ritardi ulteriori*)

Data la tabella di flusso prima di tutto si decide quale codifica adottare per gli stati interni in modo da non introdurre il fenomeno delle corse delle variabili di stato. Si crea poi la tabella delle transizioni, che viene ottenuta sostituendo nella tabella di flusso la codifica al posto degli stati interni. Poi si hanno i seguenti casi:

- La tabella delle transizioni fornisce direttamente lo stato interno successivo [a_i], una volta sintetizzata in modo da evitare la presenza di alee del primo ordine. Le uscite si ottengono sintetizzando una rete combinatoria che produce in funzione dello stato interno marcato [y_i].
- La tabella delle transizioni fornisce lo stato interno successivo che deve essere ottenuto stimolando in modo opportuno la barriera di Flip-Flop posto come meccanismo di marcatura. Per ottenere la sintesi di CN1 si deve combinare la tabella della transizioni con la tabella di applicazione del Flip-Flop-SR . La rete CN2 non cambia, in quanto non cambia la codifica degli stati interni.

Come si vede se esistono alee essenziali?
Nell'istante in cui CN1 presenta alla sottorete sequenziale un nuovo stato d'ingresso, invitandola ad aggiornare il valore di una nuova variabile di stato, allora CN1 è a regime, oppure sono a regime quelle porzioni di CN1 sensibili alla transizione della variabile di stato in questione.

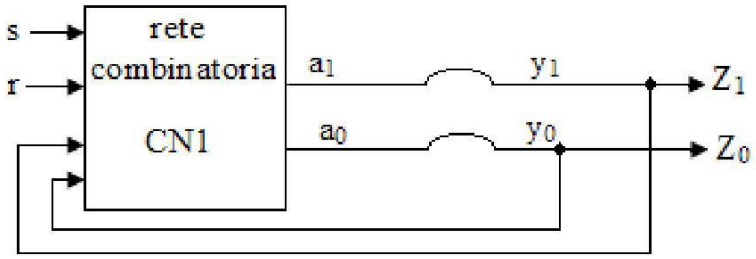
Descrizione e sintesi del Flip-Flop SR



s r	00	01	11	10	Z1	Z0
S0	(S0)	(S0)	-	S1	0	1
S1	(S1)	S0	-	(S1)	1	0

Quindi CN2 è un cortocircuito

Però cosi abbiamo il problema delle corse delle variabili, allora si usa uno stato ponte SA.



Stato	y_1 y_0
S0	01
S1	10
SA	11
SB	00

Tabella di flusso modificata

Attenti alle codifiche degli stati!

s r	00	01	11	10	Z1	Z0
00	S0	S0	-	SA	0	1
01	S1	SA	-	S1	1	0
11	-	S0	-	S1	1	1
10	-	-	-	-	0	0

Tabella delle transizioni

s r	00	01	11	10
00	--	--	--	--
01	01	01	--	11
11	--	01	--	10
10	10	11	--	10

a_1 a_0

Da cui

y_1 y_0 \ s r	00	01	11	10
00	-	-	-	-
01	1	1	-	1
11	-	1	-	0
10	0	1	-	0

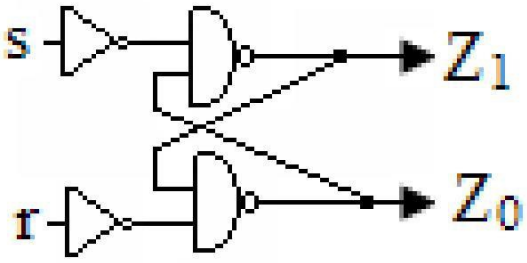
a_0

$$a_1 = s + \bar{y}_0 = \overline{\overline{s + \bar{y}_0}} = \overline{\bar{s} \cdot y_0}$$

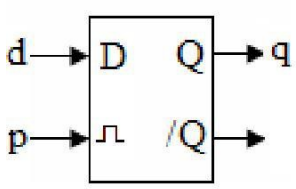
$$a_0 = r + \bar{y}_1 = \overline{\overline{r + \bar{y}_1}} = \overline{\bar{r} \cdot y_1}$$

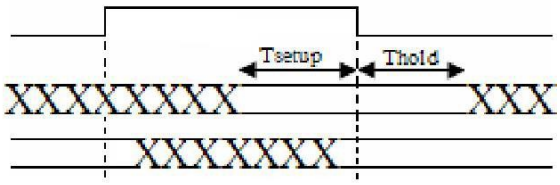
y_1 y_0 \ s r	00	01	11	10
00	-	-	-	-
01	0	0	-	1
11	-	0	-	1
10	1	1	-	1

a_1



Temporizzazione, descrizione e sintesi del flip-Flop D-latch





È una rete sequenziale asincrona che ha due variabili di ingresso d e p ed una variabile di uscita q e si evolve in accordo alle seguenti specifiche: quando il valore della variabile *p* è a 1, il flip-flop è in una fase di *campionamento* ,in cui adegua il valore della variabile di uscita *q* al valore della variabile di ingresso *d*; quando il valore della variabile di *p* è 0, il flip-flop è in una fase di *conservazione*, in cui mantiene costante il valore della variabile di uscita *q*. Per evitare transizioni multiple in ingresso ed assicurare un corretto campionamento finale del valore della variabile *d*, occorre, in accordo alle regole del corretto pilotaggio delle reti sequenziali asincrone, che il valore della variabile *d* rimanga costante almeno per un piccolo intervallo di tempo prima (*Tsetup*) e per un piccolo intervallo di tempo dopo (*Thold*)rispetto al verificarsi della transazione da 1 a 0 della variabile *p*.

Tabella di flusso

	p=0		p=1		q
	0	1	0	1	
S0	S0	S0	S0	S1	0
S1	S1	S1	S0	S1	1

Codifica stati interni

Stato	y_0
S0	0
S1	1

Tabella delle transizioni

y_0 \ dp	00	01	11	10	q
0	0	0	1	0	0
1	1	0	1	1	1

Modello strutturale

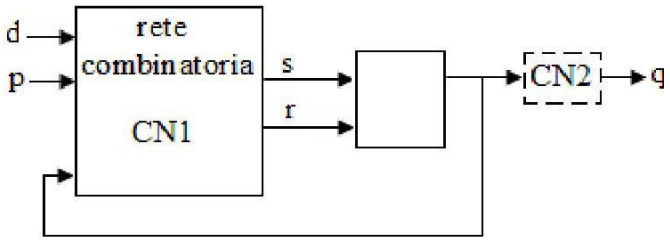
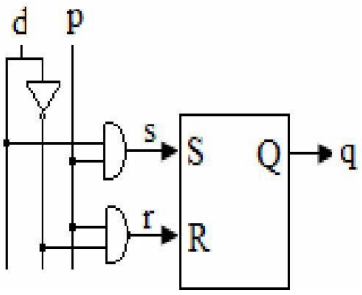
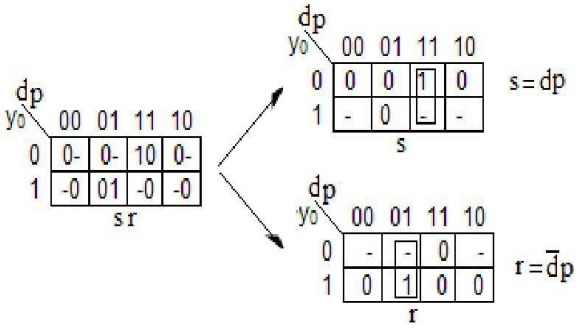


Tabella di applicazione del flip-flop- SR

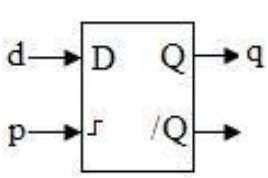
ho	voglio	s	r
0	0	0	-
0	1	1	0
1	0	0	1
1	1	-	0

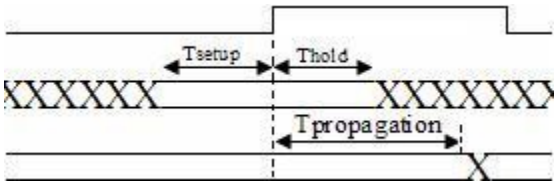
Combiniamo la tabella di applicazione del flip-flop SR con la tabella delle Transizioni per stimolare correttamente il flip-flop SR





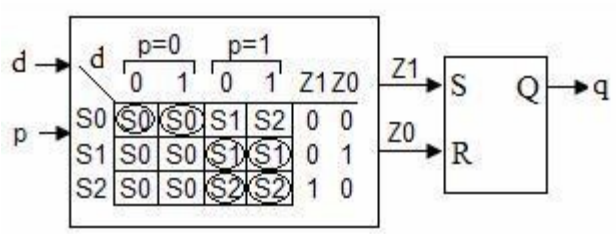
Temporizzazione, descrizione e sintesi del flip-Flop D-positive-edge-triggered





È una rete sequenziale asincrona che ha due variabili di ingresso d e p ed una variabile di uscita q e si evolve in accordo alle seguenti specifiche: quando la variabile p transisce da 0 a 1 il flip-flop entra in una brevissima fase in cui campiona il valore della variabile d e da cui esce automaticamente ed indipendentemente dal fatto che la variabile p continui a rimanere a 1 o torni a 0. Il flip-flop mantiene inalterato il valore della variabile di uscita q sia quando p vale 0 sia quando, pur essendo p transita a 1, è in corso la fase di campionamento del flip-flop D-positive-edge-triggered adegua il valore della variabile di uscita q a quel valore della variabile di ingresso d che è stato campionato, dopo di che entra in una fase di completa inattività in cui mantiene costante il valore della variabile di uscita q e da cui esce soltanto alla successiva transazione da 0 a 1 di p.
Il brevissimo intervallo di tempo in cui il flip flop è nella fase di campionamento è detto Thold, il tempo che intercorre tra la transizione da 0 a 1 della variabile p e l'adeguamento della variabile di uscita q al valore della variabile di ingresso d che è stato campionato, è detto Tpropagation (ed è maggiore di Thold, stante il fatto che l'adeguamento viene fatto a campionamento concluso).Il flip-flop è quindi *non-trasfarente* , in quanto mentre campiona il valore della variabile di ingresso d mantiene costante il valore della variabile di uscita q al valore. Per evitare transizioni multiple in ingresso ed assicurare un corretto campionamento finale del valore della variabile *d*, occorre, in accordo alle regole del corretto pilotaggio delle reti sequenziali asincrone, che il valore della variabile *d* rimanga costante almeno per un piccolo intervallo di tempo prima (*Tsetup*) e per un piccolo intervallo di tempo dopo (*Thold*)rispetto al verificarsi della transazione da 1 a 0 della variabile *p*. Un corretto pilotaggio del flip-flop deve inoltre garantire che la variabile p rimanga a 1 per un tempo un po' maggiore di Thold. Il flip-flop deve poi essere progettato in modo da sopportare senza inconvenienti la non costanza del valore della variabile d a cavallo della transizione da 1 a 0 di p. Questo Flip-flop è usualmente realizzato con una coppia di reti:

i. Campionatore sul fronte in salita
ii. Ritardatore



Tpropagation > Thold

↓

Non trasparenza

Il campionatore sul fronte in salita si occupa di pilotare il flip-flop SR in modo che abbia l'uscita corretta, cioè si setta, si resetta o si conserva a seconda di cosa c'era in d durante il Thold.
Se prendiamo come codifica la seguente:

Stato	y_1 y_0
S0	00
S1	01
S2	10

Si può notare che ci sono corse tra 01 e 10 , ma non si va mai da S1 a S2 quindi non ci sono problemi.

Modello strutturale

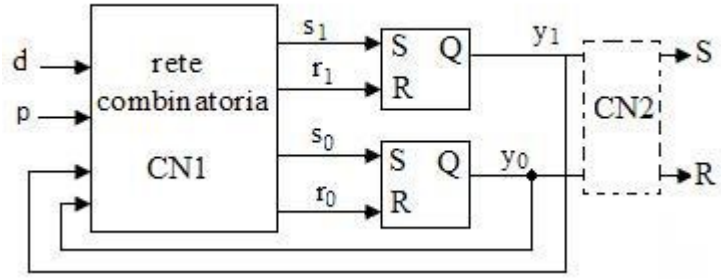


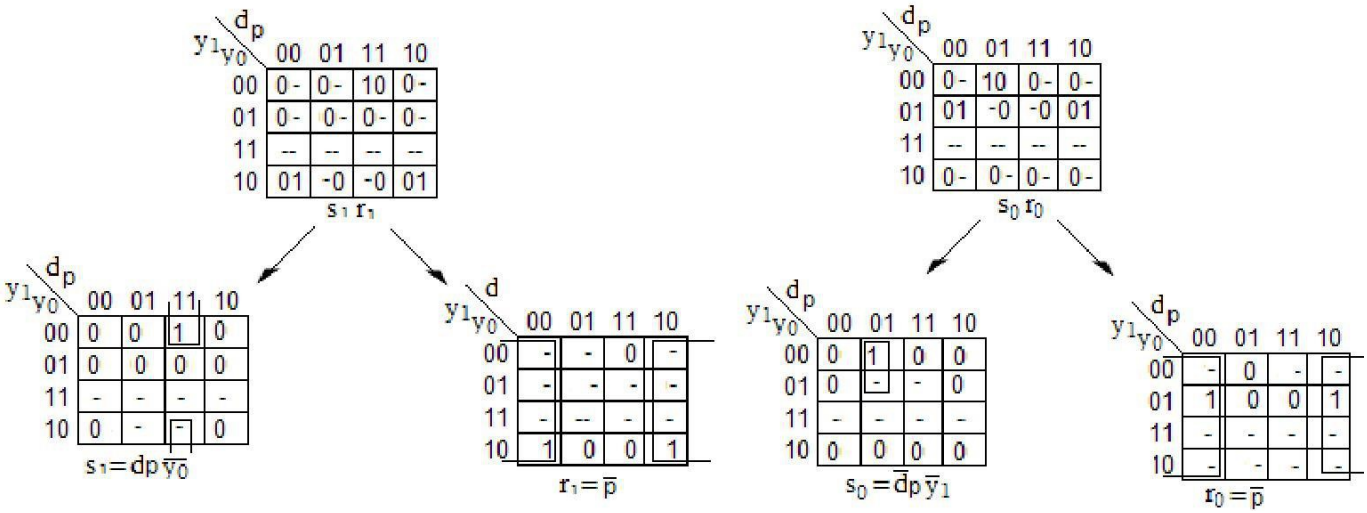
Tabella delle transizioni

y_1y_0 \ d_p	00	01	11	10
00	00	01	10	00
01	00	01	01	00
11	--	--	--	--
10	00	10	10	00

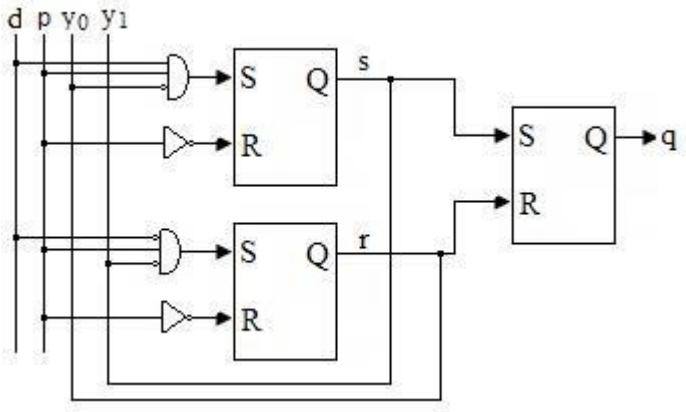
Tabella di applicazione del SR

ho	voglio	s	r
0	0	0	-
0	1	1	0
1	0	0	1
1	0	-	0

le combiniamo ed otteniamo.

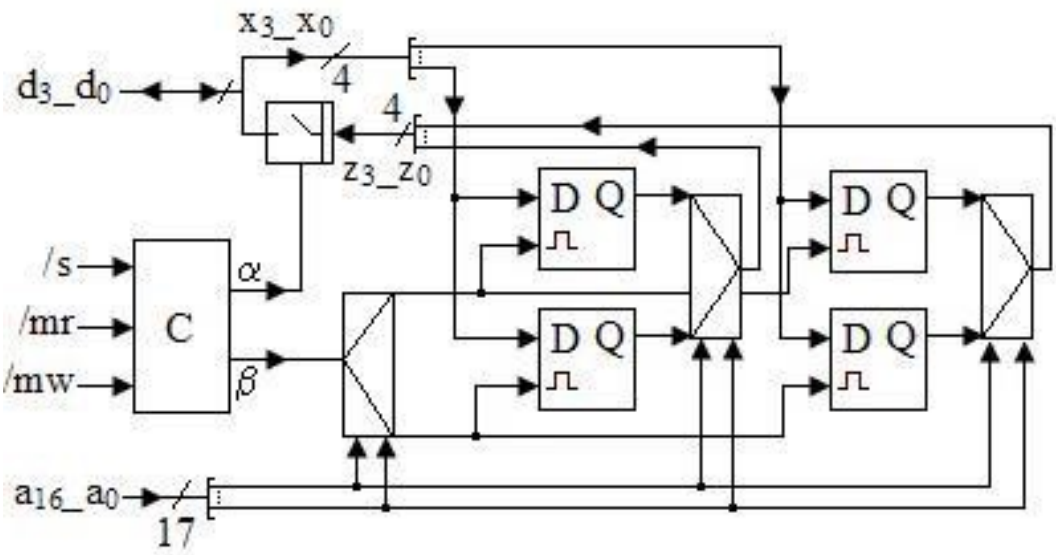


Ed infine il circuito



Struttura interna di una RAM statica e temporizzazione dei cicli di lettura e scrittura

Una memoria RAM statica può essere vista come una matrice di D-latch contornata da reti combinatorie che permettono l'accesso contemporaneo a tutti i D-latch costituenti una riga della matrice stessa. L'accesso ad una riga può essere o in scrittura (modifica) o in lettura (prelievo).



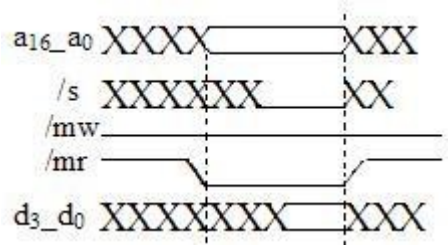
La rete combinatoria C ha la seguente tabella di verità:

/s	/mr	/mw	α	β
1	-	-	0	0
0	1	1	0	0
0	0	1	1	0
0	1	0	0	1
0	0	0	-	-

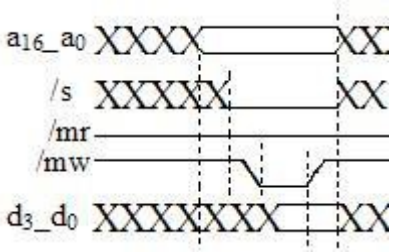
Poiché una memoria RAM statica è basata su flip-flop D-latch per garantire la trasparenza, cioè durante i cicli in scrittura,il valore delle variabili di ingresso non siano influenzate dal valore delle variabili di uscita, viene applicata una barriera di 3-state.

Vediamo ora i tipici diagrammi di temporizzazione relativi sia ad un ciclo di lettura che ad uno di scrittura.

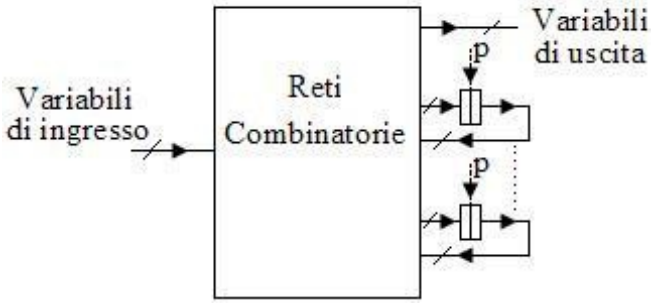
Ciclo di lettura



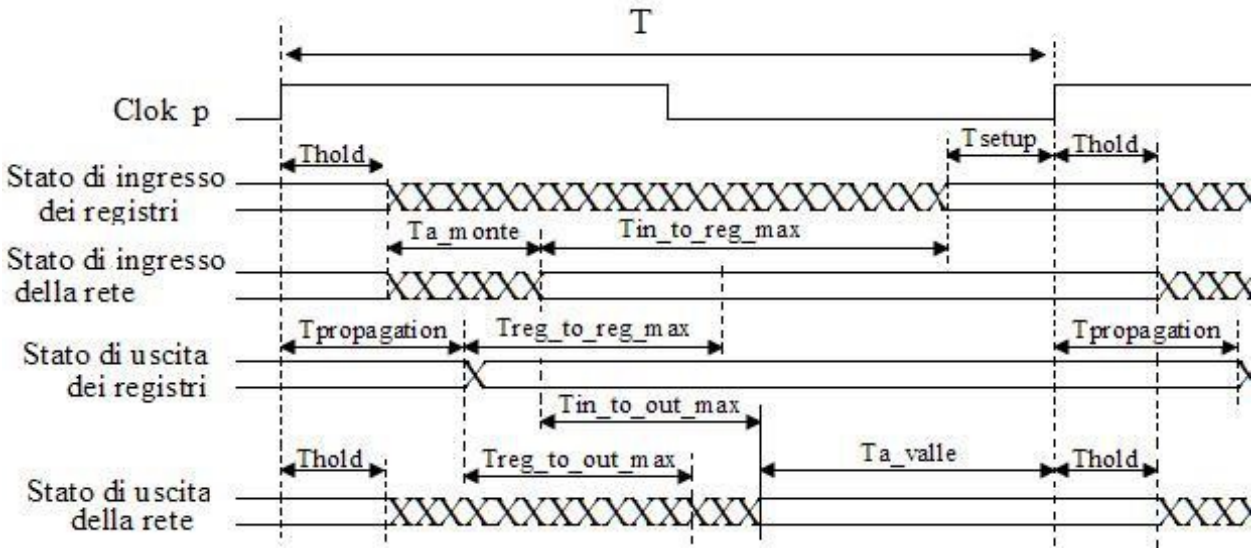
Ciclo di scrittura



Reti sequenziali sincronizzate complesse: struttura generale e temporizzazione



Poiché i registri sono non trasparenti possono essere accoppiati a reti combinatorie a formare reti sequenziali sincronizzate complesse. Le varie reti combinatorie possono essere connesse in qualsiasi modo, purché nel loro insieme costituiscano ancora una rete combinatoria,cioè non ci siano anelli che diano luogo a reti sequenziali asincrone che possano creare problemi



I registri sono reti non trasparenti; detti infatti $t_0,t_1,\ldots,t_n$ gli istanti in cui arrivano i segnali di sincronizzazione, sono verificate le due seguenti condizioni:

> Solo quelli stati di ingresso che sono presenti (e stabili) negli intervalli $(t_i\text{-}T_{setup}, t_i + T_{hold})$, per i = 0,1,…..,∞ vengono utilizzati, mentre quegli stati di ingresso che sono presenti al di fuori di questi intervalli vengono del tutto ignorati e sono completamente ininfluenti;
> gli istanti $t_i + t_{propagation}$ in cui inizia l'emissione del nuovo stato di uscita, sono esterni agli intervalli $(t_i\text{-}T_{setup}, t_i + T_{hold})$

Esaminando la rete, definiamo con:

1. $\mathbf{T_{in\_to\_reg}}$ il tempo massimo di attraversamento della più lunga catena di (sole) reti combinatorie che si può incontrare tra le variabili di ingresso della rete e l'ingresso di un registro
2. $\mathbf{T_{reg\_to\_reg}}$ (…) le variabili di uscita di un registro e le variabili di ingresso di un altro registro;
3. $\mathbf{T_{in\_to\_out}}$ (…) le variabili di ingresso e le variabili di uscita della rete;
4. $\mathbf{T_{reg\_to\_out}}$ (…) le variabili di uscita di un registro e le variabili di uscita della rete;

Se ne deduce che l'intervallo di tempo T che deve intercorrere tra due successivi segnali di sincronizzazione, deve soddisfare le seguenti condizioni:

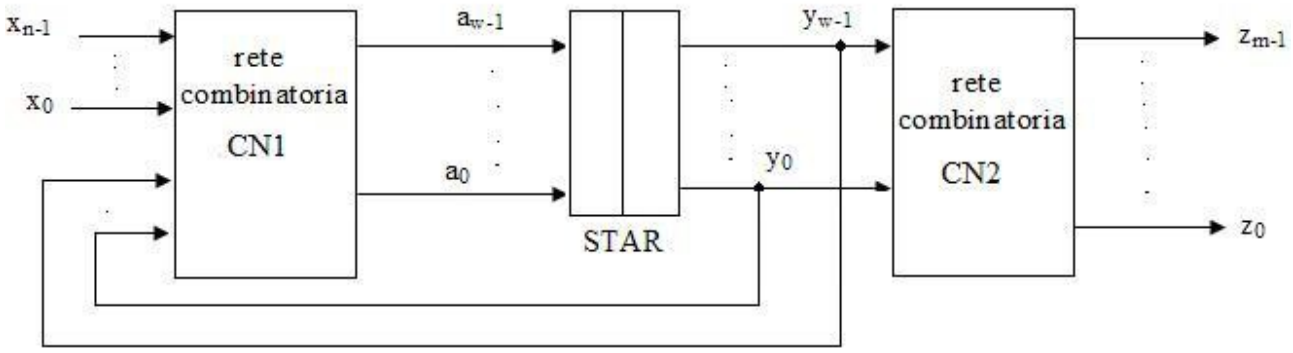
$$T \geq T_{hold} + T_{a\_monte} + T_{in\_to\_reg} + T_{setup}$$
$$T \geq T_{propagation} + T_{reg} + T_{reg\_to\_reg} + T_{setup}$$
$$T \geq T_{hold} + T_{a\_monte} + T_{in\_to\_out} + T_{a\_valle}$$
$$T \geq T_{propagation} + T_{reg} + T_{reg\_to\_out} + T_{a\_valle}$$

Le prime due condizioni garantiscono che lo stato delle variabili di ingresso di tutti i registri è stabile negli intervalli $(t_i - T_{setup}, t_i + T_{hold})$. La prima e la terza tengono conto delle esigenze del mondo esterno a monte. Le ultime due tengono conto delle esigenze del mondo esterno a valle.

Ci sarebbe anche T_{sfas} da considerare. In generale è molto piccolo quindi lo supporremo nullo.

Modello di Moore (con registro STAR)

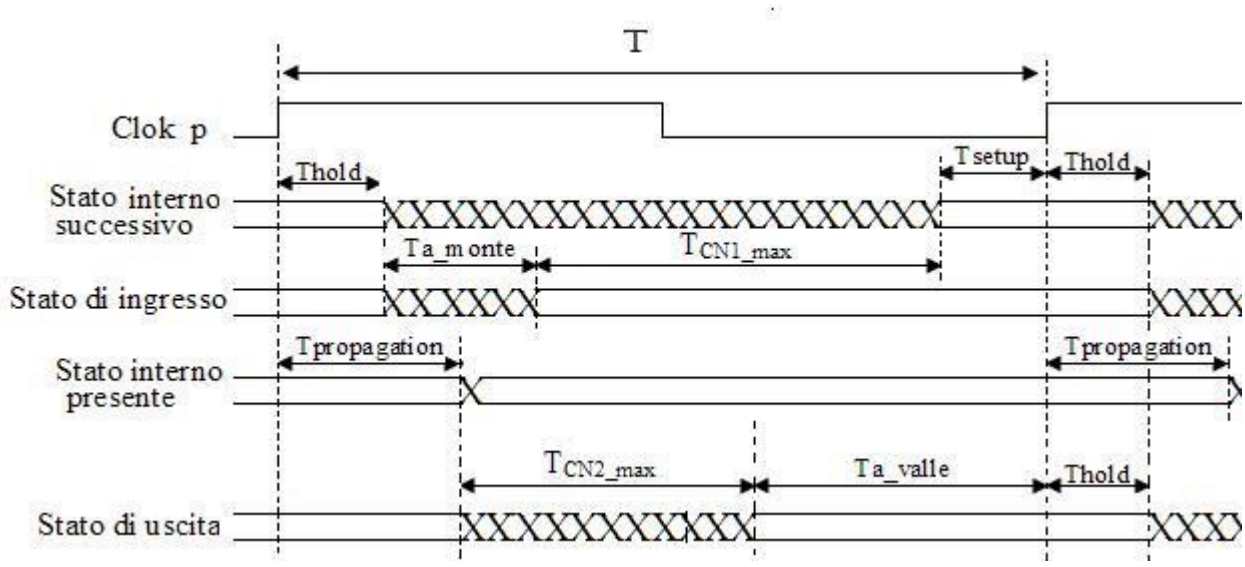
Modello strutturale:



- N variabili di ingresso.
- M variabili di uscita.
- Un meccanismo di marcatura che ad ogni istante marca uno stato interno presente $S \equiv \{S_0,\ldots,S_{k-1}\}$.
- Una legge **A:X×S→S** che aggiorna lo stato interno.
- Una legge **B:S→Z** che decide lo stato di uscita.
- Riceve segnali di sincronizzazione come transizioni da 0 a 1 del segnale di clock.
- Individua continuamente $Z=B(S)$ e lo presenta in uscita.

Il registro STAR è una batteria di D-FF, quindi il nuovo stato interno sarà presentato in uscita T_{prop} dopo il fronte in salita del clock.
Non ci sono problemi di corse né di alee essenziali.
Va a regime un po' dopo $\mathbf{t+T_{propagation}}$.

Temporizzazione:

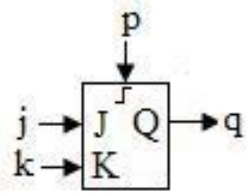


L'intervallo di tempo T tra due successivi segnali di sincronizzazione, deve soddisfare quindi tre condizioni:

$$T \geq T_{hold} + T_{a\_monte} + T_{CN1} + T_{setup}$$
$$T \geq T_{propagation} + T_{CN1} + T_{setup}$$
$$T \geq T_{propagation} + T_{CN2} + T_{a\_valle}$$

In generale la prima copre la terza, in quanto $T_{hold} + T_{a\_monte}$ è maggiore di $T_{propagation}$

Temporizzazione, descrizione e sintesi del flip-flop - JK



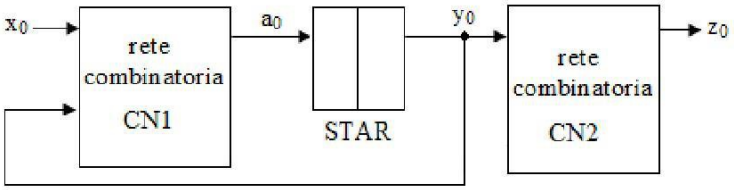
ho	voglio	j	k
0	0	0	-
0	1	1	-
1	1	-	0
1	0	-	1

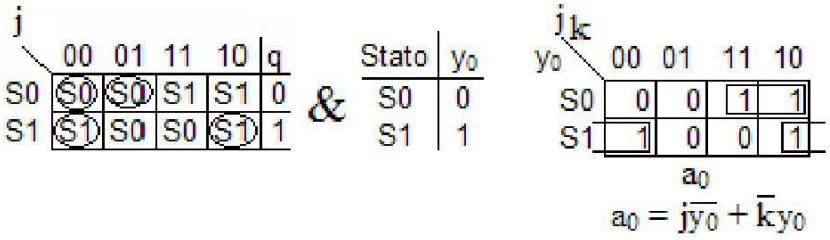
È una rete sequenziale sincronizzata con due ingressi j e k ed una di uscita p che si evolve secondo le specifiche. Quando arriva il segnale di sincronizzazione, se lo stato di ingresso è:

j = 1 e k = 0 il flip flop si setta, cioè mette a 1 il valore della variabile di uscita q j = 0 e k = 1 il flip flop si resetta, cioè mette a 0 il valore della variabile di uscita q

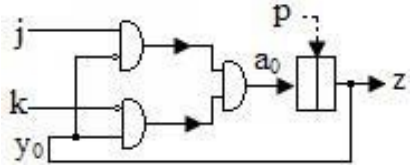
j = 0 e k = 0 il flip flop conserva, cioè mantiene inalterato il valore della variabile di uscita q j = 1 e k = 1 il flip flop commuta, cioè modifica il valore della variabile di uscita q

Utilizzando il modello strutturale di Moore otteniamo



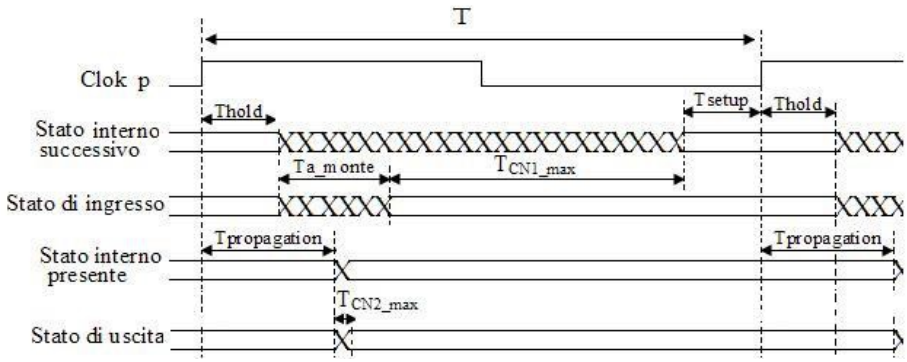


Mentre CN2 è un corto circuito

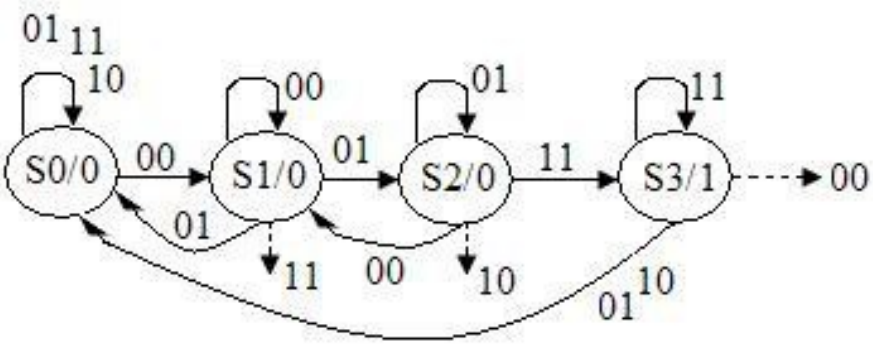


Questa sintesi di CN1 può presentare delle alee statiche del I ordine, ma nonostante questo non crea problemi

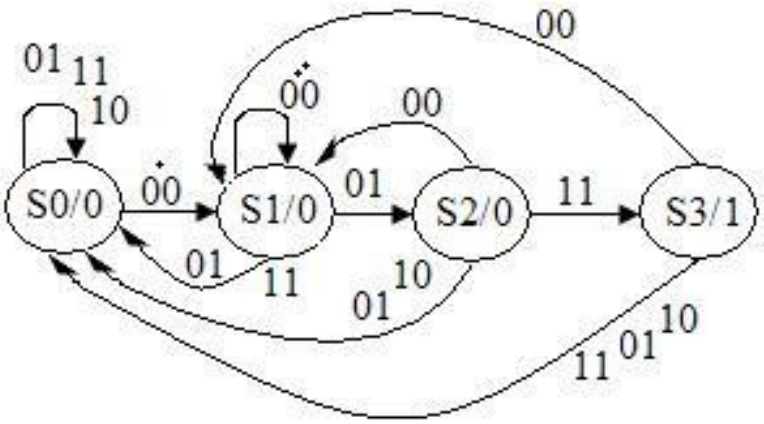
La temporizzazione è quella di Moore, con la semplificazione che CN2 essendo un cortocircuito ha $T_{CN2\_max} \sim 0$



Grafo di flusso del riconoscitore di sequenza 00,01,11 prima come rete sequenziale asincrona e poi come rete sequenziale sincronizzate di Moore



Quando gli anelli si richiudono sul nodo da cui partono siamo in una situazione di stabilità. Non siamo in presenza di un nuovo ingresso, ma bensì è il permanere dello stato precedente. Infatti uno stato di ingresso permane finchè non si arriva ad una situazione di stabilità (A[S,X] coincide con S o con A[A(S,X),X] , le leggi che soddisfano questa condizione sono dette normali.
Le linee che non si racchiudono corrispondono a comportamenti imprevedibili della rete a causa di una transizione multipla in ingresso (corrispondono a un non-specificato)



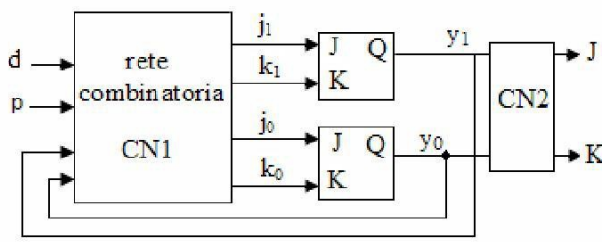
+ e ++ sono due stati di ingresso diversi, in particolare ++ non è un anello si stabilità (Moore è intrinsecamente stabile) ma è un nuovo inizio di sequenza. Quando sono in S2 infatti, se in ingresso ricevo 01 torno alla partenza perché quello non è ne lo stato d'ingresso che mi fa terminare la sequenza, ne tantomeno lo stesso che mi ha fatto arrivare in S2 al colpo di clock precedente.

Descrivere e sintetizzare il D-edge-triggered secondo il modello di circuito sequenziale sincronizzato di Moore, con un jk come elemento di marcatura

Tabella di flusso

d	p = 0		p = 1		q
	0	1	0	1	
S0	(S0)	(S0)	S1	S2	0
S1	S0	S0	(S1)	(S1)	0
S2	S3	S3	(S2)	(S2)	1
S3	(S3)	(S3)	S1	S2	1

Modello strutturale



Codifica degli stati

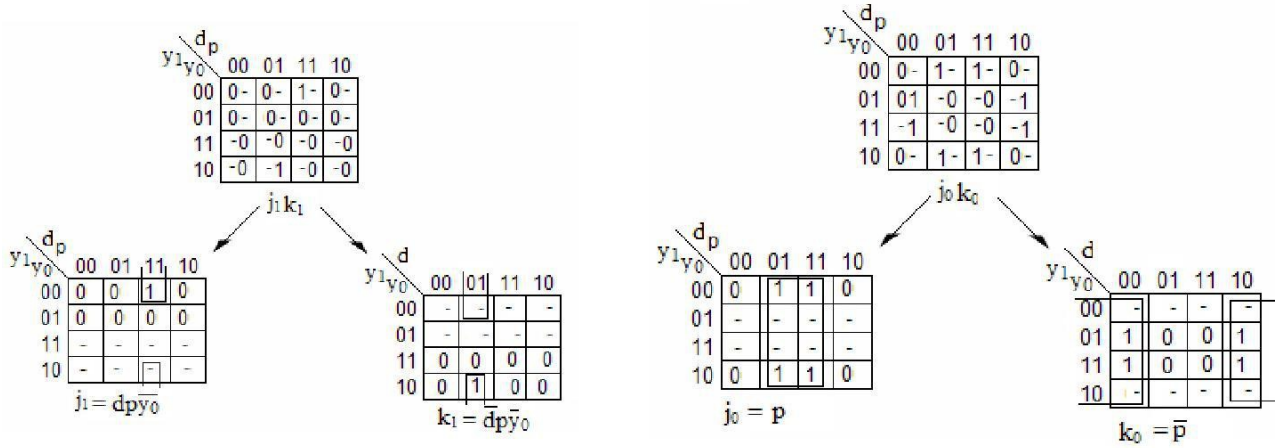
Stato	y_1	y_0
S0	0	0
S1	0	1
S2	1	1
S3	1	0

Tabella delle Transizioni

y_1 \ y_0 \ dp	00	01	11	10
00	00	01	11	00
01	00	01	01	00
11	10	11	11	10
10	10	01	11	10

ho	voglio	j	k
0	0	0	-
0	1	1	-
1	1	-	0
1	0	-	1

Sdoppio la tabella delle transizioni per studiare meglio i JK ed otteniamo:



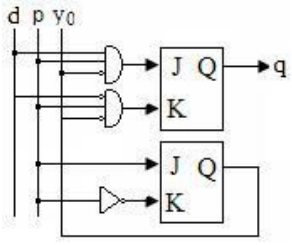
Per quanto riguarda CN2

y_1	y_0	z
0	0	0
0	1	0
1	1	1
1	0	1

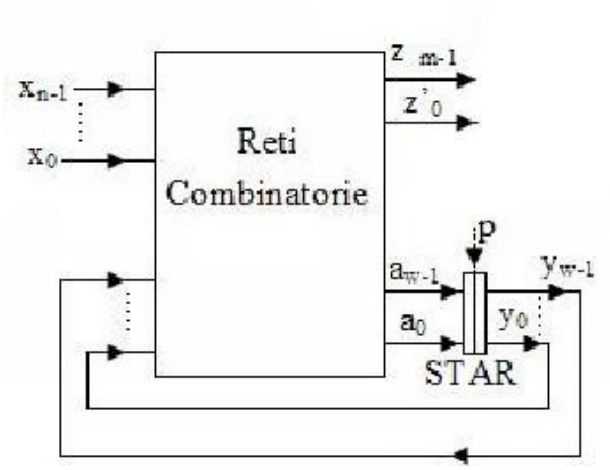
y_0 \ y_1	0	1
0	0	1
1	0	1

$z = y_1$

In conclusione il circuito che otteniamo è:



Modello di Mealy



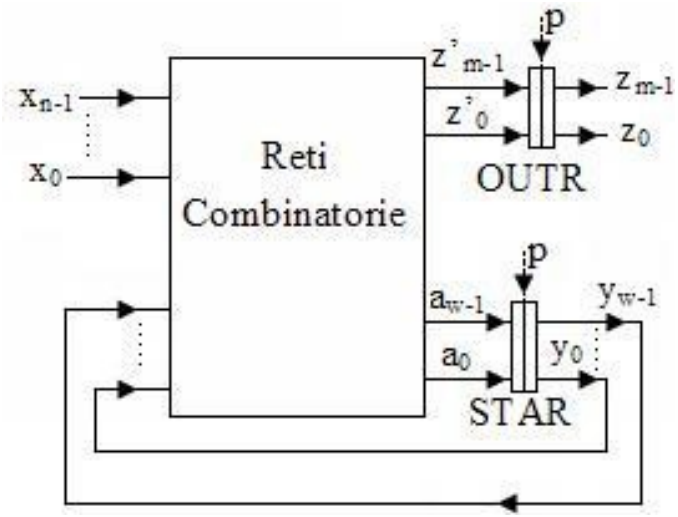
- N variabili di ingresso.
- M variabili di uscita.
- Un meccanismo di marcatura che ad ogni istante marca uno stato interno presente $S \equiv \{S_0,\ldots,S_{k-1}\}$.
- Una legge **A:X×S→S** che aggiorna lo stato interno.
- Una legge **B:X×S→Z** che decide lo stato di uscita.
- Riceve segnali di sincronizzazione come transizioni da 0 a 1 del segnale di clock.

<u>Temporizzazione:</u>

$$T \geq T_{hold} + T_{a\_monte} + T_{CN} + T_{setup}$$
$$T \geq T_{propagation} + T_{CN} + T_{setup}$$
$$T \geq T_{hold} + T_{a\_monte} + T_{CN} + T_{a\_valle}$$
$$T \geq T_{propagation} + T_{CN} + T_{a\_valle}$$

Modelli di Mealy Ritardato

Modello strutturale



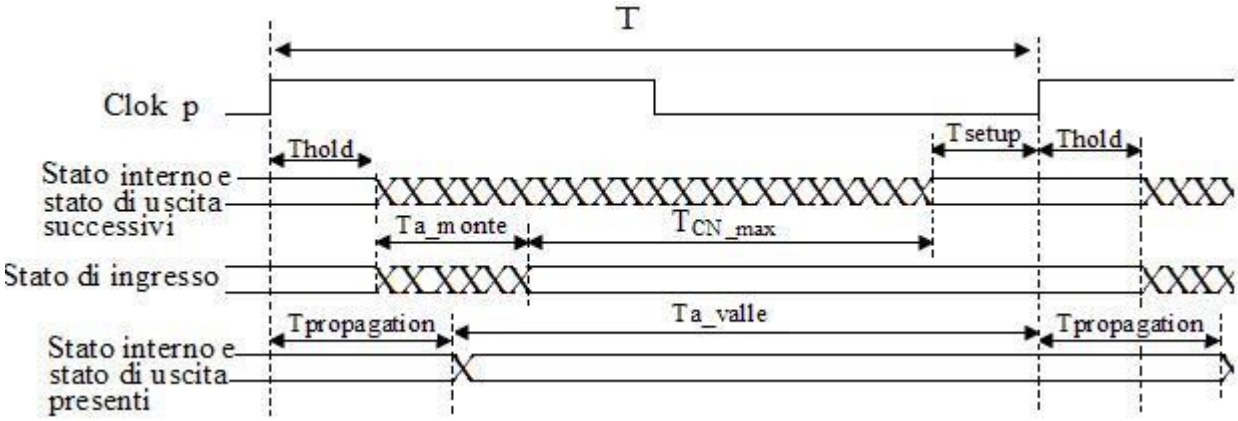
Nella tabella di flusso realizzata col modello di mealy ritardato si mettono gli stati accompagnati dall'uscita relativa

Una rete sequenziale sincronizzata di Mealy ritardato è una qualunque struttura che soddisfa i requisiti sottoelencati:

- N variabili di ingresso.
- M variabili di uscita.
- Un meccanismo di marcatura che ad ogni istante marca uno stato interno presente $S \equiv \{S_0,\ldots,S_{k-1}\}$.
- Una legge **A:X×S→S** che aggiorna lo stato interno.
- Una legge **B:X×S→Z** che decide lo stato di uscita.
- Riceve segnali di sincronizzazione come transizioni da 0 a 1 del segnale di clock.
- Ottempera alla seguente legge di evoluzione temporale: «detti X e S lo stato di ingresso e lo stato interno presenti ad un certo istante precedente l'arrivo di un segnale di sincronizzazione, individuare sia lo stato interno successivo **A(S,X)** sia lo stato di uscita successivo **B(S,X)** e promuovere il primo a rango di stato interno presente ed il secondo al rango di stato di uscita effettivo solo quando arriva il segnale di sincronizzazione, e così via all'infinito»

Una rete di Mealy Ritardato va a regime all'istante $\mathbf{t+T_{prop}}$**.**

Temporizzazione:



La rete combinatoria CN implementa le leggi A e B, il registro OUTR supporta le variabili di uscita e il registro di stato STAR supporta le variabili di stato. L'intervallo di tempo T tra due successivi segnali di sincronizzazione, deve soddisfare le tre condizioni:

$T \geq T_{hold} + T_{a\_monte} + T_{CN} + T_{setup}$
$T \geq T_{propagation} + T_{CN} + T_{setup}$
$T \geq T_{propagation} + T_{a\_valle}$

I maggiori vantaggi delle reti di Mealy Ritardato sulle reti di Moore sono:

- le variabili di uscita sono supportate direttamente da un registro, cosicché sono stabili e cambiano in tempi certi.
- minor numero di stati interni a parità di problema, grazie alla maggior flessibilità della legge B.

Definizione di rete trasparente e non-trasparente. Elencare le reti logiche rispondenti a questi modelli.

Rete trasparente: se varia lo stato di uscita quando è ancora sensibile allo stato d'ingresso.
Rete non-trasparente: se varia lo stato di uscita quando non è più sensibile a variazioni in ingresso ($T_{propagation} > T_{hold}$).

TRASPARENTI	NON TRASPARENTI
Tutte le Reti Combinatorie	Flip Flop
Latch	RSS di Moore e di Mealy Ritardato
RSS di Mealy	

Descrizione e sintesi del riconoscitore di sequenza visto come rete sequenziale sincronizzata di Mealy ritardato (11,01,10)

Modello strutturale

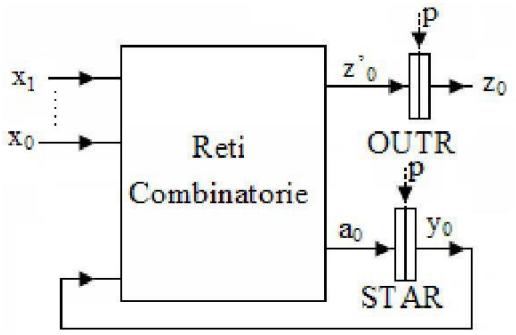


Tabella di flusso

x_1x_0	00	01	11	10
S0	S0/0	S0/0	S1/0	S0/0
S1	S0/0	S2/0	S1/0	S0/0
S2	S0/0	S0/0	S1/0	S0/1

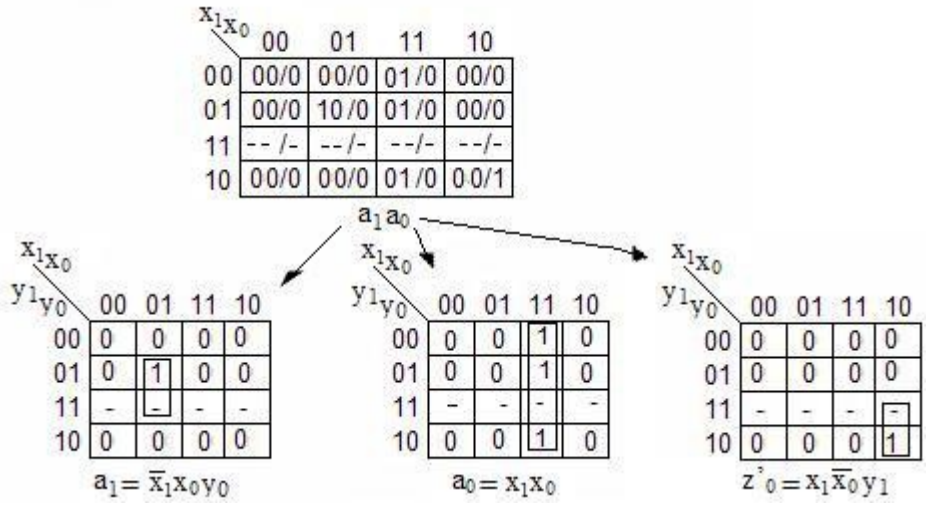
Codifica degli stati interni

Stato	y_1	y_0
S0	0	0
S1	0	1
S2	1	0

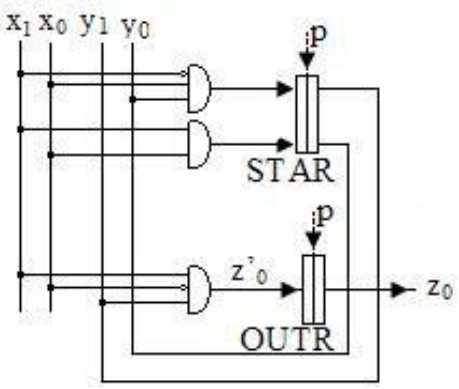
Tabella delle Transizioni

x_1x_0	00	01	11	10
00	00/0	00/0	01/0	00/0
01	00/0	10/0	01/0	00/0
11	--/-	--/-	--/-	--/-
10	00/0	00/0	01/0	00/1

La striplo

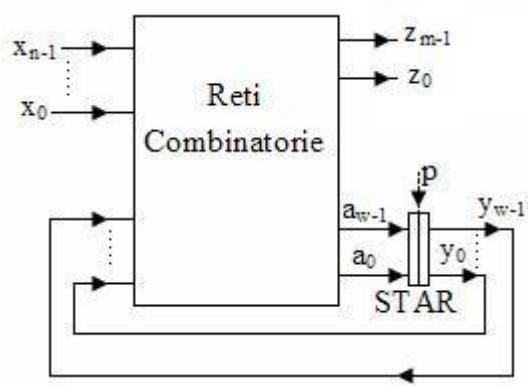


Otteniamo il seguente circuito



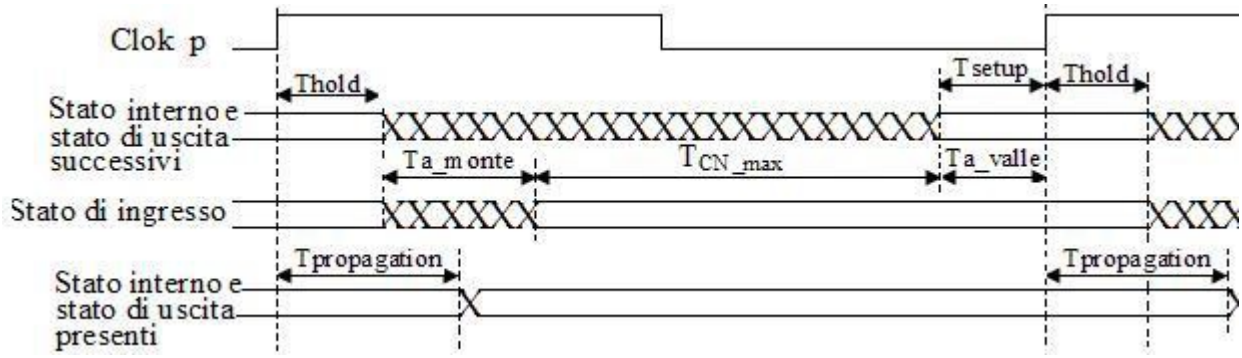
Temporizzazione di una rete di Mealy con flip flop D positive edge triggered come elementi di marcatura per gli stati interni

Modello strutturale



Si differenzia da Mealy ritardato perché l'uscita si preleva direttamente dalla rete combinatoria e non passa da OUTR. Il difetto è che in uscita si hanno troppi transitori

Temporizzazione:

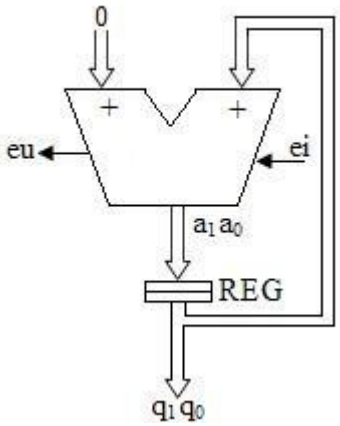


L'intervallo di tempo T tra due successivi segnali di sincronizzazione deve soddisfare le seguenti quattro condizioni:

$$T \geq T_{hold} + T_{a\_monte} + T_{CN\_max} + T_{setup}$$
$$T \geq T_{propagation} + T_{CN\_max} + T_{setup}$$
$$T \geq T_{propagation} + T_{a\_valle} + T_{CN\_max} + T_{a\_valle}$$
$$T \geq T_{hold} + T_{a\_monte} + T_{CN\_max}$$

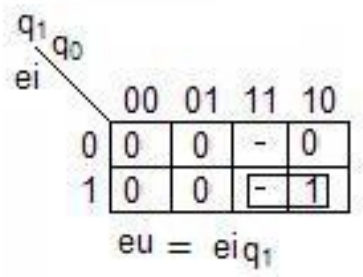
La quarta condizione, dove $T_{a\_monte} + T_{CN\_max} + T_{a\_valle}$ si sommano è di norma la più vincolante ed implica che T deve essere maggiore rispetto al caso delle reti di Moore e di Mealy ritardato della stessa "complessità".

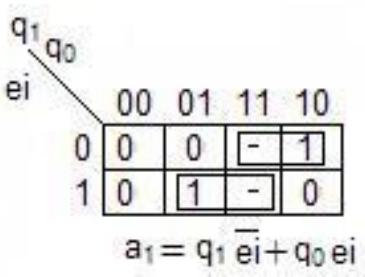
Contatore espandibile ad una cifra in base 3

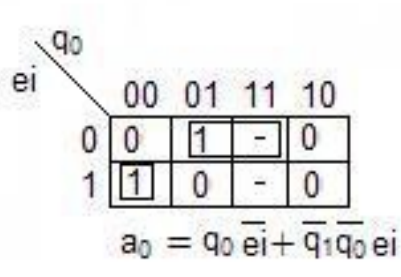


q_1	q_0	ei	eu	a_1	a_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	1	0
1	0	1	1	0	0
default			-	-	-

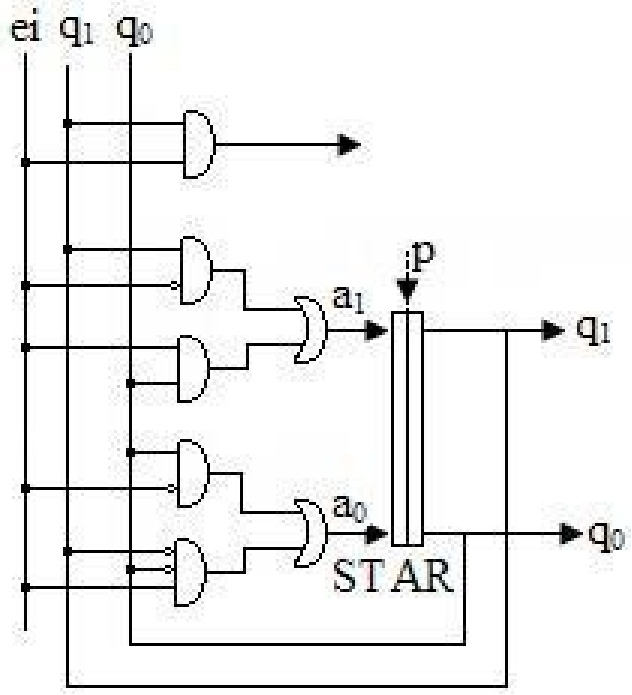
Dove le cifre sono state codificate: 0 = 00 , 1 = 01 , 2 = 10. Di conseguenza REG è dimensionato a 2 bit







Osserviamo che per la variabile d'uscita eu è una rete di Mealy, mentre per la variabile d'uscita q_1q_0 è di Moore(con CN2 cortucircuito)



Modelli Verilog varianti e considerazioni sulla temporizzazione

Esistono dei modelli strutturali per implementare la parte controllo di una unità sincronizzata, partendo dalla lista di statement che ne descrive l'evoluzione senza dover ricorrere alla tecnica di sintesi mediante tabella di flusso. Questi modelli strutturali portano ovviamente ad implementazioni non ottimizzate che sono accettabili solo nella descrizione iniziale sono presenti esclusivamente i seguenti due tipi di microsalto:

STAR <= (condizione) ? stato_interno1 : stato_interno2
STAR <= stato_interno

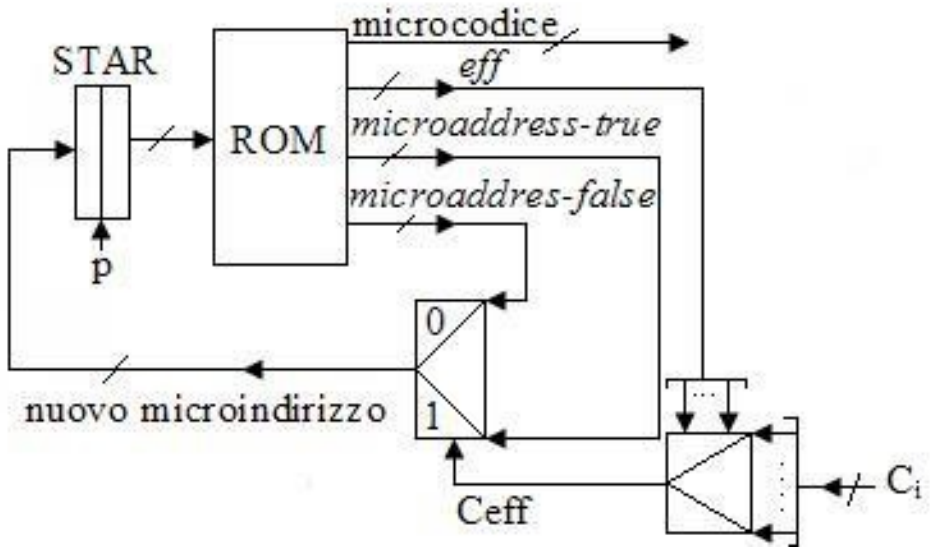
Un primo modello strutturale, *microaddress-based*, prevede una codifica su un numero minimo di bit (log2K)e porta ad un insieme di K codifiche dette *microindirizzi* (codifiche delle etichette degli statement). Il secondo modello, *microistruction-based*, prevede la codifica su un numero più ampio di bit e porta ad un insieme, diverso, di K codifiche dette *microistruzioni* (codifiche dei corpi dello statement)
Ogni microistruzione deve contenere:

microindice, lo stato che le variabili di comando assumono in corrispondenza di quello stato interno;
eff, la codifica dell'indice della variabile di condizionamento che compare nel microsalto appartenente allo statement avente per etichetta quello stato interno;
*microaddress-true,microaddres-false,*i microindirizzi dei due stati interni inclusi nel microsalto previsto nello statement;

Tutti e due i modelli utilizzano una rete combinatoria strutturata a ROM che ha in ingresso i k -indirizzi e produce le k -istruzioni.

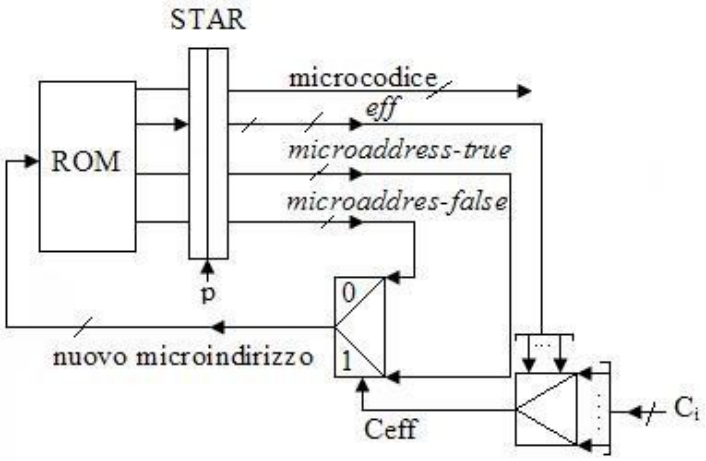
Modello Microaddress-based

Il registro STAR viene detto -istruction couter ed è dimensionato per contenere -indirizzi



Modello Microistruction-based

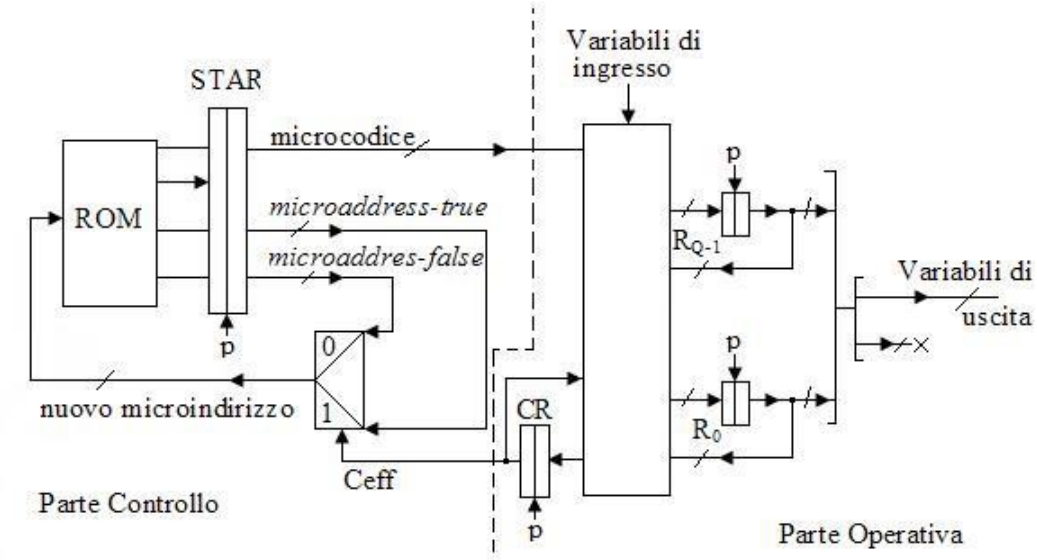
Il registro STAR viene detto -istruction register ed è dimensionato per contenere -istruzioni



Occorre notare che nel modello microistruction-based il registro STAR deve avere una notevole capacità, mentre nel modello microaddress-based difficilmente STAR ha una capacità superiore a 16 bit. Se però facciamo un confronto prendendo come parametro l'intervallo di tempo T fra due segnali di sincronizzazione successivi, le cose cambiano aspetto. In una implementazione con parte controllo microaddress-based le due reti più complesse (ROM e RC) sono in cascata, cosa che non accade nelle implementazioni in cui la parte controllo è microistruction-based .Qualora questa situazione dovesse porre un limite troppo stretto per T è possibile usare il seguente accorgimento, introdurre un registro operativo da un bit, chiamato *Condition Register* (CR) e sostituire li statement del tipo:
Sa: begin ...; STAR<= (Condizione)? Sa1: Sa2; end
Con coppie di statement del tipo:

Sa = begin ...; CR<= (Condizione)?1:0; STAR <=Sa_ponte; end
Sa_ponte : begin ...; STAR<= (CR==1)? Sa1: Sa2; end

In tal modo la rete combinatoria di condizionamento scompare in quanto inglobata nella rete combinatoria operativa. La struttura del sistema risulta quindi essere :

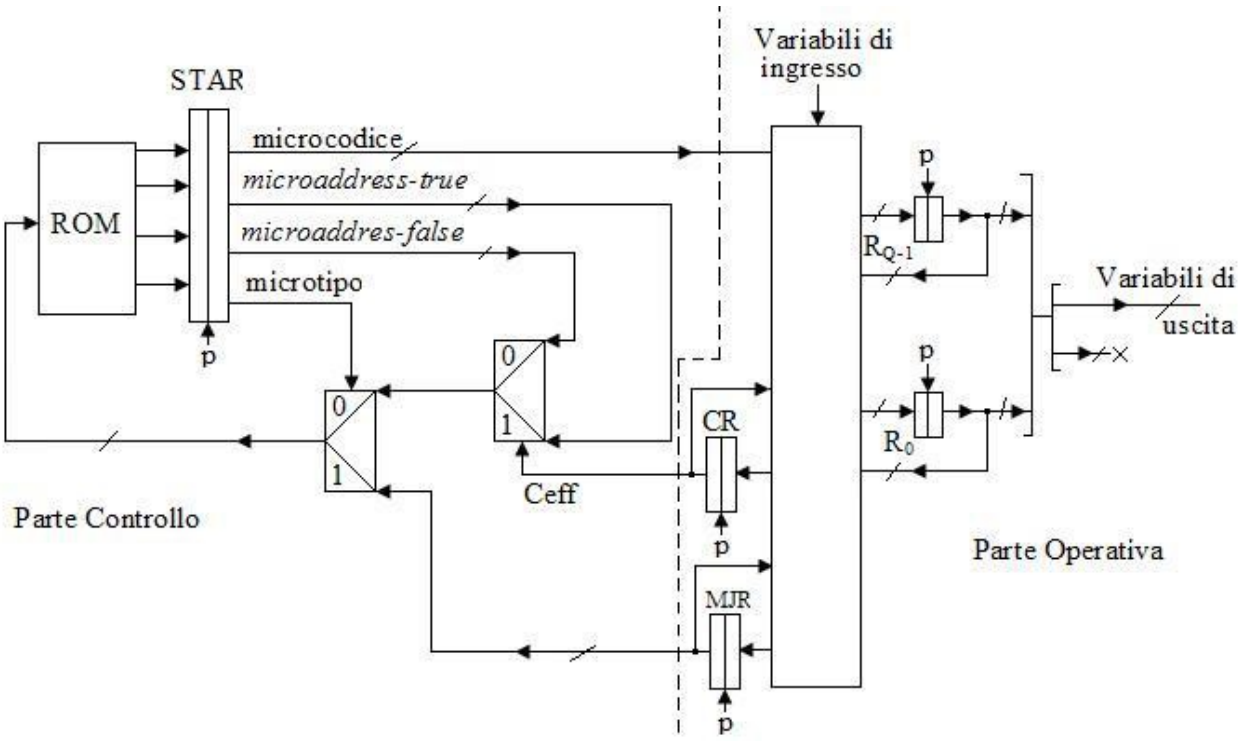


Modificare le strutture della parte controllo in modo che il processore accetti anche un istruzione del tipo STAR <= (CR==1) ? Sa : MJR

L'unità ora deve realizzare salti dei tipi:

1. STAR <= Sa;
2. STAR <= (CR==1) ? Sa : Sb;
3. STAR <= MJR;
4. STAR <= (CR==1) ? Sa : MJR;

Il 4 è un misto fra salti tradizionali e salti a molte alternative. Allora possiamo adottare la seguente struttura:



Il bit microtipo vale 0 nel caso di salti (1) e (2) ed uno per salti (3) e (4)

Modalità di Indirizzamento

Indirizzamento di registro
L'operando è il contenuto del registro specificato, che può essere AL/AH oppure DP/SP.

Indirizzamento immediato
E' possibile solo per l'operando sorgente. Prevede che l'operando sia contenuto nel campo *source* dell'istruzione stessa. esempio: OPCODE $0x20,AL

Indirizzamento di memoria

- **diretto:** un campo dell'istruzione contiene l'indirizzo;
- **indiretto:** un campo dell'istruzione specifica un registro puntatore che contiene l'indirizzo. E' utilizzato per accedere in sequenza a un vettore i cui elementi occupino locazioni adiacenti in memoria.

Indirizzamento delle porte di I/O
Viene utilizzato l'indirizzamento diretto. Le istruzioni in questione sono la IN e la OUT.

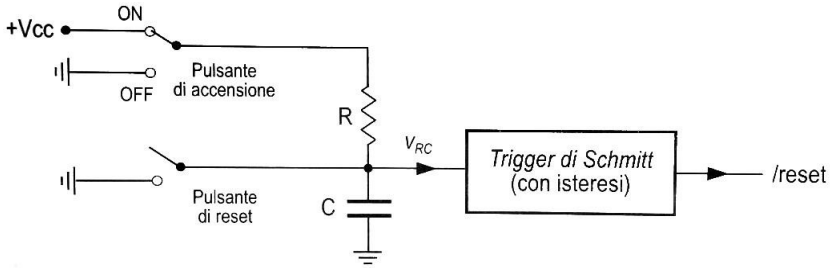
Cosa accade quando in un calcolatore viene premuto il tasto di reset?

Il processore inizializza il registro IP con'HFF0000 ed i flag F con 'H00. Inoltre i registri che supportano le variabili MR_, MW_ , IOR_, IOW_ vengono settati a 1. Il registro DIR viene messo a 0 in modo tale che le variabili d7_d0 siano in alta impedenza.
Il processore passa quindi alla fase di chiamata in cui si procura l'indirizzo dell'istruzione puntata da IP. Tale porzione di memoria deve essere quindi realizzata con tecnologia indelebile EPROM e ci deve essere stato codificato il programma di BOOTSTRAP.

```
if(reset_==0) #1
begin DIR<=0;MR_<=1; MW_<=1; IOR_<=1; IOW_<=1;F<='H00; IP<='HFF0000;
STAR<=fetch0;
end
```

Dove è connesso il filo di /reset in un calcolatore?

Il filo di /reset è un ingresso comune a tutto il calcolatore ed è generato dal seguente circuito:



Circuito che implementa la porzione del blocco always riportata sotto

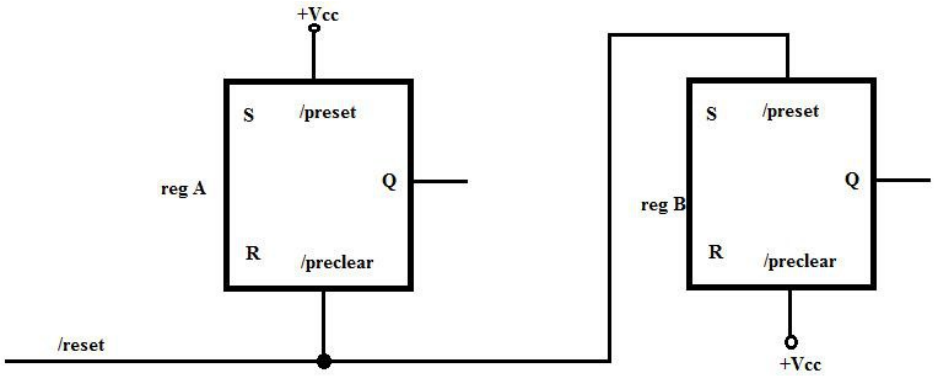
…

```
reg A,B;
```

…

```
always@(posedge p or negedge reset_)if(reset_==0) begin A<=0; B<=1; end
else…
```

Soluzione:



NB: ricordarsi dove (e come) è connesso il filo di /reset perché a volte può essere richiesto.

(In generale) come inizializzare un latch SR al reset asincrono

Per avere **q=0** bisogna connettere /**preset=1, /reset a /preclear**.
Per avere **q=1** bisogna connettere **/preclear=1, /reset a /preset.**
NB: connettere una variabile a 1 significa collegarla a **+Vcc**.

Fase di chiamata del processore

```
fetch0: begin A23_A0<=IP; IP <= IP+1; MJR<= fetch1; STAR<=readB; end
fetch1: begin OPCODE<=APP0;
MJR <= (d7_d0[7:5]==F0)?fetchEnd:
(d7_d0[7:5]==F1)?fetchEnd:
(d7_d0[7:5]==F2)?fetchF2_0:
(d7_d0[7:5]==F3)?fetchF3_0:
(d7_d0[7:5]==F4)?fetchF4_0:
(d7_d0[7:5]==F5)?fetchF5_0:
(d7_d0[7:5]==F6)?fetchF6_0:
/* default */ fetchF7_0;
STAR <= (valid_fetch(APP0)==1)?fetch2:nvi; end
fetch2 : begin STAR<=MJR; end

fetchF2_0: begin A23_A0<=DP; MJR<=fetchF2_1; STAR<=readB; end
fetchF2_1: begin SOURCE<=APP0; STAR<=fetchEnd; end

fetchF3_0: begin DEST_ADDR<=DP; STAR<=fetchEnd; end

fetchF4_0: begin A23_A0<=IP; IP<=IP+1; MJR<=fetchF4_1; STAR<=readB; end
fetchF4_1: begin SOURCE<=APP0; STAR<=fetchEnd; end

fetchF5_0: begin A23_A0<=IP; IP<=IP+1; MJR<=fetchF5_1; STAR<=readM; end
fetchF5_1: beginA23_A0<={APP2,APP1,APP0}; MJR<=fetch5_2; STAR<=readB; end
fetchF5_2: begin SOURCE<=APP0; STAR<=fetchEnd; end

fetchF6_0: begin A23_A0<=IP; IP<=IP+1; MJR<=fetchF6_1; STAR<=readM; end
fetchF6_1: begin DEST_ADDR<={APP2,APP1,APP0}; STAR<=fetchEnd;

fetchF7_0: begin A23_A0<=IP; IP<=IP+1; MJR<=fetchF7_1; STAR<=readM; end
fetchF7_1: begin DEST_ADDR<={APP2,APP1,APP0}; STAR<=fetchEnd; end

fetchEnd: begin MJR<=first_execution_state(OPCODE); STAR<=fetchEnd1; end
fetchEnd1: begin STAR<=MJR; end
```

Completare la fase di chiamata in modo che venga generate un interruzione interna di tipo 15 se (alla fine) OW=1

```
aluAL: begin AL<=alu_result(OPCODE, SOURCE, AL); F<={F[7:4],
alu_flag(OPCODE, SOURCE, AL); STAR<=aluAL1; end
```

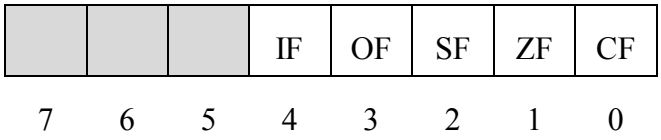
```
aluAL1: ???
//NB: OW è spesso indicato con OF ed è l'overflow flag
```

Soluzione:

```
aluAL1: begin SOURCE<=(F[3]==1)?'H0F:SOURCE;
STAR<=(F[3]==1)?int:test_intr; end
```

Altra soluzione con esercizio simile (int tipo 15 se OF o CF sono a 1):

```
aluAL1: begin SOURCE<=((F[3]==1)|(F[0]==1))?'H0F:SOURCE;
STAR<=((F[3]==1)|(CF[0]==1))?int:test_intr; end
```



In figura: il registro dei Flag F.

L'istruzione CALL *indirizzo* può essere sostituita dalla coppia

PUSH IP

JMP indirizzo

Si? No? Perché?

Si perché la PUSH IP salva il contenuto di IP nella pila corrente. JMP *indirizzo* mette in IP l'indirizzo specificato ed effettua il salto. Queste due operazioni sono equivalenti a quanto svolto dalla CALL *indirizzo*.

Fase di esecuzione di PUSH AL

```
pushAL: begin A23_A0<=SP-1; SP<=SP-1; APP0<=AL; MJR<=test_intr;
STAR<=writeB; end
```

Se viene richiesta la PUSH AH basta sostituire nel verilog AH al posto di AL.

Istruzione POP AL

```
popAL: begin A23_A0<=SP; SP<=SP+1; MJR<=popAL1; STAR<=readB; end
popAL1: begin AL<=APP0; STAR<=test_intr; end
```

Se viene richiesta la POP AH basta sostituire nel verilog AH al posto di AL.

Istruzione CALL *indirizzo*

```
call: begin A23_A0<=SP-3; SP<=SP-3; {APP2,APP1,APP0}<=IP; MJR<=call1;
STAR<=writeM; end
call1: begin IP<=DEST_ADDR; STAR<=fetch0; end
```

Istruzione RET

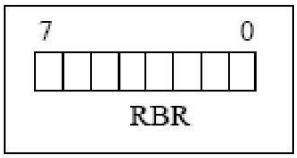
```
ret: begin A23_A0<=SP; SP<=SP+3; MJR<=ret1; STAR<=readM; end
ret1: begin IP<={APP2,APP1,APP0}; STAR<=test_intr; end
```

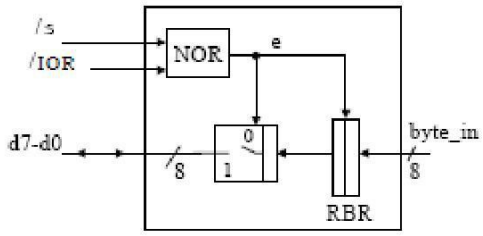
In un calcolatore chi determina (e come) gli offset delle porte?

Gli offset delle porte vengono decise dal progettista del calcolatore, il quale le implementa tramite delle maschere collegate al BUS degli indirizzi e servono proprio a riconoscere alcuni di essi.

Struttura interna delle seguenti interfacce senza handshake ingresso, uscita, ingresso/uscita

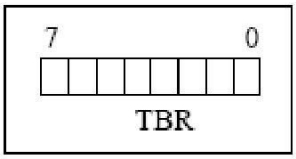
Schema funzionale di una semplice interfaccia d'ingresso senza handshake

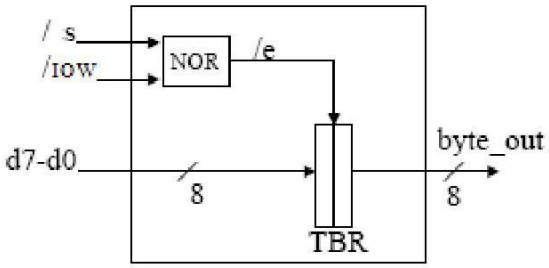




L'interfaccia contiene un registro da 8 bit mediante il quale implementa una porta dello spazio di I/O. L'indirizzo di tale porta dipende da una maschera che affianca l'interfaccia. La maschera che genera /s riceve in ingresso tutte e 16 le variabili degli indirizzi e la configurazione per la quale genera 0 corrisponde all'indirizzo nello spazio di I/O del registro RBR. Quando l'interfaccia è selezionata (/s=0) e inizia un ciclo di lettura (/ior va a zero), allora *e* passa da 0 a 1 ed il registro RBR memorizza il valore delle variabili byte_in. Inoltre le porte 3-state passano in conduzione. Receiver Buffer Register (RBR): contiene l'ultimo byte che l'interfaccia ha prelevato dal trasduttore esterno. RBR è accessibile in lettura al processore.

Schema funzionale di una semplice interfaccia d'uscita senza handshake

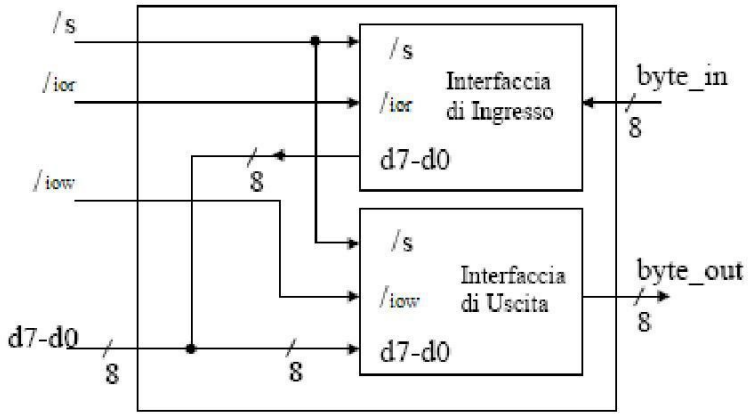




L'interfaccia contiene un registro da 8 bit mediante il quale implementa una porta dello spazio di I/O. L'indirizzo di tale porta dipende da una maschera che affianca l'interfaccia. La maschera che genera /s riceve in ingresso tutte e 16 le variabili degli indirizzi e la configurazione per la quale genera 0 corrisponde direttamente all'indirizzo nello spazio di I/O del registro TBR. Quando l'interfaccia è selezionata (/s=0) e inizia un ciclo di scrittura (/iow va a 0), anche /e passa da 1 a 0. Quando finisce il ciclo di scrittura /iow torna a 1 ed /e passa a 1, quindi il registro TBR memorizza il byte presentato dal processore. Transmitter Buffer Register (TBR): il byte che contiene è reso disponibile al trasduttore esterno. TBR è accessibile in scrittura al processore.

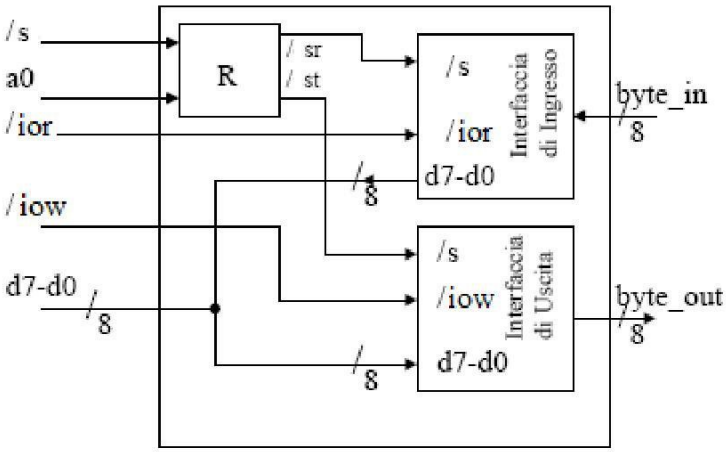
Interfaccia parallela di ingresso-uscita senza handshake

I due registri RBR e TBR implementano la stessa porta dello spazio di I/O: se il ciclo e' di lettura viene coinvolto il registro RBR, se il ciclo e' di scrittura viene coinvolto il registro TBR.
Da un punto di vista funzionale è come se l'interfaccia avesse un unico registro RTBR



R

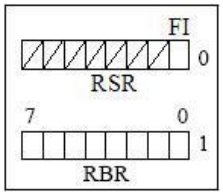
/s	a0	/sr	/st
1	-	1	1
0	0	0	1
0	1	1	0



Interfaccia parallela di ingresso/uscita che mantiene la distinzione tra i registri RBR e TBR.
Il registro viene selezionato mediante a0.

Interfaccia parallela d'ingresso con handshake, visione funzionale, struttura interna, gestione software a controllo di programma.

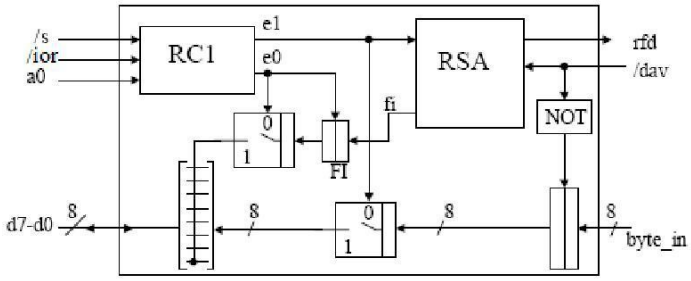
Visione funzionale:



il microprocessore può accedere all'interfaccia per compiere cicli di lettura sia da RBR che da RSR (il cui unico bit significativo è il primo, chiamato FI) FI a 1 implica un byte disponibile per il microprocessore nel registro RBR

L' interfaccia parallele con handshake è dotata di due variabili, *rfd* e */dav*, che consentono di colloquiare con il trasduttore. Le variabili */dav* e *rfd* sono rispettivamente di ingresso e di uscita per l'interfaccia di ingresso. L'handshake è gestito da una rete sequenziale asincrona che gestisce anche la variabile FI

Struttura interna:



Dove la rete combinatoria RC1 è :

/s	/ior	a0	e1	e0
0	0	0	0	1
0	0	1	1	0
others			0	0

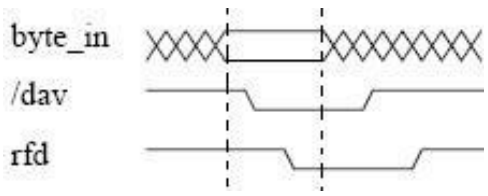
E la rete sequenziale asincrona RSA è :

e1 /dav	00	01	11	10	rfd	fi
S0	S1	(S0)	-	-	1	0
S1	(S1)	-	-	S2	1	1
S2	(S3)	-	-	(S2)	1	0
S3	(S3)	S0	-	-	0	0

Gestione software (assembler)

```
Offset          Istruzioni
'H2000          IN RSR_offset,AL
'H2003          AND $0x01,AL
'H2005          JZ 0x002000
'H2008          IN RBR_offset,AL
'H200B          RET
```

Temporizzazione del handshake fra l'interfaccia e il trasduttore:



Situazione iniziale

rfd = 1 l'interfaccia è disponibile a prelevare un dato

/dav =1 nessun dato utile è stato presentato dal trasduttore all'interfaccia

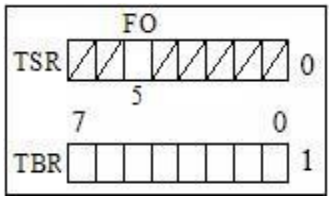
Il trasduttore presenta un byte utile come stato delle variabili *byte_in* e pone */dav* a 0

L'interfaccia preleva il byte utile e lo memorizza nel registro RBR, quindi pone *rfd* a 0

Il trasduttore riporta */dav* a 1 ed attende che l'interfaccia riporti *rfd* a 1 ad indicare la disponibilità ad accettare un nuovo dato

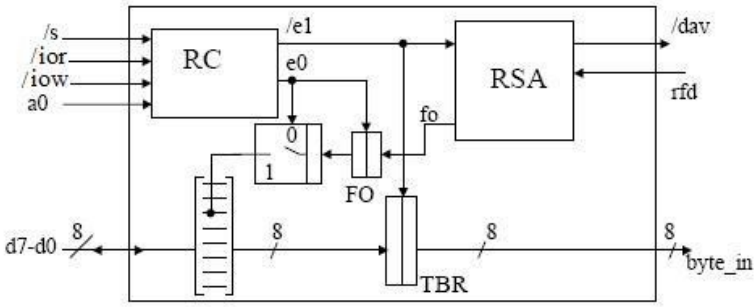
Interfaccia parallela di uscita con handshake: visione funzionale e gestione software, struttura interna

Visione funzionale:



il processore può accedere all'interfaccia per compiere cicli di scrittura nel registro TBR e cicli di lettura del registro TSR il cui unico bit significativo è il bit 5, chiamato FO. FO a 1 implica che il byte contenuto in TBR è già stato correttamente prelevato, quindi il processore ci può scrivere sopra.

Struttura interna:



Dove la rete combinatoria RC è:

/s	/ior	/iow	a0	/e1	e0
0	0	1	0	0	1
0	1	0	1	1	0
others				0	0

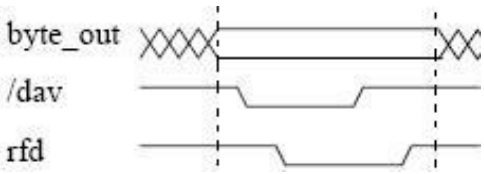
la rete sequenziale asincrona è:

/e1 rfd	00	01	11	10	/dav	fo
S0	-	(S0)	S1	-	1	1
S1	-	S2	(S1)	-	1	0
S2	S3	(S2)	-	-	0	0
S3	(S3)	S0	-	-	1	0

Gestione software (assembler)

Offset	Istruzioni
'H2010	PUSH AL
'H2011	IN TSR_offset, AL
'H2014	AND $0x20,AL
'H2016	JZ 0x002011
'H2019	POP AL
'H201A	OUT AL,TBR_offset
'H201D	RET

Temporizzazione dello handshake del trasmettitore



•Situazione iniziale:

rfd = 1 trasduttore è disponibile a prelevare un dato

/dav =1 nessun dato utile è contenuto nel registro TBR

•L'interfaccia presenta un byte utile come stato delle variabili *byte_out* e pone */dav* a 0 •Il trasduttore preleva il byte utile, quindi pone *rfd* a 0

•L'interfaccia riporta */dav* a 1 ed attende che il trasduttore riporti *rfd* a 1 ad indicare la sua disponibilità ad accettare un nuovo dato

Gestione software per l'ingresso dati

```
void array_in(word counter, byte* pointer){
      #define RSR_offset …
      #define RBR_offset …
      while(counter!=0){
            byte tmp; do{tmp=inport(RSR_offset)&0x01;} while(tmp==0x00);
            *pointer=inport(RBR_offset); pointer++; counter--;
      }
}
```

Gestione software per l'uscita dati

```
void array_out(word counter, byte* pointer){
      #define TSR_offset …
      #define TBR_offset …
      while(counter!=0){
            byte tmp; do{tmp=inport(TSR_offset)&0x20;} while(tmp==0x00);
            outport(TBR_offset,*pointer); pointer++; counter--;
      }
}
```

Interfaccia seriale: visione funzionale, sottoprogramma di uscita, struttura del trasduttore e del ricevitore

Un dispositivo trasmettitore ed un ricevitore sono in grado di scambiare dati mediante una sola linea di collegamento sulla quale viaggiano serialmente i singoli bit. I bit sono trasmessi e ricevuti in gruppi detti *trame*. Durante la trasmissione di una trama i bit sono trasmessi con cadenza regolare (l'intervallo di un tempo T tra un bit e l'altro è detto *tempo di bit*). L'intervallo di tempo che intercorre tra la fine di una trama e l'inizio della successiva non è soggetto a vincoli.

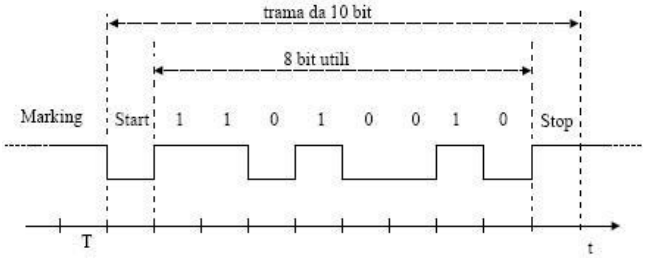
Bit-rate: numero di bit inviati nell'unità di tempo durante la trasmissione di una trama (1/T). Una trama è composta da un numero di bit che va da 7 a 12:

un *bit di start*

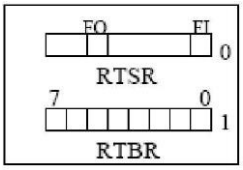
da 5 a 8 bit utili (l'informazione vera e propria) un eventuale *bit di parità*

uno o due *bit di stop*

Tra una trama e l'altra la linea viene mantenuta nello *stato di marking.* Per trasmettere il bit di start si porta la linea nello *stato di spacing.* I bit di stop vengono trasmessi mantenendo la linea nello stato di marking. I bit utili vengono trasmessi portando la linea nello stato di marking (1) o nello stato di spacing (0)

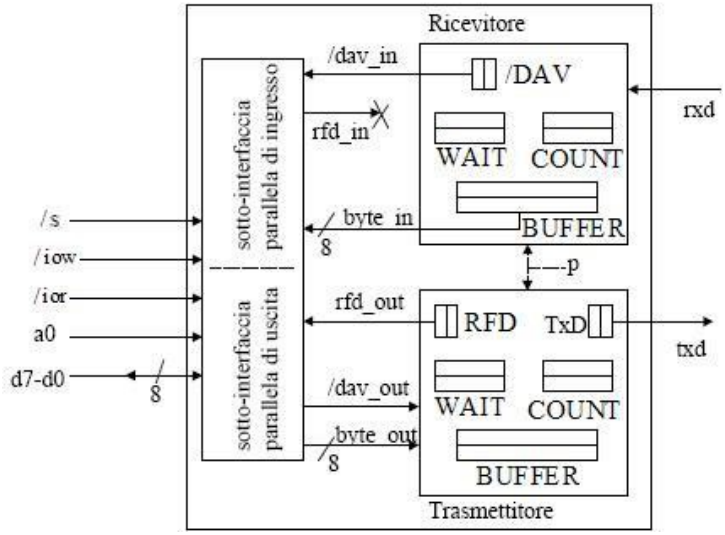


Visione funzionale:



La struttura interna dell'interfaccia comprende due sottointerfacce parallele con handshake che mascherano completamente al processore il ricevitore ed in trasmettitore.

Struttura interna:



Il ricevitore riceve segnali di sincronizzazione con un periodo 16 volte più piccolo del tempo di bit T . Mentre il trasmettitore riceve segnali di sincronizzazioni pari al tempo di bit T . Il ricevitore non può sostenere un handshake completo con la sottointerfaccia perché ogni qual volta riceve una nuova trama deve rimuovere il vecchio byte a prescindere dal fatto che la sottointerfaccia d'ingresso lo abbia prelevato o meno.

Descrizione del Ricevitore

```
module Ricevitore(dav_in_, byte_in, rxd, clock_ric, reset_)
input clocl_ric, reset_;
input rxd;
output dav_in_;
output [7:0] byte_in;
reg DAV_; assign dav_in_=DAV_;
reg [3:0] COUNT;
reg [4:0] WAIT;
reg [7:0] BUFFER; assign byte_in=BUFFER;
reg [1:0] STAR; parameter  S0=0,S1=1,Wbit=2, Wstop=3;
parameter  start_bit = 1'B0;
assign byte_in= BUFFER; assign dav_in_ = DAV_;
always @(reset_==0)#1 begin DAV_<=1; STAR=S0; end
always @(posedge clock_ric)if (reset_==1)#3
casex (STAR)
S0:   begin DAV_<=1 ; COUNT<=8 ;  WAIT<=23; STAR<=(rxd==star_bit)?Wbit:S0;
      end
S1: begin BUFFER<={rxd,BUFFER[7:1]}; COUNT <= COUNT-1; WAIT<=15;
STAR<=(COUNT==1)?Wstop:Wbit; end
Wbit: begin WAIT<=WAIT-1; STAR<=(WAIT==1)?S1:Wbit; end
Wstop: begin DAV_<=0; WAIT<=WAIT-1; STAR<=(WAIT==1)?S0:Wstop; end
endcase
endmodule
```

Descrizione del trasmettitore

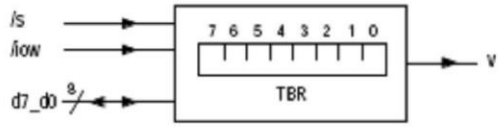
```
module Trasmettitore (rxd_out, dav_out_, byte_ou,txd,clock_tra,reset_)
input clocl_ric, reset_;
input dav_out_;
output rfd_out, txd;
input [7:0] byte_out;
reg RFD,TXD; assign rfd_out=RFD; assign txd=TXD;
reg [3:0] COUNT;
reg [7:0] BUFFER;
reg [1:0] STAR; parameter  S0=0,S1=1,S2=2;
parameter mark = 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_tra) if(reset_==1) #3
casax(STAR)
S0: begin RDF <= 1; COUNT<=10; TXD<=mark; BUFFER<={stop_bit,
byte_out,star_bit}; STAR<=(dav_out_==1)?S0:S1; end
S1: begin RFD<=1; 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
```

Perché in un ricevitore seriale il periodo del clock è minore (16 volte) del tempo di bit T ?

Per poter seguire con precisione l’evoluzione dello stato della variabile di ingresso rxd ed avere una buona centratura di tutti i bit. Più precisamente per un prelievo dei bit negli istanti temporalmente più distanti dal momento in cui rxd cambia stato e quindi negli istanti in cui il rapporto segnale/rumore è ottimale.

Interfaccia e convertitore D/A

Visione funzionale di un interfaccia con risoluzione N = 8 bit

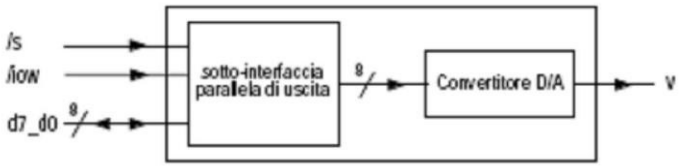


da un punto di vista funzionale è quindi sufficiente che il processore esegua l'istruzione:

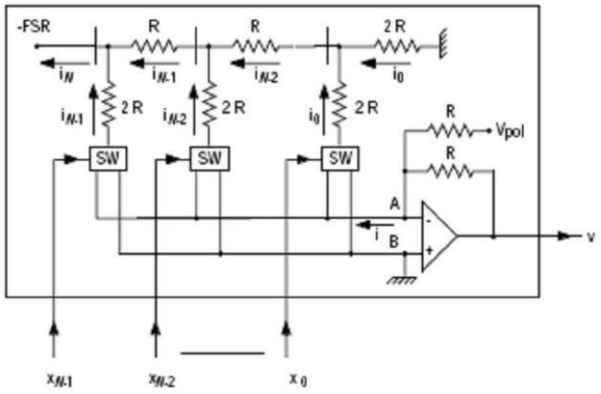
OUT AL, TBR_offset

per invitare l'interfaccia a generare una nuova tensione analogica.
Essendo il tempo impiegato dal convertitore trascurabile rispetto al tempo di esecuzione di una istruzione, non ho bisogno di handshake. L'interfaccia di uscita memorizza il byte inviato dal processore in TBR, interpreta il byte come numero x, genera e rende disponibile per un apparato esterno una tensione analogica v proporzionale a tale numero.

Struttura interfaccia



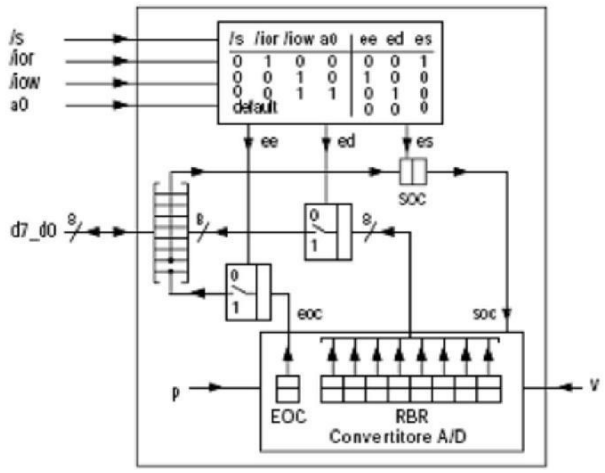
Convertitore D/A
Il convertitore interpreta il byte ricevuto in ingresso in accordo alla legge binaria bipolare, cioè X è numero naturale espresso in base 2 dal byte d'ingresso $x = -2^{N-1} + X$. La tensione generata in uscita è data da $v = K \cdot x$ dove K è una costante di proporzionalità, che dipende da N(numero di bit o risoluzione)e da una tensione di riferimento applicata al convertitore FSR (full scale range) secondo la legge $FSR = 2^N \cdot K$. La relazione $v = K \cdot x$ è relativa al convertitore ideale, quella reale è $|v - K \cdot x| \leq err$ dove $err = \varepsilon \cdot FSR/100$ dove ε è detto coefficiente d'errore. Se $LSB = 100 / 2^N$ si ha che $\varepsilon < ½\ 100 / 2^N$ il che implica $|v - K \cdot x| \leq ½ \cdot 100 / 2^N \cdot FSR/100 = K/2$
Il convertitore D/A è quindi un circuito "combinatorio" e produce delle tensioni spurie in uscita, quando lo stato di ingresso viene rimosso e sostituito con un altro che differisce dal vecchio per più di un bit. Per eliminare queste variazioni brusche e passeggere della tensione di uscita, il convertitore è di solito seguito da un filtro passa-basso. L'assenza quasi totale di transitori di corrente nella rete resistiva favorisce la velocità di risposta del convertitore.



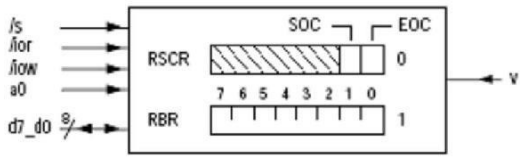
Gli switch analogici hanno la funzione di commutare la corrente verso il nodo A o B a seconda che l'ingresso x_i associato valga 0 o 1 quindi:
$i = i_{N-1} \cdot x_{N-1} + i_{N-2} \cdot x_{N-2} + ... + i_0 \cdot x_0$.
La tensione di uscita v a causa del cortocircuito virtuale è: $v = R \cdot i$ -Vpol

Interfaccia per la conversione A/D. Struttura e gestione software

Struttura:

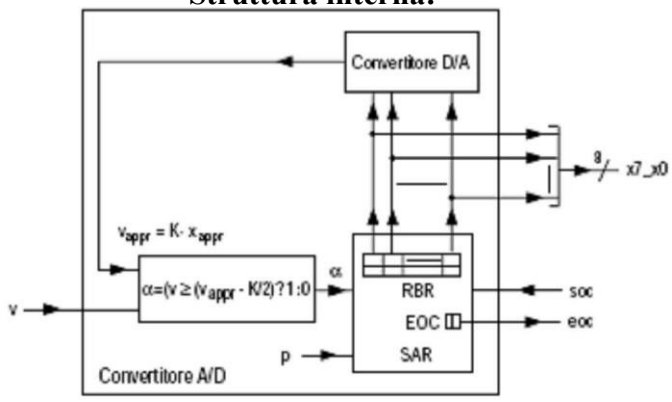


Visione funzionale:



È un interfaccia di ingresso che riceve da un apparato esterno la tensione da convertire, compie su comando del processore l'operazione di conversione costruendo un byte, memorizza tale byte nel registro RBR e lo rende disponibile al processore. L'interfaccia si basa su un convertitore A/D ad approssimazioni successive il cui registro RBR funge da registro RBR dell'intera interfaccia.
L'interfaccia è inoltre dotata di un registro RSCR (*Receiver Status Command Register*) con due elementi significativi, che chiameremo SOC (S*tart Of Conversion*, elemento n. 1) ed EOC (*End Of Conversion*, elemento n. 0) e che supportano le variabili interne *soc* ed *eoc*. L'elemento SOC, è accessibile al processore esclusivamente per cicli di scrittura, supporta l'omonima variabile
di ingresso del convertitore e permette pertanto al processore di comandare (immettendovi 1) l'avvio di una nuova operazione di conversione. L'elemento EOC è fisicamente interno al convertitore, supporta l'omonima variabile *eoc* ed è accessibile al processore esclusivamente per cicli di lettura, permettendo così al processore di avere informazioni sullo stato della conversione.

Struttura interna:



Gestione software

```
byte acquisizione(){
#define RSCR_offset…
#define RBR_offset…
byte tmp;

outport(RSCR_offset,0x02);

do{tmp=inport(RSCR_offset)&0x01;}while(tmp!=0);
outport(RSCR_offset,0x00);

do{tmp=inport(RSCR_offset)&0x01;}
while(tmp==0x00);

return inport(RBR_offset);
}
```

Descrizione dell'unità sincronizzata SAR

```
module SAR(eoc,x7_x0,soc,alpha,clockSAR,reset_);
input clockSAR,reset_;
input soc,alpha;
output eoc;
output [7:0] x7_x0;
reg EOC;
reg [7:0] RBR;
reg [3:0] STAR; parameter
S0=0,S1=1,S2=2,S3=3,S4=4,S5=5,S6=6,S7=7,S8=8,S9=9
,S10=10;
assign eoc=EOC;
assign x7_x0=RBR;
always @(reset_==0) 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
S6: begin RBR<={RBR[7:4],alpha,'B100};
STAR<=S7;end
S7: begin RBR<={RBR[7:3],alpha,'B10};
STAR<=S8;end
S8: begin RBR<={RBR[7:2],alpha,'B1}; STAR<=S9;end
S9: begin RBR<={RBR[7:1],alpha }; STAR<=S10; end
S10: begin EOC<=(soc==1)?0:1;
STAR<=(soc==1)?S1:S0; end
endmodule
```

Fase di esecuzione di INT operando

```
int:   begin   A23_A0<=SP-4;   SP<=SP-4;   {APP3,APP2,APP1,APP0}<={F,IP};
F<='H00; MJR<=int1; STAR<=writeL; end
int1: begin A23_A0<=IDTP+{SOURCE,3'B000}; MJR<=int2; STAR<=readM; end
int2: begin IP<={APP2,APP1,APP0}; STAR<=fetch0; end
```

INT operando (con meccanismo di protezione)

```
int: begin SP<=(F[5]==1)?PMSP:SP; PMSP<=(F[5]==1)?SP:PMSP;
STAR<=int1; end
int1: begin A23_A0<=SP-4; SP<=SP-4; {APP3,APP2,APP1,APP0}<={F,IP};
F<='H00; MJR<=int1; STAR<=writeL; end
int2: begin A23_A0<=IDTP+{SOURCE,3'B000}; MJR<=int2; STAR<=readM; end
int3: begin IP<={APP2,APP1,APP0}; STAR<=fetch0; end
```

Fase di esecuzione di IRET

```
iret: begin A23_A0<=SP; SP<=SP+4; MJR<=iret1; STAR<=readL; end
iret1: begin {F,IP}<={APP3,APP2,APP1,APP0}; STAR<=fetch0; end
```

IRET (con meccanismo di protezione)

```
iret: begin A23_A0<=SP; SP<=SP+4; MJR<=iret1; STAR<=readL; end
iret1: begin {F,IP}<={APP3,APP2,APP1,APP0}; STAR<=iret2; end
iret2:     begin     SP<=(F[5]==1)?PMSP:SP;     PMSP<=(F[5]==1)?SP:PMSP;
STAR<=fetch0; end
```

Fase di esecuzione di SUB (DP), AL

```
aluAL: begin AL<=alu_result(OPCODE,SOURCE,AL); F<={F[7:4],
alu_flag(OPCODE, SOURCE, AL)}; STAR<=fetch0; end

//se viene richiesta con il registro AH basta sostituirlo al posto di AL
```

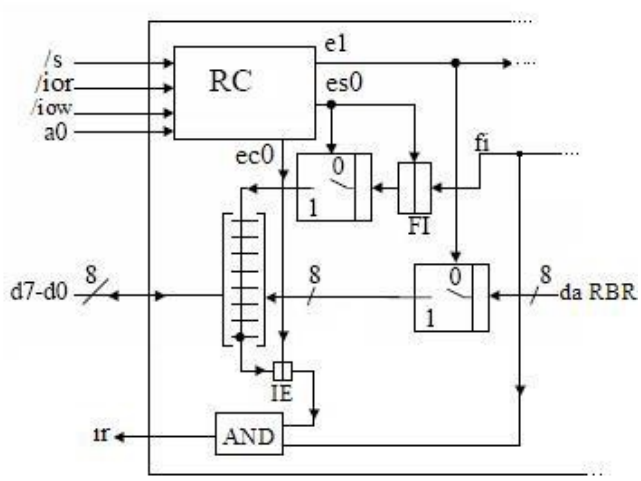
Quando e come viene accettata dal Processore una richiesta di interruzione esterna? (descrizione in verilog)

Un interruzione esterna viene accettata quando IF=1 (elemento in posizione 4 del registro dei flag).

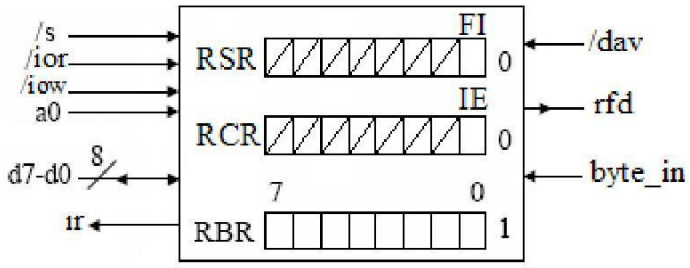
```
test_intr: begin DIR<=0; STAR<=((intr & F[4])==0)?fetch0:pre_tipo0; end
pre_tipo0: begin INTA<=1; STAR<=(intr==1)?pre_tipo0:pre_tipo1; end
pre_tipo1: begin SOURCE<=d7_d0; INTA<=0; STAR<=int; end
```

Ingresso dati ad interruzione di programma: modifiche all'interfaccia di ingresso e connessione dell'interfaccia al controllore e del controllore di processore. Gestione software all'ingresso dati

Le modifiche relative alla struttura interna dell'interfaccia ed alla visione funzionale sono:



La visione funzionale cambia così:



Dove RC è:

/s	/ior	/iow	a0	e1	es0	ec0
0	0	1	0	0	1	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
	default			0	0	0

RBR e RSR hanno la stessa funzione che avevano a controllo di programma,mentre RCR è accessibile al microprocessore in solo scrittura ed ha come unico elemento significativo è il bit 0 che si chiama IE. Se IE è uguale a 0 non si possono emettere richieste di interruzione, mentre se IE è uguale a 1 l'interfaccia è abilitata ad emettere richieste di interruzione. L'esecuzione da parte del processore dell'istruzione IN RBR_offset, AL altre a provocare il trasferimento di un byte utile da RBR ad AL, provoca il resettamento di FI e quindi di ir. Quindi tale istruzione rappresenta per l'interfaccia la notifica software che il servizio, richiesto con l'interruzione, è stato espletato.

Gestione Software:

```
void main () {
#define N
un_numero byte
buffer_in[N];
...
save_into_registers(N,&buff
er_in[0]); asm("INT $250");
...
altre elaborazioni
...
asm("INT $251");
```

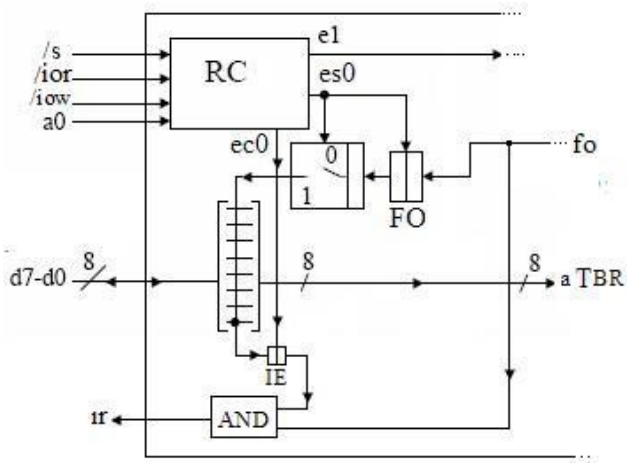
```
...
utilizzo dei dati
}
```

L'intero pacchetto software per l'ingresso dati a controllo di programma può essere strutturato come segue:

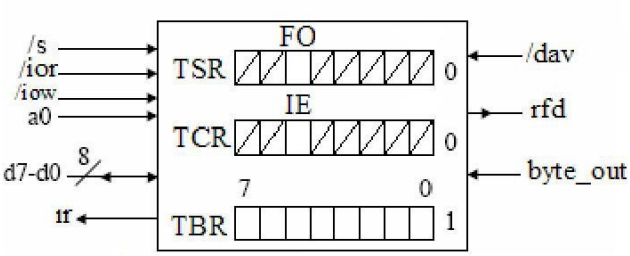
```
#define RCR_offset;
#define RBR_offset;
word counter_in;
byte* pointer_in;
byte semaforo_in;

#define rosso=0;
#define verde=1;

void_interrupt_service_subroutine_250_12_251(){
start_in:
      load_from_registers(&counter_in,&pointer_in);
semaforo_in=rosso;
outport(RCR_offset,0x01);
asm("IRET");
driver_in:
      *pointer_in=inport(RBR_offset);
      pointer_in++;
      counter_in--;
            if(counter_in===0){
            outport(RCR_offset,0x00);
            semaforo_in=verde;}
      asm("IRET");
wait_in:
      asm("STI");
      do{}while(semaforo_in==rosso);
      asm("IRET");
}
```

Interfaccia parallela di uscita gestibile ad interruzione di programma: visione dei registri, gestione software dell'uscita dati ad interruzione di programma modifiche all'interfaccia gestibile a controllo di programma

Le modifiche relative alla struttura interna dell'interfaccia ed alla visione funzionale sono



La visione funzionale cambia così:



Dove RC è:

/s	/ior	/iow	a0	e1	es0	ec0
0	0	1	0	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
	default			0	0	0

TBR e TSR hanno la stessa funzione che avevano a controllo di programma,mentre TCR è accessibile al processore soltanto in scrittura ed ha come unico elemento significativo è il bit 5 che si chiama IE. Se IE è uguale a 0 l'interfaccia non è abilitata a richiedere interruzioni al controllore e va gestita a controllo di programma, mentre se IE è uguale a 1 l'interfaccia è abilitata ad emettere richieste di interruzione mettendo a 1 ir. L'esecuzione da parte del processore dell'istruzione OUT AL, TBR_offset oltre a provocare il trasferimento di un byte utile da AL a RBR provoca il resettamento di FO e quindi di ir. Quindi tale istruzione notifica all'interfaccia che il servizio richiesto è stato espletato.

Gestione software

```
main () {
#define N
un_numero byte
buffer_out[N];
...
caricamento in buffer dei byte da emettere
...
save_into_registers(N,&buffer
_out[0]); asm("INT 252");
...
altre elaborazioni
```

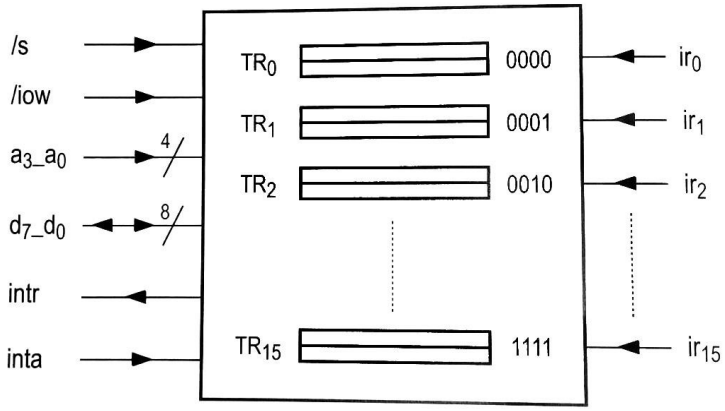
```
...
asm("INT 253");
...
}
```

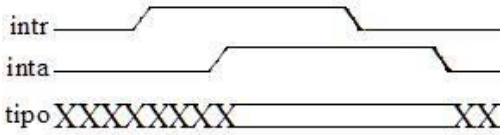
L'intero pacchetto software per l'uscita dati a controllo di programma può essere strutturato come segue:

```
struct{
#define TCR_offset…
#define TBR_OFFSET…
word counter_out;
byte* pointer_out;
byte semaforo_out;

#define rosso o
#define verde 1

void interrupt_service_subroutine_252_13_253(){
start_out:
      load_from_registers($counter_out, &pointer_out);
      semaforo_out=rosso;
      outport(TCR_offset,0x20);
      asm("IRET");

driver_out:
      outport(TBR_offset,*pointer_out);
      pointer_out++;
      counter_out--;
      if(counter_out==0){
            outport(TCR_offset,0x00);
            semaforo_out=verde;}
      asm("IRET");

wait_out:
      asm("STI");
      do{}while(semaforo_out==rosso);
      asm("IRET");
}
```

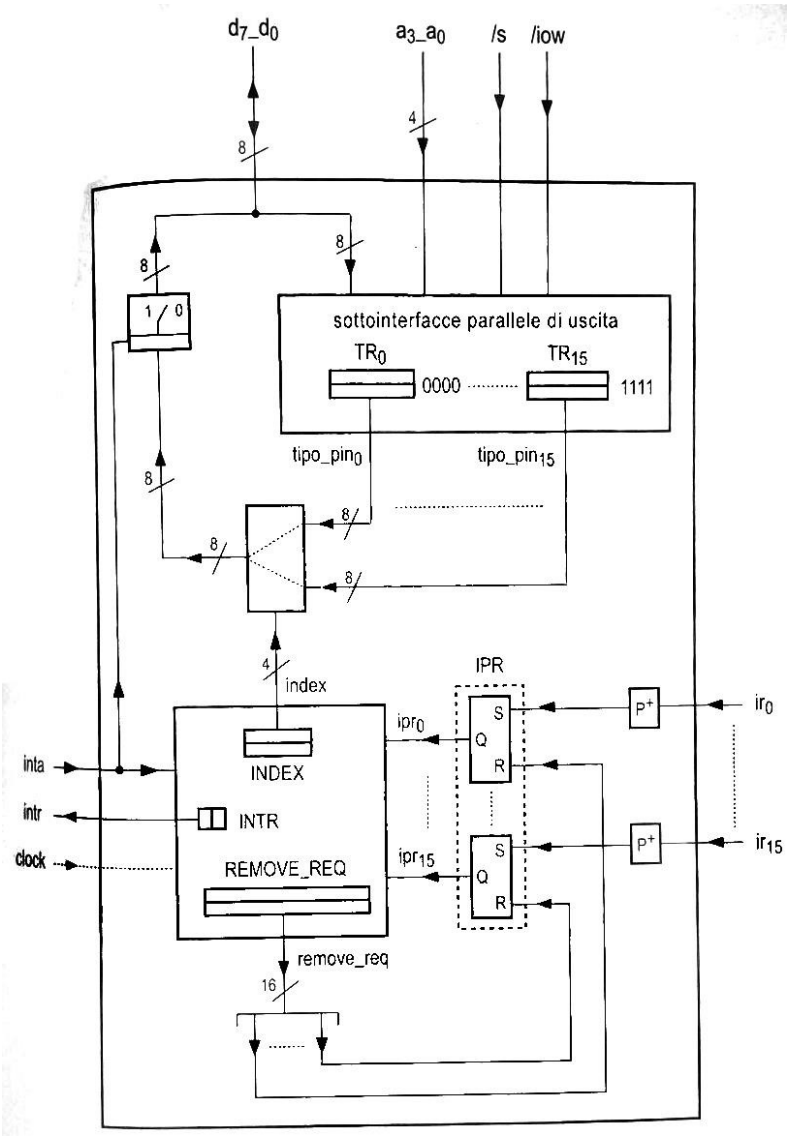
Controllore di interruzione visione funzionale con registri, struttura interna e istruzioni che lo riguardano.

Visione funzionale e piedinatura:





intr=1 vuol dire che si ha una richiesta di interruzione. inta=1 implica che il microprocessore può accettarla. intr=0 implica che il tipo è pronto. inta=0 che il tipo è stato prelevato



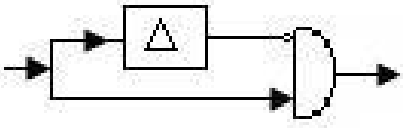
Il microprocessore esamina lo stato di intr solo quando ha finito l'esecuzione di una istruzione e prima della fase di chiamata di un'altra istruzione. La richiesta viene accettata solo se FI = 1.

⇓ **Struttura interna**

Descrizione della sottorete sequenziale sincronizzata interna al controllore

```
module Sottorete_interna_al_Controllore (remove_req, intr, inta,
ipr15_ipr0, index, clock, reset_)
input     clock,reset_;
input     inta;
input     [7:0]  ipr15_ipr0;
output intr;
output [3:0]     index;
output [15:0]     remove_req;
reg INTR; assign intr=INTR;
reg [3:0] INDEX; assign index=INDEX;
reg [15:0] REMOVE_REQ; assign remove_req=REMOVE_REQ;
reg [1:0] STAR; parameter S0 = 0, S1= 1, S2 = 2, WS = 3;
always@(reset_==0) begin INTR<= 0; REMOVE_REQ< = 'HFFFF; STAR = WS; end
always@(posedge clock) if (reset_== 1) #3
casex (STAR)
WS : begin STAR <= S0; end // stato di wait per un reset sicuro
S0 : begin REMOVE_REQ <= 'H00; INTR <= ((ipr15_ipr0)=='H0000)? 0: 1;
     INDEX <= priority (ipr15_ipr0); STAR <= (inta == 0)? S0: S1; end
S1 : begin INTR <= 0; REMOVE_REQ<= decode (INDEX); STAR <= S2; end
S2 : begin REMOVE_REQ <='H0000; STAR <= (inta==1)? S2: S0; end endcase

function [3:0] priority;
input 15:0] ipr15_ipr0;
casex(ipr15_ipr0)
'B???????1 : priority = 'B000;
'B??????10 : priority = 'B001;
'B?????100 : priority = 'B010;
'B????1000 : priority = 'B011;
'B???10000 : priority = 'B100;
'B??100000 : priority= 'B101;
'B?1000000 : priority = 'B110;
'B10000000 : priority = 'B111;
'B00000000 : priority = 'BXXX;
endcase
endfunction

function [15:0] decoder;
input [3:0] index;
casex(INDEX)
'B000 : decoder = 'B00000001;
'B001 : decoder = 'B00000010;
'B010 : decoder = 'B00000100;
'B011 : decoder = 'B00001000;
'B100 : decoder = 'B00010000;
'B101 : decoder = 'B00100000;
'B110 : decoder = 'B01000000;
'B111 : decoder = 'B10000000;
endcase
endfunction
endmodule
```

La rete combinatoria “decode”fornisce in uscita un byte che ha tutti zeri ed un 1 in corrispondenza della richiesta che ha priorità maggiore fra quelle presenti in INDEX. La rete combinatoria “priority_encoder” fornisce l’indirizzo della sorgente a più alta priorità fra quelle che hanno una richiesta di interruzione memorizzata in IRR. Una possibile struttura per il circuito P^+ atto a generare un impulso di durata quando la sua variabile di ingresso transisce da 0 a 1 è:



Istruzioni per inizializzare il controllore

```
always@(reset_==0) #1
      begin INTR<=0; REMOVE_REQ<='HFFFF; STAR<=WS; end
```

Connettere l’interfaccia in modo tale che il registro …* sia accessibile con le istruzioni OUT AL, 0x2000 e IN 0x2000, AL

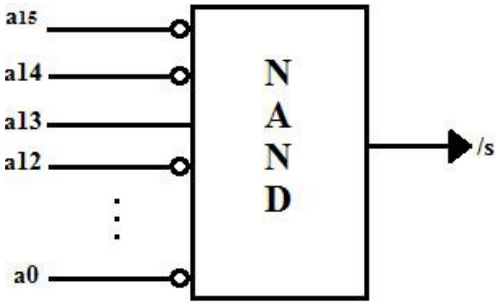
Procedimento generale:

Bisogna progettare una maschera che metta /s=0 quando le variabili a_{15}_a_0 hanno il valore 0x2000. Ragionando sugli indirizzi, 0x2000 è un numero **esadecimale** che vale $16^3*2=8192$.

Poiché $2^{13}=8192$ allora la variabile a_{13} sarà l’unica a valere 1. La risposta corretta a questa domanda è la seguente tabella:

a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	/s
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
altri																1

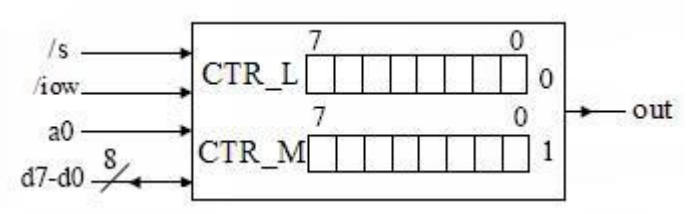
La tabella si può quindi risolvere disegnano una NAND che prende in ingresso tutte le variabili a negate eccetto la a_{13} che viene presa diretta.



***NB:** il registro richiesto può variare.

Visione funzionale,struttura e gestione dell'interfaccia TIMER ad interruzione di programma.

Visione funzionale:

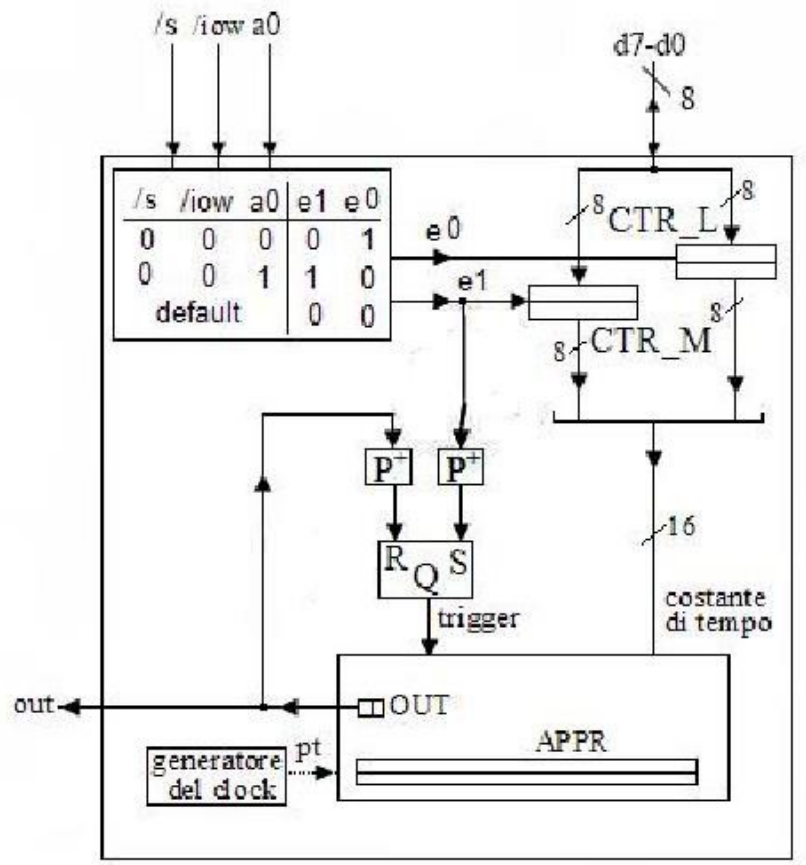


Il contenuto dei due registri, detto costante di tempo, viene interpretato dal timer come un numero naturale a 16 bit. Il processore può accedervi solo in scrittura.

Caricamento della costante di tempo:

outport (offset_di_CTR_L, constant_time_least_significant_byte);
outport (offset_di_CTR_M, constant_time_least_significant_byte);

Struttura interna:



Bibliografia

[1] Paolo Corsini: *Dalle porte and or not al sistema calcolatore: un viaggio nel mondo delle reti logiche in compagnia del linguaggio Verilog*, Edizioni ETS, Pisa, 2014.

[2] http://www.iet.unipi.it/g.stea/RetiLogiche/materiale.html