RETI LOGICHE

PRIMA PROVA SCRITTA DEL 03-06-1999

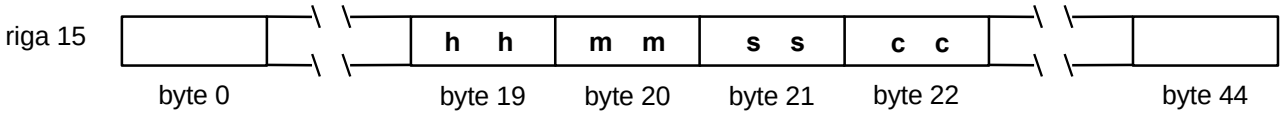
STUDENTE:_____ DOCENTE:_____

Progetto:

Sincronizzatore Televideo → Microprocessore: TLVSYNC

Specifiche funzionali:

Un sistema teletext utilizza le righe 8..15 (non visualizzate, perché comprese nell'intervallo di cancellazione verticale) di un segnale televisivo per diffondere informazioni testuali (televideo). Nella trama video organizzata su 625 righe consecutive per quadro e 25 quadri al secondo, ogni riga video di tipo teletext trasporta un pacchetto di 360 bits, corrispondenti a 45 bytes. Sui bytes 19, 20, 21 e 22 del pacchetto della riga 15 di ogni quadro viene trasmessa una informazione in BCD rappresentante ora (hh), minuto (mm), secondo (ss) e centesimo di secondo (cc) rispettivamente (codice temporale), come delineato in figura (il pacchetto inizia con il byte 0, il byte inizia con il bit più significativo).



Alla periferica TLVSYNC è affidato il compito di estrarre l'informazione di cui sopra dalla trama video e di comunicarla periodicamente (T=1 sec.) al processore, che la utilizzerà come riferimento su cui sincronizzare le proprie attività interne.
Oltre alla linea dati seriale SDA su cui vengono presentati i bit, si suppongano disponibili in ingresso a TLVSYNC una linea di clock SCK per la sincronizzazione dei bit, e una linea PSYNC per la sincronizzazione dei pacchetti, attiva per due periodi di SCK all'inizio del pacchetto della riga 8 e per un periodo di SCK all'inizio dei pacchetti delle righe i=9..15.
La periferica TLVSYNC effettua le seguenti attività:
1. si sincronizza con le trame video;
2. estrae il codice temporale per renderlo disponibile in un registro a larghezza 32 bits accessibile in lettura da PD32;
3. ad ogni nuovo secondo, lancia un interrupt al processore.

In risposta il PD32 preleva il codice temporale dalla periferica, mappata all'indirizzo 5Ah, e lo trasferisce in una locazione di memoria longword TIMESTAMP.

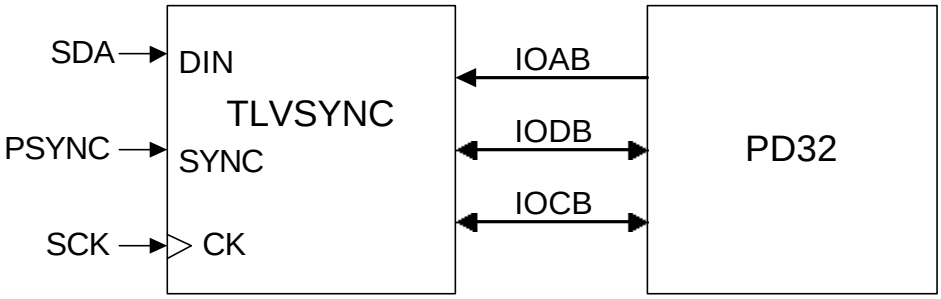
Si richiede:

- lo schema a blocchi funzionali di TLVSYNC ed il diagramma di temporizzazione dell'interfaccia con la linea seriale di ingresso;
- lo schema logico di TLVSYNC;
- la routine assembler PD32 di servizio della periferica.

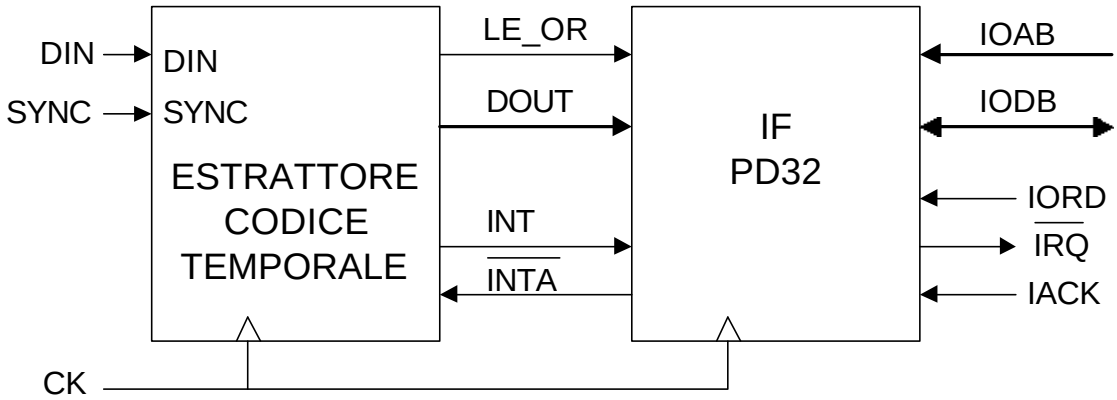
Note/suggerimenti:

- Nel codice temporale relativo ad un nuovo secondo il campo cc può non essere nullo.

TLVSYNC: sistema esterno



TLVSYNC: schema a blocchi



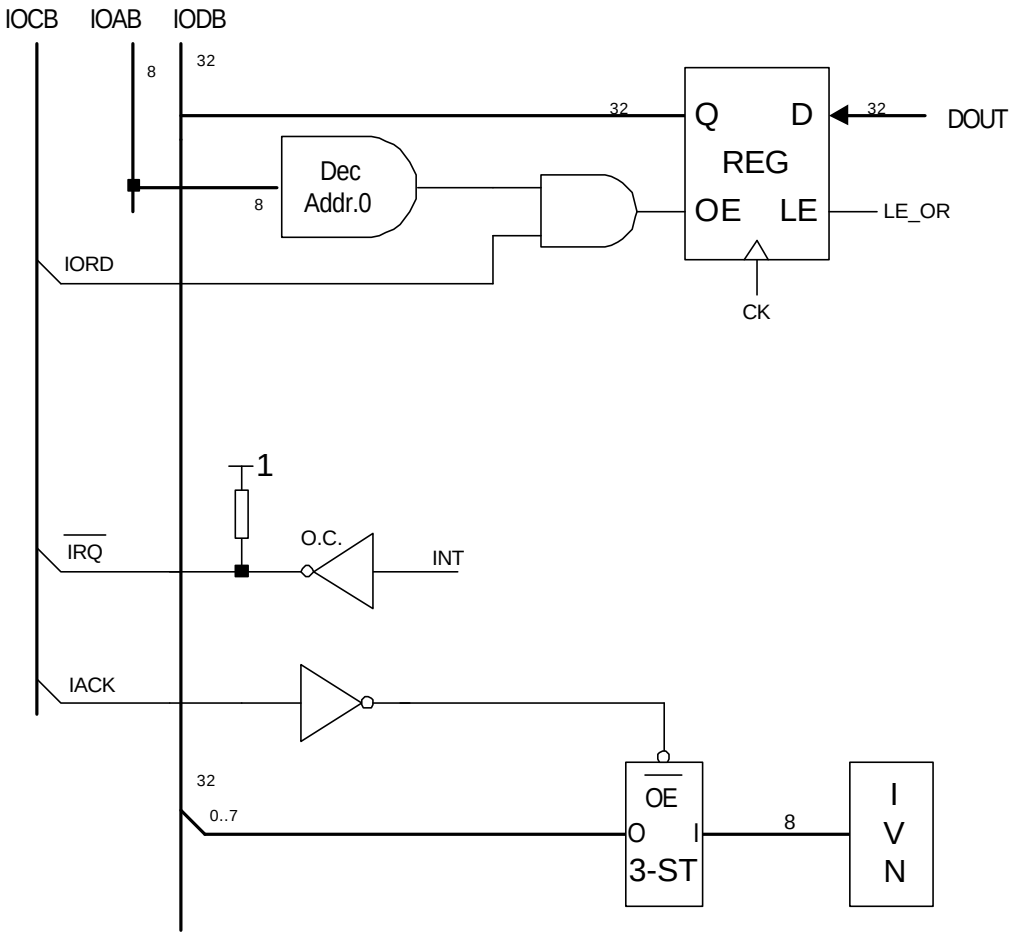
Note

Periferica di tipo input (DOUT).

In questo caso particolare il dato (codice temporale) viene prodotto dalla periferica periodicamente, senza la richiesta del micro; pertanto non c'è bisogno del flip-flop di handshake: la periferica scrive il dato nel registro di interfaccia indipendentemente dall'effettiva lettura del dato precedente da parte del micro.

La periferica elabora il segnale DIN in tempo reale con il clock dato dal segnale di sincronizzazione (SCK) dei bit di DIN.

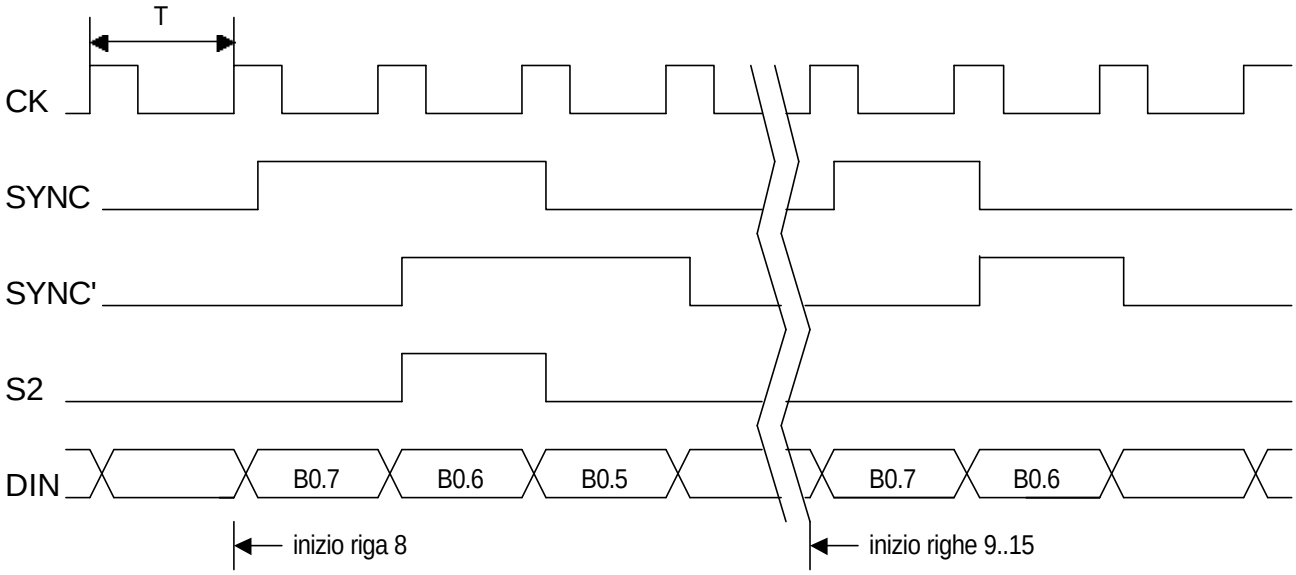
TLVSYNC: IF PD32



Driver assembler

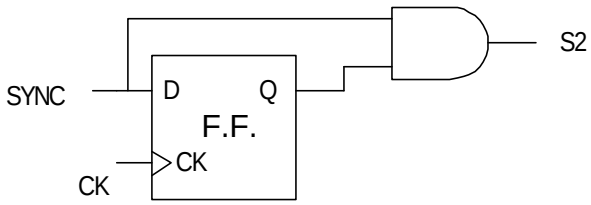
```
main:
...
TIMESTAMP DL 0
. . .
DRIVER 10h,2000h
INL 05Ah,TIMESTAMP
RTI
```

TLVSYNC: temporizzazioni

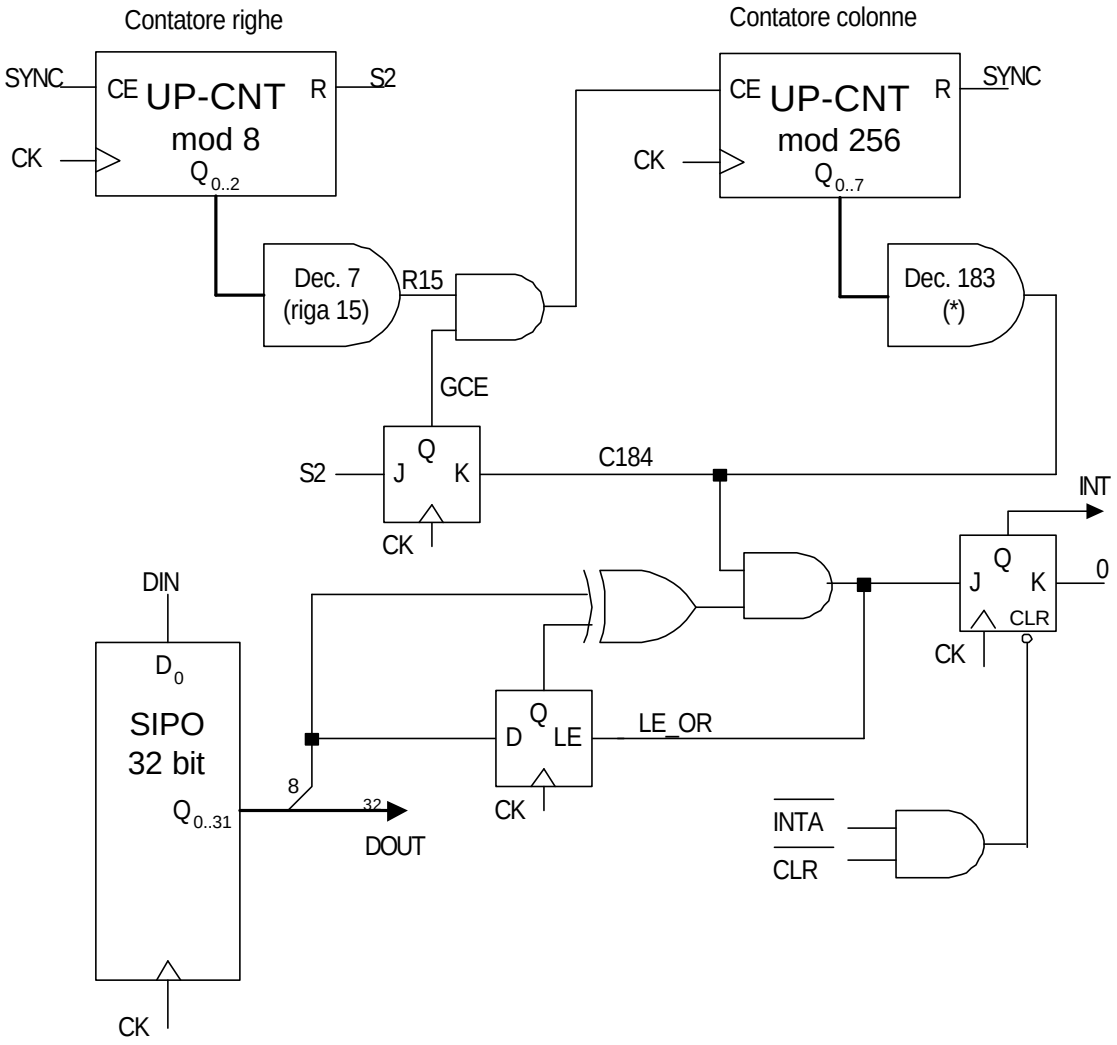


Calcolo del circuito rilevatore di inizio riga 8

S2 = SYNC SYNC' con SYNC'(t) = SYNC (t-T)

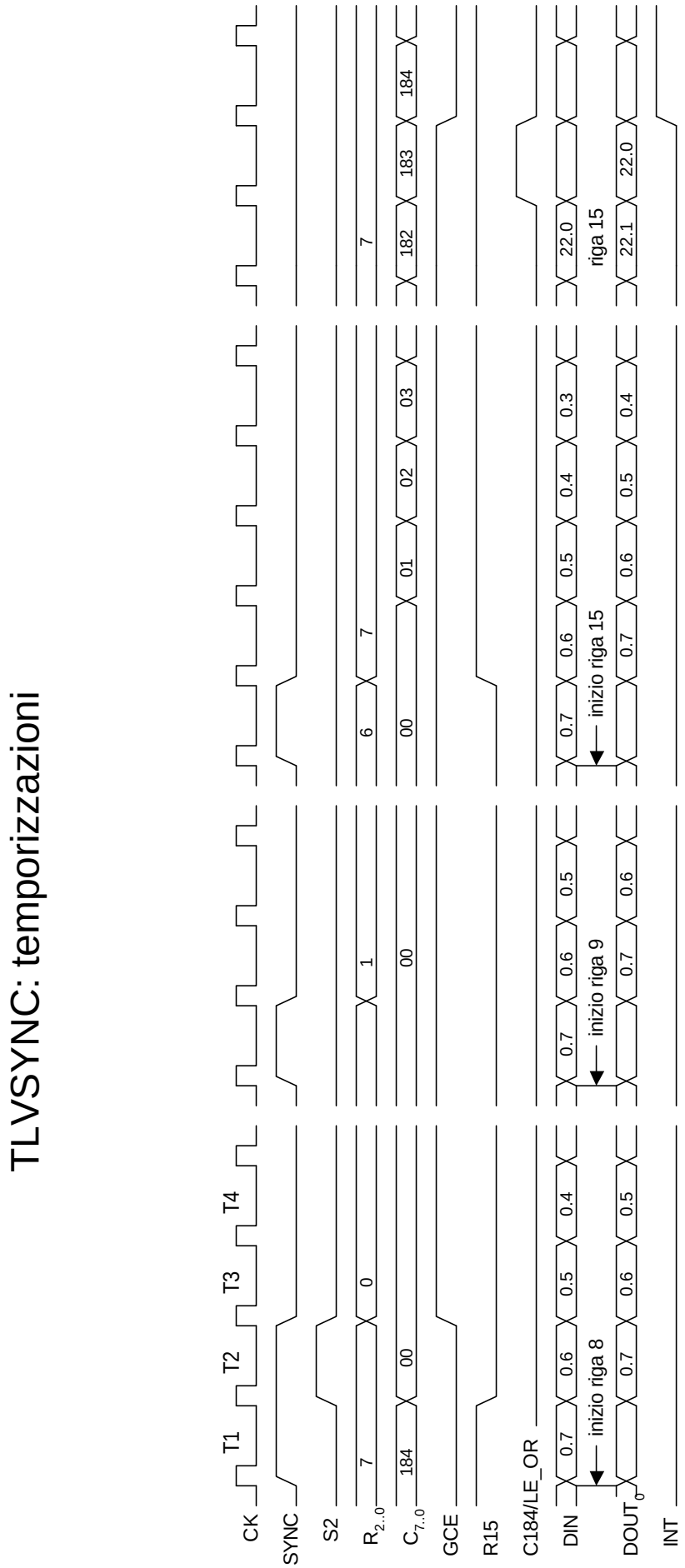


TLVSYNC: estrattore



(*) colonna 183 = 23 x 8 - 1: cfr diagramma di temporizzazione.

TLVSYNC: temporizzazioni



TLVSYNC / ESTRATTORE / COMMENTI

1 Notare che il contatore righe mantiene le uscite a 111 per tutta la durata della riga 15 (e oltre) e tuttavia il suo TC (non indicato) andrà a 1: 1) per un solo periodo di CK (SYNC), 2) all'arrivo dell'impulso SYNC che segnala l'inizio della riga successiva alla 15; infatti, TC è condizionato internamente dall'ingresso CE. Per questa ragione è stata predisposta la decodifica esterna dello stato 111 mediante una porta AND; in questo modo il segnale R15 sarà bloccato a 1 per tutta la riga 15 e abiliterà al conteggio il contatore colonne.

2 Il contatore righe nel secondo stato di attività di SYNC (inizio riga 8: cfr. diagramma di temporizzazione, ciclo T2) si trova attivi CE e R: come è ragionevole, si suppone R prevalente, e pertanto in T3 il contatore si azzera.

3 Il flip-flop SR sincrono con ingresso S2 serve a definire una finestra temporale (bit GCE) di abilitazione al conteggio del contatore colonne: la finestra viene aperta all'inizio della riga 8 (S2) e chiusa al termine della riga 15 (bit C184); in realtà il contatore colonne è disabilitato a contare prima della riga 15 dal segnale R15, e quindi in definitiva il contatore colonne evolve soltanto all'interno della riga 15 (cfr. diagramma di temporizzazione). La finestra di abilitazione è indispensabile per evitare che dopo la riga 15 il contatore colonne continui a contare (in evoluzione libera, cioè mod 256) e perciò vada ad attivare periodicamente la linea C184 (producendo caricamenti aleatori con dati casuali nel registro di interfaccia); questa linea deve attivarsi periodicamente 25 volte al secondo, soltanto in corrispondenza della ricezione di un pacchetto dati.

4 Lo shift register di tipo SIPO ha l'unica funzione linea di ritardo (digitale a 32 bit) per poter trattenere sempre gli ultimi 32 bit transitati sulla linea di ingresso; quando l'intero codice temporale sarà allineato nel SIPO e includerà una variazione nel campo secondi rispetto al pacchetto precedente, allora l'intera stringa di 32 bit sarà caricata nel registro di interfaccia sul fronte di CK immediatamente successivo.

5 Il rilevamento di un codice temporale con l'aggiornamento del campo secondi è effettuato semplicemente mediante il confronto (porta XOR) del bit 0 del byte 21 del codice temporale in transito (memorizzato nello shift register SIPO) con l'omologo nel registro di interfaccia, replicato per questo scopo in un flip-flop D (quello che contribuisce al secondo ingresso dello XOR); questo flip-flop è necessario perché nell'interfaccia è stato utilizzato un registro con buffer tri-state incorporato, e perciò non è possibile andarne a leggere il contenuto senza interferire con il bus dati I/O del PD32. Se il buffer 3-state fosse esterno al registro, allora il bit di confronto sarebbe prelevato dall'uscita 8 del registro e il flip-flop di appoggio sarebbe ridondante (= eliminato).

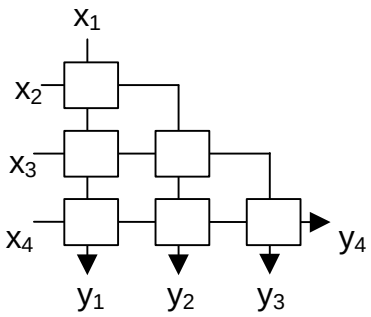
6 Il controllo, decisamente semplice, è distribuito all'interno dello SCA e pertanto non c'è bisogno di uno SCO centralizzato. Questa caratteristica è piuttosto comune nei sistemi preposti all'elaborazione dei segnali in tempo reale.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 3-6-99

Studente:_____ Docente:_____

D1 Si consideri la struttura iterativa di figura:



La generica cella invia in basso il minore dei due ingressi e a destra il maggiore/uguale dei due ingressi: definire le due funzioni logiche della cella e il comportamento ingresso-uscita dell'intera struttura.

D2 Si considerino due stati S e T di due macchine parzialmente specificate e la seguente coppia di sequenze di uscita:
S: 0-110-11001--010-1
T: 01-0-011--0111-001
In base a questa coppia di sequenze è possibile stabilire una relazione tra i due stati?

D3 Una rete sequenziale sincrona è realizzata con una PLA (t_a= 10 nsec) e un registro con flip-flop D (t_{setup}= 3nsec; t_c= 5 nsec). Calcolare la frequenza massima di funzionamento

D4 Definire la sequenza di microistruzioni per eseguire l'istruzione

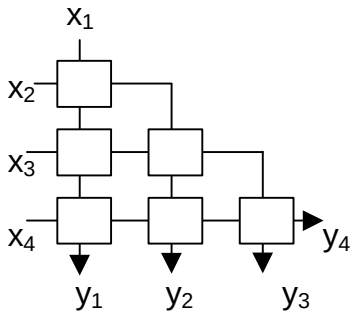
MOVQuick byte, dest

di tipo immediato, dove il dato immediato è un byte contenuto nei bit 16-23 dell'istruzione (si tratta di una istruzione non implementata dal PD32).

D5 Specificare la sequenza completa di operazioni per effettuare un trasferimento in DMA da una periferica alla memoria di un PD32.

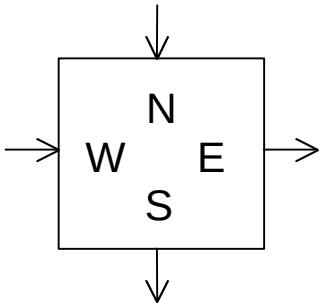
Esercizio (2S19990603-D1)

Si consideri la struttura iterativa di figura:



La generica cella invia in basso il minore dei due ingressi e a destra il maggiore/uguale dei due ingressi: definire le due funzioni logiche della cella e il comportamento ingresso-uscita dell'intera struttura.

Detti N, W i due ingressi e S, E le due uscite della cella generica, il cui simbolo è riportato per comodità nella figura sottostante, si comincia con il trasferire le specifiche verbali su una tavola di verità che leghi S e E alla coppia di variabili indipendenti N e W:



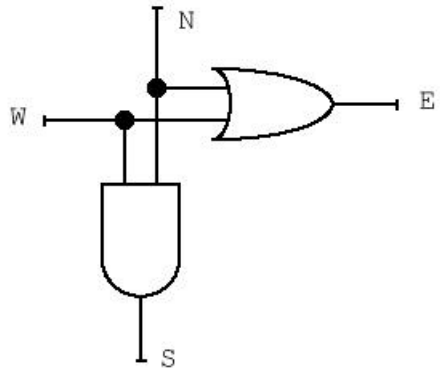
N	W	S	E
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

La sintesi è banale, visto che S e E sono rispettivamente le due funzioni elementari AND e OR delle due variabili di ingresso:

$$S = N \cdot W$$

$$E = N + W$$

Lo schema logico della cella è pertanto il seguente:



Il comportamento della rete può essere descritto formalmente mediante la tavola di verità: la tavola viene riempita con le configurazioni che assume il vettore $y_1\ y_2\ y_3\ y_4$ quando vengono applicati alla rete i vettori $x_1\ x_2\ x_3\ x_4$.

x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1
0	0	1	1	0	0	1	1
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	0	0	1	1
0	1	1	1	0	1	1	1
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	1
1	0	1	0	0	0	1	1
1	0	1	1	0	1	1	1
1	1	0	0	0	0	1	1
1	1	0	1	0	1	1	1
1	1	1	0	0	1	1	1
1	1	1	1	1	1	1	1

Ispezionando la tavola di verità si vede che in tutte le righe il vettore $y_1\ y_2\ y_3\ y_4$ descrive sequenze non decrescenti, con lo stesso numero di 0 (di 1) del vettore $x_1\ x_2\ x_3\ x_4$: Pertanto, l'effetto della rete sul vettore di ingresso è quello di ordinarne gli elementi.

Esercizio (2S19990603-D2)

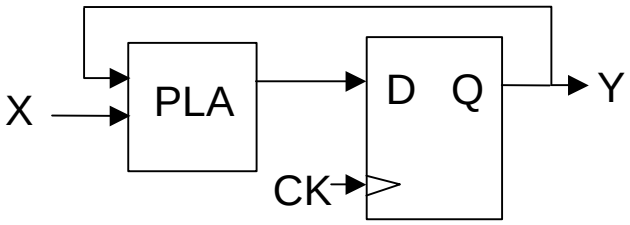
Si considerino due stati S e T di due macchine parzialmente specificate e la seguente coppia di sequenze di uscita:
S: 0-110-11001--010-1
T: 01-0-011--0111-001
In base a questa coppia di sequenze è possibile stabilire una relazione tra i due stati?

Le due sequenze hanno valori logici definiti diversi nella quarta posizione (e poi anche nell'undicesima e nella quattordicesima): se ne deduce che i due stati sono incompatibili.

Esercizio (2S19990603-D3)

Una rete sequenziale sincrona è realizzata con una PLA (t_a = 10 nsec) e un registro con flip-flop D (t_{setup}= 3nsec; t_c= 5 nsec). Calcolare la frequenza massima di funzionamento.

La struttura è quella riportata nella figura.



Il tempo di ciclo è limitato inferiormente da:

$t_{min} = t_c + t_a + t_{setup} = 18$ nsec

assumendo che non ci siano ritardi lungo i collegamenti.

Corrispondentemente la frequenza massima di CK è pari a:

$F_{max} = 1/t_{min} = 1/18$ ns $\approx$ 55 MHz

Esercizio (2S19990603_D4)

Definire la sequenza di microistruzioni per eseguire l'istruzione

MOVQuick byte, dest

di tipo immediato, dove il dato immediato è un byte contenuto nei bit 16-23 dell'istruzione (si tratta di una istruzione non implementata dal PD32).

Il problema presuppone che byte sia prescritto nel campo K dell'istruzione, e che il relativo codice operativo (32 bit) sia stato già caricato nel registro IR (fasi di fetch e decode completate).
I bit di uscita del campo K dell'IR sono collegati al bus interno (tramite un buffer 3-state, ovviamente); infatti, nelle istruzioni di I/O K può rappresentare l'indirizzo della periferica e pertanto deve poter essere caricato nel registro IOAR, e inoltre questo deve poter essere caricato anche da un qualunque registro.
Nel caso più semplice in cui dest sia un registro (Rdest), occorre prevedere la sequenza:
1: IR → T2
2: opcode per scalare i bit 23..16 (di T2) nelle posizioni 7..0; Shifter → Rdest

Se dest fosse una locazione di memoria esterna, occorrerebbe descrivere la sequenza delle micro-operazioni di scrittura in memoria (utilizzo del MAR e del MDR).

Esercizio (2S19990603-D5)

Specificare la sequenza completa di operazioni per effettuare un trasferimento in DMA da una periferica alla memoria di un PD32.

Cfr. “Appunti Integrativi”.

RETI LOGICHE

PRIMA PROVA SCRITTA DEL 17-06-1999

STUDENTE:_____ DOCENTE:_____

Progetto:

Acceleratore steganografico: STEGRAPH

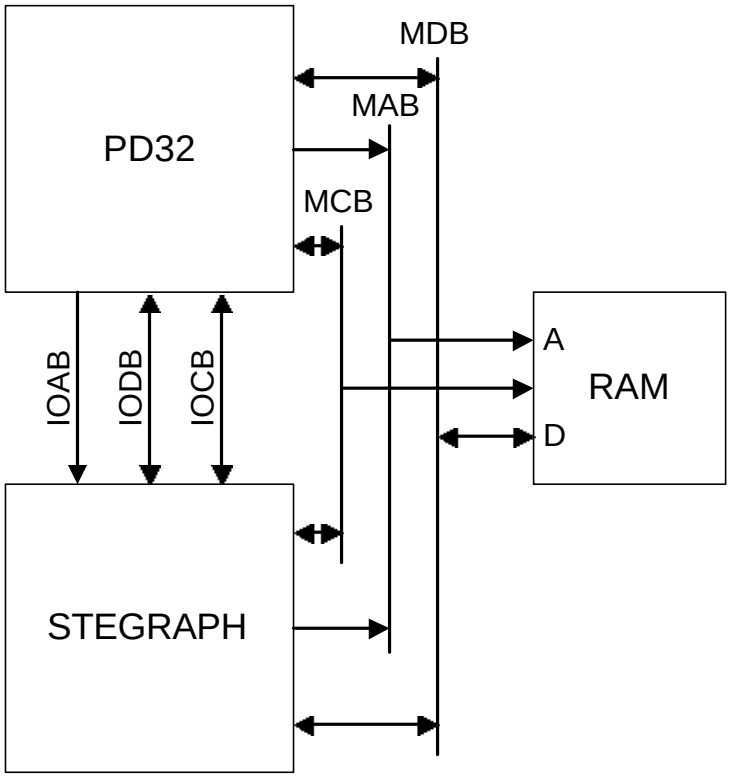
Specifiche funzionali:

STEGRAPH è un dispositivo HW per l'implementazione veloce di una tecnica di protezione dell'informazione basata sull'inserimento dei bit dei caratteri di un file di un testo al posto del bit meno significativo degli elementi di una immagine ("steganografia": l'immagine "steganografata" con tale tecnica "nasconde" il testo, in quanto appare indistinguibile visivamente dall'immagine originale).
L' immagine è rappresentata in memoria PD32 da una matrice di 600 x 800 byte disposti consecutivamente (pixel). Il testo da nascondere nell'immagine è composto da 32 Kbyte.
Il processore comunica alla periferica STEGRAPH gli indirizzi iniziali sia dei dati dell'immagine e del testo predisposti in memoria, sia dell'immagine steganografata.
La periferica STEGRAPH effettua le seguenti attività, operando in DMA:
- alterna il prelevamento di 1 byte del testo e di 8 byte dell'immagine;
- sostituisce ciascuno degli 8 bit di ogni byte di testo al bit meno significativo di 8 byte consecutivi dell'immagine;
- scrive in memoria i byte di immagine modificati;
- al termine dell'elaborazione dell'intero blocco, lancia un interrupt al processore.

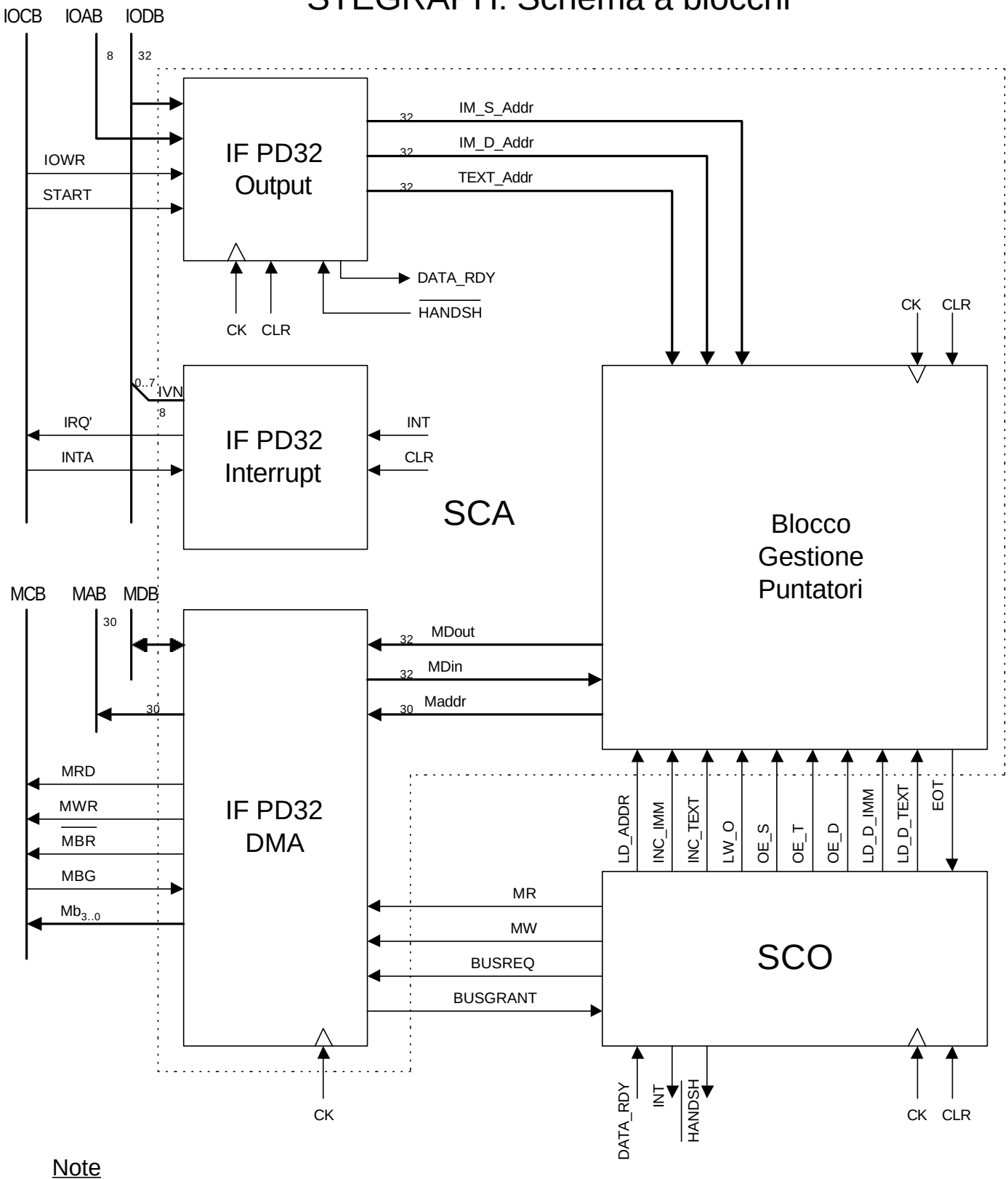
Si richiede:

- lo schema a blocchi funzionali di STEGRAPH e i diagrammi di temporizzazione delle interfacce con il PD32;
- lo schema logico dei sottosistemi SCA e SCO di STEGRAPH;
- il microprogramma dello SCO.

STEGRAPH: sistema esterno



STEGRAPH: Schema a blocchi

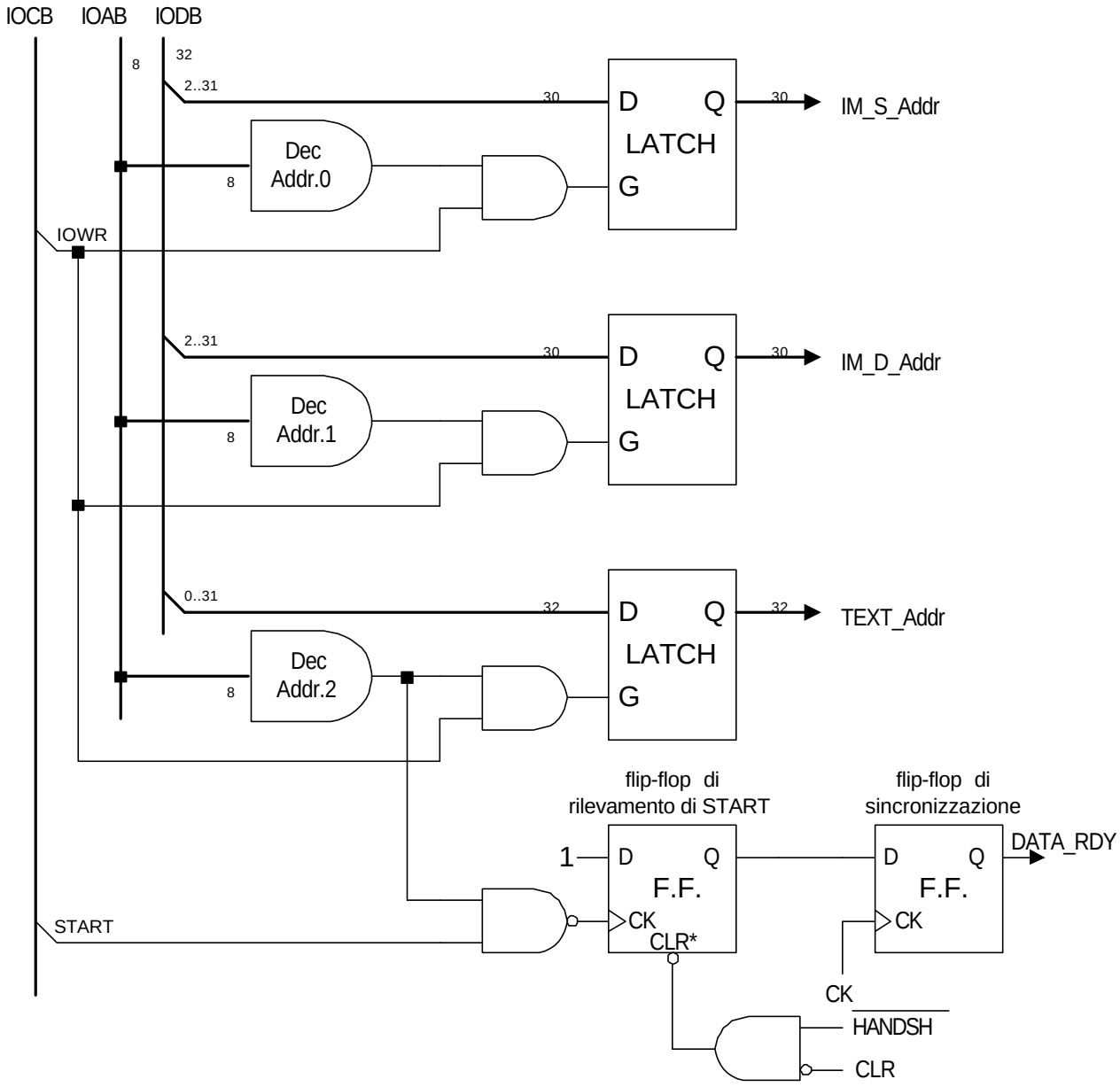


Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.

Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 usa il segnale CLR per azzerare il flip-flo di richiesta di interruzione.

STEGRAPH: IF PD32 - output

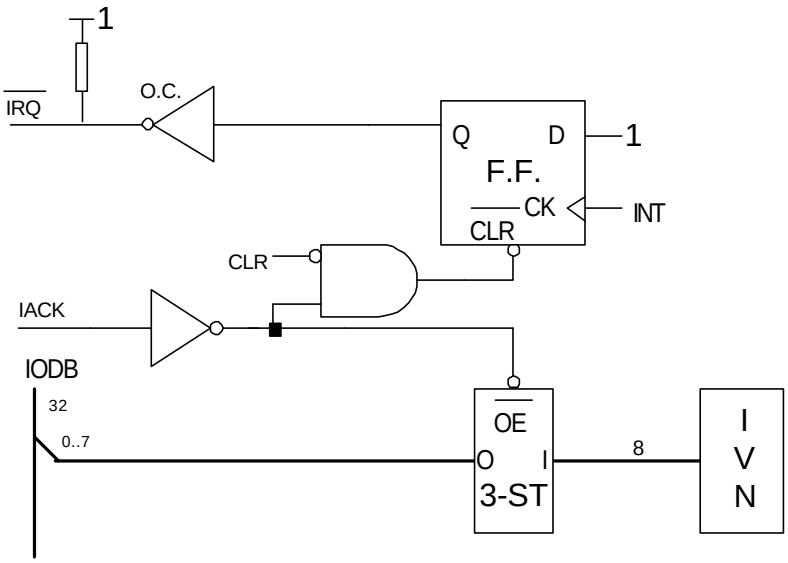


Note

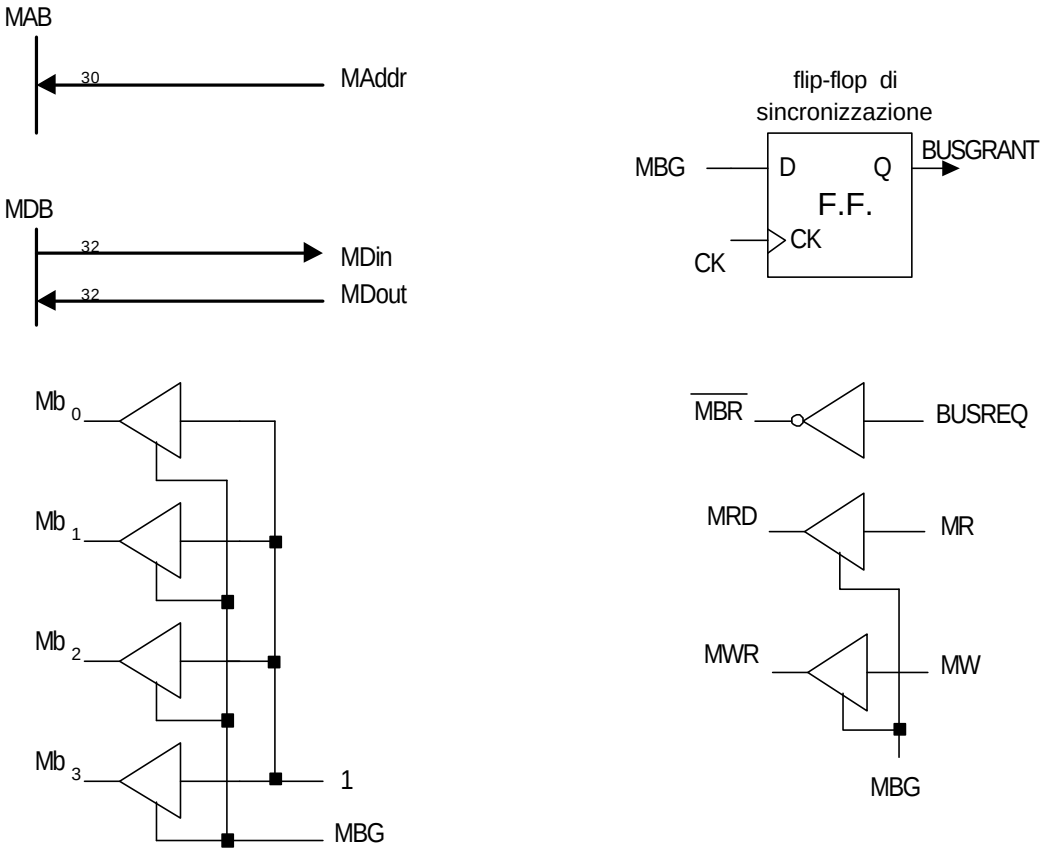
L'indirizzo di START può essere uno qualunque dei tre associati ai tre registri.

Il SW può avviare l'operazione con **START Addr2** dopo avere eventualmente riscritto uno o più registri (le informazioni memorizzate nei registri di interfaccia non vengono alterate dalla periferica STEGRAPH).

STEGRAPH: IF PD32 - interrupt

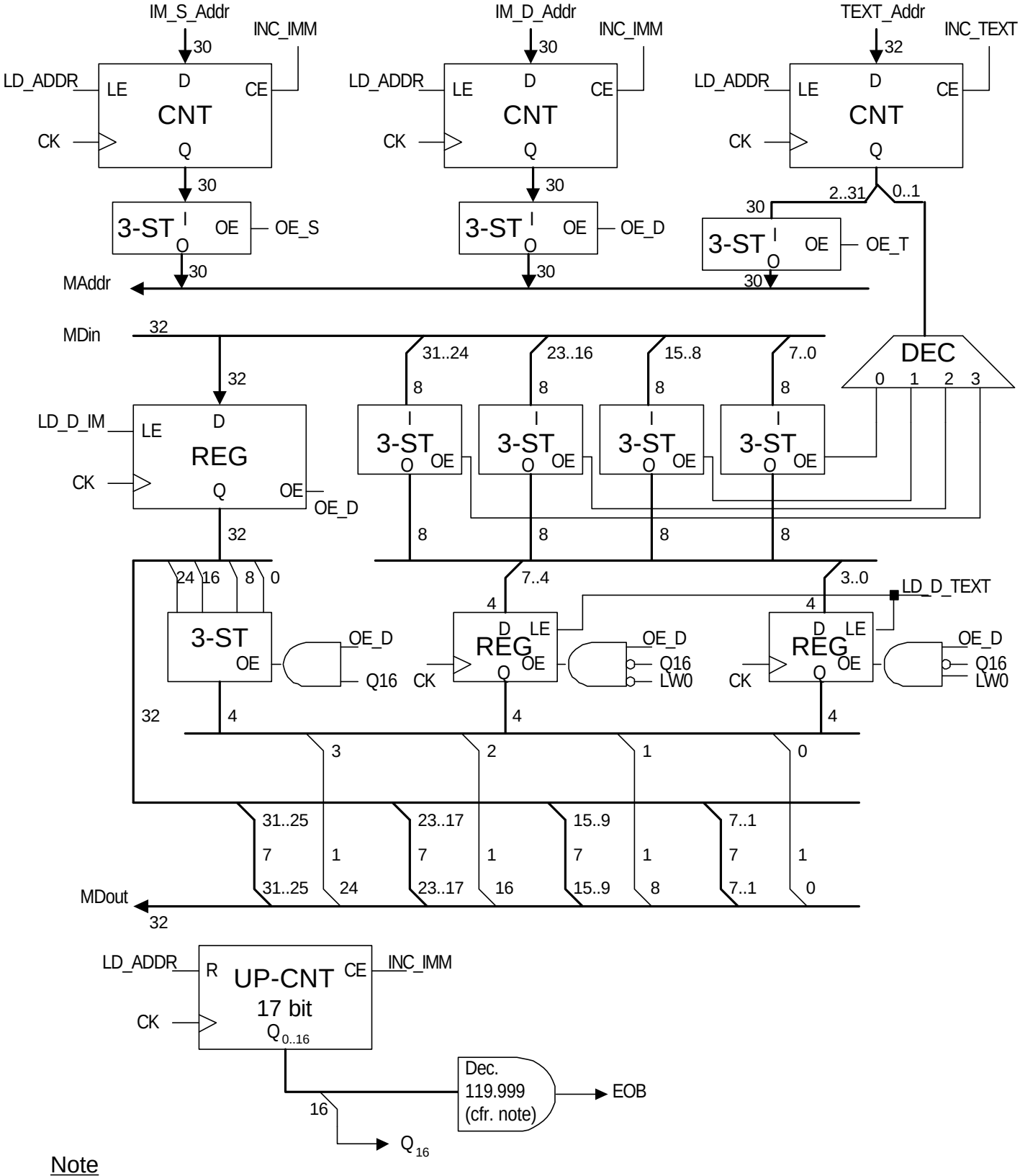


STEGRAPH: IF PD32 - DMA



Si suppone di scrivere dati a LW allineate
(una LW su un indirizzo di riga)

STEGRAPH: blocco gestione puntatori

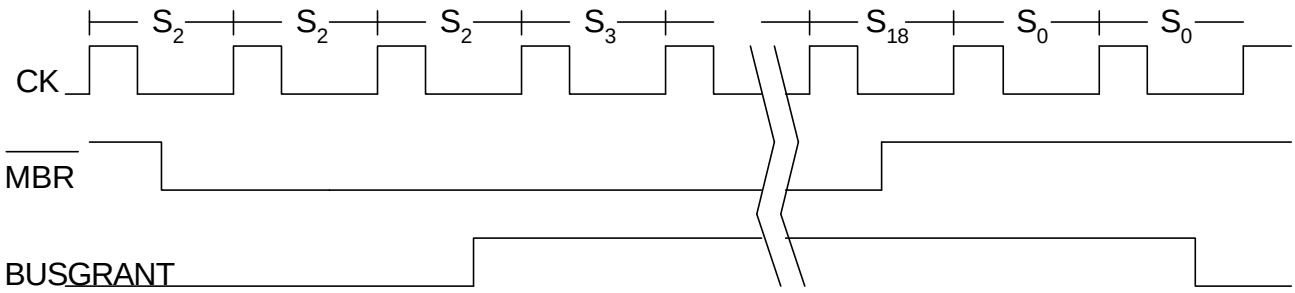


Note

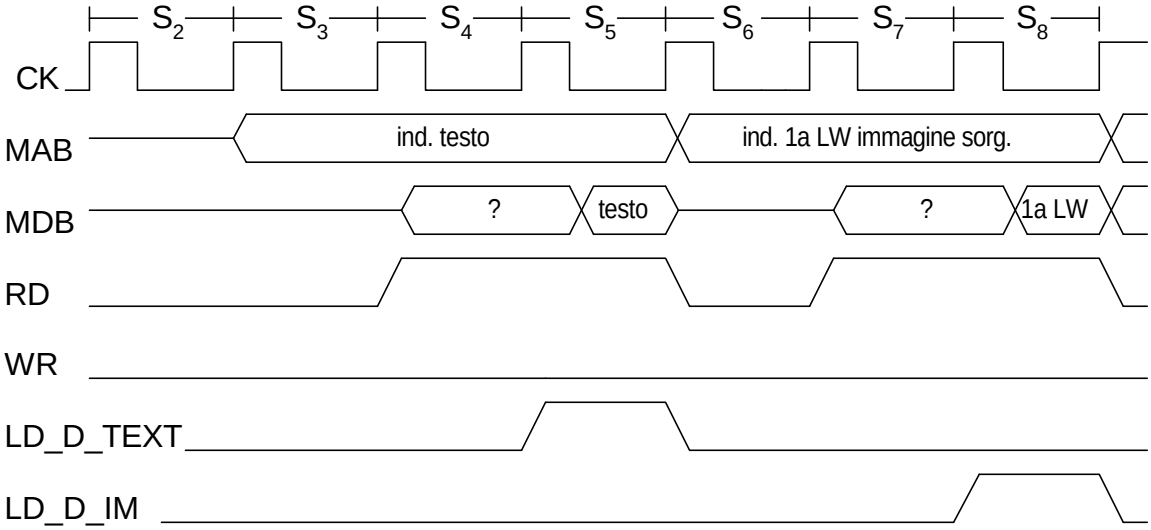
- 800 X 600 / 4 LW = 120000 lw < 2^{17}; ne consegue il conteggio 0..119999.
- 1 byte di testo <--> 2 LW di immagine; pertanto: 32 KB di testo <--> 64 KLW di immagine.
- Il contatore conta le LW (INC_IMM); pertanto il testo è finito quando Q_{16}:0->1; da quel momento in poi, e fino a quando EOB:0->1, le LW di immagine devono essere trasferite senza modifiche.

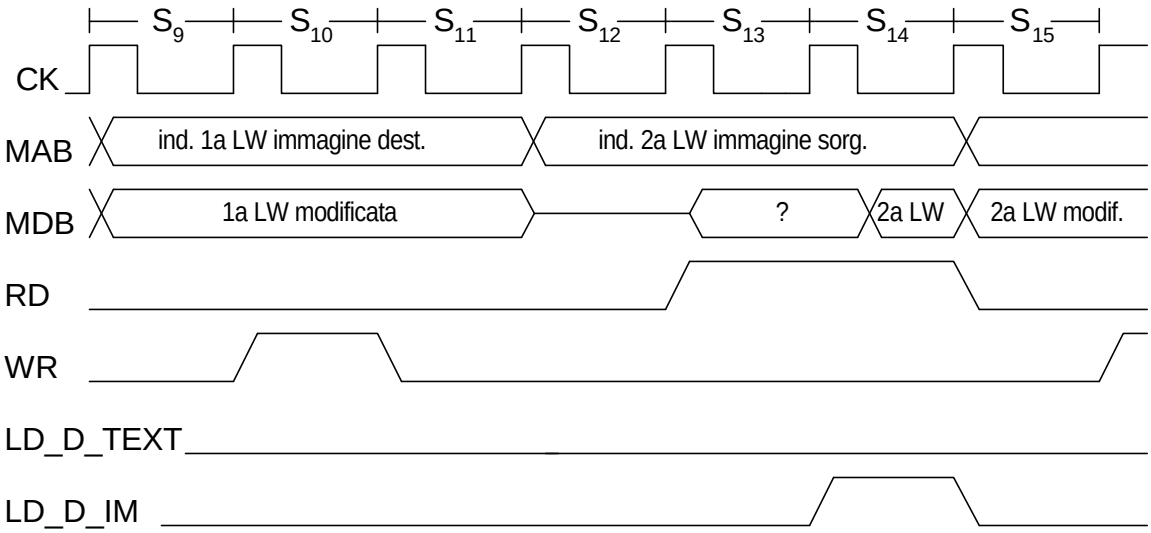
STEGRAPH: temporizzazioni

Richiesta / rilascio bus di sistema e relativi stati dello SCO



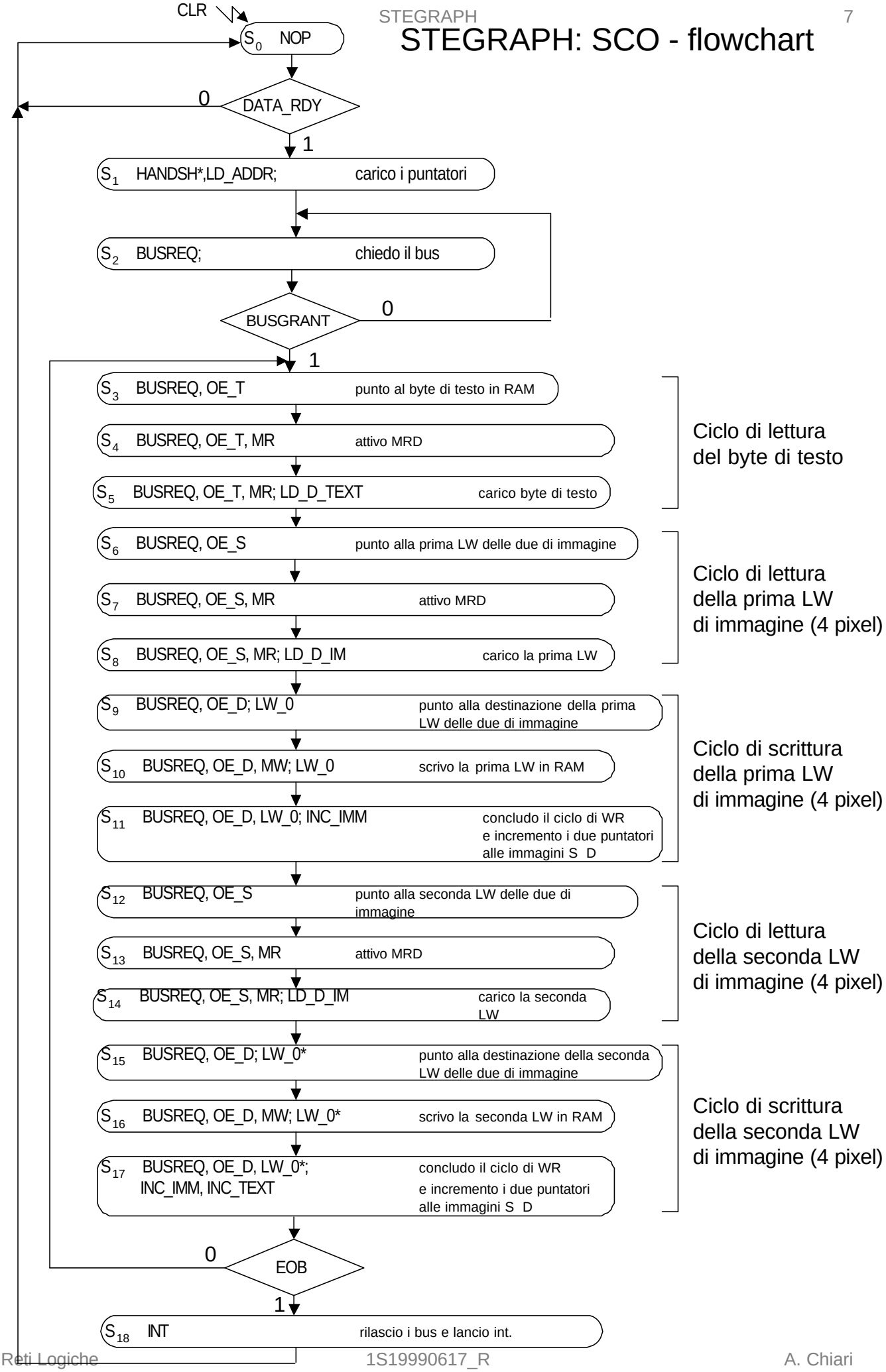
Accessi in RAM e relativi stati dello SCO





I cicli di lettura sono a tre stati (S_3 .. S_5; S_6 .. S_8; S_{12} .. S_{14}), in quanto si suppone che la RAM abbia $2T < t_A < 3T$: in mancanza di specifiche sulla velocità della periferica si suppone di utilizzare nella periferica lo stesso clock del processore.

STEGRAPH: SCO - flowchart



Note

- Il test su EOB viene effettuato sul valore non ancora aggiornato dal comando INC_IMM.

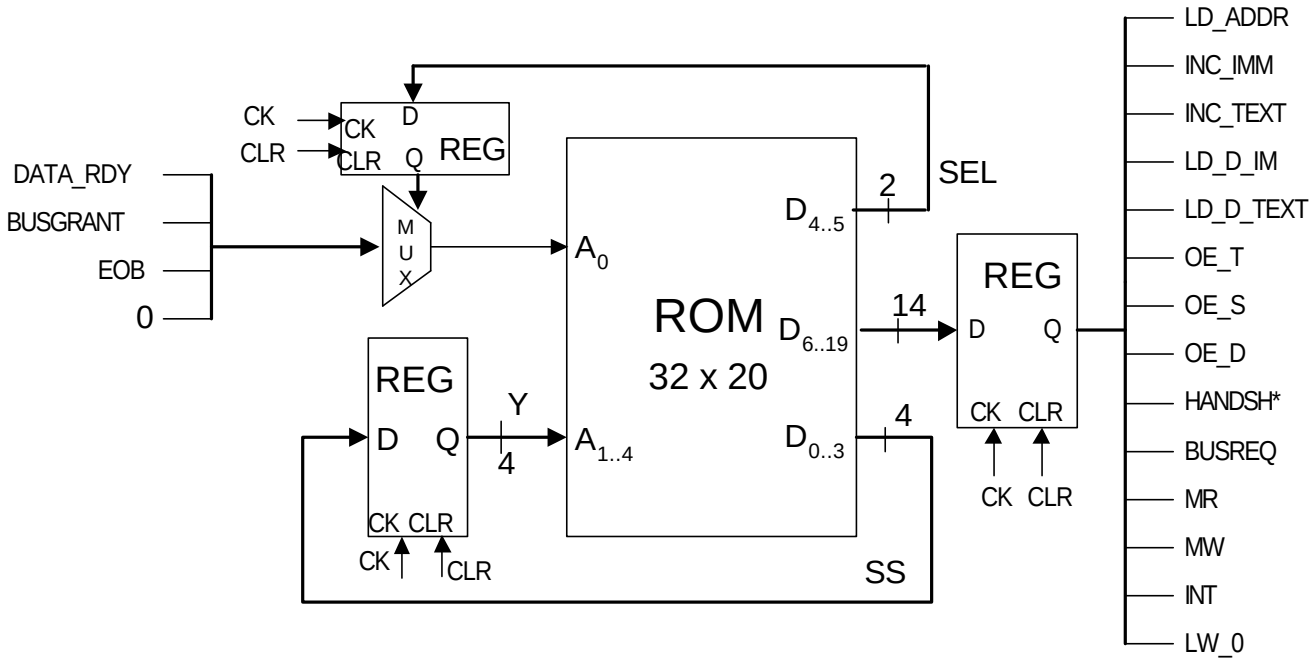
- Si può emettere la richiesta di interruzione quando è ancora BUSGRANT=1 perché INT viene memorizzata in un flip-flop e quindi sarà rilevata dal processore dopo avere ripreso l'uso dei bus (cfr. diagramma gestione DMA / INT).

- Dopo avere emesso INT in S_{18} il controllo torna in S_0 senza aspettare né INTA né BUSGRANT=0: questi due eventi si verificheranno mentre SCO in S_0 aspetta la successiva commutazione di DATA_RDY=1; infatti, il processore potrà riportare DATA_RDY a 1 soltanto dopo avere servito la richiesta di interruzione, con la quale la periferica segnala al processore la conclusione delle attività.

- Va notato che il meccanismo di comunicazione tra il processore e la periferica mediante interruzione rende superflua l'interfaccia busy-waiting, in quanto dopo avere emesso una richiesta di interruzione la periferica ha certamente concluso l'operazione richiesta dal processore e quindi è certamente pronta ad avviare un'operazione successiva (il processore non ha bisogno di effettuare un test sullo stato di pronto della periferica).

STEGRAPH: SCO - struttura HW microprogrammata

Il flow-chart è supportato da un microlinguaggio di tipo 3; scegliendo il modello strutturale di tipo D-Mealy si ottiene la struttura seguente:



RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 17-6-99

Studente:_____ Docente:_____

D1 Si consideri un codice autocorrettore di 12 bit a distanza di Hamming pari a 4; calcolare la probabilità di non rivelare errore in una parola di codice, supposto che la probabilità di errore in un bit h pari a 10^{-7}.

D2 Progettare una macchina sequenziale che accetta in ingresso simboli 0 e 1 e dia una uscita 1 ogni volta che riconosce una sequenza del tipo $01(00)^n1$ con n intero e finito.

D3 Una struttura di calcolo h costituita da tre blocchi combinatori in cascata con tempi di calcolo pari a 20, 40, 20 nsec. Trasformarla in una struttura pipeline in modo da avere tempo di ciclo pari a 25 nsec e tracciare la temporizzazione relativa ai vari stadi.

D4 Un PD32 ha il ciclo di accesso alla memoria di 200nsec. Calcolare di quanto viene rallentato durante l'operazione di trasferimento dati in DMA da un disco che abbia una velocità di lettura/scrittura di 12 Mbit/sec. Si supponga di trasferire dati di 32 bit alla volta.

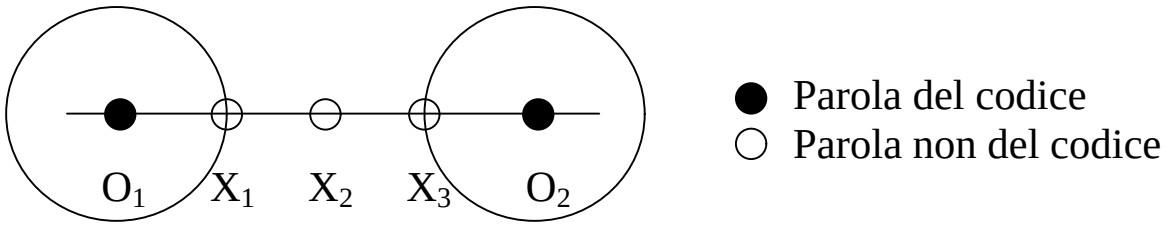
D5 Descrivere la temporizzazione di un sistema SCA-SCO di tipo DMealy-Mealy.

Esercizio (2s19990617-D1)

Si consideri un codice autocorrettore di 12 bit a distanza di Hamming pari a 4; calcolare la probabilità di non rivelare errore in una parola di codice, supposto che la probabilità di errore in un bit è pari a 10^{-7}.

Il codice dato è caratterizzato da h=4; pertanto ha la capacità di rilevare fino a 3 errori, oppure di correggere 1 errore e rilevare fino a 2 errori, a seconda dell'uso che se ne vuol fare.

La disposizione delle parole del codice è schematizzata nella figura, in cui viene anche evidenziato il tipo di azione correttiva (C=1) e rivelatrice (R=2) del codice (R+C=h-1=3).



Per calcolare la probabilità richiesta, si dovranno esaminare le possibili situazioni di errore con riferimento alla capacità di intervento prescelta (rivelazione, correzione) del codice; pertanto, nell'ipotesi di trasmissione della parola O_1, vengono presi in esame i 4 casi relativi alla ricezione di una delle parole X_1, X_2, X_3, O_2:

	Rivelazione (R=3)	Correzione (C=1) + Rivelazione (R=2)
X_1 (1 errore)	SI	SI
X_2 (2 errori)	SI	NO
X_3 (3 errori)	SI	Equivocazione (O_2)
O_2 (4 errori)	Equivocazione	Equivocazione

Nella tabella sono riportate le azioni per i quattro casi, separatamente per i due modi di funzionamento del codice.

Nel funzionamento come rivelatore di errori, il codice non rivelerà 4 (o multipli di 4) errori; infatti, in caso di ricezione di O_2, il codice non potrà che ritenere che O_2 sia anche la parola trasmessa (equivocazione). Pertanto, nell'ipotesi di indipendenza statistica degli errori e trascurando i contributi di ordine inferiore (8 o più errori), la probabilità cercata è approssimata da:

$$p_4 = \binom{12}{4} \cdot p_e{}^4 \cdot (1-p_e)^8$$

Sostituendo $p_e = 10^{-7}$ si trova: $p_4 \approx 0.5 \times 10^{-25}$

Nel funzionamento come correttore, il codice equivocherà già a partire da 3 errori, in quanto correggerà la parola X_3 in O_2; pertanto, anche qui nell'ipotesi di indipendenza statistica degli errori e trascurando i contributi di ordine inferiore (4 o più errori), la probabilità cercata è approssimata da:

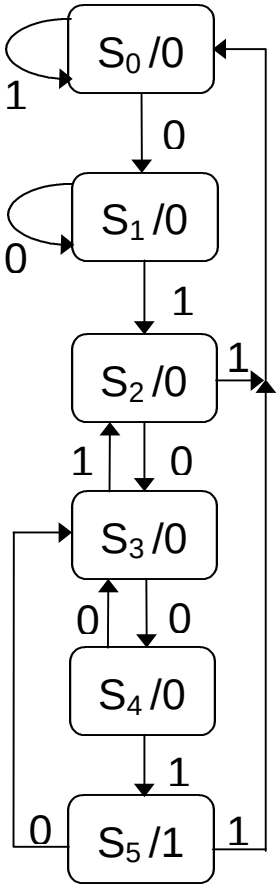
$$p_3 = \binom{12}{3} \cdot p_e{}^3 \cdot (1-p_e)^9$$

Sostituendo $p_e = 10^{-7}$ si trova: $p_3 \approx 0.2 \times 10^{-18}$

Esercizio (2S19990617-D2)

Progettare una macchina sequenziale che accetta in ingresso simboli 0 e 1 e dia una uscita 1 ogni volta che riconosce una sequenza del tipo $01(00)^n1$ con n intero e finito.

La macchina specificata può essere descritta mediante il diagramma degli stati secondo il modello di Moore:



E' da notare che da ogni stato (nodo) si esce con transizioni (archi) associate a entrambe le configurazioni (0 e 1) possibili per l'(unico) ingresso.

L'uscita dallo stato di riconoscimento S_5 per ingresso 0 si ricollega allo stato S_3 perché la macchina possa riconoscere sequenze sovrapposte.

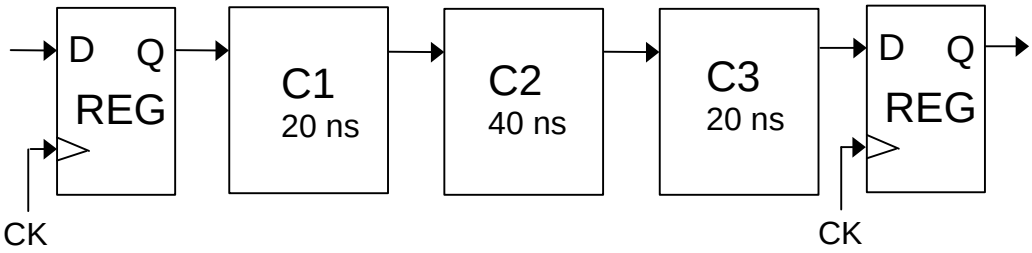
La macchina deve riconoscere una sequenza di lunghezza minima pari a 5 simboli; è stata descritta secondo il modello di Moore ed impiega 6 stati (nodi) connessi in una catena di 5 transizioni (archi), associate al riconoscimento dei 5 simboli; pertanto la macchina è sicuramente minima.

Domanda: Se si utilizzasse il modello di Mealy, quanti stati avrebbe la macchina equivalente?

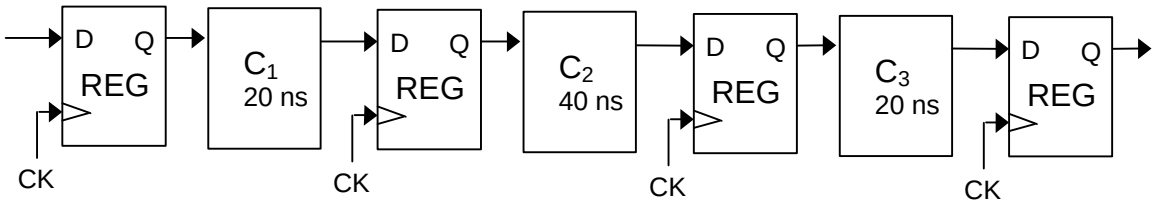
Esercizio (2S19990617-D3)

Una struttura di calcolo è costituita da tre blocchi combinatori in cascata con tempi di calcolo pari a 20, 40, 20 nsec. Trasformarla in una struttura pipeline in modo da avere tempo di ciclo pari a 25 nsec e tracciare la temporizzazione relativa ai vari stadi.

La struttura originale è disposta come segue:



Il tempo di ciclo di ck non può eccedere 80 ns (a cui va inoltre aggiunto il tempo t_{reg} + t_{setup} per tenere conto dei tempi di commutazione e di set-up dei due registri).
La struttura può essere resa più veloce mediante la registrazione dei risultati parziali elaborati dalle tre reti combinatorie, secondo un modello di tipo *pipeline* seriale:



Con questa nuova struttura il tempo di ciclo di ck non può eccedere 40 ns (a cui va ancora aggiunto il tempo t_{reg} + t_{setup} per tenere conto dei tempi di commutazione e di set-up dei due registri confinanti con la rete C_2).
Per ridurre il tempo di ciclo a 25 ns occorre raddoppiare la sola rete C_2: raddoppiare perché $1 < 40/25 < 2$, la sola C_2 perché il tempo di calcolo individuale di C_1 e C_3 è minore di 25 ns. La struttura pipeline definitiva è di tipo serie-parallelo:

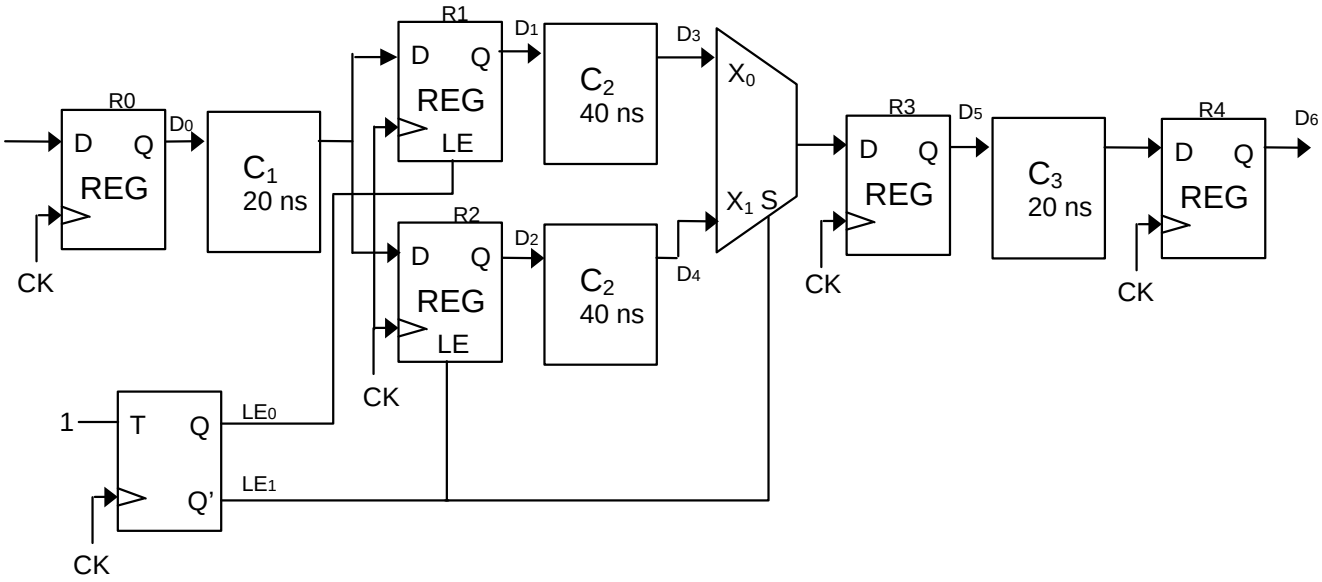
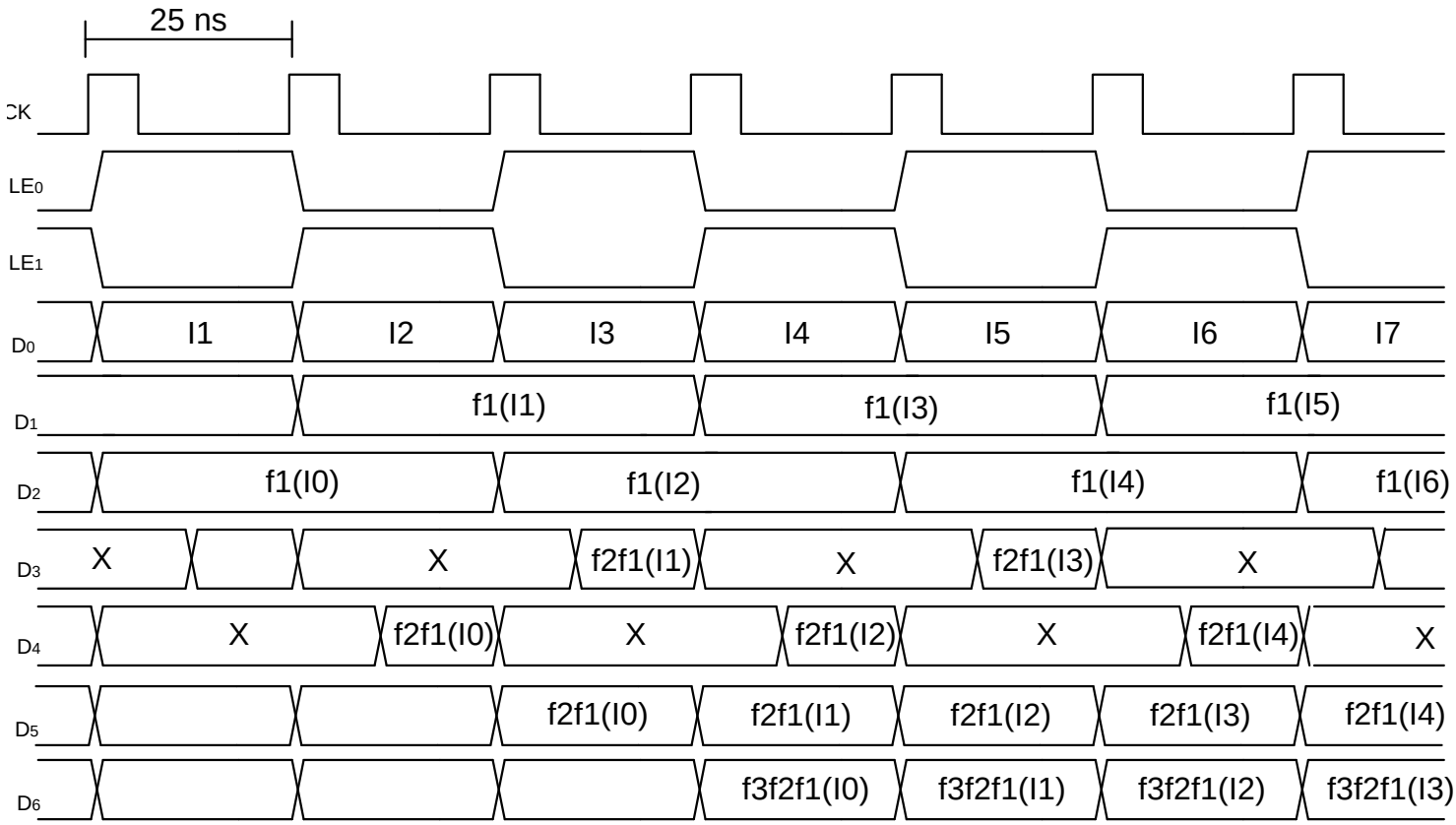


Diagramma di temporizzazione (timing)



Considerazioni sulla temporizzazione:

- La struttura pipeline complessiva è completamente sincrona.

- Quando uno dei due registri disposti in parallelo viene caricato, sullo stesso fronte di clock l'uscita della rete combinatoria relativa (in cascata) viene trasferita nel registro R3, al cui ingresso è stata deviata dal multiplexer.

- Con un periodo di 25 ns la sezione parallela x 2 è equivalente ad una catena lineare cadenzata con un periodo di 50 ns; i 10 ns in eccesso (50 - 40) rappresentano un tempo di guardia necessario per tenere conto di $t_{reg} + t_{mux} + t_{setup}$.

- I due segnali di uscita del flip-flop T cambiano stato ad ogni periodo di clock: pertanto devono essere distribuiti all'interno del circuito – dal flip-flop all'ingresso dei due registri R1 e R2 e del multiplexer – su due linee con tempi di propagazione compatibili con il tempo di ciclo.

Esercizio (2S19990617-D4)

Un PD32 ha il ciclo di accesso alla memoria di 200nsec. Calcolare di quanto viene rallentato durante l'operazione di trasferimento dati in DMA da un disco che abbia una velocità di lettura/scrittura di 12 Mbit/sec. Si supponga di trasferire dati di 32 bit alla volta.

Velocità del disco: 12 Mbit/s = 0.375 MLW/s
cioè il disco trasferisce una longword ogni $1/(0.375 \cdot 10^6)$ s = 2.666 microsec.
In 2666.666 ns il processore può effettuare in media 2666.666/200 = 13.333 cicli macchina; nello stesso tempo con il disco attivo ne potrà effettuare uno in meno.
Pertanto il rallentamento percentuale è dato da 1/13.333 = 7.5%.

E' possibile pervenire allo stesso risultato anche con il ragionamento seguente:

Il processore è in grado di prelevare dati dalla memoria alla velocità di:
32 (bit/LW) / 200(ns) bit/s = 32 · 5 Mb/s = 160 Mb/s.
D'altra parte si sa che l'interfaccia DMA del disco è in grado di prelevare dati dalla memoria al ritmo di 12 Mb/s.
Pertanto di ciascun blocco di 160 bit prelevati dalla memoria, 12 sono scambiati in DMA con il disco, e i rimanenti con il processore; pertanto il processore subisce un rallentamento percentuale pari a:
12/160 = 0.075 che corrisponde al 7.5%.

Esercizio (2S19990617-D5)

Descrivere la temporizzazione di un sistema SCA-SCO di tipo DMealy-Mealy.

Cfr. il testo “Reti Sequenziali” e gli “Appunti Integrativi”.

RETI LOGICHE
PRIMA PROVA SCRITTA DELL'APPELLO DEL 30-06-99

Studente:_____ **Docente:** _____

Un microprocessore PD32 deve comandare una macchina utensile costituita da un trapano a colonna e da una slitta di lavoro su cui viene posizionato un pezzo da forare. La slitta di lavoro può spostarsi lungo due assi X-Y in modo da presentare sulla perpendicolare della testa del trapano il punto da forare; la testa del trapano, a sua volta, è in grado di ruotare lungo un asse orizzontale per cambiare la punta foratrice (si supponga che siano possibili quattro punte diverse). Un programma memorizzato nel microprocessore dovrà dare i comandi alla macchina utensile per eseguire sul pezzo una serie di fori di diametro diverso e in posizioni prestabilite. Le informazioni che riceve la macchina sono:

- posizione del foro sul piano X-Y
- diametro del foro

Un sensore posto sulla macchina utensile indica con un segnale 0/1 quando la punta del trapano ha trapassato il pezzo da forare.

La precisione di posizionamento del foro deve essere migliore di 0,01 mm. su una escursione massima di 300 mm. I motori di azionamento sono passo-passo: essi funzionano con comando ad impulsi e un impulso corrisponde ad uno spostamento di 0,005 mm della piastra in un senso o in senso opposto a seconda del valore di un segnale di controllo che indica il senso di rotazione. La selezione della punta viene effettuata in funzione del diametro del foro. Il clock con cui funziona la macchina utensile è dato.

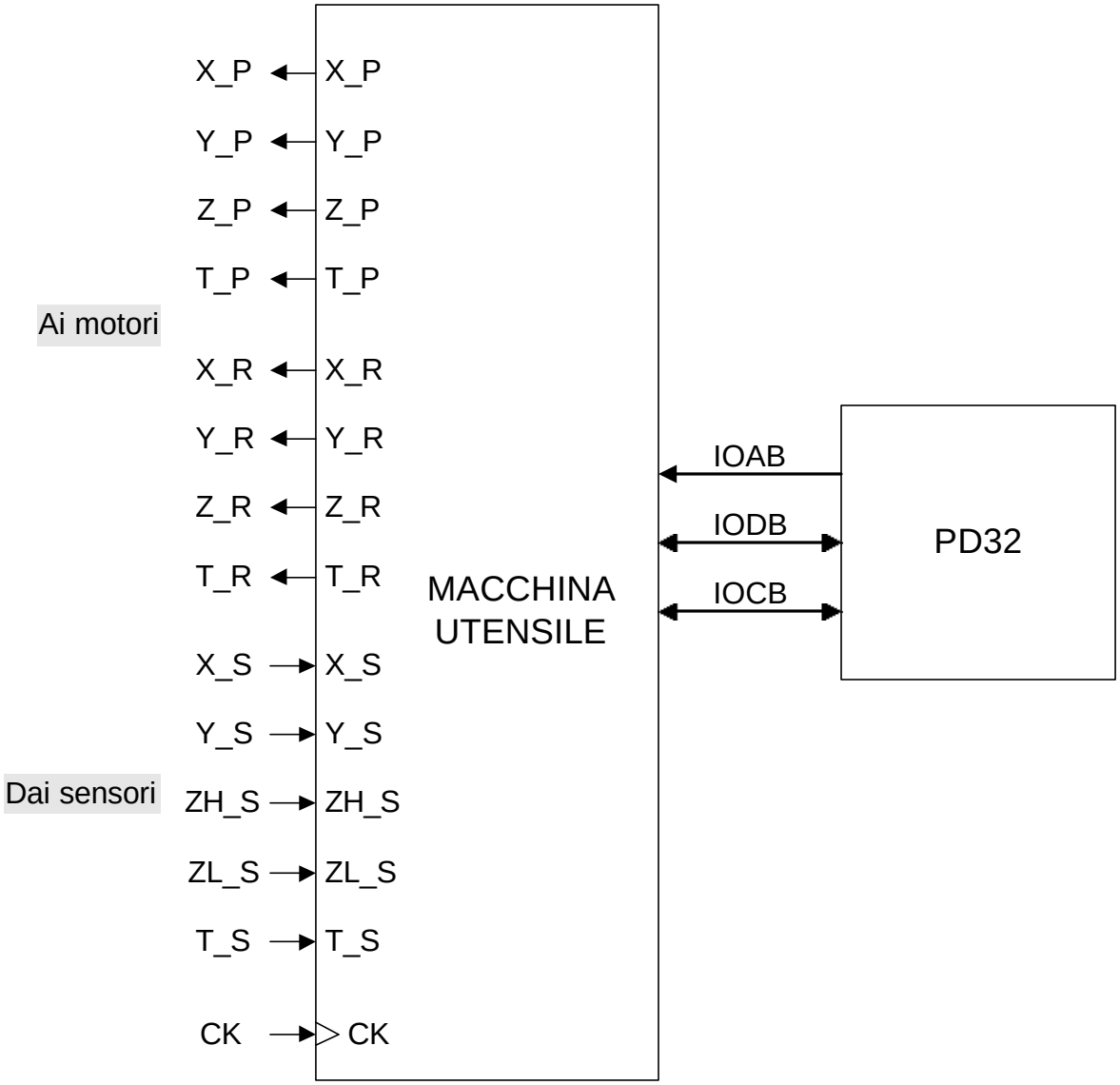
La discesa e la risalita della punta avviene mediante un motore passo passo dello stesso tipo di quelli che comandano gli spostamenti della piastra. Al termine di ogni risalita della punta, la macchina richiede al PD32 le informazioni relative al foro successivo.

Al termine del programma di foratura il microprocessore manda un comando per riportare la slitta in posizione di riposo.

Si richiede:

1) l'interfaccia tra microprocessore e macchina utensile
2) la logica di comando dei motori passo-passo
3) la struttura del programma di controllo.

TRAPANO: sistema esterno



Note

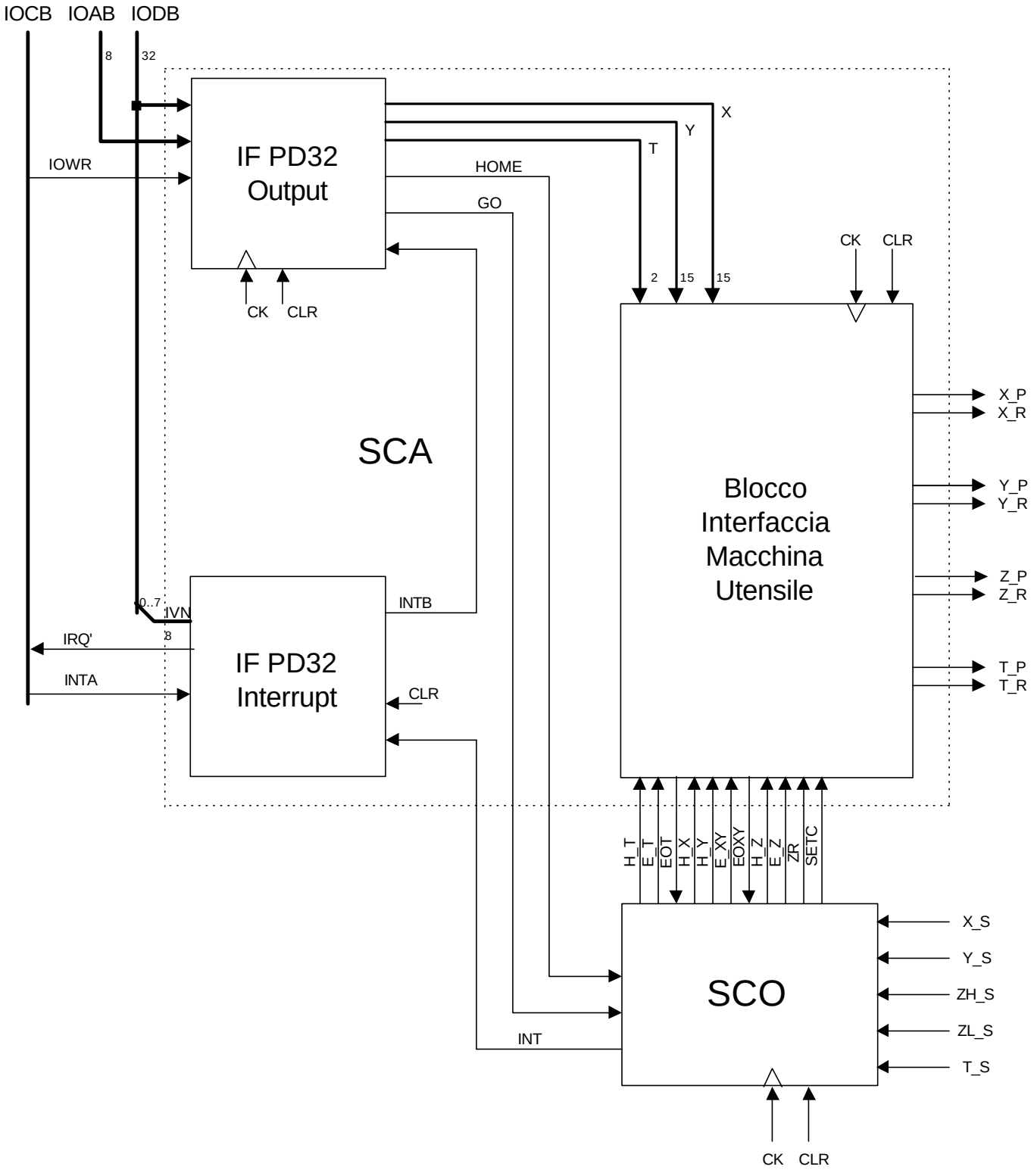
I segnali X_P, Y_P sono diretti ai tre motori passo-passo che controllano gli spostamenti della slitta lungo gli assi X, Y;
il segnale Z_P è diretto al motore passo-passo che controlla gli spostamenti verticali della punta del trapano;
il segnale T_P è diretto al motore passo-passo che controlla la selezione della punta del trapano.

I segnali con suffisso R indicano il senso di rotazione dei motori.

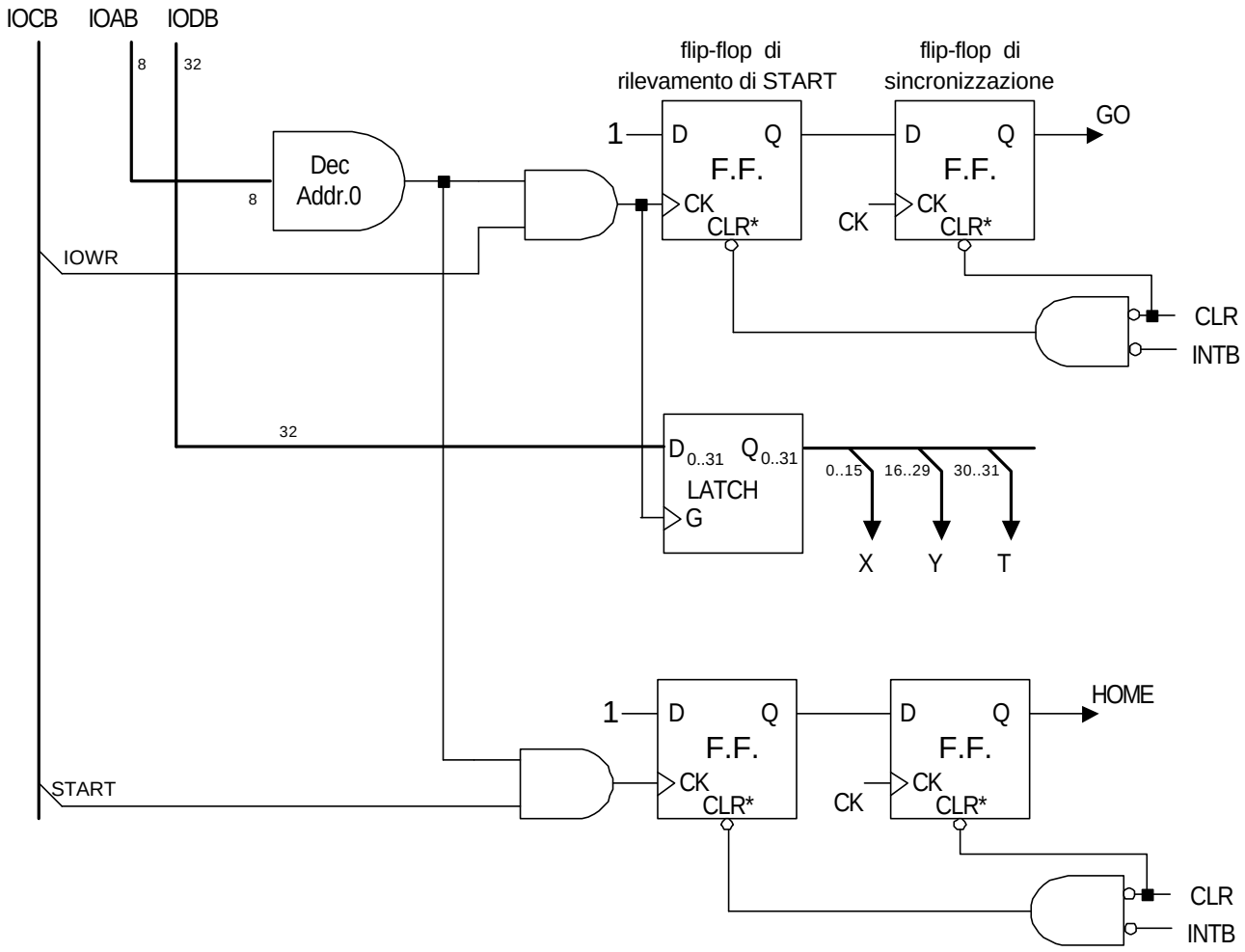
I segnali X_S, Y_S sono prodotti da due sensori di posizione posti sulle coordinate rispettive X=0, Y=0 della slitta;
il segnale ZH_S è prodotto da un sensore di posizione posto sulla quota Z=0 della punta del trapano;
il segnale ZL_S è prodotto da un sensore di posizione posto sulla quota di foratura completa;
il segnale T_S è prodotto da un sensore di posizione posto sulla punta di indice 0 del trapano.

I sensori sono necessari alla periferica per portare la macchina utensile nella posizione di riposo (X=Y=ZH=T=0) a partire da una configurazione meccanica iniziale casuale (ad esempio: collaudo nella linea di produzione, riattivazione dopo una mancanza di alimentazione elettrica).

TRAPANO: Schema a blocchi



TRAPANO: IF PD32 - output

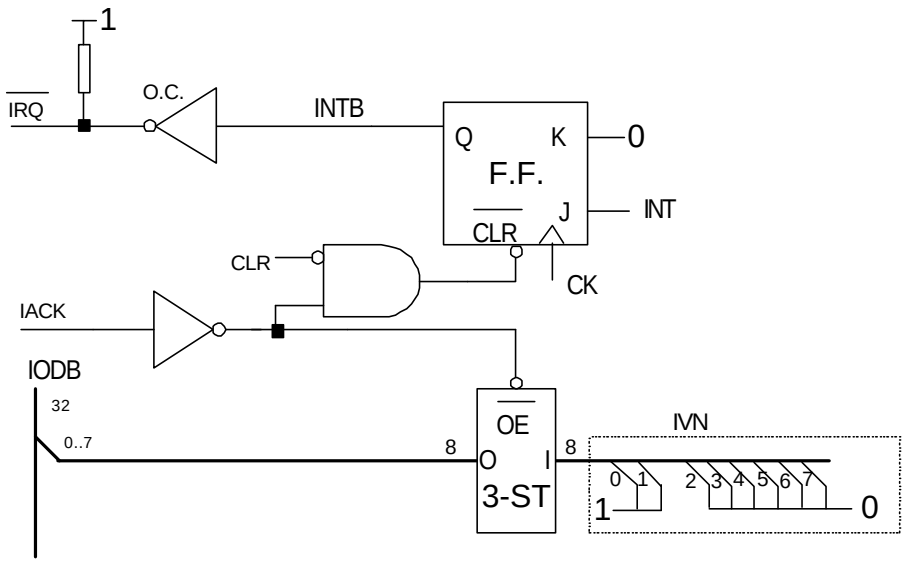


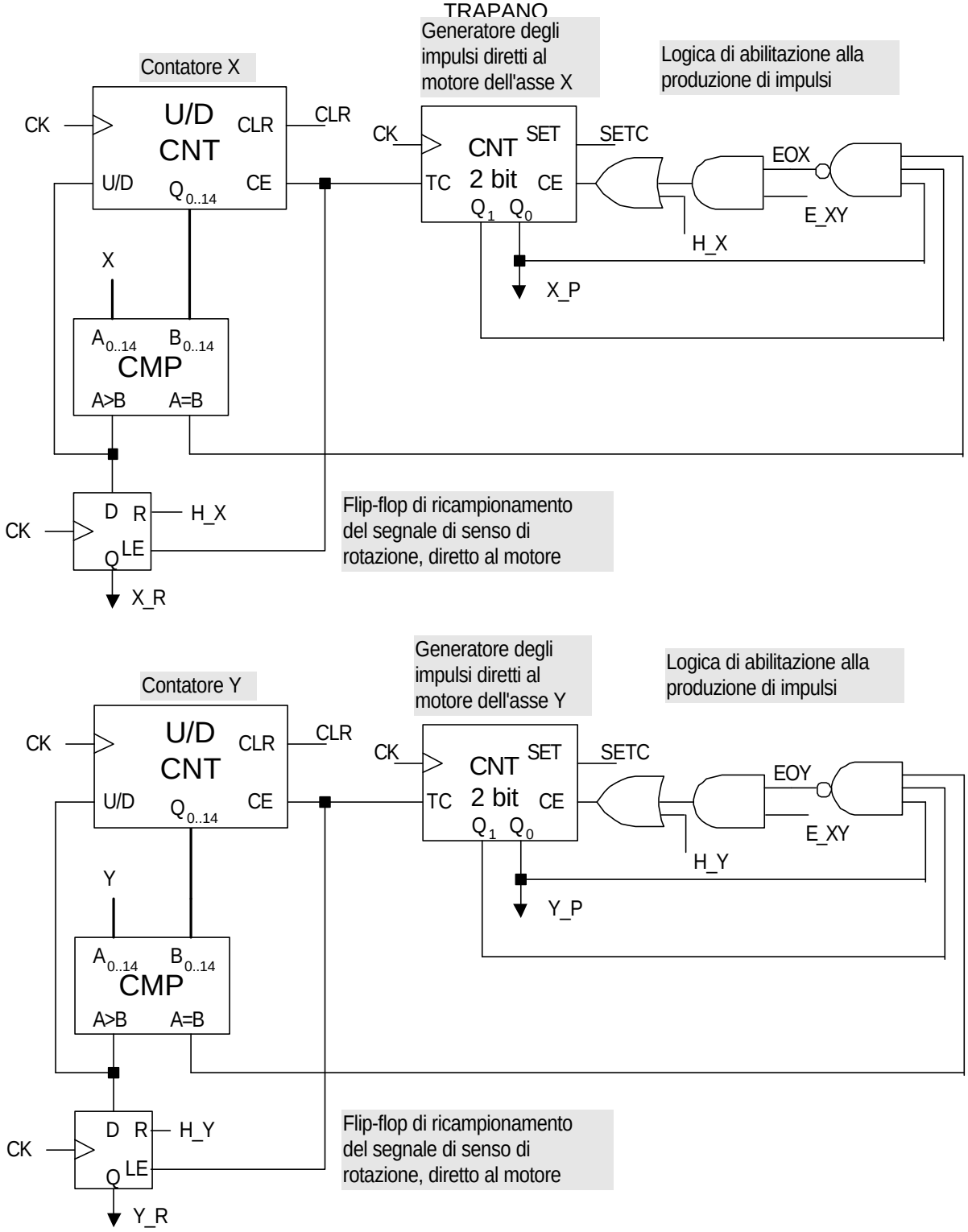
Note

Il SW avvia l'operazione con **OUTL XYT,Addr0**, dopo avere compattato i codici di X, Y e T in un'unica longword. Il programma di foratura deve terminare con l'istruzione **START Addr0** che comanda alla periferica di portare la macchina utensile nella posizione di riposo.

Ciascuno dei codici X e Y è a 15 bit, in quanto per il posizionamento del pezzo meccanico è specificata una precisione migliore di 0.01 mm su un'escursione massima della piastra di 300 mm: ciò equivale ad almeno 30000 posizioni discrete su ciascun asse, che possono essere rappresentate con un codice binario a 15 bit ($2^{14} < 30000 < 2^{15}$);
il codice T è a 2 bit in quanto deve selezionare una di quattro punte del trapano.

TRAPANO: IF PD32 - interrupt





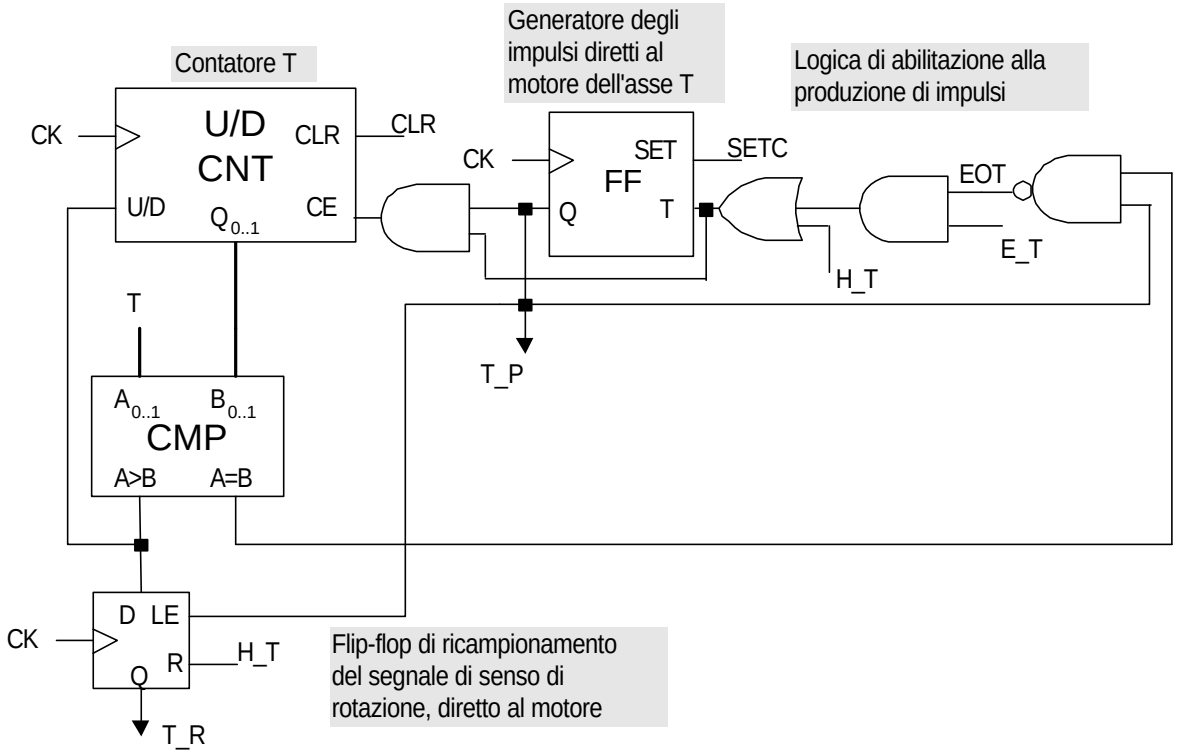
Note

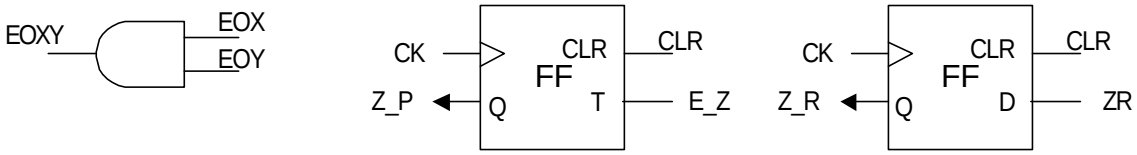
Il motore specificato provoca l'avanzamento della slitta di 0.005 mm per ogni impulso ricevuto: pertanto per far avanzare la slitta di 0.01 mm (risoluzione meccanica della slitta) è necessario applicare al motore due impulsi; essendo inoltre ciascun impulso costituito da due transizioni, è necessario applicare un prescaler modulo 4 a ciascuno dei contatori (cfr. anche i diagrammi di temporizzazione).

Il segnale E_XY abilita l'attuazione simultanea dei due motori che controllano lo spostamento della slitta lungo i due assi X e Y: le reti logiche relative all'asse X e all'asse Y sono uguali e producono impulsi per i rispettivi motori simultaneamente: nel caso generale di uno spostamento dal punto di coordinate (X0,Y0) al punto (X1,Y1) con 0<X1-X0<>Y1-Y0 il movimento composito della slitta si svilupperà lungo un asse inclinato a 45° rispetto agli assi X e Y fino a raggiungere una delle due coordinate finali e poi procederà lungo l'asse relativo all'altra coordinata (asse X o asse Y) fino a raggiungerla.

I segnali H_X e H_Y forzano i contatori a emettere impulsi verso i motori per riportare la slitta nella posizione di riposo (X=Y=0); nel contempo forzano il senso di rotazione del motore verso l'origine.

TRAPANO: interfaccia macchina utensile - 2





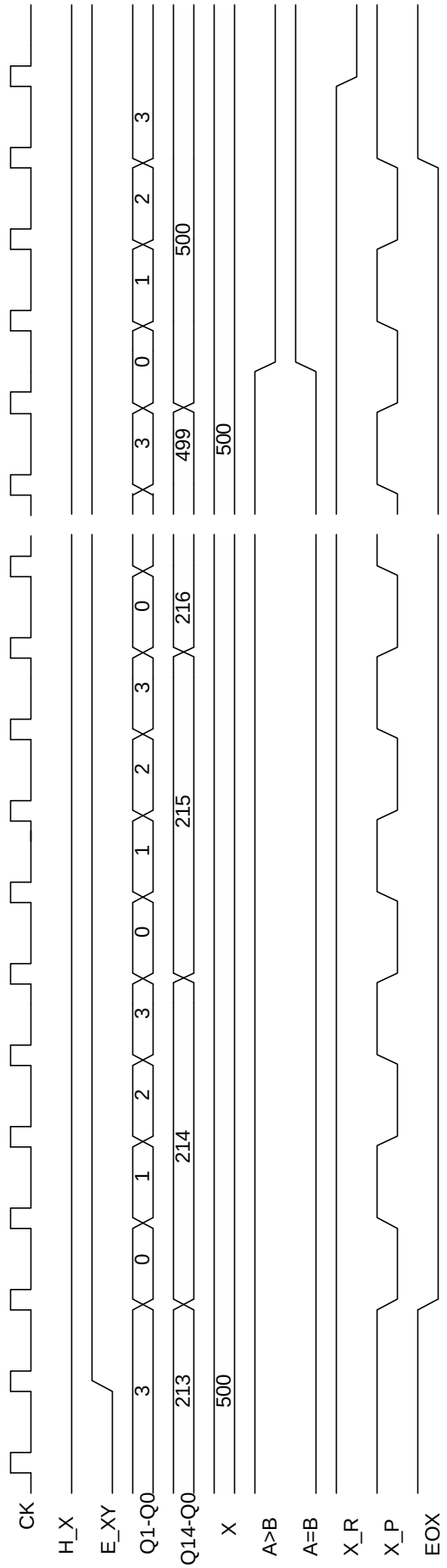
Note

Si suppone che il motore posizioni sulla colonna di foratura la punta successiva ad ogni impulso ricevuto: essendo ciascun impulso costituito da due transizioni, è necessario applicare un prescaler modulo 2, cioè un flip-flop T, al contatore mod 4 che codifica la punta selezionata.

Il segnale E_T abilita l'attuazione del motore che controlla lo spostamento delle punte del trapano.

Il segnale H_T forza il flip-flop T a emettere impulsi verso il motore per riportare il gruppo delle punte nella posizione di riposo (T=0); nel contempo fissa la direzione del motore verso l'origine.

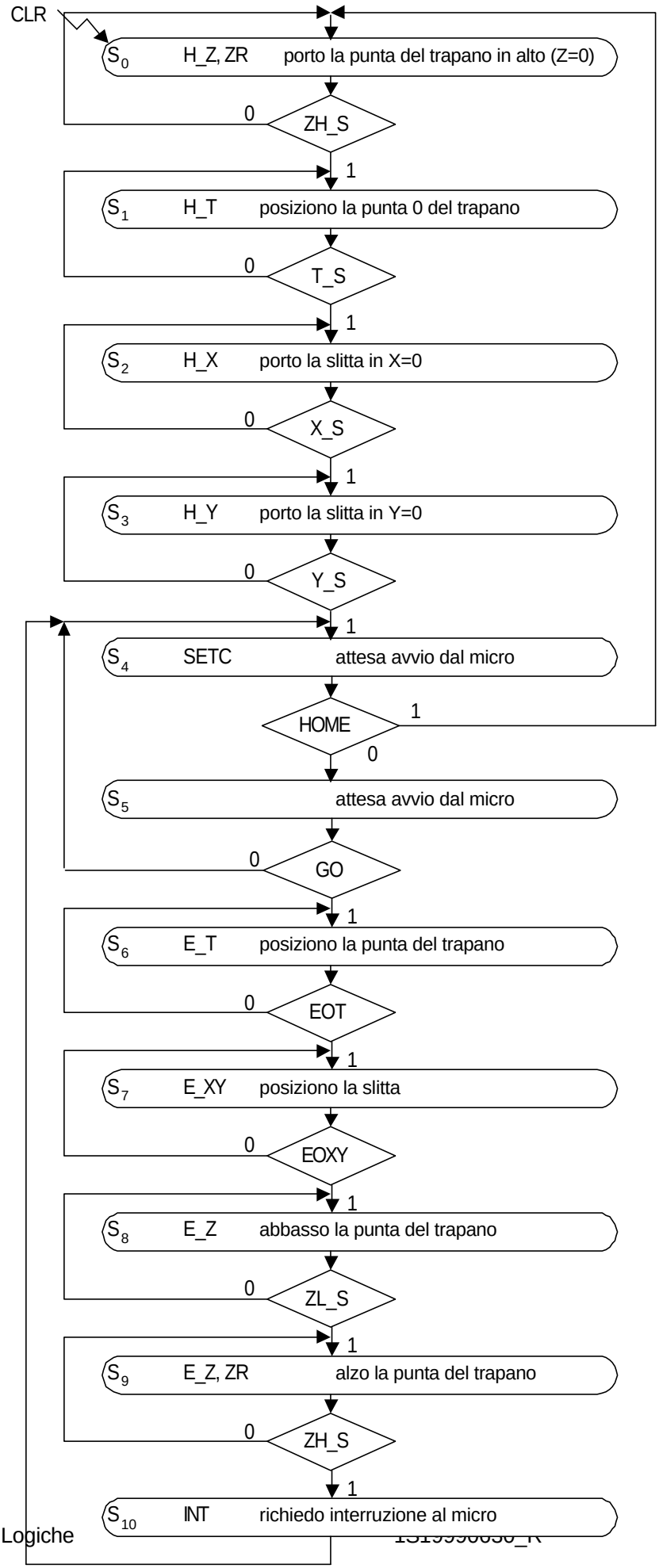
TRAPANO: temporizzazioni



Ipotesi di lavoro:
- è stato praticato un foro su un punto con X=213;
- il processore ha caricato la nuova coordinata X = 500.

Il diagramma evidenzia che dopo l'applicazione di una coppia di impulsi la slitta si sposta di un passo e assume quindi la posizione corrispondente allo stato del contatore (cfr. primo passo, da 213 a 214): pertanto è necessario concludere la sequenza degli impulsi quando il contatore registra la coordinata di destinazione (500 nell'esempio descritto) e il prescaler è sullo stato 3.

TRAPANO: SCO - flowchart



Stati S0..S3: portano la macchina utensile nella posizione di riposo. Vengono eseguiti all'attivazione della periferica e su comando del micro al termine del programma di foratura.

Stati S6..S9: controllano il posizionamento del pezzo e la sua foratura.

TRAPANO: SCO

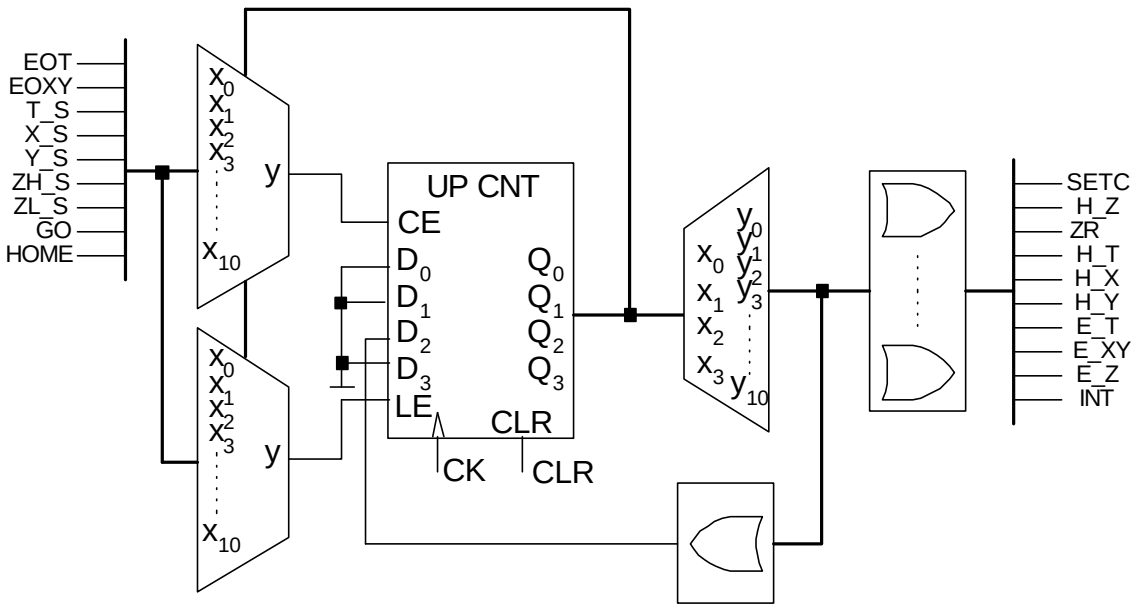
Il diagramma degli stati può essere implementato con un controllore di tipo Moore e con struttura MSI.
Il ritorno sullo stato S4 comporta per il registro di stato l'impiego di un contatore precaricabile (agli stati 0 e 4).
Le funzioni logiche costitutive delle reti combinatorie che calcolano CE (abilitazione al conteggio), LE (caricamento) e gli ingressi D0..3 del contatore e le uscite del controllore sono desumibili dal diagramma degli stati, tenendo conto della prevalenza dell'ingresso LE su CE del contatore :

$CE = S_0\,ZH\_S + S_1\,T\_S + S_2\,X\_S + S_3\,Y\_S + S_4 + S_5 + S_6\,EOT + S_7\,EOXY + S_8\,ZL\_S + S_9\,ZH\_S$
$LE = S_4\,HOME + S_5\,GO^* + S_{10}$
$D2 = S_5 + S_{10}$
$D0 = D1 = D3 = 0$

$H\_Z = S_0$
$ZR = S_0 + S_9$
$H\_T = S_1$
$H\_X = S_2$
$SETC = S_4$
$H\_Y = S_3$
$E\_T = S_6$
$E\_XY = S_7$
$E\_Z = S_8 + S_9$
$INT = S_{10}$

dove si è usato il simbolo * per indicare negazione.

Circuito dello SCO



TRAPANO: Routine Software

```
ORG 400h

nexthole DL 000800000h      ;puntatore alle coordinate di foratura
holescnt DW 0000h           ;contatore dei fori praticati
endofdrill DB 1             ;variabile binaria: programma di foratura terminato
holestab EQU 000800000h     ;ind. iniziale tavola coordinate (LW) di foratura
                            ;si presuppone che la tavola sia predisposta in memoria
holesamount EQU 16          ;numero di fori da praticare = lunghezza della tavola (in longword)
addr0 EQU 0A5h
…

CODE

mainloop:
...
        if (richiesta di foratura di un nuovo pezzo)
        movl #000800000h,nexthole   ;inizializzo variabili
        movl #0000h,holescnt
        movb #00h,endofdrill
        jsr new_drill               ;avvio programma di foratura
…
        jmp mainloop


; * * * R O U T I N E * * *

newdrill:
        push r0                     ;salvo i valori di R0 e R1 nello stack
        push r1
        movw holescnt,r0            ;verifico fine conteggio
        cmpw #holesamount,r0
        jz endofddr
        addw #0001h,r0              ;incremento conteggio
        movw r0,holescnt            ;e aggiorno contatore

        movl nexthole,r1            ;prelevo puntatore alle coordinate del foro da praticare
        movl (r1)+,r0               ;e lo incremento
        outl r0,addr0               ;scrivo le coordinate nella periferica
        movl r1,nexthole            ;salvo puntatore aggiornato
        jmp exit                    ;e esco

endofddr:
        start addr0                 ;termino programma di foratura
        movb #01h,endofdrill
exit:
        pop r1                      ;recupero i valori di R1 e R0 dallo stack
        pop r0
        ret


; * * * DRIVER * * *

driver 03h,drill
        jsr newdrill
        rti

end
```

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 30-06-99

Studente: _____ Docente: _____

D1 Disegnare lo schema di una rete a PLA per 10 variabili di ingresso, realizzata con moduli PLA ad 8 ingressi e con uscita tristate.

D2 Definire le funzioni G* e P* (generazione e propagazione di gruppo) per un addizionatore a propagazione rapida del riporto e disegnare lo schema per un adder a 16 bit utilizzando blocchi di 4 bit.

D3 Illustrare i passi per la minimizzazione di macchine parzialmente specificate, partendo dalla copertura finale.

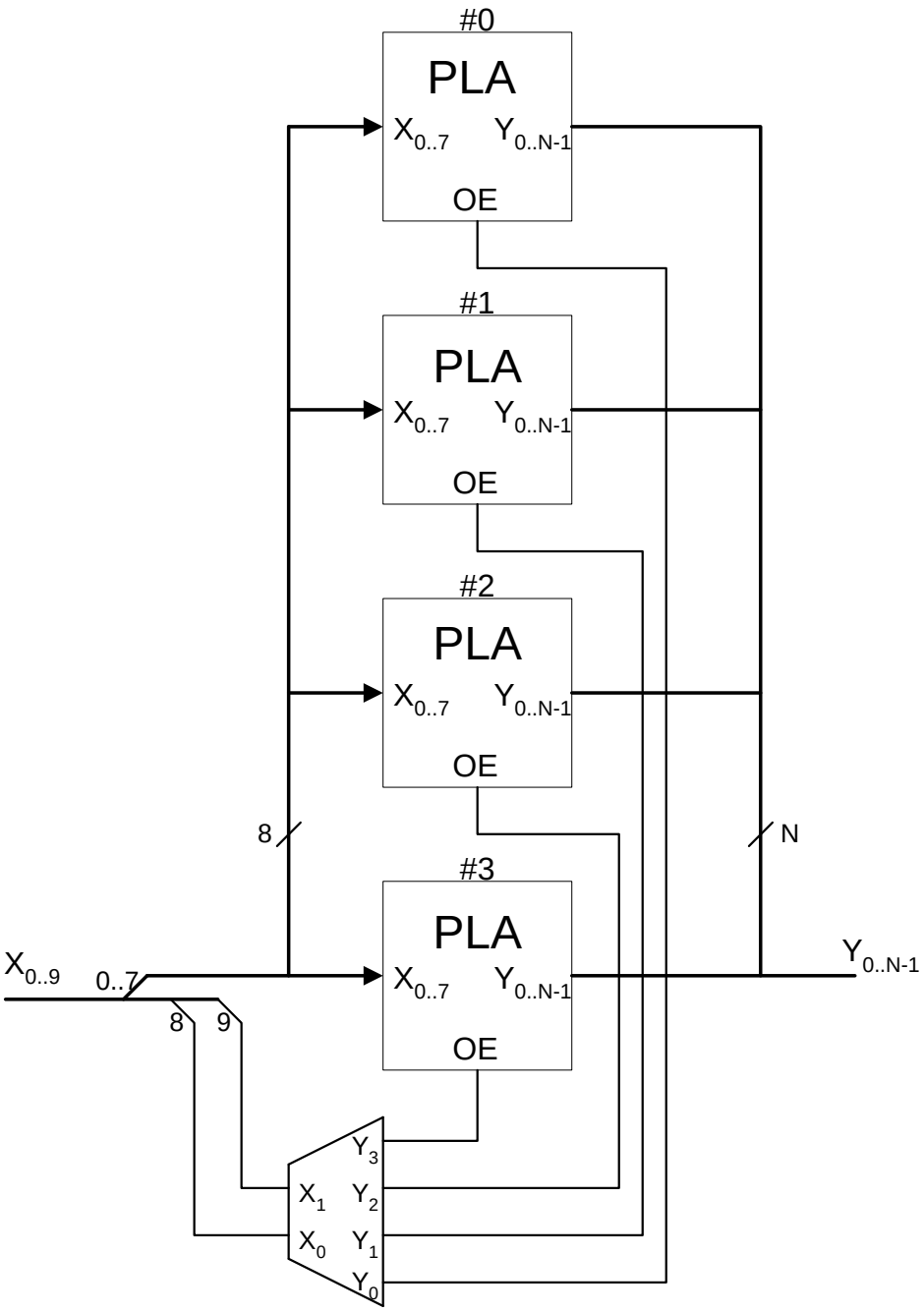
D4 Sintetizzare una macchina sequenziale che si comporti come un contatore decimale up/down con ingressi CE (count enable) U/D (up/down) ed uscita TC (terminal count).

D5 Scrivere una subroutine per trasferire il contenuto del registro R1 nel registro R2: nel trasferimento in R2 viene testato il bit 15 della long word e se questo è ad 1 viene scambiata la parola meno significativa con la più significativa.

Esercizio (2S19990630-D1)

Disegnare lo schema di una rete a PLA per 10 variabili di ingresso, realizzata con moduli PLA ad 8 ingressi e con uscita tristate.

La soluzione proposta prevede l'implementazione della funzione delle 10 variabili di ingresso $X_{0..9}$ tramite la separazione della tavola di verità in quattro sotto-tavole, ognuna relativa alle stesse 8 variabili di ingresso (ad es. $X_{0..7}$), ed individuate dalle due variabili residue (ad es. $X_{8..9}$). In questo modo le quattro sotto-tavole possono essere implementate mediante quattro moduli PLA del tipo disponibile, che condivideranno gli ingressi $X_{0..7}$; le uscite delle quattro PLA saranno selezionate mediante la decodifica delle due variabili $X_{0..7}$; in definitiva si ottiene la struttura hardware riportata di seguito:



Note

1. Si suppongono 10 bit di ingresso e N di uscita.

2. I moduli 0, 1, 2, 3 implementano ordinatamente i quattro quarti (definiti da X_9, X_8) della tavola di verità della funzione di $X_{0..9}$.

Esercizio (2S19990630-D2)

Definire le funzioni G* e P* (generazione e propagazione di gruppo) per un addizionatore a propagazione rapida del riporto e disegnare lo schema per un adder a 16 bit utilizzando blocchi di 4 bit.

Cfr. testo "Reti Combinatorie".

Esercizio (2S19990630-D3)

Illustrare i passi per la minimizzazione di macchine parzialmente specificate, partendo dalla copertura finale.

Cfr. testo "Reti Combinatorie".

Esercizio (2S19990630-D4)

Sintetizzare una macchina sequenziale che si comporti come un contatore decimale up/down con ingressi CE (count enable) U/D (up/down) ed uscita TC (terminal count).

Il contatore viene progettato come rete sincrona, il cui comportamento è descritto dalla tabella degli stati seguente; come richiesto dalla specifica, la macchina sincrona è dotata dei due segnali a livello CE (Count Enable) e U/D (1: Up; 0: Down); inoltre conviene codificare subito i 10 stati di conteggio:

funzione	CE	U/D	Y3	Y2	Y1	Y0	Y3’	Y2’	Y1'	Y0'	TC
conteggio bloccato	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	1	0	0	0	1	0
			…						…		
	0	0	1	0	0	1	1	0	0	1	0
	0	0	1	0	1	0	-	-	-	-	-
			…						…		
	0	0	1	1	1	1	-	-	-	-	-
	0	1	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	1	0	0	0	1	0
			…						…		
	0	1	1	0	0	1	1	0	0	1	0
	0	1	1	0	1	0	-	-	-	-	-
			…						…		
	0	1	1	1	1	1	-	-	-	-	-
conteggio decrem.	1	0	0	0	0	0	1	0	0	1	1
	1	0	0	0	0	1	0	0	0	0	0
			…						…		
	1	0	1	0	0	1	1	0	0	0	0
	1	0	1	0	1	0	-	-	-	-	-
			…						…		
	1	0	1	1	1	1	-	-	-	-	-
conteggio increm.	1	1	0	0	0	0	0	0	0	1	0
	1	1	0	0	0	1	0	0	1	0	0
			…						…		
	1	1	1	0	0	1	0	0	0	0	1
	1	1	1	0	1	0	-	-	-	-	-
			…						…		
	1	1	1	1	1	1	-	-	-	-	-

La sintesi completa richiede la minimizzazione di 4 MK (le 4 variabili di stato) a 6 variabili (le 4 variabili di stato + 2 di ingresso). Al passo successivo converrà impostare il calcolo delle funzioni di stato successivo per predisporre una implementazione del registro di stato del contatore con flip-flop di tipo T; cioè sarà opportuno calcolare le 4 funzioni T3, T2, T1, T0 corrispondenti alle variazioni tra le coppie Y3,Y3’, Y2,Y2’, Y1,Y1’, Y0,Y0’ rispettivamente; cioè: T3=Y3⊕Y3’, T2=Y2⊕Y2’, T1=Y1⊕Y1’, T0=Y0⊕Y0’.

Esercizio (2S19990630-D5)

Scrivere una subroutine per trasferire il contenuto del registro R1 nel registro R2: nel trasferimento in R2 viene testato il bit 15 della long word e se questo è ad 1 viene scambiata la parola meno significativa con la più significativa.

```
        org 400h                ;inizio programma

; ************************
; COSTANTI
; ************************
        STACK EQU 2800h         ;inizio area di stack
                                ;limitato a 2800h per consentire la simulazione
; ************************
; CODICE
; ************************
        code                    ;inizio istruzioni
main:
        movl #STACK,r7          ;inizializza R7 quale SP
        seti                    ;abilita interruzioni (SP è stato inizializzato)

        movl #0A5A5B4B4h,r1     ;inizializzo r1 per il test al simulatore
        jsr exchr2              ;con r1.15=1

        movl #0A5A57878h,r1     ;inizializzo r1 per il test al simulatore
        jsr exchr2              ;con r1.15=0

        halt                    ;serve solo per bloccare il simulatore

; ************************
; ROUTINE
; ************************
exchr2:
        movl r1,r2              ;trasferisco r1 in r2
        rorl #16,r2             ;scambio le due word di r2
        jc exit                 ;test sul bit 15
        rorl #16,r2             ;ri-scambio le due word di r2
exit:
        ret

        end                     ;fine della compilazione
```

```
;trasferisce il contenuto del registro R1 nel registro R2:
;nel trasferimento in R2 viene testato il bit 15 della long word
;e se questo è ad 1 viene scambiata la parola meno significativa
;con la più significativa.

  org 400h    ;inizio programma

; *************************
; COSTANTI
; *************************

  STACK   EQU 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione

; *************************
; CODICE
; *************************

  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)

  movl #0A5A5B4B4h,r1 ;inizializzo r1 per il test al simulatore
  jsr exchr2    ;con r1.15=1

  movl #0A5A57878h,r1 ;inizializzo r1 per il test al simulatore
  jsr exchr2    ;con r1.15=0

  halt      ;serve solo per bloccare il simulatore


; *************************
; ROUTINE
; *************************

exchr2:
  movl r1,r2    ;trasferisco r1 in r2
  rorl #16,r2   ;scambio le due word di r2
  jc exit     ;test sul bit 15
  rorl #16,r2   ;ri-scambio le due word di r2
exit:
  ret

  end     ;fine della compilazione
```

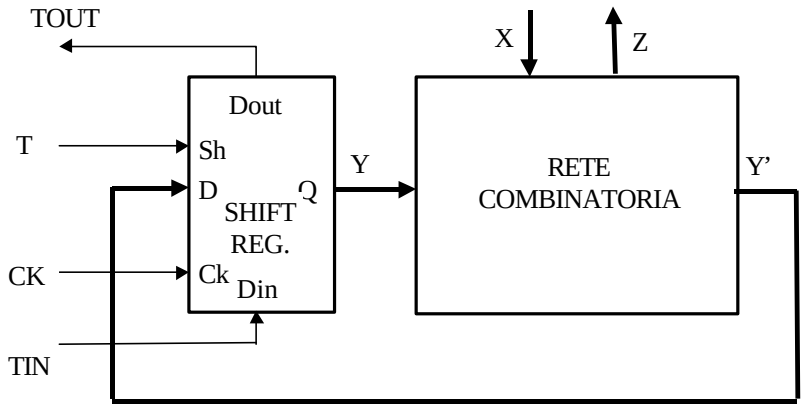
RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 15-07-99

STUDENTE:_____ DOCENTE:_____

Specifiche funzionali:

In una linea di produzione di un circuito integrato LSI si vuole allestire una stazione di collaudo (tester) per individuare i chip guasti e separarli da quelli funzionanti. Il circuito LSI può essere schematizzato (vedi figura) come una struttura di Mealy sincrona, predisposta al collaudo mediante l'impiego di un registro di stato a scalamento, di lunghezza 128 bit.



I segnali del circuito disponibili per l'interfacciamento con il tester da progettare sono i seguenti:
CK: è terminato sull'ingresso di clock del circuito integrato;
TIN, TOUT: ingresso e uscita seriali del registro di stato; vengono utilizzati solo in fase di test.
T: se vale 1 provoca lo scorrimento del registro di stato; altrimenti il registro viene caricato in parallelo (nel funzionamento normale T sarà ovviamente fissato a 0).
Il collaudo consiste nella ripetizione di 64K cicli di test, il k-esimo dei quali è composto di tre passi:
1 - il registro viene caricato serialmente attraverso TIN con una configurazione (vettore di test) Y_k;
2 - il registro viene caricato in parallelo con il vettore Y_k' calcolato dalla rete combinatoria;
3 - il registro viene scaricato serialmente attraverso TOUT ed il vettore Y_k' viene confrontato con un vettore E_k precalcolato, che rappresenta la configurazione attesa in condizioni di funzionamento corretto: in caso di completa coincidenza tra le due sequenze si passa al ciclo di test successivo, altrimenti al verificarsi della prima differenza tra le due sequenze il collaudo verrà terminato, in quanto il chip sarà ritenuto guasto ed avviato allo scarto. Soltanto in caso di completa coincidenza tra tutte le 64K coppie di vettori (Y_k', E_k) il collaudo sarà terminato con successo. Il tester comunica l'esito del collaudo del chip al processore mediante interruzione.
Tutti i vettori Y_k,Y_k', E_k sono a 128 bit. I vettori Y_k, E_k (precalcolati in fase di progetto del circuito LSI) vanno predisposti in un banco di chip ROM dotati di organizzazione interna 256K x 8 bit e tempo di accesso pari a 200 ns. Il circuito deve essere collaudato ad una frequenza di clock pari a 50 MHz.

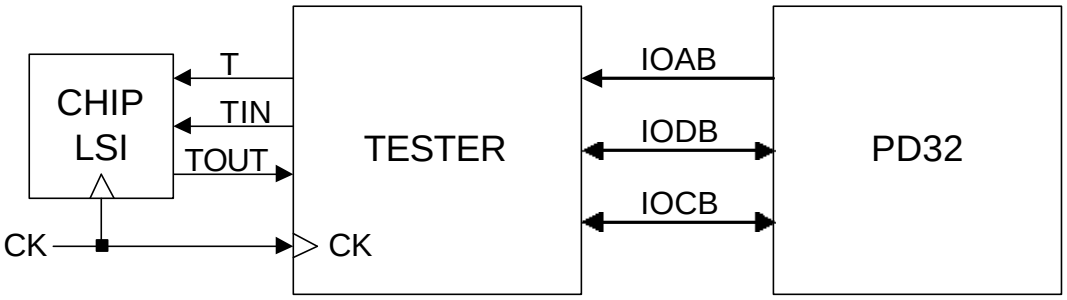
Si richiede:

- lo schema a blocchi funzionali del tester;
- il dimensionamento del banco di ROM e la configurazione interna dei dati di test;
- il diagramma di temporizzazione dell'interfaccia con il circuito integrato dato;
- lo schema logico dettagliato dello SCA e dello SCO del tester;
- l'interfaccia con il PD32.

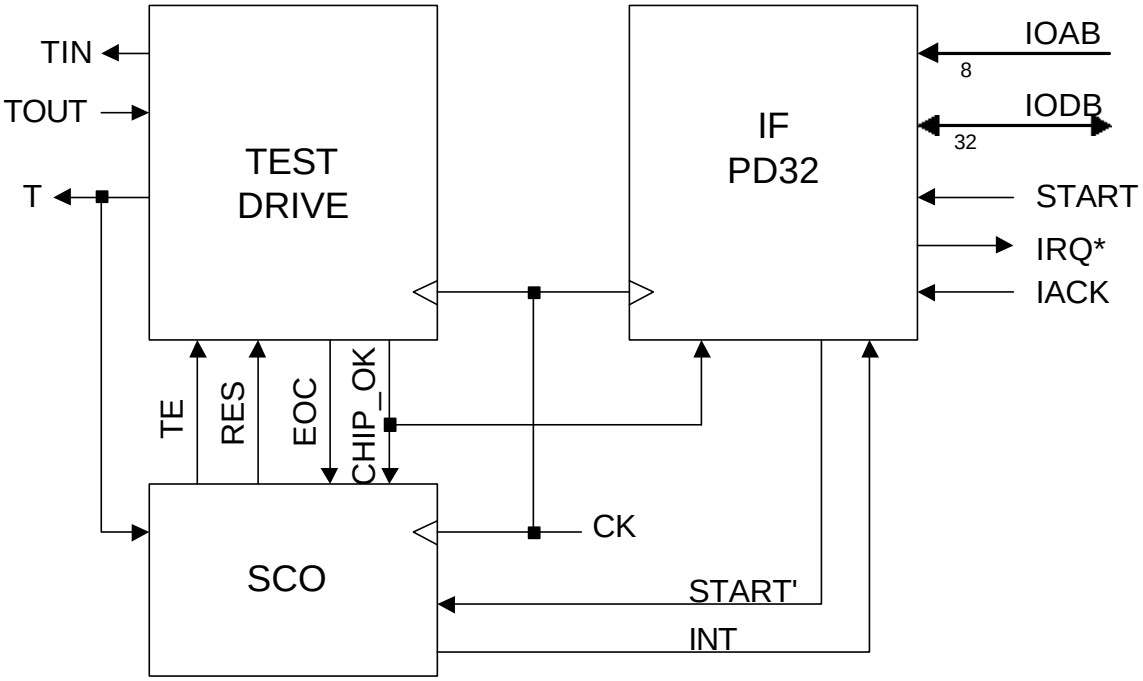
Note/Suggerimenti:

Si noti che il passo 3 del ciclo di test k-esimo può essere sovrapposto temporalmente con il passo 1 del ciclo di test (k+1)-esimo.

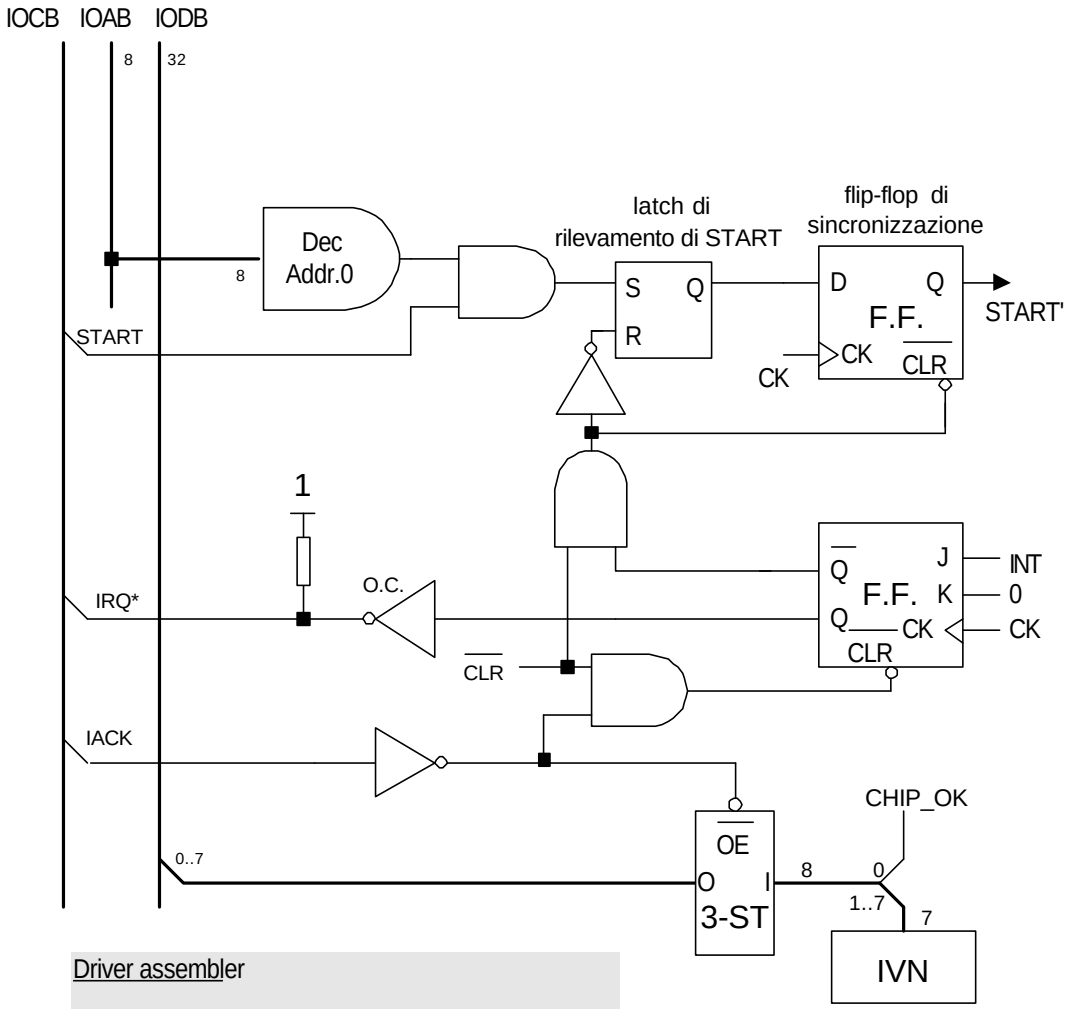
TESTER: sistema esterno



TESTER: schema a blocchi



TESTER: IF PD32



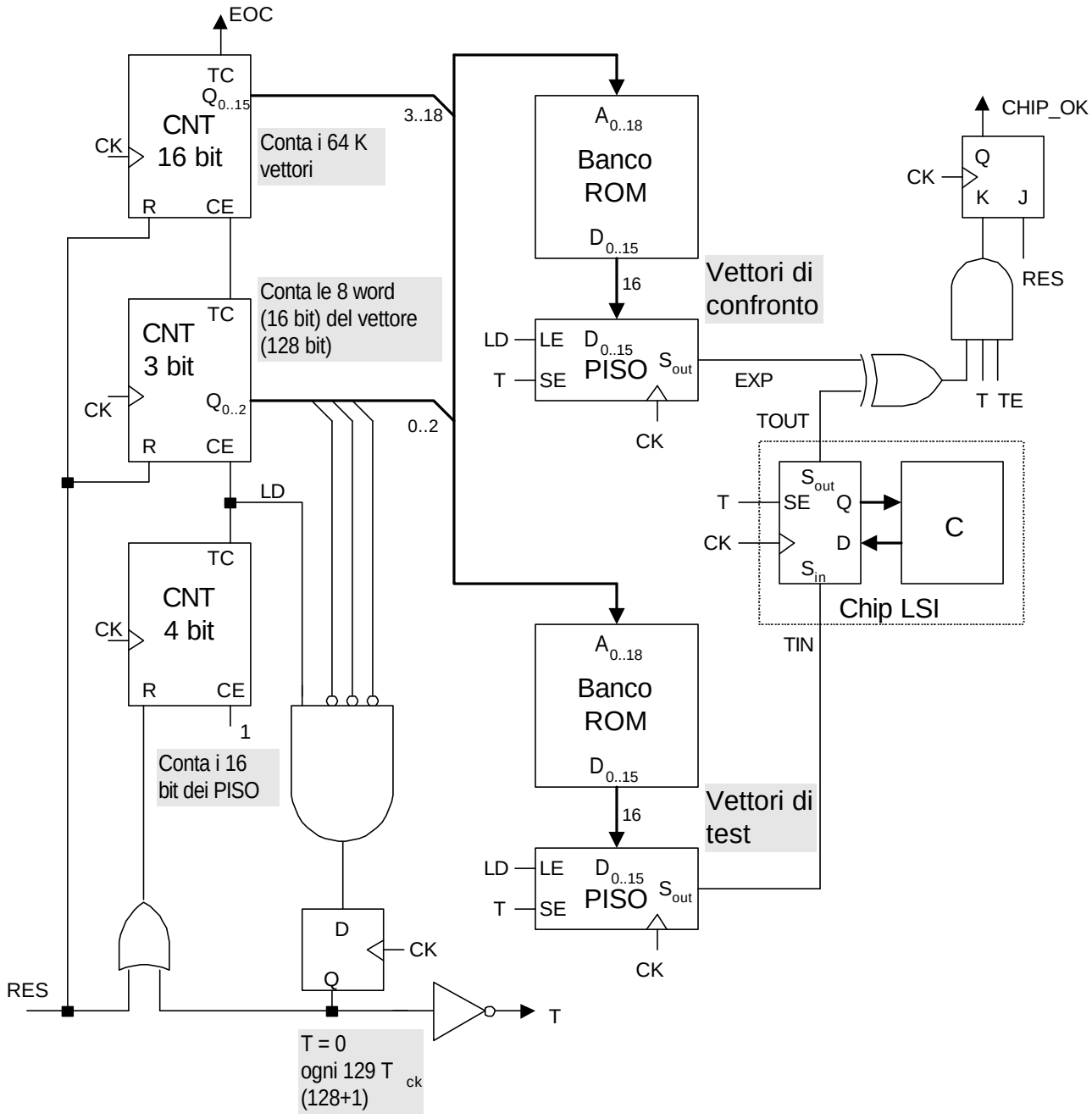
Driver assembler

```
main:
. . .

DRIVER 10h,2000h
START Attuatore_scarto
last: ;if(ultimo chip) exit
START Addr0        ;inizia test del chip successivo
exit: RTI

DRIVER 11h,2100h
START Attuatore_chip_OK
jmp last
```

Note

- Periferica di uscita;
- solo START, nessun dato.
- Il bit CHIP_OK è stato inserito nella posizione 0 dell'IVN: in questo modo il driver non ha bisogno di leggere il risultato; ci saranno due driver, associati a chip buono (11h) e chip guasto (10h) rispettivamente.

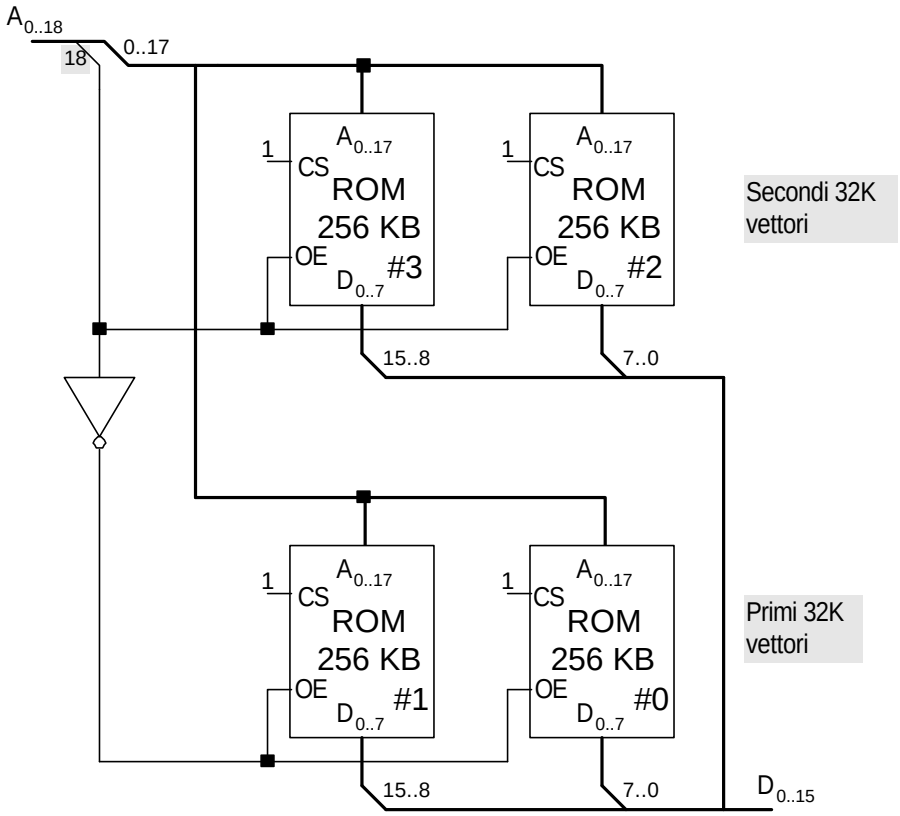
TESTER: TEST DRIVE



Dimensionamento dei registri PISO

T_{ck} = 20 ns; T_{ROM} = 200 ns; pertanto deve essere: L_{PISO} x 20 ns > 200 ns; inoltre L_{PISO} = 8 x n; cioè deve essere: 20 x 8 x n > 200; il minimo valore intero di n che soddisfa il vincolo è n = 2, che implica L_{PISO} = 16. Il valore di n va minimizzato perché è opportuno minimizzare la lunghezza dei registri PISO; inoltre, n = 2 significa che vanno montati 2 chip di ROM in parallelo (cfr. progetto del modulo Banco ROM).

TESTER: banco ROM



Organizzazione dei dati:
64K x 128 bit = 512K x 16 bit = 4 x 256 KB

Lo stesso gruppo di contatori indirizza entrambi i banchi di ROM, contenenti i vettori di test (64K x D0..127 nel diagramma di temporizzazione) e i vettori delle risposte attese (64K x E0..127 nel diagramma di temporizzazione) rispettivamente. Tuttavia va notato che il primo vettore delle risposte attese deve essere confrontato con il primo vettore scaricato dal registro di stato del chip LSI esterno sotto test (il primo dei 64K vettori R0..127 nel diagramma di temporizzazione); ciò comporta la necessità di introdurre un ritardo di 128 bit (lunghezza di un vettore di test) tra la lettura del vettore di test dalla ROM e la lettura del vettore omologo delle risposte attese (cfr. anche il diagramma di temporizzazione). Tale ritardo può essere ottenuto semplicemente se il vettore delle risposte viene memorizzato nella ROM ad un indirizzo incrementato di 8 (=128/16) rispetto all'indirizzo del vettore di test omologo. In pratica il primo vettore di test verrà memorizzato agli indirizzi 0..7 (8x16=128) mentre il primo vettore delle risposte occuperà le posizioni di indirizzo 8..15. Le coppie successive dei vettori omologhi verranno memorizzate a indirizzi mantenuti a distanza 8. L'ultimo vettore delle risposte occuperà invece le posizioni agli indirizzi 0..7, e quindi dovrà essere previsto un accesso circolare alla ROM delle risposte; questo è congruente con il fatto che il test dovrà prevedere 64K scansioni dei vettori di test + 1 scansione per recuperare il risultato (128 bit) dell'applicazione dell'ultimo vettore di test. Naturalmente durante lil caricamento del primo vettore di test nel registro di stato del circuito sotto test dovrà essere ignorato l'esito del confronto, in quanto non ancora significativo; a questo scopo è stato introdotto il segnale di mascheramento TE (Test Enable), mantenuto a 0 durante tale transitorio e lasciato a 1 a regime.

TESTER: Organizzazione dei dati nelle ROM

Vettori di test

Indirizzi	ROM #1					ROM #0					
0	$T_{0,15}$	$T_{0,14}$	…	$T_{0,9}$	$T_{0,8}$	$T_{0,7}$	$T_{0,6}$	…	$T_{0,1}$	$T_{0,0}$	Vettore di test #0
1	$T_{0,31}$	$T_{0,30}$	…	$T_{0,25}$	$T_{0,24}$	$T_{0,23}$	$T_{0,22}$	…	$T_{0,17}$	$T_{0,16}$	
…	…	…	…	…	…	…	…	…	…	…	
7	$T_{0,127}$	$T_{0,126}$	…	$T_{0,121}$	$T_{0,120}$	$T_{0,119}$	$T_{0,118}$	…	$T_{0,113}$	$T_{0,112}$	
8	$T_{1,15}$	$T_{1,14}$	…	$T_{1,9}$	$T_{1,8}$	$T_{1,7}$	$T_{1,6}$	…	$T_{1,1}$	$T_{1,0}$	Vettore di test #1
9	$T_{1,31}$	$T_{1,30}$	…	$T_{1,25}$	$T_{1,24}$	$T_{1,23}$	$T_{1,22}$	…	$T_{1,17}$	$T_{1,16}$	
…	…	…	…	…	…	…	…	…	…	…	
15	$T_{1,127}$	$T_{1,126}$	…	$T_{1,121}$	$T_{1,120}$	$T_{1,119}$	$T_{1,118}$	…	$T_{1,113}$	$T_{1,112}$	
…	…	…	…	…	…	…	…	…	…	…	…
256k-8	$T_{M-1,15}$	$T_{M-1,14}$	…	$T_{M-1,9}$	$T_{M-1,8}$	$T_{M-1,7}$	$T_{M-1,6}$	…	$T_{M-1,1}$	$T_{M-1,0}$	Vettore di test #32k-1
256k-7	$T_{M-1,31}$	$T_{M-1,30}$	…	$T_{M-1,25}$	$T_{M-1,24}$	$T_{M-1,23}$	$T_{M-1,22}$	…	$T_{M-1,17}$	$T_{M-1,16}$	
…	…	…	…	…	…	…	…	…	…	…	
256k-1	$T_{M-1,127}$	$T_{M-1,126}$	…	$T_{M-1,121}$	$T_{M-1,120}$	$T_{M-1,119}$	$T_{M-1,118}$	…	$T_{M-1,113}$	$T_{M-1,112}$	

M=32k

Indirizzi	ROM #3					ROM #2					
0	$T_{M,15}$	$T_{M,14}$	…	$T_{M,9}$	$T_{M,8}$	$T_{M,7}$	$T_{M,6}$	…	$T_{M,1}$	$T_{M,0}$	Vettore di test #32k
1	$T_{M,31}$	$T_{M,30}$	…	$T_{M,25}$	$T_{M,24}$	$T_{M,23}$	$T_{M,22}$	…	$T_{M,17}$	$T_{M,16}$	
…	…	…	…	…	…	…	…	…	…	…	
7	$T_{M,127}$	$T_{M,126}$	…	$T_{M,121}$	$T_{M,120}$	$T_{M,119}$	$T_{M,118}$	…	$T_{M,113}$	$T_{M,112}$	
…	…	…	…	…	…	…	…	…	…	…	
256k-8	$T_{L-1,15}$	$T_{L-1,14}$	…	$T_{L-1,9}$	$T_{L-1,8}$	$T_{L-1,7}$	$T_{L-1,6}$	…	$T_{L-1,1}$	$T_{L-1,0}$	Vettore di test #64k-1
256k-7	$T_{L-1,31}$	$T_{L-1,30}$	…	$T_{L-1,25}$	$T_{L-1,24}$	$T_{L-1,23}$	$T_{L-1,22}$	…	$T_{L-1,17}$	$T_{L-1,16}$	
…	…	…	…	…	…	…	…	…	…	…	
256k-1	$T_{L-1,127}$	$T_{L-1,126}$	…	$T_{L-1,121}$	$T_{L-1,120}$	$T_{L-1,119}$	$T_{L-1,118}$	…	$T_{L-1,113}$	$T_{L-1,112}$	

M=32k ; L=64k

TESTER: Organizzazione dei dati nelle ROM

Vettori delle risposte attese (per il confronto)

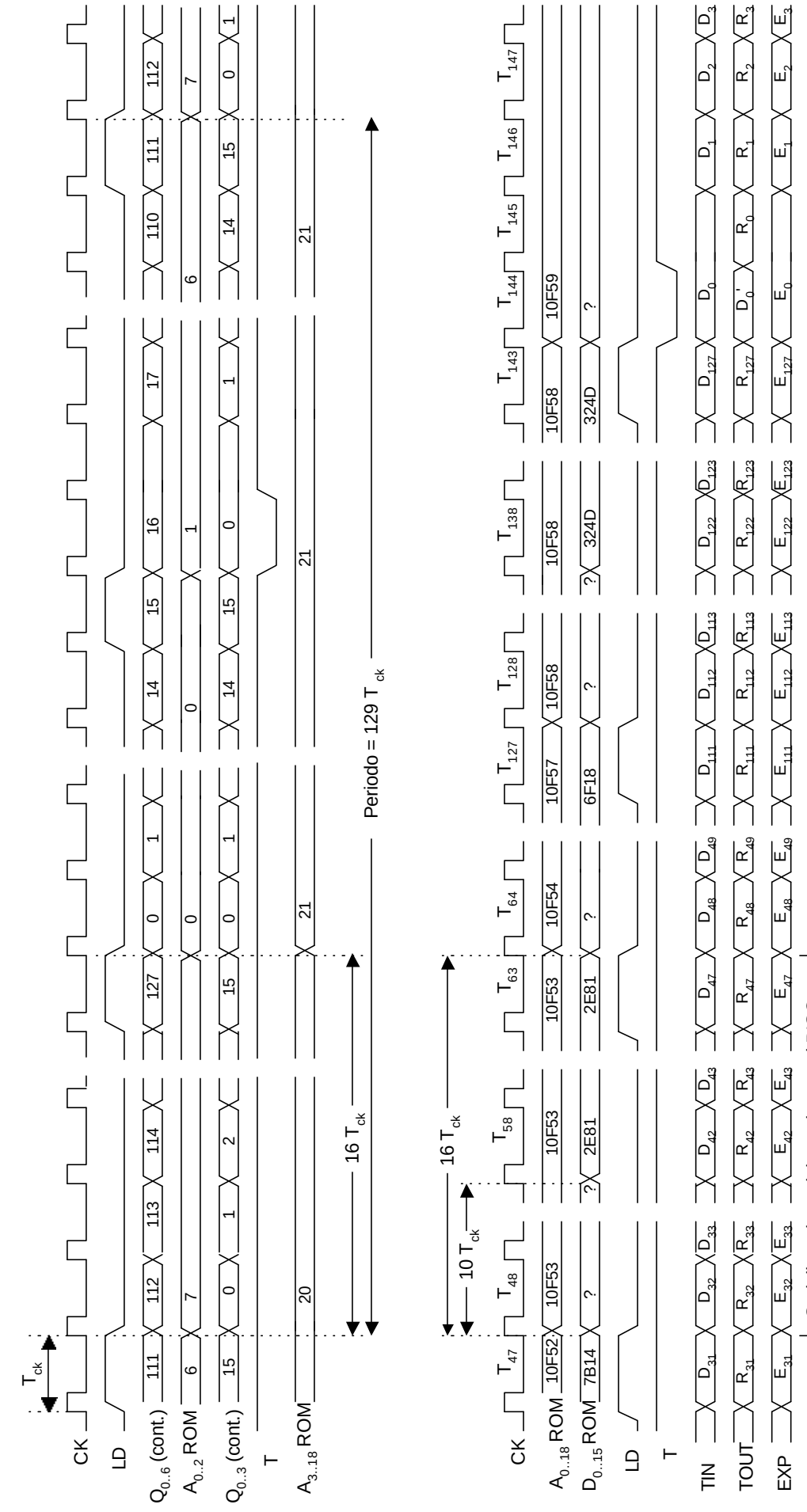
Indirizzi	ROM #1					ROM #0					
0	$E_{L-1,15}$	$E_{L-1,14}$	…	$E_{L-1,9}$	$E_{L-1,8}$	$E_{L-1,7}$	$E_{L-1,6}$	…	$E_{L-1,1}$	$E_{L-1,0}$	Vettore risposte #64K-1
1	$E_{L-1,31}$	$E_{L-1,30}$	…	$E_{L-1,25}$	$E_{L-1,24}$	$E_{L-1,23}$	$E_{L-1,22}$	…	$E_{L-1,17}$	$E_{L-1,16}$	
…	…	…	…	…	…	…	…	…	…	…	
7	$E_{L-1,127}$	$E_{L-1,126}$	…	$E_{L-1,121}$	$E_{L-1,120}$	$E_{L-1,119}$	$E_{L-1,118}$	…	$E_{L-1,113}$	$E_{L-1,112}$	
8	$E_{0,15}$	$E_{0,14}$	…	$E_{0,9}$	$E_{0,8}$	$E_{0,7}$	$E_{0,6}$	…	$E_{0,1}$	$E_{0,0}$	Vettore risposte #0
9	$E_{0,31}$	$E_{0,30}$	…	$E_{0,25}$	$E_{0,24}$	$E_{0,23}$	$E_{0,22}$	…	$E_{0,17}$	$E_{0,16}$	
…	…	…	…	…	…	…	…	…	…	…	
15	$E_{0,127}$	$E_{0,126}$	…	$E_{0,121}$	$E_{0,120}$	$E_{0,119}$	$E_{0,118}$	…	$E_{0,113}$	$E_{0,112}$	
…	…	…	…	…	…	…	…	…	…	…	…
256k-8	$E_{M-2,15}$	$E_{M-2,14}$	…	$E_{M-2,9}$	$E_{M-2,8}$	$E_{M-2,7}$	$E_{M-2,6}$	…	$E_{M-2,1}$	$E_{M-2,0}$	Vettore risposte #32k-2
256k-7	$E_{M-2,31}$	$E_{M-2,30}$	…	$E_{M-2,25}$	$E_{M-2,24}$	$E_{M-2,23}$	$E_{M-2,22}$	…	$E_{M-2,17}$	$E_{M-2,16}$	
…	…	…	…	…	…	…	…	…	…	…	
256k-1	$E_{M-2,127}$	$E_{M-2,126}$	…	$E_{M-2,121}$	$E_{M-2,120}$	$E_{M-2,119}$	$E_{M-2,118}$	…	$E_{M-2,113}$	$E_{M-2,112}$	

M=32k; L=64k

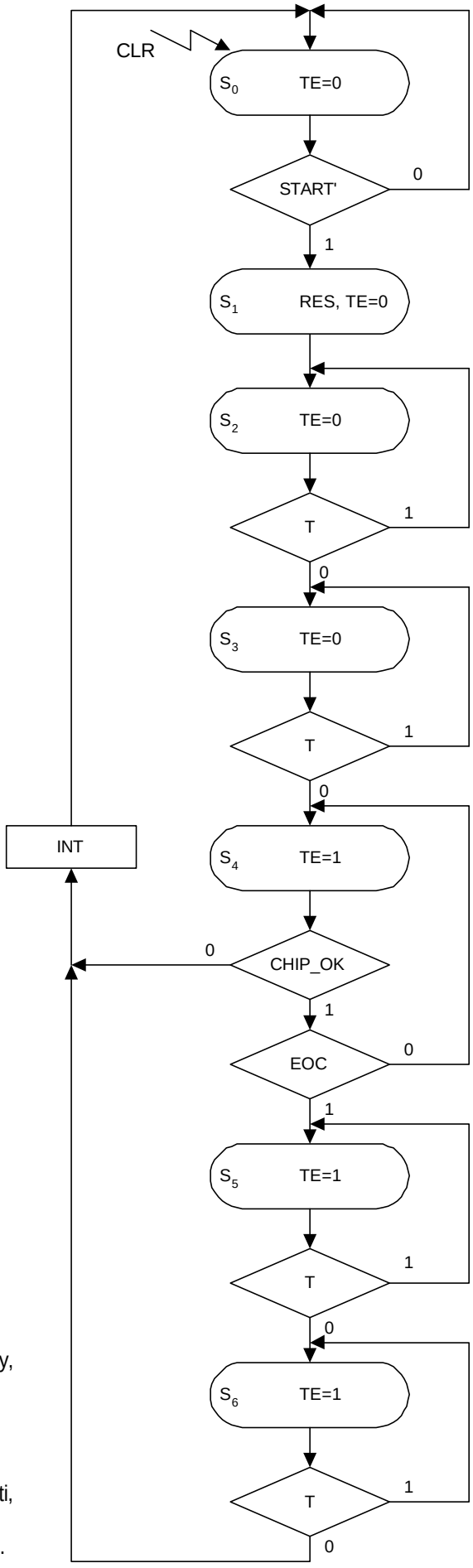
Indirizzi	ROM #3					ROM #2					
0	$E_{M-1,,15}$	$E_{M-1,,14}$	…	$E_{M-1,,9}$	$E_{M-1,,8}$	$E_{M-1,,7}$	$E_{M-1,,6}$	…	$E_{M-1,,1}$	$E_{M-1,,0}$	Vettore risposte #32k-1
1	$E_{M-1,,31}$	$E_{M-1,,30}$	…	$E_{M-1,,25}$	$E_{M-1,,24}$	$E_{M-1,,23}$	$E_{M-1,,22}$	…	$E_{M-1,,17}$	$E_{M-1,,16}$	
…	…	…	…	…	…	…	…	…	…	…	
7	$E_{M-1,,127}$	$E_{M-1,,126}$	…	$E_{M-1,,121}$	$E_{M-1,,120}$	$E_{M-1,,119}$	$E_{M-1,,118}$	…	$E_{M-1,,113}$	$E_{M-1,,112}$	
…	…	…	…	…	…	…	…	…	…	…	
256k-8	$E_{L-2,15}$	$E_{L-2,14}$	…	$E_{L-2,9}$	$E_{L-2,8}$	$E_{L-2,7}$	$E_{L-2,6}$	…	$E_{L-2,1}$	$E_{L-2,0}$	Vettore risposte #64k-2
256k-7	$E_{L-2,31}$	$E_{L-2,30}$	…	$E_{L-2,25}$	$E_{L-2,24}$	$E_{L-2,23}$	$E_{L-2,22}$	…	$E_{L-2,17}$	$E_{L-2,16}$	
…	…	…	…	…	…	…	…	…	…	…	
256k-1	$E_{L-2,127}$	$E_{L-2,126}$	…	$E_{L-2,121}$	$E_{L-2,120}$	$E_{L-2,119}$	$E_{L-2,118}$	…	$E_{L-2,113}$	$E_{L-2,112}$	

M=32k; L=64k

TESTER: temporizzazioni



TESTER: SCO Flow-Chart



TE=0 per mantenere il valore di CHIP_OK dopo INT fino a INTA (CHIP_OK è nell'IVN), mentre i contatori continuano a girare.

TE=0 assicura che RES=1 setti CHIP_OK=1.

Ciclo di attesa dei primi 16 periodi di CK, necessari a caricare il PISO con il primo pacchetto di 16 bit del primo vettore di test; in questa fase TE=0 protegge CHIP_OK=1.

Ciclo di attesa dei successivi 128 (+1) periodi di CK, necessari a caricare il registro di stato del chip LSI esterno con il primo vettore di test; in questa fase TE=0 protegge CHIP_OK=1.

Ciclo operativo del test. In questa fase TE=1 abilita l'eventuale azzeramento di CHIP_OK=1.

Ciclo di attesa dei primi 16 (+1) periodi di CK dopo il trabocco dei contatori. E' necessario per completare il caricamento degli ultimi 16 bit dell'ultimo vettore di test nel registro del chip LSI esterno sotto test.

Ciclo di attesa dei successivi 128 (+1) periodi di CK, necessari a scaricare il vettore di 128 bit (risultato dell'applicazione dell'ultimo vettore di test) dal registro di stato del chip LSI esterno sotto test.

Macchina di Mealy, le cui uscite indipendenti dagli ingressi sono state associate agli stati, per semplicità di rappresentazione.

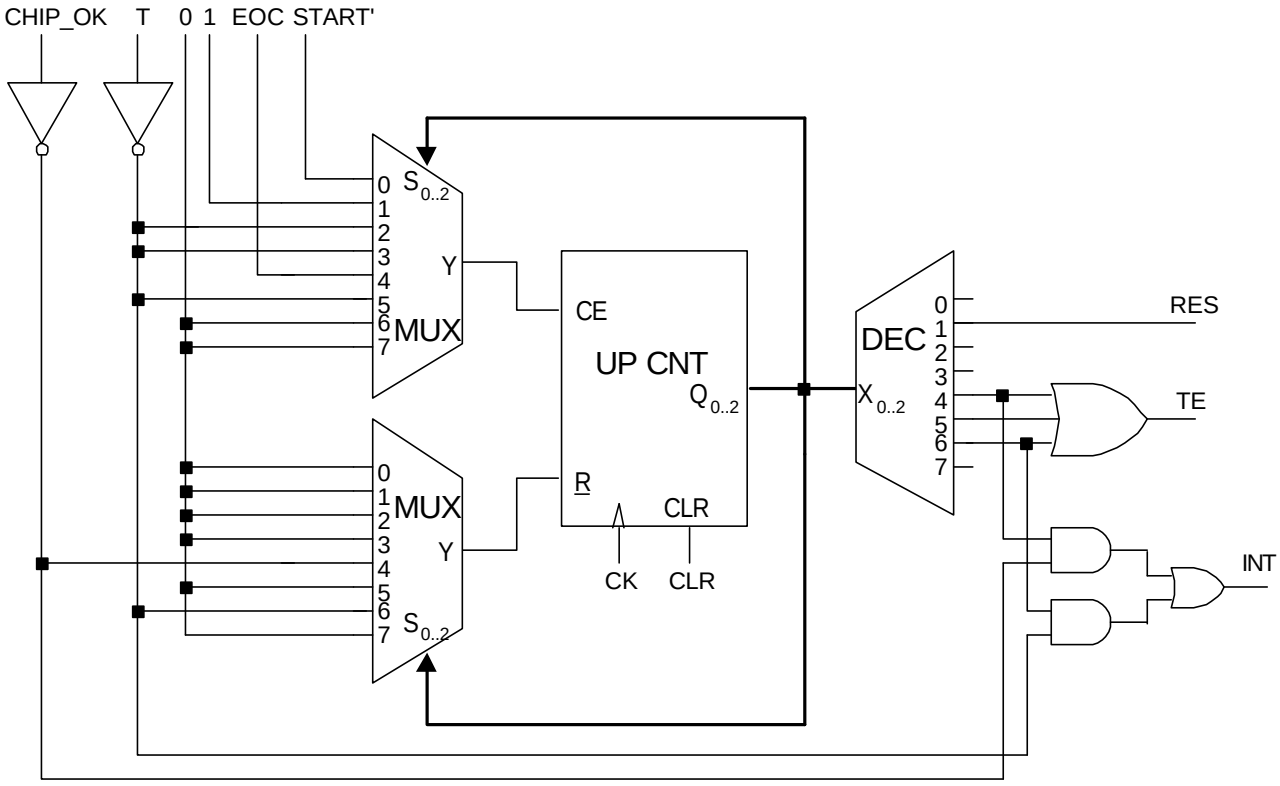
TESTER: SCO

Il diagramma degli stati può essere implementato con un controllore di tipo Mealy e con struttura MSI.
Le funzioni logiche costitutive delle reti combinatorie che calcolano CE (abilitazione al conteggio), R (azzeramento) del contatore e le uscite del controllore sono desumibili dal diagramma degli stati, tenendo conto della prevalenza dell'ingresso R su CE del contatore:

$CE = S_O\ START' + S_1 + S_2\ T^* + S_3\ T^* + S_4\ EOC + S_5\ T^*$
$R = S_4\ CHIP\_OK^* + S_6\ T^*$

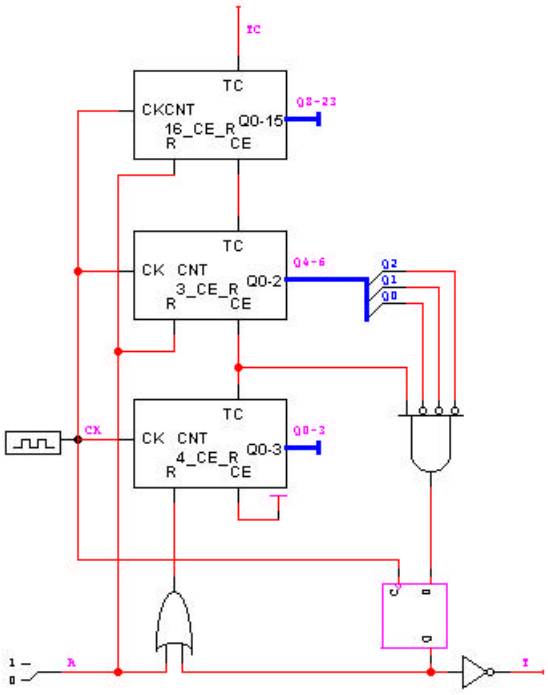
$TE = S_4 + S_5 + S_6$
$RES = S_1$
$INT = S_4\ CHIP\_OK^* + S_6\ T^* = R$

dove si è usato il simbolo * per indicare negazione.



L'identità INT = R evidenzia che si potrebbe eliminare il MUX che produce R e collegare il segnale INT all'ingresso R del contatore (cfr. la semplificazione della rete interna del MUX dovuta ai 6 ingressi posti a 0). Tuttavia in una realizzazione reale può essere preferibile mantenere la struttura generale del controllore, in quanto modificabile più facilmente nelle eventuali fasi evolutive del progetto (per esempio si potrebbe dover tornare nello stato codificato con 0 senza attivare INT).

CIRCUITO PER IL SIMULATORE DESIGN-WORKS



Per simulare il circuito raffigurato (**test_drive.cct**):

1. Copia il file test_drive.cct (contiene il circuito mostrato nella figura) e il file tester.clf (contiene la libreria dei componenti istanziati in test_drive.cct e nei suoi blocchi funzionali) in una stessa cartella;
2. Lancia il programma Design-Works;
3. Apri il file test_drive.cct; si aprirà un foglio di lavoro con il circuito mostrato nella figura;
4. Nel menu **Tools** seleziona **Simulator**; si aprirà la finestra di controllo della simulazione;
5. Nel menu **Tools** seleziona **Timing**; si aprirà la finestra delle forme d'onda;
6. Seleziona ciascuna linea etichettata e nel menu **Timing** seleziona **Add To Timing**;
7. Nel menu **Speed** della finestra del simulatore seleziona una qualsiasi velocità (diversa da Stop);
8. Con il mouse clicca sullo switch binario della linea R in modo che questa valga 1, e poi clicca di nuovo sullo switch per riportare il segnale R a 0;
9. A questo punto nella finestra delle forme d'onda vengono visualizzati i segnali.

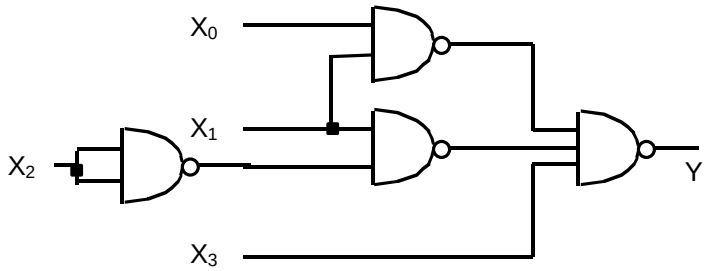
RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 15-07-99

Studente:_____ Docente:_____

D1 Rappresentare con la notazione in virgola mobile in base 2 i numeri 1999.25 e 1.75 e descrivere i passi per effettuarne la somma.

D2 Trasformare la rete combinatoria in figura in una rete equivalente a tre livelli costituita esclusivamente da porte logiche di tipo NOR.



D3 Dato un registro a scalamento a 8 bit, modificarlo in modo che la lunghezza risulti programmabile da 1 a 8 bit, in funzione dello stato logico di 3 bit di controllo.

D4 Dato lo SCA del PD32, definire un microprogramma per interpretare la fase di esecuzione dell'istruzione ipotetica XNORL R0,R1. Indicare su un diagramma temporale la dinamica dei bit di TASK e dei contenuti dei registri coinvolti.

D5 Una periferica utilizza una sezione della memoria del PD32 per appoggiare dati temporanei secondo la disciplina LIFO. Ciascun dato a 32 bit viene depositato e successivamente recuperato dalla periferica tramite interruzione al processore. Una coppia di FF con significato di stack vuoto e stack pieno vengono settati dal micro per prevenire eventuali richieste anomale da parte della periferica.
Definire l'interfaccia HW di I/O della periferica ed i moduli del programma assembler PD32.

Esercizio (2S19990715-D1)

Rappresentare con la notazione in virgola mobile in base 2 i numeri 1999.25 e 1.75 e descrivere i passi per effettuarne la somma.

---

1. Conversione di 1999.25 in codice binario (virgola fissa)

Conversione parte intera

N	**:2**	**I'**	**F'**
1999	999	1	
999		499	1
499		249	1
249		124	1
124		62	0
62		31	0
31		15	1
15		7	1
7		3	1
3		1	1
1		0	1

Conversione parte frazionaria

N	**x2**	**I'**	**F'**
0.25		0	0.50
0.50		1	0

$1999.25_{10} = 11111001111.01_2$

2. Per la conversione in virgola mobile, si assuma il formato a 32 bit, con 1 bit di segno (S), 8 di esponente (E) e 23 per la mantissa (M), con 0.5<= M< 1;

S = 0
M = 11111001111010000000000 (13 bit originali + 10 zeri a destra)
$E = 11_{10} = 00001011_2$

3. Conversione di 1.75 in codice binario (virgola fissa)

Conversione parte intera

N	**:2**	**I'**	**F'**
1		0	1

Conversione parte frazionaria

N	**x2**	**I'**	**F'**
0.75		1	0.50
0.50		1	0

$1.75_{10} = 1.11_2$

4. Conversione in virgola mobile:

S = 0
M = 11100000000000000000000 (i 3 bit originali + 20 zeri a destra)
E = 1_{10} = 00000001_2

5. Per la somma aritmetica, l'esponente minore viene sostituito con il maggiore e la mantissa relativa viene denormalizzata consistentemente con una traslazione a destra di un numero di posizioni (bit) pari alla differenza tra i due esponenti:

E(1999.25) = 11 > 1 = E(1.75) $\Rightarrow$

$\Rightarrow$ 1.75 viene riscritto come:

S' = 0
E' = 11_{10} = 00001011_2
M' = M: 00000000001110000000000 (traslazione di 11-1=10 bit a destra)

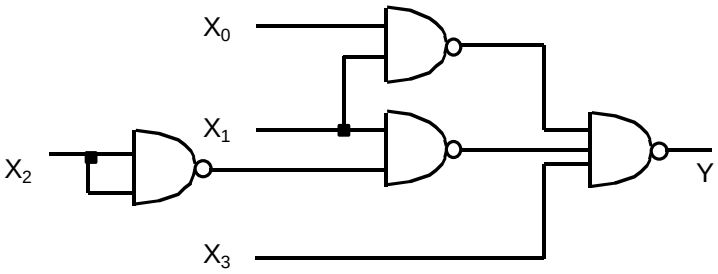
6. Ora si può procedere al calcolo della somma:

S = 0
E = 11_{10} = 00001011_2
M = 00000000001110000000000 +
<u>11111001111010000000000</u> =
11111010001000000000000

L'addizione non ha prodotto trabocco, per cui il risultato della somma (che vale 2001_{10}) è già normalizzato.

Esercizio (2S19990715-D2)

Trasformare la rete combinatoria in figura in una rete equivalente a tre livelli costituita esclusivamente da porte logiche di tipo NOR.



Questo problema può essere risolto in due fasi: 1) **analisi** della rete e identificazione dell'espressione logica della funzione y; 2) **sintesi** della funzione disponibile y nella forma canonica NOR-NOR.

1) **Analisi**

Si tratta di una rete combinatoria a due livelli di porte più il terzo livello di inversione; pertanto l'espressione di y si può trascrivere senza passare attraverso le espressioni logiche associate ai nodi intermedi:

$$y = x_o \cdot x_1 + x_1 \cdot \overline{x_2} + \overline{x_3}$$

2) **Sintesi**

Conviene trasferire la funzione sulla tavola di verità, per poi passare alla minimizzazione con le MK.

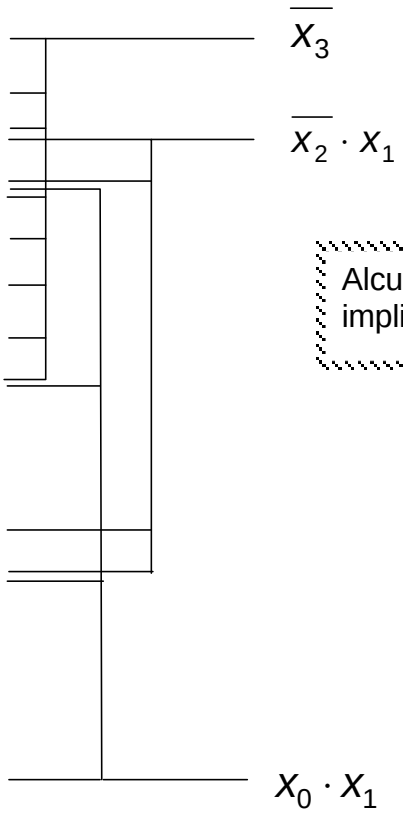
Data l'espressione di y come somma di prodotti, si può attribuire il valore 1 a y sulla tavola di verità in corrispondenza delle righe (x_3 x_2 x_1 x_0) associate agli implicanti:

$x_o \cdot \overline{x_1}$ implica le 4 righe caratterizzate dalla configurazione - - 1 1

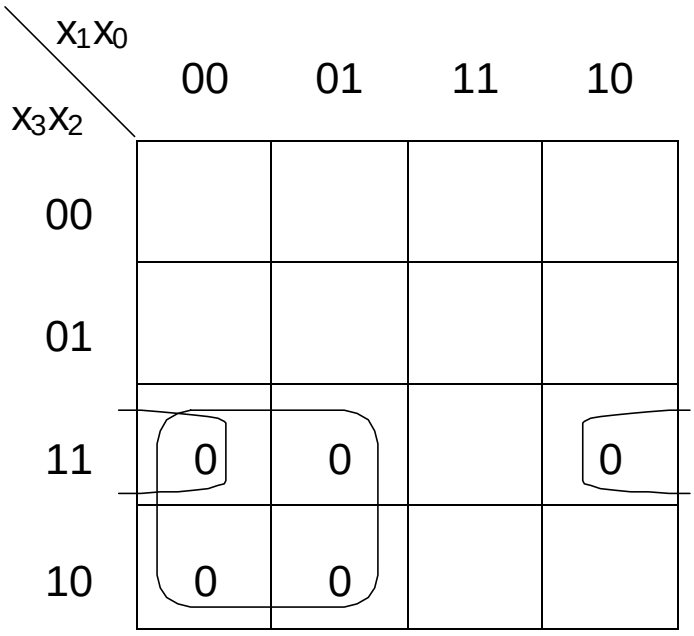
$x_1 \cdot \overline{x_2}$ implica le 4 righe caratterizzate dalla configurazione - 0 1 -

$\overline{x_3}$ implica le 8 righe caratterizzate dalla configurazione 0 - - -

x_3	x_2	x_1	x_0	y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



Alcune righe vengono implicate più volte.



Sulla MK conviene riportare gli zeri (che sono anche prevalenti), per predisporla alla sintesi della funzione nella forma AND di OR, cioè NOR-NOR

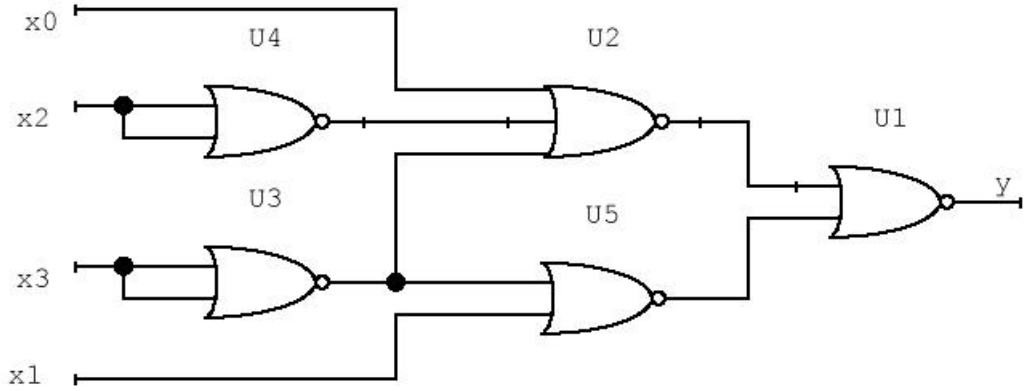
L’espressione minima in prodotto di somme è:

$$y = (\overline{x_3} + x_1) \cdot (\overline{x_3} + \overline{x_2} + x_0)$$

che può essere trasformata direttamente nella forma NOR-NOR:

$$y = ((x_3 \downarrow x_3) \downarrow x_1) \downarrow ((x_3 \downarrow x_3) \downarrow (x_2 \downarrow x_2) \downarrow x_0)$$

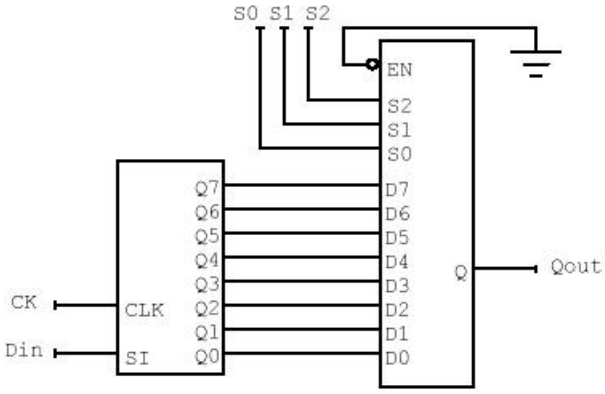
A questo punto può essere tracciata la rete logica richiesta:



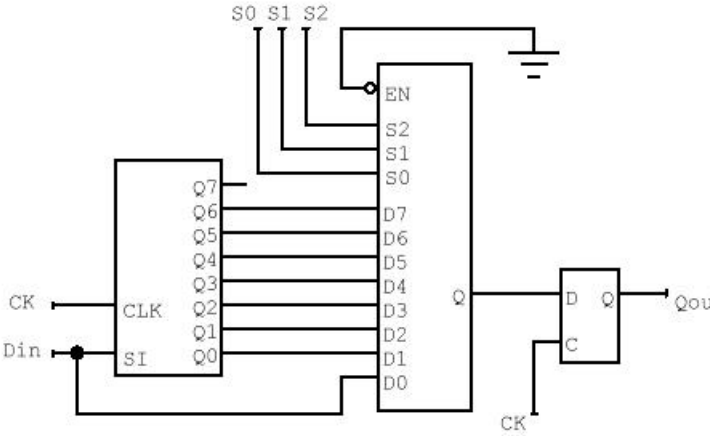
Esercizio (2S19990715-D3)

Dato un registro a scalamento a 8 bit, modificarlo in modo che la lunghezza risulti programmabile da 1 a 8 bit, in funzione dello stato logico di 3 bit di controllo.

Le uscite dei flip-flop interni al registro a scalamento rappresentano altrettante versioni dell'ingresso ritardate di 1, 2,..,8 cicli di clock. Pertanto, è sufficiente selezionare con un multiplexer tali linee esternamente al registro a scalamento. I tre bit di controllo controlleranno gli ingressi di selezione del MUX, come riportato nella figura seguente.



Va notato che l'uscita Q_{out} è prelevata dal MUX, ed è perciò affetta da alee statiche, a seguito delle commutazioni su S_0, S_1, S_2. Come è noto, è possibile eliminare le alee sull'uscita mediante l'inserimento di un flip-flop di sincronizzazione, che però introduce una latenza aggiuntiva di un ciclo di clock. Tuttavia, è possibile compensare la latenza del flip-flop di uscita riducendo di un bit il percorso da Din all'uscita del MUX; in definitiva la struttura può essere migliorata secondo la figura seguente.



Ovviamente il segnale di clock CK è lo stesso per il registro a scalamento e il flip-flop di sincronizzazione.

Esercizio (2S19990715-D4)

Dato lo SCA del PD32, definire un microprogramma per interpretare la fase di esecuzione dell'istruzione ipotetica

XNORL R0,R1

Indicare su un diagramma temporale la dinamica dei bit di TASK e dei contenuti dei registri coinvolti.

La metodologia generale per affrontare questa classe di problemi è trattata in dettaglio negli "Appunti Integrativi". In questo caso particolare va notato che il repertorio delle istruzioni PD32 non include l'operazione XNOR, che quindi andrà risolta nella successione XOR – NOT, il che richiede un doppio passaggio per
Pertanto il microprogramma dovrà eseguire la sequenza di operazioni seguente:
1: R0 → T1
2: R1 → T2
3: T1 ⊕ T2 → T1
4: NOT(T1) → R1
A questo punto si possono descrivere le microistruzioni con i bit TASK e la relativa temporizzazione.

Esercizio (2S19990715-D5)

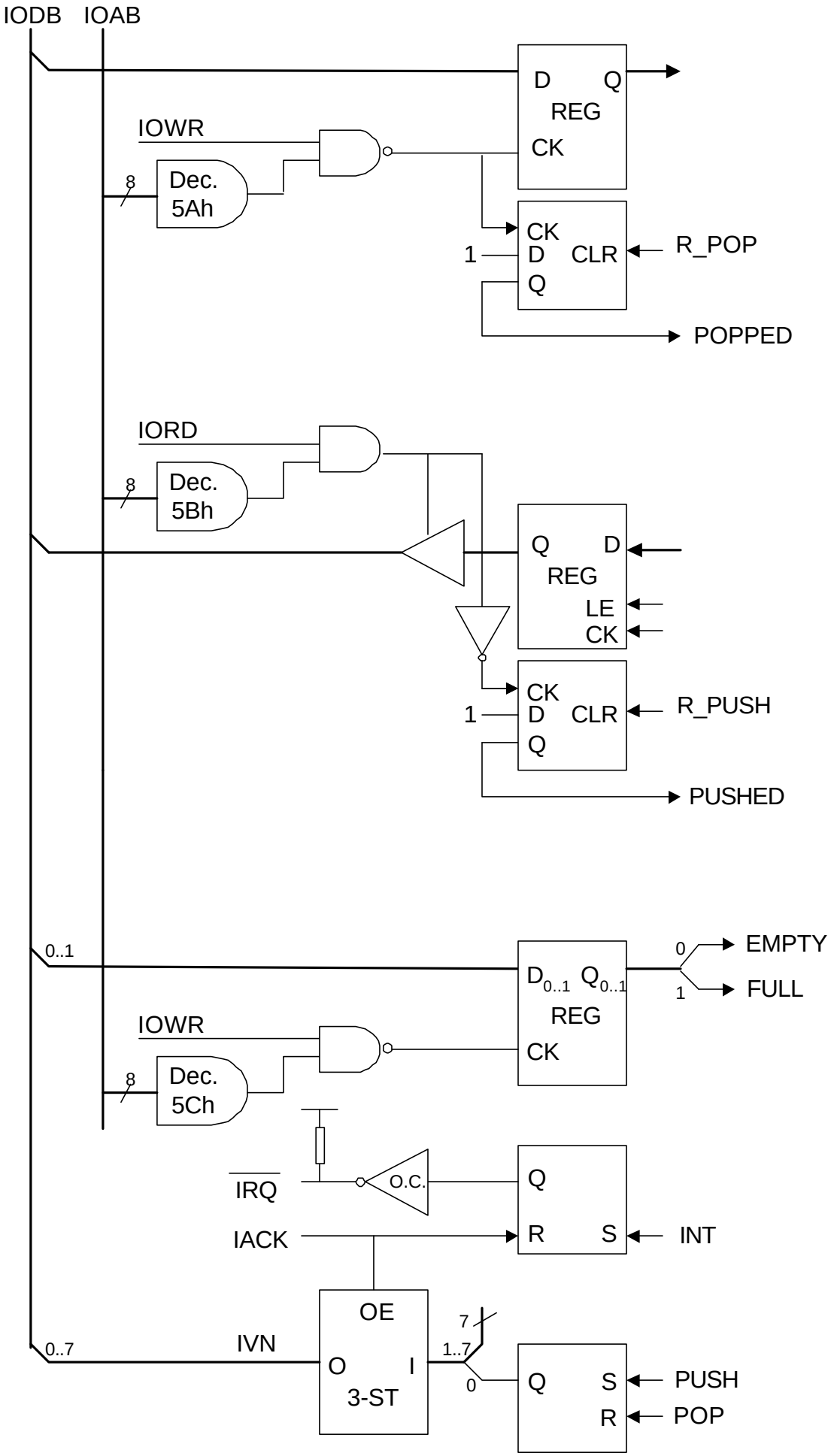
Una periferica utilizza una sezione della memoria del PD32 per appoggiare dati temporanei secondo la disciplina LIFO. Ciascun dato a 32 bit viene depositato e successivamente recuperato dalla periferica tramite interruzione al processore. Una coppia di FF con significato di stack vuoto e stack pieno vengono settati dal micro per prevenire eventuali richieste anomale da parte della periferica.
Definire l'interfaccia HW di I/O della periferica ed i moduli del programma assembler PD32.

La periferica deve poter eseguire le due operazioni di push e pop di un singolo dato mediante l'intervento del processore, il quale scambia il dato con la periferica sui bus di I/O - su base interruzione - e legge/scrive il dato in una sezione della sua memoria gestita a stack; si noti che questa area deve essere tenuta distinta dall'area di stack del processore; pertanto il software dovrà anche definire un puntatore dedicato a questa area di memoria gestita a stack su comando della periferica.

Interfaccia HW

L'interfaccia di I/O come descritta nella figura seguente include:

- un registro di input, su cui la periferica scrive il dato che vuole caricare (push) nell'area di stack, ed un flip-flop di handshake;
- un registro di output, su cui il processore restituisce alla periferica il dato che affiora dallo stack; anche a questo registro è associato un flip-flop di handshake;
- un registro a due bit (FULL, EMPTY) su cui il processore scrive lo stato di pieno/vuoto dello stack; per evitare di aggiungere un ulteriore flip-flop di handshake associato a questo registro, si può organizzare il software in modo tale che questo registro venga aggiornato prima dell'operazione di scrittura o lettura dei due registri su cui vengono scambiati i dati;
- una semplice interfaccia di richiesta e riconoscimento di interruzione; la soluzione proposta (che chiaramente non è l'unica possibile!) include la codifica del tipo di richiesta (push/pop) direttamente nel vettore di interruzione (IVN), il cui bit 0 è stato prelevato dall'uscita di un flip-flop che lo SCO della periferica carica con il valore 1/0 quando vuole richiedere una operazione di push/pop. Tale accorgimento fa risparmiare un indirizzo di I/O su cui quel flip-flop dovrebbe essere reso accessibile in lettura al micro.


IODB
IOAB
D
Q
REG
CK
IOWR
8
Dec.
5Ah
CK
1
D
CLR
R_POP
Q
POPPED
IORD
8
Dec.
5Bh
Q
D
REG
LE
CK
CK
1
D
CLR
R_PUSH
Q
PUSHED
0..1
D0..1
Q0..1
REG
CK
0
EMPTY
1
FULL
IOWR
8
Dec.
5Ch
IRQ
O.C.
Q
IACK
R
S
INT
OE
7
0..7
IVN
O
I
1..7
3-ST
Q
S
PUSH
R
POP

Routine Software

```
; lifo.asm

;La periferica deve poter eseguire le due operazioni di push
;e pop di un singolo dato
;mediante l'intervento del processore, il quale scambia il dato
;con la periferica sui bus di I/O - su base interruzione –
;e legge/scrive il dato in una sezione della sua memoria
;gestita a stack; si noti che questa area deve essere tenuta
;distinta dall'area di stack del processore;
;pertanto il software dovrà anche definire un puntatore dedicato
;a questa area di memoria gestita a stack su comando della periferica.


;*************************
; COSTANTI
;*************************

org 500h                    ;richiesta dall'assemblatore PD32 prima delle equ

pushreg     equ 05Ah    ;indirizzo reg. push (in)
popreg      equ 05Bh    ;indirizzo reg.         pop (out)
fullmpty    equ 05Ch    ;indirizzo reg. 2 bit (out)

inivn       equ 000h    ;indirizzo driver periferica lifo in (push)
outivn equ 001h         ;indirizzo driver periferica lifo out (pop)

indriver    equ 1000h   ;indirizzo driver periferica lifo in (push)
outdrive    equ 1100h   ;indirizzo driver periferica lifo out (pop)

stackbot    equ 500h    ;base dello stack della periferica
stacktop    equ 400h    ;cima dello stack della periferica:
                        ;stack ampio 256 byte e gestito dall'alto al basso

stack       equ 2800h   ;inizio area di stack PD32


;*************************
; VARIABILI
;*************************

stackpnt    dl 000h


;*************************
; CODICE
;*************************

code                                ;inizio istruzioni
```

```
main:
;       clri                            ;interruzioni disabilitate al reset

        movl #stackbot,stackpnt         ;stackpnt puntatore all'area
                                        ;del messaggio out

        movl #stack,r7                  ;inizializza R7 quale SP: deve precedere
                                        ;l'istruzione SETI
                                        ;per gestire correttamente lo stack nella
                                        ;fase di riconoscimento di una richiesta
                                        ;di interruzione

        outl #01h,fullmpty              ;programmazione reg. LIFO vuota

        seti                            ;abilita PD32 ad accettare interruzioni
                                        ;(SP è stato inizializzato)

mainloop:

;programma principale

        jmp mainloop



;*************************
; Sezione DRIVER
;*************************


;*************************
;lifo in (push)
;in
        driver inivn, indriver  ;Il driver della lifo in (push) ha IVN=inivn
                                ;e inizia dall'ind. indriver
        push r0

        movl stackpnt,r0        ;test su disponibilità residua dello stack
        cmpl #stacktop,r0
        jz exitin               ;ignora richiesta di push (stack pieno!)

        subl #4,r0              ;aggiorna puntatore allo stack (predecremento)
        movl r0,stackpnt
        cmpl #stacktop,r0
        jnz nofull
        outb #02h,fullmpty      ;scrivi 1-0 nei bit full-empty del reg. fullmpty
        jmp pushdato
nofull:
        outb #00h,fullmpty      ;scrivi 0-0 nei bit full-empty del reg. fullmpty
        jmp pushdato
pushdato:
```

```
        inl pushreg,(r0)        ;leggi dato e caricalo nello stack

exitin:
        pop r0
        rti
;*************************


;*************************
;lifo out (pop)
;out
        driver outivn, outdrive ;Il driver della lifo out (pop) ha IVN=outivn
                                ;e inizia dall'ind. outdrive
        push r0

        movl stackpnt,r0        ;test su disponibilità residua dello stack
        cmpl #stackbot,r0
        jz exitout              ;ignora richiesta di pop (stack vuoto!)

        cmpl #stackbot-4,r0
        jnz noempty
        outb #01h,fullmpty      ;scrivi 0-1 nei bit full-empty del reg. fullmpty
        jmp popdato
noempty:
        outb #00h,fullmpty      ;scrivi 0-0 nei bit full-empty del reg. fullmpty
        jmp popdato
popdato:
        outl (r0),popreg        ;estrai dato dallo stack e passalo alla perifer.

        addl #4,r0              ;aggiorna puntatore allo stack (postincremento)
        movl r0,stackpnt

exitout:
        pop r0
        rti
;*************************


end                             ;fine programma
```

```
; lifo.asm

;Una periferica utilizza una sezione della memoria del PD32 per appoggiare
;dati temporanei secondo la disciplina LIFO. Ciascun dato, a 32 bit, viene
;depositato e successivamente recuperato dalla periferica tramite interruzione
;al processore. Una coppia di FF con significato di stack vuoto e stack pieno
;vengono settati dal micro per prevenire eventuali richieste anomale da parte della
periferica.

;La periferica deve poter eseguire le due operazioni di push e pop di un singolo dato
;mediante l'intervento del processore, il quale scambia il dato con la periferica sui
bus
;di I/O - su base interruzione - e legge/scrive il dato in una sezione della sua
memoria
;gestita a stack; si noti che questa area deve essere tenuta distinta dall'area di
stack
;del processore; pertanto il software dovrà anche definire un puntatore dedicato
;a questa area di memoria gestita a stack su comando della periferica.


;**************************
; COSTANTI
;**************************


org 500h      ;richiesta dall'assemblatore PD32 prima delle equ

pushreg   equ 05Ah  ;indirizzo reg. push (in)
popreg    equ 05Bh  ;indirizzo reg. pop (out)
fullmpty  equ 05Ch  ;indirizzo reg. 2 bit (out)

inivn   equ 000h  ;indirizzo driver periferica lifo in (push)
outivn    equ 001h  ;indirizzo driver periferica lifo out (pop)

indriver  equ 1000h ;indirizzo driver periferica lifo in (push)
outdrive  equ 1100h ;indirizzo driver periferica lifo out (pop)

stackbot  equ 500h  ;base dello stack della periferica
stacktop  equ 400h  ;cima dello stack della periferica:
        ;stack ampio 256 byte e gestito dall'alto al basso

stack   equ 2800h ;inizio area di stack PD32


;**************************
; VARIABILI
;**************************

stackpnt  dl 000h


;**************************
; CODICE
;**************************

code        ;inizio istruzioni

main:
; clri      ;interruzioni disabilitate al reset

  movl #stackbot,stackpnt ;stackpnt puntatore all'area del messaggio out

  movl #stack,r7    ;inizializza R7 quale SP: deve precedere
```

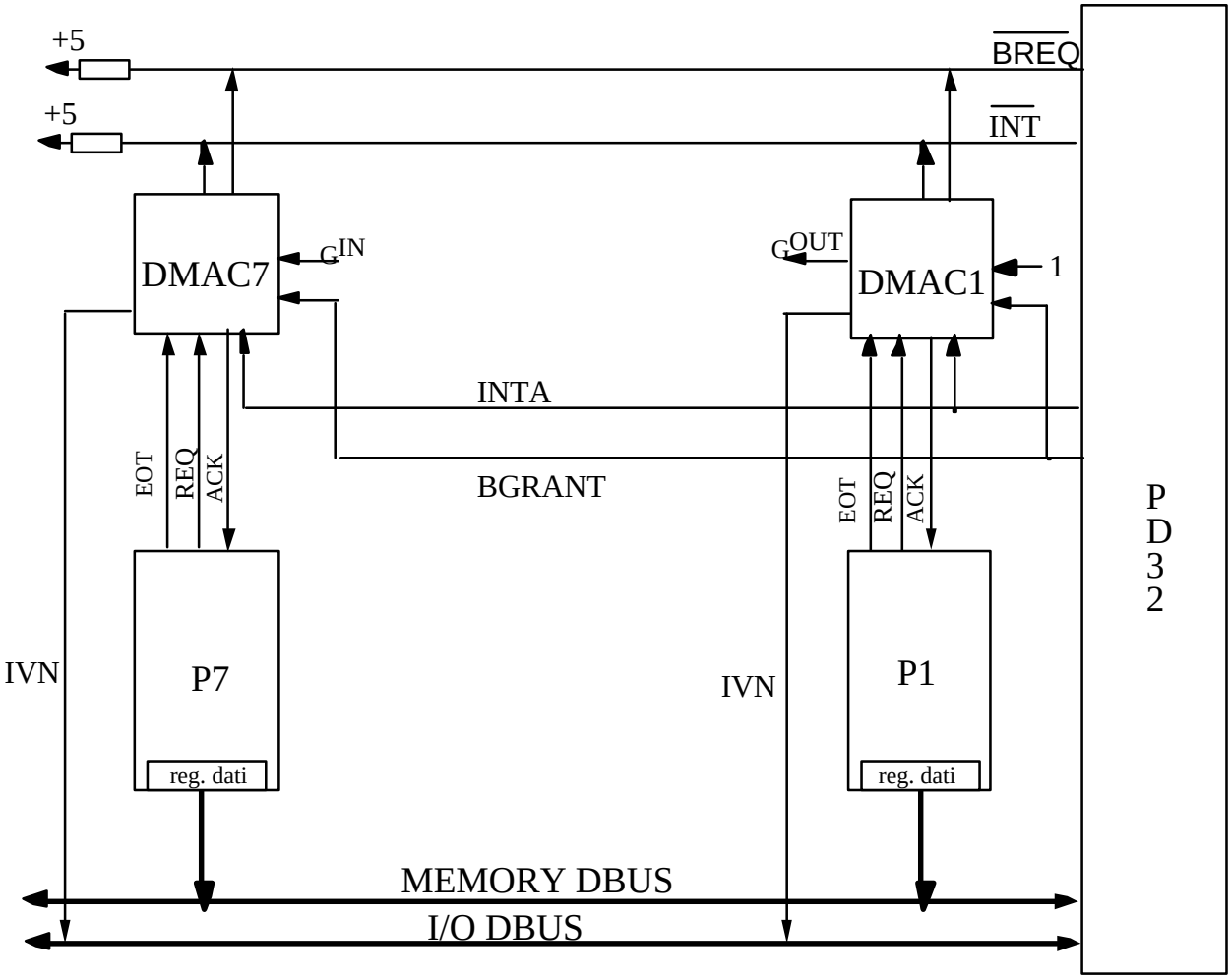
```
        ;l'istruzione SETI
        ;per gestire correttamente lo stack nella
        ;fase di riconoscimento di una richiesta
        ;di interruzione

  outl #01h,fullmpty  ;programmazione reg. LIFO vuota

  seti      ;abilita PD32 ad accettare interruzioni (SP è stato
        ;inizializzato)

mainloop:

;programma principale

  jmp mainloop



;*************************
; Sezione DRIVER
;*************************


;*************************
;lifo in (push)
;in
  driver inivn, indriver  ;Il driver della lifo in (push) ha IVN=inivn
        ;e inizia dall'ind. indriver
  push r0

  movl stackpnt,r0  ;test su disponibilità residua dello stack
  cmpl #stacktop,r0
  jz exitin   ;ignora richiesta di push (stack pieno!)

  subl #4,r0    ;aggiorna puntatore allo stack (predecremento)
  movl r0,stackpnt
  cmpl #stacktop,r0
  jnz nofull
  outb #02h,fullmpty  ;scrivi 1-0 nei bit full-empty del reg. fullmpty
  jmp pushdato
nofull:
  outb #00h,fullmpty  ;scrivi 0-0 nei bit full-empty del reg. fullmpty
  jmp pushdato
pushdato:
  inl pushreg,(r0)  ;leggi dato e caricalo nello stack

exitin:
  pop r0
  rti
;*************************



;*************************
;lifo out (pop)
;out
  driver outivn, outdrive ;Il driver della lifo out (pop) ha IVN=outivn
        ;e inizia dall'ind. outdrive
  push r0

  movl stackpnt,r0  ;test su disponibilità residua dello stack
  cmpl #stackbot,r0
  jz exitout    ;ignora richiesta di pop (stack vuoto!)
```

```
  cmpl #stackbot-4,r0
  jnz noempty
  outb #01h,fullmpty  ;scrivi 0-1 nei bit full-empty del reg. fullmpty
  jmp popdato
noempty:
  outb #00h,fullmpty  ;scrivi 0-0 nei bit full-empty del reg. fullmpty
  jmp popdato
popdato:
  outl (r0),popreg  ;estrai dato dallo stack e passalo alla perifer.

  addl #4,r0    ;aggiorna puntatore allo stack (postincremento)
  movl r0,stackpnt

exitout:
  pop r0
  rti
;*************************


end       ;fine programma
```

RETI LOGICHE
PRIMA PROVA SCRITTA DELL'APPELLO DEL 24-9-99

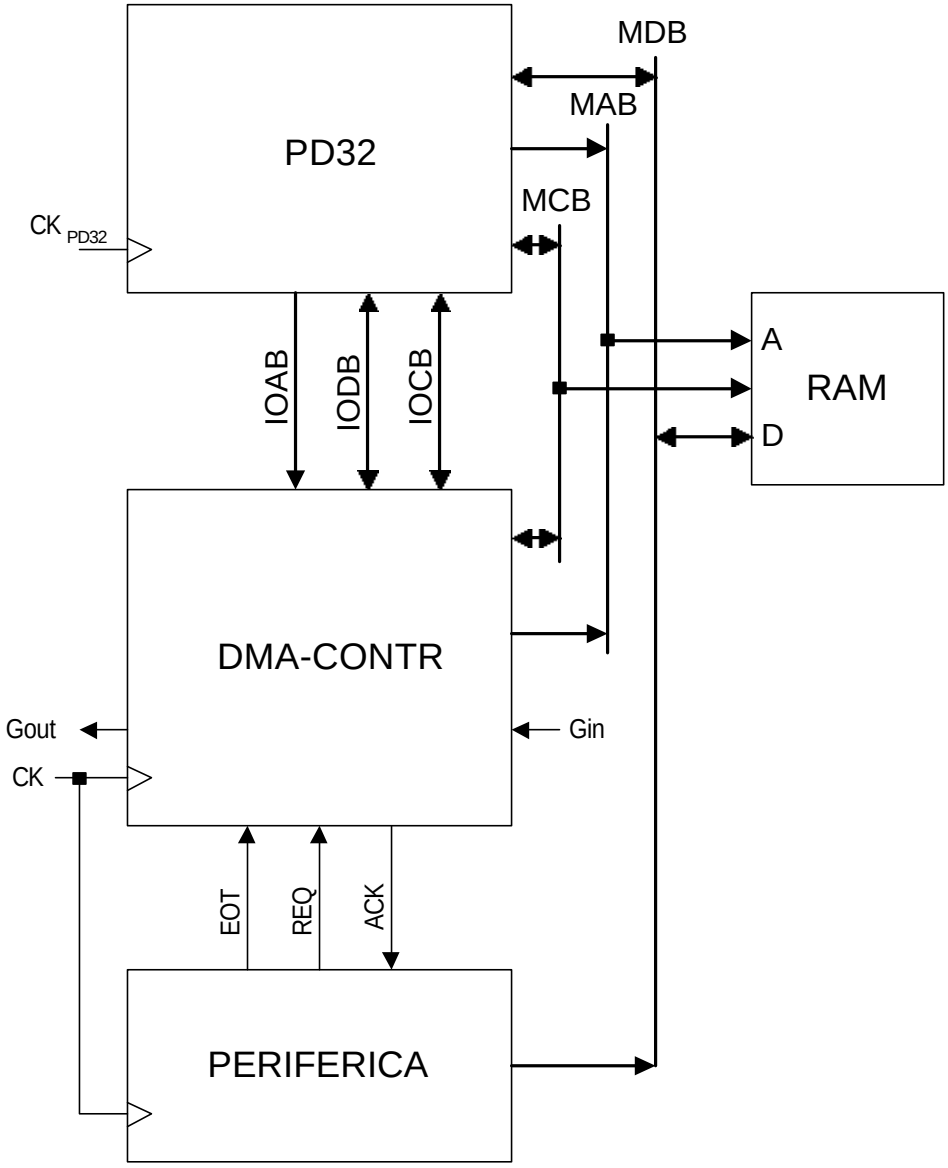
Studente:_____ Docente:_____

Un processore PD32 ha sette periferiche di ingresso operanti in DMA (un DMAC per periferica) e gestite tramite interruzioni a priorità decrescente (P1 priorità massima; P7 priorità minima. Quando la periferica i-ma vuole mettersi in comunicazione colla memoria del PD32, manda una richiesta (REQ) al proprio DMAC. Se il suo G^{IN} è alto, $DMAC_i$ invia l'interruzione al processore per ottenere l'indirizzo di memoria dove depositare i dati, nel qual caso mette a zero il suo G^{OUT} per inibire i DMAC di priorità più bassa e quando riceve il BUSGRANT inizia il trasferimento in bus stealing; DMAC termina il trasferimento quando riceve dalla sua periferica il segnale EOT. Se durante il servizio (interruzione più trasferimento dati dalla Pi alla memoria) una periferica Ph (h<i) chiede a sua volta il servizio, $DMAC_i$ si mette in stand-by (attesa) e riprenderà il servizio al termine del trasferimento dati tra Ph e PD32. Alla fine del servizio il DMAC ritorna a riposo senza ulteriori interruzioni.
Progettare il $DMAC_i$ e disegnare lo schema elettrico dettagliato.

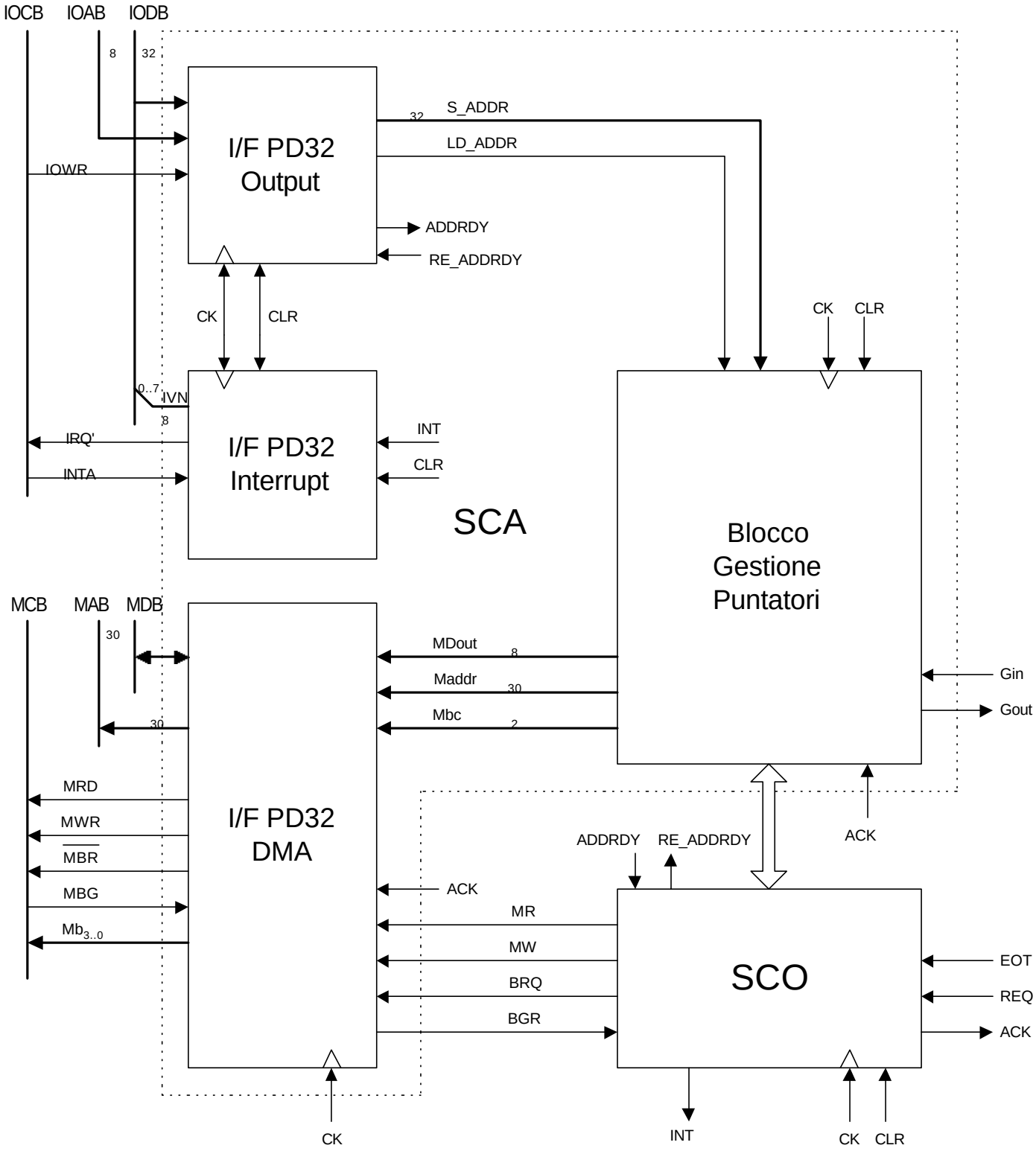


REQ e ACK sono segnali di handshaking tra periferica e DMAC; il primo REQ viene anche interpretato come richiesta di servizio. DMAC inizia il trasferimento non appena riceve l'indirizzo di memoria dal PD32; periferica e DMAC utilizzano lo stesso clock.
Si noti che le interruzioni I2-I7 sono di tipo interrompibile: specificare come devono essere gli inizi delle routine di servizio di P2-P7 per il corretto funzionamento delle interruzioni.

DMA-CONTR: sistema esterno



DMA-CONTR: Schema a blocchi

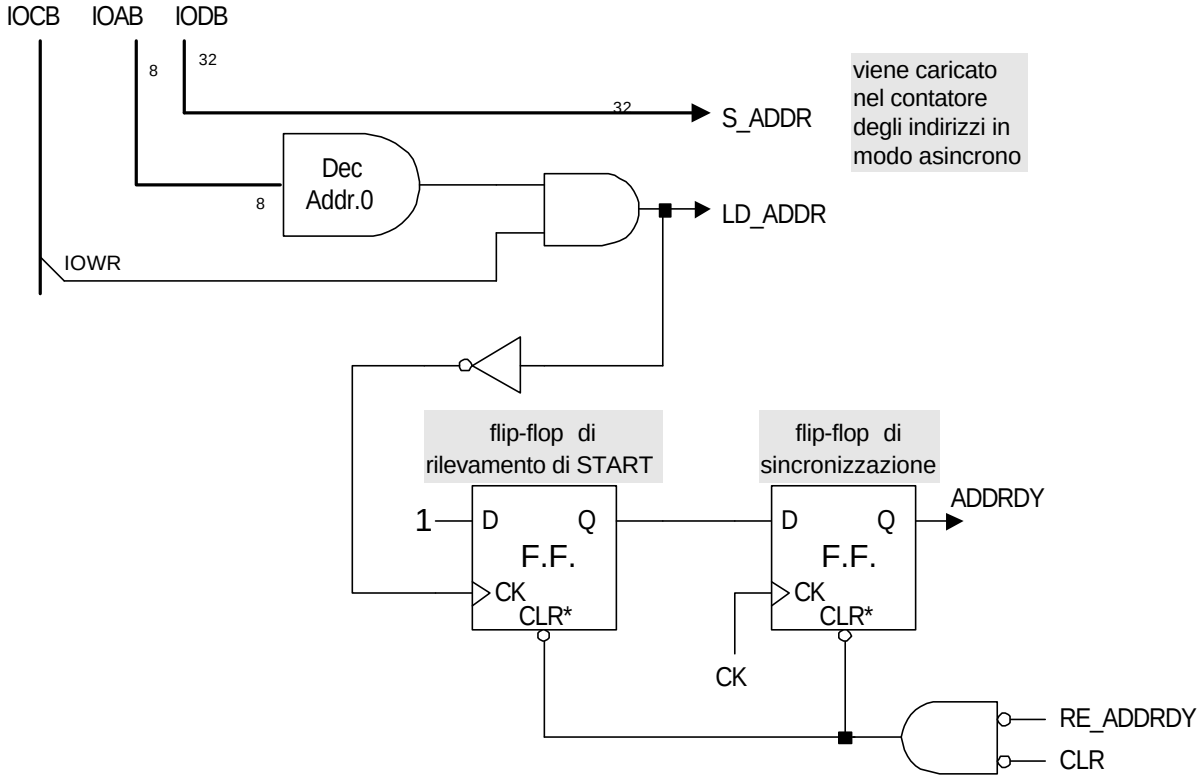


Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.

Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 interrupt usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

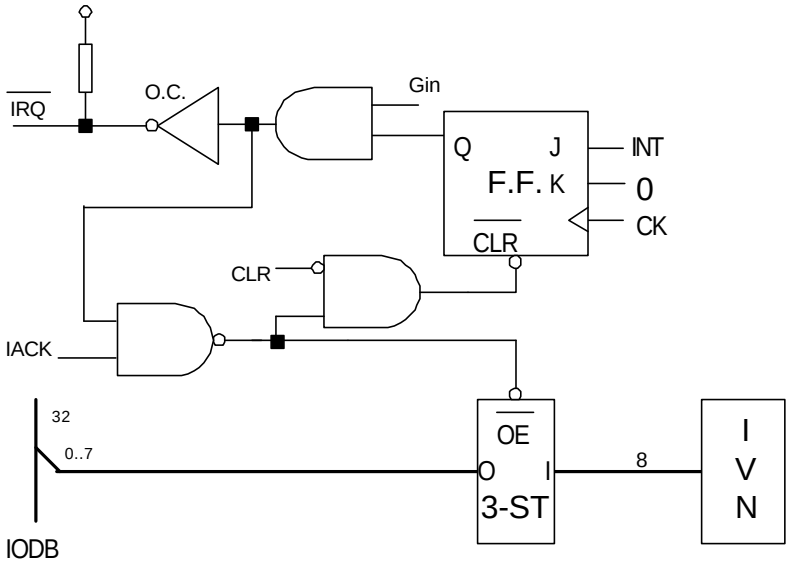
DMA-CONTR: IF PD32 - output



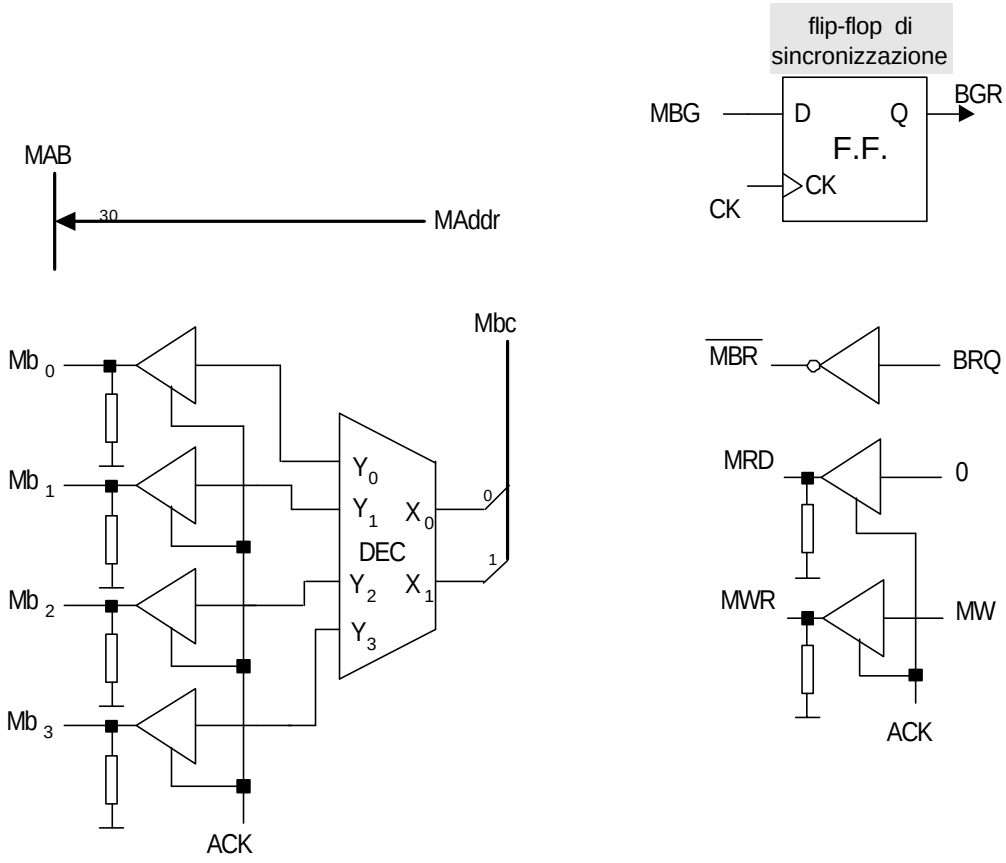
Note

Il SW avvia l'operazione direttamente con la scrittura dell'indirizzo del blocco di dati da trasferire:
OUTL S_ADDR,Addr0

DMA-CONTR: IF PD32 - interrupt



DMA-CONTR: IF PD32 - DMA

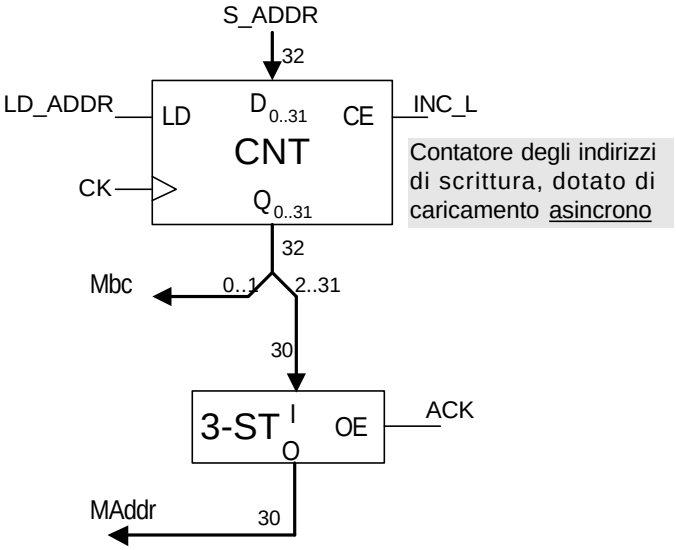


Note

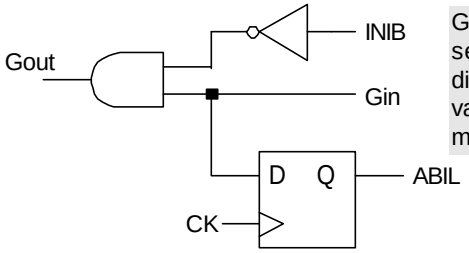
La gestione del Memory Data Bus è affidata alla periferica collegata a DMA-CONTR. Si presuppone che il segnale ACK venga utilizzato nella periferica per abilitare l'uscita del proprio data buffer sul Memory Data Bus.

La struttura hardware supporta il caso più generale di trasferimento di dati a larghezza 8 bit.

DMA-CONTR: blocco gestione puntatori

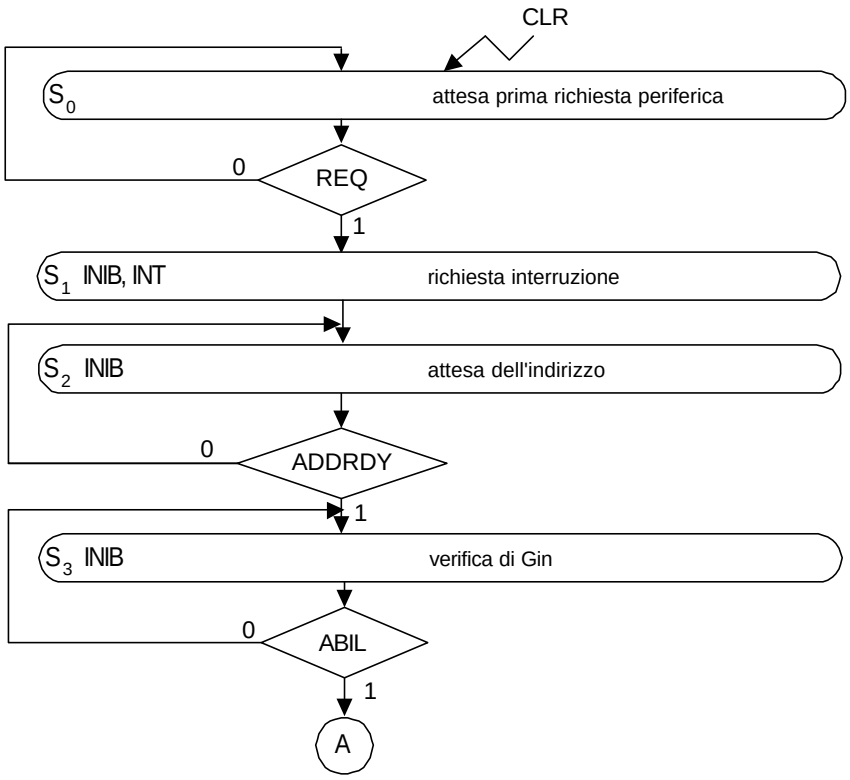


Contatore degli indirizzi di scrittura, dotato di caricamento asincrono



Generazione del segnale di inibizione diretto alle periferiche a valle nella daisy chain, meno prioritarie

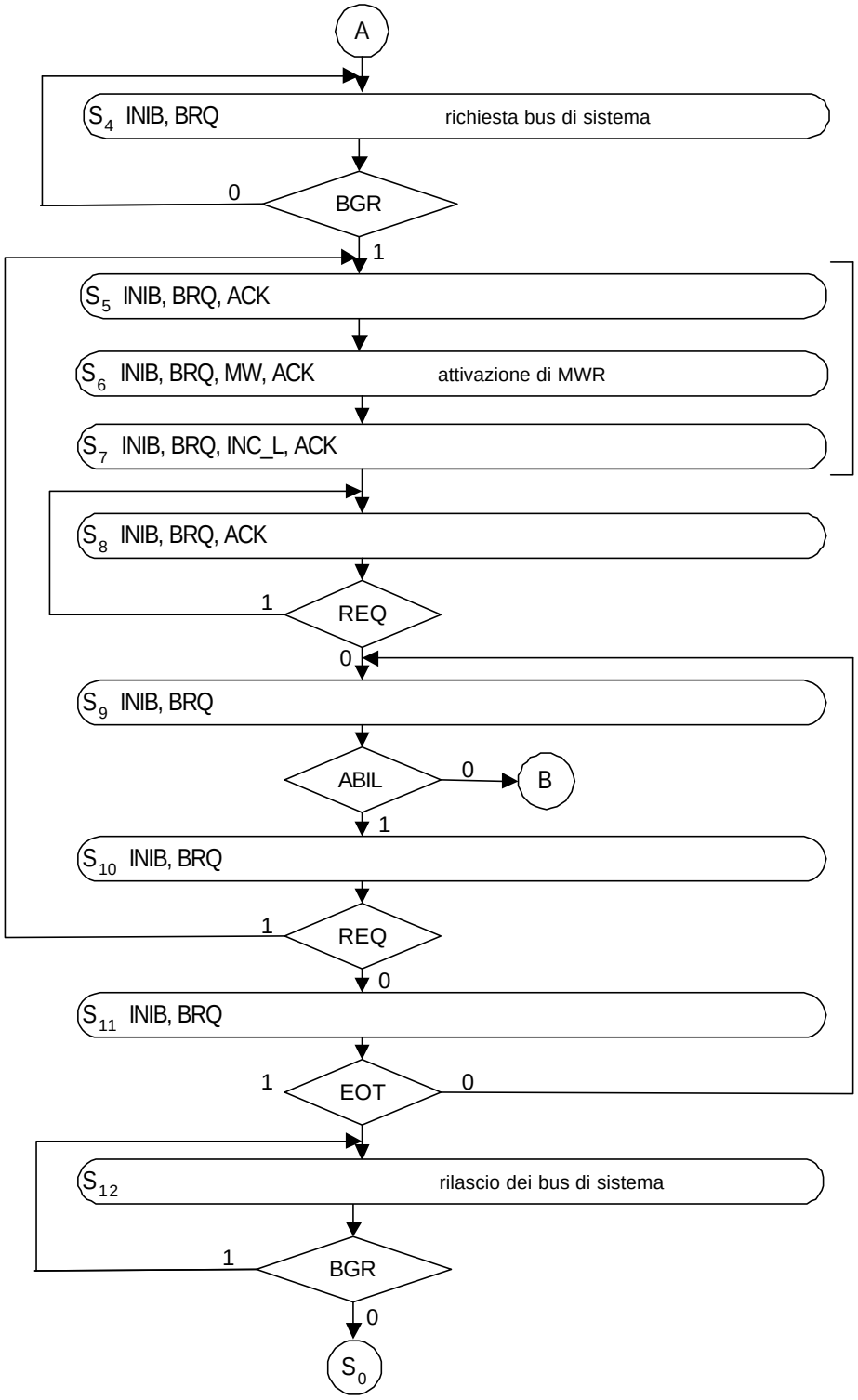
DMA-CONTR: SCO - flowchart - 1



La richiesta effettiva di interruzione è condizionata in hardware da Gin, secondo il meccanismo della daisy-chain

La routine di interruzione potrebbe essere interrotta da una periferica a priorità più elevata subito dopo la scrittura dell'indirizzo

DMA-CONTR: SCO - flowchart - 2

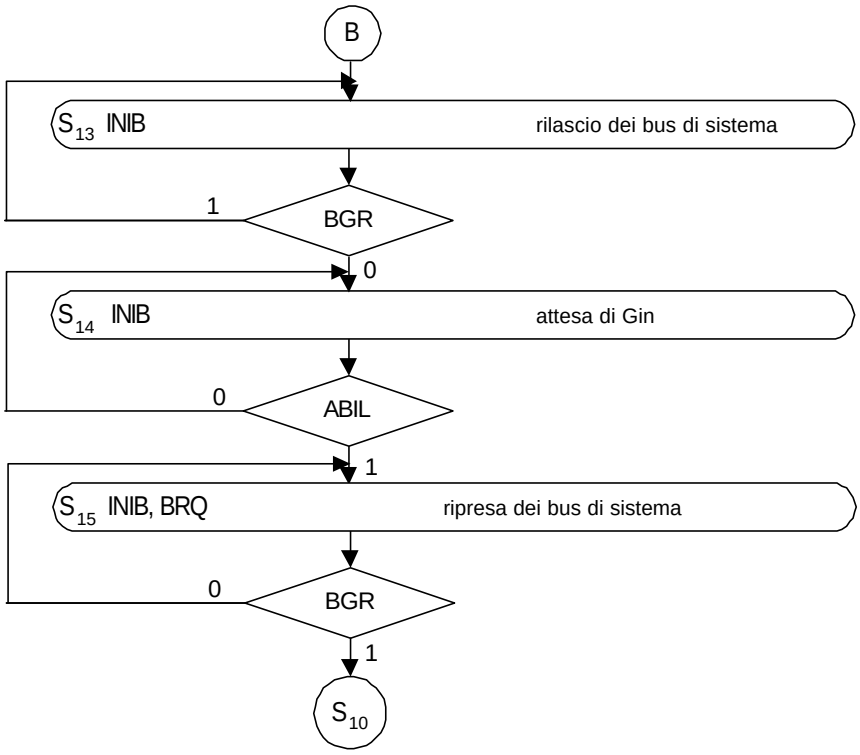


Ciclo di scrittura del dato predisposto dalla periferica

Dato trasferito: implementazione del protocollo di comunicazione a 4 fasi

Al termine del ciclo di scrittura e del protocollo, in attesa di una nuova segnalazione (REQ o EOT) da parte della periferica, viene verificata l'eventuale richiesta di sospensione dell'attività ad opera di qualche periferica a priorità più elevata

DMA-CONTR: SCO - flowchart - 3



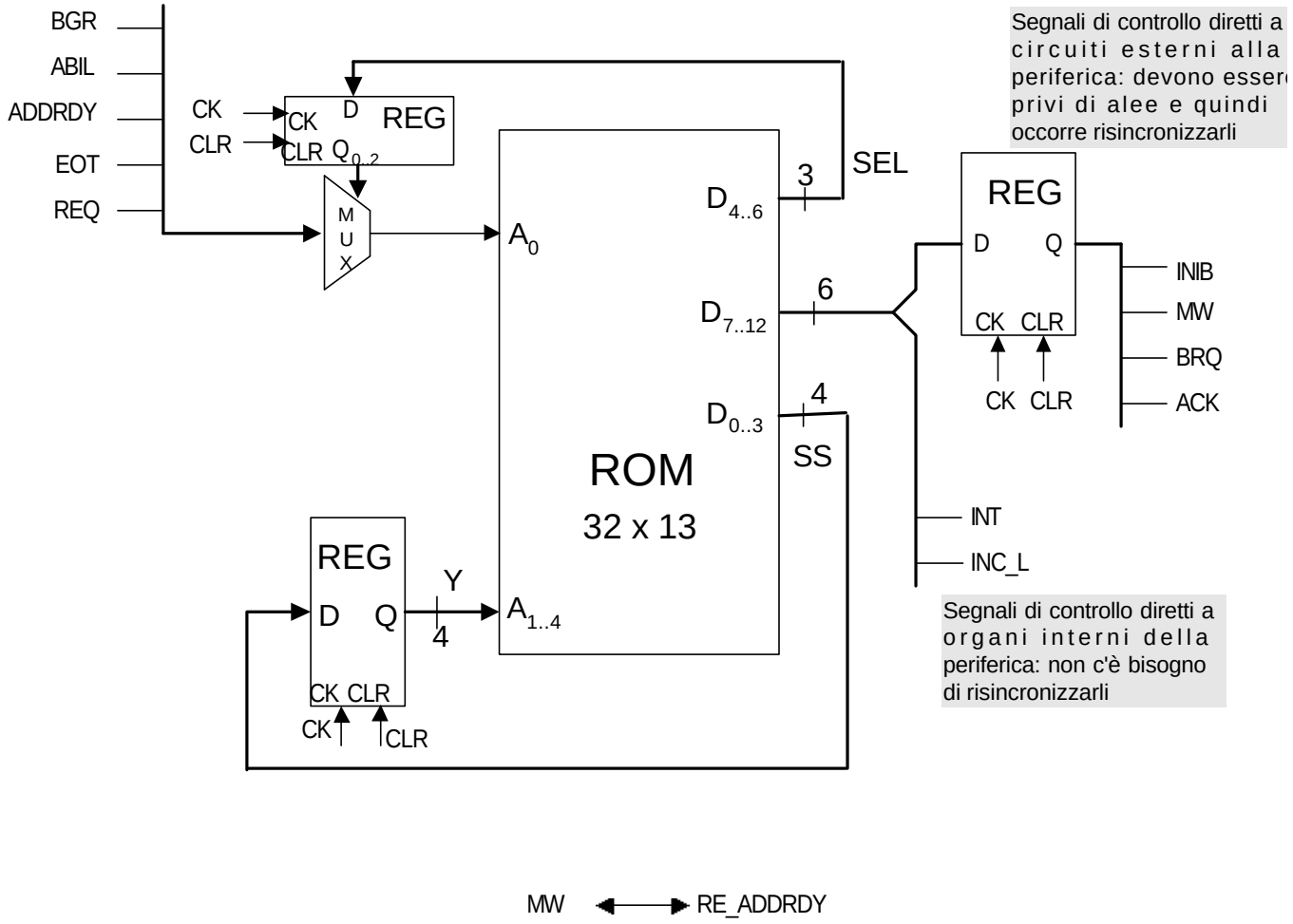
Note

Il frammento di questa sezione pone DMA-CONTR in uno stato di stand-by (S14) a seguito del rilevamento di Gin=0 dopo avere ultimato il ciclo di scrittura, che deve essere indivisibile . In stand-by lo stato della periferica è congelato, incluso il puntatore in memoria, mentre il controllo dei bus viene ceduto alla periferica a priorità più elevata che ha abbassato Gout: per questo motivo S14 è preceduto dallo stato S13 che assicura la restituzione dei bus al micro; quando Gin tornerà a 1 DMA-CONTR richiederà (S15) i bus di sistema al micro e quindi riprenderà l'attività di trasferimento in DMA dal punto in cui era stata sospesa.

Pur essendo i clock dei nodi in generali diversi, si suppone ragionevolmente che in S13 venga avvertita la commutazione 1 -> 0 di BGR (versione sincronizzata di MBG) prima della successiva commutazione 0 -> 1 ad opera della richiesta di bus effettuata dal DMA-CONTR a priorità più elevata che ha richiesto il servizio.

DMA-CONTR: SCO - struttura HW microprogrammata

Il flow-chart è stato impostato su un microlinguaggio di tipo 3: scegliendo il modello strutturale di tipo Mealy si ottiene la struttura seguente:



Note

I segnali di uscita sono stati partizionati in due blocchi: soltanto i segnali diretti all'esterno sono tamponati, tutti gli altri hanno la funzione di abilitazione dei componenti interni alla periferica e quindi non necessitano di tamponamento. Lo SCO è perciò di tipo Mealy con riguardo a questi ultimi segnali e funziona da D-Mealy con riguardo ai segnali di uscita tamponati. A questo punto va notato che i segnali non tamponati sono stati associati agli stati e non alle transizioni nel diagramma di flusso, anche se è di tipo Mealy: questo significa soltanto che i segnali in questione non dipendono dagli ingressi e quindi possono essere attribuiti agli stati per semplicità di rappresentazione .
A fronte di questa posizione, in fase di codifica del microprogramma bisognerà fare attenzione a calcolare i segnali di uscita dei due blocchi in modo da anticipare la presentazione dei segnali assoggettati a tamponamento.

DMA-CONTR: Routine PD32

I driver devono poter essere interrotti dalle altre periferiche; pertanto devono iniziare tutti con l'istruzione **seti**, ad esclusione del driver 1, associato alla periferica a massima priorità. E' la struttura hardware a daisy-chain (Gin-Gout) che fa in modo che un driver possa essere interrotto soltanto dalle periferiche a priorità più elevata.

```
driver 001h,ivn1_addr
;seti
…
movl addr_in1,Addr1
…
rti

driver 002h,ivn2_addr
seti
…
movl addr_in2,Addr2
…
rti

…

driver 007h,ivn7_addr
seti
…
movl addr_in7,Addr7
…
rti
```

RETI LOGICHE

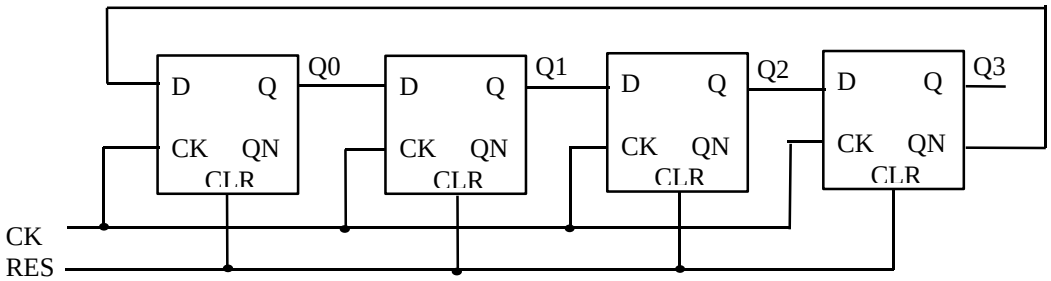
SECONDA PROVA SCRITTA DELL'APPELLO DEL 24-09-99

Studente:_____ Docente:_____

D1 Data una rappresentazione numerica in virgola fissa a 24 bit con il formato II.DDDD (I: nibble parte intera; D: nibble parte frazionaria) con notazione in complemento a 2, definire un formato in virgola mobile che includa tutti i numeri rappresentabili con la notazione originale.

D2 Data una ROM che funzioni da moltiplicatore tra nibble (gruppi di 4 bit), progettare una rete combinatoria costituita di moduli ROM definiti come sopra e addizionatori per effettuare la moltiplicazione tra byte.

D3 Analizzare il comportamento dinamico del sistema sequenziale sincrono autonomo riportato in figura (contatore ad anello).



Commentare le proprietà della sequenza degli stati scanditi a partire dall'inizializzazione e quindi generalizzarle per un anello a N flip-flop.

D4 Un processore microprogrammato può eseguire 32 diversi microprogrammi, di cui uno (fetch) comune a tutti i cicli-istruzione. I microprogrammi hanno una lunghezza massima di 200 microistruzioni. Descrivere la struttura hardware multimicroprogrammata dello SCO di tipo D-Mealy e l'allocazione dei microprogrammi nella ROM.

D5 Nella memoria PD32 a partire dall'indirizzo SAMPLES sono allocati 256 bytes che rappresentano i valori dei campioni acquisiti tramite un convertitore analogico-digitale a 8 bit. Si richiede di scrivere una routine assembler PD32 che calcoli il valor medio dei campioni e lo memorizzi all'indirizzo MEDIA.

Esercizio (2S19990924-D1)

Data una rappresentazione numerica in virgola fissa a 24 bit con il formato II.DDDD (I: nibble parte intera; D: nibble parte frazionaria) con notazione in complemento a 2, definire un formato in virgola mobile che includa tutti i numeri rappresentabili con la notazione originale.

---

1. Per poter includere tutti i numeri rappresentabili con la notazione in virgola fissa, la struttura in virgola mobile da identificare (1 bit di segno, ma **quanti bit come minimo di mantissa e di esponente?**) deve avere la capacità di rappresentare i numeri minimo e massimo, sia positivi sia negativi.

Identificazione del minimo positivo (zero escluso!):

mp: 00000000.0000000000000001 valore: 2^{-16}

Identificazione del massimo positivo:

Mp: 01111111.1111111111111111 valore: $128 - 2^{-16}$

Identificazione del massimo (assoluto) negativo:

Mn: 10000000.0000000000000000 valore: -128

Identificazione del minimo (assoluto) negativo:

mn: 11111111.1111111111111111 valore: -2^{-16}

Il bit di segno assorbe il bit in testa della rappresentazione originale, che è in complemento a 2. I restanti 23 bit della notazione originale devono poter essere tutti alloggiati nel campo della mantissa; pertanto la mantissa dovrà avere almeno 23 bit.
Per il dimensionamento dell'esponente, occorre normalizzare i numeri minimi e massimi già identificati, tenendo conto dei bit di segno e di mantissa disponibili;

mp: $+ 0.10000000000000000000000 \times 2^{-15}$

Mp: $+ 0.11111111111111111111111 \times 2^{7}$

Mn: $- 0.10000000000000000000000 \times 2^{8}$

mn: $- 0.10000000000000000000000 \times 2^{-15}$

Pertanto con il campo esponente di devono poter rappresentare, in complemento a 2, i valori numerici compresi tra –15 e 8; ne segue che occorrono 5 bit, con i quali si possono rappresentare i numeri compresi tra –16 e +15 (mentre con 4 bit si possono rappresentare i numeri compresi tra –8 e +7). In definitiva il dimensionamento minimo prevede:

S: 1 bit
M: 23 bit
E: 5 bit

Per un totale di 29 bit.

S	E: 5 bit	M: 23 bit

Ovviamente ogni altro formato con i campi E e/o M più estesi supporterà la rappresentazione di tutti i numeri specificati.

Esercizio (2S19990924-D2)

Data una ROM che funzioni da moltiplicatore tra nibble (gruppi di 4 bit), progettare una rete combinatoria costituita di moduli ROM definiti come sopra e addizionatori per effettuare la moltiplicazione tra byte.

Nella ROM andrà memorizzata la tavola di verità del moltiplicatore per operandi a 4 bit; il prodotto sarà espresso con 8 bit, quindi la tavola avrà 8 variabili di ingresso e 8 di uscita; pertanto la ROM dovrà avere almeno 8 linee di indirizzo e almeno 8 bit di dato. Indicando con $a_3a_2a_1a_0$ e $b_3b_2b_1b_0$ il moltiplicando e il moltiplicatore e con $p_7p_6p_5p_4p_3p_2p_1p_0$ il prodotto, la tavola risulta configurata così:

a_3	a_2	a_1	a_0	b_3	b_2	b_1	b_0	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
		.	.	.	.					.	.	.	.		
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0
		.	.	.	.					.	.	.	.		
0	1	1	1	1	1	1	1	0	1	1	0	1	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0
		.	.	.	.					.	.	.	.		
1	1	1	1	1	1	0	0	1	0	1	1	0	1	0	0
1	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
1	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1

Il prodotto tra due numeri a 8 bit $A=a_7a_6a_5a_4a_3a_2a_1a_0$ e $B=b_7b_6b_5b_4b_3b_2b_1b_0$ (risultato a 16 bit: $p_{15}p_{14}p_{13}p_{12}p_{11}p_{10}p_9p_8\ p_7p_6p_5p_4p_3p_2p_1p_0$) può essere effettuato operando su gruppi di 4 bit secondo lo schema di calcolo descritto di seguito; detti:

A_1 il nibble $a_7a_6a_5a_4$
A_0 il nibble $a_3a_2a_1a_0$
B_1 il nibble $b_7b_6b_5b_4$
B_0 il nibble $b_3b_2b_1b_0$

si ha:

$A = A_1 \cdot 2^4 + A_0$
$B = B_1 \cdot 2^4 + B_0$

Il prodotto $A \cdot B$ può essere riscritto come:

$$(A_1 \cdot 2^4 + A_0) \cdot (B_1 \cdot 2^4 + B_0) = A_1 \cdot B_1 \cdot 2^8 + (A_1 \cdot B_0 + A_0 \cdot B_1) \cdot 2^4 + A_0 \cdot B_0$$

o in modo equivalente secondo la struttura operativa della moltiplicazione:

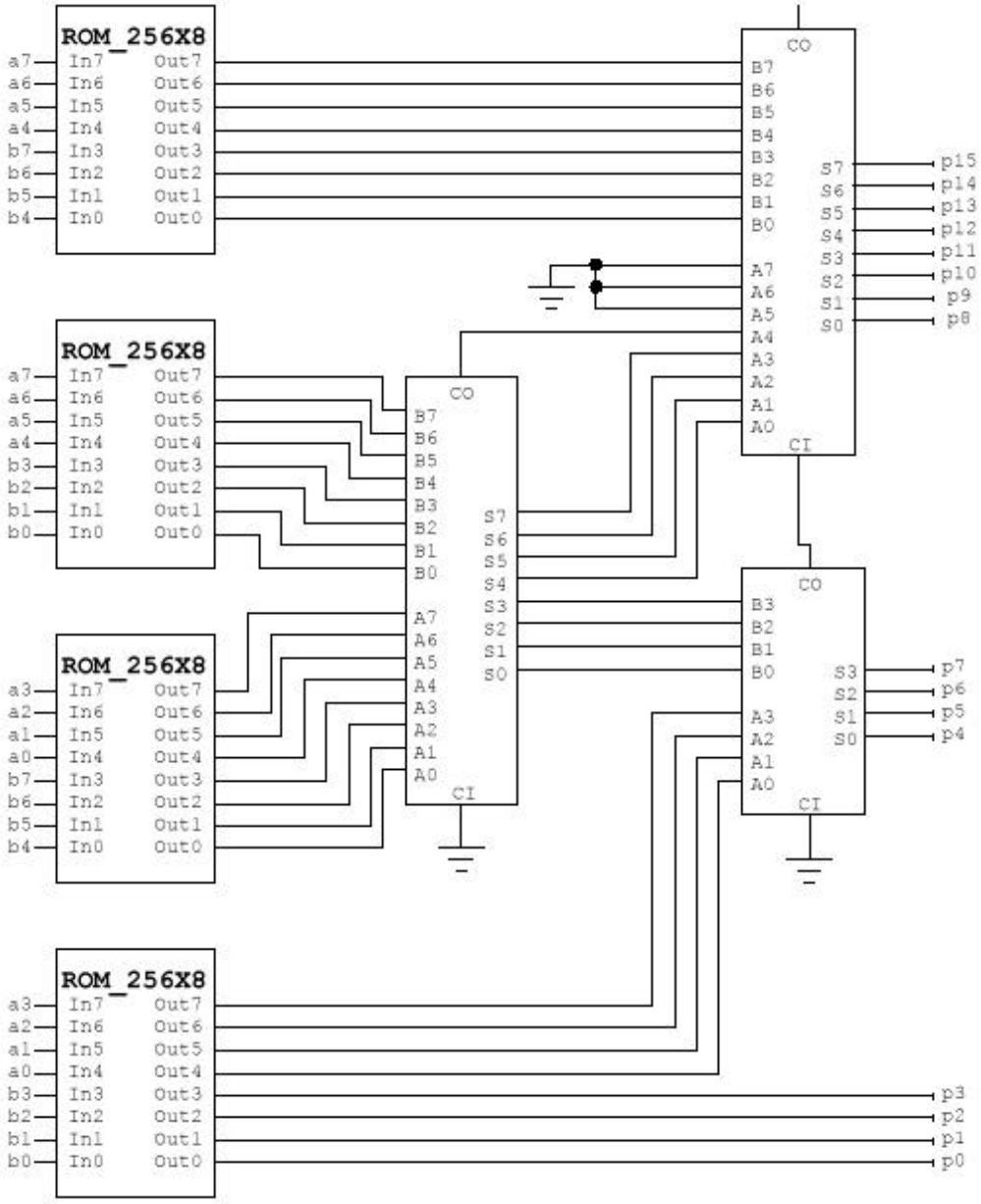
$$\begin{array}{r} A_1A_0 \\ B_1B_0 \\ \hline A_0B_0 \\ A_1B_0 \\ A_0B_1 \\ A_1B_1 \\ \hline \end{array}$$

Da entrambe queste rappresentazioni si vede che i 4 bit meno significativi del prodotto parziale A_0B_0 vengono trasferiti invariati nelle posizioni omologhe del risultato e quindi non sono soggetti a elaborazione

Questa espressione può essere implementata in modo combinatorio utilizzando i componenti:

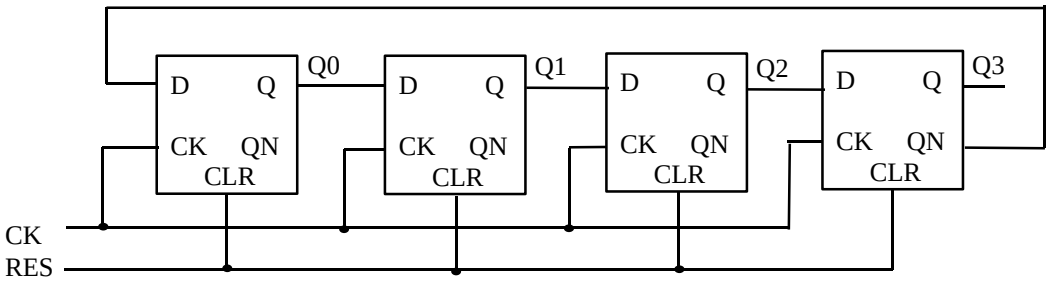
- 4 ROM per il calcolo dei prodotti parziali $A_1 \cdot B_1$, $A_1 \cdot B_0$, $A_0 \cdot B_1$ e $A_0 \cdot B_0$;
- 1 sommatore a 8 bit per il calcolo di $A_1B_0 + A_0B_1$;
- 1 sommatore a 12 bit per il calcolo di $A_1B_1A_0$ + l'uscita del sommatore a 8 bit del punto precedente.

I fattori esponenziali sono costanti, e pertanto non richiedono shifter, ma implicano semplicemente un opportuno instradamento dei prodotti parziali verso i sommatori.
Di seguito è riportato lo schema logico corrispondente. Il sommatore a 12 bit è costituito dalla cascata di due sommatori a 4 e 8 bit; il riporto del sommatore a 12 bit è sempre 0 in quanto il prodotto massimo vale $255 \times 255 < 2^{16}$, e quindi tale bit non viene riportato in uscita. Nello schema il simbolo 0 logico è sostituito dal simbolo di massa elettrica, rispondente ad una descrizione tipica degli schemi elettrici. Infine, notare che le ROM sono prive dei controlli CE, OE (così compaiono nella libreria di componenti del programma EDA utilizzato per il disegno dello schema). Se ci fossero, andrebbero ancorati staticamente al livello logico attivo; se invece il moltiplicatore fosse inserito su un bus di una struttura di calcolo, OE e/o CE andrebbero pilotati dinamicamente.


ROM_256X8
a7 In7 Out7
a6 In6 Out6
a5 In5 Out5
a4 In4 Out4
b7 In3 Out3
b6 In2 Out2
b5 In1 Out1
b4 In0 Out0
ROM_256X8
a7 In7 Out7
a6 In6 Out6
a5 In5 Out5
a4 In4 Out4
b3 In3 Out3
b2 In2 Out2
b1 In1 Out1
b0 In0 Out0
ROM_256X8
a3 In7 Out7
a2 In6 Out6
a1 In5 Out5
a0 In4 Out4
b7 In3 Out3
b6 In2 Out2
b5 In1 Out1
b4 In0 Out0
ROM_256X8
a3 In7 Out7
a2 In6 Out6
a1 In5 Out5
a0 In4 Out4
b3 In3 Out3
b2 In2 Out2
b1 In1 Out1
b0 In0 Out0
CO
B7 B6 B5 B4 B3 B2 B1 B0
A7 A6 A5 A4 A3 A2 A1 A0
CI
S7 S6 S5 S4 S3 S2 S1 S0
CO
B7 B6 B5 B4 B3 B2 B1 B0
A7 A6 A5 A4 A3 A2 A1 A0
CI
S7 S6 S5 S4 S3 S2 S1 S0
CO
B3 B2 B1 B0
A3 A2 A1 A0
CI
S3 S2 S1 S0
p15 p14 p13 p12 p11 p10 p9 p8
p7 p6 p5 p4
p3 p2 p1 p0

Esercizio (2S19990924-D3)

Analizzare il comportamento dinamico del sistema sequenziale sincrono autonomo riportato in figura (contatore ad anello).



Commentare le proprietà della sequenza degli stati scanditi a partire dall'inizializzazione e quindi generalizzarle per un anello a N flip-flop.

Successivamente all'applicazione ed al rilascio del segnale di reset, le uscite Q0..Q3 si predisporranno a 0000. Gli stati successivi del sistema sequenziale compariranno codificati ad ogni ciclo di clock sulle linee Q0..Q3 di uscita dei quattro flip-flop, che costituiscono il registro di stato del sistema sequenziale sincrono. La scansione è la seguente:

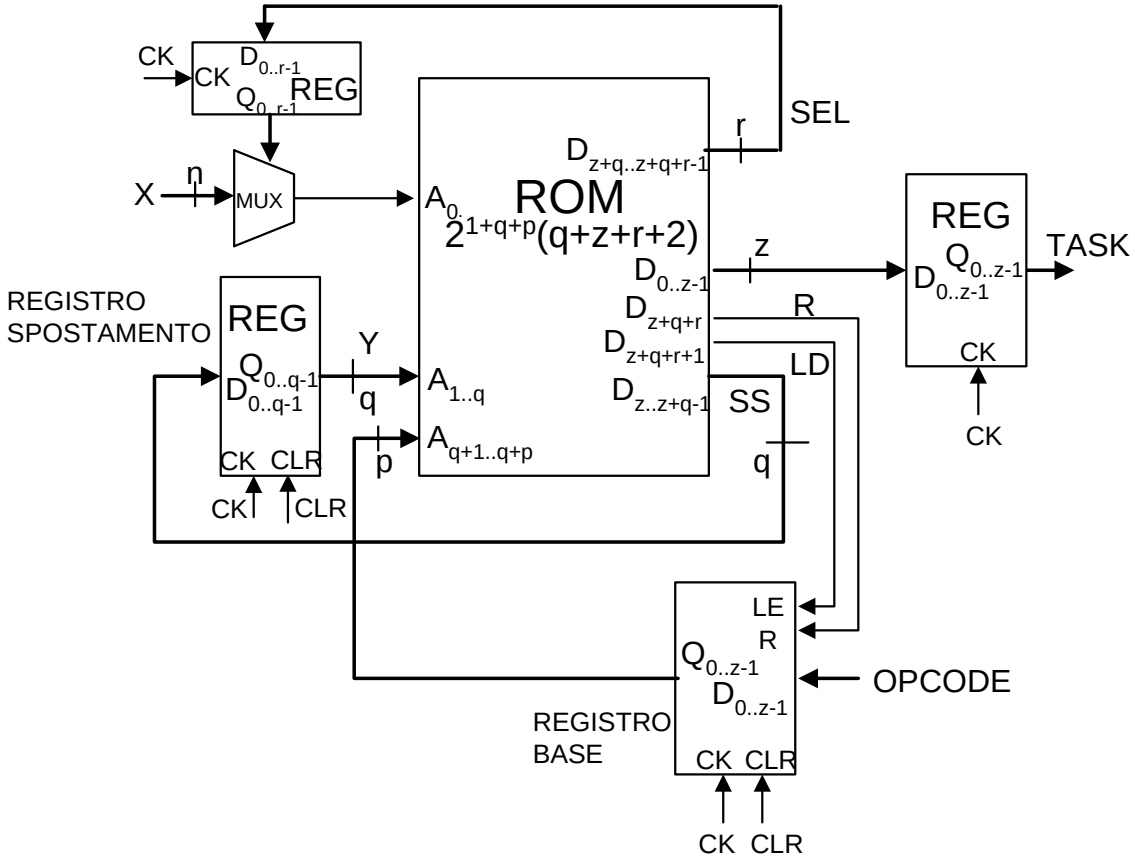
$0000 \rightarrow 1000 \rightarrow 1100 \rightarrow 1110 \rightarrow 1111 \rightarrow 0111 \rightarrow 0011 \rightarrow 0001 \rightarrow 0000$

La sequenza è ciclica con periodo 8; inoltre, i codici degli stati vengono generati mediante l'attraversamento dell'intera catena di flip-flop prima con un 1 e poi con uno 0; ciò significa che il periodo della sequenza è pari al doppio della lunghezza della catena (in questo caso: 2x4). Pertanto, nel caso generale una catena di N flip-flop disposti come nella figura del testo (contatore ad anello) descriverà una sequenza di 2xN stati (conteggio lineare). Un'ulteriore proprietà notevole della sequenza è che due stati consecutivi qualsiansi sono adiacenti.

Esercizio (2S19990924-D4)

Un processore microprogrammato può eseguire 32 diversi microprogrammi, di cui uno (fetch) comune a tutti i cicli-istruzione. I microprogrammi, descritti con il microlinguaggio di tipo 3, hanno una lunghezza massima di 200 microistruzioni. Descrivere la struttura hardware multimicroprogrammata dello SCO di tipo D-Mealy e l'allocazione dei microprogrammi nella ROM.

La struttura microprogrammata di tipo D-Mealy che supporta il microlinguaggio di tipo 3 si ottiene da quella D-Mealy con mascheramento con k = 1. La struttura particolarizzata è riportata nella figura seguente:



Il dimensionamento dei componenti deriva dalle specifiche:
p = 5; q = 8. Le altre grandezze z, n e r dipendono dallo SCA (cioè dal numero delle variabili di controllo e di condizione).
La struttura della memoria è la seguente:

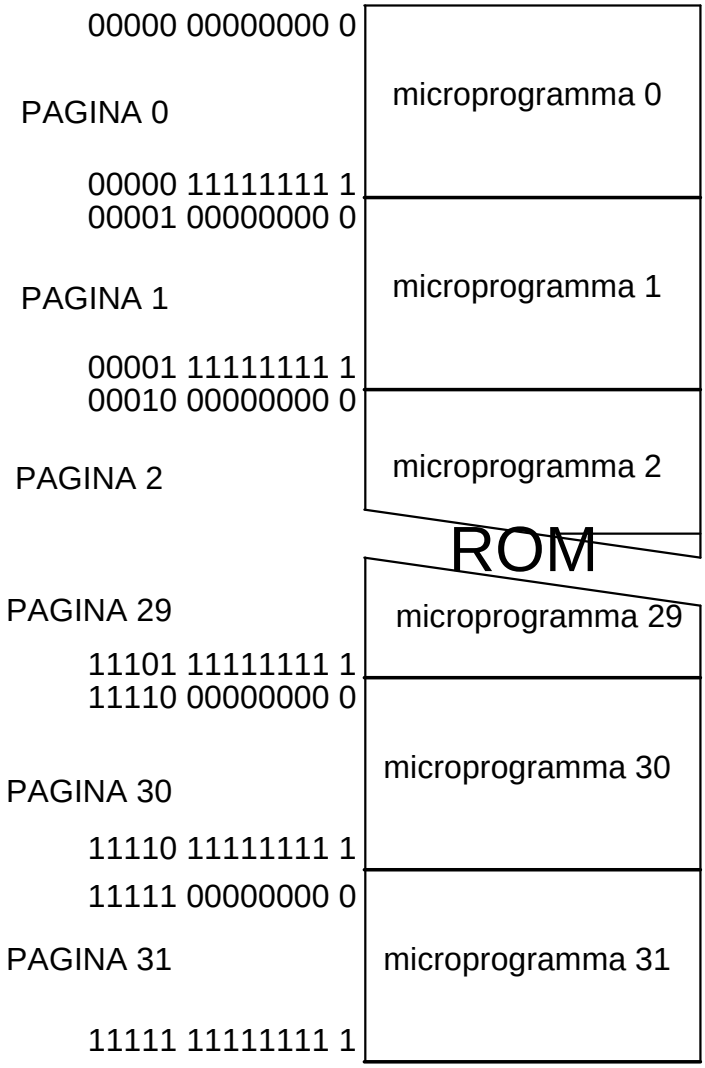
Esempio:
p=5; q=8; k=1

Bit di indirizzo della ROM:
OC_4 .. OC_0 Y_7 .. Y_0 X

Estensione ROM: 2^{5+8+1} = 16K righe
Numero pagine: 2^5 = 32
Estensione pagina: 2^{8+1} = 512 righe

La struttura può accomodare fino a 32 microprogrammi, ognuno di 256 stati al massimo e con un test di una variabile in ogni stato, corrispondenti a una lunghezza massima di 512 righe di ROM.

La larghezza e quindi la capacità della ROM dipendono dalle quantità z e r.



Il microprogramma comune a tutti i cicli-istruzione va posizionato nella pagina 0.

Esercizio (2S19990924-D5)

Nella memoria PD32 a partire dall'indirizzo SAMPLES sono allocati 256 bytes che rappresentano i valori dei campioni acquisiti tramite un convertitore analogico-digitale a 8 bit. Si richiede di scrivere una routine assembler PD32 che calcoli il valor medio dei campioni e lo memorizzi all'indirizzo MEDIA.

```
        org 400h                    ;inizio programma

; *************************
; COSTANTI
; *************************

        SAMPLES     equ 1000h   ;inizio area dati: 256 byte
        MEDIA       equ 1100h   ;SAMPLES + 256
        STACK       equ 2800h   ;inizio area di stack limitato a 2800h
                                ;per consentire la simulazione

; *************************
; CODICE
; *************************
        code                        ;inizio istruzioni

main:
        movl #STACK,r7              ;inizializza R7 quale SP
        seti                        ;abilita interruzioni (SP è stato
                                    ;inizializzato)

        xorl r0,r0                  ;r0 <- 0 inizializza l'accumulatore

;ciclo operativo
        movl #SAMPLES,r1            ;r1 puntatore all'area dei campioni
        xorl r2,r2                  ;r2 word di appoggio per i byte letti
acc:
        mvlb (r1)+,r2               ;carica il byte-campione in r2 senza ;estensione di
                                    segno
                                    ;e (post-)incrementa r1 (di 1)
        addw r2,r0                  ;incrementa l'accumulatore sulla
                                    ;larghezza word
        cmpl #SAMPLES+256,r1        ;confronto fine ciclo (256 byte)
        jnz acc

        lsrw #8,r0                  ;dividi l'accumulatore per 256
        movb r0,MEDIA               ;scrivi la media aritmetica (1 byte)
                                    ;in MEDIA

        halt                        ;termine elaborazione (serve per
                                    ;la simulazione)
        end                         ;fine programma
```

```
;Nella memoria PD32 a partire dall'indirizzo SAMPLES sono allocati
;256 bytes che rappresentano i valori dei campioni acquisiti
;tramite un convertitore analogico-digitale a 8 bit.
;Si richiede di scrivere una routine assembler PD32 che calcoli
;il valor medio dei campioni e lo memorizzi all'indirizzo MEDIA.



  org 400h    ;inizio programma


; *************************
; COSTANTI
; *************************


  SAMPLES equ 1000h   ;inizio area dati dall'ADC: 256 byte
  MEDIA equ 1100h ;SAMPLES + 256

  STACK   equ 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione


; *************************
; CODICE
; *************************


  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)


  xorl r0,r0    ;r0 <- 0 inizializza l'accumulatore

;ciclo operativo
  movl #SAMPLES,r1  ;r1 puntatore all'area dei campioni
  xorl r2,r2    ;r2 word di appoggio per i byte letti
acc:
  mvlb (r1)+,r2   ;carica il byte-campione in r2 senza estensione di segno
        ;e (post-)incrementa r1
  addw r2,r0    ;incrementa l'accumulatore sulla larghezza word
  cmpl #SAMPLES+256,r1  ;confronto fine ciclo (256 byte)
  jnz acc

  lsrw #8,r0    ;dividi l'accumulatore per 256
  movb r0,MEDIA   ;scrivi la media aritmetica (1 byte) in MEDIA


  halt      ;termine elaborazione (serve per la simulazione)

  end     ;fine programma
```

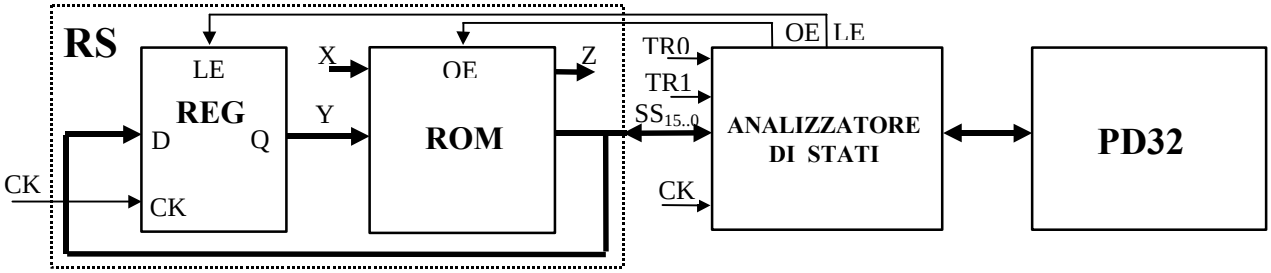
RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 19-10-1999

STUDENTE:_____ DOCENTE:_____

Specifiche funzionali:

Si vuole realizzare un dispositivo (analizzatore di stati) che faciliti il monitoraggio delle sequenze degli stati scandite da una rete sequenziale (RS) sincrona di tipo Mealy, costituita da una ROM e da un registro di stato dotato di abilitazione al caricamento. L'analizzatore da progettare e la RS data sono accoppiati come riportato in figura, tramite i segnali:
CK: clock comune ai due sistemi;
OE: Output Enable della ROM della RS;
SS: bus a 16 bit (può essere letto/scritto dall'analizzatore) dello stato successivo della RS;
LE: abilitazione al caricamento del registro di stato della RS.



L'analizzatore di stati viene dotato anche di una interfaccia bidirezionale con il processore PD32, utilizzata sia per programmare l'analizzatore con uno o più comandi di inizializzazione del ciclo di monitoraggio, sia per acquisire i risultati al termine del ciclo di monitoraggio stesso. I comandi vengono inviati in successione dal micro su una stessa porta a 19 bit (indirizzo 5Ah) dell'analizzatore, tutti con il formato riportato nella prima tabella, e determinano le azioni descritte nella seconda tabella.

OC2 18	OC1 17	OC0 16	15 DATO 0

OC2	OC1	OC0	Azione
0	0	0	Inizia il monitoraggio immediatamente
0	0	1	-
0	1	0	Inizializza il registro di stato di RS con DATO (stato iniziale del monitoraggio)
0	1	1	Inizia il monitoraggio sulla transizione positiva del segnale esterno TR0
1	0	0	Termina il monitoraggio immediatamente
1	0	1	Termina il monitoraggio dopo l'esecuzione di un numero di stati pari a DATO
1	1	0	Termina il monitoraggio al raggiungimento dello stato codificato DATO (stato finale)
1	1	1	Termina il monitoraggio sulla transizione negativa del segnale esterno TR1

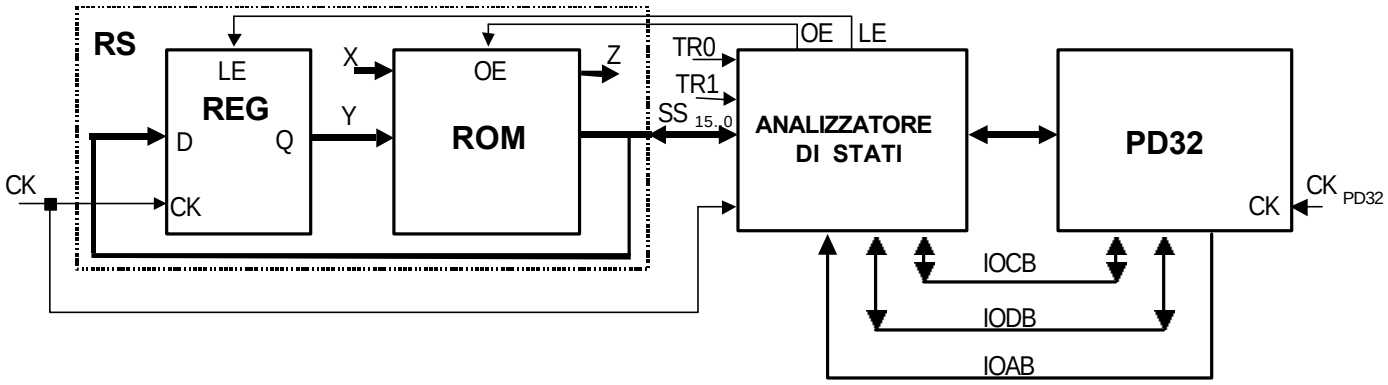
Ogni ciclo di monitoraggio della RS viene inizializzato dal micro mediante l'invio di una sequenza appropriata di comandi (ad esempio $OC_{2..0} \leftarrow$ 010 - 101 - 000, oppure: 010 - 111 - 000, oppure: 110 - 011, ecc.).
Durante il ciclo di monitoraggio l'analizzatore controlla la scansione degli stati della RS tramite LE: ad ogni passo del ciclo il vettore SS viene sia caricato nel registro di stato di RS, sia memorizzato in una RAM da 256 word, che alla fine del ciclo conterrà la sequenza degli ultimi (fino a 256) stati visitati dalla RS; si assuma di poter eseguire un ciclo di scrittura in RAM in tre periodi di clock (entro i quali lo stato della RS dovrà essere bloccato).
Al termine del monitoraggio, la periferica lancia una interruzione al micro, cui trasferisce un pacchetto di N+1 word, secondo il formato riportato nella tabella seguente, dove i pedici definiscono la sequenza temporale degli stati catturati. Si noti che può essere N < 256, nel qual caso vanno trasferiti al micro solo i codici effettivamente memorizzati nel ciclo di monitoraggio appena concluso.

N	S_0	S_1	S_2	...	S_{N-1}

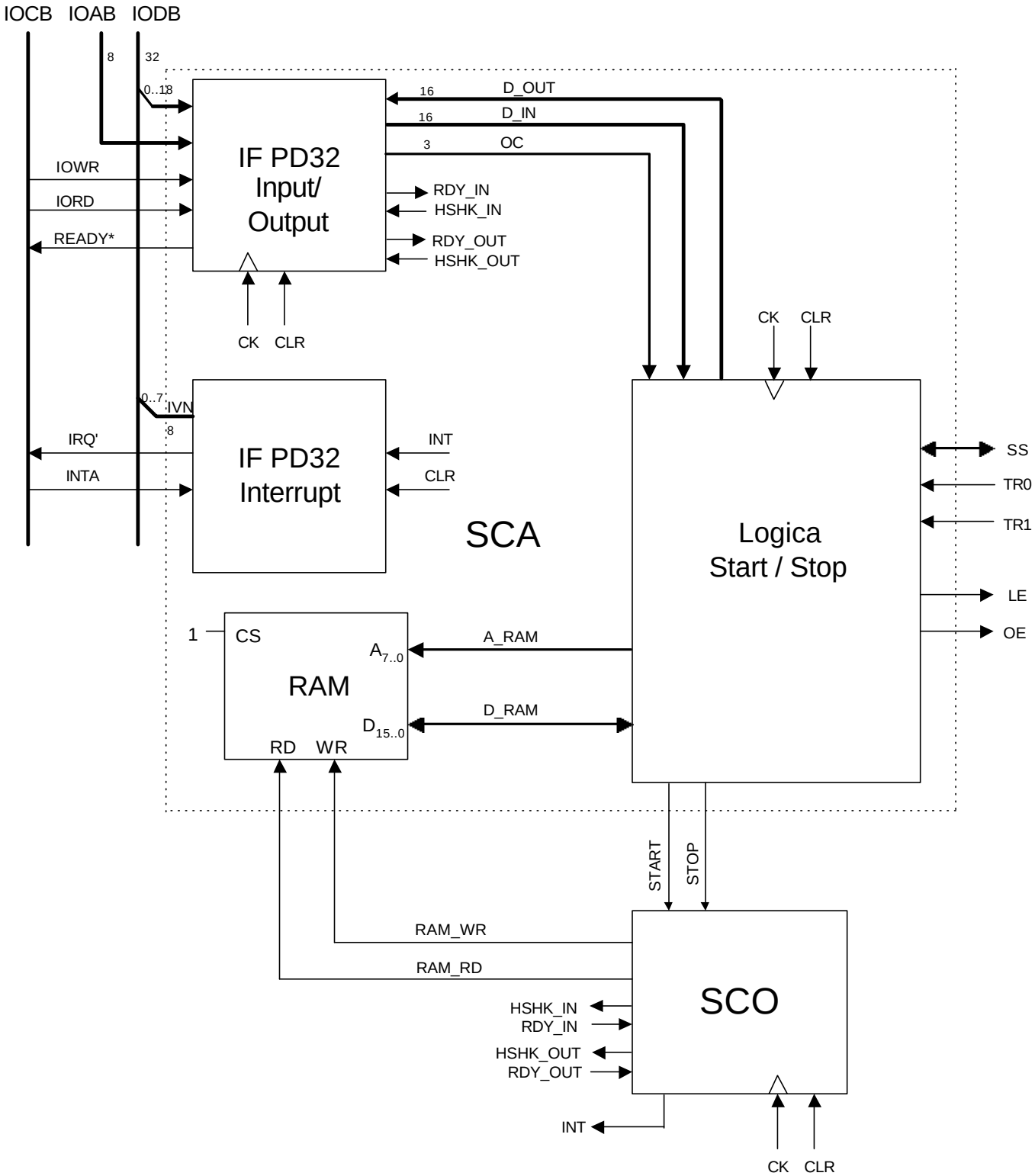
Si richiede:

- lo schema a blocchi funzionali dell'analizzatore di stati;
- l'interfaccia con il PD32;
- il diagramma di temporizzazione dell'interfaccia con la RS;
- lo schema logico dettagliato dello SCA e dello SCO dell'analizzatore;
- il rapporto tra la frequenza di CK e quella di aggiornamento del registro di stato della RS durante il monitoraggio.

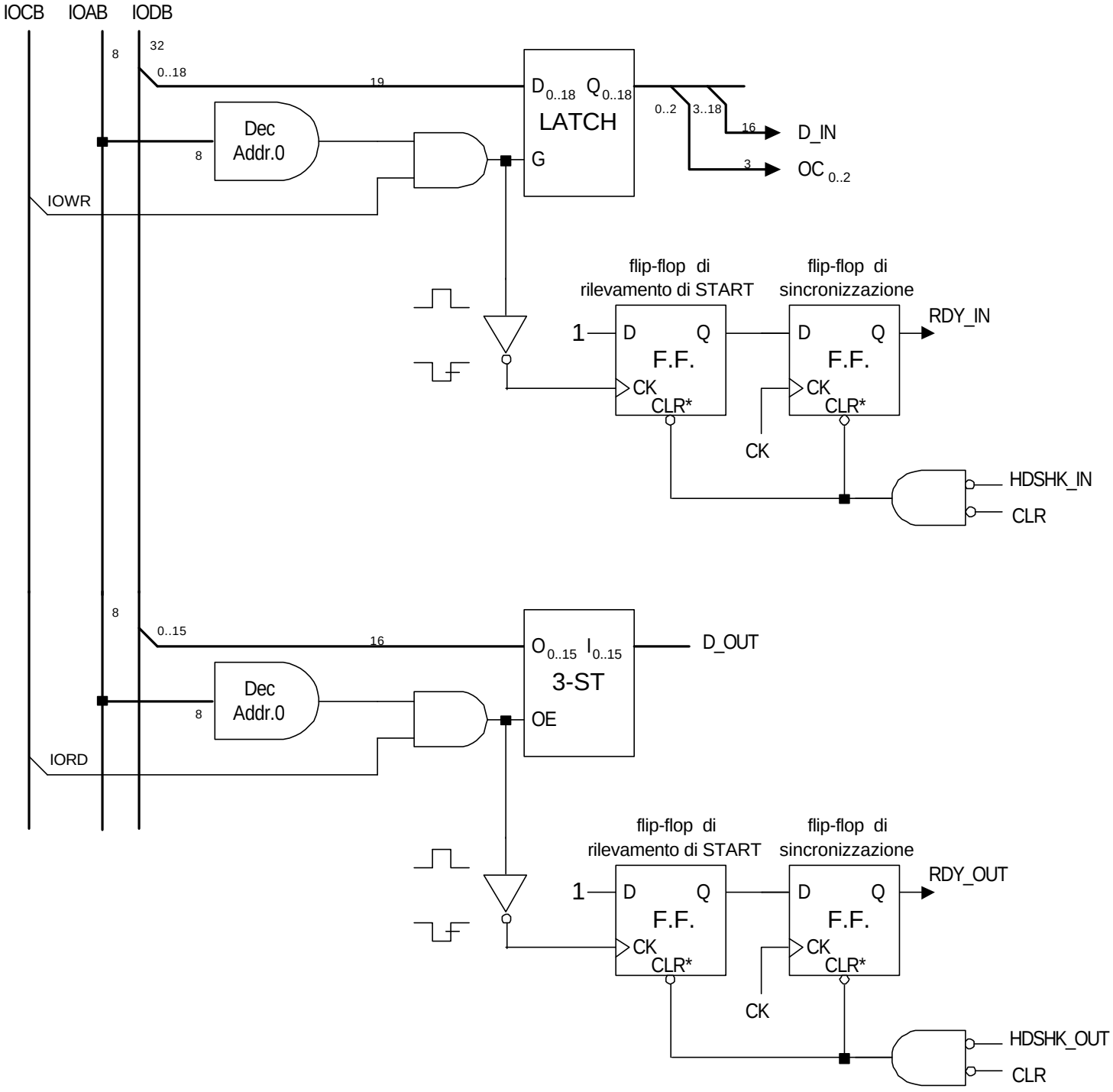
Analizzatore di stati: sistema esterno



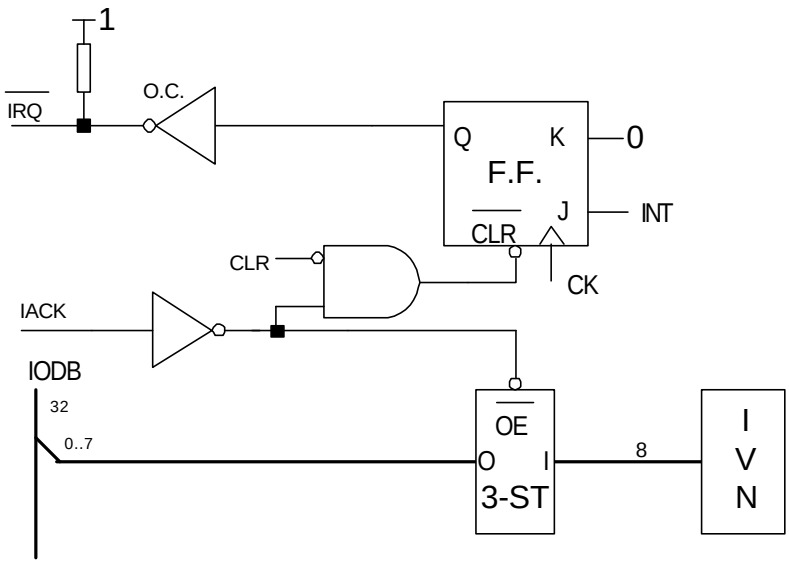
Analizzatore di stati: schema a blocchi



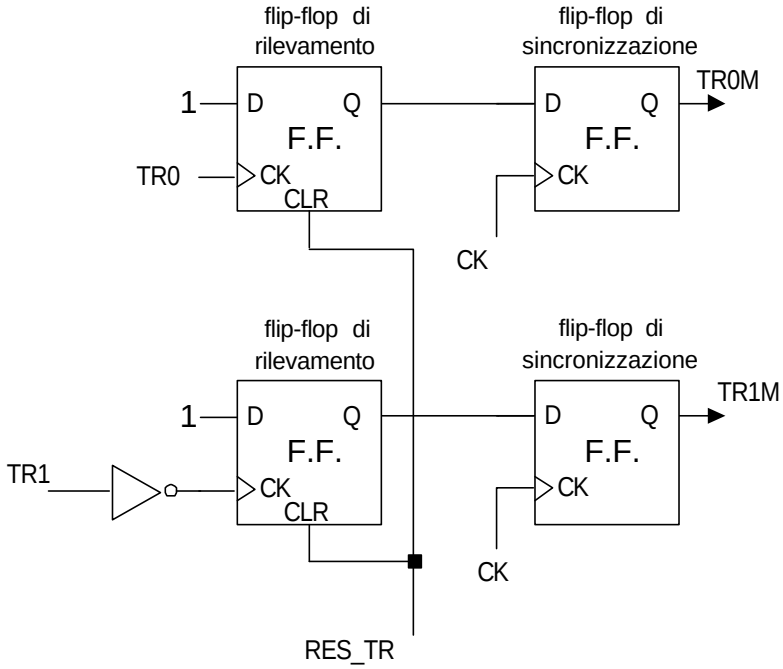
Analizzatore di stati: IF PD32 - Input/Output



Analizzatore di stati: IF PD32 - interrupt

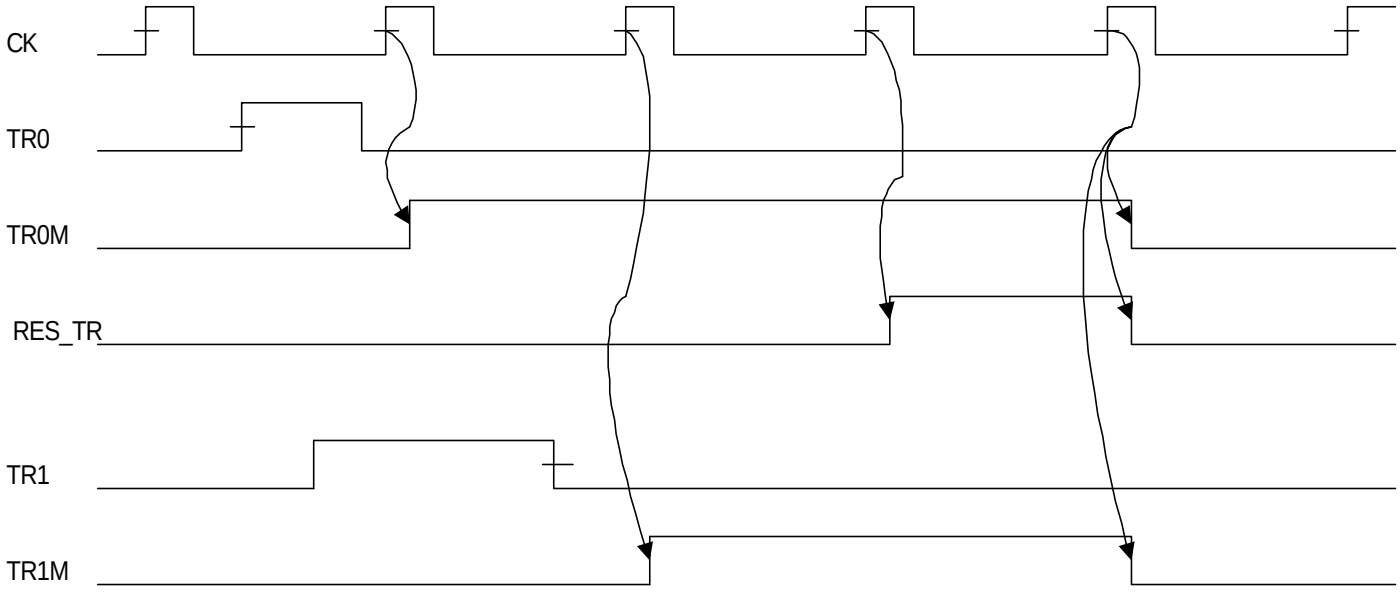


Analizzatore di stati:
logica start/stop - trigger esterni

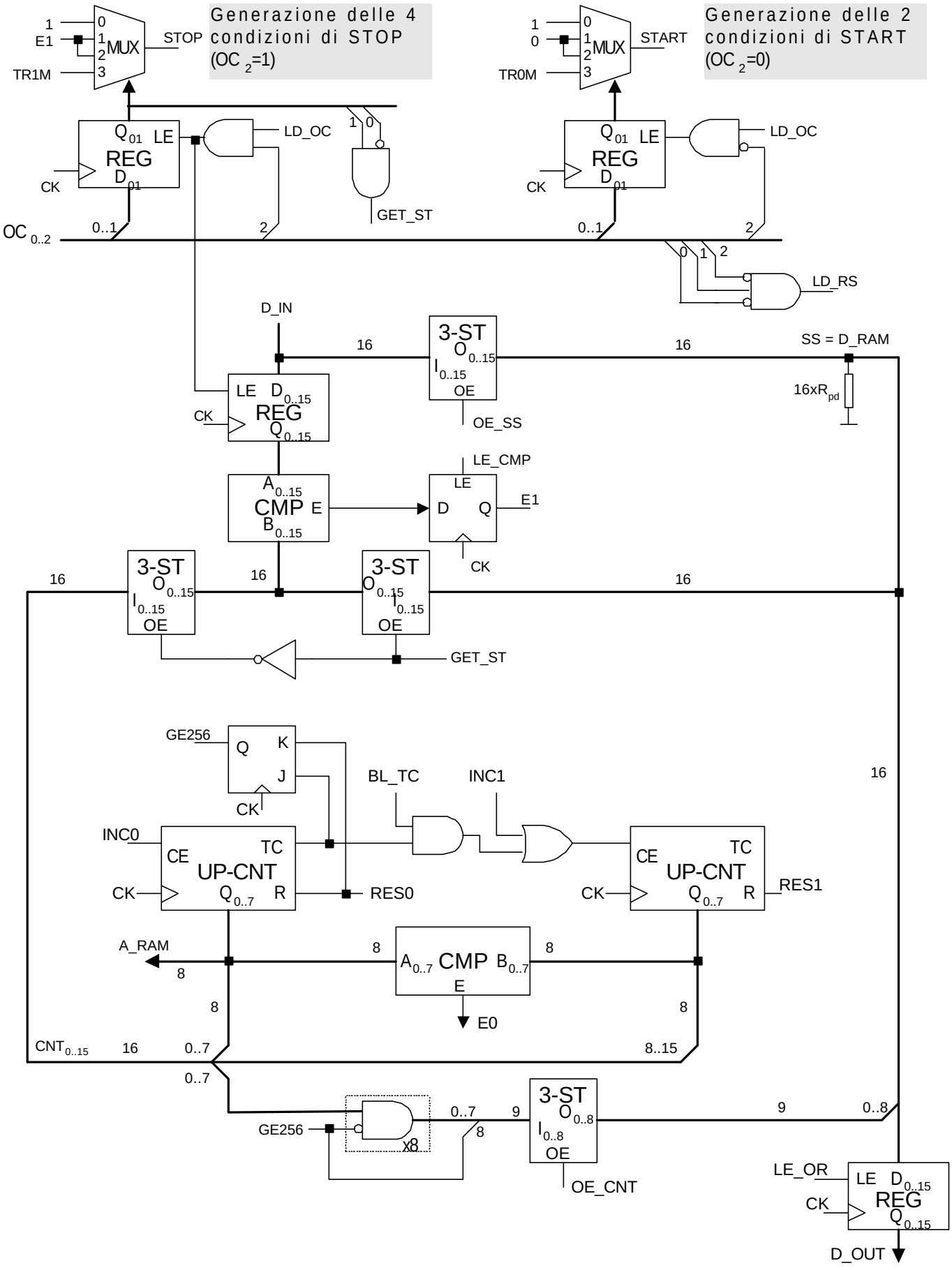


Tutti i flip-flop sono di tipo "positive edge triggered" (sensibili al fronte positivo)

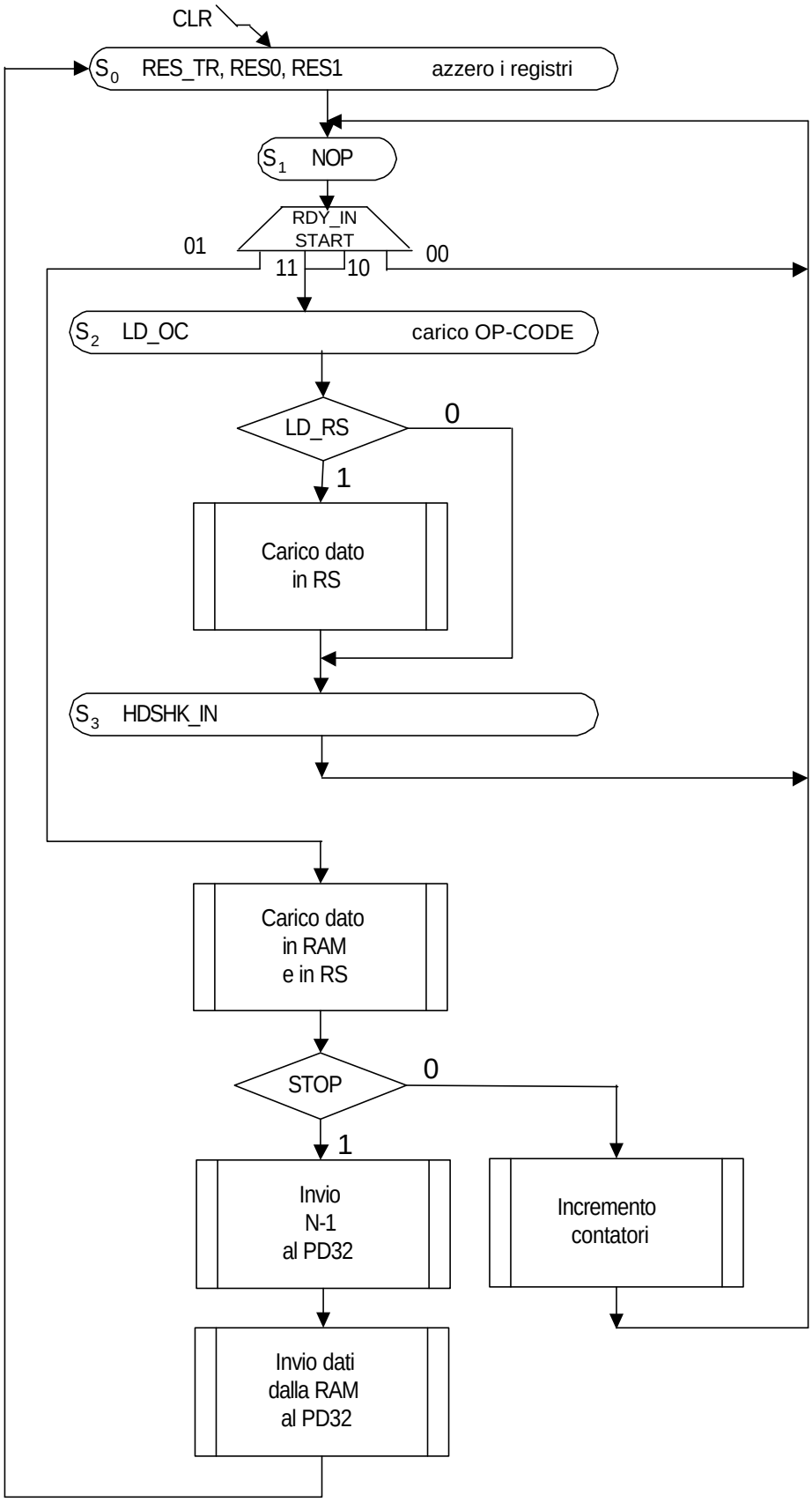
Diagramma di temporizzazione



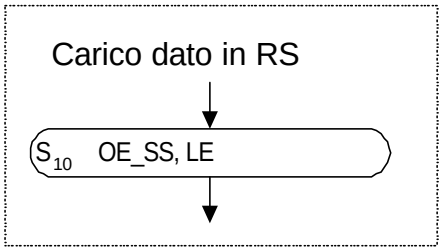
Analizzatore di stati: logica start/stop

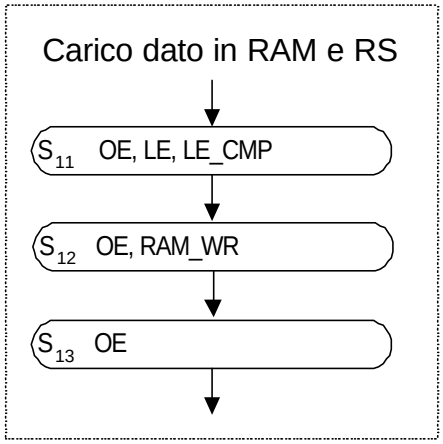


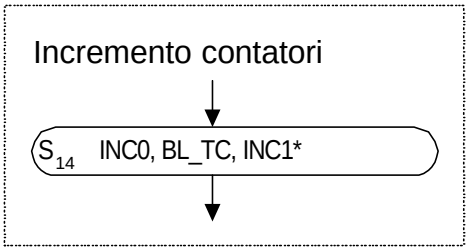
Analizzatore di stati: SCO - flowchart

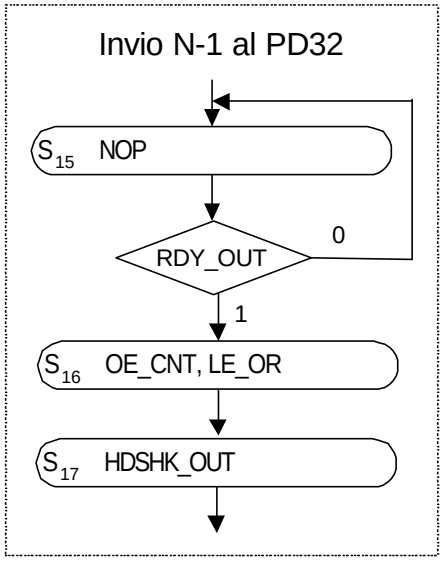


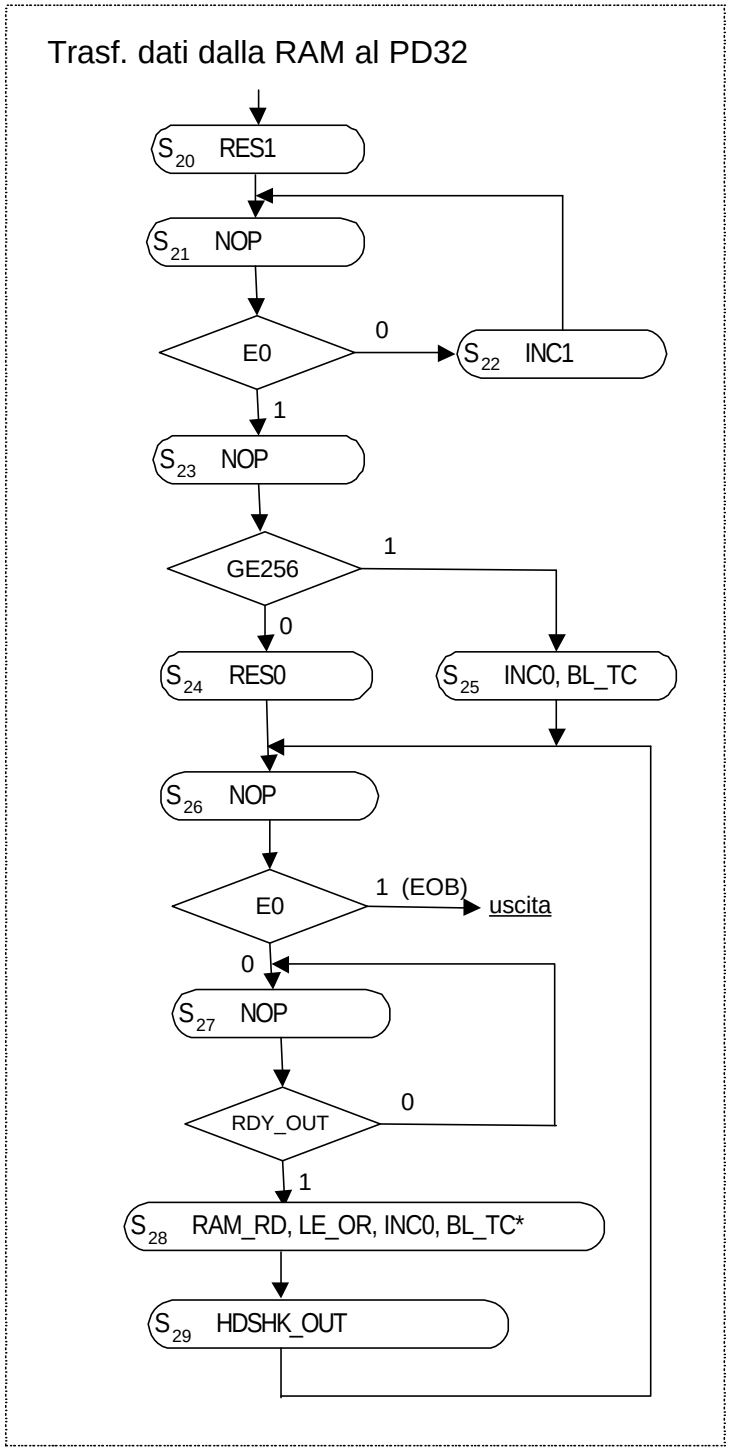
Analizzatore di stati: SCO - routine del flowchart





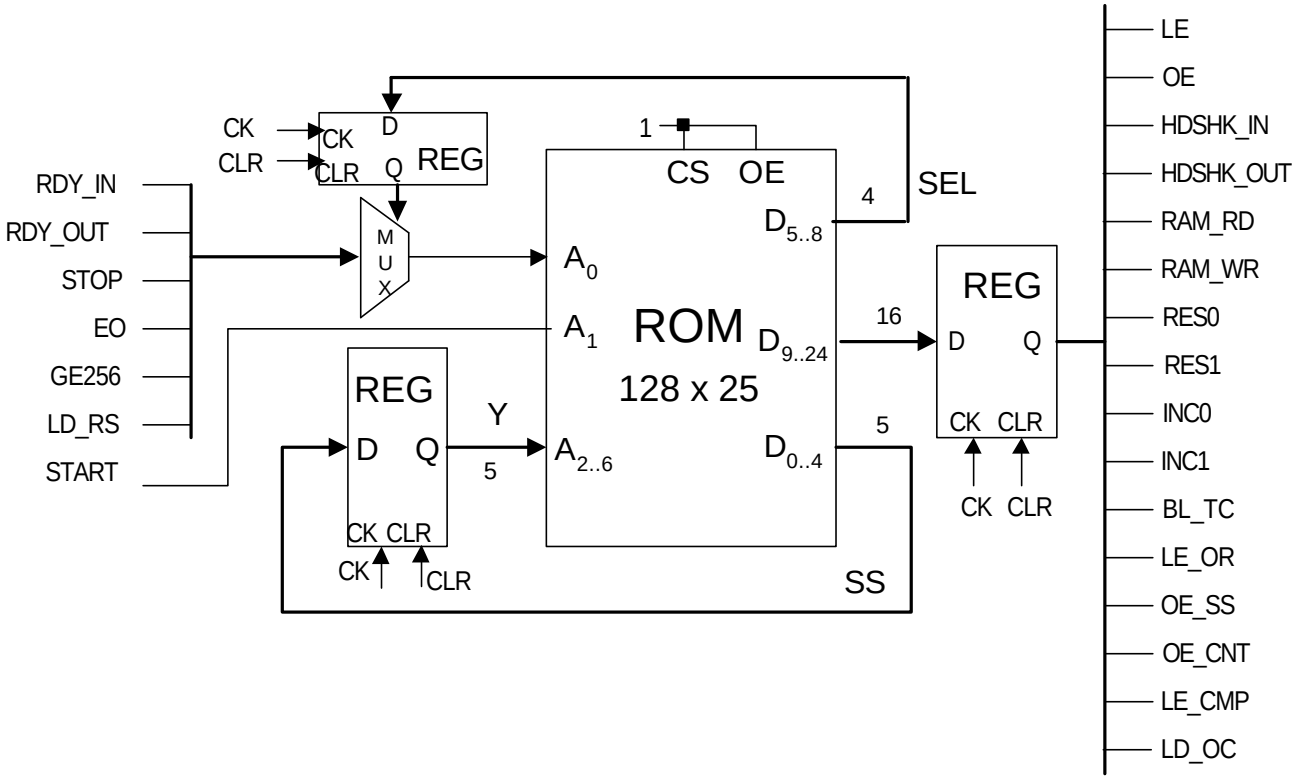






Analizzatore di stati:
SCO - struttura HW microprogrammata

Il flow-chart definito per lo SCO può essere ben supportato da una struttura microprogrammata di tipo D-Mealy, con la selezione di due variabili di ingresso:



I due mux sugli ingressi devono selezionare una sola coppia significativa per il flow-chart: (RDY_IN, START); in tutti gli altri stati viene effettuato il test su una variabile al più. Pertanto si può utilizzare un solo mux 8:1 per raccogliere gli ingressi ad esclusione di uno dei due segnali della coppia, per esempio START, che può essere collegato direttamente a un bit di indirizzo della ROM.

NOTE

- Calcolo della frequenza di aggiornamento del registro RS:
  il ciclo di aggiornamento è composto dalla sequenza dei 5 stati: S_{11}, S_{12}, S_{13}, S_{14}, S_1, a meno di ricezione di un comando, nel qual caso da S_1 il controllo passa in S_2; pertanto la frequenza massima di aggiornamento del registro RS è 1/5 della frequenza di CK (un ciclo di clock per stato).

- Le temporizzazioni sono desumibili direttamente dal flow-chart del microprogramma dello SCO (un ciclo di clock per stato).

- Si presuppone che la RAM abbia un tempo di accesso in lettura minore del periodo di CK: $t_A < T_{CK}$; con questa posizione la lettura della RAM (routine di trasferimento dei dati acquisiti dalla RAM al micro) viene effettuata in un solo ciclo di CK. Va notato che il ciclo di scrittura viene effettuato in tre cicli di CK: l'intervallo corrispondente a tre cicli di CK è sufficiente a scrivere un dato in memoria, in quanto il tempo di ciclo di scrittura della RAM, che è simile al tempo di ciclo di lettura e allo stesso tempo di accesso in lettura, è stato ipotizzato minore di un periodo di CK; l'uso di tre cicli di CK è anche necessario per rispettare i tempi di set-up e hold del dato da scrivere.

- Il contatore a 16 bit degli stati è stato separato in due contatori a 8 bit disposti in cascata e collegati con una logica interposta tra TC del primo e CE del secondo: gli 8 bit meno pesanti servono da indirizzi alla RAM interna (256 x 16 bit), mentre la logica tra TC e CE serve a collegare o separare i due contatori, azionabili quindi anche in modo indipendente. Nella routine di trasferimento dati dalla RAM al micro lo stato del primo contatore viene subito copiato nel secondo, mediante l'azzeramento (RES1) e l'incremento (INC1) continuo del secondo contatore, fino alla coincidenza (E1=1) dei due conteggi; a questo punto il controllo si diversifica in due percorsi:
  1) se il numero degli stati memorizzati è minore di 256, occorre trasferire al micro i dati memorizzati nella RAM a partire dall'indirizzo 0: pertanto il primo contatore viene azzerato e quindi incrementato ad ogni dato letto dalla RAM interna, fino a che il conteggio raggiunge lo stato (bloccato) del secondo contatore;
  2) se il numero degli stati memorizzati è maggiore o uguale di 256, occorre trasferire al micro i dati memorizzati nella RAM a partire dall'indirizzo corrente del primo contatore: pertanto il primo contatore viene incrementato ad ogni dato letto dalla RAM interna, fino a che il conteggio raggiunge lo stato (bloccato) del secondo contatore. Va notato che durante questo processo il secondo contatore deve mantenere il proprio stato e pertanto il TC del primo contatore è stato mascherato (BL_TC=0). A questo punto saranno stati trasferiti al micro i codici degli ultimi 256 stati visitati da RS, come richiesto dalla specifica.

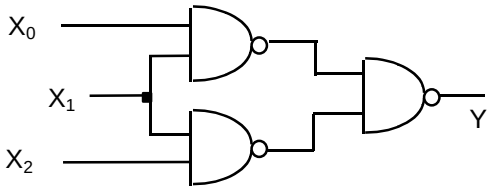
RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 19-10-99

Studente:_____ Docente:_____

D1 Illustrare i passi per arricchire il codice BCD della capacità di autocorrezione di un singolo errore.

D2 Implementare la rete combinatoria in figura con un multiplexer.



D3 Definire la struttura di un registro caricabile in modalità sincrona con dati a larghezza 8, 16 o 32 bit.

D4 Organizzare un banco di RAM di capacità 256 Mbytes, indirizzabili dal processore PD32 a partire dall'indirizzo F0000000h, utilizzando moduli RAM da 16 MBytes.

D5 Una periferica di un processore PD32 campiona periodicamente una linea analogica mediante un comparatore analogico (convertitore AD a 1 bit). I bit rilevati vengono inviati singolarmente mediante interruzione al processore, che provvede a memorizzarli consecutivamente nella RAM, 32 bit di conversione per longword, a partire dalla locazione WAVE. La produzione dei dati viene avviata da un comando di inizio operazione emesso dal micro verso la periferica e viene arrestata con un comando di fine operazione al raggiungimento di 32 K campioni (bit).
Scrivere il programma di acquisizione in assembler PD32.

Esercizio (2S19991019-D1)

Illustrare i passi per arricchire il codice BCD della capacità di autocorrezione di un singolo errore.

Questo problema è un caso particolare dell'esercizio precedente, con l'unica differenza che in questo caso il codice irridondante non è esaustivo (occupa 10 configurazioni sulle 16 disponibili con 4 bit).

Il codice irridondante ha n = 4. Il codice richiesto sarà di tipo Hamming con h = 3. Come è noto, il numero dei bit di controllo richiesti k è legato a quello n = 4 dalla relazione:

$$4 \leq 2^k - k - 1$$

da cui (per tentativi: k = 1, 2,…) si trova k = 3. Ne segue che il codice ridondante sarà composto da m = 4 + 3 = 7 bit. La parola del codice avrà la forma seguente:

$c_1 c_2\ a_3\ c_4\ a_5\ a_6\ a_7$ cioè: $c_{001}\ c_{010}\ a_{011}\ c_{100}\ a_{101}\ a_{110}\ a_{111}$

Codifica

$c_1 = \sum (a_3, a_5, a_7)$ mod2

$c_2 = \sum (a_3, a_6, a_7)$ mod2

$c_4 = \sum (a_5, a_6, a_7)$ mod2

Parole del codice ridondante

c_1	c_2	a_3	c_4	a_5	a_6	a_7	
0	0	0	0	0	0	0	w_0
1	1	0	1	0	0	1	w_1
0	1	0	1	0	1	0	w_2
1	0	0	0	0	1	1	w_3
1	0	0	1	1	0	0	w_4
0	1	0	0	1	0	1	w_5
1	1	0	0	1	1	0	w_6
0	0	0	1	1	1	1	w_7
1	1	1	0	0	0	0	w_8
0	0	1	1	0	0	1	w_9

Decodifica

Esempio 1

Sia ricevuta la parola: **1001100**; allora:

$Nc_1 = \sum (c_1, a_3, a_5, a_7)$ mod2 = $\sum (1, 0, 1, 0)$ mod2 = 0

$Nc_2 = \sum (c_2, a_3, a_6, a_7)$ mod2 = $\sum (0, 0, 0, 0)$ mod2 = 0

$Nc_3 = \sum (c_4, a_5, a_6, a_7) \bmod 2 = \sum (1, 1, 0, 0) \bmod 2 = 0$

$Nc = 000 = 0_{10} \Rightarrow$ parola corretta (w_4)

~~1~~ ~~0~~ 0 ~~1~~ 1 0 0 → parola decodificata: 0100
c_1 c_2 c_4

Esempio 2

Sia ricevuta la parola: **0001100**; allora:

$Nc_1 = \sum (c_1, a_3, a_5, a_7) \bmod 2 = \sum (0, 0, 1, 0) \bmod 2 = 1$

$Nc_2 = \sum (c_2, a_3, a_6, a_7) \bmod 2 = \sum (0, 0, 0, 0) \bmod 2 = 0$

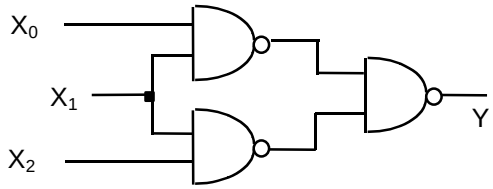
$Nc_3 = \sum (c_4, a_5, a_6, a_7) \bmod 2 = \sum (1, 1, 0, 0) \bmod 2 = 0$

$Nc = 001 = 1_{10} \Rightarrow$ (inversione del bit di controllo c_1)

⇒ 0 0 0 1 1 0 0 ← parola errata
c_1
1 0 0 1 1 0 0 ← parola corretta (w_4)
~~1~~ ~~0~~ 0 ~~1~~ 1 0 0 → parola decodificata: 0100
c_1 c_2 c_4

Esercizio (2S19991019-D2)

Implementare la rete combinatoria in figura con un multiplexer.



Questo problema può essere risolto in due fasi: 1) **analisi** della rete e identificazione dell'espressione logica della funzione y; 2) **sintesi** della funzione disponibile y mediante MUX.

1) **Analisi**

Si tratta di una rete combinatoria a due soli livelli di porte; pertanto l'espressione di y si può trascrivere senza passare attraverso le espressioni logiche associate ai nodi intermedi:

$$y = x_o \cdot x_1 + x_1 \cdot x_2$$

2) **Sintesi**

Conviene dapprima trasferire la funzione sulla tavola di verità.

x_2	x_1	x_0	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Una **prima soluzione** consiste nel presentare il vettore y in ingresso a un MUX 8/1 e le tre variabili x_2 x_1 x_0 agli ingressi di selezione.

Una **seconda soluzione** si basa sul calcolo di y in funzione di una delle variabili indipendenti; con questo accorgimento sugli ingressi di selezione del MUX compare una variabile in meno rispetto alla soluzione precedente, cioè il MUX viene dimezzato.

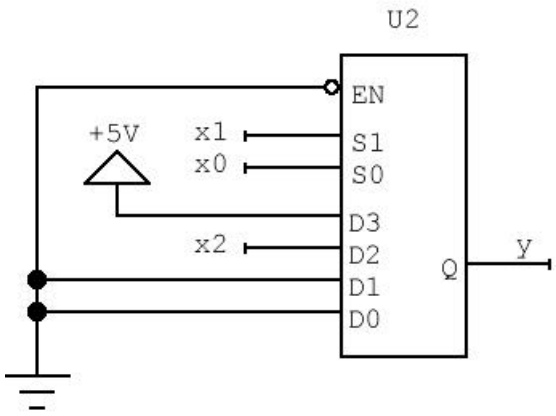
Scegliendo ad esempio x_2, si ha la tavola di verità seguente:

x_2	x_1	x_0	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Il vettore y con i valori 0, 1, x_2 può essere riscritto in una tavola di verità impostata sulle due variabili indipendenti residue x_1 x_0

x_1	x_0	y
0	0	0
0	1	0
1	0	x_2
1	1	1

Ora è sufficiente presentare il vettore y in ingresso a un MUX di tipo 4/1



Esercizio (2S19991019-D3)

Definire la struttura di un registro caricabile in modalità sincrona con dati a larghezza 8, 16 o 32 bit.

Il registro dovrà essere composto di tre sezioni indipendenti, cioè di tre registri di 8 bit (i bit da 0 a 7), 8 bit (i bit da 8 a 15) e 16 bit (i bit da 16 a 31), dotati di ingresso di abilitazione al caricamento (LD_B, LD_W, LD_LW rispettivamente), e pilotati dallo stesso clock.

Il tipo di caricamento sarà selezionato dallo stato di 2 bit: LD1, LD0, ad esempio secondo la seguente tavola di verità, in cui è prevista naturalmente una configurazione non operativa (00: NOP).

LD1	LD0	LD_LW	LD_W	LD_B
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

Dalla tavola si ottiene direttamente:

LD_B = LD1 + LD0
LD_W = LD1
LD_LW = LD1 · LD0

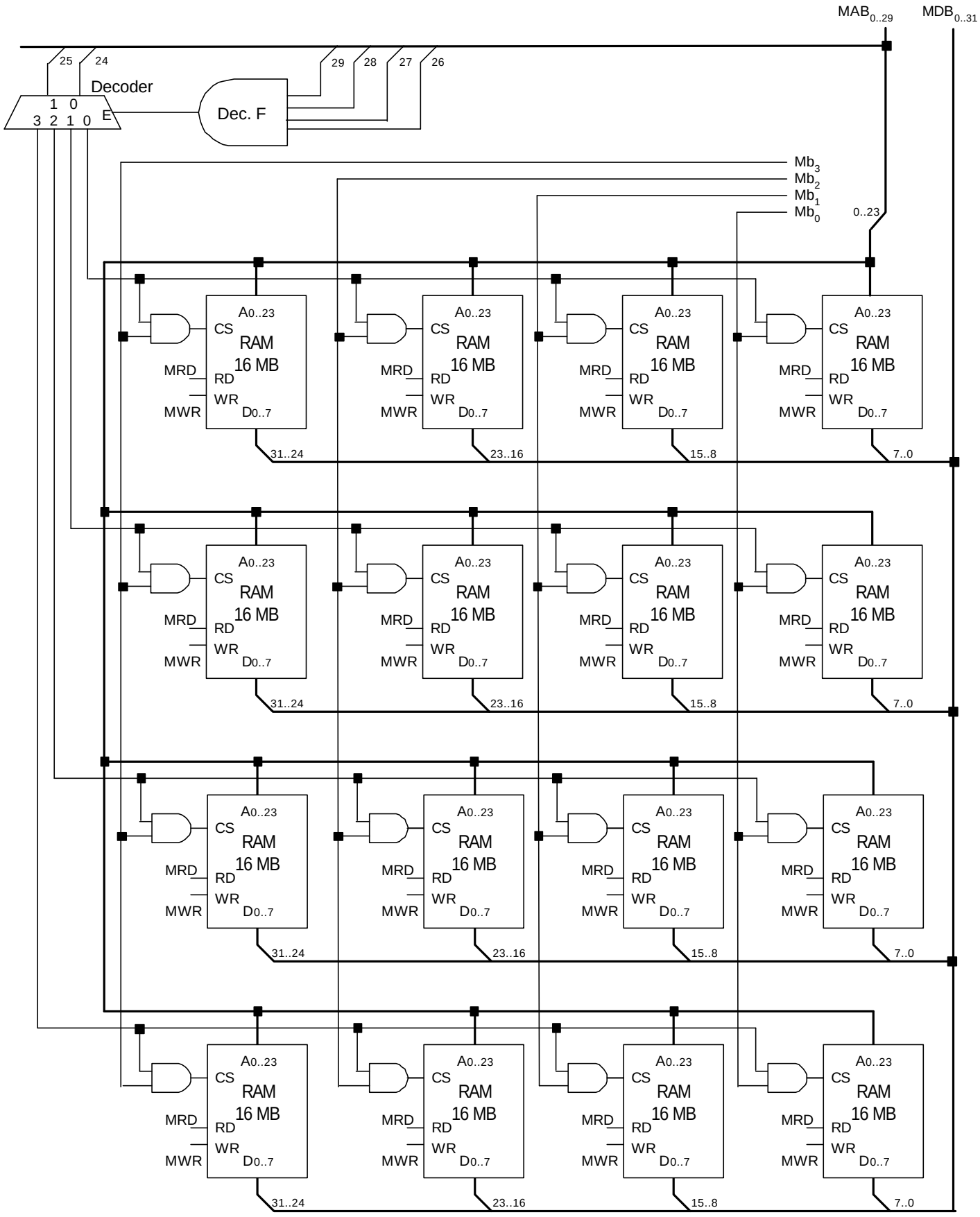
Esercizio (2S19991019-D4)

Organizzare un banco di RAM di capacità 256 Mbytes, indirizzabili dal processore PD32 a partire dall'indirizzo F0000000h, utilizzando moduli RAM da 16 MBytes.

Dalle specifiche del banco di memoria (RAM / ROM):
- indirizzabile da PD32 a partire dall'indirizzo F0000000h
- capacità: 256 MByte
- costituito da moduli di capacità 16 MByte

si può derivare direttamente l'organizzazione del banco:
256 / 16 = 16 moduli
16 moduli / 4 banchi = 4 righe di 4 moduli
256 MB = 64 MLW = 226 LW ==> i bit $MAB_{0..25}$ assumono tutte le configurazioni; i bit $MAB_{26..29}$ restano costanti (pari a Fh=1111b) e pertanto identificano il banco e vanno decodificati.
Dei bit $MAB_{0..25}$ i due più pesanti $MAB_{24..25}$ identificano le quattro righe del banco, e pertanto gli altri $MAB_{0..23}$ sono i 24 bit di indirizzo comune ai 16 moduli (2^{24} =16 M).

Le connessioni complete del banco sono riportate nella figura seguente.


MAB0..29
MDB0..31
25
24
29
28
27
26
Decoder
1 0
3 2 1 0 E
Dec. F
Mb3
Mb2
Mb1
Mb0
0..23
A0..23
CS
RAM
16 MB
MRD
RD
WR
MWR
D0..7
31..24
23..16
15..8
7..0

Esercizio (2S19991019-D5)

Una periferica di un processore PD32 campiona periodicamente una linea analogica mediante un comparatore analogico (convertitore AD a 1 bit). I bit rilevati vengono inviati singolarmente mediante interruzione al processore, che provvede a memorizzarli consecutivamente nella RAM, 32 bit di conversione per longword, a partire dalla locazione WAVE. La produzione dei dati viene avviata da un comando di inizio operazione emesso dal micro verso la periferica e viene arrestata con un comando di fine operazione al raggiungimento di 32 K campioni (bit).
Scrivere il programma di acquisizione in assembler PD32.

```
        org 400h            ;inizio programma

; ************************
; COSTANTI
; ************************

        WAVE   equ 1000h ;inizio area campioni
        STACK equ 2800h  ;inizio area di stack
                         ;limitato a 2800h per consentire la simulazione
        input    equ 080h   ;indirizzo periferica input
                            ;INPUT:       I/O=I, ind=80h, IVN=4
        output equ 081h     ;indirizzo periferica output:
                            ;inizio operazioni: dato #01h
                            ;fine operazioni: dato #02h
        bitbuff         db 0   ;byte (all'indirizzo 400h) scritto dal driver
                               ;con il valore prelevato dalla periferica
        bitflag         db 0   ;byte (all'indirizzo 401h) posto a 1 dal driver per
                               ;indicare dato pronto (bit di handshake)
        puntlw          dl 0   ;puntatore alla longword di accumulo corrente
        countbit        db 0   ;contatore dei bit accumulati nella longword di accumulo

; ************************
; CODICE
; ************************

        code                ;inizio istruzioni
main:
        movl #STACK,r7      ;inizializza R7 quale SP
        seti                ;abilita interruzioni (SP è stato inizializzato)

        jsr setpnt          ;inizializza il puntatore alla longword
                            ;il contatore dei bit della longword
        outb #01h,output    ;inizio operazioni dell'ADC

mainloop:
        movb bitflag,r0     ;test sulla produzione di un bit del device
        andb r0,r0
        jz skiploop
```

```
;segue trattamento del bit della periferica
        movb bitbuff,r0     ;copia del bit 0 di bit_buff
        rcrb #1,r0          ;nel bit di carry

        movl puntlw,r1
        movl (r1),r0        ;scalamento di carry e
        rcll #1,r0          ;di R0 a sinistra di un bit
        movl r0,(r1)        ;aggiornamento della longword corrente

;aggiornamento del contatore dei bit
        movb countbit,r0    ;incremento (di 1) del puntatore ai bit
        addb #1,r0
        movb r0,countbit

        cmpb #32,r0         ;si sono accumulati 32 bit?
        jnz pntok
        movb #0,countbit    ;azzera contatore dei bit

;incremento del puntatore alla longword di accumulo
        movl puntlw,r0      ;incremento del puntatore alle longword
        addl #4,r0          ;(incremento di 4)
        movl r0,puntlw

        cmpl #wave+4096,r0      ;si sono scritte 1024 LW (4096 byte)?
        jnz pntok
        outb #02h,output    ;termina operazioni dell'ADC
        jmp skiploop
pntok:
        movb #0,bitflag     ;resetta la locazione flag (dato consumato)

skiploop:
;spazio per altri segmenti del programma


;al termine dei quali è prevista la chiusura del loop su mainloop
        jmp mainloop


;*********************************
;SUBROUTINE
;*********************************

setpnt:

        movl #WAVE,puntlw       ;puntatore all'area di base WAVE
        movb #0,countbit        ;contatore dei bit compattati entro
                                ;la longword corrente
        ret                     ;ritorno da subroutine
```

```
;*********************************
;DRIVER DI INPUT
;*********************************
;Preleva un bit dalla periferica e setta una flag

        driver 4, 900h          ;Il driver della periferica con IVN=4
                                ;inizia dall'ind. 900h
        inb input,bitbuff       ;copia il bit (IODB_0) della periferica
                                ;nella locazione bit_buff
        movb #1,bitflag         ;setta la locazione flag (dato pronto)
        rti                     ;ritorno da interruzione


        halt                    ;termine elaborazione (serve per la simulazione)

        end                     ;fine programma
```

Commenti

Il programma prevede l'uso (tipico) di una routine di interruzione che riduce il proprio intervento alle operazioni essenziali di prelevamento del dato e di segnalazione di dato pronto (variabile *bitflag*) al programma principale, con il quale la routine di interrupt stabilisce un handshake. In questa implementazione è il programma principale ad effettuare l'elaborazione sul dato prodotto dalla periferica, e pertanto il programma principale deve passare ciclicamente su *mainloop*, per poter acquisire i dati prodotti in successione dalla periferica. Inoltre, se la periferica produce dati ad un ritmo cadenzato, senza un handshake con il processore, il programma principale deve poter passare su mainloop più frequentemente del ritmo di produzione della periferica.
Una soluzione alternativa consiste nell'affidare l'intera elaborazione al driver.
D'altra parte, il driver potrebbe essere ridotto alla sola segnalazione di dato pronto, lasciando al programma principale anche il compito di prelevare il dato dalla periferica.

```
;Una periferica di un processore PD32 campiona periodicamente
;una linea analogica mediante un comparatore analogico
;(convertitore AD a 1 bit). I bit rilevati vengono inviati
;singolarmente mediante interruzione al processore,
;che provvede a memorizzarli consecutivamente nella RAM,
;32 bit di conversione per longword, a partire dalla
;locazione WAVE. La produzione dei dati viene avviata
;da un comando di inizio operazione emesso dal micro
;verso la periferica e viene arrestata con un comando
;di fine operazione al raggiungimento di 32 K campioni (bit).



  org 400h    ;inizio programma


; ************************
; COSTANTI
; ************************


  WAVE  equ 1000h   ;inizio vettore istogramma (256 word)

  STACK   equ 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione

  input equ 080h  ;indirizzo periferica input
        ;INPUT: I/O=I, ind=80h, IVN=4
  output  equ 081h  ;indirizzo periferica output:
        ;inizio operazioni: dato #01h
        ;fine operazioni: dato #02h

  bitbuff db 0    ;byte (all'indirizzo 400h) scritto dal driver
        ;con il valore prelevato dalla periferica
  bitflag db 0    ;byte (all'indirizzo 401h) posto a 1 dal driver per
        ;indicare dato pronto (bit di handshake)
  puntlw   dl 0   ;puntatore alla longword di accumulo corrente
  countbit db 0   ;contatore dei bit accumulati nella longword di accumulo


; ************************
; CODICE
; ************************


  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)


  jsr setpnt    ;inizializza il puntatore alla longword
        ;il contatore dei bit della longword
  outb #01h,output  ;inizio operazioni dell'ADC

mainloop:

  movb bitflag,r0   ;test sulla produzione di un bit del device
  andb r0,r0
  jz skiploop

;segue trattamento del bit della periferica

  movb bitbuff,r0   ;copia del bit 0 di bit_buff
  rcrb #1,r0    ;nel bit di carry

  movl puntlw,r1
```

```
  movl (r1),r0    ;scalamento di carry e
  rcll #1,r0    ;di R0 a sinistra di un bit
  movl r0,(r1)    ;aggiornamento della longword corrente

;aggiornamento del contatore dei bit

  movb countbit,r0  ;incremento (di 1) del puntatore ai bit
  addb #1,r0
  movb r0,countbit

  cmpb #32,r0   ;si sono accumulati 32 bit?
  jnz pntok
  movb #0,countbit  ;azzera contatore dei bit

;incremento del puntatore alla longword di accumulo

  movl puntlw,r0    ;incremento del puntatore alle longword
  addl #4,r0    ;(incremento di 4)
  movl r0,puntlw

  cmpl #wave+4096,r0  ;si sono scritte 1024 LW (4096 byte)?
  jnz pntok
  outb #02h,output  ;termina operazioni dell'ADC
  jmp skiploop

pntok:
  movb #0,bitflag   ;resetta la locazione flag (dato consumato)


skiploop:
;spazio per altri segmenti del programma




;al termine dei quali è prevista la chiusura del loop su mainloop
  jmp mainloop




;*********************************
;SUBROUTINE
;*********************************

setpnt:

  movl #WAVE,puntlw ;puntatore all'area di base WAVE
  movb #0,countbit  ;contatore dei bit compattati entro
        ;la longword corrente
  ret     ;ritorno da subroutine


;*********************************
;DRIVER DI INPUT
;*********************************

;Preleva un bit dalla periferica e setta una flag


  driver 4, 900h    ;Il driver della periferica con IVN=4
        ;inizia dall'ind. 900h

  inb input,bitbuff ;copia il bit (IODB_0) della periferica
        ;nella locazione bit_buff
  movb #1,bitflag   ;setta la locazione flag (dato pronto)

  rti     ;ritorno da interruzione
```

```
  halt      ;termine elaborazione (serve per la simulazione)
  end     ;fine programma
```

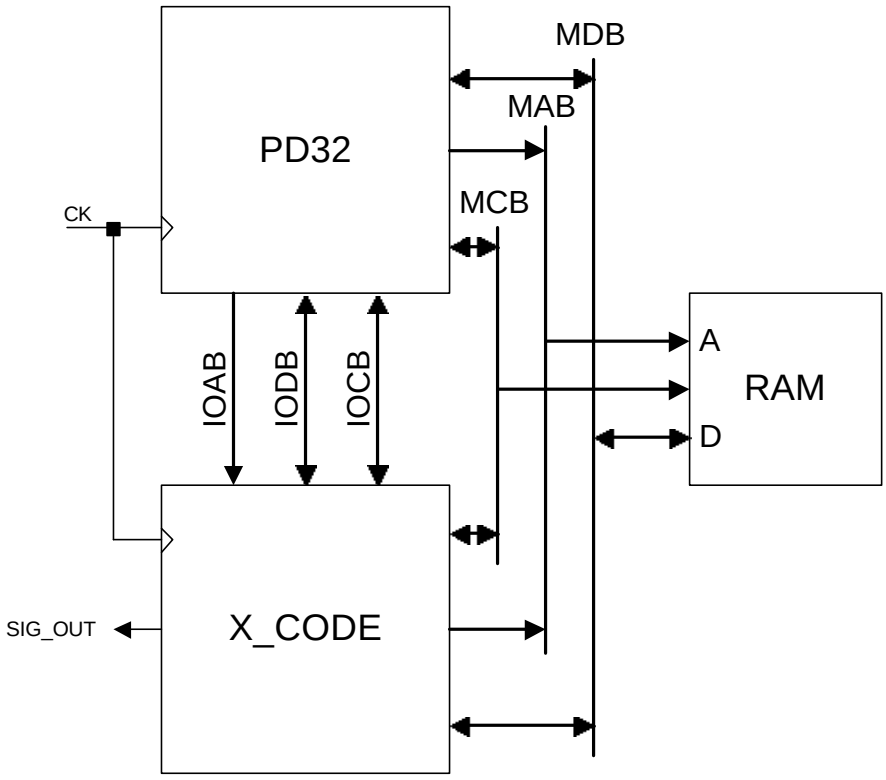
RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 17-01-2000

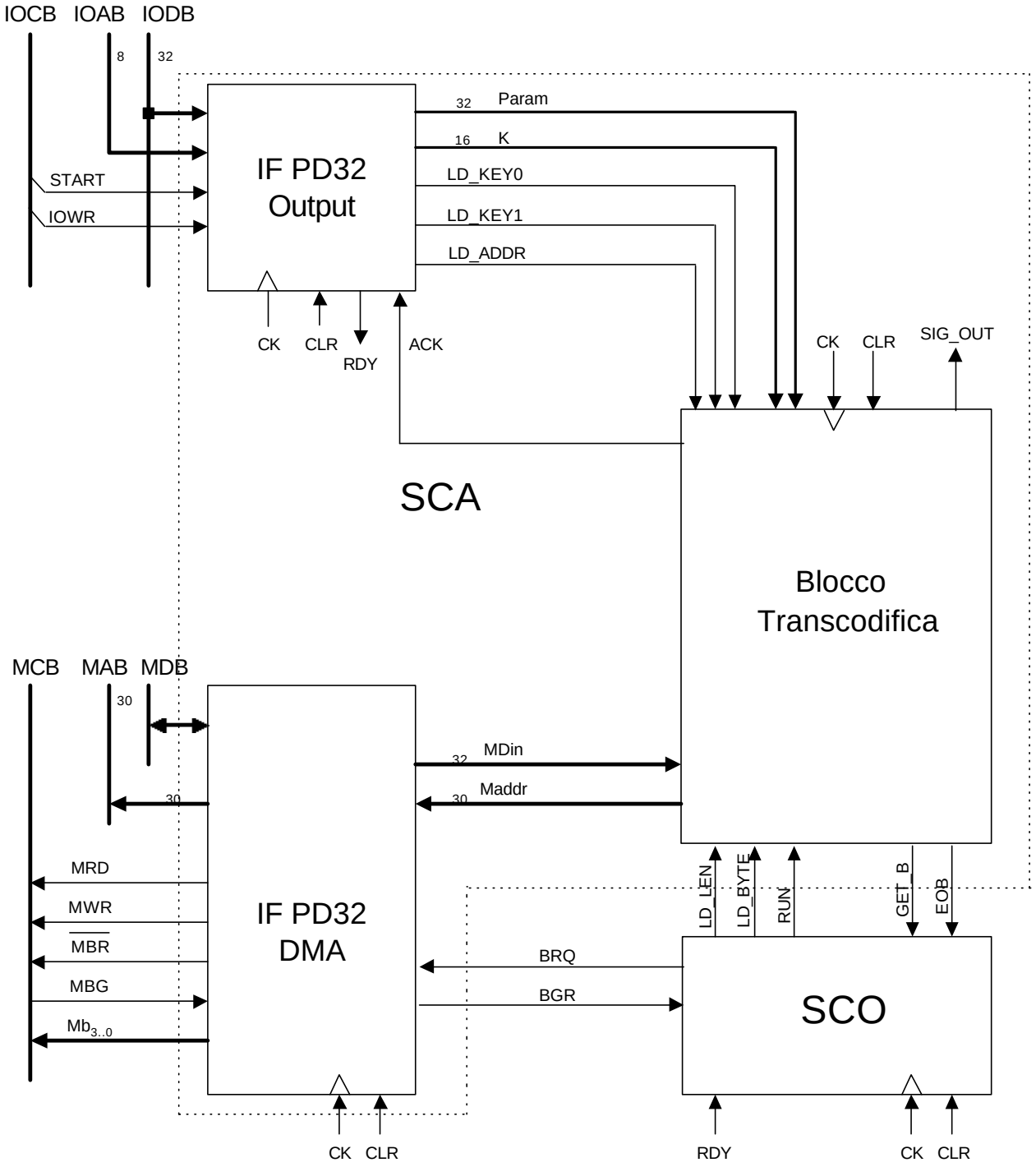
Studente:_____ Docente:_____

Il testo originale di questa prova non è disponibile

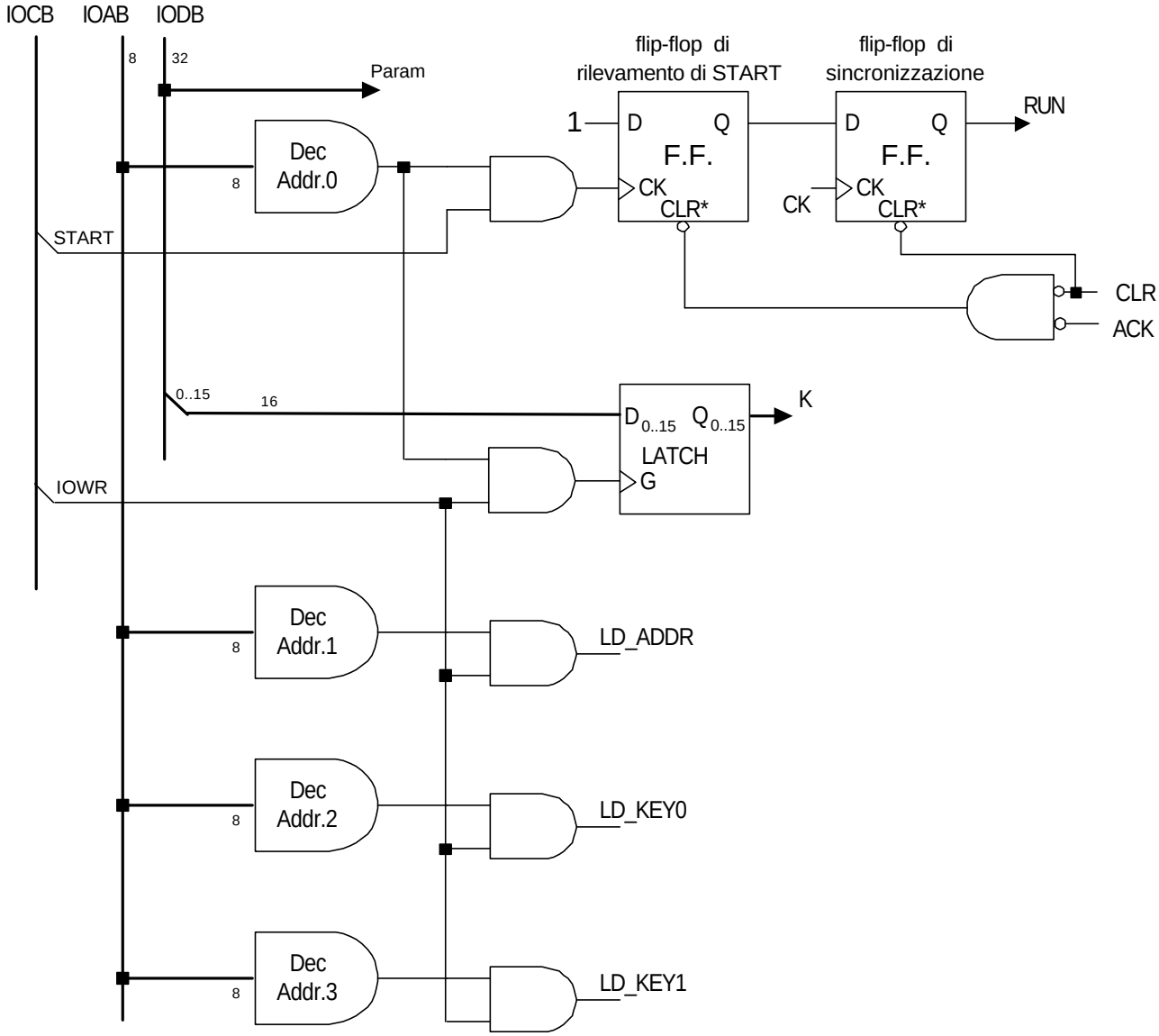
X_CODE: sistema esterno



X_CODE: Schema a blocchi



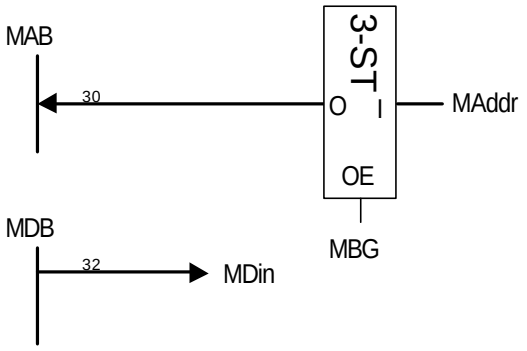
X_CODE: IF PD32 - output

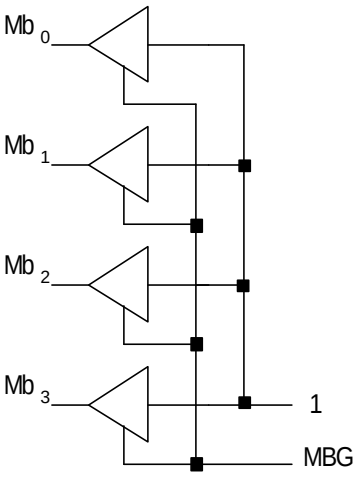


Note

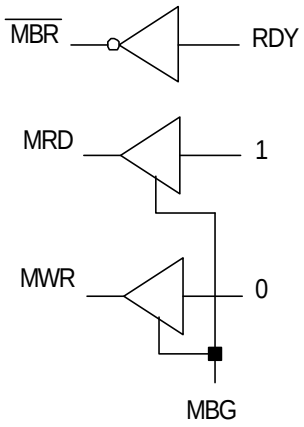
Il SW avvia l'operazione con **START Addr0**, dopo avere eventualmente riscritto gli altri registri.

X_CODE: IF PD32 - DMA



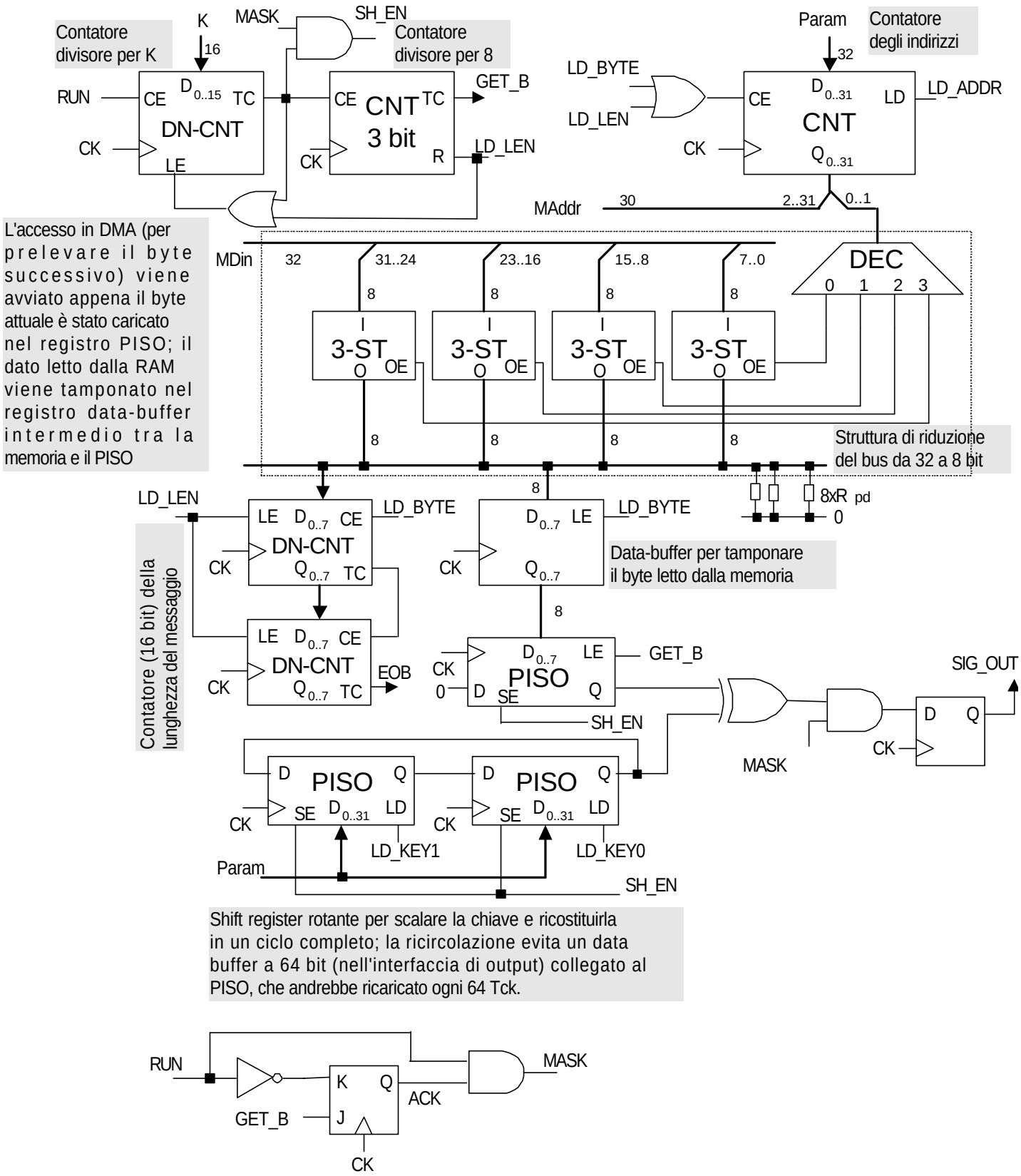


Dalla RAM vengono lette LW, di cui però viene registrato un byte alla volta.

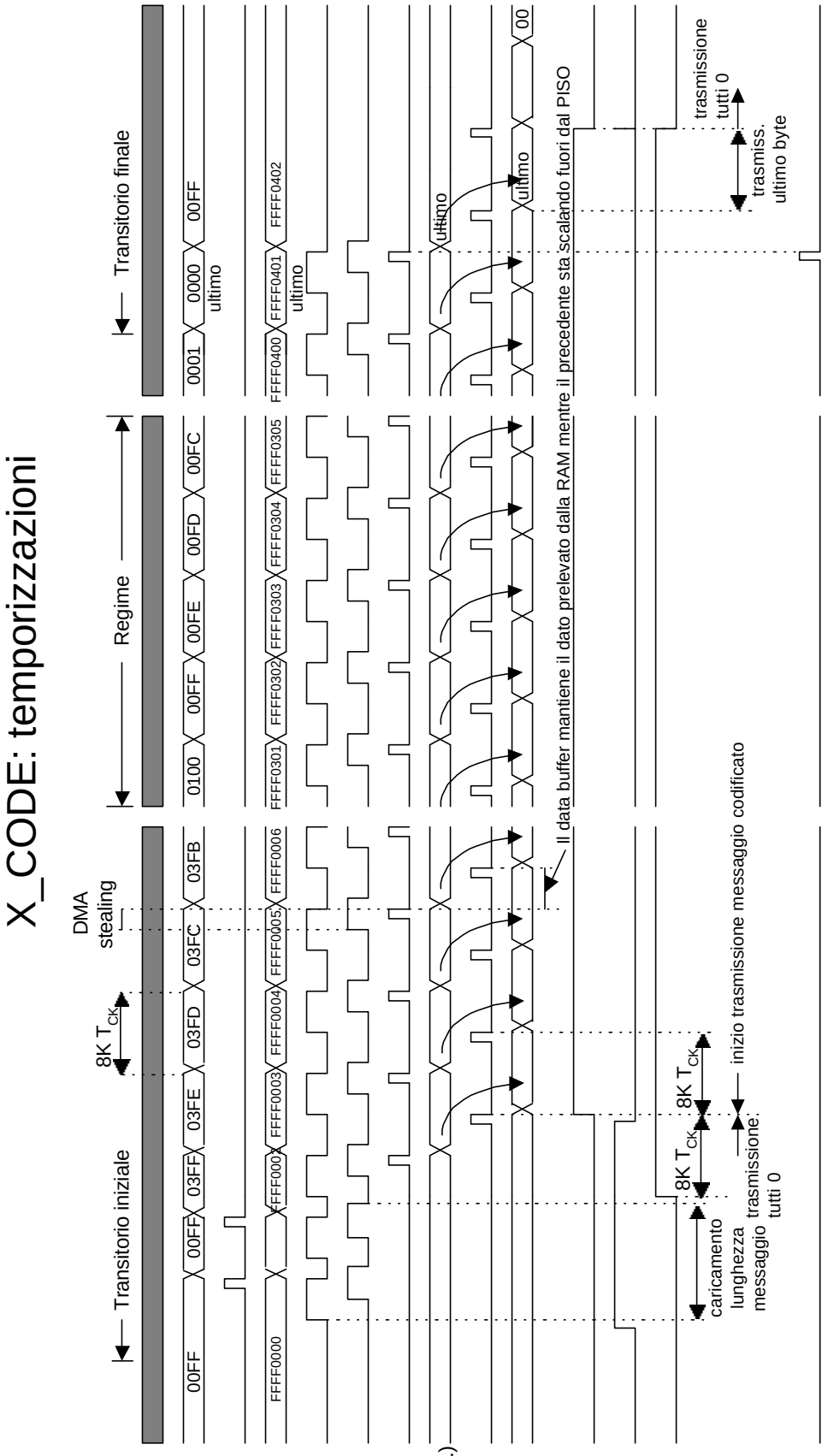


MWR deve essere pilotato (a 0), anche se non dinamicamente: la struttura di memoria deve essere governata pienamente dall'interfaccia DMA, che ne ha piena responsabilità quando MBG=1.

X_CODE: blocco generatore segnale



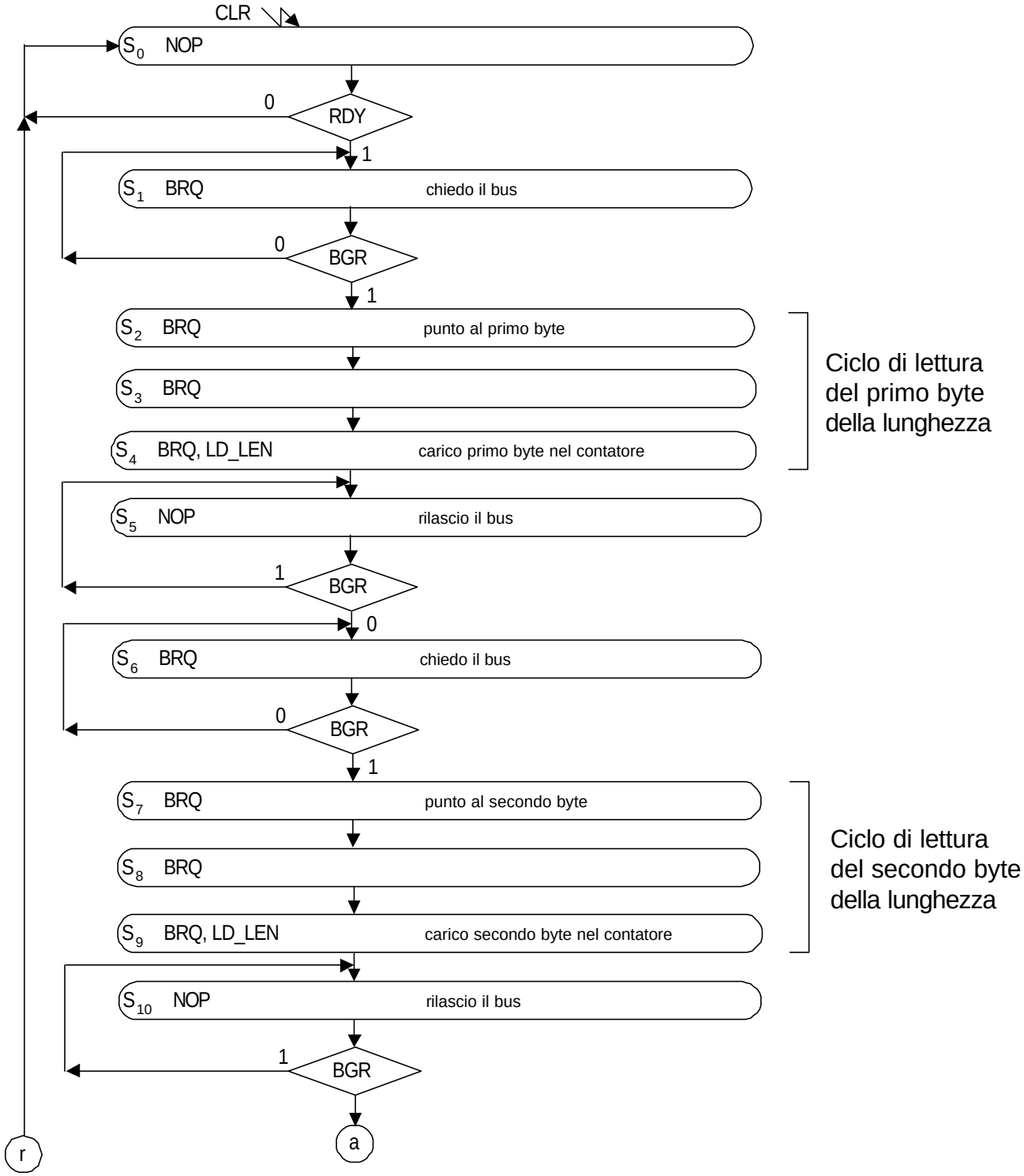
X_CODE: temporizzazioni



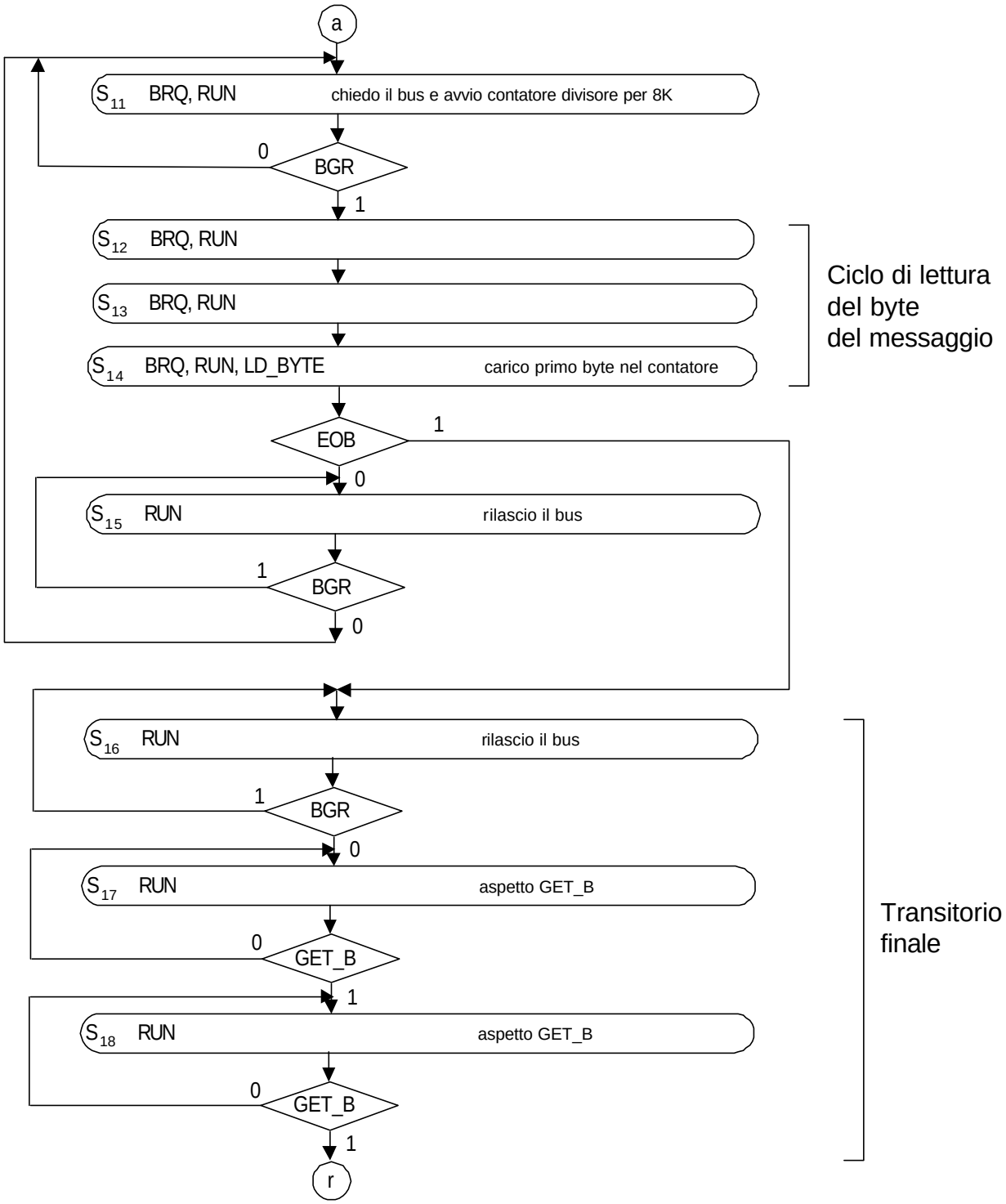
Note

Si suppone predisposto un messaggio all'indirizzo FFFF0000h, di lunghezza 1024 (400h = 3FFh +1) byte, che seguono i 2 byte di lunghezza.
I primi due byte in memoria indicano la lunghezza del corpo del messaggio decrementata di 1: in questo esempio, 3FFh per indicare una lunghezza pari a 1024.

X_CODE: SCO - flowchart

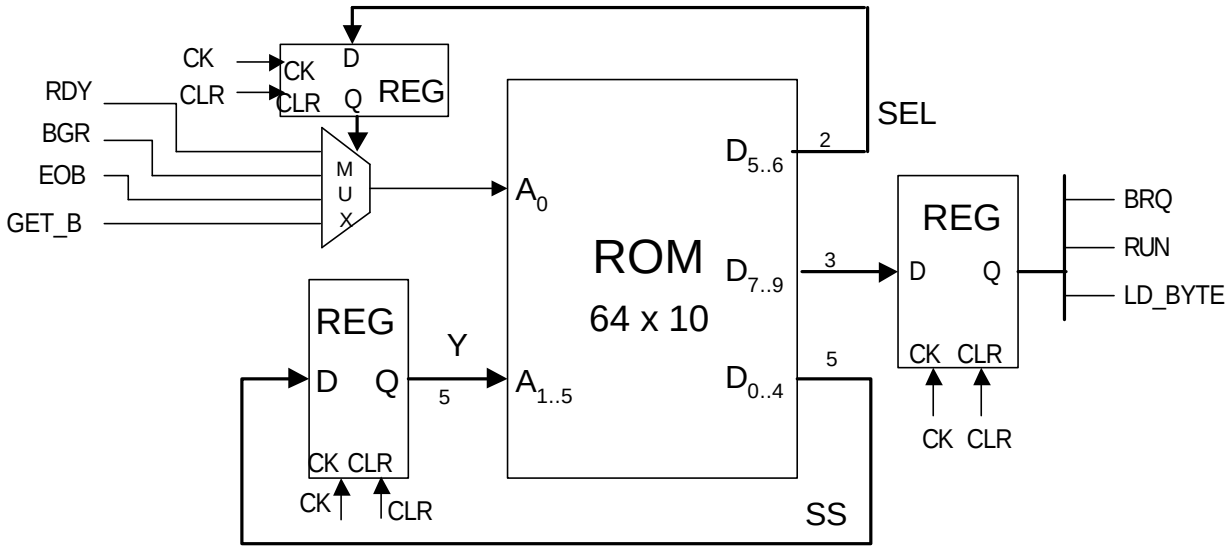


X_CODE: SCO - flowchart



X_CODE: SCO - struttura HW microprogrammata

Il microprogramma (non codificato) è supportato da un microlinguaggio di tipo 3; l'implementazione mediante il modello strutturale di tipo D-Mealy comporta la struttura seguente:



NOTE

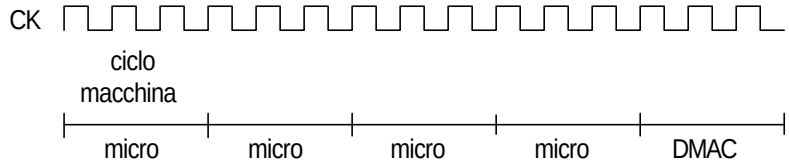
Calcolo del parametro K tale da rallentare il micro non più del 20%:
indicando con:
R il rallentamento percentuale subìto dal processore per effetto dell'inserimento della periferica in DMA,
T_{DMA} il periodo tra due accessi consecutivi in DMA, misurato in numero di cicli-macchina del processore,
e nell'ipotesi di effettuare un accesso in un singolo ciclo-macchina del micro (modalità stealing), la periferica sottrae al processore un ciclo-macchina ogni T_{DMA}, e quindi lo rallenta di:

$R = 1 / T_{DMA}$

Imponendo R <= 20% si ottiene:

$1/ T_{DMA} <= 0.2$ da cui: $T_{DMA} >= 5$

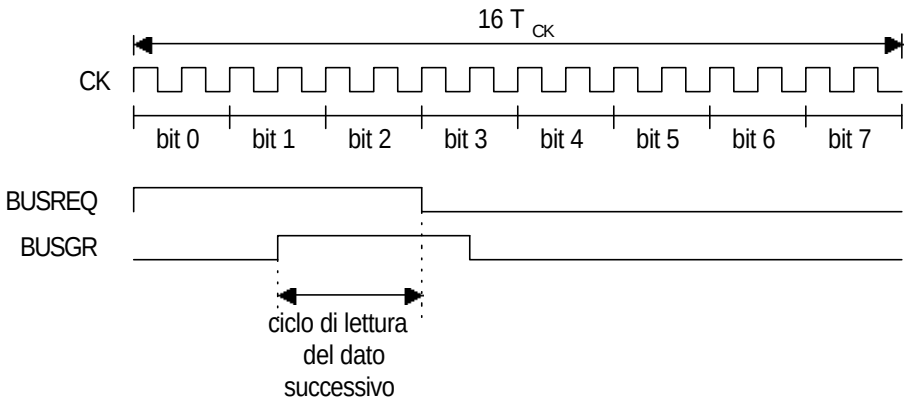
Con l'evidente significato di poter sottrarre il bus al micro per non più di un ciclo-macchina del micro ogni cinque, secondo la situazione riportata nel grafico seguente:



Un ciclo macchina ha una durata di 3 T_{CK}; pertanto 5 cicli-macchina corrispondono a 15 T_{CK}; per quanto detto sopra, la periferica potrà accedere ai bus di sistema con un periodo non inferiore a 15 T_{CK}. Poiché la periferica impiega 8K cicli di clock per trametterе un byte, allora deve essere:

$8K >= 15$ con K intero, da cui $K_{min}=2$.

La dinamica degli accessi in memoria è descritta dal diagramma temporale seguente, in cui sono evidenziate le azioni parallele di trasmissione seriale del byte attuale (nello shift-register) e dell'accesso in lettura del byte successivo, che deve essere tamponato in un registro di appoggio, in attesa di essere trasferito (al termine dei 16 periodi di clock descritti) nello shift-register di uscita.



RETI LOGICHE

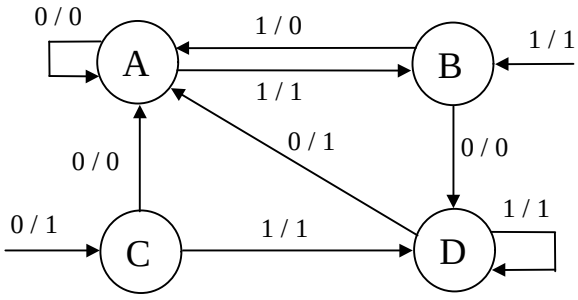
SECONDA PROVA SCRITTA DELL'APPELLO DEL 17-01-2000

Studente:_____ Docente:_____

D1 Indicare il valore numerico decimale associato alla stringa esadecimale F000000F
che espressa in binario rappresenta il numero nel formato in virgola mobile con un bit di segno, 23 bit di mantissa e 8 bit di esponente (in complemento a 2).

D2 Sintetizzare la funzione combinatoria XNOR (X_0, X_1, X_2) in logica steering.

D3 Trasformare il frammento di diagramma degli stati di tipo Mealy riportato in figura in uno equivalente di tipo Moore.



D4 Descrivere la struttura di una rete sequenziale a ingressi misti, con L ingressi a livello e P ingressi di trigger sovrapponibili.

D5 Nella memoria PD32 a partire dall'indirizzo SAMPLE sono allocati N (< 64K) bytes che rappresentano i valori dei campioni acquisiti tramite un convertitore analogico-digitale a 8 bit. Si richiede di scrivere una routine assembler PD32 che produca l'istogramma dei campioni (numero di occorrenze di ciascun valore nell'insieme dato), da disporre in 256 word a partire dall'indirizzo HYSTO.

Esercizio (2S20000117-D1)

Indicare il valore numerico decimale associato alla stringa esadecimale
F000000F
che espressa in binario rappresenta il numero nel formato in virgola mobile con un bit di segno, 23 bit di mantissa e 8 bit di esponente (in complemento a 2).

---

1. Stringa binaria:

11110000000000000000000000001111

2. Segmentazione della stringa binaria nel formato specificato:

1_11100000000000000000000_00001111 (S_M_E)

3. Interpretazione numerica dei tre campi:

S = 1: valore negativo
M: 0.5 + 0.25 + 0.125 = 0.875
E: 15

Ne segue che il numero cercato è:

$- 0.875\ 2^{15} = - 28672$

Esercizio (2S20000117-D2)

Sintetizzare la funzione combinatoria XNOR (X_0, X_1, X_2) in logica steering.

La funzione f specificata corrisponde alla tavola di verità seguente (è la funzione di parità pari, o anche la negazione della funzione XOR (X_0, X_1, X_2))

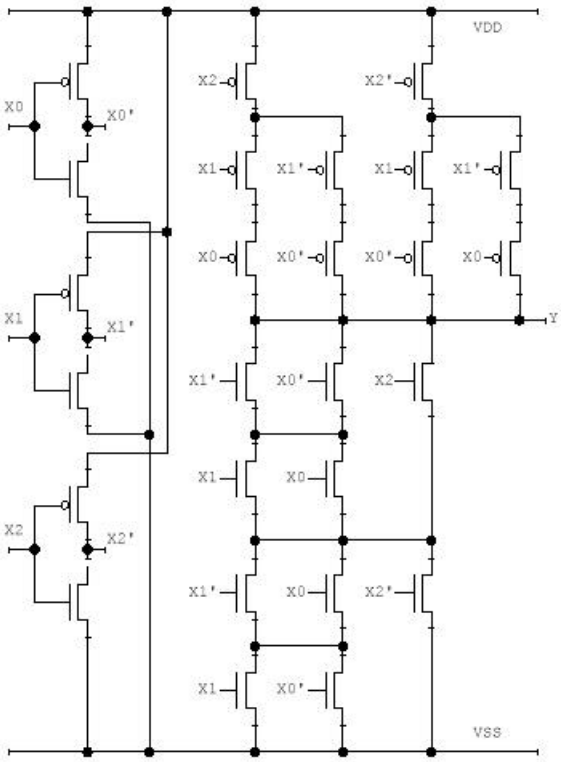
x_2	x_1	x_0	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

I mintermini non sono semplificabili (com'è evidente se si riportano su una MK); pertanto:

$$f = \overline{x_2}\,\overline{x_1}\,\overline{x_0} + \overline{x_2}\,x_1 x_0 + x_2\,\overline{x_1}\,x_0 + x_2 x_1\,\overline{x_0} =$$
$$\overline{x_2}(\overline{x_1}\,\overline{x_0} + x_1 x_0) + x_2(\overline{x_1}\,x_0 + x_1\,\overline{x_0})$$

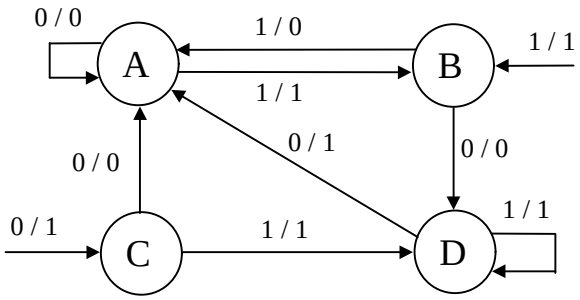
$$\overline{f} = (x_2 + (x_1 + x_0)(\overline{x_1} + \overline{x_0}))(\overline{x_2} + (x_1 + \overline{x_0})(\overline{x_1} + x_0))$$

L'implementazione segue direttamente:

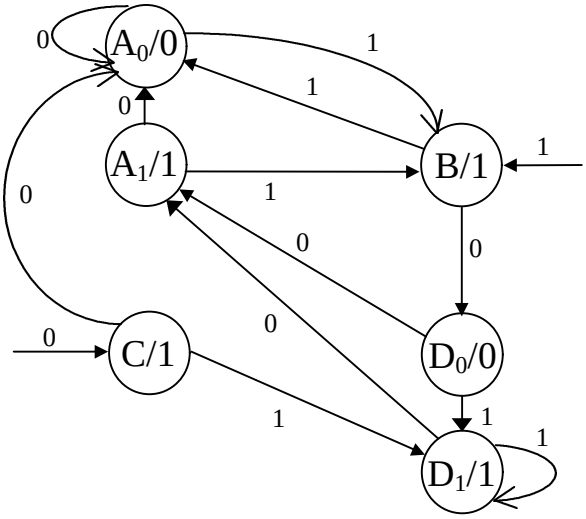


Esercizio (2S20000117-D3)

Trasformare il frammento di diagramma degli stati di tipo Mealy riportato in figura in uno equivalente di tipo Moore.



Gli stati cui si perviene tramite archi con uscite diverse devono essere diversificati nel diagramma di Moore equivalente; in questo caso solo gli stati A e D vanno raddoppiati.



Esercizio (2S20000117-D4)

Descrivere la struttura di una rete sequenziale a ingressi misti, con L ingressi a livello e P ingressi di trigger sovrapponibili.

Cfr. testo: “Reti Sequenziali”

Esercizio (2S20000117-D5)

Nella memoria PD32 a partire dall'indirizzo SAMPLE sono allocati N (< 64K) bytes che rappresentano i valori dei campioni acquisiti tramite un convertitore analogico-digitale a 8 bit. Si richiede di scrivere una routine assembler PD32 che produca l'istogramma dei campioni (numero di occorrenze di ciascun valore nell'insieme dato), da disporre in 256 word a partire dall'indirizzo HYSTO.

```
        org 400h                ;inizio programma

; ************************
; COSTANTI
; ************************
        N       equ 2000        ;N=2000
        HISTO   equ 1000h       ;inizio vettore istogramma (256 word)
        SAMPLE  equ 1200h       ;inizio area dati dall'ADC:
                                ;inizio vettore istogramma + 2x256 (byte)
        STACK   equ 2800h       ;inizio area di stack
                                ;limitato a 2800h per consentire la simulazione
; ************************
; CODICE
; ************************
        code                    ;inizio istruzioni
main:
        movl #STACK,r7          ;inizializza R7 quale SP
        seti                    ;abilita interruzioni (SP è stato inizializzato)

;ciclo di inizializzazione a 0 dell'area istogramma
        movl #histo,r1          ;r1 puntatore all'area istogramma
init:
        movw #0,(r1)+           ;scrive 0 e (post-)incrementa r1 (di 2)
        cmpl #histo+512,r1      ;confronto fine ciclo (512 byte = 256 word)
        jnz init

;ciclo operativo
        xorl r0,r0              ;r0 <- 0: puntatore all'area sample
loop:
        xorl r1,r1              ;r1 <- 0: puntatore relativo all'area istogramma
        mvlb sample(r0),r1      ;carica il primo byte di r1, SENZA ESTENDERE ;IL BIT 7
        lsll #1,r1              ;moltiplica r1 x 2 per ottenere l'indirizzo relativo
                                ;della word dell'istogramma associata al valore di ;r1
        movw histo(r1),r2       ;r2 appoggio per incrementare le occorrenze
        addw #1,r2
        movw r2,histo(r1)       ;scrive il valore incrementato nell'area ;istogramma
        addl #1,r0              ;incrementa puntatore r0 (di 1)
        cmpl #N,r0              ;test di fine ciclo
        jnz loop

        halt                    ;termine elaborazione (serve per la simulazione)
        end                     ;fine programma
```

```
;Nella memoria PD32 a partire dall'indirizzo SAMPLE
;sono allocati N (< 64K) bytes che rappresentano i valori
;dei campioni acquisiti tramite un convertitore
;analogico-digitale a 8 bit.
;Si richiede di scrivere una routine assembler PD32
;che produca l'istogramma dei campioni
;(numero di occorrenze di ciascun valore nell'insieme dato),
;da disporre in 256 word a partire dall'indirizzo HYSTO.


  org 400h    ;inizio programma


; ************************
; COSTANTI
; ************************

  N   equ 2000  ;N=2000
  HISTO   equ 1000h   ;inizio vettore istogramma (256 word)
  SAMPLE  equ 1200h   ;inizio area dati dall'ADC:
        ;inizio vettore istogramma + 2x256 (byte)


  STACK   equ 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione


; ************************
; CODICE
; ************************


  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)


;ciclo di inizializzazione a 0 dell'area istogramma
  movl #histo,r1    ;r1 puntatore all'area istogramma
init:
  movw #0,(r1)+   ;scrive 0 e (post-)incrementa r1 (di 2)
  cmpl #histo+512,r1  ;confronto fine ciclo (512 byte = 256 word)
  jnz init


;ciclo operativo
  xorl r0,r0    ;r0 <- 0: puntatore all'area sample
loop:
  xorl r1,r1    ;r1 <- 0: puntatore relativo all'area istogramma
  mvlb sample(r0),r1  ;carica il primo byte di r1, SENZA ESTENDERE IL BIT 7
  lsll #1,r1    ;moltiplica r1 x 2 per ottenere l'indirizzo relativo
        ;della word dell'istogramma associata al valore di r1
  movw histo(r1),r2 ;r2 appoggio per incrementare le occorrenze
  addw #1,r2
  movw r2,histo(r1) ;scrive il valore incrementato nell'area istogramma
  addl #1,r0    ;incrementa puntatore r0 (di 1)
  cmpl #N,r0    ;test di fine ciclo
  jnz loop


  halt      ;termine elaborazione (serve per la simulazione)
```

```
  end     ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DEL 01-02-2000

STUDENTE:_____ DOCENTE:_____

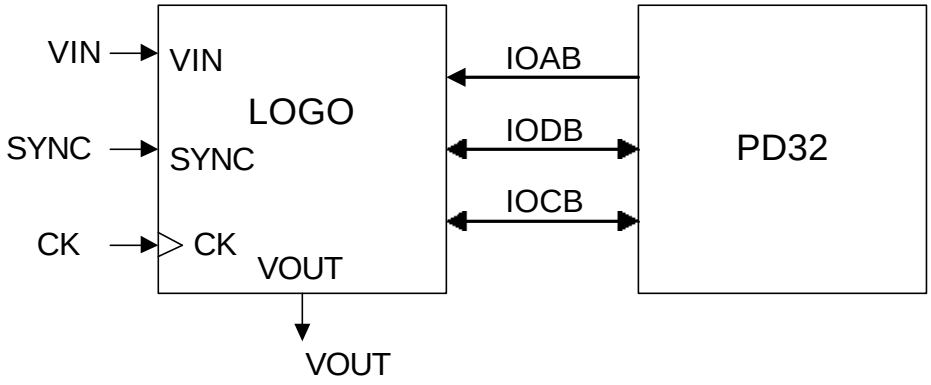
Si vuole progettare un dispositivo per l'inserimento del logo nel segnale video trasmesso da un'emittente televisiva. Il segnale video è disponibile in formato digitale secondo una trama costituita da una sequenza di immagini (50 al secondo) di 288 righe x 720 colonne; la scansione delle immagini avviene per righe, i cui elementi sono i campioni del segnale a 12 bit. Il logo è una immagine di 32 x 32 elementi a 12 bit, contenuti in un banco di ROM.
Il dispositivo riceve da un microprocessore PD32 le coordinate (riga, colonna) dell'elemento sul vertice in alto a sinistra della porzione (32 x 32) di immagine a cui va sostituito il logo.
Oltre alla linea dati VIN su cui vengono presentati i campioni, si suppongano disponibili in ingresso alla periferica una linea di clock CK sincronizzata con VIN ed una linea SYNC, attiva per due periodi di CK all'inizio della prima riga dell'immagine e per un periodo di CK all'inizio delle righe successive. Si assuma la frequenza di CK pari a 10 MHz e la disponibilità di moduli di memoria ROM con dati a 4 bit e tempo di accesso pari a 150 ns.

In uscita il dispositivo avrà una linea VOUT (che rappresenta il segnale video elaborato) SYNC e CK.

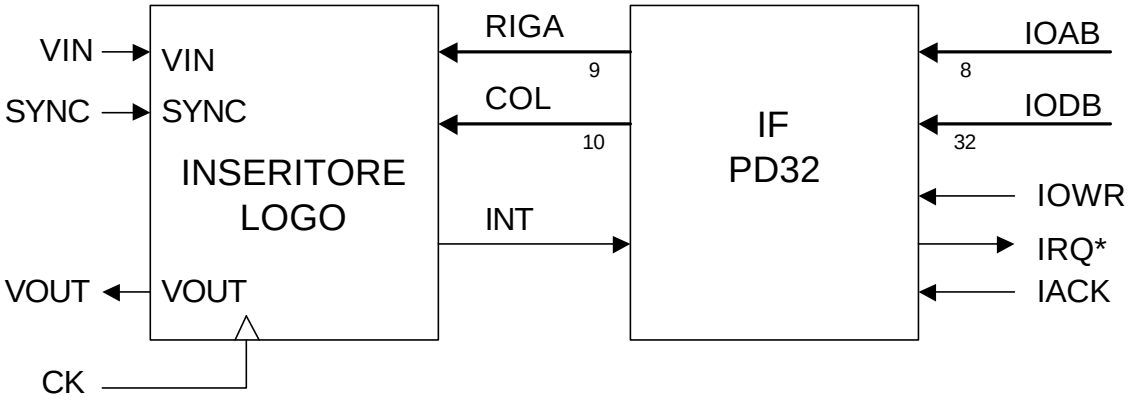
Si richiede:

- lo schema logico dettagliato della periferica e le relative temporizzazioni;
- il dimensionamento del banco di memoria ROM e la allocazione dei dati del logo.

LOGO: sistema esterno



LOGO: schema a blocchi

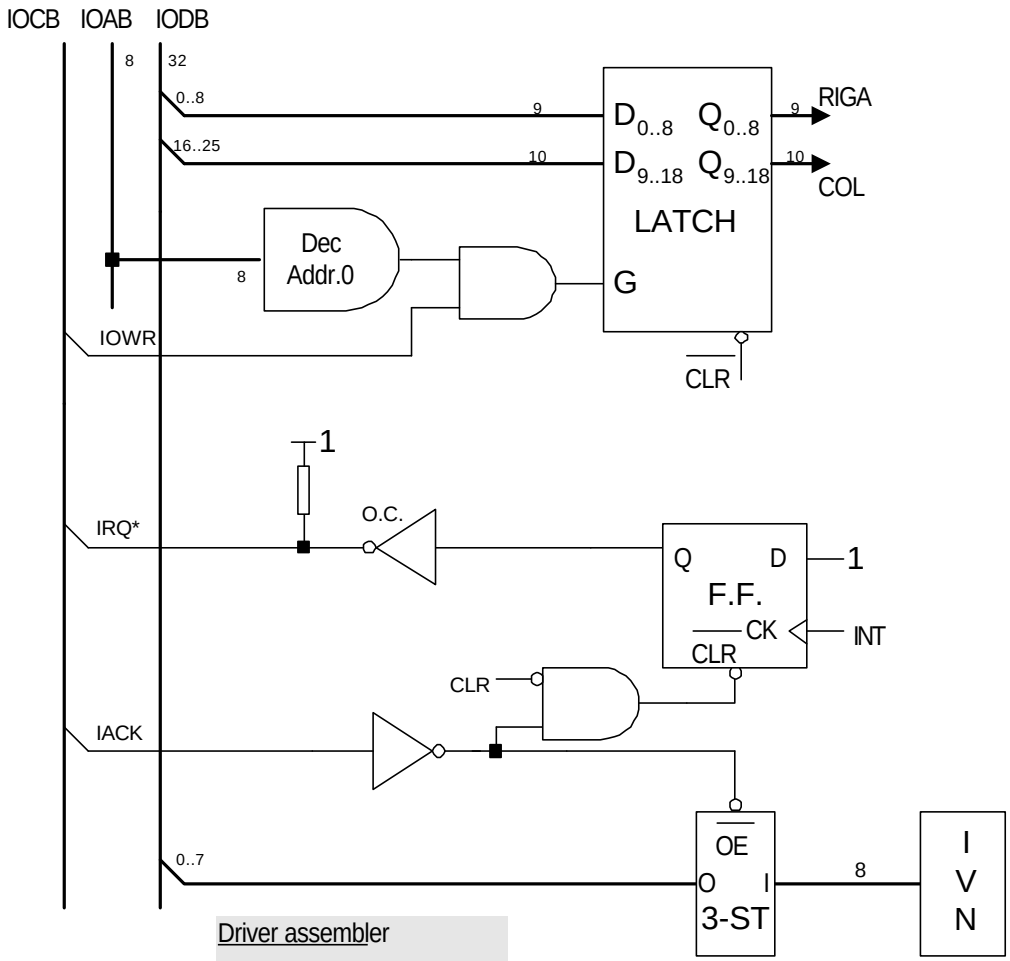


Note

Periferica di tipo output.

La periferica elabora il segnale DIN in tempo reale utilizzando come clock il segnale di sincronizzazione (CK) dei campioni VIN.

LOGO: IF PD32



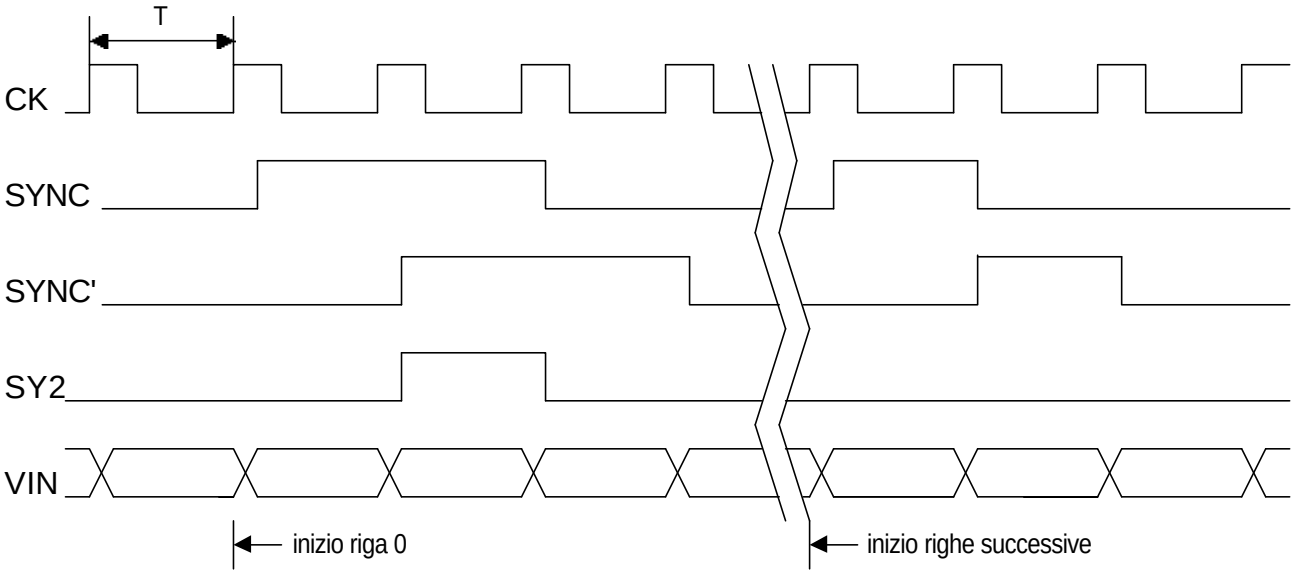
Driver assembler

```
main:
...
VERTEX DL 02000100h
. . .
DRIVER 10h,2000h
OUTL VERTEX,Addr0
RTI
```

Note

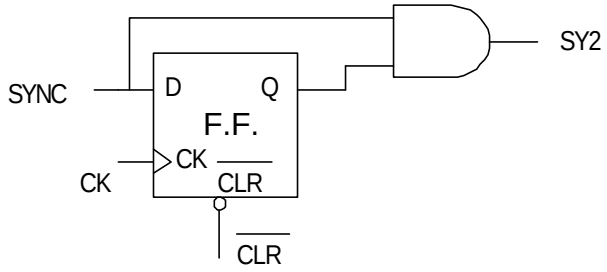
L'interfaccia di uscita include un unico latch (data buffer) caricabile dal micro con la coppia di coordinate (RIGA; COL a 19 bit) del vertice di riferimento del logo; le uscite del latch sono collegate agli ingressi dei registri di lavoro dello SCA, che si suppone vengano ricaricati periodicamente (T=1/50 s = 20 ms), all'inizio di ogni trama video (indicata dal segnale SY2). Come è noto, per avere un funzionamento affidabile occorre garantire che all'atto del caricamento dei registri l'uscita del latch sia stabile, cioè occorre precludere l'eventualità che il micro scriva nel data buffer proprio mentre i registri si stanno ricaricando. A questo scopo occorre dotare l'interfaccia di un meccanismo di sincronizzazione con il micro: considerata la durata relativamente estesa (20 ms) del periodo di ricaricamento rispetto ai tempi di esecuzione del micro, la scelta più conveniente è quella (di fatto utilizzata in pratica nelle applicazioni di questo tipo) di inviare un interrupt al micro ad ogni inizio immagine video (indicato dal segnale SYNC persistente per due cicli di clock), subito dopo avere ricaricato i registri; in questo modo il micro avrà ben 20 ms a disposizione per inviare una nuova coppia di coordinate.

LOGO: temporizzazioni

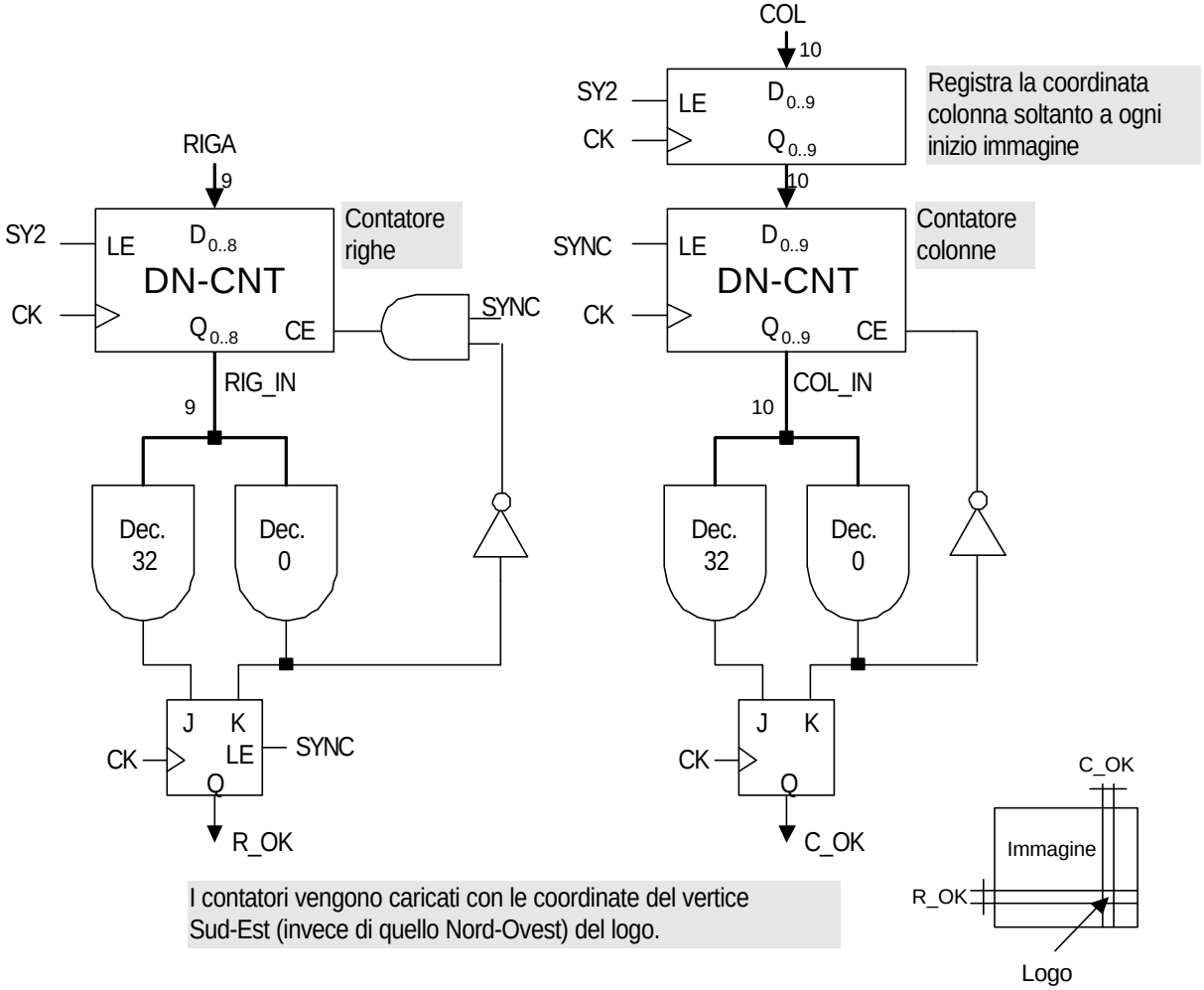


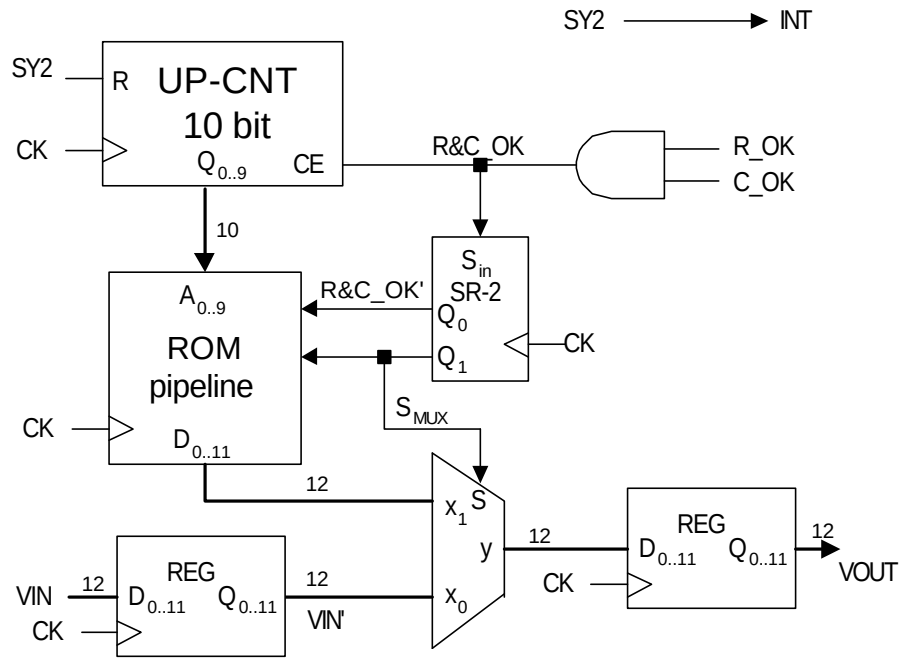
Calcolo del circuito rilevatore di inizio immagine (riga 0)

SY2 = SYNC SYNC' con SYNC'(t) = SYNC (t-T)

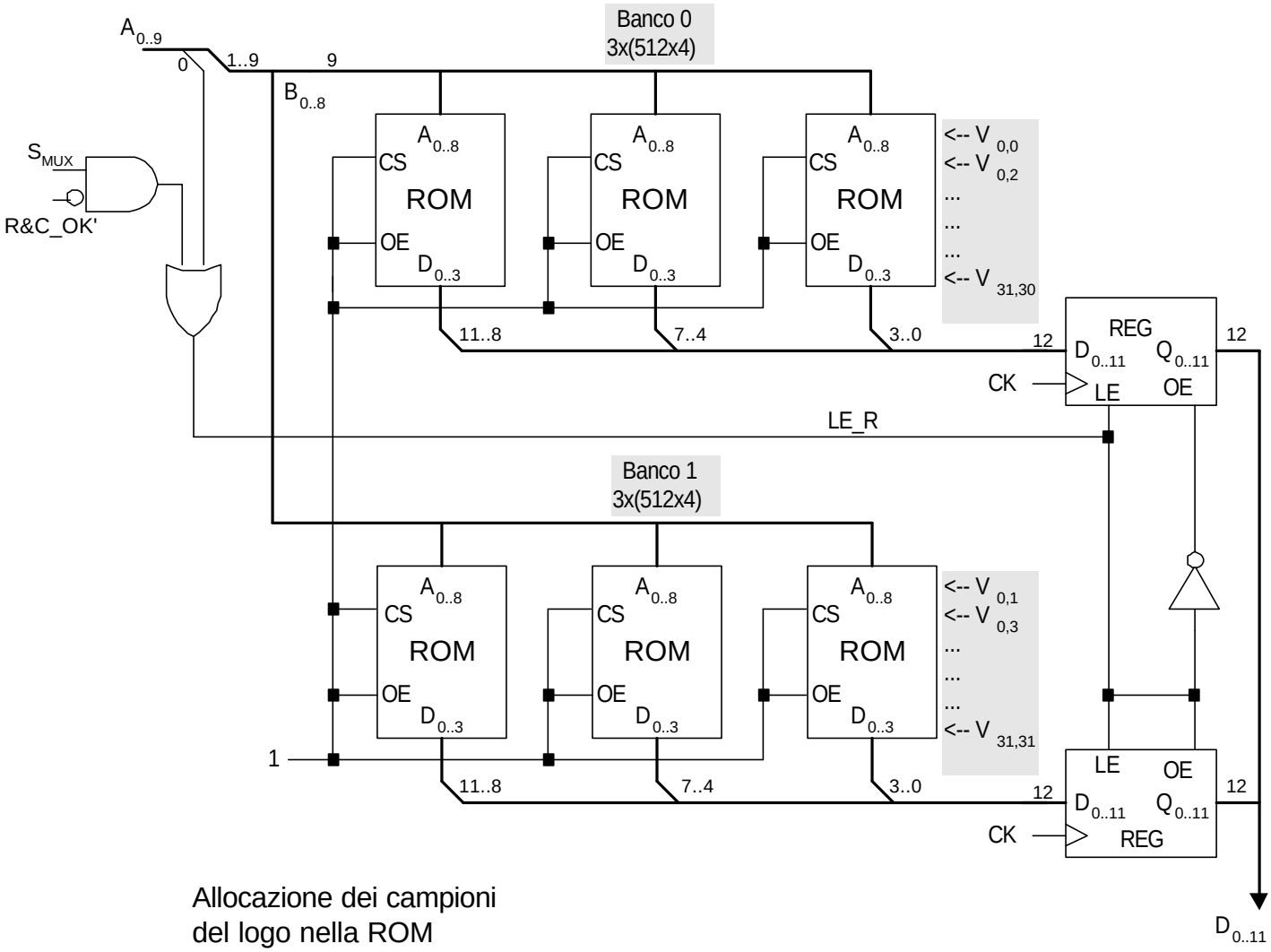


LOGO: inseritore

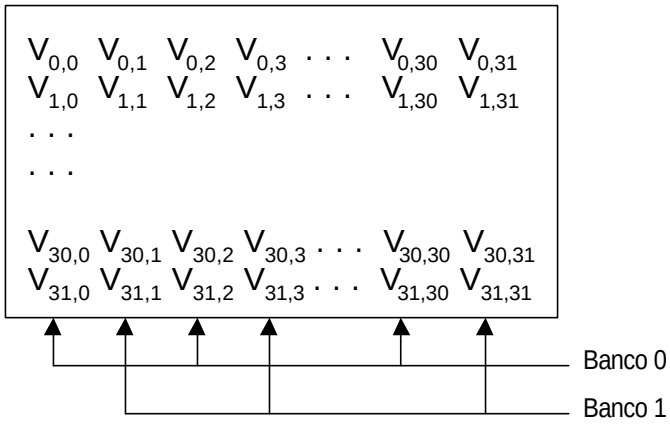




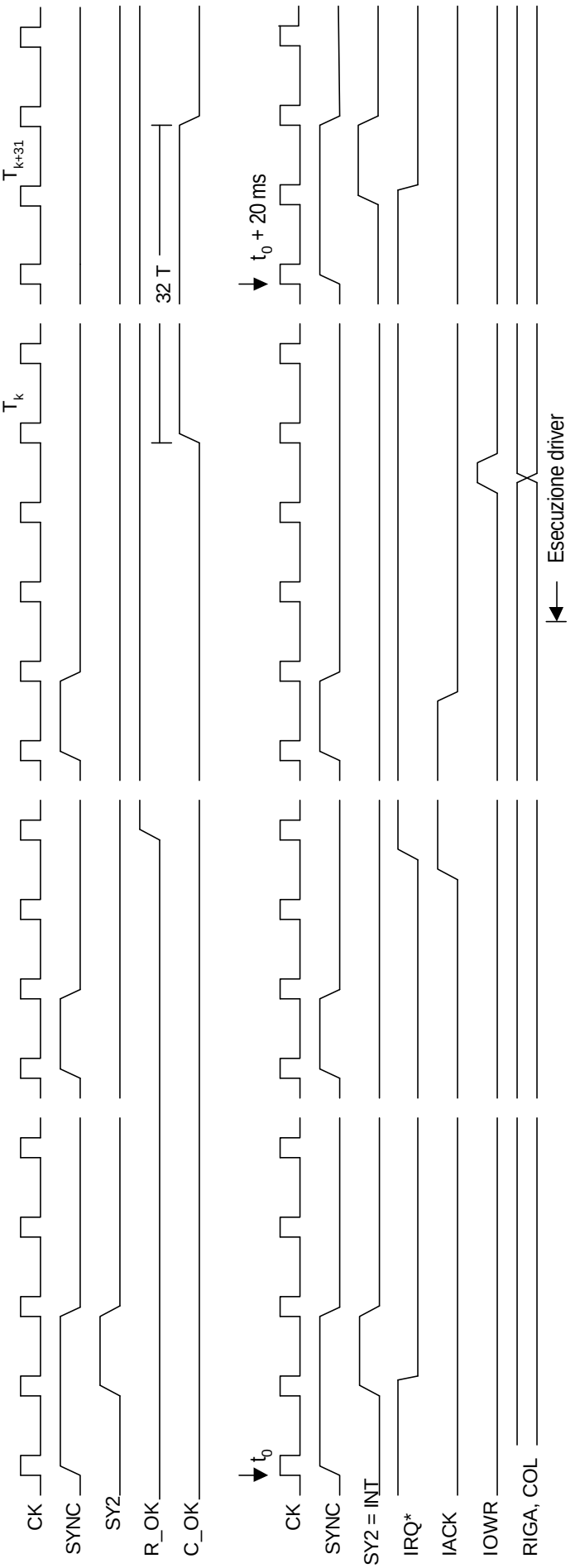
LOGO: ROM pipeline



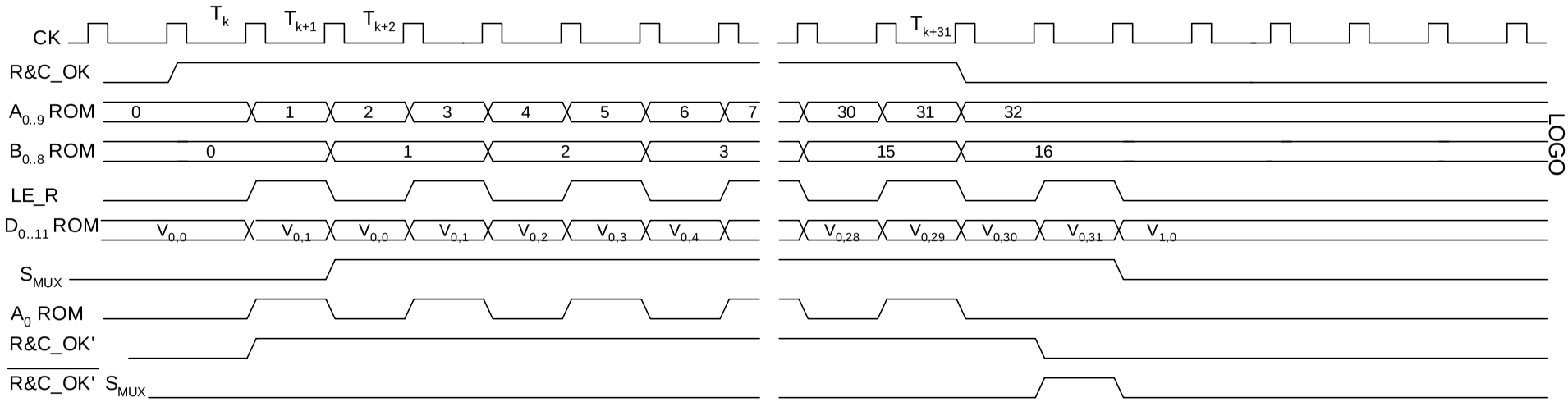
Allocazione dei campioni
del logo nella ROM



TLVSYNC: temporizzazioni



TLVSYNC: temporizzazioni

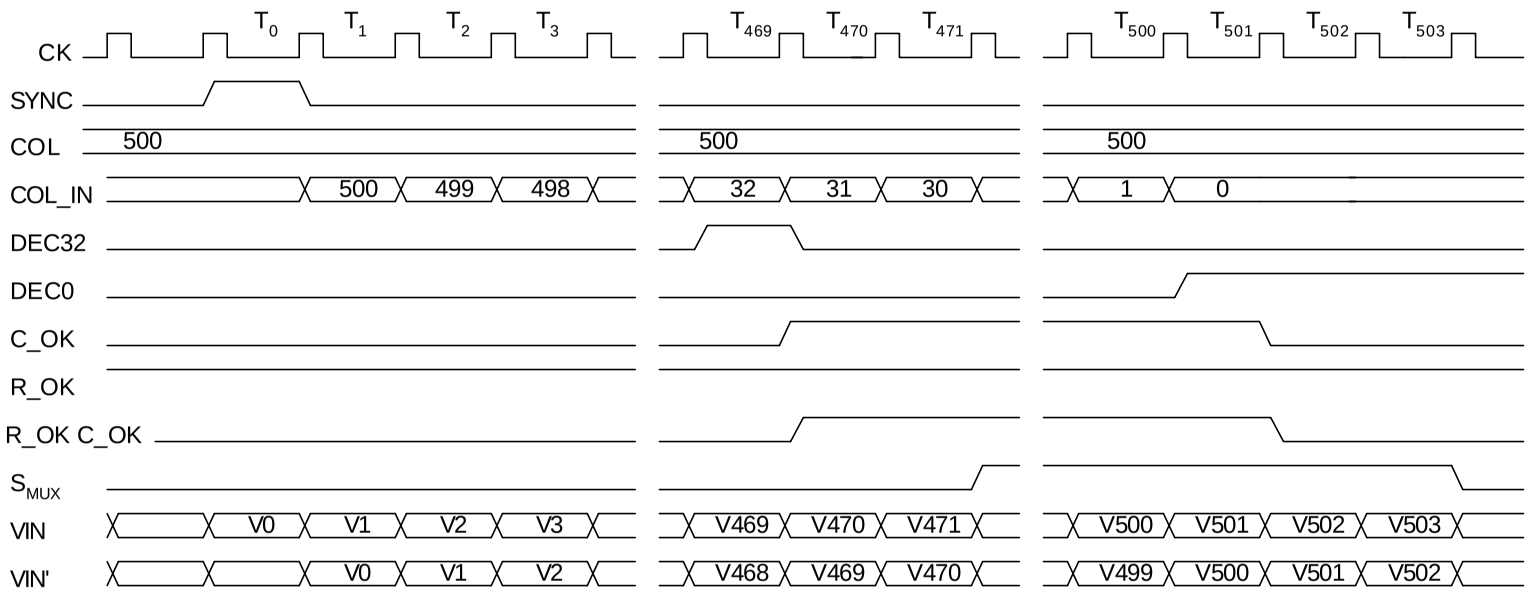


$S_{MUX} = (R\&C\_0K')'$

$LE\_R = A_0 + S_{MUX}\ \overline{R\&C\_OK'}$

LOGO

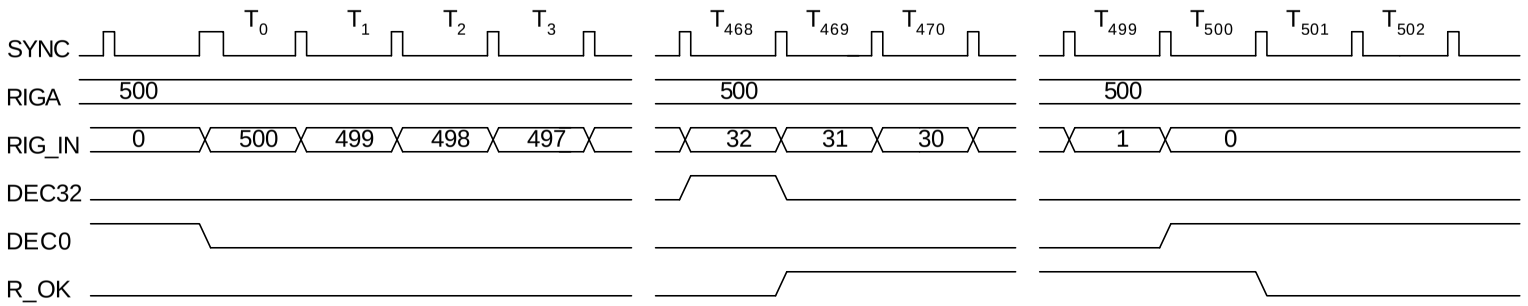
TLVSYNC: temporizzazioni



Ipotesi di lavoro:
- il processore ha caricato COL = 500;
- il primo campione video ha indice 0.

Il diagramma temporale evidenzia che:
- S_{MUX} devia sull'uscita i campioni del logo al posto dei campioni video (VIN') 471..502. Pertanto, nel manuale d'uso della periferica dovrà essere riportato che il software dovrà caricare COL = N-2 per piazzare il logo con l'ordinata del vertice Sud-Est pari a N.
- Il percorso dati VIN-VOUT include due registri, pertanto la latenza VIN-VOUT è pari a 2T_{CK}.

TLVSYNC: temporizzazioni



Ipotesi di lavoro:
- il processore ha caricato RIGA = 500;
- la prima riga video ha indice 0.

Il diagramma temporale evidenzia che:
- R_OK si attiva nelle righe 469..500. Pertanto, nel manuale d'uso della periferica dovrà essere riportato che il software dovrà caricare RIGA = N per piazzare il logo con l'ascissa del vertice Sud-Est pari a N.

Commenti al progetto

- I segnali entranti VIN e SYNC non sono stati sincronizzati in quanto nel testo viene specificato che sono sincroni con CK; in pratica si suppone che tali segnali siano resi disponibili da un circuito pre-elaboratore, che li estrae dal segnale video sorgente.

- Sezione inseritore: i registri che tamponano VIN e VOUT hanno una funzione di pipeline: servono soltanto a ridurre il tempo di transito dei segnali da e verso i circuiti esterni; infatti, detti:
  - t_{in} il tempo di propagazione lungo i fili di connessione dai terminali di ingresso alle porte logiche di un circuito integrato,
  - t_{mux} il tempo di commutazione del MUX,
  - t_{out} il tempo di propagazione lungo i fili di connessione dalle porte logiche ai terminali di uscita di un circuito integrato,
  - t_{setup} il tempo di setup dei registri,
  - t_{reg} il tempo di commutazione dei registri,

  se si fa l'ipotesi che Vin è fornito da un circuito esterno A e VOUT viene trasferito a un circuito esterno B, con A e B sincroni con la periferica LOGO, allora il tempo di transito dal registro di uscita di A al registro di ingresso di B è, in assenza dei due registri tampone interni alla perifewrica LOGO:

  $$T > t_{reg} + t_{out} + t_{in} + t_{mux} + t_{out} + t_{in} + t_{setup} = t_{reg} + t_{setup} + 2\,(t_{out} + t_{in}) + t_{mux}$$

  Con i due registri inseriti il vincolo temporale si riduce a:

  $$T > \max\,(t_{reg} + t_{out} + t_{in} + t_{setup},\, t_{reg} + t_{mux} + t_{setup},\, t_{reg} + t_{out} + t_{in} + t_{setup}) =$$
  $$= t_{reg} + t_{setup} + \max\,(t_{out} + t_{in},\, t_{mux})$$

  Per avere questo vantaggio tale accorgimento viene normalmente utilizzato nei circuiti integrati, in previsione del collegamento con altri moduli sincroni (*chip-set*).

- Banco ROM: va notato che la memoria ROM non è stata replicata come nelle strutture pipeline generali; è stata invece suddivisa in due banchi di metà capacità, impegnando la capacità globale della ROM strettamente necessaria da un punto di vista logico, senza considerare il problema del tempo di accesso ($T<t_A<2T$). Questa semplificazione è possibile perché in questo caso particolare l'accesso alla ROM è di tipo *sequenziale, e non casuale*; infatti, i campioni del logo devono essere prelevati secondo un ordinamento precostituito, e quindi possono essere estratti a coppie ($t_A<2T$) con periodo 2T. Generalizzando, se fosse stato $(n-1)T<t_A<nT$ si sarebbe potuto estrarre n campioni simultaneamente (da consumare in un tempo di durata nT). Pertanto i campioni sono stati precaricati alternativamente in due moduli ROM disposti *in parallelo* (condividono le linee di indirizzo) secondo lo schema riportato nel disegno.

- Va notato che la funzione di caricamento dei registri di uscita del banco di ROM è stata calcolata sulla base del diagramma di temporizzazione.

- I due registri di uscita del banco di ROM si caricano *insieme* quando LE_R=1; tuttavia l'uscita comune assume il valore dello stato dei due registri alternativamente, al ritmo di LE_R.

Reti Logiche

Seconda prova scritta del 1-2-2000

Studente: _____ Docente:_____

D1 Progettare una rete con due ingressi x_1 e x_2 e due uscite y_1 e y_2: $y_1 = x_2$ e $y_2 = x_1$ se $x_1 <= x_2$, altrimenti $y_1 = x_1$ e $y_2 = x_2$

D2 Descrivere, attraverso una temporizzazione, le condizioni di instabilità di un FF SR costituito da due porte NOR. Definire in che circostanza tale circuito può comportarsi come oscillatore.

D3 Minimizzare la seguente macchina sequenziale di Mealy:

	Stato Succ/uscita	
A	C/1	F/1
B	C/0	D/0
C	D/1	E/0
D	G/0	B/0
E	G/1	A/1
F	C/1	E/1
G	B/1	A/0

D4 Illustrare la temporizzazioni di un sistema SCO-SCA di tipo DMealy-Dmealy e di uno Mealy-Dmealy e descrivere le differenze sostanziali.

D5 Un convertitore A/D ha due segnali START_ACQUISITION (che attiva una conversione) e END_ACQUISITION (la conversione corrente è terminata), campiona con una risoluzione migliore dello 0,5% e, approssimativamente, il tempo di conversione è nell'ordine dei msec mentre il periodo di clock del PD32 è nell'ordine dei nano secondi. Il risultato della conversione viene restituito all'esterno dal convertitore AD attraverso un registro. Definire l'interfaccia PD32-convertitore e scrivere un programma PD32 che prelevi 256 dati dal convertitore A/D e ne faccia la media.

Esercizio (2S20000201-D1)

Progettare una rete con due ingressi x_1 e x_2 e due uscite y_1 e y_2: $y_1 = x_2$ e $y_2 = x_1$ se $x_1 <= x_2$, altrimenti $y_1 = x_1$ e $y_2 = x_2$

Traduzione delle specifiche nella tavola di verità:

si richiede di confrontare i valori numerici assunti dalle variabili indipendenti $x_1 <= x_2$ e quindi definire i valori di y_1 e y_2:

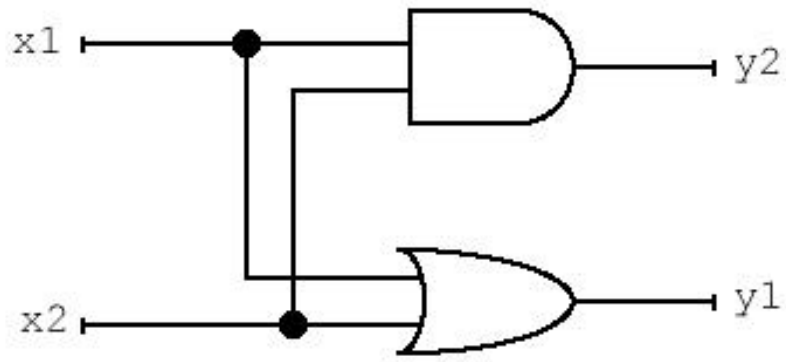
x_2	x_1	y_2	y_1
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

La sintesi è banale e si può risolvere per ispezione visiva:

$y_1 = x_2 + x_1$

$y_2 = x_2 \bullet x_1$

La traduzione nella rete logica corrispondente è immediata:



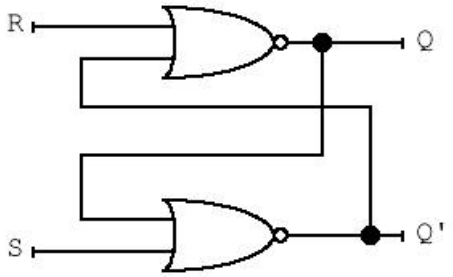
Esercizio (2S20000201-D2)

Descrivere, attraverso una temporizzazione, le condizioni di instabilità di un FF SR costituito da due porte NOR. Definire in che circostanza tale circuito può comportarsi come oscillatore.

Il flip-flop (o latch) Set-Reset, riportato nella figura, è un dispositivo asincrono, progettato per operare in modo fondamentale: il funzionamento corretto è garantito se e solo se vengono rispettate le due condizioni:

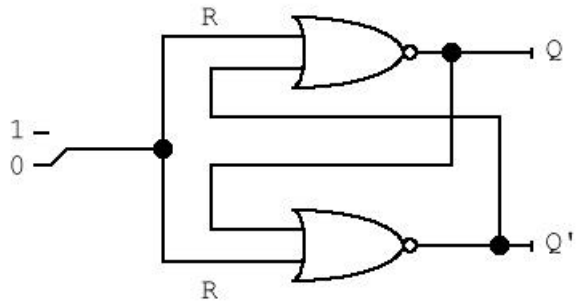
1: gli ingressi commutano uno alla volta;
2: un ingresso commuta se e solo se la rete è stabile.

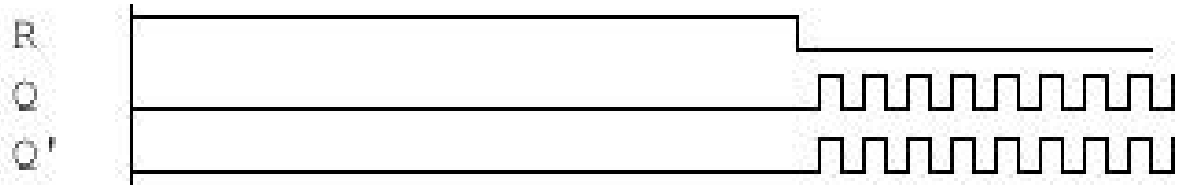
La condizione 2 implica per il latch un tempo minimo tra due commutazioni consecutive pari alla somma dei tempi di commutazione delle due porte (che sono disposte in cascata). Assumendo un tempo di commutazione t_p uguale per le due porte, l'intervallo minimo tra due variazioni consecutive dovrà essere almeno pari a $2 \bullet t_p$.



Un pilotaggio che infrange entrambe le regole è quello rappresentato nella figura seguente, con riferimento al diagramma temporale in cui sono riportate le sollecitazioni e le risposte del latch. Va notato che:

1: entrambe le uscite oscillano;
2: le due uscite oscillano in fase;
3: il periodo di oscillazione è pari a $2 \bullet t_p$.





Esercizio (2S20000201-D3)

Minimizzare la seguente macchina sequenziale di Mealy:

	Stato Succ. / Uscita	
A	C/1	F/1
B	C/0	D/0
C	D/1	E/0
D	G/0	B/0
E	G/1	A/1
F	C/1	E/1
G	B/1	A/0

---

1 – Distinguibilità degli stati a causa della diversità delle uscite

A						
X	B					
X	X	C				
X		X	D			
	X	X	X	E		
	X	X	X		F	
X	X		X	X	X	G

2 – Trascrizione delle coppie degli stati successivi

A						
X	B					
X	X	C				
X	C,G	X	D			
C,G A,F	X	X	X	E		
E,F	X	X	X	C,G A,E	F	
X	X	B,D A,E	X	X	X	G

3 – Distinguibilità tra le coppie di stati successivi

A						
X	B					
X	X	C				
X	C,G	X	D			
C,G A,F	X	X	X	E		
E,F	X	X	X	C,G A,E	F	
X	X	B,D A,E	X	X	X	G

La tabella 3 non è diversa dalla tabella 2: solo a questo punto l'algoritmo può procedere con l'individuazione delle classi di equivalenza a partire dalle coppie di stati che non sono distinguibili, cioè che sono equivalenti.

Coppie di stati equivalenti:

A,E A,F C,G A,F B,D E,F

Classi di equivalenza:

A,E,F B,D C,G

Tutti gli stati originali sono stati inclusi nelle classi di equivalenza.

Detti M=(A,E,F), P=(B,D), R=(C,G) gli stati della macchina minima, questa può essere descritta mediante la tavola di flusso seguente:

	Stato Succ. / Uscita	
M	R/1	M/1
P	R/0	P/0
R	P/1	M/0

Esercizio (2S20000201-D4)

Illustrare la temporizzazioni di un sistema SCO-SCA di tipo DMealy-Dmealy e di uno Mealy-Dmealy e descrivere le differenze sostanziali.

Cfr. il testo “Reti Sequenziali” e gli “Appunti Integrativi”.

Esercizio (2S20000201-D5)

Un convertitore A/D ha due segnali START_ACQUISITION (che attiva una conversione) e END_ACQUISITION (la conversione corrente è terminata), campiona con una risoluzione migliore dello 0,5% e, approssimativamente, il tempo di conversione è nell'ordine dei msec mentre il periodo di clock del PD32 è nell'ordine dei nano secondi. Il risultato della conversione viene restituito all'esterno dal convertitore AD attraverso un registro. Definire l'interfaccia PD32-convertitore e scrivere un programma PD32 che prelevi 256 dati dal convertitore A/D e ne faccia la media.

Considerazioni preliminari

- Una risoluzione migliore dello 0.5% significa poter operare con più di 200 livelli di quantizzazione (1/200 = 0.005); ciò comporta la disponibilità di un convertitore ADC con (almeno) 8 bit (256 livelli); pertanto si supporrà di operare sul byte.

- Il rapporto tra le velocità operative del processore e della periferica (tre ordini di grandezza) suggerisce l'opportunità del trasferimento dati mediante interruzione. Pertanto l'interfaccia hardware della periferica è senz'altro quella canonica per le interruzioni come da manuale PD32.

- Il segnale START_ACQUISITION potrà essere prelevato dall'uscita del flip-flop di handshake della periferica, settato dalla lettura del dato da parte del micro e il segnale END_ACQUISITION emesso dal convertitore potrà resettare lo stesso flip-flop.

```
    org 400h

; *************************
; COSTANTI
; *************************

    STACK    equ 2800h    ;inizio area di stack
                          ;limitato a 2800h per consentire la simulazione
    input    equ 012h     ;indirizzo periferica input
                          ;INPUT:    I/O=I, ind=12h, IVN=10


;**************************
; VARIABILI
;**************************

    buff db 0             ;buffer di appoggio del dato prelevato in input
    flag db 0             ;variabile binaria di handshake (dato pronto)
    count    db 0         ;contatore dei dati prelevati
    media    dw 0         ;locazione di destinazione del valor medio
    convert db 1          ;variabile binaria a 1 nel ciclo di 256 acquisizioni
```

```
; *************************
; CODICE
; *************************
    code                    ;inizio istruzioni

main:
    movl #STACK,r7          ;inizializza R7 quale SP
    seti                    ;abilita interruzioni (SP è stato inizializzato)


    inb input,buff          ;lettura dummy per attivare la prima conversione


mainloop:
    movb convert,r0
    andb r0,r0
    jz skiploop


    movb flag,r0            ;test sulla produzione di un byte del device
    andb r0,r0
    jz skiploop
    movb #0,flag            ;resetta la locazione flag (dato consumato)
    inb input,buff          ;legge il dato e attiva la conversione successiva

;segue trattamento del byte della periferica

    xorw r0,r0
    mvlb buff,r0            ;somma buff a media
    addw media,r0
    movw r0,media

;aggiornamento del contatore dei dati acquisiti

    movb count,r0
    cmpb #255,r0
    jnz incount

    movw media,r0
    lsrw #8,r0              ;dividi l'accumulatore per 256
    movw r0,MEDIA           ;scrivi la media aritmetica in MEDIA

    movb #0,convert         ;termina le conversioni
    jsr userproc            ;chiama il sottoprogramma utilizzatore
    jmp skiploop            ;procedi nel ciclo esterno

incount:
    addb #1,r0
    movb r0,count
    jmp skiploop
```

```
skiploop:
;spazio per altri segmenti del programma



;al termine dei quali è prevista la chiusura del loop su mainloop
    jmp mainloop


;**********************************
;SUBROUTINE
;**********************************

;userproc è il codice utilizzatore del risultato
;potrà opzionalmente settare la variabile convert

userproc:
    ret


;**********************************
;DRIVER DI INPUT
;**********************************

;Setta la flag di dato pronto


    driver 10, 800h         ;Il driver della periferica con IVN=10
                            ;inizia dall'ind. 800h
    movb #1,flag            ;setta la locazione flag (dato pronto)
    rti                     ;ritorno da interruzione



    end                     ;fine programma
```

```
;Un convertitore A/D ha due segnali START_ACQUISITION
;(che attiva una conversione) e END_ACQUISITION
;(la conversione corrente è terminata), campiona con una
;risoluzione migliore dello 0,5% e, approssimativamente,
;il tempo di conversione è nell'ordine dei msec mentre il
;periodo di clock del PD32 è nell'ordine dei nano secondi.
;Il risultato della conversione viene restituito all'esterno
;dal convertitore AD attraverso un registro. Definire
;l'interfaccia PD32-convertitore e scrivere un programma
;PD32 che prelevi 256 dati dal convertitore A/D e ne faccia la media.


;Considerazioni preliminari
;Una risoluzione migliore dello 0.5% significa poter operare con più
;di 200 livelli di quantizzazione (1/200 = 0.005); ciò comporta la
;disponibilità di un convertitore ADC con (almeno) 8 bit (256 livelli);
;pertanto si supporrà di operare sul byte.

;Il rapporto tra le velocità operative del processore e della
;periferica (tre ordini di grandezza) suggerisce l'opportunità
;del trasferimento dati mediante interruzione. Pertanto
;l'interfaccia hardware della periferica è senz'altro quella
;canonica per le interruzioni come da manuale PD32.

;Il segnale START_ACQUISITION potrà essere prelevato dall'uscita
;del flip-flop di handshake della periferica, settato dalla
;lettura del dato da parte del micro e il segnale END_ACQUISITION
;emesso dal convertitore potrà resettare lo stesso flip-flop.


  org 400h

; *************************
; COSTANTI
; *************************

  STACK   equ 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione
  input equ 012h  ;indirizzo periferica input
        ;INPUT: I/O=I, ind=12h, IVN=10



;*************************
; VARIABILI
;*************************

  buff  db 0    ;buffer di appoggio del dato prelevato in input
  flag  db 0    ;variabile binaria di handshake (dato pronto)
  count db 0    ;contatore dei dati prelevati
  media dw 0    ;locazione di destinazione del valor medio
  convert db 1    ;variabile binaria a 1 nel ciclo di 256 acquisizioni

; *************************
; CODICE
; *************************
  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)
```

```
  inb input,buff    ;lettura dummy per attivare la prima conversione


mainloop:
  movb convert,r0
  andb r0,r0
  jz skiploop


  movb flag,r0    ;test sulla produzione di un byte del device
  andb r0,r0
  jz skiploop
  movb #0,flag    ;resetta la locazione flag (dato consumato)
  inb input,buff    ;legge il dato e attiva la conversione successiva

;segue trattamento del byte della periferica

  xorw r0,r0
  mvlb buff,r0    ;somma buff a media
  addw media,r0
  movw r0,media

;aggiornamento del contatore dei dati acquisiti

  movb count,r0
  cmpb #255,r0
  jnz incount

  movw media,r0
  lsrw #8,r0    ;dividi l'accumulatore per 256
  movw r0,MEDIA   ;scrivi la media aritmetica in MEDIA

  movb #0,convert   ;termina le conversioni
  jsr userproc    ;chiama il sottoprogramma utilizzatore
  jmp skiploop    ;procedi nel ciclo esterno

incount:
  addb #1,r0
  movb r0,count
  jmp skiploop


skiploop:
;spazio per altri segmenti del programma




;al termine dei quali è prevista la chiusura del loop su mainloop
  jmp mainloop


;*********************************
;SUBROUTINE
;*********************************

;userproc è il codice utilizzatore del risultato
;potrà opzionalmente settare la variabile convert

userproc:
  ret
```

```
;*********************************
;DRIVER DI INPUT
;*********************************

;Setta la flag di dato pronto


  driver 10, 800h   ;Il driver della periferica con IVN=10
        ;inizia dall'ind. 800h
  movb #1,flag    ;setta la locazione flag (dato pronto)
  rti     ;ritorno da interruzione



  end     ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 18-02-2000

Studente:_____ Docente:_____

****

Un agricoltore deve irrigare il proprio terreno attraverso un sistema di innaffiamento automatico (SIA). Il terreno è diviso in 256 particelle ognuna delle quali possiede una unità automatica di innaffiamento (UAI) costituita da un *sensore analogico* di umidità e da un irrigatore controllato da una elettrovalvola che comanda l'avvio o l'arresto dell'irrigazione. La generica elettrovalvola ha un ingresso di comando binario per essere aperta o chiusa.
Le 256 UAI sono controllate da una periferica, chiamata SIA, connessa ad un PD32, e fornita di un clock interno di frequenza 100Hz. SIA viene avviata da un comando di START, inviato dal PD32, e comincia a rilevare ciclicamente i valori dell'umidità dalle UAI. **Per ogni valore analogico** di umidità rilevato da una UAI, la periferica SIA:

1. Provvede alla conversione attraverso *un convertitore A/D*, con una precisione pari all'1%. Tale convertitore viene attivato da un segnale di START-ACQUISITION e segnala la fine della conversione all'esterno tramite un segnale di END-CONVERSION.
2. Invia un INTERRUPT al PD32
3. Riceve dal PD32 i comandi da inviare al Flip-Flop della UAI in questione per avviare o arrestare il funzionamento della pompa

Il PD32 possiede in memoria una tabella (a partire dall'indirizzo SOGLIA) dove sono contenute le informazioni relative alle soglie di umidità minima (Smin)e massima (Smax) di ciascuna UAI.
All'interno della routine di interruzione il PD32:

1. Preleva dalla SIA l'identificatore della UAI (id) ed il valore corrente dell'umidità (U-id).
2. Confronta il valore dell'umidità con le soglie in memoria ed invia un comando di ATTIVA/DISATTIVA al SIA per l'elettrovalvola della UAI (id) secondo la seguente tabella

ATTIVA	DISATTIVA	
1	**0**	(U-id < Smin-id)
0	***1***	(U-id > Smax-id)
0	***0***	Smin-id U-id Smax-id

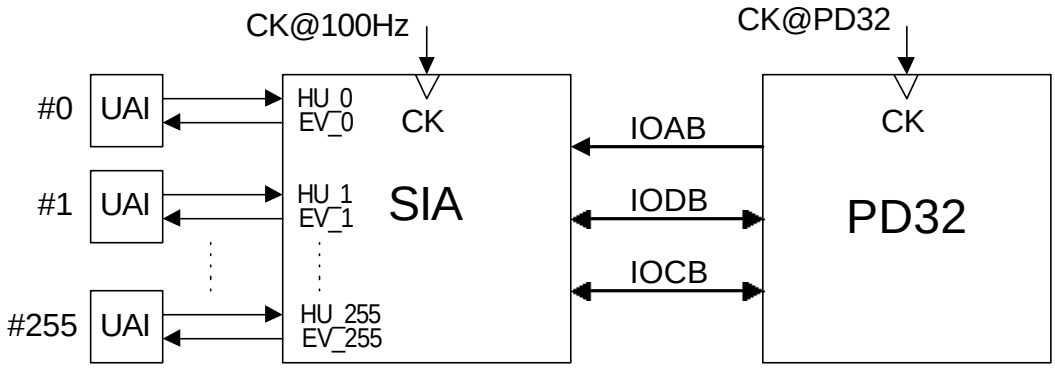
All'interno della SIA si dovranno utilizzare: 1 convertitore A/D, 17 MUX analogici (4/16) e 17 DEMUX digitali per il controllo delle 256 elettrovalvole.

Si richiede:

- lo schema logico ed elettrico dettagliato della SIA e della generica UAI.
- Il calcolo del periodo di scansione della SIA, sapendo che il tempo di conversione è di 20msec e il tempo di servizio della routine di interruzione è di 15 msec.
- la temporizzazione della SIA.

La routine di interruzione del PD32.

IRRIGATORE: sistema esterno



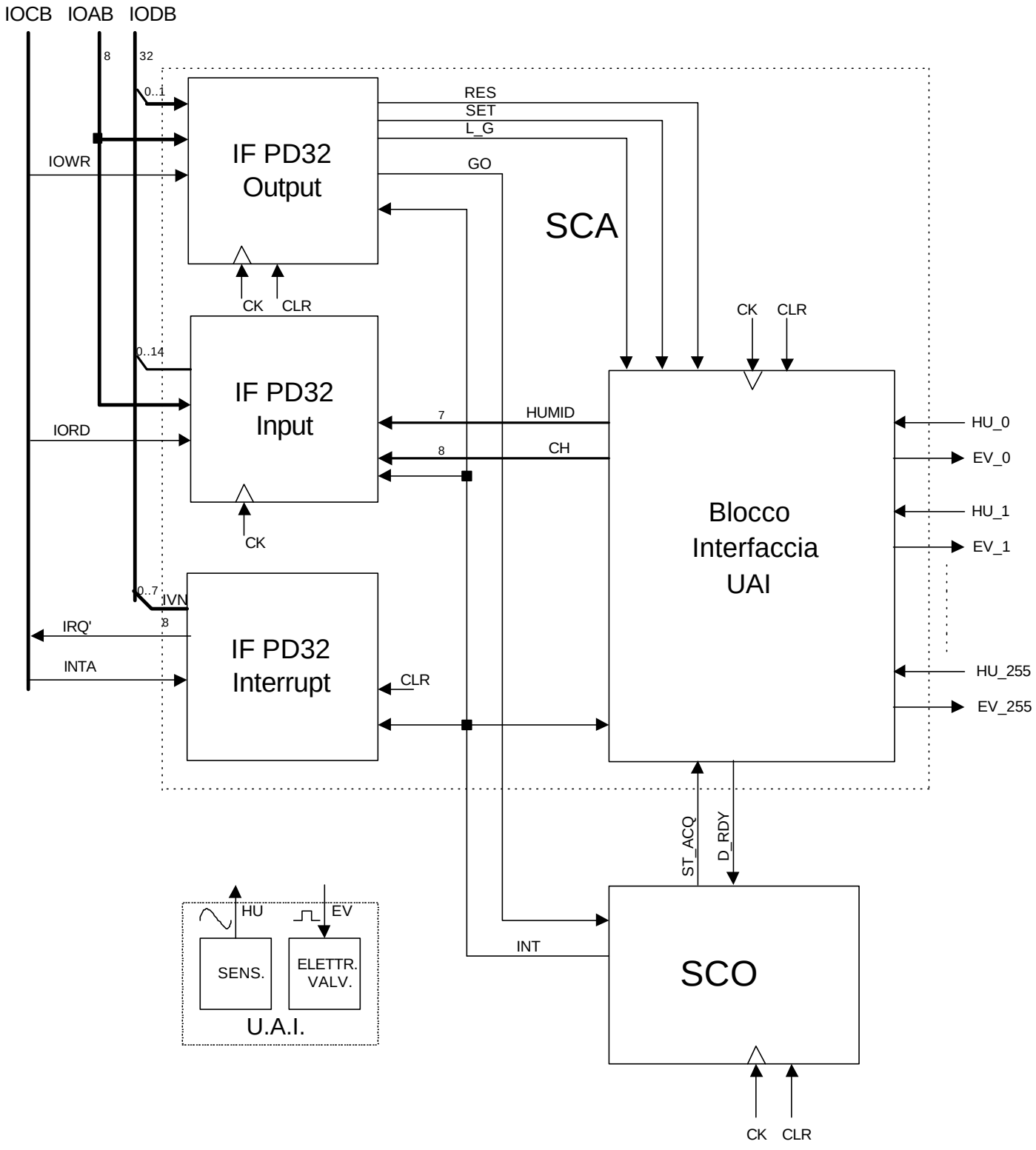
Note

Periferica di tipo input/output.

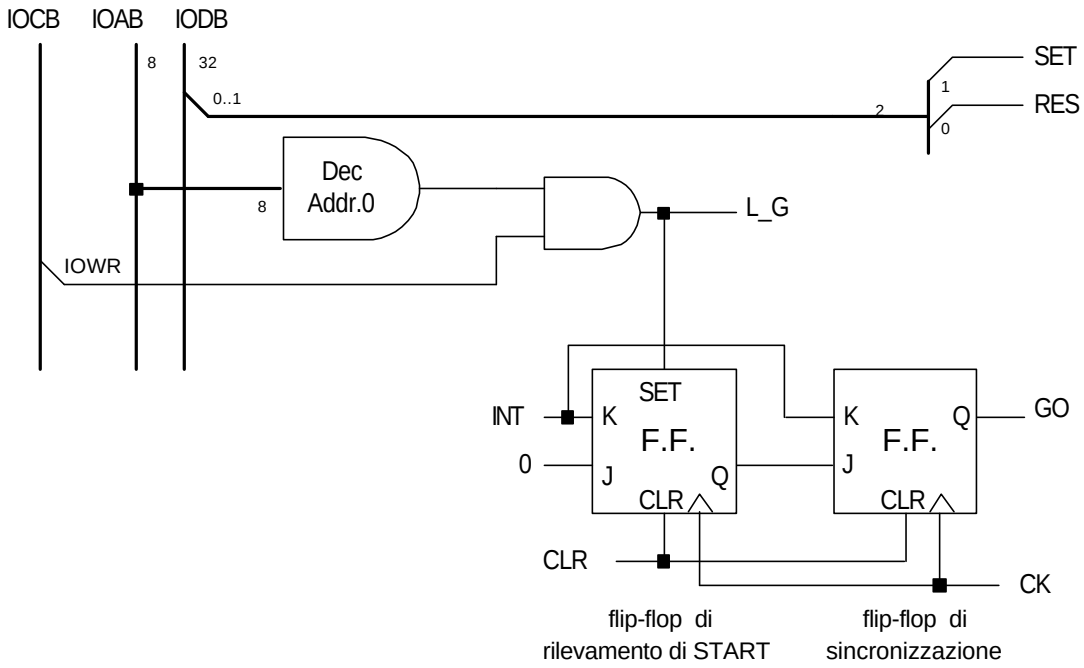
Il clock della SIA (appena 100 Hz) è molto più lento del clock del PD32 (tipicamente dell'ordine delle decine di MHz); pertanto la comunicazione da SIA a PD32 verrà attivata con il meccanismo dell'interruzione.

Ciascuna delle 256 UAI invia un segnale analogico (HU: misura il livello di umidità) a SIA e riceve da questa un unico segnale binario (EV: attiva l'elettrovalvola se e solo se vale 1).

IRRIGATORE: Schema a blocchi SIA



IRRIGATORE: IF PD32 - output



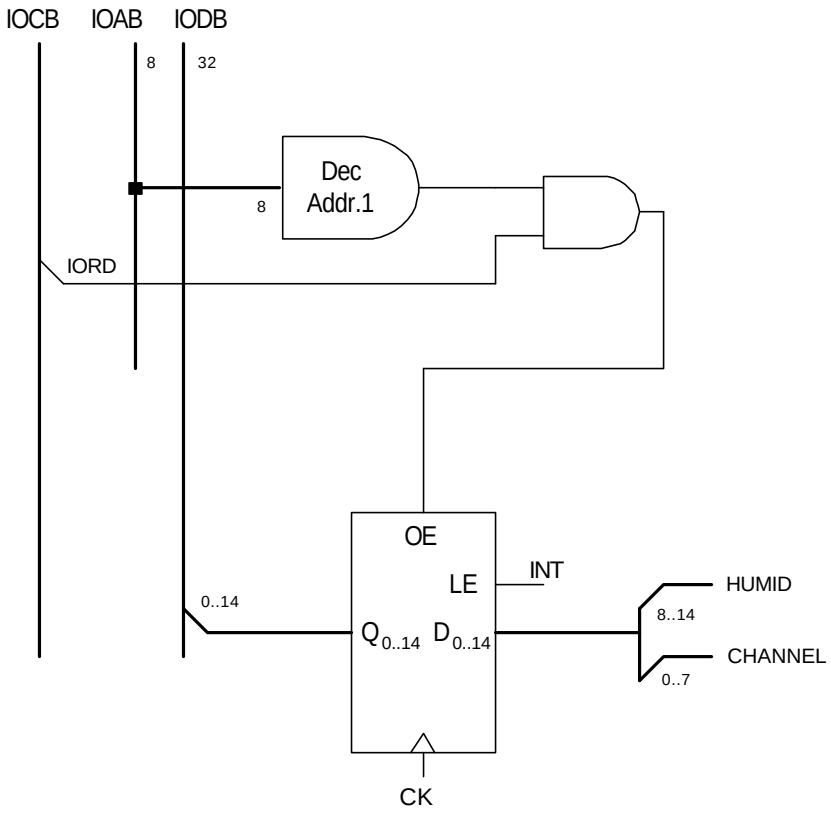
Note

Il SW avvia la periferica SIA con l'istruzione **OUTB #00,Addr0** o con qualunque altro codice di indirizzo per l'operando sorgente (p.e. OUTB R0,Addr0), purché questo sia nullo : infatti, appena attivato lo SCO di SIA sarà in uno stato iniziale di attesa del bit GO=1 e in tale stato le elettrovalvole devono essere mantenute nel loro stato iniziale di disattivazione (tutti i latch di pilotaggio azzerati dal CLR iniziale). Successivamente, dopo avere ricevuto la richiesta di interruzione, il SW aggiorna i due bit SET e RES diretti ai latch mediante l'istruzione**OUTB R0,Addr0** in cui si è supposto di avere precaricato R0 con i valori appropriati (cfr. routine software).

I valori SET e RES non vengono memorizzati nell'interfaccia, ma vengono portati in ingresso ai latch di pilotaggio delle UAI. Il segnale IOWR decodificato (Addr0 nella figura) verrà collegato ai gate dei latch tramite la decodifica dell'indirizzo (a 8 bit) del latch della UAI.

Il periodo di CK (10 ms) è molto maggiore del periodo del clock (tipicamente centinaia o anche decine di nanosecondi) del processore; per questo motivo il flip-flop di interfaccia è pilotato sul fronte (di CK) dalla periferica e sull'ingresso asincrono (di set) dal micro; infatti, se le connessioni fossero invertite la periferica bloccherebbe il flip-flop sull'ingresso asincrono per ben 10 ms, impedendo in tale intervallo al micro di modificare lo stato del flip-flop in quanto l'eventuale impulso sull'ingresso di trigger del flip-flop non avrebbe effetto.

IRRIGATORE: IF PD32 - input

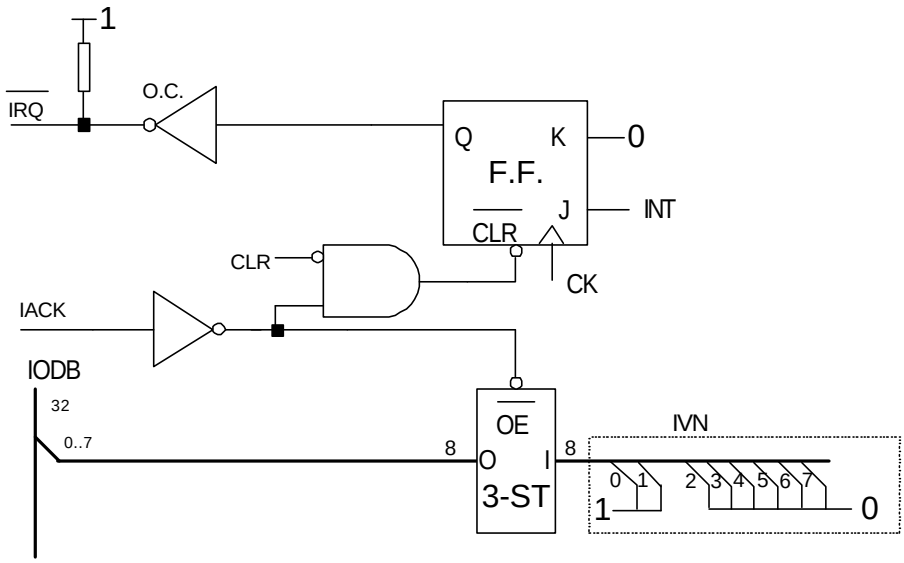


Note

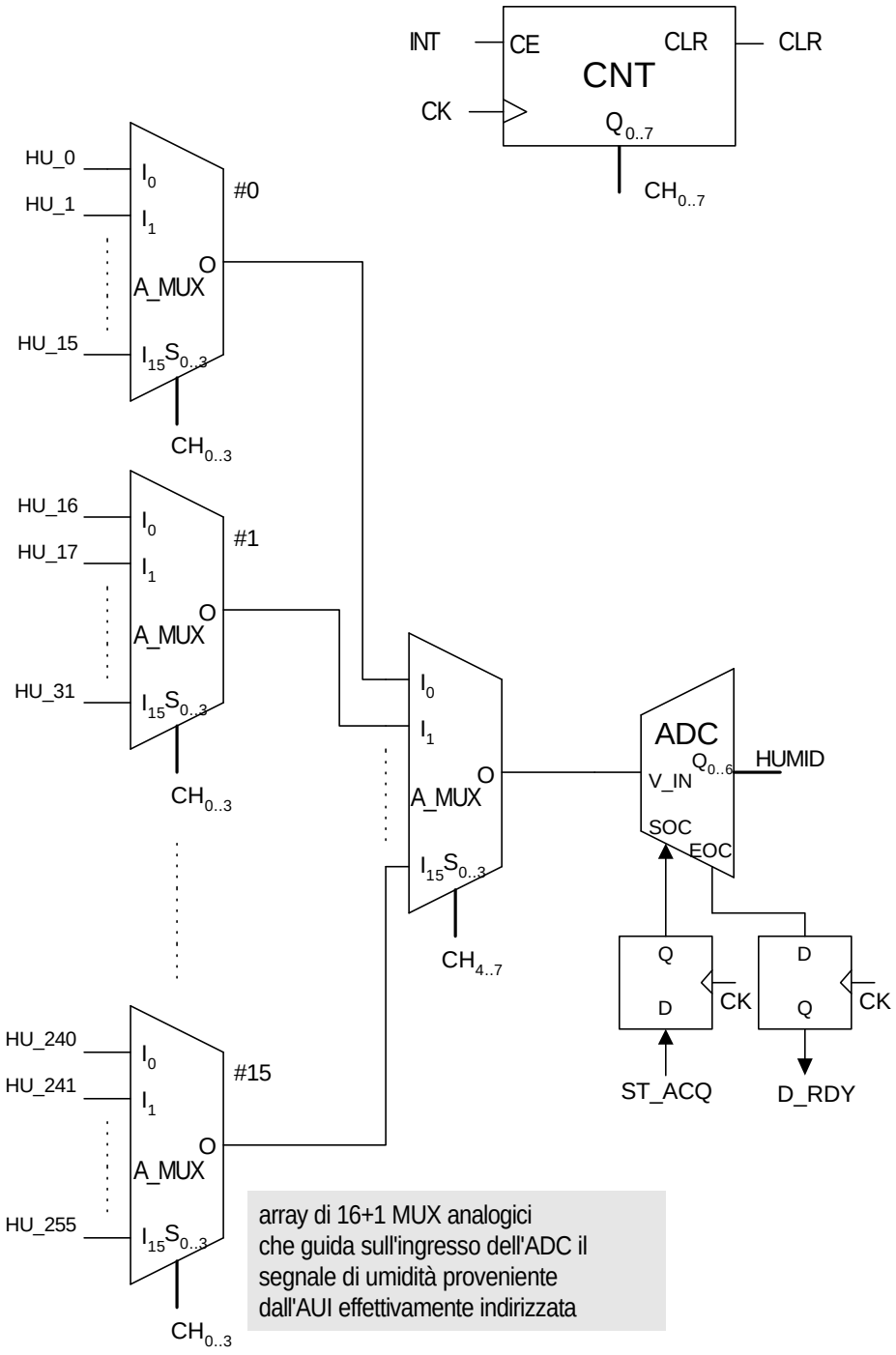
Il SW preleva il codice dell'unità UAI e del relativo livello di umidità mediante l'istruzione **INW Addr1,R1** o con qualunque altro codice di indirizzo per l'operando destinazione (es. direttamente in memoria).

Il meccanismo dell'interruzione provvede a sincronizzare la produzione e il consumo dei dati scambiati; pertanto, non è necessario un flip-flop di semaforo anche in questa interfaccia.

IRRIGATORE: IF PD32 - interrupt



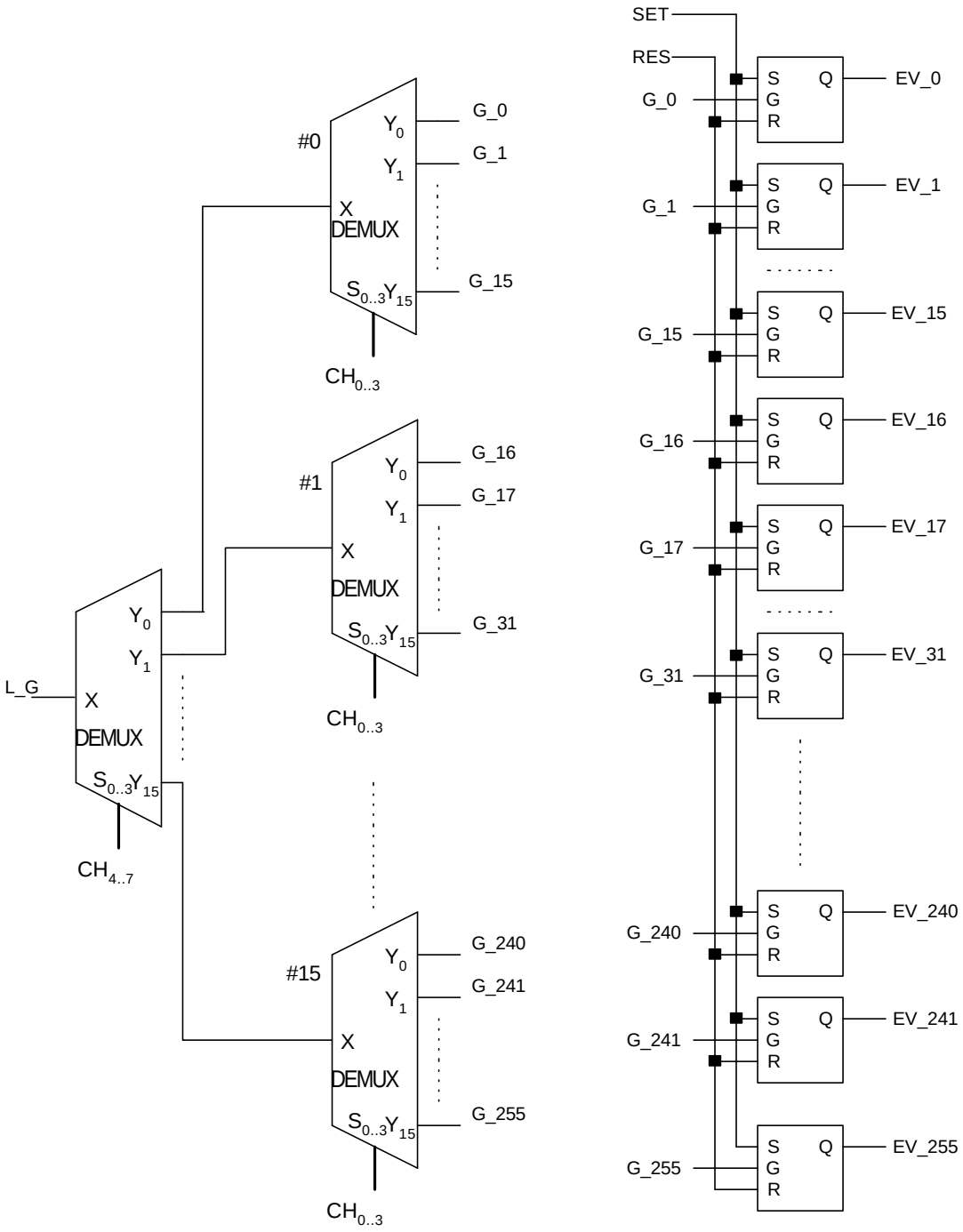
IRRIGATORE: blocco interfaccia UAI - 1



Note

Il testo specifica per il convertitore analogico-digitale ADC una precisione migliore dell'1%, che si traduce in una larghezza di almeno 7 bit per il dato di uscita ($1/2^7 = 1/128 < 0.01 < 1/64 = 1/2^6$).

IRRIGATORE: blocco interfaccia UAI - 2



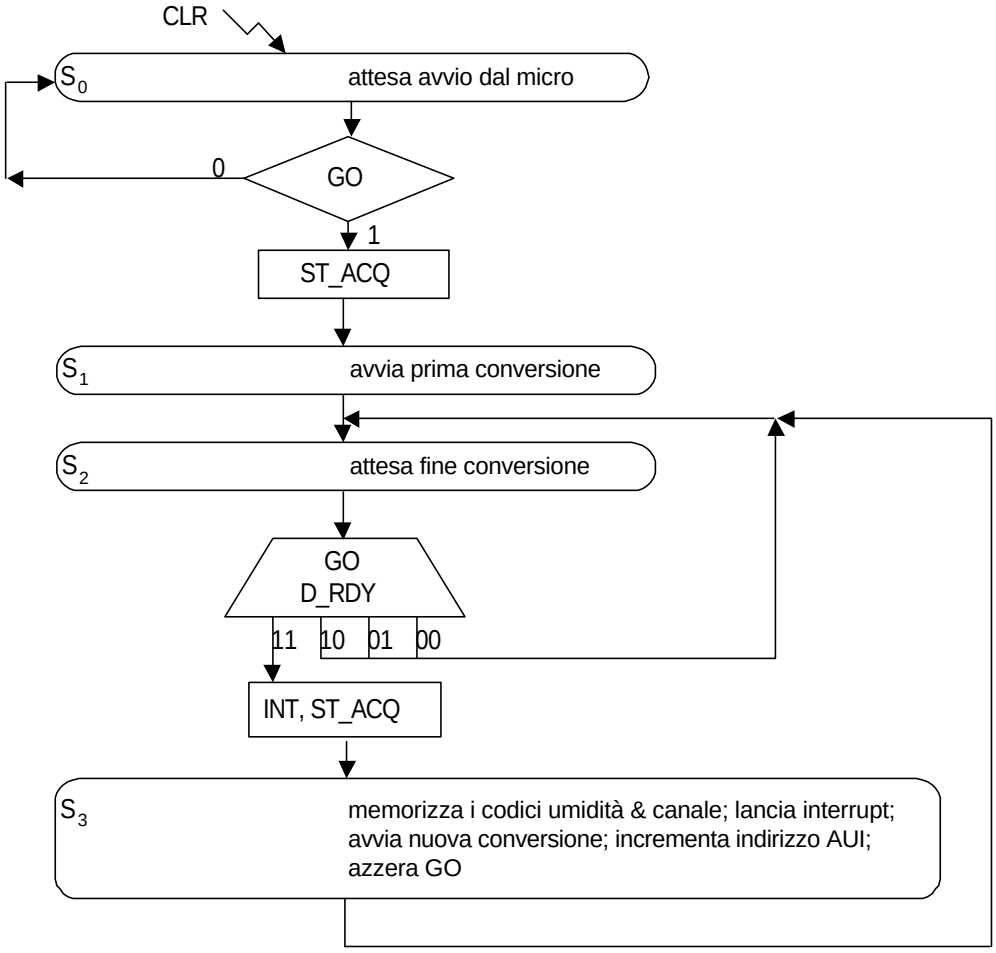
array di 1+16 DEMUX
che guida l'impulso di scrittura sul (gate del) latch effettivamente indirizzato e solo su quello

array di 256 latch SR
di tipo sincrono (dotati di ingresso di gate)

Note

Il bit L_G viene attivato da IOWR, durante l'esecuzione di un'istruzione OUTB R0,Addr0 del PD32; pertanto si assume che i codici di selezione dei DEMUX CH$_{0..7}$ siano aggiornati dal contatore interno alla SIA quando IOWR=0.
Va notato che quando L_G=0 tutte le uscite di tutti i DEMUX sono bloccate sul valore logico 0 e pertanto in quanto prive di impulsi spuri (spike) non possono provocare caricamenti indesiderati nel latch.

IRRIGATORE: SCO - flowchart



Note

Formalmente il flow-chart è stato impostato secondo il modello di Mealy; tuttavia occorre notare che i due segnali di uscita INT e ST_ACQ sono diretti verso dispositivi esterni e asincroni (il micro e il convertitore ADC rispettivamente) e perciò vanno registrati: a tale scopo sono stati predisposti due flip-flop che ne risincronizzano le variazioni con CK. Il flip-flop di interruzione è inglobato nell'interfaccia di interrupt del micro, il secondo è nel blocco di interfaccia verso le UAI esterne, assieme all'ADC. Strutturalmente la coppia di flip-flop può anche essere vista come il registro di uscita di uno SCO trasformato nel tipo D-Mealy.

Non è prevista una terminazione delle operazioni: il conteggio e l'indirizzamento delle UAI procedono ciclicamente.

Il calcolo del periodo di scansione è riportato nei commenti al diagramma di temporizzazione.

Poiché i tempi di risposta sia del micro che dell'ADC sono noti, come ovviamente il periodo di CK, i test su D_RDY e GO potrebbero essere omessi. Tuttavia, in una realizzazione reale sarebbe consigliabile inserire i test per rendere il microprogramma robusto nei confronti di eventuali variazioni dei tempi di risposta dovute a sviluppi successivi in particolare del programma applicativo del micro.

IRRIGATORE: SCO - struttura HW

- di tipo Mealy: segnali verso il processore e l'ADC registrati esternamente
- a sequenziatore: semplicità del diagramma degli stati

Calcolo dei parametri del circuito

· Equazione di abilitazione all'incremento del conteggio (stato):

CE = S0 GO + S1 + S2 GO D_RDY

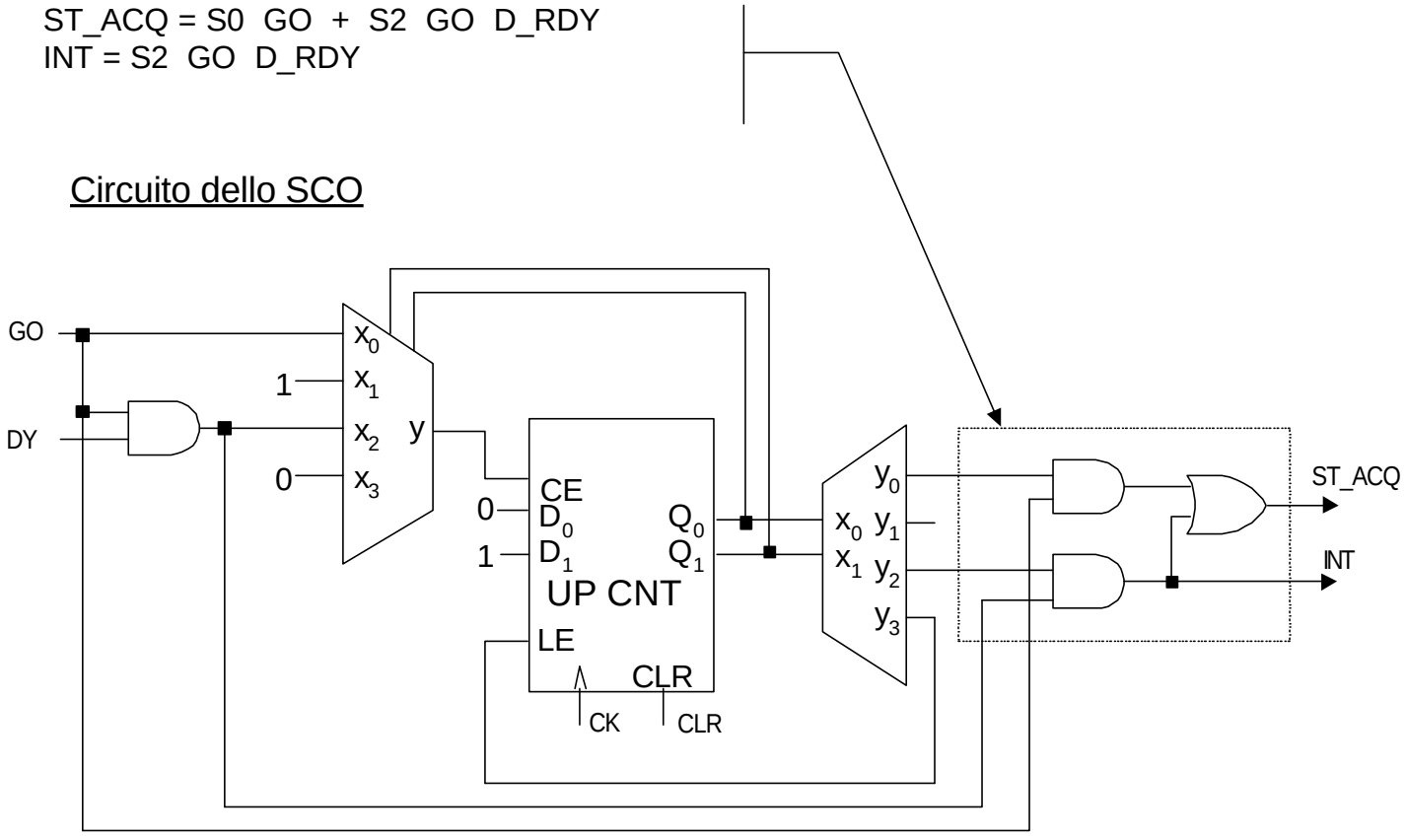
· Equazione di caricamento dello stato 2 nel contatore:

LE = S3

· Equazioni delle variabili (entranti nel registro) di uscita:

ST_ACQ = S0 GO + S2 GO D_RDY
INT = S2 GO D_RDY

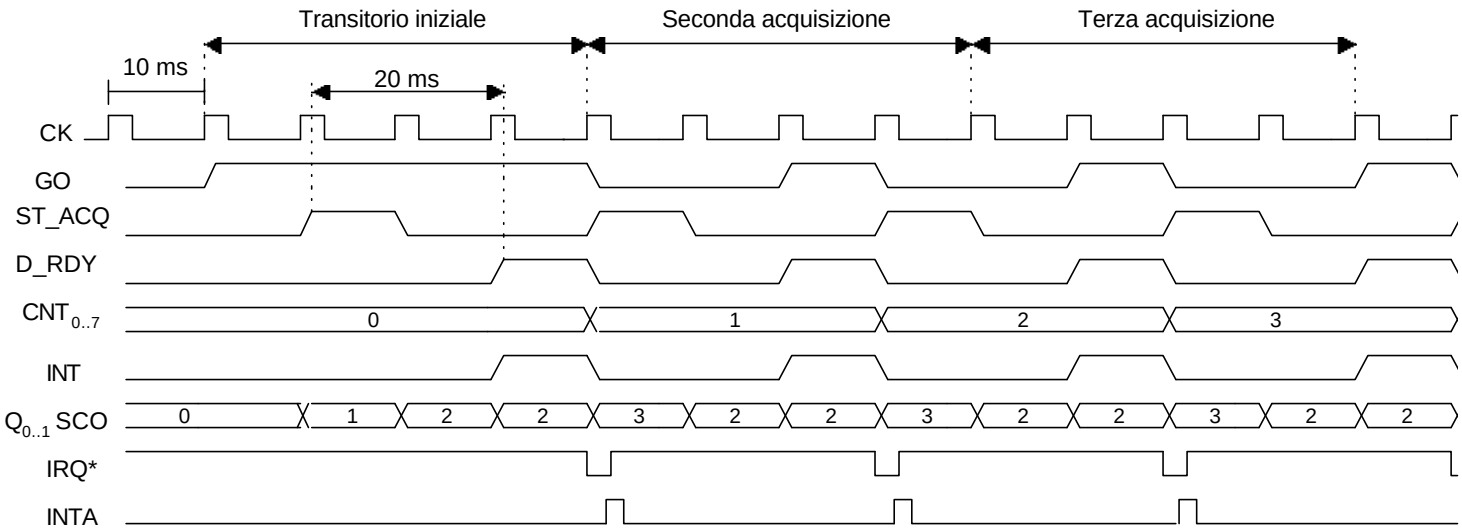
Circuito dello SCO



Note

Il flip-flop che memorizza INT, e che deve essere resettato da INTA, è quello incorporato nell'interfaccia di interrupt: per chiarezza di rappresentazione e per un posizionamento più appropriato è stato scorporato dal registro di uscita dello SCO di tipo D-Mealy.

IRRIGATORE: temporizzazioni



Note

Calcolo del periodo di scansione:
Il flow-chart gestisce la fase transitoria iniziale con gli stati S0 e S1, in cui dopo avere rilevato un segnale di avvio da parte del micro la periferica comanda la prima conversione AD. La fase di regime è gestita dagli stati S2 e S3: all'inizio di S3 viene avviata la conversione analogico-digitale; l'ADC impiega al massimo 20 ms (specifica), che è lo stesso tempo che lo SCO spende in S3 e in S2, per l'ipotesi (specifica) della durata di 10 ms di CK. All'inizio di S3 lo SCO emette anche la richiesta di interrupt che assieme alla routine di servizio impegnerà un tempo massimo di 15 ms (specifica); questo significa che, tornato in S2, alla fine del relativo periodo di CK (ne saranno passati due, cioè 20 ms, dalla richiesta di interruzione) lo SCO potrebbe subito eseguire una nuova iterazione del ciclo. In realtà il flip-flop di campionamento di GO ritarda GO=1 di un periodo di CK e quindi in definitiva gli stati S2 e S3 dello SCO potranno essere percorsi in successione come descritto nel diagramma di temporizzazione, che evidenzia un periodo di 30 ms, maggiore di un periodo di CK rispetto al minimo intervallo di ripetizione (20 ms) compatibile con le specifiche .

IRRIGATORE: programma PD32

```
; sia.asm

;Il PD32 possiede in memoria una tabella (a partire dall'indirizzo SOGLIA)
;dove sono contenute le informazioni relative alle soglie di umidità minima (Smin)
;e massima (Smax) di ciascuna UAI.
;All'interno della routine di interruzione il PD32:
;preleva dalla SIA l'identificatore della UAI (id) ed il valore corrente dell'umidità (U-id).
;confronta il valore dell'umidità con le soglie in memoria ed invia un comando
;di ATTIVA/DISATTIVA al SIA
;per l'elettrovalvola della  UAI (id) secondo la seguente tabella:
;ATTIVA          DISATTIVA
;1       0        (U-id < Smin-id)
;0       1        (U-id > Smax-id)
;0       0       Smin-id= U-id = Smax-id

;**************************
; COSTANTI
;**************************

org 400h                        ;richiesta dall'assemblatore PD32 prima delle equ

addr0           equ 000h        ;indirizzo output SIA
addr1           equ 001h        ;indirizzo input SIA

siaivn          equ 003h        ;ivn della SIA
siadrive        equ 2400h       ;indirizzo driver SIA

SOGLIA          equ 2000h       ;indirizzo iniziale tabella: lunga 512 byte

stack           equ 2800h       ;inizio area di stack PD32


;**************************
; VARIABILI
;**************************


;**************************
; CODICE
;**************************

code                            ;inizio istruzioni

main:
        movl #stack,r7          ;inizializza R7 quale SP: deve precedere
                                ;l'istruzione SETI
                                ;per gestire correttamente lo stack nella
                                ;fase di riconoscimento di una richiesta
                                ;di interruzione
        seti                    ;abilita PD32 ad accettare interruzioni (SP è stato
                                ;inizializzato)

        outb #00h,addr0         ;avvio SIA

mainloop:

;programma principale
```

```
        jmp mainloop


;**************************
; Sezione DRIVER
;**************************


;**************************
;sia
        driver siaivn, siadrive   ;Il driver della SIA ha IVN=siaivn
                                  ;e inizia dall'ind. siadrive
        push r0                   ;salva i registri r0,r1
        push r1

        inw addr1,r0
        movw r0,r1
        andl #000000FFh,r1        ;r1 <- codice UAI
        asll #1,r1                ;r1x2
        asrl #8,r0
        andl #000000FFh,r0        ;r0 <- codice umidità

        cmpb soglia(r1),r0        ;confronto r0 con la soglia minima
        jn putoff
        addl #1,r1
        cmpb soglia(r1),r0        ;confronto r0 con la soglia massima
        jnn puton
        jmp exit                  ;nop

putoff:                           ;disattivo la valvola
        outb #01h,addr1
        jmp exit
puton:                            ;attivo la valvola
        outb #02,addr1
        jmp exit
exit:
        pop r1                    ;ripristina i registri r1,r0
        pop r0
        rti
;**************************

end                               ;fine programma
```

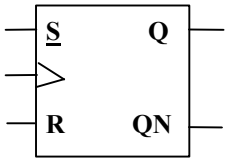
RETI LOGICHE

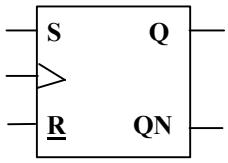
SECONDA PROVA SCRITTA DELL'APPELLO DEL 18-02-2000

Studente:_____ Docente:_____

D1 Si vuole utilizzare una PLA con 4 ingressi e 2 uscite Y0, Y1 per riconoscere se una parola del codice BCD appartiene alla prima o alla seconda metà del codice (Y0= 0/1 rispettivamente) e se è divisibile per 3 (Y1=1). Sintetizzare la PLA minima.

D2 Sintetizzare la coppia di flip-flop edge-triggered riportati in figura tramite flip-flop D. In caso di attivazione contemporanea delle linee di Set e Reset, l'ingresso sottolineato è prevalente.





D3 Codificare la tavola di flusso seguente con la tecnica "one-hot".

	00	01	11	10
a	**a**	**a**	**a**	b
b	a	**-**	c	**b**
c	d	**c**	**c**	**c**
d	**d**	a	-	c

D4 Descrivere una struttura dettagliata di interconnessione tra un banco di 8 registri, una ALU e uno shifter disposti su tre bus. Descrivere la temporizzazione per effettuare l'operazione: $(R_i + R_j) / 2 \rightarrow R_j$

D5 Un processore PD32 con un clock a 100 MHz interagisce con un banco di memoria ROM caratterizzato da un tempo di accesso pari a 80 ns. Il banco occupa uno spazio di 256 KB a partire dall'indirizzo 00010000h. Progettare un modulo di interfaccia del banco di ROM per gestire la sincronizzazione con il processore.

Esercizio (2S20000218-D1)

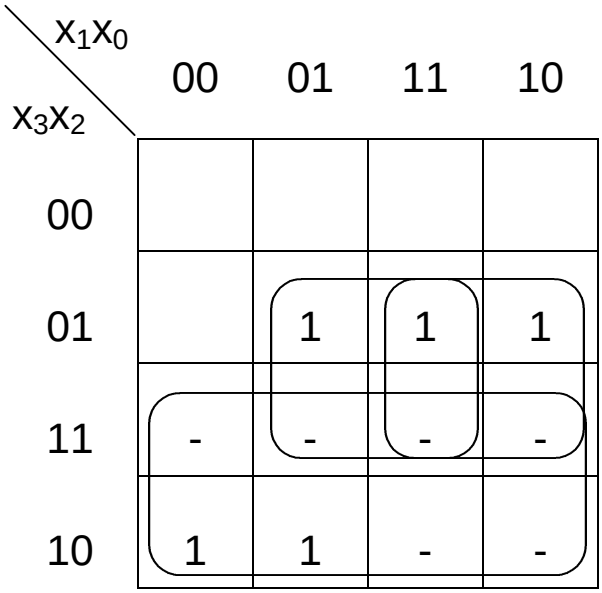
Si vuole utilizzare una PLA con 4 ingressi e 2 uscite Y0, Y1 per riconoscere se una parola del codice BCD appartiene alla prima o alla seconda metà del codice (Y0= 0/1 rispettivamente) e se è divisibile per 3 (Y1=1). Sintetizzare la PLA minima.

Le specifiche verbali possono essere trasferite sulla tavola di verità seguente

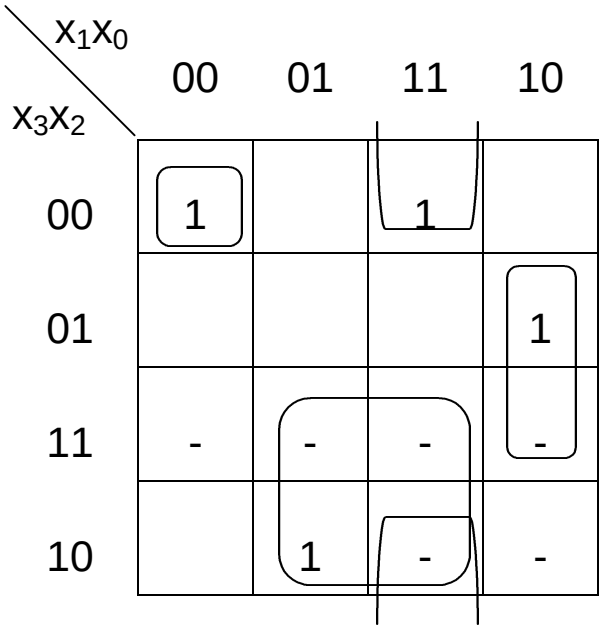
x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	1	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	1
1	0	1	0	-	-
1	0	1	1	-	-
1	1	0	0	-	-
1	1	0	1	-	-
1	1	1	0	-	-
1	1	1	1	-	-

La sintesi minima della PLA richiede la sintesi delle tre funzioni y_0, y_1 e y_0y_1, per la determinazione degli implicanti multipli.

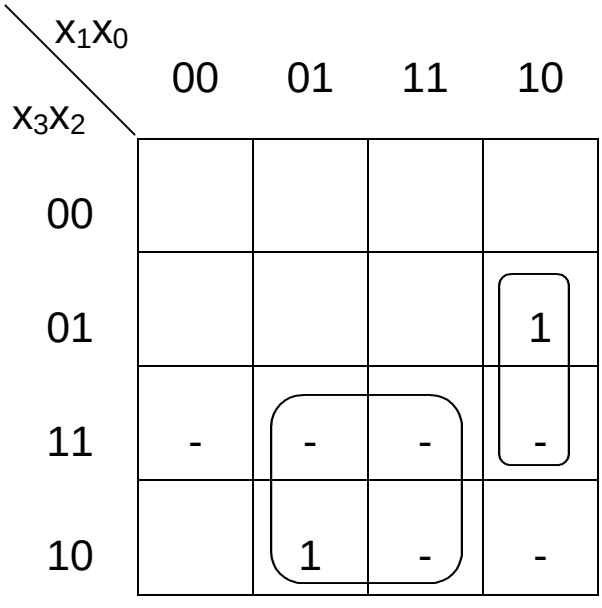
Occorreranno tre MK a 4 variabili



$$y_0 = x_2x_1 + x_2x_0 + x_3$$



$$y_1 = x_3x_0 + \overline{x_2}x_1x_0 +$$
$$+ x_2x_1\overline{x_0} + \overline{x_3}\,\overline{x_2}\,\overline{x_1}\,\overline{x_0}$$



Infine per $y_1 \cdot y_0$ si ottiene:

$$y_1 \cdot y_0 = x_3x_0 + x_2x_1\overline{x_0}$$

Ne segue che le due funzioni possono essere riscritte come:

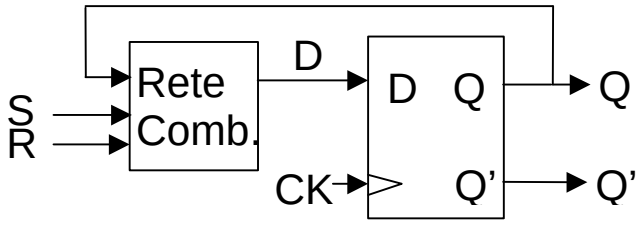
$$y_0 = x_2x_1\overline{x_0} + x_2x_0 + x_3$$
$$y_1 = x_3x_0 + \overline{x_2}x_1x_0 + x_2x_1\overline{x_0} + \overline{x_3}\,\overline{x_2}\,\overline{x_1}\,\overline{x_0}$$

utilizzando 6 implicanti invece di 7.

Esercizio (2S20000218-D2)

Sintetizzare la coppia di flip-flop edge-triggered riportati in figura tramite flip-flop D. In caso di attivazione contemporanea delle linee di Set e Reset, l'ingresso sottolineato è prevalente.

Ciascuno dei due flip-flop può essere considerato una rete sequenziale sincrona, che risponde al noto modello strutturale, che di seguito viene personalizzato al caso del flip-flop: il registro di stato è composto da un solo flip-flop D, esattamente quello che il testo del problema raccomanda di utilizzare.

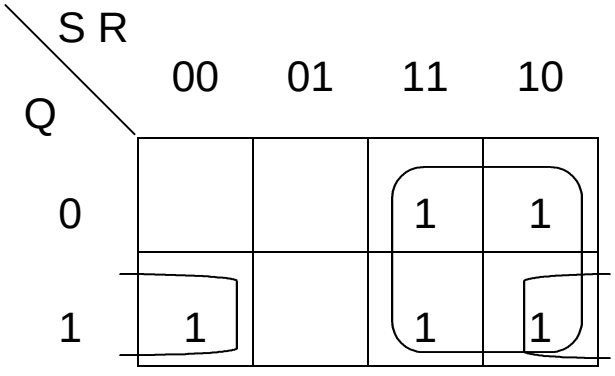


E' appena il caso di notare che in questa struttura, come in tutte quelle che rispondono allo stesso modello strutturale di rete sincrona, il clock è un segnale non soggetto a elaborazioni, di nessun tipo!

Il progetto è stato così ricondotto al calcolo della rete combinatoria. D è lo stato successivo del flip-flop (registro di stato della rete).
Per il primo dei due flip-flop (Set prevalente) la tavola di verità è:

Q	S	R	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

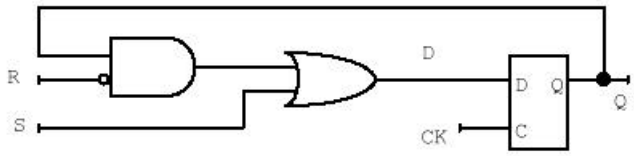
Minimizzazione mediante MK:



Si ottiene:

$$D = S + Q \cdot \overline{R}$$

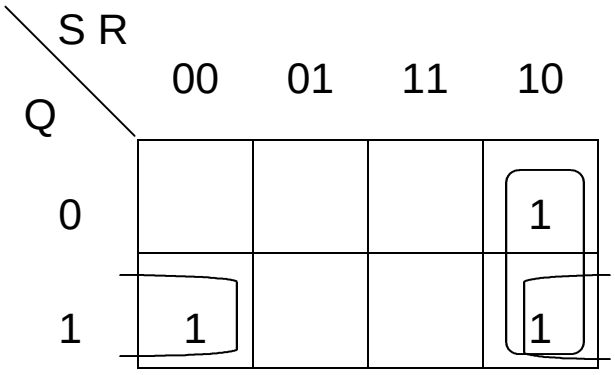
Ne consegue lo schema del flip-flop richiesto:



Per il secondo dei due flip-flop (Reset prevalente) la tavola di verità è:

Q	S	R	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

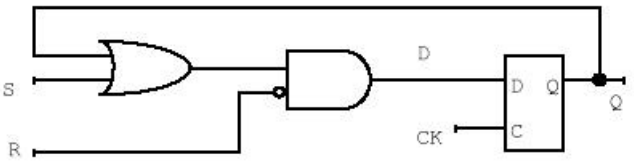
Minimizzazione mediante MK:



Si ottiene:

$$D = S\overline{R} + Q \cdot \overline{R} = (S + Q) \cdot \overline{R}$$

Ne consegue lo schema del flip-flop richiesto:



Esercizio (2S20000218-D3)

Codificare la tavola di flusso seguente con la tecnica “one-hot”.

	00	01	11	10
a	**a**	**a**	**a**	b
b	a	**-**	c	**b**
c	d	**c**	**c**	**c**
d	**d**	a	-	c

Si codificano i 4 stati in modo lineare:

a 0001
b 0010
c 0100
d 1000

quindi si scrivono le altre 6 codifiche associate agli stati di appoggio su cui verranno effettuate transizioni intermedie nel passaggio tra coppie di stati originali:

ab 0011
ac 0101
ad 1001
bc 0110
bd 1010
cd 1100

La tavola di flusso può essere riscritta tenendo conto delle transizioni per gli stati intermedi:

	00	01	11	10
a	**a**	**a**	**a**	ab
b	ab	-	bc	**b**
c	cd	**c**	**c**	**c**
d	**d**	ad	-	cd
ab	a	-	-	b
ac	-	-	-	-
ad	-	a	-	-
bc	-	-	c	-
bd	-	-	-	-
cd	d	-	-	c

Nella tavola di flusso codificata possono essere omesse le righe relative agli stati di appoggio su cui non sono previste transizioni:

	00	01	11	10
0001	**0001**	**0001**	**0001**	0011
0010	0011	-	0110	**0010**
0100	1100	**0100**	**0100**	**0100**
1000	**1000**	1001	-	1100
0011	0001	-	-	0010
1001	-	0001	-	-
0110	-	-	0100	-
1100	1000	-	-	0100

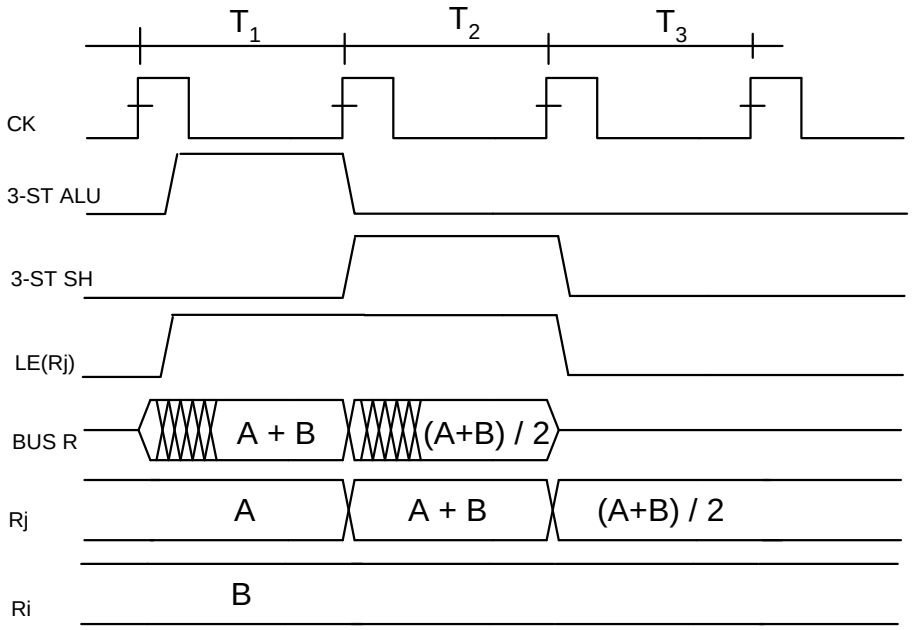
Esercizio (2S20000218-D4)

Descrivere una struttura dettagliata di interconnessione tra un banco di 8 registri, una ALU e uno shifter disposti su tre bus. Descrivere la temporizzazione per effettuare l'operazione: $(R_i + R_j) / 2 \rightarrow R_j$

La struttura è quella riportata nella fig. 8.8 del testo "Reti sequenziali", con la variazione di 8 registri invece di 4.
Per poter essere eseguita dalla struttura disponibile, l'operazione richiesta deve essere scomposta nella sequenza delle due microoperazioni seguenti:

op1: $R_i + R_j \rightarrow R_j$
op2: $R_j / 2 \rightarrow R_j$

che vengono eseguite dalla rete SCA (e dallo SCO che la pilota) nei due periodi di clock T1 e T2 del diagramma temporale seguente, in cui è tracciata la dinamica dei principali segnali coinvolti nell'esecuzione delle due micro-operazioni op1 e op2.



Si è supposto che i registri Rj e Ri contengano i valori iniziali rispettivi A e B nel ciclo di clock T_1.

Commenti all'esercizio

- Al termine di T1, sul fronte del segnale di orologio ck a cui sono sensibili i registri, viene conclusa op1 con la scrittura di R_j; al termine di T2 viene conclusa op2 (seconda scrittura di R_j).

- Va notato che il registro di destinazione R_j è utilizzato anche per appoggiare il risultato intermedio (somma).

- Durante T1 viene attivato il percorso dati (= buffer 3-state) all'uscita dell'ALU; durante T2 viene invece forzata sul bus R l'uscita dello shifter (che al termine di T2 viene trasferita in R_j).

- Durante T1 e T2 è attiva la linea di abilitazione al caricamento di R_j, che deve essere caricato al termine sia di T1 che di T2; questo viene effettuato dalla struttura tramite il decoder pilotato con il codice di R_j come indirizzo di destinazione (DA: Destination Address).

- I segnali di controllo si suppongono attivati da uno SCO, sincrono con lo SCA.

Esercizio (2S20000218-D5)

Un processore PD32 con un clock a 100 MHz interagisce con un banco di memoria ROM caratterizzato da un tempo di accesso pari a 80 ns. Il banco occupa uno spazio di 256 KB a partire dall'indirizzo 00010000h. Progettare un modulo di interfaccia del banco di ROM per gestire la sincronizzazione con il processore.

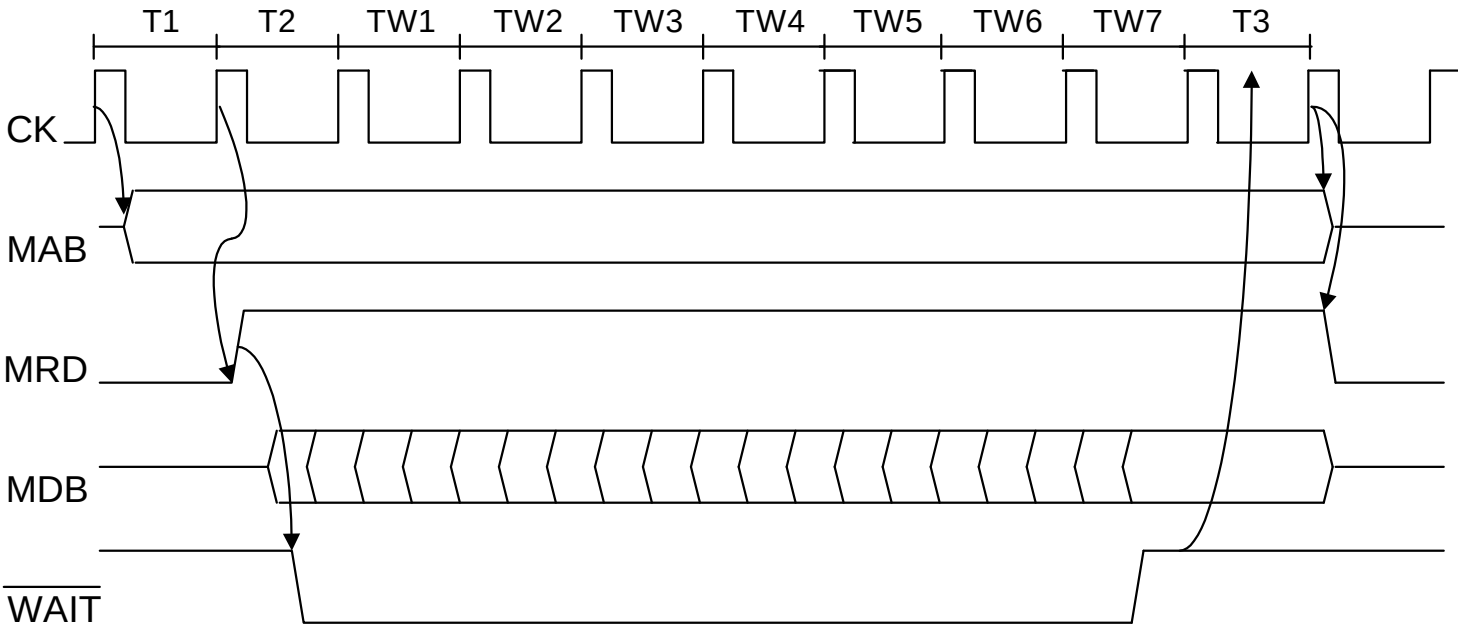
---

1 - Considerazioni preliminari

Il trasferimento corretto dei dati può essere controllato introducendo un numero opportuno di stati di wait: in una lettura con tre cicli T si presuppone che la memoria (ROM / RAM) abbia un tempo di accesso non superiore al tempo di ciclo T, in modo che il dato sia stabile in T2 e T3; pertanto, per riprodurre tale situazione in chiusura del ciclo di lettura, occorre introdurre un numero di stati di wait in modo che globalmente il ciclo di lettura duri 80 ns + 2 x 10 ns = 100 ns; in altri termini occorre introdurre 10 – 3 = 7 stati di wait.

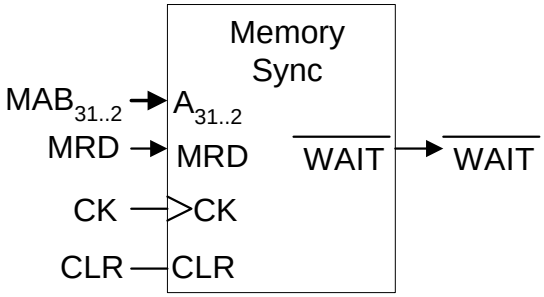
2 - Timing

Utilizzando il clock a 100 MHz del processore, il trasferimento avviene secondo il diagramma di temporizzazione seguente, essendo ogni ciclo T di durata pari a 10 ns:

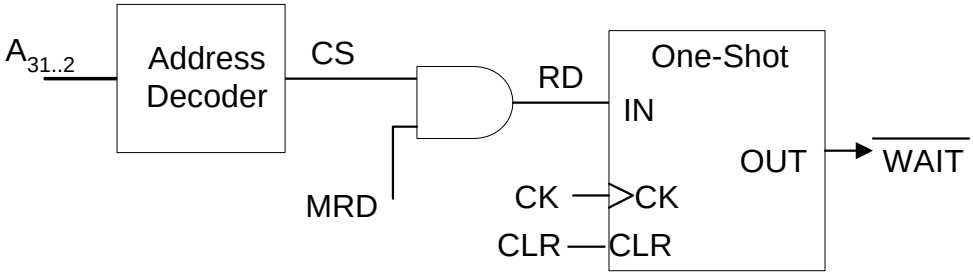


3 – Schema a blocchi

Pertanto, il controllore di interfaccia da progettare può essere schematizzato nel modo seguente:



Dal diagramma di temporizzazione si evince che si tratta di un generatore di un impulso di tipo Mealy, in quanto nello stato T2 reagisce all'attivazione di MRD azzerando WAIT' senza aspettare il clock.
Poiché questo accade dopo che in T1 il banco di ROM è stato indirizzato, il modulo Memory Sync può essere dettagliato come nella figura seguente, in cui il modulo Address Decoder è la rete combinatoria che decodifica l'indirizzamento della ROM e One-Shot è la rete sequenziale sincrona di tipo Mealy che produce l'impulso WAIT'.



4 – Progetto del modulo Address Decoder

Lo spazio di indirizzamento della ROM è rappresentato nella tabella seguente:

31 . . . 19	**18 17 16**	**15 . . . 2 (1 0)**	**¬ MAB**	
0 . . . 0	0 0 1	0 . . . 0 0 0	64 K	
0 . . . 0	0 1 0	0 . . . 0 0 0	128 K	
0 . . . 0	0 1 1	0 . . . 0 0 0	192 K	256 K
0 . . . 0	1 0 0	0 . . . 0 0 0	256 K	
0 . . . 0	1 0 0	1 . . . 1 1 1	320 K - 1	

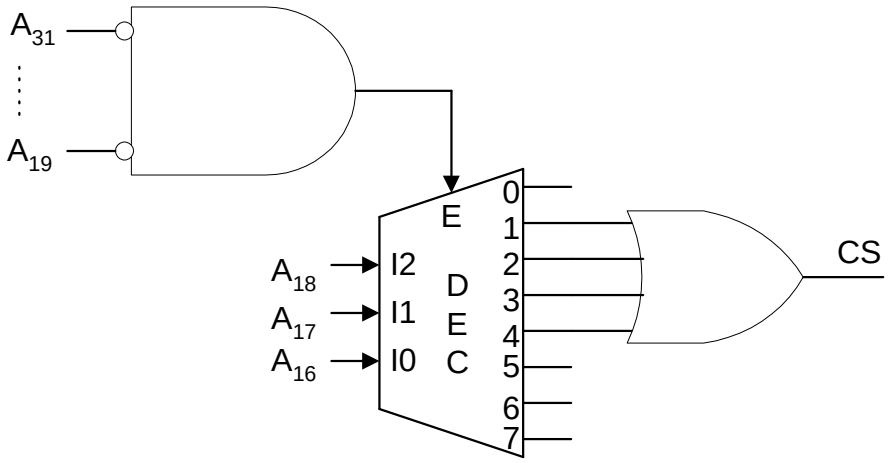
Da cui si deduce che il banco di ROM è univocamente identificato dalla seguente configurazione dei bit di indirizzo (MAB):

Bit 31..19: 0..0
Bit 18..16: 001, 010, 011, 100

Cioè la funzione di selezione (CS) della ROM è la seguente (dove i bit MAB_k sono indicati con A_k per semplicità):

$$CS = \overline{A_{31}A_{30}}..\overline{A_{19}}(\overline{A_{18}A_{17}A_{16}} + \overline{A_{18}}A_{17}\overline{A_{16}} + \overline{A_{18}}A_{17}A_{16} + A_{18}\overline{A_{17}A_{16}})$$

che può essere implementata con la rete seguente:

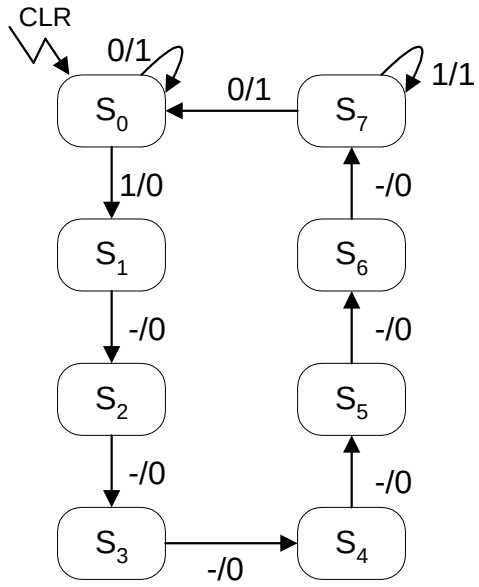


Osservazione:

Si tenga presente che la ROM per essere collocata nello spazio di indirizzamento del PD32 dovrà comunque essere separata nei 4 moduli di colonna; pertanto la funzione ricavata sarà direttamente utilizzabile all'interno del modulo Memory Sync, ma andrà composta in AND con ciascuno dei 4 Mb (3,2,1,0) per produrre i 4 CS in ingresso ai 4 moduli interni al banco di ROM.

5 – Progetto del modulo One-Shot

Il comportamento del modulo One-Shot può essere descritto con una macchina a stati finiti, riportando le informazioni del diagramma di temporizzazione su un diagramma degli stati:



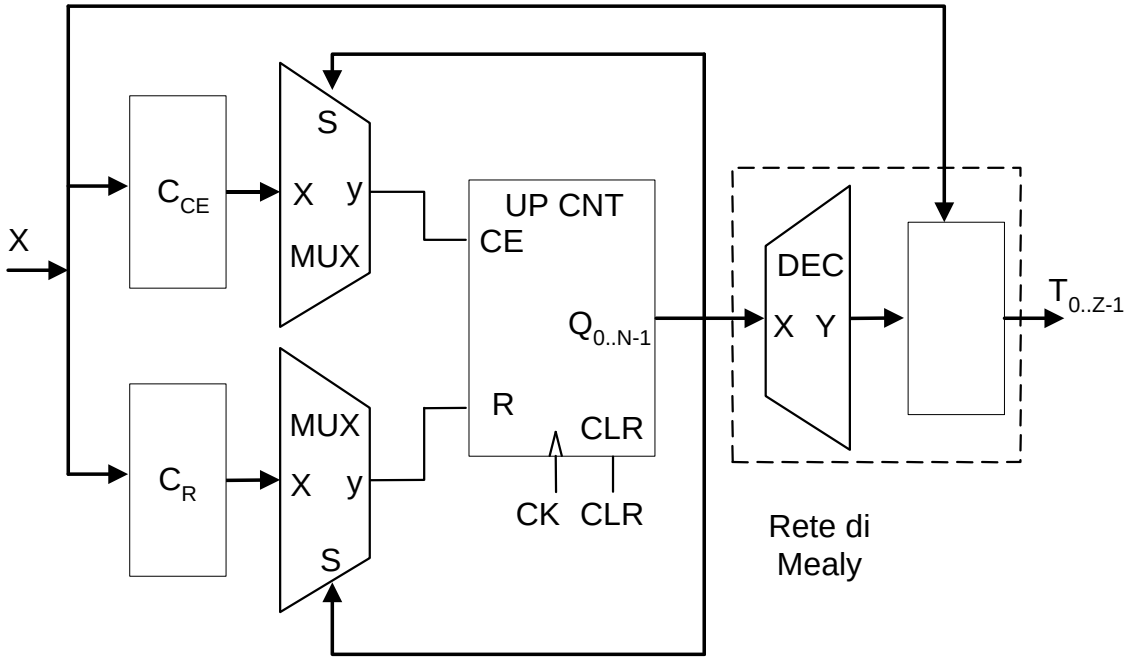
I = {0,1} valori della variabile IN
O= {0,1} valori della variabile OUT

Questa macchina può essere sintetizzata con le procedure di minimizzazione canoniche, oppure si può riconoscere l'applicabilità del sequenziatore MSI, che richiede un contatore a 3 bit (8 stati) dotato di CE e R, un decoder e due mux; si osservi che l'ingresso R (il reset) del contatore non sarebbe strettamente necessario in questo caso particolare, in quanto lo stato 0 è raggiungibile dal solo stato 7, che precede 0 nella scansione normale di conteggio.

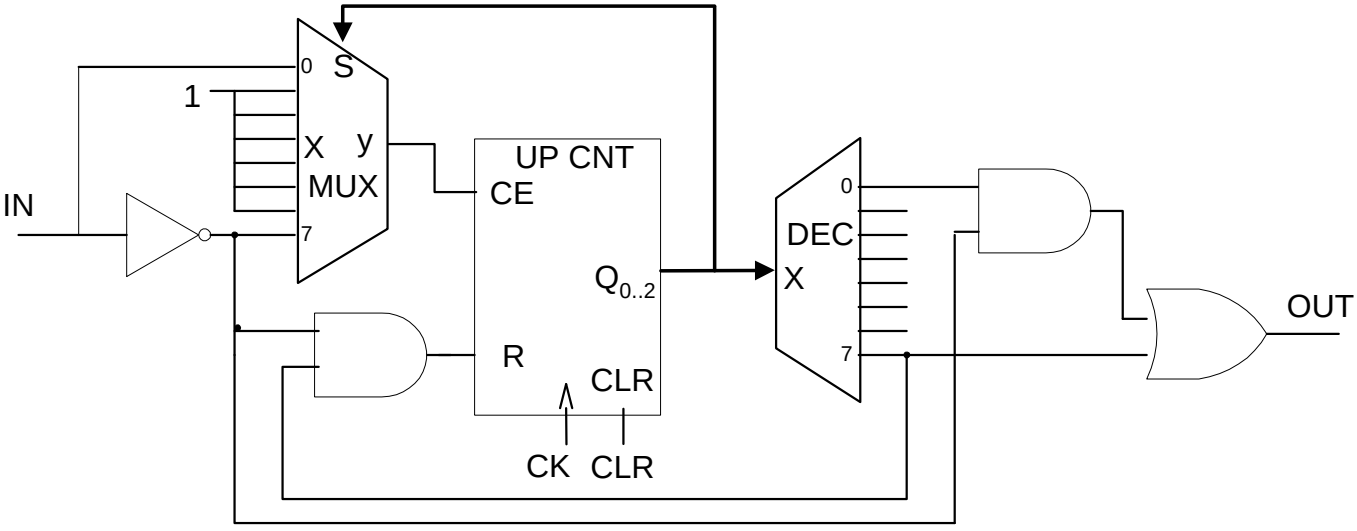
Si osservi anche che il sequenziatore MSI deve rispondere al modello di Mealy, ottenibile dal sequenziatore di Moore riportando l'(unico, in questo caso) ingresso IN alla rete di uscita che calcola OUT, secondo il modello generale riportato nella figura seguente:

Dal diagramma degli stati possono essere calcolate le funzioni del sequenziatore:

$CE = S_0\,IN + S_1 + S_2 + S_3 + S_4 + S_5 + S_6 + S_7\,IN'$
$(R = S_7\,IN')$
$OUT = S_0\,IN' + S_7$



Lo schema generale del sequenziatore si particolarizza nello schema del modulo One-Shot:



Si noti, come prevedibile, che il sistema di Mealy aggiorna l'uscita mediante decodifica di un conteggio, e pertanto il segnale OUT potrà avere impulsi spuri (glitch o spike); questo è tollerabile, in quanto OUT viene campionato dalla CPU sugli stessi fronti (di salita) di CK usato dal modulo (la CPU e Memory Sync costituiscono un sistema globalmente sincrono).
Si noti anche che non è consentito introdurre un flip-flop di campionamento su OUT perché Memory Sync deve essere di Mealy (significa che WAIT' deve poter attivarsi nello stesso ciclo – T_2 – in cui si attiva MRD).

RETI LOGICHE

PRIMA PROVA SCRITTA DEL 14-04-2000

STUDENTE:_____ DOCENTE:_____

Si vuole progettare un controllore di DMA (DMAC) per ricercare la posizione di una stringa prescritta nella memoria del processore PD32.

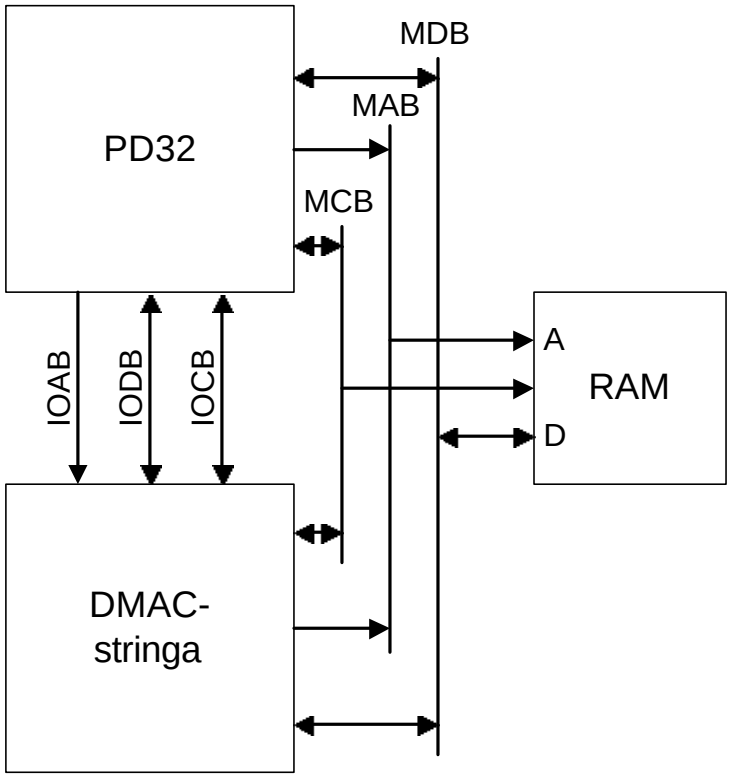
Il processore PD32 invia al DMAC:
- la lunghezza, in byte, della stringa da ricercare: 1..16;
- l'indirizzo iniziale di memoria dove è posta la stringa da ricercare nel blocco;
- gli indirizzi iniziale e finale del blocco di memoria in cui cercare la stringa.

Il DMAC:
- accede al blocco di memoria specificato a *burst* consecutivi ognuno di estensione 1 Kbyte e separati dal rilascio dei bus;
- terminerà la scansione alla prima occorrenza della stringa specificata, oppure al termine della lettura dell'intero blocco di memoria se non ha trovato la stringa; il DMAC comunicherà l'esito della ricerca mediante interruzione al processore, il quale, in caso di successo, provvederà a prelevare dal DMAC l'indirizzo iniziale della stringa.

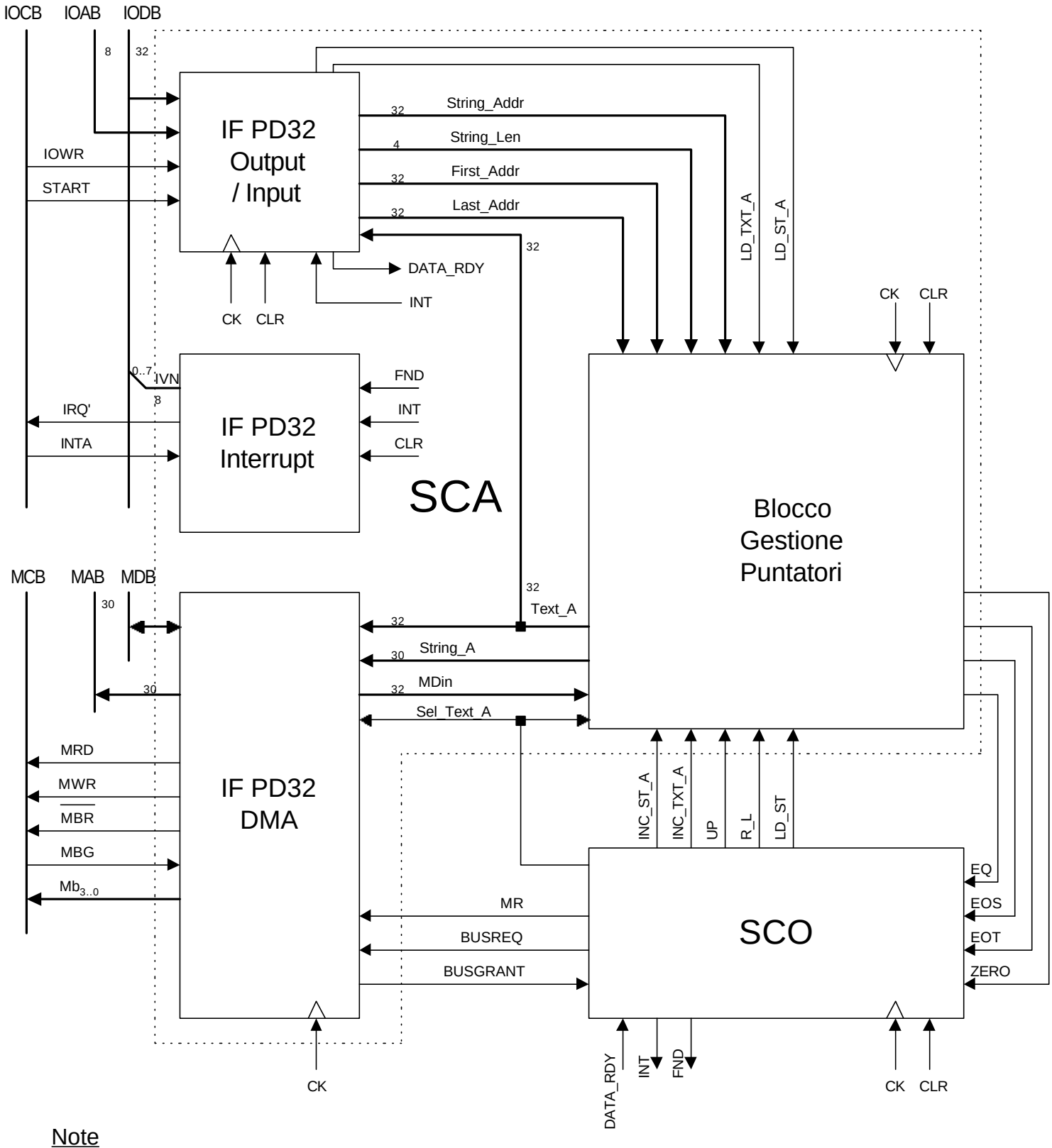
Si richiede:

- lo schema logico dettagliato della periferica e le relative temporizzazioni;
- la routine di interruzione del PD32.

DMAC-stringa: sistema esterno



DMAC-stringa: Schema a blocchi

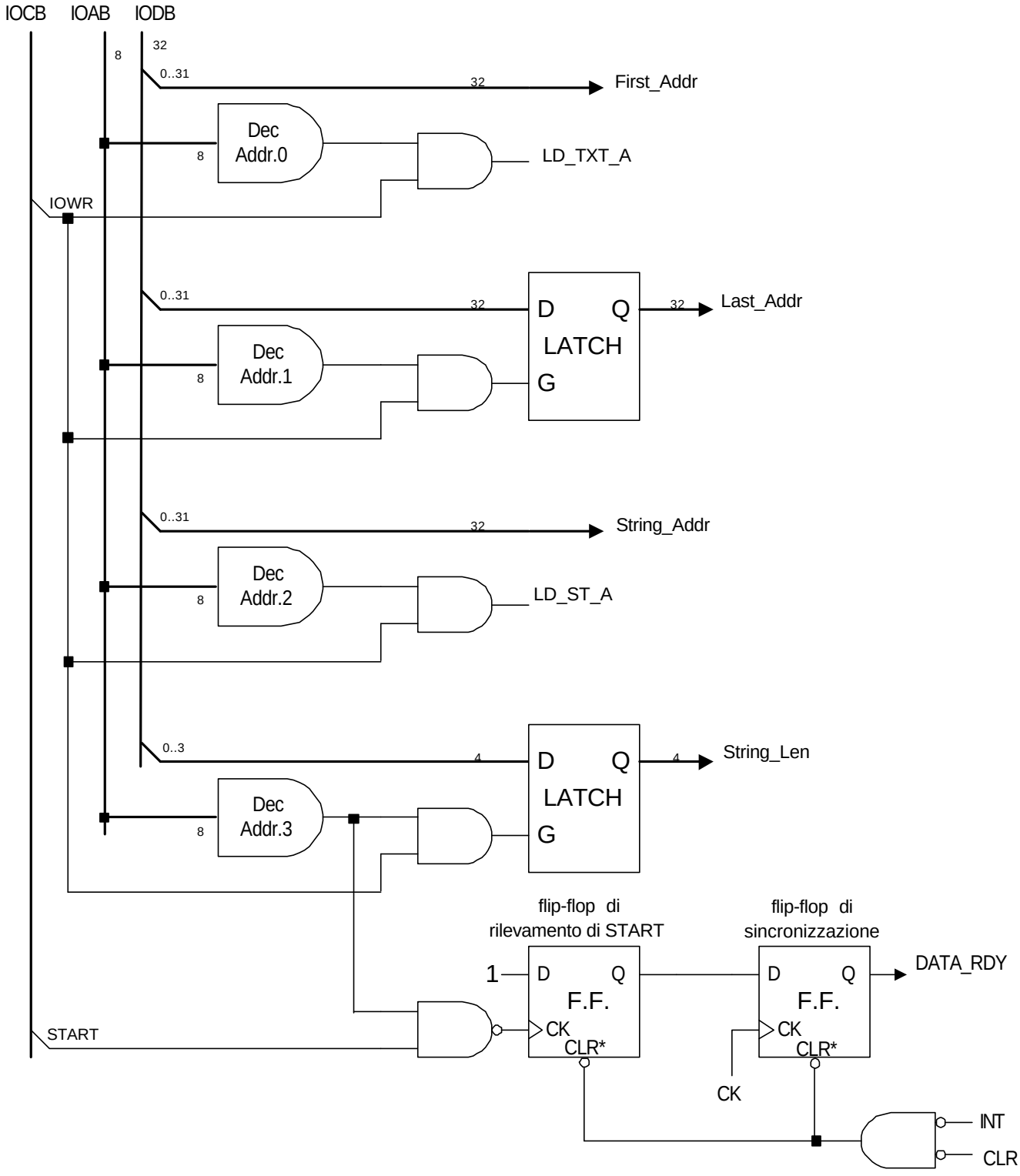


Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.

Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

DMAC-stringa: IF PD32 - output

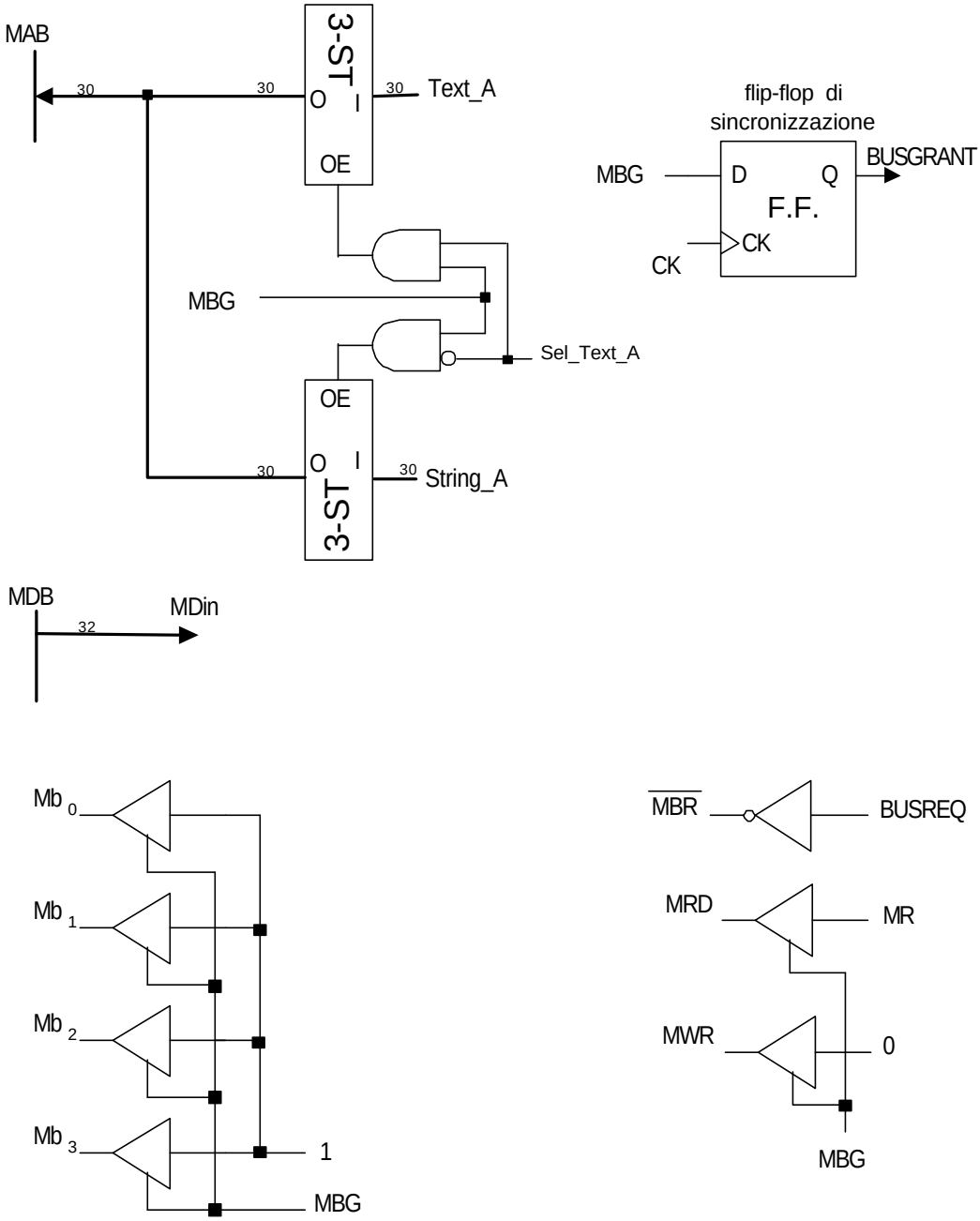


Note

L'indirizzo di START può essere uno qualunque dei quattro associati ai registri.

Il SW può avviare l'operazione con **START Addr3** dopo avere eventualmente riscritto uno o più registri (le informazioni memorizzate nei registri di interfaccia non vengono alterate dalla periferica).

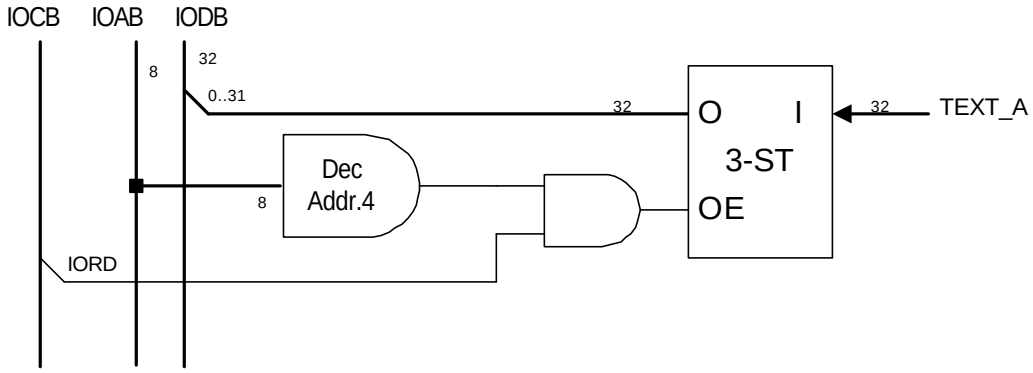
DMAC-stringa: IF PD32 - DMA



Dalla RAM vengono lette LW, di cui però viene registrato un byte alla volta.

MWR deve essere pilotato (a 0), anche se non dinamicamente: la struttura di memoria deve essere governata pienamente dall'interfaccia DMA, che ne ha piena responsabilità quando MBG=1.

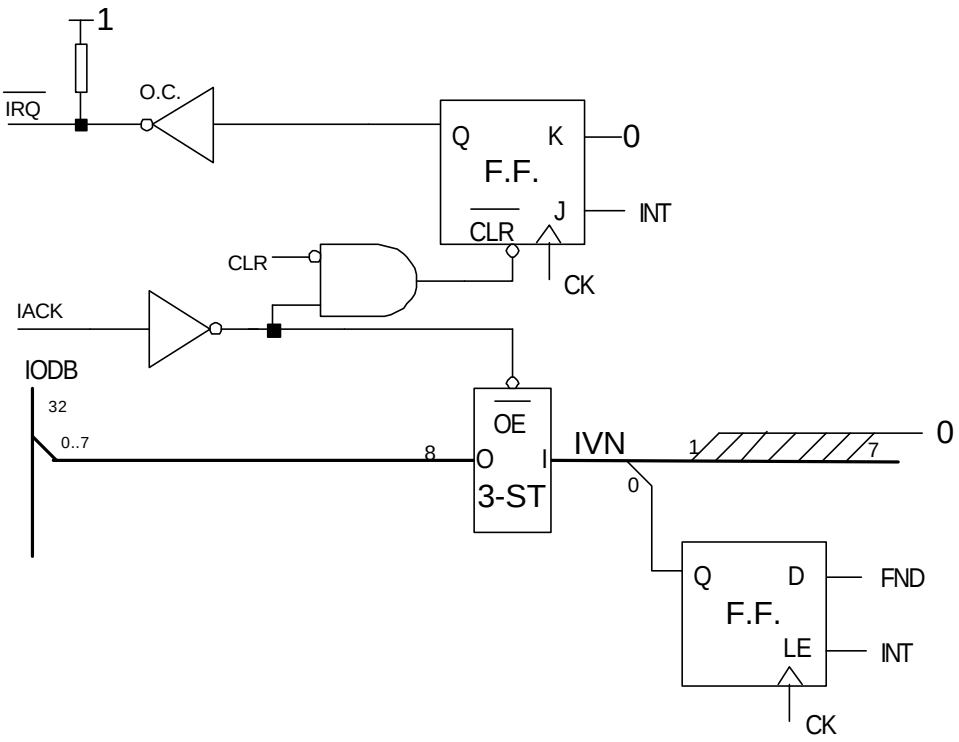
DMAC-stringa: IF PD32 - input



Note

L'interfaccia di input non necessita di un flip-flop di handshake, in quanto il meccanismo di produzione (da parte della periferica) e consuno (da parte del micro) del dato è controllato mediante l'interruzione; infatti, il processore:
- legge il dato a seguito del riconoscimento dell'interruzione,
- potrà richiedere un nuovo servizio alla periferica solo dopo avere acquisito il dato.

DMAC-stringa: IF PD32 - interrupt

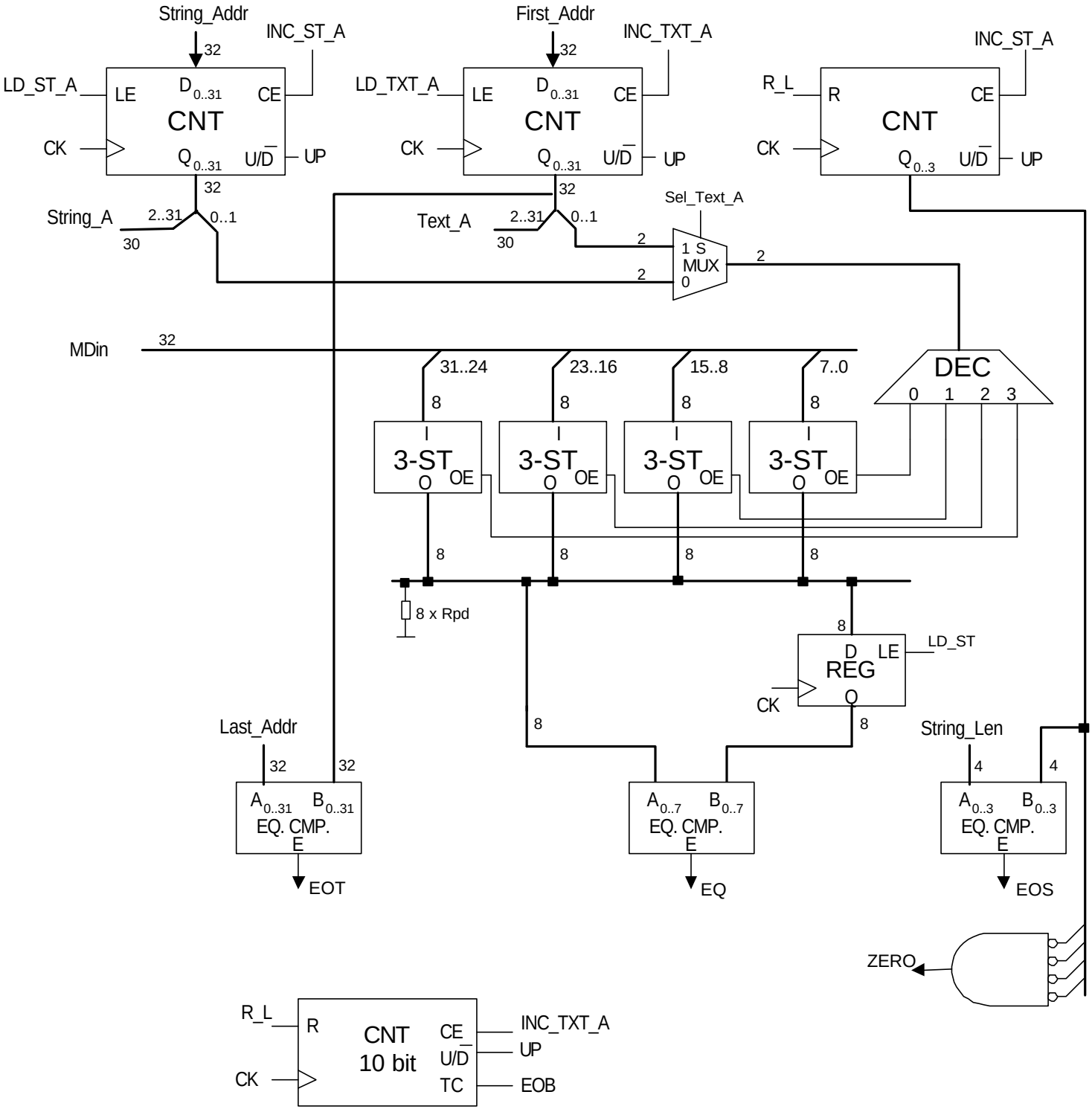


Note

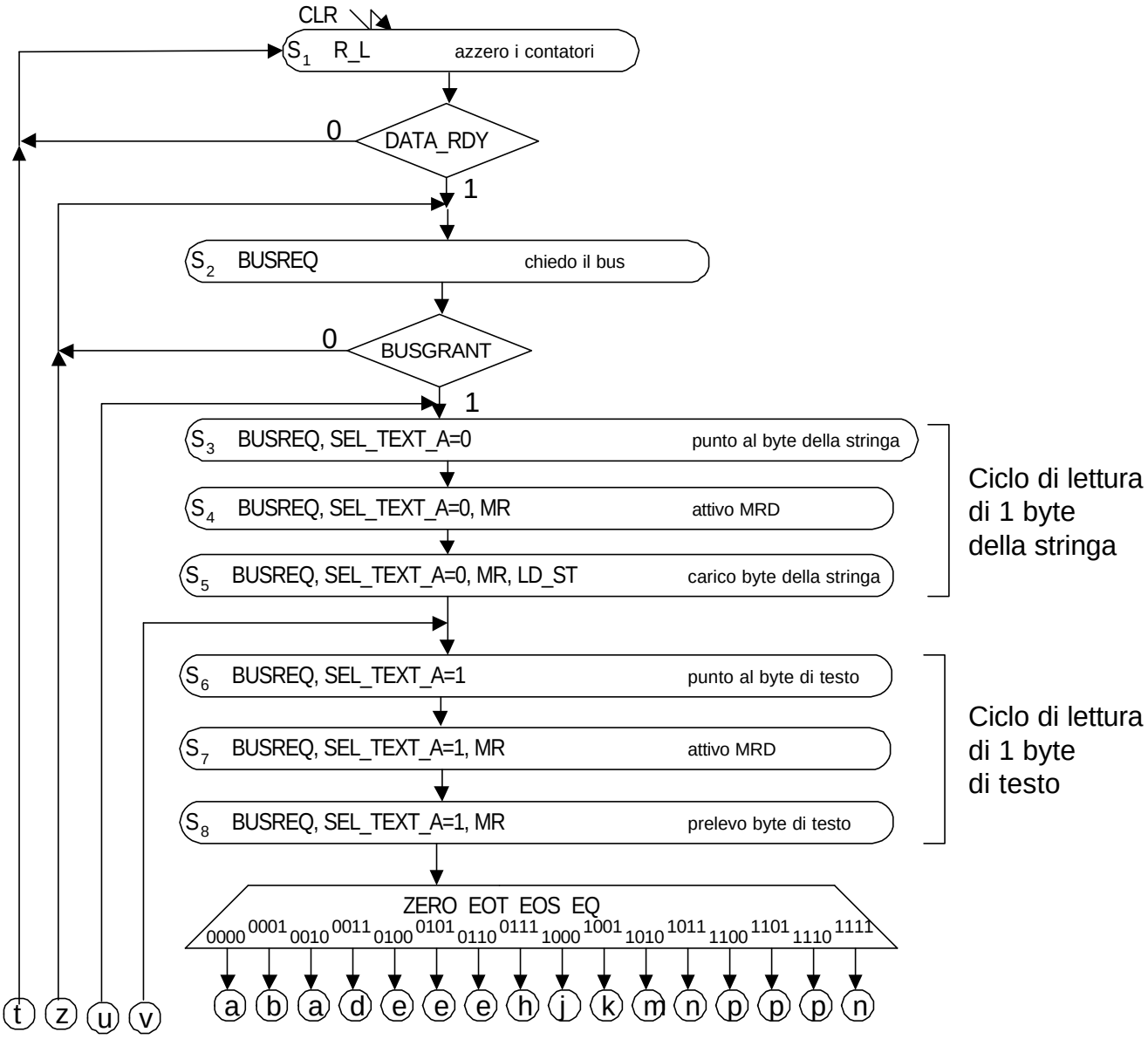
L'IVN (a 8 bit) incorpora nella posizione 0 il bit dell'esito della ricerca della stringa; gli altri 7 bit sono stati cablati a 0; questa filatura comporta l'impiego di due driver SW, corrispondenti ai due IVN 00h (stringa non trovata) e 01h (stringa trovata).

Si noti il caricamento del bit 0 dell'IVN nel flip-flop D abilitato sullo stesso fronte di CK che attiva la richiesta di interruzione.

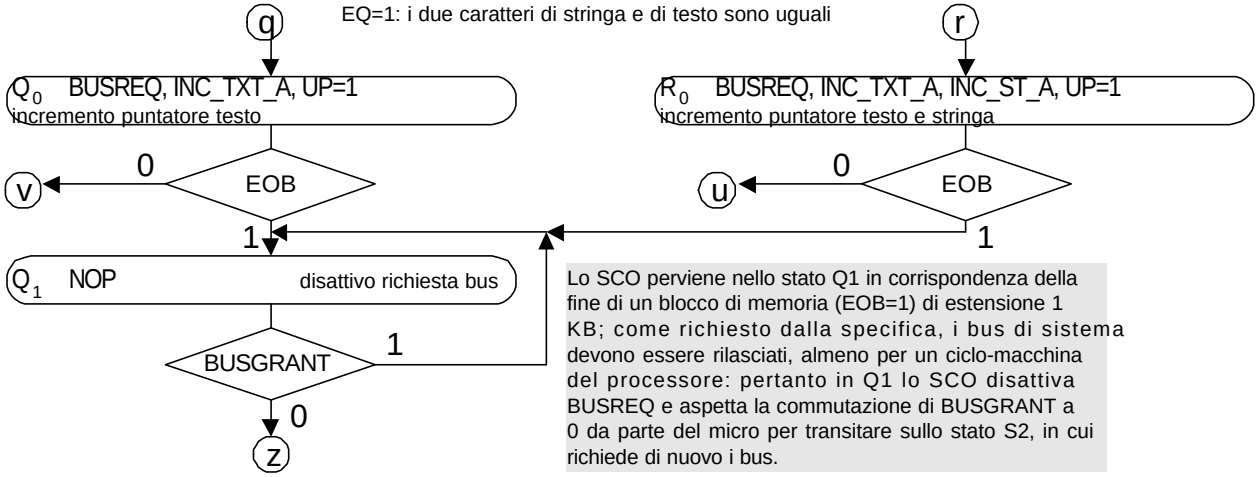
DMAC-stringa: blocco gestione puntatori



DMAC-stringa: SCO - flowchart

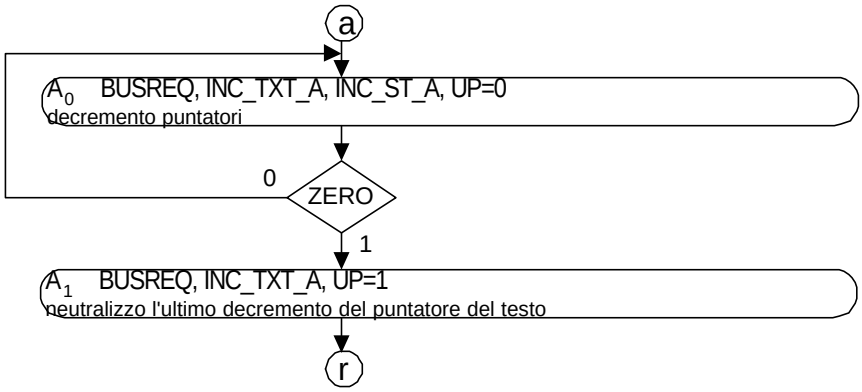


ZERO=0: analisi di una stringa candidata in corso
ZERO=1: ricerca del primo carattere della stringa prescritta in corso
EOT=0: testo non ancora esaurito
EOT=1: l'indirizzo del testo è l'ultimo disponibile
EOS=0: il carattere corrente della stringa non è quello finale
EOS=1: il carattere corrente della stringa è quello finale
EQ=0: i due caratteri di stringa e di testo sono diversi
EQ=1: i due caratteri di stringa e di testo sono uguali



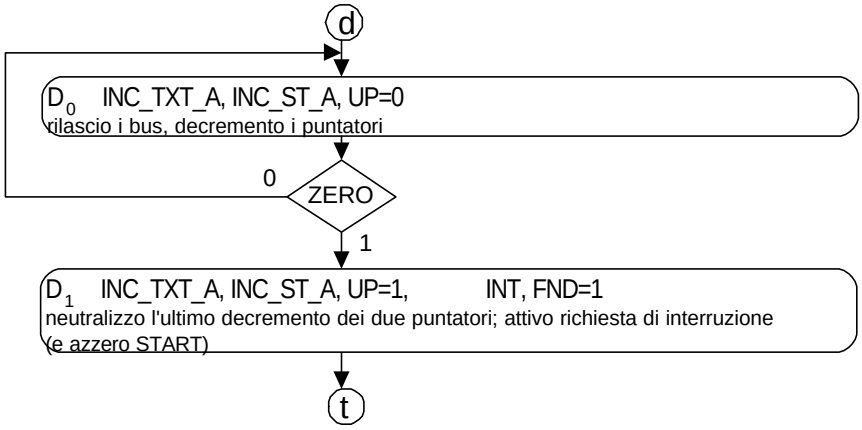
Lo SCO perviene nello stato Q1 in corrispondenza della fine di un blocco di memoria (EOB=1) di estensione 1 KB; come richiesto dalla specifica, i bus di sistema devono essere rilasciati, almeno per un ciclo-macchina del processore: pertanto in Q1 lo SCO disattiva BUSREQ e aspetta la commutazione di BUSGRANT a 0 da parte del micro per transitare sullo stato S2, in cui richiede di nuovo i bus.

DMAC-stringa: SCO - flowchart

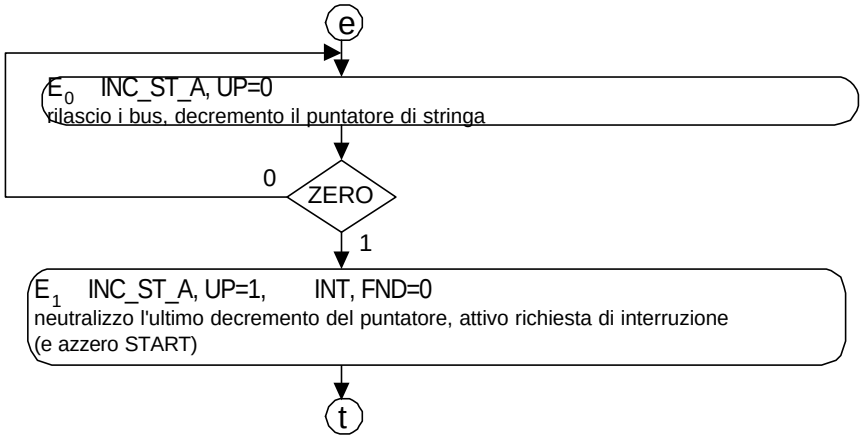


ZERO=0 EOT=0 EOS=0/1 EQ=0
analisi di una stringa candidata,
la coppia dei caratteri della stringa e del testo differiscono:
riavvolgo il puntatore di stringa al primo carattere della stringa e il puntatore del testo al primo byte successivo a quello confrontato con il primo carattere della stringa.

ZERO=0 EOT=0 EOS=0 EQ=1
analisi di una stringa candidata,
la coppia dei caratteri della stringa e del testo coincidono:
prelevo il byte successivo della stringa e del testo.

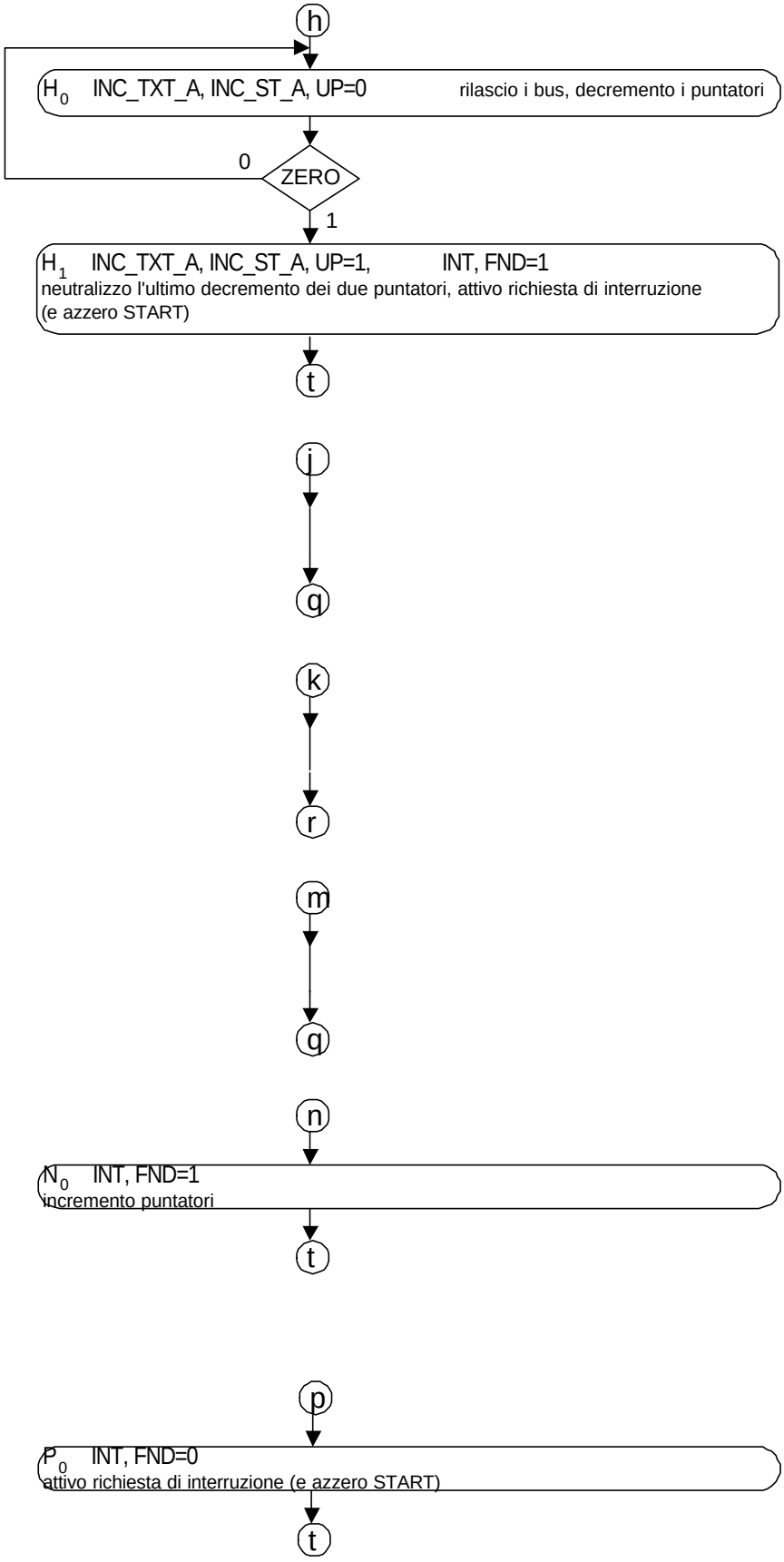


ZERO=0 EOT=0 EOS=1 EQ=1
analisi di una stringa candidata,
la coppia dei caratteri della stringa e del testo coincidono, e si è raggiunta la fine della stringa:
la stringa è stata trovata;
rilascio i bus;
riavvolgo il puntatore di stringa al primo carattere della stringa e il puntatore del testo al primo byte della stringa trovata;
dopo attivo INT insieme al bit di stringa trovata.



ZERO=0 EOT=1 EOS=0 EQ=0
analisi di una stringa candidata,
la stringa non è terminata, ma il testo sì:
rilascio i bus;
riavvolgo il puntatore di stringa al primo carattere della stringa ;
inoltre attivo INT lasciando a 0 il bit di stringa trovata.

DMAC-stringa: SCO - flowchart



ZERO=0 EOT=1 EOS=1 EQ=1
analisi di una stringa candidata,
la coppia dei caratteri della stringa e del testo coincidono, e si è raggiunta la fine della stringa;
si è raggiunta anche la fine del testo:
la stringa è stata trovata;
rilascio i bus;
riavvolgo il puntatore di stringa al primo carattere della stringa e il puntatore del testo al primo byte della stringa trovata;
inoltre attivo INT insieme al bit di stringa trovata.

ZERO=1 EOT=0 EOS=0 EQ=0
ricerca in corso del primo carattere della stringa prescritta;
il carattere del testo analizzato differisce dal primo della stringa prescritta:
prelevo il byte successivo del testo.

ZERO=1 EOT=0 EOS=0 EQ=1
ricerca in corso del primo carattere della stringa prescritta;
il carattere del testo analizzato coincide con il primo della stringa prescritta:
prelevo il carattere successivo della stringa e il byte successivo del testo.

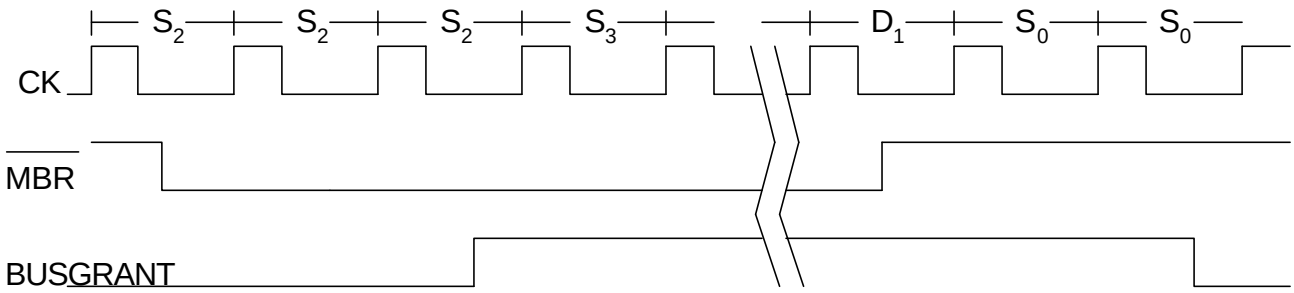
ZERO=1 EOT=0 EOS=1 EQ=0
ricerca in corso del primo carattere della stringa prescritta, che è lunga 1 solo carattere!
Il carattere del testo analizzato differisce da quello che costituisce la stringa prescritta:
evidentemente non c'è bisogno di riavvolgere i puntatori;
prelevo il carattere successivo del testo.

ZERO=1 EOT=0/1 EOS=1 EQ=1
ricerca in corso del primo carattere della stringa prescritta, che è lunga 1 solo carattere!
Il carattere del testo analizzato, eventualmente l'ultimo del testo, coincide con quello che costituisce la stringa prescritta:
rilascio i bus;
evidentemente non c'è bisogno di riavvolgere i puntatori;
attivo richiesta di interruzione ponendo a 1 il bit di stringa trovata.

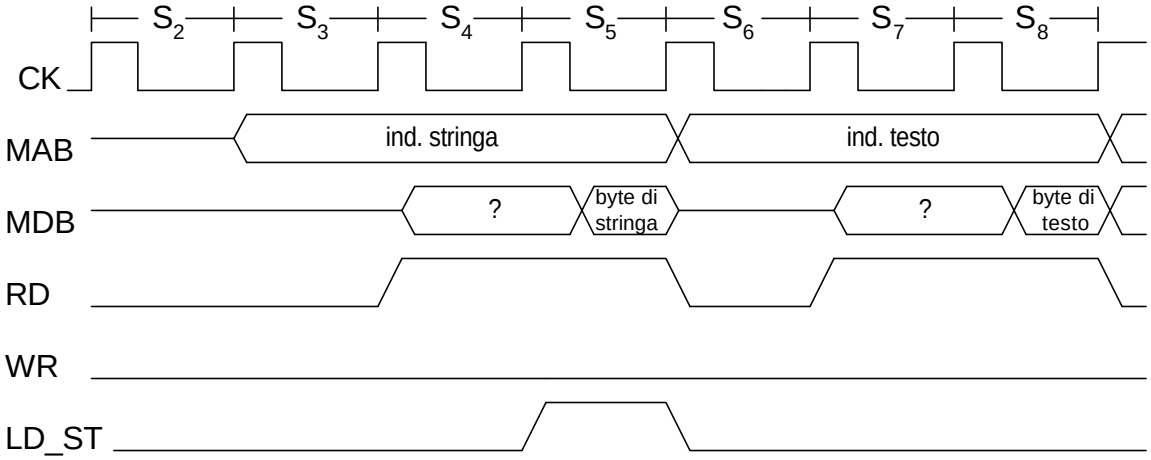
ZERO=1 EOT=1 EOS=0 EQ=0/1
ricerca in corso del primo carattere della stringa prescritta;
il testo è esaurito, la stringa non è stata trovata:
oppure:
ZERO=1 EOT=1 EOS=1 EQ=0
ricerca in corso del primo carattere della stringa prescritta, che è lunga 1 solo carattere!
Il carattere del testo analizzato differisce da quello che costituisce la stringa prescritta:
rilascio i bus;
evidentemente non c'è bisogno di riavvolgere i puntatori;
attivo richiesta di interruzione ponendo a 0 il bit di stringa trovata.

DMAC-stringa: temporizzazioni

Richiesta / rilascio bus di sistema e relativi stati dello SCO



Accessi in RAM e relativi stati dello SCO



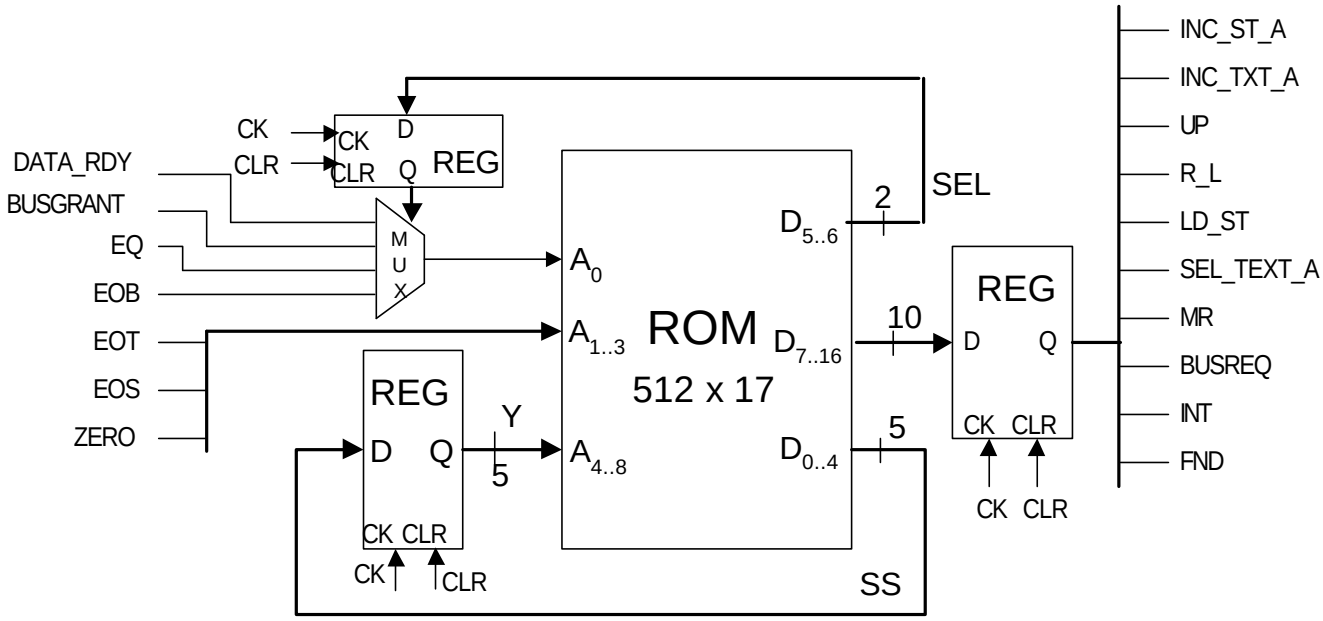
Note

In questo caso (SCO microprogrammato) il diagramma di temporizzazione assume il ruolo di commento; la sua stesura non è essenziale per la sintesi, in quanto la sequenza delle micro-operazioni è descritta dalla sequenza delle microistruzioni.

I cicli di lettura sono a tre stati (S_3 .. S_5; S_6 .. S_8), in quanto si suppone che la RAM abbia $2T < t_A < 3T$: in mancanza di specifiche sulla velocità della periferica si suppone di utilizzare nella periferica un clock a velocità non superiore rispetto a quello del processore.

DMAC-stringa: SCO - struttura HW microprogrammata

L'implementazione di un modello strutturale di tipo D-Mealy con mascheramento degli ingressi comporta la struttura seguente:



Note

- Dopo avere emesso INT il controllo torna in S_0 senza aspettare né INTA né BUSGRANT=0: questi due eventi si verificheranno mentre SCO in S_0 aspetta la successiva commutazione di DATA_RDY=1; infatti, il processore potrà riportare DATA_RDY a 1 soltanto dopo avere servito la richiesta di interruzione, con la quale la periferica segnala al processore la conclusione delle attività.

- Va notato che il meccanismo di comunicazione tra il processore e la periferica mediante interruzione rende superflua l'interfaccia busy-waiting, in quanto dopo avere emesso una richiesta di interruzione la periferica ha certamente concluso l'operazione richiesta dal processore e quindi è certamente pronta ad avviare un'operazione successiva (il processore non ha bisogno di effettuare un test sullo stato di pronto della periferica).

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 14-4-2000

Studente:_____ **Docente:**_____

D1 Un trasmettitore deve trasformare dati di 12 bit in parole di un codice Hamming a distanza 3. Determinare il numero di bit da aggiungere ai 12 bit di dato.

D2 Sintetizzare in logica steering la funzione y = b'c'd'+ acd' + bcd + ac'd.

D3 Descrivere la metodologia per trasformare una macchina sincrona impulsiva in una macchina asincrona con ingressi impulsivi applicata alla sintesi di un contatore up/down modulo 4 che ha due ingressi impulsivi up e down.

D4 Un PD32 ha il ciclo di accesso alla memoria di 50nsec. Calcolare di quanto viene rallentato da un disco che effettua operazione di trasferimento dati in DMA e che abbia una velocità di lettura/scrittura di 64 Mbit/sec. Si supponga di trasferire dati di 32 bit alla volta.

D5 Descrivere la struttura hw ed il protocollo per la sincronizzazione di due sistemi microprogrammati.

Esercizio (2S20000414-D1)

Un trasmettitore deve trasformare dati di 12 bit in parole di un codice Hamming a distanza 3. Determinare il numero di bit da aggiungere ai 12 bit di dato.

Il codice irridondante ha n = 12. Per trasformarlo in un codice di tipo Hamming con h = 3, occorre un numero k di bit di controllo legato a quello n = 12 dalla relazione:

$$12 \leq 2^k - k - 1$$

da cui (per tentativi: k = 1, 2,…) si trova k = 5. Ne segue che il codice ridondante sarà composto da m = 12 + 5 = 17 bit. La parola del codice avrà la forma seguente:

$c_1 c_2\ a_3\ c_4\ a_5\ a_6\ a_7\ c_8\ a_9\ a_{10}\ a_{11}\ a_{12}\ a_{13}\ a_{14}\ a_{15}\ c_{16}$

Esercizio (2S20000414-D2)

Sintetizzare in logica steering la funzione:
y = b'c'd'+ acd' + bcd + ac'd.

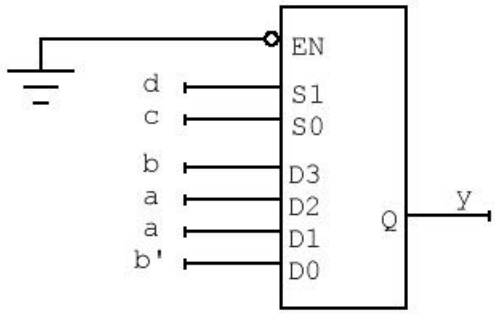
La funzione f specificata può essere riscritta mediante fattorizzazione come:

y = b'c'd'+ acd' + bcd + ac'd =
= d’(b'c' + ac) + d(bc + ac')

A questo punto y può essere implementata con la tecnica impiegata nell’esercizio precedente; oppure si può notare che y ha la forma della funzione di multiplex di due termini, controllata da d, entrambi a loro volta funzioni di multiplex controllate dalla stessa variabile di selezione c. In altri termini, y è la funzione di multiplazione controllata dalle due variabili cd, che compaiono a prodotto nelle quattro combinazioni:

y = b'c'd'+ acd' + ac'd + bcd

Pertanto y può essere implementata con un mux come in figura:



A questo punto il simbolo del mux può essere sostituito con la rete standard che implementa il mux in logica steering (cfr. testo e appunti integrativi).

Esercizio (2S20000414-D3)

Descrivere la metodologia per trasformare una macchina sincrona impulsiva in una macchina asincrona con ingressi impulsivi applicata alla sintesi di un contatore up/down modulo 4 che ha due ingressi impulsivi up e down.

Detti 0, 1, 2, 3 gli stati del contatore, f e b i simboli di ingresso che fanno rispettivamente avanzare e retrocedere il conteggio, la tavola sincrona (impulsiva) del contatore specificato è la seguente (i_0 è il simbolo dello spazio):

	i_0	f	b
0	0	1	3
1	1	2	0
2	2	3	1
3	3	0	2

La tavola della macchina asincrona impulsiva corrispondente è allora:

	i_0	f	b
0_0	$\mathbf{0_0}$	1_1	3_1
0_1	0_0	$\mathbf{0_1}$	$\mathbf{0_1}$
1_0	$\mathbf{1_0}$	2_1	0_1
1_1	1_0	$\mathbf{1_1}$	$\mathbf{1_1}$
2_0	$\mathbf{2_0}$	3_1	1_1
2_1	2_0	$\mathbf{2_1}$	$\mathbf{2_1}$
3_0	$\mathbf{3_0}$	0_1	2_1
3_1	3_0	$\mathbf{3_1}$	$\mathbf{3_1}$

Esercizio (2S20000414-D4)

Un PD32 ha il ciclo di accesso alla memoria di 50nsec. Calcolare di quanto viene rallentato da un disco che effettua operazione di trasferimento dati in DMA e che abbia una velocità di lettura/scrittura di 64 Mbit/sec. Si supponga di trasferire dati di 32 bit alla volta.

Velocità del disco: 64 Mbit/s = 2 MLW/s
cioè il disco trasferisce una longword ogni $1/(2 \cdot 10^6)$ s = 500 ns.
In 500 ns il processore può effettuare 500/50 = 10 cicli macchina; pertanto con il disco attivo ne potrà effettuare 10 –1 = 9.
Pertanto il rallentamento percentuale è dato da 1/10 = 10%.
La distribuzione degli accessi in memoria è tracciata nel grafico seguente:

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
Disc	µP	µP	µP	µP	µP	µP	µP	µP	µP	Disc	µP

Esercizio (2S20000414-D5)

Descrivere la struttura hw ed il protocollo per la sincronizzazione di due sistemi microprogrammati.

Cfr. testo “Reti sequenziali”.

RETI LOGICHE

PRIMA PROVA SCRITTA DEL 5-6-2000

STUDENTE:_____ DOCENTE:_____

Una sistema di cifratura (CFR) e' adibito alla ricezione di burst di 256 dati x_n a 8 bit prelevati in DMA dalla memoria di un PD32 con clock a 150 MHz. Su tale flusso dati CFR esegue le seguenti operazioni:

1. elabora una media mobile su quattro campioni:
$$y_n = \frac{1}{4}\sum_{k=0}^{k=3} x_{n-k}$$
2. Ogni campione y_n deve essere cifrato attraverso l'operazione. $z_n = y_n \oplus h_{n-3}$; la costante h_0 dovrà andare in xor con y_3 che è la prima media mobile completa.
3. Le medie mobili cifrate complete dovranno essere trasmesse su una linea seriale alla velocità di 400 Mbit/s senza soluzione di continuità.

I valori h_n sono costanti a 8 bit mantenuti in una ROM da 256 bytes con tempi di accesso di 30 ns.

L'inizio di tutte le operazioni verrà dato dal PD32 attraverso un segnale di START.

I 256 campioni x_n vengono prelevati dalla memoria del PD32, a partire da un indirizzo noto a CFR tramite accesso in DMA a burst (il bus viene rilasciato solo dopo aver letto tutti i campioni).

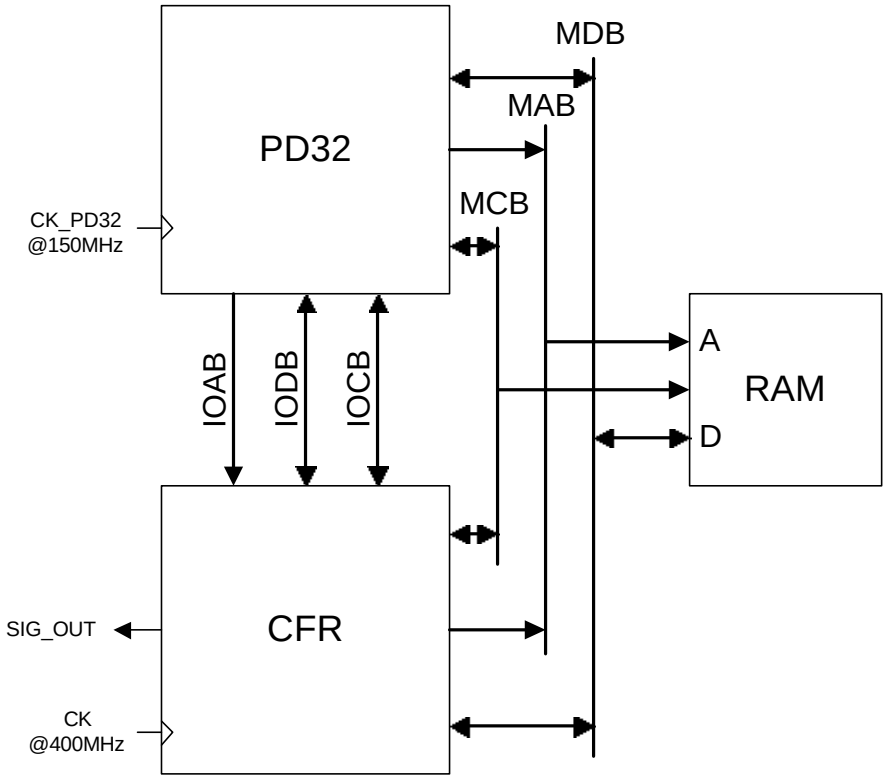
Il processo di cifratura termina una volta trasmesse tutte le medie mobili complete su linea seriale. Tale terminazione viene notificata al PD32 tramite interruzione.

Si richiede:

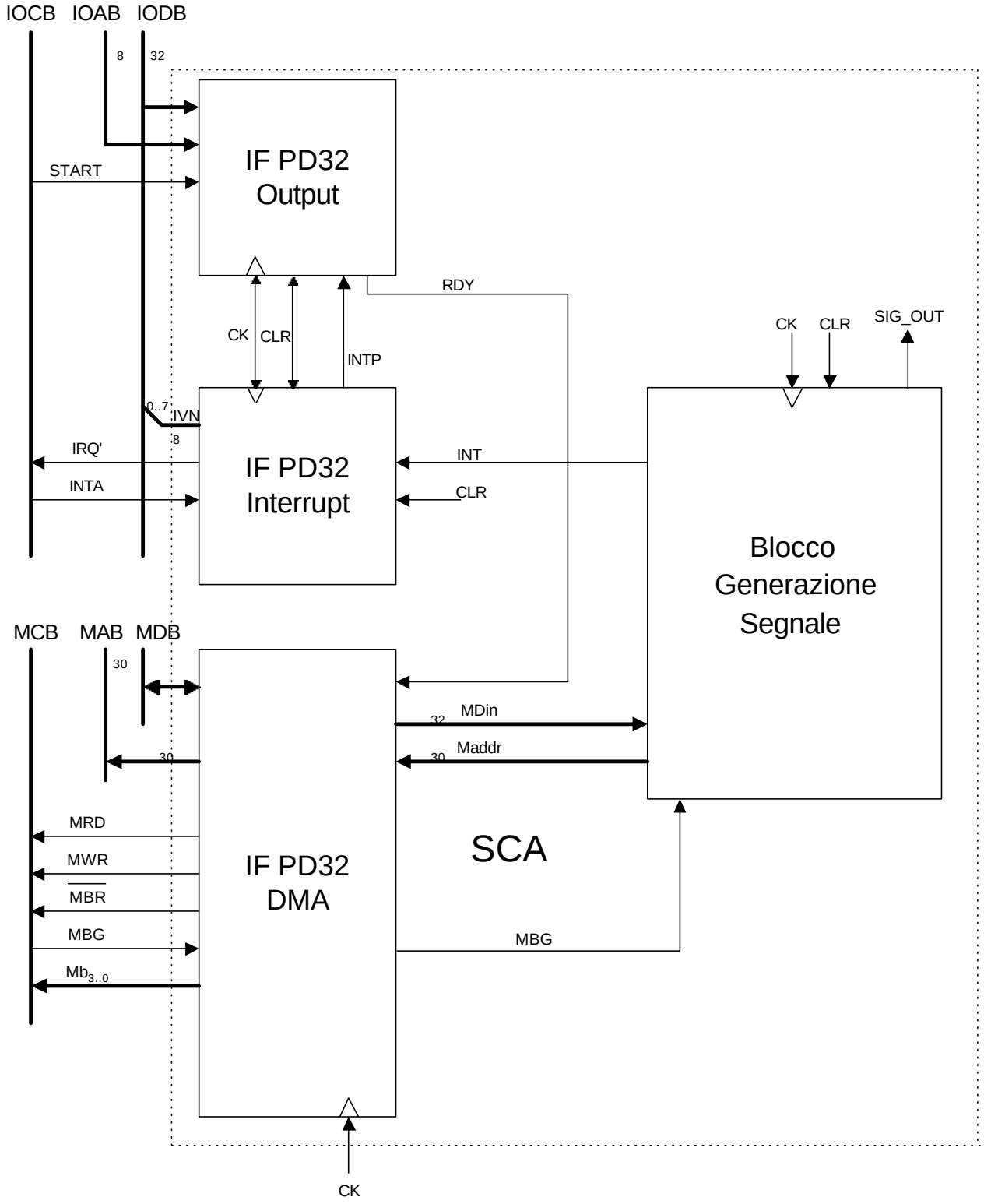
1. Lo schema di interfacciamento CFR-PD32
2. La temporizzazione delle operazioni da effettuare

Il progetto dettagliato di CFR

CFR: sistema esterno



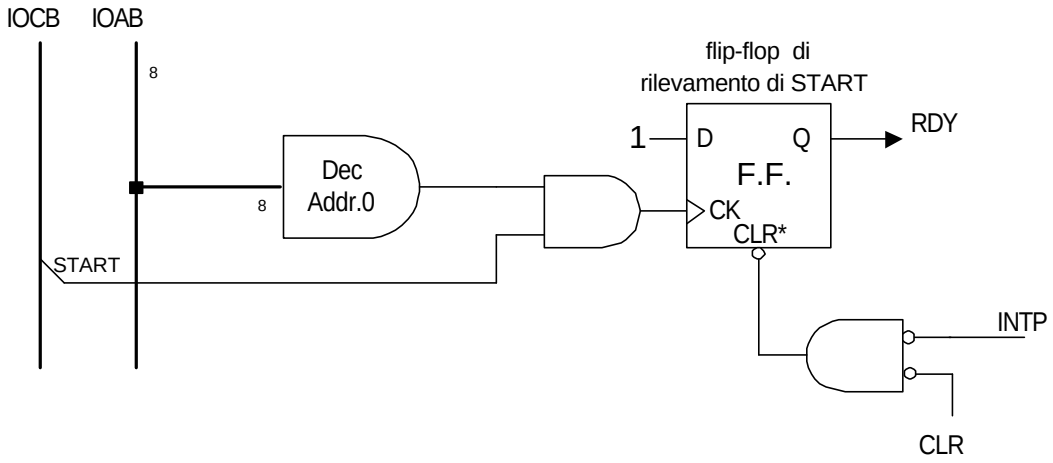
CFR: Schema a blocchi



Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.
Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

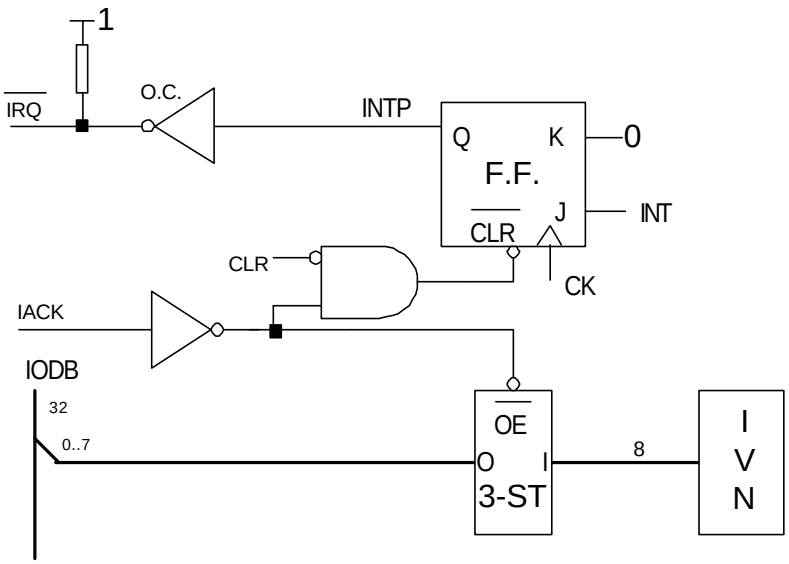
CFR: IF PD32 - output



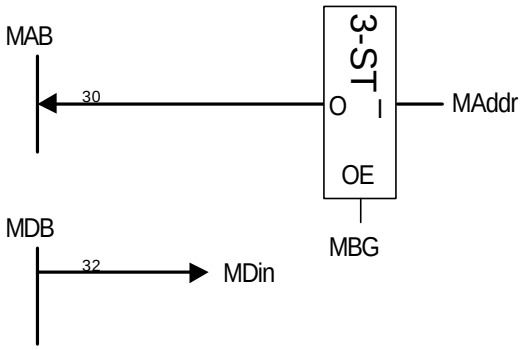
Note

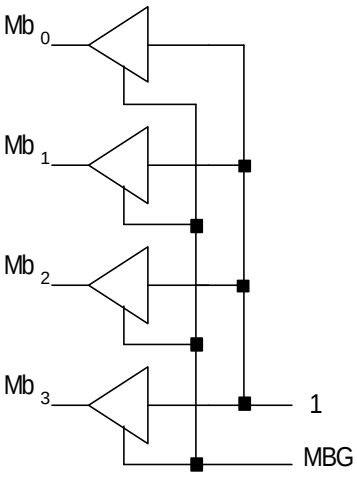
Il SW avvia l'operazione semplicemente con **START Addr0**

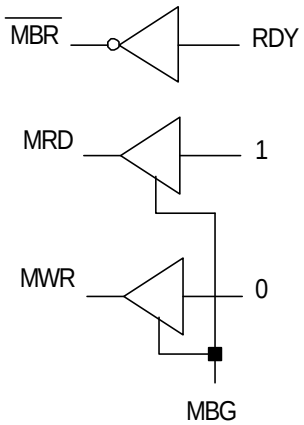
CFR: IF PD32 - interrupt



CFR: IF PD32 - DMA



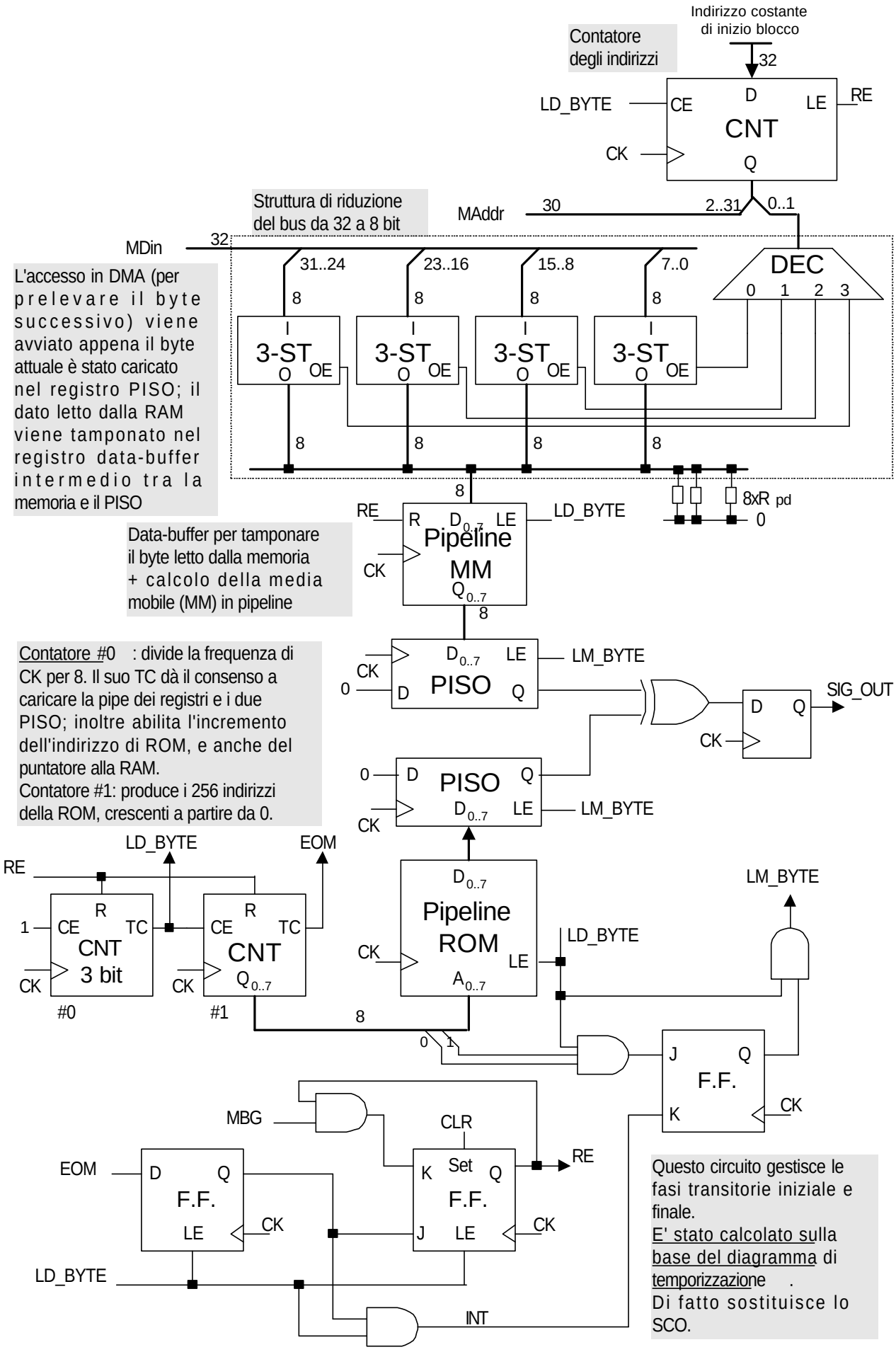




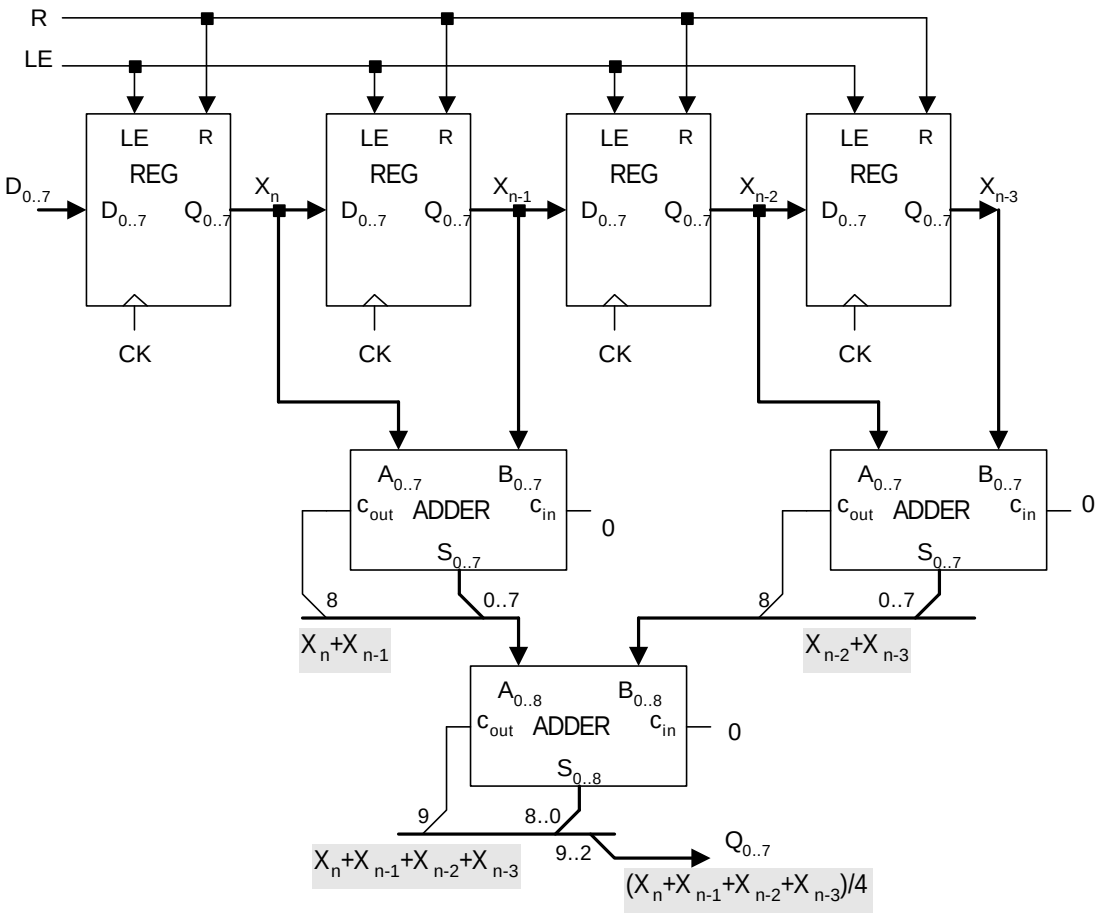
Dalla RAM vengono lette LW, di cui però viene registrato un byte alla volta.

MWR deve essere pilotato (a 0), anche se non dinamicamente: la struttura di memoria deve essere governata pienamente dall'interfaccia DMA, che ne ha piena responsabilità quando MBG=1.

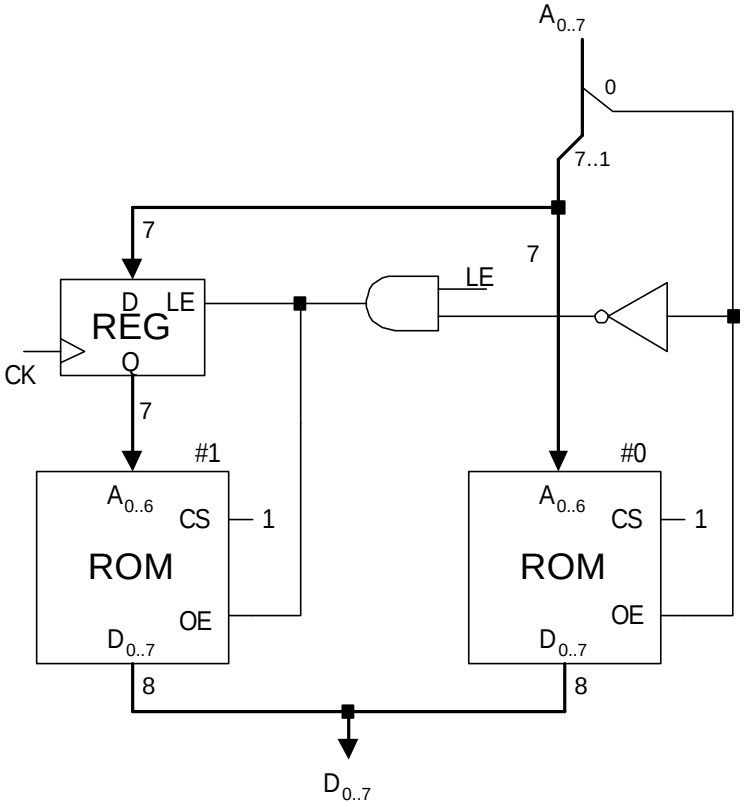
CFR: blocco generatore segnale

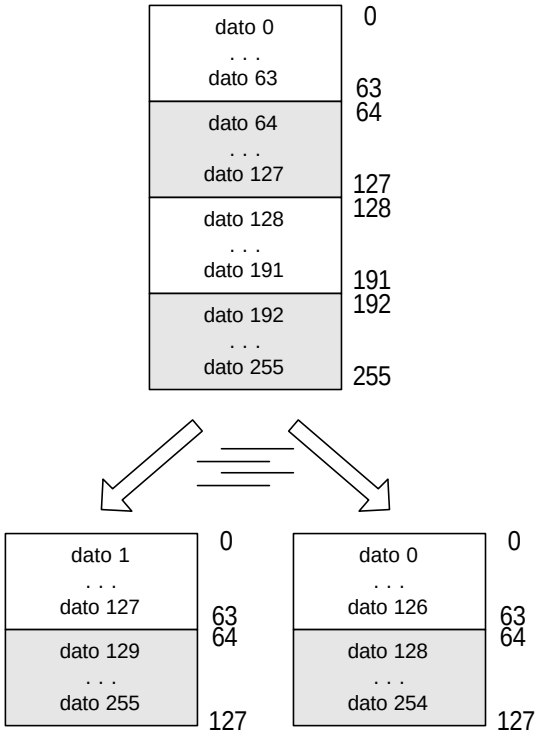


CFR: Pipeline MM



CFR: pipeline ROM



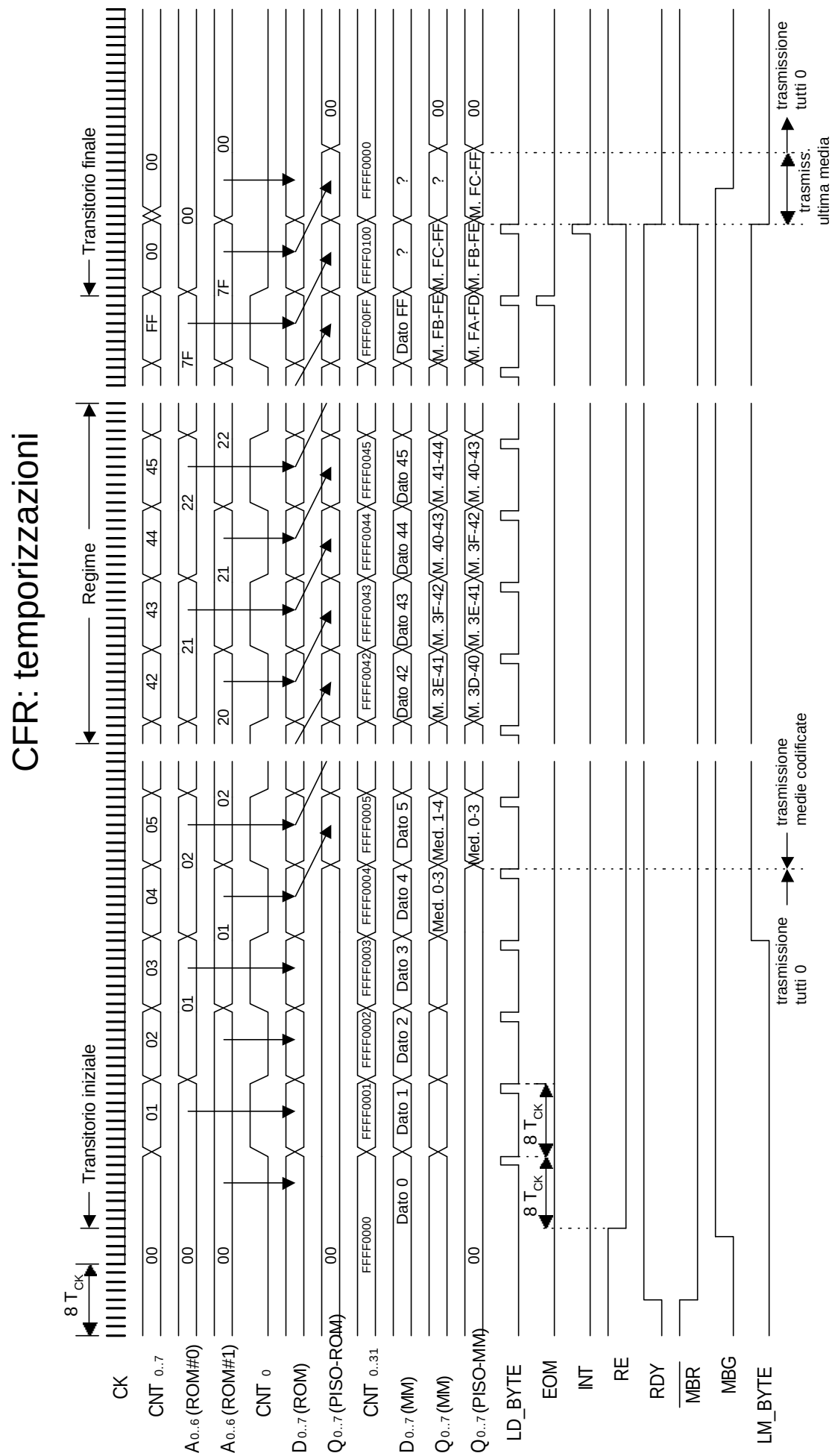


Strutture virtuale (blocchi sequenziali) e fisica (blocchi parallelizzati - pipeline) della ROM.

La struttura di pipeline parallelizza 2 moduli di ROM; infatti: $t_{CK} < t_A < t_{CK} \times 2$, essendo t_A=150ns e t_{CK}=122ns.

Nel caso generale di accesso *casuale* ai due moduli ROM, ciascuno di capacità 256 x 8 bit, dovrebbero essere predisposti due registri di ingresso da caricare a cicli di clock alterni con il dato di uscita del contatore; in questo caso specifico invece l'accesso alle ROM deve essere di tipo *sequenziale*, in particolare dall'indirizzo 0 a 255; ciò implica che in una struttura pipeline con 2 ROM da 256 indirizzi da una ROM verrebbero letti soltanto i dati disposti agli indirizzi pari, e dall'altra ROM soltanto i dati di indirizzo dispari; pertanto è possibile e conveniente memorizzare soltanto la metà dei dati in ciascuna delle due ROM: ad esempio nella ROM #0 i dati relativi agli indirizzi pari (0, 2, .., 254) generati dal contatore, nella ROM #1 i dati relativi agli indirizzi dispari (1, 3, .., 255); ovviamente i dati (byte) memorizzati in ciascuna ROM sono 256/2=128 (7 bit di indirizzo).
Gli indirizzi di ROM, variabili da 0 a 127 ogni due cicli di CK, sono disponibili sulle uscite 1..7 del contatore; pertanto è possibile connettere tali uscite direttamente all'ingresso della ROM #0, mentre per ritardarne di un periodo di CK la presentazione al modulo #1 deve essere introdotto un registro (di pipeline) in cascata al contatore, con l'ingresso di abilitazione a caricamento collegato al bit 0 (LSB) invertito del contatore.
Le uscite delle due ROM dovranno essere selezionate alternativamente al ritmo di CK; in questo caso è conveniente pilotare i buffer 3-state (disposti sul livello di uscita) delle ROM tramite lo stesso bit 0 (LSB) del contatore; va notato che quest'ultimo commuta ad ogni ciclo di CK, e quindi ci si pone nell'ipotesi ragionevole che la ROM risponda all'attivazione di OE entro un singolo periodo di CK.

CFR: temporizzazioni



CFR: temporizzazioni

Note

I valori indicati per Q0..7 (PISO-ROM) e Q0..7 (PISO-MM) sono relativi ai dati caricati nei due registri PISO, e quindi vanno riferiti al primo degli otto peri odi di clock in cui quei dati vengono scalati. Nelle posizioni vacanti entrano 0.

Il diagramma di temporizzazione evidenzia che:
la prima media completa (Med. 0-3) caricata in PISO-MM va composta (mediante XOR bit a bit) con il codice allineato temporalmente in PISO-ROM, che proviene dalla ROM#1 all'indirizzo 01;
l'ultima media completa (Med. FC-FF) caricata in PISO-MM va composta (mediante XOR bit a bit) con il codice allineato temporalmente in PISO-ROM, che proviene dalla ROM#1 all'indirizzo 7F.
Pertanto vengono codificate 253 medie complete, prima (cfr. segnale LM_BYTE) e dopo le quali sulla linea di trasmissione vengono inviati 0.
I 253 codici cifranti vanno memorizzati nelle due ROM nel seguente ordine:

$A_{0..6}$	0	1	2	3	...	127
ROM#0:	-	-	codice1	codice3	...	codice251
ROM#1:	-	codice0	codice2	codice4	...	codice252

Gli accessi alla RAM di sistema vengono eseguiti con il ritmo necessario a rifornire il registro PISO preposto allo scalamento seriale continuo dei dati, che devono uscire alla velocità di 400 Mbit/s, cioè 50 MByte/s, o 25 Mword/s, o 12.5 MLW/s. Dalla specifica è noto che il micro accede alla RAM con un clock di frequenza pari a 150 MHz, e poiché l'accesso è distribuito su 3 cicli di clock, allora ciò implica che la RAM può sopportare 150/3=50 Maccessi/s. Se ne deduce che il DMAC può leggere la RAM in DMA a burst alla velocità di 50 MByte/s: durante il ciclo di DMA mantiene la linea MR=1, aggiorna il contatore degli indirizzi e carica i dati nel registro PISO ogni 8 (= 400Mbit/s / 50MByte/s) cicli del proprio clock.
Va notato che l'accesso alla RAM a burst consente la connessione diretta tra il canale di memoria e il registro a scalamento, senza l'interposizione di un registro tampone, necessario al contrario negli accessi in stealing per tamponare il dato prelevato dalla memoria in attesa di essere trasferito nel registro (a scalamento) di uscita.

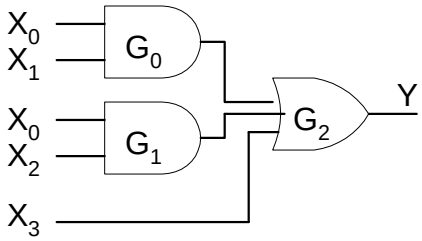
RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 5-6-2000

Studente:_____ Docente:_____

D1 Definire il formato numerico in virgola fissa per rappresentare i numeri relativi compresi tra –128.0 e 127.0 con un errore assoluto inferiore a 0.1.

D2 La porta G0 ha t_{min}=2 ns, t_{max}=4 ns, la porta G1 ha t_{min}=4 ns, t_{max}=6 ns, la porta G2 ha t_{min}=3 ns, t_{max}=5 ns; descrivere il diagramma di temporizzazione relativo ai nodi intermedi e di uscita della rete di figura a partire dalla variazione X_{3210}: 1010 $\rightarrow$ 0101 del vettore di ingresso.



D3 Progettare una rete sequenziale LLC dotata di un ingresso X e un'uscita Y che reagisce alla sequenza di ingresso 11 con la sequenza (01)*, cioè un'oscillazione con periodo 2T (T=periodo del clock) e torna a riposo (Y=0) alla ricezione di 00.

D4 Dimensionare una struttura microprogrammata di tipo D-Mealy per supportare un microprogramma con |X|=6, |S|=80, |Z|= 22, descritto con il microlinguaggio di tipo 3. Si suppongano disponibili moduli ROM a 8 bit di dato.

D5 Dare una esemplificazione di ciascuno dei modi di indirizzamento del PD32 e valutare il numero dei cicli-macchina richiesti da ciascun ciclo-istruzione.

Esercizio (2S200006005-D1)

Definire il formato numerico binario in virgola fissa per rappresentare i numeri relativi compresi tra –128.0 e 127.0 con un errore assoluto inferiore a 0.1.

Parte frazionaria

La limitazione sull'errore assoluto $\varepsilon < 10^{-1}$ può essere riscritta nella base 2 come: $\varepsilon < 2^{-4} < 10^{-1} < 2^{-3}$, da cui si ricava che la parte frazionaria della rappresentazione binaria è composta da almeno 4 bit ($D_3D_2D_1D_0$).

Parte intera

La dinamica specificata –128..+127 può essere rappresentata con 8 bit in complemento a 2 ($I_7I_6I_5I_4I_3I_2I_1I_0$).

In definitiva il formato minimo della rappresentazione numerica ricercata è dato da:

$$I_7I_6I_5I_4I_3I_2I_1I_0.D_3D_2D_1D_0$$

****

Qualche esempio di conversione 2 $\rightarrow$ 10:

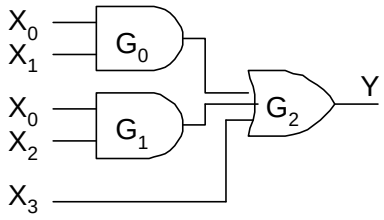
00000001.1011	1.6875
01111110.0011	126.1875
10000000.1101	-127.1875
11111111.0001	-0.9375

Qualche esempio di conversione 10 $\rightarrow$ 2:

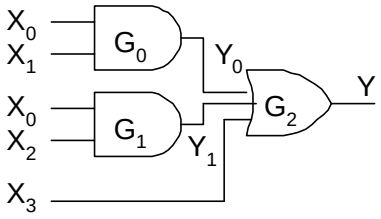
1.78	00000001.1100	(1.75)
126.19	01111110.0011	(126.1875)
-0.12	11111111.1110	(-0.125)
-126.9	10000001.0001	(-126.9375)

Esercizio (2S20000605-D2)

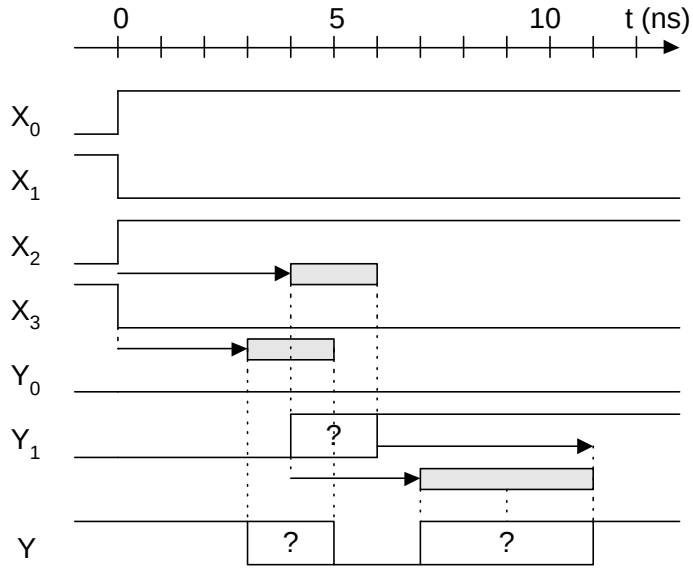
La porta G0 ha t_{min}=2 ns, t_{max}=4 ns, la porta G1 ha t_{min}=4 ns, t_{max}=6 ns, la porta G2 ha t_{min}=3 ns, t_{max}=5 ns; descrivere il diagramma di temporizzazione relativo ai nodi intermedi e di uscita della rete di figura a partire dalla variazione X_{3210}: 1010 $\rightarrow$ 0101 del vettore di ingresso.



Detti Y0 e Y1 i due nodi intermedi della figura:



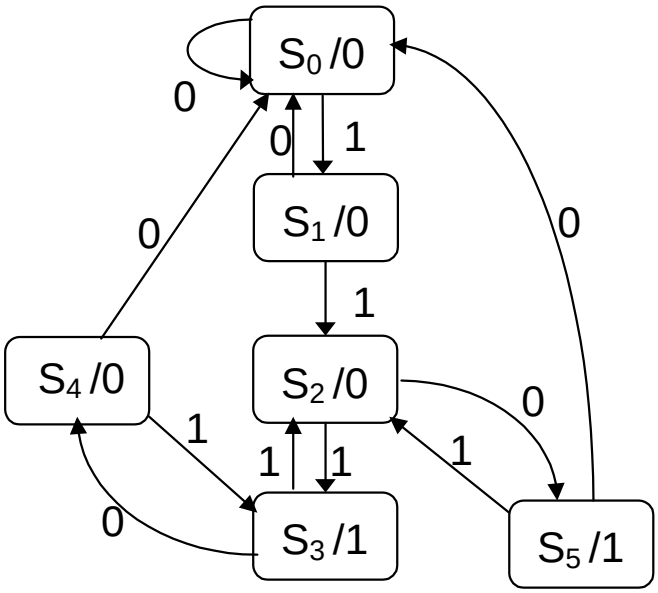
il diagramma di temporizzazione richiesto può essere tracciato come di seguito:



Esercizio (2S20000605-D3)

Progettare una rete sequenziale LLC dotata di un ingresso X e un'uscita Z che reagisce alla sequenza di ingresso 11 con la sequenza (01)*, cioè un'oscillazione con periodo 2T (T=periodo del clock) e torna a riposo (Y=0) alla ricezione di 00.

La macchina sequenziale può essere descritta con il diagramma degli stati seguente:



Una volta rilevata la sequenza 11, la macchina entra in una oscillazione sincrona e cadenzata dal clock (sistema LLC) tra gli stati S2 e S3, fintantoché X=1; poiché deve continuare a oscillare anche in presenza di un simbolo di ingresso 0 isolato (cioè preceduto e seguito dal simbolo 1), gli stati S2 e S3 vanno replicati in due stati, S4 e S5 rispettivamente, ai quali si perviene dopo avere ricevuto la sequenza 10 e dai quali si può rientrare nella coppia S2 e S3 con 1, oppure uscire dall'oscillazione e tornare nello stato di riposo S0 per la sequenza di ingresso 00.

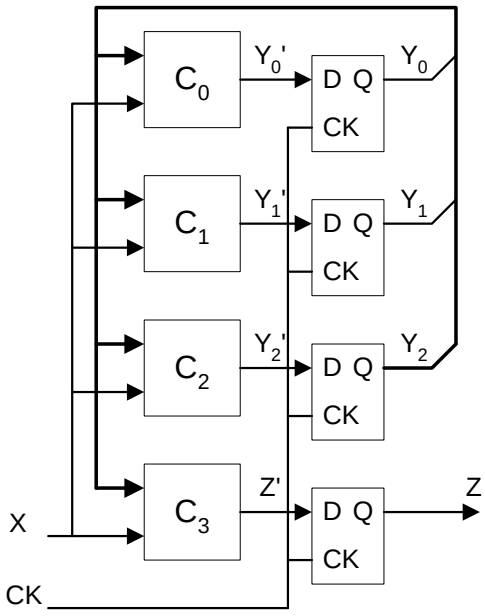
Per avere transizioni nette del segnale di uscita (oscillatore abilitato) conviene sintetizzare la rete con riferimento al modello di D-Mealy, piuttosto che di Moore:

S	Y2 Y1 Y0	Y2' Y1' Y0' / Z' @ X=0	Y2' Y1' Y0' / Z' @ X=1	Z
S0	0 0 0	0 0 0 / 0	0 0 1 / 0	0
S1	0 0 1	0 0 0 / 0	0 1 0 / 0	0
S2	0 1 0	1 0 1 / 1	0 1 1 / 1	0
S3	0 1 1	1 0 0 / 0	0 1 0 / 0	1
S4	1 0 0	0 0 0 / 0	0 1 1 / 1	0
S5	1 0 1	0 0 0 / 0	0 1 0 / 0	1

Si noti che i valori dell'uscita inclusi nei simboli di stato (Moore) sono associati alle transizioni (D-Mealy) che hanno quegli stati come destinazione.

La sintesi completa richiede la minimizzazione con 4 MK (le tre variabili di stato + l'uscita) a 4 variabili (le tre variabili di stato + l'ingresso); si noti anche che l'uscita calcolata con il modello di Moore dipenderebbe dalle sole 3 variabili di stato.

La rete ha la struttura seguente:

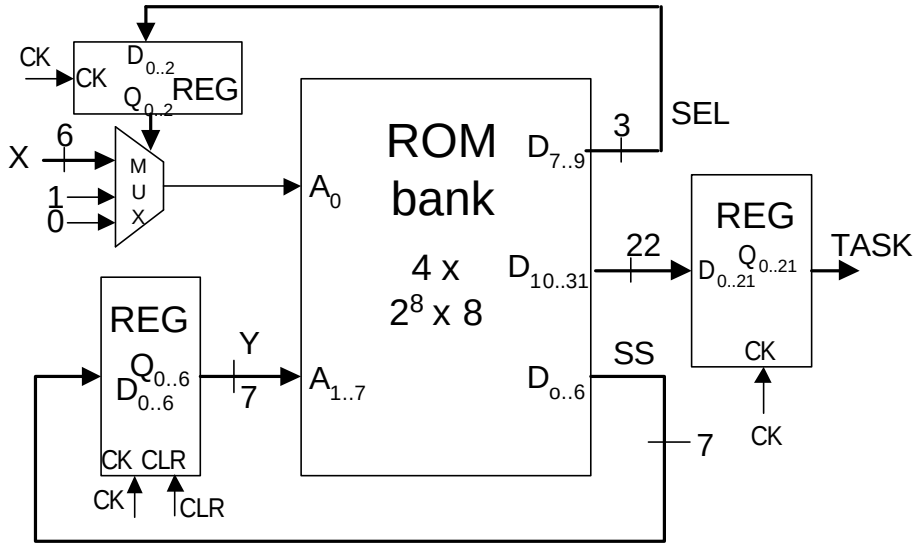


Il calcolo delle quattro reti combinatorie è lasciato come esercizio al lettore, sulla base del contenuto della tavola degli stati / uscite precedente.

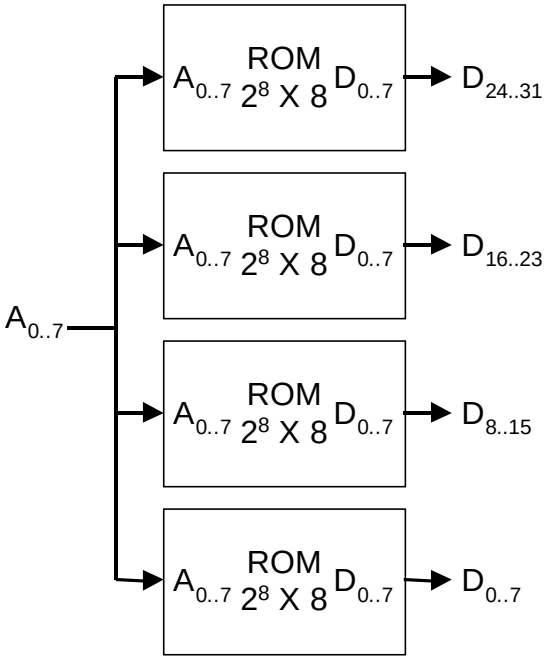
Esercizio (2S20000605-D4)

Dimensionare una struttura microprogrammata di tipo D-Mealy per supportare un microprogramma con |X|=6, |S|=80, |Z|= 22, descritto con il microlinguaggio di tipo 3. Si suppongano disponibili moduli ROM a 8 bit di dato.

Il dimensionamento dei componenti della struttura generale di D-Mealy che supporta il microlinguaggio di tipo 3 è effettuato sulla base dei numeri specificati: |X|=6, |Y|=7, |Z|= 22:



Dove il banco di ROM provvede all'espansione dei dati:



Esercizio (2S20000605-D5)

Dare una esemplificazione di ciascuno dei modi di indirizzamento del PD32 e valutare il numero dei cicli-macchina richiesti da ciascun ciclo-istruzione.

Modo 0: diretto a registro

MOVL R0,R1

Cicli-macchina: 1: fetch.

Modo 1: immediato

MOVW #05A71h,R1

Cicli-macchina: 2: fetch, prelevamento della costante (word) posizionata dopo il codice istruzione.

Modo 2: assoluto

MOVB 0FF005A00h,R1

Cicli-macchina: 3: fetch, prelevamento dell'indirizzo (longword) posizionato dopo il codice istruzione, lettura dell'operando all'indirizzo specificato.

Modo 3: indiretto con registro

MOVL R2,(R1)

Cicli-macchina: 2: fetch, scrittura del dato contenuto in R2 all'indirizzo contenuto in R1.

Modo 4: con spiazzamento

MOVW R2,base(R1)

Cicli-macchina: 3: fetch, prelevamento dell'indirizzo (longword) posizionato dopo il codice istruzione, scrittura del dato contenuto in R2 all'indirizzo contenuto in R1 aumentato di *base*.

Modo 5: relativo

JMP offset(PC)

Cicli-macchina: 2: fetch, prelevamento della costante ‘offset’ (longword) posizionata dopo il codice istruzione.

Modo 6: con predecremento

MOVW R1,-(R5)

Cicli-macchina: 2: fetch, scrittura del dato contenuto in R1 all’indirizzo ottenuto decrementando (di 2, in questo caso) il contenuto in R5.

Modo 7: con postincremento

MOVW R1,(R5)+

Cicli-macchina: 2: fetch, scrittura del dato contenuto in R1 all’indirizzo contenuto in R5. Va notato che R5 verrà incrementato (di 2, in questo caso) dopo l’accesso in memoria.

RETI LOGICHE

PRIMA PROVA SCRITTA DEL 14-06-2000

STUDENTE:_____ DOCENTE:_____

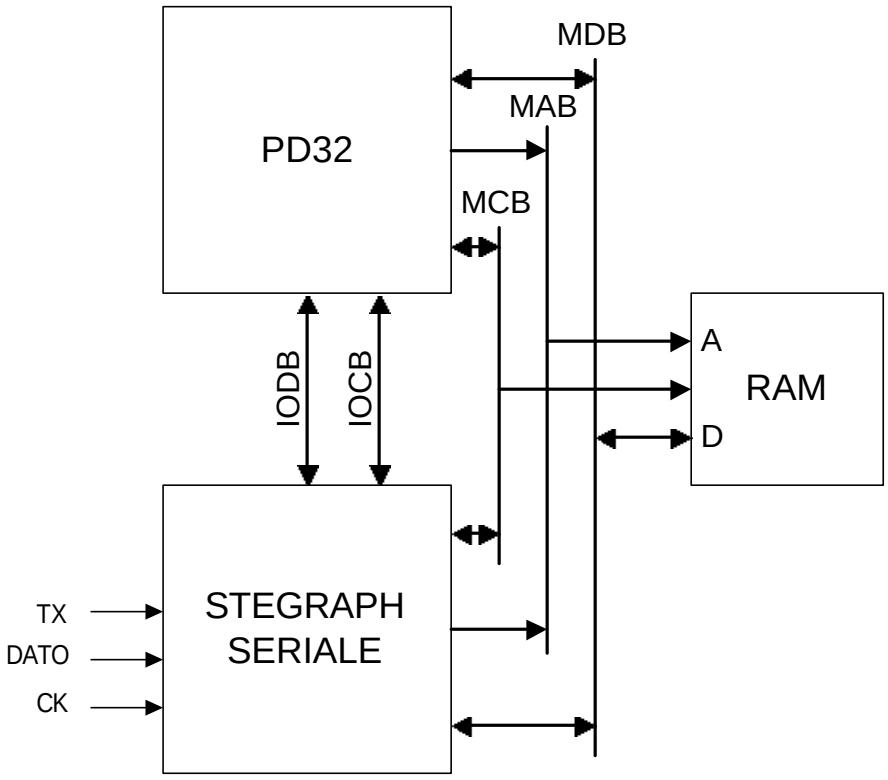
Una periferica di PD32 riceve dati seriali ad una velocità di 20 Mbit/s su tre linee: linea *dato*, linea *clock*, e linea *trasmissione*. La linea trasmissione va ad 1 sincrona con il clock e vi resta per tutto il tempo di trasmissione dati.

Ogni byte rappresenta un pixel di una immagine steganografata, vale a dire che il bit meno significativo di ogni pixel è un bit di un testo. La periferica deve estrarre i byte di testo e trasferire in memoria di lavoro del PD32 sia il file immagine, sia il file testo a partire da due indirizzi di memoria noti alla periferica. Alla fine delle operazioni la periferica invia una interruzione al processore. Il clock del PD32 è di 25 MHz. Si supponga che i bit dei pixel e quelli dei byte di testo vengano presentati sulla linea *dato* con pesi decrescenti.

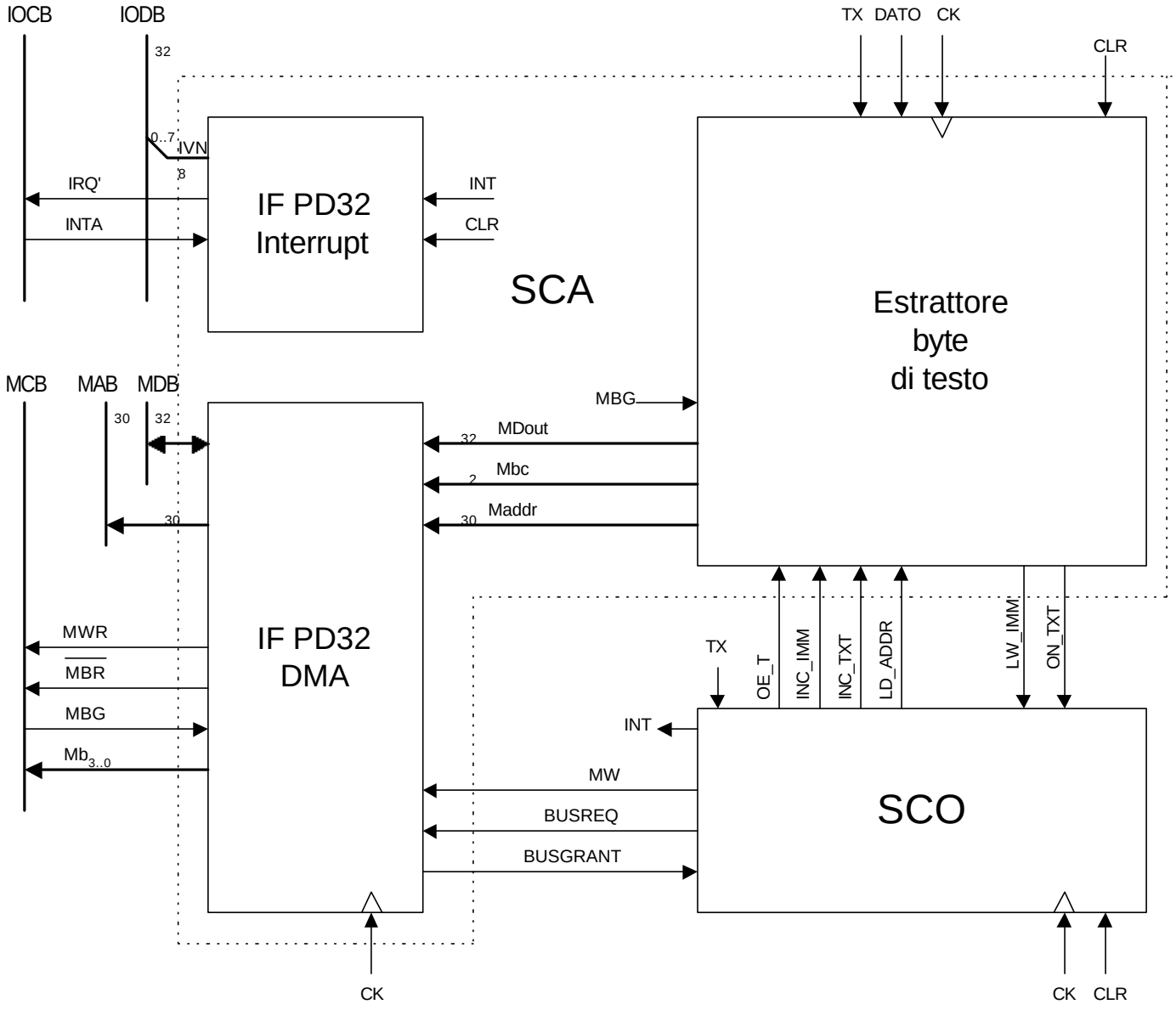
Si richiede:

1. Lo schema a blocchi del sistema periferica-PD32
2. La temporizzazione delle operazioni
3. Lo schema logico della periferica.

STEGRAPH seriale: sistema esterno



STEGRAPH SERIALE: Schema a blocchi

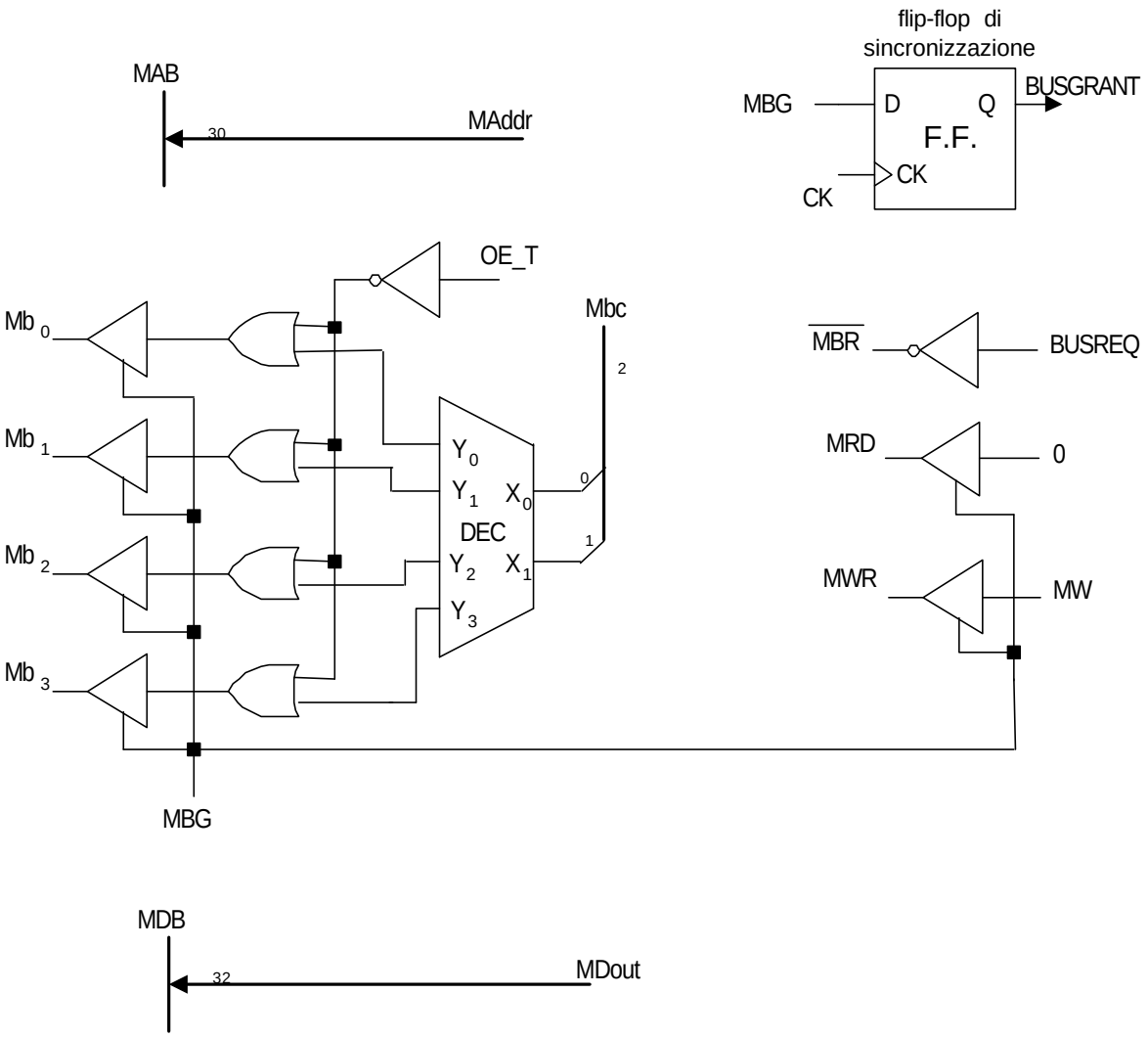


Note

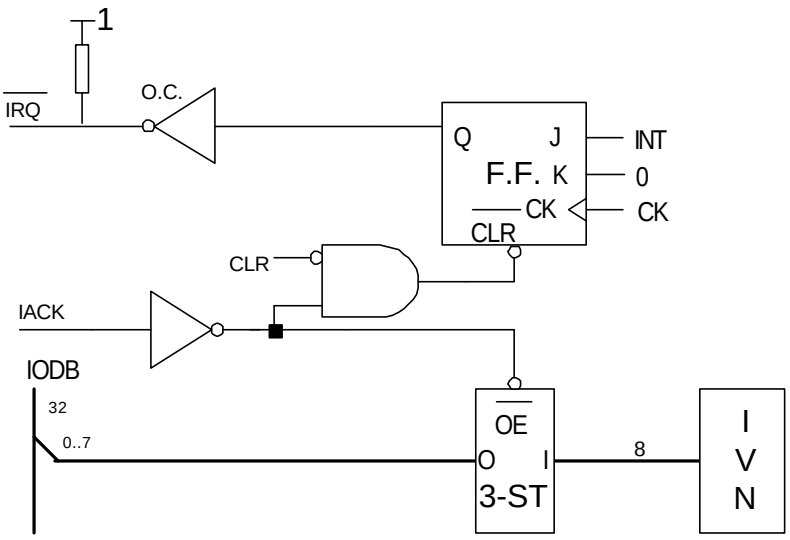
Il segnale CK della linea di comunicazione definisce la risoluzione temporale degli eventi, e quindi può essere utilizzato come segnale di clock di tutto il sistema.

Il blocco IF PD32 interrupt ha l'ingresso CLR asincrono diretto al flip-flop di richiesta di interruzione, per evitare il rischio di una falsa segnalazione di richiesta di servizio al processore all'inizio dell'attivazione della periferica.

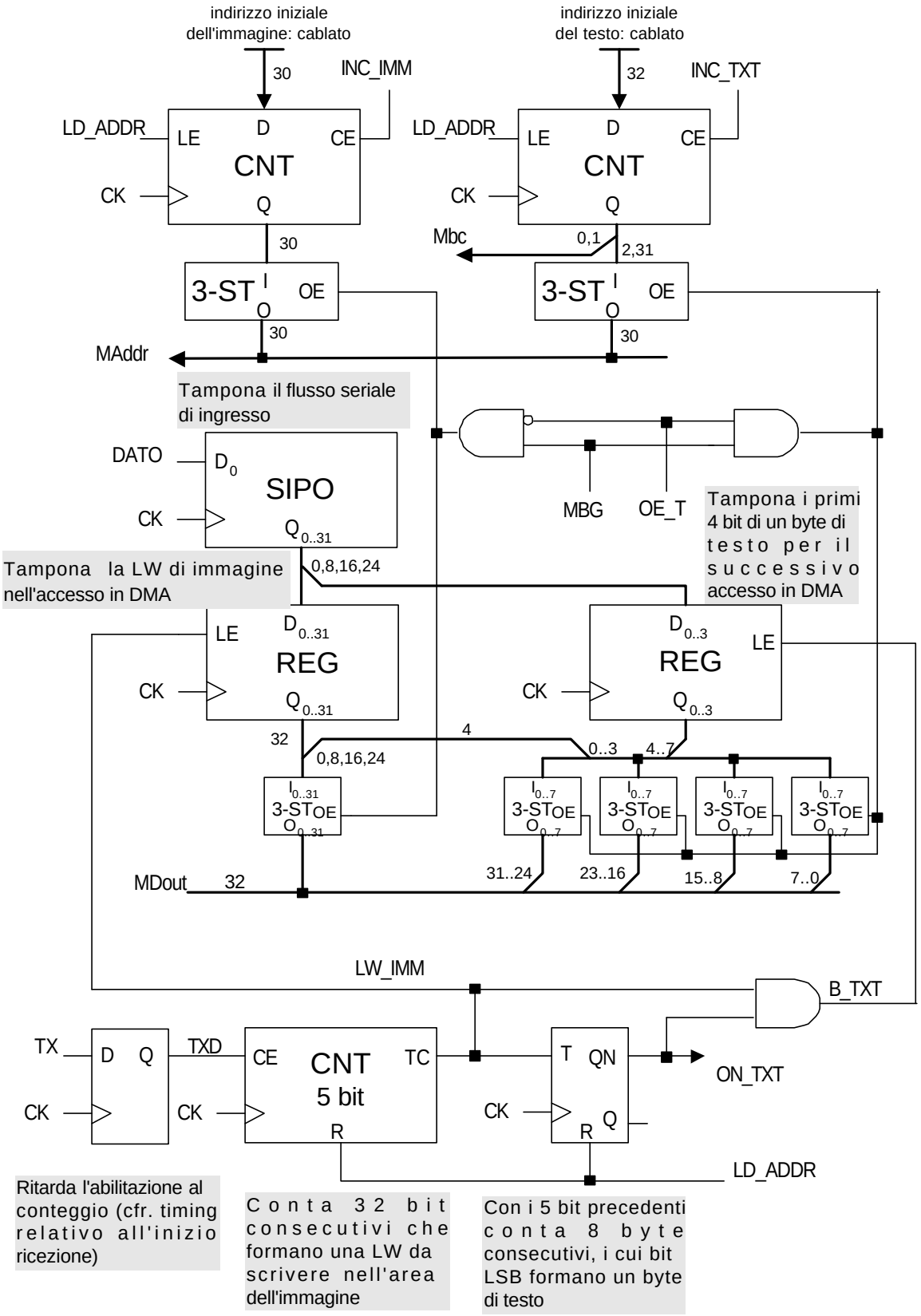
STEGRAPH SERIALE: IF PD32 - DMA



STEGRAPH SERIALE: IF PD32 - interrupt

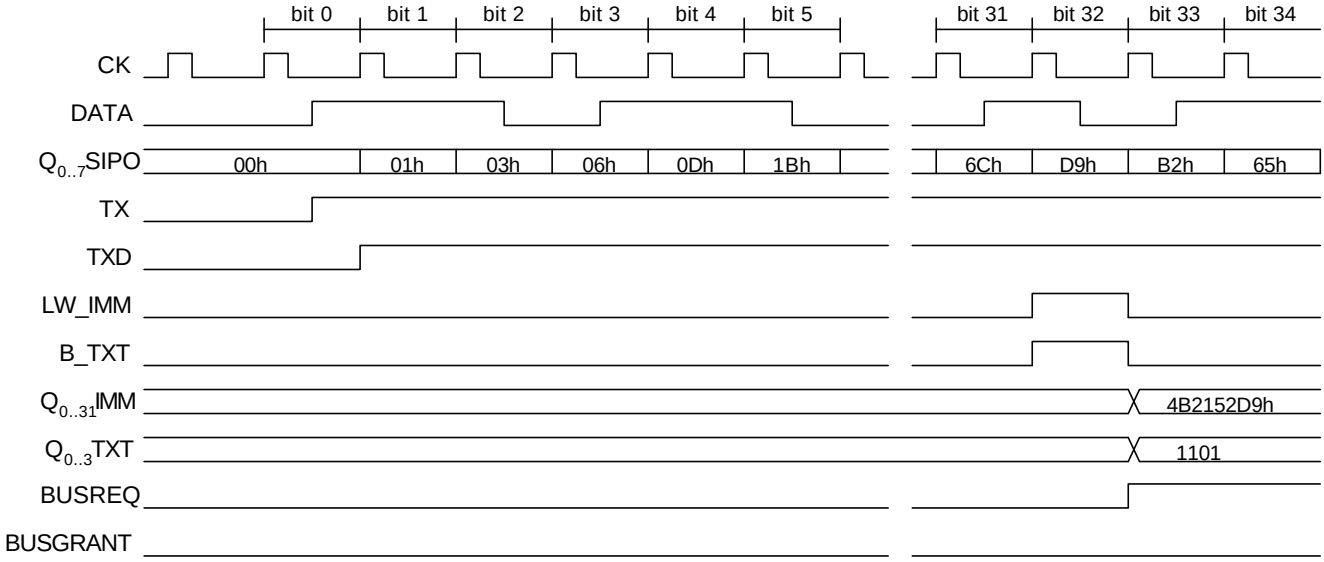


STEGRAPH SERIALE: blocco gestione puntatori

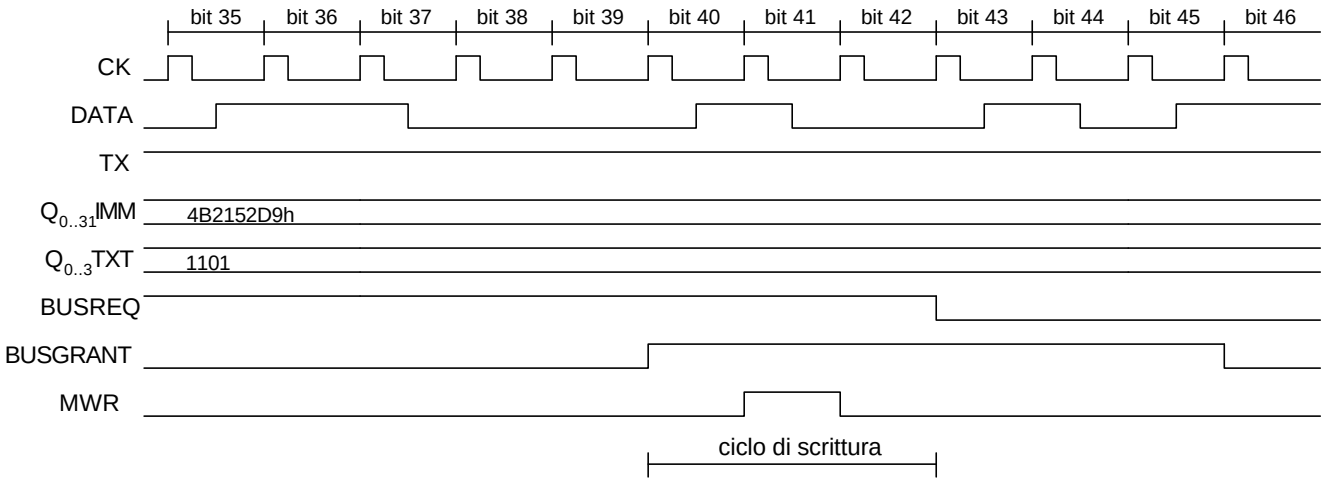


STEGRAPH SERIALE: temporizzazioni

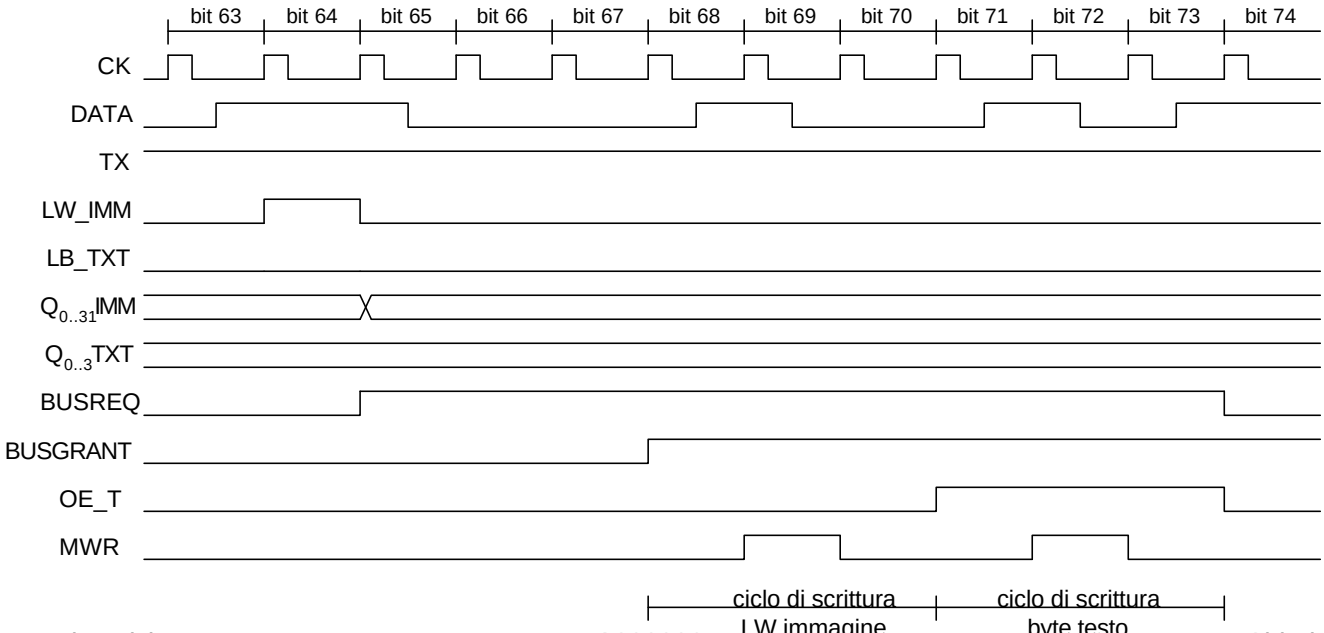
Inizio ricezione



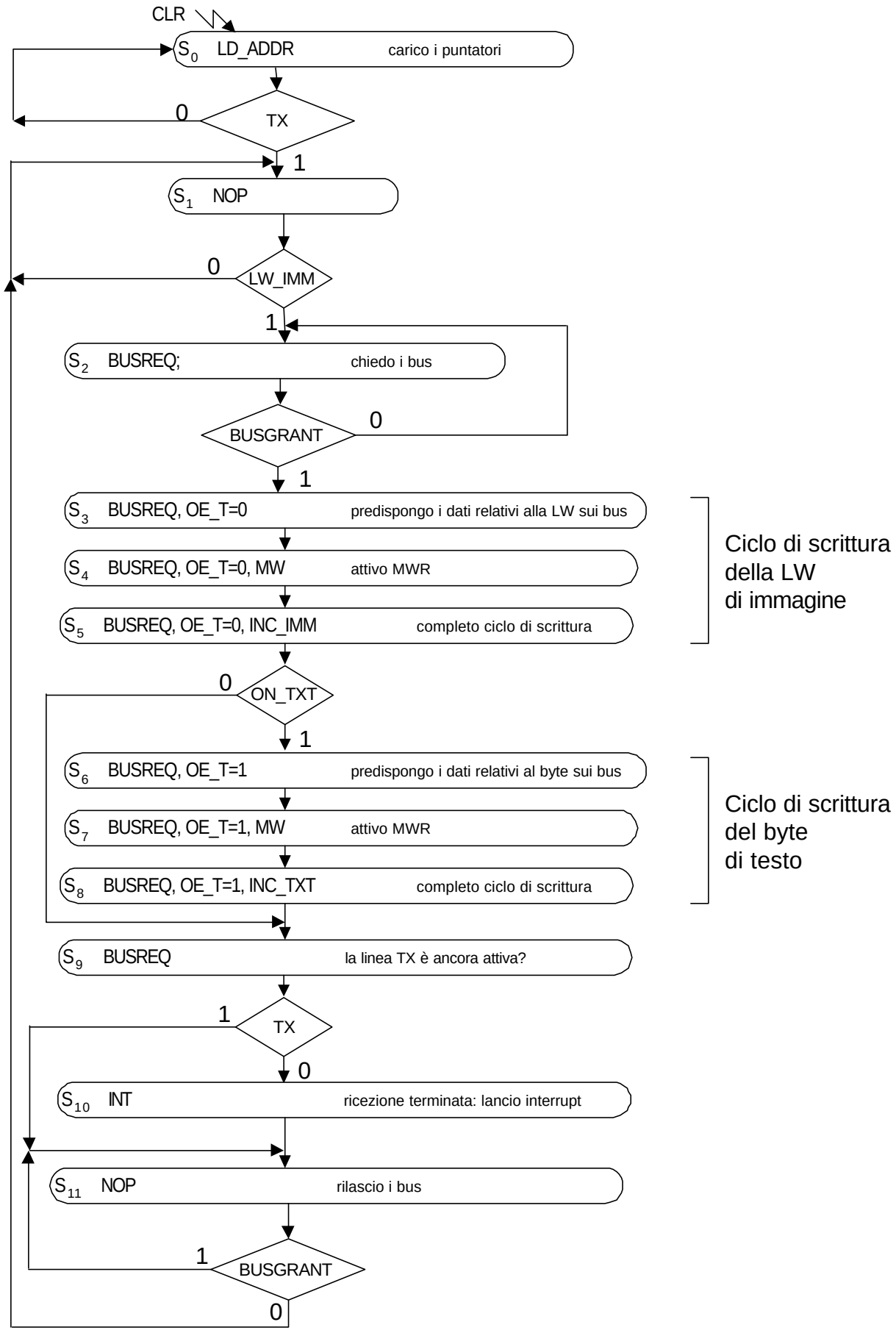
Accesso in DMA: scrittura di una LW di immagine



Accesso in DMA: scrittura di una LW di immagine e di un byte di testo

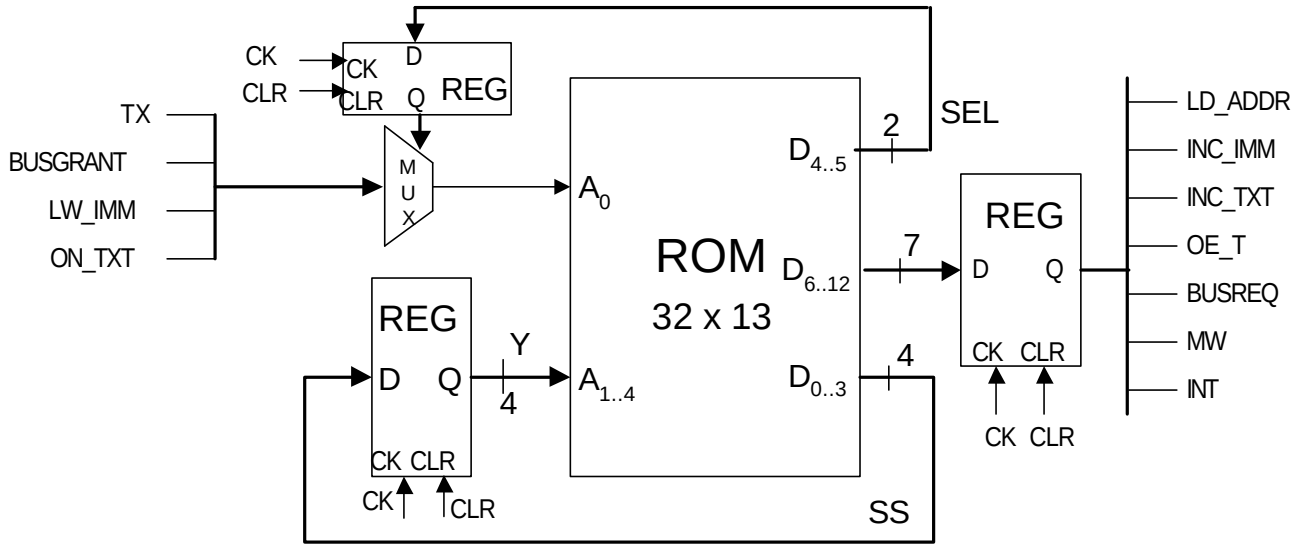


STEGRAPH SERIALE: SCO - flowchart



STEGRAPH SERIALE:
SCO - struttura HW microprogrammata

Il flow-chart è supportato da un microlinguaggio di tipo 3; scegliendo il modello strutturale di tipo D-Mealy si ottiene la struttura seguente:



Note

- Il segnale di clock della periferica è quello CK della linea di trasmissione, in quanto il suo periodo definisce la risoluzione temporale degli eventi significativi all'interno della periferica: l'intervallo che separa la memorizzazione di due bit consecutivi.

- La frequenza del clock del processore (25 MHz) è maggiore di quella della periferica (20 MHz), cioè il periodo di clock della periferica è maggiore di quello del clock del micro; pertanto la periferica può eseguire cicli di memoria di durata pari a 3 periodi del suo clock.

- Il dimensionamento del registro SIPO, a 32 bit, è determinato dalle seguenti considerazioni:
  - la periferica deve accedere alla memoria in DMA;
  - è possibile e perciò preferibile effettuare l'accesso in DMA con modalità "stealing"; questo evita di bloccare il processore durante il trasferimento (di cui non è nota a priori la durata);
  - il registro SIPO disaccoppia il flusso entrante in tempo reale dal dato che deve essere presentato in memoria in modo stabile durante il ciclo di scrittura; tale dato può essere a 8 bit, oppre a 16 bit, oppure a 32 bit in un singolo accesso in memoria; l'ipotesi di scrivere in memoria a byte comporta che: 1) ogni 8 periodi di clock occorre accedere in memoria, 2) ogni 8 byte di immagine bisogna scrivere anche un byte di testo, con un impegno di 6 cicli di clock netti per la sola scrittura, a cui va aggiunto il tempo di cessione e ripresa dei bus; il margine esiguo di 8-6=2 soli cicli di clock comporta un rischio inaccettabile per tale approccio, che quindi viene scartato. L'ipotesi di scrivere in memoria a word comporta che: 1) ogni 16 periodi di clock occorre accedere in memoria, 2) ogni 4 word di immagine bisogna scrivere anche un byte di testo, con un impegno di 6 cicli di clock netti per la sola scrittura, a cui va aggiunto il tempo di cessione e ripresa dei bus; quest'ultima operazione dovrebbe concludersi entro il margine di 16-6=10 cicli di clock per non causare errori di acquisizione. Lavorando a 32 bit tale margine viene elevato a 32-6=26 cicli di clock, al costo dell'incremento della lunghezza dei registri; in questo caso: 1) ogni 32 periodi di clock occorre accedere in memoria, 2) ogni 2 longword di immagine bisogna scrivere anche un byte di testo. Il tempo di guardia di cui si è discusso viene speso negli stati S9, S11 e S1 dello SCO.

- Si può emettere la richiesta di interruzione quando è ancora BUSGRANT=1 perché INT viene memorizzata in un flip-flop e quindi sarà rilevata dal processore dopo avere ripreso l'uso dei bus (cfr. diagramma gestione DMA / INT).

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 14-6-2000

Studente:_____ Docente:_____

D1 Una linea di comunicazione binaria rumorosa produce l'inversione di singoli bit separati da un intervallo minimo di 10^2 T, essendo T il periodo di cifra. Determinare un codice (k bit di controllo ogni n bit del codice irridondante) a singola correzione da utilizzare sulla linea per non avere errori in ricezione, mantenendo il rendimento (n/(n+k)) del codice non inferiore al 90%.

D2 Disegnare la struttura e la cella di uno shifter di tipo n-logn stadi a 4 bit e valutarne la complessità (numero di transistori) in una realizzazione CMOS e le prestazioni (tempo di calcolo).

D3 Sintetizzare la macchina minima nella tabella come rete autosincronizzante.

S	00	01	11	10	Z
A	**A**	**A**	D	C	0
B	A	A	**B**	**B**	0
C	**C**	A	B	**C**	1
D	C	**D**	**D**	C	1

D4 Trasformare una microistruzione che effettua il test di due variabili in un frammento di microprogramma nel microlinguaggio di tipo 2.

D5 Data l'istruzione assembler PD32: MOVW (R2)+,ALFA descriverne:
- l'allocazione in memoria;
- il numero dei cicli-macchina compresi nel ciclo istruzione.

Esercizio (2S20000614-D1)

Una linea di comunicazione binaria rumorosa produce l'inversione di singoli bit separati da un intervallo minimo di 10^2 T, essendo T il periodo di cifra. Determinare un codice (k bit di controllo ogni n bit del codice irridondante) a singola correzione da utilizzare sulla linea per non avere errori in ricezione, mantenendo il rendimento (n/(n+k)) del codice non inferiore al 90%.

Occorre un codice con C=1 con n e k da determinare in base alla relazione costitutiva del codice di Hamming con h=3:

$n \leq 2^k - k - 1$ (1)

e ai due vincoli:

- lunghezza: $n+k \leq 100$ (2)
- rendimento: $n/(n+k) \geq 0.9$ (3)

Secondo la (1) k bit di controllo possono essere aggiunti a 2^k - k - 1 bit al più del codice irridondante; per selezionare una coppia (n,k) che soddisfi le (2) e (3) conviene tabellare l'espressione a secondo membro della (1) in funzione di k:

k	2^k - 1- k	Rend. Max.
2	1	2/3 = 0.66
3	4	4/7 = 0.57
4	11	11/15 = 0.73
5	26	26/31 = 0.84
6	57	57/63 = 0.90
7	120 $\rightarrow$ 93	93/100 = 0.93
8	247 $\rightarrow$ 92	92/100 = 0.92
9	502 $\rightarrow$ 91	91/100 = 0.91
10	1013$\rightarrow$90	90/100 = 0.90
11	2036$\rightarrow$89	89/100 = 0.89

In pratica conviene arrestare la serie al primo valore di 2^k - 1- k che eccede 100, cui corrisponde la coppia (93,7). Si noti che a questa coppia è associato anche il massimo rendimento; infatti, per i valori di k minori di 7 il valore massimo di n decresce secondo la (1); invece per i valori di k maggiori di 7 esiste il vincolo n+k=100 (= e non < per massimizzare il rendimento), che decrementa n della stessa quantità di cui viene incrementato k. Quest'ultimo è un risultato generale, legato al fatto che per codificare n bit secondo Hamming sono sufficienti non più di k bit come espresso dalla (1); nel caso specifico 90 bit di codice irridondante possono essere protetti con (almeno) 7 bit di controllo e ogni ulteriore bit di controllo (nella tabella, k=10) è ridondante per la specifica richiesta, e riduce il rendimento del codice.

La (2) è soddisfatta dalle coppie (n,k) con n limitato superiormente dai rispettivi valori nella seconda colonna; ad esempio il valore k = 7 può essere accoppiato con i valori di n tali che: $58 \leq n \leq 93$, dovendo essere per la (2): $n+k \leq 100$.

La (3) può essere verificata con riferimento ai valori tabellati nella colonna 3, relativi al massimo rendimento del codice, dato da:
$(2^k - k - 1) / (2^k - 1)$
I soli valori di k che rispettano la (3) sono 6..10, e tenendo conto delle considerazioni precedenti i valori di interesse si restringono a 6 e 7;

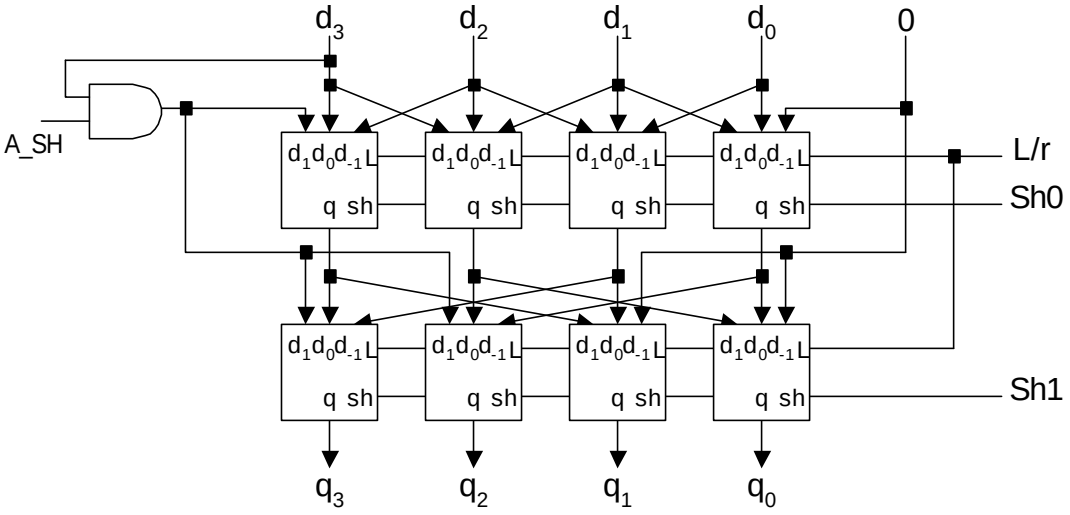
per k=6 il minimo valore di n che può essere usato è dato dalla soluzione di:
$n/(n+6) \geq 0.9 \Rightarrow 0.1\ n \geq 5.4 \Rightarrow n \geq 54$; quindi si potrà scegliere n tale che:
$54 \leq n \leq 57$ (k=6)

per k=7 il minimo valore di n che può essere usato è dato dalla soluzione di:
$n/(n+7) \geq 0.9 \Rightarrow 0.1\ n \geq 6.3 \Rightarrow n \geq 63$; quindi si potrà scegliere n tale che:
$63 \leq n \leq 93$ (k=7)

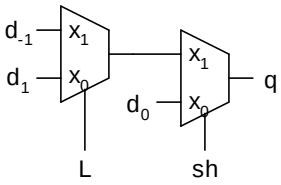
In principio i codici che soddisfano le specifiche sono 4+31=35. La scelta effettiva potrà essere condizionata al soddisfacimento di altri criteri: ad esempio il rendimento, nel qual caso verrà preferito il codice (93,7), oppure le dimensioni, e allora sarà preferito il codice (54,6).

Esercizio (2S20000614-D2)

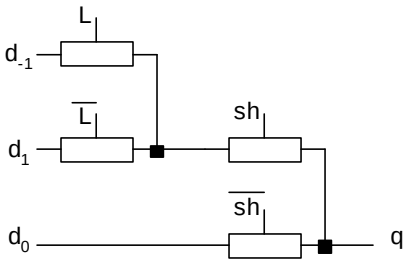
Disegnare la struttura e la cella di uno shifter di tipo n-logn stadi a 4 bit e valutarne la complessità (numero di transistori) in una realizzazione CMOS e le prestazioni (tempo di calcolo).



Barrel-shifter a 4 bit. Shift aritmetico/logico controllato da A_SH



Realizzazione della cella con multiplexer



Realizzazione della cella con 4 pass-transistor (8 transistor)

Costo della rete: 8 celle x 8 transistori/cella = 64 transistori.

Livello della rete: 2 celle x 2 pass-transistor/cella = 4 pass-transistor ⇒

Tempo di calcolo della rete: 4 T essendo T il tempo di commutazione di un pass-transistor.

Esercizio (2S20000614-D3)

Sintetizzare la macchina minima nella tabella come rete autosincronizzante.

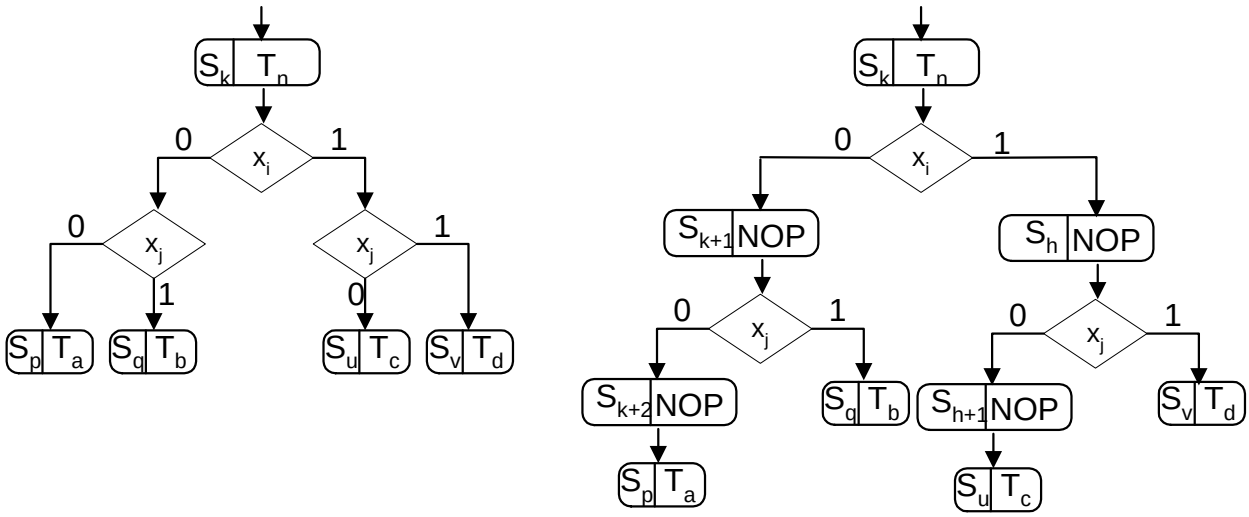
S	00	01	11	10	Z
A	**A**	**A**	D	C	0
B	A	A	**B**	**B**	0
C	**C**	A	B	**C**	1
D	C	**D**	**D**	C	1

Si tratta del flip-flop con due ingressi S(et) e R(eset) entrambi attivi sul fronte, la cui sintesi completa è riportata negli appunti integrativi del corso con riferimento a diverse tecniche, tra cui quella richiesta.

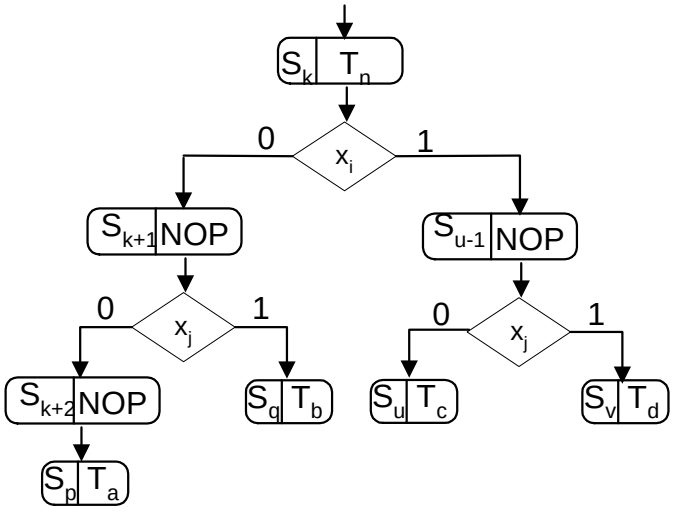
Esercizio (2S20000614-D4)

Trasformare una microistruzione che effettua il test di due variabili in un frammento di microprogramma nel microlinguaggio di tipo 2.

Nella figura a sinistra è indicata la microistruzione con il doppio salto. Nella figura a destra è indicata l'implementazione con il microlinguaggio di tipo 2, che, come è noto, utilizza microistruzioni: 1) con un test singolo; 2) con uno solo dei due possibili stati successivi diverso da quello ottenuto mediante incremento unitario.



La soluzione mostrata, valida in generale, implica la disponibilità delle locazioni consecutive di indirizzi k, k+1, k+2 e h, h+1.
Va notato che dipendentemente dalla (particolare) disponibilità anche delle locazioni di indirizzi u e u-1, si può semplificare ulteriormente la ramificazione, riducendo il numero delle microistruzioni di appoggio (NOP) come mostrato dal seguente frammento di microprogramma:



Esercizio (2S20000614-D5)

Data l'istruzione assembler PD32: MOVW (R2)+,ALFA descriverne:
- l'allocazione in memoria;
- il numero dei cicli-macchina compresi nel ciclo istruzione.

---

1)
L'allocazione dell'istruzione in memoria è schematizzata nella figura seguente:

OC-7..0	Addr
OC-15..8	
OC-23..16	
OC-31..24	Addr+3
ALFA-7..0	Addr+4
ALFA-15..8	
ALFA-23..16	
ALFA-31..24	Addr+7

Per un impegno globale di 8 byte (opcode: 4 byte, ALFA: 4 byte)

2)
Cicli-macchina del ciclo-istruzione:
- fetch;
- prelevamento di ALFA;
- prelevamento della word puntata da R2;
- scrittura della word appena prelevata nella cella di indirizzo ALFA.

In totale vengono eseguiti 4 cicli-macchina; le attività interne (post-incremento, etc.) sono attribuite allo stesso ciclo macchina che procura i dati su cui operano.

RETI LOGICHE

PRIMA PROVA SCRITTA DEL 28-06-2000

STUDENTE:_____ DOCENTE:_____

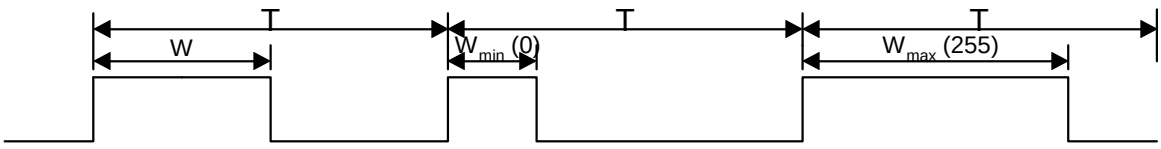
Una periferica remota monitorizza un segnale analogico mediante un convertitore analogico-digitale (ADC) a 8 bit, dotato di un ingresso di inizio conversione (SOC) e con un tempo di conversione massimo pari a 10 microsecondi. I campioni x_n vengono filtrati secondo la relazione:

$$y_n = 0.25\ x_{n-2} + 0.5\ x_{n-1} + 0.25\ x_n$$

I valori y_n vengono trasmessi su una linea di comunicazione seriale senza soluzione di continuità mediante modulazione PWM (modulazione della larghezza dell'impulso): ciascun elemento y_n viene trasformato in un impulso di durata W_n secondo la relazione:

$$W_n = T/4 + y_n\ T/512$$

essendo $0 \leq y_n \leq 255$ e T l'intervallo costante tra i fronti di salita di due impulsi consecutivi, come mostrato nella specifica temporale grafica.

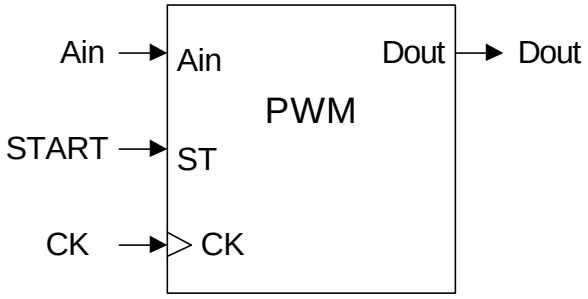


L'attività dell'interfaccia viene attivata da un segnale di START; la periferica deve convertire e trasmettere 64K campioni. Il clock della periferica è di 32 MHz.

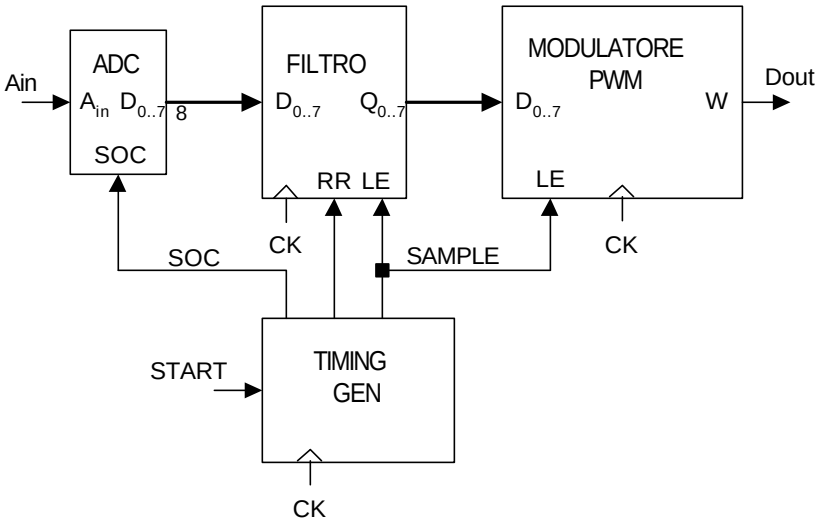
Si richiede:

1. La temporizzazione delle operazioni;
2. Lo schema logico della periferica.

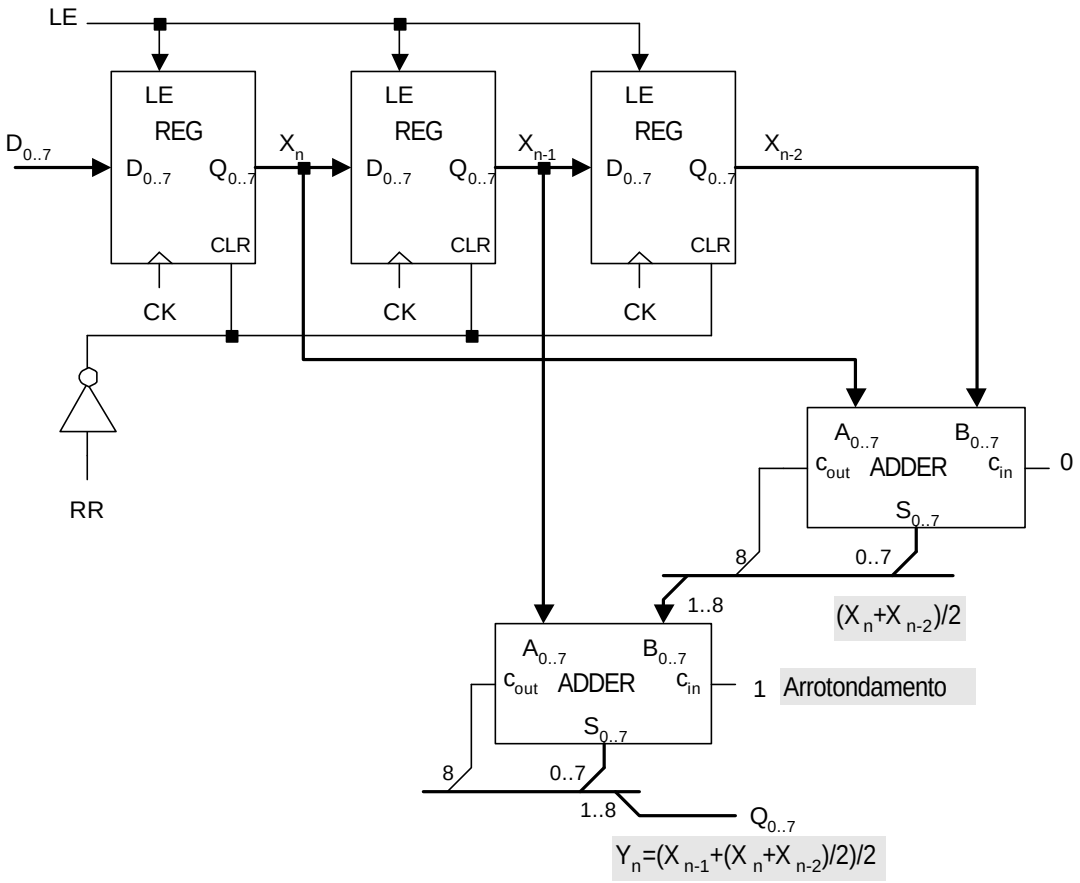
PWM: connessione esterna



PWM: schema a blocchi



PWM: filtro

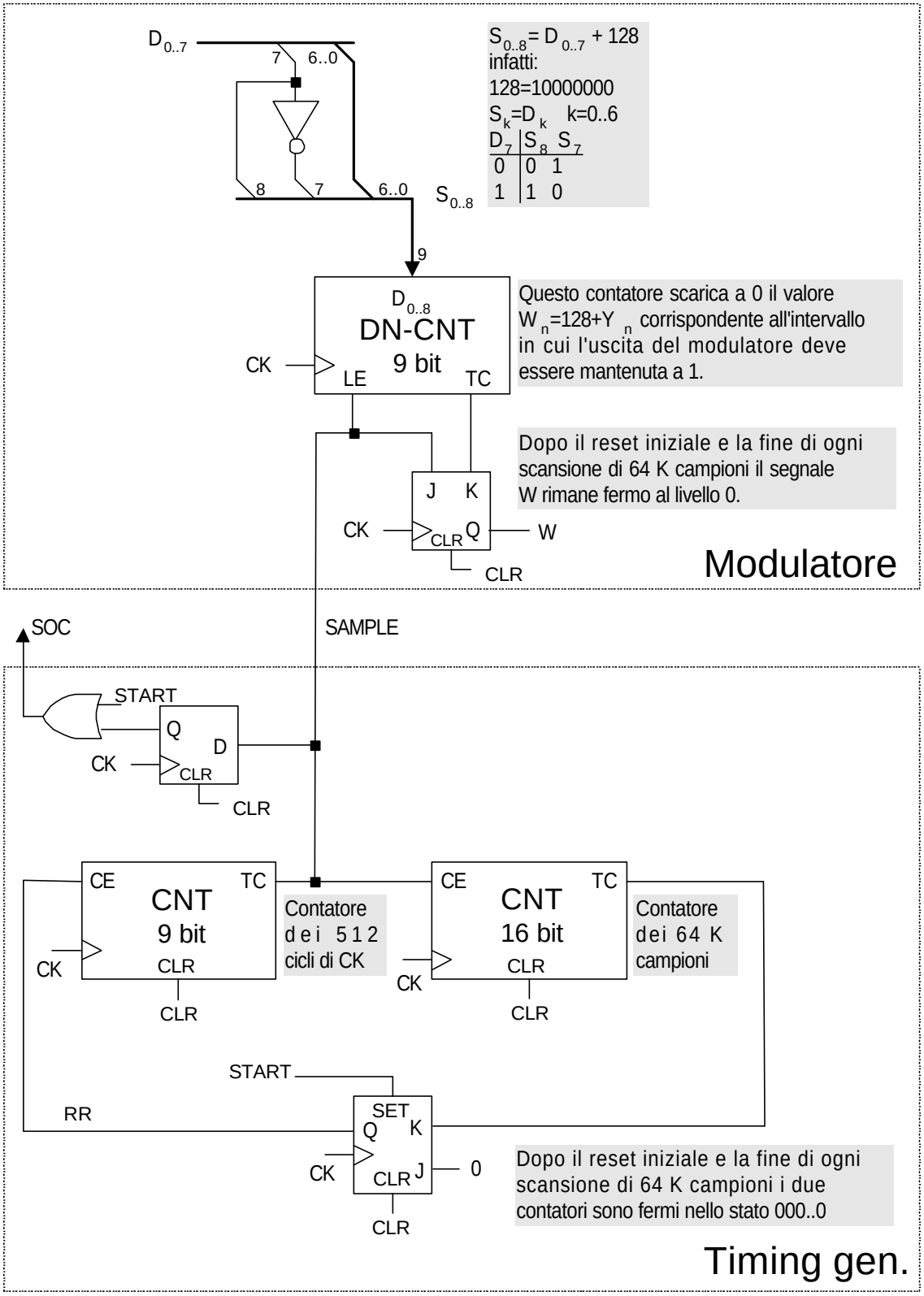


Esempio di addizione pesata:

X_n=	10001001
X_{n-2}=	10101010
X_n+X_{n-2}=	100110011
$(X_n+X_{n-2})/2$=	10011001.1
X_{n-1}=	11000110
$X_{n-1}+(X_n+X_{n-2})/2$=	101011111.1
$(X_{n-1}+(X_n+X_{n-2})/2)/2$=	10101111.11
arrotondamento a 8 bit: +	0.1
$0.5X_{n-1}+0.25(X_n+X_{n-2})$=	10110000

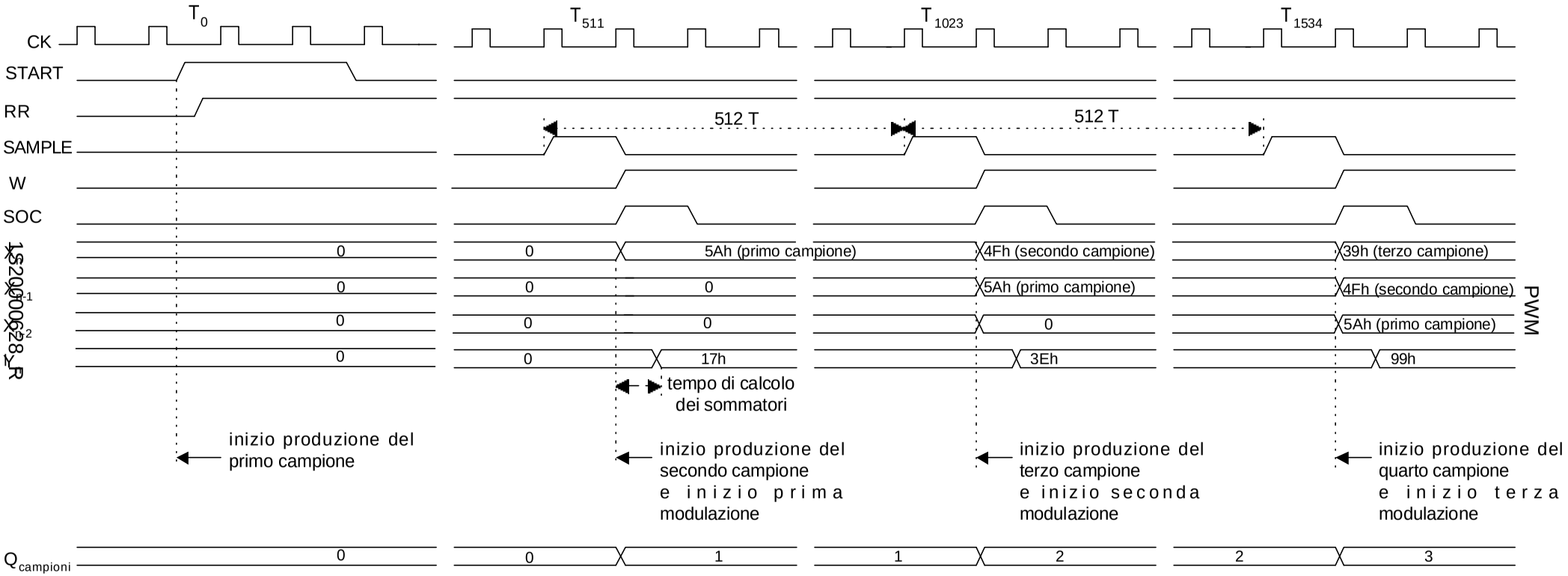
Si nota che il bit a destra della virgola di $(X_n+X_{n-2})/2$ non contribuisce al risultato finale, e quindi può essere tralasciato nei calcoli successivi, cioè il risultato di $(X_n+X_{n-2})/2$ può essere troncato a 8 bit. Secondo lo schema di calcolo illustrato il risultato parziale così ottenuto va sommato agli 8 bit di X_{n-1} con il carry-in del sommatore forzato a 1 per effettuare l'arrotondamento; il risultato definitivo - a 8 bit - è prelevato sui 7 bit più pesanti della somma affiancati dal bit di carry-out.

PWM: modulatore + Timing gen.



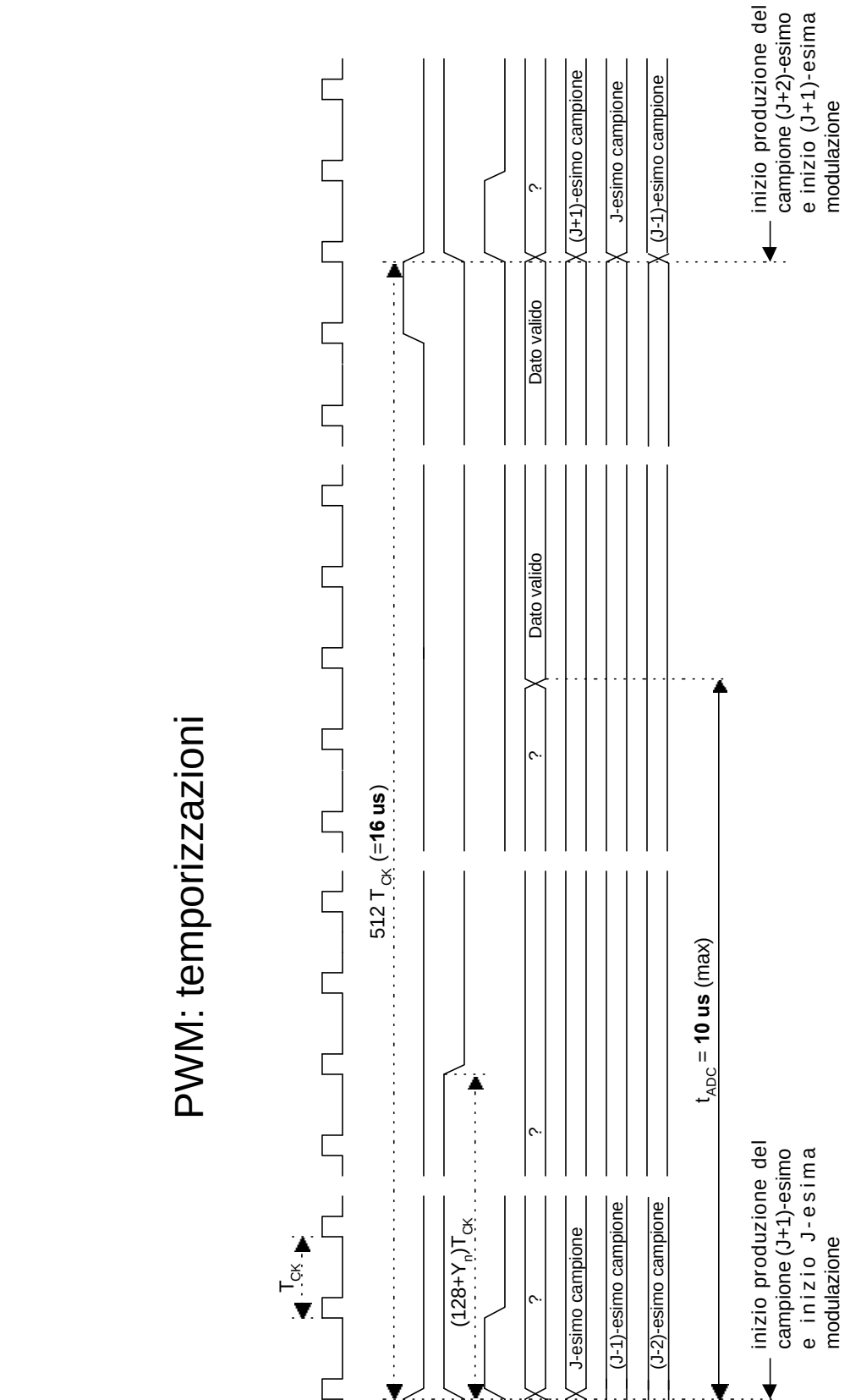
PWM: temporizzazioni

Transitorio iniziale



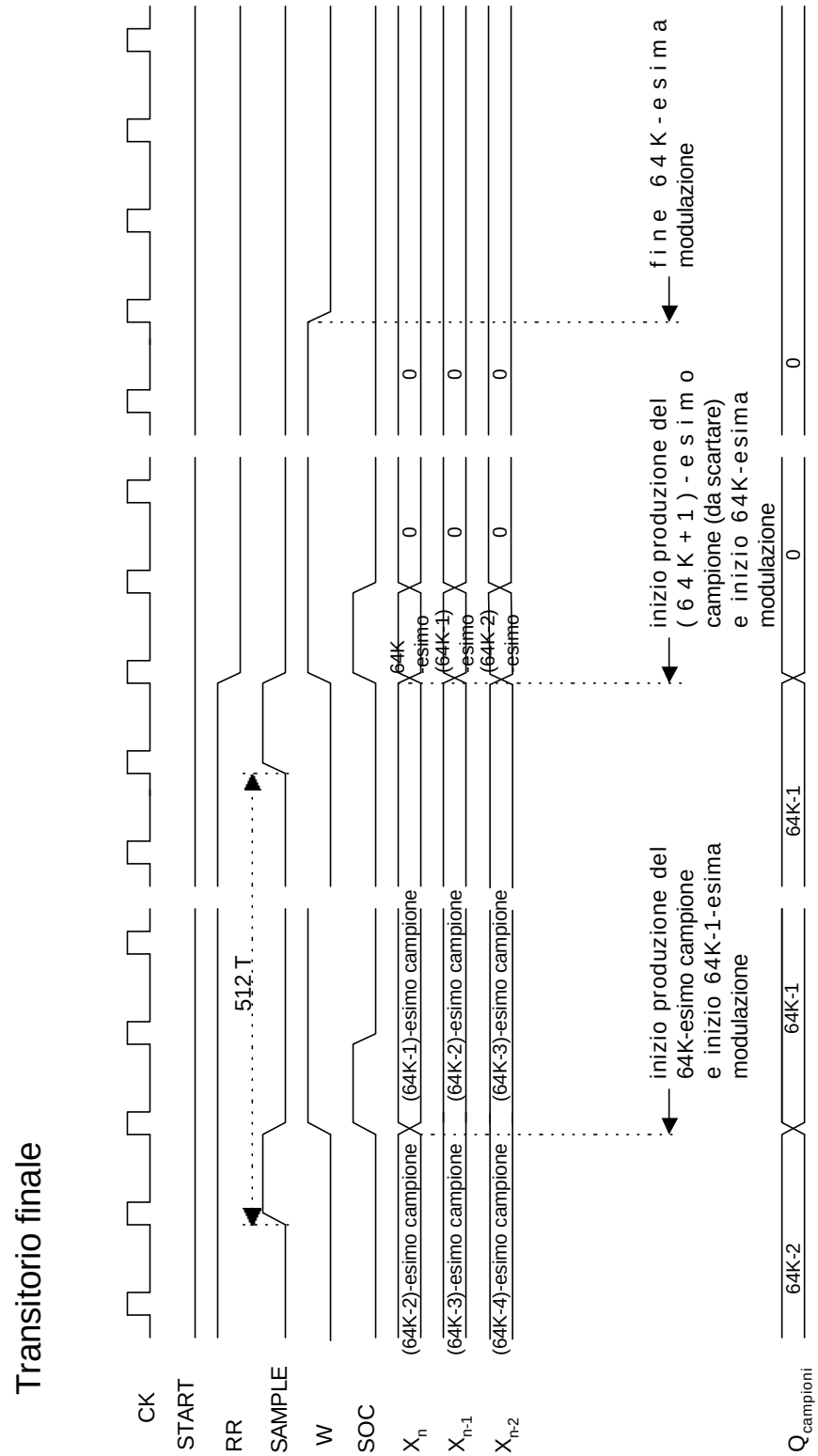
PWM: temporizzazioni

Modulazione



PWM: temporizzazioni

Transitorio finale



Commenti al progetto

- Si suppone che il segnale esterno di avvio START sia privo di transizioni spurie.

- Nella fase transitoria di avvio vengono trasmessi i tre valori che provengono dalla pipeline dei tre registri, inizializzati a 0 e non ancora completamente caricati con i primi tre campioni; tali valori corrispondono al tempo di salita del segnale all'uscita del filtro.

- La risoluzione temporale prescritta vale T/512; pertanto il periodo di CK verrà scelto in modo tale che: $T=512\ T_{CK}$. Essendo F_{ck} = 32 MHz, allora T=512/32 us = 16 us. Cioè verrà acquisito un nuovo campione ogni 16 us. E' specificato un tempo di conversione t_{ADC} dell'ADC di 10 us; essendo $t_{ADC} < T_{CK}$ si può comandare l'acquisizione di un nuovo campione (SOC) e simultaneamente prelevare il campione della conversione precedente (SAMPLE), senza aspettare il consenso del segnale End-Of-Conversion; in realtà il comando SOC viene emesso un ciclo di CK in ritardo rispetto al caricamento del campione disponibile all'uscita dell'ADC, a causa del flip-flop D che risincronizza SAMPLE per eliminarne le alee, ma che inevitabilmente lo ritarda, peraltro senza alcuna penalizzazione per le temporizzazioni.

- Essendo $T=512\ T_{CK}$, allora $T/4=128\ T_{CK}$; il valore 128 è addizionato a quello di Y_n e la somma caricata nel contatore a decremento che pilota il flip-flop di uscita (cfr. modulo filitro).

RETI LOGICHE

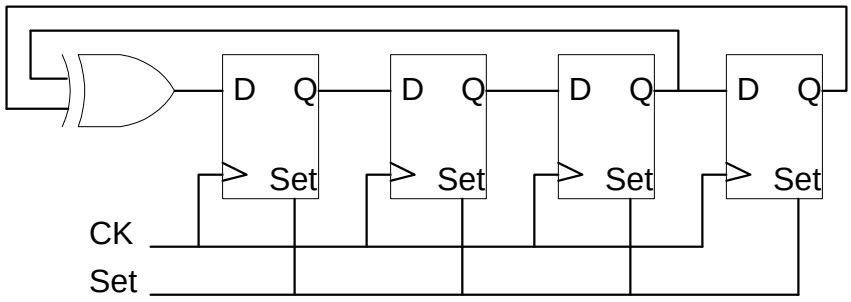
SECONDA PROVA SCRITTA DELL'APPELLO DEL 28-6-2000

Studente:_____ **Docente:**_____

D1 Descrivere le capacità di correzione e/o rivelazione di un codice a distanza di Hamming pari a 4.

D2 Si vuole utilizzare una ROM di capacità 1 Mbyte per segnalare quali tra i primi 8×2^{20} numeri interi sono primi. Descrivere l'organizzazione dei dati nella ROM ed il circuito esterno di pilotaggio.

D3 Analizzare il comportamento del registro a scalamento (di Fibonacci) in figura, a partire da uno stato iniziale di tutti 1.



D4 Descrivere la struttura di uno SCO multimicroprogrammato che supporti un microlinguaggio di tipo 2.

D5 Scrivere una routine assembler PD32 per effettuare la somma tra due numeri a 4 cifre BCD memorizzati in RAM agli indirizzi DEC0 e DEC1; il risultato, anch'esso in BCD, va memorizzato all'indirizzo DEC2. Una variabile binaria VF è predisposta in memoria per segnalare un eventuale overflow.

Esercizio (2S20000628-D1)

Descrivere le capacità di correzione e/o rivelazione di un codice a distanza di Hamming pari a 4.

Per ipotesi due generiche parole del codice O1 e O2 sono separate da tre parole illegali secondo la metrica di Hamming (X1, X2, X3 nella rappresentazione grafica)

O1 --------- X1 --------- X2 --------- X3 --------- O2

Poiché h>3 il codice può essere utilizzato come rivelatore puro oppure come correttore e rivelatore:

Rivelazione di errore

Il numero R di bit errati che possono essere rilevati è dato dalla relazione:
$R = h-1$
In questo caso $R = 4-1 = 3$
Nella rappresentazione grafica ciò corrisponde a ricevere X1 o X2 o X3, avendo trasmesso O1 oppure O2.

Correzione e rivelazione di errore

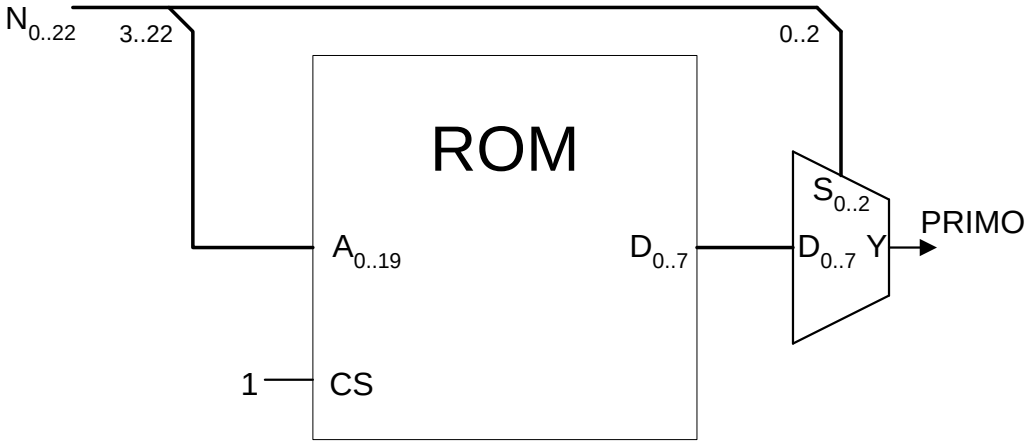
Il numero C di bit errati che possono essere corretti è dato dalla relazione:
$2C \leq h-1$
In questo caso $C = 1$
Nella rappresentazione grafica ciò corrisponde a ricevere X1 avendo trasmesso O1, oppure a ricevere X3 avendo trasmesso O2. In questa modalità di impiego del codice si può sfruttare X2 per rilevare un ulteriore errore (non correggibile, in quanto si avrebbe l'indeterminazione se assegnarlo a O1 o a O2)
In generale, una volta determinato il numero C di bit errati che possono essere corretti, il numero R dei bit in eccesso rilevabili può essere ricavato dalla relazione:
$2C+R = h-1$
In questo caso: $2+R=3 \Rightarrow R=1$
In definitiva è: C=1; R=1.

Esercizio (2S20000628-D2)

Si vuole utilizzare una ROM di capacità 1 Mbyte per segnalare quali tra i primi 8×2^{20} numeri interi sono primi. Descrivere l'organizzazione dei dati nella ROM ed il circuito esterno di pilotaggio.

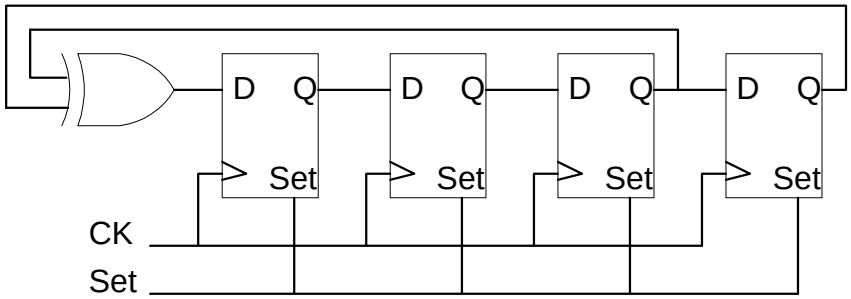
L'idea sull'utilizzo della ROM è quella di memorizzare in ogni singolo bit indirizzabile della ROM l'informazione binaria primo (1) – non primo (0) relativa al numero presentato sulle linee di indirizzo della ROM.

La capacità della ROM è pari a 8×2^{20} bit, però i bit non sono accessibili singolarmente, ma a righe di 8 bit, in quanto l'organizzazione interna della ROM è di 2^{20} x 8 bit (1 Megabyte); questo significa che i numeri, espressi a 23 bit, dovranno essere scomposti in due stringhe: i venti bit più significativi verranno presentati sulle linee di indirizzo della ROM, che risponderà con 8 bit, i quali saranno selezionati con un multiplexer controllato dai tre bit meno significativi del numero di cui si vuol sapere se è primo.
La struttura hardware è graficata nella figura seguente.



Esercizio (2S20000628-D3)

Analizzare il comportamento del registro a scalamento (di Fibonacci) in figura, a partire da uno stato iniziale di tutti 1.



Lo stato iniziale di tutti 1 può essere ottenuto con un impulso sulla linea Set. La sequenza degli stati viene scandita dal circuito con la cadenza del clock CK; il sistema è autonomo (è privo di ingressi diversi da CK), pertanto è forzato a ripetere ciclicamente una stessa sequenza, di lunghezza massima non maggiore di $2^4 = 16$ (registro con 4 flip-flop). La sequenza può essere determinata seguendo l'evoluzione dello stato dei quattro flip-flop:

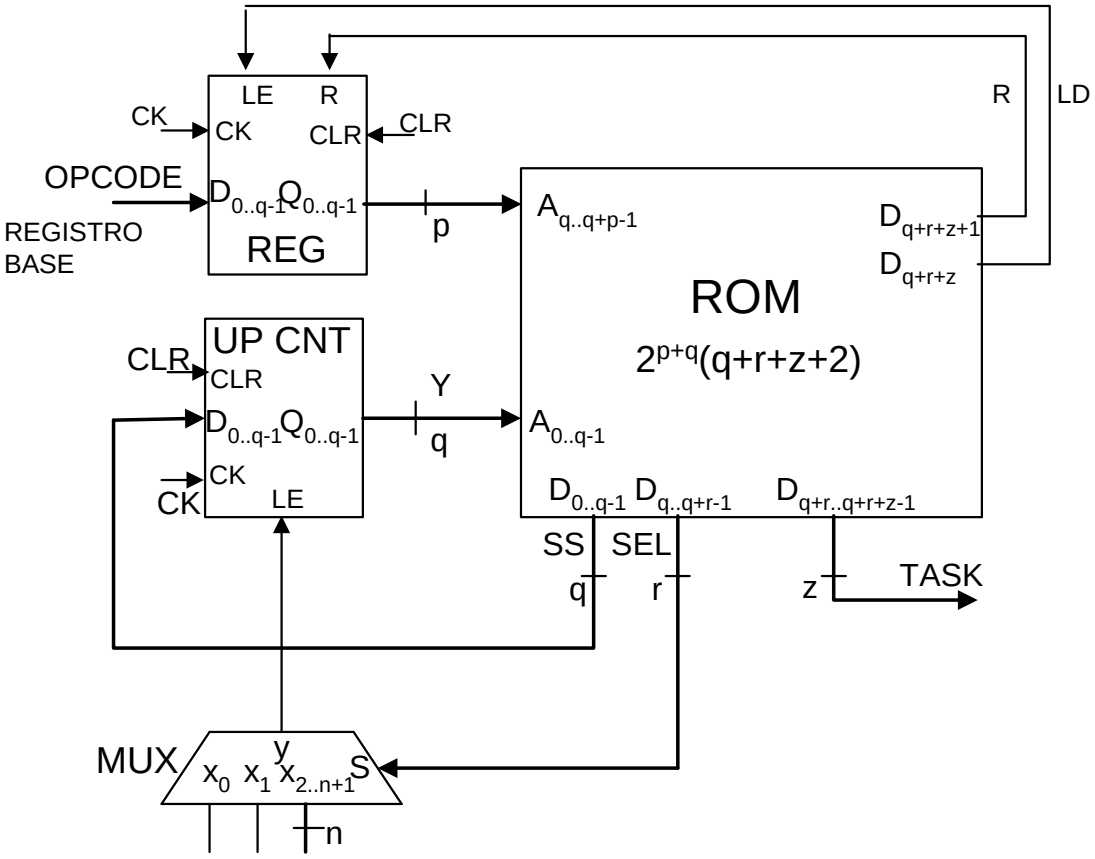
1111
0111
0011
0001
1000
0100
0010
1001
1100
0110
1011
0101
1010
1101
1110

Le proprietà che si possono notare nella sequenza trovata sono:
- ha lunghezza pari a 15;
- la sequenza non include il codice 0000; a questo proposito va notato che se il registro fosse inizializzato a 0000, lo stato successivo sarebbe ancora 0000, e quindi il registro resterebbe definitivamente ancorato allo stato 0000.

Esercizio (2S20000628-D4)

Descrivere la struttura di uno SCO multimicroprogrammato che supporti un microlinguaggio di tipo 2.

Occorre aggiungere il registro base e la relativa logica di caricamento alla struttura HW microprogrammata di tipo 2; il registro spostamento è in questo caso il registro contatore. La struttura complessiva è riportata nella figura seguente.



Esercizio (2S20000628-D5)

Scrivere una routine assembler PD32 per effettuare la somma tra due numeri a 4 cifre BCD memorizzati in RAM agli indirizzi DEC0 e DEC1; il risultato, anch'esso in BCD, va memorizzato all'indirizzo DEC2. Una variabile binaria VF è predisposta in memoria per segnalare un eventuale overflow.

```
;Effettua la somma tra due numeri a 4 cifre BCD memorizzati in RAM
;agli indirizzi DEC0 e DEC1;
;il risultato, anch'esso in BCD, va memorizzato all'indirizzo DEC2.
;Segnala un eventuale overflow nella variabile binaria VF.

        org 400h                ;inizio programma

; ************************
; COSTANTI
; ************************

        STACK EQU 2800h         ;inizio area di stack
                                ;limitato a 2800h per consentire la simulazione

; ************************
; VARIABILI IN MEMORIA
; ************************

        DEC0 DW 02804h  ;addendi BCD: possono essere ridefiniti
        DEC1 DW 01957h  ;a scopo di test della routine
        DEC2 DW 00000h
        VF      DB 000h

; ************************
; CODICE
; ************************

        code                ;inizio istruzioni

main:
        movl #STACK,r7          ;inizializza R7 quale SP
        seti                ;abilita interruzioni (SP è stato inizializzato)

        jsr add4bcd

        halt                ;serve solo per bloccare il simulatore

; ************************
; ROUTINE
; ************************
add4bcd:
        push r0             ;salva i reg. nello stack
```

```
        push r1
        push r2
        push r3
        push r4
        push r5

        movb #0h,VF     ;inizial. variabile di trabocco a 0
        movw dec0,r0    ;r0 contiene il primo addendo
        movw dec1,r1    ;r1 contiene il secondo addendo
        xorl r2,r2      ;r2 conta i 4 nibble
        xorl r3,r3      ;r3 contiene la somma

loop:
        movb r0,r4      ;metti in r4 il byte meno significativo di r0
        movb r1,r5      ;metti in r5 il byte meno significativo di r1
        andb #00Fh,r4   ;e isola il nibble meno significativo di r4
        andb #00Fh,r5   ;e isola il nibble meno significativo di r4
        addb r4,r5      ;somma i due nibble -> r5
        addb VF,r5      ;incluso il bit di overflow della somma precedente
        movb #0,VF
        cmpb #10,r5     ;test se overflow
        jn update
        movb #1,VF      ;registra il bit di overflow
        addb #6,r5      ;correggi il nibble
        andb #0Fh,r5
update:
        orw r5,r3       ;metti il nibble-somma in r3
        rorw #4,r3      ;e ruota r3 a dx
        asrw #4,r0      ;scala gli operandi a dx
        asrw #4,r1

        addb #1,r2      ;incrementa il contatore
        cmpb #4,r2      ;test su fine conteggio
        jnz loop

        movw r3,DEC2    ;scrivi il risultato in memoria

        pop r5          ;ricarica i reg. dallo stack
        pop r4
        pop r3
        pop r2
        pop r1
        pop r0
        ret

        end             ;fine programma
```

```
;Effettua la somma tra due numeri a 4 cifre BCD memorizzati in RAM agli indirizzi DEC0
e DEC1;
;il risultato, anch'esso in BCD, va memorizzato all'indirizzo DEC2.
;Segnala un eventuale overflow nella variabile binaria VF.



  org 400h    ;inizio programma

; *************************
; COSTANTI
; *************************

  STACK   EQU 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione

; *************************
; VARIABILI IN MEMORIA
; *************************

  DEC0  DW 02804h ;addendi BCD: possono essere ridefiniti
  DEC1  DW 01957h   ;a scopo di test della routine
  DEC2  DW 00000h
  VF  DB 000h

; *************************
; CODICE
; *************************

  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)

  jsr add4bcd

  halt      ;serve solo per bloccare il simulatore


; *************************
; ROUTINE
; *************************


add4bcd:
  push r0     ;salva i reg. nello stack
  push r1
  push r2
  push r3
  push r4
  push r5

  movb #0h,VF   ;inizial. variabile di trabocco a 0

  movw dec0,r0    ;r0 contiene il primo addendo
  movw dec1,r1    ;r1 contiene il secondo addendo

  xorl r2,r2    ;r2 conta i 4 nibble
  xorl r3,r3    ;r3 contiene la somma

loop:
```

```
  movb r0,r4    ;metti in r4 il byte meno significativo di r0
  movb r1,r5    ;metti in r5 il byte meno significativo di r1
  andb #00Fh,r4   ;e isola il nibble meno significativo di r4
  andb #00Fh,r5   ;e isola il nibble meno significativo di r4
  addb r4,r5    ;somma i due nibble -> r5
  addb VF,r5    ;incluso il bit di overflow della somma precedente
  movb #0,VF
  cmpb #10,r5   ;test se overflow
  jn update
  movb #1,VF    ;registra il bit di overflow
  addb #6,r5    ;correggi il nibble
  andb #0Fh,r5
update:
  orw r5,r3   ;metti il nibble-somma in r3
  rorw #4,r3    ;e ruota r3 a dx
  asrw #4,r0    ;scala gli operandi a dx
  asrw #4,r1

  addb #1,r2    ;incrementa il contatore
  cmpb #4,r2    ;test su fine conteggio
  jnz loop

  movw r3,DEC2    ;scrivi il risultato in memoria

  pop r5      ;ricarica i reg. dallo stack
  pop r4
  pop r3
  pop r2
  pop r1
  pop r0
  ret

  end     ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 12-7-2000

Un sistema multiprocessore costituito da 8 nodi opera a scambio di messaggi. A tal fine ogni processore si avvale di una propria periferica di comunicazione attestata su un bus comune ad 8 bit secondo lo schema di figura.

La periferica generica è attivata in trasmissione dal suo processore, che le comunica: l'indirizzo della periferica di destinazione (8 bit con codifica lineare), l'indirizzo della memoria del processore da cui prelevare il messaggio e la lunghezza del messaggio stesso. La struttura dei dati da inviare sul bus è:

<indirizzo periferica di destinazione><messaggio>

Quando una periferica vuole entrare in trasmissione esegue le seguenti azioni:

1. effettua un test sulla linea *stato del bus*: se la trova alta (bus libero) la forza a zero e, predisposto l'indirizzo di destinazione come primo byte da spedire, salta al punto 2; altrimenti attende che il bus si liberi (*per semplicità, si assuma che al più una periferica sia in attesa del bus)*;
2. invia il dato sul bus, quindi alza la linea *dato pronto*; attende il segnale *ack* per azzerare la linea *dato pronto*, quindi preleva il prossimo byte da spedire (in modalità DMA) e attende che *ack* torni a zero;
3. se la trasmissione è terminata libera il bus alzando la linea *stato del bus*; altrimenti salta al punto 2.

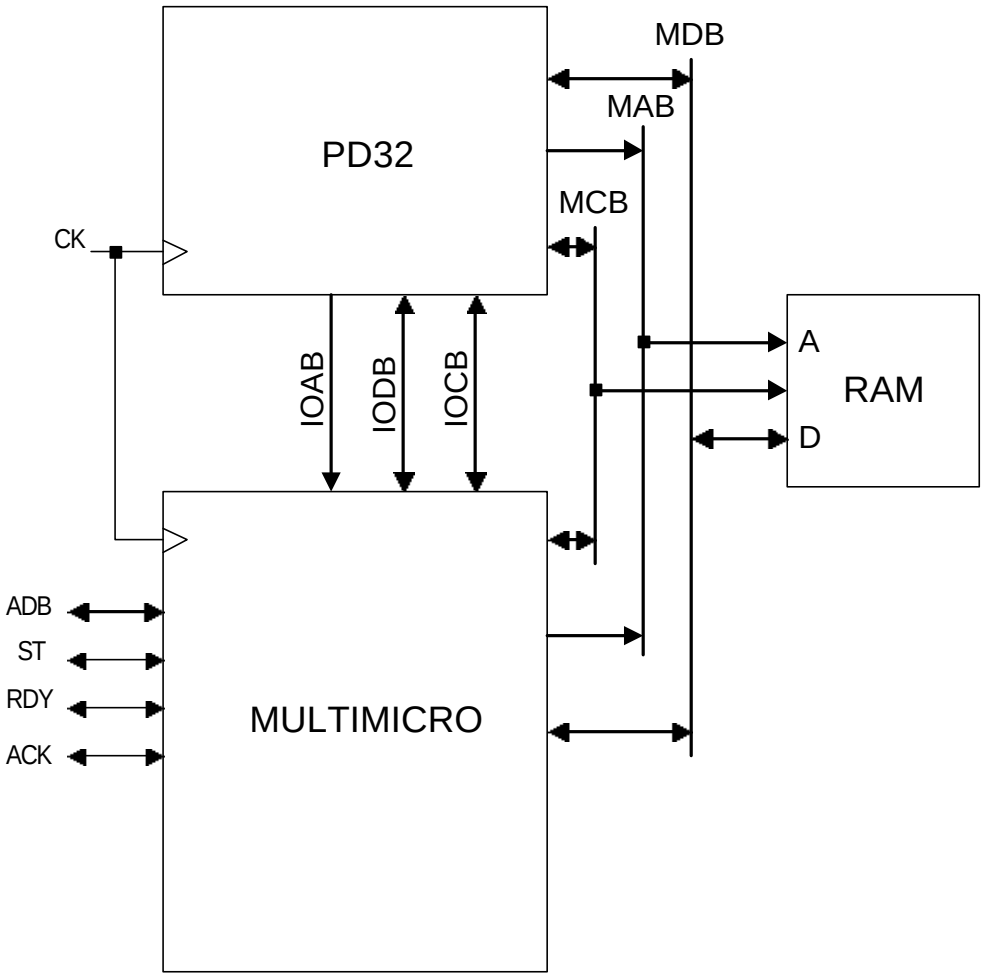
Quando una periferica vede la linea *stato del bus* commutare a zero entra in ricezione ed esegue le seguenti azioni:

1. attende il segnale *dato pronto*, legge l'indirizzo sul bus e verifica se è il proprio: in caso affermativo completa le fasi del protocollo (alza *ack*, aspetta *dato pronto* a zero e pone *ack* a zero) e quindi salta al punto 2 per continuare la ricezione; altrimenti torna nello stato di riposo;
2. attende il segnale di *dato pronto* o la segnalazione di fine trasmissione; se riceve il segnale *dato pronto*, salta al punto 3. Se riceve la segnalazione di fine trasmissione (commutazione della linea *stato del bus* a uno logico) salta al punto 4;
3. legge il dato dal bus, alza il segnale *ack* e trasferisce il dato in memoria a partire da un indirizzo noto alla periferica. Infine, quando *dato pronto* è stato portato a zero dal trasmittente, la periferica ricevente azzera il segnale *ack* e torna al punto 2.
4. invia un interrupt al processore, che quindi accede alla periferica per leggere la lunghezza del messaggio appena trasferito.

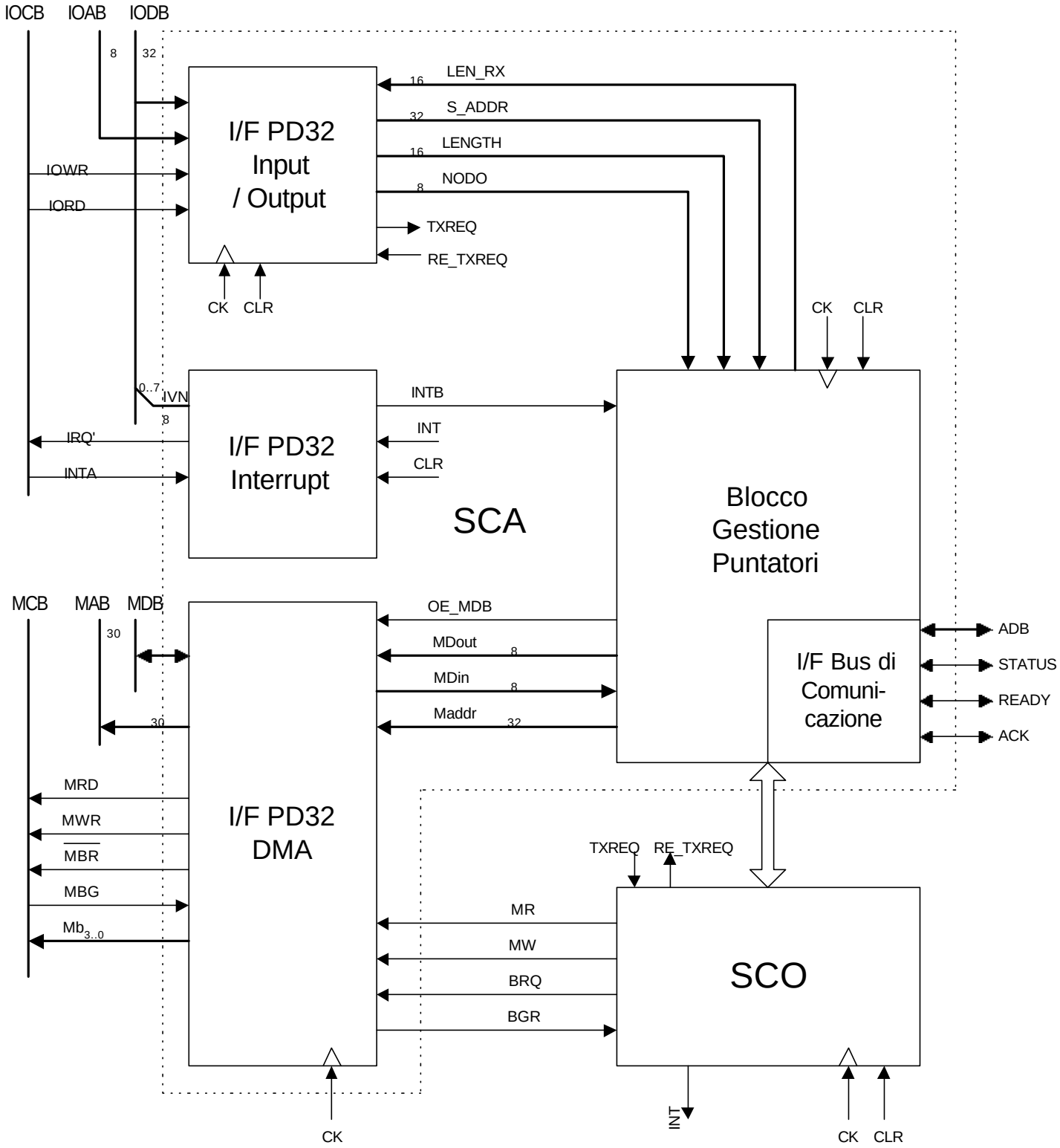
Si richiede:

- Lo schema a blocchi della periferica,
- Il microprogramma e la struttura della SCO della periferica
- L'organizzazione della SCA della periferica.

MULTIMICRO: sistema esterno



MULTIMICRO: Schema a blocchi

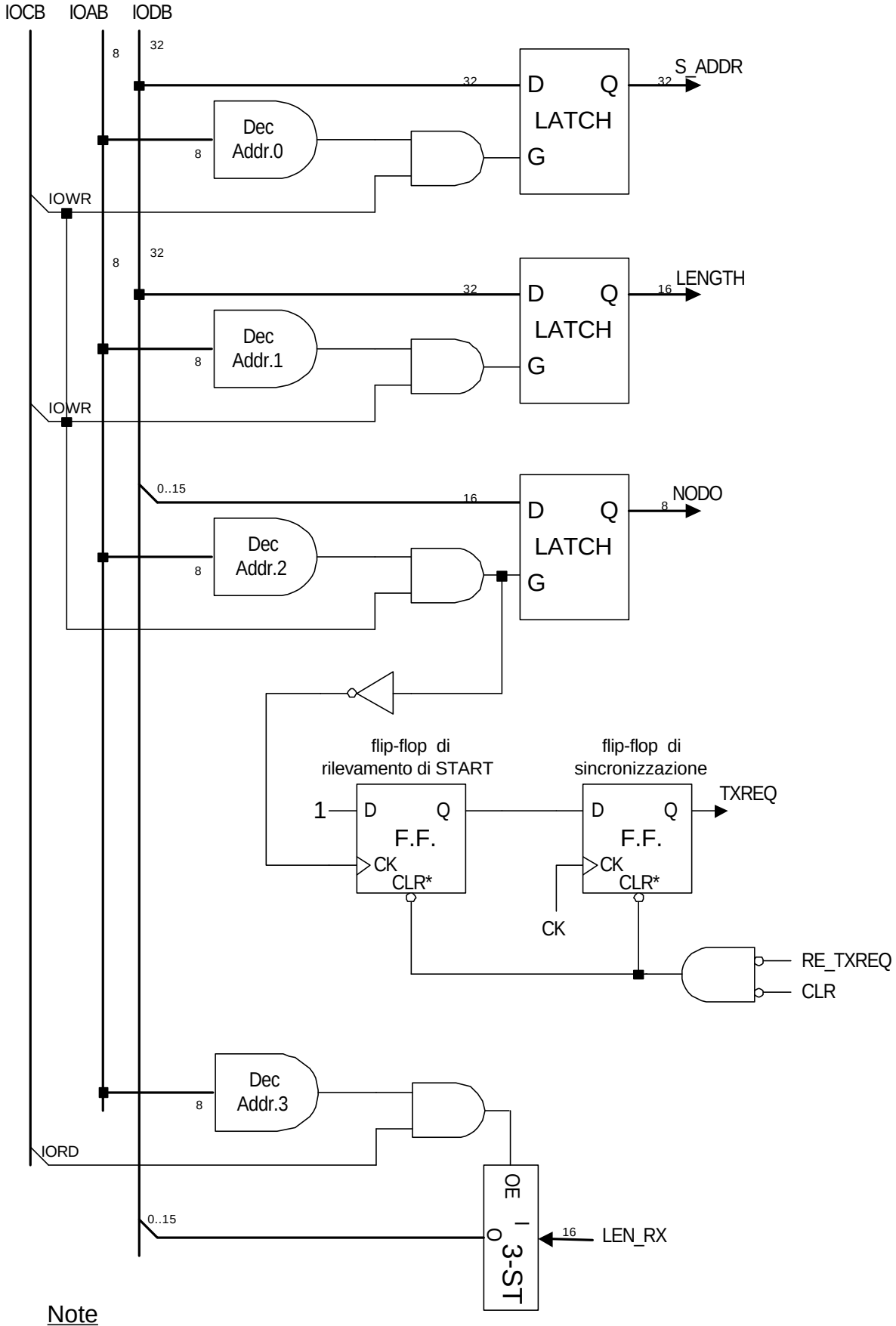


Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.

Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 interrupt usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

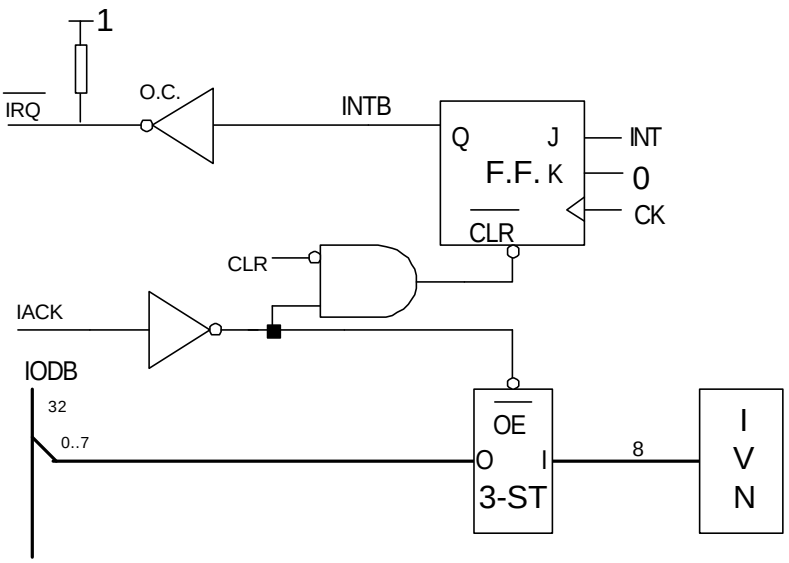
COMM_PRO: IF PD32 - output



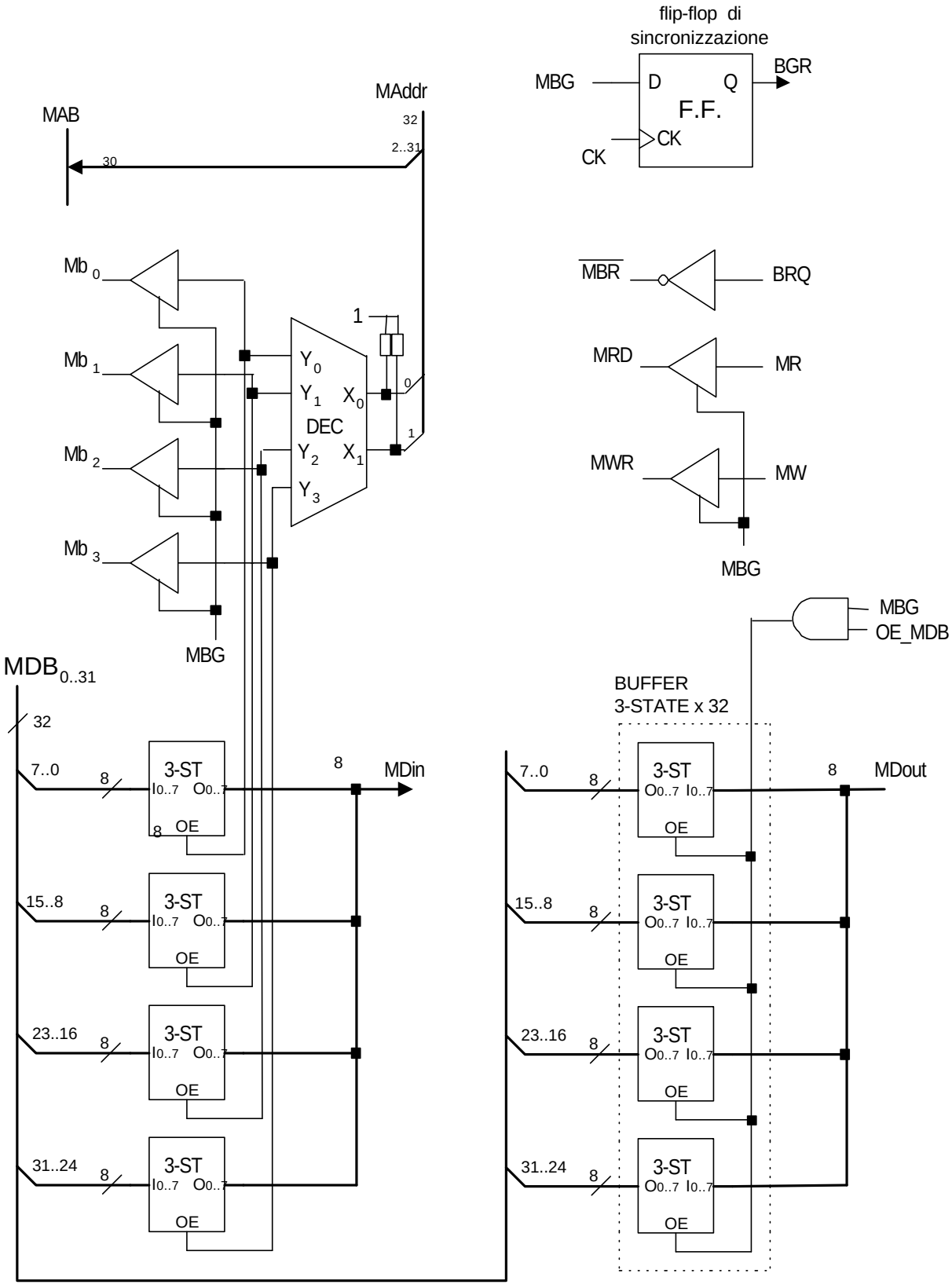
Note

Il SW avvia l'operazione direttamente con la scrittura dell'indirizzo della lunghezza in byte del blocco di dati da trasferire, dopo avere riscritto l'altro registro.

MULTIMICRO: IF PD32 - interrupt



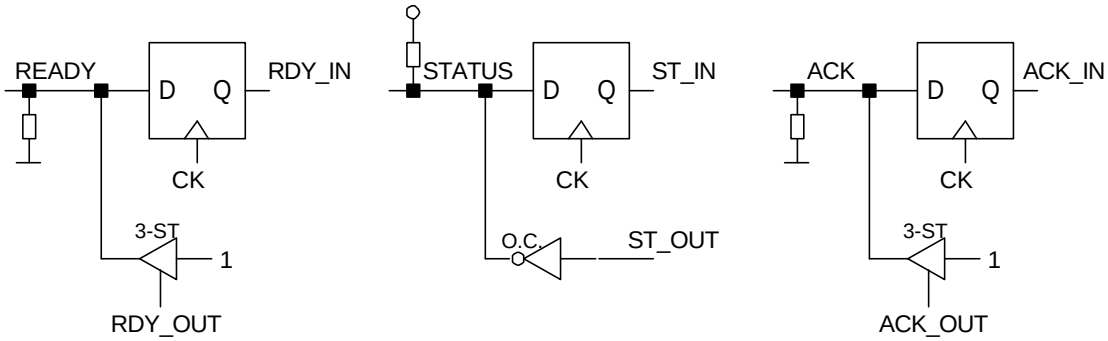
MULTIMICRO: IF PD32 - DMA

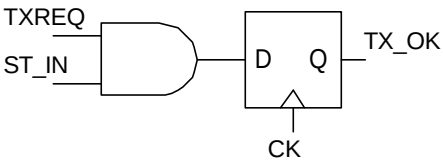


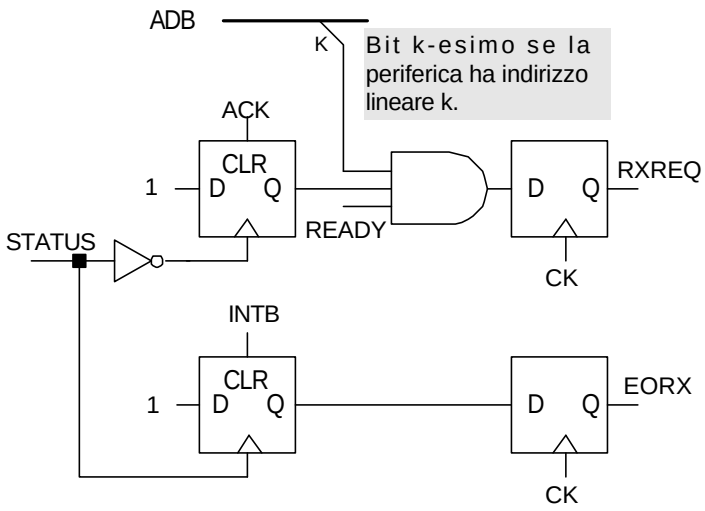
Blocco di riduzione dei 32 bit del MDB agli 8 bit del bus interno MDin (lettura).

Blocco di espansione degli 8 bit del bus interno MDout ai 32 bit del MDB (scrittura). L'abilitazione dei 3-state deve essere condizionata (cfr. porta AND su MBG) da un segnale di accesso in scrittura (OE_MDB), per evitare di interferire sul MDB negli accessi in lettura.

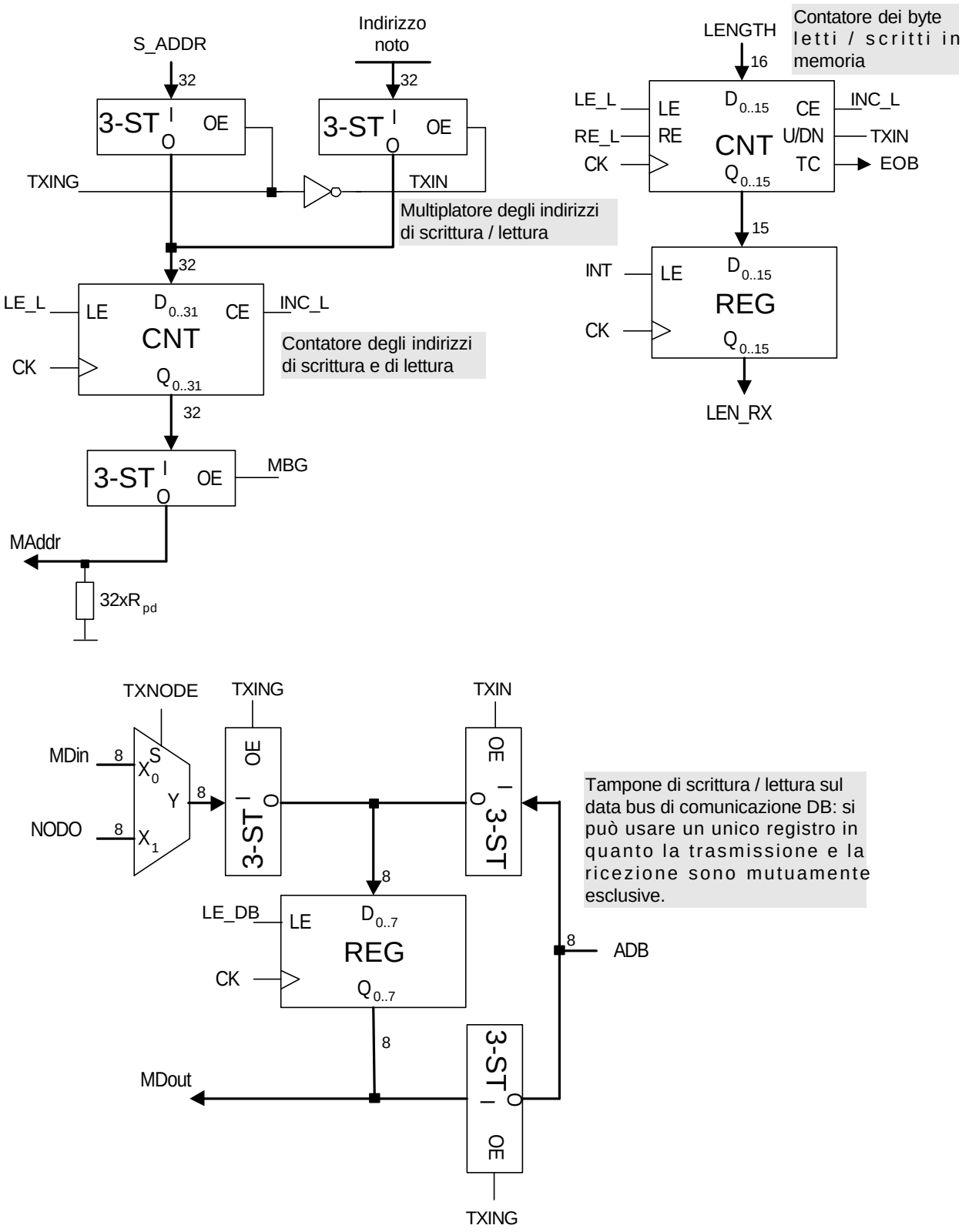
MULTIMICRO: interfaccia bus di comunicazione



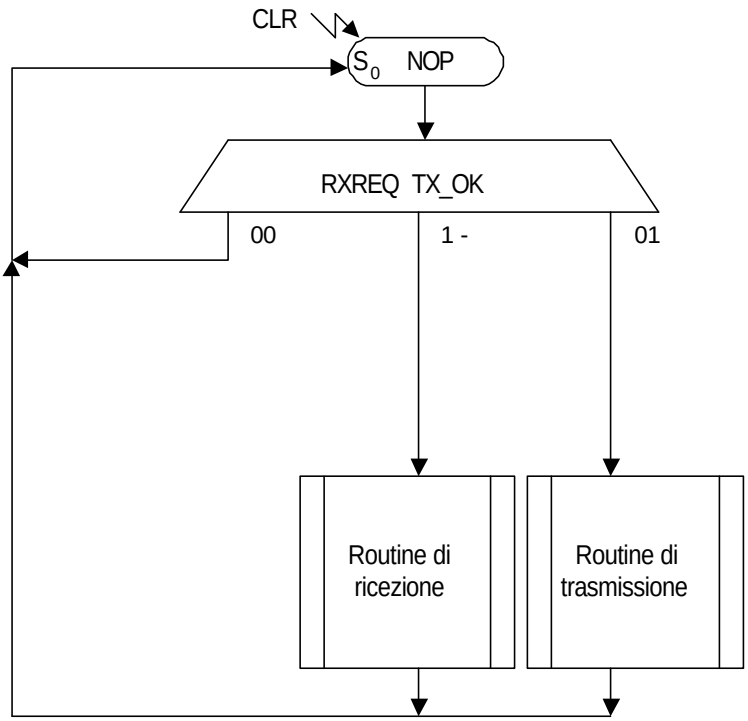




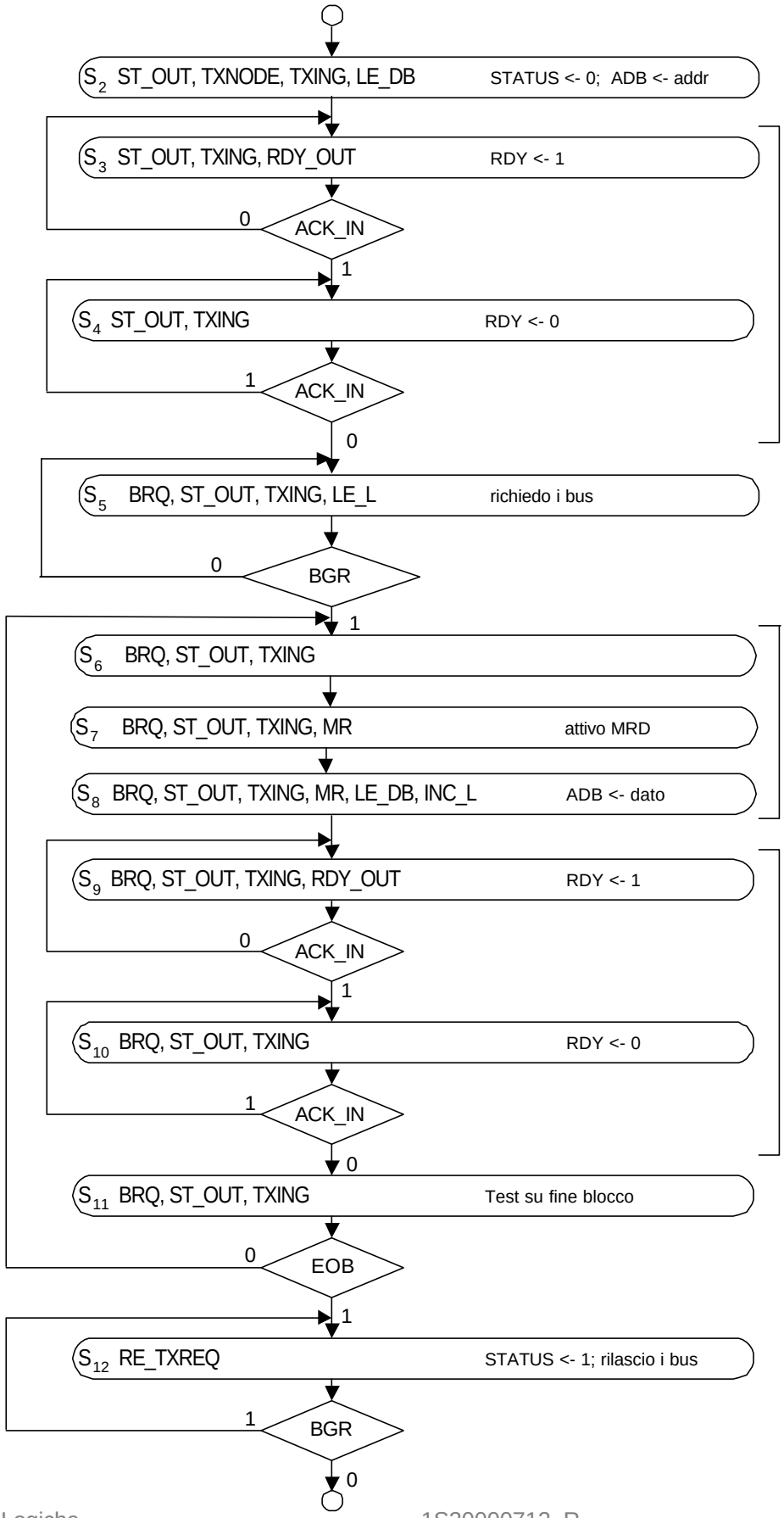
MULTIMICRO: blocco gestione puntatori



MULTIMICRO: SCO - flowchart



MULTIMICRO: SCO - flowchart
Routine di trasmissione

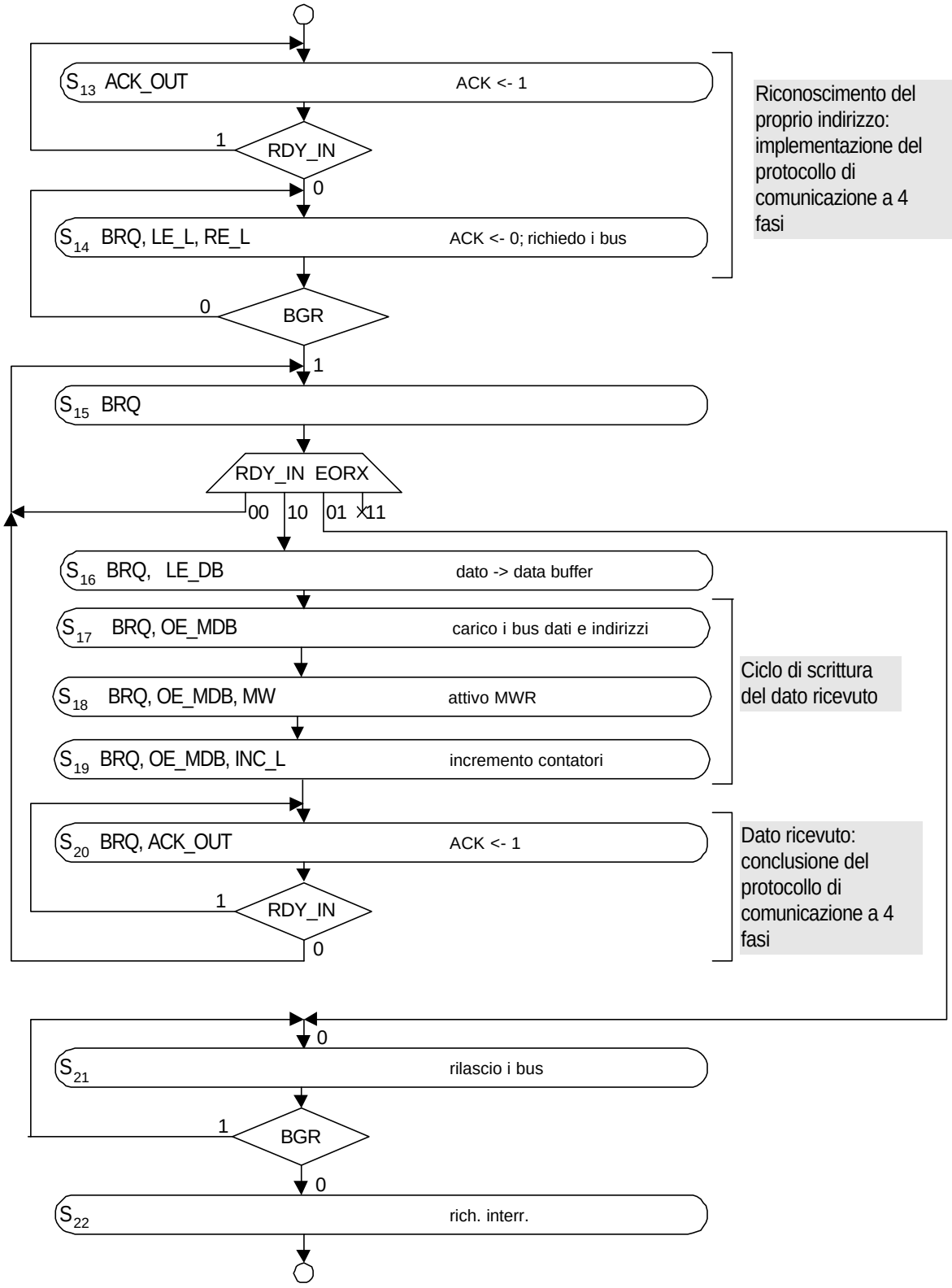


Trasmissione dell'indirizzo del nodo destinatario: implementazione del protocollo di comunicazione a 4 fasi

Ciclo di lettura del dato da trasmettere

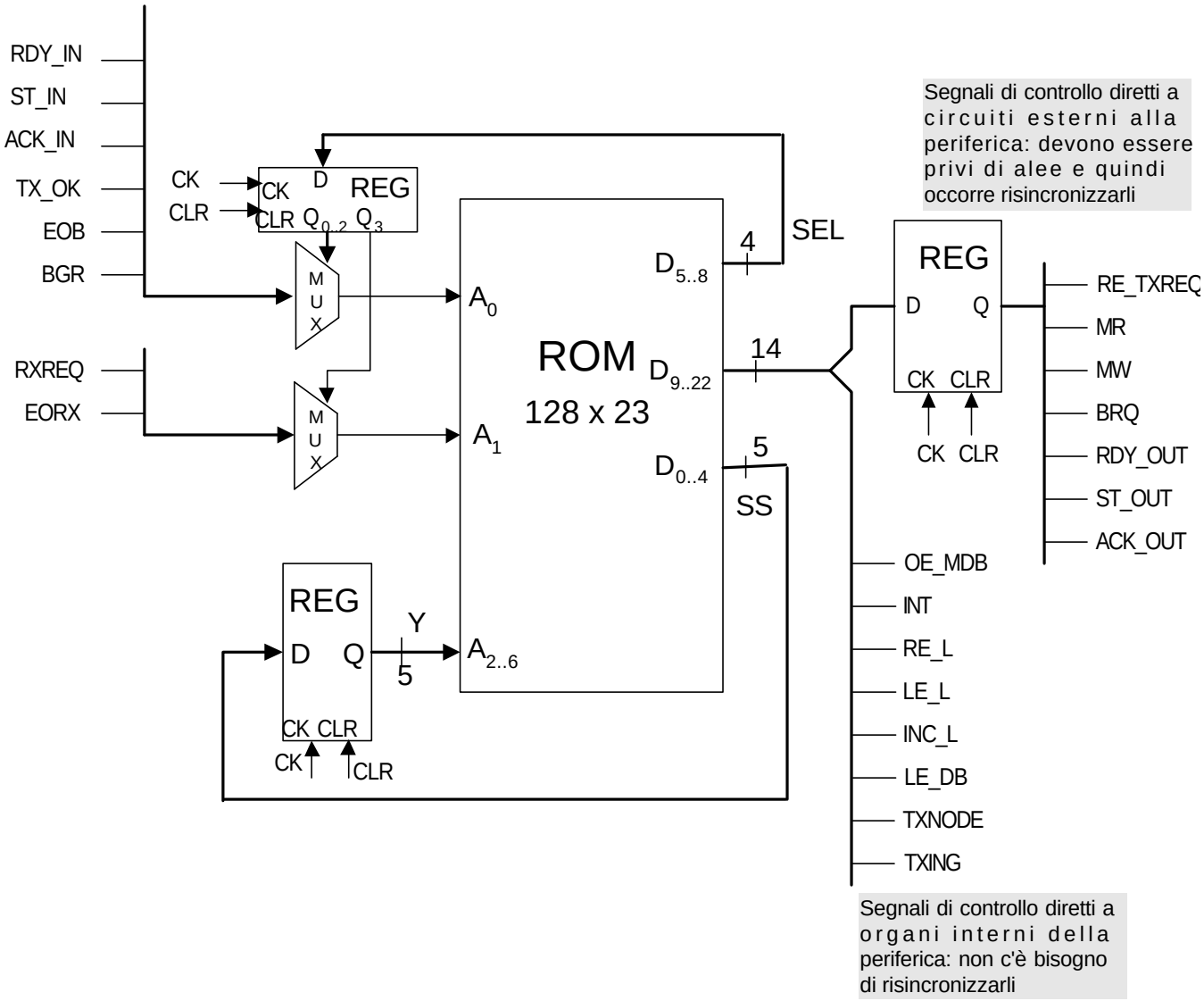
Trasmissione del dato: implementazione del protocollo di comunicazione a 4 fasi

MULTIMICRO: SCO - flowchart
Routine di ricezione



MULTIMICRO: SCO - struttura HW microprogrammata

Scegliendo il modello strutturale di tipo Mealy si ottiene la struttura seguente:



Note

I segnali di uscita sono stati partizionati in due blocchi: soltanto i segnali diretti all'esterno sono tamponati, tutti gli altri hanno la funzione di abilitazione dei componenti interni alla periferica e quindi non necessitano di tamponamento. Lo SCO è perciò di tipo Mealy con riguardo a questi ultimi segnali e funziona da D-Mealy con riguardo ai segnali di uscita tamponati. A questo punto va notato che i segnali non tamponati sono stati associati agli stati e non alle transizioni nel diagramma di flusso, anche se è di tipo Mealy: questo significa soltanto che i segnali in questione non dipendono dagli ingressi e quindi possono essere attribuiti agli stati per semplicità di rappresentazione .
A fronte di questa posizione, in fase di codifica del microprogramma bisognerà fare attenzione a calcolare i segnali di uscita dei due blocchi in modo da anticipare la presentazione dei segnali assoggettati a tamponamento.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 12-7-2000

Studente:_____ **Docente:**_____

D1 Date le due stringhe numeriche da interpretare nella rappresentazione in virgola mobile nel formato a 32 bit: S, E, M:
83500000h
01500000h
calcolare la stringa corrispondente al prodotto dei due valori.

D2 Implementare un decodificatore con 6 ingressi di dato e un ingresso di abilitazione utilizzando moduli decodificatori con 4 ingressi di dato e un ingresso di abilitazione.

D3 Codificare la macchina asincrona descritta dalla tabella con il metodo degli stati instabili.

S	00	01	11	10	Z
A	**A**	**A**	D	B	0
B	**B**	D	D	**B**	1
C	A	A	**C**	**C**	0
D	A	**D**	**D**	C	1

D4 Descrivere il modello strutturale di una rete con L ingressi a livello e P ingressi impulsivi; caratterizzarne i componenti con riferimento alle caratteristiche temporali dei segnali impulsivi (larghezza minima, intervallo minimo di ripetizione).

D5 Scrivere una routine assembler PD32 che calcoli la funzione booleana:
$Y = X_0 X_1 + X_2 X_3 + X_4 X_5 + X_6 X_7$
essendo X_0 . . X_7 i bit di una variabile predisposta in memoria all'indirizzo BITS. Il valore di Y va memorizzato all'indirizzo successivo.

Esercizio (2S20000712-D1)

Date le due stringhe numeriche da interpretare nella rappresentazione in virgola mobile nel formato a 32 bit: S, E, M:
83500000h
01500000h
calcolare la stringa corrispondente al prodotto dei due valori.

Il primo passo è provvedere alla conversione delle stringhe nel codice binario per isolare i tre campi di ognuna:

STRINGA	Segno	Esponente	Mantissa	Valore decimale
83500000h	1	00000110	10100000000000000000000	$-0.625\ 2^6 = -40$
01500000h	0	00000010	10100000000000000000000	$+0.625\ 2^2 = 2.5$

Il prodotto (-100 decimale) si deve ritrovare eseguendo i calcoli in virgola mobile in base 2:

Segno

$S = (S_0 + S_1) \text{mod} 2 = 1 = 1_2$

Mantissa

$M = M_0 \cdot M_1 = (5 \cdot 2^{-3}) \cdot (5 \cdot 2^{-3}) = 25 \cdot 2^{-6} = (25 \cdot 2^{-5}) \cdot 2^{-1}$

dove il termine tra parentesi nell'ultimo membro rappresenta la mantissa normalizzata e il fattore esterno deve essere combinato con l'esponente;
pertanto la mantissa è data da: 11001000000000000000000

Esponente

$E = E_0 + E_1 - 1 = 6 + 2 - 1 = 7 = 0111_2$

In cui il termine –1 tiene conto del calcolo della mantissa.

Ora la tavola può essere completata con il risultato richiesto:

STRINGA	Segno	Esponente	Mantissa	Valore decimale
83E40000h	1	00000111	11001000000000000000000	$-0.78125\ 2^7 = -100$

Esercizio (2S20000712-D2)

Implementare un decodificatore con 6 ingressi di dato e un ingresso di abilitazione utilizzando moduli decodificatori con 4 ingressi di dato e un ingresso di abilitazione.

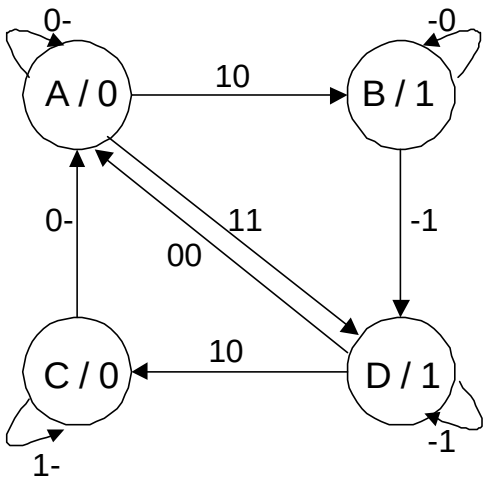
Cfr. “Appunti Integrativi”.

Esercizio (2S20000712-D3)

Codificare la macchina asincrona descritta dalla tabella con il metodo degli stati instabili.

S	00	01	11	10	Z
A	**A**	**A**	D	B	0
B	**B**	D	D	**B**	1
C	A	A	**C**	**C**	0
D	A	**D**	**D**	C	1

Conviene descrivere la macchina specificata con il diagramma degli stati, su cui si evidenziano gli eventuali cicli dispari:



Sul grafo si possono notare due cicli dispari (ABD, ACD), che possono essere resi pari aggiungendo uno stato instabile su ciascun ciclo; in particolare si può notare che un unico stato instabile posizionato tra A e D renderebbe entrambi i cicli pari simultaneamente; tale stato sarebbe instabile per i due ingressi diversi 00 e 11 e guiderebbe il punto di lavoro della macchina da D a A e viceversa rispettivamente.

Tuttavia, si può notare che:
- il passaggio da A a D deve avvenire per ingresso 11;
- in D si perviene anche da B con ingresso 11;
- esiste già un collegamento tra A e B (adiacenza da soddisfare);

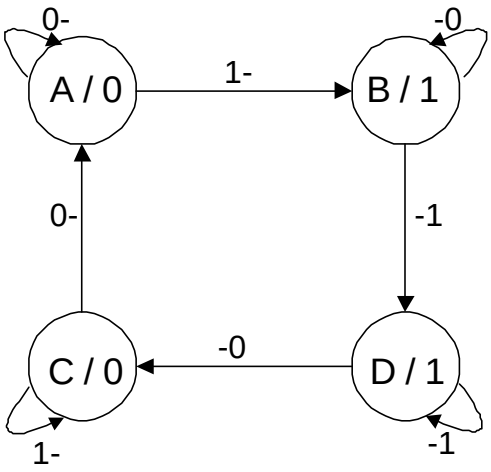
ne segue che si può modificare il diagramma connettendo A con B per ingresso 11, rendendo B instabile per ingresso 11. Con tale modifica il passaggio da A a D avverrebbe tramite il passaggio per B.

Analogamente, si può notare che:
- il passaggio da D a A deve avvenire per ingresso 00;
- in A si perviene anche da C con ingresso 00;
- esiste già un collegamento tra D e C (adiacenza da soddisfare);

ne segue che si può modificare il diagramma connettendo D con C per ingresso 00, rendendo C instabile per ingresso 00. Con tale modifica il passaggio da D a A avverrebbe tramite il passaggio per C.

Inoltre i due stati resi instabili producono il valore dell'uscita dello stato di destinazione, riproducendo la stessa situazione che si sarebbe imposta se si fossero introdotti due stati instabili aggiuntivi.

In definitiva il diagramma degli stati può essere trasformato nel seguente:



che è privo di cicli dispari e definisce connessioni adiacenti tra quattro stati che possono essere calati senz'altro su una MK a due variabili.

La tavola di flusso corrispondente al diagramma modificato diventa:

S	00	01	11	10	Z
A	**A**	**A**	B	B	0
B	**B**	D	D	**B**	1
C	A	A	**C**	**C**	0
D	C	**D**	**D**	C	1

Esercizio (2S20000712-D4)

Descrivere il modello strutturale di una rete con L ingressi a livello e P ingressi impulsivi; caratterizzarne i componenti con riferimento alle caratteristiche temporali dei segnali impulsivi (larghezza minima, intervallo minimo di ripetizione).

Cfr. testo “Reti Sequenziali”.

Esercizio (2S20000712-D5)

Scrivere una routine assembler PD32 che calcoli la funzione booleana:
$Y = X_0 X_1 + X_2 X_3 + X_4 X_5 + X_6 X_7$
essendo $X_0 . . X_7$ i bit di una variabile predisposta in memoria all'indirizzo BITS. Il valore di Y va memorizzato all'indirizzo successivo.

```
;Scrivere una routine assembler PD32 che calcoli la funzione booleana:
;Y = X0 X1 + X2 X3 + X4 X5 + X6 X7
;essendo X0 . . X7 i bit di una variabile predisposta in memoria
;all'indirizzo BITS. Il valore di Y va memorizzato all'indirizzo successivo.

        org 400h                ;inizio programma

; *************************
; COSTANTI
; *************************

        STACK       EQU 2800h   ;inizio area di stack
                                ;limitato a 2800h per consentire la simulazione

; *************************
; VARIABILI IN MEMORIA
; *************************

        BITS   DB 027h, 00h     ;variabile specificata, risultato iniz. a 0

; *************************
; CODICE
; *************************

        code                    ;inizio istruzioni

main:
        movl #STACK,r7          ;inizializza R7 quale SP
        seti                    ;abilita interruzioni (SP è stato inizializzato)

        jsr andor

        halt                    ;serve solo per bloccare il simulatore


; *************************
; ROUTINE
; *************************

andor:
        push r0                 ;salvo i registri nello stack
        push r1
```

```
        movb #000h,BITS+1       ;inizial. risultato in RAM a 0
        movb BITS,r0            ;carico gli 8 bit in r0
        movb r0,r1              ;e li copio anche in r1…
        asrb #1,r1              ;coposiziono x1, x3, x5 e x7 in r1
                                ;con x0, x2, x4 e x6 in r0
        andb #055h,r0           ;azzero le posizioni 1, 3, 5, 7
        andb #055h,r1           ;di r0 e r1
        andb r1,r0              ;AND delle quattro coppie di bit
        jz exit                 ;se l'AND dà tutti 0, allora l'OR è 0
        movb #001h,BITS+1       ;altrimenti è 1, e lo scrivo in RAM
exit:
        pop r1                  ;ripristino i registri usati
        pop r0
        ret

        end                     ;fine della compilazione
```

```
;Scrivere una routine assembler PD32 che calcoli la funzione booleana:
;Y = X0 X1 + X2 X3 + X4 X5 + X6 X7
;essendo X0 . . X7 i bit di una variabile predisposta in memoria
;all'indirizzo BITS. Il valore di Y va memorizzato all'indirizzo successivo.


  org 400h    ;inizio programma

; *************************
; COSTANTI
; *************************

  STACK   EQU 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione

; *************************
; VARIABILI IN MEMORIA
; *************************

  BITS  DB 027h, 00h  ;variabile specificata, risultato inz. a 0


; *************************
; CODICE
; *************************

  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)

  jsr andor

  halt      ;serve solo per bloccare il simulatore


; *************************
; ROUTINE
; *************************

andor:
  push r0     ;salvataggio dei registri nello stack
  push r1

  movb #000h,BITS+1 ;inizial. risultato in RAM a 0
  movb BITS,r0    ;carico gli 8 bit in r0
  movb r0,r1    ;e li copio anche in r1
  asrb #1,r1    ;coposiziono x1, x3, x5 e x7 in r1
        ;con x0, x2, x4 e x6 in r0
  andb #055h,r0   ;azzero le posizioni 1, 3, 5, 7
  andb #055h,r1   ;di r0 e r1
  andb r1,r0    ;AND delle quattro coppie di bit
  jz exit     ;se l'AND dà tutti 0, allora l'OR è 0
  movb #001h,BITS+1 ;altrimenti è 1, e lo scrivo in RAM
exit:
  pop r1      ;ripristino i registri usati
  pop r0
  ret

  end     ;fine della compilazione
```

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 14-9-2000

Studente:_____ **Docente:**_____

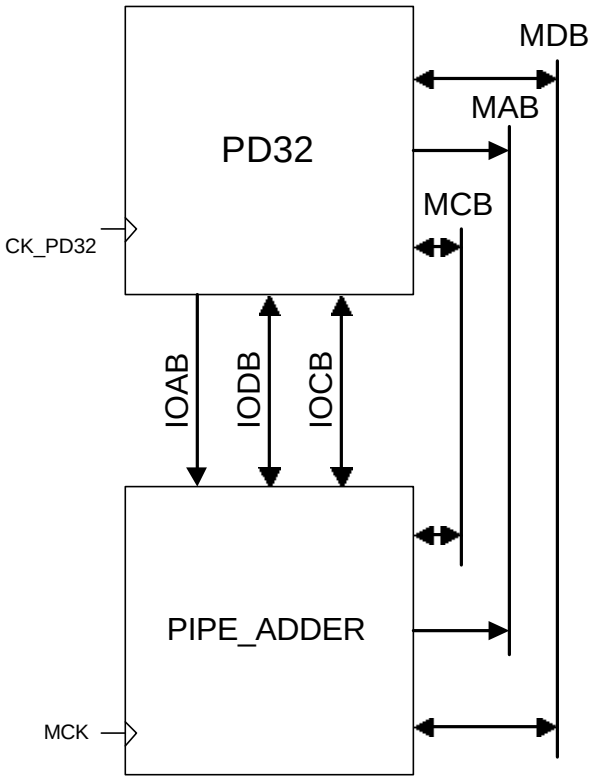
Una periferica riceve due flussi dati da due convertitori A/D a 8 bit con tempo di conversione di 5nsec. Per ogni coppia di tali dati viene calcolata la media aritmetica mediante un *addizionatore pipeline* realizzato da half-adder e porte logiche. Il tempo di ritardo di una porta logica è di 1nsec ed il tempo di setup dei registri è di 1nsec.
La somma in uscita dall'addizionatore deve essere memorizzata in una memoria locale di 1Kbyte.
Una volta riempita la memoria, la periferica interrompe la memorizzazione e invia un interrupt ad un processore PD32. Quest'ultimo provvede alla lettura dei dati dalla memoria interna alla periferica che vede nel suo spazio di indirizzamento a partire dall'indirizzo FFFFFC00.

L'addizionatore pipeline è costituito da otto stadi: ogni stadio deve essere senza propagazione di riporto. Il processo di acquisizione è attivato dal PD32. La memoria locale ha un tempo minimo di ciclo di scrittura di 5nsec.

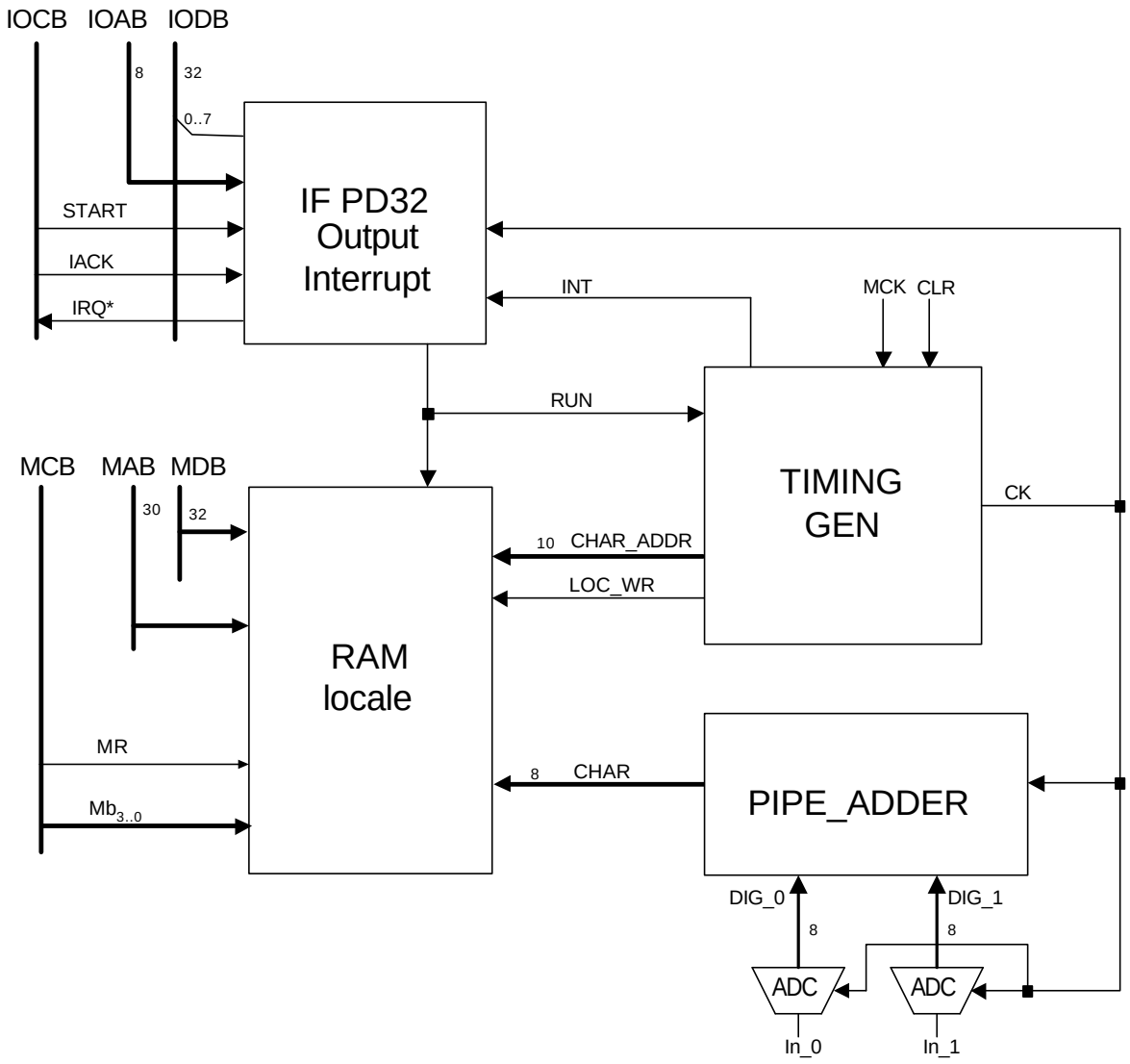
Avvertimenti importanti:
1- si tenga presente la latenza della pipeline nel processo di memorizzazione delle somme;
2- si ricordi che il ciclo di scrittura della RAM deve essere di tre periodi di clock.

Si richiedono:
1) lo schema logico della periferica;
2) la frequenza massima di funzionamento della periferica.

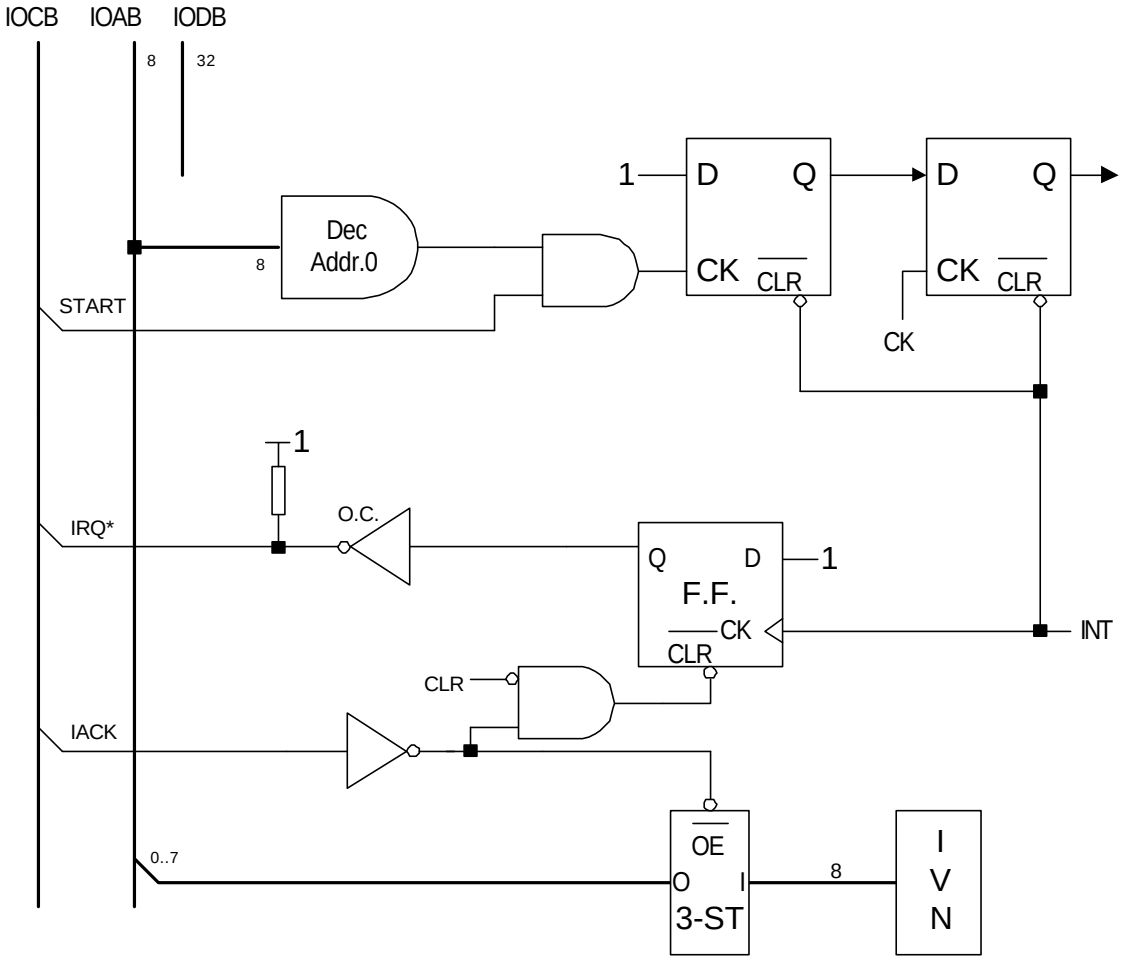
PIPE-ADDER: sistema esterno



PIPE_ADDER: Schema a blocchi



PIPE_ADDER: IF PD32 - output - interrupt

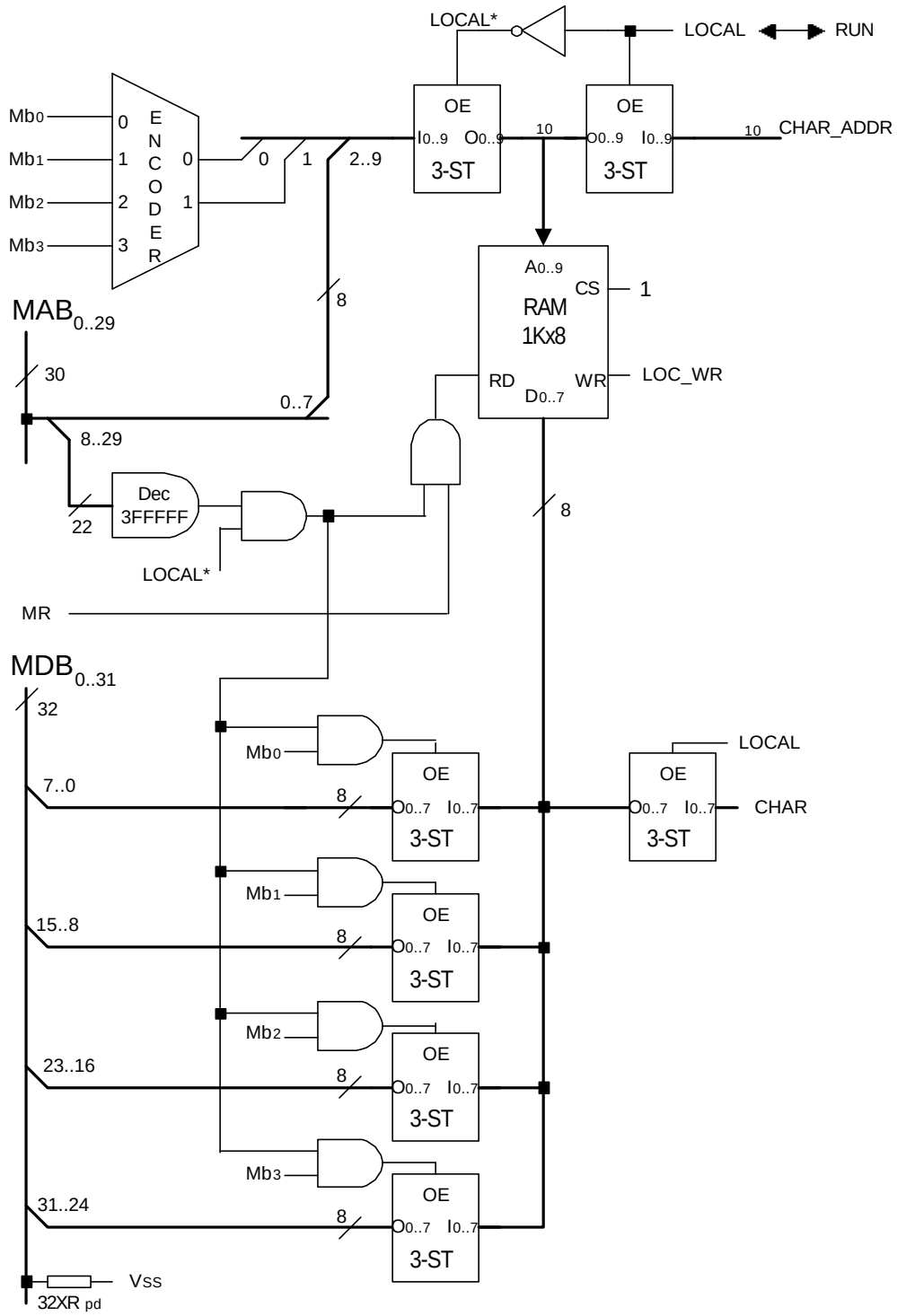


Routine assembler

```
ORG 400h
. . .
Addr0 EQU 5Ah
FLAG DB 00h
. . .
CODE
. . .
main:
. . .
cmpb #00h,FLAG
jz altro
jsr usa_dati
jsr nuova_conv
altro:
. . .
jmp main

nuova_conv:
outb #0,FLAG
start Addr0
ret

usa_dati:
. . .
ret

DRIVER 10h,2000h
movb #01h,FLAG
rti
```

PIPE_ADDER: RAM LOCALE

- 8 bit
- PERIFERICA: WR
- PD32: RD; ind. iniziale: FFFFFC00h

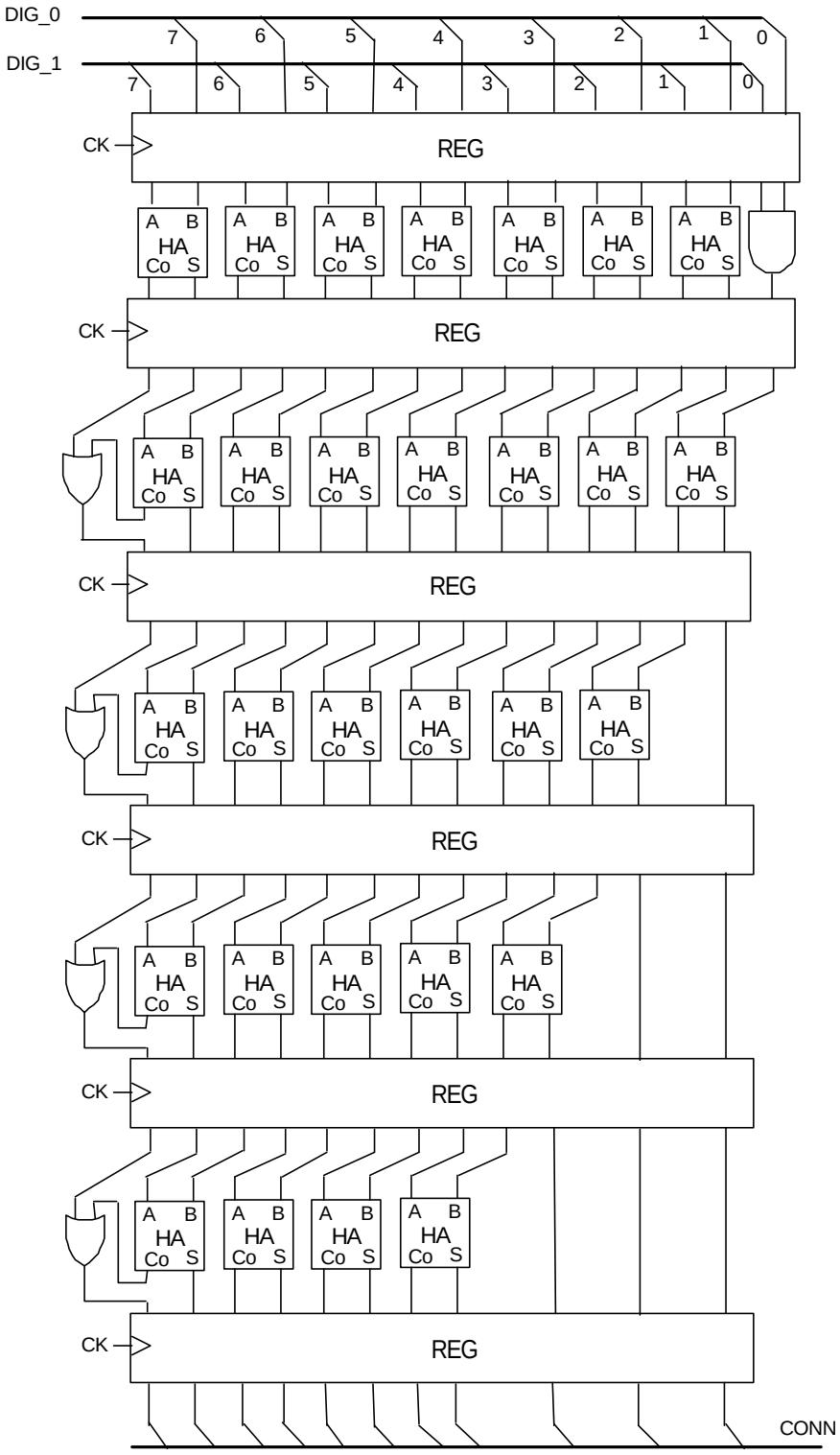


- LOCAL viene attivato (disattivato) all'inizio (al termine) dell'elaborazione richiesta dal micro.

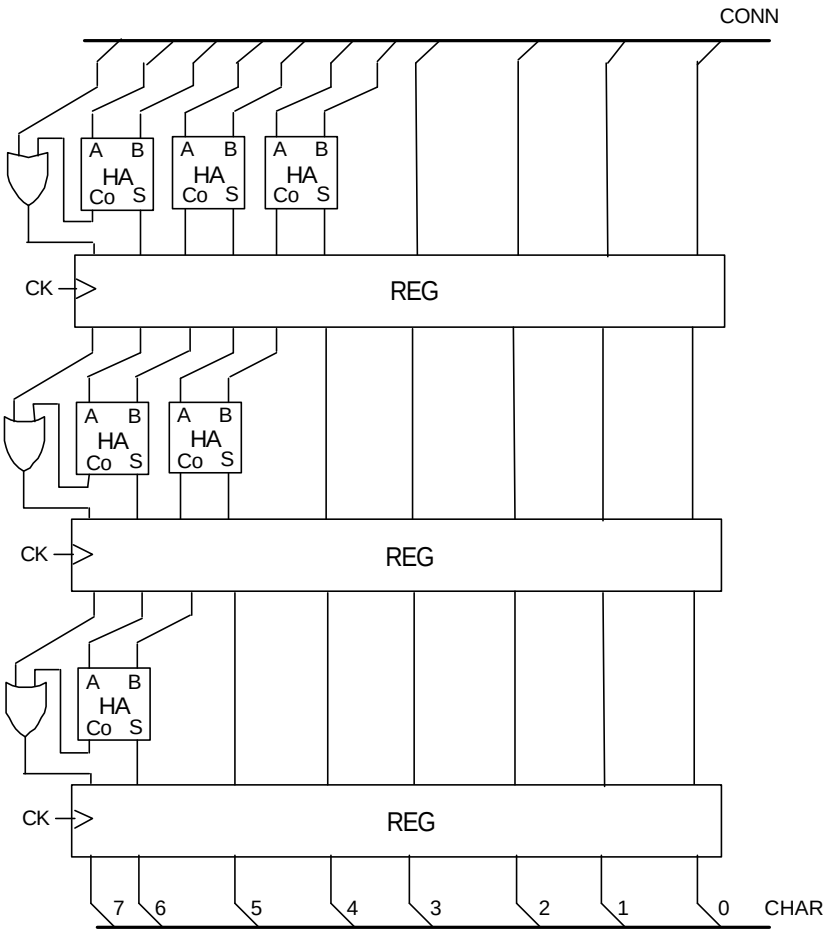
- Il PD32 legge la RAM locale a byte (es: MOV**B** RAM_addr,R0).

- Indirizzi PD32 a 32 bit della RAM locale: FFFFFC00h.. FFFFFFFFh; MAB (30 bit): 3FFFFF00h..3FFFFFFFh; parte costante: 3FFFFFh.

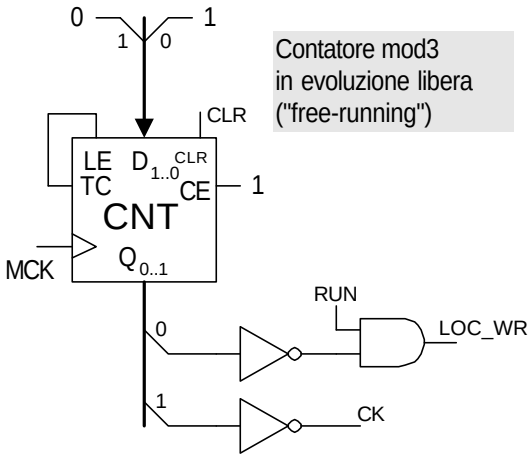
PIPE_ADDER: pipeline



PIPE_ADDER: pipeline



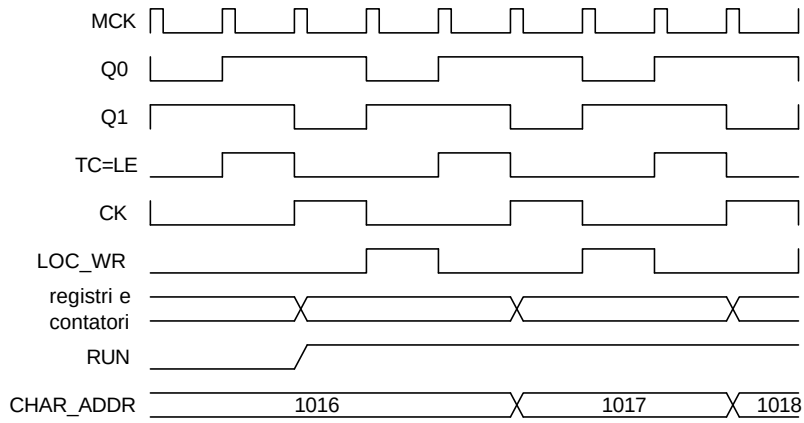
PIPE-ADDER: Timing gen.

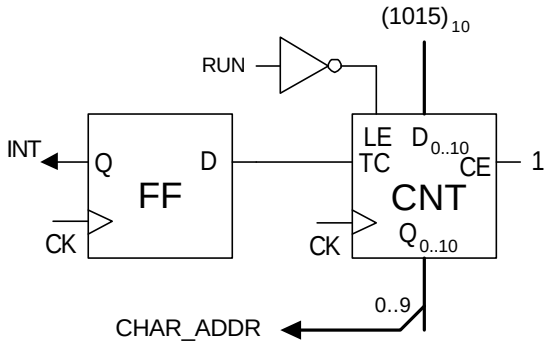


Contatore mod3
in evoluzione libera
("free-running")

I segnali di timing (CK, LOC_WR) vengono distribuiti ai blocchi esterni (SCA, RAM) con la giusta relazione di fase.

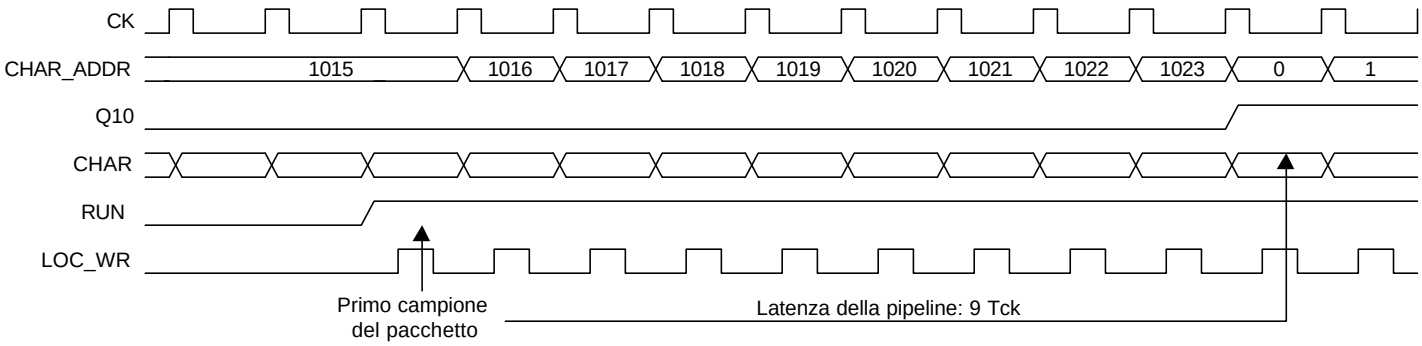
Temporizzazione: generatore dei segnali di timing



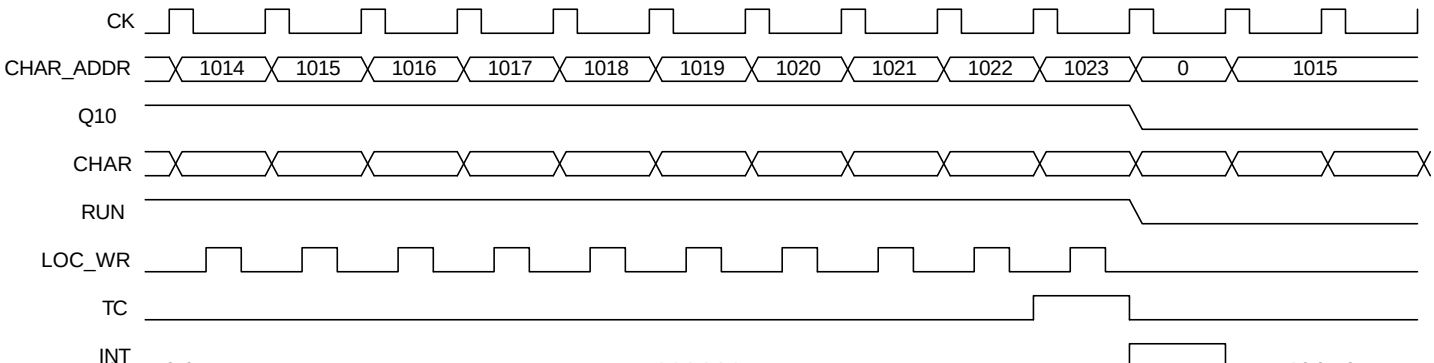


Contatore a 11 bit generatore degli indirizzi - a 10 bit: il bit Q10 gestisce la latenza (9 Tck) del sommatore pipeline mediante la scrittura dei primi 9 dati di uscita della pipeline nelle ultime 9 locazioni della RAM, in modo da scrivere il primo valore utile all'indirizzo 0. Le ultime 9 locazioni della RAM verranno riscritte con i valori corretti degli ultimi 9 campioni acquisiti. Questo meccanismo richiede ovviamente un bit di conteggio in eccesso delle linee di indirizzo della RAM.

Temporizzazione: inizio acquisizione



Temporizzazione: fine acquisizione



NOTE

- Calcolo della frequenza di CK:
  La velocità della pipeline è limitata dalla cascata del convertitore ADC (5 ns) e del registro (t_{setup}= 1 ns), che impongono un periodo di CK minimo pari a 6 ns. Va aggiunto che per poter scrivere in RAM un dato in ciascun periodo di CK occorre generare un impulso di MWR centrato nel periodo di CK e di durata 1/3 del periodo di CK; questo vincolo impone la disponibilità di un segnale di orologio (MCK) ad una frequenza 3 volte quella di CK, quindi con periodo pari a 2 ns (F_{MCK}=500 MHz).

- La soluzione completamente sincrona richiederebbe la distribuzione di MCK a tutti i registri, insieme ad un segnale di abilitazione a frequenza 1/3 F_{MCK} ricavato da un contatore; tuttavia, si può notare che tale contatore deve produrre, oltre al segnale di abilitazione per tutti i registri dello SCA (la pipeline e il contatore-indirizzi), un solo altro segnale impulsivo di scrittura della RAM, legato al primo da uno sfasamento preciso (cfr. punto precedente); in questo caso il primo segnale può essere impiegato direttamente come clock (CK) dello SCA, avendo cura di predisporre l'impulso di scrittura nella corretta relazione di fase con CK, mediante opportuna decodifica del contatore. Il vantaggio più notevole che ne deriva è la semplificazione dei registri dello SCA, che con tale accorgimento non devono essere predisposti con l'abilitazione al caricamento. Il generatore di timing diventa un blocco asincrono rispetto allo SCA, nei cui confronti agisce da semplice *generatore di clock esterno*; è opportuno sottolineare che di conseguenza anche in questo caso lo scambio tra i registri dello SCA è completamente sincrono.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 14-9-2000

Studente:_____ **Docente:**_____

D1 Estendere la rappresentazione in virgola mobile nel formato a 32 bit con un codice a correzione di un singolo errore su ciascuno dei tre campi S, E, M separatamente.

D2 Progettare un contatore modulo 4 dotato di ingresso di abilitazione al conteggio e avanzamento in avanti/indietro secondo il codice di Gray.

D3 Indicare il numero massimo $|S|_{max}$ degli stati definibili su una tavola primitiva di una macchina asincrona con |X| variabili di ingresso e |Z| di uscita.

D4 Descrivere la struttura di una SCA di un processore con 8 registri interni in grado di eseguire operazioni a due operandi in un periodo di clock.

D5 Si dispone di un PD32 per effettuare il collaudo di un circuito integrato combinatorio con 5 ingressi e una uscita, sospettato di essere guasto. Il collaudo consiste nella verifica del comportamento del circuito combinatorio secondo la tavola di verità memorizzata all'indirizzo TRUTAB della memoria del PD32. L'esito (sano/guasto) del test va registrato in una variabile all'indirizzo TEST. Disegnare l'interfaccia di I/O del processore e scrivere la routine di test.

Esercizio (2S20000914-D1)

Estendere la rappresentazione in virgola mobile nel formato a 32 bit con un codice a correzione di un singolo errore su ciascuno dei tre campi S, E, M separatamente.

La rappresentazione numerica specificata riserva 1 bit a S, 8 bit a E e 23 bit a M. Occorre trasformare ognuno di tali dati separatamente in un codice di Hamming a distanza 3, aggiungendo un numero k di bit di controllo come stabilito dalla relazione:

$$n \leq 2^k - k - 1$$

L'applicazione della relazione ai tre casi in questione è sintetizzata nella tabella seguente:

campo	n	k	n+k
S	1	2	3
E	8	4	12
M	23	5	28

Complessivamente si hanno 11 bit di controllo per i 32 bit della parola del codice irridondante.

Osservazione:

Si noti la codifica del segno, S, con tre bit:
0: 000; 1: 111;
la decodifica corrisponderà al criterio a maggioranza, cioè 2 su 3:

Parola ricevuta	Simbolo decodificato
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

Esercizio (2S20000914-D2)

Progettare un contatore modulo 4 dotato di ingresso di abilitazione al conteggio e avanzamento in avanti/indietro secondo il codice di Gray.

Il contatore viene progettato come rete sincrona, il cui comportamento è descritto dalla tabella degli stati seguente; come richiesto dalla specifica, la macchina sincrona è dotata dei due segnali a livello CE (Count Enable) e U/D (1: Up; 0: Down); inoltre conviene codificare subito i 4 stati, secondo il codice di Gray: 00, 01, 11, 10.

funzione	CE	U/D	Y1	Y0	Y1’	Y0’
conteggio bloccato	0	0	0	0	0	0
	0	0	0	1	0	1
	0	0	1	1	1	1
	0	0	1	0	1	0
	0	1	0	0	0	0
	0	1	0	1	0	1
	0	1	1	1	1	1
	0	1	1	0	1	0
conteggio decrem.	1	0	0	0	1	0
	1	0	0	1	0	0
	1	0	1	1	0	1
	1	0	1	0	1	1
conteggio increm.	1	1	0	0	0	1
	1	1	0	1	1	1
	1	1	1	1	1	0
	1	1	1	0	0	0

T1	T0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1	0
0	1
1	0
0	1
0	1
1	0
0	1
1	0

A destra sono riportate le due funzioni T1 e T0 corrispondenti alle variazioni tra le coppie Y1,Y1’ e Y0,Y0’ rispettivamente; cioè: $T1=Y1 \oplus Y1'$ e $T0=Y0 \oplus Y0'$. In questo modo il calcolo viene impostato per predisporre una implementazione del registro di stato del contatore con flip-flop di tipo T.

La sintesi completa richiede la minimizzazione di 2 MK (le 2 variabili di stato) a 4 variabili (le 2 variabili di stato + 2 di ingresso):

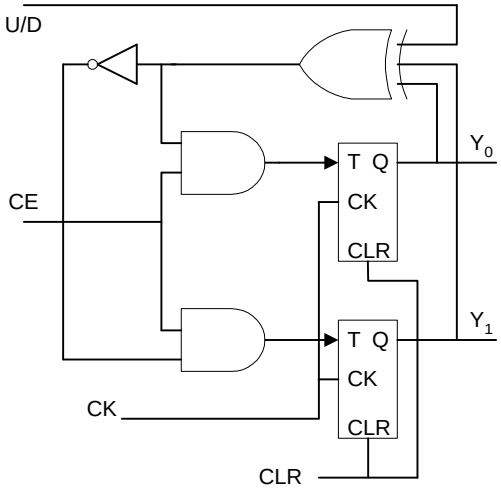
Y1 Y0 / CE U/D	00	01	11	10
00				
01				
11		1		1
10	1		1	

$T1 = CE\,(U/D \oplus Y1 \oplus Y0)$

Y1 Y0 / CE U/D	00	01	11	10
00				
01				
11	1		1	
10		1		1

$T0 = CE\,(U/D \oplus Y1 \oplus Y0)$

Nelle due espressioni booleane è stata messa in evidenza la funzione XOR, piuttosto che minimizzare a tre livelli, per compattezza di rappresentazione.
La rete sequenziale è costruita di conseguenza:



Si noti l'ingresso CLR per inizializzare il contatore a 00 indipendentemente da CK.

Esercizio (2S20000914-D3)

Indicare il numero massimo $|S|_{max}$ degli stati definibili su una tavola primitiva di una macchina asincrona con |X| variabili di ingresso e |Z| di uscita.

Su ogni riga della tavola primitiva viene inserito uno e un solo stato stabile; pertanto, sulla tavola per ogni vettore di uscita si possono definire tanti stati quante sono le colonne, cioè $2^{|X|}$. Ogni stato stabile può poi condividere la stessa colonna con tanti stati quanti sono i vettori di uscita distinti, cioè $2^{|Z|}$. In definitiva il numero massimo di stati ammonta a: $2^{(|X|+|Z|)}$.

Esercizio (2S20000914-D4)

Descrivere la struttura di una SCA di un processore con 8 registri interni in grado di eseguire operazioni a due operandi in un periodo di clock.

Cfr. testo “Reti Sequenziali”.

Esercizio (2S20000914-D5)

Si dispone di un PD32 per effettuare il collaudo di un circuito integrato combinatorio con 5 ingressi e una uscita, sospettato di essere guasto. Il collaudo consiste nella verifica del comportamento del circuito combinatorio secondo la tavola di verità memorizzata all'indirizzo TRUTAB della memoria del PD32. L'esito (sano/guasto) del test va registrato in una variabile all'indirizzo TEST. Disegnare l'interfaccia di I/O del processore e scrivere la routine di test.

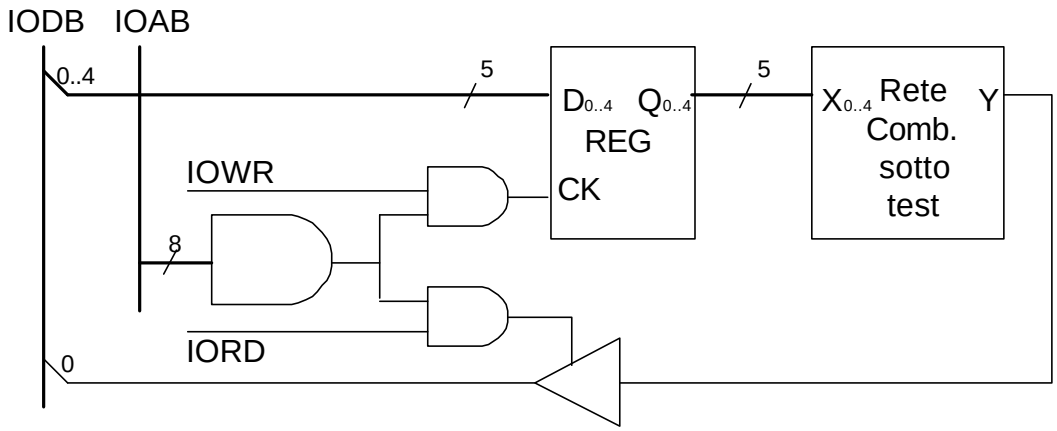
Impostazione del sistema

HW: La periferica-tester dovrà essere provvista di un registro su cui il processore scrive (con una istruzione OUT) le 2^5=32 configurazioni degli ingressi del circuito combinatorio, corrispondenti alle 32 righe della tavola di verità della funzione che il circuito sotto test deve implementare (se sano); l'uscita del circuito combinatorio deve poter essere letta dal processore (con una istruzione IN), che quindi (cioè il suo SW) si aspetta di trovare nella periferica una interfaccia dotata di un registro di uscita e un buffer (3-state) di ingresso.

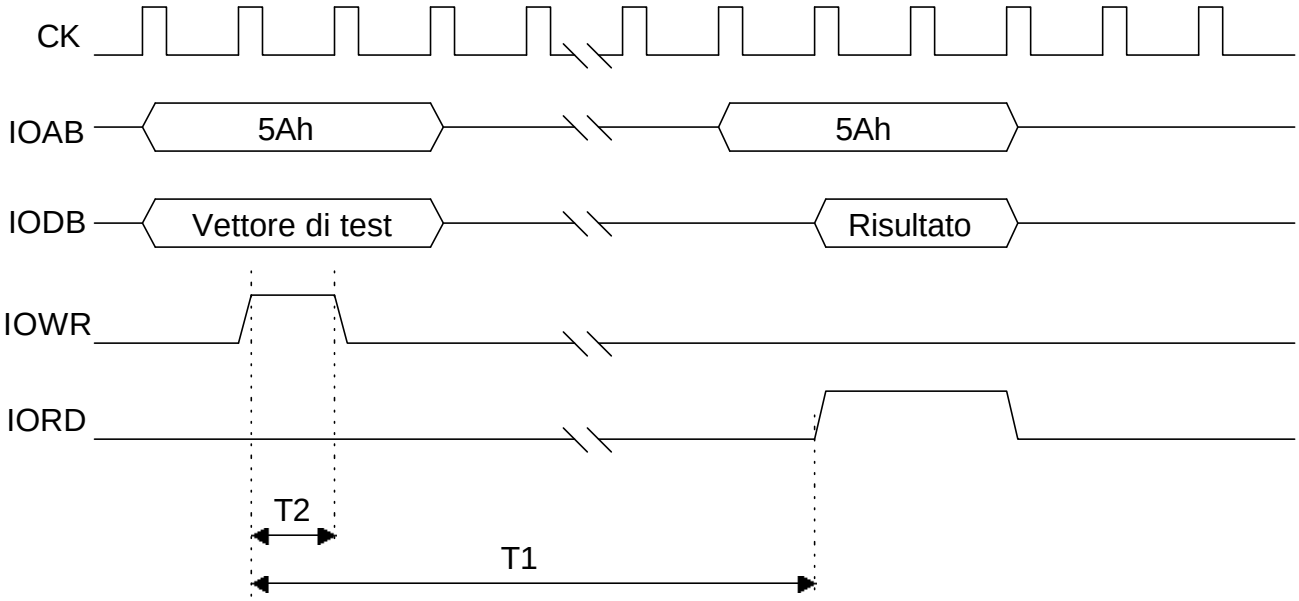
SW: Per quello che riguarda il SW, una routine provvederà ad eseguire un ciclo in cui un contatore ad incremento produce le 32 configurazioni della tavola di flusso; il ciclo comprenderà un'istruzione di OUT, una di IN (immediatamente seguente la OUT, in quanto il circuito è combinatorio e si suppone che possa produrre la risposta valida nel tempo che passa tra l'esecuzione delle due istruzioni), l'accumulo di ogni singolo bit in una Longword, che alla fine del ciclo verrà confrontata con la Longword memorizzata all'indirizzo TRUTAB. In alternativa si può procedere ad un test comparativo bit a bit, con l'obiettivo di arrestare il test appena si dovesse trovare una disuguaglianza tra il bit atteso e quello effettivamente rilevato.

Interfaccia HW

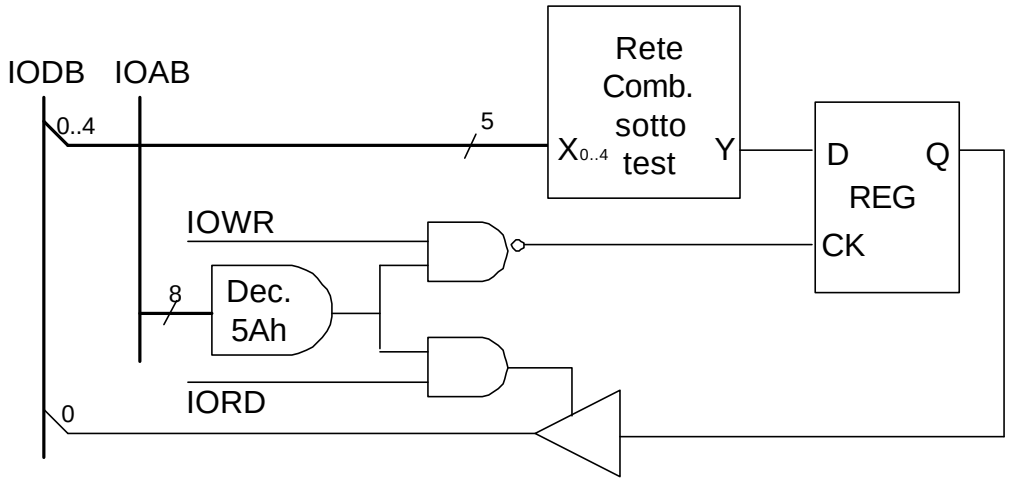
L'interfaccia di I/O è descritta nella figura seguente, in cui si è ipotizzato di scrivere (i cinque bit di stimolo alla rete combinatoria) e leggere (l'uscita della rete) allo stesso indirizzo di periferica. Il registro ha la funzione di mantenere i cinque bit di test in ingresso alla rete mentre il processore esegue l'istruzione successiva di lettura del risultato. Il cablato sul bit 0 dell'IO Data Bus; di questo dovrà tenere conto il SW (cfr. routine SW).



La dinamica dell'interfaccia è descritta dal diagramma di temporizzazione seguente, in cui è evidenziato il tempo T1 che ha a disposizione la rete combinatoria sotto test per produrre il risultato; le due istruzioni OUT e IN sono consecutive (cfr. routine SW).



Una seconda soluzione consiste nel memorizzare l'uscita, di un solo bit, invece dell'ingresso della rete combinatoria, a 5 bit, come descritto nella figura seguente. L'idea è quella di stimolare la rete combinatoria sotto test direttamente dal bus dati I/O e memorizzare poi il risultato sul fronte di discesa del segnale IOWR (attivo alto); qui si assume l'ipotesi del tutto ragionevole che la rete combinatoria sotto test abbia un tempo di calcolo non maggiore di un periodo di clock del processore (cfr. il diagramma di temporizzazione).



Il vantaggio di questo secondo approccio è evidentemente la riduzione della complessità del registro (dal numero dei bit di ingresso alla costante 1), e globalmente l'interfaccia si riduce a un flip-flop e un buffer 3-state (che al limite possono essere integrati in un unico componente), oltre alla logica di decodifica.
La dinamica dell'interfaccia è descritta dallo stesso diagramma di temporizzazione precedente, in cui è evidenziato il tempo T2 che ha a disposizione la rete combinatoria

sotto test per produrre il risultato; le due istruzioni OUT e IN sono consecutive (cfr. routine SW, valida per entrambe le interfacce HW).

Routine SW

```
org 400h                        ;inizio programma

; *************************
; COSTANTI
; *************************

    STACK       EQU 2800h   ;inizio area di stack
                            ;limitato a 2800h per consentire la simulazione
    TESTIN      EQU 05Ah    ;indirizzo di input della periferica
    TESTOUT     EQU 05Bh    ;indirizzo di output della periferica
                            ;i due indirizzi di I/O sono stati diversificati
                            ;perché richiesto dal simulatore

; *************************
; VARIABILI IN MEMORIA
; *************************

    TRUTAB      DL 0A51E780Fh   ;tavola di verità della funzione attesa
    TEST        DB  00h         ;inizializzazione risultato


; *************************
; CODICE
; *************************

    code                ;inizio istruzioni

main:
    movl #STACK,r7      ;inizializza R7 quale SP
    seti                ;abilita interruzioni (SP è stato inizializzato)

    jsr tester

    halt                ;serve solo per bloccare il simulatore


; *************************
; ROUTINE
; *************************
```

```
tester:
    push r0             ;salvataggio nello stack
    push r1
    push r2

    movb #0,TEST        ;inizializzazione del risultato 0:OK (1:KO)
    xorl r0,r0          ;r0 contatore che produce i vettori consecutivi di test



;ciclo di test

next:
    outb r0,TESTOUT
    inb TESTIN,r1       ;in r1 viene raccolto il risultato
    andl #01h,r1        ;nei bit 1..7 di R1 viene forzato 0
    asll #1,r2          ;in R2 viene accumulato il risultato parziale
                        ;dapprima scalando R2 di 1 bit a sinistra,
    orl r1,r2           ;e poi copiando il bit 0 di R1 nel bit 0 di R2.
    addb #1,r0          ;incremento del contatore
    cmpb #32,r0         ;test sulla fine del ciclo
    jnz next
    movl TRUTAB,r1
    cmpl r2,r1          ;test sulla rispondenza al risultato atteso
    jz exit
    movb #1,TEST        ;risposta diversa dal risultato atteso: circuito guasto

exit:
    pop r2              ;ripristino dello stack
    pop r1
    pop r0
    ret


    end                 ;fine programma
```

```
;Nella memoria PD32 all'indirizzo TRUTAB
;è allocata la funzione della rete combinatoria
;che deve essere verificata nel ciclo
;di test.


  org 400h    ;inizio programma

; *************************
; COSTANTI
; *************************

  STACK   EQU 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione
  TESTIN  EQU 05Ah  ;indirizzo di input della periferica
  TESTOUT EQU 05Bh  ;indirizzo di output della periferica

; *************************
; VARIABILI IN MEMORIA
; *************************

  TRUTAB  DL 0A51E780Fh ;tavola di verità della funzione attesa
  TEST  DB  00h   ;inizializzazione risultato


; *************************
; CODICE
; *************************


  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)

  jsr tester

  halt      ;serve solo per bloccare il simulatore


; *************************
; ROUTINE
; *************************

tester:
  push r0     ;salvataggio nello stack
  push r1
  push r2

  movb #0,TEST    ;inizializzazione del risultato 0:OK (1:KO)
  xorl r0,r0    ;r0 contatore che produce i vettori consecutivi di test

;ciclo di test

next: outb r0,TESTOUT
  inb TESTIN,r1   ;in r1 viene raccolto il risultato
  andl #01h,r1    ;nei bit 1..7 di R1 viene forzato 0
  asll #1,r2    ;in R2 viene accumulato il risultato parziale
        ;dapprima scalando R2 di 1 bit a sinistra,
  orl r1,r2   ;e poi copiando il bit 0 di R1 nel bit 0 di R2.
  addb #1,r0    ;incremento del contatore
  cmpb #32,r0   ;test sulla fine del ciclo
```

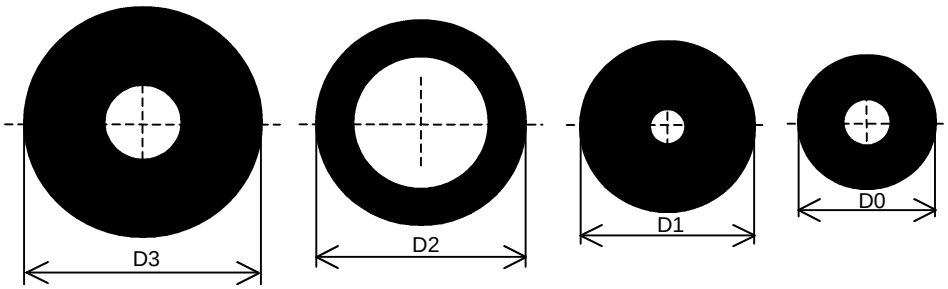
```
  jnz next
  movl TRUTAB,r1
  cmpl r2,r1    ;test sulla rispondenza al risultato atteso
  jz exit
  movb #1,TEST    ;risposta diversa dal risultato atteso: circuito guasto

exit: pop r2      ;ripristino dello stack
  pop r1
  pop r0
  ret



  end     ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DEL 13-10-2000

STUDENTE:_____ DOCENTE:_____

Su un nastro trasportatore di una linea di produzione di componenti meccanici transitano dischi di quattro diametri distinti $D_0<D_1<D_2<D_3$, dotati di un foro centrale, come schematizzato nella figura seguente.



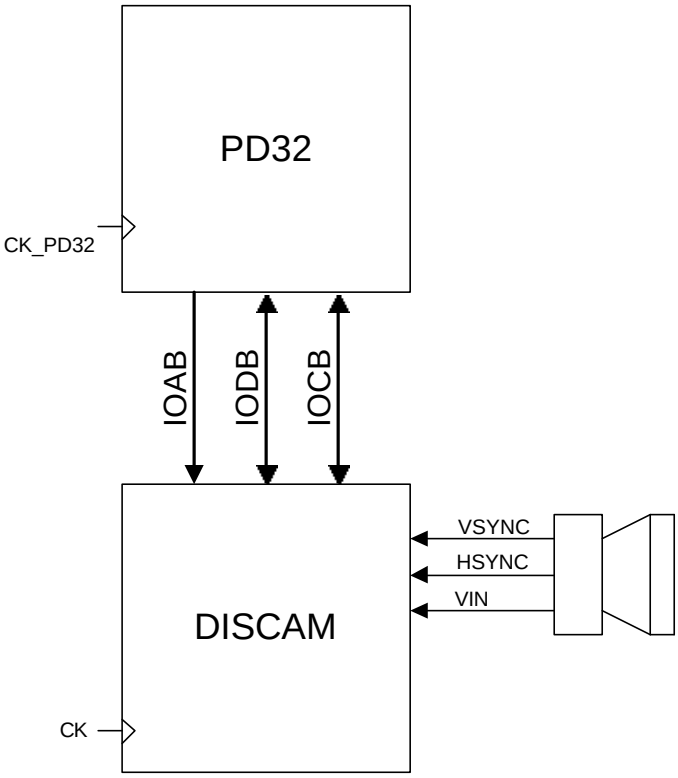
Si vuole progettare un sistema di riconoscimento automatico dei quattro diversi tipi di disco, allo scopo di indirizzarli verso le rispettive linee d'imballaggio. Il sistema di riconoscimento automatico include una videocamera (VC) che inquadra una striscia del nastro trasportatore (sfondo bianco) su cui sono posti in sequenza i singoli dischi (neri) da riconoscere, un'unità di riconoscimento (UR), e un microprocessore PD32.
VC genera una sequenza d'immagini scandite per righe, che sono trasferite ad UR mediante tre segnali:
- Vsync: impulso d'inizio immagine;
- Hsync: impulso d'inizio riga dell'immagine;
- Vin: segnale analogico proporzionale all'intensità luminosa dei punti sulla riga dell'immagine (nero: livello minimo del segnale; bianco: livello massimo del segnale).

UR provvede alla quantizzazione a due soli livelli di Vin mediante un comparatore analogico e si sincronizza su Vsync e Hsync per effettuare la misurazione del diametro del disco rappresentato dal segnale d'immagine.
All'inizio della selezione di un lotto di produzione il PD32 comunica alla periferica UR i tre valori di soglia espressi in numero di campioni sulla riga video (10 bit ciascuno): $(D_0+D_1)/2$, $(D_1+D_2)/2$, $(D_2+D_3)/2$, allo scopo di consentirne il confronto con il valore del diametro rilevato per ciascun disco. Successivamente ad ogni iterazione del ciclo di lavorazione il PD32 provvede a far avanzare il nastro trasportatore in modo da posizionare un disco da riconoscere nel campo di ripresa della videocamera e invia un comando di start alla periferica UR. A riconoscimento avvenuto, UR invia un interrupt al PD32, il quale preleverà il codice associato al disco rilevato e quindi provvederà ad azionare gli attuatori predisposti per deviare il disco sulla linea d'imballaggio appropriata.
Si noti che la misura del diametro diventa significativa quando il segnale Vin digitalizzato presenta una quadrupla variazione: 1-0-1-0-1, cioè quando la riga esplorata dalla videocamera corrisponde ad una sezione del disco col foro.
Si supponga disponibile in ingresso a UR un segnale di clock di frequenza (da non calcolare) adeguata alla misurazione del diametro massimo con la risoluzione di 10 bit.

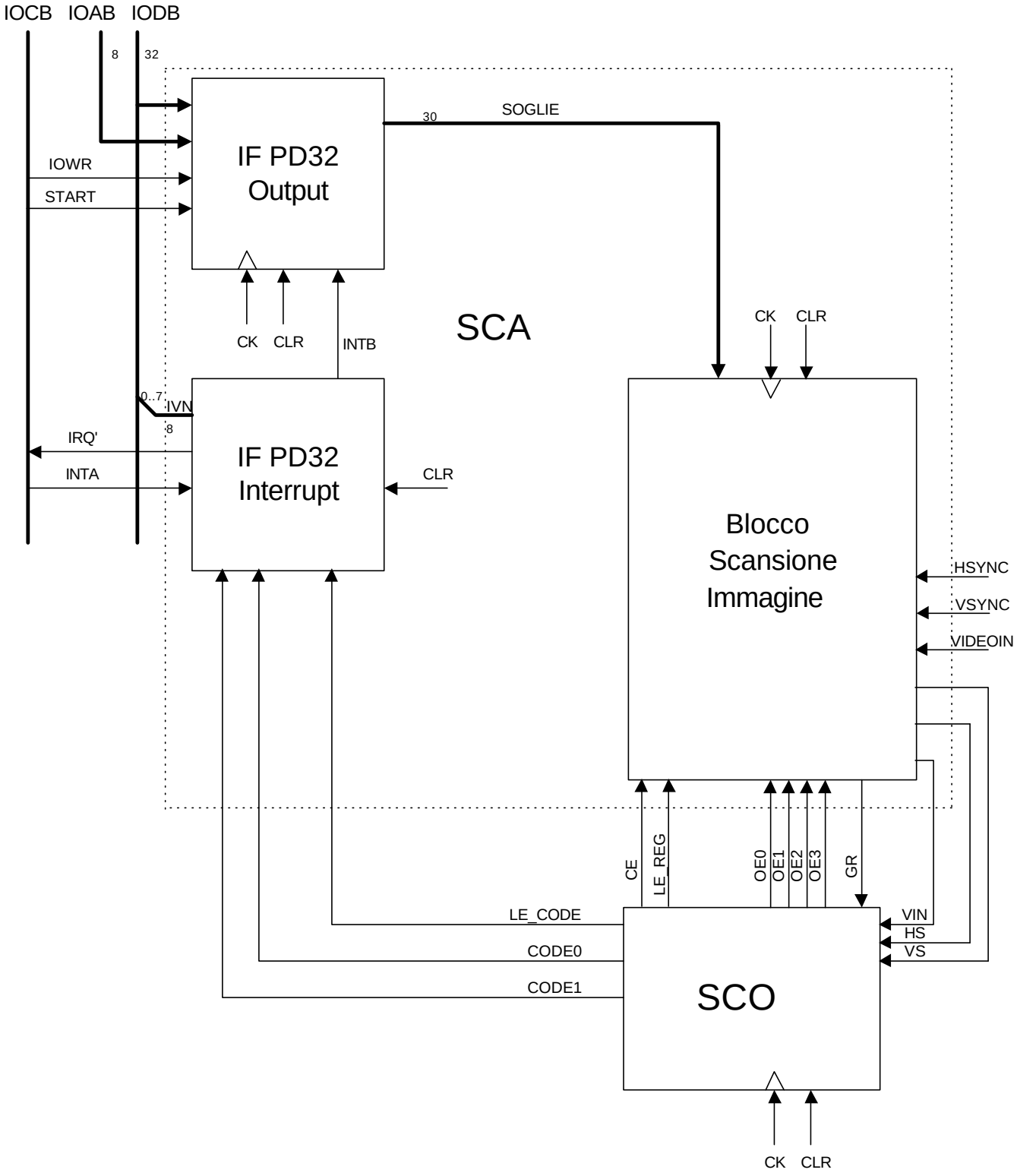
<u>Si richiede:</u>

1. la temporizzazione delle operazioni;
2. lo schema logico della periferica UR ed il microprogramma relativo;
3. le routine d'interfacciamento del PD32.

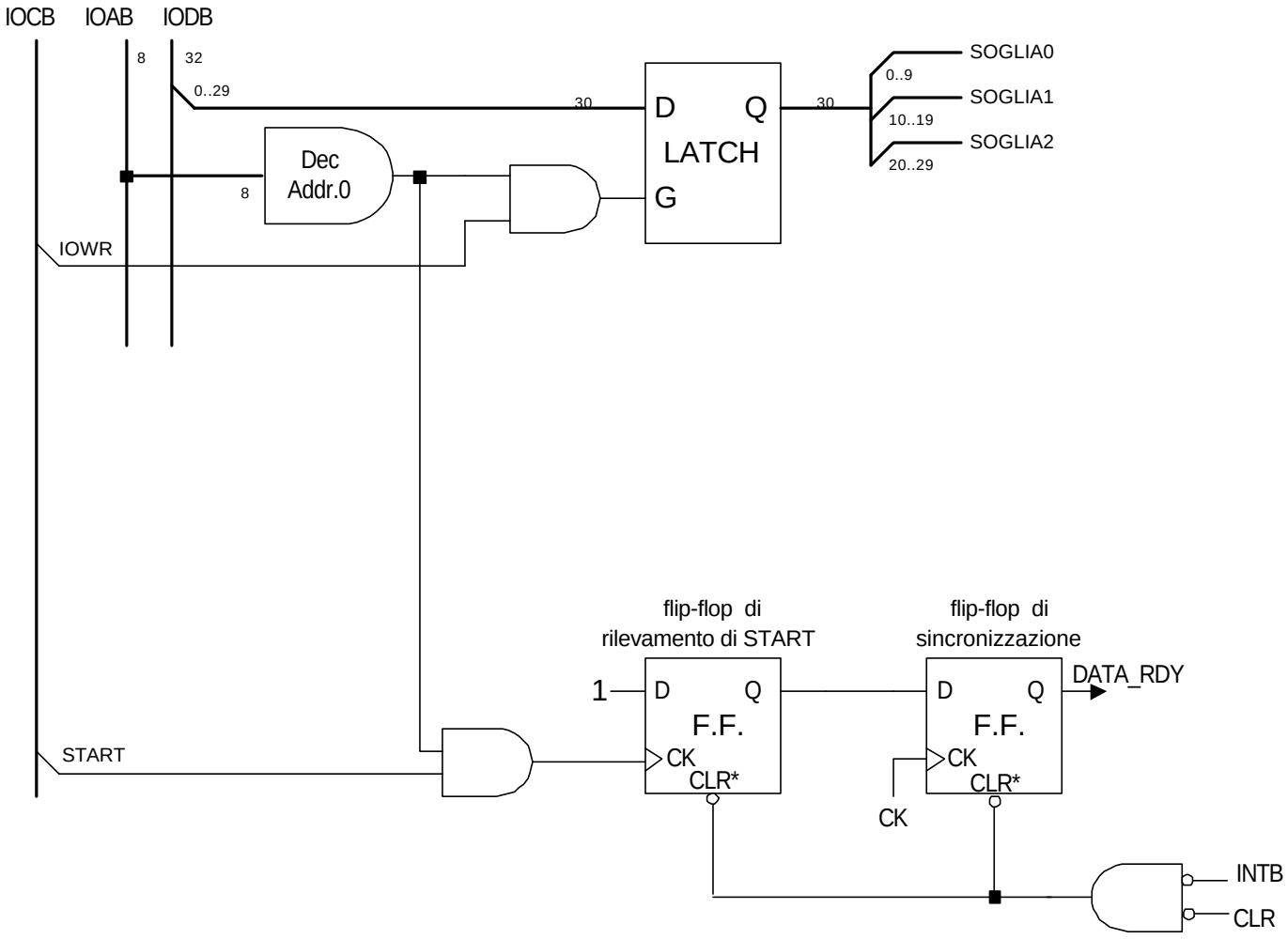
DISCAM: sistema esterno



DISCAM: Schema a blocchi



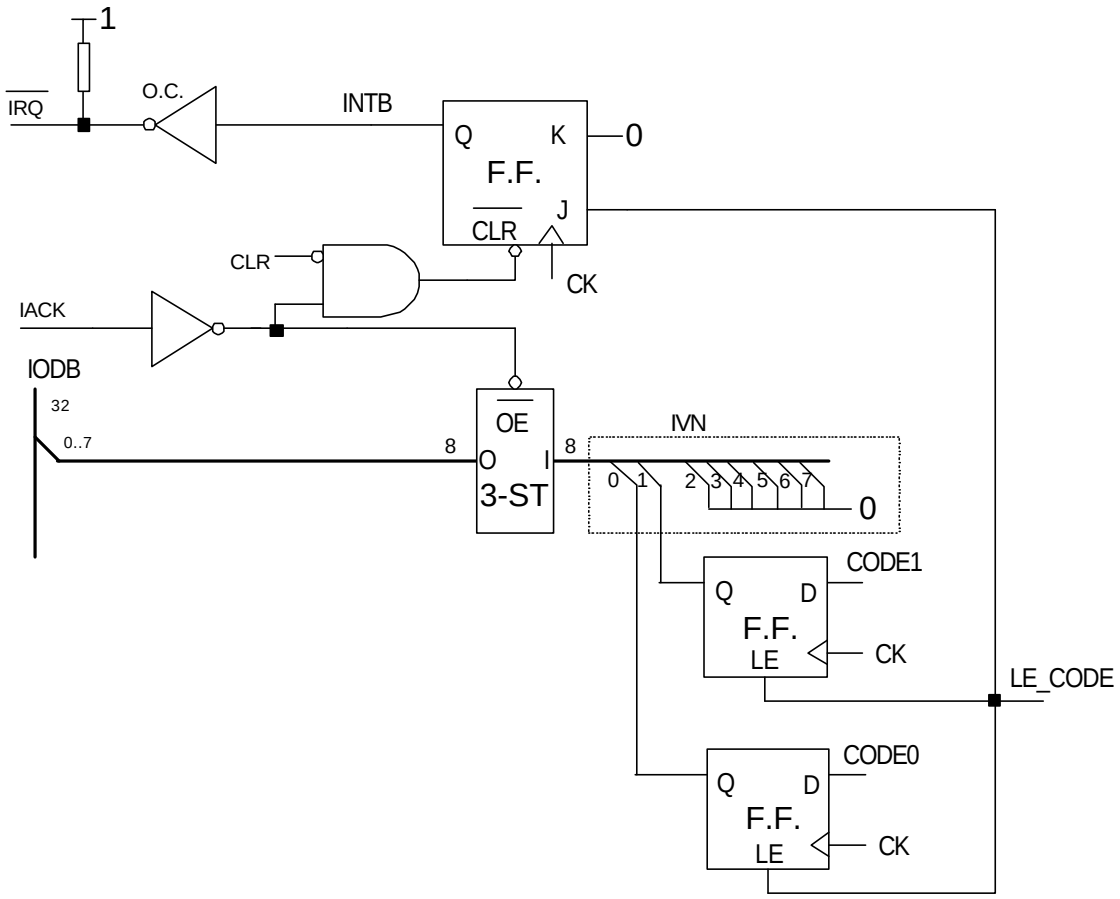
DISCAM: IF PD32 - output



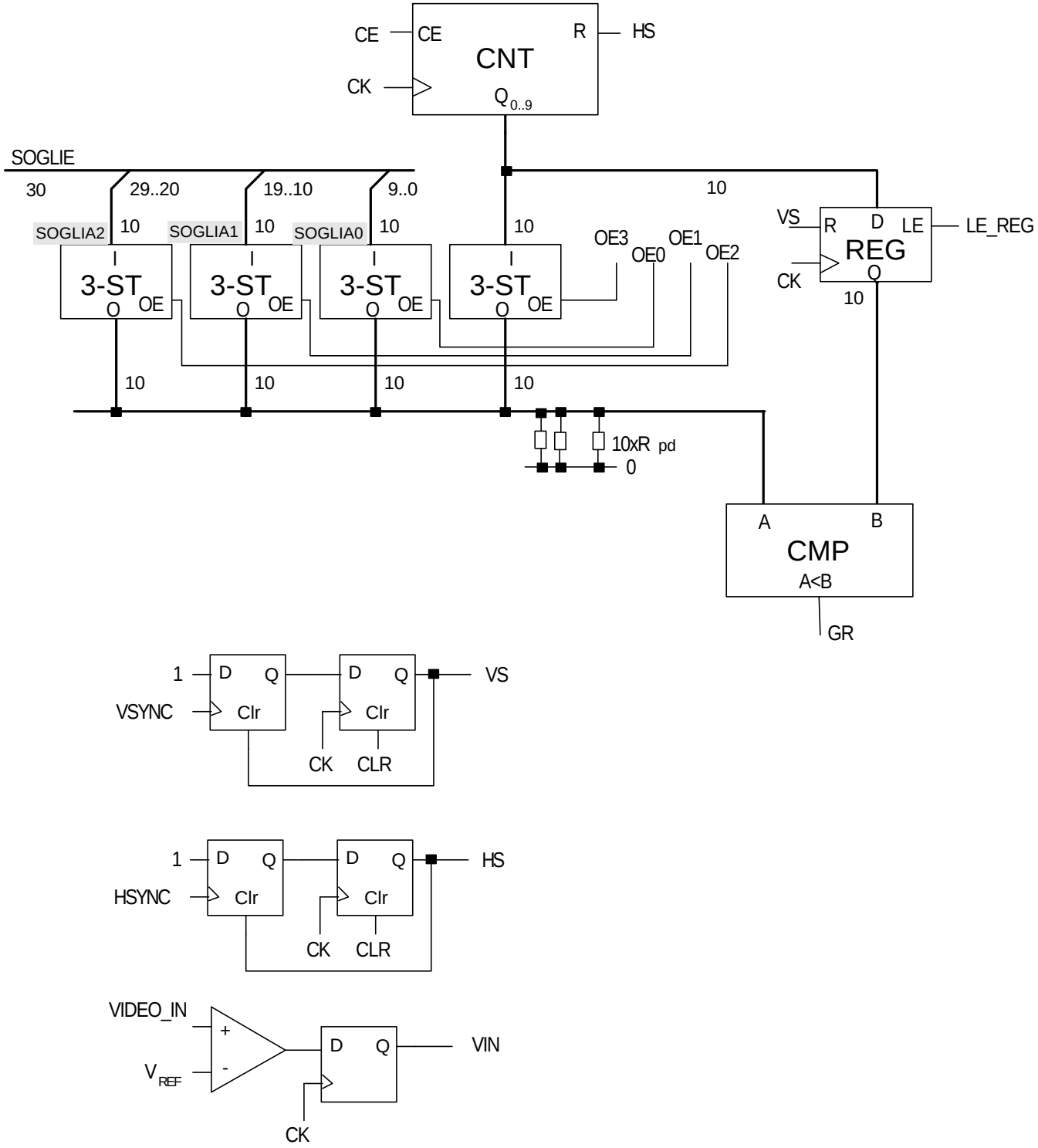
Note

Il SW avvia l'operazione con **START Addr0** dopo avere eventualmente riscritto il registro di interfaccia con i valori delle soglie con l'istruzione **OUTL Soglie,Addr0** (cfr. routine software).

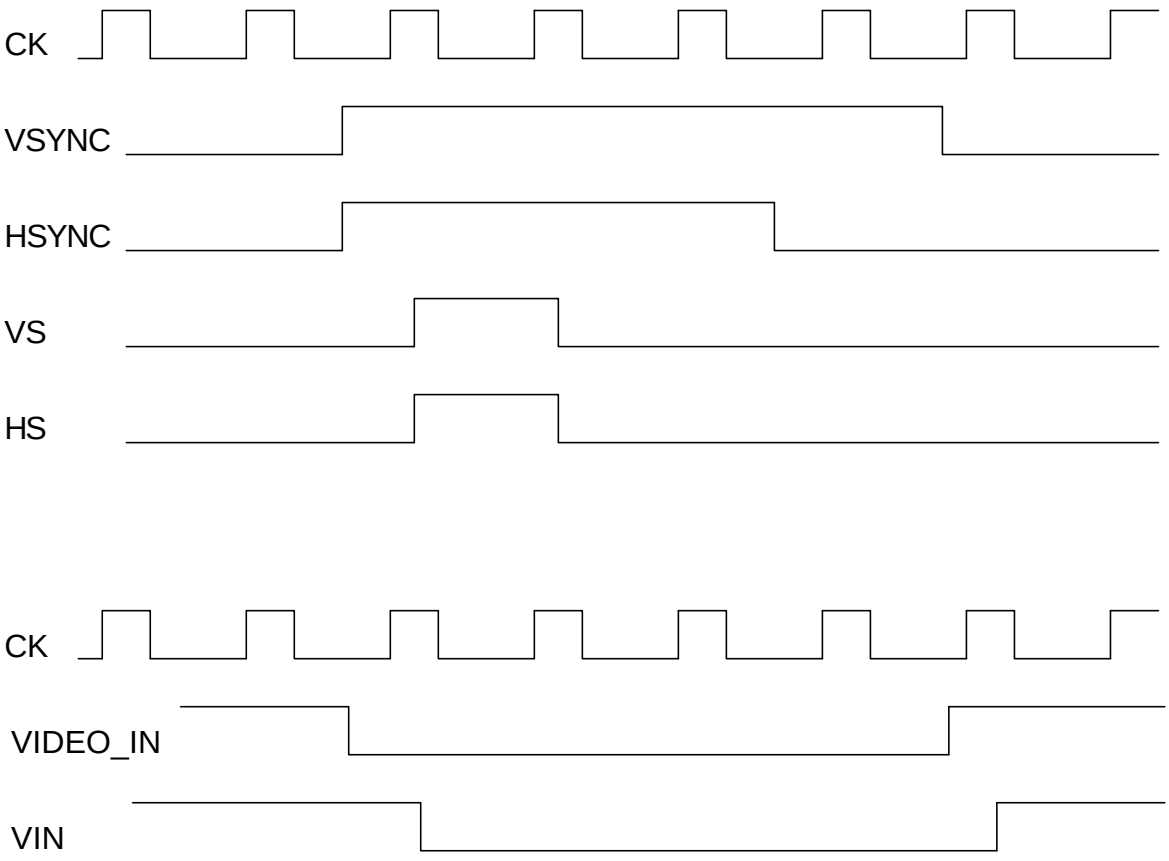
DISCAM: IF PD32 - interrupt



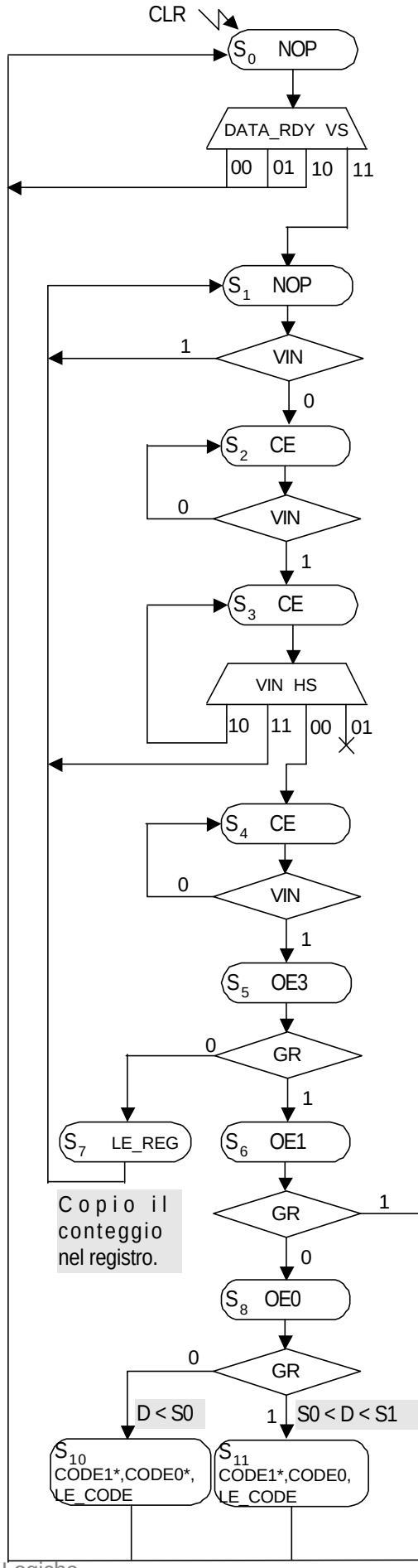
DISCAM: blocco scansione immagine



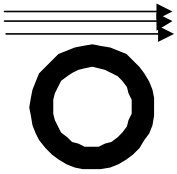
DISCAM: temporizzazioni



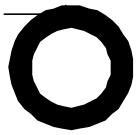
DISCAM: SCO - flowchart



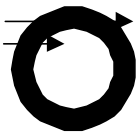
In S_1 lo SCO aspetta l'eventuale commutazione di VIN a 0 (intercettazione della corona circolare - nera) del disco.



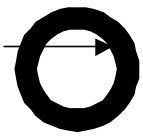
La corona circolare è stata intercettata: in S_2 lo SCO aspetta il ritorno di VIN a 1 (fine della corona circolare e eventuale inizio del foro); il conteggio della lunghezza è abilitato.



E' tornato il bianco: il conteggio continua ad essere abilitato, perché potrebbe trattarsi del foro: in questo caso tornerà il nero prima di HS; nel caso contrario all'arrivo di HS il controllo torna in S_1.

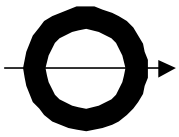


E' tornato il nero: è appena terminata la scansione del foro, e il conteggio della misura della corda continuerà ad essere abilitato fintantoché VIN=0.



Fine dell'esplorazione di una corda che intercetta il foro: per sapere se è il diametro occorre confrontare il conteggio presente con il conteggio precedente, salvato nel registro di appoggio.

Il registro di appoggio contiene la misura del diametro: confronto con la soglia intermedia.

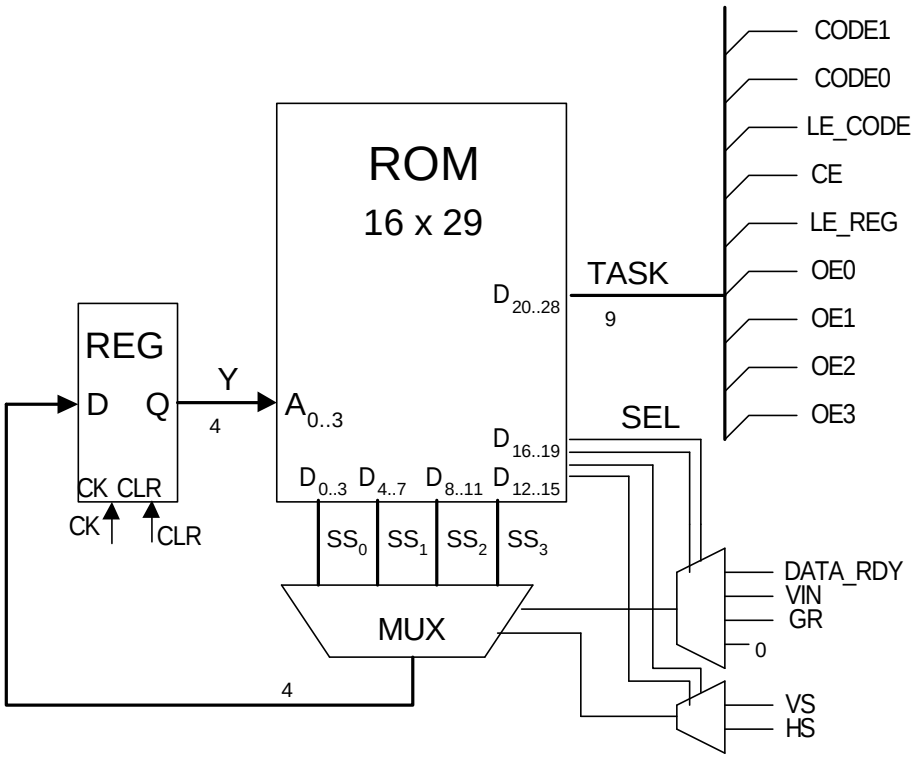


Copio il conteggio nel registro.

Confronto con una delle due altre soglie.

DISCAM: SCO - struttura HW microprogrammata

Si può scegliere il modello strutturale di tipo Moore, in quanto i segnali di TASK sono tutte abilitazioni (allo SCA), che non è necessario registrare.



DISCAM: Routine Software

```
ORG 400h

addr0 EQU 0A5h
addr_disc0 EQU 000800000h     ;ind. iniziale driver disco 0 (lungh. 256 byte)
addr_disc1 EQU 000800100h     ;ind. iniziale driver disco 1 (lungh. 256 byte)
addr_disc2 EQU 000800200h     ;ind. iniziale driver disco 2 (lungh. 256 byte)
addr_disc3 EQU 000800300h     ;ind. iniziale driver disco 3 (lungh. 256 byte)

…

CODE
…
jsr int_discam                ;carica i valori delle tre soglie
…

mainloop:

…
jsr start_discam              ;avvia nuova scansione disco
…

jmp mainloop



; * * * R O U T I N E * * *

init_discam:
outl soglie,addr0             ;caricamento soglie
ret


start_discam:
;attivazione attuatore avanzamento nastro trasportatore
start Addr0                   ;comando di avvio operazioni
ret



; * * * DRIVER * * *

driver 00h,addr_disc0
;attivazione attuatore smistamento disco0
rti

driver 01h,addr_disc1
;attivazione attuatore smistamento disco1
```

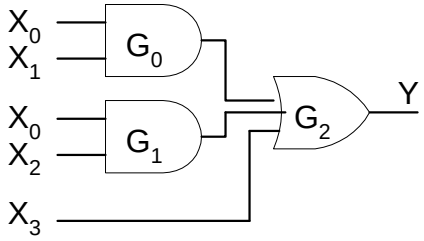
```
rti

driver 02h,addr_disc2
;attivazione attuatore smistamento disco2
rti

driver 03h,addr_disc3
;attivazione attuatore smistamento disco3
rti
```

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 13-10-2000

Studente:_____Docente:_____

D1 La porta G0 ha t_{min}=2 ns, t_{max}=4 ns, la porta G1 ha t_{min}=4 ns, t_{max}=6 ns, la porta G2 ha t_{min}=3 ns, t_{max}=5 ns; descrivere il diagramma di temporizzazione relativo ai nodi intermedi e di uscita della rete di figura a partire dalla variazione X_{3210}: 0010 $\rightarrow$ 1101 del vettore di ingresso.



D2 Dato un flip-flop D, costruire un flip-flop T con ingresso di reset sincrono (dominante).

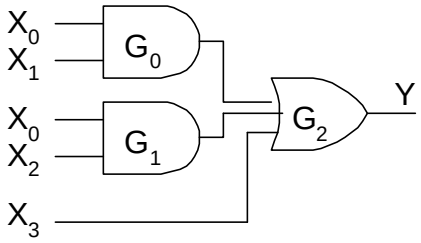
D3 Descrivere la struttura di una tavola di transizione degli stati di una macchina sincrona con due ingressi impulsivi e tre ingressi a livello.

D4 Un sistema SCA-SCO A riceve una sequenza di dati da un sistema B e li passa ad un terzo sistema C, tramite due porte di comunicazione, di ingresso e di uscita rispettivamente. Il sistema A dispone di un unico registro per tamponare ogni singolo dato in transito; A, B, C operano con clock distinti. Descrivere di A: l'hardware delle interfacce verso B e C e il firmware per la sincronizzazione del trasferimento dei dati.

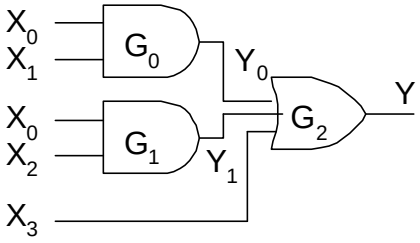
D5 Una periferica produce dati alla velocità di R Byte/s, che deve trasferire nella memoria del PD32 mediante accesso in DMA: determinare il valore di R al di sopra del quale è necessario utilizzare la tecnica a burst (e sotto il quale è conveniente utilizzare la tecnica a bus stealing), supponendo che i cicli macchina del processore siano di 100 ns.

Esercizio (2S20001013-D1)

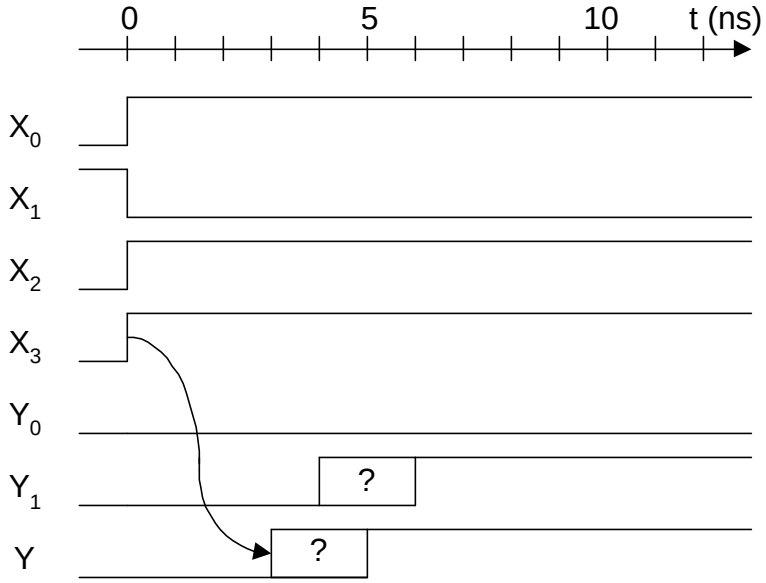
La porta G0 ha t_{min}=2 ns, t_{max}=4 ns, la porta G1 ha t_{min}=4 ns, t_{max}=6 ns, la porta G2 ha t_{min}=3 ns, t_{max}=5 ns; descrivere il diagramma di temporizzazione relativo ai nodi intermedi e di uscita della rete di figura a partire dalla variazione X_{3210}: 0010 $\rightarrow$ 1101 del vettore di ingresso.



Detti Y0 e Y1 i due nodi intermedi della figura:



il diagramma di temporizzazione richiesto può essere tracciato come di seguito:

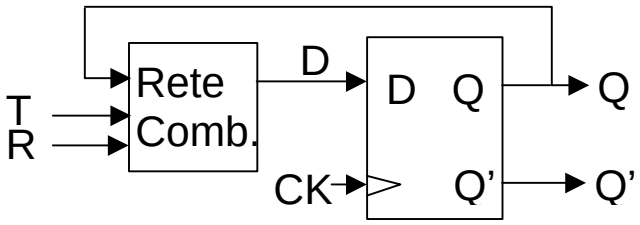


Si noti che Y è forzato a 1 da X_3 (a 1), indipendentemente dalle variazioni transitorie (imprevedibili) su Y_1, protratte dal modello oltre la stabilizzazione di Y.

Esercizio (2S20001013-D2)

Dato un flip-flop D, costruire un flip-flop T con ingresso di reset sincrono (dominante).

Il flip-flop può essere considerato una rete sequenziale sincrona, che risponde al noto modello strutturale, che di seguito viene personalizzato al caso del flip-flop: il registro di stato è composto da un solo flip-flop D, esattamente quello che il testo del problema raccomanda di utilizzare.



E' appena il caso di notare che in questa struttura, come in tutte quelle che rispondono allo stesso modello strutturale di rete sincrona, il clock è un segnale non soggetto a elaborazioni, di nessun tipo!

Il progetto è stato così ricondotto al calcolo della rete combinatoria. D è lo stato successivo del flip-flop (registro di stato della rete).
la tavola di verità è:

Q	T	R	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Minimizzazione mediante MK:

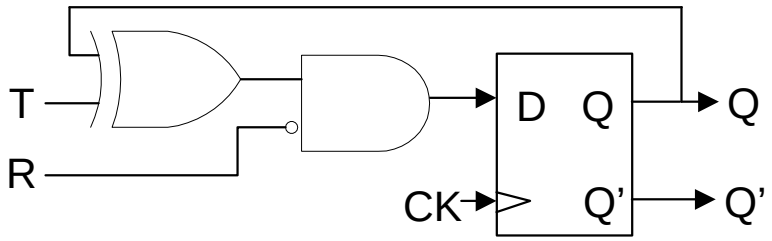
Q \ T R	00	01	11	10
0				1
1	1			

Si ottiene:

$$D = \overline{R} \cdot T \cdot \overline{Q} + \overline{R} \cdot \overline{T} \cdot Q =$$

$$= \overline{R} \cdot (T \oplus Q)$$

Ne consegue lo schema del flip-flop richiesto:



Esercizio (2S20001013-D3)

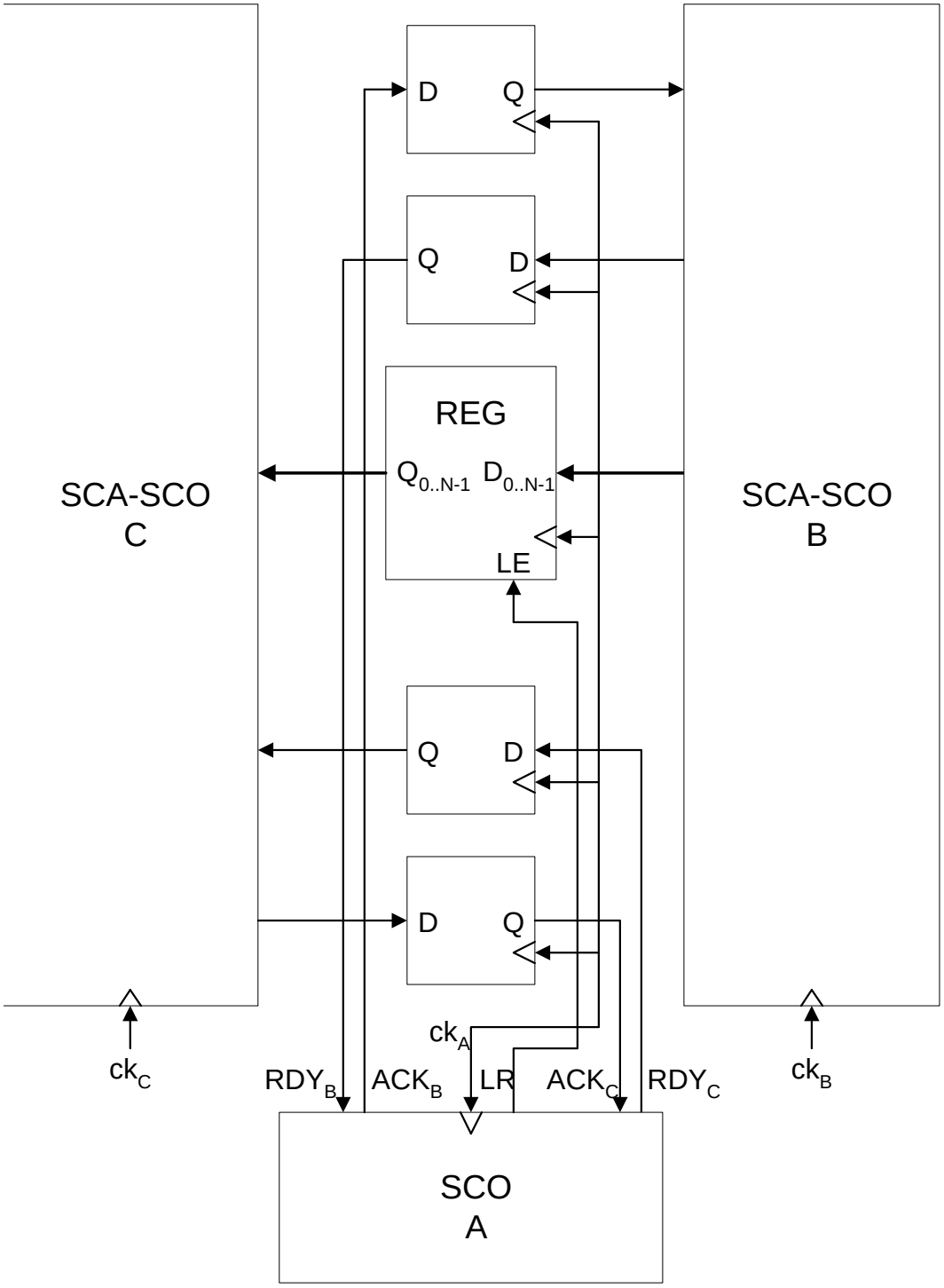
Descrivere la struttura di una tavola di transizione degli stati di una macchina sincrona con due ingressi impulsivi e tre ingressi a livello.

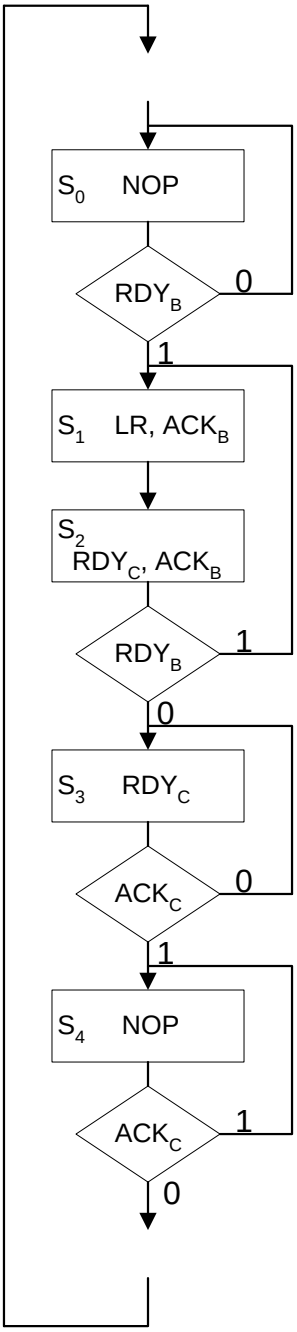
Detti P_0 e P_1 i due segnali impulsivi, $X_0\,X_1\,X_2$ i tre ingresso a livello, la tavola di transizione (sincrona) degli stati è del tipo seguente:

	$P_0\uparrow$								$P_1\uparrow$							
S	X_0 X_1 X_2								X_0 X_1 X_2							
	000	001	011	010	110	111	101	100	000	001	011	010	110	111	101	100
S_0																
S_1																
S_2																
S_N																

Esercizio (2S20001013-D4)

Un sistema SCA-SCO A riceve una sequenza di dati da un sistema B e li passa ad un terzo sistema C, tramite due porte di comunicazione, di ingresso e di uscita rispettivamente. Il sistema A dispone di un unico registro per tamponare ogni singolo dato in transito; A, B, C operano con clock distinti. Descrivere di A: l'hardware delle interfacce verso B e C e il firmware per la sincronizzazione del trasferimento dei dati.





Tutti i flip-flop sono pilotati sull'ingresso CLR asincrono da un segnale di inizializzazione
(non indicato per semplicità di rappresentazione)

Lo stato S_2 anticipa RDY_C (dato caricato da B = = dato pronto per C) mentre aspetta RDY_B=0

Esercizio (2S20001013-D5)

Una periferica produce dati alla velocità di R Byte/s, che deve trasferire nella memoria del PD32 mediante accesso in DMA: determinare il valore di R al di sopra del quale è necessario utilizzare la tecnica a burst (e sotto il quale è conveniente utilizzare la tecnica a bus stealing), supponendo che i cicli macchina del processore siano di 100 ns.

---

1. Ipotesi di accesso a “bus stealing”

Il processore impiega un tempo massimo pari a 100ns (un ciclo macchina) per cedere l’uso dei bus di sistema. La periferica impiega altri 100 ns (tecnica del bus stealing) nel ciclo di scrittura della memoria. Pertanto la periferica riesce a trasferire in memoria i dati con singoli accessi in DMA con un periodo temporale di almeno 200 ns, cioè con un ritmo massimo pari a 1/200 ns = 5 Mdati/s. Al di sopra di questa velocità è necessario utilizzare l’accesso a burst.
Si può specificare ancora più in dettaglio la velocità di produzione se si distinguono i casi di accesso in RAM a byte, word e longword:
1 - se la periferica scarica byte in memoria, allora la velocità massima in stealing è di 5 Mbyte/s;
2 - se la periferica scarica word in memoria, allora la velocità massima in stealing è di 5 Mword/s = 10 Mbyte/s;
3 - se la periferica scarica longword in memoria, allora la velocità massima in stealing è di 5 Mlongword/s = 20 Mbyte/s.
Si vede che l’incremento della velocità di produzione deve essere supportato da un incremento della larghezza del registro dei dati sul MDB.

2. Ipotesi di accesso a “ burst”

Evidentemente l’accesso a burst può essere applicato anche al di sotto della velocità di 5 Mdati/s; in questo caso il processore risulta di fatto bloccato (non avverte neanche la richiesta di interruzione) per l’intero trasferimento.

La casistica può essere riassunta nel prospetto seguente:

DMA	R < 5 Mdati/s	5 Mdati/s < R
STEALING	preferibile	impossibile
BURST	possibile	necessario

RETI LOGICHE

PRIMA PROVA SCRITTA DEL 12-01-2001

STUDENTE:_____DOCENTE:_____

Si vuole realizzare un dispositivo per pilotare un display alfanumerico costituito da 2 linee x 16 caratteri a matrice di punti con organizzazione 5 colonne x 8 righe (fig. 1). Il progetto dell'unità (DISP_CONT) prevede anche un'interfaccia con un processore PD32 che produce i messaggi di 32 caratteri ASCII da stampare sul display.

1	2	/	0	1	/	2	0	0	1		0	9	:	0	0
*	B	U	O	N	A		G	I	O	R	N	A	T	A	*

Fig. 1 - Formato del display.

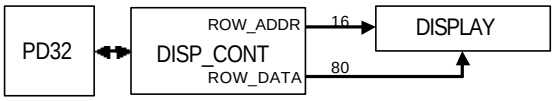


Fig. 2 – Interfacciamento del display.

DISP_CONT invia al modulo display (16 righe x 80 colonne) i seguenti segnali (fig. 2):
- un codice lineare (ROW_ADDR) di 16 bit che indirizza la riga del display che si vuole abilitare alla scrittura;
- il vettore (ROW_DATA) degli 80 bit (= 5 punti x 16 caratteri) che si vogliono scrivere sulla riga indirizzata da ROW_ADDR.

Il display deve essere scritto per righe ciclicamente in modo da rinfrescare il messaggio 100 volte al secondo.

L'unità DISP_CONT include:
- un singolo modulo RAM di 32 byte in cui il processore PD32 scrive il messaggio che la periferica deve leggere e quindi stampare sul display; a questo scopo il modulo RAM locale viene visto dal processore all'indirizzo iniziale FFFF0000h;
- un modulo ROM con funzione di generatore di caratteri, contenente una tabella di (256 x 8) parole di 5 bit che rappresentano le mappe grafiche dei caratteri ASCII; a scopo esemplificativo nella figura 3 è riportato il frammento della tabella nella ROM relativo ai due caratteri 'R' e 'S'.

Prima di scrivere un nuovo messaggio (32 caratteri ASCII) nella RAM locale, il micro invia un comando di fine scansione alla periferica, che termina il ciclo di lettura corrente e quindi rilascia il controllo della RAM; mentre il micro aggiorna la RAM locale la periferica cessa di indirizzare le righe del display (tutti i punti grafici si disattivano); al termine della scrittura del messaggio nella RAM il micro invia un comando di inizio scansione alla periferica, che comincia a leggere il nuovo messaggio dalla RAM locale e a scriverlo nel display.

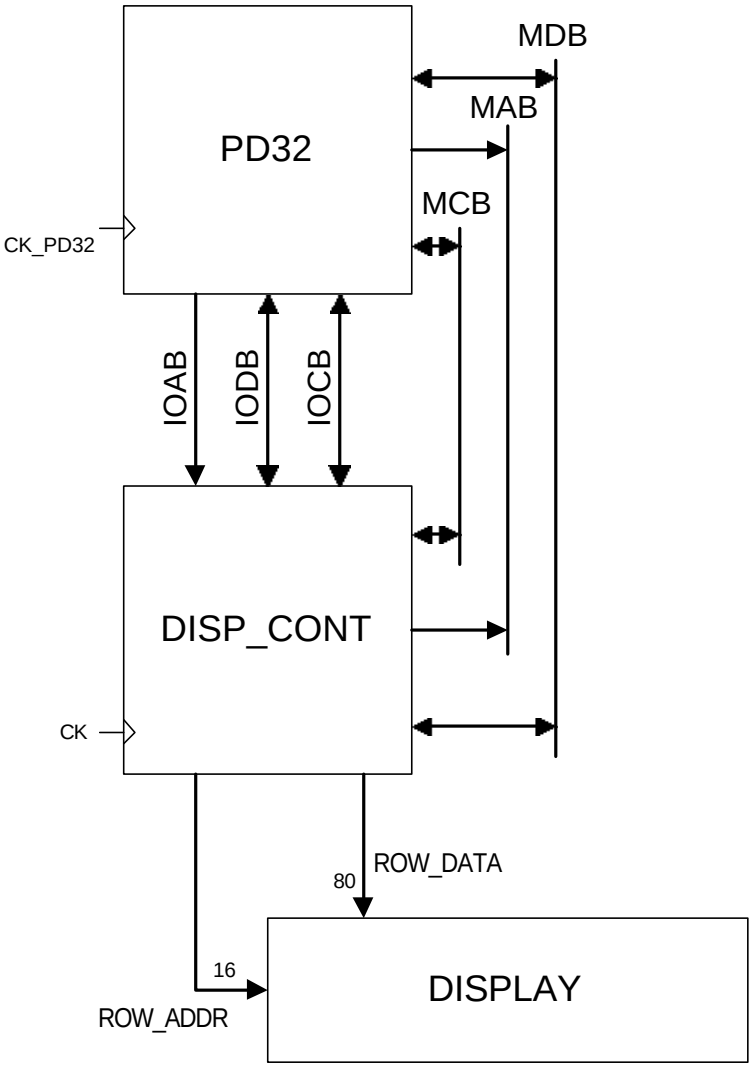
1	1	1	1	0	Inizio carattere 'R'
1	0	0	0	1	
1	0	0	0	1	
1	1	1	1	0	
1	0	1	0	0	
1	0	0	1	0	
1	0	0	0	1	
0	0	0	0	0	
0	1	1	1	0	Inizio carattere 'S'
1	0	0	0	1	
1	0	0	0	0	
0	1	1	1	0	
0	0	0	0	1	
1	0	0	0	1	
0	1	1	1	0	
0	0	0	0	0	

Fig. 3 - Frammento del contenuto della ROM.

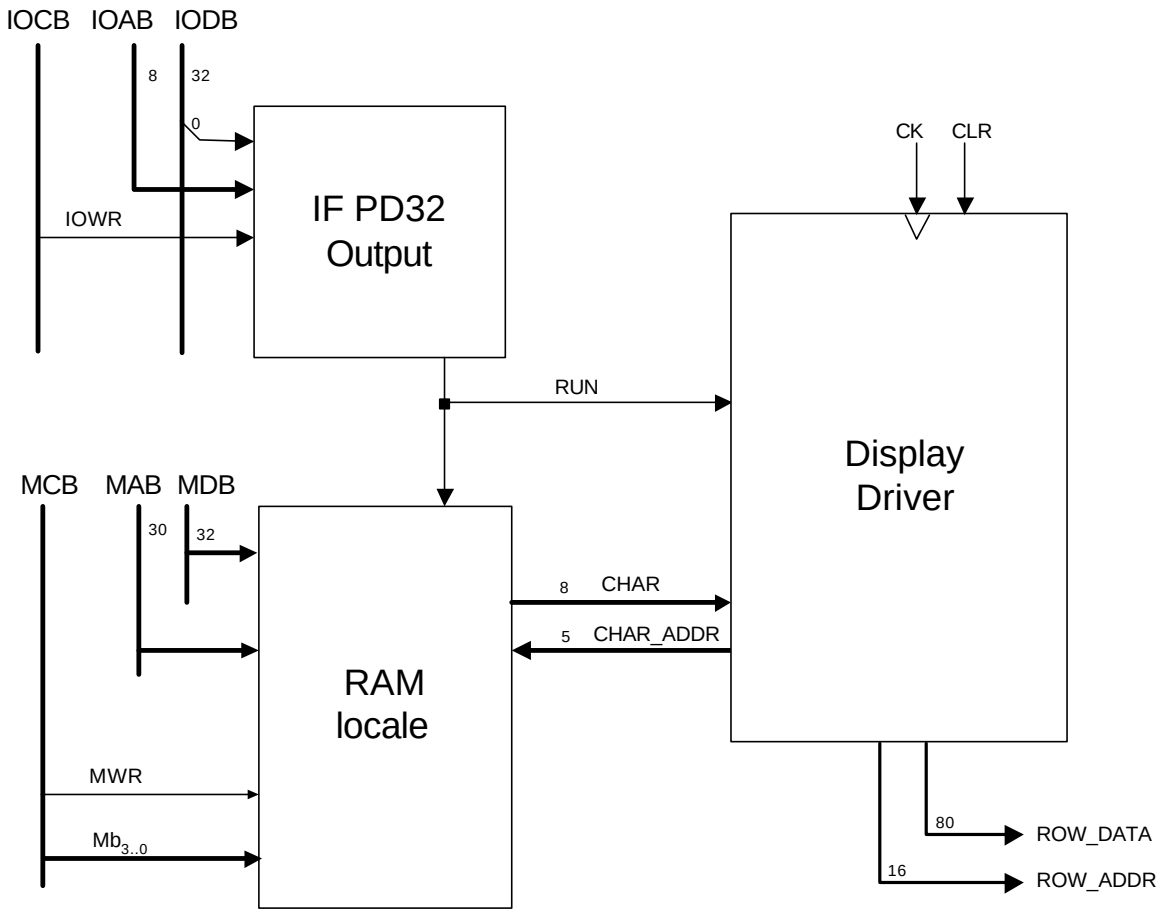
Si richiede:

1. la temporizzazione delle operazioni;
2. lo schema logico della periferica DISP_CONT;
3. la frequenza del clock della periferica DISP_CONT;
4. le routine d'interfacciamento del PD32.

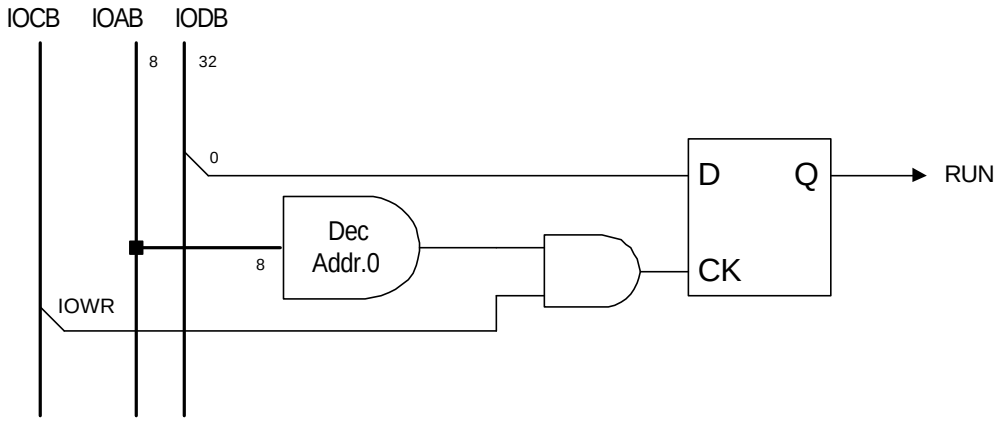
DISPCONT: sistema esterno



DISPCONT: Schema a blocchi



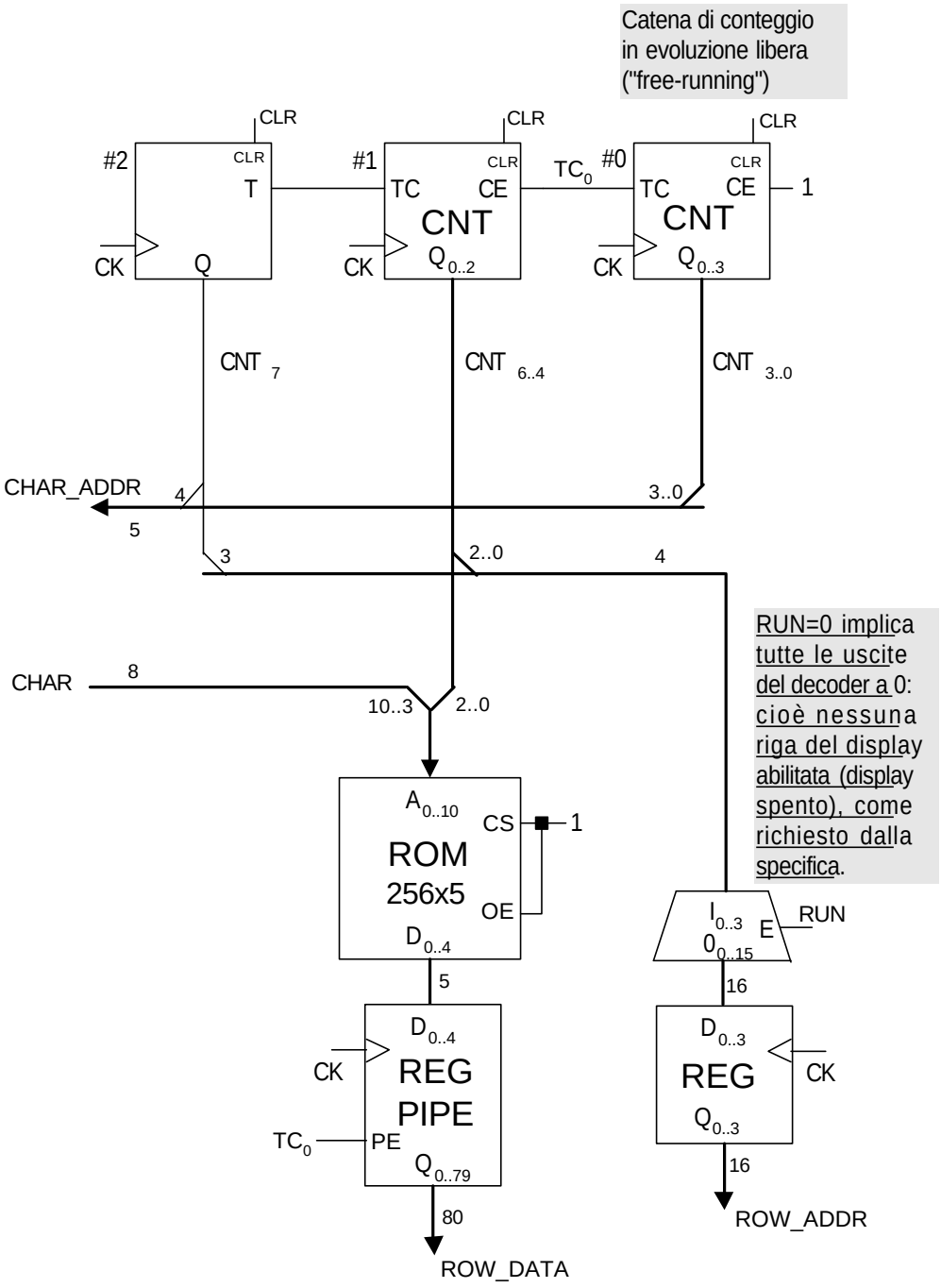
DISPCONT: IF PD32 - output



Note

Il SW abilita la periferica a leggere il display con l'istruzione:
OUTB #1,Addr0
e la disabilita per poter aggiornare la RAM con l'istruzione:
OUTB #0,Addr0

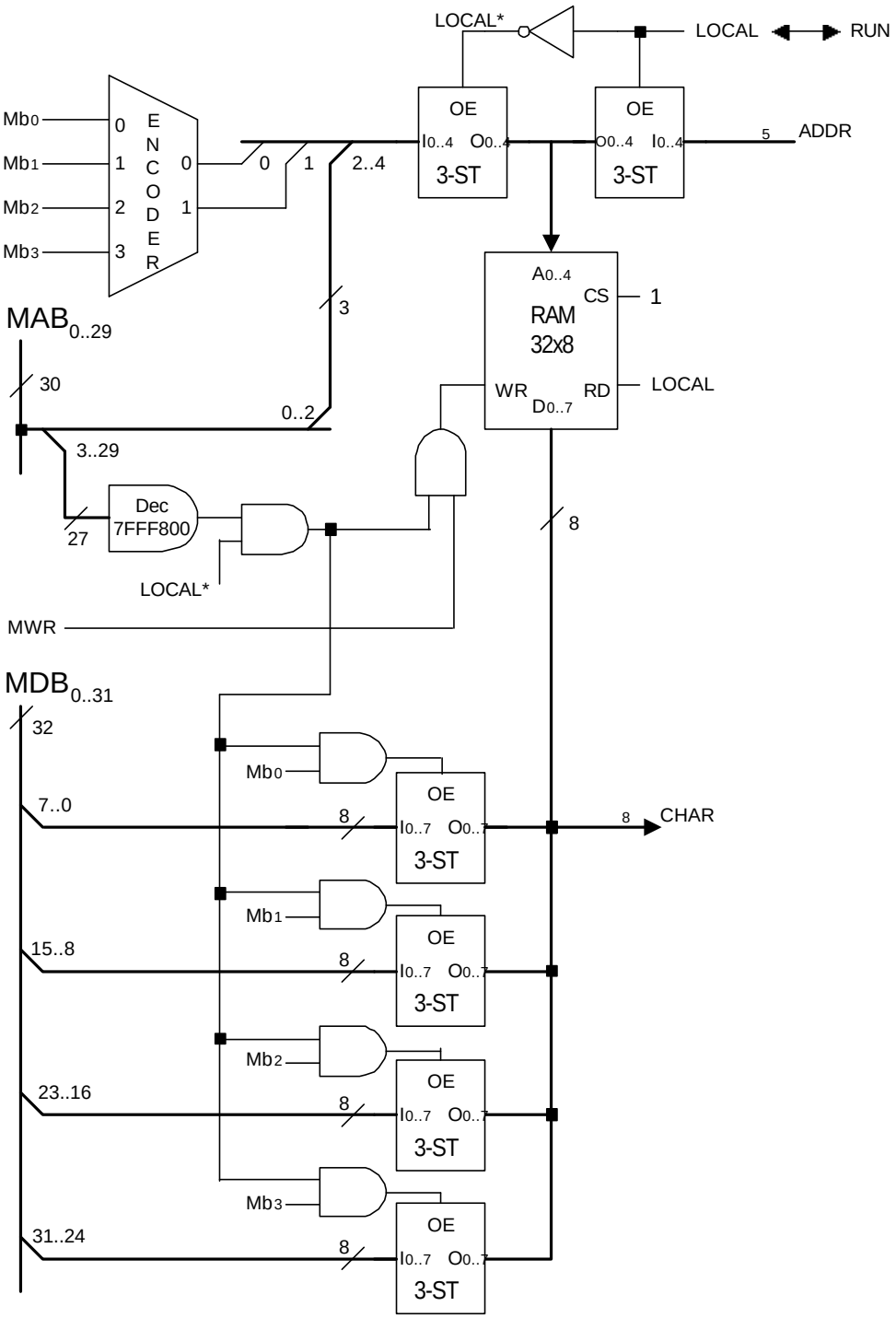
DISPCONT: display driver



Contatori #0,#2 (4+1 bit): producono i 32 indirizzi della RAM, per scandire i 16 + 16 caratteri nel messaggio.
Contatori #1,#2 (3+1 bit): contano le 16 (=8x2) righe grafiche delle due linee di caratteri del display.

DISPCONT: RAM LOCALE

- 8 bit
- PERIFERICA: RD
- PD32: WR; ind. iniziale: FFFF0000h

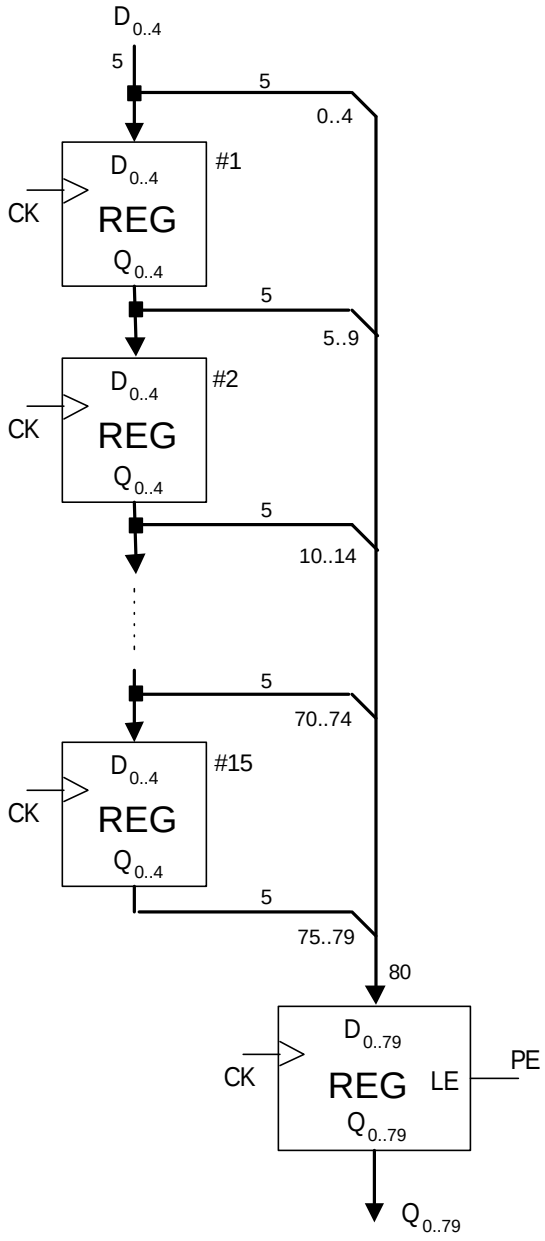


- LOCAL viene attivato (disattivato) all'inizio (al termine) dell'elaborazione richiesta dal micro.

- Il PD32 scrive nella RAM locale a byte (MOV**B** data,RAM_addr).

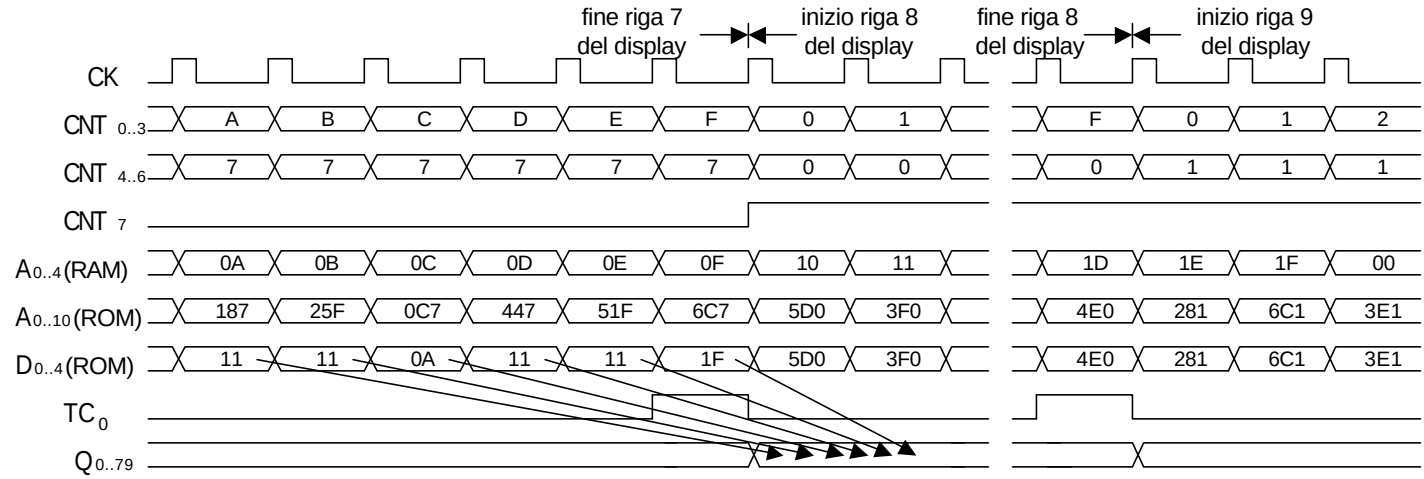
- Indirizzi PD32 a 32 bit della RAM locale: FFFF0000h.. FFFF001Fh; MAB (30 bit): 3FFFC000h..3FFFC007h; parte costante: 7FFF800h.

DISPCONT: REG pipe



DISPCONT: temporizzazioni

Accessi alla RAM / ROM



DISPCONT: Routine Software

```
ORG 400h

Addr0 EQU 0A5h

…

CODE
…

mainloop:
...
jsr new_message

jmp mainloop


; * * * R O U T I N E * * *

new_message:
outb #0,Addr0           ;disabilita DISPCONT a leggere la RAM
jsr write_message       ;aggiorna RAM locale di DISPCONT
outb #1,Addr0           ;abilita DISPCONT a leggere la RAM
ret
```

NOTE

- Calcolo della frequenza di CK:
  Al display deve essere presentata 100 volte al secondo una sequenza di 16 vettori di 16 x 5 configurazioni grafiche; per prelevare queste ultime in un secondo devono essere effettuati 100 x 16 x 16 accessi in lettura dalla ROM. Immaginando di effettuare un accesso in un singolo periodo di CK, la frequenza di CK vale:
  F_{CK} = 16 x 16 x 100 x Hz = 25.6 KHz.
  Tale frequenza è sufficientemente ridotta da poter effettuare in un singolo ciclo di CK un accesso alla ROM, e anche un accesso alla RAM in cascata.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 12-01-2001

STUDENTE:_____DOCENTE:_____

D1 Da un modulo RAM vengono letti dati a larghezza 9 bit, di cui uno occasionalmente errato. Determinare a quanti bit si riducono i dati utili (codice irridondante) nei due casi di applicazione di una codifica a rivelazione e a correzione dell'errore rispettivamente.

D2 Progettare una rete combinatoria iterativa per la complementazione a 2 di un numero a N bit.

D3 Progettare un contatore mod 6 dotato di ingresso di abilitazione al conteggio.

D4 Illustrare la temporizzazione relativa ad un sistema SCO-SCA di tipo D-Mealy-Mealy.

D5 Una porzione di memoria RAM del PD32 di capacità pari a 256 byte è riservata all'indirizzo iniziale FIFO_BASE per essere gestita dal processore come un buffer circolare di tipo FIFO. All'indirizzo FIFO_STATUS va registrato lo stato - vuoto, pieno - del buffer FIFO. Scrivere le due routine assembler per operare sul buffer l'inserimento e l'estrazione di un singolo byte.

Esercizio (2S20010112-D1)

Da un modulo RAM vengono letti dati a larghezza 9 bit, di cui uno occasionalmente errato. Determinare a quanti bit si riducono i dati utili (codice irridondante) nei due casi di applicazione di una codifica a rivelazione e a correzione dell'errore rispettivamente.

---

1. Codifica a semplice rivelazione di errore

Ponendo R=1 nella relazione R=h-1 si ottiene h=2. La distanza di Hamming di un codice irridondante (h=1) può essere elevata a 2 semplicemente introducendo un singolo bit di controllo con funzione di parità; pertanto è k=1. Dovendo poi essere per il codice ridondante n+k=9 (dato del problema) si deduce che n=8.

2. Codifica a correzione di errore:

Ponendo C=1 nella relazione $2C \leq h-1$ si ottiene il valore minimo h=3, che indica l'uso di un codice di Hamming a distanza 3. La relativa relazione vincolare:

$2^k-k-1 \geq n$

va accoppiata al vincolo posto dal problema:

$n+k=9$ da cui:

$2^k-k-1 \geq 9-k$ da cui:

$2^k \geq 10$ da cui:

$k=4$ e perciò:

$n=5$.

Esercizio (2S20010112-D2)

Progettare una rete combinatoria iterativa per la complementazione a 2 di un numero a N bit.

Cfr. Appunti Integrativi, parte 3 (“Reti combinatorie complesse”), dove sono presentate due implementazioni alternative dalla stessa complessità circuitale.

Esercizio (2S20010112-D3)

Progettare un contatore mod 6 dotato di ingresso di abilitazione al conteggio.

Il contatore viene progettato come rete sincrona, il cui comportamento (contatore ad incremento) è descritto dalla tabella degli stati seguente, comprensiva delle variabili di eccitazione dei 3 ($6<2^3$) flip-flop T, scelti per l'implementazione. La macchina sincrona è dotata dell'unico segnale a livello CE (Count Enable); conviene codificare subito i 6 stati.

funzione	CE	Y2	Y1	Y0	Y2'	Y1'	Y0'	T2	T1	T0
conteggio bloccato	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	1	0	0	0
	0	0	1	0	0	1	0	0	0	0
	0	0	1	1	0	1	1	0	0	0
	0	1	0	0	1	0	0	0	0	0
	0	1	0	1	1	0	1	0	0	0
	0	1	1	0	-	-	-	-	-	-
	0	1	1	1	-	-	-	-	-	-
conteggio abilitato	1	0	0	0	0	0	1	0	0	1
	1	0	0	1	0	1	0	0	1	1
	1	0	1	0	0	1	1	0	0	1
	1	0	1	1	1	0	0	1	1	1
	1	1	0	0	1	0	1	0	0	1
	1	1	0	1	0	0	0	1	0	1
	1	1	1	0	-	-	-	-	-	-
	1	1	1	1	-	-	-	-	-	-

A destra sono riportate le due funzioni T2, T1 e T0 corrispondenti alle variazioni tra le coppie Y2,Y2' e Y1,Y1' e Y0,Y0' rispettivamente; cioè: T2=Y2⊕Y2' e T1=Y1⊕Y1' e T0=Y0⊕Y0'. In questo modo il calcolo viene impostato per predisporre una implementazione del registro di stato del contatore con flip-flop di tipo T.
Ovviamente soltanto 6 stati sono possibili, pertanto i 2 (= 8-6) stati impossibili hanno successori non specificati (dcc) nella tavola.
La sintesi completa richiede la minimizzazione di 3 MK (le 3 variabili di stato) a 4 variabili (le 3 variabili di stato + 1 di ingresso):

Y1Y0 / CEY2	00	01	11	10
00				
01			-	-
11	1	1	-	-
10	1	1	1	1

T 0 = CE

Il risultato T0 = CE era atteso, in quanto il bit 0 del conteggio commuta in ogni stato se CE=1, anche nel passaggio da 5 (101) a 0 (000), esattamente come in un contatore mod 8. Inoltre, un contatore mod 8 commuta da 5 (101) a 6 (110), il che anticipa una diversità delle due equazioni che calcolano T1 e T2 nel contatore mod 6 rispetto a quelle standard (T1=CE YO, T2=CE Y0 Y1) del contatore mod 8.

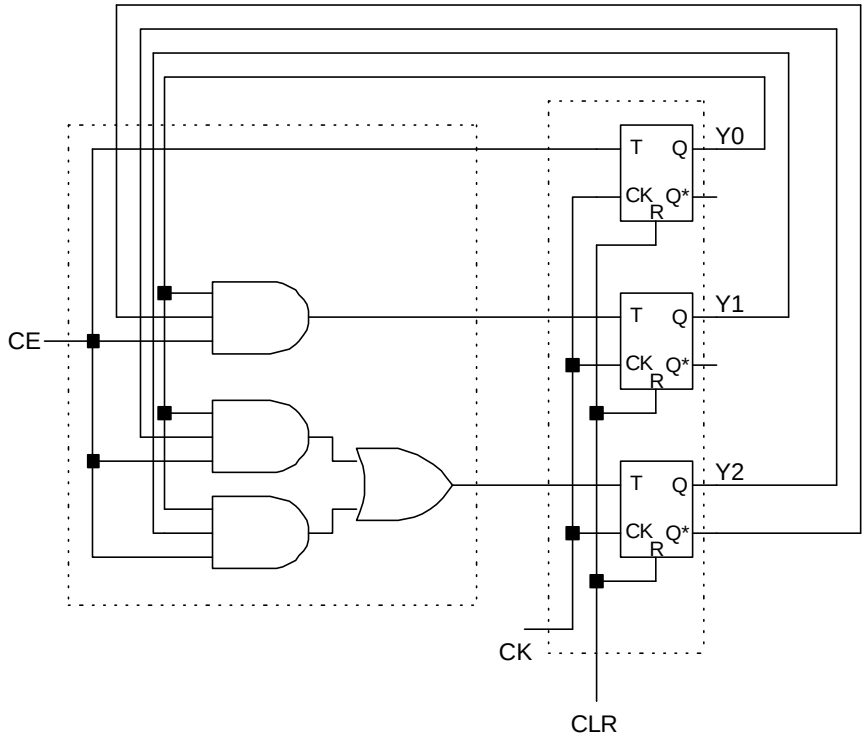
Y1Y0 / CEY2	00	01	11	10
00				
01			-	-
11			-	-
10		1	1	

T 1 = CE Y2* Y0

Y1Y0 / CEY2	00	01	11	10
00				
01			-	-
11		1	-	-
10			1	

T 2 = CE Y2 Y0 + CE Y1 Y0

La rete sequenziale è costruita di conseguenza:



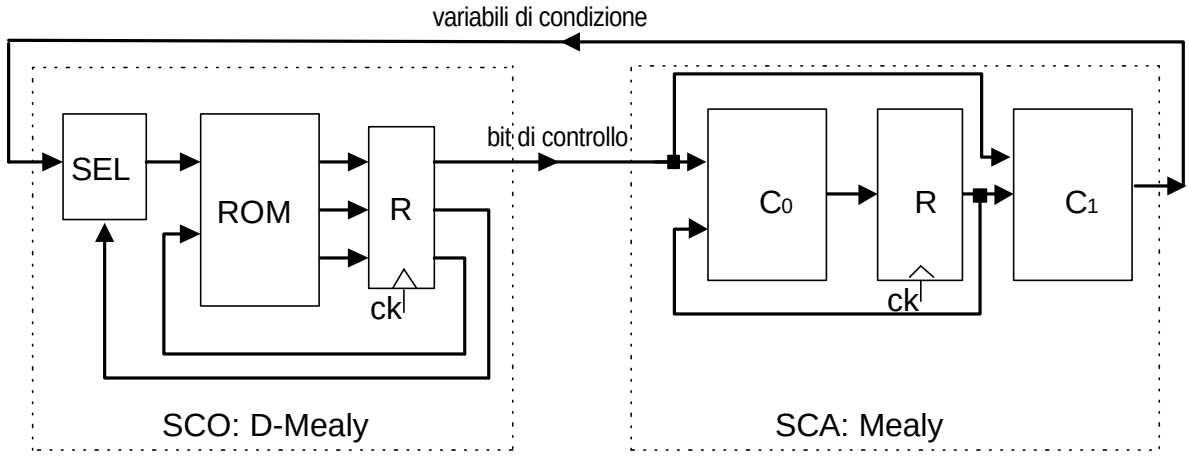
Si noti il collegamento della linea CLR sugli ingressi R (reset asincrono) dei flip-flop per inizializzare il contatore a 00 indipendentemente da CK.

Nella figura sono evidenziati il registro di stato del contatore, costituito dai tre flip-flop T, e la rete combinatoria (ingressi: CE dall'esterno e il vettore di stato retroazionato) che calcola lo stato successivo, come previsto dal modello strutturale generale delle reti sequenziali sincrone di tipo Moore.

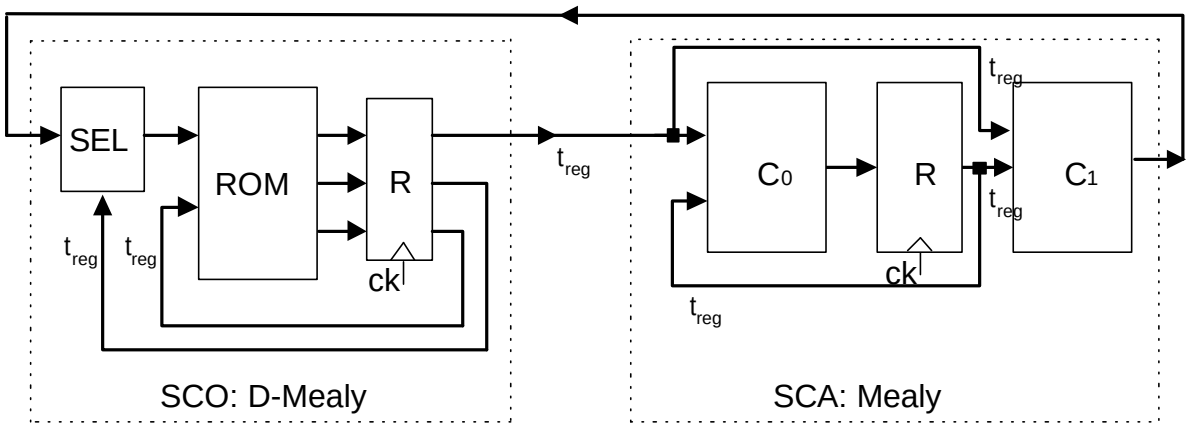
Esercizio (2S20010112-D4)

Illustrare la temporizzazione relativa ad un sistema SCO-SCA di tipo D-Mealy-Mealy.

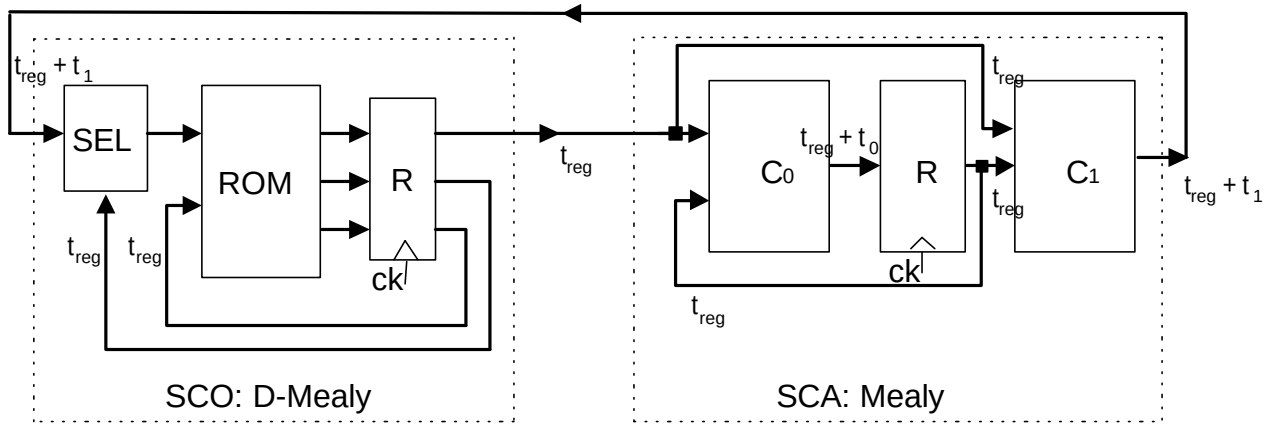
I sotto-sistemi specificati vengono accoppiati come nella figura.



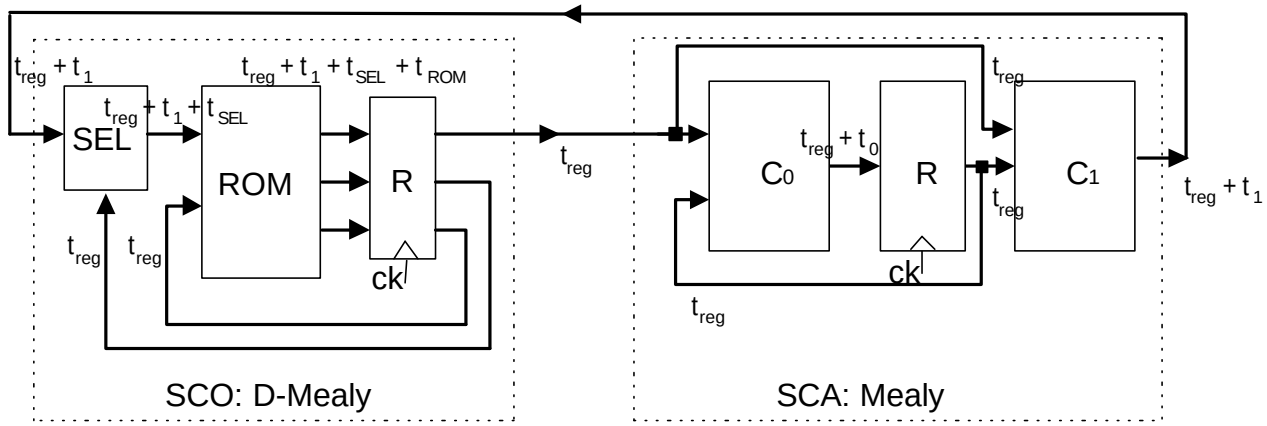
Per valutare la frequenza massima di lavoro (o il periodo minimo del segnale CK) del sistema in funzione dei parametri delle due reti, occorre determinare i tempi massimi in cui diventano stabili i segnali su tutti i nodi dei percorsi dei dati (i "cammini") che hanno origine sulle uscite dei registri e termine sugli ingressi dei registri, a partire dall'applicazione di un fronte del segnale CK; perciò le prime etichette che si possono determinare sono quelle all'uscita dei registri (t_{reg}):



Su questa base si possono ora aggiungere le etichette (i tempi massimi di stabilizzazione) delle uscite delle reti combinatorie alimentate dai registri (per comodità lo schema logico del sistema viene replicato):



Poi si prosegue con i percorsi combinatori in cascata:



Tutti i nodi della rete sono stati etichettati con i rispettivi tempi massimi di stabilizzazione; quindi si può determinare il massimo tra le due quantità etichettate sugli ingressi dei due registri:

$t_{reg} + t_1 + t_{sel} + t_{ROM}$
$t_{reg} + t_0$

a cui occorre aggiungere il tempo t_{setup} di set-up dei registri; in definitiva si ottiene:

$T_{CK} > t_{reg} + t_{setup} + \max\{ t_1 + t_{sel} + t_{ROM}, t_0\}$

Esercizio (2S20010112-D5)

Una porzione di memoria RAM del PD32 di capacità pari a 256 byte è riservata all'indirizzo iniziale FIFO_BASE per essere gestita dal processore come un buffer circolare di tipo FIFO. All'indirizzo FIFO_STATUS va registrato lo stato - vuoto, pieno - del buffer FIFO. Scrivere le due routine assembler per operare sul buffer l'inserimento e l'estrazione di un singolo byte.

Di seguito viene riportato il listato del programma, collaudato con il simulatore PD32. La versione in formato testo, utilizzabile con il simulatore, è disponibile separatamente nel materiale didattico del corso.

```
; fifo.asm

;Una porzione di memoria RAM del PD32 di capacità pari a 256 byte è riservata
;all'indirizzo iniziale FIFO_BASE per essere gestita dal processore
;come un buffer circolare di tipo FIFO.
;All'indirizzo FIFO_STATUS va registrato lo stato - vuoto, pieno -
;del buffer FIFO. Si vogliono scrivere le due routine assembler per operare
;sul buffer l'inserimento e l'estrazione di un singolo byte.

;Ovviamente in fase di inserimento o estrazione di un dato si dovrà
;rispettivamente scrivere o leggere un byte dal buffer RAM,
;e quindi si occuperà o si libererà rispettivamente una locazione
;in memoria; pertanto il software dovrà definire un puntatore
;alla prima locazione libera in cui poter scrivere e un altro
;puntatore alla prima locazione che dovrà essere letta.
;La gestione di tipo FIFO del buffer comporta che il puntatore
;di lettura "insegue" quello di scrittura, senza ovviamente
;poterlo scavalcare; infatti, questo evento corrisponderebbe
;allo svuotamento della FIFO (tutti i dati precedentemente scritti
;sono stati letti) e a segnalarlo provvede il bit FIFO_VUOTA.
;Analogamente il puntatore di scrittura non può scavalcare
;il puntatore di lettura; infatti, questo evento corrisponderebbe
;al riempimento della FIFO (sono stati scritti 256 byte dopo
;l'ultima lettura) e a segnalarlo provvede il bit FIFO_PIENA.
;Per quanto detto iL bit FIFO_PIENA verrà inizializzato a 0,
;il bit FIFO_VUOTA a 1.
;La gestione del buffer FIFO è circolare proprio in quanto l'estrazione
;(lettura) dei dati libera posizioni che possono essere riscritte,
;e quindi dopo avere scritto il byte all'indirizzo FIFO_BASE+255
;se la FIFO non è piena si potrà scrivere il byte successivo
;all'indirizzo FIFO_BASE.
;Si presuppone che l'esecuzione delle routine di accesso in
;scrittura / lettura segua il test sullo stato della FIFO;
;quindi le routine richieste non effettuano il test a priori,
;ma a posteriori, per modificare eventualmente il bit di stato
;appropriato dopo avere eseguito l'operazione richiesta.
```

```
;*************************
; COSTANTI
;*************************

org 400h                ;richiesta dall'assemblatore PD32 prima delle equ

FIFOBASE   equ 2000h    ;base del buffer
FIFOTOP    equ 20FFh    ;limite superiore del buffer
FIFOTOP1   equ 2100h    ;FIFOTOP1 equ FIFOTOP+1
stack      equ 2800h    ;inizio area di stack PD32


;*************************
; VARIABILI
;*************************

FIFOSTAT   db 01h       ;FIFOSTAT.1=FIFOPIENA; FIFOSTAT.0=FIFOVUOTA
WRITEAT    dl 0h
READAT     dl 0h

;*************************
; CODICE
;*************************

code                            ;inizio istruzioni

main:
;       clri                    ;interruzioni disabilitate al reset

        movl #stack,r7          ;inizializza R7 quale SP: deve precedere
                                ;l'istruzione SETI
                                ;per gestire correttamente lo stack nella
                                ;fase di riconoscimento di una richiesta
                                ;di interruzione

        movb #01h,FIFOSTAT      ;inzializzazione FIFO vuota

        seti                    ;abilita PD32 ad accettare interruzioni (SP è stato

                                ;inizializzato)

        movl #FIFOBASE,WRITEAT      ;inizializzazione puntatori alla FIFO
        movl #FIFOBASE, READAT

mainloop:

;programma principale di test delle due subroutine

        movb #0A5h,r0
        jsr wrfifo              ;prima scrittura: la FIFO deve risultare non vuota (e non piena)
```

```
        movb #05Ah,r0
        jsr wrfifo          ;seconda scrittura
        jsr rdfifo          ;prima lettura: la FIFO deve risultare non vuota (e non piena)
        jsr rdfifo          ;seconda lettura: la FIFO deve risultare vuota (e non piena)

                            ;segue ciclo di riempimento del buffer FIFO (256 scritture)
        movb #0C3h,r0       ;valore da scrivere
        movb #000h,r1       ;contatore del ciclo
writeon:                    ;ciclo di riempimento
        jsr wrfifo
        addb #1,r1          ;incremento il contatore
        jnz writeon
        nop                 ;la FIFO deve risultare piena (e non vuota)
                            ;inoltre, il puntatore di scrittura deve puntare a FIFO_BASE+2
                            ;(test della gestione circolare), come quello di lettura

        halt                ;per il simulatore

;*************************
; Sezione subroutine
;*************************


;*************************
wrfifo:
;presuppone r0 precaricato con il byte da scrivere

        push r1
        movl WRITEAT,r1         ;uso r1 per puntare in memoria
        movb r0,(r1)+           ;copio il byte meno significativo di r0 nel buffer
        movl r1,WRITEAT         ;incremento WRITE_AT di 1
        cmpl #FIFOTOP1,r1       ;test sul raggiungimento della fine dello spazio del buffer
        jnz isitfull
        movl #FIFOBASE,r1       ;fine del buffer: carico WRITEAT con la base del buffer
        movl r1,WRITEAT
isitfull:
        cmpl READAT,r1          ;verifico se FIFO piena
        jnz eowr
        movb FIFOSTAT,r1        ;FIFO piena: leggo FIFOSTAT
        orb #02h,r1             ;commuto FIFOSTAT.1 a 1
        movb r1,FIFOSTAT        ;e riscrivo FIFOSTAT
eowr:
        movb FIFOSTAT,r1 ;dopo una scrittura la FIFO è non vuota: leggo FIFOSTAT
        andb #02h,r1     ;commuto FIFOSTAT.0 a 0
        movb r1,FIFOSTAT ;e riscrivo FIFOSTAT

        pop r1
        ret
```

```
;*************************
rdfifo:
;legge un byte dal buffer e lo restituisce in r0

        push r1
        movl READAT,r1          ;uso r1 per puntare in memoria
        movb (r1)+,r0           ;copio il byte meno significativo di r0 nel buffer
        movl r1,READAT          ;incremento WRITEAT di 1
        cmpl #FIFOTOP1,r1       ;test sul raggiungimento della fine dello spazio del buffer
        jnz isitmpty
        movl #FIFOBASE,r1       ;fine del buffer: carico READAT con la base del buffer
        movl r1,READAT
isitmpty:
        cmpl WRITEAT,r1         ;verifico se FIFO vuota
        jnz eord
        movb FIFOSTAT,r1        ;FIFO vuota: leggo FIFOSTAT
        orb #01h,r1             ;commuto FIFOSTAT.0 a 1
        movb r1,FIFOSTAT        ;e riscrivo FIFOSTAT
eord:
        movb FIFOSTAT,r1 ;dopo una lettura la FIFO è non piena: leggo FIFOSTAT
        andb #01h,r1     ;commuto FIFOSTAT.1 a 0
        movb r1,FIFOSTAT ;e riscrivo FIFOSTAT

        pop r1
        ret

;*************************

end                     ;fine programma
```

```
; fifo.asm

;Una porzione di memoria RAM del PD32 di capacità pari a 256 byte è riservata
;all'indirizzo iniziale FIFO_BASE per essere gestita dal processore
;come un buffer circolare di tipo FIFO.
;All'indirizzo FIFO_STATUS va registrato lo stato - vuoto, pieno -
;del buffer FIFO. Si vogliono scrivere le due routine assembler per operare
;sul buffer l'inserimento e l'estrazione di un singolo byte.

;Ovviamente in fase di inserimento o estrazione di un dato si dovrà
;rispettivamente scrivere o leggere un byte dal buffer RAM,
;e quindi si occuperà o si libererà rispettivamente una locazione
;in memoria; pertanto il software dovrà definire un puntatore
;alla prima locazione libera in cui poter scrivere e un altro
;puntatore alla prima locazione che dovrà essere letta.
;La gestione di tipo FIFO del buffer comporta che il puntatore
;di lettura "insegue" quello di scrittura, senza ovviamente
;poterlo scavalcare; infatti, questo evento corrisponderebbe
;allo svuotamento della FIFO (tutti i dati precedentemente scritti
;sono stati letti) e a segnalarlo provvede il bit FIFO_VUOTA.
;Analogamente il puntatore di scrittura non può scavalcare
;il puntatore di lettura; infatti, questo evento corrisponderebbe
;al riempimento della FIFO (sono stati scritti 256 byte dopo
;l'ultima lettura) e a segnalarlo provvede il bit FIFO_PIENA.
;Per quanto detto iL bit FIFO_PIENA verrà inizializzato a 0,
;il bit FIFO_VUOTA a 1.
;La gestione del buffer FIFO è circolare proprio in quanto l'estrazione
;(lettura) dei dati libera posizioni che possono essere riscritte,
;e quindi dopo avere scritto il byte all'indirizzo FIFO_BASE+255
;se la FIFO non è piena si potrà scrivere il byte successivo
;all'indirizzo FIFO_BASE.
;Si presuppone che l'esecuzione delle routine di accesso in
;scrittura / lettura segua il test sullo stato della FIFO;
;quindi le routine richieste non effettuano il test a priori,
;ma a posteriori, per modificare eventualmente il bit di stato
;appropriato dopo avere eseguito l'operazione richiesta.


;**************************
; COSTANTI
;**************************


org 400h      ;richiesta dall'assemblatore PD32 prima delle equ

FIFOBASE  equ 2000h ;base del buffer
FIFOTOP   equ 20FFh ;limite superiore del buffer
FIFOTOP1  equ 2100h ;FIFOTOP1 equ FIFOTOP+1
stack   equ 2800h ;inizio area di stack PD32


;**************************
; VARIABILI
;**************************

FIFOSTAT  db 01h    ;FIFOSTAT.1=FIFOPIENA; FIFOSTAT.0=FIFOVUOTA
WRITEAT   dl 0h
READAT    dl 0h

;**************************
; CODICE
;**************************

code        ;inizio istruzioni

main:
; clri      ;interruzioni disabilitate al reset
```

```
  movl #stack,r7    ;inizializza R7 quale SP: deve precedere
        ;l'istruzione SETI
        ;per gestire correttamente lo stack nella
        ;fase di riconoscimento di una richiesta
        ;di interruzione

  movb #01h,FIFOSTAT  ;inzializzazione FIFO vuota

  seti      ;abilita PD32 ad accettare interruzioni (SP è stato
        ;inizializzato)

  movl #FIFOBASE,WRITEAT  ;inizializzazione puntatori alla FIFO
  movl #FIFOBASE, READAT

mainloop:

;programma principale di test delle due subroutine

  movb #0A5h,r0
  jsr wrfifo    ;prima scrittura: la FIFO deve risultare non vuota (e non piena)
  movb #05Ah,r0
  jsr wrfifo    ;seconda scrittura
  jsr rdfifo    ;prima lettura: la FIFO deve risultare non vuota (e non piena)
  jsr rdfifo    ;seconda lettura: la FIFO deve risultare vuota (e non piena)

        ;segue ciclo di riempimento del buffer FIFO (256 scritture)
  movb #0C3h,r0   ;valore da scrivere
  movb #000h,r1   ;contatore del ciclo
writeon:      ;ciclo di riempimento
  jsr wrfifo
  addb #1,r1    ;incremento il contatore
  jnz writeon
  nop     ;la FIFO deve risultare piena (e non vuota)
        ;inoltre, il puntatore di scrittura deve puntare a FIFO_BASE+2
        ;(test della gestione circolare), come quello di lettura

  halt      ;per il simulatore

;*************************
; Sezione subroutine
;*************************


;*************************
wrfifo:
;presuppone r0 precaricato con il byte da scrivere

  push r1
  movl WRITEAT,r1   ;uso r1 per puntare in memoria
  movb r0,(r1)+   ;copio il byte meno significativo di r0 nel buffer
  movl r1,WRITEAT   ;incremento WRITE_AT di 1
  cmpl #FIFOTOP1,r1 ;test sul raggiungimento della fine dello spazio del buffer
  jnz isitfull
  movl #FIFOBASE,r1 ;fine del buffer: carico WRITEAT con la base del buffer
  movl r1,WRITEAT
isitfull:
  cmpl READAT,r1    ;verifico se FIFO piena
  jnz eowr
  movb FIFOSTAT,r1  ;FIFO piena: leggo FIFOSTAT
  orb #02h,r1   ;commuto FIFOSTAT.1 a 1
  movb r1,FIFOSTAT  ;e riscrivo FIFOSTAT
eowr:
  movb FIFOSTAT,r1  ;dopo una scrittura la FIFO è non vuota: leggo FIFOSTAT
  andb #02h,r1    ;commuto FIFOSTAT.0 a 0
  movb r1,FIFOSTAT  ;e riscrivo FIFOSTAT

  pop r1
  ret
```

```
;*************************
rdfifo:
;legge un byte dal buffer e lo restituisce in r0

  push r1
  movl READAT,r1    ;uso r1 per puntare in memoria
  movb (r1)+,r0   ;copio il byte meno significativo di r0 nel buffer
  movl r1,READAT    ;incremento WRITEAT di 1
  cmpl #FIFOTOP1,r1 ;test sul raggiungimento della fine dello spazio del buffer
  jnz isitmpty
  movl #FIFOBASE,r1 ;fine del buffer: carico READAT con la base del buffer
  movl r1,READAT
isitmpty:
  cmpl WRITEAT,r1   ;verifico se FIFO vuota
  jnz eord
  movb FIFOSTAT,r1  ;FIFO vuota: leggo FIFOSTAT
  orb #01h,r1   ;commuto FIFOSTAT.0 a 1
  movb r1,FIFOSTAT  ;e riscrivo FIFOSTAT
eord:
  movb FIFOSTAT,r1  ;dopo una lettura la FIFO è non piena: leggo FIFOSTAT
  andb #01h,r1    ;commuto FIFOSTAT.1 a 0
  movb r1,FIFOSTAT  ;e riscrivo FIFOSTAT

  pop r1
  ret

;*************************

end       ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 26-01-2001

STUDENTE:_____DOCENTE:_____

Si vuole realizzare un modulatore per segnali digitali (DIGIMOD) per trasferire messaggi predisposti nella RAM di un processore PD32 su una linea analogica alla velocità di 64 kbit/s.
Quando il processore vuole trasmettere un messaggio invia alla periferica DIGIMOD l'indirizzo iniziale e la lunghezza - in byte - del messaggio.
In risposta la periferica DIGIMOD esegue le seguenti attività:

- preleva il messaggio mediante accesso in DMA di tipo stealing;
- converte ogni coppia di bit del messaggio da trasmettere in una forma d'onda sinusoidale digitalizzata costituita da 256 campioni a 8 bit memorizzati in un banco di ROM. La forma d'onda viene letta dal banco di ROM con una fase che dipende dalla coppia di bit da trasmettere, secondo la seguente tavola:

Coppia di bit	Fase	Forma d'onda
00	0°	
01	90°	
10	180°	
11	270°	

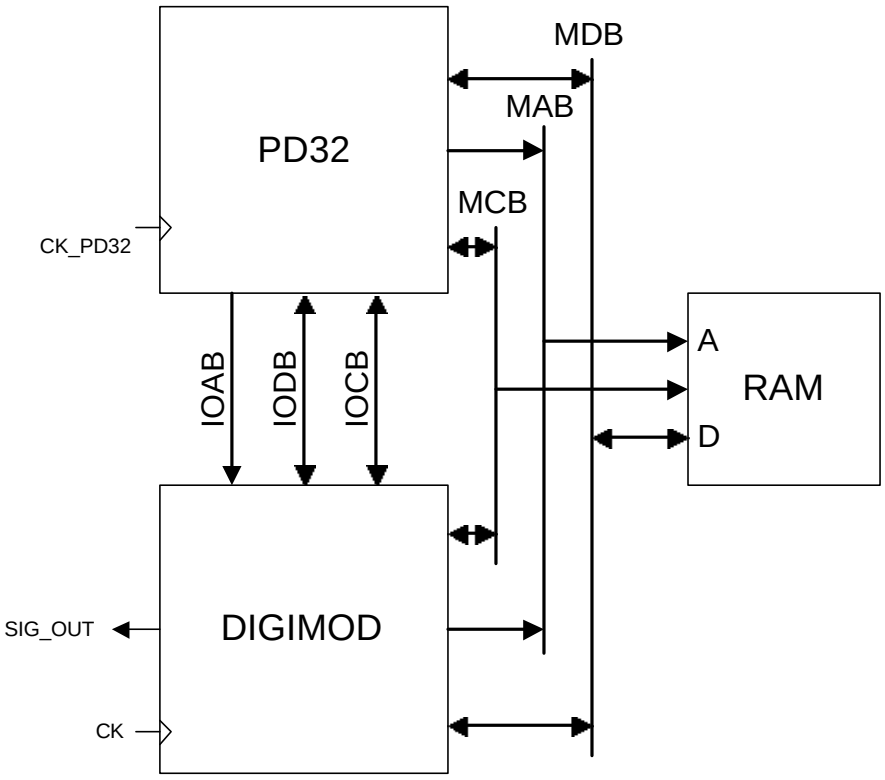
- invia la sequenza dei campioni della forma d'onda ad un DAC dotato di un tempo di conversione adeguato;
- segnala il termine della trasmissione al processore con una interruzione.

Si dispone di moduli ROM con tempo di accesso pari a 150 ns. Si suppongano trascurabili i tempi di commutazione delle porte logiche e dei registri.
Il processore ha un clock a 25 MHz.

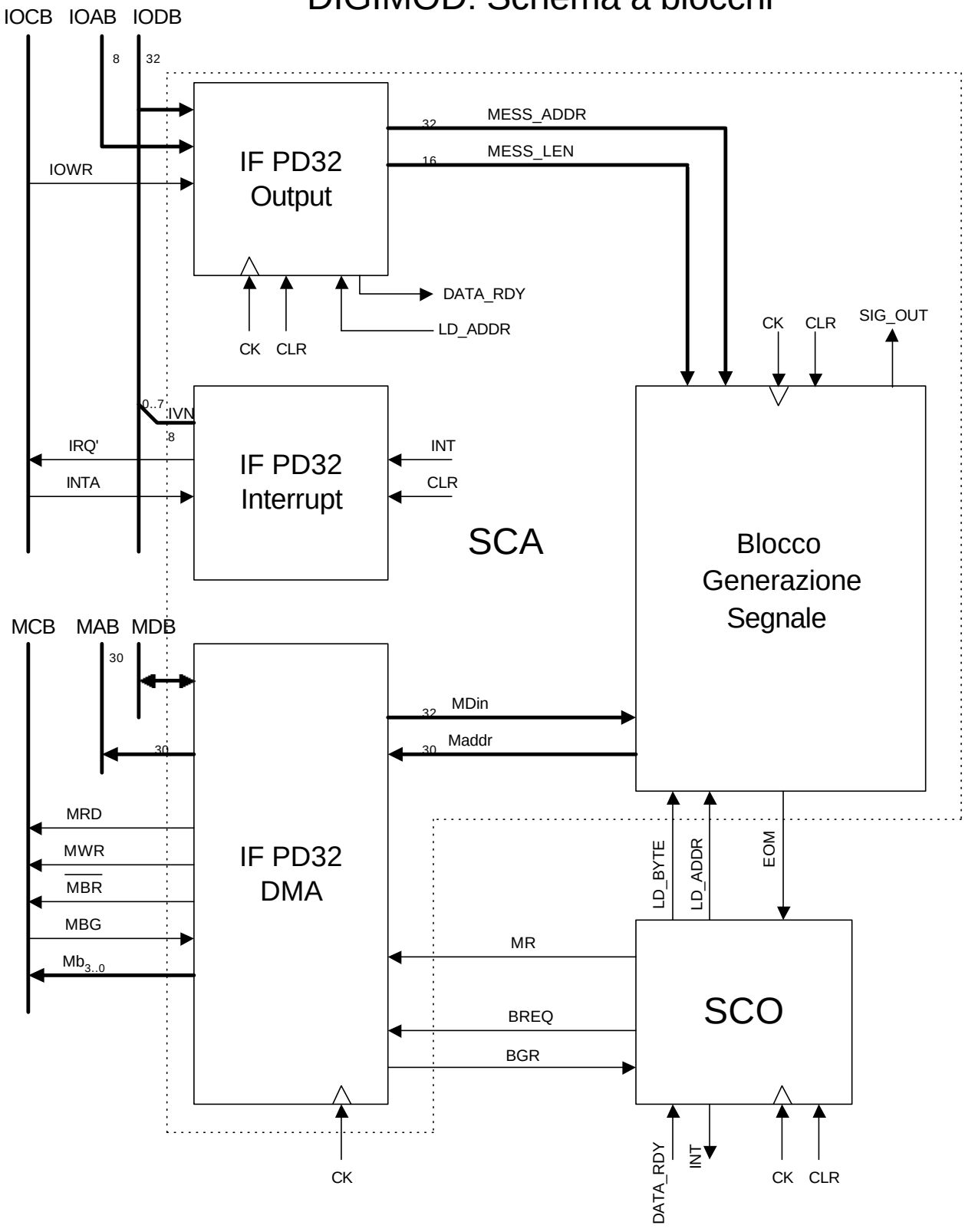
Si richiede:

1. la temporizzazione delle operazioni;
2. lo schema logico della periferica DIGIMOD;
3. il dimensionamento del banco di ROM e la distribuzione dei dati nei moduli ROM;
4. la frequenza del clock della periferica DIGIMOD;
5. le routine d'interfacciamento del PD32.

DIGIMOD: sistema esterno



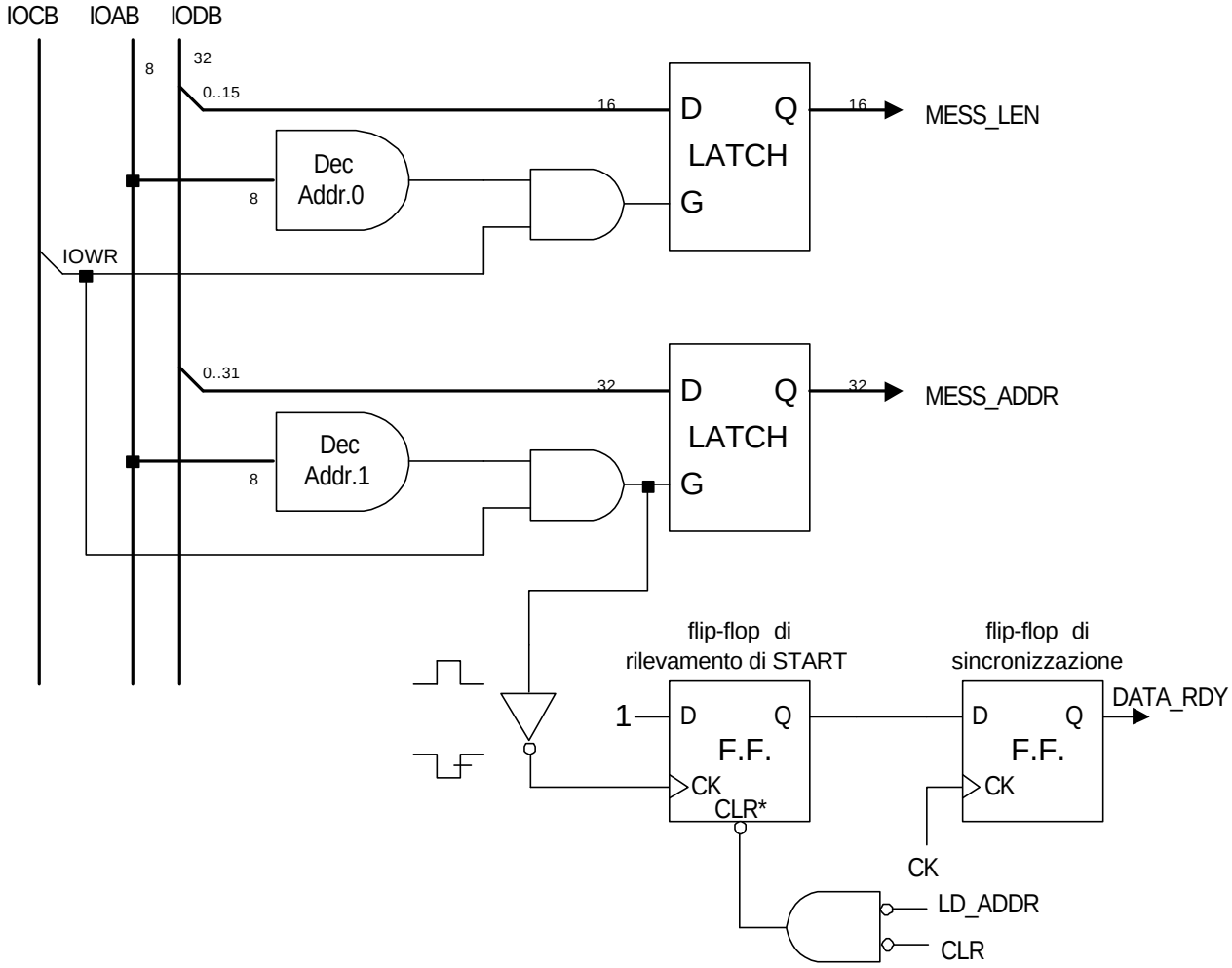
DIGIMOD: Schema a blocchi



Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.
Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

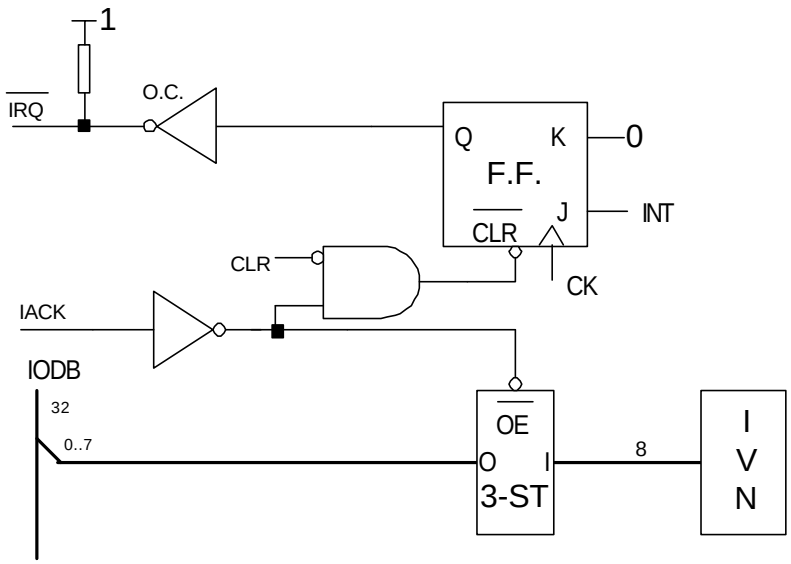
DIGIMOD: IF PD32 - output



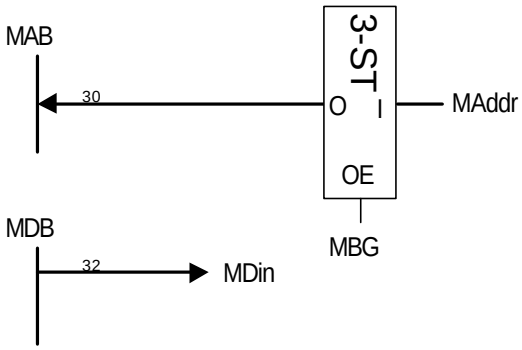
Note

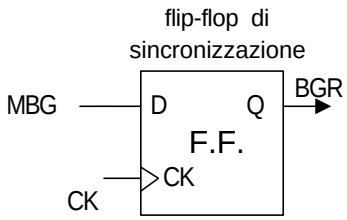
Il SW avvia l'operazione con **OUT MESS_ADDR,Addr1** dopo avere riscritto l'altro registro di interfaccia con la lunghezza del messaggio (cfr. routine software).

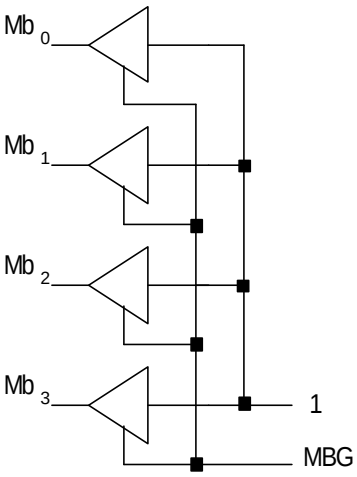
DIGIMOD: IF PD32 - interrupt

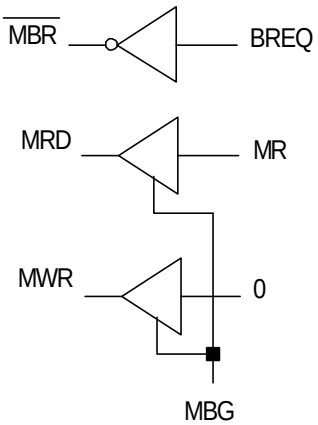


DIGIMOD: IF PD32 - DMA





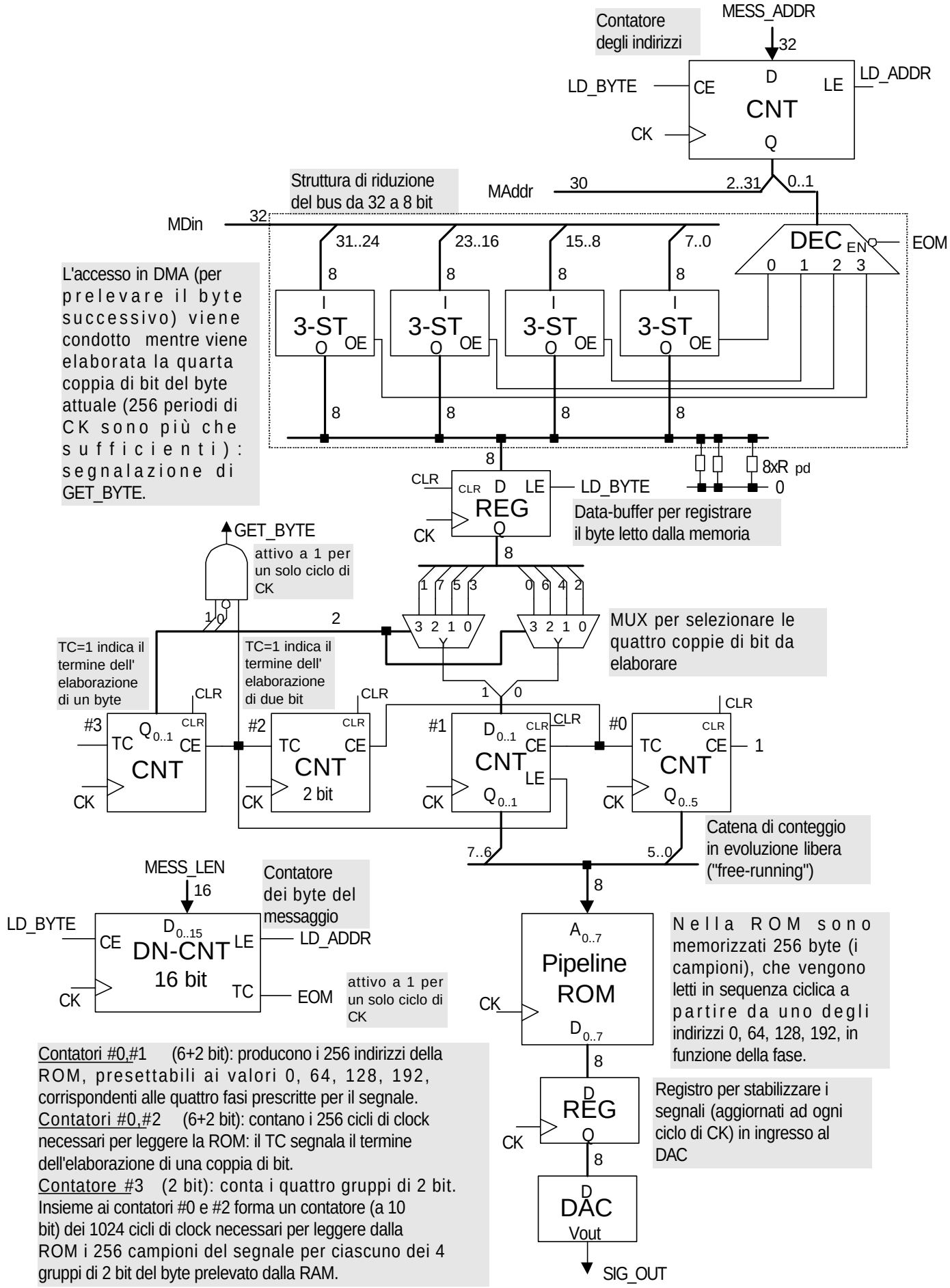




Dalla RAM vengono lette LW, di cui però viene registrato un byte alla volta.

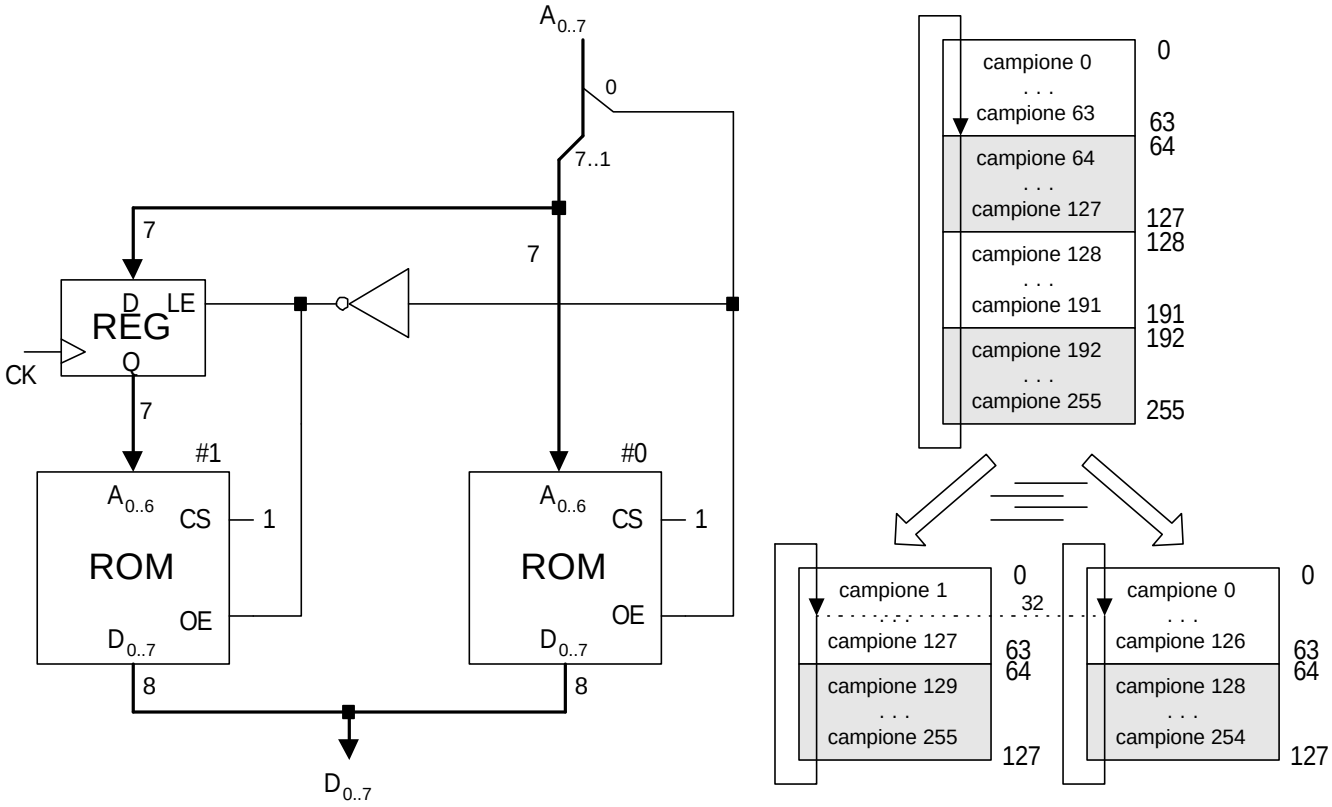
MWR deve essere pilotato (a 0), anche se non dinamicamente: la struttura di memoria deve essere governata pienamente dall'interfaccia DMA, che ne ha piena responsabilità quando MBG=1.

DIGIMOD: blocco generatore segnale



Contatori #0,#1 (6+2 bit): producono i 256 indirizzi della ROM, presettabili ai valori 0, 64, 128, 192, corrispondenti alle quattro fasi prescritte per il segnale.
Contatori #0,#2 (6+2 bit): contano i 256 cicli di clock necessari per leggere la ROM: il TC segnala il termine dell'elaborazione di una coppia di bit.
Contatore #3 (2 bit): conta i quattro gruppi di 2 bit. Insieme ai contatori #0 e #2 forma un contatore (a 10 bit) dei 1024 cicli di clock necessari per leggere dalla ROM i 256 campioni del segnale per ciascuno dei 4 gruppi di 2 bit del byte prelevato dalla RAM.

DIGIMOD: pipeline ROM



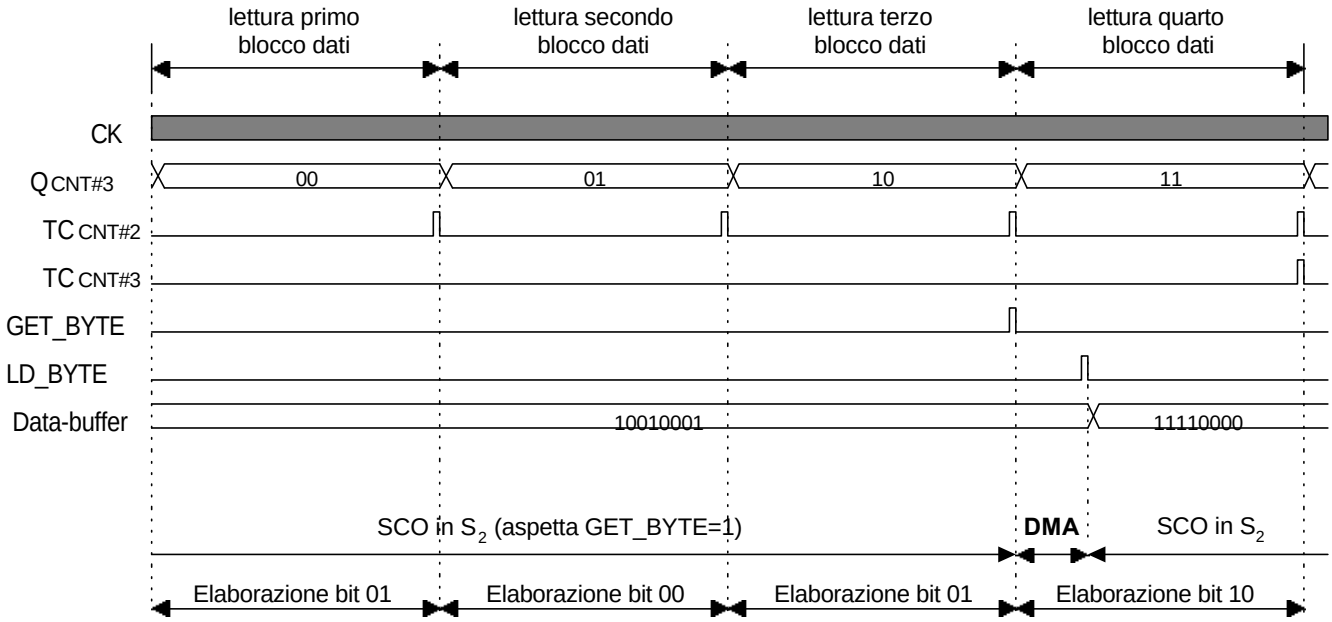
Strutture virtuale (blocchi sequenziali) e fisica (blocchi parallelizzati - pipeline) della ROM. Esempio di accesso sequenziale relativo alla fase 90° (coppia di bit 01), di tipo circolare (lettura di tutti i 256 campioni, a partire dal 65-esimo fino al 64-esimo).

La struttura di pipeline parallelizza 2 moduli di ROM; infatti: $t_{CK} < t_A < t_{CK} \times 2$, essendo t_A=150ns e t_{CK}=122ns.

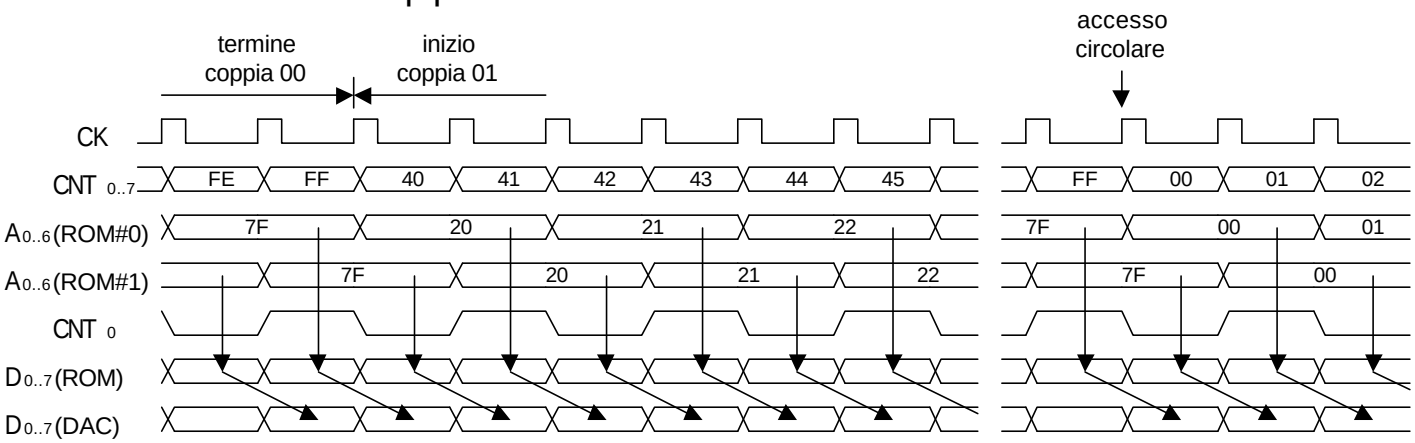
Nel caso generale di accesso *casuale* ai due moduli ROM, ciascuno di capacità 256 x 8 bit, dovrebbero essere predisposti due registri di ingresso da caricare a cicli di clock alterni con il dato di uscita del contatore; in questo caso specifico invece l'accesso alle ROM deve essere di tipo *sequenziale*, in particolare dall'indirizzo 0 a 255; ciò implica che in una struttura pipeline con 2 ROM da 256 indirizzi da una ROM verrebbero letti soltanto i dati disposti agli indirizzi pari, e dall'altra ROM soltanto i dati di indirizzo dispari; pertanto è possibile e conveniente memorizzare soltanto la metà dei dati in ciascuna delle due ROM: ad esempio nella ROM #0 i dati relativi agli indirizzi pari (0, 2, .., 254) generati dal contatore, nella ROM #1 i dati relativi agli indirizzi dispari (1, 3, .., 255); ovviamente i dati (byte) memorizzati in ciascuna ROM sono 256/2=128 (7 bit di indirizzo).
Gli indirizzi di ROM, variabili da 0 a 127 ogni due cicli di CK, sono disponibili sulle uscite 1..7 del contatore; pertanto è possibile connettere tali uscite direttamente all'ingresso della ROM #0, mentre per ritardarne di un periodo di CK la presentazione al modulo #1 deve essere introdotto un registro (di pipeline) in cascata al contatore, con l'ingresso di abilitazione a caricamento collegato al bit 0 (LSB) invertito del contatore.
Le uscite delle due ROM dovranno essere selezionate alternativamente al ritmo di CK; in questo caso è conveniente pilotare i buffer 3-state (disposti sul livello di uscita) delle ROM tramite lo stesso bit 0 (LSB) del contatore; va notato che quest'ultimo commuta ad ogni ciclo di CK, e quindi ci si pone nell'ipotesi ragionevole che la ROM risponda all'attivazione di OE entro un singolo periodo di CK.

DIGIMOD: temporizzazioni

Sistema

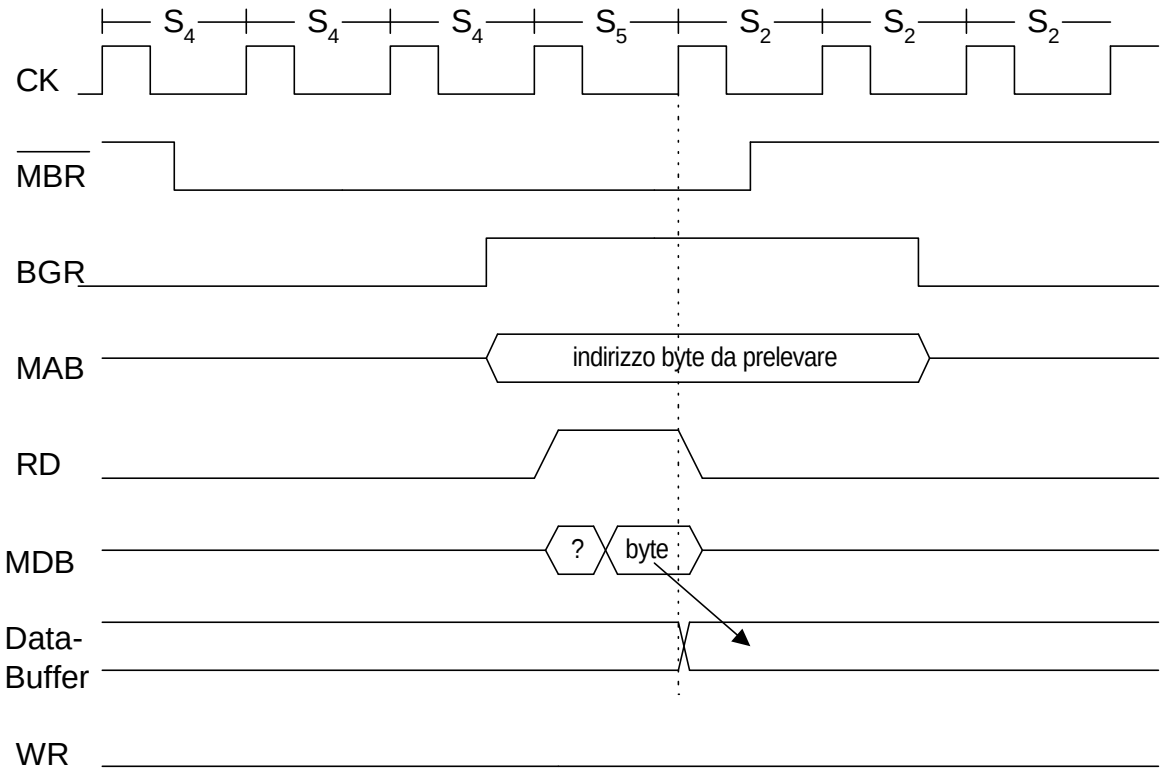


Accessi alla ROM-pipeline

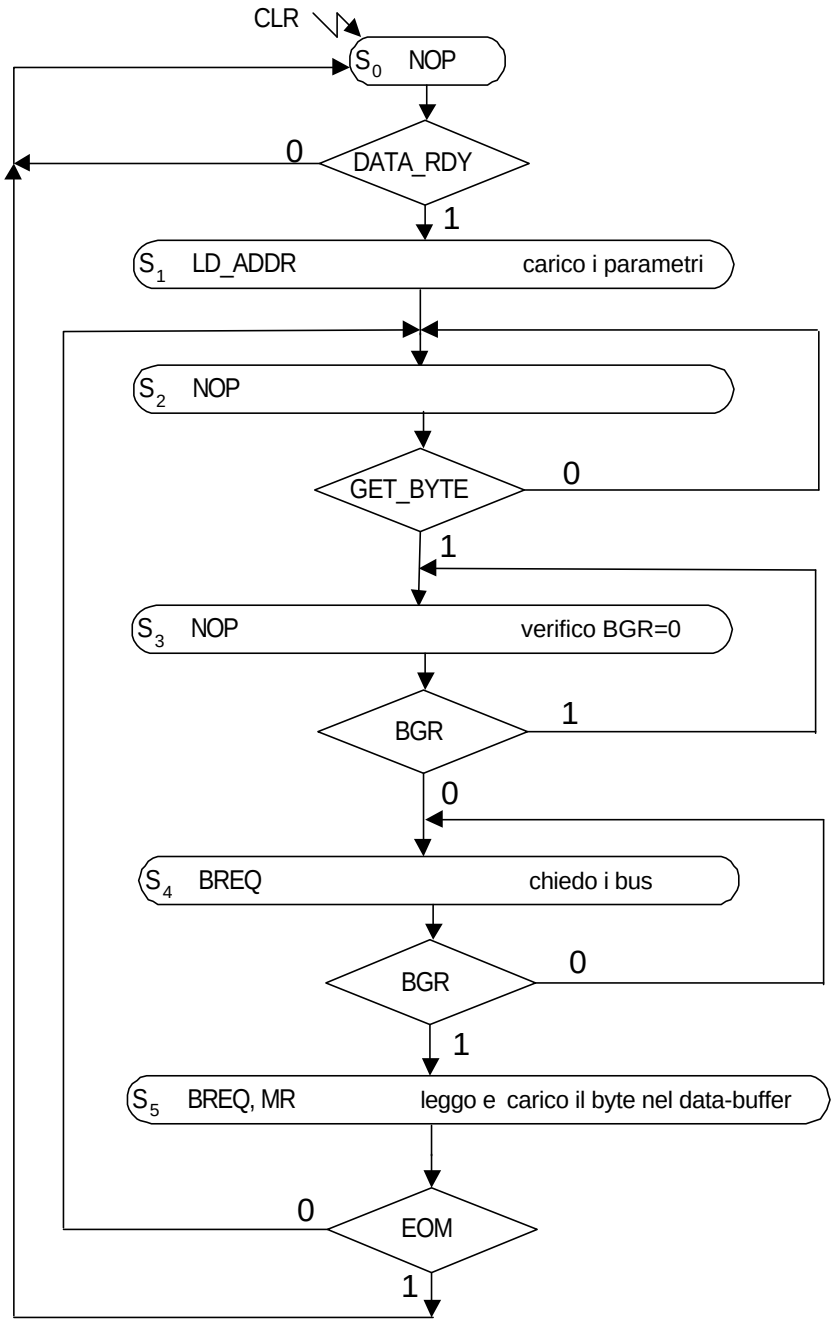


DIGIMOD: temporizzazioni

Richiesta / rilascio bus di sistema, accesso in RAM e relativi stati dello SCO



DIGIMOD: SCO - flowchart



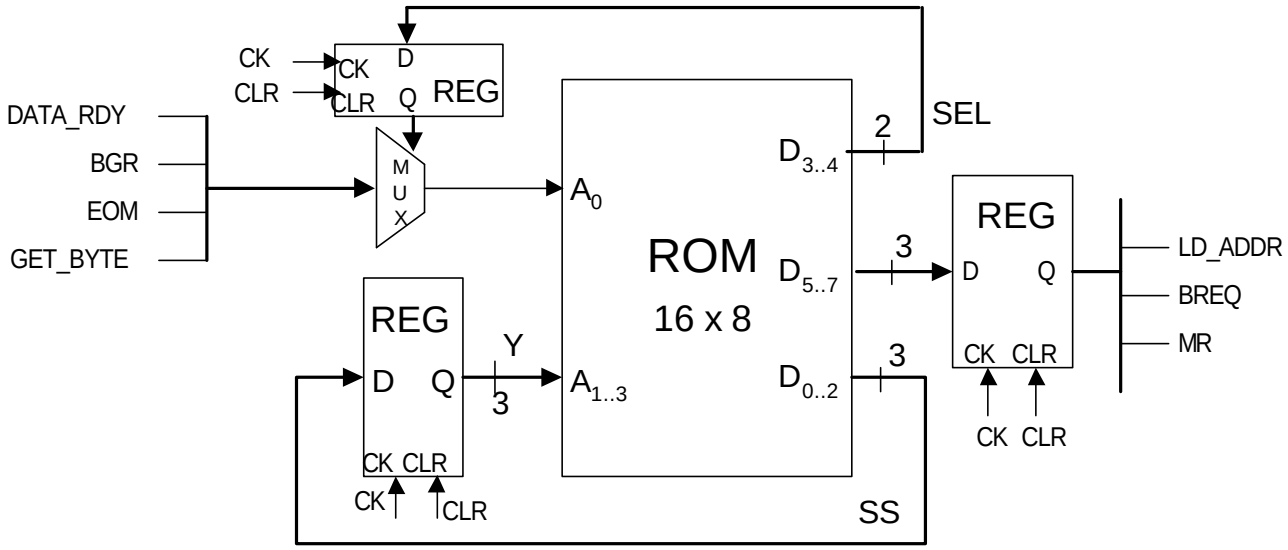
In S_2 lo SCO persiste per la frazione di tempo F=(256-3-R)/256 di un ciclo operativo (S_2,S_3,S_4,S_5), essendo R il numero (aleatorio) di cicli di CK della periferica che impiega il PD32 a cedere i bus. Ad esempio, R=7 implica F=96%.

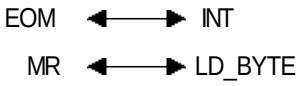
S_3 è previsto dal protocollo a quattro fasi; è tuttavia evidente che dopo l'attesa in S_2 per una frazione in eccesso del 90% dell'intero ciclo operativo, sarà normalmente BGR=0.

S_5: stato di lettura del byte in RAM

DIGIMOD: SCO - struttura HW microprogrammata

Il flow-chart è supportato da un microlinguaggio di tipo 3; conviene scegliere il modello strutturale di tipo D-Mealy, per la presenza del registro di TASK, necessario per presentare i segnali BREQ e MR diretti a dispositivi esterni (LD_ADDR è diretto anche all'ingresso di CLR asincrono del flip-flop di start) senza spike.





Una realizzazione alternativa per lo SCO può essere quella con componenti MSI (contatore, decoder e multiplexer); il flow-chart presenta un ritorno nello stato 2, oltre ovviamente a quello nello stato 0, che implica l'uso di un contatore dotato di ingresso precaricabile (0 o 2).

DIGIMOD: SCO - struttura HW a sequenziatore

Calcolo dei parametri del circuito

· Equazione di abilitazione all'incremento del conteggio (stato)

CE = S0 DATA_RDY + S1 + S2 GET_BYTE + S3 BGR* + S4 BGR

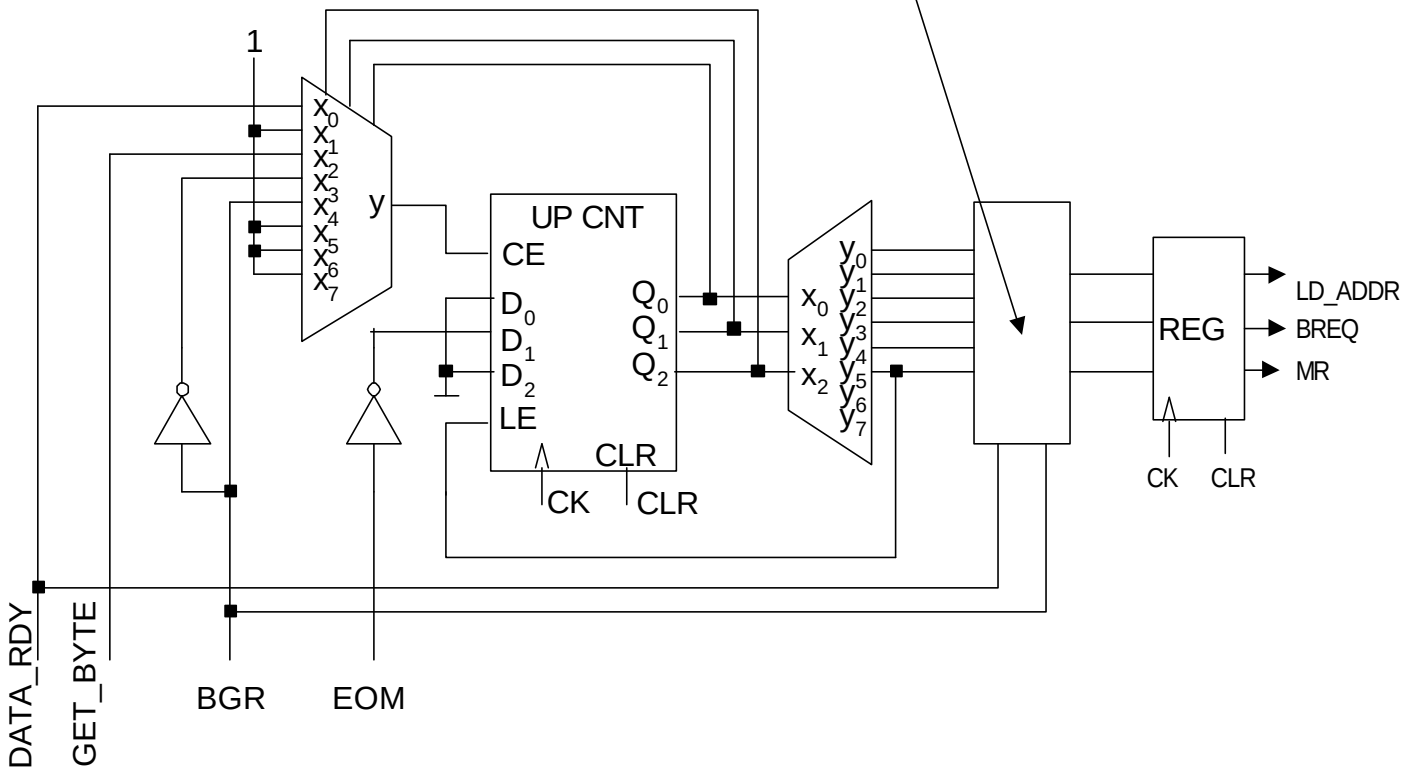
· Equazione di caricamento dello stato del contatore

LE = S5

· Equazioni delle variabili (entranti nel registro) di uscita

LD_ADDR = S0 DATA_RDY
MR = S4 BGR
BREQ = S3 BGR* + S4

Circuito dello SCO



DIGIMOD: Routine Software

```
ORG 400h

MESS_LEN DW 8000       ;inizializzo lunghezza del messaggio a 8000 byte
MESS_ADDR DL 0FF000000h    ;indirizzo (32 bit) iniziale del messaggio
Addr0 EQU 0A5h
Addr1 EQU 0A6h
FLAG DB 0


stack       equ 2800h   ;inizio area di stack
digiivn     equ 00Bh    ;ivn digimod
digidr      equ 1000h   ;indirizzo driver digimod
…

CODE

movl #stack,r7          ;inizializzo R7 (SP)
seti

mainloop:
...
cmpb #0,FLAG            ;test su messaggio spedito
jz altro
jsr start_digimod       ;richiesta di spedizione del messaggio successivo
                        ;(si suppone comunque predisposto)

altro:
;predisposizione del messaggio successivo
;altri task

jmp mainloop

; * * * R O U T I N E * * *

start_digimod:
outw MESS_LEN,Addr0     ;caricamento lunghezza messaggio
outl MESS_ADDR,Addr1    ;caricamento indirizzo iniziale messaggio e start
movb #00,FLAG           ;reset flag di messaggio trasmesso
ret

; * * * DRIVER * * *

driver digiivn,digidr
movb #01,FLAG           ;set flag di messaggio trasmesso
rti
```

NOTE

- Calcolo della frequenza di CK:
  l'intervallo minimo di lavoro della periferica è quello che separa la presentazione di due campioni successivi al DAC. Per ogni coppia di bit del messaggio devono essere presentati 256 campioni al DAC; la velocità prescritta di consumo del messaggio è pari a 64 kbit/s = 32 k(coppie di bit)/s, pertanto la velocità di presentazione dei campioni al DAC deve essere pari a: $32 \cdot 256$ kcampioni/s = 2^{13} kcampioni/s = 8192 Kcampioni/s = 8.192 Mcampioni/s: in definitiva la frequenza F_{ck} deve essere pari a 8.192 MHz.

- Il tempo di ciclo T_{ck} è pari a 1 / 8.192 μs = 1000 / 8.192 ns ≅ 122 ns. Il confronto con il tempo di accesso della ROM t_A implica il ricorso ad una struttura pipeline con parallelismo 2 per la ROM; infatti: $T_{ck} < t_A < 2\ T_{ck}$.

- la frequenza F_{ck} è tale che $3\ F_{ck} < F_{PD32}$, oppure: $3\ T_{PD32} < T_{CK}$; tale relazione implica che la periferica può accedere in lettura alla RAM di sistema in un solo periodo T_{ck}, in quanto per ipotesi di lavoro è: $t_A < 3\ T_{PD32}$ e quindi sarà certamente verificato che: $t_A < 3\ T_{PD32} < T_{CK}$.

- L'accesso in DMA è stato predisposto durante l'elaborazione della quarta coppia di bit del byte prelevato dalla RAM; questa scelta è motivata dalla durata di tale intervallo di elaborazione, pari a 256 T_{ck}, corrispondente a $256 \cdot 25/8.192 = 787\ T_{PD32}$; 787 periodi di clock rappresentano un intervallo sufficientemente lungo per ritenere ragionevolmente possibile una transazione in DMA a singolo accesso (stealing). Mentre l'accesso in DMA è in corso i due ultimi bit del byte corrente sono elaborati nel contatore a valle del registro data-buffer all'uscita del canale di memoria; pertanto appena il nuovo byte prelevato dalla RAM sarà disponibile potrà essere tamponato nel data-buffer, che potrà essere sovrascritto, in quanto il dato precedente sarà stato completamente utilizzato (i due ultimi bit sono ancora in fase di elaborazione).

- Prima e dopo la trasmissione di un messaggio predisposto dal micro, la periferica invia continuamente la sequenza nulla (00…); questa è una ipotesi di lavoro realistica in molte applicazioni, ma che non vuole escludere altre soluzioni alternative altrettanto valide, quale ad esempio quella di disattivare la portante ed inviare un livello continuo in assenza di messaggio in linea. L'ipotesi scelta è implementata azzerando il data-buffer e la catena di conteggio alla prima attivazione della periferica – tramite l'ingresso di clear asincrono - e di nuovo il data-buffer anche al termine della trasmissione di ogni messaggio – tramite il caricamento del vettore nullo; quest'ultimo viene prodotto con la disabilitazione del decoder che pilota i quattro buffer 3-state del blocco di riduzione del bus da 32 a 8 bit; infatti, con i driver 3-state disabilitati il bus assume la configurazione 0..0 per effetto degli 8 resitori di pull-down, ancorati a 0 logico.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 26-01-2001

STUDENTE:_____DOCENTE:_____

D1 Determinare il valore decimale del numero rappresentato in complemento a 2 nel formato II.Dh dalla stringa esadecimale:
81.9

D2 Sintetizzare in logica CMOS la rete combinatoria governata dall'equazione:
$Y = X_0 X_1 + X_0 X_2 + X_3$

D3 Progettare un generatore della sequenza di periodo $6T_{CK}$:
010011

D4 Determinare la frequenza massima del clock di un sistema SCO-SCA di tipo Moore-Mealy in funzione dei parametri delle due reti.

D5 Una periferica dotata di un clock di F MHz produce dati ad una velocità di F Mbit/s, che scarica nella RAM di un PD32 in DMA con modalità "stealing". Determinare il rallentamento percentuale subito dal processore, nell'ipotesi che sia sincrono con la periferica ed esegua cicli macchina di 3 periodi di clock.

Esercizio (2S20010126-D1)

Determinare il valore decimale del numero rappresentato in complemento a 2 nel formato II.Dh dalla stringa esadecimale:
81.9

---

1. Trasformazione della stringa esadecimale in codice binario:

$81.9_{16} = 10000001.1001_2$

Il bit più pesante è 1, quindi il numero rappresentato è negativo, nel formato specificato: 8 bit per la parte intera (+ 4 bit dopo la virgola fissa).

2. Complementazione a 2 del numero binario e sua valutazione decimale:

$01111110.0111_2 = (2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^{-2} + 2^{-3} + 2^{-4})_{10} = 126.4375_{10}$

da cui segue che il numero proposto vale: -126.4375

Allo stesso risultato si può pervenire eseguendo la valutazione della stringa interpretandola come numero positivo, ottenendo:
129.5625
A questo punto, ricordando che con 8 bit di parte intera (indipendentemente dall'estensione della parte dopo la virgola) i numeri rappresentati in complemento a 2 sono di fatto espressi in complemento a $2^8 = 256$, si riconosce che 129.5625 (> 128) è la rappresentazione del numero negativo:
129.5625 - 256 = -126.4375

Esercizio (2S20010126-D2)

Sintetizzare in logica CMOS la rete combinatoria governata dall'equazione:
$Y = X_0 X_1 + X_0 X_2 + X_3$

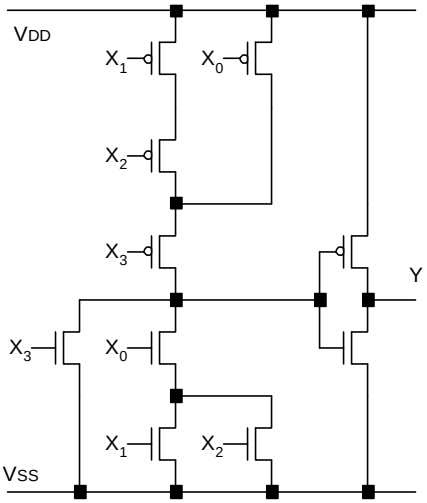
In logica CMOS complementare vanno sintentizzate le due funzioni diretta e negata, applicando possibilmente la fattorizzazione:

$Y = X_0 X_1 + X_0 X_2 + X_3 = X_0 (X_1 + X_2) + X_3$

applicando il teorema di De Morgan a quest'ultima espressione si ottiene subito la forma fattorizzata di Y* (il simbolo * indica la negazione):

$Y^* = (X_0^* + X_1^* X_2^*) X_3^*$

A questo punto si osserva che la funzione diretta ha tutti i letterali diretti e la funzione negata ovviamente li ha tutti negati; come è noto, in questo caso conviene scambiare le due funzioni, attribuendo Y* alla rete p-mos e Y alla rete n-mos, e introdurre un invertitore sull'uscita. A questo punto si può tracciare la rete di transistori che implementa la porta generalizzata richiesta:



Esercizio (2S20010126-D3)

Progettare un generatore della sequenza di periodo $6T_{CK}$:
010011

Il generatore può essere ottimizzato come un sistema sequenziale sincrono privo di ingressi a livello (sistema autonomo), che quindi procede in evoluzione libera al ritmo del clock.
Il primo passo è la descrizione del diagramma degli stati o, in modo equivalente, la tavola degli stati (macchina di Moore):

S	**S'**	Z
S_0	S_1	0
S_1	S_2	1
S_2	S_3	0
S_3	S_4	0
S_4	S_5	1
S_5	S_0	1

Gli stati S_0 .. S_5 si succedono ciclicamente privi di condizionamento.

La sequenza dei bit specificati viene assegnata alla colonna delle uscite.

Codificando gli stati con i pedici (tre bit) dei simboli si ottiene la tavola degli stati codificata a lato.
La successione degli stati rappresenta un conteggio; quindi è conveniente sintetizzare la rete con flip-flop T; pertanto si può inserire anche la colonna dei bit di ingresso dei flip-flop T.

S	$\mathbf{Y_2Y_1Y_0}$	$\mathbf{Y_2'Y_1'Y_0'}$	$\mathbf{T_2T_1T_0}$	**Z**
S_0	000	001	001	0
S_1	001	010	011	1
S_2	010	011	001	0
S_3	011	100	111	0
S_4	100	101	001	1
S_5	101	000	101	1
	110	- - -	- - -	-
	111	- - -	- - -	-

Le equazioni di eccitazione dei flip-flop T_1 e T_2 (si vede che T_0=1) e di Z possono essere ricavate con l'ausilio delle MK:

Y_1Y_0 / Y_2	00	01	11	10
0		1	1	
1			-	-

$T_1 = Y_2^* Y_0$

Y_1Y_0 / Y_2	00	01	11	10
0			1	
1		1	-	-

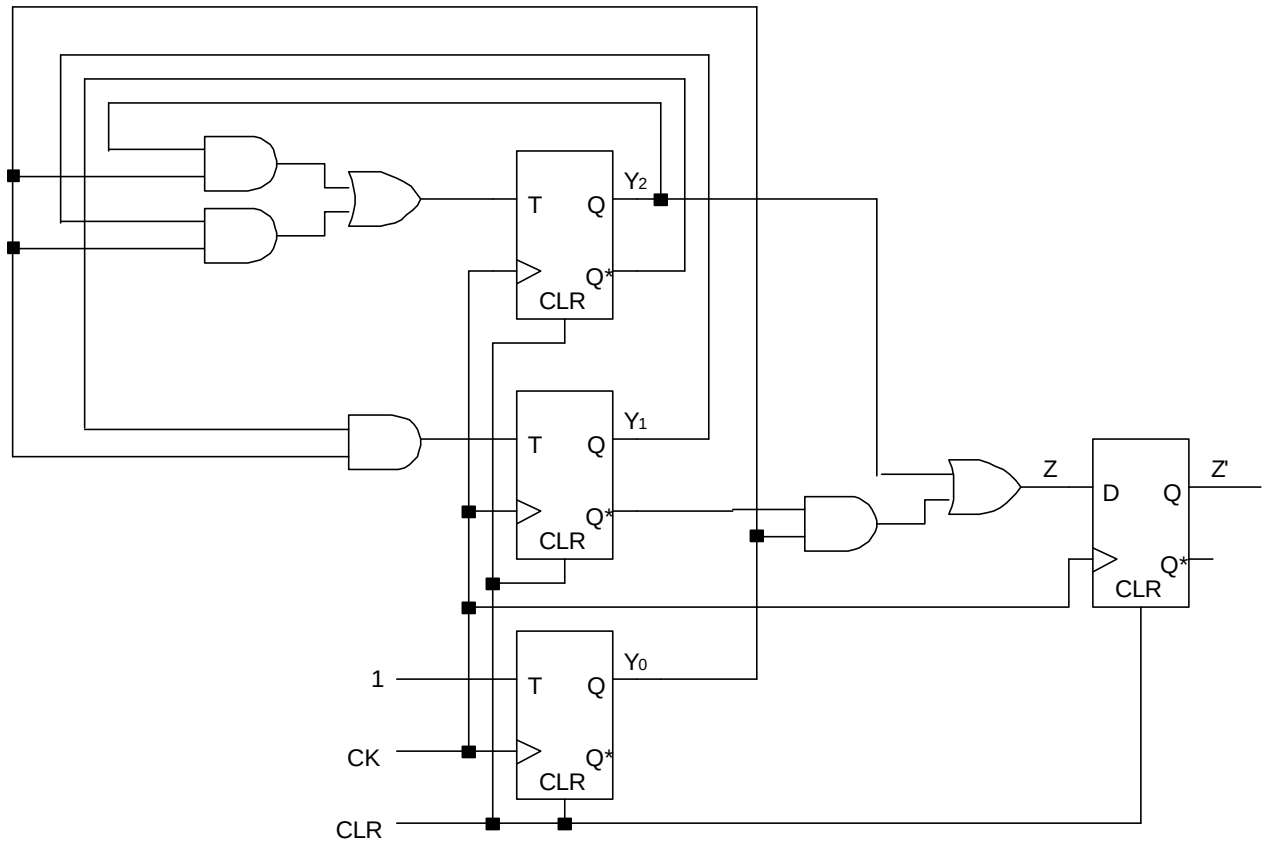
$T_2 = Y_2 Y_0 + Y_1 Y_0$

Y_2 \ Y_1Y_0	00	01	11	10
0		1		
1	1	1	-	-

$$Z = Y_2 + Y_1^* Y_0$$

In cui il simbolo * è stato usato con il significato di negazione.

La rete sequenziale che ne deriva è graficata di seguito:



Si notino:

- il pilotaggio degli ingressi di clear asincrono dei flip-flop per:
  - inizializzare il controllo su S0;
  - prevenire il problema del lock-out (cfr. condizioni non specificate).

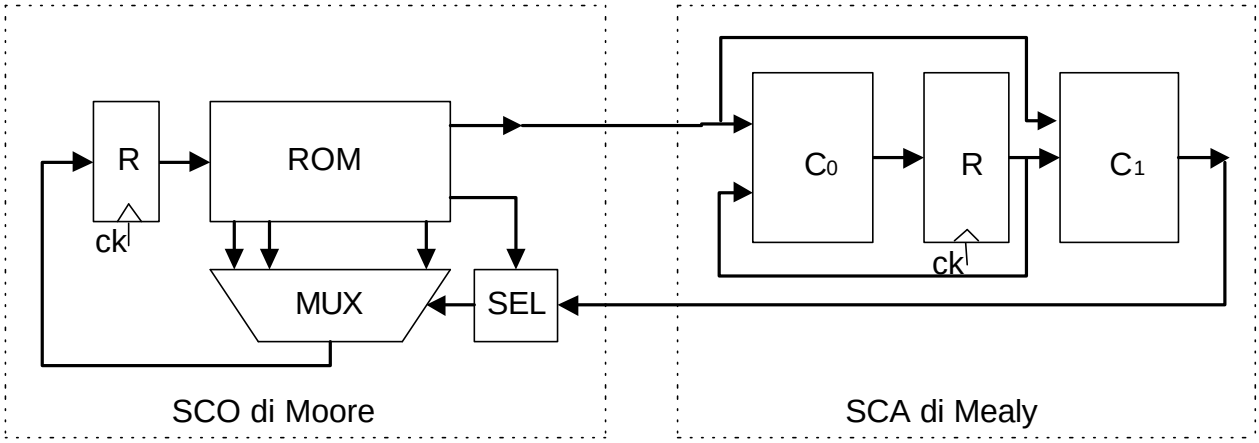
- Il flip-flop D sull'uscita Z, per presentare all'esterno la forma d'onda richiesta priva di spike (Z').

- la struttura canonica della rete sequenziale sincrona, il cui registro di stato è costituito in questo caso dai flip-flop T e D.

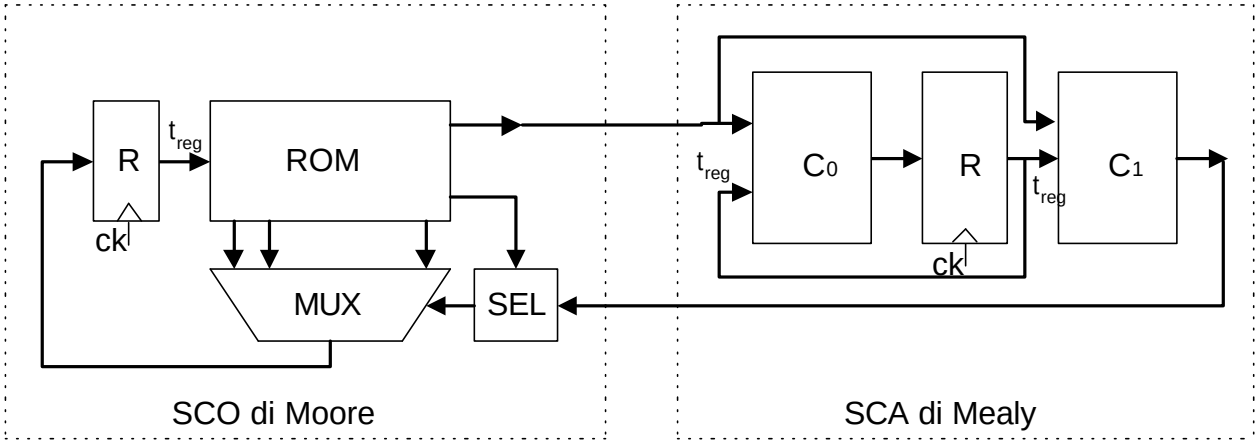
Esercizio (2S20010126-D4)

Determinare la frequenza massima del clock di un sistema SCO-SCA di tipo Moore-Mealy in funzione dei parametri delle due reti.

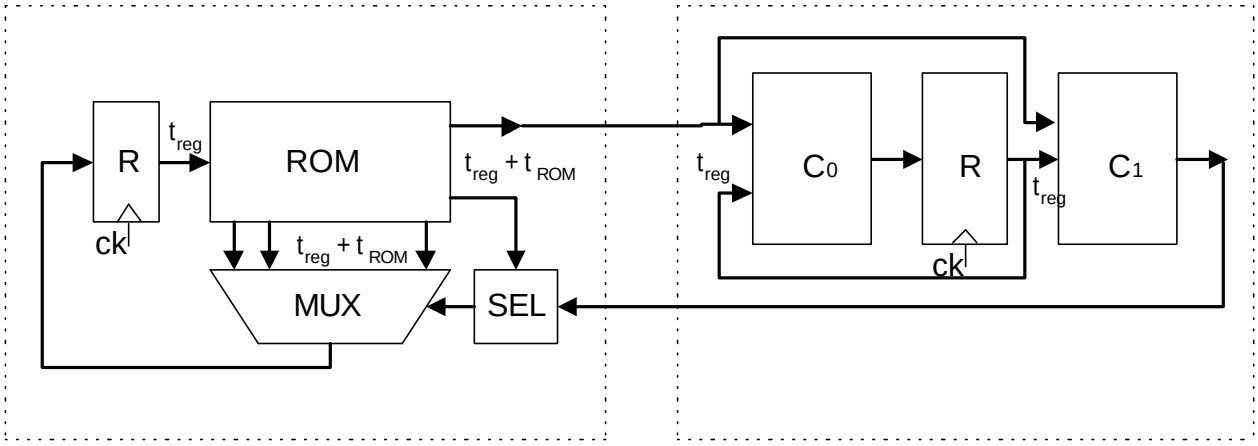
Il sistema interconnesso SCO-SCA specificato si ottiene mediante l'accoppiamento illustrato:



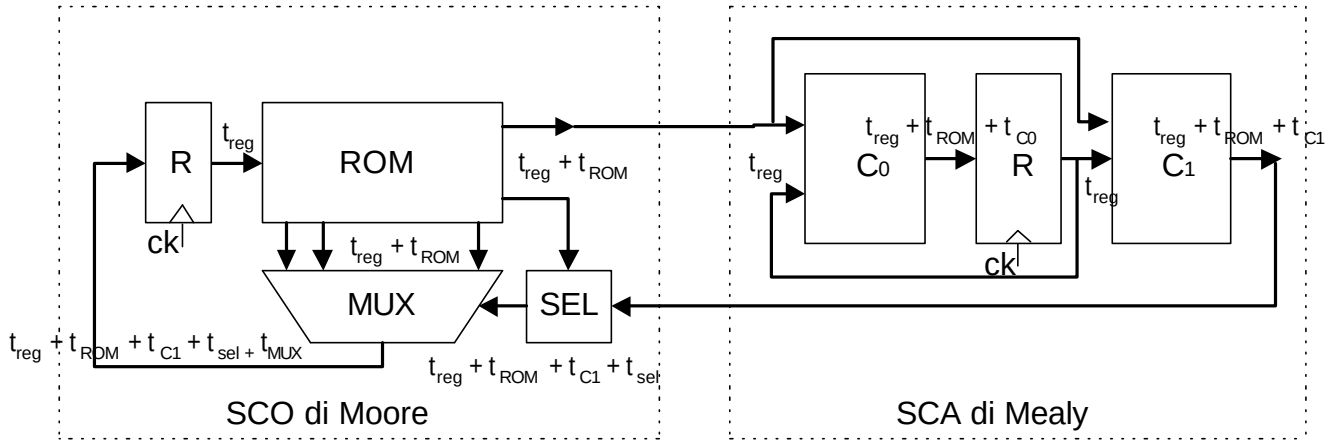
Per valutare la frequenza massima di lavoro (o il periodo minimo del segnale CK) del sistema in funzione dei parametri delle due reti, occorre determinare i tempi massimi in cui diventano stabili i segnali su tutti i nodi dei percorsi dei dati (i "cammini") che hanno origine sulle uscite dei registri e termine sugli ingressi dei registri, a partire dall'applicazione di un fronte del segnale CK; perciò le prime etichette che si possono determinare sono quelle all'uscita dei registri (t_{reg}):



Su questa base si possono ora aggiungere le etichette (i tempi massimi di stabilizzazione) delle uscite delle reti combinatorie alimentate dai registri (per comodità lo schema logico del sistema viene replicato):



Poi si prosegue con i percorsi combinatori in cascata:



Tutti i nodi della rete sono stati etichettati con i rispettivi tempi massimi di stabilizzazione; quindi si può determinare il massimo tra le due quantità etichettate sugli ingressi dei due registri:

$t_{reg} + t_{ROM} + t_{C0}$
$t_{reg} + t_{ROM} + t_{C1} + t_{sel} + t_{MUX}$

a cui occorre aggiungere il tempo t_{setup} di set-up dei registri; in definitiva si ottiene:

$T_{CK} > t_{reg} + t_{setup} + t_{ROM} + \max\{ t_{C1} + t_{sel} + t_{MUX}, t_{C0}\}$

Esercizio (2S20010126-D5)

Una periferica dotata di un clock di F MHz produce dati ad una velocità di F Mbit/s, che scarica nella RAM di un PD32 in DMA con modalità “stealing”. Determinare il rallentamento percentuale subito dal processore, nell’ipotesi che sia sincrono con la periferica ed esegua cicli macchina di 3 periodi di clock.

Si possono supporre tre casi, relativi agli accessi alla RAM per scaricare 8, 16 oppure 32 bit:

1. Accesso a byte:
la periferica impiega 8 periodi di clock per produrre un byte, che può scaricare in RAM in 3 periodi di clock all’interno dei successivi 8 periodi di clock (mentre produce e accumula il byte successivo); ne segue che la periferica occupa i bus di sistema per 3 periodi di clock su 8 e quindi il rallentamento subito dal processore vale:
R = 3/8 = 0.375 = 37.5%

2. Accesso a word:
la periferica impiega 16 periodi di clock per produrre una word, che può scaricare in RAM in 3 periodi di clock all’interno dei successivi 16 periodi di clock (mentre produce e accumula la word successiva); ne segue che la periferica occupa i bus di sistema per 3 periodi di clock su 16 e quindi il rallentamento subito dal processore vale:
R = 3/16 = 0.1875 = 18.75%

3. Accesso a longword:
la periferica impiega 32 periodi di clock per produrre una longword, che può scaricare in RAM in 3 periodi di clock all’interno dei successivi 32 periodi di clock (mentre produce e accumula la longword successiva); ne segue che la periferica occupa i bus di sistema per 3 periodi di clock su 32 e quindi il rallentamento subito dal processore vale:
R = 3/32 = 0.09375 = 9.375%

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 12-02-2001

STUDENTE:_____DOCENTE:_____

Si vuole progettare un co-processore grafico (ICON_PRO) dotato della funzione di riduzione delle dimensioni delle immagini memorizzate nella RAM del PD32.
Quando il micro vuole ridurre un'immagine invia a ICON_PRO:

- l'indirizzo iniziale dell'immagine originale, predisposta in memoria per righe;
- le dimensioni dell'immagine, entrambe espresse a 12 bit: numero dei pixel in orizzontale e in verticale;
- il fattore R di riduzione, espresso a 8 bit;
- l'indirizzo iniziale di destinazione dell'immagine ridotta.

In risposta ICON_PRO esegue le seguenti attività:

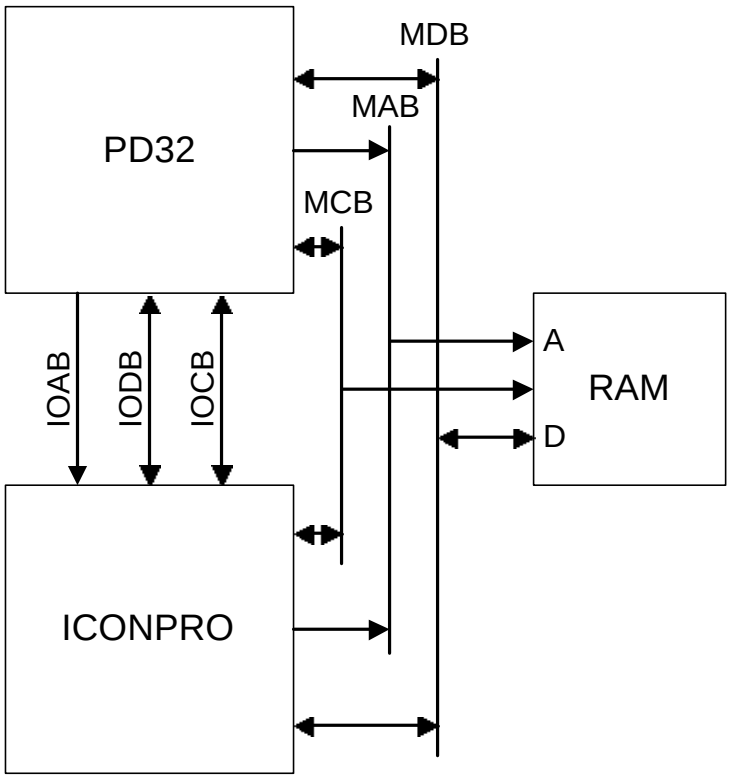
- accede alla RAM in DMA con modalità a burst, in corrispondenza di 1 riga ogni R righe dell'immagine, iniziando dalla prima riga;
- per ogni riga considerata copia nell'area RAM di destinazione 1 pixel ogni R pixel, iniziando dal primo pixel della riga;
- segnala il termine dell'elaborazione al processore con una interruzione.

I pixel sono a 8 bit.
ICON_PRO utilizza il clock del processore.

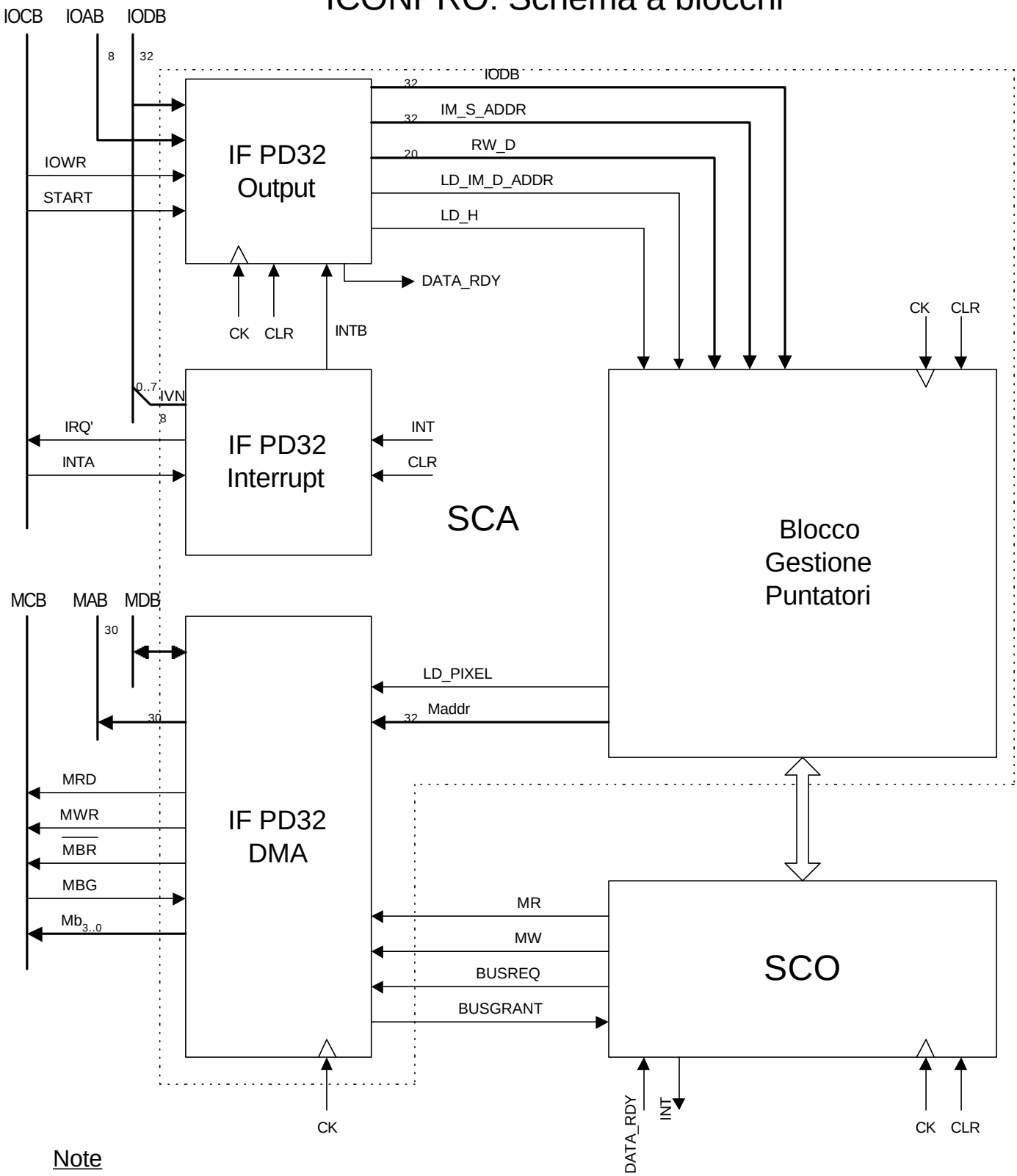
Si richiede:

1. la temporizzazione delle operazioni;
2. lo schema logico della periferica ICON_PRO;
3. le routine d'interfacciamento del PD32.

ICONPRO: sistema esterno



ICONPRO: Schema a blocchi

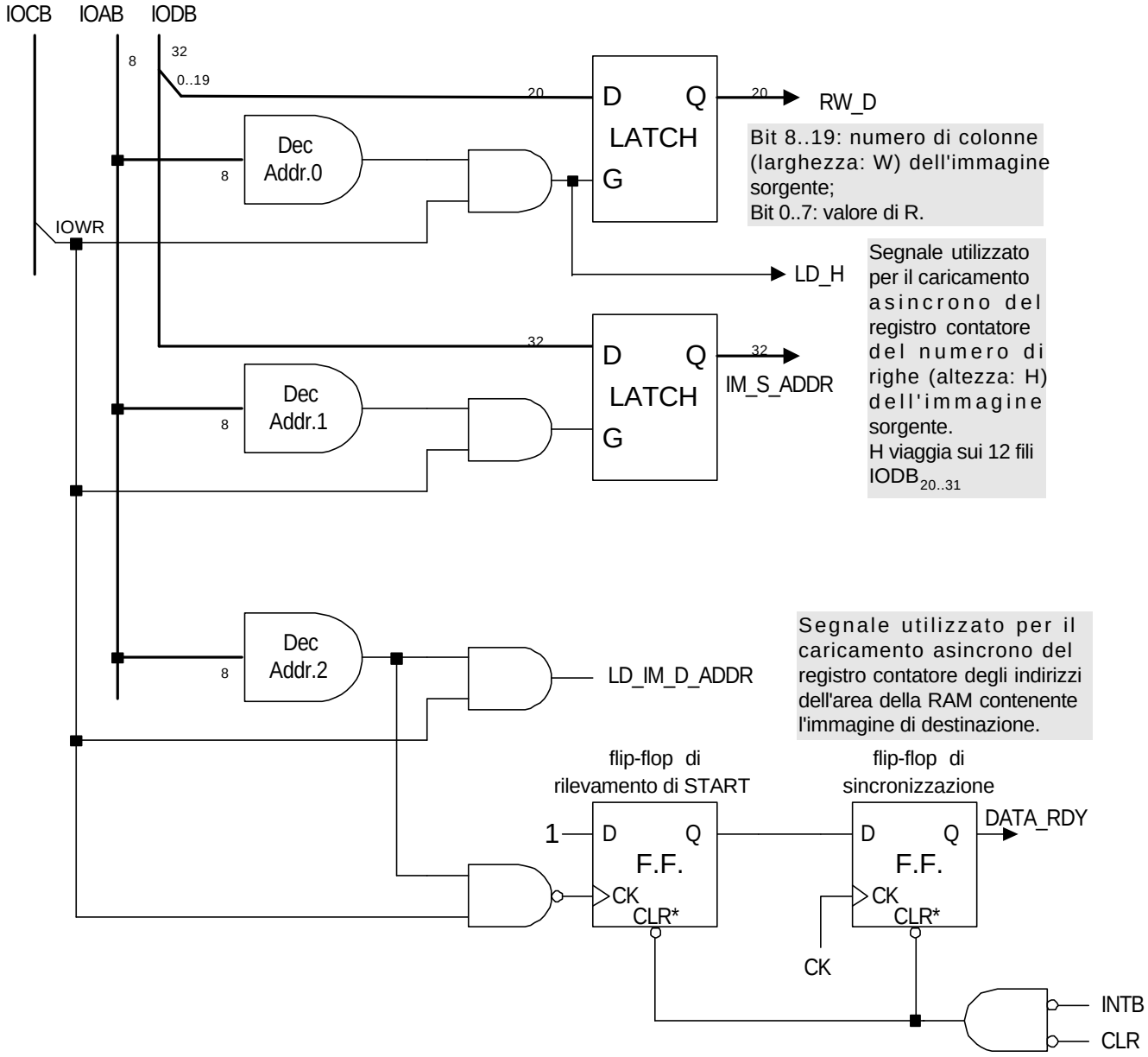


Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.

Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 interrupt usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

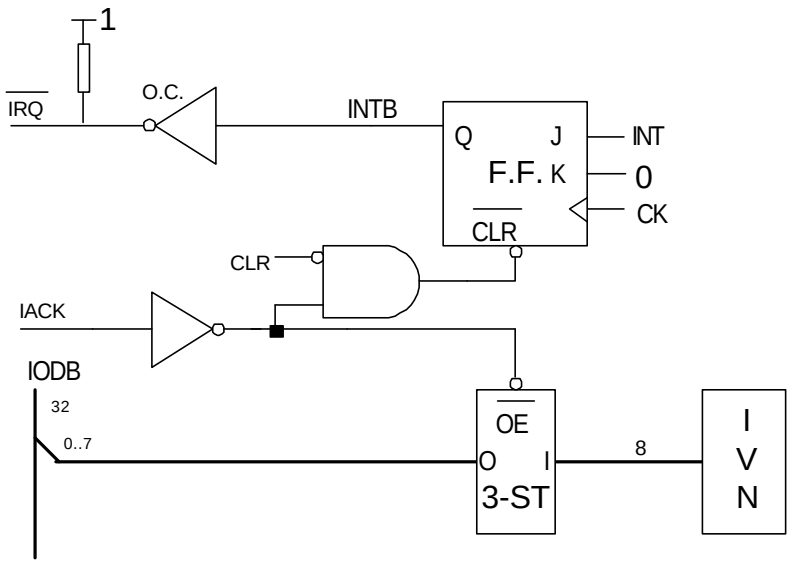
ICONPRO: IF PD32 - output



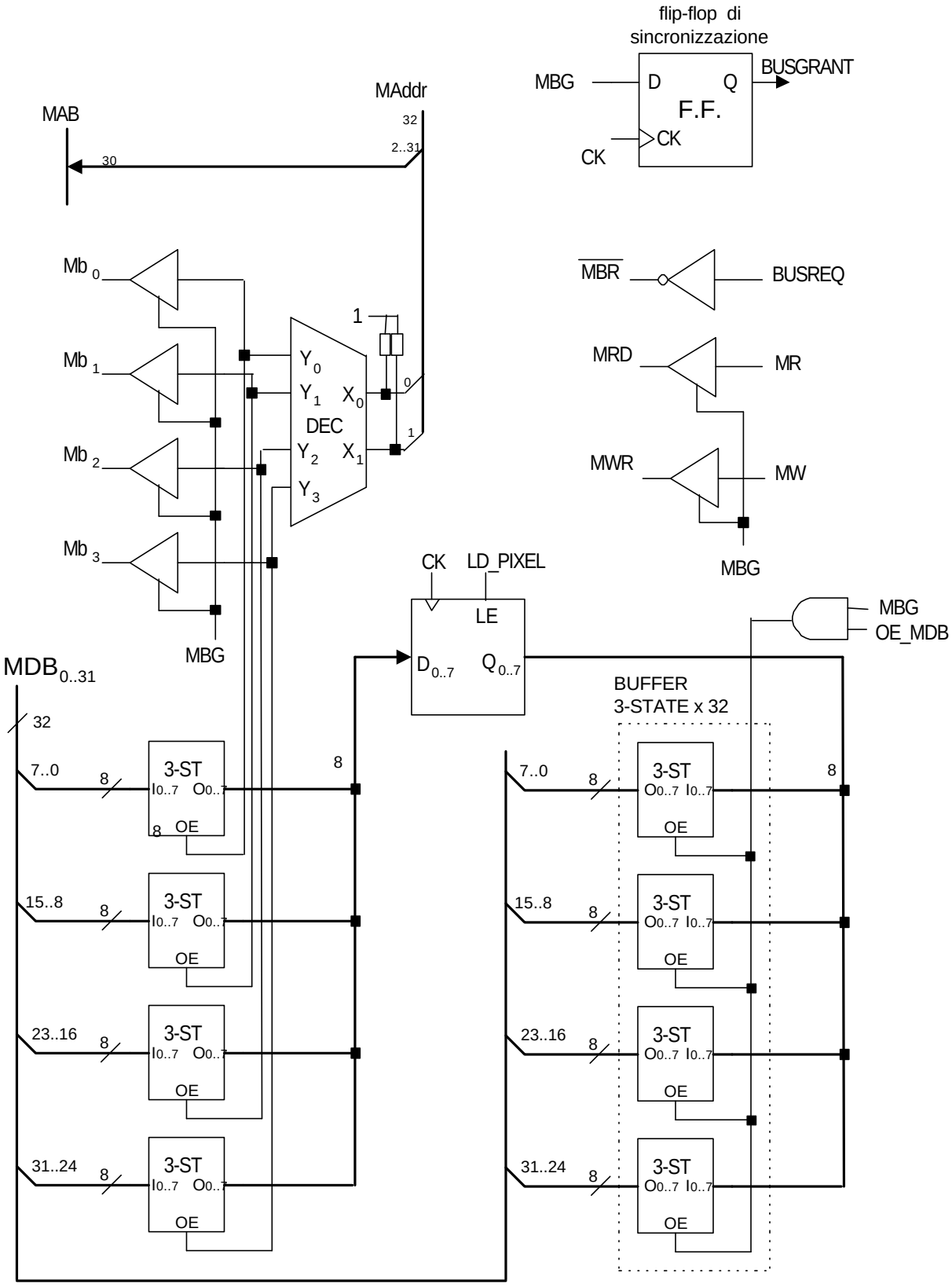
Note

In questa implementazione il SW avvia l'operazione direttamente con la scrittura dell'indirizzo dell'immagine di destinazione, dopo avere riscritto gli altri registri.

ICONPRO: IF PD32 - interrupt



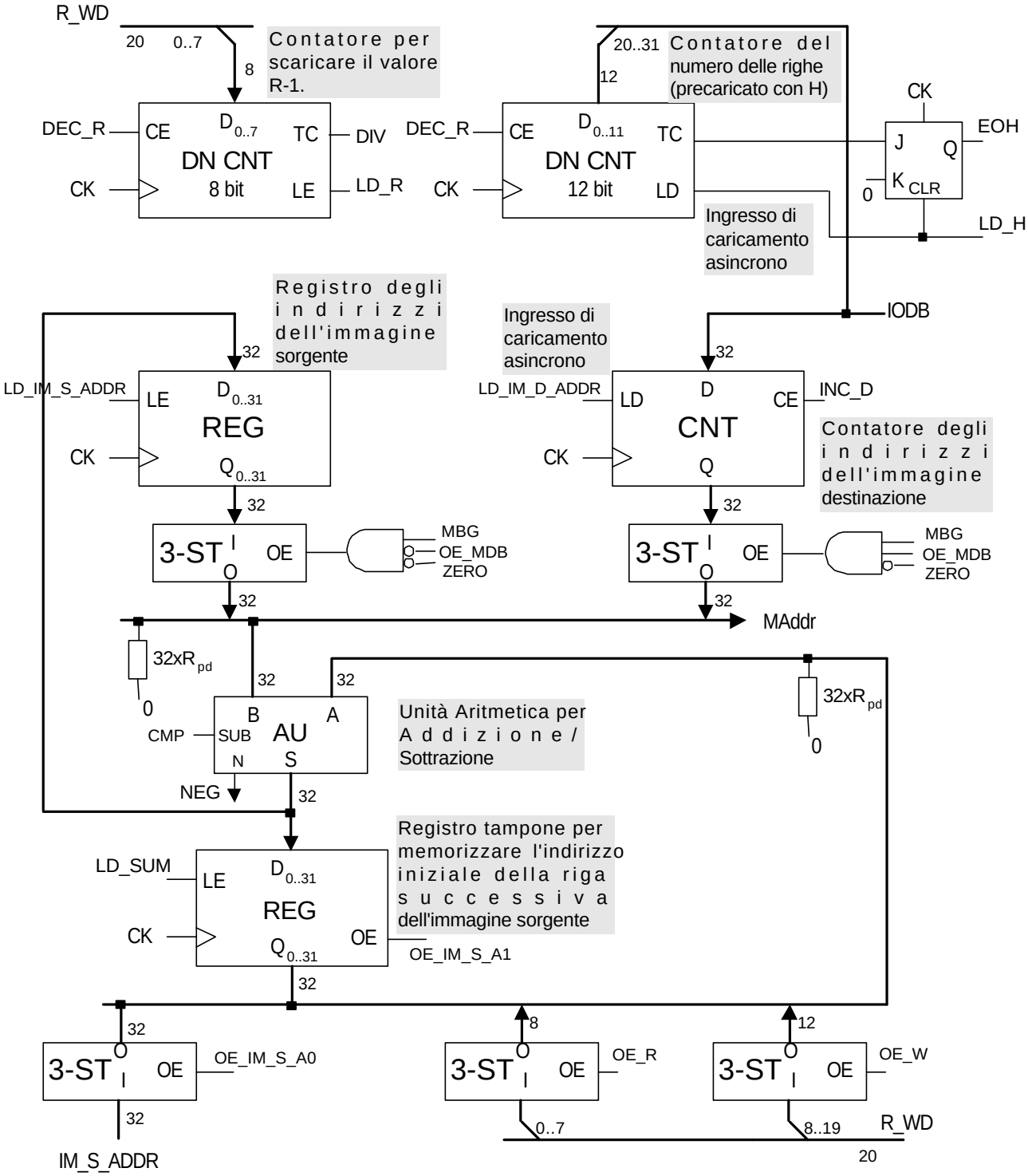
ICONPRO: IF PD32 - DMA



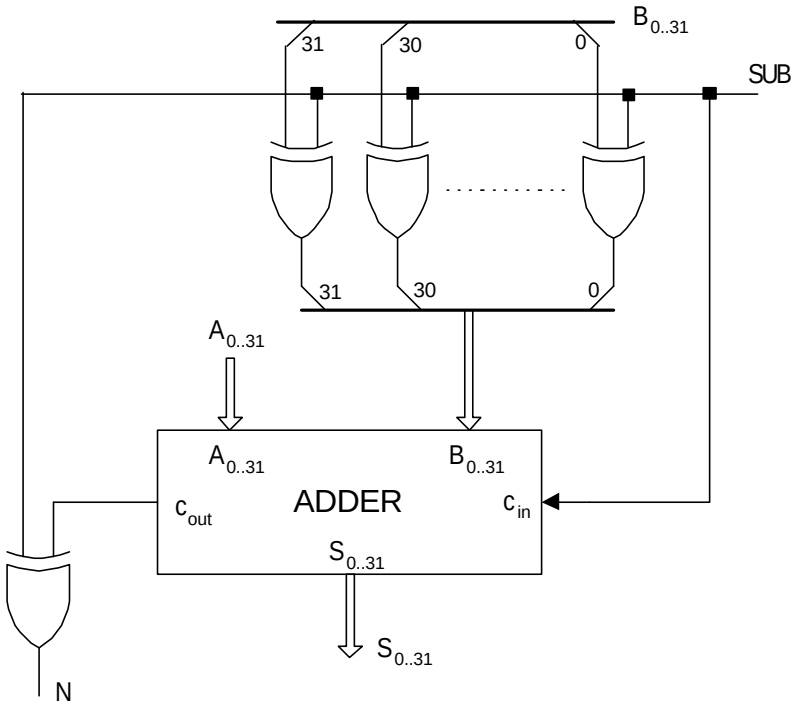
Blocco di riduzione dei 32 bit del MDB agli 8 bit del bus interno MDin (lettura).

Blocco di espansione degli 8 bit del bus interno MDout ai 32 bit del MDB (scrittura). L'abilitazione dei 3-state deve essere condizionata (cfr. porta AND su MBG) da un segnale di accesso in scrittura (OE_MDB), per evitare di interferire sul MDB negli accessi in lettura.

ICONPRO: blocco gestione puntatori



ICONPRO: Unità Aritmetica



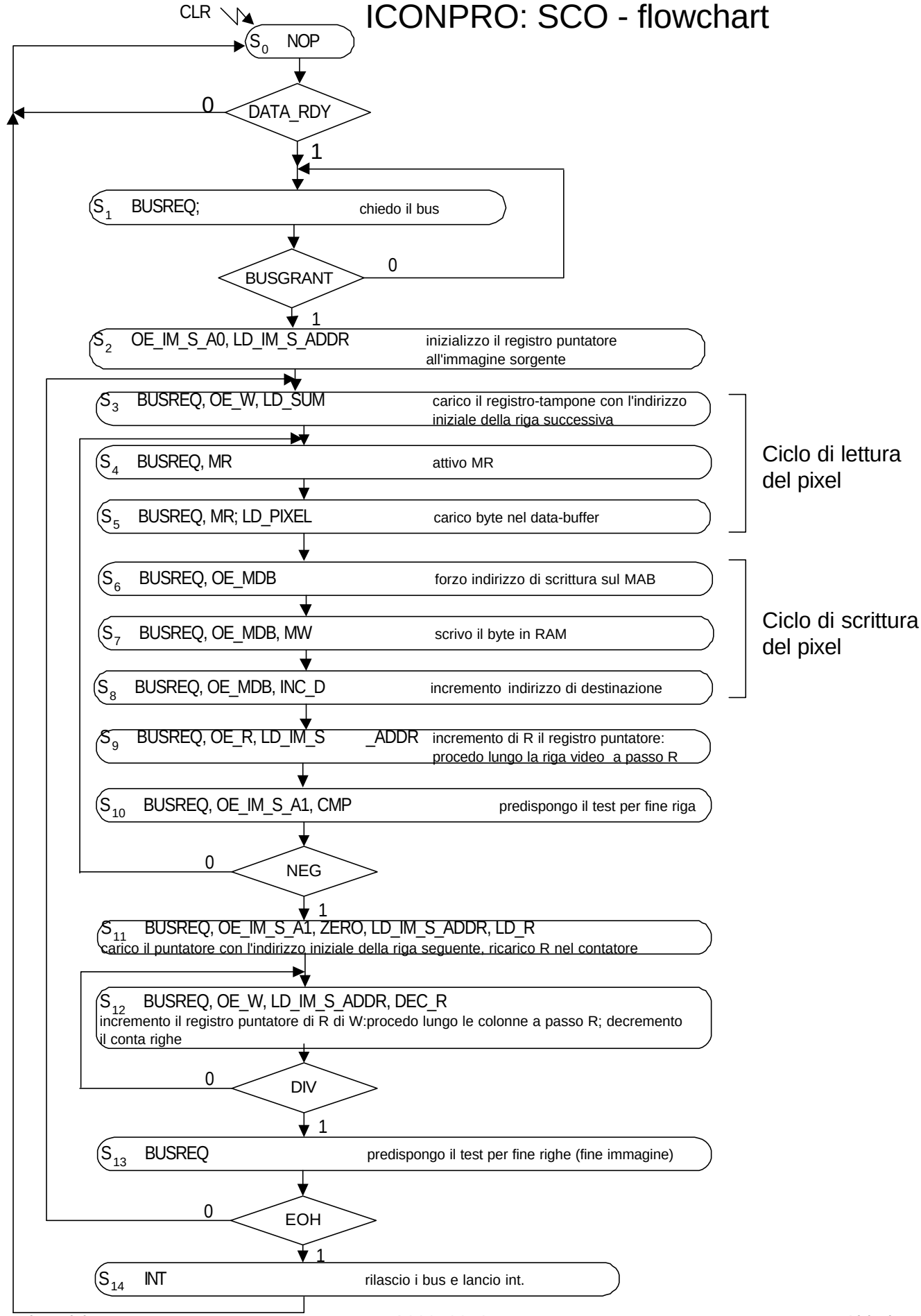
Note

- Esegue A-B quando SUB=1, esegue A+B quando SUB=0,
  come somma algebrica in complemento a 2. Infatti:
  SUB=0: S=A+B=A+B+SUB
  SUB=1: S=A-B=A+(2 ^{32}-B)=A+(2 ^{32}-1+1 -B)=A+(B*+1)=A+B*+SUB
  dove il simbolo * indica negazione bit a bit (complemento a 1).

- 0<=A,B<=2 ^{32}-1 da cui: -(2 ^{32}-1)<=A-B<=2 ^{32}-1 da cui la necessità di 33 bit (0..32) per la differenza.

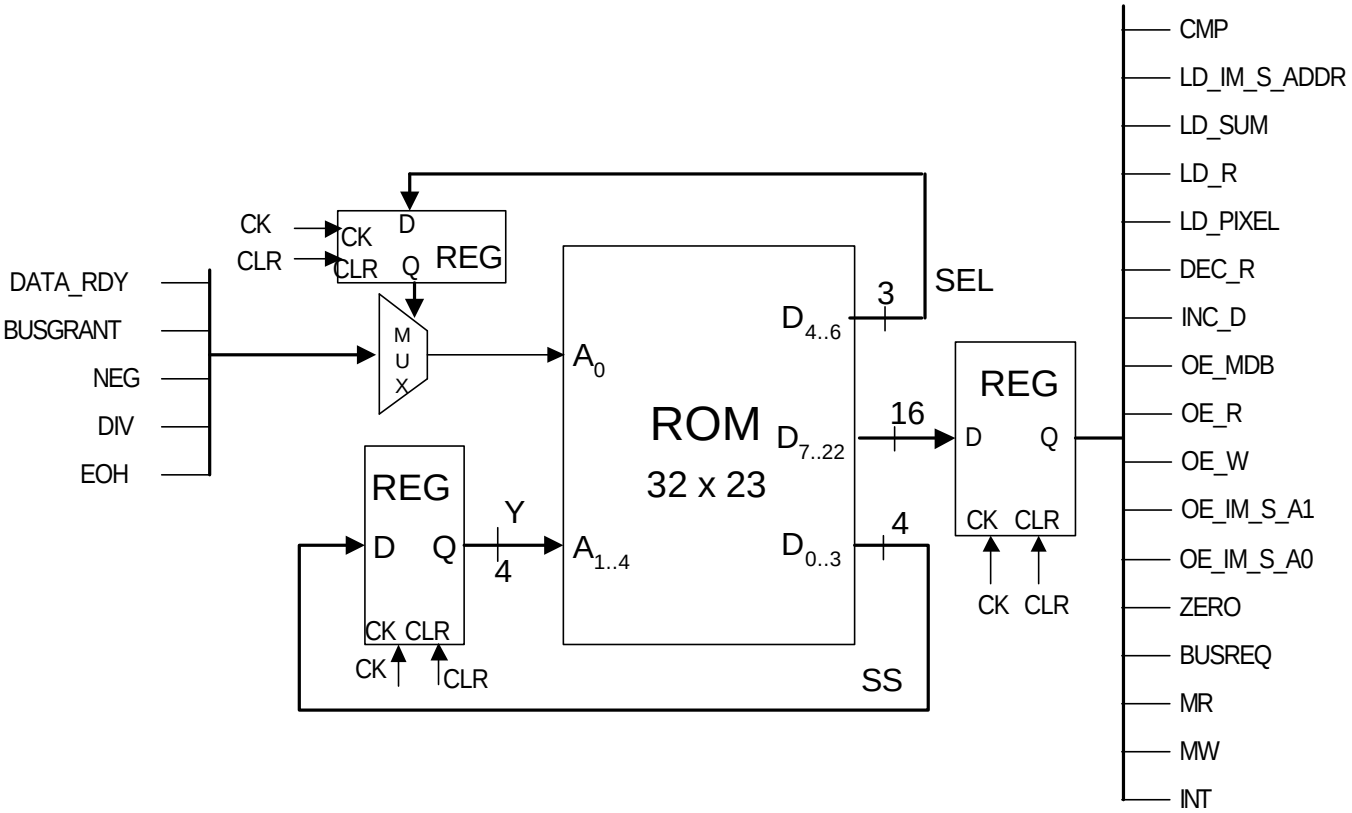
- Lo XOR calcola il 33-esimo bit di somma (A $_{32}$=0, B $_{32}$=SUB); la sua uscita rappresenta
  il segno della somma (algebrica).

ICONPRO: SCO - flowchart



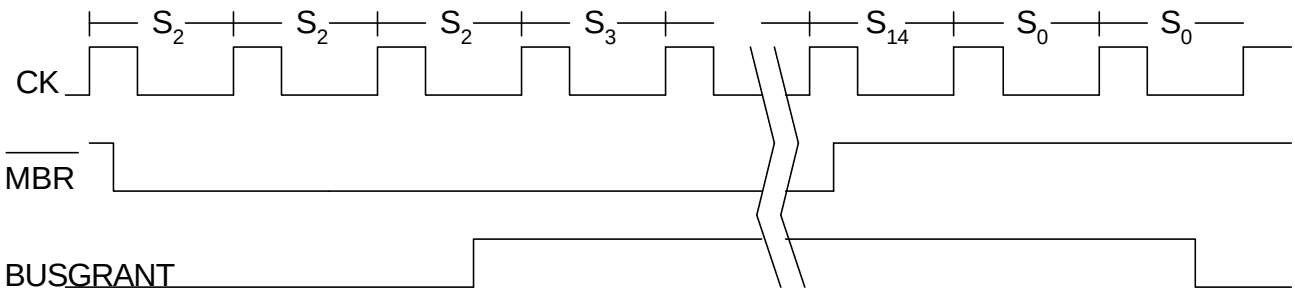
ICONPRO: SCO - struttura HW microprogrammata

Il flow-chart è supportato da un microlinguaggio di tipo 3; scegliendo il modello strutturale di tipo D-Mealy si ottiene la struttura seguente:

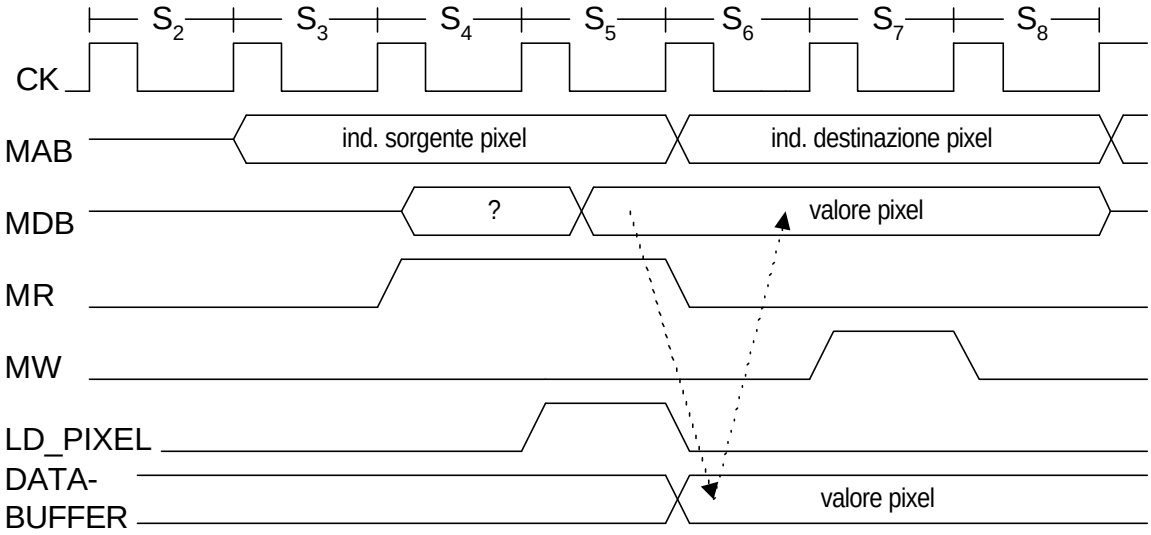


ICONPRO: temporizzazioni

Richiesta / rilascio bus di sistema e relativi stati dello SCO



Accessi in RAM e relativi stati dello SCO



I cicli di lettura sono a tre stati (S_3 .. S_5; S_6 .. S_8), in quanto si suppone che la RAM abbia $2T < t_A < 3T$: in mancanza di specifiche sulla velocità della periferica si suppone di utilizzare un segnale di clock a frequenza tale da poter emulare i cicli di memoria del processore.

ICONPRO: Routine Software

```
ORG 400h

S_IMAGE_ADDR DL 00FF0000h       ;indirizzo (32 bit) iniziale dell'immagine sorgente
D_IMAGE_ADDR DL 01000000h       ;indirizzo (32 bit) iniziale dell'immagine dest.
HWR  DL 02008010h               ;H-W-R
W       EQU 1024
H       EQU 512
R       EQU 16
Addr0 EQU 0A5h
Addr1 EQU 0A6h
Addr2 EQU 0A7h
FLAG DB 0
stack           equ 2800h   ;inizio area di stack
iconivn         equ 00Bh    ;ivn digimod
icondr          equ 1000h   ;indirizzo driver ICONPRO
…
CODE

movl #stack,r7              ;inizializzo R7 (SP)
seti
…
mainloop:
...
cmpb #0,FLAG                ;test su messaggio spedito
jz altro
jsr start_iconpro           ;richiesta di riduzione di un'immagine
                            ;(si suppone predisposta)
altro:
;predisposizione dell'immagine successiva
;altri task
jmp mainloop

; * * * R O U T I N E * * *

start_iconpro:
outl HWR,Addr0              ;caricamento parametri H, W, R
outl S_IMAGE_ADDR,Addr1     ;caricamento indirizzo iniziale immagine orig.
outl S_IMAGE_ADDR,Addr2     ;caricamento indirizzo iniziale imm. dest. e start
movb #00,FLAG               ;reset flag di immagine elaborata
ret

; * * * DRIVER * * *

driver iconivn,icondr
movb #01,FLAG               ;set flag di immagine elaborata
rti
```

Note

- La periferica preleva i byte in RAM lungo una riga dell'immagine con passo K; al raggiungimento (lunghezza della riga multipla di K) o superamento (lunghezza della riga non divisibile per K) dell'ultimo indirizzo della riga dell'immagine, il puntatore viene ricaricato con l'indirizzo iniziale della stessa riga (riavvolgimento a inizio riga); quindi viene incrementato della lunghezza della riga per K volte, determinando in questo modo un avanzamento del puntatore lungo la prima colonna, pur senza effettuare alcuna lettura; a questo punto la periferica entra nella riga successiva dell'immagine, ripercorre i passi descritti in precedenza. Il ciclo ha termine quando l'indirizzo iniziale della riga da visitare eccede quello massimo riservato all'immagine.

- Si può emettere la richiesta di interruzione quando è ancora BUSGRANT=1 perché INT viene memorizzata in un flip-flop e quindi sarà rilevata dal processore dopo avere ripreso l'uso dei bus (cfr. diagramma gestione DMA / INT).

- Dopo avere emesso INT in S_{14} il controllo torna in S_0 senza aspettare né INTA=1 né BUSGRANT=0: questi due eventi si verificheranno mentre SCO in S_0 aspetta la successiva commutazione di DATA_RDY=1; infatti, il processore potrà riportare DATA_RDY a 1 soltanto dopo avere servito la richiesta di interruzione, con la quale la periferica segnala al processore la conclusione delle attività.

- Va notato che il meccanismo di comunicazione tra il processore e la periferica mediante interruzione rende superflua l'interfaccia busy-waiting, in quanto dopo avere emesso una richiesta di interruzione la periferica ha certamente concluso l'operazione richiesta dal processore e quindi è certamente pronta ad avviare un'operazione successiva (il processore non ha bisogno di effettuare un test sullo stato di pronto della periferica).

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 12-02-2001

STUDENTE:_____DOCENTE:_____

D1 Progettare una rete iterativa per determinare se un vettore di n bit X_0 X_1...X_{n-1} è non-decrescente (per definizione lo è se $X_k \leq X_{k+1}$ k = 0..n-2; esempi sono i tre vettori 0001111, 0000000, 1111111).

D2 Risolvere lo stesso problema del punto D1 mediante una rete sequenziale dotata di una linea di ingresso su cui viene presentata serialmente la sequenza di bit X_0 X_1...X_{n-1}; su una seconda linea di ingresso viene segnalata la presenza di X_0 (inizio della sequenza).

D3 Descrivere un sommatore veloce a 16 bit segmentato in moduli di 4 bit.

D4 Dato lo SCA del PD32, scrivere un microprogramma per implementare l'istruzione ipotetica:
EXCH Ri,Rj
che scambia gli operandi nei due registri.

D5 Scrivere una routine in assembler PD32 per determinare se la word all'indirizzo VECT rappresenta un vettore non decrescente (cfr. definizione al punto D1); registrare il risultato all'indirizzo VECT+2.

Esercizio (2S20010212-D1)

Progettare una rete iterativa per determinare se un vettore di n bit $X_0\ X_1 ... X_{n-1}$ è non-decrescente (per definizione lo è se $X_k \leq X_{k+1}$ k = 0..n-2; esempi sono i tre vettori 0001111, 0000000, 1111111).

La cella generica nella posizione k-esima della rete iterativa può essere progettata in modo da rilevare la transizione 10 (violazione della non-decrescenza) tra i due bit di ingresso adiacenti X_{k-1} e X_k (fig. 1). L'esito del confronto locale Y_k (1: non-decrescenza) viene passato alla cella successiva se dalla cella precedente Y_{k-1}=1 (la sottorete a monte non ha rilevato violazioni), altrimenti viene propagato il valore 0 (è stata rilevata almeno una violazione) sino all'uscita della rete.

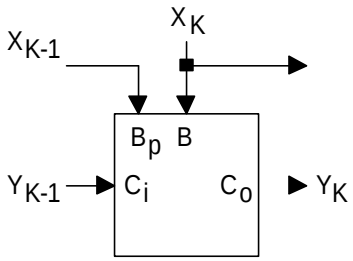


Fig. 1 - Cella della rete iterativa.

Consistentemente con le posizioni precedenti la struttura della rete iterativa (fig. 2) deve essere inizializzata con il carry-in della prima cella a 1 (inizializza la condizione di non decrescenza del vettore di ingresso).

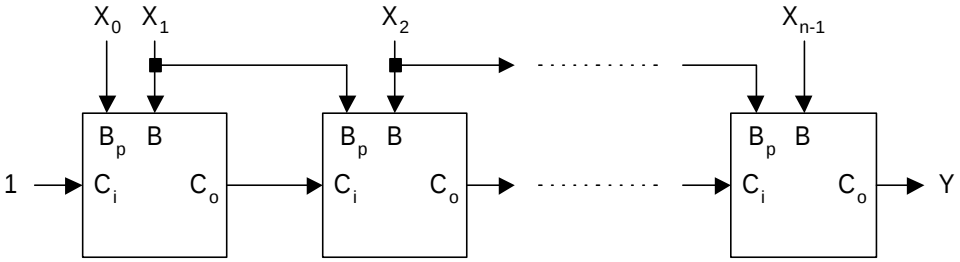


Fig. 2 - Struttura della rete iterativa.

Il progetto della cella è impostato con la tavola di verità che descrive l'unica uscita C_o in funzione delle variabili di ingresso C_i, B_p, B:

C_i	B_p	B	C_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Se C_i=0 (prime quattro righe della tavola) allora C_o=0;
altrimenti viene propagato a valle della cella l'esito del confronto tra B_p e B: in questo caso C_o=0 (violazione) solo per B_p=1 & B=0 (coppia decrescente).

La sintesi procede con la minimizzazione sulla MK:

B_pB / C_i	00	01	11	10
0				
1	1	1	1	

$C_o = C_i B_p^* + C_i B = C_i (B_p^* + B)$

dove * indica negazione.

Da cui segue la rappresentazione grafica della rete logica della cella (fig. 3):

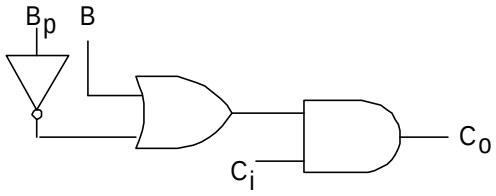
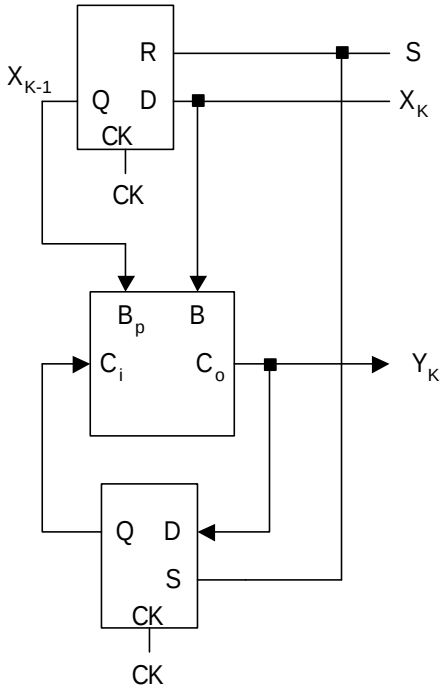


Fig. 3 – Struttura della cella.

Esercizio (2S20010212-D2)

Risolvere lo stesso problema del punto D1 mediante una rete sequenziale dotata di una linea di ingresso su cui viene presentata serialmente la sequenza di bit $X_0\ X_1 ... X_{n-1}$; su una seconda linea di ingresso viene segnalata la presenza di X_0 (inizio della sequenza).

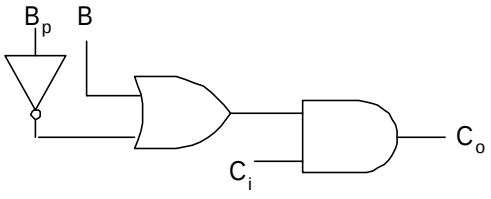
Si può utilizzare il progetto della rete iterativa definita al punto D1, se si osserva che la multiplazione spaziale della rete iterativa può essere trasformata in uno schema di multiplazione temporale:



Gli ingressi S e R dei flip-flop sono rispettivamente di set e reset di tipo sincrono. Se la libreria di componenti disponibile non li include, occorre costruirli con flip-flop D (con l'ausilio di una sola porta logica elementare, come mostrato in appositi esempi).

Il segnale di inizio sequenza S comporta il caricamento delle condizioni iniziali nei due flip-flop di stato, in modo sincrono e consistente con le posizioni dell'esercizio D1: 0 nel flip-flop di stato, 1 nel flip-flop di risultato. La rete sequenziale che ne risulta è di tipo Mealy, di cui si riconosce la struttura del modello generale (i due flip-flop sono il registro di stato). Va notato che quando è presente X_0 il flip-flop che lo ritarda deve essere già a 0 e analogamente il flip-flop di risultato deve essere già a 1; pertanto questo schema richiede un clock *che precede* X_0.

La rete combinatoria è quella dell'esercizio D1 (riportata a lato per comodità).



In alternativa all'approccio presentato sopra si può pervenire ad una rete della stessa complessità partendo da una descrizione della macchina sincrona, mediante il diagramma degli stati.

Esercizio (2S20010212-D3)

Descrivere un sommatore veloce a 16 bit segmentato in moduli di 4 bit.

Com'è noto dal testo "Reti Combinatorie", all'interno del sommatore a 4 bit i riporti sono calcolati a tempo costante mediante le equazioni:

$c_1 = G_1 + P_1\, c_{in}$
$c_2 = G_2 + P_2\, c_1 = G_2 + P_2(G_1 + P_1\, c_{in}) = G_2 + P_2G_1 + P_2P_1\, c_{in}$
$c_3 = G_3 + P_3\, c_2 = G_3 + P_3(G_2 + P_2G_1 + P_2P_1\, c_{in}) = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1\, c_{in}$
$c_4 = G_4 + P_4\, c_3 = G_4 + P_4(G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1\, c_{in}) =$
$= G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1 + P_4P_3P_2P_1\, c_{in}$

che si può riscrivere come:

$c_4 = G^* + P^* c_{in}$

essendo:

$G^* = G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1$

$P^* = P_4P_3P_2P_1$

ed avendo presupposto:

$P_i = a_i \oplus b_i$
$G_i = a_i\, b_i$

Il sommatore a 16 bit viene costruito con 4 sommatori a 4 bit; i riporti c_4, c_8, c_{12}, c_{16} in uscita dai quattro sommatori sono calcolati a tempo costante mediante un blocco di generazione dei riporti inter-gruppo secondo le equazioni precedenti, in cui i simboli G e P vengono sostituiti rispettivamente dai simboli G* e P*:

$c_4 = G_1^* + P_1^* c_{in}$
$c_8 = G_2^* + P_2^* G_1^* + P_2^* P_1^* c_{in}$
$c_{12} = G_3^* + P_3^* G_2^* + P_3^* P_2^* G_1^* + P_3^* P_2^* P_1^*\, c_{in}$
$c_{16} = G_4^* + P_4^* G_3^* + P_4^* P_3^* G_2^* + P_4^* P_3^* P_2^* G_1^* + P_4^* P_3^* P_2^* P_1^*\, c_{in}$

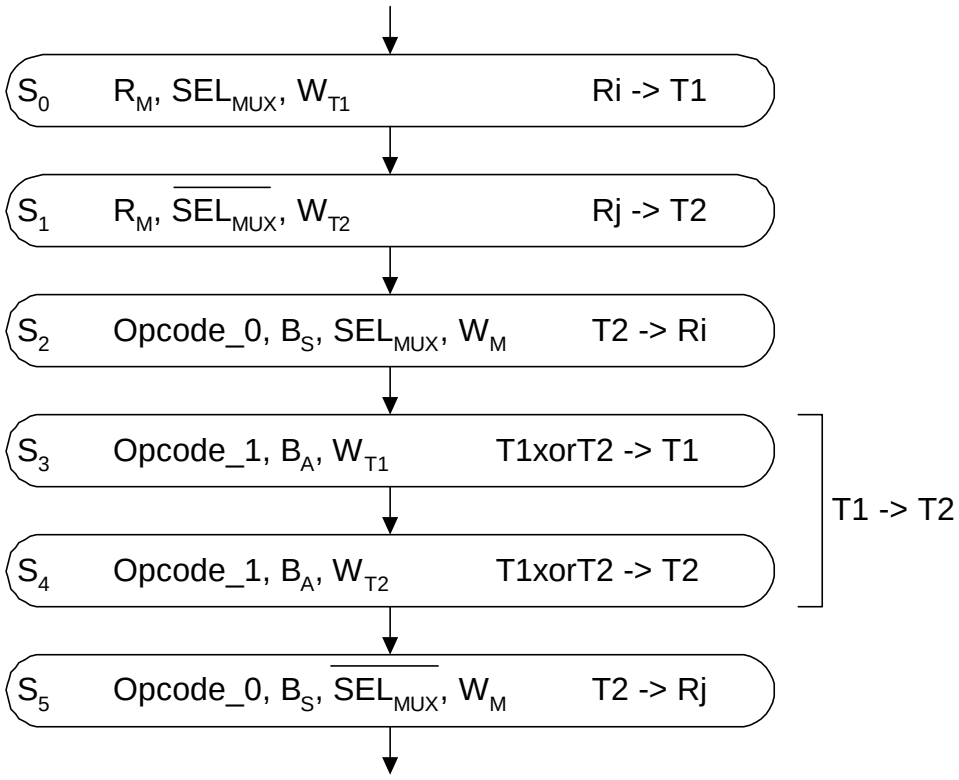
All'interno dei sommatori i bit di somma sono calcolati dalle espressioni:

$s_i = P_i \oplus c_{i-1}$ i=1..16.

Esercizio (2S20010212-D4)

Dato lo SCA del PD32, scrivere un microprogramma per implementare l'istruzione ipotetica:
EXCH Ri,Rj
che scambia gli operandi nei due registri.

Con riferimento allo schema dello SCA del PD32 e alle operazioni (4 aritmetiche + 4 logiche) definite per l'ALU, il frammento di microprogramma seguente impiega 6 stati.



NOTE

Opcode_0: trasparenza per lo shifter (shift di 0 posizioni)
Opcode_1: xor per l'ALU

La coppia di stati S3 e S4 serve a portare l'operando da T1 a T2, da cui può essere trasferito all'uscita dello shifter e quindi immesso nel bus interno.

Esercizio (2S20010212-D5)

Scrivere una routine in assembler PD32 per determinare se la word all'indirizzo VECT rappresenta un vettore non decrescente (cfr. definizione al punto D1); registrare il risultato all'indirizzo VECT+2.

```
;nondecre.asm

;determina se la word all'indirizzo VECT rappresenta un vettore
;non decrescente; registra il risultato all'indirizzo VECT+2.

        org 400h                ;inizio programma

VECT    dw 0FF00h

        code                    ;inizio istruzioni

main:
        jsr isnondec

        halt                    ;arresta l'elaborazione


; *************************
; SUBROUTINES
; *************************

isnondec:
        push r0                 ;salvo r0
        push r1                 ;salvo r1

        movw VECT,r0            ;copio il dato in r0
        movw r0,r1              ;e anche in r1
        asll #1,r1              ;traslo r1 di 1 bit a sx
        xorl #0FFFFFFFFh,r0     ;e calcolo f=r1 AND r0*
        andl r0,r1
        movb #1,r1
        jz exit                 ;1: sequenza OK
        movb #0,r1
exit:
        movw r1,VECT+2

        pop r1                  ;ripristino r1 e r0
        pop r0
        ret

        end                     ;fine programma
```

```
;nondecre.asm

;determina se la word all'indirizzo VECT rappresenta un vettore
;non decrescente; registra il risultato all'indirizzo VECT+2.


  org 400h    ;inizio programma

VECT  dw 0FF00h

  code      ;inizio istruzioni

main:
  jsr isnondec

  halt      ;arresta l'elaborazione


; ************************
; SUBROUTINES
; ************************

isnondec:
  push r0     ;salvo r0
  push r1     ;salvo r1

  movw VECT,r0    ;copio il dato in r0
  movw r0,r1    ;e anche in r1
  asll #1,r1    ;traslo r1 di 1 bit a sx
  xorl #0FFFFFFFFh,r0 ;e calcolo f=r1 AND r0*
  andl r0,r1
  movb #1,r1
  jz exit     ;1: sequenza OK
  movb #0,r1
exit:
  movw r1,VECT+2

  pop r1      ;ripristino r1 e r0
  pop r0
  ret

  end     ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 20-04-2001

STUDENTE:_____DOCENTE:_____

Si vuole progettare un mixer audio digitale controllato da un processore PD32, dotato della funzione di dissolvenza, che consente di sostituire un brano musicale con il successivo in modo graduale in un intervallo di tempo finito T programmabile, applicando un'attenuazione progressivamente crescente al primo e decrescente al secondo, fino alla completa sostituzione.
Il mixer può prelevare le sequenze dei campioni relativi a 4 brani musicali digitalizzati da altrettante linee di ingresso a 16 bit, sincronizzate esternamente alla velocità di 40 Kcampioni/s da un segnale CK disponibile in ingresso alla periferica.
Durante la fase di dissolvenza il mixer produce i campioni z_n del segnale di uscita miscelando i campioni x_n del segnale corrente e quelli y_n del segnale successivo secondo la combinazione lineare:

$$z_n = a_k\, x_n + b_k\, y_n \qquad \text{con } 0 \le a_k,\, b_k \le 1;\; a_k + b_k = 1$$

applicando in sequenza ciascuna coppia di coefficienti a_k, b_k con k=0..127 a un numero N programmabile di campioni consecutivi. Pertanto il parametro N controlla la durata T della dissolvenza secondo la relazione: T = 128 N (N e T espressi in numero di campioni).

Le 128 coppie di coefficienti a_k, b_k, espressi a 8 bit, si trovano memorizzate in una ROM di 128 x 16 bit.
Tutti i possibili prodotti, a 16 bit, sono memorizzati in una seconda ROM di 16 Mword (cioè 2^{16+8} x 16 bit).

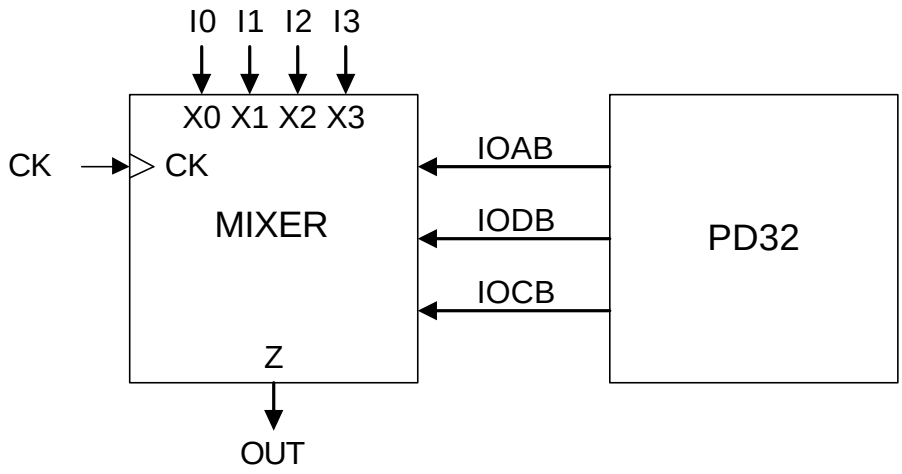
Quando il micro vuole sostituire un brano musicale invia al mixer:
- Il codice della porta da cui va prelevato il segnale del brano successivo;
- Il numero N dei campioni che devono essere miscelati con la stessa coppia di coefficienti;
- un segnale di start per avviare l'operazione.

In risposta il mixer:
- seleziona il canale indicato dal micro;
- nella fase di dissolvenza per ognuna delle 128 coppie di coefficienti a_k, b_k, con k=0..127, produce una sequenza di N campioni di uscita, secondo la relazione lineare specificata;
- al termine della fase di dissolvenza connette il canale selezionato all'uscita: $z_n=y_n$.

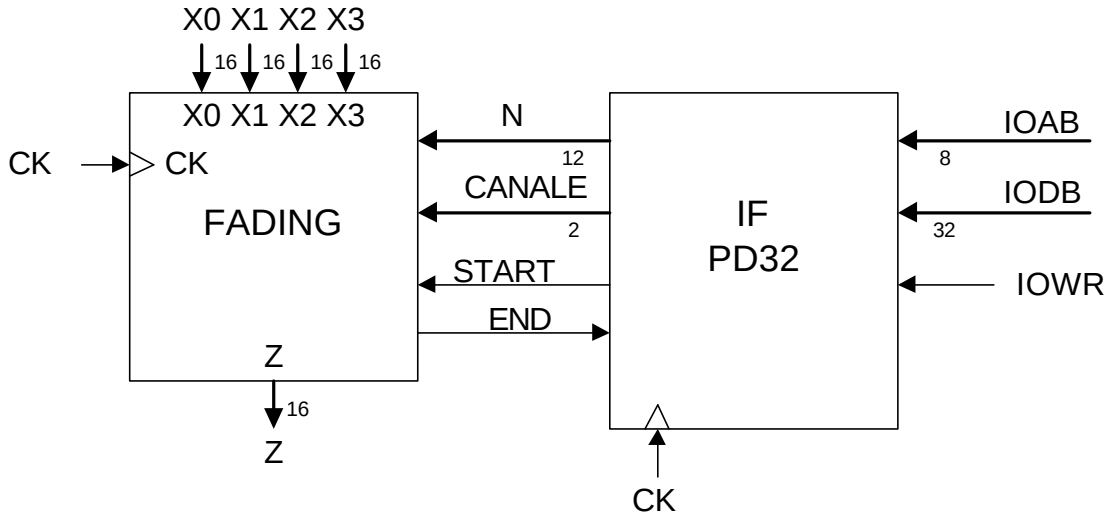
Si richiede:

1. la temporizzazione delle operazioni;
2. lo schema logico della periferica;
3. la routine d'interfacciamento del PD32.

MIXER: sistema esterno



MIXER: schema a blocchi



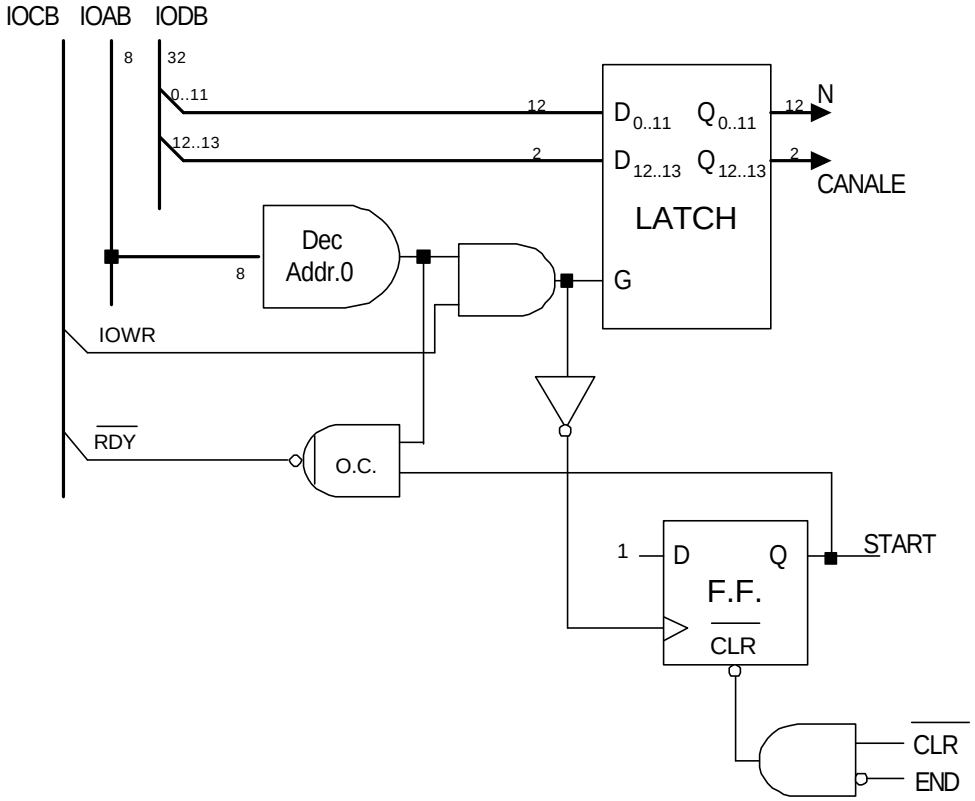
Note

Periferica di tipo output.

La periferica elabora i segnali X0..X3 - a 16 bit - in tempo reale utilizzando come clock il segnale di sincronizzazione (CK) dei campioni esterno.

N è dimensionato per consentire una durata massima della dissolvenza di 10 sec: 40000 x 10 / 128 = 3125 < 2^{12}; con 12 bit la dissolvenza è limitata a oltre 13 sec.

MIXER: IF PD32



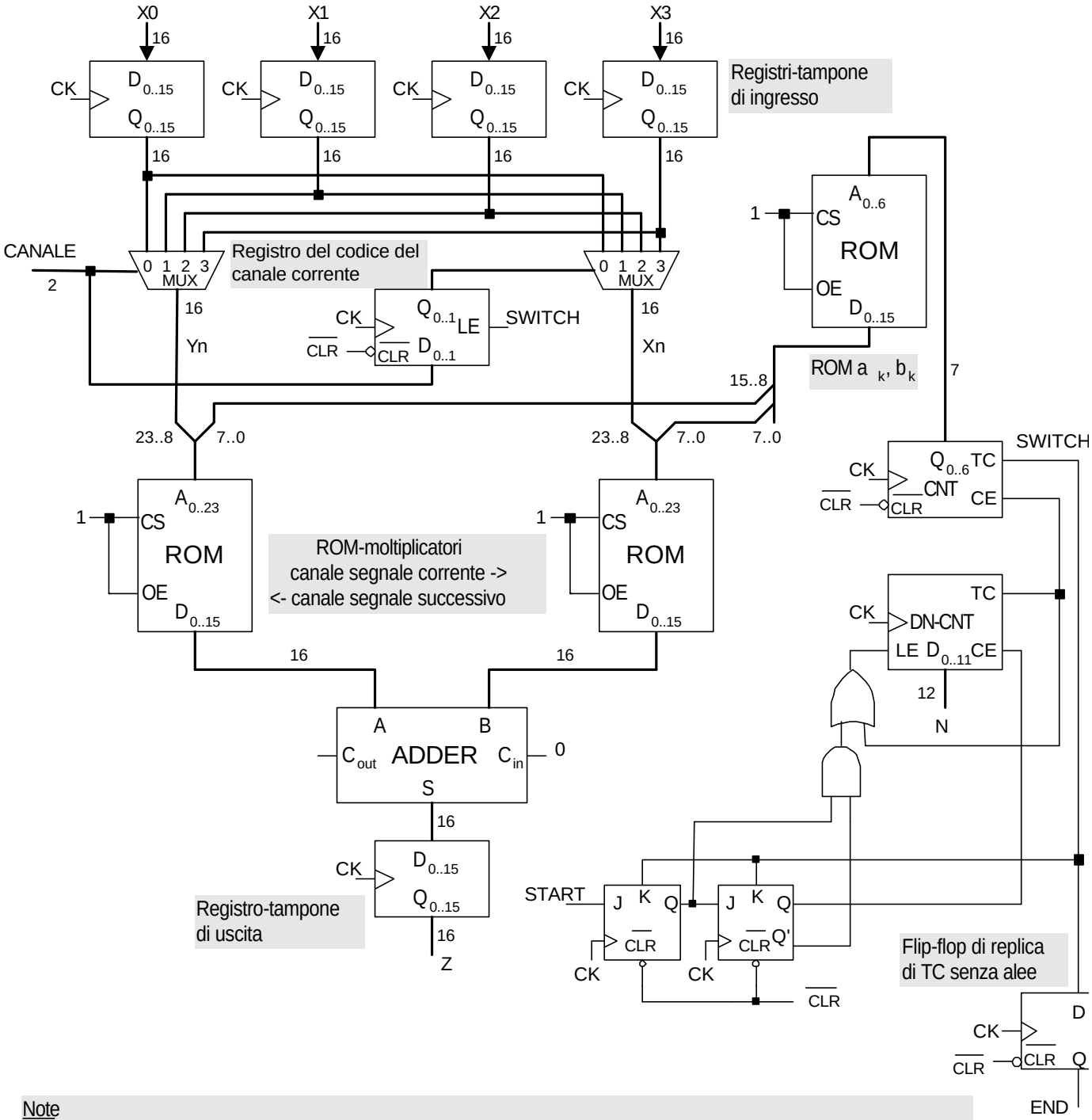
Codice assembler

```
ORG 400h
MIXER_ADDR EQU 80h ;(Addr0 nello schema)
. . .
MIXER_DATA DW 2800h ; prossimo canale: il 2; durata del fading:  circa 6.5 sec.
. . .
CODE
main:
. . .
stay:
jnr MIXER_ADDR,stay
jsr mixer
. . .
. . .
;***********
; SUBROUTINE
;***********
mixer:
outw MIXER_DATA,MIXER_ADDR
ret
```

Note

L'interfaccia di uscita include un solo latch (data buffer) caricabile dal micro con la coppia di dati N (12 bit) e CANALE (2 bit); le uscite 0..11 del latch sono collegate agli ingressi di un contatore dello SCA, che deve essere ricaricato periodicamente .

MIXER: blocco "fading"

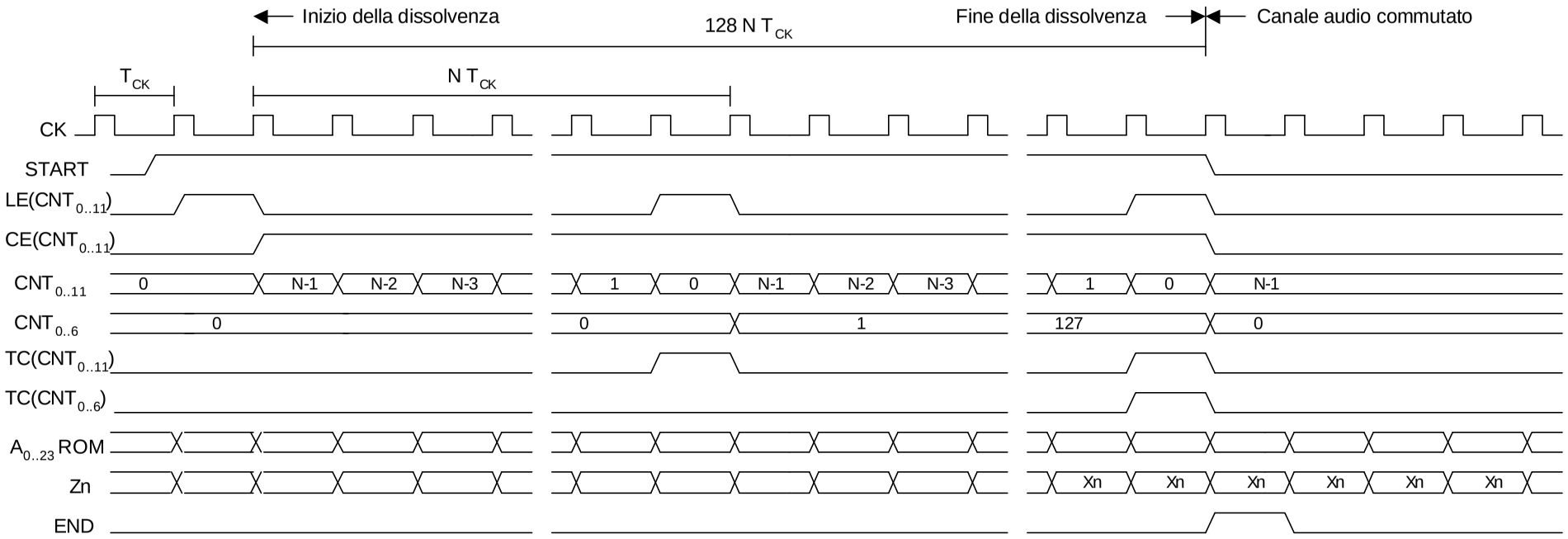


Note

- In ogni singolo ciclo di CK viene elaborato un campione di uno (a regime) o due (fase di dissolvenza) dei segnali di ingresso, e prodotto un campione del segnale di uscita.
  La durata del ciclo di CK, 25 microsec., consente l'elaborazione in cascata dei MUX, delle ROM e dell'addizionatore senza l'interposizione di registri di pipeline.

- Al termine della dissolvenza il canale Y diventa il canale X per effetto delle due azioni:
  1- il registro sui selettori del MUX di ingresso del canale X viene caricato con il codice del canale Y;
  2- in ingresso alle due ROM-moltiplicatore viene presentata la coppia di coefficienti (memorizzati all'indirizzo 0 della ROM 128x16):
  a_0=1 (canale X), b_0=0 (canale Y), che riproduce la sequenza X_n in uscita dal sommatore, e quindi sul canale di uscita.

MIXER: temporizzazioni



Dal diagramma temporale è evidente che il contatore a 12 bit conta mod N se viene caricato con il valore N-1; a questo provvederà direttamente il microprocessore (chi scrive il SW deve esserne informato dal manuale della periferica).

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 20-4-2001

STUDENTE:_____DOCENTE:_____

D1. Rappresentare il numero **-1.25** nei formati:
- virgola fissa nella notazione in modulo e segno;
- virgola fissa nella notazione in complemento a 2;
- virgola mobile.

D2. Progettare una rete combinatoria dotata di due ingressi X_0 e X_1, una uscita Y, e di due ingressi C_0 e C_1 che ne programmino il funzionamento (porta logica universale) secondo la tabella seguente:

C_1	C_0	Y
0	0	X_0X_1
0	1	X_0+X_1
1	0	$(X_0X_1)'$
1	1	$(X_0+X_1)'$

D3. Dato un segnale di clock alla frequenza di 32,768 KHz, definire una catena di conteggio che produca su due linee di uscita un impulso ogni secondo e ogni minuto (primo) rispettivamente.

D4. Descrivere la struttura dettagliata del banco dei registri R0..R7 del PD32.

D5. Indicare quali segnali del bus di controllo del PD32 sono scambiati (*hand-shaking*) con un protocollo di comunicazione strutturato in più fasi e descriverne le relative interfacce in un sistema SCA-SCO esterno.

Esercizio (2S20010420-D1)

Rappresentare il numero **-1.25** nei formati:
- virgola fissa nella notazione in modulo e segno;
- virgola fissa nella notazione in complemento a 2;
- virgola mobile.

---

1 - Virgola fissa nella notazione in modulo e segno:

$0.25_{10} = 0.01_2$ cioè sono necessari almeno 2 bit a destra della virgola per rappresentare la componente frazionaria del numero specificato;
per la parte intera è sufficiente 1 bit di cifra, a cui va aggiunto 1 bit di segno.
Scegliendo ad esempio un formato del tipo:
SIIIIIII.DDDD (bit di segno, 7 bit di parte intera, 4 bit di parte decimale), si ha:
$-1.25_{10} = 10000001.0100_2$

2 - Virgola fissa nella notazione in complemento a 2:

Scegliendo un formato del tipo:
IIIIIIII.DDDD (8 bit di parte intera, 4 bit di parte decimale), si ha:

$1.25_{10} = 00000001.0100_2$

da cui, complementando a 2, si ottiene:

$-1.25_{10} = 11111110.1100_2$

3 - Virgola mobile:

Scegliendo un formato a 32 bit del tipo:
bit di segno, 8 bit di esponente, 23 bit di mantissa, si ha:

$-1.25_{10} = -0.625\ 2^1 = 1\ 00000001\ 10100000000000000000000_2$

Esercizio (2S20010420-D2)

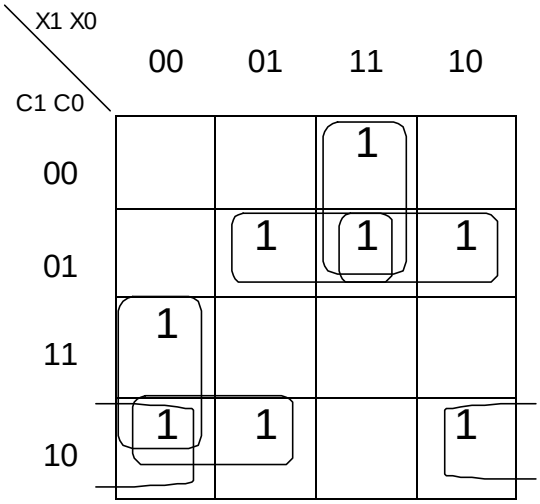
Progettare una rete combinatoria dotata di due ingressi X_0 e X_1, una uscita Y, e di due ingressi C_0 e C_1 che ne programmino il funzionamento (porta logica universale) secondo la tabella seguente:

C_1	C_0	Y
0	0	X_0X_1
0	1	X_0+X_1
1	0	$(X_0X_1)'$
1	1	$(X_0+X_1)'$

Il progetto della rete richiesta è avviato con la trascrizione formale della specifica; in questo caso tale operazione si risolve nell'espansione della tavola di verità specificata in modo da eliminare i simboli delle variabili X_0, X_1 dalla colonna Y:

funzione	C_1	C_0	X1	X0	Y
AND	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	0
	0	0	1	1	1
OR	0	1	0	0	0
	0	1	0	1	1
	0	1	1	0	1
	0	1	1	1	1
NAND	1	0	0	0	1
	1	0	0	1	1
	1	0	1	0	1
	1	0	1	1	0
NOR	1	1	0	0	1
	1	1	0	1	0
	1	1	1	0	0
	1	1	1	1	0

A questo punto si effettua la minimizzazione su una MK:



Scegliendo di sintetizzare la rete nella forma AND-OR, si ottiene l'espressione SP:

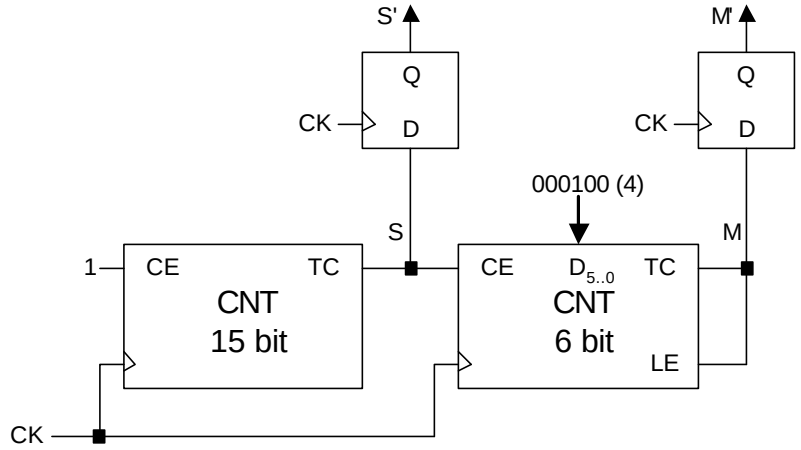
Y = C1'X1X0 + C1X1'X0' +
+ C1'C0X1 + C1C0'X0' +
+ C1'C0X0 + C1C0'X0'

A questo punto può essere facilmente tracciata la rete combinatoria a due livelli (più il terzo di inversione) corrispondente all'espressione minima trovata.

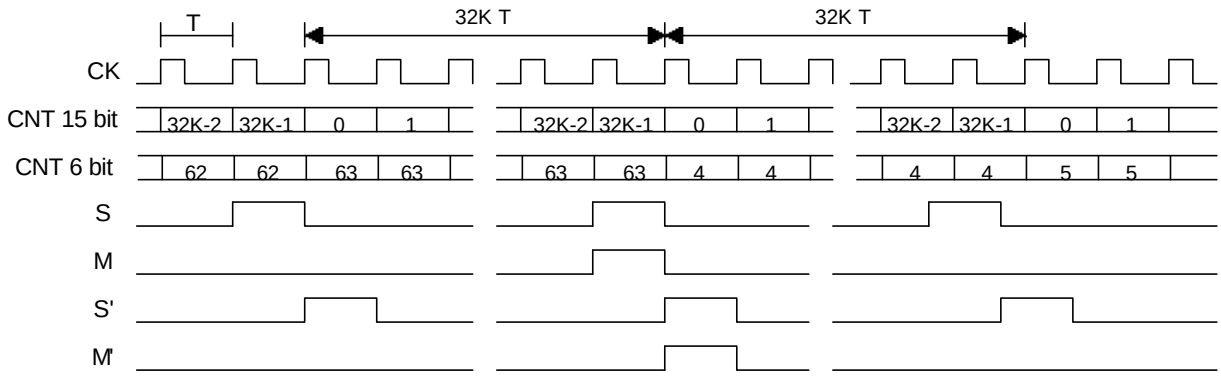
Esercizio (2S20010420-D3)

Dato un segnale di clock alla frequenza di 32,768 KHz, definire una catena di conteggio che produca su due linee di uscita un impulso ogni secondo e ogni minuto (primo) rispettivamente.

La frequenza Fck = 32768 Hz = 2^{15} Hz può essere divisa mediante un contatore a 15 bit, che produrrà sull'uscita TC un impulso di durata pari a Tck ogni secondo. Per contare i secondi in un minuto si potrà interconnettere un secondo contatore mod 60 in cascata al primo, come mostrato nella figura seguente:

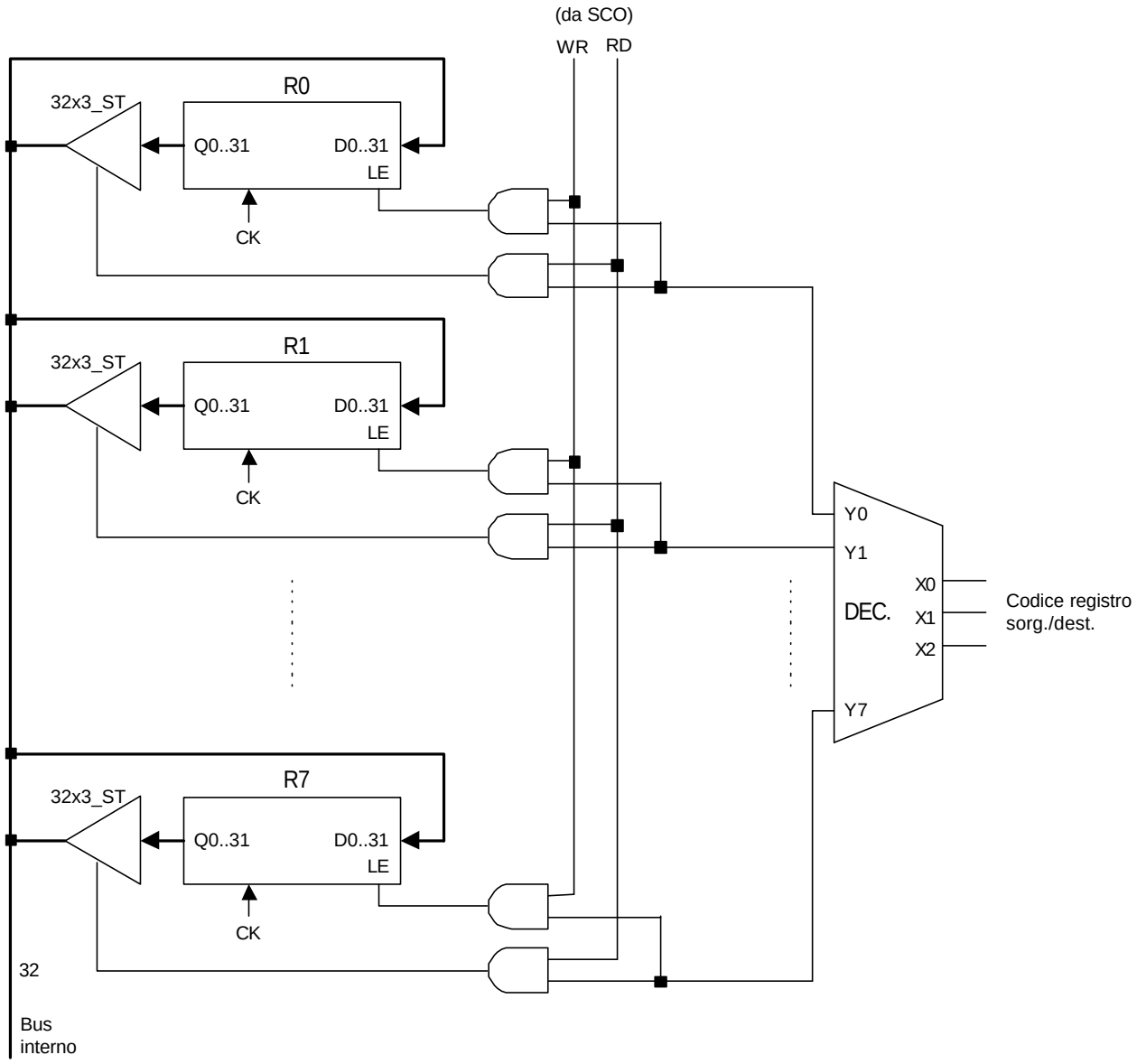


Il contatore a 15 bit si aggiorna in evoluzione libera (*free-running*) e percorre cicli di scansione di durata 32 K cicli di CK, al termine dei quali si azzera per rotolamento. Il secondo contatore invece deve essere inizializzato esplicitamente, in quanto deve contare mod 60 e non mod 64; la soluzione proposta utilizza il caricamento del conteggio iniziale 4, comandato dal segnale di fine conteggio TC.
I due segnali di trabocco, S e M, sono anche risincronizzati per produrre due segnali privi di alee utilizzabili da dispositivi esterni; ovviamente se i due impulsi dovessero servire soltanto per uso interno al dispositivo sincrono, non ci sarebbe bisogno di ricampionarli.
Il funzionamento dell'orologio è descritto dal diagramma di temporizzazione seguente:



Esercizio (2S20010420-D4)

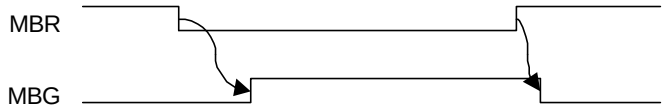
Descrivere la struttura dettagliata del banco dei registri R0..R7 del PD32.



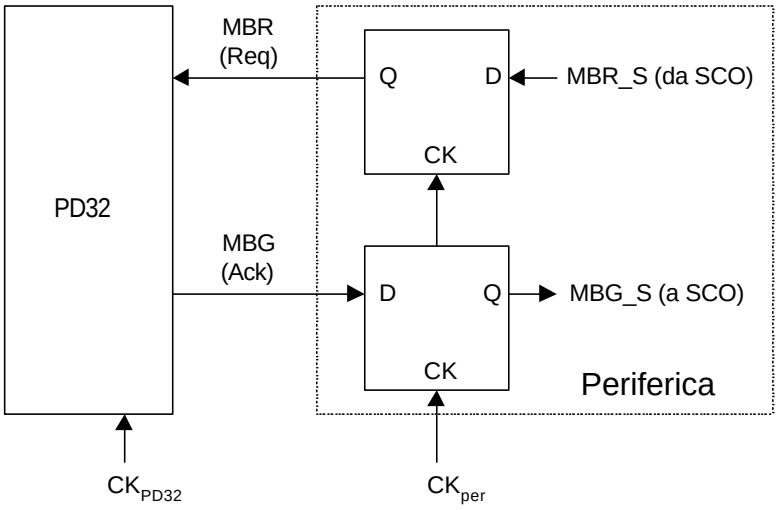
Esercizio (2S20010420-D5)

Indicare quali segnali del bus di controllo del PD32 sono scambiati (*hand-shaking*) con un protocollo di comunicazione strutturato in più fasi e descriverne le relative interfacce in un sistema SCA-SCO esterno.

La coppia di segnali MBR – MBG è gestita dal processore a quattro fasi, in cooperazione con una periferica, secondo il diagramma temporale seguente:



Com'è noto, questo protocollo è supportato dall'interfaccia hardware riportata nella figura seguente, relativa al lato periferica:



RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 6-6-2001

STUDENTE:_____DOCENTE:_____

Si vuole progettare un co-processore matematico (STAT_PRO) specializzato nel calcolo delle funzioni statistiche istogramma e valor medio di un blocco di dati a 8 bit predisposti nella RAM di un processore PD32.
Quando il micro vuole avvalersi di STAT_PRO gli invia:

- l'indirizzo iniziale di un blocco di 64Kbyte da elaborare;
- l'indirizzo iniziale dell'area di destinazione dell'istogramma, di tipo XXXX0000h;
- un comando di avvio dell'elaborazione.

In risposta STAT_PRO esegue le seguenti attività:

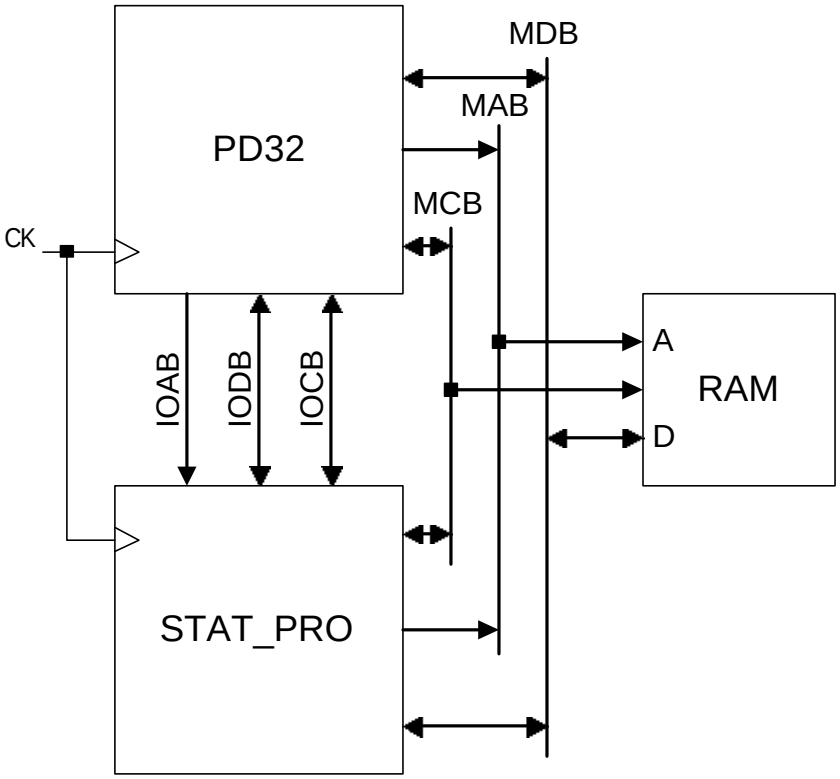
- accede alla RAM in DMA con modalità "burst";
- inizializza (a 0) l'area di RAM riservata all'istogramma;
- calcola l'istogramma (numero delle occorrenze di ciascun valore) dei dati e lo memorizza in un buffer di memoria di 256 (i dati da elaborare sono a 8 bit) word (ciascuna somma può variare tra 0 e 64K-1) consecutive;
- durante la produzione dell'istogramma calcola anche il valor medio a 16 bit (formato in virgola fissa 8.8) dei dati, e al termine dell'elaborazione lo memorizza nella word successiva all'istogramma;
- segnala il termine dell'elaborazione al processore mediante handshake.

STAT_PRO utilizza il clock del processore.

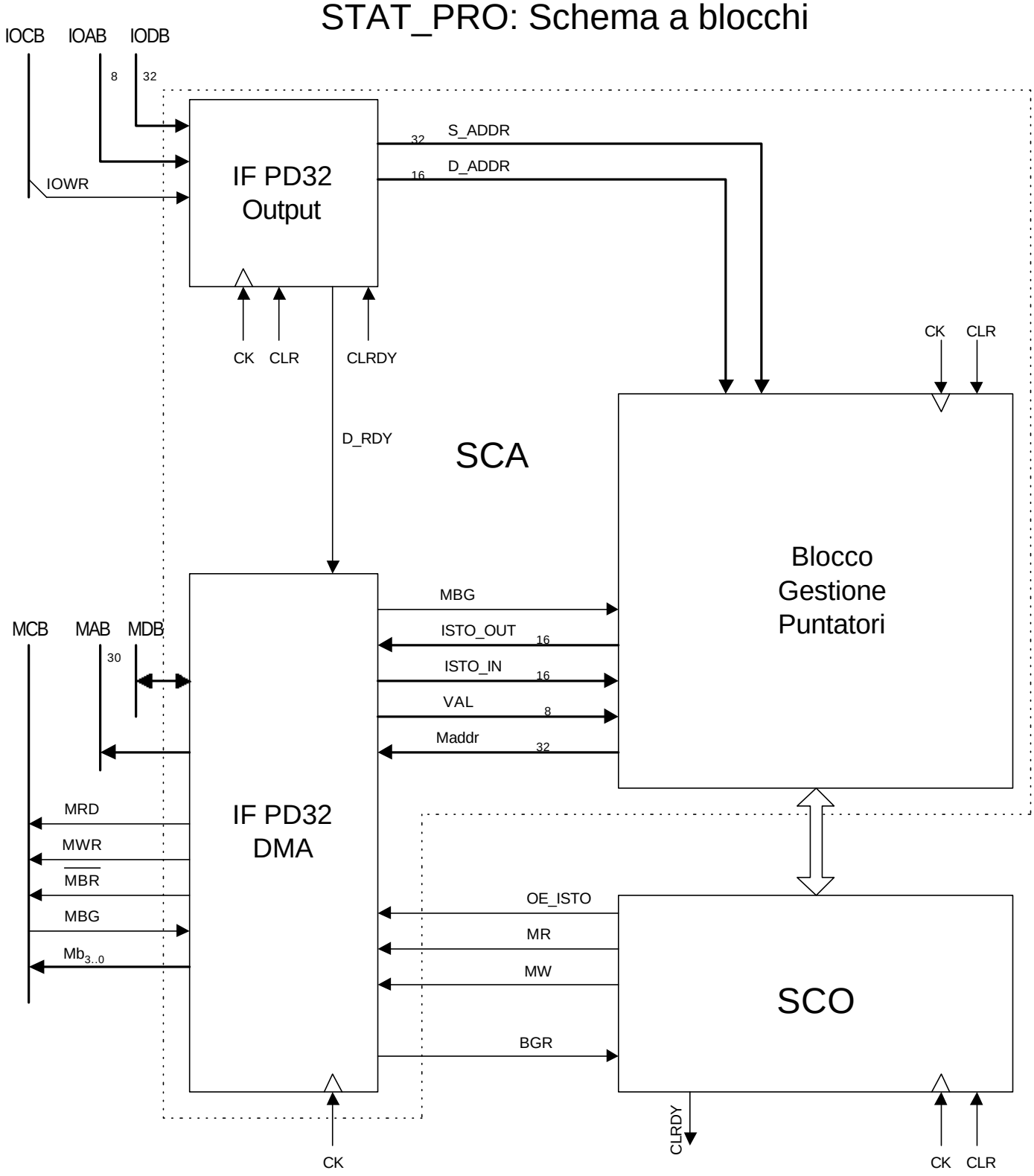
Si richiede:

1. lo schema logico della periferica STAT_PRO;
2. la temporizzazione delle operazioni;
3. il software di interfacciamento del PD32.

STAT_PRO: sistema esterno



STAT_PRO: Schema a blocchi

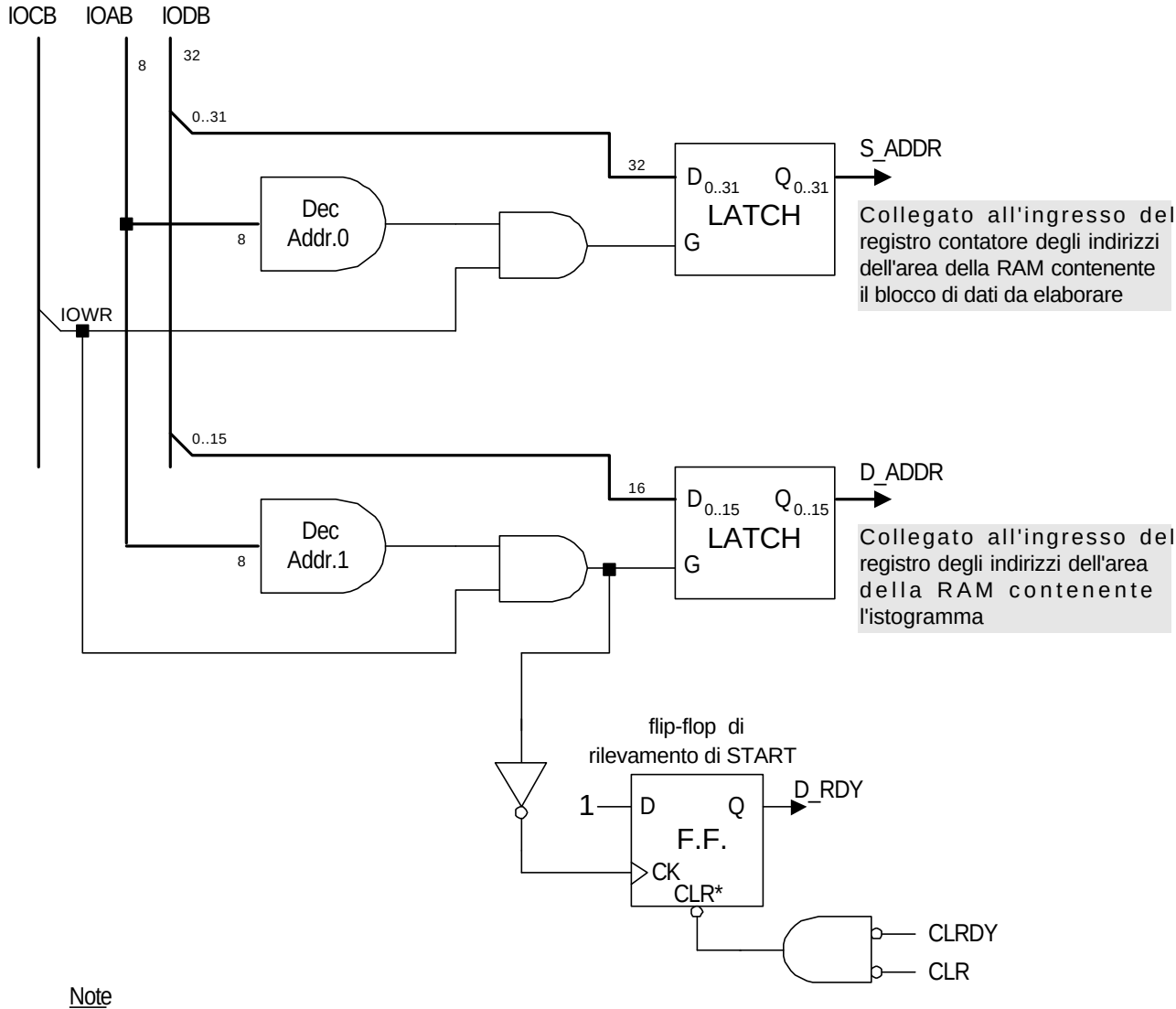


Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.

Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica.

STAT-PRO: IF PD32 - output



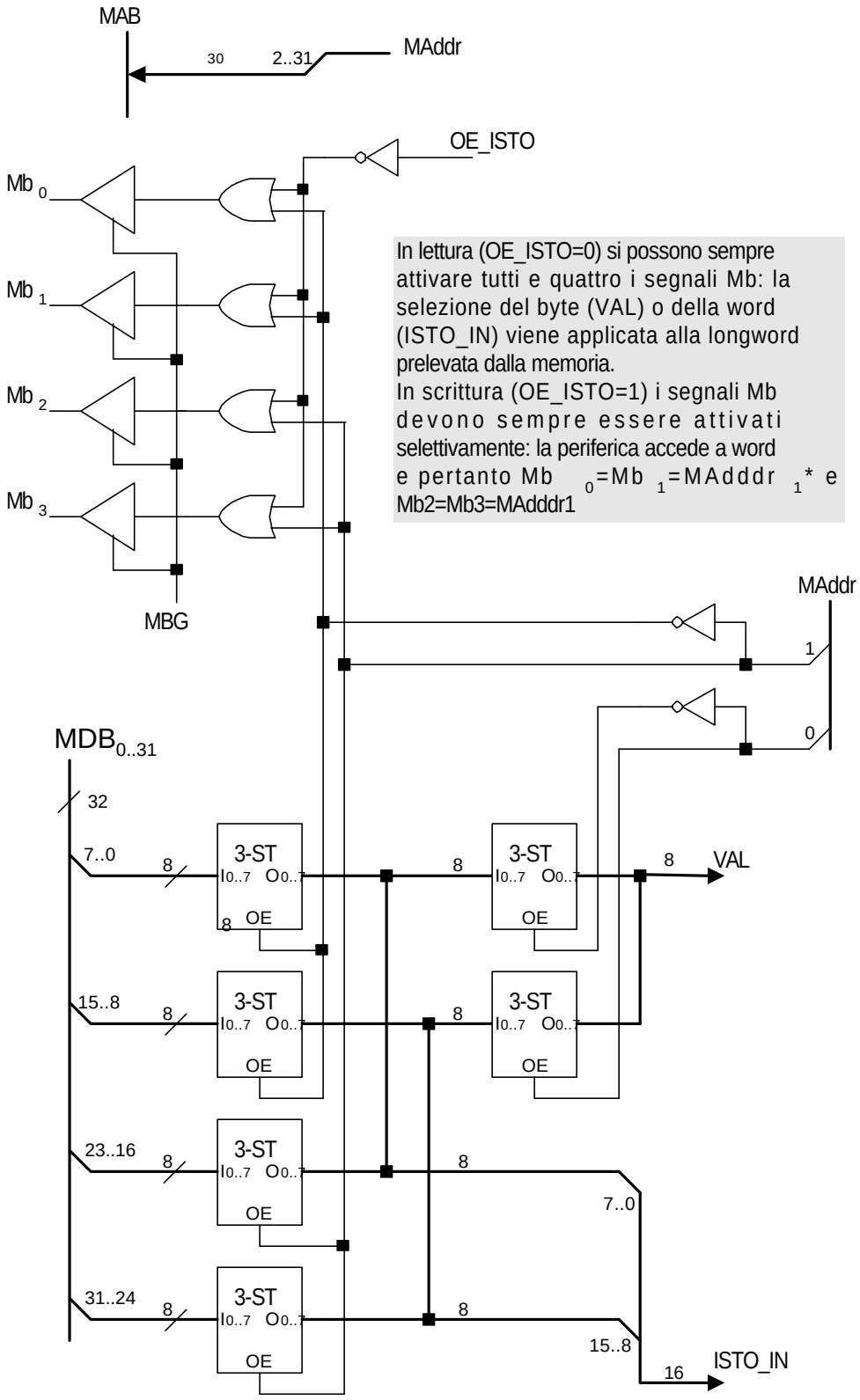
Note

In questa implementazione il SW avvia l'operazione direttamente con la scrittura dell'indirizzo dell'area di memoria di destinazione (istogramma), dopo avere eventualmente riscritto l'altro registro:
outl BLOCK,Addr0
outw ISTO,Addr1
supponendo BLOCK e ISTO due locazioni di memoria dove sono contenuti gli indirizzi iniziali del blocco di dati da elaborare e dell'istogramma rispettivamente.

Non è necessario campionare il segnale D_RDY, in quanto questo non viene utilizzato dalla periferica; infatti, viene collegato direttamente alla linea di richiesta di bus del micro. Sarà invece sincronizzata la linea di bus grant, in quanto letta dallo SCO.

Va notato che in questo caso (reazione a un comando del micro e accesso in DMA a burst) non è necessario che la periferica emetta una richiesta di interruzione al termine delle operazioni, in quanto al termine del ciclo-macchina che nell'istruzione
outw ISTO,Addr1
setta il flip-flop D_RDY della periferica il micro sicuramente avrà ceduto i bus alla periferica (va notato che l'uscita D_RDY è collegata direttamente, cioè non mediante lo SCO, alla richiesta di bus) e resterà bloccato per tutto il tempo di lavoro della periferica (che accede in DMA a burst); pertanto, al termine delle operazioni quando la periferica restituirà i bus al micro, questo eseguirà l'istruzione successiva (alla outw), in cui già potrà utilizzare il risultato della periferica.

STAT_PRO: IF PD32 - DMA - 1

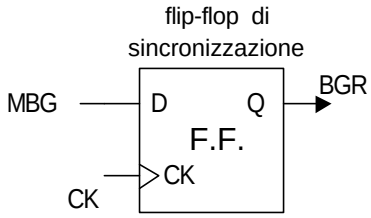


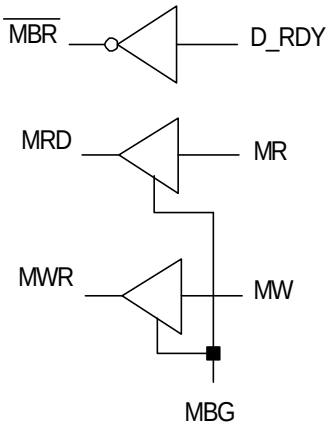
In lettura (OE_ISTO=0) si possono sempre attivare tutti e quattro i segnali Mb: la selezione del byte (VAL) o della word (ISTO_IN) viene applicata alla longword prelevata dalla memoria.
In scrittura (OE_ISTO=1) i segnali Mb devono sempre essere attivati selettivamente: la periferica accede a word e pertanto $Mb_0 = Mb_1 = MAdddr_1$* e Mb2=Mb3=MAdddr1

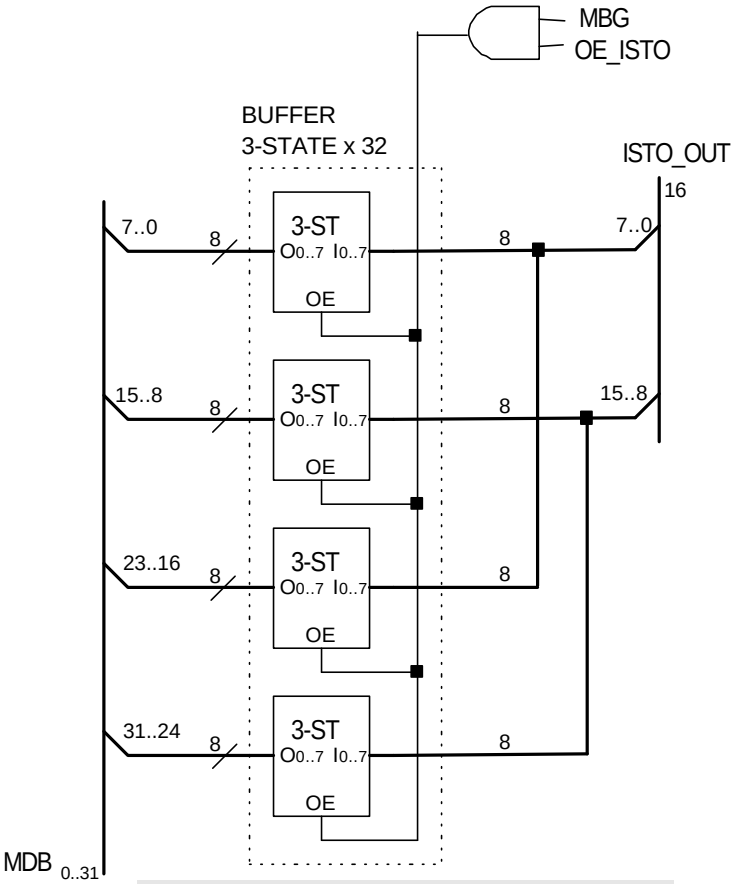
Blocco di riduzione dei 32 bit del MDB ai 16 bit di ISTO_IN (lettura).

Blocco di riduzione dei 32 bit del MDB agli 8 bit di VAL (lettura).

STAT_PRO: IF PD32 - DMA - 2

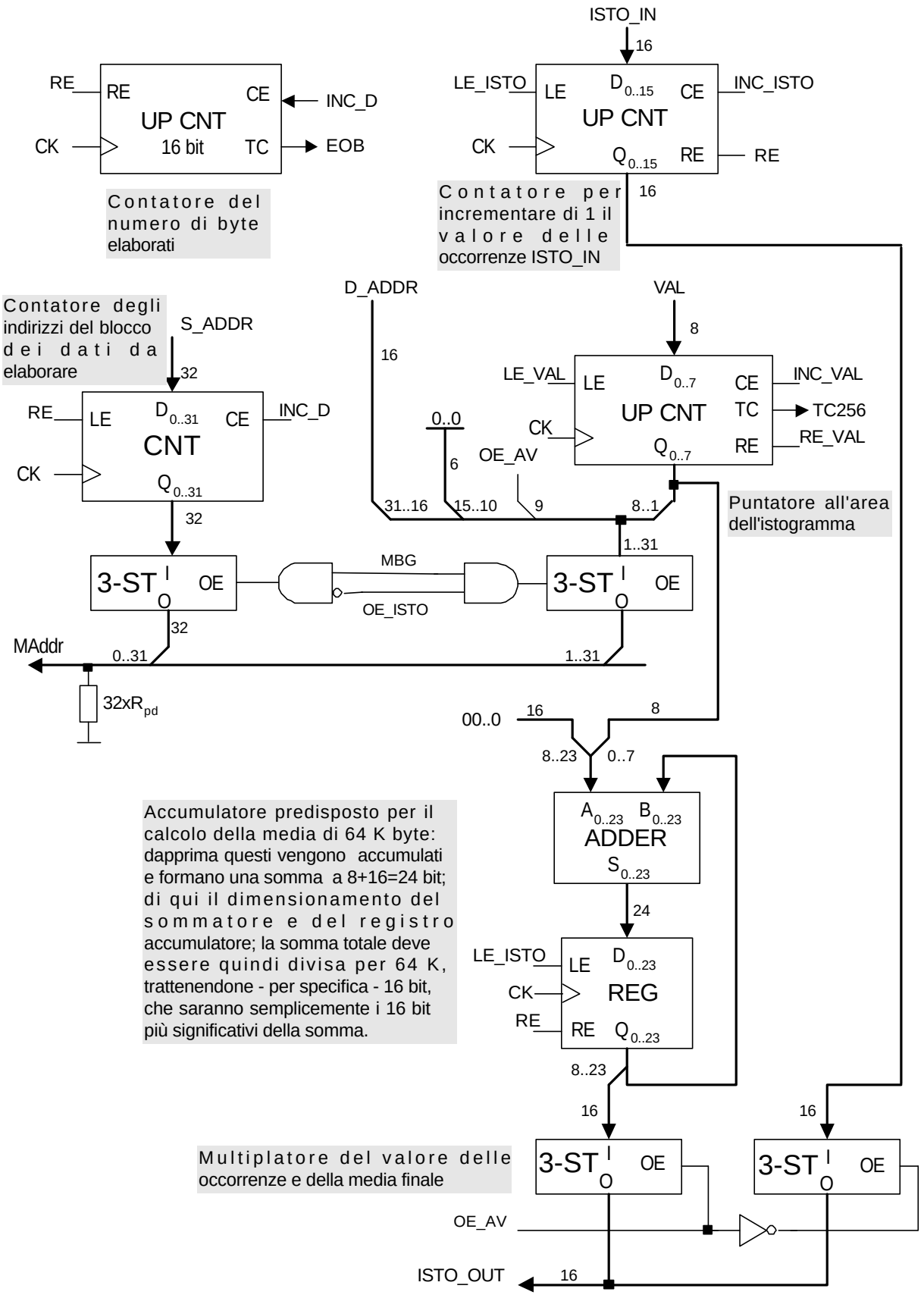




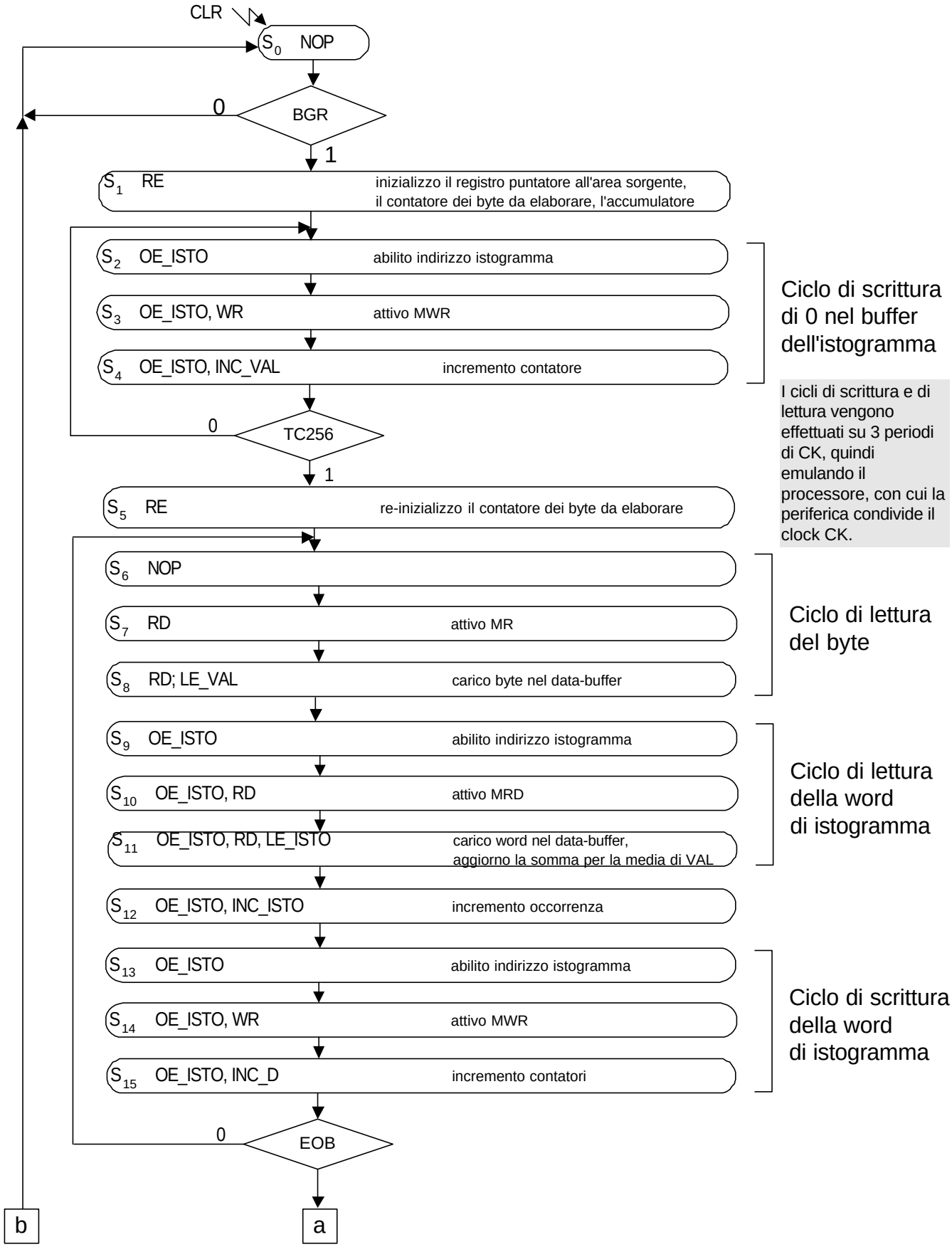


Blocco di espansione dei 16 bit del bus interno ISTO ai 32 bit del MDB (scrittura).
L'abilitazione dei 3-state deve essere condizionata (cfr. porta AND su MBG) da un segnale di accesso in scrittura (OE_ISTO), per evitare di interferire sul MDB durante gli accessi in lettura.

STAT-PRO: blocco gestione puntatori



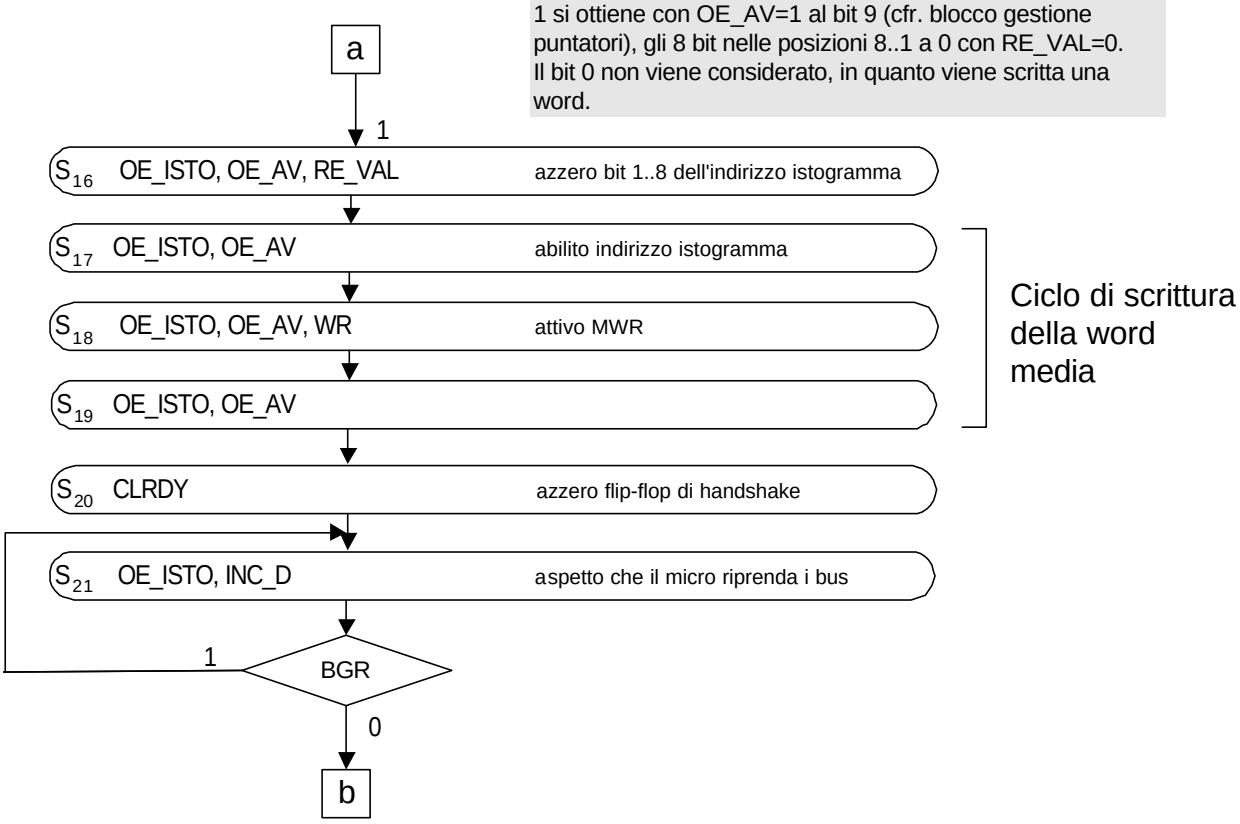
STAT_PRO: SCO - flowchart



I cicli di scrittura e di lettura vengono effettuati su 3 periodi di CK, quindi emulando il processore, con cui la periferica condivide il clock CK.

STAT_PRO: SCO - flowchart

Quando EOB=1 si deve scrivere la media nella word successiva all'istogramma, all'indirizzo XXXX0200; il valore 1 si ottiene con OE_AV=1 al bit 9 (cfr. blocco gestione puntatori), gli 8 bit nelle posizioni 8..1 a 0 con RE_VAL=0. Il bit 0 non viene considerato, in quanto viene scritta una word.

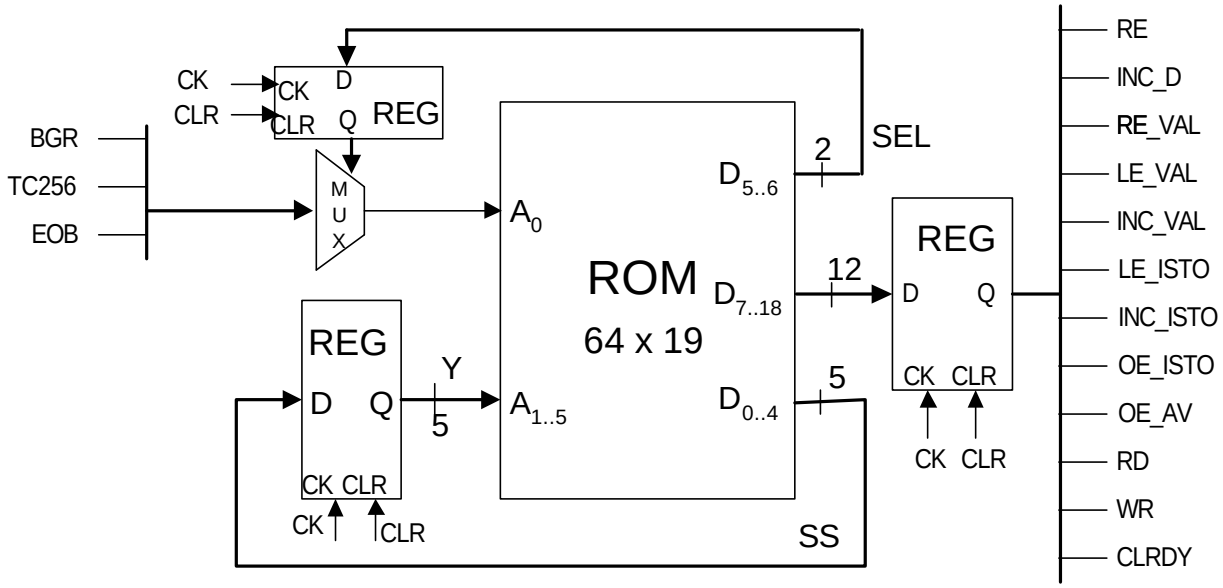


Note

La temporizzazione delle micro-operazioni è dettata dalla scansione delle microistruzioni del diagramma di flusso.

STAT-PRO: SCO - struttura HW microprogrammata

Il flow-chart è supportato da un microlinguaggio di tipo 3; scegliendo il modello strutturale di tipo D-Mealy si ottiene la struttura seguente:



RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 6-6-2001

STUDENTE:_____DOCENTE:_____

D1 Valutare il numero espresso dalla stringa F1h in un'aritmetica in complemento a 2 con 8 bit; quindi rappresentare lo stesso numero in un formato in virgola mobile a 32 bit.

D2 Un convertitore ADC a 8 bit produce un flusso di dati alla velocità di 20 Mbyte/s che devono essere protetti con un codice a correzione di singolo errore mediante moduli ROM con tempo di accesso pari a 80 ns; dimensionare il banco di ROM codificatore.

D3 Disponendo di un flip-flop di tipo D, progettare un flip-flop di tipo JK dotato di ingresso CE (clock enable) di abilitazione alla commutazione.

D4 Descrivere una struttura multimicroprogrammata di tipo Moore.

D5 Descrivere l'allocazione in memoria dell'istruzione PD32:
MOVW #5A3Ch,R0
memorizzata a partire dall'indirizzo ABCDEF02h, e indicare il numero dei cicli-macchina e dei cicli di bus impegnati dal relativo ciclo-istruzione.

Esercizio (2S20010606-D1)

Valutare il numero espresso dalla stringa F1h in un'aritmetica in complemento a 2 con 8 bit; quindi rappresentare lo stesso numero in un formato in virgola mobile a 32 bit.

---

1. Trasformazione della stringa esadecimale in codice binario:

$F1_{16} = 11110001_2$

Il bit più pesante è 1, quindi il numero rappresentato è negativo, nel formato specificato con 8 bit per la sola parte intera.

2. Complementazione a 2 del numero binario e sua valutazione decimale:

$00001111_2 = (2^3 + 2^2 + 2^1 + 2^0)_{10} = 15_{10}$

da cui segue che il numero proposto vale: -15.

Allo stesso risultato si può pervenire eseguendo la valutazione della stringa interpretandola come numero positivo, ottenendo:

$11110001_2 = (2^7 + 2^6 + 2^5 + 2^4 + 2^0)_{10} = 141_{10}$

A questo punto, ricordando che con 8 bit i numeri rappresentati in complemento a 2 sono di fatto espressi in complemento a $2^8 = 256$, si riconosce che 141 (> 128) è la rappresentazione del numero negativo:
141 - 256 = -15.

3. In virgola mobile il numero –15 è scritto come: $(-1)\ 2^4\ 15/16$ che viene rappresentato dalla stringa a 32 bit:

1 00000100 11110000000000000000

Esercizio (2S20010606-D2)

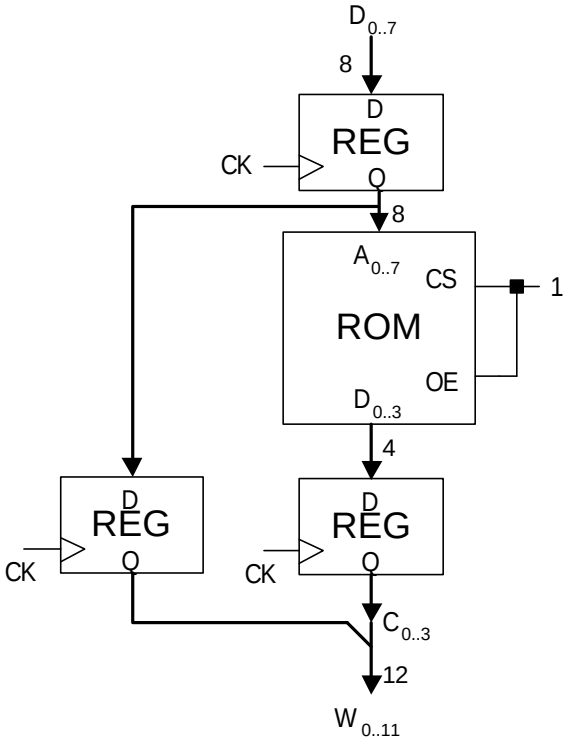
Un convertitore ADC a 8 bit produce un flusso di dati alla velocità di 20 Mbyte/s che devono essere protetti con un codice a correzione di singolo errore mediante moduli ROM con tempo di accesso pari a 80 ns; dimensionare il banco di ROM codificatore.

Il primo passo consiste nel calcolare il numero k dei bit di controllo richiesti per la codifica a correzione di errore specificata: ponendo C=1 nella relazione $2C \leq h-1$ si ottiene il valore minimo h=3, che indica l'uso di un codice di Hamming a distanza 3. La relativa relazione vincolare:

$2^k-k-1 \geq n = 8$

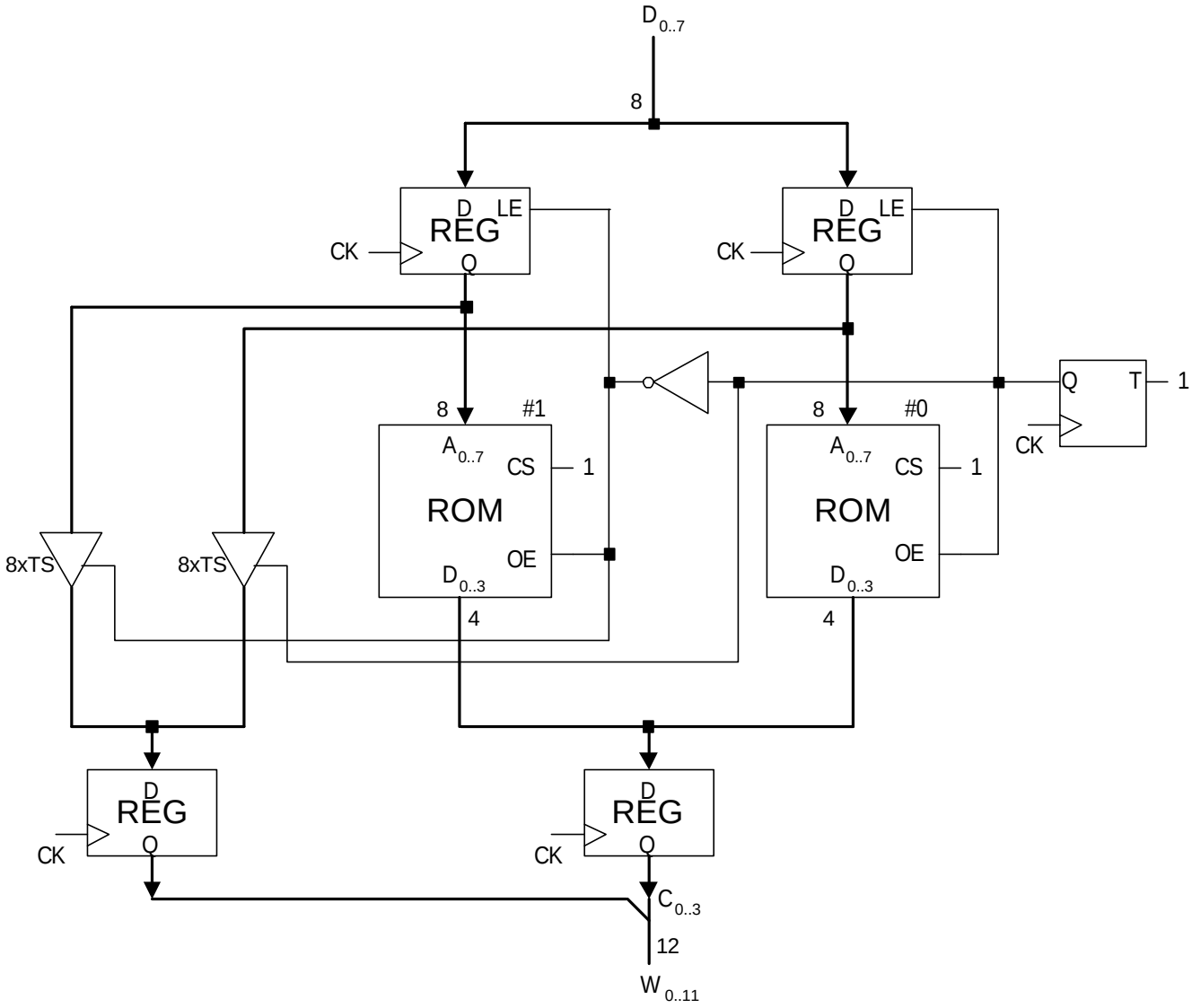
restituisce k=4; pertanto la parola codificata sarà di 8+4=12 bit.

Se il tipo di ROM disponibile avesse un tempo di accesso inferiore al periodo di alternanza dei dati (= 50 ns) sarebbe sufficiente il circuito seguente:



in cui il dato $D_{0..7}$ è prodotto dal convertitore ADC, il cui ingresso SOC si suppone collegato a CK, di frequenza 20 MHz. Va notato che la ROM è utilizzata per produrre i soli bit di controllo; infatti, i bit del codice irridondante possono essere semplicemente propagati sul livello di uscita. Per completezza l'uscita della ROM è stata registrata, e di conseguenza è stato registrato sullo stesso livello di uscita anche il dato presentato in ingresso alla ROM: i due registri di uscita formano un unico registro a 12 bit, la cui uscita è la parola codificata, con i bit di dato e quelli di controllo allineati temporalmente.

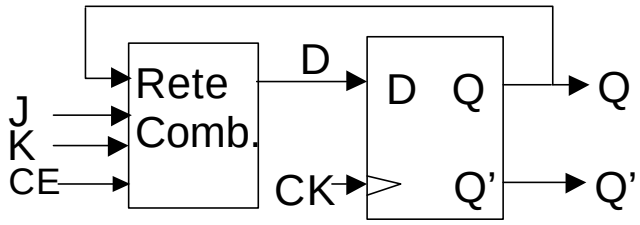
D’altra parte, poiché il tipo di ROM effettivamente disponibile ha un tempo di accesso pari a 80 ns, quindi maggiore del periodo di alternanza dei dati (= 50 ns) diventa necessario applicare uno schema di pipeline alla ROM; il grado di parallelismo si ricava dalla relazione:
50 ns < 80 ns < 2 x 50 ns
cioè occorre raddoppiare la ROM. Tenendo conto anche in questo caso della necessità di allineare temporalmente i bit di dato con quelli di controllo, si ottiene lo schema seguente:



Esercizio (2S20010606-D3)

Disponendo di un flip-flop di tipo D, progettare un flip-flop di tipo JK dotato di ingresso CE (clock enable) di abilitazione alla commutazione.

Il flip-flop può essere considerato una rete sequenziale sincrona, che risponde al noto modello strutturale, che di seguito viene personalizzato al caso del flip-flop: il registro di stato è composto da un solo flip-flop D, esattamente quello che il testo del problema raccomanda di utilizzare.



E' appena il caso di notare che in questa struttura, come in tutte quelle che rispondono allo stesso modello strutturale di rete sincrona, il clock è un segnale non soggetto a elaborazioni, di nessun tipo!

Il progetto è stato così ricondotto al calcolo della rete combinatoria. D è lo stato successivo del flip-flop (registro di stato della rete).
la tavola di verità è:

CE	Q	J	K	D
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

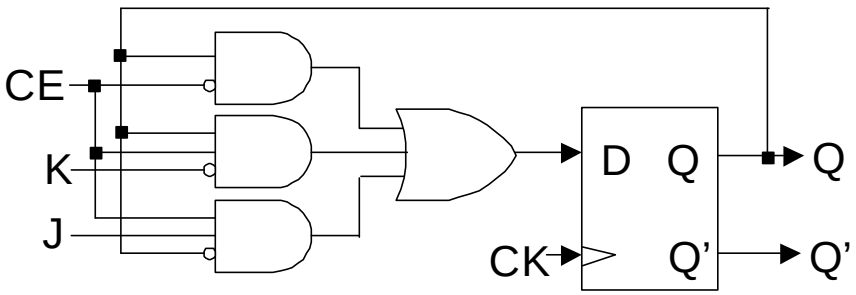
Minimizzazione mediante MK:

CE Q \ J K	00	01	11	10
00				
01	1	1	1	1
11	1			1
10			1	1

Selezionando gli implicanti principali si ottiene:

$$D = \overline{CE} \cdot Q + CE \cdot Q \cdot \overline{K} + CE \cdot \overline{Q} \cdot J$$

Ne consegue lo schema del flip-flop richiesto:



Esercizio (2S20010606-D4)

Descrivere una struttura multimicroprogrammata di tipo Moore.

Cfr. testo “Reti sequenziali” e gli “Appunti integrativi”.

Esercizio (2S20010606-D5)

MOVW #5A3Ch,R0
memorizzata a partire dall'indirizzo ABCDEF02h, e indicare il numero dei cicli-macchina e dei cicli di bus impegnati dal relativo ciclo-istruzione.

I byte che formano l'istruzione specificata sono 6, allocati in memoria agli indirizzi descritti di seguito:

Codice	ABCDEF02h
operativo	ABCDEF03h
della	ABCDEF04h
istruzione	ABCDEF05h
3Ch	ABCDEF06h
5Ah	ABCDEF07h

I cicli-macchina compresi nel ciclo-istruzione sono 2:
- fetch dell'istruzione;
- prelevamento del dato immediato 5A3Ch.

I cicli di bus compresi nel ciclo-macchina del fetch sono 2: infatti, la longword è disposta a partire da un indirizzo non di tipo mod 4; nel primo ciclo di bus viene prelevata la coppia di byte agli indirizzi ABCDEF02-3h; nel secondo ciclo di bus viene prelevata la coppia di byte agli indirizzi ABCDEF04-5h.

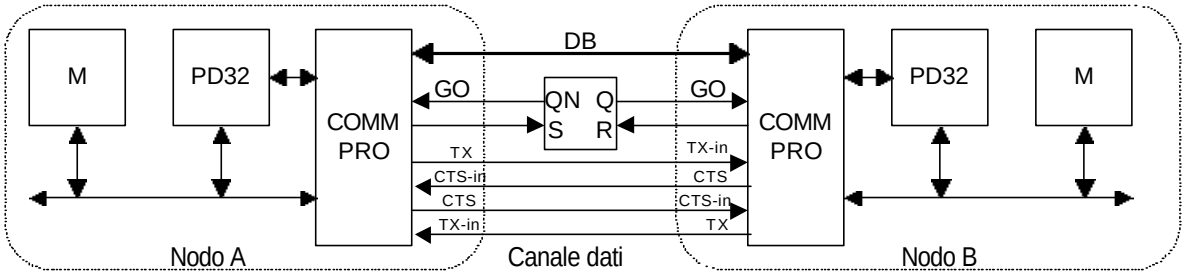
il ciclo-macchina relativo al prelevamento del dato immediato 5A3Ch richiede un unico ciclo di bus: infatti, i due byte della word relativa sono ai due indirizzi ABCDEF06-7h, non di tipo mod 4.

In definitiva **i cicli di bus nel ciclo-istruzione sono 3**.

RETI LOGICHE
PRIMA PROVA SCRITTA DELL'APPELLO DEL 15-6-2001

STUDENTE:_____DOCENTE:_____

Si vuole realizzare un calcolatore parallelo costituito da due nodi di elaborazione interconnessi tramite un canale dati, come schematizzato in figura: ciascun nodo è dotato di un processore PD32, della relativa memoria (M) e di un co-processore di comunicazione (COMM_PRO).



Il canale dati è costituito da:
- il bus DB a 8 bit mediante il quale i due COMM_PRO scambiano i dati;
- un flip-flop SR di handshake, con le linee di ingresso Set e Reset e quelle di uscita QN e Q collegate con i COMM_PRO dei nodi A e B rispettivamente, in modo che ognuno dei due nodi legga sul proprio ingresso GO il valore 0 dopo la propria azione sul latch, e 1 dopo la reazione del nodo complementare;
- una linea TX che viene posta a 1 durante la trasmissione di un messaggio;
- una linea CTS che viene posta a 1 per indicare la disponibilità del nodo a ricevere un messaggio.

Quando un processore ha predisposto in RAM un messaggio da trasmettere invia al proprio COMM_PRO:
- l'indirizzo iniziale del blocco di dati da trasmettere;
- la lunghezza, in byte, del blocco da trasmettere;
- setta un flip-flop FTX.

La disponibilità a ricevere un messaggio in RAM è segnalata dal micro al suo co-processore con il set di un flip-flop FRX.

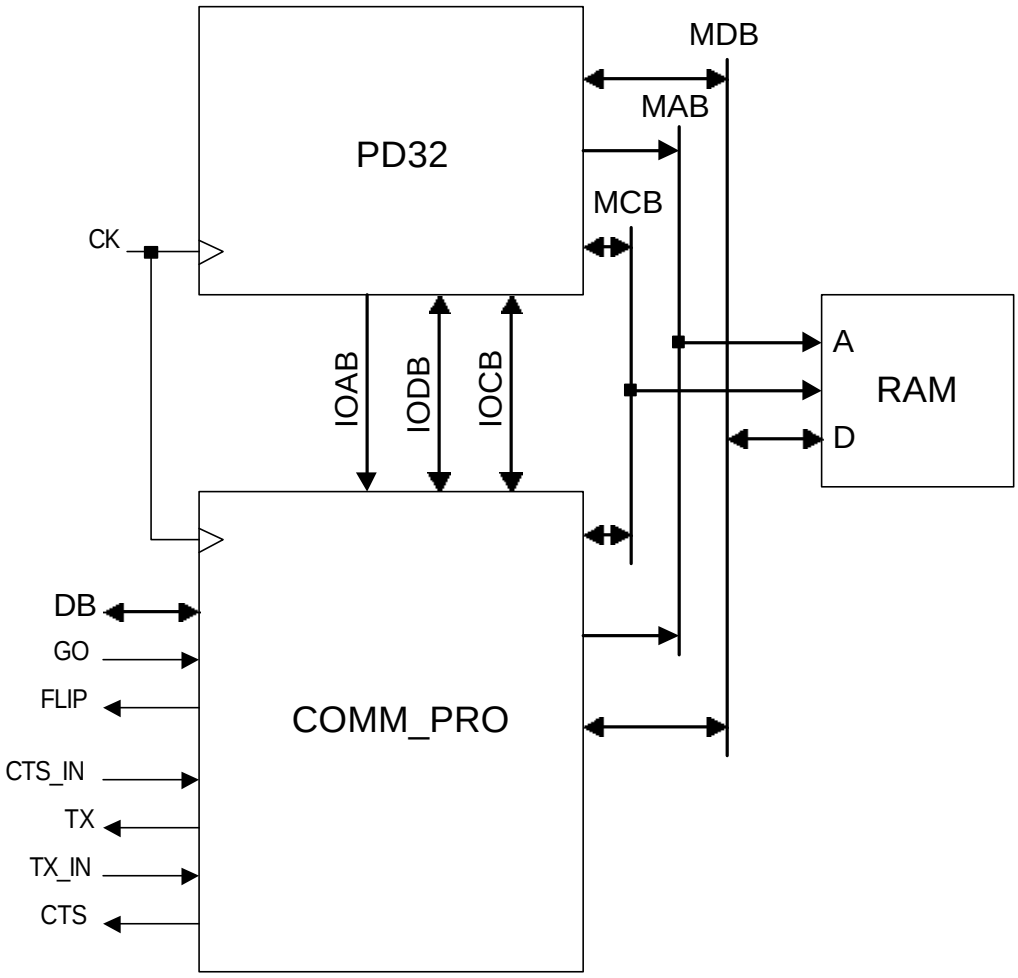
Il co-processore si deve comportare nel seguente modo:
quando GO=1 esegue il **case** su TX_in, CTS_in e FTX:
se TX_in=1 entra nella routine di ricezione; **se** TX_in=0 e CTS_in=FTX=1 entra nella routine di trasmissione, **altrimenti** aziona il flip-flop (GO commuta a 0) e si rimette in attesa di GO=1.

Routine di trasmissione:
1. accede in DMA alla memoria del microprocessore in modalità burst, pone TX a 1, trasferisce il primo byte su DB e aziona il FF di semaforo (GO commuta a 0);
2. per trasferire il byte successivo attende GO=1, preleva il byte dalla memoria, lo trasferisce su DB e aziona il FF di semaforo (GO commuta a 0);
3. torna a 2 fino a che il messaggio non è terminato;
4. quando GO=1, pone TX=0 e aziona il FF di semaforo (GO commuta a 0);
5. scrive il codice FFh nella prima locazione del blocco di dati trasmesso e resetta FTX.

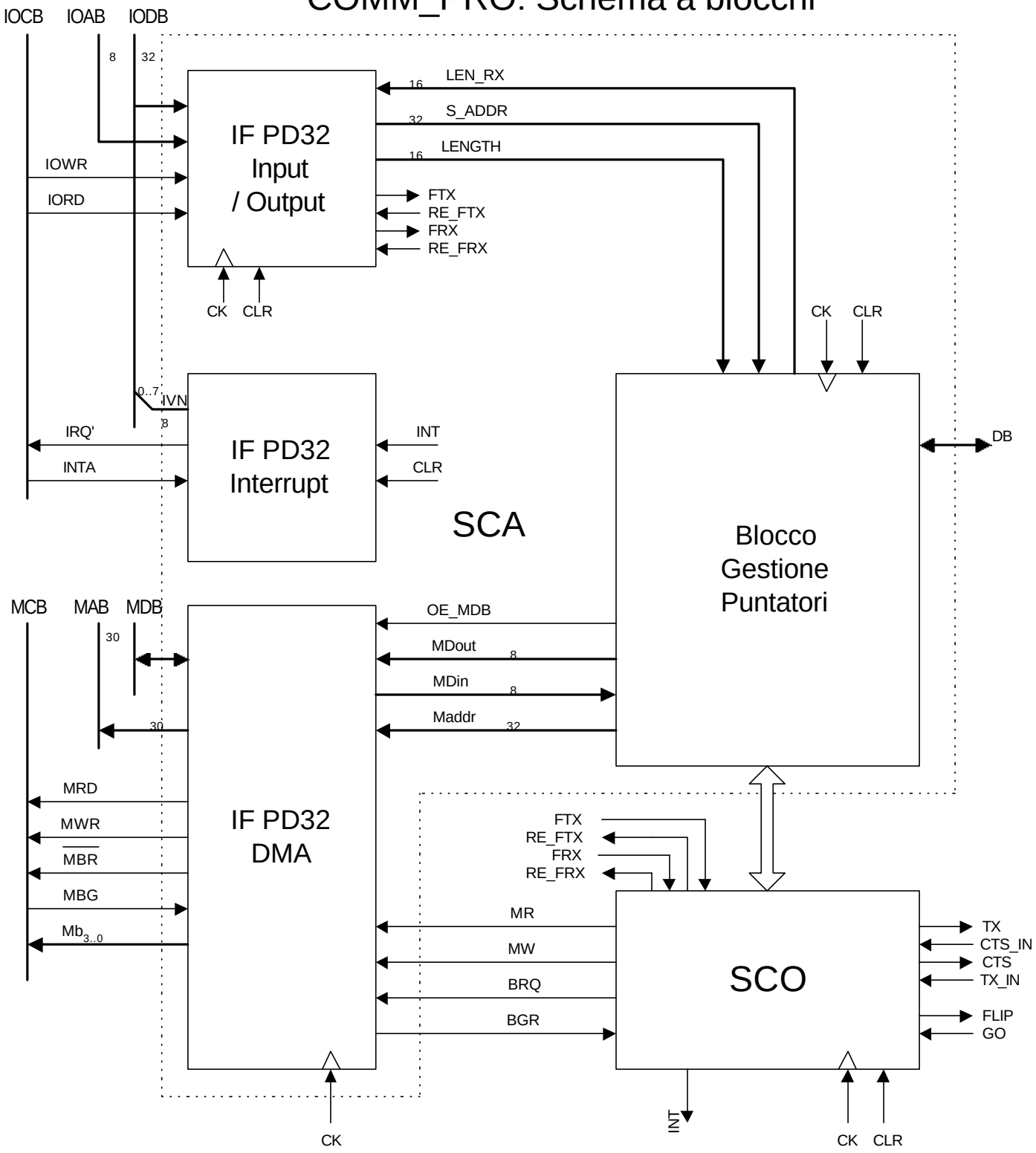
Routine di ricezione:
1. accede in DMA alla memoria del microprocessore in modalità burst a un indirizzo noto, trasferisce il primo byte da DB in RAM e aziona il FF di semaforo (GO commuta a 0);
2. per trasferire i byte successivi in ogni passo del ciclo attende GO=1, esegue il test su TX_in: se TX_in=1 trasferisce il singolo byte da DB nella locazione successiva della RAM, e aziona il FF di semaforo (GO commuta a 0); se TX_in=0 (trasmissione terminata) aziona il FF di semaforo (GO commuta a 0);
3. rende disponibile al processore la lunghezza del messaggio ricevuto in un registro; resetta il flip-flop FRX, e quindi lancia una richiesta di interruzione al processore.

Si richiede:
1. lo schema logico della periferica COMM_PRO;
2. il microprogramma.

COMM_PRO: sistema esterno



COMM_PRO: Schema a blocchi

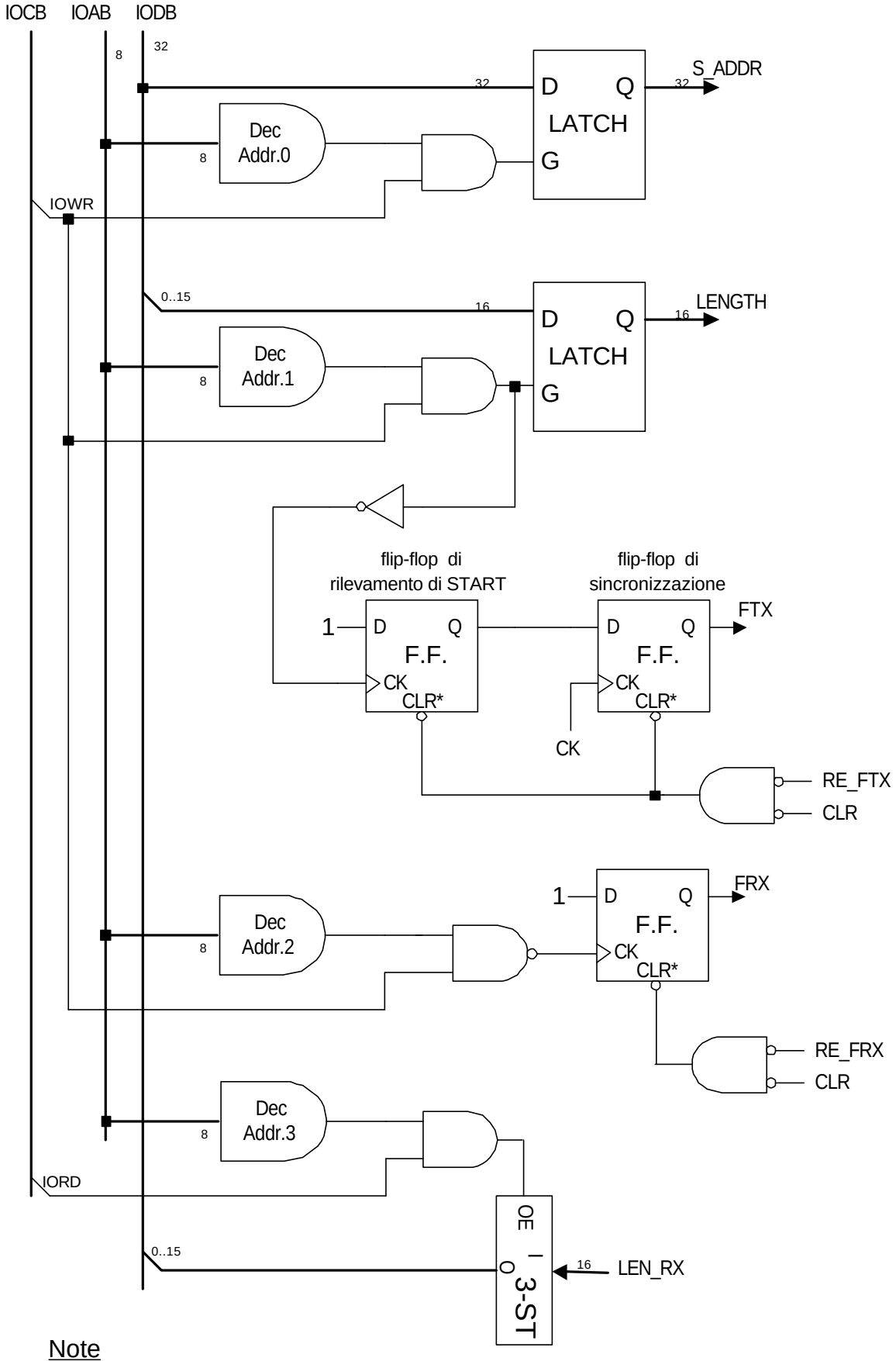


Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.

Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 interrupt usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

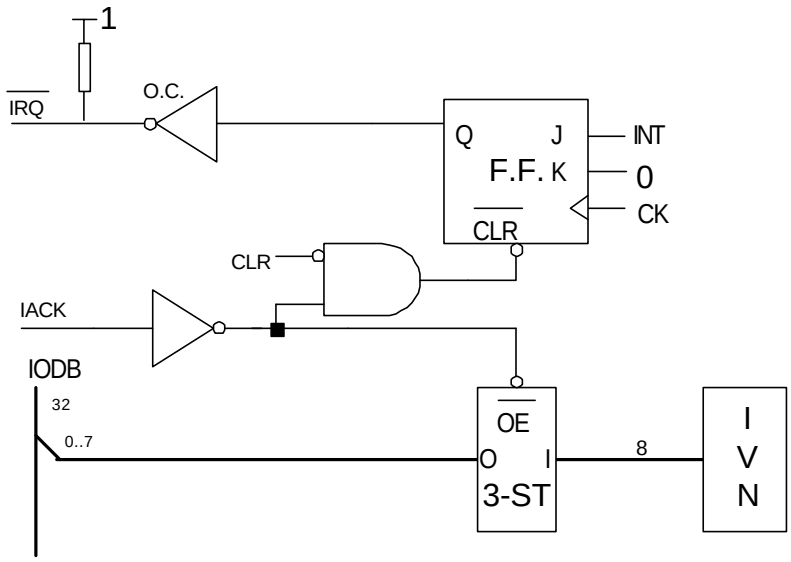
COMM_PRO: IF PD32 - output



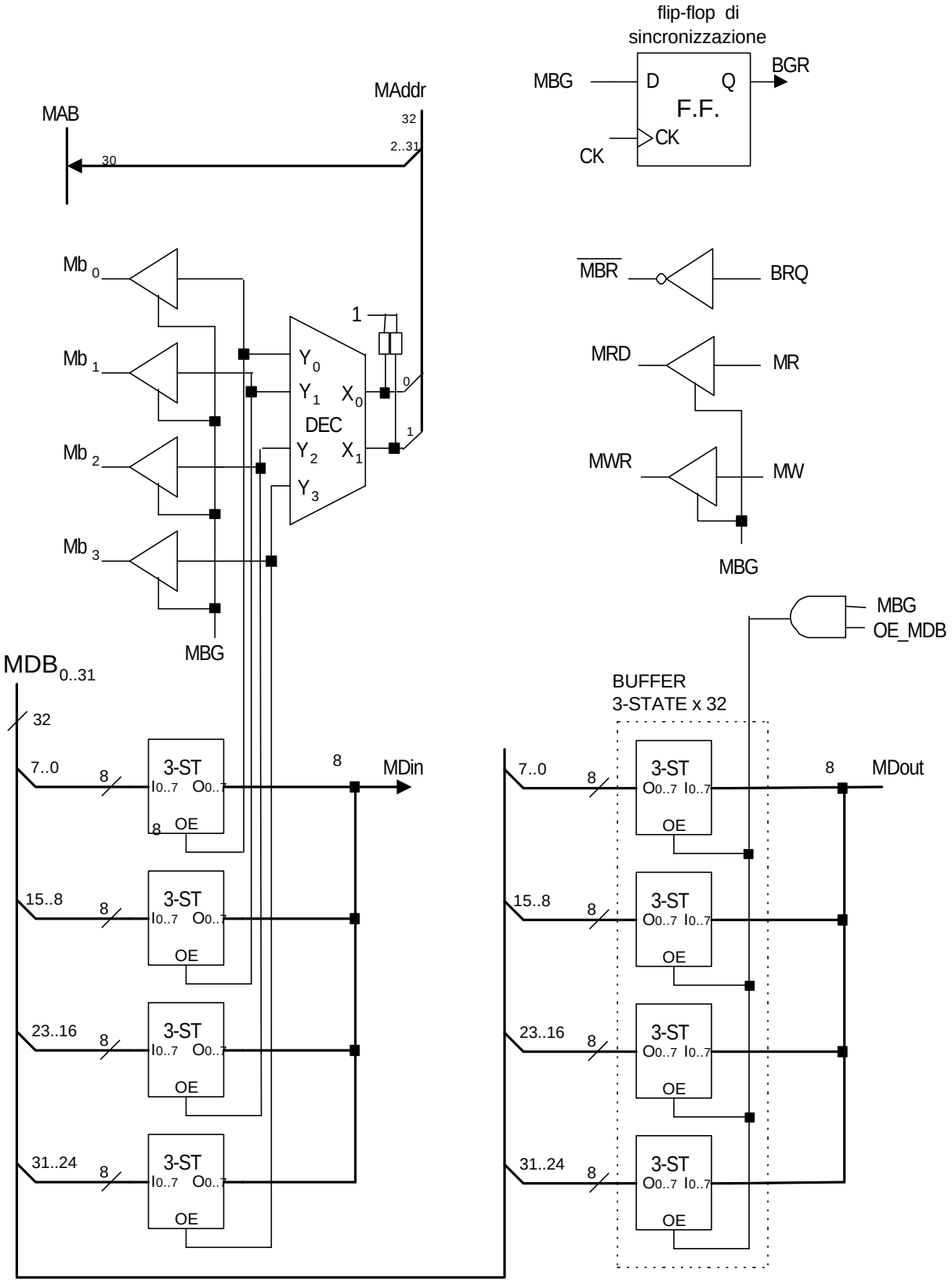
Note

Il SW avvia l'operazione direttamente con la scrittura dell'indirizzo della lunghezza in byte del blocco di dati da trasferire, dopo avere riscritto l'altro registro.

COMM_PRO: IF PD32 - interrupt



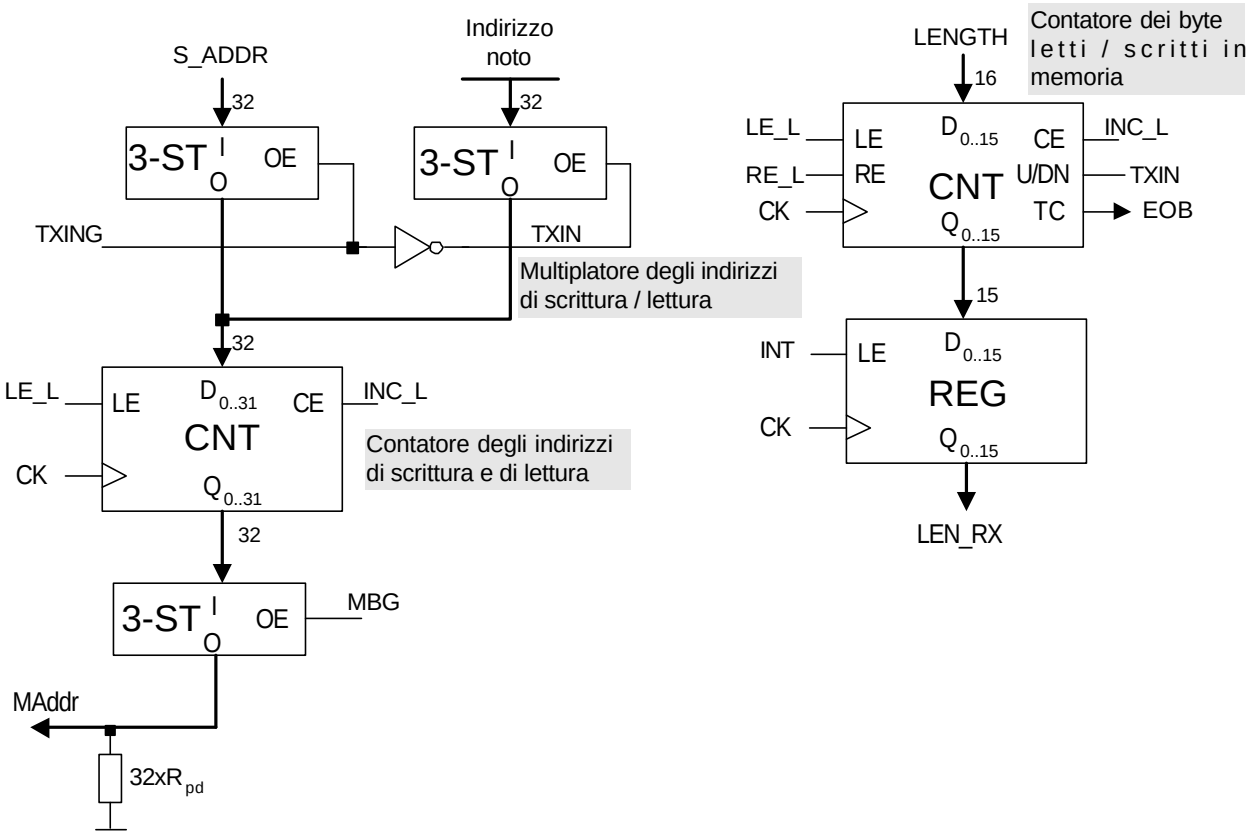
COMM_PRO: IF PD32 - DMA

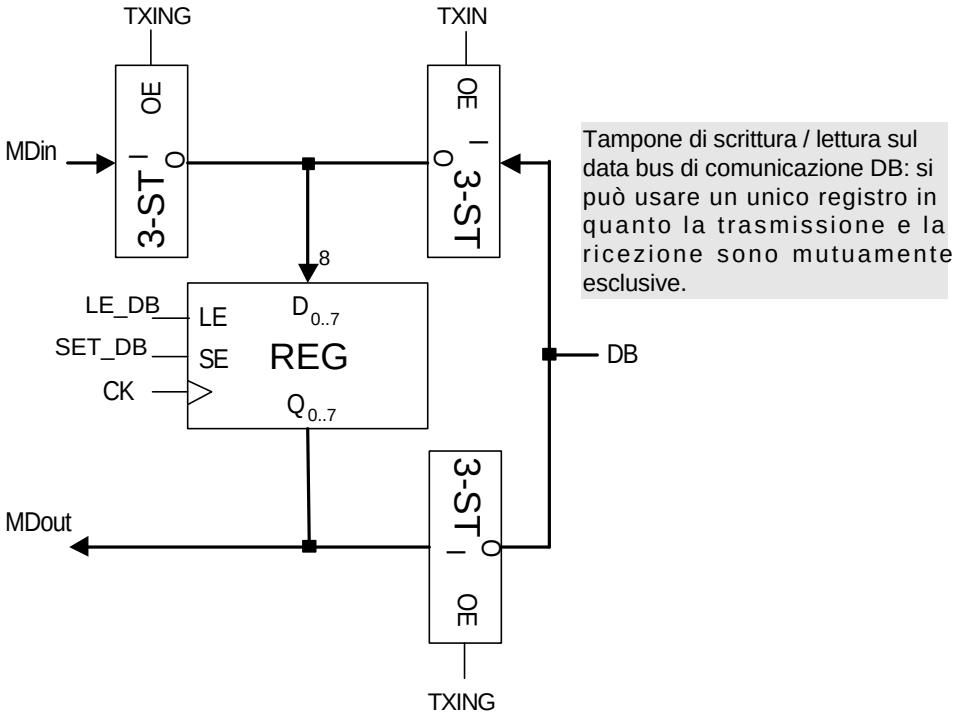


Blocco di riduzione dei 32 bit del MDB agli 8 bit del bus interno MDin (lettura).

Blocco di espansione degli 8 bit del bus interno MDout ai 32 bit del MDB (scrittura). L'abilitazione dei 3-state deve essere condizionata (cfr. porta AND su MBG) da un segnale di accesso in scrittura (OE_MDB), per evitare di interferire sul MDB negli accessi in lettura.

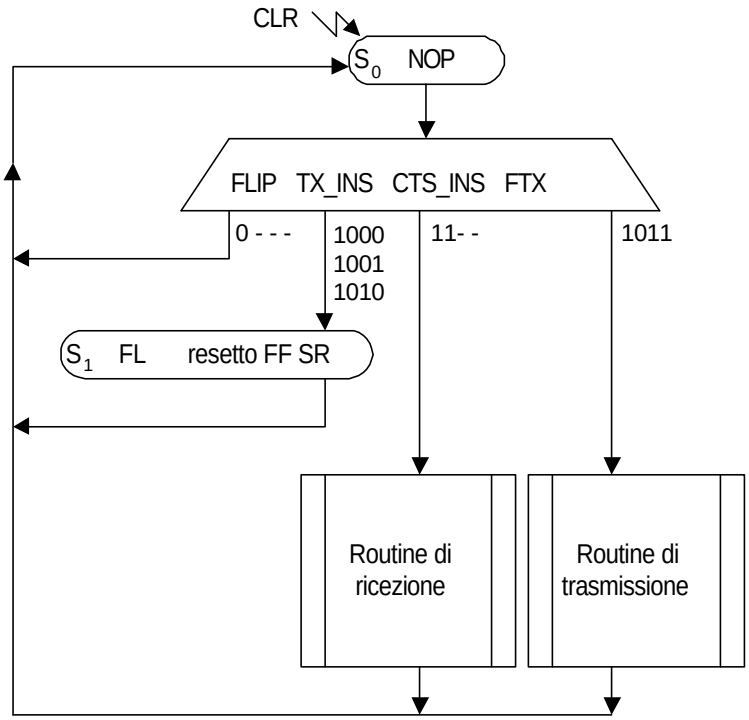
COMM-PRO: blocco gestione puntatori



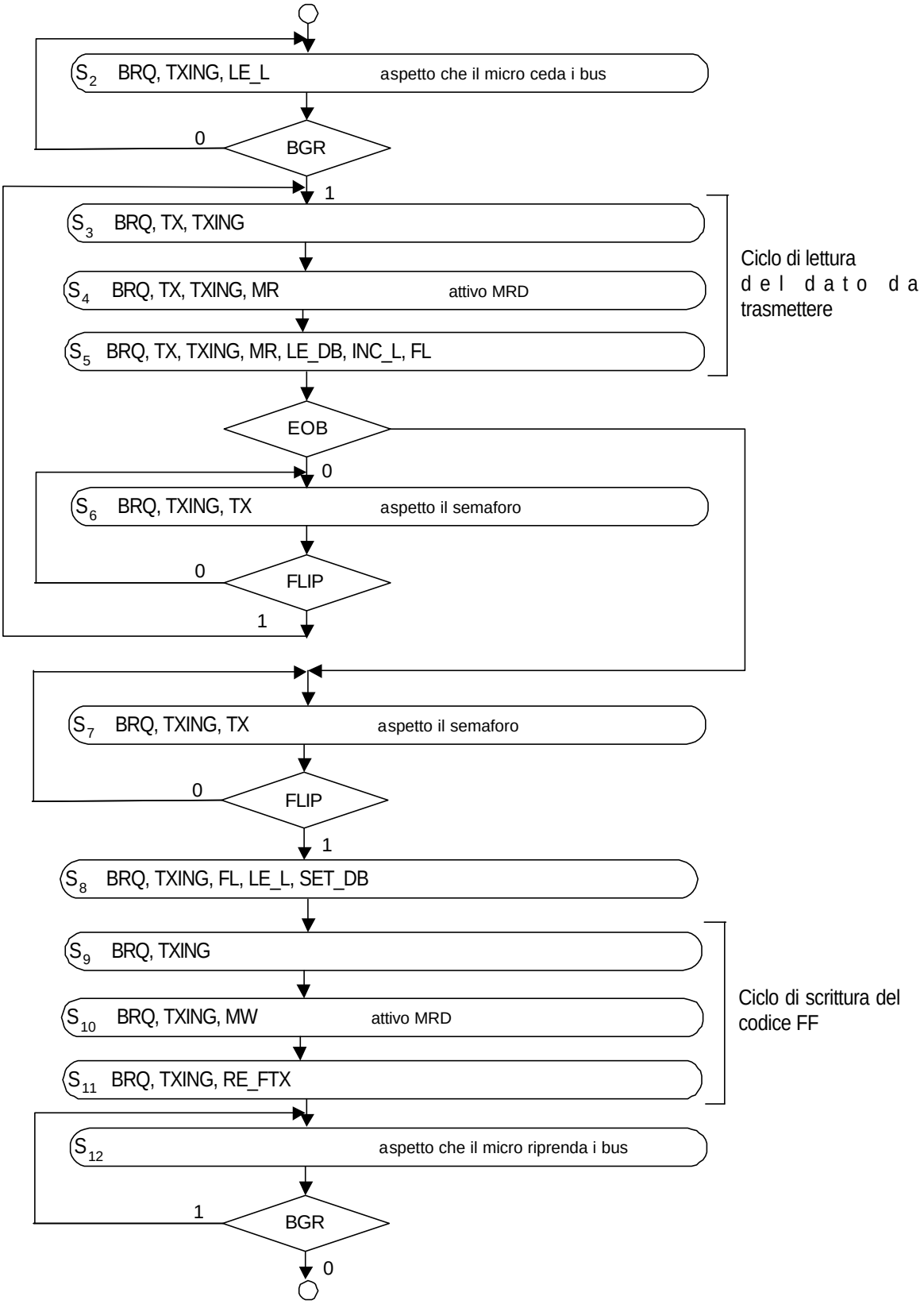


Tampone di scrittura / lettura sul data bus di comunicazione DB: si può usare un unico registro in quanto la trasmissione e la ricezione sono mutuamente esclusive.

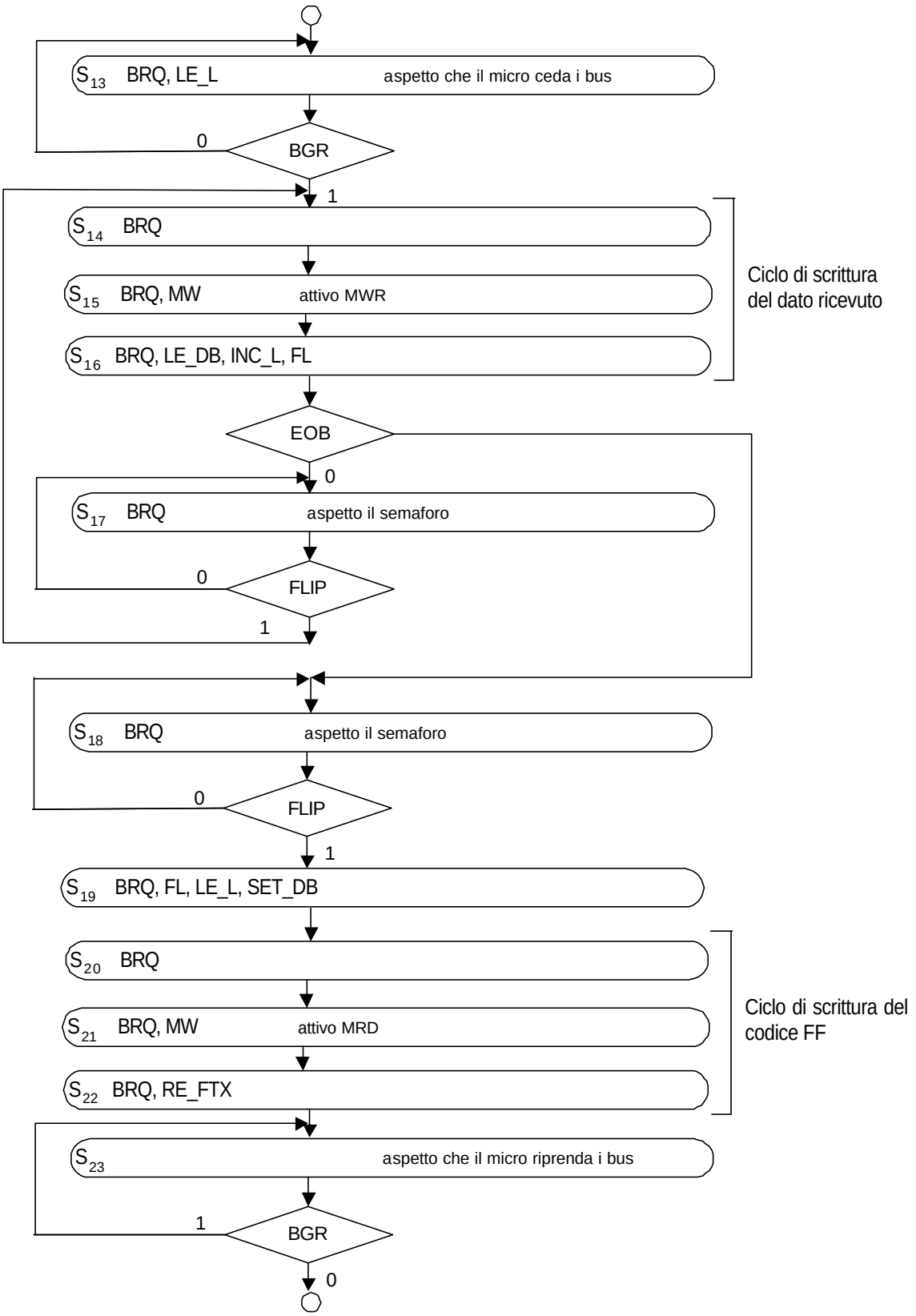
COMM_PRO: SCO - flowchart



COMM_PRO: SCO - flowchart
Routine di trasmissione

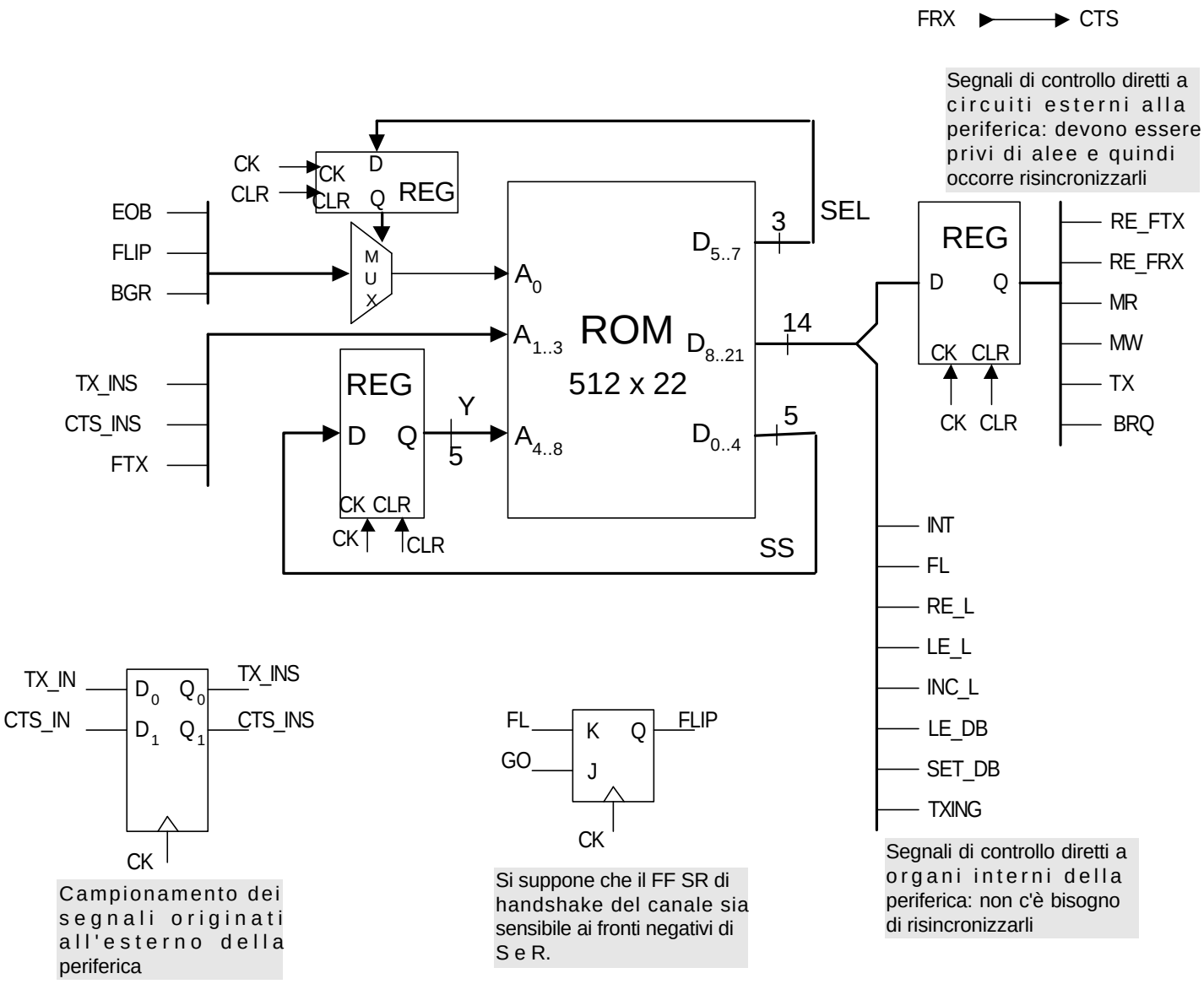


COMM_PRO: SCO - flowchart
Routine di ricezione



COMM_PRO: SCO - struttura HW microprogrammata

Scegliendo il modello strutturale di tipo Mealy si ottiene la struttura seguente:



Note

I segnali di uscita sono stati partizionati in due blocchi: soltanto i segnali diretti all'esterno sono tamponati, tutti gli altri hanno la funzione di abilitazione dei componenti interni alla periferica e quindi non necessitano di tamponamento. Lo SCO è perciò di tipo Mealy con riguardo a questi ultimi segnali e funziona da D-Mealy con riguardo ai segnali di uscita tamponati. A questo punto va notato che i segnali non tamponati sono stati associati agli stati e non alle transizioni nel diagramma di flusso, anche se è di tipo Mealy: questo significa soltanto che i segnali in questione non dipendono dagli ingressi e quindi possono essere attribuiti agli stati per semplicità di rappresentazione .
A fronte di questa posizione, in fase di codifica del microprogramma bisognerà fare attenzione a calcolare i segnali di uscita dei due blocchi in modo da anticipare la presentazione dei segnali assoggettati a tamponamento.

Il test sullo stato del flip-flop SR di handshake non viene effettuato sulla sua uscita GO, ma sull'uscita del flip-flop di ricampionamento interno, come al solito; va notato che è questo stesso flip-flop di campionamento che aziona a sua volta il flip-flop SR.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 15-6-2001

STUDENTE:_____DOCENTE:_____

D1 Indicare il valore dei numeri binari seguenti, espressi nel formato 8.4 in complemento a 2:

11111111.0000
11111111.1000
11111111.1100
11111111.1111

D2 Sintetizzare in logica CMOS la funzione combinatoria:
$Y = X_0X_1+X_0X_2+X_1X_2$.

D3 Un canale dati costituito da tre linee seriali D, S, CK trasporta su D stringhe numeriche a N bit in complemento a 2 con pesi decrescenti; il primo bit di ogni sequenza numerica su D è segnalato da S=1; D e S sono sincronizzate dal segnale di clock CK. Progettare un sistema sequenziale che analizza le due linee e produce i tre bit di stato N, Z, P che indicano se il numero transitato sul canale è negativo, nullo e con parità pari rispettivamente.

D4 Calcolare la frequenza massima di funzionamento della rete sequenziale del punto D3 nell'ipotesi di disporre di porte combinatorie con t_p=5ns (tempo massimo di propagazione, positivo e negativo) e flip-flop con t_{setup}=4ns, t_{FF}=10ns (tempo massimo di commutazione) e t_{hold}=1ns.

D5 Scrivere una routine assembler per copiare il bit meno significativo di ogni elemento di un vettore di 32 byte predisposto nella RAM del PD32 nella longword successiva al vettore.

Esercizio (2S20010615-D1)

Indicare il valore dei numeri binari seguenti, espressi nel formato 8.4 in complemento a 2:

11111111.0000
11111111.1000
11111111.1100
11111111.1111

In tutti i numeri proposti il bit più pesante è 1, quindi i numeri rappresentati sono negativi. In tutti i quattro casi si può effettuare la complementazione a 2 del numero binario e la sua valutazione decimale; per il primo si ottiene:

11111111.0000_2 -> 00000001.0000_2 -> 1.0 da cui: $11111111.0000_2 = -1$

Analogamente per gli altri numeri si trova:

11111111.1000_2 -> 00000000.1000_2 -> 0.5 da cui: $11111111.1000_2 = -0.5$

11111111.1100_2 -> 00000000.0100_2 -> 0.25 da cui: $11111111.1100_2 = -0.25$

11111111.1111_2 -> 00000000.0001_2 -> 0.0625 da cui: $11111111.1111_2 = -0.0625$

Allo stesso risultato si può pervenire eseguendo la valutazione delle stringhe originali interpretandole come numeri positivi; ad esempio per il primo numero si ottiene:

$11111111.0000_2 = (2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0)_{10} = 255_{10}$

A questo punto, ricordando che con 8 bit di parte intera numeri rappresentati in complemento a 2 sono di fatto espressi in complemento a $2^8 = 256$, si riconosce che 255 (> 128) è la rappresentazione del numero negativo:
255 - 256 = -1.
Per gli altri tre numeri si procede analogamente.

Esercizio (2S20010615-D2)

Sintetizzare in logica CMOS la funzione combinatoria:
$Y = X_0X_1+X_0X_2+X_1X_2$.

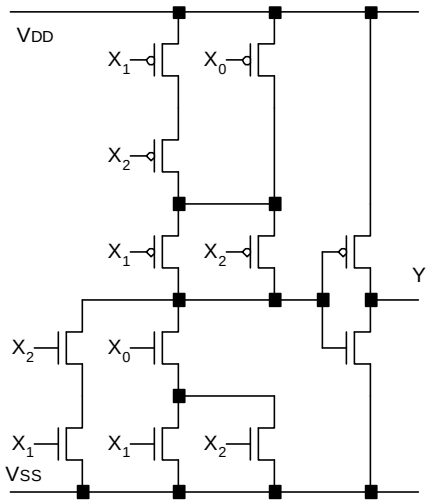
In logica CMOS complementare vanno sintentizzate le due funzioni diretta e negata, applicando possibilmente la fattorizzazione:

$Y = X_0 X_1 + X_0 X_2 + X_1 X_2 = X_0 (X_1 + X_2) + X_1 X_2$

applicando il teorema di De Morgan a quest'ultima espressione si ottiene subito la forma fattorizzata di Y* (il simbolo * indica la negazione):

$Y^* = (X_0^* + X_1^* X_2^*) (X_1^* + X_2^*)$

A questo punto si osserva che la funzione diretta ha tutti i letterali diretti e la funzione negata ovviamente li ha tutti negati; come è noto, in questo caso conviene scambiare le due funzioni, attribuendo Y* alla rete p-mos e Y alla rete n-mos, e introdurre un invertitore sull'uscita. A questo punto si può tracciare la rete di transistori che implementa la porta generalizzata richiesta:



Esercizio (2S20010615-D3)

Un canale dati costituito da tre linee seriali D, S, CK trasporta su D stringhe numeriche a N bit in complemento a 2 con pesi decrescenti; il primo bit di ogni sequenza numerica su D è segnalato da S=1; D e S sono sincronizzate dal segnale di clock CK. Progettare un sistema sequenziale che analizza le due linee e produce i tre bit di stato N, Z, P che indicano se il numero transitato sul canale è negativo, nullo e con parità pari rispettivamente.

La macchina sequenziale può essere descritta con il diagramma degli stati riportato in fig. 1, in cui le etichette dei bit di ingresso sono nell'ordine S,D e quelle dei bit di uscita N, Z, P:

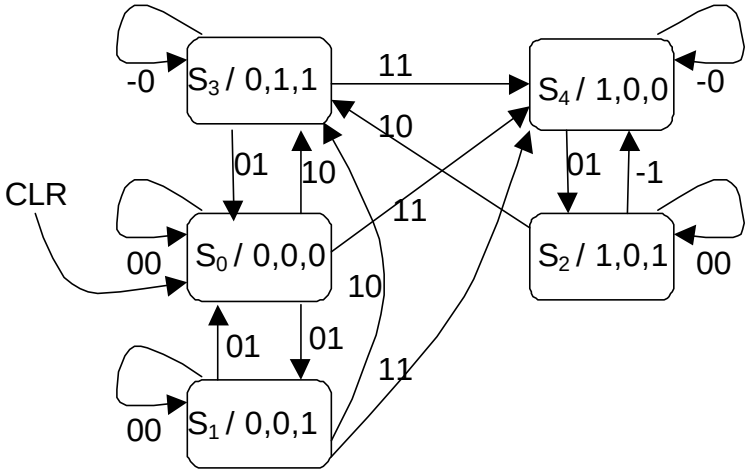


Fig. 1- Diagramma degli stati della macchina.

Con tale impostazione la macchina esprime il giudizio N, Z, P anche sulle stringhe parziali dei numeri che transitano sul canale. Se interessasse il giudizio soltanto sul numero finale, allora si potrebbe impostare una macchina di Mealy con le uscite definite soltanto sugli archi con ingressi 10 e 11. Si intende che i giudizi sui numeri completi vanno letti in modalità sincrona negli intervalli in cui S=1, in cui D rappresenta il primo bit (il segno) del nuovo numero.

Va notato che i vettori di uscita sono soltanto 5 e non 8, perché le tre configurazioni di uscita 010, 11- non sono possibili.

Conviene codificare i cinque stati con gli stessi codici dei segnali di uscita assegnati ai cinque stati definiti; infatti, in tal modo si evita la rete combinatoria delle uscite, in quanto queste potranno essere prelevate direttamente dai flip-flop del registro di stato. Con questa posizione si predispone per rete logica la forma riportata in fig. 2:

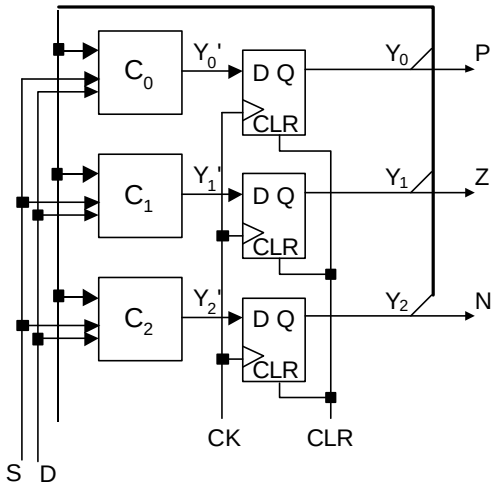


Fig. 2 - Struttura generale della rete.

Va notato che la specifica del sincronismo di D e S con CK consente di utilizzare i segnali D e S nel sistema sequenziale sincrono da progettare, senza sottoporli a risincronizzazione; infatti, D e S commutano per l'ipotesi specificata come se fossero prodotti da un sistema esterno sincrono con quello da progettare: nella fig. 3 è tracciato anche il diagramma temporale dei segnali entranti nell'ipotesi che vengano prelevati dall'uscita di un registro di uscita del generatore esterno. E' appena il caso di ribadire l'opportunità di sintetizzare la rete in modo sincrono, evitando quindi di applicare i segnali D e/o S a ingressi di CLR /SET /CK dei flip-flop nel sistema da progettare!

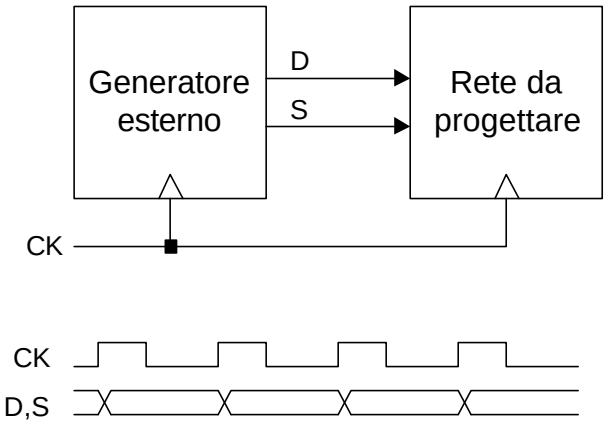


Fig. 3 - Sincronizzazione esterna dei segnali di ingresso.

Va anche notato che la lunghezza delle stringhe numeriche non è nota a priori, e la terminazione di una stringa è segnalata dalla presenza di S, nella cella di bit iniziale della stringa successiva (fig. 4).

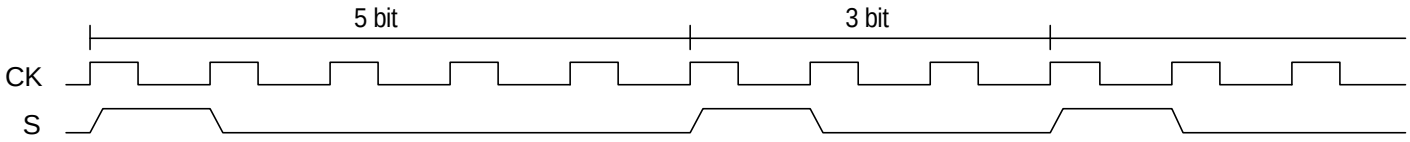


Fig. 4 – Esempio di andamento dei segnali di ingresso.

Con le posizioni precedenti la tavola degli stati e delle uscite assume la forma:

S	Y2 Y1 Y0	Y2' Y1' Y0' @ SD=00	Y2' Y1' Y0' @ SD=01	Y2' Y1' Y0' @ SD=11	Y2' Y1' Y0' @ SD=10	N Z P
S0	0 0 0	0 0 0	0 0 1	1 0 0	0 1 1	000
S1	0 0 1	0 0 1	0 0 0	1 0 0	0 1 1	001
S2	1 0 1	1 0 1	1 0 0	1 0 0	0 1 1	101
S3	0 1 1	0 1 1	0 0 0	1 0 0	0 1 1	011
S4	1 0 0	1 0 0	1 0 1	1 0 0	0 1 1	100
	0 1 0	- - -	- - -	- - -	- - -	- - -
	1 1 0	- - -	- - -	- - -	- - -	- - -
	1 1 1	- - -	- - -	- - -	- - -	- - -

La sintesi completa richiede la minimizzazione con 3 MK (le tre variabili di stato) a 5 variabili (le tre variabili di stato + i due ingressi), ed è riportata nella fig. 5.

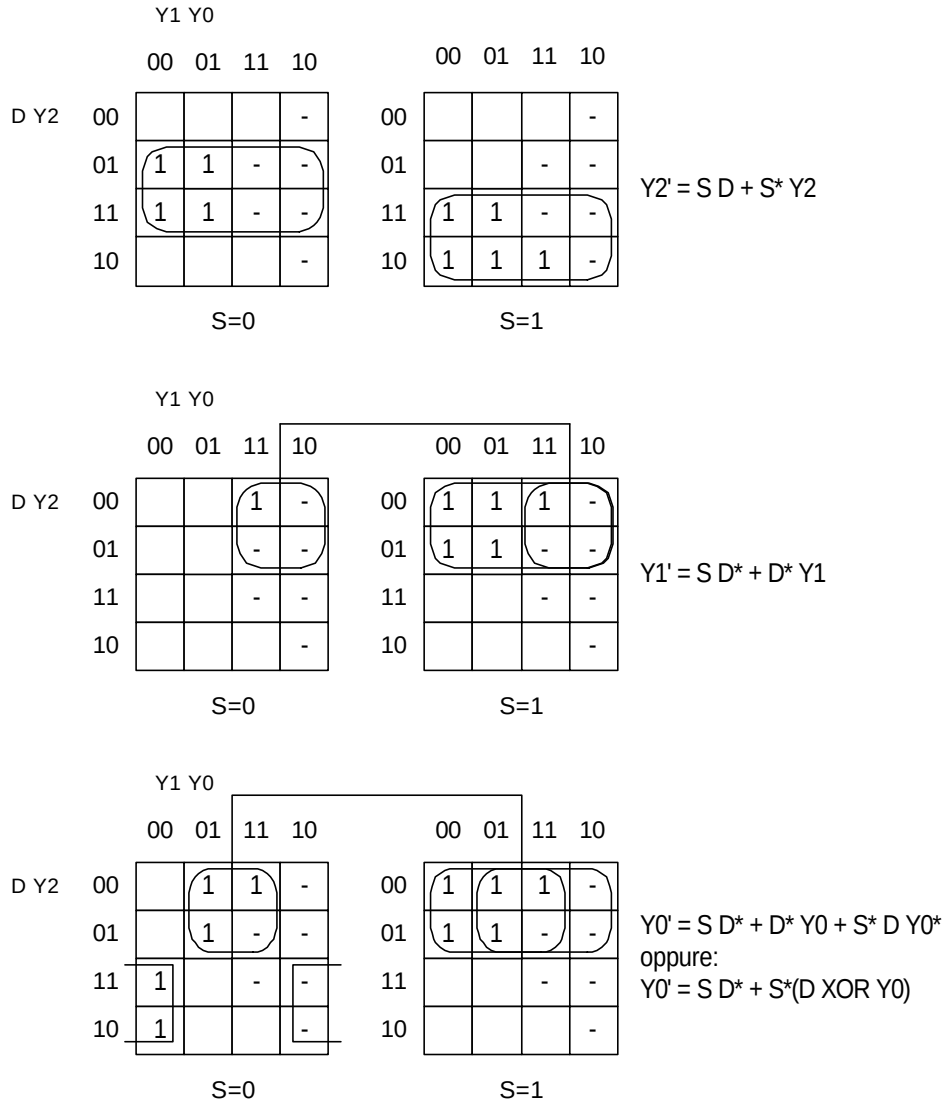


Fig. 5 – Minimizzazione sulle MK.

Va notata la particolare struttura delle equazioni restituite dalla metodologia di sintesi: ciascuna variabile di stato dipende dai due ingressi e soltanto dalla componente dello stato rappresentata da se stessa; questo risultato significa che le variabili di stato sono disaccoppiate tra loro e quindi descrivono funzioni sequenziali indipendenti tra di esse. In effetti il risultato dell'indipendenza tra N, Z, P ottenuto per via analitica riflette la possibilità di calcolare le funzioni N, Z e P individualmente. Verrà mostrato più avanti come questa considerazione possa essere utilizzata per affrontare la sintesi del sistema richiesto in un modo più diretto. Va anche notato che il disaccoppiamento delle componenti dello stato è un beneficio che consegue dalla codifica degli stati con gli stessi codici di N,Z,P.

Un'altra osservazione sulla struttura delle equazioni: tutte e tre le espressioni AND-OR contengono un implicante che include S in forma diretta e anche la variabile D, ma non le variabili di stato: tale implicante è responsabile dell'inizializzazione del flip-flop di stato all'inizio della stringa (S=1), e la struttura dell'implicante riflette il fatto che l'inizializzazione dei flip-flop di stato (P, N, Z) deve dipendere soltanto dal valore D (primo bit della nuova stringa), e non dalla memoria della stringa precedente.

La rete logica completa e dettagliata è riportata nella fig. 6, in cui è evidente l'influenza delle singole componenti di stato soltanto su se stesse.

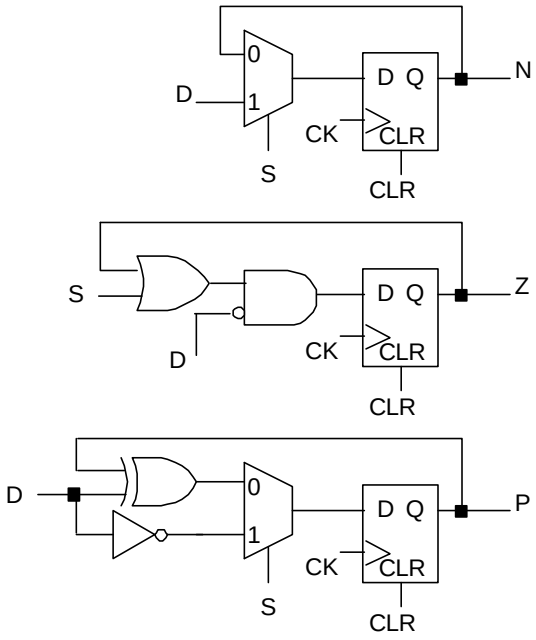


Fig. 6 – Rete logica dettagliata.

I risultati analitici possono essere interpretati da un punto di vista funzionale:

- il bit N viene inizializzato con il bit D iniziale (S=1): quindi il flip-flop memorizza D quando S=1, mantiene lo stato quando D=0; pertanto il flip-flop è alimentato da un MUX controllato da S e con l'ingresso sull'uscita del flip-flop stesso e l'ingresso 1 su D; si riconosce facilmente che la struttura coincide con un flip-flop D dotato di abilitazione al caricamento (LE), con LE=S, D=D, Q=N;
- il bit Z viene inizializzato con il bit D* quando S=1; in seguito, quando S=0, il valore Z=1 è mantenuto se e solo se Z=1 e D=0;
- il bit P viene inizializzato con il bit D* quando S=1; in seguito, quando S=0, il valore di P viene commutato ogni volta che D=1.

Procedimento alternativo

D'altra parte, le ultime considerazioni esposte possono essere effettuate *a priori* per progettare la rete richiesta, una volta riconosciuta l'indipendenza delle tre funzioni N, Z, P, senza passare per la descrizione con il diagramma degli stati. Infatti, ciascuna delle tre reti sarà del tipo rappresentato in fig. 7, dove si è scelto di utilizzare un flip-flop di tipo D:

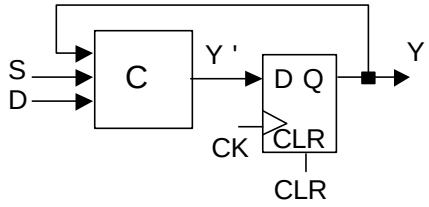


Fig. 7 – Struttura della rete sequenziale che produce il singolo bit.

Il progetto della rete sequenziale si è ridotto a quello della rete combinatoria C, il cui calcolo può essere impostato con la tavola di verità:

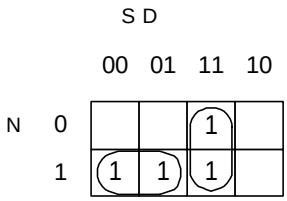
Bit N

S	D	N	N'
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Se S=0 (bit interno alla stringa) il bit N non può cambiare; quindi N'=N.

Se S=1 al bit N viene assegnato il valore di D (è il primo bit della stringa): quindi N'=D.

Passando alla minimizzazione sulla MK si ottiene:



$N' = S\,D + S^*\,N$

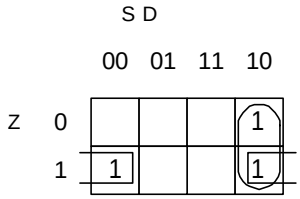
che è la stessa equazione ottenuta con il primo procedimento (cfr. fig. 5).

Bit Z

S	D	Z	Z'
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Se S=0 (bit interno alla stringa) il bit Z=1 viene mantenuto se e solo se D=0; quindi Z'=ZD*.
Se S=1 al bit Z viene assegnato il valore di D* (è il primo bit della stringa): quindi Z'=D*.

Passando alla minimizzazione sulla MK si ottiene:



$Z' = S\,D^* + Z\,D^*$

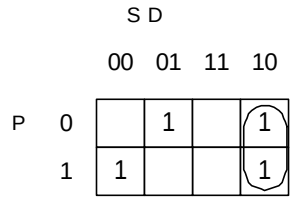
che è la stessa equazione ottenuta con il primo procedimento (cfr. fig. 5).

Bit P

S	D	P	P’
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Se S=0 (bit interno alla stringa) il bit P viene commutato se D=1: quindi P’=D XOR P.
Se S=1 il bit P viene inizializzato con D* (è il primo bit della stringa): quindi P’=D*.

Passando alla minimizzazione sulla MK si ottiene:



$P' = S\,D^* + S^*\,(D \text{ XOR } P)$

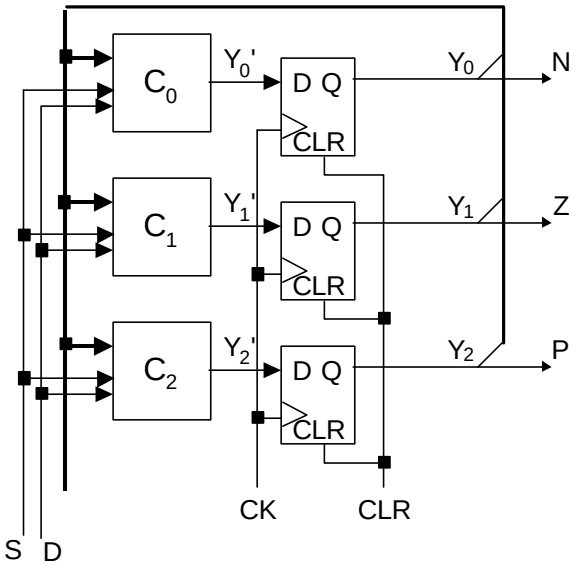
che è la stessa equazione ottenuta con il primo procedimento (cfr. fig. 5); anche in questo caso si è preferito scrivere la funzione P mediante lo XOR, piuttosto che la forma minima.

In conclusione si sono riottenute in tutti e tre i casi le stesse equazioni e quindi le stesse reti logiche del metodo precedente.

Esercizio (2S20010615-D4)

Calcolare la frequenza massima di funzionamento della rete sequenziale del punto D3 nell'ipotesi di disporre di porte combinatorie con t_p=5ns (tempo massimo di propagazione, positivo e negativo) e flip-flop con t_{setup}=4ns, t_{FF}=10ns (tempo massimo di commutazione) e t_{hold}=1ns.

La rete logica cui si fa riferimento ha la struttura seguente:



dove le reti combinatorie C0, C1, C2 sono ottimizzate e quindi costituite da tre livelli di logica. Pertanto il periodo di CK deve soddisfare la disequazione seguente:

$T_{CK} > t_{FF} + t_C + t_{setup} = t_{FF} + 3\,t_p + t_{setup} = (10 + 15 + 4)$ ns = 29 ns.

Va notato che il tempo di hold non ha alcuna funzione nel calcolo di T_{CK}; t_{hold} è completamente assorbito da t_{FF}, in quanto entrambi iniziano sul fronte efficace di CK.

Esercizio (2S20010615-D5)

Scrivere una routine assembler per copiare il bit meno significativo di ogni elemento di un vettore di 32 byte predisposto nella RAM del PD32 nella longword successiva al vettore.

```
;copybits.asm

;copia il bit meno significativo di ogni elemento di un vettore di 32 byte
;predisposto nella RAM del PD32 nella longword successiva al vettore.

        org 400h                ;inizio programma

; *************************
; COSTANTI
; *************************

        STACK       equ 2800h   ;inizio area di stack

; *************************
; VARIABILI
; *************************

        LW0   DL 0F00FF00h
        LW1   DL 0F00FF01h
        LW2   DL 0F00FF00h
        LW3   DL 0F00FF01h
        LW4   DL 0F00FF01h
        LW5   DL 0F00FF01h
        LW6   DL 0F00FF01h
        LW7   DL 0F00FF01h
        LW8   DL 0F00FF00h
        LW9   DL 0F00FF00h
        LW10 DL 0F00FF00h
        LW11 DL 0F00FF00h
        LW12 DL 0F00FF01h
        LW13 DL 0F00FF01h
        LW14 DL 0F00FF01h
        LW15 DL 0F00FF01h
        LW16 DL 0F00FF00h
        LW17 DL 0F00FF01h
        LW18 DL 0F00FF00h
        LW19 DL 0F00FF01h
        LW20 DL 0F00FF01h
        LW21 DL 0F00FF01h
        LW22 DL 0F00FF01h
        LW23 DL 0F00FF01h
        LW24 DL 0F00FF00h
        LW25 DL 0F00FF00h
        LW26 DL 0F00FF00h
```

```
        LW27 DL 0F00FF00h
        LW28 DL 0F00FF01h
        LW29 DL 0F00FF01h
        LW30 DL 0F00FF00h
        LW31 DL 0F00FF00h


        code                    ;inizio istruzioni

main:
        movl #STACK,r7          ;inizializza R7 quale SP
        seti                    ;abilita interruzioni (SP è stato inizializzato)

        jsr copybits

        halt                    ;arresta elaborazione


; ************************
; SUBROUTINES
; ************************

copybits:
        push r0                 ;salva r0: contatore
        push r1                 ;salva r1: puntatore al buffer
        push r2                 ;salva r2: appoggio dato
        push r3                 ;salva r3: costruisce dato finale

        movb #32,r0             ;inizializza r0 al conteggio di 32
        movl #LW0,r1

cpyb:   movl (r1)+,r2
        asrl #1,r2              ;bit r2.0 -> carry
        rcrl #1,r3              ;carry -> bit r3.31
        subb #1,r0
        jnz cpyb
        movl r3,(r1)            ;risultato -> RAM

        pop r3
        pop r2
        pop r1
        pop r0
        ret

        end                     ;fine programma
```

```
;copybits.asm

;copia il bit meno significativo di ogni elemento di un vettore di 32 byte
;predisposto nella RAM del PD32 nella longword successiva al vettore.



  org 400h    ;inizio programma

; ************************
; COSTANTI
; ************************

  STACK   equ 2800h ;inizio area di stack

; ************************
; VARIABILI
; ************************

  LW0 DL 0F00FF00h
  LW1 DL 0F00FF01h
  LW2 DL 0F00FF00h
  LW3 DL 0F00FF01h
  LW4 DL 0F00FF01h
  LW5 DL 0F00FF01h
  LW6 DL 0F00FF01h
  LW7 DL 0F00FF01h
  LW8 DL 0F00FF00h
  LW9 DL 0F00FF00h
  LW10  DL 0F00FF00h
  LW11  DL 0F00FF00h
  LW12  DL 0F00FF01h
  LW13  DL 0F00FF01h
  LW14  DL 0F00FF01h
  LW15  DL 0F00FF01h
  LW16  DL 0F00FF00h
  LW17  DL 0F00FF01h
  LW18  DL 0F00FF00h
  LW19  DL 0F00FF01h
  LW20  DL 0F00FF01h
  LW21  DL 0F00FF01h
  LW22  DL 0F00FF01h
  LW23  DL 0F00FF01h
  LW24  DL 0F00FF00h
  LW25  DL 0F00FF00h
  LW26  DL 0F00FF00h
  LW27  DL 0F00FF00h
  LW28  DL 0F00FF01h
  LW29  DL 0F00FF01h
  LW30  DL 0F00FF00h
  LW31  DL 0F00FF00h


  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)

  jsr copybits

  halt      ;arresta elaborazione


; ************************
; SUBROUTINES
; ************************
```

```
copybits:
  push r0     ;salva r0: contatore
  push r1     ;salva r1: puntatore al buffer
  push r2     ;salva r2: appoggio dato
  push r3     ;salva r3: costruisce dato finale

  movb #32,r0   ;inizializza r0 al conteggio di 32
  movl #LW0,r1

cpyb: movl (r1)+,r2
  asrl #1,r2    ;bit r2.0 -> carry
  rcrl #1,r3    ;carry -> bit r3.31
  subb #1,r0
  jnz cpyb
  movl r3,(r1)    ;risultato -> RAM

  pop r3
  pop r2
  pop r1
  pop r0
  ret

  end     ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 28-6-2001

Studente:_____ **Docente:**_____

Una periferica di un PD32, denominata S-DES, viene adibita alla cifratura di messaggi. Il PD32 carica un messaggio in memoria a partire da un indirizzo noto allineato alla longword. I primi 32 bit indicano la lunghezza del messaggio. Inoltre il PD32 deve inviare a S-DES una chiave di cifratura K a 16 bit.
Non appena il PD32 invia la chiave a S-DES, quest'ultimo avvia il processo di cifratura.

S-DES ha una struttura a quattro stadi di pipeline. Una volta attivato, S-DES preleva il messaggio dalla memoria in DMA con accesso a burst e a longword che, a partire dalla seconda, carica in due registri da 16 bit D_0 e S_0 che alimentano la pipeline. Lo stadio i-esimo (i=1..4) della pipeline include una coppia di registri a 16 bit D_i e S_i ed esegue le seguenti operazioni:

$D_i = S_{i-1}$
$S_i = F(S_{i-1} \text{ XOR } K_i) \text{ XOR } D_{i-1}$

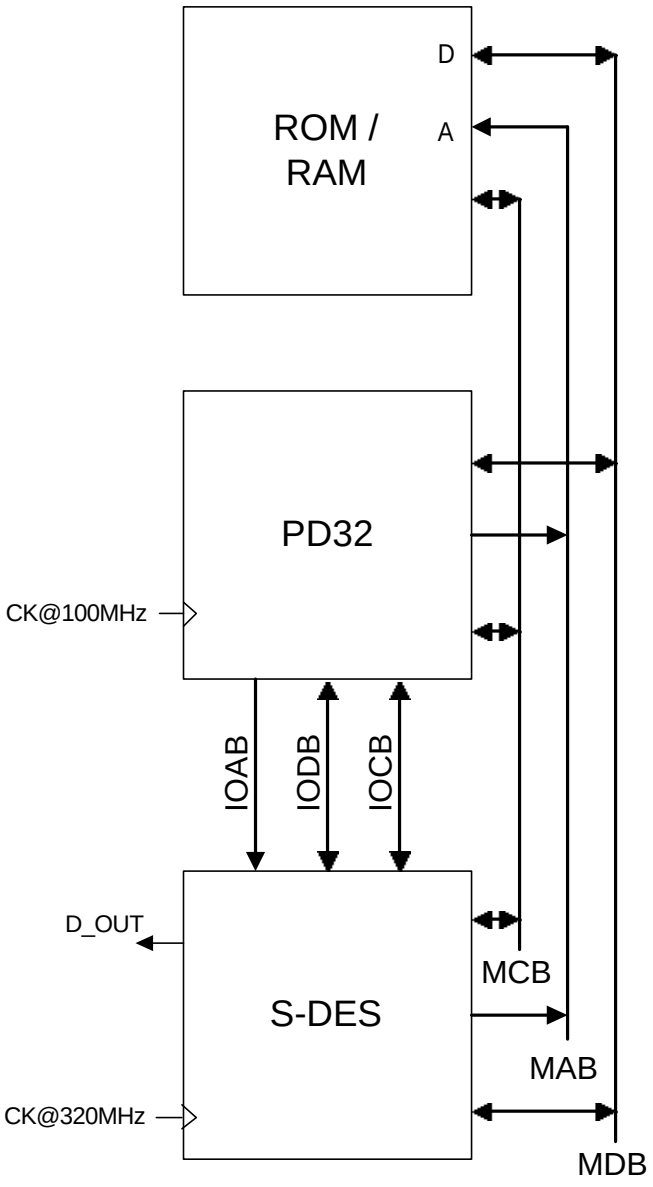
Il valore di K_i si ottiene ruotando di 4bit verso sinistra il valore della chiave K_{i-1}. Il valore di K_0 è uguale a K.

I registri D_4 e S_4 all'uscita della pipeline contengono i 32 bit cifrati. Tali bit devono essere inviati su una linea seriale ad una velocità di 320 Mbit/sec. Il ritardo delle porte logiche è di 10 ns, il tempo di commutazione dei registri è di 10 ns e quello di set-up è di 2 nsec. La funzione F viene implementata attraverso moduli di ROM il cui tempo di accesso è di 25 ns. Il clock del PD32 è di 100Mhz. Alla fine del trasferimento S-DES invia un interrupt al PD32.

Si richiede:

1. lo schema logico di S-DES;
2. il diagramma dettagliato di timing che ne specifichi il funzionamento.

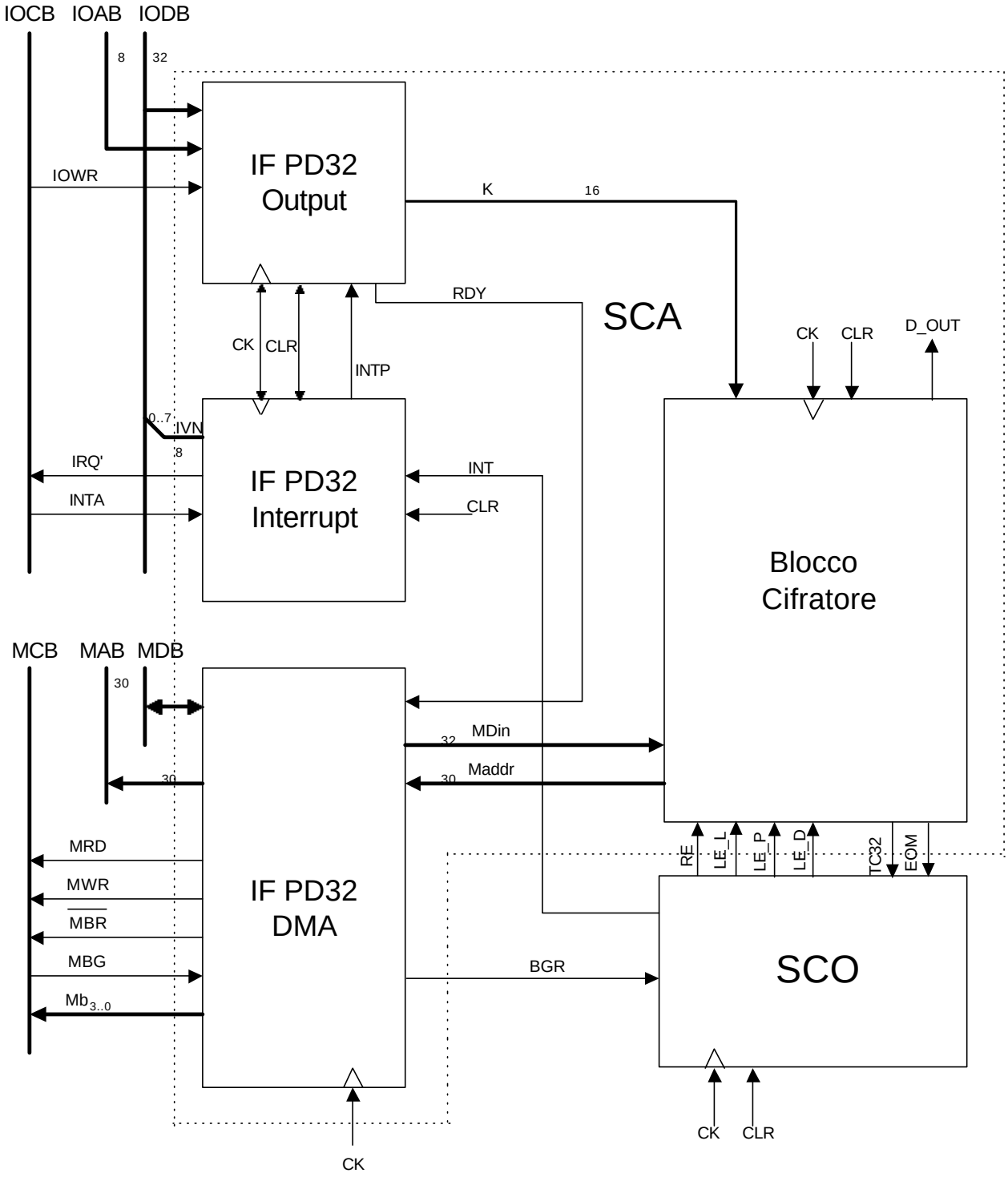
S-DES: sistema esterno



Note

La specifica impone di serializzare i dati prelevati dalla memoria, e trattati secondo l'algoritmo specifico di S-DES, alla velocità di 320 Mbit/s. Questa è ovviamente la velocità più elevata alla quale le varie sezioni della periferica si muovono, e quindi per poter lavorare la periferica deve disporre di un clock alla frequenza di 320 MHz. Questo sarà IL clock della periferica, le cui dinamiche più lente (ad esempio, il caricamento del registro PISO finale che alimenta la linea seriale) saranno ricavate con segnali di abilitazione (al caricamento, al conteggio, ecc.) ricavati da divisori di frequenza (tipicamente, Terminal Count di contatori).

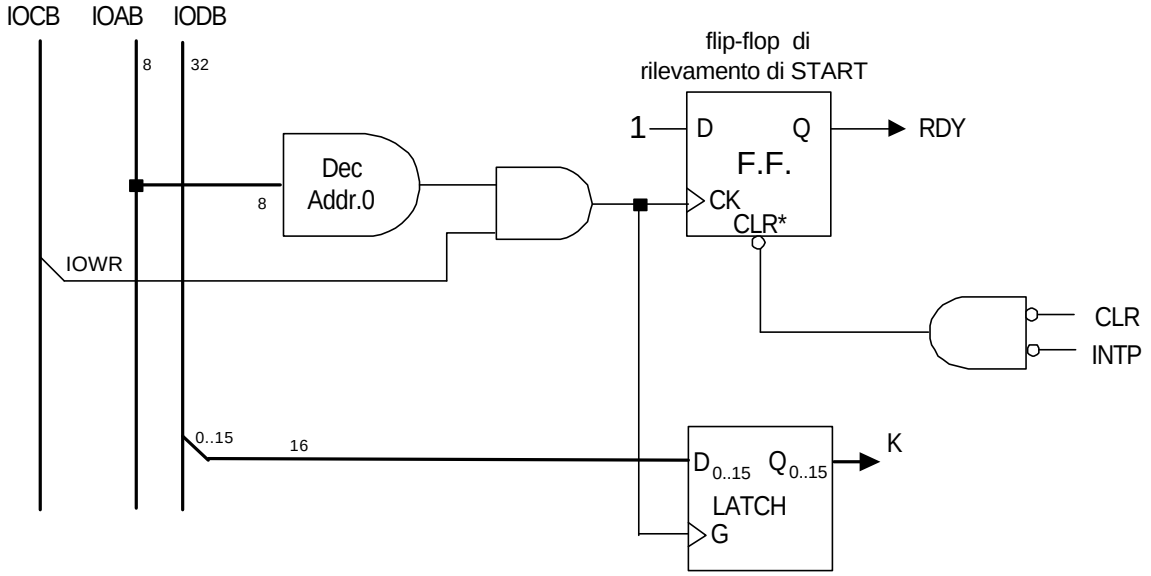
S-DES: Schema a blocchi



Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.
Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

S-DES: IF PD32 - output

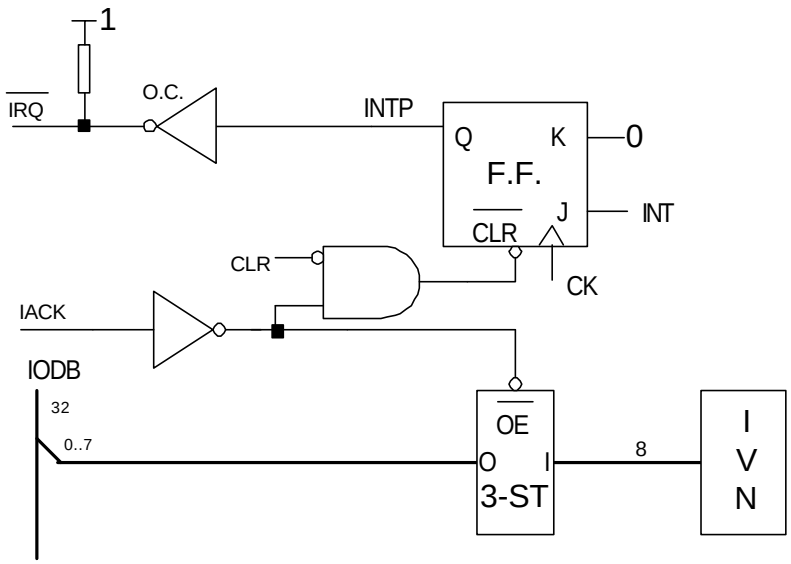


Note

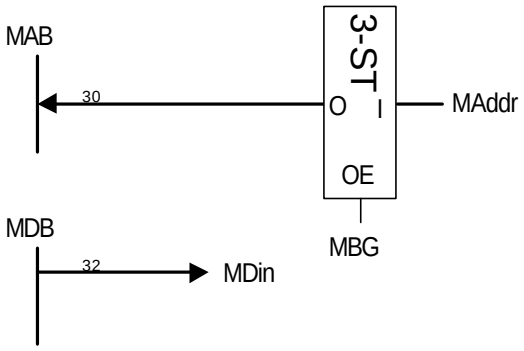
Il SW avvia l'operazione con **OUTW key,Addr0**, dove key è una variabile di tipo word che memorizza la chiave.

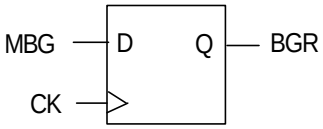
Il segnale GO non viene ricampionato, in quanto non viene letto dalla periferica, ma inviato direttamente alla linea MBR del processore. Ovviamente MBG verrà letto dalla periferica e quindi, al solito, sarà ricampionato.

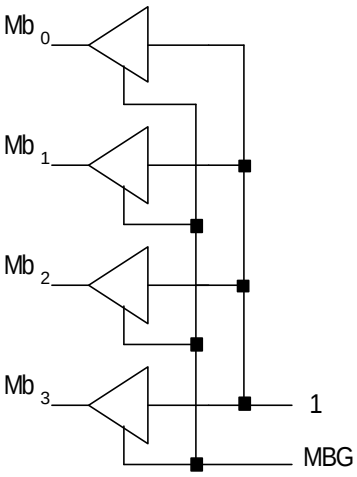
CFR: IF PD32 - interrupt

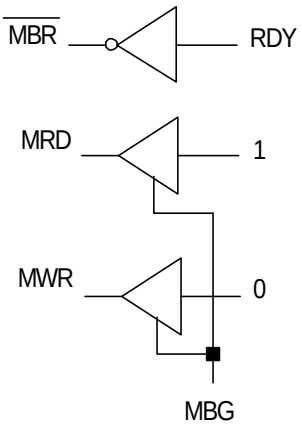


CFR: IF PD32 - DMA





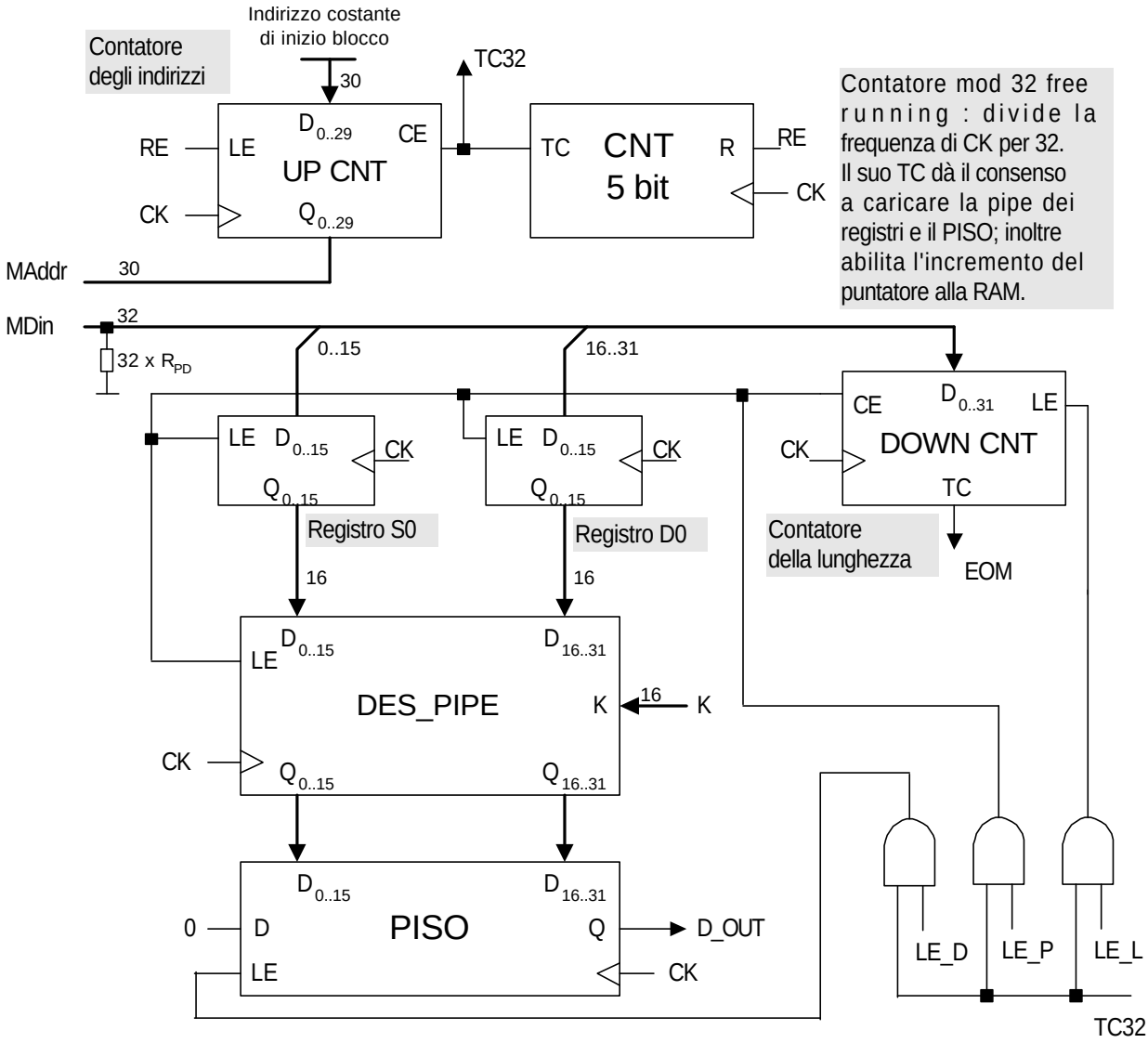




Dalla RAM vengono lette LW.

MWR deve essere pilotato (a 0), anche se non dinamicamente: la struttura di memoria deve essere governata pienamente dall'interfaccia DMA, che ne ha piena responsabilità quando MBG=1.

S-DES: blocco cifratore



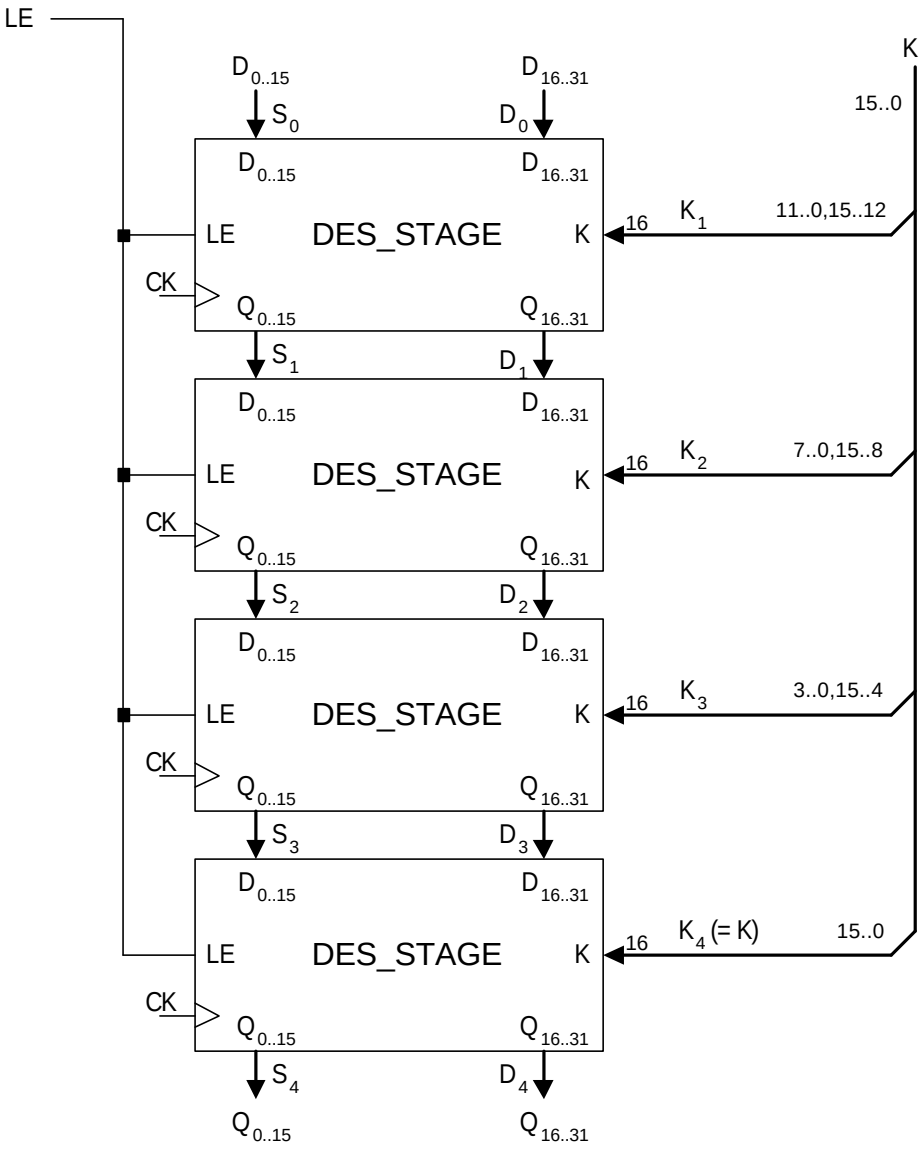
Note

La specifica impone di serializzare i dati prelevati dalla memoria con larghezza di 32 bit senza soluzione di continuità alla velocità di 320 Mbit/s. Pertanto occorre verificare la possibilità di prelevare dalla memoria 10 MLW per secondo, cioè un dato (in particolare una longword) ogni 100 ns; pertanto occorre verificare che il tempo di accesso in lettura della memoria sia inferiore a 100 ns: dalla specifica si sa che il micro lavora con quella stessa memoria con un clock di frequenza 100 MHz, accedendovi con cicli di lettura/scrittura di 3 periodi di clock, quindi della durata di 30 ns. Si conclude che la velocità di linea prescritta è compatibile con le prestazioni della memoria del sistema.

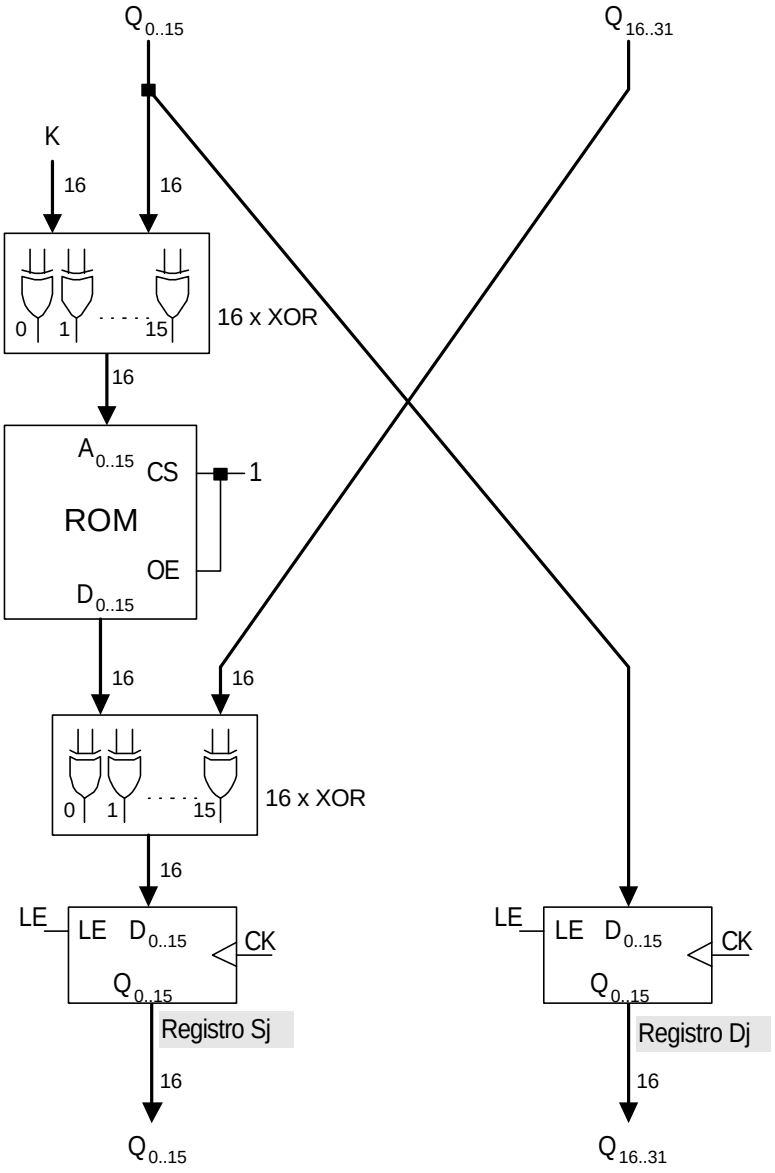
La periferica deve accedere alla memoria in DMA, modalità bur st, e in sola lettura; inoltre deve prelevare un dato (a 32 bit) ogni 32 periodi del suo clock (a 320 MHz). Verificato che 32 periodi di CK hanno una durata (100 ns) superiore al tempo di ciclo di lettura (30 ns) della memoria, la linea MRD può essere attivata per l'intero accesso in DMA, limitandosi a commutare gli indirizzi della memoria e caricarne i dati in un registro ogni 32 colpi di CK (cfr. diagramma di temporizzazione).

Il registro PISO di uscita potrebbe essere lo stesso registro S3-D3 dello stadio di uscita della pipeline. Si è preferito mantenerli distinti per ragioni di modularità.

S-DES: blocco DES_PIPE



S-DES: blocco DES_STAGE

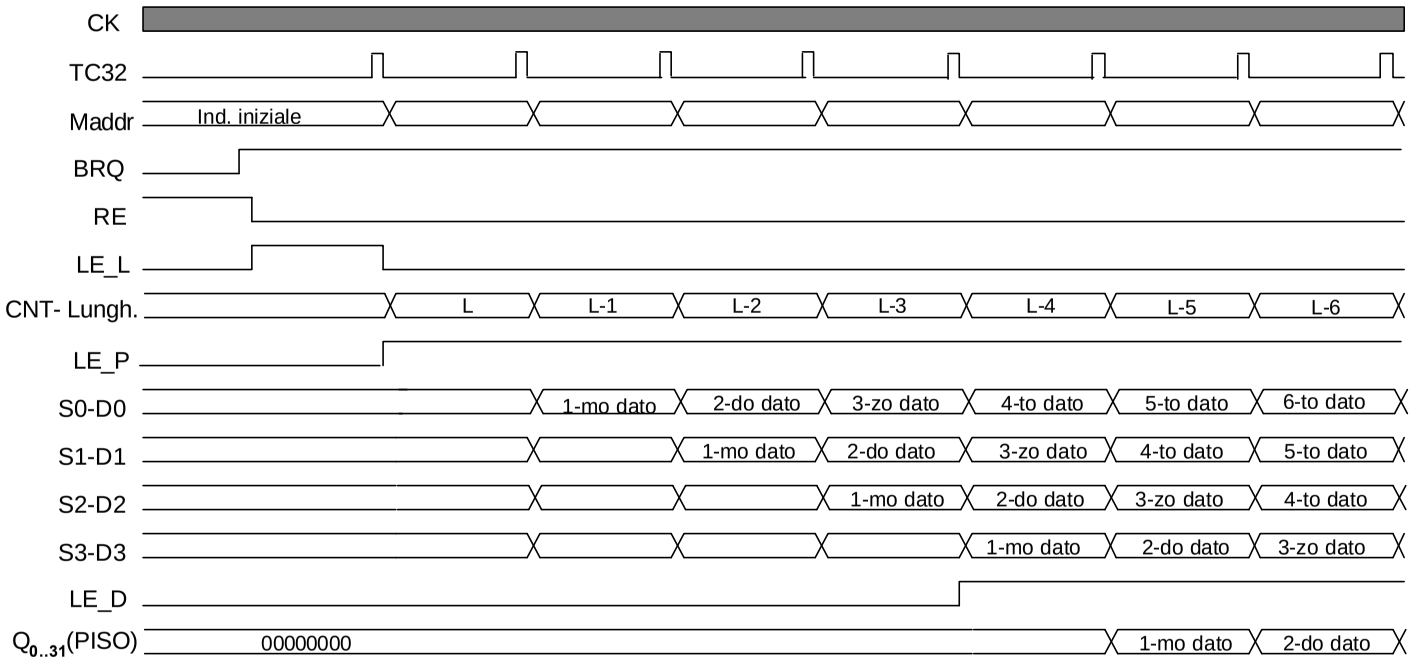


Note

Il tempo di elaborazione della catena combinatoria vale:
$t_{REG} + 2\ t_{XOR} + t_{ROM} + t_{setup}$ = (10 + 2x3x10 + 25 + 2)ns = 97 ns
dove si è considerato un tempo di ritardo di 3 porte logiche per lo XOR.
Il tempo di elaborazione calcolato è inferiore al tempo richiesto per serializzare 32 bit sulla linea di uscita alla velocità di 320 Mbit/s, che ammonta a 100 ns. Pertanto la struttura predisposta è consistente con la velocità di CK.

S-DES: temporizzazioni

Temporizzazioni dei cicli di lettura: transitorio iniziale

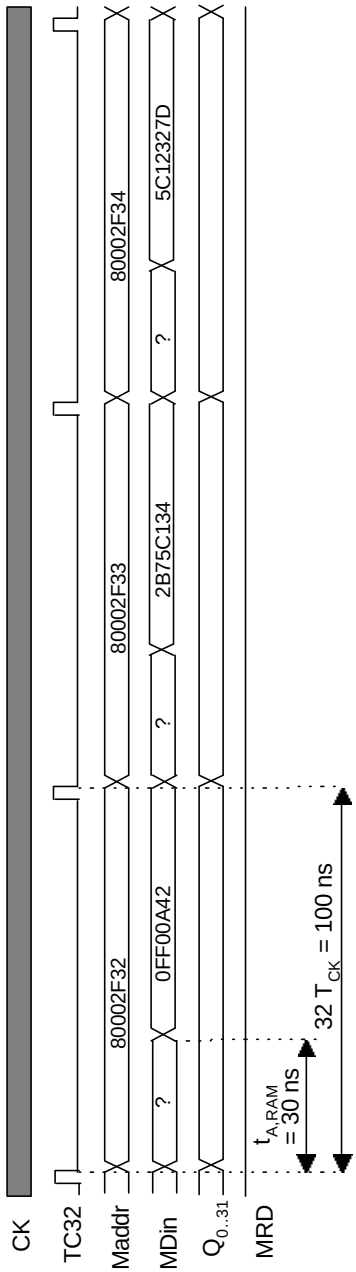


Note

Il transitorio iniziale è costituito da un (inevitabile) ritardo di latenza nella presentazione del primo dato elaborato; durante tale intervallo lo SCO deve mantenere a zero le uscite del registro PISO di uscita; quindi deve impedirne il caricamento con dati non significativi attivando il segnale LE_D con un ritardo indicato proprio dal diagramma di temporizzazione; di tale indicazione si dovrà tenere conto nel diagramma di flusso dello SCO.

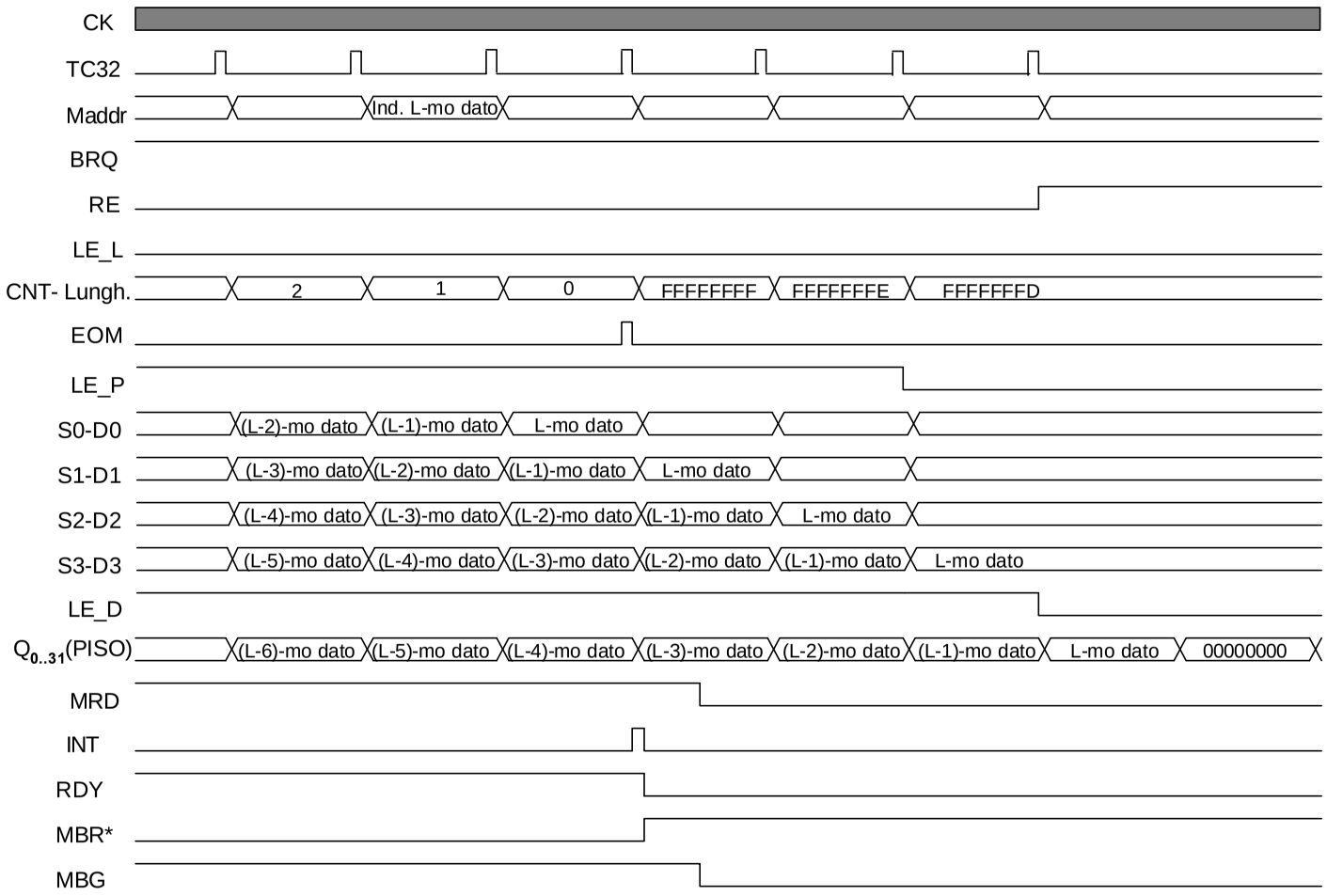
S-DES: temporizzazioni

Temporizzazioni dei cicli di lettura a regime



S-DES: temporizzazioni

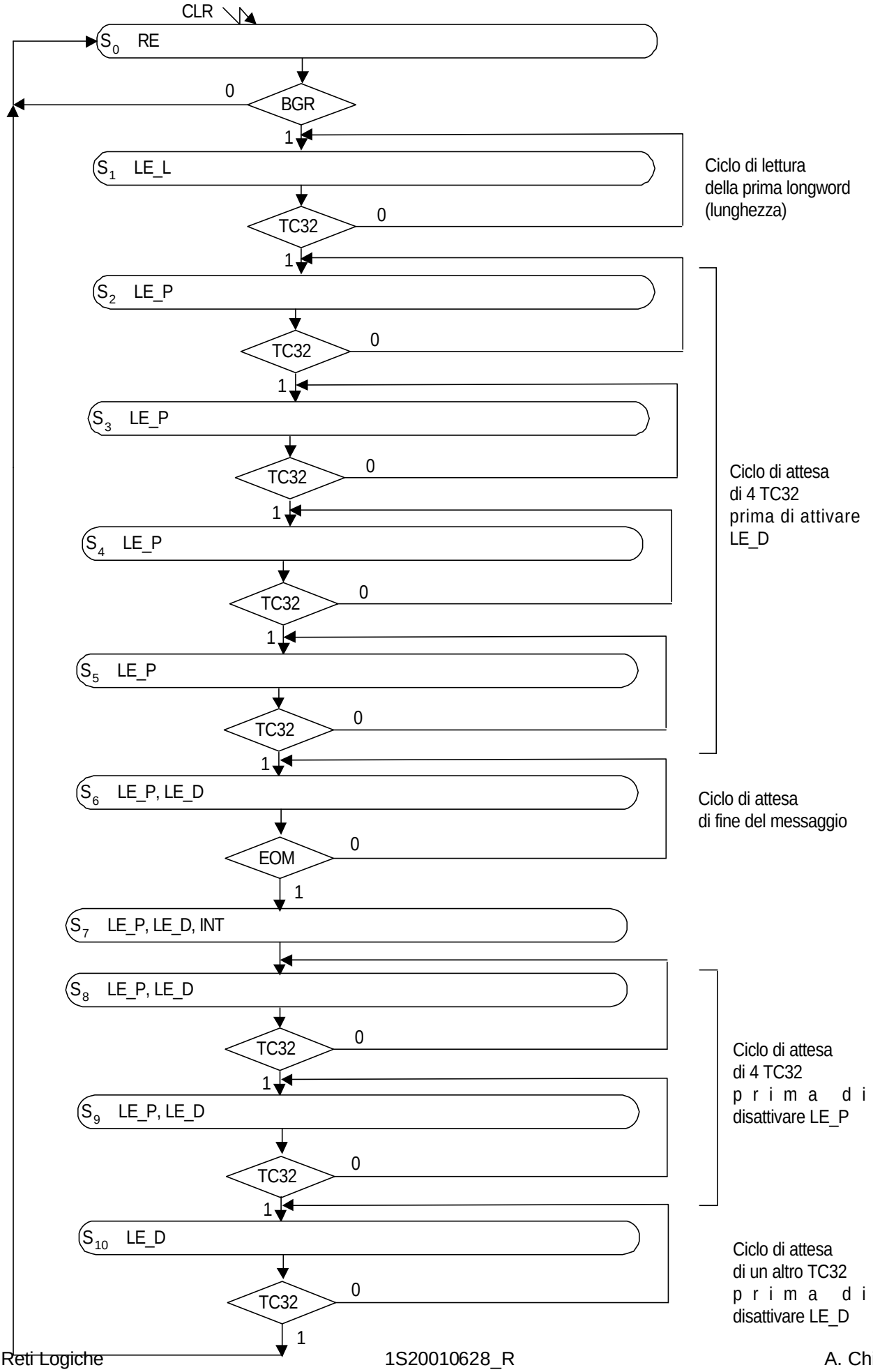
Temporizzazioni dei cicli di lettura: transitorio finale



Note

Nel transitorio finale lo SCO deve aspettare l'attraversamento della pipeline dell'ultimo dato del messaggio; perciò deve aspettare l'emissione dell'ultimo dato di uscita prima di tornare nello stato iniziale in cui disabilita il caricamento del registro PISO di uscita.

S-DES: SCO - flowchart



S-DES: SCO - struttura HW

- di tipo Moore
- a sequenziatore: linearità del diagramma degli stati

Calcolo dei parametri del circuito

· Equazione di abilitazione all'incremento del conteggio (stato):

CE = S0 BGR + (S1+S2+S3+S4+S5+S8+S9) TC32 + S7 + S6 EOM

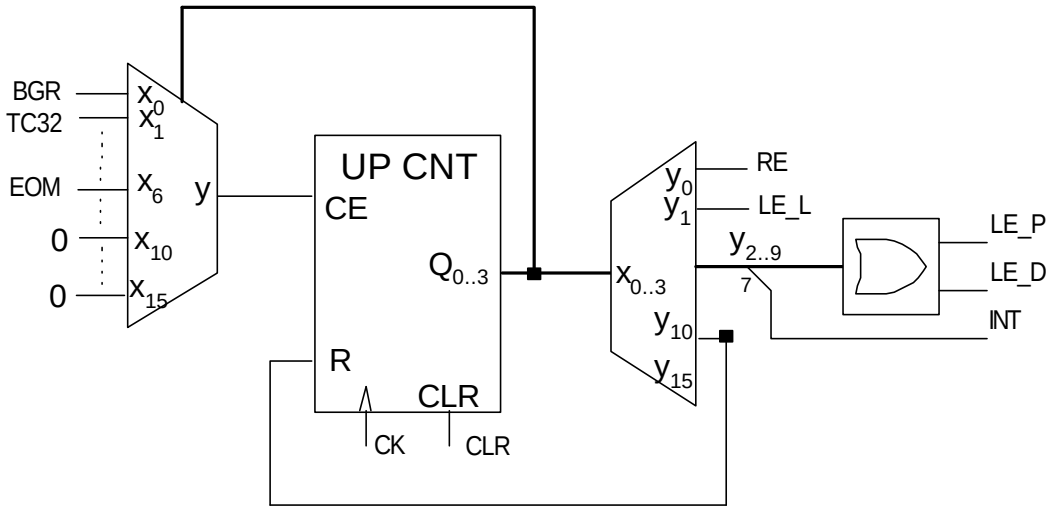
· Equazione di azzeramento del contatore:

R = S10

· Equazioni delle variabili di uscita:

RE = S0
LE_L = S1
LE_P = S2+S3+S4+S5+S6+S7+S8+S9
LE_D = S6+S7+S8+S9+S10
INT = S7

Circuito dello SCO



RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 28-6-2001

STUDENTE:_____DOCENTE:_____

D1 Determinare il numero delle possibili funzioni booleane $f(x_0,x_1,..,x_{N-1})$ vincolate da f(0,0,..,0)=1 e f(1,1,..,1)=0.

D2 Progettare una rete iterativa che calcoli il valore assoluto |X| di un numero X espresso in complemento a 2 su N bit.

D3 In un canale dati una linea seriale D sincronizzata da un segnale CK trasporta caratteri ASCII a 7 bit separati da un bit a 0. Progettare una rete sequenziale sincrona che segnali con un bit di uscita SYNC la posizione dei bit di separazione (D=0) intervallati da 8 Tck; nello stato iniziale e in caso D=1 sull'ottavo bit (mancato allineamento dei byte) la condizione di allineamento deve essere ricercata a partire dal primo 0 successivo.

D4 Calcolare la frequenza massima di funzionamento di un sistema SCO-SCA di tipo D-Mealy – D-Mealy in funzione dei parametri delle due reti.

D5 Descrivere il contenuto dei frammenti ritenuti significativi della mappa della memoria del PD32 dopo l'elaborazione da parte dell'assemblatore del codice seguente:

```
driver 100,10000000h
rti
```

Esercizio (2S20010628-D1)

Determinare il numero delle possibili funzioni booleane $f(x_0,x_1,..,x_{N-1})$ vincolate da f(0,0,..,0)=1 e f(1,1,..,1)=0.

Come è noto, con N variabili binarie si possono definire 2^(2^N) funzioni booleane. Infatti, una funzione di N bit è definita su 2^N punti (0..00, 0..01, 0..10, 0..11, 1..11), su cui può assumere anch'essa due soli valori, in quanto booleana; pertanto le possibili configurazioni che può assumere una funzione booleana generica di N bit sono appunto 2^(2^N). Con il duplice vincolo specificato, una funzione di N bit non ha più 2^N gradi di libertà, in quanto su due punti è prefissata: nel caso specificato i gradi di libertà si riducono le possibili configurazioni che può assumere una funzione vincolata in due punti (non importa quali) sono: 2^(2^N - 2), e quindi tale è il numero delle funzioni richiesto. Tale valore può essere riscritto anche nella forma (2^(2^N))/4, che mette in evidenza che in presenza di un duplice vincolo il numero delle possibili funzioni booleane si riduce a ¼ del numero delle funzioni non vincolate.

Esercizio (2S20010628-D2)

Progettare una rete iterativa che calcoli il valore assoluto |X| di un numero X espresso in complemento a 2 su N bit.

La rete iterativa deve effettuare la complementazione a 2 del numero espresso dal vettore di ingresso $X = X_{N-1}\ X_{N-2}\ .\ .\ X_1\ X_0$ se e solo se $X_{N-1}=1$, altrimenti deve copiare il vettore di ingresso su quello di uscita. La rete può essere impostata in modo da implementare l'algoritmo iterativo di complementazione a 2, condizionatamente a $X_{N-1}=1$; questo comporta:

- il trasporto di un bit di comando di complementazione da una cella a quella successiva;
- la diffusione del bit $X_{N-1}=1$ a tutte le celle.

Consistentemente con le posizioni precedenti la struttura della rete iterativa (fig. 1) deve essere inizializzata con il carry-in della prima cella a 0 (inizializza la condizione di non complementazione di X_0). La diffusione del bit X_{N-1} viene impostata formalmente in modo iterativo mediante il trasporto filato del bit da una cella alla successiva. Va anche osservato che il vettore di uscita ha N-1 bit, uno in meno del vettore di ingresso, in quanto la specifica richiede la produzione del valore assoluto di X, quindi non interessa il bit di segno Y_{N-1}; in definitiva la rete iterativa è composta da N-1 celle.

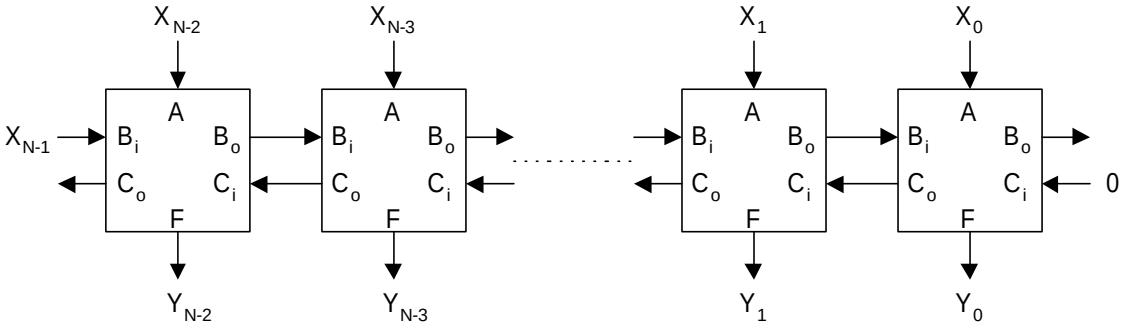


Fig. 1 - Struttura della rete iterativa.

Il progetto della cella è impostato con la tavola di verità che descrive le tre uscite B_o, C_o, F in funzione delle tre variabili di ingresso B_i, C_i, A:

B_i	C_i	A	B_o	C_o	F
0	0	0	0	-	0
0	0	1	0	-	1
0	1	0	0	-	0
0	1	1	0	-	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	0

Se $C_i=0$ (prime quattro righe della tavola) allora $C_o=0$; altrimenti viene propagato a valle della cella l'esito del confronto tra B_p e B: in questo caso $C_o=0$ (violazione) solo per $B_p=1$ & B=0 (coppia decrescente).

In cui si è impostato $B_o=B_i$ per modellare la diffusione del bit X_{N-1} a tutte le celle della rete mediante il trasporto filato del bit da una cella alla successiva.

La sintesi procede con la minimizzazione sulla MK:

C_iA / B_i	00	01	11	10
0	-	-	-	-
1		1	1	1

$C_o = A + C_i$

C_iA / B_i	00	01	11	10
0		1	1	
1		1		1

$F = B_i^*A + B_i\,(A \oplus Ci)$

dove * indica negazione.

All'espressione minima di F è stata preferita quella con l'operatore XOR che mette in evidenza la proprietà strutturale dell'algoritmo.

Da cui segue la rappresentazione grafica della rete logica della cella (fig. 2):

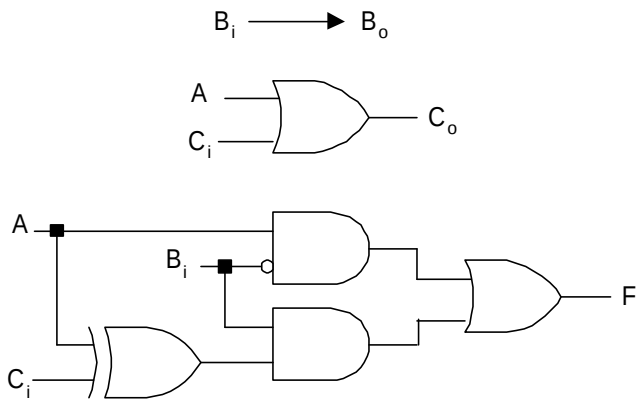
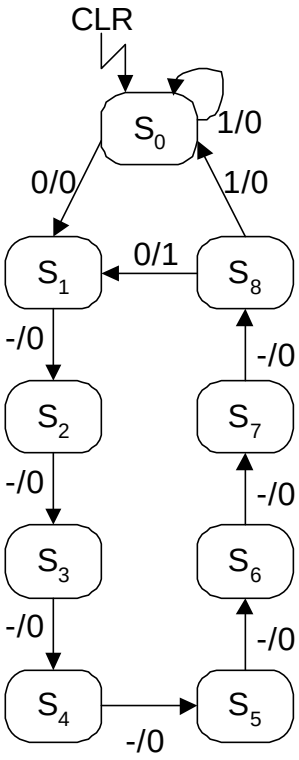


Fig. 2 – Struttura della cella.

Esercizio (2S20010628-D3)

In un canale dati una linea seriale D sincronizzata da un segnale CK trasporta caratteri ASCII a 7 bit separati da un bit a 0. Progettare una rete sequenziale sincrona che segnali con un bit di uscita SYNC la posizione dei bit di separazione (D=0) intervallati da 8 Tck; nello stato iniziale e in caso D=1 sull'ottavo bit (mancato allineamento dei byte) la condizione di allineamento deve essere ricercata a partire dal primo 0 successivo.

La macchina sequenziale può essere descritta con il diagramma degli stati seguente, in cui le etichette sugli archi si riferiscono ai valori dei bit D/SYNC, secondo il modello di Mealy.

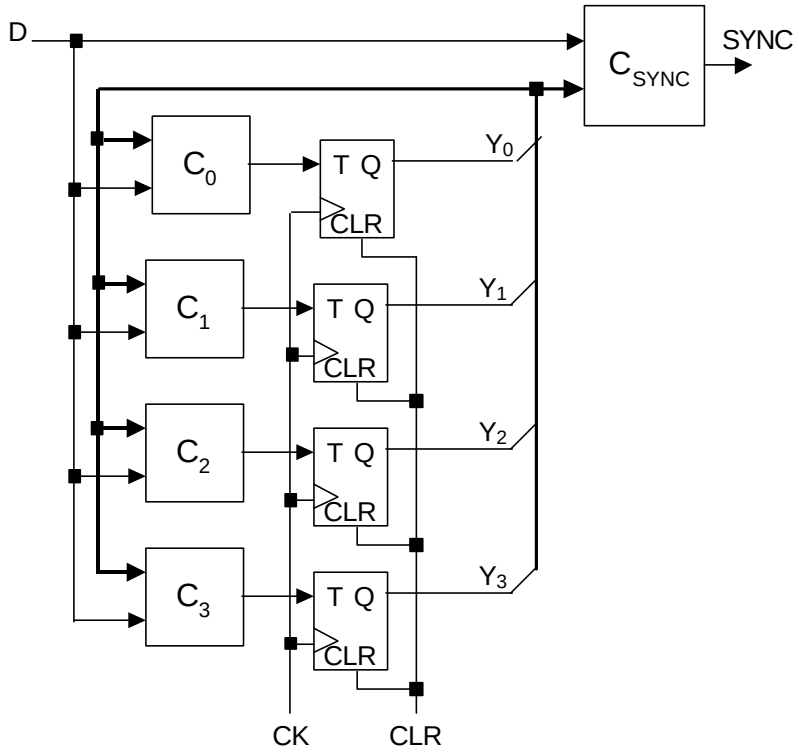


La macchina è minima, in quanto gli otto stati S1..S8 sono necessari per separare il riconoscimento di D=0 a distanza 8 T_{CK}; inoltre, lo stato S0 ha il ruolo diverso di ricercare il primo bit D=0 nella fase di aggancio alla trama di ingresso.

La tavola degli stati e delle uscite assume la forma:

S	Y3..0	Y'3..0 /SYNC @ D=0	Y'3..0 /SYNC @ D=1	T3..0 @ D=0	T3..0 @ D=1
S0	0 0 0 0	0 0 0 1 / 0	0 0 0 0 / 0	0001	0000
S1	0 0 0 1	0 0 1 0 / 0	0 0 1 0 / 0	0011	0011
S2	0 0 1 0	0 0 1 1 / 0	0 0 1 1 / 0	0001	0001
S3	0 0 1 1	0 1 0 0 / 0	0 1 0 0 / 0	0111	0111
S4	0 1 0 0	0 1 0 1 / 0	0 1 0 1 / 0	0001	0001
S5	0 1 0 1	0 1 1 0 / 0	0 1 1 0 / 0	0011	0011
S6	0 1 1 0	0 1 1 1 / 0	0 1 1 1 / 0	0001	0001
S7	0 1 1 1	1 0 0 0 / 0	1 0 0 0 / 0	1111	1111
S8	1 0 0 0	0 0 0 1 / 1	0 0 0 0 / 0	1001	1000

Data la linearità del diagramma degli stati, tipica di un contatore, conviene utilizzare flip-flop di tipo T per il registro di stato. Per questa ragione alla tavola codificata degli stati/uscite è stata aggiunta la tavola delle funzioni T3..0.
La sintesi completa richiede la minimizzazione con 5 MK (le quattro variabili di stato + l'ingresso D) a 5 variabili (le quattro variabili di stato + l'ingresso D). Su questa base la rete assume la struttura seguente:

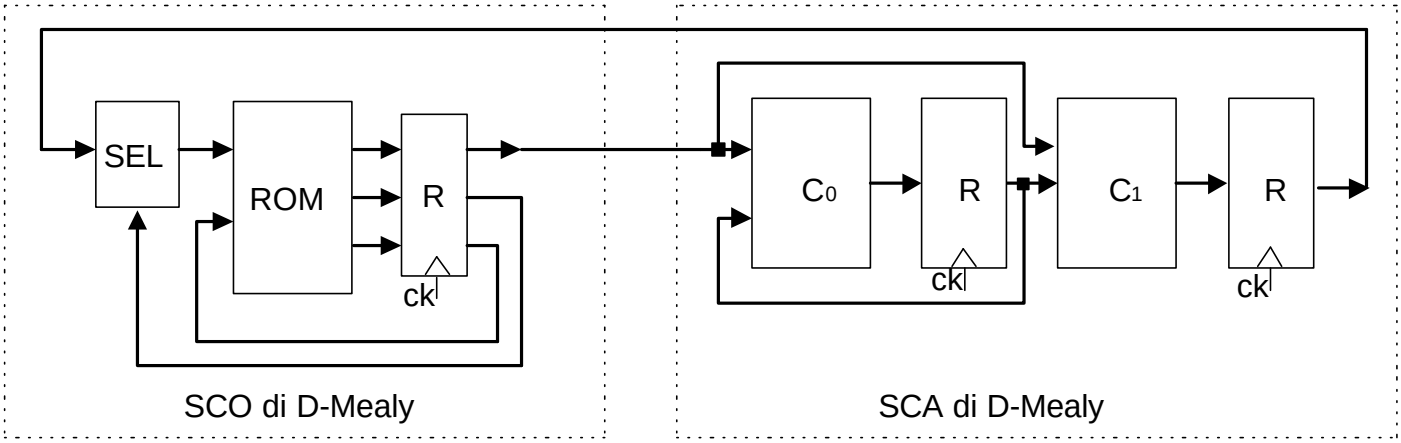


Il calcolo delle tre reti combinatorie è lasciato come esercizio al lettore, sulla base del contenuto della tavola degli stati / uscite precedente.

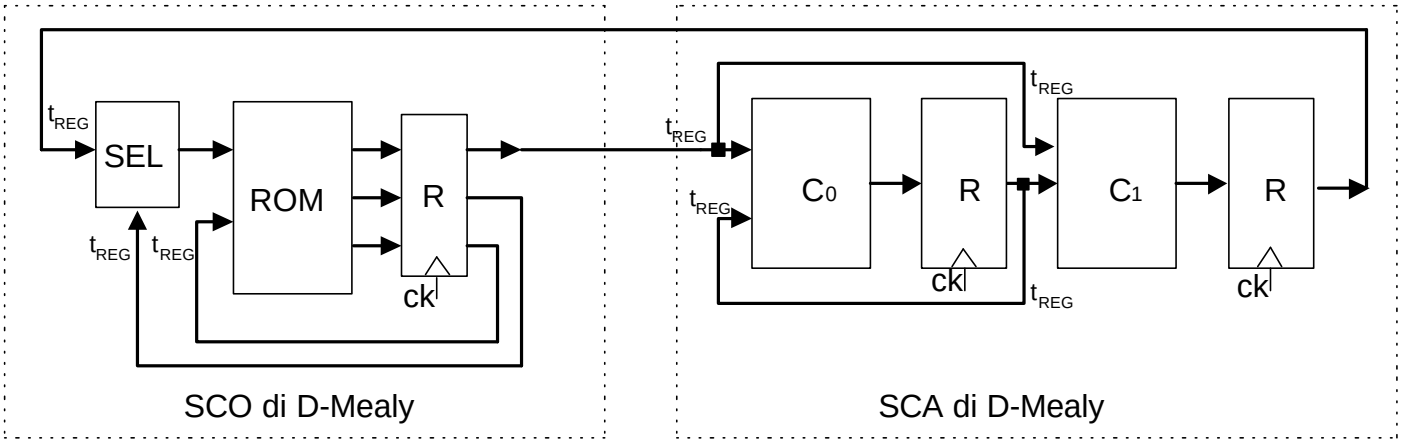
Esercizio (2S20010628-D4)

Calcolare la frequenza massima di funzionamento di un sistema SCO-SCA di tipo D-Mealy – D-Mealy in funzione dei parametri delle due reti.

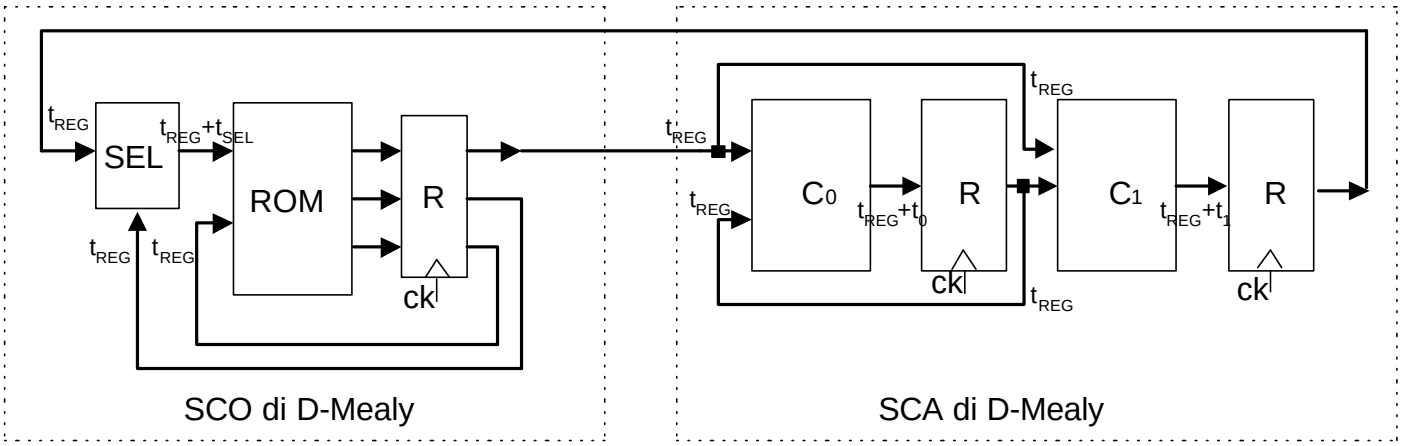
I sotto-sistemi specificati vengono accoppiati come nella figura.



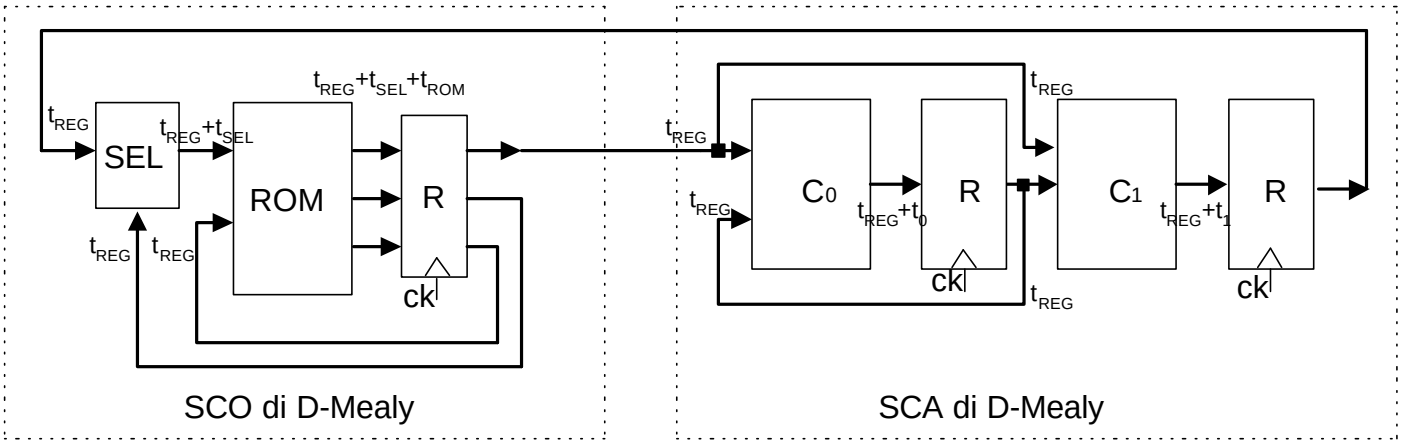
Per valutare la frequenza massima di lavoro (o il periodo minimo del segnale CK) del sistema in funzione dei parametri delle due reti, occorre determinare i tempi massimi in cui diventano stabili i segnali su tutti i nodi dei percorsi dei dati (i "cammini") che hanno origine sulle uscite dei registri e termine sugli ingressi dei registri, a partire dall'applicazione di un fronte del segnale CK; perciò le prime etichette che si possono determinare sono quelle $_{reg}$):



Su questa base si possono ora aggiungere le etichette (i tempi massimi di stabilizzazione) delle uscite delle reti combinatorie alimentate dai registri (per comodità lo schema logico del sistema viene replicato):



Poi si prosegue con i percorsi combinatori in cascata (la ROM in questo caso):



Tutti i nodi della rete sono stati etichettati con i rispettivi tempi massimi di stabilizzazione; quindi si può determinare il massimo tra le tre quantità etichettate sugli ingressi dei tre registri:

$t_{reg} + t_{sel} + t_{ROM}$
$t_{reg} + t_0$
$t_{reg} + t_1$

a cui occorre aggiungere il tempo t_{setup} di set-up dei registri; in definitiva si ottiene:

$T_{CK} > t_{reg} + t_{setup} + \max\{t_{sel} + t_{ROM}, t_0, t_1\}$

Esercizio (2S20010628-D5)

Descrivere il contenuto dei frammenti ritenuti significativi della mappa della memoria del PD32 dopo l'elaborazione da parte dell'assemblatore del codice seguente:
driver 100,10000000h
rti

---

driver 100,10000000h è una direttiva destinata all'assemblatore, che la interpreta come una richiesta di allocare a partire dall'indirizzo 10000000h il codice della routine di servizio identificata dall'IVN 100; in questo caso la routine di servizio (il driver) include la sola istruzione di ritorno **rti** (è una buona pratica progettuale quella di scrivere in questo modo i driver inutilizzati, per evitare un fuori-programma in risposta a richieste di interruzione spurie).Inoltre, l'assemblatore provvederà ad allocare il codice **10000000h** nella tavola dei vettori delle interruzioni, precisamente in 4 byte a partire dall'indirizzo 4x100=400. In definitiva l'assemblatore trasformerà il codice sorgente specificato nel codice oggetto seguente, destinato alla memoria del PD32:

Codice	Indirizzo
00h	400
00h	401
00h	402
10h	403
Codice	10000000h
operativo	10000001h
dell'istruzione	10000002h
RTI	10000003h

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 11-7-2001

STUDENTE:_____DOCENTE:_____

Si vuole progettare un driver per un maxi-schermo basato su una matrice di 4x4 monitor televisivi (VIDEO-WALL) come rappresentato in figura. Il segnale video digitale in ingresso a VIDEO-WALL è costituito da una sequenza di campioni (pixel) a 8 bit sincronizzati da un segnale di clock CK e organizzati in una trama (immagine) di 512 righe x 1024 pixel consecutivi per riga. Le righe della trama sono separate tra di esse da un intervallo temporale prefissato, ma non noto a VIDEO-WALL: pertanto, l'inizio di ogni riga della trama viene segnalato da un impulso su una linea HS, l'inizio della prima riga di ogni trama anche da un impulso su una linea VS; entrambi gli impulsi sono attivi nel solo periodo di CK precedente a quello del primo pixel della riga cui si riferiscono.

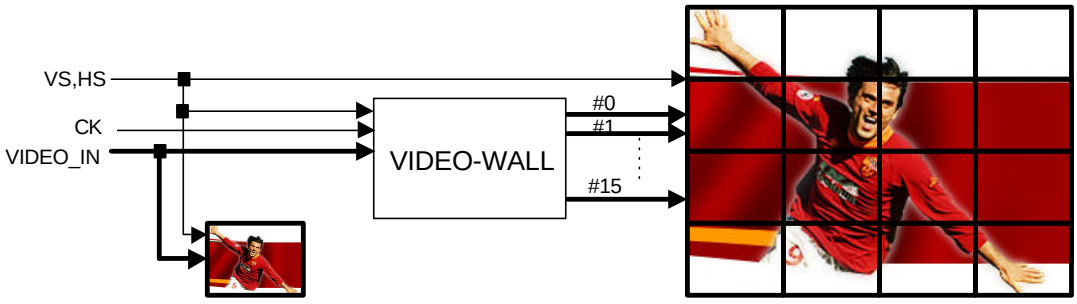


Immagine presentata su un monitor

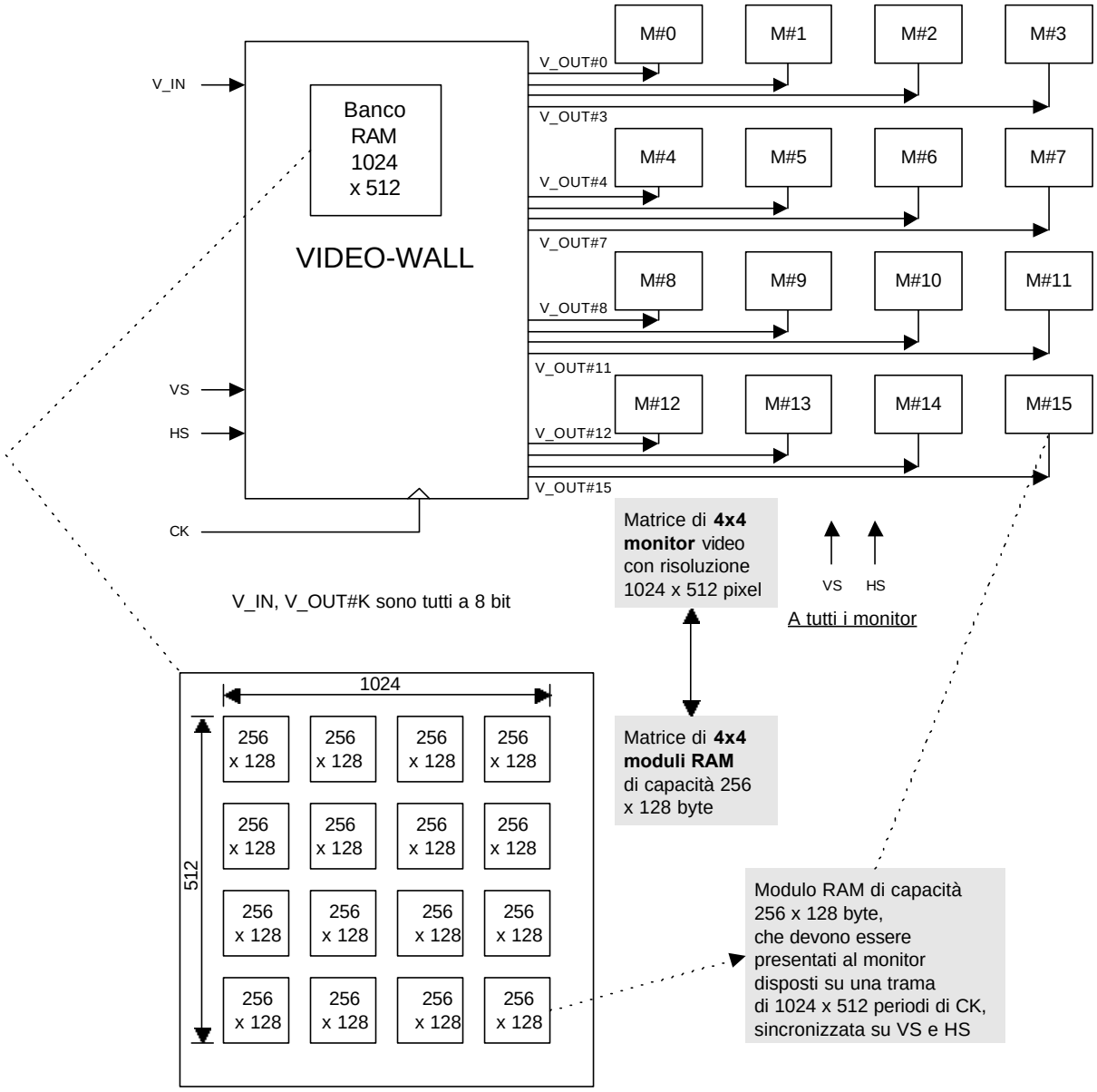
Stessa immagine presentata su 16 monitor pilotati da VIDEO-WALL

VIDEO-WALL tampona i 512 Kpixel dell'immagine di ingresso in un banco di 4x4 moduli RAM di capacità 32 KByte cadauno; nello stesso tempo di scrittura di un'immagine completa, in cui tutti i 16 moduli del banco vengono scritti uno dopo l'altro, ciascuno dei 16 moduli RAM deve essere anche completamente letto per rifornire il rispettivo monitor televisivo con i pixel relativi a un sedicesimo dell'immagine precedente (cfr. figura). Si noti che ogni byte scritto nel modulo RAM dovrà essere ripresentato in lettura quattro volte sia lungo l'asse orizzontale che lungo quello verticale per produrre un ingrandimento del relativo pixel di un fattore 16. In questo modo si ottiene anche che il periodo di scrittura di una intera immagine nel banco 4x4 è uguale al periodo di lettura di ciascuno dei 16 moduli.
Per evitare conflitti negli accessi alle memorie, è previsto che vengano alternati cicli di scrittura e di lettura: poiché ogni modulo RAM contiene solo 1/4 dei pixel di una riga, in ogni periodo della durata di quattro cicli di CK vengono alternati un ciclo di scrittura di quattro pixel consecutivi (tre periodi di CK) e un ciclo di lettura (un periodo di CK) di un singolo pixel.
La frequenza di CK è pari a 15 MHz. Il tempo di ciclo di scrittura/lettura della RAM è di 50 ns. Si suppongano disponibili chip RAM a 8 bit di dato.

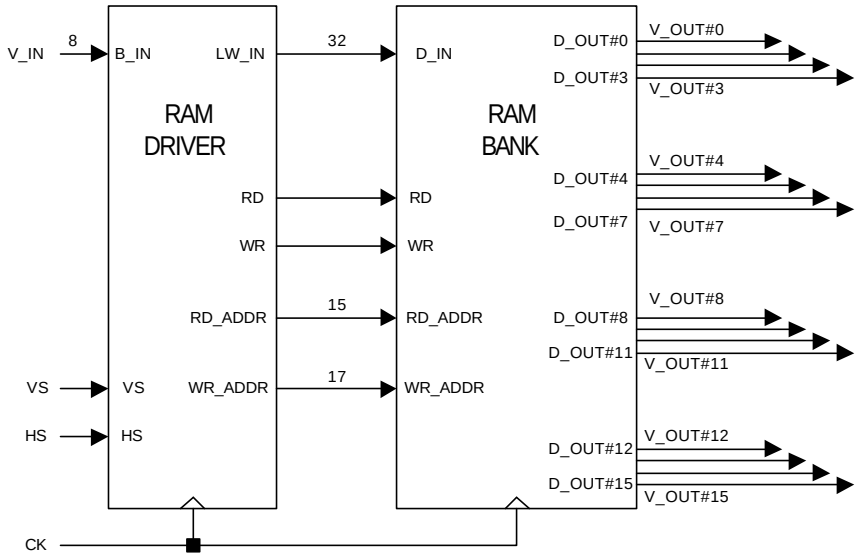
Si richiede:

1. lo schema logico di VIDEO-WALL;
2. l'organizzazione del banco di memoria;
3. la temporizzazione delle operazioni.

VIDEOWALL: sistema esterno



VIDEOWALL: schema a blocchi

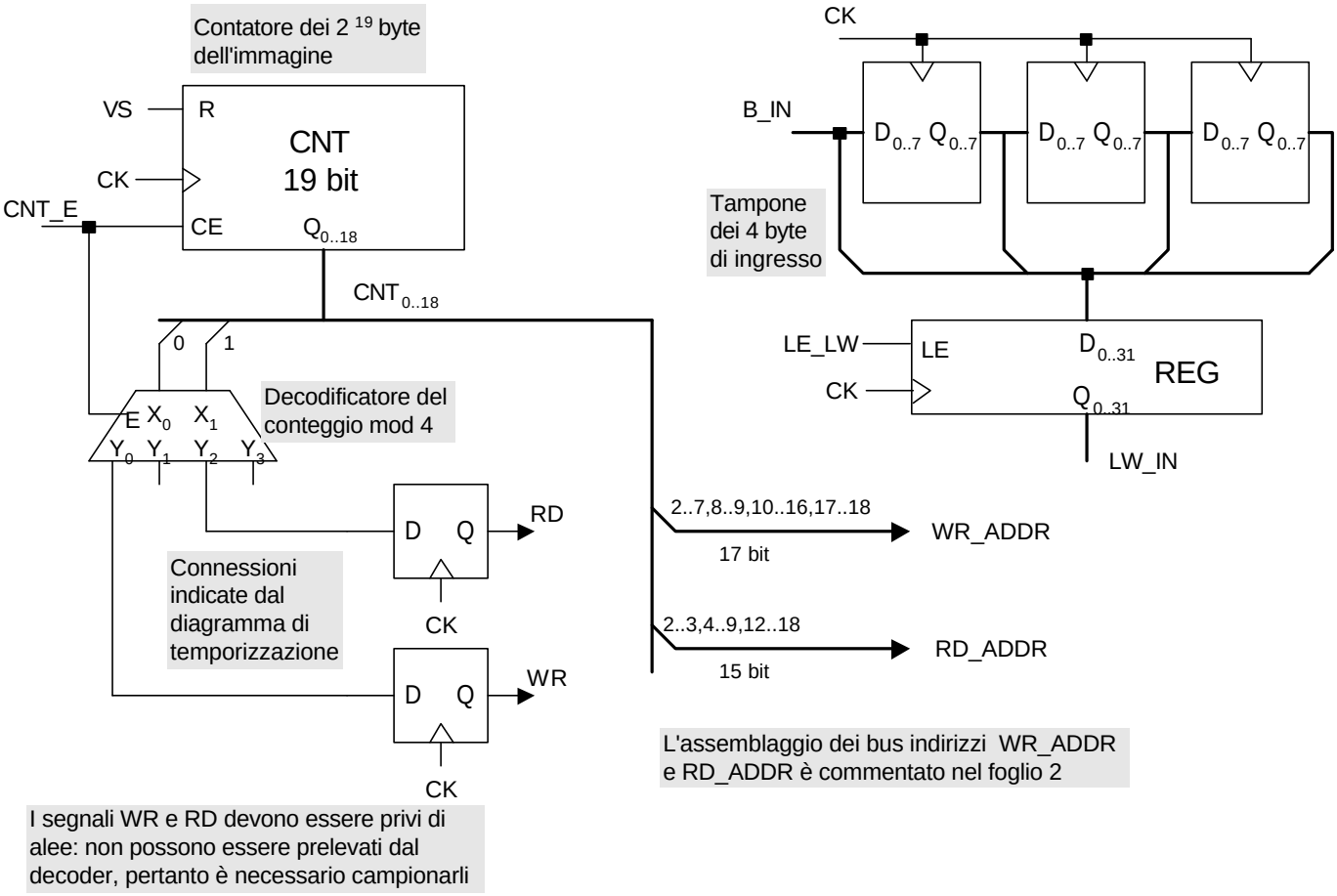


Note

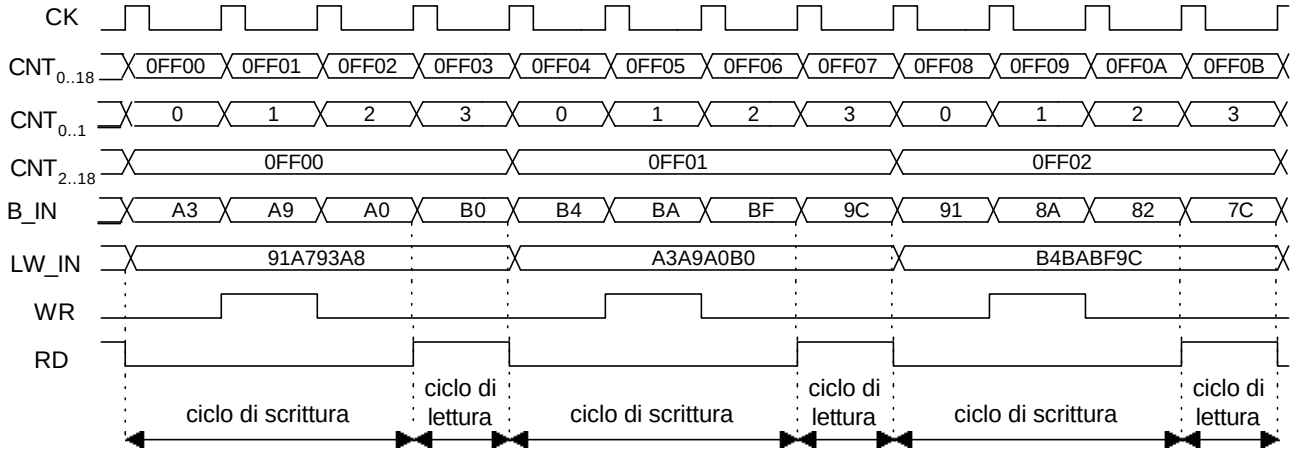
Il blocco RAM_DRIVER:
- assembla i dati a 32 bit (come specificato nel testo) da scrivere nel banco di RAM, a partire dai byte (valori dei pixel) che si avvicendano in ingresso al ritmo di CK;
- produce gli indirizzi di scrittura e di lettura; infatti, il banco di RAM deve essere indirizzato da due processi, uno di scrittura, l'altro di lettura, secondo scansioni diverse: il processo di scrittura deve scrivere la RAM per intero nel tempo di trasmissione di un'immagine: 16x32Kbyte = 2^{19} byte corrispondenti alla risoluzione 1024x512 dell'immagine; quindi tutti i 16 moduli del banco specificati nel testo devono essere scritti, ovviamente in modo mutuamente esclusivo, in quanto ogni byte deve essere scritto in uno e un solo modulo RAM; diversamente il processo di lettura deve essere applicato simultaneamente a tutti i 16 moduli RAM, in modo che questi possano simultaneamente rifornire di dati i monitor rispettivi; ovviamente ciascun modulo RAM conterrà soltanto una porzione di dimensioni 256x128 byte, pari a 1/16 dell'immagine intera (1024x512); tuttavia, tale porzione deve essere presentata al monitor rispettivo nello stesso tempo di scansione di una immagine intera nel formato 1024x512, cioè in 2^{19} periodi di CK (più il tempo di inattività tra le righe); pertanto il processo di lettura dovrà presentare al proprio monitor lo stesso pixel per 4 periodi di CK consecutivi: ciò procurerà l'effetto di espansione orizzontale del pixel; inoltre, dovrà leggere ciascuna riga per 4 volte: ciò procurerà l'effetto di espansione verticale dei pixel.
Gli indirizzi devono distinguere 2^{19} byte da scrivere - secondo specifica - a 32 bit, quindi si devono poter indirizzare 2^{17} longword in scrittura; pertanto le linee di indirizzamento saranno 17 in scrittura.
La specifica di scrivere 32 bit in ogni modulo implica che ciascun modulo sia costituito da 4 chip di RAM da 8 bit di dato disposti in parallelo (applicazione della tecnica di espansione dei dati di una memoria).
In lettura si devono distinguere 256x128 byte (i monotor sono riforniti a byte) e perciò le linee di indirizzamento saranno 8+7=15;
- produce i segnali di scrittura e lettura della RAM.

Il blocco RAM_BANK:
- include una matrice di 4x4 moduli RAM, ognuno dei quali costituito a sua volta di 4 chip da 8 bit di dato; quindi include 16x4=64 chip RAM di capacità e organizzazione 8 Kbyte; con questa posizione si sono definiti tre livelli di astrazione della memoria secondo la gerarchia banco - modulo - chip.
- provvede alla generazione dei segnali di CS dei chip RAM in funzione degli indirizzi; infatti, i CS sono diversi nei processi di scrittura (un solo modulo, di 4 chip, scritto alla volta) e lettura (tutti i moduli abilitati simultaneamente).
Per la scrittura, dei 17 bit di indirizzo presentati al banco, 2+2=4 bit saranno utilizzati per indirizzare i singoli moduli della matrice dei 4x4 moduli, i restanti 17-4=13 bit costituiscono i bit di indirizzo comuni ai moduli (8K x 32 bit), e quindi ai chip (8K x 8 bit).
Per la lettura, dei 15 bit di indirizzo presentati al banco, 2 bit saranno utilizzati per codificare i quattro byte che vengono letti dalla RAM; infatti, la RAM viene indirizzata a longword mentre i monitor vanno riforniti a byte, quindi sarà necessario estrarre un byte alla volta da ogni longword letta; i restanti 15-2=13 bit costituiscono i bit di indirizzo comuni ai moduli (8K x 32 bit), e quindi ai chip (8K x 8 bit);
- provvede alla commutazione dei dati e degli indirizzi di scrittura e lettura all'interno di un periodo di 4 cicli di CK, come specificato dal testo;
- tampona ogni dato letto dalla RAM per sostituirlo ogni 4 cicli di CK; pertanto include 16 registri a 8 bit dotati di abilitazione al caricamento, con le uscite collegate ai monitor rispettivi.

VIDEO-WALL: RAM DRIVER - 1



Temporizzazioni dei cicli di scrittura e di lettura: fase di regime



Note

Il ciclo di scrittura deve essere distribuito su 3 periodi di CK, per coprire i tempi di set-up, di larghezza d'impulso e di hold; il ciclo di lettura invece può essere confinato in un singolo ciclo di CK, dati i tempi specificati nel testo per il ciclo di CK e i tempi di ciclo di scrittura/lettura della RAM (1/15 MHz > 50 ns).

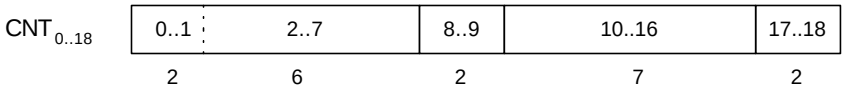
Il decoder che produce i segnali RD e WR è abilitato da CNT_E per evitare che al di fuori delle righe video venga emesso il segnale WR; infatti questo è ricavato dall'uscita Y0 del decoder, che se non abilitata sarebbe a 1 al di fuori delle righe video perché il contatore ha i primi 10 bit pari a 0 (cfr. diagrammi di temporizzazione dei transitori).

VIDEO-WALL: RAM DRIVER - 2

Generazione dei bit di indirizzo

Scrittura

Il contatore che produce i 19 bit $CNT_{0..18}$ incrementa lo stato a ogni colpo di CK, quindi conta i byte (pixel) dell'immagine in ingresso, che deve essere scritta nella RAM. Pertanto, i bit del contatore indirizzano ogni singolo byte nel processo di scrittura. A questo riguardo nello schema seguente viene commentato il ruolo dei bit di conteggio e quindi vengono individuati i bit di indirizzamento e quelli necessari alla produzione dei CS dei moduli RAM.



- bit 0..7: indirizzamento dei 256 byte (quarto di riga video) del modulo;
- bit 2..7: indirizzamento delle 64 longword (quarto di riga video) del modulo;
- bit 0..1: codifica dei 4 byte (pixel) in ingresso che formano una longword da scrivere in RAM; poiché in scrittura non vengono distinti i byte nelle longword, i bit 0..1 non faranno parte dell'indirizzo, ne fanno parte i 6 bit 2..7;

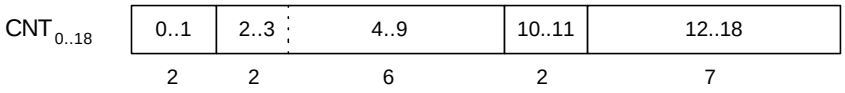
- bit 8..9: indirizzamento della colonna (4 moduli) della matrice di 4x4 moduli; i bit 8..9 devono essere decodificati e contribuire alla formazione dei CS dei moduli RAM;

- bit 10..16: indirizzamento delle 128 righe video del modulo; i bit 10..16 fanno parte dell'indirizzo del modulo;

- bit 17..18: indirizzamento della riga (4 moduli) della matrice di 4x4 moduli; i bit 17..18 devono essere decodificati e contribuire alla formazione dei CS dei moduli RAM.

Lettura

Il contatore che produce i 19 bit $CNT_{0..18}$ incrementa lo stato a ogni colpo di CK, quindi conta i byte (pixel) anche dell'immagine (256x128 byte) che deve essere letta dalla RAM e fornita in uscita ai monitor a piena risoluzione (1024x512 pixel). Pertanto, i bit del contatore indirizzano ogni singolo byte anche nel processo di lettura. Va notato che questa circostanza rende possibile l'uso di un solo contatore per entrambe le scansioni di scrittura e di lettura. Nello schema seguente viene commentato il ruolo dei bit di conteggio nel processo di lettura e quindi vengono individuati i bit di indirizzamento dei moduli RAM.



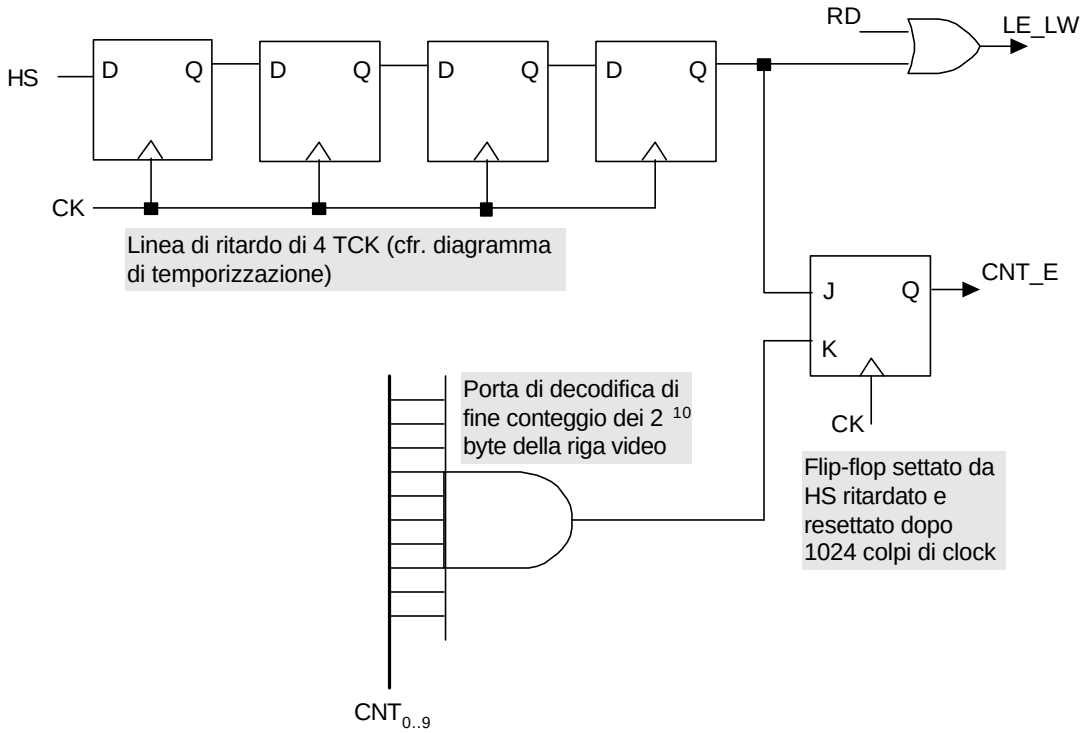
- bit 0..1: fungono da divisori per 4 della frequenza di CK per abilitare la lettura di un byte ogni 4 periodi di CK; pertanto, i bit 0..1 non faranno parte dell'indirizzo di lettura;

- bit 2..9: indirizzamento dei 256 byte (quarto di riga video) del modulo;
- bit 4..9: indirizzamento delle 64 longword (quarto di riga video) accessibili nel modulo;
- bit 2..3: codifica dei 4 byte (pixel) che formano una longword letta dalla RAM; poiché anche in lettura non vengono distinti i byte nelle longword, i bit 2..3 non faranno parte dell'indirizzo, ne fanno parte i 6 bit 4..9; i bit 2..3 dovranno essere utilizzati per selezionare un byte dalla longword letta dalla RAM, quindi controlleranno un riduttore di bus da 32 a 8 bit;

- bit 10..11: provocano (assorbendo per 3 volte il trabocco della sezione di conteggio precedente, senza modificare lo stato di quella successiva) la rilettura della sequenza dei 256 byte che costituiscono un quarto di riga video nel modulo per 4 volte consecutive; pertanto, i bit 10..11 non contribuiscono all'indirizzo;

- bit 12..18: indirizzamento delle 128 righe video del modulo; i bit 12..18 fanno parte dell'indirizzo del modulo.

VIDEO-WALL: RAM DRIVER - 3

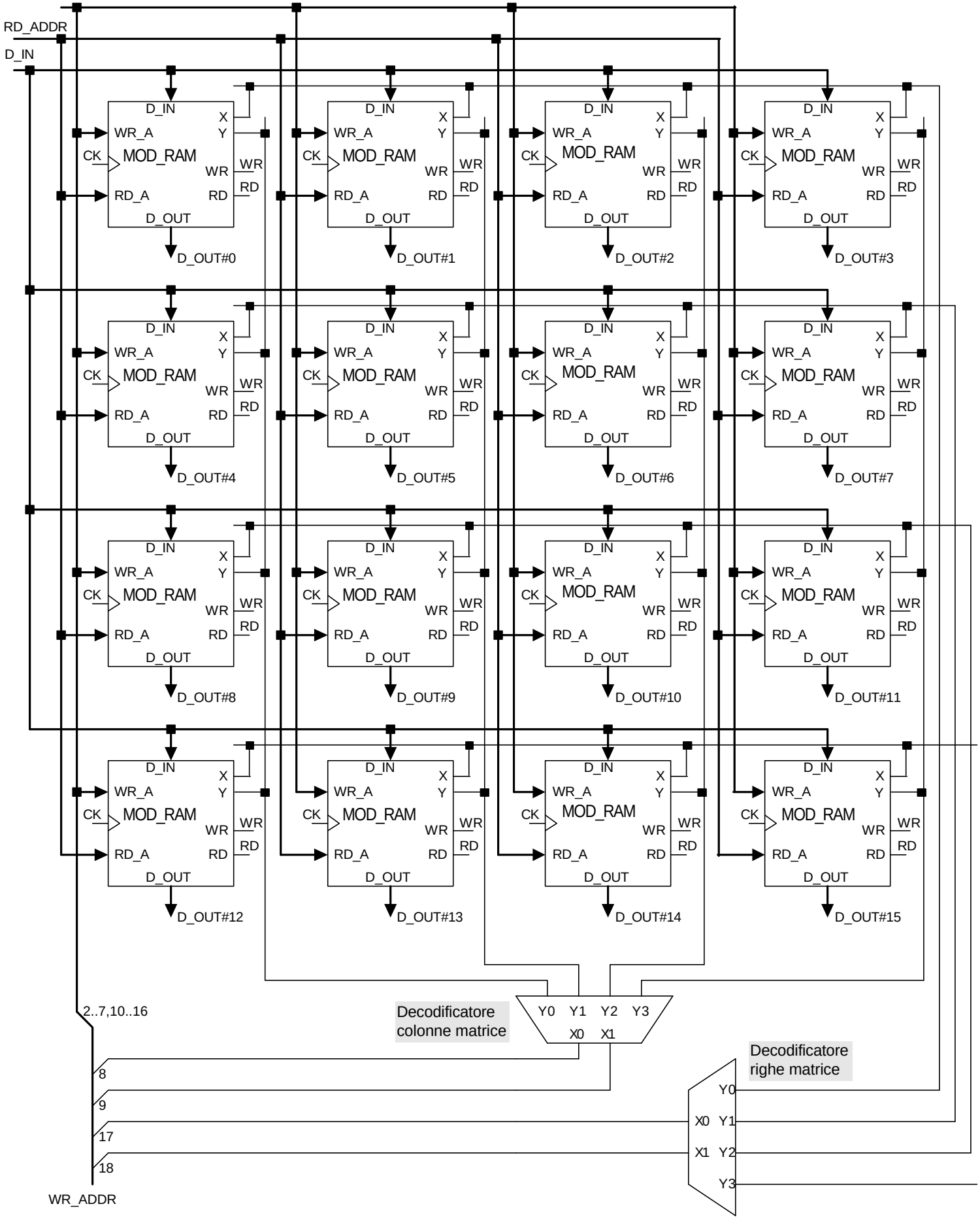


Note

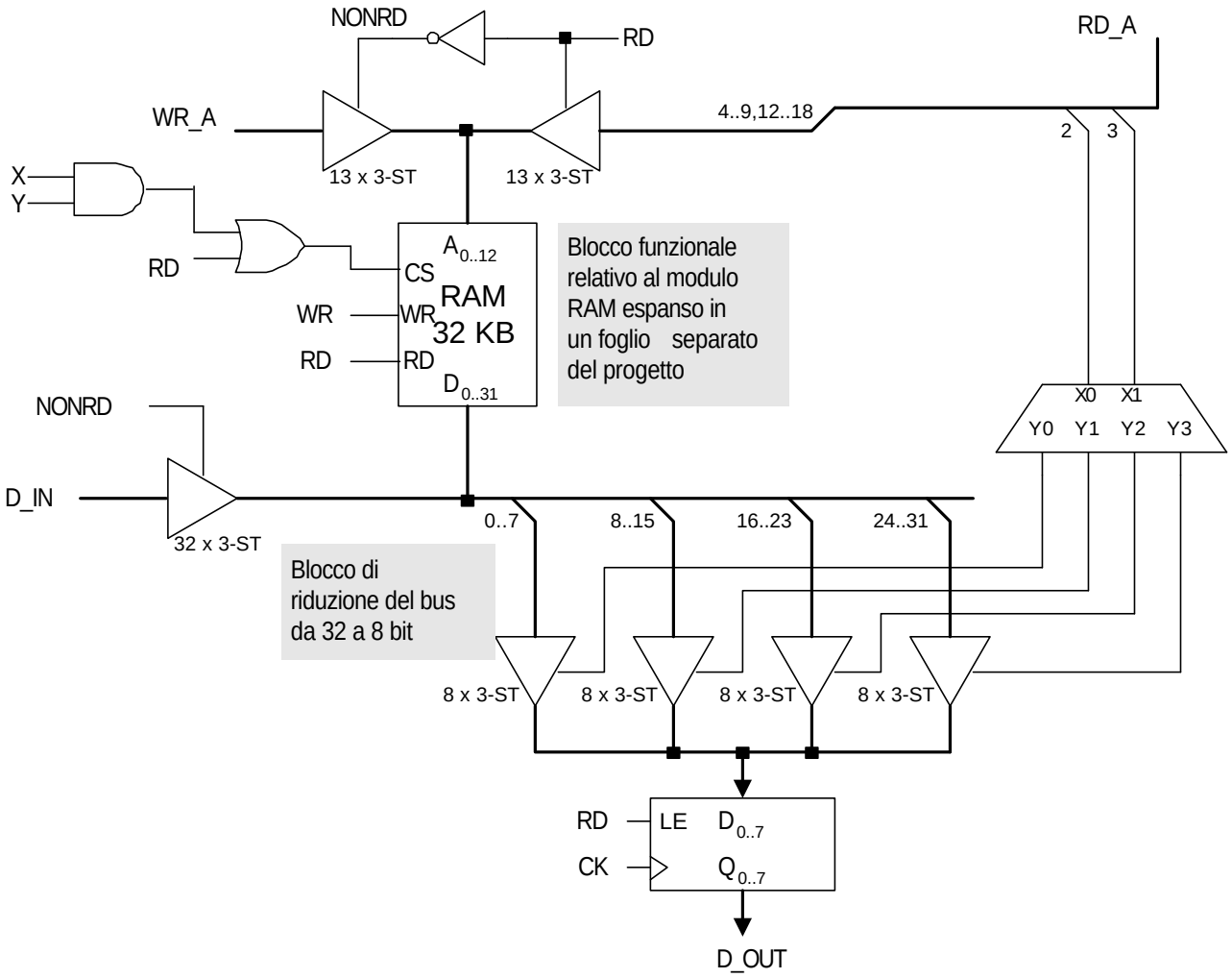
Il circuito indicato gestisce le fasi transitorie iniziale e finale ed è stato derivato sulla base delle indicazioni fornite dai diagrammi di temporizzazione relativi.

E' sufficiente la porta AND collegata al contatore puntatore della RAM per decodificare la fine del conteggio di una riga video perché questa ha una lunghezza (in numero di pixel) pari a una potenza di 2; se non fosse così (ad es. nel caso della risoluzione video 800 x 600) sarebbe necessario utilizzare un secondo contatore per il solo conteggio dei pixel, riservando il contatore originale alla gestione dell'indirizzamento della memoria.

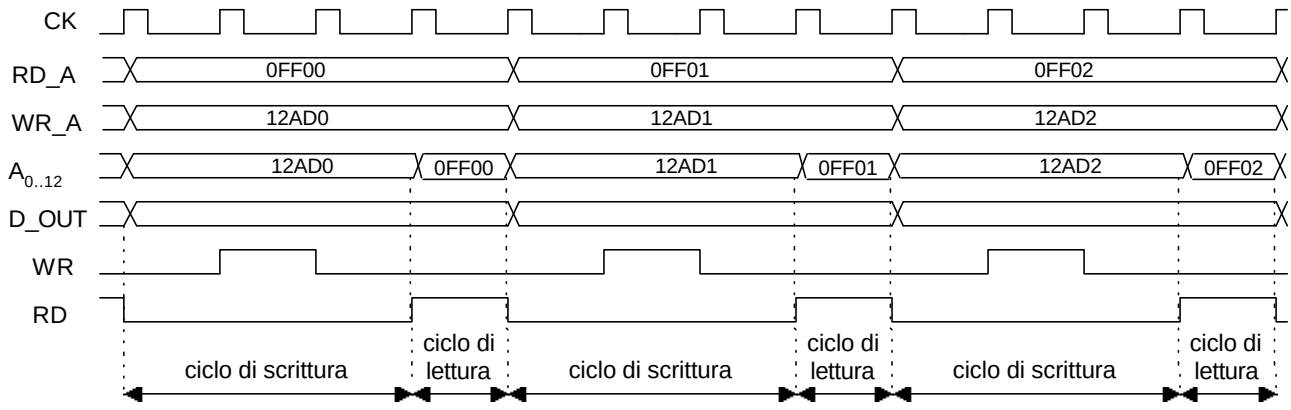
VIDEO-WALL: RAM BANK



VIDEO-WALL: MOD_ RAM



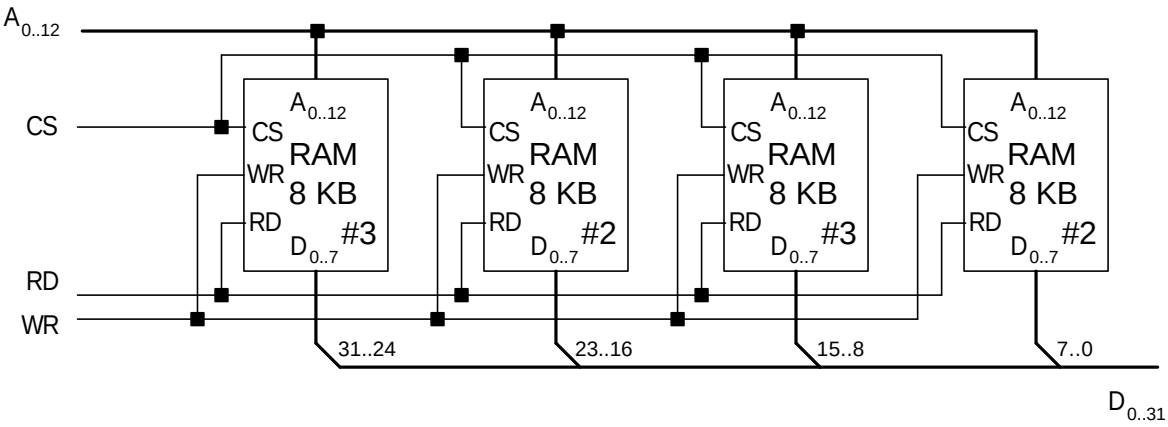
Temporizzazioni dei cicli di scrittura e di lettura

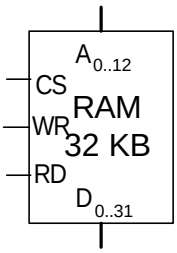


Note

- Come anticipato, i moduli RAM vengono scritti (in 3 cicli di CK) separatamente, ma letti simultaneamente (in un singolo ciclo di CK); pertanto i CS devono essere mutuamente esclusivi in scrittura (coordinate X-Y), ma tutti attivi in lettura: di qui la rete combinatoria che calcola CS mostrata in figura.
- Il segnale RD è utilizzato per alternare i segnali RD_A e WR_A sulle linee di indirizzo della RAM rispettivamente nel quarto ciclo di CK, l'unico in cui viene eseguita la lettura, e nei primi tre cicli di CK, in cui viene effettuato il ciclo di scrittura della RAM; la multiplazione riguarda sia gli indirizzi che i dati. RD è usato anche per abilitare il caricamento del registro di uscita.
- Ogni 4 cicli di CK viene letta una LW, da cui viene estratto un singolo byte e caricato nel registro di uscita; perciò ogni longword viene letta dalla RAM 4 volte e ogni volta ne viene caricato un byte diverso nel registro di uscita. Il periodo di lettura, pari a 4 cicli di CK, procura il mantenimento di ogni byte nel registro di uscita per 4 cicli di CK, e quindi attua l'espansione del pixel di un fattore 4 lungo l'asse orizzontale (riga video).

VIDEO-WALL: modulo RAM





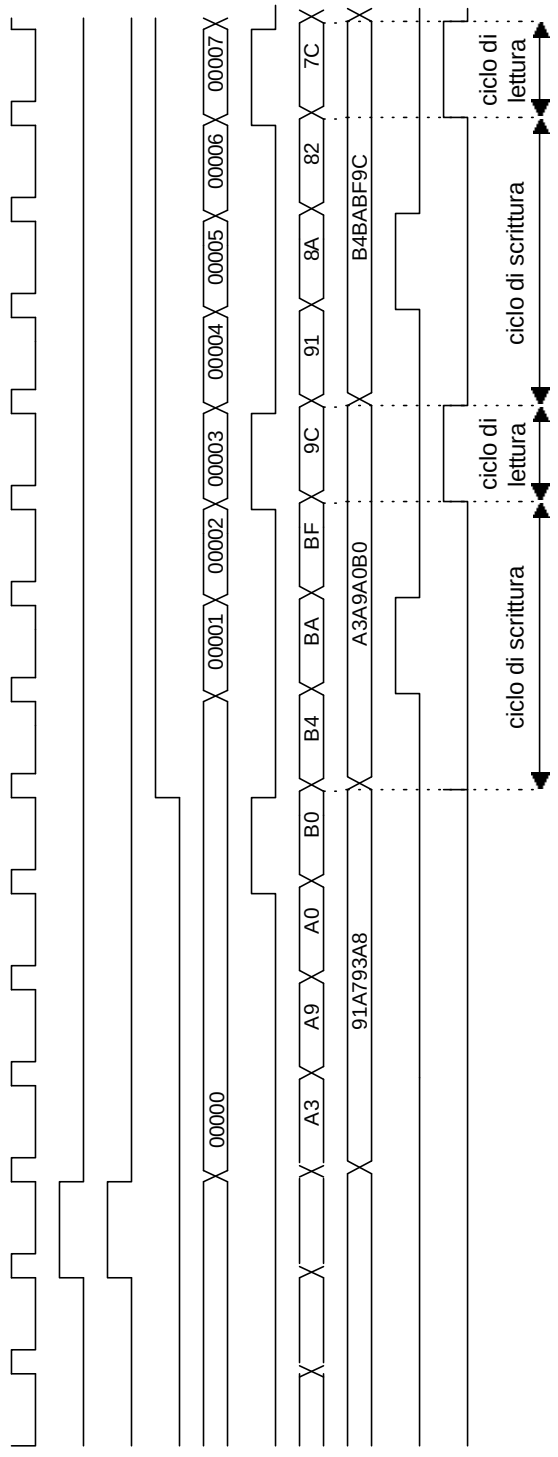
Simbolo del blocco funzionale relativo al modulo RAM

Note

La specifica di scrivere 32 bit in ogni modulo implica che ciascun modulo sia costituito da 4 chip di RAM da 8 bit di dato disposti in parallelo; la struttura può essere ricavata come una diretta applicazione della tecnica di espansione dei dati di una memoria (cfr. "Appunti integrativi"): il modulo, visto esternamente come una RAM con organizzazione 8K x 32 bit, è realizzato fisicamente con 4 chip di organizzazione 8K x 8 bit, collegati in parallelo.

VIDEO-WALL: temporizzazioni

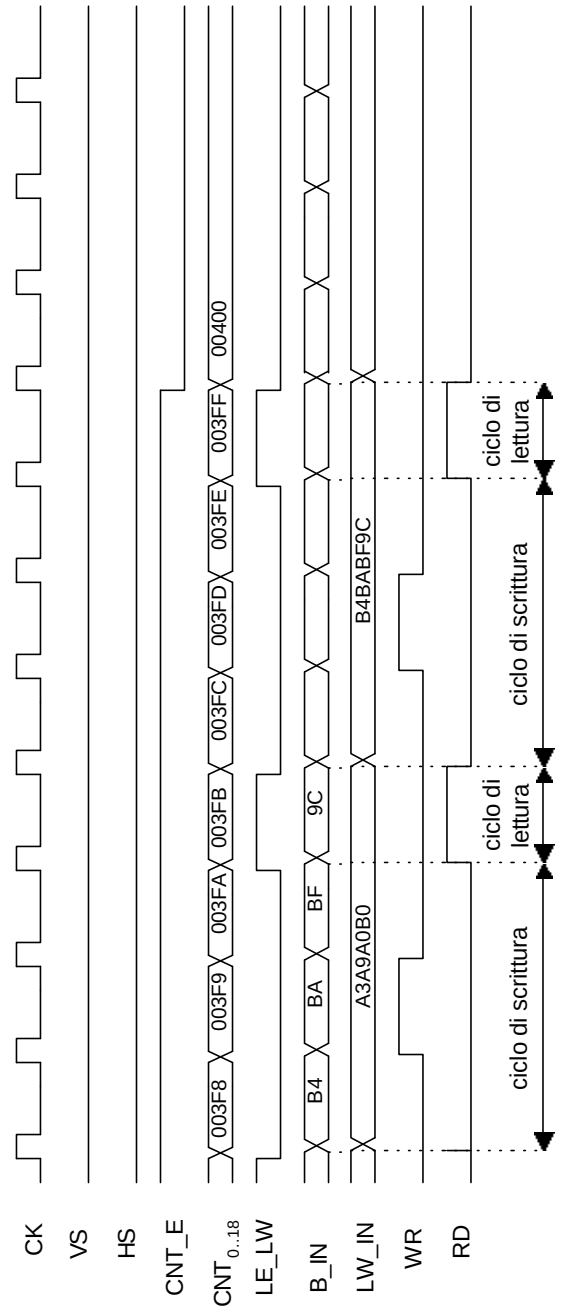
Temporizzazioni dei cicli di scrittura e di lettura: transitorio iniziale
- inizio immagine (prima riga video)



- inizio righe video successive:
  come sopra con le seguenti variazioni

VIDEO-WALL: temporizzazioni

Temporizzazioni dei cicli di scrittura e di lettura: transitorio finale



RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 11-7-2001

STUDENTE:_____DOCENTE:_____

D1 Si dice che una funzione booleana *preserva lo 0* se e solo se: f(0,0,..,0)=0 e che *preserva l'1* se e solo se: f(1,1,..,1)=1. Giudicare il comportamento a questo riguardo delle funzioni: AND, OR, NOT, XOR.

D2 Descrivere una rete combinatoria a pass-transistor per implementare la funzione di commutazione:

$$y = \overline{x_0}\,\overline{x_1}\,x_2 + \overline{x_0}\,x_1\,\overline{x_2} + x_0\,\overline{x_1}\,\overline{x_2}$$

D3 Definire la cella generica di uno shift-register bidirezionale e precaricabile, dotato di abilitazioni separate allo scalamento e al caricamento.

D4 Descrivere le interfacce hardware e le micro-operazioni firmware per supportare un protocollo di comunicazione tra due unità microprogrammate interconnesse tramite un canale dati.

D5 Un processore PD32 che deve funzionare da orologio riceve un'interruzione ogni secondo. Scrivere un programma assembler per aggiornare il codice binario dei secondi, minuti e ore in tre celle di memoria predisposte agli indirizzi SEC, MIN, ORA rispettivamente.

Esercizio (2S20010711-D1)

Si dice che una funzione booleana *preserva lo 0* se e solo se: f(0,0,..,0)=0 e che *preserva l'1* se e solo se: f(1,1,..,1)=1. Giudicare il comportamento a questo riguardo delle funzioni: AND, OR, NOT, XOR.

---

1. AND(0,0,..,0)=0
   AND(1,1,..,1)=1
   giudizio: AND preserva sia lo 0 sia l'1

2. OR(0,0,..,0)=0
   OR(1,1,..,1)=1
   giudizio: OR preserva sia lo 0 sia l'1

3. NOT(0)=1
   NOT(1)=0
   giudizio: NOT non preserva né lo 0 né l'1

4. XOR(0,0,..,0)=0
   XOR(1,1,..,1)=1 se il numero delle variabili indipendenti è dispari
   XOR(1,1,..,1)=0 se il numero delle variabili indipendenti è pari
   giudizio: XOR preserva lo 0; preserva l'1 se e solo se il numero delle variabili indipendenti è dispari

Esercizio (2S20010711-D2)

Descrivere una rete combinatoria a pass-transistor per implementare la funzione di commutazione:

$$y = \overline{x_0}\,\overline{x_1}\,x_2 + \overline{x_0}\,x_1\,\overline{x_2} + x_0\,\overline{x_1}\,\overline{x_2}$$

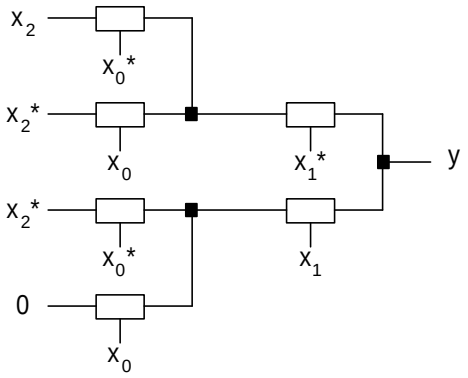
Si deve ricorrere alla sintesi mediante multiplexer. Pertanto il primo passo consiste nella preparazione della tavola di verità di y:

x_2	x_1	x_0	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Quindi si procede alla formulazione della tavola di verità di y in funzione di una delle variabili indipendenti; infatti, con questo accorgimento sugli ingressi di selezione del MUX compaiono due sole variabili anziché tre. Scegliendo ad esempio x_2, si ottiene la tavola di verità seguente:

x_1	x_0	y
0	0	x_2
0	1	x_2*
1	0	x_2*
1	1	0

Da cui si ricava la rete di pass-transistor corrispondente:



in cui si suppongono disponibili le variabili dirette e negate (con l'asterisco).

Esercizio (2S20010711-D3)

Definire la cella generica di uno shift-register bidirezionale e precaricabile, dotato di abilitazioni separate allo scalamento e al caricamento.

Le operazioni possibili sono quattro:
- NOP
- Caricamento del dato esterno
- Scalamento a sinistra
- Scalamento a destra

Pertanto sono sufficienti due segnali di abilitazione per codificare le quattro operazioni, ad esempio secondo la seguente tabella:

Operazione	C1 C0
NOP	0 0
Caricamento del dato esterno	0 1
Scalamento a sinistra	1 0
Scalamento a destra	1 1

In tale ipotesi il circuito della cella si ottiene applicando direttamente la funzione selettiva della tabella alla rete combinatoria del modello strutturale universale delle reti sequenziali sincrone: la rete combinatoria si specializza in un multiplexer incorporato nella cella che include un singolo flip-flop D, come disposto in fig. 1.

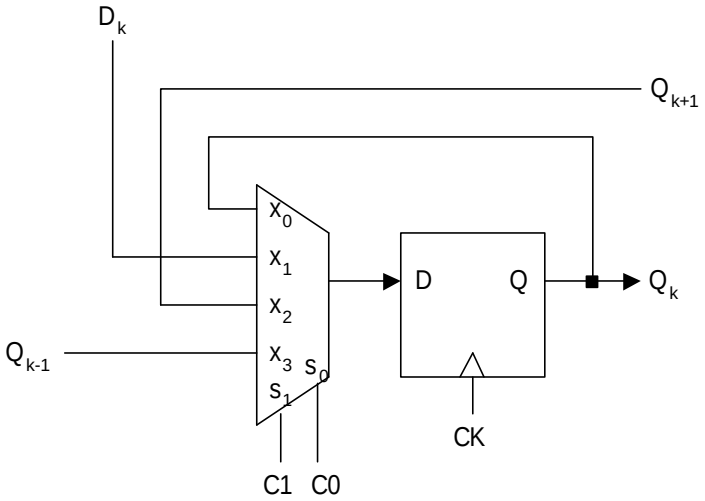


Fig. 1 – Struttura della cella, con i segnali di selezione codificati.

Se invece è richiesto di separare i segnali di abilitazione al caricamento e allo scalamento, i segnali di selezione delle operazioni della cella dello shift-register diventano tre:
LE: segnale di abilitazione al caricamento del dato esterno; è prevalente;
SE: segnale di abilitazione allo scalamento;
L/R: segnale di indicazione della direzione dello scalamento: sinistra (1) / destra (0).

La tabella di codifica delle quattro operazioni diventa la seguente:

Operazione	LE SE L/R
NOP	0 0 -
Caricamento del dato esterno	1 - -
Scalamento a sinistra	0 1 1
Scalamento a destra	0 1 0

Anche in questo caso il circuito della cella si ottiene applicando direttamente la funzione selettiva della tabella alla rete combinatoria del modello strutturale universale delle reti sequenziali sincrone: in questo caso la rete combinatoria diventa un multiplexer di tipo 8/1 incorporato nella cella che include un singolo flip-flop D, come disposto in fig. 2.

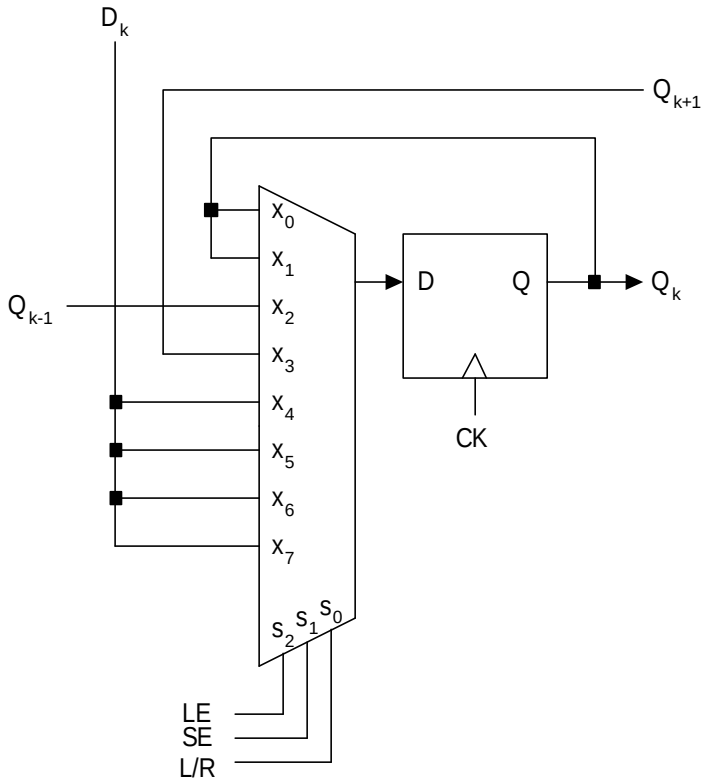


Fig. 2 – Struttura della cella, con i segnali di abilitazione separati.

Esercizio (2S20010711-D4)

Descrivere le interfacce hardware e le micro-operazioni firmware per supportare un protocollo di comunicazione tra due unità microprogrammate interconnesse tramite un canale dati.

Esercizio (2S20010711-D5)

Un processore PD32 che deve funzionare da orologio riceve un'interruzione ogni secondo. Scrivere un programma assembler per aggiornare il codice binario dei secondi, minuti e ore in tre celle di memoria predisposte agli indirizzi SEC, MIN, ORA rispettivamente.

```
        org 400h

; *************************
; COSTANTI
; *************************

        STACK           equ 2800h    ;inizio area di stack
                        ;limitato a 2800h per consentire la simulazione

;*************************
; VARIABILI
;*************************

        SEC   db 0
        MIN   db 0
        ORA   db 0
        FLAG db 0       ;FLAG=1 segnala un nuovo secondo

; *************************
; CODICE
; *************************
        code            ;inizio istruzioni

main:
        movl #STACK,r7  ;inizializza R7 quale SP
        seti            ;abilita interruzioni (SP è stato inizializzato)


mainloop:
        movb flag,r0    ;test sulla segnalazione di nuovo secondo
        andb r0,r0
        jz skiploop
        movb #0,flag    ;resetta la locazione flag (dato consumato)
        jsr newsec      ;e aggiorna l'orologio

skiploop:
;spazio per altri segmenti del programma


;al termine dei quali è prevista la chiusura del loop su mainloop
        jmp mainloop
```

```
;*********************************
;SUBROUTINE
;*********************************
newsec:
        push r0
        movb SEC,r0
        cmpb #59,r0         ;fine minuto?
        jz newmin
        addb #1,r0          ;SEC+1
        movb r0,SEC
        jmp exit

newmin:
        xorb r0,r0          ;SEC <- 0
        movb r0,SEC
        movb MIN,r0
        cmpb #59,r0         ;fine ora?
        jz newhour
        addb #1,r0          ;MIN+1
        movb r0,MIN
        jmp exit

newhour:
        xorb r0,r0          ;MIN <- 0
        movb r0,MIN
        movb ORA,r0
        cmpb #23,r0         ;fine dì o giorno?
        jz zero
        addb #1,r0          ;ORA+1
        movb r0,ORA
        jmp exit

zero:
        xorb r0,r0          ;ORA <- 0
        movb r0,ORA

exit:
        pop r0
        ret

;*********************************
;DRIVER
;*********************************
;Setta la flag di nuovo secondo

        driver 10, 800h     ;Il driver della periferica con IVN=10
                            ;inizia dall'ind. 800h
        movb #1,flag        ;setta la locazione flag (dato pronto)
        rti                 ;ritorno da interruzione

        end                 ;fine programma
```

```
;Un processore PD32 che deve funzionare da orologio riceve un'interruzione ogni
secondo.
;Scrivere un programma assembler per aggiornare il codice binario dei secondi,
minuti e ore
;in tre celle di memoria predisposte agli indirizzi SEC, MIN, ORA rispettivamente.


  org 400h

; ************************
; COSTANTI
; ************************

  STACK   equ 2800h ;inizio area di stack
        ;limitato a 2800h per consentire la simulazione



;*************************
; VARIABILI
;*************************

  SEC db 0
  MIN db 0
  ORA db 0
  FLAG  db 0    ;FLAG=1 segnala un nuovo secondo

; ************************
; CODICE
; ************************
  code      ;inizio istruzioni

main:
  movl #STACK,r7    ;inizializza R7 quale SP
  seti      ;abilita interruzioni (SP è stato inizializzato)


mainloop:
  movb flag,r0    ;test sulla segnalazione di nuovo secondo
  andb r0,r0
  jz skiploop
  movb #0,flag    ;resetta la locazione flag (dato consumato)
  jsr newsec    ;e aggiorna l'orologio

skiploop:
;spazio per altri segmenti del programma


;al termine dei quali è prevista la chiusura del loop su mainloop
  jmp mainloop


;*********************************
;SUBROUTINE
;*********************************

newsec:
  push r0

  movb SEC,r0
  cmpb #59,r0   ;fine minuto?
  jz newmin
  addb #1,r0    ;SEC+1
  movb r0,SEC
  jmp exit

newmin:
```

```
  xorb r0,r0    ;SEC <- 0
  movb r0,SEC
  movb MIN,r0
  cmpb #59,r0   ;fine ora?
  jz newhour
  addb #1,r0    ;MIN+1
  movb r0,MIN
  jmp exit

newhour:
  xorb r0,r0    ;MIN <- 0
  movb r0,MIN
  movb ORA,r0
  cmpb #23,r0   ;fine dì o giorno?
  jz zero
  addb #1,r0    ;ORA+1
  movb r0,ORA
  jmp exit
zero:
  xorb r0,r0    ;ORA <- 0
  movb r0,ORA

exit:
  pop r0
  ret


;*********************************
;DRIVER
;*********************************
'

;Setta la flag di nuovo secondo


  driver 10, 800h   ;Il driver della periferica con IVN=10
        ;inizia dall'ind. 800h
  movb #1,flag    ;setta la locazione flag (dato pronto)
  rti     ;ritorno da interruzione



  end     ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 21-9-2001

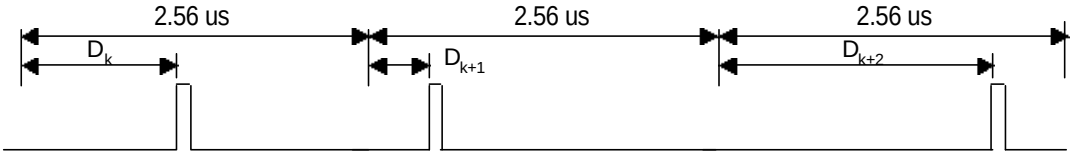
STUDENTE:_____DOCENTE:_____

Si vuole realizzare un modulatore per trasferire su una linea di comunicazione seriale la sequenza dei campioni a 8 bit di un un segnale digitale predisposta nella memoria di un processore PD32. Quando il processore vuole trasmettere un segnale invia al modulatore l'indirizzo iniziale e la lunghezza - in byte – dell'area di memoria in cui la sequenza è disponibile.
In risposta la periferica esegue le seguenti attività:

- preleva i campioni della sequenza mediante accesso in DMA di tipo stealing;
- forza i valori <= 3 al livello di soglia 4 e i valori >= 253 al livello di saturazione 252;
- trasmette i valori ottenuti sulla linea di comunicazione seriale specificata senza soluzione di continuità mediante modulazione PPM (Modulazione della Posizione dell'imPulso): a ciascun valore B_k assegna un intervallo di durata pari a 2.56 microsec. in cui trasmette un impulso di durata pari a 40 ns e ritardato rispetto all'inizio dell'intervallo della quantità:

$$D_k = B_k \ 10 \text{ ns}$$

con $4 <= B_k <= 252$, come mostrato nella specifica temporale grafica;



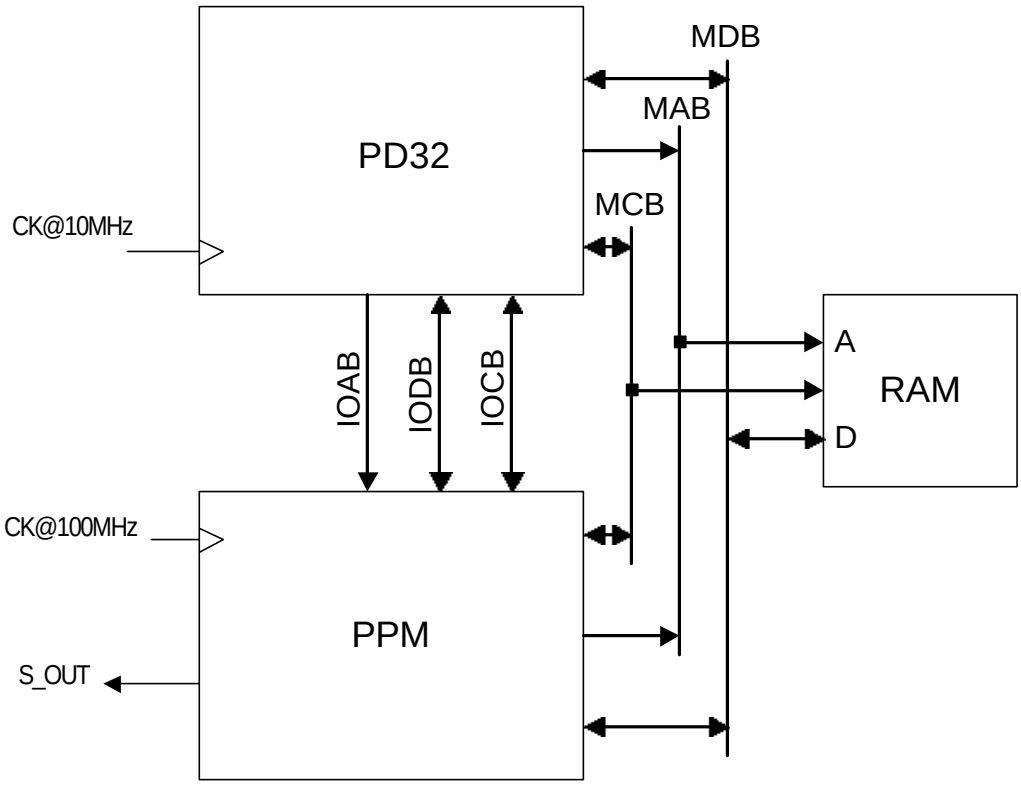
- segnala il termine della trasmissione al processore con una interruzione.

Per applicare la soglia e la saturazione si supponga disponibile una ROM. Il processore ha un clock a 10 MHz.

Si richiede:

1. la temporizzazione delle operazioni;
2. lo schema logico della periferica;
3. l'organizzazione dei dati all'interno della ROM;
4. le routine d'interfacciamento del PD32.

PPM: sistema esterno



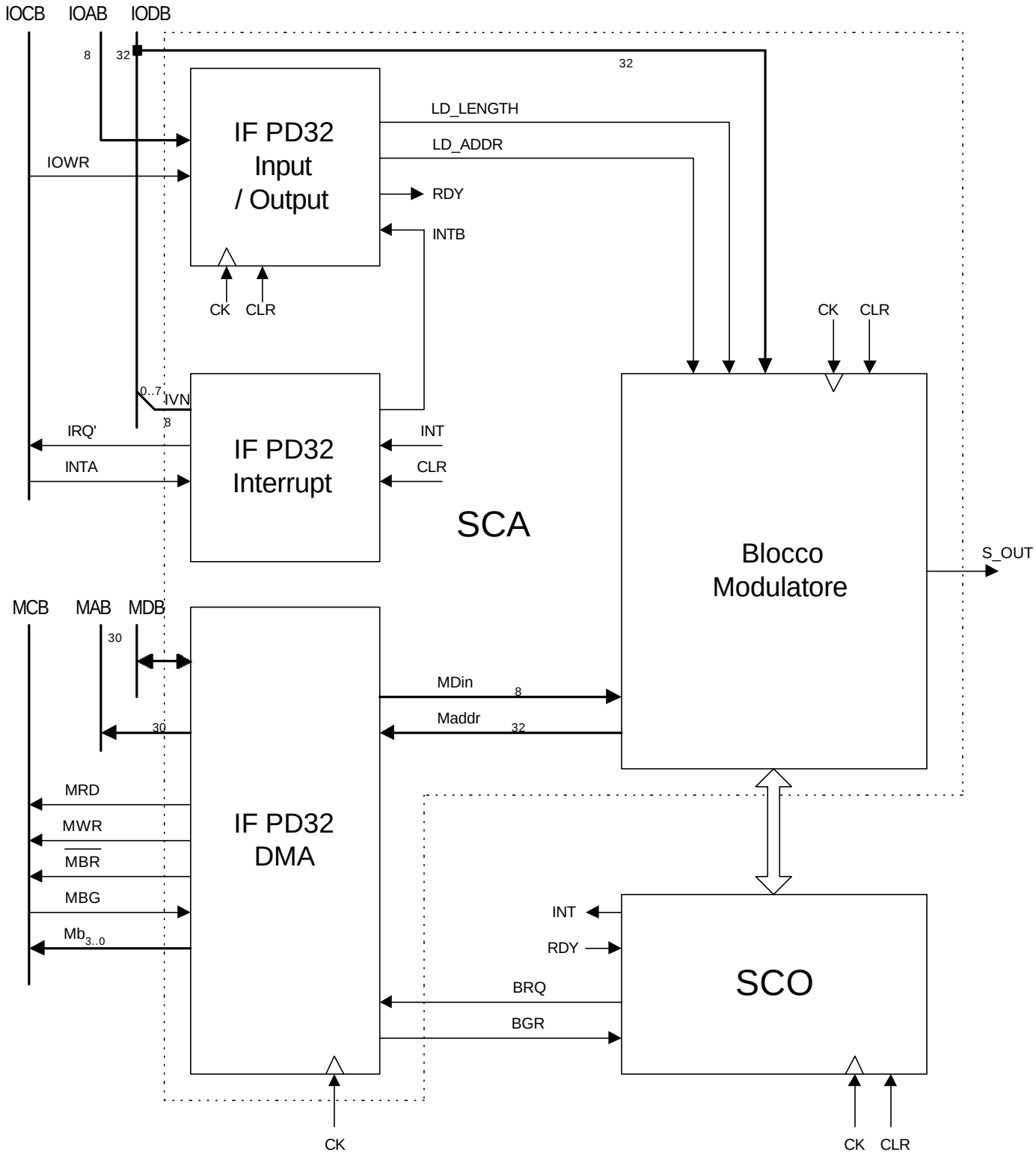
Note

La frequenza del clock della periferica PPM viene scelta in modo da supportare la risoluzione temporale specificata nel testo: 10 ns. Pertanto viene utilizzato un clock a 100 MHz.

Il processore è dotato di un clock a frequenza 10 MHz, ovvero con periodo pari a 100ns: poiché il processore legge e scrive in memoria con tre cicli del suo clock, ciò implica che la RAM può essere pilotata in lettura correttamente con un ciclo di durata di almeno 3x100ns=300 ns. Pertanto la periferica per emulare il ciclo di lettura del processore dovrà impiegare almeno 3x(100/10)=30 cicli del proprio clock.

La specifica richiede alla periferica un accesso alla memoria in bus-stealing, ovvero il prelevamento di un dato (byte) ogni 256 cicli di CK. Pertanto lo SCA della periferica dovrà includere un contatore mod 256, la cui uscita di fine conteggio segnalerà l'evento di inizio periodo, in corrispondenza del quale dovranno essere avviate due operazioni:
1) il caricamento di un secondo contatore, con la funzione di modulatore, con il valore da modulare, prelevato dalla memoria all'inizio del ciclo di 256 Tck appena concluso; l'uscita di fine conteggio di questo secondo contatore dovrà settare un flip-flop di uscita (S_OUT), che dovrà essere resettato dopo 4 periodi di CK per produrre l'impulso di uscita di durata 40 ns;
2) la richiesta di bus al micro e il conseguente accesso alla RAM per leggere il dato da modulare nel periodo (256 Tck) successivo. A questo proposito va osservato che il tempo impiegato dalla periferica per prelevare un dato è dato dalla somma di due contributi: tG+tA, essendo tG il tempo necessario al processore per cedere i bus di sistema e tA è il tempo di accesso della memoria; tG non è noto a priori, in quanto dipende dallo stato interno del processore al momento della richiesta dei bus da parte della periferica, tuttavia può essere limitato superiormente, in quanto il processore impiegherà al massimo il tempo di un suo ciclo-macchina per riconoscere la richiesta di bus: il caso peggiore è quello in cui la richiesta viene attivata dalla periferica quando il processore ha appena iniziato un ciclo macchina relativo a una lettura o scrittura di una longword disallineata in RAM; infatti, tale ciclo-macchina sarà composto di due cicli di bus e quindi impegnerà 6 cicli di clock, quindi 600 ns. D'altra parte tA è noto e vale 300 ns; quindi l'attività di richiesta dei bus e lettura del dato durerà al massimo 900 ns, che è compatibile (<<) con il periodo di elaborazione di un singolo dato da parte della periferica, pari a 256x 10ns=2.56 microsec. Va osservato che se il periodo di elaborazione di un singolo dato da parte della periferica fosse ad esempio 800 ns, allora l'accesso in stealing non sarebbe possibile, e occorrerebbe impostare un accesso a burst, in modo da azzerare la componente tG (che andrebbe computata una sola volta, all'inizio dell'accesso a burst) e trovare verificato che tA=300ns<800ns.

PPM: Schema a blocchi

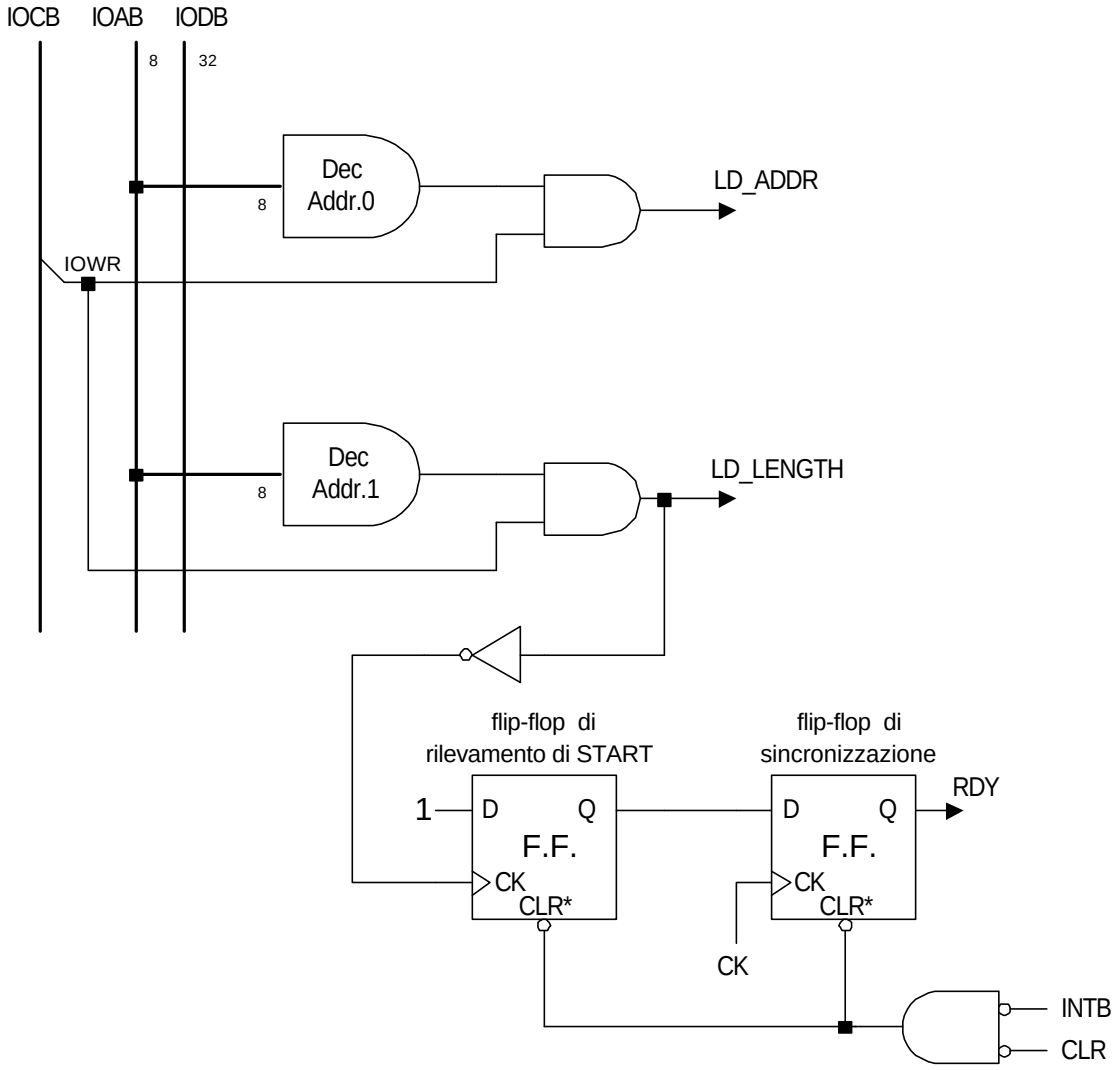


Note

I blocchi IF PD32 output e DMA sono dotati di ck per sincronizzare i segnali entranti.

Il blocco IF PD32 output ha anche l'ingresso CLR asincrono diretto al flip-flop di handshake, per evitare il rischio di una falsa segnalazione di richiesta del processore all'inizio dell'attivazione della periferica. Per lo stesso motivo il blocco IF PD32 interrupt usa il segnale CLR per azzerare il flip-flop di richiesta di interruzione.

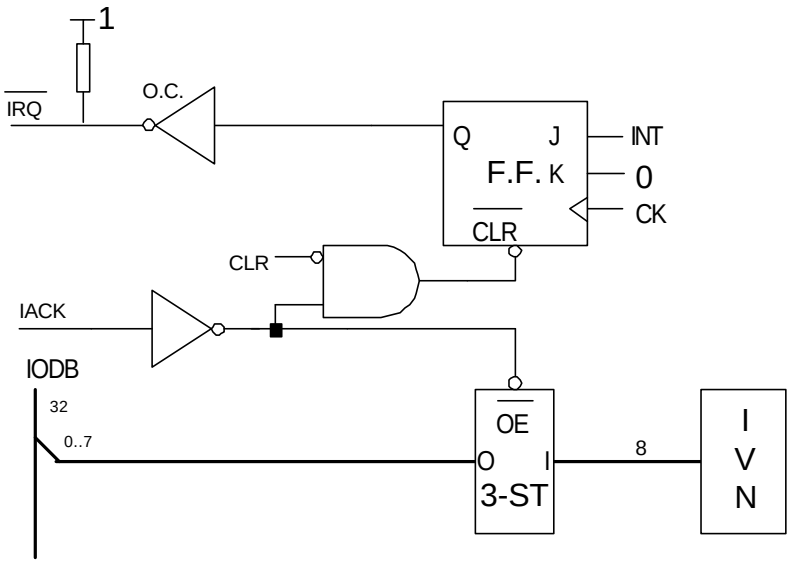
PPM: IF PD32 - output



Note

Il SW avvia l'operazione direttamente con l a scrittura dell'indirizzo della lunghezza in byte del blocco di dati da trasferire, dopo avere scritto l'indirizzo iniziale del blocco. Conviene caricare entrambi i dati direttamente nei contatori di lavoro in modo asincrono, in quanto i dati vengono utilizzati da PPM una volta soltanto nel corso di una elaborazione e perciò possono essere risparmiati i due latch nell'interfaccia di output, altrimenti necessari.

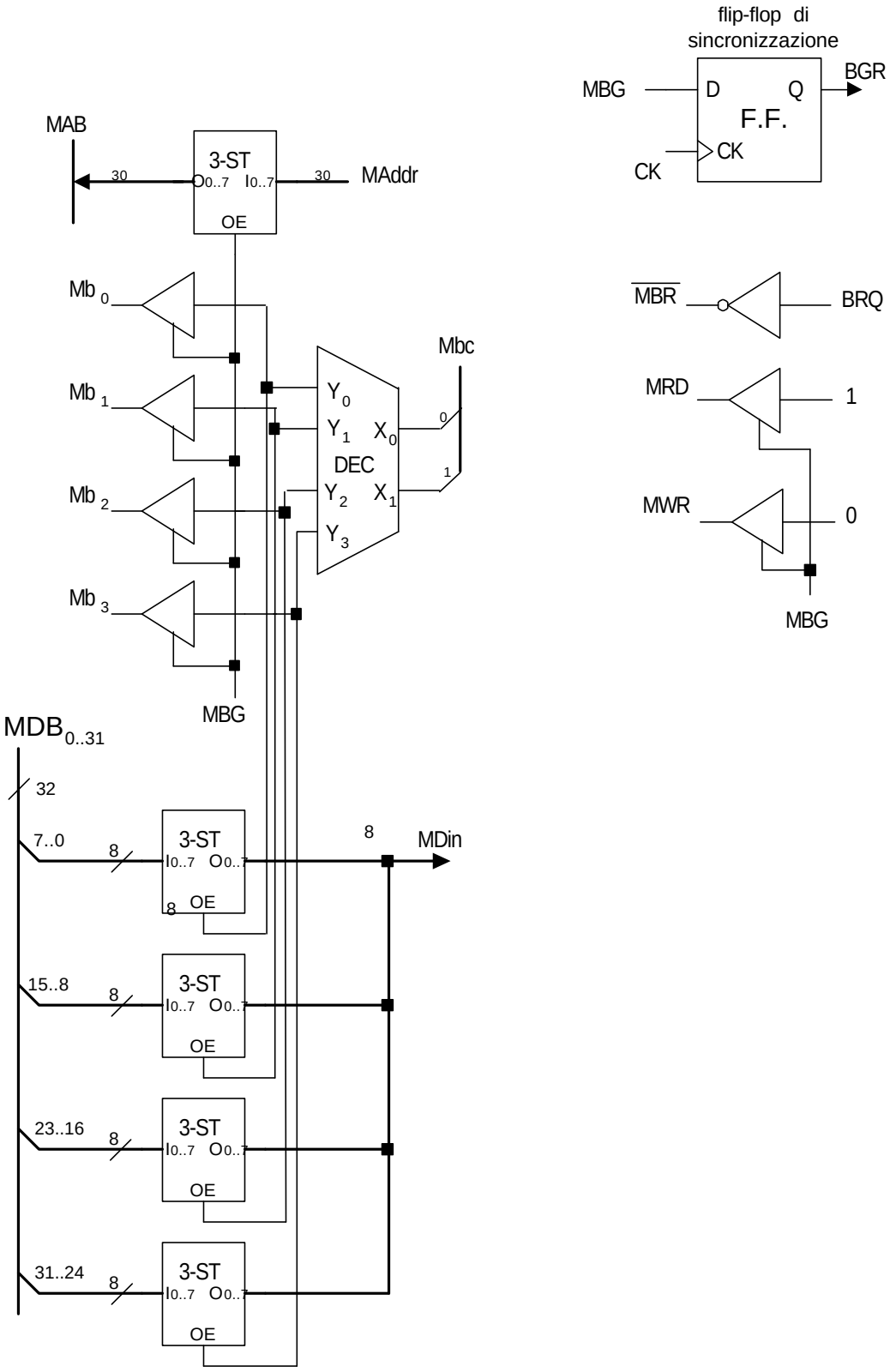
PPM: IF PD32 - interrupt



Note

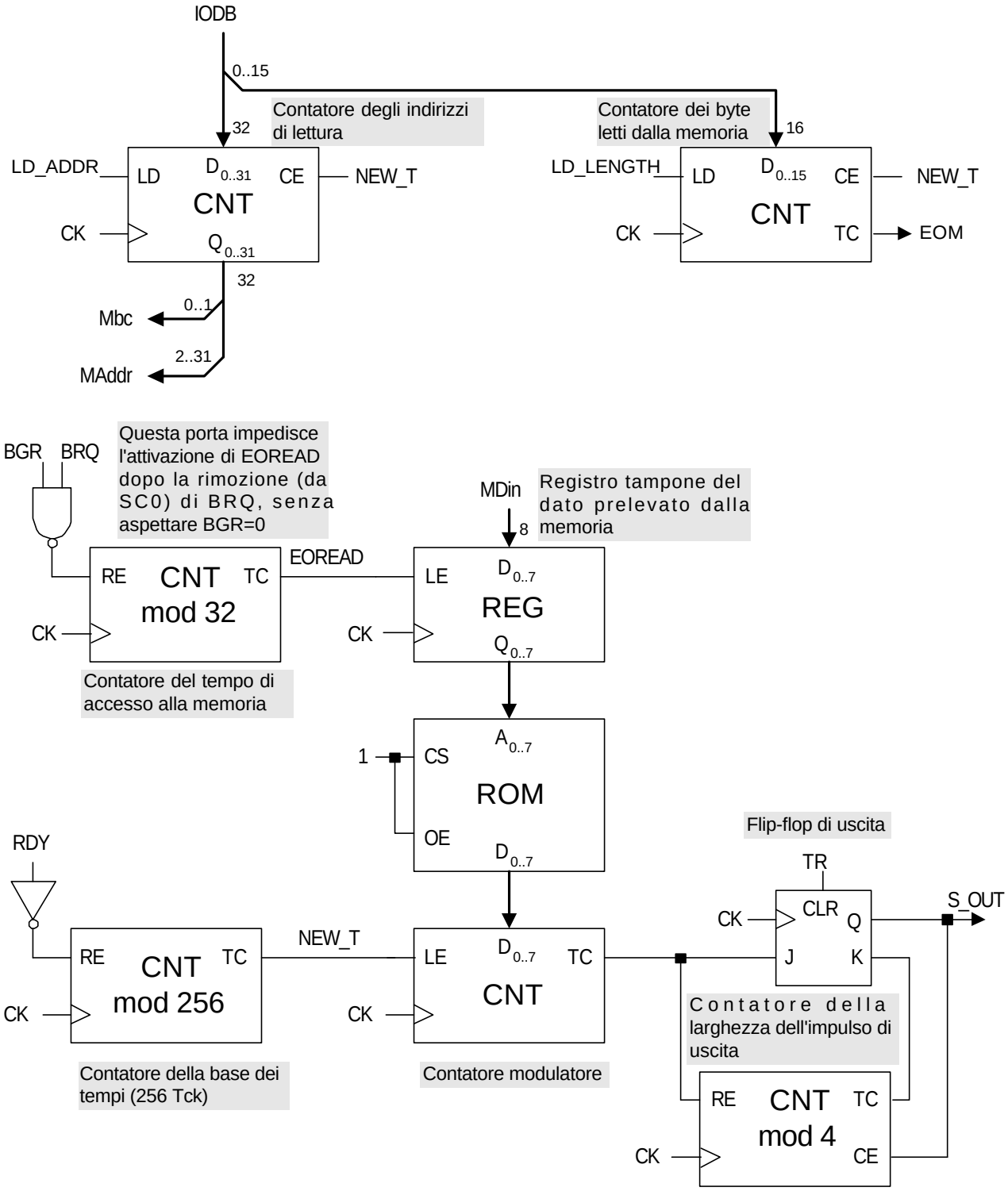
Il segnale di richiesta di interruzione viene terminato sull'ingresso J di un flip-flop sincrono JK; in questo modo INT può essere emesso da uno SCO qualsiasi, anche con uscite prodotte mediante reti combinatorie: gli spike su INT non hanno alcun effetto sul flip-flop sincronizzato.
Va osservato che diversamente se si usasse un latch SR occorrerebbe garantire l'assenza di spike su INT e questo requisito imporrebbe l'uso di uno SCO di tipo D-Mealy: in tal caso il segnale di richiesta di interruzione richiederebbe un flip-flop D e un latch SR, e quindi sarebbe più costoso della soluzione presentata.
Un'altra considerazione riguarda la generalità e l'opportunità della connessione mostrata in figura, piuttosto che quella inversa: SCA pilota l'ingresso asincrono di SET di un flip-flop D, IACK terminato sull'ingresso di trigger del flip-flop con D=0. Infatti, mentre in linea di principio le due connessioni sembrerebbero intercambiabili, quando si tiene conto del tempo di risposta del micro in relazione al periodo di CK della periferica, la connessione ipotizzata in alternativa potrebbe fallire nella memorizzazione del riconoscimento dell'interruzione: infatti, se il periodo di CK della periferica fosse maggiore del tempo di riconoscimento dell'interruzione da parte del micro, la commutazione di INTA non verrebbe sentita dal flip-flop, in quanto la sua linea di SET asincrono sarebbe ancora attiva. Diversamente, la connessione mostrata cattura la richiesta sul fronte di INT e perciò è in grado di reagire al riconoscimento della richiesta di interruzione, indipendentemente dalla velocità del processore; inoltre, quando IACK=1 la periferica non può ancora avere avuto una nuova richiesta dal processore, e quindi non si troverà mai nella posizione di emettere un fronte, che non potrebbe essere sentito, sulla linea INT quando IACK=1.

PPM: IF PD32 - DMA



Blocco di riduzione dei 32 bit del MDB agli 8 bit del bus interno MDin (lettura).

PPM: blocco modulatore



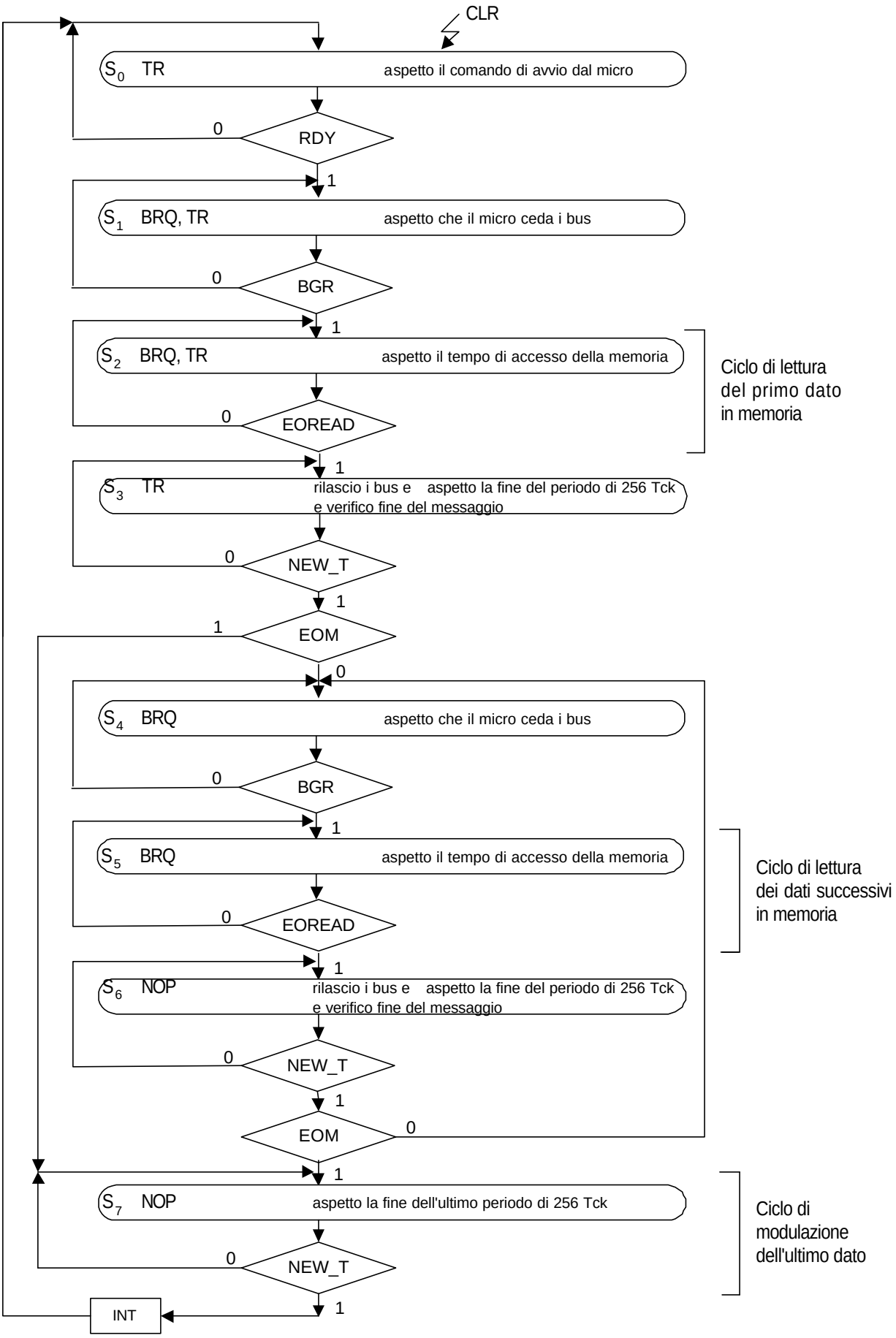
PPM: organizzazione dei dati della ROM

INDIRIZZI	DATI	
00000000	00000100	
00000001	00000100	SOGLIA
00000010	00000100	
00000011	00000100	
00000100	00000100	
00000101	00000101	
00000110	00000110	
⋮	DATI = INDIRIZZI	TRASPARENZA
11111010	11111010	
11111011	11111011	
11111100	11111100	
11111101	11111100	
11111110	11111100	SATURAZIONE
11111111	11111100	

Note

La ROM ha dimensioni 256 x 8 bit; infatti, trasforma un byte in un byte.

PPM - flowchart



PPM - flowchart

Note

Gli stati S1, S2, S3 gestiscono il transitorio iniziale: effettuano la lettura del primo dato e in questo primo ciclo di 256 Tck mantengono il segnale di uscita a 0, mediante l'attivazione del bit TR.

Gli stati S4, S5, S6 gestiscono il regime: effettuano la lettura dei dati successivi al primo, modulano i dati letti nei cicli di 256 Tck precedenti.

Lo stato S7 gestisce il transitorio finale: non effettua alcun accesso in memoria, modula l'ultimo dato del messaggio, letto nel ciclo di 256 Tck precedente.

I segnali di TASK: BRQ, TR sono diretti a dispositivi asincroni e quindi devono essere prelevati da uscite di flip-flop (per garantire l'assenza di alee, che potrebbero provocare false commutazioni nei dispositivi di destinazione). Diversamente, il bit INT è diretto a un flip-flop JK predisposto nell'interfaccia di interruzione, e quindi può essere emesso anche in modo combinatorio; per questi motivi il diagramma di flusso è stato descritto con la notazione di tipo Mealy (cfr. emissione del bit INT), con la variazione di associare le uscite registrate (BRQ, TR) direttamente agli stati successivi, in cui effettivamente si attivano, in linea con la notazione di tipo D-Mealy.

Il diagramma di flusso ammette due salti a stati non consecutivi e diversi dallo stato codificato con 0: da S3 a S7 e da S6 a S4. Tale situazione può ancora essere gestita tramite una struttura di controllo basata su componenti MSI: è necessario utilizzare un contatore precaricabile con i codici degli stati di destinazione (in questo caso: 0, 4, 7), piuttosto che semplicemente resettabile, e ovviamente dotato di ingresso di abilitazione al conteggio, per raggiungere gli stati con codice incrementato.

PPM : SCO - struttura HW a sequenziatore

Calcolo dei parametri del circuito

· Equazione di abilitazione all'incremento del conteggio (stato)

CE = S0 RDY + S1 BGR + S2 EOREAD + S3 NEW_T EOM* + S4 BGR + S5 EOREAD + S6 NEW_T EOM

· Equazione di caricamento dello stato del contatore

LE = S3 NEW_T EOM + S6 NEW_T EOM* + S7 NEW_T

· Equazioni dei bit del dato precaricabile nel contatore
D0 = S3
D1 = S3
D2 = S3 + S6
(Infatti S3 -> S7 (111); S6 -> S4 (100))

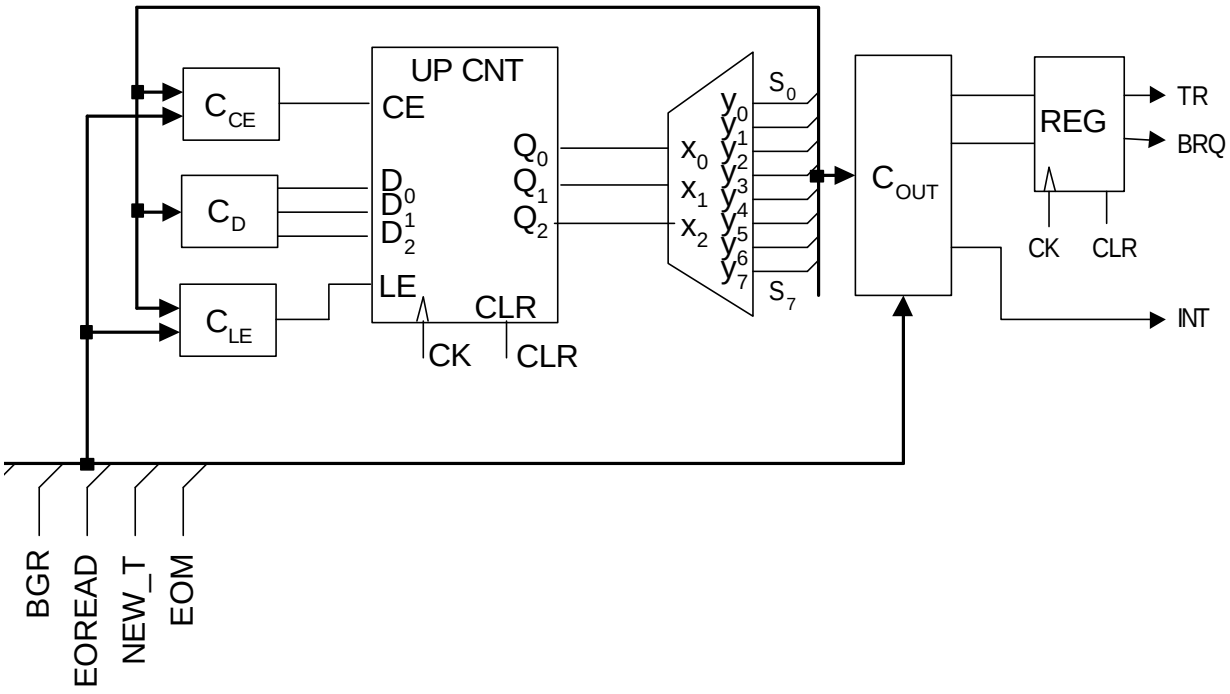
· Equazioni delle variabili entranti nel registro di uscita

TR = S0 + S1 + S2 + S3 NEW_T* + S7 NEW_T
BRQ = S0 RDY + S1 + S2 EOREAD* + S3 NEW_T EOM* + S4 + S5 EOREAD* + S6 NEW_T EOM*

· Equazioni delle variabili di uscita non registrate

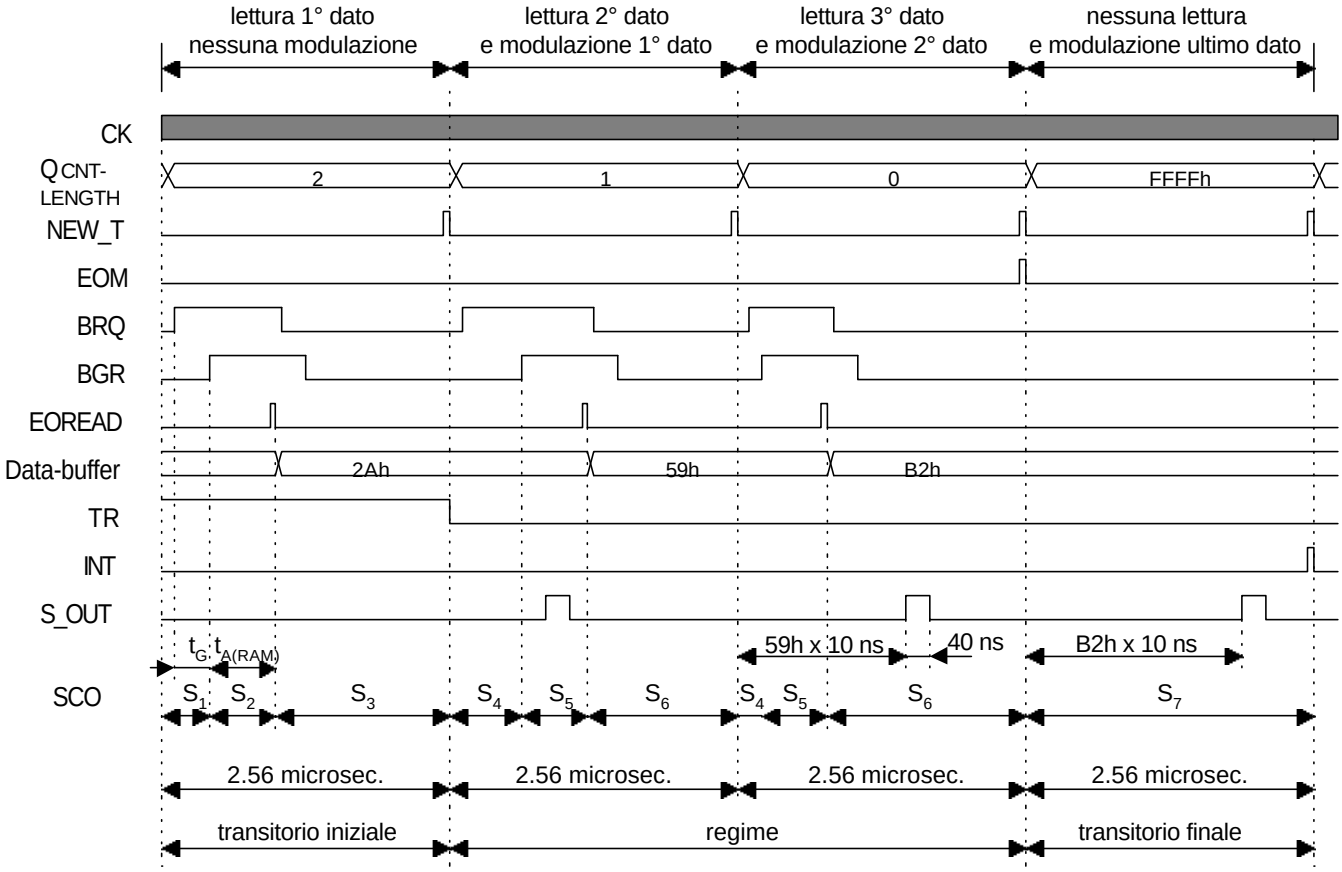
INT = S7 NEW_T

Circuito dello SCO

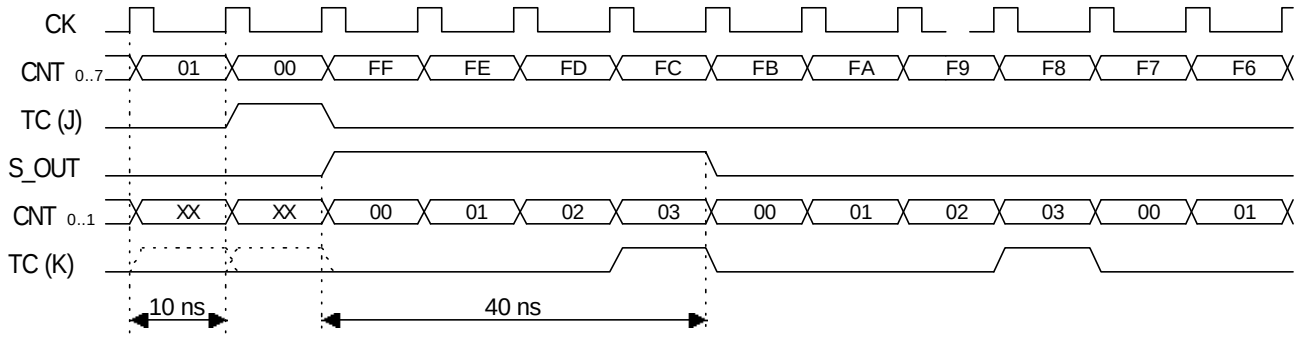


PPM: temporizzazioni

Sistema



Generazione dell'impulso di uscita



Note

Il diagramma temporale relativo alla modulazione di 3 dati evidenzia che il micro deve caricare il valore N-1 nel registro-contatore della lunghezza del blocco per modulare N dati.

PPM: Routine Software

```
ORG 400h

BLOCK_ADDR DL 00FF0000h          ;indirizzo (32 bit) iniziale del blocco dati
BLOCK_LENG DW 01000h             ;lunghezza (16 bit) del blocco dati
FLAG DB 0

Addr0 EQU 0A5h
Addr1 EQU 0A6h

stack       equ 2800h   ;inizio area di stack
ppmivn      equ 00Bh    ;ivn PPM
ppmdr       equ 1000h   ;indirizzo driver PPM
…
CODE

movl #stack,r7           ;inizializzo R7 (SP)
seti
…
mainloop:
…
cmpb #0,FLAG             ;test su messaggio spedito
jz altro
jsr start_ppm            ;richiesta di trasmissione blocco dati
                         ;(si suppone predisposto in memoria)
altro:
;predisposizione del blocco dati successivo
;altri task
jmp mainloop

; * * * R O U T I N E * * *

start_ppm:
outl BLOCK_ADDR,Addr0        ;caricamento indirizzo iniziale blocco
outw BLOCK_LENG,Addr1        ;caricamento lunghezza blocco
movb #00,FLAG                ;reset flag di blocco trasmesso
ret

; * * * DRIVER * * *

driver ppmivn,ppmdr
movb #01,FLAG                ;set flag di blocco trasmesso
rti
```

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 21-9-2001

STUDENTE:_____DOCENTE:_____

D1 Si definisce derivata di una funzione booleana $f(x_0, x_1, .., x_{N-1})$ rispetto alla variabile x_j la funzione:

$$\delta f(x_0, x_1, .., x_{N-1}) / \delta x_j = [f(x_0, x_1, .., x_{N-1})]_{xj=0} \oplus [f(x_0, x_1, .., x_{N-1})]_{xj=1}$$

Calcolare le espressioni delle due derivate rispetto a x_1 e a x_4 della funzione booleana:
$f = x_0 x_1 + x_1 x_2 + x_2 x_3$

D2 Sintetizzare una rete combinatoria a tre ingressi e due uscite M e U che esprimano rispettivamente la maggioranza e l'unanimità dei bit di ingresso a 1.

D3 Descrivere in dettaglio le matrici dei transistori di una PLA che implementa le funzioni:
$f_0 = a\ b' + a'\ b$
$f_1 = a\ c + a'\ b$
$f_2 = b\ d + a\ b'$
$f_3 = a\ c + b\ d$

D4 Dimensionare lo SCO e indicare l'organizzazione dei microprogrammi di un processore con 32 microprogrammi, di cui uno di fetch; la lunghezza dei microprogrammi varia da un minimo di 50 a un massimo di 200 stati; i microprogrammi devono poter effettuare un test simultaneo su 4 variabili, selezionabili in modo qualsiasi tra le 16 variabili di ingresso; i bit di task sono 32; lo SCO deve essere di tipo D-Mealy.

D5 Scrivere una routine assembler PD32 per conteggiare il numero di bit pari a 1 nella longword in memoria all'indirizzo predisposto in r0; il risultato va restituto nel registro r1. Calcolare la lunghezza in byte della routine.

Esercizio (2S20010921-D1)

Si definisce derivata di una funzione booleana $f(x_0, x_1, .., x_{N-1})$ rispetto alla variabile x_j la funzione:

$$\partial f(x_0, x_1, .., x_{N-1}) / \partial x_j = [f(x_0, x_1, .., x_{N-1})]_{xj=0} \oplus [f(x_0, x_1, .., x_{N-1})]_{xj=1}$$

Calcolare le espressioni delle due derivate rispetto a x_1 e a x_4 della funzione booleana:
$f = x_0\, x_1 + x_1\, x_2 + x_2\, x_3$

---

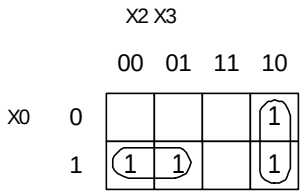
1. Derivazione rispetto a x_1

$[x_0\, x_1 + x_1\, x_2 + x_2\, x_3]_{x1=0} = x_2\, x_3$

$[x_0\, x_1 + x_1\, x_2 + x_2\, x_3]_{x1=1} = x_0 + x_2 + x_2\, x_3 = x_0 + x_2$

dove nell'ultimo passaggio è stato applicato il teorema dell'assorbimento.
L'espressione minima della funzione derivata può essere ottenuta tabellando le due funzioni ricavate sopra, calcolandone lo XOR punto per punto, e quindi minimizzando la funzione trovata con una MK:

$x_0\, x_2\, x_3$	$x_2 x_3$	$x_0 + x_2$	$\partial f / \partial x_1$
0 0 0	0	0	0
0 0 1	0	0	0
0 1 0	0	1	1
0 1 1	1	1	0
1 0 0	0	1	1
1 0 1	0	1	1
1 1 0	0	1	1
1 1 1	1	1	0



da cui:

$\partial f / \partial x_1 = x_0\, x_2' + x_2\, x_3'$

2. Derivazione rispetto a x_4

La funzione f non dipende da x_4 e quindi:

$[f]_{x1=0} = [f]_{x1=1} = f = x_0\, x_1 + x_1\, x_2 + x_2\, x_3$ da cui:

$\partial\, (x_0\, x_1 + x_1\, x_2 + x_2\, x_3) / \partial x_4 = f \oplus f = 0$

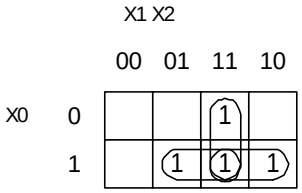
Esercizio (2S20010921-D2)

Sintetizzare una rete combinatoria a tre ingressi e due uscite M e U che esprimano rispettivamente la maggioranza e l'unanimità dei bit di ingresso a 1.

Dette x_0 x_1 x_2 le tre variabili di ingresso, la specifica va riportata sulla tavola di verità delle due funzioni M e U:

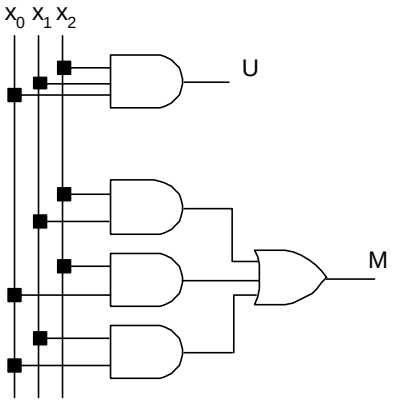
x_0	x_1	x_2	M	U
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Con un'ispezione visiva si ricava che: $U = x_0\ x_1\ x_2$
Per la funzione M la sintesi procede con la minimizzazione sulla MK:



Da cui: $M = x_0\ x_1 + x_0\ x_2 + x_1\ x_2$

La rete richiesta è tracciata nella figura seguente:



Esercizio (2S20010921-D3)

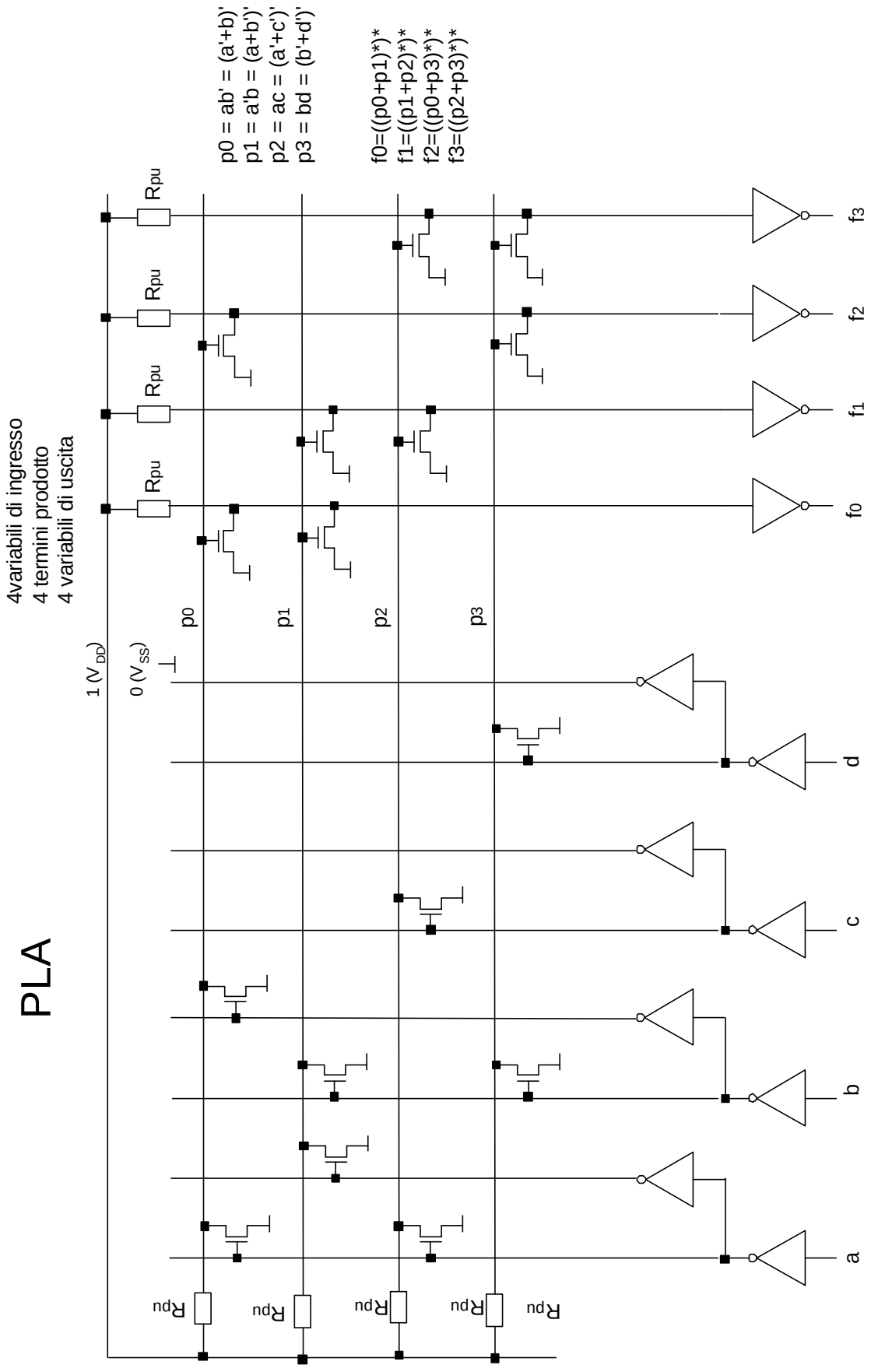
Descrivere in dettaglio le matrici dei transistori di una PLA che implementa le funzioni:

f_0= a b’ + a’ b
f_1= a c + a’ b
f_2= b d + a b’
f_3= a c + b d

Come è noto, entrambe le matrici di transistori (i “piani”) di ingresso e di uscita della PLA implementano funzioni di tipo NOR (cablato); pertanto è necessario riscrivere le espressioni AND-OR come NOR-NOR; per operare tale trasformazione si può applicare il teorema di De Morgan alle quattro espressioni separatamente:

f_0= a b’ + a’ b = (a’+b)’ + (a+b’)’ = (((a’+b)’ + (a+b’)’)’)’
f_1= a c + a’ b = (a’+c’)’ + (a+b’)’ = (((a’+c’)’ + (a+b’)’)’)’
f_2= bd + a b’ = (b’+d’)’ + (a’+b)’ = (((b’+d’)’ + (a’+b)’)’)’
f_3= a c + b d = (a’+c’)’ + (b’+d’)’ = (((a’+c’)’ + (b’+d’)’)’)’

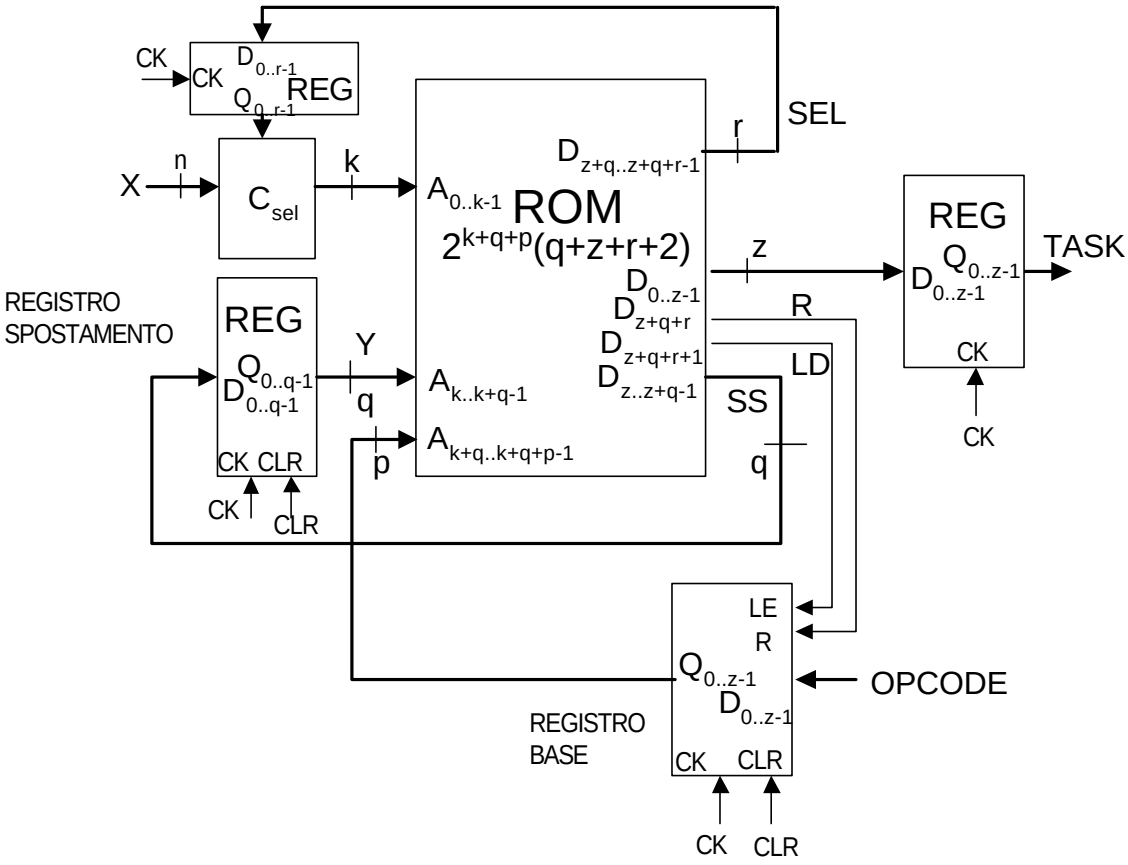
il livello di inversione più esterno ristabilisce la polarità corretta della funzione e, come è noto, viene tradotto in hardware con un invertitore sul livello di uscita, a valle dei due piani NOR-NOR.
Nella matrice di ingresso vanno connessi i transistori in corrispondenza degli incroci tra le singole righe dei prodotti che si vogliono realizzare e le colonne associate alle variabili che compaiono nelle ingole espressioni NOR di cui sopra, come riportato nella figura.


PLA
4variabili di ingresso
4 termini prodotto
4 variabili di uscita
p0 = ab' = (a'+b)'
p1 = a'b = (a+b')'
p2 = ac = (a'+c')'
p3 = bd = (b'+d')'
f0=((p0+p1)*)*
f1=((p1+p2)*)*
f2=((p0+p3)*)*
f3=((p2+p3)*)*
1 (VDD)
0 (VSS)
Rpu
p0
p1
p2
p3
a
b
c
d
f0
f1
f2
f3

Esercizio (2S20010921-D4)

Dimensionare lo SCO e indicare l'organizzazione dei microprogrammi di un processore con 32 microprogrammi, di cui uno di fetch; la lunghezza dei microprogrammi varia da un minimo di 50 a un massimo di 200 stati; i microprogrammi devono poter effettuare un test simultaneo su 4 variabili, selezionabili in modo qualsiasi tra le 16 variabili di ingresso; i bit di task sono 32; lo SCO deve essere di tipo D-Mealy.

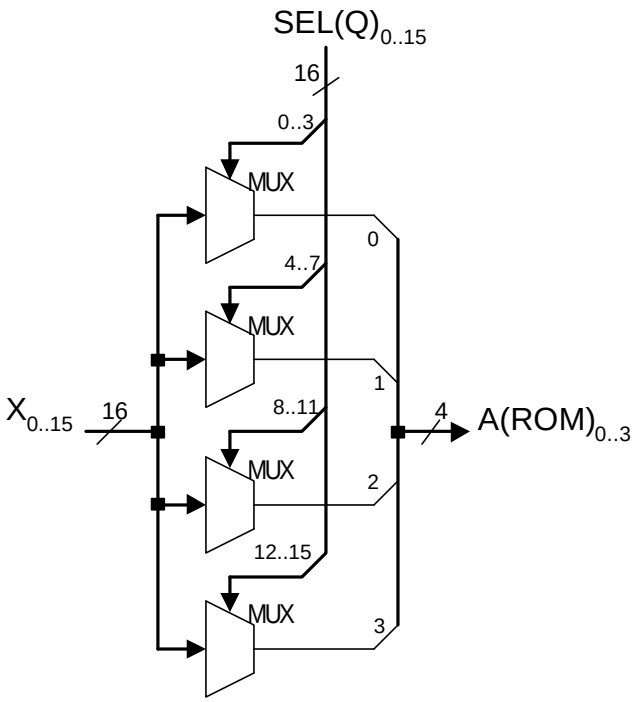
La struttura di uno SCO multimicroprogrammato di tipo D-Mealy (cfr. "Appunti integrativi")



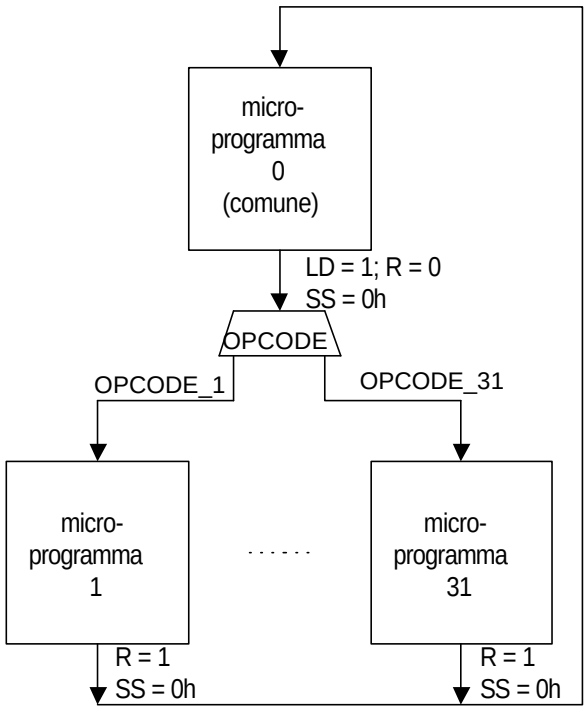
Il dimensionamento dei componenti deriva dalle specifiche:

numero dei microprogrammi=32 ⇒ p=5;
numero massimo degli stati per microprogramma=200 ⇒ q=8;
(il numero minimo degli stati per microprogramma non interessa)
z=32 (come da specifica);
k=4 (come da specifica);
il calcolo di r è basato sulla specifica di poter selezionare 4 variabili indipendenti da un gruppo di 16: sono necessari 4 MUX uguali, ciascuno con 16 ingressi di dato e 4 ingressi di selezione; pertanto r=4x4=16. Con questa posizione la ROM assume l'organizzazione: 2^{17}x 58 bit. La ROM risulta segmentata in 32 pagine ciascuna di lunghezza 2^{12} celle in cui vanno memorizzate le 2^8 x 2^4 microistruzioni codificate, corrispondenti a 256 stati su cui viene effettuato il test su 4 variabili.

La rete Csel è descritta nella figura seguente:



Infine, l'organizzazione dei microprogrammi è descritta nella figura seguente:



in cui sono evidenziate le attivazioni dei segnali R e LD, oltre all'azzeramento del campo SS della microistruzione (cfr. "Appunti integrativi").

Esercizio (2S20010921-D5)

Scrivere una routine assembler PD32 per conteggiare il numero di bit pari a 1 nella longword in memoria all'indirizzo predisposto in r0; il risultato va restituto nel registro r1. Calcolare la lunghezza in byte della routine.

```
; conta1.asm

;conta il numero di bit pari a 1 nella longword in memoria
;all'indirizzo predisposto in r0; il risultato va restituto
;nel registro r1. E' indicata la lunghezza in byte della routine.

        org 400h                ;inizio programma

        lwditest dl 0F0F0F0Fh   ;longword per il test della routine

        code                    ;inizio codice istruzioni

main:   movl #0400h,r0          ;inizializza r0 per il test della routine
        xorl r1,r1              ;azzera r1 (sempre a scopo di test)
        jsr conta1

        halt                    ;arresta elaborazione

; ************************
; SUBROUTINES
; ************************

conta1:
        push r2         ;salva r2                       4 byte
        push r3         ;salva r3                       4 byte

        movb #31,r2     ;inizializza r2 al conteggio di 32  5 byte
        xorl r1,r1      ;inizializza l'accumulatore r1 a 0  4 byte
        movl (r0),r3    ;carica longword in r3           4 byte

bitsh:  rorl #1,r3      ;copia bit LSB nel Carry        4 byte
        adcb #0,r1      ;somma il Carry all'accumulatore 5 byte
        subb #1,r2      ;decrementa contatore           5 byte
        jnn bitsh       ;test su fine ciclo             8 byte

        pop r3          ;ripristina r3                  4 byte
        pop r2          ;ripristina r2                  4 byte
        ret             ;                               4 byte
        end             ;fine programma
```

La routine è lunga 55 byte.

```
; conta1.asm

;conta il numero di bit pari a 1 nella longword in memoria
;all'indirizzo predisposto in r0; il risultato va restituto
;nel registro r1. E' indicata la lunghezza in byte della routine.


  org 400h    ;inizio programma

  lwditest dl 0F0F0F0Fh ;longword per il test della routine

  code      ;inizio codice istruzioni

main: movl #0400h,r0    ;inizializza r0 per il test della routine
  xorl r1,r1    ;azzera r1 (sempre a scopo di test)
  jsr conta1

  halt      ;arresta elaborazione


; ************************
; SUBROUTINES
; ************************

conta1:
  push r2     ;salva r2       4 byte
  push r3     ;salva r3       4 byte

  movb #31,r2   ;inizializza r2 al conteggio di 32  5 byte
  xorl r1,r1    ;inizializza l'accumulatore r1 a 0  4 byte
  movl (r0),r3    ;carica longword in r3      4 byte

bitsh:  rorl #1,r3    ;copia bit LSB nel Carry    4 byte
  adcb #0,r1    ;somma il Carry all'accumulatore  5 byte
  subb #1,r2    ;decrementa contatore     5 byte
  jnn bitsh   ;test su fine ciclo     8 byte

  pop r3      ;ripristina r3        4 byte
  pop r2      ;ripristina r2        4 byte
  ret     ;         4 byte


  end     ;fine programma
```

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 25-10-2001

STUDENTE:_____DOCENTE:_____

Si vuole progettare un decodificatore (VIDEC) di segnali video criptati da un trasmettitore che inverte (da dx a sx) la normale scansione (da sx a dx) dei punti dell'immagine di alcune righe predeterminate. Il segnale video digitale D_IN in ingresso a VIDEC è costituito da una sequenza di campioni (pixel) a 8 bit sincronizzati da un segnale di clock CK e organizzati in una trama (immagine) di 600 righe x 800 pixel. Le righe della trama sono separate da un intervallo temporale prefissato, non noto a VIDEC: pertanto, l'inizio di ogni riga della trama viene segnalato da un impulso su una linea HS, l'inizio della prima riga di ogni trama anche da un impulso su una linea VS; entrambi gli impulsi sono attivi nel solo periodo di CK precedente a quello del primo pixel della riga cui si riferiscono.

VIDEC effettua la decriptazione del segnale agendo sulle singole righe:

- ripristina l'ordine di presentazione corretto (da sx a dx) dei pixel relativi alle righe che arrivano invertite;
- trasferisce inalterate in uscita le righe regolari.

L'indicazione delle righe da ripristinare (1 bit per riga: 0/1 indica una riga corretta/invertita) è contenuta in una ROM da 75 byte.
Per poter accedere agli 800 elementi di una generica riga in ordine inverso, VIDEC memorizza gli 800 elementi dell'intera riga in ingresso in un banco di chip RAM, che poi rileggerà durante l'intervallo relativo alla riga successiva; simultaneamente alla memorizzazione della riga in ingresso attuale VIDEC legge gli 800 elementi della riga precedente da un secondo banco di chip RAM identico al primo. Pertanto i due banchi di RAM vengono utilizzati simultaneamente, uno in lettura e l'altro in scrittura, e scambiati di ruolo (lettura/scrittura) con la cadenza delle righe. Si noti che la scrittura dei campioni delle righe nella RAM verrà effettuata sempre per indirizzi crescenti, mentre la lettura dovrà essere effettuata per indirizzi crescenti o decrescenti nei casi in cui la riga da leggere sia corretta o invertita rispettivamente. I campioni delle righe lette alternativamente dai due banchi devono essere inviati su una linea di uscita D_OUT a 8 bit sincrona con CK.
I chip RAM che costituiscono i due banchi hanno i dati a 8 bit e un tempo di ciclo di scrittura/lettura pari a 200 ns.
La frequenza di CK è pari a 16 MHz (corrispondente a un periodo pari a 62.5 ns).

Si richiede:

1. la temporizzazione delle operazioni;
2. lo schema logico di VIDEC;
3. l'organizzazione dei banchi RAM.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 25-10-2001

STUDENTE:_____DOCENTE:_____

D1 Una funzione f(x) si definisce *autoduale* se e solo se:

f(X) = f*(X*)

dove il simbolo * indica complementazione, riferita alla funzione f e alle singole variabili indipendenti del vettore X.
Indicare le funzioni autoduali di due variabili.

D2 Descrivere un moltiplicatore combinatorio a matrice romboidale per una coppia di operandi a 4 bit.

D3 Dati moduli di conteggio mod 2 e mod 5, assemblare un contatore mod 100.

D4 Indicare le strutture dei sistemi di controllo di supporto ai microlinguaggi di tipo 2 e 3 e motivarne l'applicabilità.

D5 Scrivere una routine assembler PD32 per calcolare i bit di parità tra i bit in posizione omologa di 32 longword disposte consecutivamente a partire dall'indirizzo BLOCK. I 32 bit di risultato devono essere scritti nella longword successiva.

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 10-01-2002

STUDENTE:_____DOCENTE:_____

Si vuole progettare una periferica per la localizzazione di una sequenza di brani musicali predisposti nella memoria di un processore PD32. Ciascun brano musicale è rappresentato in memoria da una sequenza di un numero non noto a priori di campioni a 16 bit; i brani sono separati da una sequenza di almeno N campioni di valore inferiore a una soglia prefissata TH.

Il processore PD32 invia alla periferica:
- gli indirizzi iniziale e finale del blocco di memoria dove è stata predisposta la sequenza dei brani musicali da localizzare;
- il valore N, a 24 bit;
- il valore TH, a 16 bit.

In risposta la periferica esegue le seguenti attività:
- accede al blocco di memoria specificato in DMA in modalità*burst*;
- in corrispondenza dell'inizio di ciascun brano restituisce i bus di memoria e invia un'interruzione al processore, che provvede a recuperare l'indirizzo iniziale del brano; quindi riprende l'esplorazione del blocco residuo di memoria in DMA;
- al termine dell'esplorazione dell'intero blocco di memoria invia un'interruzione di fine operazione al processore.

Si richiede:

- lo schema logico dettagliato della periferica e le relative temporizzazioni;
- il microprogramma;
- le routine di interruzione del PD32.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 10-01-2002

STUDENTE:_____DOCENTE:_____

D1 Convertire nella notazione binaria in virgola fissa il numero:
1936.27
con un errore inferiore a 10^{-3}; quindi indicarne la rappresentazione nella notazione in virgola mobile a 32 bit.

D2 Semplificare l'espressione booleana:
ab+bc+ca*
dove il simbolo * indica complementazione.

D3 Progettare la rete combinatoria da aggiungere a un flip-flop D per trasformarlo in un flip-flop T dotato di ingresso di reset sincrono.

D4 Progettare un circuito logico per aggiungere uno stato di wait ad ogni accesso effettuato dal PD32 a una pagina di memoria di lunghezza 256 Kbyte disposta all'indirizzo 1FFC0000h. Si supponga disponibile il segnale CK del PD32.

D5 Descrivere la disposizione in memoria dell'istruzione assembler PD32:
MOVL Alfa(R2),-(R3)
memorizzata all'indirizzo 10000001h; quindi indicare il numero dei cicli macchina e di bus eseguiti nel ciclo istruzione relativo nell'ipotesi che:
Alfa=01239ABDh;
R2=00000003h;
R3=0000FFFFh.

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 30-01-2002

STUDENTE:_____DOCENTE:_____

Si vuole realizzare una periferica per trasferire messaggi predisposti nella RAM di un processore PD32 su un canale di comunicazione binario.
Quando il processore vuole trasmettere un messaggio invia alla periferica:

- l'indirizzo iniziale del messaggio;
- la lunghezza, a 16 bit, espressa in byte, del messaggio;
- il rapporto, intero a 8 bit, tra la frequenza del clock – comune alla periferica e al processore - e la frequenza di cifra da impostare sul canale di comunicazione.

In risposta la periferica esegue le seguenti attività:

- preleva il messaggio mediante accesso in DMA di tipo stealing;
- ad ogni byte del messaggio aggiunge un bit di parità;
- per ogni bit del messaggio dotato di parità trasmette una coppia di bit secondo il codice (*Manchester*) riportato nella tavola seguente:

Bit	Codice
0	01
1	10

- segnala il termine della trasmissione di ogni blocco di lunghezza 1 Kbyte del messaggio originale al processore mediante interruzione;
- segnala il termine della trasmissione del messaggio al processore mediante interruzione.

Quando non è occupata a trasmettere un messaggio del processore, la periferica trasmette una serie di bit pari a 0, anche questi da codificare.
Il processore e la periferica utilizzano lo stesso clock.
Si supponga che il processore esegua cicli-macchina di durata variabile da 3 a 8 cicli di clock.

Si richiede:

1. la temporizzazione delle operazioni;
2. lo schema logico della periferica;
3. il calcolo del rallentamento percentuale del processore;
4. le routine d'interfacciamento del PD32.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 30-01-2002

STUDENTE:_____DOCENTE:_____

D1 Determinare il numero minimo di bit di controllo inseriti in un secondo in un flusso di dati seriale codificato per la correzione di singoli errori e quindi trasmesso alla velocità di 64 Kbit/s su un un canale di comunicazione binario che ne inverte al massimo un bit ogni mille.

D2 Progettare una rete combinatoria a 4 ingressi e 4 uscite con la funzione di ordinare il vettore di ingresso in senso crescente (esempi: 0100 $\rightarrow$ 0001; 1010 $\rightarrow$ 0011).

D3 Valutare la frequenza massima di lavoro di un sistema SCO-SCA di tipo Moore-Moore in funzione dei parametri dinamici delle due reti.

D4 Scrivere un programma assembler PD32 per determinare il maggiore di due numeri rappresentati in virgola mobile agli indirizzi alfa e beta. Il risultato (0/1: il massimo è in alfa/beta) va scritto all'indirizzo gamma.

D5 Una periferica produce dati al ritmo di 125 Kbit/s, e li trasferisce a pacchetti di N bit a una CPU bit mediante interrupt; il driver relativo viene eseguito in 10 microsecondi. Dimensionare la larghezza minima N del registro di interfaccia della periferica in modo che il trasferimento non impegni più del 5% del tempo di CPU.

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 14-02-2002

STUDENTE:_____DOCENTE:_____

Una periferica (MEDMOB) riceve in ingresso una grandezza analogica, la converte in digitale attraverso un convertitore A/D ad 8 bit, ne esegue la media mobile su una finestra di quattro campioni attraverso la formula seguente:

$M(k) = \Sigma_{i=0..3} S(k-i)/4$

Le medie mobili vengono inserite in una memoria RAM locale da 4Kbyte con tempo di ciclo di scrittura pari a 75 nsec.

Il PD32 avvia MEDMOB attraverso un apposito segnale. Una volta attivata, MEDMOB campiona l'ingresso analogico alla frequenza di 10Mhz.

A seguito del verificarsi di uno di questi due eventi:

1. la memoria RAM è piena,
2. 8 medie mobili consecutive superano un certo valore di soglia impostato dal PD32 in un registro interno alla periferica prima di iniziare le operazioni di acquisizione,

MEDMOB blocca l'acquisizione dati ed invia un interrupt al processore.
Nel caso si verifichi il primo evento, il PD32 provvede alla lettura dei dati dalla memoria RAM locale che vede nel suo spazio di indirizzamento a partire dalla locazione FFFFF000h.

Specifiche dispositivi

- Il convertitore A/D non emette segnale di fine della conversione ed ha un tempo di conversione di 60nsec;
- tempo di calcolo di comparatori e sommatori: 60 nsec;
- si considerino trascurabili tutti gli altri tempi;
- all'interno della RAM devono essere inserite soltanto medie mobili calcolate su quattro valori di ingresso;
- il ciclo di scrittura della RAM deve avvenire in tre periodi di clock.

Si richiede:

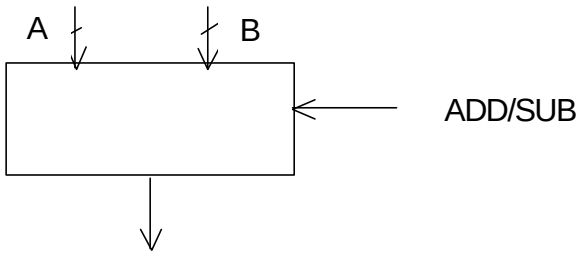
- lo schema logico di MEDMOB;
- la temporizzazione di MEDMOB;
- il software di gestione del PD32.

RETI LOGICHE

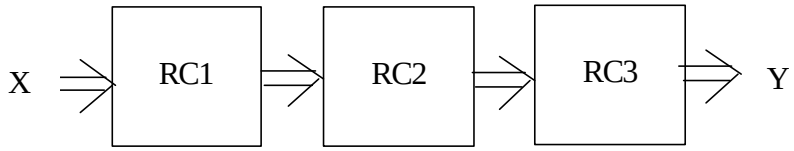
SECONDA PROVA SCRITTA DELL'APPELLO DEL 14-02-2002

STUDENTE:_____DOCENTE:_____

D1 Progettare una rete combinatoria che esegua su due operandi A e B rappresentati in complemento a due con n bit le operazioni A+B (se ADD/SUB=1) e A-B (se ADD/SUB=0).



D2 Si consideri un circuito formato da tre reti combinatorie in cascata: RC1, RC2 e RC3. I tempi di propagazione sono: T_{pmax1}= 30nsec, T_{pmin1}= 10nsec T_{pmax2}= 20nsec T_{pmin2}= 5nsec, T_{pmax3}= 50nsec T_{pmin3}= 20nsec. Calcolare la frequenza massima di variazione dell'ingresso alla rete RC1 considerando $\Delta x=0$ ed un tempo di stabilità minima Y dell'uscita di 10nsec.



D3 Trasformare il circuito della domanda D2 in un circuito a pipeline e definire la frequenza massima di funzionamento.

D4 Sintetizzare una rete sequenziale che si comporti come un contatore decimale "down" con ingresso CE (count enable) ed uscita TC (terminal count).

D5 Nella memoria PD32 a partire dall'indirizzo SAMPLES sono allocati 256 bytes che rappresentano i valori dei campioni acquisiti tramite un convertitore analogico-digitale a 8 bit. Si richiede di scrivere una routine assembler PD32 che calcoli i valori delle medie mobili definite su una finestra di quattro campioni ($M(k) = \Sigma_{i=0..3}\ S(k-i)/4$) e le memorizzi a partire dall'indirizzo MEDIA_MOBILE nella memoria del PD32.

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 19-04-2002

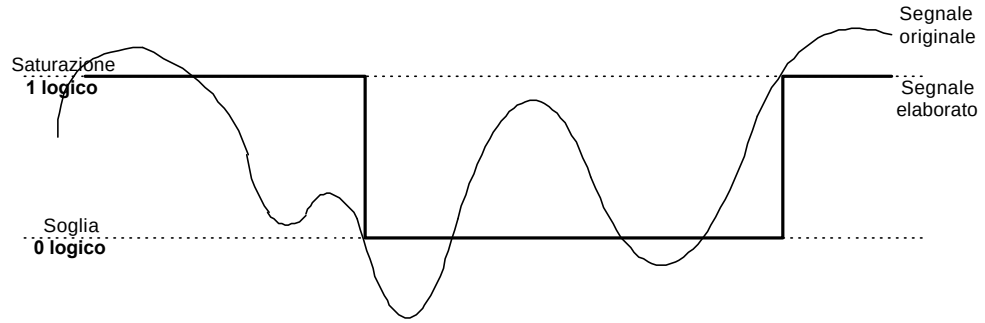
STUDENTE:_____DOCENTE:_____

Si vuole progettare una periferica per squadrare mediante isteresi un segnale (vedi figura) acquisito mediante un convertitore ADC a 8 bit nella memoria di un processore PD32.
Per avviare l'elaborazione il processore invia alla periferica:

- l'indirizzo iniziale del blocco di memoria in cui sono predisposti i campioni, a 8 bit, del segnale originale;
- la lunghezza, espressa a 16 bit, del blocco di memoria;
- l'indirizzo iniziale del blocco di memoria in cui vanno scritti i valori del segnale elaborato;
- una coppia di byte che rappresentano i valori di soglia e saturazione rispettivamente.

In risposta la periferica esegue le seguenti attività:

- accede alla memoria del processore in DMA a *burst* per prelevare i campioni del segnale originale;
- per ogni byte (valore di un campione) letto dalla memoria produce un singolo bit pari a:
  - 0 se il valore del campione è minore del valore di soglia;
  - 1 se il valore del campione è maggiore del valore di saturazione;
  - il valore del bit associato al campione precedente altrimenti (isteresi: vedi figura);
- aggrega i singoli bit prodotti a gruppi di 8, che scrive in memoria all'indirizzo specificato;
- segnala il termine dell'elaborazione mediante interruzione al processore.



Si richiede:

1. la temporizzazione delle operazioni;
2. lo schema logico della periferica;
3. il microprogramma;
4. le routine assembler del PD32.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 19-04-2002

STUDENTE:_____ DOCENTE:_____

D1 Un addizionatore iterativo a N bit è costruito con porte logiche di tipo NAND con tempo di propagazione pari a t_G; descrivere la struttura della cella e valutare il tempo minimo e quello massimo impiegato dalla rete per calcolare la somma di due operandi a N bit.

D2 Si vogliono memorizzare su ROM i valori della funzione sen(x) nel primo quadrante goniometrico, con x espresso con la coppia di numeri interi in virgola fissa [gradi, primi] ($0 \le$ gradi ≤ 90, $0 \le$ primi ≤ 59, $0°0' \le x \le 90°0'$) e sen(x) rappresentato in virgola mobile con 32 bit; assumendo la disponibilità di moduli ROM con organizzazione 4Kx8 bit, descrivere un'implementazione e la relativa organizzazione dei dati.

D3 Descrivere la cella di un contatore precaricabile dotato delle abilitazioni al conteggio e al caricamento.

D4 Descrivere la struttura di un sistema di controllo multimicroprogrammato per supportare un microlinguaggio di tipo 3.

D5 Scrivere una routine assembler PD32 per scambiare l'ordine dei byte ($B_3B_2B_1B_0 \rightarrow B_0B_1B_2B_3$) in ciascuna delle longword allineate in un blocco di memoria di indirizzo iniziale BASELW e lunghezza NLW.

RETI LOGICHE

PRIMA PROVA SCRITTA DELL'APPELLO DEL 4-6-2002

STUDENTE:_____DOCENTE:_____

Si vuole progettare un co-processore (FILE_PRO) che supporti il calcolo della lunghezza dei file predisposti nella memoria di un processore PD32.
Quando il micro vuole avvalersi di FILE_PRO gli invia:
- l'indirizzo iniziale, a 32 bit, di un file da elaborare;
- una stringa a 32 bit di terminazione del file;
- un comando di avvio dell'elaborazione.

In risposta FILE_PRO esegue le seguenti attività:
- accede alla RAM in DMA a "bus stealing";
- preleva i dati del file a partire dall'indirizzo iniziale e:
  a) fino al rilevamento della stringa di terminazione del file, oppure:
  b) fino alla lunghezza massima di 1 Mbyte;
- nel caso a) lancia un'interruzione al processore, che provvederà a recuperare il valore della lunghezza calcolata da un registro della periferica;
- nel caso b) lancia un'interruzione di allarme al processore.

FILE_PRO utilizza il clock del processore.

Si richiede:

1. l'hardware della periferica FILE_PRO;
2. il firmware di controllo;
3. il software di interfacciamento del PD32.

RETI LOGICHE

SECONDA PROVA SCRITTA DELL'APPELLO DEL 4-6-2002

STUDENTE:_____ DOCENTE:_____

D1 Progettare una rete combinatoria CMOS a pass-transistor per implementare la funzione:
$y = x_0 x_1 + x_0 x_2 + x_1 x_2$

D2 Dato un flip-flop D, realizzare un flip-flop T con ingresso di reset sincrono.

D3 Valutare la frequenza massima di ck di un sistema SCA-SCO di tipo Mealy-Moore in funzione dei parametri delle due reti.

D4 Descrivere la struttura di uno SCO multimicroprogrammato che supporti un microlinguaggio di tipo 2.

D5 Descrivere l'allocazione in memoria dell'istruzione PD32:
MOVL #99AA55FFh,(R0)
memorizzata a partire dall'indirizzo 30000003h, e indicare il numero dei cicli-macchina e dei cicli di bus eseguiti nel relativo ciclo-istruzione, nell'ipotesi che R0=80000000h.