Interfaccia IFBFY

Un'interfaccia per CPU PD-32 opera come coprocessore per il calcolo della *FFT butterfly*:

$$\begin{aligned} X &= A + W_k B \\ Y &= A - W_k B \end{aligned} \qquad \text{con } |W_k| \le 1,\ |A| < 1,\ |B| < 1$$

dove *A*, *B*, W_k, *X*, *Y* sono **numeri complessi**, le cui parti reali e immaginarie sono espresse in notazione fixed-point in complemento a 2, con 2 bit di parte intera e 30 bit di parte frazionaria (formato Q2.30). Il coefficiente W_k da utilizzare nel corso delle operazioni è identificato dall'indice *k* all'interno di una ROM residente sull'interfaccia, che contiene 4096 coefficienti. Per l'esecuzione delle operazioni aritmetiche, l'interfaccia deve utilizzare **un solo** modulo addizionatore per numeri interi a 32 bit e **un solo** modulo moltiplicatore per numeri interi con ingressi a 32 bit e uscita a 64 bit.

Il software di controllo dell'interfaccia è organizzato come subroutine che riceve come parametri:

- un intero *k*, indice del coefficiente W_k da utilizzare,
- il puntatore a una tavola residente in memoria che contiene gli operandi *A* e *B*;

Dopo aver consegnato all'interfaccia l'indice *k* e gli operandi *A* e *B*, la subroutine avvia le operazioni e, al completamento, preleva dall'interfaccia i risultati *X* e *Y*.

Progettare l'hardware dell'interfaccia e codificare la subroutine di controllo.

[*] Università di Roma "La Sapienza", Laurea Specialistica in Ingegneria Informatica. Tema assegnato all'appello di Reti Logiche del 12 ottobre 2006.

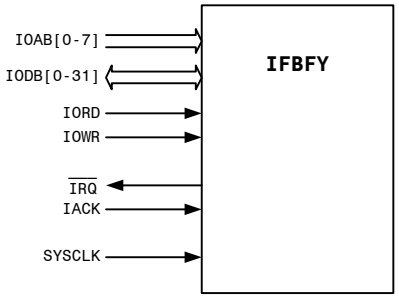


Fig. 1 – L'interfaccia è attestata sull'I/O Bus del PD-32; il completamento delle operazioni verrà segnalato mediante protocollo standard di interrupt. Viene anche utilizzato il System Clock come base dei tempi per le sequenze di operazioni.

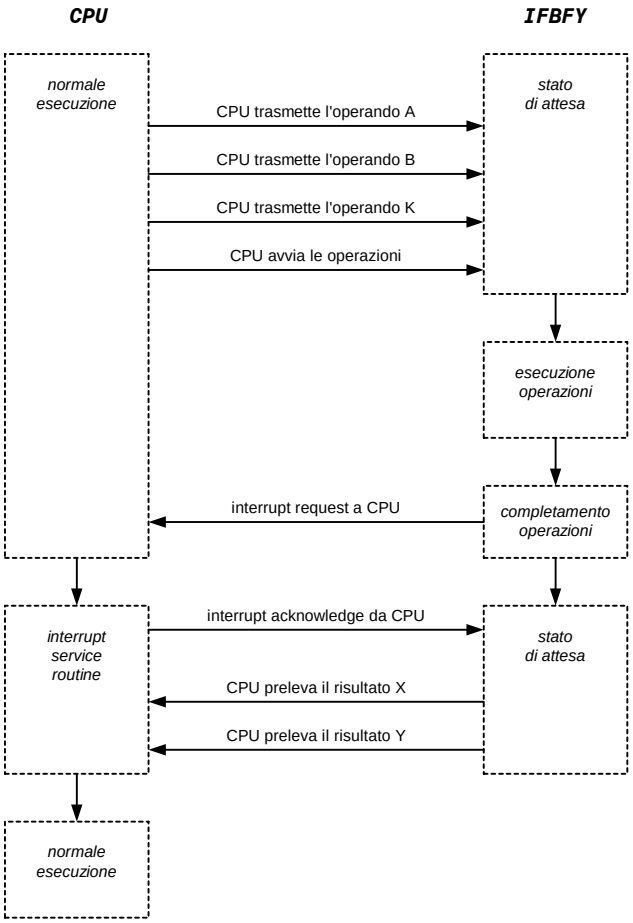


Fig. 2 – Sequenza degli eventi tra CPU e IFBFY.

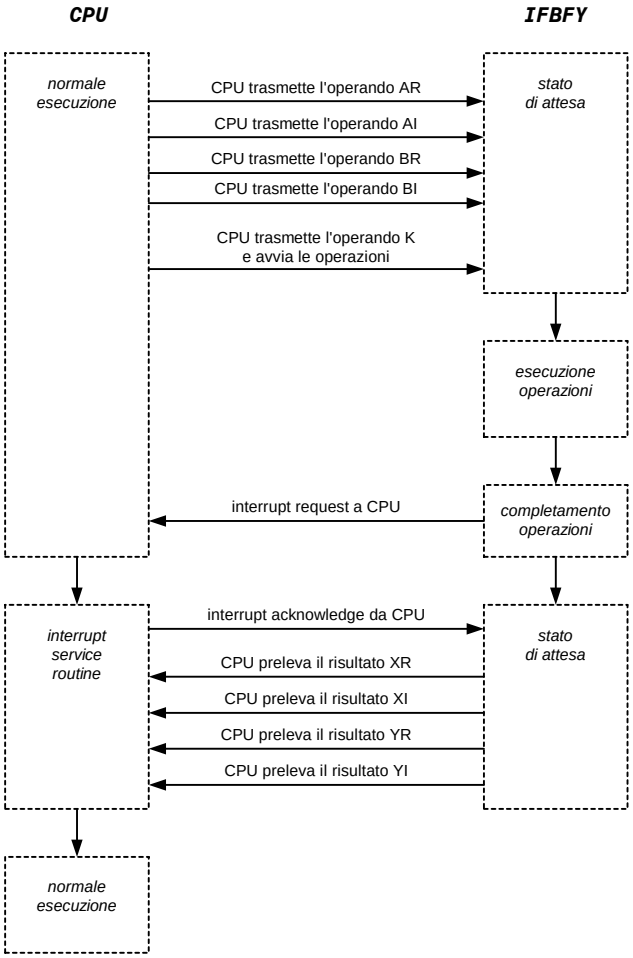


Fig. 3 – Sia gli operandi che i risultati sono numeri complessi, costituiti da parte reale e immaginaria ciascuna a 32 bit; essendo l'I/O Data Bus a soli 32 bit, la CPU deve trasmettere e ricevere le due parti separatamente. Inoltre, se si assume che l'operando K sia sempre l'ultimo ad essere trasmesso, questa operazione può essere interpretata anche come avvio delle operazioni su IFBFY, senza necessità di un comando apposito.

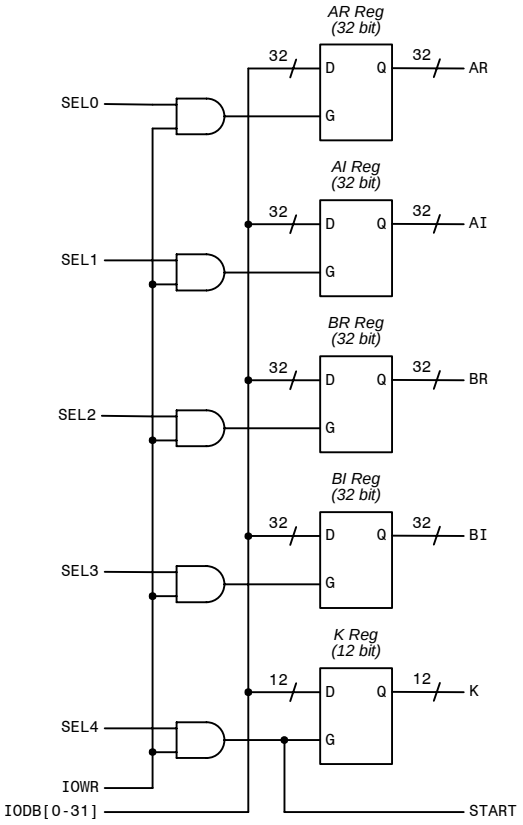


Fig. 4 – Porte di input, realizzate con D Latch, nelle quali vengono immagazzinati gli operandi trasmessi dalla CPU mediante istruzioni di output (che attivano i corrispondenti cicli di I/O Write sull'I/O Data Bus). Si noti come l'operando *K* sia un intero compreso tra 0 e 4095: il corrispondente registro è dunque da 12 bit, e ad esso si attestano solo i bit 0-11 (meno significativi) di IODB. Osserviamo anche come il segnale di START (inizio operazioni) venga generato nel momento stesso in cui la CPU scrive il valore dell'operando *K*. I segnali SEL di abilitazione al caricamento verranno derivati da opportune decodifiche dell'I/O Address Bus.

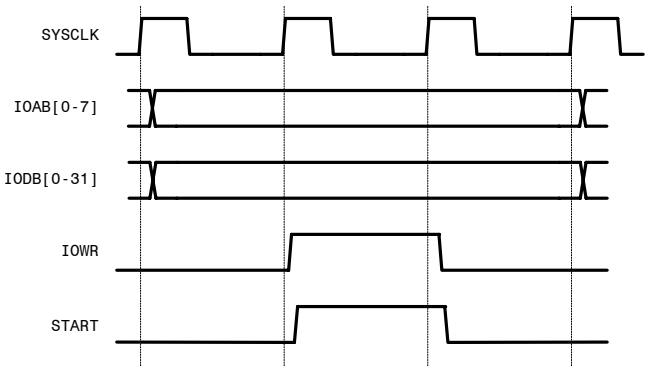


Fig. 5 – Il segnale START, essendo derivato da un ciclo di I/O Write, risulterà sincronizzato al System Clock.

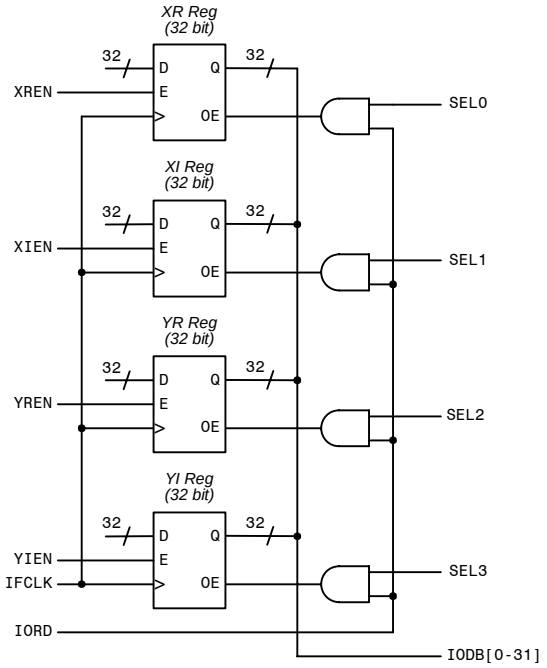


Fig. 6 – I risultati *X* e *Y* vengono memorizzati su registri D con uscite tri-state, mediante le quali si realizzano le funzioni di porte di input verso l'I/O Data Bus. Il clock di questi registri è il clock di interfaccia IFCLK, che sarà definito più avanti. Gli ingressi dati ai registri rimangono per il momento non specificati; i segnali di abilitazione al caricamento verranno generati da SCO. Osserviamo come, non essendoci possibilità di conflitto, i segnali SEL per la lettura delle rispettive porte di input possono coincidere con i segnali SEL di Fig. 4.

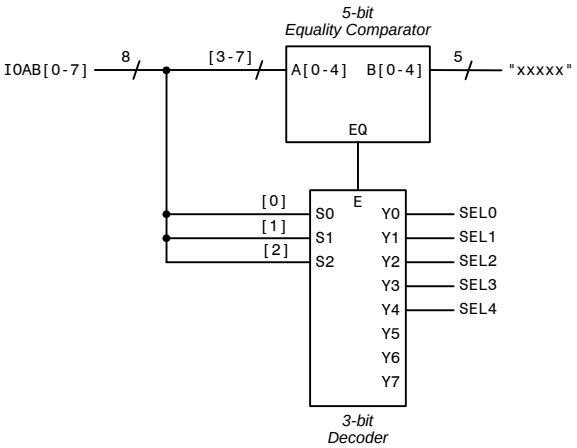


Fig. 7 – Generazione dei segnali SEL. L'I/O Address viene separato in due componenti: i 5 bit più significativi si assumono costanti e uguali a una costante logica cablata ("*xxxxx*" nella figura), mentre i 3 bit meno significativi vengono decodificati in modo da identificare univocamente le porte di input/output. A meno dei circuiti per la generazione dell'interrupt, che prenderemo in considerazione in seguito, il sottosistema di I/O è adesso completo.

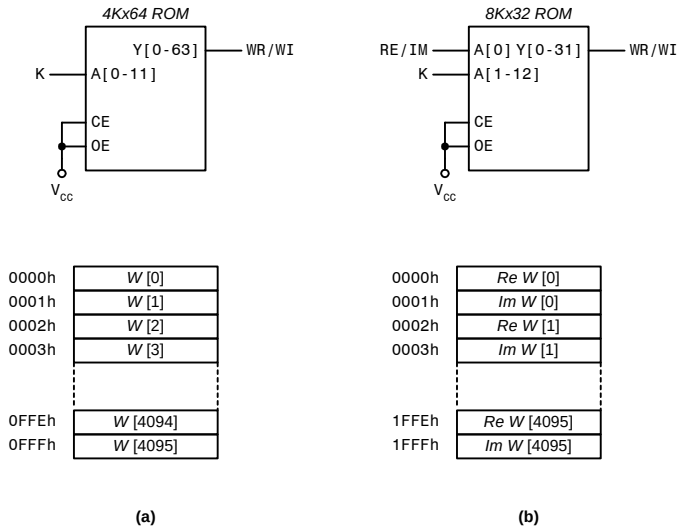


Fig. 8 – L'operando *K* deve essere usato come indice per accedere a uno di 4096 possibili valori *W*[*k*] residenti in ROM. Poiché ciascun *W* è a sua volta un numero complesso, costituito da una parte reale e una immaginaria, ciascuna a 32 bit, vi sono due possibili organizzazioni della ROM: (*a*) ciascuna parola di ROM contiene entrambe le parti (reale e immaginaria) di *W*, l'indirizzo della ROM coincide con *K*, e la ROM è organizzata in 4K parole da 64 bit; (*b*) le parti reale e immaginaria di ciascun *W* sono allocate in parole consecutive della ROM, per cui l'indirizzo meno significativo di ROM sarà utilizzato per accedere a ciascuna delle due parti, mentre i 12 bit di *K* vanno applicati ad altrettante linee di indirizzo di ROM; quest'ultima è pertanto organizzata in 8K parole da 32 bit. L'organizzazione (*a*) sarà da utilizzare qualora sia necessario disporre contemporaneamente della parte reale e di quella immaginaria di *W*, mentre l'organizzazione (*b*) sarà da preferire se le due parti non dovranno essere contemporaneamente disponibili, poiché in tal modo si evita la necessità di un multiplexer per la selezione di una delle due parti. La scelta tra le due organizzazioni viene dunque demandata a una fase successiva, quando saranno chiari i requisiti sulla disponibilità delle due parti di *W*.

(a)

$$A = A_R + jA_I$$
$$B = B_R + jB_I$$
$$W = W_k = W_R + jW_I$$
$$X = X_R + jX_I$$
$$Y = Y_R + jY_I$$

(b)

$$Z = WB =$$
$$= (W_R B_R - W_I B_I) + j(W_R B_I + W_I B_R) =$$
$$= Z_R + jZ_I$$

(c)

$$X_R + jX_I = A_R + jA_I + (W_R + jW_I)(B_R + jB_I) =$$
$$= A_R + jA_I + (W_R B_R - W_I B_I) + j(W_R B_I + W_I B_R) =$$
$$= (A_R + Z_R) + j(A_I + Z_I)$$
$$Y_R + jY_I = A_R + jA_I - (W_R + jW_I)(B_R + jB_I) =$$
$$= A_R + jA_I - (W_R B_R - W_I B_I) + j(W_R B_I + W_I B_R) =$$
$$= (A_R - Z_R) + j(A_I - Z_I)$$

Fig. 9 – Con le posizioni (*a*) e (*b*), le computazioni che il Sottosistema di Calcolo (SCA) dovrà eseguire sono illustrate in (*c*). Osserviamo come la variabile temporanea complessa *Z* intervenga sia nel calcolo di *X* che nel calcolo di *Y*, e dunque potrà essere valutata una sola volta.

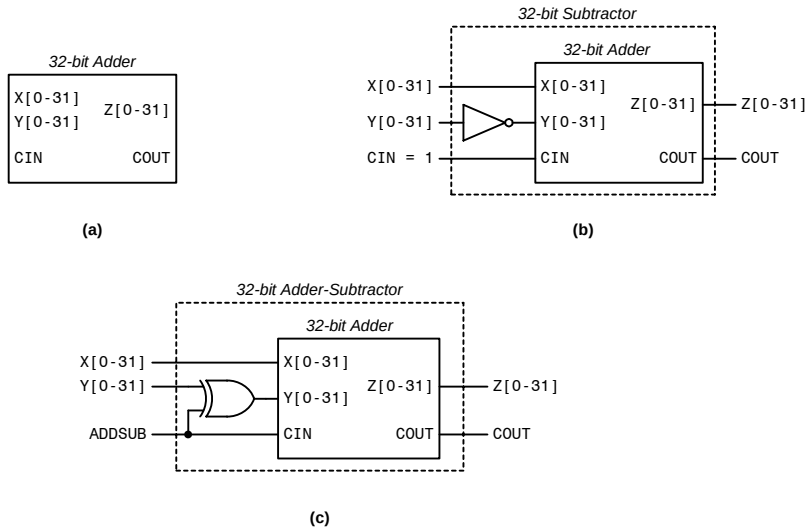


Fig. 10 – Nel corso dei calcoli, è necessario eseguire sottrazioni, oltre che addizioni. Un modulo addizionatore (*a*) può essere trasformato in un circuito sottrattore semplicemente complementando a 1 il secondo operando e forzando CarryIn = 1 (*b*). Le due operazioni, addizione e sottrazione, possono allora essere eseguite da un singolo circuito (*c*) dotato di ingresso di controllo ADDSUB: se ADDSUB = 0, il circuito esegue un'addizione, se ADDSUB = 1 esso esegue una sottrazione.

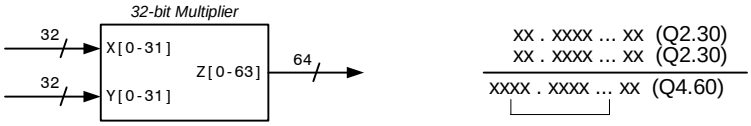


Fig. 11 – Il modulo moltiplicatore, illustrato a sinistra, produce un risultato a 64 bit. Poiché uno degli operandi è sempre una componente di W, che secondo specifiche ha modulo non maggiore di 1, allora il risultato della moltiplicazione è ancora esprimibile in formato Q2.30 senza possibilità di overflow. Ciò significa che, dei 64 bit in uscita dal moltiplicatore, riterremo soltanto gli ultimi 2 bit della parte intera (bit 61-60) e i primi 30 della parte frazionaria (bit 59-30), scartando gli altri.

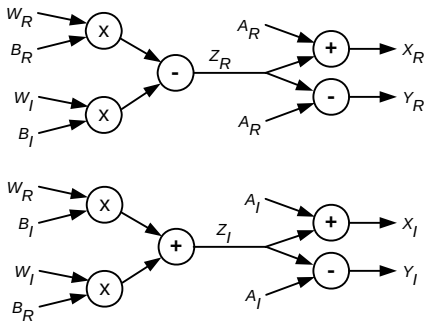


Fig. 12 – La sequenza dei calcoli che devono essere eseguiti dallo SCA, secondo le equazioni di Fig. 9.

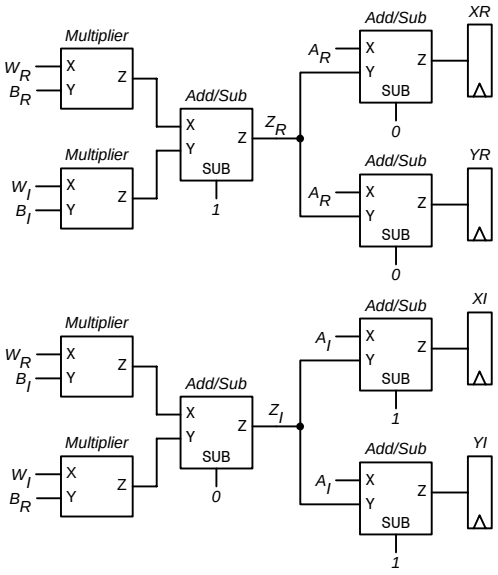


Fig. 13 – Se non avessimo vincoli sul numero di moduli aritmetici da utilizzare, i calcoli richiesti potrebbero essere eseguiti con questa struttura.

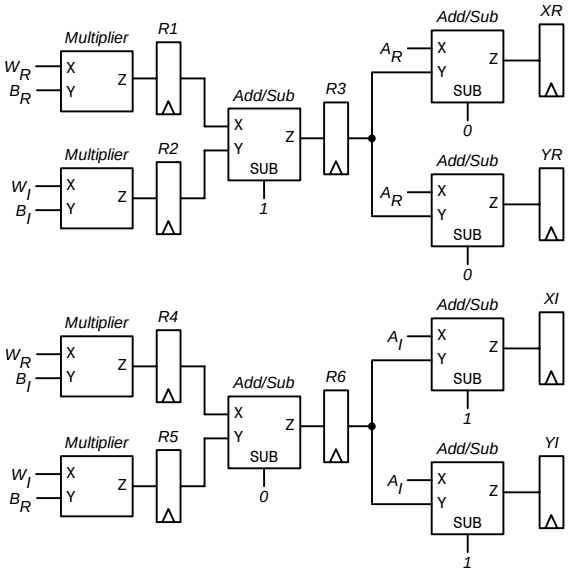


Fig. 14 – Tuttavia, dovendo utilizzare un solo addizionatore e un solo moltiplicatore, è chiaro che le varie operazioni dovranno essere eseguite una per volta, e di conseguenza sarà necessario memorizzare i risultati intermedi in appositi registri.

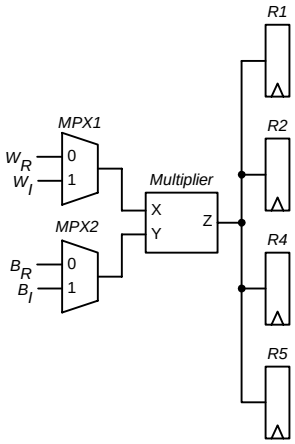


Fig. 15 – Dalla Fig. 14 appare chiaro che il moltiplicatore dovrà accettare come primo operando WR oppure WI, come secondo operando *BR* oppure *BI*, e il suo risultato dovrà essere scritto su uno dei registri temporanei R1, R2, R4, R5. La struttura che ne deriva è qui illustrata. Osserviamo che non è necessario disporre contemporaneamente degli operandi *WR* e *WI*, e pertanto la ROM dei coefficienti *Wk* avrà la struttura mostrata in Fig. 8(b).

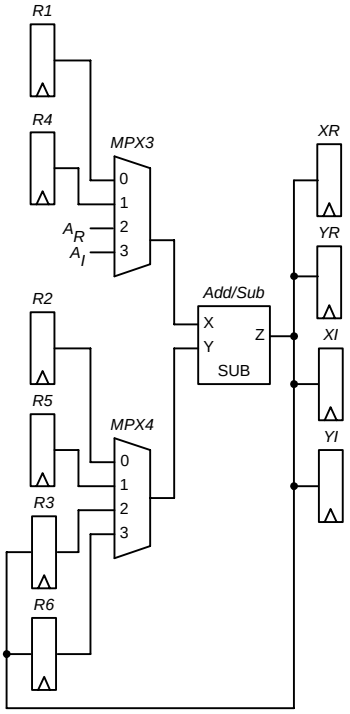


Fig. 16 – Ancora dalla Fig. 14, si ricava che il modulo addizionatore/sottrattore dovrà accettare come primo operando R1, R4, AR oppure AI, come secondo operando R2, R3, R5 oppure R6, e il suo risultato dovrà essere scritto in XR, YR, XI, YI, R3 oppure R6. È qui illustrata la struttura risultante.

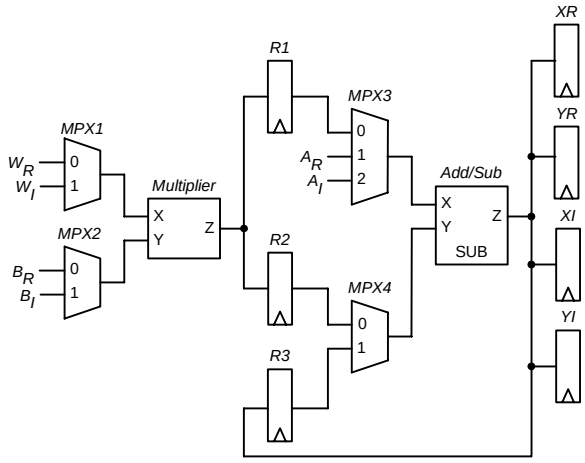


Fig. 17 – Osserviamo che, se il calcolo delineato in Fig. 14 viene condotto con opportuna sequenza, eseguendo prima la parte superiore relativa a XR e YR e poi la parte inferiore relativa a XI e YI, i registri R4, R5 e R6 diventano superflui, potendo riutilizzare al loro posto i registri R1, R2 e R3 rispettivamente. Tenendo conto di questa osservazione, i circuiti di Fig. 15 e Fig. 16 possono essere riuniti nello schema qui illustrato. Tutti i multiplexer illustrati verranno realizzati utilizzando le uscite tri-state dai registri.; fa eccezione il multiplexer per WR e WI che, come osservato in Fig. 15, può essere immaginato come incorporato entro la ROM.

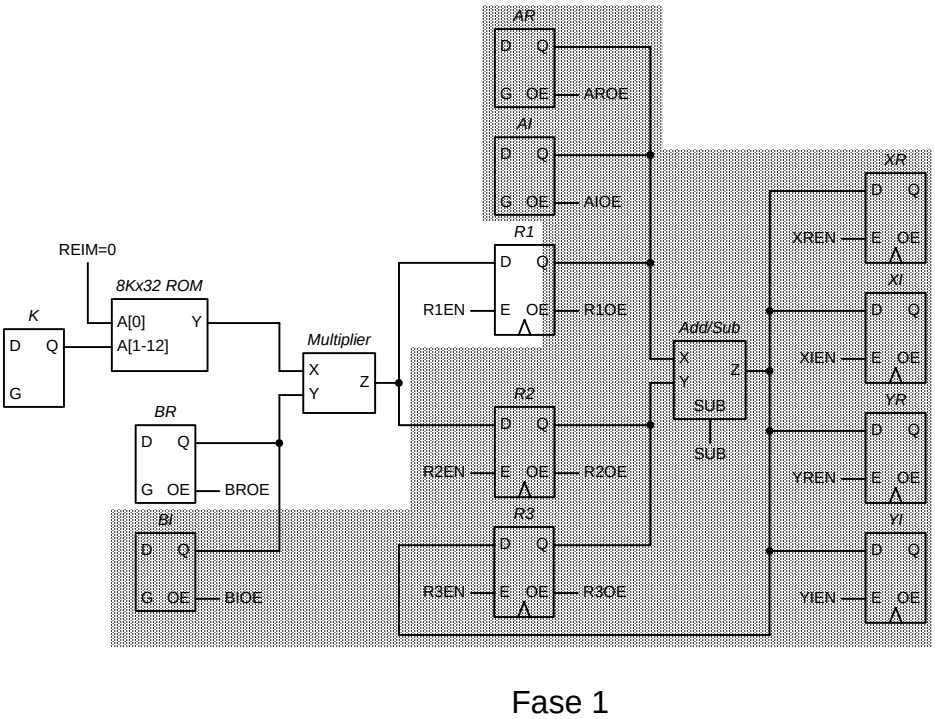
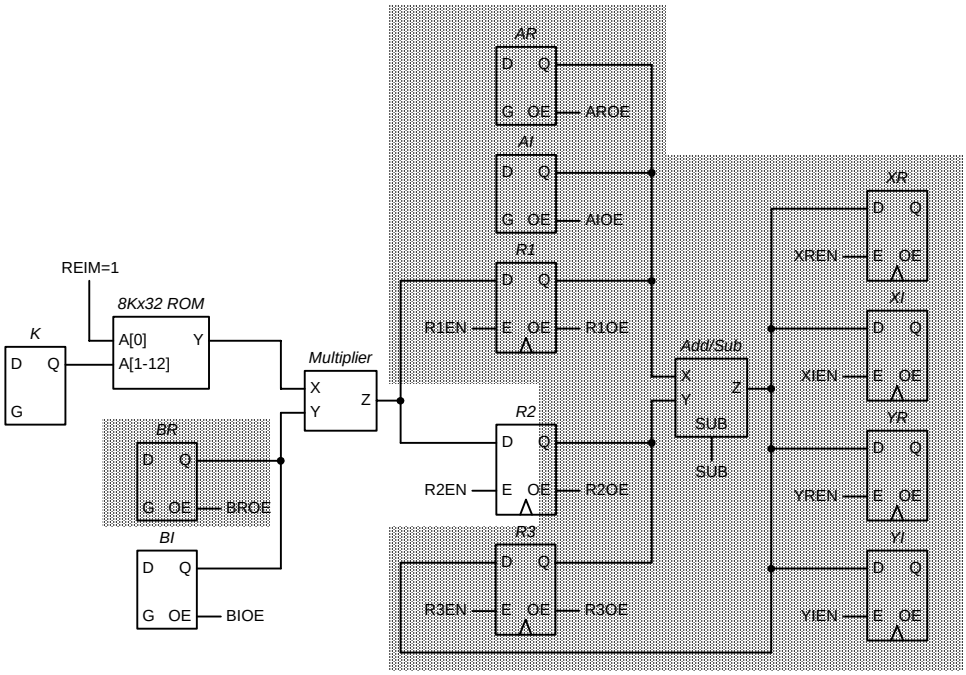
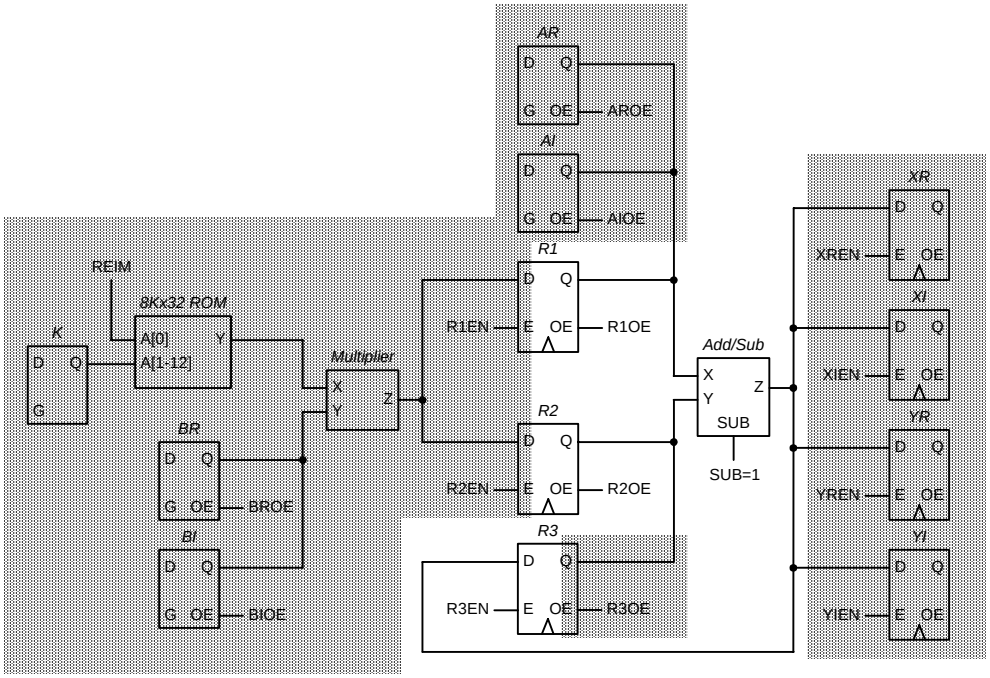


Fig. 18 – La prima fase delle operazioni consiste nel calcolo del prodotto tra WR e BR, con risultato immagazinato in R1.



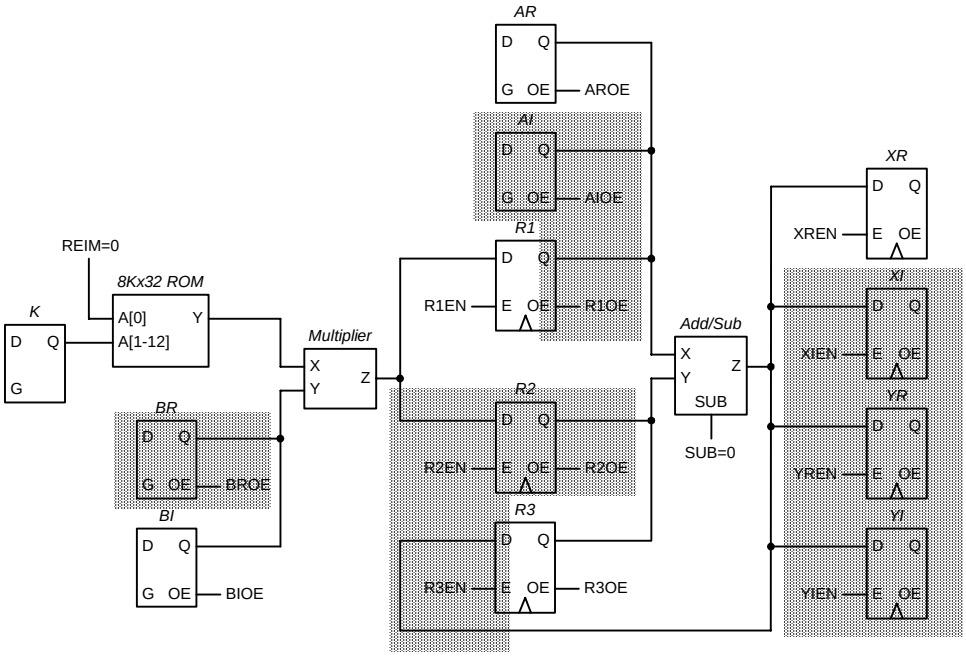
Fase 2

Fig. 19 – La fase 2 consiste nel calcolo del prodotto tra WI e BI, con memorizzazione del risultato in R2.



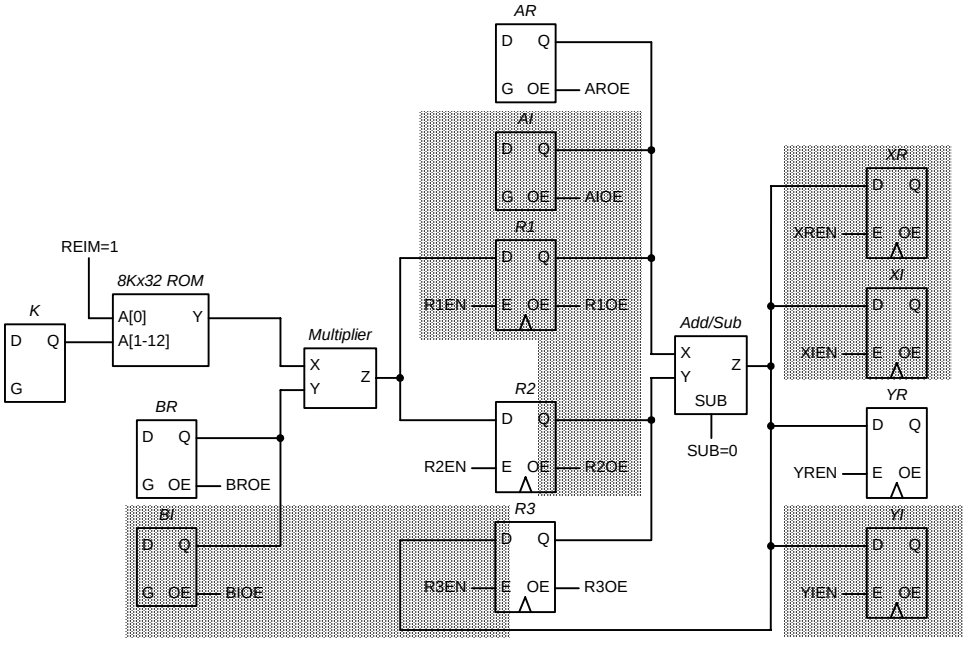
Fase 3

Fig. 20 – La fase 3 consiste nel calcolo di ZR, differenza tra il contenuto di R1 e il contenuto di R2, e nella sua memorizzazione in R3.



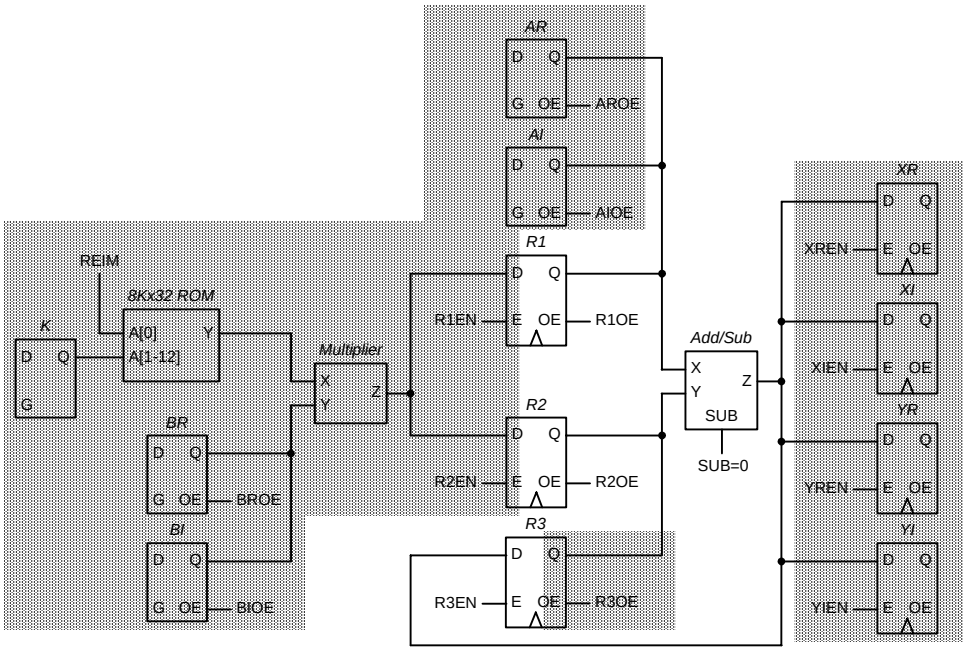
Fase 4

Fig. 21 – Il calcolo di XR e YR coinvolge solo il modulo addizionatore/sottrattore, mentre il modulo moltiplicatore è libero da impegni e dunque può essere utilizzato per iniziare i calcoli illustrati nella porzione inferiore dello schema di Fig. 14. Pertanto, la fase 4 consiste sia del calcolo di XR che dell'esecuzione del prodotto tra WR e BI.



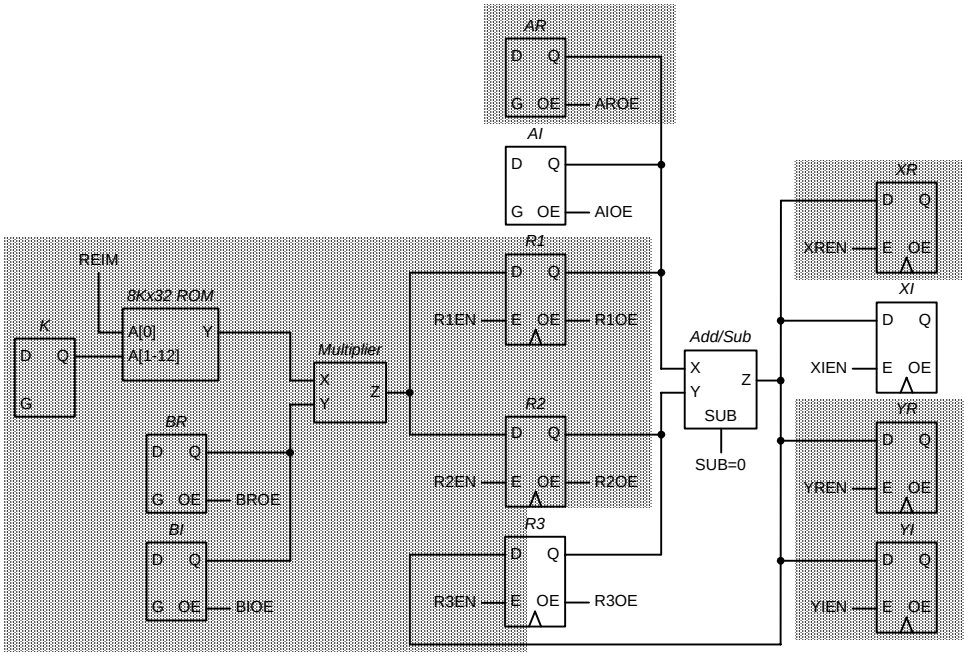
Fase 5

Fig. 22 – La fase 5 consiste sia del calcolo di YR che in quello del prodotto tra WI e BR.



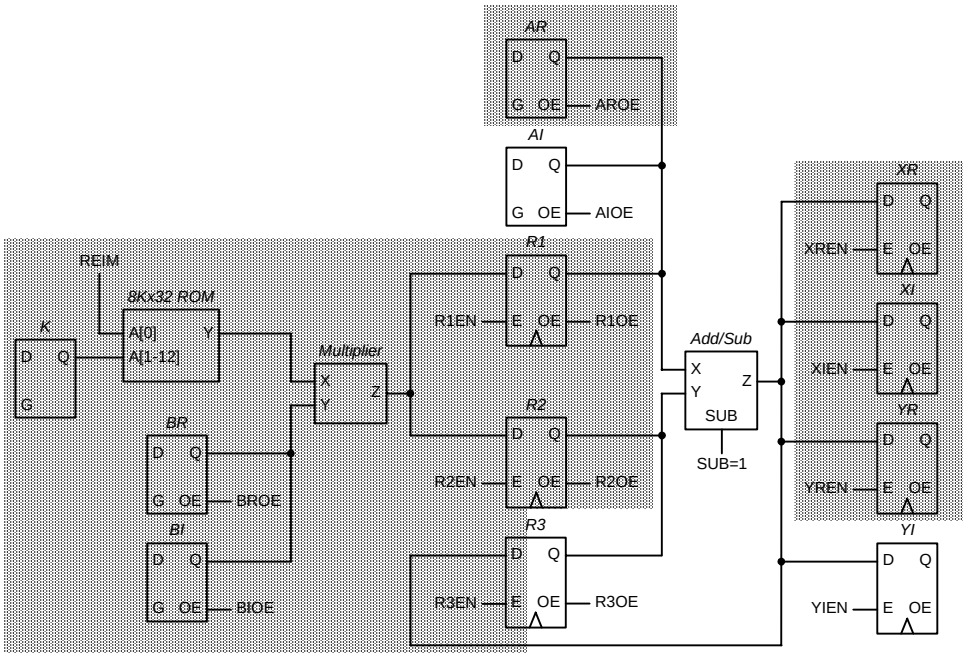
Fase 6

Fig 23 – La fase 6 consiste nel calcolo della variabile temporanea ZI.



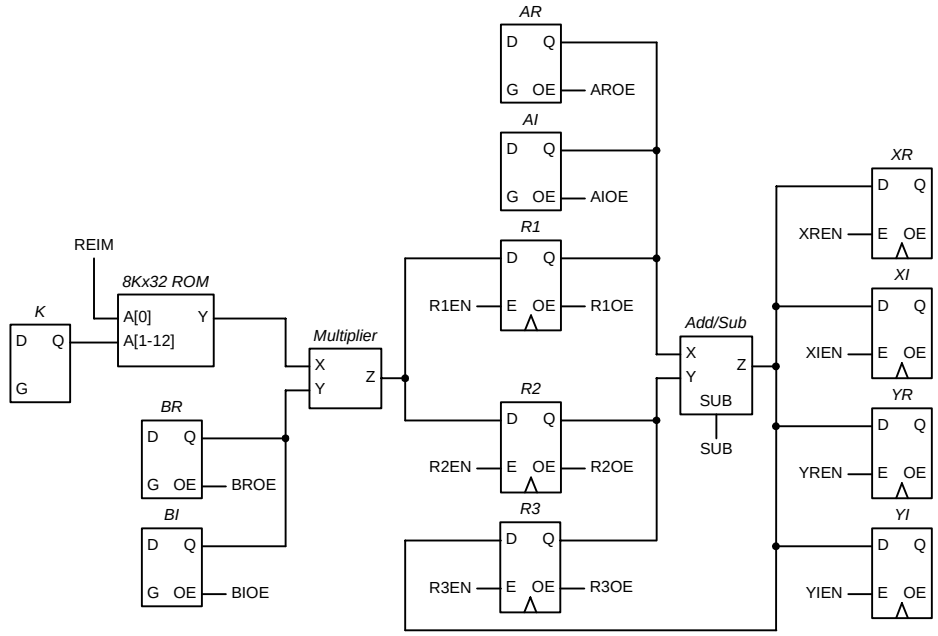
Fase 7

Fig. 24 – Nella fase 7 viene calcolato XI.



Fase 8

Fig. 25 – Nella fase 8, infine, viene calcolato l'ultimo risultato YI.



Fase	*REIM*	*SUB*	*BROE*	*BIOE*	*R1EN*	*R1OE*	*R2EN*	*R2OE*	*R3EN*	*R3OE*	*AROE*	*AIOE*	*XREN*	*YREN*	*XIEN*	*YIEN*
1	0	-	1	0	1	0	0	0	0	0	0	0	0	0	0	0
2	1	-	0	1	0	0	1	0	0	0	0	0	0	0	0	0
3	-	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	1	0	0	0	0	1	1	0	1	0	0	0
5	1	0	1	0	0	0	1	0	0	1	1	0	0	1	0	0
6	-	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0
7	-	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0
8	-	1	0	0	0	0	0	0	0	1	0	1	0	0	0	1

Fig. 26 – Schema finale del Sottosistema di Calcolo, con i segnali di controllo per i vari registri e i valori che essi devono assumere nel corso delle varie fasi.

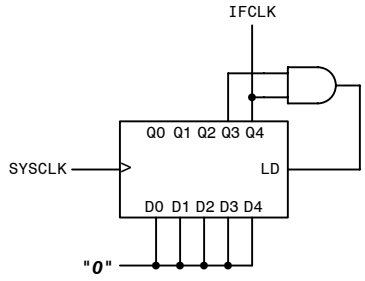


Fig. 27 – La scelta del clock di interfaccia è chiaramente dettata da considerazioni relative ai tempi di assestamento dei circuiti coinvolti. Il modulo più lento nello schema di Fig. 26 è evidentemente il moltiplicatore, il cui ritardo di propagazione può essere stimato in parecchie decine di nanosecondi, diciamo 70–80; a questo tempo occorre aggiungere i tempi di setup e di hold dei registri, per un totale (conservativo) inferiore a 100 nsec. Assumendo un System Clock a frequenza di 250 MHz, si deduce che il clock di interfaccia può essere da questo ricavato mediante divisione di frequenza per 25, a sua volta ottenuta mediante un contatore binario a 5 bit modificato per tale modulo di conteggio. IFCLK ha duty cycle diverso dal 50%, ma questo è del tutto ininfluente ai fini del corretto funzionamento dell'interfaccia. (È chiaramente possibile prevedere uno SCO con durata variabile delle singole fasi, in maniera da rendere più veloci le fasi che non coinvolgono il moltiplicatore; ma questo implicherebbe un livello di ottimizzazione al di là dei requisiti per il presente tema d'esame.)

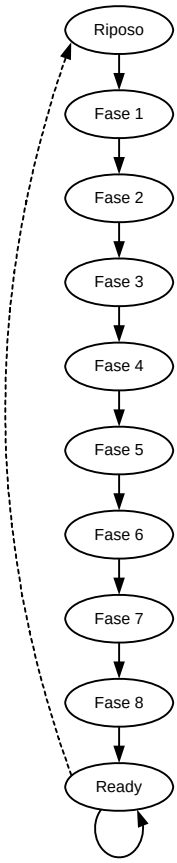


Fig. 28 – Diagramma di stato dello SCO. Al momento della scrittura del valore K da parte della CPU, la FSM viene forzata in modo asincrono nello stato di riposo (freccia tratteggiata); se quest'ultimo è codificato come `0000`, tale transizione speciale può essere realizzata utilizzando il Clear dei flip-flop delle variabili di stato. Esaurite le varie fasi del calcolo, la FSM entra in uno stato persistente (Ready) in cui si aspetta semplicemente l'arrivo di un nuovo segnale di Start. Nel passaggio dalla Fase 8 allo stato di Ready verrà segnalato alla CPU il completamento delle operazioni. Infine, il segnale di System Reset dovrà forzare la FSM nello stato di Ready, senza tuttavia segnalare alla CPU la fine delle operazioni.

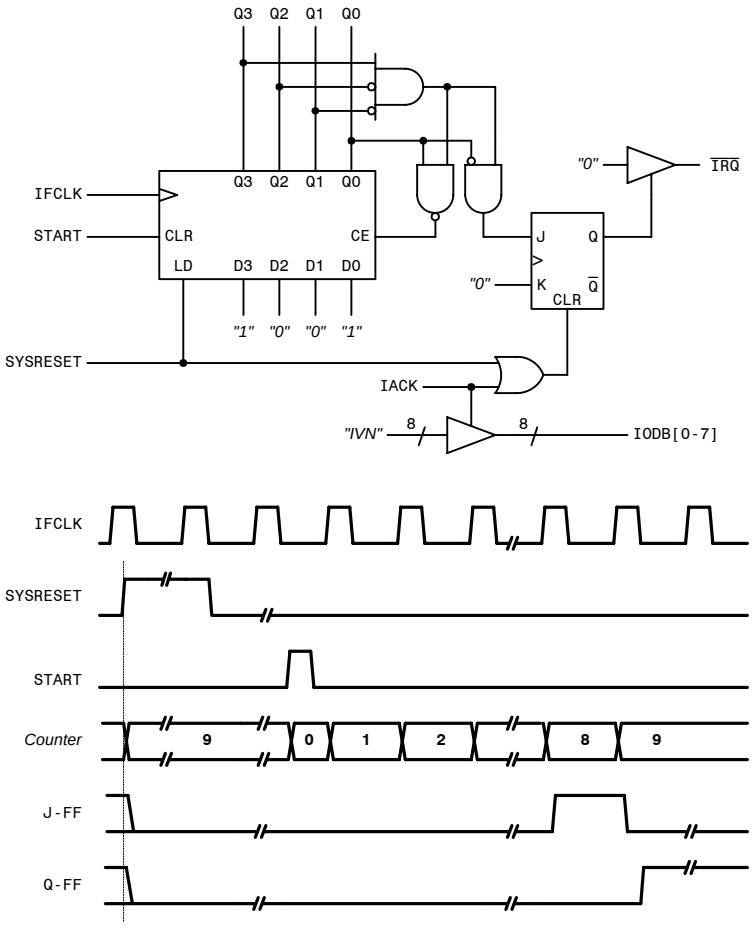


Fig. 29 – La struttura del diagramma di stato dello SCO suggerisce una realizzazione mediante contatore. Il segnale di Start viene applicato al Clear (asincrono) in modo da forzare lo stato di Riposo (codificato con `0000`), mentre il segnale di System Reset viene applicato al Load (asincrono) in modo da forzare lo stato di Ready (codificato con `1001`).

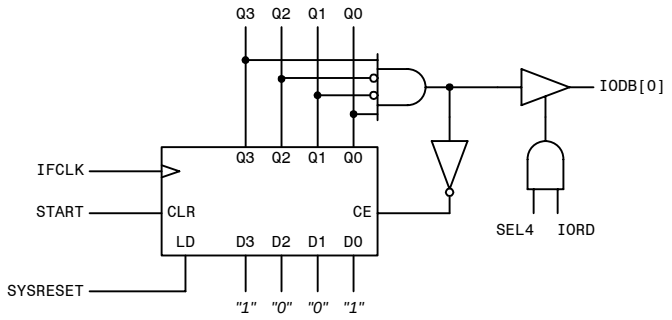
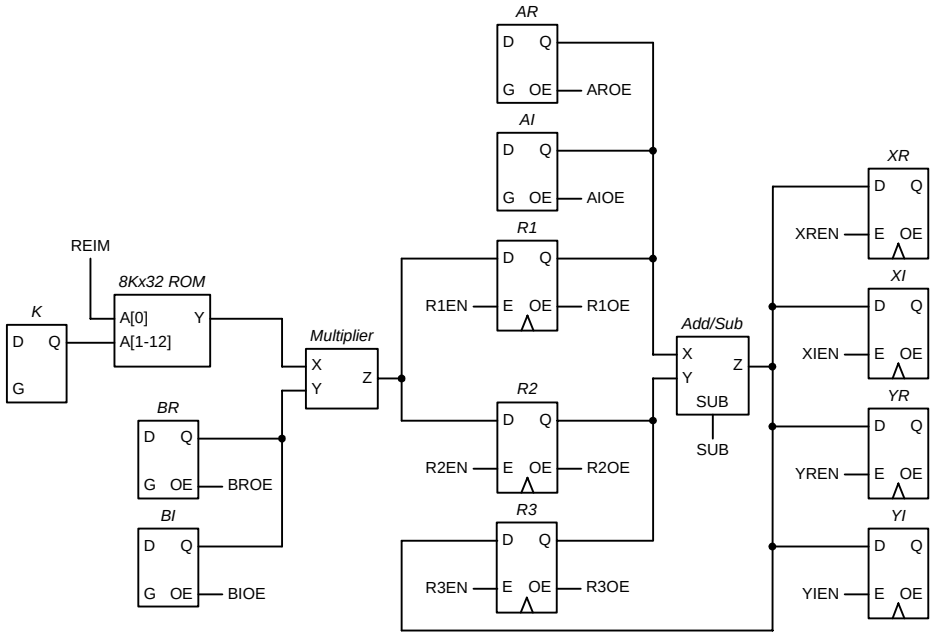


Fig 30 – In alternativa all'emissione di un interrupt, la notifica di fine delle operazioni può essere inviata attraverso un bit di una porta di input; la CPU adotterà in tal caso un protocollo di Busy Waiting.



Fase	*REIM*	*SUB*	*BROE*	*BIOE*	*R1EN*	*R1OE*	*R2EN*	*R2OE*	*R3EN*	*R3OE*	*AROE*	*AIOE*	*XREN*	*YREN*	*XIEN*	*YIEN*
1	0	-	1	0	1	0	0	0	0	0	0	0	0	0	0	0
2	1	-	0	1	0	0	1	0	0	0	0	0	0	0	0	0
3	-	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	1	0	0	0	0	1	1	0	1	0	0	0
5	1	0	1	0	0	0	1	0	0	1	1	0	0	1	0	0
6	-	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0
7	-	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0
8	-	1	0	0	0	0	0	0	0	1	0	1	0	0	0	1

Fig. 31 – Uscite da SCO (task) per tutti i vari stati, compresi quello di Riposo (`0000`) e quello di Ready (`1001`). È essenziale, in questi ultimi due stati, disabilitare i caricamenti per tutti i registri.

```
DEVBASE EQU ...          ; Indirizzo base di I/O della periferica

; --- Indirizzi di I/O delle porte di output

AR_PORT EQU DEVBASE+0    ; Porta per Real(A)
AI_PORT EQU DEVBASE+1    ; Porta per Imag(A)
BR_PORT EQU DEVBASE+2    ; Porta per Real(B)
BI_PORT EQU DEVBASE+3    ; Porta per Imag(B)
K_PORT EQU DEVBASE+4     ; Porta per K

; --- Indirizzi di I/O per le porte di input

XR_PORT EQU DEVBASE+0    ; Indirizzo di I/O della porta di input per Real(X)
XI_PORT EQU DEVBASE+1    ; Indirizzo di I/O della porta di input per Imag(X)
YR_PORT EQU DEVBASE+2    ; Indirizzo di I/O della porta di input per Real(Y)
YI_PORT EQU DEVBASE+3    ; Indirizzo di I/O della porta di input per Imag(Y)
RDY_PORT EQU DEVBASE+4   ; Indirizzo di I/O della porta di input per Ready
```

Fig. 32 – Dicihiarazione degli indirizzi delle porte di I/O.

```
;============================================================
; Subroutine di controllo (protocollo di Interrupt)
;
; Condizioni di input:
;    R1 contiene il valore K, 0 <= K < 4096
;    R2 contiene il puntatore a una tavola cosi' organizzata:
;
;           +------------------------+
;           |    Real(A) - 4 byte    |
;           |    Imag(A) - 4 byte    |
;           |    Real(B) - 4 byte    |
;           |    Imag(B) - 4 byte    |
;           +------------------------+
;
; Gli interrupt si assumono gia' abilitati. La routine di servizio
; interrupt notifichera' della sua avvenuta esecuzione scrivendo
; un valore diverso da zero nella locazione IFBFY_FLAG.

IFBFY_FLAG DB 0

IFBFY_CTRL:
   MOVB #0, (IFBFY_FLAG)      ; Azzera il flag
   OUTL (R2)+, #AR_PORT       ; Emette Real(A)
   OUTL (R2)+, #AI_PORT       ; Emette Imag(A)
   OUTL (R2)+, #BR_PORT       ; Emette Real(B)
   OUTL (R2)+, #BI_PORT       ; Emette Imag(B)
   OUTW R1, #K_PORT           ; Emette K ed avvia le operazioni
         ....                 ; (Eventuali altre operazioni)
AGAIN:                        ; Loop di attesa
   CMPB #0, (IFBFY_FLAG)      ; Test del flag
   JZ AGAIN                   ; Attesa se ISR non ancora eseguita
   RET
```

Fig. 33 – Subroutine di controllo dell'interfaccia nel caso di protocollo di interrupt. L'acquisizione dei risultati è delegata alla routine di servizio interrupt.

```
;============================================================
; Interrupt Service Routine
;
; I risultati prelevati da IFBFY vengono posti in una tavola
; cosi' organizzata:
;
;           +------------------------+
;           |    Real(X) – 4 byte    |
;           |    Imag(X) – 4 byte    |
;           |    Real(Y) – 4 byte    |
;           |    Imag(Y) – 4 byte    |
;           +------------------------+
;
; il cui puntatore e' contenuto nella locazione RESPTR.

RESPTR DL 0                  ; Puntatore alla tavola dei risultati
IFBFY_IVN  EQU ...           ; Interrupt Vector Number di IFBFY

DRIVER #IFBFY_IVN, IFBFY_ISR ; Associazione IVN / Intpt Service Routine

IFBFY_ISR:
   PUSH R1                   ; Salvataggio del contesto
   MOVL (RESPTR), R1         ; Legge il puntatore alla tavola
   INL #XR_PORT, (R1)+       ; Estrae Real(X)
   INL #XI_PORT, (R1)+       ; Estrae Imag(X)
   INL #YR_PORT, (R1)+       ; Estrae Real(Y)
   INL #YI_PORT, (R1)+       ; Estrae Imag(Y)
   MOVB #1, (IFBFY_FLAG)     ; Notifica di avvenuta esecuzione
   POP R1                    ; Ripristino del contesto
   RTI                       ; Ritorno da interrupt
```

Fig. 34 – Routine di servizio interrupt.

```
;============================================================
; Subroutine di controllo (protocollo Busy Waiting)
;
; Condizioni di input:
;    R1 contiene il valore K, 0 <= K < 4096
;    R2 contiene il puntatore a una tavola cosi' organizzata:
;
;           +------------------------+
;           |    Real(A) – 4 byte    |
;           |    Imag(A) – 4 byte    |
;           |    Real(B) – 4 byte    |
;           |    Imag(B) – 4 byte    |
;           +------------------------+
;
; I risultati prelevati da IFBFY vengono posti in una tavola
; cosi' organizzata:
;
;           +------------------------+
;           |    Real(X) – 4 byte    |
;           |    Imag(X) – 4 byte    |
;           |    Real(Y) – 4 byte    |
;           |    Imag(Y) – 4 byte    |
;           +------------------------+
;
; il cui puntatore e' contenuto nella locazione RESPTR.

IFBFY_CTRL:
   OUTL (R2)+, #AR_PORT       ; Emette Real(A)
   OUTL (R2)+, #AI_PORT       ; Emette Imag(A)
   OUTL (R2)+, #BR_PORT       ; Emette Real(B)
   OUTL (R2)+, #BI_PORT       ; Emette Imag(B)
   OUTW R1, #K_PORT           ; Emette K ed avvia le operazioni
   MOVL (RESPTR), R2          ; Legge il puntatore alla tavola
AGAIN:                        ; Loop di attesa
   INB #RDY_PORT, R1          ; Legge lo stato dell'interfaccia
   AND #01h, R1               ; Isola il bit 0 e lo testa
   JZ AGAIN                   ; Ripete il loop se IFBFY non pronta
   INL #XR_PORT, (R2)+        ; Estrae Real(X)
   INL #XI_PORT, (R2)+        ; Estrae Imag(X)
   INL #YR_PORT, (R2)+        ; Estrae Real(Y)
   INL #YI_PORT, (R2)+        ; Estrae Imag(Y)
   RET
```

Fig. 35 – Subroutine di controllo nel caso di protocollo Busy Waiting.