

2.1.9 Gestione di convertitore A/D con tecnica di busy waiting e di interrupt

Una periferica di input è il convertitore A/D, dispositivo che fornisce il valore numerico binario del campione di un segnale analogico, rilevato in un prefissato istante di tempo (*istante di campionamento*). Questo può essere gestito dal sistema di calcolo sia con la tecnica del busy waiting sia con la tecnica dell'interrupt secondo i meccanismi descritti in precedenza, anche se, solitamente è preferito questo secondo metodo. Prima di descrivere il protocollo di acquisizione si farà qualche richiamo su questo convertitore.

Convertitore A/D

Si tratta di un componente, realizzato in forma integrata, che riceve in ingresso un *segnale analogico* di una certa ampiezza, interno a un prefissato campo di variazione (*range*), e fornisce in uscita un numero a n-bit (tipicamente 8, 12, 14, 16 bit) ad essa proporzionale (a meno di un eventuale offset).

Lo schema di Fig.2.1.9.1 fornisce l'indicazione dei passaggi logici che subisce l'informazione dall'ingresso all'uscita per un convertitore ad 8 bit..

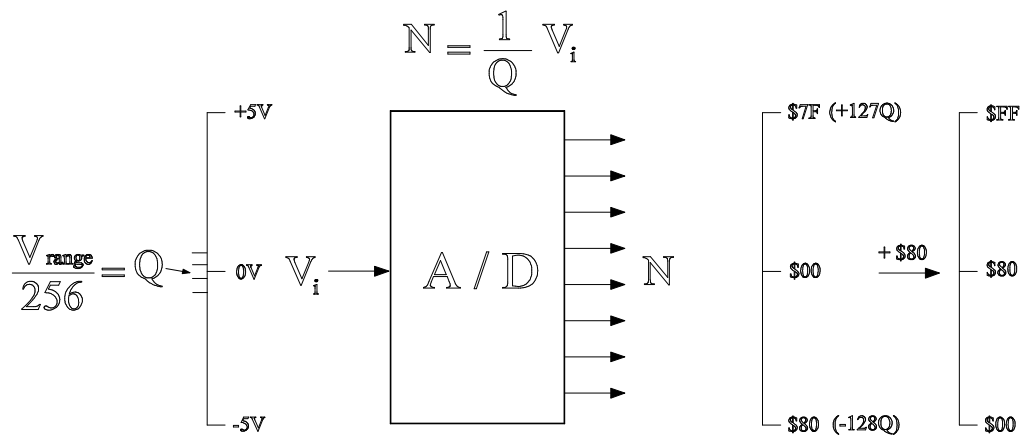


Fig.2.1.9.1 Legami logici tra segnali di ingresso e di uscita di un convertitore A/D a 8 bit con $V_{\text{range}}=10\text{V}$.

Si ricorda che un segnale analogico è definito in tutti gli istanti di tempo, a **tempo continuo**, e può assumere **tutti i valori** nel range entro il quale è definito. Per questo, motivo nel processo di conversione analogico digitale, si pongono due tipi di problemi:

- l'uscita del dispositivo di **Fig.2.1.9.1** è formata da numeri con $n=8$ cifre binarie. Come è noto si possono rappresentare con numeri di questa lunghezza solo $2^n=2^8=256$ valori diversi. Anche se l'ingresso è formato da infiniti valori bisogna accettare di rappresentarne in uscita solo un numero finito, pari a 2^n .
- il dispositivo A/D per convertire un certo valore V_i impiega un tempo finito, detto **tempo di conversione** t_{conv} , più o meno lungo in funzione della tecnologia con cui è costruito, che si aggiunge al **tempo di acquisizione** necessario al sistema che usa l'A/D per memorizzare il campione prodotto. Non è, quindi, possibile convertire tutti i valori dell'ingresso con continuità nel tempo.

Per questi motivi il segnale di ingresso, prima di essere trasformato in numero, deve subire due trasformazioni: il **campionamento** e la **quantizzazione**, come mostra sinteticamente la **Fig.2.1.9.2** per un A/D con uscita a 3 bit. L'intervallo di campionamento T_c deve essere scelto come frutto di un compromesso: sufficientemente grande da contenere il tempo di conversione t_{conv} più il tempo necessario ad acquisire l'uscita dell'A/D, e sufficientemente piccolo da garantire che tra un campione ed il consecutivo, il segnale di ingresso non abbia variazioni importanti che non vengono rilevate ([teorema del campionamento](#)). Ciò, ad esempio, è assicurato quando, unendo graficamente gli estremi dei campioni con linee rette (interpolazione lineare), il grafico del segnale, all'interno dell'intervallo di campionamento, si discosta di poco dai segmenti di retta.

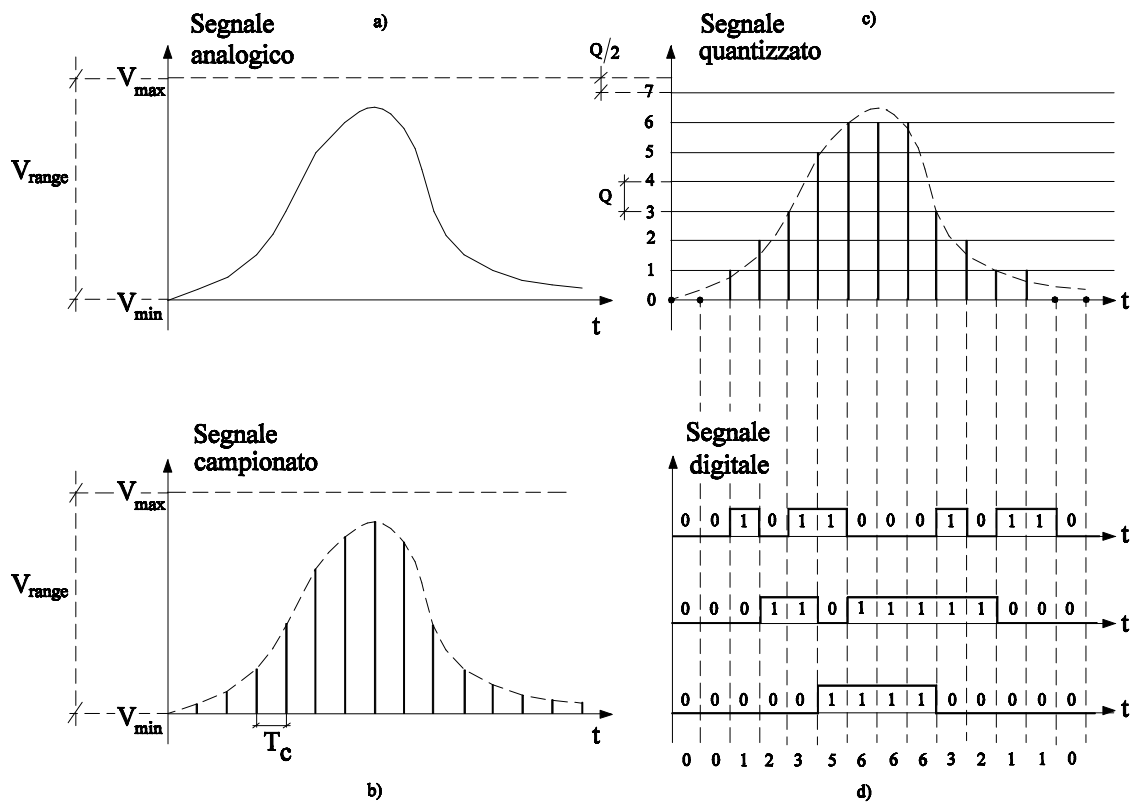


Fig.2.1.9.2 a) Segnale analogico b) Segnale campionato c) Segnale quantizzato d) Segnale digitale con 3 bit

La **Fig.2.1.9.2 c)** mostra che il processo di quantizzazione assegna gli otto valori numerici dell'uscita a otto *valori nominali* di tensione di ingresso, a iniziare dal valore 0 Volt (corrispondente al numero 0), intervallati di una quantità detta *precisione* o *quanto* data da:

$$Q = \frac{V_{range}}{2^3} = \frac{10V}{8digit} = 1,25 \frac{V}{digit} \quad (2.1.9.1)$$

Leggere in uscita il numero 3, in base a questa corrispondenza, significa dire che l'ingresso è $3 \cdot Q = 3 \cdot 1,25 = 3,75$ Volt. Si noti che questa corrispondenza segue il legame ingresso uscita indicato in **Fig.2.1.9.1**: $N = (1/Q) \cdot V_i$.

Il convertitore A/D è costruito in modo da assegnare ad ogni valore numerico dell'uscita non solo il valore nominale, ricavato dalla corrispondenza precedente, ma anche tutti i valori di tensione compresi nell'intervallo $\pm Q/2$ centrato attorno al valore nominale. Così, nel caso esemplificato, il numero 3 in uscita significa che la tensione di ingresso può assumere qualunque valore dell'intervallo centrato su $3 \cdot Q = 3,75$ Volt (valore nominale) e delimitato dagli estremi $3 \cdot Q - Q/2 = 3,75 - 0,625 = 3,125$ Volt e $3 \cdot Q + Q/2 = 3,75 + 0,625 = 4,375$ Volt.

Queste considerazioni consentono di rappresentare il legame ingresso-uscita di un A/D nel modo mostrato dal grafico di **Fig.2.1.9.3**.

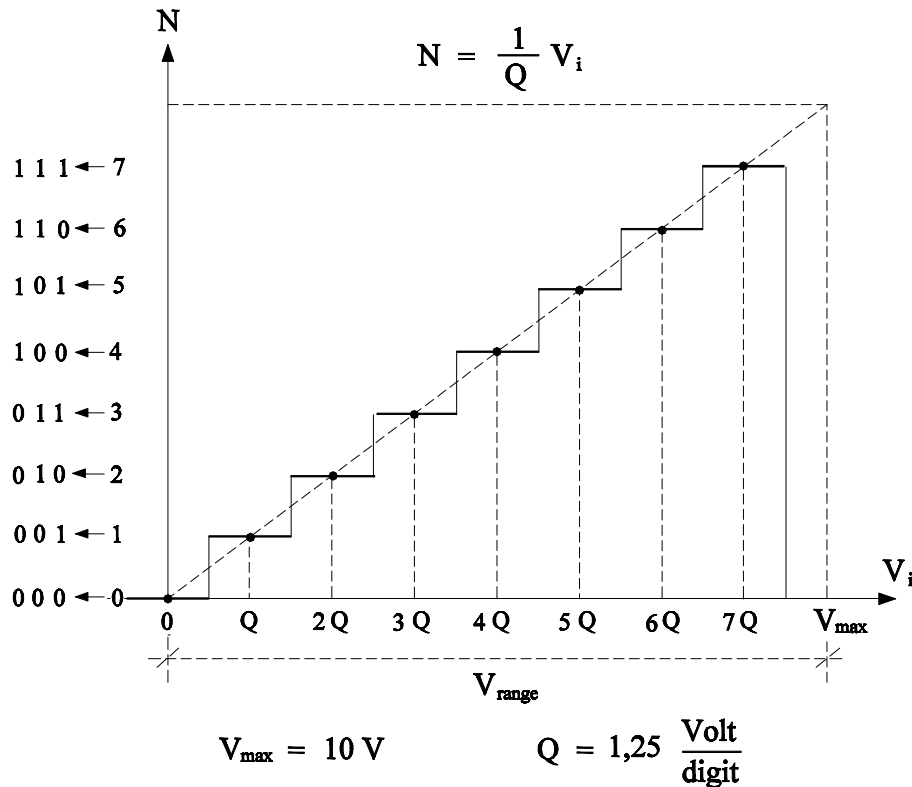


Fig.2.1.9.3 Legame ingresso uscita convertitore A/D a 3 bit.

In pratica si può dire che la lettura del numero N in uscita al convertitore consente di affermare che la tensione di ingresso ha un valore pari a $N \cdot Q$ (*valore nominale*) con un errore pari a $\pm Q/2$, intervallo in cui può trovarsi il *valore vero* dell'ingresso. Ovviamente minore è Q e minore è l'errore commesso. La diminuzione di Q richiede un aumento del numero dei bit di uscita dell' A/D, con aumento dei costi.

Con un convertitore a 8 bit di uscita (valore di largo uso), come nel caso di Fig.2.1.9.1, si ottiene un quanto pari a

$$Q = \frac{V_{\text{range}}}{2^8} = \frac{10\text{V}}{256\text{digit}} = 39 \frac{\text{mV}}{\text{digit}} \quad (2.1.9.2)$$

In questo l'errore commesso per ogni lettura può essere al massimo $\pm Q/2 = 20 \text{ mV/digit}$.

Nel caso di **Fig.2.1.9.1** il segnale di ingresso può avere anche valori negativi, per cui l'uscita rappresenta ancora, nel codice binario in complemento a due, il numero di volte che il quanto è contenuto nel campione presente all'ingresso, con precisione pari $\pm Q/2$. A questi valori è conveniente aggiungere $80 = 128$ per codificare il valore numerico in uscita in *codice binario eccesso 128*. Ciò semplifica le elaborazioni aritmetiche dato che non serve utilizzare l'aritmetica dei numeri con segno.

Il processo di conversione inizia quando l'A/D riceve un **segnale di start**, inviatogli dal calcolatore, e termina dopo un intervallo di tempo, detto **tempo di conversione**, più o meno lungo in funzione della tecnologia con cui è costruito, con l'invio da parte dell'A/D del **segnale di fine conversione**. Per il corretto funzionamento del convertitore A/D è necessario che **il segnale di ingresso V_i siano tenuto costante per tutto il tempo di conversione**.

Per questo scopo è utilizzato il circuito di **Sample and Hold** capace di mantenere il segnale di ingresso costante, per tutto il tempo di conversione (*Hold*), e di fornire il campione corretto, del segnale da convertire, quando riceve l'impulso di start conversion (*Sample*).

Sample and Hold

Lo schema di principio di questo dispositivo è rappresentato in **Fig.2.1.9.4**.

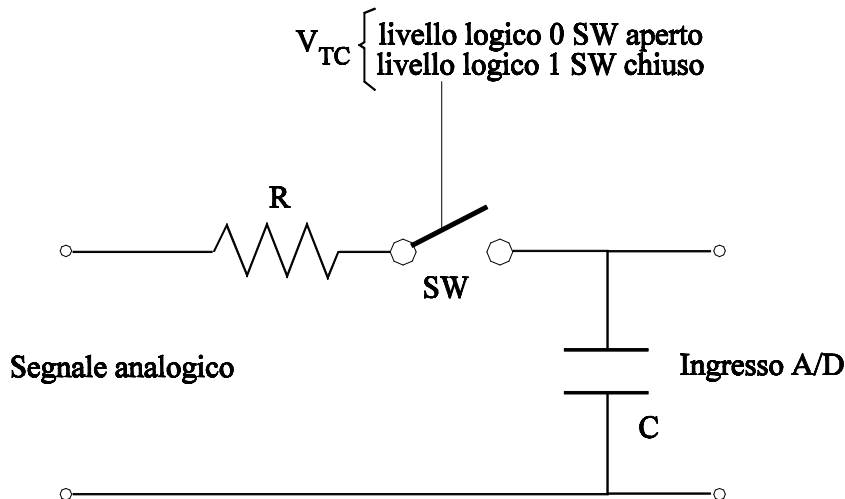


Fig.2.1.9.4 Schema di principio di un circuito *Sample and Hold*.

Quando il segnale di comando V_{TC} passa dal valore logico 1 al valore logico 0 l'interruttore si apre ed il condensatore carico conserva il valore del Segnale analogico presente in ingresso nell'istante in cui avviene questa transizione (*istante di campionamento*), nell'ipotesi che il circuito di ingresso dell'A/D presenti impedenza infinita. In questa fase, detta di **hold**, il segnale analogico continua la sua evoluzione nel tempo, mentre il condensatore funziona da memoria del campione presente in ingresso nell'istante in cui è avvenuta la transizione di V_{TC} da 1 a 0. La **Fig.2.1.9.5** illustra questo aspetto.

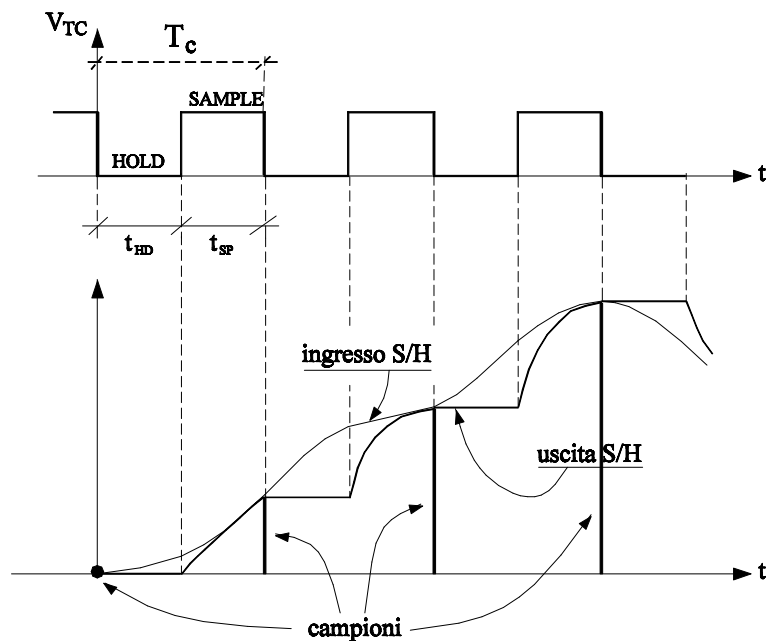


Fig.2.1.9.5 Legami ingresso uscita del S/H rispetto al segnale di comando.

Quando il segnale di comando ritorna a 1 l'interruttore SW si chiude ed il condensatore si carica al valore assunto dall'ingresso in un tempo che dipende dalla costante RC . In questa fase, detta di **sample** l'uscita dell'A/D deve raggiungere il

valore dell'ingresso prima del nuovo istante di campionamento. La somma dei due intervalli di tempo di *hold* e di *sample* è il periodo di campionamento T_c .

Con queste precisazioni un semplice dispositivo di acquisizione può essere quello rappresentato in **Fig.2.1.9.6**

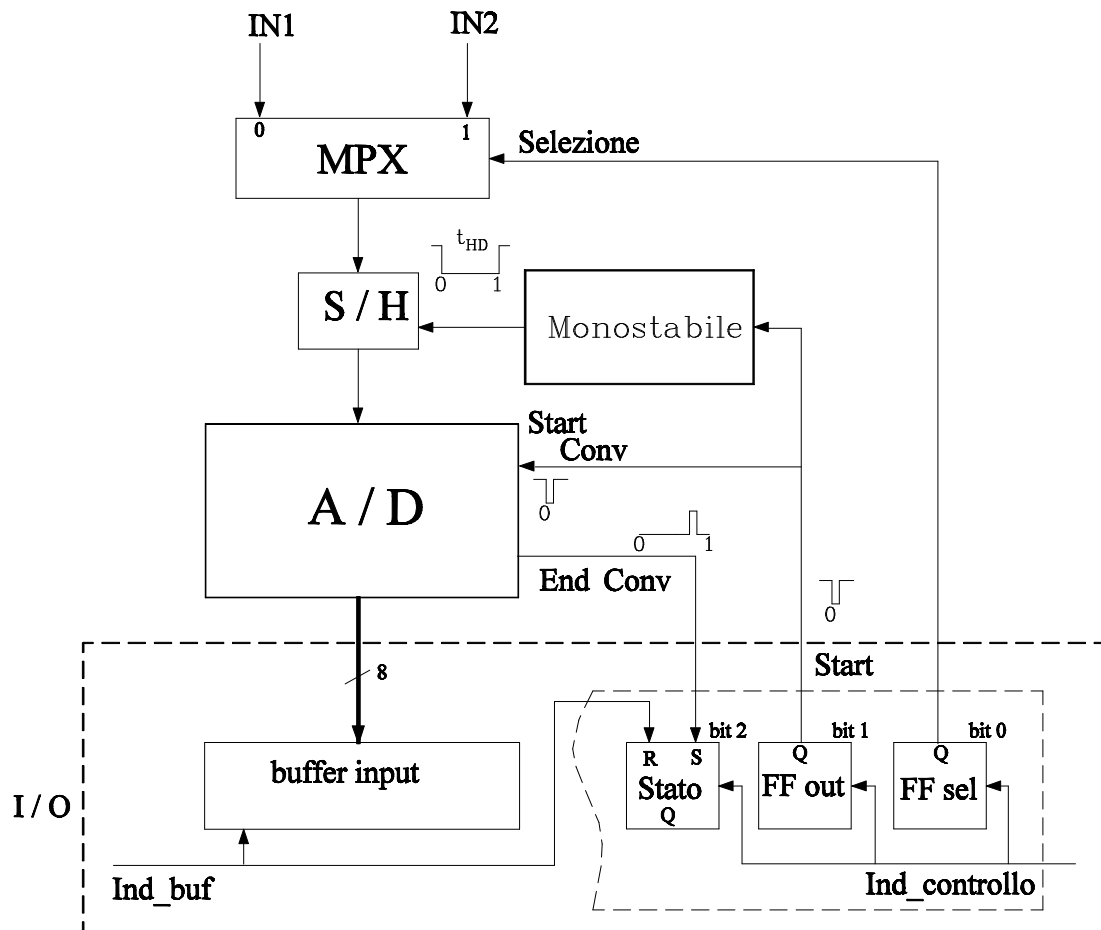


Fig.2.1.9.6 Periferica che consente l'acquisizione di due segnali analogici mediante A/D con busy waiting.

Nella suddetta figura è rappresentato un sistema che consente di acquisire due segnali analogici inviando l'opportuno segnale di selezione ad un multiplexer a 2 vie (0 seleziona **IN1**, 1 seleziona **IN2**). Seguendo i principi esposti in precedenza sulla gestione di periferiche di ingresso, il calcolatore invia il segnale di **Start** che stimola il convertitore a iniziare la conversione. L'A/D dopo il tempo di conversione (circa 20 μ s per un convertitore A/D ad approssimazioni successive) invia al calcolatore il segnale di fine conversione (**End**) ponendo a 1 il bit di stato. Il reset del bit di stato è prodotto quando viene letto il dato posto dall'A/D sul buffer di input, secondo la logica descritta per i dispositivi di input.

L'interfaccia verso il bus del calcolatore richiede oltre al buffer di input e al **FF stato** (FF: flip-flop), già descritto in precedenza, anche altri due FF che selezionano l'ingresso analogico (**FF sel**) e che danno il segnale di inizio conversione all'A/D (**FF out**). Per semplificare la logica interna al chip di interfaccia i tre FF possono essere considerati come un unico registro di indirizzo **Ind_controllo**. Si mantiene, comunque la logica descritta precedentemente, per cui: scrivendo con una istruzione all'indirizzo **Ind_Controllo** si scrive solo sui due flip flop **FF out** e **FF sel**, leggendo allo stesso indirizzo si legge solo il flip flop **FF stato**. Dal lato processore (omesso in figura) i collegamenti al bus avvengono secondo le modalità descritte precedentemente.

Con questa logica, l'acquisizione di campioni attraverso il canale **IN1**, in numero pari a quello indicato dalla variabile **CONT**, può avvenire con il seguente sottoprogramma eseguito dal processore descritto in precedenza.

;Programma chiamante		;Subroutine di acquisizione da A/D			
	•	1 ^A	Acquisisci:	MOVE	D0, -(SP) ;Salva D0
	•	2 ^A		MOVE	inizioCopia, D0
MOVE	#10, D0	3 ^A		MOVE	D0, memDato
MOVE	D0, COUNT	4 ^A		MOVE	Ind_buf, D0 ;Reset bit stato
JSR	Acquisisci	5 ^A		MOVE	#2, D0 ;bit0=0 sel. IN1 e
	•	6 ^A		MOVE	D0, Ind_controllo ;bit1=1 riposo
	•	7 ^A	StartAD:	MOVE	#0, D0 ;bit0=0 sel. IN1 e
	•	8 ^A		MOVE	D0, Ind_controllo ;bit1=0 attivo
MOVE	#100, D0	9 ^A		MOVE	#2, D0 ;bit1=1 riposo
MOVE	D0, COUNT	10 ^A		MOVE	D0, Ind_controllo ;bit2=0 conversione
JSR	Acquisisci	11 ^A	Aspetta:	MOVE	Ind_controllo, D0 ;non completata
	•	12 ^A		BEQ	Aspetta ;Fine conv., leggi dato
	•	13 ^A		MOVE	Ind_buf, D0 ;e memorizzato
	•	14 ^A	memDato:	MOVE	D0, Ind_array + ... ;Conta dato acquisito
<ultima istruzione programma>		15 ^A		MOVE	COUNT, D0
;Area di memoria contenente i		16 ^A		SUB	#1, D0
;campioni acquisiti		17 ^A		MOVE	D0, COUNT
Ind_array:	primo camp.	18 ^A		BEQ	Fine
Ind_array+1:	secondo camp.	19 ^A		MOVE	memDato, D0 ;Aggiungi uno all'ope-
Ind_array+2:	terzo campione	20 ^A		ADD	#1, D0 ;rando Ind_array + ...
Ind_array+3:	quarto camp.	21 ^A		MOVE	D0, memDato
Ind_array+4:	quinto camp.	22 ^A		JMP	StartAD
Ind_array+5:	sesto camp.	23 ^A	Fine:	MOVE	(SP)+, D0 ;Ripristina D0
	•	24 ^A		RTS	
	•		inizioCopia:	MOVE	D0, Ind_array ;Istr. copia primo dato

La prima istruzione della subroutine ACQUISISCI e l'ultima, prima della **RTS**, salva e rispettivamente, ripristina sullo stack il registro D0, in modo che al ritorno al programma chiamante D0 contiene gli stessi valori presenti prima della chiamata a sottoprogramma (*salvataggio del contesto*). La seconda e terza istruzione copiano all'indirizzo **memDato** l'istruzione memorizzata all'indirizzo **inizioCopia**. La quarta legge in **D0** un dato inutile, con il solo scopo di fare il reset del bit di stato. La quinta e la sesta istruzione pongono a riposo il segnale **start** (1 logico) e selezionano l'ingresso **IN1** del MUX. La settima e l'ottava pongono a zero il bit di Start mentre la nona e la decima lo riportano a uno completando l'impulso (1→0→1) necessario ad avviare l'A/D per la conversione e campione. Queste due istruzioni, verranno ripetute tante volte quanti sono i campioni da acquisire, a intervalli di tempo determinati dal tempo di conversione dell'A/D.

La undicesima e dodicesima istruzione aspettano, dopo ogni Start, che l'A/D abbia finito la conversione e ponga a 1 il bit di stato (bit 2 del *registro di controllo*). Finalmente la tredicesima istruzione legge il valore numerico del dato acquisito e lo pone nel buffer di memoria assegnato. Questa istruzione (di indirizzo **memDato**), copiata originalmente dalla locazione **inizioCopia**, ha l'operando che vale inizialmente **Ind_array**; successivamente questo indirizzo viene incrementato ciclicamente di uno (con le istruzioni dalla diciannovesima alla ventunesima) per consentire a questa istruzione di memorizzare i campioni in locazione consecutive.

Le quattro successive istruzioni (dalla quindicesima alla diciottesima) contano il dato acquisito e decidono se continuare o meno questo processo. Se **COUNT** si è azzerato, le acquisizioni sono finite e la diciottesima istruzione (**BEQ Fine**) porta l'esecuzione della subroutine alla ventitreesima e ventiquattresima istruzione che producono il ritorno al programma chiamante.

In questa descrizione si è implicitamente assunto che il *registro di controllo* sia un registro a otto bit (come il registro **Buffer_input**), dato che è unico l'indirizzo che li seleziona (**Ind_controllo**).

A fianco della subroutine è stato disegnato uno scorcio del programma chiamante che, con apposita istruzione di salto a sottoprogramma (**JSR**) richiede da software questo servizio ogni volta che deve fare acquisizioni. Prima di fare la chiamata questo programma pone nella variabile **COUNT** il numero dei campioni che devono essere convertiti (passaggio del parametro **COUNT**).

Ovviamente con questa tecnica (del *busy waiting*) la CPU è "fermata" ad attendere il completamento delle acquisizioni, prima di poter proseguire le sue elaborazioni sul programma principale.

Per ottimizzare l'attività della CPU è conveniente gestire questo processo con la tecnica dell'interrupt. Per questo scopo è necessario aggiungere al dispositivo di interfaccia la **bit di mask** (bit 3 del registro di controllo), secondo lo schema logico indicato in **Fig. 2.1.8.1**. Si ottiene il sistema di **Fig.2.1.9.7**.

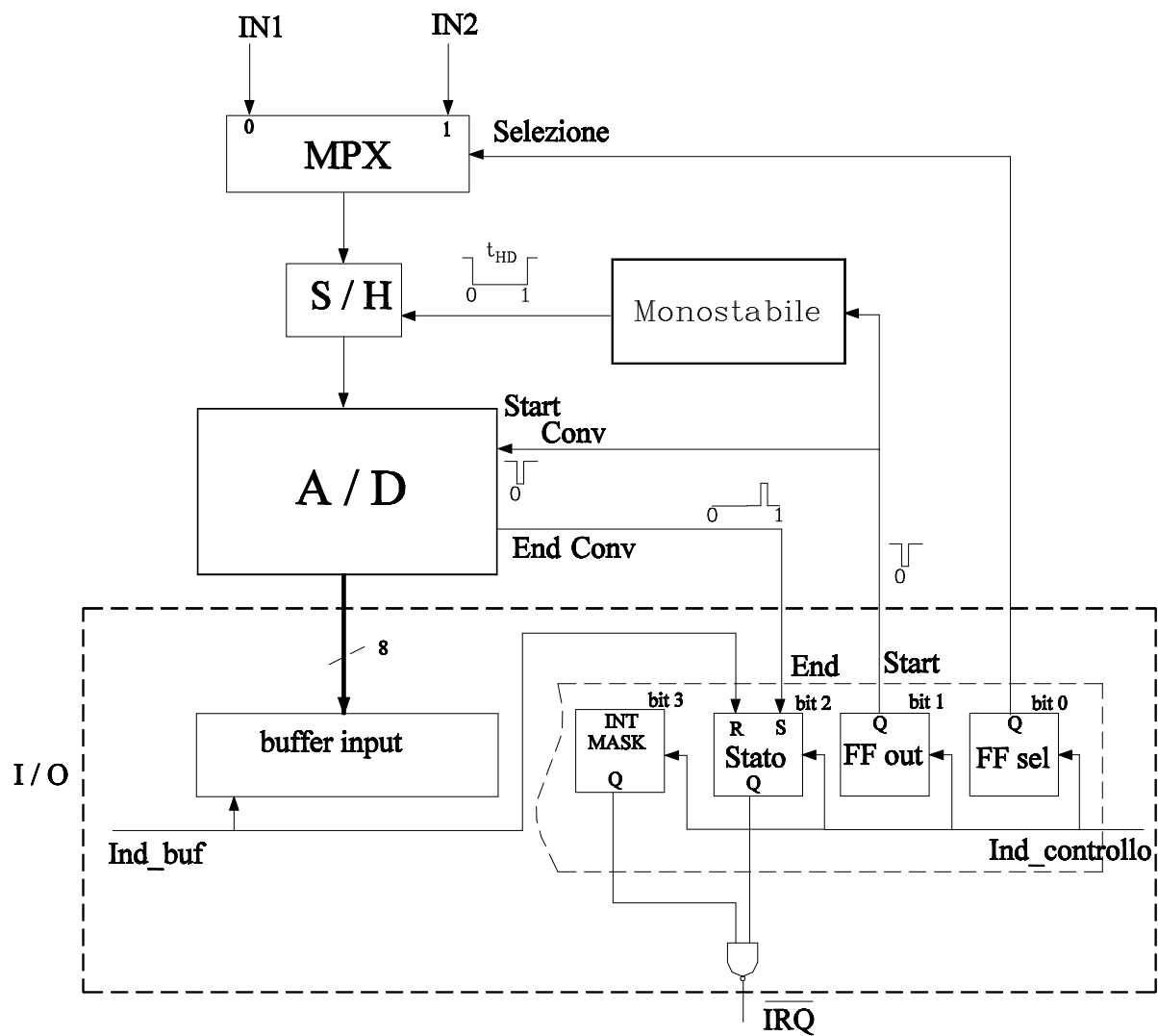


Fig.2.9.1.7 Periferica che consente l'acquisizione di due segnali analogici mediante A/D con tecnica dell'interrupt.

Il software che consente l'acquisizione dei campioni indicati da COUNT si modifica nel seguente modo:

;Programma chiamante		;Subroutine di acquisizione da A/D	
•	•	1 ^A	IRQAD: MOVE D0, -(SP) ;Salva D0
MOVE	#IRQAD, D0	2 ^A	MOVE Ind_buf, D0 ;Fine conv., leggi dati
MOVE	D0, 20	3 ^A	memData: MOVE D0, Ind_array + ... ;e memorizzalo
•	•	4 ^A	MOVE COUNT, D0 ;Conta dato acquisito
MOVE	#100, D0	5 ^A	SUB #1, D0
MOVE	D0, COUNT	6 ^A	MOVE D0, COUNT
MOVE	#2, D0	7 ^A	BEQ Fine
MOVE	D0, Ind_controllo	8 ^A	MOVE memData, D0 ;Aggiungi uno all'operando
MOVE	#0, D0	9 ^A	ADD #1, D0 ;raddoppia Ind_array + 1
MOVE	D0, Ind_controllo	10 ^A	MOVE D0, memData
MOVE	#2, D0	11 ^A	MOVE #0, D0 ;Avvia nuova acquisizione
MOVE	D0, Ind_controllo	12 ^A	MOVE D0, Ind_controllo
MOVE	Ind_buf, D0	13 ^A	MOVE #2, D0
MOVE	inizioCopia, D0	14 ^A	MOVE D0, Ind-controllo
MOVE	D0, memData	15 ^A	JMP Exit
MOVE	#10, D0	16 ^A	Fine: MOVE #2, D0 ;bit 3=0 (Int Mask=0)
MOVE	D0, Ind_controllo	17 ^A	MOVE D0, Ind-controllo ;disabilita interruzione
•	•	18 ^A	Exit: MOVE (SP)+, D0 ;Ripristina D0
•	•	19 ^A	RTI
;Array campioni acquisiti		inizioCopia: MOVE D0, Ind_array ;Istr. copia primo dato	
Ind_array:	primo camp.		
Ind_array+1:	secondo camp.		
Ind_array+2:	terzo campione		
Ind_array+3:	quarto camp.		
•	•		

In questo caso il programma principale, in fase di inizializzazione del sistema, pone l'indirizzo della prima istruzione della routine di risposta all'interrupt, all'indirizzo di memoria indicato all'interno della CPU nella locazione **vectADDR**. Dalla Fig.2.1.8.2 si vede che questo indirizzo è 20.

Successivamente, quando il programma principale decide acquisire 100 campioni, inizializza la variabile **COUNT** con questo valore. Produce l'impulso di **Start** come nel caso del **busy waiting**. Fa il reset del bit di stato con una lettura inutile del buffer di **I/O**, copia all'indirizzo **memData** l'istruzione che sarà utilizzata per salvare il primo campione e, infine, abilita le interruzioni ponendo a uno il bit **Int Mask** nell'interfaccia (vedi Fig.2.9.1.7). In questo modo l'A/D è abilitato a produrre interrupt.

Si osservi che le ultime due istruzioni di questo blocco "**MOVE #10, D0**" e "**MOVE D0, Ind_controllo**" pongono nel registro di controllo i seguenti bit: $10_{10}=1010_2$ che producono:

- bit 3=1 memorizzato in **Int Mask** per abilitare le interruzioni,
- bit 2=0 non viene scritto perché il bit di stato può solo essere letto,
- bit 1=1 memorizzato in **FF out** che conferma il bit di start a riposo
- bit 0=0 memorizzato in **FF sel** per selezione nel MUX il canale **IN1**.

Dopo aver eseguito queste istruzioni il programma principale si disinteressa di questo dispositivo periferico per eseguire altri tipi di attività.

Quando l'A/D ha terminato l'acquisizione del primo campione invia alla CPU il segnale di interrupt, che, riconosciuto (per semplicità si suppone che l'A/D sia l'unica sorgente di interrupt) ed accettato, manda in esecuzione, come descritto nel paragrafo 2.1.8, la routine **IRQAD**. La prima istruzione salva sullo Stack il registro **D0**, prima di utilizzarlo, dato che il dato in esso contenuto può essere importante per il programma interrotto. La seconda e terza istruzione legge, rispettivamente, il valore del campione dal buffer dell'interfaccia e lo memorizza nella matrice. Si ricorda che la seconda istruzione azzerava il bit di stato nell'I/O e quindi rimuove la causa dell'interrupt attuale. Le due successive istruzioni aggiornano il conteggio dei dati acquisiti e, nel caso siano meno di quelli indicati dalla variabile **COUNT**, produce l'incremento dell'operando dell'istruzione di indirizzo **memData** e lo **Start** di una nuova acquisizione generando l'impulso $1 \rightarrow 0 \rightarrow 1$ con le istruzioni dalla undicesima alla quattordicesima. Nel caso, invece, sia stato letto l'ultimo dato (**COUNT=0**), questa routine "spegne" il convertitore (con le istruzioni "**MOVE #2, D0**" e "**MOVE D0, Ind_controllo**") impedendogli di produrre ulteriori interruzioni (bit 3=0 memorizzato in **FF Int mask**). Il programma principale leggendo, quando lo ritiene opportuno, il **FF Int mask**, è in grado di determinare se l'acquisizione di tutti i campioni è stata completata o meno.

Le ultime due istruzioni ripristinano il contesto ponendo in **D0** il valore originario e riportando la CPU ad eseguire l'istruzione del programma principale successiva a quella colpita dall'interruzione.