Funzionamento e impostazioni modulo ADC su PIC24F

esempi con PIC24FJ64GB002

Libreria <adc.h> MPLAB C30

Desideriamo che l'open source e la cultura libera si diffondano anche in Italia, perchè siamo convinti che anche nel nostro paese esistano persone speciali capaci di cambiare il mondo. Al fine di incoraggiare coloro che lavorano e producono in questa direzione, evitate di copiare il materiale altrui e piuttosto, se siete davvero interessati, collaborate con gli autori originali. Così facendo eviterete anche di fare brutte figure e di essere esclusi dalla comunità seria: quella che lavora con passione, dedizione, metodo, originalità e collaborazione attiva con appassionati e specialisti del settore.

Se volete utilizzare questo lavoro o parti di esso per la ripubblicazione, contattate l'autore.

Dedicato ai soliti ignoti, a quelli che remano contro la cultura e contro il nostro amato paese. Ricordate: essere furbi non equivale ad essere intelligenti.

Il modulo ADC sui PIC24F

PIC24FJ64GB002

Su settorezero.com abbiamo analizzato, tempo fa, il funzionamento del modulo A/D relativamente ai PIC serie 12/16:

http://www.settorezero.com/wordpress/corso-programmazione-picmicro-in-c-lezione-11-parte-1-il-convertitore-analogico-digitale-principi-di-funzionamento/

In linea di massima i principi teorici di funzionamento sono molto simili, per cui una lettura dell'articolo su citato vi farà sicuramente partire avvantaggiati. La messa in opera e l'utilizzo del modulo ADC sui pic24 è però un'operazione molto più articolata in quanto sono presenti più blocchi funzionali che, d'altra parte, rendono il modulo ADC dei PIC a 16 bit ancor più versatile nell'utilizzo.

Per la trattazione prenderò in esempio il modulo ADC del **PIC24FJ64GB002** che è forse uno dei più semplici. Altri PIC a 16bit, ivi compresi le serie PIC24F, PIC24H, dsPIC30 e dsPIC33, potrebbero avere alcune impostazioni aggiuntive o differenti, per cui in ogni caso fate sempre riferimento al datasheet ed al reference manual del pic in vostro possesso.

Terminologia

Prima di cominciare riassumo qui i termini più utilizzati in maniera da tenerli sottomano qualora la memoria facesse cilecca o semplicemente per fare un "refresh" :

Fosc : Frequenza Oscillatore. E' la frequenza alla quale opera il picmicro. Non è necessariamente la frequenza del quarzo in quanto potrebbe essere attivo un PLL che la moltiplica. Con Fosc si intende quindi la frequenza *finale* alla quale il picmicro opera.

Fcy : E' la frequenza di un ciclo di istruzioni, ovvero i colpi di clock necessari affinchè un'istruzione *semplice* venga eseguita. Per i PIC24F, PIC24H e dsPIC33 Fcy vale Fosc/2. Per i dsPIC30 Fcy vale Fosc/4.

Tcy : Tempo necessario affinchè venga eseguito un ciclo istruzioni. Vale 1/Fcy.

Supponiamo di avere il nostro PIC24FJ64GB002 con un quarzo da 8MHz ed il PLL attivo. La frequenza operativa finale vale 32MHz. Per cui:

Fosc=32MHz, Fcy=16MHz, Tcy=0.0625µS (62.5nS)

Campionamento : fase durante la quale il modulo ADC si connette al pin dal quale prelevare il segnale da analizzare. Durante questa fase il segnale analogico viene caricato in un condensatore, chiamato C_{HOLD}.

Conversione : è la fase successiva al campionamento. Il modulo si disconnette dalla sorgente esterna e collega C_{HOLD} al modulo che esegue la conversione.

Tad : tempo di acquisizione analogico/digitale. E' la somma del tempo di campionamento + tempo di conversione.

Modulo Sample and Hold (o modulo S/H) : è la parte del modulo ADC che si occupa della sola fase di campionamento ed include uno stadio di amplificazione.

SAR : Successive Approximations Register, è il tipo di convertitore analogico/digitale presente sui pic, che eseguono una conversione basandosi sul *metodo delle approssimazioni successive*. Per il suo funzionamento si avvale di un DAC e di un comparatore. Maggiori informazioni su wikipedia:
http://en.wikipedia.org/wiki/Successive_Approximation_ADC

Vdd : Tensione di alimentazione (positiva) del picmicro. Sui PIC24F, PIC24H e dsPIC33 vale massimo 3.3V, i soli dsPIC30 possono essere alimentati anche a massimo 5V.

Vss : Riferimento per la tensione di alimentazione, vale 0V ed è spesso identificato anche con GND

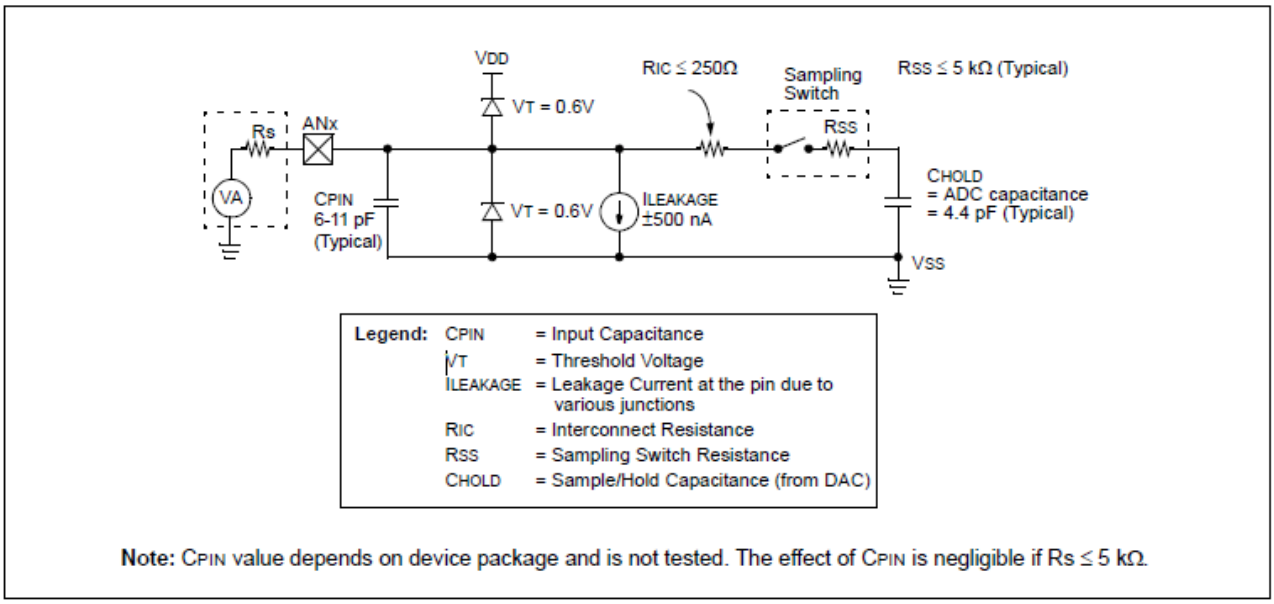
Tensione di riferimento Bandgap (o *Bandgap Voltage Reference*) : è una tensione di riferimento precisa, indipendente dalla temperatura, che vale 1.2V. Maggiori informazioni su wikipedia:
http://en.wikipedia.org/wiki/Bandgap_voltage_reference

Vddcore : Tensione di alimentazione del *core*, vale 2.5V. E' una tensione che può essere generata dal picmicro internamente qualora il regolatore interno venga abilitato (tramite la selezione eseguita sui pin ENVREG o DISVREG a seconda del modello) oppure fornita dall'esterno se il regolatore viene disabilitato. Per maggiori informazioni fare riferimento al paragrafo *On-Chip voltage regulator* del pic in vostro possesso.

NOTE

Modello stadio di ingresso modulo ADC

FIGURE 22-2: 10-BIT A/D CONVERTER ANALOG INPUT MODEL



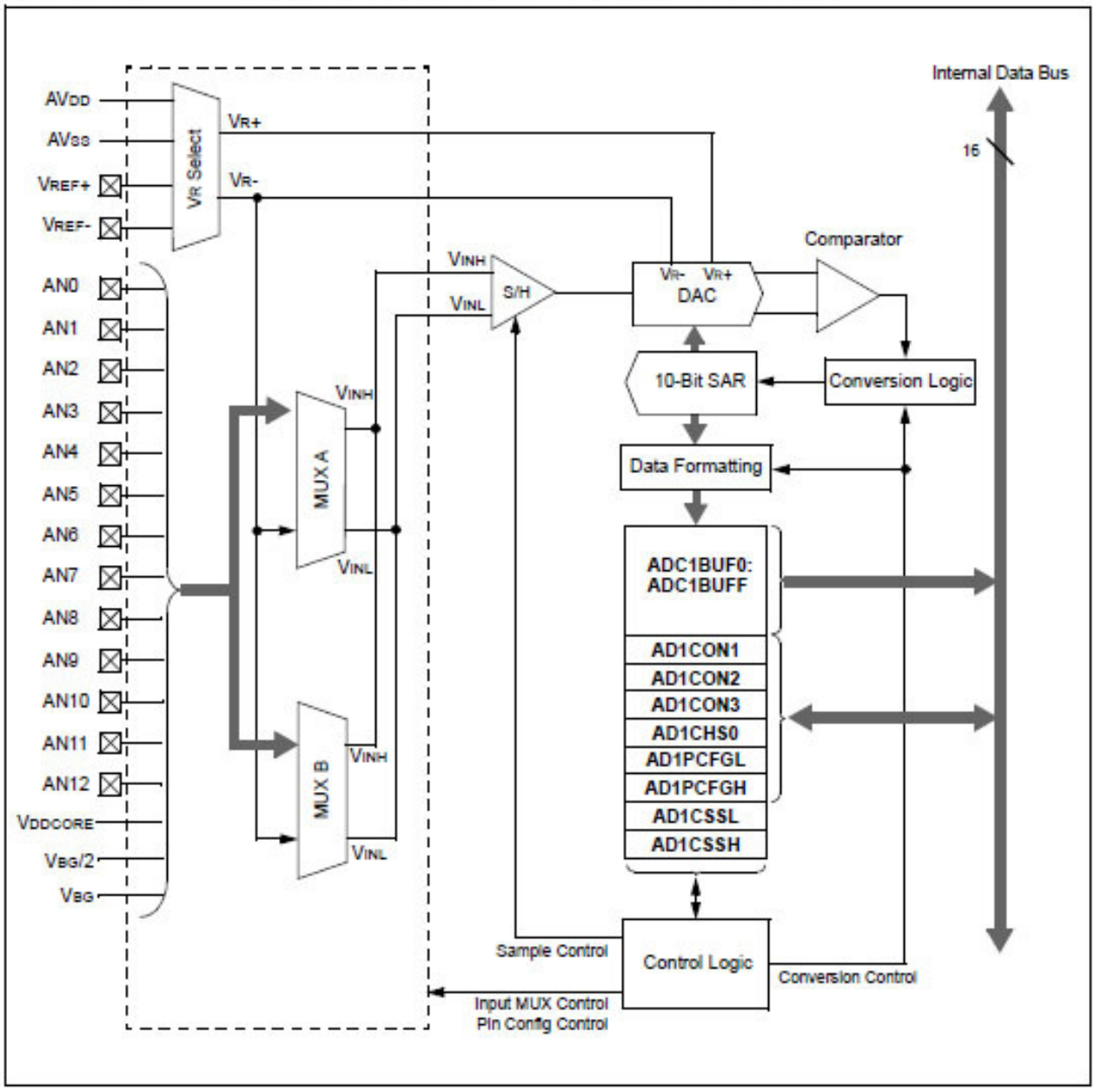
La sorgente di segnale esterna, rappresentata dal simbolo VA (la quale ha anch'essa una propria resistenza interna, R_S) viene collegata al pin di acquisizione Analogica (ANx) del nostro picmicro.

All'interno del pic abbiamo una capacità in ingresso: C_{PIN} ,trascurabile se Rs vale meno di 5KΩ. Seguono due diodi zener che servono a creare una soglia di sicurezza. Abbiamo quindi una perdita di corrente ($I_{LEAKAGE}$) dovuta alle varie giunzioni. Segue una resistenza di interconnessione (R_{IC}) tra la parte esterna del modulo e quella di campionamento vera e propria. Abbiamo quindi lo switch di campionamento (il quale ha anch'esso una sua resistenza, R_{SS}, molto bassa) che serve a collegare la sorgente di segnale da acquisire al condensatore di campionamento (C_{HOLD}).

Durante la fase di campionamento, lo switch viene chiuso e C_{HOLD} si carica allo stesso livello di tensione della sorgente da acquisire. Finito il campionamento lo switch si apre e il livello di tensione di C_{HOLD} viene trasferito al modulo successivo che si occupa di eseguire la conversione.

Diagramma a blocchi del modulo ADC

FIGURE 22-1: 10-BIT HIGH-SPEED A/D CONVERTER BLOCK DIAGRAM



Nel modulo ADC del picmicro preso ad esempio è presente un primo blocco, identificato dalla sigla VR Select, che permette di impostare le tensioni di riferimento con cui *confrontare* i valori analogici da leggere (V_{R+} e V_{R-}). Comunemente i due valori di riferimento sono i *riferimenti interni* di tensione Vss e Vdd.

Nulla vieta di impostare dei riferimenti *esterni* da applicare ai pin con i contrassegni V_{REF+} e V_{REF-} o di utilizzare un riferimento interno ed uno esterno.

Abbiamo quindi due Multiplexer: MUX A e MUX B che hanno il compito di eseguire la selezione del canale da inviare al modulo S/H. I due multiplexer possono lavorare in maniera alternata (si attiva prima l'uno e poi l'altro) oppure può lavorare unicamente il MUX A. Il MUX A, inoltre, ha la possibilità di eseguire in automatico la scansione dei canali impostati.

> Nota
> *Sui pic16 queste caratteristiche non esistono: è presente un solo multiplexer, e per giunta l'ingresso da cui prelevare il segnale deve essere impostato a mano ogni volta. Qui abbiamo invece la possibilità di compilare una sorta di "lista" nella quale specificare da quali pin eseguire la conversione e il MUX A esegue la scansione in automatico ad intervalli prestabiliti.*

Vediamo un'altra caratteristica "nuova": ogni multiplexer ha due ingressi: uno positivo (V_{INH}) e uno negativo (V_{INL}). L'ingresso positivo è quello che accetta il canale da inviare all'ingresso *non invertente* dell'amplificatore del modulo S/H. L'ingresso negativo, invece, normalmente viene connesso al riferimento negativo di tensione ma in alternativa (anche se nel diagramma non è indicato) può essere connesso ad AN1.

I due ingressi, in pratica, vengono inviati all'operazionale del modulo S/H e quello negativo viene sottratto dal positivo, per tale motivo generalmente l'ingresso negativo viene posto a 0V (V_{SS}) per avere una lettura *assoluta*. Il modulo S/H esegue quindi il campionamento come abbiamo visto. Terminato il campionamento, la corrente viene infine trasferita all' ADC che esegue una conversione a 10bit.

I multiplexer possono accettare in ingresso, oltre al segnale proveniente dai pin esterni, anche delle tensioni interne che sono la tensione di bandgap (VBG), la tensione di bandgap diviso 2 (VBG/2) e la tensione di alimentazione del core (Vddcore) in maniera da avere dei confronti più precisi e specifici.

Il risultato della conversione A/D viene immagazzinato in un buffer costituito da 16 words (ovvero 16 "blocchi", o celle di memoria che dir si voglia) da 16 bytes ognuno. Abbiamo cioè 16 registri che hanno i nomi ADC1BUF0, ADC1BUF1, ADC1BUF2 ... fino ad ADC1BUFF (l'ultimo numero è in pratica il numero del registro espresso in formato esadecimale).

La presenza di 16 celle di memoria è necessaria per la funzione di scansione automatica dei canali, ma non solo. Immaginiamo di eseguire la scansione automatica di 16 ingressi A/D. Il MUX A può eseguirla in automatico e il risultato della conversione su ogni pin viene memorizzato nel proprio buffer (la tensione sul pin AN0 viene memorizzata nella "cella" zero, ovvero ADC1BUF0, e così via fino alla sedicesima cella ovvero la numero 15, ADC1BUFF, nella quale sarà memorizzato il risultato della conversione su AN15).

Il buffer inoltre può essere trattato come un unico buffer da 16 words o come due buffers separati da 8 words ciascuno, a cosa serve questa caratteristica? Supponiamo ad

esempio di eseguire l'acquisizione di un solo canale analogico; l'avere due buffer separati permette di memorizzare il risultato della prima conversione in una cella e il risultato della seconda conversione in un'altra cella. In questo modo abbiamo due risultati separati per volta. Questa caratteristica è ancora più interessante quando si sfrutta il fatto che l'interrupt su fine conversione può essere impostato per scattare ad ogni singola conversione, ma anche dopo 2 conversioni... dopo 3... con un mumero di conversioni fino a 16 (fino a 32 su altri modelli di picmicro).

> Nota
> *Farete caso al fatto che i registri dell'ADC hanno tutti un nome che inizia per ADC1. Questo è stato fatto per evidenziare che su alcuni picmicro c'è più di un modulo ADC, in questo caso è possibile utilizzare lo stesso codice anche sui pic con più di un modulo senza correre il rischio di incorrere in errori che si avrebbero se, ad esempio, sui pic con un solo modulo sarebbero stati usati dei nomi inizianti semplicemente con "ADC".*

Per poter eseguire tutte queste funzioni è implicito che il modulo ADC dei PIC24F si avvale di parecchi registri, i quali spesso non sono molto intuitivi nelle impostazioni.

Insieme al C30 viene fornita una libreria che in qualche modo aiuta nei settaggi, anche se in realtà, una volta imparato lo scopo dei bit nei registri, qualcuno potrebbe obiettare che sia quantomeno inutile utilizzare tale libreria e che è più pratico settare i registri senza ricorrere a funzioni. Ognuno fa la scelta che ritiene più opportuna! In ogni caso affianco alla descrizione dei parametri da settare nelle funzioni ho anche riportato i corrispondenti bit su cui si va ad agire nei vari registri, per cui la trattazione è abbastanza completa.

La libreria viene inclusa in automatico ponendo ad inizio programma la direttiva include apposita:

```
#include <adc.h>
```

> Nota
> *Tale file header in realtà include le librerie necessarie in base al pic utilizzato nel progetto. Per cui per i dispositivi aventi un convertitore A/D a 12bit, verranno caricate le che servono a gestire un ADC a 12bit e così via. I nomi delle funzioni utilizzate dalle librerie hanno difatti il suffisso ADC10 o ADC12 in base al tipo di convertitore montato sul picmicro. Le "nostre" funzioni, quindi, avranno tutte un nome che inizia per "ADC10".*

La documentazione ufficiale di tali librerie (e delle altre!) si trova, come sempre, nella cartella del compilatore (`Programmi\Microchip\MPLAB C30\docs`). In particolare la documentazione che illustra il funzionamento di queste librerie aggiuntive è riportato nella sottocartella `\periph_lib`.

Nella versione attuale della documentazione, purtroppo, non è presente la spiegazione per l'utilizzo delle librerie ADC10 per i PIC24 e dsPIC33 ma unicamente quella per i dsPic30 (`dsPIC30F_ADC10_Library_Help.htm`).
In ogni caso le librerie sono molto simili, cambiano solo alcuni nomi mnemnonici e alcune impostazioni sono mancanti o differenti.

In questa trattazione, come già detto, farò riferimento al PIC24FJ64GB002, per cui alcuni nomi mnemnonici è probabile che causino errori con PIC diversi da questo, quindi in caso di dubbi andate a spulciarvi il sorgente della libreria (`support\peripheral_24F` per i pic 24F oppure `support\peripheral_30F_24H_33F` per gli altri pic a 16bit). In questi files sono difatti contenute le librerie base con l'elenco di tutti i nomi mnemonici sfruttati dalle funzioni.

Impostazione modulo ADC

La prima cosa da fare è mettere in funzione ed impostare il modulo ADC tramite l'apposita funzione `OpenADC10`:

```
void OpenADC10 (
unsigned int config1,
unsigned int config2,
unsigned int config3,
unsigned int configport,
unsigned int configscan);
```

Dobbiamo quindi assegnare un valore ai 5 parametri da passare alla funzione. Ogni parametro è la combinazione di uno o più valori da unire tra loro mediante un AND. Vediamo l'elenco dei parametri, alla fine sarà tutto più chiaro con un esempio.

config1 serve ad impostare il registro **AD1CON1**

- Accensione modulo
  Corrispondenza nel registro: bit 15 (*ADON*)
  Valori possibili:
  `ADC_MODULE_ON` modulo acceso
  `ADC_MODULE_OFF` modulo spento

- Funzionalità del modulo in modalità IDLE
  Corrispondenza nel registro: bit 13 (*ADSIDL*)
  Valori possibili:
  `ADC_IDLE_CONTINUE` il modulo continua a funzionare anche in idle
  `ADC_IDLE_STOP` il modulo si ferma se il pic va in idle

- Formato del risultato
  Corrispondenza nel registro: bits 9÷8 (*FORM<1:0>*)
  Valori possibili:

ADC_FORMAT_SIGN_FRACT	frazionale con segno (-0.500 ÷ 0.499)
ADC_FORMAT_FRACT	frazionale senza segno (0.000 ÷ 0.999)
ADC_FORMAT_SIGN_INT	integer con segno (-512 ÷ 511)
ADC_FORMAT_INTG	integer senza segno (0 ÷ 1023)

Nota
Già potete notare un'altra differenza con il modulo ADC dei pic a 8bit: non abbiamo più l'impostazione "giustificazione a destra" e "giustificazione a sinistra" del risultato.

Nelle tabelle seguenti è possibile capire come viene presentato il risultato della conversione utilizzando i vari formati:

Figure 17-10: A/D Output Data Formats

RAM Contents:							d09	d08	d07	d06	d05	d04	d03	d02	d01	d00
Read to Bus:																
Integer	0	0	0	0	0	0	d09	d08	d07	d06	d05	d04	d03	d02	d01	d00
Signed Integer	$\overline{d09}$	$\overline{d09}$	$\overline{d09}$	$\overline{d09}$	$\overline{d09}$	$\overline{d09}$	$\overline{d09}$	d08	d07	d06	d05	d04	d03	d02	d01	d00
Fractional (1.15)	d09	d08	d07	d06	d05	d04	d03	d02	d01	d00	0	0	0	0	0	0
Signed Fractional (1.15)	$\overline{d09}$	d08	d07	d06	d05	d04	d03	d02	d01	d00	0	0	0	0	0	0

Table 17-2: Numerical Equivalents of Various Result Codes: Integer Formats

VIN/VREF	10-Bit Output Code	16-Bit Integer Format/ Equivalent Decimal Value		16-Bit Signed Integer Format/ Equivalent Decimal Value	
1023/1024	11 1111 1111	0000 0011 1111 1111	1023	0000 0001 1111 1111	511
1022/1024	11 1111 1110	0000 0011 1111 1110	1022	0000 0001 1111 1110	510
		•••			
513/1024	10 0000 0001	0000 0010 0000 0001	513	0000 0000 0000 0001	1
512/1024	10 0000 0000	0000 0010 0000 0000	512	0000 0000 0000 0000	0
511/1024	01 1111 1111	0000 0001 1111 1111	511	1111 1111 1111 1111	-1
		•••			
1/1024	00 0000 0001	0000 0000 0000 0001	1	1111 1110 0000 0001	-511
0/1024	00 0000 0000	0000 0000 0000 0000	0	1111 1110 0000 0000	-512

Table 17-3: Numerical Equivalents of Various Result Codes: Fractional Formats

VIN/VREF	10-Bit Output Code	16-Bit Fractional Format/ Equivalent Decimal Value		16-Bit Signed Fractional Format/ Equivalent Decimal Value	
1023/1024	11 1111 1111	1111 1111 1100 0000	0.999	0111 1111 1100 0000	0.499
1022/1024	11 1111 1110	1111 1111 1000 0000	0.998	0111 1111 1000 0000	0.498
		•••			
513/1024	10 0000 0001	1000 0000 0100 0000	0.501	0000 0000 0100 0000	0.001
512/1024	10 0000 0000	1000 0000 0000 0000	0.500	0000 0000 0000 0000	0.000
511/1024	01 1111 1111	0111 1111 1100 0000	0.499	1111 1111 1100 0000	-0.001
		•••			
1/1024	00 0000 0001	0000 0000 0100 0000	0.001	1000 0000 0100 0000	-0.499
0/1024	00 0000 0000	0000 0000 0000 0000	0.000	1000 0000 0000 0000	-0.500

- Impulso (trigger) per fine campionamento e avvio conversione
  Corrispondenza nel registro: bits 7÷5 (*SSRC<2:0>*)
  Valori possibili:
  ADC_CLK_AUTO contatore interno al modulo
  ADC_CLK_CMTU riservato al modulo CMTU
  ADC_CLK_TMR5 comparazione del timer5
  ADC_CLK_TMR3 comparazione del timer3
  ADC_CLK_INT0 transizione del pin INT0
  ADC_CLK_MANUAL a mano, portando a zero il bit SAMP

- Modalità di avvio campionamento
  Corrispondenza nel registro: bit 2 (*ASAM*)
  Valori possibili:
  ADC_AUTO_SAMPLING_ON il campionamento viene avviato in automatico non appena l'ultima conversione è stata terminata
  ADC_AUTO_SAMPLING_OFF il campionamento viene avviato manualmente impostando a 1 il bit SAMP

- Avvio campionamento
  Corrispondenza nel registro: bit 1 (*SAMP*)
  Valori possibili:

ADC_SAMP_ON	avvio campionamento/campionamento in corso
ADC_SAMP_OFF	stop campionamento/campionamento terminato

- Fine conversione
  Corrispondenza nel registro: bit 0 (*DONE*)
  Questo bit è di sola lettura, viene sfruttato da alcune macro per rilevare la fine conversione qualora stiamo facendo funzionare il modulo in modalità manuale.

config2 serve ad impostare il registro **AD1CON2**

- Tensione di riferimento
  Corrispondenza nel registro: bits 15÷14 (*VCFG<2:0>*)
  Valori possibili:

ADC_VREF_AVDD_AVSS	riferimento+ = Vdd	riferimento- = Vss
ADC_VREF_EXT_AVSS	riferimento+ = Vref+	riferimento- = Vss
ADC_VREF_AVDD_EXT	riferimento+ = Vdd	riferimento- = Vref-
ADC_VREF_EXT_EXT	riferimento+ = Vref+	riferimento- = Vref-

  Nota
  Relativamente al PIC24FJ64GB002, quando si usano i riferimenti di tensione esterna, questi vanno collegati ai pin 2 (Vref+) e 3 (Vref-) perdendo l'eventuale funzionalità del posizionamento alternativo del modulo I2C.

- Scansione canali
  (*funzione offerta solo dal MUX A, il MUX B non ha questa opzione)*
  Corrispondenza nel registro: bit 10 (*CSCNA*)
  Valori possibili:

ADC_SCAN_ON	scansione attiva
ADC_SCAN_OFF	scansione non attiva

- Modalità generazione interrupt
  Corrispondenza nel registro: bits 5÷2 (*SMPI<3:0>*)
  Valori possibili:

ADC_INTR_EACH_CONV	interrupt dopo ogni conversione
ADC_INTR_2_CONV	interrupt dopo 2 conversioni
...	
ADC_INTR_x_CONV	interrupt dopo x conversioni (x da 2 a 16)

Nota
Su altri pic a 16bit al posto di ADC_INTR_EACH_CONV *abbiamo* ADC_INTR_1_CONV*, altri pic ancora possono generare l'interrupt fino a dopo 32 conversioni.*

- Modalità funzionamento buffer
  Corrispondenza nel registro: bit 1 (*BUFM*)
  Valori possibili:
  ADC_ALT_BUF_ON due buffer da 8 words ciascuno, utilizzati in maniera alternata
  ADC_ALT_BUF_OFF unico buffer da 16 words

- Utilizzo alternato dei due multiplexers
  Corrispondenza nel registro: bit 0 (*ALTS*)
  Valori possibili:
  ADC_ALT_INPUT_ON utilizza prima l'ingresso del sul MUX A e dopo quello del MUX B e così via, alternativamente
  ADC_ALT_INPUT_OFF utilizza solo l'ingresso del MUX A

config3 serve ad impostare il registro **AD1CON3**

- Sorgente di clock
  Corrispondenza nel registro: bit 15 (*ADRC*)
  Valori possibili:
  ADC_CONV_CLK_INTERNAL_RC clock da oscillatore RC integrato
  ADC_CONV_CLK_SYSTEM clock da oscillatore di sistema

- Tempo di auto-campionamento
  Questa impostazione viene utilizzata quando ASAM=1
  Corrispondenza nel registro: bits 12÷8 (*SAMC<4:0>*)
  Valori possibili:
  ADC_SAMPLE_TIME_0 tempo di campionamento: 0, non consigliato
  ADC_SAMPLE_TIME_1 tempo di campionamento: 1*Tad
  ...
  ADC_SAMPLE_TIME_x tempo di campionamento: x*Tad (x da 0 a 31)

- Tempo di conversione
  Corrispondenza nel registro: bits 7÷0 (*ADCS<7:0>*)
  Valori possibili:
  ADC_CONV_CLK_1Tcy tempo di conversione: Tcy
  ...
  ADC_CONV_CLK_xTcy tempo di conversione: x*Tcy (x da 1 a 256)

configport serve ad impostare il registro **AD1PCFG**

In questo registro viene selezionato quali porte devono funzionare come analogiche e quali no

- Abilitazione ingresso riferimento bandgap
  Corrispondenza nel registro: bit 15 (*PCFG15*)
  Valori possibili:
  `BG_VREF_DISABLED` riferimento bandgap disattivato
  `BG_VREF_ENABLED` riferimento bandgap attivo

- Abilitazione ingresso riferimento metà del valore di bandgap
  Corrispondenza nel registro: bit 14 (*PCFG14*)
  Valori possibili:
  `BG_VREF_DIV2_DISABLED` riferimento bandgap/2 disattivato
  `BG_VREF_DIV2_ENABLED` riferimento bandgap/2 attivo

- Abilitazione ingresso tensione di riferimento regolatore interno
  Corrispondenza nel registro: bit 13 (*PCFG13*)
  Valori possibili:
  `VOLT_REG_DISABLED` riferimento Vddcore disattivato
  `VOLT_REG_ENABLED` riferimento Vddcore attivo

- Abilitazione ingressi analogici/digitali
  Corrispondenza nel registro: bits 12÷0 (*PCFG<12:0>*)
  Valori possibili (anche più di uno, combinati con un AND):
  `ENABLE_ANx_DIG` ingresso ANx come digitale
  `ENABLE_ANx_ANA` ingresso ANx come analogico

  Nota:
  In realtà normalmente non serve utilizzare il valore ENABLE_ANx_DIG *dato che sui pic a 16bit le porte analogiche normalmente sono disattivate a favore di quelle digitali.*

configscan serve ad impostare il registro **AD1CSSL**

In questo registro viene riposto l'elenco delle porte da far scansionare dal MUX A quando la funzione di scansione automatica (bit CSCNA) è abilitata.

- Abilitazione scansione ingresso bandgap
  Corrispondenza nel registro: bit 15 (*CSSL15*)
  Valori possibili:
  `ADC_SCAN_BG_VREF` ingresso bandgap incluso nella scansione

`ADC_SCAN_SKIP_BG_VREF` ingresso bandgap non incluso nella scansione

- Abilitazione scansione ingresso metà valore di bandgap
  Corrispondenza nel registro: bit 14 (*CSSL14*)
  Valori possibili:
  `ADC_SCAN_BG_VREF_DIV2` ingresso bandgap/2 incluso nella scansione
  `ADC_SCAN_SKIP_BG_VREF_DIV2` ingresso bandgap/2 non incluso nella scansione

- Abilitazione scansione ingresso tensione di riferimento regolatore interno
  Corrispondenza nel registro: bit 13 (*CSSL13*)
  Valori possibili:
  `ADC_SCAN_VOLT_REG` Ingresso Vddcore incluso nella scansione
  `ADC_SCAN_SKIP_VOLT_REG` Ingresso Vddcore non incluso nella scansione

- Abilitazione scansione ingresso analogico
  Corrispondenza nel registro: bits 12÷0 (*CSSL<12:0>*)
  `ADC_SCAN_ANx` ingresso analogico ANx incluso nella scansione
  `ADC_SCAN_SKIP_ANx` ingresso analogico ANx non incluso nella scansione

  (x da 0 a 16. Sui pic a 28 pin non sono presenti AN6, AN7, AN8 e AN12)

In alternativa è possibile utilizzare l'unico valore `ENABLE_ALL_INPUT_SCAN` o `DISABLE_ALL_INPUT_SCAN` il cui funzionamento, dai nomi, dovrebbe risultare ovvio.

Ora facciamo un esempio di utilizzo della funzione di configurazione:

```
// Impostazione AD1CON1
unsigned int config1 =
ADC_MODULE_ON & // modulo ON
ADC_IDLE_STOP & // in modalità idle il modulo è spento
ADC_FORMAT_INTG & // risultato come integer a 10bit
ADC_CLK_AUTO & // conversione avviata da contatore interno
ADC_AUTO_SAMPLING_ON & // campionamento eseguito in automatico
dopo conversione
ADC_SAMP_ON; // avvia il primo campionamento

// Impostazione AD1CON2
unsigned int config2 =
ADC_VREF_AVDD_AVSS & // riferimenti di tensione : VDD e VSS
ADC_SCAN_ON & // scansiona canali per MUX A
ADC_INTR_3_CONV & // interrupt dopo 3 conversioni
ADC_ALT_BUF_OFF & // unico buffer costituito da 16 words
ADC_ALT_INPUT_OFF; // utilizza esclusivamente MUX A
```

```
// Impostazione AD1CON3
unsigned int config3 =
ADC_SAMPLE_TIME_2 & // tempo di autocampionamento = 2TAD
ADC_CONV_CLK_INTERNAL_RC & // sorgente di clock ricavata da RC
interna
ADC_CONV_CLK_1Tcy; // clock convertitore ad = TCY

// apro il modulo. Imposto le porta AN0, AN1 e AN2 come analogiche
e di conseguenza le inserisco anche nell'elenco della scansione

OpenADC10(config1, config2, config3,
ENABLE_AN0_ANA & ENABLE_AN1_ANA & ENABLE_AN2_ANA,
ADC_SCAN_AN0 & ADC_SCAN_AN1 & ADC_SCAN_AN2);
```

Il codice in esempio configura il modulo come descritto dalle note ed esegue in automatico il campionamento dei 3 canali selezionati. Allo scattare dell'interrupt, impostato per aversi ogni 3 conversioni, i valori dei canali AN0, AN1 e AN2 saranno memorizzati rispettivamente nei buffers ADC1BUF0, ADC1BUF1, ADC1BUF2. I valori possono essere letti tramite l'apposita funzione che vedremo dopo.

Nelle tabelle seguenti, estratte dal Family Reference Manual dei PIC24F, è possibile capire come opera il modulo ADC e come vengono memorizzati i risultati sfruttando le varie opzioni di configurazione e in particolar modo come operano le varie selezioni di scansione automatica, buffer alternato, interrupt e utilizzo alternato dei due Multiplexers:

Example 17-7: Converting a Single Channel 16 Times per Interrupt

A/D Configuration:

- Select AN0 for S/H+ Input (CH0SA<3:0> = 0000)
- Select VR- for S/H- Input (CH0NA = 0)
- Configure for No Input Scan (CSCNA = 0)
- Use Only MUX A for Sampling (ALTS = 0)
- Set AD1IF on Every 16th Sample (SMPI<3:0> = 1111)
- Configure Buffers for Single, 16-Word Results (BUFM = 0)

Operational Sequence:

1. Sample MUX A Input AN0; Convert and Write to Buffer 0h
2. Sample MUX A Input AN0; Convert and Write to Buffer 1h
3. Sample MUX A Input AN0; Convert and Write to Buffer 2h
4. Sample MUX A Input AN0; Convert and Write to Buffer 3h
5. Sample MUX A Input AN0; Convert and Write to Buffer 4h
6. Sample MUX A Input AN0; Convert and Write to Buffer 5h
7. Sample MUX A Input AN0; Convert and Write to Buffer 6h
8. Sample MUX A Input AN0; Convert and Write to Buffer 7h
9. Sample MUX A Input AN0; Convert and Write to Buffer 8h
10. Sample MUX A Input AN0; Convert and Write to Buffer 9h
11. Sample MUX A Input AN0; Convert and Write to Buffer Ah
12. Sample MUX A Input AN0; Convert and Write to Buffer Bh
13. Sample MUX A Input AN0; Convert and Write to Buffer Ch
14. Sample MUX A Input AN0; Convert and Write to Buffer Dh
15. Sample MUX A Input AN0; Convert and Write to Buffer Eh
16. Sample MUX A Input AN0; Convert and Write to Buffer Fh
17. Set AD1IF Flag (and generate interrupt, if enabled)
18. Repeat (1-16) after Return from Interrupt

Results Stored in Buffer (after 2 cycles):

Buffer Address	Buffer Contents at 1st AD1IF Event	Buffer Contents at 2nd AD1IF Event
ADC1BUF0	AN0, Sample 1	AN0, Sample 17
ADC1BUF1	AN0, Sample 2	AN0, Sample 18
ADC1BUF2	AN0, Sample 3	AN0, Sample 19
ADC1BUF3	AN0, Sample 4	AN0, Sample 20
ADC1BUF4	AN0, Sample 5	AN0, Sample 21
ADC1BUF5	AN0, Sample 6	AN0, Sample 22
ADC1BUF6	AN0, Sample 7	AN0, Sample 23
ADC1BUF7	AN0, Sample 8	AN0, Sample 24
ADC1BUF8	AN0, Sample 9	AN0, Sample 25
ADC1BUF9	AN0, Sample 10	AN0, Sample 26
ADC1BUFA	AN0, Sample 11	AN0, Sample 27
ADC1BUFB	AN0, Sample 12	AN0, Sample 28
ADC1BUFC	AN0, Sample 13	AN0, Sample 29
ADC1BUFD	AN0, Sample 14	AN0, Sample 30
ADC1BUFE	AN0, Sample 15	AN0, Sample 31
ADC1BUFF	AN0, Sample 16	AN0, Sample 32

Example 17-9: Scanning and Converting All 16 Channels per Single Interrupt

A/D Configuration:

- Select Any Channel for S/H+ Input (CH0SA<3:0> = xxxx)
- Select VR- for S/H- Input (CH0NA = 0)
- Use Only MUX A for Sampling (ALTS = 0)
- Configure MUX A for Input Scan (CSCNA = 1)
- Include All Analog Channels in Scanning (AD1CSSL = 1111 1111 1111 1111)
- Set AD1IF on Every 16th Sample (SMPI<3:0> = 1111)
- Configure Buffers for Single, 16-Word Results (BUFM = 0)

Operational Sequence:

1. Sample MUX A Input AN0; Convert and Write to Buffer 0h
2. Sample MUX A Input AN1; Convert and Write to Buffer 1h
3. Sample MUX A Input AN2; Convert and Write to Buffer 2h
4. Sample MUX A Input AN3; Convert and Write to Buffer 3h
5. Sample MUX A Input AN4; Convert and Write to Buffer 4h
6. Sample MUX A Input AN5; Convert and Write to Buffer 5h
7. Sample MUX A Input AN6; Convert and Write to Buffer 6h
8. Sample MUX A Input AN7; Convert and Write to Buffer 7h
9. Sample MUX A Input AN8; Convert and Write to Buffer 8h
10. Sample MUX A Input AN9; Convert and Write to Buffer 9h
11. Sample MUX A Input AN10; Convert and Write to Buffer Ah
12. Sample MUX A Input AN11; Convert and Write to Buffer Bh
13. Sample MUX A Input AN12; Convert and Write to Buffer Ch
14. Sample MUX A Input AN13; Convert and Write to Buffer Dh
15. Sample MUX A Input AN14; Convert and Write to Buffer Eh
16. Sample MUX A Input AN15; Convert and Write to Buffer Fh
17. Set AD1IF Flag (and generate interrupt, if enabled)
18. Repeat (1-16) after Return from Interrupt

Results Stored in Buffer (after 2 cycles):

Buffer Address	Buffer Contents at 1st AD1IF Event	Buffer Contents at 2nd AD1IF Event
ADC1BUF0	Sample 1 (AN0, Sample 1)	Sample 17 (AN0, Sample 2)
ADC1BUF1	Sample 2 (AN1, Sample 1)	Sample 18 (AN1, Sample 2)
ADC1BUF2	Sample 3 (AN2, Sample 1)	Sample 19 (AN2, Sample 2)
ADC1BUF3	Sample 4 (AN3, Sample 1)	Sample 20 (AN3, Sample 2)
ADC1BUF4	Sample 5 (AN4, Sample 1)	Sample 21 (AN4, Sample 2)
ADC1BUF5	Sample 6 (AN5, Sample 1)	Sample 22 (AN5, Sample 2)
ADC1BUF6	Sample 7 (AN6, Sample 1)	Sample 23 (AN6, Sample 2)
ADC1BUF7	Sample 8 (AN7, Sample 1)	Sample 24 (AN7, Sample 2)
ADC1BUF8	Sample 9 (AN8, Sample 1)	Sample 25 (AN8, Sample 2)
ADC1BUF9	Sample 10 (AN9, Sample 1)	Sample 26 (AN9, Sample 2)
ADC1BUFA	Sample 11 (AN10, Sample 1)	Sample 27 (AN10, Sample 2)
ADC1BUFB	Sample 12 (AN11, Sample 1)	Sample 28 (AN11, Sample 2)
ADC1BUFC	Sample 13 (AN12, Sample 1)	Sample 29 (AN12, Sample 2)
ADC1BUFD	Sample 14 (AN13, Sample 1)	Sample 30 (AN13, Sample 2)
ADC1BUFE	Sample 15 (AN14, Sample 1)	Sample 31 (AN14, Sample 2)
ADC1BUFF	Sample 16 (AN15, Sample 1)	Sample 32 (AN15, Sample 2)

Example 17-10: Converting a Single Channel Once per Interrupt, Dual Buffer Mode

A/D Configuration:

- Select AN3 for S/H+ Input (CH0SA<3:0> = 0011)
- Select VR- for S/H- Input (CH0NA = 0)
- Configure for No Input Scan (CSCNA = 0)
- Use Only MUX A for Sampling (ALTS = 0)
- Set AD1IF on Every Sample (SMPI<3:0> = 0000)
- Configure Buffer as Dual, 8-Word Segments (BUFM = 1)

Operational Sequence:

1. Sample MUX A Input AN3; Convert and Write to Buffer 0h
2. Set AD1IF Flag (and generate interrupt, if enabled); Write Access Automatically Switches to Alternate Buffer
3. Sample MUX A Input AN3; Convert and Write to Buffer 8h
4. Set AD1IF Flag (and generate interrupt, if enabled); Write Access Automatically Switches to Alternate Buffer
5. Repeat (1-4)

Results Stored in Buffer (after 2 cycles):

Buffer Address	Buffer Contents at 1st AD1IF Event	Buffer Contents at 2nd AD1IF Event
ADC1BUF0	Sample 1 (AN3, Sample 1)	(undefined)
ADC1BUF1	(undefined)	(undefined)
ADC1BUF2	(undefined)	(undefined)
ADC1BUF3	(undefined)	(undefined)
ADC1BUF4	(undefined)	(undefined)
ADC1BUF5	(undefined)	(undefined)
ADC1BUF6	(undefined)	(undefined)
ADC1BUF7	(undefined)	(undefined)
ADC1BUF8	(undefined)	Sample 2 (AN3, Sample 2)
ADC1BUF9	(undefined)	(undefined)
ADC1BUFA	(undefined)	(undefined)
ADC1BUFB	(undefined)	(undefined)
ADC1BUFC	(undefined)	(undefined)
ADC1BUFD	(undefined)	(undefined)
ADC1BUFE	(undefined)	(undefined)
ADC1BUFF	(undefined)	(undefined)

Example 17-11: Converting Two Inputs by Alternating MUX A and MUX B

A/D Configuration:

- Select AN1 for MUX A S/H+ Input (CH0SA<3:0> = 0001)
- Select VR- for MUX A S/H- Input (CH0NA = 0)
- Configure for No Input Scan (CSCNA = 0)
- Select AN15 for MUX B S/H+ Input (CH0SB<3:0> = 1111)
- Select VR- for MUX B S/H- Input (CH0NB = 0)
- Alternate MUX A and MUX B for Sampling (ALTS = 1)
- Set AD1IF on Every 8th Sample (SMPI<3:0> = 0111)
- Configure Buffer as Two, 8-Word Segments (BUFM = 1)

Operational Sequence:

1. Sample MUX A Input AN1; Convert and Write to Buffer 0h
2. Sample MUX B Input AN15; Convert and Write to Buffer 1h
3. Sample MUX A Input AN1; Convert and Write to Buffer 2h
4. Sample MUX B Input AN15; Convert and Write to Buffer 3h
5. Sample MUX A Input AN1; Convert and Write to Buffer 4h
6. Sample MUX B Input AN15; Convert and Write to Buffer 5h
7. Sample MUX A Input AN1; Convert and Write to Buffer 6h
8. Sample MUX B Input AN15; Convert and Write to Buffer 7h
9. Set AD1IF Flag (and generate interrupt, if enabled); Write Access Automatically Switches to Alternate Buffer
10. Repeat (1-9); Resume Writing to Buffer with Buffer 8h (first address of alternate buffer)

Results Stored in Buffer (after 2 cycles):

Buffer Address	Buffer Contents at 1st AD1IF Event	Buffer Contents at 2nd AD1IF Event
ADC1BUF0	Sample 1 (AN1, Sample 1)	(undefined)
ADC1BUF1	Sample 2 (AN15, Sample 1)	(undefined)
ADC1BUF2	Sample 3 (AN1, Sample 2)	(undefined)
ADC1BUF3	Sample 4 (AN15, Sample 2)	(undefined)
ADC1BUF4	Sample 5 (AN1, Sample 3)	(undefined)
ADC1BUF5	Sample 6 (AN15, Sample 3)	(undefined)
ADC1BUF6	Sample 7 (AN1, Sample 4)	(undefined)
ADC1BUF7	Sample 8 (AN15, Sample 4)	(undefined)
ADC1BUF8	(undefined)	Sample 9 (AN1, Sample 5)
ADC1BUF9	(undefined)	Sample 10 (AN15, Sample 5)
ADC1BUFA	(undefined)	Sample 11 (AN1, Sample 6)
ADC1BUFB	(undefined)	Sample 12 (AN15, Sample 6)
ADC1BUFC	(undefined)	Sample 13 (AN1, Sample 7)
ADC1BUFD	(undefined)	Sample 14 (AN15, Sample 7)
ADC1BUFE	(undefined)	Sample 15 (AN1, Sample 8)
ADC1BUFF	(undefined)	Sample 16 (AN15, Sample 8)

Configurazione canali in ingresso ai multiplexers

Abbiamo visto che il modulo ADC dei pic a 16bit possiede due multiplexers: MUX A e MUX B. Ogni multiplexer ha un ingresso positivo, che viene inviato all'ingresso non invertente dell'amplificatore del modulo S/H, e un ingresso negativo che viene inviato all'ingresso invertente. Bisogna quindi specificare questi 4 ingressi da utilizzare tramite la funzione `SetChanADC10`:

```
void SetChanADC10 (
unsigned int channel);
```

channel serve ad impostare il registro **AD1CHS**

- MUX B – selezione ingresso negativo
  Corrispondenza nel registro: bit 15 (CH0NB)
  Valori possibili:
  `ADC_CH0_NEG_SAMPLEB_AN1` ingresso- MUX B collegato a AN1
  `ADC_CH0_NEG_SAMPLEB_VREFN` ingresso- MUX B collegato a riferimento negativo scelto nella config2 di OpenADC10

- MUX B – selezione ingresso positivo
  Corrispondenza nel registro: bits 12÷8 (CH0SB<4:0>)
  Valori possibili:
  `ADC_CH0_POS_SAMPLEB_AVSS` ingresso+ MUX B collegato a Vss
  `ADC_CH0_POS_SAMPLEB_AVDD` ingresso+ MUX B collegato a Vdd
  `ADC_CH0_POS_SAMPLEB_VBG` ingresso+ MUX B collegato a Bandgap
  `ADC_CH0_POS_SAMPLEB_VBGDIV2` ingresso+ MUX B collegato a Bandgap/2
  `ADC_CH0_POS_SAMPLEB_VDDCORE` ingresso+ MUX B collegato a Vddcore
  `ADC_CH0_POS_SAMPLEB_ANx` ingresso+ MUX B collegato a porta ANx

  (x da 0 a 16. Sui pic a 28 pin non sono presenti AN6, AN7, AN8 e AN12)

- MUX A – selezione ingresso negativo
  Corrispondenza nel registro: bit 7 (CH0NA)
  Valori possibili:
  `ADC_CH0_NEG_SAMPLEA_AN1` ingresso- MUX A collegato a AN1
  `ADC_CH0_NEG_SAMPLEA_VREFN` ingresso- MUX A collegato a riferimento negativo scelto nella config2 di OpenADC10

- MUX A – selezione ingresso positivo
  Corrispondenza nel registro: bits 4÷0 (CH0SA<4:0>)
  Valori possibili:
  `ADC_CH0_POS_SAMPLEA_AVSS` ingresso+ MUX A collegato a Vss

ADC_CH0_POS_SAMPLEA_AVDD	ingresso+ MUX A collegato a Vdd
ADC_CH0_POS_SAMPLEA_VBG	ingresso+ MUX A collegato a Bandgap
ADC_CH0_POS_SAMPLEA_VBGDIV2	ingresso+ MUX A collegato a Bandgap/2
ADC_CH0_POS_SAMPLEA_VDDCORE	ingresso+ MUX A collegato a Vddcore
ADC_CH0_POS_SAMPLEA_ANx	ingresso+ MUX A collegato a porta ANx

(x da 0 a 16. Sui pic a 28 pin non sono presenti AN6, AN7, AN8 e AN12)

Attivando la modalità di scansione automatica, l'impostazione dell'ingresso positivo del MUX A non viene presa in considerazione in quanto la scansione viene eseguita in ordine crescente tra le porte selezionate.

Ora facciamo un esempio di utilizzo della funzione di impostazione dei canali in ingresso ai multiplexers:

```
unsigned int channel =
ADC_CH0_NEG_SAMPLEB_VREFN & // ingresso negativo MUX B su
riferimento negativo
ADC_CH0_POS_SAMPLEB_AN0 & // ingresso positivo MUX B su AN0
ADC_CH0_NEG_SAMPLEA_VREFN & // ingresso negativo MUX A su
riferimento negativo
ADC_CH0_POS_SAMPLEA_AN1; // ingresso positivo MUX A su AN1

// Eseguo l'impostazione
SetChanADC10 (channel);
```

Se utilizziamo esclusivamente il MUX A in realtà non è necessario eseguire l'impostazione anche per il MUX B...

Abilitazione interrupt su fine conversione

Abbiamo visto che l'interrupt può essere impostato per scattare subito dopo la prima conversione o dopo un certo numero di conversioni. Se vogliamo sfruttare una ISR per eseguire la lettura del buffer, allora dobbiamo anche abilitare l'interrupt ed eventualmente impostarne la priorità con la funzione `ConfigIntADC10`:

```
void ConfigIntADC10 (unsigned int config);
```

- Livello di priorità interrupt
  Valori possibili:
  ADC_INT_PRI_x Livello di priorità x (con x da 0 a 7, default 4)

- Abilitazione interrupt
  Valori possibili:

`ADC_INT_ENABLE`	interrupt abilitato
`ADC_INT_DISABLE`	interrupt disabilitato

Esempio di utilizzo:

```
// abilito l'interrupt su fine conversione e imposto il livello di priorità a 4
ConfigIntADC10 (ADC_INT_PRI_4 & ADC_INT_ENABLE);
```

L'evento di interrupt si può anche abilitare o disabilitare sfruttando le due macro `EnableIntADC1` e `DisableIntADC1`.

Il livello di priorità può anche essere impostato utilizzando la macro
`SetPriorityIntADC1(priority)`

Il flag di interrupt può essere azzerato sfruttando la macro
`ADC1_Clear_Intr_Status_Bit`

Esempio di ISR per la cattura dell'interrupt su fine conversione modulo ADC:

```
void __attribute__((interrupt,no_auto_psv)) _ADC1Interrupt(void)
     {
     static unsigned char DataReady = 1; // esempio di flag da controllare nel main per capire quando il risultato della conversione è pronto
     ADC1_Clear_Intr_Status_Bit;
     }
```

Maggiori informazioni sulla gestione degli interrupt sui pic a 16bit possono essere trovate nel seguente articolo su settorezero.com :

http://www.settorezero.com/wordpress/la-gestione-degli-interrupt-sui-pic12-pic16-pic18-pic24-dspic/

Altre funzioni

Lettura buffer

```
unsigned int ReadADC10 (unsigned int buffer_index);
```

Esegue la lettura del buffer indicato da buffer_index (valore da 0 a 15). Restituisce in pratica il valore contenuto nei registri ADC1BUF0 ÷ ADC1BUFF

Esempio:

```
// leggo il contenuto del buffer 0
static unsigned int buffer_value = ReadADC10(0);
// leggo il contenuto del buffer 15
static unsigned int buffer_value_2 = ReadADC10(15);
```

Ferma il campionamento ed avvia la conversione

```
ConvertADC10();
// oppure, che è la stessa identica cosa:
StopSampADC1();// non ho sbagliato a scrivere, c'è 1 e non 10
```

Questa funzione verrà utilizzata esclusivamente nel caso in cui abbiamo scelto di eseguire il campionamento in modalità manuale (ovvero abbiamo impostato l'auto-sampling ad OFF). Il primo campionamento viene avviato dalla funzione di configurazione se abbiamo impostato `ADC_SAMP_ON`, oppure se l'abbiamo impostato ad OFF, può essere avviato manualmente ponendo SAMP=1, cosa che va fatta comunque ogni volta nel caso di impostazione manuale.

Stato conversione

```
char BusyADC10 (void);
```

Da usare in caso di modalità di campionamento manuale. Restituisce il valore invertito del bit DONE. In pratica tale funzione restituisce 1 se il modulo è occupato (ovvero sta eseguendo una conversione) e 0 se la conversione è terminata e quindi il dato è disponibile.

Spegnimento modulo ADC

```
void CloseADC10 (void);
```

Spegne il modulo ADC, disabilita gli interrupt del modulo ADC e pone a zero il flag di interrupt su fine conversione.

NOTE

«Non ho particolari talenti, sono solo appassionatamente curioso»
A. Einstein