LaurTec

ATmega 2560

Corso di programmazione per microcontrollori Atmel

Autore : *Marcello Pinna*

ID: UT0005-IT

Copyright © 2020 Mauro Laurenti

INFORMATIVA

Come prescritto dall'art. 1, comma 1, della legge 21 maggio 2004 n.128, l'autore avvisa di aver assolto, per la seguente opera dell'ingegno, a tutti gli obblighi della legge 22 Aprile del 1941 n. 633, sulla tutela del diritto d'autore.
Tutti i diritti di questa opera sono riservati. Ogni riproduzione ed ogni altra forma di diffusione al pubblico dell'opera, o parte di essa, senza un'autorizzazione scritta dell'autore, rappresenta una violazione della legge che tutela il diritto d'autore, in particolare non ne è consentito un utilizzo per trarne profitto.
La mancata osservanza della legge 22 Aprile del 1941 n. 633 è perseguibile con la reclusione o sanzione pecuniaria, come descritto al Titolo III, Capo III, Sezione II.
A norma dell'art. 70 è comunque consentito, per scopi di critica o discussione, il riassunto e la citazione, accompagnati dalla menzione del titolo dell'opera e dal nome dell'autore.

AVVERTENZE

I progetti presentati non hanno la marcatura CE, quindi non possono essere utilizzati per scopi commerciali nella Comunità Economica Europea.

Chiunque decida di far uso delle nozioni riportate nella seguente opera o decida di realizzare i circuiti proposti, è tenuto pertanto a prestare la massima attenzione in osservanza alle normative in vigore sulla sicurezza.
L'autore declina ogni responsabilità per eventuali danni causati a persone, animali o cose derivante dall'utilizzo diretto o indiretto del materiale, dei dispositivi o del software presentati nella seguente opera.
Si fa inoltre presente che quanto riportato viene fornito così com'è, a solo scopo didattico e formativo, senza garanzia alcuna della sua correttezza.

L'autore ringrazia anticipatamente per la segnalazione di ogni errore.

Tutti i marchi citati in quest'opera sono dei rispettivi proprietari.

Indice

Capitolo IX....4
Il modulo ADC – Convertitore Analogico Digitale....4
Alimentazione del modulo ADC, AVCC....7
Registri associati al convertitore ADC....8
Funzione per l'inizializzazione dell'ADC....13
Selezione ed impostazione dei canali ADC....14
Inizio e termine della conversione....14
Calcolo della conversione....16
Visualizzazione del risultato....16
Programma di esempio....17
History....21

Capitolo IX

Il modulo ADC – Convertitore Analogico Digitale

Il modulo ADC (*Analog Digital Converter*), come dice il nome stesso, è un convertitore che permette di codificare un segnale analogico in ingresso, in una grandezza digitale. In particolare il convertitore ADC integrato nella MCU Atmega 2560 è di tipo SAR (*Successive Approximation Register*). Esso è collegato ad un multiplexer analogico che gestisce da 8 a 16 canali, ognuno dei quali può misurare indipendentemente tensioni diverse riferite sempre a massa (*single-ended*). Essendo presente un multiplexer, i canali possono essere scansionati uno alla volta ciclicamente, a seconda del canale selezionato. I primi otto canali, utilizzati in questo capitolo, sono rappresentati dai pin 0 - 7 della PORTF, gli altri otto dai pin 0 - 7 della PORTK. In Figura 1 è mostrato il diagramma a blocchi del convertitore ADC, a cui seguirà una breve descrizione delle sezioni di interesse.

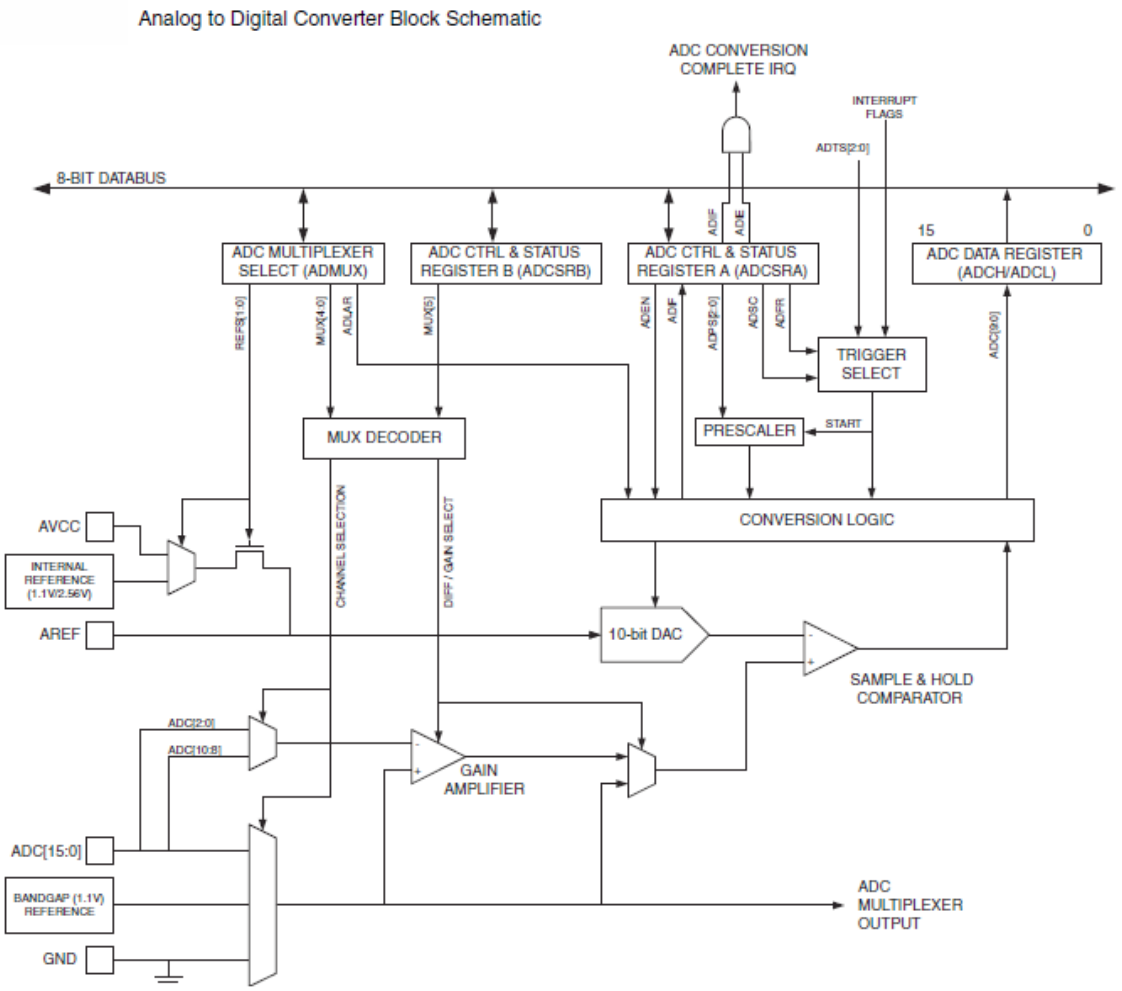

Figura 1: *Diagramma a blocchi del convertitore ADC.*

Circuito di ingresso di un pin analogico single-ended

In Figura 2 è mostrato il modello semplificato dello schema elettrico di un ingresso analogico (*single-ended*).

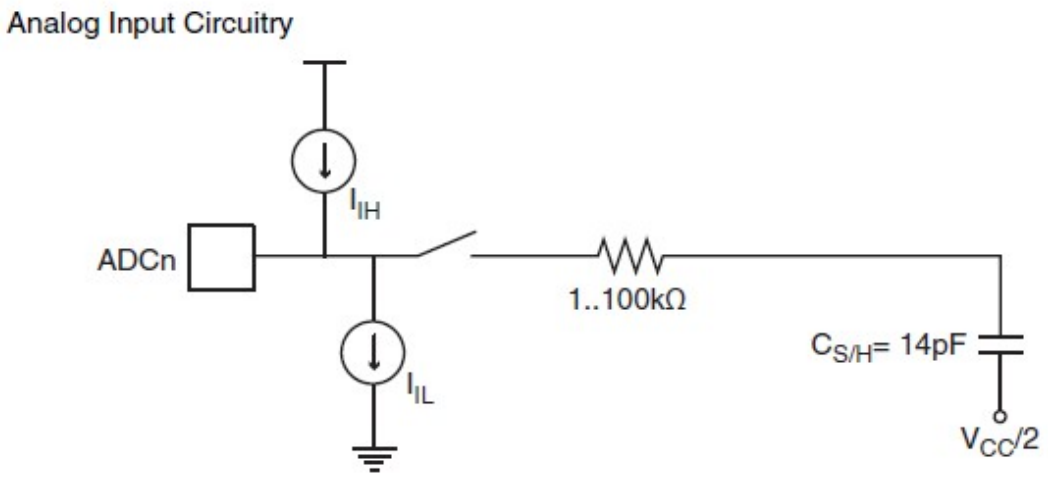

Figura 2: *Schema elettrico semplificato di un pin analogico.*

Questo circuito è ottimizzato per segnali analogici con impedenza di circa 10KΩ o meno. Con sorgenti di segnali ad impedenza più alta si possono verificare problemi di lettura durante la commutazione tra un pin ed un altro del multiplexer. In particolare il condensatore in ingresso del *Sample* & *Hold* (S&H), ovvero $C_{S/H}$, può non essere caricato correttamente durante il tempo di campionamento. E' cura del programmatore e progettista, garantire che i tempi di campionamento siano idonei alle impedenza della sorgente di tensione utilizzata. Questo può essere fatto o aumentando il tempo di campionamento (nei limiti concessi) o diminuendo l'impedenza della sorgente del segnale, ovvero facendo uso di un buffer (amplificatore a guadagno unitario).

Comparatore Sample & Hold

Il modulo ADC contiene al suo interno un circuito definito *Sample* & *Hold* (S&H) letteralmente significa Campiona e Mantieni. Il S&H assicura che il segnale in ingresso sia letto (fase di acquisizione o campionamento) e mantenuto a livello costante durante il processo di conversione. Questi processi sono basati su tempi precisi, ovvero su cicli di clock generati dalla frequenza in uscita al prescaler. Il sistema, prima che il segnale sia acquisito, richiede uno *Startup Time* minimo per stabilizzarsi, questo serve per permettere che la tensione di riferimento usata per il convertitore ADC si stabilizzi, come anche il resto dei componenti usati nel convertitore stesso. Durante l'acquisizione si necessita di altro tempo, chiamato *Sample Time*, per caricare il condensatore interno a un valore stabile, ovvero uguale alla tensione in ingresso che si vuole misurare, per ottenere dei risultati accurati. Nella fase finale di conversione, si necessita di ulteriore tempo, chiamato *Hold Time*, per ottenere il segnale analogico convertito in digitale. La somma dei tempi è chiamata *Conversion Time*, ovvero tempo di conversione. In Figura 3 è riportata la Tabella, che riassume il numero di cicli di clock necessari per le varie fasi. Si capisce che in base alla frequenza del clock utilizzati i tempi varieranno di conseguenza.

ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5
Normal conversions, differential	1.5/2.5	13/14

Figura 3: *Tempi di conversione.*

Auto Trigger

La funzione di *Auto Trigger*, permette di avviare automaticamente una conversione, da diverse sorgenti, ad intervalli fissi. Viene abilitato impostando ad 1 il bit ADATE presente nel registro ADCSRA, mentre la sorgente del *Trigger* viene selezionata impostando ad 1 il bit ADTS presente nel registro ADCSRB. Quando viene rilevato un segnale sul *Trigger* selezionato, il prescaler viene azzerato, dando così inizio ad una conversione. Allo stesso tempo viene impostato un flag di interrupt, anche se tutti i bit relativi non sono stati abilitati per questo evento. Il suo utilizzo non verrà trattato in questo tutorial ma riporto comunque il relativo diagramma a blocchi in (Figura 4).

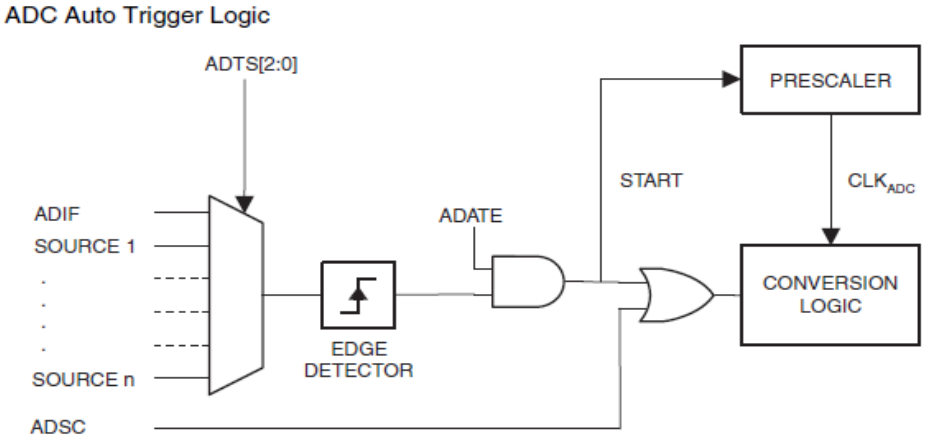

Figura 4: *Logica della funzione Auto Trigger.*

Prescaler

Affinché sia possibile la conversione, è necessaria una frequenza di campionamento f_adc compresa tra 50KHz e 200KHz. Per ottenere tale valore di frequenza, il modulo ADC è provvisto di un prescaler, con valori uguali a 2, 4, 8, 16, 32, 64 e 128, che genera una f_adc valida per qualsiasi frequenza in ingresso alla CPU uguale o superiore a 100KHz. Il prescaler è impostato dai bit ADPS presenti nel registro ADCSRA, inizia a contare dal momento in cui il convertitore è acceso e viene resettato a conversione finita. In Figura 5 è riportato il suo diagramma a blocchi.

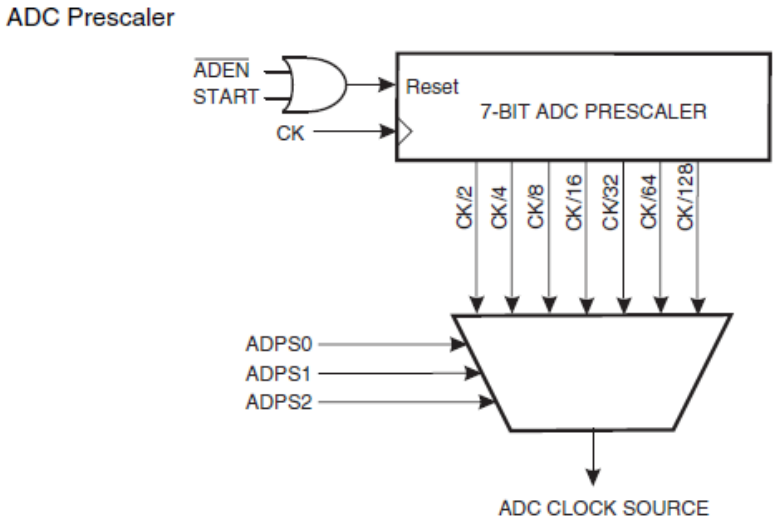

Figura 5: *Schema a blocchi del pre-scaler associato al modulo ADC.*

La frequenza di clock per l'ADC è calcolata con la seguente formula:

f_adc = fc / N

dove: fc = frequenza di clock in ingresso
N = prescaler

Alimentazione del modulo ADC, AVCC

Al fine di poter effettuare la conversione, il modulo ADC deve fare un confronto tra la tensione in ingresso da convertire ed una tensione di riferimento, fissa e stabile, che può avere origine interna o esterna. In questo corso ho deciso di adottare la soluzione più semplice, ovvero un riferimento esterno rappresentato dal pin AVCC. Ogni fluttuazione sulla tensione di alimentazione, derivante dal regolatore lineare o convertitore DC-DC, si riflette sull'accuratezza delle misure effettuate con il modulo ADC. Poiché anche l'Atmega 2560, nelle sue ridottissime dimensioni, genera emissioni elettromagnetiche (EMI) che possono creare interferenze o errori nel calcolo finale della conversione, è opportuno separare l'alimentazione digitale da quella analogica, collegando un filtro passa-basso, composto da un condensatore ed da una induttanza. Lo schema di collegamento è riportato in Figura 6, il quale mostra il dettaglio del filtro passa basso collegato tra l'alimentazione della sezione digitale della MCU, ovvero VCC, e la parte analogica AVCC.

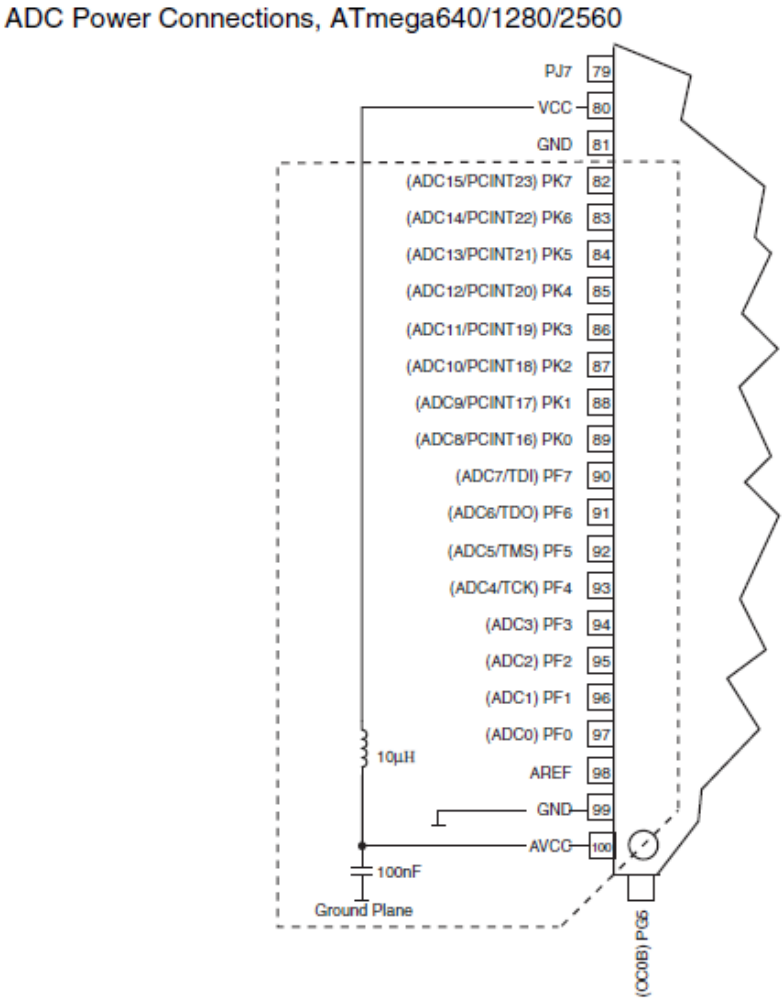

Figura 6: *Schema elettrico associato all'alimentazione AVCC.*

Registri associati al convertitore ADC

ADMUX: ADC Multiplexer Selection Register

Questo è il registro che permette di selezionare la tensione di riferimento, la giustificazione del risultato ed i canali usati per la lettura. In Figura 7 è riportato il dettaglio dei bit che è possibile impostare.

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 7: *Schema ADC multiplexer selection register.*

I bit 7 e 6 REFS1 e REFS0 (*Reference Selection*) selezionano la tensione di riferimento secondo le combinazioni riportate in Figura 8.

Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection[1]
0	0	AREF, Internal V_{REF} turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Note: 1. If 10x or 200x gain is selected, only 2.56V should be used as Internal Voltage Reference. For differential conversion, only 1.1V cannot be used as internal voltage reference.

Figura 8: *Voltage reference selection bits.*

Il bit 5 **ADLAR** (ADC *Left Adjust Result*), se posto ad 1, giustifica il risultato della conversione a sinistra. Di default esso è impostato a 0, ovvero con giustificazione a destra.

I bit 4 - 0 MUX4 - MUX0 (*Analog Channel and Gain Selection*) selezionano i canali di ingresso *single-ended* ADC, come riportato in Figura 9.

Table 26-4. Input Channel Selections

MUX5:0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
000000	ADC0	N/A		
000001	ADC1			
000010	ADC2			
000011	ADC3			
000100	ADC4			
000101	ADC5			
000110	ADC6			
000111	ADC7			

Figura 9: *Input channel selection.*

ADCSRB: ADC Control and Status Register B

Questo registro è usato con il comparatore analogico oppure per selezione la sorgente di auto Trigger. In Figura 10 è riportato il dettaglio dei bit che è possibile impostare.

ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 10: *ADC Control and Status Register.*

Il bit 6 **ACME** (*Analog Comparator Multiplexer Enable*), se posto ad 1, abilita il comparatore analogico.

Il bit 3 **MUX5**, in combinazione con i bit MUX4 - MUX0 (ADMUX) seleziona i canali analogici da usare con il comparatore.

I bit 2 - 0 **ADTS2 - ADTS0** selezionano la sorgente dell'auto *Trigger* in caso di suo utilizzo, come riportato in Figura 11.

ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Note: Free running mode cannot be used for differential channels (see chapter "Differential Channels" on page 273).

Figura 11: *ADC Auto Trigger source selection.*

ADCSRA: ADC Control and Status Register A

Questo è il registro che controlla diversi parametri. In Figura 12 è riportato il dettaglio dei bit che è possibile impostare.

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 12: *ADC Control e Status register A.*

Il bit 7 **ADEN** (ADC *Enable*), se posto ad 1, abilita l'ADC.

Il bit 6 **ADSC** (ADC *Start Conversion*), se posto ad 1, fa partire la conversione singola. Questo bit verrà azzerato al termine della conversione e dovrà essere reimpostato manualmente ad 1 per una nuova conversione.

Il bit 5 **ADATE** (ADC *Auto trigger Enable*), se posto ad 1, abilita l'auto *Trigger*.

Il bit 4 **ADIF** (ADC *Interrupt Flag*) è un flag di evento interrupt che viene sempre impostato ad 1 a fine conversione e dopo l'aggiornamento del *Data Register*, anche se l'evento non viene richiesto.

Il bit 3 **ADIE** (ADC *Interrupt Enable*), se posto ad 1, abilita l'interrupt sul vettore ADC. L'interrupt verrà eseguito solo se anche I-bit nel registro SREG verrà impostato ad 1. In questo caso, il flag ADIF è azzerato automaticamente dall'hardware oppure manualmente dall'utente.

I bit 2 - 0 **ADPS2 - ADPS0** (ADC *Prescaler Selection*) selezionano il fattore di divisione in base alle combinazioni riportate in Figura 13.

Table 26-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Figura 13: *ADC Prescaler selection.*

ADCL - ADCH: ADC Data Register

Sono i due registri ad 8 bit che compongono il *Data Register* e che contengono il risultato di una conversione nei bit ADC9 - ADC0. In Figura 14 e 15 sono riportati il relativi registri e posizione dei bit con ADLAR impostato a 0 (giustificazione a destra) e ADLAR impostato a 1 (giustificazione a sinistra).

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figura 14: *ADCH - ADCL Data Register* .

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figura 15: *ADCH 1 ADCL Data Register.*

Quando il risultato della conversione a 10 bit è giustificato a destra (ADLAR = 0), l'aggiornamento del *Data Register* avviene leggendo per primo il registro ADCL e successivamente il registro ADCH. Nel caso di una giustificazione a sinistra (ADLAR = 1), dove non è richiesta una precisione superiore a 8 bit, esso avviene leggendo solamente il registro ADCH

DIDR0: Digital Input Disable Register 0

Questo registro è relativo ai bit 7-0 della PORTF e serve per disabilitare gli ingressi digitali quando li si usa come ingressi analogici. In Figura 16 è riportato il dettaglio dei bit che è possibile impostare.

DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0	
(0x7E)	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 16: *DIDR0 – Digital Input Disable Register 0.*

Ogni pin della PORTF può lavorare, oltre che in modo analogico, anche in modo digitale. Impostando ad 1 i corrispondenti bit nel registro DIDR0 verranno disabilitati i relativi input digitali. Quando disabilitati, gli ingressi digitali sono letti sempre come 0 anche nel registro PINx.

DIDR2: Digital Input Disable Register 2

Questo registro è relativo ai bit 15-8 della PORTK e serve per disabilitare gli ingressi digitali quando li si usa come ingressi analogici. In Figura 17 è riportato il dettaglio dei bit che è possibile impostare.

DIDR2 – Digital Input Disable Register 2

Bit	7	6	5	4	3	2	1	0	
(0x7D)	ADC15D	ADC14D	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D	DIDR2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 17: *DIDR2 – Digital Input Disable Register 2.*

Ogni pin della PORTK può lavorare, oltre che in modo analogico, anche in modo digitale. Impostando ad 1 i corrispondenti bit nel registro DIDR2 verranno disabilitati i relativi input digitali. Quando disabilitati, gli ingressi digitali sono letti sempre come 0 anche nel registro PINx.

Funzione per l'inizializzazione dell'ADC

In questo tutorial, come detto, si prenderà in esame solo la PORTF e si lavorerà in modo conversione singola sui pin *single-ended*. La logica di inizializzazione dell'ADC è la seguente:

1. Scegliere una sorgente di tensione di riferimento.
2. Calcolare il valore ottimale del prescaler.
3. Abilitare l'ADC.

Come tensione di riferimento si è scelto AVCC. In base alle combinazioni previste nella Figura 8, per selezionarla si deve portare a 1 il solo bit REFS0 nel registro ADMUX:

```
ADMUX = (1 << REFS0);    // AREF = AVCC
```

Ora si deve calcolare un valore per il prescaler, affinché la sua frequenza di uscita sia compresa tra 50 KHz e 250 KHz in base alla frequenza di clock in ingresso. Nella demoboard usata, è presente un quarzo da 16MHz per cui si deve calcolare il valore f_adc per ogni divisore di frequenza N. La seguente Tabella riporta gli unici valori che possono essere presi in considerazione:

f_C	N	f ADC
16.000.000	2	8.000.000
	4	4.000.000
	8	2.000.000
	16	1.000.000
	32	500.000
	64	250.000
	128	150.000

Tra i due valori N è preferibile scegliere quello più alto perché più bassa è la frequenza di campionamento maggiore è l'accuratezza del risultato. Ciononostante bisogna sempre garantire che la frequenza di campionamento sia sempre maggiore del doppio della banda del segnale in ingresso, soddisfacendo cosi il teorema di Nyquist.

Si ricorda che per segnali variabili nel tempo devono sempre prevedere un filtro passa basso prima dell'ADC, per limitare la banda ad un valore noto e per poter cosi impostare la frequenza di campionamento in maniera opportuna.

Nel caso di segali in ingesso in continua, si può scegliere N più grande possibile, al fine di ridurre i consumi. Si sceglie quindi il prescaler N = 128 e si impostano ad 1 i bit ADPS2, ADPS1 e ADPS0 nel registro nel registro ADCSRA, impostando ad 1 allo stesso tempo anche il bit ADEN per abilitare l'ADC:

```
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
```

Scelti i parametri, si può quindi implementare la seguente funzione per l'inizializzazione dell'ADC:

```
void adc_init(){

 // AREF = AVcc
 ADMUX = (1<<REFS0);

 // Abilito l'ADC, prescaler = 128 (16000000 Hz/128 = 125000 Hz)
 ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```

Selezione ed impostazione dei canali ADC

La selezione dei canali viene effettuata tramite l'impostazione dei bit MUX4 - MUX0 nel registro ADMUX, come riportato in Figura 9. La selezione di un singolo canale comporta semplicemente la scrittura manuale ad 1 dei bit interessati (ad esempio, se si usa il canale 3 basta scrivere ADMUX = (1<<MUX0) | (1<<MUX1). Come detto all'inizio del capitolo, tutti gli ingressi ADC sono connessi al convertitore tramite un multiplexer. Tutto questo ragionamento è riassunto nelle seguenti seguenti istruzioni:

```
// Definisco la bitmask per la lettura sequenziale dei sette canali
//adc della PORTF
#define BITMASK 0b00000111

// Creo la variabile che conterrà la lettura adc.....
int adc_read (char ch)

// Seleziono sequenzialmente i canali 0-7
ch &= BITMASK;

// Azzero gli ultimi tre bit di ADMUX e lo imposto con il nuovo canale
//da leggere
ADMUX = (ADMUX & 0b11111000)|ch;
```

Inizio e termine della conversione

Per far partire la conversione singola bisogna impostare ad 1 il bit ADSC nel registro ADCSRA:

```
ADCSRA = (1 << ADSC);
```

Poi bisogna attendere che la conversione sia finita e che ADSC sia azzerato dall'hardware:

```
while (ADCSRA & (1<<ADSC));
```

Unendo queste ultime due istruzioni a quelle per la selezione ed impostazione dei canali ADC, ad eccezione del *define* BITMASK che va incluso tra le macro, si implementa la seguente funzione di lettura:

```
// Creo la variabile che conterrà la lettura adc.
int adc_read (char ch){

 // Seleziono sequenzialmente i canali 0-7
 ch &= BITMASK;

 // Azzero gli ultimi tre bit di ADMUX e lo imposto con il nuovo canale
// da leggere
 ADMUX = (ADMUX & 0b11111000)|ch;

 // Imposto  ADSC ad 1 e faccio partire la conversione singola
 ADCSRA |= (1<<ADSC);

 // Aspetto che sia finita ta conversione e che ADSC si azzeri
 while (ADCSRA & (1<<ADSC));

 // Leggo il valore adc
 return (ADC);
}
```

Sistemando nel main del programma la BITMASK e le due funzioni si ottiene la seguente sequenza finale:

```
// Definisco la bitmask per la lettura sequenziale dei sette canali adc
// della PORTF
#define BITMASK 0b00000111

// Funzione per inizializzare l'adc
void adc_init(){

 // AREF = AVcc
 ADMUX = (1<<REFS0);

 // Abilito l'ADC, prescaler = 128 (16000000/128 = 125000)
 ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

// Routine per la lettura adc
// Creo la variabile che conterrà la lettura adc
int adc_read(char ch){

 // Seleziono sequenzialmente i canali 0-7
 ch &= BITMASK;

 // Azzero gli ultimi tre bit di ADMUX e lo imposto con il nuovo canale
// da leggere
 ADMUX = (ADMUX & 0b11111000)|ch;

 // Imposto  ADSC ad 1 e faccio partire la conversione singola
 ADCSRA |= (1<<ADSC);

 // Aspetto che sia finita la conversione e che ADSC si azzeri
 while(ADCSRA & (1<<ADSC));
```

```
// Leggo il valore adc
return (ADC);
}
```

Calcolo della conversione

Come detto precedentemente, ogni pin *single-ended* del multiplexer è riferito a GND, ovvero a 0V. Durante la conversione, il segnale analogico in ingresso è confrontato costantemente con una tensione di riferimento. Nelle nostre applicazioni abbiamo selezionato AVCC, che generalmente assume i valori di 3,3V o 5V. La risoluzione del SAR ADC è 10 bit per cui il valore minimo (ovvero il bit 0) rappresenta GND, il valore massimo meno un bit meno significativo (bit 1023) rappresenta la tensione di riferimento sul pin AVCC. Il valore minimo di tensione convertito, ovvero il quanto del modulo ADC, è calcolato con la seguente formula:

$V/(2^{nr.\ bit}) = AVCC/1024$

Quindi, applicando per esempio una tensione di riferimento pari a 5V, V/bit è uguale a

5/1024 = 0,00488V

Supponendo di leggere in uscita alla conversione un valore ADC pari a 256 bit, la tensione in ingresso è uguale a (AVCC/1024) * 256 = 0,00488V * 256 = 1,25V.

Il risultato della conversione viene memorizzato in due registri ad 8 bit chiamati rispettivamente ADCH e ADCL che compongono il *Data Register*, giustificato per impostazione predefinita a destra. L'aggiornamento di questo registro è effettuato leggendo per primo il registro ADCL e successivamente il registro ADCH. Nel caso si necessitasse di una conversione in cui non è richiesta una precisione superiore a 8bit, è sufficiente giustificare il risultato a sinistra e leggere solo il registro ADCH.

Visualizzazione del risultato

Affinché il risultato della conversione possa essere visualizzato esternamente, è necessario dell'hardware aggiuntivo, ad esempio un display LCD oppure un'interfaccia seriale. Si è visto prima che la funzione che legge ogni ingresso analogico e ne memorizza il risultato è:

```
int adc_read (char ch)
```

dove ch assume costantemente i valori compresi tra 0 e 7. Quindi per il canale 0 essa equivale ad adc_read(0), per il canale 1 ad adc_read(1) e così via. Per la visualizzazione di questi risultati si deve prima creare una variabile univoca in cui essi vengono copiati, ad esempio `int adc_result0`, `int adc_result1` ecc., successivamente creare un buffer che

ne ne invii il contenuto, come stringa, all'hardware dedicato, come in queste righe di codice che inviano ad un display LCD il risultato di una lettura fatta sul pin PF0:

```
// Creo la variabile che conterrà il valore adc letto sul canale 0 (PF0)
int adc_result0;

// Array che contiene l'intero da inviare al display lcd
char buffer [10];

// Copio il valore adc letto nel canale 0 in adc_result
adc_result0 = adc_read(0)

// Visualizzo il risultato, convertito in stringa, sull'lcd
itoa (adc_result0, buffer, 10);
lcd_puts(buffer);
```

Programma di esempio

Questo è un semplice esempio di programma che, legge il valore di tensione su un canale dell'ADC al quale è collegato un potenziometro, e lo invia ad un display LCD. Lo schema elettrico con i relativi componenti usati, è riportato in Figura 18.

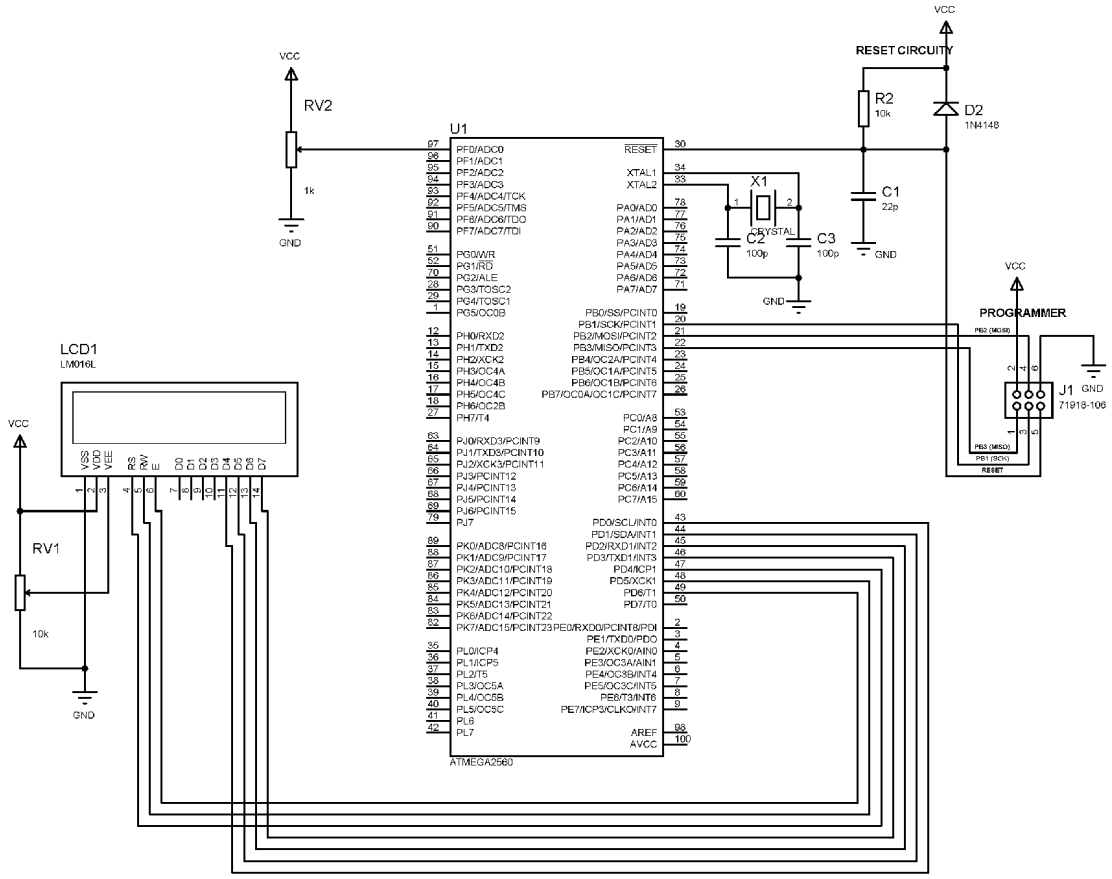

Figura 18: *Schema elettrico dell'esempio.*

```
#pragma config BODLEVEL = BODLEVEL_2V7
#pragma config OCDEN = CLEAR
#pragma config JTAGEN = SET
#pragma config SPIEN = SET
#pragma config WDTON = CLEAR
#pragma config EESAVE = CLEAR
#pragma config BOOTSZ = BOOTSZ_4096W_1F000
#pragma config BOOTRST = SET
#pragma config CKDIV8 = CLEAR
#pragma config CKOUT = CLEAR
#pragma config SUT_CKSEL = SUT_CKSEL_EXTXOSC_8MHZ_XX_16KCK_65MS

// Definisco la frequenza della CPU
#define F_CPU   16000000UL

// Deninisco la bitmask per la lettura sequenziale dei sette canali adc
// della PORTF
#define BITMASK 0b00000111

// Includo i file per la gestione I/O, dei ritardi e dell'lcd
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include "lcd.h"
#include "lcd.c"


// Funzione per initializzare l'adc
void adc_init(){

 // AREF = AVcc
 ADMUX = (1<<REFS0);

 // Abilito l'ADC, prescaler = 128
 // 16000000/128 = 125000
 ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

// ROUTINE PER LA LETTURA ADC
// Creo la variabile che conterrà la lettura
int adc_read(char ch){

 // Seleziono sequenzialmente i canali 0-7
 ch &= BITMASK;

 // Azzero gli ultimi tre bit di ADMUX e lo imposto con il nuovo canale
// da leggere
 ADMUX = (ADMUX & 0b11111000)|ch;

 // Imposto ADSC ad 1 e faccio partire la singola conversione
 ADCSRA |= (1<<ADSC);

 // Aspetto che sia finita ta conversione e che ADSC si azzeri
 while(ADCSRA & (1<<ADSC));

 // Leggo il valore adc
 return (ADC);
}
```

```
int main(void){

    // Creo la variabile che conterrà il valore adc letto dal
    // potenziometro sul canale 0 (PF0) A0
    int adc_result0;

     // Array che contiene l'intero da inviare al display lcd
    char buffer[10];

    // Anitializzo l'adc ed il display lcd
    adc_init();
    lcd_init(LCD_DISP_ON_CURSOR);

   // Scrivo l' etichetta nel display LCD
   lcd_puts("milliVolt = ");
   _delay_ms(50);

 while (1){
       // Leggo il valore adc e lo converto in millivolt
        // (5V/1023= 4,8876)
       adc_result0 = adc_read(0) * 4.8876;

       // Visualizzo il risultato, convertito in stringa, sull'lcd
       itoa(adc_result0, buffer, 10);
       lcd_gotoxy(12,0);
       lcd_puts(buffer);

       if ((adc_result0) < 1000){

          lcd_gotoxy(15,0);
          lcd_puts(" ");
       }

       if ((adc_result0) < 100){
          lcd_gotoxy(14,0);
          lcd_puts("  ");
       }

       if ((adc_result0) < 10){
          lcd_gotoxy(13,0);
          lcd_puts("   ");
       }
 }
}
```

Come si può dedurre da questo esempio, la fase di inizializzazione e la routine di lettura dei canali sono sempre le stesse. Ciò che cambia è il numero delle variabili che conterranno i risultati delle conversioni e quelle in cui questi verranno copiati. Negli esempi successivi, quando verrà trattata anche la periferica USART, si vedrà come visualizzare i dati contemporaneamente sul display LCD e sulla porta seriale. Nella cartella degli esempi sono allegati altri esempi.

Indice Alfabetico

A
ACME....10
ADATE....6, 10
ADC Enable....10
ADCH....11 e seg., 16
ADCL....11 e seg., 16
ADCSRA....6, 10, 13 e seg.
ADCSRB....6, 9
ADEN....10, 13
ADIE....11
ADIF....10
ADLAR....9, 12
ADMUX....8, 10, 14
ADPS....6
ADPS2 - ADPS0....11
ADSC....10, 14
ADTS....6
ADTS0....10
ADTS2....10
Analog Channel and Gain Selection....9
Auto Trigger....6
Auto trigger Enable....10
AVCC....7, 16
B
BITMASK....15
C
CPU....6
D
Data Register....10 e seg.
DIDR0....12
DIDR2....12
E
EMI....7
F
f_adc....13
I
interrupt....10
Interrupt Enable....11
Interrupt Flag....10
L
LCD....16
Left Adjust Result....9
M
MCU....7
MUX0....9
MUX4....9
MUX5....10
N
Nyquist....13
P
PORTF....4, 12
PORTK....4
Prescaler Selection....11
R
Reference Selection....9
REFS0....9
REFS1....9
S
S&H....5
Sample & Hold....5
SAR....4, 16
single-ended....4, 9
SREG....11
Start Conversion....10
T
Trigger....6, 10
U
USART....19
V
VCC....7

History

Data	Versione	Autore	Revisione	Descrizione Cambiamento
16/10/20	1.0	Marcello Pinna	Mauro Laurenti	Versione Originale.