Steve Blackbird (TardoFreak)

PIERIN PIC18 - MISURARE LA TEMPERATURA CON GLI NTC

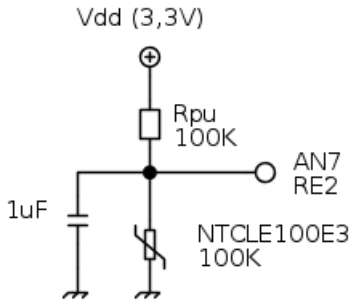
30 June 2013

Introduzione

Quando si uniscono la conoscenza dell' elettronica, la matematica e la possibilità di far funzionare il tutto mediante un microcontrollore si ottengono risultati straordinari. Questo breve articolo, nel suo piccolo, ne è un esempio. Lo scopo è quello di illustrare come interfacciare un sensore analogico non lineare come è l' NTC utilizzando pochissimi componenti e sfruttando le possibilità matematiche del linguaggio C per poter misurare la temperatura.
L' interfacciamento con un sensore di temperatura è un must per le applicazioni didattiche ed hobbistiche. Se è vero che oggi si trovano in commercio una moltitudine di sensori di temperatura integrati è anche vero che un NTC da pochi centesimi può essere un' ottima soluzione per misurare, ad esempio, la temperatura ambiente.
Tutto è nato osservando una scheda di valutazione dove un semplice NTC inserito in un partitore di tensione, permette la misurazione della temperatura ambiente. Con l' aiuto di chi l' elettronica la conosce molto bene l' implementazione è risultata particolarmente semplice.

Il circuito

Il circuito è di una semplicità disarmante.



Ed è anche poco costoso. Il NTC utilizzato è un **NTCLE100E3** da 100K. Il motivo di tale scelta è molto poco poetico e poco ragionato: ce l' avevo in un cassettino. E' un oggetto che ha comunque una discreta precisione, costa quanto mezzo caffè ed è sufficiente per una misura della temperatura ambiente al fine di realizzare, ad esempio, un controllo del riscaldamento.
Purtroppo gli ingressi analogici del PIC montato sul PIERIN presentano un' impedenza d' ingresso

(durante il sample and hold) bassa. Per ovviare a questo inconveniente è stato montato in parallelo allo NTC un condensatore da 1uF poliestere che funge da serbatoio (detto in parole molto povere). Per evitare disturbi si fa una misura ratiometrica, cioè la tensione di riferimento negativa dell' ADC è il GND e quella positiva è la Vdd. Il circuito risulta così insensibile alle variazioni della tensione di alimentazione.
La soluzione ideale è comunque quella di utilizzare un inseguitore di tensione, ne consiglio vivamente l' uso.
Il vero problema sta nel calcolare la temperatura partendo dalla tensione del partitore. Per risolvere questo problema è necessario conoscere bene il funzionamento degli NTC ed i metodi per condizionare il segnale. Una completa trattazione dell' argomento la si può trovare negli articoli scritti da **IsidorKZ**

- **Progetto Termostato I - Proprieta` delle NTC**
- **Progetto Termostato II - Blocchi base**
- **Progetto Termostato III - Dimensionamento e calcoli**
- **Progetto termostato IV - NTC: errori dei modelli e tolleranze**
- **Progetto termostato V: Linearizzazione "ingenua" di una NTC**
- **Progetto termostato VI: Rassegna bibliografica e linearizzabilita`**
- **Progetto termostato VII: Linearizzazione analitica di una NTC**

che mi ha anche suggerito la formuletta magica che permette di risalire alla temperatura misurando la tensione del partitore:

$$T = \frac{298,15 \times B}{298,15 \times \ln\left(\frac{R_{PU}}{R_{NTC}} \times \frac{n}{2^N - n}\right) + B} - 273,15$$

Dove

- 298,15 è il risultato della somma 275,15 + 25 che è la temperatura alla quale il nostro NTC ha valore di resistenza 100K
- B è la costante termica del NTC (espressa in kelvin)
- $\frac{R_{PU}}{R_{NTC}}$ è il rapporto fra la resistenza collegata alla Vdd e la resistenza nominale del NTC a 25°C. Nel nostro caso vale 1 ma per coprire con una buona linearità il range -20°C/50°C dovrebbe valere 1,2. In pratica bisognerebbe usare una resistenza da 120K.
- n è il valore intero letto dall' ADC
- N è la risoluzione in bit dell' ADC. Nel nostro caso è di 10 o 12 bit quindi 2^N vale 1024 o 4096

L' implementazione

Di solito per la misura di temperature tramite NTC vengono utilizzate delle tabelle. In pratica si calcola una volta il valore che dovrebbe dare l' ADC (nel range di temperature da misurare con passi di 0.1° ad esempio), si legge il valore ottenuto dall' ADC, lo si confronta con quello più simile

e si ottiene così la temperatura.
Effettivamente questo è un sistema veloce e che occupa poco spazio in FLASH ma è legato al tipo di NTC ed al circuito. Il C offre la possibilità di eseguire calcoli in virgola mobile ed una libreria standard, la **math.h** per le funzioni matematiche. Utilizzando queste funzioni e facendo i calcoli in virgola mobile possiamo così essere svincolati da circuito e dal tipo di NTC visto che la formula è di tipo generale.

La lettura dall' ADC

Come mi è stato suggerito da chi l' elettronica la conosce non ho fatto una sola lettura dall' ADC ma ho implementato una sorta di filtro digitale a pettine facendo 20 letture con cadenza 1ms. In questo modo ho suddiviso il periodo della frequenza di rete in 20 istanti dove ho fatto altrettante letture. Così facendo ho ridotto l' errore dovuto alle interferenze della frequenza di rete eliminando anche la seconda, la quarta e la quinta armonica. In pratica ho sommato tutte le letture, diviso il risultato per venti (divisione float in modo da portarmi anche i decimali) ed ho utilizzato il risultato ottenuto per calcolare la temperatura
Ho anche utilizzato una simpatica caratteristica dell' ADC: l' autocalibrazione. In effetti, prima di procedere con le venti letture, ho dato un comando di calibrazione all' ADC in modo da eliminare gli offset interni.

Tempi

Uno degli interrogativi era: quanto tempo ci mette il micro a calcolare la temperatura? Ovviamente ci sono i 20 ms dovuti alle letture (anche se la singola lettura impiega 45us) e poi c'è il calcolo matematico che dura 800 us. A questo punto non vale neanche la pena di pensare a sviluppi in serie per calcolare il logaritmo.

Spazio in FLASH

Con questo programma non ho badato a spese, in termini di occupazione di FLASH. Infatti ho utilizzato la libreria **stdio.h** per poter utilizzare la funzione **sprintf** (che non è proprio piccolina), la libreria **math.h** per il logaritmo ed il pacchetto di matematica in virgola mobile. Il risultato è un programma lungo 10015 bytes che corrisponde ad un' occupazione del 7% della FLASH disponibile. E' bene però ricordarsi che queste librerie, una volta incluse nel progetto, possono essere usate più volte quindi potrei benissimo fare altri calcoli e manipolare stringhe senza che questo mi vada ad occupare una quantità esagerata di memoria. Le subroutines ci sono già, ed il programma può chiamarle più volte.

Il programma

Per scrivere il programma sono partito dal progetto che ho scritto per utilizzare il display LCD alfanumerico. Ho aperto il progetto, l' ho salvato con un altro nome in un' altra cartella e ci ho aggiunto un po' di codice, molto poco a dire il vero. Oltre ai comandi per l' ADC le uniche cose di

rilevo sono state introdurre un flag nella interrupt ciclica per la sincronizzazione ed una opzione nella configurazione del micro. Le vediamo una per una.

File di configurazione

L' unica variante che ho introdotto è la seguente

```
// Seleziona la risoluzione del convertitore A/D
//#pragma config ADCSEL = BIT12
//#define ADC_12BIT
```

che impone al convertitore una risoluzione di 12 bit. Come potete notare la direttiva **#pragma** e la **#define** sono messi come commenti. Questo perché se è vero che con il bootloader si possono anche programmare i bit di opzione è anche vero che **questa pratica non è priva di pericoli!** La sperimentazione con gli NTC si può fare benissimo con la conversione a 10bit senza correre il rischio di danneggiare il bootloader del PIERIN, cioè senza scriverei bit di configurazione.
Chi invece utilizza il PIERIN con il PicKit può fare quello che più gli aggrada e quindi "scommentare" le due direttive e lavorare a 12 bit.

Tipi standard

Subito dopo le inclusioni, fra le quali ho inserito

```
#include <stdio.h>
#include <math.h>
```

ho voluto inserire una serie di typedef per assegnare i nomi che di solito uso io nei miei programmai per identificare i vari tipi numerici

```
typedef unsigned char       uint8_t;
typedef unsigned short int uint16_t;
typedef short int           int16_t;
typedef unsigned long int  uint32_t;
typedef long int            int32_t;
```

Li ho scritti solo per una questione di abitudine e di comodità. Se ho bisogno di un intero a 16 bit senza segno lo dichiaro come **uint16_t** invece di scriverci un poema del tipo "unsigned short int". Notare la "**_t**" finale. E' una convenzione che viene utilizzata sovente per indicare che il simbolo in questione è uno specificatore di tipo.

Interrupt ciclica

Anche in questo programma ho la mia bella interupt ciclica a cui sono particolarmente affezionato. Come al solito contiene la gestione del timer software ma questa volta chi ho aggiunto un flag che mi serve per la sincronizzazione che ho dichiarato nelle variabili globali come

```
volatile uint8_t sincronismo;
```

subito dopo la variabile che funge da timer software. La funzione è questa

```
void lowPriorityInterrupt()
{
  // Verifica quale flag ha causato l' interrupt
  // Esegui la parte di codice di servizio dell' interrupt
  // Azzera il flag che ha causato l' interrupt
  // ...
  // Gestione dell' interrupt del timer 2
  if(PIR1bits.TMR2IF)
  {
    // gestione del flag di sincronismo
    if (sincronismo) sincronismo = 0;

    // gestione del timer software. Il timer software deve decrementarsi
    // fino ad arrivare a 0. Una volta arrivato a 0 resta fermo a 0.
    if (timer_delay) timer_delay--;

    // Resetta il flag che ha generato l' interrupt
    PIR1bits.TMR2IF = 0;
  }
}
```

e come si può vedere è identica a quella solita se non per la presenza dell' istruzione

```
if (sincronismo) sincronismo = 0;
```

Questa istuzione (o meglio la variabile coinvolta) mi serve per sincronizzarmi con l' interrupt. Infatti se do un valore qualsiasi alla variabile "sincronismo" e rimango in attesa che diventi zero sono sicuro di aver intercettato l' istante dopo l' esecuzione della interrupt (che ha riportato la variabile a zero). Mi sono quindi sincronizzato con la interrupt e questo mi servirà più avanti per l' acquisizione dal ADC.

Le costanti

La costante B del NTC ed il rapporto del partitore sono dichiarati all' inizio del programma.

```
rom const float NTC_Bconst = 4190;
rom const float NTC_partRatio = 1;
```

La costante B si legge dal datasheet del NTC mentre il rapporto del partitore lo si calcola facendo il rapporto

$\frac{R_{PU}}{R_{NTC}}$
Che in questo caso vale 1 essendo la $R_{PU} = 100K\Omega$ e la $R_{NTC} = 100K\Omega$ a 25°C

Il main

Tralasciando l' inizializzazione del timer (già spiegata in **questo articolo**) incontriamo l' inizializzazione dell' ADC. Anche per l' ADC ho scritto la funzione di de-inizializzazione.

```
// Inizializza tutti i pin come digitali
ANCON0 = 0xFF;
ANCON1 = 0x1F;
// Inizializza il pin AN7 (RE2) come ingresso analogico
ANCON0bits.PCFG7 = 0;  // 12/08/2013 corretto BUG segnalato da c1b8

// De-inizializza l' ADC
adc_deInit();
// Seleziona Vref positiva Vdd
ADCON0bits.VCFG0 = 0;
// Seleziona Vref negatica Vss
ADCON0bits.VCFG1 = 0;
// Seleziona il tempo di conversione 20TAD
ADCON1bits.ACQT = 7;
// Clock di conversione FOSC/64
ADCON1bits.ADCS = 6;
// Formato del risultato allineato a destra
ADCON1bits.ADFM = 1;
```

E questo è il ciclo infinito di di funzionamento

```
for(;;)
{
  char s[20];
  int32_t valore;
  uint16_t gradi, decimi;
  float temp, valf;
  uint8_t letture;


  // Calibrazione ADC
  ADCON1bits.ADCAL = 1;
  ADCON0bits.GO = 1;
  while(ADCON0bits.GO);
  ADCON1bits.ADCAL = 0;
```

```
  // Seleziona l' ingresso da misurare
  ADCON0bits.CHS = 7;

  // Effettua 20 letture in 20 ms.
  valore = 0;
  for (letture = 0; letture < 20; letture++)
  {
    // Sincronizzazione con interrupt ciclica
    sincronismo = 1;
    while(sincronismo);
    // Fa partire la conversione 45 us.
    ADCON0bits.GO = 1;
    // Aspetta la fine della conversione
    while(ADCON0bits.GO);
    // Legge il valore convertito
    valore += ADRES;
  }

  valf = (float) valore / 20;

  // Calcola la temperatura 800 us.
  #ifdef ADC_12BIT
    temp = (298.15*NTC_Bconst/(298.15*log(valf/(4096-valf)*NTC_partRatio)
            +NTC_Bconst)-273.15)*10;
  #else
    temp = (298.15*NTC_Bconst/(298.15*log(valf/(1024-valf)*NTC_partRatio)
            +NTC_Bconst)-273.15)*10;
  #endif

  valore = temp;
  gradi = valore / 10;
  decimi = valore % 10;

  sprintf(s,"Temperatura %2d.%1d",gradi,decimi);
  LCD_setPos(2,2);
  LCD_writeStr(s);
  timer_delay = 500;
  while(timer_delay);

}
```

Le variabili che uso per questo programma sono dichiarate a livello di blocco. Questo perché servono solo qui e quindi è inutile dichiararle a livello di funzione (main) o come globali, anche se lo si può fare. In sequenza troviamo queste operazioni:

- La calibratura dell' ADC. Questa è una bellissima caratteristica del microcontrollore e vale la pena di utilizzarla visto che è veramente semplice. E' sufficiente infatti mettere il bit ADCAL a 1, fargli fare una conversione e poi rimetterlo a zero per ritrovarsi l' ADC calibrato. Un applauso al PIC18F47J53.
- La selezione dell' ingresso analogico su cui fare la misurazione.
- Un ciclo di 20 letture. Prima di ogni lettura si aspetta che il flag **sincronismo** sia messo a zero dalla interrupt ciclica per poi procedere con la lettura. Avremo così la certezza di essere (quasi perfettamente) sincronizzati con la temporizzazione e quindi fare le letture con cadenza di 1 ms.
- La media delle letture viene calcolata in float in modo da avere la media vera e non approssimata ad una unità.
- Il calcolo della temperatura applicando la formula che ho descritto all' inizio e la formula viene selezionata (a seconda che si lavori con il convertitore a 12 bit o a 10bit) tramite compilazione condizionata. Da notare il ***10** alla fine. Questo l' ho introdotto per avere il risultato in decimi di grado.
- La visualizzazione su display LCD
- L' attesa di mezzo secondo prima di procedere con una nuova elaborazione.

Utilizzare più NTC

In questo esempio ho utilizzato un solo ingresso per un solo NTC ma nulla vieta di utilizzare più ingressi collegati a più NTC. In effetti si potrebbe pensare ad una applicazione che richiede la misura della temperatura ambiente di diverse stanze, la temperatura esterna e, crepi l' avarizia (di ingressi analogici ce ne sono molti) anche le temperature dell' acqua in ingresso e in uscita dalla caldaia. In tal caso, qualora ce ne fosse bisogno, la procedura per le venti acquisizioni e l' equazione per il calcolo della temperatura potrebbero essere inserite in altrettante funzioni da richiamare per ogni sensore fornendo alla funzione di calcolo il parametro B (non è detto che gli NTC siano tutti dello stesso tipo) ed il rapporto del partitore (che può variare da sensore a sensore).

Conclusioni

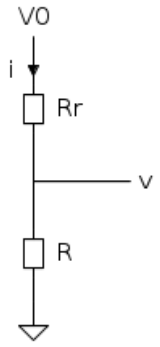
In conclusione possiamo dire che **la matematica è quella che risolve i problemi** ed il microcontrollore è il cosiddetto "braccio armato" che può servire per trasformare le soluzioni matematiche in un qualcosa di reale e funzionante. Un NTC "della mutua" (come si dice a Torino) o "della baiona" da 40 centesimi, un condensatore, una resistenza all' 1% e una giusta dose di matematica sono sufficienti per per realizzare un termometro digitale. Ovviamente la precisione è quella che è, dipende dal NTC. Utilizzando un NTC migliore e tarando il sistema si possono fare misure decisamente più precise.
Abbiamo anche conosciuto le potenzialità del convertitore AD ed il modo in cui utilizzarlo ed anche

le potenzialità del linguaggio C quando bisogna fare calcoli in virgola mobile, e le potenzialità della libreria standard math.h
Direi che a questo punto di esempi e di blocchi funzionali ce ne sono a sufficienza per intraprendere una sperimentazione interessante e divertente con il PIERIN. Ora possiamo gestire un display LCD, ritardi, leggere temperature e tensioni, utilizzare la RS232 e, se a questo ci aggiungiamo degli ingressi e delle uscite di vario tipo (relè, driver per LED e compagnia bella) si può già pensare di realizzare qualcosa di gustoso, soprattutto facendo fare al microcontrollore il suo lavoro, e cioè controllare.
Ciò detto, scusandomi per eventuali errori o imprecisioni che avrei piacere mi fossero fatte notare, non mi resta che augurare ancora una volta **BUONA SPERIMENTAZIONE!**

I sorgenti del programma sono all' interno del file **pierin_tamb.rar**

Addendum

Riporto qui di seguito la spiegazione di **DirtyDeeds** dell' equazione che permette di calcolare la temperatura mediante la lettura della tensione del partitore.



Abbiamo che la resistenza incognita R è la resistenza dell'NTC. La relazione tra tale resistenza e la temperatura termodinamica T del sensore può essere scritta in modo approssimato come

$$R = R_0 e^{B\left(\frac{1}{T} - \frac{1}{T_0}\right)}$$

dove R_0 è la resistenza dell'NTC alla temperatura T_0 (potrebbe essere una temperatura qualunque, ma tipicamente si prende T0 = 298,15 K, corrispondenti a 25 °C) e B è un parametro, chiamato *temperatura caratteristica*, specifico del materiale componente l'NTC.

Determinando R a partire dall'equazione

$$R = R_r \frac{n}{2^N - n}$$

si ha

$$R_\mathrm{r}\frac{n}{2^N-n} = R_0 \mathrm{e}^{B\left(\frac{1}{T}-\frac{1}{T_0}\right)}$$

ovvero

$$\frac{R_\mathrm{r}}{R_0}\frac{n}{2^N-n} = \mathrm{e}^{B\left(\frac{1}{T}-\frac{1}{T_0}\right)}$$

Prendendo il logaritmo di ambo i membri si ottiene

$$\ln\left(\frac{R_\mathrm{r}}{R_0}\frac{n}{2^N-n}\right) = \frac{B}{T}-\frac{B}{T_0}$$

da cui

$$T = \frac{BT_0}{T_0\ln\left(\frac{R_\mathrm{r}}{R_0}\frac{n}{2^N-n}\right)+B}$$

T è una temperatura termodinamica: poiché siamo interessati a una temperatura Celsius, bisogna sottrarre 273,15 K:

$$t = \frac{BT_0}{T_0\ln\left(\frac{R_\mathrm{r}}{R_0}\frac{n}{2^N-n}\right)+B} - 273{,}15\,\mathrm{K}$$

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Tardofreak:pierin-pic18-misurare-la-temperatura-con-gli-ntc"