**Laboratorio 1 – ArduinoTemp**

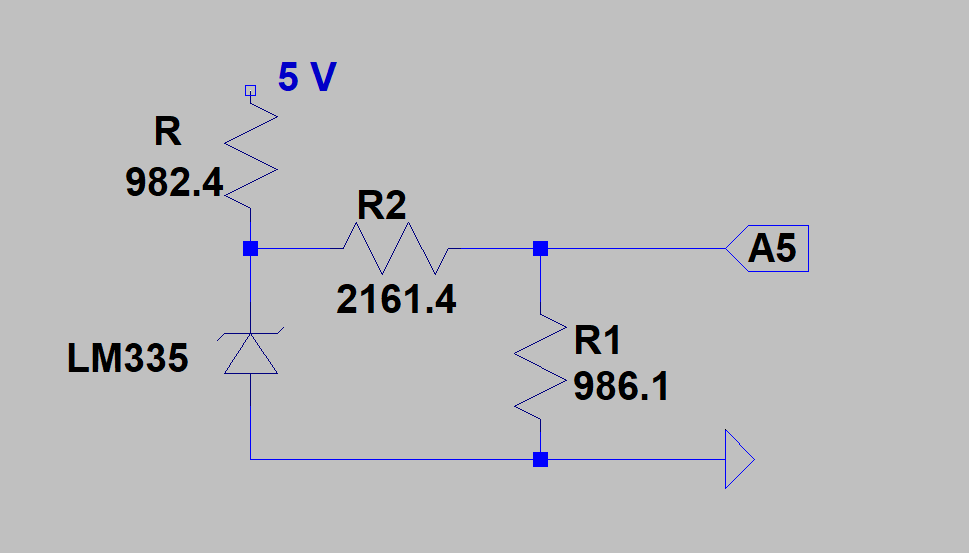
L’obiettivo del laboratorio 1 è la realizzazione di un termometro minimale impiegando un sensore di temperatura elettronico LM335 e una scheda Arduino Uno (la nostra è la numero 26).

Come prima cosa misuriamo la tensione di riferimento interna Vref di Arduino, dichiarata pari a 1.1 V ± 10% dal costruttore, tramite il multimetro HP 34401A; questa tensione dobbiamo poi confrontarla con il valore salvato all’interno della EEPROM di Arduino, per verificare se c’è stata una deriva nel tempo.

Come risultato della nostra misurazione leggiamo (1.09734 ± 0.00005) V; per questa misurazione abbiamo posto sul percorso della tensione un condensatore da 220 nF in modo da rendere più stabile il segnale, come consigliato dal datasheet dell’ATmega328/P.

Risaliamo al valore salvato all’interno della EEPROM tramite una struct CalCalculations, instanziando l’oggetto calibration (del tipo struct appena scritto) e tramite il comando EPROM.get(0,calibrations) troviamo che Vref = 1.0977 V. Non abbiamo informazioni sull’incertezza con cui è stato calcolato questo valore, in ogni caso nel calcolo della temperatura utilizzeremo poi il valore della tensione di riferimento misurato da noi, poiché si riferisce a quella del dispositivo nelle stesse condizioni in cui poi verrà utilizzata.

Come sensore utilizziamo un LM335 come detto precedentemente; quest’ultimo ha una sensibilità di 10mV/°C e zero allo zero assoluto. Il sensore si comporta come uno zener e possiamo utilizzarlo in un circuito molto semplice.

Alimentiamo il sensore tramite l’uscita a 5V di Arduino, che è caratterizzata da variazioni fino a 10%; questa tensione però non va bene come riferimento per l’ADC in quanto queste variazioni provocano un’incertezza troppo alta sulla misura della temperatura. Per ovviare a questo problema usiamo la tensione di 1.1 V di riferimento misurata precedentemente con il multimetro.

Questa tensione di riferimento è fissa. A questo punto è poi necessario un partitore per portare la tensione del sensore entro la dinamica dell’ADC.

Vediamo prima di tutto quanta tensione cade sul sensore: la temperatura nel laboratorio è ~26 °C (299.15 K) al momento del calcolo ed equivale ad una tensione sul sensore di circa 3 V. Per essere precisi possiamo utilizzare un worst case considerando la temperatura a 40°C per dimensionare le R in modo che il sensore venga polarizzato correttamente. Infatti il sensore ha bisogno di una corrente di almeno 500 nA per poter funzionare e questa corrente di polarizzazione non può nemmeno essere troppo alta in quanto surriscalderebbe il sensore aumentando l’incertezza della sua misura in maniera non accettabile.

Tutte le resistenze usate sono misurate con il multimetro quindi hanno l’incertezza del multimetro.

Supponiamo di usare R = (982.4 ± 0.1) Ω. Nel caso peggiore qui mettiamo resistore da 680 Ω per consentire a sensore di essere polarizzato (? Necessario ?).

I valori dei resistori del partitore sono R1 = (986.1 ± 0.1) Ω e R2 = (2.1614 ± 0.0003) kΩ. Scegliamo un valore né troppo alto per non aumentare troppo la resistenza in ingresso all’ADC (che deve essere inferiore a circa 10 kΩ), né troppo basso per non assorbire tutta la corrente che scorre in R.

Per la lettura usiamo come pin il 5 che non ha diodi di protezione, in questo modo non dobbiamo considerare l’impedenza d’ingresso dell’adc che si aggira intorno alle decine di G Ω. Il diodo di protezione posto in parallelo all’ingresso ha infatti l’effetto di abbassare l’impedenza.

Scriviamo un semplice programma in Arduino per calcolare la temperatura:

void setup() {

Serial.begin(9600); //apriamo una linea seriale con baud rate=9600

analogReference(INTERNAL);//impostiamo il riferimento interno vRef per l’adc

int prova=analogRead(A5); //eseguiamo una lettura di prova

delay(10);

}

void loop() {

unsigned int N=analogRead(A5); //leggiamo il valore di tensione sul pin A5

float tensione\_VREF=1.0972;

float H=0.3133032621;

int K=100;

float temperatura = (tensione\_VREF / 1024 \* N /H) \* K - 273.15;

//nella linea precedente calcoliamo direttamente il valore della temperatura

//per poi inviarlo sulla linea seriale.

Serial.println("temperatura calcolata con arduino= ");

Serial.println(temperatura, 7); //inviamo il dato calcolato precedentemente

Serial.println("ADC= ");

Serial.println(N);//inviamo il valore letto dall’adc per una ulteriore verifica

delay(2000); //impostiamo un attesa di 2 secondi tra due letture consecutive

}

Come si può notare abbiamo usato diverse variabili float. Benché queste richiedano più memoria per essere salvate ed elaborate e di conseguenza aumentino il tempo di calcolo abbiamo deciso di usarle comunque, vista la semplicità dell’operazione richiesta e l’assenza di requisiti temporali da rispettare. La formula utilizzata è la seguente: H, valore del partitore, è uguale a R1/(R1+R2), N è il valore decimale del dato letto dall’adc e L e la sensibilità del sensore (uguale a 100 nella formula utilizzata). Abbiamo fatto eseguire questa formula direttamente ad Arduino.

Eseguiamo 286 misure, con 2 secondi di attesa tra una misura e quella successiva. Siccome la variazione di temperatura in una stanza è ragionevolmente lenta e non rilevabile da un sensore come quello utilizzato, non ci aspettiamo una dispersione significativa dei dati misurati, salvo derive dovute a fattori che sono stati comunque considerati non influenti.

Notiamo come la maggior parte delle temperature misurate durante i 572 secondi di esecuzione del programma oscillino tra due valori: 28.14° e 28.49°. Non sono presenti derive significative.

Per costruire questo grafico abbiamo tenuto conto del valore letto dall’adc e non di quello calcolato tramite la formula, sull’asse verticale sono riportate le temperature associate a tali codici.

La temperatura misurata tramite il termo igrometro, durante le rilevazioni eseguite con Arduino, è di 28.1 °C.

Ricordiamo che nel calcolo dell’incertezza abbiamo ipotizzato nullo l’influenza dei cavi e dei contatti, l’influenza del surriscaldamento delle resistenze e degli effetti elettromagnetici.

Avessimo voluto diminuire l’effetto dell’incertezza dovuto al surriscaldamento del sensore a causa della tensione continua in uscita da Arduino, avremmo potuto utilizzare un pin digitale e mandare in uscita un’onda quadra con DC = 10% e misurando solamente durante in questi impulsi. In questo modo avremmo diminuito l’effetto del surriscaldamento [quest’ultima parte è da rivedere se volessimo inserirla davvero all’interno della relazione].

Come ultima cosa abbiamo calcolato l’incertezza della nostra misura di temperatura.

Gli elementi che introducono quest’incertezza sono H, N e Vref, mentre l’effetto del sensore può essere valutato come un offset.

L’incertezza, calcolata secondo il modello deterministico, è la somma dei seguenti termini:

* Contributo incertezza tensione di riferimento interna ;
* Contributo incertezza conversione adc ;
* Contributo incertezza resistenza R2 ;
* Contributo incertezza sensoresensore;
* Contributo incertezza resistenza R1.

Le varie incertezze assolute sono:

* R1=0.1Ω;
* R2=0.3Ω;
* Vref=0.00005V
* sensore = 6 (MAX uncalibrated temperature error);
* N= 2LSB.

L’errore complessivo dell’adc di Arduino con una frequenza di campionamento inferiore a 200kHz è di 2 LSB.

Segue che pertanto prendendo la misura di temperatura come la media pesata dei vari campioni, cioè 28.34° possiamo dichiarare la misura finale come T= (28.3 ±6.7) °C.

**Laboratorio 3 - ArduinoTemp in connessione a PC**

L’obiettivo del laboratorio 3 è realizzare in C# un’interfaccia PC per il termometro elettronico con Arduino Uno che permetta di: eseguire una misura solo se richiesto, tramite l’uso di un bottone apposito; calcolare la temperatura con la relativa incertezza e visualizzarla; prevedere un sistema di taratura a una temperatura definibile dall’utente stesso, con eventuale correzione dell’algoritmo per calcolare incertezza e temperatura dell’ambiente.  
Scriviamo un programma Arduino che permetta di eseguire una misura solo se richiesto, ossia solo se l’utente, tramite la linea seriale che collega il microcontrollore al PC, invii uno specifico comando.

Lo schema circuitale del termometro è lo stesso del Laboratorio 1; cambiano, seppur di poco, i valori delle resistenze: R1 = (984.16 ± 0.11) Ω, R2 = (2170.22 ± 0.23) kΩ e R = (983.82 ± 0.11) Ω.

Scriviamo il programma Arduino:

#programmaArduino, quello caricato presenta variabili globali quali resistenze ecc… che possono essere leiminate, oltre a commenti e zozzerie varie…

#N.B il programma C# INSERITO PRESENTA DELLE CORREZIONI RISPETTO A QUELLO CARICATO SU DROPBOX! TENETENE CONTO QUALORA IN FASE DI FORMATTAZIONE LO SI DEBBA MODIFICARE!!!

Durante l’esecuzione dell’algoritmo, Arduino misura e invia su richiesta una stringa contenente il carattere # immediatamente seguito dalla tensione ai capi di R1, quest’ultima espressa in LSB: il simbolo # segna l’inizio dei caratteri “buoni” da elaborare, consentendo di scartare eventuali messaggi precedenti inviati sulla linea seriale dal boot loader, che comprometterebbero il corretto funzionamento del programma. L’elaborazione del dato viene fatta in C# attraverso la funzione chiamata processThisLine, sfruttando così la potenza di calcolo del PC, piuttosto che quella notevolmente più limitata di Arduino.

Scriviamo l’interfaccia C#:

#foto dell’interfaccia

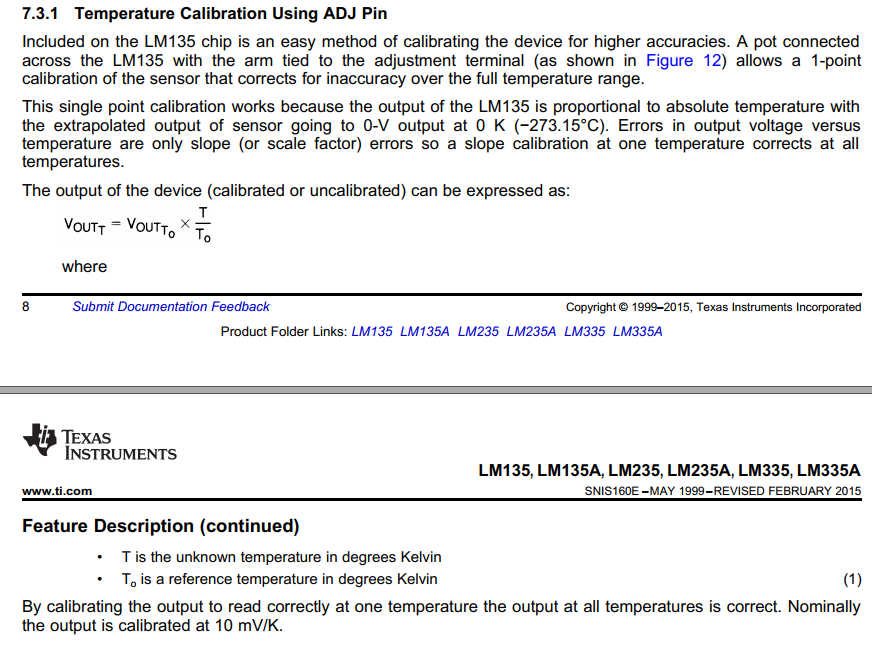
L’interfaccia consta di 3 pulsanti e di 3 caselle di testo: buttonOpenClose, per aprire o chiudere la gestione seriale e per abilitare o disabilitare i pulsanti buttonSend e buttonCal; buttonSend, per chiedere ad Arduino di eseguire una misura della temperatura; buttonCal, per calibrare il sensore a una temperatura a scelta dell’utente; textBoxDataReceived, dove viene stampato l’esito della misura; textBoxMessage, dove è possibile controllare progressi e stato dell’algoritmo di misurazione o calibrazione; texBox\_temp, dove l’utente può inserire la temperatura, espressa in K, a cui calibrare lo strumento.

La gestione della linea seriale è implementata facendo uso di un delegate, necessario per evitare operazioni “cross thread”: deleghiamo il compito di aggiornare l’interfaccia grafica a una procedura all’interno del thread grafico, AppendATextCrossThread.

Senza alcuna calibrazione, il valore numerico di temperatura e l’incertezza a esso relativa vengono calcolati entrambi in modo analogo a quanto fatto nel corso del laboratorio 1. Dopo la prima calibrazione, la temperatura T ± T verrà calcolata nella maniera seguente (adc\_taratura è il valore letto dall’ADC di Arduino durante la calibrazione; temp\_taratura è la temperatura alla quale viene eseguita la calibrazione, inserita dall’utente; adc è il valore letto dall’ADC durante la misura):

* ;
* T = = ;

L’incertezza relativa alla T\_taratura è quella del termoigrometro GFTH 95 disponibile in laboratorio; l’incertezza del sensore è pari a 2°C (MAX calibrated temperature error). La calibrazione del sensore è realizzata in accordo a quanto scritto sul datasheet del sensore, sul quale leggiamo che gli errori sula tensione in uscita fornita dal dispositivo non sono errori di offset, ma errori di guadagno. È possibile calibrare lo strumento diversamente, trattando gli errori all’uscita come errori di offset: tale opzione potrebbe essere presa in considerazione qualora lo strumento venga sempre calibrato a temperature sufficientemente maggiori rispetto allo 0K; in tali condizioni l’errore di guadagno può essere trattato con buona approssimazione come un errore di offset. Di seguito, uno screenshoot della parte del datasheet del sensore inerente alla sua possibile calibrazione.



Valgono le stesso considerazioni sul modello di misurazione fatte durante il Laboratorio 1.