Antenna analyzer - Misuratore impedenza antenna

Rev.0 del 16/9/2010 di Marco Ducco IK1PXM

Ho progettato e realizzato, con il minor impegno necessario, un dimostratore del funzionamento di un misuratore di impedenza che utilizza per i calcoli e il pilotaggio display una scheda Arduino 2009. Lo chiamo dimostratore, in inglese sarebbe un “breadboard” montaggio sperimentale, non ancora un prototipo in quanto non ha neppure lontanamente l'aspetto fisico-meccanico di un apparato finito.

Il misuratore si basa sul metodo della misura delle tre tensioni, più diffuso di quelli a ponte da quando sono disponibili i microcontrollori per svolgere i calcoli, è usato nell' antenna analyzer commerciale MFJ-249 e nei modelli successivi e nell'ottimo aerial analyser di VK5JST e distribuito in kit per meno di 100 Euro.

Se il mio scopo fosse acquisire un buon analizzatore di impedenza a basso prezzo, comprerei il kit australiano; tuttavia, il mio interesse e divertimento è studiare un problema risolvibile con la programmazione di microcontrollori e, aggiungendo un poco di elettronica di contorno, realizzare qualcosa che ne mostra una possibile soluzione.
Descrivo lo schema e il programma applicativo per permettere di procedere a terminare il progetto e alla costruzione di un prototipo, ed eventualmente passare a una serie facilmente replicabile.
Potrei fornire assistenza a distanza, l'aggiornamento del SW per eventuali correzioni e migliorie, e se proprio indispensabile, il microcontrollore già programmato; il mio e.mail è riportato nello schema elettrico.

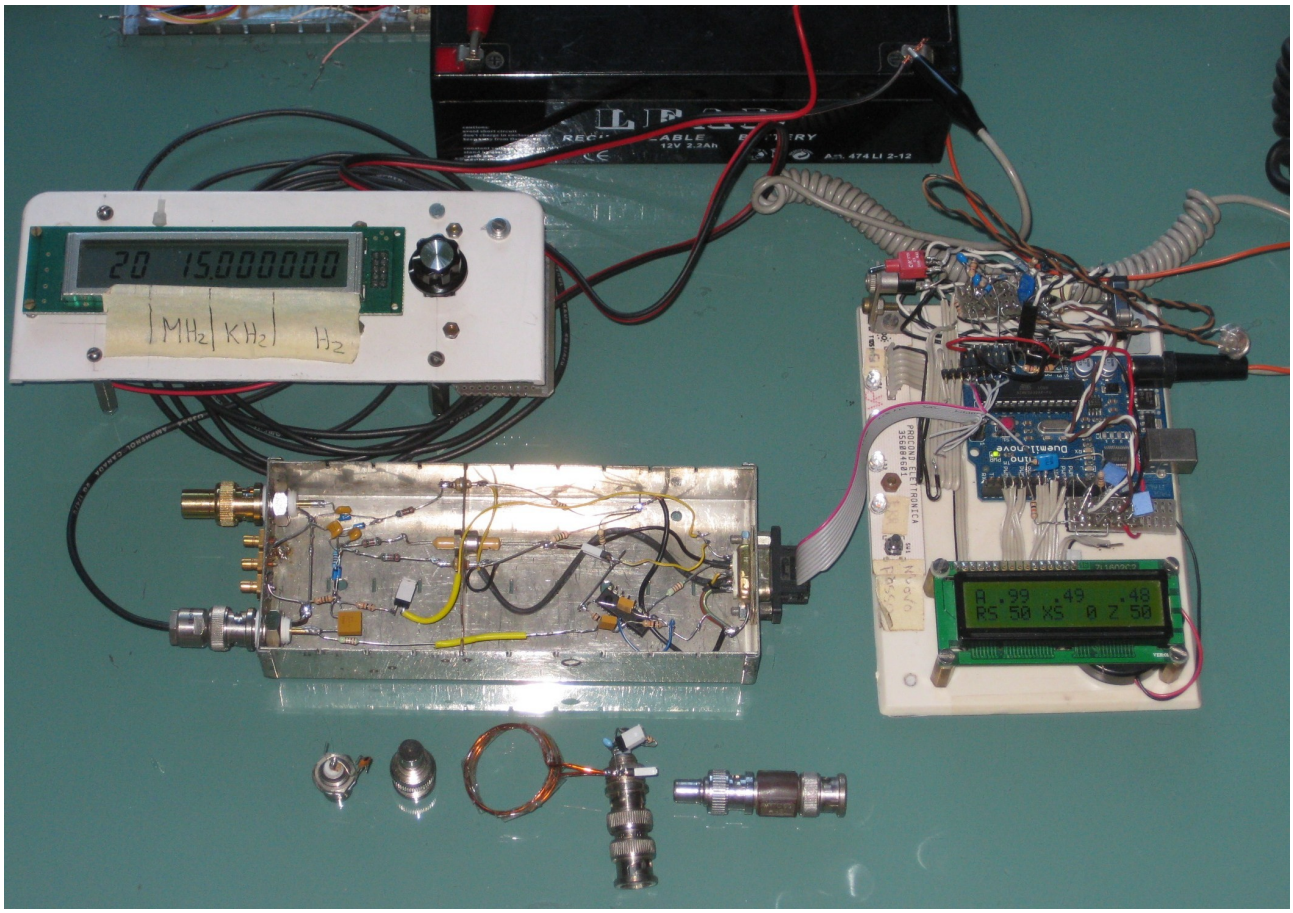
Il dimostratore ha prestazioni elettrico funzionali simili a quelle degli strumenti precedentemente citati, ma come generatore di segnale utilizza il sintetizzatore che avevo a disposizione. Detto sintetizzatore, costituito del kit DSPLL di K5BCQ, basato sul Si570 con un proprio display e comando frequenza (ringrazio Pino IK1JNS per l'approvvigionamento e Massimo IK1XPD per montaggio, diagnosi e riparazione), ha una risoluzione inutilmente elevata e una uscita ad onda quadra che deve venire filtrata per trasformarla in sinusoidale. Per filtrarla ho usato un filtro passabasso del 5° ordine con frequenza di taglio a 6dB di circa 37 MHz, permette di rendere sinusoidale il segnale quando il segnale è nella banda 15 - 30 MHz di mio interesse (Sotto i 15 MHz non attenua più a sufficienza la terza armonica a 45 MHz, sopra i 30 inizia ad attenuare la fondamentale).
Come generatore si può anche adoperare l'RTX di stazione, commutato su AM o FM con potenza al minimo, con un opportuno attenuatore in uscita in modo da erogare al massimo 0,5 W.
Per una soluzione definitiva suggerirei un oscillatore libero ad ampia escursione di frequenza, oppure un sintetizzatore con uscita sinusoidale comandato dallo stesso microcontrollore.

Prestazioni preliminari del misuratore:
Campo di misura: Resistenza da 10 a 200 Ω, Reattanza da 0 (5) a 200 Ω, precisione 5-10% del valore misurato. Come gli altri strumenti NON misura il segno della reattanza.
Frequenza: da 3 a fino a 150 Mhz, dipende dalle modalità del cablaggio e della meccanica che influiscono sui parametri parassiti del circuito.

Non ho ancora applicato il misuratore a una antenna; le misure su una antenna poss0no essere affette da maggiori errori perché l'antenna può captare tensioni indotte di frequenza anche diversa da quella di interesse (emittenti FM, onde medie, disturbi) che si sommano a quella del generatore

creando valori massimi di tensione errati.
Se non si possono eliminare i disturbi, si possono neutralizzare aumentando la tensione del generatore. Con una Ra da 50 Ω ¼ W, la massima tensione sopportabile è V = SQRT(0.25*50)=3,5 Veff cautelativamente corrispondenti ai 5 Vdc massimi misurabili dai circuiti di ingresso del uC.
Si può aumentare la potenza mettendo resistori più grandi, e inserendo partitori all'ingresso degli ingressi analogici; ma maggiore potenza implica maggiore assorbimento delle batterie, maggiori dimensioni e quindi uno strumento meno maneggiabile.

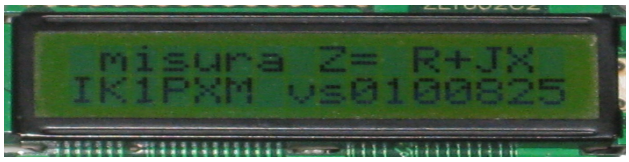
Un circuito come quello dei Vector Network Analyzer che funziona sulla misura della componente del valore efficace sintonizzato sulla frequenza del segnale, funziona meglio perchè è suscettibile solo ai disturbi prossimi alla frequenza di interesse, ma è alquanto più complicato da realizzare.



Nella foto si osservano:
-Il generatore sintetizzato con la frequenza impostata su 15 MHz.
-La scatola metallica con il circuito dell'apparato, come carico in misura ha un tappo BNC da 50 Ω, le connessioni della scatola con la scheda sono sezionate da un connettore a vaschetta a 9 poli.
-Una base in plastica contenente il cablaggio della scheda Arduino 2009 con il display e la scatola. Si intravede anche il circuito di un decodificatore morse che non ho rimosso per comodità.
-Una serie di carichi resistivi e capacitivi e un circuito RLC usati per provare lo strumento.

Visualizzazioni sul display

Si passa da una visualizzazione alla successiva premendo il pulsante Ps1



Iniziale per 2 secondi all'accensione

Tensioni VG,VR,VC; Resistenza, Reattanza serie, Impedenza

Tens. VG,VR,VC; Resist., Reatt. parall. equivalenti, Imp.

Tensioni VG,VR,VC; Coeff. Gamma rifles. e SWR

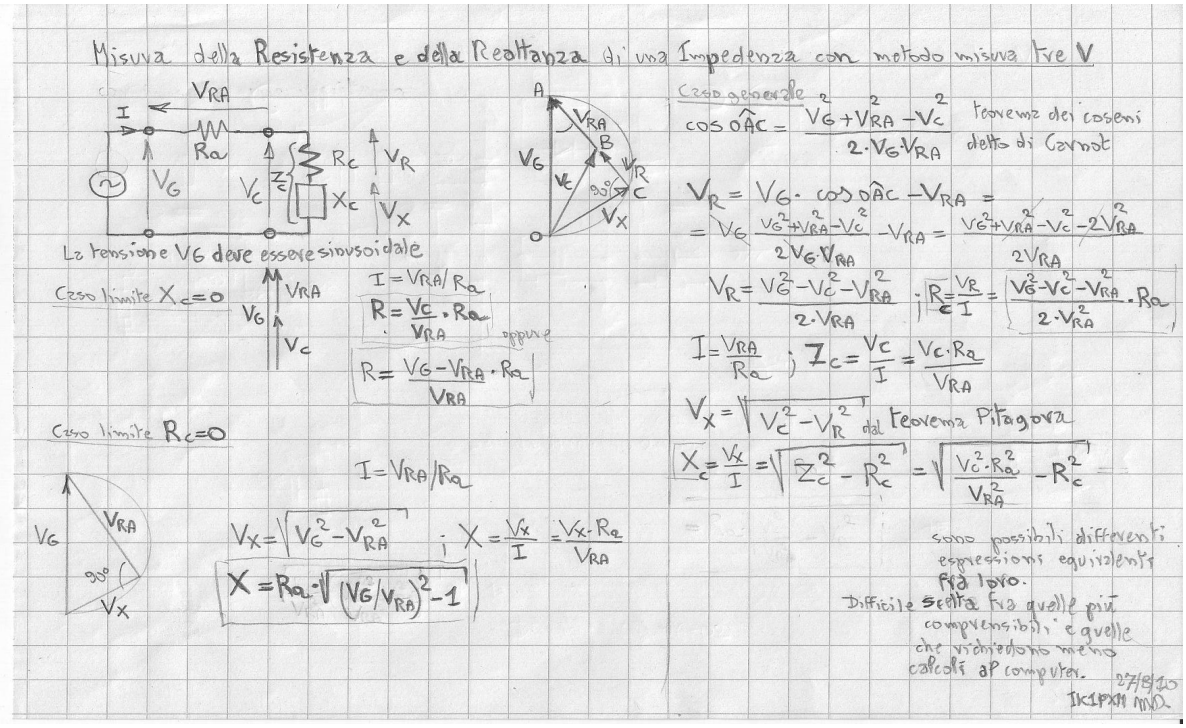
F generatore, L in uH/100, C in pF corrisp. a X e F

La visualizzazione delle tensioni efficaci VG,VR,VC serve come diagnostica, può venire eliminata. Se si avvia il microcontrollore con il pulsante Ps1 premuto, vengono visualizzate le effettive tensioni continue acquisite (serve per la manutenzione, diagnostica).

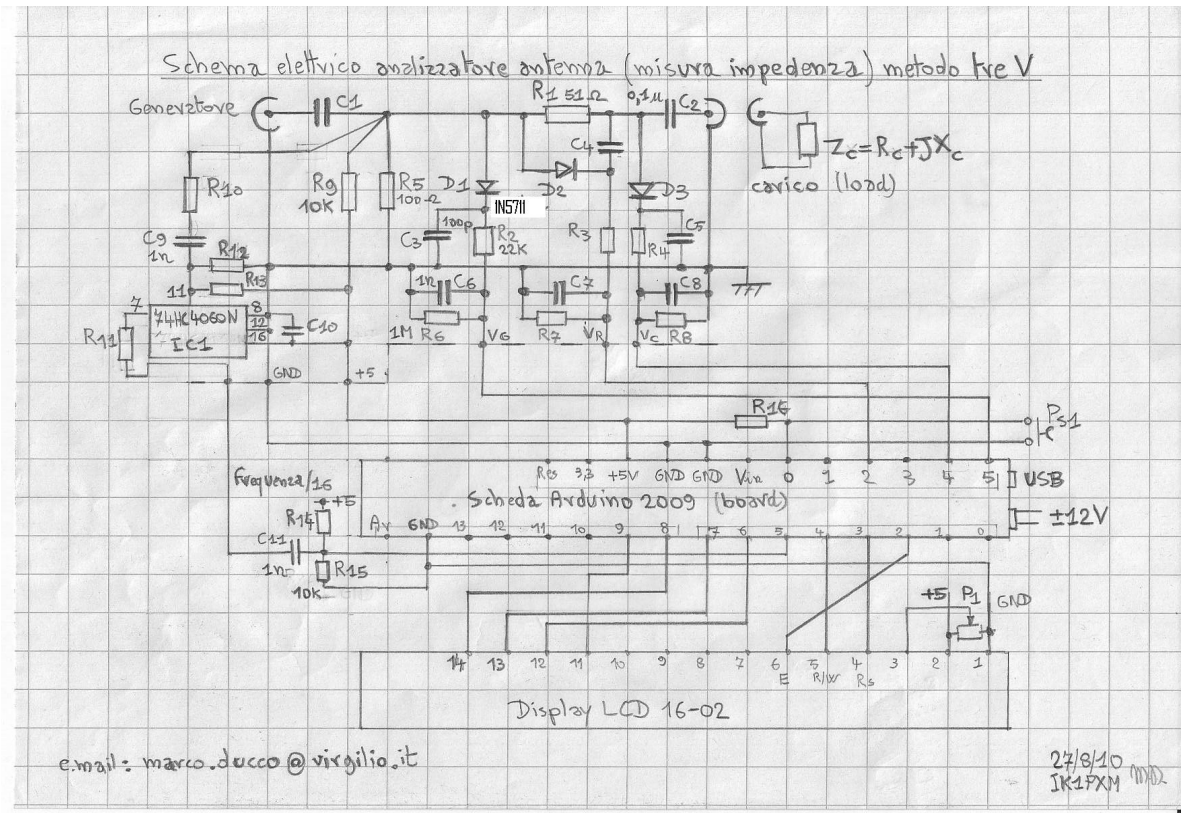
Nel primo spazio a sinistra della prima riga può comparire un codice di condizione anomala che consiste in uno dei seguenti caratteri:

A: somma tensioni inferiore a quella di alimentazione, si applica formula carico solo resistivo.
B: le tre tensioni costituiscono un triangolo acuto, si applica formula carico solo reattivo.
C: impedenza carico troppo bassa
D: impedenza carico troppo alta
E: tensione alimentazione troppo bassa.

Teoria del principio della misura



Schema elettrico



Elenco componenti:
C1, C2 0,1 uF tutti i condensatori sono ceramici
C3, C4, C5 470 pF
C6, C7, C8 10nF
C9, C10, C11 1nF

R1 51 Ω 2%; tutti i resistori sono da ¼ W 5%
R2, R3, R4 22 kΩ
R5 100 Ω
R6, R7, R8 1 MΩ
R9 10 kΩ
R10, R11 2,2 kΩ
R12, R13,R14,R15 10 kΩ
R16 47 kΩ

D1, D2, D3 1N5711 diodi schottky
IC1 74HC4060N

Scheda Arduino 2009 (oppure Arduino mini)
Display LCD due righe di 16 caratteri

Descrizione delle funzioni dei componenti

-R1 51 Ω è la resistenza di riferimento per il calcolo, la precisione dei calcoli dipende dal suo valore, ma una precisione migliore del 2% è inutile. Il valore di 50 Ω è scelto per avere la migliore precisione di lettura attorno a tale valore; si può cambiare il valore purché venga cambiato il corrispondente coefficiente nei calcoli (Magari mettere 100 Ohm per il range 20-400) .

-R2, R3, R4 circa 20 kΩ, valore non critico, penso che da 5k a 47 k vada tutto bene; insieme al rispettivo C6,C7,C8 costituiscono un filtro passa basso con costante di tempo di circa 10 ms, per lisciare la tensione applicata agli ingressi analogici del microcontrollore. Limitano la corrente negli ingressi analogici in caso di sovratensioni (spero che bastino a proteggerli !).

-R6, R7, R8 1MΩ, servono a "caricare" leggermente l'uscita del raddrizzatore, senza e senza tensione del generatore, le tensioni acquisite ballavano per i disturbi sulle connessioni un poco lunghe ed era antiestetico.

-R5 100 Ω, ho messo detto valore basso per caricare la linea a 50 Ω del segnale proveniente dal generatore. Un valore di 1 kΩ caricherebbe di meno a vuoto, però quando connesso in parallelo al carico da misurare prossimo a 50 Ω l'impedenza complessiva sarebbe 90 Ω,
non troppo più alta dei 50 Ω.

-R9 = 10kΩ, valore calcolato in modo che con R5 crea un partitore che applica
5V*(100 /(100+10k))= 50 mV che polarizza tutti i diodi e li avvicina alla conduzione.

-C3, C4, C5 470 pF, memorizzano la tensione massima, valore non critico, probabilmente da 47 pF a 1000 pF vanno tutti bene.

-C6, C7, C8 10 nF, valore non critico, insieme a R2,R3,R4 filtrano le tensioni del valore massimo.

-IC1, 74HC4060 ripple binary counter a 12 stadi. Usato come divisore frequenza per 16.
Il contatore della scheda Arduino è in grado di contare un'onda quadra avente una frequenza massima di circa 7 Mhz, dato che le frequenze di mio interesse arrivano fino a 40 Mhz, ho usato IC1 come prescaler. Il 74HC4060 funziona con segnali fino a 100 Mhz. Sarebbe meglio adoperare un vero prescaler in grado di funzionare fino a 150 MHz.

-R12, R13 10 kΩ, valore non critico, costituiscono un partitore che polarizza a metà dell'escursione l'ingresso logico di IC1, in modo da facilitare la commutazione quando viene applicato un segnale di ampiezza ridotta.

-R14, R14 10 kΩ, come sopra.

-C9, C11 1 nF, valore non critico disaccoppia la polarizzazione del partitore.

-R10 2,2 kΩ, visto che IC1 funziona già con un segnale di 1 Vpp, ho attenuato l'ingresso per caricare meno il generatore.

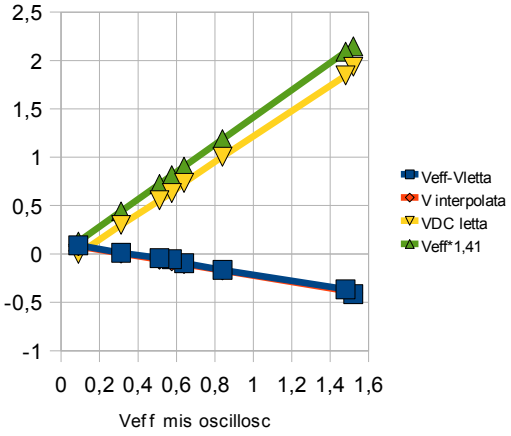
-R11 2,2 kΩ; senza R11, l'uscita a onda quadra ripida del contatore (se può funzionare fino a 100 MHz, l'uscita genera armoniche almeno fino a questa frequenza) induceva disturbi sui fili di connessione raddrizzati all'indietro da D2 per cui la tensione VR risultava sempre maggiore di zero. R11 (con le capacità parassita delle connessioni) attenua le armoniche ed elimina il disturbo.

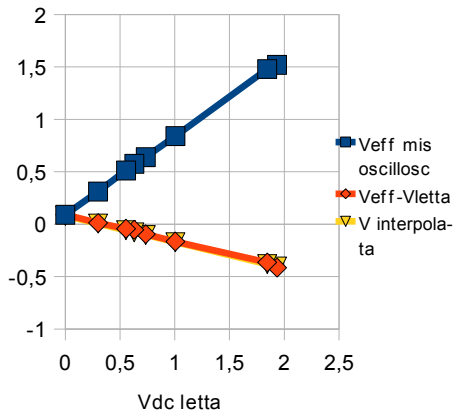
-R16 47kΩ, valore non critico, resistenza di pull-up del pulsante Ps1.

-D1,D2,D3 1N5711 diodi schottky. Più performanti dei vecchi 1N34, OA95.

Compensazione della non linearità del circuito raddrizzatore
Il raddrizzatore ideale, è un circuito aperto quando la tensione fra anodo e catodo è minore di zero, mentre è un cortocircuito altrimenti. In queste condizioni un filtro capacitivo non caricato memorizza il valore massimo della tensione sinusoidale applicata.
Nella realtà, esistono varie cause di errori:
In prima approssimazione i diodi entrano in conduzione solo quando la tensione fra anodo e catodo è maggiore di circa 0,2 V per i diodi al germanio e 0,6 V per quelli al silicio.
In conduzione non sono un cortocircuito, ma presentano una resistenza serie differenziale variabile fra 100 e 1 k Ω.
La tensione ai capi del condensatore non viene mantenuta costante, ma tende a scaricarsi sulla resistenza di carico e per la corrente inversa di perdita del diodo.
I portatori di carica (elettroni) che si muovono nel semiconduttore, non hanno velocità infinita: quando la tensione fra anodo e catodo passa attorno allo zero, gli elettroni rimangono intrappolati nella giunzione e per qualche nanosecondo il diodo rimane in conduzione.
Ho rilevato la caratteristica del circuito raddrizzatore, misurando per qualche valore, con un oscilloscopio digitale il valore efficace della tensione sinusoidale del generatore a 15 MHz, e leggendo sul display dello stesso analizzatore (avendo programmato appositamente la funzione) il valore della corrispondente tensione misurata.
I risultati sono riportati nei due grafici; i dati sono gli stessi, un grafico è funzione della Veff e contiene la Veff*1,41 che è l'uscita teorica del raddrizzatore, l'altro è funzione della Vdc; si osserva

che la Veff è già circa 0,1V quando la Vdc letta è ancora a zero.
La curva rossa rappresenta la funzione di compensazione da sommare alla tensione continua per ottenere la V efficace. Detta funzione di compensazione vale VD = 0,08 - 0.25 * Vdc e come mostrato nel listato del programma viene utilizzata nei calcoli (Non è necessario utilizzare proprio la V efficace; dato che interessano solo i rapporti fra i valori, si potrebbe utilizzare la Vmassima o qualunque altra grandezza proporzionale).





Programma applicativo
In appendice riporto il listato del programma applicativo.
Lo allego alla descrizione con un poco di preoccupazione per due motivi:
-La stesura del programma è l'attività che considero a maggior valore aggiunto e che richiede più tempo. Leggendo il listato si entra in possesso del completo risultato del knowhow; se il testo venisse plagiato mi spiacerebbe. Tuttavia, nel programma ho utilizzato senza difficoltà la routine di misura della frequenza diffusa da Martin Nawrath DH3JO che mi ha permesso di comprendere la gestione dei contatori e degli interrupt del processore e quindi ricambio fiducioso l'altruismo diffondendo il programma.
-L'altra preoccupazione è per le possibili critiche. Quando si diffonde il solo programma esecutivo, gli utenti vedono che funziona, ma non sanno quanto è inefficiente, mal scritto, critico, pieno di errori che solo per fortuna non lo mandano in crash.
Ho sviluppato il programma un poco alla volta senza una visione d'insieme: la struttura logica è migliorabile, il testo non è perfetto, i commenti non sono esaustivi, vengono usate più variabili dello stretto indispensabile, ma non ho più voglia di abbellirlo. L'ho detto e mi sento assolto...hi !

Confronto con gli altri apparati
Questo progetto non funziona meglio degli altri apparati, al più in modo uguale.

L'analizzatore MFJ249 adopera componenti a montaggio superficiale per il circuito di misura; tali componenti sono minuscoli, presentono bassi parametri parassiti e permettono un funzionamento fino a 450 MHz.

Gli altri apparati posseggono svariati potenziometri di regolazione:
- Il MFJ249 ne ha uno di offset e uno di guadagno per ognuno dei tre canali analogici.
- Quello VK5JST ne ha uno di guadagno per ognuno dei tre canali.

Io non li ho messi per semplificarmi la parte hardware, perché al momento ho fatto un unico prototipo, e mi viene più comodo realizzare le tarature cambiando i coefficienti nei calcoli.
Non sono sicuro che, per la tolleranza dei componenti, altri esemplari replichino gli stessi risultati.

MFJ e VK5JST compensano la caduta di tensione nei diodi raddrizzatori aggiungendo un diodo nella rete di reazione dell'amplificatore di recupero offset che in parte compensa anche la variazione della caduta con la temperatura ambiente.
Ho realizzato la compensazione via software; è possibile che al variare della temperatura ambiente la compensazione costante introduca errori.

MFJ e VK5JST amplificano i segnali dai raddrizzatori fino a 4 volte per ottimizzare l'escursione dei canali di ingresso che hanno campo 0-5V con conversione a 8 bit (256 valori).
Io non ho amplificato perché i canali in ingresso 0-5V di Arduino hanno risoluzione 10 bit (1024 valori); ogni valore lo leggo poi per 20 volte di seguito e ne faccio la media, ottenendo una risoluzione equivalente di 20480 valori.
Riconosco che ottimizzare il campo di misura è una cosa buona; io non ho voglia di aggiungere troppo hardware e non lo ho fatto, può essere una miglioria.

Ho adoperato diodi schottky 1N5711, VK5JST adopera diodi al germanio a punta di contatto 1N34, vecchissimi ma diffusi, forse di minori prestazioni.

A mio vantaggio gioca l'utilizzo della scheda Arduino che adopera un chip Atmega168 più potente e moderno di quello PICAXE 28X1 (PIC16F886) adoperato da VK5JST.
Ho esaminato il SW sorgente distribuito da VK5JST; è scritto in BASIC, mi sono tornati in mente i programmi scritti negli anni 80 con il Commodore 64, pieni di PEEK, POKE, e GOTO.
Inoltre i calcoli sono fatti solo con numeri interi, servono con una serie di scalamenti per contenere il risultato nel campo voluto.
Arduino ha un processore floating point hardware, si possono adoperare numeri reali e quindi non preoccuparsi degli scalamenti; è programmabile in C++ che è un linguaggio che obbliga a strutturare correttamente la logica del programma.

Riassumo il mio pensiero: VK5JST ha fatto un ottimo lavoro, utilizzando nel 2004 un microcontrollore economico di ridotte prestazioni. Quando si progetta un kit per produzione di serie si deve pensare a ridurre al minimo il costo dei materiali. Adoperando Arduino 2009 che monta un Atmega 328 progettato nel 2008, ho avuto le attività alquanto facilitate. Per me il costo ricorrente non è così importante perché ho fatto un solo esemplare. La scheda Arduino2009 costa poi solo circa 20 € e il sistema di sviluppo è gratis (www.arduino.cc).

Considerazioni finali
Siamo in un periodo economico incerto, in Italia dobbiamo tutti diventare più bravi per continuare a contrastare la concorrenza di altri progettisti elettronici informatici e mantenere il nostro benessere.
Sono da poco in pensione, scrivo per divertimento, diffondo queste informazioni sperando che siano non solo di interesse per l' hobby, ma utili a incrementare le conoscenze professionali di chi, più giovane di me, è ancora economicamente attivo.

A few phrases for search engines:
misura impedenza analizzatore antenna progetto arduino IK1PXM
aerial analyser antenna analyzer arduino project

Appendice: Listato del sorgente programma applicativo

```
/* Marco Ducco IK1PXM. Misura Z= R+JX carico con metodo misura tre tensioni
2/8/010  Inizio stesura sviluppata nell'ambiente  Arduino 0017  ora superato dalla versione 0018
5-8/8/010 compensazione caduta diodi con tensione
17/8  modificato hardware: LCD d4 ora connesso al pin 9 era al pin 5; il pin 5 usato per ingresso counter nel sw misura impedenza
24/8 aggiunto misura frequenza
*/
#include <LiquidCrystal.h>
LiquidCrystal lcd(3,4,2,9,6,7,8);//Piedini display LCD-Arduino: rs pin 3, rw pin 4, enable pin 2; d4,d5,d6,d7 pin: 9,6,7,8

int nu_ana[6];
char vet1[17];
#define inp1 14 //ana0 selezione  alla accensione test  e poi selezione visualizzazioni

float MVAG,MVRAG,MVZG,MVA,MVRA,MVZ,VD,COSOAB,SENOAB,MVRL,MVXL,XSC,XSC1,XSC2,RSC1,RSC2,RSC,RPC,XPC,ZC;
float SRSC,SXSC,SZC;
float R0,DENG,GR,GX,MG,ROS;

int IS;
char vet2[17] = {"               "};
char txt0[17] = {"RS    XS    Z   "};
char txt1[17] = {"RP    XP    Z   "};
char txt2[17] = {"G    SWR    Z   "};
char txt3[17] = {"               "};
char txt4[17] = {"     cuH      pF"};
        //     "0123456789ABCDEF"
int crit= 0;
const float RA = 52. ; // valore resistenza in serie alla alimentazione
boolean test;
byte disp= 0;
byte CASO;

#include <avr/interrupt.h>
#include <WProgram.h>
//#include <avr/pgmspace.h>

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

unsigned long f_freq;
volatile unsigned char f_ready;
volatile unsigned char f_mlt;
volatile unsigned int f_tics;
volatile unsigned int f_period;
volatile unsigned int f_comp;

long int frqkHz;
float LScuH,CSpF;

void setup()
{
 lcd.begin(16,2); // con Arduino0017 occorre definire ncolonne e nrighe del display
 Serial.begin(19200);
 Serial.println (" inizio ");
 lcd.setCursor(0, 0);  lcd.print (" misura Z= R+JX ");
 lcd.setCursor(0, 1);  lcd.print ("IK1PXM vs0100825");
 delay(2000); //
 lcd.clear(); //pulisci il display
 if( digitalRead(inp1) == 0)
 {test = true ; lcd.setCursor(0, 0);  lcd.print (" test Vgrezze ");delay(2000); }
 else
 {test = false;} // selezione modalità test

}

void loop()
{
  if (crit < 8) {crit = crit+1;}  //selezione visualizzazione
  else
    { switch (disp)
       { case 0: if( digitalRead(inp1) == 0) { disp = 1; crit = 0;};break;
         case 1: if( digitalRead(inp1) == 0) { disp = 2; crit = 0;};break;
         case 2: if( digitalRead(inp1) == 0) { disp = 3; crit = 0;};break;
         case 3: if( digitalRead(inp1) == 0) { disp = 0; crit = 0;};break;
```

```
    }
  }

  for (int j=2;j<6;j++) { nu_ana[j] = 0; for (int i=0;i<20;i++) {nu_ana[j]= nu_ana[j]+ analogRead(j);}}//legge tutti input ora 20 volte (era 10)
  //il max è  1023 per 5V. Si sommano 30 valori; 30690 = 500 centesimi di V     500/306900 = 0,0162919
  //il max è  1023 per 5V. Si sommano 20 valori; 20460 = 500 centesimi di V     500/20460 = 0,0244
  //Serial.print (" = "); Serial.print (nu_ana[5]);Serial.print (" "); Serial.print (volt);
  const float kconv = 0.0162919 ; // 0.0488;
  MVAG= (kconv * nu_ana[5]);
  MVRAG= (kconv * nu_ana[2]);
  MVZG= (kconv * nu_ana[4]);
  //VD compensazione della caduta nel diodo, in funzione della tensione raddrizzata
  VD= 8 - MVAG*0.25   ;  MVA = MVAG+VD; // centesimi di Volt efficaci Modulo V Alimentazione
  VD= 8 - MVRAG*0.25; MVRA = MVRAG+VD; // centesimi Volt efficaci Modulo Vresistenza alimentazione
  VD= 8 - MVZG*0.25   ;  MVZ = MVZG+VD; // centesimi di Volt efficaci modulo di Vz

// Condizioni per calcoli per casi limite per dati imprecisi:
CASO = 0;

if ((MVRA + MVZ) < MVA) //somma cadute inferiore alla valimentaz, supponi carico puramente resistivo
  {XSC = 0; RSC1 = (MVZ/MVRA)*RA; RSC2 = (MVZ/(MVA-MVZ))*RA; RSC= (RSC1+RSC2)/2; ZC=RSC; //due diversi metodi equivalenti, si
fa la media dei risultati
  CASO = 1;}

if ((MVRA*MVRA+MVZ*MVZ)> (MVA*MVA))//il triangolo delle tensioni è acuto, non retto, supponi carico tutto reattivo,
  { RSC=0; XSC1= sqrt(MVA*MVA-MVRA*MVRA); XSC2 = (MVZ/MVRA)*RA; XSC= (XSC1+XSC2)/2;ZC=XSC; //due diversi metodi
equivalenti, si fa la media dei risultati
  CASO = 2;}

if (MVRA > (10.*MVZ)) {RSC=0; XSC = 0; ZC=0; CASO = 3;}//impedenza troppo bassa (< 5 Ohm)

if ((10.*MVRA) < MVZ) {RSC=999; XSC = 999; ZC=999; CASO = 4;}//impedenza troppo alta (> 500 Ohm)

if (MVA < 50){RSC= 301; XSC = 301; ZC=301; CASO = 5;} //tensione alimentazione bassa imponi valori fittizi

if (CASO ==0 )  //nessuno dei casi particolari precedenti
  {
  //Formule di elaborazione misure delle tre tensioni
  //OAB angolo con vertice fra VA e VRA
  COSOAB = (MVA*MVA+MVRA*MVRA-MVZ*MVZ)/(2.*MVA*MVRA);
  if (COSOAB > 1.) {COSOAB =1.;} //limita per evitare errore calcolo seno nel caso di dati imprecisi con MVZ =0
  if (COSOAB < 0.) {COSOAB =0.;}  //limita nel caso di dati imprecisi con MVRA = 0
  SENOAB = sqrt(1-COSOAB*COSOAB);
  MVRL = (MVA*COSOAB - MVRA);    MVXL = MVA*SENOAB;
  RSC = (MVRL/MVRA)*RA;  XSC = (MVXL/MVRA)*RA;  //Resistenza/impedenza  Serie del Carico in ohm
 //ZC = sqrt(RSC*RSC+XSC*XSC);
  if (MVRA > 0.){ZC = (MVZ/MVRA)*RA;} else {ZC = 999.;}// diverso metodo  per calcolo Z indipendente da RS e XS
  }

//media dei valori calcolati per ridurre e rallentare le oscillazioni dei valori sul display
  SRSC = SRSC + RSC; SXSC = SXSC + XSC; SZC = SZC + ZC; IS++;
  if( IS >9)
{ RSC = SRSC/IS; XSC = SXSC/IS; ZC = SZC / IS;
   IS=0; SRSC=0;SXSC=0;SZC=0;

  //calcolo RPC e XPC valori equivalenti in parallelo dai valori serie calcolati
  if (RSC > 0){RPC = RSC+ XSC*XSC/RSC;}else {RPC=9999;}
  if (XSC > 0){XPC = XSC+ RSC*RSC/XSC;}else {XPC=9999;}

//Calcolo coefficiente di riflessione Gamma  e SWR
/* Corrispondenza formale fra R, G, SWR con R0=50 Ohm:
  R:  25,   100,   150,  12.5,  200,  50,  75,   476
  G;   0.33,  0.33,  0.5,  0.6,   0.6,  0,  0.2,  0.81
  SWR:  2,     2,     3,    4,     4,    1,  1.5,  9.9
*/
 R0= 50.; //impedenza  di riferimento
 DENG = (RSC+R0)*(RSC+R0)+ XSC*XSC;//espressione al denominatore
 GR = (RSC*RSC - R0*R0 + XSC*XSC )/DENG ;//componente reale di gamma
 GX = 2*R0*XSC/DENG; //componente immaginaria di gamma
 MG = sqrt(GR*GR+GX*GX); if (MG>=1){MG=0.99;}//modulo di gamma limita a 0.99 per limitaz display
 if (MG < 0.81) {ROS = (1+MG)/(1-MG);} else {ROS = 9.9;} //rapporto onde stazionarie

 if (disp==0)
 {visualV();for (int i=0;i<16;i++){vet2[i] =txt0[i];}; intasc3(int(RSC), & vet2[ 2]); intasc3(int(XSC), & vet2[8]); intasc3(int(ZC), & vet2[13]);}

 if (disp==1)
```

```
 {visualV();for (int i=0;i<16;i++){vet2[i] =txt1[i];}; intasc3(int(RPC), & vet2[ 2]); intasc3(int(XPC), & vet2[8]); intasc3(int(ZC), & vet2[13]);}

 if (disp==2)
 {visualV();for (int i=0;i<16;i++){vet2[i] =txt2[i];}
  intasc(int(1000.*MG), & vet2[ 1],3,3);vet2[4]=' '; intasc(int(10*ROS), & vet2[7],3,1);vet2[7]='R'; intasc3(int(ZC), & vet2[13]);
 }
 if (disp==3)
 {
  //...mis frequenza
  f_comp= 16; // Set compensation to 8   ; (16 con 160ms) (8 con 100ms mis. 4MHz)  (112 buono con 1000ms misurando 4MHz)
  start(160); // Start 100 counting with gatetime of 100ms
  while (f_ready == 0) //wait until counter ready
  frqkHz=f_freq/10; // read result
  //Serial.print(frqkHz);Serial.print("  "); // print result
  //... fine mis frequenza

  for (int i=0;i<16;i++){vet2[i] =txt4[i];}
  if (frqkHz > 2000)
   { LScuH= (100000./6.28)*XSC/frqkHz;  intasc3(int(LScuH), & vet2[0]); //induttanza equivalente alla X serie in centesimi di microHenry
     if (XSC > 0){ CSpF= 159235668./(XSC*frqkHz); intasc3(int(CSpF), & vet2[8]);} //capacità serie equivalente in picoFarad
   }

  for (int i=0;i<16;i++){vet1[i] =txt3[i];}  intasc5(frqkHz, & vet1[2]); vet1[9]='M';vet1[10]='H';vet1[11]='z';
  }
}

if (CASO==0){vet1[0]=' ';} //visualizza il tipo di algoritmo usato
if (CASO==1){vet1[0]='A';}
if (CASO==2){vet1[0]='B';}
if (CASO==3){vet1[0]='C';}
if (CASO==4){vet1[0]='D';}
if (CASO==5){vet1[0]='E';}

 lcd.noCursor();
 lcd.setCursor(0, 0); for (byte i=0; i<16;i++){lcd.write(vet1[i]);} // scrive riga alta
 lcd.setCursor(0, 1); for (byte i=0; i<16;i++){lcd.write(vet2[i]);} // scrive riga bassa
}

void visualV()
{
if (test)
 {intasc(int(MVAG), & vet1[0],4,2); intasc(int(MVRAG), & vet1[5],4,2);  intasc(int(MVZG), & vet1[11],4,2);  vet1[10]=' ';}//tensioni grezze
else
 {intasc(int(MVA), & vet1[0],4,2); intasc(int(MVRA), & vet1[5],4,2);  intasc(int(MVZ), & vet1[11],4,2);  vet1[10]=' ';}//tensioni compensate
}


/*
  Using Counter1 for counting Frequency on T1 / PD5 / digitalPin 5
  Using Timer2 for Gatetime generation
  Martin Nawrath KHM LAB3  Kunsthochschule für Medien Köln
  Academy of Media Arts  http://www.khm.de http://interface.khm.de/index.php/labor/experimente/
  History: Dec/08 - V0.0

The Frequency input is fixed to digital pin 5. This pin is mapped to the alternate port
 function T1 which is the input 16 Bit Hardware Counter1. To obtain a higher resolution than 16 Bit,
 the counter overflows are counted also and are calculated with the counter value to the final
 long integer result. The Frequency Source output must have a digital level so that weak Signals have
 to be amplified for instance by an single transistor or a 74HC14 inverter.
 The maximum input frequency is about 8 MHz when signal duty cycle is 50%.
 If you like to measure higher frequencies you have to use an prescaler or divider circuit
 which can be used from other counter projects published in the web.
 The Gate Time for the counting period can be chosen in the start() function where values
 of 10, 100 or 1000 ms are practicable for a resolution of 100, 10 and 1 Hz but any value can be used.
 The internal resolution of the gatetime is 2 ms so that the time can be varied in the increment of 2.
 If you wish to minimize the indication error the value of FreqCounter::f_comp variable can compensate
 slight gatetime errors. Compared to an commercial ACECO counter it is possible to trim the deviation
 to almost zero over the whole range. For gatetimes of 10,100,100 the values 1, 10 and 100 where found
 to be good for our Duemilanova boards.
*/

void start(int ms)
{
  f_period=ms/2;
  if (f_comp ==0) f_comp=1;
              // hardware counter setup ( refer atmega168.pdf chapter 16-bit counter1)
```

```
 TCCR1A=0;         // reset timer/counter1 control register A
 TCCR1B=0;         // reset timer/counter1 control register A
 TCNT1=0;          // counter value = 0
 // set timer/counter1 hardware as counter , counts events on pin T1 ( arduino pin 5)
 // normal mode, wgm10 .. wgm13 = 0
 sbi (TCCR1B ,CS10);   // External clock source on T1 pin. Clock on rising edge.
 sbi (TCCR1B ,CS11);
 sbi (TCCR1B ,CS12);

 // timer2 setup / is used for frequency measurement gatetime generation
 // timer 2 presaler set to 256 / timer 2 clock = 16Mhz / 256 = 62500 Hz
 TCCR2A=0;
 TCCR2B=0;
 cbi (TCCR2B ,CS20);  sbi (TCCR2B ,CS21); sbi (TCCR2B ,CS22);

 //set timer2 to CTC Mode
 cbi (TCCR2A ,WGM20); sbi (TCCR2A ,WGM21); cbi (TCCR2B ,WGM22);
 OCR2A = 124;

 f_ready=0;               // reset period measure flag
 f_tics=0;              // reset interrupt counter
 sbi (GTCCR,PSRASY);       // reset presacler counting
 TCNT2=0;                 // timer2=0
 TCNT1=0;                 // Counter1 = 0

 cbi (TIMSK0,TOIE0);       // disable Timer0  //disable  millis and delay
 sbi (TIMSK2,OCIE2A);      // enable Timer2 Interrupt

 TCCR1B = TCCR1B | 7;      //  Counter Clock source = pin T1 , start counting now
}

//****************************************************************
//  Timer2 Interrupt Service is invoked by hardware Timer2 every 2ms = 500 Hz
//  16Mhz / 256 / 125 = 500 Hz
//  here the gatetime generation for freq. measurement takes place:

ISR(TIMER2_COMPA_vect)
{
                                    // multiple 2ms = gate time = 100 ms
 if (f_tics >= f_period)
   {      // end of gate time, measurement ready

         // GateCalibration Value, set to zero error with reference frequency counter
   delayMicroseconds(f_comp); // 0.01=1/ 0.1=12 / 1=120 sec
   TCCR1B = TCCR1B & ~7;                          // Gate Off  / Counter T1 stopped
   cbi (TIMSK2,OCIE2A);                           // disable Timer2 Interrupt
   sbi (TIMSK0,TOIE0);                            // enable Timer0 again // millis and delay
   f_ready=1;            // set global flag for end count period

                                                  // calculate now frequeny value
   f_freq=0x10000 * f_mlt;  // mult #overflows by 65636
   f_freq += TCNT1;                 // add counter1 value
   f_mlt=0;
   }
  f_tics++;                // count number of interrupt events
 if (TIFR1 & 1)
   {                                // if Timer/Counter 1 overflow flag
   f_mlt++;             // count number of Counter1 overflows
   sbi(TIFR1,TOV1);                               // clear Timer/Counter 1 overflow flag
   }
 // PORTB = PORTB ^ 32;                                      // int activity test
}


//funzione converte un numero intero max 9999 nei 4 caratteri ascii corrispondenti e posiziona punto
// il risultato avviene  trasmesso tramite puntatore   31/5/09
void intasc(int num, char *na, int nc,int np)
{
  # define sepdec  46  // 44 virgola;  46 punto
 // variabili locali usate dalla funzione:
 // nc numero cifre da 3 a 4; np posizione virgola da 0 a 3 prima della ultima cifra
 boolean cifrasignif = false;
 boolean cifredecim = false; // se true ora sono cifre decimali in cui lo zero va sempre scritto
 int nummigl,numcent,numdeci,numunit,nummiglper1k,numcentper100,numero;
numero = num; // il dato in ingresso non viene modificato
 if (nc > 4) { nc = 4;}; if (nc < 3) { nc = 3;}
```

```
 if (np > 3) { np = 3;}; if (np < 0) { np = 0;}

 if (numero > 9999) { numero = 9999;} if (numero < 0) { numero = 0;}
int ic = 0 ; //indice cifra
nummigl = numero/1000;nummiglper1k = nummigl*1000;

if (nc == 4)  //scrivi la quarta cifra solo se richiesto
{
if (nummigl == 0) { na[ic]=32;ic++;} // forza il carattere spazio ( 32 in ASCII)
else
{ na[ic]=nummigl + 48; ic++; //carica la cifra da 0 a 9 in ASCII
cifrasignif = true;//la cifra ï¿½ significativa
}
}
if (np == 3) { na[ic]= sepdec ;cifredecim = true; ic++;}  // inserisci simbolo inizio decimali
numcent = (numero-nummiglper1k)/100; numcentper100 = numcent*100;
if (numcent == 0)
//{     if (cifrasignif == false) { na[ic]=32;ic++;} else { na[ic]=48;ic++; } }
{     if ((cifrasignif == true)||(cifredecim == true))  { na[ic]=48;} else { na[ic]=32;}ic++; }


else
{  na[ic] = numcent + 48; ic++; cifrasignif = true;}

if (np == 2) { na[ic]= sepdec ; cifredecim = true; ic++;}
numdeci = (numero-nummiglper1k-numcentper100)/10;
if (numdeci == 0)
//{     if (cifrasignif == false) {  na[ic]=32; ic++; } else { na[ic]=48;ic++; } }
{     if ((cifrasignif == true)||(cifredecim == true))  { na[ic]=48;} else { na[ic]=32;}ic++; }

else
{  na[ic] = numdeci + 48;ic++;}
if (np == 1) { na[ic]= sepdec ;cifredecim = true; ic++;}
//la cifra delle unitï¿½ viene sempre scritta
numunit = (numero-nummiglper1k-numcentper100-numdeci*10); na[ic] = numunit + 48;
}


//funzione converte un numero intero max 999 nei 3 caratteri ascii corrispondenti
void intasc3(int num, char *na)
{
 boolean cifrasignif = false;
 int numcent; int numdeci; int numunit; int numcentper100; int numero;
numero = num; // il dato in ingresso non viene modificato
if (numero > 999) { numero = 999;} if (numero < 0) { numero = 0;}
byte ic = 0 ; //indice cifra
numcent = numero/100; numcentper100 = numcent*100;
if (numcent == 0){na[ic]=32;} else {  na[ic] = numcent + 48; cifrasignif = true;} ic++;
numdeci = (numero-numcentper100)/10;
if (numdeci == 0) {if ((cifrasignif == true)) {na[ic]=48;}else{ na[ic]=32;}} else {na[ic] = numdeci + 48;} ic++;
numunit = (numero-numcentper100-numdeci*10); na[ic] = numunit + 48;//la cifra delle unita viene sempre scritta
}

void intasc5(long num, char *na) //converte numero intero max 99999 nei 5 caratteri corrispondenti
{
  # define sepdec  46  // 44 virgola;  46 punto
  boolean cifrasignif = false;
  long nudecmi, numigl, numcent, numdeci, numunit, nudecmiper10k, numiglper1k, numcentper100, n;
  n = num;if (n > 99999) { n = 99999;} if (n < 0) { n = 0;} // il dato in ingresso non viene modificato
  byte ic = 0 ; //indice cifra
  nudecmi = n/10000; nudecmiper10k= nudecmi*10000;
  if (nudecmi == 0){na[ic]=32;} else {  na[ic] = nudecmi + 48; cifrasignif = true;} ic++;
  numigl = (n-nudecmiper10k)/1000; numiglper1k = numigl*1000;
  if (numigl == 0){if ((cifrasignif == true))  { na[ic]=48;} else { na[ic]=32;}} else{ na[ic] = numigl + 48;} ic++;
  na[ic]= sepdec ; cifrasignif = true; ic++;  //scrivi  se e dove vuoi il separatore decimali
  numcent = (n-nudecmiper10k-numiglper1k)/100; numcentper100 = numcent*100;
  if (numcent == 0) {if ((cifrasignif == true))  { na[ic]=48;} else { na[ic]=32;}} else{ na[ic] = numcent + 48;cifrasignif = true;} ic++;
  numdeci = (n-nudecmiper10k-numiglper1k-numcentper100)/10;
  if (numdeci == 0) {if ((cifrasignif == true))  { na[ic]=48;} else { na[ic]=32;}} else { na[ic] = numdeci + 48;} ic++;
  numunit = (n-nudecmiper10k-numiglper1k-numcentper100-numdeci*10); na[ic] = numunit + 48;//la cifra delle unita viene sempre scritta
}
```