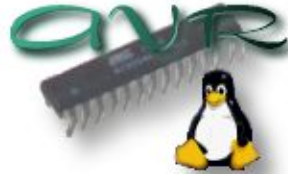


## Programmare il microcontrollore AVR con GCC



by Guido Socher ([homepage](#))

### Abstract:

L'AVR è un microcontroller ad 8 bit in tecnologia RISC prodotto dalla Atmel. Si tratta di un microcontrollore assai diffuso basato su di un singolo circuito integrato contenente una EEPROM, della RAM, un convertitore da analogico a digitale, molteplici linee di ingresso ed uscita digitali, dei timer, e molteplici altre cose (per esempio in alcuni modelli è presente anche una porta seriale UART di tipo RS232).

### About the author:

Guido adora Linux non solo per il fatto che sia divertente scoprire le grandi possibilità di questo sistema, ma anche per le persone che sono coinvolte nel suo sviluppo.

La cosa più interessante è il fatto che sotto Linux è disponibile un completo ambiente di programmazione per il medesimo. Potrete quindi programmare il vostro Microcontrollore in C per mezzo di GCC. In questo articolo vi spiegherò come installare ed utilizzare il GCC. Vi spiegherò dettagliatamente anche come caricare il software prodotto nel microcontrollore. Tutto quello che vi serve sono pochi componenti: un microcontrollore AT90S4433, un quarzo da 4MHz, del filo e pochi altri componenti assai economici.

Questo articolo vuole essere solo una introduzione. In un ulteriore articolo costruiremo un display a cristalli liquidi con alcuni pulsanti, ingressi analogici e digitali, un watchdog di tipo hardware e LED. L'idea di base è quella di utilizzarlo come pannello di controllo generico per un server Linux, ma prima di tutto dobbiamo imparare a creare un ambiente di sviluppo adatto, ed è per l'appunto questa la funzione di questo articolo.

---

## Installazione del software: che ci serve.

Per poter utilizzare l'ambiente di sviluppo GNU C abbiamo bisogno del seguente software:

binutils-2.11.2.tar.bz2	Scaricabile da: <a href="ftp://ftp.informatik.rwth-aachen.de/pub/gnu/binutils/">ftp://ftp.informatik.rwth-aachen.de/pub/gnu/binutils/</a> o <a href="ftp://gatekeeper.dec.com/pub/GNU/binutils/">ftp://gatekeeper.dec.com/pub/GNU/binutils/</a>
-------------------------	--

gcc-core-3.0.3.tar.gz	Scaricabile da: ftp://ftp.informatik.rwth-aachen.de/pub/gnu/gcc/ o ftp://gatekeeper.dec.com/pub/GNU/gcc/
avr-libc-20020106.tar.gz	Le librerie C per AVR sono scaricabili da: http://www.amelek.gda.pl/avr/libc/ Se volete potete scaricarle da questa <a href="#">pagina</a> in questo server.
uisp-20011025.tar.gz	Il programmatore per l'AVR è scaricabile da: http://www.amelek.gda.pl/avr/libc/ Se volete potete scaricarle da questa <a href="#">pagina</a> in questo server.

Installeremo tutti i programmi nella cartella /usr/local/atmel. Questo al fine di tenere separati i programmi, nella fattispecie il compilatore, dal compilatore C che normalmente trovate in Linux. Creiamo quindi la directory con il seguente comando:

```
mkdir /usr/local/atmel
```

## Installazione del software: Le GNU binutils

Il pacchetto contenente le binutils fornisce tutte gli strumenti basilari necessari per creare i file oggetto. Include un assembler AVR (avr-as), un linker (avr-ld), strumenti per la gestione delle librerie (avr-ranlib, avr-ar), programmi per la creazione di file-oggetto caricabili nella EEPROM del microcontrollore (avr-objcopy), un disassemblatore (avr-objdump) e strumenti di utilità come avr-strip e avr-size.

Eeguire i seguenti comandi per compilare ed installare le binutils:

```
bunzip2 -c binutils-2.11.2.tar.bz2 | tar xvf -
cd binutils-2.11.2
./configure --target=avr --prefix=/usr/local/atmel
make
make install
```

Aggiungete ora la file /etc/ld.so.conf la riga /usr/local/atmel/lib ed eseguite il comando /sbin/ldconfig per rigenerare la cache del linker.

## Installazione del software: AVR gcc

avr-gcc sarà il nostro compilatore C.

Eeguire i seguenti comandi per compilarlo ed installarlo:

```
tar zxvf gcc-core-3.0.3.tar.gz
cd gcc-core-3.0.3
./configure --target=avr --prefix=/usr/local/atmel --disable-nls --enable-language=c
make
make install
```

## Installazione del software: Le librerie C per AVR

Le librerie C sono ancora in fase di sviluppo. Il procedimento di installazione delle medesime potrebbe variare da versione a versione. Vi raccomando di utilizzare la versione che cito se volete seguire le mie istruzioni alla lettera. Ho testato questa versione con successo su tutti i programmi che scriveremo in questo e nei seguenti articoli.

Definiamo dapprima alcune variabili d'ambiente (queste istruzioni sono valide per chi usa *bash* come shell):

```
export CC=avr-gcc
export AS=avr-as
export AR=avr-ar
export RANLIB=avr-ranlib
export PATH=/usr/local/atmel/bin:${PATH}
```

```
./configure --prefix=/usr/local/atmel/avr --target=avr --enable-languages=c --host=avr
make
make install
```

## Installazione del software: Il programmatore

Il programmatore è un particolare software che carica nella EEPROM il codice-oggetto preparato.

Il programore uisp è un buon software. Può essere utilizzato direttamente dal Makefile. Si deve solamente aggiungere la regola "make load" avendo così la possibilità di compilare e caricare il software con un solo comando.

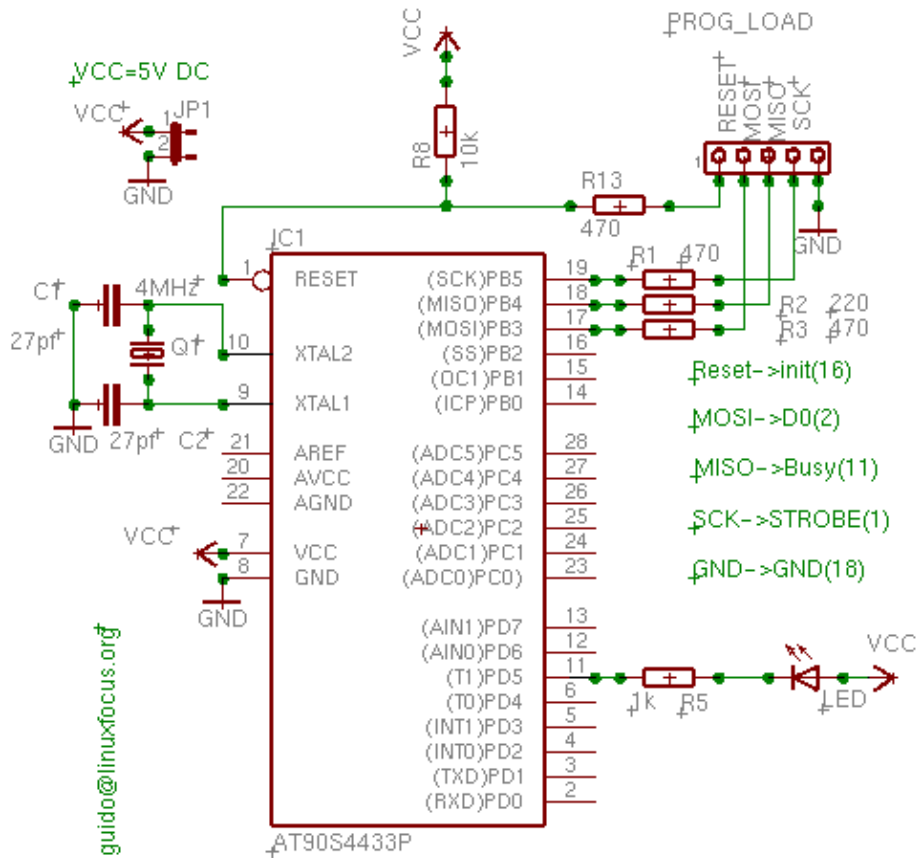
Per installare uisp seguite le seguenti istruzioni:

```
tar zxvf uisp-20011025.tar.gz
cd uisp-20011025/src
make
cp uisp /usr/local/atmel/bin
```

## Un semplice programma per il test

Inizieremo con un semplice e breve programma per testare il circuito. Lo scopo di questo è di testare il nostro ambiente di sviluppo. Possiamo utilizzare l'ambiente per compilarlo, caricarlo nel microcontrollore e testare il programma. Il programma fa semplicemente lampeggiare un diodo LED.

Vi suggerisco di creare un piccolo circuito stampato per il microcontrollore. Potrete, in un secondo momento, ampliare questo circuito per adattarsi ai vostri esperimenti. Una buona soluzione può essere quella di ricorrere ad una basetta sperimentale. È però bene che non tentiate di collegare l'AVR ed il quarzo direttamente sulla basetta. Vi consiglio di utilizzare dei corti spezzoni di cavo per le linee di input ed output, in quanto questo tipo di basette non sono adatte all'uso ad alte frequenze. Il quarzo ed i relativi condensatori di compensazione dovrebbero essere molti vicini al microcontrollore.




Le resistenze applicate al connettore per la programmazione non sono necessarie nel nostro caso. Queste vi servono solo nel caso vogliate utilizzare la porta B di ingresso/uscita per altri scopi.

## Materiale necessario

Vi servono i componenti indicati nella tabella qui sotto. Tutti sono assai diffusi ed economici. Solo il microcontrollore è un poco più costoso, all'incirca 7,50 Euro. Sebbene sia un microcontrollore assai diffuso potreste non trovarlo in tutti i negozi di componenti elettronici, ma i grossi distributori lo hanno sicuramente. (Alcune catene sono presenti anche in internet, qui ve ne sono alcune per la Germania: [www.reichelt.de](http://www.reichelt.de), [www.conrad.de](http://www.conrad.de); per la Francia: [www.selectronic.fr](http://www.selectronic.fr); per l'Italia: [www.finim.com](http://www.finim.com), [www.bigchip.it](http://www.bigchip.it), [www.centro-ovest.it](http://www.centro-ovest.it). Probabilmente nei vostri paesi potrete trovare siti similari).

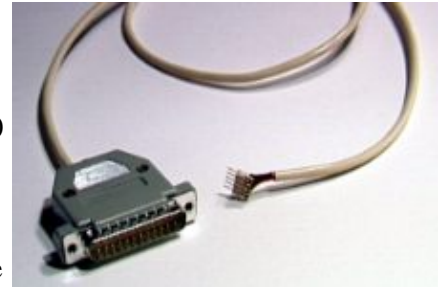
	<p>1 processore RISC Atmel 8 bit Avr, modello AT90S4433.</p>
	<p>Zoccolo per IC 2 x 14 pin o 1 zoccolo 28 pin ampio 7.5mm Lo zoccolo a 28 pin è un poco più complesso da trovare. Di solito gli zoccoli a 28 pin hanno una ampiezza fila/fila di 14mm, ma per il nostro processore ci serve a 7.5.</p>

	<p>1 resistenza da 10K (colori delle bande: marrone,nero,arancio)  3 resistenze da 470 Ohm (colori delle bande: giallo,viola,marrone)  1 resistenza da 1K (colori delle bande: marrone,nero,rosso)  1 resistenza da 220 Ohm (colori delle bande: rosso,rosso,marrone)  1 cristallo di quarzo da 4MHz  2 condensatori ceramici da 27pF</p>
	<p>Un qualsivoglia tipo di connettore a 5pin per interfacciare il programmatore. Di norma, io ricorro ai connettori in striscia (passo 2.54) e spezzo la dimensione che mi interessa, essendo questi a 60 contatti.</p>
	<p>basetta millefori</p>
	<p>1 connettore Canon DB25 (Femmina) per da collegarsi alla porta parallela del PC.</p>
	<p>1 diodo LED</p>
	<p>Una basetta sperimentale. Noi qui, non la si userà, ma potrebbe esservi utile per ulteriori esperimenti con l' AVR. Vi suggerisco di saldare il microcontrollore, il quarzo ed i relativi condensatori sulla basetta millefori e collegare gli ingressi (ingressi ed uscite) per mezzo di spezzoni di cavo alla basetta sperimentale.</p>

In aggiunta alle suindicate parti Vi serve un alimentatore stabilizzato da 5 Volt. Potete anche ricorrere ad una batteria da 4.5 Volt come fonte di energia.

## Costruiamo il programmatore.

Il modello AT90S4433 permette la programmazione in circuito (ISP). Ecco una cosa interessante: non siete obbligati a scollegare il microcontrollore dalla basetta per programmarlo. Potrete notare che un programmatore hardware può esser acquistato, pronto all'uso, per 50–150 Euro. Con Linux, il software uisp ed una porta parallela disponibile nel vostro PC, potrete avere a disposizione un semplice ed affidabile programmatore per l' AVR. Si tratta, in definitiva, di un semplice cavo. Le connessioni del cavo per il programmatore devono seguire il seguente schema:



pin dell' AVR	Pin della Porta parallela
SCK (19)	Strobe (1)
MISO (18)	Busy (11)
MOSI (17)	D0 (2)
Reset (1)	Init (16)
GND	GND (18)

È bene che il cavo non superi i 70cm.

## Scrivere il software.

L'AT90S4433 può esser programmato con il C nel suo dialetto base, con l' aiuto di GCC. Avere le conoscenze dell'assembler AVR può esser utile ma non necessario. Le libc sono fornite con una [scheda avr libc](#) (in lingua inglese) che ne documenta la maggior parte delle funzioni. Harald Leitner ha scritto un documento – in lingua inglese – con moltissimi esempi utili sull'uso dell' AVR e del GCC ([haraleit.pdf, 286Kb](#), l'originale si trova presso <http://www.avrfreaks.net/AVRGCC/>). Dal sito della Atmel potete scaricare il data sheet completo dell'integrato. Dalla [home page \(www.atmel.com\)](#), scegliete avr products → (Microcontrollers) AVR 8 bit RISC → Datasheets. Qui trovare una copia locale del documento ([avr4433.pdf, 2361Kb](#)). Esso descrive tutti i registri e come utilizzare la CPU.

Una cosa che dovrete sempre tenere a mente quando utilizzate il 4433 è che esso possiede solo 128Bytes di SRAM e 4KBytes di EEPROM. Questo implica che non si possano dichiarare strutture di dati o stringhe molto ampie, o che il vostro programma non esegua troppe funzioni annidate o delle ricorsioni. Scrivere una riga come:

```
char string[90];
```

sarà già troppo. Un intero è costituito da 16 bit. Se si serve valore intero piccolo utilizzate:

```
unsigned char i; /* 0–255 */
```

In ogni caso rimarrete sorpresi di quanti programmi potrete scrivere. È un processore veramente potente!

Molto più utile della teoria è un esempio reale. Scriveremo un programma che farà lampeggiare il nostro diodo LED con un intervallo di 0.5 secondi. Non molto utile, ma è pur sempre un buon inizio da cui partire per testare il nostro ambiente di sviluppo ed il programmatore.

```
void main(void)
{
    /* abilita PD5 come uscita */
    sbi(DDRD,PD5);
    while (1) {
        /* led on, pin=0 */
        cbi(PORTD,PD5);
        delay_ms(500);
    }
}
```

```

    /* forza l'output a 5V, ovvero il LED è spento */
    sbi(PORTD,PD5);
    delay_ms(500);
}
}

```

Questo esempio vi mostra come sia semplice scrivere un programma. Potete vedere qui solo la funziona principale (main); la funzione delay\_ms è inclusa nel [listato completo \(avrledtest.c\)](#). Per utilizzare il pin PD5 come segnale d'uscita dovrete settare il flag bit nel registro delle direzioni dei dati per la porta d ( DDRR). Dopo di ciò dovrete settare il PD5 a 0V con la funzione cbi(PORTD,PD5) (clear bit PD5) oppure settarla a 5V con sbi(PORTD,PD5) (set bit PD5). Il valore di "PD5" è definito nel file io4433.h che è incluso da io.h. Non dovrete preoccuparvi di questo. Se avete già scritto programmi per ambienti multi utente o per sistemi multi task, come Linux, allora saprete che un programma non deve MAI avere un loop infinito. Questo indurrebbe ad uno spreco dei tempi della CPU, andando a rallentare sensibilmente il sistema. Nel caso dell'AVR le cose sono diverse. Non abbiamo molteplici task e non vi sono altri programmi in esecuzione. Non abbiamo alcun sistema operativo. È comunque semplice, dopotutto, occuparlo con loop senza fine.

## Compilazione e caricamento del programma

Prima di iniziare verificate di avere la cartella /usr/local/atmel/bin nel vostro PATH. Se necessario modificate .bash\_profile o .tcshrc aggiungendo:

```

export PATH=/usr/local/atmel/bin:${PATH} (se usate la shell bash)
setenv PATH /usr/local/atmel/bin:${PATH} (se usate la shell tcsh)

```

Noi si utilizzerà una porta parallela e uisp per programmare l'AVR. Uisp utilizza l'interfaccia ppdev del kernel. Di conseguenza dovrete avere caricato in memoria i seguenti moduli:

```

# /sbin/lsmmod
parport_pc
ppdev
parport

```

Controllare con il comando /sbin/lsmmod che essi siano effettivamente caricati in memoria. In caso contrario caricateli con i seguenti comandi:

(Vi ricordo che per effettuare questa operazione dovrete utilizzare l'utente root)

```

modeprobe parport
modeprobe parport_pc
modeprobe ppdev

```

È una buona idea eseguire questa serie di comandi automaticamente all'avvio del sistema. Potrete aggiungerli ad uno degli script rc (nel caso di Redhat /etc/rc.d/rc.local).

Per poter utilizzare l'interfaccia ppdev come utente standard del sistema, l'utente root deve darvi il privilegio di scrittura. Questi privilegi possono essere configurati con il seguente comando (basta eseguirlo una solo volta):

```

chmod 666 /dev/parport0

```

Siate sicuri che non vi sia alcun daemon che sta utilizzando la porta parallela. Se avete dei servizi che la stanno utilizzando fermateli prima di collegare il nostro cavo-programmatore. Ora tutto è pronto per la compilazione del nostro microcontrollore.

Il pacchetto ([avrledtest-0.1.tar.gz](#)) per il nostro test contiene al suo interno un make file. Tutto quello che dovrete fare è digitare i comandi:

```
make
make load
```

Questi faranno sì che il software venga compilato e caricato. Non mi addentrerò ulteriormente nella spiegazione dei comandi. Potere vederli, se desiderate, all'interno del [Makefile](#), e noterete che ricorrerete spesso a questi comandi, potremmo dire che sono sempre gli stessi. Sinceramente nemmeno io li ricordo tutti a memoria. Per me è sufficiente sapere che devo digitare "make load". Se volete scrivere un programma diverso, non dovrete fare altro che sostituire tutte le occorrenze di avrledtest con il nome del vostro programma all'interno del Makefile.

## Alcune binutils interessanti

Molto più interessanti dell'attuale processo di compilazione sono alcune binutils..

```
avr-objdump -h avrledtest.out
```

Questo comando mostra la dimensione delle diverse sezioni del nostro programma. *.text* è il codice delle istruzioni che viene caricato nella memoria EEPROM. *.data* è una regione di dati inizializzati; per esempio `static char str[]="hello";`

e *.bss* è un set di dati non inizializzati. Entrambi, nel nostro caso valgono zero. *.eeprom* è per le variabili memorizzate nella eeprom. Io non sono mai ricorso a quest'ultima. *.stab* e *.stabstr* sono utilizzare per informazioni di debug e non sono gestite all'interno dell'AVR.

```
avrledtest.out:      file format elf32-avr

Sections:
Idx Name              Size      VMA           LMA           File off  Algn
  0 .text              0000008c  00000000  00000000  00000094  2**0
                     CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data              00000000  00800060  0000008c  00000120  2**0
                     CONTENTS, ALLOC, LOAD, DATA
  2 .bss               00000000  00800060  0000008c  00000120  2**0
                     ALLOC
  3 .eeprom            00000000  00810000  00810000  00000120  2**0
                     CONTENTS
  4 .stab              00000750  00000000  00000000  00000120  2**2
                     CONTENTS, READONLY, DEBUGGING
  5 .stabstr           000005f4  00000000  00000000  00000870  2**0
                     CONTENTS, READONLY, DEBUGGING
```

Potere utilizzare anche il comando `avr-size` per ottenere questo in un formato più compatto:

```
avr-size avrledtest.out
```

```
text    data    bss     dec     hex filename
 140      0       0     140     8c avrledtest.out
```

Quando si lavora con l'AVR è importante tenere d'occhio che la somma di text, data, bss non sia superiore a 4k e che la somma data, bss, stack (non potete veder la dimensione dello stack, essa dipende da quante funzione annidate avete) non sia superiore a 128Bytes.

Vi sono altri comandi interessanti, come:



avr-objdump -S avrledtest.out

Esso genera un listato assembler del vostro codice.

## Conclusioni

Ora avete di che iniziare per creare i vostri progetti con l'hardware AVR ed il GCC. Seguiranno altri articoli di LinuxFocus con esempi più complessi e molto altro hardware interessante.

## Bibliografia

- Libc e uisp: [/www.amelek.gda.pl/avr/libc/](http://www.amelek.gda.pl/avr/libc/)
- GCC e binutils: <ftp://gatekeeper.dec.com/pub/GNU/>
- avrfreaks (Fate attenzione, vi sono persone che ancora utilizzano windows in questo sito !?): <http://www.avrfreaks.net/>
- l'assembler tavrasm per Linux: [www.tavrasm.org](http://www.tavrasm.org)
- AVR webring: [R.webring.com/hub?ring=avr&list](http://R.webring.com/hub?ring=avr&list)
- Versioni compilate di gcc: [combio.de/avr/](http://combio.de/avr/)
- Tutto il software ed i documenti [citati in questo articolo](#)
- Il sito web della Atmel: [www.atmel.com/](http://www.atmel.com/)

Webpages maintained by the LinuxFocus Editor team

© Guido Socher

"some rights reserved" see [linuxfocus.org/license/](http://linuxfocus.org/license/)

<http://www.LinuxFocus.org>

Translation information:

en --> -- : Guido Socher ([homepage](#))

en --> it: Toni Tiveron <[toni/at/amicidelprosecco.com](mailto:toni/at/amicidelprosecco.com)>