

ATMEL tinyAVR

Programmazione dei microcontrollori “piccoli” di Atmel

***Guida per l'uso dei microcontrollori
della famiglia tinyAVR di Atmel
con l'IDE 1.6.7 di Arduino***

Scritta da Leonardo Miliani

Versione 1.6.7-1

Indice della guida:

1. **Introduzione alle MCU**

1. [Attiny25/45/85](#)
2. [Attiny24/44/84](#)
3. [Attiny2313/4313](#)
4. [Note](#)

2. **Impostare il supporto nell'IDE 1.6.7 di Arduino**

3. **Collegamenti**

4. **Programmazione di un Attiny85**

1. [Sketch di esempio: facciamo lampeggiare 3 LED](#)

5. **Comunicare con la seriale**

1. [Sketch di esempio: comunicazione Attiny/Arduino via seriale](#)

6. **Usare l'I2C**

1. [Sketch di esempio: leggere e scrivere su una EEPROM via I2C](#)

7. **Usare l'SPI**

8. **Conclusioni e ringraziamenti**

9. **Licenza d'uso e garanzia**

1. Introduzione alle MCU

Oltre ai famosi ATmega328P usati nelle schede Arduino UNO, agli ATmega2560 usati sulle Arduino MEGA2560, agli ATmega32U4 usati sulle Arduino Leonardo/Micro/Esplora/Yun e ad altri chip usati in schede Arduino meno recenti o poco diffuse, Atmel produce anche una linea di microcontrollori ad 8 bit dalle ridotte dimensioni, la famiglia **tinyAVR** ("tiny" in inglese significa "piccolo"), offerti in diversi contenitori, fra cui quello DIP, molto pratico per l'uso hobbistico. Essi sono l'ideale strumento da usare in tutti quei progetti in cui non si necessita di un numero elevato di linee di I/O, si hanno problemi di spazio, si necessita di dispositivi dal ridotto consumo energetico, senza però rinunciare alla potenza dei processori maggiori: i tinyAVR possono infatti lavorare con frequenze fino a 20 MHz (usando un quarzo esterno). Possono anche operare a clock di 1 MHz e, in stand-by, arrivare a consumare fino a circa 200 nA (nanoAmpere) di corrente.

I modelli disponibili sono diversi ma non tutti sono utilizzabili con l'IDE di Arduino: esso, infatti, necessita di un apposito set di librerie che non solo permettano all'utente l'accesso alle loro periferiche, ma anche che offrano all'IDE gli strumenti per poterli programmare. Nel gergo Arduino, il set di librerie per gestire un microcontrollore si chiama core, e per i tinyAVR ne esiste uno molto noto, che si chiama "Tiny". Nonostante non offra il supporto di un numero elevato di unità, offre però una buona compatibilità con l'IDE, il che significa poter usare quelle funzioni tipiche dell'Arduino, tipo digitalWrite, analogRead, pinMode, anche con le MCU che supporta. Il core Tiny attualmente supporta le seguenti MCU:

- **Attiny25/45/85**: microcontrollori molto piccoli (contenitore DIP8), con max. 6 linee di I/O (5 "regolari" più 1 pin di reset attivabile anche come I/O)
 - Flash: 2/4/8 KB rispettivamente
 - SRAM: 128/256/512 byte rispettivamente
 - EEPROM: 128/256/512 byte rispettivamente
 - Timer: 1 timer ad 8 bit ed 1 timer a 16 bit
 - ADC a 10 bit a 4 canali
 - Clock: 1/8 MHz con oscillatore interno, 16 Mhz con PLL interno, fino a 20 MHz con quarzo esterno
 - Comunicazione: SPI, USI
 - Assenti in HW: USART/I2C



Atmel ATtiny85

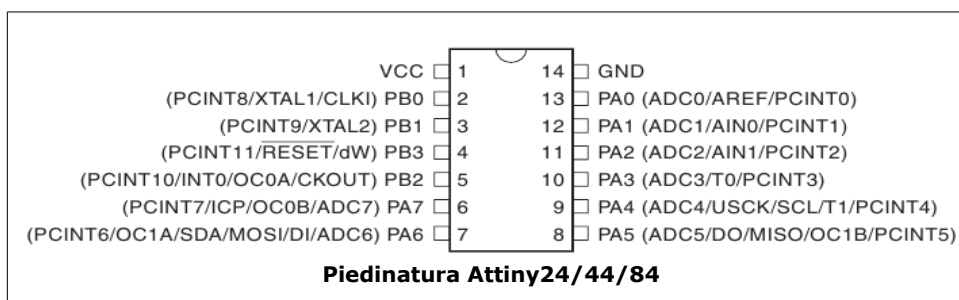


- **Attiny24/44/84:** microcontrollori di dimensioni medio/piccole (contenitore DIP14), con max. 12 linee di I/O (11 "regolari" più 1 pin di reset attivabile anche come I/O)

- Flash: 2/4/8 KB rispettivamente
- SRAM: 128/256/512 byte rispettivamente
- EEPROM: 128/256/512 byte rispettivamente
- Timer: 1 timer ad 8 bit ed 1 timer a 16 bit
- ADC a 10 bit a 8 canali
- Clock: 1/8 Mhz con oscillatore interno, fino a 20 Mhz con quarzo esterno
- Comunicazione: SPI/USI
- Assenti in HW: USART/I2C



Atmel ATTiny84

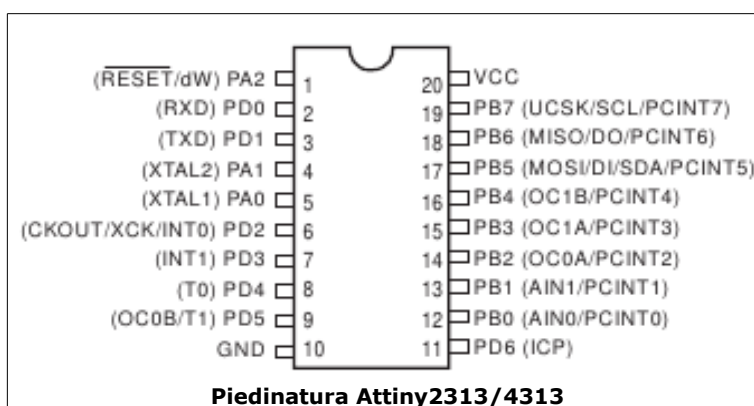


- **Attiny2313/4313:** microcontrollori di dimensioni medie (contenitore DIP20), con max. 18 linee di I/O (17 "regolari" più 1 pin di reset attivabile anche come I/O)

- Flash: 2/4 kB rispettivamente
- SRAM: 128/256 byte rispettivamente
- EEPROM: 128/256 byte rispettivamente
- Timer: 1 timer ad 8 bit ed 1 timer a 16 bit
- Clock: 1/8 Mhz con oscillatore interno, fino a 20 Mhz con quarzo esterno
- Comunicazione: USART/SPI
- Assenti in HW: I2C/ADC



Atmel ATTiny2313



Nota 1:

i microcontrollori Attiny 25/45/85 sono perfettamente identici e differiscono tra loro solo per il diverso quantitativo di memoria. Stesso discorso vale per gli Attiny24/44/84 e per gli Attiny2313/4313.

Nota 2:

la presente guida mostra degli esempi di collegamenti e di programmazione mediante l'uso di un Attiny85. Le stesse informazioni di questo documento sono applicabili anche agli altri microcontrollori indicati, apportando le eventuali minime variazioni (es. i piedini utilizzati e le dichiarazioni nel codice) rispetto a quanto suggerito.

Nota 3:

la presente guida è stata scritta su un sistema Mac OS 10.11 e ad esso si fa riferimento per quanto riguarda il posizionamento dei file ed i comandi da terminale. Ove possibile, sono riportate anche le indicazioni per i sistemi Windows e Linux: dove non presenti, le modifiche sono semplici ed applicabili dall'utente secondo il sistema in uso.

Nota 4:

la presente guida si applica alla versione 1.6.7 o successive dell'IDE di Arduino. Le modifiche sono state collaudate esplicitamente per questa versione dell'IDE: le versioni precedenti dell'IDE sono supportati da vecchie edizioni di questa guida, mentre il ramo 1.0 è supportato da un'altra guida. Entrambe sono scaricabili da [qui](#).

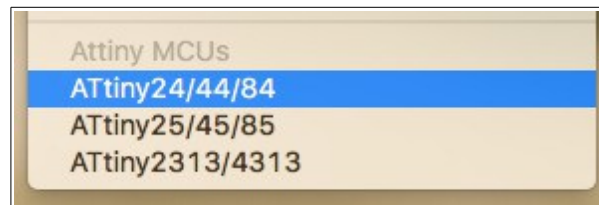
2. Impostare il supporto nell'IDE 1.6.7 di Arduino

Come detto, i microcontrollori tinyAVR analizzati nel precedente capitolo non sono nativamente supportati dall'IDE di Arduino per cui bisogna effettuare delle modifiche software affinché possano essere utilizzati con l'ambiente di Arduino. Per far ciò dobbiamo scaricare un set di librerie definito "**core**" che permette di programmare questi microcontrollori con il software di Arduino e che supporta nativamente le funzioni principali, offrendo all'utente la possibilità di usare le comuni funzioni di Arduino.

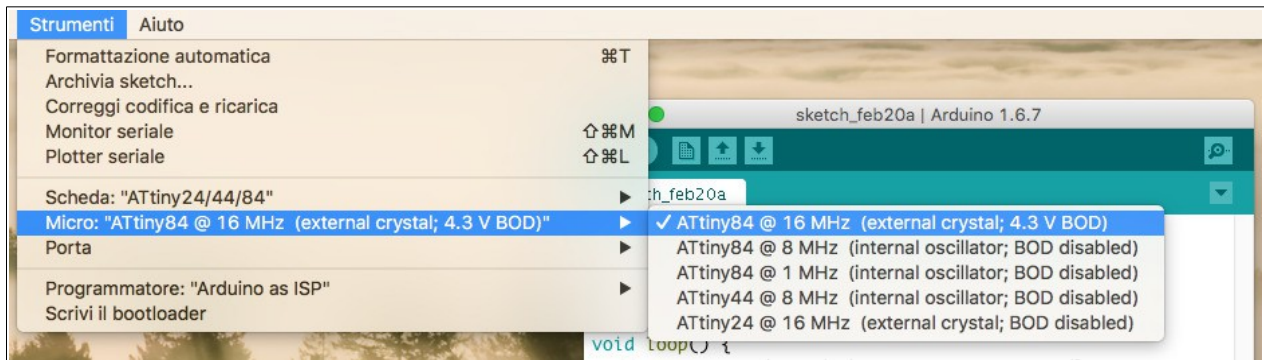
Per l'installazione bisogna seguire alcuni passi:

1. Chiudete l'IDE. E' importante eseguire tutti i passaggi con l'IDE non in esecuzione, altrimenti il software non vede gli aggiornamenti alle librerie indicate.
2. Entrate nella cartella utilizzata per salvare i vostri sketch: se non sapete dove risiede la cartella che contiene i vostri sketch, selezionate dall'IDE la voce "*File/Preferenze*" (o "*Arduino/Preferenze*", a seconda del sistema): la prima riga che leggete nella finestra che si aprirà è quella relativa al percorso dove risiedono i vostri sketch (ricordatevi di richiudere l'IDE). Adesso spostatevi all'interno della cartella di nome `/hardware`: se non c'è, createla.
3. Adesso dobbiamo installare il core allegato a questa guida: non dovete fare altro che copiare la cartella `/tiny` all'interno della cartella che avete appena raggiunto al precedente punto.
4. Al termine dell'operazione, controllate che la cartella `/tiny/avr` contenga i seguenti file:
 1. `boards.txt`: contiene le definizioni per i nuovi microcontrollori. Questo file è presente solo se avete utilizzato il core allegato, altrimenti nel core originale trovate il file `Prospective boards.txt`, che contiene tutti i chip supportati e che dovete utilizzare come base per creare il vostro file `boards.txt`. Attenzione: il file allegato presenta i chip raggruppati per famiglia: nel resto della guida si farà riferimento a questa disposizione; se create il vostro file `boards.txt`, tenete conto delle differenze nelle voci dei menu;
 2. `platform.txt`: contiene le indicazioni necessarie all'IDE per poter fare l'upload degli sketch e programmare i fuse dei microcontrollori;
 3. `avrdude.conf`: è il file che istruisce il programma avrdude che esegue la scrittura degli sketch sui microcontrollori. La versione allegata corregge un bug presente nel file allegato all'IDE.
 4. cartelle `/bootloaders`, `/cores`, `/libraries`: contengono i file per supportare nell'IDE e programmare i chip;
 5. `README`, `license.txt`: file informativi.
5. Adesso copiate la cartella `/ATtiny` all'interno della vostra cartella degli sketch (vedi punto 2).

Adesso aprite l'IDE 1.6.7: se avete eseguito tutti i passaggi correttamente, dovrete avere sotto al menu "*Strumenti/Board*" 3 nuove voci:



Selezionate una linea di microcontrollori, ad esempio "Attiny 24/44/84", e poi riaprite il menu "Strumenti": vedrete comparsa una nuova voce, "Micro", all'interno della quale potrete selezionare la MCU della linea precedentemente scelta:



Controlliamo anche se gli sketch di test sono presenti. Aprite "File/Cartella degli sketch": dovrete trovare il nuovo sottomenu "Attiny" con all'interno altre voci, come in figura:



3. Collegamenti

Per poter programmare un microcontrollore della famiglia Attiny dobbiamo utilizzare la comunicazione ISP (In-System Programming) dato che questi chip non riservano un'area per accogliere un bootloader come l'ATmega328P dell'Arduino UNO né possiedono il supporto per la connessione seriale in hardware. La programmazione ISP sfrutta la periferica SPI (Serial Peripheral Interface) integrata in questi controllori (è presente anche negli ATmega328P). Due dispositivi possono dialogare tramite SPI usando solo 3 linee, MOSI, MISO e SCK, che vanno collegate accoppiando quelle del dispositivo da programmare con quelle del dispositivo che programma. Per la programmazione noi utilizzeremo una normale scheda Arduino UNO come programmatore ISP grazie allo sketch *ArduinoISP* integrato nell'IDE:

1. collegate la vostra scheda Arduino al computer;
2. aprite l'IDE di Arduino;
3. selezionate da *"File/Esempi"* lo sketch *"ArduinoISP"*;
4. selezionate da *"Strumenti/Board"* la scheda *"Arduino UNO"* e poi da *"Strumenti/Porta"* la porta a cui è connessa la scheda. Questa dipende dal sistema in uso: su Windows sarà del tipo `COM10`; su Linux sarà del tipo `/dev/ttyACM0`; su Mac OS X sarà del tipo `/dev/tty.usbmodem14211`;
5. cliccate sull'icona di caricamento oppure scegliete *"File/Carica"* per trasferire lo sketch sull'Arduino UNO;
6. al termine dell'operazione scollegate la scheda dal computer.

Il vostro Arduino ora contiene il software capace di emulare un programmatore STK500, grazie al quale potete utilizzare la scheda per scrivere gli sketch sull'Attiny che sceglierete.

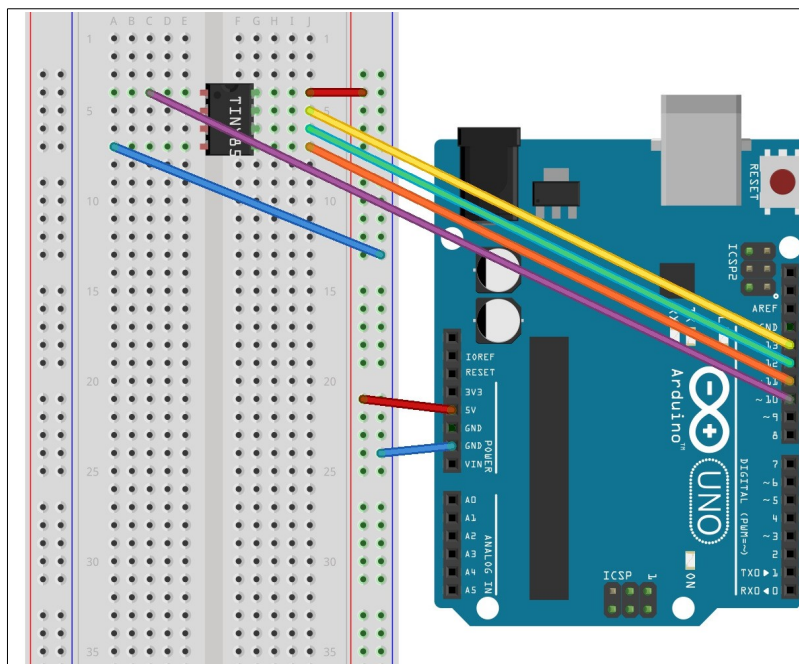
Prestate attenzione al fatto che se usate una versione della scheda precedente alla UNO R3 potreste avere dei problemi di autoreset, ossia al momento dell'invio dello sketch all'Attiny l'Arduino si resetterà al posto del chip da programmare. Questo problema è dato dalla versione del bootloader installato: l'Arduino UNO R1 e le prime UNO R2 montavano l'Optiboot 4.0, che era afflitto da questo problema, mentre le UNO R2 più recenti e tutte le UNO R3 montano l'Optiboot 4.4 che non soffre di questo bug. La soluzione definitiva è quella di scrivere il bootloader nuovo (contenuto nell'IDE) sulla scheda, ma per questa operazione vi serve un'altra scheda Arduino oppure un programmatore di terzi (tipo USBtinyISP o altro). Una soluzione temporanea è quella di utilizzare un condensatore elettrolitico da 10 uF da inserire nei pin RST (catodo del condensatore) e 5V (anodo del condensatore), da usare soltanto quando si spedisce lo sketch all'Attiny e da togliere subito dopo (blocca il reset dell'Arduino, rendendolo non programmabile).

Munitevi adesso di una breadboard e di alcuni cavetti per effettuare i collegamenti. Ricreate le connessioni rappresentate nella figura più sotto (aiutatevi con i colori).

Nota bene:

Ogni microcontrollore ha le linee MOSI/MISO/SCK disposte su pin differenti. Controllate il datasheet per il modello in uso ed adeguate i collegamenti (i pin MOSI/MISO/SCK sono indicati nella piedinatura del chip presente nelle prime pagine del datasheet).

Connessione	Arduino UNO	Attiny85
VCC	Pin 5V	Pin 8
GND	Pin GND	Pin 4
MOSI	Pin 11	Pin 5
MISO	Pin 12	Pin 6
SCK	Pin 13	Pin 7
RESET	Pin 10	Pin 1



La prima operazione da effettuare è sempre quella di impostare i *fuse* del microcontrollore per selezionare la frequenza desiderata. A seconda del microcontrollore utilizzato, si possono selezionare diverse frequenze. Il core Tiny normalmente offre le scelte a 1 e 8 Mhz per il semplice motivo che queste velocità sono generate utilizzando l'oscillatore interno, quindi senza l'uso di quarzi esterni che andrebbero ad occupare 2 dei già pochi pin che questi chip offrono. Inoltre questi chip sono spesso usati in progetti alimentati a batteria, dove frequenze inferiori permettono di usare tensioni di alimentazioni più basse, con evidente risparmio energetico.

Per selezionare la frequenza desiderata basta "scrivere il bootloader sul microcontrollore": ho virgolettato questa frase perché, come detto, in realtà i micro della famiglia Attiny non riservano un'area di memoria per il bootloader. Questa operazione viene effettuata solo per modificare i *fuse* direttamente dall'IDE, ricorrendo all'uso di un bootloader vuoto che serve ad ingannare l'IDE di Arduino. Esempio: impostiamo un Attiny85 a 8 Mhz.

1. Realizziamo i collegamenti come nello schema su esposto;
2. colleghiamo l'Arduino al computer (su di esso avremo già caricato lo sketch ArduinoISP);
3. apriamo l'IDE e da "Strumenti/Board" selezioniamo la linea di MCU "Attiny 25/45/85";
4. da "Strumenti/Micro" selezioniamo la voce "Attiny85@8 Mhz (internal oscillator; BOD disabled)";
5. da "Strumenti/Programmatore" selezioniamo la voce "Arduino as ISP": attenzione a non fare confusione con la voce "ArduinoISP", che si riferisce alla nuova schedina programmatrice rilasciata da Arduino;
6. da "Strumenti" selezioniamo la voce "Scrivi il bootloader"

L'IDE a questo punto effettuerà l'operazione di caricamento del finto bootloader sul chip Attiny85, durante la quale imposterà anche i *fuse*. Questa operazione serve inoltre a cancellare completamente (*erase*) la memoria del microcontrollore: se volete quindi effettuare una ripulitura completa di tutta la flash, potete riscrivere il bootloader sul microcontrollore (ricordo che avrdude programma solo la Flash realmente occupata dal vostro programma, lasciando inalterato il contenuto delle celle non occupate).

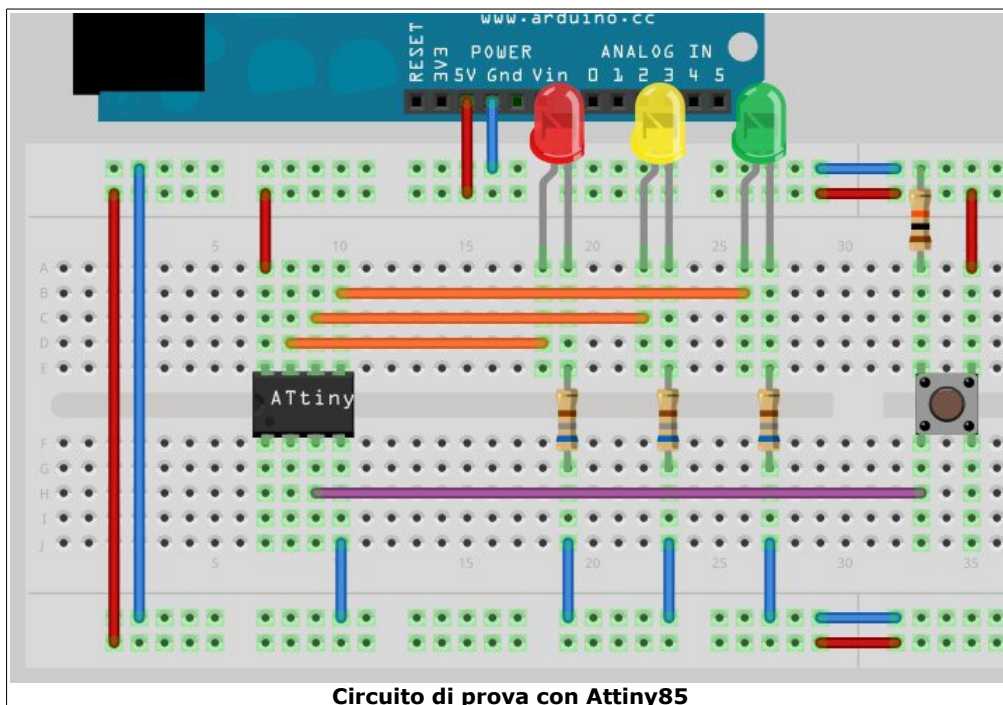
4. Programmazione di un ATtiny85

Adesso siamo pronti ad usare i micro tinyAVR per i nostri progetti tramite l'IDE di Arduino, con cui poter scrivere il codice ed inviarlo al chip con semplici click.

Iniziamo con un piccolo progetto, un circuito con 3 LED il cui schema di lampeggio può essere cambiato premendo un pulsantino, il tutto gestito da un Attiny85. Lo sketch che dobbiamo caricare è quello selezionabile da "File/Cartella degli sketch/Attiny/test_3_led".

Controllate da "Strumenti/Micro" di aver ancora selezionata la voce "Attiny85 @ 8 Mhz" (se così non fosse, risSelectedatela), poi premete il pulsante "Carica" (oppure selezionate la corrispondente voce dal menu "File").

Adesso scollegate l'Arduino, poi togliete i cavetti della programmazione ISP e realizzate il circuito rappresentato in figura:



Vi occorrono:

- 1 pulsantino da montaggio su PCB;
- 3 resistenze per i LED (io ho usato il valore di 680 ohm);
- 1 resistenza da 10 Kohm;
- 3 LED, di 3 colori differenti, es.: rosso, giallo, verde.

Collegate i 3 LED così:

- LED rosso: pin positivo al pin 7 dell'ATtiny85, pin negativo a GND tramite resistenza da 680 Ohm;
- LED giallo: pin positivo al pin 6 dell'ATtiny85, pin negativo a GND tramite resistenza da 680 Ohm;

- LER verde: pin positivo al pin 5 dell'ATtiny85, pin negativo a GND tramite resistenza da 680 Ohm;

Il pulsantino va collegato come nello schema, con la resistenza di pull-up a GND sulla stessa linea del pin che collegate al pin 3 dell'ATtiny85, mentre uno degli altri pin va collegato a +5V. Adesso ricollegate l'Arduino al PC, in modo da prelevare dal bus USB la tensione necessaria ad alimentare anche il circuito sulla breadboard.

Cosa si vede?

Il primo LED, quello rosso, lampeggia.

Cosa si può fare?

Premendo il pulsantino, inizia a lampeggiare il LED giallo. Un'altra pressione porta al lampeggio del LED verde. Ancora una pressione ed i 3 LED lampeggiano in sequenza. Premendolo altre 2 volte si può scegliere se tenere tutti i LED accessi fissi oppure spenti.

Come funziona il codice?

Il codice legge la pressione del pulsantino e modifica il valore di una variabile che seleziona la modalità di lampeggio. Per il lampeggio è usato il calcolo del tempo mediante `millis()` e non la funzione `delay()`, che bloccherebbe l'esecuzione del codice rendendo di fatto impossibile avere contemporaneamente il lampeggio dei LED e la lettura del pulsantino.

5. Comunicare con la seriale

Dopo questo primo esempio, che ci ha fatto imparare come collegare l'ATtiny85 e come programmare il micro tramite scheda Arduino, procediamo con un altro esempio, facendo in modo che l'ATtiny85 comunichi usando la seriale. Per questo scopo utilizzeremo ancora un Attiny85 perché questo microcontrollore, come i suoi fratelli Attiny25 e Attiny45 nonché gli Attiny24/44/84, non possiedono il supporto per l'USART (seriale) in hardware, che solo l'Attiny2313/4313, tra i microcontrollori supportati dal core Tiny, possiede: ciò significa che per comunicare tramite seriale dobbiamo implementare il protocollo via software.

Per far ciò ci viene in soccorso la libreria **SoftwareSerial**, di serie nell'IDE, che fa proprio questo: emula via software la comunicazione seriale normalmente gestita via hardware. La libreria va però modificata perché di serie presuppone di lavorare con microcontrollori che hanno più di 1 timer ad 8 bit ma siccome l'Attiny85 ne ha solo uno, il codice restituisce un errore. Un altro problema nasce dal fatto che la libreria non supporta in ricezioni i microcontrollori Attiny per cui dobbiamo inserire un'ulteriore modifica al suo codice affinché essa possa funzionare in maniera completa.

Nota bene:

la SoftwareSerial lavora esclusivamente con il microcontrollore a 8 Mhz, mancano al suo interno le specifiche per operare ad 1 Mhz.

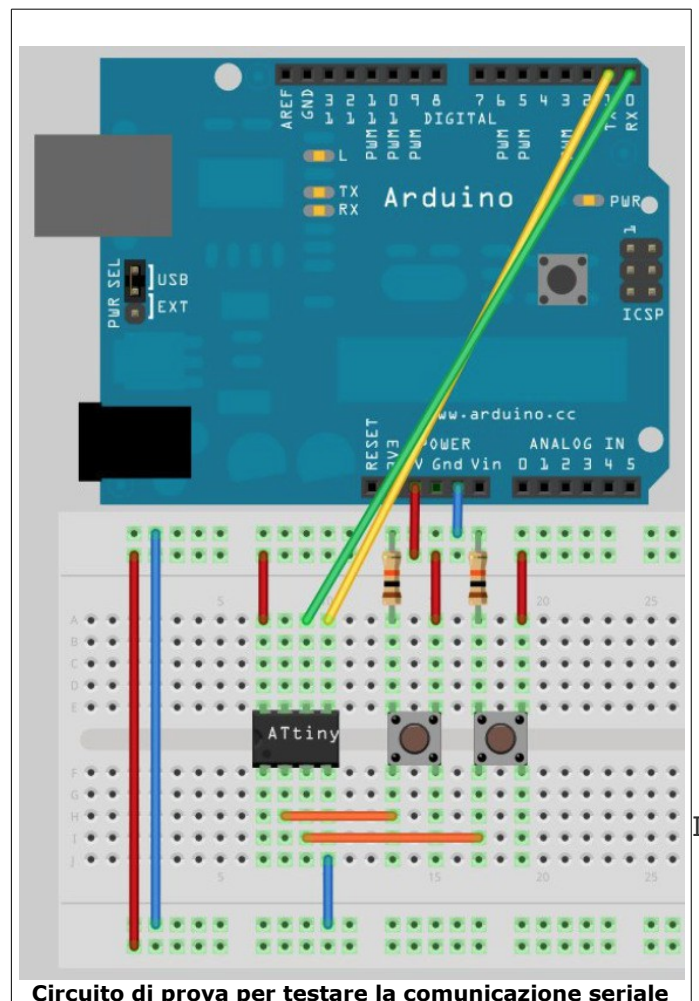
Ricollegiamo l'Attiny85 per la programmazione ISP e carichiamo sul chip lo sketch denominato *test_serial.ino*.

Prepariamo un semplice circuito sulla breadboard come quello qui a lato. Servono 2 pulsantini e 2 resistenze da 10 Kohm.

Collegamenti:

- collegare i pulsantini ai pin 2 e 3 dell'ATtiny85, ognuno con la resistenza di pull-down a GND;
- collegare poi il pin 5 dell'ATtiny85 al pin digitale 1 dell'Arduino (quello con la scritta TX) ed il pin 6 dell'ATtiny85 con il pin digitale 0 dell'Arduino (RX);
- collegare +5V e GND come nell'esempio

La libreria SoftwareSerial può operare su qualsiasi pin del micro: noi, per comodità, useremo i pin liberi più vicini all'Arduino ma nulla toglie di scegliere quelli che più aggradano. I pulsantini serviranno per inviare dei dati diversi all'Arduino.



Lo sketch è selezionabile da "*File/Cartella degli sketch/Attiny/test_seriale_tiny*": è molto semplice e non fa altro che mettersi in ascolto sulla seriale e stampare i caratteri che riceve su di essa e va caricato come visto in precedenza. Adesso scollegiamo l'Arduino dal PC, scollegiamo la breadboard dall'Arduino e carichiamo nell'IDE lo sketch che si chiama *Attiny/test_serial_arduino*. Colleghiamo l'Arduino, selezioniamo "Arduino UNO" dal menu Board e poi facciamo l'upload dello sketch.

Scollegiamo nuovamente la scheda, attacchiamo il circuito della breadboard all'Arduino e poi ricollegiamo l'Arduino al PC. Apriamo un terminale (va bene anche quello dell'IDE di Arduino) e premiamo i pulsantini sulla breadboard. Se tutto è collegato bene, vedremo comparire caratteri diversi ("1" oppure "2") alla pressione dei 2 pulsantini.

6. Usare l'I2C

L'I2C (più precisamente I²C) è un bus sviluppato da Philips per permettere la comunicazione tra più dispositivi tramite l'uso di 2 linee comuni. Ogni dispositivo è identificato all'interno del bus mediante un indirizzo, o ID, che indirizza in maniera univoca il dispositivo. Ogni dispositivo può operare sia come "master" (colui che richiede i dati) sia come "slave" (colui che riceve la richiesta di dati). Sfortunatamente i micro della famiglia Attinyx5 non supportano in hardware l'I2C ma hanno invece un'interfaccia chiamata USI (Universal Serial Interface) che può emulare l'I2C mediante il protocollo 2-Wire (o TWI), molto simile all'I2C.

In questo caso ci viene in aiuto la comunità, che ha modificato la libreria Wire per renderla compatibile con i micro Attiny producendo la libreria [TinyWire](#). Per poter però usare gli Attiny col clock maggiore, la TinyWire è stata modificata perché presuppone di lavorare con gli Attiny di serie, quindi solo con il clock a 1 Mhz e non anche con quello a 8 Mhz permesso dai tinyAVR. Inoltre questa libreria non supporta di serie gli ATtiny84 per cui un'altra modifica si è resa necessaria.

Le librerie TinyWireM e TinyWireS incluse nel core contengono già le modifiche indicate: sono da usarsi rispettivamente nel caso si voglia impostare l'ATtiny come "master" (M) oppure come "slave" (S).

Nota bene:

di default, i buffer di ricezione e trasmissione delle librerie TinyWireM e TinyWireS sono preimpostati rispettivamente a 16 e 32 byte. Tenete a mente questi valori perché la creazione dei buffer va ad occupare spazio nella SRAM e, visto il limitato quantitativo di memoria dei micro della famiglia Tiny, se aveste problemi di esaurimento della SRAM uno degli indiziati potrebbe proprio essere uno dei buffer. Se volete modificare questi valori intervenite nei seguenti file:

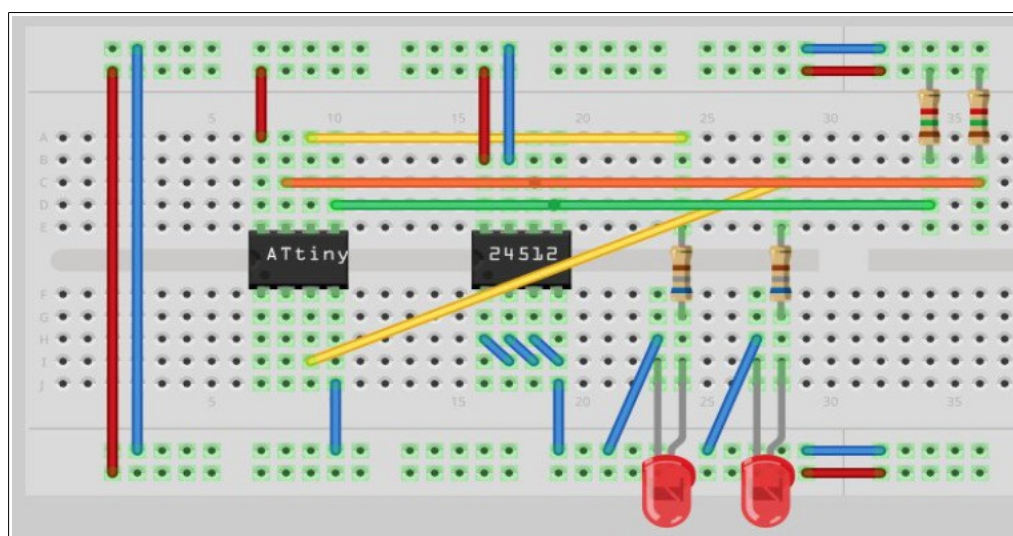
- TinyWireM: file **TinyWireM.h**
 - modificare il valore evidenziato nella seguente riga:
`#define USI_BUF_SIZE 16 // bytes in message buffer`
- TinyWireS: file **usiTwISlave.h**
 - modificare i valori evidenziati nelle seguenti righe:
`#define TWI_RX_BUFFER_SIZE (32)`
`#define TWI_TX_BUFFER_SIZE (32)`

Per provare la comunicazione I2C useremo un chip di memoria EEPROM 24LC512 (va bene anche un taglio inferiore, come un 24LC256 o un 24LC128). Procuriamoci i seguenti componenti: 2 LED, 2 resistenze per i LED (io le ho usate da 680 per sicurezza) e 2 resistenze da 1,5 Kohm. Eseguiamo i collegamenti:

- Attiny:
 - pin 8 a +5V
 - pin 4 a GND
- 24LC512:
 - pin 1, 2, 3, 4 e 7 a GND

- pin 8 a +5V
- LED1:
 - pin positivo a pin 6 dell'ATtiny con resistenza da 680 Ohm in serie
 - pin negativo a GND
- LED2:
 - pin positivo a pin 3 dell'ATtiny con resistenza da 680 Ohm in serie
 - pin negativo a GND
- Bus I2C:
 - collegare il pin 5 dell'ATtiny al pin 5 del 24LC512 e poi a +5V tramite resistenza di pull-up da 1,5 Kohm
 - collegare il pin 7 dell'ATtiny al pin 6 del 24LC512 e poi a +5V tramite resistenza di pull-up da 1,5 Kohm

Ecco uno schema dei collegamenti:



Adesso carichiamo sull'Attiny lo sketch denominato *Attiny/test_i2c* sull'Attiny usando l'Arduino come programmatore ISP e poi osserviamo come si comportano i LED.

All'avvio dello sketch, il micro accende entrambi i LED per una breve frazione di secondo per informare l'utente che è partito il programma. A questo punto lo sketch scrive nei primi 10 byte della EEPROM dei valori di test. Poi vengono spenti entrambi i LED e riaccessi solo il secondo, ad informare l'utente che è iniziato il ciclo contenuto in `loop()`. Questo rilegge ad intervalli il contenuto delle celle della EEPROM programmate in precedenza ed emette un breve lampeggio dal primo LED nel caso in cui l'informazione letta sia il valore "1" oppure tenendo spento il LED nel caso legga "0". Dopo la lettura del 10 byte il ciclo riparte.

7. Usare l'SPI

Sui microcontrollori della famiglia Attiny non c'è una periferica SPI vera e propria, la gestione delle connessioni Three-Wire (a 3 fili) compatibili con l'SPI è fatta dalla periferica USI, già vista nel precedente capitolo circa l'I2C/Two-Wire. La periferica USI va gestita via software ma, per fortuna, esiste una libreria apposita sviluppata per gestire la periferica come SPI: la libreria si chiama [tinyISP](#) ed è stata originariamente scritta da Jack Christensen. Io l'ho modificata aggiungendo il supporto agli Attiny2313/4313.

La libreria permette la gestione della comunicazione SPI per il momento solo con il microcontrollore impostato come master. Inoltre non supporta il pin SS (Slave Select) che, se necessario, andrà implementato via software.

La libreria permette una comunicazione discretamente veloce, a detta dell'autore siamo a circa 15 volte la velocità ottenibile dalla funzione shiftOut e mediamente ad un clock di circa 1/10 di quello del sistema (se la MCU viaggia ad 1 MHz, la tinyISP spedisce i dati a circa 100 kHz). Per usare la tinyISP l'autore fornisce un paio di esempi, disponibili nell'IDE sotto "*Esempi/tinySPI-master*". Per questioni di memoria, se userete la libreria con MCU Attiny con meno di 4 kB di Flash, non vi resterà molto spazio per il vostro codice.

8. Conclusioni e ringraziamenti

I microprocessori della famiglia Attiny sono ottimi chip dalle interessanti caratteristiche: offrono un'ottima potenza di calcolo (supportano un clock massimo di 20 Mhz, uguale a quello dei fratelli maggiori Atmega) in un package dalle ridotte dimensioni. L'oscillatore interno ad 8 Mhz permette di recuperare 2 dei 6 pin di I/O del micro, cosa molto importante considerato appunto il ridotto numero di linee di comunicazione dell'Attiny.

Questo micro soffre, a mio avviso, di una scarsa diffusione: eppure le potenzialità ci sono. L'unico appunto è il non adeguato supporto, situazione parzialmente risolta dalle librerie sviluppate dalla comunità. Che però difettano di una documentazione chiara, costringendo, come si è visto, a modifiche da parte dell'utente trovate alle volte per pura fortuna e dopo innumerevoli tentativi.

Si ringrazia:

- l'utente PaoloP del forum di Arduino perché ha dato un importante contributo modificando in diverse parti i file di configurazione per rendere compatibile il core Tiny con le vecchie versioni dell'IDE, lavoro riutilizzato anche per la presente guida;
- l'utente Coding Badly del forum di Arduino per aver scritto il core Attiny;
- BroHogan per aver scritto la libreria TinyWire;
- Jack Christensen per aver scritto la libreria TinyISP;
- lo staff di Arduino per aver realizzato questo magnifico progetto.

9. Licenza d'uso e garanzia

Il presente documento è rilasciato sotto licenza **Creative Commons Italia Attribuzione-Condividi allo stesso modo-Non commerciale 4.0**. Ciò significa che sei libero di:

- riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera;
- modificare quest'opera

Alle seguenti condizioni:

- Attribuzione – Devi attribuire la paternità dell'opera nei modi indicati dall'autore o da chi ti ha dato l'opera in licenza e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
- Non commerciale – Non puoi usare quest'opera per fini commerciali.
- Condividi allo stesso modo – Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.

Nota bene:

ogni volta che usi o distribuisce quest'opera, devi farlo secondo i termini di questa licenza, che va comunicata con chiarezza. La copia completa della licenza è reperibile a questo indirizzo:

http://www.creativecommons.it/ccitfiles/cc_by_nc_sa_4.0_it_consultazione_pubblica.pdf

Nota bene 2:

i file contenuti nell'allegato sono distribuiti secondo le licenze dei rispettivi autori per quanto riguarda il core Tiny e le librerie, secondo la licenza Creative Commons Italia Attribuzione-Condividi allo stesso modo-Non commerciale 4.0 o successiva per quanto riguarda gli sketch per Arduino scritti dall'Autore della presente.

Nota bene 3:

tutto il materiale è rilasciato "così com'è", senza nessun tipo di garanzia né responsabilità, diretta o indiretta, derivanti dall'utilizzo di qualsiasi informazione contenuta in questa guida o nei file allegati, che saranno di esclusiva pertinenza dell'utilizzatore.

Guida scritta da [Leonardo Miliani](#)

Versione 1.0 – Data di prima stesura: 10/04/2011

Versione 1.2 – Data di revisione: 17/02/2012

Versione 1.5-04 – Data di revisione: 11/11/2013

Versione 1.5.7 – Data di revisione: 14/07/2014

Versione 1.5.8 – Data di revisione: 13/10/2014

Versione 1.5.8-2 – Data di revisione: 29/11/2014

Versione 1.6.7-1 – Data di revisione: 20/02/2016