ATMEL AVR ATtiny

I microcontrollori “piccoli” di Atmel

Guida alla programmazione valida per l'uso con le versioni 0022 dell'IDE di Arduino e del core Tiny

Scritta da Leonardo Miliani

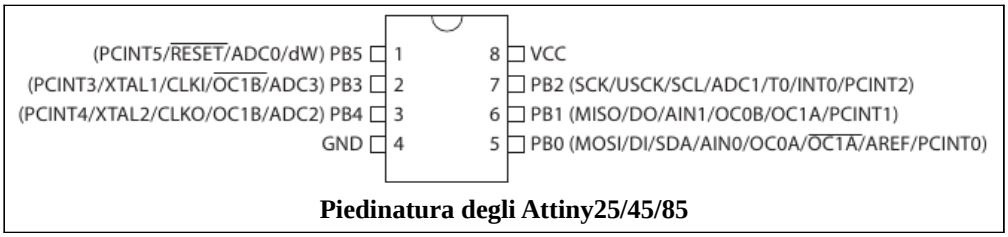
Indice della guida:

1. Introduzione

Oltre ai famosi ATmega168/328 usati nelle schede Arduino, Atmel produce anche una linea di microcontrollori ad 8 bit dalle ridotte dimensioni, gli **ATtiny** (“tiny” in inglese significa “piccolo”), offerti in diversi package, fra cui il DIP, molto pratico per l'uso hobbistico. Essi sono l'ideale in tutti quei progetti in cui non si necessita di un numero elevato di linee di I/O o si hanno problemi di spazio, senza però voler rinunciare alla potenza dei processori maggiori: gli Attiny possono infatti lavorare con clock fino a 20 MHz (usando un quarzo esterno). I modelli disponibili sono diversi ma noi analizzeremo solo i componenti di 3 linee, che sono quelle utilizzabili tramite un apposito set di librerie, all'interno dell'IDE di Arduino:

- **Attiny25/45/85**: microcontrollori molto piccoli (package 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 rispettivamente
  - 1 timer ad 8 bit ed 1 timer a 16 bit
  - ADC a 10 bit e 4 canali
  - clock: 1/8 Mhz con oscillatore interno, fino a 20 Mhz con quarzo esterno
  - Assenti in HW: Seriale/I2C

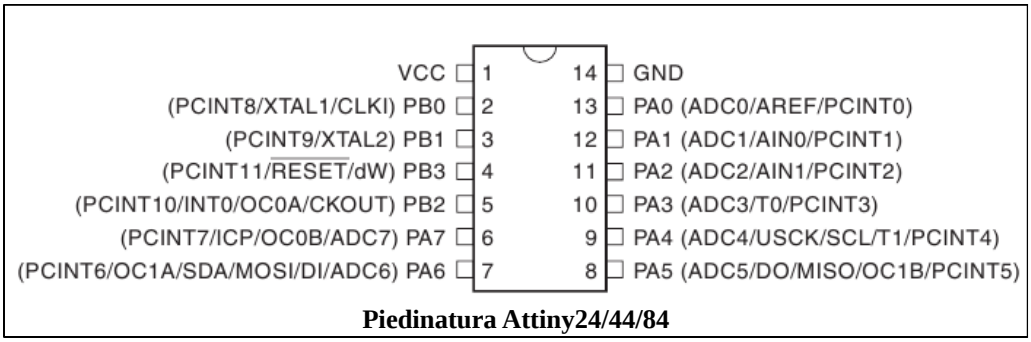
Atmel ATtny85



Piedinatura degli Attiny25/45/85

- **Attiny24/44/84**: microcontrollori di dimensioni medio/piccole (package 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
  - 1 timer ad 8 bit ed 1 timer a 16 bit
  - ADC a 10 bit e 8 canali
  - clock: 1/8 Mhz con oscillatore interno, fino a 20 Mhz con quarzo esterno
  - Assenti in HW: Seriale/I2C

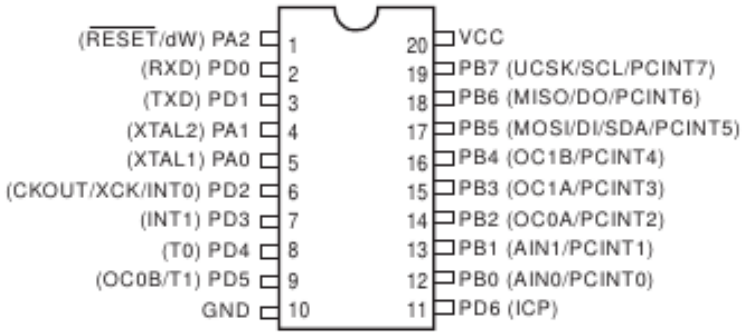
Atmel ATtiny84



Piedinatura Attiny24/44/84

- **Attiny2313**: microcontrollori di dimensioni medie (package DIP20), con max. 18 linee di I/O (17 “regolari” più 1 pin di reset attivabile anche come I/O)
  - Flash: 2 kB
  - SRAM: 128 byte
  - EEPROM: 128 byte
  - 1 timer ad 8 bit ed 1 timer a 16 bit
  - Seriale
  - clock: 1/8 Mhz con oscillatore interno, fino a 20 Mhz con quarzo esterno
  - Assenti in HW: I2C/ADC

Atmel ATtiny2313



Piedinatura Attiny2313

Nota:
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.

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) rispetto a quanto suggerito.

Nota 3:
la presente guida è stata scritta prendendo in esame un sistema così configurato:

- Kubuntu Linux 11.04 a 32 bit
- Arduino IDE versione 0022
- core Tiny versione 0022

Le informazioni riportate si applicano anche ad altri sistemi (Windows, MacOS) tenendo conto delle dovute e piccole modifiche da applicare circa i percorsi delle cartelle (le *path*) e la sede delle stesse cartelle.

Impostare il supporto nell'IDE di Arduino

Questi micro 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 direttamente dall'IDE, supportando le loro funzioni (timer, interrupt, pin) ed offrendo nei menu le voci per poterli gestire.

Il core che dobbiamo scaricare è detto **Tiny**, ed è prelevabile da questo link:
http://arduino-tiny.googlecode.com/files/arduino-tiny-0022-0008.zip

Il file che serve è quello denominato *arduino-tiny*, che contiene il core vero e proprio. Gli altri 2 sono opzionali. Analizziamoli in dettaglio:

- **arduino-tiny-0022-0008** (la versione può cambiare nel tempo): contiene il set di librerie (core) vero e proprio. La versione indicata è compatibile con l'IDE 0022
- **PinChangeInterrupt-0001**: contiene una libreria per poter gestire gli interrupt relativi ai cambi di stato dei pin (PcINT)
- **TinyTuner-0003**: serve per calibrare l'oscillatore interno dei microcontrollori Attiny25/45/85 per ottenere frequenze superiori ad 8 MHz (non sperimentato)

Scaricati i file, inserite il contenuto del pacchetto arduino-tiny-xxxx-xxxx.zip all'interno della cartella /hardware ed il contenuto degli altri 2 archivi nella cartella /libraries, entrambe contenute nella vostra cartella degli sketch: quest'ultima è generalmente /home/utente/sketchbook/ (sotto Linux) o /Documenti/Arduino (sotto Windows). Se le suddette cartelle /libraries e /hardware/ non sono presenti, createle.

Adesso dobbiamo modificare un file affinché sia possibile programmare gli Attiny dall'IDE:

1. apriamo il file /tiny/boards.txt;
2. commentinamo tutte le voci che terminano con **.upload.using=pololu**
3. decommentiamo tutte le righe che terminano con **.upload.using=arduino:arduinoisp**

Avviate ora l'IDE di Arduino: se tutto è stato fatto correttamente, compariranno sotto il menu “Tools/Board” diverse nuove voci riguardanti l'Attiny24/44/84, l'Attiny25/45/85 e l'Attiny2313. Se tali voci non sono presenti, i file non sono stati posizionati correttamente e l'IDE non può ancora supportare i nuovi microcontrollori quindi dovete ricontrollare i passaggi precedenti.

Attiny a 1 oppure 8 Mhz

Gli Attiny hanno un oscillatore interno a 8 MHz ed un divisore x8 che può essere abilitato oppure no. Di fabbrica tutti i microcontrollori Atmel escono con tale divisore abilitato per cui la frequenza a cui lavorano i micro nuovi è di **1 MHz**. Essa può essere più che sufficiente per molti progetti ma, dato che la modifica è semplice e la maggior velocità può risultare utile in diversi frangenti, io consiglio di riprogrammare i “*fuse*“del micro per disabilitare tale divisore (i fuse sono particolari registri modificabili solo esternamente che controllano alcune funzionalità dei microcontrollori). Per far ciò è necessario usare un programmatore ISP esterno: va bene qualsiasi dispositivo suppotato da avrdude, il software che andremo ad utilizzare per l'operazione, come l'USBtinyISP di Adafruit oppure lo stesso Arduino UNO: in quest'ultimo caso basta caricare lo sketch “ArduinoISP” (presente negli esempi contenuti nell'IDE stessa) sull'Arduino

Nota bene:
nel caso si utilizzi l'Arduino UNO, bisogna sapere una cosa: le prime versioni della scheda soffrivano del problema dell'autoreset della scheda, ossia che durante il tentativo di programmazione di un Attiny l'Arduino si resettava, impedendo di portare a termine l'operazione. Le versioni dell'Arduino UNO che soffrono di questo problema sono la prima e la seconda, denominata UNO R2 e riconoscibile rispetto all'altra per il chip Atmega8U2 posizionato a 45°. L'ultima versione denominata UNO R3 non soffre di questo problema.

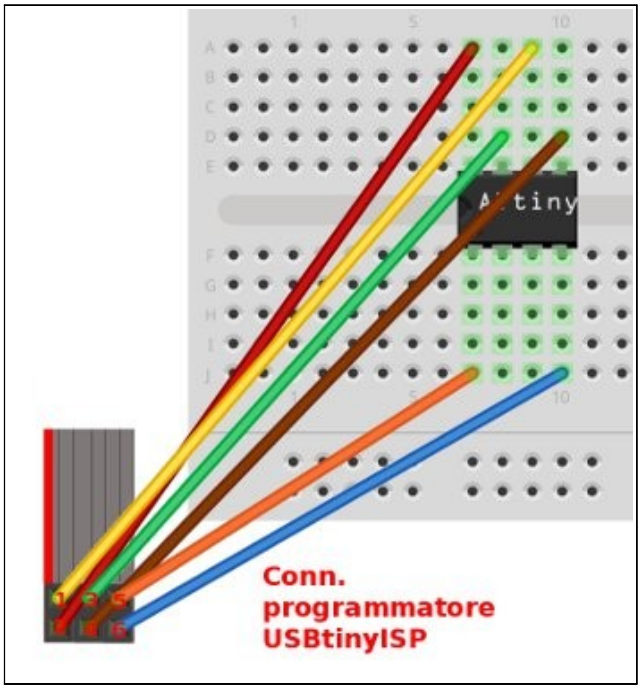
Per risolverlo si può operare in 2 maniere:

1. soluzione temporanea: tenere a portata di mano un condensatore da 10 uF da inserire tra il pin 5V (polo positivo del condensatore) ed un pin GND (polo negativo del condensatore) dell'Arduino per disabilitare

l'autoreset della scheda al momento di utilizzare l'Arduino come programmatore ISP, come si vede nella figura al paragrafo "Programmazione dell'Attiny85" (più sotto).

2. Soluzione definitiva: installare su microcontrollore della propria UNO R1 o UNO R2 il bootloader Optiboot 4.4 contenuto nell'IDE 1.0. Per far ciò serve però un programmatore esterno (tipo l'USBtinyISP) per fare l'operazione direttamente sul micro montato sull'Arduino oppure ricorrere alla tecnica della programmazione ISP di un Atmega328 vergine montato su una breadboard (vedi apposita guida redatta da Michele Menniti e scaricabile da questa pagina).

Per programmare i *fuse* basta montare il chip su una breadboard e provvedere a collegare tramite dei fili (o dei connettori con pin maschio su entrambe le estremità) il microcontrollore al connettore ISP del programmatore. Qui sotto c'è l'esempio di collegamento di un Attiny85 ad un programmatore USBtinyISP, in tabella sono riportati anche i collegamenti rispetto ad un Arduino UNO



SEGNALE	PIN USBTINYISP	PIN ARDUINO UNO	PIN ATTINY
MISO	1	12	6
5V	2	5V	8
SCK	3	13	7
MOSI	4	11	5
RST	5	10	1
GND	6	GND	4

Fatto questo, collegate il programmatore USBtinyISP al computer e, da terminale, date il seguente comando:
avrdude -P /dev/ttyACM0 -U lfuse:w:0xe2:m -p t85 -c usbtiny

Se invece volete utilizzare l'Arduino UNO come programmatore ISP, potete utilizzare la versione di avrdude distribuita insieme all'IDE di Arduino, che è una versione vecchia ma modificata per poter riconoscere l'Arduino come programmatore ISP oppure, se avete Linux, la versione presente nella vostra distribuzione, basta che sia abbastanza recente (5.10/5.11), ricordandovi però di utilizzare il file avrdude.conf distribuito con l'IDE. Supposto di usare avrdude distribuito con l'IDE, aprite un terminale in /arduino-0022/hardware/tools e digitate il seguente comando:
./avrdude -P /dev/ttyACM0 -C ./avrdude.conf -U lfuse:w:0xe2:m -p t85 -c stk500v1 -b 19200

Analizziamo i parametri:

- "**-P /dev/ttyACM0**": specifica la porta a cui è connesso il programmatore. Nel caso di un sistema Linux, può essere /dev/ttyACM0 come nell'esempio ma la porta dipende da sistema a sistema e da scheda a scheda (potrebbe anche essere ad esempio /dev/ttyS0 oppure /dev/ttyUSB0).

- "**-U lfuse:w:0xe2:m**": indica su quale memoria del microcontrollore operare e che operazione deve essere eseguita. Nel nostro esempio "**lfuse**" indica il fuse basso ("low fuse"), "**w**" indica la scrittura ("write"), **0xe2** è il valore esadecimale da programmare, che disabilita il divisore x8 sul clock senza toccare le altre impostazioni di default del microcontrollore e "**m**" indica ad avrdude di usare direttamente il parametro ("m" per "immediate"). *N.B.: se volete sperimentare gli altri parametri dei fuse, potete usare l'utile Calcolatore online di fuse.*
- "**-p t85**": indica il tipo di microcontrollore, nel nostro esempio "**t85**" sta per ATtiny 85. Alternativamente possiamo utilizzare "**t84**" per un Attiny84 oppure "**t2123**" per un Attiny2313.
- "**-c stk500v1**": indica il programmatore da usare, in questo caso "stk500v1" si riferisce al programmatore Atmel STK500 versione 1 ma va bene anche per l'Arduino dato che lo sketch ArduinoISP emula via software proprio questo programmatore. "usbtiny" va bene invece per l'USBtinyISP.
- "**-C ./avrdude.conf**": indica il file di configurazione dei micro da usare per la programmazione.
- "**-b 19200**": indica la velocità di comunicazione in baud tra avrdude ed il programmatore. Se si usa l'USBtinyISP questo parametro non serve, mentre è indispensabile usare una velocità massima di 19200 baud nel caso si utilizzi l'Arduino perché lo sketch ArduinoISP ha preimpostata questa velocità, inoltre essendo lo sketch un emulatore non è capace di funzionare oltre una certa velocità.

Nota bene: usando un Arduino UNO R1 o R2, ricordatevi di disabilitare l'autoreset con il condensatore, da inserire prima di lanciare avrdude, oppure di utilizzare l'Optiboot 4.4 altrimenti l'Optiboot 4.0 originale resetterà la scheda impedendo la programmazione dell'Attiny.

Programmazione di un ATtiny85

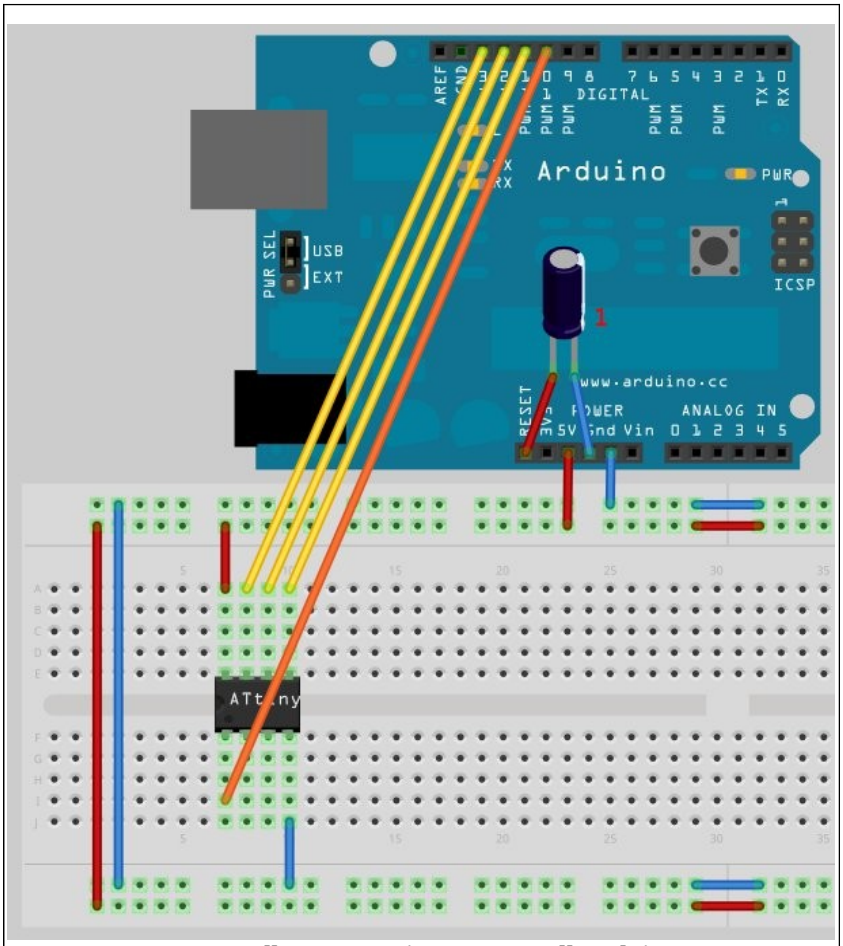
Adesso siamo pronti ad usare i micro per i nostri progetti. Grazie al core Tiny che andiamo ad utilizzare, gli ATtiny possono ora essere programmati usando una scheda Arduino tramite l'IDE stesso di Arduino. In questo modo abbiamo a disposizione un editor per scrivere il codice ed anche un'interfaccia per poter programmare in modo semplice il micro che stiamo usando.

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. La prima cosa da fare è caricare sull'Arduino lo sketch che lo trasforma in un programmatore ISP:

1. collegate l'Arduino al PC;
2. aprite l'IDE di Arduino;
3. aprite lo sketch "ArduinoISP" da "File/Examples";
4. selezionate la vostra scheda Arduino da "Tools/Board" (UNO o 2009);
5. selezionate la porta a cui è collegata l'Arduino da "Tools/Serial Port";
6. effettuate l'upload dello sketch.

Adesso è tempo di preparare l'ATtiny85 per la programmazione. Scollegate l'Arduino e collegate il micro esterno come in figura, con i seguenti collegamenti:

PIN ATTINY85	PIN ARDUINO
4	GND
8	+5V
1	10
5	11
6	12
7	13



Come collegare un micro esterno all'Arduino

In caso abbiate un Arduino UNO R1 o R2, ricordatevi che dovete disabilitare l'autoreset. Se avete scelto di installare l'Optiboot 4.4 siete a posto, altrimenti se avete scelto il condensatore non mettetelo adesso.

Ora chiudete l'IDE e collegate l'Arduino al PC.

Riaprite l'IDE e copiate lo sketch seguente, che servirà a far lampeggiare i LED:

```
#define RED 2
#define YELLOW 1
#define GREEN 0
#define BUTTON 4

byte value=1;
byte state=0;

byte val1=0;
byte val2=0;
byte change=0;
byte sequence=0;
unsigned long timer;

void setup() {
    pinMode(RED, OUTPUT);
    pinMode(YELLOW, OUTPUT);
    pinMode(GREEN, OUTPUT);
```

```
    pinMode(BUTTON, INPUT);
}

void loop() {
    //funzione di Debounce per il pulsantino - da qui....
    val1=digitalRead(BUTTON);
    delay(10);
    val2=digitalRead(BUTTON);
    if (val1==val2) {
        if (val1!=change) {
            if (val1==LOW) {
                write_leds(LOW, LOW, LOW);
                value++;
                if (value>5) { value=0; }
                sequence = 0;
                timer=millis()-100;
            }
        }
    }
    //...a qui
    //controlla se non sia ora di far lampeggiare un LED
    change=val1;
    if (millis() > timer) {
        timer = millis()+500;
        switch (value) {
            case 0:
                //tutti spenti
                write_leds(LOW, LOW, LOW);
                break;
            case 1:
                //alterna il LED rosso
                if (sequence == 0) {
                    write_leds(LOW, LOW, LOW);
                } else {
                    write_leds(HIGH, LOW, LOW);
                }
                sequence ^= 1;
                break;
            case 2:
                //alterna il LED giallo
                if (sequence == 0) {
                    write_leds(LOW, LOW, LOW);
                } else {
                    write_leds(LOW, HIGH, LOW);
                }
                sequence ^= 1;
                break;
            case 3:
```

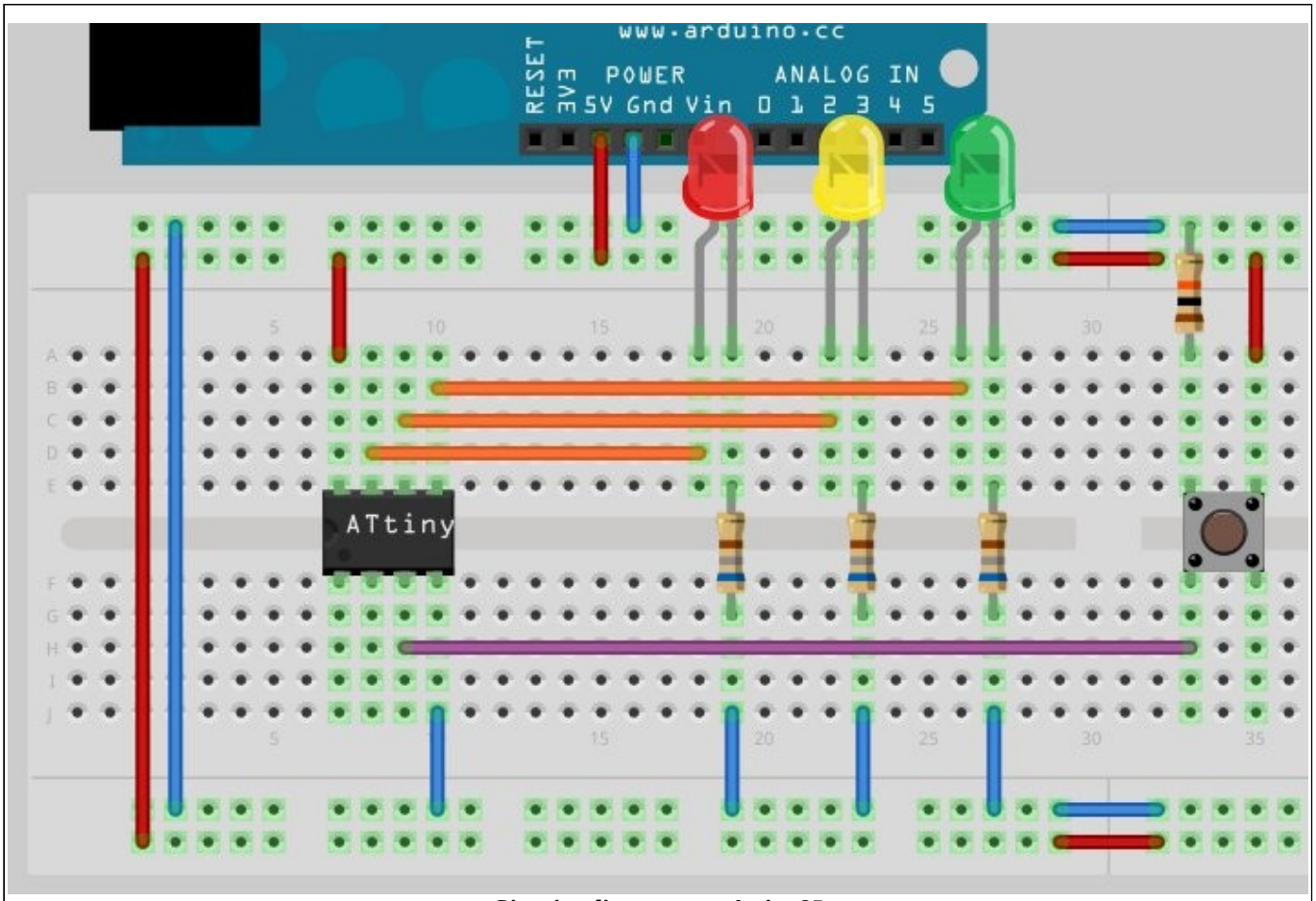
```
                //alterna il LED verde
                if (sequence == 0) {
                    write_leds(LOW, LOW, LOW);
                } else {
                    write_leds(LOW, LOW, HIGH);
                }
                sequence ^= 1;
                break;
            case 4:
                //alterna i 3 LED in sequenza
                switch (sequence) {
                    case 0:
                        write_leds(HIGH, LOW, LOW);
                        break;
                    case 1:
                        write_leds(LOW, HIGH, LOW);
                        break;
                    case 2:
                        write_leds(LOW, LOW, HIGH);
                        break;
                }
                sequence++;
                if (sequence > 2) { sequence = 0; }
                break;
            case 5:
                //tutti accesi
                write_leds(HIGH, HIGH, HIGH);
                break;
        }
    }
}

void write_leds(byte state1, byte state2, byte state3) {
    //modifica le porte dei 3 LED
    digitalWrite(RED, state1);
    digitalWrite(YELLOW, state2);
    digitalWrite(GREEN, state3);
}
```

Scritto lo sketch, è ora di inviarlo al microcontrollore. Prima di fare l'upload eseguite le seguenti operazioni:

1. selezionare “Attiny85 (w/Arduino as ISP)” da “Tools/Board”;
2. se avete optato per il condensatore anti-autoreset (solo per la UNO R1 e R2), collegatelo adesso:
    1. collegare il pin positivo (quello più lungo) sul pin RESET di Arduino;
    2. collegare il pin negativo (quello più corto) su uno dei 2 pin GND di Arduino.
3. Adesso inviate lo sketch: se i collegamenti sono corretti e l'IDE è stato opportunamente modificato ed il micro selezionato correttamente, dovreste vedere lampeggiare i led LED/RX/TX sull'Arduino e l'IDE che informa dell'upload in corso.

Nota Bene:
se ricevete un paio di segnalazioni di errore inerenti un certo “PAGEL” non fateci caso, l'upload andrà lo stesso a buon fine. Questo errore dipende da alcune definizioni mancanti in uno dei file di configurazione del compilatore.

Quando l'IDE segnalerà che l'upload è stato effettuato (ci vuole molto di più rispetto al normale upload sull'Arduino per via della ridotta velocità di trasmissione), scollegate l'Arduino dal PC. Adesso sfilate il condensatore e assemblate sulla breadboard il seguente schema:



Circuito di prova con Attiny85

Vi occorrono:

- 1 pulsantino da montaggio su PCB;
- 3 resistenze da 220/330/680 Ohm per i LED (uno di questi valori va bene, io ho usato quelle da 680);
- 1 resistenza da 10 Kohm;
- 3 LED, possibilmente di 3 colori differenti: 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;
- LER 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.

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 allo standard USART (seriale) in hardware, che solo l'Attiny2313, 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 **NewSoftSerial**, che fa proprio questo: simula via software la comunicazione seriale normalmente eseguita 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.

Procuriamoci la libreria, scaricandola da questo link:
http://arduiniana.org/NewSoftSerial/NewSoftSerial12.zip

Scompattiamo l'archivio nella cartella /sketchbook/libraries, in modo da averla separata dalle altre distribuite insieme all'IDE.

Adesso provvediamo alle modifiche necessarie a risolvere i suddetti problemi:

1. posizioniamoci nella cartella /sketchbook/libraries/NewSoftSerial;
2. apriamo il file **NewSoftSerial.cpp** con un editor di testo;
3. subito sotto alla riga ***DEFINITIONS*** inseriamo il seguente codice:

```
#if defined( __AVR_ATtiny24__ ) || defined( __AVR_ATtiny44__ ) ||
defined( __AVR_ATtiny84__ )
#define __AVR_ATtinyX4__
#endif
```

```
#if defined( __AVR_ATtiny25__ ) || defined( __AVR_ATtiny45__ ) ||
defined( __AVR_ATtiny85__ )
#define __AVR_ATtinyX5__
#endif
```

4. cerchiamo la funzione `void NewSoftSerial::enable_timer0(bool enable)`
5. modifichiamo il codice da così:

```
{
  if (enable)
#if defined(__AVR_ATmega8__)
  sbi(TIMSK, TOIE0);
#else
  sbi(TIMSK0, TOIE0);
#endif
 else
#if defined(__AVR_ATmega8__)
  cbi(TIMSK, TOIE0);
#else
  cbi(TIMSK0, TOIE0);
#endif
}
```

a così:

```
{
#if defined(__AVR_ATmega8__) || defined(__AVR_ATtinyX5__)
  sbi(TIMSK, TOIE0);
#else
  sbi(TIMSK0, TOIE0);
#endif
#if defined(__AVR_ATmega8__) || defined(__AVR_ATtinyX5__)
  cbi(TIMSK, TOIE0);
#else
  cbi(TIMSK0, TOIE0);
#endif
}
```

Questa modifica serve per rendere la libreria compatibile con gli Attiny85. Adesso dobbiamo modificare più in profondità la libreria per far sì che possa funzionare in ricezione anche con i microcontrollori Attiny25/45/85 e Attiny24/44/84. Salviamo il file. Ora dobbiamo aprire il file **icrmacros.h** e prima della riga

```
#elif defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
```

inseriamo il seguente codice:

```
#elif defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || (__AVR_ATtiny84__)
#define digitalPinToPCICR(p)    (((p) >= 0 && (p) <= 10) ? (&GIMSK) : ((uint8_t
*)NULL))
#define digitalPinToPCICRbit(p) (((p) <= 2) ? 5 : 4)
```

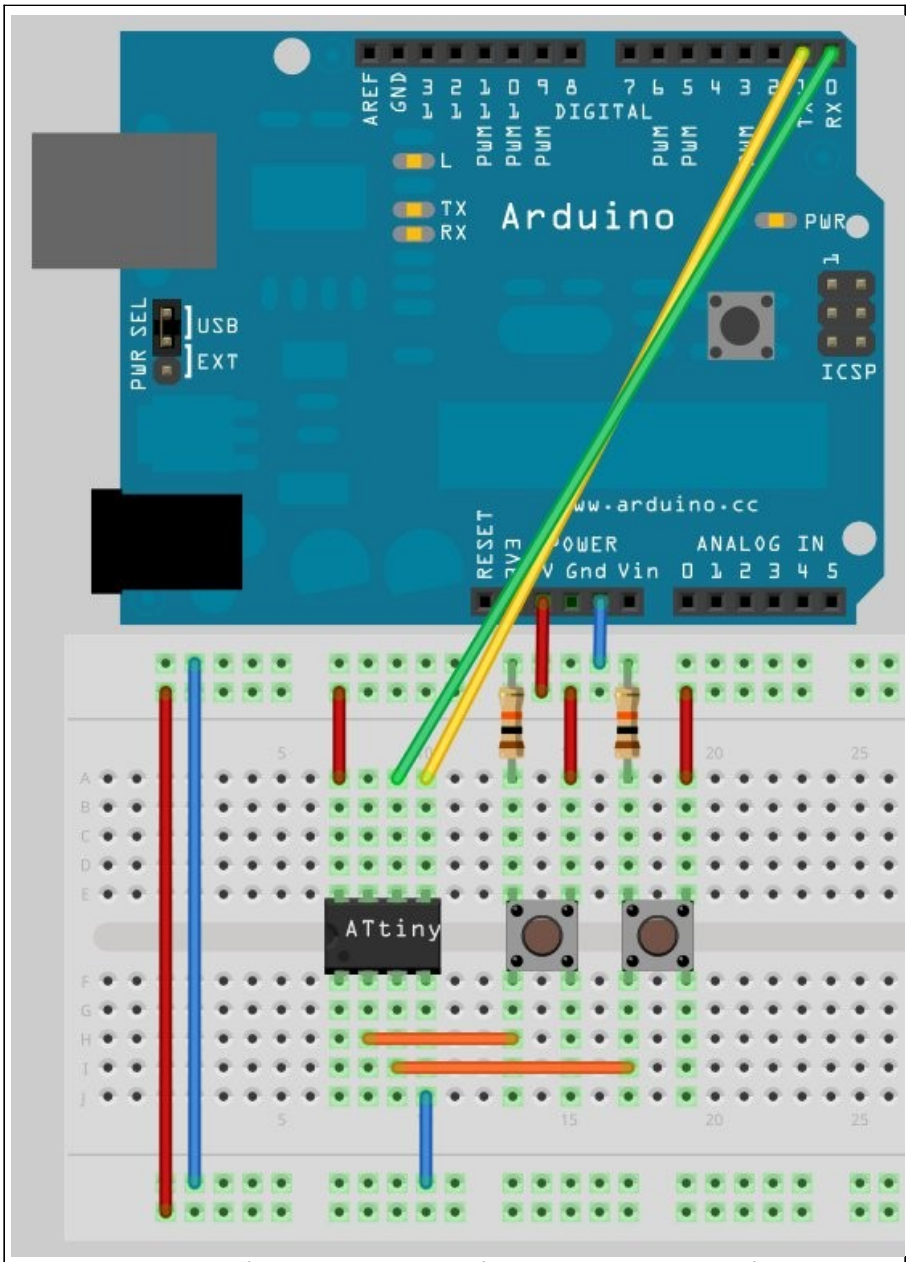
```
#define digitalPinToPCMSK(p)    (((p) <= 2) ? (&PCMSK1) : (((p) <= 10) ? (&PCMSK0) : ((uint8_t *)NULL)))
#define digitalPinToPCMSKbit(p) (((p) <= 2) ? (p) : (((p) - 10) * -1))
#elif defined(__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || (__AVR_ATtiny85__)
#define digitalPinToPCICR(p)    (((p) >= 0 && (p) <= 5) ? (&GIMSK) : ((uint8_t *)NULL))
#define digitalPinToPCICRbit(p) 5
#define digitalPinToPCMSK(p)    (((p) >= 0 && (p) <= 5) ? (&PCMSK) : ((uint8_t *)NULL))
#define digitalPinToPCMSKbit(p) (p)
```

Salviamo anche questo file. Adesso la nostra libreria è pronta all'uso. 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 NewSoftSerial 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.



Circuito di prova per testare la comunicazione seriale

Adesso avviamo l'IDE di Arduino e scriviamo il seguente sketch, che andremo a caricare sull'ATtiny85 usando il metodo descritto in precedenza:

```
#include <NewSoftSerial.h>

#define BUTTON1 3
#define BUTTON2 4
#define RX_PIN 0
#define TX_PIN 1

NewSoftSerial mySerial(RX_PIN, TX_PIN);

byte val11=0;
byte val12=0;
byte val21=0;
byte val22=0;
byte change1=0;
byte change2=0;

void setup() {
  pinMode(BUTTON1, INPUT);
  pinMode(BUTTON2, INPUT);
  mySerial.begin(9600);
}

void loop() {
  //funzione di Debounce per i pulsantini - da qui....
  val11=digitalRead(BUTTON1);
  delay(10);
  val12=digitalRead(BUTTON1);
  if (val11==val12) {
    if (val11!=change1) {
      if (val11==LOW) {
        mySerial.print("1");
      }
    }
  }
  change1=val11;
  val21=digitalRead(BUTTON2);
  delay(10);
  val22=digitalRead(BUTTON2);
  if (val21==val22) {
    if (val21!=change2) {
      if (val21==LOW) {
        mySerial.print("2");
      }
    }
  }
  change2=val21;
  //...a qui
}
```

Una volta caricato questo sketch, scolleghiamo l'Arduino dal PC, scolleghiamo la breadboard dall'Arduino e carichiamo sull'Arduino un semplicissimo sketch che non fa altro che mettersi in ascolto sulla seriale e stampare i caratteri che riceve su di essa:

```
void setup(){
  delay(3000);
  Serial.begin(9600);
}

void loop(){
  while (Serial.available()) {
    Serial.println(Serial.read(), BYTE);
  }
}
```

Ricolleghiamo l'Arduino, facciamo l'upload dello sketch, poi scolleghiamo nuovamente la scheda. Ricolleghiamo la breadboard all'Arduino seguendo lo schema visto in precedenza e poi colleghiamo 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 alla pressione dei 2 pulsantini.

Usare l'I2C

L'I2C (più precisamente I^2C) 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 tramite un indirizzo, una specie di ID che identifica 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.

Anche in questo caso ci viene in aiuto la comunità, che ha modificato la libreria Wire per renderla compatibile con i micro Attinyx5 producendo la libreria **TinyWire**. Però, come nel precedente caso della libreria NewSoftSerial, se intendiamo usare gli Attiny col clock maggiore, anche la TinyWire va modificata perché presuppone di lavorare con gli Attiny di serie, quindi con clock a 1 Mhz contro gli 8 Mhz dei nostri Attiny modificati.

Ecco come fare:
preleviamo la libreria TinyWire da questa pagina. Essa è disponibile sia come TinyWireM che come TinyWireS, da usarsi rispettivamente nel caso si voglia impostare l'ATtiny come “master” (M) o come “slave” (S). Preleviamole entrambe, tanto prima o poi potrebbero servire.

1. Fatto questo, scompattiamo i 2 archivi nella cartella /sketchbook/libraries, dove alla fine avremo 2 nuove cartelle denominate /TinyWireM e /TinyWireS.
2. Adesso provvediamo alla modifica della libreria TinyWireM. Entriamo in /sketcbook/libraries/TinyWireM ed apriamo il file **USI_TWI_Master.h**, sostituendo la seguente riga di codice:
   `#define SYS_CLK 1000.0`
   con
   ```
   #if (F_CPU==8000000UL)
   #define SYS_CLK   8000.0
   // For use with _delay_us()
   #define T2_TWI 40
   #define T4_TWI 32
   ```

```
#elif (F_CPU==1000000UL)
#define SYS_CLK 1000.0
// For use with _delay_us()
#define T2_TWI 5
#define T4_TWI 4
#endif
```

Con questa modifica la libreria compilerà sia per micro con clock a 1 che ad 8 Mhz. Per clock superiori basta adattare il codice assegnando a SYS_CLK la frequenza del micro in kHz ed a T2_TWI e T4_TWI i valori di (5*Clock_in_MHz) e (4*Clock_in_MHz) rispettivamente. N.B: *modifica NON testata: prove eseguite solo max ad 8 Mhz. Modificare anche il punto 5.*

3. Ora scorriamo il file finché non troviamo le definizioni dei pin per i microcontrollori Attinyx5. Subito sotto, prima delle definizioni per gli Attiny2313, inseriamo la seguente porzione di codice per rendere compatibile la libreria anche con la famiglia Attinyx4:

```
#if defined( __AVR_ATtiny24__ ) | defined( __AVR_ATtiny44__ ) |
defined( __AVR_ATtiny84__ )
#define DDR_USI             DDRA
#define PORT_USI            PORTA
#define PIN_USI             PINA
#define PORT_USI_SDA        PORTA6
#define PORT_USI_SCL        PORTA4
#define PIN_USI_SDA         PINA6
#define PIN_USI_SCL         PINA4
#endif
```

4. adesso aprire il file **USI_TWI_Master.cpp** e sostituiamo la seguente riga di codice:

```
#define F_CPU 1000000UL
```

con queste righe:

```
#include <avr/interrupt.h>
#if (F_CPU!=8000000UL) && (F_CPU!=1000000UL)
#error Clock frequency non supported
#endif
```

In questo modo la libreria userà la frequenza impostata nell'IDE, facendo un controllo per verificare che il clock sia a 1 oppure 8 Mhz: frequenze superiori non sono state provate, ed oltretutto bisogna usare un quarzo esterno sacrificando 2 pin del microcontrollore.

5. Ora è il turno della TinyWireS. Posizioniamoci in /sketchbook/libraries/TinyWireS ed apriamo il file **usiTwiSlave.c**. Nelle prime righe del file cerchiamo la sezione denominata “device dependent defines”, che contiene le definizioni dei pin in base ai microcontrollori usati. In questa sezione dobbiamo inserire, in un punto qualunque, la definizione per gli Attiny24/44/84, copiando il seguente codice:

```
#if defined( __AVR_ATtiny24__ ) | \
#defined( __AVR_ATtiny44__ ) | \
#defined( __AVR_ATtiny84__ )
#define DDR_USI         DDRA
#define PORT_USI        PORTA
#define PIN_USI         PINA
#define PORT_USI_SDA    PA6
#define PORT_USI_SCL    PA4
#define PIN_USI_SDA     PINA6
#define PIN_USI_SCL     PINA4
#define USI_START_COND_INT  USISIF
```

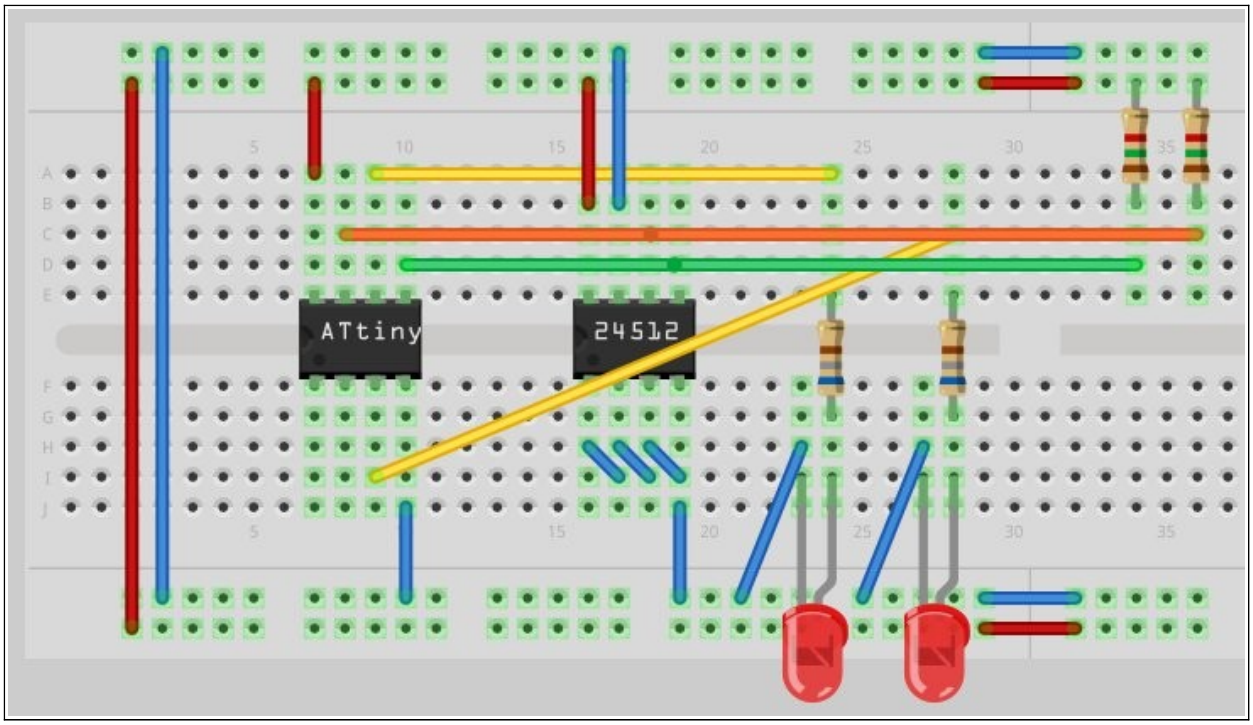
```
#define USI_START_VECTOR    USI_START_vect
#define USI_OVERFLOW_VECTOR USI_OVF_vect
#endif
```

Salviamo tutti i file: la libreria TinyWire (sia Master che Slave) è adesso pronta all'uso.

Nota bene:
di default, i buffer 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 ) // jjg was 16`
  - `#define TWI_TX_BUFFER_SIZE ( 32 ) // jjg was 16`

Per provare la comunicazione I2C useremo un chip di memoria EEPROM 24LC512 (va bene anche un taglio inferiore, come un 24LC256). Prepariamo un circuito come il seguente:



Per realizzarlo occorrono 2 LED, 2 resistenze da 220/330/680 Ohm per i LED (uno di questi valori va bene, io ho usato quelle da 680) 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

Adesso carichiamo sull'Attiny il seguente sketch:

```
#include <TinyWireM.h> // I2C Master lib for ATTinys which use USI

#define EEPROM_ADDR 0x50 // 7 bit I2C address for EEPROM 24LC512
#define LED1_PIN 4 // ATtiny Pin 3
#define LED2_PIN 1 // ATtiny Pin 6


void setup(){
  pinMode(LED1_PIN,OUTPUT);
  pinMode(LED2_PIN,OUTPUT);
  TinyWireM.begin(); // initialize I2C lib
  digitalWrite(LED1_PIN, HIGH); // show it's alive
  digitalWrite(LED2_PIN, HIGH); // show it's alive
  delay(500);
  clear_eeprom();
  digitalWrite(LED2_PIN, LOW);
  digitalWrite(LED1_PIN, LOW);
  delay(1000);
}

void clear_eeprom() {
  unsigned int i;
  byte a[10]={1, 0, 0, 1, 1, 1, 0, 1, 0, 1};

  for (i=0; i<10; i++) {
    writeData(EEPROM_ADDR, i, a[i]);
    delay(5);
  }
}

void loop(){
  unsigned int i;
  byte temp, light;

  digitalWrite(LED1_PIN, HIGH);
  for (i=0; i<10; i++) {
```

```
    temp = readData(EEPROM_ADDR, i);
    if (temp == 0) {
      digitalWrite(LED2_PIN, LOW);
    } else {
      digitalWrite(LED2_PIN, HIGH);
    }
    delay(500);
    digitalWrite(LED2_PIN, LOW);
    delay(100);
  }
  digitalWrite(LED1_PIN, LOW);
  delay(500);
}

void writeData(int device, unsigned int add, byte data) {
  byte temp;
  // writes a byte of data 'data' to the chip at I2C address 'device', in memory
location 'add'
  TinyWireM.beginTransmission(device);
  TinyWireM.send((byte)(add >> 8)); // left-part of pointer address
  TinyWireM.send((byte)(add & 0xFF)); // and the right
  TinyWireM.send(data);
  temp = TinyWireM.endTransmission();
  delay(5);
}

byte readData(int device, unsigned int add) {
  // reads a byte of data from memory location 'add' in chip at I2C address 'device'
  byte temp, result; // returned value
  TinyWireM.beginTransmission(device); // these three lines set the pointer position
in the EEPROM
  TinyWireM.send((byte)(add >> 8)); // left-part of pointer address
  TinyWireM.send((byte)(add & 0xFF)); // and the right
  TinyWireM.endTransmission();
  TinyWireM.requestFrom(device, 1); // now get the byte of data...
  result = TinyWireM.receive();
  return result; // and return it as a result of the function readData
}
```

Carichiamo lo sketch 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 riacceso 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.

Conclusioni

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.

File allegati

Tutti i file citati nella presente guida, vale a dire:

- il core Tiny*
- la libreria TinyWireM/S*
- la libreria NewSoftSerial*
- gli sketch di esempio

**: con tutte le modifiche citate nella guida già eseguite*

sono inseriti in un pacchetto allegato, ordinati per percorso. Per l'installazione, basta scompattare all'interno della propria cartella /sketchbook il contenuto del pacchetto e tutte le cartelle si posizioneranno nel loro correto percorso.

Licenze