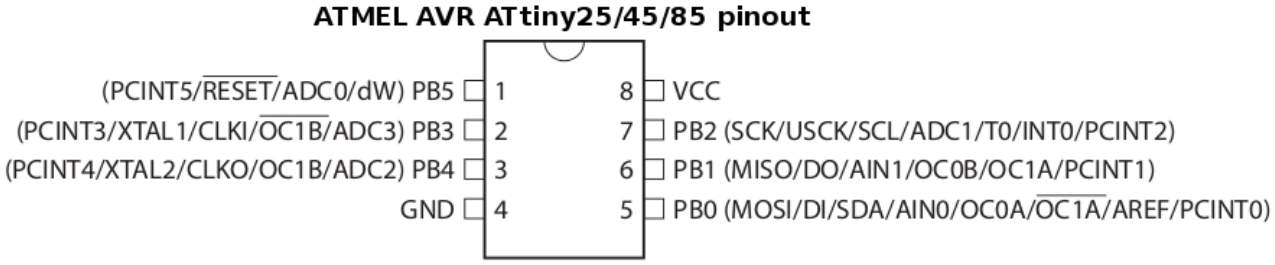
ATMEL AVR ATtiny25/45/85

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 un DIP8 molto pratico per l'uso hobbistico, 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 3: 25, 45 e 85, che differiscono l'uno dall'altro per la memoria integrata.

Piedinatura:



Caratteristiche tecniche principali:

- CPU: AVR 8 bit
- Clock: da 0 a 20 MHz, oscillatore interno regolabile a 1 o 8 MHz
- 3 modalità di risparmio energetico
- Watchdog
- Frequenza di funzionamento: da 2,7V a 5,5V
- Memoria:
  - Flash: 2/4/8 kB (per Attiny25/45/85)
  - SRAM: 128/256/512 byte (25/45/85)
  - EEPROM: 128/256/512 byte (25/45/85)
- Periferiche:
  - 2 timer ad 8 bit
  - 2 canali PWM (pin 5 e 6)
  - 4 ingressi impostabili in analogico (pin 1, 2, 3 e 7)
  - USI (Universal Serial Interface) programmabile per emulare protocolli SPI e I2C

Nota:
come detto, i 3 microcontrollori Attiny 25/45/85 sono perfettamente identici. Nel corso di questa guida utilizzeremo l'Attiny85 per la maggior quantità di memoria che offre. Nulla toglie di sostituirlo, a seconda dei propri progetti, con uno degli altri micro usando le stesse informazioni di questo documento, o al massimo apportando minime variazioni alle modifiche suggerite.

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 Arduino.

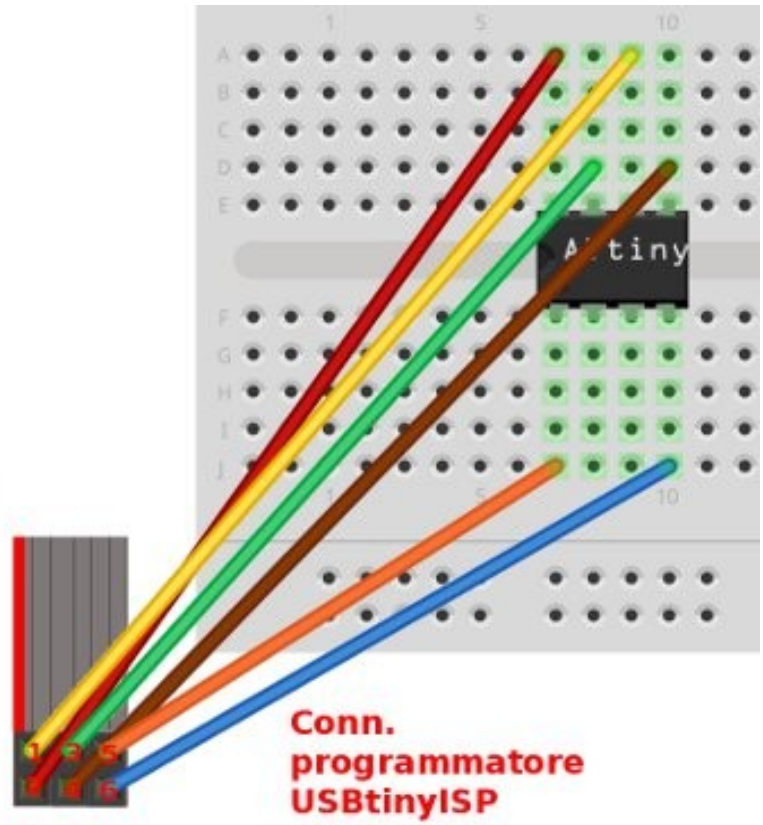
Per prima cosa bisogna collegarsi a questa pagina poi scaricare il file attiny45_85.zip che contiene i file necessari per far riconoscere all'IDE i microcontrollori Attiny25/45/85, e scompattarlo all'interno della cartella /hardware contenuta nella propria cartella degli sketchbook.
Se tale cartella non esiste, crearla. Alla fine si avrà un percorso come il seguente:
~/path_alla propria_cartella_sketchbook/hardware/attiny45_85

Adesso, aprendo l'IDE di Arduino, compariranno sotto il menu “Tools/Board” alcune nuove voci riguardanti l'Attiny45 e l'Attiny85. Se tali voci sono presenti, i file sono stati posizionati correttamente e l'IDE adesso supporta i nuovi microcontrollori.

Attiny a 1 oppure 8 Mhz

L'Attiny85 ha un oscillatore interno a 8 MHz ed un divisore x8 che può essere abilitato oppure no. Di fabbrica gli Attiny escono con tale divisore abilitato per cui la frequenza a cui lavora il micro 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. Per far ciò è necessario usare un programmatore ISP esterno: io ho utilizzato con successo l'USBtinyISP di Adafruit ma qualsiasi altro programmatore supportato da avrdure va più che bene.

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 come in figura:



Le connessioni da eseguire sono le seguenti:

PIN CONN. ISP	PIN ATTINY
1	6
2	8
3	7
4	5
5	1
6	4

Fatto questo, collegate il programmatore al computer e, da terminale, date il seguente comando:

avrdude -P /dev/ttyACM0 -U lfuse:w:0xe2:m -p t85 -c usbtiny

Che va interpreta nel seguente modo:

- parametro “-P”
  specifica l'indirizzo del programmatore. Nel mio caso ho messo /dev/ttyACM0 ma dipende da sistema a sistema, potrebbe anche essere ad esempio /dev/ttyS0.
- Parametro “-U”
  indica su quale memoria del microcontrollore operare e che operazione deve essere eseguita. Nel nostro esempio “lfuse” indica il fuse basso (“low fuse”), “w” sta per scrittura (“write”), 0xe2 è il valore esadecimale da programmare, che disabilita il divisore x8 senza toccare le altre impostazioni di default del microcontrollore e “m” indica ad avrdude di usare direttamente il parametro (“m” per “immediate”).
  Se volete giocare con gli altri parametri dei fuse, potete usare l'utile Calcolatore online di fuse.
- Parametro “-p”
  indica il tipo di microcontrollore, nel nostro esempio “t85” sta per ATtiny 85.
- Parametro “-c”
  indica il programmatore da usare, in questo caso “usbtiny” sta per USBtinyISP.

Fatto questo, avrdude dovrebbe comunicare l'avvenuta riprogrammazione dei fuse. Se tutto è andato a buon fine, è tempo di modificare i file che impostano la velocità del core del microcontrollore. Per far ciò entrate in /sketchbook/hardware/attiny45_85 (la cartella che avete creato in precedenza) ed aprite il file boards.txt. In questo file cambiate tutte le voci
attinyx5avrisp.build.f_cpu=1000000L

in
attinyx5avrisp.build.f_cpu=8000000L

dove **x5** sta per il tipo di microcontrollore in vostro possesso (45 oppure 85).

Fatto questo, gli sketch che compilerete saranno regolati per operare alla nuova frequenza del core, ossia 8 MHz.

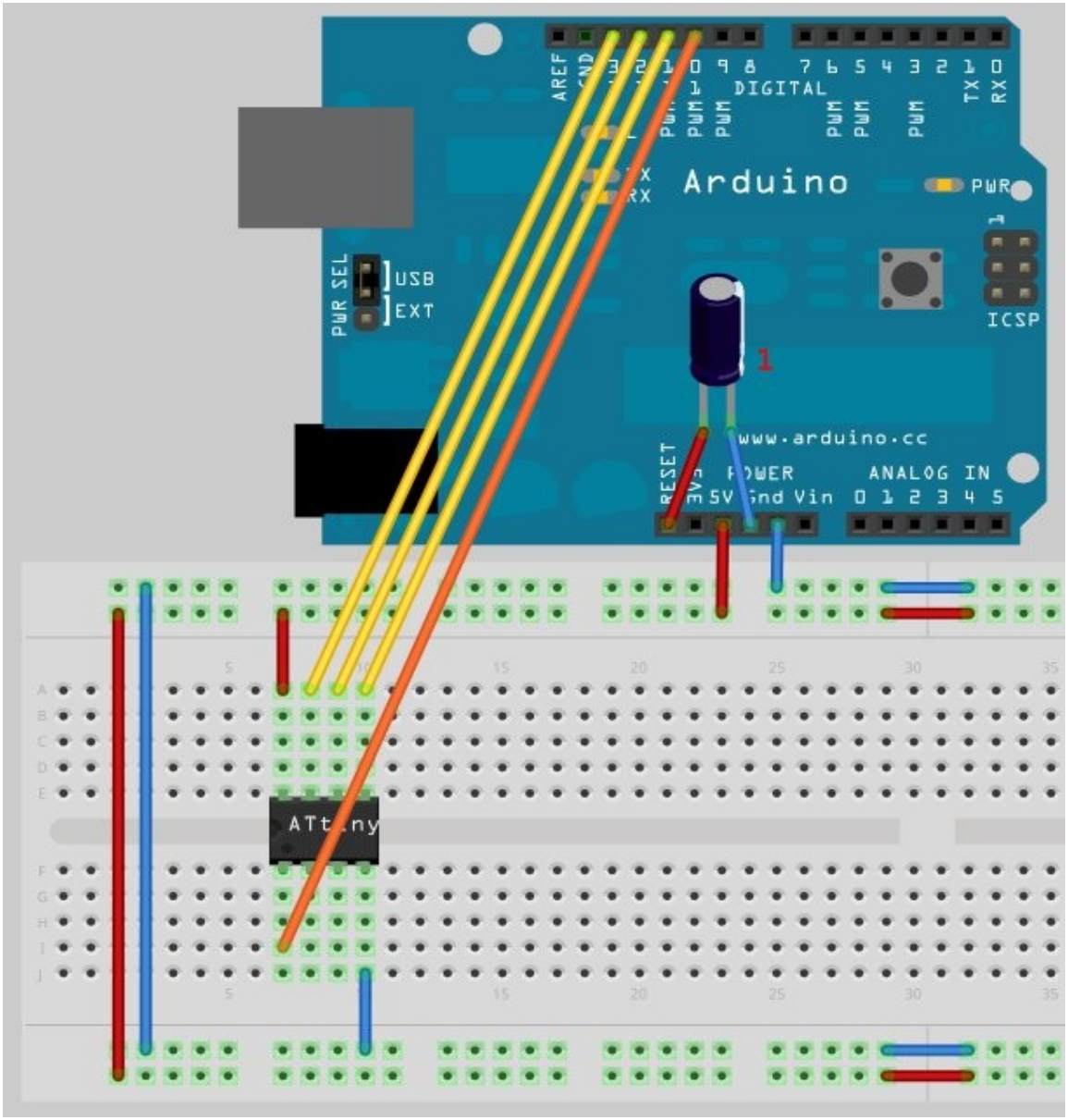
Programmazione dell'ATtiny85

Adesso siamo pronti ad usare il micro per i nostri progetti. L'ATtiny può essere ora programmato usando una scheda Arduino tramite l'IDE di Arduino stesso. In questo modo abbiamo a disposizione un editor per scrivere il codice ed anche un'interfaccia per poter programmare in modo semplice il micro.

Iniziamo con un piccolo progetto, un circuito con 3 LED il cui schema di lampeggio può essere cambiata premendo un pulsantino. 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 segue:



Collegate i pin come segue:

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

Se possedete un Arduino UNO, procuratevi un condensatore elettrolitico da 10 uF: vi servirà per disabilitare l'autoreset di questa scheda e poter programmare il micro esterno.
Se avete una 2009 dovrebbe occorrervi una resistenza da 120 Ohm al posto del condensatore.

Adesso collegate tutti i pin ma NON mettete il condensatore, poi 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);
}
```

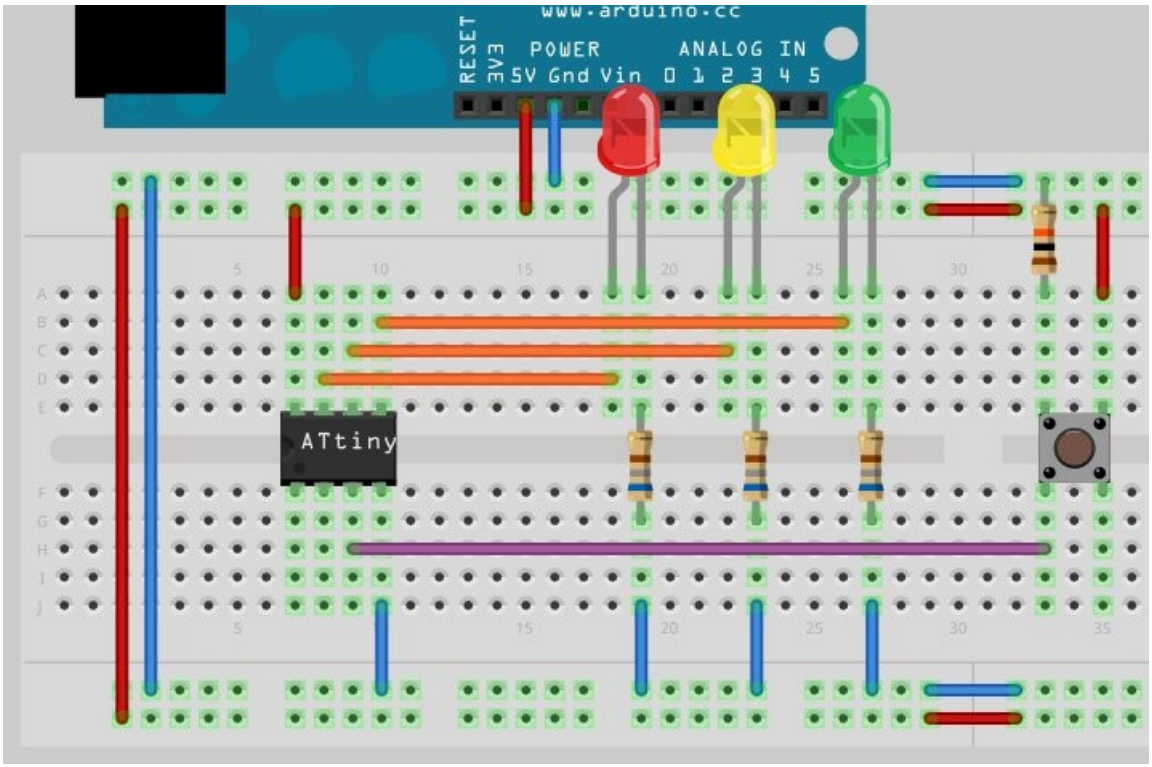
Scritto lo sketch, è ora di inviarlo al microcontrollore. Prima di fare l'upload eseguite le seguenti operazioni:

1. selezionare “Attiny85 (w/Arduino as ISP)” da “Tools/Board”
2. inserite il condensatore (solo per la UNO):
    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.

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.

NB:
se ricevete un paio di segnalazioni di errore inerenti un certo “PAGEL” non fateci caso. L'upload andrà lo stesso a buon fine.

Quando l'IDE segnalerà che l'upload è stato effettuato (ci vuole molto di più rispetto all'upload sull'Arduino), scollegate l'Arduino dal PC. Adesso sfilate il condensatore e costruite sulla breadboard il seguente schema:



Vi occorrono:

- 1 pulsantino da montaggio su PCB;
- 3 resistenze da 680 Ohm;
- 1 resistenza da 10 Kohm;
- 3 LED (possibilmente) di differenti colori.

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 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.

Questo compito è già più difficile perché il micro non implementa il supporto allo standard UART in hardware: 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 da micro quali l'ATmega328. La libreria va però modificata perché di serie presuppone di lavorare con micro che hanno più di 1 timer ma siccome l'Attiny85 na ha solo uno, il codice restituisce un errore.
Procediamo:

1. preleviamo la libreria da questa pagina (link diretto).
2. Andiamo nella nostra cartella /sketchbook e creiamo, se non è presente, una cartella denominata “libraries”. Dentro ad essa decomprimiamo la libreria.

Adesso dovremmo avere i file della libreria in /sketchbook/libraries/NewSoftSerial.

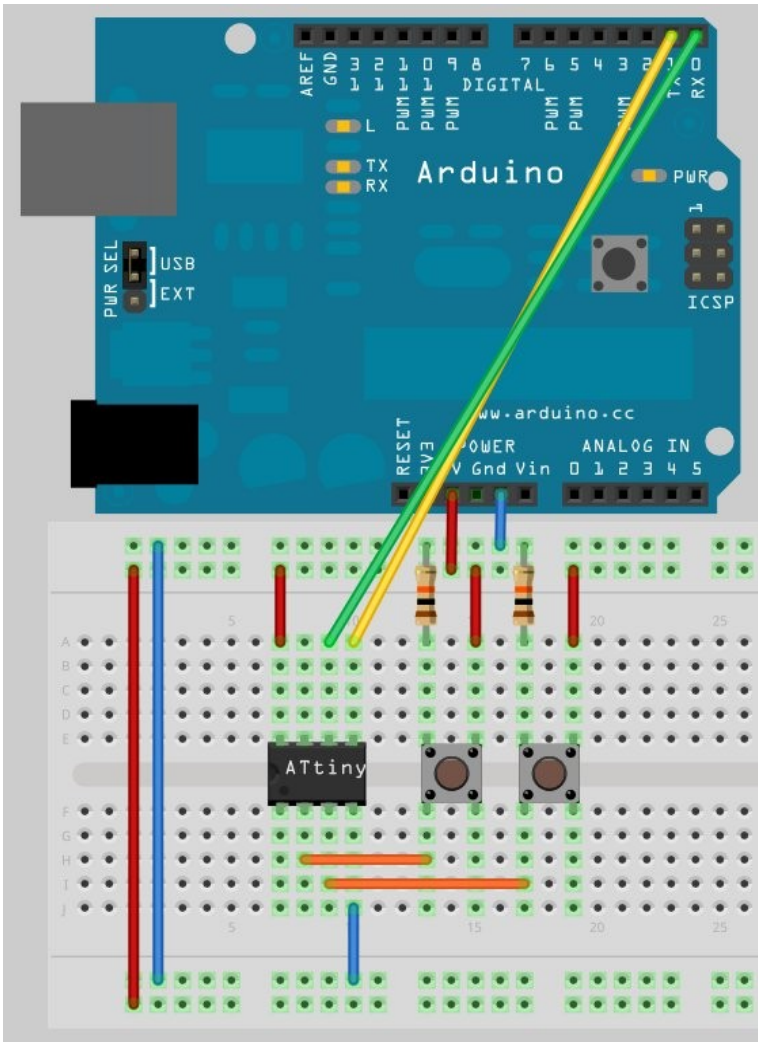
3. Aprire il file NewSoftSerial.cpp con un editor di testo.
4. Cercare la funzione `void NewSoftSerial::enable_timer0(bool enable)`
5. Modificare il codice da così:

```
{
  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 (enable)
    sbi(TIMSK, TOIE0);
  else
    cbi(TIMSK, TOIE0);
}
```

Salviamo il 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:

1. collegare i pulsantini ai pin 2 e 3 dell'ATtiny85, ognuno con la resistenza di pull-down a GND.
2. 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).
3. 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.
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(){
    if (Serial.available()>0) {
        Serial.println(Serial.read());
    }
}
```

Una volta eseguito l'upload, 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, 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:

1. 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.
2. Fatto questo, scompattiamo i 2 archivi nella cartella /sketchbook/libraries, dove alla fine avremo 2 nuove cartelle denominate /TinyWireM e /TinyWireS.
3. Adesso provvediamo alla modifica della libreria TinyWireM (la libreria TinyWireS non necessita di modifiche). Entriamo in /sketcbook/libraries/TinyWireM ed apriamo il file **USI_TWI_Master.h**.
4. Sostituire
   `#define SYS_CLK   1000.0`
   con
   `#define SYS_CLK   8000.0`

   Questa modifica va vedere il clock ad 8 Mhz.
5. Sostituire
   `#define T2_TWI    5`

```
#define T4_TWI    4
```

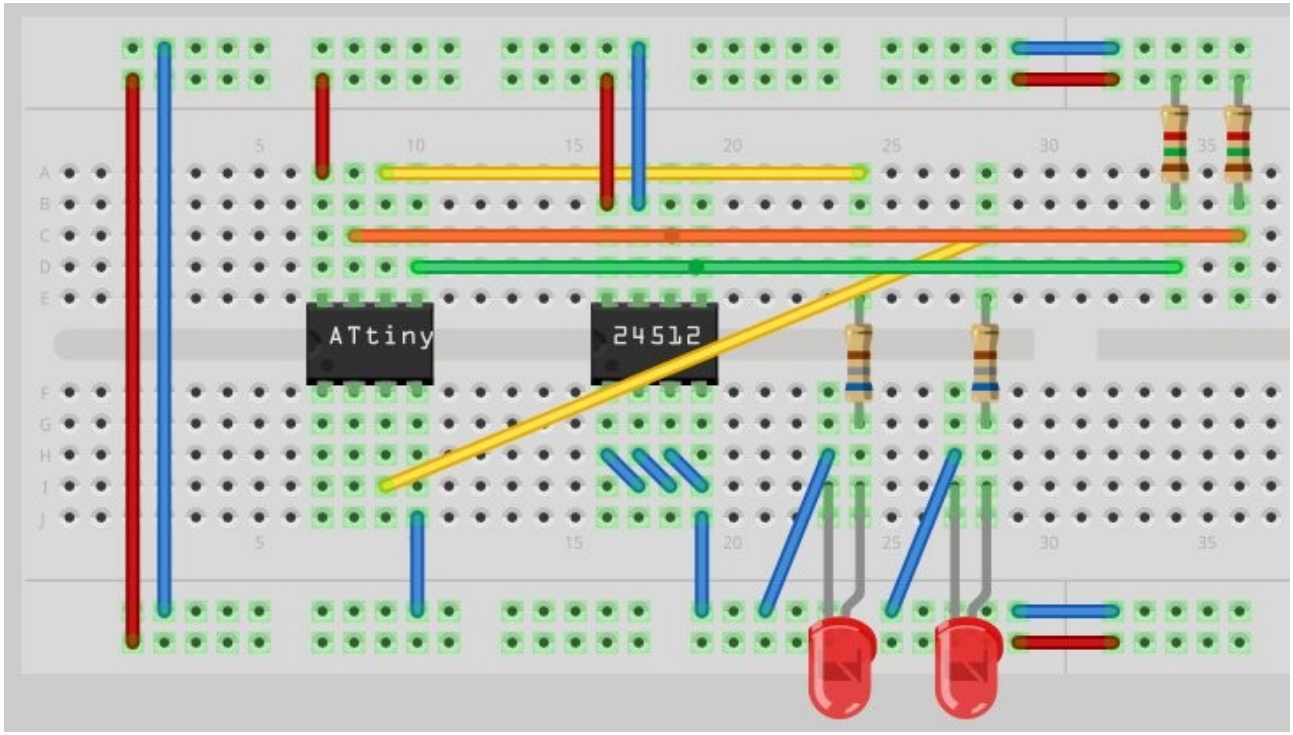
con
```
#define T2_TWI    40
#define T4_TWI    32
```
Questa modifica varia i timing in modo da adattarsi al clock maggiore.

6. Adesso aprire il file **USI_TWI_Master.cpp**.
7. Sostituire
```
#define F_CPU 1000000UL
```

con
```
#define F_CPU 8000000UL
```
sempre per impostare il clock maggiore.

Salviamo i file: la libreria è adesso pronta all'uso.

Per testarla 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 680 Ohm e 2 resistenze da 1,5 Kohm. 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(10);
    }
}


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(10);
}

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.

Leonardo Miliani
lenardo@leonardomiliani.com