Tutorial ATmega8 - #3:

USART

Conoscenze richieste: Programmazione in linguaggio C, Tuorial ATmega8 - #1,#2

Tempo richiesto: 1 - 2 ore circa

Linguaggio tecnico usato: Semplice ed amichevole

Autore: Emanuele Aimone

Contents

INTRODUZIONE

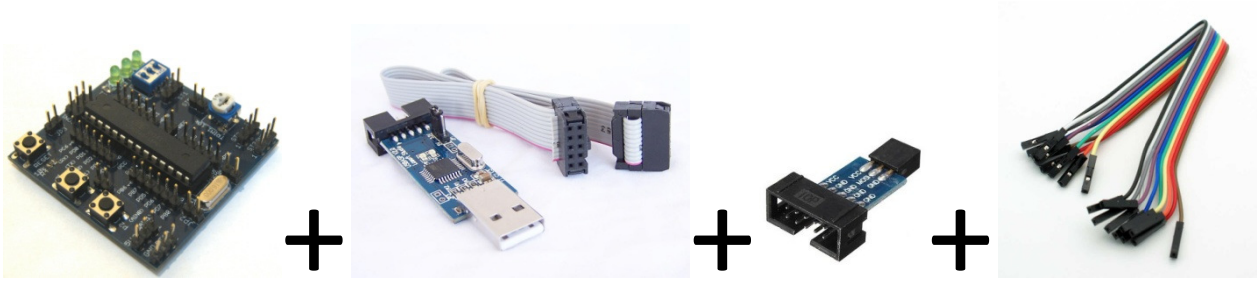
In questo tutorial viene configurata la USART per creare un echo dei dati ricevuti. Ogni dato ricevuto dall'ATmega8 sul pin RX viene ritrasmesso indietro attraverso il pin TX.

GLI ELEMENTI USATI IN QUESTO TUTORIAL

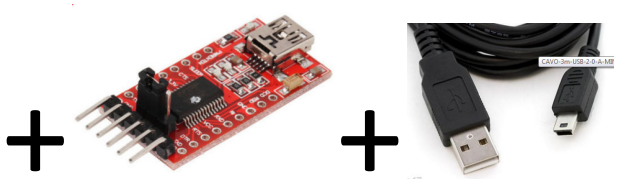
Per il tutorial abbiamo bisogno di una semplice demoboard, il programmatore USBasp, un convertitore USB a Seriale TTL (preferibili se basato su chip FTDI), il compilatore WinAVR, e il caricatore di firmawareExtreme Burner v1.4.3.

L'hardware

Ddemoboard ATmega8 + USBAsp+ AdattatoreUSBAsp + Jumpers



Convertitore USB a Seriale TTL basato su chip FTDI + Cavetto con connettore mini USB



I COLLEGAMENTI

La porta seriale sull'ATmega8 la troviamo sulla porta PD0 (RX pin2) e PD1 (TX pin3), quindi bisogna collegare su questi pin il nostro convertitore USB a Seriale TTL.

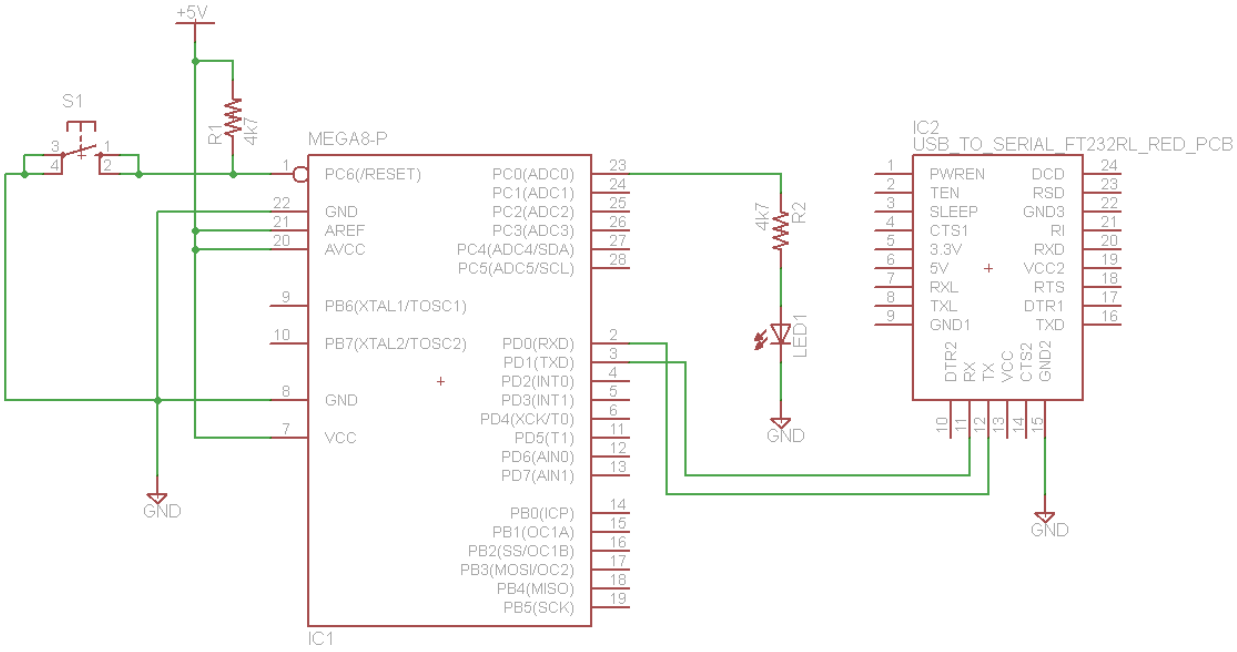
I rispettivi pin TX e RX del convertitore e della demoboard devono essere in collegati in modo incrociato:

RX (PD0 della debomoard) <--> TX (Convertitore USB a Seriale TTL)

TX (PD1 della debomoard) <--> RX (Convertitore USB a Seriale TTL)

A questo punto colleghiamo anche le masse insieme in modo che abbiano la stessa massa di riferimento.

Il led lo potete collegare sul pin PC0 (pin 23) della demoboard.



IL FIRMWARE

Il firmware farà lampeggiare brevemente il led collegato sul pin PC0 (pin23) della demoboard ogni volta che un dato viene ricevuto sulla porta seriale dell'ATmega8.

Tutti i dati ricevuti verranno ritrasmessi indietro.

Echo dei dati ricevuti via seriale

Vediamo subito l'esempio di programma che dobbiamo scrivere:

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define F_CPU 8000000UL  // 8 MHz

//Qui sotto basta cambiare USART_BAUDRATE con il baud rate voluto
#define USART_BAUDRATE 38400
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

#define TIME_LED_OFF            50//mS


//Variabili globali:
volatile unsigned char ultimo_byte_ricevuto;
volatile unsigned char cont_byte_ricevuti;
volatile unsigned char decontLed;
volatile unsigned int decontSend;


//Funzioni:
void USART_Init(void); //inizializza i registri per la USART
void timer0_init(void); //inizializza i registri del Timer0


//Funzione eseguita quando si scatena l'interrupt della USART.
ISR(USART_RXC_vect){
        ultimo_byte_ricevuto = UDR; //il dato ricevuto è contenuto dentro
                                    //il registro UDR
        cont_byte_ricevuti++;   //incremento il contatore

        PORTC &= 0xFE; // Led OFF
        decontLed = TIME_LED_OFF; //precarica la variabile in modo che il
                                    //led venga acceso dopo 50mS
}
```

```
//Funzione eseguita quando si scatena l'interrupt del Timer0.
//L'interrupt viene generato quando TCNT0 va in overflows.
ISR(TIMER0_OVF_vect){ // timer0 overflow interrupt
        TCNT0 += 6;//precarica il registro TCNT0 in modo di avere un
                //interrupt generato in 1mS

if(decontLed>1){
                decontLed--;//fino a quando non è a 1 (passati 50mS)
                        //decrementa la variabile
        }else{
                if(decontLed==1){
decontLed--;//adesso va a 0 e quindi non passerà di qui
//fino a quando decontLed non verrà ricaricato
                        PORTC |= 0x01; // Led ON
                }
        }

}

//il nostro codice principale, da dove il microcontrollore parte:
int main(void){

   //variabile vista solo localmente dalla funzione main()
   //usata per capire quando è stato ricevuto un'altro byte
   //via seriale
Unsigned char ultimo_cont_byte_ricevuti;

//configura PC0 (PIN23) come Output
//e tutti gli altri PCx vengono configurati come Input
DDRC = 0x01;

//Imposta il pin TX della seriale come Output
DDRD= 0x02;

cli(); //Disabilita gli interrupts globali

USART_Init(); //inizializza i registri per la USART
timer0_init(); //inizializza i registri del Timer0

sei(); //Abilita gli interrupts globali

//accendiamo il led che ci notificherà (spegnendosi per 50mS)
//se passano dati via seriale
PORTC |= 0x01; // Led ON

   //azzero il contatore di byte ricevuti
cont_byte_ricevuti = 0;

//inzializzo la variabile per capire poi se ricevo un byte
ultimo_cont_byte_ricevuti = cont_byte_ricevuti;
```

```
//invia un messaggio al PC
USART_SendByte('C');
USART_SendByte('i');
USART_SendByte('a');
USART_SendByte('o');
USART_SendByte('!');

decontSend=0;

   // Ciclo infinito
do{
                    //se il contatore locale e quello incrementato dalla funzione
                    //interrupt della USART sono diversi allora è ovvio che un byte
                    //è stato ricevuto
                    if(ultimo_cont_byte_ricevuti!=cont_byte_ricevuti){
                              //ora li rimetto uguali per non passare di qui all'infinito
                              ultimo_cont_byte_ricevuti = cont_byte_ricevuti;

                              //invio indietro al PC il byte ricevuto
                              USART_SendByte(ultimo_byte_ricevuto);
                    }
}while(1);
}


//inizializza i registri per la USART
Void USART_Init(void){
//Imposta il baud rate
  UBRRL = BAUD_PRESCALE; //carica i primi 8bit nel registro UBRR
  UBRRH = (BAUD_PRESCALE >> 8); //adesso carica i sucessivi 8bit nel registro UBRR

//per settare il numero di bit per trasmissione, la parità e gli stop bit
 //della seriale bisogna modificare
 //il registro UCSRC che però di default ha i valori che servono a noi:
 //8bit dati, no parità, 1 stop bit

// Abilita la ricezione, trasmissione e l'interrupt della USART
 UCSRB = ((1<<TXEN)|(1<<RXEN) | (1<<RXCIE));
}


//carica il byte nel relativo registro di invio e invia il dato
voidUSART_SendByte(unsigned char data){

//Se l'ultimo dato caricato non è ancora stato trasmesso allora
 //aspetta che venga inviato
while( !(UCSRA &(1<<UDRE)) );

//Carica e trasmette il dato
 UDR = data;
}
```

```
//inizializza i registri del Timer0
void timer0_init(void){
        //Vedere pagina 70 del datasheet per il diagramma a blocchi
        //del Timer0

        //Registro TIMSK: Pagina 72 del datasheet
        //Il bit chiamato TOIE0 viene impostato a 1 per abilitare
        //l'interrupt del Timer0
        TIMSK |= (1 << TOIE0);

//Registro TCCR0: Pagina 71 del datasheet
        //imposta il clock con prescaler a 64
        //Qui per essere sicuri che l'interrupt abbia cadenza 1mS
        //dobbiamo calcolare in base alla frequenza dell'oscillatore
        //la divisione che imponiamo con il prescaler
        TCCR0 |= (1 << CS01) | (1 << CS00);

        //Registro TCNT0: Pagina 72 del datasheet
        //precarica il contatore usato dal Timer0
        TCNT0 = 6;
}
```

Quando il microcontrollore si avvia inizializza i registri. Vediamo quindi i registri da inizializzare per la seriale:

```
void USART_Init(void){
//Imposta il baud rate
  UBRRL = BAUD_PRESCALE; //carica i primi 8bit nel registro UBRR
  UBRRH = (BAUD_PRESCALE >> 8); //adesso carica i successivi 8bit nel registro UBRR

//per settare il numero di bit per trasmissione, la parità e gli stop bit
 //della seriale bisogna modificare
 //il registro UCSRC che però di default ha i valori che ci servono:
 //8bit dati, no parità, 1 stop bit

// Abilita la ricezione, trasmissione e l'interrupt della USART
 UCSRB = ((1<<TXEN)|(1<<RXEN) | (1<<RXCIE));
}
```

Quindi il registro "UBRRL" insieme al registro "UBRRH" servono per impostare la velocità della porta seriale.

A pagina 132 del datasheet dell'ATmega8 troviamo le formule per il calcolo del baud rate:

Table 52. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

BAUD Baud rate (in bits per second, bps)

fOSC System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers (0 - 4095)

Quindi conoscendo il nostro clock (8MHz impostato da fuses) e sapendo che la seriale che vogliamo è di tipo semplice ed asincrona (U2X=0) possiamo usare la prima formula in alto a destra per sapere il valore da assegnare ai registri "UBRRL"+"UBRRH", i quali non solo altro che 2 registri a 8bits:

"UBRR Low"+"UBRR High"

questi due registri formano il registro "UBRR" da 16bits che vediamo nella formula.

Facendo un veloce calcolo per il baud rate voluto a 38400:

UBRR = (FOSC / 16 * BAUD) -1 = (16MHz / 16 * 38400) -1 = 25

Quindi impostiamo:

UBRRL = 25

UBRRH = 0

Comunque all'inizio del firmware abbiamo un define che ci semplifica l'impostazione del registro "UBRR":

```
//Qui sotto basta cambiare USART_BAUDRATE con il baud rate voluto
#define USART_BAUDRATE 38400
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
```

Proseguendo nella configurazione dei registri vediamo che dovremmo impostare il registro "UCSRC", ma questo ultimo per nostra fortuna di default ha gia impostati i valori più comunemente usati in una comunicazione seriale (vedi pagina 150 del datasheet dell'ATmega8):

- Asincrona

- Parità disabilitata

- Stop bit a 1

- 8bit di dati

Ultimo registro, il "UCSRB" serve per impostare l'interrupt ed abilitare la trasmissione e ricezione (vedi pagina 149 del datasheet dell'ATmega8):

- TXEN = 1 -> abilita trasmissione

- RXEN = 1 -> abilita ricezione

- RXCIE = 1 -> abilita interrupt

Dopo l'impostazione dei registri vediamo che prima di entrare nel ciclo infinito invia via seriale la parola "Ciao!"

```
USART_SendByte('C');
USART_SendByte('i');
USART_SendByte('a');
USART_SendByte('o');
USART_SendByte('!');
```

Come avrete capito la funzione "USART_SendByte(....);" invia un byte al computer, questo byte può essere un numero da 0 a 255 o una lettera come nel caso dell'invio del "Ciao!".

La lettera in sostanza viene tradotta in un numero da 0 a 255, se guardiamo una tabella ASCII possiamo vedere in quale numero viene tradotto:

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Se inviamo 'C' quindi invieremo il numero 67 (in esadecimale è 0x43).

Quindi l'istruzione:

```
USART_SendByte('C');
```

equivale a:

```
USART_SendByte(67);
```

Entrando ora nel ciclo infinito vediamo le seguenti istruzioni che vengono infinitamente eseguite:

```
if(ultimo_cont_byte_ricevuti!=cont_byte_ricevuti){
                        //ora li rimetto uguali per non passare di qui all'infinito
                        ultimo_cont_byte_ricevuti = cont_byte_ricevuti;

                        //invio indietro al PC il byte ricevuto
                        USART_SendByte(ultimo_byte_ricevuto);
                }
```

Questa routine si occupa di controllare il contatore locale "ultimo_cont_byte_ricevuti" con quello incrementato nella routine eseguita quando si scatena l'interrupt di ricezione dato dalla seriale.

Quindi, visto che i due contatori sono diversi, allora spedisce indietro il byte ricevuto.

Quando si scatena l'interrupt di ricezione esegue la seguente funzione:

```
ISR(USART_RXC_vect){
        ultimo_byte_ricevuto = UDR; //il dato ricevuto è contenuto dentro
                                    //il registro UDR
        cont_byte_ricevuti++;   //incremento il contatore

        PORTC &= 0xFE; // Led OFF
        decontLed = TIME_LED_OFF; //precarica la variabile in modo che il
                                    //led venga acceso dopo 50mS
}
```

questa funzione non fa altro che prendere il dato appena ricevuto dal registro "UDR" e lo salva nella variabile globale "ultimo_byte_ricevuto" e quindi incrementa la variabile "cont_byte_ricevuti" e spegne il led il quale verrà riacceso in modo autonomo dal Timer0 (visto nel tutorial #2) dopo il tempo "TIME_LED_OFF".

Quindi come visto il programma è molto semplice. Ora non resta che creare delle funzioni che, a fronte di un dato seriale ricevuto, esegua delle istruzioni.

Esempio: si potrebbe provare a collegare su PC1 (pin 24 dell'ATmega8) un altro led e modificando le istruzioni dentro il ciclo infinito si potrebbe farlo accendere e spegnere.

Quindi per prima cosa ricordiamoci di settare come output tale porta PC1:

```
DDRC = 0x03; //Porta PC0 (pin 23) e porta PC1 (pin 24) impostati come output
```

Poi modifichiamo le istruzioni dentro il ciclo infinito così:

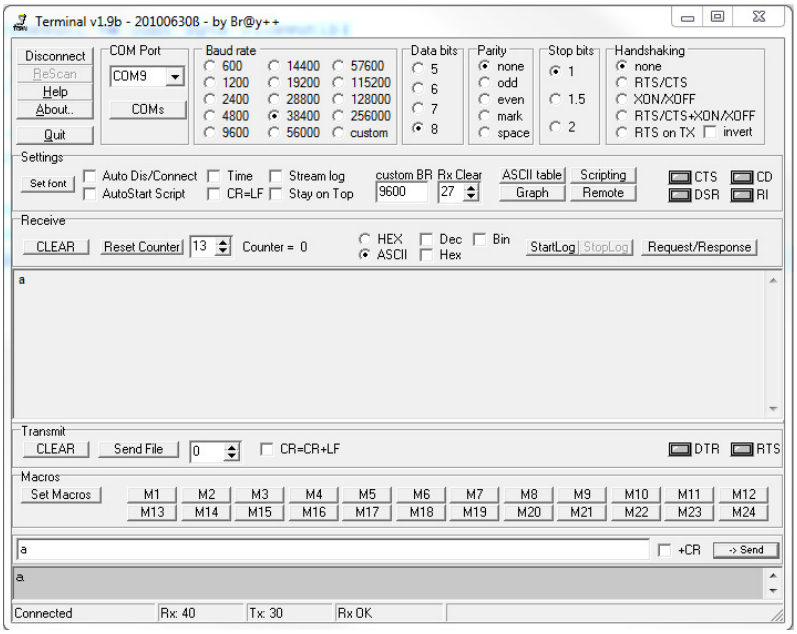
```
                if(ultimo_cont_byte_ricevuti!=cont_byte_ricevuti){
                        //ora li rimetto uguali per non passare di qui all'infinito
                        ultimo_cont_byte_ricevuti = cont_byte_ricevuti;

                        //invio indietro al PC il byte ricevuto
                        USART_SendByte(ultimo_byte_ricevuto);


                        if(ultimo_byte_ricevuto== 'a' ||ultimo_byte_ricevuto== 'A'){
                                if((PORTC & 0x02) == 0){
                                        PORTC |= 0x02;    // Led ON
                                }else{
                                        PORTC &= 0xFD;    // Led OFF
                                }
                        }
                }
```

quindi quando si schiaccia da tastiera il tasto 'a' o 'A' viene cambiato di stato il led attaccato a PC1 (pin24).

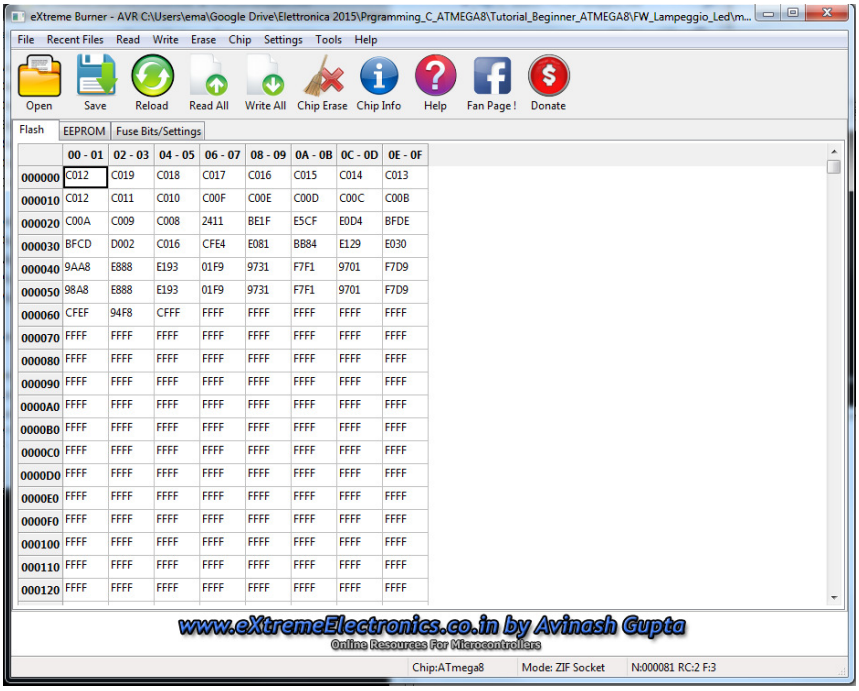
Ora per testare avviamo un Serial Terminal come questo:



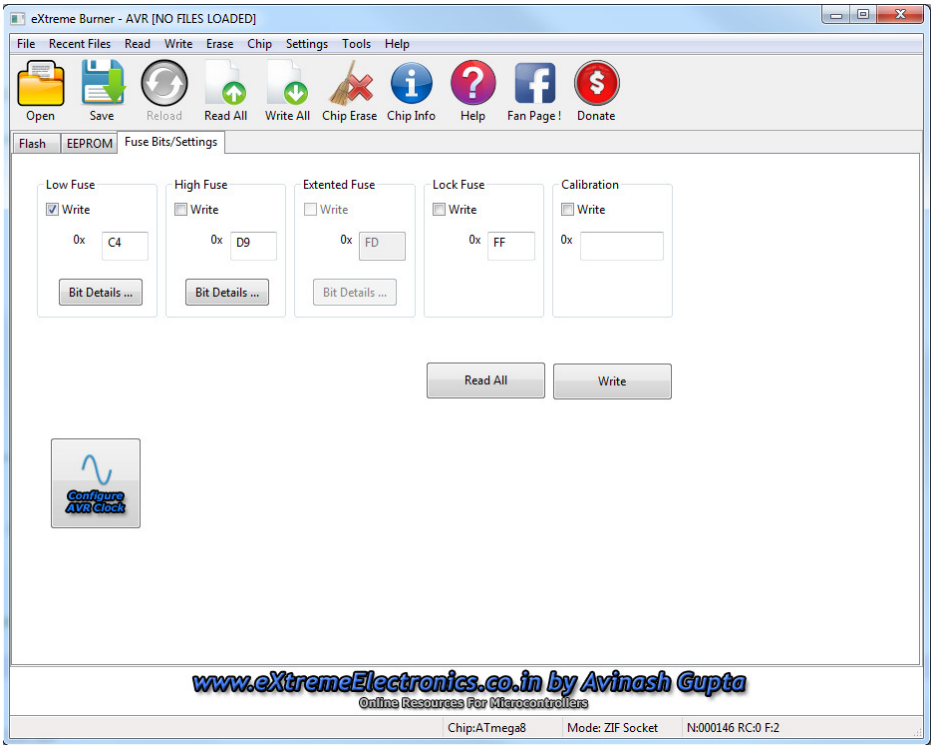
configurarlo come da immagine (COM Port *quella_vostra*, Baud rate 38400, Data bits 8, Parity none, Stop bits 1, Handshaking none) e quindi inviare la lettera 'a'.

Caricare il firmware

Apriamo Extreme Burner v1.4.3 e clicchiamo sul tasto Open per andare a caricare il file main.hex:



Ora è molto importante impostare i Fuse, andiamo sul tab chiamato "Fuse Bits/Settings" e impostiamo i Fuse esattamente come l'immagine che segue:



Quindi:

Low Fuse = 0xC4

High Fuse = 0xD9

Lock Fuse = 0xFF

Calibration = 0xB9B9B9B9

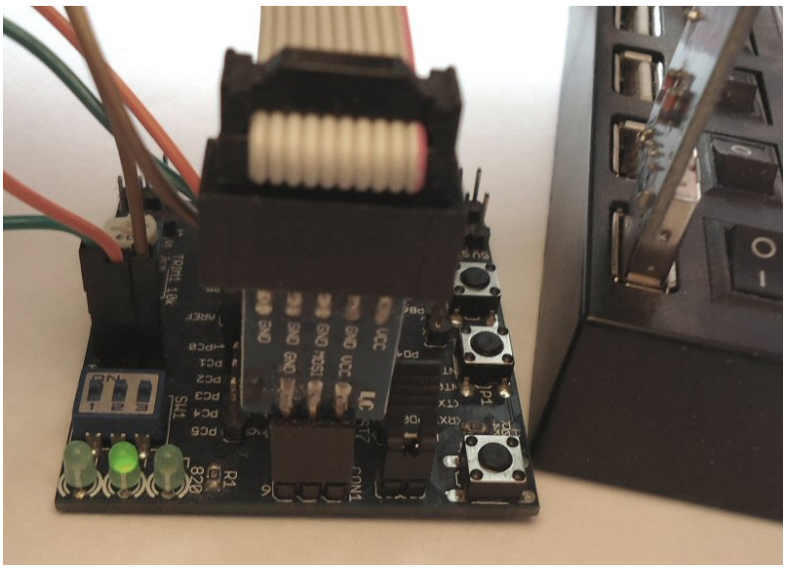
Mettete anche la spunta su write su di almeno il "Low Fuse"!

Per capire come impostare il clock possiamo vedere il datasheet dell'ATmega8 a pagina 26 oppure ci affidiamo al fuse calculator:

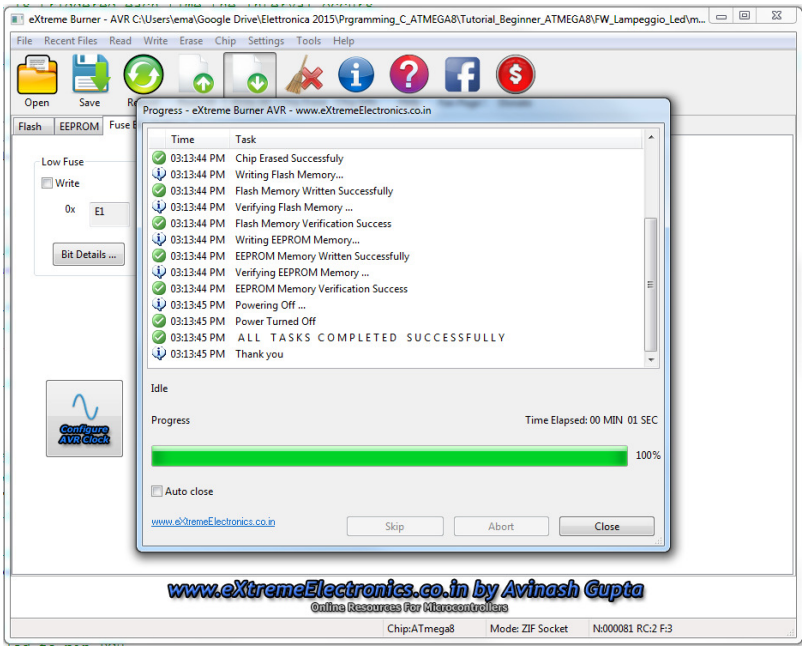
http://www.engbedded.com/fusecalc/

Ricordate di non cambiare i fuse se non capite cosa state facendo, potreste non poter più riprogrammare il vostro ATmega8.

Ora possiamo finalmente caricare il programma, assicuriamoci dei collegamenti come da foto:



Clicchiamo su Write All, se il caricamento è andato a buon fine allora avremo un schermata tipo questa:


eXtreme Burner - AVR C:\Users\ema\Google Drive\Elettronica 2015\Prgramming_C_ATMEGA8\Tutorial_Beginner_ATMEGA8\FW_Lampeggio_Led\m...
File Recent Files Read Write Erase Chip Settings Tools Help
Open Save
Flash EEPROM Fuse
Low Fuse
Write
0x E1
Bit Details ...
Configure AVR Clock
Progress - eXtreme Burner AVR - www.eXtremeElectronics.co.in
Time Task
03:13:44 PM Chip Erased Successfuly
03:13:44 PM Writing Flash Memory...
03:13:44 PM Flash Memory Written Successfully
03:13:44 PM Verifying Flash Memory ...
03:13:44 PM Flash Memory Verification Success
03:13:44 PM Writing EEPROM Memory...
03:13:44 PM EEPROM Memory Written Successfully
03:13:44 PM Verifying EEPROM Memory ...
03:13:44 PM EEPROM Memory Verification Success
03:13:45 PM Powering Off ...
03:13:45 PM Power Turned Off
03:13:45 PM ALL TASKS COMPLETED SUCCESSFULLY
03:13:45 PM Thank you
Idle
Progress
Time Elapsed: 00 MIN 01 SEC
100%
Auto close
www.eXtremeElectronics.co.in
Skip
Abort
Close
www.eXtremeElectronics.co.in by Avinash Gupta
Online Resources For Microcontrollers
Chip:ATmega8
Mode: ZIF Socket
N:000081 RC:2 F:3