Steve Blackbird (TardoFreak)

IL PIC18F47J53 È LA MIA NUOVA MASCOTTE!

12 March 2013

Non ne so il motivo ma sono sempre stato attratto dai micro-controllori "piccoli e cattivi", infatti il **PIC18F14K50** mi piace tantissimo e lo trovo molto divertente proprio per questo. Purtroppo l' oggettino, oltre ad avere il suo punto di forza (a parere mio eh!) nel USB che ha a bordo ha anche i suoi limiti: pochi pin e non tantissima FLASH. E' vero che esiste il fratello maggiore, il ben conosciuto **PIC18F4550** che, anche se offre un numero di pin maggiore (40 o 44) e nella versione DIL è ottimo per le sperimentazioni casalinghe, anche lui non ha tantissima FLASH: 32KB.
Ma c'è un micro che offre di più allo stesso prezzo: il **PIC18F47J53** che di memoria FLASH (128KB) e di RAM (4KB) ne ha abbastanza per farci cose molto sfiziose.

Orbene, capita che ne avevo uno nel cassettino, lo avevo comprato un paio di anni fa ma non l'avevo mai utilizzato perché purtroppo soffro di una sindrome che definirei "acquisto compulsivo di micro-controllori" e quando vedo un bell' oggettino non riesco a non comprarlo (nessuno è perfetto). Per fortuna è una malattia che non mi porta alla rovina, visti i bassi prezzi dei micro di oggi. Dicevo, ce l' avevo nel cassettino e, per provare un adattatore per TQFP l' ho montato.
Un giorno, per rilassarmi un po' e staccarmi dagli ARM, ho deciso di provarlo, di cucirci intorno un circuitino e farlo funzionare. In questo modo non mi sarei allontanato troppo dal lavoro di programmazione, avrei aggiunto un briciolo di conoscenza e mi sarei rilassato un po', anche se mio figlio sostiene che ho un modo singolare per rilassarmi. Vabbè, era di sera, alla tele davano solo cretinate (nessun bel documentario di astronomia o di fisica, film belli neanche cercandoli con il lanternino) e quindi ho fatto l' hobbista.
In questo breve articolo descrivo passo per passo l'approccio che ho utilizzato per la realizzazione di questo prototipino che forse un giorno utilizzerò per farci un lavoro.

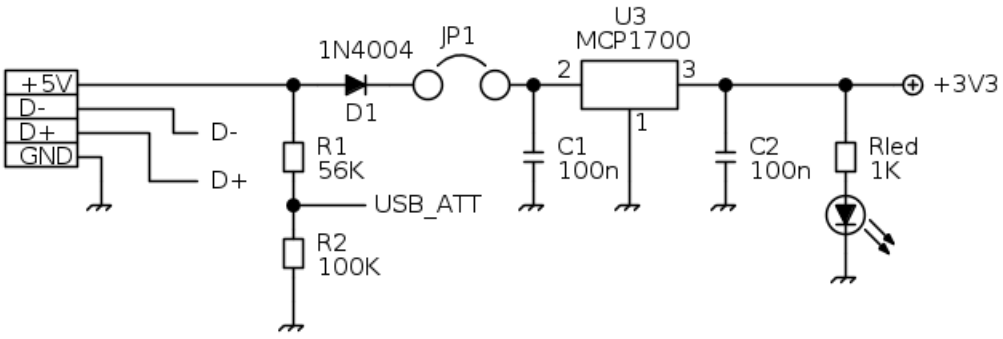
La realizzazione

L'idea era quella di avere un sistemino con il quale giocare con il micro e, chiaramente, l' USB. Sono andato a recuperare la documentazione della **PIC18F47J53 FS USB PIM Demo Board** per ricavarci un circuito. Non mi sono voluto inventare nulla e quindi ho usato il circuito paro paro, quindi un quarzo da 12MHz e tutto il resto.
Partire da un qualcosa già fatto, è un buon sistema proprio perché elimina una variabile che è quella dell' hardware. Ovviamente l'hardware deve avere un senso, si

possono omettere le parti che non interessano ma tenere ad esempio, gli I/O come quelli dell'hardware di riferimento permette di provare i programmi dimostrativi senza dovere apportare modifiche al firmware. Per modificare c'è sempre tempo, prima bisogna però partire da un qualcosa che funziona.

Alimentazione e USB

L'alimentazione, per comodità, l'ho ottenuta dal connettore dell' USB. Il problema è che dall' USB ci prendo un 5V mentre il micro lavora a 3,3V. Quindi, sempre tenendo sott' occhio il circuito della scheda che ho indicato prima, ho realizzato la sezione di alimentazione secondo questo circuito.

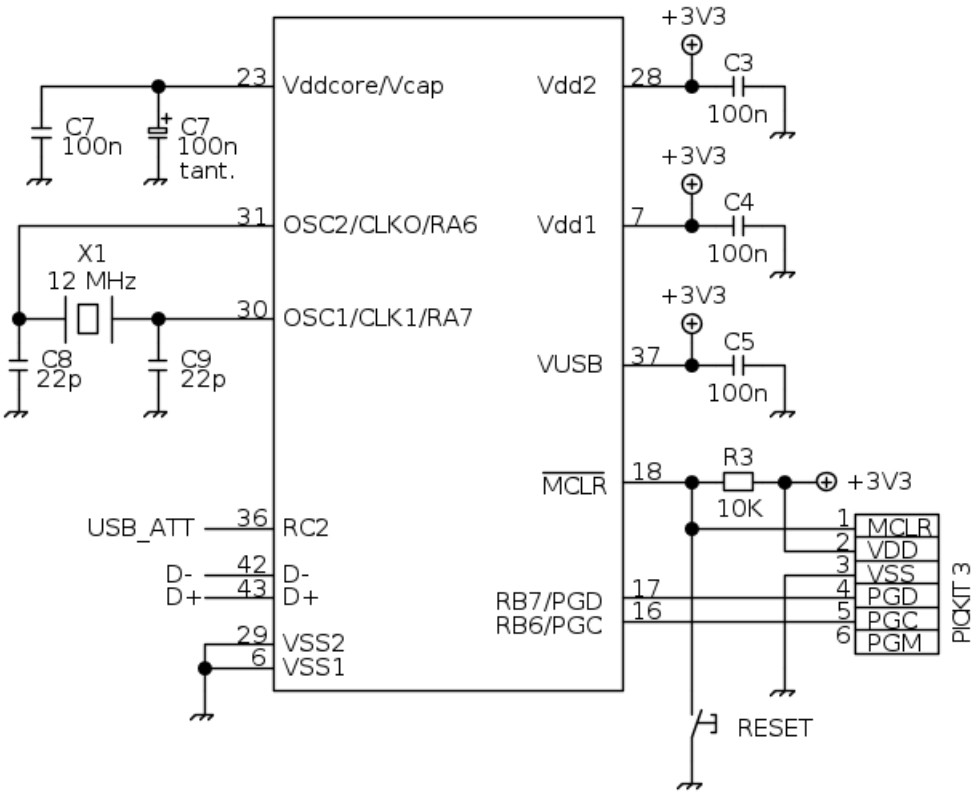


Come si può notare prendo l' alimentazione dal +5V del connettore USB e, attraverso il diodo D1 ed il ponticello JP1 la mando al regolatore **MCP1700** che mi genera il 3,3V per l' alimentazione di tutta la baracca. JP1 serve come interruttore nel caso dovessi un domani usare un' alimentazione che arriva da altrove e D1 l' ho messo per avere la possibilità, tramite una altro diodo, di tenere il circuito alimentato o con il 5V della USB o da un' altra alimentazione. Sia D1 che JP1 non sono necessari, il fatto è che so come vanno le cose con i prototipi e quindi li ho messi.
R1 ed R2 sono parte dello schema della scheda e servono per indicare al micro che l' USB è connessa. Serve in alcune applicazioni, io ho fatto che metterle.
Non poteva mancare il LED con la sua resistenza in serie per farmi capire se il circuito è alimentato o meno.

La circuiteria intorno al micro

Per la circuiteria necessaria per far funzionare il micro c'è poco da dire, è sufficiente guardare il **datasheet** del micro, lo schema della scheda della Microchip e, ancora una volta, non inventarsi niente. La scheda della Microchip monta un quarzo da 12MHz, io ho preferito montarlo su uno zoccolo in modo da poterlo sostituire, magari con un altro di frequenza diversa (ho una gran quantità di quarzi ad 8MHz) ma ho preferito andare sul sicuro. Stessa cosa per quanto riguardano i condensatori di

bypass dell' alimentazione.
Questo è lo schema per le alimentazioni, i quarzo e il debug con il PicKit3



Una piccola chiosa su questo punto. Nel datasheet, alla sezione "Guidelines for getting started with PIC18FJ microcontrollers", sarebbe ovvio immaginare di trovare gli componenti che sono montati sulla scheda di sviluppo della Microchip, giusto? No, sbagliato! Ci sono differenze.
Ad esempio la rete sul pin MCLR è diversa da quella indicata dallo schema della scheda. Il datasheet indica la presenza di un condensatore, di una resistenza da 10K verso l' alimentazione e di una resistenza da 100 a 470 ohm. Nel circuito della scheda invece la resistenza su MCLR è di 330 ohm verso la Vcc. Quale usare? Beh, io non mi sono lambiccato il cervello più di tanto e ci ho messo la classica resistenza da 10K verso la Vcc punto e basta. Questo perché devo collegare la baracca al PicKit 3 e, nel manuale dell' emulatore, viene indicata questa resistenza. Il perché della differenza fra le due configurazioni (quella del datasheet e quella della scheda) non so spiegarla. Misteri della Microchip!

Per la programmazione/debug

Non ostante sia fornito di emulatore ICD3 ho preferito usare il PicKit3 solo per un motivo di convenienza. Con il PicKit 3 mi è bastato collegare una strip di pin angolati

per avere un comodo collegamento con l' emulatore. L' ICD3 ha un connettore particolare che non può essere montato agevolmente su una millefori. In fondo si tratta di un esperimento ed il PicKit3 è decisamente abbastanza per fare tutti gli esperimenti. E' inutile cercarsi rogne, ed io non sono andato a cercarmele.

Il firmware di prova

Per provare il circuito ho utilizzato il progetto **USB Device - CDC - Basic Demo - C18 - PIC18F47J53 PIM.mcp** che si trova nella Microchip application library. In poche parole il progetto simula una seriale viurtuale implementando la classe CDC della USB. Si carica il programma così com' è, si apre un programma di emulazione di terminale come **Hyper Terminal** e si può vedere il funzionamento del demo. E' una cretinata nel senso che invia un eco del carattere che si invia dalla tastiera, ma è sufficiente per verificare il buon funzionamento del circuito e della USB.
Partendo da questo demo si possono creare progetti basati su USB.
Ho anche provato il **HID bootolader** che si trova nella libreria. E' un programmino interessante. Il funzionamento l' ho scoperto da solo. Se si avvia il micro con l' applicazione programmata il programma esegue l' applicazione, se invece si alimenta il circuito e gli si fornisce un "reset a caldo" (un reset mentre il circuito è alimentato) il programma esegue il bootloader tramite il quale si può caricare l' applicativo in FLASH. Non è un sistema molto intuitivo ma funziona, l' ho verificato. Per fornire il reset a caldo ho semplicemente montato un pulsantino fra il pin 18 (MCLR) e la Vss.

Il mio modello firmware

Arrivato a questo punto ho dovuto ricavare un modello da cui partire per sviluppare programmi per questo micro. Lasciando perdere per un attimo l' USB, ho ricavato questo sorgente che è il classico "hello world" dei micro (la generazione di un qualcosa che posso vedere usando un oscilloscopio) partendo dal main del demo della classe CDC e poi tagliando via tutte le parti che non m' interessavano per lasciare solo alcune sezioni importanti. Dopo questo lavoro di forbici ed aspirapolvere questo è il sorgente:

```
#include "p18f47j53.h"

#pragma config WDTEN = OFF          //WDT disabled (enabled by SWDTEN bit)
#pragma config PLLDIV = 3           //Divide by 3 (12 MHz oscillator input)
#pragma config STVREN = ON            //stack overflow/underflow reset enabled
#pragma config XINST = OFF          //Extended instruction set disabled
#pragma config CPUDIV = OSC1        //No CPU system clock divide
#pragma config CP0 = OFF            //Program memory is not code-protected
#pragma config OSC = HSPLL          //HS oscillator, PLL enabled, HSPLL used by USB
//#pragma config T1DIG = ON           //Sec Osc clock source may be selected
```

```
//#pragma config LPT1OSC = OFF        //high power Timer1 mode
#pragma config FCMEN = OFF          //Fail-Safe Clock Monitor disabled
#pragma config IESO = OFF           //Two-Speed Start-up disabled
#pragma config WDTPS = 32768        //1:32768
#pragma config DSWDTOSC = INTOSCREF //DSWDT uses INTOSC/INTRC as clock
#pragma config RTCOSC = T1OSCREF    //RTCC uses T1OSC/T1CKI as clock
#pragma config DSBOREN = OFF        //Zero-Power BOR disabled in Deep Sleep
#pragma config DSWDTEN = OFF        //Disabled
#pragma config DSWDTPS = 8192       //1:8,192 (8.5 seconds)
#pragma config IOL1WAY = OFF        //IOLOCK bit can be set and cleared
#pragma config MSSP7B_EN = MSK7     //7 Bit address masking
#pragma config WPFP = PAGE_1        //Write Protect Program Flash Page 0
#pragma config WPEND = PAGE_0       //Start protection at page 0
#pragma config WPCFG = OFF          //Write/Erase last page protect Disabled
#pragma config WPDIS = OFF          //WPFP[5:0], WPEND, and WPCFG bits ignored

#define USE_HID_BOOTLOADER

#define CLOCK_FREQ 48000000
#define GetSystemClock() CLOCK_FREQ

//------------------------------------------------------------------------------
// Prototipi locali
void YourHighPriorityISRCode();
void YourLowPriorityISRCode();


#if defined(USE_HID_BOOTLOADER)
        #define REMAPPED_RESET_VECTOR_ADDRESS                   0x1000
        #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS        0x1008
        #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS        0x1018
#else
        #define REMAPPED_RESET_VECTOR_ADDRESS                   0x00
        #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS        0x08
        #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS        0x18
#endif

#if defined(USE_HID_BOOTLOADER)
extern void _startup (void);        // See c018i.c in your C18 compiler dir
#pragma code REMAPPED_RESET_VECTOR = REMAPPED_RESET_VECTOR_ADDRESS
void _reset (void)
{
    _asm goto _startup _endasm
```

```
}
#endif
#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR = REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void)
{
     _asm goto YourHighPriorityISRCode _endasm
}
#pragma code REMAPPED_LOW_INTERRUPT_VECTOR = REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void)
{
     _asm goto YourLowPriorityISRCode _endasm
}

#if defined(USE_HID_BOOTLOADER)
//Note: If this project is built while one of the bootloaders has
//been defined, but then the output hex file is not programmed with
//the bootloader, addresses 0x08 and 0x18 would end up programmed with 0xFFFF.
//As a result, if an actual interrupt was enabled and occured, the PC would jump
//to 0x08 (or 0x18) and would begin executing "0xFFFF" (unprogrammed space).  This
//executes as nop instructions, but the PC would eventually reach the REMAPPED_RESET_VE
//(0x1000 or 0x800, depending upon bootloader), and would execute the "goto _startup".
//would effective reset the application.

//To fix this situation, we should always deliberately place a
//"goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS" at address 0x08, and a
//"goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS" at address 0x18.  When the output
//hex file of this project is programmed with the bootloader, these sections do not
//get bootloaded (as they overlap the bootloader space).  If the output hex file is not
//programmed using the bootloader, then the below goto instructions do get programmed,
//and the hex file still works like normal.  The below section is only required to fix
//scenario.
#pragma code HIGH_INTERRUPT_VECTOR = 0x08
void High_ISR (void)
{
     _asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS _endasm
}
#pragma code LOW_INTERRUPT_VECTOR = 0x18
void Low_ISR (void)
{
     _asm goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS _endasm
}
#endif        //end of "#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PROG
```

```
//-----------------------------------------------------------------------------
// Funzioni di servizio delle interrupt
//-----------------------------------------------------------------------------
#pragma code
//These are your actual interrupt handling routines.
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode()
{
        //Check which interrupt flag caused the interrupt.
        //Service the interrupt
        //Clear the interrupt flag
        //Etc.


}        //This return will be a "retfie fast", since this is in a #pragma interrupt se
#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode()
{
        //Check which interrupt flag caused the interrupt.
        //Service the interrupt
        //Clear the interrupt flag
        //Etc.

}        //This return will be a "retfie", since this is in a #pragma interruptlow sect

//-----------------------------------------------------------------------------
// Variabili globali
//-----------------------------------------------------------------------------
#pragma udata

//-----------------------------------------------------------------------------
// Funzioni
//-----------------------------------------------------------------------------
#pragma code


//-----------------------------------------------------------------------------
// MAIN
void main(void)
{
  unsigned char c;

  PORTD = 0;
```

```
  LATD = 0;
  TRISD = 0;
  c = 0;
  for(;;)
  {
    LATD = c;
    c++;
  }
}
```

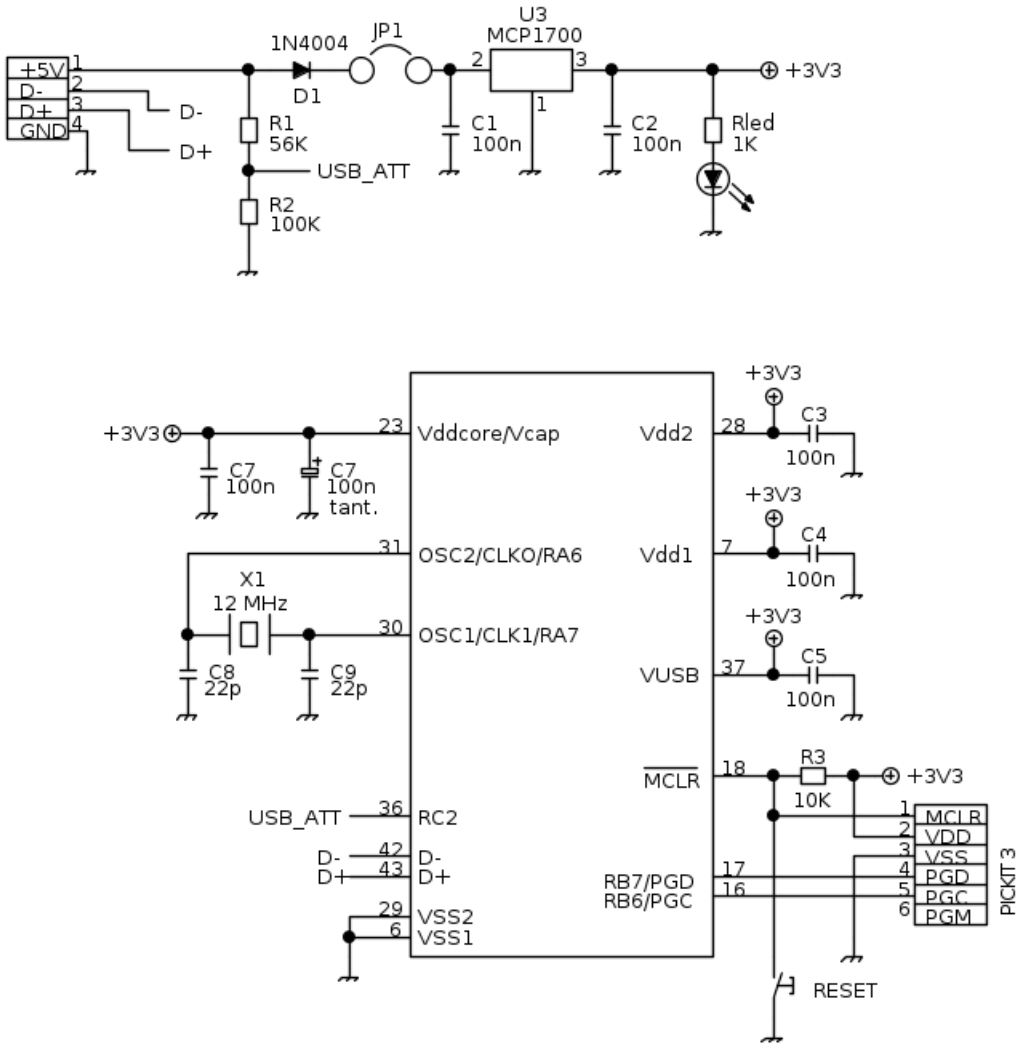
E' evidente che l' ho ottenuto dal programma di demo della USB ma è importante notare alcune cose:

- all' inizio si trova la configurazione dei fuses
- subito dopo i fuses definisco il simbolo chiamato **USE_HID_BOOTLOADER** che se definito mi mappa i vettori di RESET e di INTERRUPT in modo tale da poter utilizzare il bootloader che ho provato. Se metto sotto commento questa #define il programma verrà compilato per l' uso senza bootloader.
- I simboli **CLOCK_FREQ** e **GetSystemClock()** sono utili per gli eventuali programmi perché danno la possibilità al programmatore di avere un riferimento alla frequenza di funzionamento (può sempre servire)
- di seguito ci sono già le funzioni vuote dentro le quali scrivere il codice di servizio delle interrupt. Sono già scritte e quindi basta metterci l' ambaradan dentro e tutto funzionerà a meraviglia. Una vera comodità ih ih ih.
- La sezione "variabili globali" ci ricorda che con il C18 è necessario utilizzare la direttiva **#pragma udata** quando si dichiarano le variabili globali (non è ANSI ma ce ne faremo una ragione)
- Anche la sezione "Funzioni" ci ricorda che dobbiamo selezionare la FLASH utilizzando la direttiva **#pragma code** per informare il compilatore che quello che viene dopo è codice da inserire nella FLASH e che è un codice da eseguire. Anche questo non è ANSI ed anche in questo caso, oibò, dobbiamo farcene una ragione.
- Per ultimo troviamo un main che è un monumento alla stupidità ed alla banalità. E' ovvio che se andrò a guardare con l' osccilloscopio il pin RD0 ci vedrò un' onda quadra di una certa frequenza e, magia delle magie, su RD1 ci troverò un' onda quadra con frequenza esattamente metà di quella che vedo su RD0 e via discorrendo.

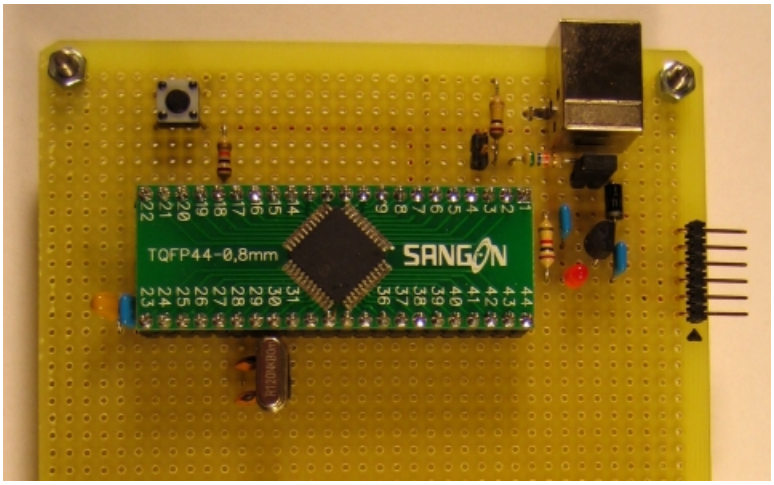
Ovviamente, al posto del programma da dementi che ho messo nel main, sarebbe bene metterci un programma che abbia un senso, che faccia un qualcosa ma questo è solo il modello su cui lavorare. L' idea, o meglio, la speranza è quella di farci girare magari un utile programma o comunque un qualcosa e, tempo permettendo, cercherò di farlo.

Il circuito completo

Questo è lo schema del circuito completo del protoipo che ho realizzato e le foto di come l' ho montato sulla millefori.

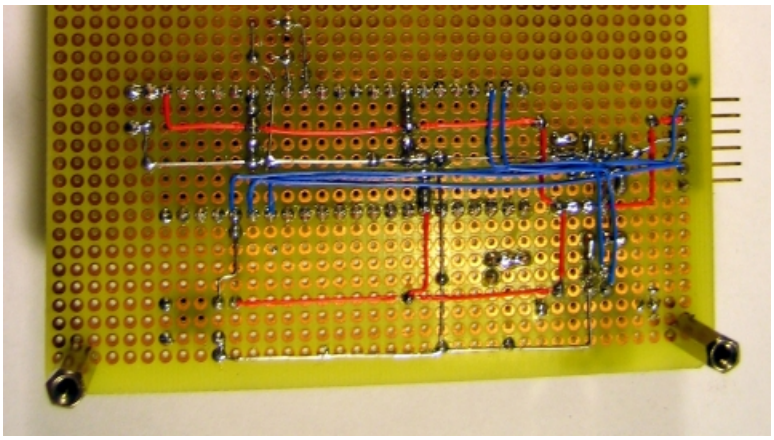


Ho usato un connettore USB tipo B perché facilmente inseribili in una millefori.



P47J53_LC.jpg

Sulla destra si trova la strip con i pin per il collegamento con il PicKit 3, in altro a destra il connettore USB, in alto a sinistra il pulsantino del reset ed in basso il quarzo montato su zoccolo.



P87J53_LS.jpg

Il circuito è filato con fili da wire-wrap saldati. Come si può notare è un circuito semplice. Ho usato il filo rosso per la VDD (+3,3V), filo rigido (a dire il vero sono spezzoni di reofori che recupero dai componenti) ed il filo blu per i collegamenti USB e per il PicKit 3.
Nelle foto ho solo riportato la parte montata, in effetti ho utilizzato una millefori formato eurocard (100z160 mm.) per avere spazio per montarci qualcosa di altro, sempre se ne avrò il tempo. Spero quindi un giorno di riempirla con diversi componenti su cui giocare un po' e magari scrivere un altro articolo.

Conclusioni

Penso che il senso dell' articolo sia abbastanza chiaro: **non serve cercarsi guai visto che sono già loro che cercano noi!**. E' un principio di buon senso che invito ad applicare anche quando si inizia una sperimentazione da zero. In effetti, come ripetuto più volte, non mi sono inventato niente, ho solo preso quel poco che mi serviva partendo dal datasheet e dallo schema elettrico di una scheda di valutazione in commercio. A mio avviso è il sistema migliore per iniziare e procedere senza intoppi.
Quindi si può benissimo partire con la sperimentazione di un micro-controllore senza doversi per forza acquistare una scheda di valutazione o sviluppo. Questo vale chiaramente per chi sperimenta per diletto, nel caso del lavoro le cose cambiano: bisogna avere un qualcosa di sicuro, affidabile e possibilmente senza perderci troppo tempo per utilizzare invece il tempo per il lavoro, quello che permette di mettere insieme il pranzo con la cena.
In ogni caso ... **buona sperimentazione a tutti!**