Guida all'uso di Opgui e OpenProg

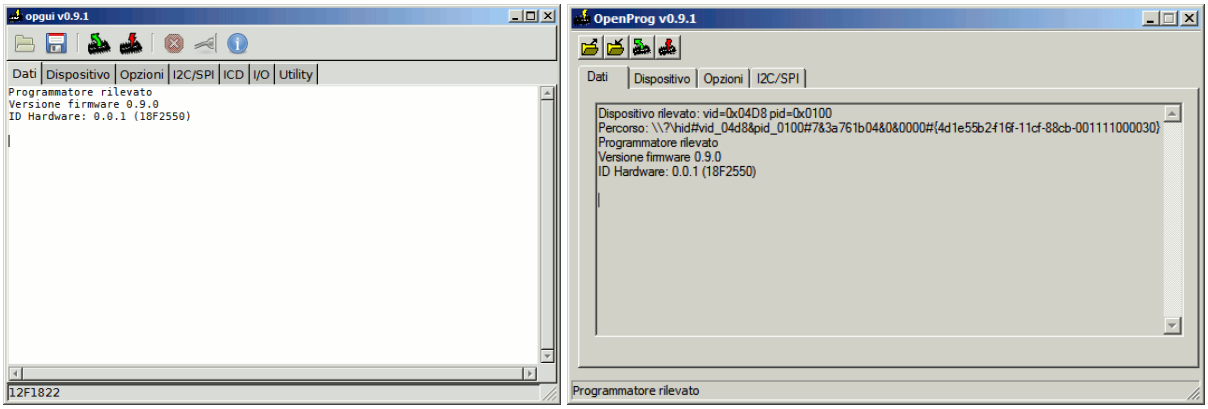
v.1.2

Novembre 2014

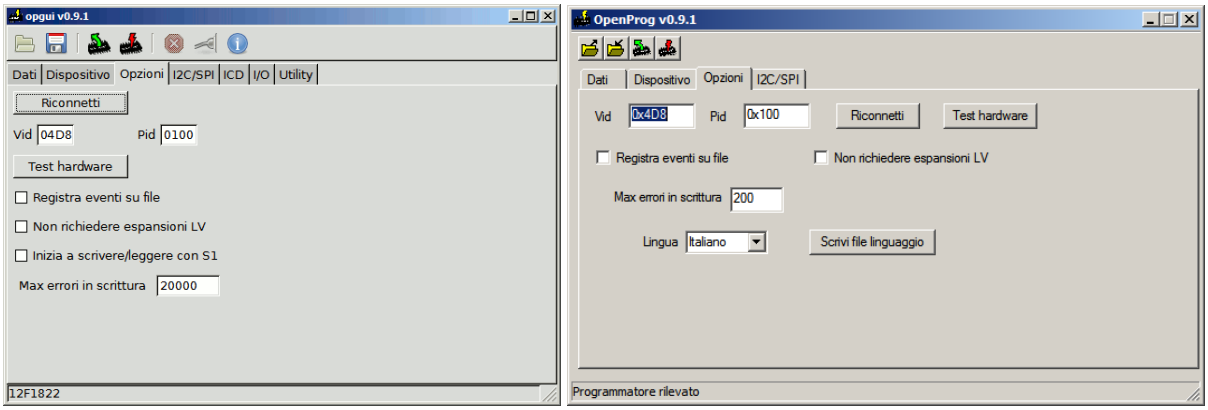
La presente guida si riferisce ai programmi Opgui e OpenProg, che servono a controllare il programmatore USB Open Programmer; OpenProg funziona solo su Windows, Opgui sia su Linux che su Windows (previa installazione del GTK runtime); il debugger ICD e alcune altre funzioni sono disponibili solo con Opgui.
Maggiori informazioni sul progetto al sito: www.openprog.altervista.org
Da qui in poi si presume che il programmatore sia funzionante e venga riconosciuto dal sistema.
Ogni figura contiene due schermate, Opgui a sinistra, OpenProg (se disponibile) a destra.

Passo 1: connessione al programmatore

All'avvio il programma effettua una ricerca del programmatore; se questo è presente nel sistema lo segnala:

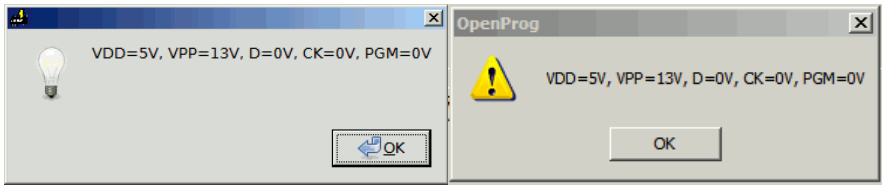


Altrimenti dalla scheda *Opzioni* il comando *Riconnetti* effettua una nuova ricerca; il programmatore viene inoltre azzerato.



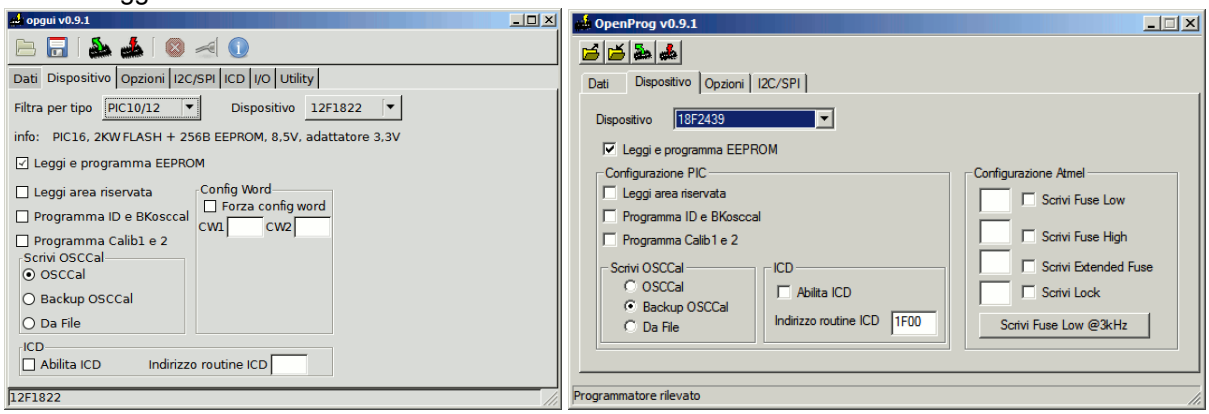
Sempre tra le opzioni si può specificare il codice VID e PID del programmatore, nel caso il firmware sia stato ricompilato con codici diversi da 0x4D8:0x100.
Registra eventi su file genera un file con tutti i dati in transito da e verso il programmatore, utile nel caso di errori.
Non richiedere espansioni LV disabilita il controllo delle schede di espansione.

Inizia a scrivere/leggere con S1: a volte è utile iniziare a leggere/scrivere alla pressione del tasto S1, per esempio nel caso di collegamenti ICSP volanti.
Max errori in scrittura specifica dopo quanti errori la scrittura viene interrotta.
Il *test hardware* serve per verificare il funzionamento del programmatore; è bene farlo almeno una volta prima di usarlo.
A ogni passo un messaggio indica la tensione presente sui segnali di programmazione ICSP; se la tensione misurata corrisponde (all'incirca) a quella visualizzata allora il circuito è funzionante; viene anche effettuata una scansione di tutte le linee di I/O per verificarne la funzionalità.
Il capitolo Problemi più comuni elenca una serie di errori che impediscono il corretto funzionamento del circuito.



Passo 2: scelta del dispositivo

Sulla scheda "Dispositivo" è possibile scegliere il dispositivo da programmare, tenendo conto che:
I PIC serie LF sono usati come quelli F, es. 16F628 = 16LF628;
gli Atmel AVR con vari suffissi sono raggruppati se usano lo stesso algoritmo,
es. ATmega8A=ATmega8;
le EEPROM 24xx e 25xx comprendono tutte le versioni con VDDmax=5V, es. 242LC56, 24AA256, ecc.
Sono anche supportate alcune memorie FLASH SPI a 3,3V; in questo caso è necessario un adattatore per abbassare la tensione di alimentazione a 3,3V.
le EEPROM 93xx richiedono due algoritmi diversi, uno per la serie 93S, uno per le altre, indicate come 93x; le 93xA sono uguali alle 93x ma con organizzazione a 8 invece che 16 bit.
Leggi e programma EEPROM: opzione da abilitare se serve usare la EEPROM interna presente in quasi tutti i microcontrollori; in tal caso se il file da scrivere non contiene dati EEPROM viene generato un messaggio di errore.



Configurare i PIC

Le varie opzioni sono usate solo se il dispositivo le supporta:
Leggi area riservata: quasi tutti i PIC hanno un'area di memoria oltre le Config Word che è riservata per test di produzione o calibrazione; per esaminarla basta abilitare questa opzione prima della lettura; nei PIC24-30-33 viene letta anche l'area di memoria executive.
Programma ID e BKosccal: scrive le locazioni ID (indirizzo 0x2000-2003 nei PIC16) e il valore di calibrazione di backup se specificati nel file (ossia se sono diversi da 0x3FFF).
Programma Calib1 e 2: scrive le locazioni di calibrazione 1 e 2 se specificati nel file.

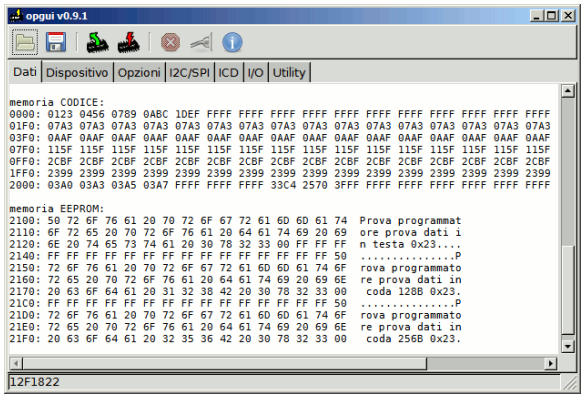
Scrivi OSCCal: in alcuni dispositivi (es. i 12F5xx) il valore di calibrazione dell'oscillatore interno è scritto, in fabbrica, all'ultimo indirizzo della memoria programma e in alcuni casi anche in una locazione di riserva oltre l'area ID; dopo una cancellazione è consigliato ripristinare la calibrazione; le opzioni sono: *OSCCal*, ossia il valore originario, che è stato automaticamente salvato prima della cancellazione; *Backup Osccal*, cioè la copia di riserva, se esiste; *Da File*, che permette di specificare un valore arbitrario nel file.
Forza config word: forza il valore delle config word (1-7, dipende dal dispositivo) durante la scrittura indipendentemente da quanto specificato nel file hex; i valori si intendono in esadecimale.
Abilita ICD: serve a scrivere l'indirizzo del debugger residente in memoria; per usare la funzione ICD è anche necessario abilitare il bit DEBUG nella config word e includere la funzione debugger monitor nel progetto; vedi capitolo Usare il debugger ICD.

Configurare gli Atmel AVR:

A differenza dei PIC, i byte di configurazione degli AVR non sono mappati nella memoria programma (e quindi nel file da scrivere) e devono essere inseriti a mano.
Non è necessario scriverli tutti, potrebbe andar bene anche il valore già presente.
Per ogni byte da modificare bisogna scrivere il valore esadecimale e abilitare la scrittura.
Maggiori informazioni si trovano sui datasheet dei dispositivi.
La frequenza della CPU limita la velocità di comunicazione a fCPU/4; l'algoritmo cerca automaticamente la massima velocità possibile, e questo influenza il tempo totale di lettura/scrittura; nel caso di dispositivi molto grandi potrebbe essere conveniente cambiare la frequenza (scrivendo un file vuoto e impostando i Fuse necessari) prima di scrivere il file desiderato.
In alcuni modelli è possibile impostare la frequenza della CPU a valori molto bassi (16kHz), tali da impedire la comunicazione col programmatore.
In questi casi si deve usare il comando "Scrivi Fuse Low @ 3 kHz" e specificare un valore di Fuse Low che imposti una frequenza maggiore; successivamente si può scrivere col metodo normale.

Passo 3: caricare un file (solo se bisogna scrivere)

Il programma supporta file hex per tutti i dispositivi e anche file binari per le memorie seriali.
Nei PIC la EEPROM è mappata nello spazio di memoria principale e quindi nello stesso file hex.
Invece per i micro Atmel serve un ulteriore file (hex o binario, spesso con estensione .eep).
Dopo il caricamento sulla scheda "Dati" appare il contenuto della memoria, comprese le locazioni di configurazione.



Passo 4: scrivere o leggere dal dispositivo

Basta usare gli appositi pulsanti sulla barra degli strumenti.

Durante l'operazione di scrittura o lettura l'interfaccia grafica risponde con un certo ritardo e indica lo stato di avanzamento totale in percentuale.
Premere "ferma" per terminare l' operazione immediatamente.
I tempi di scrittura variano a seconda della dimensione della memoria, da qualche secondo per dispositivi con 1KB di memoria ai minuti per quelli con 128KB; spesso la EEPROM interna dei microcontrollori ha un tempo di scrittura per cella molto maggiore rispetto a quello della memoria programma, anche se essendo piccola non rallenta più di tanto l'operazione; in generale la scrittura di un programma corto in un micro grande impiega comunque pochissimo.
La cancellazione, quando necessaria, è effettuata automaticamente prima della scrittura.
La verifica del codice scritto è automatica; in alcuni dispositivi è fatta durante la scrittura, in altri alla fine; in ogni caso una volta completata l'operazione appare il numero di errori rilevati.
Gli errori relativi alla (alle) config word meritano un discorso a parte: di solito non tutti i bit sono scrivibili, alcuni sono fissi a 0; quindi se nel file è specificato 1 l'errore è falso.
Un caso frequente è quello di un programma scritto su un dispositivo diverso da quello per cui è stato compilato; i bit di configurazione fissi potrebbero non coincidere, generando errori fasulli.
Consiglio di confrontare sempre il valore delle config word nel datasheet con quello specificato nel file da scrivere.
La lettura è generalmente più veloce della scrittura; per i dispositivi che lo supportano (quasi tutti i microcontrollori) viene letto anche l'ID del dispositivo.

Passo 5: salvare un file (se è stata fatta una lettura)

Stesse considerazioni del passo 3.
Nel caso di microcontrollori il programma non salva per quanto possibile le locazioni vuote (quelle con contenuto ≥ 0x3FFF).

Altre operazioni

Cancellare un dispositivo

Ogni dispositivo viene cancellato automaticamente prima della programmazione; se comunque si volesse cancellarlo basterebbe scrivere un file hex con dati validi (<0x3FFF) oltre lo spazio di memoria.
Per esempio, per i PIC12-16:
:020000040000FA
:0144010000BA
:00000001FF
E per i PIC18:
:020000040002F8
:020000000000FE
:00000001FF

Cambiare Configuration Word

E' possibile forzare le config word 1-7 indipendentemente da quanto specificato nel file hex tramite l'opzione *forza config word* dalla scheda *Dispositivo* (solo Opgui e solo PIC10-12-16-18).
Altrimenti ci sono due metodi:

cambiare il sorgente e ricompilare;
modificare il file hex: per i PIC16 la config word si trova all'indirizzo 0x2007, che nel file corrisponde a 0x400E; la riga giusta inizia con ":02400E", e l'ultimo byte è il checksum, calcolato come complemento a 2 della somma dei byte precedenti:
:02400E00XXXXYY con XXXX nuovo valore e YY checksum
Per esempio se XXXX=0370 YY=-(02+40+0E+03+70)=-C3=3D

Verificare che un dispositivo sia vuoto

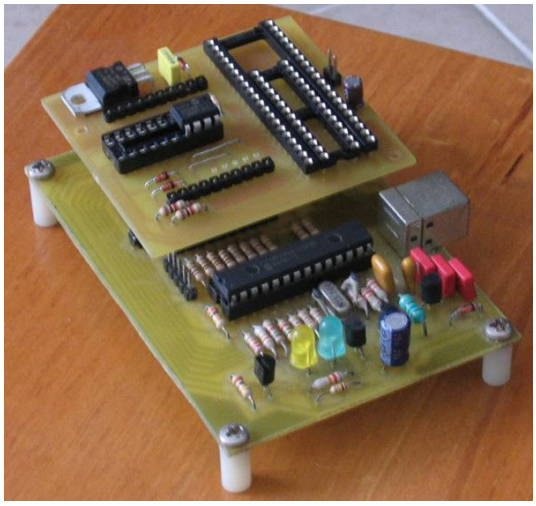
Basta leggerlo ed esaminare i dati; vengono mostrate solo le righe con dati validi (<0x3FFF per i PIC, <0xFF per gli altri), quindi se non sono visualizzati dati il dispositivo è vuoto.

Abilitare il programmatore su Linux

Le versioni Linux dei programmi si interfacciano col dispositivo /dev/usb/hiddevX (dove X è il numero assegnato dal sistema al programmatore) e hanno bisogno dei diritti di lettura su questo file I/O.
Per esempio con hiddev0 bisogna scrivere:
>sudo chmod a+r /dev/usb/hiddev0
Per abilitare permanentemente un utente alla lettura si può procedere come segue (su Ubuntu e altre distribuzioni basate su Debian, da verificare per le altre):
da root creare un file /etc/udev/rules.d/10-openprogrammer.rules
Nel caso si voglia abilitare un gruppo di utenti scrivervi:
KERNEL=="hiddev[0-9]", ATTRS{idProduct}=="0100", ATTRS{idVendor}=="04d8", GROUP="<gruppo>", SYMLINK+="openprogrammer"
in cui <gruppo> è uno dei gruppi a cui appartiene l'utente (per un elenco digitare "groups", utilizzare un gruppo adeguato e se necessario aggiungere l'utente al gruppo scelto ("addgroup <utente> <gruppo>").
Oppure, nel caso si vogliano abilitare tutti gli utenti, cambiare solo i permessi di lettura:
KERNEL=="hiddev[0-9]", ATTRS{idProduct}=="0100", ATTRS{idVendor}=="04d8", MODE="0664", SYMLINK+="openprogrammer"
Quindi riavviare udev per applicare le modifiche:
>udevadm control --reload-rules
>udevadm trigger
Ora, ogni volta che il sistema vede il programmatore, il corrispondente */dev/usb/hiddevX* ha i permessi giusti e
viene creato anche un link */dev/openprogrammer*
Se collegando il programmatore non appare il file /dev/usb/hiddevX (e il LED2 non lampeggia a 1 Hz) è sufficiente eseguire alcune volte lsusb per forzare l'enumerazione da parte del sistema, eventualmente staccando e riattaccando il cavo.

Usare le espansioni

Sulla scheda base del programmatore c'è spazio solo per i PIC fino a 20 piedini (con alimentazione a 5V) e le EEPROM I2C, per cui per usare tutti gli altri dispositivi è necessario collegare delle schede di espansione (vedi sito web).
Queste sono state progettate in modo che i connettori di espansione coincidano con quelli del modulo base, quindi basta sovrapporre le schede per agganciarle verticalmente tramite i connettori (ecco perché è consigliato usare un connettore femmina sulla base).



Importante!! I dispositivi a 3,3V non devono essere usati senza le espansioni a 3,3V, pena danni irreparabili; il programma verifica la presenza di questi adattatori prima di iniziare a programmare, ma ovviamente bisogna fare attenzione a selezionare il dispositivo giusto; i dispositivi a 3,3V sono: 12F1xxx,16F1xxx,18FxxJxx,18FxxKxx,24Fxxx,24Hxxx,33Fxxx.
Alcuni di questi hanno varianti che funzionano anche a 5V; se è necessario programmarli a 5V è possibile disabilitare il controllo dell'espansione abilitando *Non richiedere espansioni LV* tra le opzioni.
Un altro errore da evitare è inserire i 24F-33F nello zoccolo per 30F, che funziona a 5V.

Programmare via ICSP

E' possibile programmare i PIC direttamente sul circuito di applicazione, col metodo ICSP (In Circuit Serial Programming); questo richiede 5 segnali, VPP, VCC, PGD, PGC, GND, presenti nei connettori di espansione.
Su alcune schede di espansione è presente un connettore dedicato; non bisogna confonderlo con quello sul modulo base, chiamato ICSP-IN, che serve da ingresso per programmare il micro principale.
Sul circuito di applicazione bisogna evitare di caricare troppo i segnali ICSP, sia come corrente che come capacità; spesso si usano dei ponticelli per isolare il micro durante la programmazione.
Bisogna anche tenere conto che la massima corrente assorbibile da VDD è di 200-300 mA; se si va oltre la tensione cala e alla fine il programmatore si riavvia.

Usare il debugger ICD

Solo su Opgui è presente una scheda "ICD", che permette di avviare il debugger in-circuit per i dispositivi PIC16 che supportano questa modalità.
Come per la programmazione ICSP è necessario connettere i 5 segnali dal programmatore al circuito di applicazione; questa volta però VDD è opzionale, il circuito può funzionare con la propria alimentazione purché sia uguale a quella ICSP (5V o 3.3V se presente una espansione a 3.3V).
Durante una sessione di debug il dispositivo esegue una particolare routine, detta funzione di debug o debug monitor, quando incontra una condizione di halt; è questa routine che comunica col PC.
Prima di partire bisogna eseguire una serie di operazioni:

- compilare/assemblare il programma includendo la funzione di debug in uno dei due modi:
    a) inserire il codice asm del debugger nel codice principale, facendolo iniziare nella parte finale della memoria programma; a questo scopo basta inserire prima della routine una direttiva ORG 0x1F00, cioè 256 a word dalla fine per micro da 0x2000 word, oppure ORG 0xF00 per quelli da 0x1000 word, e così via. Le variabili di debug occupano i registri 0x6B-0x72, quindi nel programma principale non si devono usare queste locazioni.
    b) Includere il modulo precompilato debugger_mon.o e uno script per linker corrispondente; il modulo va aggiunto tra gli "Object files" del progetto MPLAB; lo

script per linker va modificato rispetto a quello standard che si trova nella directory lkr di MPASM; si deve aggiungere una pagina di codice per il debugger (0x1F00 per micro da 0x2000 word, 0xF00 per quelli da 0x1000 word, e così via):

CODEPAGE NAME=pageICD START=0x1F00 END=0x1FFF
SECTION NAME=ICD ROM=pageICD // ICD routine

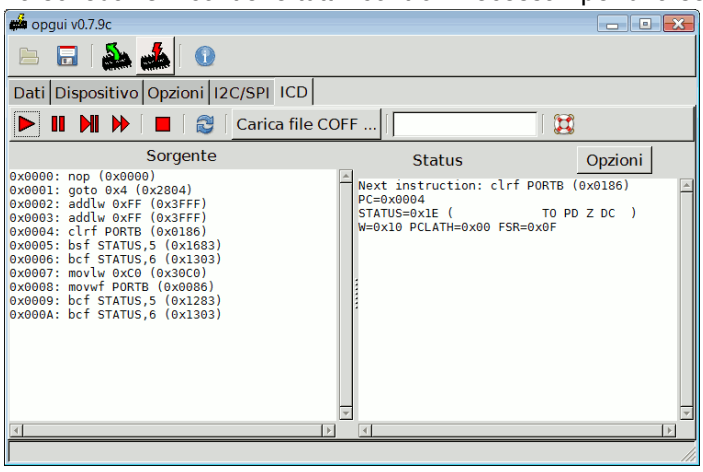
Bisogna anche accorciare la pagina precedente in modo che finisca a 0x1EFF.
Queste modifiche sono identiche a quelle usate dalla Microchip per il loro debugger (vedi la prima parte dello script, tra #IFDEF _DEBUG e #ELSE), tranne il nome della nuova pagina che è pageICD e non debug3.
Se alle strette con la memoria programma si può anche spostare la nuova pagina verso la fine, tenendo presente un minimo di circa 164 locazioni.
Per prevenire l'utilizzo delle variabili di debug (0x6B-0x72) sarebbe poi meglio accorciare le sezioni disponibili per il programma:

DATABANK NAME=gpr0 START=0x20 END=0x6A
SHAREBANK NAME=gprnobnk START=0x73 END=0x7F

E comunque non vanno utilizzate le locazioni 0x6B-0x72 nel codice programma.
Con altri ambienti di sviluppo le operazioni sono concettualmente simili.

- Abilitare il debugger tra le opzioni di configurazione (che verranno scritte nella config word):
  __FUSES (…) &_DEBUG_ON
- Inserire una istruzione NOP all'indirizzo 0 (quindi prima del solito *goto main*).
- Su Opgui abilitare l'opzione ICD nella scheda *dispositivo* e impostare correttamente l'indirizzo della routine ICD (cioè dove inizia la pagina pageICD, vedi sopra); in questo modo il dispositivo salta alla funzione di debug quando incontra la giusta condizione.
  Non è necessario farlo ogni volta, in quanto l'indirizzo ICD non viene modificato con la programmazione normale.
- Scrivere il programma assemblato sul micro.

A questo punto il micro è pronto e si può connettere via ICSP al programmatore.
La scheda *ICD* contiene tutti i controlli necessari per una sessione di debug:



Nel riquadro *Sorgente* appare il codice disassemblato nell'intorno del punto di esecuzione attuale; nell'altro riquadro invece lo stato dei registri.
Una caratteristica molto utile è quella di poter usare il vero file sorgente; ciò è possibile caricando tramite l'apposito pulsante il file .coff generato durante la compilazione del progetto; questo file contiene le definizioni di tutte le variabili usate e la corrispondenza tra indirizzo in memoria e linea di codice.

Per iniziare premere "vai/continua" ; il micro esce dal reset e si ferma all'indirizzo 1; il programma aggiorna lo stato dei registri e mostra il codice sorgente (o il disassemblato se non è stato caricato il file coff).

Per andare avanti una istruzione alla volta premere “passo” ; per fare lo stesso saltando le chiamate premere “passo sopra” .

“Vai/continua” riprende l’esecuzione fino a che non si preme “ferma” .

“Arresta” riattiva il reset e spegne l’alimentazione.

Andare passo-passo fino al punto di interesse potrebbe risultare troppo lungo; in questo caso è possibile inserire un breakpoint; basta fare doppio click sulla linea di codice su cui ci si vuole fermare (nel riquadro stato appare un messaggio di conferma) e avviare nuovamente l’esecuzione con “vai/continua”.

Il micro esegue il suo programma alla velocità nominale fino ad incontrare l’indirizzo di break; a quel punto finisce l’istruzione corrente ed salta alla funzione debug; questo significa che si ferma **dopo** il breakpoint, cosa da tenere conto quando ci si vuole fermare prima o dopo una chiamata.

Per esaminare una variabile basta fare doppio click sul suo nome (nel sorgente, e solo quando c’è il file coff); il valore verrà aggiornato per tutta la sessione di debug; doppio click ancora per rimuoverla.

Tramite le opzioni del riquadro di stato si possono visualizzare anche interi banchi di memoria o la EEPROM interna.

Altre interessanti funzionalità sono disponibili scrivendo sulla linea di comando ICD (nella barra degli strumenti ICD); oltre ai comandi base (run, step, step over, halt, stop) ci sono:

break, per impostare a mano il breakpoint;

print, per visualizzare variabili e locazioni in RAM, memoria programma, o EEPROM;

watch, per osservare variabili di continuo;

freeze, per fermare le periferiche come watchdog e timer quando il dispositivo è fermo;

help, elenca tutti i comandi;

Per cambiare il valore di una variabile o locazione di memoria si scrive:

variabile=x

es. PORTB=1F

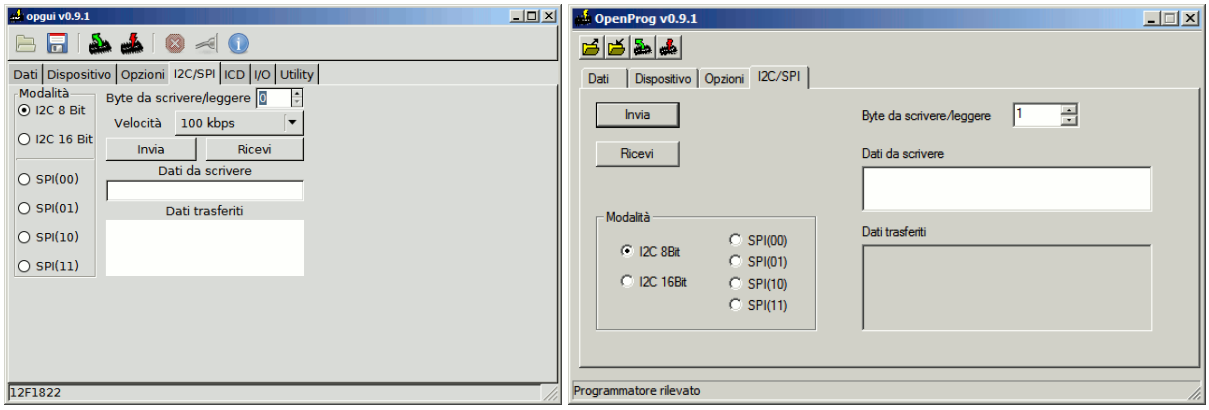
o: 0x23=FE

Consiglio di scaricare ed esaminare il progetto di esempio, compilato per 16F873 ma adattabile a tutti i dispositivi; contiene anche il sorgente del debugger e il modulo .o; bisogna comunque ricompilarlo per aggiornare il file coff con i percorsi dei file.

Una spiegazione più tecnica su come funziona il debugger si trova qui.

Comunicare via I2C e SPI

La scheda I2C/SPI permette di comunicare con dispositivi generici I2C e SPI.



I2C

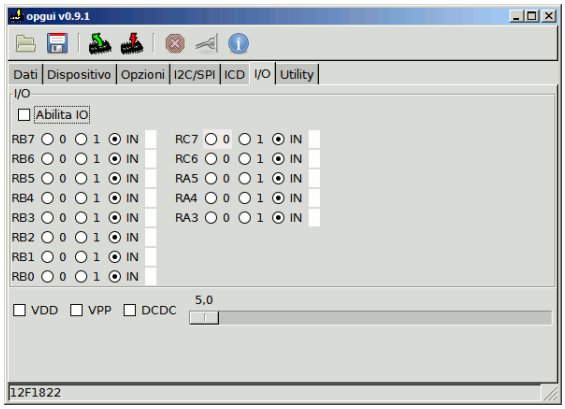
Selezionare la modalità I2C tra 8 bit e 16 bit, e la velocità tra 100kbps e 500kbps.

Con “byte da scrivere/leggere” si intendono i byte effettivi di dati; a questi si aggiunge in ogni trasferimento un byte di controllo e uno di indirizzo, oppure due di indirizzo in modalità 16 bit; il bit RW nel byte di controllo viene gestito automaticamente.
Quindi per una lettura si inseriscono tra i “dati da scrivere” due o tre byte; invece per una scrittura gli stessi due o tre byte più quelli da trasferire.
Non specificare niente equivale a scrivere 0 in ogni byte necessario.
Nel caso il dispositivo non risponda (o non ci sia) viene visualizzato un errore di acknowledge.
Ad esempio per scrivere manualmente 5 byte su una memoria 24xx16 all'indirizzo 64 i “dati da scrivere” saranno: A0 40 1 2 3 4 5, mentre per leggere vengono considerati solo i primi due.

SPI
Impostare la modalità SPI tra le 4 standard, e la velocità tra 100kbps e 800kbps.
Specificare il numero di byte da trasferire e i byte stessi se necessario; non specificare niente equivale a scrivere 0.
Il segnale CS (RB3) è alto durante ogni trasferimento.

Controllare i segnali I/O e le alimentazioni

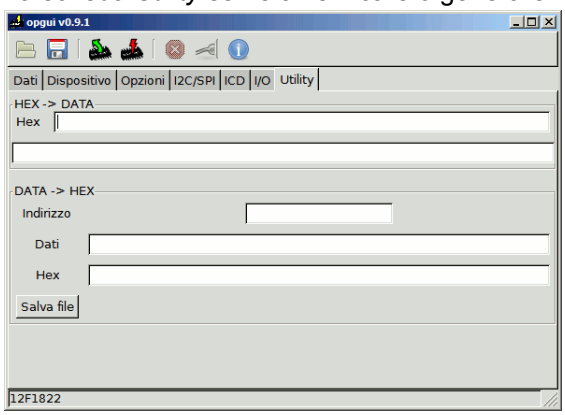
La scheda *I/O* permette di controllare in maniera interattiva tutte le linee di I/O, le due alimentazioni, e il convertitore DCDC; tutte le opzioni devono essere disabilitate prima di programmare un dispositivo.



Quando *Abilita IO* è selezionato ogni linea può essere forzata alta, bassa, o in alta impedenza (cioè in ingresso); il valore di ingresso è aggiornato ogni 100 ms.
VDD e *VPP* controllano gli omonimi segnali sul programmatore; DCDC accende il regolatore switching alla tensione specificata (tra 5V e 15V, ma il regolatore potrebbe non riuscire a fornire esattamente quella tensione).

Generare o verificare una linea di un file hex

La scheda *Utility* serve a verificare o generare le stringhe dei file .hex.



Hex>data parte da una stringa ed estrae indirizzo e dati; *data>hex* fa il contrario.

La stringa può essere salvata su file.

Cambiare lingua dell'interfaccia utente

È possibile cambiare la lingua di Opgui tramite le opzioni da linea di comando:
-langfile per generare il file con le stringhe; questo contiene tutte le stringhe utilizzate assieme al loro identificativo.
A questo punto bisogna tradurre tutte le stringhe e cambiare nome alla sezione (il nome è in cima tra parentesi quadre []).
In presenza del file languages.rc, se esiste una sezione chiamata come la lingua del sistema operativo (es. it_it per l'italiano), questa viene automaticamente selezionata.
-lang <lingua> permette invece di forzare la lingua indipendentemente dal sistema.

Con OpenProg è possibile cambiare la lingua selezionandola nella scheda *Opzioni*; il programma contiene le stringhe in Italiano e Inglese.
Per aggiungere altre lingue bisogna per prima cosa generare il file languages.rc tramite *Scrivi file linguaggio* e tradurlo come descritto sopra; il nome spunterà tra le scelte possibili per la lingua.

Inviare comandi al programmatore

Con Opgui è possibile inviare al programmatore un pacchetto USB con una sequenza di comandi:
-command [comandi]
Il pacchetto viene poi riempito con 0 fino alla sua dimensione (64 byte).
La risposta appare di seguito.
Una descrizione dei comandi disponibili si trova nel sito web del progetto.

Problemi più comuni

Descrizione	**Cause e rimedi**
Errore regolatore HV	Componenti del regolatore DCDC montati male o di valore errato
Errore di sincronizzazione con Atmel AVR	Dispositivo montato nello zoccolo sbagliato; caso frequente: ATtiny2313 va inserito nello zoccolo a 28p. Comunicazione seriale disabilitata nel dispositivo: riabilitarla con un programmatore parallelo. Cavo di collegamento al dispositivo troppo lungo: accorciarlo. Non tutte le alimentazioni del dispositivo sono collegate: collegarle. Il piedino X2 è fissato a 0: lasciarlo flottante
PGD (RB5) non si muove	Micro principale programmato male: non bisogna usare l'opzione LVP durante la programmazione.
Programmatore non rilevato, LED2 lampeggia velocemente	Reset durante l'enumerazione USB. Condensatore da 10µF staccato o di valore errato.
Errore VUSB < 4.5V	Assorbimento anomalo su VDD, controllare i transistor. Sovraccarico sull'Hub USB dovuto ad altri dispositivi collegati: staccarli tutti.
Regolatore 3,3V non rilevato	Il dispositivo selezionato richiede una espansione con regolatore a 3,3V. Nota: l'espansione è richiesta se almeno una variante del dispositivo richiede 3,3V; es. 16F1936 (5V) e 16LF1936 (3,3V) Se si è sicuri di poter programmare a 5V è possibile saltare il controllo abilitando *Non richiedere espansioni LV* tra le opzioni.
Tutti gli errori del test hardware	Corto tra le piste o piste aperte: controllare attentamente il circuito stampato o il montaggio su millefori.

Se hai dei dubbi o dei suggerimenti puoi contattarmi all'indirizzo **albertom78@gmail.com**