WALTERmwp

PIERINPIC18, IL DISPLAY E IL MULTIPLEXER

13 November 2015

Un Saluto a tutti quanti ...

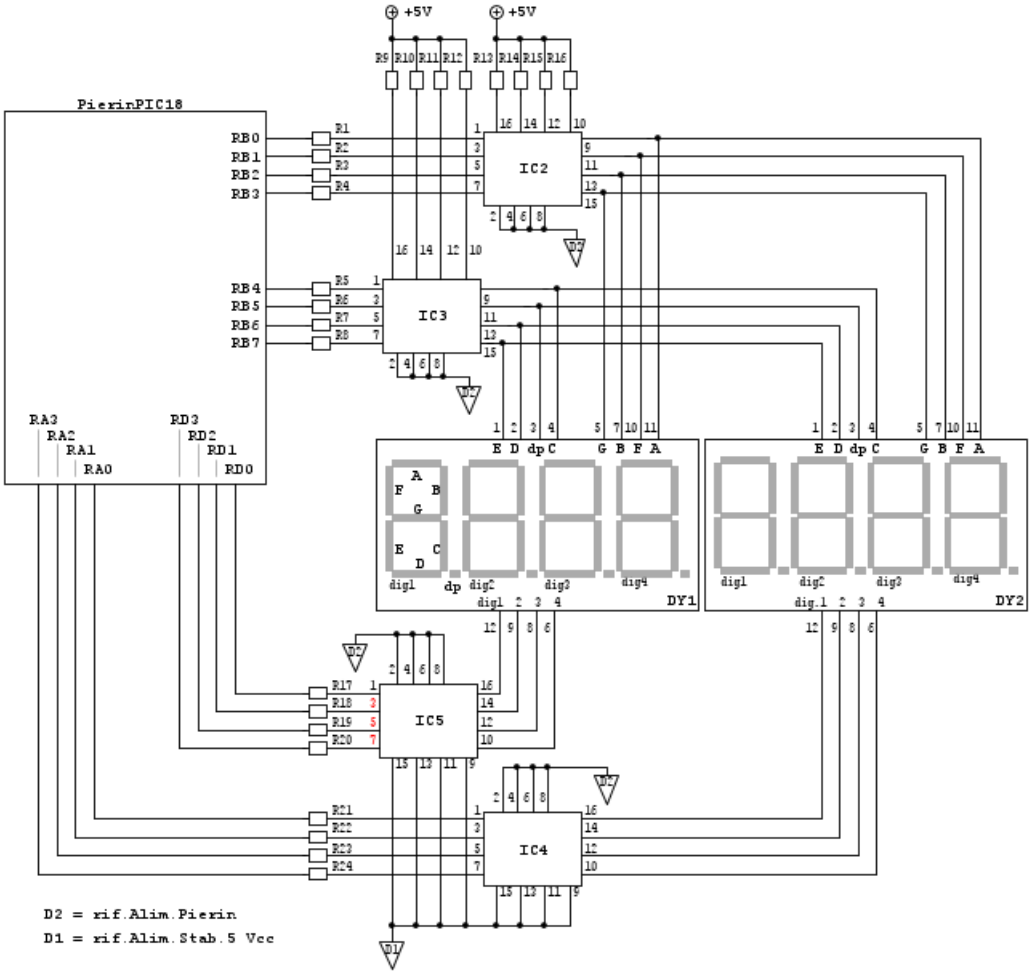
... l'antefatto

... L'intenzione era quella di realizzare un cronometro, con funzioni specifiche, particolari, così ho cominciato a cercare tra il materiale a disposizione.
Intanto mi serviva qualcosa di adeguato per la visualizzazione del tempo (... se è un cronometro ! ...) ma la scelta era quasi obbligata, per la necessità, al fine di ottenere un buon effetto visivo, quindi mi sono deciso per l'utilizzo di piccoli displays; in tale circostanza infatti un lcd non si pone proprio come alternativa.
Dal punto di vista estetico sono anche ben proporzionati e compatibili per un montaggio su circuito stampato.
Poi, ovviamente, dovevo scegliere un microcontrollore "pronto" all'assolvimento del compito e qualche altro componente a complemento ma ...
" *... mentre valuto tra quelli che "sedimentano" nel solito cassetto, mi accorgo del* ***PierinPIC18*** *che già mi osservava dalla mensola, poi guardo il cassetto, poi ancora il Pierin, ... che nel frattempo dalla mensola non si è spostato ... e continua a guardare quindi, va beh, perché no ... intanto si può sviluppare impiegandolo come muletto, la sua agilità sta anche in questo per cui mettiamo giù del codice utilizzando lui, la scelta del microcontrollore si può rimandare ...* "

Si parte col PierinPIC18 ma ... con riguardo ...

E così, posticipata la decisione, ho iniziato lo sviluppo con la schedina.
Però, una cosa è il firmware e un'altra l'hardware quindi, siccome alla schedina ci tengo e non voglio che si stressi, mi sono proposto di proteggerla preventivamente separandola dal carico che,

in tal caso, sono i leds a segmento dei displays; piccoli adesso ma, volendo, se ne potrebbero collegare di dimensioni maggiori con possibile conseguente variazione dell'assorbimento di corrente, cosa che invece non deve gravare sulla schedina, "lei" non se ne deve "preoccupare".
Per questo, e in ogni caso, opportuno che il **PierinPIC18** non dia troppa confidenza al mondo esterno.
Insomma, attenzione, " *ho un PierinPIC18 e non ho paura di usarlo* " ... ma di danneggiarlo si, quindi, cautela.
Così ho messo giù lo schemino per gestire due displays ognuno dei quali composto da quattro digits, cioè quattro numeri, otto in tutto e questo è il risultato:



schema 1

Elenco componenti:

- PierinPIC18 = schedina con microcontrollore PIC18F47J53
- IC2, IC3, IC4, IC5 = TLP521-4 (4 optoisolatori per integrato)
- R1 - R24 = 1 kohm

- DY1, DY2 = TOF-3461 (WD03641AUR-20M) (quattro digits)

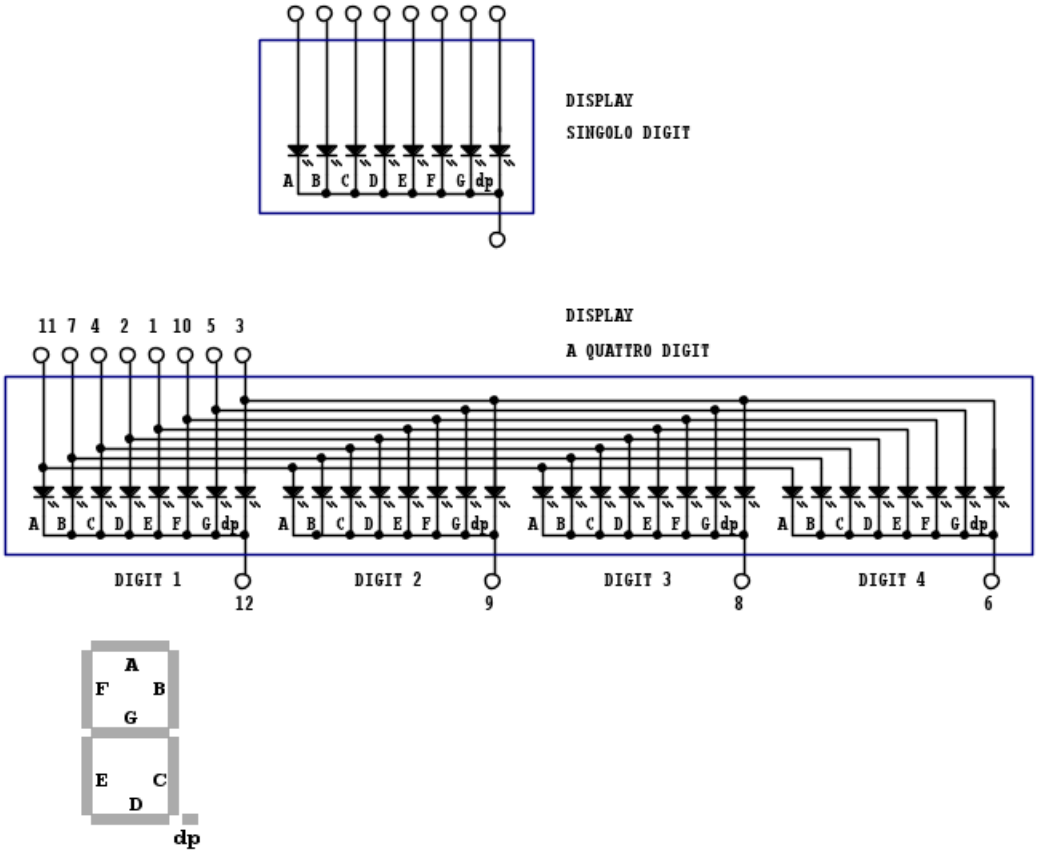
Non è davvero nulla di particolare, anzi, ma l'occasione per ricorrere a questa schedina (il suo papà putativo la chiama “schedino”) è un po un pretesto per tornare a parlare di un oggetto che, probabilmente, si trova su tante "altre" mensole e, considerato che utilizzandola si "gioca in casa", perché non pubblicare un'articoletto proponendo la gestione di una interfaccia ?
Così ora vi ritrovate "impelagati" in questa lettura ...
Va bene, l’obiettivo è definito quindi continuiamo con un minimo di considerazioni sulle scelte fatte, il circuito, i suoi componenti e il software(firmware), insomma ... quella "roba" lì ....

E’ molto semplice, al di fuori del **PierinPIC18** troviamo due displays uguali, quattro integrati, tutti uguali anche loro, e ventiquattro resistenze, anche loro, guarda caso, tutte uguali: praticamente una noia ... e invece no, col Pierin la realizzazione diventa piacevole e divertente, beh per chi l’apprezza come attività.

Pin, digit, display e il multiplexing

Si scrive di multiplexing perché in pratica si "realizza" un multiplexer per gestire i visualizzatori.
Per i più costituisce ovvia conoscenza ma un cenno al principio del multiplexing è doveroso se non altro a scopo didattico e nei confronti di chi ha poca confidenza con " ‘sta roba " nel quotidiano, dunque: consiste, tramite le medesime risorse (uscite del micro in questa circostanza), nel gestire più dispositivi periferici simili, se non uguali tra loro, conferendo l’effetto pratico e o visivo di una apparente contemporaneità allo svolgimento dell'azione che interessa i dispositivi stessi ... non si è capito niente, vero ?
... E ho usato il termine "didattico", ne avessi impiegato un altro ...
Comunque, è una tecnica molto comune che si utilizza per dare soluzione alla gestione di più dispositivi periferici quando per farli funzionare non si dispone di altrettante risorse e, di fatto, si mette in pratica una "selezione"; è il caso di una tastiera a matrice (molti tasti e pochi input/output) piuttosto che l’acquisizione di più segnali analogici tramite AD converter (appunto, un solo AD e più segnali) e altro ancora perché molti esempi si possono fare in ambito elettronico e nelle telecomunicazioni (su EY l’argomento è già stato trattato anche in altri articoli).

Per la visualizzazione abbiamo dunque optato per i displays; è noto che in commercio si trovano a singolo digit e in tal caso, scrivendo display o digit, si identifica la medesima cosa (vedi schema 2, display singolo digit) e il digit coincide con il componente.
Or bene, il digit “disegnato” è composto da sette segmenti, cioè sette leds, più un ottavo cioè un punto (punto decimale o di separazione); in funzione dei leds che vengono accesi si raffigura(compone) un simbolo, un numero.
Per "pilotare" ogni segmento, accendere o spegnere il singolo led, si deve impegnare una uscita del microcontrollore (perché noi vogliamo usare un micro ...) per cui otto uscite in tutto.



schema 2

Però, i numeri che vogliamo visualizzare sono otto quindi dobbiamo utilizzare otto digits quindi, per “raggiungere” ogni led di ogni digit ci occorrerebbero 64 uscite, 8 leds per 8 digits, che non ci sono e anche se le avessimo a disposizione sarebbe veramente uno spreco impegnarle in tal modo.
Allora cosa si fa ?
Si cerca di ottimizzare le risorse facendo in modo che la singola uscita del microcontrollore, con la quale si intende pilotare un segmento, non solo si colleghi al relativo pin del segmento in causa, ma a tutti i pins di tutti i displays che fanno riferimento al medesimo segmento; tranquilli, ha senso una "roba" del genere e lo verifichiamo poco più avanti.
Avendo però a disposizione, come tipologia di componente, un display dotato di quattro digits (non uno), scopriamo che i collegamenti tra i medesimi segmenti sono già realizzati all'interno del componente stesso e, nello specifico, dalla parte dell'anodo (vedi schema 2, display a quattro digit): il segmento A del digit 1 è collegato a quello del digit 2, fino al quarto digit e così via per gli altri segmenti.
Ma essendo due i displays dobbiamo completare il collegamento tra i segmenti del primo e quelli del secondo, cioè tra DY1 e DY2 (vedi schema 1), così da avere effettivamente ogni segmento del medesimo tipo (stessa lettera ...) di ogni digit collegato a tutti gli altri.

" *... Si, ma perché si ottimizza se si collegano insieme gli anodi dei leds, come può funzionare 'sta roba se i segmenti sono messi insieme ? Io vedo che se per esempio "metto" a 1 il* ***pin 5*** *del display intervengo sul* ***segmento G*** *del* ***digit1****, ma non solo, perché interesso anche i segmenti G degli altri tre digits, lo si vede chiaramente nello schema 2, quindi ? E poi questo basta per accenderli ?* ... "

... e questa è l'ovvia e legittima obiezione ... allora, è presto "detto", questo tipo di display è a "catodo comune" (ve ne sono anche con "anodo comune") e l'accensione del segmento(del led) avviene portando un livello di tensione positivo sul pin corrispondente all'anodo (1, 2, 3, 4, 5, 7, 10, 11) rispetto a quello applicato al pin che hanno in comune gli otto leds dello stesso digit (12, 9, 8 oppure 6) ... qui, tra l'altro, vediamo la piedinatura "dual in line" del display ...

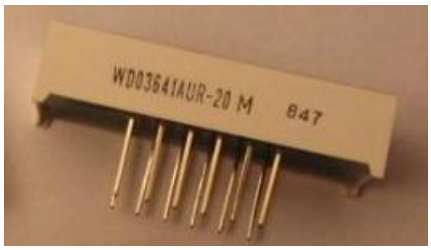


immagine 1

con dodici pins come nello schema 2 (display a quattro digits).
A complemento, nella seguente immagine, un insieme dei componenti impiegati ...

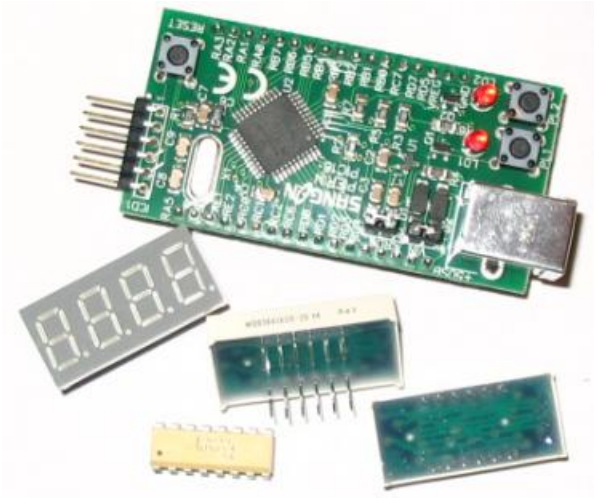
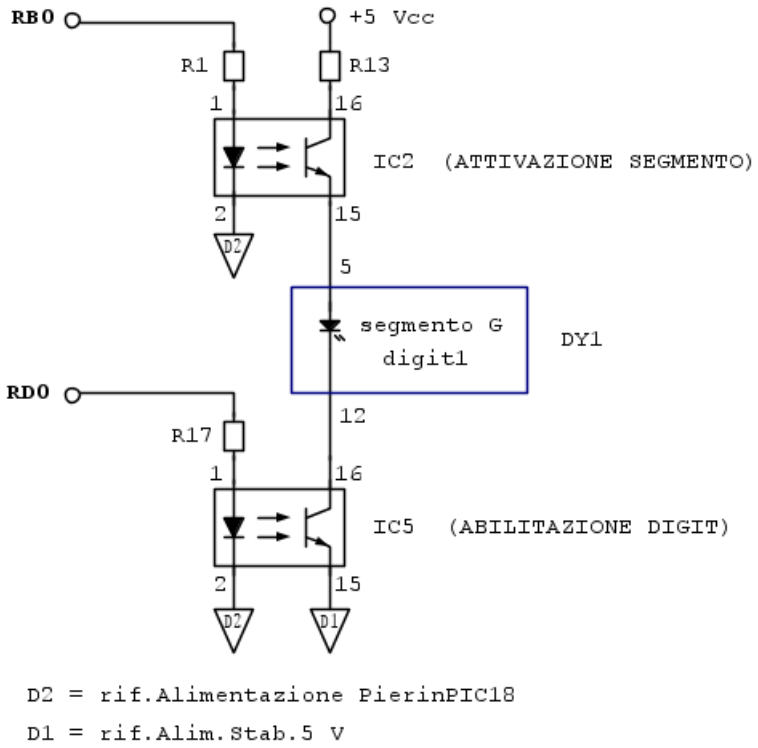


immagine 2

Per essere più precisi, assegnando opportunamente lo stato uno(1) o zero(0) ad ogni uscita del microcontrollore (**porta B** in tal caso) si stabilisce quale segmento del digit si intende "accendere" o lasciare spento (A, B, C ...); trattando il **segmento G** (è un esempio) si deve mettere a 1 l'uscita **RB0** (del micro) che agisce, tramite l'optoisolatore, sul **pin 5** del display, ma, come osservato nella domanda, tenderemmo a "pilotare" contemporaneamente tutti e quattro i **segmenti G** quindi, per

completare l'operazione, volendo appunto intervenire solo sul **digit1**, è necessario "portare" verso "massa" (il riferimento comune dell'elettronica dell'alimentatore) solo il **pin 12**, associato al digit 1, e questo avviene attivando l'uscita del micro **RD0** che agisce, anch'essa, tramite optoisolatore come tutte le altre uscite.
Osserviamo che se lasciamo **RB0** a 1 e invece di portare verso massa il **pin 12** del display, portiamo verso massa il **pin 9** (tramite **RD1**), si ottiene l'accensione del **segmento G** del **digit2** e lo spegnimento di quello del **digit1**.
Guardando lo **schema 2** e il seguente, ci facciamo un'idea più definita:



schema 3

Questo dettaglio che mette in evidenza **RB0**, come uscita che pilota un segmento (il **G**), e **RD0**, come uscita che abilita il **digit1** (il primo a sinistra nel display a quattro digit), è da considerare quale tipologia di collegamento per tutte le porte del micro impiegate nel controllo del display stesso.
A complemento possiamo prendere nota della configurazione dei pins dell'optoisolatore ...

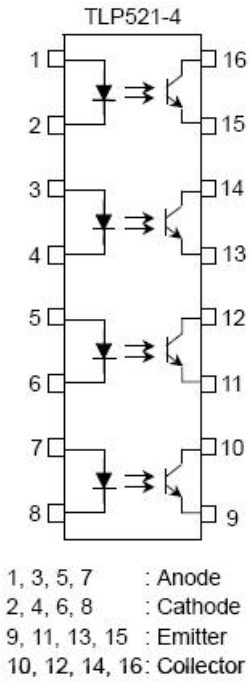


immagine 3

... e nella seguente tabella troviamo l'associazione tra pin del microcontrollore, segmento o digit e pin del display:

porta del micro	**segmento o digit**	**pin del display**
portB 0	segmento G	pin 5
portB 1	segmento B	pin 7
portB 2	segmento F	pin 10
portB 3	segmento G	pin 11
portB 4	segmento E	pin 1
portB 5	segmento D	pin 2
portB 6	segmento dp	pin 3
portB 7	segmento C	pin 4
portD 0	digit1 DY1	pin 12
portD 1	digit2 DY1	pin 9
portD 2	digit3 DY1	pin 8
portD 3	digit4 DY1	pin 6
portA 0	digit1 DY2	pin 12
portA 1	digit2 DY2	pin 9
portA 2	digit3 DY2	pin 8
portA 3	digit4 DY2	pin 6

tabella 1

Tornando a quanto accennato all'inizio, in merito alla "possibile variazione di carico", è bene considerare con attenzione la contemporanea accensione di tutti i leds di un digit.
Questa situazione comporta, per lo NPN dell'optoisolatore che abilita il digit, l'esposizione ad una intensità di corrente di circa otto volte superiore rispetto a quella che attraversa il transistor dell'optoisolatore che attiva il segmento.
Per esempio, con resistenze da 1 Kohm, ci si ritrova una corrente che attraversa il led del segmento pari a 0,518 mA; quella misurata a valle del catodo è invece pari a 4,57 mA.
In tal caso, è palese, sono valori ampiamente tollerabili ma se si inseriscono dei cambiamenti è bene tenere presente questo aspetto.
Per gli optoisolatori qui impiegati la corrente massima di collettore è di 50 mA: non è poi così difficile arrivarci collegandoci dei display più grandi.
Dipende poi dalla luminosità che si vuole ottenere.
Anche la frequenza di aggiornamento dei digits finisce per condizionare il risultato finale: naturalmente a livello di codice c'è un pochino di margine, ma non troppo.
I valori sono stati misurati in condizione di "staticità" ovvero senza applicare il multiplexing, dunque mantenendo sempre attive le uscite interessate per la verifica del caso.
E' possibile aumentare i valori delle resistenze su base analitica ma poi bisogna verificare, "in pratica", la "leggibilità" dei numeri: la luminosità dipende dall'intensità di corrente in assoluto e dalla frequenza di aggiornamento per ogni digit.
Ovviamente si possono utilizzare altri optoisolatori o cambiare proprio componente d'interfaccia; comunque, come ho scritto, non necessario, ma buona pratica separare l'unità di controllo dall'utenza (improprio definirla tale ma in questo caso il display) che si deve controllare.

Intanto che ci siamo, prima di proseguire, diamo un'occhiata a quanto è stato montato su breadboard:

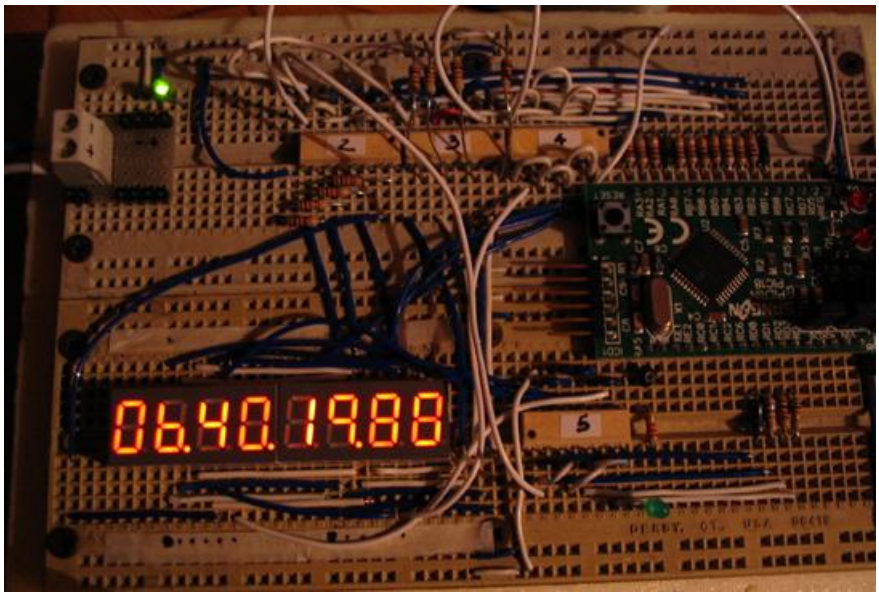


immagine 4

... e questo è l'effetto in "notturna" ...

immagine 5

Il valore visualizzato non è quello del debito pubblico di un paese della comunità europea ... ma l'espressione del tempo trascorso (dall'avvio o dalla accensione) nel formato " **hh.mm.ss.dc** " dove **hh**=ore, **mm**=minuti, **ss**= secondi, **dc**= (**d**)ecimi di secondo e (**c**)entesimi di secondo.
I quattro campi sono distinti tra loro, nell'intenzione, dal punto decimale acceso ma in pratica separa ben poco e non può sopperire alla mancanza della distanza tra le cifre, però quello c'è e facciamocelo bastare . . .

due righe per descrivere il firmware ...

Nel codice è sviluppata la dinamica per gestire contestualmente la **porta B**, la **porta D** e la **porta A**: si predispone sulla **porta B** l'informazione che si vuole visualizzare (il numero che si intende riprodurre) e sulla **porta D** oppure sulla **porta A** si attiva l'uscita associata al digit sul quale si desideri che appaia l'informazione ... "... *tutto qua ?* ..." ... beh, dal punto di vista applicativo si potrebbe rispondere di si ma naturalmente il programmino è qualcosa di più, ma lo vediamo osservando i dettagli.

Dunque, "dicevamo", il programma, ah no ... 'momento 'momento, un'altra precisazione è d'obbligo: la "struttura" che sta "intorno" alla parte applicativa, non l'ho realizzata io, infatti l'ho mutuata da quanto aveva già predisposto **TardoFreak** per coloro che avrebbero utilizzato la schedina; in sostanza ho preso il progetto riportato nell'articolo:
pierin-pic18-il-programma-demo-timer-e-super-velocita'-per-il-micro

Nel file sorgente *main.c* ho scritto ovviamente tutto il necessario facendo riferimento però alla predisposizione del timer 2 e agli interrupts a bassa priorità (i files inclusi sono rimasti inalterati).
Nel caso qualcuno interessato all'impiego della schedina, ma anche solo al microcontrollore, non l'avesse ancora fatto, suggerisco la lettura anche degli altri articoli, molto propedeutici:
- **pierin-pic18-getting-started**
- **pierin-pic18-per-imparare-e-sperimentare-con-i-microcontrollori**
- **pierin-pic18-come-creare-un-progetto-in-c-con-mplab-partendo-dal-modello-di-**

base

Ora passiamo davvero al programma ...
Nella zona dedicata alle variabili globali troviamo tutte quelle che ci servono:

```
//-----------------------------------------------------------------------------
// Variabili globali
//-----------------------------------------------------------------------------
#pragma udata
volatile unsigned int timer_delay; // Timer software
volatile unsigned int timer_vision;// Timer software
volatile unsigned int timeIsSet=0; // flag rilevazione griglia temporale
//
unsigned short c=0x00;                    // controllo scansione digits
unsigned short sel_digit=0x01;            // abilitazione digit
unsigned short checktime=0x01;            // segnale griglia temporale

unsigned short stop=0;                    // congelamento visualizzazione
//
// variabili gruppo orologio
unsigned short time_ms=0, time_sec=0, time_min=0, time_ora=0;
//
// campi con caratteri predefiniti
unsigned char image[12]={0,0,0,0,0,0,0,0,0,0,0,0};
// campi per numero del carattere da visualizzare
unsigned char image_v[12]={0,0,0,0,0,0,0,0,0,0,0,0};
// campi per visualizzazione in stato di congelamento
unsigned char image_lock[12]={0,0,0,0,0,0,0,0,0,0,0,0};
```

estratto codice 1

I due timer software sono le variabili che, decrementate e comparate determinano il "momento" in cui:

- eseguire l'aggiornamento del valore del cronometro
- eseguire l'aggiornamento dei displays

Sono due azioni ben distinte e separate.
L'altra variabile, la "**c**", stabilisce quale digit deve essere aggiornato nel momento in cui è stabilito che l'aggiornamento deve essere eseguito: è in base al suo valore che "**sel_digit**" viene modificata per controllare la **porta D** o la **porta A**.
La "**checktime**" è di "servizio" mentre la "**stop**" la utilizziamo per abilitare l'azione di "congelamento" del tempo in visualizzazione.
Le quattro che seguono, beh, si capisce, servono al conteggio del tempo.

L'array "**image[ ]** " contiene invece il valore(i valori) che, di volta in volta, deve essere preso e assegnato alle uscite della **porta B** ovvero alle "linee" affinché possa essere riprodotto sul singolo digit il numero; essendo dieci i caratteri (... da zero a nove ...) si è scelto di usare i campi con indice da 0 a 9.
L'array "**image_v[ ]** " contiene il valore quale risultato della conversione, in decine o unità, di ogni variabile facente parte del gruppo orologio; il valore del campo **image_v[ ]** viene quindi usato per scegliere tra quelli dell'array "**image[ ]** "; questa è l'assegnazione:

```
...
image[11]=image[image_v[c]];   //(**)visualizzare valore corrente
...
```

estratto codice 2

Ad esempio, se **c** vale 2 significa che dobbiamo aggiornare il secondo digit (a partire da sinistra) ovvero il numero delle unità dell'ora.
Se si è giunti a **13 ore** il numero dev'essere il **3** e questo significa che il valore contenuto nel campo **image_v[2]**, in base a quanto calcolato nella sezione precedente di "aggiornamento del valore del cronometro", corrisponde a 3.
Di conseguenza è il campo image[image_v[c]] cioè **image[3]** che viene assegnato alla variabile (image[11]) che, in ultima istanza, viene trasferita su **port B**.
E non c'è un motivo specifico per il quale ci si avvale del campo 11: è solo impiegato come variabile di appoggio ed è il solo motivo per il quale l'array è sovra dimensionato; invece di **image[11]** si può usare altra variabile a piacere.
Più avanti si può leggere l'estratto che documenta l'aggiornamento del display e l'istruzione di assegnazione sulla **Port B**; intanto questa è la riga specifica:

```
...
//***********************************************
LATB=image[11]; //QUI aggiorna uscite per segmenti
//***********************************************
...
```

estratto codice 3

Subito dopo troviamo due definizioni fondamentali che dovrebbero trovare la migliore collocazione in uno dei files inclusi mentre qui le lasciamo per raccogliere tutto in unico sorgente:

```
//-------------------------------------------------------------------------
// Define
//-------------------------------------------------------------------------
#define def_timer_delay 49;        // periodo in ms per scansione incremento tempo
#define def_timer_vision 3;        // periodo in ms per aggiornamento digit
```

estratto codice 4

L'impostazione della define **def_timer_vision** stabilisce il periodo di aggiornamento di un digit del display, in millisecondi.
La variabile **timer_vision**, inizializzata con la define, viene decrementata ogni millisecondo e giunta a zero viene abilitata la scansione per l'aggiornamento del display; l'esecuzione effettiva avviene magari qualche microsecondo dopo ma, più o meno, il tempo è quello.

Se consideriamo il **3**, tre millisecondi, vuol dire che in circa 21 millisecondi ogni digit viene "rinfrescato" una volta; significa quasi 50 volte in un secondo: per ottenere l'effetto della persistenza può andare bene.
Aumentando il valore di **def_timer_vision** inizia lo "sfarfallio" e poi l'effetto "scorrimento" da sinistra verso destra (guardando frontalmente).

Per capire il motivo per il quale si considera un determinato valore da assegnare alle uscite del micro è necessario tenere presente la tabella 1, replicata nei commenti dello script ...

```
//************************************************************************
// IDENTIFICAZIONE e ASSOCIAZIONE USCITE MICRO / DISPLAY A 4 DIGIT
//************************************************************************
//associazione uscita porta/segmento digit/pin display
// PORTB 0 /segmento G /display pin 5
// PORTB 1 /segmento B /display pin 7                  +--B3--+
// PORTB 2 /segmento F /display pin 10                B2      B1
// PORTB 3 /segmento A /display pin 11                 +--B0--+
// PORTB 4 /segmento E /display pin 1                 B4      B7
// PORTB 5 /segmento D /display pin 2                  +--B5--+ o B6
// PORTB 6 /segmento dp /display pin 3
// PORTB 7 /segmento C /display pin 4
//
//associazione uscita PORTA D/digit display/pin display
// PORTD 0 /digit 1 display DY1 /display pin 12
// PORTD 1 /digit 2 display DY1 /display pin 9
// PORTD 2 /digit 3 display DY1 /display pin 8
// PORTD 3 /digit 4 display DY1 /display pin 6
//associazione uscita PORTA A/digit display/pin display
// PORTA 0 /digit 1 display DY2 /display pin 12
// PORTA 1 /digit 2 display DY2 /display pin 9
// PORTA 2 /digit 3 display DY2 /display pin 8
// PORTA 3 /digit 4 display DY2 /display pin 6
//
// significato: (hh)   (hh)   (mm)   (mm)  | (ss)   (ss)   (ms)   (ms)
// digit      : dig1   dig2   dig3   dig4  | dig5   dig6   dig7   dig8
// porta      :  D0     D1     D2     D3   |  A0     A1     A2     A3
```

```
// display     :          DY1           +          DY2
//
```

estratto codice 5

Se voglio visualizzare ad esempio il numero tre, devo mettere a **1** le uscite **RB0**, **RB1**, **RB3**, **RB5** e **RB7**; il che equivale a scrivere sulla **porta B** il valore **0xAB**.
Queste sono le relative dieci inizializzazioni:

```
// Inizializza le variabili dalle quali recuperare la composizione
// del carattere che si vuole riprodurre col digit
image[0]=0xBE;                          //carattere 0
image[1]=0x82;                          //carattere 1
image[2]=0x3B;                          //carattere 2
image[3]=0xAB;                          //carattere 3
image[4]=0x87;                          //carattere 4
image[5]=0xAD;                          //carattere 5
image[6]=0xB5;                          //carattere 6
image[7]=0x8A;                          //carattere 7
image[8]=0xBF;                          //carattere 8
image[9]=0x8F;                          //carattere 9
```

estratto codice 6

tre porzioni di codice

Il programma è un loop infinito all'interno del quale distinguiamo chiaramente due sezioni dedicate all'assolvimento delle funzioni base e la parte finale:

- aggiornamento cronometro e definizione del numero da visualizzare
- gestione in multiplexing della **porta B**
- gestione dei pulsantini

Questo il codice per l'aggiornamento del cronometro:

```
//*****************************************************************
// SEZIONE DEDICATA ALL'INCREMENTO/CONTROLLO DELLE VARIABILI UTILIZZATE
// PER IL CRONOMETRO E PER LA DEFINIZIONE DEL NUMERO DEL CARATTERE DA
// VISUALIZZARE
//*****************************************************************
if(timeIsSet)
{    //!!! qui entra ogni "timer_delay" ms !!!
    timeIsSet=0;
    // variabile per gestione controllo frequenza su output RA5
    checktime^=0x01;
```

```
//**********************************************************************
// GESTIONE CRONOMETRO
//**********************************************************************
time_ms+=5;
//calibrare questa costante di comparazione con "time_ms"
//in base al valore stabilito per "def_timer_delay"
if(time_ms>95)
{        time_ms=0;
    if(++time_sec>59)
    {    time_sec=0;
         if(++time_min>59)
         {    time_min=0;
              if(++time_ora>23) time_ora=0;
         }
    }
}
//**********************************************************************
// DEFINIZIONE DEL NUMERO DEL CARATTERE DA VISUALIZZARE
//**********************************************************************
//millisecondi
if(time_ms>9)
{    image_v[7]=time_ms/10;            //centinaia di ms
     image_v[8]=time_ms-image_v[7]*10; //decine di ms
}
else
{    image_v[7]=0;                     //centinaia di ms
     image_v[8]=time_ms;               //decine di ms
}
//secondi
if(time_sec>9)
{    image_v[5]=time_sec/10;           //decine di sec
     image_v[6]=time_sec-image_v[5]*10;//unità di sec
}
else
{    image_v[5]=0;                     //decine di sec
     image_v[6]=time_sec;              //unità di sec
}
//minuti
if(time_min>9)
{    image_v[3]=time_min/10;           //decine di min
     image_v[4]=time_min-image_v[3]*10;//unità di min
}
```

```
    else
    {    image_v[3]=0;                    //decine di min
         image_v[4]=time_min;             //unità di min
    }
    //ore
    if(time_ora>9)
    {    image_v[1]=time_ora/10;          //decine di ore
         image_v[2]=time_ora-image_v[1]*10;//unità di ore
    }
    else
    {    image_v[1]=0;                    //decine di ore
         image_v[2]=time_ora;             //unità di ore
    }
}
```

estratto codice 7

Notare quanto abbiamo prima scritto in merito ai campi dell'array **image_v[ ]**.
Questa è la sezione per la gestione del multiplexer:

```
//*********************************************************************
// SEZIONE GESTIONE DEL MULTIPLEXER PER LA VISUALIZZAZIONE
//
//*********************************************************************
if(!timer_vision)
{
    if(++c>8) {c=1; sel_digit=0x01;}

    if(sel_digit>0x08)                //selettore digit coordinato per abilitazione
        sel_digit=0x01;        //da PORTA oppure PORTD

    //(*)esegue copia del numero di carattere da visualizzare
    if(!stop)
    {
        image_lock[c]=image_v[c]; //(*)

        //(**) in funzione del numero di carattere presente nella posizione
        //indicata dalla variabile "c" si seleziona dall'array "image[]" il
        //campo con valore da caricare su PORTB
        image[11]=image[image_v[c]]; //(**)visualizzare valore corrente
    }
    else
        image[11]=image[image_lock[c]];        //visualizzazione congelata
```

```
    //stabilisce se accendere punto decimale e aggiorna di conseguenza
    if(c==2 || c==4 || c==6) image[11]|=0x40;

    //*********************************************
    LATB=image[11]; //QUI aggiorna uscite per segmenti
    //*********************************************

    //in base alla variabile "c" determina uscita da attivare per abilitare
    //il digit di pertinenza; i primi quattro digit sono sotto il controllo
    //della PORTD(0:3), gli altri quattro gestiti tramite PORTA(0:3)
    if(c<5)
        LATD=LATD & 0xF0 | sel_digit;         //aggiorna uscite PortD per selezione dig
     else
        LATD&=0xF0;  //aggiorna uscite PortD per selezione digit
    if(c>4)
        LATA=LATA & 0xF0 | sel_digit; //aggiorna uscite PortA per selezione digit
    else
        LATA&=0xF0; //aggiorna uscite PortA per selezione digit

    sel_digit<<=1; //sposta di una posizione il bit di selezione del digit

    // ripristina la variabile del timer "software"
    timer_vision=def_timer_vision;
}
```

estratto codice 8

Ritroviamo il criterio di assegnazione del campo dell'array **image[ ]** alla variabile di appoggio (image[11]) e poi alla **porta B**.
La gestione della **porta D** e della **porta A**, che condividono l'abilitazione dei digits, è fatta in modo tale da lasciare inalterati gli stati dei pins (degli I/O) non utilizzati per tale funzione, ricorrendo al "mascheramento" tramite gli operatori "bitwise".

Le ultime sei righe di codice provvedono alla variazione dell'uscita **RA5** (disponibile per un eventuale controllo con strumentazione) alternando lo stato logico 1/0 così da proporre la frequenza con la quale si aggiorna il cronometro (vedere prima sezione) e alla interpretazione della richiesta di "congelamento" della visualizzazione:

```
    //uscita per oscilloscopio
    if(checktime) LATA=LATA | 0x20;
    else LATA=LATA & 0xDF;
```

```
//---- verifica PL1
if (!PORTDbits.RD4) {LATDbits.LATD6 = 1; stop=0;}
else LATDbits.LATD6 = 0;

//--- verifica PL2
if (!PORTDbits.RD5) {LATDbits.LATD7 = 1; stop=1;}
else LATDbits.LATD7 = 0;
```

estratto codice 9

Una nota a complemento in merito a qualche riscontro eseguito nel corso dell'esperienza.

L'impostazione della define **def_timer_delay** definisce il periodo di aggiornamento del cronometro.
Costituisce infatti l'inizializzazione della variabile **timer_delay** che, decrementata ogni millisecondo, viene comparata per determinarne l'azzeramento in concomitanza del quale si porta a valore significativo la variabile **timeIsSet**.

L'esito positivo del test sulla variabile **timeIsSet** è condizione che abilita all'aggiornamento del cronometro di un numero di millisecondi pari alla frazione determinata dalla stessa define: se vale **200**, l'incremento del cronometro corrisponderà a un quinto di secondo.

La "griglia" temporale che scandisce l'accesso alla routine di interrupt, all'interno della quale sono poi gestite le variabili di riferimento per i periodi di aggiornamento cronometro e visualizzazione, è determinata e conseguenza dell'uso del **PLL**.
In funzione dei vincoli esistenti (si veda il cap.3.0 "oscillator configurations" del datasheet del microcontrollore) e ai calcoli fatti, il valore da assegnare al registro **PR2**, per ottenere l'interrupt ogni millisecondo dovrebbe corrispondere a **150**.
E' possibile però che lo switch context o altri fattori (tolleranze dei componenti) possano indurre ad operare delle rettifiche.
Per questo, invece di 150, al registro ho assegnato **149**.
Può darsi che ad altri non occorra intervenire a questo livello ma sarebbe di interesse sapere se il caso dovesse necessitare o meno della rettifica.
Nel dubbio, si può sempre commettere un errore (che ancora adesso non escludo a priori) o una svista, prima di pubblicare, ho cercato il riscontro da parte di qualcun altro, così mi sono rivolto a **Paolino** che, dando la sua disponibilità, cosa della quale lo ringrazio, ha fatto le verifiche del caso confermando quanto sopra riportato.

... ma il programma cosa fa ?

Domanda legittima e risposta semplice: ben poco.
Come già scritto non è nulla di articolato ma può rappresentare un utile riferimento riprendendo

la gestione dei displays o costruendo sullo stesso **main** qualcosa di più sofisticato.
Di fatto, al momento, la sua operatività si riduce a questo:

- all'accensione, o a seguito della pressione del pulsantino di reset, il cronometro si azzera e istantaneamente riprende il conteggio
- se premo il pulsantino PL2 la visualizzazione viene "congelata" (freeze) ma il cronometro prosegue il conteggio
- se premo il pulsantino PL1 la visualizzazione riprende proponendo il valore al quale nel frattempo è giunto il cronometro per proseguire con la dinamica dell'aggiornamento.

Ecco, da qui in avanti ... le nostre strade si dividono ... insomma la "roba" è qui, è poca cosa, ma si aggiunge al materiale esistente che riguarda il Pierin; naturalmente, la "logica" la si può trasportare anche su un'altro componente (microcontrollore) a piacere.

... per chi è un po più "timido" ... wake-up PiErYiners, wake-up ...

Gli strumenti (hardware e software) necessari per operare con il **PierinPIC18** sono indicati negli articoli realizzati da **TardoFreak** e già sopracitati, quindi non occorre aggiungere altro.
Quello che qui vi ho proposto si è basato su loro utilizzo, nulla di più.
Tra l'altro non sono ricorso nemmeno al PicKit3 in parziale alternativa del quale ci si avvale appunto del **bootloader** predisposto proprio per questo motivo sul microcontrollore stesso.
Ma è chiaro che per gestire la schedina in questo modo è necessario avviare anche l'applicazione su PC cioè il programma **HIDBootloader(Windows).exe** (si, io in ambiente Windows XP Pro).

Allora, nel caso qualcuno si fosse trovato di fronte a qualche incertezza, "spendiamo" altre due righe per vedere nel dettaglio cosa accade quando si usa questo "strumento" ...
Alla attivazione, se il Pierin non è collegato alla **USB** o, pur essendo collegato, sullo stesso Pierin non è attivo il **bootloader**, l'interfaccia sul PC si presenta così ...

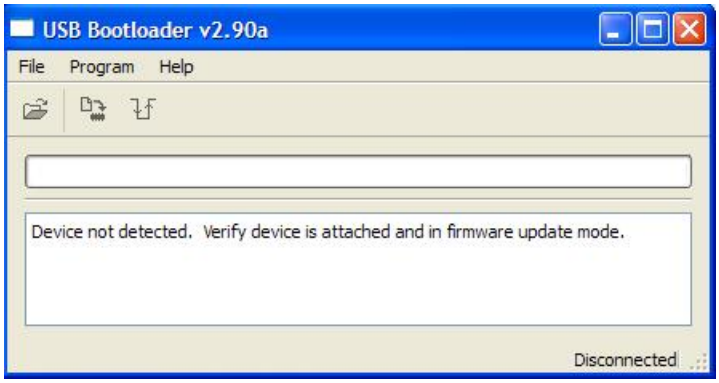


immagine 6

La schedina, in questa condizione, riceve solo l'alimentazione dalla porta del PC (... poi dipende anche da come sono disposti i ponticelli ...) e fa "funzionare" l'ultimo programma che è stato

caricato: se non è mai stato fatto nulla è presente quello che ha predisposto **TardoFreak**.
Quindi per fare interagire la schedina con lo **HIDBootloader**, si deve fare in modo che su **PierinPIC18** "parta" il firmware relativo al **bootloader**. Per farlo bisogna premere il pulsantino di **reset** e poi anche il **PL2**, quindi rilasciare il **reset** e poi il **PL2**; se sul PC il sonoro è abilitato si sente il suono del sistema che segnala il rilevamento della periferica sulla porta USB. L'interfaccia si presenta ora così ...

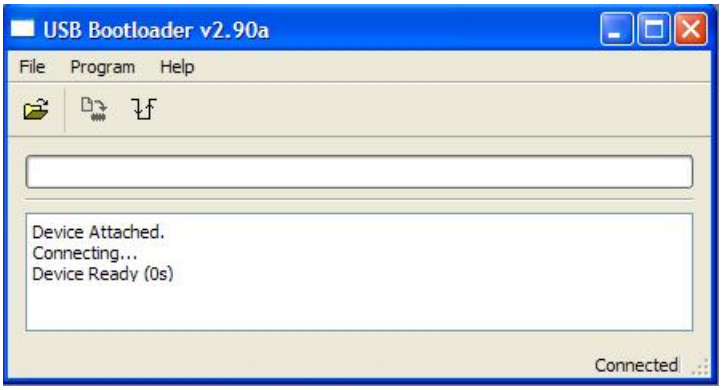


immagine 7

Si procede selezionando " *File* " e nel menu a tendina " *Import Firmware Image* " ...

immagine 8

e a questo punto si apre la canonica "finestra" (in tal caso *Open Hex File*) che invita alla ricerca e selezione del file (il programma, cioè il firmware) che si intende trasferire sul Pierin. Il file è situato all'interno della cartella "output" che a sua volta è contenuta in quella dell'applicazione ovvero quella che abbiamo utilizzato tramite l'**IDE**. Il file, con estensione "**.hex**" ...

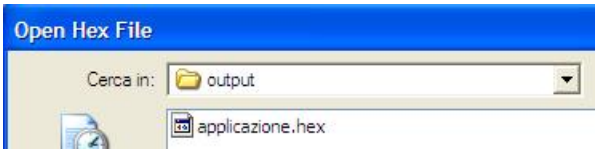


immagine 9

... è a questo punto facilmente individuabile; dunque lo si seleziona, si "preme" il pulsante sottostante " *Apri* " e come conseguenza l'interfaccia ci segnala che ...

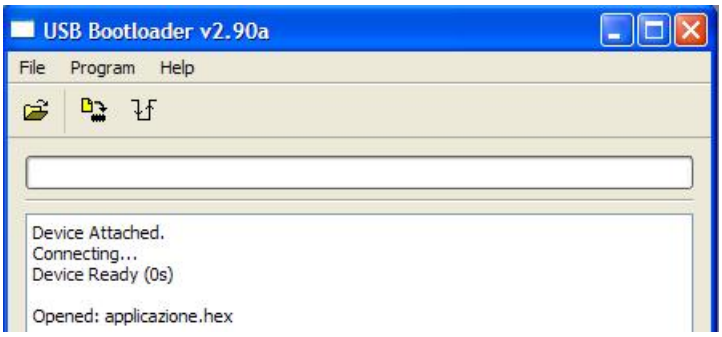


immagine 10

il bootloader è pronto al trasferimento verso/sulla schedina ... ah, nel frattempo, avrete notato che i due leds sul Pierin alternandosi lampeggiano freneticamente.

Non resta che selezionare, nella parte superiore della finestra, l'iconcina " *Erase/Program/Verify Device* " per avviare l'operazione; per un istante i led si "fermano" e, precisamente, sino a che non è terminata la fase di cancellazione; poi tornano a lampeggiare e l'interfaccia, dopo avere segnalato nella finestra i vari passaggi, si presenta così ...

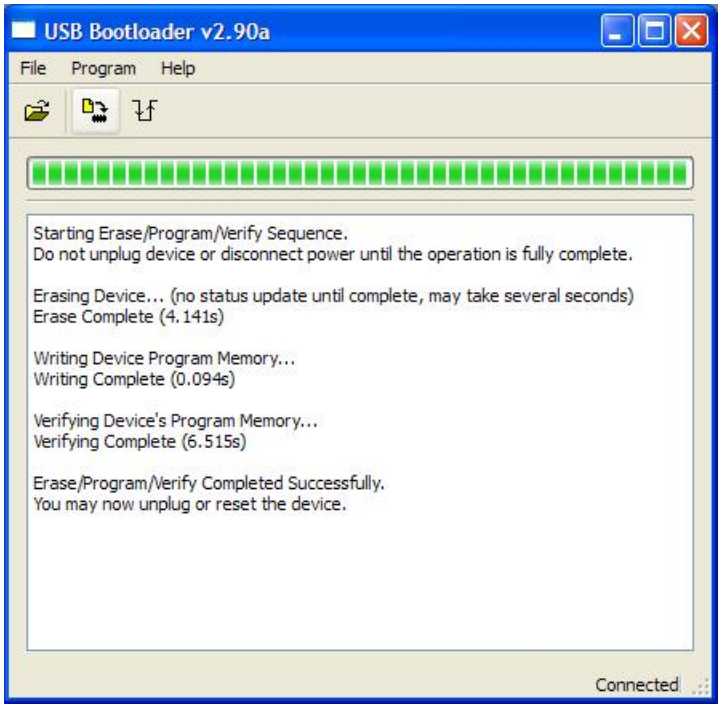


immagine 11

Giunti a questo punto ci si trova con il programma "caricato" sul Pierin (intanto il leds continuano a lampeggiare ...) e non resta altro da fare che premere il pulsante di reset e rilasciarlo per fare "partire" l'applicazione (il nuovo programma); il fatto che il Pierin non risulta più collegato

alla interfaccia **HIDBootloader** lo si capisce anche perché il PC emette il suono che segnala la "disconnessione" di una periferica dalla porta USB.

download del progetto

Questo "paragrafetto" è soggetto a possibili modifiche.
Al momento, per evitare di coinvolgere **Admin** negli aggiornamenti degli allegati sfrutto come appoggio, per mettere a disposizione il software, un Post del thread dedicato al **PierinPIC18** e precisamente **questo**.
Vi trovate sia il progetto realizzato nell'**IDE MPLAB version 8.92**, compilatore **C18**, sia quello predisposto per **XC8** da **Paolino** che cortesemente lo ha adattato.

Attenzione però: preciso che, quello per **C18**, è da utilizzare con il bootloader, quello per **XC8** non è da utilizzare con il bootloader ma bensì col **PicKit3**.

Però non escludo che in seguito **Paolino** lo possa rendere compatibile con il bootloader.

... in conclusione ...

Se l'oggettino, ovvero il **PierinPIC18**, è nelle vostre disponibilità, a mio semplice parere vale l'impegno per utilizzarlo e farci qualcosa di interessante, anche se, come al solito, il limite maggiore rimane la fantasia.
Quello che ho proposto, lo riscrivo, è una cosa facile, anche prevedibile, ma potrebbe comunque tornare utile a qualcuno, magari per integrare la propria applicazione, rendendo "leggibili" misure o parametri.
Tra l'altro, mentre scrivevo l'articolo ho scoperto che proprio **TardoFreak** aveva proposto in uno dei suoi primi articoli la gestione dei display, questo è il riferimento: **PIC e display a 7 segmenti con multiplexing**.
Il sito potrebbe contenere altri contributi simili ma non ho fatto ricerche quindi, nel caso, non se ne abbiano gli interessati se non sono menzionati: segnalatemelo e aggiungo il link.
Nella dissertazione posso essere stato impreciso, se non aver fatto degli errori, per cui osservazioni pertinenti sono ben accette.
Se siete arrivati fin "qua" posso solo ringraziarvi per il tempo dedicato alla lettura.
"WALTERmwp"

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Waltermwp:pierinpic18-il-display-e-il-multiplexer"