Versione N° 1.1

LaurTec

Leggere e Scrivere dati via USB

Emulazione porta seriale (Classe CDC)

Autore : Mauro Laurenti

email: info.laurtec@gmail.com

ID: AN4001-IT

INFORMATIVA

Come prescritto dall'art. 1, comma 1, della legge 21 maggio 2004 n.128, l'autore avvisa di aver assolto, per la seguente opera dell'ingegno, a tutti gli obblighi della legge 22 Aprile del 1941 n. 633, sulla tutela del diritto d'autore.

AVVERTENZE

I progetti presentati non hanno la certificazione CE, quindi non possono essere utilizzati per scopi commerciali nella Comunità Economica Europea.

Chiunque decida di far uso delle nozioni riportate nella seguente opera o decida di realizzare i circuiti proposti, è tenuto pertanto a prestare la massima attenzione in osservanza alle normative in vigore sulla sicurezza.

L'autore declina ogni responsabilità per eventuali danni causati a persone, animali o cose derivante dall'utilizzo diretto o indiretto del materiale, dei dispositivi o del software presentati nella seguente opera.

Si fa inoltre presente che quanto riportato viene fornito così com'è, a solo scopo didattico e formativo, senza garanzia alcuna della sua correttezza.

L'autore ringrazia anticipatamente per la segnalazione di ogni errore.

Indice

Introduzione

Collegare un circuito al PC è una di quelle cose che entusiasma tutti gli appassionati di elettronica ed informatica. Un tempo il passo era senza dubbio per la porta parallela, la cui semplicità d'uso permetteva di ottenere con poco sforzo applicazioni interessanti. La porta parallela è da tempo scomparsa dai portatili, che a meno di una docking station non hanno più molti connettori. Lo standard RS232, ancora molto utilizzato in ambito industriale e sistemi embedded sta ancora sopravvivendo, ma molti portatili non l'hanno più...

La fine della porta parallela e seriale è dovuta all'introduzione del bus USB che permette di svolgere, tra le molte mansioni, quelle per cui erano nate la porta parallela e seriale, ovvero gestire la stampante e i modem. Un sistema che si colleghi al computer per mezzo di una porta USB ha ancora una lunga vita...Impariamo ad utilizzarla facendo uso del PIC18F4550...

Prima di iniziare...

La porta USB (Universal Serial Bus) per sua natura risulta più complessa sia della porta parallela che della porta seriale. Ciononostante la porta USB è stata ideata in maniera tale che potesse essere utilizzata potendosi dimenticare spesso della complessità che è racchiusa nel suo protocollo. Nei paragrafi che seguiranno non si entrerà in merito ai dettagli associati al protocollo, infatti si farà uso di driver già pronti e della libreria Microchip MCHPFSUSB versione 2.8, utilizzabile per ogni PIC18 con modulo USB[1]. In particolare si farà uso del PIC18F4550 e della schede di sviluppo EasyUSB e Freedom II, ciononostante altri PIC con modulo USB e schede di sviluppo potranno essere utilizzate.

Dal momento che non si entrerà in merito al protocollo USB e poiché si farà uso di una porta seriale emulata, le seguenti conoscenze sono richieste prima di procedere:

- Conoscenza base della porta USB[2]
- Conoscenza base del protocollo RS232[3]
- Buone conoscenza del C18[4]
- Buona conoscenza del PIC18F4550[5]
- Buona conoscenza della scheda di sviluppo EasyUSB o Freedom II[6]

Oltre alle conoscenze sopra menzionate sarà necessario avere la libreria MCHPFSUSB, di cui si farà uso, e bisognerà avere MPLAB e il compilatore C18 già installato.
Probabilmente arrivati a questo punto o avete pensato...continuo con la semplice porta seriale o ci avete ripensato...
In realtà vi renderete presto conto che facendo uso di driver e librerie già scritti, il livello di conoscenza richiesto per realizzare un dispositivo (device) USB non è elevato.

---

1 La Libreria supporta anche PIC a 16 e 32 bit.
2 Fare riferimento all'argomento USB dell'enciclopedia online Wikipedia.
3 Le conoscenze necessarie per affrontare questo Tutorial possono essere trovate nel Tutorial “Il protocollo RS232” scaricabile gratuitamente dal sito www.LaurTec.it..
4 Le conoscenze necessarie per affrontare questo Tutorial possono essere trovate nel libro “C18 Step by Step” scaricabile gratuitamente dal sito www.LaurTec.it.
5 Oltre alle conoscenze tecniche descritte sul libro “C18 Step by Step” è bene approfondire le conoscenze sulla circuiteria di clock e sul modulo USB, direttamente sul datasheet del PIC18F4550. Molti dettagli dei registri non serviranno ma è bene aver letto almeno una volta i paragrafi relativi agli argomenti citati.
6 Maggiori dettagli sulla scheda di sviluppo EasyUSB e Freedom II possono essere trovati al sito www.LaurTec.it.

Impostare la scheda EasyUSB

Prima di iniziare vediamo come impostare impostare le schede di sviluppo. La scheda EasyUSB riportata in Figura 1 è compatibile con la scheda Microchip PICDEM™ FS USB per cui è possibile eseguire i vari esempi per PIC18F4550 presentati dalla Microchip stessa senza dover apportare alcun cambiamento.

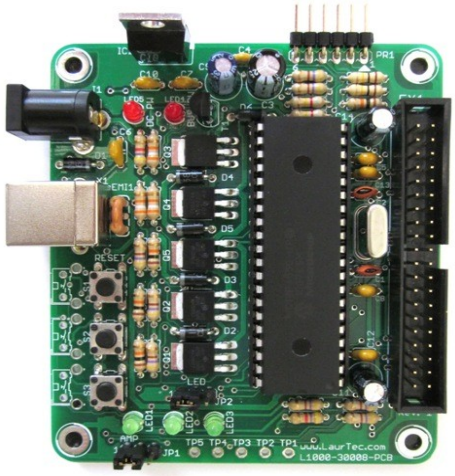

Figura 1: Scheda di sviluppo EasyUSB

Ciononostante gli esempi che fanno uso del sensore di temperatura, del trimmer e della porta seriale non possono essere eseguiti senza l'aggiunta di Hardware esterno. La ragione è semplicemente legata al fatto che la scheda di sviluppo EasyUSB è stata pensata per poter essere utilizzata in applicazioni generiche per cui hardware non sempre utilizzabile è stato eliminato.

Si ricorda che il PCB della scheda EasyUSB può essere richiesta, previa donazione di supporto[7], alla sezione servizi del sito www.LaurTec.it. Per coloro che dovessero avere difficoltà nel reperire i componenti, è disponibile anche la possibilità di richiedere il kit da montare o già montato.

[7] Donazioni di supporto permettono la realizzazione di nuovi progetti e realizzazioni di articoli come quello che stato leggendo.

Impostare la scheda Freedom II

La scheda di sviluppo Freedom II, riportata in Figura 2, è progettata per scopi didattici e permette di sviluppare una grande varietà di esercitazioni. Proprio per tale ragione, diversamente dalla scheda EasyUSB, bisogna impostarla al fine di poter operare correttamente con il modulo USB.

Si ricorda che il PCB della scheda Freedom II può essere richiesto, previa donazione di supporto[8], alla sezione servizi del sito www.LaurTec.it. Per coloro che dovessero avere difficoltà nel reperire i componenti, è disponibile anche la possibilità di richiedere il kit da montare o già montato.

Figura 2: *Sistema di sviluppo Freedom II*

Come prima cosa la scheda deve montare il PIC18F4550, al quale si farà riferimento nei progetti di esempio che verranno a breve illustrati. Oltre al PIC, per utilizzare propriamente l'USB è necessario impostare alcuni Jumper, questi sono gli stessi che è necessario impostare per utilizzare il modulo I2C. Queste impostazioni sono necessarie poiché i PIC con modulo USB hanno il modulo I2C in una posizione non standard rispetto agli altri PIC. In particolare i Jumper che è necessario impostare sono JP7 (SDA), JP8 (SCL) e JP11 (USB_CP). I Jumper JP7 (SDA) e JP8 (SCL) devono essere impostati come in Figura 3, in modo da avere rispettivamente i pin 2 e 3 connessi assieme. Questa impostazione riorganizza le linee SCL ed SDA, ovvero il bus I2C.

Figura 3: *Impostazioni Jumper JP7 e JP8 per PIC*

[8] Donazioni di supporto permettono la realizzazione di nuovi progetti e realizzazioni di articoli come quello che stato leggendo.

In aggiunta a questi Jumper bisogna anche impostare il Jumper JP11 (USB_CP), per mezzo del quale si va a collegare un condensatore al pin RC3. Questo condensatore serve per permettere al regolatore lineare interno al PIC, di operare in condizioni di stabilità; il regolatore interno permette di generare la tensione di riferimento di 3.3V. Questa tensione è utilizzata per i resistori di pull-up che identificano la modalità *full speed* e *low speed*. L'impostazione *full speed* e *low speed* deve essere fatta per mezzo dei resistori interni al PIC, infatti Freedom II non possiede i resistori esterni, altrimenti necessari. In Figura 4 è riportata l'impostazione del Jumper JP11 (USB_CP)[9].

Per ragioni legate al numero di applicazioni e impostazioni, si è preferito non supportare l'alimentazione della scheda Freedom II direttamente da USB. Infatti durante la fase sperimentale è facile commettere errori, i quali nel caso peggiore, potrebbero danneggiare il PC[10].

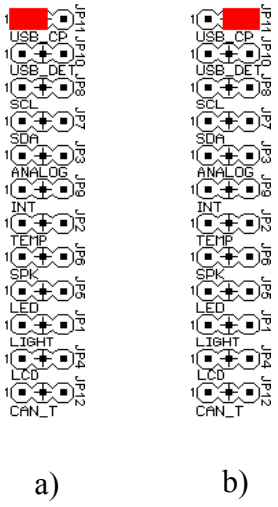

Figura 4: *Impostazioni Jumper JP11 a) condensatore connesso, b) condensatore disconnesso*

Pur non prelevando l'alimentazione dal PC, Freedom II supporta il rilevamento della connessione del cavo USB. Questo viene fatto per mezzo di semplici resistori, come riportato in Figura 5.

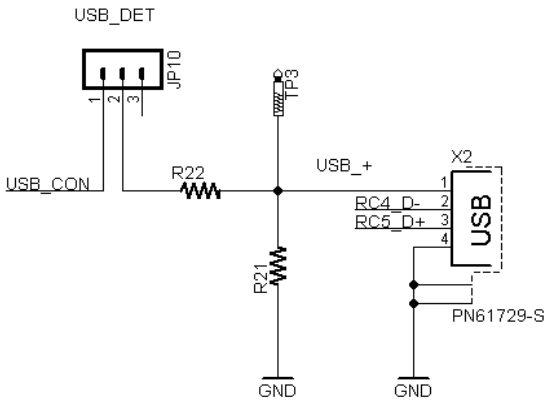

Figura 5: *Schema elettrico della sezione USB*

In particolare, tale configurazione è quella suggerita sul datasheet del PIC18F4550. Il resistore R21

[9] Si fa notare che per i PIC in cui non è presente il modulo USB il condensatore deve essere disconnesso. Se non dovesse essere disconnesso, il bus I2C potrebbe non lavorare propriamente a causa dell'eccessivo carico capacitivo.

[10] La scheda EasyUSB è invece ottimizzata per poter gestire tutte le diverse combinazioni che si possono avere tramite alimentazione esterna o via USB.

rappresenta il resistore di pull-down, ovvero fissa un livello logico 0 quando il cavo USB è sconnesso. R22 è invece un resistore di protezione, per mezzo del quale si limita la corrente in caso di cortocircuito. I valori suggeriti dalla Microchip, ovvero 100KΩ possono essere utilizzati qualora non si faccia uso dei resistori di pull-up interni alla PORTB, utilizzati dai pulsanti BT1-BT4.
Valori inferiori di resistori possono essere utilizzati mantenendo però a mente che, qualora si voglia far uso della modalità "*Suspended Mode*" come da specifica USB 2.0, la corrente del sistema deve essere mantenuta inferiore a 500µA.

Per impostare propriamente il rilevamento del cavo USB, se richiesto, è necessario impostare i Jumper JP10 (USB_DET) e JP9 (INT), come mostrato in Figura 6.

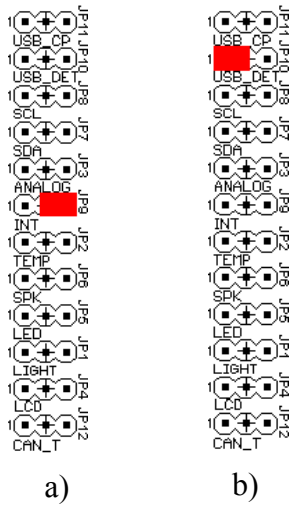

Figura 6: *Impostazioni Jumper JP9 a) e JP10 b) per attivare il rilevamento del cavo USB*

Si fa notare che il Jumper JP9 (INT) è condiviso con il segnale d'interrupt del Clock/Calendar. Nei progetti che seguiranno non si farà uso di questo segnale.

Oltre al modulo USB è necessario impostare la stringa LED, in modo da attivarla.

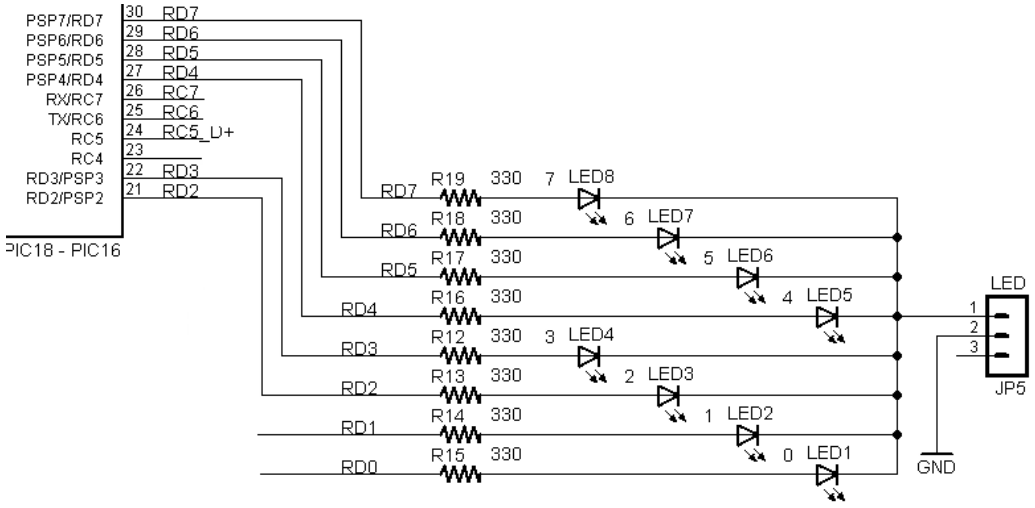

Figura 7: *Schema elettrico dell'interfaccia grafica a LED*

Come visibile dallo schema, la stringa LED può essere disabilitata per mezzo del Jumper JP5

(LED). La stringa LED risulta attiva se il Jumper è posizionato come in Figura 8 a) mentre risulta disattiva se posizionato come Figura 8 b).

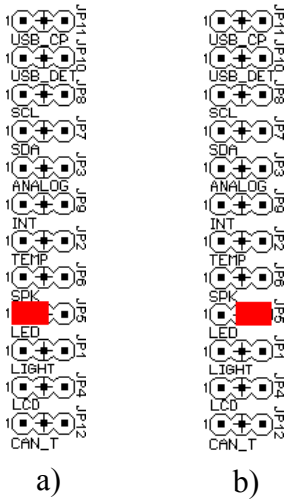

Figura 8: *Impostazioni Jumper JP5. a) LED ON, b) LED OFF*

In ultimo si tenga a mente che la retro-illuminazione del display LCD, è connessa al pin RC1, il quale è utilizzato per segnalare quando il modulo USB è stato propriamente riconosciuto dal'Host ovvero il PC. Il modulo LCD non verrà però utilizzato, infatti la retroilluminazione e il display sono tra loro indipendenti.

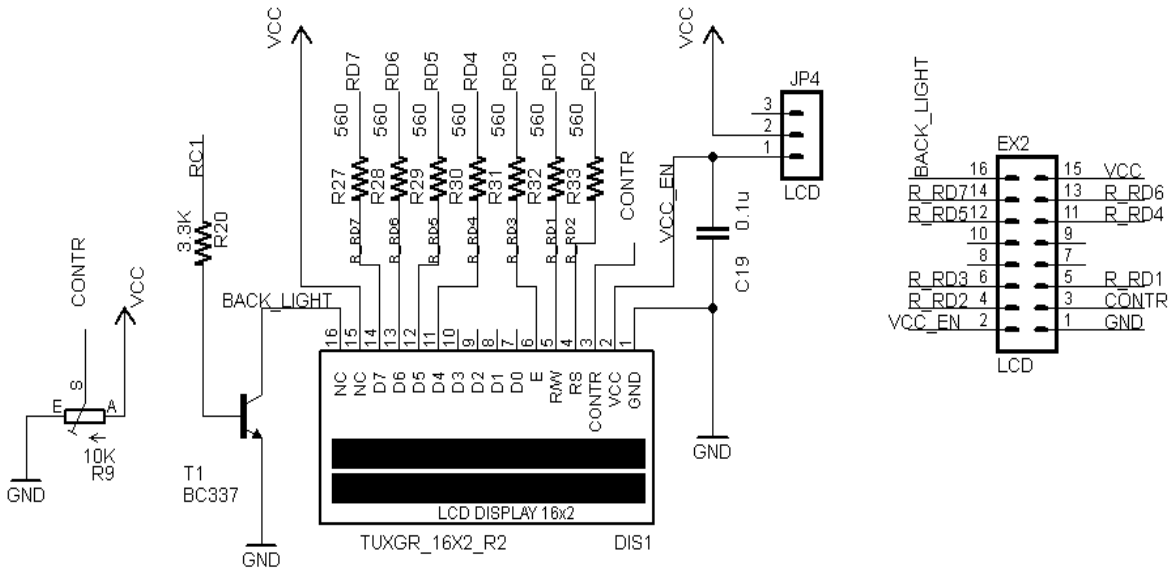

Figura 9: *Schema elettrico del display LCD*

Oltre agli 8 LED verranno utilizzati i 4 pulsanti presenti sulla scheda Freedom II (2 pulsanti nella scheda EasyUSB), lo stato dei pulsanti verrà infatti letto e trasmesso al PC. Lo schema elettrico relativo ai pulsanti è riportato in Figura 10.

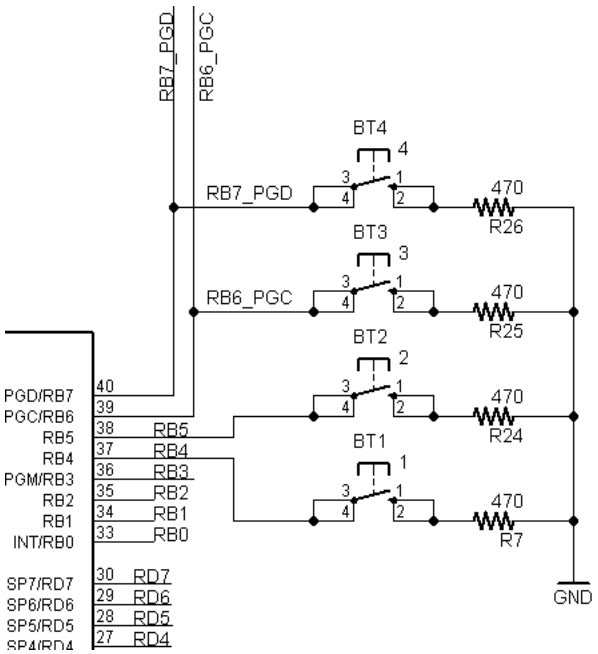

Figura 10: *Schema elettrico dei pulsanti*

Nota:

Altre schede di sviluppo possono essere utilizzate senza problemi. Visto il numero esiguo di componenti anche un semplice montaggio su mille fori o breadboard potrebbe essere utilizzato.

Impostare il PIC18F4550

Oltre alle impostazioni fisiche della scheda sono presenti delle impostazioni del PIC18F4550 che bisogna tenere a mente. Alcune di queste devono essere abilitate proprio in funzione del fatto che la scheda di sviluppo utilizzata è EasyUSB o Freedom II. Nel seguente paragrafo non si parlerà in dettaglio di ogni registro di configurazione, infatti quasi tutto è gestito dalla libreria Microchip. Ciononostante dal momento che la libreria richiede alcune impostazioni basate sul PIC utilizzato, la scheda di sviluppo e l'applicazione, è bene sapere alcune cose...Partiamo con l'analizzare la circuiteria di clock, interna al PIC18F4550 riportata in Figura 11.

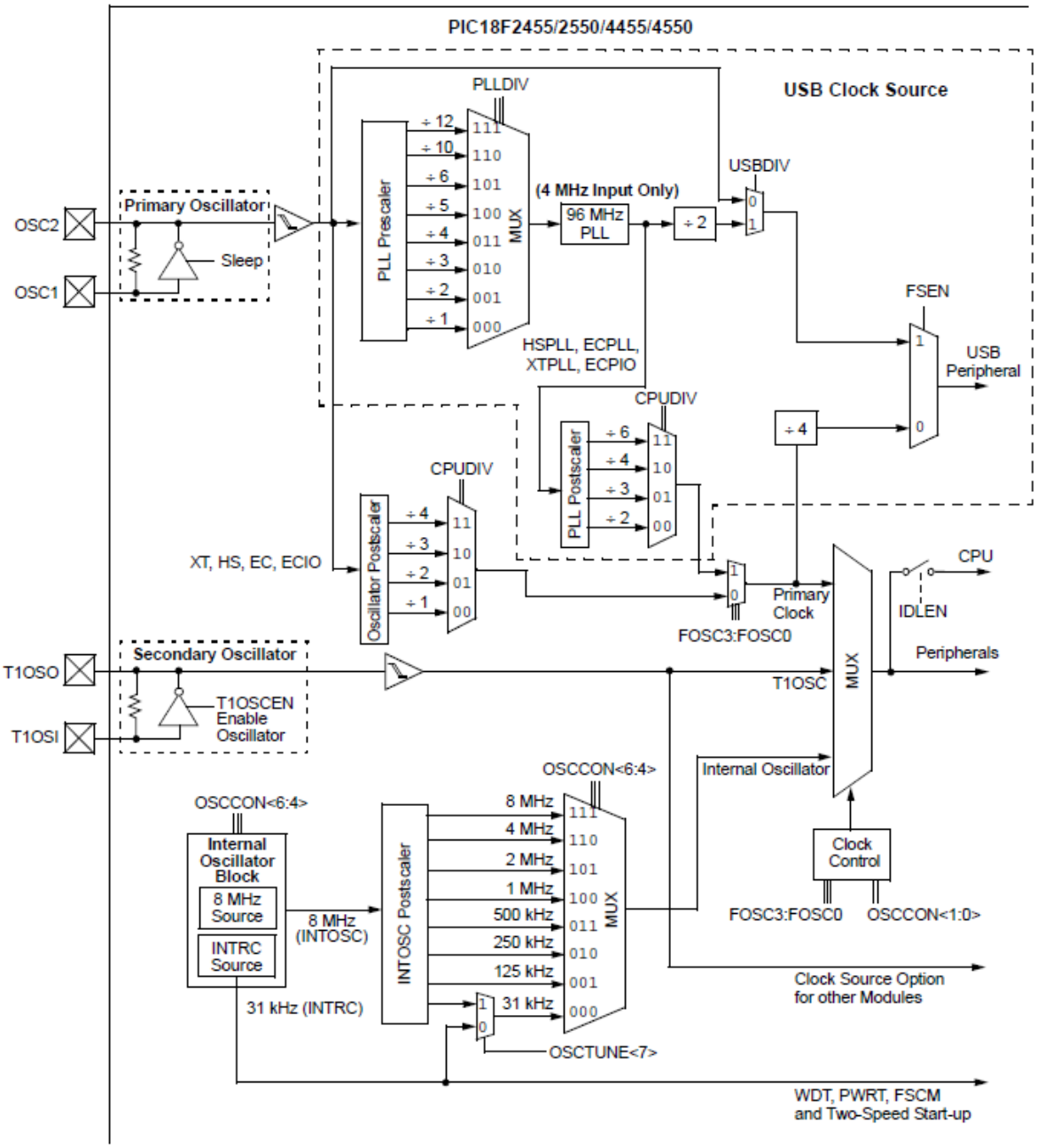

Figura 11: *Schema a blocchi della circuiteria di Clock*

Come tutti i PIC18, anche il PIC18F4550 possiede due Oscillatori esterni, ovvero il primario e il

secondario; in aggiunta possiede il clock interno ottenuto con un quarzo da 8MHz. Diversamente dagli altri PIC, l'oscillatore primario viene dedicato al modulo USB qualora questo venga abilitato. Ciononostante il clock delle periferiche può essere derivato dal clock in uscita dal PLL (Phase Lock Loop) come anche dall'oscillatore secondario o quello interno. Il PIC18F4550 supporta le specifiche USB 1.0 e USB 2.0 ed in particolare la modalità *low speed* e *full speed*. Qualora si voglia far uso della sola modalità *low speed* è necessario che il quarzo esterno sia di 6MHz. Se invece si vuol far uso della modalità *full speed* è possibile, per il quarzo esterno, utilizzare una più ampia scelta di valori. In particolare il valore del quarzo scelto deve essere tale per cui, diviso per uno dei valori selezionati per mezzo del mux PLLDIV, (1,2,3,4,5,6,10,12) il valore derivante sia di 4MHz. Questo è necessario poiché il PLL che segue il mux, ottiene 96MHz a partire da 4MHz di riferimento. Il clock di 96MHz diviso 2, ovvero 48MHz viene utilizzato dal modulo USB. Il clock da 96MHz, come detto, può essere utilizzato per generare il clock delle periferiche, questo viene però ottenuto per mezzo di una ulteriore divisione, selezionabile tra i valori 2,3,4,6 per mezzo del mux CPUDIV; questo significa che la CPU può avere un clock massimo di 48MHz. Utilizzando questa frequenza vi renderete conto che il PIC sarà un po' più caldo del normale...ma tutto entro le specifiche. Le impostazioni ora descritte devono essere effettuare per mezzo dei registri di configurazione ovvero facendo uso della direttiva `#pragma`. Un esempio di configurazione è :

```
#pragma config FOSC = HSPLL_HS
#pragma config PLLDIV = 5
#pragma config CPUDIV = OSC1_PLL2
```

In questa impostazione si è abilitato la modalità HS con PLL ovvero con modulo USB. Il registro PLLDIV è impostato a 5 poiché si è supposto che il quarzo utilizzato sia di 20MHz. Impostando la divisione a 5 si ha che il PLL ha al suo ingresso i 4MHz necessari per il suo corretto funzionamento.
Il clock delle periferiche è anche ottenuto dal clock primario, in particolare dal clock in uscita dal PLL, il cui valore è però diviso per 2. Si ricorda che le impostazioni che è possibile utilizzare per i vari parametri sono riportati nel file *hlpPIC18ConfigSet* presente nella directory *doc* della directory d'installazione del compilatore C18.

Oltre al clock, un'altra impostazione importante riguarda il regolatore lineare interno presente nel PIC18F4550; questo viene utilizzato per generare i 3.3V necessari per i resistori di pull-up che definiscono la modalità *low speed* e *full speed*[11]. In particolare i resistori, dovendo essere collegati tra una delle linee dati e 3.3V, possono essere esterni al PIC, come anche il regolatore di 3.3V. Dal momento che il PIC18F4550 possiede al suo interno sia il regolatore che i resistori perché non usarli? Sia EasyUSB che Freedom II sfruttano il regolatore lineare e i resistori interni, dunque è necessario abilitarli; il regolatore interno viene abilitato per mezzo della direttiva `#pragma`, scrivendo:

```
#pragma config VREGEN = ON
```

Le configurazioni ora descritte, ovvero facente uso della direttiva `#pragma` vanno propriamente impostate anche facendo uso della libreria che verrà introdotta a breve. I parametri che verranno di seguito descritti sono invece gestiti dalla libreria per mezzo di costanti che starà a noi impostare. Questo diverso metodo di gestione è legato al fatto che questi secondi parametri sono contenuti in registri che è possibile cambiare durante l'esecuzione del programma.

Il registro da impostare, appena lasciato in sospeso, è quello relativo ai resistori interni di pull-up. Questo viene fatto impostando ad 1 il bit UPUEN del registro UCFG, ovvero:

[11] Si ricorda che i resistori di pull-up utilizzati sono del valore 1.5KΩ ±5%, in particolare la modalità low speed è segnalata collegando il resistore di pull-up tra la linea D- e +3.3V, mentre la modalità full-speed viene indicata per mezzo di un resistore di pull-up collegato tra la linea D+ e 3.3V.

```
// Abilito resistori di pull-up per USB
UCFGbits.UPUEN = 0x01;
```

Come detto questa riga di codice non sarà necessaria poiché gestita dalla libreria. Altri registri, ovvero bit da impostare sono:

```
// Abilito il transceiver interno (normalmente già attivo)
UCFGbits.UTRDIS = 0x01;

// 1 Abilito modalità full speed, 0 modalità low speed
UCFGbits.FSEN = 0x01;

// Abilito modulo USB
UCONbits.USBEN = 0x01;
```

Anche in questo caso, come vedremo, sarà tutto gestito dalla libreria.

Impostare la libreria USB MCHPFSUSB

Arrivati a questo punto è bene sapere che gran parte delle operazioni necessarie per impostare ed usare la porta USB verranno gestite dalla libreria Microchip[12]. La libreria che verrà descritta è MCHPFSUSB versione 2.8[13] contenuta nel Package "*Microchip Application Libraries*" scaricabile gratuitamente dal sito della Microchip. Tale libreria dovrebbe essere chiamata più propriamente Framework visto che rappresenta un insieme di progetti facilmente modificabili dal progettista finale, ovvero voi[14]!

Procedendo nell'istallazione delle librerie è possibile vedere che vengono installate librerie di vario utilizzo, oltre a quella di nostro interesse relativa all'USB. Le varie versioni disponibili nel Package sono riportate in Figura 12.

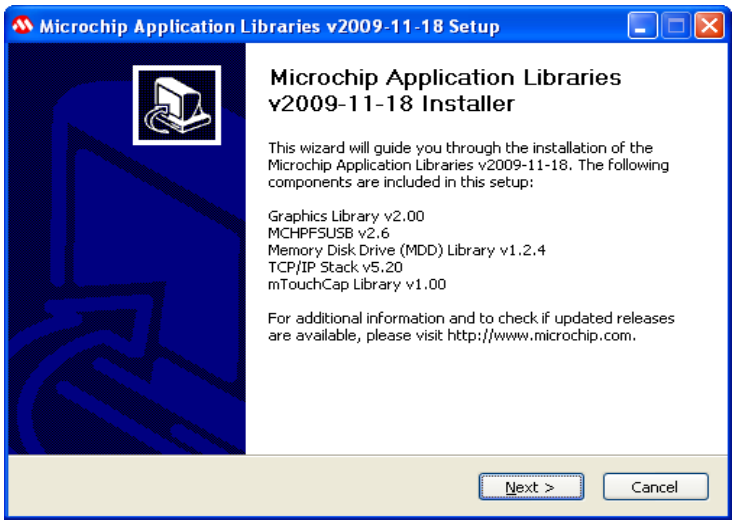

Figura 12: *Versioni delle librerie contenute nel Package d'installazione*

Il percorso d'installazione di default è "C:\Microchip Solutions [data versione]". All'interno di questa cartella sono presenti diverse cartelle contenenti numerosi progetti di esempio, come riportato in Figura 13. Il progetto di nostro interesse è contenuto nella cartella:

[percorso installazione]/USB Device - CDC - Serial Emulator/CDC - Serial Emulator/

All'interno di questa cartella sono presenti vari progetti realizzati in MPLAB utilizzabili con vari microcontrollori ed evaluation board, il progetto di nostro interesse è:

USB Device - CDC - Serial Emulator - C18 - PICDEM FSUSB

Tale progetto è stato scelto poiché compatibile con EasyUSB e con poche modifiche è possibile adattarlo alla scheda di sviluppo Freedom II. Prima di procedere oltre è bene leggere la documentazione disponibile con il Package scaricato in modo da avere una visione completa delle funzionalità e limiti della libreria, la documentazione può essere trovata nel percorso:

---

[12] Normalmente ogni produttore di dispositivi elettronici fornisce *Application Notes*, firmware e driver che semplificano molto la vita del progettista.

[13] E' bene far notare che dal sito della Microchip è possibile ancora scaricare la libreria MCHPFSUSB versione 1.2. Questa libreria viene scaricata in combinazione con la vecchia Application Note AN956. La libreria versione 1.2 è stata comunque testata su Freedom II, apportando modifiche molto simili a quelle descritte per la versione 2.6.

[14] Ogni volta che modificate un progetto del Framework è bene lavorare su una copia piuttosto che l'originale, permettendo dunque di usare l'originale sempre come master.

[percorso installazione]\Microchip\Help

Come visibile in Figura 14 la documentazione offerta è di facile lettura e navigabilità, il che vi permetterà di trovare facilmente le informazioni necessarie.

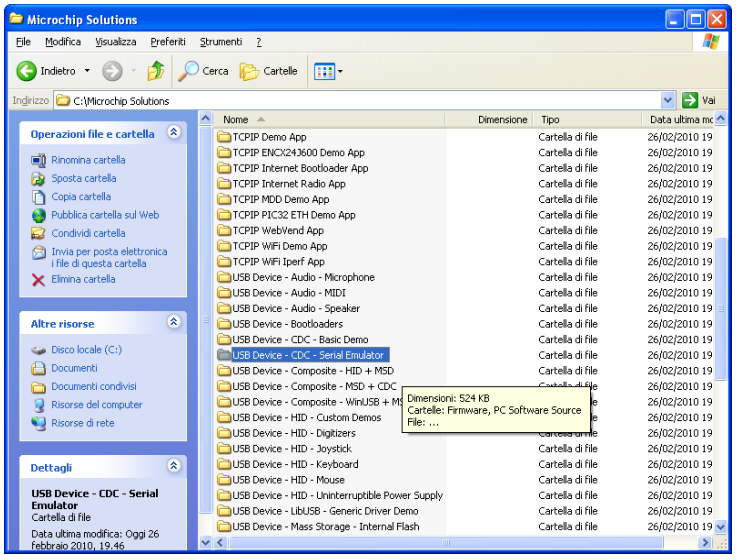

Figura 13: *Contenuto della cartella principale d'installazione del Package*

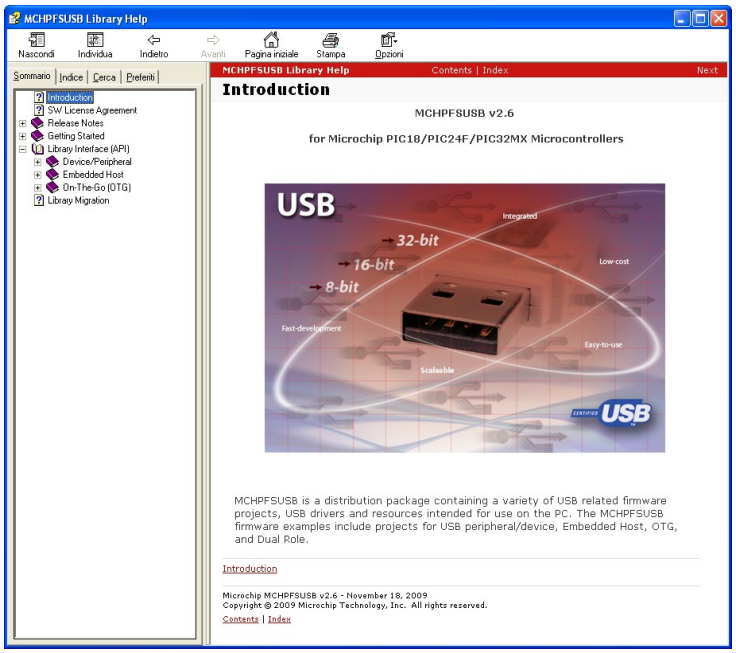

Figura 14: *Documentazione della Libreria*

Giunti a questo punto è possibile aprire il nostro progetto, giusto per spaventarci un po'! Il progetto è infatti piuttosto organico, visto che è concepito per supportare diverse impostazioni e microcontrollori. Ad occhio noterete che una buona parte del codice è in realtà composto da direttive o circondato da direttive.

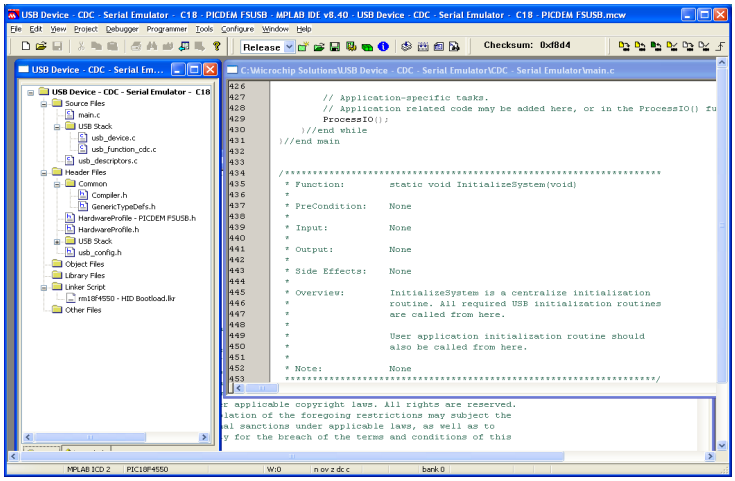

Figura 15: *Contenuto della cartella principale d'installazione del Package*

Questo significa che molto del codice presente non verrà utilizzato, o perché in realtà codice non è, o perché compreso tra direttive che lo escludono.

Il progetto selezionato rappresenta un progetto USB in cui viene implementata la classe CDC (Communication Device Class), in particolare il progetto permette di trattare la porta USB del nostro sistema EasyUSB o Freedom II, dal lato PC, come una normale porta seriale. Questo significa che potremo leggere e scrivere sulla porta USB come su di una semplice porta seriale. Infatti la classe CDC permette di richiamare il driver interno a Windows XP, usbser.sys per mezzo del quale Windows comunica attraverso il bus USB emulando la porta seriale[15].

Una volta che collegheremo il nostro sistema, propriamente programmato, Windows capirà che è richiesto un nuovo driver e richiederà di installarlo. L'installazione non sarà in realtà necessaria poiché come detto il driver è già presente, ciononostante sarà necessario installare il file .ini, presente nella cartella /inf del nostro progetto. L'installazione del file deve essere fatta solo una volta per mezzo della guida d'installazione del driver che si aprirà una volta collegata la scheda di sviluppo. Una volta installato il driver, Windows creerà una porta seriale COM virtuale, per mezzo della quale sarà possibile leggere e scrivere dati, utilizzando un semplice terminale come HyperTerminal o RS232 Terminal[16].

Dal lato PIC, leggere e scrivere dati sarà semplice come da lato PC, sono infatti disponibili funzioni di scrittura e lettura alle quali è possibile passare le nostre variabili da scrivere o leggere. I dettagli verranno visti nel primo esempio...

[15] Le impostazioni della porta seriale emulata vengono riflesse nelle impostazioni della UART interna al PIC. In questo modo è possibile realizzare un convertitore USB – RS232. Il convertitore è possibile realizzarlo direttamente per mezzo della scheda Freedom II mentre per EasyUSB è necessario aggiungere il transceiver RS232.

[16] RS232 Terminal è scaricabile gratuitamente dal sito www.Laurtec.it.

Funzioni di libreria

La Libreria USB, ovvero il Framework, mette a disposizione diverse funzioni per mezzo delle quali è possibile leggere e scrivere attraverso la porta USB come per una porta seriale. Queste possono essere trovate nel file usb_function_cdc.c[17] , mentre la loro dichiarazione è nel file usb_function_cdc.h. Dal momento che in realtà, alcune delle funzioni che verranno a breve descritte, sono delle macro, queste sono dichiarate solo nel file usb_function_cdc.h. Nella versione della libreria 1.2 le macro erano precedute da una *m* minuscola, per esempio *m*USBUSARTIsTxTrfReady ( ). La m è stata eliminata nella versione 2.8, ma per mantenere una compatibilità con la vecchia libreria è in realtà presente anche la macro con il nome che inizia con *m* minuscola, tali macro non sono però riportate tra la lista delle macro disponibili. Le funzioni e macro che la libreria mette a disposizione sono:

- **`USBUSARTIsTxTrfReady()`**

  Questa funzione è in realtà una macro, e permette di controllare se il modulo USB è pronto per poter inviare dei dati. Questa macro deve essere richiamata prima di effettuare una qualunque trasmissione dati. La macro ritorna 1 se il modulo USB può trasmettere dati, 0 altrimenti. Questa funzione non deve essere richiamata/controllata per mezzo dell'istruzione while, visto che si creerebbe una istruzione bloccante che impedirebbe al resto della libreria di funzionare correttamente, dunque di aggiornare lo stato delle variabili (si vedano gli esempi per maggiori dettagli).

  Tipo

  Macro

  Variabile in ingresso

  void

  Ritorno

  1: Il modulo USB può trasmettere dati
  0: Il modulo USB non può momentaneamente trasmettere dati.

- **`void putUSBUSART(char *data, BYTE  length)`**

  Questa funzione permette di scrivere dati dalla RAM al modulo USB. Il puntatore `data` rappresenta il puntatore alla variabile o Array in cui è contenuta l'informazione da trasmettere. Tale funzione deve essere utilizzata quando è nota la lunghezza dell'Array, infatti il secondo parametro da passare rappresenta proprio la lunghezza dell'Array. Qualora si passasse una variabile semplice di tipo `char`, la lunghezza è 1. Si noti che il tipo della variabile `length` è di tipo BYTE, definita come:

```
typedef unsigned char           BYTE;
```

[17] Nella versione della libreria 1.2 associata all'Application Note AN956, tali funzioni si trovano nel file cdc.c.

Tipo

Funzione

Variabile in ingresso

***data:** Puntatore alla struttura dati residente in memoria RAM
length: lunghezza della struttura dati (numero di byte).

Ritorno

void

- **`BYTE getsUSBUSART(char *buffer, BYTE len)`**

Questa funzione, viene utilizzata per leggere i dati dal modulo USB. La funzione accetta come parametri il puntatore alla struttura dati della variabile, o Array in cui scrivere i dati, e la lunghezza, ovvero il numero di byte che ci si aspetta siano presenti. Nel caso in cui siano presenti più byte verranno letti solo quelli richiesti (i rimanenti possono essere letti in letture successive), mentre qualora non siano presenti i byte richiesti, vengono ritornati solo quelli presenti. La funzione non è bloccante, ovvero non aspetta per l'arrivo di tutti i byte. Ritorna 0 se non sono presenti byte da leggere.

Tipo

Funzione

Variabile in ingresso

***buffer:** Puntatore alla struttura dati, residente in memoria RAM, dove scrivere i dati letti.
len: Numero di byte da leggere.

Ritorno

Numero di byte letti. Vale 0 se non sono presenti dati da leggere.

- **`void putrsUSBUSART(const rom char *data)`**

Questa funzione permette di scrivere una stringa terminata dal carattere '\0' memorizzata in memoria flash (tipo rom). Prima di chiamare tale funzione è necessario accertarsi che il modulo USB possa accettare richieste di trasmissione dati. La funzione richiede solo il puntatore alla struttura che contiene i dati da trasmettere, ovvero l'indirizzo della memoria flash dove è contenuta la stringa d'interesse. La lunghezza non è richiesta poiché la stringa è terminata dal carattere '\0'.

Tipo

Funzione

Variabile in ingresso

***data:** Puntatore alla struttura dati, residente in memoria flash.

Ritorno

void

- **`void putsUSBUSART(char *data)`**

  Questa funzione si comporta come la funzione precedente tranne che per il fatto che il puntatore è di tipo normale, ovvero ad una struttura dati presenti in RAM. Anche in questo caso la stringa deve essere terminata con il carattere '\0'.

  Tipo

  Funzione

  Variabile in ingresso

  ***data:** Puntatore alla struttura dati, residente in memoria RAM.

  Ritorno

  void

Oltre alle funzioni e macro descritte ne sono presenti altre che è possibile richiamare, ciononostante per gli scopi di questo articolo hanno meno interesse[18]. Si rimanda alla documentazione ufficiale della Libreria, ed in particolare all'header file usb_function_cdc.h, in cui è possibile trovare maggiori dettagli sulle modalità di funzionamento delle funzioni appena introdotte. Si consiglia inoltre di controllare anche i file sorgenti, in maniera tale da comprendere in maggior dettaglio il loro funzionamento.

Poiché ogni volta che si debbano scrivere dati, bisogna richiamare la funzione **`mUSBUSARTIsTxTrfReady()`** è importante che non si effettuino scritture in successione di dati, senza controllare preventivamente la possibilità effettiva di poterli trasmettere.

Oltre alle funzioni di libreria può essere utile il controllo della variabile di sistema offerta dalla libreria USB, ovvero la variabile `USBDeviceState`. Tale variabile viene aggiornata in funzione dello stato di configurazione del sistema.

Lo stato DETACHED_STATE e ATTACHED_STATE non devono essere utilizzati su Freedom II poiché Freedom II non supporta l'alimentazione da USB (tale stati sono supportati da EasyUSB). In particolare queste due costanti indicano quando il cavo USB è collegato e scollegato. Per quanto riguarda gli altri valori disponibili, il significato è il seguente:

- **`POWERED_STATE`**
  Il cavo USB è collegato ma il sistema non è stato configurato dall'Host.

[18] Alcune funzioni e macro non sono riportate nella documentazione, poiché ne viene scoraggiato il loro diretto uso.

- **`DEFAULT_STATE`**
  Il sistema è in stato di Default per permettere la comunicazione da parte dell'Host.

- **`ADDRESS_STATE`**
  L'Host ha iniziato la comunicazione con il sistema per poterlo configurare e al tempo stesso configurare il sistema operativo. Questa è anche la fase in cui Windows® avvia la sequenza di caricamento del driver.

- **`CONFIGURED_STATE`**
  Il sistema è propriamente configurato e il sistema operativo ha caricato il driver opportuno.

Per poter leggere la variabile `USBDeviceState` si procede allo stesso modo con cui si farebbe con una variabile normale, dunque si può fare un controllo per mezzo dell'istruzione `if`. E' bene ricordare che non bisogna utilizzare l'istruzione `while` al fine di attendere un cambio di stato. Questa procedura sarebbe infatti di tipo bloccante, dunque la variabile non verrebbe mai aggiornata dal servizio di aggiornamento della libreria, causando dunque lo stallo del modulo USB. Maggiori dettagli a riguardo sono descritti negli esempi che seguono.

Utilizzo della Libreria

Come visto la Libreria è integrata all'interno di un progetto, da cui il nome più appropriato è quello di Framework. In particolare una volta installata la libreria sono presenti i vari progetti di esempio che è possibile modificare. Nel nostro caso si farà uso, come detto, del progetto CDC, ovvero del progetto in cui, per mezzo della emulazione di una porta seriale, è possibile facilmente utilizzare la porta USB. Al fine di poter utilizzare il progetto CDC con il sistema Freedom II e EasyUSB, è necessario apportare piccole modifiche al progetto stesso, in maniera da rendere il progetto compilato compatibile con le schede di sviluppo (come si vedrà per la scheda EasyUSB non sono praticamente richieste modifiche, è però raccomandabile controllare che le impostazioni siano compatibili con la vostra applicazione). Per poter adeguare e controllare il progetto CDC, aprirlo utilizzando l'ambiente di sviluppo MPLAB ed apportare le seguenti modifiche:

- **Modifica 1**

Disabilitare il remapping dei vettori d'interruzione e reset, disabilitando l'utilizzo del boot-loader. Questo deve essere fatto commentando la riga di codice, come sotto riportato, che è possibile trovare nel file Hardware Profile – PICDEM FSUSB.h:

```
//Uncomment the following line to make the output HEX of this
//project work with the HID Bootloader
//#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
```

Questa modifica non deve essere effettuata per il sistema EasyUSB, qualora si voglia utilizzare il Bootloader interno.

- **Modifica 2**

Controllare che le seguenti righe di codice, presenti nel file Hardware Profile – PICDEM FSUSB.h siano commentate:

```
//The PICDEM FS USB Demo Board platform supports the USE_SELF_POWER_SENSE_IO
//and USE_USB_BUS_SENSE_IO features.  Uncomment the below line(s) if
//it is desireable to use one or both of the features.
//#define USE_SELF_POWER_SENSE_IO
```

Questo permette di disabilitare il rilevamento del cavo e attivare l'alimentazione da USB, non supportata da Freedom II per ragioni di sicurezza.

Accertarsi inoltre che la seguente riga di codice, presente nello stesso file, sia commentata:

```
//#define USE_USB_BUS_SENSE_IO
```

Queste righe possono non essere commentate qualora si stia usando la scheda EasyUSB; infatti la scheda EasyUSB supporta il power sense.

- **Modifica 3**

Nel file main.c, all'interno della funzione main disabilitare la chiamata alla funzione ProcessIO, come sotto riportato:

```
// Application-specific tasks.
```

```
// Application related code may be added here, or in the ProcessIO() function

//ProcessIO();
```

Non richiamando tale funzione si disabilita la chiamata alle funzioni di esempio presenti all'interno del progetto.

- **Modifica 4**

Come spiegato, la circuiteria di clock deve essere propriamente impostata al fine di avere il clock giusto per operare. La libreria è già impostata per lavorare con un quarzo 20MHz ed avere un clock per le periferiche direttamente dal clock del PLL. Le impostazioni presenti nel file main.c sono:

```
/** CONFIGURATION **************************************************/

// Configuration bits for PICDEM FS USB Demo Board (based on PIC18F4550)
#if defined(PICDEM_FS_USB)

        #pragma config PLLDIV   = 5         // 20 MHz crystal
        #pragma config CPUDIV   = OSC1_PLL2
        #pragma config USBDIV   = 2         // Clock source from 96MHz PLL/2
        #pragma config FOSC     = HSPLL_HS
```

Oltre a tali impostazioni sono anche presenti altre relative ad altri PIC e configurazioni disponibili. Qualora si avessero esigenze particolari è possibile cambiare i valori sopra riportati ed adeguarli al proprio progetto.

- **Modifica 5**

Oltre alla circuiteria di clock è necessario abilitare il regolatore interno al PIC18F4550, in maniera da permettere il corretto funzionamento del modulo USB. La libreria attuale possiede il regolatore già abilitato, dunque non è necessario fare alcuna modifica, ma accertarsi di avere la seguente riga di codice, sempre nel file main.c, su ON:

```
#pragma config VREGEN   = ON      //USB Voltage Regulator
```

- **Modifica 6**

In ultimo, ci sono le impostazioni relative all'abilitazione dei resistori interni, utilizzati per selezionare la modalità di trasmissione *low speed* o *full speed* e l'utilizzo del transceiver interno. Queste impostazioni è possibile effettuarle nel file usb_config.h.

```
/* Parameter definitions are defined in usb_device.h */
#define USB_PULLUP_OPTION USB_PULLUP_ENABLE
//#define USB_PULLUP_OPTION USB_PULLUP_DISABLED

#define USB_TRANSCEIVER_OPTION USB_INTERNAL_TRANSCEIVER
//External Transceiver support is not available on all product families.
// Please refer to the product family datasheet for more information if this
// feature is available on the target processor.
//#define USB_TRANSCEIVER_OPTION USB_EXTERNAL_TRANSCEIVER

#define USB_SPEED_OPTION USB_FULL_SPEED
//#define USB_SPEED_OPTION USB_LOW_SPEED //not valid option for PIC24F devices
```

Le impostazioni attuali sono già compatibili con Freedom II ed EasyUSB, ma è bene accertarsi di avere le impostazioni come sopra. Negli ultimi anni la libreria ha infatti avuto molte modifiche[19].

A questo punto, con le poche modifiche descritte, la libreria è già pronta per operare con la nostra scheda di sviluppo Freedom II e EasyUSB. Naturalmente oltre alle modifiche ora apportate, il progetto può essere ripulito da tutto quel codice che non deve essere utilizzato. In particolare sarà possibile eliminare gli esempi presenti e tutte le impostazioni utilizzate per le schede di sviluppo e PIC supportati. La pulizia della libreria sarà cosa facile una volta presa esperienza con la stessa, in ogni modo si consiglia di compilare il progetto ad ogni modifica apportata e testarlo nuovamente sulla scheda di sviluppo. In questo modo eviterete di eliminare parti di codice che vi sono sembrate "di troppo". Personalmente non ho l'abitudine di ripulire il codice in maniera tale da poter riportare facilmente ogni programma nella versioni più aggiornate della libreria senza dover ripulire ogni volta il codice.

[19] Sono presenti ancora tracce di vecchie variabili e nomi di file che non esistono più nella nuova libreria almeno fino alla versione 2.6.

...ci siamo quasi

Prima di iniziare cerchiamo di capire come utilizzare il tutto. La libreria, negli ultimi anni ha subito molti cambiamenti. Molti di questi cambiamenti è possibile vederli confrontando la libreria presente descritta nell'Application Note AN956, ovvero la versione 1.2 e la versione 2.8. Un importante miglioramento che si ha nella libreria 2.8 è la gestione del modulo USB per mezzo delle interruzioni. Nelle prime versioni della libreria la gestione era infatti effettuata solo in polling, rendendo l'utilizzo della libreria meno snella. La versione 2.8 può ancora gestire il modulo USB per mezzo del polling, ovvero controllo continuo dello stato del modulo USB. Per attivare questa funzione è necessario togliere il commento dalla riga di codice presente nel file usb_config.h

```
//#define USB_POLLING
```

e commentare l'altra riga attiva di default.

```
#define USB_INTERRUPT
```

Bene, cerchiamo ora di riassumere il tutto...

Il progetto che abbiamo sotto le mani è già funzionate e possiede già la funzione main all'interno del file main.c. Questo significa che il progetto può essere compilato con successo senza scrivere il nostro codice. Volendo ovviamente utilizzare la libreria USB all'interno di un nostro progetto, quello che bisogna fare è semplicemente scrivere il nostro programma all'interno della funzione main, richiamando le funzioni della libreria USB per inviare e scrivere dati nella porta USB. Il PC per ricevere e scrivere dati via USB dovrà utilizzare un semplice programma che gestisca la porta seriale RS232 creata per emulazione dal driver caricato alla connessione del nostro hardware (device).

La funzione `main` è così definita:

```
/********************************************************************
 * Function:        void main(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Main program entry point.
 *
 * Note:            None
 *******************************************************************/
#if defined(__18CXX)
void main(void)
#else
int main(void)
#endif
{
    InitializeSystem();

    #if defined(USB_INTERRUPT)
        USBDeviceAttach();
    #endif
```

```
    while(1)
    {
        #if defined(USB_POLLING)
        // Check bus status and service USB interrupts.
        USBDeviceTasks();
                  // Interrupt or polling method.  If using polling, must call
                  // this function periodically.  This function will take care
                  // of processing and responding to SETUP transactions
                  // (such as during the enumeration process when you first
                  // plug in).  USB hosts require that USB devices should accept
                  // and process SETUP packets in a timely fashion.  Therefore,
                  // when using polling, this function should be called
                  // frequently (such as once about every 100 microseconds) at any
                  // time that a SETUP packet might reasonably be expected to
                  // be sent by the host to your device.  In most cases, the
                  // USBDeviceTasks() function does not take very long to
                  // execute (~50 instruction cycles) before it returns.
        #endif


  // Application-specific tasks.
  // Application related code may be added here, or in the ProcessIO() function.
  //ProcessIO();
    }//end while
}//end main
```

All'inizio della funzione `main` è presente la funzione d'inizializzazione del modulo USB, che bisognerà lasciare in ogni programma, e un controllo per verificare se l'hardware è attaccato.

Dopo l'inizializzazione del modulo USB bisognerà aggiungere le nostre impostazioni relative alle porte del PIC, ovvero impostare i registri TRIS delle porte del PIC. Questo è necessario poiché altrimenti verrebbero utilizzate le impostazioni relative alla scheda di sviluppo Microchip. Dopo le impostazioni è possibile scrivere il nostro programma, il quale, se deve essere ripetuto può anche essere scritto all'interno del ciclo `while` che segue l'inizializzazione. All'interno del ciclo `while` è presente la chiamata alla funzione ProcessIO, che abbiamo commentato e la chiamata alla funzione:

```
USBDeviceTasks();
```

Questa non verrà richiamata visto che abbiamo abilitato la modalità interrupt. Questa chiamata di funzione viene infatti eseguita solo se si controlla il modulo USB per mezzo del polling, ovvero controllo continuo del modulo USB per verificare eventuali cambi di stato.

Esempio I

In questo primo esempio si mostra semplicemente lo stato del sistema attraverso dei LED che lampeggiano. Inoltre viene mostrata la procedura d'installazione che è richiesta quando il nostro sistema viene collegato per la prima all'Host, ovvero al PC[20]. Il programma di esempio consiste nella funzione main che dovrà sostituire quella originale nel progetto Microchip. Il resto del progetto deve rimanere invariato. Il programma può essere usato sia per Freedom II che per EasyUSB.

```
/********************************************************************
 * Function:        void main(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Main program entry point.
 *
 * Note:            None
 *******************************************************************/

void main(void) {

  // Inizializzazione base del Modulo USB
  InitializeSystem();

  // Inizializzazione del modulo USB per utilizzo interruzioni
  USBDeviceAttach();

  // Imposto PORTA tutti ingressi
  LATA = 0x00;
  TRISA = 0xFF;

  // Imposto PORTB tutti ingressi
  LATB = 0x00;
  TRISB = 0xFF;

  // Imposto PORTC tutti ingressi
  LATC = 0x00;
  TRISC = 0xFF;

  // Imposto PORTD tutte uscite per uso LED
  LATD = 0x00;
  TRISD = 0x00;

  // Imposto PORTE tutti ingressi
  LATE = 0x00;
  TRISE = 0xFF;


   while(1) {

  // Richiamo il controllo dei LED.
  BlinkUSBStatus ();
```

[20] Una volta che il sistema viene installato, cambiando la porta USB verrà nuovamente richiesta l'installazione del driver, questa volta però, l'installazione automatica troverà il driver opportuno grazie al file .inf già installato.

```
    }
}
```

Si noti che le parti della funzione main che non sono utilizzate, sono state rimosse, in particolare, visto che si sta utilizzando il PIC18F4550 si è eliminata la scelta del tipo di funzione main. La seconda modifica è relativa alla eliminazione della chiamata alla funzione:

```
USBDeviceTasks();
```

Infatti questa deve essere richiamata solo se si sta utilizzando la modalità Polling, mentre noi abbiamo abilitato la modalità Interrupt. Inoltre si è variato il registro di configurazione, in particolare il valore CPUDIV, da:

```
#pragma config CPUDIV   = OSC1_PLL2
```

a:

```
#pragma config CPUDIV   = OSC4_PLL6
```

Questo è richiesto in modo da avere un clock per le periferiche 6 volte inferiore al clock in uscita dal PLL ottenendo un lampeggio più lento dei LED[21].

Oltre a queste piccole modifiche si noti che ogni singola porta è stata impostata manualmente, permettendo di sovrascrivere le impostazioni effettuate dalla funzione d'inizializzazione. Il lampeggio dei LED avviene grazie alla funzione:

```
BlinkUSBStatus ();
```

richiamata in maniera ciclica all'interno del ciclo `while` infinito. Tale funzione viene fornita all'interno della libreria USB stessa, o meglio nel progetto di esempio. In particolare controlla i LED collegati sulla PORTD, ovvero RD0 e RD1. Dal momento che sia EasyUSB che Freedom II possiedono i LED su questi stessi pin, non è necessario apportare alcuna modifica alla funzione stessa.

Compilando e caricando il programma all'interno del PIC, si avrà che inizialmente, resettando il sistema e avendo il cavo USB scollegato, solo il LED1 è accesso fisso, ovvero siamo nello stato POWERED_STATE. Collegando il cavo USB il LED1 inizierà a lampeggiare e Windows avvierà la procedura d'installazione del driver come in Figura 16, questo è lo stato ADDRESS_STATE. Si selezioni l'opzione “Non ora” e si vada avanti.

[21] Per rallentare il conteggio si sarebbe potuto anche chiamare la funzione BlinkUSBStatus meno frequentemente o modificare i tempi di lampeggio della funzione stessa. La modifica del registro di configurazione non sarebbe stata necessaria in caso di utilizzo della modalità polling, visto che l'esecuzione continua della funzione USBDeviceTasks rallenta a sufficienza la CPU, dunque le chiamate a BlinkUSBStatus.

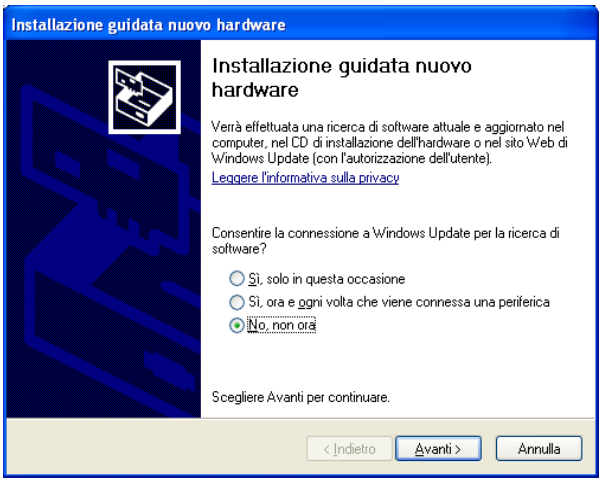

Figura 16: *Avvio della fase d'installazione del driver.*

Successivamente selezionare l'opzione come riportato in Figura 17.

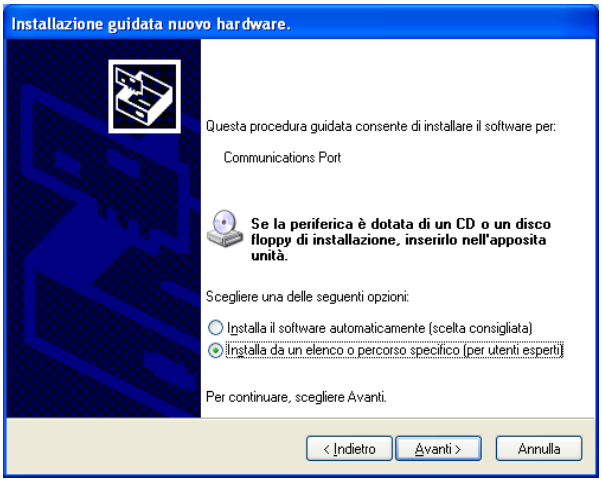

Figura 17: *Selezione manuale del driver.*

Questo permetterà di selezionare il percorso dove è presente il nostro file .inf, ovvero il file .inf disponibile all'interno della cartella del progetto CDC.

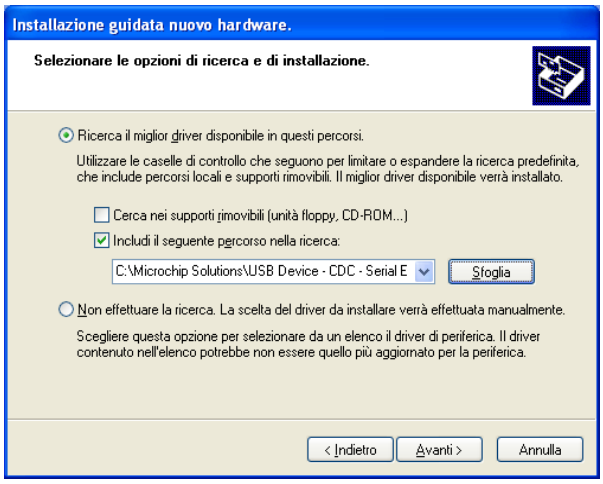

Figura 18: *Selezione manuale del precorso.*

Se tutto è stato propriamente impostato, il sistema operativo troverà il file .inf al cui interno sono presenti le informazioni che identificano il nostro progetto, ed avvierà l'installazione del driver opportuno, come riportato in Figura 19.

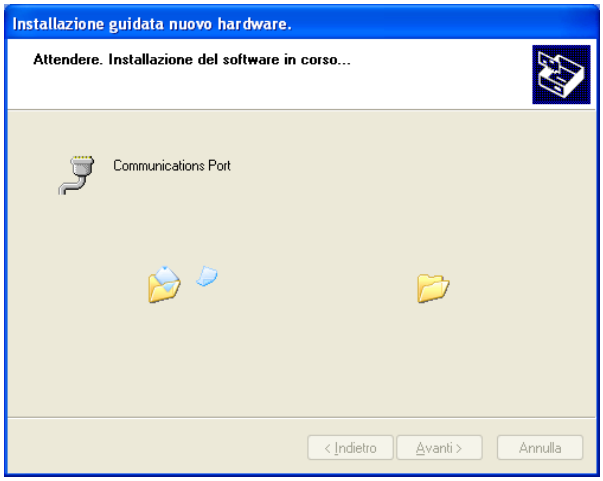

Figura 19: *Installazione del driver.*

A fine installazione, quando comparirà la finestra di Figura 20, il LED1 e LED2 inizieranno a lampeggiare alternativamente. Questo è lo stato CONFIGURED_STATE, ovvero la porta seriale emulata è stata creata e siamo pronti a comunicare con il nostro dispositivo.

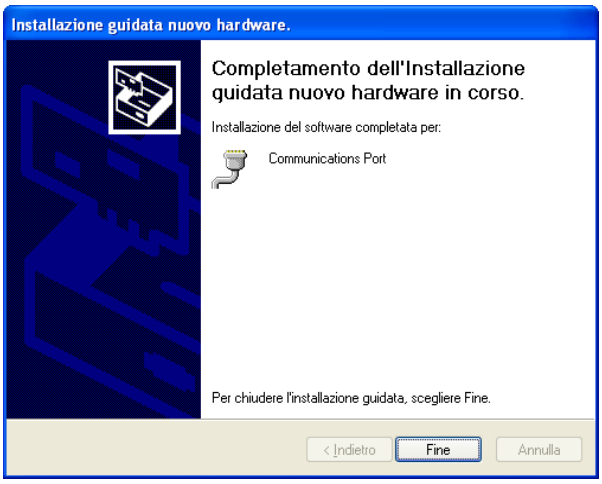

Figura 20: *Finestra di conferma di avvenuta installazione.*

Andando al pannello di controllo e visualizzando le periferiche di sistema, è possibile vedere, come mostrato in Figura 21, che nella lista delle porte COM è presente la nostra *Communications Port* appena installata.

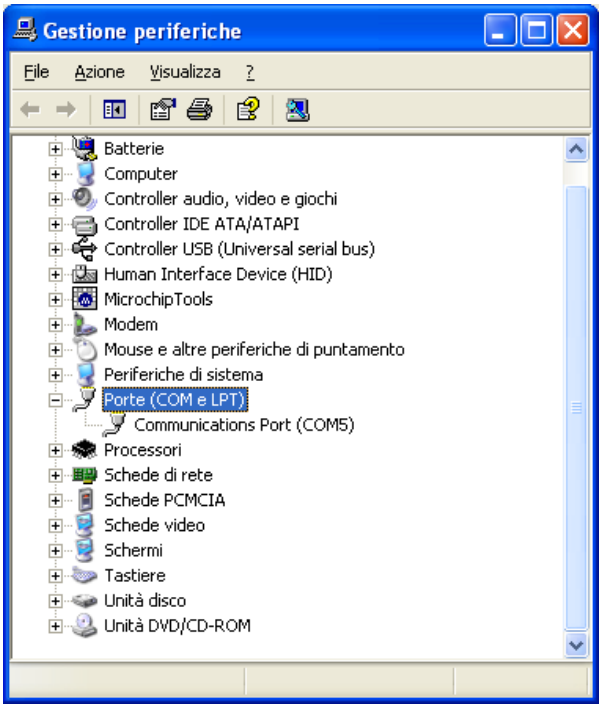

Figura 21: *Visualizzazione della lista delle periferiche di sistema.*

Esempio II

In questo secondo esempio viene mostrato come leggere e scrivere dati tramite USB. In particolare la trasmissione e ricezione dati viene fatta per mezzo di HyperTerminal propriamente impostato e abilitato a lavorare sulla porta COM aperta con l'emulazione. Questo significa che HyperTerminal deve essere avviato solo dopo aver compilato, caricato il programma e connessa la scheda alla porta USB. Il programma di esempio permette di scrivere su HyperTerminal il tasto che viene premuto BUTTON: 1, BUTTON: 2, BUTTON: 3, BUTTON: 4 rispettivamente (EasyUSB possiede solo due pulsanti). Scrivendo invece sul terminale, o meglio premendo un numero da 1 a 8 si accenderà il LED1...LED8 rispettivamente (EasyUSB possiede solo i primi 4 LED). Vediamo come fare:

```
/********************************************************************
 * Function:        void main(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Main program entry point.
 *
 * Note:            None
 ********************************************************************/

void main(void) {

  // Variabile utilizzata alla lettura del Buffer in Ingresso
  char dataIN = 0x00;

  // Variabile da trasmettere contenente il testo
  char dataOUT[11];

  // Inizializzazione Testo
  dataOUT[0] = 'B';
  dataOUT[1] = 'U';
  dataOUT[2] = 'T';
  dataOUT[3] = 'T';
  dataOUT[4] = 'O';
  dataOUT[5] = 'N';
  dataOUT[6] = ':';
  dataOUT[7] = ' ';
  dataOUT[9] = 13;
  dataOUT[10] = '\0';

  // Inizializzazione base del Modulo USB
  InitializeSystem();

  // Inizializzazione del modulo USB per utilizzo interruzioni
  USBDeviceAttach();


  // Imposto PORTA tutti ingressi
  LATA = 0x00;
  TRISA = 0xFF;

  // Imposto PORTB tutti ingressi
```

```
LATB = 0x00;
TRISB = 0xFF;

// Imposto PORTB tutti ingressi
LATC = 0x00;
TRISC = 0xFF;

// Imposto PORTD tutte uscite
LATD = 0x00;
TRISD = 0x00;

// Imposto PORTE tutti ingressi
LATE = 0x00;
TRISE = 0xFF;

// Abilita i resistori di pull-up sulla PORTB
EnablePullups();


while (1) {

    CDCTxService();

    // Controllo pressione BT1
    if (PORTBbits.RB4 == 0 )
          if(USBUSARTIsTxTrfReady()) {
                dataOUT[8] = '1';
                putsUSBUSART(dataOUT);
          }

    // Controllo pressione BT2
    if (PORTBbits.RB5 == 0 )
          if(USBUSARTIsTxTrfReady()) {
                dataOUT[8] = '2';
                putsUSBUSART(dataOUT);
          }

    // Controllo pressione BT3
    if (PORTBbits.RB6 == 0 )
          if(USBUSARTIsTxTrfReady()) {
                dataOUT[8] = '3';
                putsUSBUSART(dataOUT);
          }

    // Controllo pressione BT4
    if (PORTBbits.RB7 == 0 )
          if(USBUSARTIsTxTrfReady()) {
                dataOUT[8] = '4';
                putsUSBUSART(dataOUT);
          }


    dataIN = 0;

    // Leggo un byte dal buffer in ingresso
    getsUSBUSART (&dataIN, 1);

    // Controllo se è stato premuto 1 sulla tastiera
    if (dataIN == 0x31)
          LATD = 0x01;

    // Controllo se è stato premuto 2 sulla tastiera
```

```
      if (dataIN == 0x32)
            LATD = 0x02;

      // Controllo se è stato premuto 3 sulla tastiera
      if (dataIN == 0x33)
            LATD = 0x04;

      // Controllo se è stato premuto 4 sulla tastiera
      if (dataIN == 0x34)
            LATD = 0x08;

      // Controllo se è stato premuto 5 sulla tastiera
      if (dataIN == 0x35)
            LATD = 0x10;

      // Controllo se è stato premuto 6 sulla tastiera
      if (dataIN == 0x36)
            LATD = 0x20;

      // Controllo se è stato premuto 7 sulla tastiera
      if (dataIN == 0x37)
            LATD = 0x40;

      // Controllo se è stato premuto 8 sulla tastiera
      if (dataIN == 0x38)
            LATD = 0x80;

    }
}
```

Il programma inizia con l'inizializzare le variabili e il buffer che verranno utilizzati nel programma stesso, rispettivamente per leggere e scrivere dati dalla porta USB.

```
// Variabile utilizzata alla lettura del Buffer in Ingresso
char dataIN = 0x00;

// Variabile da trasmettere contenente il testo
char dataOUT[11];

// Inizializzazione Testo
dataOUT[0] = 'B';
dataOUT[1] = 'U';
dataOUT[2] = 'T';
dataOUT[3] = 'T';
dataOUT[4] = 'O';
dataOUT[5] = 'N';
dataOUT[6] = ':';
dataOUT[7] = ' ';
dataOUT[9] = 13;
dataOUT[10] = '\0';
```

E' possibile vedere che il buffer di uscita, ovvero la stringa che verrà scritta in uscita alla pressione del pulsante è inizializzata carattere per carattere, in particolare viene terminata, come richiesto da una stringa, con il carattere '\0'. Il carattere con indice pari ad 8 non viene scritto, questo poiché la posizione 8 rappresenta il numero del pulsante premuto e viene aggiornata alla pressione del pulsante prima di inviare la stringa via USB[22]. Dopo questa sezione, viene fatta l'inizializzazione del modulo USB, con la rispettiva chiamata delle funzioni:

[22] Il carattere 9 è impostato a 13 poiché tale valore in ASCII equivale ad un ritorno a capo. A seconda delle impostazioni del Terminal il suo effetto è quello di passare alla riga successiva o tornare a inizio testo.

```
  // Inizializzazione base del Modulo USB
  InitializeSystem();

  // Inizializzazione del modulo USB per utilizzo interruzioni
  USBDeviceAttach();
```

Dopo l'inizializzazione del modulo USB vengono inizializzate le porte del PIC in maniera da poter avere PORTD come uscita (per controllare i LED) e PORTB4-PORTB7 come ingressi. Oltre a queste impostazioni si noti la chiamata alla funzione:

```
// Abilita i resistori di pull-up sulla PORTB
EnablePullups();
```

richiesta per abilitare i resistori di pull-up presenti nella PORB. Questa chiamata di funzione è possibile solo aggiungendo il file portb.h tra gli include file:

```
#include <portb.h>
```

Questo file deve essere incluso all'inizio del file main.c, subito dopo le altre direttive `#include`. Si noti che i resistori di pull-up non sono necessari qualora sia faccia uso della scheda EasyUSB.
Una volta inizializzato il sistema, inizia il ciclo infinito ottenuto per mezzo di un `while`. All'interno del ciclo infinito viene richiamata la funzione di libreria :

```
CDCTxService();
```

Questa funzione, di cui non si è precedentemente parlato, deve essere richiamata almeno una volta per ogni ciclo e serve per gestire il buffer di trasmissione, in particolare permette il corretto funzionamento della funzione USBUSARTIsTxTrfReady () quindi la possibilità di controllare se il buffer di uscita USB è pronto per la trasmissione[23]. Se si volesse per esempio ricevere solo informazioni dal PC non sarebbe necessario richiamare tale funzione. Dal momento che in questo esempio oltre che a ricevere dati dal PC trasmettiamo anche dati verso il PC, è invece necessario richiamarla ad ogni ciclo `while`. A questo punto è bene anche mettere in evidenza il fatto che la funzione USBUSARTIsTxTrfReady () non deve essere richiamata in un ciclo `while`, visto che questo risulta bloccante, in particolare creerebbe uno stallo nel ciclo `while` poiché non verrebbe più chiamata la funzione CDCTxService, che permetterebbe di sbloccare lo stallo. Dopo questa breve digressione vediamo il programma vero e proprio:

```
// Controllo pressione BT1
if (PORTBbits.RB4 == 0 )
  if(USBUSARTIsTxTrfReady()) {
      dataOUT[8] = '1';
      putsUSBUSART(dataOUT);
  }

// Controllo pressione BT2
if (PORTBbits.RB5 == 0 )
  if(USBUSARTIsTxTrfReady()) {
      dataOUT[8] = '2';
      putsUSBUSART(dataOUT);
  }

// Controllo pressione BT3
```

[23] Non chiamando la CDCTxService, la funzione USBUSARTIsTxTrfReady ritorna sempre 0, come se non fosse possibile trasmettere dati.

```
if (PORTBbits.RB6 == 0 )
  if(USBUSARTIsTxTrfReady()) {
      dataOUT[8] = '3';
      putsUSBUSART(dataOUT);
  }

// Controllo pressione BT4
if (PORTBbits.RB7 == 0 )
  if(USBUSARTIsTxTrfReady()) {
      dataOUT[8] = '4';
      putsUSBUSART(dataOUT);
  }
```

Come visibile, il codice è piuttosto simmetrico, in particolare questa parte di codice controlla se è stato premuto un pulsate. Qualora sia stato premuto un pulsante, prima di inviare le informazioni del tasto premuto, si richiama la funzione:

```
if(USBUSARTIsTxTrfReady())
```

Se il modulo è pronto per trasmettere dati si carica il valore del pulsante premuto all'interno del carattere 8 del buffer[24] dataOUT e il buffer (stringa) viene trasmesso in uscita:

```
dataOUT[8] = '1';
putsUSBUSART(dataOUT);
```

Questo viene fatto per ogni pulsante e ripetuto ad ogni ciclo `while`. Il pezzo di programma appena esposto non è snello ma è chiaro, dunque di facile lettura. Dopo aver controllato i pulsanti viene controllato il buffer in ingresso, leggendo un un byte dal buffer:

```
dataIN = 0;

// Leggo un byte dal buffer in ingresso
getsUSBUSART (&dataIN, 1);
```

Si noti che, poiché non si è fatto uso di un Array si è passato l'indirizzo della variabile per mezzo dell'operatore &, ovvero l'indirizzo della variabile. Qualora dataIN fosse stato un Array, si ricorda che il valore dataIN senza parentesi quadre, rappresenta l'indirizzo dove inizia l'Array stesso. Si noti inoltre che la lunghezza del buffer, essendo dataIN una normale variabile, è pari ad 1. Dopo aver letto l'eventuale byte dal buffer in ingresso viene effettuato un confronto con il valore ASCII corrispondente ad 1,2,3,4,5,6,7,8. ovvero 0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38 rispettivamente.

```
// Controllo se è stato premuto 1 sulla tastiera
if (dataIN == 0x31)
  LATD = 0x01;

// Controllo se è stato premuto 2 sulla tastiera
if (dataIN == 0x32)
  LATD = 0x02;

// Controllo se è stato premuto 3 sulla tastiera
if (dataIN == 0x33)
  LATD = 0x04;

// Controllo se è stato premuto 4 sulla tastiera
if (dataIN == 0x34)
  LATD = 0x08;
```

[24] Si noti che il valore del pulsante è caricato come carattere e non come numero.

```
// Controllo se è stato premuto 5 sulla tastiera
if (dataIN == 0x35)
   LATD = 0x10;

// Controllo se è stato premuto 6 sulla tastiera
if (dataIN == 0x36)
   LATD = 0x20;

// Controllo se è stato premuto 7 sulla tastiera
if (dataIN == 0x37)
   LATD = 0x40;

// Controllo se è stato premuto 8 sulla tastiera
if (dataIN == 0x38)
   LATD = 0x80;
```

A seconda del valore premuto viene acceso il rispettivo LED. Si noti che per accendere solo un LED, qualora per esempio si prema il numero 8, sulla PORTD non deve essere scritto il valore 8, ma il valore binario 1000 0000, ovvero 0x80.

Una volta scritto il programma è possibile compilarlo e caricare il tutto all'interno del PIC18F4550. Avviato il sistema e collegato alla porta USB, Windows riconoscerà automaticamente il dispositivo e caricherà il driver utilizzato per l'Esempio 1, visto che il PID e VID[25] sono gli stessi. Una volta che il driver è stato installato, verrà creata una porta seriale emulata. Per mezzo di HyperTerminal aprire una connessione con questa porta seriale, utilizzando le impostazioni riportate in Figura 22[26].

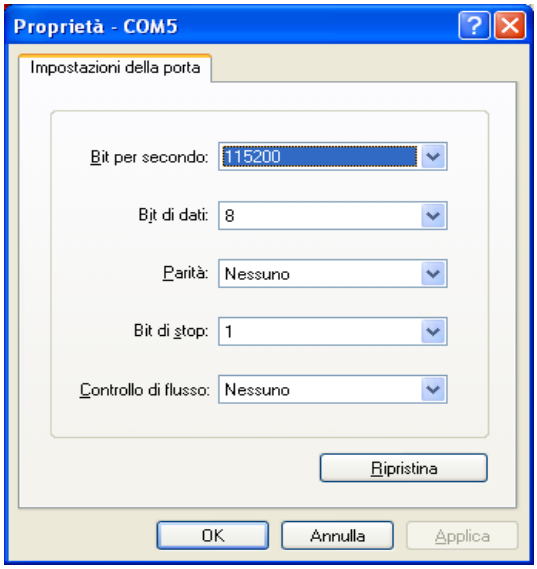

Figura 22: *Impostazioni porta COM del terminale*

Una volta avviata la connessione con la porta emulata, premendo uno dei tasti 1,2,3,4,5,6,7,8 accenderete il LED corrispondente, mentre premendo uno dei quattro tasti scriverete sul terminale il tasto premuto. In Figura 23 è mostrato il terminale quando vien premuto il pulsante BT1.

[25] Si rimanda a fine articolo per la trattazione del PID e VID.

[26] La frequenza (Baud-rate) di 115200Kbits/sec è la massima supportata dal driver attuale di emulazione della porta seriale. Le impostazioni selezionate sono riflesse anche nell'UART interna al PIC. La libreria fornisce funzioni, non trattate in questo articolo per controllare il valore del Baud-rate.

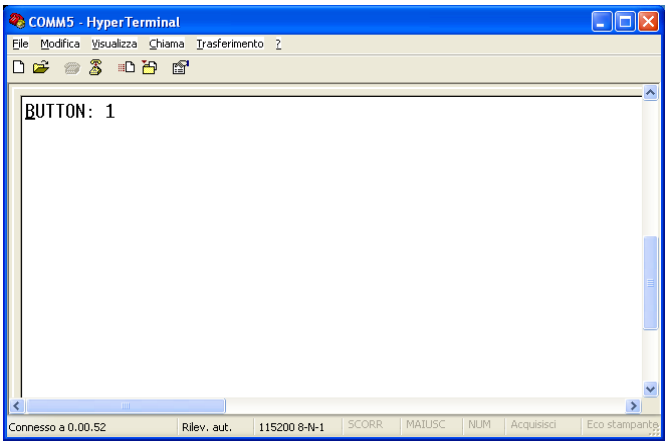

Figura 23: V*isualizzazione del messaggio trasmesso via USB alla pressione del Pulsante 1.*

Da quanto appena esposto si capisce che questo esempio può essere utilizzato per poter controllare dei Relay ed accendere delle luci. Inoltre i pulsanti potrebbero essere sostituiti con dei sensori che controllano la chiusura di porte o finestre. Le applicazioni reali sapete bene che possono essere molte altre...

...non scordiamoci il PID e VID

Tutti contenti abbiamo i nostri progetti funzionanti...però non bisogna dimenticarsi del VID (Vendor ID) e PID (Product ID), visto che ogni progetto, se commerciale, deve avere una combinazione unica VID, PID. I progettisti che vogliono mettere sul mercato un prodotto devono attenersi alle regole del consorzio USB, che rilascia sotto pagamento il numero VID, accompagnato da 65536 combinazioni di PID, ovvero 2 alla 16 combinazioni. Il prezzo del VID dipende dal tipo di contratto che viene fatto ma attualmente (Febbraio 2010) il costo minimo è di 2000$. Fortunatamente per casi particolari la Microchip come anche altre case costruttrici di dispositivi USB possiedono un programma di Sublicensing per mezzo del quale è possibile utilizzare il VID del produttore ed ottenere un PID. Tra le varie condizioni richieste nell'accordo di Sublicensing vi è la condizione che VID e PID siano utilizzati su prodotti utilizzanti il componente del produttore.

Il PID e VID sono contenuti all'interno del file .inf utilizzato per installare propriamente il driver. Il formato con cui lo si trova è USB\VID_xxxx&PID_yyyy. Un esempio è:

```
%DESCRIPTION%=DriverInstall, USB\VID_04D9&PID_000A
```

che è appunto il VID e PID utilizzato dalla Microchip all'interno del file .inf. L'informazione del VID e PID oltre che essere propriamente scritti all'interno del file .inf devono essere scritti anche all'interno del nostro codice. Infatti il dispositivo hardware comunica queste informazioni all'Host (PC), che le utilizza per la ricerca del driver opportuno, ovvero il driver che secondo il file .inf supporta il VID e PID del dispositivo hardware. Nella libreria USB versione 2.8 della Microchip, il VID e il PID possono essere trovati all'intero del file usb_descriptors.c, come sotto riportato (numeri rossi).

```
/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc =
{
    0x12,                   // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE,  // DEVICE descriptor type
    0x0200,                 // USB Spec Release Number in BCD format
    CDC_DEVICE,             // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    USB_EP0_BUFF_SIZE,      // Max packet size for EP0, see usb_config.h
    0x04D8,                 // Vendor ID
    0x000A,                 // Product ID: CDC RS-232 Emulation Demo
    0x0100,                 // Device release number in BCD format
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x00,                   // Device serial number string index
    0x01                    // Number of possible configurations
};
```

Nella libreria USB versione 1.2 il VID e il PID possono essere trovati all'intero del file usbdsc.c.

In questi file, oltre al VID e PID è possibile anche cambiare gli identificatori che descrivono il tipo di dispositivo e il produttore.

Indice Alfabetico

Bibliografia

[1] www.LaurTec.com : sito ufficiale di EasyUSB e Freedom II dove poter scaricare ogni aggiornamento e applicazione. Il PCB di EasyUSB e Freedom II sono disponibili alla sezione servizi previa donazione di supporto al sito stesso.

[2] www.microchip.com : sito dove scaricare i datasheet del PIC18F4550 e la libreria MCHPFSUSB.

[3] www.usb.org : sito ufficiale del consorzio USB, dove è possibile scaricare le specifiche dello standard USB.

History

Data	Versione	Nome	Descrizione Cambiamento
20.03.10	1.0	Mauro Laurenti	Versione Originale.
18.12.10	1.1	Mauro Laurenti	Aggiornato il documento dal Framework 2.6 al Framework 2.8. Migliorata la spiegazione di alcuni punti. Aggiunta la descrizione di EasyUSB