Versione N° 1.0e

LaurTec

EasyUSB

Progettare sistemi embedded con USB

Autore : *Mauro Laurenti*

ID: PJ7008-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

La porta USB è ormai il sistema più usato per lo scambio d'informazione tra sistemi elettronici e per connettere ogni qualsivoglia dispositivo ad un PC. Anche se non di facile utilizzo come la porta parallela e la porta seriale, librerie e hardware dedicati possono agevolare notevolmente l'apprendimento e l'utilizzo della porta USB. Utilizzare l'interfaccia USB permette di rendere un nuovo progetto al passo con i tempi permettendo di seguire le aspettative dell'utente finale.
La scheda di sviluppo EasyUSB ha lo scopo di agevolare lo studio della porta USB permettendo con un hardware essenziale ma al tempo stesso fondamentale, di realizzare periferiche collegabili al PC. Il connettore di espansione compatibile con Freedom II la rendono anche idonea quale valida alternativa economica per avere una mini piattaforma di sviluppo con la quale esercitarsi. EasyUSB è compatibile con i programmatori Microchip e può essere richiesta con PIC18F4550 programmato con USB Bootloader, ovvero con la possibilità di iniziare a sviluppare le proprie applicazioni senza bisogno di un programmatore.

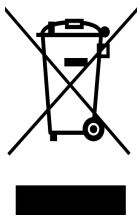
Specifiche Tecniche

Alimentazione : 9V-12V DC
Assorbimento : 65mA medio @ 12V (LED attivi), 55mA medio USB powered (LED attivi).
Dimensioni : 73 x 84 mm
Part Number : L1000-30008
Versione : 1
Peso : 53g

Il sistema EasyUSB supporta il seguente hardware:

- Supporto USB 2.0 low speed e full speed
- 2 pulsanti + Reset
- 4 Led di utilizzo generico
- 5 PMOS programmabili per il power ORing
- Connettore espansione 40 pin
- Programmabilità per mezzo dell'USB Bootloader
- Programmazione on-board e Debug compatibile con gli strumenti Microchip

Smaltimento



Secondo la Direttiva Europea 2002/96/EC tutti i dispositivi elettrici/elettronici devono essere considerati rifiuti speciali e non devono essere gettati tra i rifiuti domestici. La gestione e lo smaltimento dei rifiuti elettrici/elettronici viene a dipendere dalle autorità locali e governative. Un corretto smaltimento dei rifiuti permette di prevenire conseguenze negative per l'ambiente e ai suoi abitanti. E' obbligo morale, nonché legale, di ogni singolo cittadino, di attenersi alla seguente Direttiva. Per ulteriori chiarimenti l'Autore è a disposizione.

Analisi del progetto

In Figura 1 è riportato lo schema elettrico della scheda di sviluppo EasyUSB. Dalle dimensioni ridotte e dall'hardware disponibile è possibile subito capire che la scheda può essere utilizzata per espandere altri sistemi di elettronici aggiungendo agli stessi la funzionalità di una porta USB. La scheda si presta, grazie alla presenza di 2 pulsanti e 4 LED, anche allo sviluppo di piccoli sistemi di prova, estensibile poi ad una qualsivoglia complessità grazie al connettore di espansione EX1, compatibile con la scheda di sviluppo Freedom II.

Per una corretta descrizione e facile comprensione dello schema, l'hardware verrà introdotto a blocchi funzionali.

Il microcontrollore

Come prima cosa cerchiamo di capire con quali microcontrollori sia possibile utilizzare sulla scheda di sviluppo EasyUSB. La scheda nasce con lo scopo primario di permettere lo sviluppo di applicazioni USB, pertanto è ottimizzata per il supporto di microcontrollori della famiglia PIC18 a 40 pin che possiedono il modulo USB ovvero il modulo SIE (Serial Interface Engine), in particolare per il PIC18F4455, PIC18F4550 e PIC18F4553. Ciononostante altri PIC potrebbero essere utilizzati allo scopo di realizzare sistemi di altra natura, ma non tutto l'hardware potrebbe risultare utilizzabile

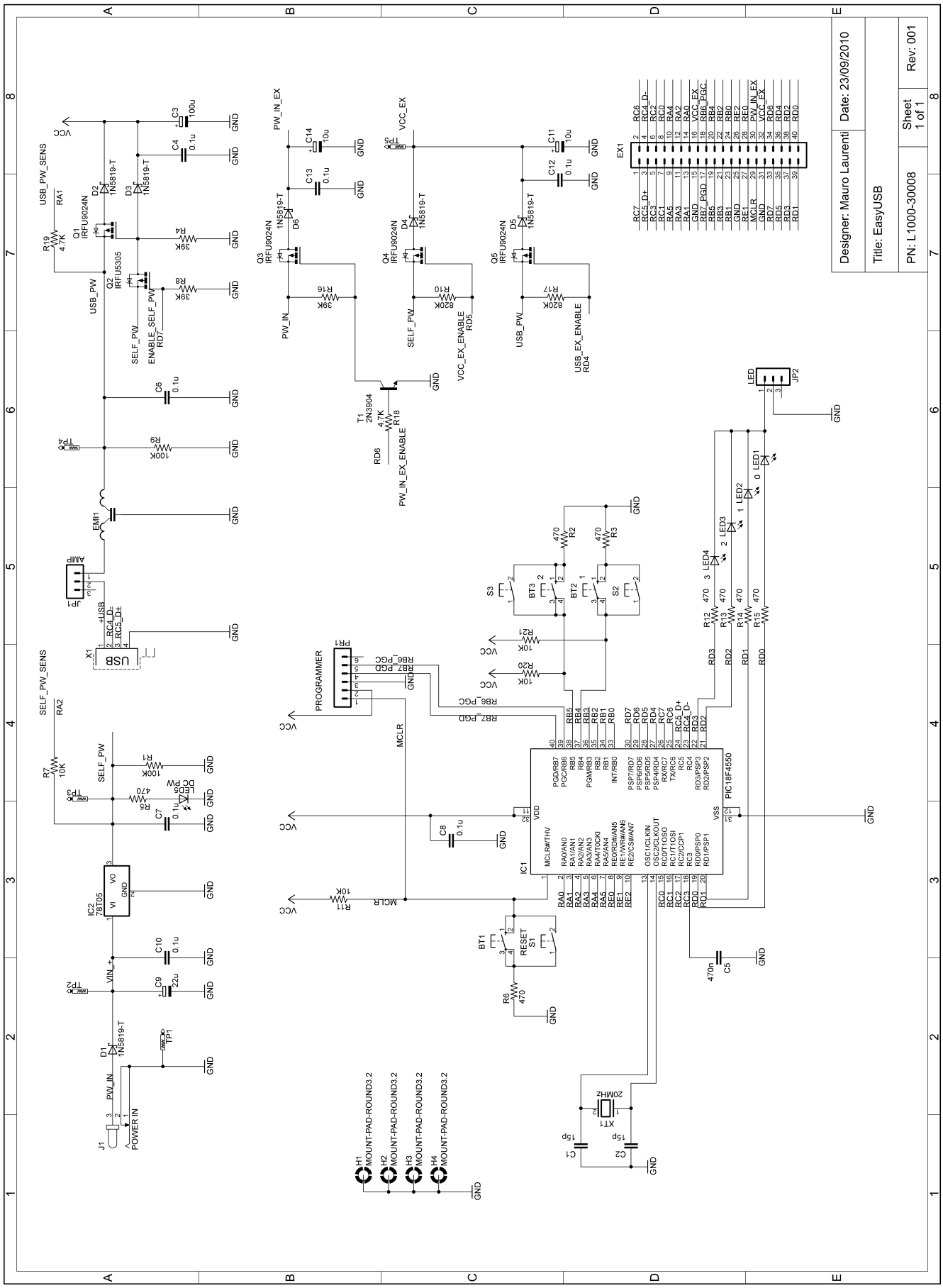


Figura 1: *Schema elettrico di EasyUSB*

Lista Componenti

Resistori

R1 = 100KΩ %5 1/4W
R2 = 470Ω %5 1/4W
R3 = 470Ω %5 1/4W
R4 = 39KΩ %5 1/4W
R5 = 470Ω %5 1/4W
R6 = 470Ω %5 1/4W
R7 = 10KΩ %5 1/4W
R8 = 39KΩ %5 1/4W
R9 = 100KΩ %5 1/4W
R10 = 820KΩ %5 1/4W
R11 = 10KΩ %5 1/4W
R12 = 470Ω %5 1/4W
R13 = 470Ω %5 1/4W
R14 = 470Ω %5 1/4W
R15 = 470Ω %5 1/4W
R16 = 39KΩ %5 1/4W
R17 = 820KΩ %5 1/4W
R18 = 4.7KΩ %5 1/4W
R19 = 4.7KΩ %5 1/4W
R20 = 10KΩ %5 1/4W
R21 = 10KΩ %5 1/4W

Condensatori

C1 = 15pF ceramico
C2 = 15pF ceramico
C3 = 100µF elettrolitico 50V
C4 = 0.1µF ceramico 50V
C5 = 470nF ceramico 50V
C6 = 0.1µF ceramico 50V
C7 = 0.1µF ceramico 50V
C8 = 0.1µF ceramico 50V
C9 = 22µF elettrolitico 35V
C10 = 0.1µF ceramico 100V
C11 = 10µF elettrolitico 100V
C12 = 0.1µF ceramico 100V
C13 = 0.1µF ceramico 25V
C14 = 10µF elettrolitico 50V
EMI = Filtro EMI (DSS6 NF31C 223)

Circuiti Integrati

IC1 = PIC18F4550
IC2 = 7805

Transistor

T1 = Transistor NPN 2N3904
Q1 = P-MOS IRFU9024N
Q2 = P-MOS IRFU9024N
Q3 = P-MOS IRFU9024N
Q4 = P-MOS IRFU9024N
Q5 = P-MOS IRFU9024N

Diodi

LED1 = led 3mm rosso
LED2 = led 3mm rosso
LED3 = led 3mm rosso
LED4 = led 3mm verde
LED5 = led 3mm verde
D1-D6= 1N5819

Quarzi

Q1 = 20MHz

Pulsanti

BT1 = micro-pulsante per PCB verticale (opzionale)
BT2 = micro-pulsante per PCB verticale (opzionale)
BT3 = micro-pulsante per PCB verticale (opzionale)
S1 = micro-pulsante per PCB
S2 = micro-pulsante per PCB
S3 = micro-pulsante per PCB

Connettori

EX1 = ICD 40 pin (maschio)
X1 = Connettore USB per periferiche (tipo B)
J1 = Connettore cilindrico alimentatore 2.1mm
JP1 = Jumper 3 pin (AMP)
JP2 = Jumper 3 pin (LED)
PR1 = Jumper 6 pin 90 gradi

Il quarzo

Ogni microcontrollore ha bisogno di un clock per poter funzionare. Il clock rappresenta la base del tempo che il microcontrollore utilizza per l'esecuzione delle istruzioni per cui è stato programmato. Alcuni microcontrollori della Microchip, tra cui il PIC18F4550, hanno la possibilità di generare un clock interno. Questo significa che lo schema base per utilizzare il microcontrollore consiste semplicemente nella connessione dell'alimentazione e della circuiteria di Reset[1]. Ciononostante la presenza del quarzo può in ogni modo essere necessaria per quelle applicazioni in cui la stabilità del clock è importante (per esempio l'utilizzo del modulo USB). Infatti per mezzo del quarzo esterno è possibile ottenere una stabilità del clock superiore a quella ottenibile utilizzando l'oscillatore interno.

Si fa notare che, sulla scheda EasyUSB, i pin associati al quarzo, qualora si facesse uso dell'oscillatore interno, non possono essere utilizzati come I/O pin. Questa scelta discende dal fatto, che se si fossero aggiunte le piste per supportare l'opzione di utilizzare i pin del quarzo, si sarebbero potuti avere, problemi di oscillazione.

In ultimo è bene far notare che i condensatori C1 e C2 da 15pF sono idonei per un quarzo da 20MHz. Nel caso si facesse uso di un quarzo a frequenza differente è bene accertarsi che i condensatori C1 e C2 siano ancora idonei a permettere la sua oscillazione. Maggiori informazioni a riguardo, è possibile trovarle nel datasheet del microcontrollore utilizzato. Per completezza si riportano in Tabella 1 i valori tipici di C1 e C2 per quarzi differenti[2]. Il datasheet dal quale è stata prelevata l'informazione è quello del PIC18F4550.

Frequenza	Valore C1 e C2
4MHz	27pF
8MHz	22pF
20MHz	15pF

Tabella 1: V*alori tipici di C1 e C2*

La circuiteria per generare il clock interno dei PIC18F4550, al fine di poter supportare il modulo USB risulta più complessa dei normali PIC18, lo schema a blocchi è riportato in Figura 2.

[1] Dal momento che i PIC possiedono una circuiteria di Reset interna, quella esterna non è obbligatoria....ma dal momento che non è escluso che il nostro programma si blocchi, è bene prevedere un Reset esterno.

[2] Per una più completa descrizione dell'oscillatore dei PIC è bene far riferimento sempre al datasheet. Infatti i PIC accettano anche risonatori ceramici, per i quali è necessario utilizzare valori di C1 e C2 differenti. In ultimo, ma non meno importante, i PIC hanno differenti modalità di oscillazione, da selezionare opportunamente a seconda della frequenza del quarzo.

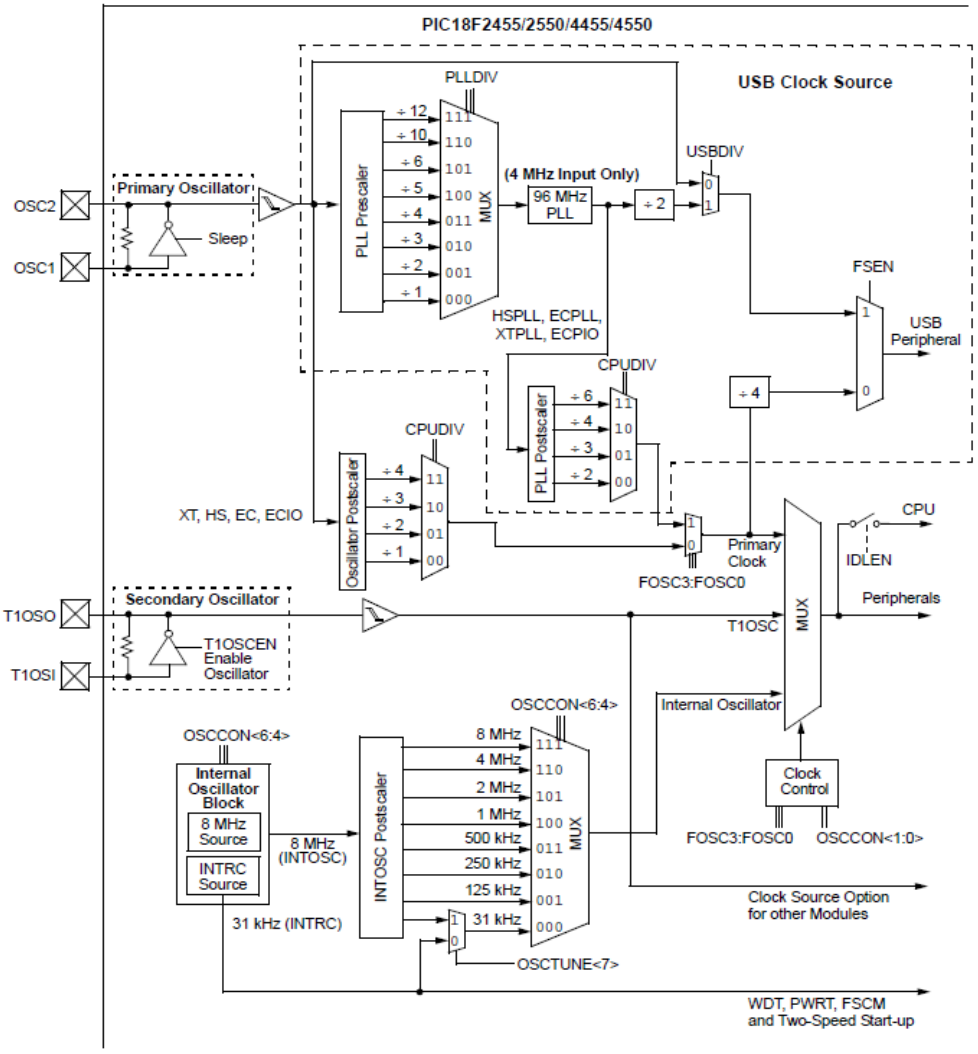


Figura 2: *Schema a blocchi della circuiteria di Clock del PIC18F4550*

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 selezionabili 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[3].

[3] Si fa notare che essere "più caldo" è un modo per dire che viene dissipata più potenza, quindi viene assorbita una

Le impostazioni ora descritte devono essere effettuate per mezzo dei registri di configurazione ovvero, in C18, 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 configurazione si è abilitata 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 3.3V necessari per i resistori di pull-up che definiscono la modalità *low speed* e *full speed*[4]. 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? EasyUSB sfrutta sia il regolatore lineare[5] che i resistori interni, è dunque necessario abilitarli; il regolatore interno viene abilitato per mezzo della direttiva `#pragma`, scrivendo:

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

Le configurazioni ora descritte, ovvero che utilizzano la direttiva `#pragma`, vanno propriamente impostate anche facendo uso della libreria (USB Framework) della Microchip. 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:

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

Come detto questa riga di codice non sarà necessaria se si fa uso della libreria Microchip, poiché gestita dalla libreria stessa. Altri registri, ovvero bit da impostare sono:

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

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

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

corrente maggiore.

[4] 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.

[5] Il condensatore C5 collegato al pin RC3 permette di stabilizzare il regolatore interno al PIC.

Anche per queste variabili il Framework della Microchip gestisce il tutto, anche se sarà compito del programmatore accertarsi che le impostazioni siano effettivamente come richieste dall'applicazione.

L'alimentatore

La sezione di alimentazione della scheda EasyUSB è la parte più complessa, dal momento che la scheda supporta sia l'alimentazione esterna che quella da porta USB. Nel primo caso si parla di Self Powered Device (Dispositivo con alimentazione propria) mentre nel secondo caso si parla USB Powered Device (Dispositivo alimentato da USB). La sezione di alimentazione si divide in tre parti funzionali:

- Alimentatore DC-DC per alimentazione da alimentatore
- Power ORing per alimentazione da USB
- Controllo dell'alimentazione dei dispositivi esterni

Vediamo in dettaglio ogni singolo blocco:

Alimentatore DC-DC per alimentazione da alimentatore

In Figura 3 è riportato lo schema elettrico della circuiteria che permette di regolare la tensione in ingresso proveniente da un normale alimentatore DC. Questo può essere collegato alla scheda facendo uso del connettore standard cilindrico da 2,1 mm (J1) secondo la seguente polarità: -(•)-+
L'alimentatore deve avere una tensione di 9V con possibilità di erogare almeno 600mA. Nel caso si faccia uso di alimentazione a 12V IC2 richiede un dissipatore termico idoneo per la potenza che l'integrato deve dissipare[6].

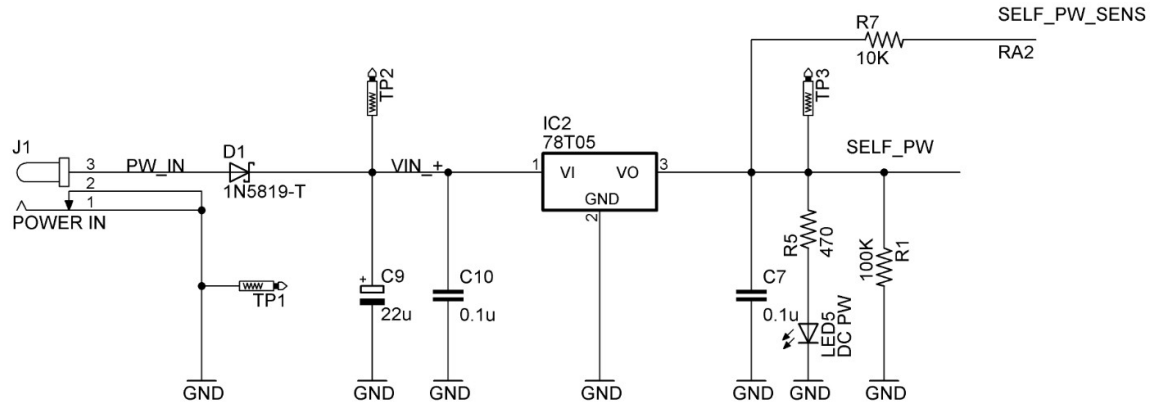


Figura 3: *Schema elettrico del convertitore DC-DC*

E' possibile notare che in ingresso è presente un diodo Schottky al fine di proteggere il convertitore 78T05 e la circuiteria a valle, da una eventuale inversione di polarità dell'alimentatore[7]. Il vantaggio di usare un diodo Schottky rispetto a diodi normali al silicio risiede nel fatto che la caduta di potenziale che si viene ad avere ai suoi capi è di circa 0,2V-0,4V mentre quella di un normale diodo al silicio quale il diodo 1N4004 ha una caduta di tensione di 0,6V-0,8V. L'avere una caduta di tensione inferiore si traduce nel fatto che l'alimentatore risulta più efficiente ovvero consuma meno potenza[8]. Dal momento che si fa però d'uso del regolatore lineare 78T05 che per sua natura non è molto efficiente,

[6] Fino ad un massimo di 2W non è necessario un dissipatore esterno, mentre per potenze maggiori, sia che l'alimentazione sia di 9V che di 12V, è bene prevedere un dissipatore termico opportuno. Qualora per ragioni di spazio non si voglia utilizzare un dissipatore termico è possibile sostituire l'integrato 7805 con un regolatore switching compatibile con la serie 78xx. La società RECOM realizza per esempio l'integrato R-785.0-1.0 pin compatibile con la serie 78xx ma che racchiude un sub sistema intero con alimentatore switching.

[7] Si noti che la tensione PW_IN riportata al connettore EX1 non risulta protetta da una eventuale inversione di polarità per mezzo del diodo D1, bensì dal PMOS Q3 che non riuscirà ad entrare in conduzione.

[8] La potenza dissipata nel diodo è data dal prodotto della caduta di tensione ai suoi capi per la corrente che lo attraversa.

l'utilizzo del diodo Schottky si sarebbe potuto risparmiare, facendo uso di un normale diodo 1N4004 (come si è fatto per esempio per il progetto Freedom II). La ragione per cui ho in ogni modo deciso di mettere il diodo Schottky è legata al fatto che lo stesso diodo viene utilizzato negli altri blocchi che verranno a breve descritti. In questi blocchi la presenza del diodo Schottky è più importante e non si può evitare. Dal momento che il diodo 1N5819 e il diodo 1N4004 hanno lo stesso package, al fine di evitare errori di assemblaggio e montare il diodo 1N4004 in punti in cui è richiesto necessariamente il diodo 1N5819, ho preferito avere un solo diodo. Questa scelta dimostra che frequentemente lo scegliere un componente piuttosto che un altro può essere semplicemente influenzata dall'uso del buon senso, naturalmente sempre supportato dalla ragione. In ogni modo la regola di avere meno tipi di componenti possibili, ovvero quanti più componenti dello stesso tipo, è una buona regola che può semplificare il montaggio e in alcuni casi risparmiare soldi (potendo comprare quantità più grandi dello stesso componente).

In uscita al regolatore è presente il LED5 di colore rosso. Questo segnala se la scheda di sviluppo è alimentata. Tale spia è particolarmente utile nel caso in cui si alimenti la scheda con polarità errata; infatti se così fosse, tale LED non si accenderebbe. E' bene far notare che il LED5, seppur acceso, non indica che l'alimentazione ha un valore corretto. Infatti il regolatore 78T05[9], per poter funzionare correttamente deve avere al suo ingresso almeno 7.5V. Un valore di tensione più basso potrebbe ancora riuscire a far accendere il LED5, ma la tensione in uscita al 78T05 non sarà quella di 5V. Un test che è possibile fare in caso di problemi, è per mezzo dei cosiddetti Test Point, TP1 e TP3, che altro non sono che dei fori metallizzati in cui è possibile facilmente posizionare i puntali del tester. Si capisce che se tutto funziona correttamente, tra TP1 e TP3 si dovrebbe misurare una tensione di 5V. In Tabella 2 è riportato un riassunto dei Test Point e delle tensioni che è possibile misurare.

Test Point	Funzione
TP1	GND
TP2	+8.6V (se alimentato a 9V)
TP3	+5V
TP4	+5V USB
TP5	4.6V VCC_EX (EX1)

Tabella 2: *Funzione dei Test Point*

La presenza o meno dell'alimentazione esterna è possibile leggerla per mezzo del segnale SELF_PW_SENS prelevato dal resistore R7. Il resistore R1 ha il compito di collegare a massa l'uscita qualora non siano presenti i +5V.

Ulteriore nota meritano i condensatori, sia in ingresso che in uscita. Sebbene l'alimentatore da collegare debba avere una tensione DC in uscita, frequentemente questi hanno un ripple piuttosto alto. Al fine di ottenere 5V quanto più stabili possibile, il filtraggio effettuato dai condensatori in ingresso, nonché in uscita[10], risulta vitale. Per esempio un ripple troppo alto potrebbe causare il Reset del microcontrollore.

Power ORing per alimentazione da USB

Il sistema EasyUSB può essere alimentato anche dalla porta USB del PC in maniera da rendere il sistema quanto più compatto e versatile possibile[11]. Si ricorda che un sistema che preleva la corrente

[9] Si fa notare che il limite massimo di corrente del regolatore 78T05 è di 1A. Si raccomanda un idoneo dissipatore termico nel caso in cui si faccia uso di hardware esterno connesso al connettore di espansione o di tensioni superiori a 9V.

[10] I condensatori in uscita hanno prevalentemente il compito di migliorare la risposta in frequenza del regolatore IC2.

[11] Dal momento che la scheda è programmabile via USB e l'alimentazione è a sua volta prelevabile dalla stessa porta, è possibile iniziare a programmare semplicemente connettendo la scheda al PC.

dalla porta USB, una volta connesso deve prelevare, secondo le specifiche USB 2.0, non più si 100mA. Solo dopo aver richiesto maggior corrente all'USB Host, fino ad un massimo di 500mA, qualora la richiesta gli venga approvata, il sistema può iniziare a prelevare correnti maggiori di 100mA.

Per raggiungere questa flessibilità sull'alimentazione si è fatto uso del power ORing. La parola Power ORing significa fare l'OR delle alimentazioni, ovvero o l'una o l'altra delle alimentazioni disponibili può essere utilizzata. In Figura 4 è riportato lo schema elettrico della sezione di alimentazione relativa alla porta USB. É possibile vedere che in ingresso è presente il Jumper JP1 nominato AMP; questo deve essere sempre chiuso al fine di rendere l'alimentazione da USB possibile. La sua utilità risiede nel fatto che per mezzo di questo Jumper è possibile facilmente misurare la corrente assorbita dal sistema attraverso la porta USB. Infatti basterà aprire il ponte e collegare un amperometro per controllare l'assorbimento; questo risulta utile qualora si vogliano monitorare e verificare gli assorbimenti del sistema.

Dopo il Jumper è presente un filtro passa basso a T EMI (Electromagnetic Interference), il nome di filtro a T discende dalla sua forma. La presenza del filtro permette di limitare i disturbi provenienti dal PC e al tempo stesso limitare i disturbi che il sistema potrebbe iniettare per via condotta[12] al PC.
Dopo il filtro è presente la resistenza R9 il cui compito è di mantenere la linea a massa qualora non sia presente la connessione con il cavo USB. In questo modo leggendo lo stato dell'alimentazione per mezzo del pin RA1 (USB_PW_SENS) è possibile leggere 0 logico. Si osservi che la resistenza è di valore piuttosto alto poiché le specifiche USB prevedono che in caso sia attiva la modalità sleep il sistema USB non prelevi più di 500µA. Si osservi che diversamente da altri sistemi commerciali non si è collegato un LED sulla porta USB al fine di segnalare la presenza della tensione, se cosi si fosse fatto il sistema non sarebbe stato compatibile con le specifiche USB in modalità sleep a causa della corrente assorbita dal LED. Il LED per segnalare la presenza dell'alimentazione è in realtà presente ed è il LED4 (USB_PW), il quale è posizionato vicino al LED5 dell'alimentazione esterna ma è controllato dal PIC. In questo modo volendo entrare in stato di sleep è possibile disattivare il diodo LED rimanendo delle specifiche USB.

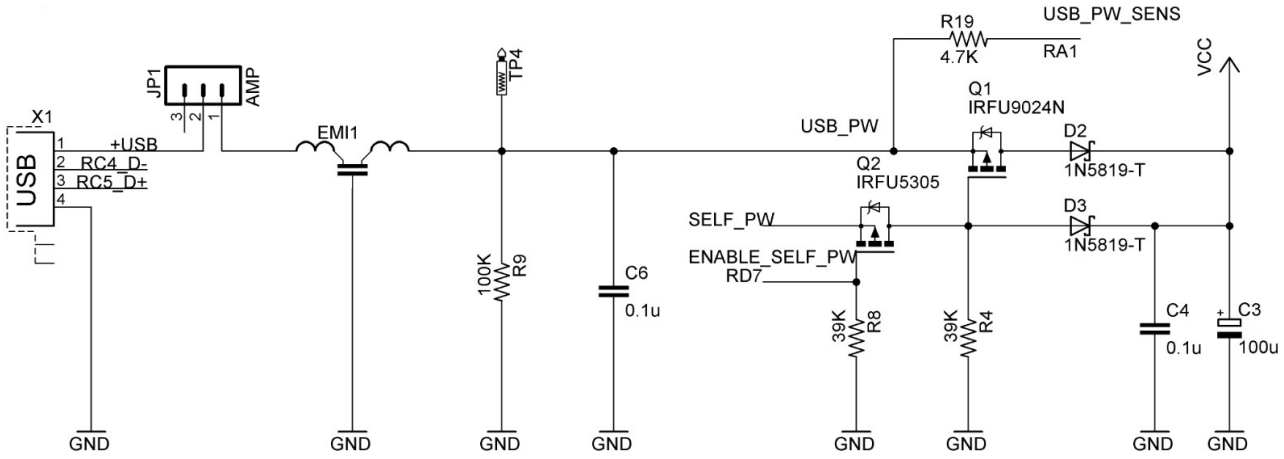


Figura 4: *Schema elettrico della sezione di Power ORing (USB alimentatore)*

La scelta di utilizzare l'alimentazione da rete o da USB è possibile farla per mezzo dei transistor PMOS Q1 e Q2. In particolare il passaggio da un'alimentazione all'altra è automatico e l'alimentazione da rete ha maggior priorità, ovvero quando si collega l'alimentatore esterno la corrente viene prelevata dall'alimentatore di rete piuttosto che dalla porta USB. Vediamo come funziona.

[12] Le interferenze di natura condotta rappresentano quella categoria d'interferenza che si propaga lungo i cavi. Da un punto di vista delle normative europee ha interesse limitare interferenze condotte nel range tra 150KHz e 30MHz. Al disopra di queste frequenze si parla più propriamente d'interferenza radiata.

I transistor PMOS si comportano come degli interruttori, quando il Gate si trova ad una tensione negativa rispetto al Source il transistor è come un interruttore chiuso. Supponiamo che il sistema EasyUSB sia collegato solo alla porta USB, dallo schema è possibile vedere che Q1 ha il Gate collegato a massa per mezzo della resistenza R4 mentre il suo Source (piedino con la freccia) è collegato a +5V della porta USB. Questo equivale a dire che il Gate è ad una tensione negativa di 5V rispetto al Source, quindi il transistor è un interruttore chiuso e i +5V (o meglio la corrente) passano attraverso il diodo D2 e vanno infine a Vcc, alimentando il sistema (solo la scheda e non il connettore EX1). Fino a quando non è collegato nessun alimentatore esterno, Q2 risulta un interruttore aperto visto che il suo Source è collegato a massa tramite R1. Quando si collega un eventuale alimentatore esterno, i +5V uscenti dal regolatore 78T05 (IC2) alimentano il Source di Q2 il cui Gate essendo a massa permette al transistor di chiudersi facendo passare i +5V i quali si troveranno sul Gate di Q1, il quale verrà dunque interdetto, disattivando l'alimentazione da USB in favore di quella dell'alimentatore di rete. Per avere la massima flessibilità nella gestione delle alimentazioni, qualora si voglia per esempio alimentare il microcontrollore per mezzo della porta USB e il connettore EX1 per mezzo dell'alimentatore di rete, è possibile disattivare Q2 per mezzo della linea RD7. Il passaggio da un'alimentazione ad un altra è bene farla sempre dopo una lettura della presenza della stessa al fine di accertarsi di non creare Reset imprevisti del microcontrollore[13]. Si fa presente che i pin di rilevamento di USB_PW_SENS e SELF_PW_SENS sono compatibili con la scheda di sviluppo PICDEM™ FS USB della Microchip.

La presenza dei diodi D2 e D3 permettono di proteggere i circuiti prima dei MOS qualora sia presente una tensione a valle mentre a monte l'alimentazione sia scollegata o disattivata. In questo caso infatti il diodo di protezione interno al transistor (usato normalmente a protezione della giunzione del transistor stesso) entrerebbe in conduzione causando correnti spiacevoli verso l'USB o il regolatore IC2. Al fine di limitare la caduta di tensione ai capi dei diodi e permettere il corretto funzionamento del microcontrollore entro le specifiche USB, il diodo è di tipo Schottky.

Sempre per limitare la caduta di potenziale nella gestione delle alimentazioni, ovvero nell'implementazione del Power ORing, si è preferito il transistor MOS al transistor BJT (Transistor standard) dal momento che la Ron dei transistor MOS e quindi la caduta di potenziale ai capi del Source e Drain è minore di quella ottenibile con i transistor BJT. Si si pensi per esempio che la Vce in saturazione di un transistor BJT è di circa 0.2V-0.3V (per basse correnti) mentre la resistenza del PMOS scelto è di 0,175Ω che con una corrente di 100mA (massima corrente al momento in cui si attacca il dispositivo USB) comporta una caduta di tensione di soli 0.02V[14].

Controllo dell'alimentazione dei dispositivi esterni

Le specifiche USB sono tali per cui i sistemi che supportano tale standard risultano particolarmente sensibili alla problematica dei consumi. In particolare la corrente massima prevista per un dispositivo USB compatibile con le specifiche USB 2.0 è di 500mA, ma al suo avvio o meglio connessione con l'Host o Hub, il suo consumo non deve essere maggiore di 100mA[15]. Il sistema durante la sua enumerazione può richiedere più corrente, la quale gli può essere accordata o meno. Nel caso in cui l'Host o Hub non gli dovesse accordare la corrente richiesta, il sistema deve poter operare o comunque non creare problemi, facendo uso dei soli 100mA. Da questo discende che i nostri dispositivi USB

[13] Qualora non si faccia uso del pin di selezione dell'alimentazione RD7, questo deve essere impostato come ingresso. Qualora si decida di impostarlo come uscita deve avere sempre di valore logico 0 in maniera da avere il Gate di Q2 collegato a massa.

[14] Dal momento che la resistenza Ron è comparabile con la resistenza delle piste, la caduta di tensione effettivamente misurata nell'intero ramo, ovvero fino al diodo è in realtà maggiore, ma certamente molto minore della tensione Vce di un transistor in saturazione, alla quale si deve comunque sommare la caduta di tensione sulle piste.

[15] Le specifiche USB 3.0 estendono questi limiti. In particolare quando si attacca un dispositivo è concesso un assorbimento fino a 150mA, mentre il limite massimo concesso dall'Host è di 900mA. Si noti che questi limiti valgono solo se il dispositivo è High Power Superspeed, in caso contrario deve operare secondo i limiti USB 2.0. Questo significa che collegare un dispositivo USB 2.0 ad una porta USB 3.0 non porta vantaggi dal punto di vista dei limiti di corrente.

devono essere progettati cercando di limitare i consumi. Il sistema EasyUSB al fine di supportare la progettazione di sistemi fino a 500mA, ma al tempo stesso permettere un corretto avvio del sistema base, permette di controllare l'alimentazione di eventuale hardware esterno per mezzo dei PMOS di potenza Q3, Q4 e Q5. In Figura 5 è riportata la sezione dello schema elettrico che permette di disattivare l'alimentazione di Hardware esterno al fine di garantire che il sistema EasyUSB si possa avviare correttamente entro i limiti dei 100mA. Solo dopo aver accordato eventuale corrente aggiuntiva EasyUSB può facilmente attivare Vcc per l'hardware esterno attivando o il PMOS Q4 o il PMOS Q5. In particolare Q4 permette di alimentare i dispositivi esterni per mezzo della porta USB mentre Q5 per mezzo dell'alimentatore esterno. La configurazione dei transistor Q4 e Q5 è molto simile a quella di Q1 e Q2 ovvero abbiamo a che fare con un Power ORing, ciononostante in questo caso il controllo dei transistor è interamente delegato al Microcontrollore, ovvero l'attivazione dei dispositivi a valle di Q4 e Q5 avviene solo sotto l'arbitraggio del Microcontrollore.

Qualora l'hardware aggiunto a EasyUSB dovesse eccedere i 100mA è bene, all'avvio del microcontrollore porre ad 1 logico le uscite RD4 e RD5 al fine di disattivare i PMOS.

I transistor Q4 e Q5 vengono attivati ponendo a 0 logico rispettivamente il pin di controllo RD5, RD4, ovvero collegando a massa il Gate. Nel caso in cui sia presente sia l'alimentazione da USB che quella esterna, solo un transistor deve essere attivato. Come messo in evidenza in precedenza, qualora si voglia disaccoppiare l'alimentazione del PIC e quella dell'hardware esterno, il PIC potrebbe essere alimentato da USB mentre l'hardware esterno dall'alimentatore di rete.

Arrivati a questo punto avrete forse notato che Q4 e Q5 diversamente da Q1 e Q2, hanno sul gate un resistore di 820K piuttosto che 39K. La ragione di questo valore più alto è legata al fatto che quando SELF_PW o USB_PW dovesse mancare e la linea di controllo RD5 o RD4 è posta ad 1 al fine di disattivare il PMOS d'interesse, si potrebbe leggere un valore errato di PW_IN o SELF_PW.

Questo dipende dal fatto che le linee di SELF_PW_SENS e USB_PW_SENS potrebbero leggere il valore logico 1 imposto da RD4 o RD5. Ponendo la resistenza di gate da 820K questo non avviene poiché i 5V corrispondenti ad 1 logico si trovano sul partitore di tensione formato dalla rispettiva resistenza di Gate e il resistore R1 e R9 da 100K, garantendo di avere un valore inferiore a 0,8V necessari per far riconoscere al PIC un livello 0 logico sulle linee SELF_PW e USB_PW.

Oltre al controllo dell'alimentazione dei 5V, provenienti dalla regolatore IC2 o dalla stessa porta USB, EasyUSB, qualora sia presente l'alimentatore di rete permette di controllare anche Vin. Questa funzione risulta particolarmente utile poiché è come avere un Open Drain a disposizione per comandare direttamente un motore un Relay o qualunque altro carico[16], rimanendo naturalmente nelle specifiche degli assorbimenti consentiti dal sistema EasyUSB o dall'alimentatore.

Come avrete già notato Q3 non si collega al PIC come gli altri PMOS, le ragioni per questa variante sono due:

- Qualora non si fosse usato il transistor T1 il PIC si sarebbe rotto.
- Il transistor Q5 sarebbe rimasto sempre attivo

Le ragioni non sono molte ma più che valide a giustificare l'inserimento del transistor T1.

Supponiamo che il transistor T1 non sia presente e che il pin RD6 sia collegato direttamente al Gate del PMOS. Dal momento che R16 si trova collegata a PW_IN ovvero direttamente all'alimentatore di rete che può avere valori compresi tra 9V-12V, porterebbe il pin del PIC a trovarsi delle tensioni operative superiori a quelle di alimentazione causando la rottura dello stesso[17]. Detto questo

[16] Nel caso si faccia uso di carichi induttivi sarà è richiesto un diodo di ricircolo.

[17] La resistenza R16 limiterebbe la corrente nei diodi interni al PIC evitando probabilmente la rottura dello stesso, ma far funzionare qualcosa basandosi sull'attivazione continua delle protezioni interne, non è una buona pratica. Le protezioni di un sistema, di qualunque natura siano devono essere in generale considerate una protezione del sistema da eventi straordinari.

supponiamo che il PIC non si rompa, e che il transistor T1 non sia ancora presente. Come detto un PMOS si comporta come un interruttore chiuso quando il suo Gate risulta sufficientemente negativo rispetto al Source (per i PMOS usati Vgs è nel range 2V-4V[18]) mentre se il Gate non ha un potenziale sufficientemente negativo il transistor rimane aperto.

Supponiamo di voler attivare il transistor Q3; come per gli altri PMOS poniamo RD6 a 0 logico ed il nostro PMOS è effettivamente attivo. Volendo per una qualunque ragione disattivare il transistor Q3, poniamo ad 1 la nostra uscita RD6...ed il PMOS rimane ancora attivo! Il problema è che l'uscita RD6 quando è posta ad 1 vale 5V ma il Source del transistor Q3 vale 9V-12V quindi il Gate risulterà ancora negativo rispetto al Source di almeno 4V garantendo il mantenimento in conduzione dello stesso.

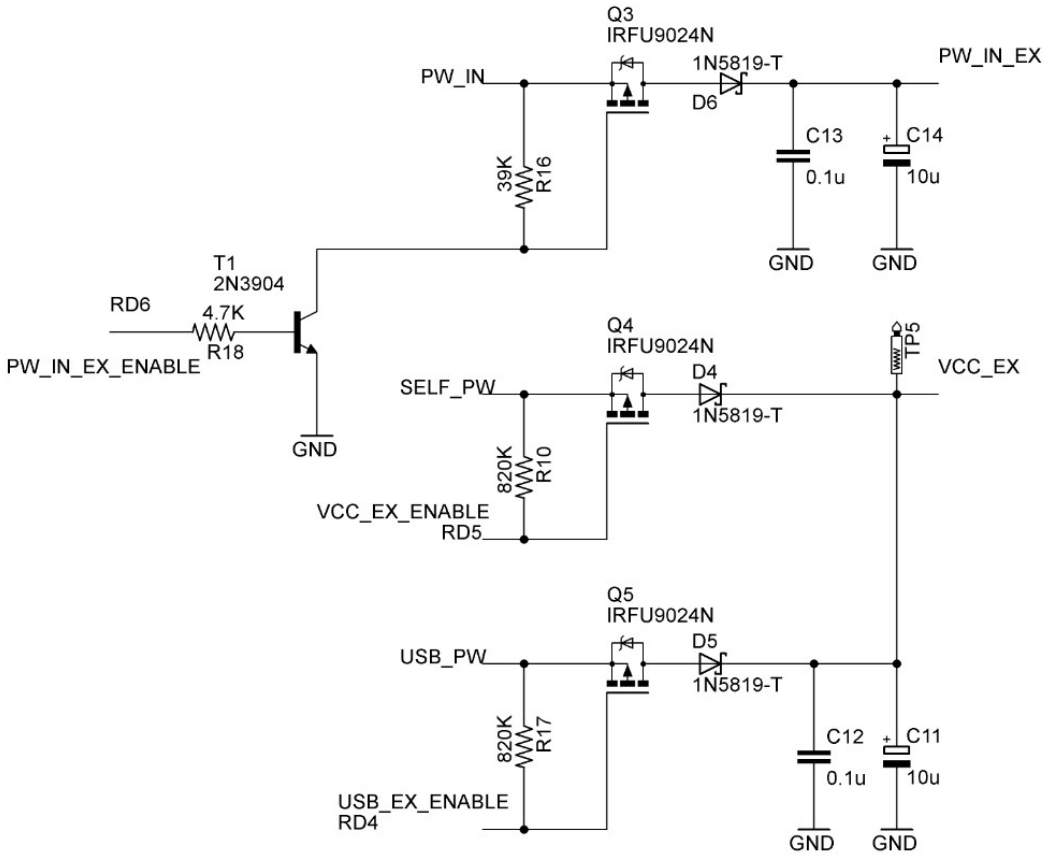


Figura 5: *Schema di controllo delle alimentazioni esterne*

Grazie alla presenza del transistor T1, i problemi sopra citati scompaiono poiché RD6 non è più collegato a PW_IN. Inoltre quando RD6 vale 1 il Gate di Q3 è collegato a massa, quindi Q3 è attivo, mentre quando RD6 vale 0, il transistor T1 è interdetto per cui su R16 non scorre corrente e il Gate di Q3 è collegato a PW_IN, per cui Q3 si apre. Il transistor T1 oltre che a proteggere il PIC si comporta anche come una NOT ovvero inverte il segnale. Infatti gli altri transistor (Q4 e Q5) sono attivi quando il pin di controllo vale 0, mentre Q3 è attivo quando il suo pin di controllo vale 1.

Da quanto appena descritto si sarà capito che EasyUSB permettendo di disattivare tutto l'Hardware connesso sul connettore di espansione, permette di aderire alle specifiche USB 2.0, anche nel caso di sleep del dispositivo.

[18] I valore di tensione Vgs necessario per attivare un MOS è riportato in ogni datasheet e può raggiungere anche tensioni nel range di 10V-20V. MOS con queste tensioni Vgs non si possono usare in applicazioni in cui si hanno a disposizione solo livelli TTL compatibili, a meno di non elevare la tensione per mezzo di pompe di carica o moltiplicatori di tensione.

Nota

La flessibilità della scheda EasyUSB e il Power Oring richiede particolare attenzione al fine di evitare che alimentazioni differenti vengano connesse in parallelo. In particolare qualora non si faccia uso dei PMOS Q3, Q4 e Q5, è bene impostare i pin del PIC come ingressi o se impostati come uscite impostare RD6 a 0 mentre RD5 e RD4 a 1, in maniera da disattivare tutti i PMOS.

E' bene in ultimo mettere nuovamente in evidenza che il controllo dei PMOS Q4 e Q5 deve essere fatto in maniera opportuna al fine di evitare che entrambi i PMOS siano attivi in contemporanea.

La presenza del condensatore C11 permette di mantenere l'uscita stabile al passaggio da un'alimentazione ad un altra.

Limiti di corrente

La complessità dello stadio di alimentazione permette di raggiungere la flessibilità sopra descritta, ma questo richiede di tenere a mente diversi limiti di corrente, visto i vari percorsi che la corrente può compiere per alimentare EasyUSB e i dispositivi ad essa collegati per mezzo del connettore di espansione. Di seguito sono riportati vari casi e limiti:

- **Alimentazione solo da USB**

  Quando il sistema è alimentato solo da USB, deve assorbire al suo avvio non più di 100mA. Qualora si abbia bisogno di correnti maggiori è necessario richiederlo, e la richiesta deve essere approvata dall'Host o Hub. Solo dopo l'approvazione è possibile erogare la corrente richiesta fino al massimo di 500mA. Tale corrente rappresenta quella utilizzabile dal sistema EasyUSB e dalle periferiche connesse al connettore EX1. In questo caso la tensione 9V-12V non è presente su EX1 visto che non è presente l'alimentatore esterno.

- **Alimentazione da USB e da alimentatore esterno**

  Quando il sistema è alimentato da entrambe le alimentazioni al sistema non è comunque permesso di assorbire dalla USB più di 100mA alla sua connessione, fino a quando non viene approvato un nuovo budget di corrente. Qualora per alimentare il sistema connesso su EX1 siano richieste correnti maggiori di 500mA per la tensione di 5V, è possibile attivare il PMOS Q4 (disattivando preventivamente, Q5). La corrente massima prelevabile dall'alimentatore di 5V è di 1A incluso l'assorbimento del sistema EasyUSB qualora Q2 sia attivo e Q1 disattivo.

- **Utilizzo della tensione dell'alimentatore**

  La tensione dell'alimentatore 9V-12V può naturalmente essere utilizzata solo se l'alimentatore di rete è presente. La massima corrente che è possibile prelevare dal connettore EX1 facendo uso dell'alimentazione proveniente da PW_IN è di 1A[19]. Questo limite discende da limiti termici visto che i PMOS non possiedono alette di raffreddamento. Le piste per la sezione di alimentazione legata a PW_IN sono dimensionate per correnti, a temperatura ambiente, fino a 2A ma è bene tenersi entro i limiti di 1A per la quale la scheda EasyUSB è specificata per operare senza problemi.

Nota

Qualora si abbiano esigenze particolari in termini di risposta in frequenza nei confronti di variazioni di carico esterno alla scheda EasyUSB, potrebbe essere richiesta l'aggiunta di condensatori di filtro e di bulk al fine di migliorare la risposta in frequenza dell'alimentatore presente a bordo della scheda EasyUSB.

[19] Naturalmente l'alimentatore esterno deve essere idoneo per erogare tale corrente.

Pulsanti

La scheda EasyUSB possiede 2 pulsanti, comodamente accessibili, che possono essere utilizzati per applicazioni generiche. Nonostante la semplicità di questo hardware, la loro presenza permette di semplificare e accelerare lo sviluppo di nuove applicazioni. Infatti non è necessario avere fili volanti né tanto meno fare saldature. Lo schema elettrico relativo ai pulsanti è riportato in Figura 6.

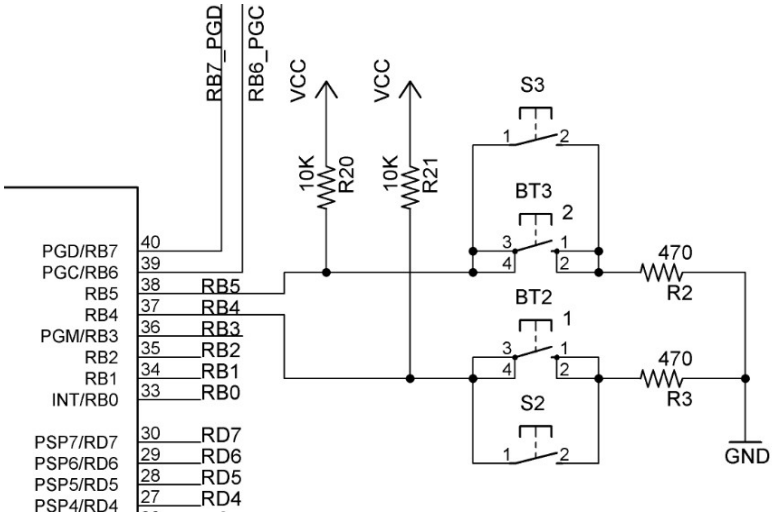


Figura 6: *Schema elettrico dei pulsanti*

Si può osservare che i pulsanti sono collegati rispettivamente agli ingressi RB4-RB5. La scelta di questi pin discende dal fatto che la PORTB possiede linee d'interrupt sulla variazione del valore logico del pin ed inoltre il Bootloader della Microchip possiede di default il controllo del pin RB4 al fine di verificare la necessità o meno di avviare il Bootloader. Un'altra peculiarità della PORTB è che possiede resistori di pull-up interni, dunque si possono risparmiare i resistori di pull-up esterni. Diversamente da Freedom II, si è preferito utilizzare in ogni modo dei resistori di pull-up in maniera da rendere il sistema compatibile con la scheda di sviluppo USB della Microchip (PICDEM™ FS USB).

I resistori R2, R3 risultano fondamentali per la protezione del PIC nel caso in cui si stiano utilizzando i pin RB4-RB5 come output invece di input. In questa situazione la pressione di un tasto potrebbe cortocircuitare verso massa l'uscita, causandone la rottura. Le resistenze R2, R3 evitano che questo possa accadere, limitando la corrente di cortocircuito.

Sul PCB è previsto sia il montaggio orizzontale che il montaggio verticale dei pulsanti. Il montaggio orizzontale risulta più pratico in fase di sviluppo mentre, qualora il sistema venga integrato in una scatola di montaggio, utilizzare pulsanti verticali può risultare più pratico, visto che sono posti al lato del PCB stesso[20]. Questo accorgimento spiega perché siano presenti i pulsanti BT2 e BT3 rispettivamente in parallelo ai pulsanti S2 e S3. Sulla serigrafia vista la vicinanza dei vari pulsanti è riportata solo la nomenclatura dei pulsanti S2 e S3. Anche il tasto di Reset può essere montato sia orizzontalmente che verticalmente.

[20] Il connettore USB e J1 sporgono dal PCB per poter agevolare ulteriormente l'inserimento di EasyUSB all'interno di una scatola di montaggio.

Altre periferiche

Oltre alle periferiche appena descritte, il sistema EasyUSB possiede altre piccole periferiche che è possibile sfruttare nelle proprie applicazioni.

- Il tasto di RESET BT1 permette di resettare il sistema qualora si sia verificato uno stallo dello stesso o si voglia semplicemente riavviare il sistema per altre ragioni, quale per esempio riprogrammare il dispositivo per mezzo del Bootloader. Tale pulsante è individuabile per mezzo della serigrafia RESET presente sul PCB. Il tasto di Reset è protetto da sovracorrenti per mezzo del resistore R6. Questo permette di proteggere il programmatore nel caso in cui durante la programmazione venisse premuto inavvertitamente il tasto di Reset. Infatti non tutti i programmatori sono protetti contro i corto circuiti.

- Power LED, per la segnalazione della presenza dell'alimentatore di rete. Il LED5, nominato *DC PW* diversamente dagli altri presenti sulla scheda non è controllato dal PIC; la ragione è legata al fatto che, essendo collegato un alimentatore esterno, le problematiche associate al risparmio sono ridotte.

- Nonostante EasyUSB sia progettata in maniera da ridurre al minimo l'Hardware superfluo e poter quindi essere utilizzata come scheda di espansione senza costi eccessivi e ingombri inutili, possiede 4 LED ad uso generico. I LED possono essere abilitati o disabilitati per mezzo del Jumper JP2 nominato LED. Ponendo il Jumper sulla posizione 1-2 i LED risultano attivi e controllabili dal PIC per mezzo dei rispettivi pin RD0-RD3. Chiudendo il Jumper tra i pin 2-3 i LED risultano disattivi. Nonostante i LED siano per applicazioni generiche il LED4 è posizionato in prossimità del LED5 utilizzato per la segnalazione della presenza dell'alimentatore di rete. La ragione è legata al fatto che il LED4 (individuato dalla serigrafia BUS PW) è consigliabile usarlo per la segnalare quando l'alimentazione proviene dalla porta USB. Il LED4 è di color rosso mentre i LED1-LED3 sono di colore verde.

Connettore di espansione

Il sistema EasyUSB possiede a bordo l'hardware essenziale per ottenere periferiche USB di base. Hardware esterno può essere collegato per mezzo del connettore di espansione EX1 compatibile con il sistema Freedom II. In Figura 7 è riportata la piedinatura.

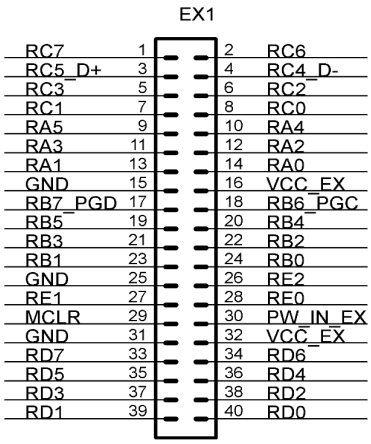


Figura 7: *Connettore di espansione EX1.*

Il progetto PJ7007[21] riportato in Figura 8, può essere utilizzato come scheda di espansione per i vari prototipi USB.

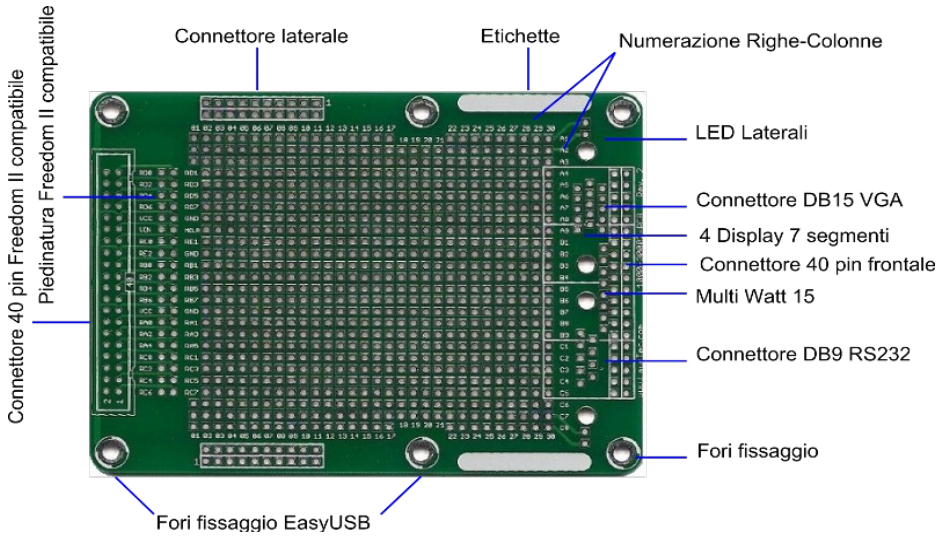


Figura 8: *Scheda di prototipizzazione per la scheda EasyUSB (compatibile Freedom II)*

Oltre progetto PJ7007 è disponibile anche la scheda di espansione dedicata a EasyUSB, ovvero il progetto PJ7009. Il vantaggio della scheda di espansione EasyUSB è legato al fatto che questa è delle stesse dimensioni di EasyUSB, permettendo di ottenere prototipi compatti. Un esempio di montaggio è riportato in Figura 9. Maggiori dettagli sul progetto PJ7009 possono essere trovati nella scheda tecnica

[21] La scheda di espansione PJ7007 può essere richiesta alla sezione servizi in forma di KIT. Il KIT include tutti i connettori e cavo di connessione con la scheda EasyUSB.

dedicata[22].

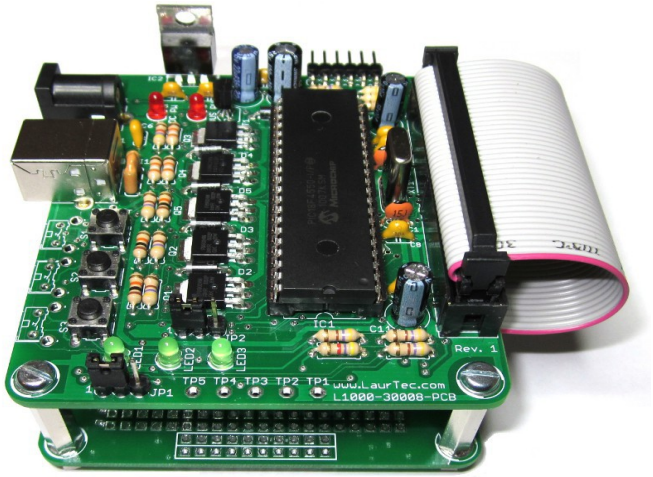


Figura 9: *Scheda di espansione EasyUSB.*

Si fa notare che il connettore EX1, oltre a tutti i pin del PIC porta in uscita GND, +5V e VIN+, permettendo così di alimentare direttamente l'hardware esterno. Come spiegato nel paragrafo relativo all'alimentatore le varie alimentazioni non sono dirette ma controllate per mezzo di transistor PMOS al fine di poter ridurre la corrente assorbita dal sistema. E' a cura dell'utilizzatore accertarsi che il consumo dell'hardware esterno rientri nei limiti di potenza ammessi dalla scheda EasyUSB e dalle specifiche USB 2.0.

In Figura 10 è riportata la scheda di espansione PJ7011 che possiede una breadboard con pulsanti e LED al fine di poter aggiungere altri componenti discreti senza dover effettuare alcuna saldatura. Maggiori informazioni sulla scheda di espansione possono essere trovati nella documentazione ufficiale PJ7011-IT scaricabile dal sito LaurTec[23].

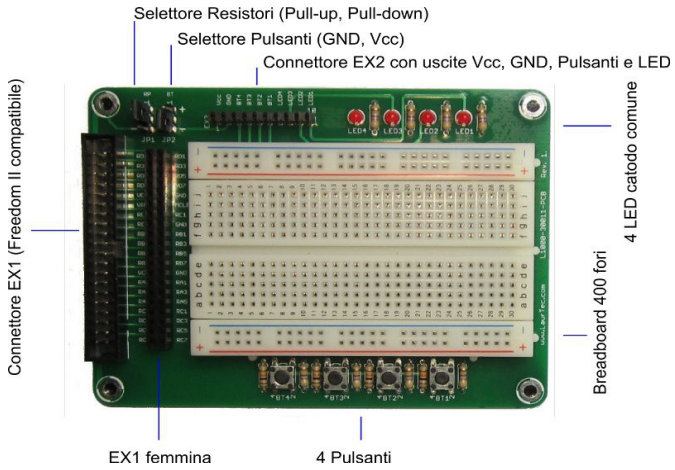


Figura 10: *Scheda di espansione PJ7011.*

[22] La scheda di espansione PJ7009 può essere richiesta alla sezione servizi in forma di KIT. Il KIT include tutti i connettori e cavo di connessione con la scheda EasyUSB.

[23] La scheda di espansione PJ7011 può essere richiesta alla sezione servizi in forma di KIT. Il KIT include tutti i connettori e cavo di connessione con la scheda EasyUSB.

Connettore di Programmazione e Debug

Un'altra importante caratteristica del sistema EasyUSB è la sua integrazione con gli strumenti Microchip. In particolare il suo connettore di programmazione e debug è conforme alle specifiche utilizzate per i programmatori PICKIT, ICD e loro derivati. Il connettore utilizzato è in particolare la versione del connettore lineare a 6 pin come riportato in Figura 11[24]. I dettagli sulle connessioni necessarie al programmatore sono riportate in Figura 12.

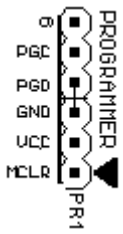


Figura 11: *Serigrafia del connettore del programmatore/debug*

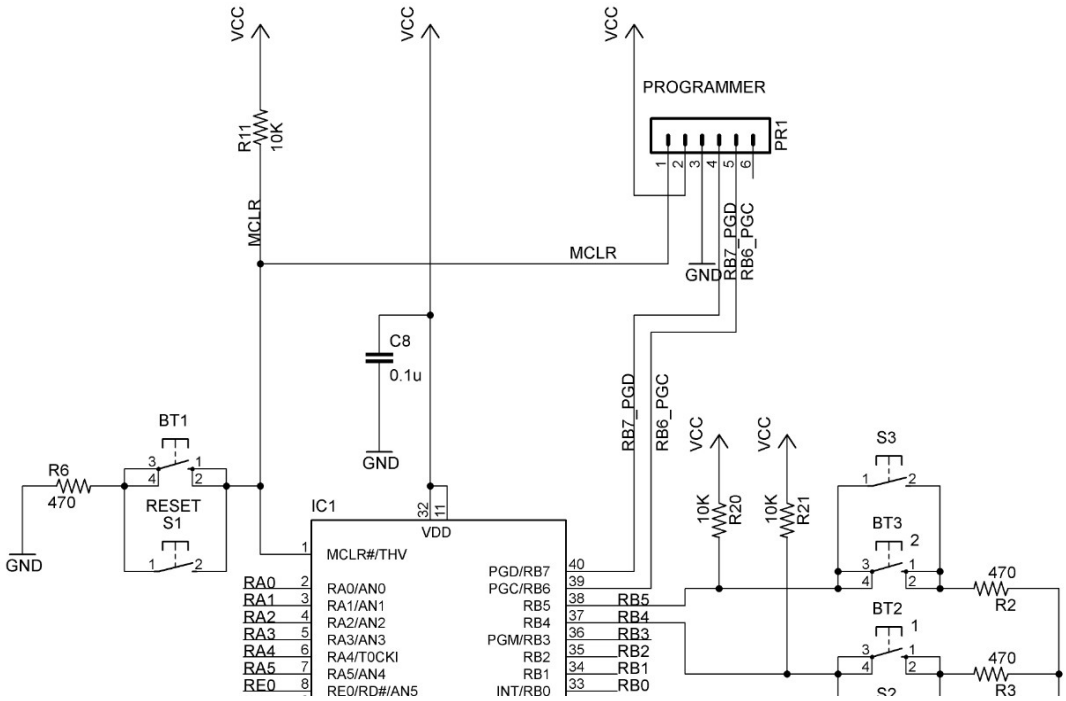


Figura 12: *Connessioni del programmatore/debug*

Il fatto di utilizzare il pin-out compatibile con i programmatori Microchip permette d'integrare facilmente il sistema con l'ambiente di sviluppo Microchip, MPLAB®. Questo porta anche il vantaggio di avere sempre il supporto per gli ultimi modelli dei PIC sempre a portata di mouse!

Si ricorda che la scheda EasyUSB oltre che poter essere programmata con il programmatore può essere programmata anche per mezzo dell'USB Bootloader (maggiori dettagli sono riportati nei paragrafi seguenti).

Nota:

Durante la fase di programmazione è bene non toccare la scheda.

[24] Si raccomanda l'utilizzo di un connettore a 90 gradi in maniera da collegare il programmatore senza problemi.

Layout Periferiche

Durante la fase di sviluppo risulta di fondamentale importanza una guida rapida del Layout utilizzato per le varie periferiche. Sebbene questo possa essere ricavato anche dallo schema elettrico, aver a disposizione una tabella in cui sono riportati direttamente i pin può risultare più utile. Si ricorda che a seconda delle scelte progettuali alcune parti della scheda potrebbero non essere disponibili, in particolar modo se si dovesse far uso di hardware esterno.

Pin	Periferica	Altre Funzioni
RA0	Disponibile	I/O o in. analogico
RA1	USB_PW_SENS	I/O vincolato
RA2	SELF_PW_SENS	I/O vincolato
RA3	Disponibile	I/O o in. analogico
RA4	Disponibile	I/O o in. TOKI
RA5	Disponibile	I/O o in. analogico
RB0	Disponibile	SDA per bus I2C in PIC con USB;
RB1	Disponibile	SCL per bus I2C in PIC con USB;
RB2	Disponibile	I/O o INT
RB3	Disponibile	I/O
RB4	Pulsante BT1	I/O vincolato
RB5	Pulsante BT2	I/O vincolato
RB6	Disponibile	I/O, INT su PORTB
RB7	Disponibile	I/O, INT su PORTB
RC0	Disponibile	I/O
RC1	Disponibile	I/O, PWM 2
RC2	Disponibile	I/O o PWM
RC3	Condensatore LDO	I/O vincolato
RC4	D- in PIC con USB.	I/O vincolato
RC5	D+ in PIC con USB.	I/O vincolato
RC6	Disponibile	I/O, linea TX per trasmissioni RS232
RC7	Disponibile	I/O, linea RX per trasmissioni RS232
RD0	LED1	I/O se non si usano i LED
RD1	LED2	I/O se non si usano i LED
RD2	LED3	I/O se non si usano i LED
RD3	LED4	I/O se non si usano i LED
RD4	Gate Q5	I/O vincolato
RD5	Gate Q4	I/O vincolato
RD6	Gate Q3	I/O vincolato
RD7	Gate Q2	I/O vincolato
RE0	Disponibile	I/O o in. analogico
RE1	Disponibile	I/O o in. analogico
RE2	Disponibile	I/O o in. analogico

Tabella 3: *Tabella riassuntiva delle connessioni tra il PIC e le periferiche di sistema. Per una visione completa delle funzionalità di ogni pin si rimanda al datasheet del PIC utilizzato. In Tabella sono riportate solo le funzioni principali dei pin.*

Istruzioni per il montaggio

EasyUSB è un sistema realizzato su PCB a doppia faccia. Il PCB può essere richiesto alla sezione servizi del sito www.LaurTec.it per mezzo di semplice donazione di supporto. Il suo assemblaggio non risulta particolarmente complicato ma necessita certamente di attenzione.

Per semplificare il montaggio, il PCB è realizzato con serigrafia dei componenti e relativo nome. Lo schema di montaggio è riportato in Figura 13. Il PCB reale di EasyUSB è riportato in Figura 14.

Per il montaggio dei componenti è consigliabile seguire la regola legata all'altezza dei componenti stessi; dunque è bene iniziare dai resistori per poi passare ai transistor Q1-Q5 e ai diodi. Questa regola ha solamente un'utilità pratica associata al fatto che frequentemente, per fare le saldature, il PCB verrà posto sotto sopra. Per i primi componenti è bene accertarsi che il codice dei colori sia corretto mentre per i diodi è necessario che il verso d'inserzione sia rispettato secondo la serigrafia riportata in Figura 13. In particolare i diodi hanno un anello colorato che segnala la posizione del catodo, questo anello è riportato anche sulla serigrafia[25].

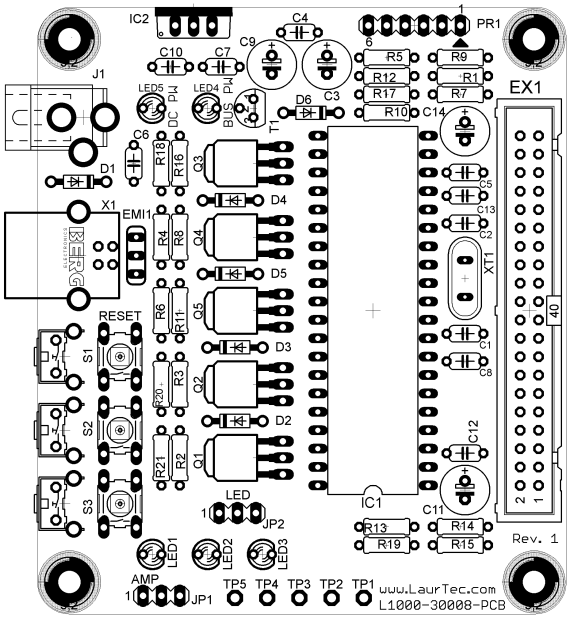


Figura 13: *Serigrafia di EasyUSB*

Successivamente si può procedere al montaggio delle capacità di filtro da 0.1uF e delle capacità ceramiche. Dal momento che il condensatore C5 da 0.47uF è molto simile ai condensatori da 0.1uF è bene isolarlo dagli altri e montarlo come primo condensatore. Si ricorda che l'ordine fin qui proposto non è obbligatorio ma può risultare pratico per il montaggio.

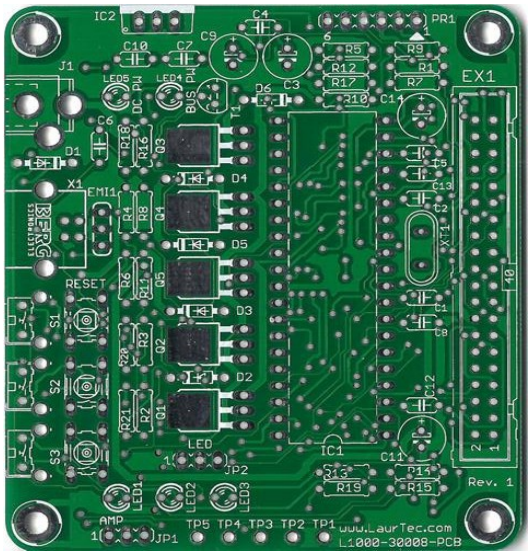


Figura 14: *PCB EasyUSB*

IC1 è bene montarlo su apposito zoccoletto in modo da poterlo cambiare ed evitare il suo danneggiamento in fase di saldatura. Gli integrati sono infatti sensibili alla temperatura, come d'altronde tutti i semiconduttori. Un'esposizione ad alte temperature può infatti portare alla rottura dell'integrato.

Ulteriore accorgimento va riservato per le capacità polarizzate, per le quali bisogna rispettare il verso legato alla polarizzazione. Sulla serigrafia del PCB di Figura 13 è facilmente individuabile il terminale positivo delle capacità indicato con un +. Se sul PCB non si dovesse ben leggere qualche carattere a causa di via[26] far sempre riferimento alla Figura 13.

A montaggio completato EasyUSB apparirà simile a Figura 15; infatti si possono avere differenze da montaggio a montaggio a seconda dei componenti che si vuole montare o dalla versione del circuito

[25] I diodi LED hanno il catodo segnalato da una smussatura sulla capsula del diodo stesso. Un altro modo per individuare l'anodo e il catodo è controllare la lunghezza dei terminali. L'anodo risulta il terminale più lungo.

[26] I via rappresentano i fori metallizzati che permettono la realizzazione dei PCB a doppia faccia. Il loro uso permette infatti ad un segnale di poter passare da un lato all'altro del PCB.

stampato stesso[27]. In ultimo al fine di montare storti i transistor Q1-Q5 si consiglia di saldare un solo pin e di controllare se il transistor è rimasto allineato. Avere componenti montati in maniera simmetrica renderà il montaggio più professionale. Si noti che la parte metallica dei transistor Q1-Q5 non deve essere necessariamente saldata sul PCB.

Maggiori dettagli sugli strumenti e tecniche per il montaggio dei KIT elettronici possono essere trovate nel Tutorial AN9001-IT scaricabile gratuitamente dal sito www.LaurTec.it .

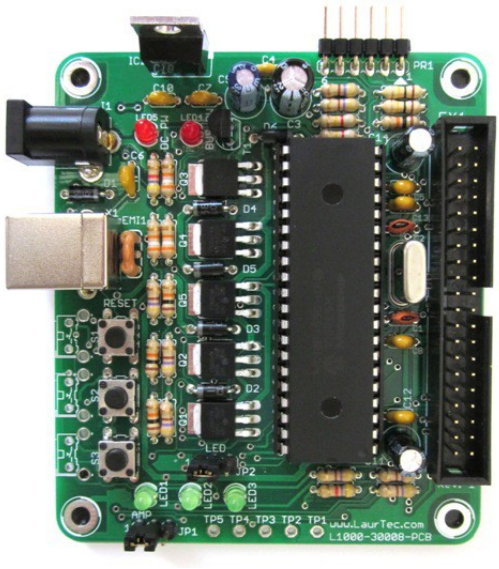


Figura 15: *EasyUSB a montaggio ultimato*

[27] Il circuito stampato può subire variazioni senza che il seguente articolo venga aggiornato.

USB Bootloader

La scheda EasyUSB è progettata per poter essere utilizzata come scheda di sviluppo compatibile con l'ambiente di sviluppo Microchip. In particolare i vari programmatori della famiglia PICKIT 2, PICKIT 3, ICD possono essere direttamente collegati per programmare il PIC18F4550 presente sulla scheda. Ciononostante al fine di agevolare lo sviluppo con il minimo necessario, la scheda richiesta in KIT possiede il PIC programmato con un Bootloader USB, scaricabile anche dal sito www.LaurTec.it alla sezione dedicata al progetto.

Cosa è un Bootloader

Il Bootloader rappresenta un programma che viene preventivamente caricato nel PIC ed il cui compito è quello di permettere di programmare il microcontrollore senza aver bisogno di alcun programmatore[28]. Tale funzionalità risulta molto pratica qualora si vogliano progettare sistemi in cui sia possibile aggiornare il firmware, sia per inserire nuove funzioni o miglioramenti che risolvere problemi derivanti da bug. In pratica il Bootloader sfrutta la caratteristica di programmabilità della memoria Flash on the fly, ovvero il programma in esecuzione nel PIC può riprogrammarsi. In particolare il Bootloader al fine di programmare il PIC deve avere come ingresso il nuovo programma da scrivere. Il nuovo programma o versione di programma può essere fornito usando un qualunque bus di comunicazione previsto dal microcontrollore o anche usare i pin standard.

Tra le caratteristiche principali dei Bootloader vie è quella di essere piccoli e praticamente inesistenti per l'applicazione principale. Il Bootloader viene infatti eseguito, generalmente, solo all'avvio del microcontrollore, se si accorge che è richiesto un aggiornamento del programma (a breve vederemo come) il Bootloader inizia la comunicazione con la periferica o sistema che gli invierà il nuovo programma, altrimenti cessa la propria esecuzione ed avvia il programma principale, che disporrà dell'intero microcontrollore come se il Bootloader non esistesse (eccetto che per la memoria Flash utilizzata).

In particolare la scheda EasyUSB viene fornita con un Bootloader USB ovvero la cui programmazione ed aggiornamento del PIC avviene per mezzo della porta USB[29].

Esempio d'uso del Bootloader

Il Bootloader fornito con EasyUSB viene eseguito solo all'avvio del PIC, ovvero dopo un Reset del sistema, infatti il programma è caricato subito dopo il Reset Vector[30]. Una volta avviato il Bootloader controlla subito se ha ragione di essere eseguito o meno, controllando il valore del pulsante S2[31]. Se il pulsante è premuto, il sistema EasyUSB entra in modalità Bootloader, attendendo che il PC invii il nuovo programma.

Per caricare il nuovo programma bisogna far uso del Programma *EasyUSB Bootloader* come riportato in Figura 16. Il programma è scaricabile dal sito www.LaurTec.it alla sezione dedicata al progetto EasyUSB[32]. Anche in questo caso l'applicazione è quella standard fornita dalla Microchip nel

[28] Da quanto detto si capisce che la prima programmazione avviene però per mezzo di un programmatore.

[29] Il Bootloader utilizzato è quello fornito dalla Microchip stessa ed opportunamente modificato per la scheda EasyUSB. In particolare è stato cambiato il PID e VID della periferica, che nel caso di EasyUSB è PID: 0xFC5D VID:0x04D8. Oltre al Bootloader USB la Microchip fornisce anche Bootloader RS232, CAN e attraverso il protocollo TCP/IP.

[30] Si rimanda al libro "C18 Step by Step" per maggiori informazioni sul Reset Vector.

[31] Condizioni di avvio differenti potrebbero essere impostate cambiando il codice sorgente del Bootloader USB fornito dalla stessa Microchip.

[32] Il programma per poter essere eseguito necessita del .NET Framework 4.0. Inoltre se non già presenti è anche necessario installare le librerie ridistribuibili per VC++ scaricabili dal sito della Microsoft alla sezione "Microsoft Visual C++ 2010 Redistributable Package" ed installare la versione a 32 o 64 bit a seconda del proprio PC.

progetto *USB HID Bootloader* opportunamente modificato per riconoscere la periferica EasyUSB, ovvero è stato modificato il PID e VID della periferica HID[33] da riconoscere. In Figura 16 è possibile vedere che fintanto che il sistema EasyUSB non è collegato o non è in modalità Bootloader, l'applicazione richiede che il sistema EasyUSB venga messo in tale modalità, e qualora lo sia già deve essere collegato al PC.

Se il sistema non è in modalità Bootloader per farlo entrare in modalità Bootloader basterà premere il tasto Reset tenendo premuto anche il pulsante S2, lasciare poi il tasto Reset tenendo premuto S2. La pressione di S2 verrà rilevata all'avvio del Bootloader che rimarrà dunque in esecuzione. La modalità Bootloader è riconoscibile dal fatto che i LED1 si accende.

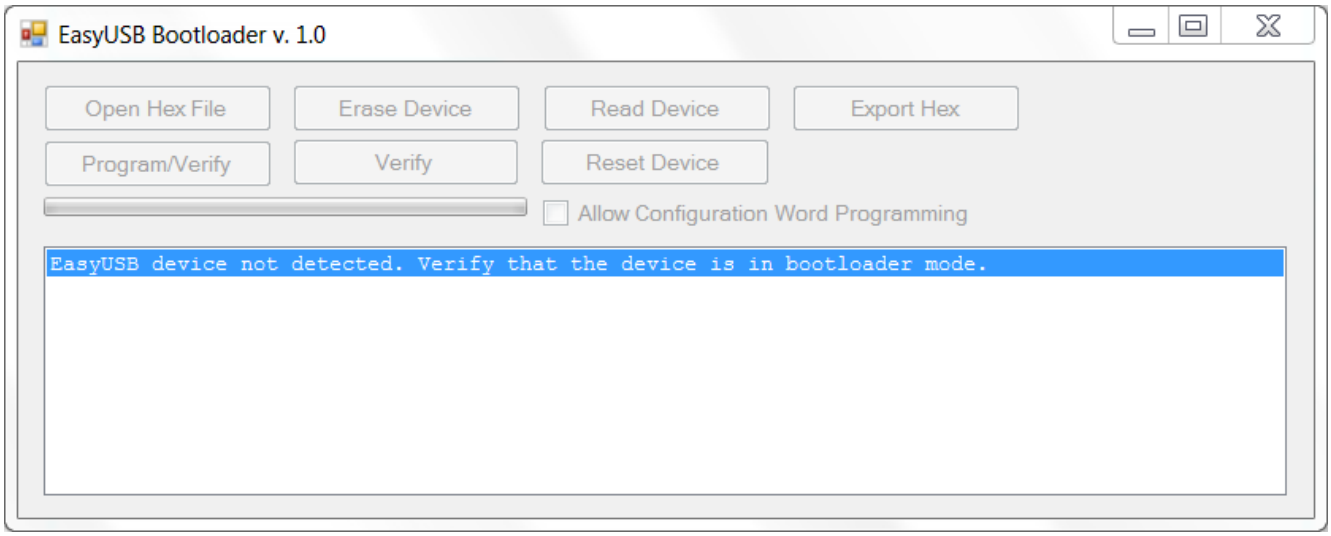


Figura 16: *Applicazione EasyUSB Bootloader*

Quando il sistema EasyUSB viene collegato al PC e viene riconosciuto in modalità Bootloader, il LED1 e LED2 si accendono in maniera alternata. A questo punto l'applicazione *EasyUSB Bootloader* visualizza il nuovo messaggio come riportato in Figura 17.

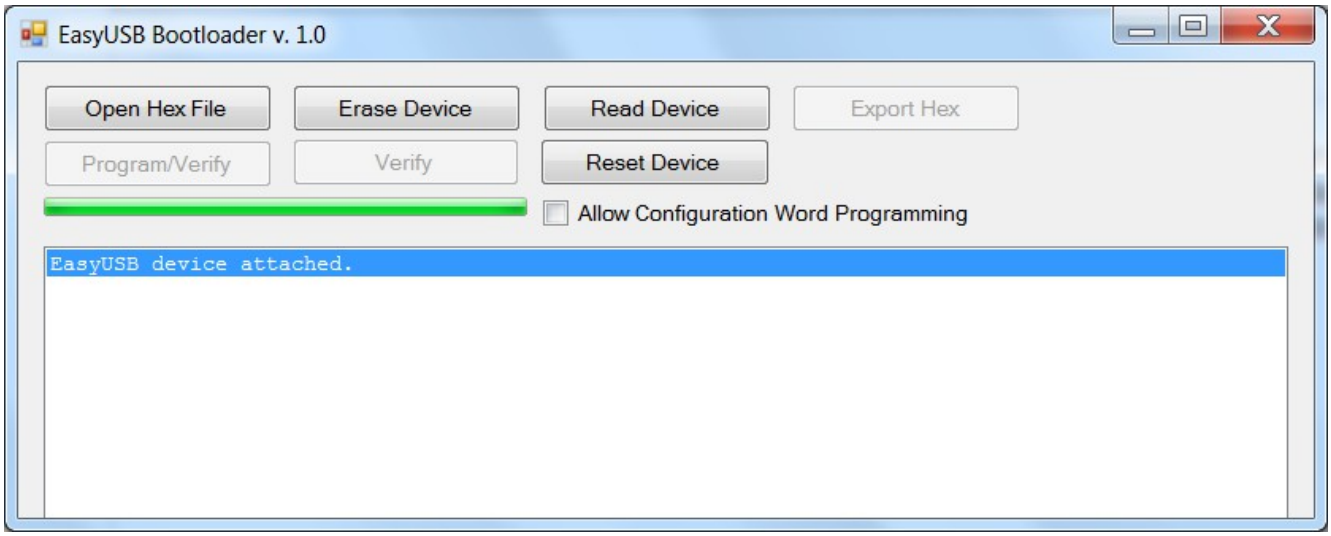


Figura 17: *Sistema EasyUSB riconosciuto in modalità Bootloader.*

[33] Il Bootloader viene riconosciuto come periferica HID.

A questo punto senza necessitare di alcun programmatore è possibile caricare il nostro programma compilato, ovvero il nostro file .hex. Per fare questo basterà aprire il file per mezzo del tasto *Open Hex File* e premere poi il tasto *Program/Verify*, che si attiverà dopo aver caricato il file.

Il check box presente per abilitare o meno la possibilità di sovrascrivere la configurazione del PIC discende dal fatto che il Bootloader per funzionare richiede di una specifica configurazione che non deve essere cambiata, al fine di non compromettere la sua funzionalità, in particolare non devono essere alterate le configurazioni relative al Clock e al PLL. Dal momento che ogni sistema possiede una propria esigenza per la configurazione del PIC, potrebbe essere necessario sovrascrivere la configuration word del Bootloader, per cui il check box normalmente disattivo deve essere abilitato. Nonostante si possa sovrascrivere la configuration word si deve avere la precauzione di non compromettere la funzionalità del Bootloader stesso.

Programmare per il Bootloader

A questo punto devo ammettere di aver mentito sapendo di mentire. Il Bootloader se presente altera leggermente il programma che andremo a scrivere. Per esempio gli esempi proposti sul testo *"C18 Step by Step"*, non funzionerebbero con il Bootloader. La ragione risiede nel fatto che il Bootloader è scritto subito dopo il Reset Vector e anche gli esempi del testo "*C18 Step by Step"* sono scritti per risiedere subito dopo il Reset Vector; dunque si capisce che i programmi scritti normalmente tenderebbero a sovrascrivere il Bootloader impedendo una nuova esecuzione dello stesso.

Sembrerebbe allora che tutti gli esempi presentati nel testo "*C18 Step by Step"* abbiano un problema...ma in realtà questo non è un problema se si fa uso di un programmatore esterno e non si voglia avere la funzionalità offerta da un Bootloader di aggiornare il programma stesso.

Naturalmente arrivati a questo punto bisogna vedere come risolvere la situazione, ovvero scrivere programmi che possano funzionare con il Bootloader.

Se avete già letto il testo *"C18 Step by Step"* saprete già che una volta selezionato il PIC non si ha la necessità di inserire il file per il linker, poiché questo verrà inserito in automatico dal compilatore. Questo, come detto nel testo è valido fino a quando non si abbia la necessità di riorganizzare la memoria interna del PIC. L'utilizzo di un Bootloader è uno di quei casi in cui è necessario riorganizzare la memoria interna, permettendo a due applicazioni di risiedere all'interno di uno stesso PIC, infatti il Bootloader altro non è che una seconda applicazione.

Il file di Linker può essere realizzato modificando il file di linker originale, ma dal momento che il file originale può servire anche in altre applicazioni, è bene in realtà fare una copia del file e modificare quest'ultima. Nel caso del Bootloader della Microchip che risiede nelle prime 0xFFF locazioni di memoria flash, è possibile far uso direttamente del file script offerto dalla Microchip stessa, ovvero del file *rm18f4550 – HID Bootloader.lkr*. Questo file come visibile dal nome è utilizzabile solo per il PIC18F4550 ma confrontandolo con altri file di Linker è facile apportare le modifiche in altri file.

Per semplicità l'esempio presentato di seguito è scaricabile come progetto dal sito www.LaurTec.it in particolare il progetto può essere utilizzato come piccolo Framework per realizzare semplici programmi che debbano essere caricati nella scheda EasyUSB per mezzo del Bootloader. L'esempio presentato contiene al suo interno il file di Linker necessario per il PIC18F4550.

Vediamo come creare il nostro Framework. Come primo passo dopo aver creato un progetto per PIC18F4550 si deve procedere con l'inserimento del file di Linker e di tutti i file sorgenti che verranno utilizzati, come riportato in Figura 18.

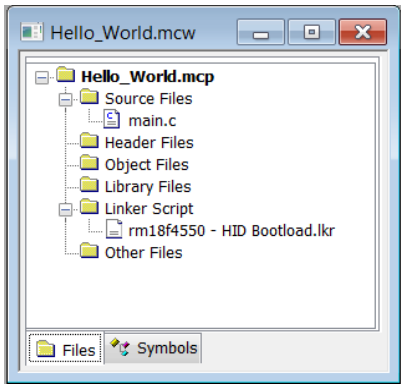


Figura 18: *Inserimento nel progetto del Linker File (Script)*

Per ragioni di chiarezza sotto è riportato il file di Linker utilizzato nel Framework. È possibile vedere che la memoria assegnata per il Bootloader è dall'indirizzo 0x000 a 0xFFF ed è definita *Protected*, ovvero il programma non può farne uso.

```
// File: rm18f4550 - HID Bootload.lkr

// Use this linker for the USB application that will be self programmed by the HID
bootloader.
// The HID bootloader project itself uses the BootModified.18f4550.lkr file
instead.

// THIS LINKER SCRIPT HAS BEEN MODIFIED...  This version is intended to be used
// with the "PROGRAMMABLE_WITH_USB_HID_BOOTLOADER" bootloader.  The HID
// bootloader occupies memory ranges 0x000-0xFFF.  In order for the code generated
// by this project to work with the bootloader, the linker must not put any code
// in the 0x00-0xFFF address range.

// This linker script was originated from the 18f4550.lkr file provided by
// the MCC18 distribution.

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f4550.lib

CODEPAGE   NAME=bootloader START=0x0               END=0xFFF          PROTECTED
CODEPAGE   NAME=vectors    START=0x1000            END=0x1029         PROTECTED
CODEPAGE   NAME=page       START=0x102A            END=0x7FFF
CODEPAGE   NAME=idlocs     START=0x200000          END=0x200007       PROTECTED
CODEPAGE   NAME=config     START=0x300000          END=0x30000D       PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE          END=0x3FFFFF       PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000          END=0xF000FF       PROTECTED

ACCESSBANK NAME=accessram  START=0x0            END=0x5F
DATABANK   NAME=gpr0       START=0x60           END=0xFF
DATABANK   NAME=gpr1       START=0x100          END=0x1FF
DATABANK   NAME=gpr2       START=0x200          END=0x2FF
DATABANK   NAME=gpr3       START=0x300          END=0x3FF
DATABANK   NAME=usb4       START=0x400          END=0x4FF          PROTECTED
DATABANK   NAME=usb5       START=0x500          END=0x5FF          PROTECTED
DATABANK   NAME=usb6       START=0x600          END=0x6FF          PROTECTED
DATABANK   NAME=usb7       START=0x700          END=0x7FF          PROTECTED
ACCESSBANK NAME=accesssfr  START=0xF60          END=0xFFF          PROTECTED

SECTION    NAME=CONFIG     ROM=config
```

```
STACK SIZE=0x100 RAM=gpr3

SECTION        NAME=USB_VARS    RAM=usb4
```

Dopo aver chiarito l'esigenza del Linker è possibile iniziare a scrivere il programma sorgente, che nel caso del nostro piccolo Framework è già strutturato. Il programma di esempio permetterà di accendere il LED1 e LED2 rispettivamente premendo i pulsanti S2 e S3. Il programma è molto semplice ma richiede molte linee di codice...fortunatamente già scritte! Se si è nuovi al linguaggio C18 si consiglia prima di leggere o meglio iniziare a leggere il testo *"C18 Step by Step"*. Se si ha già esperienza di programmazione con programmi in cui si fa uso delle interruzioni, il programma che segue non dovrebbe spaventare:

```
#include <p18f4550.h>

#pragma config FOSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config PBADEN = OFF

//OSC = HS        Impostato per lavorare ad alta frequenza
//WDT = OFF       Disabilito il watchdog timer
//LVP = OFF       Disabilito programmazione LVP
//PBADEN = OFF    Disabilito gli ingressi analogici


void New_High_Priority_ISR_Code();
void New_Low_Priority_ISR_Code();

//*************************************************************
//            Definizione nuovi indirizzi
//*************************************************************

#define REMAPPED_RESET_VECTOR_ADDRESS            0x1000
#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS   0x1008
#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS    0x1018

//Vedi il file c018i.c
extern void _startup (void);


//*************************************************************
//            Rimappatura Interrupt Service Routine
//*************************************************************

#pragma code REMAPPED_RESET_VECTOR = REMAPPED_RESET_VECTOR_ADDRESS
void _reset (void){
  _asm goto _startup _endasm
}

#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR = REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void) {

  _asm goto New_High_Priority_ISR_Code _endasm
}

#pragma code REMAPPED_LOW_INTERRUPT_VECTOR = REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void) {

  _asm goto New_Low_Priority_ISR_Code _endasm
```

```
}

//************************************************************
//            Standard Interrupt Service Routine
//************************************************************

#pragma code HIGH_INTERRUPT_VECTOR = 0x08
void High_ISR (void) {

  _asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS _endasm
}


#pragma code LOW_INTERRUPT_VECTOR = 0x18
void Low_ISR (void) {
  _asm goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS _endasm
}

#pragma code


//************************************************************
//         Nuovi Interrupt Service Routines
//************************************************************

#pragma interrupt New_High_Priority_ISR_Code
void New_High_Priority_ISR_Code() {
  // Gestione Interrupt alta Priorita'
}


#pragma interruptlow New_Low_Priority_ISR_Code
void New_Low_Priority_ISR_Code(){
  // Gestione Interrupt bassa Priorita'
}


//************************************************************
//              Funzione main ()
//************************************************************
void main (void){

  // 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 tutti ingressi e RD0-RD1 come uscite
  LATD = 0x00;
  TRISD = 0b11111100;

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

```
  // Ciclo infinito
  while (1) {

      // Controllo Switch S1
      if (PORTBbits.RB4 == 0) {
            LATDbits.LATD0 = 0x01;
      } else {
            LATDbits.LATD0 = 0x00;
      }

      // Controllo Switch S2
      if (PORTBbits.RB5 == 0) {
            LATDbits.LATD1 = 0x01;
      } else {
            LATDbits.LATD1 = 0x00;
      }
  }
}
```

...ok, credo che vi siete spaventati!
Vediamo i vari blocchi che permettono a questo programma di non interferire con il Bootloader. Il programma inizia come ogni programma standard dunque nulla di nuovo. Successivamente viene la definizione dei nuovi indirizzi che verranno utilizzati dalla nostra applicazione.

```
//************************************************************
//            Definizione nuovi indirizzi
//************************************************************

#define REMAPPED_RESET_VECTOR_ADDRESS            0x1000
#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS   0x1008
#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS    0x1018
```

Si noti che il nuovo Reset Vector non è più a 0x0000 ma a 0x1000, ovvero subito dopo la parte di memoria assegnata al Bootloader. Altri indirizzi che vengono variati sono quelli dei vettori delle interruzioni, in particolare quello dell'Interrupt a bassa e ad alta priorità. Si noti che anche questi vettori, normalmente posizionati all'indirizzo 0x0008 e 0x0018 risultano traslati di 0x1000 locazioni[34]. I `define` ora scritti non sono però sufficienti a traslare effettivamente i vettori, quanto scritto è solo un modo pratico per avere in un solo punto gli indirizzi utilizzati nel nostro Remapping (come si dice in inglese) ovvero rimappatura della memoria programma.

Dopo la definizione dei nuovi indirizzi si possono impostare le chiamate alle funzioni per la gestione delle interruzioni, in particolare le chiamate vengono poste proprio agli indirizzi ridefiniti sopra.

```
//************************************************************
//            Rimappatura Interrupt Service Routine
//************************************************************

#pragma code REMAPPED_RESET_VECTOR = REMAPPED_RESET_VECTOR_ADDRESS
void _reset (void){
  _asm goto _startup _endasm
}

#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR = REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void) {
```

[34] L'aver traslato di 0x1000 discende solo dal fatto che si è fatto uso del Bootloader della Microchip. Facendo uso di altri Bootloader questo numero potrebbe essere più grande come anche più piccolo.

```
   _asm goto New_High_Priority_ISR_Code _endasm
}

#pragma code REMAPPED_LOW_INTERRUPT_VECTOR = REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void) {

   _asm goto New_Low_Priority_ISR_Code _endasm
}
```

Le chiamate alle funzioni di gestione vengono effettuate per mezzo di un semplice `goto`, ovvero con un salto assoluto all'indirizzo identificato dalla nostra funzione. In particolare all'indirizzo 0x1000 viene effettuata una chiamata alla funzione `_startup`, ovvero quella funzione che tacitamente viene sempre chiamata dal compilatore per gestire il Reset del PIC. Quanto appena fatto sembrerebbe sufficiente per il Remapping del vettore di Reset e delle interruzioni. Effettivamente per il vettore di Reset è finita ma non per le interruzioni ad alta e bassa priorità. Il Bootloader, pur risiedendo nelle prime 0x1000 locazioni di memoria non altera i vettori di Interrupt, dunque in caso di Interrupt, il Program Counter verrà indirizzato ai vettori delle Interruzioni standard, per tale ragione bisogna impostare un salto dai vettori standard ai nuovi vettori delle Interruzioni. Quanto detto viene fatto dalla seguente sezione di codice :

```
//******************************************************************
//            Standard Interrupt Service Routine
//******************************************************************

#pragma code HIGH_INTERRUPT_VECTOR = 0x08
void High_ISR (void) {

   _asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS _endasm
}


#pragma code LOW_INTERRUPT_VECTOR = 0x18
void Low_ISR (void) {
   _asm goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS _endasm
}

#pragma code
```

Dopo aver impostato i salti ai nuovi vettori delle Interruzioni, possono essere definite le funzioni per la gestione delle Interruzioni, ma che nel nostro caso sono vuote poiché non utilizzate.

```
//******************************************************************
//         Nuovi Interrupt Service Routines
//******************************************************************

#pragma interrupt New_High_Priority_ISR_Code
void New_High_Priority_ISR_Code() {
   // Gestione Interrupt alta Priorita'
}


#pragma interruptlow New_Low_Priority_ISR_Code
void New_Low_Priority_ISR_Code(){
   // Gestione Interrupt bassa Priorita'
}
```

Infine, scritto tutto questo codice, possiamo scrivere la nostra funzione `main ()`. Assumendo che abbiate già un minimo di esperienza nello scrivere programmi non spiegherò quanto scritto nel main!

Nella funzione main, grazie alla struttura e riorganizzazione effettuata dal file di Linker e dal codice scritto, è possibile scrivere i semplici esempi del testo C18 Step by Step, ricompilare il progetto e caricare il file .hex generato dalla compilazione del programma per mezzo del Bootloader, ovvero dell'applicazione *EasyUSB Bootloader*.

Questo è il prezzo da pagare per poter scrivere un programma e caricarlo all'interno del PIC senza far uso di un programmatore e avere la funzione di poter aggiornare il programma senza avere necessità che il cliente abbia un programmatore. Se pensate che il gioco non valga la candela pensate se il vostro cellulare, decoder, scheda madre non avessero un bootloader che permettesse di aggiornare il sistema...credete forse di comprare dispositivi bug free!?

In ultimo è bene aggiungere una nota importante, il programma qui presentato non fa uso della porta USB, ma in realtà è possibile scrivere un'applicazione che oltre ad essere caricata nel PIC per mezzo della USB, faccia uso a sua volta della porta USB. Questo è possibile dal momento che il Bootloader fa uso di uno Stack USB separato, che quando non viene usato (Bootloader non attivo) non interferisce con altre applicazioni USB. Per scrivere applicazioni USB è possibile far uso del Framework offerto gratuitamente dalla stessa Microchip. Gli esempi presentati dalla Microchip sono già impostati per poter lavorare con il Bootloader qui presentato, semplicemente togliendo il commento alla linea di codice che definisce o meno l'utilizzo del Bootloader, che nel nostro caso è del tipo HID (Human Interface Device), naturalmente includendo il File di Linker opportuno.

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

Studiando il Framework della Microchip vi renderete conto che definendo il nome:

```
#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
```

ci sono diversi controlli che attivano o meno tutti i blocchi che sono stati spiegati nel piccolo Framework di esempio. Per esempio nel seguente segmento di codice si ridefiniscono gli indirizzi:

```
#if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)
  #define REMAPPED_RESET_VECTOR_ADDRESS             0x1000
  #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS    0x1008
  #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS     0x1018
```

Naturalmente l'utilizzo del Framework Microchip è fondamentale nel caso in cui la nostra applicazione stia utilizzando la porta USB, altrimenti il piccolo Framework di esempio può essere sufficiente.

Si fa presente che EasyUSB, diversamente da Freedom II supporta le impostazioni relative al Power Managment come la scheda di sviluppo della Microchip FSUSB[35]:

```
#define USE_SELF_POWER_SENSE_IO
#define USE_USB_BUS_SENSE_IO
```

[35] EasyUSB diversamente dalla scheda di sviluppo Microchip FSUSB non possiede la porta seriale, il sensore di temperatura e il trimmer. Questo discende dal fatto che EasyUSB nasce per dare la possibilità di usare la scheda in applicazioni generiche, dando dunque priorità al power managment piuttosto che a sensori che potrebbero non essere utilizzati. I LED e i pin per il Power Managment sono in una posizione standard come per la scheda FSUSB.

Indice Alfabetico

Bibliografia

[1] www.LaurTec.com : sito ufficiale di EasyUSB dove poter scaricare ogni aggiornamento e applicazione. Il PCB di EasyUSB è reso disponibile alla sezione servizi sotto donazione di supporto al sito stesso.
[2] www.microchip.com : sito dove scaricare i datasheet del PIC18F4550 e il Framework USB.
[3] www.usb.org : sito ufficiale del consorzio USB

History

Data	Versione	Nome	Descrizione Cambiamento
18.11.10	1.0	Mauro Laurenti	Versione Originale.
01.12.10	1.0a	Mauro Laurenti	Aggiornata la nota 29 sulla necessità di avere le librerie VC++ oltre al Framework .NET 4.
19.12.10	1.0b	Mauro Laurenti	Inserita Nota sulla disponibilità del kit di espansione EasyUSB progetto PJ7009.
27.03.11	1.0c	Mauro Laurenti	Aggiornata l'immagine della scheda di espansione Freedom II – EasyUSB (PJ7007). Diminuita la risoluzione delle immagini per ridurre le dimensioni del file.
10.03.12	1.0d	Mauro Laurenti	Inserita foto della scheda di espansione PJ7011 con breadboard.
07.10.12	1.0e	Mauro Laurenti	Correzioni di battitura.