**Micro-GT traffic light (semaforo)  
Per imparare a programmare i PIC e anche i PLC.**

https://www.youtube.com

Questo modellino in scala è orientato specificamente alla didattica in ambito dei microcontrollori PIC. Si tratta di una variante della piattaforma Micro-GT mini di cui mantiene tutte le caratteristiche circuitali di base.

Senza modifiche circuitali potrà essere impiegato per le esercitazioni di programmazione dei PLC.

A seconda del livello raggiunto dagli allievi sarà possibile utilizzarla con MCU della famiglia 16, ad esempio il noto 16F876A oppure il più performante 18F2550 che permetterà l'interfacciamento con la porta USB con l'aggiunta di un solo condensatore al pin 14 e il conseguente spostamento del pin di controllo di una luce della lanterna semaforica.

Il sistema potrà lavorare standalone oppure interfacciato al computer rendendo gli innumerevoli esercizi sviluppabili di interesse non solo per gli elettronici ma anche per gli informatici.

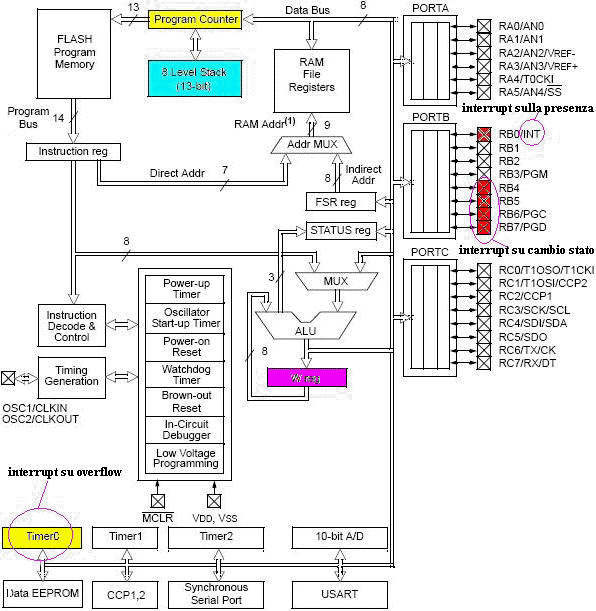
Per velocizzare i tempi di programmazione dell'area flasch durante la fase di apprendimento si consiglia l'uso del bootloader anche se il modello dispone di una porta ICSP per il collegamento diretto al programmer PICKIT3 o alla porta ICSP presente nel modello versatile IDE della Micro-GT.

**Descrizione del sistema e finalità.**

Il modellino dell'incrocio a due strade dispone di due attraversamenti pedonali che andranno programmati usando gli interrupt. A tal fine i due pulsanti di richiesta sono collegati a ingressi dedicati, RB0 e RB5, che come scopriremo vanno usati in maniera diversa.

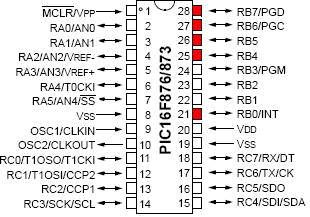
Nell'immagine che segue vediamo i pin di I/O digitale in grado di generare in maniera hardware un segnale di interruzione del programma in corso al fine di lanciare una routine di servizio generalmente detta ISR (iterrupt service routine).

Il processore in esame è il PIC16F876A, ma per analogia si potrà cercare sui databook del modello che si vorrà utilizzare la medesima funzione e nello specifico se è abbinabile al pin hardware in cui è stato pre collegato il pulsante di richiesta pedonale.



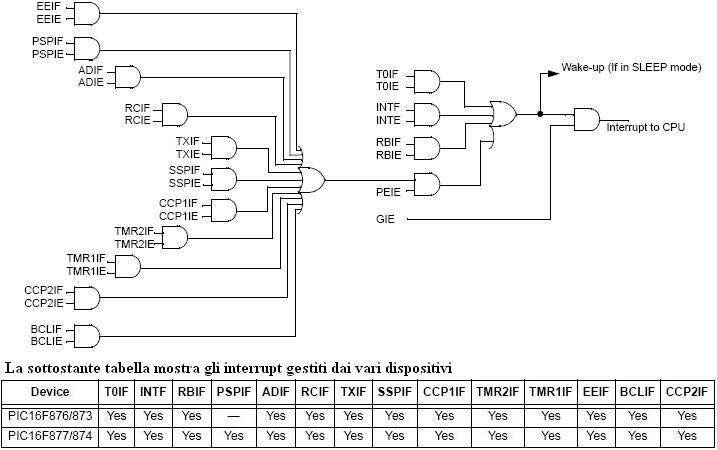
PIC16F876A, in rosso i pin in grado di generare l'interrupt.

Vediamo in pinout di questo PIC per identificare dove sono posizionati i pin in questione.



Va detto che molti PIC di nuova generazione hanno i pin rimappabili che consentono di spostare l'I/O e le funzioni ad esso abbinate dove risulta più utile e conveniente.

All'interno del PIC esiste una sezione hardware che ha il compito di rilevare la presenza della richiesta di interrupt e decidere se questa va servita o ignorata. Si tratta di una cascata di porte AND che vengono tutte inibite da un unico bit, detto "abilitatore generale degli interrupt" o GIE (general interrupt enable).



Se teniamo come riferimento il PIC 16F876A i bit di abilitazione, gestione, e controllo degli interrupt si trovano in alcuni registri di cui il più importante è denominato "interrupt control" accessibile tramite la sintassi dell'hitech C con la parola chiave INTCON. Vediamone le funzionalità.

**Registro INTCON**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF |

Il registro INTCON è un area di lettura e scrittura (R/W) in cui è possibile abilitare dei bit abbinati alle varie fonti di interrupt.

Ogni bit di flag viene impostato alto quando si verifica la richiesta di interrupt dall'esterno o da una periferica interna (di quelle in grado di farlo) ma la richiesta di interruzione viene ignorata se non è abilitato il bit GIE, come già detto. GIE è il settimo bit.

In definitiva questo registro è il controllo di ingresso della rete logica vista sopra, difatti i bit da 0 a 6 sono usati per la configurazione e abilitazione gestendo anche le 4 fonti esterne

.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-x |      |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF |      |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 | | R = bit leggibile  W= bit scrivibile  U = bit libero, letto come ‘0’ reset |

Vediamo le funzioni abbinate ai singoli bit di INTCON.

**bit 7:**

GIE: bit Global Interrupt Enable  
1 = Abilita il mascheramento (lascia passare) tutti i segnali di interrupt presenti.  
0 = Disabilita tutti gli interrupt anche se presenti a 1 nei flag.

**bit 6**

PEIE: Perifeferal interrupt enable bit

1 = abilita tutti gli interrupt dalle periferiche non mascherate  
0 = disabilita tutti gli interrupt dalle periferiche

**bit 5**

T0IE: TMR0 abilita interrupt che segnala Overflow  del Timer0

1 = abilita interrupt TMR0  
0 = Disabilita interrupt TMR0

**bit 4:**

INTE: RB0/INT abilita Interrupt sul pin del PIC  
1 = abilita interrupt  RB0/INT  
0 = disabilita interrupt  RB0/INT 

**bit 3:**

RBIE: RB bit di abilitazione interrupt sul cambio di stato sul PORTB

1 = abilita su cambio di stato del PORT RB  
0 = disabilita sul cambio di stato del PORT RB

**bit 2:**

T0IF: interrupt su tempo scaduto di TMR0 (overflow interrupt flag bit)  
1 = Se posto a 1 il TMR0 è scaduto e deve essere ripristinato a zero via software  
0 = se TMR0 non è scaduto

**bit 1:**

INTF: RB0/INT Interrupt Flag bit  
1 = alto significa che c'è richiesta di interrupt su RB0/INT

0 = basso significa che non c'è richiesta di interrupt  RB0/INT

**BIT 0:**

RBIF: RB alza il falg se c'è cambio di stato su uno dei bit a PORTB

1 = Quando almeno un bit  tra RB7 e RB4 cambia stato (deve essere resettato via software)  
0 = nessuno dei pin tra RB7 e RB4 ha cambiato stato

Dalla lettura delle caratteristiche tecniche nel data book del 16F87x si vede che il pin RB0, associato al segnale INT (ovvero interrupt) è adatto ad agire sulla presenza, quindi uno logico, per lanciare automaticamente l'ISR, mentre i pin da RB4 a RB7, possono lanciare un interrupt sul cambio di stato.

Sarà anche possibile mascherare l'interrupt ad esempio accettandolo sul fronte di salita oppure sul fronte di discesa.

E' dunque evidente che i due tipi di interrupt si gestiscono in maniera differente rendendo l'esercizio associato al modello più impegnativo.

Un concetto fondamentale è che il programmatore non dovrà mai chiamare la routine di servizio perché' questa è evocata automaticamente dalle fonti di cui abbiamo parlato prima, e queste potranno essere hardware sia interne che esterne al PIC.

Vi sono due fondamentali regole da non dimenticare quando si programma con interrupt.

1. Non è possibile richiamare funzioni all'interno della routine di servizio.
2. Prima di uscire dalla routine di servizio bisogna ricordarsi di porre a zero il flag che ci ha portato al suo interno, questo va fatto comunque, nel caso la situazione che lo ha richiesto sia ancora presente la ISR verrà evocata di nuovo.

Vediamo il codice della funzione di interrupt usata per il firmware demo rilasciato con questo articolo e che sarà la base da cui partire per lo sviluppo di esercizi più complessi. Questa versione risponde all'evento di presenza sul pin RB0 associato al segnale INT.

Due cicli for nidificati ricordano vagamente l'effetto di una Delay.

Il ciclo più interno è la base tempi mentre quello che lo contiene sarà un suo fattore moltiplicativo.

Usando le inline macro, che notoriamente non sono delle funzioni, è possibile dare una base tempi dell'ordine del micro secondo o di tre ordine di grandezza maggiore, quindi del milli secondo.

Fattori moltiplicativi di questa base tempi ci restituiscono il ritardo desiderato.

Aumentando la nidificazione è possibile ottenere tempi anche molto lunghi anche se il calcolo preciso del tempo trascorso potrebbe risultare poco preciso.

Questa imprecisione e' comunque di poco conto in un'applicazione come i tempi semaforici.

// funzione lanciata al verificarsi dell'interrupt

// pressione del tasto RB0 o RB4

int interrupt evento(){

   if (INTF == 1) { // è stato premuto RB0 che corrisponde al pulsante walk\_mazz

      mazz\_ped\_green = 1; mazz\_ped\_red = 0;

      //forzamanto immediato del rosso veicolare

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      for (int mult\_base = 0; mult\_base < 10000; mult\_base++) { // delay simulata con cicli a vuoto nidificati, e' possibile ottenere anche tempi molto lunghi

        for (int time\_base = 0; time\_base < 100; time\_base++) {

        }

      }

      mazz\_ped\_green = 0; mazz\_ped\_red = 1;

   }

   INTF = 0;  // Viene resettato il flag per impedire l'entrata  in loop infinito nella ISR

}

Va detto che la routine per funzionare ha bisogno della corretta impostazione dei registri del PIC.  Qui sotto e' esposto cosa fare.

 INTCON = 0b10010000;  // abilito GIE dell'INCON register

I settaggi di tutti i registri necessari a far funzionare questo progetto si trovano nel modulo di intestazione (header file) con nome settaggi.h

/\*

 \*  Settaggi del PIC

 \*

 \*/

void settaggi(){

    #define \_LEGACY\_HEADERS  // permette di acquisire la sintassi dei compilatori precedenti

    \_\_CONFIG (HS & WDTEN & PWRTDIS & BORDIS & LVPDIS & DUNPROT & WRTEN & DEBUGDIS & UNPROTECT); //Fuses

    //ADCON1 = 0b10000000; // abilita gli ingressi analogici AN0 .. AN7 su PORTA

    ADCON1 = 0b00000111; // disabilita gli ingressi analogici AN0 .. AN7 su PORTA

    CMCON =  0b00000111;  // disabilita comparatori

    /\*

     \* setto le porte A B C D E come ingressi ( = 1 ) o come uscite ( = 0 )

     \* regola importante, per evitare qualsiasi tipo di disturbo se non uso una

     \* porta la setto come output

     \*

     \* N.B.

     \*

     \* TRISX = 0 equivale a TRISX = 0b00000000

     \*

     \* TRISX = 1 NON equivale a TRISX = 0b11111111

     \* ma equivale a TRISX = 0b00000001

     \*/

    TRISA = 0b000000;  // PORTA tutti output

    TRISB = 0b00010001;  // imposto input per leggere gli interrup

    // tutti output

    TRISC = 0b11000000; // per abilitare le comunicazione seriale

    //TRISD = 0;

    //TRISE = 0;

    PORTA = 0;

    PORTB = 0;

    PORTC = 0;

    //PORTD = 0;

    //PORTE = 0;

    // interrupt

  INTCON = 0b10010000;  // abilito GIE dell'INCON register

    /\* registro INTCON

     \* GIE general inerrupt enable

     \* PIEIE interrupt generato da periferiche interne o esterne (COM - USB)

     \* T0IE abilitazione interrupt da timer zero TMR0

     \* INTE (sempre 1, lavora insieme a GIE) interrupt abilitato

     \* RBIE abilita interrupt su cambio di stato del RB4,RB5,RB6,RB7

     \* T0IF verifica presenza interrupt da TMR0

     \* INTF verifica presenza interrupt dal PORTB

     \* RBIF presenza interrupt RB0

     \*/

   OPTION = 0b10000000;    // OPTION\_REG

    /\*

     \* registro OPTION\_REG

     \* bit 7-RBPU abilita internal pull UP sul PORTB

     \* bit 6-INTEDG abilita interrupt sui fronti

     \* bit 5-TOCS resetta timer 0

     \* bit 4-TOSE abilita timer 0

     \* bit 3-PSA    // prescaler assignement bit

     \* bit 2-PS2    // ultimi tre bit: prescaler

     \* bit 1-PS1    // da vedere sul databook

     \* bit 0-PS0

     \*/

//Inizializzazione uart

    TXSTA = 0x20; // TX abilitato

    RCSTA = 0X90; // Registro RX USART

    BRGH = 0; // USART High velocity

    SPBRG = 15; // Asynchronous abbinato a BRGH = 0, Baud Rate Generator (15=19200 baud 20Mhz)

    //Inizializzazione TIMER1

    T1CON = 0; // TIMER 1 resettato

    T1CKPS1 = 0, T1CKPS0 = 0; // TIMER1 prescaler = 1

    T1OSCEN = 0; // TIMER1 external oscillator disable

    TMR1CS = 0; // TIMER1 internal clock (fosc/4)

    TMR1ON = 0; // TIMER1 in STOP

}

Come possiamo notare, l'impostazione TXSTA = 0x20; predispone il PIC per trasmettere le stringhe al computer in cui potra' essere messo in esecuzione il programma di interfaccia. Questo potra' essere scritto in qualunque linguaggio ad esempio java o visul basic.

La struttura del progetto, sviluppato per MPlab X, e' nella figura qui sotto.



Il progetto demo, privo di comunicazione seriale e che gestisce un unico passaggio pedonale (in via Mazzini), con chiamata su presenza che lancia l'interrup è scaricabile da questo link:

[Download progetto MPLAB X, sorgenti per Hitech C16 e MCU  16F876A](http://www.grix.it/UserFiles/ad.noctis/MicroGT_traffic_light.zip)

**Schema elettrico.**

Lo schema è diviso in pochi ed essenziali blocchi caratterizzati da configurazioni circuitali di base.

In alto a destra abbiamo un alimentatore in grado di fornire 1 ampere, valore più che adeguato in questa circostanza.

Notiamo che il valore del condensatore elettrolitico C10 a monte del regolatore di tensione ha un valore maggiore, quasi doppio, di C9 a valle del regolatore. Si tratta di una strategia abbastanza comune che permette di mantenere il flusso di corrente sempre diretto e mai inverso attraverso il LM7805, anche in caso di mancanza improvvisa di tensione a monte.

I due condensatori ceramici da 100 nF, collegati direttamente al pin centrale del regolatore aiutano a impedire l'auto oscillazione dello stesso contenendo le varie cause che possono causare il surriscaldamento.

E' importante notare che il circuito potrebbe essere alimentato non solo con fonti alternate ma anche con un normale alimentatore switching, ad esempio recuperato da un vecchio modem, stampante o dispositivo analogo. In questo caso non dovremmo preoccuparci con che polarità ci colleghiamo al morsetto X1.  Il ponte di diodi presenterà comunque la giusta polarità al regolatore di tensione.

Al centro dello schema vi è la sezione di input.

Questa è composta di soli pulsanti in chiusura (nella vecchia nomenclatura N.A.).

I due sono configurati in maniera differente per la questione esposta al paragrafo precedente e legata a come gli interrupt della chiamata pedonale si presenteranno.

E' evidente che su RB0 viene lanciato 1  mentre su RB4 si lancia uno zero.

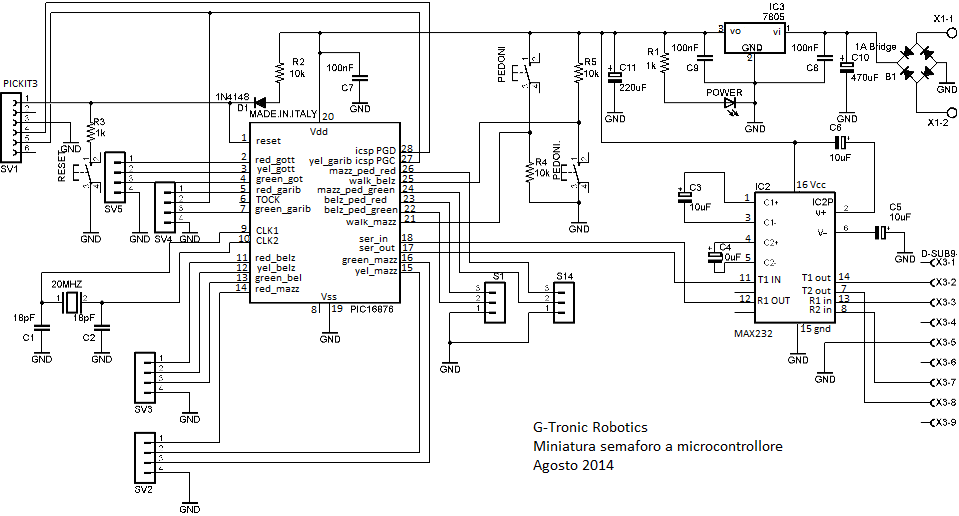
Sulla sinistra abbiamo la porta ICSP alla quale potremo collegare il programmer, ad esempio il PICKIT 2 oppure 3. Grazie alla presenza del diodo D1 i circa 14 Volt generati dal pin 1 del programmer possono raggiungere solo ed esclusivamente il pin 1 del PIC, essendo questo l'unico tollerante ad una tensione così alta.

Semplificando un po', possiamo dire che internamente è collegato un comparatore che commutando a circa 13 volt dice all'architettura interna di cambiare la funzionalità da esecuzione di quanto contenuto nell'area programma della memoria a scrittura della medesima.

Si instaura così un flusso di dati sul pin 4, verso il pin 28 del pin, governati da un segnale di clock iniettato dal programmer tramite il suo pin 5 e accettato dal pin 27 del pin.

Va detto che i programmer ICSP possono anche alimentare i dispositivi che vanno a programmare ma nel nostro caso si è preferito eliminare questa funzionalità ed imporre che la scheda sia accesa durante il riversamento ICSP. Questo è il motivo per cui il pin 2 del connettore ICSP è scollegato.

Non si può invece fare a meno di mettere in comune la massa del programmer con la massa della scheda a microcontrollore e questo avviene tramite il pin 3 del connettore ICSP che è appunto a massa.



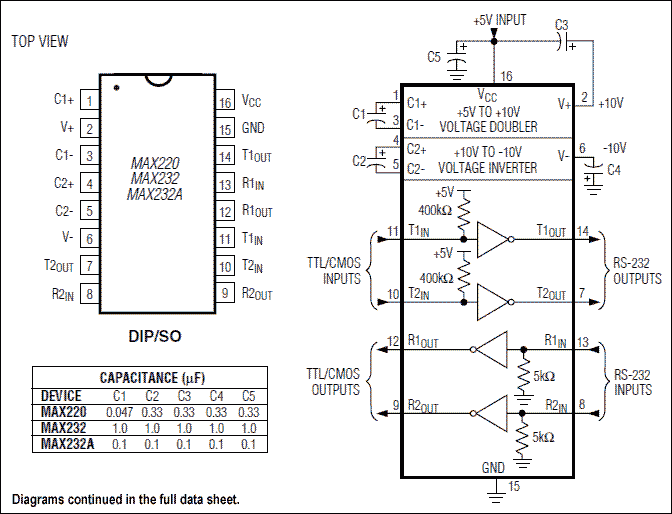
Sul lato destro e in basso dello schema vi è la porta di comunicazione del modellino. Si tratta di un classico connettore CANNON DB9, che per attenersi allo standard previsto per i DTE e i DCE sarà montato obbligatoriamente di tipo femmina. (vedi nota a fine articolo). Questi comunica con la periferica USART, che in questo PIC funziona con tensioni TTL tramite il circuito integrato MAX232 che permette l'interfacciamento con lo standard EIA-RS232C, che notoriamente non lo è.

Lo standard TTL prevede l'unico logico a +5V e lo zero a massa, mentre il protocollo seriale ha tensioni più che doppie e per giunta duali, ovvero che presentano anche un valore negativo.

Si verifica inoltre un'inversione della logica rispetto ai livelli di tensione.

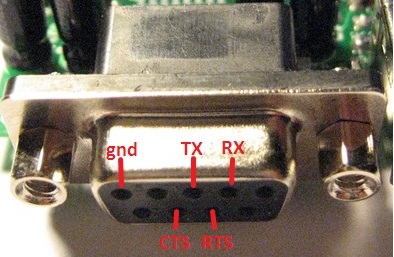
La domanda che sorge spontanea è: Da dove si può ricavare una tensione negativa se la tensione con cui alimento il circuito è singola e più bassa della maggiore presente nei cavi di trasmissione?

Il chip della MAXIM è in effetti sia un traslatore di livello, che permette di creare il valore che va da 6 a 12 volt normalmente considerati validi per lo standard in questione, che una pompa di elettroni, collegata al condensatore elettrolitico C5, che trova così una giustificazione del fatto che il suo reoforo positivo sia a massa. Questa genera la tensione inferiore allo zero di cui abbiamo bisogno.

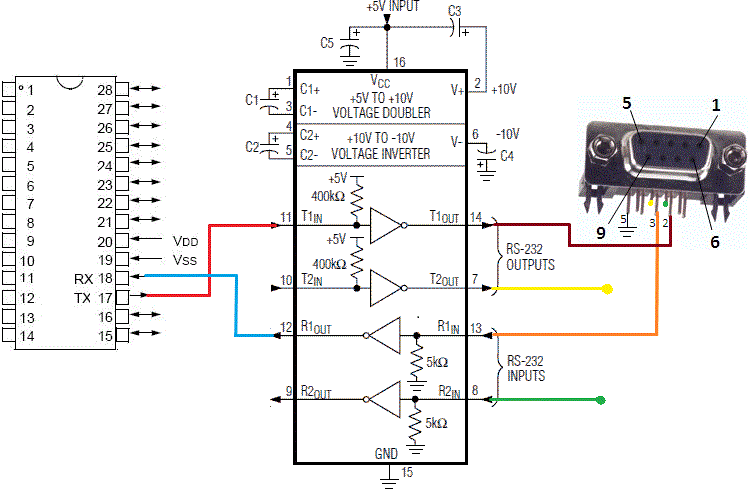


interno del chip MAX232 fonte databook della Maxim.

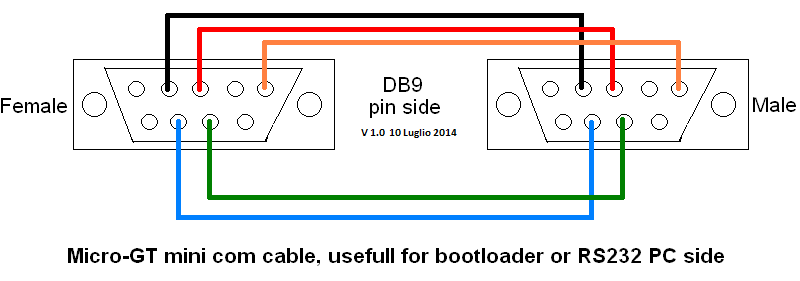
Risulta quindi evidente che il collegamento diretto dei segnali TX e RX, cosi come sono presenti in campo, alla periferica USART, senza la presenza dell'interfaccia costituita dal MAX232, porta all'immediata distruzione del microcontrollore.



La porta sarà impiegata per la programmazione quando si userà la tecnica del bootloader e per la supervisione o comunicazione da parte del PC in cui si potrà porre in esecuzione un programma svolta in qualunque linguaggio.



Il cavo e' il medesimo della Micro-GT mini ed è mostrato nell'immagine sottostante.  I connettori sono visti nel lato delle saldature.

.

Dallo schema elettrico si vede che le lanterne semaforiche hanno le luci poste in posizioni fisse, quindi si rende necessario fornire una lista delle attribuzioni, ovvero una assegnazione di etichette mnemoniche da utilizzarsi durante lo svolgimento del programma.

RA0 red\_gott

RA1 yel\_gott

RA2 green\_gott

RA3 red\_garib

RB6 yel\_garib  //Evitate RA4 perché' è un'uscita open drain e necessiterebbe di un PULLUP, inoltre conflittua con gli interrup perché' sovrapposta a TOCK

RA5 green\_garib

RC0 red\_belz

RC1 yel\_belz

RC2 green\_belz

RC3 red\_mazz

RC4 yel\_mazz

RC5 green\_mazz

RB0 walk\_mazz

RB1 belz\_ped\_green

RB2 belz\_ped\_red

RB3 mazz\_ped\_green

RB4 walk\_belz

RB5 maz\_ped\_red

Le etichette assumono un ovvio significato osservando il PCB nel paragrafo successivo.

Le abbreviazioni red, yel e green hanno un ovvio significato, mentre la parte finale dell'etichetta indica la via in cui è istallata la luce.

Con questo ragionamento diventa facile associare, a titolo di esempio, l'etichetta "green\_mazz" alla luce verde del semaforo provenendo da via Mazzini.  Il verde si accenderà associando il livello alto al pin RC5, ovvero il pin 16 dell'MCU 16F876a.

L'assegnazione nel codice C avverrà tramite la direttiva "define" in questo modo:

#define RC5 green\_mazz

La riga non deve essere chiusa con il punto e virgola dato che si tratta di una direttiva a preprocessore piuttosto che un costrutto del linguaggio.

Alcuni compilatori non gradiscono molto i caratteri speciali in mezzo al nome dell'etichetta quindi in questo caso si suggerisce di toglierlo trasformando in:

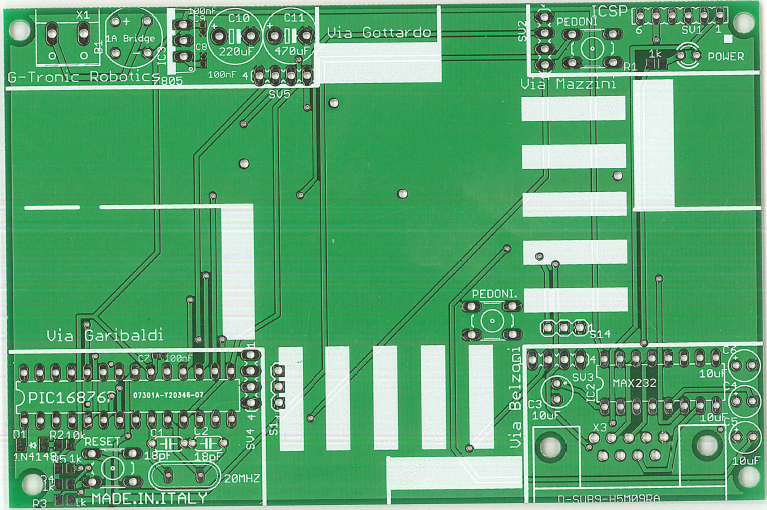
#define RC5 GreenMazz

In alcuni casi anche la lunghezza dell'etichetta può mettere in crisi il compilatore quindi se incontrate dei problemi procedete accorciandola.

I pulsanti di chiamata pedonale sono abbinati ai pin con interrupt hardware, come già detto, ma in qualche caso potrà essere comodo spostarli. Volutamente sono stati posti ininterrupt da gestire in maniera differente così da complicare un po’ l'esercizio.

**Il circuito stampato.**

Il PCB è molto compatto e risulta molto ben proporzionato. I pali semaforici sono alti 5cm e sono estraibili perché innestabili tramite un connettore a stripline. L'ingombro della base è di soli 12cm per 8cm, il che è molto vantaggioso durante lo stoccaggio, specialmente se una scuola dovesse averne molti esemplari. In pratica si può riporre tutta la dotazione delle postazioni scolastiche in una scatola non più grande di una scatola da scarpe ed infilare il tutto al sicuro nell'armadietto del professore.



Lato componenti del modellino del semaforo Micro-GT (120mmx80mm).

Per questioni estetiche le strade sono lasciate libere da componenti e le sezioni sono state distribuite in maniera molto chiara nelle zone adiacenti.

In basso a sinistra abbiamo la scheda logica di controllo che ospita il PIC a 28 pin, quarzo e il pulsante di reset. La posizione di questo è stata studiata per essere molto accessibile dato che se si caricherà il firmware tramite bootloader verrà premuto molto spesso.

In basso a destra c'è l'interfaccia di comunicazione che in alcuni casi potrà essere omessa con un risparmio sul costo complessivo del modellino. Questa sezione potrà essere non installata se:

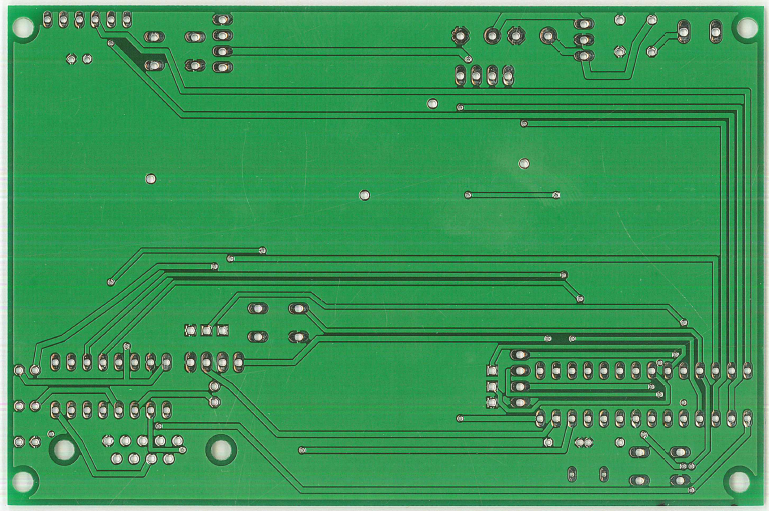
1. Non si intende programmare il PIC tramite il bootloader ma si possiede il PICKIT3.
2. Non si intende sviluppare un'interfaccia a PC, ad esempio in java, in  python, in .net ecc.

In alto a sinistra abbiamo la sezione di alimentazione in cui il ponte di diodi impedisce la distruzione in caso di inversione della polarità del generatore continuo che verrà collegato. Si consiglia un alimentatore compreso tra i 7V e 12V recuperato ad esempio da un vecchio modem, hub, ecc.

Benché funzionerà ugualmente, consiglio di portare il valore del condensatore C10 a 470uF, o in alternativa di scambiare i valori di C11 e C10, questo allo scopo di favorire la fase di scarica quando si spegne il circuito, ma come detto anche con i valori indicati non avrete problemi.

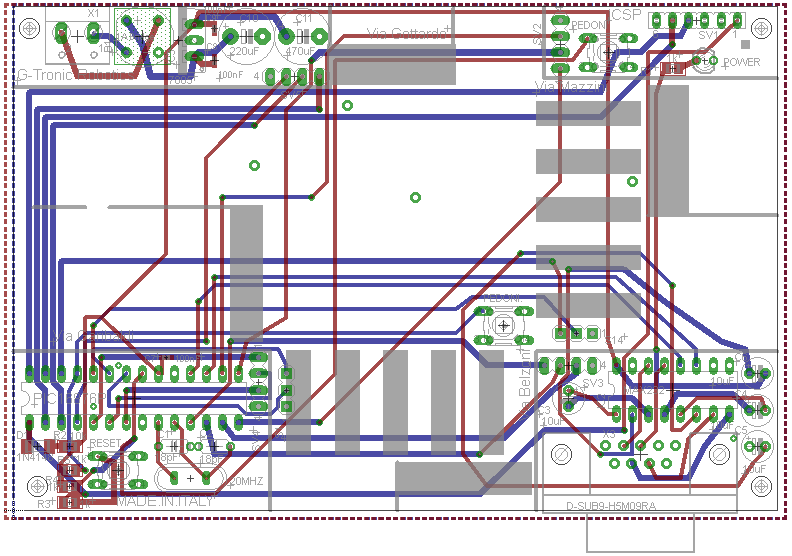
In alto a destra (il pcb è visto da sotto) abbiamo la porta di programmazione ICSP a cui collegare il PICKIT3 o la versione 2. Qualsiasi altro programmer munito di questa porta potrà essere collegato. Il pin 1, indicato con un triangolino nel PICKIT3 è mostrato sul PCB con il quadratino bianco. Nella sezione vi è inoltre il LED che segnala che il modellino è acceso.

Vi sono nello stampato degli ampi piani di massa che rendono il circuito immune ad eventuali disturbi e esteticamente più gradevole, vi è un vantaggio anche dal punto di vista della robustezza.

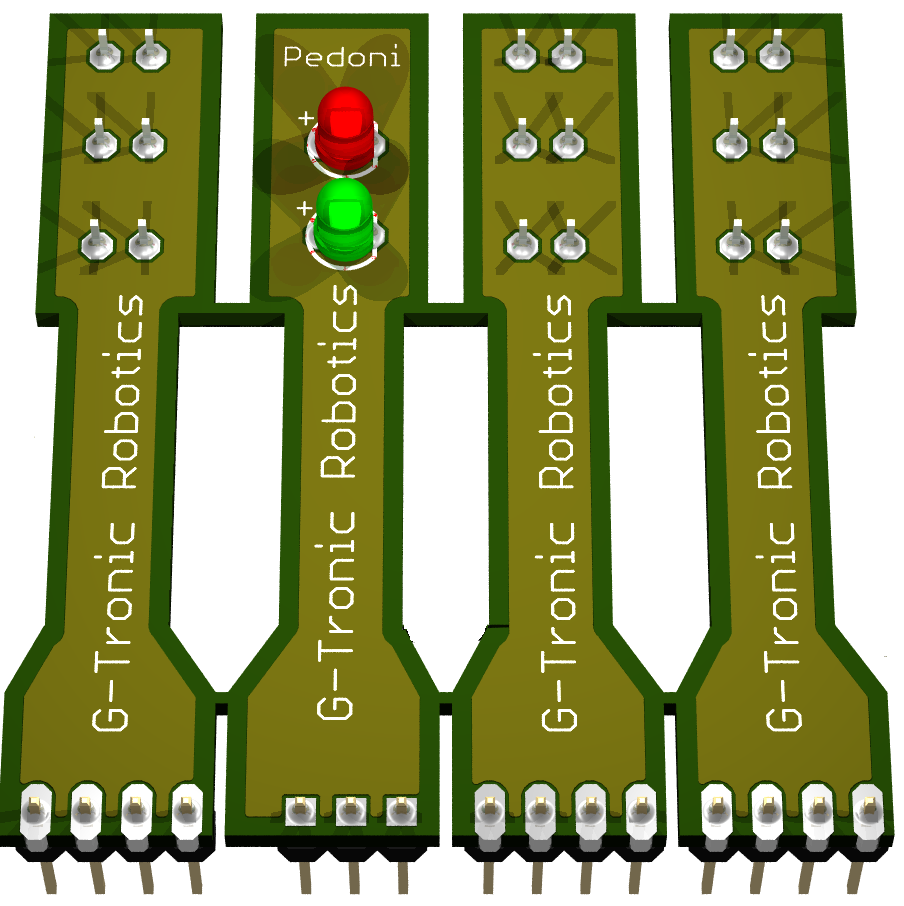


Base del semaforo vista da sotto.

Vediamo la fase di sbroglio eseguita con il CAD Eagle versione 6.3

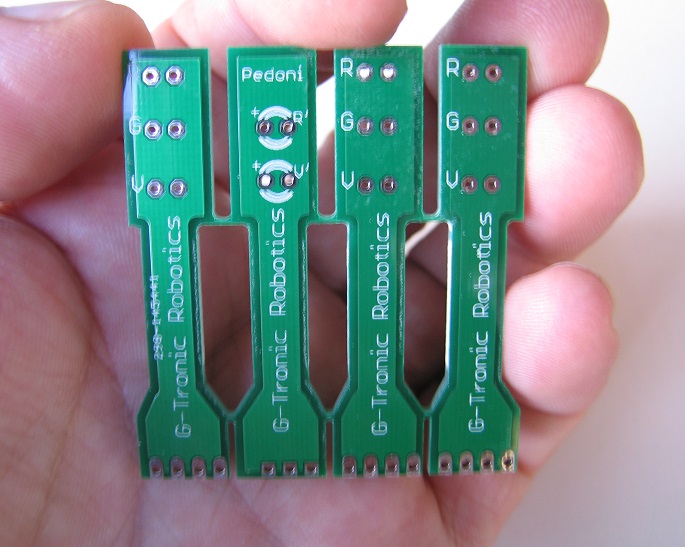


L'anteprima 3D dei pali semaforici è mostrata nella foto seguente:



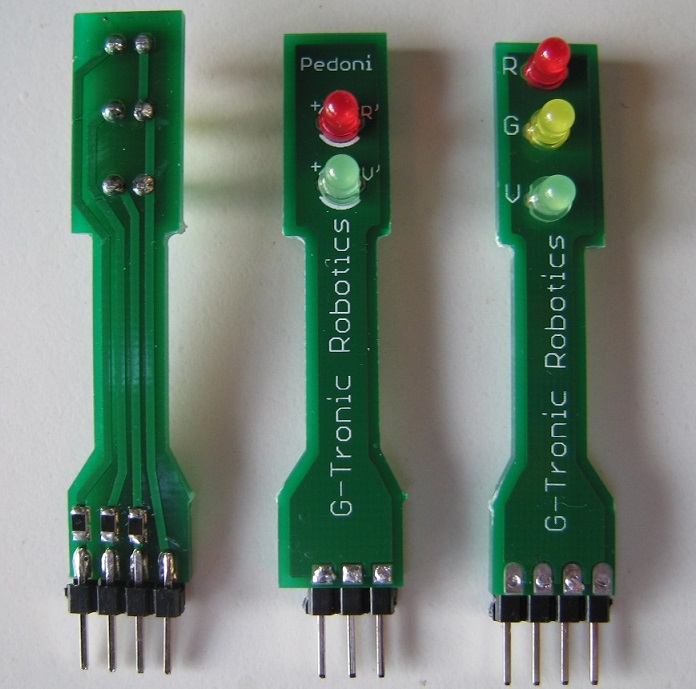
il PCB è contenuto in un quadrato di 5cm di lato, contiene tre semafori standard e un pedonale.

Nella foto sotto vediamo il PCB 5cmX5cm che contiene i semafori.

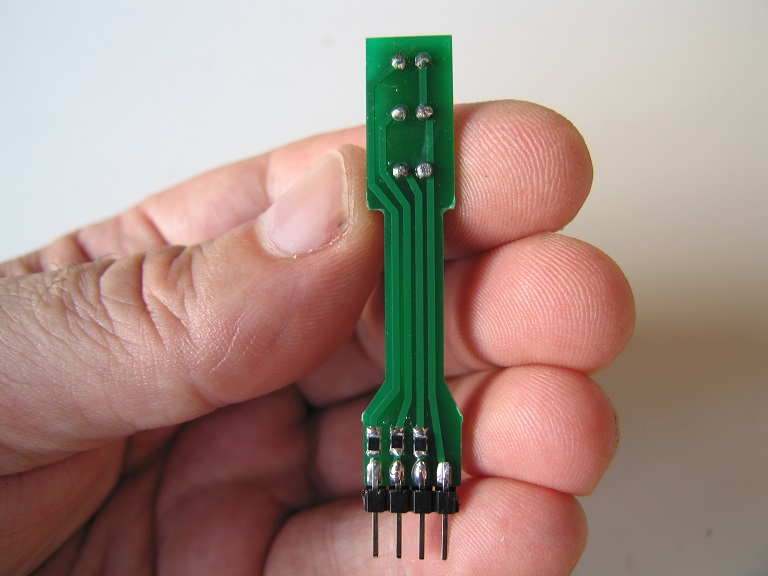


Va comunque detto che non tutti i produttori di PCB consento di fare il group (l'assieme di elementi diversi sullo stesso supporto) perché hanno l'obbiettivo di massimizzare i profitti.

Gli esemplari vanno separati usando una semplice forbice. Gli strip line è bene che siano maschi sui pali semaforici e femmina sulla base, come da convenzione spina-presa ma saranno montati con sviluppo parallelo al PCB e non perpendicolare come mostrato nel disegno. Questa imprecisione è stata introdotta dal CAD 3D ed è solo un'immagine indicativa. Vediamo come si montano i connettori nella foto reale.



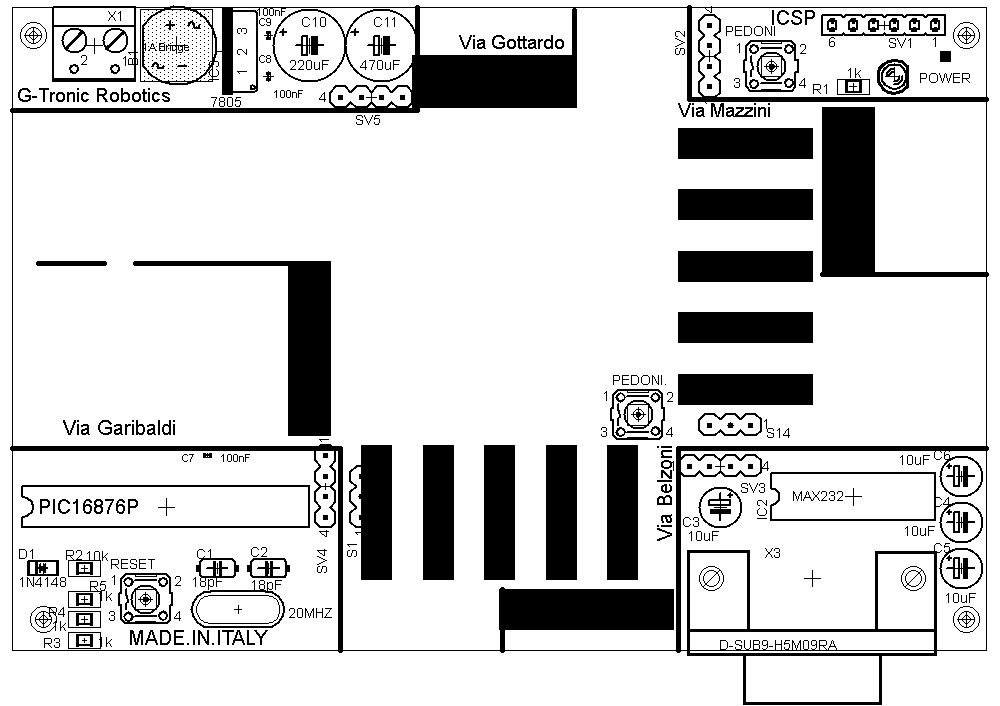
Quando si vorrà utilizzare il modellino con i PLC, che come sappiamo usano tensioni a 24V dc, dovremmo montare delle resistenze da 10K al posto di quelle indicate nella prossima immagine che ci da anche un'idea della reale dimensione dell'oggetto comparato con la mano.



Va segnalato che i catodi devono essere rivolti alla massa che nella foto è il collegamento comune più a destra.

In caso di utilizzo del modellino con un PLC si dovrà considerare che normalmente l'I/O digitale funziona a 24V d.c. (salvo diversa indicazione), quindi le resistenze che vedete collegate subito sopra allo stripline dovranno essere calcolate in maniera opportuna. La corrente di accensione potrà essere di circa 2mA, si procederà quindi al calcolo applicando la legge di Ohm alla maglia, e tenendo conto di una caduta anodo catodo di circa 2V.

Di grande aiuto nella fase di montaggio è l'immagine delle serigrafie in cui possiamo vedere i valori dei componenti.

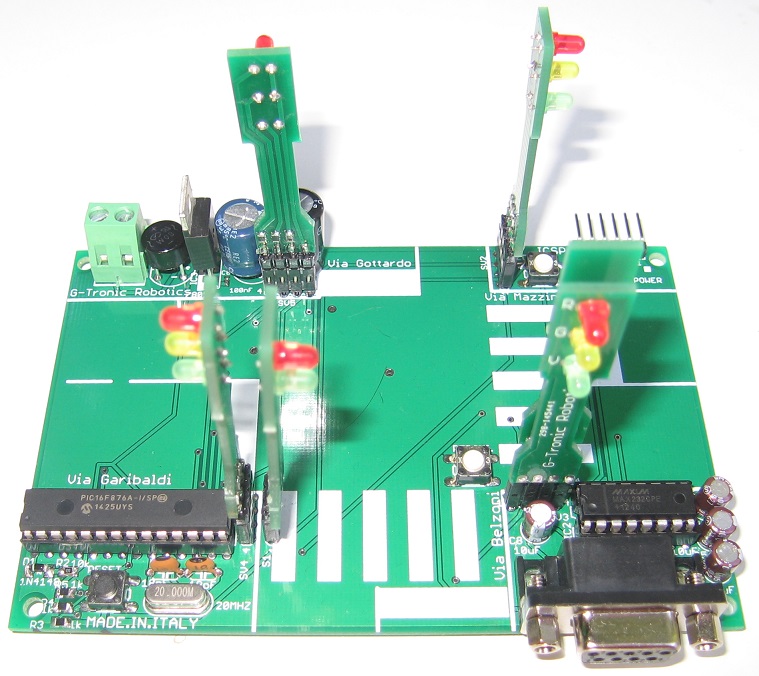


Come era intuitivo i disegni delle strisce pedonali e i nomi delle strade sono sul medesimo piano delle serigrafie.

Punto di orgogli di questo progetto è la scritta "Made in Italy".

**Il circuito assemblato.**  
Il circuito assemblato mostra una buona robustezza e la messa in opera del modellino sarà quasi immediata da parte degli allievi che dovranno svolgere gli esercizi di programmazione.

La foto mostra come orientare i pali semaforici. In pratica le lampade semaforiche staranno sempre alla destra di chi con l'ipotetica vettura si approssima all'incrocio.



[Download galleria fotografica miniatura semaforo con 16F876A](http://www.grix.it/UserFiles/ad.noctis/Micro-GT_traffic_light_miniature_PIC16F876A.zip).

**Esempio di firmware.**

Quanto segue è solo un esempio di firmware che implementa un semplice ciclo semaforico, ma molteplici varianti sono possibili a seconda della fantasia di chi si cimenterà sul modellino.

La funzione di interrupt e' predisposta e funzionante, benché' sia limitata alla sola chiamata e all'azione sulla lanterna semaforica pedonale di via Belzoni.

Viene lasciato, come utile esercizio per chi si vorrà cimentare, un'implementazione più articolata di questa chiamata pedonale.

Nella versione qui svolta il ciclo semaforico si avvia con la viabilità consentita (il verde) da via Belzoni verso via Gottardo, ne consegue che il transito veicolare è inibito da via Garibaldi verso via Mazzini.

Durante il normale ciclo semaforico gli attraversamenti pedonali sono inibiti perpendicolarmente al verde della viabilità.

Quando in una strada si presenta il giallo il verde pedonale lampeggia in rapida sequenza avvisando che sta per scattare il rosso.

Il rosso pedonale apparirà prima della comparsa del verde di viabilità veicolare perpendicolare così che, se si rispettano le regole della strada, non è possibile che si verifichi l'investimento di un pedone.

Ogni luce semaforica è indipendente, nel senso che gli è stato assegnato un pin del PIC, in questa maniera il programmatore potrà inventare cicli più complessi.

Si consigliano rielaborazioni in cui si distingue tra il ciclo diurno e il ciclo notturno.

Si potrà prevedere un controllo manuale a cura di un vigile che fa da operatore.

Si potrà pensare alla situazione di semaforo spento segnalata con le luci gialle lampeggianti, il controllo da PC delle varie modalità di funzionamento e molte altre cose ancora..

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\*   Codice scheda: Micro-GT traffic light

\*   Firmware version:1.0

\*   autore:

\*   Data:

\*   MCU: PIC16F876a

\*   Piattaforma hardware: Micro-GT IDE

\*   Piattaforma software: MPLAB X 2.10

\*   Compilatore: HI-TECH C Compiler for PIC10/12/16 MCUs(Lite Mode) V9.70

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <pic.h>

#include <htc.h>           // abilita gli interrupt

#include "delay.h" // funzioni di ritardo

#include "settaggi.h" // setta entrate uscite e fuses

// semaforo via Gottardo

#define red\_gott    RA0 // USO IL PORTD NEL 16F877A AL POSTO DEL PORT A DEL 16F876A

#define yel\_gott    RA1

#define green\_gott  RA2

// semaforo via Garibaldi

#define red\_garib   RA3

#define yel\_garib   RB6

#define green\_garib RA5

// semaforo via Belzoni

#define red\_belz    RC0

#define yel\_belz    RC1

#define green\_belz  RC2

// semaforo via Mazzini

#define red\_mazz    RC3

#define yel\_mazz    RC4

#define green\_mazz  RC5

// pulsante chiamata pedonale via Belzoni

#define walk\_belz       RB4

// semaforo pedonale via Belzoni

#define belz\_ped\_green  RB1

#define belz\_ped\_red    RB2

// pulsante chiamamta pedonale via Mazzini

#define walk\_mazz       RB0

// semaforo pedonale via mazzini

#define mazz\_ped\_green  RB3

#define mazz\_ped\_red    RB5

// funzione lanciata al verificarsi dell'interrupt

// pressione del tasto RB0 o RB4

int interrupt evento(){

   if (INTF == 1) { // è stato premuto RB0 che corrisponde al pulsante walk\_mazz

      mazz\_ped\_green = 1; mazz\_ped\_red = 0;

      //forzamanto immediato del rosso veicolare

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      for (int mult\_base = 0; mult\_base < 10000; mult\_base++) { // delay simulata con ciclo vuoto

        for (int time\_base = 0; time\_base < 100; time\_base++) {

        }

      }

      mazz\_ped\_green = 0; mazz\_ped\_red = 1;

   }

   INTF = 0;

}

void main(){

settaggi();

   // semafori pedonali sempre rossi a meno della pressione dei pulsanti

      belz\_ped\_green = 0; belz\_ped\_red = 1;

      mazz\_ped\_green = 0; mazz\_ped\_red = 1;

while(1){

      /\* situazione semafori

       \* gottardo belzoni garibaldi mazzini

       \* verde    verde   rosso     rosso

       \*/

      red\_gott = 0; yel\_gott = 0; green\_gott = 1;

      red\_belz = 0; yel\_belz = 0; green\_belz = 1;

      belz\_ped\_green = 0; belz\_ped\_red = 1;

      mazz\_ped\_green = 1; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayS(8);

      /\* situazione semafori

       \* gottardo belzoni garibaldi mazzini

       \* arancio  arancio  rosso     rosso

       \*/

      red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 0; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 1; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 0; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

            red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 1; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 0; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 1; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 0; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 1; mazz\_ped\_red = 0;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 0; yel\_gott = 1; green\_gott = 0;

      red\_belz = 0; yel\_belz = 1; green\_belz = 0;

            mazz\_ped\_green = 0; mazz\_ped\_red = 1; //rosso pedonale anticipato in v. Mazzini

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayMs(255);

      /\* situazione semafori

       \* gottardo belzoni garibaldi mazzini

       \* rosso    rosso   rosso     rosso

       \*/

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 0; belz\_ped\_red = 1;

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayS(1);

      /\* situazione semafori

       \* gottardo belzoni garibaldi mazzini

       \* rosso    rosso   verde     verde

       \*/

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      mazz\_ped\_green = 0; mazz\_ped\_red = 1;

      belz\_ped\_green = 1; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 0; green\_garib = 1;

      red\_mazz = 0; yel\_mazz = 0; green\_mazz = 1;

      DelayS(8);

      /\* situazione semafori

       \* gottardo belzoni garibaldi mazzini

       \* rosso    rosso   arancio   arancio

       \*/

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 0; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 1; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 0; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 1; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 0; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 1; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 0; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 1; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 0; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 1; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      belz\_ped\_green = 1; belz\_ped\_red = 0;

      red\_garib = 0; yel\_garib = 1; green\_garib = 0;

      red\_mazz = 0; yel\_mazz = 1; green\_mazz = 0;

      DelayMs(255);

      /\* situazione semafori

       \* gottardo belzoni garibaldi mazzini

       \* rosso    rosso   rosso     rosso

       \*/

      red\_gott = 1; yel\_gott = 0; green\_gott = 0;

      red\_belz = 1; yel\_belz = 0; green\_belz = 0;

      mazz\_ped\_green = 0; mazz\_ped\_red = 1; //anticipo rosso pedonale V. Belzoni

      red\_garib = 1; yel\_garib = 0; green\_garib = 0;

      red\_mazz = 1; yel\_mazz = 0; green\_mazz = 0;

      DelayS(1);

CLRWDT(); //resetta il watch dog timer

}

}

 Scarica il file .hex della versione semplificata esposta sopra -> [Download Micro-GT triffic light basic demo](http://www.grix.it/UserFiles/ad.noctis/MicroGT_traffic_light_basic_demo.zip)

**Versione completa.**

La versione completa, per microcontrollore, comporta l'interfacciamento del modellino con il computer e quindi lo sviluppo di un software di comunicazione che sfrutti la porta COM presente onboard.

Innanzitutto va detto che non saremo vincolati al baudrate per cui è stato impostato il firmware del bootloader dato che il caricamento del programma setta i registri interni secondo la modalità che indicheremo in code.

Questo significa che potremmo impostare l'interfaccia PC e porta di comunicazione del modello a 9600 bps, ma il bootloader, quando evocato, continuerà a funzionare a 38400 bps.

Ci adegueremo alla stringa standard di comunicazione seriale Micro-GT, e questa sarà @,C1,C2,C3,C4,-> quindi con un totale di 6 caratteri di cui l'ultimo, la freccia, è in realtà il doppio carattere nr ovvero il classico ASCII  line feed & carriage return abbinati al tasto di invio. Ricordiamo che sono espressi nella posizione esadecimale 0xA e 0xD della suddetta tabella.

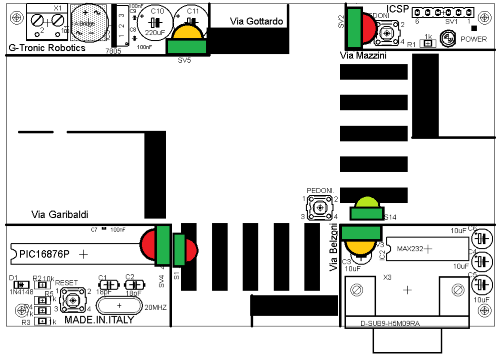
/\*  
\* convenzioni  
\* la micro GT invia la seguente stringa:  
\*  
\* @ C1 C2 C3 C4 INVIO  
\*  
\* @ inizio stringa  
\*  
\* C1 semaforo via Gottardo (quello in via Belzoni si comporta allo stesso modo)  
\* possibili valori: 1 = verde 2 = arancione 3 = rosso  
\*  
\* C2 semaforo via Garibaldi (quello in via Mazzini si comporta allo stesso modo)  
\* possibili valori: 1 = verde 2 = arancione 3 = rosso  
\*  
\* C3 semaforo pedonale via Mazzini  
\* possibili valori: 1= verde 3 = rosso  
\*  
\* C4 semaforo pedonale via Belzoni  
\* possibili valori: 1= verde 3 = rosso  
\*  
\* INVIO fine stringa  
\*/

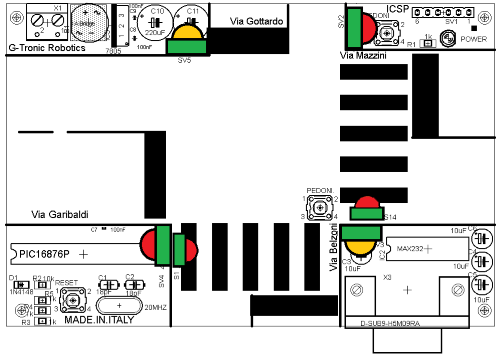
**quindi la stringa 1333 significa:**  
**Belzoni (e Gottardo) verde  
Garibaldi (e Mazzini) rosso  
pedonale Mazzini rosso  
pedonale Belzoni rosso**

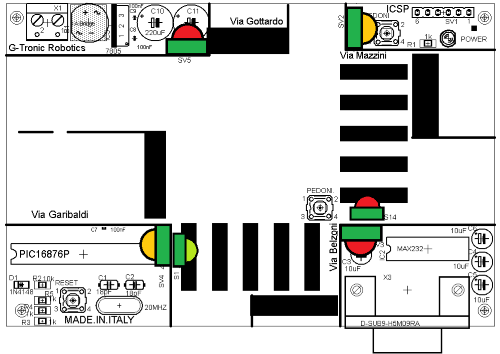
**e così via per tutte le combinazioni possibili.**

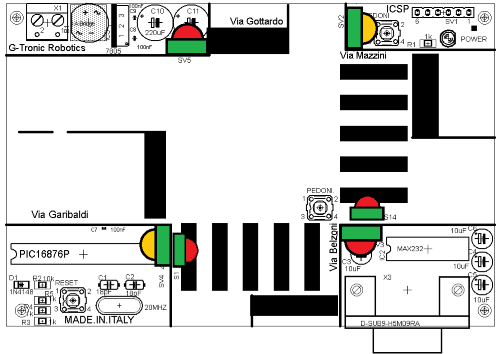
**L'interfaccia Java, sul lato PC, si compone di una classe che si occupa della comunicazione seriale, della decodifica della stringa e anche dello scambio di cinque immagine di formato png. che rappresentano lo stato dei semafori.**

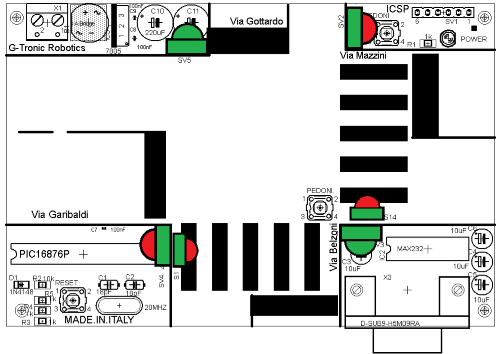
**Le immagini sono state ricavate dal layout del PCB già mostrato sopra.**

****

****

****

****

****

[**download immagini semaforo per fare da soli il programma lato PC.**](http://www.grix.it/UserFiles/ad.noctis/immagini%20semaforo.zip)

Quando si presenta la condizione corrispondente il microcontrollore genera la stringa corrispondente, che inviata via seriale evoca sul lato PC quella corretta delle cinque immagine mostrate qui sopra.

[Il bootloader e il downloader possono essere scaricati da questo link.](http://www.grix.it/UserFiles/ad.noctis/bootloader%20876A%20Micro-GT%20traffic%20light.zip)

vediamo il sorgente della classe semaforo.java .

import java.awt.BorderLayout;  
import java.awt.EventQueue;  
import java.util.Scanner;  
  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JLabel;  
import javax.swing.ImageIcon;  
  
  
public class Semafori extends JFrame {  
  
    private JPanel contentPane;  
    private static JLabel lblSemafori;  
  
    /\*\*  
     \* Launch the application.  
     \*/  
    public static void main(String[] args) {  
          
        GestioneSeriale rs232 = new GestioneSeriale();  
        //Scanner in = new Scanner(System.in);  
          
        /\*  
         \* convenzioni  
         \* la micro GT invia la seguente stringa:  
         \*   
         \* @ C1 C2 C3 C4 INVIO  
         \*   
         \* @ inizio stringa  
         \*   
         \* C1 semaforo via gottardo (quello in via belzoni si comporta allo stesso modo)  
         \* possibili valori: 1 = verde    2 = arancione    3 = rosso  
         \*   
         \* C2 semaforo via garibaldi (quello in via mazzini si comporta allo stesso modo)  
         \* possibili valori: 1 = verde    2 = arancione    3 = rosso  
         \*   
         \* C3 semaforo pedonale via mazzini  
         \* possibili valori: 1= verde    3 = rosso  
         \*   
         \* C4 semaforo pedonale via belzoni  
         \* possibili valori: 1= verde    3 = rosso  
         \*   
         \* INVIO fine stringa  
         \*/  
        int belzoni = 0;  
        int garibaldi = 0;  
        int pedonale\_mazzini = 0;  
        int pedonale\_belzoni = 0;  
        int invio = 0;  
        int situazione = 0;  
          
        EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                try {  
                    Semafori frame = new Semafori();  
                    frame.setVisible(true);  
                } catch (Exception e) {  
                    e.printStackTrace();  
                }  
            }  
        });  
          
        while(true) {  
              
              
            if (rs232.leggiSer("COM6") == '@'){    // sto ricevendo la stringa  
                  
                belzoni = rs232.leggiSer("COM6");  
                  
                garibaldi = rs232.leggiSer("COM6");  
                  
                pedonale\_mazzini = rs232.leggiSer("COM6");  
                  
                pedonale\_belzoni = rs232.leggiSer("COM6");  
                  
                invio = rs232.leggiSer("COM6");    // fine ricezione stringa  
                  
                situazione = belzoni \* 1000 + garibaldi \* 100 + pedonale\_mazzini \* 10 + pedonale\_belzoni;  
            }  
              
              
            // istruzioni per il collaudo rapido  
            //System.out.println("inserisci situazione");  
            //situazione = in.nextInt();  
              
              
            if (situazione == 1333){  
                lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_verde\_belzoni - pedonali rossi.png"));  
            }  
              
            if (situazione == 2333){  
                lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_giallo\_belzoni - pedonali rossi.png"));  
            }  
              
            if (situazione == 3133){  
                lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_verde\_Garibaldi - pedonali rossi.png"));  
            }  
              
            if (situazione == 3233){  
                lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_giallo\_Garibaldi - pedonali rossi.png"));  
            }  
              
            if (situazione == 1313){  
                lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_verde\_belzoni - pedonali\_mazzini\_verde.png"));  
            }  
              
            if (situazione == 2313){  
                lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_giallo\_belzoni - pedonale\_mazzini\_verde.png"));  
            }  
              
            if (situazione == 3131){  
                lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_verde\_Garibaldi - pedonali\_belzoni\_verde.png"));  
            }  
              
            if (situazione == 3231){  
                lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_giallo\_Garibaldi - pedonali\_belzoni\_verde.png"));  
            }  
        }  
          
          
    }// fine main  
      
  
    /\*\*  
     \* Create the frame.  
     \*/  
    public Semafori() {  
        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
        setBounds(100, 100, 600, 400);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        contentPane.setLayout(new BorderLayout(0, 0));  
        setContentPane(contentPane);  
          
        lblSemafori = new JLabel("semafori");  
        lblSemafori.setIcon(new ImageIcon("C:UsersMassimoDocumentsWorkSpaceESemaforisrcMicro-GT\_Semaforo\_verde\_belzoni - pedonali rossi.png"));  
        contentPane.add(lblSemafori, BorderLayout.CENTER);  
    }  
  
}

 La comunicazione è gestita da questa classe "GestioneSeriale.java"

import java.io.IOException;  
import java.io.InputStream;  
import java.io.OutputStream;  
import java.util.Enumeration;  
import javax.comm.CommPortIdentifier;  
import javax.comm.PortInUseException;  
import javax.comm.SerialPort;  
import javax.comm.UnsupportedCommOperationException;  
  
  
public class GestioneSeriale {  
    static Enumeration portList;  
    static CommPortIdentifier portId;  
    static String messageString = "0";  
    static SerialPort serialPort;  
    static OutputStream outputStream;  
    static InputStream inputStream;  
      
      
  // metodo che invia l'array x alla porta seriale com    
  public String ser(byte[] x , String com){  
    
    
              // il metodo statico getPortIdentifiers ottiene una enumerazione  
              // delle porte com presenti ed attive nel PC  
            portList = CommPortIdentifier.getPortIdentifiers();  
        while (portList.hasMoreElements()) {  
                        
  
            portId = (CommPortIdentifier) portList.nextElement();  
            if (portId.getPortType() == CommPortIdentifier.PORT\_SERIAL)   
                             
  
               if (portId.getName().equals(com)) {  
                              
  
                //if (portId.getName().equals("/dev/term/a")) {  
                    try {  
                        serialPort = (SerialPort)  
                            portId.open("SimpleWriteApp", 2000);  
                    } catch (PortInUseException e) { return "Port In Use";   }  
                    try {  
                        outputStream = serialPort.getOutputStream();  
                    } catch (IOException e) {}  
                    try {  
                        serialPort.setSerialPortParams(19200,  
                            SerialPort.DATABITS\_8,  
                            SerialPort.STOPBITS\_1,  
                            SerialPort.PARITY\_NONE);  
                    } catch (UnsupportedCommOperationException e) {}  
                    try {  
                        outputStream.write(x);  
                         
                    } catch (IOException e) {return "Failed to Send Data"; }  
  
  }  
              
  }  
     return "Data Sent";  
         
} // void ser  
    
    
//metodo che legge un numero alla porta seriale com    
 public int leggiSer(String com){  
       
     int r = 0;    // carattere letto dalla seriale  
   
   
              // il metodo statico getPortIdentifiers ottiene una enumerazione  
              // delle porte com presenti ed attive nel PC  
           portList = CommPortIdentifier.getPortIdentifiers();  
       while (portList.hasMoreElements()) {  
                       
  
           portId = (CommPortIdentifier) portList.nextElement();  
           if (portId.getPortType() == CommPortIdentifier.PORT\_SERIAL){  
                            
  
              if (portId.getName().equals(com)) {  
                   try {  
                       serialPort = (SerialPort)  
                           portId.open("SimpleWriteApp", 2000);  
                   } catch (PortInUseException e) {}  
                   try {  
                       inputStream = serialPort.getInputStream();  
                   } catch (IOException e) {}  
                   try {  
                       serialPort.setSerialPortParams(19200,  
                           SerialPort.DATABITS\_8,  
                           SerialPort.STOPBITS\_1,  
                           SerialPort.PARITY\_NONE);  
                   } catch (UnsupportedCommOperationException e) {}  
                     
                   //inputStream.read(r);  
                   try {  
                    r = inputStream.read();  
                } catch (IOException e) {  
                    // TODO Auto-generated catch block  
                    e.printStackTrace();  
                }  
  
              }  
             
           }  
       }// while  
         
    return r;  
        
} // void leggiSer  
    
    
    
  public void close(){  
      serialPort.close();  
  }  
  
}

[Scarica il pacchetto completo dell'interfaccia java (contiene anche le immagini)](http://www.grix.it/UserFiles/ad.noctis/Micro-GT%20traffic%20light%20java%20interface.zip)

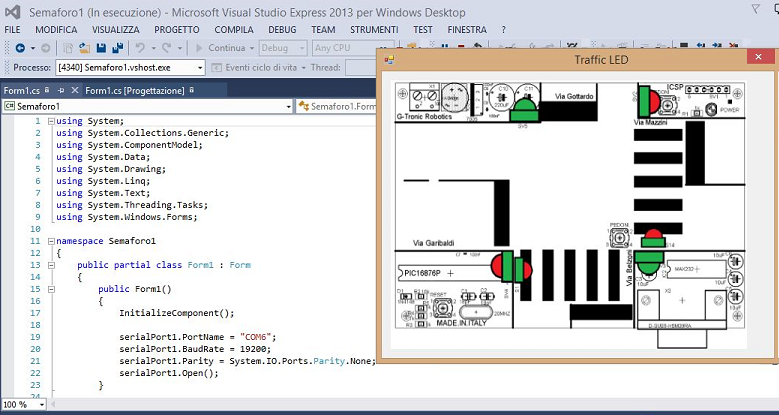
**Versione C# dell'interfaccia sul PC.**

La versione di visual studio usata per l'implementazione è la "Microsoft Visual studio Express 2013", corrispondente alla .net 4.5

http://www.grix.it/UserFiles/ad.noctis/Image/semaforo/versione%20dotnet.png

La versione di C# è la 5.

Nella prossima immagine vediamo l'interfaccia .net in esecuzione.



Bisogna porre attenzione al fatto che la versione fornita, come visibile nell'immagine, è impostata per la comunicazione seriale sulla porta COM6. Dato che spesso il cavo seriale sarà interfacciato al notebook tramite un adattatore USB questa potrà essere assegnata dal sistema operativo ad una porta diversa.

In caso di difetto di comunicazione bisogna quindi verificare il numero della porta assegnato andando in pannello di controllo, gestione periferiche, porta COM e LPT ed eventualmente riallineare la questione.

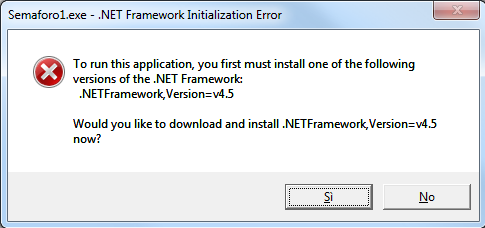
Potremmo indifferentemente cambiare il numero della porta da gestione periferiche oppure sul software (vedi la riga 19), facendo in modo che tutto coincida .

è possibile scaricare il file zippato che contiene il progetto da questo link:

[scarica il progetto semaforo1](http://www.grix.it/UserFiles/ad.noctis/Semaforo1.zip).

Potrete lanciare l'interfaccia anche facendo doppio click sul file Semaforo1.exe che si trova nella cartella bin del progetto dopo averlo scompattato.

Uno dei principali problemi del funzionamento delle applicazioni sviluppate in .net è la presenza del corretto framework sul pc in cui si lancia l'esecuzione. In caso questa non fosse presente verrà restituito l'errore mostrato nella figura.

.

Il problema viene in genere risolto creando un opportuno installer (setup) che si cura anche della verifica delle presenza e dell'eventuale installazione del framework necessario.

Nel caso si volesse eseguire l'aggiornamento al framework v4.5 in maniera manuale si dovrà cliccare su questo link e seguire le istruzioni:

<http://dotnetsocial.cloudapp.net/GetDotnet?tfm=.NETFramework,Version=v4.5>

Quando accettiamo l'istallazione automatica comparirà il seguente splash



Va tenuto in presente che l'istallazione del framework richiederà circa 3,5 giga di spazio disponibile sul disco e che sarà piuttosto lunga.

Guarda il video dell'interfaccia C# in funzione su youtube.

https://www.youtube.com

Dal prossimo link puoi scaricare il pacchetto di installazione (setup). Nei PC in cui è installato un visual studio non dovrebbe dare nessun problema. E' stato testato con Windows 8.1. Sviluppato con C# 5, e con .net 4.5

[Download traffic light setup (pacchetto di installazione completo).](http://www.grix.it/UserFiles/ad.noctis/setupSemaforo.zip)

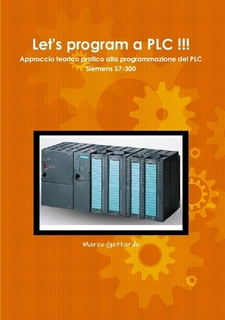
Ci sono uno miriade di varianti che possono essere proposte dagli insegnanti ai propri allievi sia per quanto riguarda il firmware che il software di interfaccia.

Vediamo come applicare il modellino alle lezioni di Automazione con riferimento diretto ai PLC di marca Siemens.

**Applicazione ai PLC Siemens S7-300.**

Per una completa comprensione dell'argomento trattato in questo paragrafo si consiglia la lettura del libro

[Let's Program a PLC !!!  edizioni www.lulu.com](http://www.lulu.com/shop/marco-gottardo/lets-program-a-plc/paperback/product-20513270.html)

[](http://www.lulu.com/shop/marco-gottardo/lets-program-a-plc/paperback/product-20513270.html) 

Consideriamo il solo PCB, pulito da ogni cosa che sia specifica del funzionamento tramite MCU (il PIC 16F876a), l'eventuale supervisione potrà essere svolta con un pannello operatore SIEMENS TP170 o altro. (per questo articolo si è usato questo modello perché unico disponibile in laboratorio, ancora in commercio ma per la verità un pò obsoleto).

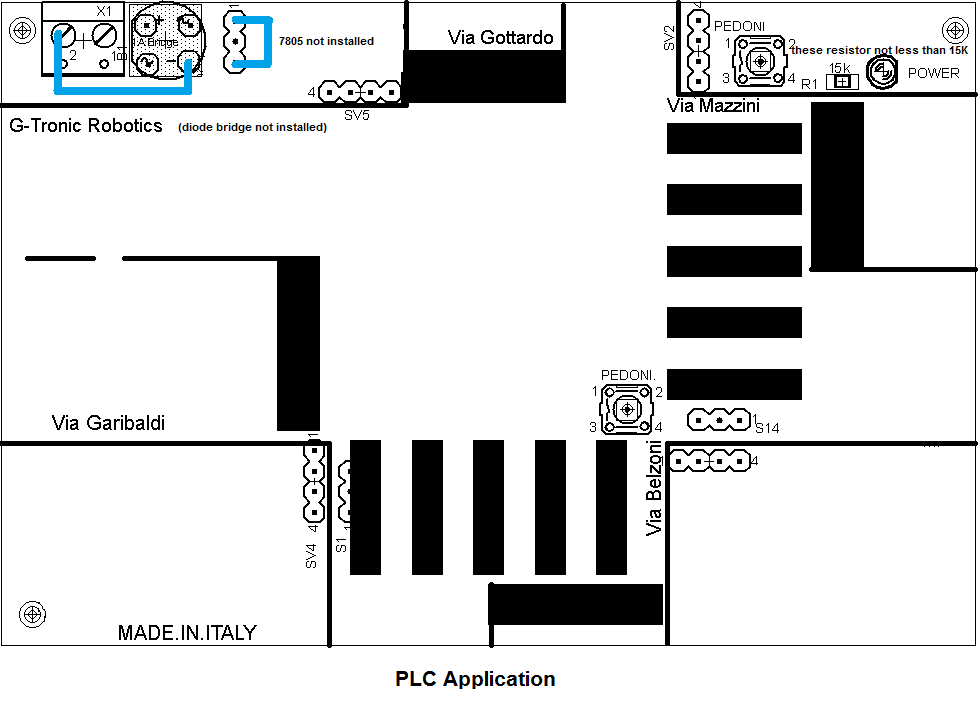
Il pannello operatore è visibile nell'immagine.



pannello operatore con membrana tattile TP170 Siemens (monocromatico).

Con l'ausilio del software WinCC flexible, creiamo un progetto che condivida le variabili del progetto Simatic sviluppato con STEP7 V5.4, di cui faremo un accenno tra poco.

Usiamo come immagine di sfondo il bitmap che ricaveremo salvando in quel formato l'immagine qui sotto, usando il paint.



Le connessioni rappresentate in blue, potranno essere cancellate perché rappresentano i ponticelli di bypass della sezione di alimentazione della scheda a microcontrollore PIC quando il modello è in uso canonico.

Una tecnica di sviluppo alternativa, per chi possiede un TP a colori, è quella di sfruttare le 5 immagini, con i semafori colorati, del paragrafo precedente, e richiamarle con la stessa logica della versione Java o C#.

Le uscite digitali del PLC vanno portate ai fori disponibili per lo zoccolo del PIC.

Al morsetto a vite va portato il cavo X2, che andrà alla linea M dell'alimentatore del PLC, altrimenti le maglie non si chiudono e non vedremo accendersi i LED dei semafori.

La linea L+, la porteremo solo per fare vedere che il modellino è collegato grazie all'accensione del LED power, ma non sarà essenziale al funzionamento.

Va ricordato che normalmente i PLC funziona con il livello industriale della tensione continua che vale 24V, quindi le resistenze del LED di accensione e dei due pulsanti della chiamata pedonale vanno riviste.

Un valore accettabile per queste tre resistenze è 15K che permetterà il transito di un accettabile milli ampere e mezzo sul LED e poco più verso massa quando verranno premuti i pulsanti.

In effetti, per il pulsante collegato a Vcc la resistenza non è neppure necessaria, mentre ci vuole, e poi sarà richiesta l'acquisizione in logica negata, per il pulsante che nello schema canonico è collegato tramite un pullup.

La porta di comunicazione non serve nello stampato dato che il pannello operatore è normalmente collegato tramite un cavo detto MPI alla porta di programmazione del PLC, come visibile qui sotto.



I disegni dei semafori potranno essere sovrapposti al bitmap che fa da sfondo usando i tag nativi del WinCC i quali sono presenti nella libreria abbinata al modello TP170.

Per quei pannelli TP, che ancora dispongono della porta seriale, come quello che è qui mostrato, la comunicazione non può scendere sotto ala velocità di 19200.

Lo scambio dei dati avviene usando un'area condivisa tra la CPU del PLC e il pannello operatore il quale mantiene il programma nella propria memoria. Tale area di memoria condivisa è genericamente chiamata DB, e viene programmata nel sistema a lato PLC, tramite il SIMATIC Step 7 (non il WinCC che invece opera sul lato TP).

N.B. Il ciclo semaforico è semplificato per questioni di chiarezza di esposizione.

Facciamo una attribuzione più comoda per l'esempio che ci accingiamo a presentare.

Strada  Garibaldi->Mazzini  diventa Semaforo A1 e Semaforo A2

Strada Belzoni -> Gottardo diventa Semaforo B1 e Semaforo B2.

Da questo otteniamo la seguente tabella a cui assegniamo i tempi di permanenza di ogni situazione di transito.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **RA**  **A124.5** | **GA**  **A124.4** | **VA**  **A124.3** | **RB**  **A124.2** | **GB**  **A124.1** | **VB**  **A124.0** |
| **33** 20sec | 1 | 0 | 0 | 0 | 0 | 1 |
| **35**5sec | 1 | 0 | 0 | 0 | 1 | 1 |
| **12** 20sec | 0 | 0 | 1 | 1 | 0 | 0 |
| **28**5sec | 0 | 1 | 1 | 1 | 0 | 0 |
| **X** | 1 | 0 | 0 | 0 | 0 | 1 |

la tabella di verità mostra lo stato dei pali semaforici e i tempi ciclici di permanenza.

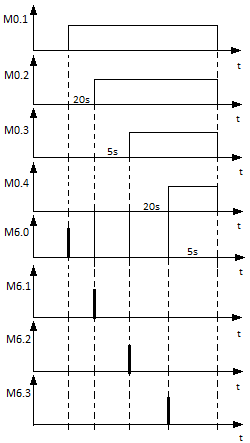
Ricaviamo i diagrammi corsa tempo.

Vediamo come generare dei segnali impulsivi, trigger o segnali di sincronia, che saranno utili per richiamare un blocco solo una volta per la durata di un ciclo di scansione.

Come vedremo i blocchi FC2,FC3,FC4,FC5 dovranno eseguire una sola volta il prelievo della variabile dal blocco dati DB1 quindi li richiameremo in funzione del fronte di salita che si presenta al cambio di stato dei Merker M0.1, M0.2, M0.3, M0.4 che invece rimangono alti per tutto il resto del programma fino alla generazione dell’impulso di  reset che porta il sistema a zero macchina.

Poi il ciclo si ripete.   La durata complessiva del ciclo semaforico, secondo le specifiche è di 30 secondi, ma per accelerare la fase di test, da eseguirsi con PLC SIM (un tool integrato nel pacchetto Step 7 che permette di simulare il PLC), si accorceranno i tempi da 20 a 4 secondi e da 5 a 2 secondi.

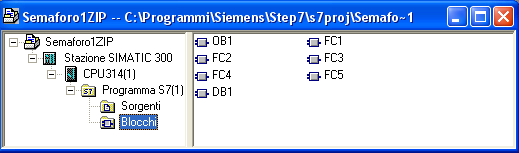
Nel complesso, generiamo una rampa temporale con un contatore e la intercettiamo su quattro soglie che avranno l’effetto di generare l’impulso di trigger visibile ne diagramma corsa tempo sottostante.



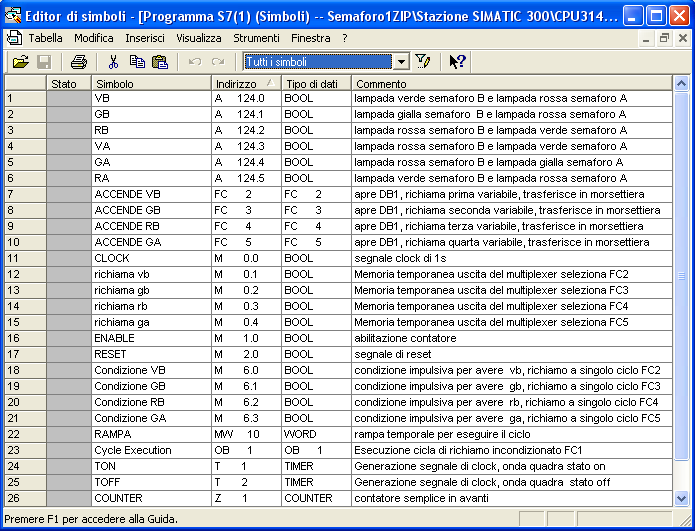
Il blocco organizzativo OB1 contiene solo il salto incondizionato al blocco FC1, ovvero la seguente sola riga di codice:

UC FC1 //contenuto di OB1 in esecuzione ciclica

L’anteprima del progetto è nella prossima immagine, essa è ricavata dal simatic manager, e configurando il PLC sul catalogo hardware.



La stesura della tabella delle attribuzioni è un passaggio fondamentale per lo sviluppo di qualsiasi automazione. Questa va compilata prima di iniziare a programmare i blocchi dato che le etichetti potranno essere richiamate semplicemente facendo tasto destro del mouse sopra ad ogni etichetta di contatti , timer, contatori, ecc.  
Il collegamento tra l’editor e la tabella dei simboli è sempre attivo in qualunque finestra , ovvero blocco, stiate programmando.  
È possibile inserire nuovi simboli, se mancanti in tabella, senza aprire L’editor dei simboli, semplicemente facendo tasto destro sull’indirizzo mancante e richiamando la voce “modifica simboli”.



La tabella dei simboli deve essere il più dettagliata possibile e nel campo commento riportare lo stato di riposo del cablaggio hardware dei contatti, ovvero se questi sono stati cablati in apertura (N.C.) i in chiusura (N.A.).

Si ricorda che è bene indicare lo stato dinamico del contatto e non la sua condizione statica, ecco che un contatto chiuso  N.C. va ad aprirsi durante l’azione, quindi lo chiameremo “in apertura”.

Dopo avere concluso la tabella dei Simboli possiamo, dal simatic manager, cliccare sul blocco FC1 e iniziare l’editazione del programma come riportato in seguito.

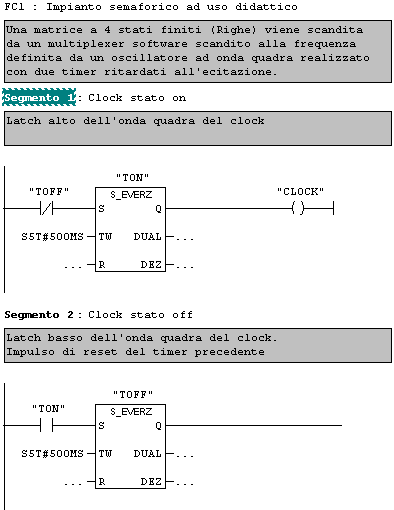
Come spiegato nel libro "Let's Program a PLC !!!", è possibile assegnare a dei Merker la funzione di lampeggio, ma in questo programma usiamo i due timer ritardati all’eccitazione.

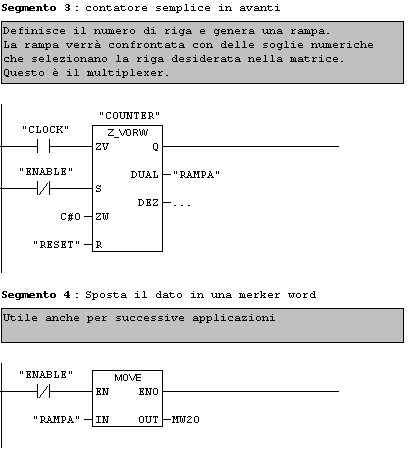
I motivi sono due:

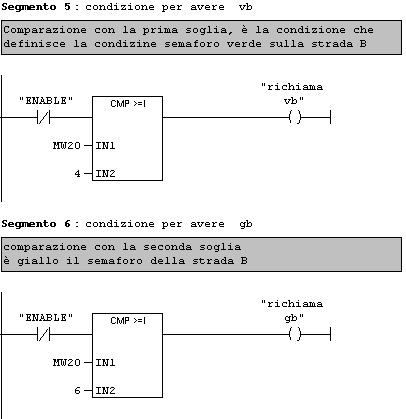
1.       Usiamo i Timer per scopi didattici, infatti un utile esercizio sarà quello di ricavare i diagrammi di eccitazione ricavando che il primo timer esegue l’onda quadra mentre il secondo solo un impulso di trigger usato per resettare il primo.

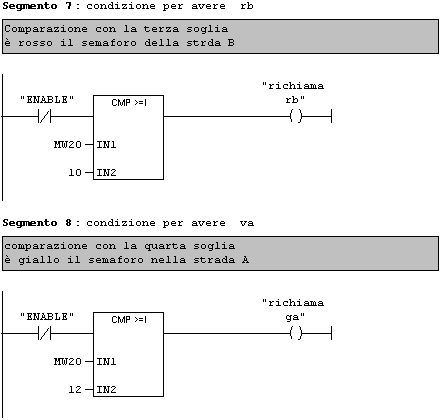
2.       Le variabili di preset temporali, qui fissate a mezzo secondo, potranno essere caricate da un DB oppure direttamente da un cursore virtuale “Tag” su un pannello operatore tattile, rendendo regolabile dall’utente dell’impianto il tempo impostato nel clock. Questo viene lasciato come esercizio al lettore.

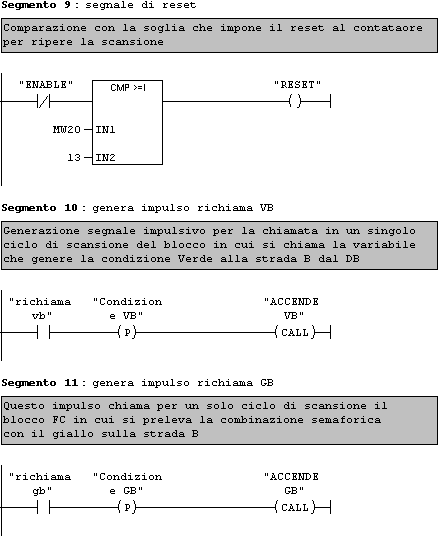
Apriamo ora il blocco FC1 e inseriamo quanto segue, i chiari  commenti ad ogni segmento e l’uso della visualizzazione simbolica rendono il sorgente di facile comprensione.

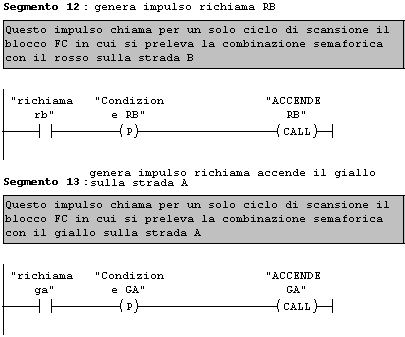






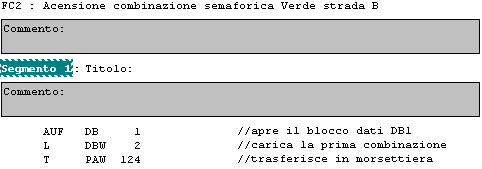


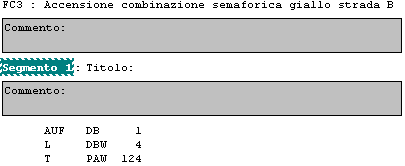


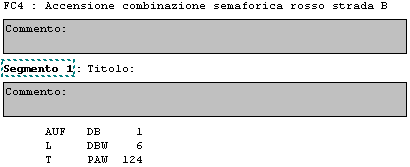


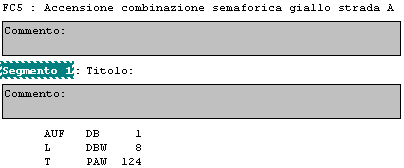
Vediamo adesso in quattro blocchi, FC2,FC3,FC4 , essi aprono il DB1, prelevano la parola dati all’indirizzo base indicato sulla colonna di destra del blocco dati stesso e la copiano nell’immagine di periferica di uscita.

L’effetto è quello di avere l’accensione della combinazione di tutte le lampade semaforiche con un unico comando di trasferimento.

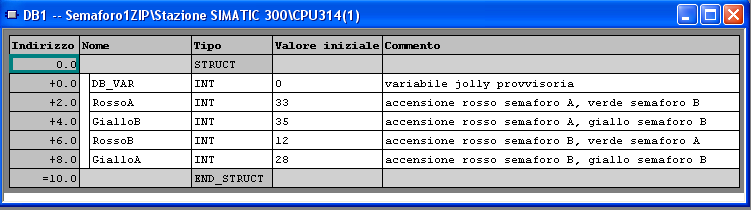








Il blocco dati, “DB1”, da cui si prelevano le combinazioni sotto forma di costanti numeriche intere ha il seguente aspetto.



I numeri interi, immessi nella colonna "valore iniziale" vengono prelevati e inviati in forma binaria alla morsettiera d'uscita digitale del PLC a cui è collegato il modellino del semaforo.

Questo esempio è molto semplice però permette, da un unico punto si accesso, ad esempio il TP170, di variare i tempi di ciclo, ad esempio variando l'ingresso dei due timer presenti ai segmenti 1 e 2 (quando si voglia mantenere la proporzione tra le viabilità abilitate), oppure anche modificando i valori di conteggio degli zhaller (contatori), o dei comparatori.

Se non si possiede un TP, il programma funziona comunque e i tempi possono essere cambiati modificando e aggiungendo delle variabili sul DB1.

Nella sostanza il programma qui fornito è un ottimo template che si presta bene a una miriade di varianti.

Un ottimo spunto per molte tesine scolastiche.

Approfondimenti in tema si trovano nel libro indicato sopra.

Un esempio di una lezione di PLC tenuta al centro culturale ZIP (zona industriale di Padova) è mostrato nel video qui sotto.  Le riprese sono state fatte con il metodo "Buona la Prima" e senza preliminari accordi con gli allievi che si sono semplicemente presentati a lezione armati di telecamera.

https://www.youtube.com  
  
nbsp;

**Varianti e suggerimenti per chi dispone del solo PCB.**

Come tutti sappiamo l'Italia sta vivendo un periodo economicamente tragico quindi risparmiare anche un solo euro potrebbe essere significativo.

Ecco alcuni consigli per minimizzare la spesa ed avere comunque il modellino funzionante:

1. Se disponente di postazioni di programmazione ICSP già in uso agli studenti, potete non assemblare la porta di comunicazione DB9 con il MAX232 e i suoi 4 elettrolitici. In questo caso significa che non volete interfacciarvi al PC perchè non è previsto l'insegnamento di un linguaggio ad alto livello (java, python, .net, ecc) con cui realizzare una supervisione.
2. Se non disponete di alimentatori di recupero da collegare al morsetto a vite, potete non assemblare tutta la parte dell'alimentatore e entrare con un vecchio cavo USB, ad esempio recuperato da una stampante o un mouse. Potrete tagliare i fili verde e bianco e usare solo il rosso da saldare sulla piazzola predisposta per il positivo del condensatore elettrolitico C11 e il nero da collegare al posto del negativo del medesimo condensatore.
3. Potrete in questo modo alimentare la scheda tramite una porta USB libera del PC.
4. Se ritenete che ne valga il risparmio potrete non assemblare i pali semaforici per l'attraversamento pedonale  e i relativi pulsanti e pullup ovvero assemblarne uno solo. Nel primo caso perdete la possibilità di esporre la tecnica di programmazione con interrupt che è il focus di questo esercizio.

**Note di fine articolo.**

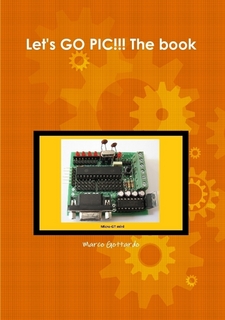
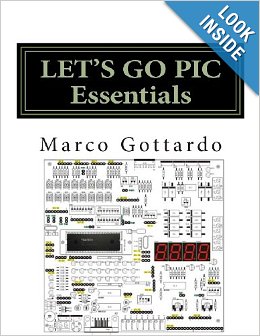
Ufficialmente il protocollo RS232 definisce il protocollo di interfacciamento tra un DTE (data terminal equipment) e un DCE (data communication equipment) tramite il quale avviene uno scambio di dati. I segnali essenziali sono i pin 3 e 2 rispettivamente TX e RX del lato DCE.Va ricordato che secondo le regole impostate nelle telecomunicazioni il DTE è il terminale computer mentre il DCE rappresenta la periferica di comunicazione quale era il modem seriale o una periferica quale il mouse e in questo caso il semaforo miniaturizzato.

**Bibliografia:**

Le nozioni utili o di approfondimento usate in questo articolo sono contenute nel libro "Let's GO PIC!!!" e il libretto in formato e-book "[Primi passi con i PIC sul sistema Micro-GT mini](http://www.lulu.com/shop/marco-gottardo/primi-passi-con-i-pic-sul-sistema-micro-gt-mini/ebook/product-21607555.html)" **Gratuito**, segnalo che esiste una versione ridotta "Let's GO PIC!!! essential" orientata alle scuole.

L'ebook è gratuito e per scaricarlo dal link si deve effettuare la login su www.lulu.com dopo avere creato un account.

Per vedere le anteprime cliccare sulle copertine.

[](http://www.lulu.com/shop/marco-gottardo/lets-go-pic-the-book/paperback/product-20373044.html)      [](http://www.amazon.com/LETS-GO-PIC-Essentials-Micro-GT/dp/1492851523/ref=pd_sim_sbs_b_1/177-1249559-8094700)   [](http://www.lulu.com/shop/marco-gottardo/primi-passi-con-i-pic-sul-sistema-micro-gt-mini/ebook/product-21607555.html)

*"Questo articolo contiene informazioni e link divulgativi sui sistemi Micro-GT anche assimilabili a pubblicità senza alcun guadagno economico per* [*grix.it*](http://grix.it/) *ma al solo fine di agevolare, per chi lo voglia, la reperibilità dei componenti utilizzati e/o approfondirne gli argomenti trattati."*

[*Questo progetto è ridistribuibile secondo i termini di licenza*](http://creativecommons.org/licenses/by-sa/3.0/deed.it)*[http://www.grix.it/UserFiles/Max%20Pet/Image/numeri%20primi/GNU.gif](http://creativecommons.org/licenses/by-sa/3.0/deed.it)Creative Commons Attribuzione-Condividi allo stesso modo 3.0 Italia*