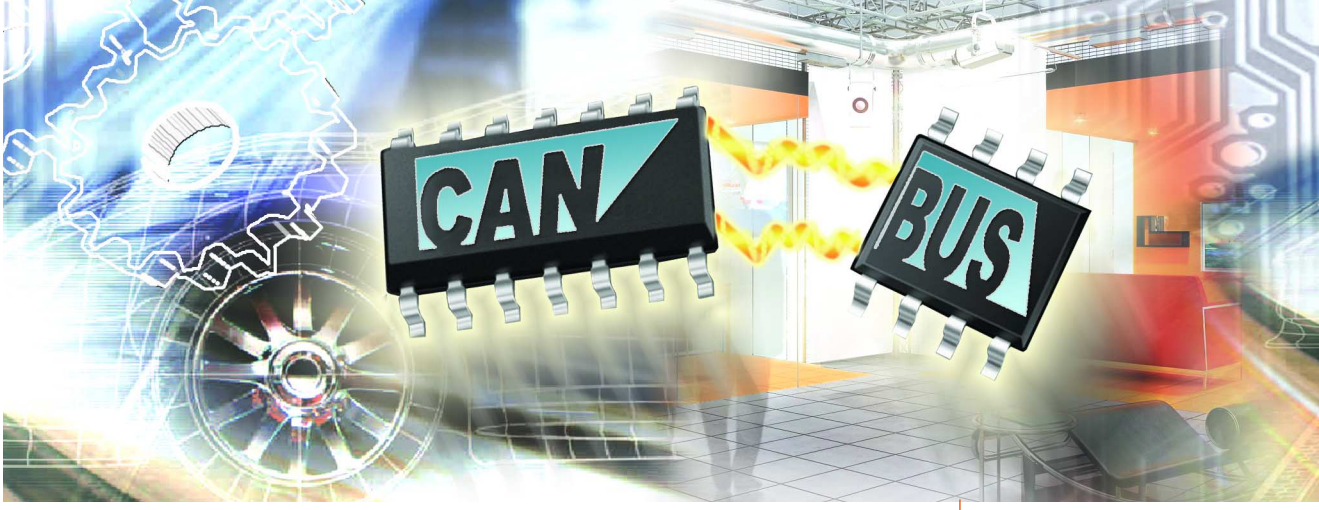


a cura di Carlo Tauraso

Alla scoperta del CAN-BUS

Nato come protocollo di comunicazione seriale per fare comunicare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. Parte qui il nostro Corso sulla materia, nel quale, alla teoria di funzionamento, affiancheremo esempi applicativi mirati all'automazione per la casa.

1

Nel corso degli anni si è assistito ad una massiccia penetrazione dell'elettronica nell'ambiente automobilistico, a partire dall'ambito sportivo: chi segue le gare di Formula1 si sarà più volte chiesto quanto l'elettronica possa contribuire al successo di un pilota. Ebbene, troverete la risposta una volta acquisiti gli elementi per farvi la giusta opinione, elementi che esponiamo a partire da questa pagine, dove introduciamo un argomento di grande attualità quale è il CAN-Bus: si tratta dello standard per la gestione elettronica del motore nelle moderne autovetture, quello adottato dalle cosiddette "centraline", tanto decantate dai patiti del "tuning" automobilistico. Ma non solo: nato per l'automobile, per le sue proprietà, il CAN-Bus viene utilizzato anche nella robotica e nel settore della domotica, tanto in voga in questi ultimi anni in cui si parla di elettrodomestici intelligenti, tutti collegati insieme e gestiti da un unico apparato che ne ottimizza i consumi facendoli funzionare negli orari migliori e in base al loro assorbimento, evitando il sovraccarico della linea elettrica.

Chi tra voi ha avuto la sventura di subire un danno alla centralina dell'auto oppure ha tentato di installare un accessorio supplementare su una BMW appena uscita dal concessionario, avrà sicuramente rimpianto i tempi del "fai da te", quando bastava guardare i collegamenti e non era necessario scomodare un tecnico elettronico esperto per capirci qualcosa. Oggi, per aprire consapevolmente il vano motore dobbiamo avere qualche nozione di programmazione di microcontrollori e, soprattutto, conoscere in che modo l'elettronica riesce ad integrarsi con la meccanica. Infatti nei motori montati nelle vetture del- >

l'ultimo decennio molti elementi a comando meccanico sono stati soppiantati da attuatori elettrici gestiti da un controllore elettronico che li aziona in base ai valori dei parametri letti da opportuni sensori: ad esempio, l'acceleratore non è più comandato da un filo mosso dal pedale corrispondente, ma da un servocomando controllato sia da un potenziometro applicato al pedale, sia dalla centralina sulla base di parametri quali la portata dell'aria aspirata, la temperatura del liquido di raffreddamento, l'inserimento del condizionatore ecc. Ancora, l'anticipo di iniezione (ormai riguarda tutti i motori: diesel e a benzina) viene regolato non da sistemi centrifughi ma da servocomandi, sulla base del rilevamento del numero di giri, della temperatura dell'aria aspirata e di quella del combustibile.

Questo corso si prefigge di descrivere nel dettaglio il funzionamento del CAN-Bus, senza tralasciare, come è ormai tradizione della nostra rivista, la parte pratica: costruiremo, infatti, una demoboard che ci permetterà di sperimentare dal vivo l'efficienza di tale sistema. In particolare, faremo vedere come sia possibile creare una rete di dispositivi che svolgono dei compiti in maniera integrata, come avviene nella nostra autovettura o negli impianti di automazione per la casa. Lo schema potrà essere duplicato per creare ulteriori nodi e aggiungere al sistema funzionalità sempre più sofisticate.

Non mancheranno riferimenti al mondo reale, in maniera tale da poter osservare le applicazioni pratiche dei nostri esperimenti. Naturalmente ci concentreremo, in particolare, sullo sviluppo firmware, il vero cuore pulsante tanto dell'elettronica automobilistica, quanto della domotica: va infatti premesso che, hardware a parte, una centralina può svolgere diverse funzioni a seconda di come viene programmata.

Per capire quanto la programmazione sia importante, sempre restando nel campo automobilistico pensate che ci sono stati casi in cui l'impossibilità di aggiunta di un accessorio in una vettura derivava da un controllo previsto nel firmware ed eseguito dalla centralina; in casi del genere il problema è stato risolto dal meccanico semplicemente riprogrammando la memoria flash della centralina con una nuova versione di software che non prevedesse tale controllo. E il tutto senza… sporcarsi le mani.

E che dire di quei "furbi" che modificano i parametri di funzionamento del motore per aumentarne le performance? Altro che spoiler e carburanti particolari: basta un buon editor esadecimale ed è possibile far guadagnare anche qualche decina di cavalli. Per gli esempi che faremo useremo, come microcontrollori, quelli della famiglia PIC18F4XX di Microchip. Come linguaggio, il nostro riferimento sarà il C18 che abbiamo già incontrato nel Corso PIC-USB, ma che ora sarà necessario approfondire. Raccomandiamo quindi di seguire con attenzione il tutorial relativo a questo linguaggio la cui pubblicazione prende il via in questo stesso fascicolo. Iniziamo con alcuni essenziali concetti teorici di base.

CAN Controller Area Network

Il CAN-Bus nacque in Germania nella metà degli anni '80 del secolo scorso ad opera di Robert Bosch (quello della famosa Bosch, leader nel settore degli impianti d'iniezione elettronica e degli equipaggiamenti elettronici per autoveicoli) il cui intento era introdurre un sistema di comunicazione seriale robusto e semplice, che permettesse di rendere le automobili più affidabili e il loro motore più efficiente e quindi capace di ridurre sia il consumo di carburante che le emissioni inquinanti. Nel corso degli anni il sistema ha subito delle evoluzioni soprattutto in materia

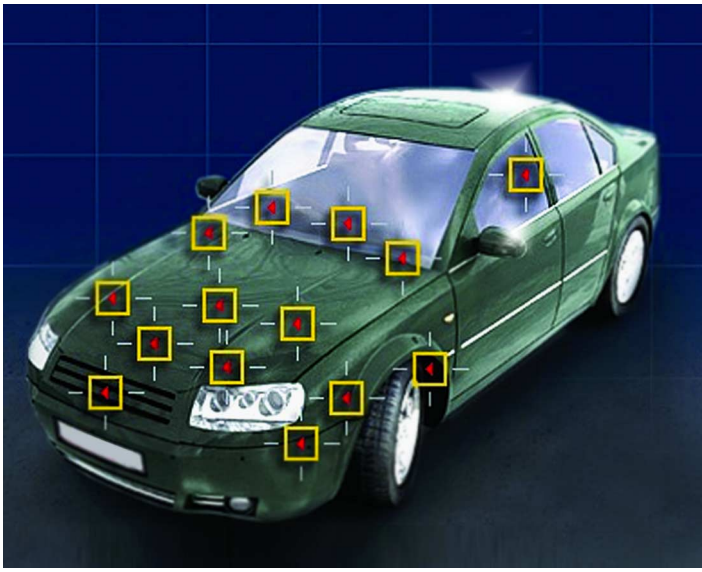
Fig. 1

di velocità e controllo degli errori ed ha guadagnato credibilità anche nell'ambiente industriale, specie nel campo della robotica. Tutte le definizioni e le spiegazioni di funzionamento sono state raccolte in alcune specifiche molto precise; in questa trattazione ci riferiremo sia alla versione iniziale (1.2) che alla sua successiva evoluzione, ossia la 2.0. Organismi internazionali come la ISO (International Standards Organization) e la SAE (Society of Automotive Engineers) hanno provveduto ad emanare diversi documenti che

descrivono protocolli basati sul CAN e utilizzabili in campi particolari: si veda ad esempio il J1939 per le applicazioni su camion e autobus o l'ISO11898 per quelle ad alta velocità. Per poter spiegare meglio il funzionamento di questo bus ci riferiremo al modello ISO/OSI (Open System Interconnection) che viene utilizzato laddove è necessario chiarire l'interoperabilità tra dispositivi all'interno di una rete. In generale, il modello è organizzato in sette livelli funzionali, ognuno dei quali ha delle incombenze specifiche. Vale il principio generale secondo cui ogni livello offre servizi al livello più elevato, nascondendo i dettagli su come tali servizi vengano implementati. Ogni livello che chiamiamo *n* su un dispositivo porta avanti una conversazione con il livello *n* di un altro dispositivo, grazie ai servizi implementati dai livelli inferiori.
Per esplicitare il concetto si pensi ad un caso molto semplice ma esemplificativo: il direttore di un'azienda italiana (livello *n*) vuole parlare con il direttore di un'azienda cinese; per farlo scrive una lettera e la porta al suo traduttore (che chiamiamo livello *n*-1, dato che nella gerarchia aziendale ne esegue le direttive) e quest'ultimo traduce il contenuto della lettera e lo passa alla segretaria (livello *n*-2, perché esegue le direttive provenienti da un livello sotto *n*). Ella provvede a redigere un fax e ad inviarlo tramite linea telefonica (livello fisico) alla sua omologa cinese; la segretaria cinese (livello *n*-2) riceve il fax, lo passa al traduttore della sua azienda (livello *n*-1) che lo traduce e lo passa al direttore cinese (livello *n*). Come si vede, la comunicazione tra i due direttori (cioè i due livelli *n*) avviene in maniera completamente trasparente, senza cioè che essi debbano conoscere i dettagli realizzativi dell'operazione (traduzione italiano/cinese, compiti delle segretarie e funzionamento del fax). Il tutto si svolge attraverso la corresponsione di servizi tra i vari livelli a partire dal più alto fino ad arrivare al livello fisico per il mittente e viceversa per il destinatario. Se preferite, diversamente da una struttura in cui un elemento di controllo agisce su tutti quelli che lo circondano, qui si lavora per gerarchia e ogni livello esegue un proprio compito incaricando il livello sottostante e rendendo conto a quello immediatamente superiore. Ovviamente il tutto funziona presupponendo che ogni livello svolga il compito affidatogli. I livelli ISO/OSI sono riassunti nel diagramma di Fig. 1. Non vogliamo tediarvi con la spiegazione della funzione di ciascun livello (vi rimandiamo a testi come l'ottimo Computer Networks di A. Tanenbaum, edizioni Pearson, Prentice Hall, disponibile nella versione italiana edita col titolo "Reti di calcolatori") anche perchè il CAN sviluppa soltanto gli ultimi due livelli: quello Fisico e quello Data Link, lasciando ai progettisti una certa libertà nell'implementazione dei livelli superiori, al fine di ottimizzare il funzionamento del bus caso per caso.

Livello Fisico: si occupa principalmente delle caratteristiche meccaniche, elettriche, del mezzo fisico e dell'interfaccia di rete; in generale le sue incombenze sono quelle di garantire la corretta trasmissione di ciascun bit sul canale di comunicazione, pertanto ha a che fare con la codifica e decodifica degli stessi (livelli di tensione assegnati ai valori 0 e 1 ecc.) e della loro temporizzazione (durata di ogni bit).

Livello Data Link: fa in modo che il mezzo trasmittente appaia al livello superiore come una linea di trasmissione esente da errori non rilevati; le sue incombenze sono relative alla suddivi-



sione in frame, alla rilevazione degli errori ed alla loro segnalazione, alla regolazione del traffico per evitare l'overload dei dispositivi, al filtro dei messaggi in arrivo e alla loro validazione. In particolare, nel nostro caso la rete sarà di tipo broadcast, quindi si avrà la condivisione del mezzo trasmittente tra più entità che avranno il medesimo diritto di accedervi. Pertanto è stato previsto un sottolivello chiamato MAC (Medium Access Control) che permette la gestione dell'arbitraggio del canale, ossia la scelta dell'unità che >

in un certo momento può trasmettere. Notate che quest'ultimo è lo stesso utilizzato nelle interfacce di rete per computer. Questa prima spiegazione ci permette di introdurre un modello particolare che è stato utilizzato nelle prime specifiche ufficiali scritte da Bosch e che si basa su quattro livelli fondamentali: Livello Fisico, Livello Transfer, Livello Object, Livello Application. Se lo mettiamo a confronto con quello ISO/OSI, vediamo che in pratica il livello Data Link viene suddiviso nel Transfer Layer e nell'Object Layer, in maniera da distinguere le funzioni proprie del trasporto di informazioni come la divisione in frame e il controllo errori, da quelle più vicine al dispositivo come la gestione del messaggio ricevuto. Infine, si è voluto raggruppare in un unico livello generico, l'Application Layer, tutti gli strati superiori. Il livello fisico, invece, rimane tale e quale alla definizione data nel modello di riferimento ISO/OSI. D'ora in poi faremo riferimento solo al modello così modificato, anche perchè è quello utilizzato in tutta la documentazione tecnica relativa al CAN Bus.
La situazione può essere riassunta attraverso il diagramma di Fig. 2.
Dalla figura appaiono chiaramente le funzioni di ciascun livello. Nella versione successiva delle specifiche CAN (2.0) viene introdotta una serie di feature supplettive che vanno a completare il quadro, soprattutto per quanto riguarda il controllo degli errori. Il modello di riferimento viene avvicinato maggiormente a quello ISO/OSI rinominando come LLC (Logical Link Control) il livello Object e come Transfer il MAC (Medium Access Control) proprio come avviene nella descrizione degli standard Ethernet.
Dal punto di vista funzionale, però, le cose non cambiano e le incombenze dei vari livelli vengono integrate, anzichè essere modificate radicalmente. La tipologia di errori rilevati aumenta ma rientra sempre nelle funzioni tipiche del Livello Transfer. Molto interessante è l'introduzione del cosiddetto "Fault Confinement" cioè la capacità di isolare i dispositivi che entrano in una situazione di malfunzionamento permanente, allo scopo di proteggere complessivamente la rete. Il nodo è in grado di passare da una modalità di funzionamento normale a una disconnessione totale (bus-off) a seconda della gravità dell'errore. In questo modo l'ampiezza di banda viene preservata, condizione estremamente importante nel caso di sistemi critici. Questa feature viene assegnata al LLC e quindi al livello Object. Il diagramma presentato vale comunque, visto che la gestione dello status del dispositivo è assimilabile a tale funzionalità.

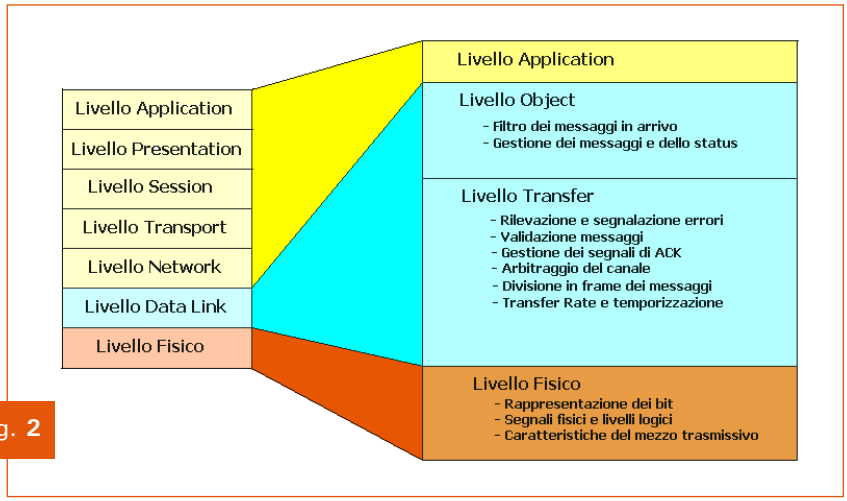


Fig. 2

Le regole di base

Il CAN si basa sullo scambio di messaggi di diverso tipo aventi però tutti un formato e una lunghezza ben definite; nel momento in cui il bus è libero, qualsiasi stazione può iniziare ad inviare un nuovo messaggio, quindi ci troviamo in un ambiente multimaster. Chi ha seguito il Corso PIC-USB si sarà subito accorto che la situazione è ben differente dal concetto di centralità dell'host che avevamo descritto in quella occasione: qui siamo in democrazia! Ciascun nodo non ha un indirizzo o un identificativo univoco, fatto che comporta delle conseguenze consistenti: innanzitutto è possibile aggiungere un nodo alla rete in qualunque momento senza fissare delle informazioni di configurazione compatibili con la rete. Si pensi a quando si mette in rete un PC ed è pertanto necessario assegnare alla sua interfaccia un indirizzo ed una maschera di sottorete ben definiti. In pratica il nodo introdotto inizia immediatamente a scambiare messaggi con gli altri dispositivi della rete senza la necessità di programmarlo opportunamente per farlo riconoscere dagli altri. I messaggi sono contraddistinti da un identificatore che definisce non l'indirizzo del destinatario, bensì il

significato dei dati contenuti nel messaggio. Ciò fa sì che ciascun nodo possa stabilire in base ad un suo filtro quale messaggio considerare e quale no. È perciò possibile realizzare un ambito di multicast, in cui un gruppo di dispositivi reagisce elaborando il medesimo messaggio.
Si pensi ad una serie di attuatori che effettuano una determinata operazione di messa in sicurezza a seguito di una segnalazione di allarme di una sonda. Inoltre, il valore dell'identificatore del messaggio ne stabilisce anche la priorità. Il bit rate in un bus CAN è sempre fisso.
In una rete CAN, teoricamente non esiste un numero massimo di nodi. Naturalmente gli unici limiti sono quelli dettati dall'incremento dei ritardi di propagazione e dalle caratteristiche elettriche del mezzo trasmissivo utilizzato. Infine, si consideri che ciascun nodo verifica la consistenza di ogni messaggio ricevuto segnalando opportunamente quelli non consistenti. Vedremo che il controllo degli errori e la segnalazione delle situazioni di failure sono decisamente importanti.

Il protocollo

Dopo aver descritto il modello di funzionamento e le regole di base, iniziamo ad entrare nei dettagli: il CAN è un protocollo del tipo CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Vediamo che cosa significa:
Carrier Sense: prima di inviare un messaggio, ogni nodo deve monitorare la linea di comunicazione fino a rilevare un certo periodo di inattività.
Multiple Access: una volta rilevato il periodo di inattività, ciascun nodo ha la stessa possibilità di accedere al canale di comunicazione.
Collision Detection: nel caso in cui tentino contemporaneamente la trasmissione, due nodi rilevano la collisione e si comportano di conseguenza.
Considerando che tutti i nodi possono utilizzare il canale in maniera paritaria, l'arbitraggio e il rilevamento delle collisioni è essenziale. Nei sistemi CAN si utilizza un arbitraggio di tipo "non-destructive bitwise" che permette di mantenere intatti i messaggi anche nel caso in cui si verifichi una collisione. Per farlo correttamente, il bus può assumere due livelli logici complementari: dominante o recessivo. In pratica i bit a 0 sono dominanti mentre quelli a 1 sono recessivi. Il nodo che sta trasmettendo deve monitorare la linea per stabilire se il bit logico che sta per inviare ha lo stesso valore di quello presente nel canale di comunicazione. Durante l'arbitraggio, un bit dominante ha sempre la meglio su un bit recessivo, quindi minore è il valore dell'identificatore del messaggio, maggiore è la sua priorità: ciò perché, siccome lo zero prevale sull'1, più è alto il valore binario, più sono numerosi i livelli logici 1 presenti. Pertanto, se un nodo tenta l'invio di un bit a 1 e trova che la linea assume uno stato logico dominante, sicuramente è stata realizzata una collisione; a questo punto il nodo sospende immediatamente la trasmissione.
L'altro messaggio (che ha una maggiore priorità) continua ad essere inviato fino al termine. Il nodo che ha perso il controllo del canale tenterà di nuovo la trasmissione dopo aver rilevato un certo periodo di inattività sul bus. Si noti che in tutta questa procedura la collisione non comporta la distruzione del messaggio a maggiore priorità, né il rallentamento della trasmissione.
Il protocollo usato dal CAN-Bus è basato sullo scambio di messaggi, che vengono identificati per priorità e contenuto e non per l'indirizzo del destinatario; pertanto i messaggi vengono indirizzati non a un nodo in particolare, ma a tutti quelli presenti sulla rete. Ciascun nodo deciderà di volta in volta se il messaggio ricevuto dovrà essere elaborato oppure no. Qualcuno obietterà che tutto ciò comporta sicuramente un traffico di rete piuttosto sostenuto; ed è vero. Ecco perché, per ovviare a tale situazione, il CAN Bus prevede la possibilità che, all'occorrenza, un nodo ne interroghi uno in particolare, invece di lasciare le cose al caso. Questa modalità si chiama RTR (Remote Transmit Request) e fa sì che un nodo, anziché attendere passivamente delle informazioni da un altro dispositivo, le richieda direttamente quando gli occorrono.
Possiamo rendere più evidente il concetto pensando alla centralina della nostra automobile: per questioni di sicurezza, alcuni sensori (come quelli che controllano gli air-bag) dell'impianto elettrico risultano sicuramente più critici rispetto ad altri (per esempio quello che legge la tensione della batteria) nel senso che il loro corretto funzionamento è più rilevante di quanto non lo sia quello di altri che, anche se si guastano, consentono comunque di viaggiare in sicurezza. Stabilendo una certa scala di importanza, si può quindi stabilire che le sonde a bassa priorità vengano interrogate solo quando necessario via RTR mantenendo una comunicazione più frequente verso le sonde più importanti. È chiaro che quest'opportunità permette di mantenere sotto controllo il traffico di rete, favorendo la trasmissione >

dei messaggi fondamentali per il funzionamento complessivo del sistema.

Le tipologie di messaggi

Un CAN Bus può veicolare diverse tipologie di messaggi (o frame) che vengono contraddistinti ciascuno da un campo identificatore. Precisiamo immediatamente che dalla versione 2.0 esistono due formati di messaggi che differiscono per il numero di bit del campo identificatore: i frame con un identificatore lungo 11 bit sono detti "Standard" mentre quelli con identificatore lungo 29 bit vengono detti "Extended". Il sistema usa quattro tipologie di messaggi che riassumiamo in tabella 1.

Tabella 1

Tipologia	Descrizione
DATA FRAME	Contiene i dati da trasferire tra un mittente ed uno o più destinatari.
REMOTE FRAME	Viene usato da un nodo per interrogare un altro dispositivo e richiedere l'invio di un data frame con il medesimo identificatore e contenente le informazioni necessarie. Implementa la modalità RTR (Remote Transmit Request).
ERROR FRAME	È emesso da un nodo quando rileva un errore.
OVERLOAD FRAME	È utilizzato per aumentare il ritardo tra un data frame (o un remote frame) e quello successivo.

Ogni data frame o remote frame è separato da quello successivo da un apposito ritardo chiamato INTERFRAME SPACE. Si tenga ben presente che è possibile utilizzare il formato "Standard" e quello "Extended" in entrambe le tipologie, pertanto un dispositivo certificato CAN 2.0 deve riuscire a gestire identificatori sia a 11 che a 29 bit. Nel prosieguo definiamo come nodo trasmittente il dispositivo che genera un messaggio e lo immette sul bus. Esso rimane trasmittente finché, terminato l'invio, il canale ridiventa libero (stato di Idle) oppure, durante l'invio, perde il possesso del canale. Un nodo ricevente è un qualsiasi dispositivo in rete che non è trasmittente durante l'occupazione del bus. Analizziamo, ora, le caratteristiche di ciascuna tipologia di messaggio.

DATA FRAME

Un data frame è composto da 7 campi chiamati: START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD, CRC FIELD, ACK FIELD, END OF FRAME. Ognuno di essi ha una funzione specifica; vediamo come sono disposti all'interno di questa tipologia di messaggio (Fig. 3). Vediamo nel dettaglio il significato di ciascun campo:

START OF FRAME (SOF): Definisce l'inizio sia dei data frame che dei remote frame e consiste in un unico bit dominante. Ogni nodo può iniziare una trasmissione soltanto nel momento in cui rileva un periodo di inattività del bus.

ARBITRATION FIELD: La sua struttura è differente a seconda dell'utilizzo del formato "Standard" o "Extended".

Nella prima versione esso è costituito da un campo Identificatore lungo 11 bit e da un singolo bit chiamato RTR, che permette di stabilire se il frame è relativo all'interrogazione diretta di un nodo oppure no.

Nella versione estesa l'Identificatore è lungo 29 bit ed è spezzato in due campi di 11 (Base ID) e 18 bit (Extended ID) divisi da due bit chiamati rispettivamente SRR e IDE. Il tutto termina sempre con il bit RTR; i seguenti diagrammi serviranno a chiarirvi le idee (Fig. 4).

In entrambi i casi i 7 bit maggiormente significativi non possono essere tutti recessivi. Il bit RTR (Remote Transmission Request) è dominante nei data frame, mentre risulta pari a 1 nei remote frame. Nella versione estesa il bit SRR (Substitute Remote Request) è sempre recessivo e sostituisce il bit RTR del formato standard.

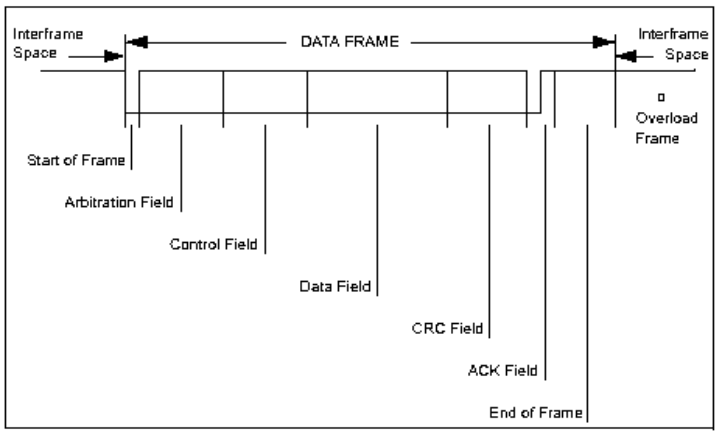
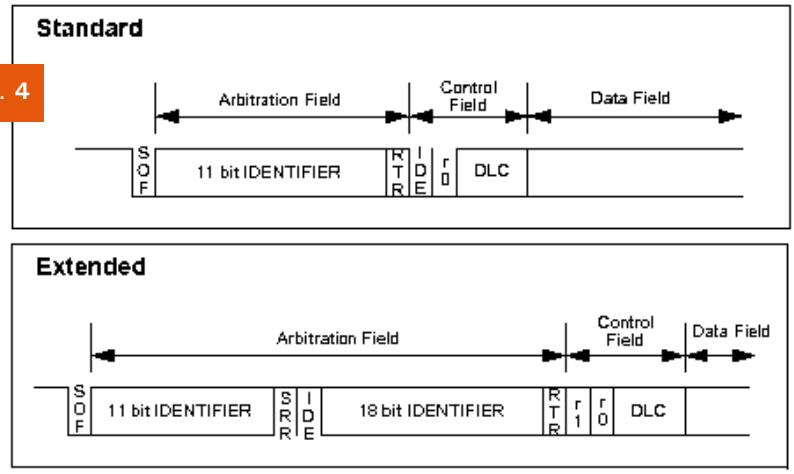


Fig. 3

Fig. 4



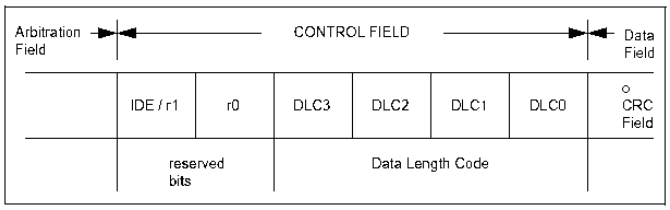
possibili combinazioni. Il diagramma di Fig. 5 rappresenta il frame nei due possibili formati.
Il Data Length Code è lungo 4 bit ed il suo valore binario corrisponde al numero di bit contenuti nel Data Field. La Tabella 2 raggruppa tutte le possibili combinazioni.
Il massimo numero di byte che un Data Frame può contenere è 8.
Ricordiamo che il valore 0 corrisponde a un livello dominante, mentre il valore 1 coincide con il livello recessivo.

Si noti che, nell'eventualità di una collisione, il messaggio standard prevale su quello esteso.
Il bit IDE (Identifier Extension) è inserito nel Control Field dei messaggi standard ed è sempre dominante; rimane parte del campo identificatore nel formato esteso ed è recessivo. In altre parole, esso stabilisce se ci troviamo di fronte ad un messaggio in formato standard (IDE dominante) o esteso (IDE recessivo).

CONTROL FIELD: questo campo è costituito da 6 bit ed ha un formato differente a seconda che i messaggi utilizzati siano del tipo standard o extended; nel primo caso include il DATA LENGTH CODE, il bit IDE (sempre dominante) e il bit riservato r0. Nel secondo caso, invece, l'IDE viene eliminato e i bit riservati sono due, cioè r0 e r1. Essi vengono trasmessi sempre come dominanti anche se le specifiche stabiliscono che il ricevente può accettarli sia dominanti che recessivi in tutte le

DATA FIELD: contiene i byte che devono essere trasferiti al nodo destinatario; questi ultimi possono essere al massimo otto e vengono trasmessi sempre a partire dal bit più significativo.
CRC FIELD: contiene una sequenza di controllo a 15 bit che termina con un apposito delimitatore; la sequenza deriva dall'applicazione di un

Fig. 5



algoritmo di calcolo basato sulla divisione tra il valore assunto da due funzioni polinomiali, che evitiamo di descrivere per non appesantire troppo il discorso. Ci limitiamo a dire che la proce- >

Tabella 2

Numero di byte	Data Length Code			
	DLC3	DLC2	DLC1	DLC0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

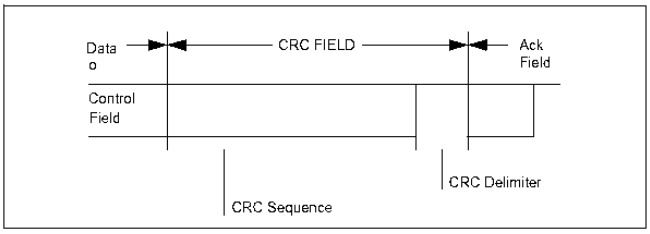


Fig. 6

dura è ottimizzata per l'utilizzo su frame aventi una lunghezza inferiore a 127 bit (Codice BCH). Il delimitatore del caso consiste in un singolo bit recessivo.

ACK FIELD: contiene un bit chiamato ACK SLOT ed uno che funge da delimitatore. Un nodo trasmittente invia entrambi i bit come recessivi; il dispositivo ricevente informa il mittente del corretto arrivo del messaggio, sovrascrivendo l'ACK SLOT con bit dominante. Il delimitatore, invece, è sempre recessivo. Si faccia attenzione al fatto che l'ACK SLOT risulta delimitato sempre da due bit recessivi: il delimitatore CRC e quello di ACK (Fig. 7).

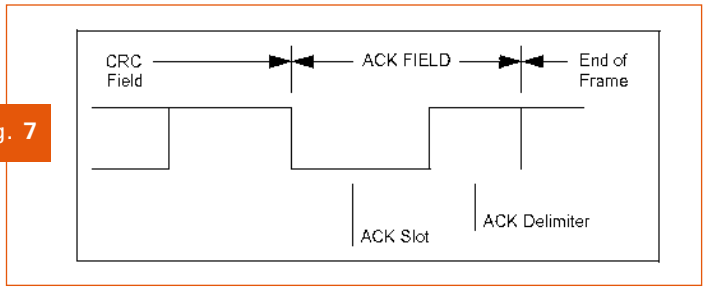


Fig. 7

END OF FRAME: ciascun Data Frame e Remote Frame è delimitato da una sequenza di 7 bit recessivi che ne stabilisce la fine. Una volta determinata la struttura dei Data Frame, risulta senz'altro più facile comprendere il contenuto dei Remote Frame, che sono dei messaggi utilizzati da un nodo per interrogare direttamente un altro dispositivo collegato alla stessa rete. Vediamoli dunque nel dettaglio.

REMOTE FRAME

Questa tipologia di messaggi può essere inviata nel formato sia standard che esteso; in entrambi i casi si compone di sei campi, che sono START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, CRC FIELD, ACK FIELD, END OF FRAME. La caratteristica che li differenzia dai Data Frame è che non contengono alcun Data Field, quindi i Remote Frame non veicolano alcun byte di informazione, ma costituiscono esclusivamente una richiesta verso l'altro nodo. Il Data Length Code contiene il numero di byte che il trasmittente si aspetta in risposta. Infine, il bit RTR è sempre recessivo. È chiaro, quindi, che il valore del bit RTR stabilisce in maniera univoca se il messaggio trasmesso è un Data Frame (RTR dominante) o un Remote Frame (RTR recessivo). Il diagramma corrispondente è visibile in Fig. 8.

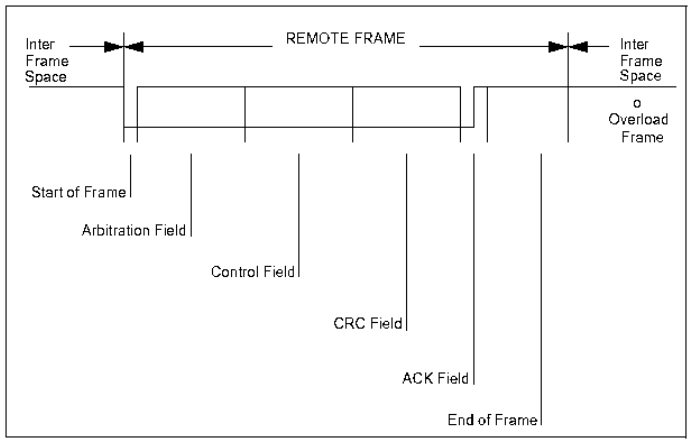


Fig. 8

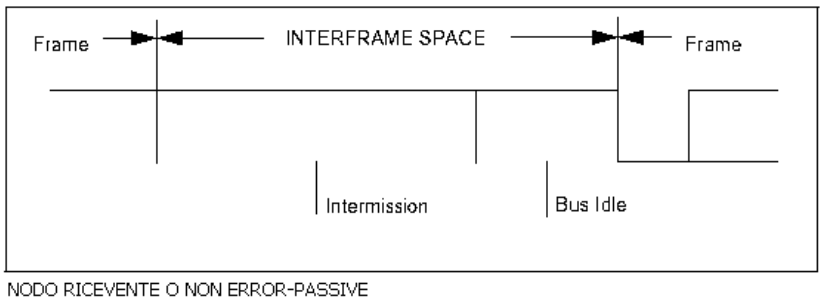
Qualcuno si sarà chiesto come fa il nodo trasmittente a capire qual è la risposta alla propria richiesta, visto che tutti gli altri dispositivi possono utilizzare il medesimo canale immettendo i propri dati, anche non inerenti alla richiesta stessa; ebbene, la soluzione del problema sta nel fatto che l'identificatore di un Remote Frame è identico a quello del Data Frame utilizzato in risposta. In questo modo il nodo trasmittente è sicuro di riconoscere, tra i vari messaggi che transitano lungo il canale, quello relativo alla propria richiesta.

INTERFRAME SPACING

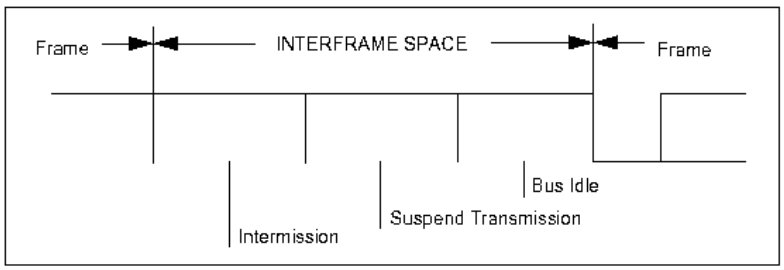
Ciascun messaggio Data Frame o Remote Frame è separato da quello successivo attraverso un'ap-

posita sequenza di bit chiamata Interframe Spacing. Gli Error Frame e gli Overload Frame vengono invece separati in modo diverso.
Lo spazio tra due frame è, in generale, costituito da due campi chiamati INTERMISSION e BUS IDLE; tuttavia esistono due casi in cui si aggiunge un terzo campo chiamato SUSPEND TRANSMISSION.
Per capirne la struttura dobbiamo precisare che quando viene rilevato un errore un nodo può trovarsi in tre possibili stati: Error-Active, Error-Passive, Bus-Off; in tal caso la trasmissione di un messaggio errato viene bloccata e il nodo attende di poter di nuovo occupare il canale per effettuarne nuovamente la trasmissione. Lo stato di Error-Passive significa che il nodo può inviare sul canale soltanto flag identificativi dell'errore costituiti da dei bit recessivi. Vi rimandiamo al paragrafo relativo al controllo degli errori per i dettagli del caso. La sequenza interframe cambia a seconda dello stato in cui si trova il nodo.
Ora, se il nodo è il ricevente del precedente messaggio oppure non è nello stato Error-Passive, si utilizzano solo i primi due bit; nel caso in cui il nodo è il trasmittente del precedente messaggio e si trova nello stato Error-Passive, si aggiunge il terzo bit. Esplichiamo il concetto dando uno sguardo ai relativi diagrammi (Fig. 9 e 10).
Nel concreto, la sequenza di Intermission consiste in tre bit recessivi. Durante tale sequenza i nodi non possono iniziare la trasmissione di Data Frame né di Remote Frame; vedremo che l'unica segnalazione possibile è quella di una condizione di overload. Dopo i tre bit di Intermission vengono trasmessi i bit di Bus Idle che segnalano ai nodi il fatto che il canale è libero per la prossima trasmissione.
Lo stato di Idle del bus può chiaramente avere durata differente e termina nel momento in cui il canale presenta un livello logico alto, livello che viene interpretato come un SOF (Start of Frame).
Un nodo, nel caso in cui dopo un messaggio si trovi nello stato Error-Passive, invia 8 bit recessivi che costituiscono la sequenza Suspend Transmission; soltanto una volta inviata tale sequenza, può riprendere a trasmettere o riconoscere lo stato di Idle del bus.
Nel caso in cui un altro nodo tenti l'invio di un messaggio durante la predetta Suspend Transmission, la stazione in Error-Passive diventerà la ricevente di tale messaggio.

Fig. 9



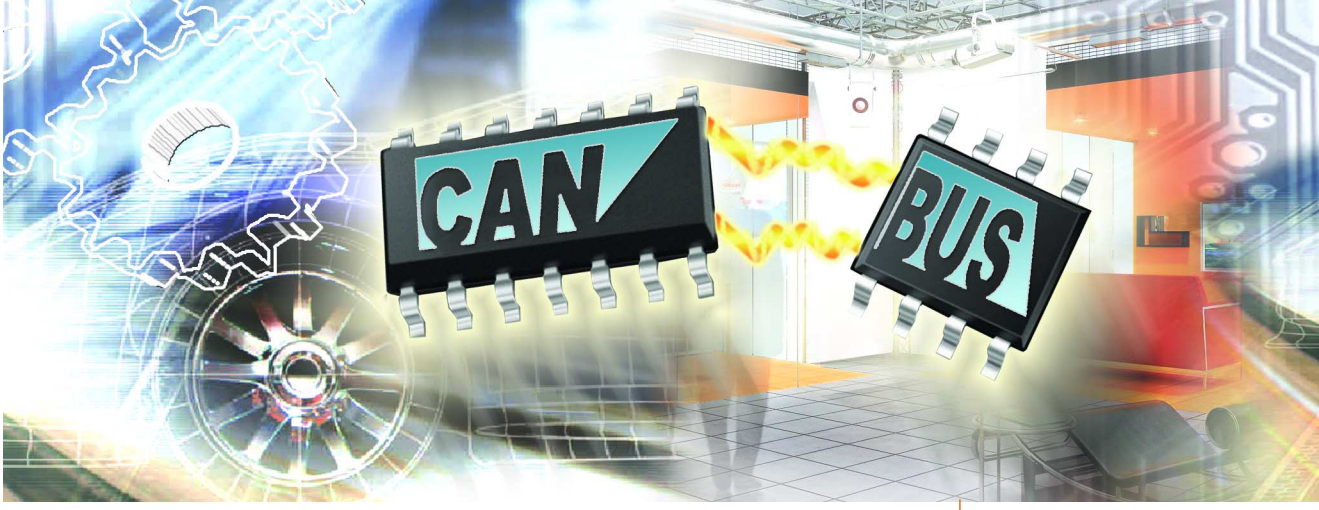
NODO RICEVENTE O NON ERROR-PASSIVE



NODO TRASMITTENTE E ERROR-PASSIVE

Fig. 10

Bene, in questa prima puntata, prettamente teorica, abbiamo analizzato il modello funzionale del CAN Bus, dettagliando le due più importanti tipologie di messaggi che i vari nodi si scambiano tra di loro. Nel prossimo numero della rivista, analizzeremo gli Overload Frame e soprattutto gli Error Frame, descrivendo il sistema di controllo degli errori, la cui importanza è fondamentale perché serve a comprendere l'architettura che sta dietro al sistema di comunicazione.
Lasceremo quindi i concetti teorici per iniziare la sperimentazione, introducendo lo schema della demo-board che utilizzeremo durante tutto il Corso per gli esempi applicativi.



a cura di Carlo Tauraso

Alla scoperta del CAN-BUS

Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa seconda puntata approfondiamo l'analisi dei messaggi di errore e presentiamo in anteprima lo schema della nostra demoboard.

Continuiamo il discorso relativo alle tipologie di messaggi che vengono veicolati in una rete basata su CAN (Controller Area Network). Ci eravamo lasciati dopo aver analizzato i due più importanti frame che un nodo CAN può immettere sul canale di comunicazione: i Data Frame e i Remote Frame. Vediamo nel dettaglio gli altri due tipi: gli Overload Frame e gli Error Frame.

Overload Frame

Questa tipologia di messaggi è costituita da due campi chiamati rispettivamente OVERLOAD FLAG e OVERLOAD DELIMITER. Quest'ultimo è una sequenza di 8 bit recessivi. Ricordiamo che un livello logico dominante corrisponde ad uno 0 mentre un livello logico recessivo corrisponde ad un 1. Il frame viene utilizzato per segnalare tre condizioni di sovraccarico:

1) Il nodo ricevente si trova in uno stato interno per cui non è in grado di ricevere ulteriori Data Frame o Remote Frame quindi richiede che venga ritardata la loro trasmissione;
2) In una sequenza di Intermission i primi due bit vengono rilevati come dominanti. Ricordiamo che la sequenza di Intermission in condizioni normali è costituita da 3 bit recessivi, pertanto rilevare dei bit dominanti significa che la regola è stata violata;
3) Il nodo rileva un livello dominante nell'ultimo bit di un campo delimitatore di un Overload Frame o di un Error Frame (l'ottavo bit).

Ci sono delle regole precise anche per determinare quando un nodo può immettere nel canale un Overload Frame. >

Per le prime due condizioni è possibile farlo nel momento in cui è prevista la trasmissione del primo bit di Intermission. Negli altri due casi il nodo può inserire il primo bit del messaggio di Overload subito dopo aver rilevato i bit dominanti che caratterizzano le due condizioni. Nel caso 1 la trasmissione del successivo Data Frame o Remote Frame può essere ritardata immettendo fino ad un massimo di due Overload Frame.
Nel diagramma di Fig. 1 vediamo com'è strutturato questo tipo di frame. Avrete sicuramente

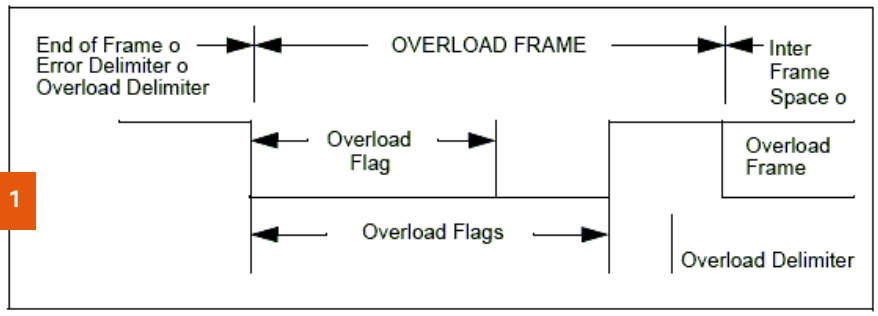


Fig. 1

notato la presenza di due voci relative all'Overload Flag, vediamo che cosa significa tutto ciò. L' Overload Flag è composto da 6 bit dominanti ed ha la stessa forma di un flag che vedremo utilizzato nelle segnalazioni di errore. Per capire bene perchè nel diagramma si mette in sovrapposizione una sequenza Overload Flags dobbiamo considerare che lo stream di 6 bit rompe, a causa della sua lunghezza, il formato fisso della sequenza di Intermission pertanto anche tutti gli altri nodi rilevano a loro volta una condizione di sovraccarico. Ciò fa sì che essi trasmettano a loro volta una sequenza di Overload Flag. Ecco, quindi spiegata la sovrapposizione dovuta alla risposta di tutti i nodi.
La stessa cosa avviene anche durante le segnalazioni di errore come vedremo nei prossimi paragrafi. Dopo la trasmissione di un Overload Flag il nodo monitorizza il canale finchè non rileva una transizione da un livello dominante ad uno recessivo. A questo punto tutti gli altri nodi avranno terminato la trasmissione del proprio flag pertanto inizia un ulteriore invio di 7 bit recessivi.

Error Frame

Anche l'Error Frame come l'Overload Frame si compone di due campi. Essi sono chiamati rispettivamente ERROR FLAG e ERROR DELIMITER. Esistono due forme di ERROR FLAG: una denominata ACTIVE ERROR FLAG composta da 6 bit dominanti, l'altra denominata PASSIVE ERROR FLAG composta da 6 bit recessivi. Vediamo il diagramma relativo in Fig. 2.

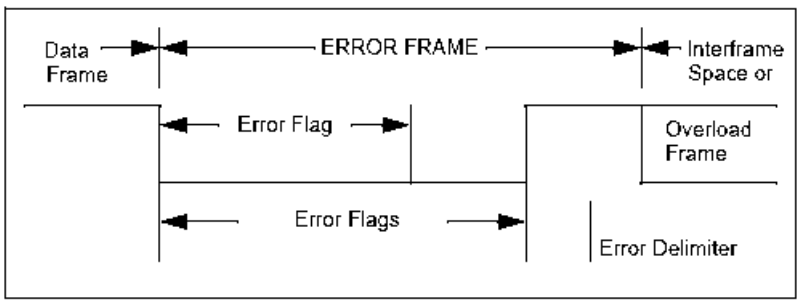


Fig. 2

Anche in questo caso si nota la presenza della sovrapposizione di più Error Flag. Il meccanismo è simile a quello degli Overload Frame e per spiegarlo facciamo un piccolo passo indietro precisando come avviene la codifica della sequenza di bit contenuta nei vari frame. I campi START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD e CRC sono codificati secondo una metodologia chiamata di "bit stuffing". Quando un nodo trasmittente verifica che nella sequenza da inviare ci sono 5 bit identici (tutti dominanti o tutti recessivi) esso automaticamente inserisce un sesto bit complementare. Il protocollo CAN utilizza, infatti, un metodo di trasmissione detto NRZ (Non-return-to Zero) secondo il quale il livello logico del bit sul bus viene mantenuto per tutta la sua durata. Siccome la comunicazione sul bus CAN è asincrona, i nodi riceventi utilizzano il bit complementare (in particolare la transizione tra bit recessivo e dominante) per risincroniz-

zarsi e scartandolo per quanto riguarda l'interpretazione dei dati. Negli altri campi dei Data Frame e Remote Frame, negli Overload Frame e negli Error Frame si usa invece un formato fisso. Ebbene, quando viene trasmesso l'Error Active Flag (che è lungo 6 bit) si viola la regola del bit stuffing e quindi tutti i nodi rilevano una condizione di errore ed inviano a loro volta un loro Error Flag chiamato Echo Error Flag. Ecco, quindi, che si verifica una sovrapposizione di bit dominanti come visualizzato nel diagramma. La lunghezza di questa sequenza va da un minimo di 6 bit ad un massimo di 20 bit. Nel caso venga inviato un Error Passive Flag la situazione è un pò diversa visto che i bit sono recessivi. Se il nodo che invia il flag è l'unico trasmittente, allora la sequenza viola la regola di "bit stuffing" pertanto tutte le altre stazioni risponderanno con i propri flag. Nel caso in cui il nodo non sia l'unico trasmittente o sia in ricezione, l'invio della sequenza sul bus non ha alcuna conseguenza a causa della natura recessiva dei bit. In particolare nel caso in cui il nodo rilevi la presenza anche di un solo bit dominante deve bloccare la ricezione, attendere il termine della sequenza di Intermission, rilevare lo stato di idle del bus e provare di nuovo a trasmettere. Fino a qui tutto bene, ma che cosa determina l'invio di un active-flag o di un passive-flag? La risposta è tutta nei possibili errori rilevati e nei relativi stati di errore assumibili da un nodo.

Errori rilevati

Nel protocollo CAN esistono 5 tipologie di errori rilevabili da ciascun nodo. Nel momento in cui un nodo si accorge dell'errore lo rende pubblico a tutti gli altri attraverso la trasmissione di un Error Frame. Vediamo i dettagli caso per caso:

BIT ERROR: nel momento in cui un'unità invia un bit sul bus effettua anche un monitoraggio di quest'ultimo. Se sta trasmettendo un bit recessivo e rileva invece un bit dominante o viceversa si accorge che c'è un problema. Esistono delle eccezioni in cui non si rileva l'errore come: l'invio di un bit recessivo durante la sequenza dell' ARBITRATION FIELD o l'ACK SLOT, o l'invio di un PASSIVE ERROR FLAG e il rilevamento di un bit dominante. Una volta accortosi di un BIT ERROR, il nodo genera un Error Frame e il messaggio originale viene ritrasmesso dopo una sequenza di Intermission.

STUFF ERROR: questo errore viene rilevato nel momento in cui sul bus si presenta il sesto bit consecutivo con il medesimo livello logico (6 bit tutti recessivi o tutti dominanti), durante la trasmissione di un campo sottoposto alla codifica di "bit stuffing" (START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD e CRC).

CRC ERROR: ogni volta che un nodo riceve un messaggio, ricalcola il valore del CRC confrontandolo con quello contenuto nel frame ricevuto. Nel caso i due valori non coincidano provvede a segnalare l'incongruenza attraverso un Error Frame. Anche se soltanto uno dei nodi non riceve il messaggio correttamente, esso viene ritrasmesso dopo un'opportuna sequenza di Intermission.

FORM ERROR: viene generato ogni volta che un campo a formato fisso contiene uno o più bit non leciti. Ad esempio nel caso in cui un nodo rilevi uno o più bit dominanti in uno dei seguenti componenti: END OF FRAME, INTERFRAME SPACE, ACKNOWLEDGE DELIMITER, CRC DELIMITER. Il messaggio originale viene chiaramente ritrasmesso dopo il periodo di Intermission.

ACKNOWLEDGMENT ERROR: viene generato ogni volta che un nodo trasmittente non rileva un bit dominante durante l'ACK SLOT. Ricordiamo che l'ACK SLOT viene inviato dal trasmittente come recessivo. Nel caso non si rilevi il livello dominante significa che nessun nodo ha ricevuto correttamente il messaggio. Viene emesso un Error Frame ed il messaggio viene ritrasmesso dopo la solita sequenza di Intermission.

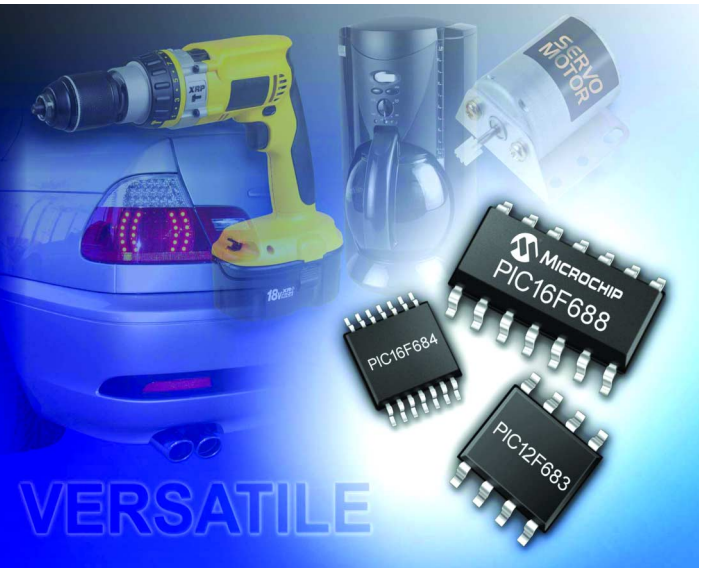
Stati d'errore

Ogni nodo nel momento in cui viene emesso l'Error Frame può trovarsi in tre possibili stati: ERROR ACTIVE, ERROR PASSIVE, BUS OFF. Questo permette di realizzare il cosiddetto "Fault Confinement" cioè isolare le fonti di errori di comunicazione garantendo la continuazione del funzionamento del bus e preservando la larghezza di banda. Il sistema si basa sulla presenza di due contatori in ciascuna unità di rete. Tali contatori sono chiamati: TRANSMIT ERROR COUNT (TEC) e RECEIVE ERROR COUNT (REC). Il valore di questi due registri sono modificati attraverso le seguenti regole:

1) Quando un ricevente rileva un errore, il REC viene incrementato di 1, tranne nel caso di un BIT ERROR durante la trasmissione di un ACTIVE ERROR FLAG o di un OVER- >

LOAD FLAG.

2) Quando un ricevente rileva un livello dominante come primo bit dopo l'invio di un ERROR FLAG, il REC viene incrementato di 8.
3) Quando un trasmittente invia un ERROR FLAG il TEC viene incrementato di 8 tranne nel caso in cui esso sia relativo ad un stuff error dovuto alla presenza di un bit di stuffing precedente al RTR che doveva essere recessivo ma viene monitorato come dominante, e nel caso in cui il trasmittente invii un PASSIVE ERROR FLAG perchè viene rilevato un ACKNOWLEDGMENT ERROR e durante la sequenza recessiva del flag non viene ricevuto alcune livello dominante.
4) Quando un trasmittente rileva un BIT ERROR mentre invia un ACTIVE ERROR FLAG o un OVERLOAD FLAG il TEC è incrementato di 8.
5) Quando un ricevente rileva un BIT ERROR mentre riceve un ACTIVE ERROR FLAG o un OVERLOAD FLAG il REC è incrementato di 8.



6) Ogni nodo tollera fino a 7 bit consecutivi dominanti dopo l'invio di un ACTIVE ERROR FLAG, PASSIVE ERROR FLAG o OVERLOAD FLAG. Dopo aver rilevato il quattordicesimo bit dominante (nel caso di ACTIVE ERROR FLAG o OVERLOAD FLAG) o l'ottavo successivo ad un PASSIVE ERROR FLAG (e quindi anche per tutti gli ulteriori 8 bit dominanti) ogni trasmittente ed ogni ricevente incrementa rispettivamente il proprio TEC e il proprio REC di 8.
7) Dopo ogni messaggio inviato correttamente il TEC è decrementato di 1 fino ad arrivare a 0.
8) Dopo ogni messaggio ricevuto correttamente il REC è decrementato di 1 se è compreso tra 1 e 127 mentre per valori superiori gli viene assegnato un numero compreso tra 119 e 127.

Ora, se consideriamo i valori assumibili dal TEC e dal REC possiamo dire che **una stazione si trova nello stato ERROR ACTIVE se entrambe hanno un valore inferiore a 128, ERROR PASSIVE se uno dei due supera il valore di 127, BUS OFF quando il TEC è superiore a 255.** Analizziamo i tre casi separatamente:

ERROR ACTIVE: un nodo che si trova in questo stato può partecipare attivamente alla comunicazione sul bus sia in ricezione che attraverso l'invio di un ACTIVE ERROR FLAG costituito da 6 bit dominanti. La rottura della regola di "bit stuffing" comporta l'emissione degli ECHO ACTIVE ERROR FLAG come abbiamo già visto. Viene considerata la modalità normale di funzionamento di un nodo CAN.

ERROR PASSIVE: in questo caso il nodo può trasmettere soltanto dei PASSIVE ERROR FLAG cioè 6 bit recessivi. E' chiaro che nel caso sia l'unico nodo trasmittente la sequenza viola la regola di “bit stuffing” e tutti gli altri nodi rispondono con degli Error Flag (ACTIVE o PASSIVE a seconda del loro stato). In tutti gli altri casi abbiamo già visto che i bit trasmessi non hanno alcun effetto visto che sono recessivi. Anzi, rilevandone uno dominante il nodo deve interrompere la trasmissione ed attendere il termine dell'Intermission per provare di nuovo l'invio.

BUS OFF: questo stato è relativo solo al caso in cui il TEC è superiore a 255 pertanto la ricezione di errori dall'esterno non può portare un nodo in Bus Off. In tale condizione il nodo non può nè ricevere nè trasmettere messaggi o Error Frame di alcun tipo. Questo permette di realizzare l'isolamento dei nodi che sono fonte di errori. Il protocollo CAN prevede una procedura di recovery che permette ad un nodo in Bus Off di diventare Error Active e di ricominciare a trasmettere sempreché la condizione di malfunzionamento venga rimossa. In pratica tale procedura prevede che vengano posti a 0 sia il TEC che il REC del nodo con problemi di funzionamento dopo aver rilevato per 128 volte una sequenza di 11 bit recessivi sul bus.

In generale si consideri che un contatore degli errori con un valore attorno ai 96 sta ad indicare

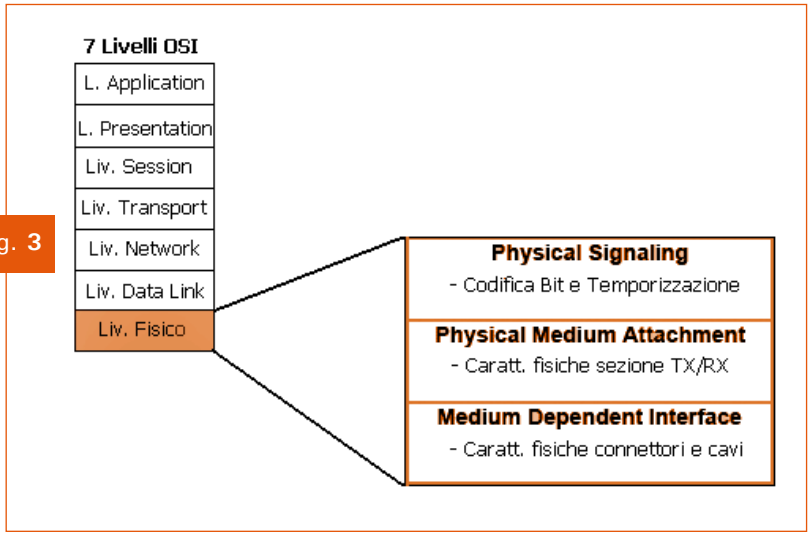


Fig. 3

potuto creare delle difficoltà nel far dialogare dispositivi nati con interfacce fisiche differenti. Vogliamo sottolineare questo fatto perchè anche nella nostra demo-board (come per tutti i dispositivi CAN) utilizzeremo un chip prodotto da Microchip che fungerà da interfaccia tra la logica di un microcontrollore e il bus. Tenete, quindi, a mente la presenza di due oggetti fisici: un controller CAN ed un transceiver CAN. Quest'ultimo implementa proprio tutte le specifiche inserite nello standard ISO-11898. Nel numero precedente abbiamo visto come si presentava il modello funzionale del Bus CAN. Vediamo, ora come si integra nel livello più basso. Dal disegno di Fig.3 si vede come lo strato fisico venga suddiviso in tre sottolivelli con delle incombenze specifiche:

PS (Physical Signaling): riguarda la codifica dei bit e la loro sincronizzazione, ne troviamo riferimenti già nelle specifiche originali CAN.

PMA (Physical Medium Attachment): si occupa delle caratteristiche fisiche della sezione trasmittente e ricevente di un nodo. Non viene precisato dalle specifiche originali CAN ma viene introdotto dallo standard ISO-11898.

MDI (Medium Dependent Interface): si occupa delle caratteristiche fisiche dei connettori e dei cavi di collegamento. Non viene precisato dalle specifiche originali CAN ma viene intro- >

un bus fortemente disturbato pertanto può essere vantaggioso monitorare il verificarsi di tale situazione anche se le specifiche lasciano ampio spazio alle personalizzazioni.

Lo strato fisico

Le specifiche ufficiali relative al CAN Bus non descrivono in dettaglio l'implementazione del livello fisico tantomeno quello degli strati più elevati lasciando ai progettisti ampia libertà di scelta. In particolare ci si riferisce al livello fisico solo nella sua parte più elevata relativa alla codifica delle sequenze di bit ed alla loro sincronizzazione. La ISO (International Standards Organization) ha provveduto a raccogliere in uno standard chiamato ISO-11898 una serie di specifiche che definiscono completamente lo strato fisico per il CAN Bus. La cosa era nata essenzialmente per dettare alcune regole per lo scambio di informazioni ad alta velocità su reti CAN. Successivamente è divenuto un documento di riferimento per tutti i progettisti di dispositivi CAN al fine di garantire la compatibilità tra di essi. Se da un lato, infatti, la logica di funzionamento del bus è identica per tutti, la libertà nell'implementazione dello strato fisico avrebbe

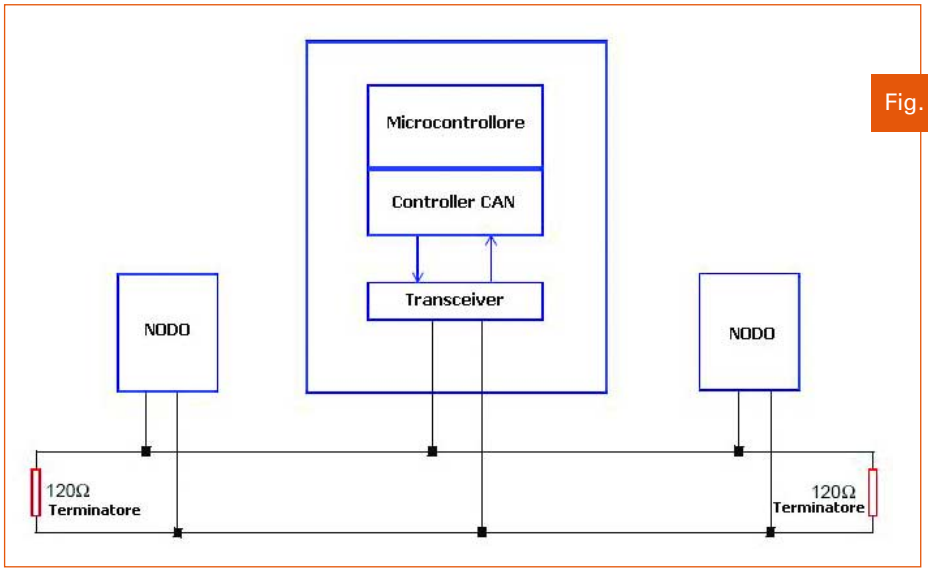


Fig. 4

dotto dallo standard ISO-11898. Tutto ciò è riassunto, come detto, in Fig. 3. Sulla base di questo modello un tipico nodo CAN è costituito da un controller che implementa la logica dei livelli elevati e da un transceiver che si occupa dell'interfacciamento con il bus e che quindi contiene l'implementazione del livello più propriamente fisico. La situazione è rappresentata dall'immagine di Fig. 4. In pratica il microcontrollore che utilizzeremo avrà un modulo CAN che fungerà da controller ed utilizzerà un apposito chip esterno (MCP2551) come interfaccia fisica verso il bus.

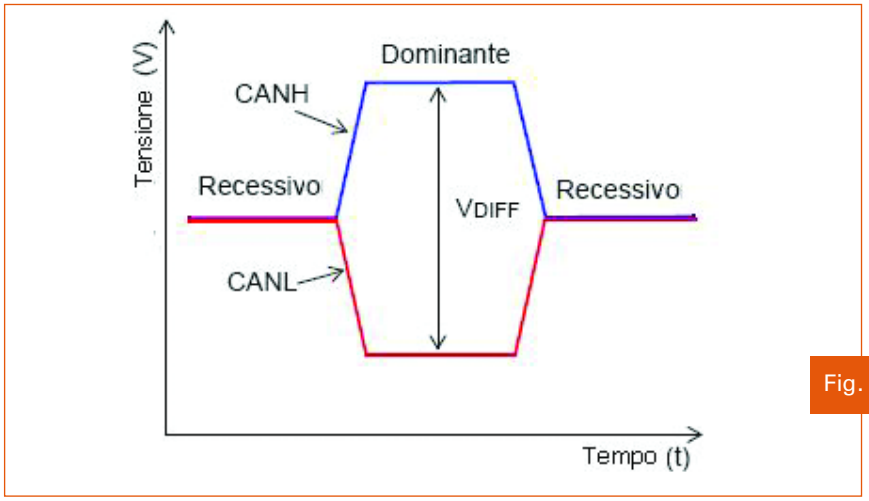


Fig. 5

Livelli logici

Abbiamo visto che in un bus CAN si rilevano due livelli differenti chiamati: dominante (uguale a 0) e recessivo (uguale a 1). Nei documenti ISO si specifica che essi vengono rappresentati attraverso la differenza di tensione tra le due linee del bus (CANH e CANL). Nello stato recessivo la differenza di tensione tra le due linee è inferiore ad una soglia precisa pari a 0,5V, nello stadio ricevente, e 1,5V nello stadio trasmittente. Analogamente nello stato dominante la differenza è superiore a tale soglia. Proprio in riferimento a questa modalità di rappresentazione dei livelli logici il bus CAN viene anche detto bus a funzionamento differenziale. Il diagramma di Fig. 5 sintetizza la situazione.

Cavi e Connettori

Non esiste un dettaglio specifico sulle caratteristiche meccaniche dei connettori o dei cavi utilizzabili per collegare un nodo al bus. Si precisa però che essi devono essere compatibili con le specifiche elettriche, alcune delle quali riassumiamo in Tabella 1.

Si nota dalla tabella che le specifiche sono tali da descrivere dei transceiver piuttosto robusti in grado di sopportare delle tensioni che raggiungono i 32V e dei transienti tra -150V e +100V. Lo standard, in pratica, prevede l'utilizzo in ambienti piuttosto severi e quindi una altrettanto rigida protezione delle comunicazioni.

Secondo la documentazione ogni transceiver deve essere in grado di "lavorare" ad 1 Mbps su un bus lungo al massimo 40 metri. Con lunghezze maggiori è necessario diminuire il transferrate. L'aumento della lunghezza del bus comporta un aumento del ritardo di propagazione del segnale ed oltre certi limiti ciò determina un errato arbitraggio del canale.

Per eliminare la riflessione del segnale vengono previsti dei terminatori a ciascuna estremità del bus con una resistenza di 120 ohm. In particolare, secondo le specifiche, si possono adottare tre tipologie di terminatori:

Tabella 1

Parametro	min	max
Tensione su CANH e CANL	-3V	+32V
Transienti su CANH e CANL	-150V	+100V
Tensione Livello Recessivo OUT	+2V	+3V
Differenza di Tensione Livello Recessivo OUT	-500mV	+50mV
Tensione Livello Dominante OUT (CANH)	+2,75V	+4,50V
Tensione Livello Dominante OUT (CANL)	+0,50V	+2,25V
Differenza di Tensione Livello Dominante OUT	+1,5V	+3,0V
Resistenza Interna Differenziale cioè la resistenza misurata tra CANL e CANH durante uno stato recessivo e con il nodo sconnesso dal bus	10Kohm	100Kohm

1) *Standard*: sono composti da una singola resistenza da 120 ohm (Fig. 6).
2) *Split*: la singola resistenza viene suddivisa in due resistenze da 60 ohm ciascuna. Esse vengono collegate attraverso un condensatore connesso a massa (Fig. 7).
3) *Biased*: è simile al precedente soltanto che tra le due resistenze viene inserito un partitore di tensione che assicura una tensione dimezzata (Vdd/2), vedi Fig. 8.

Fig. 6

Fig. 7

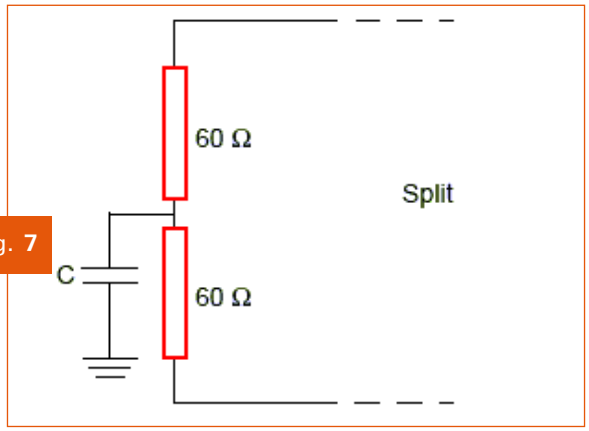
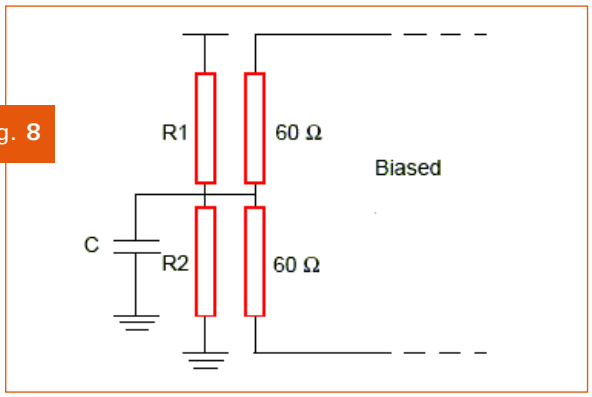


Fig. 8



Dopo questo breve percorso che ci ha portato a scoprire i dettagli di funzionamento, non ci resta che iniziare un po' di sperimentazione per vedere come tutte queste cose vengono realizzate.

La demo-board

Il circuito che utilizzeremo per realizzare alcuni esperimenti di sviluppo sul Bus CAN utilizza un microcontrollore Microchip PIC18F458 ed un transceiver MCP2551 per l'interfacciamento fisico. Fondamentalmente costruiremo un singolo nodo il cui schema potrà essere duplicato per realizzare delle reti con funzioni sempre più complesse.
Il sistema è dotato di diversi altri componenti di contorno:

1) **Slot per SD Card**: utilizzato per la memorizzazione di dati derivanti da campionamenti o elaborazioni, viene sfruttato in modalità SPI. E' stato necessario realizzare la conversione dei livelli di tensione utilizzati nella comunicazione con le SD che notoriamente funzionano in un range che va da 2,7V a 3,6V. Abbiamo utilizzato quindi un regolatore di tensione a 3,3V (LM1086) per l'alimentazione, mentre per i diversi livelli 0-5V e 0-3V abbiamo utilizzato una serie di diodi schotty con resistenze di pull-up sulle linee che vanno dal micro alla SD ed un integrato 74HCT125 per l'uscita da SD a micro. Nel primo caso la linea viene mantenuta ad una tensione di circa 3.3V, non appena sul pin del micro viene presentato un valore logico alto il diodo è interdetto e la tensione sul pin della card sarà la tensione di pull-up. Quando, invece, viene presentato un valore logico basso, il diodo si porta in conduzione collegando a massa anche il pin della card. Per quanto riguarda la connessione di direzione inversa, cioè tra card e PIC, la cosa è differente. Gli integrati basati su logica ACT/HCT accettano livelli TTL in input e presentano in uscita livelli CMOS. In particolare, quando sono alimentati a 5 volt "vedono" un segnale a 3 volt come un normale TTL a 5 volt e forniscono in uscita un segnale a 5 volt che va benissimo per comandare una linea di input del PIC senza necessità di pull-up. Ecco, quindi, che il 74HCT125 diventa un perfetto traduttore CMOS -> TTL.
2) **EEPROM 24LC64**: per la memorizzazione dei dati di configurazione o come buffer temporaneo.
3) **Serie di 4 Led Luminosi**: per segnalare i vari stati di funzionamento del dispositivo.
4) **Porta RS232**: per permettere una connessione diretta PC-dispositivo, sia per sperimentare l'invio di comandi che per ricevere messaggistica di controllo sul funzionamento. I diversi livelli 0-5V e -12v/+12V vengono tradotti dal sempre utile MAX232 nella sua con- >

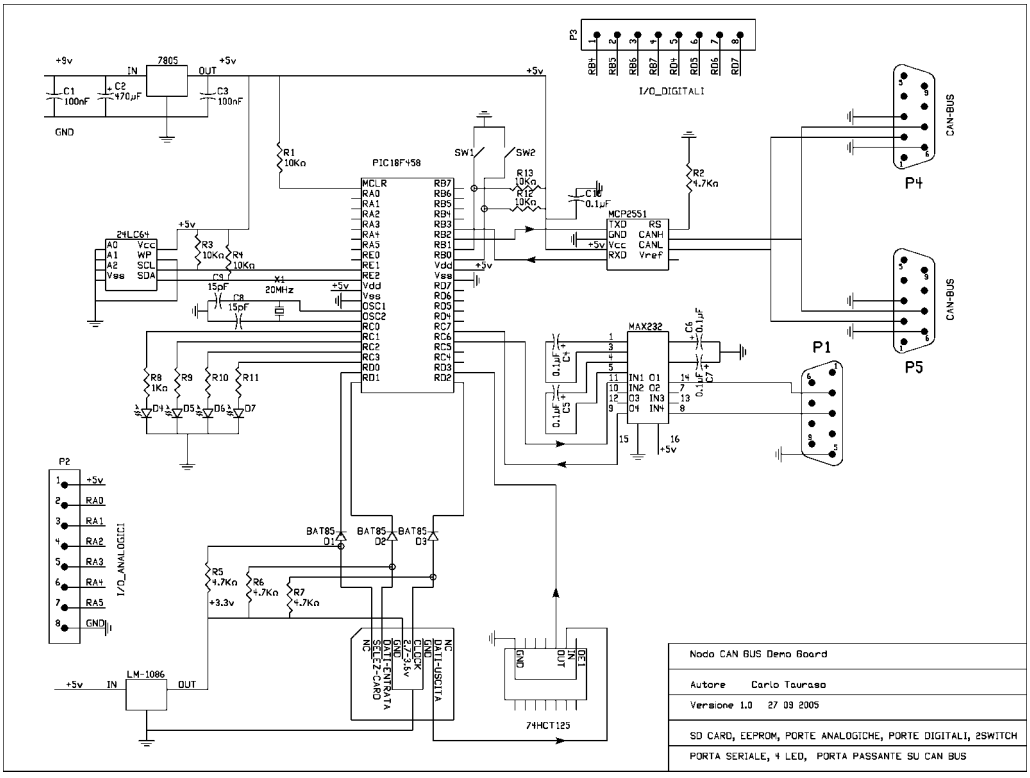
figurazione di base.

5) **Due pulsanti**: per simulare comandi logici attivati direttamente sul dispositivo CAN.
6) **Strip di I/O digitali**: sono linee del PIC che una volta configurate possono essere utilizzate in ingresso o in uscita per svolgere vari tipi di funzione a seconda dell'esperimento da svolgere.
7) **Strip di ingressi analogici**: sono linee del PIC che fanno capo ad un modulo A/D. Possono venir usate per il campionamento di segnali analogici e sono dotate di poli d'alimentazione a 5V.

Il transceiver MCP2551 viene utilizzato come circuito di traduzione tra i segnali TTL presenti sui pin del microcontrollore e quelli che fanno funzionare il Bus CAN. In particolare questo chip ha diverse modalità di funzionamento, quella scelta da noi è denominata "SLOPE-CONTROL". Infatti, connettendo il pin RS attraverso una resistenza a massa si fa sì che si riducano i tempi di "rise" e "fall" dei segnali sui pin CANH e CANL, riducendo la possibilità di generare interferenze elettromagnetiche. Il chip è pienamente compatibile con le specifiche dettate dallo standard ISO-11898, anzi in certi casi le supera ampiamente come nel caso dei transienti che riesce a sopportare arrivando a circa 250V. La doppia porta sulla scheda ci permette di connettere altri nodi sulla rete creando una sorta di catena agli estremi della quale potremo inserire dei terminatori. Naturalmente è possibile duplicare il circuito anche senza riproporre tutti i componenti di contorno ma soltanto quelli che serviranno per le funzionalità del nodo che stiamo aggiungendo. Il transceiver è in grado di operare ad un transfer rate di 1 Mbps e supporta fino a 112 nodi connessi sul medesimo bus (con resistenza interna differenziale minima di 20 kohm e terminatore con resistenza nominale di 120 ohm). E' chiaro che tali caratteristiche sono più che sufficienti per i nostri scopi. A pie' di pagina (Fig.9) pubblichiamo in anteprima lo schema della demoboard.

Una volta realizzato tale circuito, vi consigliamo di dare un'occhiata anche alle puntate di approfondimento sullo sviluppo C18 e la relativa integrazione nell' IDE MPLAB. Nei nostri esperimenti utilizzeremo, infatti, questo ambiente di sviluppo come riferimento fondamentale. Non ci resta che darvi appuntamento alla prossima puntata nella quale cominceremo a far "lavorare" il nostro primo nodo CAN.

Fig. 9



a cura di Carlo Tauraso

Alla scoperta del CAN-BUS

Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa terza puntata entriamo nel vivo dello sviluppo del modulo CAN e analizziamo l'hardware che useremo nei nostri esperimenti.

3

Nella precedente puntata del nostro corso vi abbiamo mostrato in anteprima lo schema elettrico della demoboard per le applicazioni CAN e annunciato l'imminente pubblicazione delle nozioni inerenti alla progettazione e alla messa in esercizio di nodi CAN. Eccoci dunque, puntuali, ad affrontare l'argomento annunciato il mese scorso.

Librerie CAN C18

Per sviluppare correttamente il firmware per i nostri nodi CAN dobbiamo innanzitutto considerare il fatto che Microchip rende disponibile un'interessante libreria C18, peraltro disponibile e scaricabile liberamente dal sito della rivista. La presenza di una serie di funzioni che nascondono opportunamente i dettagli implementativi del protocollo CAN facilita notevolmente il lavoro del programmatore, che può concentrarsi maggiormente sulla logica della propria applicazione. Il pacchetto che prendiamo in considerazione è contenuto in un archivio compresso chiamato *ECAN.zip*. La libreria in questione permette la gestione di un nodo CAN nelle modalità sia standard che estesa, pertanto sarà possibile manipolare anche i messaggi con un *arbitration field* avente un identificatore lungo. Essa rappresenta >

Tabella 1

Nome File	Descrizione
ECAN.c	contiene l'implementazione in C18 delle funzioni CAN
ECAN.h	contiene le dichiarazioni delle funzioni CAN
ECAN.def	contiene una serie di definizioni che stabiliscono la fase di inizializzazione del modulo CAN e la sua modalità di funzionamento.

un ottimo punto di partenza per tutti coloro che vogliono avvicinarsi allo sviluppo firmware sul CAN-Bus.
Per realizzare il nostro primo progetto CAN vediamo come è strutturata questa libreria. Essa si compone di tre file (Tabella1).
In merito al file *ECAN.def* c'è da dire che può essere personalizzato a seconda delle proprie necessità. Ad esempio, tale libreria permette di sfruttare tre possibili modalità di funzionamento:

MODO0: compatibile con la versione standard del protocollo CAN;

tive alle operazioni di interfacciamento SPI. Qui possiamo creare il file *ECAN.def* e modificarne le definizioni utilizzando un'interfaccia grafica molto intuitiva.
Ricordiamo che per avere a disposizione tutti i listati necessari, dopo aver installato il pacchetto bisogna integrarlo con i moduli in versione 1.03, che si possono liberamente scaricare dal sito ufficiale Microchip (*www.microchip.com*). Avviamo l'applicativo e selezioniamo il modulo *ECAN (Polled)*; trascinando quest'ultimo nell'elenco *Selected Model* vedremo apparire nel riquadro in

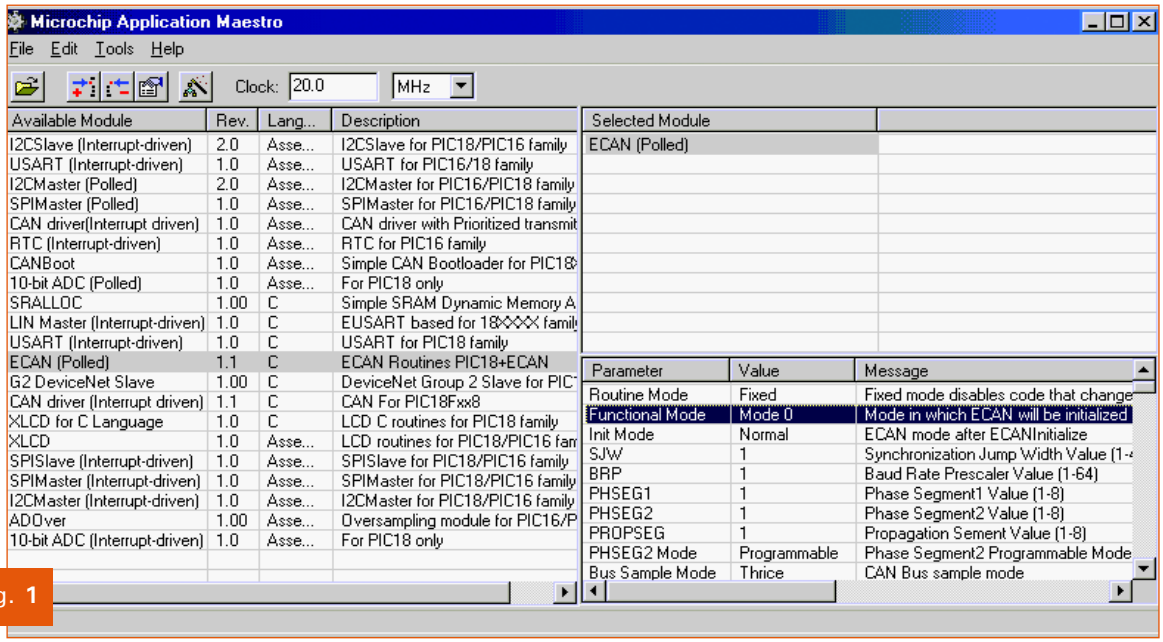


Fig. 1

MODO1: implementa le feature supplettive della versione estesa del protocollo CAN;
MODO2: compatibile con la versione estesa del protocollo CAN ed integrata dall'implementazione di un buffer FIFO in ricezione.

basso a destra un elenco di parametri che sono proprio quelli definiti nel file *ECAN.def*. Ogni parametro è affiancato dal valore predefinito e da un messaggio che ne descrive il significato così come si vede in Fig. 1.
Se facciamo doppio clic su uno dei parametri abbiamo la possibilità di modificarne il valore. Supponiamo di voler cambiare la modalità di funzionamento del modulo CAN attraverso il campo chiamato *Functional Mode*.

Nel file .def troviamo la seguente definizione: ***#define*** *ECAN_FUNC_MODE_VAL ECAN_MODE_0*. Associando uno dei tre possibili valori (*ECAN_MODE_0*, *ECAN_MODE_1*, *ECAN_MODE_2*) essa permette proprio di stabilire quale delle tre modalità utilizzare. Per una configurazione dettagliata è possibile utilizzare il modulo incluso nel pacchetto *Application Maestro* di Microchip.
Se ricordate avevamo incontrato questo sistema di sviluppo parlando delle memorie Flash; in quella occasione avevamo presentato la modalità di generazione delle funzioni rela-

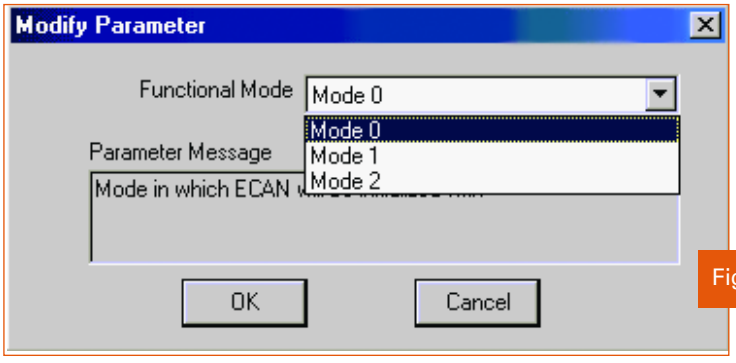


Fig. 2

Successivamente al doppio clic, apparirà la finestra di dialogo visibile in Fig. 2, nel cui menu a tendina selezioniamo, ad esempio, la modalità *Mode1*.
Generiamo il codice attraverso il pulsante comando evidenziato in rosso in Fig. 3 (si può impartire anche con la scorciatoia da tastiera *CTRL+G*).

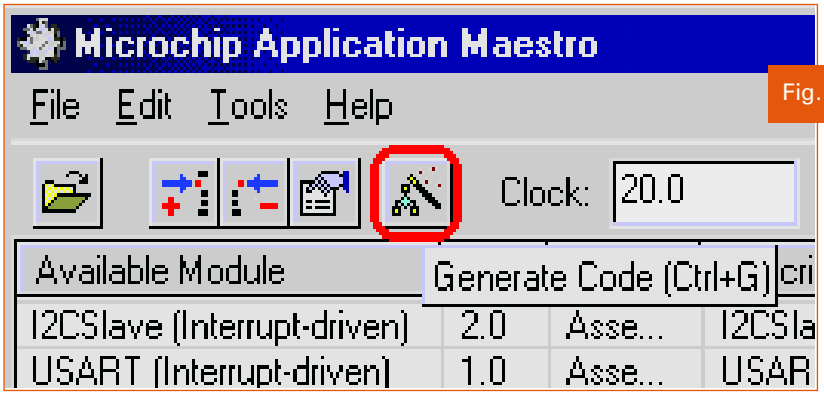


Fig. 3

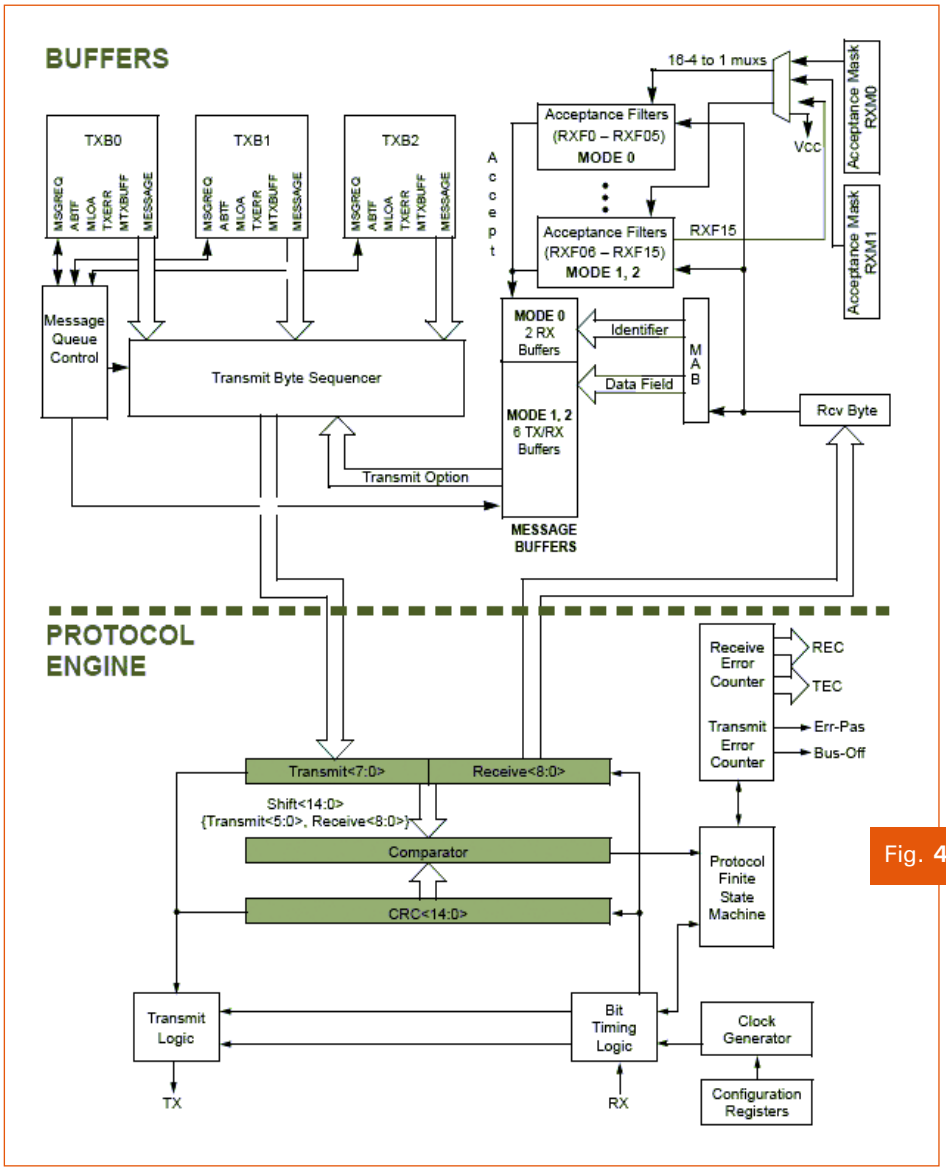


Fig. 4

Nella directory selezionata troveremo un file chiamato *ECANPoll.def* che conterrà la definizione: ***#define*** *ECAN_FUNC_MODE_VAL ECAN_MODE_1*.
Questo è senz'altro il miglior metodo per effettuare una personalizzazione dettagliata del file in questione, anche perché mette al sicuro il programmatore da eventuali errori di sintassi, soprattutto per quanto riguarda i possibili valori assegnabili ai singoli parametri. Naturalmente è possibile anche effettuare un editing diretto del file all'interno dell'ambiente MPLAB-IDE. In entrambi i casi bisogna conoscere il significato di ciascun parametro. Per farlo dobbiamo prima vedere in che modo un PIC implementa fisicamente un modulo CAN.

Hardware del Modulo CAN

Prendiamo come riferimento un PIC18F458. Il modulo CAN è composto da una serie di buffer, registri di controllo e da un motore che gestisce la ricezione e la trasmissione di messaggi. Osserviamo lo schema (incluso nella documentazione Microchip) di Fig. 4. Per quanto riguarda il motore si può identificare una serie di blocchi logici specializzati. Ciascuno di >

essi esegue una ben determinata funzione, secondo quanto esposto qui di seguito:

1) **Transmit Error Counter (TXERRCNT)**: stabilisce la frequenza con cui avvengono gli errori di trasmissione; nel caso si verifichi un overflow il nodo viene posto in "bus-off", mentre se sul bus vengono trasmessi 128 pacchetti composti da 11 bit recessivi, il valore del contatore viene azzerato;

2) **Receive Error Counter (RXERRCNT)**: stabilisce la frequenza con cui avvengono gli errori di ricezione; nel momento in cui il contatore assume un valore maggiore di 127 il nodo viene posto in "error-passive" (Err-Pass);

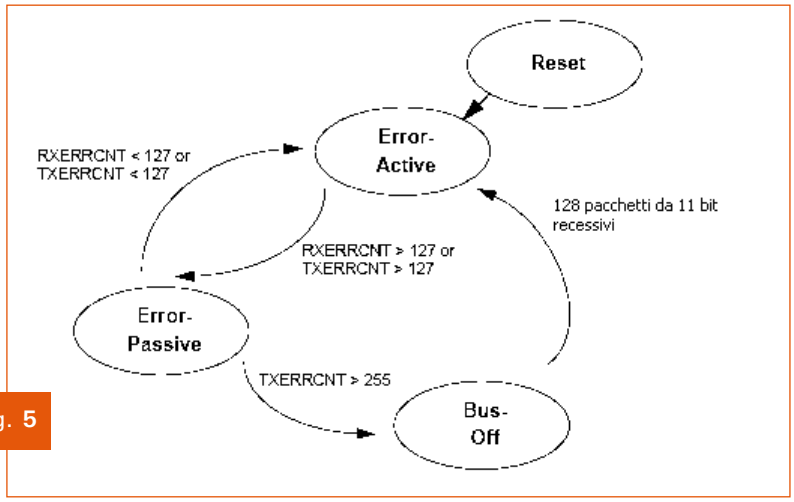


Fig. 5

3) **CRC Register**: Effettua il controllo del CRC per i messaggi in ricezione e lo ricalcola per quelli in trasmissione;

4) **Comparator**: permette di effettuare operazioni logiche di confronto tra i vari registri, tipicamente per il controllo del CRC;

5) **Bit Timing Generator/Logic**: si occupa di controllare la sincronizzazione delle sequenze di bit in ingresso ed uscita; ricordiamo, infatti, che ciascun nodo utilizza un bit-rate costante;

6) **Transmit/Receive Shift**: si tratta di due registri di shift che permettono di trasferire le sequenze di bit in ingresso ed uscita dal motore alla struttura di buffer relativa;

7) **Protocol FSM (Finite State Machine)**: implementa i vari aspetti del protocollo CAN utilizzando un modello di macchina a stati finiti; in pratica la comunicazione avviene attraverso la transizione tra diversi stati legati ad una serie di eventi; ad esempio, nel caso in cui venga rilevata la presenza di una collisione durante uno stato di trasmissione si ha l'immediato blocco della trasmissione e la transizione in uno stato di attesa; il relativo sotto-sistema è essenziale per la gestione dell'arbitraggio del canale;

8) **Transmit Logic**: effettua tutte le operazioni necessarie all'invio corretto di messaggi sul bus come ad esempio l'ascolto del canale per evitare collisioni.

La logica di gestione degli errori comporta una serie di passaggi tra diversi stati, come si vede dal diagramma di Fig. 5.

Molto importante è la struttura di buffer realizzata all'interno del PIC, perchè permette di effettuare una gestione efficiente dei messaggi ricevuti attraverso opportuni filtri, nonchè di quelli in uscita, attraverso una coda basata su diversi livelli di priorità. Un PIC come il 18F458 contiene tre buffer in trasmissione (*TXB0*, *TXB1*, *TXB2*) e due in ricezione (*RXB0*, *RXB1*) due maschere di input (*RXM0*, *RXM1*) e sei filtri di input (*RXF0*, *RXF1*, *RXF2*, *RXF3*, *RXF4*, *RXF5*). Analizziamo prima la parte relativa alla trasmissione. Ciascun buffer di trasmissione è associato a un registro di controllo chiamato *TXBnCON* (Transmit Buffer n Control Registers) composto dai seguenti bit:

TXREQ (Transmit Request Status Bit): quando è posto a 1 viene inoltrata la richiesta di invio di un messaggio; il suo valore è azzerato quando la trasmissione termina con successo;

TXABT (Transmission Aborted Status Bit): se è uguale a 1 la trasmissione è stata annullata;

TXLARB (Transmission Lost Arbitration Bit): se è uguale a 1 il nodo ha perso il controllo del canale durante la trasmissione;

TXERR (Transmission Error Detected Status Bit): se è uguale a 1 si è verificato un errore durante la trasmissione;

TXPRI0-TXPRI1 (Transmit Priority bits): è una coppia di bit che stabilisce la priorità con cui il messaggio in uscita verrà trattato dallo spooler che gestisce la coda di trasmissione; si va da un minimo di 0 ad un valore massimo di 3 (11 binario); chiaramente tale valore stabilisce l'ordine con cui i messaggi vengono trasmessi ma non influenza assolutamente il campo identificatore del messaggio stesso.

Accanto al registro di controllo TXBnCON troviamo una serie di registri operativi abbinati a ciascun buffer, che permettono il salvataggio dei vari campi di cui è composto il messaggio; i registri in questione sono:

TXBnSIDH (Transmit Buffer n Standard Identifier High Byte Registers): contiene il valore del byte più significativo dell'identificatore in formato standard relativo al messaggio contenuto nel buffer *n* (0<=n<3);

TXBnSIDL (Transmit Buffer n Standard Identifier Low Byte Registers): contiene il valore del byte meno significativo dell'identificatore in formato standard nonchè un bit di abilitazione del formato esteso e i due bit più significativi dell'identificatore in formato esteso relativo al messaggio contenuto nel buffer *n* (0<=n<3);

TXBnEIDH (Transmit Buffer n Extended Identifier High Byte Registers): contiene il valore del byte più significativo dell'identificatore in formato esteso relativo al messaggio contenuto nel buffer *n* (0<=n<3);

TXBnEIDL (Transmit Buffer n Extended Identifier Low Byte Registers): contiene il valore del byte meno significativo dell'identificatore in formato esteso relativo al messaggio contenuto nel buffer *n* (0<=n<3);

TXBnDm (Transmit Buffer n Data Field Byte m Registers): contiene gli "m" byte dati (0<=m<8) relativi al messaggio contenuto nel buffer *n* (0<=n<3). Ricordiamo che il campo dati può essere lungo fino a 8 byte;

TXBnDLC (Transmit Buffer n Data Length Code Registers): contiene la lunghezza del campo dati che va da un minimo di 0 ad un massimo di 8 byte ed un bit chiamato *TXRTR* (Transmission Frame Remote Transmission Request bit) che attiva o meno il flag RTR per cui un nodo anzichè attendere passivamente delle informazioni da un altro dispositivo le richiede direttamente.

Per quanto riguarda la ricezione di messaggi dobbiamo considerare che la struttura si compone di due buffer di ricezione ognuno dei quali è in grado di contenere un intero messaggio. Esistono, quindi, dei registri esattamente complementari a quelli visti per la trasmissione che contengono i vari campi del messaggio (*RXBnSIDH*, *RXBnSIDL* ecc.). A supporto di tali registri c'è il cosiddetto *MAB* (Message Assembly Buffer) il quale non fa altro che acquisire il prossimo messaggio presente sul bus fornendo un terzo livello di bufferizzazione. Il messaggio in questione viene trasferito al relativo buffer *RXBn* soltanto se soddisfa i criteri stabiliti dai filtri di input. Dobbiamo fare una distinzione tra filtro e maschera di input. Una maschera stabilisce la posizione dei bit dell'identificatore da >

Tabella 2

Modalità	Valore REQOP (CANCON bit 7..5)	Descrizione
CONFIGURATION	1xx (i valori x sono di tipo "not-care" pertanto possono essere sia ad 1 che a 0)	In questa modalità è possibile accedere a tutti i registri di configurazione del nodo (cosa che non può avvenire nelle altre modalità). Il nodo non riceve nè trasmette, i contatori per gli errori vengono azzerati, ed i flag relativi agli interrupt rimangono costanti. In questo modo si proteggono da un accesso non controllato i registri fondamentali relegando la loro valorizzazione ad una ben determinata fase di configurazione.
DISABLE	001	Il modulo CAN viene disattivato mantenendo soltanto la possibilità di intercettare un interrupt di Wake-Up basato sul traffico presente sul bus.
NORMAL	000	È la modalità di funzionamento standard. Il nodo opera nel pieno delle sue funzionalità ricevendo ed inviando messaggi.
LISTEN ONLY	011	In questa modalità il nodo riceve tutti i messaggi che transitano sul bus, anche quelli con errori. Può essere utilizzata per attività di monitoraggio del canale. Attraverso i registri di filtro è possibile selezionare i messaggi in ingresso da controllare. Il nodo opera in una modalità "silenziosa" nel senso che non trasmette nessun messaggio in uscita, inoltre i contatori degli errori vengono azzerati e disattivati.
LOOPBACK	010	Il modulo CAN viene configurato in maniera tale che i messaggi in uscita vengano caricati nel buffer di ingresso in maniera da essere interpretati come messaggi in entrata. Si tratta di una modalità molto interessante per attività di testing e di sviluppo visto che permette di verificare il comportamento del nodo alla ricezione di un ben determinato messaggio.

estrarre mentre il filtro stabilisce il valore che tali bit devono avere affinchè il messaggio venga accettato dal nodo. Si faccia attenzione che il buffer *RXB0* si occupa dei messaggi ad alta priorità ed ha due filtri associati mentre *RXB1* elabora i messaggi a bassa priorità ed ha 4 filtri associati. Inoltre è possibile fare in modo che *RXB0* funzioni in una modalità "Double-Buffer". In pratica, nel momento in cui *RXB0* contiene un messaggio valido, se ne riceve un altro non si verifica un errore di overflow ma il messaggio viene passato al buffer *RXB1* e viene trattato secondo i filtri di quest'ultimo.
Accanto ai registri associati alle operazioni di trasmissione e ricezione il PIC contiene diverse altre strutture di controllo, vediamo quelle che interessano direttamente le modifiche effettuate nei parametri del file *ECAN.def*.

CIOCAN (CAN I/O CONTROL REGISTER): questo registro contiene due bit. Il primo, chiamato *CANCAP*, permette di far sì che il modulo CAN generi un time-stamp per ogni messaggio ricevuto. Una volta attivo, il modulo CAN genera un segnale per il pin *CCP1*; quest'ultimo viene configurato in maniera tale che i quattro bit *3:0* di *CCP1CON* siano a 0011b (CCP special event trigger for CAN events). In questo modo alla ricezione di un messaggio viene estratto il valore del *Timer1* o del *Timer3* che viene utilizzato proprio come time-stamp. Il secondo bit, chiamato *ENDRHI*, stabilisce in che modo viene comandato il pin *CANTX1* in uno stato recessivo (assume la tensione Vdd oppure utilizza una modalità tipo "Tri-State").
CANCON (CAN CONTROL REGISTER): con tre bit di questo registro è possibile stabilire la modalità operativa del nostro nodo. In pratica è possibile scegliere tra 5 opzioni: configuration, disable, normal, listen only, loopback. Ne riassumiamo le caratteristiche nella Tabella 2.
Nella realtà è possibile attivare una sesta modalità operativa chiamata "Error Recognition Mode", ponendo a 11b (=3 in decimale) due bit (denominati *RXM*) del registro *RXBnCOM*. In tal modo tutti i messaggi in entrata, sia validi che invalidi, vengono accettati e trasferiti ai buffer di ricezione.
BRGCON1:BRGCON3: insieme di tre registri che permette di stabilire il bit-rate, le metodologie di sincronizzazione e il tipo di campionamento da usare. È importante considerare che ogni nodo deve utilizzare il medesimo bit-rate nominale definito come numero di bit trasmessi al secondo in un trasmettitore ottimale. Il protocollo CAN utilizza una codifica di tipo NRZ (Non-Return-to-Zero) che non integra nello stream trasmesso il segnale di clock. Pertanto i nodi in ricezione devono recuperare il clock e sincronizzarsi con quelli in trasmissione. Per farlo correttamente la classe di PIC18FXX8 utilizza un DPLL (Digital Phase Lock Loop) che suddivide ciascun bit-time in più segmenti sulla base di un intervallo di tempo minimo chiamato "Time Quanta" (TQ). Siccome non è detto che tutti i nodi utilizzino la medesima frequenza di sistema (quella, per intenderci, derivante dall'oscillatore collegato ai pin OSC1 e OSC2), è necessario effettuare una precisa configurazione di ciascuno di essi determinando il valore del BRP (Baud Rate Prescaler intero compreso tra 0 e 63) ed il numero dei "Time Quanta" per ciascun segmento. Premetto che durante gli esperimenti del nostro corso saremo facilitati dal fatto che utilizzeremo per tutti i nodi la medesima configurazione circuitale e quindi la medesima frequenza di clock. Il bit-time nominale si può definire come il reciproco del bit-rate nominale ed è composto da 4 segmenti:

- *Synchronization Segment (Sync_Seg)*: permette la sincronizzazione tra i diversi nodi, durante questo intervallo di tempo che è lungo 1 TQ , il nodo si aspetta la transizione di livello logico del segnale in ingresso.
- *Propagation Time Segment (Prop_Seg)*: permette di compensare il ritardo derivante dal tempo di propagazione del segnale sul bus e quello derivante dal naturale ritardo interno dei nodi. La sua lunghezza può essere configurata tra un minimo di 1 ad un massimo di 8 TQ.
- *Phase Buffer Segment 1 (Phase_Seg1) e Phase Buffer Segment 2 (Phase_Seg2)*: permettono di ottimizzare la posizione del punto di campionamento (cioè del punto nel quale viene letto il livello logico presente sulla linea di ingresso e quindi il valore del bit ricevuto) all'interno del bit-time. La loro lunghezza può essere configurata da un minimo di 1 ad un massimo di 8 TQ.

La figura 6 permette di chiarire la successione dei segmenti (si consideri che graficamente ogni coppia di linee verticali corrisponde ad un TQ). Il bit time nominale va da un minimo di 8 TQ ad un massimo di 25 TQ. Si definisce Information

Processing Time (IPT) il tempo intercorrente tra il punto di campionamento e la determinazione del livello logico del bit successivo. Secondo le specifiche CAN questo intervallo deve essere inferiore o uguale a 2 TQ. Per il PIC18F458 esso è pari a 2, pertanto, durante la programmazione, la lunghezza del Phase Buffer Segment 2 deve essere maggiore o uguale a 2. Sulla base della frequenza del clock di sistema e quindi del quarzo utilizzato, possiamo stabilire il bit rate nominale attraverso un semplice calcolo; premesso che:
TQ (µs.) = [2x(BRP+1)]/Fosc (MHz) e
TBIT(µs.) = TQ (µs.) x numero di TQ per bit time nominale e *bit rate nominale (bit/s) = 1/TBIT*, se poniamo Fosc=20 MHz, BRP=4 e il bit time nominale pari a 16 TQ (il doppio di quanto visto nella Fig. 6) avremo:
TQ = [2x(4+1)]/20 = 0,5 µs.
e *TBIT = 16x0,5 = 8 µs.*, oltre a
bit rate nominale = 1/(8x10⁻⁶)= 125.000 bit/s = 125 kbps.

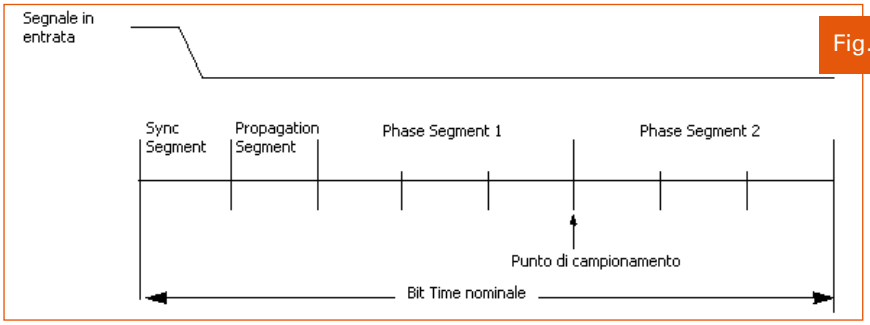


Fig. 6

Per effettuare una risincronizzazione, il PIC può operare sulla lunghezza dei segmenti *Phase 1* e *Phase 2*, fermo restando che la lunghezza del primo segmento può essere solo allungata, mentre quella del secondo può essere diminuita di un intervallo che viene chiamato *Synchronization Jump Width* (*SJW*). Il valore dell'intervallo va da un minimo di una ad un massimo di quattro volte TQ e può essere configurato tramite il registro *BRGCON1*. In linea generale, considerando la buona stabilità degli oscillatori ceramici, nella maggior parte dei casi è più che sufficiente un valore pari ad 1. Dopo aver visto la struttura hardware con cui un PIC implementa il modulo CAN, possiamo analizzare con maggior consapevolezza la tabella dei parametri precisabili all'interno del file *ECAN.def* (vedere la Tabella 3 a fondo pagina).
Finalmente abbiamo raggiunto il punto di contatto tra la parte teorica e quella pratica di questo corso. La presenza di una libreria C18 in grado di gestire sia la modalità standard che quella estesa del protocollo CAN, ha comportato la necessità di sottoporvi qualche paragrafo teorico in più, ma a questo punto abbiamo tutte le informazioni necessarie e sufficienti per iniziare il nostro primo sviluppo firmware per CAN bus. Utilizzeremo lo stesso approccio usato anche in >

Tabella 3

Parametro	Valori Possibili	Descrizione
ECAN_Bn_AUTORTR_MODE	ECAN_AUTORTR_MODE_DISABLE ECAN_AUTORTR_MODE_ENABLE	Configura il buffer di trasmissione n (con n compreso tra 0 e 5) nella modalità RTR (Remote Transmission Request) che permette di gestire automaticamente l'interrogazione di un nodo remoto. Può essere utilizzato nel modo 1 e 2 quindi con piena compatibilità con lo standard esteso.
ECAN_Bn_MODE_VAL	ECAN_RECEIVE_ALL_VALID ECAN_RECEIVE_STANDARD ECAN_RECEIVE_EXTENDED ECAN_RECEIVE_ALL	Configura il buffer di ricezione n (con n compreso tra 0 e 5) precisando le tipologie di messaggi che è abilitato a ricevere. Può essere utilizzato nel modo 1 e 2.
ECAN_Bn_TXRX_MODE_VAL	ECAN_BUFFER_TX ECAN_BUFFER_RX	Configura il buffer n (con n compreso tra 0 e 5) in trasmissione o ricezione. Può essere utilizzato nel modo 1 e 2.
ECAN_BRP_VAL	da 1 a 8	Stabilisce il valore da assegnare al BRP (Baud Rate Prescaler).
ECAN_BUS_SAMPLE_MODE_VAL	ECAN_BUS_SAMPLE_MODE_THRICE ECAN_BUS_SAMPLE_MODE_ONCE	Stabilisce se il campionamento della linea di ricezione necessario per stabilire il suo livello logico deve avvenire in una sola fase (al punto di campionamento) o in tre fasi (una precedente al punto di campionamento e due in corrispondenza dello stesso).
ECAN_CAPTURE_MODE_VAL	ECAN_CAPTURE_MODE_DISABLE ECAN_CAPTURE_MODE_ENABLE	Permette di attivare o disattivare la possibilità di generare un timestamp per ogni messaggio ricevuto attraverso il campionamento del segnale su CCP1. È necessario configurare il registro CCP1CON abilitando la modalità "CCP special event trigger for CAN events".

Continuazione Tabella **3**

Parametro	Valori Possibili	Descrizione
ECAN_FILTER_MODE_VAL	ECAN_FILTER_MODE_ENABLE ECAN_FILTER_MODE_DISABLE	Attiva o disattiva l'uso di un filtro passa-basso per la rilevazione dell'attività del bus nella modalità di wake-up. In pratica configura il flag WAKFIL del registro BRGCON3.
ECAN_FUNC_MODE_VAL	ECAN_MODE_0 ECAN_MODE_1 ECAN_MODE_2	Definisce la modalità operativa del nodo secondo quanto visto nei precedenti paragrafi.
ECAN_INIT_MODE	ECAN_INIT_NORMAL ECAN_INIT_CONFIGURATION ECAN_INIT_LOOPBACK ECAN_INIT_DISABLE ECAN_INIT_LISTEN_ONLY	Definisce il modo di funzionamento iniziale secondo quanto visto nei precedenti paragrafi.
ECAN_LIB_MODE_VAL	ECAN_LIB_MODE_FIXED ECAN_LIB_MODE_RUNTIME	Definisce la possibilità di modificare le configurazioni del nodo a run-time (come ad esempio la modalità di funzionamento).
ECAN_PHSEG1_VAL	da 1 a 8	Stabilisce la lunghezza del segmento Phase 1.
ECAN_PHSEG2_MODE_VAL	ECAN_PHSEG2_MODE_PROGRAMMABLE ECAN_PHSEG2_MODE_AUTOMATIC	Permette di stabilire se il segmento Phase 2 deve essere gestito automaticamente dal modulo ECAN o può essere programmato liberamente.
ECAN_PHSEG2_VAL	da 1 a 8	Stabilisce la lunghezza del segmento Phase 2.
ECAN_PROPSEG_VAL	da 1 a 8	Stabilisce la lunghezza del segmento Propagation.
ECAN_RXB0_DBL_BUFFER_MODE_VAL	ECAN_DBL_BUFFER_MODE_DISABLE ECAN_DBL_BUFFER_MODE_ENABLE	Attiva o disattiva la modalità double-buffer per RXB0. È utilizzabile solo in modo 0.
ECAN_RXB0_MODE_VAL	ECAN_RECEIVE_ALL_VALID ECAN_RECEIVE_STANDARD ECAN_RECEIVE_EXTENDED ECAN_RECEIVE_ALL	Stabilisce la tipologia di messaggi in ingresso accettati da RXB0.
ECAN_RXB1_MODE_VAL	ECAN_RECEIVE_ALL_VALID ECAN_RECEIVE_STANDARD ECAN_RECEIVE_EXTENDED ECAN_RECEIVE_ALL	Stabilisce la tipologia di messaggi in ingresso accettati da RXB1.
ECAN_RXF0_MASK_VAL	ECAN_RXM0 ECAN_RXM1 ECAN_RXMF15	Collega RXF0 ad una ben determinata maschera. È utilizzabile nel modo 1 e 2. Si noti che è possibile configurare il filtro RXF15 come una maschera di input.
ECAN_RXFn_BUFFER_VAL	RXB0 RXB1 B0 B1 B2 B3 B4 B5	Collega un filtro ad uno specifico buffer in ricezione. È utilizzabile nel modo 1 e 2.
ECAN_RXFn_MODE_VAL	ECAN_RXFn_ENABLE ECAN_RXFn_DISABLE	Permette di abilitare o disabilitare i filtri in ricezione RXFn con n che va da 0 a 15. Può essere usato nella modo 1 e 2. Nel modo 0 n va da 0 a 5 e i filtri relativi sono tutti attivi.
ECAN_RXFn_MSG_TYPE_VAL	ECAN_MSG_STD ECAN_MSG_XTD	Permette di definire la modalità standard o estesa per i filtri RXFn. Nel modo 0 sono disponibili solo RXF0-RXF5.
ECAN_RXFn_VAL	un valore a 11 bit o a 29 bit	Assegna il valore a 11 bit o a 29 bit al filtro relativo RXFn.
ECAN_RXMn_MSG_TYPE	ECAN_MSG_STD ECAN_MSG_XTD	Definisce la modalità standard o estesa per le maschere di input RXM0-RXM1.
ECAN_SJW_VAL	da 1 a 4	Stabilisce la lunghezza del SJW (Synchronization Jump Width).
ECAN_TX2_MODE_VAL	ECAN_TX2_MODE_DISABLE ECAN_TX2_MODE_ENABLE	Abilita o disabilita il pin CANTX2 nei PIC che lo prevedono.
ECAN_TX2_SOURCE_VAL	ECAN_TX2_SOURCE_COMP ECAN_TX2_SOURCE_CLOCK	Definisce la sorgente di segnale per il pin CANTX2 che può essere il complemento del segnale presente su CANTX1 oppure il clock del modulo CAN.
ECAN_TXDRIVE_MODE_VAL	ECAN_TXDRIVE_MODE_TRISTATE ECAN_TXDRIVE_MODE_VDD	Definisce in che modo il pin CANTX1 viene comandato in uno stato recessivo.
ECAN_WAKEUP_MODE_VAL	ECAN_WAKEUP_MODE_ENABLE ECAN_WAKEUP_MODE_DISABLE	Attiva o Disattiva la modalità di Wake-up per il nodo.

precedenti corsi, secondo cui le singole istruzioni (in questo caso sarebbe meglio dire funzioni) verranno spiegate in dettaglio al momento del loro utilizzo all'interno del codice.
Il nostro primo esperimento ci permetterà di capire come un nodo possa acquisire dei dati dall'esterno (ad esempio da una sonda termometrica) e renderli disponibili sul bus affinchè possano essere utilizzati dagli altri nodi. È il primo passo verso la descrizione della cooperazione tra più dispositivi CAN, obiettivo del quale è arrivare a considerare un insieme di nodi CAN come un unico sistema in grado di svolgere più compiti, distribuendo le informazioni necessarie.

Corso di programmazione: CAN BUS

a cura di Carlo Tauraso

Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa quarta puntata analizziamo come un modulo può acquisire dei dati e renderli disponibili sul bus.

4

Il mese scorso ci siamo lasciati dopo aver esaminato l'hardware che realizza il nostro modulo CAN, con la promessa di vedere come esso possa acquisire dati dall'esterno e renderli disponibili ad altri moduli sul bus.

Esperimento1: Sonda Termo CAN

Iniziamo a mettere in pratica i concetti teorici visti nelle precedenti puntate, attraverso un primo esperimento che ci permetterà di vedere da vicino la comunicazione standard tra due nodi CAN. La nostra prima rete, infatti, sarà costituita da una coppia di nodi: uno avrà il compito di rilevare la temperatura ambientale attraverso una sonda DS18B20 e di inviare i relativi valori su un bus CAN, l'altro provvederà a memorizzare i valori ricevuti su una EEPROM 24LC256. Non ci preoccuperemo di formattare i dati ricevuti, ma li registreremo così come vengono forniti dalla sonda. Ci concentreremo, infatti, sulla comunicazione CAN, analizzando due funzioni fondamentali della libreria ECAN: "*ECANSendMessage*" e "*ECANReceiveMessage*". Esse costituiscono la base di qualsiasi firmware che coinvolga la comunicazione su bus CAN. Vogliamo, inoltre, descrivere due aspetti tipici dell'ambiente C18: l'integrazione e il riutilizzo del codice. In pratica lo sviluppo nascerà attraverso il riutilizzo di librerie di pubblico dominio e la loro integrazione e personalizzazione finalizzata agli obiettivi che intendiamo raggiungere. Questa modalità operativa permette di aumentare la propria produttività e di realizzare firmware facilmente personalizzabile e manutenibile. Il nostro progetto deve essere in grado di dialogare con una sonda termometrica attraverso il protocollo *one-wire*, comunicare i dati su CAN bus, scriverli su una EEPROM attraverso l'inter- >

Tabella 1

Funzionalità	File	Descrizione
Protocollo onewire	onewire.c onewire.h	Raggruppa tutte le funzioni che permettono di effettuare il reset di un dispositivo onewire, inviargli e ricevere un byte dallo stesso. È una riscrittura delle istruzioni OWIN, OWOUT del PICBasic.
CAN Bus	ecan.c ecan.h ecan.def	È la libreria di base che utilizzeremo per tutti i progetti su CAN Bus. Nasce per la famiglia PIC18 a 64,68,80 pin ma come vedremo si può adattare con estrema facilità anche ai più modesti 18F458 o comunque a tutti i PIC18 dotati di modulo CAN.
EEPROM	xeeprom.c xeeprom.h	Si tratta di una libreria utilizzata per l'accesso, lettura, scrittura di EEPROM con indirizzamento a 16 bit. È ottima per sfruttare la 24LC256 montata sui nostri nodi CAN.
RS-232	usart.h	È una libreria standard distribuita con il compilatore C18. Permette di gestire tutti gli aspetti della comunicazione seriale secondo lo standard RS-232. Noi la useremo per i messaggi di controllo inviati al PC dal nodo trasmittente.

faccia I²C. Infine, nel nodo trasmittente implementeremo anche un po' di messaggistica a scopo diagnostico e di controllo, che lavorerà su RS232. Tutte queste funzionalità sono raggruppabili in apposite librerie. La struttura generale del progetto è sintetizzata in Tabella1.
In aggiunta utilizzeremo due file che sfrutteremo in diverse altre occasioni:

1) ***C18cfg.asm***: contiene i valori per i bit di configurazione del PIC, come quello relativo alla disabilitazione del Watchdog Timer o della sorgente di clock del sistema; nello stesso inseriamo il file *.inc* relativo al microcontrollore che stiamo utilizzando (in questo caso *p18f458.inc*).

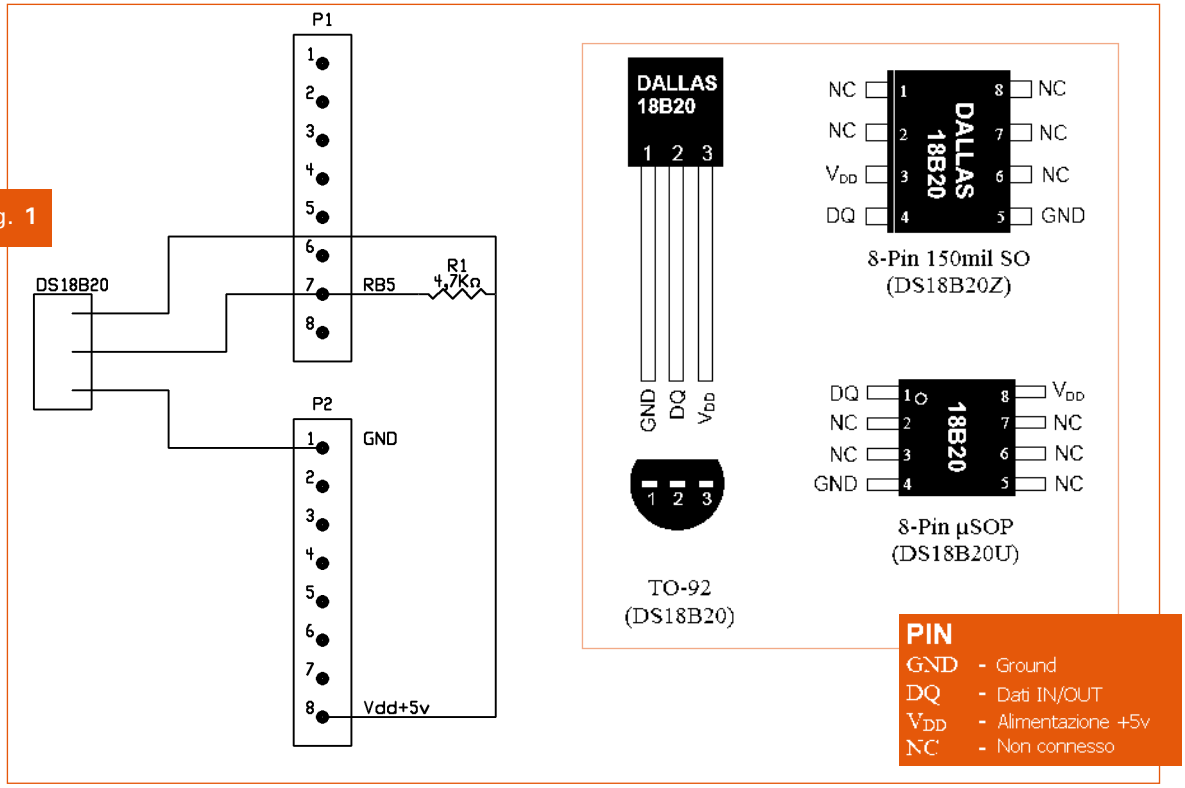
2) ***compilatore.h***: all'interno di questo file raggruppiamo tutte le definizioni necessarie al rimappaggio dei registri del microcontrollore e di quei valori che interesseranno l'intero codice del nostro progetto.
Per esempio, ridefiniamo il pin relativo alla linea dati della sonda termometrica (RB5) come PORTB_RB5: scopo dell'operazione è facilitarne l'utilizzo.

Per rendere più chiaro lo sviluppo, analizziamo separatamente i due nodi. Nel pacchetto scaricabile dal sito della rivista troverete due cartelle denominate, rispettivamente, *firmNODOTX* e *firmNODORX*, allo scopo di distinguere i due sotto-progetti. Partiamo dal nodo trasmittente.

Nodo TX

Sul nodo trasmittente colleghiamo una sonda ter-

Fig. 1



mometrica DS18B20. Utilizziamo il pin RB5 del PIC18F458 come linea dati. I due restanti terminali della sonda devono essere collegati uno alla massa (GND) e l'altro alla linea del +5 volt (alla Vdd). È necessario utilizzare una resistenza di pull-up da 4,7kohm, da collegare al piedino di uscita del sensore Dallas; per rendere le cose più semplici, possiamo utilizzare direttamente le strip laterali della nostra demo-board, secondo lo schema di Fig. 1.
Una volta effettuato il collegamento della sonda, predisponiamo il nostro firmware.
Fondamentalmente dobbiamo realizzare un ciclo di lettura dei registri della sonda relativi alla temperatura ed il loro invio sul bus CAN. Per venire incontro all'utente che dovrà utilizzare il sistema, facciamo in modo che il campionamento avvenga non appena viene premuto lo switch 2 della demo-board, quello collegato al pin RB0 del PIC. L'invio delle letture si fermerà non appena verrà premuto a lungo lo stesso tasto. Abbiamo deciso di effettuare una misura ad intervalli regolari di circa 1 secondo. Lo stato del nodo verrà segnalato dai tre led colorati; in particolare, all'avvio verrà acceso il led verde per segnalare che il nodo è pronto a ricevere il comando di start attraverso SW2.
Durante il campionamento vedremo lampeggiare il led rosso. Infine, dopo l'ulteriore pressione del tasto il nodo terminerà le operazioni di trasmissione e farà illuminare il led giallo. Tutte le fasi di elaborazione si potranno seguire attraverso una sessione di Hyper Terminal aperta direttamente sulla porta seriale dove avremo collegato la nostra demo-board; la COM dovrà essere impostata secondo i seguenti parametri: velocità di 19.200 bps, 8 bit di dati, nessuna parità, 1 bit di stop (19200-8-N-1).
Il primo problema che abbiamo dovuto affrontare è stato quello di scrivere una piccola libreria che permettesse di eseguire tutte le operazioni relative alla comunicazione con la sonda; è stato facile risolverlo facendo riferimento ai datasheet del componente, dove è spiegato dettagliatamente come la DS18B20 va interrogata, quali risposte dà e in che formato esprime la temperatura.
Naturalmente, le funzioni così costruite saranno utilissime ogni volta che dovremo dialogare con un componente *one-wire*. La sonda in questione ci permetterà di misurare la temperatura in gradi Celsius, con una accuratezza di mezzo grado in un range che va da -10°C a +85°C.
Il valore viene espresso con due byte, i cui bit rappresentano sia il modulo che il segno; è possibile definire quanti bit usare per l'uno e quanti destinare all'altro: nel nostro caso optiamo per la formula 11 bit di valore assoluto (modulo) e cinque di segno. Dunque, ad ogni richiesta la sonda risponde con due byte i cui bit verranno resi disponibili in due appositi registri, secondo il formato illustrato nella Fig. 3. I bit S permettono di stabilire il segno della temperatura rilevata (S=0 per valori positivi, S=1 per valori negativi). Ad esempio con un valore pari a 00A2h avremo una temperatura pari a 10,125°C mentre con FFF8h avremo una temperatura di -0,5°C.

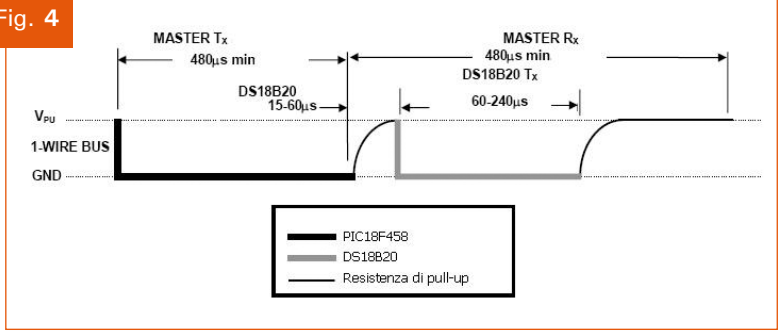
Il protocollo "OneWire"

Il sistema di comunicazione previsto per il DS18B20 richiede l'utilizzo di un protocollo particolare ma molto efficace. Qualsiasi operazione viene iniziata dal dispositivo master, che in questo caso è rappresentato dal nostro PIC. La prima fase da considerare è il reset del dispositivo, che consiste in un impulso proveniente dal master, cui la sonda risponde inviando un segnale di presenza da essa generato. Per chiarirvi le idee riguardo a come avviene la fase di reset date un'occhiata allo schema di >

Fig. 3

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LS Byte	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
MS Byte	S	S	S	S	S	2^6	2^5	2^4

Fig. 4



Listato 1

```
/*********************************************************************
 * Funzione:         unsigned char OWReset(void)
 * Input:              Nessuno
 * Output:           1 = Dispositivo presente
 *                       0 = Dispositivo non presente
*********************************************************************/
unsigned char OWReset(void)
{
     unsigned char pd;

     pd = 0;
     TRISB_RB5 = 0;
     PORTB_RB5 = 0;
     Delay10TCYx(240);
     TRISB_RB5 = 1;
     Delay10TCYx(40);
     if (PORTB_RB5 ==0)
     pd = 1;
     Delay10TCYx(200);
     return pd;
}
```

Il PIC mette a livello basso la linea DQ per 480µs.

Il PIC rilascia il bus, si pone in ricezione e attende 80µs in maniera da rilevare l'impulso di presenza proprio all'interno dell'intervallo relativo.

Se viene rilevato il livello logico basso la sonda ha risposto correttamente quindi viene valorizzato ad 1 il parametro in uscita.

Fig. 4. Durante questa fase il microcontrollore porta a livello basso la linea dati per un intervallo minimo di 480 µs. Successivamente si pone in ricezione e, allo scopo, disimpegna il bus; quindi la resistenza di pull-up porta la linea a livello logico alto. Non appena la sonda rileva questo cambiamento, attende per un intervallo di tempo che va da un minimo di 15 ad un massimo di 60 µs, poi invia un impulso di presenza ponendo la linea a livello basso per un intervallo che va da 60 a 240 µs. Al termine rilascia nuovamente il bus, pertanto la resistenza di pull-up riporta la linea dati a livello alto.
L'intera procedura può essere sviluppata facilmente in C18. Raggruppiamo le relative istruzioni nella funzione visibile nel Listato 1.
La funzione *Delay10TCYx(n)* fa parte della libreria standard C18 e permette di inserire un ritardo pari a 10^n cicli di clock. È chiaro che la temporizzazione dipende dalla frequenza di clock dell'oscillatore utilizzato (nel nostro caso 20 MHz). Per sfruttarla è sufficiente inserire la seguente istruzione:

#include <delays.h>

La "*OWReset*" ritorna un valore pari a 1 se il PIC riceve l'impulso di presenza da parte della sonda, quindi può essere utilizzata all'interno di un'espressione logica. Analogamente, abbiamo preparato la funzione "*OWTX*" per trasmettere un byte al dispositivo e la "*OWRX*" per ricevere sempre un byte dallo stesso. Per ragioni di spazio non analizziamo il loro listato. I più curiosi possono soddisfare tutti i loro dubbi aprendo il file di progetto *CANTX.mcp* e facendo doppio clic sul file "*onewire.c*". Tutte le dichiarazioni relative a ciascuna funzione sono state raccolte nel file *onewire.h*. Ecco un elenco sintetico delle diverse procedure:
- unsigned char **OWReset** (void): esegue l'inizializzazione della sonda rilevandone la presenza sul bus; ritorna il valore 1 se la sonda è presente;
- void **OWTX** (unsigned char): invia al dispositivo collegato il byte passato come parametro;
- unsigned char **OWRX** (void): riceve un byte dal dispositivo collegato e lo passa attraverso il parametro di uscita;
- unsigned char **OWRX1** (void): riceve un unico bit dal dispositivo; viene utilizzato per verificare se il rilevamento della temperatura è terminato, oppure no.

Per quanto riguarda il rilevamento della temperatura, è necessario utilizzare delle specifiche sequenze al fine di comandare il DS18B20. Inviando la coppia di byte **CCh-44h** viene avvia-

Tabella 2

Byte 0	Temperatura LSB
Byte 1	Temperatura MSB
Byte 2	Registro TH
Byte 3	Registro TL
Byte 4	Registro Configurazione
Byte 5	Riservato FFh
Byte 6	Riservato 0Ch
Byte 7	Riservato 10h
Byte 8	CRC

Listato 2

```
OWReset();
OWTX(0xCC);
OWTX(0x44);
while (OWRX1());
OWReset();
OWTX(0xCC);
OWTX(0xBE);
tempera.bytes.LSB = OWRX();
tempera.bytes.MSB = OWRX();
for (indice=1;indice<=7;indice++)
conv=OWRX();
```

Reset della sonda e avvio del rilevamento della temperatura.

Attesa fino al termine del rilevamento.

Richiesta dei 9 byte contenuti nei registri della sonda.

Registrazione dei primi due byte.

Scartati gli ultimi 7 byte.

ta una conversione. Durante l'operazione il dispositivo invia uno 0, mentre appena termina trasmette un 1 logico. A questo punto è possibile inviare la sequenza **CCh-BEh** per leggere i registri della sonda, strutturati come in Tabella 2. Per i nostri scopi è sufficiente estrarre i primi due byte. Utilizzeremo, infatti, la configurazione predefinita della sonda e durante la lettura scarteremo i rimanenti 7 byte.

Il codice C18 che permette di reperire le informazioni che dovremo, poi, trasferire via CAN bus, è quello presente nel Listato 2. Si noti che i due byte relativi al valore della temperatura rilevata vengono trasferiti in due campi da 8 bit di cui si compone la variabile *tempera*. Per effettuare questa assegnazione in maniera corretta è stato necessario definire un'opportuna struttura, meglio descritta dal codice visibile nel Listato 3.

Listato 3

```
union
    {
        unsigned short Val;
        struct
        {
            unsigned char LSB; /*Last significant byte*/
            unsigned char MSB; /*Most significant byte*/
        } bytes;
    } tempera;      /*Temperatura divisa in due campi*/
```

La comunicazione RS232

Per poter inviare dei messaggi ad un PC in maniera da permettere all'utente di seguire le varie fasi dell'elaborazione, dobbiamo utilizzare un'altra libreria distribuita assieme al compilatore C18. Anche in questo caso sfruttiamo un'istruzione di "include" relativa al file *usart.h* che contiene la dichiarazione delle varie funzioni utilizzabili; vediamo, nel concreto, quelle necessarie al nostro obiettivo. Innanzitutto dobbiamo inizializzare la porta di comunicazione secondo i parametri definiti durante l'analisi iniziale del progetto. La funzione corrispondente è la seguente: >

Tabella 3

Valori	Descrizione
USART_TX_INT_ON USART_TX_INT_OFF	Attiva/Disattiva il segnale di interrupt in trasmissione
USART_RX_INT_ON USART_RX_INT_OFF	Attiva/Disattiva il segnale di interrupt in ricezione
USART_ASYNCH_MODE USART_SYNCH_MODE	Attiva la modalità di trasmissione/ricezione asincrona o sincrona
USART_EIGHT_BIT USART_NINE_BIT	Trasmette/Riceve dati a 8 bit o a 9 bit
USART_SYNC_SLAVE USART_SYNC_MASTER	In modalità di trasmissione/ricezione sincrona configura il modulo come slave o come master
USART_SINGLE_RX USART_CONT_RX	Stabilisce se la ricezione deve avvenire in maniera continuativa o per un singolo pacchetto
USART_BRGH_HIGH USART_BRGH_LOW	Stabilisce se il modulo deve essere inizializzato per un baud rate elevato o meno

void OpenUSART (unsigned char ***config***, unsigned int ***spbrg***);

Il primo parametro è creato attraverso un'operazione di AND tra una serie di valori definiti nel file *usart.h* che permettono di stabilire il preciso funzionamento del modulo seriale del PIC18F458, modulo che fa capo ai pin RC6 e RC7 (vedere la Tabella 3).
Il secondo parametro precisa la velocità di comunicazione. Nel caso di una ricezione/trasmissione di tipo asincrono, il valore viene calcolato sulla base della seguente formula:

FOSC/(16* (nr di bit al secondo + 1))

utilizzando come FOSC il valore della frequenza di oscillazione del cristallo utilizzato nel circuito. Nel nostro caso, considerando un FOSC equivalente a 20.000.000 (20MHz) e una velocità di 19.200bps, il valore del parametro è:

((20.000.000/19.200)/16)-1=64,104

Ne deriva che usiamo il 64. Tale risultato presuppone l'utilizzo di un prescaler pari a 16. L'istruzione che utilizziamo per l'inizializzazione della porta è la seguente:

OpenUSART(USART_TX_INT_OFF&USART_RX_INT_OFF&USART_ASYNCH_MODE&USART_EIGHT_BIT&USART_CONT_RX&USART_BRGH_HIGH, 64);

Per inviare una qualsiasi stringa attraverso la porta seriale, possiamo utilizzare la *putrsUSART*. Essa prende in ingresso il puntatore ad una stringa di caratteri; in generale si utilizza la *putsUSART* per stringhe registrate nella memoria dati, mentre la *putrsUSART* è usata per stringhe della memoria di programma.
La dichiarazione che troviamo all'interno di *usart.h* è la seguente:

void putrsUSART(const rom char ***data**);

L'utilizzo è molto semplice. È sufficiente passare come argomento la stringa racchiusa tra apici, come si vede nell'istruzione seguente:

putrsUSART("Elettronica In \n\r");

Utilizziamo \n e \r per i caratteri, rispettivamente, di *new line* e *carriage return* necessari per fare andare a capo il cursore.

Le definizioni per il bus CAN

Dopo aver visto un po' di codice di contorno, andiamo ad analizzare il cuore del nostro firmware e quindi la parte necessaria ad effettuare la comunicazione sul bus CAN. Innanzitutto dobbiamo considerare il file *ECAN.def* che contiene tutti i parametri necessari a configurare il modulo CAN del PIC18F458. Per comodità utilizziamo l'interfaccia grafica di Microchip Application Maestro, tuttavia per effettuare le relative modifiche siete liberi di utilizzare un qualsiasi editor di testo.
La Tabella 4 rappresenta l'intera sequenza di parametri utilizzati nei nostri nodi; per la descrizione dei campi vi rimandiamo alle spiegazioni del numero precedente.
Vengono usati i puntini di sospensione laddove i valori si ripetono per più parametri numerati sequenzialmente; ad esempio, il valore *0x0L* per il parametro *ECAN_RXF2_VAL* si ripete anche per *ECAN_RXF3_VAL*, *ECAN_RXF4_VAL*, *ECAN_RXF5_VAL*.
Si presti attenzione al fatto che utilizzeremo una modalità di funzionamento costante per tutta l'elaborazione (*ECAN_LIB_MODE_FIXED*) con protocollo standard (*ECAN_MODE_0*) e gestione delle operazioni sia di trasmissione che di ricezione

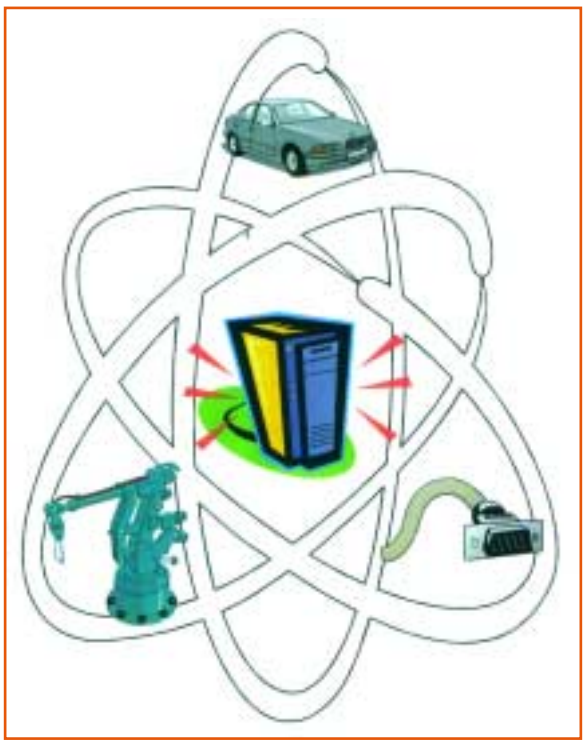


Tabella 4

Parametro	Valore
ECAN_LIB_MODE_VAL	ECAN_LIB_MODE_FIXED
ECAN_FUNC_MODE_VAL	ECAN_MODE_0
ECAN_INIT_MODE	ECAN_INIT_NORMAL
ECAN_SJW_VAL	2
ECAN_BRP_VAL	4
ECAN_PHSEG1_VAL	8
ECAN_PHSEG2_VAL	8
ECAN_PROPSEG_VAL	8
ECAN_PHSEG2_MODE_VAL	ECAN_PHSEG2_MODE_PROGRAMMABLE
ECAN_BUS_SAMPLE_MODE_VAL	ECAN_BUS_SAMPLE_MODE_THRICE
ECAN_WAKEUP_MODE_VAL	ECAN_WAKEUP_MODE_ENABLE
ECAN_FILTER_MODE_VAL	ECAN_FILTER_MODE_DISABLE
ECAN_TXDRIVE_MODE_VAL	ECAN_TXDRIVE_MODE_VDD
ECAN_TX2_MODE_VAL	ECAN_TX2_MODE_DISABLE
ECAN_TX2_SOURCE_VAL	ECAN_TX2_SOURCE_COMP
ECAN_CAPTURE_MODE_VAL	ECAN_CAPTURE_MODE_DISABLE
ECAN_RXB0_MODE_VAL	ECAN_RECEIVE_ALL_VALID
ECAN_RXB0_DBL_BUFFER_MODE_VAL	ECAN_DBL_BUFFER_MODE_DISABLE
ECAN_RXB1_MODE_VAL	ECAN_RECEIVE_ALL_VALID
ECAN_B1_TXRX_MODE_VAL	ECAN_BUFFER_TX
ECAN_B1_MODE_VAL	ECAN_RECEIVE_ALL_VALID
ECAN_B1_AUTORTR_MODE	ECAN_AUTORTR_MODE_DISABLE
...	...
ECAN_B5_TXRX_MODE_VAL	ECAN_BUFFER_TX
ECAN_B5_MODE_VAL	ECAN_RECEIVE_ALL_VALID
ECAN_B5_AUTORTR_MODE	ECAN_AUTORTR_MODE_DISABLE
ECAN_RXF0_MODE_VAL	ECAN_RXFn_ENABLE
ECAN_RXF0_MSG_TYPE_VAL	ECAN_MSG_STD
ECAN_RXF0_VAL	0x0L
ECAN_RXF0_BUFFER_VAL	RXB0
ECAN_RXF0_MASK_VAL	ECAN_RXM0
ECAN_RXF1_MODE_VAL	ECAN_RXFn_ENABLE
ECAN_RXF1_MSG_TYPE_VAL	ECAN_MSG_STD
ECAN_RXF1_VAL	0x0L
ECAN_RXF1_BUFFER_VAL	RXB0
ECAN_RXF1_MASK_VAL	ECAN_RXM0
ECAN_RXF2_MODE_VAL	ECAN_RXFn_ENABLE
ECAN_RXF2_MSG_TYPE_VAL	ECAN_MSG_STD
ECAN_RXF2_VAL	0x0L
ECAN_RXF2_BUFFER_VAL	RXB1
ECAN_RXF2_MASK_VAL	ECAN_RXM1
...	...
ECAN_RXF5_MODE_VAL	ECAN_RXFn_ENABLE
ECAN_RXF5_MSG_TYPE_VAL	ECAN_MSG_STD
ECAN_RXF5_VAL	0x0L
ECAN_RXF5_BUFFER_VAL	RXB1
ECAN_RXF5_MASK_VAL	ECAN_RXM1
ECAN_RXF6_MODE_VAL	ECAN_RXFn_ENABLE
ECAN_RXF6_MSG_TYPE_VAL	ECAN_MSG_STD
ECAN_RXF6_VAL	0x0L
ECAN_RXF6_BUFFER_VAL	RXB0
ECAN_RXF6_MASK_VAL	ECAN_RXM0
...	...
ECAN_RXF15_MODE_VAL	ECAN_RXFn_ENABLE
ECAN_RXF15_MSG_TYPE_VAL	ECAN_MSG_STD
ECAN_RXF15_VAL	0x0L
ECAN_RXF15_BUFFER_VAL	RXB0
ECAN_RXF15_MASK_VAL	ECAN_RXM0
ECAN_RXM0_MSG_TYPE	ECAN_MSG_STD
ECAN_RXM0_VAL	0x0L
ECAN_RXM1_MSG_TYPE	ECAN_MSG_STD
ECAN_RXM1_VAL	0x0L

(*ECAN_INIT_NORMAL*). Considerando che il bit time nominale è pari a 25 TQ (il massimo, secondo lo standard CAN) e che utilizziamo una frequenza di oscillazione pari a 20 MHz il bit rate raggiungibile è pari a 80.000 bit/s cioè 80 kbps. La velocità in questione è certamente sufficiente, almeno se si considera che i nostri pacchetti avranno una parte dati lunga appena 2 byte, quelli relativi al valore della temperatura rilevato. Osservando gli altri parametri vi accorgerete che vengono disabilitate tutte le funzioni di filtro dei messaggi (filtri e maschere sono azzerati) precisando che verranno accettati tutti quelli considerati validi. Pur essendo una semplificazione, questo punto ci permetterà di apprezzare la differenza di funzionamento nel caso venga attivato uno dei filtri disponibili sul modulo. Nel nostro primo esempio diamo la massima libertà nel colloquio tra i due nodi.

Modifiche implementative

La libreria che utilizziamo nel nostro corso è una versione modificata di quella generalmente reperibile presso il sito della Microchip. Le differenze sono in parte formali ed in parte sostanziali. Nel primo caso sono state principalmente eliminate definizioni e strutture relative all'utilizzo di altri compilatori; nell'altro abbiamo eliminato alcune istruzioni specifiche della classe >

```
//      ECANCON_MDSEL1 = ECAN_FUNC_MODE_VAL >> 7;
//      ECANCON_MDSEL0 = ECAN_FUNC_MODE_VAL >> 6;
```

superiore di PIC (64, 68, 80 pin) per cui la libreria è stata inizialmente sviluppata. Le modifiche sono soltanto due, ma è bene considerarle nel caso qualcuno voglia realizzare un progetto su questo gruppo di microcontrollori. Le due modifiche sono entrambe relative alla funzione *ECANInitialize* utilizzata per la configurazione e l'avvio del modulo.

Queste due istruzioni fanno riferimento a due bit del registro *ECANCON*, che è implementato nei PIC 18F6585, 18F8585, 18F6680, 18F8680, ma non nel PIC18F458. La struttura del registro è quella visibile in Fig. 5.

I due bit in questione permettono di stabilire la modalità di funzionamento del modulo CAN. Per il PIC18F458 si utilizza esclusivamente il *MODO0*, che corrisponde a MDSEL0=0 e MDSEL1=0.

Chiaramente, la presenza di queste due istruzioni comporta un errore in compilazione, in quanto il simbolo *ECANCON* risulta sconosciuto. Ecco perché nella nostra versione le due istruzioni sono state eliminate.

Analogamente è stato cancellato qualsiasi riferimento al registro soprastante. La seconda modifica riguarda la valorizzazione di due bit di configurazione relativi all'identificativo di messaggio che il nodo può filtrare (Standard o Extended). I due bit sono utilizzabili solo nei *MODO1* e *2*.

Anche in questo caso ci troviamo di fronte ad una struttura implementata nella classe superiore e non in quella inferiore.

Mettiamo a confronto il registro RXM0SIDL nei due tipi di PIC (Fig. 6 e 7). Come si vede chiaramente, il bit *EXIDEN* non è utilizzato sul PIC18F458, pertanto possiamo eliminare il codice corrispondente mantenendo soltanto la parte relativa alla valorizzazione della maschera. Si veda il codice seguente:

```
#if ( ECAN_RXM0_MSG_TYPE == ECAN_MSG_STD )
    _SetStdRXMnValue(0, ECAN_RXM0_VAL);
//    RXM0SIDL_EXIDEN = 0;
#else
    _SetXtdRXMnValue(0, ECAN_RXM0_VAL);
 //   RXM0SIDL_EXIDEN = 1;
#endif
```

Fig. 5

ECANCON: ENHANCED CAN CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
MDSEL1(1, 2)	MDSEL0(1, 2)	FIFOWM	EWIN4	EWIN3	EWIN2	EWIN1	EWIN0
bit 7							bit 0

La libreria ECAN ha una buona versatilità e permette lo sviluppo di codice per entrambe le classi di PIC. Naturalmente è necessario fare un po' di attenzione alle varie funzionalità che vogliamo attivare. Banalmente, sulla famiglia inferiore l'utilizzo del pin TX2 del modulo non è possibile, perchè fisicamente non esiste.

La routine di inizializzazione

La prima funzione della libreria ECAN che andiamo ad analizzare è quella che permette la configurazione del modulo CAN ed il suo avvio.

Fig. 6

RXMnSIDL: RECEIVE ACCEPTANCE MASK n STANDARD IDENTIFIER MASK REGISTERS, LOW BYTE [0 ≤ n ≤ 1]

R/W-x	R/W-x	R/W-x	U-0	R/W-0	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDEN(1)	—	EID17	EID16
bit 7							bit 0

Fig. 7

RXMnSIDL: RECEIVE ACCEPTANCE MASK n STANDARD IDENTIFIER MASK, LOW BYTE REGISTERS

R/W-x	R/W-x	R/W-x	U-0	U-0	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	—	—	EID17	EID16
bit 7							bit 0

Sostanzialmente si tratta di una procedura che pone il modulo CAN nella modalità *Configuration*, poi esegue tutta una serie di inizializzazioni sulla base dei parametri contenuti nel file *ECAN.def* , quindi precisa il bit rate, i buffer di trasmissione e ricezione, eventuali filtri >

Listato 4

```
    ECANSetOperationMode(ECAN_OP_MODE_CONFIG);

    BRGCON1 = ((ECAN_SJW_VAL-1) << 6) | (ECAN_BRP_VAL-1);
    BRGCON2 = (ECAN_PHSEG2_MODE_VAL << 7)       | \
              (ECAN_BUS_SAMPLE_MODE_VAL << 6)   | \
              ((ECAN_PHSEG1_VAL-1) << 3)        | \
              (ECAN_PROPSEG_VAL-1);
    BRGCON3 = (ECAN_WAKEUP_MODE_VAL << 7)       |
              (ECAN_FILTER_MODE_VAL << 6)       |
              (ECAN_PHSEG2_VAL-1);
    CIOCON = ECAN_TX2_SOURCE_VAL << 7            | \
             ECAN_TX2_MODE_VAL << 6           | \
             ECAN_TXDRIVE_MODE_VAL << 5         | \
             ECAN_CAPTURE_MODE_VAL;

 #if ( ECAN_FUNC_MODE_VAL == ECAN_MODE_0 )
    RXB0CON = (ECAN_RXB0_MODE_VAL << 5) | (ECAN_RXB0_DBL_BUFFER_MODE_VAL << 2);
    RXB1CON = ECAN_RXB1_MODE_VAL << 5;
#endif

#if ( (ECAN_RXF0_MODE_VAL == ECAN_RXFn_ENABLE) || (ECAN_FUNC_MODE_VAL == ECAN_MODE_0) )
    // Set Standard or Extended value.
    #if ( ECAN_RXF0_MSG_TYPE_VAL == ECAN_MSG_STD )
        _SetStdRXFnValue(RXF0, ECAN_RXF0_VAL);
    #else
        _SetXtdRXFnValue(RXF0, ECAN_RXF0_VAL);
    #endif
#endif

#if ( (ECAN_RXF1_MODE_VAL == ECAN_RXFn_ENABLE) || (ECAN_FUNC_MODE_VAL == ECAN_MODE_0) )
    #if ( ECAN_RXF1_MSG_TYPE_VAL == ECAN_MSG_STD )
        _SetStdRXFnValue(RXF1, ECAN_RXF1_VAL);
    #else
        _SetXtdRXFnValue(RXF1, ECAN_RXF1_VAL);
    #endif
#endif

.............

#if ( (ECAN_RXF5_MODE_VAL == ECAN_RXFn_ENABLE) || (ECAN_FUNC_MODE_VAL == ECAN_MODE_0) )
    #if ( ECAN_RXF5_MSG_TYPE_VAL == ECAN_MSG_STD )
        _SetStdRXFnValue(RXF5, ECAN_RXF4_VAL);
    #else
        _SetXtdRXFnValue(RXF5, ECAN_RXF5_VAL);
    #endif
#endif

#if ( ECAN_RXM0_MSG_TYPE == ECAN_MSG_STD )

   _SetStdRXMnValue(0, ECAN_RXM0_VAL);
//    RXM0SIDL_EXIDEN = 0;
#else
{
    _SetXtdRXMnValue(0, ECAN_RXM0_VAL);
//    RXM0SIDL_EXIDEN = 1;
}
#endif

#if ( ECAN_RXM1_MSG_TYPE == ECAN_MSG_STD )
    _SetStdRXMnValue(1, ECAN_RXM1_VAL);
//    RXM1SIDL_EXIDEN = 0;
#else
    _SetXtdRXMnValue(1, ECAN_RXM1_VAL);
//    RXM1SIDL_EXIDEN = 1;
#endif

#if ( ECAN_INIT_MODE != ECAN_INIT_CONFIGURATION )
    ECANSetOperationMode(ECAN_INIT_MODE);
#endif
}
```

Mette il modulo in "Configuration Mode".

**Precisa il bit rate sulla base dei parametri contenuti in ECAN.def nonchè il funzionamento della linea TX2.
Nel nostro caso i bit relativi del registro CIOCON sono disabilitati.**

Precisa la modalità di funzionamento dei buffer.

Precisa la modalità di funzionamento dei filtri.

Precisa la modalità di funzionamento delle maschere stabilendo ad esempio se attivare o meno la possibilità di filtrare messaggi con identificatore standard o extended.

Al termine commuta il modulo nella modalità stabilita attraverso il parametro ECAN_INIT_MODE. Nel nostro caso il valore è pari a ECAN_INIT_NORMAL che attiva il nodo sia in trasmissione che ricezione.

Listato 5

```
void ECANSetOperationMode(ECAN_OP_MODE mode)
{
    CANCON &= 0x1F;      // cancella la modalità precedente
    CANCON |= mode;      // valorizza i bit relativi alla nuova modalità

    while( ECANGetOperationMode() != mode ); //  Attende che il modulo cambi di stato
}
```

ecc. Fatte tutte queste cose, il modulo viene posto nella modalità di funzionamento stabilita nel parametro *ECAN_INIT_MODE* (vedere le ultime righe del Listato 4).
La funzione in questione non ha parametri in ingresso, né in uscita; qui di seguito trovate la sua dichiarazione (è contenuta nel file *ECAN.def*):

void ECANInitialize(void);

Se andiamo a scorrere il listato corrispondente, riusciremo a identificare facilmente le diverse fasi che contraddistinguono questa procedura; riferitevi al Listato 4, nel quale avrete senz'altro notato l'utilizzo di una sotto-procedura chiamata *ECANSetOperationMode*.
Il codice di quest'ultima è molto semplice ed è visibile nel Listato 5. La cosa diventa ancora più chiara se diamo un'occhiata alla struttura del registro *CANCON* (Fig. 8).
Se prestiamo un po' d'attenzione, vediamo come con la prima operazione di AND non si fa altro che azzerare i tre bit più significativi, mantenendo inalterati gli altri cinque.
A seconda del valore assegnato a questi bit, viene cambiata la modalità di funzionamento:

1xx: Configuration Mode
011: Listen Only
010: Loopback
001: Disabilitato
000: Normale

Successivamente, tramite l'OR, il registro viene valorizzato attraverso il parametro *mode* passato alla funzione. Nell'ultimo "while" non si fa altro che verificare che il modulo raggiunga lo stato corrispondente.
La relativa istruzione è un AND logico e viene inserita attraverso una dichiarazione che troviamo nel file *ECAN.h*, come si vede nella riga seguente:

#define ECANGetOperationMode() (CANCON & ECAN_OP_MODE_BITS)

L'inizializzazione del modulo CAN viene fatta semplicemente richiamando la *ECANInitialize*, come vedremo nella prossima puntata, dove parleremo della conclusione di questa prima parte del nostro esperimento. Sempre nella prossima puntata analizzeremo, quindi, l'invio di messaggi standard sul bus CAN e, successivamente, la struttura del nodo ricevente. Potremo, infine, collegare opportunamente i due circuiti, alimentarli e seguire le fasi della comunicazione tra i due nodi CAN direttamente attraverso lo schermo del nostro PC. Appuntamento, dunque, alla prossima puntata del corso CAN.

Fig. 8

CANCON: CAN CONTROL REGISTER

R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
REQOP2	REQOP1	REQOP0	ABAT	WIN2	WIN1	WIN0	—
bit 7							bit 0

Corso di programmazione: CAN BUS

a cura di Carlo Tauraso

Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa quinta puntata approfondiamo il firmware relativo ai nodi analizzando il main program relativo.

5

Nella puntata precedente abbiamo analizzato alcuni aspetti relativi all'inizializzazione del nodo di trasmissione, e le funzioni di contorno (RS232 e 1-Wire), ora non ci resta che descrivere la sequenza di istruzioni inclusa nel main.

Nodo TX: il codice

Se analizziamo il listato possiamo idealmente suddividere l'intero processo in due fasi distinte:

1) Avvio hardware della scheda;
2) Ciclo di trasmissione.

Nella prima parte si eseguono una serie di istruzioni che permettono di inizializzare correttamente i registri che controllano le porte di I/O del PIC. Successivamente viene aperta la porta seriale per l'invio della messaggistica di controllo al PC e si effettua l'avvio della sonda DS18B20 attraverso un impulso di reset. Nel momento in cui viene ricevuta la risposta dalla stessa si inizializza il CAN Bus richiamando la funzione *"ECANInitialize()"*. Il sistema, a questo punto, attende che venga premuto lo switch 2 (quello collegato a RB0). Appena ciò avviene entra nella seconda fase che prevede un ciclo temporizzato. In pratica, si rileva la temperatura ogni secondo finchè non viene premuto lo stesso pulsante d'avvio. I due byte che corrispondono ai due registri *"Temperature MSB/LSB"* della sonda vengono prima registrati in un vettore, per poi essere trasferiti al nodo ricevente attraverso un messaggio CAN standard.
Utilizziamo la funzione *"ECANSendMessage()"* con la seguente sintassi:
BOOL ECANSendMessage (*unsigned long* **id**, *BYTE* ***data**, *BYTE* **dataLen**, *ECAN_TX_MSG_FLAGS* **msgFlags**) >

I parametri da passare sono:

id: È un valore a 32 bit che corrisponde all'identificativo del messaggio. Ricordiamo che esistono due possibili identificativi: uno standard a 11 bit ed uno esteso a 29 bit. Il valore deve essere allineato a destra, e gli eventuali bit rimanenti devono essere azzerati. Nel nostro caso utilizziamo un identificativo per messaggi standard.

data: È un puntatore ad un vettore di byte (massimo 8) che contiene i dati da inviare. Utilizziamo due byte per contenere il valore a 16 bit trasferito dalla sonda termometrica.

dataLen: Corrisponde al numero di byte da inviare (massimo 8 per messaggio). Nel nostro caso è pari a 2.

msgFlags: È il risultato di un'operazione di OR logico tra un valore riguardante la priorità del messaggio, uno relativo all'identificativo usato ed uno riguardante il tipo di messaggio. Specifichiamo soltanto che il messaggio da inviare ha un identificativo di tipo standard.

rità si stabilisce l'ordine con cui i buffer di trasmissione vengono scaricati. La funzione una volta eseguita risponde con un valore booleano che è impostato a *"true"* se il messaggio è stato inserito correttamente in un buffer libero per la trasmissione. Nel caso tutti i buffer siano pieni viene ritornato un valore *"false"*. Nel codice utilizziamo una while per controllare che il messaggio venga effettivamente trasmesso. In pratica l'istruzione viene eseguita finché non c'è un buffer libero. Ad ogni invio si effettua un'apposita segnalazione attraverso il led rosso e si verifica se lo switch 2 risulta premuto oppure no.

Il sistema continua, quindi a campionare i valori di temperatura e ad inviarli sul bus finchè non si preme a lungo lo switch che insiste sulla linea RB0. Tra un campionamento e l'altro si utilizza una funzione di ritardo facente parte della libreria standard del C18, *"Delay10KTCYx()"*. Al termine viene acceso il led verde per segnalare all'utente che l'elaborazione è finita. Vediamo nel concreto il codice corrispondente (Listato 1).

Tabella 1

Valore	Descrizione
ECAN_TX_PRIORITY_0	Il messaggio viene inviato con priorità 0 che corrisponde al valore minimo.
ECAN_TX_PRIORITY_1	Il messaggio viene inviato con priorità 1
ECAN_TX_PRIORITY_2	Il messaggio viene inviato con priorità 2
ECAN_TX_PRIORITY_3	Il messaggio viene inviato con priorità 3 che corrisponde al valore massimo

I valori utilizzabili sono riassunti nelle seguenti tabelle (vedi Tabella 1, Tabella 2, Tabella 3).

Si osservi che i diversi valori di priorità non fanno altro che valorizzare i due bit meno significativi del registro TXBnCON con n il numero del buffer relativo. Pertanto imponendo una prio-

ECANSendMessage: il dietro le quinte

Che cosa accade realmente quando richiamiamo la funzione *"ECANSendMessage()"*? Se facciamo una ricerca nella libreria ECAN.c troveremo una sequenza interessante.

Per rendere le cose più chiare consideriamo sol-

Tabella 2

Valore	Descrizione
ECAN_TX_STD_FRAME	Il messaggio viene inviato con identificativo standard a 11 bit
ECAN_TX_XTD_FRAME	Il messaggio viene inviato con identificativo esteso a 29 bit

Tabella 3

Valore	Descrizione
ECAN_TX_NO_RTR_FRAME	Il messaggio inviato è normale
ECAN_TX_RTR_FRAME	Il messaggio inviato è di tipo RTR (Remote Transmission Request)

Listato 1

```
void main(void)
{
    BYTE data[2]; //Vettore contenente dati da inviare al nodo RX
    BYTE dataLen; //Nr di byte da inviare al nodo RX
    BYTE CONTAG; //Contatore Generico
    BOOL fine;  //Determina la fine del ciclo di trasmissione

    ADCON1=0x07;
    ADCON0=0x00;
    CMCON=0x07;
    TRISA = 0b00000000;
    TRISB = 0b00101011;
    TRISC = 0b10000000;
    TRISD = 0b00001000;
    TRISE = 0b00000000;
    PORTC_RC0=0;
    PORTC_RC1=1;
    PORTC_RC2=0;
    OpenUSART(USART_TX_INT_OFF&USART_RX_INT_OFF&USART_ASYNCH_MODE&USART_EIGHT_BIT&
    USART_CONT_RX&USART_BRGH_HIGH, 64);
    putrsUSART("Avvio NODO CAN \n\r");
    ............
    if (OWReset())
    putrsUSART("DS18B20 OK \n\r");
    else
    putrsUSART("DS18B20 NO-OK \n\r");

    ECANInitialize();
    putrsUSART("CAN OK \n\r");

    putrsUSART("Premere SW2 per avviare trasmissione \n\r");
    while (PORTBbits.RB0 == 1);
    PORTC_RC1=0;
    fine=FALSE;
    while (!fine)
    {
     OWReset();
     OWTX(0xCC);
     OWTX(0x44);
     while (OWRX1());
     OWReset();
     OWTX(0xCC);
     OWTX(0xBE);
     data[1] = OWRX();
     data[0] = OWRX();
     for (dataLen=1;dataLen<=7;dataLen++)
     CONTAG=OWRX();
     while(!ECANSendMessage(0x123, data, 2, ECAN_TX_STD_FRAME));

     Delay10KTCYx(5000);
     PORTC_RC2 == ~PORTC_RC2;

     if (PORTBbits.RB0 == 0)
     fine = TRUE;
     }
     PORTC_RC2=0;
     PORTC_RC1=1;
    while(1);
}
```

Nella fase di inizializzazione hardware vengono disabilitati i moduli A/D, i comparatori e si stabiliscono la direzione (input/output) dei vari pin che compongono le porte del PIC.

La porta seriale viene configurata a 19.200 bps (8,N,1) ed inizia la visualizzazione di messaggistica sullo stato della scheda.

Viene inviato il segnale di reset alla sonda e si verifica il suo corretto funzionamento. Lo stato risultante dell'operazione viene comunicato tramite la seriale.

Viene inizializzato il CAN bus.

Attesa della prima pressione del tasto SW2.

Dopo aver rilevato la temperatura, si richiede alla sonda l'invio dei valori che vengono salvati direttamente nel vettore usato per la trasmissione del messaggio.Si ricordi che la sonda a fronte della richiesta da parte del PIC risponde con 9 byte pertanto gli ultimi 7 vengono scartati.

Il PIC invia il messaggio standard includendo il valore a 16 bit ricevuto dalla DS18B20. L'istruzione viene ripetuta finchè non si trova un buffer libero per completare l'invio correttamente.

Ad ogni invio si inserisce un ritardo e viene fatto lampeggiare il led rosso.

Alla seconda pressione del tasto che insiste sul pin RB0 del PIC viene valorizzata la variabile booleana che porta all'uscita dal ciclo while.

tanto il caso in cui il modulo CAN funzioni nel modo 0 cioè quello standard. Se facciamo attenzione vediamo che dapprima viene creato un vettore di puntatori per i registri del PIC usati come buffer di trasmissione TXB0CON, TXB1CON, TXB2CON.

Successivamente vengono controllati in sequenza i TXREQ di ciascun buffer per verificare se è vuoto e quindi pronto a trasmettere. Nel momento in cui viene trovato un TXREQ a 0 vengono elaborati i flag del messaggio. In pratica si valorizzano i bit TXPRI1:TXPR0 del TXBnCON corrispondente per la priorità, il bit TXRTR del TXBnDLC per la tipologia di messaggio, e i bit DLC3:DLC0 del TXBnDLC per il numero di byte da inviare.

A seconda che i flag prevedano un identificatore standard o esteso viene richiamata una particolare funzione chiamata _CANIDToRegs. Quest'ultima non fa altro che valorizzare correttamente i registri TXBnSIDH, TXBnSIDL, TXBnEIDH, TXBnEIDL a seconda del valore a >

11bit/29bit che viene passato. Infine, attraverso un ciclo for si valorizza il vettore dati TXBnDm corrispondente con i valori del vettore trasmesso come parametro alla funzione ECANSendMessage.
Il buffer, a questo punto è pronto per l'invio dei dati. Il bit TXREQ viene messo a 1 affinché il modulo CAN prenda in carico la trasmissione.
Nel caso in cui il ciclo di controllo dei buffer termini senza che ne sia stato trovato uno libero si invia in risposta un valore *"False"*.
Il codice relativo è illustrato nel Listato 2.

Listato 2

```
BYTE i,j;
BYTE *ptr, *tempPtr;
BYTE* pb[9];
BYTE temp;
  # define  buffers  2;

pb[0]=(BYTE*)&TXB0CON;
pb[1]=(BYTE*)&TXB1CON;
pb[2]=(BYTE*)&TXB2CON;
for ( i = 0; i < buffers; i++ )
{
     ptr = pb[i];
     tempPtr = ptr;
     if ( !(*ptr & 0x08) )
     {
          *ptr &= ~ECAN_TX_PRIORITY_BITS;
          *ptr |= msgFlags & ECAN_TX_PRIORITY_BITS;

           if ( msgFlags & ECAN_TX_RTR_BIT )
               temp = 0x40 | dataLen;
          else
               temp = dataLen;

         *(ptr+5) = temp;

          if ( msgFlags & ECAN_TX_FRAME_BIT )
               temp = ECAN_MSG_XTD;

           else
               temp = ECAN_MSG_STD;

           _CANIDToRegs((BYTE*)(ptr+1), id, temp);

           ptr += 6;

           for ( j = 0 ; j < dataLen; j++ )
               *ptr++ = *data++;

            if ( !(*tempPtr & 0x04) )
               *tempPtr |= 0x08;

           return TRUE;
     }
}
return FALSE;
}
```

Creazione del vettore relativo ai registri usati come buffer di trasmissione.

Verifica se il buffer è libero attraverso il TXREQ di TXBnCON (è il bit3 ecco perchè si fa l'AND con il valore 0x08). Se il bit è uguale a 1 passa al prossimo buffer. Si noti che per migliorare l'efficienza del codice si utilizzano dei puntatori locali (ptr) anzichè l'indice del vettore.

Stabilisce la priorità del messaggio sulla base del parametro msgFlags passato alla funzione.

Stabilisce il numero di byte da inviare e la tipologia di messaggio. In pratica determina il valore del registro TXBnDLC.

Stabilisce se l'identificatore del messaggio è di tipo standard o esteso.

La funzione richiamata valorizza i registri TXBnSIDH, TXBnSIDL, TXBnEIDH, TXBnEIDL sulla base dell'identificatore definito precedentemente.

Carica i dati da inviare nei registri TXBnDm.

Il flag TXREQ viene messo a 1.

In questo caso nessun buffer risulta libero.

Nodo RX

Il nodo in ricezione è un clone del circuito precedente con un firmware modificato per acquisire i dati provenienti dalla sonda e trasferirli sequenzialmente nella EEPROM. Anche in questo caso possiamo idealmente suddividere la sequenza in due fasi:
1) Avvio hardware della scheda;
2) Ciclo di ricezione.
Durante l'avvio il sistema configura i registri TRIS per le linee di input/output del PIC ed inizializza la EEPROM. Per l'accesso a questa memoria (24LC256) abbiamo utilizzato una libreria chiamata XEEPROM.c che implementa la scrittura e la lettura sequenziale del supporto sfruttando un indirizzamento a 16 bit. Per l'avvio utilizziamo una prima funzione chiamata *"XEEInit()"* la cui sintassi è la seguente:
void **XEEInit** (unsigned char *baud*);
ed i parametri assumono i seguenti:
baud: definisce la velocità di comunicazione

con il chip. La libreria utilizza il modulo MSSP del PIC in modalità master I²C e questo valore viene utilizzato per inizializzare il registro SSPADD. Quest'ultimo stabilisce la frequenza di clock utilizzata dal pin SCL (RC3) secondo la formula OSC/4 (SSPADD+1). In particolare i 7 bit meno significativi di questo registro contengono il valore che viene caricato nel BRG (Baud Rate Generator). Quest'ultimo è un contatore che si decrementa due volte per ogni ciclo di clock del sistema (TCY) e viene utilizzato per sincronizzare la linea SCL. Naturalmente, viene ricaricato in maniera automatica proprio con il valore conservato nel registro SSPADD. Per maggiori chiarimenti date un'occhiata ai datasheet del PIC18F458 nella sezione MSSP.

Richiamiamo tale funzione utilizzando una macro per il calcolo del baudrate sulla base della frequenza dell'oscillatore usato per generare il clock di sistema. L'istruzione è:

XEEInit (**EE_BAUD(CLOCK_FREQ, 400000)**);

Se andiamo a verificare che cosa fa la macro (CLOCK=20.000.000) troviamo il codice seguente:

#define EE_BAUD(CLOCK, BAUD) (((CLOCK / BAUD) / 4) - 1)

Una volta avviata la comunicazione con il chip sarà possibile iniziare una sessione di scrittura sequenziale attraverso la funzione *"XEEBeginWrite()"* la cui sintassi è:

void XEEBeginWrite(unsigned char *control*, XEE_ADDR *address*)

control: è il byte di controllo che nel nostro caso viene precisato attraverso una definizione introdotta nel file XEEPROM.h (#define EEPROM_CONTROL (0xa0)). Per il chip 24LC256 la sequenza è composta da 4bit iniziali fissi (1010b), da 3 bit che permettono di selezionare il chip (000b perchè i pin A0,A1,A2 sono collegati a GND), un bit finale a 0. La sequenza binaria corrispondente è 10100000b. I 3 bit messi a 0 servono per selezionare il dispositivo nel caso si utilizzino più chip collegati sullo stesso bus I2C. In tal caso è necessario differenziare la sequenza A0,A1,A2 attraverso dei pull-up.

address: è l'indirizzo della cella iniziale per l'operazione di scrittura. Nel nostro caso il tipo XEE_ADRR corrisponde ad un *"unsigned short int"* quindi un valore a 16bit.

La sessione di scrittura viene terminata richiamando un'ultima funzione *"XEEEndWrite()"* che provvede ad inviare il segnale di stop alla memoria finalizzando l'operazione.
La sintassi è:

XEE_RESULT XEEEndWrite (*void*);

Il parametro *"XEE_RESULT"* passato in uscita corrisponde alla struttura visibile nel Listato 3 e permette di stabilire se l'operazione è andata a buon fine oppure no:

Listato 3

```
typedef enum _XEE_RESULT
{
    XEE_SUCCESS = 0,
    XEE_READY = 0,
    XEE_BUS_COLLISION,
    XEE_NAK,
    XEE_VERIFY_ERR,
    XEE_BUSY
} XEE_RESULT;
```

Dopo questa parentesi sulla gestione della EEPROM continuiamo l'analisi del firmware inserito nel nodo di ricezione.
Il programma principale dopo aver configurato le linee di ingresso e di uscita, inizializzato il bus I²C per la comunicazione con la EEPROM, e avviato il bus CAN attraverso la funzione *"ECANInitialize()"* entra nella seconda fase: il ciclo di ricezione. Si tratta di un while infinito nel quale viene richiamata dapprima la funzione *"ECANReceiveMessage()"*.
Quest'ultima è la funzione complementare della *"ECANSendMessage()"* e permette di elaborare i messaggi che arrivano al modulo CAN del nostro PIC.
Vediamo la sua sintassi nel dettaglio:

BOOL **ECANReceiveMessage**(unsigned long **id*, BYTE **data*, BYTE **dataLen*, ECAN_RX_MSG_FLAGS **msgFlags*)

id: puntatore ad una locazione di memoria a 32bit che conterrà l'identificativo a 11 o 29 bit del messaggio ricevuto.
data: puntatore ad un buffer che conterrà i dati ricevuti.
dataLen: puntatore ad una locazione che con- >

terrà il numero di byte da ricevere.

msgFlags: puntatore ad una locazione che conterrà le caratteristiche del messaggio ricevuto. Anche in questo caso si tratta di un valore che nasce da un'operazione di OR logico tra più flag i cui valori possibili sono specificati nella Tabella 4.

La funzione ritorna un valore booleano a *"False"* se non è stato ricevuto alcun messaggio e a *"True"* in caso contrario. Si faccia attenzione che tutti i parametri passati sono dei puntatori che devono essere inizializzati correttamente.

Come per il nodo di trasmissione abbiamo inserito la funzione in un ciclo while per verificare la presenza o meno di messaggi in arrivo. In pratica il PIC continua ad eseguire la serie di istruzioni finchè un messaggio non risulterà disponibile in uno dei buffer di ricezione.

A questo punto il sistema non fa altro che scrivere ciascuno dei byte ricevuti in locazioni successive della EEPROM attraverso delle *"XEEWrite()"*.

Si faccia attenzione che l'operazione di scrittura viene avviata durante l'inizializzazione con una *"XEEBeginWrite()"*, pertanto, è sufficiente richiamare le funzioni di scrittura una dopo l'altra inserendo dei ritardi di stabilizzazione (obbligatori per questo tipo di memorie). Dopo ogni ricezione i due byte vengono trasferiti scaricando il buffer relativo. Nel momento in cui viene premuto il pulsante SW2 viene arrestata la ricezione finalizzando l'operazione di scrittura sulla EEPROM. Dopo l'accensione del led e dopo aver tolto l'alimentazione, sarà possibile, estrarre la

Tabella 4

Valore	Descrizione
ECAN_RX_OVERFLOW	Il buffer di ricezione è andato in overflow.
ECAN_RX_INVALID_MSG	È stato ricevuto un messaggio non valido.
ECAN_RX_XTD_FRAME	È stato ricevuto un messaggio con identificatore esteso.
ECAN_RX_STD_FRAME	È stato ricevuto un messaggio con identificatore standard.
ECAN_RX_DBL_BUFFERED	Il messaggio è di tipo *"double-buffered"*.

Listato 4

```
void main(void)
{
    unsigned long id;
    BYTE data[2]; //Vettore con i dati ricevuti
    BYTE dataLen; //Variabile con il numero di byte ricevuti
    ECAN_RX_MSG_FLAGS flags; //Flag per la tipologia di msg ricevuto

    ADCON1=0x07;
    ADCON0=0x00;
    CMCON=0x07;
    TRISA = 0b00000000;
    TRISB = 0b00001011;
    TRISC = 0b10000000;
    TRISD = 0b00001000;
    TRISE = 0b00000000;
    PORTC_RC0 = 0;
    PORTC_RC1 = 0;
    PORTC_RC2 = 0;

    XEEInit(EE_BAUD(CLOCK_FREQ, 400000));
    ECANInitialize(); //Inizializza CAN Bus
    PORTC_RC1=1;

    XEEBeginWrite(EEPROM_CONTROL, 0x00);
    while (PORTBbits.RB0==1)
    {
    while( !ECANReceiveMessage(&id, data, &dataLen, &flags) );
    PORTC_RC1=0;
    XEEWrite(data[0]);
    Delay10KTCYx(50);
    XEEWrite(data[1]);
    Delay10KTCYx(50);

    PORTC_RC2 == ~PORTC_RC2;
    }
    XEEEndWrite();
    PORTC_RC2=0;
    PORTC_RC1=0;
    while(1);
}
```

Inizializzazione pin di I/O e spegnimento led di segnalazione.

Avvio del bus I²C e del bus CAN.

Inizio della sessione di scrittura sulla EEPROM a partire dall'indirizzo 0.

Il PIC attende la ricezione del messaggio tramite CAN bus.

Dopo la ricezione dei due byte avviene la scrittura in locazioni successive della 24LC256 con delle pause di stabilizzazione di alcuni millisecondi.

Lampeggio del led rosso.

Finalizzazione della scrittura su EEPROM.

EEPROM e leggere i dati che sono stati registrati. Il codice relativo è il descritto nel Listato 4.

ECANReceiveMessage: il dietro le quinte

Anche in questo caso se andiamo a vedere che cosa accade nel momento in cui richiamiamo la funzione *"ECANReceiveMessage()"* troviamo delle istruzioni interessanti.
Innanzitutto vengono controllati sequenzialmente i vari RXBnCON per verificare se qualcuno di essi ha il bit RXFUL (bit7) valorizzato. Ciò significa che il buffer relativo ha ricevuto un messaggio. In tal caso viene dapprima azzerato il flag d'interrupt riguardante l'avvenuta ricezione. In questo modo viene riattivato il segnale di interrupt per ulteriori arrivi.
Poi si verifica il COMSTAT (Communication Status Register) sul bit di overflow (RXBnOVFL) registrandolo nei *"msgFlags"* ed azzerandolo.
Ricordiamo che utilizziamo la lettera *"n"* per indicare l'ennesimo buffer, quindi RXB0OVFL è il bit riguardante il buffer 0 cioè RXB0CON. Per ciascun buffer viene salvato il puntatore relativo in una variabile temporanea. Quindi si effettua un salto d'esecuzione all'etichetta *"_SaveMessage"*.
Si tratta di una piccola procedura che non fa nient'altro che estrarre i byte ricevuti valorizzando i parametri utilizzati richiamando la funzione *"ECANReceiveMessage()"*.
Si verifica la tipologia di messaggio per i *msgFlags*, si estraggono i byte per il vettore *data* aggiornando il contatore *dataLen*. Al termine il buffer relativo viene rilasciato affinché sia pronto per ricevere un nuovo messaggio.
Abbiamo volutamente tralasciato la procedura di gestione dei filtri sui frame in arrivo perchè al momento non li utilizziamo (sarà argomento del prossimo esperimento).
Qui di seguito vediamo il codice risultante opportunamente commentato (Listato 5).

Collegamento e messa in funzione

Per eseguire il primo esperimento dobbiamo inserire il firmware CANTX.hex e CANRX.hex rispettivamente nel nodo trasmittente ed in quel- >

Listato 5

```
{
    BYTE *ptr, *savedPtr;
    char i;
    BYTE_VAL temp;

#if ( (ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME) || \
      (ECAN_LIB_MODE_VAL == ECAN_LIB_FIXED) && (ECAN_FUNC_MODE_VAL == ECAN_MODE_0) )
    {
        if ( RXB0CON_RXFUL )
        {
            PIR3_RXB0IF = 0;

            if ( COMSTAT_RXB0OVFL )
            {
                *msgFlags |= ECAN_RX_OVERFLOW;
                COMSTAT_RXB0OVFL = 0;
            }
            ptr = (BYTE*)&RXB0CON;
        }
        ...........
--------->       Il codice soprastante viene ripetuto anche per RXB1CON
        ...........
        else
            return FALSE;
        goto _SaveMessage;
    }
#endif

_SaveMessage:
    savedPtr = ptr;
    *msgFlags = 0;

     temp.Val = *(ptr+5);
    *dataLen = temp.Val & 0b00001111;

    if ( temp.bits.b6 )
        *msgFlags |= ECAN_RX_RTR_FRAME;
    temp.Val = *(ptr+2);
    if ( temp.bits.b3 )
    {
```

Il flag RXFUL a 1 indica che è stato ricevuto un messaggio, pertanto si azzera il bit di interrupt relativo.

Nel caso si sia verificato un overflow lo si registra nei msgFlags e si azzera il bit relativo.

Nel caso nessuno dei buffer di ricezione abbia un messaggio da elaborare la funzione ritorna un valore booleano *"False"* altrimenti salta all'etichetta *"_SaveMessage"*.

Carica il numero di byte del messaggio ricevuto.

Carica i *msgFlags* che stabiliscono la tipologia di messaggio: RTR, esteso, standard.

(Continuazione del Listato 5)

```
        *msgFlags |= ECAN_RX_XTD_FRAME;
        temp.Val = ECAN_MSG_XTD;
    }
    else
        temp.Val = ECAN_MSG_STD;

    _RegsToCANID(ptr+1, id, temp.Val);

    ptr += 6;
    temp.Val = *dataLen;
    for ( i = 0; i < temp.Val; i++ )
        *data++ = *ptr++;

    if ( PIR3_IRXIF )
    {
        *msgFlags |= ECAN_RX_INVALID_MSG;
        PIR3_IRXIF = 0;
    }

   *savedPtr &= 0x7f;

    return TRUE;
}
```

La funzione *"_RegsToCANID"* è complementare a quella trovata nella ECANSendMessage e quindi estrae l'ID del messaggio dai relativi registri.

Carica la sequenza di byte dati ricevuti.

Nel caso si riceva un messaggio non valido l'evento viene registrato nei *msgFlags* e si azzera il relativo bit di interrupt IRXIF. PIR3 (Peripheral Interrupt Request Register 3).

Si segnala la disponibilità del buffer per una nuova lettura azzerando il bit RXFUL.

lo ricevente. Questi ultimi possono essere collegati attraverso un cavetto tripolare inserendo direttamente sui connettori a 9 pin le resistenze da 120 ohm che fungono da terminatori. Lo schema è il visibile in Fig. 1.
Colleghiamo quindi i terminali del cavetto sulla porta CAN1 di ciascuna scheda.
È possibile realizzare anche un cavetto senza terminatori ed aggiungere questi ultimi sulle porte CAN2 attraverso dei connettori DB9 femmina. Le porte CAN1 e CAN2 sono, infatti, collegate in parallelo. Connettiamo quindi la porta seriale della scheda di trasmissione alla RS232 del nostro PC.
Avviamo una sessione HyperTerminal direttamente sulla COM utilizzata configurandola a 19200 bps 8N1. A questo punto alimentiamo il nodo TX.
La finestra dell'HyperTerminal apparirà come in Fig. 2.
Noterete l'accensione del led verde che segnala l'attesa di un input da parte dell'utente. Alimentiamo quindi il nodo di ricezione. Anche su quest'ultima scheda si accenderà il led verde. Siamo pronti ad avviare il campionamento. Premiamo e rilasciamo rapidamente il pulsante SW2 del nodo TX. Vedremo che su entrambe le schede comincerà a lampeggiare il led rosso. Il nodo di trasmissione effettua il campionamento dei dati relativi alla temperatura e li invia attra-

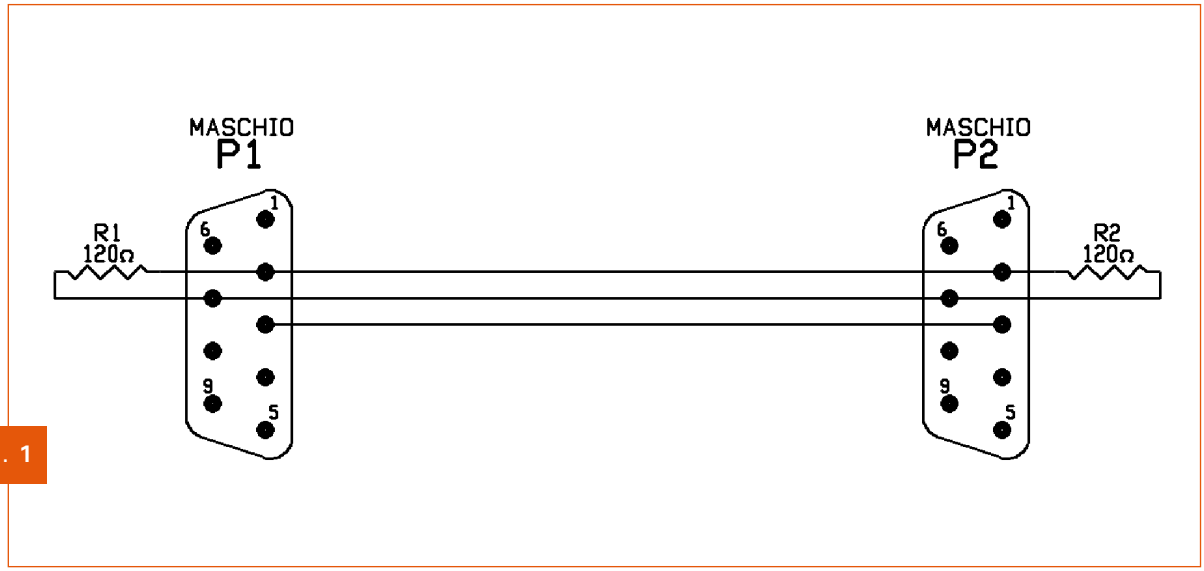


Fig. 1

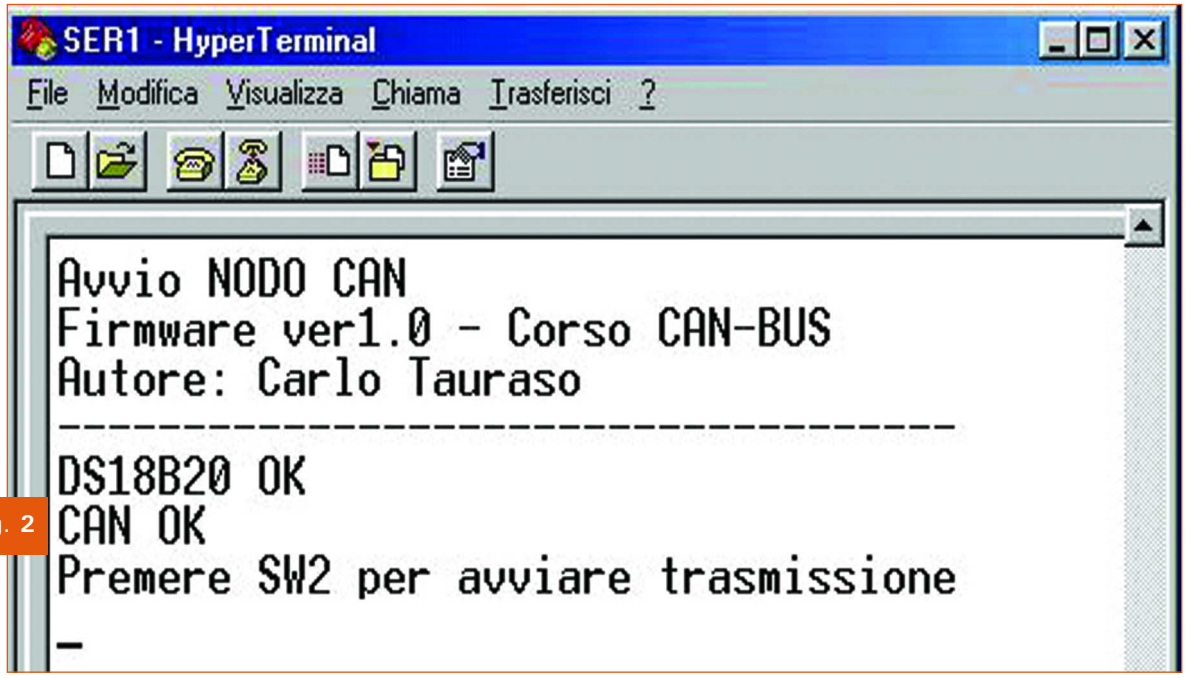


Fig. 2

verso il bus CAN al nodo di ricezione. Qui i due byte ricevuti vengono trasferiti sequenzialmente nella EEPROM. Ad ogni invio sul terminale viene visualizzato un messaggio del tipo TX MSG. Attendiamo qualche minuto per collezionare un pò di dati. Premiamo il pulsante SW2 sul nodo di ricezione. Si accende il led verde. A questo punto possiamo tenere premuto lo stesso pulsante SW2 sul nodo TX finchè non si accende anche lì il led verde. Stacchiamo l'alimentazione su entrambe le schede ed estraiamo la EEPROM dal nodo RX. Se la leggiamo attraverso IC-PROG vedremo nel primo settore la sequenza corrispondente ai valori a 16 bit scaricati dalla sonda. Lo si vede chiaramente nell'immagine di Fig. 3.

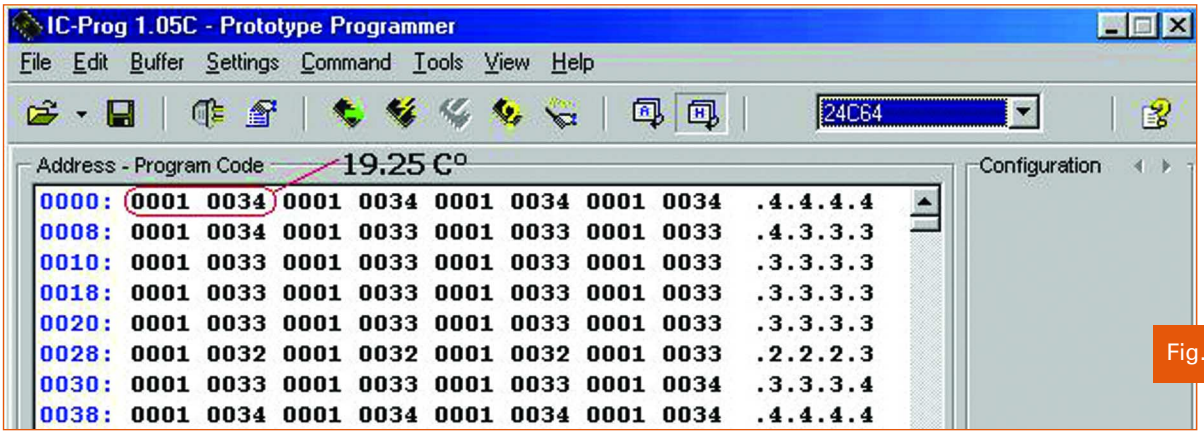


Fig. 3

Siamo giunti al termine del primo esperimento sul CAN bus. Abbiamo visto le configurazioni e le funzioni di base che ci permettono di sfruttare il CAN per il trasferimento e la ricezione dei dati. Nella prossima puntata renderemo la cosa un pò più complessa introducendo il concetto di filtro di messaggi. Assegneremo quindi una funzione specifica al nodo in ricezione facendo sì che esso elabori solo un tipo di messaggi scartando gli altri. Si tratta di un'argomento fondamentale perchè permette di associare più significati ai dati trasferiti indirizzando ciascuna informazione soltanto a quei nodi in grado di elaborarla. Sicuri di aver solleticato la vostra curiosità vi diamo appuntamento alla prossima puntata nella quale descriveremo anche la costruzione della demo-board utilizzata durante i nostri esperimenti (il cui schema è stato proposto nella terza puntata). Ricordiamo anche che i listati completi dei firmware cui facciamo riferimento sono scaricabili gratuitamente dal nostro sito Internet (www.elettronicain.it).

Corso di programmazione: CAN BUS

a cura di Carlo Tauraso

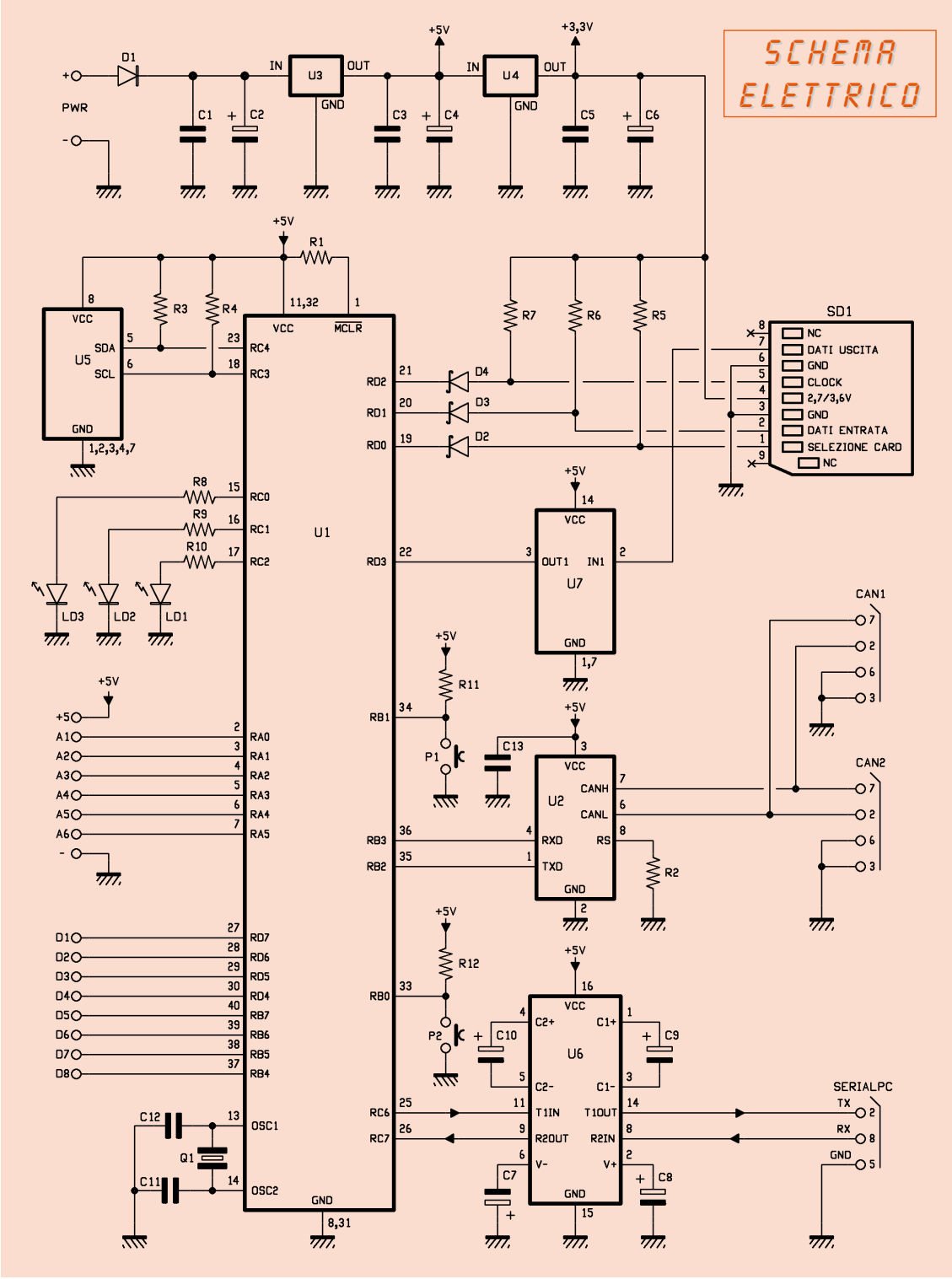
Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa sesta puntata, oltre a rendere disponibili tutti i dettagli della demoboard utilizzata nel Corso, vediamo come filtrare i messaggi in arrivo su un nodo.

6

Nella seconda puntata di questo Corso abbiamo presentato e descritto brevemente lo schema elettrico della demoboard utilizzata per i nostri esperimenti. Più di un lettore ci ha nel frattempo contattati chiedendo se era possibile presentare anche lo schema pratico di tale circuito. In questa sesta puntata, prima di riprendere il filo del discorso occupandoci della possibilità di filtrare i messaggi in arrivo sul nodo CAN, riproponiamo tutti i dettagli costruttivi della demoboard, compreso il circuito stampato. Come si vede nello schema, il dispositivo utilizza un microcontrollore Microchip PIC18F458 (U1) ed un transceiver MCP2551 (U2) per l'interfacciamento fisico. In pratica abbiamo realizzato un singolo nodo il cui schema potrà essere duplicato per realizzare delle reti con funzioni sempre più complesse. Il sistema è dotato di diversi altri componenti di contorno:

1) Slot per SD Card: utilizzato per la memorizzazione di dati derivanti da campionamenti o elaborazioni, viene sfruttato in modalità SPI. E' stato necessario realizzare la conversione dei livelli di tensione utilizzati nella comunicazione con le SD che notoriamente funzionano in un range che va da 2,7V a 3,6V. Abbiamo utilizzato quindi un regolatore di tensione a 3,3V (LM1086 - U4) per l'alimentazione, mentre per i diversi livelli 0-5V e 0-3V abbiamo utilizzato una serie di diodi schotty con resistenze di pull-up sulle linee che vanno dal micro alla SD ed un integrato 74HCT125 per l'uscita da SD a micro. Nel primo caso la linea viene mantenuta ad una tensione di circa 3,3V, non appena sul pin del micro viene presentato un valore logico alto il diodo è interdetto e la tensione sul pin della card sarà la tensione di pull-up. Quando, invece, viene presentato un valore logico basso, il diodo si porta >

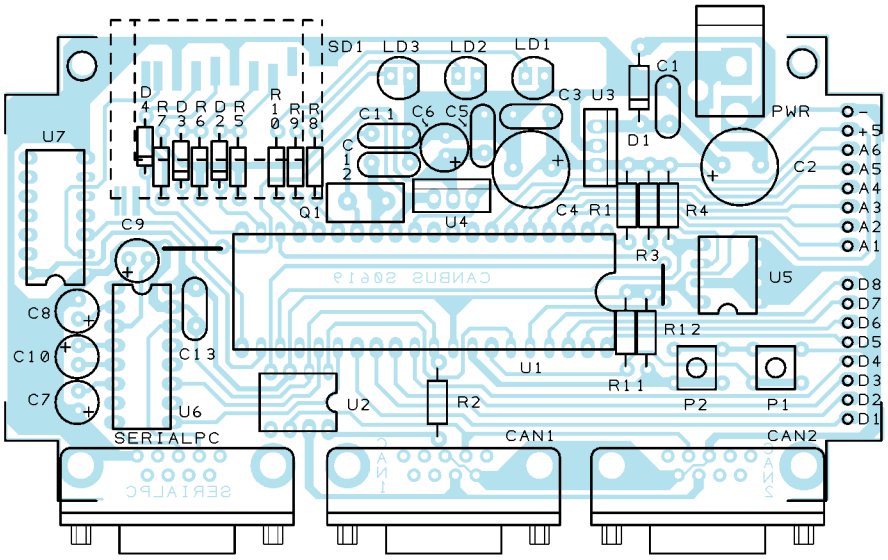
La **demoboard: schema elettrico**



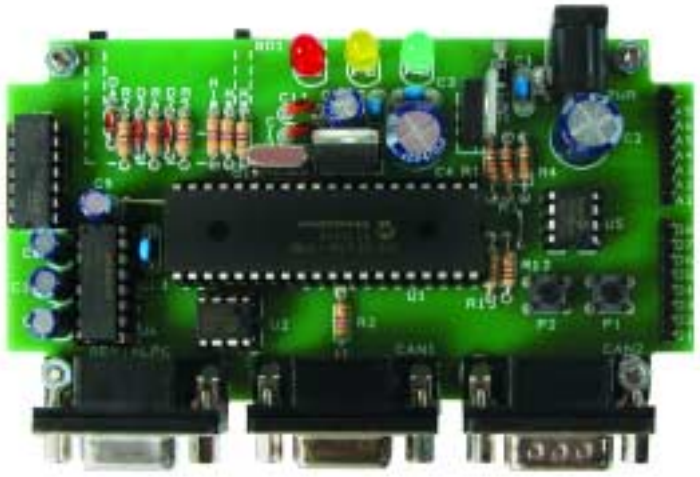
in conduzione collegando a massa anche il pin della card. Per quanto riguarda la connessione di direzione inversa, cioè tra card e PIC, la cosa è differente. Gli integrati basati su logica ACT/HCT accettano livelli TTL in input e presentano in uscita livelli CMOS. In particolare, quando sono alimentati a 5 volt "vedono" un segnale a 3 volt come un normale TTL a 5 volt e forniscono in uscita un segnale a 5 volt che va benissimo per comandare una linea di input del PIC senza necessità di pull-up.

Ecco, quindi, che il 74HCT125 diventa un per-

PIANO DI MONTAGGIO



ELENCO COMPONENTI:



R1, R3, R4: 10 kohm
R2: 4,7 kohm
R5÷R7: 4,7 kohm
R8÷R10: 470 ohm
R11, R12: 10 kohm
C1: 100 nF multistrato
C2: 470 µF 25 VL elettrolitico
C3: 100 nF multistrato
C4: 470 µF 25 VL elettrolitico
C5: 100 nF multistrato
C6: 220 µF 16 VL elettrolitico
C7÷C10: 1 µF 100 VL elettrolitico
C11, C12: 10 pF ceramico
C13: 100 nF multistrato
U1: PIC18F458 (MF619)
U2: MCP2551
U3: 7805
U4: LM1086-3.3
U5: 24LC256
U6: MAX232
U7: 74HCT125
LD1: led 5 mm verde
LD2: led 5 mm giallo
LD3: led 5 mm rosso

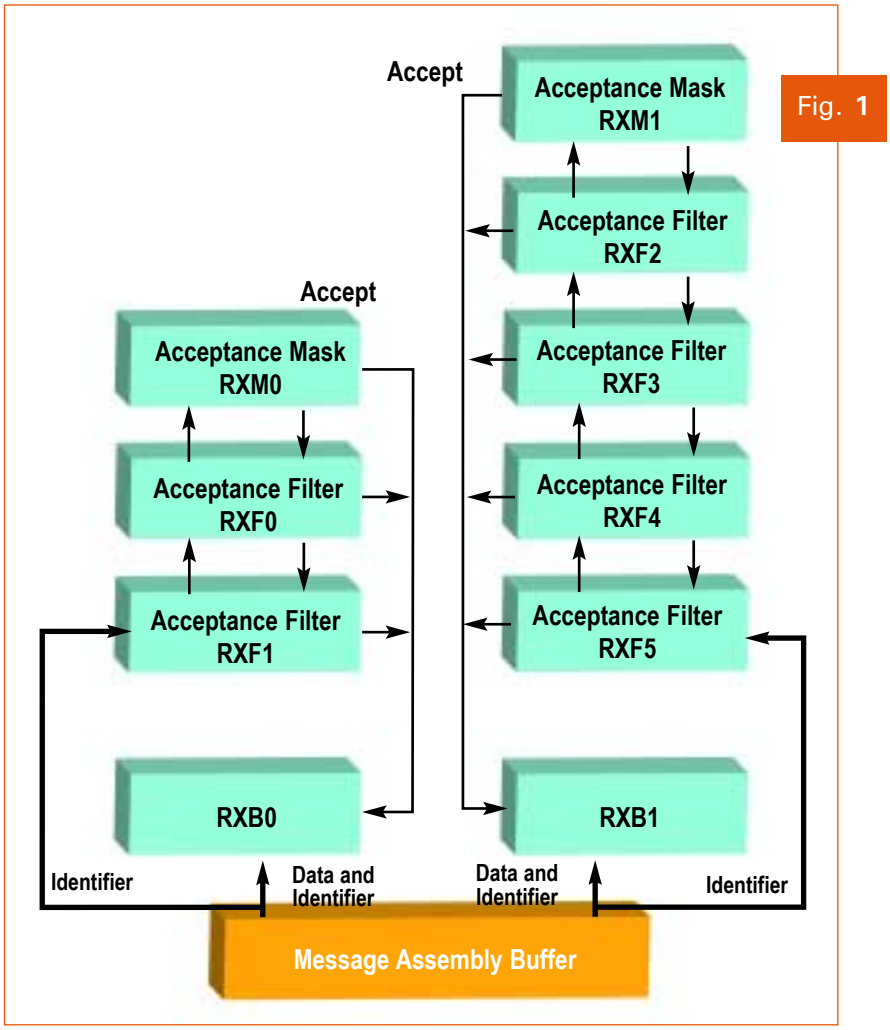
Q1: quarzo 20 MHz
D1: 1N4007
D2÷D4: BAT85
SD1: connettore SD-CARD da CS
P1, P2: microswitch
SERIALPC: Connettore DB9 femmina
CAN1: Connettore DB9 femmina
CAN2: Connettore DB9 maschio

Varie:
- Plug alimentazione
- Zoccolo 4+4 (2 pz.)
- Zoccolo 7+7
- Zoccolo 8+8
- Zoccolo 20+20 passo doppio
- Strip maschio 8 pin (2 pz.)
- Circuito stampato codice S0619

fetto traduttore CMOS -> TTL.
2) EEPROM 24LC64: per la memorizzazione dei dati di configurazione o come buffer temporaneo.
3) Serie di Led Luminosi: per segnalare i vari stati di funzionamento del dispositivo.
4) Porta RS232: per permettere una connessione diretta PC-dispositivo, sia per sperimentare l'invio di comandi che per ricevere messaggistica di controllo sul funzionamento.
I diversi livelli 0-5V e -12v/+12V vengono tradotti dal sempre utile MAX232 (U6) nella sua >

configurazione di base.
5) Due pulsanti: per simulare comandi logici attivati direttamente sul dispositivo CAN.
6) Strip di I/O digitali: sono linee del PIC che una volta configurate possono essere utilizzate in ingresso o in uscita per svolgere vari tipi di funzione a seconda dell'esperimento da svolgere.
7) Strip di ingressi analogici: sono linee del PIC che fanno capo ad un modulo A/D. Possono venir usate per il campionamento di segnali analogici e sono dotate di poli d'alimentazione a 5V.

Fig. 1



Il transceiver MCP2551 viene utilizzato come circuito di traduzione tra i segnali TTL presenti sui pin del microcontrollore e quelli che fanno funzionare il Bus CAN. In particolare questo chip ha diverse modalità di funzionamento, quella scelta da noi è denominata "SLOPE-CONTROL". Infatti, connettendo il pin RS attraverso una resistenza a massa si fa sì che si riducano i tempi di "rise" e "fall" dei segnali sui pin CANH e CANL, riducendo la possibilità di generare interferenze elettromagnetiche. Il chip è pienamente compatibile con le specifiche dettate dallo standard ISO-11898, anzi in certi casi le supera ampiamente come nel caso dei transienti che riesce a sopportare arrivando a circa 250V. La doppia porta sulla scheda ci permette di connettere altri nodi sulla rete creando una sorta di catena agli estremi della quale potremo inserire dei terminatori. Naturalmente è possibile duplicare il circuito anche senza riproporre tutti i componenti di contorno ma soltanto quelli che serviranno per le funzionalità del nodo che stiamo aggiungendo. Il transceiver è in grado di operare ad un transfer rate di 1 Mbps e supporta fino a 112 nodi connessi sul medesimo bus (con resistenza interna differenziale minima di 20 kohm e terminatore con resistenza nominale di

Tabella 1

Maschera (bit n)	Filtro (bit n)	Identificatore (bit n)	Risultato
0	0	0	Accetta
0	1	0	Accetta
0	0	1	Accetta
0	1	1	Accetta
1	0	0	Accetta
1	0	1	Rifiuta
1	1	0	Rifiuta
1	1	1	Accetta

120 ohm). Torniamo ora all'argomento di questa puntata.

Filtrare i messaggi

Introduciamo il secondo esperimento del nostro corso considerando una feature interessante del modulo CAN implementato sui PIC: la possibilità di filtrare i messaggi in arrivo sul nodo. Questa funzionalità è importante perchè si può affidare ad un nodo un ruolo ben preciso all'interno della rete facendo sì che elabori esclusivamente un certo tipo di messaggi scartando gli altri. In questo modo si può comprendere come sia possibile realizzare una serie di scambi di informazioni con significati decisamente diversi utilizzando lo stesso supporto fisico e senza paura che ciò comporti un problema per il sistema nella sua completezza. Per capire come avviene tutto ciò dobbiamo prendere confidenza con due termini fondamentali: maschere e filtri. Un PIC 18FXX8 contiene 2 maschere (una per ciascun buffer di ricezione) e complessivamente 6 filtri. Osserviamo lo schema di Figura 1. Come si vede maschere e filtri vengono associati ai due buffer di ricezione dividendoli in due gruppi. Il primo è costituito da due filtri (RXF0, RXF1) e una maschera (RXM0) ed è collegato al buffer ad alta priorità RXB0. Mentre il secondo è costituito da quattro filtri (RXF2, RXF3, RXF4, RXF5) e una maschera (RXM1) ed è collegato al buffer a bassa priorità RXB1. Maschere e filtri vengono utilizzati per stabilire se un messaggio presente nel MAB (Message Assembly Buffer) debba essere caricato nel buffer relativo oppure no. Nel momento in cui viene ricevuto un messaggio valido l'identificatore viene confrontato con i valori presenti nei filtri. Se si verifica la corrispondenza con uno di essi il messaggio viene trasferito nel buffer altrimenti no. La maschera stabilisce, quali bit dell'identificatore devono essere utilizzati per il confronto.

Tabella 2.1

FILHIT0	Descrizione
1	Accettazione dovuta al filtro RXF1
0	Accettazione dovuta al filtro RXF0

Tabella 2.2

FILHIT1	Descrizione
111	Valore Riservato
110	Valore Riservato
101	Accettazione dovuta al filtro RXF5
100	Accettazione dovuta al filtro RXF4
011	Accettazione dovuta al filtro RXF3
010	Accettazione dovuta al filtro RXF2
001	Accettazione dovuta al filtro RXF1
000	Accettazione dovuta al filtro RXF0

Per capire tale processo vediamo la Tabella 1 considerando che il messaggio viene caricato soltanto se il risultato positivo (Accetta) si ripete per tutti gli n bit dell'identificatore.
Nel caso in cui la maschera sia tutta a 0 vengono accettati i messaggi con qualsiasi valore di identificatore. Altrimenti vengono accettati solo quei messaggi che hanno un identificatore in cui i bit corrispondenti agli 1 della maschera hanno un valore binario esattamente uguale ai bit che si trovano nella medesima posizione all'interno del filtro. Ad esempio supponiamo di avere una maschera RXM0 pari a 00001111b. Questo significa che dobbiamo considerare solo i 4 bit meno significativi dell'identificatore. Supponiamo che nei due filtri RXF0 e RXF1 ci siano i valori 00001010b e 00001011b. Se nel MAB arriva un messaggio con identificatore 10100000b esso non verrà accettato mentre se ne arriva uno con identificatore 10101011b verrà trasferito immediatamente nel buffer RXB0 grazie al valore del filtro RXF1. Un altro fattore molto importante da considerare è che è possibile stabilire nel firmware quale è il filtro che ha permesso la ricezione del messaggio. Nei due registri di controllo RXB0CON e RXB1CON ci sono i campi FILHIT0 e FILHIT1 che precisano proprio il filtro che ha determinato l'accettazione dell'identificatore. Per il buffer RXB0 il FILHIT0 corrisponde al bit 0 mentre per l'RXB1 il FILHIT1 corrisponde ai 3 bit meno significativi del registro di controllo. Ecco le tabelle dei possibili valori e dei relativi significati (vedi Tabella 2.1 e Tabella 2.2).
Si faccia attenzione che FILHIT1 potrà assumere i valori 000b e 001b soltanto se nel registro di controllo RXB0CON è stato valorizzato a 1 il flag RXB0DBEN. Quest'ultimo stabilisce che i messaggi arrivati su RXB0 possano venir trasferiti su RXB1 se necessario. Si pensi al caso in cui arrivi un secondo messaggio ad alta priorità prima che il primo sia stato letto. In condizioni normali si genererebbe una condizione di overflow (RXB0OVFL=1) mentre se il flag è stato messo a 1 il secondo messaggio viene trasferito direttamente nel RXB2 scongiurando il sovraccarico. Nel caso in cui più filtri comportino l'ac- >

cettazione del messaggio il campo FILHIT conterrà il valore binario corrispondente al numero più basso del filtro che ha causato la ricezione. Ad esempio se RXF5 e RXF3 causano l'accettazione di un messaggio FILHIT1 conterrà il valore 011b equivalente al numero decimale 3 (RXF3). Ciò è dovuto al fatto che l'identificativo del messaggio viene comparato con i filtri in ordine crescente da RXF0 a RXF5. In questo modo si dà maggiore priorità ai filtri con un numero basso. Il programmatore può sfruttare questo fatto assegnando ai filtri con alta priorità l'identificativo di quei messaggi che dovranno essere processati prima possibile dal nodo che li ha ricevuti. Infine bisogna fare attenzione che maschere e filtri sono accessibili soltanto nella modalità di configurazione, pertanto è bene configurarli all'avvio del nodo.

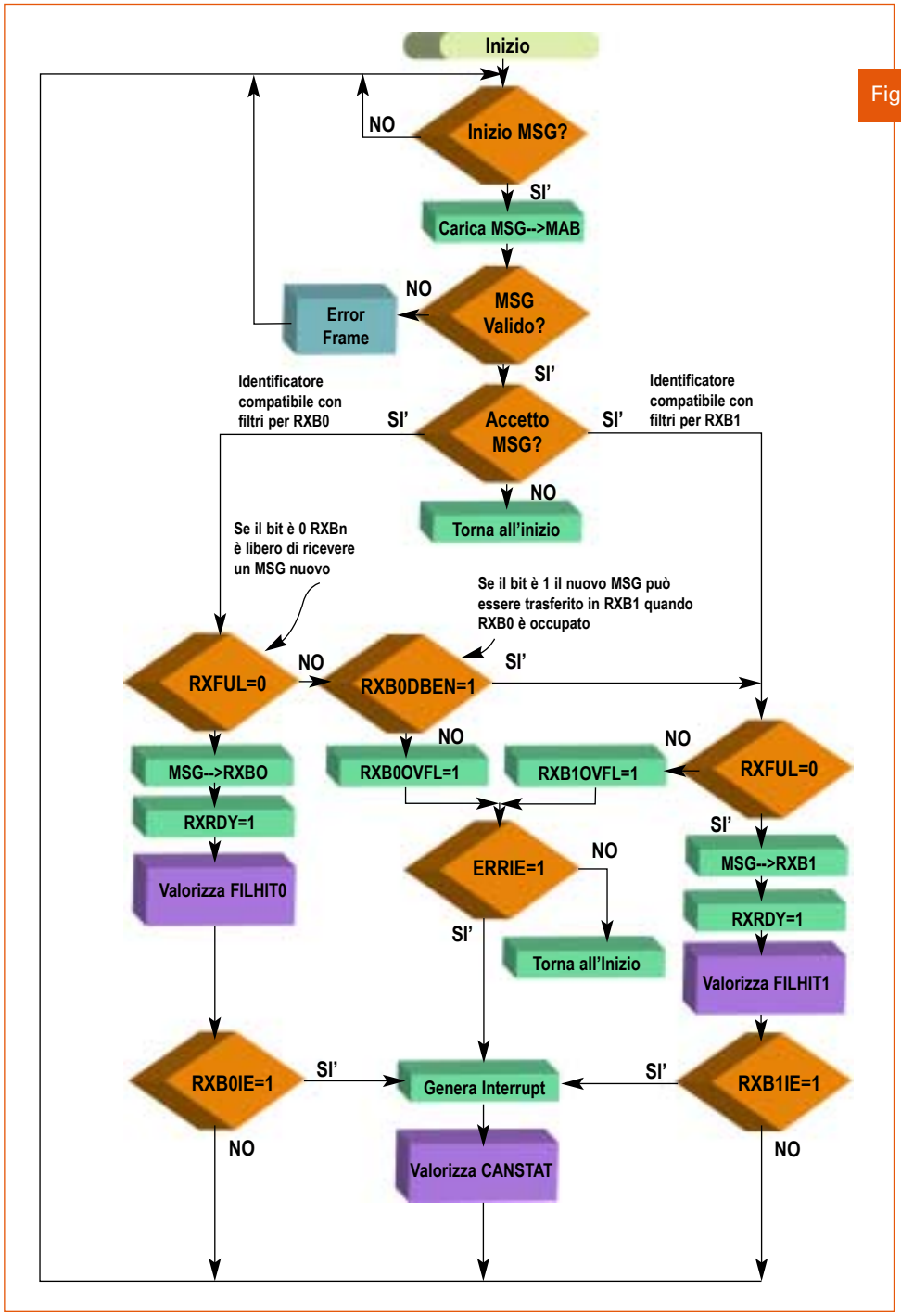


Fig. 2

L'intera procedura di ricezione può essere descritta attraverso il seguente diagramma di flusso. Nel momento in cui viene rilevato l'arrivo di un messaggio esso viene caricato nel MAB (Message Assembly Buffer). Se il messaggio è valido inizia il confronto con i filtri. Se la comparazione va a buon fine si aprono due sequenze differenti a seconda che il filtro sia di competenza del buffer RXB0 o RXB1. Nel primo caso è possibile mettendo a 1 il flag RXB0DBEN far sì che il messaggio venga trasferito al registro RXB1 se l'RXB0 è pieno evitando la generazione di un errore di overflow (RXB0OVFL=1). Se non viene attivato l'RXB0DBEN l'esecuzione continua trasferendo i dati nel buffer relativo. Viene valorizzato il flag RXRDY che segnala la presenza di un messaggio in ricezione. Viene valorizzato il campo FILHITn sulla base del filtro che ha permesso l'accettazione del messag-

Tabella 3

Id Binario	Id Esadecimale	Descrizione
00100100011	123h	Temperatura superiore al MAX
00100100001	121h	Temperatura inferiore al MAX

gio. Infine, se il flag RXB1IE o RXB0IE è a 1 significa che viene generato un segnale di interrupt ogni volta che un messaggio viene trasferito nel registro sul quale il flag è attivato. Dopo la generazione dell'interrupt vengono valorizzati 3 bit meno significativi del registro CANSTAT che permettono di identificare la sorgente di interrupt e quindi il buffer che ha ricevuto il messaggio (vedi Figura 2).

Tabella 4

Nome Campo	Valore	Descrizione
ECAN_RXF0_MODE_VAL	ECAN_RXFn_ENABLE	Permette di stabilire se il filtro RXF0 viene attivato o meno. In realtà utilizzando il "MODE 0" questa definizione è superflua visto che in questo caso risultano già attivati tutti i filtri RXF0..RXF5.
ECAN_RXF0_MSG_TYPE_VAL	ECAN_MSG_STD	Stabilisce la tipologia di identificatore utilizzabile dal filtro RXF0. Nel nostro caso utilizziamo quello standard a 11bit.
ECAN_RXF0_VAL	0x123L	Stabilisce il valore assegnato al filtro. Utilizziamo RXF0 per i segnali d'allarme.
ECAN_RXF0_BUFFER_VAL	RXB0	Stabilisce il collegamento tra i filtri ed uno dei buffer di ricezione. Nel "MODE 0" non possiamo assegnare altri tipi di link quindi usiamo quello standard.
ECAN_RXF0_MASK_VAL	ECAN_RXM0	Anche in questo caso la scelta è obbligata dalla modalità di funzionamento. Qui stabiliamo il link tra il filtro e la maschera di bit relativa.
ECAN_RXF1_MODE_VAL	ECAN_RXFn_ENABLE	Questo valore è equivalente a quello relativo al filtro RXF1 e risulta comunque superfluo nel "MODE 0".
ECAN_RXF1_MSG_TYPE_VAL	ECAN_MSG_STD	Stabilisce la tipologia di identificatore utilizzabile dal filtro RXF1. Nel nostro caso utilizziamo quello standard a 11bit.
ECAN_RXF1_VAL	0x121L	Stabilisce il valore assegnato al filtro. Utilizziamo RXF1 per il campionamento di valori entro il limite massimo assegnato.
ECAN_RXF1_BUFFER_VAL	RXB0	Stabilisce il collegamento tra i filtri ed uno dei buffer di ricezione. Nel "MODE 0" non possiamo assegnare altri tipi di link quindi usiamo quello standard.
ECAN_RXF1_MASK_VAL	ECAN_RXM0	Anche in questo caso la scelta è obbligata dalla modalità di funzionamento. Qui stabiliamo il link tra il filtro e la maschera di bit relativa.
ECAN_RXM0_MSG_TYPE	ECAN_MSG_STD	Stabilisce la tipologia di identificatore utilizzabile dalla maschera RXM0. Nel nostro caso utilizziamo quello standard a 11bit.

Secondo Esperimento: allarme termo

Passiamo allo sviluppo lato firmware di quanto abbiamo descritto nei precedenti paragrafi. Come abbiamo già anticipato in altre puntate di questo corso la libreria che stiamo utilizzando permette di modificare i parametri di funzionamento del modulo CAN al momento della compilazione o a runtime. Per rendere le cose più comprensibili abbiamo deciso di seguire la prima di queste due strade evidenziando i parametri necessari e raggruppandoli in un unico file chiamato ECAN.def. L'insieme di definizioni che lo compongono si possono modificare aprendolo all'interno del MPLAB IDE o ancor più semplicemente ricorrendo all'interfaccia di Microchip Application Maestro. Andiamo, quindi, ad introdurre il nostro secondo esperimento. Vogliamo far in modo che il nodo TX generi due tipologie di messaggi riconosciuti ed opportunamente elaborati dal nodo ricevente. In pratica stabiliremo un livello massimo di temperatura oltre il quale il nodo invierà un messaggio di allarme con un identificatore particolare. Quest'ultimo non appena verrà ricevuto dal nodo RX farà scattare un allarme. Per semplicità accenderemo uno dei led >

ed invieremo una stringa AT attraverso la seriale per effettuare una chiamata telefonica via modem. Si tratta di un progetto puramente didattico che però vuole darvi un esempio di come si possa sfruttare la possibilità di generare diverse tipologie di messaggi. In questo caso specializziamo il ruolo dei due nodi. Il nodo TX funge da sonda di rilevamento mentre quello RX funge da centralina di controllo registrando i dati provenienti dalla sonda e attuando eventuali procedure nel caso tali dati escano da determinati parametri. Si tratta di una metodologia utilizzata diffusamente nei progetti CAN reali, si pensi a quanto avviene nei sistemi montati sulle automobili. Prima di tutto dobbiamo configurare il modulo CAN del nodo RX in maniera che sia in grado di discriminare diverse tipologie di messaggi. Abbiamo deciso di utilizzare la maschera RXM0 configurandola in maniera da considerare esclusivamente gli ultimi 2 bit dell'identificatore standard a 11 bit. Per i messaggi in condizioni di temperatura normali cioè inferiori al limite massimo stabilito utilizzeremo i due bit a 01b mentre per quelli in condizioni di allarme valorizzeremo i due bit a 11b. Riassumiamo i due tipi di identificatore in Tabella 3. Per farlo correttamente andiamo a modificare le definizioni del file ECAN.def riguardo ai campi visualizzati in Tabella 4.
Avrete notato che la gestione di questi dati al momento della compilazione è senz'altro la maniera più semplice di configurare il funzionamento del nostro modulo CAN. In particolare, se fate attenzione le modifiche rispetto al file .def utilizzato nel primo esperimento sono veramente minime. In pratica abbiamo soltanto inserito i valori relativi alla maschera e ai due filtri che ci servono. L'utilizzo della modalità normale (MODE 0) oltre ad essere quella più compatibile è anche quella che prevede l'attivazione automatica dei 6 filtri di cui dispone il modulo CAN evitando la necessità di ulteriori configurazioni. Dopo aver generato il nuovo file ECAN.def andiamo a considerare una nuova funzione della libreria che va ad affiancarsi a quelle che abbiamo utilizzato per inviare e ricevere messaggi dal bus (ECANSendMessage, ECANReceiveMessage).
In Figura 3 vediamo una fase di generazione del file ECAN.def tramite l'interfaccia "Microchip Application Maestro" presentata nelle precedenti puntate.

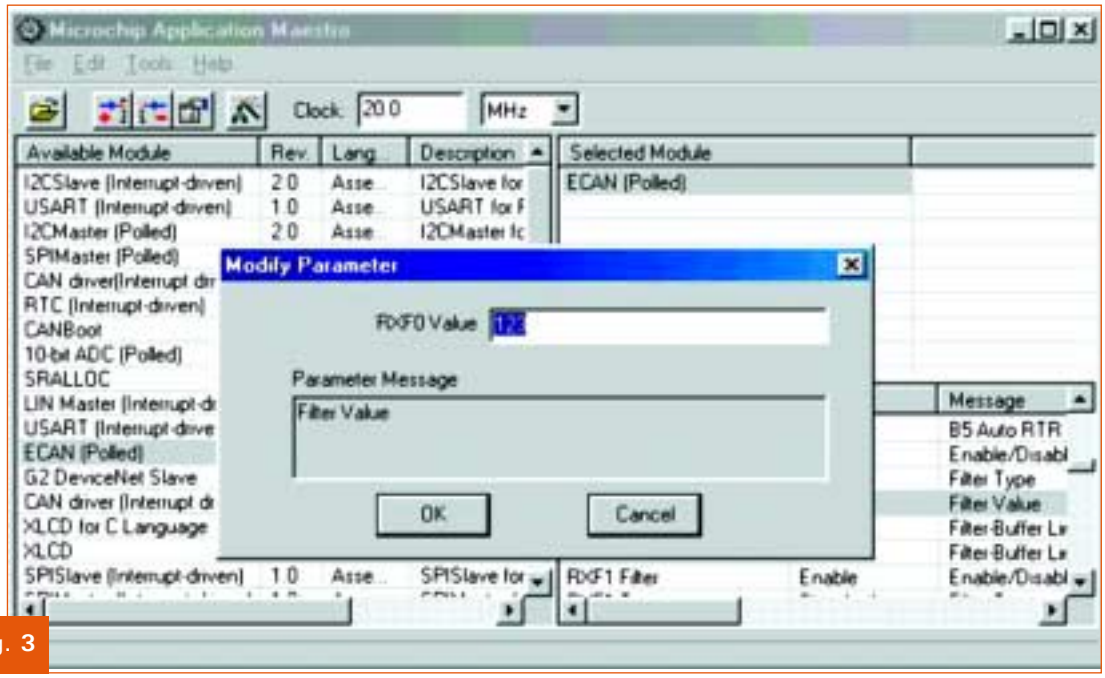


Fig. 3

Identificare il filtro di accettazione

È chiaro che per riuscire a capire se stiamo ricevendo un valore normale o un segnale d'allarme dobbiamo essere in grado di discriminare i due messaggi e sapere quindi quale filtro ha causato la ricezione nel buffer RXB0.
Qualcuno avrà già intuito che è necessario chiamare in causa i campi FILHIT0 e FILHIT1 dei registri di controllo RXB0CON e RXB1CON.
Abbiamo a disposizione un'interessante funzione la cui sintassi è la seguente:

BYTE **ECANGetFilterHitInfo** (void)

Dopo la sua esecuzione riceveremo in uscita il valore corrispondente al filtro che ci interessa. In pratica se il filtro che ha permesso la ricezione è l'RXF0 riceveremo uno 0 se è l'RXF1 riceveremo un 1 e così via. Ricordiamo che nel "MODE 0" sono attivi soltanto i 6 filtri RXF0...RXF5. Ma vediamo che cosa accade quando richiamiamo questa funzione.
Se apriamo il file ECAN.h in MPLAB troviamo una macro che contiene la seguente definizione:

```
#define ECANGetFilterHitInfo()   (_ECANRxFilterHitInfo.Val)
 extern BYTE_VAL _ECANRxFilterHitInfo;
```

La definizione esterna viene ripresa poi nel file ECAN.c attraverso la dichiarazione:

```
BYTE_VAL   _ECANRxFilterHitInfo;
```

Il valore ritornato viene valorizzato in diversi punti della "*ECANReceiveMessage()*". Appena richiamata questa funzione esso viene azzerato, poi nelle diverse fasi di ricezione viene valorizzato sulla base delle sequenze di bit inserite nei campi FILHIT0 e FILHIT1.
Ecco le istruzioni che vengono eseguite nei due diversi casi.

1) MESSAGGIO RICEVUTO IN RXB0

```
_ECANRxFilterHitInfo.bits.b0 = RXB0CON_FILHIT0;
```

FILHIT0 può avere solo due valori 0 o 1, pertanto si valorizza il bit meno significativo del parametro di output.
In questo caso riceveremo il valore 0 se il filtro interessato all'accettazione è RXF0 e 1 se il filtro è RXF1.

2) MESSAGGIO RICEVUTO IN RXB1

```
_ECANRxFilterHitInfo.Val = RXB1CON & 0x07;
```

FILHIT1 si compone di tre bit che corrispondono proprio a quelli meno significativi del registro di controllo RXB1CON. Ecco che per estrarli si effettua l'AND logico con il valore 7 corrispondente proprio alla sequenza binaria 00000111b. Una volta estratti si possono tranquillamente assegnare al parametro di output visto che seguono esattamente la sequenza numerica con cui sono denominati i diversi filtri.
Ora che abbiamo tutti gli strumenti e le conoscenze teoriche necessarie possiamo passare allo sviluppo vero e proprio.

Nodo TX: il firmware

Lo schema circuitale che utilizzeremo è lo stesso visto nel precedente esperimento. Sfrutteremo una sonda DS18B20 collegata sulla strip di cui è dotata la nostra demoboard. In particolare utilizzeremo il pin RB5 come linea dati del bus 1-Wire. Questa volta faremo uso anche della EEPROM del nodo di trasmissione per caricare il valore massimo di temperatura oltre il quale è necessario inviare un messaggio di allarme. Si tratta di un dato che potrebbe essere tranquillamente inserito nel firmware ma ciò non rappresenterebbe un buon esempio di programmazione visto che il sistema non potrebbe essere personalizzato se non ricompilando il listato. Riserviamo, quindi il primo settore per i dati di configurazione. Manterremo l'invio di messaggistica attraverso la porta seriale in maniera da poter monitorare le varie fasi dell'elaborazione. Se viene rilevato un aumento di temperatura tale da superare il valore stabilito il nodo deve inviare il messaggio di allarme. Ciò non comporta il blocco del campionamento pertanto il nodo TX continuerà ad inviare le temperature rilevate e il nodo RX continuerà a registrarle. Vediamo nel concreto il codice utilizzato analizzando le solite due fasi di inizializzazione e di trasmissione. Premesso che il limite massimo di temperatura viene considerato positivo la variabile che lo conterrà è stata dichiarata attraverso una "*union*" che ci permette di valorizzare ed accedere facilmente ai singoli byte che compongono il valore a 16bit. Accanto alle variabili già viste nel firmware del precedente esperimento abbiamo aggiunto un'ulteriore variabile booleana per stabilire lo stato di allarme quindi il fatto che la temperatura ha superato il limite massimo. Analogamente, definiamo un'ulteriore variabile a 16 bit (*comp*) per effettuare il confronto diretto con quella caricata dalla >

Listato 1

```
union
    {
        unsigned short Val;
        struct
        {
            unsigned char LSB;
            unsigned char MSB;
        } bytes;
    } maxtemp;  /*Limite massimo di temperatura*/

void main(void)
{
    BYTE data[2]; //Vettore contenente dati da inviare al nodo RX
    BYTE dataLen; //Nr di byte da inviare al nodo RX
    BYTE CONTAG; //Contatore Generico
    BOOL fine;  //Determina la fine del ciclo di trasmissione
    BOOL allarme;  //Attivazione segnale d'allarme
    unsigned short comp; //Valore di confronto

    ADCON1=0x07;
    ADCON0=0x00;
    CMCON=0x07;
    TRISA = 0b00000000;
    TRISB = 0b00101011;
    TRISC = 0b10000000;
    TRISD = 0b00001000;
    TRISE = 0b00000000;
    PORTC_RC0=0;
    PORTC_RC1=1;
    PORTC_RC2=0;

    OpenUSART(USART_TX_INT_OFF&USART_RX_INT_OFF&USART_ASYNCH_MODE&USART_EIGHT_BIT&USART_CONT_RX&USART_BRGH_HIGH, 64);
    putrsUSART("Avvio NODO CAN \n\r");

    if (OWReset())
    putrsUSART("DS18B20 OK \n\r");
    else
    putrsUSART("DS18B20 NO-OK \n\r");

    XEEInit(EE_BAUD(CLOCK_FREQ, 400000)); //Inizializza EEPROM
    if (XEEBeginRead(EEPROM_CONTROL, 0x00) == XEE_SUCCESS)
    {
    putrsUSART("EEPROM OK \n\r");
    maxtemp.bytes.MSB=XEERead();
    maxtemp.bytes.LSB=XEERead();
    XEEEndRead();
    putrsUSART("TEMP.MAX OK \n\r");
    }
    else
    putrsUSART("EEPROM NO-OK \n\r");

    ECANInitialize();
    putrsUSART("CAN OK \n\r");

    putrsUSART("Premere SW2 per avviare trasmissione \n\r");
```

Dichiarazione del limite massimo di temperatura salvato in EEPROM. La union permette di accedere al valore sia complessivamente che attraverso la divisione nel byte più significativo e meno significativo.

Fase di inizializzazione hardware attraverso la configurazione delle linee di I/O, la disattivazione di moduli non necessari e il reset dei led di segnalazione.

Avvio della porta seriale per la messaggistica di controllo.

Invio del segnale di reset alla sonda. Ciò permette di rilevare la presenza della stessa ed il corretto funzionamento del bus 1-Wire.

Dopo l'inizializzazione della EEPROM viene caricato il valore del limite massimo di temperatura nella variabile maxtemp. Al termine si finalizza l'operazione segnalando lo stato della stessa via seriale.

Avvio del modulo CAN. Si ricordi che è in questa fase che vengono caricati i parametri di configurazione raggruppati nel file ECAN.def

EEPROM. Dopo le dichiarazioni notiamo chiaramente la fase di inizializzazione che configura i pin in ingresso ed uscita del PIC disabilitando i moduli non necessari. Analogamente si avvia la porta seriale per la messaggistica di controllo, il bus 1-Wire per la sonda DS18B20 ed infine la memoria EEPROM. Da quest'ultima carichiamo i due byte corrispondenti al valore del limite massimo di temperatura partendo dalla locazione 0. Si faccia attenzione che il valore salvato mantiene lo stesso formato utilizzato dalla sonda durante il campionamento per facilitarne il confronto. Al termine si ha l'avvio del bus CAN (Listato 1). Il ciclo di trasmissione inizia dopo che l'utente ha premuto il pulsante SW2 (linea RB0). Successivamente viene avviato il campionamento inviando il comando di conversione alla sonda. Si ricordi che la lettura dei registri della DS18B20 riguarda 9 byte di cui ci interessano soltanto i primi due mentre gli altri devono essere scartati. Si carica il valore a 16 bit nei due byte del vettore data per essere pronti all'invio. A questo punto inseriamo una sequenza nuova. Considerando che i primi 5 bit del byte maggiormente significativo trasferito dalla sonda sono riservati al segno effettuiamo una verifica per stabilire se il valore ricevuto è negativo. Abbiamo limitato il caso alla determinazione di

Listato 2

```
    while (PORTBbits.RB0 == 1);
    PORTC_RC1=0;
    fine = FALSE;
    while (fine == FALSE)
    {
     OWReset();
     OWTX(0xCC);
     OWTX(0x44);
     while (OWRX1());
     OWReset();
     OWTX(0xCC);
     OWTX(0xBE);
     data[0] = OWRX();
     data[1] = OWRX();
     for (dataLen=1;dataLen<=7;dataLen++)
     CONTAG=OWRX();
     allarme = FALSE;
     if (data[1] <= 7)
     {
      comp = data[1];
      comp = comp << 8;
      comp = comp + data[0];
      if (comp > maxtemp.Val)
      allarme = TRUE;
      else
      allarme = FALSE;
     }
     if (allarme)
     while(!ECANSendMessage(0x123, data, 2, ECAN_TX_STD_FRAME));
     else
     while(!ECANSendMessage(0x121, data, 2, ECAN_TX_STD_FRAME));

     putrsUSART("TX MSG\n\r");
     Delay10KTCYx(5000); //ritardo tra un rilievo e l'altro
     PORTC_RC2 = ~PORTC_RC2;
     if (PORTBbits.RB0 == 0)
     fine = TRUE;
     }
     PORTC_RC2=0;
     PORTC_RC1=1;
     while(1);
}
```

Non appena il tasto SW2 viene premuto la linea RB0 viene collegata a GND quindi il livello logico diventa basso. Viene spento il led verde e si inizializza la variabile booleana che stabilisce la fine del ciclo di trasmissione.

Viene rilevata la temperatura attraverso la DS18B20. Il segnale di reset è seguito dal comando di conversione. Si attende il termine del processo e poi viene inviato il comando di lettura dei 9 byte relativi ai diversi registri della sonda.

Per nostra definizione il limite massimo di temperatura è superiore a 0. Siccome i 5 bit più significativi del valore trasmesso dalla sonda riguardano il segno effettuiamo le verifiche rispetto al limite soltanto se il byte MSB è inferiore a 8.

Il valore utilizzato per il confronto è caricato trasferendo dapprima gli 8 bit più significativi, effettuando un shift a sinistra e poi sommando il valore con quelli meno significativi. La struttura definita per maxtemp ci permette, invece, di accedere direttamente al valore a 16bit attraverso il campo Val.

Se la temperatura rilevata è superiore al limite massimo viene inviato un messaggio di allarme con identificativo 123h. Nel caso, invece, la variabile allarme sia "FALSE" viene trasmesso un messaggio normale con identificativo 121h.

un limite relativo alla temperatura massima superiore a 0. Nel caso in cui il dato rilevato sia superiore a 7 (ultimi 3 bit tutti a 1) significa che i 5 bit superiori sono valorizzati a 1 pertanto la temperatura è negativa. In tale caso il nodo non invia un messaggio di allarme visto che certamente la temperatura è inferiore al limite. Altrimenti si confrontano i singoli byte campionati. Se anche uno solo di essi è superiore a quello registrato come limite massimo il sistema deve inviare un messaggio di allarme. Le due situazioni relative al superamento o meno del limite comportano la valorizzazione opposta della variabile booleana "*allarme*". Si effettua, quindi, una verifica condizionale su quest'ultima. Se l'espressione risultante è vera si esegue un ciclo while per la trasmissione di un messaggio con identificatore 123h. Altrimenti il valore a 11 bit relativo viene posto pari a 121h. Dopo l'invio viene fatto lampeggiare il led rosso e si segnala l'evento tramite un apposito messaggio trasferito su seriale. Verificato che l'utente non sta tenendo premuto il pulsante SW2 il ciclo ricomincia daccapo. Il codice inserito nel main è visibile nel Listato 2.

Nodo RX: il firmware

Anche per quanto riguarda il nodo di ricezione utilizziamo lo stesso schema circuitale visto nel precedente esperimento. Ci preoccuperemo soltanto di collegare la porta seriale ad un modem analogico o al cellulare attraverso l'apposito cavo dati RS-232. Naturalmente possiamo testare quest'ultima funzionalità anche soltanto collegando la scheda ad un PC e verificando attraverso HyperTerminal il corretto invio della stringa AT. Nel momento in cui viene collegata l'alimentazione si accende il led verde. Il nostro nodo è pronto. Non appena cominciano ad arrivare i messaggi esso non fa altro che filtrarli estraendo di volta in volta il valore di temperatura prove- >

niente dalla sonda (ad ogni ricezione il led rosso lampeggia). Scrive, quindi, tale valore nella EEPROM ed effettua la chiamata telefonica accendendo il led giallo nel caso in cui l'identificatore del messaggio ricevuto sia pari a 123h. Manteniamo per semplicità lo stesso formato dati proveniente dalla sonda. Come abbiamo già fatto per il nodo TX riserviamo il primo settore della EEPROM (512 byte) per i dati di configurazione. Nel nostro caso utilizziamo un comando Hayes "ATDT" (Dial Tone). Tuttavia modificando opportunamente la stringa si può, ad esempio, effettuare l'invio di un messaggio SMS. Inserendo nella EEPROM la sequenza ASCII seguente è possibile definire il numero di cellulare e il testo del messaggio:

AT+CMGS="*<numero-cell>*"*<CR>*<*testo-SMS*><*CTRL+Z*>#

in esadecimale

41-54-2B-43-4D-47-53-3D-22 22-0D 1A
A T + C M G S = " " CR CTRL+Z

Bisogna considerare che gli SMS possono essere gestiti in due possibili formati "PDU MODE" e "TEXT MODE". Per realizzare questo tipo di invio è necessario settare la modalità testo attraverso il comando "AT+CMGF=1". Chi vuole realizzare qualcosa di più complesso può sfruttare la possibilità di inserire più comandi AT+ sulla stessa linea separandoli con un ";". Il firmware effettua l'invio della stringa completa finchè non trova il carattere "*#*".

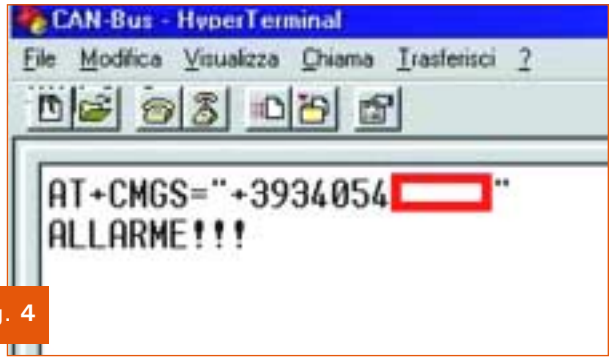


Fig. 4

Dopo questa breve digressione torniamo al nostro progetto. Il firmware può essere suddiviso in due fasi: una di inizializzazione ed una relativa al ciclo di ricezione. La sequenza è simile al precedente esperimento. In questo caso dobbiamo ricordare che durante l'esecuzione della funzione ECANInitialize() vengono caricate le configurazioni descritte nei precedenti paragrafi e salvate nel file ECAN.def. Vengono, quindi, attivati i filtri RXF0 e RXF1 assieme alla maschera RXM0. Discriminiamo i messaggi in arrivo sulla base degli ultimi due bit proprio grazie al valore che abbiamo inserito in RXM0.
Nel ciclo di ricezione registriamo il valore di temperatura in arrivo direttamente nella EEPROM a partire dal secondo settore (ricordate che il primo è utilizzato dai dati di configurazione). È necessario aggiungere la "***ECANGetFilterHitInfo()***" che permette di capire quale dei filtri è stato utilizzato per accettare il messaggio.
Nel caso il valore ritornato da questa funzione sia pari a 0 significa che il nodo ha ricevuto un messaggio con identificatore pari a 123h e quindi la temperatura ha superato il limite massimo previsto.
A questo punto il firmware invia attraverso la seriale il comando AT o AT+. Il nodo RX, nel caso si verifichi una situazione di allarme effettua un'unica volta la trasmissione della stringa e continua a registrare i valori provenienti dal nodo TX. Il sistema viene arrestato premendo il tasto SW2 (quello che insiste sul pin RB0) che comporta l'accensione del led verde e lo spegnimento di quello rosso.
A questo punto è possibile bloccare anche il nodo TX sempre premendo il tasto SW2. La verifica dell'invio della stringa AT può essere fatta collegando la porta RS-232 del nodo RX con il nostro PC (19200,8-N-1). Apriamo una sessione HyperTerminal sulla COM utilizzata per la connessione. Se aumentiamo la temperatura della sonda (basta stringerla tra le dita) avendo avuto l'accortezza di non fissare un limite massimo troppo elevato ad un certo punto si accenderà il led giallo e vedremo comparire sul nostro schermo il comando che abbiamo inserito in EEPROM. Ecco come si presenta la schermata dopo l'esecuzione della sequenza AT+ necessaria all'invio di un SMS con il messaggio di allarme (Figura 4).
Anche per questo numero siamo arrivati al termine della nostra puntata di approfondimento sullo sviluppo CAN.
Nella prossima puntata vedremo nel concreto il codice main utilizzato per il nodo RX commentando le relative istruzioni. Eseguiremo il secondo esperimento osservando i valori registrati in EEPROM e introdurremo la nostra ultima prova pratica di questo Corso.

Corso di programmazione: CAN BUS

a cura di Carlo Tauraso

Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa settima puntata, analizziamo nel concreto il codice main utilizzato per il nodo RX commentando le relative istruzioni.

Nella precedente puntata abbiamo introdotto un nuovo esperimento che prevede la realizzazione di un nodo di ricezione CAN predisposto per effettuare il filtro dei messaggi. In pratica il nodo di trasmissione invia due tipi di messaggi, uno relativo ad un rilievo di temperatura normale ed uno relativo ad un rilievo di temperatura fuori soglia. La ricezione di quest'ultimo messaggio deve generare uno stato di allarme nel nodo di ricezione effettuando l'invio di una stringa Hayes sulla porta seriale del PC.

Listato nodo RX

Tutta la configurazione dei filtri sul nodo di ricezione avviene modificando il file ECAN.def. Ricordiamo che abbiamo utilizzato l'RXF0 per i messaggi con ID=0123h relativi allo stato di allarme. Normalmente, invece, il nodo riceve messaggi con ID=0121h che trasportano valori di temperatura al di sotto della soglia considerata eccessiva. Tali valori vengono registrati all'interno della EEPROM del nodo RX. Il file ECAN.def può essere rigenerato attraverso il tool Microchip Application Maestro. Usando questa interfaccia grafica si minimizza la probabilità di commettere degli errori tentando l'editing diretto del file. Selezionate la libreria ECAN(Polled) e trascinatela nel pannello "Selected Module". Potete modificare tutti i parametri necessari attraverso la lista sottostante facendo doppio clic sulla riga relativa. Al termine, il codice complessivo viene creato facendo clic sul pulsante "Generate Code" (CTRL+G). Il form relativo è visibile in Figura 1. Nel precedente capitolo abbiamo visto che le modifiche rispetto al file originale riguardano esclusivamente la configurazione di 2 filtri e di una maschera associata. Per rendere la cosa più semplice da comprendere abbiamo raccolto nelle >

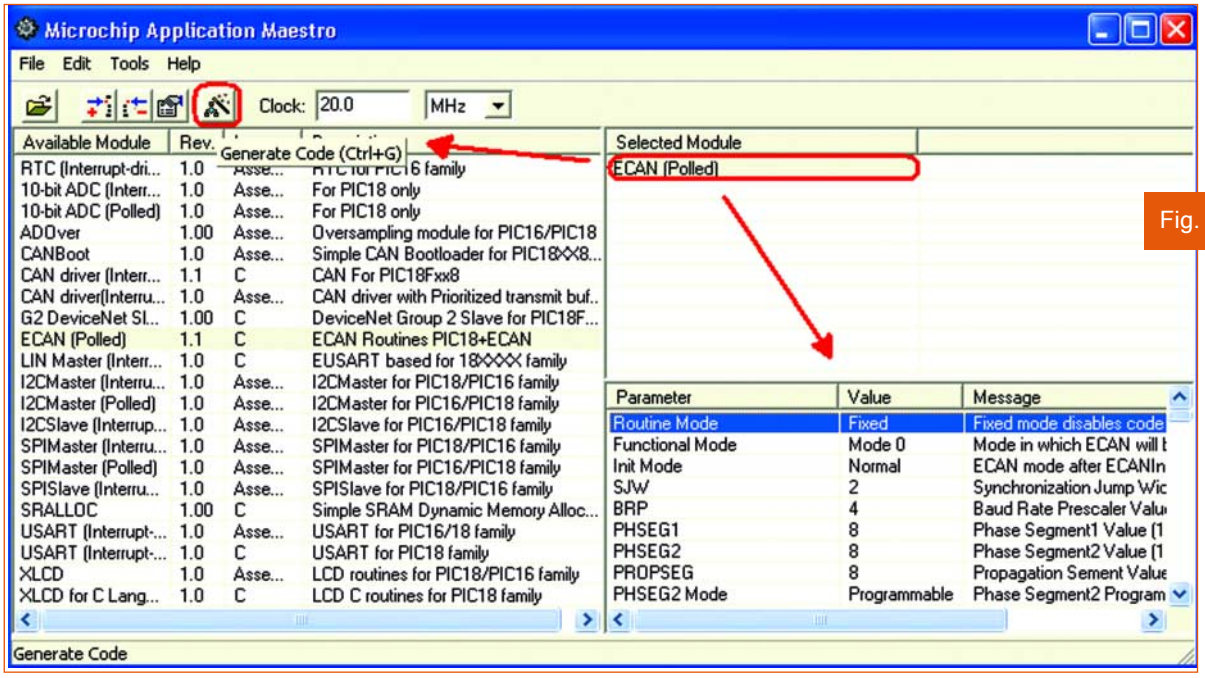


Fig. 1

figure 2 e 3 l'elenco completo dei parametri. Associamo al filtro RXF0 l'identificativo 123 relativo allo stato di allarme. Sul RXF1, invece, inseriamo il valore 120 relativo ad un rilievo di temperatura nella norma.

La discriminazione dei messaggi avviene verificando i due bit meno significativi dell'ID a 11 bit inviato dal nodo TX. Quest'ultima funzionalità viene attivata configurando la maschera associata RXM0(=3=11b).

Parameter	Value	Message
Routine Mode	Fixed	Fixed mode disables code that changes paramters...
Functional Mode	Mode 0	Mode in which ECAN will be initialized with
Init Mode	Normal	ECAN mode after ECANInitialize
SJW	2	Synchronization Jump Width Value (1-4)
BRP	4	Baud Rate Prescaler Value (1-64)
PHSEG1	8	Phase Segment1 Value (1-8)
PHSEG2	8	Phase Segment2 Value (1-8)
PROPSEG	8	Propagation Sement Value (1-8)
PHSEG2 Mode	Programmable	Phase Segment2 Programmable Mode.
Bus Sample Mode	Thrice	CAN Bus sample mode
Wakeup Mode	Enable	CAN Bus activity mode
Bus Filter Mode	Disable	CAN Bus Low pass Filter
CANTX1 Drive Mode	Vdd	State to which CANTX1 will be driven in recessive...
CANTX2 Pin	Digital I/O	CANTX2 Pin Mode
CANTX2 Source	Completemn...	CANTX2 Source
CAN Capture Mode	Disable	CAN Capture Mode
RXB0 Receive Mode	All Valid	RXB0 Receive Mode
RXB0 Double Buffer	Disable	RXB0 Double Buffer Mode
RXB1 Receive Mode	All Valid	RXB1 Receive Mode
B0 TX/RX Mode	Transmit	B0 TX/RX Mode
B0 Receive Mode	All Valid	B0 Receive Mode
B0 Auto RTR	Disable	B0 Auto RTR Mode
B1 TX/RX Mode	Transmit	B1 TX/RX Mode
B1 Receive Mode	All Valid	B1 Receive Mode
B1 Auto RTR	Disable	B1 Auto RTR Mode
B2 TX/RX Mode	Transmit	B2 TX/RX Mode
B2 Receive Mode	All Valid	B2 Receive Mode
B2 Auto RTR	Disable	B2 Auto RTR Mode
B3 TX/RX Mode	Transmit	B3 TX/RX Mode
B3 Receive Mode	All Valid	B3 Receive Mode
B3 Auto RTR	Disable	B3 Auto RTR Mode
B4 TX/RX Mode	Transmit	B4 TX/RX Mode
B4 Receive Mode	All Valid	B4 Receive Mode
B4 Auto RTR	Disable	B4 Auto RTR Mode
B5 TX/RX Mode	Transmit	B5 TX/RX Mode
B5 Receive Mode	All Valid	B5 Receive Mode
B5 Auto RTR	Disable	B5 Auto RTR Mode

Fig. 2

Analizziamo ora la sequenza di istruzioni che abbiamo utilizzato per la ricezione dei messaggi. Il ciclo viene ripetuto fino alla pressione dello switch SW2.

La funzione "ECANReceiveMessage" permette di valorizzare i parametri id, data, datalen, flags attraverso i relativi campi trasmessi nel pacchetto proveniente dal nodo TX. Il valore a 16 bit ricevuto (corrispondente alla temperatura acquisita dal nodo di trasmissione) viene registrata sequenzialmente nella EEPROM 24LC256 di cui è dotata la scheda. Successivamente, dopo aver segnalato l'avvenuta operazione di scrittura attraverso il lampeggio del led rosso, si richiama la funzione "ECANGetFilterHitInfo". Come abbiamo già anticipato, essa permette di stabilire il filtro che ha determinato l'accettazione del messaggio. Il valore intero relativo viene restituito in uscita e quindi verificato attraverso una struttura condizionale. Nel caso in cui esso sia pari a 0 viene eseguita l'istruzione che trasferisce il comando Hayes sulla porta seriale. La situazione di allarme viene segnalata anche attraverso l'accensione del led giallo.

Il pin relativo viene utilizzato anche come flag in maniera da evitare la ripetizione del comando ATDT una volta che il superamento della soglia

Parameter	Value	Message
RXF0 Filter	Enable	Enable/Disable
RXF0 Type	Standard	Filter Type
RXF0 Value	123	Filter Value
RXF0 Buffer	RXB0	Filter-Buffer Link
RXF0 Mask	RXM0	Filter-Buffer Link
RXF1 Filter	Enable	Enable/Disable
RXF1 Type	Standard	Filter Type
RXF1 Value	121	Filter Value
RXF1 Buffer	RXB0	Filter-Buffer Link
RXF1 Mask	RXM0	Filter-Buffer Link
RXF2 Filter	Enable	Enable/Disable
RXF2 Type	Standard	Filter Type
RXF2 Value	0	Filter Value
RXF2 Buffer	RXB1	Filter-Buffer Link
RXF2 Mask	RXM1	Filter-Buffer Link
RXF3 Filter	Enable	Enable/Disable
RXF3 Type	Standard	Filter Type
RXF3 Value	0	Filter Value
RXF3 Buffer	RXB1	Filter-Buffer Link
RXF3 Mask	RXM1	Filter-Buffer Link
RXF4 Filter	Enable	Enable/Disable
RXF4 Type	Standard	Filter Type
RXF4 Value	0	Filter Value
RXF4 Buffer	RXB1	Filter-Buffer Link
RXF4 Mask	RXM1	Filter-Buffer Link
RXF5 Filter	Enable	Enable/Disable
RXF5 Type	Standard	Filter Type
RXF5 Value	0	Filter Value
RXF5 Buffer	RXB1	Filter-Buffer Link
RXF5 Mask	RXM1	Filter-Buffer Link
RXF6 Filter	Disable	Enable/Disable
RXF6 Type	Standard	Filter Type
RXF6 Value	0	Filter Value
RXF6 Buffer	RXB0	Filter-Buffer Link
RXF6 Mask	RXM0	Filter-Buffer Link
RXF7 Filter	Disable	Enable/Disable
RXF7 Type	Standard	Filter Type
RXF7 Value	0	Filter Value
RXF7 Buffer	RXB0	Filter-Buffer Link
RXF7 Mask	RXM0	Filter-Buffer Link
RXF8 Filter	Disable	Enable/Disable
RXF8 Type	Standard	Filter Type
RXF8 Value	0	Filter Value
RXF8 Buffer	RXB0	Filter-Buffer Link
RXF8 Mask	RXM0	Filter-Buffer Link
RXF9 Filter	Disable	Enable/Disable
RXF9 Type	Standard	Filter Type
RXF9 Value	0	Filter Value
RXF9 Buffer	RXB0	Filter-Buffer Link
RXF9 Mask	RXM0	Filter-Buffer Link
RXF10 Filter	Disable	Enable/Disable
RXF10 Type	Standard	Filter Type
RXF10 Value	0	Filter Value
RXF10 Buffer	RXB0	Filter-Buffer Link
RXF10 Mask	RXM0	Filter-Buffer Link
RXF11 Filter	Disable	Enable/Disable
RXF11 Type	Standard	Filter Type
RXF11 Value	0	Filter Value
RXF11 Buffer	RXB0	Filter-Buffer Link
RXF11 Mask	RXM0	Filter-Buffer Link
RXF12 Filter	Disable	Enable/Disable
RXF12 Type	Standard	Filter Type
RXF12 Value	0	Filter Value
RXF12 Buffer	RXB0	Filter-Buffer Link
RXF12 Mask	RXM0	Filter-Buffer Link
RXF13 Filter	Disable	Enable/Disable
RXF13 Type	Standard	Filter Type
RXF13 Value	0	Filter Value
RXF13 Buffer	RXB0	Filter-Buffer Link
RXF13 Mask	RXM0	Filter-Buffer Link
RXF14 Filter	Disable	Enable/Disable
RXF14 Type	Standard	Filter Type
RXF14 Value	0	Filter Value
RXF14 Buffer	RXB0	Filter-Buffer Link
RXF14 Mask	RXM0	Filter-Buffer Link
RXF15 Filter	Disable	Enable/Disable
RXF15 Type	Standard	Filter Type
RXF15 Value	0	Filter Value
RXF15 Buffer	RXB0	Filter-Buffer Link
RXF15 Mask	RXM0	Filter-Buffer Link
RXM0 Type	Standard	Mask Type
RXM0 Value	3	Mask Value
RXM1 Type	Standard	Mask Type
RXM1 Value	0	Mask Value

Fig. 3

è stato rilevato. Nel momento in cui l'allarme rientra, il led viene spento. La registrazione dei valori avviene comunque in maniera da rappresentare fedelmente l'andamento della temperatura in tutte le situazioni. Rispetto al precedente listato abbiamo modificato la scrittura in EEPROM in maniera da finalizzare l'operazione di scrittura sequenziale ad ogni rilievo e non soltanto al termine del ciclo.

In secondo luogo abbiamo inserito durante l'inizializzazione del nodo di trasmissione tre cicli di lettura della sonda in maniera da evitare falsi allarmi. All'avvio, infatti, tali sonde contengono nei due registri riservati alla temperatura un valore predefinito pari a 0550h (corrispondente a ben 85°C), che avrebbe potuto comportare una segnalazione d'allarme inesistente.

Il codice definitivo è visibile nel Listato 1.

È importante tenere presente che la configurazione dei filtri e delle maschere deve avvenire in "Configuration Mode" . Nel momento in cui si entra nel "Normal Mode" i registri relativi non risultano accessibili per eventuali modifiche. La valorizzazione dei vari registri è affidata alla ECANInitialize attraverso 2 definizioni che troviamo all'interno della ECAN.c:

1) #define _SetStdRXFnValue(f, val)
2) #define _SetStdRXMnValue(m, val)

Queste istruzioni permettono di valorizzare rispettivamente i registri RXFnSIDH, RXFnSIDL, RXMnSIDH, RXMnSIDL.

Avevamo già visto come la ECANInitialize richiamasse tali funzioni, ora possiamo entrare nel dettaglio e vedere quali istruzioni vengono effettivamente eseguite. Nel primo caso si ha:

##f##SIDH = (long)ECAN_##f##_VAL >> 3L;
##f##SIDL = (long)ECAN_##f##_VAL << 5L;

Come si vede chiaramente nel momento in cui si richiama la funzione attraverso la:

_SetStdRXFnValue(RXF0, ECAN_RXF0_VAL);

le istruzioni compilate risultanti saranno: >

Listato 1

```
ECANInitialize();
  PORTC_RC1=1;
  IND = 0x00;

  while (PORTBbits.RB0==1)
  {
  while(!ECANReceiveMessage(&id, data, &dataLen, &flags));
  PORTC_RC1=0;

  XEEBeginWrite(EEPROM_CONTROL, IND);
  XEEWrite(data[1]);
  Delay10KTCYx(50);
  XEEWrite(data[0]);
  Delay10KTCYx(50);
  IND = IND + 2;
  XEEEndWrite();

  PORTC_RC2 = ~PORTC_RC2;

  switch (ECANGetFilterHitInfo())
   {
  case 0:
       if (PORTC_RC0 == 0)
       {
       putrsUSART("ATDT 115\n\r");
       PORTC_RC0 = 1; //LED GIALLO
       }
       break;
  case 1:PORTC_RC0 = 0;
       break;
   }

  }

  PORTC_RC2=0;
  PORTC_RC1=1;
```

Inizializzazione CAN bus, accensione led verde e azzeramento dell'indirizzo iniziale per la scrittura in EEPROM.

Finchè SW2 non viene premuto il led verde viene spento e si provvede ad inizializzare i campi id, data, datalen e flags sulla base dei messaggi ricevuti sul CAN bus.

Ciclo di scrittura del valore a 16bit relativo alla temperatura trasmessa dal nodo CAN su EEPROM. L'operazione viene finalizzata ad ogni rilievo. Successivamente si fa lampeggiare il led rosso.

Una volta rilevato l'indice del filtro che ha provocato l'accettazione del messaggio si gestisce l'evento. Se l'indice è pari a 0 l'id ricevuto è pari a 123 quindi si tratta di un segnale d'allarme. Allora viene inviata la stringa AT sulla seriale e si accende il led giallo. Se l'indice è pari a 1 la temperatura è inferiore al livello di soglia quindi si provvede soltanto a spegnere il led (allarme rientrato).

È stato premuto SW2 pertanto il led rosso viene spento e si accende il led verde che segnala il termine della procedura.

RXFnSIDH

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

RXFnSIDL

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	-	EXIDEN	-	EID17	EID16
bit 7							bit 0

Fig. 4

RXF0SIDH = (long)ECAN_RXF0_VAL >> 3L;
RXF0SIDL = (long)ECAN_RXF0_VAL << 5L;

Nella tabella inclusa nel file ECAN.def abbiamo assegnato al parametro ECAN_RXF0_VAL il valore 123. Pertanto nel byte alto andrà tale valore shiftato di 3 bit a destra, mentre in quello basso andrà tale valore shiftato di 5 bit a sinistra. Il perchè lo si vede facilmente dalla struttura dei due registri (vedi Fig. 4). L'identificativo standard (11bit) che dovrà essere verificato è allineato a partire dal bit più significativo del RXFnSIDH. Pertanto, sarà dapprima

necessario estrarre gli 8 bit più significativi di ECAN_RXF0_VAL. Poi, siccome gli ultimi 3 bit del SID sono posizionati ai bit 7, 6, 5 del registro sarà necessario effettuare uno shift a sinistra per assegnarli correttamente a partire dal valore incluso in ECAN_RXF0_VAL.
Nella seconda definizione le istruzioni eseguite sono:

```
RXM##m##SIDH = (long)ECAN_RXM##m##_VAL >> 3L;
RXM##m##SIDL = (long)ECAN_RXM##m##_VAL << 5L;
```

Richiamando la funzione attraverso la:

_SetStdRXMnValue(0, ECAN_RXM0_VAL);

le istruzioni risultanti saranno:

```
RXM0SIDH = (long)ECAN_RXM0_VAL >> 3L;
RXM0SIDL = (long)ECAN_RXM0_VAL << 5L;
```

In questo caso il valore assegnato alla maschera ECAN_RXM0_VAL è pari a 3, pertanto stabiliamo che i due bit meno significativi dell'ID trasmesso permetteranno di discriminare i messaggi accettati dal filtro 0 e dal filtro 1.
Anche qui l'operazione di shift è necessaria a causa della particolare struttura dei due registri come da Figura 5.
Il SID in questo caso deve essere inteso come maschera di bit. Pertanto nella verifica con il valore contenuto nel filtro associato si utilizzano soltanto i bit valorizzati a 1.
Nel nostro caso sono rappresentati dal SID0 e dal SID1.
L'analisi di queste istruzioni di inizializzazione ci permette di mettere in evidenza un piccolo problemino riscontrato nella libreria ECAN distribuita assieme alla Application Note 878.

AN878: qualche grattacapo con i filtri

In un primo momento, durante la stesura di questo corso, abbiamo fatto riferimento principalmente ad una Application Notes disponibile sul sito di Microchip: la AN878. Si tratta di un documento sicuramente interessante per quanti intraprendono lo sviluppo CAN su PIC perchè presenta una libreria completa di funzioni per l'invio e la ricezione di messaggi su bus CAN. Inoltre, permette anche lo sviluppo di funzionalità più complesse (id a 32bit) presenti nei moduli ECAN della famiglia superiore (18F6585, 18F6680, 18F8585, 18F8680). Il sistema funziona correttamente con un utilizzo di base in cui i nodi accettano tutti i messaggi in ingresso. I problemi sono iniziati appena abbiamo iniziato a "fare sul serio" inserendo due filtri su RXF0 e RXF1. Stranamente i messaggi non venivano accettati in nessuno dei due casi e passavano al filtro di livello superiore (RXF2) che li elaborava perchè esso risultava azzerato e quindi non operava nessun tipo di restrizione. Siamo così andati a vedere nel dettaglio cosa stava succedendo e ci siamo accorti di un piccolo errore nella ECANInitialize che comporta una valorizzazione errata dei registri RXFnSIDL, RXMnSIDL. Il tutto nasce banalmente da uno shift sbagliato sul valore del byte meno significativo. Questo comporta l'errata valorizzazione proprio dei bit che nella nostra applicazione risultano i più importanti perchè sono quelli su cui si basa la discriminazione tra i diversi tipi di messaggio. Se andate a vedere l'archivio .zip distribuito con la >

RXMnSIDH

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

RXMnSIDL

R/W-x	R/W-x	R/W-x	U-0	U-0	U-0	R/W-x	R/W-x
SID2	SID1	SID0	-	-	-	EID17	EID16
bit 7							bit 0

Fig. 5

application note vi accorgerete che nella ECAN.c alla definizione _SetStdRXFnValue(f, val) l'istruzione eseguita per il byte meno significativo è:

##f##SIDL = (long)ECAN_##f##_VAL >> 5L

Questo comporta uno shift di 5 bit a destra del valore inserito in ECAN_RXF0_VAL con conseguente disallineamento del valore reale. Sottolineiamo che questo comportamento si riferisce esclusivamente alla modalità fixed dove tutte le configurazioni vengono fatte in compilazione. Nella filosofia di utilizzo spiegata nella AN878, infatti, si precisa che la libreria può funzionare in due modalità: fixed e run-time. Nel primo, cioè quello utilizzato finora, la configurazione di tutto il modulo CAN del PIC viene effettuata all'interno del file ECAN.def. Questo approccio comporta una significativa diminuzione della lunghezza del codice utilizzato. Naturalmente risulta possibile utilizzare il registro CANCON per rientrare in Configuration Mode e correggere il valore assegnato erroneamente ai filtri. Ciò può essere fatto esclusivamente dopo aver ricevuto 11 bit recessivi (bus senza traffico). Da contatti avuti con Microchip si è convenuto però che tale modifica non è in linea con la filosofia di utilizzo fixed spiegata nella AN878, inoltre tale pratica comporta il momentaneo (anche se estremamente breve) isolamento del nodo CAN visto che durante la configurazione non è in grado nè di ricevere nè di trasmettere alcunchè. Risulta infatti non molto sensato far di tutto per concentrare le configurazioni unicamente in fase di inizializzazione per poi doverle modificare in corso d'opera. È previsto quindi un fixing della libreria relativa, nel frattempo per il nostro corso ci riferiamo alla ECAN (Polled) generata attraverso il Tool Maestro distribuito da Microchip che risulta perfettamente funzionante. Vedremo prossimamente come l'errore non comporti alcun problema nel momento in cui la libreria viene utilizzata in run-time visto che l'assegnazione ai registri filtro viene fatta attraverso le funzioni CANIDToRegs e RegsToCANID che sono implementate in maniera corretta. Per chiarezza confrontiamo che cosa accade quando usiamo la libreria errata e quella corretta:

LIBRERIA ERRATA:
RXF0SIDH = 123 >>3 = 00100100b
RXF0SIDL = 123 >>5 = 00001001b

Il SID è pari a: 100100000b = 120h. Siccome abbiamo definito come discriminanti i 2 bit meno significativi il filtro RXF0 accetterà soltanto i messaggi con un ID che finiscono per 00. Analogamente se consideriamo l'RXF1 avremo la sequenza:

RXF1SIDH = 121 >>3 = 00100100b
RXF1SIDL = 121 >>5 = 00001001b

Il SID è sempre pari a: 100100000b = 120h. Siccome abbiamo definito come discriminanti i 2 bit meno significativi il filtro RXF1 accetterà (anche lui) soltanto i messaggi con un ID che finiscono per 00b. È chiaro che in entrambe le situazioni il nostro firmware commetterà un errore e non riconoscerà nessuno dei messaggi inviati dal noto TX. Quest'ultimo, infatti trasmette id equivalenti a 11b o 01b. Se vogliamo essere precisi i messaggi verranno comunque accettati dal nodo di ricezione (attraverso i filtri superiori come RXF2) ma il firmware non li gestirà scartandoli. Vediamo invece cosa accade con la libreria corretta.

LIBRERIA CORRETTA:
RXF0SIDH = 123 >>3 = 00100100b
RXF0SIDL = 123 <<5 = 01100000b

Il SID è pari a: 00100100011b = 123h. Il valore è corretto pertanto verranno accettati solo i messaggi con ID che finiscono per 11b. Se prendiamo RXF1 si ha:

RXF1SIDH = 121 >>3 = 00100100b
RXF1SIDL = 121 <<5 = 00100000b

In questo caso il SID è pari a 00100100001b = 121h. Il valore è ancora corretto pertanto verranno accettati solo i messaggi con ID che finiscono per 01b. Si capisce quindi come l'utilizzo della prima libreria possa far funzionare il sistema soltanto nel caso in cui non vengano attivati i filtri.

La messa in funzione e l'analisi dei risultati

Per effettuare correttamente il nostro esperimento bisogna innanzitutto caricare nel PIC del nodo TX il nuovo firmware (CANTX.hex). Analogamente è necessario inserire nella EEPROM della stessa scheda il file EEP-

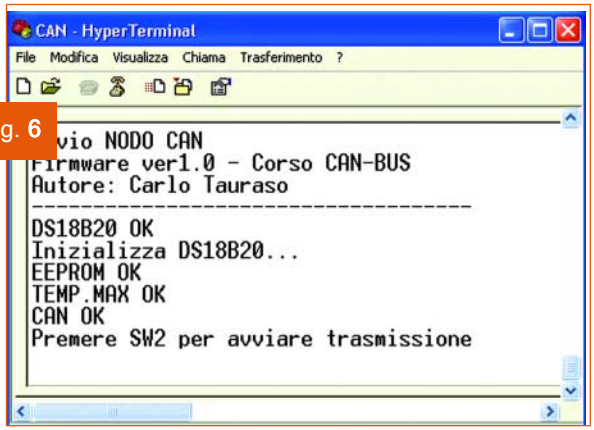


Fig. 6

CANTX.bin. Esso contiene nei primi due byte il valore massimo di temperatura oltre il quale ci si trova in una situazione di allarme. Noi per prova abbiamo utilizzato il 0190h corrispondente a 25°C. Naturalmente potete modificare tale valore in maniera che risulti compatibile con la temperatura dell'ambiente in cui vi trovate e il tipo di riscaldamento che intendete adottare. A questo punto prendiamo il nodo RX e carichiamo nel PIC il firmware (CANRX.hex). Colleghiamo, le porte CAN1 dei due nodi attraverso il solito cavetto con terminatori. Prepariamo sul PC una sessione Hyperterminal (19200, 8-N-1) e colleghiamo inizialmente la porta seriale del nodo TX. Alimentiamo il nodo di trasmissione. Se tutto funziona correttamente vedremo apparire sul terminale le stringhe di Figura 6.

Il nodo TX è pronto per iniziare la trasmissione dei valori di temperatura, il led verde risulta acceso. A questo punto spostiamo il cavo seriale sulla porta del nodo RX ed alimentiamolo. Vedremo apparire la stringa “Avvio nodo CAN” e si accenderà il led verde. Siamo pronti per avviare il sistema. Premiamo SW2 sul nodo TX. Immediatamente vedremo lampeggiare il led rosso su entrambe le schede segno che la comunicazione avviene correttamente. I valori ricevuti vengono registrati sulla EEPROM del nodo RX. Riscaldiamo ora la sonda del nodo TX in maniera da raggiungere il valore di soglia. Non appena questo avviene vedremo comparire sul nostro terminale la scritta “ATDT 115” (vedi Figura 7). Contemporaneamente sul nodo di ricezione si accende il led giallo. Si noti che il comando AT viene inviato una sola volta, dopodiché il firmware attende il rientro dell'allarme. Lasciamo raffreddare la sonda e vedremo che ad un certo punto il led giallo si spegnerà segno che la temperatura sarà tornata sotto il valore di soglia. Premiamo SW2 sul nodo RX finchè non si accende il led verde. Analogamente premiamo SW2 sul nodo TX finchè non si accende il led verde. Stacchiamo l'alimentazione ed estraiamo la EEPROM dal nodo RX. Proviamo a leggerla ad esempio con IC-Prog. Vedremo la sequenza di valori fino al raggiungimento dello stato di allarme evidenziato nei rettangoli di colore rosso (vedi Figura 8).

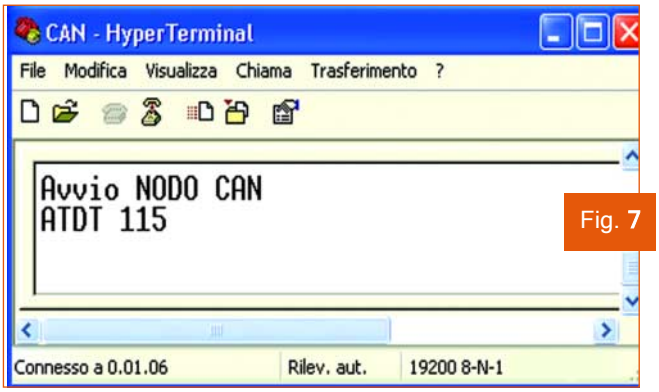


Fig. 7

Risulta evidente che dopo i primi 10 rilievi la temperatura supera i 25°C (abbiamo tenuto nel palmo della mano la sonda per qualche secondo ad una temperatura ambientale di 21°C). Il superamento della soglia viene rilevato per altre 10 misure, poi la sonda raffreddandosi fa scendere i valori al di sotto del limite 0190h e quindi l'allarme viene fatto rientrare. Il nodo in ricezione, se fate attenzione, non decide la situazione di allarme sulla base del valore trasferito ma sul tipo di messaggio ricevuto. La responsabilità dell'invio di un avviso di allarme è tutta del nodo TX. In questo modo è possibile distribuire la capacità di elaborazione del sistema. In pratica il nodo TX può analizzare lo stato di diversi tipi di sonde inviando un allarme specifico al nodo RX il quale avrà soltanto la responsabilità di gestire l'evento in maniera opportuna. Entrambi i firmware diventano più semplici visto che il problema viene diviso in due sotto-problemi più specifici.

Naturalmente qualcuno potrebbe obiettare che avremmo potuto effettuare uno switch anche direttamente sul valore dell'ID. In realtà la differenza (pur essendo sottile) porta formalmente ad un funzionamento operativo non conforme alle specifiche. In particolare la configurazione di filtri e maschere permette di lasciare l'incombenza di verifica dei messaggi di arrivo direttamente al motore incluso nel modulo CAN senza dover includere particolari sequenze condizionali nel >

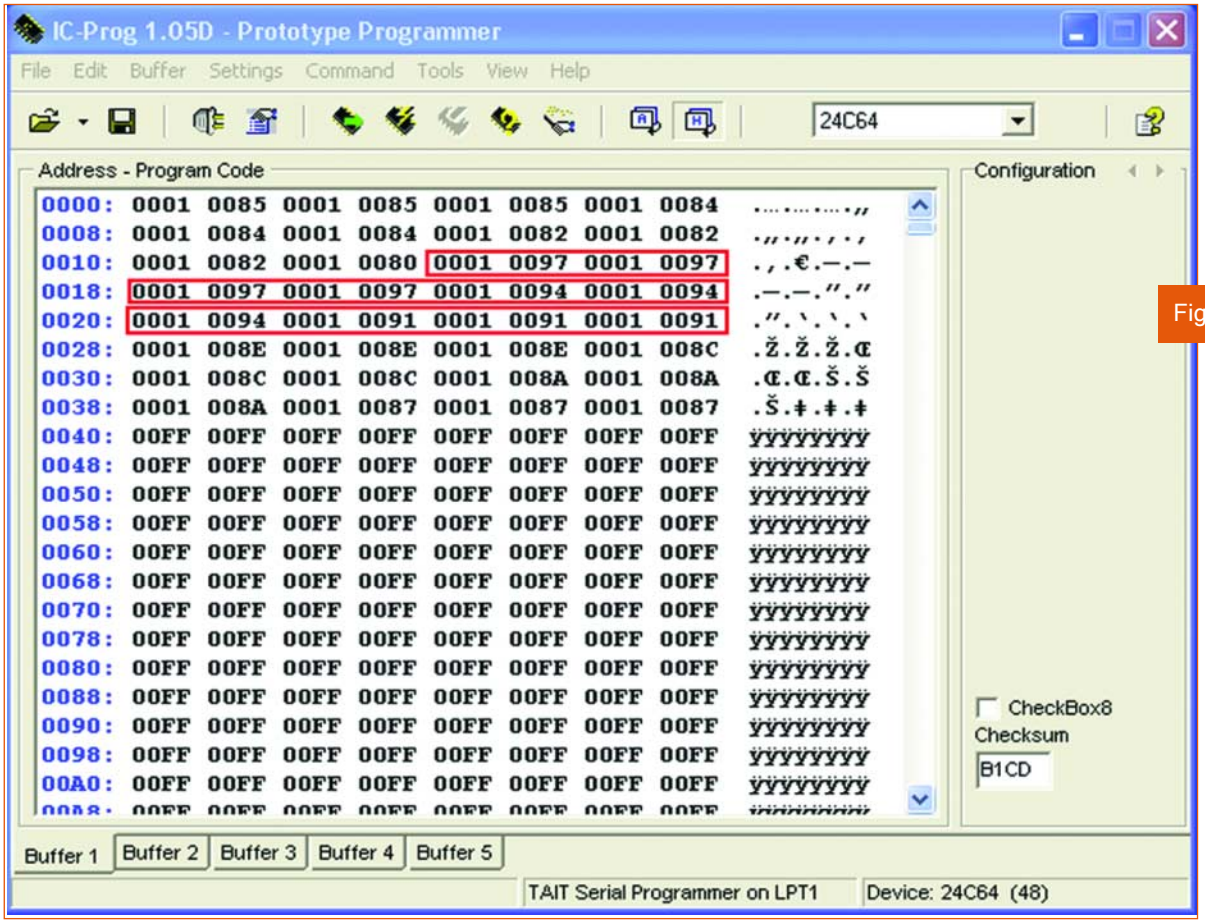


Fig. 8

firmware. Questa cosa diventa importante nella modalità estesa che utilizza ID più lunghi e quindi più onerosi da confrontare. Infine, è buona norma per ogni programmatore firmware quella di sfruttare al massimo l'efficienza di moduli hardwired evitando inutili complessità nella logica. Un'ultima considerazione va sicuramente alla possibilità di far scartare immediatamente i messaggi che non interessano al motore CAN senza quindi neanche prevederne la ricezione lato firmware.

Conclusioni

Siamo arrivati quasi al termine di questo nostro percorso teorico-pratico in cui abbiamo visto come sia possibile sviluppare un firmware in grado di far comunicare dei PIC attraverso il CAN bus. Facciamo un attimo il punto della situazione.

Nel primo esperimento abbiamo analizzato le funzioni fondamentali per l'invio e la ricezione dei messaggi. Poi abbiamo reso la cosa un pò più complicata realizzando un nodo di ricezione in grado di discriminare tra messaggi di differenti tipologie, elaborando solo quelle necessari. Siamo arrivati quindi al secondo esperimento in cui si sfruttavano i filtri proprio per sviluppare un sistema in grado di gestire in maniera "intelligente" le informazioni ricevute. Abbiamo cioè descritto le funzioni fondamentali che permettono di sfruttare le caratteristiche del CAN bus che lo differenziano da altri bus di comunicazione come l'USB o l'RS232 facendolo assomigliare ad una rete a basso livello.

I messaggi sono assimilabili a dei pacchetti di broadcasting mentre l'insieme di filtri e maschere permette di implementare un sistema di indirizzamento per certi aspetti simile a quello che ritroviamo nel livello Data Link.

La differenza fondamentale risiede nel fatto che l'indirizzo non è caratteristico di un unico nodo ma in teoria potrebbe essere elaborato da uno qualunque dei nodi presenti sulla rete. Sta al programmatore la scelta di come far fluire le informazioni e di che ruolo assegnare ad ogni nodo all'interno del suo sistema. Per completare il nostro discorso vogliamo ora descrivere un'ultima modalità di funzionamento della libreria ECAN: il cosiddetto run-time mode.

Può succedere in qualche applicazione che risulti necessario effettuare la modifica della configurazione CAN sulla base di determinate condizioni. Come? Lo vedremo nella prossima puntata.

Corso di programmazione: CAN BUS

a cura di Carlo Tauraso

Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa ottava puntata andiamo a considerare una particolare modalità di funzionamento della libreria ECAN.

Dopo aver analizzato le istruzioni necessarie ad effettuare lo scambio di messaggi tra due nodi CAN, introducendo il concetto di filtro e maschera andiamo a considerare una particolare modalità di funzionamento della libreria ECAN. Come avevamo già anticipato, questa libreria può essere utilizzata in "Fixed mode" o in "Run Time mode". Fino ad ora ci siamo occupati esclusivamente della prima, stabilendo tutte le configurazioni dei nostri nodi durante la compilazione modificando il file ECAN.def. Ora dobbiamo considerare una nuova possibilità secondo cui potrebbe essere necessario cambiare le caratteristiche del nodo durante il suo funzionamento. L'altra volta abbiamo realizzato un nodo TX che monitorava la temperatura ambiente attraverso una sonda DS18B20 ed inviava i valori ad un nodo RX. I messaggi inviati erano diversi a seconda che la temperatura si trovasse entro un determinato range oppure no. Il nodo RX discriminava i messaggi in arrivo attivando un segnale di allarme nel momento in cui la temperatura superava un determinato limite. Il tutto veniva gestito unicamente filtrando i messaggi e senza verificare nel concreto i valori inviati che venivano registrati direttamente nella EEPROM. Ora vogliamo integrare questo progetto analizzando la possibilità di modificare la configurazione del nodo RX attraverso alcuni particolari messaggi. In pratica, nel firmware RX prevediamo una funzione che all'avvio risulta disattivata e che viene eseguita soltanto nel momento in cui il nodo riceve un messaggio con un ID specifico. L'attivazione avviene riconfigurando il nodo ed inserendo un nuovo filtro. In questo modo tutti i messaggi successivi verranno correttamente elaborati anzichè essere scartati. Vediamo nel concreto come è possibile entrare in configurazione mentre un nodo è operativo. >

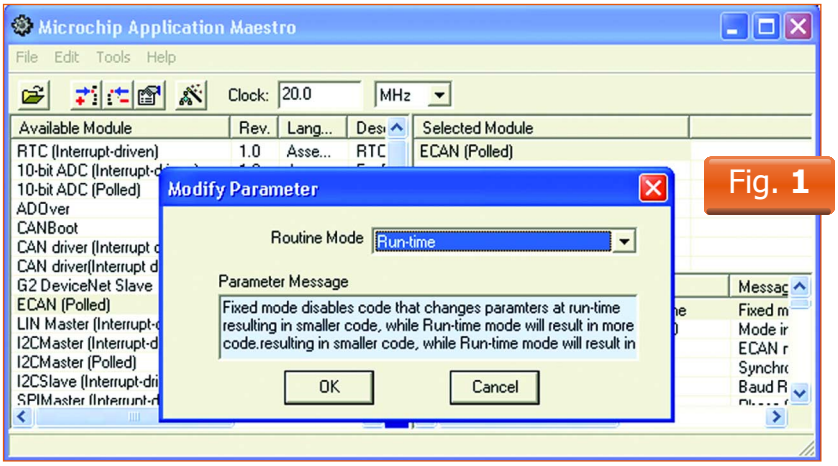


Fig. 1

ECAN che vengono utilizzate. Partiamo dalla ECANSetOperationMode che permette di precisare la modalità operativa del modulo CAN del PIC.

ECANSetOperationMode

Questa funzione viene richiamata nel momento in cui il nodo deve essere riconfigurato. La prima cosa da fare, infatti, è quella di portare il modulo CAN in configurazione. La funzione in questione riceve in ingresso un parametro che precisa la modalità operativa secondo una definizione che troviamo nella ECAN.h. Riassumiamo in Tabella 1 i valori possibili (per il dettaglio delle modalità rimandiamo al capitolo 3 del corso). La definizione relativa inclusa nel file ECAN.h è visibile nel Listato 1. I valori precisati nell'enumerazione devono venir assegnati al registro CANCON del PIC che è strutturato come in

Run-Time Mode

Per attivare questa modalità è necessario modificare il file ECAN.def che contiene tutti i parametri di configurazione iniziali del nodo. L'opzione fondamentale è la ECAN_LIB_MODE_VAL che permette di abilitare o disabilitare la riconfigurazione del nodo a runtime. Rigeneriamo il file in questione sempre attraverso il Microchip Application Maestro. In Figura 1 vediamo il passaggio dal valore ECAN_LIB_MODE_FIXED al ECAN_LIB_MODE_RUNTIME. Una volta effettuata la modifica possiamo avviare la creazione del file attraverso il comando "Generate Code". Naturalmente è bene considerare che il codice generato sarà più pesante in termini di lunghezza e di occupazione di memoria pertanto è consigliabile utilizzare questa modalità solo se l'applicazione che si sta sviluppando lo richiede. Bisogna anche tenere presente che durante la riconfigurazione il nodo blocca la ricezione e la trasmissione isolandosi per tutta l'operazione che avviene comunque molto rapidamente. Se non si prevede di modificare la configurazione dei nodi mentre sono operativi è sicuramente meglio utilizzare la precedente modalità. Per capire bene come avviene la riconfigurazione del nodo dobbiamo analizzare alcune funzioni della libreria

Tabella 1

Valore	Descrizione
ECAN_OP_MODE_NORMAL	Specifica la modalità operativa NORMAL
ECAN_OP_MODE_SLEEP	Specifica la modalità operativa DISABLE
ECAN_OP_MODE_LOOP	Specifica la modalità operativa LOOPBACK
ECAN_OP_MODE_LISTEN	Specifica la modalità operativa LISTEN-ONLY
ECAN_OP_MODE_CONFIG	Specifica la modalità operativa CONFIGURATION

Listato 1

```
typedef enum _ECAN_OP_MODE
{
  ECAN_OP_MODE_BITS     = 0xe0,
  ECAN_OP_MODE_NORMAL   = 0x00,
  ECAN_OP_MODE_SLEEP    = 0x20,
  ECAN_OP_MODE_LOOP     = 0x40,
  ECAN_OP_MODE_LISTEN   = 0x60,
  ECAN_OP_MODE_CONFIG   = 0x80
} ECAN_OP_MODE;
```

Fig. 2

CANCON: CAN CONTROL REGISTER

R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
REQOP2	REQOP1	REQOP0	ABAT	WIN2	WIN1	WIN0	-
bit 7							bit 0

Listato 2

```
void ECANSetOperationMode(ECAN_OP_MODE mode)
{
  CANCON &= 0x1F;

  CANCON |= mode;

 while( ECANGetOperationMode() != mode );
}
```

Valorizzazione dei bit ABAT, WIN0, WIN1, WIN2 che permettono di bloccare le trasmissioni e fissare il buffer di ricezione.

Valorizzazione dei bit REQOP0, REQOP1, REQOP2 che permettono di stabilire la modalità operativa del modulo CAN del PIC.

Ciclo che blocca l'esecuzione di ulteriori istruzioni finché il modulo CAN del PIC è entrato nella modalità selezionata.

Figura 2. I tre bit più significativi REQOP2, REQOP1, REQOP0 stabiliscono proprio la modalità operativa. Facendo riferimento ai datasheet del PIC18F458, il "CONFIGURATION mode" viene attivato mettendo ad 1 il REQOP2 indipendentemente dal valore degli altri due bit. La definizione stabilisce ECAN_OP_MODE_CONFIG = 80h = 10000000b che valorizza proprio tale bit. Il ritorno al "NORMAL mode" avviene azzerando i tre bit. Nel listato della funzione si vede chiaramente che il passaggio alla nuova modalità operativa avviene bloccando tutte le trasmissioni, ponendo ad 1 il bit ABAT (Abort All Pending Transmission bit) e fissando il buffer di ricezione in maniera che sia accessibile da qualunque banco di memoria attraverso i bit WIN0, WIN1, WIN2 (Window Address bits).
Non è possibile entrare in configurazione mentre è in corso una trasmissione. Dopo aver assegnato il valore corrispondente al registro CANCON, inizia un ciclo richiamando la ECANGetOperationMode finchè il modulo entra nella modalità operativa selezionata. È chiaro che la funzione è di tipo bloccante pertanto non viene eseguita nessuna ulteriore istruzione finché non esce dal ciclo. In realtà la Microchip prevede anche una macro non bloccante chiamata ECANSetOperationModeNoWait.

In quest'ultimo caso il ciclo non è incluso, pertanto la funzione ritorna immediatamente il controllo lasciando l'incombenza della verifica della modalità alle istruzioni successive. In questo modo il programmatore può inserire tra una verifica e l'altra delle istruzioni che non interagiscono con il modulo CAN ma sono necessarie per il funzionamento corretto del dispositivo. Vediamo il Listato 2 della ECANSetOperationMode.
La macro ECANGetOperationMode è esattamente complementare alla ECANSetOperationMode visto che permette di leggere i tre bit più significativi del CANSTAT rilevando quindi se il modulo CAN del PIC si trova in "CONFIGURATION Mode" o in un altra modalità. Il registro CANSTAT presenta la struttura visibile in Figura 3.
I tre bit OPMODE2, OPMODE1, OPMODE0 rappresentano le varie modalità operative del modulo CAN e prendono i medesimi valori del registro CANCON quindi per discriminarli è possibili riutilizzare la medesima enumerazione di prima. La macro ECANGetOperationMode viene definita nella ECAN.h. Vediamola nel dettaglio:

#define ECANGetOperationMode() (CANSTAT & ECAN_OP_MODE_BITS)

Si vede chiaramente che il valore del parametro in uscita viene determinato attraverso un AND logi- >

CANSTAT: CAN STATUS REGISTER

Fig. 3

R-1	R-0	R-0	U-0	R-0	R-0	R-0	U-0
OPMODE2	OPMODE1	OPMODE0	-	ICODE2	ICODE1	ICODE0	-
bit 7							bit 0

BRGCON1: BAUD RATE CONTROL REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BRP7	BRP6	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
bit 7							bit 0

SJW1: SJW0: Synchronized Jump Width bits

11 = Synchronized Jump Width Time = 4 x TQ
10 = Synchronized Jump Width Time = 3 x TQ
01 = Synchronized Jump Width Time = 2 x TQ
00 = Synchronized Jump Width Time = 1 x TQ

BRP5: BRP0: Baud Rate Prescaler bits

111111 = TQ = (2 x 64)/FOSC
111111 = TQ = (2 x 63)/FOSC
:
:
000001 = TQ = (2 x 2)/FOSC
000000 = TQ = (2 x 1)/FOSC

Fig. 4

co tra il valore del registro CANSTAT e l'ECAN_OP_MODE_BITS che equivale a E0h = 11100000. Vengono, quindi, estratti i 3 bit più significativi. Dopo essere entrati in "CONFIGURATION mode", il nodo non trasmette né riceve, possiamo scrivere i registri di configurazione, e quelli che controllano maschere e filtri. La libreria prevede delle funzioni apposite per far questo. Analizziamole separatamente.

ECANSetFunctionalMode

Permette di specificare la modalità di funzionamento del modulo CAN. Se ricordate all'inizio avevamo descritto in dettaglio le tre modalità: MODE0, MODE1, MODE2. Esse sono stabilite attraverso i 2 bit MDSEL0, MDSEL1 che si trovano nel registro ECANCON implementato solo nella famiglia superiore di PIC (18F6680/8680/8585/6585). In secondo luogo il PIC18F458 prevede soltanto il MODE0, pertanto nella libreria utilizzata nel corso abbiamo provveduto a commentare le istruzioni relative. Questa funzione, quindi, non la utilizziamo e ci limitiamo ad includere la sua descrizione solo per completezza.

ECANSetBaudRate

Permette di stabilire i parametri SJW, BRP, PHSEG1, PHSEG2, PROPSEG che avevamo descritto nel paragrafo relativo al file ECAN.def. I parametri passati in ingresso vengono utilizzati direttamente per valorizzare i registri BRGCON1, BRGCON2, BRGCON3 del PIC. Consideriamo ad esempio la struttura del registro di Figura 4.
Se osserviamo il listato della macro incluso nel file ECAN.h vediamo che attraverso uno shift a sinistra di 6 bit si allinea il valore relativo al salto di sincronizzazione ai due bit più significativi del registro BRGCON1. Il valore impostato viene decrementato di 1 per poter utilizzare due soli bit per rappresentare tutti i valori possibili (da 1 a 4). Utilizzando il valore reale sarebbero stati necessari 3 bit: da 000 a 100. Effettuato lo shift si esegue un OR con il valore del prescaler sempre decrementato di 1. Vengono così valorizzati gli ultimi 6 bit necessari del registro (Listato 3).

Listato 3

```
#define ECANSetBaudRate(sjw, brp, phseg1, phseg2, propseg)
    BRGCON1 = ((sjw-1) << 6) | (brp-1);
    BRGCON2 |= (((phseg1-1) << 3) | (propseg-1));
    BRGCON3 = phseg2;
```

Listato 4

```
#define ECANSetPHSEG2Mode(mode)     BRGCON2_SEG2PHTS = mode

   #define ECAN_PHSEG2_MODE_AUTOMATIC    0
   #define ECAN_PHSEG2_MODE_PROGRAMMABLE  1
```

ECANSetPHSEG2Mode

Permette di stabilire se il Phase Segment2 può venir programmato oppure deve essere gestito autonomamente dal modulo. I valori possibili in ingresso sono due: ECAN_PHSEG2_MODE_PROGRAMMABLE e ECAN_PHSEG2_MODE_AUTOMATIC come stabilito nelle definizioni incluse nel file ECAN.h (vedi Listato 4).
Il valore stabilito come si vede va a valorizzare direttamente il bit7 (SEG2PHTS) del registro BRGCON2.

ECANSetBusSampleMode

Stabilisce il tipo di campionamento del segnale di ricezione (una unica fase o tre fasi prima del punto di campionamento). I valori possibili in ingresso sono due:
ECAN_BUS_SAMPLE_MODE_THRICE, ECAN_BUS_SAMPLE_MODE_ONCE come stabilito nelle definizioni incluse nel file ECAN.h (vedi Listato 5). Anche in questo caso avviene una valorizzazione diretta del bit6 (SAM) del registro BRGCON2.

Listato 5

```
#define ECANSetBusSampleMode(mode)  BRGCON2_SAM = mode

   #define ECAN_BUS_SAMPLE_MODE_ONCE     0
   #define ECAN_BUS_SAMPLE_MODE_THRICE    1
```

ECANSetWakeupMode

Stabilisce se attivare o meno la modalità di gestione del WakeUp in base all'attività presente sul bus. Si tratta di una feature disponibile nella classe superiore e viene attivata settando il bit7 (WAKDIS) del registro BRGCON3. Sul PIC18F458 tale bit non esiste.

ECANSetFilterMode

Permette di attivare o disattivare il filtro passa-basso per rilevare l'attività del bus. I valori possibili in ingresso sono due: ECAN_FILTER_MODE_DISABLE, ECAN_FILTER_MODE_ENABLE come stabilito nelle definizioni incluse nel file ECAN.h (Listato 6). La macro insiste direttamente sul bit6 del registro BRGCON3.

Listato 6

```
#define ECANSetFilterMode(mode)     BRGCON3_WAKFIL = mode

   #define ECAN_FILTER_MODE_DISABLE     0
   #define ECAN_FILTER_MODE_ENABLE      1
```

ECANSetTxDriveMode

Permette di stabilire in che modo viene comandata la linea di trasmissione in uno stato recessivo. I valori possibili in ingresso sono due: ECAN_TXDRIVE_MODE_TRISTATE e ECAN_TXDRIVE_MODE_VDD come stabilito >

Listato 7

```
#define ECANSetTxDriveMode(mode)    CIOCON_ENDRHI = mode

   #define ECAN_TXDRIVE_MODE_TRISTATE 0
   #define ECAN_TXDRIVE_MODE_VDD     1
```

nelle definizioni incluse nel file ECAN.h (vedi Listato 7).
La macro come nei casi precedenti va a valorizzare un bit specifico di un registro del PIC. Si tratta del bit5 del registro CIOCON.

ECANSetCANTX2, ECANSetCANTX2Source, ECANDisableCANTX2

Si tratta di 3 macro che permettono di abilitare e configurare il funzionamento della seconda linea di trasmissione CAN presente sui PIC della famiglia più elevata. Sul PIC18F458 esiste un unica linea di trasmissione che fa capo al pin RB2.

ECANSetCaptureMode

Permette di attivare o disattivare la possibilità di generare un time-stamp per ogni messaggio ricevuto attraverso il campionamento del segnale su CCP1. Ciò comporta chiaramente una configurazione dei comparatori ed in particolare della linea CCP1.

I valori possibili in ingresso sono due: ECAN_CAPTURE_MODE_DISABLE e ECAN_CAPTURE_MODE_ENABLE come stabilito nelle definizioni incluse nel file ECAN.h (Listato 8). Come si vede dal listato la macro agisce direttamente sul bit4 (CANCAP) del registro CIOCAN.

ECANSetRXB0DblBuffer

Stabilisce se attivare o meno la modalità "double buffer" sul buffer di ricezione RXB0. Anche in questo caso esistono soltanto due valori possibili in ingresso definiti sempre nel file ECAN.h. Da notare che questa opzione è disponibile in MODE0 o se si utilizza la libreria in "Runtime Mode" (vedi Listato 9). La macro valorizza il bit2 (RXB0DBEN) del registro RXB0CON.

ECANSetRxBnRxMode

Questa macro prende in ingresso due parametri: il buffer di ricezione e il tipo di messaggi che quest'ultimo è abilitato a ricevere. Nel Listato 10 il parametro buffer viene utilizzato per stabilire

Listato 8

```
#define ECANSetCaptureMode(mode)    CIOCON_CANCAP = mode

    #define ECAN_CAPTURE_MODE_DISABLE   0
    #define ECAN_CAPTURE_MODE_ENABLE   1
```

Listato 9

```
#if ( (ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME) || \
     (ECAN_FUNC_MODE_VAL == ECAN_MODE_0) )

    #define ECANSetRXB0DblBuffer(mode)     RXB0CON_RXB0DBEN = mode
#endif

    #define ECAN_DBL_BUFFER_MODE_DISABLE   0
    #define ECAN_DBL_BUFFER_MODE_ENABLE    1
```

Listato 10

```
#define ECANSetRxBnRxMode(buffer, mode)
      ##buffer##CON_RXM1 = mode >> 1;
      ##buffer##CON_RXM0 = mode;

    #define ECAN_RECEIVE_ALL_VALID 0
    #define ECAN_RECEIVE_STANDARD  1
    #define ECAN_RECEIVE_EXTENDED  2
    #define ECAN_RECEIVE_ALL     3
```

Listato 11

```
#define ECANSetRXF0Value(val, type)
  RXFCON0_RXF0EN = 1;
   _CANIDToRegs((BYTE*)&RXF0SIDH, val, type)

  struct
   {
    struct
     {
      unsigned SIDL:3;      // SIDL<5:7>
      unsigned SIDH:5;      // SIDH<0:4>
     } BYTE1;
    struct
     {
      unsigned SIDHU:3;      // SIDH<5:7>
      unsigned EIDL_LN:5;     // EIDL<0:4>
     } BYTE2;
    struct
     {
      unsigned EIDL_UN:3;     // EIDL<5:7>
      unsigned EIDH_LN:5;     // EIDH<0:4>
     } BYTE3;
    struct
     {
      unsigned EIDH_UN:3;     // EIDH<5:7>
      unsigned EIDHU:2;      // SIDL<0:1>
      unsigned :3;
     } BYTE4;
   } ID_VALS;

  struct
   {
    BYTE BYTE_1;
    BYTE BYTE_2;
    BYTE BYTE_3;
    BYTE BYTE_4;
   } BYTES;
} CAN_MESSAGE_ID;
```

Questa macro viene ripetuta per ciascun filtro disponibile RXF0...RXF5. Essa richiama la funzione CANIDToRegs che permette di valorizzare i registri RXFnSIDH, RXFnSIDL.

Struttura per rimappare i bit di ciascun registro presente nelle due famiglie di PIC. Se osservate bene il BYTE1 e il BYTE2 permettono di coprire tutti i bit previsti per gli ID standard nel 18F458.

Struttura per accedere ai singoli byte del ID del messaggio.

se ci si sta riferendo al registro RXB0CON o al registro RXB1CON.
Vengono valorizzati il bit6 (RXM1) e il bit5 (RXM0) del registro di controllo del buffer di ricezione RXB0 o RXB1. In questo modo si stabilisce se il buffer viene abilitato a ricevere ad esempio solo i messaggi con un un ID standard (11bit) o con un ID esteso, oppure se riceve tutti i messaggi, anche quelli errati.

ECANSetBnRxMode

Questa macro è analoga alla precedente soltanto che permette di gestire il bit più significativo (RXM1) della coppia che controlla i messaggi ricevibili dal nodo. In questo modo è possibile solo stabilire se il buffer può ricevere tutti i tipi di messaggi (anche quelli errati) oppure solo quelli validi. La macro può agire su 6 buffer di ricezione (non in Mode0) e quindi risulta riservata per la famiglia superiore.

ECANSetBnAutoRTRMode

Anche per questa macro dobbiamo riferirci alla famiglia di PIC superiore. Serve per attivare la modalità RTR (Remote Transmission Request) per ciascuno dei 6 buffer disponibili. In questo modo è possibile gestire l'interrogazione remota del nodo.

ECANSetBnTxRxMode

Questa macro configura ciascuno dei 6 buffer disponibili in trasmissione o ricezione. Viene uti- >

lizzata nella famiglia di PIC superiore. Se fate attenzione, tutte queste funzioni che abbiamo elencato hanno un collegamento diretto con i parametri definiti all'interno del file ECAN.def. Avvicinando la prima tabella riassuntiva (relativa al Microchip Application Maestro) pubblicata sul numero precedente troverete una diretta corrispondenza tra i parametri elencati e le relative funzioni descritte in questo paragrafo. Passiamo ora a considerare il gruppo di funzioni più importante perché permette di gestire filtri e maschere.

Modificare filtri e maschere a Runtime

La cosa che risulta essere particolarmente interessante a livello operativo è la possibilità di aggiungere un filtro o modificare una maschera durante la fase operativa del nodo. Precisiamo che in realtà la modifica avviene isolando il nodo dal bus, quindi durante la configurazione esso non può trasmettere o ricevere alcun messaggio. Vediamo nel dettaglio le funzioni e le macro corrispondenti:

ECANSetRXFnValue

Queste macro prendono in ingresso due parametri: uno relativo al valore da assegnare al filtro ed uno relativo al tipo di ID da filtrare. Quest'ultimo parametro prevede due possibili valori corrispondenti alle due define incluse nel file ECAN.h: ECAN_MSG_STD (ID standard a 11bit), ECAN_MSG_XTD (ID esteso a 29bit). In realtà la libreria prevede ben 16 macro, una per ciascun filtro previsto. In Mode0 e utilizzando un 18F458 è possibile accedere ai primi 6 registri RXF0...RXF5. Se osserviamo il listato seguente vediamo che dapprima si valorizza il bit RXFnEN per attivare il filtro (parametro ECAN_RXFn_MODE_VAL in ECAN.def). Il registro RXFCONn è implementato nella famiglia superiore mentre per il 18F458 i 6 filtri risultano comunque attivi. Naturalmente nel caso in cui il filtro è azzerato accetta tutti i messaggi in ingresso. Poi si richiama una funzione denominata CANIDToRegs. Se ricordate, nell'altra puntata avevamo accennato a questa funzione mentre parlavamo del problema rilevato nella AN878. Essa permette di valorizzare correttamente i registri RXFnSIDH, RXFnSIDL con l'ID del messaggio che vogliamo filtrare ed è in grado di gestire sia l'ID standard che quello extended. La funzione prende in ingresso un puntatore al buffer da aggiornare, il valore a 32 bit contenente l'ID, la tipologia standard o extended dell'ID. L'identificativo del messaggio da filtrare viene organizzato in una struttura a 4 byte in maniera da rimappare tutti i bit dei registri dedicati sia nella famiglia inferiore che in quella superiore. Vediamo nel dettaglio il Listato 11 relativo.

Se consideriamo i due registri RXFnSIDH e RXFnSIDL di cui riportiamo la struttura nelle due figure seguenti vediamo come il codice effettui la valorizzazione dei bit SID10...SID0 attraverso delle semplici operazioni di shift. In particolare si estraggono i 5 bit più significativi del BYTE1 per i 5 bit meno significativi (SID7, SID6, SID5, SID4, SID3) del registro RXFnSIDH. Si estraggono i 3 bit meno significativi del BYTE2 per i 3 più significativi del

Fig. 5

RXFnSIDH: RECEIVE ACCEPTANCE FILTER n STANDARD IDENTIFIER FILTER, HIGH BYTE REGISTERS

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

RXFnSIDH: RECEIVE ACCEPTANCE FILTER n STANDARD IDENTIFIER FILTER, LOW BYTE REGISTERS

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	-	EXIDEN	-	EID17	EID16
bit 7							bit 0

Listato 12

```
void _CANIDToRegs(BYTE* ptr,
          unsigned long val,
          BYTE type)
{
  CAN_MESSAGE_ID *Value;

  Value =  (CAN_MESSAGE_ID*)&val;

  if ( type == ECAN_MSG_STD )
   {
     *ptr = Value->BYTES.BYTE_1 >> 3;


     *ptr |= (Value->BYTES.BYTE_2 << 5);


    ptr++;

     *ptr = Value->BYTES.BYTE_1 << 5;

    // Copy SID<2:0> to SIDL<7:5>

   }
}
```

Effettuando uno shift a destra di 3 bit si estraggono i 5 bit più significativi di BYTE1 che valorizzano SID7...SID3 nel registro RXFnSIDH.

Effettuando uno shift a sinistra di 5 bit si estraggono i 3 bit meno significativi di BYTE2 che valorizzano SID10...SID8 nel registro RXFnSIDH.

Incrementando il puntatore si va ad indirizzare il registro RXFnSIDL.

Effettuando uno shift a sinistra di 5 bit si estraggono i 3 bit meno significativi di BYTE1 che valorizzano SID2...SID0 nel registro RXFnSIDL.

registro RXFnSIDH (SID10, SID9, SID8). Il puntatore viene incrementato per accedere al registro RXFnSIDL. A questo punto si estraggono i 3 bit rimanenti del BYTE1 per i 3 più significativi del RXFnSIDL (SID2, SID1, SID0) come specificato in Figura 5 e nel Listato 12.
Nel momento in cui ci troveremo in Configuration mode sarà, quindi, possibile attivare un filtro su uno dei 6 buffer presenti sul nostro PIC per fare in modo che il microcontrollore cominci a filtrare i messaggi con l'ID stabilito. Ad esempio possiamo attivare un filtro per tutti i messaggi standard con ID=121h semplicemente richiamando la funzione:

ECANSetRXF0Value(0x121, ECAN_MSG_STD);

ECANSetRXMnValue

Queste macro prendono in ingresso due parametri: uno relativo al valore da assegnare alla maschera ed uno relativo al tipo di ID che viene filtrato. Quest'ultimo parametro prevede due possibili valori corrispondenti alle due define incluse nel file ECAN.h: ECAN_MSG_STD (ID standard a 11bit), ECAN_MSG_XTD (ID esteso a 29bit). La libreria prevede 2 macro, una per RXM0 ed una RXM1. Anche in questo caso nella macro si richiama la CANIDToRegs allineando il valore dell'ID al registro RXMnSIDH. La spiegazione del precedente paragrafo può essere tranquillamente ripetuta anche per questo caso. Naturalmente, il valore trasferito è sempre l'ID del messaggio che viene filtrato ma qui il registro valorizzato ha un significato differente. Il RXMnSIDH stabilisce, infatti, i bit dell'ID che vengono considerati durante l'operazione di filtro. In pratica nel confronto vengono utilizzati solo i bit dell'ID corrispondenti ai bit 1 della maschera. Quindi il programmatore potrebbe decidere anche di filtrare non un singolo ID ma un gruppo di messaggi. È quello che abbiamo fatto nel nostro precedente esperimento. Se ricordate abbiamo valorizzato RXM0 con un valore pari a 3 considerando quindi soltanto i 2 bit meno significativi dell'ID. È chiaro che in questo modo abbiamo creato quattro grandi gruppi di messaggi: quelli che terminano per 00, 01, 10, 11. Nel firmware del nodo trasmittente abbiamo poi utilizzato soltanto due messaggi: uno con ID=123 ed uno con ID=121. Il primo appartiene al gruppo di quelli che terminano per 11 mentre il secondo appartiene al gruppo di quelli che terminano per 01. Se consideriamo i due registri RXMnSIDH e RXMnSIDL di cui riportiamo la >

Fig. 6

RXMnSIDTH: RECEIVE ACCEPTANCE MASK n STANDARD IDENTIFIER MASK, HIGH BYTE REGISTERS

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

RXMnSIDL: RECEIVE ACCEPTANCE MASK n STANDARD IDENTIFIER MASK, LOW BYTE REGISTERS

R/W-x	R/W-x	R/W-x	U-0	U-0	U-0	R/W-x	R/W-x
SID2	SID1	SID0	-	-	-	EID17	EID16
bit 7							bit 0

struttura nelle due figure seguenti vediamo come il codice effettui la valorizzazione dei bit SID10...SID0 attraverso le solite operazioni di shift. Da notare che la struttura dei registri per quanto riguarda il SID rispecchia la stessa vista nei RXFnSIDH, RXFnSIDL (vedi Figura 6).

Facendo riferimento al listato della CANIDToRegs incluso nel paragrafo precedente si estraggono i 5 bit più significativi del BYTE1 per i 5 bit meno significativi (SID7, SID6, SID5, SID4, SID3) del registro RXMnSIDH. Si estraggono i 3 bit meno significativi del BYTE2 per i 3 più significativi del registro RXMnSIDH (SID10, SID9, SID8). Il puntatore viene incrementato per accedere al registro RXMnSIDL. A questo punto si estraggono i 3 bit rimanenti del BYTE1 per i 3 più significativi del RXMnSIDL (SID2, SID1, SID0). Vediamo come si presenta il Listato 13 della macro incluso nel file ECAN.h.

In Configuration mode è possibile precisare il valore della maschera RXM0 per filtrare tutti i messaggi standard con ID=123 semplicemente richiamando la funzione:

ECANSetRXM0Value(0x123, ECAN_MSG_STD);

Analogamente è possibile filtrare tutti i messaggi considerando soltanto i 2 bit meno significativi dell'ID richiamando la funzione:

ECANSetRXM0Value(0x03, ECAN_MSG_STD);

Nella libreria generata attraverso il Microchip Application Maestro troverete anche altre funzioni come ad esempio la ECANLinkRXFnFnToBuffer o la ECANLinkRXFnThrunToMask che permettono rispettivamente di collegare in maniera dinamica filtri e buffer o filtri e maschere. Si tratta di funzionalità che sono state implementate nei moduli integrati della classe superiore di PIC attraverso i registri RXFBCONn e MSELn. Dopo aver chiarito tutta la struttura che sta alla base della gestione dei parametri di configurazione a Run-Time siamo pronti per introdurre il nostro ultimo esperimento. Utilizzeremo la possibilità di riconfigurazione per modificare o aggiungere un filtro sul nodo di ricezione nel momento in cui viene inviato un particolare messaggio. In questo modo è possibile attivare o disattivare determinate sequenze di istruzioni che possono servire per far fronte a determinate situazioni. Ad esempio è possibile pensare ad un differente funzionamento del nodo di ricezione nel caso in cui si trovi in una condizione di allarme e quindi prevedere un diverso gruppo di messaggi per gestire tale condizione. Analizzeremo il listato firmware dei due nodi e concluderemo il nostro percorso. Alla prossima.

Listato 13

```
#define ECANSetRXM0Value(val, type)
        _CANIDToRegs((BYTE*)&RXM0SIDH, val, type)
#define ECANSetRXM1Value(val, type)
        _CANIDToRegs((BYTE*)&RXM1SIDH, val, type)
```

Corso di programmazione: CAN BUS

a cura di Carlo Tauraso

Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa nona puntata presentiamo un'evoluzione di un caso concreto del quale ci eravamo già occupati in precedenza.

9

Siamo arrivati al capitolo conclusivo di questo corso al quale, tuttavia, faranno seguito altre due puntate dedicate ad un sistema in grado di registrare in real-time i messaggi scambiati tra due nodi. In queste pagine, dopo avere recentemente analizzato l'insieme di funzioni necessarie a riconfigurare un nodo CAN, passiamo ad occuparci di un caso concreto. Si tratta di un'evoluzione dell'esperimento presentato nella sesta puntata, un nodo di trasmissione che monitorava la temperatura ambiente ed inviava i valori rilevati ad un nodo di ricezione. I valori venivano normalmente inviati utilizzando messaggi CAN con ID=121. Nel momento in cui la temperatura superava uno specifico limite memorizzato nella EEPROM, il nodo TX cominciava ad inviare messaggi con ID=123 che venivano riconosciuti dal nodo RX come messaggi di allarme. Tutte le temperature venivano registrate sulla EEPROM del nodo RX che, in caso di superamento del limite massimo, provvedeva ad inviare attraverso la seriale un comando Hayes per chiamare il 115. Il primo esperimento prevedeva soltanto l'invio dei valori e la loro registrazione sulla EEPROM del nodo RX. Ora vogliamo attivare la funzione di allarme dopo l'invio da parte del nodo TX di un messaggio particolare con ID=120.

Il firmware di ricezione prevede già la funzione di allarme che risulta essere disattivata all'avvio. Soltanto dopo l'arrivo del messaggio corrispondente il filtro d'allarme viene attivato e gli ID=123 vengono gestiti attraverso la chiamata telefonica. In tutti gli altri casi il nodo si limita a registrare i valori di temperatura su EEPROM. Per stabilire il momento in cui effettuare la riconfigurazione leghiamo l'evento alla pressione del tasto che insiste sul pin RB1. Proveremo, quindi, a verificare il diverso comportamento del nodo RX con allarme attivato e disattivato. Iniziamo con il firmware del nodo TX. >

Nodo TX: il firmware

Nel codice del nodo di trasmissione abbiamo fatto una modifica piuttosto semplice. In pratica nel momento in cui RB1 va a livello basso (pulsante premuto) si avvia il ciclo di trasmissione di un messaggio con ID=120. Per rendere le cose più semplici il nodo invia sulla seriale un apposito messaggio per avvertire l'utente che la sequenza è stata eseguita. Vediamo il Listato 1.

Se registriamo i messaggi inviati mentre la temperatura ambiente non è superiore al limite prestabilito e forziamo una riconfigurazione vedremo una sequenza come quella riportata in Fig. 1. Abbiamo evidenziato con un rettangolo rosso il messaggio di riconfigurazione inviato in corrispondenza della pressione del tasto. I 2 byte trasferiti riportano il valore dell'ultima temperatura rilevata e non vengono considerati dal nodo di ricezione.

Quest'ultimo dovrà essere in grado di intercettare il messaggio in questione, entrare in "configuration mode" e modificare il filtro relativo allo stato di allarme.

Passiamo ora al nodo di ricezione.

Listato 1

```
if (PORTBbits.RB1 == 0)
  {

  while(!ECANSendMessage(0x120, data, 2, ECAN_TX_STD_FRAME));


  putrsUSART("TX RICONFIG\n\r");


  Delay10KTCYx(5000);

  }
```

Quando viene premuto il pulsante RB1 va a livello logico basso.

Si invia un messaggio con ID=120. I 2 byte allegati riportano l'ultimo valore di temperatura rilevato.

Viene trasmesso un messaggio di avvertimento sulla seriale e successivamente il PIC rimane in pausa.

Nodo RX: il firmware

Le modifiche sul nodo RX sono decisamente più complesse. Innanzitutto è necessario generare un nuovo file ECAN.def. Per farlo ci appoggiamo sempre al tool Application Maestro di Microchip. I parametri essenziali da considerare sono: Routine Mode, RXF0 Value, RXF1 Value, RXF2 Value, RXM0 Value, RXM1 Value. Il primo deve passare da FIXED a RUNTIME. In

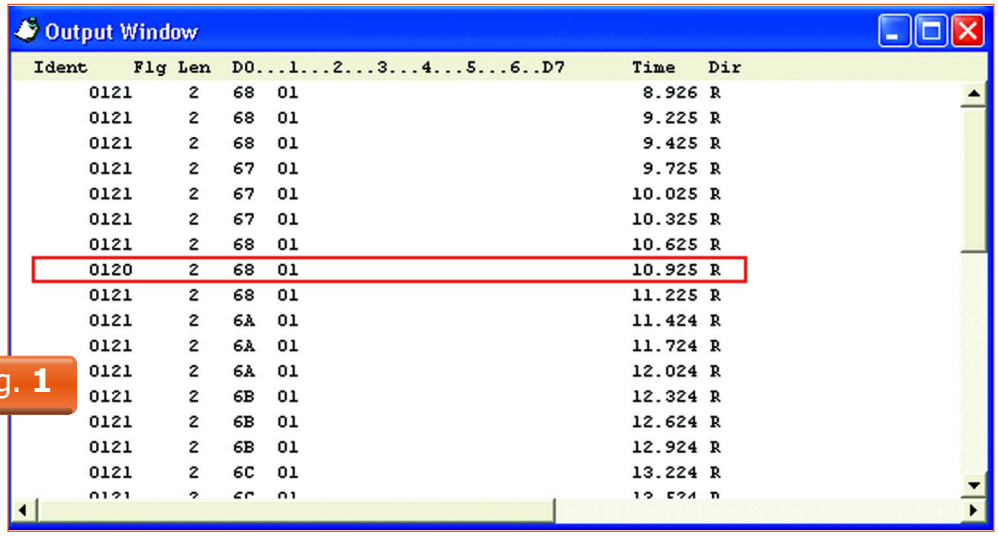


Fig. 1

questo modo facciamo sì che il modulo CAN si avvii con una configurazione predefinita modificabile anzichè costante (Figura 2).
Gli altri valori riguardano esclusivamente il funzionamento di filtri e maschere. Consideriamo che il nodo di ricezione potrà ricevere inizialmente tre possibili messaggi visibili in Tabella 1.

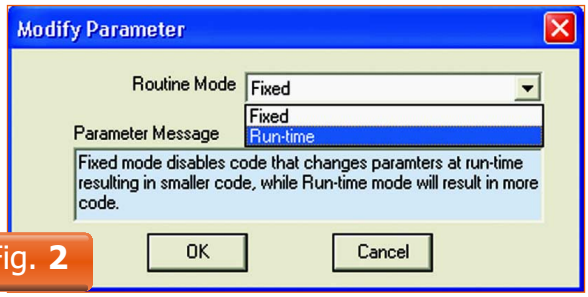
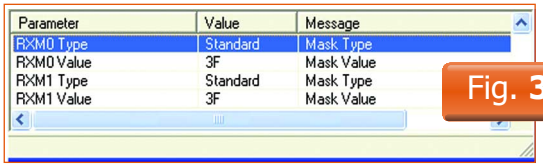


Fig. 2

Tabella 1

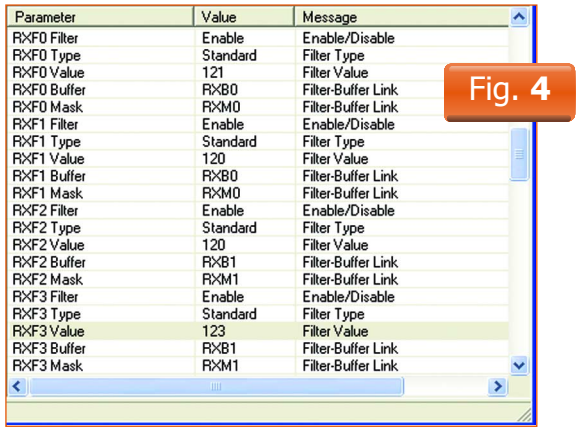
ID	Descrizione
120	Il nodo deve venir riconfigurato pertanto devono essere fermate ricezioni e trasmissioni ed aggiunto un filtro sull'ID = 121 per gestire lo stato di allarme.
123	La temperatura ha raggiunto il limite prestabilito, il nodo inizialmente deve registrare soltanto il valore in EEPROM e non fare altro. Dopo la riconfigurazione, invece, oltre a registrare il valore dovrà anche gestire l'allarme attraverso l'invio della stringa AT.
121	La temperatura è inferiore al limite prestabilito, il nodo registra il valore in EEPROM sia prima che dopo la riconfigurazione.

Per discriminarli correttamente aumentiamo la lunghezza della maschera portandola da 2 a 6 bit. Si faccia attenzione che la ricezione di un messaggio riguardante uno stato di allarme prefigura due diversi scenari a seconda che sia stata fatta o meno la riconfigurazione. Nello scenario predefinito il nodo si limita a registrare il valore mentre in quello "riconfigurato" deve inviare anche la stringa AT attraverso la seriale. Realizziamo, cioè, un sistema le cui funzionalità vengono estese attraverso un messaggio particolare inviato dal nodo di trasmissione. Naturalmente si tratta di un esempio didattico visto che la differenza sta solo nell'invio di una stringa. Una volta capito si possono implementare ed assegnare funzioni ben più complesse. La differenza sta tutta nella possibilità di riconfigurare il funzionamento di un nodo dall'esterno senza la necessità di agire sul firmware per far fronte a determinate condizioni. La differenza può essere per certi aspetti sottile ma nello stesso tempo potente visto che potremmo prevedere molti scenari differenti. Un nodo potrebbe ad esempio essere assimilato ad una macchina a stati finiti ad ognuno dei quali si assegna un determinato gruppo di messaggi da gestire. Il nodo TX potrebbe comandare le transizioni tra uno stato e l'altro del nodo RX a seconda di particolari input esterni (pressione di tasti, sonde ecc). Ad ogni transizione il nodo viene riconfigurato ed i filtri modificati per poter gestire i diversi gruppi di messaggi. Ma torniamo al nostro esperimento. Sulla base di quanto detto possiamo immaginare che il nodo ha uno stato iniziale che chiameremo MONITOR in cui registra soltanto i valori di temperatura ed uno stato finale che chiameremo GESTORE in cui se la temperatura supera un certo limite il nodo gestisce l'evento con una chiamata telefonica. In questo caso chiameremo il passaggio di stato riconfigurazione, e tale passaggio sarà anche di tipo irreversibile nel senso che sarà necessario togliere alimentazione al nodo RX per tornare allo stato iniziale.
Riassumiamo nelle figure 3 e 4 i parametri relativi a maschere e filtri. Abbiamo reso la cosa un pò più complicata per illustrare contemporaneamente anche l'aspetto della priorità assegnata ai filtri. Se vi ricordate avevamo spiegato che i filtri RXFn funzionano in maniera da rendere prioritari quelli con n inferiore. In questo caso abbiamo assegnato sia a RXF1 che a RXF2 un ID=120. Vi sarete già chiesti quale sarà il risultato della fun- >



Parameter	Value	Message
RXM0 Type	Standard	Mask Type
RXM0 Value	3F	Mask Value
RXM1 Type	Standard	Mask Type
RXM1 Value	3F	Mask Value

Fig. 3



Parameter	Value	Message
RXF0 Filter	Enable	Enable/Disable
RXF0 Type	Standard	Filter Type
RXF0 Value	121	Filter Value
RXF0 Buffer	RXB0	Filter-Buffer Link
RXF0 Mask	RXM0	Filter-Buffer Link
RXF1 Filter	Enable	Enable/Disable
RXF1 Type	Standard	Filter Type
RXF1 Value	120	Filter Value
RXF1 Buffer	RXB0	Filter-Buffer Link
RXF1 Mask	RXM0	Filter-Buffer Link
RXF2 Filter	Enable	Enable/Disable
RXF2 Type	Standard	Filter Type
RXF2 Value	120	Filter Value
RXF2 Buffer	RXB1	Filter-Buffer Link
RXF2 Mask	RXM1	Filter-Buffer Link
RXF3 Filter	Enable	Enable/Disable
RXF3 Type	Standard	Filter Type
RXF3 Value	123	Filter Value
RXF3 Buffer	RXB1	Filter-Buffer Link
RXF3 Mask	RXM1	Filter-Buffer Link

Fig. 4

Listato 2

```
    switch (ECANGetFilterHitInfo())
     {

case 0:
            PORTC_RCO = 0;
            putrsUSART("TEMP<=LIM\n\r");
            break;


case 1:
          PORTC_RCO = 0;
          putrsUSART("INI-CONFIG\n\r");
          ECANSetOperationMode(ECAN_OP_MODE_CONFIG);
          ECANSetRXF2Value(0x123, ECAN_MSG_STD);
          ECANSetOperationMode(ECAN_OP_MODE_NORMAL);
          putrsUSART("FINE-CONFIG\n\r");
          break;


case 2:
          if (PORTC_RCO == 0)
          {
          putrsUSART("ATDT115\n\r");
          PORTC_RCO = 1; //LED GIALLO
          }
          break;


case 3:
          PORTC_RCO = 0;
          putrsUSART("TEMP>LIM\n\r");
          break;


      }
```

Filtro RXF0, messaggio con ID=121h la temperatura risulta essere inferiore al limite prestabilito, il nodo invia soltanto un messaggio di segnalazione sulla seriale.

Filtro RXF1, messaggio con ID=120h il nodo TX ha richiesto una riconfigurazione. Richiamando la ECANSetOperationMode si entra nella modalità di configurazione. Trasmissioni e ricezioni vengono bloccate. Il valore dei registri ECANRXF2SIDH e ECANRXF2SIDL viene impostato in maniera da intercettare gli ID=123h. Terminata l'operazione viene richiamata nuovamente la ECANSetOperationMode per rientrare nella modalità normale. Trasmissioni e ricezioni riprendono regolarmente.

Filtro RXF2, messaggio con ID=123h ricevuto successivamente all'operazione di riconfigurazione. Si faccia attenzione che all'avvio questo filtro è configurato per intercettare un ID=120h ma viene sempre preceduto dal RXF1 a causa della priorità. Solo dopo la configurazione è in grado di rilevare i messaggi con ID=123h corrispondente ad una temperatura superiore al limite prestabilito. Il nodo quindi, invia una stringa contenente il comando Hayes "ATDT115". L'invio avviene un'unica volta (grazie alla if iniziale) per tutta la durata dello stato di allarme assieme all'accensione del led giallo.

Filtro RXF3, messaggio con ID=123h la temperatura risulta essere superiore al limite prestabilito ma il nodo non è stato ancora riconfigurato. Ci si limita quindi ad inviare la stringa di segnalazione su seriale. Nel momento in cui viene effettuata la riconfigurazione questo filtro verrà sempre preceduto dal RXF2 a causa della priorità.

zione GetFilterHitInfo nel momento in cui il messaggio verrà ricevuto dal nodo. Ebbene il risultato sarà sempre pari a 1 perchè RXF1 ha una priorità maggiore di RXF2. In secondo luogo abbiamo assegnato a RXF3 l'ID=121 cioè quello relativo allo stato di allarme. Questo ci servirà per fare in modo che il nodo si comporti differentemente in stato di allarme prima e dopo la riconfigurazione. A livello firmware è necessario agire principalmente nell'istruzione di switch successiva alla ricezione del messaggio. Vediamola in dettaglio nel Listato 2.

La registrazione dei valori avviene nel momento in cui il messaggio viene ricevuto estraendo i due byte dati. Nello switch, invece, si gestisce soltanto la situazione di allarme. Per permettere una verifica più semplice inviamo tramite la seriale dei messaggi di segnalazione.

Quando il nodo riceve un ID=121 la temperatura rilevata non supera il limite massimo e il filtro relativo è l'RXF0. Se riscaldiamo la sonda, ad un certo punto il nodo TX inizierà ad inviare messaggi con ID=123.

Ci troviamo cioè in uno stato di allarme ed il filtro relativo è RXF3. In questo caso non è stata effettuata ancora la riconfigurazione pertanto il nodo RX si limita a registrare il valore in EEPROM e ad inviare su seriale il messaggio "TEMP>LIM". I filtri RXF2 e RXF1 non vengono considerati finché il nodo di trasmissione non

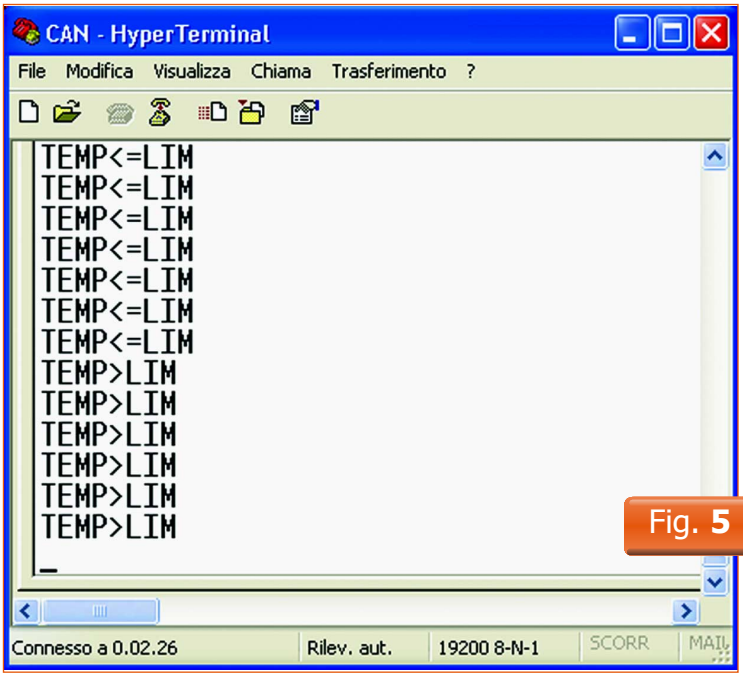


Fig. 5

EEPROM. In Figura 5 vediamo la sequenza visualizzata nella finestra dell'Hyper Terminal al raggiungimento della temperatura limite.

Che cosa accade nel momento in cui il nodo RX viene riconfigurato? Premiamo il pulsante che insiste sul pin RB1 della scheda di trasmissione.

La riconfigurazione avviene molto rapidamente tant'è che non ci si accorge neanche del blocco della ricezione da parte del nodo RX. Sullo schermo del PC vedrete comparire due stringhe (INI-CONFIG e FINE-CONFIG) che stabiliscono il momento in cui il nodo entra in configurazione e il momento in cui esso ritorna alla modalità normale (Figura 6).

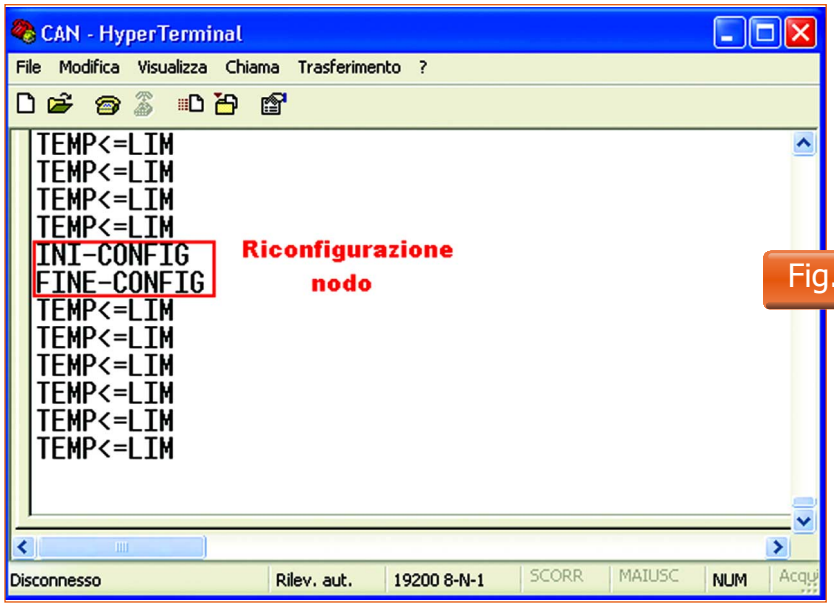


Fig. 6

invia un messaggio con ID=120. Inseriamo sul PIC di ciascun nodo il firmware distribuito sul sito della rivista. Ricordatevi che il nodo TX deve avere nei primi due byte della EEPROM il valore della temperatura limite secondo il formato DS18B20. Proviamo a connettere i due nodi utilizzando sempre il cavo con i soliti terminatori costruito durante il corso. Colleghiamo la porta seriale del nodo RX al PC e avviamo una sessione Hyper Terminal a 19200bps (8-N-1). Alimentando i due nodi, si deve accendere il led verde su entrambe le schede. Premiamo il pulsante su RB0 del nodo TX. Immediatamente il led rosso nelle due schede deve iniziare a lampeggiare segnalando che i messaggi vengono correttamente inviati dal TX al RX. Osserviamo i messaggi che compaiono sullo schermo del PC. Proviamo a riscaldare la sonda (raffreddatela o aumentate il limite se la temperatura ambiente è superiore a quella registrata nella EEPROM).

Ad un certo punto supereremo la soglia definita e il nodo TX inizierà ad inviare messaggi con ID=123. Il nodo RX non farà altro che trasmettere al PC la stringa "TEMP>LIM".

I valori continueranno ad essere registrati sulla

Se a questo punto proviamo a riscaldare nuovamente la sonda per portarla al limite massimo di temperatura prestabilito ci accorgeremo che il comportamento del nodo RX è cambiato. Anziché trasmettere soltanto la stringa TEMP>LIM come avveniva prima della riconfigurazione, ora invia la stringa "ATDT115" una sola volta accendendo anche il led giallo per tutta la durata dello stato di allarme. Abbiamo, in pratica, modificato una funzione durante l'operatività del nodo di ricezione senza accedere al codice relativo ma soltanto inviando un particolare messaggio sul bus. Vediamo in Figura 7 come si pre- >

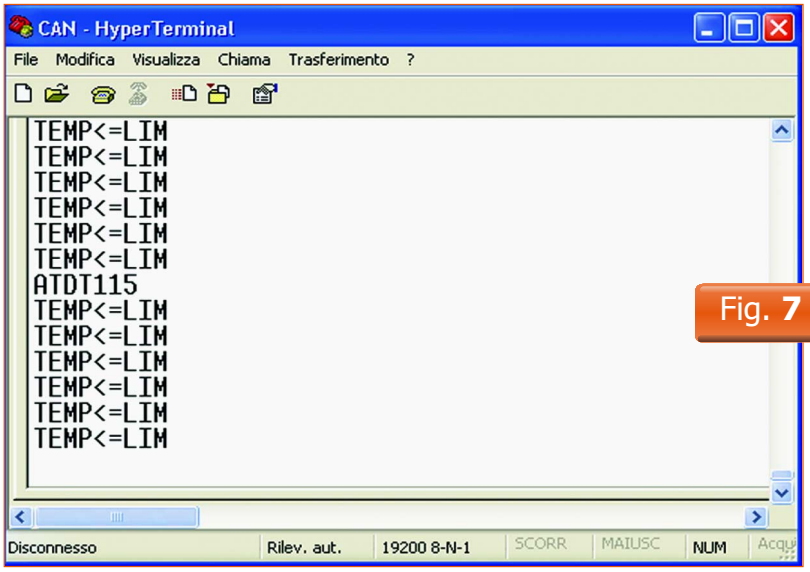


Fig. 7

parte del ECANPoll.c si determina il numero di buffer da utilizzare a seconda della modalità di funzionamento del nodo. Vengono letti i due bit più significativi del registro ECANCON per stabilire il modo operativo e sulla base del risultato si valorizza il limite "buffers".
Nel nostro caso si utilizza soltanto il MODE0, l'ECANCON non esiste ed il numero di buffer massimo è pari a 2. Pertanto la sezione considerata viene modificata così come si vede nel Listato 3.

Listato 3

```
#if ( ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME )
    mode = ECANCON&0xC0;  XXXX
    if ( mode == ECAN_MODE_0 )
        buffers = 2;
    else
        buffers = 8;
#endif
```

```
#if ( ECAN_LIB_MODE_VAL == ECAN_LIB_MODE_RUN_TIME )
        buffers = 2;
#endif
```

senta la sequenza relativa.
Si vede chiaramente come dopo aver superato la soglia il nodo invia la stringa con il comando Hayes ed attende continuando a registrare i valori ma senza generare ulteriori segnalazioni sulla seriale. Non appena la temperatura ritorna a valori normali, l'allarme rientra e nella sessione Hyper Terminal ricominciano a comparire i messaggi corrispondenti. Al termine dell'esperimento potete premere il pulsante che insiste su RB0 del nodo RX. Si accende il led verde e sullo schermo viene visualizzata la stringa "Fine". Poi potete passare al nodo TX ed effettuare la medesima procedura fino all'accensione del led verde. Naturalmente si può anche staccare direttamente l'alimentazione.

Modifiche libreria ECAN

Per far funzionare correttamente i due nodi ci siamo appoggiati sempre alla libreria ECAN (Polled) distribuita da Microchip. Abbiamo dovuto però operare alcune modifiche per evitare errori di compilazione per il riferimento a strutture implementate nella famiglia superiore di PIC e non nel 18F458 utilizzato per questo e gli altri esperimenti. Ad esempio in una prima

Considerando il ridotto numero di buffer, sono stati chiaramente eliminati tutti i riferimenti ai 6 buffer programmabili supplettivi presenti nella classe superiore.
Ad esempio la funzione ECANPointBuffer che deve ritornare il puntatore al primo byte del buffer di ricezione viene modificata mantenendo soltanto i riferimenti al RXB0CON e al RXB1CON che sono gli unici registri del 18F458. La modifica è apprezzabile nel riquadro del Listato 4.
Analogamente sono state eliminate strutture condizionali ed assegnazioni relative a strutture compatibili con il MODE1 e il MODE2. Ad esempio, laddove si verificava attraverso l'ECANCON la modalità operativa MODE0, si è inserita un'espressione sempre vera mentre lad-

Listato 4

```
static BYTE* _ECANPointBuffer(BYTE b)
{
    BYTE* pt;
    switch(b)
    {
    case 0:
        pt=(BYTE*)&RXB0CON;
        break;
    case 1:
        pt=(BYTE*)&RXB1CON;
        break;
    case 2:
        pt=(BYTE*)&B0CON;
        break;
    case 3:
        pt=(BYTE*)&B1CON;
        break;
    case 4:
        pt=(BYTE*)&B2CON;
        break;
    case 5:
        pt=(BYTE*)&B3CON;
        break;
    case 6:
        pt=(BYTE*)&B4CON;
        break;
    default:
        pt=(BYTE*)&B5CON;
        break;
    }
    return (pt);
}
```

```
static BYTE* _ECANPointBuffer(BYTE b)
{
    BYTE* pt;
    switch(b)
    {
    case 0:
        pt=(BYTE*)&RXB0CON;
        break;
    case 1:
        pt=(BYTE*)&RXB1CON;
        break;
    }
    return (pt);
}
```

dove si faceva riferimento a registri essenziali per altre modalità si sono eliminate le relative istruzioni.
Ad esempio il registro COMSTAT viene rimappato a seconda della modalità operativa. Nel 18F458 tale registro ha la struttura riportata in Figura 8 che nella classe superiore diventa come riportata in Figura 9.
Il bit7 ha un significato differente in MODE2 pertanto la sezione di codice relativa viene eliminata.
Infine, nelle dichiarazioni incluse nel file ECANPoll.h e relative alle funzioni ECANSetRXFnValue, abbiamo eliminato il riferimento ai registri RXFCONn per quanto riguarda il bit di attivazione RXFnEN proprio perché nel 18F458 i 6 filtri risultano essere già tutti attivi. Pertanto le funzioni diventano soltanto una ridefinizione della CANIDToRegs che abbiamo visto nella scorsa puntata e che serve soltanto ad estrarre correttamente i vari bit che compongono l'ID di un messaggio per trasferirli nei vari registri come la coppia RXF0SIDH/RXF0SIDL.
Le 6 funzioni diventano come rappresentato nel Listato 5 della pagina seguente.
Si vede chiaramente come la ridefinizione non faccia altro che richiamare la CANIDToRegs passando un puntatore ad un registro diverso a seconda del numero associato al filtro. Ad esempio per la ECANSetRXF0Value viene passato il puntatore a RXF0SIDH.
Sarà poi la CANIDToRegs ad occuparsi di incrementare l'indirizzo nel momento in cui dovrà accedere al byte inferiore RXF0SIDL. Per il dettaglio riferitevi al riquardo corrispondente pubblicato nel capitolo 8. >

COMSTAT: COMMUNICATION STATUS REGISTER

Fig. 8

R/C-0	R/C-0	R-0	R-0	R-0	R-0	R-0	R-0
RXB0OVFL	RXB0OVFL	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN
bit 7							bit 0

Fig. 9

COMSTAT: COMMUNICATION STATUS REGISTER

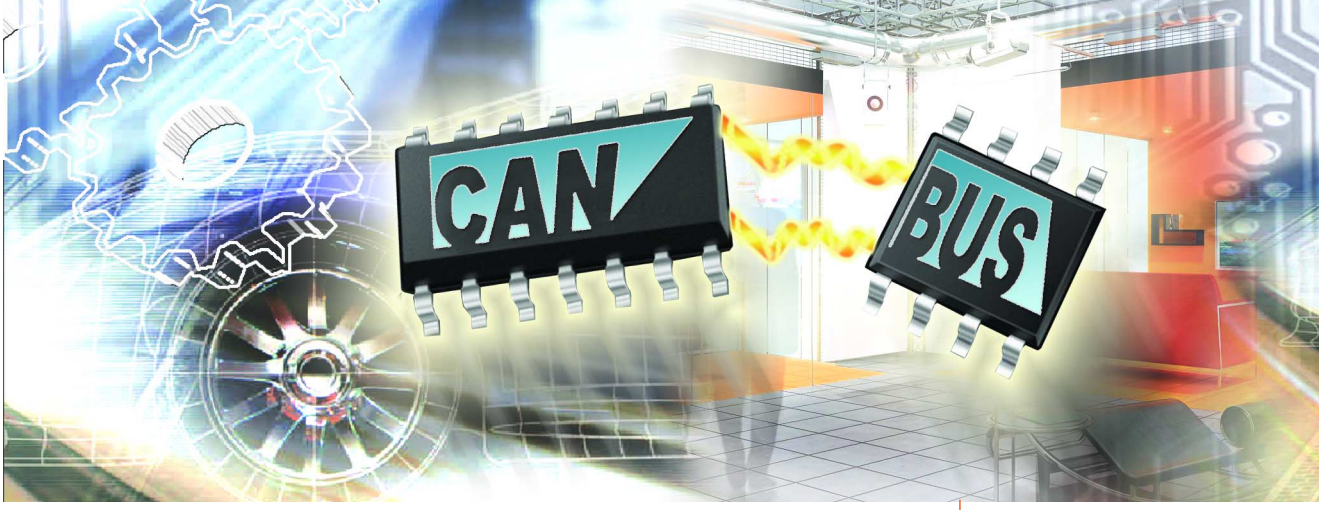
	R/C-0	R/C-0	R-0	R-0	R-0	R-0	R-0	R-0
Mode 0	RXB0OVFL	RXB0OVFL	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN
	U-0	R/C-0	R-0	R-0	R-0	R-0	R-0	R-0
Mode 1	-	RXBnOVFL	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN
	R/C-0	R/C-0	R-0	R-0	R-0	R-0	R-0	R-0
Mode 2	FIFOEMPTY	RXBnOVFL	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN

Conclusioni

Siamo giunti al termine del nostro percorso nel mondo della programmazione firmware per l'interfacciamento su CAN bus. Abbiamo tentato di rendere il discorso quanto più interessante possibile partendo dalle semplici funzioni di invio e ricezione dei messaggi, passando attraverso la configurazione di filtri e maschere ed arrivando alla riconfigurazione di un nodo a run-time.
La panoramica è completa e permette a tutti voi di intraprendere la progettazione di circuiti in grado di dialogare in maniera efficiente attraverso un bus CAN. Naturalmente questo non significa che l'apprendimento di tecniche di sviluppo e di progettazione in questo settore finisca qui. Questo corso come quello sull'USB rappresenta una base fondamentale di conoscenza comune che ci permette di dialogare con voi in maniera più efficace ed immediata. Da questo punto in poi dobbiamo sentirci liberi di parlare di ID, maschere, filtri, modalità operative essendo certi che il nostro discorso verrà compreso senza ulteriori spiegazioni. Per iniziare, nel prossimo numero presenteremo uno strumento che troverete molto utile visto che vi permetterà di registrare su PC i vari messaggi inviati su un bus CAN e di inviare dal PC messaggi singoli o in sequenze determinate da appositi script. Lo stesso sistema vi permetterà anche di indagare a fondo sul funzionamento dei filtri e dei vari registri del modulo CAN integrato nel PIC vedendo dal vivo come avviene la riconfigurazione di un nodo operativo.
Il tutto verrà fatto direttamente sulla demoboard utilizzata nel corso. Non mi resta che darvi appuntamento al prossimo mese.

Listato 5

```
#define ECANSetRXF0Value(val, type)  _CANIDToRegs((BYTE*)&RXF0SIDH, val, type)
#define ECANSetRXF1Value(val, type)  _CANIDToRegs((BYTE*)&RXF1SIDH, val, type);
#define ECANSetRXF2Value(val, type)  _CANIDToRegs((BYTE*)&RXF2SIDH, val, type);
#define ECANSetRXF3Value(val, type)  _CANIDToRegs((BYTE*)&RXF3SIDH, val, type);
#define ECANSetRXF4Value(val, type)  _CANIDToRegs((BYTE*)&RXF4SIDH, val, type);
#define ECANSetRXF5Value(val, type)  _CANIDToRegs((BYTE*)&RXF5SIDH, val, type);
```



Corso di programmazione: CAN BUS

a cura di Carlo Tauraso

10

Nato come protocollo di comunicazione seriale per fare colloquiare tra loro tutti i sistemi elettronici presenti a bordo delle autovetture, si sta affermando anche nell'automazione industriale e nella domotica. In questa e nella puntata successiva presentiamo un sistema in grado di registrare in real-time i messaggi scambiati tra due nodi.

A conclusione di questo corso (e prima di presentare alcuni progetti pratici) abbiamo ritenuto opportuno dedicare altre due puntate alla pubblicazione di un interessante sistema in grado di registrare in real-time i messaggi scambiati tra due nodi. La cosa interessante è che non sarà necessario effettuare alcuna modifica hardware alla demoboard utilizzata nel corso, semplicemente dovremo sostituire il firmware del PIC con il *monitorCAN.hex* scaricabile gratuitamente dal sito della rivista. Ne è venuto fuori un progetto molto interessante che rappresenta uno strumento di diagnostica completo per tutti quei sistemi che utilizzano il CAN Bus, ad iniziare da quelli di cui ci siamo occupati nel corso. Dal punto di vista software ci siamo appoggiati al CANKing, un CAN Monitor Freeware distribuito dall'azienda svedese Kvaser (*www.kvaser.com*). Dal lato software abbiamo creato dei template che permettono di configurare tale programma in modo che possa funzionare con i nodi sviluppati durante il corso dando la possibilità non solo di monitorare i messaggi ma anche di entrare in Configuration mode a runtime, modificando i registri del modulo CAN in modo da vederli in funzione in diretta. Sarà così possibile, quindi, sperimentare in maniera approfondita tutti gli argomenti trattati durante le varie puntate del corso senza lasciare nulla al caso. Per quanto riguarda il firmware è stato necessario creare un codice che risultasse compatibile sia con i comandi inviati via RS-232 dal software che con la configurazione hardware della demoboard. Il sistema rappresenta un "banco di prova" indispensabile per tutti coloro che vogliono effettuare dei test approfonditi sui propri prototipi. Molto spesso, infatti, quando si ha a che fare con diversi firmware che dialogano attraver- >

so una struttura di rete è fondamentale capire che cosa avviene lungo il canale di comunicazione.
Ne sanno qualcosa coloro che per mestiere o semplice passione hanno avuto a che fare con lo sviluppo software/firmware in ambiente Ethernet: quante volte un buon monitor ha permesso di risolvere un problema di comunicazione consentendo di identificare il terminale o il processo che lo ha determinato. Vedere i messaggi che transitano sul bus di una rete CAN può essere estremamente utile ed interessante, sia per lo sviluppo del firmware da inserire sui vari nodi che per comprendere a fondo come avviene la comunicazione. Se poi volessimo tentare delle sperimentazioni particolari quali, ad esempio, l'elaborazione parallela, il sistema diventerebbe fondamentale per capire se la suddivisione dei compiti tra i vari nodi avviene correttamente. Attraverso il template denominato "Registri" da noi predisposto si può indagare sul funzionamento dei registri del modulo CAN vedendo come l'invio di un messaggio con ID predefinito venga intercettato attraverso un filtro. In questa puntata proveremo a monitorare i messaggi inviati dal nodo RX utilizzando lo stesso firmware del secondo esperimento del corso. Vedremo chiaramente l'aumento di temperatura nei messaggi trasmessi fino al cambiamento dell'ID quando si supera la soglia stabilita. Iniziamo, dunque, con la procedura di installazione del CANKing.

CANKing: la procedura d'installazione

Il pacchetto autoestraente CANKingPIC.exe è scaricabile gratuitamente dal sito della rivista. Facendo doppio clic su di esso si avvia una procedura di installazione molto semplice. Il file Readme proposto contiene alcune informazioni sull'aggiornamento del browser Internet Explorer dovuto all'utilizzo della COMCTL4.7 e di un help basato su HTML. Diciamo subito che potete anche saltare questo passo visto che basta aver installato l'IE 4.0 per avere già tutto ciò che serve. Dopo aver accettato la licenza di utilizzo viene visualizzata la directory di installazione. Consigliamo di mantenere quella proposta dal software; nel caso sia necessario modificarla, ricordatevi di farlo anche quando effettuate l'installazione dei Template. Dopo aver fatto nuovamente clic sul pulsante "Avanti", viene visualizzato un form che permette di scegliere quali parti dell'archivio installare. Eliminate la spunta sia sull'opzione COMCTL che su quella relativa all'HTML Help file system in maniera da effettuare l'installazione del programma senza ulteriori aggiornamenti del sistema (Figura 1). A questo punto è possibile scegliere il gruppo di programmi nel quale verrà inserito il link per avviare il software. Anche in questo caso consigliamo di lasciare l'opzione proposta. Viene avviata la copia dei file al termine della quale fate clic sul pulsante "Finish". Viene quindi creata una voce di menu chiamata CANKing sotto Start->Programmi->Microchip. La struttura software è adesso pronta per essere configurata. Ci sono diverse parti da modificare per far funzionare correttamente il sistema con i nodi utilizzati durante le varie fasi

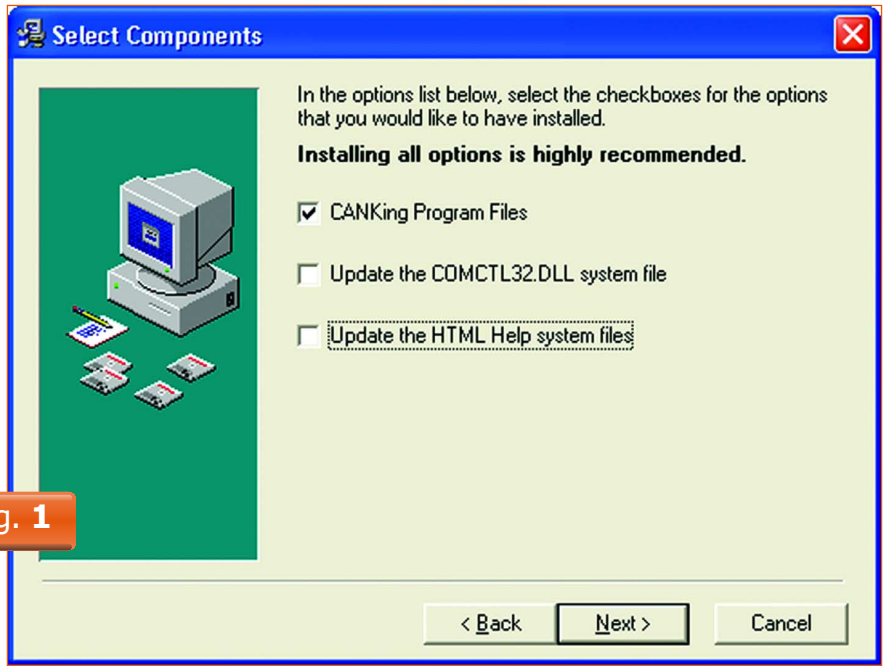


Fig. 1

del corso. Ci limiteremo a descrivere quelle essenziali per prendere confidenza con l'ambiente operativo in modo da poterlo personalizzare facilmente. Per rendere le cose più semplici anche a quanti vogliono essere immediatamente operativi, abbiamo predisposto un ulteriore file autoestraente che effettua automaticamente tutte le configurazioni creando due modelli di funzionamento che permettono sia di usare il software come Monitor che di analizzare il funzionamento dei registri CAN del PIC18F452.

Template Corso CAN: la procedura d'installazione

Dopo aver effettuato l'installazione del CANKing, passiamo alla sua integrazione con i Template predisposti. Anche in questo caso l'operazione è molto semplice. Dopo aver fatto doppio clic sul file CANTemp.exe, viene proposto un primo menu nel quale è possibile selezionare il linguaggio utilizzato nella procedura. Avviata l'installazione viene visualizzato un avviso relativo al fatto che il programma rappresenta un'integrazione di un software pre-esistente che deve essere già presente sul PC. Viene, quindi proposta la directory d'installazione (Figura 2). Lasciate il percorso proposto o modificatelo attraverso il pulsante "Seleziona" posizionandovi sulla cartella Templates della directory utilizzata per l'installazione del CANKing. Facendo clic su "Continua" viene avviata la copia dei file. Al termine viene presentata una finestra di avviso: fate clic su OK per uscire (Figura 3). Prendiamo ora il nodo RX utilizzato durante il corso ed aggiorniamo il firmware contenuto nella sua memoria flash attraverso il CANmon.hex, anch'esso scaricabile gratuitamente dal sito della rivista. Il file binario della memoria EEPROM non è da modificare.

Fig. 4

Fig. 2

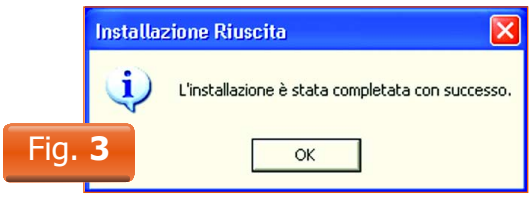


Fig. 3

CANKing: uso

Avviate il software. La prima volta viene visualizzata una finestra attraverso la quale è possibile accedere all'help online ed alle informazioni di installazione. Selezionate la voce "Start using CANKing" e "Don't ask me again" per evitare che il form venga riproposto ogni volta che avviate il programma. Fate clic su OK. Viene visualizzato un messaggio per avvisare l'utente che l'utilizzo del software di monitoraggio su un sistema CAN reale può portare a dei malfunzionamenti che in alcuni casi possono avere conseguenze piuttosto gravi (Figura 5). Se la cosa può sembrare eccessiva, non bisogna dimenticare che molto spesso il CAN Bus viene utilizzato per sistemi piuttosto critici, sia in ambito industriale che automobilistico. Pur essendo la sicurezza degli utenti molto importante, è evidente che nel nostro caso potete tranquillamente selezionare la voce "Don't show this warning in the future" e fare clic su "OK, I know what I'm doing". Noi, infatti, utilizzeremo il sistema solo in un ambiente sperimentale. Il warning viene definitivamente disattivato premendo il tasto F7. Viene quindi visualizzata una nuova >

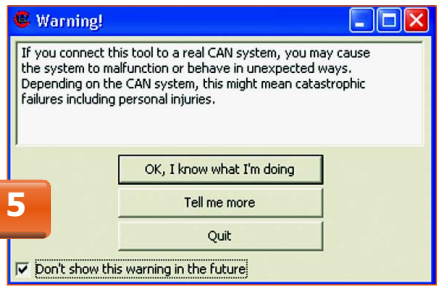


Fig. 5

finestra che permette di aprire un Template o un progetto salvato: scegliete la seconda opzione come indicato in Figura 6. Verrà visualizzato un elenco di modelli disponibili il primo dei quali, denominato TX/RX, permette di monitorare i messaggi trasferiti sul bus. Il secondo, invece, permette di effettuare un debug dei registri del PIC18F458 dedicati al modulo CAN, permettendo di comprenderne a fondo il funzionamento (Figura 7). Prima di aprire il primo modello predisponete i due nodi utilizzati durante il corso. Inserite nel nodo TX il firmware distribuito col capitolo 6 (quello relativo al secondo esperimento) e collegate il nodo TX a quello RX precedentemente aggiornato mediante il solito cavo provvisto di terminatori. Collegate quindi la seriale del nodo RX ad una porta disponibile sul vostro PC. Alimentate il nodo RX che d'ora in poi chiameremo nodo monitor; se tutto funziona correttamente debbono accendersi sia il led giallo che quello verde. Il sistema è così pronto a intercettare i messaggi e a ricevere i comandi da PC. Selezionando il modello TX/RX si apriranno una serie di finestre ognuna delle quali ha una funzione particolare. Selezioniamo innanzitutto la porta seriale attraverso la quale il nodo comunica con il PC; la selezione avviene attraverso la voce di menu Options->PIC18+CAN... come si vede in Figura 8.

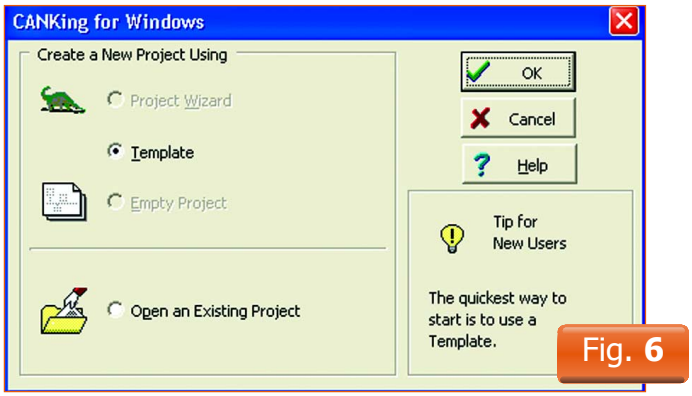


Fig. 6

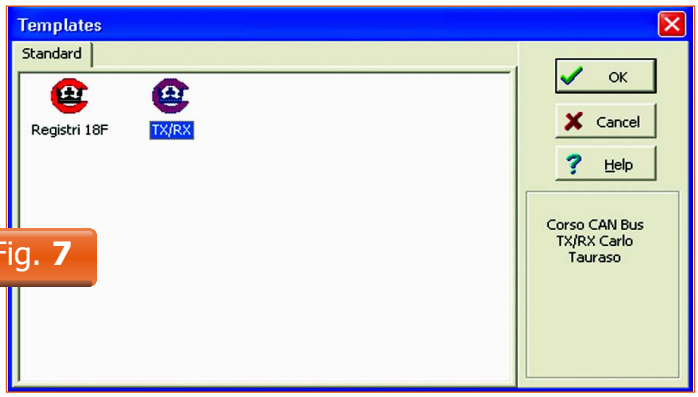


Fig. 7

La finestra relativa presenta alcuni comandi attraverso i quali è possibile modificare la frequenza dell'oscillatore che controlla il clock del PIC, variare la velocità di comunicazione della seriale, cambiare la porta in uso ed effettuare il reset della card all'avvio. Selezionate una porta libera impostando la frequenza a **20MHz** e la velocità a **38.400bps** (Figura 9): dopo aver fatto clic su "OK" ed aver applicato la modifica vedrete un cambiamento nella finestra "Evaluation Board" che aggiorna lo stato del nodo. Quest'ultimo risulta correttamente collegato al PC (Connected) ed attende

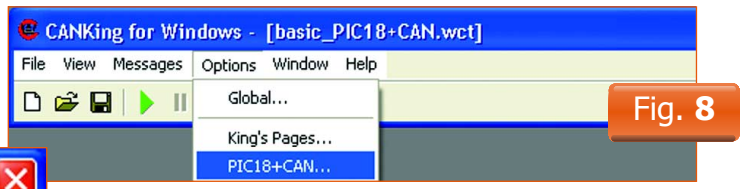


Fig. 8

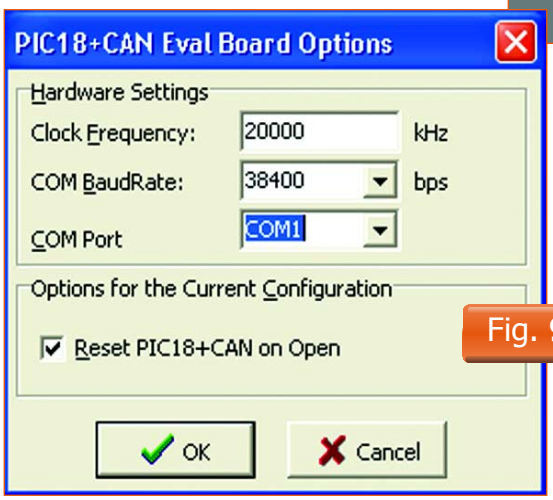


Fig. 9

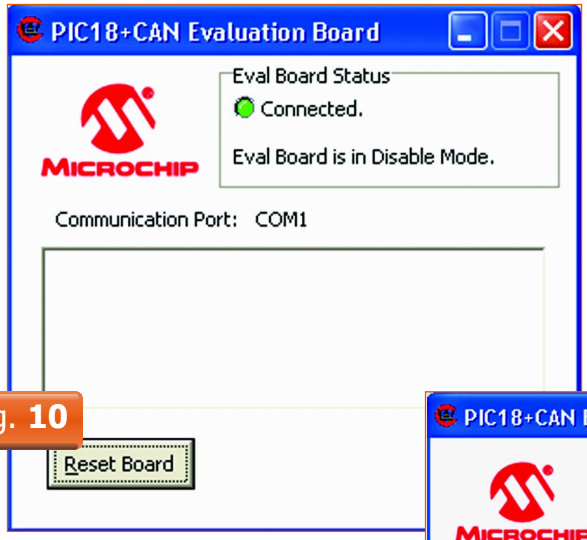


Fig. 10

il collegamento con il bus (Disable Mode) come mostrato in Figura 10. Una volta selezionata la porta seriale ed applicata la modifica sarà possibile avviare il software anche con il nodo Monitor non alimentato. La finestra "Evaluation Board" presenterà uno stato "Not Present". Alimentate il nodo Monitor: vedrete comparire una stringa di avviso nel pannello sottostante ("Avvio nodo MONITOR Corso CAN...") come si vede in Figura 11. Fate clic sul pulsante "Reset Board" e lo stato passerà a Connected/Normal Mode.

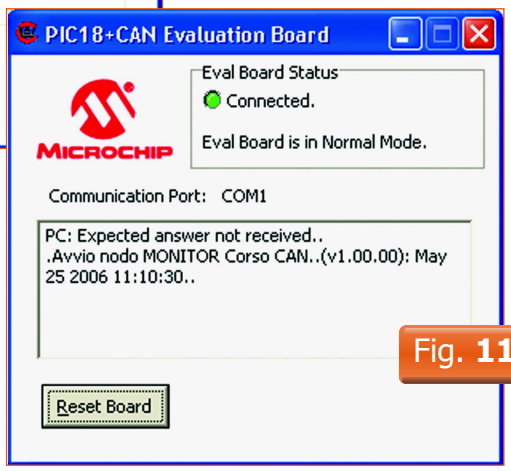


Fig. 11

Nodo TX: sniffing dei messaggi

Siamo ora pronti a vedere come si effettua realmente un monitoraggio dei messaggi. Il modello selezionato si occupa di predisporre tutte le configurazioni necessarie sia per quanto riguarda i parametri di comunicazione del bus che per la formattazione dei risultati. Alimentiamo il nodo TX.

Inizialmente si accende il led verde e il nodo attende la pressione dello switch che controlla la RB0.

Posizioniamoci sulla finestra "CAN Controller" e facciamo clic sul pulsante "Go On Bus". Il led verde posto sul pannello si accende a significare che il sistema è pronto. Premete il pulsante collegato a RB0 del nodo TX: vedrete lampeggiare il led rosso ed immediatamente si animeranno le finestre "CAN Controller" e "Output Window". I messaggi ricevuti vengono registrati sequenzialmente nella finestra "Output Window" mentre nell'altra vedrete comparire le statistiche relative come riportato in Figura 12. Le informazioni vengono registrate sequenzialmente riportando i valori in esadecimale.

Provate a riscaldare (o raffreddare a seconda della temperatura ambiente) la sonda termica del nodo TX fino a portarla al limite predefinito. Nell'esperimento avevamo utilizzato un valore in EEPROM pari a 0190h che corrisponde a circa 23°C. Vedrete ad un certo punto la variazione da parte del nodo TX dell'ID dei messaggi inviati col valore che passa da 121h a 123h segnalando così lo stato di allarme. La sequenza relativa è visibile in Figura 13. Il formato di registrazione è molto semplice: la prima colonna riporta l'ID del messaggio, la seconda la lunghezza dei dati trasmessi, le successive i valori dei byte. Le ultime due colonne riportano un riferimento temporale e la direzione del messaggio (Ricezione o Trasmissione). Potete provare a monitorare i messaggi per tutti gli esperimenti che trovate nel corso, verificando il reale funzionamento dei nodi utilizzati.

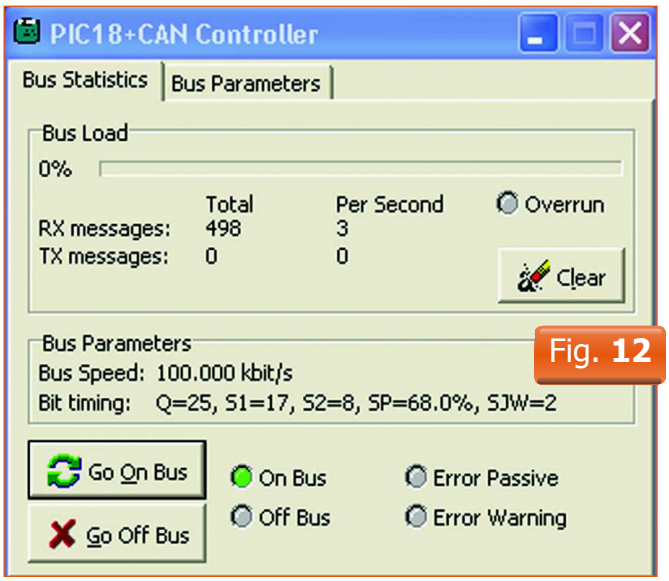


Fig. 12

CANKing: personalizzazioni

Il software in questione si presta a diverse personalizzazioni che possono >

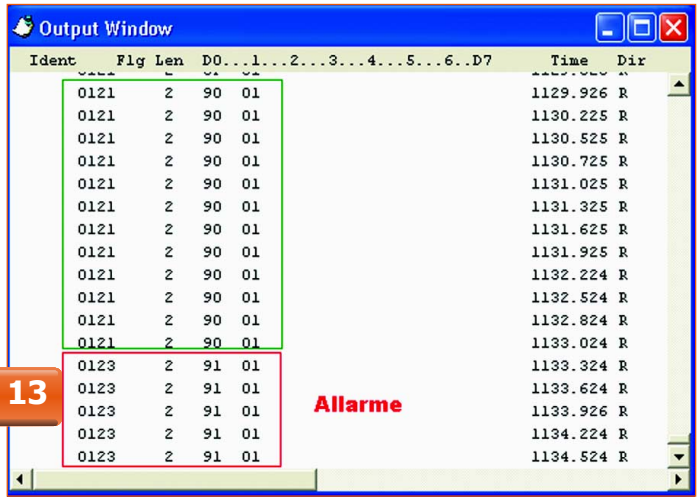


Fig. 13

quello del nodo RX presentato nel capitolo 6. Utilizzando CTRL+U o la voce Messages-> Universal-> Universal apparirà una finestra all'interno della quale potrete inserire il campo ID, la lunghezza dei dati (DLC) ed i valori da inviare. In Figura 16 si vede un tipico messaggio relativo ad una acquisizione con temperatura nella norma (ricordatevi che in questo caso i campi sono decimali quindi = 121h = 289d). Ogni volta che fate clic sul pulsante "Send" viene immesso sul bus il messaggio relativo come si vede chiaramente nella "Output Window" di Figura 17. Ma la cosa più interessante è senz'altro la possibilità di raccogliere una sequenza di messaggi in uno script da inviare tutto assieme, scegliendo anche la temporizzazione con cui tali messaggi vengono trasmessi. Questo fatto permette di simulare un ambiente CAN molto più realistico. Ogni volta che inviamo un messaggio esso viene salvato nella cosiddetta History List. Per visualizzarla è sufficiente usare la voce di menu View->History List; in Figura 18 vediamo come si presenta tale lista dopo aver inviato per dieci volte consecutive il messaggio singolo visto nel paragrafo precedente. La sequenza può essere salvata in uno

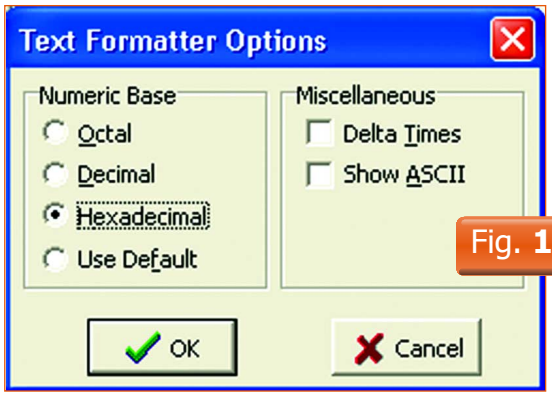


Fig. 14

poi essere salvate in appositi file progetto. Ad esempio è possibile modificare la formattazione dei risultati ottenuti attraverso la finestra "Select Formatters". Basta selezionare uno dei formati disponibili e fare clic sul pulsante "Use". Selezionando la voce nel pannello "Active Formatters" è possibile modificare anche alcune opzioni di formattazione (se disponibili) come, ad esempio, il salvataggio in esadecimale o decimale dei valori rilevati (vedi Figura 14). Analogamente risulta fattibile la modifica dei parametri di funzionamento del bus attraverso la pad "Bus Parameters" presente sul form "CAN Controller" (Figura 15). Naturalmente bisogna prestare molta attenzione a quello che si sta facendo perché se non si modificano anche i parametri di funzionamento (a livello firmware) dei diversi nodi utilizzati, il monitoraggio non può avvenire correttamente. A quanti vogliono intraprendere personalizzazioni particolari segnaliamo che è possibile effettuare delle modifiche direttamente nella struttura XML dei file .wct che contengono tutti i dati utilizzati nei Template. Lo si può fare facilmente utilizzando un piccolo applicativo chiamato treedit.exe che viene installato nella stessa directory di CANKing.

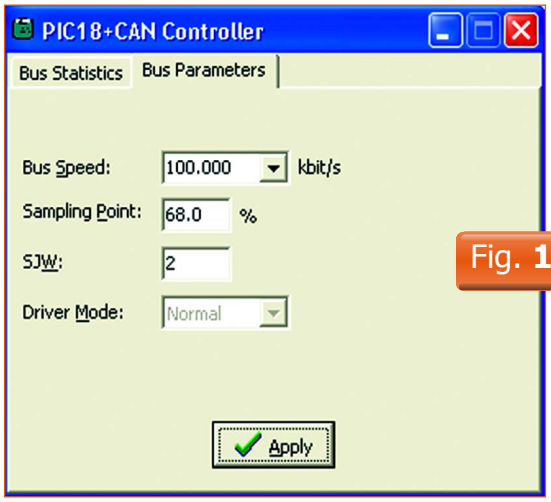


Fig. 15

CANKing: iniettare messaggi

Oltre a permettere il monitoraggio, il sistema è anche in grado di immettere sul bus un messaggio o sequenze di messaggi. Ad esempio l'invio singolo può avvenire attraverso la combinazione CTRL+U che prevede una struttura universale. Provate a cambiare il firmware del nodo TX con

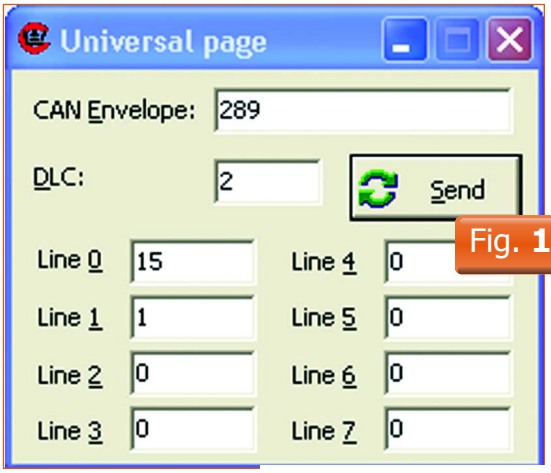


Fig. 16

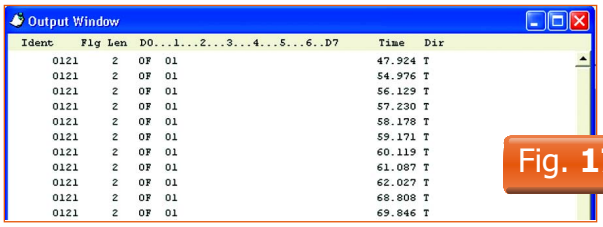


Fig. 17

script con estensione .hst che potrà essere riutilizzato in altre occasioni; potremo inviare l'intera sequenza o singoli messaggi da selezionare facendo clic sulla riga relativa. È possibile scegliere la temporizzazione da utilizzare visualizzando la finestra "Timed Transmission" attraverso la voce di menù View -> Timed Transmission.

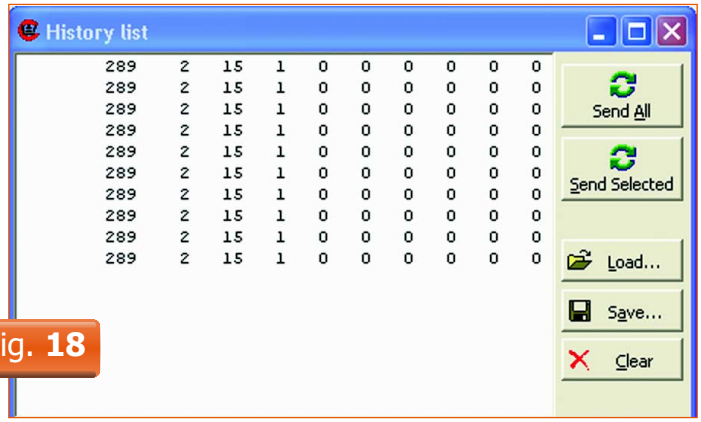


Fig. 18

Come si vede in Figura 19, attraverso il form relativo è possibile definire l'intervallo di tempo tra un messaggio e l'altro definendo se la trasmissione deve avvenire una sola volta (One shot) oppure ripetuta ciclicamente. Una volta stabilito il modo con cui vogliamo effettuare la trasmissione, possiamo fare clic sul pulsante "Send All". Il processo può essere controllato attraverso la "Output Window" e fermato e riavviato attraverso i pulsanti "Send", "Pause", "Stop" presenti sul form "Timed Transmission". Nel momento in cui si avvia la trasmissione vedremo lampeggiare il led rosso del nodo di ricezione così come avevamo impostato nel firmware del secondo esperimento. È del tutto evidente che la demoboard assieme al CANKing rappresenta un sistema didattico e di testing veramente completo, permettendo sia la diagnostica che la sperimentazione. Il nostro spazio per questo numero è terminato ma il CANKing con i nostri Template ha ancora diverse funzionalità interessanti. Il mese prossimo vedremo come utilizzare il secondo modello "Registri 18F" per studiare il funzionamento dei registri del modulo CAN integrato nel 18F458 sperimentando la riconfigurazione a runtime vista nell'ultimo capitolo del corso.

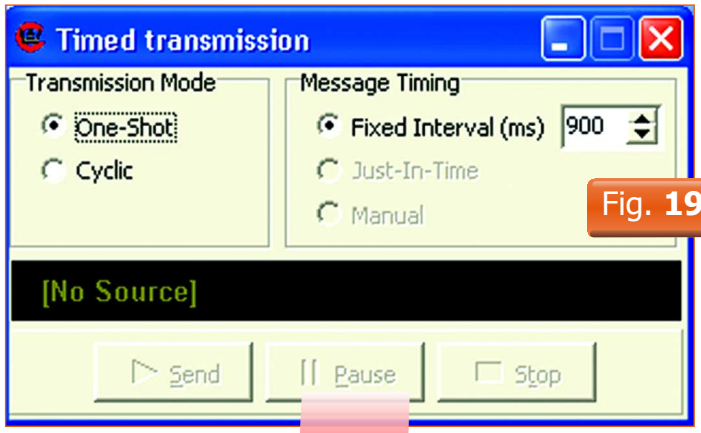


Fig. 19

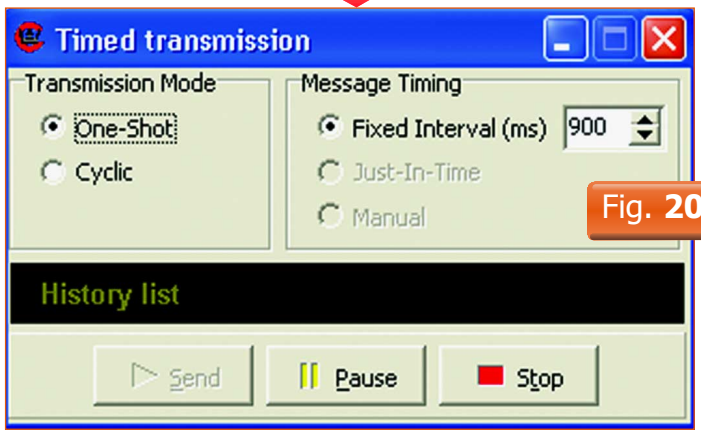
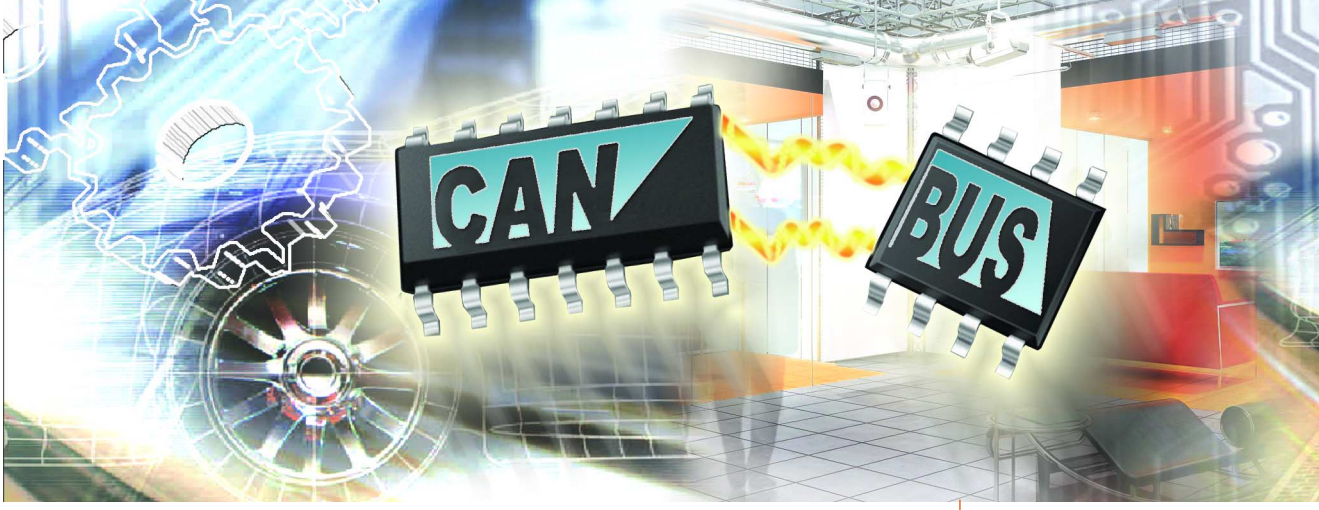


Fig. 20



Corso di programmazione: CAN BUS

a cura di Carlo Tauraso

Ultima puntata del Corso dedicato al protocollo CAN; in queste pagine concludiamo l'analisi del funzionamento dei registri del modulo CAN integrato nel PIC18F458 utilizzando sempre il software CANKing. Dal prossimo numero, terminata l'analisi teorica, proporremo delle applicazioni pratiche basate sui moduli Velbus.

11

Nella scorsa puntata abbiamo visto come sia possibile registrare ed iniettare messaggi su un bus CAN attraverso il CANKing e la nostra demoboard. Ora concludiamo questa ultima puntata utilizzando il secondo modello che abbiamo preparato per indagare il funzionamento dei registri inclusi nel modulo CAN integrato nel PIC18F458. Colleghiamo il nodo Monitor alla seriale del PC. Potete alimentare direttamente il nodo prima o dopo l'avvio del software. Ricordiamo che il firmware da inserire nel nodo Monitor è direttamente scaricabile dal sito della rivista *www.elettronicain.it* assieme al software CANKing e ai template. L'installazione dei relativi pacchetti è stata descritta nello scorso numero in maniera dettagliata. Per quanto riguarda il nodo di trasmissione ci riferiamo al codice scaricabile dal sito della rivista. Si tratta di una versione modificata di quello presentato nel capitolo 6 che prevedeva l'invio di messaggi con ID differente a seconda che la temperatura fosse inferiore o superiore ad un limite prestabilito.

CANKing: accedere ai registri

Avviamo il CANKing e scegliamo tra i template quello denominato "Registri 18F" come appare evidenziato in Figura 1.

Dopo aver fatto clic sul pulsante OK vedrete comparire un bel po' di finestre, ognuna delle quali permette di visualizzare un ben determinato gruppo di registri. Per fare delle prove specifiche ricordatevi che è sempre possibile scegliere i form da visualizzare attraverso il menù "View" dove abbiamo riparato l'elenco delle funzioni attive. Facendo clic a sinistra della voce relativa è possibile attivare o disattivare la visualizzazione della finestra corrispondente. Osservate con attenzione l'immagine di Figura 2. Prima di ana- >

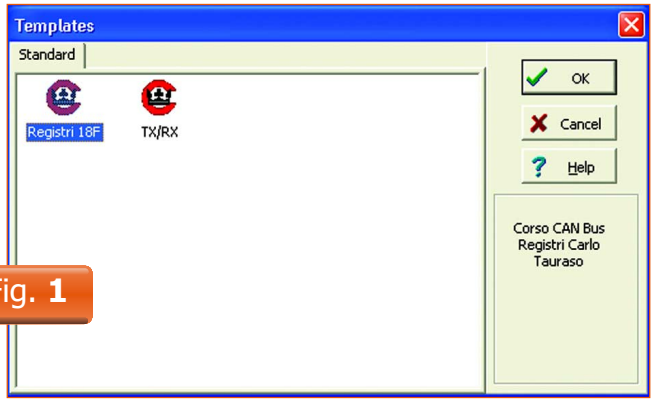


Fig. 1

pannello sottostante ("Avvio nodo Monitor CAN"), fate clic sul pulsante "Reset Board" e la scheda passa allo stato Connected/Normal Mode. Quest'ultima è senz'altro la procedura più indicata. Sul form relativo ci sono due nuovi pulsanti: Load Reg e Save Reg. Essi permettono di salvare o di caricare i valori di tutti i registri del modulo CAN visibili nelle varie finestre. In questo modo è possibile simulare una precisa condizione del modulo indagando sul suo funzionamento.

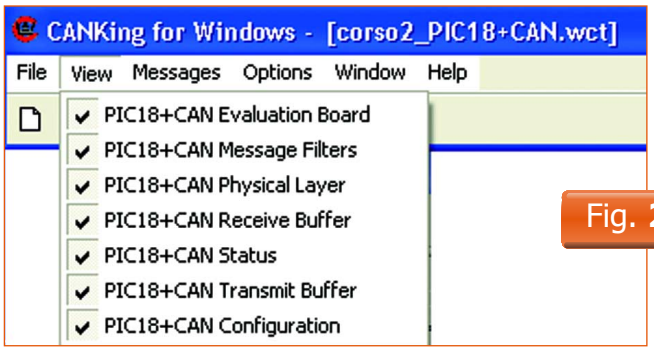


Fig. 2

La prima finestra da considerare è la "**PIC18+CAN Physical Layer**" che permette di controllare i tre registri BRGCON1, BRGCON2, BRGCON3. Essi rendono possibile la configurazione dei parametri che regolano la comunicazione sul bus come il Prescaler, Propagation Segment, Phase Segment, Syncronization ecc. Se fate clic su Read, vedrete comparire i valori impostati per il modulo CAN della scheda Monitor. Nella barra di stato viene visualizzato il Bit Rate e la frequenza di clock applicata al microcontrollore. Fate attenzione, nella griglia tutti i bit evidenziati in grigio si possono soltanto leggere, quelli che presentano un trattino non sono implementati, mentre soltanto quelli con sfondo bianco possono essere sovrascritti. Potete scrivere direttamente un 1 o uno 0 oppure potete fare doppio clic sul bit per farlo variare da 0 a 1 e viceversa. Facendo clic sulla

lizzare ciascuna funzionalità separatamente, configuriamo i parametri di comunicazione attraverso la voce di menù Options->PIC18+CAN. Selezionate la porta seriale da utilizzare, la velocità a 38.400bps e la frequenza di clock a 20MHz. Questi parametri vengono salvati all'interno del registry di Windows, pertanto, la prima volta, bisogna precisarli, poi, nelle successive occasioni, verranno ricaricati automaticamente. Dopo aver effettuato la modifica si consiglia comunque di riavviare il programma. La form di configurazione è visibile in Figura 3.

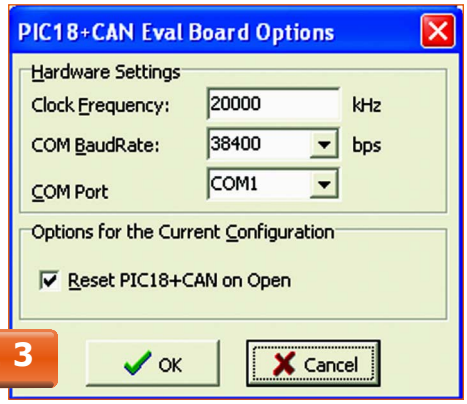


Fig. 3

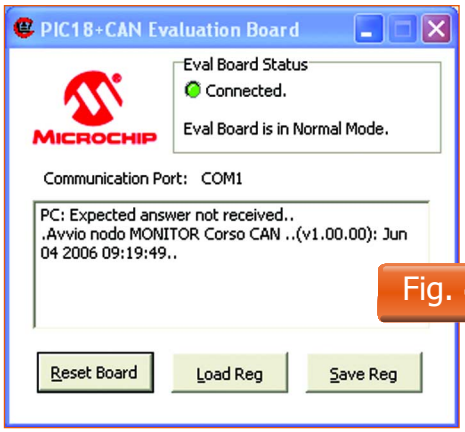
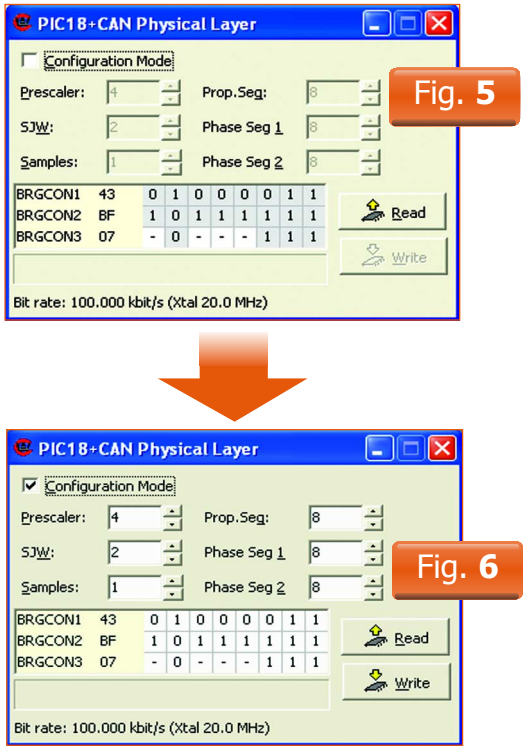


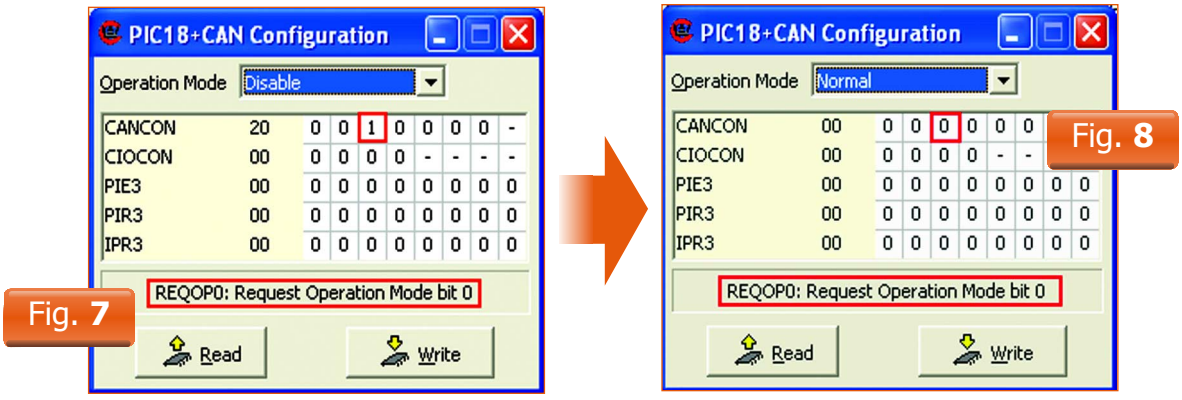
Fig. 4

Posizionatevi sulla finestra "Evaluation Board" che abbiamo considerato anche nel precedente numero. Se avete avviato il software con scheda alimentata verrà visualizzato uno stato del tipo: Connected/Disable Mode. Se, invece, avete avviato il software con scheda non alimentata lo stato è: Eval Board not found/Answer not received. A questo punto collegatela all'alimentatore, vedrete comparire un messaggio di avvio nel



check box "Configuration Mode" è possibile forzare la modalità di configurazione. Le operazioni di trasmissione e ricezione vengono bloccate ed è possibile agire sui registri. La griglia, infatti, diventa bianca.
I valori possono essere scritti sia per singoli bit, sia accedendo ai campi soprastanti che li raggruppano nei vari parametri che abbiamo già descritto dettagliatamente durante il Corso. La procedura di configurazione, è illustrata dalle schermate riportate nelle Figure 5 e 6.

gliendolo direttamente tra quelli disponibili. Desideriamo farvi notare che se posizionate il cursore su uno qualsiasi dei bit che compongono i registri vedrete, nell'etichetta sottostante, la sua denominazione (questo vale per tutte le griglie presenti nelle varie finestre). In questo modo è possibile vedere direttamente la struttura dei registri e verificare il funzionamento di ciascun bit. Se avete avviato il software senza aver alimentato la scheda, il modulo si trova normalmente in "Disable Mode". Da qui potete entrare direttamente in "Normal Mode" selezionando la voce relativa. Ricordatevi che per effettuare la modifica reale dei registri è necessario fare clic sul pulsante "Write". Vedrete come vengono modificati i 3 bit più significativi del nodo CANCON (Request Operation Mode 0-1-2). Contemporaneamente, nella finestra "Evaluation Board", lo stato della scheda viene aggiornato passando, dalla modalità "Disable" a "Normal".
Nelle immagini seguenti mostriamo come avviene il passaggio: in rosso appaiono evidenziati il bit modificato e la sua denominazione (Figura 7 e 8). Una volta precisata la configurazione dei registri che stabiliscono la comunicazione sul bus e la modalità operativa del nodo possiamo spostarci alle finestre operative e di monitoraggio. Innanzitutto consideriamo la "**PIC18 + CAN Status"**. È una finestra di monitoraggio che permette di controllare lo status del modulo CAN integrato nel PIC.
I registri CANSTAT (CAN Status Register) e COMSTAT (Communication Status Register)



Un'altra finestra importante per il funzionamento corretto del nodo è la **"PIC18+CAN Configuration"**. Essa permette di visualizzare il contenuto dei registri di configurazione del modulo CAN come CANCON. Attraverso la list box "Operation Mode", è possibile impostare la modalità di funzionamento del modulo CAN sce-

riportano rispettivamente i bit inerenti la modalità operativa e la presenza di errori in trasmissione o ricezione con i relativi contatori. Potete provare a collegare il nodo di trasmissione effettuando un po' di letture per vedere se si verificano degli errori. Normalmente il form deve apparire come riportato in Figura 9. Ogni volta che >

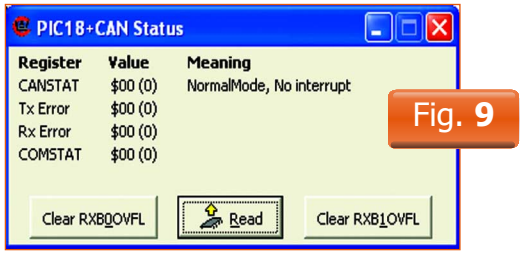


Fig. 9

eseguite una lettura viene aggiornata anche la finestra relativa ai buffer di ricezione che vedremo nel prossimo paragrafo.
I due pulsanti laterali RXB0OVFL e RXB1OVFL permettono di azzerare i flag di overflow relativi ai due buffer di ricezione. Sicuramente vi ricordate che tale condizione si verifica quando non si scaricano i messaggi in ricezione. Si tratta, quindi, di una condizione molto semplice da simulare. Per rendere la cosa più facile abbiamo provveduto a modificare il firmware del nodo TX in maniera tale che ogni messaggio venga inviato alla pressione del tasto su RB0.
Quindi: alimentate il nodo TX, attendete lo spegnimento del led verde, successivamente premete il tasto RB0 ogni volta che volete effettuare l'invio di un messaggio al nodo Monitor. La trasmissione viene segnalata dall'accensione e dallo spegnimento del led rosso.
È stato necessario inserire un controllo in più ed un ritardo per evitare il "debounce". In questo modo siamo sicuri che venga inviato un solo messaggio alla volta evitando trasmissioni multi-

Listato 1

```
while (1)
     {
      while (PORTBbits.RB0 == 1);
      PORTC_RC2=1;
      OWReset();
      OWTX(0xCC);
      OWTX(0x44);
      while (OWRX1());
      OWReset();
      OWTX(0xCC);
      OWTX(0xBE);
      data[0] = OWRX();
      data[1] = OWRX();
for (dataLen=1;dataLen<=7;dataLen++)
      CONTAG=OWRX();
      allarme = FALSE;
      if (data[1] <= 7)
      {
       comp = data[1];
       comp = comp << 8;
       comp = comp + data[0];
       if (comp > maxtemp.Val)
       allarme = TRUE;
       else
       allarme = FALSE;
      }
      if (allarme)
      while(!ECANSendMessage(0x123, data, 2, ECAN_TX_STD_FRAME));
      else
      while(!ECANSendMessage(0x121, data, 2, ECAN_TX_STD_FRAME));
      putrsUSART("TX MSG\n\r");
      while (PORTBbits.RB0 == 0);
      Delay10KTCYx(50000);
      PORTC_RC2=0;
      }
```

Ciclo di attesa della pressione del tasto e relativa accensione del led rosso.

Sequenza di rilievo della temperatura e valorizzazione del vettore data.

La temperatura rilevata viene confrontata con il valore salvato in EEPROM. Se è superiore si valorizza la variabile booleana "allarme".

A seconda del valore di "allarme" il nodo invia un messaggio con ID=121h o ID=123h.

Dopo aver atteso il rilascio del pulsante, il nodo fa una pausa per evitare il "debounce". Al termine spegne il led rosso, per segnalare di essere pronto ad un nuovo invio.

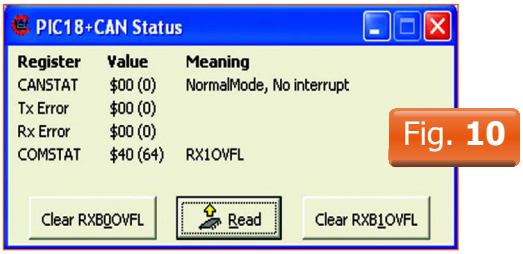


Fig. 10

ple. In pratica il ciclo di ricezione diventa come descritto nel Listato 1. A questo punto provate a collegare i due nodi (naturalmente dovete essere già in Normal Mode e i parametri di comunicazione devono essere stati configurati correttamente), agite per tre volte sul pulsante SW0, e successivamente leggete i registri di stato con il tasto "Read". Vedrete comparire l'indicazione dell'errore di overflow come visibile in Figura 10.

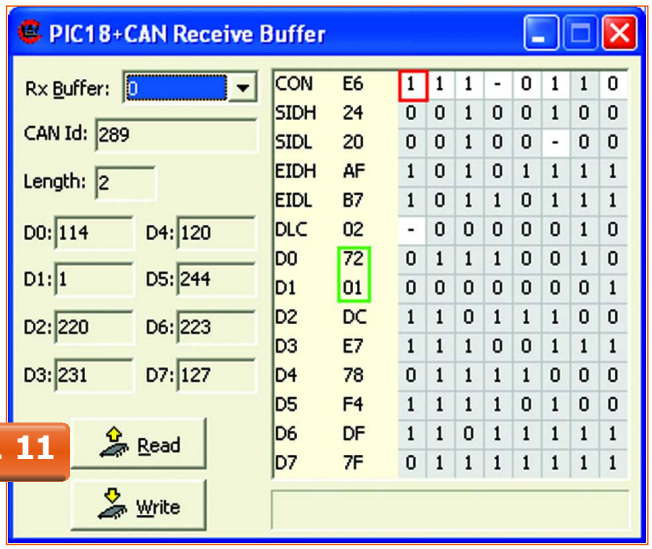


Fig. 11

Prima di eliminare il problema facendo clic sul pulsante "Clear RXB1OVFL" vediamo che cosa è successo. Per farlo dobbiamo introdurre due nuove finestre: la "**PIC18+CAN Receive Buffer**" e la **"PIC18+CAN Transmit Buffer"**. Esse permettono la visualizzazione dei buffer di ricezione e trasmissione. Posizionatevi sulla prima. Nella griglia di destra si vede il dettaglio dei bit di ciascun registro, mentre, sulla sinistra, vengono sintetizzati i campi di editing che evidenziano l'ID del messaggio, la sua lunghezza e i valori relativi. Attraverso la list box "RX Buffer" è possibile selezionare il buffer di ricezione da visualizzare (RX0 o RX1). Osserviamo RX0. Il primo messaggio ricevuto ha un ID=121h (289 in decimale) quindi con temperatura inferiore a quella massima predisposta. Potete tranquillamente ignorare i valori dei campi oltre D1 visto che abbiamo utilizzato sempre sequenze a due byte. Su D0 e D1 (evidenziati con il rettangolo verde) trovate rispettivamente il byte meno significativo e più significativo della temperatura rilevata dalla DS18B20. Fate attenzione che i valori nei campi di editing sono in decimale. Il valore relativo in esadecimale lo trovate nella griglia. È possibile trasformare il valore di un campo da decimale a esadecimale e viceversa semplicemente premendo il tasto F4. Nel nostro esperimento avevamo predisposto un valore massimo pari a 0190h. Nell'immagine seguente si vede la ricezione di un valore pari a 0172h pertanto l'ID risulta corretto secondo quanto avevamo definito nel firmware del nodo TX. Fate attenzione al bit più significativo di RX0CON (evidenziato in rosso). Corrisponde al RXFUL (Receive Full status bit). Per il motore di ricezione dopo l'arrivo del primo messaggio il buffer RXB0 risulta pieno ed il prossimo messaggio verrà intercettato dal RXB1 per evitare la perdita di dati (vedi Figura 11).

Nelle precedenti puntate del Corso avevamo spiegato come la presenza di un doppio buffer permettesse ad un nodo di leggere fino a due messaggi successivi senza che si verifichi il sovraccarico del sistema di ricezione. In effetti, è proprio quello che accade. Alla seconda pressione il messaggio viene intercettato dal buffer RXB1. Lo si vede nella figura seguente. Anche in questo caso, dopo la ricezione, il modulo mette a 1 il bit RXFUL. >

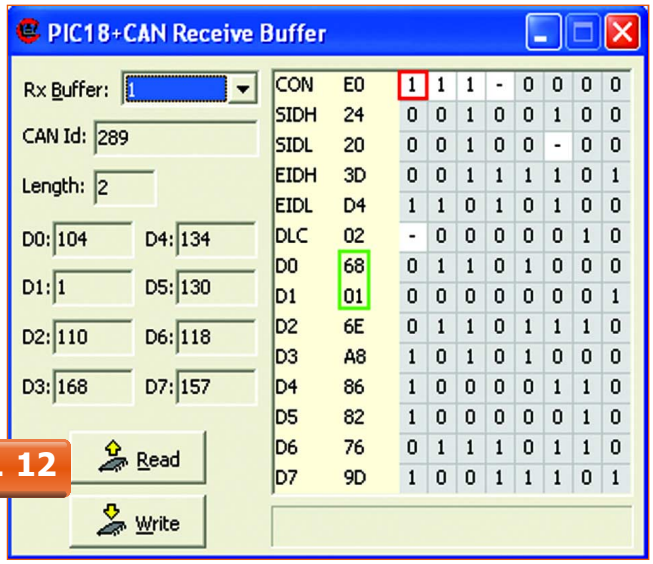


Fig. 12

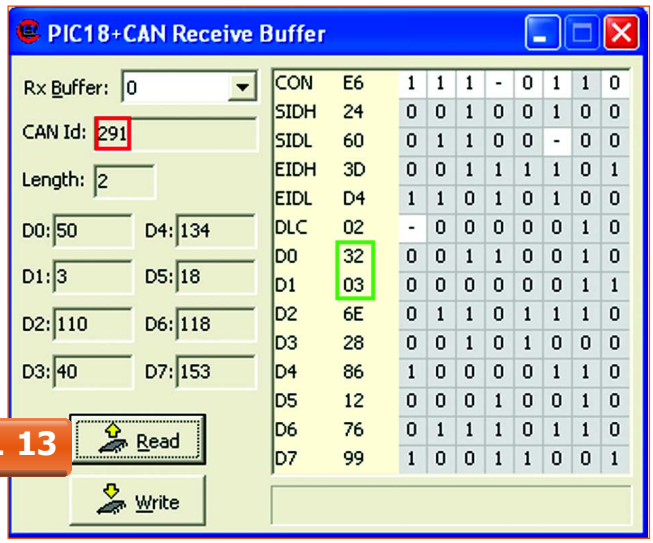


Fig. 13

di shift presente nella prima versione della libreria ECAN. Si vede chiaramente come i 3 bit meno significativi dell'ID standard a 11 bit si trovino in corrispondenza dei tre bit più significativi del SIDL. Potete provare a riscaldare la sonda fino a superare la temperatura limite. Si verifica facilmente il cambiamento dell'ID del messaggio inviato che passa da 121h a 123h. Ecco come si presenta la finestra relativa. Abbiamo evidenziato in rosso l'ID del messaggio 291=123h ed in verde il valore di temperatura rilevato 0332h (Figura 13).
La finestra "PIC18+CAN Transmit Buffer" è esattamente complementare a quella

Nel momento in cui premiamo per la terza volta il tasto sul nodo di trasmissione il messaggio arriva al nodo Monitor, ma il motore ha tutti e due i buffer pieni (RXFUL=1 sia su RXB0 che su RXB1), pertanto viene generato l'errore di overflow. È chiaro che, a livello firmware, soltanto prevedendo un regolare scaricamento dei buffer di ricezione adeguato alla velocità di comunicazione sul bus, è possibile evitare questo tipo di errore. La situazione di RXB1 dopo la seconda pressione è quella rapprentata nella Figura 12.

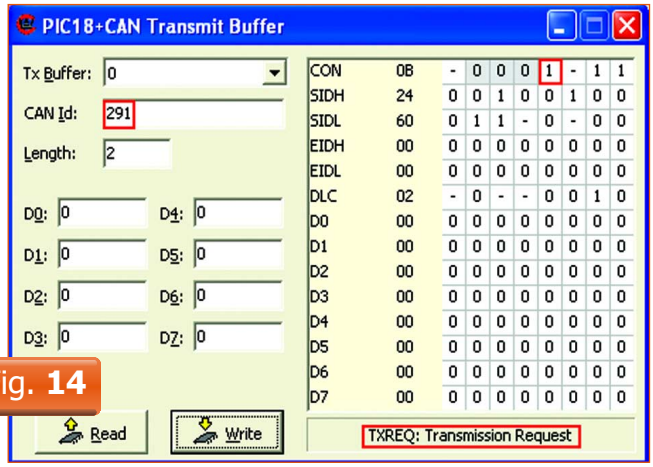


Fig. 14

Facendo clic sul pulsante "Clear RXB1OVFL" si azzera COMSTAT, ma questo non è sufficiente ad evitare il ripresentarsi dell'errore. Infatti, in questo modo, non facciamo altro che azzerare un flag di segnalazione, ma a livello operativo il motore CAN troverà sempre occupati i due buffer. Quindi, anche se azzeriamo il flag nel momento in cui premiamo di nuovo il pulsante sul nodo di trasmissione, generiamo l'errore. Dobbiamo perciò scaricare i buffer. Ricordate che ogni qual volta decidiamo di richiamare la "ECANReceiveMessage", le istruzioni eseguite, dopo aver estratto i dati ricevuti, si preoccupano di mettere a posto il registro RXBnCON. Pertanto, azzerate COMSTAT, posizionatevi su RXFUL sia del buffer RXB0 che su RXB1 e scrivete uno 0. Dopo aver fatto clic su "Write" il nostro nodo sarà di nuovo pronto a ricevere messaggi senza errori. La griglia laterale della finestra di ricezione permette di vedere la struttura dei due registri SIDH e SIDL di cui abbiamo parlato diffusamente nel Corso e che ci avevano creato qualche problema a causa dell'operazione appena vista. Essa permette di controllare i registri relativi ai buffer di trasmissione. Per completezza abbiamo inserito nella successiva immagine l'invio di un messaggio con ID=123h. Se trasformate il nodo TX in nodo RX, inserendo il firmware che avevamo distribuito nel capitolo 6, potete sperimentare lo stato di allarme. Abbiamo volutamente lasciato a 0 i due byte relativi alla temperatura per far vedere che, l'entrata nello stato d'allarme deriva esclusivamente dall'ID del messaggio. Il nodo di ricezione non entra nel merito del valore di temperatura trasferito, ma si affida totalmente ai filtri. Fate attenzione che non è sufficiente fare clic su "Write" per inviare il messaggio, ma è necessario valorizzare anche il bit TXREQ che abbiamo evidenziato in rosso (Figura 14). Soltanto in questo caso il motore CAN integrato nel modulo prenderà in considerazione quanto scritto nei registri ed invierà il messaggio non appena sarà possibile, proprio come avviene quando richiamiamo la "ECANSendMessage". A questo punto se provate a fare clic su "Write" vedrete sul nodo RX

accendersi il led giallo e, se avete collegato anche la seriale, comparirà nella finestra dell'HyperTerminale la chiamata al 115. Una volta valorizzato il bit TXREQ è possibile effettuare ulteriori trasmissioni semplicemente facendo clic sul pulsante "Write". I 3 bit in grigio rappresentano i flag relativi agli errori di trasmissione: Transmission Aborted, Transmission Lost Arbitration, Transmission Error Detected. Potete tranquillamente simulare una situazione di errore semplicemente staccando l'alimentazione al nodo RX. In precedenza fate clic su "Write", in maniera da tentare l'invio del messaggio e poi fate clic su "Read" della finestra "PIC18+CAN Status". Osservate le immagini di Figura 15 e 16. Nella seconda delle due, potete notare la valorizzazione del contatore TXERROR a 128 e del TXBP (Transmitter Bus Passive Bit). Vi rimandiamo alle spiegazioni del Corso per i dettagli. Nel momento in cui alimentate di nuovo il nodo di ricezione e provate ad inviare altri messaggi, vedrete che il contatore comincerà a diminuire fino a tornare a zero. Durante questa fase, nel momento in cui il contatore si trova tra 128 e 95, viene valorizzato correttamente il flag di warning come si vede in Figura 17.

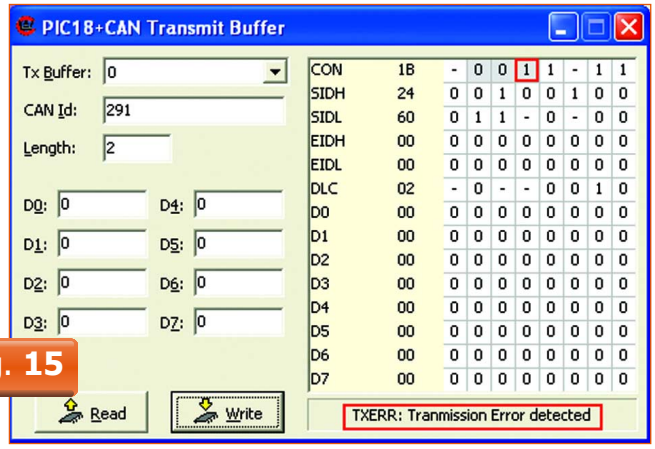


Fig. 15

In alto, il bit TXERR è a 1 (a seguito di un errore). A lato, valorizzazione di TXERROR e TXBP.

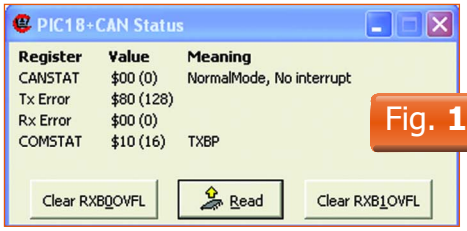


Fig. 16

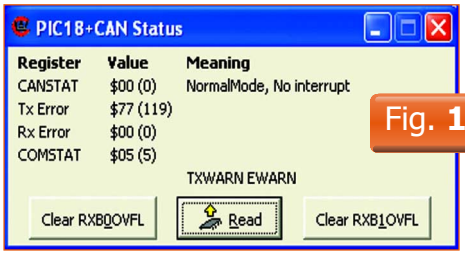


Fig. 17

L'ultima finestra che consideriamo è la **"PIC18+CAN Message Filters"**. Essa contiene una serie di campi relativi ai filtri e alle maschere definiti all'interno del modulo CAN. Attraverso questo form è possibile entrare in "Configuration Mode" e valorizzare maschere e filtri. La cosa interessante è che, grazie ai due pulsanti e ad una serie di led, è possibile testare direttamente il funzionamento degli stessi. Provate a fare clic sulla checkbox nell'angolo in alto a sinistra. Il nodo entra nella modalità di configurazione. Se avete ripristinato il firmware del nodo di trasmissione con quello distribuito nel capitolo 6 (per intenderci quello che invia sequenze continue di messaggi) vedrete che il lampeggio del led rosso si ferma, visto che tutte le operazioni di trasmissione e ricezione vengono bloccate. Ora, proviamo a valorizzare i filtri e le relative maschere come abbiamo fatto nell'ultimo capitolo del corso (vedi Figura 18). Avevamo visto come era possibile riconfigurare a runtime il valore di RXF2 attivando la chiamata al 115 durante lo stato d'allarme. Potete testare il funzionamento dei filtri semplicemente valorizzan- >

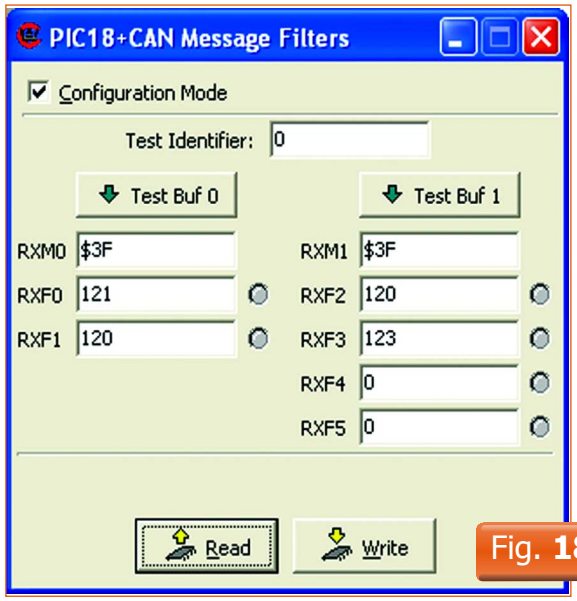


Fig. 18

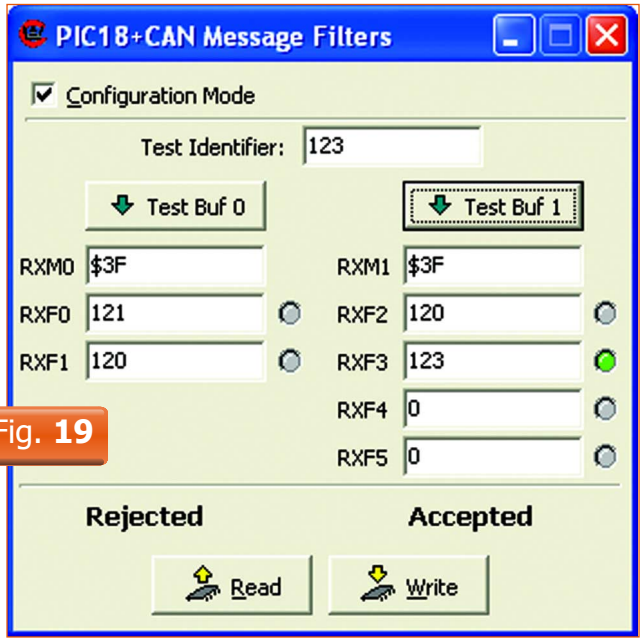


Fig. 19

in risposta un valore pari a 2, corrispondente a RXF2. In Figura 20 si vede chiaramente il risultato dell'intercettazione.

Conclusioni

In queste due puntate conclusive abbiamo visto, nel dettaglio, come sia possibile monitorare i messaggi su un bus CAN utilizzando, un piccolo programma, la nostra demoboard ed il firmware relativo. L'obiettivo che ci eravamo prefissati era quello di presentare degli strumenti in grado di supportare lo sviluppatore nella diagnosi del proprio firmware offrendo allo stesso tempo un punto di vista preferenziale per capire come funziona il modulo CAN integrato nel PIC. Il software si presta ad utilizzi di vario genere e più che rappresentare un sistema professionale di diagnostica, assomiglia di più ad un piccolo laboratorio didattico col quale è possibile sbizzarrirsi per sciogliere anche quei piccoli dubbi che spesso seguono ad una trattazione teorica.
Dal prossimo numero, terminata l'analisi teorica, ci occuperemo di domotica proponendo una serie di realizzazioni pratiche basate sui moduli della serie Velbus, tutti realizzati con chip Microchip. Questa serie comprende attualmente una dozzina di moduli che coprono le principali esigenze, dai controlli di luminosità a quelli per serrande; il sistema prevede inoltre numerosi moduli di ingresso e due circuiti di interfaccia per PC.

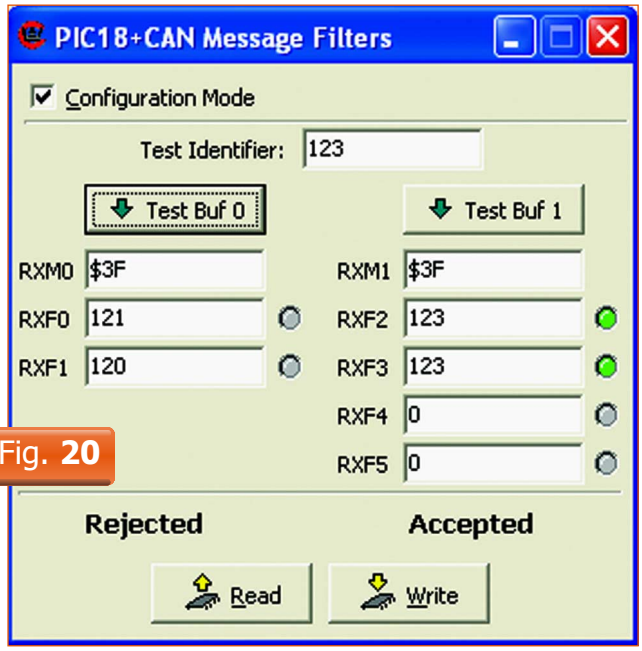


Fig. 20

do il campo "Test Identifier" con l'ID del messaggio che il nodo dovrebbe ricevere. Supponiamo, ad esempio, che il nodo RX riceva un messaggio con ID =123h. Facciamo clic su "TestBuf0", vedremo che per il primo buffer il messaggio non viene accettato. Nel momento in cui facciamo clic su "TestBuf1" vedremo accendersi il led verde in corrispondenza di RXF3 e comparire la stringa "Accepted", che segnala l'intercettazione del messaggio da parte del filtro corrispondente. Nell'immagine di Figura 19 si vede come appare il form dopo la prova. Simuliamo ora che cosa avviene nel momento in cui il nodo viene riconfigurato. Il filtro RXF2 diventa uguale a 123h. Proviamo ora a fare clic su "Test Buf 0". Chiaramente, per il primo buffer il messaggio viene scartato. Ma vediamo che cosa accade appena premiamo il tasto "Test Buf 1". Vedremo accendersi il led verde in corrispondenza di RXF2 e RXF3 proprio come deve avvenire affinché lo stato di allarme sia gestito correttamente. Si noti che, l'accensione di entrambe i led non significa che entrambi i filtri vengono considerati, in quanto vale sempre la regola della priorità. I filtri RXFn hanno una priorità maggiore quanto più piccolo è n. Pertanto quando richiamiamo la "ECANGetFilterHitInfo", in realtà riceveremo