UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

LAUREA TRIENNALE IN INGEGNERIA ELETTRONICA

Controller Area Network: specifiche e applicazione agli standard J1939 e CANopen

Laureando:
Alex Zecchinato

Relatore:
Prof. Roberto Corvaja

Anno Accademico 2022/2023

17 luglio 2023

Indice

Elenco delle figure

Elenco delle tabelle

Prefazione

L’idea di questa tesi nasce durante lo svolgimento di un tirocinio universitario svoltosi a seguito della fruizione della borsa di studio Mille e una Lode, promossa dall’Università degli Studi di Padova.

L’azienda con cui ho collaborato opera da anni nel settore del mantenimento del verde, realizzando macchinari per la gestione della vegetazione. Fin da subito è emersa la necessità di apprendere nozioni basilari del protocollo CAN e dello standard J1939, nonché argomenti principali di questa tesi.

La seguente tesi vuole essere il proseguimento della ricerca iniziata nel luglio 2022, con l'obiettivo di riunire le informazioni raccolte in un unico documento nel modo più chiaro possibile.

Mi sembra doveroso ringraziare l’azienda Energreen S.p.A. e Giammarco B., il quale ha sempre risposto alle mie numerose domande durante il periodo trascorso in azienda.

Introduzione

L'automazione industriale è da qualche decennio protagonista di un'evoluzione in termini di comunicazione tra dispositivi, abbandonando vecchi sistemi punto-punto a favore di sistemi di comunicazione basati su collegamenti comuni detti bus. È comune riferirsi a tali sistemi di comunicazione come bus di campo.

Parallelamente, anche il settore automotive ha subito un'importante crescita dell'elettronica all'interno dei veicoli. Fin dagli albori, però, nacque la necessità di ridurre la quantità di cavi di collegamento senza compromettere la sicurezza della trasmissione.

Nel 1986 Bosch, consapevole di dover soddisfare elevati standard di sicurezza e garantire cablaggi ridotti, sviluppò la prima versione del protocollo CAN per conto di Mercedes.

I due mondi negli ultimi anni si sono incontrati e, ad oggi, il protocollo CAN è certamente uno dei più importanti standard di comunicazione a livello industriale, considerando anche il numero di protocolli nati sulla base dello standard CAN.

Scopo di questa tesi è quello di fornire un'analisi sufficientemente dettagliata del protocollo CAN ed esaminare alcuni degli standard per livelli superiori basati sul protocollo CAN.

Struttura del documento

Nel primo capitolo si presentano alcuni concetti introduttivi al protocollo CAN, in particolare si espongono alcune nozioni sui bus di campo. Di seguito si illustra il modello OSI che definisce le basi teoriche di un qualunque protocollo di comunicazione.

Nel secondo capitolo si presenta in dettaglio lo standard CAN 2.0B, pilastro fondamentale per protocolli più evoluti come J1939 e CANopen. Si espongono i dettagli inerenti alla trasmissione del singolo bit con un accenno al mezzo trasmissivo. Successivamente, si presentano i quattro tipi di messaggi permessi dal protocollo, con uno sguardo alle operazioni di filtraggio e mascheratura.

Nel terzo capitolo si espongono le numerose varianti del protocollo CAN attualmente disponibili.

Nel quarto capitolo si illustrano alcuni standard per livelli superiori, attualmente in uso a livello industriale, basati sul protocollo CAN.

Infine, vengono riportate le conclusioni che riassumono il lavoro svolto.

Capitolo 1 Concetti introduttivi al bus di campo

1.1 Generalità

In passato la soluzione di interconnessione industriale più diffusa fu quella punto-punto, ovvero collegamenti specifici tra dispositivi di campo (sensori, attuatori, ecc.) e dispositivi di controllo (PLC, DSP, microcontrollori). Il collegamento così realizzato è poco efficiente e piuttosto costoso a causa dell'elevato numero di fili; inoltre, il cablaggio risulta molto complesso ed ingombrante.

La soluzione più adeguata è quella di adottare un collegamento comune, connettendo tutti i dispositivi di campo ad un unico bus detto bus di campo (*fieldbus*).

I principali vantaggi derivanti dall'adozione di tale soluzione sono:

- Riduzione della complessità del sistema e, di conseguenza, diminuzione degli errori di installazione.
- Riduzione del costo del cablaggio.
- Possibilità di espandere il sistema con estrema facilità.

D'altro canto, però, è necessario definire delle regole tra i dispositivi affinché lo scambio di dati possa avvenire su un collegamento comune. L'insieme di tali regole è detto protocollo di comunicazione.

Tipicamente lo scambio di dati è un'operazione che coinvolge più fasi, ognuna delle quali necessita di protocolli specifici. Esiste un modello di riferimento per lo scambio di dati tra due entità, il quale suddivide le diverse fasi in livelli, detto modello OSI.

1.2 Modello OSI

La necessità di uno standard per i sistemi informatici e di telecomunicazioni come il modello OSI (*Open System Interconnection*) nasce nella metà degli anni '70 quando i costruttori di sistemi informativi ebbero la necessità di connettere sistemi di diversa natura (differenti mezzi trasmissivi, diversi protocolli per la gestione dei dati, ecc.).

In letteratura si definiscono due tipologie di reti:

- *Reti chiuse*: interconnessione di dispositivi con le stesse caratteristiche (tipicamente prodotti dallo stesso costruttore) in grado di comunicare solo con entità della stessa tipologia.
- *Reti aperte*: sono sistemi informativi nei quali dispositivi con diverse caratteristiche possono comunicare in quanto rispettano lo stesso standard (come il modello OSI).

Il modello OSI definisce delle regole affinché dispositivi differenti, connessi in modo tale da formare reti aperte, possano comunicare ed interagire correttamente. Non solo, il modello fornisce una base rigida su cui sviluppare standard per la connessione di sistemi.

1.2.1 Struttura del modello OSI

Il modello OSI è composto da sette strati, detti anche livelli o *layer*, che suddividono in modo gerarchico la comunicazione tra due sistemi. L'architettura a livelli, di cui è presente una rappresentazione in Figura 1.1, nasce per ridurre la complessità progettuale di un sistema.
Ogni *layer* ha lo scopo di fornire servizi al *layer* superiore[1] attuando il principio dell'incapsulamento dei dati: ogni strato manipola i dati, tipicamente chiamati pacchetti o PDU (*Protocol Data Unit*), aggiungendo informazioni necessarie alla trasmissione.

Figura 1.1: Layer del modello OSI secondo la rappresentazione a pila.

[1] Generalmente si può dire che il livello N fornisce servizi alle entità di livello N+1 e, similmente, le entità di livello N si basano sui servizi del livello N-1. Fanno eccezione le entità del livello fisico che comunicano direttamente tramite il mezzo trasmissivo senza ricorrere a servizi di altri livelli.

Entità di diversi livelli comunicano mediante un'interfaccia detta SAP (*Service Data Point*), ossia un punto di connessione tra *layer* appositamente creato per lo scambio di dati.
Nello specifico, si supponga di dover scambiare un dato (PDU) dal livello N+1 verso il livello N: il PDU viene trasmesso al livello inferiore attraverso il SAP diventando un SDU (*Service Data Unit*) per il livello N, quest'ultimo aggiungerà delle informazioni all'SDU, dette PCI (*Protocol Control Information*). L'insieme di SDU e PCI forma il PDU del livello N.
In sintesi, i dati prodotti dal livello N (detti N-PDU) sono costruiti anteponendo ai dati del livello N+1 (N+1-PDU oppure N-SDU) la PCI del livello N.
Figura 1.2 e Figura 1.3 mostrano lo scambio di dati secondo il modello OSI.

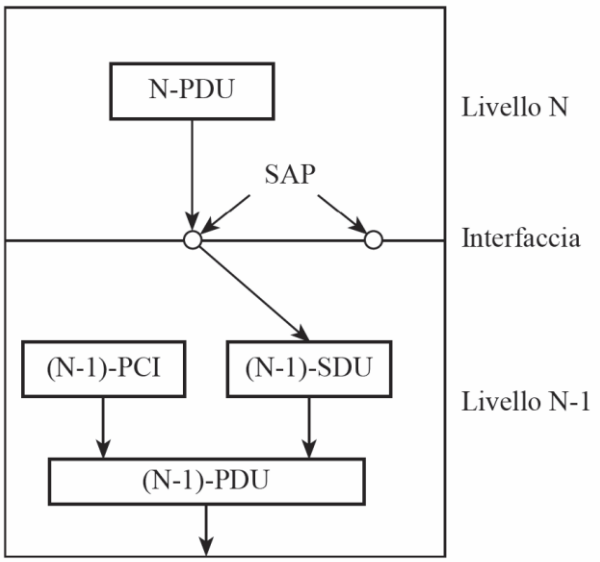


Figura 1.2: Scambio di dati tra due layer secondo il modello OSI.

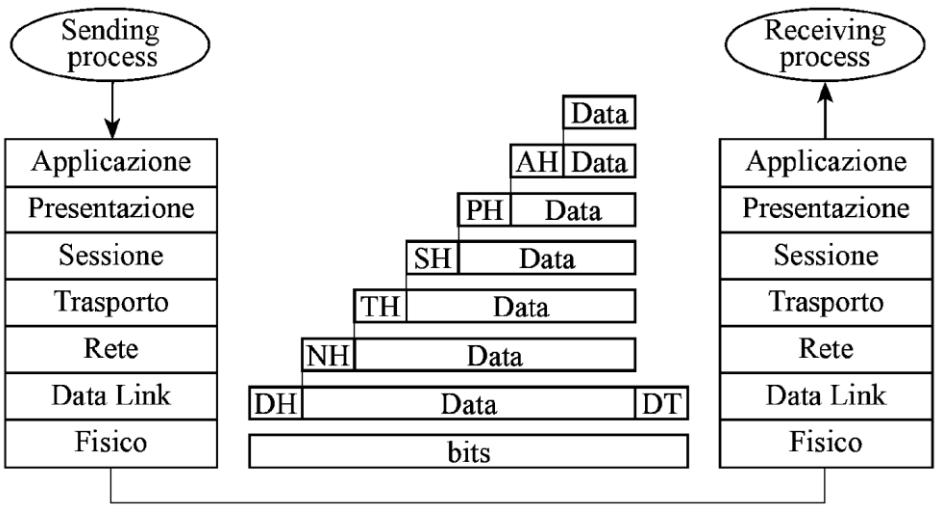


Figura 1.3: Struttura di un messaggio ed effetto dell'incapsulamento.

Il modello OSI fa uso di una strategia *peer-layer,* ovvero ogni livello comunica con il corrispondente livello del nodo destinatario in modo indiretto, solo il livello fisico comunica direttamente mediante il mezzo trasmissivo.

In altre parole, facendo riferimento a Figura 1.3, le informazioni aggiunte volta per volta da uno specifico *layer* sono destinate a raggiungere esclusivamente il proprio omonimo nel nodo destinatario.

Si consideri, a titolo esemplificativo, una certa applicazione che fa uso di un protocollo compatibile con lo standard OSI: l'informazione aggiunta in testa ad un messaggio dal layer *rete* attraverserà il layer *datalink*, che aggiungerà ulteriore informazione in testa al pacchetto, per poi essere trasmesso fino al nodo ricevente; quest'ultimo percorrerà i livelli in senso opposto a quello appena descritto cosicché ogni *layer* abbia accesso, in testa o in coda, unicamente alle informazioni codificate dall'omonimo *layer* nel nodo trasmittente. L'esempio può essere esteso facilmente considerando tutti e sette i livelli.

In riferimento a quanto presentato, la comunicazione tra due entità appartenenti allo stesso livello è realizzata attraverso dei protocolli, ovvero delle procedure specifiche che permettono di trasferire l'informazione dalla sorgente al destinatario. In letteratura si definiscono dei protocolli per ogni livello, tuttavia, è bene ricordare che nessun dato può essere trasferito direttamente dal livello N: il dato deve essere rielaborato dal livello sottostante fino al livello fisico, il quale è l'unico che può occuparsi della trasmissione.

1.2.2 Livello fisico (Physical layer)

Obbiettivo di questo livello è quello di trasmettere un flusso di dati grezzi (ovverosia bit) attraverso un mezzo trasmissivo: cavi elettrici, fibra ottica, aria, ecc.

Le principali funzioni di questo *layer* sono:

1. Abilitazione e disabilitazione delle connessioni tra nodo e mezzo trasmissivo.
2. Trasmissione dei dati. In questo caso per pacchetto (PDU) si intende il singolo bit.
3. Definizione del tipo di segnale: durata (*bit time*), livello di tensione alto e basso.
4. Definizione di tutto ciò che riguarda l'implementazione hardware del mezzo trasmissivo: tipologia di connessione, connettori, ecc.

A volte il livello fisico viene ulteriormente suddiviso nei seguenti sottolivelli (Tabella 1.1):

- *Physical Signaling* (PLS): responsabile della codifica e decodifica del singolo bit in opportuni segnali elettrici, della temporizzazione e della sincronizzazione del bit[2].

[2] Maggiori informazioni si possono trovare al capitolo 2, paragrafo 1.

- *Physical Medium Attachment* (PMA): definisce le caratteristiche del *transceiver*[3].
  Per quanto riguarda il bus CAN esistono diverse possibilità: *CAN High-Speed*, *CAN Low-Speed*, *Fault-Tolerant transceiver*, *Truck/Trailer transceiver*, *Single-Wire* e altri.
- *Medium Dependent interface* (MDI): descrive l'implementazione fisica tra *transceiver* e mezzo trasmissivo. In altre parole, definisce tipologia di cavi e connettori.

Data Link Layer	Logical Link Control (LLC)
	Medium Access Control (MAC)
Physical Layer	Physical Signaling (PLS)
	Physical Medium Attachment (PMA)
	Medium Dependent Interface (MDI)

Tabella 1.1: Suddivisione dei livelli Data Link e Fisico.

1.2.3 Livello Data Link (Data Link layer)

Il compito di questo livello è quello di permettere un trasferimento di dati affidabile e corretto attraverso il livello fisico. Tale compito si traduce nella supervisione costante dei messaggi alla ricerca di possibili errori che, grazie ad opportune tecniche di correzione, non raggiungono il livello di rete.

Tutto ciò è possibile suddividendo i dati in *frame* e aggiungendo delle informazioni di controllo sia in testa che in coda al messaggio. Si noti che per questo livello il PDU coincide con un *frame.*

Più nello specifico, il controllo dell'errore avviene aggiungendo all'SDU una sequenza di BIT, detta *checksum*, usata in ricezione per controllare la correttezza del pacchetto ricevuto.

In breve, il *checksum* viene calcolato sul pacchetto di dati da inviare tramite un opportuno algoritmo, il quale può variare dalla semplice somma dei bit del messaggio fino a funzioni ben più complesse, e ricalcolato sul messaggio ricevuto; se i due *checksum* differiscono è presente un errore di trasmissione.

Il livello *Data Link* è suddiviso nei sottolivelli:

[3] È un circuito di interfacciamento con il bus, necessario per garantire i corretti livelli di tensione e corrente dei segnali trasmessi sul bus.

- *Logical Link Control* (LLC): ha il compito di gestire il collegamento logico tra nodi della rete definendone il metodo di comunicazione. Permette anche il controllo del flusso, sincronizzando dispositivi lenti che tentano di comunicare con dispositivi veloci che, per tale scopo, vengono rallentati. In questo modo si evita che il buffer del nodo lento venga saturato dalla maggiore velocità di comunicazione del noto veloce.
- *Media Access Control* (MAC): nelle reti *multi-master* è possibile che più nodi accedano al mezzo trasmissivo contemporaneamente, determinando una collisione.
  La gestione dell'accesso multiplo al mezzo viene effettuata dal MAC che evita conflitti tra nodi.
  Questo sottolivello esegue anche la sincronizzazione del *frame* nel *bitstream*, ovvero determina il punto d'inizio e di fine di ogni frame nel flusso di bit trasmessi.
  Molto spesso è compito del MAC assegnare ai nodi connessi alla rete un numero identificativo, detto *MAC address*[4].

Tra le funzioni del livello in analisi vi è anche quella di correggere eventuali errori mediante ritrasmissione. Per ogni pacchetto inviato il destinatario risponde con un messaggio di *acknowledgement* (ACK) contenente lo stato della trasmissione. L'ACK può essere trasmesso in un messaggio a sé stante oppure, ottimizzando l'uso del mezzo trasmissivo, in coda ad un messaggio che viaggia in direzione opposta (verso il nodo sorgente); tale tecnica è detta *piggyback.*
Il mittente deve provvedere al rinvio di tutti i dati di cui è stato ricevuto un segnale ACK negativo oppure se non è stato ricevuto tale segnale entro un tempo prestabilito.
In conclusione, si può affermare che il livello *Data Link* mette in atto una serie di strategie che rendono, per quanto riguarda il livello rete e superiori, il mezzo trasmissivo una linea quasi esente da errori.

Ai fini della comprensione del protocollo CAN non risulta necessario conoscere gli ulteriori livelli del modello OSI in quanto lo stesso protocollo definisce solamente i primi due.

1.3 Standard di livello fisico

In generale, i segnali adoperati nei sistemi digitali, i quali implementano le funzionalità descritte dal protocollo di comunicazione a livello *Data Link*, non sono adatti alla

[4] Come si vedrà non è il caso del protocollo CAN. In questo protocollo, infatti, i nodi sono sprovvisti di indirizzi.

trasmissione lungo una linea di comunicazione (bus) in quanto tali sistemi non sono in grado di fornire opportuni valori di tensione e corrente. Inoltre, la trasmissione nel bus richiede specifici accorgimenti per ridurre i disturbi estremamente comuni in ambito industriale.
Risulta fondamentale l'adozione di circuiti di interfacciamento detti *line driver* (integrati anche nei *transceiver*) i quali implementano specifici standard di livello fisico, ovvero convertono i segnali *single-ended* provenienti dal sistema digitale in segnali adatti al bus.
Di seguito si presenteranno alcuni degli standard di livello fisico più usati[5] a livello industriale.

1.3.1 RS232

Lo standard definisce due livelli di tensione: +12V per il livello logico 0 e -12V per il livello logico 1. Il segnale è non bilanciato (*single-ended*) ovvero la tensione associata ai bit è misurata rispetto un riferimento comune.
Lo standard, sebbene ancora in uso, è ormai superato da standard più evoluti che permettono di raggiungere distanze di collegamento superiori.

1.3.2 RS422 e RS485

In ambito industriale la qualità dei segnali è spesso compromessa dall'elevato inquinamento elettromagnetico. Tipicamente i disturbi nascono dalle seguenti situazioni:

- *Accoppiamento induttivo*: quando il bus si trova vicino a conduttori percorsi da corrente viene indotta una tensione sul bus ad opera del campo magnetico concatenato. L'effetto risultante è una variazione della componente di modo differenziale dei segnali circolanti sul bus.
- *Accoppiamento capacitivo*: conduttori caratterizzati da tensioni variabili possono generare disturbi sul bus con effetti sulla componente di modo comune dei segnali.

Gli standard RS422 e RS485 utilizzano conduttori intrecciati al fine di ridurre l'area offerta al campo magnetico e, conseguentemente, ridurre i disturbi di modo differenziale. Spesso il doppino è schermato per eliminare i disturbi dovuti ad accoppiamenti capacitivi. Gli standard adottano segnali differenziali, ovvero l'informazione è associata alla differenza tra i livelli di tensione dei due conduttori. L'uso della trasmissione differenziale elimina i disturbi dovuti ad

[5] Gli standard presentati sono di tipo seriale ma ne esistono anche per comunicazioni parallele (ad esempio IEEE 488)

accoppiamenti capacitivi poiché i disturbi di modo comune affliggono entrambi i conduttori approssimativamente nello stesso modo.
Lo standard RS485 permette una trasmissione *multi-master*, ovvero a più trasmettitori. Al fine di evitare possibili collisioni tra trasmettitori è necessario che questi siano provvisti di specifici circuiti in grado di isolare il trasmettitore dal bus mediante uno stato di alta impedenza.

Capitolo 2 Standard CAN 2.0B

In questo capitolo si introduce la versione 2.0B del protocollo CAN. Si presenteranno le peculiarità del protocollo, alcuni dettagli implementativi e la struttura dei messaggi che possono circolare sul bus.
Innanzitutto, è bene ribadire che il protocollo CAN, in tutte le sue sfaccettature, definisce solamente il livello fisico ed il livello *Data Link*. Si vuole mettere in evidenza che le specifiche sono piuttosto incomplete per quanto riguarda il livello fisico; ulteriori dettagli come i connettori e altre informazioni necessarie all'implementazione del collegamento fisico possono essere trovate in documenti specifici (ISO fornisce solo alcune di queste specifiche). Per i motivi citati è possibile trovare in commercio soluzioni implementative piuttosto diverse (ad esempio diversi connettori).

2.1 Storia

Controller Area Network (CAN) è il protocollo di comunicazione maggiormente utilizzato nel campo automotive. Il protocollo CAN venne ideato nella prima metà degli anni '80 da Robert Bosch come sistema di comunicazione *affidabile* e *robusto*. È stato sviluppato con l'intento di sostituire cablaggi complessi con un unico bus di soli due fili. Negli anni il protocollo è maturato a tal punto da essere adottato anche al di fuori del settore automobilistico, attualmente, infatti, viene utilizzato in applicazioni aereospaziali, navali, robotiche e altro.
A livello industriale l'affidabilità del protocollo è particolarmente apprezzata, motivo del notevole successo anche in questo campo.

2.2 Peculiarità del protocollo CAN

In questa sezione si desidera mettere in evidenza le caratteristiche fondamentali del protocollo in quanto rappresentano senza dubbio il motivo del notevole successo dello stesso:

- *Semplicità e flessibilità del cablaggio*: il layer fisico del protocollo, ovvero la sua implementazione hardware, consiste in un doppino intrecciato (schermato o meno a seconda delle esigenze). Le periferiche, comunemente chiamate nodi, possono essere aggiunte o rimosse con estrema facilità in quanto non vengono identificate con indirizzi. Il protocollo CAN è orientato ai messaggi (*message-oriented*) e non agli

indirizzi (*adress-oriented*) come altri protocolli; questo significa che i nodi non necessitano di alcun identificatore.

- *Tempi di risposta rigidi*: il protocollo CAN è appositamente progettato per ottenere tempi di risposta entro vincoli temporali piuttosto stringenti, anche con un elevato numero di dispositivi connessi al bus. Tutto ciò è reso possibile dall'efficiente gestione delle priorità di cui si parlerà in seguito.
- *Immunità ai disturbi*: si tratta di una specifica estremamente importante la quale permette al bus di funzionare correttamente in ambito industriale. Il protocollo gestisce in modo autonomo l'eventualità di interruzione di uno dei due fili o corto circuito di una linea del bus verso l'alimentazione.
- *Affidabilità*: il sistema di controllo (*CAN controller*) è in grado di rilevare eventuali errori con cinque diversi metodi (due a livello di bit e tre a livello di messaggio). In caso di errore avviene la ritrasmissione del messaggio in modo completamente autonomo, senza intervento da parte del software che gestisce la rete.
- *Confinamento degli errori:* ogni nodo è in grado di autodiagnosticare propri malfunzionamenti e di autoescludersi dal bus se questi permangono. Tale capacità permette di isolare un nodo guasto mantenendo l'integrità del restante sistema.
- *Priorità dei messaggi*: i messaggi trasmessi possiedono una priorità definita nel campo identificatore (ID) del messaggio. La gestione delle priorità è molto efficiente e avviene a livello hardware secondo un modello basato su bit dominanti (0) e recessivi (1). È bene tenere a mente che in questi casi la priorità maggiore è assegnata all'identificatore di valore minore**.**
- *Bus multi-master e multicast:* conseguenza diretta della priorità dei messaggi è la possibilità di accesso da parte di più nodi al bus; in questo caso la priorità del messaggio determina quale nodo possa continuare la trasmissione. Il bus si definisce anche *multicast*, ovvero ogni nodo collegato al bus può trasmettere informazioni a più destinatari (ossia altri nodi collegati al bus).

2.3 Livello fisico

Come è stato sottolineato nel precedente capitolo, l'obiettivo principale del livello fisico è quello di tradurre i singoli bit in segnali. Per quanto riguarda il protocollo CAN le uniche specifiche fornite dai documenti tecnici riguardano la durata temporale dei bit *(bit time)*, la

tipologia di trasformazione bit-segnale *(bit encoding* e *physical encoding)* e la sincronizzazione tra nodi della rete.

2.3.1 Struttura della rete

Esistono diverse soluzioni implementative per il livello fisico[1] ma la più comune è quella definita nella norma ISO 11898-2 (*High-Speed CAN*) la quale richiede l'uso del doppino intrecciato (eventualmente schermato) e segnali differenziali: gli stati logici, recessivo (1) e dominante (0), sono rappresentati dalla differenza di potenziale tra le due linee, spesso denominate CANH e CANL. Il bit recessivo è codificato mantenendo la differenza tra le linee minore di 0.5V, mentre nello stato dominante la differenza di potenziale è maggiore di 0.9V.
In Figura 2.1 si mostra graficamente quanto presentato; come si può notare lo stato recessivo è ottenuto mantenendo CANH e CANL al livello di circa 2.5V, mentre nello stato dominante si incrementa la linea CANH di 1V e si abbassa la linea CANL di 1V, ottenendo così una differenza CANH-CANL di circa 2V.

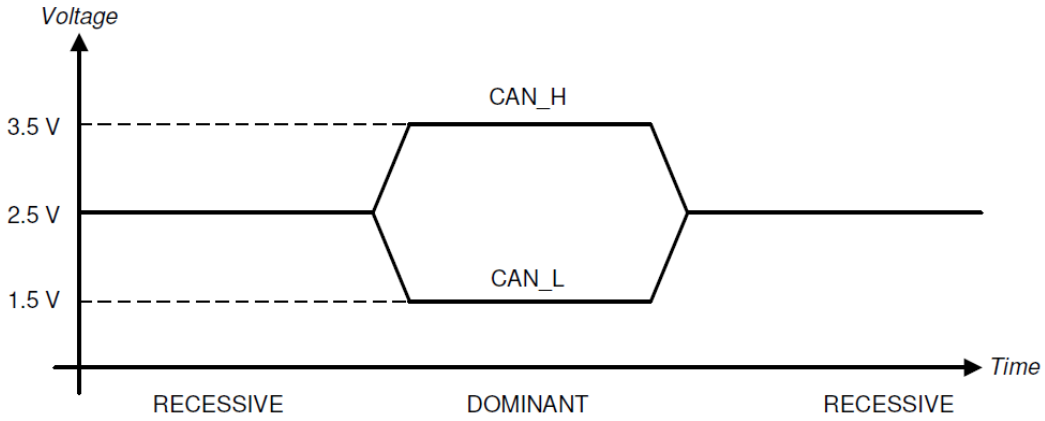


Figura 2.1: Stato dominante e recessivo.

Apparentemente, la soluzione di livello fisico adottata assomiglia molto allo standard RS485 presentato nel capitolo 1, tuttavia ci sono delle differenze marginali, ad esempio, nei livelli di tensione adoperati.
In Figura 2.2 si può osservare la tipica struttura di un nodo, il quale è formato da tre dispositivi: un microprocessore[2], una periferica che implementi il protocollo a livello *Data*

[1] Vedere il Capitolo 3.

[2] Il microprocessore può essere un microcontrollore oppure, se le specifiche lo richiedono, un DSP (*Digital Signal Processor*).

Link (*CAN controller,* spesso integrato nel microprocessore) ed un circuito di interfacciamento con il bus (*transceiver*).

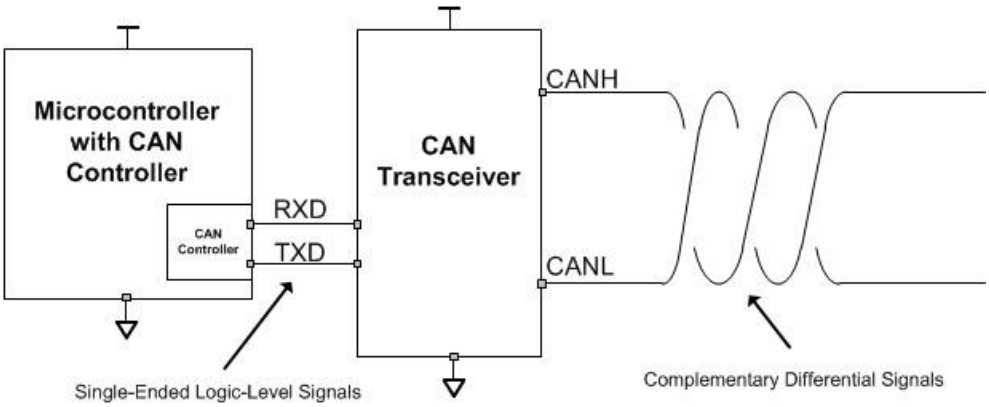


Figura 2.2: Struttura dei nodi.

I compiti svolti dal *CAN controller* e dal *transceiver* sono presentati in Tabella 2.1 e saranno meglio specificati in seguito.

CAN controller	Notifica di sovraccarico
	Filtraggio e mascheratura del messaggio
	Incapsulamento dei dati
	Codifica dei dati e *bit stuffing*
	Rilevazione di errori
Transceiver	Interfacciamento tra *CAN controller* e bus
	Isolamento tra *CAN controller* e bus

Tabella 2.1: Funzionalità del CAN controller e del transceiver.

Il *transceiver* è a tutti gli effetti un componente bidirezionale: può operare da trasmettitore, quando il flusso di dati è destinato al bus, oppure da ricevitore se l'informazione è diretta verso il nodo.

Nel primo caso il *transceiver* converte i segnali *single-ended* (RXD e TXD) provenienti dal *CAN controller* in segnali a logica differenziale (CANH e CANL).

Quando il *transceiver* opera come ricevitore deve eseguire la conversione contraria a quella appena descritta, convertendo segnali differenziali in segnali di tipo *single-ended* con opportuni livelli di tensione (livello logico).

Infine, è compito del *transceiver* proteggere il *CAN controller* da sovraccarichi, cortocircuiti ed altri guasti del bus isolandolo dal mezzo trasmissivo.
In una rete CAN devono essere sempre presenti resistori di terminazione (Figura 2.3) in quanto permettono di migliorare la qualità del segnale.

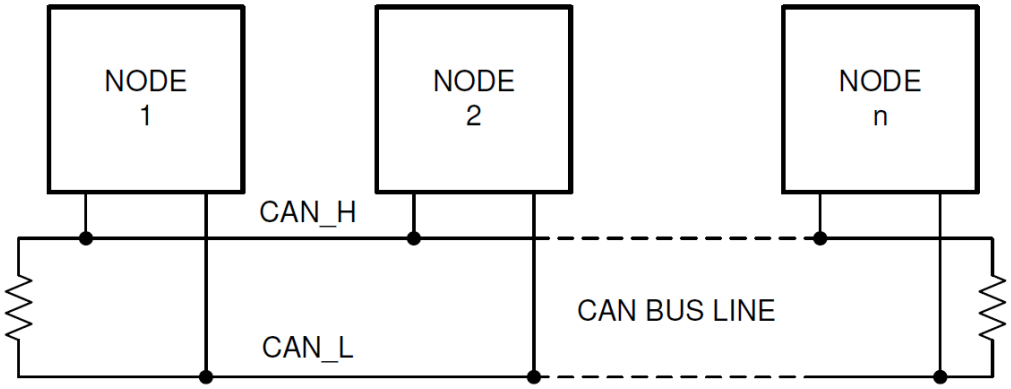


Figura 2.3: Struttura di una rete CAN.

Un segnale che viaggia lungo una linea di trasmissione sarà parzialmente riflesso nella direzione opposta, causando errori in ricezione. L'uso di resistori di terminazione e adottare la buona pratica di mantenere le linee trasmissive di lunghezza ridotta, permette di eliminare possibili riflessioni. Inoltre, i resistori di terminazione sono indispensabili per ottenere correttamente il livello recessivo, in quanto nei *transceiver* tale livello è dovuto alle sole capacità dissipative della rete[3].
Esistono diversi metodi di terminazione (Figura 2.4):

- *Standard*: prevede un singolo resistore ad entrambe le estremità (vedere Figura 2.3) di valore pari all'impedenza del cavo. Lo standard ISO 11898 impone l'uso di cavi con impedenza nominale pari a 120 Ω, perciò, anche i resistori di terminazione sono di tale valore.
- *Split*: si utilizzano due resistori da 60 Ω tra i quali viene posto un condensatore di valore compreso da 10 nF e 100 nF. Questo metodo è sempre più popolare in quanto permette di aumentare la compatibilità elettromagnetica (EMC). I due resistori devono essere di valore il più simile possibile per evitare l'insorgenza di rumore di tipo differenziale.

[3] L'analisi della struttura dei *transceiver* esula dallo scopo di questo documento, tuttavia si possono trovare articoli dedicati, ad esempio [8] e [9].

- *Biased Split*: la presenza di un riferimento di tensione non nullo tra i resistori da 60 Ω permette di aumentare la compatibilità elettromagnetica e le prestazioni del bus. Per i resistori valgono le medesime considerazioni fatte nel caso precedente.

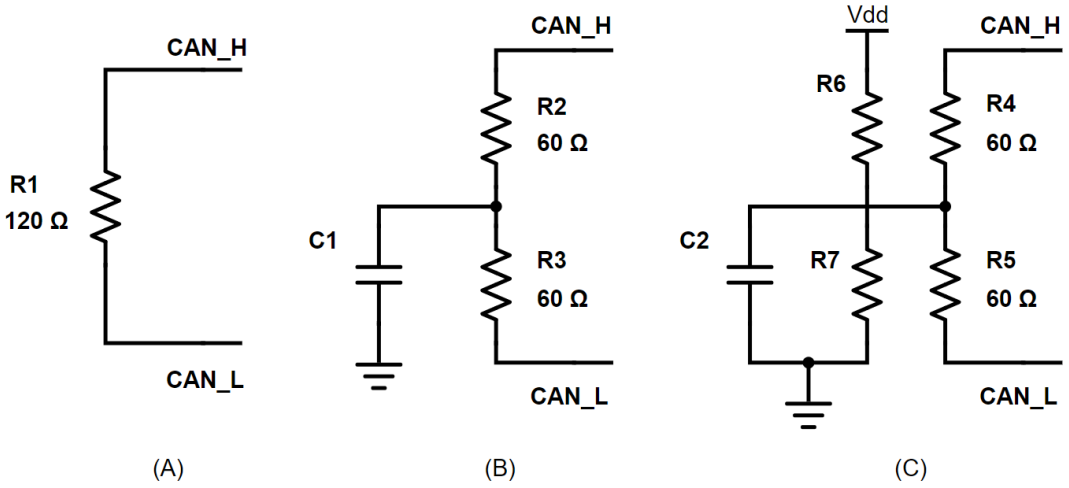


Figura 2.4: Terminazione Standard (a), Split (b) e Biased Split (c).

La lunghezza del bus deve rispettare alcuni limiti in funzione del *bit rate* adoperato, i quali garantiscono un *bit time* corretto in tutta la rete (vedere sezione successiva).
Esistono delle tabelle, come quella presentata di seguito, che consentono di determinare in prima approssimazione la massima lunghezza tollerata da uno specifico *bit rate*.

Bit rate	Bit time (μs)	Bus length (m)
1 Mb/s	1	30
800 kb/s	1.25	50
500 kb/s	2	100
250 kb/s	4	250
125 kb/s	8	500
62.5 kb/s	16	1,000
20 kb/s	50	2,500
10 kb/s	100	5,000

Tabella 2.2: Velocità di trasmissione e corrispondente lunghezza massima.

In conclusione, si ritiene opportuno sottolineare che lo standard CAN non specifica la tipologia di connettori, motivo per il quale è piuttosto comune riscontrare diverse varianti.
Nel corso degli anni sono emerse alcune preferenze circa la tipologia di connettore da utilizzare, una di queste è sicuramente il connettore 9-pin DSUB.

2.3.2 Trasmissione del bit: bit timing, bit encoding e sincronizzazione

In più occasioni si è introdotto il concetto di bit dominante (bit 0) e recessivo (bit 1) tralasciando, però, parecchi dettagli a riguardo. Innanzitutto, l'origine di questi nomi è strettamente legata a come viene gestita l'informazione nel bus: quando due o più nodi trasmettono contemporaneamente un messaggio viene messo in atto un sistema non distruttivo (senza perdita di informazione) detto arbitraggio[4]. L'arbitraggio avviene bit per bit secondo una logica AND cablata: durante la trasmissione, ogni nodo trasmittente verifica lo stato del bus e confronta il bit ricevuto con il bit trasmesso (operazione eseguita dal CAN controller sui segnali digitali TXD e RXD). Se viene ricevuto un bit dominante durante la trasmissione di un bit recessivo, il nodo interrompe la trasmissione. D'altra parte, se un nodo non nota differenze tra bit ricevuti e trasmessi per tutto l'intervallo di verifica, ha vinto l'arbitrato e può continuare a trasmettere il messaggio.
Durante la trasmissione di un segnale sono presenti delle non idealità che inevitabilmente riducono il tempo utile associato ad un bit, nello specifico:

- I *transceiver* determinano ritardi nella trasmissione. Anche il mezzo trasmissivo presenta un tempo di propagazione non nullo di cui si deve tenere conto nella durata del bit.
- La transizione da bit recessivo a dominante, che può avere luogo durante una trasmissione, deve propagarsi tra tutti i nodi della rete prima che termini la durata di un bit.
- La sincronizzazione, presentata in dettaglio successivamente, può impiegare parte del tempo associato ad un bit.

La durata di un bit (*bit time*) deve tenere conto non solo di molteplici fattori ma anche dei casi peggiori in cui questi possono presentarsi. Il *bit time* è sempre multiplo di un quanto temporale, nonché unità minima che determina la risoluzione della durata. La durata di un bit[5] è suddivisa come segue:

- *Synchronization segment* (SYNC_SEG): tempo necessario per la sincronizzazione, è pari ad un quanto temporale.
- *Propagation segment* (PROP_SEG): tempo richiesto per compensare i ritardi di propagazione nella rete. Normalmente è compreso tra 1 e 8 volte il quanto temporale.

[4] In letteratura si parla di tecniche di accesso al mezzo. La tecnica descritta fa parte della famiglia CSMA, in particolare si tratta della CSMA-BA (*Carrier Sense Multiple Access with Bitwise Arbitration*).

[5] Alla pagina [10] è presente un ottimo strumento in grado di calcolare il *bit time* in modo piuttosto completo.

- *Phase segment* (PHASE_SEG1 e PHASE_SEG2): sono intervalli temporali, di durata programmabile, appositamente pensati per correggere eventuali errori nella sincronizzazione. Il *Phase segment* è compreso tra 1 e 8 volte il quanto temporale.
- *Sample point* (SAMPLE_POINT): si tratta dell'intervallo dedicato al campionamento del segnale. Deve tenere conto del tempo di conversione dei convertitori A/D presenti nei circuiti di interfaccia.

Una rappresentazione del *bit time* è presente in Figura 2.5.

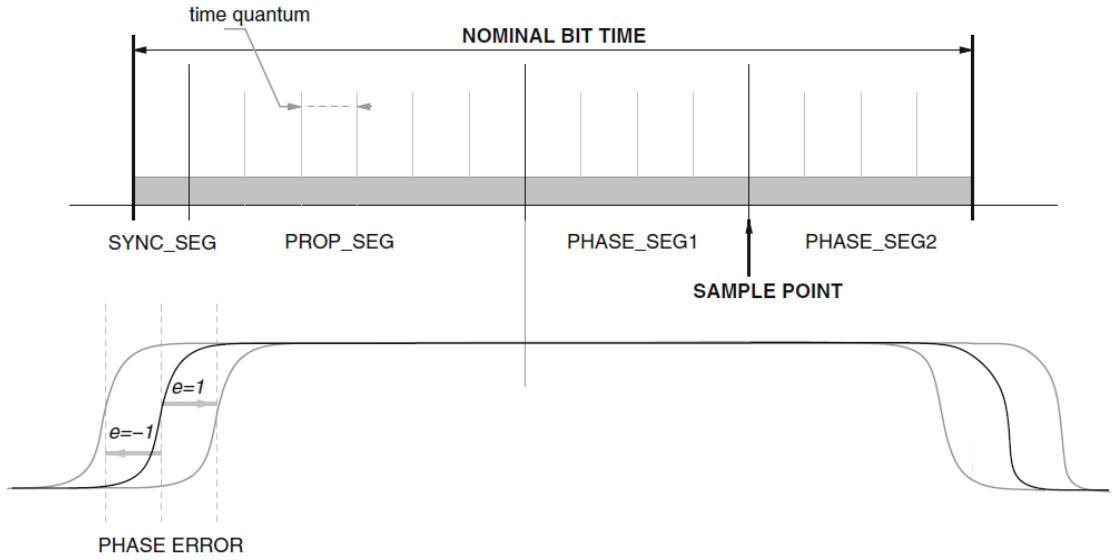


Figura 2.5: Componenti del bit time.

I sistemi di comunicazione che fanno uso del protocollo CAN sono di tipo asincrono, dunque senza trasmissione di un segnale di sincronismo (clock). Di conseguenza, è di vitale importanza ai fine della trasmissione la sincronizzazione tra nodi comunicanti della rete, ovvero la capacità da parte dei nodi di elaborare in contemporanea lo stesso segnale con il medesimo riferimento temporale. Per raggiungere la perfetta sincronizzazione è necessario che il *bit rate* del ricevitore sia allineato con la frequenza di trasmissione della sorgente. La sincronizzazione, che ha luogo durante le transizioni di stato, avviene secondo due modalità:

- *Hard synchronization*: la sincronizzazione avviene all'inizio del frame quando il bit *Start Of Frame* (vedere la sezione successiva) cambia stato da recessivo a dominante;
- *Re-synchronization*: avviene durante la trasmissione modificando il *Phase segment* del *bit time*. Nello specifico, può accadere che il fronte di un bit si verifichi al di fuori del *Synchronization segment, perciò,* è possibile accorciare PHASE_SEG1 o allungare PHASE_SEG2 in accordo al tipo e all'entità dell'errore di fase.

L'informazione digitale viene codificata dal *CAN controller* in segnali digitali secondo la codifica *Non Return to Zero* (NRZ), la quale assicura il numero minimo di transizioni. In realtà il tipo di codifica utilizzata non aiuta nella sincronizzazione, specialmente in tutte quelle situazioni in cui si hanno lunghe ripetizioni dello stesso bit, provocando delle piccole variazioni nelle frequenze di clock interne ai nodi che, a lungo andare, possono determinare la perdita di sincronizzazione. Per evitare tutto ciò viene messa in atto una tecnica detta *bit stuffing* che prevede l'introduzione di un bit complementare quando si hanno almeno cinque bit uguali nella sequenza trasmessa. In Figura 2.6 sono rappresentate le operazioni principali eseguite sui dati.

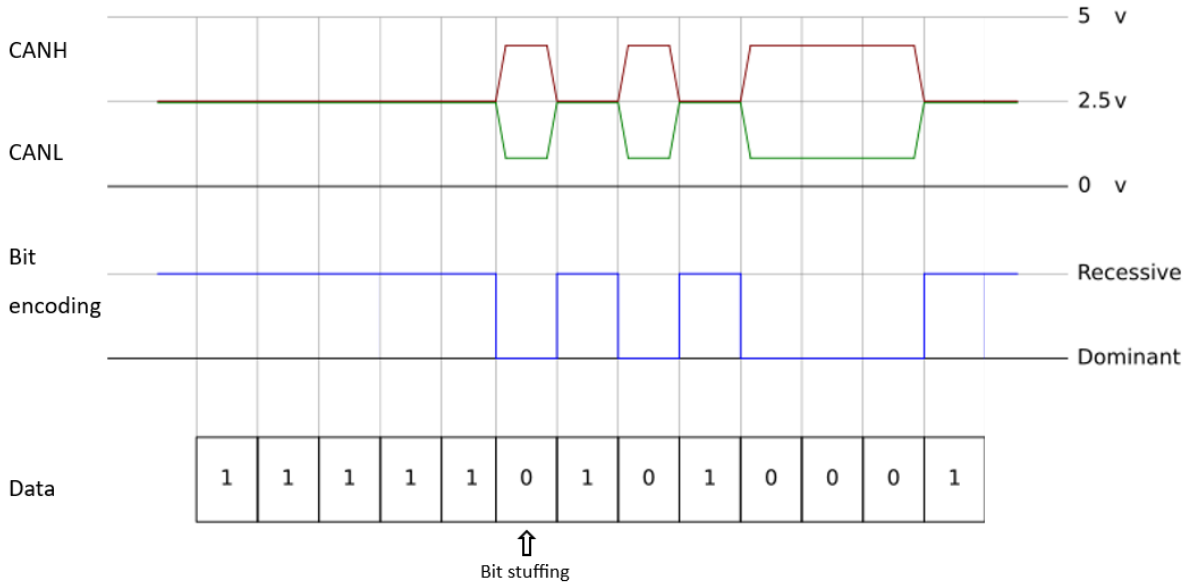


Figura 2.6: Trasformazioni eseguite sui dati.

2.4 Struttura dei messaggi

Nel bus CAN possono essere trasmessi quattro diversi messaggi: *Data Frame*, *Remote Frame*, *Overload Frame* ed *Error Frame*. I messaggi, detti anche *frame*, differiscono per il loro contenuto e verranno trattati singolarmente in seguito.

È di vitale importanza sapere che la trasmissione è di tipo *big-endian*[6], ovvero il dato viene spedito a partire dal bit più significativo.

Dopo ogni messaggio (indipendentemente dal tipo) viene spedito un *Interframe Space,* il quale consiste in tre bit recessivi con lo scopo di concedere tempo ai nodi di elaborare i

[6] spesso indicato con Motorola nei fogli tecnici. L'ordine *little-endian* è anche denominato Intel.

messaggi appena trasmessi. Dopo l'*Interframe Space* il bus rimane in uno stato recessivo (*idle*) fino al successivo messaggio.

2.4.1 Data Frame

Il Data Frame (Figura 2.7) è utilizzato per trasmettere informazioni tra un nodo sorgente ed uno o più destinatari. Il protocollo CAN si distingue da altri protocolli di comunicazione per l'assenza di indirizzi identificativi per i nodi. Ogni nodo connesso al bus può leggere parte del contenuto del messaggio e decidere autonomamente se recepire il messaggio o meno sulla base delle proprie funzionalità.
Tutti i messaggi presentano, quindi, una parte univoca, detta identificatore, che permette ai nodi connessi al bus di capire se il contenuto è rilevante o meno per le proprie funzionalità. Il meccanismo con il quale un nodo è in grado di scartare un messaggio è in realtà molto semplice: in fase di programmazione della periferica è possibile definire delle maschere[7] che verranno adoperate dal nodo, confrontandole con l'identificatore del messaggio, per verificare la compatibilità tra messaggio e nodo stesso.
L'assenza di indirizzi rende il bus estremamente flessibile, permettendo l'aggiunta di nodi senza dover modificare eventuali codici sorgenti (si parla anche di *plug and play*).
Il campo indirizzo distingue la versione 2.0B (formato esteso), che presenta identificatore a 29 bit, dalla 2.0A con identificatore a 11 bit (formato standard).
Ogni messaggio ha inizio interrompendo lo stato di inattività del bus (stato *idle*) mediante un bit dominante detto *Start Of Frame* (SOF). Successivamente vengono trasmessi sei campi, ovvero le macro-parti che compongono il messaggio, di cui verrà proposta una trattazione specifica nel seguito.

[7] Sono stringhe di bit. Per ulteriori dettagli vedere la sezione 2.5.

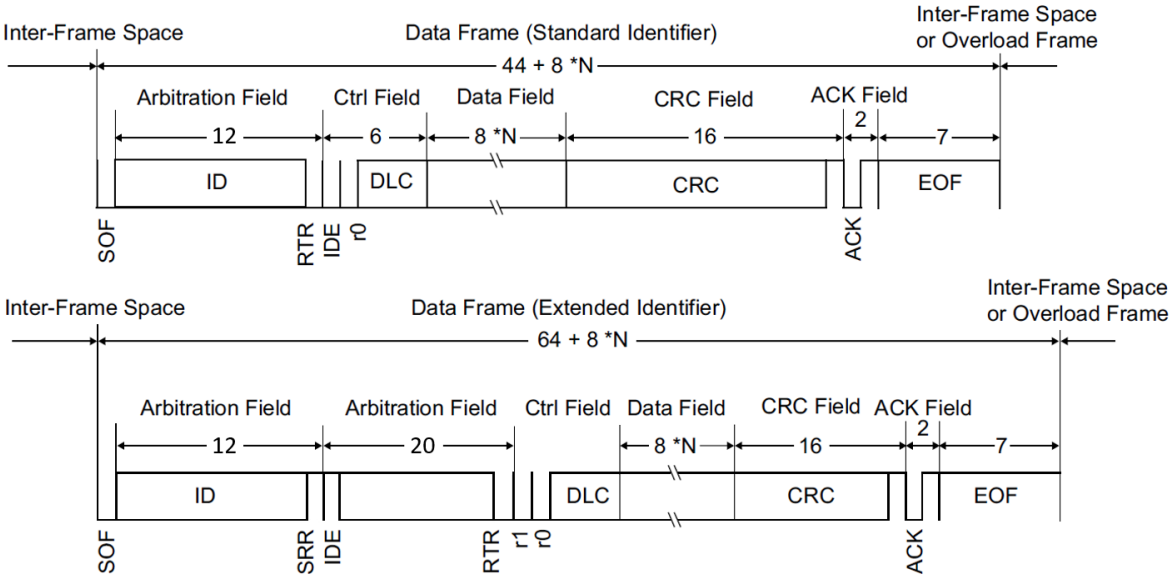


Figura 2.7: Struttura di un messaggio nella versione 2.0A e 2.0B (da [6]).

2.4.1.1 Arbitration Field

L'arbitration field è formato dall'identificatore (ID) a 11 bit per il formato standard del protocollo e 29 bit per il formato esteso. Ai fini di garantire la compatibilità tra i due formati l'identificatore a 29 bit è trasmesso in due parti: i primi 11 bit possono essere adoperati per trasmettere un messaggio nella versione standard per mezzo dell'infrastruttura della versione estesa; gli ulteriori 18 bit, separati dai primi mediante due bit di controllo, si utilizzano esclusivamente per messaggi propri della versione estesa. Per tale motivo i *CAN controller* compatibili con il formato esteso possono comunicare senza problemi con *CAN controller* nel formato standard.

La compatibilità tra i due formati è resa possibile dal bit IDE (*IDentifier Extension*) che permette di riconoscere *Data Frame* estesi (IDE recessivo) da *Data Frame* standard; in quest'ultimo caso il bit IDE è dominante e fa parte del *Control Field*.

Di particolare importanza è il bit RTR (*Remote Transmission Request*) che distingue *Data Frame* (RTR dominante) da *Remote Frame* (RTR recessivo).

La presenza di un bit sempre recessivo SRR (*Substitute Remote Request*) nel *Data Frame* esteso è necessaria al fine di mantenere deterministico l'arbitraggio tra messaggi standard ed estesi.

Come già preannunciato, nel protocollo CAN non esistono indirizzi ma si alternano fasi di contesa del bus tra nodi e fasi di trasmissione da parte del nodo che ha ottenuto la priorità.

La fase di contesa ha luogo durante la trasmissione dell'identificatore: se il bus risulta nello stato *idle*, un nodo può tentare di trasmettere un messaggio interrompendo lo stato *idle* per

mezzo di un bit dominante (SOF). I nodi contendenti si sincronizzano sul fronte del SOF e trasmettono contemporaneamente in modo seriale i bit dell'identificatore. Chiaramente, possono nascere delle collisioni tra i bit trasmessi dai diversi nodi, le quali vengono risolte tramite logica AND bit per bit determinando, alla fine del campo ID, un solo nodo vincitore.
Il meccanismo per il quale i nodi perdenti bloccano la trasmissione si basa sulla costante rilettura del bus dopo ogni bit trasmesso: un nodo, confrontando il valore presente nel bus con quello del proprio identificatore, può capire se escludersi dall'arbitraggio, in tal caso i valori sono diversi, oppure continuare a trasmettere i bit dell'identificatore. Secondo tale logica, il nodo con priorità maggiore è quello con identificatore minore in quanto presenta un maggior numero di bit dominanti (0) nella parte più significativa dell'ID.
Un esempio di arbitraggio è mostrato in Figura 2.8: tre nodi cercano di trasmettere simultaneamente un messaggio per cui si ha una fase di contesa del bus che ha inizio non appena uno dei nodi invia il SOF. La contesa termina nell'istante in cui il nodo 1 trasmette un bit dominante, modificando il bit trasmesso dal nodo 2. Si noti che l'identificatore del nodo 1 (0x15A) è il minore tra i tre.

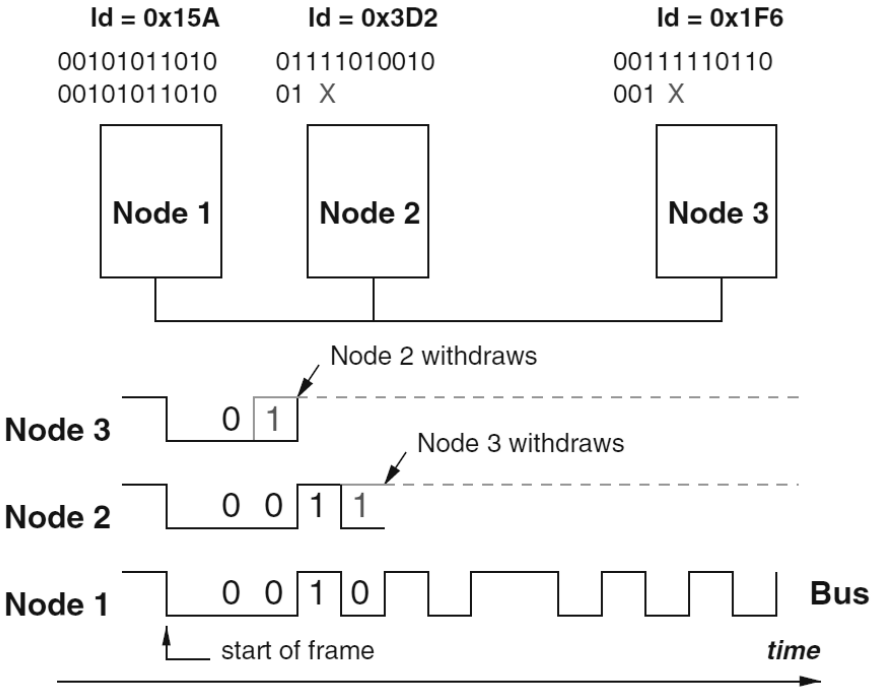


Figura 2.8: Risoluzione deterministica dei conflitti secondo logica AND.

2.4.1.2 Control Field

Il *Control Field* contiene sei bit: i primi due bit sono riservati e destinati a future evoluzioni del protocollo, i restanti quattro definiscono la lunghezza in byte dell'informazione trasmessa (DLC o *Data Length Content*).
Nello specifico, nel formato standard il primo bit riservato viene rinominato IDE ed è sempre dominante. Non sono presenti altre differenze in questo campo tra i due formati.

2.4.1.3 Data Field

È lo spazio dedicato al dato da spedire. È possibile spedire fino ad otto byte rigorosamente nell'ordinamento *big-endian*.
Si vuole far notare che spedire un dato tramite protocollo CAN è piuttosto inefficiente in quanto per spedire otto byte nel formato esteso sono necessari 128 bit, ovvero il doppio (si veda Figura 2.7). In realtà, è piuttosto comune tra le comunicazioni seriali una tale inefficienza e si tenga a mente che il protocollo è incentrato principalmente sull'affidabilità.

2.4.1.4 CRC Field & ACK Field

Il campo CRC (*Cyclic Redundancy Check*) è formato da 15 bit dedicati al *checksum* e da un bit di delimitazione sempre recessivo.
Il controllo di ridondanza ciclico è uno dei tanti metodi esistenti per l'individuazione di errori nella trasmissione[8]. L'algoritmo per il calcolo del CRC si basa su una matematica piuttosto complessa, di cui si riportano solo i risultati: il CRC permette di ottenere una stringa di N bit calcolati sulla base di un polinomio ottenuto dai dati trasmessi, la cui divisione con un secondo polinomio – detto generatore e prefissato – genera un resto. Il resto di tale divisione determina gli N bit da posporre al messaggio da trasmettere.
A livello teorico l'algoritmo risulta molto complesso ma nella pratica la procedura è molto semplice[9] e può essere svolta tramite circuiti basati su logica XOR e sullo scorrimento ciclico della stringa di bit da cui si estrae il codice. In pratica non è nemmeno necessario calcolare alcun polinomio in quanto esiste una relazione diretta con la notazione binaria.

[8] Permette solo la rilevazione e non la correzione. Eventualmente, è sempre possibile rispedire il messaggio.

[9] L'algoritmo è così comune che molti microcontrollori integrano delle periferiche ad-hoc per il calcolo automatizzato del CRC.

I modi con cui si ottengono i codici di CRC esulano dagli scopi di questo documento in quanto sono già disponibili diversi testi con spiegazioni dettagliate sugli algoritmi per il calcolo del CRC.
Per quanto riguarda il protocollo CAN la stringa di bit su cui viene effettuato il calcolo del CRC è formata da tutti i bit a partire dal SOF fino all'ultimo bit del *Data Field* (ovvero tutti i bit del messaggio trasmessi fino al *CRC Field*). Mentre il polinomio generatore è il seguente:

$$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$$

Il campo ACK consiste in due bit: durante il tempo associato al primo bit i nodi che hanno ricevuto correttamente il messaggio devono modificare il livello della linea mediante la trasmissione di un bit dominante. Il secondo bit è sempre recessivo ed è un bit di delimitazione.

2.4.1.5 End Of Frame

Il campo *End Of Frame* (EOF) è formato da sette bit recessivi che segnalano la fine del messaggio. Il campo EOF, ACK e il delimitatore del campo CRC non sono soggetti a *bit stuffing*, contrariamente ai restanti bit del messaggio.

2.4.2 Overload Frame

Viene utilizzato da nodi in condizione di sovraccarico per richiedere ulteriore tempo al fine di completare l'elaborazione dei dati. Un *Overload Frame* può essere generato soltanto durante *l'Interframe Space* ed è formato da sei bit dominanti (*Overload Flag*) seguiti da otto bit recessivi (*Overload Delimiter*). Questo messaggio, interrompendo l'*Interframe Space*, può essere facilmente riconosciuto dagli altri nodi, i quali rispondono con ulteriori *Overload Flag*[10] (in totale risultano 12 bit dominanti). Ogni nodo può generare al massimo due *Overload Frame* consecutivi per compensare il ritardo nell'elaborazione dei dati. Un esempio di *Overload Frame* è presente in Figura 2.9. Infine, questo frame è esente da *bit stuffing* e generalmente, viste le attuali capacità dei *CAN controller*, non è un messaggio molto utilizzato.

[10] Detti anche *Overload Echo* essendo trasmessi in risposta al nodo che ha generato l'*Overload Frame*

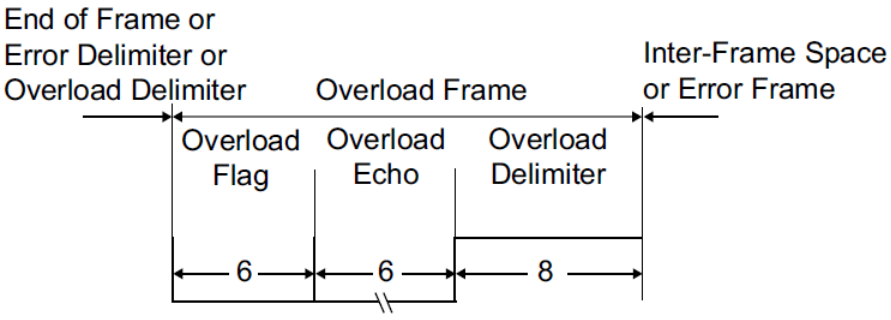


Figura 2.9: Struttura dell'Overload Frame (da [6]).

2.4.3 Remote Frame

È una tipologia di messaggio, strutturalmente identico al *Data Frame*, utilizzato per richiedere la trasmissione di un *Data Frame* di cui si conosce l'ID (Figura 2.10).
Presenta le seguenti caratteristiche:

- Il bit RTR è recessivo;
- L'identificatore coincide con l'identificatore del messaggio richiesto;
- Il *Data field* è vuoto;
- Il campo che indica la lunghezza dei dati (DLC) si riferisce al messaggio richiesto.

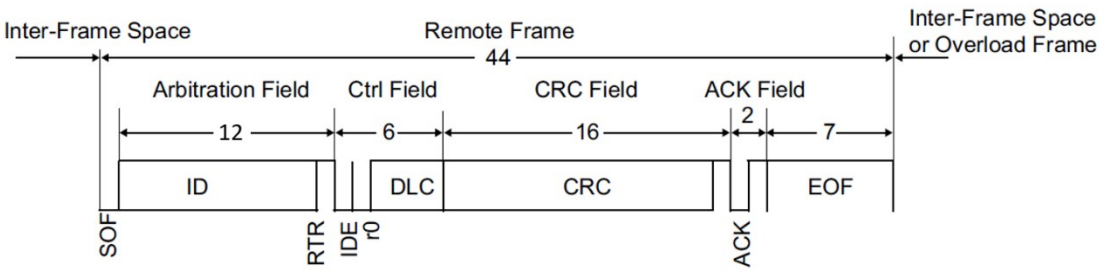


Figura 2.10: Struttura del Remote Frame (da [6]).

2.4.4 Error Frame

Il protocollo CAN è completamente incentrato sull'affidabilità, mettendo in atto una serie di misure per rilevare errori a livello di messaggio, diagnosticare lo stato di un nodo e confinare i nodi malfunzionanti. Tutte queste situazioni vengono segnalate per mezzo di un particolare messaggio detto *Error Frame*. Prima di descrivere la struttura di un *Error Frame* è bene riassumere i diversi errori che possono nascere durante una trasmissione.

Il protocollo è in grado di verificare la presenza di errori in cinque diversi modi (due a livello di bit e tre a livello di messaggio), generando omonomi errori. Questi sono:

- *Bit error*: è una forma di autocontrollo da parte di un nodo che possa trasmettere sul bus. Il transceiver alterna continuamente operazioni di scrittura e lettura del bus per verificare la coerenza con il bit trasmesso. Ad eccezione dei campi ID e ACK, che sono soggetti a sovrascrittura da parte di altri nodi, per tutti gli altri campi se il bit trasmesso differisce dal bit letto sorge un errore.
- *Stuff error*: si ha quando il ricevitore legge almeno sei bit consecutivi uguali in uno dei campi soggetti a *bit stuffing* (tutto il frame ad eccezione dei campi ACK, EOF e CRC delimiter). La violazione della regola del *bit stuffin*g comporta obbligatoriamente un errore di trasmissione.
- *CRC error*: viene generato se il CRC calcolato dal ricevitore differisce dal CRC trasmesso. Teoricamente il CRC è in grado di rilevare tutti gli errori fino a cinque bit corrotti, tutti gli errori di tipo *burst*[11] fino a 15 bit e tutti gli errori caratterizzati da un numero dispari di bit corrotti. Tutti i restanti casi di errore non sono rilevati con una probabilità pari a $3 \cdot 10^{-5}$. In realtà, i casi pratici dimostrano che l'analisi teorica è piuttosto ottimistica.
- *Form error*: sono errori che segnalano una violazione della struttura del messaggio. Ad esempio, la presenza di un bit dominante nel campo EOF genera un errore di questa tipologia.
- *Acknowledgement error*: nasce nel momento in cui il bus non viene portato a livello dominante nell'intervallo temporale dedicato.

In presenza di uno qualsiasi tra gli errori sopracitati viene generato un *Error Frame* che consiste in due campi: *Error Flag* ed *Error Delimiter*.

Esistono due forme di *Error Flag* a seconda dello stato[12] del nodo:

1. *Active error flag*: è composto da sei bit dominanti. Nasce una voluta violazione della regola di *bit stuffing* che viene riconosciuta dagli altri nodi della rete generando come risposta un ulteriore E*rror Flag* (detto *Flag Echo*).
2. *Passive error flag*: in modo del tutto analogo al caso precedente, è composto da sei bit recessivi.

[11] Sono errori che coinvolgono completamente e in modo continuativo una serie di bit.

[12] Ogni nodo classifica il proprio stato in base al numero di errori generati. Per ulteriori informazioni vedere la sezione successiva.

Il campo *Error Delimiter* consiste in otto bit recessivi. In Figura 2.11 è rappresentata la struttura dell'*Error Frame* con *Active error flag*. La struttura con *Passive error flag* si può ottenere trasformando tutti i bit dominanti in recessivi.

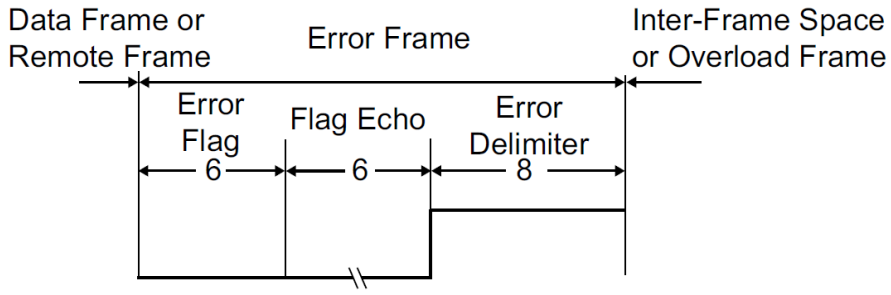


Figura 2.11: Struttura dell'Error Frame (da [6]).

Confrontando Figura 2.9 con Figura 2.11 si può notare una certa somiglianza. In effetti *l'Error Frame* e l'*Overload Frame* sono strutturalmente identici ma il primo può essere generato in qualunque momento mentre l'*Overload Frame* solo durante l'*Interframe Space.* Inoltre, *l'Error Frame* causa l'aggiornamento di alcuni contatori (vedere la sezione successiva) e la ritrasmissione del messaggio.

2.4.4.1 Confinamento degli errori

Il bus CAN può escludere nodi malfunzionanti se questi minacciano l'integrità della rete. In realtà è il nodo stesso che si autoesclude valutando il valore di due contatori, aggiornati sulla base degli *Error Frame* trasmessi e ricevuti, i quali forniscono una misura della probabilità di malfunzionamento della periferica: *Transmit Error Count* e *Receive Error Count*.
Un nodo può assumere tre possibili stati:

- *Error active*: la periferica può continuare la propria attività ma in caso di errore trasmette E*rror Frame* con *Active error flag*. Nonostante lo stato di errore, si ritiene che la periferica funzioni correttamente.
- *Error passive*: il nodo può trasmettere E*rror Frame* solo con *Passive error flag*. La periferica comincia a manifestare un comportamento anomalo.
- *Bus off*: la periferica viene disabilitata, ad opera del *transceiver*, in quanto non funziona correttamente.

Le regole di decisione dello stato di una periferica si basano sul valore dei contatori sopracitati ed in Figura 2.12 è riportato il diagramma di stato. Si vuole precisare che l'incremento dei contatori non è necessariamente unitario, tanto meno positivo e, spesso, sono

presenti delle specifiche tabelle[13] che determinano l'aggiornamento del conteggio in base al tipo di errore rilevato.

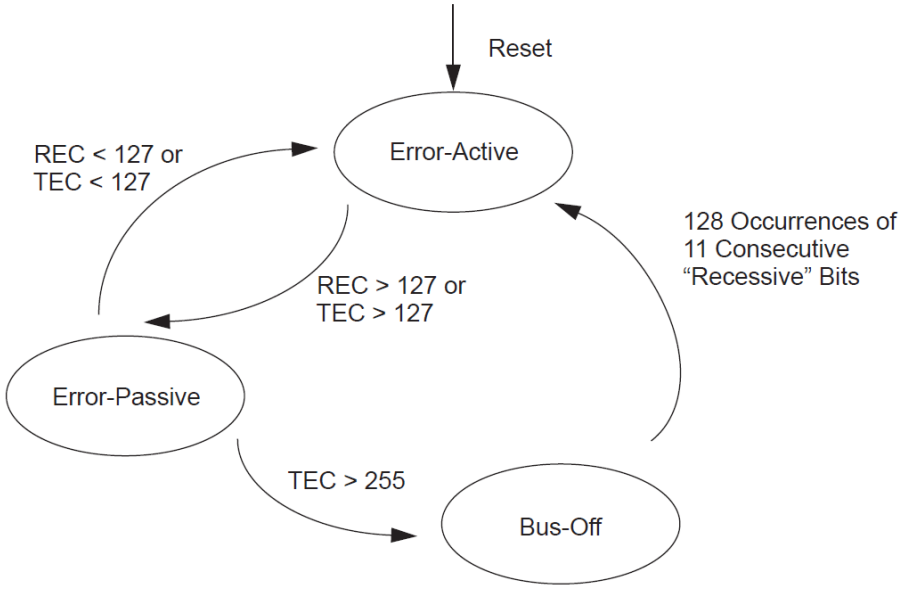


Figura 2.12: Diagramma di stato degli errori.

2.5 Ricezione del messaggio: maschere e filtri

Ogni nodo può accettare un messaggio o meno in base all'identificativo del frame stesso. Di seguito si presenteranno le modalità con cui la maggior parte dei *CAN controller* implementano la capacità di selezione dei messaggi coerenti con l'attività del nodo.

Solitamente i *CAN controller* presentano diversi registri nei quali viene salvato il messaggio ricevuto dopo alcune operazioni decisionali da parte di filtri e maschere[14] associate ai registri stessi. Generalmente il nome di tali registri inizia con la dicitura Rx. Quindi, un *CAN controller* presenterà un registro, che funge da buffer, per salvare il messaggio inviato nel bus indipendentemente dal suo campo identificativo (ID) e solo dopo alcune operazioni di mascheratura e filtraggio il frame verrà memorizzato nei registri Rx (a patto che il messaggio sia idoneo).

Una maschera consiste in una stringa di bit che determina quali bit dell'identificatore devono essere sottoposti a filtraggio. Il filtro è, ancora una volta, una stringa di bit che viene

[13] Fare riferimento alle specifiche ISO.

[14] Una maschera, così come un filtro, sono dei registri il cui valore può essere determinato dal programmatore in fase di progetto e, a livello hardware, vengono utilizzati come secondo operando assieme l'ID del messaggio per determinare il risultato di operazioni logiche che attuano la selezione.

comparata con l'ID sottoposto a mascheratura; se la comparazione va a buon fine il messaggio può essere caricato nel registro associato.

In Figura 2.13 è riportato lo schema concettuale dei registri del *CAN controller* MCP2515 [4]; come si può notare è presente un buffer (MAB) che trasferisce i frame verso i registri RXB0 e RXB1 solo dopo il consenso da parte delle operazioni di filtraggio e mascheratura.

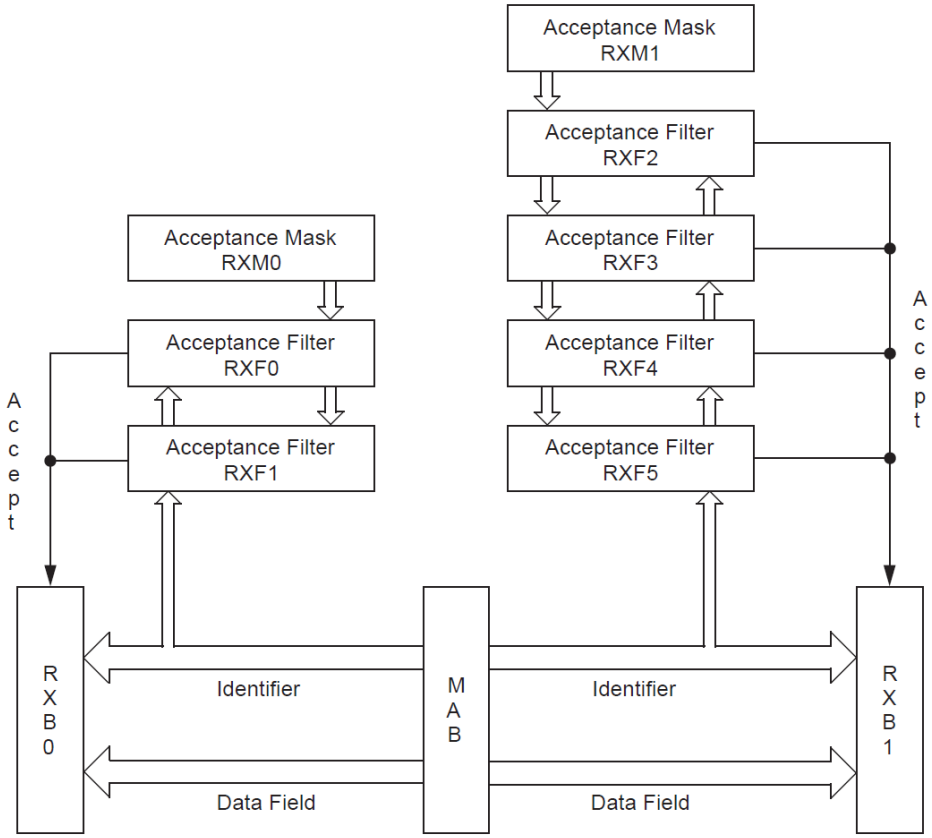


Figura 2.13: Organizzazione dei registri del CAN controller MCP2515.

L’esempio riportato in Figura 2.14 mostra come avvengono le operazioni di mascheratura e filtraggio: alcuni bit dell’ID vengono scartati per effetto della mascheratura, infine i filtri determinano se l’ID può essere accettato o meno. Come si può notare, l’ID non è compatibile con il filtro denominato *filter2*.

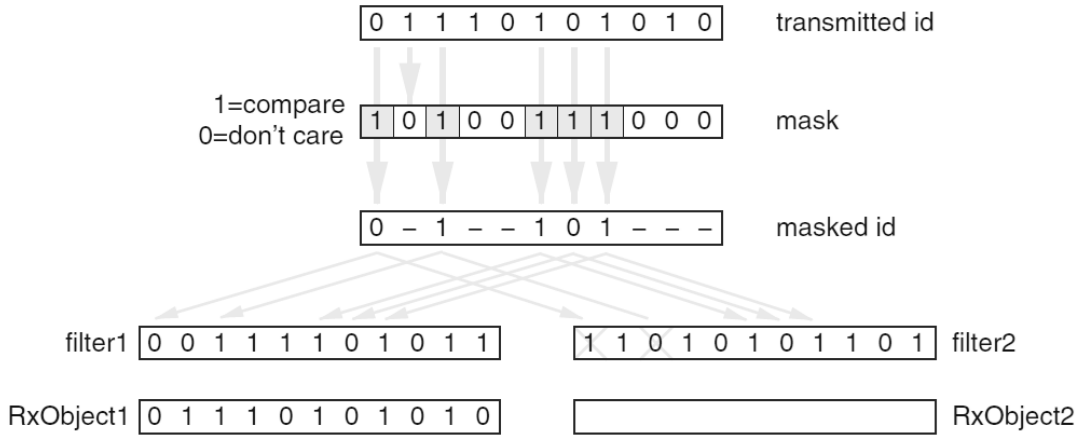


Figura 2.14: Esempio di mascheratura e filtraggio a livello di bit.

A livello hardware vengono implementati dei circuiti logici che svolgono operazioni bit per bit (*bitwise*) secondo la logica riportata in Tabella 2.3. Chiaramente le operazioni svolte in Figura 2.14 sono compatibili con la Tabella 2.3.

Mask Bit n	Filter Bit n	Message Identifier Bit	Accept or Reject Bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Note: x = don’t care

Tabella 2.3: Tabella di verità associata alle operazioni di filtraggio e mascheratura.

Capitolo 3 Versioni del protocollo

Di seguito seguirà una panoramica delle diverse versioni del protocollo con l'obiettivo di presentarne le principali differenze.

Una prima distinzione può essere fatta a livello *Data Link*, ovvero sulla diversa struttura del messaggio conseguente ad una differente tecnica di assemblaggio dei dati.

In letteratura è piuttosto comune riferirsi alle seguenti versioni con l'appellativo generazione; in particolare, si distinguono tre diverse generazioni:

1. *CAN* (prima generazione): è lo standard che assomiglia maggiormente alle specifiche originali del protocollo. È bene specificare che la prima generazione si può trovare in due versioni: CAN 1.0 e CAN 2.0 (nelle ulteriori versioni standard ed estesa). Il CAN 1.0 si riferisce al protocollo datato 1986 superato dalla versione CAN 2.0 la quale, contrariamente alla versione 1.0, rappresenta uno standard ISO ed è la versione trattata in questo documento.

   La norma di riferimento è la ISO 11898: inizialmente una singola norma definiva interamente tutte le specifiche (*Data Link layer* e *Physical layer*) mentre, attualmente, è suddivisa in sei parti:

   - ISO 11898-1: *Data Link layer.*
   - ISO 11898-2: *Physical layer.*
   - ISO 11898-3: *Low-Power*, *Low-Speed Physical layer.*
   - ISO 11898-4: TTCAN.
   - ISO 11898-5: *Low-Power*, *High-Speed Physical layer.*
   - ISO 11898-6: *Physical layer* con funzione *selective wake-up.*

   Come già anticipato, il CAN 2.0 si presenta nel formato standard (CAN 2.0A - identificatore a 11 bit) e formato esteso, detto anche CAN 2.0B e caratterizzato da identificatore a 29 bit.
2. *CAN FD* (seconda generazione): le attuali richieste del mercato hanno reso il protocollo CAN, sia nella versione 2.0A che 2.0B, poco attraente per applicazioni caratterizzate da elevati *bit rate*. Per tale motivo, a partire dal 2012, il protocollo è stato migliorato incrementando la velocità di trasmissione e introducendo messaggi di lunghezza variabile (fino a 64 byte); tale versione del protocollo è conosciuta come CAN FD (*Controller Area Network Flexible Data-Rate*). Il protocollo di seconda generazione accetta messaggi della precedente generazione, contrariamente un nodo

basato su un sistema di prima generazione emetterà un *Error Frame* in caso di ricezione di un messaggio di seconda generazione.

Alla base del protocollo di seconda generazione vi è la capacità incrementare il *bit rate* durante la trasmissione: quando un nodo inizia a trasmettere avviene la fase di contesa del bus durante la quale la velocità è limitata ad 1 Mbit/s, successivamente, terminata la fase di arbitraggio, il *bit rate* può essere aumentato[1] fino ai limiti imposti dai *transceiver* della rete. In media, tenendo conto che il frame di seconda generazione presenta un maggior numero di bit di controllo, si ottiene un *throughput*[2] sei volte superiore se il *bit rate* incrementa di otto volte rispetto quello dei frame di prima generazione [7].

Alcuni dei bit riservati del *Data Frame* di prima generazione sono qui utilizzati per distinguere un messaggio di seconda generazione da uno di prima generazione. A questo punto si può comprendere il motivo per cui un nodo di prima generazione, la cui aspettativa è che i bit riservati siano dominanti, genera un errore incontrando un messaggio di seconda generazione, il quale presenta il bit r1 recessivo se il messaggio presenta identificatore a 29 bit. Se il messaggio di seconda generazione presenta identificatore a 11 bit allora il bit r0 è recessivo. Esclusivamente per il protocollo di seconda generazione il bit r1, oppure r0 se il messaggio presenta ID a 11 bit, viene chiamato FDF (*Flexible Data rate Format*).

3. *CAN XL*[3] (terza generazione): si tratta dell'ultima evoluzione del protocollo, la quale introduce diverse novità rispetto le precedenti generazioni. Nel CAN XL, contrariamente alle due generazioni precedenti, le funzioni di arbitraggio e identificazione del frame vengono svolte su due campi separati: una stringa di 11 bit è finalizzata alla contesa del bus (arbitraggio) mentre un campo di 32 bit, detto *Acceptance Field*, viene usato per scopi identificativi.

   Nella terza generazione del protocollo il *Data Field* può arrivare a contenere fino a 2048 byte.

   Il CAN XL permette di adottare due diverse codifiche di linea: la NRZ (*Non Return to Zero*) e PWM (*Pulse Width Modulation*), la quale permette di raggiungere *bit rate* fino a 10 Mbit/s. Similmente a quanto avveniva per il *bit rate* nella precedente generazione,

[1] La commutazione del *bit rate* avviene in corrispondenza di un preciso bit e deve terminarsi entro tale *bit time*.

[2] Tale termine viene utilizzato per esprimere la capacità, intesa come quantità di dati utili trasmessi, del nodo.

[3] Al momento della scrittura di questa tesi è in corso la procedura di standardizzazione del protocollo. Le specifiche relative al CAN XL si possono trovare alla pagina [7].

nel CAN XL è possibile commutare tra diverse modalità (diversi *bit rate* e diverse codifiche di linea) durante uno specifico *bit time*.

Infine, il nuovo protocollo introduce diverse migliorie in ambito sicurezza e affidabilità introducendo, ad esempio, due CRC annidati[4].

Un'altra differenziazione consiste nelle funzionalità del *transceiver*: indipendentemente dal protocollo messo in atto a livello *Data Link*, sono possibili molteplici soluzioni per il livello fisico. In altre parole, esistono modi diversi per tradurre i *frame* in segnali adatti al bus:

1. *CAN High-Speed transceiver*: è la modalità di trasmissione più comune in grado di raggiungere *bit rate* di 1 Mbit/s. Viene descritta dettagliatamente nella norma ISO 11898-2.
2. *CAN Low-Power*: le attuali richieste del mercato[5] hanno reso necessaria l'implementazione di *transceiver* a basso consumo. In particolare, attraverso specifici segnali di controllo è possibile ibernare tutti i *transceiver* della rete. Informazioni più dettagliate possono essere trovate nelle norme di riferimento ISO 11898-3 (*Low-Power, Low-Speed Physical layer*) e ISO 11898-5 (*Low-Power, High-Speed Physical layer*).
3. *CAN selective wake-up*: permette di disabilitare solo alcune delle periferiche creando dei gruppi funzionali, ottenendo un grado di flessibilità maggiore rispetto la modalità precedente.

Quelle presentate sono solo alcune delle soluzioni implementative per il livello fisico e *Data Link*. In commercio esistono molti altri standard, proprietari oppure specifici di un settore come lo standard SAE J2411.

[4] Il primo codice è di 13 bit mentre il secondo, calcolato sul messaggio comprendente anche il primo CRC, è di 32 bit.

[5] Si pensi al campo automobilistico dove il crescente numero di funzionalità è in contrasto con la durata delle batterie. In questi casi risulta vitale poter disabilitare tutte le periferiche non vitali al fine di preservare la carica della batteria.

Capitolo 4 Standard per livelli superiori

Come già più volte preannunciato, il protocollo CAN definisce solo i primi due layer del modello OSI e, spesso, tralasciando molti dettagli sulla realizzazione, lasciando al progettista il compito di definire parecchi particolari implementativi.
Di conseguenza nasce la necessità di ulteriori standard in grado di definire con maggior dettaglio le specifiche del livello fisico e, per quanto possibile, ricoprire un numero maggiore di livelli secondo il modello OSI. L'adozione di tali standard migliora notevolmente la portabilità e l'intercambiabilità del prodotto.

4.1 J1939

Lo standard J1939, ideato dalla *Society of Automotive Engineers* (SAE), nasce per colmare le lacune presenti nelle specifiche del protocollo CAN. Inizialmente limitato al campo dei veicoli pesanti, ad oggi lo standard è utilizzato in svariate applicazioni come macchine agricole, sistemi marini e veicoli militari.
Fin da subito si tiene a precisare che i fogli tecnici relativi alle specifiche implementative dello standard non sono disponibili gratuitamente e sono commercializzati dalla SAE.
In particolare, le specifiche seguono la struttura del modello OSI: ad ogni livello è associato almeno un documento come riportato in Tabella 4.1.

OSI	*J1939*
Applicazione	SAE J1939/71
Presentazione	
Sessione	
Trasporto	SAE J1939/21
Rete	SAE J1939/31
Data Link	SAE J1939/21
Fisico	SAE J1939/11 SAE J1939/12 SAE J1939/14 SAE J1939/15

Tabella 4.1: Relazione tra modello OSI e standard J1939

Lo standard J1939 si basa sul protocollo CAN 2.0B, ovvero con identificatore a 29 bit[1].

4.1.1 Livello fisico

Anche se, a grandi linee, le specifiche del livello fisico sono quelle del protocollo CAN 2.0B, lo standard J1939 definisce il livello fisco in modo molto più rigoroso e dettagliato rispetto alle specifiche base.
Esiste un'unica possibilità implementativa per il layer fisico[2], di cui è possibile visionare i dettagli in Tabella 4.2.

Specifica		*valore*
Comunicazione		Differenziale
Baud rate		250 kbit/s
Numero massimo di nodi		30
Lunghezza massima della linea		40 m
Lunghezza massima collegamento bus-nodo		1 m
Cavo		doppino intrecciato con schermo
Connettori	*diagnostica*	9-pin Deutsch HD10-9-1939P
	Generico	3-pin Deutsch DT06-3S-E008

Tabella 4.2: Implementazione del layer fisico secondo J1939

Si può facilmente notare che le specifiche J1939 sono particolarmente conservative affidando la trasmissione a *bit rate* molto contenuti e lunghezze di rete ben al di sotto del limite massimo. Si prenda visione della Tabella 2.2 dove, per un *bit rate* di 250 kbit/s, la lunghezza ammessa è di 250 metri, ben al di sopra di quanto specificato da SAE. Chiaramente, specifiche così conservative aumentano la robustezza del sistema complessivo.
Nello standard J1939 è comune chiamare i nodi della rete con il termine ECU (*Electronic Control Unit*) in quanto, in ambito automotive, è piuttosto normale che ogni sottosistema abbia la propria unità computazionale (si pensi al motore oppure alla trasmissione).

[1] Una delle maggiori promotrici del protocollo CAN 2.0B fu proprio SAE in quanto, come sarà chiaro nelle sezioni successive, lo standard J1939 necessita una maggior capacità di indirizzamento.

[2] Sono riportate le specifiche per la versione J1939/11. Per le versioni J1939/14 e J1939/15 cambiano alcuni parametri (vedere [13]).

Ogni ECU presenta un *indirizzo* a scopo identificativo, il quale, evidentemente, rappresenta la prima differenza con il protocollo CAN.

4.1.2 Parameter Group

Una caratteristica fondamentale del J1939 è la rigida standardizzazione dei messaggi che circolano sul bus.

Lo standard raggruppa parametri correlati a livello funzionale in *Parameter Group* (PG) i quali vengono identificati tramite un numero univoco di quattro cifre esadecimali (16 bit) detto *Parameter Group Number* (PGN).

La mappa tra il valore di uno specifico parametro e il dato binario da trasmettere per mezzo del messaggio CAN è univocamente definita nelle specifiche dello standard. In Tabella 4.3 è riportato un esempio di specifiche simili a quanto si può trovare nei fogli tecnici forniti da SAE: nella parte alta della tabella sono riportate le informazioni sul tipo di messaggio (*PDU format* e *PDU specific*), specificando anche il PGN e la priorità del messaggio; tali informazioni verranno trattate nello specifico in seguito. La mappa da effettuare per ottenere la conversione del dato nella stringa di bit corretta fa riferimento al valore SPN.

Engine Temperature 1

Transmission repetition rate	1s
Data length	8 bytes
Data page	0
PDU format	254
PDU specific	100
Default priority	6
PGN	65124 (0xFE64)

Start position	*Length*	*Parameter Name*	*SPN*
1	1 byte	Example Parameter 1	SPN_1
2.1	4 bits	Example Parameter 2	SPN_2
3-4	2 byte	Example Parameter 3	SPN_3
5-6	2 byte	Example Parameter 4	SPN_4
7.5	4 bits	Example Parameter 5	SPN_5
8	1 byte	Example Parameter 6	SPN_6

Tabella 4.3: Specifiche per Parameter Group.

Fin da subito risulta fondamentale comprendere come si debba posizionare il dato convertito nel *Data Field* di un messaggio CAN, a tale scopo si consideri lo schema di Tabella 4.4, in relazione alla Tabella 4.3, dove si mostra la posizione di ogni singolo bit.

Byte 1								Byte 2								Byte 3								Byte 4							
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
SPN_1								Non definiti				SPN_2				SPN_3 (primo byte)								SPN_3 (secondo byte)							

Byte 5								Byte 6								Byte 7								Byte 8							
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
SPN_4 (primo byte)								SPN_4 (secondo byte)								SPN_5				Non definiti				SPN_6							

Tabella 4.4: Disposizione dei dati nel messaggio CAN.

4.1.3 Codifica dell'identificatore

La codifica dei 29 bit dell'ID del messaggio CAN è standardizzata dal J1939 e permette di trasferire diverse informazioni: priorità, identificatore del PGN, tipo di trasmissione. La codifica dell'ID si ottiene accostando correttamente le informazioni presenti nella parte alta di Tabella 4.3, opportunamente convertite in valori esadecimali.

Innanzitutto, è bene sapere che sono possibili due forme di trasmissione: punto-punto e *broadcast*. Nel primo caso la comunicazione avviene tra due soli nodi e, conseguentemente, è necessario conoscere l'indirizzo del nodo ricevente (*Destination Address* - DA). Invece, nella comunicazione *broadcast*, il messaggio può essere letto da tutti i nodi della rete.

Per codificare l'identificatore è necessario disporre delle specifiche SAE J1939/71 di cui si è già riportato un esempio in Tabella 4.3.

In riferimento a Tabella 4.5, i primi tre bit più significativi dell'identificatore rappresentano il campo priorità, il quale permette di risolvere eventuali collisioni e mantenere tempi di latenza limitati durante la trasmissione di un messaggio. Come spesso accade, la priorità più alta è associata al numero minore (caratteristica ereditata dal protocollo CAN).

Nei fogli tecnici viene riportata una priorità consigliata per ogni *Parameter Group*, tuttavia, come specificato nel documento J1939/21, è possibile assegnare una qualsiasi priorità nel range possibile (0-7) al fine di mantenere tempi di latenza accettabili per ogni messaggio.

A seguire, sono presenti un bit riservato (di valore zero se non diversamente specificato) e un bit che identifica il campo *Data Page,* il quale risulta utile per aumentare le possibilità di indirizzamento (ovvero il numero di messaggi).

I successivi 16 bit sono occupati dal campo PGN che può essere suddiviso in due parti: *PDU format* e *PDU specific*, quest'ultimo assume significato diverso a seconda del tipo di trasmissione.

Se la comunicazione è di tipo punto-punto allora i primi otto bit del PGN, ovvero il campo *PDU format*, devono assumere valore minore di 240. In questo caso, il campo *PDU specific* (nonché i rimanenti otto bit del PGN) rappresenta l'indirizzo del nodo ricevente (*Destination Address* - DA).

Invece, se la trasmissione avviene verso più nodi, il *PDU format* è di valore maggiore o uguale a 240 ed il *PDU specific* è usato per aumentare la capacità di indirizzamento. Per tale motivo nella comunicazione *broadcast* si può trovare il termine *Group Extension* in sostituzione al termine *PDU specific*.

Si conclude con l'indirizzo del nodo trasmittente (*Source Address* - SA) che viene codificato nei primi otto bit meno significativi del messaggio.

In generale, per comporre l'ID del messaggio, è sufficiente posizionare i dati presenti in Tabella 4.3 all'interno del ID come mostrato in Tabella 4.5.

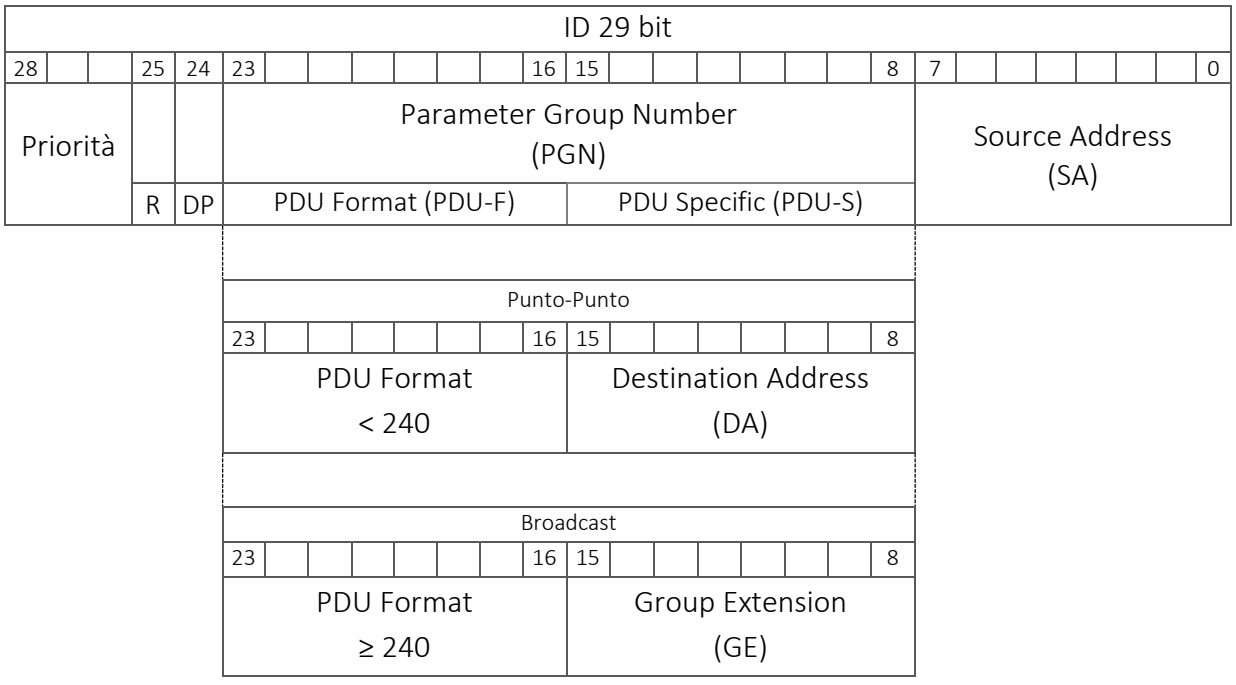


Tabella 4.5: Struttura dell'identificatore per trasmissioni Punto-Punto e Broadcast.

Evidentemente l'uso di una standardizzazione così rigida è svantaggioso per il numero di messaggi indirizzabili[3]: nel caso di comunicazione punto-punto si hanno solamente 240

[3] Si tenga a mente che il protocollo CAN 2.0B permette di indirizzare, in assenza di qualsiasi standardizzazione, più di 500 milioni di messaggi (2^{29}).

messaggi in aggiunta ai 4096 messaggi indirizzabili per la trasmissione *broadcast*. A questo punto risulta evidente lo scopo del campo *Data Page* che permette di raddoppiare il numero di possibili messaggi (4336 per DP=0 e 4336 per DP=1).
Il secondo svantaggio è rappresentato dalla standardizzazione del campo *Data Field* del messaggio CAN in quanto, per una specifica applicazione, non sono sempre necessari tutti i parametri del *Parameter Group* e non è possibile spedirne solo alcuni.

4.1.4 Codifica del Data Field

La codifica del *Data Field* è rigidamente standardizzata dal J1939 ed è descritta nel documento J1939/71 secondo i campi SPN (*Suspect Parameter Number*). Le specifiche relative al *Parameter Group* (Tabella 4.3) definiscono la posizione del dato all'interno del *Data Field* mentre le specifiche relative agli SPN definiscono la mappa da eseguire per ottenere il valore binario a partire dal dato. In Tabella 4.6 è riportato un tipico esempio di SPN. La mappa da eseguire è la seguente:

$$valore\ binario = ((dato - Offset)/Resolution)_2$$

Analogamente:

$$dato = (valore\ binario)_{10} * Resolution + Offset$$

Nelle specifiche è riportato anche l'intervallo entro il quale può variare il valore dell'SPN.

SPN_3	*Engine ECU Temperature*		
Data Length:	2 bytes		
Resolution:	0.03125 °C/bit	Offset:	-273 °C
Data Range:	-273 to 1734.96875 °C		
PGN reference:	65124		

Tabella 4.6: Specifiche esemplificative di SPN.

Si vuole far notare che la necessità dei parametri *Resolution* e *Offset* è dovuta al fatto che i valori trasmessi sul bus sono positivi ed interi, perciò, per ottenere numeri frazionari e negativi è necessario ricorrere a fattori moltiplicativi e costanti additive, qui chiamate *Resolution* e *Offset*. Questa forma di rappresentazione dei dati, estremamente semplificata rispetto le codifiche proprie dell'informatica (virgola mobile, complemento a due, ecc.),

permette a microcontrollori con basse capacità computazionali di elaborare grandi quantità di dati (si noti che il parametro *Resolution* di Tabella 4.6 è una divisione per 32, facilmente ottenibile come *shift* binario).

4.1.5 Transport Protocol

Può accadere che alcuni *Parameter Group* non siano rappresentabili con soli 8 byte (dimensione del *Data Field* di un messaggio CAN 2.0B).
Ricorrendo ad un protocollo di trasporto è possibile trasmettere *Parameter Group* che richiedono fino a 1785 byte, suddividendoli in più pacchetti da 8 byte. I nodi riceventi dovranno ricostruire il dato una volta ricevuti tutti i pacchetti.
Anche in questo caso sono possibili trasmissioni *broadcast* e punto-punto. In entrambi i casi la trasmissione ha inizio mediante un messaggio, denominato BAM per la comunicazione *broadcast* e RTS per la punto-punto, contenente informazioni utili ai nodi riceventi (PGN, numero di byte, numero di pacchetti, ecc.) per preparare le adeguate risorse (registri di accettazione).
Nel caso di comunicazione *broadcast* dopo il messaggio informativo vengono spediti tutti i pacchetti senza che i ricevitori possano rispondere, mentre, per la comunicazione punto-punto, nasce un *handshake*[4] tra trasmettitore e ricevitore il quale dirige la comunicazione richiedendo una porzione specifica del messaggio.

4.2 CANopen

Lo standard CANopen, similmente al J1939, definisce dei protocolli di alto livello mantenendo per il livello fisico e *Data Link* le specifiche del protocollo CAN. CANopen nacque in Europa negli anni ‘90 ed è attualmente gestito da *CAN in Automation* (CIA).

4.2.1 Livello fisico

Le specifiche in termini di *bit time*, *bit rate,* codifica bit-segnale, lunghezza e tipo di linea sono identiche a quelle presenti nello standard ISO 11898 in quanto CANopen adotta il protocollo CAN per i primi due livelli del modello OSI.

[4] Con questo termine si intende la procedura che ha luogo durante la fase inziale di una comunicazione al fine di stabilire regole di comunicazione e trasmettere informazioni utili alla trasmissione stessa.

Il documento CIA DR-303-1 permette l'uso di svariati connettori e definisce per ciascuno di essi l'assegnazione dei pin. I connettori maggiormente diffusi sono il *9-pin D-sub* ed il *5-pin mini style* dei quali si forniscono le specifiche relative all'assegnazione dei pin in Tabella 4.7.

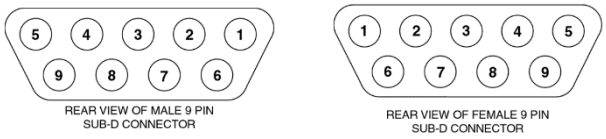
5-pin mini style			*9-pin D-sub*		
male connector / female connector					
Pin	Segnale	Descrizione	Pin	Segnale	Descrizione
1	CAN_SHLD	Schermo (opzionale)	1	CAN_SHLD	Schermo (opzionale)
2	CAN_V+	Alimentazione esterna (opzionale)	2	-	Riservato
3	CAN_GND	Ground	3	CAN_GND	Ground
4	CAN_H	Linea CANH	4	CAN_L	Linea CANL
5	CAN_L	Linea CANL	5	-	Riservato
			6	CAN_V+	Alimentazione esterna (opzionale)
			7	-	Riservato
			8	CAN_H	Linea CANH
			9	GND	Ground (opzionale)

Tabella 4.7: Assegnazione dei pin secondo specifiche CIA DR-303-1.

4.2.2 Object Dictionary

Il cuore di ogni nodo CANopen è l'*Object Dictionary* (OD), una tabella contenente tutti i dati del nodo, siano essi la descrizione delle funzionalità di un nodo, la configurazione del nodo oppure i dati prodotti. Un nodo può leggere e scrivere, ove possibile, l'OD di altri nodi.

L'*Object Dictionary* è una collezione di *entry* individuate per mezzo di un indirizzo composto da 16 bit; ogni *entry* può contenere fino a 256 *subentry* identificate mediante un sotto-indirizzo (in letteratura *Subindex*) di 8 bit.

Le possibili *entry* sono 65536 di cui solo alcune devono essere obbligatoriamente implementate in quanto necessarie al funzionamento del protocollo. Il fatto che solo alcune delle *entry* siano obbligatorie crea innumerevoli buchi all'interno dell'OD. In Tabella 4.8 si mostra la struttura dell'OD, formato dalle seguenti parti:

- *Data Types*: le *entry* in questa sezione (0001h – 0FFFh secondo Tabella 4.8) non contengono alcun dato ma servono per definire il tipo di dato. Il CANopen definisce il tipo di dato *Standard* (INTEGER, UNSIGNED, REAL, ecc.) e *Complex,* il quale è un'aggregazione di più dati *Standard*.
- *Communication Entries*: in questa sezione sono presenti delle *entry* finalizzate a descrivere aspetti legati alla comunicazione tra nodi. Alcune di queste *entry* devono essere obbligatoriamente implementate affinché il nodo si possa definire di tipo CANopen.
- *Manufacturer Specific*: questa sezione è a cura del produttore e viene utilizzata per personalizzare l'applicazione svolta dal nodo.
- *Device Profile Parameters*: in questa sezione sono presenti i parametri di nodi che svolgono applicazioni standardizzate da CIA (vedere sezione successiva).

In [11] è possibile trovare informazioni dettagliate sull'OD e sulle entry che lo compongono.

Indirizzo	Descrizione
0000h	Riservato
0001h – 0FFFh	Data Types
1000h – 1FFFh	Communication Entries
2000h – 5FFFh	Manufacturer Specific
6000h – 9FFFh	Device Profile Parameters
A000h – FFFFh	Riservato

Tabella 4.8: Struttura dell'Object Dictionary.

4.2.3 Device Profiles

L'OD è formato da un numero piuttosto elevato di *entry* e nel caso si dovesse ricercare una specifica entry il tempo impiegato sarebbe non trascurabile. Inoltre, alcune applicazioni sono piuttosto standard e non necessitano di particolari personalizzazioni. L'uso di *Device Profiles* permette di aggirare questi problemi in quanto un nodo (master) che tenta di accedere all'OD di un altro nodo (*slave*) può leggere l'*entry* denominata *Identity Object*, presente all'indirizzo 1018h nella sezione *Communication Entries*, ottenendo le informazioni sul *Device Profile*. Supponendo che il nodo master conosca i diversi *Device Profile*, una volta riconosciuto il tipo di profilo saprà esattamente dove sono collocate le *entry* ed il tipo di *entry*.

CIA definisce una serie di profili standard tipicamente denominati DS4xx oppure DSP4xx.
Se il profilo standard non dovesse essere sufficiente si può sempre ricorrere alla personalizzazione mediante *entry* collocate nella sezione *Manufacturer Specific* dell'OD.
In conclusione, l'uso di *Device Profiles* è vantaggioso per gli aspetti presentati ma anche per la portabilità e l'intercambiabilità dei nodi.

4.2.4 Electronic Data Sheets e Device Configuration Files

L'implementazione ed il mantenimento dell'OD può essere piuttosto difficile per il numero di entry che la tabella potrebbe contenere. Per risolvere tale problema esistono dei file, detti *Electronic Data Sheets* (EDS), appositamente ideati per descrivere l'OD elettronicamente. Specifici programmi possono leggere tali file al fine di implementare l'OD, modificarlo, trasmetterlo, ecc.
L'EDS si presenta come un listato di informazioni e può essere modificato manualmente, tuttavia l'uso di un programma ad-hoc permette di semplificare l'operazione.
Il *Device Configuration File* (DCF) viene utilizzato per contenere i valori (minimo, massimo, default, ecc.) di ogni *entry* dell'OD.

4.2.5 Comunicazione tra Object Dictionary

A questo punto è bene chiarire le diverse tipologie di indirizzi utilizzati nel CANopen:

- *Index* e *Subindex* dell'OD: sono gli indirizzi usati per identificare una specifica *entry* (presentati precedentemente).
- *COB ID*: si tratta dell'indirizzo dei messaggi CAN 2.0A (ID nei capitoli precedenti) usati per la comunicazione tra OD.
- *Node ID*: è l'indirizzo utilizzato per identificare un nodo. Il range ammesso è 0-127.

Il *Communication Object Identifier* (COB ID) assume valori diversi a seconda della funzione che svolge il messaggio con tale identificatore. Il range possibile (0-2^{11} oppure 0-800h) è suddiviso in più sezioni, ognuna delle quali rappresenta una differente funzione svolta dal messaggio CAN. In Tabella 4.9 sono presenti solo alcuni degli indirizzi possibili, una lista completa è presente in [11].

COB ID	Funzione del messaggio	Regola di costruzione
000h	Network Management	-
080h	Sync	-
081h-0FFh	Emergency	80h + Node ID
181h-1FFh	1st transmit PDO	180h + Node ID
201h-27Fh	1st receive PDO	200h + Node ID
581h-5FFh	Transmit SDO	580h + Node ID
601h-67Fh	Receive SDO	600h + Node ID

Tabella 4.9: Alcuni degli indirizzi standard.

4.2.5.1 Service Data Objects

Un primo modo per accedere all'OD di un nodo è quello di utilizzare un *Service Data Object* (SDO), ovvero dei messaggi CAN 2.0A la cui struttura è standard.

Ogni nodo implementa, oltre all'OD, un server che gestisce le richieste di lettura e scrittura del proprio OD generate da un nodo master (client). Le richieste di lettura e scrittura si svolgono mediante messaggi SDO (Tabella 4.1).

Per implementare una comunicazione punto-punto tra due nodi servono due messaggi SDO: uno per la trasmissione (*Receive SDO* per il server) ed uno per la ricezione (*Transmit SDO* per il server). Le regole per ottenere l'ID dell'SDO (COB ID) sono presente in Tabella 4.9 e permettono di ottenere 127 canali dal client verso il server e altri 127 canali in direzione opposta.

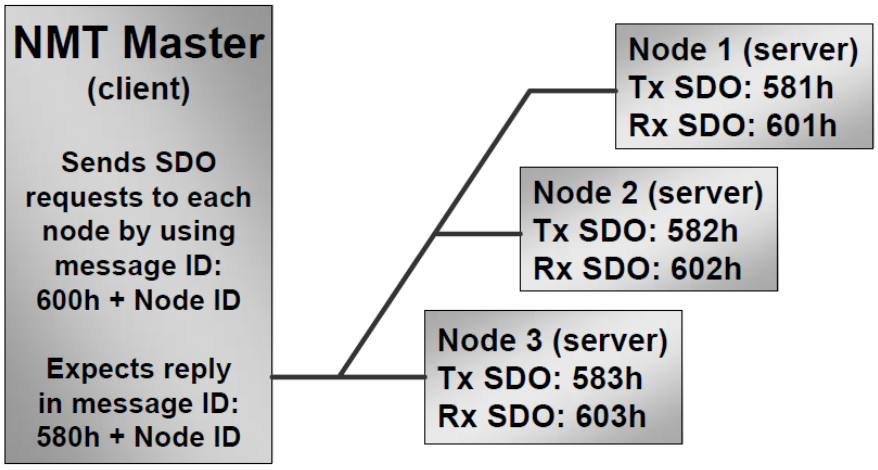


Figura 4.1: Comunicazione mediante SDO.

Il *Data Field* del messaggio CAN, di cui si riporta una rappresentazione per blocchi di singoli byte in Tabella 4.10, è mappato come segue:

- *Byte 1*: viene detto *specifier* e contiene informazioni di controllo come il tipo di messaggio (lettura, scrittura o errore), il numero di byte trasmessi ed il tipo di trasmissione che può coinvolgere più messaggi (*segmented transfer*) oppure un singolo messaggio (*expedited transfer*).
- *Byte 2-4*: trasferiscono *Index* e *Subindex* della entry. L'ordine di trasmissione è quello presentato in Tabella 4.10.
- *Byte 5-8*: utilizzati per trasferire dati aggiuntivi se necessario. Questi possono essere trasferiti in soli 4 byte (*expedited transfer*) oppure possono richiedere più messaggi (*segmented transfer*).

Specifier	Index (0-7)	Index (8-15)	Subindex	Data			

Tabella 4.10: Struttura del Data Field di un SDO.

L'SDO fornisce un semplice servizio di scrittura e lettura dell'OD; tuttavia, questo sistema di comunicazione è abbastanza inefficiente.

4.2.5.2 Process Data Objects

La comunicazione mediante SDO per quanto semplice non permette di trasferire dati in modo rapido in quanto è sempre necessario il comando da parte di un nodo master. Un SDO trasferisce una sola *entry* rendendo la comunicazione inefficiente per il trasferimento di molti dati. In alternativa è possibile utilizzare un servizio di comunicazione detto *Process Data Object* (PDO).

Esistono due tipologie di PDO: *Transfer PDO* (TPDO), se i dati sono prodotti dal nodo, oppure *Receive PDO* (RPDO), se i dati sono ricevuti dal nodo. Per ogni PDO esistono dei parametri, divisi tra *Communication Parameters* e *Mapping Parameters*, salvati nella sezione *Communication Entries* dell'OD (in particolare occupano gli indirizzi 1400h-1BFFh dell'OD). In aggiunta, esistono due tipi di dato complesso dedicati ai parametri del PDO detti PDO_COMMUNICATION_PARAMETER e PDO_MAPPING, descritti rispettivamente all'indirizzo 0020h e 0021h (sezione *Data Types*).

La trasmissione di un PDO può avvenire in quattro modi differenti:

- *Event Driven*: prevede la trasmissione di un PDO ogni qual volta si determini un evento nei dati trasmessi dallo specifico PDO. Nel CANopen non tutti i cambiamenti di stato dei dati generano eventi ma dipende da come è configurato il messaggio. Questo sistema può dar luogo ad un problema: un dato che cambia in maniera continuativa nel tempo genera molti PDO che porterebbero alla saturazione della banda disponibile. Esiste una semplice soluzione: impostare un tempo minimo tra due PDO, detto *Inhibit Timer.*
- *Time Driven*: il PDO viene trasmesso ad intervalli di durata prefissata. Ogni nodo presenta un timer, detto *Event Timer*, dedicato alla generazione degli intervalli. È un metodo poco efficiente in quanto si utilizzano risorse per spedire dati anche in assenza di eventi, tuttavia, permette di ottenere latenze ridotte.
- *Individual Polling*: il PDO è trasmesso in seguito ad una procedura di richiesta detta *Remote Request*. In genere non è molto usato.
- *Synchronized Polling*: Sfrutta un segnale di sincronismo globale per trasmettere nello stesso momento dati da più nodi. È vantaggioso per tutte quelle applicazioni dove è necessario conoscere il valore di più variabili nello stesso momento.

I *Communication Parameters* sono dei tipi di dato complessi che trasmettono informazioni come il COB ID del messaggio PDO, l'*Inhibit Time*, il numero di *entry* che verranno trasmesse nel *Data Field*, ecc.
I *Mapping Parameters*, invece, definiscono quali *entry* devono essere trasmesse in un PDO. Le *entry* possono appartenere ad OD differenti.

4.2.6 Network Management

Ogni nodo CANopen deve implementare una macchina a stati per la gestione del proprio stato operativo (*NMT state machine*). Un nodo, infatti, deve poter transitare tra quattro possibili stati: *Initialization*, *Pre-operational*, *Operational* e *Stopped*.
Lo stato *Initialization* si raggiunge non appena il nodo viene alimentato e permette a quest'ultimo di inizializzare i parametri interni. Se un messaggio di *boot-up* viene trasmesso con successo, si transita nello stato *Pre-operational* dove le comunicazioni sono limitate a soli messaggi SDO. Nello stato *Operational* sono abilitate le comunicazioni mediante messaggi PDO, perciò, il nodo può trasferire dati in grandi quantità. Infine, nello stato *Stopped* tutte le comunicazioni sono interrotte.

Le transizioni sono gestite attraverso degli appositi messaggi, inoltre, un nodo master può forzare delle transizioni di stato di un nodo slave attraverso messaggi dedicati (*NMT master message*).

4.2.7 Esempio

In questa sezione si analizza una possibile soluzione di collegamento di un sensore di temperatura facente uso del protocollo CANopen, attualmente disponibile in commercio [12]. Innanzitutto, il datasheet presenta le specifiche relative all'interfaccia CAN: in Figura 4.2 si può notare che il mezzo trasmissivo è quello richiesto dalla norma ISO 11898 a conferma che i primi due livelli del modello OSI adottati dal CANopen coincidono con quelli del protocollo CAN. Viene anche specificato il *Device Profile* utilizzato, ovvero il DS404 di CIA.

CAN Interface	
Physical layer	2-wire interface, 5 V level according to ISO 11898 Protected against short-circuit
Max. Bit rate	1 Mbit/s
Signal rise time	Bit rate < 125 kbit/s 12 V/µs (without bus) Bit rate ≥ 125 kbit/s > 24 V/µs (without bus)
Bus termination	External
Protocol	CANopen DS301, Device Profile DS404

Figura 4.2: Specifiche relative all'interfaccia CAN.

Il profilo DS404 (*Device Profile for Measuring Devices and Closed-Loop Controllers*), visionabile in parte in Figura 4.3, è un profilo utilizzabile da una vasta gamma di dispositivi, perciò, alcune *entry* non risultano necessarie per una specifica applicazione e possono essere eliminate (si noti la distanza tra le *entry* di Figura 4.3 a causa della mancata implementazione di alcuni elementi).

Verificata la compatibilità hardware le operazioni da svolgere per configurare il dispositivo sono le seguenti:

1. Operazioni preliminari;
2. Impostare la comunicazione;
3. Modificare l'*Object Dictionary*.

3.4 **Object dictionary: Device profile**

Index (HEX)	Sub Index	Name	Type	Access	Default	Comment
6110		Ai_Sensor_Type				
	00	Number of entries	Unsigned8	ro	1	
	01	Ai_Sensor_Type_1	Unsigned16	ro	100	100 = temperature sensor
6124		Ai_Input_Offset				
	00	Number of entries	Unsigned8	ro	1	
	01	Ai_Input_Offset_1	Float32	rw	0	Temperature offset; will be added to the current temperature value; Min./max. value=min./max. temperature of DST T92C
6125		Ai_Input_Autozero				
	00	Number of entries	Unsigned8	ro	1	
	01	Ai_Input_Autozero_1	Unsigned32	wo		Autozero for temperature 0x6F72657A (ASCII: "zero")

Figura 4.3: Specifiche relative all'Object Dictionary, sezione Device Profile.

4.2.7.1 Operazioni preliminari

Affinché un nodo soddisfi le specifiche CANopen è necessario che implementi un sistema di monitoraggio detto *heartbeat*: ogni nodo slave deve trasmettere un messaggio di un byte contenente lo stato del nodo stesso. Il messaggio viene trasmesso con periodicità impostabile tramite la *entry* con indirizzo 1017h (*Producer heartbeat time*). In aggiunta, un nodo può monitorare lo stato di un secondo nodo impostando l'*entry* con indirizzo 1016h (*Consumer heartbeat time*). Nell'esempio in analisi sono possibili entrambe le funzioni e le relative *entry* sono visibili in Figura 4.4.

Index (HEX)	Sub Index	Name	Type	Access	Default	Comment
1016		Consumer heartbeat time				
	00	Number of entries	Unsigned8	ro	1	
	01	Consumer heartbeat time	Unsigned32	rw	0	
1017	00	Producer heartbeat time	Unsigned16	rw	0	

Figura 4.4: Sezione dell'Object Dictionary relativa al monitoraggio dei nodi.

La gestione della rete avviene da parte di un nodo master, il quale, ad esempio, verifica che il messaggio di *heartbeat* sia presente. In questo esempio si suppone che tale nodo sia già presente e configurato.

Il collegamento di un nodo all'interno della rete prevede una serie di operazioni preliminari: innanzitutto, se possibile, è necessario allineare il bit rate del nodo a quello adoperato nella rete. In Figura 4.5 si riporta la sezione *Manufacturer specific profile* del dispositivo [12]; si può notare che è possibile modificare il *bit rate,* l'ID del nodo e molte altre informazioni. Tutte le *entry* presenti in tale sezione dell'OD sono definite dal costruttore e non presenti nel profilo standard DS404.

3.3 Object dictionary: Manufacturer specific profile

Index (HEX)	Sub Index	Name	Type	Access	Default	Comment
3000	00	Temperature overflow counter	Unsigned16	rw	0	Counter for overtemperature
3001	00	Temperature underflow counter	Unsigned16	rw	0	Counter for undertemperature
4E00		Manufacturer serial number				
	00	Number of entries	Unsigned8	ro	4	
	01	Serial number - Part 1	Unsigned8	ro		
	02	Serial number - Part 2	Unsigned32	ro		
	03	Serial number - Part 3	Unsigned16	ro		
	04	Serial number - complete	Visible_ string	ro		00.000000.0000
4F00	00	Bit rate	Unsigned8	rw	4	[1] see following table. Changes take effect after reset node or power on
4F01	00	Node ID	Unsigned8	rw	1	1 – 127; Changes take effect after reset node or power on

Figura 4.5: Specifiche relative all'Object Dictionary, sezione Manufacturer specific profile.

4.2.7.2 Impostare la comunicazione

Un nodo può leggere autonomamente uno specifico TPDO trasmesso da un secondo nodo. Per ottenere tale automatismo è necessario collegare il TPDO con un RPDO del nodo ricevente, scrivendo l'indirizzo del TPDO (180h più l'ID del nodo trasmittente) all'interno di una delle *entry* dedicate agli RPDO presenti nell'OD del nodo ricevente (indirizzi da 1400h a 15FFh), precisamente nella *subentry* con *subindex* 01h.

In Figura 4.6 è presente un estratto della sezione *Communication Entries* del dispositivo in analisi. Il sensore di temperatura non riceve alcuna informazione, infatti, le *entry* relative agli RPDO (1400h – 17FFh) non sono implementate. Il sensore sfrutta un unico messaggio TPDO

per trasmettere l'informazione sulla temperatura di cui sono riportati *Communication Parameters* e *Mapping Parameters*.

In questo documento non si è trattato della struttura dei PDO, tantomeno delle regole di mappatura tra le entry di un TPDO e quelle di un RPDO (maggiori informazioni sono presenti alla fine del secondo capitolo in [11]). Tuttavia, risulta piuttosto semplice mappare un dato tra TPDO e RPDO: per trasmettere la temperatura come intero (1A00h, 01h) è sufficiente scrivere l'identificativo associato a tale dato (0x91300120) nella prima *subentry* disponibile di un RPDO *mapping* del nodo ricevente, tale RPDO deve essere precedentemente collegato al TPDO come illustrato precedentemente.

Index (HEX)	Sub Index	Name	Type	Access	Default	Comment
1800		Transmit PD01 parameter				
	00	Number of entries	Unsigned8	ro	5	
	01	COB ID used by PDO	Unsigned32	rw	0x181	(0x00000180 + Node-ID)
	02	Transmission type	Unsigned8	rw	0x01	Only 0x01 (sync) or 0xFF (async) with delta and/or event timer
	03	Inhibit time	Unsigned16	rw	0	
	04	Reversed	Unsigned8	rw	0	
	05	Event timer	Unsigned16	rw	1000	1000:default value of DS404
1A00		Transmit PDO1 mapping				
	00	Number of entries	Unsigned8	rw	2	
	01	PDO mapping for the 1. application object to be mapped	Unsigned32	rw	0x91300120	Temperature as int32: 0x91300120 Temperature as float32: 0x61300120 Status temperature as uint8: 0x61500108 Meaning of status bits (if set): Bit 0: temperature value invalid Bit 1: positive overload Bit 2: negative overload
	02	PDO mapping for the 2. application object to be mapped	Unsigned32	rw	0x61500108	
	03	PDO mapping for the 3. application object to be mapped	Unsigned32	rw	0	
	04	PDO mapping for the 4. application object to be mapped	Unsigned32	rw	0	

Figura 4.6: Specifiche relative all'Object Dictionary, sezione Communication Entries.

Tutte le operazioni descritte possono essere agevolmente svolte per mezzo di programmi appositi, i quali permettono di collegare RPDO e TPDO automaticamente, aggiornare le *entry* tramite file ESD, ecc.

4.2.7.3 Modificare l'Object Dictionary

Alcune applicazioni non standard potrebbero richiedere delle *entry* aggiuntive, non presenti nei profili standard forniti da CIA. Partendo da una base standard è possibile modificare l'OD, aggiungendo le entry necessarie nella sezione *Manufacturer specific profile.* Naturalmente è possibile non implementare alcune entry.

Nell'esempio analizzato sono state apportate alcune modifiche all'OD ma finalizzate principalmente alla comunicazione; ulteriori modifiche sono raramente necessarie vista la semplicità del dispositivo.

Conclusioni

Lo standard CAN è un protocollo di comunicazione molto usato nell'automazione industriale per reti di campo ed in ambito automotive per semplificare cablaggi. Il protocollo CAN differisce per parecchi aspetti da altri protocolli di comunicazione: l'assenza di indirizzi per i nodi, il sistema di risoluzione delle collisioni, il meccanismo di selezione dei messaggi coerenti con le funzionalità del nodo e l'autoesclusione di quest'ultimo dal bus in caso di malfunzionamento sono solo alcune delle peculiarità del protocollo, tutte finalizzate all'affidabilità della comunicazione. La comparsa di un numero sempre maggiore di standard per livelli superiori, come J1939 e CANopen, garantisce longevità e stabilità del protocollo. Negli ultimi anni, però, l'incessante richiesta di protocolli in grado di trasmettere informazioni in quantità e velocità sempre maggiori ha favorito protocolli basati su interconnessioni Ethernet, i più diffusi sono PROFINET ed EtherCAT e rientrano nella categoria *Industrial Ethernet*.

Il protocollo CAN, nonostante la concorrenza presente nel mercato, continua ad essere la miglior scelta in specifici settori qualora sia necessario un protocollo semplice ma efficace.

L'ultima innovazione in ambito CAN è rappresentata dal protocollo CAN XL, nato per colmare il divario in termini di velocità tra CAN classico ed *Industrial Ethernet.* Il protocollo CAN XL può rappresentare un possibile futuro sviluppo per questa tesi in quanto attuale concorrente dell'*Industrial Ethernet* [14].

Bibliografia

[1] S. Gai, P. L. Montessoro, and P. Nicoletti, "IL MODELLO ISO/OSI" in *Reti Locali: Dal Cablaggio all'internetworking*, Scuola Superiore G. Reiss Romoli, 1995.

[2] S. Buso, *Introduzione alle applicazioni industriali di MICROCONTROLLORI E DSP*. Bologna: Esculapio, 2018.

[3] D. M. Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and using the Controller Area Network Communication Protocol: Theory and Practice*. Berlin: Springer, 2014.

[4] "MCP2515, Stand-Alone CAN Controller with SPI Interface" Microchip Technology. Available: https://ww1.microchip.com/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf.

[5] S. Broyles, "A System Evaluation of CAN Transceivers" Texas Instruments, Jul-2018. Available: https://www.ti.com/lit/an/slla109a/slla109a.pdf.

[6] "RM0364 Reference manual" STMicroelectronics, Oct-2015. Available: https://www.st.com/reference_manual/rm0364-stm32f334xx.pdf.

[7] *CAN in Automation (CiA): CAN knowledge*. [Online]. Available: https://www.can-cia.org/can-knowledge/. [Accessed: Mar-2023].

[8] J. Griffith, "Why are termination networks in can transceivers so important?" *TI E2E*. [Online]. Available: https://e2e.ti.com/blogs_/b/industrial_strength/posts/the-importance-of-termination-networks-in-can-transceivers. [Accessed: Mar-2023].

[9] J. Griffith, "Learn the inner workings of a CAN bus driver and how to debug your system" *TI E2E*. [Online]. Available: https://e2e.ti.com/the-inner-workings-of-a-can-bus-driver. [Accessed: Feb-2023].

[10] "Controller Area Network (can bus) protocol" *Kvaser*. [Online]. Available: https://www.kvaser.com/can-protocol-tutorial/. [Accessed: Feb-2023].

[11] O. Pfeiffer, A. Ayre, and C. Keydel, *Embedded Networking with CAN and CANopen*, First Edition. Copperhill Technologies Corporation, 2003.

[12] "Operation guide - Temperature transmitter CANopen DST T92C" Danfoss. Available: https://assets.danfoss.com/documents/67324/AQ267742448328en.

[13] "SAE J1939 know-how" *Vector*. [Online]. Available: https://www.vector.com/int/en/know-how/protocols/sae-j1939/#c150199. [Accessed: Mar-2023].

[14] "CAN XL comparisons" bosch-semiconductors.com. [Online]. Available: https://www.bosch-semiconductors.com/can_xl_vs_10base-t1s_v2.pdf. [Accessed: Mar-2023].