Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Elettronica

ANALISI E APPLICAZIONE DEL PROTOCOLLO CAN ALL'AUTOMOTIVE

Relatore:

Prof. Daniele Vogrig

Laureando:

Andrea Brasiliani

Matricola: 1117331

ANNO ACCADEMICO 2021/2022

INDICE

INTRODUZIONE

Il continuo evolversi dell'elettronica nel secolo scorso ha prodotto sistemi di comunicazione sempre più efficienti per gestire la grande mole di dati trasmessa.

Lo scopo di questo elaborato è fornire una breve analisi di uno dei protocolli che si è distinto in ambito industriale e soprattutto nei sistemi automobilistici.

Nato dall'esigenza degli ingegneri di sviluppare reti di comunicazione per i dispositivi impiegati nelle industrie, il protocollo CAN ha trovato largo impiego nel settore dell'automotive proprio per le sue caratteristiche di robustezza, velocità e di controllo della gestione degli errori.

Verranno descritte le caratteristiche principali che fanno di questo protocollo uno dei più dinamici in senso di intervento e facilità di programmazione e che lo rendono adattabile ad ogni tipo di necessità, le realtà di esperti che lo utilizzano e il modo in cui viene applicato alla soluzione dei problemi che insorgono nelle reti di telecomunicazione del nuovo millennio.

1. La comunicazione seriale: introduzione al CAN

La trasmissione seriale è il modo ad ora più semplice ed economico per far comunicare i microcontrollori o DSP (*Digital Signal Processor*) con le proprie periferiche.

Questo modello trova ampia applicazione in ambito industriale dove viene sfruttato per allestire sistemi di comunicazione tra anche centinaia di dispositivi, preferendolo al metodo della comunicazione in parallelo che se pur più veloce, richiede maggiore cura necessitando di circuiti hardware più dettagliati con conseguenti maggiorazioni dei costi di fabbricazione.

I principali protocolli che si basano sulla comunicazione seriale sono il CAN (*Controller Area Network*), l'I2C (*Inter Integrated Circuit*) o l'SPI (*Serial Peripheral Interface*).

Nella comunicazione seriale si distinguono due modalità di funzionamento: sincrona e asincrona. La comunicazione sincrona permette velocità più elevate, ma richiede una maggiore complessità del supporto fisico. Al contrario, quella asincrona richiede un collegamento di complessità minima ma concede velocità minori. Normalmente, le unità periferiche presenti a bordo dei microcontrollori permettono la gestione di entrambe le modalità seppur la modalità asincrona venga preferita tutte le volte che non vi siano particolari esigenze di velocità.

1.1. La comunicazione seriale asincrona

La comunicazione seriale asincrona avviene senza che i clock del trasmettitore e del ricevitore siano sincronizzati. La convenzione adottata per far sì che il ricevitore estragga il sincronismo dalla stessa linea riservata al trasferimento dei dati è di far precedere ad ogni dato (8 bit) un bit di START e di farlo seguire da un bit di STOP. La semplicità di configurazione di una rete di comunicazione asincrona fa di questa un vero e proprio standard per le trasmissioni da punto a punto a bassa velocità.

La configurazione del canale di comunicazione si basa su un insieme di pochi parametri standard:

- Numero dei bit di start (1).
- Numero dei bit dati (da 5 a 8).
- Numero dei bit di parità (1 o 0).
- Numero dei bit di stop (1, 1.5, 2), dove il valore tra parentesi è in funzione del periodo di clock: es. 1.5 indica che il valore del segnale rimarrà invariato per un periodo e mezzo di clock.
- Velocità del canale: 1200,2400,4800,9600,19200 bit o baud al secondo(bps).

L'impostazione di questi parametri avviene configurando alcuni registri dell'unità periferica destinata alla gestione della comunicazione, UART, Universal Asynchronous Receiver Transmitter.

Il bus CAN (Controller Area Network) è uno standard di comunicazione seriale asincrona, pensato per la trasmissione dati in ambiente industriale, soprattutto per applicazioni real-time relative ai sistemi di automotive.

2. Origini e sviluppo del CAN bus

Oggi, quasi ogni nuova autovettura prodotta in Europa è dotata di almeno una rete CAN, utilizzato anche in altri tipi di veicoli quali treni e navi. Una standardizzazione del CAN è stata sviluppata anche per i camion: il collegamento in rete tra camion e rimorchio è standardizzato come ISO 11992, usato in Europa dal 2006.

All'inizio degli anni '80, gli ingegneri della Bosch stavano valutando i sistemi di bus seriali esistenti riguardo al loro possibile utilizzo nelle autovetture. Poiché nessuno dei protocolli di rete disponibili era in grado di soddisfare le esigenze espresse dagli ingegneri automobilistici, Uwe Kiencke[1] iniziò lo sviluppo di un nuovo sistema di bus seriale nel 1983.

Gli ingegneri della Mercedes-Benz furono ben presto coinvolti nella fase di specificazione del nuovo sistema di bus seriale, così come Intel per fornire i semiconduttori.

Il professor Dr. Wolfhard Lawrenz dell'Università di Scienze Applicate di Braunschweig-Wolfenbüttel (oggi: Università di Scienze Applicate di Ostfalia), Germania, che era stato assunto come consulente, diede al nuovo protocollo di rete il nome ‘’Controller Area Network’’ (CAN).

Nel febbraio del 1986, Robert Bosch GmbH presenta il sistema di bus seriale CAN al congresso di Detroit della *Society of Automotive Engineers* (SAE).

2.1. I primi circuiti integrati CAN

Dopo il congresso di Detroit del 1986 seguirono molte presentazioni e pubblicazioni che descrivevano questo innovativo protocollo di comunicazione, finché a metà del 1987, due mesi prima del previsto, Intel consegnò il primo chip controller CAN, l'82526. Era la primissima implementazione hardware del protocollo CAN. In soli quattro anni, un'idea era diventata realtà.

Poco tempo dopo, Philips Semiconductors introdusse l'82C200. Questi due primi antenati dei controller CAN erano abbastanza diversi per quanto riguarda il filtraggio dell'accettazione e la gestione dei frame. Da un lato, il concetto FullCAN favorito da Intel richiedeva meno carico di CPU (unità di elaborazione centrale) dal microcontrollore collegato rispetto all'implementazione BasicCAN scelta da Philips. D'altra parte, il dispositivo FullCAN era limitato per quanto riguarda il numero di frame che potevano essere ricevuti. Il controller BasicCAN richiedeva anche meno silicio.

[1] Prof. Dr. Ing. Uwe Kiencke, Capo del Dipartimento di sviluppo di sistemi avanzati, Robert Bosch GmbH, nel periodo 1981-1987.
Fonte: https://www.iiit.kit.edu/kiencke.php

Nei controller CAN di oggi, viene implementato un misto di entrambi i concetti di filtraggio dell'accettazione e di gestione dei frame. Questo ha reso i termini BasicCAN e FullCAN obsoleti e fuorvianti.

La specifica CAN di Bosch (versione 2.0) è stata presentata per la standardizzazione internazionale nei primi anni '90. Dopo diverse controversie politiche, soprattutto per quanto riguarda la "*Vehicle Area Network*" (VAN) sviluppata da alcuni grandi produttori di automobili francesi, lo standard ISO 11898 è stato pubblicato nel novembre 1993. Oltre al protocollo CAN, ha anche standardizzato uno strato fisico per bit-rate fino a 1 Mbit/s.

In parallelo un modo a bassa potenza e tollerante ai guasti di trasmissione dati via CAN è stato standardizzato in ISO 11519-2. Questo non è mai stato implementato a causa di debolezze nello standard.

Nel 1995, lo standard ISO 11898 è stato modificato da un *addendum* che descriveva il formato di frame esteso usando un identificatore CAN a 29 bit. Sfortunatamente, tutte le specifiche e le standardizzazioni CAN pubblicate contenevano errori o erano incomplete.

Per evitare implementazioni CAN incompatibili, Bosch ha fatto in modo che tutti i chip CAN siano conformi al modello di riferimento Bosch CAN. Inoltre, l'Università di Scienze Applicate di Braunschweig/Wolfenbüttel, Germania, ha condotto test di conformità CAN per diversi anni, guidati dal Prof. Lawrenz. I modelli di test utilizzati sono basati sulla serie di standard ISO 16845 conformance test plan. Oggi, diverse case di prova offrono servizi di test di conformità CAN.

2.2. La standardizzazione del protocollo CAN

Il lavoro sperimentale e di ricerca condotto da Bosch e Lawrenz ha portato nel corso degli anni alla revisione delle specifiche CAN fino ad arrivare ad ottenerne la standardizzazione.

La norma ISO 11898-1 descrive il ‘’CAN data link layer’’, ISO 11898-2 standardizza il ‘’Non fault-tolerant CAN physical layer’’, e ISO 11898-3 specifica il ‘’Fault-tolerant CAN physical layer’’.

La serie ISO 11992 (interfaccia di camion e rimorchi) e la serie ISO 11783 (macchine agricole e forestali) specificano profili applicativi basati sull'approccio di rete SAE J1939.

Anche se il protocollo CAN è stato originariamente sviluppato per essere utilizzato nelle autovetture, le prime applicazioni provenivano da diversi segmenti di mercato. Specialmente nel nord Europa, il protocollo CAN è stato popolare fin dal suo esordio. In Finlandia, il produttore di ascensori Kone usava il bus CAN. L'ufficio di ingegneria svedese Kvaser suggerì CAN come protocollo di comunicazione all'interno delle macchine ad alcuni produttori di macchine tessili (Lindauer Dornier

e Sulzer) e ai loro fornitori. A questo proposito, sotto la guida di Lars-Berno Fredriksson, queste aziende fondarono il ‘’CAN Textile User's Group’’.

Entro il 1989 avevano sviluppato principi di comunicazione che hanno contribuito a formare l'ambiente di sviluppo ‘’CAN Kingdom’’.

Anche se il CAN Kingdom non è un alto livello di applicazione rispetto al modello di riferimento OSI (Open Systems Interconnection), può essere considerato come l'antenato dei protocolli di livello superiore basati sul CAN.

L’inizio degli anni ‘90 segna il momento giusto per fondare un gruppo di utenti per promuovere il protocollo CAN e per favorire il suo utilizzo in molte applicazioni. Nel gennaio del 1992, Holger Zeltwanger, a quel tempo editore della rivista VMEbus (editore Franzis), ha riunito utenti e produttori per stabilire una piattaforma neutrale per il miglioramento tecnico del protocollo CAN. Due mesi dopo, il gruppo internazionale di utenti e produttori ’’CAN in Automation’’ (CiA) fu ufficialmente fondato. Fu l’inizio della pubblicazione della Newsletter CAN.

La prima pubblicazione tecnica, rilasciata dopo poche settimane, riguardava lo strato fisico: il CiA raccomandava di usare solo transceiver CAN conformi alla ISO 11898.

Uno dei primi compiti del CiA era la specifica di un livello di applicazione CAN. Usando il materiale esistente di Philips Medical Systems, insieme all'aiuto di altri membri del CiA, fu sviluppato il ‘’CAN Application Layer’’ (CAL), chiamato anche *’’Green Book’’*.

Mentre si sviluppavano le specifiche utilizzando il CAN, uno dei compiti principali del CiA era quello di organizzare lo scambio di informazioni tra gli esperti CAN e quelli che volevano diventare più informati. Pertanto, dal 1994, si tiene la conferenza internazionale CAN (iCC).

Un altro approccio accademico è stato perseguito nella LAV: associazione tedesca dei veicoli agricoli. Dalla fine degli anni '80, era stato sviluppato un sistema di bus basato su CAN per veicoli agricoli (LBS). Ma prima che il lavoro potesse essere completato con successo, il comitato internazionale decise a favore di una soluzione statunitense, J1939 (ISO 11783). Questo profilo di applicazione, che è anche basato su CAN, è stato definito dai comitati della SAE Truck and Bus Association.

All'inizio del 2011, General Motors e Bosch hanno iniziato lo sviluppo di alcuni miglioramenti del protocollo CAN per quanto riguarda il *throughput* più elevato. L'industria automobilistica ha sofferto in particolare per il download di pacchetti software sempre più numerosi nelle unità di controllo elettronico (ECU). Questo compito che richiede tempo doveva essere abbreviato da un sistema di comunicazione più performante.

Alla fine del 2018, CiA ha iniziato lo sviluppo di CAN XL, la terza generazione di protocolli di livello di collegamento dati basati su CAN. È stato avviato su richiesta di Volkswagen.

Il carico utile massimo (campo dati) del CAN XL è di 2048 byte. Un'altra nuova caratteristica è la separazione della funzione di priorità (campo di priorità a 11 bit) e la funzione di indirizzo/contenuto (campo di accettazione a 32 bit). Questo include anche un nuovo approccio di collegamento dei media fisici utilizzando la codifica PWM (Pulse Width Modulation) invece della tradizionale codifica NRZ (Not Return to Zero).

A seguito si presenta un estratto della norma ISO (11898):

*''**ISO 11898-1:2015** specifies the characteristics of setting up an interchange of digital information between modules implementing the CAN data link layer. Controller area network is a serial communication protocol, which supports distributed real-time control and multiplexing for use within road vehicles and other control applications.*

(specifica le caratteristiche dell'impostazione di uno scambio di informazioni digitali tra moduli che implementano il livello di collegamento dati CAN. Controller Area Network è un protocollo di comunicazione seriale che supporta il controllo distribuito in tempo reale e il multiplexing per l'uso all'interno di veicoli stradali e altre applicazioni di controllo).

***ISO 11898-1:2015** specifies the Classical CAN frame format and the newly introduced CAN Flexible Data Rate Frame format. The Classical CAN frame format allows bit rates up to 1 Mbit/s and payloads up to 8 byte per frame. The Flexible Data Rate frame format allows bit rates higher than 1 Mbit/s and payloads longer than 8 byte per frame.*

(specifica il formato Classical CAN frame e il nuovo formato CAN Flexible Data Rate Frame. Il formato Classical CAN frame consente bit rate fino a 1 Mbit/s e payload fino a 8 byte per frame. Il formato frame Flexible Data Rate consente bit rate superiori a 1 Mbit/s e payload più lunghi di 8 byte per frame).

***ISO 11898-1:2015** describes the general architecture of CAN in terms of hierarchical layers according to the ISO reference model for open systems interconnection (OSI) according to ISO/IEC 7498-1. The CAN data link layer is specified according to ISO/IEC 8802-2 and ISO/IEC 8802-3.*

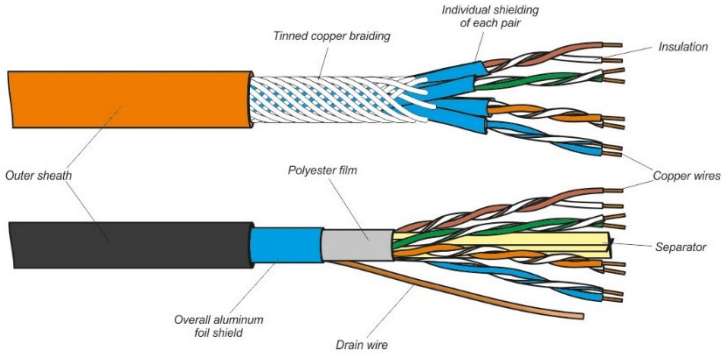
(descrive l'architettura generale di CAN in termini di livelli gerarchici secondo il modello di riferimento ISO per l'interconnessione di sistemi aperti (OSI) secondo ISO/IEC 7498 1. Il livello di collegamento dati CAN è specificato secondo ISO/IEC 8802 2 e ISO/IEC 8802 3).

***ISO 11898-1:2015** contains detailed specifications of the following: logical link control sub-layer; medium access control sub-layer; and physical coding sub-layer.'' (contiene specifiche dettagliate di quanto segue: sottolivello di controllo del collegamento logico; sottolivello di controllo accessi medio; e sottolivello di codifica fisica).*

3. Il bus CAN

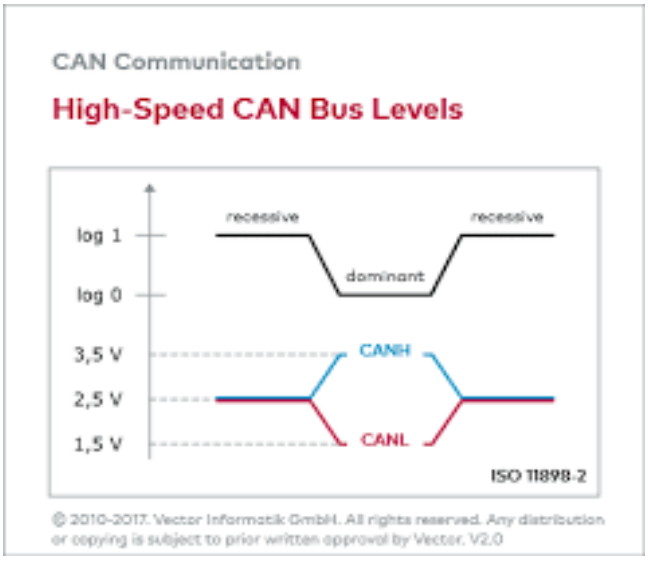
3.1. RS485: il layer fisico

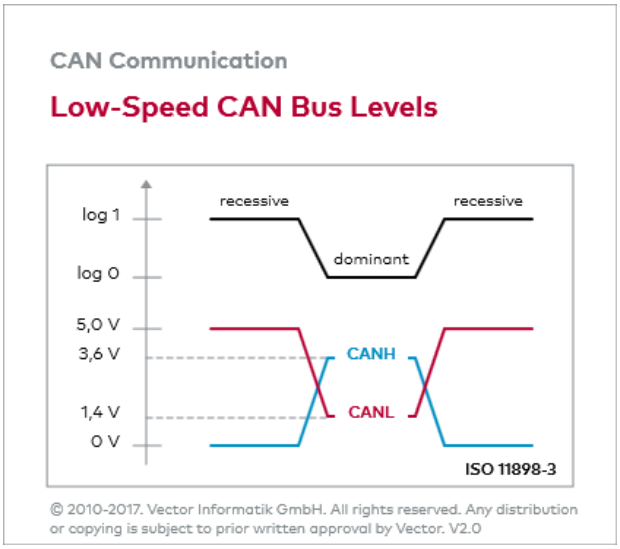
Il CAN bus fonda la sua struttura sullo standard fisico RS485, che definisce il numero di conduttori richiesti, i livelli di tensione e le modalità di pilotaggio delle linee. Standard come l'RS422 o l'RS485 sono stati ideati per garantire una buona qualità della comunicazione seriale anche in ambienti industriali dove il tasso d'inquinamento elettromagnetico è particolarmente elevato. Per questa ragione il bus di collegamento RS485 è generalmente formato da un cavo con due conduttori attorcigliati (doppino o *twisted-pair*) che danno vita ad una schermatura di tali disturbi.



Questo supporto fisico affida così la trasmissione dell'informazione all'amplificazione di modo *differenziale* eliminando i disturbi provenienti da quella di modo comune (per la quale il segnale portante verrebbe interferito appunto dalla media dei due valori presenti all'ingresso del bipolo che è ingresso dell'amplificatore comparatore posto come entrata del dispositivo di comunicazione).

I due fili intrecciati sono associati a due canali di comunicazione attraverso i quali trascorre il segnale: Can High che assume valori di tensione maggiori del suo alter ego, Can Low. L'ampiezza dei due segnali infatti è diametralmente opposta al valore medio di 2,5V ed è rispettivamente 3,5V per l'High e1,5V per il Low.





Un bus formato da due soli fili è un bus detto *half-duplex*: ciò non toglie la possibilità di raddoppiare il bus portandolo a quattro fili e ottenendo un bus *full-duplex* fornendo così ai dispositivi la possibilità di trasmettere e ricevere allo stesso tempo.

Un buon bus RS485 lo si ottiene restando tra i 400 e i 500m di lunghezza massima. Ovviamente riducendo la velocità del bus è possibile arrivare anche a 1200m.

Poiché i moduli collegati sulla rete risentono anche notevolmente delle differenze di massa che si vengono a creare, soprattutto su reti molto lunghe, è consigliabile usare degli accoppiatori RS485 separati galvanicamente. La maggiore immunità al rumore così ottenuta consente al canale di comunicazione di raggiungere velocità di trasmissione anche relativamente elevate.

Baud Rate	Capacità tot.max del cavo (pF)
1.200	400.000
2.400	200.000
4.800	100.000
9.600	50.000
19.200	25.000
38.400	12.000
57.600	8.000
115.200	4.000

Poiché il CAN è stato inizialmente progettato per l'uso in auto, per ottenere l'accettazione del mercato si richiedeva una gestione efficiente degli errori.

Con il rilascio della versione 2.0B della specifica CAN, la velocità massima di comunicazione (bit rate) è stata aumentata a 1Mbit/s, 8 volte più veloce rispetto alla specifica della versione 1.0. A questa velocita, solo la maggior parte dei parametri tempo-critici possono essere trasmessi in serie, senza preoccupazioni di latenza.

Il bus deve essere provvisto di resistenze di terminazione da 120 Ω per rimuovere le riflessioni del segnale all'estremità del bus.

Per capire se dobbiamo mettere i terminatori di linea di trasmissione possiamo usare il seguente parametro:
se la durata di un singolo bit trasmesso (SBTx) è maggiore di almeno 10 volte del tempo (Tp) che il segnale impiega a percorrere tutta la linea allora la terminazione non serve.
Non Servono Terminazioni se: **SBTx > (10 * Tp).**
Assumendo che:

1. la velocità di propagazione di un segnale in un cavo elettrico è circa 2/3 della velocità della luce(c);
2. ciascun cavo è caratterizzato da parametri specifici quali: capacità, induttanza, resistenza che possiamo riassumere nell'impedenza del cavo Zo;

Facciamo un esempio pratico considerando:

1. velocità 9600 baud;
2. lunghezza rete 1200 m;

Con questi parametri il tempo che il segnale impiega per la sua propagazione sarà:

$$t = \frac{1}{0.66 \cdot c} \cdot 1.2 = 6\mu s$$

9600 baud (9600 bit/s) per cui si calcola il tempo del singolo bit che è:

1 / 9600 = 0,000104166 s che equivale a 104,166 us (microsecondi)

Ritornando alla formula SBTx **> (10** * t**)** si ha:

104,166 > (10 * 6)

104,166 > 60

In questo caso si può *non* utilizzare la terminazione di linea.

Se invece fosse necessario terminare la linea, quello che si fa è mettere sul nodo più lontano dal master una resistenza a carbone collegata tra A e B il cui valore deve essere pari all'impedenza caratteristica della linea ZL.

Inoltre, l'intreccio dei due conduttori rende possibile cancellare quasi totalmente le tensioni indotte su ciascun filo da eventuali campi magnetici variabili, dato che due spire adiacenti avranno tensioni uguali in modulo ma opposte. La tensione totale quindi indotta su ciascuno dei fili sarà pressoché nulla.

3.2. Message Frame: i messaggi CAN

Con il termine *multicast* nelle reti di calcolatori, si indica la distribuzione simultanea di informazione verso un gruppo di destinatari, cioè la possibilità di trasmettere la medesima informazione a più dispositivi finali, senza dover indirizzare questi ultimi singolarmente e senza avere, quindi, la necessità di duplicare per ciascuno di essi l'informazione da diffondere.

Il CAN ne rappresenta una vera e propria istanza con una particolare caratterizzazione dovuta alla specifica modalità con cui viene gestito la contesa e l'accesso al bus.

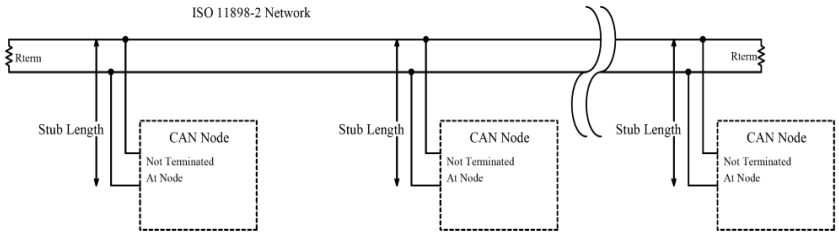
In aggiunta a questo, il protocollo CAN ha un elenco esaustivo degli errori che è in grado di rilevare per assicurare l'integrità dei messaggi. I nodi CAN hanno la capacità di determinare condizioni di fallimento e la transizione ai diversi modi basato sulla gravità dei problemi incontrati. Essi hanno anche la capacità di riconoscere i brevi disturbi dai fallimenti permanenti e modificare di conseguenza la loro funzionalità.

I nodi CAN possono transitare dal funzionamento di nodo normale (in grado di trasmettere e ricevere messaggi normalmente), allo spegnimento completo (*bus-off*), a seconda della gravità degli errori riscontrati. Questa funzione è chiamata *Fault Confinement*. Nessun nodo CAN in fallimento è in grado di monopolizzare tutta la larghezza di banda di *network* perché i fallimenti sono limitati ai vari nodi che saranno spenti prima di sovraccaricare la rete. Questo è molto potente perché il confinamento dell'errore garantisce la larghezza di banda per sistemi critici.

Esistono 2 modalità possibili, la normale e la *high-speed*: la prima con *failure detection*, la seconda senza. Le tensioni di alimentazione del sistema associate sono rispettivamente a 5v o 3.3v, adatte ad applicazioni a 12v e 24v anche se le tensioni che i suoi due canali, CAN HIGH e CAN LOW, possono tollerare vanno da -40v a +40v.

Il protocollo CAN è un protocollo basato sui messaggi, non sull'indirizzamento. Ciò significa che i messaggi non vengono trasmessi da un nodo ad un altro nodo sulla base di indirizzi. Il messaggio stesso contiene la priorità e il contenuto dei dati trasmessi ed un'etichetta che lo rende certificato per

tale rete di comunicazione a patto che il numero dell'ID sia già stato configurato e conosciuto dagli altri dispositivi partecipanti; le varie unità interfacciate al bus recepiscono tutti e soli i messaggi il cui contenuto sia rilevante per la funzione che svolgono e a tale scopo nella programmazione delle unità di controllo del bus viene , infatti, definita una serie di filtri sui messaggi che permettono l'implementazione di questo meccanismo di selezione.



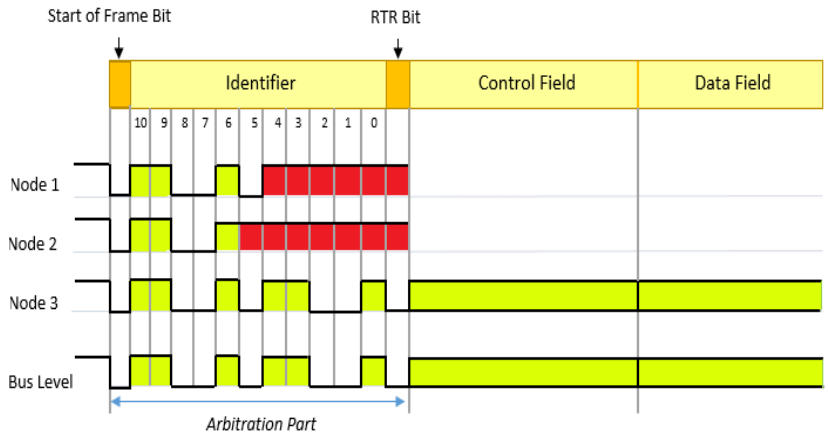
Tutti i nodi del sistema ricevono i messaggi trasmessi sul bus e danno la notifica se il messaggio è stato ricevuto correttamente (procedura *aknowledgement*).

Spetta a ogni nodo del sistema decidere se il messaggio ricevuto deve essere immediatamente scartato o tenuto per essere processato.

Tutti questi aspetti che indicizzano la natura della rete come una *multicast*, ribadendo che è un modello in cui non c'è alcun "gestore" o Master del bus ed ogni nodo ha la possibilità di iniziare una trasmissione (vedi paragrafo 3.1), fanno sorgere però la necessità di risolvere il problema della contesa per l'accesso al canale da parte di tutti i dispositivi.

Il protocollo di comunicazione CAN è un CSMA / CD. La sigla CSMA è l'acronimo di *Carrier Sense Multiple Access*. Ciò significa che ogni nodo sulla rete deve monitorare il bus per individuare un periodo di assenza di attività prima di provare a inviare un messaggio sul bus (procedura di *Carrier Sense*). Inoltre, una volta che questo periodo di non attività si verifica, ogni nodo sul bus ha una uguale possibilità di trasmettere un messaggio (*Multiple Access*). La sigla CD sta per *Collision Detection*. Se due nodi della rete iniziano a trasmettere allo stesso tempo, i nodi rileveranno la ''collisione'' e adottano i provvedimenti del caso.

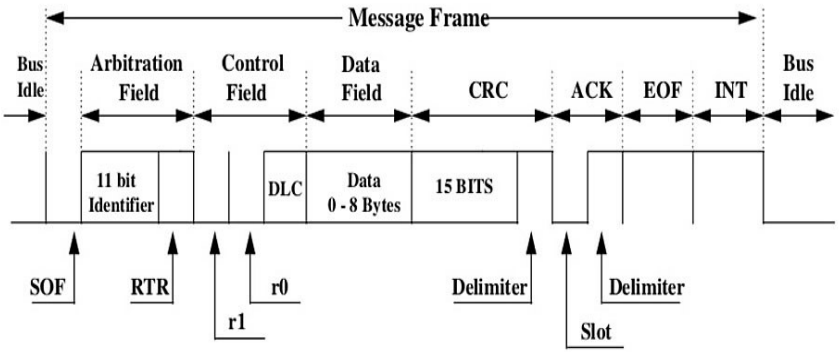
La soluzione adottata dai progettisti prevede che quando due o più unità si contendono il bus per la trasmissione di un messaggio, l'assegnazione avvenga in modo automatico, sulla base della priorità associata implicitamente ad ogni identificatore.



I valori binari più bassi prevalgono su quelli più alti in base ad una logica di tipo *wired-and*. Questo effetto si ottiene grazie alla scelta del livello logico basso come bit dominante, di quello alto come bit recessivo. Si afferma dunque sul bus il messaggio il cui identificatore ha il codice numericamente più basso: se un nodo trasmette un bit dominante e contemporaneamente un altro nodo trasmette un bit recessivo allora sul bus sarà presente quello dominate (dal punto di vista elettrico e di interfaccia si può pensare che il bit 1 sia forzato sul bus da un resistore di *pull-up* mentre lo zero dominante da un transistor con il *drain* sul bus).

Il nodo che si afferma sul bus CAN potrà trasmettere quattro tipi di messaggio:

- *Data Frame*
- *Remote Frame*
- *Overload Frame*
- *Error Frame*



Lo *Start Of Frame* (SOF) è un bit dominante (zero) che segue una fase di IDLE del bus nella quale questo si trova nello stato alto. Si può osservare come il messaggio si apra con il campo di 11 bit che rappresenta l'ID del messaggio seguito da 1 bit di *Remote Transmission Request* (RTR) che qualifica il messaggio come un *Data Frame* se il bit è basso e sottintende un invio di dati, come

Remote Frame se il bit è alto e corrisponde ad una richiesta di dati. Seguono 6 bit di controllo: i primi 2 bit sono riservati (sempre sullo stato basso), i 4 seguenti servono per indicare il numero di byte che verranno trasmessi nel caso si trattasse di un *Data Frame*. Il protocollo prevede l'invio di un massimo di 8 byte.

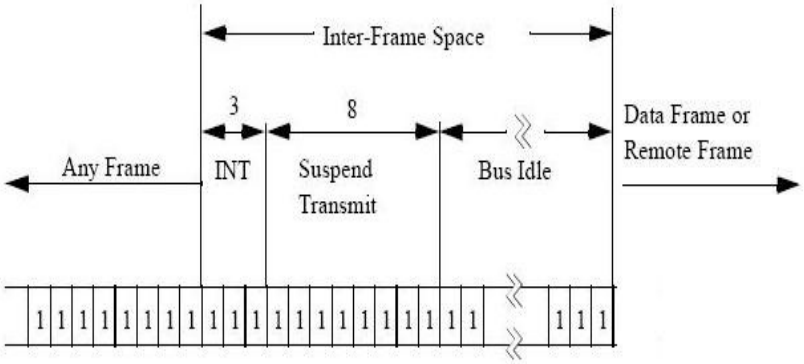
Il campo successivo nominato '*'Cyclic Redundancy Check Code"* (CRC) assolve l'importante compito del controllo per la rilevazione di errori che minano l'integrità del dato.

Infine, vi sono 1 bit sempre alto e 2 bit di *aknowledgement*.

I nodi che hanno ricevuto correttamente il messaggio portano il bit precedente l'*End Of Frame* (*aknowledgement slot*) a zero. Quindi il campo sopra citato, *End Of Frame* (EOF), consistente di 7 bit di tipo recessivo segnala la fine del messaggio appena trasmesso sul bus.

Il bus viene considerato libero dopo un intervallo di almeno 3 bit recessivi. Nei rimanenti 4 bit time le altre unità possono inviare sul bus un *Overload Frame* per segnalare se sono ancora occupate in elaborazioni. In tal caso il nodo in questione invierà un bit basso appena dopo l'EOF e le altre unità risponderanno a loro volta con lo stesso valore sul bus per impedire l'inizio di un nuovo frame.

Nella comunicazione CAN la sequenza dei vari frame è regolata secondo un certo *timing*. Tra due frame (DATA o Remote) deve passare un certo intervallo di tempo definito che permetta ai singoli nodi di processare il messaggio ricevuto. Questo intervallo è detto *Inter Frame Space* (INT) e la sua struttura è riportata in figura.



Dopo i bit di chiusura del frame precedente, un qualunque nodo che volesse impegnare il bus deve aspettare una precisa sequenza di durata minima pari a 11bit: 3 INT e 8 di sospensione della trasmissione. Passato questo intervallo il nodo che volesse comunicare può farlo altrimenti il bus va nello stato di IDLE in cui si ha una sequenza di bit recessivi (1 logico).

3.3. Error Frame: la gestione degli errori

Nel protocollo CAN è stata pensata anche una parte di gestione delle cinque condizioni di errore che possono verificarsi in una trasmissione dovuti alla collocazione dei nodi in ambienti industriali spesso densi di rumore elettromagnetico.

Il primo tipo di errore è denominato errore di CRC: un CRC a 15-bit è calcolato dal nodo che trasmette caricandolo nel campo CRC. Tutti i nodi presenti nella rete ricevono questo messaggio, calcolano il CRC e verificano che il CRC ricevuto corrisponda con quello da loro calcolato. Se questi non corrispondono, si verifica un errore di CRC e un frame di errore (*Error Frame*) è generato dal nodo. Il tipico *Error Frame* è composto da 6 bit dello stesso segno che saranno di tipo dominante per il nodo che ha rilevato l'errore, di tipo recessivo per i nodi che gli stanno rispondendo. Se almeno un nodo non ha correttamente visualizzato il messaggio, il messaggio viene poi nuovamente inviato dopo un intervallo di tempo.

Un altro tipo di errore è definito come errore di *Acknowledge*. Nel campo di *Acknowledge* di un messaggio, il nodo di trasmissione controlla se, nell'intervallo di tempo dedicato al ACK (ACK slot) in cui sta inviando un bit recessivo, è presente un bit dominante. La presenza di un bit dominante permette di conoscere che almeno un nodo del bus ha ricevuto correttamente il messaggio. Se il bit è recessivo vuol dire che nessun nodo ha ricevuto il messaggio correttamente. Si è verificato un errore di *Acknowledge*. Un frame di errore è quindi generato e il messaggio originale verrà ripetuto.

Terza tipologia di errore è quello di forma: si verifica un errore di forma nel caso in cui un nodo rileva un bit "dominante" in uno dei seguenti quattro segmenti del messaggio: *end of frame*, *Inter frame Space*, ACK Del o in CRC Del. Il messaggio originale è nuovamente inviato dopo un intervallo di tempo corretto.

Diversamente un errore di bit si verifica se un nodo trasmettitore invia un bit "dominante" e rileva un bit recessivo, o se si invia un bit recessivo e rileva un bit dominante nella fase in cui compara il livello effettivo del bus e il bit che sta inviando. Se durante la fase di arbitrato o l'intervallo di ACK un bit dominante è rilevato al posto del recessivo che si sta trasmettendo e nessun errore di bit è generato è perché un normale arbitrato o ack sta avvenendo. Se un bit di errore è rilevato, allora un frame di errore è generato e il messaggio originale è nuovamente inviato.

Infine, abbiamo l'errore stuff. Il protocollo CAN utilizza un metodo di trasmissione NRZ. Ciò significa che il livello di bit è posto sul bus per l'intero tempo del bit. La comunicazione CAN è anche asincrona, e la codifica NRZ è utilizzata per consentire al nodo ricevente di sincronizzarsi recuperando il clock dal flusso di dati ricevuto. I nodi riceventi si sincronizzano sulla transizione da bit recessivo a dominante (transizione 0→1). Se nella stringa di dati vi sono più di 5 bit dello stesso tipo, il CAN automaticamente inserisce un bit di polarità opposta nel flusso di dati. Il nodo(i) ricevente

utilizzerà questo bit per la sincronizzazione, ma ignorerà questo bit per la definizione dei dati. Se, tra l'inizio del frame e il delimitatore di CRC, vengono rilevati 6 bit consecutivi con la stessa polarità, la regola di stuff è stata violata e si verifica errore, un frame di errore viene inviato e il messaggio è ripetuto.

Quindi tutti i tipi di errori rilevati sono resi pubblici a tutti gli altri nodi attraverso l'*Error Frame* come spiegato precedentemente nel paragrafo.

L'effetto della rilevazione di un errore è l'immediata eliminazione dell'ultimo messaggio ricevuto; tutte le unità che abbiano recepito il messaggio errato saranno comunque protette.

Ogni nodo può essere in uno dei tre stati di errore: attivo, passivo e Bus-Off. Un nodo entra nello stato di errore quando sia il contatore di errore di trasmissione (TCE) e sia il contatore di errori in ricezione (REC) sono al di sotto di 128.

Lo stato di errore attivo è la modalità operativa normale che consente di trasmettere e ricevere senza restrizioni. Un nodo, nello stato di errore attivo, può quindi prendere parte attiva nella comunicazione di bus, compreso l'invio di un *flag* di errore attivo che consiste in 6 bit dominanti consecutivi. Il flag di errore viola la regola di bit stuff e comportando l'invio da parte di tutti gli altri nodi di un flag di errore, *chiamato Error Echo Flag*.

Un nodo entra nello stato di errore passivo quando sia il contatore d'errore di trasmissione o di ricezione supera il valore di 127. Nodi nello stato di errori passivi non sono autorizzati a trasmettere flag di errori attivi sul bus, ma invece possono trasmettere il flag di errori passivi che consiste in 6 bit recessivi. Se il nodo nello stato di errore passivo è attualmente l'unico trasmettitore sul bus allora il *flag* di errore passivo violerà la regola di stuff e i nodi riceventi risponderanno con il loro *flag* di errore (sia attivo o passivo a seconda del loro stato di errore). Se il nodo in errore passivo in questione non è l'unico trasmettitore (vale a dire durante l'arbitrato) o è un ricevitore, il flag di errore non avrà alcun effetto sul bus a causa della natura recessiva dei suoi bit. Quando un nodo in errore passivo trasmette e rileva un bit ‘’dominante’’, deve vedere il bus come inattivo per un tempo pari a 8 bit dopo un intervallo prima di riconoscere il bus come disponibile. Dopo questo tempo, esso tenterà di ritrasmettere.

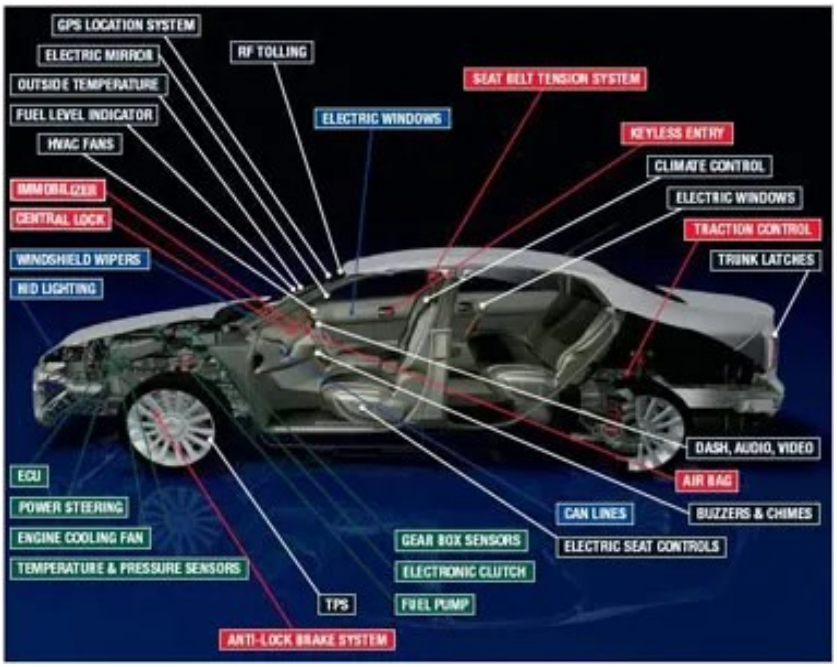
Un nodo va nello stato di *Bus-Off* quando il contatore di errori di trasmissione è superiore a 255 (gli errori di ricezione non possono portare un nodo nello stato di *Bus-Off*). In questo stato, il nodo non può inviare o ricevere messaggi, riconoscere messaggi o trasmettere frame di errore di qualsiasi tipo. Così si è ottenuto il confinamento di fallimento. Vi è una sequenza di recupero del bus definito dal protocollo CAN, che permette ad un nodo, che è in *Bus-Off*, di recuperare e di tornare nello stato di *Errore-Active*, cominciando a trasmettere di nuovo se la condizione di guasto viene rimossa.

4. L’applicazione del CAN all’automotive

4.1. I servizi offerti dal bus CAN

Dalla metà degli anni ’90, i sistemi del motore e della trasmissione si sono basati su una rete di sensori che trasmettono le informazioni al computer del motore, *body computer*, che quindi regola il dosaggio del carburante, i punti di cambio della trasmissione e altri fattori.

I veicoli moderni possono avere 70 o più sottosistemi elettronici, ciascuno con la propria unità di controllo elettronico (ECU). Insieme a quelli ovvi, come la trasmissione o i freni antibloccaggio, cose come alzacristalli elettrici, specchietti, porte, audio / GPS, servosterzo elettrico e persino il cruise control hanno tutti la loro mini-rete.



Man mano che questi sottosistemi diventavano più sofisticati, gli ingegneri hanno visto la necessità che fossero in grado di comunicare tra loro, oltre che con il computer centrale.

I sensori richiedono input da altri sensori, ad esempio un veicolo con controllo della velocità adattivo o avvisi di deviazione dalla corsia necessita di informazioni da una varietà di sensori. La centralina ABS calcola la velocità dell’auto e la comunica al *body computer* che comunicherà questo dato a tutte le centraline che ne necessitano (gestione motore, cruscotto, climatizzazione, sistema satellitare, ecc..). Lo stesso vale per tutti i vari segnali: ad esempio il numero di giri è rilevato dalla centralina gestione motore e comunicato al cruscotto e agli altri nodi del sistema (configurazione VeNICE: *Vehicle Network Integration Component Electronics*). Un veicolo che utilizza tergicristalli automatici e un sensore pioggia può informare il sistema ABS di innestare leggermente i freni,

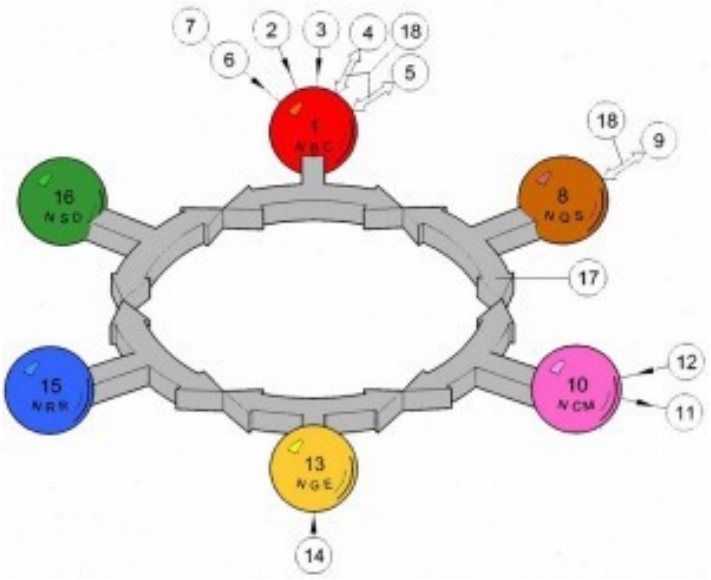
abbastanza a lungo da eliminare l'acqua piovana dai rotori. Il controllo della trazione, l'ABS[2] e i sistemi di stabilità del veicolo devono tutti essere in grado di comunicare tra loro affinché un guidatore possa mantenere il controllo durante una frenata di panico o una sterzata. È qui che entra in gioco la rete CAN bus.

Ogni auto ha il suo livello di ‘‘condivisione dati’’: il sistema di dialogo con centralizzazione delle informazioni è in grado di controllare anche l'emissione della corrente per l'illuminazione e di mantenere stabili dati come assorbimento o tensione.

Di seguito un esempio di collegamento di una rete che implementa la configurazione VeNICE:

1. (NBC) Nodo Body Computer;
2. Plancetta comandi;
3. Devio guida;
4. Modulo sirena allarme;
5. Modulo sensori;
6. Centralina ABS;
7. Sensori ABS;
8. (NQS) Nodo Quadro Strumenti;
9. Centralina cambio automatico;
10. (NCM) Nodo Controllo Motore
11. Attuatori NCM;
12. Sensori NCM;
13. (NGE) Nodo Guida Elettrica;
14. Sensore di coppia NGE;
15. (NRR) Nodo Radio o *Radionavigatore*;
16. (NSD) Nodo Strumento di Diagnosi;
17. Rete CAN;
18. Linea seriale.

[2] Dal tedesco Antiblockiersystem, sistema antibloccaggio.

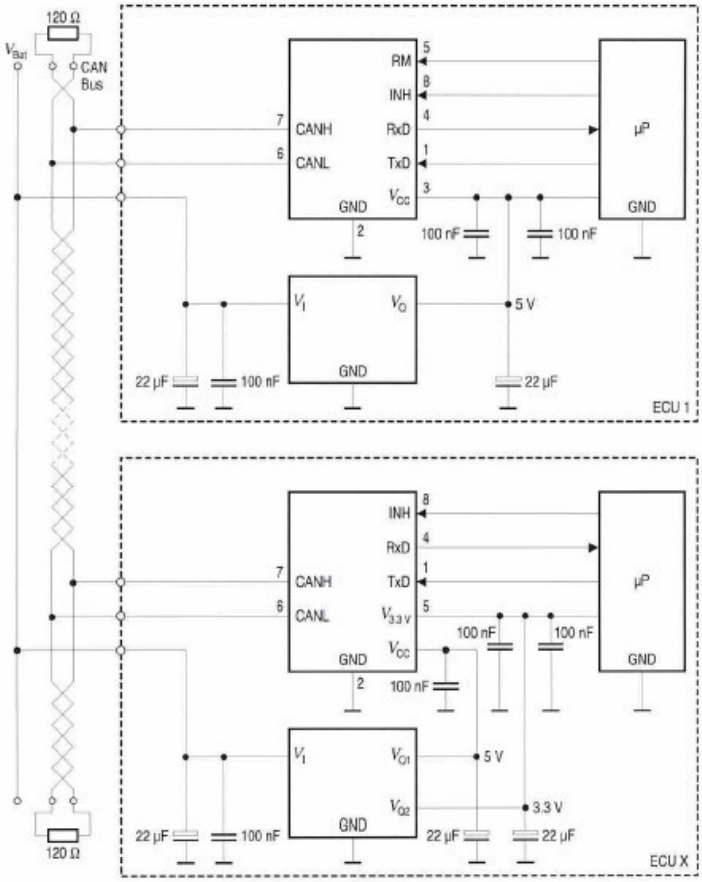


Se c'è un problema con i sensori o i processori che comunicano tra loro, questo verrà registrato come codice di guasto specifico del CAN-bus nel computer di trasmissione principale e illuminerà la spia "motore di controllo". Può anche causare altri sintomi come problemi di guidabilità, un quadro strumenti guasto o persino una condizione di mancato avviamento.

I codici di errore di comunicazione generalmente includono una "U" nella loro nomenclatura. Per un codice di errore che indica una mancata accensione del cilindro o una cattiva lettura da un sensore di posizione MAP, MAF o manovella, un meccanico esperto può usare i suoi poteri di deduzione per risolvere il problema che causerebbe quella cattiva lettura e correggerlo, magari usando un voltmetro per vedere quali valori provengono da quel sensore o dai relativi sensori e vedere se rientrano nei parametri normali. Tuttavia, questo è molto più difficile con una configurazione del bus CAN. Ci sono dozzine di questi moduli e miglia di cablaggio che li collegano, e uno scanner OBD-II convenzionale potrebbe fornire codici di errore di base. Ma non è di alcuna utilità quando si tratta delle complessità del sistema bus CAN. Non può, ad esempio, comunicare con i moduli per il sistema A / C, il controllo della stabilità del veicolo, l'uscita dalla corsia o il controllo della velocità adattivo. I tecnici possono diagnosticare problemi con il software e i sistemi automobilistici utilizzando laptop e scanner, avanzati e costosi, specifici per CAN.

Data la complessità ad operare su questi sistemi, dovuta alla gran mole di conoscenze e precisione strumentale che un tecnico deve possedere per intervenire in caso di malfunzionamento, sono nati gruppi di esperti per questo genere di assistenza. Tra questi F.lli Galletto S.r.l. che opera nel settore Automotive del nostro territorio nazionale partendo dalle riparazioni delle autovetture fino allo sviluppo e alla realizzazione di software e hardware di comunicazione e diagnosi per le varie unità di controllo dei veicoli. Service officiale Bosch e Volkswagen, nel reparto di ricerca e sviluppo

si studiano ogni giorno soluzioni per ovviare tutti i problemi provenienti dalle case madri dei veicoli. Si lavora su schede elettroniche ed ECU di ogni tipo di mezzo: automobili, trattori, barche, moto. Tali schede ricevono una miriade di interventi da quando sono immesse nel mercato, per le più svariate esigenze degli utilizzatori: dalle "mappature della memoria" per via degli aggiornamenti di monitoraggio continui che devono essere eseguiti per esempio ai camion o alle auto, o per aumentare le prestazioni del mezzo come richiedono tutte le squadre di racing (rally, Go-kart, motocross etc.), alla vera e propria manutenzione per malfunzionamento. Si generano quindi una serie di problemi di non banale soluzione.



Esempio di collegamento tra centraline in un sistema di gestione dei moduli

Esistono due metodi principali per connettersi alla ECU tramite la porta OBD (*On-Board Diagnostics*), al banco (*bench mode*) o in modalità di avvio (*boot mode*).

Il modo in cui la ECU può essere "letta" dipende da una serie di fattori, tra cui lo strumento che si sta utilizzando, il modello hardware e persino la versione del software che si sta cercando di leggere. Un altro fattore è il livello di informazioni che si desidera estrarre dalla memoria, a quali aree della mappa (area di memoria con una classe di dati salienti del sistema) si desidera accedere o se si necessiti di un backup completo del file.

Si parla di un vero e proprio sistema *Client-Server* che si cerca di instaurare tra il proprio hardware (*Client*), in cui è installato il software *Master* della trasmissione, e la centralina interrogata (*Server*) col fine di realizzare le azioni di lettura e scrittura della memoria principale della centralina di tipo *FLASH* (*EEPROM* [3]) dove risiede il *firmware*, vera e propria anima software della stessa.

- *Bench Mode Read/Write*

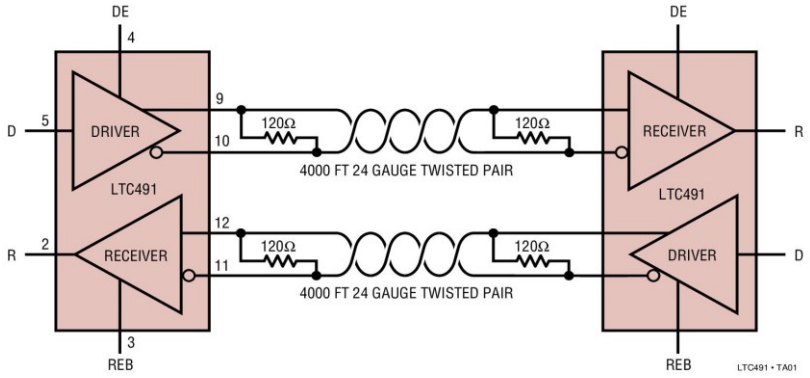
‘’Lavorare al banco’’ è il termine usato per prendere una ECU dall'auto e collegarla direttamente ai pin a cui normalmente si collegherebbe il telaio dell'auto. Per questo sono necessari una fonte di alimentazione esterna e un cavo di avvio. Si collega l'alimentazione a più pin, insieme a una messa a terra e ad altre forme di comunicazione, come CAN+ e CAN- o KLine. Questo è anche noto come funzionamento in modalità di servizio (*service mode*) poiché è il metodo utilizzato da produttori come Bosch e in generale è possibile leggere l'intera memoria di archiviazione (memoria *FLASH*).

- *Boot Mode Read/Write*

La lettura e scrittura del boot è ancora un lavoro svolto al banco ed è forse l'aspetto più tecnico in quanto richiede l'apertura della centralina e il collegamento dei pin direttamente al truciolare. I principi di base sono gli stessi della modalità di servizio: ci si collega direttamente ai pin di connessione, tuttavia, per bypassare il chip di sicurezza si collegano i pin di boot o di terra direttamente alla scheda. È quindi possibile leggere e scrivere l'intera memoria, aggirando la protezione fisica presente nella quasi totalità delle schede prodotte. La modalità di avvio è spesso la prima opzione disponibile sul mercato quando viene sviluppata una nuova soluzione hardware poiché dal punto di vista degli sviluppatori è il primo passo per programmare/riprogrammare tutti i protocolli e i blocchi contenuti nella ECU.

Saldando quindi i cavi ai pin desiderati e prendendo accortezze di livello elettrico (modifica del sistema di trasmissione e/o di alimentazione tramite resistori e capacitori), si cerca quindi di prelevare un segnale elettricamente pulito con l’oscilloscopio. Calcoli supplementari poi serviranno per ricavare da tale segnale il vero e proprio dato di comunicazione e risalire quindi alle informazioni utili nella memoria per riportare il dispositivo al suo completo e normale stato di attività.
L’impostazione di tutto il sistema fisico e logico, del software di comando dell’intero processo di comunicazione (*driver*), richiede un attento studio di tutte le caratteristiche dei dispositivi hardware impiegati dalle case di costruzione delle schede per poter risalire ai dati importanti (*baud* [4] *o bit rate*, *bit di stop* ed eventuali *parity bit*) per rilevare poi i *frame* di comunicazione corretti.

[3] *Electrically Erasable Programmable Read-Only Memory*: memoria di sola lettura il cui contenuto può essere cancellato elettricamente mediante la trasmissione di un voltaggio specifico al registro fisico selezionato.
[4] Il termine baud (pronunciato /ˈbɔːd/, da Émile Baudot, inventore del codice Baudot utilizzato per le telescriventi), in telecomunicazioni e informatica, rappresenta il numero di simboli (dati) che viene trasmesso in un secondo.



Tutti i microcontrollori e molti DSP, incorporano l'hardware necessario alla gestione di una trasmissione seriale, in modalità asincrona UART e/o sincrona USRT, SPI (vedi paragrafo 1.1) e ormai anche per interfacciare il processore con un bus CAN, integrando così le funzioni di comunicazione e controllo di processo.

I controllori CAN integrati sono di solito di tipo "*basic*" (vedi paragrafo 2.1: configurazione *Philips Semiconductor*) e presentano alcune limitazioni rispetto alle funzionalità dei controllori dedicati su singolo chip, in particolare nella gestione dei filtri sui messaggi e nelle dimensioni dei *buffer* [5] di ricezione e trasmissione. Ciononostante, il mercato offre un sacco di soluzioni hardware e le aziende impegnate nello sviluppo della tecnologia CAN, oltre alla più volte citata Bosch, son molte.

Tutti i giorni i programmatori hanno a che fare con processori NXP, Infineon (ex reparto della *Siemens*), ARM, STM, PIC, Renesas, Fujitsu, etc.. oltre ai chip Intel e Texas Instruments. Da anni anche la Motorola impegna i suoi microcontrollori e DSP a supporto di tali sistemi (i più usati e famosi sono della famiglia MC6800, e quelli dedicati al CAN, MPC555/MPC556). Le centraline ormai sono prodotte dalla stragrande maggioranza delle grandi corporazioni mondiali, si ricordano maggiormente le Bosch (delle serie ME e EDC), le Hitachi, Marelli e Delphi Delco.

Per quanto riguarda le protezioni dei dati, esistono molte ed intricate soluzioni applicate tramite algoritmi matematici di complessità rilevante per cifrare sia i messaggi in comunicazione che controllare ad ogni accensione dell'ECU se il contenuto della memoria è stato alterato da qualche intervento esterno al fine di garantire l'integrità del sistema e la sicurezza dei guidatori all'interno dei veicoli.

La principale società europea (tedesca) ETAS, fornisce software di base per veicoli, middleware e strumenti di sviluppo per la realizzazione di veicoli *software-defined*. Le soluzioni

[5] *Buffer*: è una zona di memoria usata per compensare differenze di velocità nel trasferimento o nella trasmissione di dati, oppure per velocizzare l'esecuzione di alcune operazioni come, ad esempio, le operazioni sulle stringhe di caratteri.

olistiche[6] di sicurezza informatica nel settore automobilistico sono offerte tramite il marchio ESCRYPT (proprietà di Bosch Gmbh) che rilascia e certifica secondo gli standard ISO, gli algoritmi e le chiavi crittografiche delle password associate. ETAS è rappresentato in dodici paesi in Europa, Nord e Sud America e Asia.

Gli algoritmi utilizzati per il mascheramento delle comunicazioni sono molteplici, di seguito se ne elencano i quattro usati maggiormente:

- RSA- Acronimo che deriva dai tre nomi dei ricercatori che l'hanno sviluppato (R. Rivest, A. Shamir, L. Adleman). È un algoritmo di cifratura asimmetrica che si basa sull'uso di due chiavi distinte, chiave pubblica e chiave privata, per la codifica e la decodifica. Il funzionamento del metodo RSA si può schematizzare con i seguenti punti:

  1. si scelgono due numeri primi, *p e q*;
  2. si calcola il loro prodotto $N = p \times q$, chiamato *modulo* (dato che tutta l'aritmetica seguente è *modulo N*);
  3. si sceglie poi un numero *e (*chiamato *esponente pubblico*), più piccolo di *N* e primo rispetto a f (*N)* = $(p - 1) \times (q - 1)$, dove f è la funzione di Eulero;
  4. si calcola il numero *d* (chiamato *esponente privato*) tale che $e \times d \rightarrow 1 \bmod ((p - 1) \times (q - 1))$.

  La chiave pubblica è rappresentata dalla coppia di numeri (*N, e*), mentre la chiave privata è rappresentata da (*N, d)*.

- MD5- Questo algoritmo, ideato da R. Rivest, rappresenta lo stadio finale della naturale evoluzione della famiglia di algoritmi denominati *Message Digest [7] algorithm* (MD). L'algoritmo di *hashing* MD5 risulta idoneo per processare messaggi di lunghezza variabile producendo un *digest* di 128 bit. I messaggi in ingresso vengono processati a blocchi di 512 bit. Se il messaggio in ingresso non risulta scomponibile in un numero intero di blocchi, viene aggiunto un opportuno numero di bit non significativi (pad data) in modo da rispettare questa proprietà. A questo punto l'ingresso viene spezzato in blocchi che vengono processati in cascata, uno dopo l'altro, applicando ricorsivamente le funzioni interne all'algoritmo MD5 (alla stessa famiglia appartengono algoritmi come MD2, MD4 e RIPEMD160).

[6] Si parla in questi casi di evolvere verso un modello olistico, cioè sviluppare in modo collaborativo un approccio al contrasto delle minacce, includendo il quadro organizzativo e favorendo integrazione e cooperazione ad ogni livello, digitale, logico, fisico.

[7] La *Message Digest function* è una funzione che mappa un messaggio arbitrariamente lungo in una stringa di lunghezza prefissata, cercando di far in modo che da questa stringa non si possa risalire al messaggio che l'ha generata (funzione unidirezionale).

- CHECKSUM- Letteralmente ‘’somma di controllo’’, l’ultimo byte di ogni pacchetto trasmesso contiene un valore codificato in funzione dei byte precedenti del messaggio. Il destinatario, perciò, potrà ricalcolarsi il *checksum* (ovvero quest’ultimo valore) e se il calcolo sarà diverso allora vorrà dire che c’è un errore nei dati trasmessi.
- CRC- *Cyclic redundancy check* è uno dei metodi più diffusi per l’implementazione del *checksum.* Il nome deriva dal fatto che i dati di uscita sono prodotti da un’elaborazione di quelli in ingresso facendoli scorrere ciclicamente in una rete logica che realizza una specifica funzione.

Molti circuiti integrati possiedono moduli fisici costruiti *ad hoc* con processore a sé stante solo per il calcolo degli algoritmi sopra elencati in modo che durante l’elaborazione la CPU principale continui indisturbata la sua esecuzione.

4.2. Analisi delle routine di servizio delle ECU

La programmazione delle periferiche di tipo USART è relativamente semplice. La creazione di *driver* che permettano di inviare messaggi sulla linea seriale è dunque rapida, anche nel linguaggio *assembly* del processore. La programmazione invece delle periferiche di controllo per il bus CAN è decisamente più complessa, richiedendo sia la definizione dei parametri della trasmissione che del protocollo di collegamento. Per facilitarne il compito i costruttori del dispositivo forniscono librerie specifiche tipicamente progettate per compilatori C o C++. Enti di terze parti, come Microsoft, offrono ambienti di sviluppo, ad esempio Visual Studio, studiati appositamente per riunire tutte le funzioni definite in queste librerie.

Nell’esperienza di tirocinio curriculare, svolta dal soggetto scrivente il presente elaborato, grazie a programmi di filtraggio e analisi dei canali di comunicazione, programmi *sniffer* (Kvaser CAN KING, vedi paragrafo 2.1), si sono potuti estrarre alcuni esempi di comunicazione.

Nella rete *Client-Server*, citata precedentemente, si può vedere come un PC (*client*) interroghi una ECU (*server*) al fine di testare o, in gergo, ‘’*debuggare*’’ il funzionamento della centralina stessa.

Gli estratti delle analisi che seguono, sono da leggere basandosi sulla tabella di *fig.4.2.1*, che associa i comandi (numeri in esadecimale inviati sotto forma di byte sul CAN bus dal nostro dispositivo laptop) descritti nella documentazione ISO 14229 del 2006.

Service	defaultSession	non-defaultSession
DiagnosticSessionControl - 10 hex	x	x
ECUReset - 11 hex	x	x
SecurityAccess - 27 hex	N/A	x
CommunicationControl - 28 hex	N/A	x
TesterPresent - 3E hex	x	x
AccessTimingParameter - 83 hex	N/A	x
SecuredDataTransmission - 84 hex	N/A	
ControlDTCSetting - 85 hex	N/A	x
ResponseOnEvent - 86 hex	x [a]	x
LinkControl - 87 hex	N/A	x
ReadDataByIdentifier - 22 hex	x [b]	x
ReadMemoryByAddress - 23 hex	x [c]	x
ReadScalingDataByIdentifier - 24 hex	x [b]	x
ReadDataByPeriodicIdentifier - 2A hex	N/A	x
DynamicallyDefineDataIdentifier - 2C hex	x [d]	x
WriteDataByIdentifier - 2E hex	x [b]	x
WriteMemoryByAddress - 3D hex	x [c]	x
ClearDiagnosticInformation - 14 hex	x	x
ReadDTCInformation - 19 hex	x	x
InputOutputControlByIdentifier - 2F hex	N/A	x
RoutineControl - 31 hex	x [e]	x
RequestDownload - 34 hex	N/A	x
RequestUpload - 35 hex	N/A	x
TransferData - 36 hex	N/A	x
RequestTransferExit - 37 hex	N/A	x

[a] It is implementation-specific whether the ResponseOnEvent service is also allowed during the defaultSession.

[b] Secured dataIdentifiers require a SecurityAccess service and therefore a non-default diagnostic session.

[c] Secured memory areas require a SecurityAccess service and therefore a non-default diagnostic session.

[d] A dataIdentifier can be defined dynamically in the default and non-default diagnostic session.

[e] Secured routines require a SecurityAccess service and therefore a non-default diagnostic session. A routine that needs to be stopped actively by the client also requires a non-default session.

Figura 4.2 1 ISO 14229, 2006

La tabella mostra i servizi consentiti durante la sessione di comunicazione predefinita (*default session*) e la sessione non predefinita (*non-default Session*). Qualsiasi sessione non predefinita è legata ad un *timer* di sessione diagnostica che deve essere mantenuto attivo dal *Client*. L'acronimo N/A sta per *Not/Ammitted*, sottolineando che la routine in questione ammette solo la *non-default session.*

I comandi elencati si riferiscono a richieste che il nostro dispositivo *Client* può inviare alla ECU; quest'ultima in caso di risposta positiva (assolvimento della domanda e del servizio richiesto) risponderà inviando come primo *byte data* della comunicazione di ritorno, un numero esadecimale corrispondente a quello del comando ricevuto ed aumentato di 0x40 (in decimale, 64). Nel caso di

risposta negativa, invece, si avrà di ritorno il valore 0x7F seguito dal numero del comando a cui la centralina non riesce ad eseguire la routine associata (caso di errore) per un qualsiasi motivo.

A titolo di esempio si può prendere il primo comando della lista, *DiagnosticSessionControl request* (0x10):

- Possibile struttura di un messaggio di richiesta (*Request*) della routine *DiagnosticSessionControl request.*

Message direction:	client → server		
Message type:	Request		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value (hex)**	**Mnemonic**
#1	DiagnosticSessionControl request SID	10	DSC
#2	diagnosticSessionType = programmingSession, suppressPosRspMsgIndicationBit = FALSE	02	DS_ECUPRGS

- Possibile struttura di un messaggio di risposta POSITIVA (*Response*) della routine *DiagnosticSessionControl request.*

Message direction:	server → client		
Message type:	Response		
A_Data byte	**Description (all values are in hexadecimal)**	**Byte value (hex)**	**Mnemonic**
#1	DiagnosticSessionControl response SID	50	DSCPR
#2	diagnosticSessionType = programmingSession	02	DS_ECUPRGS

Le possibilità di strutturare il messaggio sono innumerevoli, e dipendono soprattutto dal fatto che dopo il corpo principale del frame (Identificatore e comando associato) si possano aggiungere altri valori pre-programmati dallo sviluppatore del modulo, che richiamino *sub-routine* dedicate e specifiche dei costruttori dell'ECU progettata.

Di seguito gli estratti derivanti dal lavoro di ricerca svolto.

```
1
2 Attesa :000090 ms  07 B0 11 02 10 81 00 00 00 00
3 Attesa :000003 ms  07 C1 F1 02 50 81 00 00 00 00
4
```

Figura 4.2 2

In Figura 4.2.2 si può vedere lo scambio di messaggi tra un PC e un ECU di una *Ferrari 458*.

Il programma *sniffer* ci presenta il ritardo tra un messaggio e l'altro in *ms* (millisecondi), dopo l'etichetta "attesa". In questo caso si ha a che fare con l'ID (*Identificator*) di tipo *extended* (vedi

paragrafo 2.2) a 28 bit, presentato ugualmente in 4 byte (32 bit), sarà il *driver* a filtrare i bit in eccesso per capire il vero valore rappresentato.

ID del PC nella rete bus CAN: 07 B0 11.

ID dell'ECU nella rete bus CAN: 07 C1 F1.

Solitamente dopo i primi byte relativi agli identificatori del messaggio vi è un byte che dichiara quanti byte di dati utili vi sono da leggere nel frame (*Data Length Counter*), gli altri sono aggiunti per raggiungere il massimo di dati possibili (8 byte) rendendo il messaggio valido per la trasmissione. Lo 0x81 è un valore associato specificatamente dai programmatori, come chiaro esempio di *sub-routine* (la macchina in questione svolgerà dei compiti precedentemente programmati in fase di sviluppo dalla casa madre).

```
240  Attesa :000096 ms  06 F1 12 02 3E 01 00 00 00 00
241  Attesa :000033 ms  06 12 F1 01 7E FF FF FF FF FF
242  Attesa :000095 ms  06 F1 12 02 3E 01 00 00 00 00
243  Attesa :000033 ms  06 12 F1 01 7E FF FF FF FF FF
244  Attesa :000105 ms  06 F1 12 02 3E 01 00 00 00 00
245  Attesa :000034 ms  06 12 F1 01 7E FF FF FF FF FF
246  Attesa :000095 ms  06 F1 12 02 3E 01 00 00 00 00
247  Attesa :000033 ms  06 12 F1 01 7E FF FF FF FF FF
248  Attesa :000095 ms  06 F1 12 02 3E 01 00 00 00 00
249  Attesa :000033 ms  06 12 F1 01 7E FF FF FF FF FF
```

Figura 4.2 3 Conversazione PC con ECU Ferrari 458

In questo caso si ha sempre la stessa comunicazione ma gli ID sono cambiati (pur mantenendo lo stesso formato *extended*) perché il PC del tecnico sta cercando di comunicare con il modulo di gestione del cambio automatico ma tale unità non risponde per problemi. Questo è l'esempio più comune di un caso di errore in cui la centralina risponde al comando di *TesterPresent 0x3E* (entrare in modalità di tester del sistema con il conseguente sblocco di alcune protezioni) con la condizione di errore (0x3E + 0x40 = 0x7E). I byte postumi sono sempre messi come riempimento del frame del messaggio.

Nel prossimo esempio (*Figura 4.2.4*) si possono vedere i passaggi principali di una routine di recupero (*Recovery Service*) di una memoria interna *FLASH* di una scheda che non risponde più per qualche malfunzionamento. Per entrare di nuovo in comunicazione si è dovuto utilizzare il metodo del *Boot Mode* (vedi paragrafo 4.1). Gli ID sono standard con valore a 11 bit rappresentato sempre su 2 byte (07 E0 PC; 07 E8 ECU).

```
1
2  013 ms   07 E0 02 10 85 00 00 00 00 00 startDiagnosticSession
3  005 ms   07 E8 02 50 85 00 00 00 00 00
4
5  196 ms   07 E0 02 27 01 00 00 00 00 00 securityAccess(request SEED)
6  005 ms   07 E8 05 67 01 70 EB 1F 00 00 SEED : 70 EB 1F 00
7
8  010 ms   07 E0 05 27 02 F3 9B 16 00 00 securityAccess(send KEY) KEY : F3 9B 16 00
9  004 ms   07 E8 02 67 02 00 00 00 00 00
10
11 004 ms   07 E0 04 B1 00 B2 AA 00 00 00
12 004 ms   07 E8 03 7F B1 78 00 00 00 00 WAIT
13 777 ms   07 E8 03 F1 00 B2 00 00 00 00 OK
14
15 035 ms   07 E0 10 09 34 00 01 00 00 00 requestDownload
16 011 ms   07 E8 30 00 00 00 00 00 00 00
17 005 ms   07 E0 21 00 04 00 00 00 00 00
18 006 ms   07 E8 03 7F 34 78 00 00 00 00 WAIT
19 007 ms   07 E8 03 74 04 01 00 00 00 00
```

Figura 4.2 4 *Diagnostic Session service* in modalità *boot mode* su una centralina Bosch della famiglia EDC (vedi paragrafo 4.1).

Dopo il comando 0x10, si dichiara al mezzo automobilistico che si intende iniziare una routine di diagnostica degli errori in modalità d'avvio con la *sub-routine* 0x85. Segue la richiesta del seme (*SEED*) per generare la chiave crittografica con il comando 0x27 e *sub-routine* 0x1 (*requestSEED*).

L'ECU presa in esame risponde inviando il seme richiesto (70 EB 1F 00), il quale verrà elaborato dal PC secondo uno degli algoritmi riportati nel *paragrafo 4.1* e ne restituirà il risultato (chiave o *KEY*) con la *sub-routine* 0x2 del comando 0x27 (*securityAccess, sendKEY:* F3 9B 16 00).

Con lo stesso dato d'ingresso (inizio), se l'algoritmo presente in ambo le unità è il medesimo, allora si avrà lo stesso risultato. La centralina confronterà questo risultato con quello ricevuto ed in caso i due combacino perfettamente (abbiano valori identici), la prova sarà superata ed il Client avrà completo accesso alle funzionalità dell'ECU.

Si può notare come l'esame sia passato dopo un comando di *WAIT* (attesa elaborazione dati in corso) da parte della centralina e si possa procedere con la richiesta di scrittura (*requestDOWNLOAD*) associata al valore esadecimale 0x34.

Una volta riscritti i dati dell'intera memoria *FLASH* si interroga la centralina con le parti sanate tramite il comando 0x23 (*readmemorybyAddress*), leggendo i dati contenuti nelle aree di memoria ricodificate ed indicizzate dai byte seguenti il byte di comando (vedi *figura 4.2.5*).

```
 7  Attesa :000004 ms  07 E0 06 23 18 00 00 00 F0 00 readMemoryByAddress(ADDRESS 18 00 00; MEMORY SIZE 00; transmission mode F0 ;
 8  Attesa :000004 ms  07 E8 10 F4 63 18 00 00 44 43
 9  Attesa :000002 ms  07 E0 30 22 00 00 00 00 00 00
10  Attesa :000001 ms  07 E8 21 4D 33 37 2D 42 31 45
11  Attesa :000001 ms  07 E8 22 2D 55 47 45 38 32 47
12  Attesa :000001 ms  07 E8 23 2D 5A 32 32 2D 43 31
13  Attesa :000001 ms  07 E8 24 34 30 2D 41 34 2D 31
14  Attesa :000011 ms  07 E8 25 33 32 2D 4E 2D 44 45
15  Attesa :000014 ms  07 E8 26 2D 50 2D 41 2D 31 31
16  Attesa :000014 ms  07 E8 27 30 39 30 35 00 11 32
```

Figura 4.2 5 Letture delle diverse zone di memoria di un microcontrollore che gestisce il traffico sul bus CAN.

I byte che seguono 0x23 indicano la zona di memoria da cui iniziare la lettura (indirizzo fisico espresso in esadecimale, *ADDRESS nell' immagine*) e addirittura, in questo caso, quanti byte si vogliono leggere 0xF0 (240 in notazione decimale). I dati esposti in questo caso sono da convertire nel formato *ASCII*[8] e si riferiscono al modello di telaio e centralina che monta l'autovettura.

Infine, la quasi totalità delle comunicazioni, si conclude con il comando di chiusura trasferimento dati (*RequestTransferExit*) 0x37 e lo spegnimento della comunicazione con il 0x11, l'ECU inizierà una sequenza di operazioni macchina a livello di CPU per salvare i buffer di dati ricevuti e/o scritti nella memoria e tornare al normale funzionamento una volta reinstallata nel veicolo (*figura 4.2.6*).

```
Attesa :000050 ms  07 E0 02 11 01 00 00 00 00 00
```

Figura 4.2 6 Richiesta di servizio 0x11: uscita da una sessione di comunicazione.

[8] ASCII (acronimo di *American Standard Code for Information Interchange*, letteralmente "codice standard americano per lo scambio di informazioni") è un codice per la codifica di caratteri.

Conclusioni

Il protocollo CAN è stato ottimizzato per i sistemi che devono trasmettere e ricevere relativamente piccole quantità di informazioni (rispetto a Ethernet o USB, che hanno lo scopo di spostare blocchi di dati molto più grandi) in maniera affidabile a uno o tutti i nodi della rete. La funzionalità CSMA/CD permette ogni nodo di avere pari opportunità di accesso al bus, e permette un buon trattamento delle collisioni.

Dal momento che è un protocollo basato sui messaggi e non sull'indirizzamento, tutti i nodi connessi al bus ricevono tutti i messaggi e riconoscono ogni messaggio, indipendentemente dal fatto che siano dati importanti o meno. Questo permette al bus di operare nella modalità nodo-a-nodo con formati di messaggio multicast senza dover inviare diversi tipi di messaggi. La trasmissione dei messaggi, veloce e robusta con il confinamento del fallimento è anche un grande valore aggiunto per il CAN perché i nodi difettosi sono automaticamente esclusi dal bus non permettendo a qualsiasi nodo di interrompere la rete. Questo garantisce effettivamente che la larghezza di banda sarà sempre disponibile per i messaggi critici che circolano in rete.

Con tutti questi vantaggi realizzati nel protocollo CAN e il suo slancio nel mondo automobilistico, altri mercati cominceranno a vedere ed implementar il CAN nei loro sistemi.

Una nuova sfida attende il protocollo CAN nei nuovi sistemi di guida autonoma dove sarà di vitale importanza la massima connettività dei nodi della rete per una comunicazione *real-time* dei dati e il monitoraggio delle prestazioni del mezzo a distanza, garantendo sempre alta qualità di trasmissione per poter integrarsi al meglio nelle future reti di *Internet of Things* (IOT) con protocolli come *ethernet* e *USB*.

BIBLIOGRAFIA

- Buso Simone, *Introduzione alle applicazioni industriali di microcontrollori e DSP*, Esculapio 2018.
- Robert Bosch Gmbh, *Automotive Handbook*, 11^ edizione Wiley 2022.
- *ISO 14229*, 2006.
- Singh Simon, *Codici & Segreti*, Bur 2015.

SITOGRAFIA

- www.datajob.com
- www.eipro.elettronicain.it
- www.carpedia.it
- www.veicolux.it
- www.computersec.it
- www.modofluido.hydac.it
- www.sti-innsbruck.at
- www.microst.it
- www.sis.se
- www.emcu.it
- www.electronics-tutorials.ws