
**UNIVERSITA' POLITECNICA DELLE MARCHE**

FACOLTA' DI Ingegneria.

---

Corso di Laurea triennale/magistrale Ingegneria Elettronica.

# Sniffing e successivo debbugging dati provenienti dal BUS di centraline elettroniche, convertiti secondo protocollo di comunicazione CAN-DBC tramite emulatore CL2000.

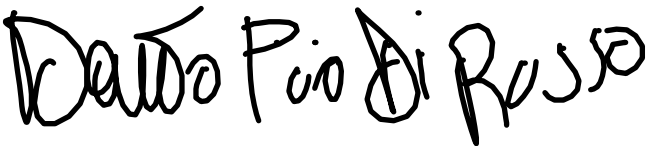
# Sniffing and debbugging of data from ECU (Elettronic Control Units)'s BUS, converted according to CAN-DBC communication protocol via CL2000 device.

Relatore: Chiar.mo/a

Prof. Morini Antonio

Tesi di Laurea di:

Di Russo Donato Pio



A.A. 2021/2022

# INDICE:

# Introduzione:

- **Esperienza aziendale :**

L'elaborato sviluppato in queste pagine tratterà in maniera dettagliata la mia esperienza da tirocinante accademico presso l'azienda molisana **Sensor ID srl di Campochiaro (CB)**. Tale relazione verrà sfruttata come tesi di tipo progettuale per la conclusione del percorso di studi triennale presso l'Università Politecnica delle Marche per il corso di ingegneria elettronica.

Tengo vivamente a ringraziare la suddetta azienda che, rendendomi pienamente partecipe del progetto, mi ha interfacciato per la prima volta con il mondo del lavoro scaturendo in me la necessità di rendere la mia performance il più efficace possibile al fine di concludere progetti. Inoltre mi ha messo nelle migliori condizioni possibili fornendomi tutti gli strumenti di cui avessi bisogno e rendendosi disponibile ad affrontare con me qualsiasi problematica si palesasse.

- **Progetto svolto :**

Mi è stato assegnato un progetto di tipo prettamente grafico. L'azienda Tazzari GL di Imola (BO) richiedeva che venissero **sniffati i dati presenti nel BUS di un simulatore di centralina elettronica** di un motore elettrico e successivamente graficati **quelli di maggiore rilievo quali ad esempio:**

**- State Of Charge della batteria;**

**- Voltaggio minimo;**

**- Voltaggio massimo;**

**- Corrente massima rilevata dal sensore;**

**- Corrente richiesta dal motore;**

**- Corrente effettivamente fornita;**

# Capitolo 1:

- **Storia e funzionamento della centralina elettronica (ECU) :**

Le prime "centraline", con l'avvento dei motori alimentati a petrolio raffinato, erano vere e proprie innovazioni di tipo meccanico. Infatti, l'utilizzo del carburante era determinato dalla taratura del **carburatore** che banalmente aumentava e diminuiva il flusso di carburante che entrava nel motore. Questo metodo però fu da subito definito poco preciso e per niente efficiente sia per i consumi spropositati di carburante sia per la manutenzione meccanica necessaria per il funzionamento. Negli anni '70 furono introdotte una coppia di **elettrovalvole** sul carburatore in grado di renderne il funzionamento più rapido e dinamico alle variazioni del livello dell'acceleratore.

**Figura 1. Pompa di iniezione con elettrovalvola.**

Diverso fu il compito affidato alla centralina all'inizio degli anni '80; difatti con l'introduzione dei sistemi di iniezione, all'ECU ( **unità di controllo motore** ovvero la centralina ) spettava il **controllo dell'accensione dei motori a benzina.** Modificando e ottimizzando l'anticipo, vale a dire il momento in cui la candela a incandescenza "scocca" la scintilla durante la fase di combustione, si notò un notevole miglioramento delle prestazioni, una buona riduzione dei guasti meccanici e un relativo risparmio in termini di consumo. Sempre negli stessi anni fu fondamentale l'introduzione delle **sonde lambda, sensore O2, HEGO** in grado di segnalare alla centralina i valori di ossigeno nei gas di scarico garantendo così il controllo sull'iniezione. Il principio di funzionamento è semplice: se il **rapporto lambda** fosse diverso da 1 ( che corrisponde al rapporto "perfetto", quindi per **ogni kg di carburante devono essere aspirati 14,7 kg di aria** ) verrebbero rilevate quantità di ossigeno non sufficienti ( miscela ricca di carburante ) o superiori a quelle necessarie ( miscela magra di carburante ).

All'inizio degli anni '90 iniziarono a farsi strada anche le centraline elettroniche per i motori diesel. La differenza, nonostante il funzionamento sia praticamente lo stesso, è : **la modalità di accensione del combustibile.** Nel motore a benzina, dato l'alto grado di infiammabilità, l'impianto elettronico genera l'accensione tramite una scintilla, mentre nel motore a gasolio ( carburante molto meno infiammabile ) l'accensione è garantita dall'aumento elevato della pressione all'interno della camera di scoppio.

Con l'avvento del nuovo millennio, la ricerca di una maggior sicurezza dinamica e passiva dell'autovettura hanno spinto lo sviluppo di sistemi elettronici destinati a svolgere questi compiti, a loro volta pilotati dal "cervello del motore": la centralina elettronica. Ora la domanda sorge spontanea: come funzionano le centraline moderne? La risposta è presente sulla fonte dalla quale ho raggruppato la maggior parte delle informazione sul funzionamento e la storia dell'ECU ovvero il seguente sito: **ECU testing, precisamente nella sezione "panoramica sulle centraline elettroniche".** Per il funzionamento dell'unità di controllo c'è bisogno di 4 diversi elementi fondamentali:

- **Sensori che leggono gli ingressi:** includono i segnali dei sensori di temperatura e di pressione i segnali di on/off il sensore di temperatura del liquido di raffreddamento, oppure il sensore della posizione del pedale dell'acceleratore. Vengono prese in considerazione anche le richieste dei moduli antibloccaggio dei freni ( ABS ) per il monitoraggio degli sforzi di trazione.

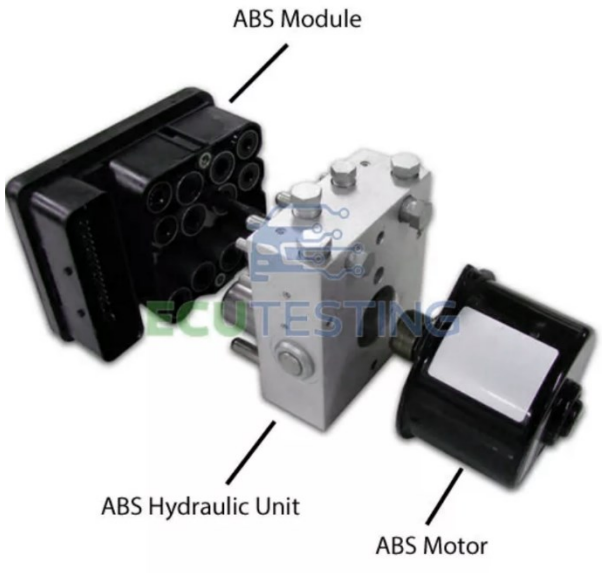



Figura 2. Sistema di pompa ABS con centralina.

- **Il processore:** dopo che la centralina ha raccolto i dati provenienti dai sensori ( in ), il processore elabora le uscite in base alle indicazioni del software installato sull'unità di controllo. Inoltre ha il fondamentale ruolo di **registrare** le proprie informazioni, come le regolazioni per la miscelazione e il chilometraggio.
- **Le uscite:** le centraline elettroniche agiscono sul motore al fine di consentire precisamente l'erogazione di potenza necessitata, controllare l'impulso degli iniettori di carburante, passando dall'apertura di un corpo **farfallato elettronico per il monitoraggio dell'otturazione dell'aria nel processo di combustione**, all' attivazione di una ventola per il raffreddamento del radiatore e molte altre.
- **Gestione della potenza:** oltre al ruolo di gestire la potenza meccanica alla centralina è affidato il ruolo di monitorare il voltaggio da fornire alla componentistica elettronica che compone il quadro dei sistemi. Infatti si passa dai 5 V continui per i sensori ai 200 V caratteristici per l'alimentazione degli iniettori di carburante. Oltre ad essere corretta la gestione della potenza deve essere sicura, infatti alcuni componenti raggiungono l'erogazione di circa 30 Amp, ciò vuol dire che il processo di raffreddamento dei componenti elettronici presenta un ruolo chiave.

Un esempio con cui mi sono interfacciato che può far capire il funzionamento della linea di comunicazione è il seguente:

se per ipotesi venissero collegati all'autovettura piuttosto che i classici **fari anabbaglianti alogeni** ( 12V per 50 W ) dei **LED** con consumi di corrente minori la centralina se ne accorgerebbe in maniera inevitabile. Dopo aver conferito l'alimentazione che permetterebbe di mettere in funzione i LED questi dopo poco si spegnerebbero per poi riaccendersi in loop finché, nell'estrema condizione, imbattersi in malfunzionamento e bruciarsi. Questo è dovuto alla comunicazione tra fornitore di tensione necessaria per accendere i fari ( la centralina ) e l'utilizzatore ( i LED ). Fornita la tensione necessaria all'accensione vi sarà un messaggio di ritorno da parte dell'utilizzatore, detectato in maniera **anomala** dalla centralina ( dovuto al minore assorbimento di energia )**.** Tale dato verrà segnalato tramite lo **spegnimento dei LED** o **tramite segnali di errore nella lettura dati**. Al contempo però i LED continueranno a richiedere tensione e la centralina continuerà ad inviarla agendo allo stesso modo in una sorta di loop deleterio.

[Problema che però può essere tranquillamente risolto in quanto alla richiesta di x < 12 Volt basterà bilanciare e simulare il carico ( non risultante dai dati preimpostati nella centralina ) collegando dei resistori].

Il PC di bordo ha quindi un importante compito: quello di detectare prima e confrontare poi i dati che vengono comunicati dai componenti utilizzatori rispetto ai valori preimpostati ( caratterizzati da curve ). Qualora le curve

rivelate non corrispondessero ai margini stabiliti **a priori** il display digitale agirà di conseguenza avvisando il proprietario o il tecnico dell'anomalia.

Si è riuscito quindi negli anni a diminuire sempre di più la tolleranza dello sfasamento dei valori ammessi **guadagnandone in sicurezza** per il conducente e in termini di **manutenzione** per l'autovettura.

Inoltre, essendo la linea CAN-bus una vera e propria rete di comunicazione dati (per intenderci simile al cavo LAN dei PC ), la centralina che usufruisce di questi dati può essere considerata come un vero e proprio computer in quanto, negli ultimi tempi, essa è dotata di un sistema operativo, è in grado di eseguire programmi ed è collegata in rete con altre centraline.

Con l'introduzione delle restrizioni riguardanti le emissioni, alle centraline elettroniche sono stati collegati dei sistemi di monitoraggio ormai imprescindibili dalle autovetture moderne. Eccone elencati alcuni:

- Sistema di riciclo dei gas di scarico ( EGR ), fondamentale per i motori **turbo**;
- Convertitore catalitico e riduzione catalitica selettiva;
- Reazione all'iniezione all'aria di scarico ( scarico AIR );
- Filtro antiparticolato diesel;
- Turbocompressore e supercharger;
- Stratificazione del carburante
- …

Il ruolo sicuramente più importante che ha acquisito l'ECU negli ultimi 20 anni è sicuramente quello di memorizzazione di dati e dei cosiddetti **codici di guasto.** Difatti se viene registrato un valore fuori tolleranza rispetto ai valori salvati sul software esso viene registrato così che il tecnico possa recuperarlo e ancora più importante viene **"bypassato"** dalla logica del software riducendo l'efficienza del motore fino ad arrivare all'estrema condizione di spegnimento di quest'ultimo. Oltre a questi codici il tecnico può monitorare in live i valori rilevati dai sensori che possono essere **fuori tolleranza senza far scattare un codice di guasto** e regolarli di conseguenza.

Tornando finalmente a ciò che concerne la mia esperienza ho iniziato a lavorare con un **simulatore di centralina elettronica su un motore di un autovettura elettrica** per poi passare alla centralina di **un motore endotermico**, precisamente di una **Ford Fiesta del 2017.** Nei prossimi capitoli entreremo nel dettaglio.

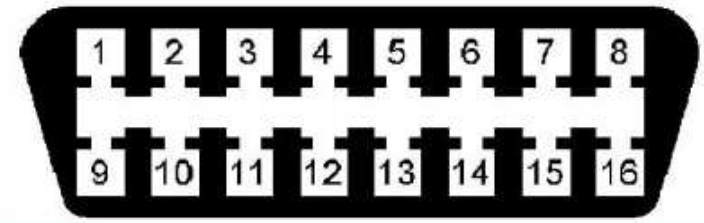
# Capitolo 2:

## • Approccio di tipo conoscitivo: Protocolli di comunicazione:

Il mio primo approccio è stato di tipo informativo in quanto non ero a conoscenza dei protocolli di comunicazione che avrei da lì in poi utilizzato. Per quanto riguarda il motore elettrico ho utilizzato un simulatore di ECU inviato dall'azienda di Imola, esso **simula i parametri di funzionamento del motore** in stato di **carica** e **scarica** della batteria del motore elettrico. Il simulatore è ovviamente fornito di un abitacolo di tipo **EOBD** ( European On Board Diagnostic ) che comunica tramite un protocollo di comunicazione omonimo: Il protocollo **OBD-II**, che ha la capacità di **autodiagnostica e segnalazione dei problemi del veicolo**.

In generale il protocollo utilizzato per l'OBD è il **CAN** ( Controller Area Network ) e già da qui mi sono imbattuto in una differenziazione fondamentale: il protocollo CAN può dialogare con il **"Body Computer", al quale sono collegati tutti i componenti elettrici del veicolo**. Venne introdotto negli anni '80 da Bosh e permette la comunicazione digitale di tipo "**broadcast**" ( processo di trasmissione verso e da tutti gli elementi elettronici presenti nella rete ) in tempo reale con livello di sicurezza molto elevato, tramite un **bus seriale.** Ma affinché ci sia comunicazione tra i dispositivi CAN era necessaria **un'univocità tra le autovetture e i rispettivi dati digitali**. A questa problematica si è ovviato introducendo il protocollo OBD nato espressamente come **standard per tutti i mezzi di trasporto.** Inoltre come facevo notare precedentemente l'OBD-II non richiama soltanto il protocollo bensì anche il connettore presente sul motore. Esso è fornito di alcuni PIN caratteristici che vengono riportati di seguito su un' immagine trovata sul web. Introducendo per altro il fatto che il connettore OBD-II è lo stesso per tutte le autovetture ma la comunicazione viene decodificata tramite cinque differenti protocolli:

- VPM ( Variable Pulse Width ): General Motors
- PWM *(Pulse-width modulation)*: Ford
- ISO 9141: Chrysler in Asia ed Europa
- ISO 14230 KWP2000 (Keyword Protocol 2000 su linea K)
- ISO 15765 (tramite linea CAN, come SAE-J2284




| Pin. | Pin description OBD-II (female) |
|---|---|
| 1 | Vendor Option |
| 2 | Bus + (SAE J1850 VPWM) (SAE J1850 PWM) |
| 3 | Vendor Option |
| 4 | Chassis Ground |
| 5 | Signal Ground (K-Line / L-Line) |
| 6 | CAN (SAE J2284) High |
| 7 | K-Line (ISO 9141-2) (ISO 14230-4) (KWP 2000) (reading parameters) |
| 8 | Vendor Option |
| 9 | Vendor Option |
| 10 | Bus - (SAE J1850 VPWM) (SAE J1850 PWM) |
| 11 | Vendor Option |
| 12 | Vendor Option |
| 13 | Vendor Option |
| 14 | CAN (SAE J2284) Low |
| 15 | L-Line (ISO 9141-2) (ISO 14230-4) (KWP 2000) (blink codes) |
| 16 | Battery Power (+ 12 Volt) |

Transmission protocol with variable pulse modulation (SAE VPW).
System used in prevalence from General Motors.

Transmission protocol with variable pulse modulation (SAE PWM).
System used in prevalence from FORD.

Protocollo di trasmissione a modulazione di impulsi variabili (SAE J1850).
System used from GM and FORD in the United States until 1996.
Transmission protocol used mainly by BOSCH
Transmission protocol with serial communication (ISO 9141-2) (ISO 14230-4).

System used by: FIAT, Gruppo WV, MITSUBISHI, CHRYSLER
NISSAN, VOLVO, JEEP, DODGE,(etc).

Transmission protocol with serial communication (KWP 2000)
System used from OPEL.

Transmission protocol with serial communication (ISO 9141-2)

**Figura 3. Funzionamento PIN porta OBD.**

Ad esempio in alcune autovetture il pin 15 ( descritto come : blink codes ), se collegato a massa ( ovvero al pin 5 ), permette l'autodiagnostica delle anomalie del motore tramite lampeggio delle spie ed implica che il guidatore deve fermarsi, e così via.

- ## Comunicazione tramite rete CAN-BUS:

Intorno agli anni '80 la Bosch in collaborazione con Intel iniziarono a dar vita al primo chip ( chiamato 82256 ) ed in concomitanza con questo la **norma** per il primo protocollo esteso "CAN 2.0 B" con l'obbiettivo di riuscire a comunicare con reti CAN-bus per la diagnostica **automotive**. Il nome CAN sta a significare **Control Area Network** in grado di stabilire comunicazione seriale tra più unità di controllo elettronico ( ECU ) per **bus** ( in gergo elettronico sta a significare "collettore", quindi raccoglitore di dati ) di tipo **multicast.**

Il CAN è stato espressamente progettato per funzionare anche in ambienti disturbati dalla presenza di onde elettromagnetiche e può utilizzare una linea ( come mezzo trasmissivo ) a **differenza di potenziale bilanciata** come **l'RS-485**. L'immunità dai disturbi elettromagnetici può essere ulteriormente migliorata utilizzando cavi a **doppino intrecciato**.

Il doppino intrecciato ha come caratteristica di appunto intrecciare due cavi chiamati rispettivamente **CAN-H e CAN-L** rispettivamente con il significato di high e low. I quali sono entrambi centrati sul valore di 2,5 V e possono aumentare o diminuire di 1 V. La loro differenza ( che può essere pari a 0 o 2 V ) genera il valore binario finale che viene effettivamente rivelato. Questa logica ha un'importante applicazione, infatti come citavo prima, qualsiasi presenza di disturbi elettromagnetici influenzano in egual modo entrambi i cavi, così da non influenzare il valore binario risultante.

Risulta fondamentale la presenza di due resistenze di valore pari a 120 Ohm poste in corrispondenza dei terminali del sistema di comunicazione. Tutte le attività ausiliare ( ovvero tutte le ECU di cui la vettura necessita ) verranno collegate in parallelo con la rete. Così facendo avremo che posizionando un tester in modalità ohmetro tra i due cavi noteremo due possibili situazioni :

- CIRCUITO FUNZIONANTE : se il valore risultante del parallelo delle resistenze è 60 Ohm.
- CIRCUITO GUASTO : se il valore risultante è pari a 120 Ohm. In pratica corrisponde al fatto che una delle due resistenze non chiude il circuito.

Eccone schematizzato il circuito. ( Fonte MST tutorial_reti di comunicazione ).

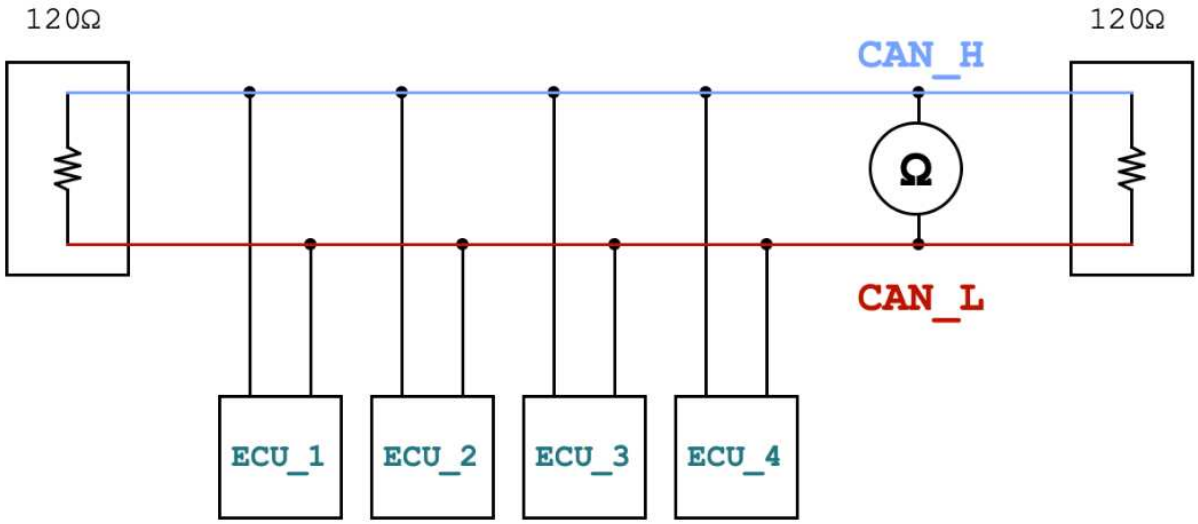



Figura 4. Rete di comunicazione CAN BUS.

Inizialmente applicata in ambito automotive, come bus per autoveicoli, la suddetta comunicazione, è attualmente utilizzata anche in ambito industriale di tipo **embedded**. Il bit rate può superare 1 Mbits/s per linee lunghe meno di 40 m oppure garantire collegamenti più lunghi con velocità limitate ( ad esempio: 500 m per 125 kbits/s di velocità ).

**Il protocollo di comunicazione del CAN è standardizzato come ISO 11898-1 ( 2015 )**. Questo standard definisce principalmente lo strato ( layer ) di scambio dati il **data link layer** , composto dallo strato sottostante di tipo "logico" : **LLC, Logical Link Control** e dallo strato ancora sottostante della Media Access Controll ( **MAC** ) e da alcuni aspetti di ordine fisico ( **Phisycal Layer** ) tutti descritti dal modello **ISO/OSI**. La configurazione dei protocolli è lasciato al progettista della rete.

- ## Simulatore di centraline elettroniche, emulatore CL2000 :

I dati su cui ho lavorato sono immagazzinati all'interno di un simulatore di centraline elettroniche, precisamente di un **motore elettrico** ( ad esempio appartenente ad una bicicletta elettrica ).

Il **dispositivo simulatore**, in quanto tale, è fornito di uno **switch** in grado di commutare dallo **stato di carica elettrica a quello di scarica**, **di una porta CAN-BUS** e di un **LED** che ci avvisa dell'inizio della comunicazione. Inoltre vengono fornite delle caratteristiche peculiari: il valore della **resistenza interna della linea CAN-bus, che come ricordavo precedentemente, è pari a 120 ohm** e **l'alimentazione 12V in DC** di cui necessita per comunicare con un emulatore che rende disponibili i dati al nostro PC.

L'**emulatore** in questione è il **CL2000** il quale è in grado ( in quanto munito di una scheda SD da 8 GB interna ) di loggare i dati sniffati e tenerli in memoria così da poterne usufruire in qualsiasi momento. I dati caricati possono essere tranquillamente inviati al nostro computer tramite l'uso di un cavo USB, inoltre abbiamo delle opzioni avanzate di processing come ad esempio il **downsampling**, **"bit-rate detection" automatico**, **cyclic logging** ( quindi eliminare ciclicamente i dati impostando una memoria parziale ) e soprattutto **il filtraggio dei trasmission message** che ci permette una cernita preliminare dei dati di cui necessitiamo rispetto ad altri.

Il CL2000 è fornito di un **tasto di reset** ( fondamentale per gli scopi successivi ), **tre LED** : di colore giallo, verde e rosso, che fungono da indicatori di eventuali problemi o modalità di funzionamento che verrano esplicitate di seguito, e di un **connettore CAN-bus.** Il funzionamento dei LED è il seguente :

1) LED verde acceso sta a significare che l'emulatore è alimentato;

2) LED rosso che lampeggia TRE VOLTE sta a significare che non è stata registrata una valida configurazione e ne verrà caricata una standard sul device.;

3) LED giallo che lampeggia TRE VOLTE sta a significare che è stata caricata una configurazione valida;

4) LED giallo che "flesha" ( lampeggia più volte ) sta a significare che la comunicazione è abilitata ( la velocità del lampeggio è dovuta alla velocità della comunicazione, ovviamente in scala );

5) LED rosso che "flesha" mi identifica che i dati vengono caricati sull'SD;

6) LED giallo rimane costantemente acceso finché non viene detectato o scelto da utente un bit rate;

7) LED giallo e rosso sempre ON se viene identificato un problema di comunicazione.;

Una delle migliori caratteristiche del CL2000 è la possibilità di caricare i dati loggati direttamente sull'SD, ciò rende più semplice la comunicazione in quanto non vi è necessità di utilizzare un software specifico per la comunicazione live dei dati ( come avviene ad esempio utilizzando Arduino o Raspberry). Comunicazione resa ancora migliore dalla presenza del **bit rate detected** automatico il quale garantisce che venga prelevata la **frequenza di comunicazione** dal solo primo scambio di dati.

**Due sono i metodi di comunicazione** a disposizione dell'utente:

- **MSD mode**: Se il dispositivo è alimentato tramite il cavo USB collegato a un PC, il dispositivo attiverà il "**Mass storage device mode**" ( MSD mode ). In questa modalità la memoria interna sarà riservata per l'accesso al PC. Il computer riconoscerà il dispositivo come memoria USB standard in modo tale che la memoria del dispositivo diventi facilmente accessibile. I dati registrati possono ora essere trasferiti al PC senza la necessità di software specifici. Inoltre, **è possibile accedere alla configurazione del dispositivo** e modificarla.
- **Logging mode**: Se il dispositivo è acceso senza il cavo USB collegato a un PC (l'alimentazione viene quindi fornita tramite il connettore CAN-bus), verrà attivata **la modalità di registrazione** ( Logging mode ). Quando il dispositivo entra in questa modalità la memoria interna è riservata alla registrazione dei dati e non è accessibile tramite la porta USB. L'interfaccia seriale si attiva se il cavo USB viene inserito dopo che il dispositivo è entrato in Logging. Quando il dispositivo entra in modalità di registrazione, tenterà di leggere il file di configurazione del dispositivo situato sulla scheda di memoria. Se un il file di configurazione viene trovato, la configurazione viene caricata sul dispositivo. Se non è presente alcuna configurazione valida trovata, ne verrà loggata una  predefinita sulla scheda di memoria.

La differenzazione è dunque semplice ed è spiegata da questa immagine presa dal **datasheet del dispositivo CL 2000** :

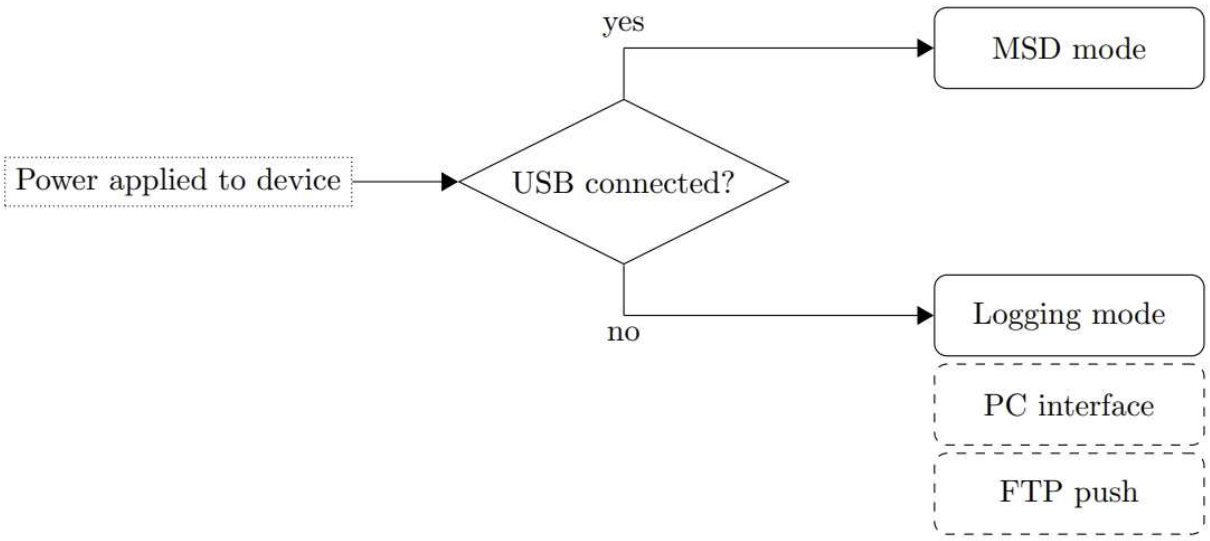



Figura5. MSD e Logging mode.

Inoltre, come facevo notare precedentemente, viene utilizzato come connettore uno di tipo CAN ( precisamente il DB-9 ) che si trova sul retro del dispositivo. Il quale ha le seguenti caratteristiche ( immagine sempre presa dal datasheet del CL2000 ) :

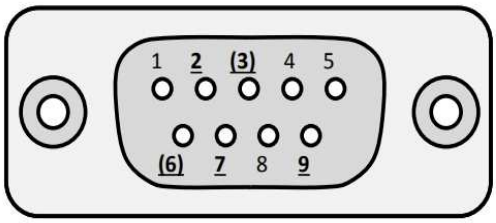



Figure 6: Front view of the CLX000 CAN-bus connecto

| Pin No. | Function |
|---|---|
| 1 | Not connected |
| 2 | CAN-L |
| (3) | GND (3 and 6 internally connected) |
| 4 | Not connected |
| 5 | Not connected |
| (6) | GND (3 and 6 internally connected) |
| 7 | CAN-H |
| 8 | Not connected |
| 9 | Device supply (see section 3 on page 2) |

Figura 6. Funzionamento PIN porta DB9.

# Capitolo 3 :

- **Configurazione dispositivo :**

Prima di tutto ho intenzione di specificare tutte le caratteristiche fisiche del dispositivo. Mi preme dire che bisogna mantenere alta la soglia dell'attenzione quando sussiste il collegamento tra il dispositivo, il PC ( tramite USB ) e l'alimentazione ( 12V DC ) in quanto la differenza di voltaggio tra il connettore DB-9 e la rete USB potrebbe non essere collegata a terra. Infatti, a differenza dei computer fissi che sono collegati a "ground", i laptop non lo sono. Mantenendo quindi il focus su come scollegare gli elementi per evitare loop di tensione che possono danneggiare irreparabilmente il dispositivo.

Altre caratteristiche fondamentali :

- **L'ISO 11898-2 :** definisce i requisiti fisici di base di una rete CAN-BUS ad alta velocità, eccone alcuni : lunghezza massima delle righe del messaggio ( dovuta al bit rate ) presente nel network del CAN-BUS, line termination ( 120 Ohm di resistenze alla fine della linea di trasmissione dati ), twisted data lines, Ground offsets range -2V to +7V.
- **Grounding :** lo standerd ISO 11898-2 tollera un certo livello di offset del ground tra i nodi. Per garantire che l'offset rimanga all'interno dell'intervallo, **si consiglia di utilizzare un singolo punto di riferimento di terra per tutti i nodi collegati al CAN-bus.** Ciò potrebbe richiedere che il filo di terra venga trasportato insieme ai data lines.
- **Lunghezza dello stub :** considerando lo stub come la lunghezza dal "main data line wire" ai nodi connessi al CAN-BUS si consiglia di mantenerlo il più corto possibile.
- **Supply quality :** la **tensione nominale** deve essere mantenuta sempre entro le specifiche. Il dispositivo è internamente **protetto contro gli eventi di bassa tensione** che potrebbero essere causati da rumore generato dal filo di alimentazione, eventi di ESD ( scariche elettrostatiche ), etc. Se la linea di alimentazione è condivisa con carichi induttivi, appunto, è necessario prestare attenzione che picchi di tensione non raggiungano la fase di logging. Ovviamente nell'ambito automotive ci si può facilmente imbattere in problematiche di questo tipo, come ad esempio : scollegamenti della batteria, contatti tra relè indesiderati, la presenza dell'alternatore etc..

Come primo passo per la configurazione del dispositivo nel datasheet c'è espressamente specificata la procedura da seguire per la buona riuscita del caricamento del **firmware** sui circuiti:

1) In primo luogo, dopo aver verificato la disponibilità del PC dei driver adeguati per la configurazione ( previa altrimenti aggiornamento e/o ricerca di quelli specifici **Driver Firmware Upgrade** ) , ho scaricato sul computer il firmware indicato;

2) Successivamente mi sono assicurato che l'emulatore ( CL2000 ) non fosse alimentato;

3) Premo il tasto di **reset** ( anche detto **DFU button** pulsante del Driver Firmware Upgrade);

4) Collego il device al cavo USB e il cavo al PC per alimentare il device;

5) Il CL2000 entrerà di conseguenza nella **DFU mode** ( e ce ne accorgeremo in quanto **i LED giallo e verde rimarranno spenti**, mentre soltanto **quello ROSSO sarà acceso** );

6) Rilasciare il pulsante DFU;

Caricato il firmware vengono automaticamente impostate le così dette **Phisical Layer** che corrispondono a :

1) Bit rate: 115200 bit/s.
2) Data bits: 8.
3) Stop bits: 1.
4) Parity bit: None.
5) Flow control: None.

Viene specificato il **Data Link Layer** come il metodo per avere una trasmissione sicura e libera da qualsiasi tipo di errore ed ha una configurazione ben precisa data dalla presenza dello start e stop message così da rivelare precisamente il messaggio informativo chiamato **Payload**. Inoltre fondamentale è il valore del **CRC** ( acronimo di **Cyclic Redundancy Check** ) che rappresenta una funzione alla quale viene dato in ingresso generalmente una serie di byte ( byte frame ) e risponde calcolando un certo valore numerico dalla lunghezza di 8/16/32… bit. Ne calcola inoltre il **MSB** e **LSB** ( **most** e **less significant byte** ) noto il messaggio del Payload. Così facendo si ammortizza di gran lunga la possibilità di commettere errori di interpretazione del messaggio.

| **Start flag** | **Payload** | **CRC** | **Stop flag** |
|---|---|---|---|
| 0x7E | ... | MSB\|LSB | 0x7E |

L'inizio e la fine di un frame sono sempre definiti in modo univoco dal flag start/stop con un valore di un byte **0x7E**. Per decodificare il frame del messaggio, il destinatario deve esaminare i dati tra i flag di avvio/arresto. Se il byte di controllo viene rilevato, il mittente del messaggio deve eseguire le seguenti azioni:

1) Scartare il byte di controllo ( 0x7E ).
2) Ripristinare il byte seguente integrando il suo 5° bit ( indicizzato da zero ).

Dopo aver “riempito” il Data Link Layer del dato ne verrà verificata l’integrità (**Frame integrity verification** ) tramite il calcolo del CRC noto il Payload, così facendo nell’ipotesi in cui il MSByte e LSByte precedentemente caricati dovessero combaciare con quelli appena calcolati dal CRC ( **CRC16 check sum** in questo caso ) il dato verrà considerato valido.

[ CRC16 check sum che corrisponde precisamente a questo polinomio: **x^16 + x^15 + x ^2 + 1=0** con rappresentazione pari a 0X805 ].

**CAN-bus message received**:

L’application data contiene un messaggio ricevuto dal dispositivo con il seguente formato:

| **ID** | **Time** | **Time ms** | **Message ID** | **Data Length** | **Data** |
|---|---|---|---|---|---|
| 1 | 4 byte | 2 byte | 4 byte | 1 byte | 0…8 byte |

- **Log file :**

Dopo aver lavorato sulla configurazione del firmware e sulle caratteristiche di lettura possiamo tranquillamente collegare l'emulatore ad un qualsiasi device di generazione dati, che sia una centralina elettronica vera e propria come un semplice simulatore di quest'ultima. Nell' ipotesi di collegamento in **MSD mode**, avendo quindi precedentemente sniffato e caricato i dati sull'SD, avremo uno scheletro del cosiddetto **log file** di questo tipo :

# Logger type : CL2000

# HW re v : 6 . xx

# FW re v : 5. 3 1

# Logger ID : OPEL

# S e s si o n No . : 70

# S p l i t No . : 1

# Time : 20170101 T022232

# Value s e p a r a t o r : ”;”

# Time fo rma t : 4

# Time s e p a r a t o r : ””

# Time s e p a r a t o r ms : ””

# Date s e p a r a t o r : ””

# Time and d a te s e p a r a t o r : ”T”

# Bit−r a t e : 500000

# S i l e n t mode : f a l s e

# C y cli c mode : f a l s e

**Timestamp ; Type ;ID ;Length;Data**

Data: 01 T022231683 ; 8 ; 7 d f ; 0 2 0 1 0 d5555555555

01 T022231688 ; 0 ; 7 e8 ; 0 3 4 1 0 d00aaaaaaaa

01 T022231693 ; 8 ; 7 d f ; 0 2 0 1 0 c5555555555

01 T022231701 ; 0 ; 7 e8 ; 0 4 4 1 0 c0000aaaaaa

01 T022231783 ; 8 ; 7 d f ; 0 2 0 1 0 d5555555555

01 T022231788 ; 0 ; 7 e8 ; 0 3 4 1 0 d00aaaaaaaa

01 T022231793 ; 8 ; 7 d f ; 0 2 0 1 0 c5555555555

01 T022231801 ; 0 ; 7 e8 ; 0 4 4 1 0 c0000aaaaaa

I file di registro vengono scritti in un formato simile al **CSV** utilizzando il set di caratteri **ASCII**. Il file di registro può essere aperto direttamente nella maggior parte dei semplici editor di testo. Le voci di registro sono formattate in base alle impostazioni del **file di configurazione** e terminato da un singolo carattere "**avanzamento riga**".

- **Device configuration :**

Il **file di configurazione del dispositivo** (*.ini formato del file) viene utilizzato per configurare il comportamento del CL2000. È possibile accedere al file di configurazione da un PC quando il dispositivo è in **MSD mode**. Per ripristinare la configurazione predefinita, eliminare la configurazione dalla memoria del dispositivo e dal ciclo di alimentazione.

Un esempio di configurazione del device:

Il formato di file necessario è un semplice formato di configurazione basato su tipo testo che può essere modificato da qualsiasi semplice editor di testo. Nella configurazione del CL2000 possono essere espressi commenti tramite il (;) e può essere aggiunto dall'utente come note. I commenti non saranno interpretati dal dispositivo e non devono seguire alcuna sintassi specifica.

Le principali impostazioni seguono delle direttive :

- **logger ID :** Viene rappresentato con una stringa di identificazione del logger. La stringa viene copiata nell'intestazione del file di registro in modo tale che È possibile identificare **i file di registro di diversi logger**. Può contenere fino a 15 caratteri;

```
; CLX000 configuration file

[revision]                      ; Configuration file revision (do not modify)
revision = 14                   ; Configuration revision number

[log]                           ; Log file configuration
loggerID = id0001               ; Logger identification string (max 15 characters)
loggingEnb = true               ; Default logging state
valueSeparator = 59             ; Log file separator ASCII char (DEC)
timestampFormat = 4             ; Timestamp format ( 0 = kkk, 6 = YYYYMMDDhhmmsskkk)
timestampTimeSeparator = 0      ; Timestamp time separator ASCII char (DEC, 0 = none)
timestampTimeMsSeparator = 0    ; Timestamp millisecond separator ASCII char (DEC, 0 = none)
timestampDateSeparator = 0      ; Timestamp date separator ASCII char (DEC, 0 = none)
timeTimeDateSeparator = 84      ; Timestamp date / time separator ASCII char (DEC, 0 = none)
fileSplitSize = 20              ; File split size in MB (DEC, range: 1-512)
fileSplitTime = 0+0             ; File split time in sec (DEC, 0+0 = none, range: 60-86400)
cyclicLogging = false           ; Delete oldest stored file when full
cyclicLoggingDataLimit = 0      ; Limit on total logged data in MB (DEC, 0 = none)

[heartbeat]
heartbeatEnb = false            ; Enable heartbeat signal
extendedID = true               ; Use extended 29 bit message ID (2.0B)
msgID = 00435353                ; CAN message ID of heartbeat signal (HEX)

[control]
controlEnb = false              ; Enable control signal
extendedID = true               ; Use extended 29 bit message ID (2.0B)
msgID = 00435354                ; CAN message ID of control signal (HEX)

[dataFields]                    ; Data fields in log file
timestamp = true                ; Log message timestamp
type = true                     ; Log type of identifier
id = true                       ; Log message ID
dataLength = false              ; Log number of message data bytes
data = true                     ; Log message data

[can]                           ; CAN bus configuration
bitrate = 0                     ; CAN-bus bit rate (DEC), 0 = auto-detect
silent = false                  ; Listen-only mode

[channel1]                      ; CAN channel 1 configuration
channelEnb = true               ; Enable CAN channel
destination = 3                 ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false              ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1         ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false            ; Enable below message filtering
msgID = 00000000                ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF            ; Message ID filter mask (HEX) (0 is invalid)

[channel2]                      ; CAN channel 2 configuration
channelEnb = true               ; Enable CAN channel
destination = 3                 ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = true               ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1         ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = false            ; Enable below message filtering
msgID = 00000000                ; Message ID filter (HEX)
msgIDMask = 1FFFFFFF            ; Message ID filter mask (HEX) (0 is invalid)

[channel3]                      ; CAN channel 3 configuration
channelEnb = false              ; Enable CAN channel
destination = 3                 ; 1 = Logger, 2 = Interface, 3 = Both
extendedID = false              ; Use extended 29 bit message IDs (2.0B)
downSamplePrescaler = 1         ; Down-sampling prescaler (DEC, range: 1-256)
filteringEnb = true             ; Enable below message filtering
msgID = 00000001                ; Message ID filter (HEX)
```

Figura 7.

- **loggingEnb :** Imposta lo stato di registrazione predefinito, se è impostato su **false**, l'opzione Logger non registrerà i messaggi fino a quando non avrà ricevuto un segnale di controllo valido che abilita la registrazione;
- **valueSeparetor :** Corrisponde al valore di separazione utilizzato preso dalla tabella ASCII (nel nostro caso 59 corrisponde ad uno spazio );

- **timestampFormat :** Specifica il livello di dettaglio del timestamp utilizzato nel file di registro. Si noti che un timestamp più lungo **occuperà più spazio e ridurrà la velocità massima di registrazione**. Il formato supporta sette livelli, da 0 a 6. I separatori di timestamp possono essere selezionati utilizzando le impostazioni del timestamp. I livelli sono elencati di seguito con kkk come millisecondi;
- **fileSplitsize :** Definisce la quantità di memoria dedicata al log file ed oltre la quale verrà **ripartito** il file. La memoria utilizzabile ha un intervallo che va da 1MB a 512MB;
- **fileSplitTime :** Piuttosto che definirlo in termini di memoria ora lo definiamo in termini di tempo il limite oltre il quale viene generato un altro file. Definendo 0+0 si costringe il device a disconnettere questa modalità ( e corrisponde al PERIODO+OFFSET dal quale si sarebbe dovuti partire );
- **cyclicLogging :** L'impostazione di questa voce su **true** abilita la modalità di registrazione ciclica. Con registrazione ciclistica si intende che i file di registro meno recenti vengano eliminati quando la scheda di memoria diventa piena, consentendo alla registrazione di continuare;
- **cyclicDataLoggingLimit :** Imposta un limite alla quantità totale di dati archiviati nella memoria in MB. Quando viene raggiunto il limite, i file di registro meno recenti vengono eliminati per mantenere la totale quantità di dati memorizzati al di sotto del limite. La funzionalità richiede che cyclicLogging sia abilitato. **Il limite deve essere compreso tra 2 ∗ fileSplitSize fino alla dimensione totale della scheda di memoria;**

**HeartBeat :** Il dispositivo genera un messaggio che può essere trasmesso periodicamente ogni secondo con le seguenti informazioni nel **payload** ( messaggio da 8 byte ) :

- **Se è stata o meno abilitata la modalità di logging, the device wall time ( se supportata dall'hardware, corrisponde all'Epoch time ) e lo spazio ancora presente sulla scheda SD in MB**, ha il seguente formato :

| Byte No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Content | 0xAA | State | | Epoch Time | | | Space Left | |

**Control Signal :** La modalità logging può essere configurata durante l'esecuzione abilitando **true** utilizzando il control signal. Inoltre questo può essere utilizzato quando soltanto il messaggio di tipo CAN-bus viene catturato, mentre bisogna disabilitarlo altrimenti;

**CAN-bus configuration :** L'impostazione riguardo il messaggio di tipo CAN è fondamentale : viene posto tra le richieste la detenzione del **bitrate** che nel nostro caso poniamo a 0 affinché venga autonomamente detectato e l'impostazione **silent** che se settata a true il dispositivo loggarà messaggi CAN-bus senza imbattersi in interferenze esterne e non verranno notificati messaggi di acknowledge ( ACK );

**Channel configuration : Le sezioni [channelX] configurano i canali e il filtro dei messaggi associato**. Un messaggio CAN passerà il meccanismo di filtro se l'ID viene accettato da uno dei quattro filtri dei rispettivi canali. La sezione contiene le seguenti voci:

- **channelEnb**: Sets the enabled/disabled state of the specific channel (true is enable);
- **destination**: Messages can either be logged (1), send to the interface (2) or both (3);
- **extendedID**: Sets the ID format used by the channel (true is extended format);
- **downSamplePrescaler**: Down-samples the data according to the prescaler value;
- **filteringEnb**: Enables the filtering mechanism;
- **msgID**: The filter message ID;

- **msgIDMask**: The filter ID mask;

Il **prescaler downsampling** può essere utilizzato per limitare il numero di messaggi registrati nella scheda di memoria. Il prescaler esegue il downsampling dei messaggi in arrivo in base all'ID del messaggio. Il valore Prescaler pari a 1 disabilita efficacemente il prescaler e passa tutti i messaggi ( che vengono accettati dal filtro del canale ). Un valore prescaler pari a 2 passa un messaggio ogni due con un **specifico messaggio ID**. Il valore massimo del prescaler è 256. Il meccanismo prescaler è limitato a 10 messaggi ID per canale. Se vengono ricevuti più di 10 ID univoci ( e passati dal commandfilter ) su un canale specifico, solo i primi 10 ID messaggio univoci ricevuti vengono sottocampionati in base al valore del prescalator, i messaggi rimanenti vengono tutti passati ( come se il prescaler fosse 1 ).

- ## Software utilizzati :

Dopo aver compreso a pieno la configurazione da attuare per il buon funzionamento del CL2000 sono passato allo interrogativo riguardante il software che avrei dovuto utilizzare per rendere effettivamente leggibili i dati presenti nel simulatore e quindi poterli graficare. Svolgendo qualche ricerca mi sono imbattuto in quelli principali per lo sniffing di dati : **WIRESHARK e SAVVYCAN.**

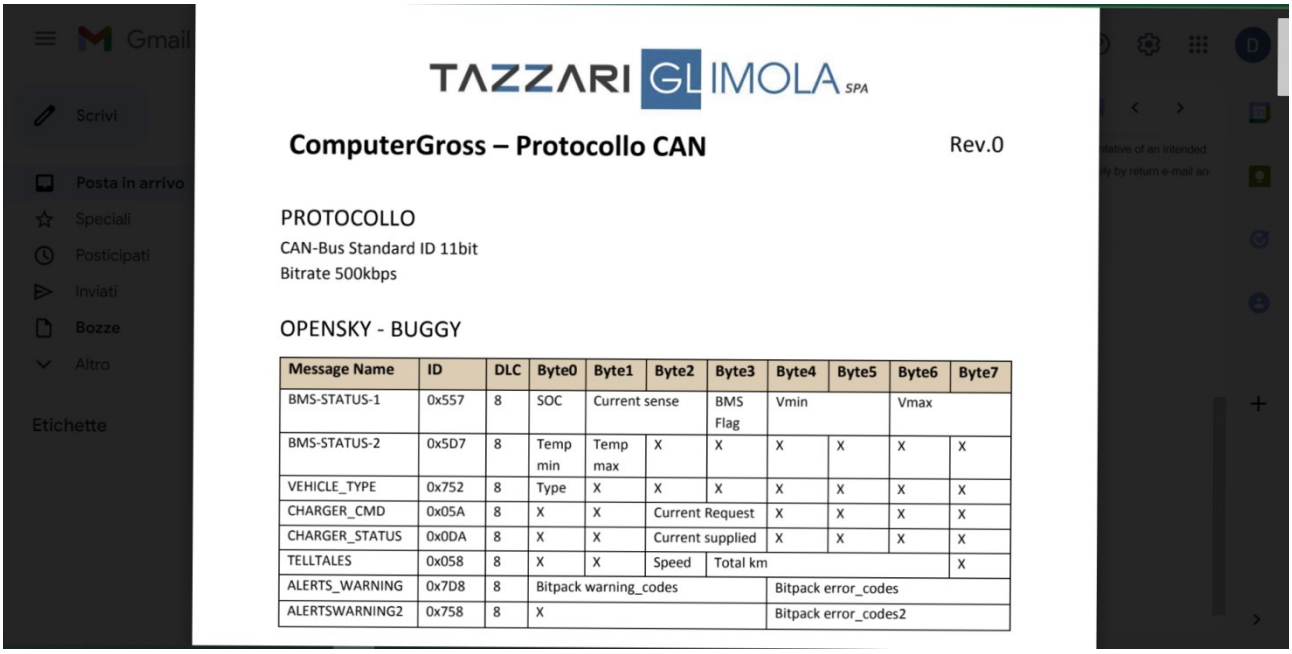
Parliamo in entrambi i casi di software utilizzati nell'ambito delle telecomunicazioni come troubleshooter per l'analisi di rete e lo sviluppo di protocolli per la didattica, avendo entrambi le caratteristiche di uno standard analizzatore di protocollo. Le funzionalità oltre ad essere di sniffing sono anche **grafiche.** Permettono all'utente di osservare in tempo reale tutto il traffico presente sulla rete ( ad esempio **Ethernet me è applicabile a qualsiasi rete fisica, come ad esempio qualsiasi che corrisponda allo standard IEEE 802.11 proprio come la CAN-BUS** ). Ottimi in quanto facilmente i dati acquisiti possono essere modificati, filtrati e/o convertiti.

C'è una caratteristica che però li differenzia : nel casi di Wireshark c'è bisogno di un software esterno che mi consenta ( letteralmente ) di avviare la comunicazione dopo la ricezione da parte del pc di una serie di **ACK data**, il software in questione è il Can Vas. Differenza che mi ha fatto prediligere SavvyCan che invece aveva la possibilità di configurare la porta di tipo COMX direttamente sul software stesso, rendendo il passaggio più breve e intuitivo.

Durante le mie prime comunicazioni su SavvyCan utilizzavo come decodifica dei miei dati acquisiti il protocollo OBD II il quale però effettivamente non mi dava i risultati sperati. Documentandomi poi ho capito che i dati presenti nel simulatore non sono presenti nel protocollo OBD II. L'alternativa era il protocollo CAN-DBC il quale iniziava a dare uno spiraglio di buona riuscita di comunicazione.

DBC è un formato di file **utilizzato per specificare come i "segnali" vengono memorizzati nei "messaggi"**. Un messaggio è essenzialmente un **pacchetto univoco di dati inviati sul CAN- bus**. Normalmente questo messaggio è differenziato dall'ID del frame. In base al frame di riferimento viene identificato uno specifico dato fisico. Purtroppo però non sono direttamente caricati i vari dati fisici a cui ogni frame dell'ID fa riferimento ( questo problema si paleserà anche e soprattutto in seguito con il prelievo dei dati dal bus di una autovettura ).

Qui subentra l'azienda Tazzari la quale mi ha allegato ciò di cui necessitavo, ovvero : ad ogni messaggio letto ( con l'ID corrispondente ) il preciso byte al quale ricollegare il dato fisico.



TAZZARI GL IMOLA SPA

**ComputerGross – Protocollo CAN** Rev.0

PROTOCOLLO

CAN-Bus Standard ID 11bit

Bitrate 500kbps

OPENSKY - BUGGY

| Message Name | ID | DLC | Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
|---|---|---|---|---|---|---|---|---|---|---|
| BMS-STATUS-1 | 0x557 | 8 | SOC | Current sense | | BMS Flag | Vmin | | Vmax | |
| BMS-STATUS-2 | 0x5D7 | 8 | Temp min | Temp max | X | X | X | X | X | X |
| VEHICLE_TYPE | 0x752 | 8 | Type | X | X | X | X | X | X | X |
| CHARGER_CMD | 0x05A | 8 | X | X | Current Request | | X | X | X | X |
| CHARGER_STATUS | 0x0DA | 8 | X | X | Current supplied | | X | X | X | X |
| TELLTALES | 0x058 | 8 | X | X | Speed | Total km | | | | X |
| ALERTS_WARNING | 0x7D8 | 8 | Bitpack warning_codes | | | | Bitpack error_codes | | | |
| ALERTSWARNING2 | 0x758 | 8 | X | | | | Bitpack error_codes2 | | | |

Figura 8. ID per la conversione dati tramite CAN-Dbc per il simulatore di motore elettrico.

Come anticipato precedentemente preferirò d'ora in poi l'utilizzo di SavvyCan: ed ecco cosa specificavo in precedenza. Parliamo della configurazione della porta COM ( in questo caso 5 ) alla quale è collegato il cavo USB in grado di permettere la seguente comunicazione.

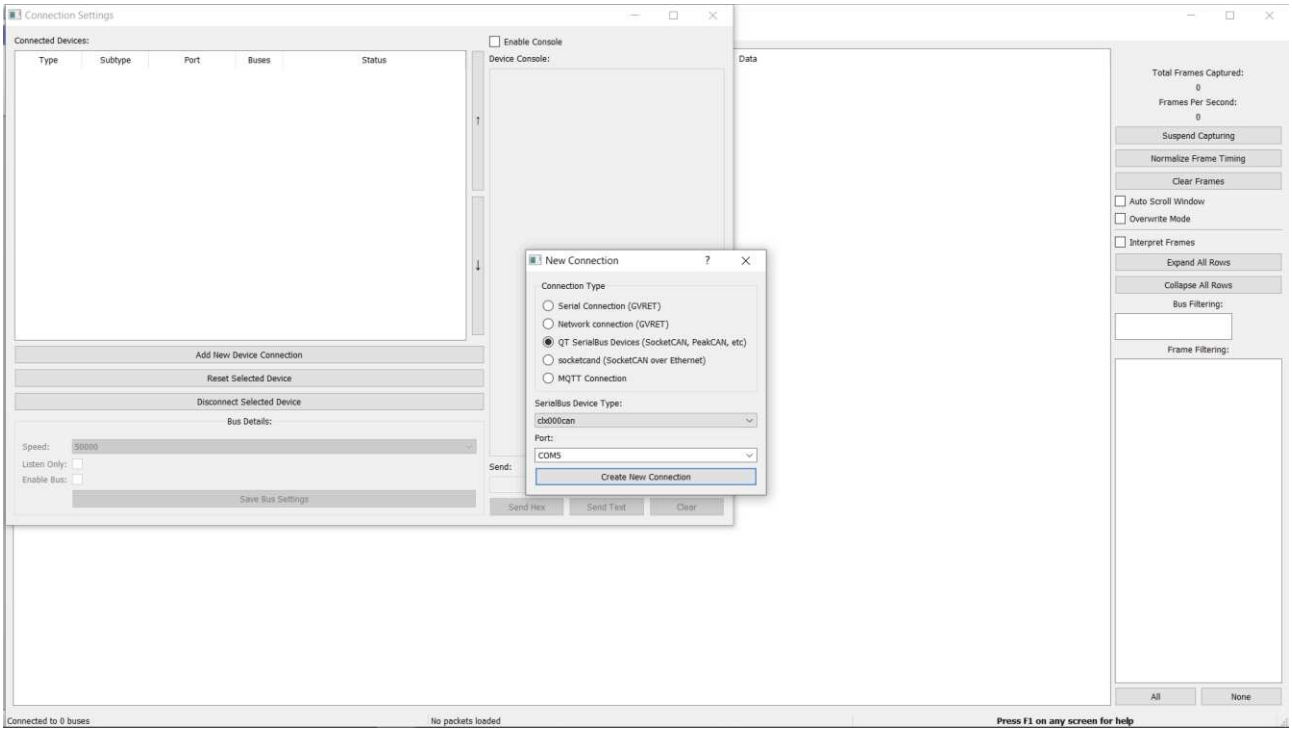



Figura 9. Immagine relativa al collegamento diretto alla porta COM5 di comunicazione su SavvyCan.

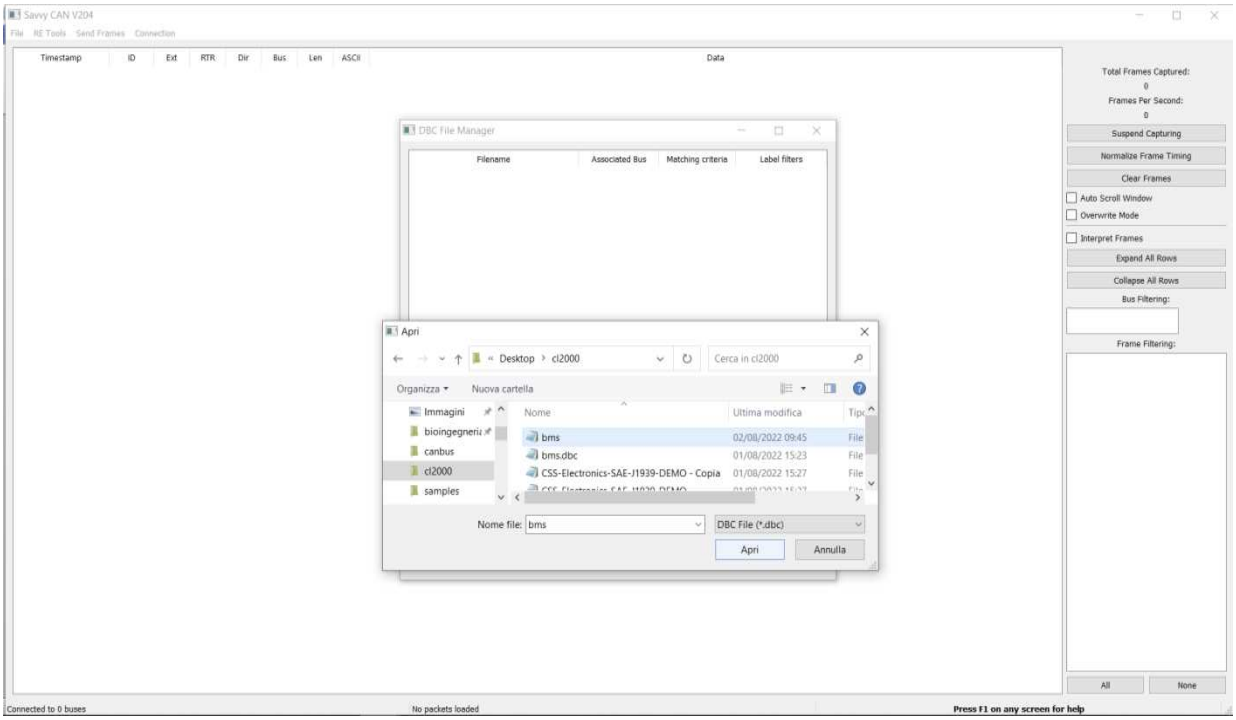



Figura 10. Immagine relativa al caricamento del file.dbc preliminare così che io possa interpretare i dati prelevati.

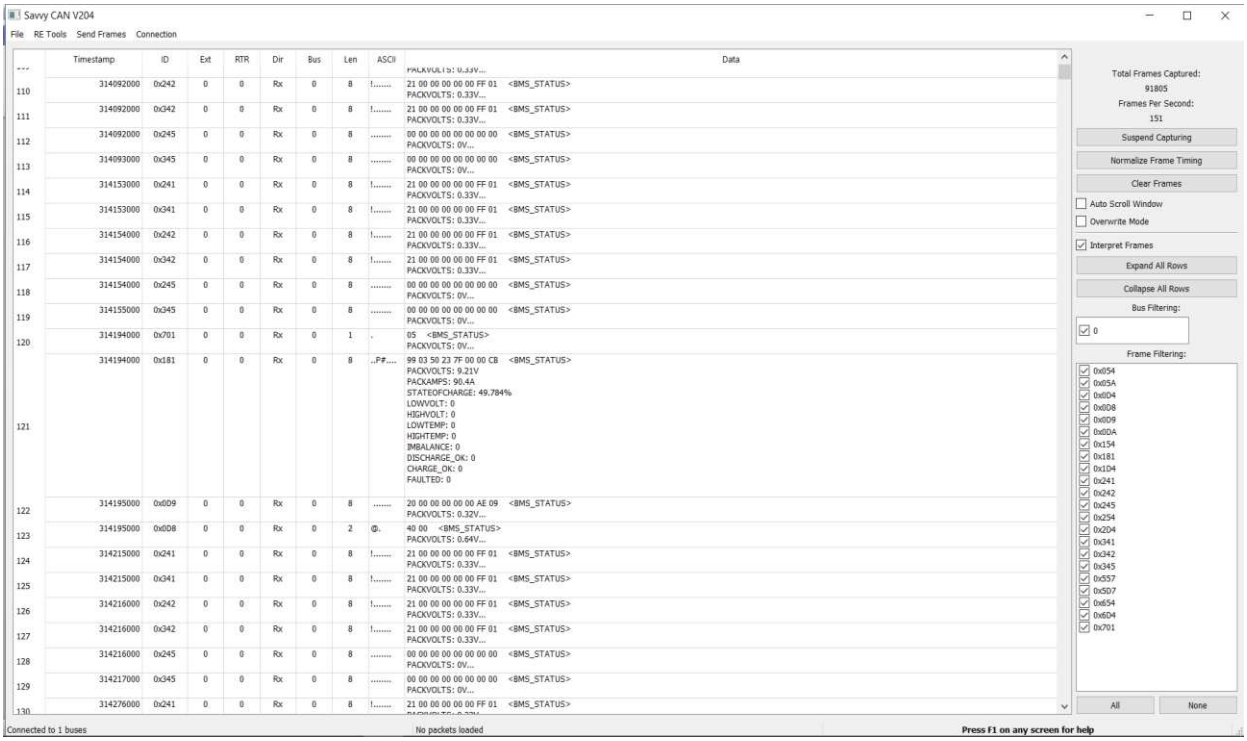



Figura 11.

- **Primo sniffing dati dal simulatore, debbugging e rappresentazione grafica:**

Di seguito verranno riportati i dati di cui l'azienda Tazzari necessitava. Al netto delle valutazioni svolte per l'utilizzo dei SW ho deciso di utilizzare SavvyCan in primis per la facilità di instaurare la comunicazione direttamente dal software e in secondo luogo perché il suddetto SW dispone di una finestra grafica integrata di agevole utilizzo. Il primo dato qui sotto riguarda lo **stato di carica (SOC) della batteria del motore elettrico.**

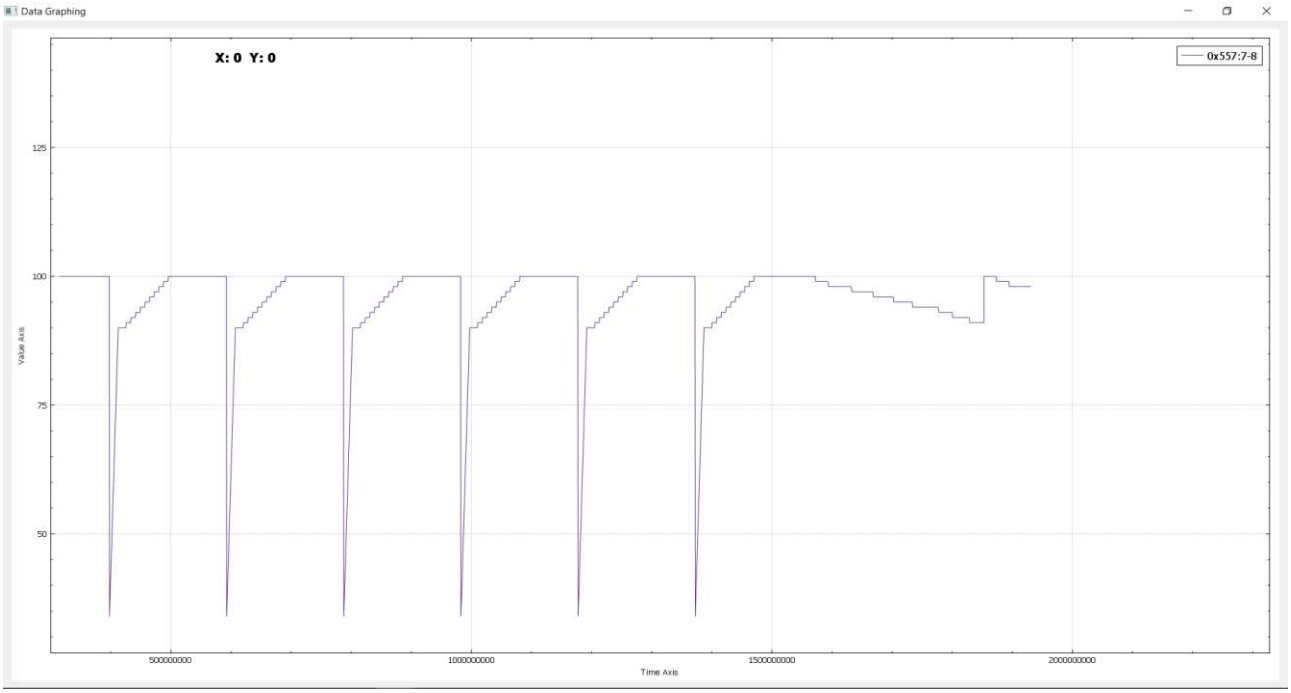



**Figura 12. SOC(state of charge) dove possiamo trovare sull'asse delle ascisse il tempo (ms), su quella delle ordinate la percentuale di carica (%mA).**

**Ovviamente i dati, in quanto accelerati nel tempo, limitati in numero ( di campioni ) e continui ( in termini che una volta comunicati interamente vengono inviati di nuovo dal simulatore in quanto la richiesta da parte del PC viene puntualmente soddisfatta dall'erogazione dati dalla centralina ), risulteranno periodici.**

Inoltre invito a notare l'esempio di scarica dopo lo switch da carica a scarica e **che i valori**, come è possibile intuire , per quanto riguarda lo **state of charge**, **sono di tipo percentuale ( % )** a seguito di un adeguato prescaler ( 100:256=x: valore sniffato corrispondente al primo byte dell' ID 0x557 ).

Un altro esempio fondamentale per gli obbiettivi che mi sono stati posti è stato il plotting del pacchetto di **voltaggio** precisamente **massimo** e **minimo** rilevato. Per definirli sono stati dedicati ad entrambi due byte di memoria (quarto e quinto byte per il V min e quinto e sesto per il V max dell' ID 0x557 ), così facendo avremo come valore massimo 65 536 ( 2^16 ) imponendo uno scaler pari a 0,0001 che rende reale il valore in questione ( in Volt ).

1) **V min ( carica ):**

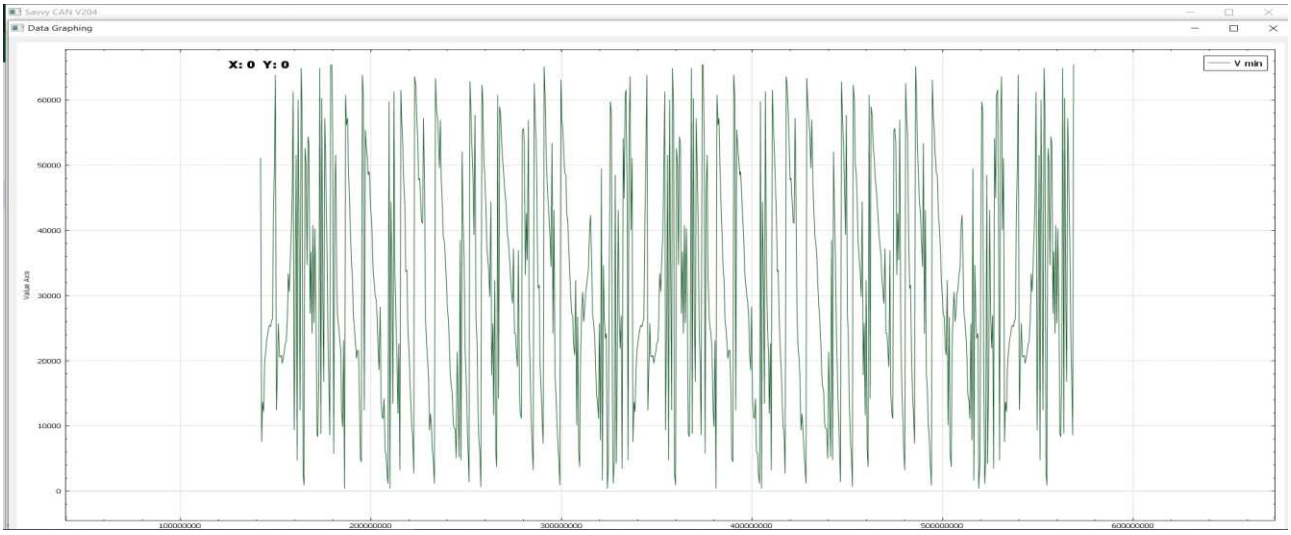



Figura 13. Vmin (fase di carica) adeguatamente prescalato, con sull'asse x il tempo (ms) sulle y il Voltaggio (V).

**2) V max ( carica ):**

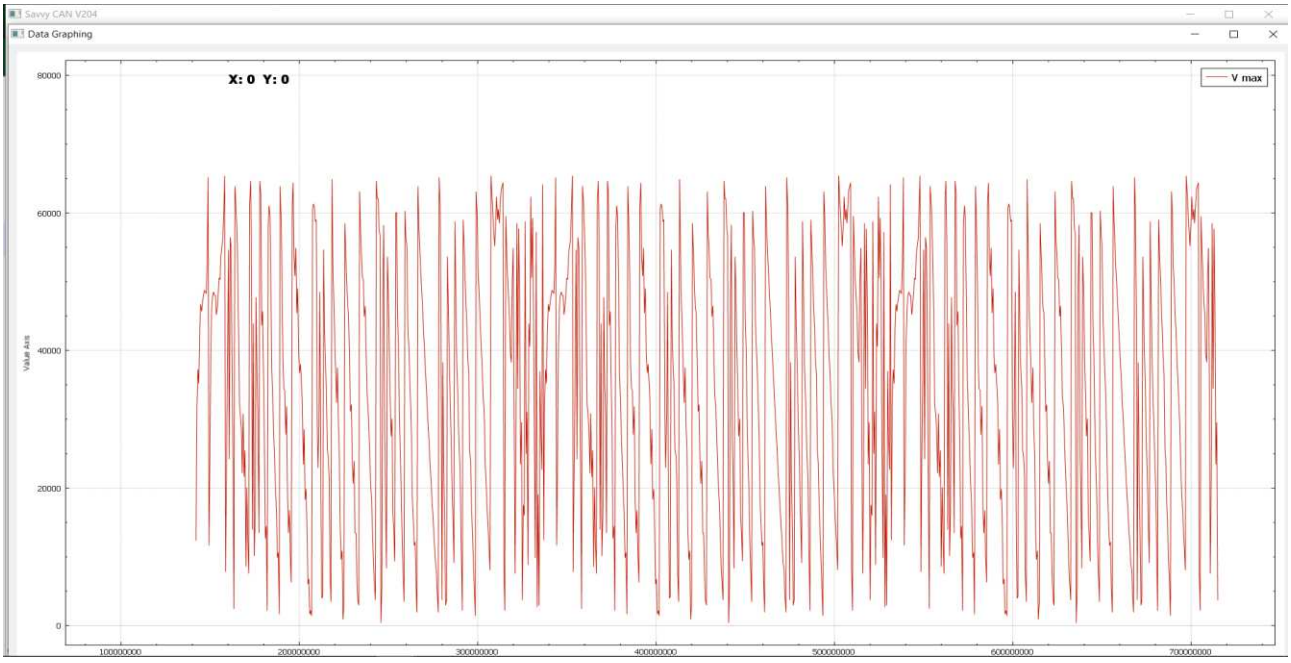



Figura 14. Vmax (fase di carica) adeguatamente prescalato, con sull'asse x il tempo (ms) sulle y il Voltaggio (V).

Contrapposti al processo di scarica che di seguito riporto:

**3) V min ( scarica ):**

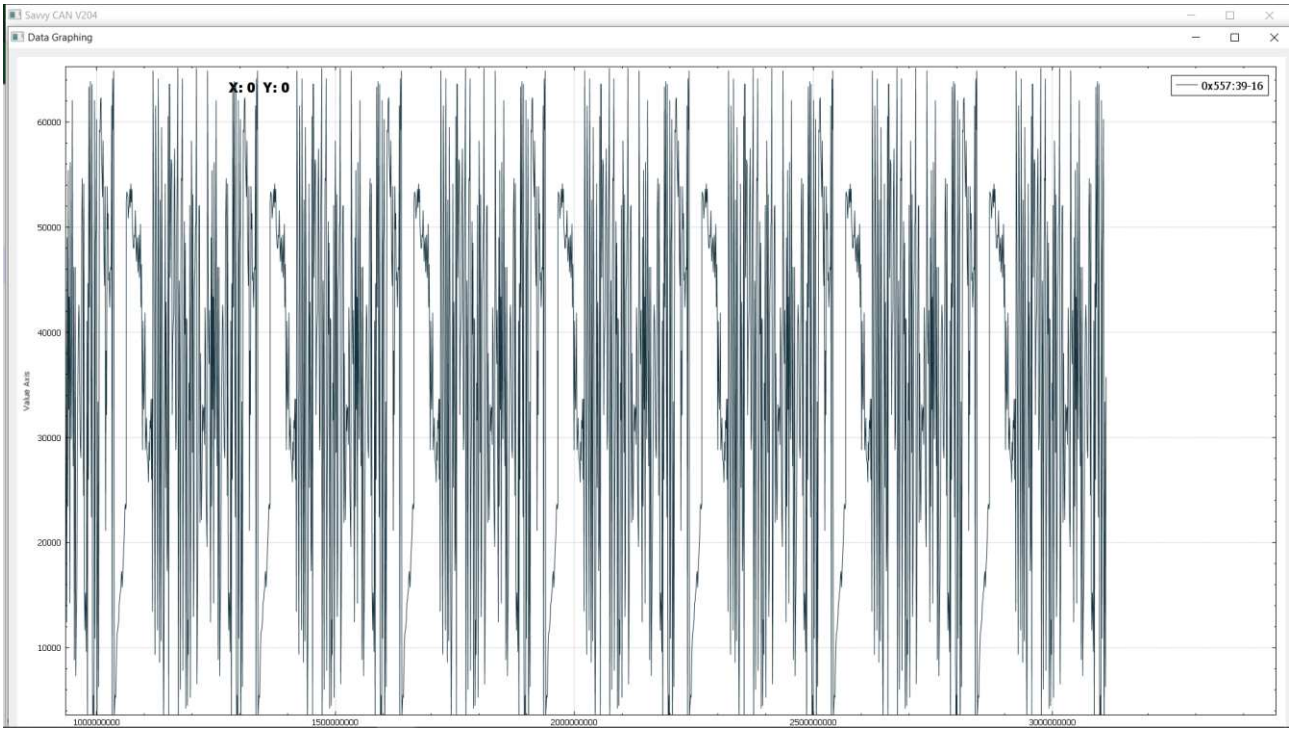



Figura 15. Vmin (fase di scarica) adeguatamente prescalato, con sull'asse x il tempo (ms) sulle y il Voltaggio (V).

### 4) V max ( scarica ):

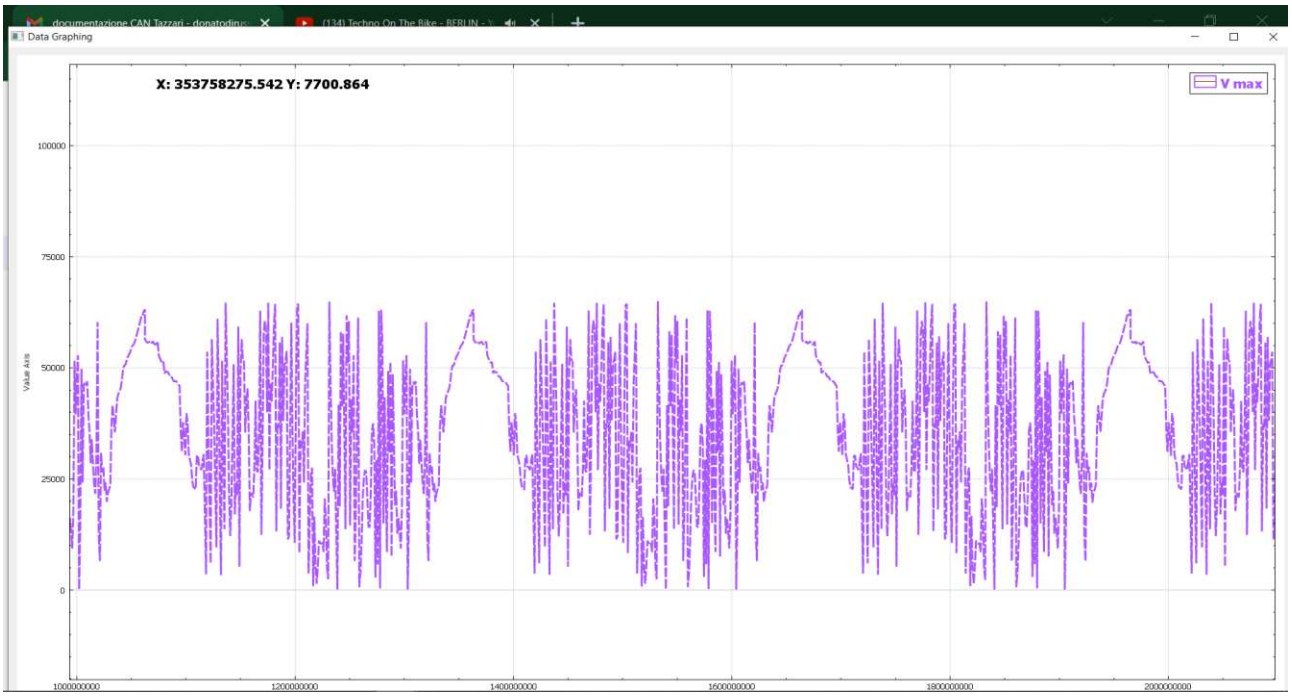



Figura 16. Vmin (fase di scarica) adeguatamente prescalato, con sull'asse x il tempo (ms) sulle y il Voltaggio (V).

Come ultimo parametro richiesto abbiamo la **corrente richiesta dal motore e quella erogata da quest'ultimo** durante il processo di carica che corrispondono rispettivamente al secondo e terzo byte dell' ID 0x05A e 0x0DA ( entrambi i valori scalati di 0.1 con unità di misura Ampere ):

### 5) Current requested:

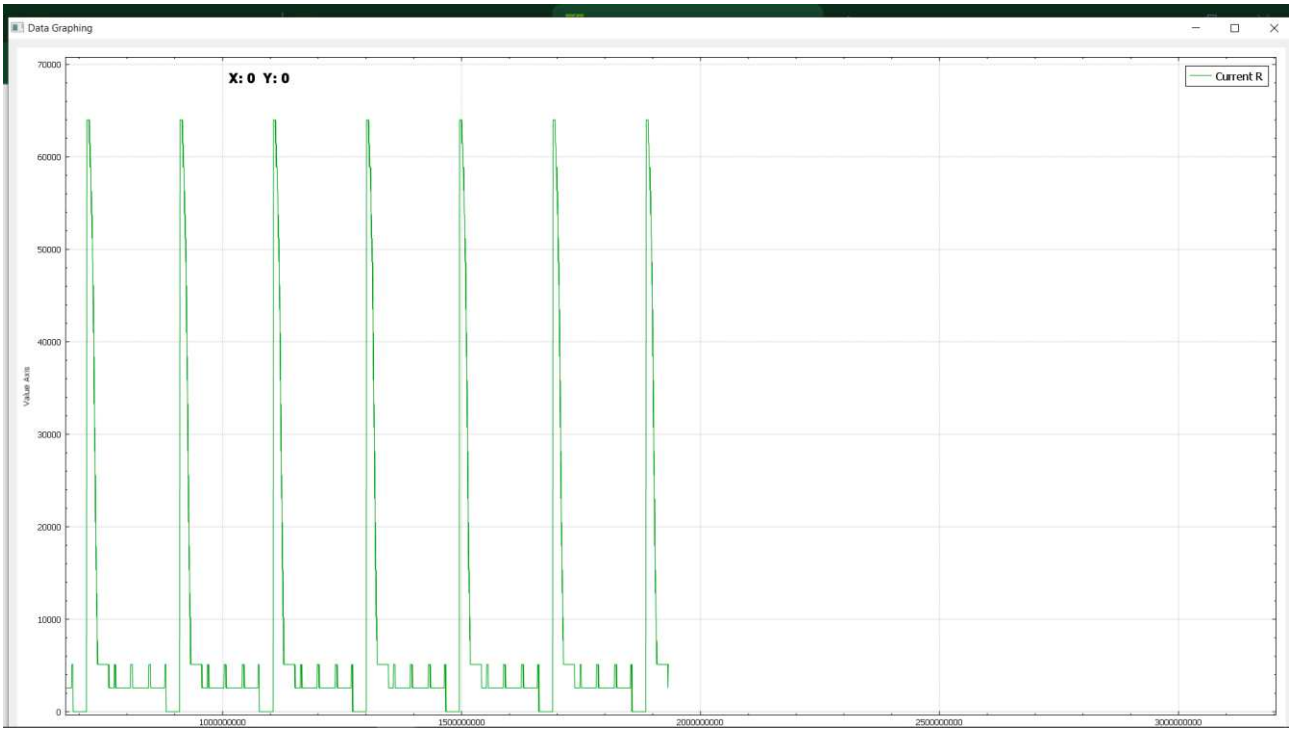



**Figura 17. Corrente richiesta dal motore, con sull'asse delle x il tempo (ms), sull'asse delle y il valore della corrente opportunamente prescalato (mA).**

### 6) Current supplied:

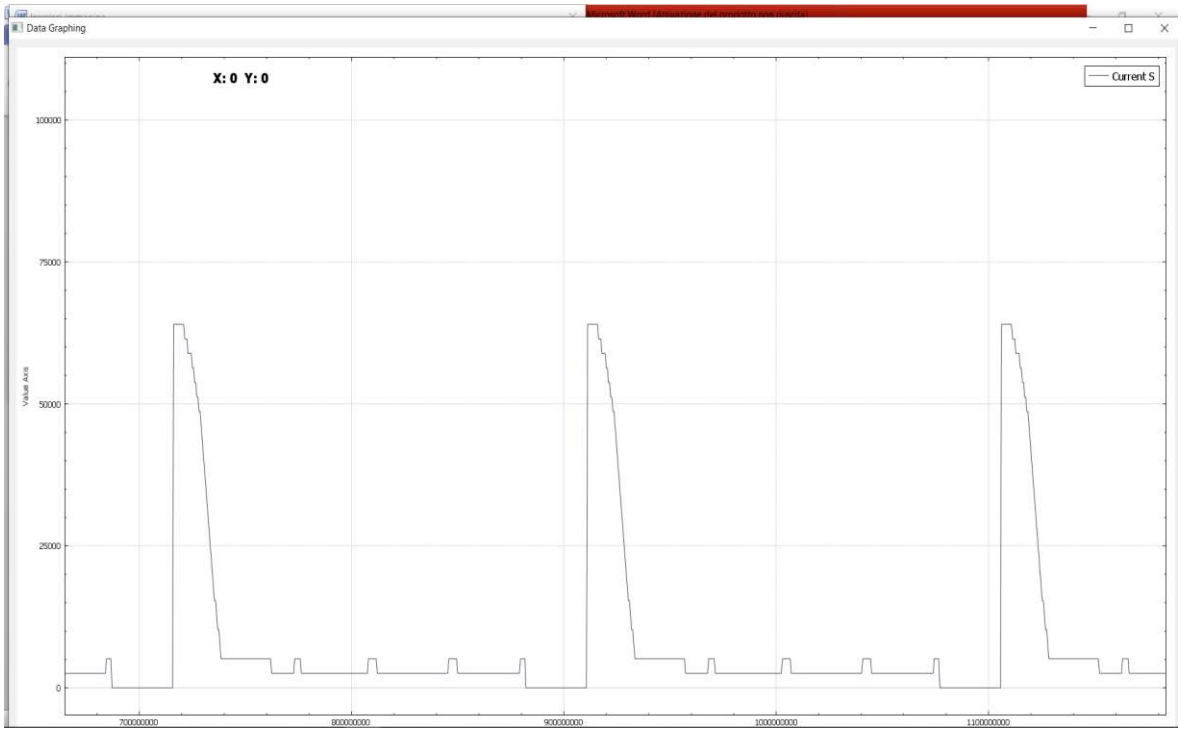



**Figura 18. Corrente erogata dal motore durante il funzionamento, con sull'asse delle x il tempo (ms), sull'asse delle y il valore della corrente opportunamente prescalato (mA).**

Questi erano i dati che venivano richiesti e ne sono stato messo a conoscenza direttamente dall'azienda Tazzari tramite un feedback molto gradito. Detto ciò avevo ancora qualche ora di tempo previste dalle normative universitarie. Quindi i ragazzi dell'azienda mi hanno reso partecipe di un altro progetto che a differenza del precedente non avrei facilmente concluso.

# Capitolo 4 :

- **Seconda commissione aziendale :**

Dopo aver raggiunto il primo obbiettivo compiendo quindi il debug dei dati generati e comunicati dal simulatore di centralina elettronica di un motore elettrico, mi è stato proposto di continuare il mio studio su un tema sicuramente più vasto e dettagliato. **Passare allo studio di un'ECU di un motore termico.**

Lo sniffing di dati automitive avviene tramite linea CAN-BUS del veicolo, a cui sono collegate tutte le centraline elettroniche che compongono il sistema di controllo elettronico dell'autovettura. La linea scambia dati continuamente secondo i protocolli di comunicazione seguenti: **OBD-II e Can-DBC.**

L'OBD è un protocollo standardizzato per tutte le automobili e, in quanto tale, presenta un pacchetto che converte dati cosiddetti **"vitali"**. Questo pacchetto, reso disponibile dalla casa produttrice stessa, è dotato di tutti i codici ID necessari per una diagnostica completa di dati inerenti alla "salute della macchina". Segnati come codici alfanumerici descritti dalle lettere iniziali **P, B, C** o **U** ne riporto alcuni esempi:

P100 Mass or Volume Air flow Circuit Malfunction (Malfunzionamento circuito flusso aria volume o massa);

C0000 Vehicle Speed Information Circuit Malfunction (Malfunzionamento circuito dati velocità veicolo);

B1200 Climate control push button circuit failure (Guasto nel circuito del pulsante climatizzatore);

I tipo U sono problemi legati alla rete di comunicazione.

Il protocollo sopra citato, però, non ha permesso di concorrere all'obbiettivo che è stato posto. Qui subentra il protocollo Can-DBC il quale, a differenza del precedente permette la cattura di dati **sullo stato dei componenti dell'autovettura.** Sta a significare che i codici presenti protocollo in questione identificano, ad esempio, la velocità dell'autovettura in movimento, i giri motore, il livello acceso o spento dei fari, l'apertura o la chiusura dei finestrini etc. proprio su quest'ultimo dato ho lavorato. L'azienda richiedeva che fosse trovato il codice corrispondente allo stato aperto/chiuso dei finestrini in modo da poterlo manovrare tramite la tecnologia RFID.

Il loro obbiettivo è quello di riuscire a dare una soluzione alla problematica **dell'abbandono accidentale di bambini nell'autovettura**. Questo è possibile implementando un sensore di pressione sotto il seggiolino sul quale è presente il bambino. Se questo rivela il peso e al contempo viene rivelata la lontananza del genitore (il quale si presuppone abbia con se l'**RFID tag ad esempio posto all'interno del telecomando dell'autovettura**), farà reagire l'**RFID reader impostato sulla macchina** che comunicherà con l'ECU e, avendo noto il codice ID corrispondente allo stato dei finestrini, li abbasserà.

Ovviamente non ero incaricato alla realizzazione del sistema, bensì a trovare il codice ID che deve essere sollecitato per la buona riuscita dell'idea.

Trovare il codice ID in quanto, come ho detto, il protocollo Can-DBC comunica lo stato dei componenti dell'autovettura, non è un' impresa semplice. **La conversione tramite Can-DBC permette si lo sniffaggio dei codici ID, ma non i corrispondenti dati fisici ai quali questi codici si riferiscono,** a meno di avere le licenze che la casa madre mette a disposizione (ovviamente a pagamento). Inoltre una cosa che rende ancora più difficile la pratica è che ogni autovettura ha i propri codici Can-DBC, quindi in base alla casa, all'anno di produzione, al segmento, al modello etc. ci saranno differenti codici.

Il mio scopo non era soltanto quello di riuscire a trovare il codice ID corrispondente allo stato dei finestrini, ma incrementare il database di corrispondenza **sicura** tra codici ID e dati fisici tramite i grafici e i valori esadecimali che mi si palesavano tramite lo sniffing. **La macchina in questione è stata una Ford Fiesta del 2017 aziendale.**

- ## Sniffing dati da un' ECU di motore endotermico:

Come primo approccio ho provato a sniffare dati da fermo così da riuscire a farmi un'idea su quale fosse il metodo migliore e meno dispendioso per prelevare i dati di cui avevo bisogno e mi sono imbattuto in una caratteristica di SavvyCan che, in tempo reale, riusciva a conferirmi i bit di quale byte venivano modificati.

**Propongo un immagine esemplificativa di un frame dell'ID 0x400:**

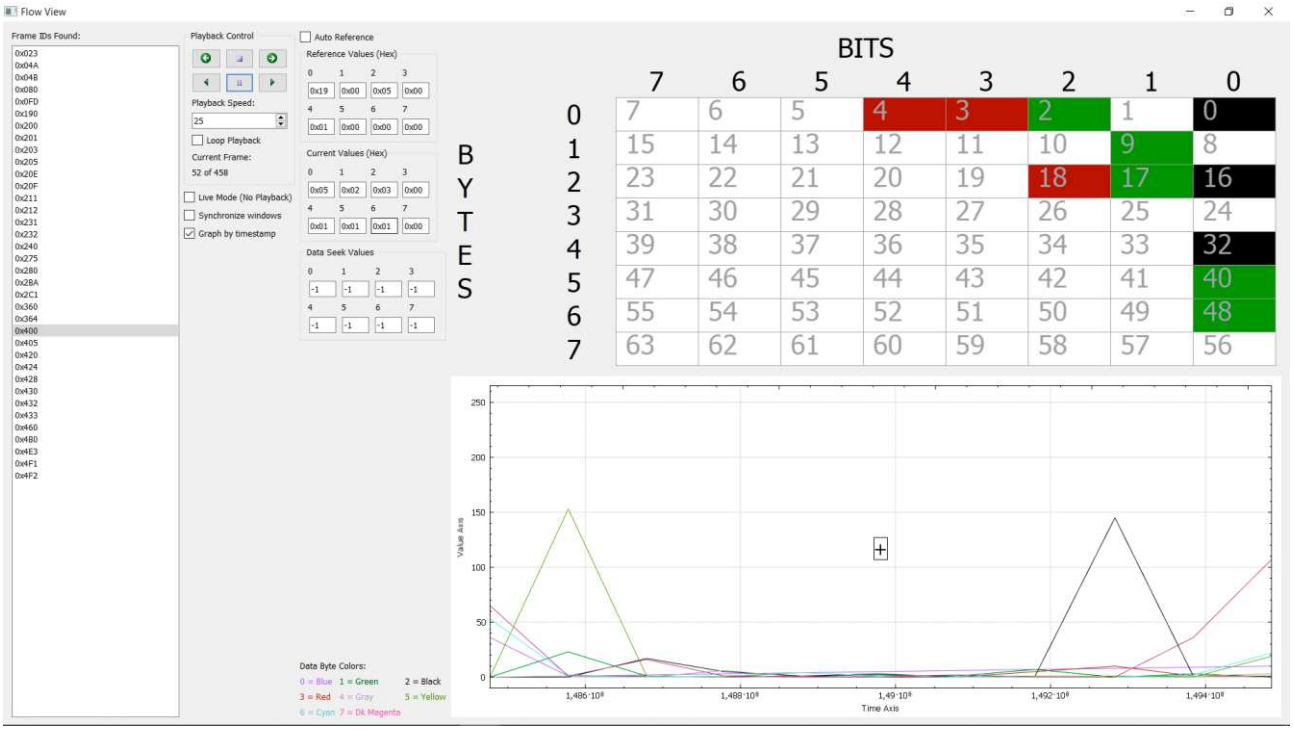



Figura 19.

Così che sono arrivato alla prima conclusione, riuscire ad attuare una cernita di quali bytes da fermo rimanessero costanti e quali invece anche da fermo modificassero il proprio valore.

Tra tutti gli ID quelli rimasti costanti potevano concorrere ad essere quello dei km percorsi, in quanto sniffati a macchina ferma, oppure dei finestrini nell'ipotesi in cui non li muovessi. Analogo ragionamento vale per quelli che invece variavano, erano papabili ID per il dato dell'iniezione di carburante ad esempio o anche stesso del consumo medio (aumentando ad esempio i giri motore oppure accendendo l'aria condizionata).

Come secondo step, fatta la cernita, inizio a plottare qualche byte specifico che mi faccia intendere cosa rappresenta e, grazie all'aiuto dei colleghi, abbiamo raggiunto il primo risultato:

infatti filtrando i valori rimasti costanti risulta esserci un dato che, convertito in esadecimale, **corrispondeva esattamente al numero di chilometri della Ford Fiesta**:

Savvy CAN V204

File RE Tools Send Frames Connection

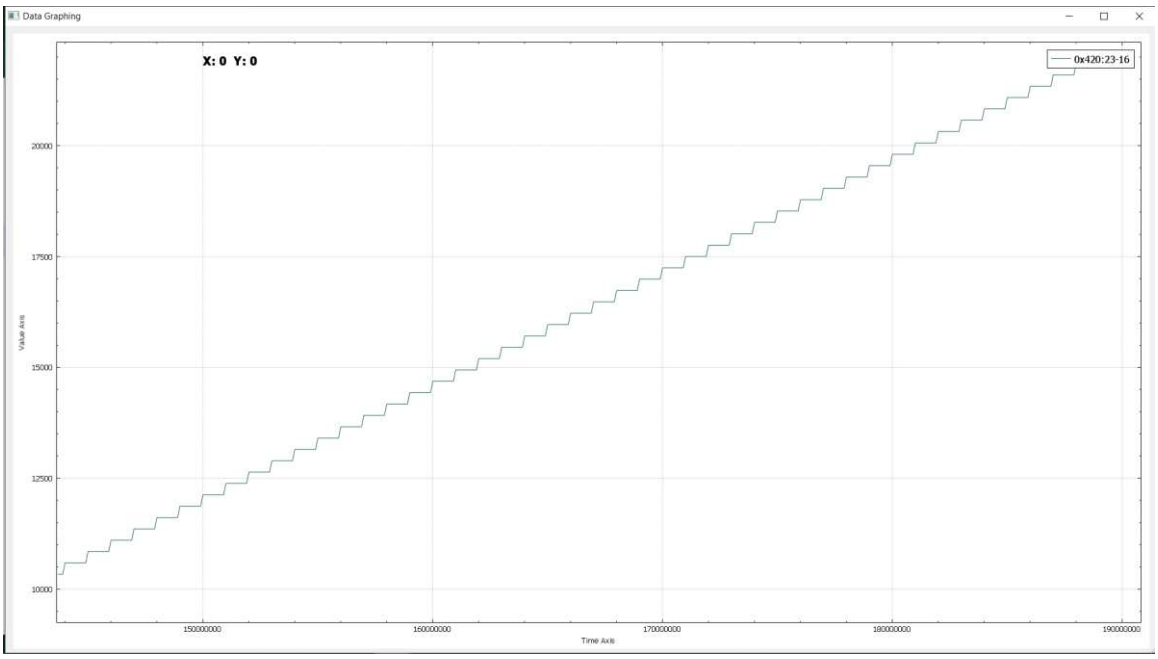
| | Timestamp | ID | Ext | RTR | Dir | Bus | Len | ASCII | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 143746000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 2 | 144755000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 3 | 145758000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 4 | 146765000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 5 | 147774000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 6 | 148780000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 7 | 149787000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 8 | 150798000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 9 | 151807000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 10 | 152809000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 11 | 153816000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 12 | 154820000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 13 | 155827000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |
| 14 | 156830000 | 0x4F2 | 0 | 0 | Rx | 0 | 8 | ..... | 03 EF AE 00 00 00 00 00 |

Figura 20.

**3 EF AE corrisponde esattamente a 257 966.**

D'ora in poi verranno proposte soltanto delle immagini di confronto tra i dati catturati a macchina ferma e quelli con macchina in movimento senza delle e vere e proprie conclusioni in quanto lo studio necessiterebbe troppo tempo per ricavare i valori fisici a cui gli ID corrispondono.

**Macchina ferma :**




**1) Graphic window ID: 0x420 dal bit 23 per due byte.**

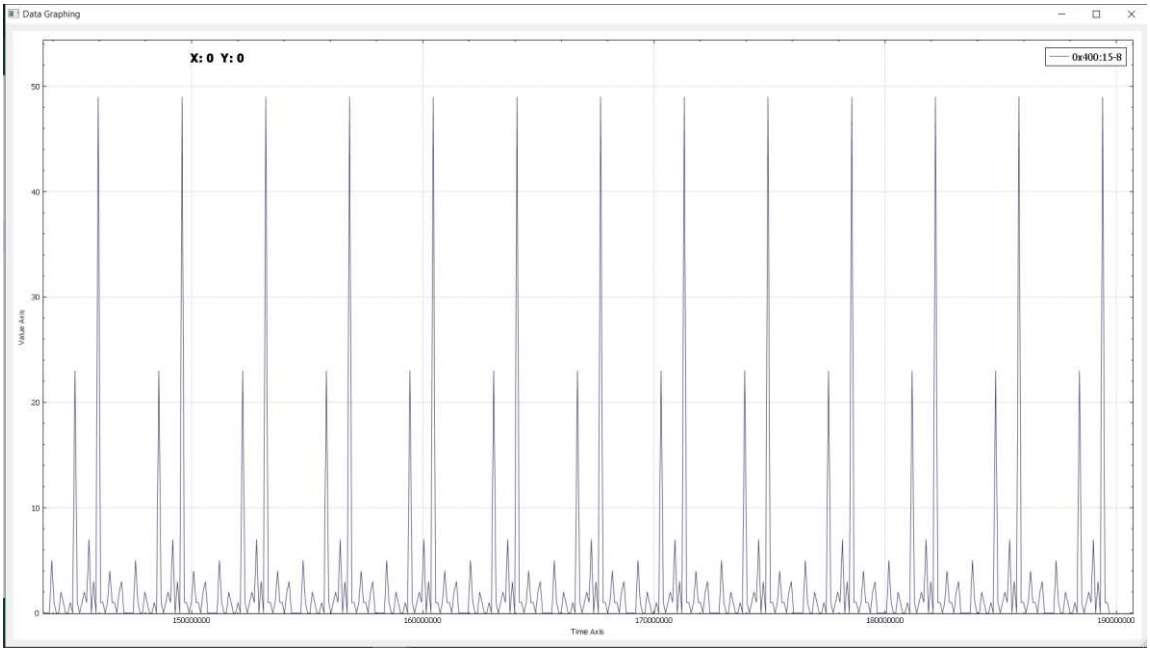



Figura 21.

**2) Graphic window ID: 0x400 dal bit 15 per un byte.**

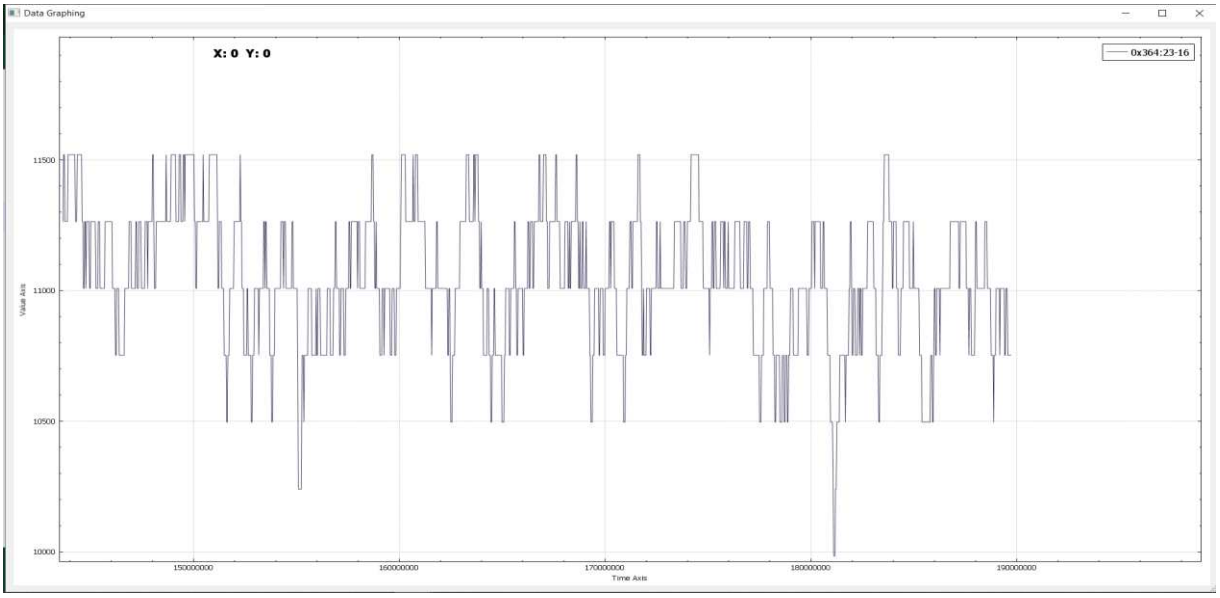



Figura 22.

### 3) Graphic window ID: 0x364 dal bit 23 per due byte.

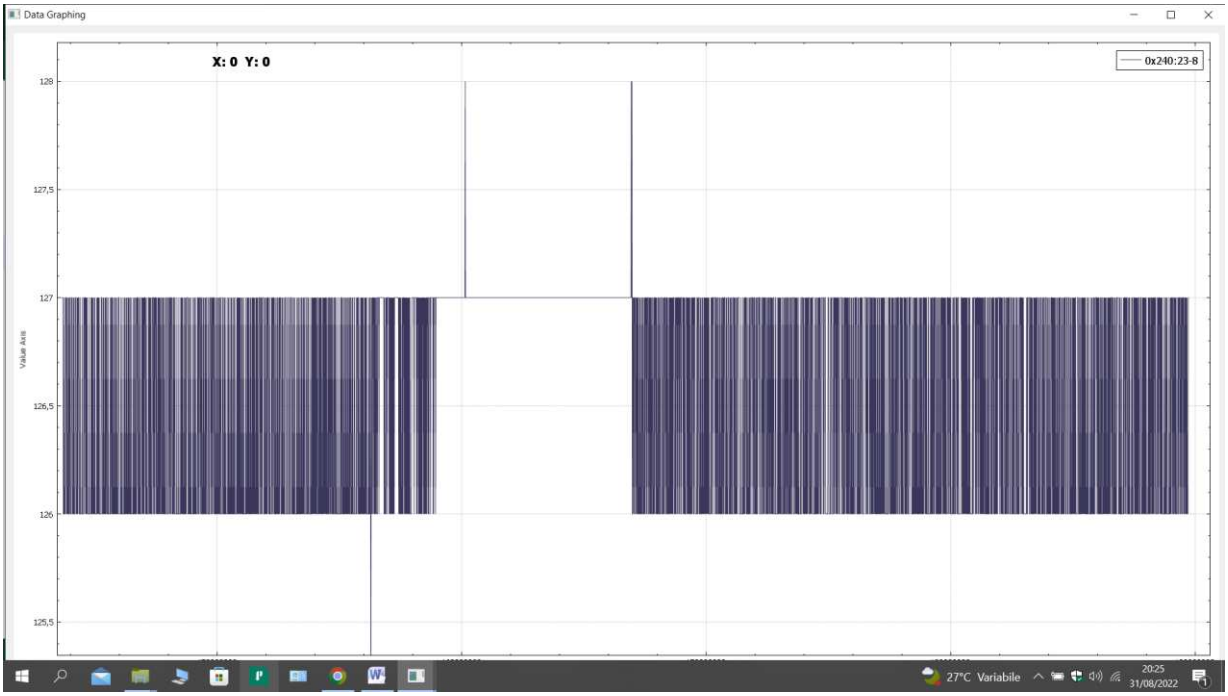



Figura 23.

### 4) Graphic window ID: 0x240 dal bit 23 per un byte.

E così per tutti gli ID sniffati, ora compio il confronto con gli stessi ID presi negli stesse bit con la stessa lunghezza di byte e vedremo come variano. Nei secondi grafici che vi sto per proporre infatti dopo un periodo limitato di stazionarietà del veicolo ( molto simile ai grafici precedenti ) risalteranno i comportamenti differenti delle curve dovute al moto. Con le dovute eccezioni ( come ad esempio l'ID 0x400 con lunghezza un byte dal quindicesimo bit ) dove notiamo lo stesso grafico in entrambi i casi.

**Macchina in movimento:**

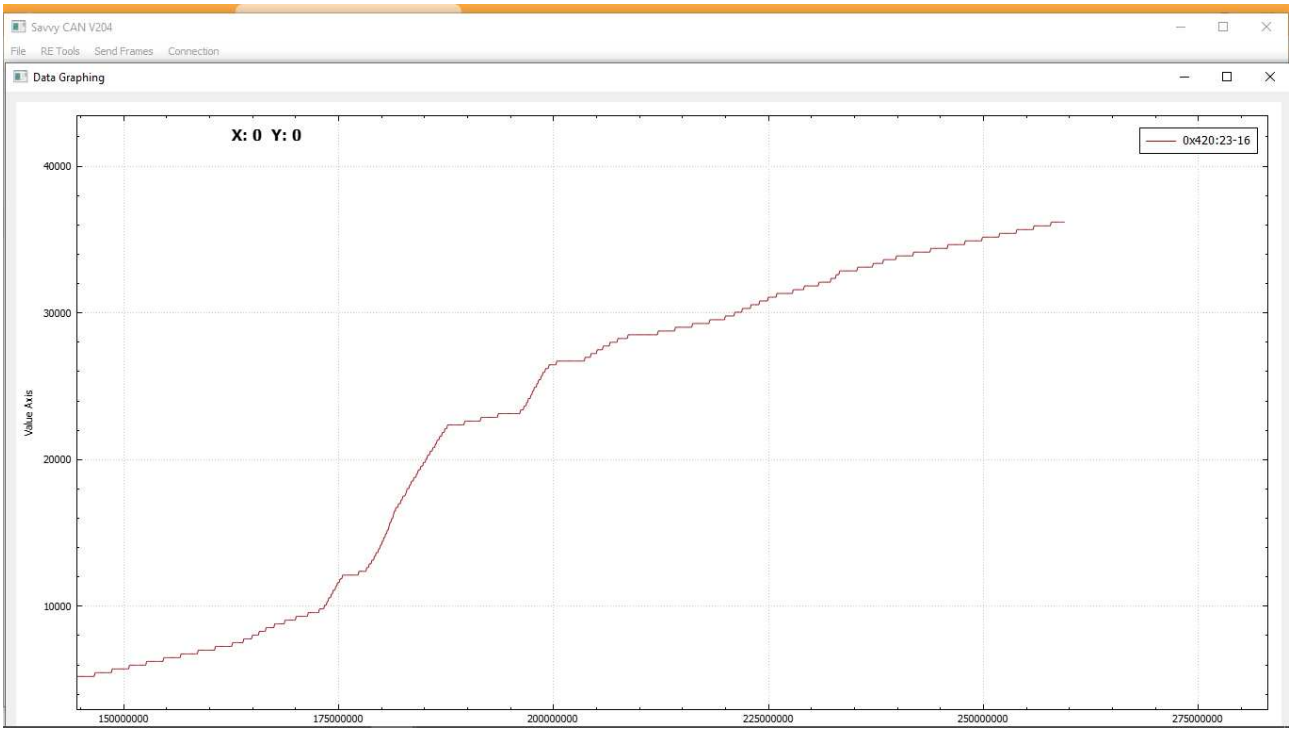



Figura 24.

1) **Graphic window ID: 0x420**

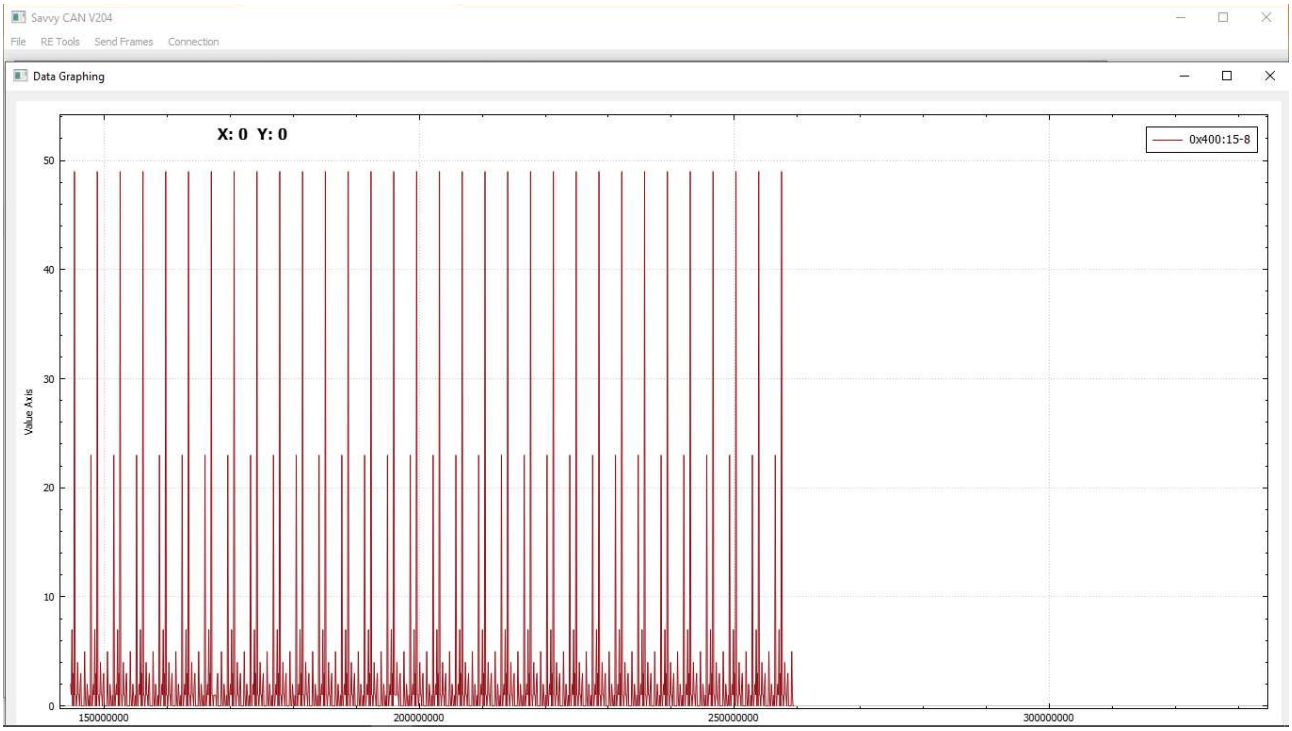



Figura 25.

2) **Graphic window ID: 0x400 dal bit 15 per un byte. ( da notare il grafico uguale al caso della macchina ferma ).**

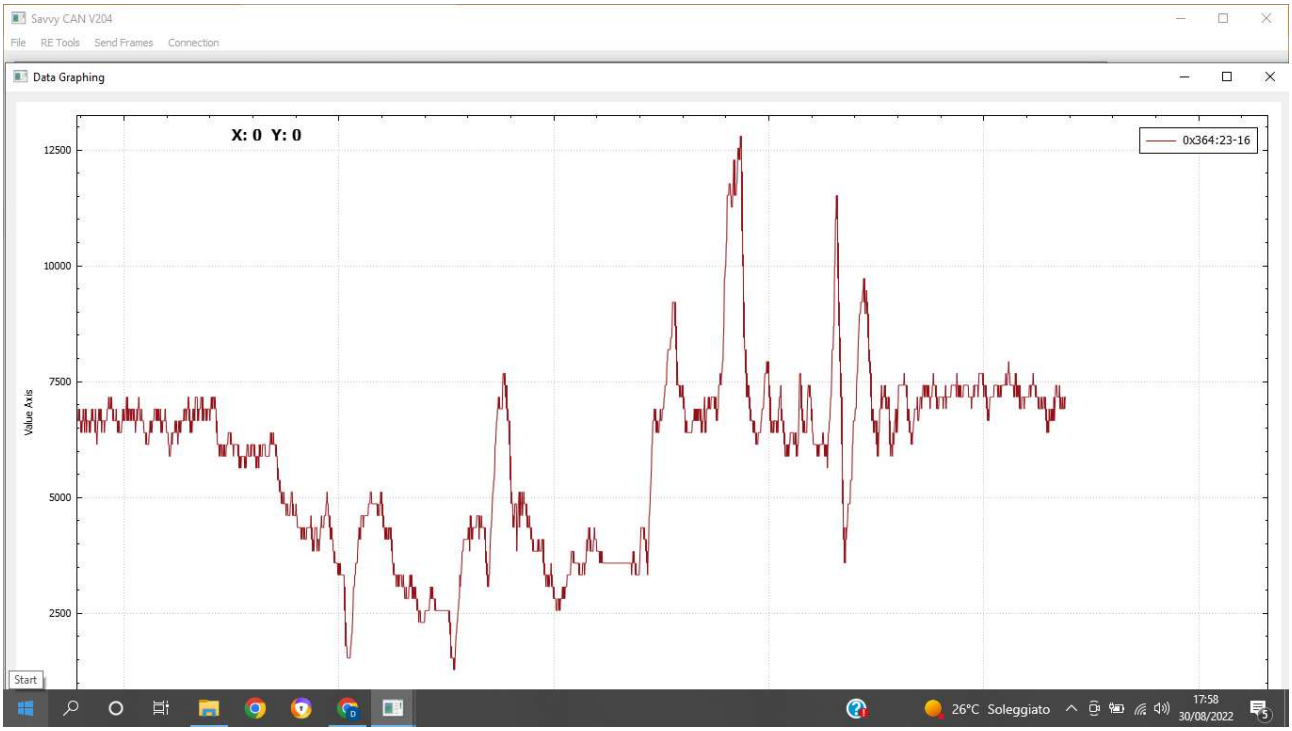



Figura 26.

**3) Graphic window ID: 0x364 dal bit 23 per due byte.**

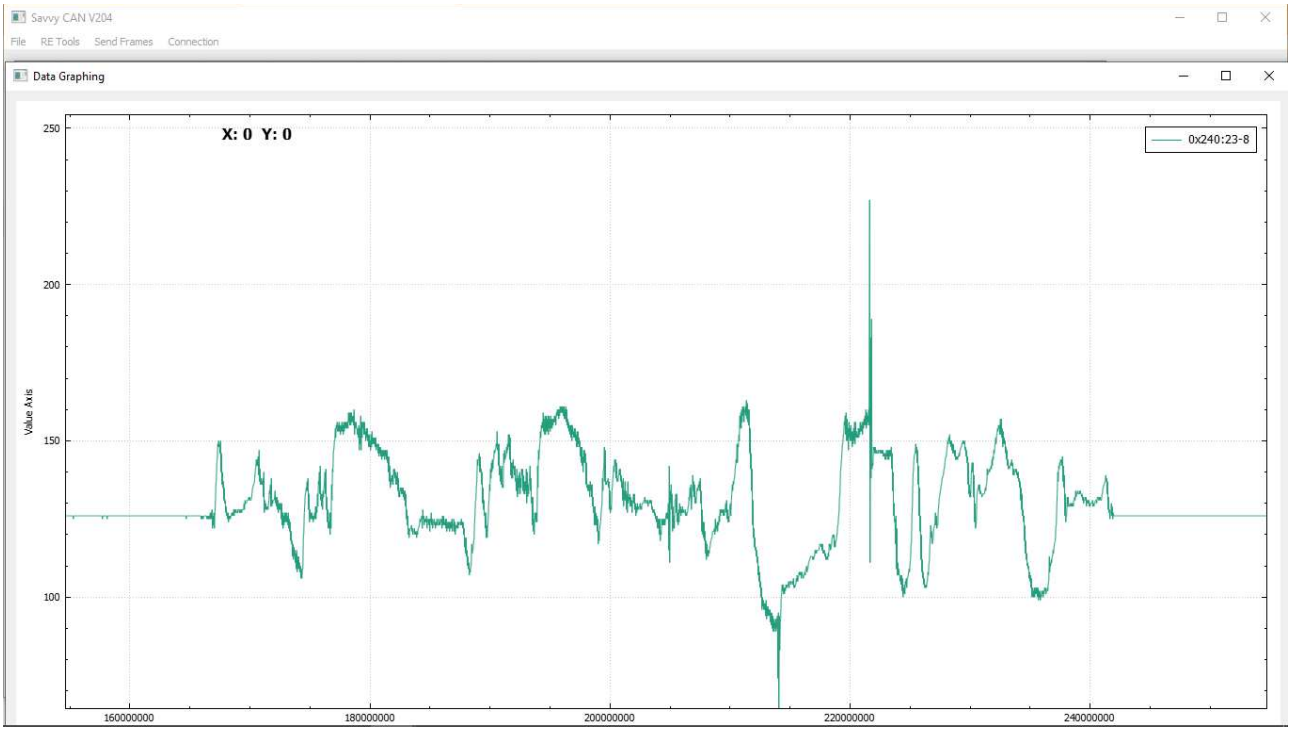



Figura 27.

**4) Graphic window ID: 0x240 dal bit 23 per un byte.**

- ## Conclusione ed altre applicazioni:

Eccoci giunti alla conclusione. Considero l'esperienza svolta molto formativa, sia come primo approccio aziendale e quindi lavorativo, ma anche come esperienza accademica in quanto ho dovuto studiare protocolli e sistemi di comunicazione a me ignoti, comprendere datasheet e utilizzare software . Riuscire inoltre a rendere funzionante il sistema di comunicazione mi ha dato molta soddisfazione e mi ha fatto capire quanto sia importante la meticolosità e la tranquillità nel lavoro. Tranquillità che è caratteristica del luogo in cui ho lavorato e le persone con le quali l'ho condiviso; ci tengo quindi a ringraziare sinceramente l'azienda SensorID s.r.l, che mi ha ospitato, perché mi ha messo nelle migliori condizioni possibili per svolgere il mio operato.

Purtroppo però, mi rammarica non aver avuto abbastanza tempo per poter completare in pieno la seconda commissione che mi è stata assegnata e magari, chissà, riuscire ad andare oltre con il lavoro in questione che mi è sembrato molto interessante.

Interessante è stato l'ambito lavorativo, che ha grandissimo margine di sviluppo soprattutto nell'ambito **industriale, automotive, nel settore ferroviario, della domotica etc.** Insomma tutto ciò su cui può essere attuata un'automatizzazione, viene controllato da centraline elettroniche.

Ormai imprescindibili anche per la regolazione e l'ottimizzazione di impianti energetici di tipo **eolico, idrico o solare, etc**. Sicuramente uno degli ambiti più in voga negli ultimi tempi date le problematiche legate all'inquinamento per la produzione di energia e i costi che questo comporta. Centraline, ad esempio nell'ambito di impianti solari, in grado di leggere i dati prelevati da sensori di temperatura direttamente collegati a i pannelli. Se la temperatura dei pannelli supera quella del serbatoio interno verrà azionata una pompa di calore (ad aria oppure ad acqua, dipende dall'impianto) pilotata dalla centralina. La stessa centralina che, nell'ipotesi di inattività della pompa per più di un certo periodo, la aziona per mantenerla in funzione.

Ovviamente il modus operandi rimane molto simile a quello studiato in dettaglio per l'ambito automotive: la linea di comunicazione rimane quella CAN-BUS nonostante possa presentarsi l'ipotesi di una comunicazione tramite rete internet su linea ETHERNET.

Grandi applicazioni vengono presentante anche nell'ambito **chimico**: ad esempio nei processi di trasformazione dei composti chimici in cui sistemi elettronici monitorano la velocità di reazione, il grado di omogeneità, la concentrazione dei soluti e dei solventi, etc.

Insomma un mondo molto vario e vasto che non può essere totalmente studiato all'interno di un elaborato di tesi. Nonostante questo però l'averlo approcciato mi ha fatto capire quanto si fondamentale comprenderne il funzionamento essendo componenti che caratterizzano la quotidianità.

Per casualità sono parte di un'azienda che si occupa proprio di questi sistemi di controllo e di comunicazione in ambito ferroviario; un lavoro che mi sta valorizzando e per il quale sto dando molto di me stesso per migliorarmi giorno dopo giorno.

Concludo quindi dicendo che la scelta di un tirocinio aziendale apre assolutamente la mente ed amplifica le tue conoscenze in modo metterle in pratica, approcciando il tirocinante ad uno scenario differente da quello accademico, ma strettamente correlato. Consiglio a tutti di avere un'esperienza del genere.

- **BIBLIOGRAFIA:**


- **www.ecutesting.it ;**
- **www.rubinolab.com;**
- **CLX000 MANUAL FW_5.8 (datasheet componente simulatore 2000);**
- **www.Bosch.it ;**
- **www.wireshark.org ;**
- **https://SavvyCan.com ;**

# DEDICHE :