Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

Meccanica e scienze avanzate dell'ingegneria

Ciclo XXIX

Settore Concorsuale di afferenza: 09-C1

Settore Scientifico disciplinare: ING-IND/08

Sviluppo di un framework per l'automazione dei test al banco e la prototipazione di algoritmi di controllo di motori a combustione interna

Presentata da:

Ing. Marco Cangini

Coordinatore Dottorato

Prof. Nicolò Cavina

Relatore

Prof. Enrico Corti

Esame finale anno 2017

Alla Mia Famiglia.

A chi lo è sempre stato…
…e a chi lo è diventato

Grazie

Abstract.

In recent years, increasingly stringent limits on polluting exhaust gas emissions, have led to a considerable increase in the complexity of internal combustion engines. This complication determines an exponential increase in the number of tests to be carried out in the testing room. The typical methods of test management can no longer be used, but it is essential to create a system that optimizes the tests. To drastically reduce the execution time, it is necessary to implement an architecture able to facilitate the exchange of data between systems present in the testing room, and, in addition, to define the test automation strategies. The approach to certain innovative methods still represents a complicated task in many groups of the powertrain control strategies development, although, once developed, leads to great benefits during the testing phase. The work illustrates the methods implemented for the management of these strategies. First it is described the approach used in the calibration of spark advance to maintain acceptable levels of knock during the calibration process. Following is shown the test automation system allowing full control of the engine operating point and management of the acquisition by checking the stability of the achieved conditions. The last part shows rapid prototyping systems for the control of innovative engine components.

Indice

Indice delle figure

Introduzione

L'ottimizzazione delle prestazioni dei motopropulsori endotermici alternativi manifesta una tendenza verso una sempre maggiore complessità. I limiti più stringenti sulle emissioni inquinanti e sulla produzione di CO2, hanno portato a una maggiore complessità delle strategie di controllo, rendono lungo e costoso il processo di calibrazione del sistema di controllo del motore. Tipicamente la sua messa a punto prevede una serie di test specifici al banco prova, seguiti da una fase di trattamento dei dati, per sintetizzare gli algoritmi di controllo appropriati. Oltre a questo aspetto c'è da valutare anche il costante aumento del numero di dispositivi elettrici, al fine di massimizzare l'efficienza globale del propulsore. Si tratta quindi di una serie di dispositivi da testare, proteggere, verificare, controllare che portano a un aumento considerevole delle leve di controllo. La conseguenza immediata di tali cambiamenti è la necessità di identificare il comportamento di un maggior numero di strumenti strettamente interconnessi e questo porta a un inevitabile aumento del numero di test sperimentali per la corretta calibrazione del motopropulsore. Di fatto, il sistema sala prove nell'ultimo decennio, ha visto moltiplicato il numero di misure e di conseguenza il numero di strumenti utilizzati e di fatto diventa sempre più importante il processo di comunicazione di dati fra strumenti diversi. Oltre a questo, assume una crescente rilevanza l'automatizzazione delle prove motore, al fine di ottimizzare il numero di test e velocizzare la durata di ogni singolo procedura di acquisizione dati. La strumentazione tipica di una sala prove, tuttavia, non si presta facilmente a un approccio di questo tipo. Essa è infatti costituita da dispositivi eterogenei (sistema di controllo banco, strumenti di gestione parametri centralina, strumenti di analisi combustione, misuratori di particelle, ecc), ciascuno specificamente progettato per svolgere una gamma limitata di funzioni e risulta difficile sfruttare in modo organico le informazioni prodotte da ciascuno strumento. Le considerazioni fatte hanno portato negli ultimi anni allo sviluppo di protocolli di comunicazione veloci che permettono l'interazione fra i vari strumenti e la condivisione di dati. L'approccio a certe metodologie innovative rappresenta ancora, per molti gruppi di sviluppo di strategie di controllo del motopropulsore, aspetti problematici, anche se, una volta sviluppate, portano a grandi benefici durante la fase di sperimentazione sui MCI. L'attività di dottorato di questi tre anni si pone in questo contesto. Infatti con la collaborazione di Alma Automotive Srl e dei gruppi Powertrain e R&D del gruppo Maserati Alfa Romeo si sono sviluppate metodologie di gestione di

queste strategie e si è operato allo sviluppo di sistemi che possano essere di supporto alla calibrazione, alla sperimentazione e alla prototipazione. Le attività svolte, prevedono lo sviluppo di un collettore dei dati provenienti da tutti questi dispositivi e strumenti, in cui sono implementate strategie di controllo complesse al fine di ottimizzare le prestazioni e verificare layout motore particolari direttamente in sala prove.

Il primo capitolo da una panoramica dei sistemi tipici presenti in sala prove e delle metodologie utilizzate nell'ambito della sperimentazione motore. Il capitolo due illustra le scelte fatte in fase di definizione degli hardware necessari per l'acquisizione di segnali motore e lo sviluppo e l'esecuzione di algoritmi di controllo per la gestione di strategie da eseguire Real Time (RT), compatibilmente con la frequenza di ciclo motore. I capitoli tre e quattro definiscono le funzioni dei principali strumenti presenti in una sala prove e le metodologie sviluppate per la loro gestione e per l'interazione coi sistemi sviluppati.

Nei capitoli successivi, dal quinto al settimo, vengono descritte le applicazioni sviluppate utilizzando le metodologie descritte in precedenza e gli algoritmi di controllo implementati.

Nel quinto capitolo viene illustrato il metodo utilizzato per l'automatizzazione di processi di calibrazione dell'anticipo di accensione, permettendone la calibrazione con l'ausilio di sistemi in grado di monitorare il fenomeno della detonazione e in grado di intervenire prontamente in caso di superamento di sogli definite pericolose di tale fenomeno, al fine di mantenere in sicurezza il motore. Per i moderni controlli è infatti richiesto lo sviluppo di strategie in grado di variare in tempo reale il valore di anticipo con algoritmi che prevedono un'azione legata al valore di detonazione rilevato, prevedendo inoltre di rientrare dell'anticipo decurtato, al fine di mantenere il motore su soglie di detonazione considerate tollerabili.

Le innumerevoli leve di controllo presenti nei moderni MCI, richiedono un sempre maggiore numero di test e risulta quindi fondamentale, lo sviluppo di un sistema in grado di automatizzare i test in sala prove e in grado di interfacciarsi con sistemi di ottimizzazione del numero di test da effettuare (*DOE, design of experiment*). Nel capitolo sei verrà mostrata l'architettura per l'implementazione di questi sistemi che si basano sullo sviluppo di protocolli di comunicazione veloci, in grado di gestire tutti gli strumenti presenti in sala prove. In questo modo possiamo controllare il motore e i suoi punti di funzionamento, regolando in maniera precisa gli attuatori presenti, al fine di eseguire test in condizioni di prova stabili.

Nel capitolo sette, sono descritte le metodologie utilizzate per la gestione di componenti motore innovativi e layout motore complessi. È quindi possibile eseguire modelli per la gestione di prototipi e la verifica di essi direttamente sul sistema fisico, al fine di valutare gli effetti in particolare su consumi e emissioni, così da indirizzare le strategie di sviluppo dei propulsori. Con queste metodologie sono state sviluppate le basi per algoritmi di controllo di strumenti innovativi.

1 Sala prove motore

Una sala prova motore può prevedere l'utilizzo di molti sistemi (Hardware e Software) a seconda delle operazioni che si vogliono eseguire sul motore; fondamentalmente però sono tre i sistemi realmente necessari per poter effettuare lo sviluppo e la calibrazione del motore in maniera completa:

- sistemi per il controllo e la calibrazione della ECU (Engine Control Unit);
- sistemi per il controllo del banco prova (ovvero del freno, delle apparecchiature per la regolazione dei parametri di sala prove e dei sensori di sala);
- sistemi per l'analisi della combustione all'interno dei cilindri.

In particolare per le sale prova di Maserati, ma in generale nelle realtà più comuni presenti in ambito industriale, i software più comunemente utilizzati sono:

- INCA e i moduli ETAS per la funzione di controllo della centralina;
- AVL PUMA per la funzione di controllo banco;
- AVL INDICOM (con l'HW AVL INDISET) per l'analisi della combustione.

Una struttura tipica per le celle è rappresentata in *figura 1.1*, in cui si riconoscono gli strumenti appena citati. Per permettere la comunicazione tra questi tre sistemi (e quindi tra i rispettivi HW), AVL fornisce un ulteriore programma, CAMEO, che permette di unire tutti i dati ricevuti, essendo interfacciabile con questi e molti altri strumenti attraverso l'istallazione di librerie appositamente create da AVL. Il maggiore vantaggio di CAMEO è la sua capacità di collegarsi direttamente con PUMA in modo flessibile (PUMA usa un linguaggio proprietario per la comunicazione con SW esterni, che chiaramente è noto agli sviluppatori di CAMEO); con esso si riescono a generare delle prove automatizzate e ad avere il controllo (regime e carico prima di tutto) del banco.

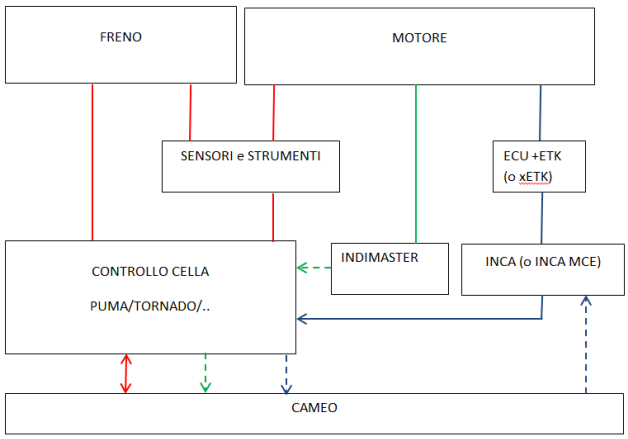


Figura 1.1: Struttura di sala con l'utilizzo di AVL CAMEO.

1.1 Tool automazione e controllo (TAC)

Il principale svantaggio di questo approccio però è che la comunicazione tra i vari strumenti non è Real Time (RT) e questo può, in certe condizioni, mettere in pericolo il motore e la cella. L'esempio seguente permette meglio di capire cosa si intende per RT. In caso di detonazione in camera di combustione, utilizzando CAMEO e la struttura di *figura 1.1*, l'informazione di knock rilevata da INDISET viene trasmessa al PUMA, che a sua volta la trasmette a CAMEO, il quale è in grado di agire sulla ECU. Questa struttura impiega alcuni secondi ad essere eseguita, un intervallo temporale sicuramente non accettabile e che fa in modo che il motore compia diversi cicli di combustione lontano da condizioni di sicurezza. Inoltre, le librerie per interfacciare CAMEO con i vari strumenti, soprattutto quelli non forniti da AVL, sono poco personalizzabili, non sempre disponibili e abbastanza costose. Altro aspetto fondamentale, CAMEO funge semplicemente da programma di automazione e al suo interno, non possono essere implementate logiche per la ricerca e l'ottimizzazione di strategie innovative di sperimentazione motore. Da qui nascono le motivazioni che hanno spinto il management Maserati Alfa Romeo ad optare per una soluzione, sviluppata ad hoc in collaborazione con Alma-Automotive, che presentasse le seguenti caratteristiche:

- comunicazione RT tra tutti gli strumenti, o almeno tra quelli che effettuano il controllo del motore provvedendo a mantenerlo sempre in sicurezza;
- possibilità di sviluppo di un sistema di automazione delle prove attraverso l'implementazione di strategie in grado di variare i parametri di centralina in funzione di altre variabili,
- capacità di raccogliere informazioni da tutti gli strumenti di sala, con la possibilità di aggiungerne di nuovi qualora sia necessario.

La scelta di rivolgersi ad un gruppo di ricerca è motivata non solo dalla necessità di avere un sistema di automazione con caratteristiche innovative in termini di velocità di intervento e gestione delle prove, ma anche dalla prospettiva di un sistema più sofisticato di sostegno e sviluppo della calibrazione. Le caratteristiche richieste a questo sistema permettono un campo di applicazione vasto, con diversi vantaggi per tutto il processo di calibrazione: infatti esso permette una riduzione della necessità di intervento dell'operatore in sala (con conseguente risparmio di tempo), può essere utilizzato per sviluppare e uniformare le metodologie di delibera tecnica del motore (per esempio

calibrazioni di controllo lambda, pressione rail o “spazzolate” di anticipo, piani quotati e cicli di emissione), oltre ovviamente ad essere flessibile e personalizzabile secondo le richieste di innovazione che lo sviluppo dei motori richiede. A queste richieste va aggiunta anche la possibilità di testare architetture motore particolari, per mezzo di sistemi di “Rapid Control Prototyping” (RCP) al fine di valutare preventivamente e in maniera rapida gli effetti di una soluzione motore innovativa.

Quindi, il TAC, a valle di queste considerazioni, è da considerarsi come un collettore di informazioni in grado di intervenire con algoritmi e strategie di controllo motore definite per l’ottimizzazione delle prestazioni del motore e per l’efficientamento della sperimentazione. In questo elaborato, definiremo come TAC il sistema presente in sala prove e utilizzato per le diverse tipologie descritte in precedenza e come RCP i sistemi di test implementati in vettura.

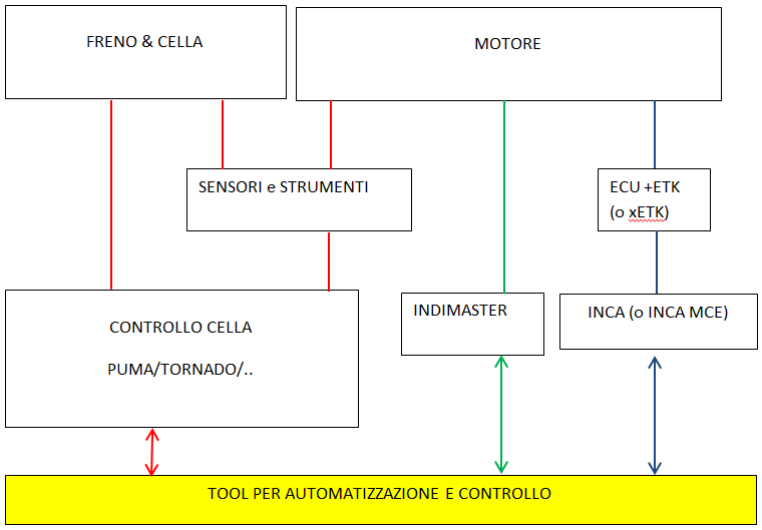


Figura 1.2: struttura di interconnessione degli strumenti di sala prova utilizzando questo tool per l’automazione ed il controllo (TAC).

2 Piattaforma HW e SW

Prima di affrontare nel dettaglio gli algoritmi e le metodologie utilizzate per lo sviluppo degli strumenti trattati è stato ritenuto opportuno soffermarsi scelta dell'architettura hardware software. Questo approccio permette di sviluppare gli algoritmi avendo già chiaro ciò che è possibile fare e ciò che non è consentito per via di vincoli strutturali dipendenti dall'hardware in modo di ottimizzare gli algoritmi per una specifica struttura di calcolo e ottenere da subito un buon livello di prestazioni, anziché ragionare in termini generali e dover poi modificare profondamente il codice per garantire i requisiti sulla piattaforma scelta. Queste considerazioni valgono principalmente per l'hardware, tuttavia, dato che ogni sistema di calcolo dispone di strumenti di sviluppo dedicati, si riflette anche sull'ambiente software.

2.1 Requisiti

I requisiti principali che l'hardware e il software debbono garantire per questo tipo di sistemi, sono i seguenti:

1. capacità di eseguire calcoli complessi a frequenze compatibili con quelle di ciclo motore;
2. visualizzare su un'interfaccia operatore tutte le grandezze di interesse;
3. garantire l'esecuzione in tempo reale e il calcolo delle grandezze entro il ciclo motore successivo a quello di calcolo;
4. essere modulare ed espandibile;
5. avere diversi fattori di forma per potersi meglio adattare ai diversi ambiti operativi, per esempio essere imbarcabile a bordo veicolo;
6. essere aggiornabile con l'evolversi della tecnologia mantenendo la compatibilità col software preesistente;
7. consentire il riutilizzo di codice già sviluppato in linguaggi di programmazione diversi, per esempio Matlab, Simulink e C;
8. avere costi contenuti, sia come hardware che come strumenti di sviluppo;
9. possibilità di interfacciamento con tutti quei bus di comunicazione tipici dell'ambito automotive (linee CAN, seriali, ethernet);
10. non richiedere conoscenze di programmazione di basso livello;
11. avere librerie già precompilate per l'esecuzione di protocolli base.

I requisiti sopracitati sono di carattere generale ma sono comunque difficili da soddisfare tutti contemporaneamente. Tuttavia, la conoscenza pregressa dei sistemi di acquisizione dati e controllo di National Instruments ha facilitato la ricerca. I requisiti più difficili da realizzare sono il primo e il quarto, ovvero di poter manipolare dati provenienti dai diversi "interlocutori" presenti in sala prove e di produrre risultati in tempo utile per essere utilizzate, eventualmente, come retroazione per il sistema di controllo motore.

La tecnologia applicata ai semiconduttori ha portato allo sviluppo di dispositivi programmabili, gli FPGA, acronimo di Field Programmable Gate Array. Questi circuiti integrati si differenziano da quelli comunemente utilizzati perché permettono l'esecuzione di algoritmi sviluppati dall'utente che vengono implementati in hardware, anziché in software [1]. Solitamente un microprocessore dispone di una unità di calcolo e di una serie di istruzioni che può eseguire. Queste istruzioni sono eseguite sequenzialmente e ogni unità di calcolo può eseguire tipicamente una sola istruzione per volta. Qualora sia necessario eseguire calcoli paralleli un sistema operativo RT si occupa di rendere seriali tali operazioni e di garantire i requisiti temporali imposti, in modo trasparente all'utente. Un chip FPGA non dispone di unità di calcolo e di un set limitato di istruzioni che può svolgere, ma è costituito da milioni di celle elementari capaci di implementare semplici logiche combinatorie. Il modo in cui vengono interconnesse le unità elementari definisce l'algoritmo implementato. Questo elimina il vincolo di esecuzione seriale su un'unica unità di calcolo, ma permette l'esecuzione parallela e distribuita dei calcoli, garantendo che ogni blocco di codice non interferirà sugli altri. Questo rappresenta un enorme vantaggio e garanzia nell'implementazione di questi sistemi, che devono rispettare determinati requisiti temporali. Il problema di questa soluzione è ancora una volta il linguaggio di programmazione, il VHDL (Very High speed integrated circuits Hardware Description Language), che è di basso livello e di difficile apprendimento. National Instruments fornisce una serie di schede di acquisizione dei dati basate su FPGA, compatibili con il linguaggio di programmazione LabView, un ambiente di sviluppo grafico ad alto livello che non richiede conoscenze specifiche di linguaggi di programmazione testuali. La disponibilità di tale classe di schede è stata determinante nella scelta di adottare hardware e software prodotti da National Instruments.

Per assolvere al meglio tutti i compiti ai vari livelli, è quindi possibile definire il layout via via più conforme con le caratteristiche di ogni applicazione, composta da sistemi hardware specifici. Come anticipato, al più basso livello è presente hardware basato su FPGA, che si occupa di elaborare i dati man mano che vengono acquisiti. Il livello

intermedio, che deve produrre algoritmi complessi deterministicamente, ma con limiti temporali meno stringenti, è affidato ad hardware RT, ovvero composto da un microprocessore unito ad un sistema operativo certificato per esecuzione in tempo reale in grado di garantire il completamento dei calcoli entro determinate finestre temporali [2]. L'ultimo livello, il più alto, dedicato all'elaborazione sulle statistiche e alla visualizzazione dei risultati in modo grafico, ha necessità di svolgere una notevole mole di calcoli, senza però avere scadenze temporali stringenti. Può quindi essere affidato ad un normale personal computer, dotato di processori di ultima generazione e sistema operativo Microsoft Windows.

Nello schema in *figura 2.1* è visibile un altro livello, relativo all'interfacciamento con il mondo esterno. Esso è connesso sia alla parte FPGA che alla parte RT, ciò riflette la possibilità da parte dei diversi livelli di accedere alle risorse di acquisizione e generazione dati. Il livello più alto non è fisicamente connesso con alcun segnale proveniente dal sistema sotto osservazione, ma comunica attraverso un'interfaccia Fast Ethernet con il processore RT scambiando tutte le informazioni di cui necessita.

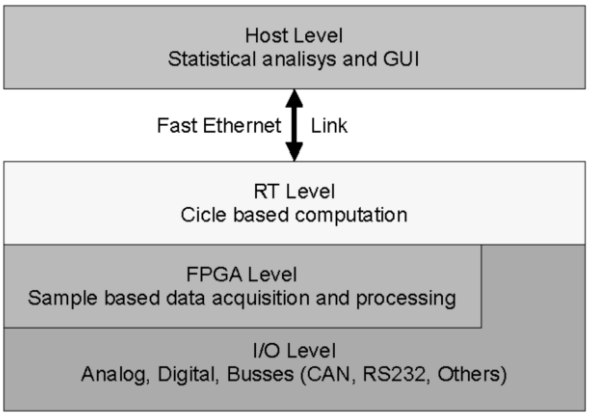


Figura 2.1: Schema dell'architettura hardware

Conoscere le potenzialità, in termini di frequenze di acquisizione, tempi di calcolo e gestione dei task e degli algoritmi programmati, pone la base per la scelta delle soluzioni da applicare in merito alle singole funzioni da sviluppare. Un requisito fondamentale e l'ambiente in cui una tale applicazione deve svolgere le funzioni per cui è stata sviluppata. Da qui l'esigenza di un approfondimento sulla forma e modularità dell'HW.

2.1.1 Modularità e fattori di forma

I requisiti 5, 6 e 7 sottolineano l'importanza di aspetti come la modularità, l'integrabilità e l'espandibilità del sistema in funzione delle diverse applicazioni. Si è cercato di

utilizzare una classe di dispositivi, caratterizzati da prestazioni diverse, capaci di eseguire gli stessi algoritmi software senza necessità di adattamenti particolari.

I livelli FPGA e RT hanno il compito di eseguire i calcoli a frequenze elevate, per poter eventualmente utilizzare tali informazioni per scopi di controllo. Il compito dell'interfaccia utente ha finalità molto diverse. Essa deve visualizzare tutti i dati raccolti, sintetizzarli tramite analisi statistiche e memorizzarli su un supporto di memorizzazione per successive rielaborazioni che possono essere eseguite con frequenza più bassa: l'importante è che tutti i dati vengano elaborati nel tempo a disposizione, senza accumulare coda. Per questo non sono imposti requisiti temporali stringenti, ma è richiesta una grande capacità di calcolo per processare la mole di dati generata dai livelli sottostanti. Privilegiando quindi la potenza di calcolo, la scelta della piattaforma hardware da dedicare al compito di interfaccia utente ricade su computer fissi o portatili di ultima generazione, dotati di processori multicore ad elevata frequenza e con ampia capacità di memorizzazione dati. La connessione con il livello RT avviene tramite interfaccia Fast Ethernet, comunemente presente in tutti i computer.

L'ultimo requisito citato all'inizio del capitolo, il numero 10, è forse quello più importante ai fini della possibile realizzazione del presente progetto. È infatti necessario potersi concentrare sullo sviluppo degli algoritmi senza doversi preoccupare troppo degli aspetti tecnici della scrittura del codice. Questo sarebbe impossibile usando strumenti di sviluppo tradizionali, come i linguaggi di programmazione testuali a basso e medio livello, come il “C”. È altresì importante poter lavorare in un ambiente di sviluppo comune, in grado di gestire i livelli FPGA, RT e Host allo stesso modo, piuttosto che dover apprendere l'uso di strumenti di sviluppo dedicati. Fortunatamente l'ambiente LabView soddisfa in gran parte tali requisiti, rendendo possibile la programmazione di alto livello di tutti i componenti hardware, con la stessa sintassi, seppur con alcune limitazioni.

2.1.2 Hardware utilizzati

Per quanto riguarda la scelta degli HW da implementare, possiamo considerare due aree differenti:

- Le applicazioni RCP, che prevedono l’acquisizione di segnali analogici a frequenze elevate. Per queste applicazioni sono state sviluppate, come descriveremo in seguito, delle parti di codice che prevedono l’acquisizione di segnali e la generazione di output “fasati” con il motore. Da qui l’importanza della parte FPGA, per

l'acquisizione di segnali ad altre frequenze. Per queste applicazioni sono state utilizzate due diverse tipologie di Hardware:

1. Un sistema CompactRIO (cRIO) dispone di un processore RT con sistema operativo certificato (VxWorks o recentemente Zynq®-7000[1]). L'unità di calcolo è di classe PowerPC con frequenze di clock fino a 667 MHz. Il processore RT comunica con i moduli di I/O tramite il chip FPGA presente nello chassis. A seconda del modello, possono essere presenti interfacce Ethernet, USB, RS232 e RS 485. Lo chassis in un sistema CompactRIO rende disponibili una serie di slot di espansione, ma include al suo interno un dispositivo FPGA. Tale dispositivo comunica con tutti i moduli di acquisizione e generazione dati e con il processore tramite una veloce interfaccia DMA (Direct Memory Access). Anche in questo caso, esso lavora alla frequenza di clock di 40MHz.
2. Miracle2, HW sviluppato da Alma Automotive su base National Instruments. Processore Xilinx Zynq®-7000 dual core con frequenze fino 667 MHz e sistema operativo NI Linux Real Time certificato, fino a 512MB DRAM. L'interfaccia di accesso all'hardware sviluppata da Alma Automotive [3] prevede:

Analog Input	24 channels (2 high voltage channels ±40V), 16bit, ±10V differential input range, up to 400ksps, simultaneous sampling, with antialiasing filter (100kHz) ±0.1% accuracy
Analog Output	400ksps, simultaneous sampling, with antialiasing filter (100kHz) ±0.1% accuracy
Digital Input (accept up to 25 V)	8 @ 5V, 500 kHz
Digital Input (protected up to 25 V)	8 @ 5V, 10MHz
Digital Output (protected up to 25 V)	16 @ 5V, 10MHz

Figura 2.2: Tabella riassuntiva input/output Miracle 2

- Per le altre applicazioni che non prevedono l'acquisizione di segnali ad alta frequenza, ma lo scambio di dati attraverso protocolli di comunicazione, la parte più rilevante è la potenza di calcolo e il determinismo temporale delle operazioni. Infatti a queste applicazioni, si richiede la capacità di eseguire algoritmi complessi ed elaborati a frequenze elevate, ma non è richiesta l'acquisizione e l'elaborazione di segnali a frequenze tipiche di un sistema FPGA. A tal proposito sono stati utilizzati PC standard

[1] La famiglia Zynq®-7000 [17] è basata sull'architettura Xilinx programmabile SoC. Questi prodotti integrano un sistema dual-core o single-core ARM® Cortex™-A9 processing system (PS) e 28 nm Xilinx programmable logic (PL) in un unico dispositivo. Lo Zynq®-7000 integra la programmabilità software di un processore base ARM® con HW FPGA, eseguendo processi chiave e al contempo, integrando CPU, DSP, ASSP, e funzionalità miste su un singolo dispositivo.

con elevate potenze di calcolo convertiti mediante sistema operativo LabView Real Time di National Instruments in HW compatibili con il software di programmazione.

2.1.3 Caratteristiche software

In contrasto con i linguaggi di programmazione testuali, LabVIEW sfrutta una programmazione grafica ad icone basata sul flusso dei dati, che permette di realizzare elaborati programmi di analisi o di controllo senza scrivere fisicamente alcuna riga di programma. Infatti i programmi realizzati con LabVIEW sono detti Virtual Instrument (VI) in quanto nell' aspetto fisico e nel modo di interagire riproducono strumenti reali come oscilloscopi e multimetri mentre si tratta di oggetti virtuali. L'applicazione è costituita da icone collegate tra loro da "cavi" virtuali che trasportano le informazioni che assumono una forma e un colore differente a seconda del tipo di dato trasportato (numero, stringa di testo, matrice, booleano, etc.). Da notare che a differenza degli elaborati scritti secondo un codice testuale tipici dei linguaggi di programmazione tradizionali, la programmazione grafica di LV semplifica enormemente l'apprendimento e la scrittura dei programmi da parte dell'utente; inoltre qui è il flusso dei dati a determinare l'ordine di esecuzione dei vari blocchi nel programma, e non una sequenza di istruzioni.

Un blocco di elaborazione non è altro che la rappresentazione di una serie di operazioni che vengono eseguite sui dati in ingresso a seconda della funzione del blocco stesso: può ad esempio essere una semplice operazione matematica o di un intero programma a sé stante, poiché, come nei linguaggi di programmazione tradizionali, un programma può richiamare una sub-routine. L'interfaccia con l'utente di ogni VI è il Front Panel in cui sono posizionati controlli e indicatori che rappresentano rispettivamente gli input e gli output interattivi del VI. I controlli sono manopole, potenziometri, quadranti, pulsanti, interruttori, comandi scorrevoli, caselle numeriche o di testo e altri meccanismi di introduzione di dati, mentre gli indicatori sono grafici, LED, tabelle e altri componenti che consentano di visualizzare gli output acquisiti o generati dal Block Diagram. Dopo aver realizzato il pannello di controllo, è necessario implementare le funzionalità richieste dall'applicazione nello schema a blocchi, unendo ed elaborando le icone che rappresentano gli oggetti del pannello frontale. Quest'ultima parte è il corpo centrale di un VI in quanto contiene il codice grafico, sotto forma di diagramma a blocchi, che gestisce gli oggetti presenti nel pannello di controllo. Nella realizzazione del codice si possono utilizzare, oltre ai capisaldi della programmazione classica come i cicli while, for e la struttura case, gli elementi messi a disposizione dalle librerie di Lab VIEW come

le strutture temporizzate (Timed Structures) che permettono di stabilire una priorità tra le varie parti del programma o funzione già implementate, le cosiddette Buit-in functions, per l'acquisizione, l'analisi e l'esposizione di dati. L'enorme flessibilità di LabVIEW risiede anche nella possibilità di stabilire una scala gerarchica e una priorità dei VI, consentendo questo modo di realizzare strutture anche molto complicate senza mai perdere di vista il quadro di insieme.

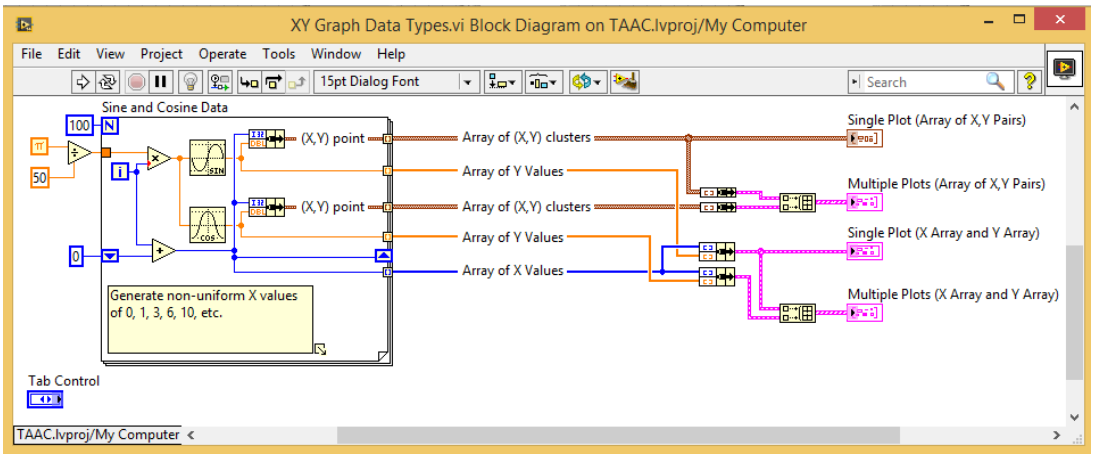


Figura 2.3: Esempio di un diagramma a blocchi un vi Labview

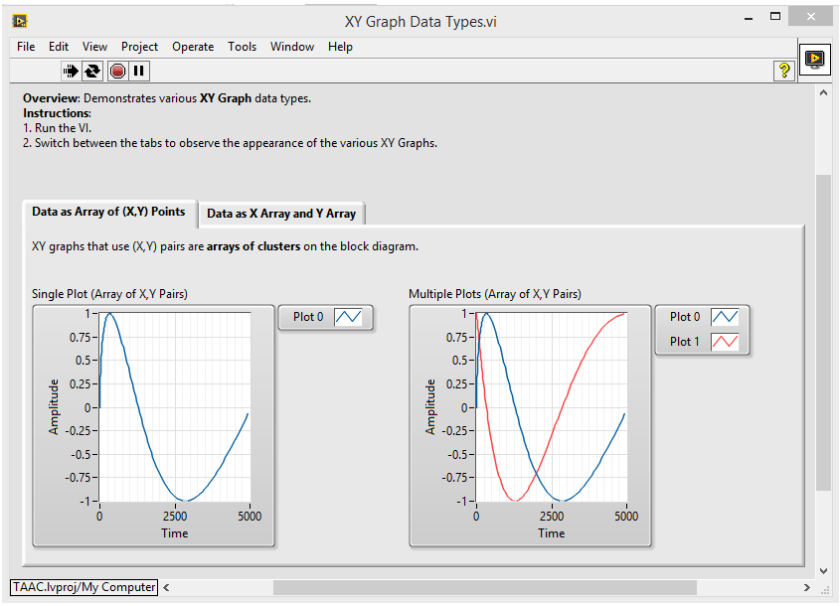


Figura 2.4: Pannello frontale del vi di figura 2.3.

3 Gestione banco e analisi combustione

PUMA OPEN è un'architettura HW-SW commerciale di AVL che permette il totale controllo della sala sia in modalità manuale sia in automatico. È una delle applicazioni più utilizzate in ambito industriale, nelle moderne sale prove motore. Al di là dei dettagli riguardanti gli strumenti di gestione della sala, ci soffermiamo sulla parte di gestione software delle varie istanze presenti in sala prove. Esso s'interfaccia con i sistemi Windows ed In Time, quest'ultimo è addetto alla gestione dei processi e delle analisi RT. Attraverso l'interfaccia PUMA Application Desktop è possibile interagire con tutto il sistema applicativo di PUMA. Nel Parameter Menager (PAM) sono fornite tutte le informazioni per l'interazione ed il controllo del freno, delle apparecchiature di sala e degli attuatori motore. Infatti, il PUMA, regolando mediante controllo della coppia resistente esercitata dal freno e agendo sull'attuatore della farfalla, regola il regime di rotazione del motore e la coppia da esso erogata. Tramite il PLC (sistema che permette la gestione della sala e dei limiti di allarme), il sistema in questione regola le temperature del liquido di raffreddamento, dell'olio e dell'aria aspirata dal motore. Le grandezze acquisite possono essere monitorate per evitare il possibile danneggiamento del motore, applicando delle reazioni predefinite, quali lo spegnimento motore al superamento dei limiti imposti dall'utente (monitoraggio dei limiti può essere sia continuo sia specifico della singola prova).

3.1 PUMA Parameter manager

Il PAM raccoglie tutti i diversi file di configurazione caricati all'interno del software per il corretto funzionamento del sistema, essi sono così suddivisi:

- System Parameter (*SIS*);
- Test Field Parameters (*STZ*);
- Unit Under Test Parameters (*UPR*);
- Test Parameters (*PRV*);
- Driver Parameters (*DRV*)

Il *SIS* contiene tutte le informazioni per la configurazione della sala prova, in altre parole: del freno, dei dispositivi di analisi, dei dispositivi di formula e dei canali temporanei (attraverso FEM).

I file *STZ* sono invece utilizzati per l'acquisizione dei parametri durante la misura, in pratica, contengono una o più liste di parametri (Data Storage Table, DST), che in fase di

misura possono essere acquisiti. In base al tipo di acquisizione scelto si possono utilizzare tre modalità differenti: automatica, semiautomatica o manuale. Per misurazioni con motore in regime stazionario l'acquisizione avviene tramite un certo numero di canali che garantiscono una frequenza di campionamento di 10 Hz, se invece si deve misurare un transitorio, si adopera un recorder che ha una frequenza di campionamento massima di 1 kHz. Per visualizzare tutti i dati acquisiti durante la registrazione, bisogna servirsi di un altro applicativo presente in PUMA Application Desktop, il PUC. Tramite quest'ultimo è possibile visualizzare in tempo reale la registrazione e creare un layout per una rapida post elaborazione e visualizzazione dei dati acquisiti. I parametri, che in Puma sono chiamati normname, possono essere catalogati, in funzione della specifica prova, in diverse chiavi (tipologie).

I file ***UPR*** contengono le informazioni specifiche del propulsore utilizzato in sala, dei parametri di controllo del motore e, in caso di test in dinamico, dei parametri caratteristici del veicolo e del sistema di trasmissione. La maggior parte delle prove svolte in sala prevede una variazione dei parametri di centralina durante lo svolgimento del test, in generale è possibile, durante l'esecuzione della prova, visualizzare in Real-Time grandezze di centralina grazie all'interfaccia ASAP3 (As Soon As Possible)[2], usata per l'interazione del Puma con il software dell'ECU (INCA). L'associazione dei parametri di centralina con quelli di sala (normname) è assegnata ai blocchi MEI, questi si presentano sotto forma di tabelle che, prima dell'esecuzione del test, dovranno essere impostate su "on line" con l'ECU.

Il File ***PRV*** contiene tutte le informazioni per creare un Ciclo Automatico, come la descrizione dei passi stazionari, delle tabelle di memorizzazione, dei dispositivi di misura ed in particolare i passi da eseguire in linguaggio Assembly. Nella versione di PUMA OPEN AVL presa in esame, è disponibile un linguaggio di programmazione (proprietario) grafico ad alto livello: Block Sequence Editor e Step Sequence Editor (BSQ/SSQ). La creazione di un test automatico avviene attraverso la programmazione in BSQ nell'interfaccia AVL EXPLORER. Il PC su cui generalmente è installato il software del Puma è dotato di due monitor su cui sono visualizzate in real time le grandezze analizzate (*figura 3.1*).

[2] Il protocollo ASAP3 e uno standard ASAM che verrà trattato nel capitolo riguardante l'interazione di sistemi con i sistemi di gestione e controllo della ECU

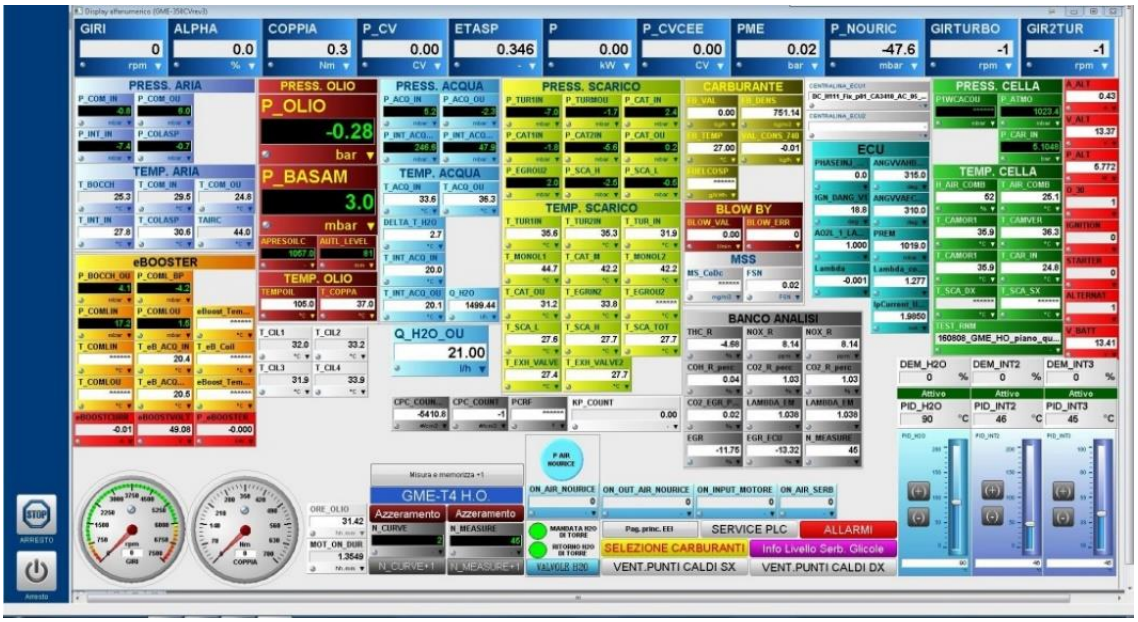


Figura 3.1: Interfaccia Operatore PUMA Open AVL

Lo sperimentatore può selezionare quali grandezze visualizzare, tramite indicatori di tipo digitale o con una strumentazione virtuale di tipo analogico, disponendo a proprio piacimento i vari indicatori. Inoltre, la natura grafica legata al linguaggio BSQ permette un facile monitoraggio del punto di esecuzione della prova da parte dello sperimentatore. Tutti i parametri possono essere monitorati per la messa in sicurezza del propulsore e dei dispositivi presenti in sala motore, questi sono divisi in due gruppi:

- Limiti assoluti del sistema (SAL) presenti nel blocco SIS (*figura 3.2*);
- Limiti assoluti (GWA) presenti nel blocco PRV.

I primi sono parametri analizzati in continuo e caratteristici della sala, i secondi sono dipendenti dalla prova. Per ogni allarme è possibile un'azione specifica:

- Stop motore;
- Minimo;
- Messaggio;
- Routine di eccezione.

L'ultimo punto elencato (Driver Parameters) permette di realizzare dei criteri di azione creati appositamente per il singolo limite.

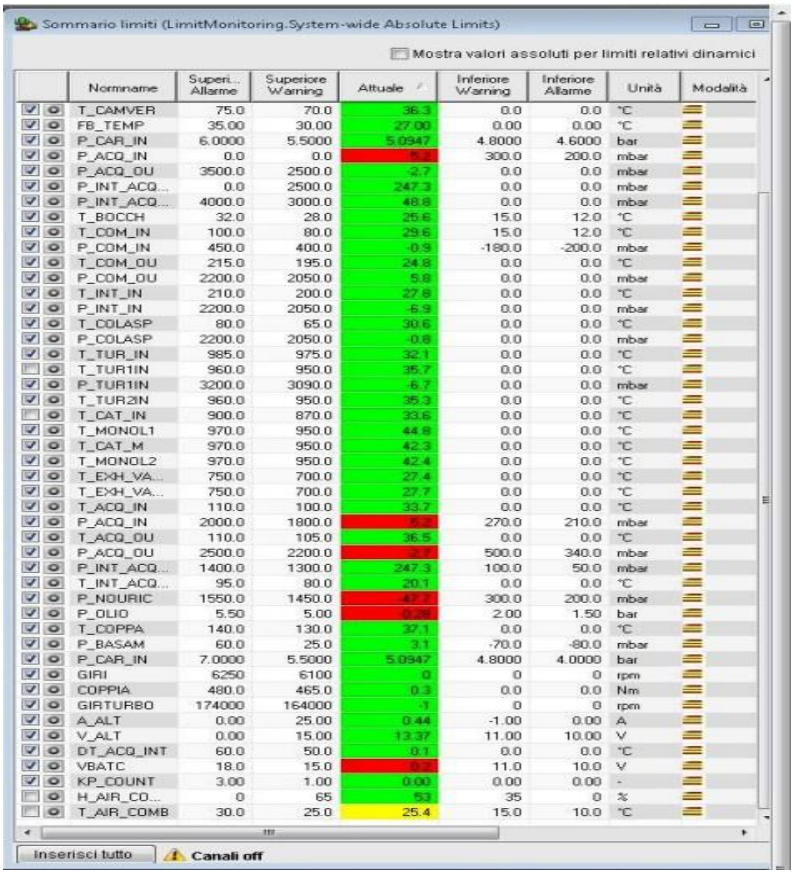


Figura 3.2: Sommario Limiti Sistema

Per dare origine e gestire i normname (parametri fondamentali di sala), si utilizza l'applicativo NED. I normame sono classificati in:

- grandezze misurate: tutti i parametri misurati da sala prova come, ad esempio, giri motore, coppia, pressione massima in camera di combustione;
- grandezze calcolate: parametri il cui valore non è direttamente misurato ma ottenuto attraverso una formula come ad esempio: potenza, pressione media effettiva;
- grandezze speciali: parametri di input dei test in automatico, le date, nomi, tolleranze grandezze.

3.2 Comunicazione con Puma Open AVL

AVL mette a disposizione, per poter gestire PUMA Open attraverso un SW di calibrazione automatica (ACS, Automatic Calibration System), l'interfaccia ASAM-ACI (Automatic Calibration Interface), sviluppata al fine di ridurre il tempo e il costo dei test in modalità manuale oltre che il rischio di errori da parte dell'operatore. L'interfaccia ASAM-ACI viene usata per collegare un sistema di calibrazione automatica (ACS) con l'AVL PUMA OPEN. Il protocollo è stato appositamente progettato per l'interazione

ottimizzata tra AVL CAMEO e PUMA OPEN per le applicazioni di calibrazione automatica dove il sistema di calibrazione deve poter accedere direttamente alle funzioni di PUMA al fine di subordinarle alle attività di automazione. Per il funzionamento dell'Interfaccia ASAM-ACI è necessaria l'installazione di IONA Orbix 3.0.1[3], applicabile per interfacciarsi solo con CAMEO.

La *figura 3.3* illustra il possibile schema di una procedura di calibrazione automatica. Quando PUMA OPEN AVL viene impostato su AUTOMATICO, ACS (ad esempio CAMEO) ne assumere il controllo ed esegue delle procedure definite dall'utente in speciali algoritmi di ottimizzazione per la calibrazione.

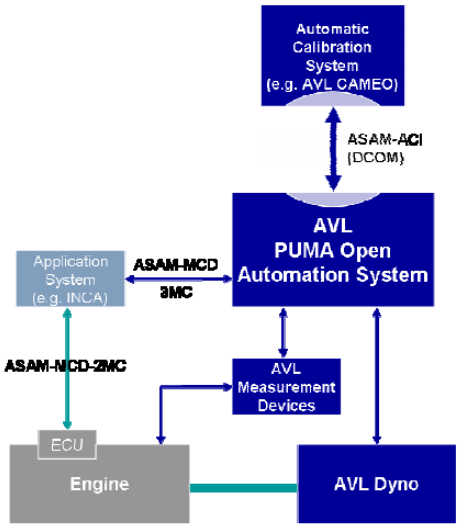


Figura 3.3: Struttura ACS

L'interfaccia ASAM ACI è definita "object-oriented" e si basa su una gerarchia client-server mettendo a disposizione 6 diversi tipi di service che devono ovviamente essere implementati sull'ACS:

- *controller Service*: trasmette a PUMA i valori da assegnare alle leve di controllo del banco disponibili;
- *recorder Service*: fornisce all'ACS i diversi valori misurati su PUMA, e anche elaborazioni di essi (medie, variazioni,ecc.);
- *player Service*: controlla i parametri di PUMA per variazioni a passo costante;
- *watcher Service*: permette un monitoring dei limiti impostati su PUMA;

[3] Orbix è un ORB CORBA (Object Request Broker), prodotto software da Micro Focus (originario di Iona Technologies), che aiuta i programmatori a creare applicazioni distribuite. Orbix è un'implementazione della specifica OMG (Object Management Group) CORBA. [16]

- *device Service*: permette di attivare le misurazioni di strumenti configurati correttamente con PUMA (per esempio lo smoke meter);
- *calibrator Service*: configura una connessione con la ECU per effettuare una calibrazione.

Per sviluppare questa architettura, è necessario utilizzare un linguaggio di alto livello (C++, Java, ecc) che richiede la capacità di programmare con lo standard CORBA (*Common Object Request Broker Architecture*), sviluppato da OMG proprio allo scopo di permettere la comunicazione tra strumenti diversi a prescindere dal linguaggio di programmazione con cui sono stati sviluppati.

Quello che ha penalizzato la scelta di CAMEO e di una struttura complicata (*figura 3.3 e 3.4*) per automatizzare il banco prova è soprattutto l'incapacità di intervenire sulle strategie definite dall'automazione Real-Time oltre che il suo elevato costo. Per quanto riguarda il primo limite, la sequenza di operazioni per l'eventuale gestione del motore in caso di pericolo è stata valutata non soddisfacente, in particolare per la gestione e la messa in sicurezza del motore in caso di detonazione. Nel momento in cui si verifica tale fenomeno, i valori che quantificano il potenziale pericolo (provenienti da sistemi indicating) giungono a PUMA che successivamente li trasmette a CAMEO. Da qui, ne consegue la possibilità di avere due soluzioni: intervenire drasticamente sul motore portandolo al minimo o anche allo spegnimento, oppure agire sulla ECU, controllando per esempio l'anticipo di accensione.

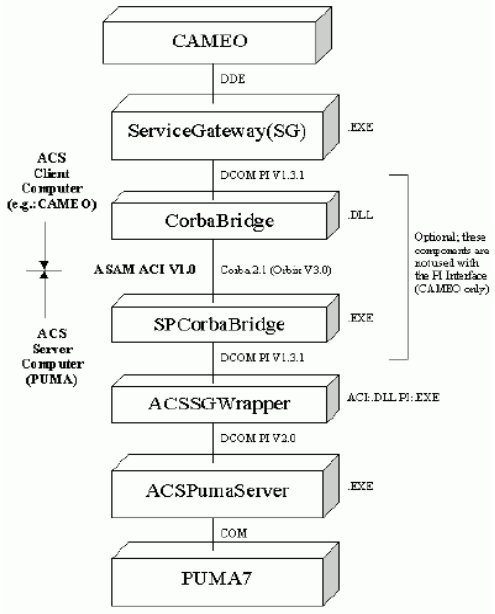


Figura 3.4: Schema comunicazione ACS CAMEO AVL

La prima soluzione ancorché drastica risulta piuttosto efficace; mentre la seconda necessita, con la soluzione CAMEO, di comunicare con la ECU tramite ASAP3, con dinamiche di intervento incompatibili con la messa in sicurezza del motore da tale fenomeno. Per il raggiungimento degli obiettivi di automazione, compatibilmente con la possibilità di intervenire in tempo reale sui parametri motore, si è scelto di escludere CAMEO e il protocollo di comunicazione ASAM-ACI. Ci si è quindi orientati verso soluzioni più semplici dal punto di vista dell'implementazione della comunicazione con PUMA, sviluppando i complessi algoritmi di controllo su un sistema esterno programmato con LV. Si è quindi sviluppata la comunicazione tra i due sistemi attraverso il protocollo CAN-bus (Controller Area Network) che in PUMA OPEN AVL è parametrizzato secondo l'ASAM-MCD2 standard o in alternativa con lo standard CANdb. Attraverso questa comunicazione si ottengono frequenze di trasmissione dati elevate oltre che una semplice parametrizzazione di PUMA Open (utilizzando file CANdb ASAM-MCD2). Per la parametrizzazione dei canali I / O possono essere letti dati di ASAM MCD2 (*.A2L) oppure VECTOR DB (*.DBC) file. Tramite un browser i dati sono disponibili per la parametrizzazione del sistema PUMA OPEN.

3.2.1 Database CAN

Il Controller Area Network, noto anche come CAN-bus, è uno standard seriale per bus di campo (principalmente in ambiente automotive), di tipo multicast, introdotto negli anni ottanta dalla Robert Bosch GmbH, per collegare diverse unità di controllo elettronico (ECU) [4]. Il CAN è stato espressamente progettato per funzionare senza problemi anche in ambienti fortemente disturbati dalla presenza di onde elettromagnetiche e può utilizzare come mezzo trasmissivo una linea a differenza di potenziale bilanciata come la RS-485. L'immunità ai disturbi può essere ulteriormente aumentata utilizzando cavi di tipo twisted pair (doppino intrecciato). Sebbene inizialmente applicata in ambito automotive, come bus per autoveicoli, attualmente è usata in molte applicazioni industriali di tipo embedded, dove è richiesto un alto livello di immunità ai disturbi. Il bit rate può raggiungere 1 Mbit/s per reti lunghe meno di 40 m. Velocità inferiori consentono di raggiungere distanze maggiori (ad es. 125 kbit/s per 500 m). Il protocollo di comunicazione del CAN è standardizzato come ISO 11898-1 (2015). Questo standard descrive principalmente lo strato (layer) di scambio dati (data link layer), composto dallo strato sottostante (sublayer) "logico" (Logical Link Control, LLC) e dallo strato sottostante del Media Access Control, (MAC) e da alcuni aspetti dello strato "fisico"

(physical layer) descritto dal modello ISO/OSI (ISO/OSI Reference Model). I protocolli di tutti gli altri layer sono lasciati alla libera scelta del progettista della rete.

Per tradurre il frame di dati, un dispositivo CAN deve essere provvisto di un database che descrive i canali contenuti nel messaggio. Il CAN Database File (File CANdb) è un file di testo contenente queste informazioni. Esso consente di trovare i dati in un frame e convertirlo in unità ingegneristiche. Il campo di dati può andare da 0 a 8 byte e può contenere più segnali (o canali) CAN di diversa lunghezza. Le caratteristiche del frame e del segnale quando si crea un file .dbc devono essere inserite correttamente analizzando sia il tipo di segnale, sia verificando la gestione del segnale stesso dalla sorgente (PUMA) presa in esame. I nomi dei segnali all'interno del file rappresenteranno le variabili in Input e Output da PUMA.

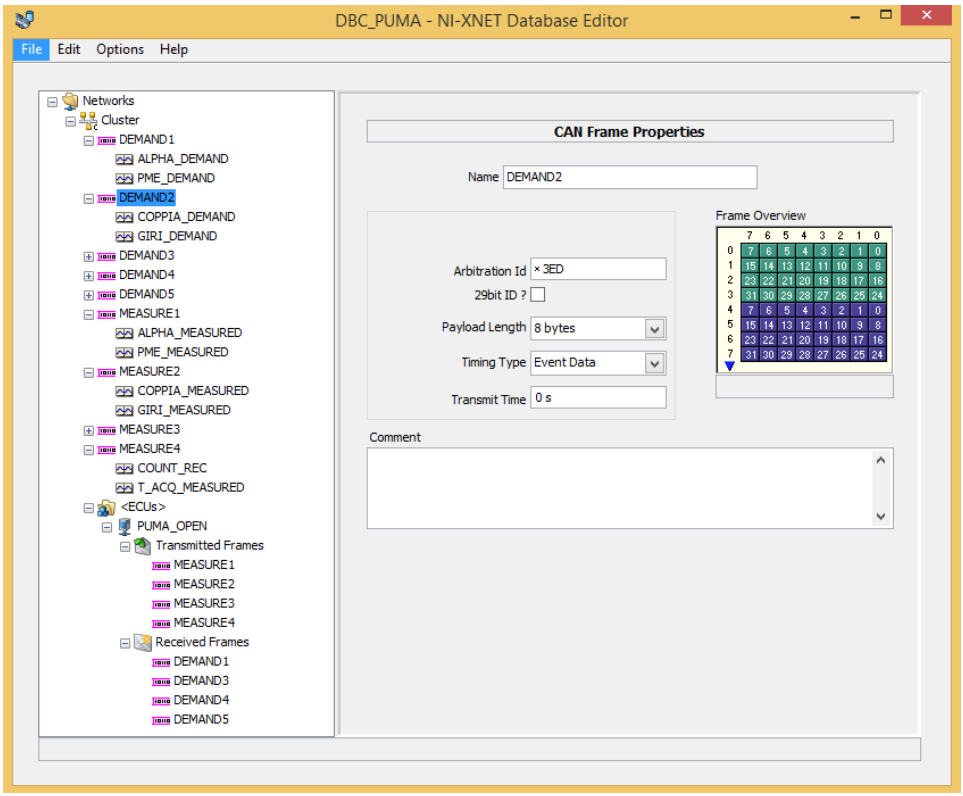


Figura 3.5: Database creato per interfacciare PUMA ai nostri sistemi.

Dall'esempio sopra illustrato, si può vedere che sono state distinte le variabili di misura "Measured" da quelle di richiesta "Demand", in questo modo sarà più semplice individuarle.

3.2.2 Normname Input e Output PUMA

La possibilità di creare un flusso in entrata e in uscita da PUMA, dipende anche dal System Parameter (SIS). All'interno del SIS sono contenuti tutti i parametri di sistema tra cui la parte relativa alla configurazione per la comunicazione attraverso il protocollo CAN-bus.

Le operazioni fondamentali sono:

- caricare il dbc (*figura 3.6*);
- impostare la scheda e la porta HW corretta;
- inserire il Bitrate;
- assegnare le variabili del dbc alle variabili di Analogic Input (*figura 3.7*) e Output;
- Iiserire i Data Type su PUMA corrispondenti a quelli utilizzati nel dbc.

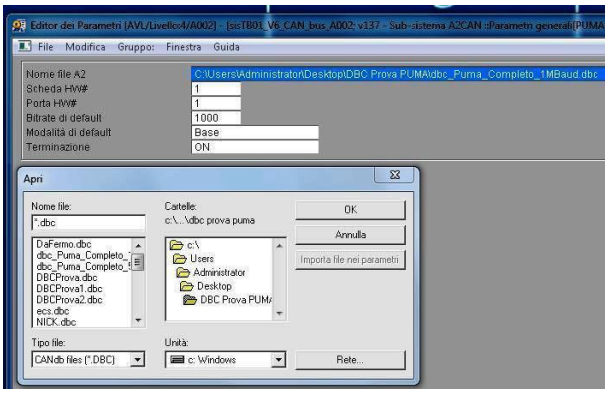
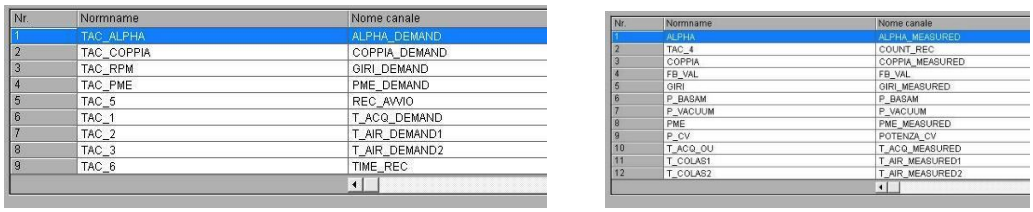


Figura 3.6: Procedura Caricamento file dbc



Nr.	Normname	Nome canale
1	TAC_ALPHA	ALPHA_DEMAND
2	TAC_COPPIA	COPPIA_DEMAND
3	TAC_RPM	GIRI_DEMAND
4	TAC_PME	PME_DEMAND
5	TAC_5	REC_AVVIO
6	TAC_1	T_ACQ_DEMAND
7	TAC_2	T_AIR_DEMAND1
8	TAC_3	T_AIR_DEMAND2
9	TAC_6	TIME_REC

Nr.	Normname	Nome canale
1	ALPHA	ALPHA_MEASURED
2	TAC_4	COUNT_REC
3	COPPIA	COPPIA_MEASURED
4	FB_VAL	FB_VAL
5	GIRI	GIRI_MEASURED
6	P_BASAM	P_BASAM
7	P_VACUUM	P_VACUUM
8	PME	PME_MEASURED
9	P_CV	POTENZA_CV
10	T_ACQ_OU	T_ACQ_MEASURED
11	T_COLAS1	T_AIR_MEASURED1
12	T_COLAS2	T_AIR_MEASURED2

Figura 3.7: A sx , analogic Input Variable e a dx analogic Output Variable

A differenza dei normname di Output, quelli di Input hanno bisogno dell'associazione di variabili libere a cui successivamente sarà attribuito il parametro che si vuole gestire. Infatti, al fine di non creare problemi di conflittualità di variabili gestite da altri utenti, sono state create all'interno del SIS di PUMA un numero pari a 24 variabili denominate "TAC".

3.3 Struttura Script BSQ/SSQ

L'associazione dei parametri scambiati attraverso il CAN bus con quelli di sala (normname) è deputata ai blocchi MEI che si presentano sotto forma di tabelle. AVL mette a disposizione un linguaggio di programmazione grafico ad alto livello: Block Sequence Editor e Step Sequence Editor (BSQ/SSQ).

La programmazione dei test in BSQ avviene in un altro applicativo dell'Application Manager, l'Explorer. Per interagire con il sistema di automazione sviluppato, è stato necessario creare una struttura di (BSQ) e (SSQ) per far eseguire test in automatico direttamente al banco prova motore.

Il linguaggio basato su BSQ/SSQ, presenta innumerevoli punti di forza:

- semplicità di scrittura data dalla sua interfaccia grafica;
- elevata flessibilità per l'esecuzione di test avanzati;
- relativa rapidità nella creazione e/o modifica di una prova;
- elevata adattabilità alle esigenze dello sperimentatore;
- controllo della variazione dei punti operativi attraverso lo Step Sequence Editor (SSQ);
- interazione con fogli Excel per l'inizializzazione dei parametri o la realizzazione di punti motore;
- possibilità di realizzare cicli altamente dinamici.

Utilizzando lo Step Sequence è possibile impostare e quindi far variare il punto operativo ed il tempo di stabilizzazione prima della misura. In tal modo, nell'esecuzione della prova in automatico è possibile ottenere un controllo ottimale dei parametri ed una completa ripetibilità delle prove. PUMA OPEN AVL offre quindi la possibilità di utilizzare script anche in sistemi di automazione. Ciò consente agli utenti di accedere a una serie selezionata di funzioni del sistema di automazione attraverso questi componenti di scripting. Gli Script si basano su funzioni fornite (contesti script) da Microsoft nel sistema operativo MS Windows TM. Questi contesti sono incorporati nella API PO (Application Programmers Interface Puma Open) Script e costituiscono un ambiente di esecuzione separato. Sulla base della definizione di un tale contesto, il cliente può definire script che possono essere richiamati direttamente all'interno del BSQ o SSQ. Questo tipo di linguaggio mette a disposizione degli script in visual basic attraverso i quali, dopo un'opportuna programmazione è possibile interagire attraverso dispositivi esterni, ad esempio fogli excel o matlab, per l'inizializzazione dei parametri, la realizzazione di DoE

o l'analisi RT di particolari grandezze. Nel caso trattato si è invece creata una struttura che permettesse la gestione di PUMA attraverso il sistema automazione interno.

3.4 Gestione parametri motore

AVL EXPLORER di PUMA Open offre una libreria all'interno di cui si possono trovare alcuni blocchetti relativi agli script BSQ/SSQ di cui si è precedentemente parlato. Solitamente questi vengono utilizzati per effettuare delle prove in automatico relativi a punti statici che non prevedono l'intervento di applicazioni esterne. Inserendo adeguatamente i blocchi BSQ/SSQ e creando le corrette assegnazioni di script si può creare un ciclo che permette al PUMA di interagire con un sistema di automazione esterno, nel nostro caso con il TAC. Di seguito è illustrato un esempio di struttura di ciclo automatico (*figura 3.8*) creato appositamente per gestire il numero di giri, carico (percentuale di alzata pedale), temperatura acqua, temperatura aria in aspirazione e la parte di acquisizioni dati.

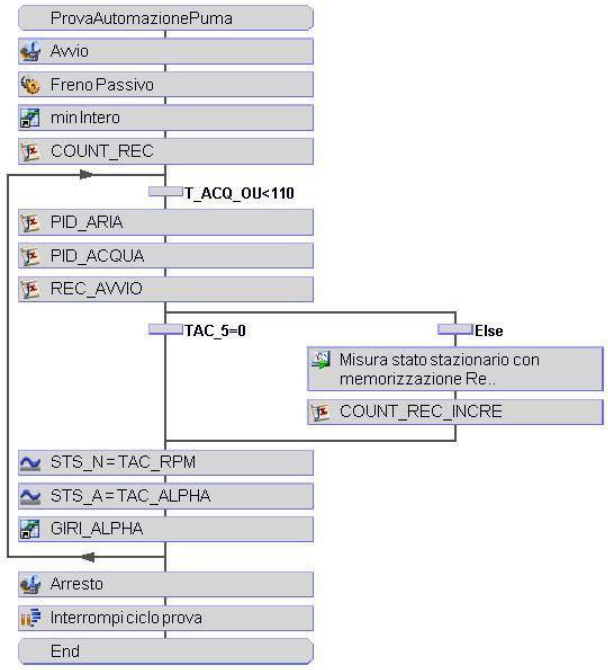


Figura 3.8: Struttura Ciclo Automatico

La struttura ad albero esegue in ordine seriale gli script caricati, per questo bisogna creare un ciclo e se necessario bisogna ripetere più volte la stessa procedura. Ogni ciclo deve iniziare con l'AVVIO del motore al minimo. Per motivi di sicurezza è conveniente impostare il freno nella modalità passiva. Nella sequenza while inserita nella fase sopra rappresentata, è stata inserita la condizione di fine ciclo ad una temperatura di acqua

motore superiore a 110°C. Questo significa che tutti i comandi all'interno del ciclo while si ripeteranno fino a quando la temperatura dell'acqua motore non raggiungerà i 110°C (questa è una condizione limite). Per attribuire alle variabili, inserite all'interno del SIS di PUMA Open AVL, la possibilità di condurre alcuni comandi della sala motore, solitamente gestiti da PUMA, bisogna creare degli script di assegnazione al cui interno saranno inseriti i normname relativi alle variabili che si vogliono modificare dall'esterno (vedi *figura 3.9*).

Figura 3.9: Script di assegnazione della variabile che regola la temperatura aria in aspirazione

La parte relativa alla registrazione dati, è stata gestita attraverso un ciclo If/Else che in base al segnale inviato avvia e interrompe la registrazione. Sono state implementate due tipi di strutture per due tipologie di acquisizioni dati.

- Recorder dati che acquisisce tutti i punti con la frequenza indicata;
- Misura dati, effettua la media dei punti basandosi sulla frequenza di acquisizione nell'intervallo di tempo specificato (*figura 3.10*).

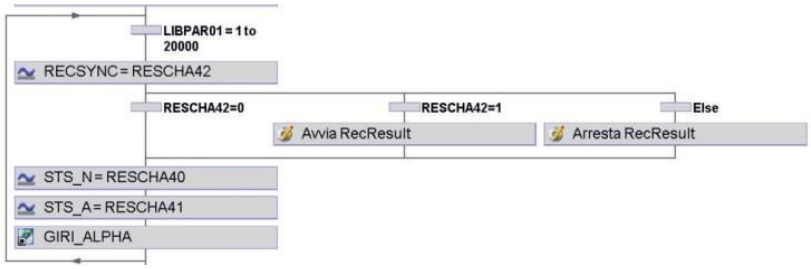


Figura 3.10: Struttura BSQ Recorder

Per gestire anche il tempo di acquisizione è stata creata una variabile all'interno dello script di Misura e di Recorder che permette di variare la lunghezza temporale dell'acquisizione dei dati.

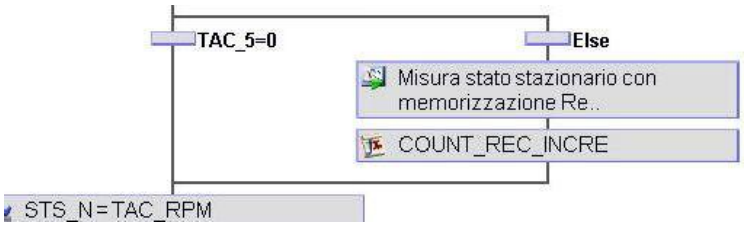


Figura 3.11: Struttura BSQ Misura

All'interno di questi blocchetti bisogna anche specificare il canale di registrazione e il gruppo di canali che si vuole acquisire (appartenenti ad Indicom, Inca e ad altri strumenti presenti in sala e che comunicano con PUMA).

La gestione di parametri come carico e regime, utilizza una struttura di linguaggio indipendente da tutte le altre. Infatti, non si può solo creare un semplice script di assegnazione per variare il numero di giri, ma bisogna creare uno Step Sequence che permetta la modifica in tempo reale. PUMA controlla il regime e il carico attraverso delle Grandezze Speciali, elencate di seguito:

- Giri Demand *STS_N*
- Alpha Demand *STS_A*
- PME Demand *Sbv_x*
- Coppia Demand *Sbv_t_engine*

Inserendo la parte relativa allo script SSQ nel ciclo prova e scegliendo come funzione la parte di Dinamica Motore, con una frequenza di ciclo pari a 0,1 s sarà possibile impostare gli RPM e Il carico del motore in modo automatico da un PC host. Per motivi di sicurezza all'interno del ciclo sono state specificate alcune impostazioni che permettono all'operatore di chiudere la connessione di PUMA con l'esterno e di riprendere in manuale il completo controllo della cella motore.

3.5 AVL indicom

Il software in uso nelle sale prove MASERATI per l'analisi dei segnali di pressione provenienti da sensori montati in modo da rilevare la pressione in camere di combustione, è INDICOM di AVL. Il software consente la visualizzazione del valore di pressione in camera di combustione nei cilindri (*figura 3.12)* e l'analisi della combustione. Per la comunicazione con il TAC, il protocollo di comunicazione in utilizzo è quello CAN, protocollo diffuso in ambito automotive grazie alla stabilità di comunicazione. Per la comunicazione con il TAC via CAN è necessario creare un database (questo è fatto direttamente su INDICOM) e condividerlo con il software di gestione del protocollo CAN

sviluppato per il TAC. Nel database CAN sono contenute le definizioni di tutti i canali che saranno trasmessi via CAN al TAC utili per l'automazione, ad esempio: valori di CA MFB50, IMEP, COV% e KP_PK (indicatore di detonazione equivalente al MAPO) fino a un massimo di 56 canali. Per l'acquisizione dei dati, il PC Real-Time è fondamentale per lavorare a frequenze molto elevate e fisse (cosa che non accadrebbe con un'applicazione Host).

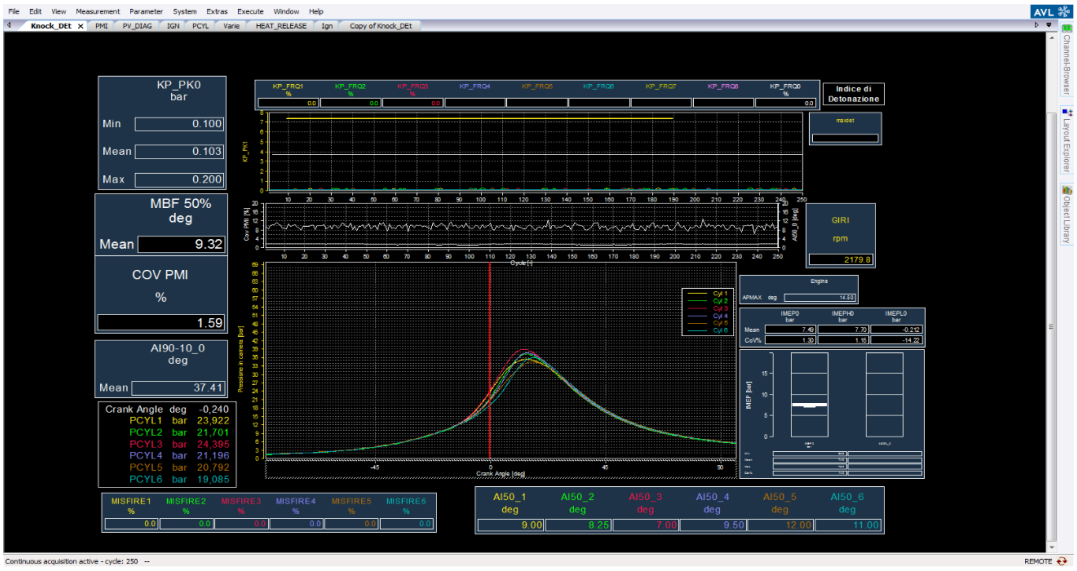


Figura 3.12: Interfaccia grafica INDICOM AVL

4 Gestione ECU

4.1 Introduzione

Come è noto, la gestione degli attuatori motore, al banco come in veicolo, è demandato alla ECU, un vero e proprio sistema di controllo, dotato di processore con un determinato programma con input, output e parametri di controllo.

Gli input corrispondono a segnali che arrivano acquisiti dalla ECU dai numerosi sensori presenti sul motore (segnale ruota fonica, pressione aspirazione, ecc.), mentre, per quanto riguarda gli output, essi rappresentano i valori delle leve di controllo disponibili su un motore (anticipo di accensione, timing di iniezione, apertura valvole, ecc.) che la ECU attua per gestirne il funzionamento. Infine i parametri di controllo sono le variabili che permettono al calibratore di decidere quale reazione (output) deve avere la ECU in determinate condizioni di funzionamento (input). Queste variabili possono essere modificate durante la fase di calibrazione (quando si hanno a disposizione ECU di sviluppo), e, una volta ottimizzate, vengono deliberate per essere implementate nelle ECU di serie e installate sui veicoli di produzione.

4.1.1 Centralina Sviluppo

Tipicamente in una ECU di sviluppo viene inserito un componente elettronico aggiuntivo che prende il nome di: ETK (Emulator TastKopf). Durante le fasi della calibrazione un cospicuo volume di informazioni sono scambiate tra PC di calibrazione ed ECU, siano esse misure o calibrazioni. L'ETK è responsabile della gestione della comunicazione tra PC ed ECU.

Tramite la connessione con ETK è quindi possibile eseguire:

- emulazione di dati;
- emulazione di codice,
- controllo dell'emulazione di dati nella ECU.

In una centralina di sviluppo in aggiunta al codice di controllo sono memorizzati due set di dati (*figura 4.1*):

- La WorkingPage (WP): è il set di dati in cui vengono apportate le modifiche ai parametri di funzionamento durante la procedura di calibrazione.
- La Reference Page (RP): è un set di dati protetto da scrittura.

È possibile passare dalla RP alla WP in qualsiasi momento. Per poter gestire due set di dati una centralina deve subire modifiche sia hardware che software.

Esistono diverse tipologie di ETK. Ad esempio, l'ETK parallela sostituisce completamente la memoria FLASH della ECU con una RAM. La WP e la RP sono memorizzate in una Dual Port RAM, cioè una memoria ad accesso simultaneo, nella quale è possibile modificare i parametri durante il funzionamento e trasmetterli alla ECU. In questo specifico caso, anche il codice di controllo è salvato sulla ETK. Nel caso di mancanza di alimentazione i dati contenuti nella RAM vengono persi, ma al successivo avvio il codice e il dataset contenuti nella ETK FLASH vengono nuovamente scaricati nella RAM e il funzionamento è garantito anche senza la connessione a un PC.

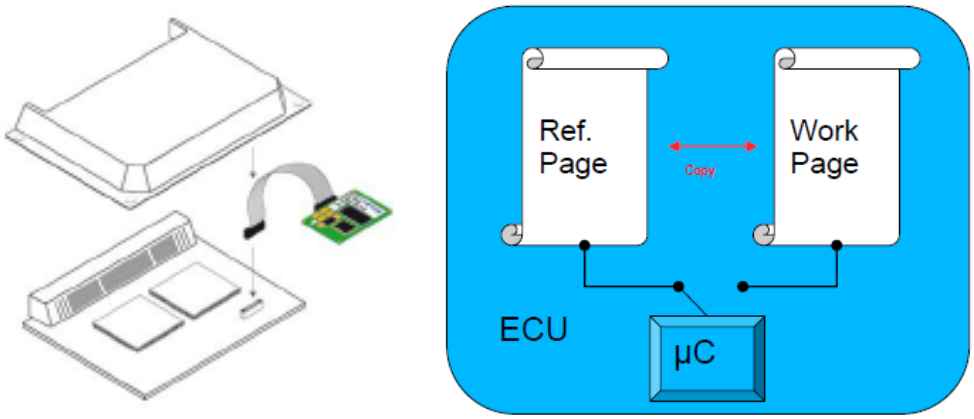


Figura 4.1: centralina di sviluppo. A sx l'aggiunta dell'ETK (o xETk); a dx lo schema della gestione dei set di dati

4.1.2 ETAS INCA

INCA è il software commerciale di ETAS fra i più diffusi nelle sale prove motori, che permette al calibratore di leggere i valori degli input e degli output, definiti Measurements, e di agire sui parametri di controllo, chiamati Calibrations, al fine di identificare i valori che permettono il miglior funzionamento del motore [5]. Attraverso esso è possibile gestire in locale (sul PC di gestione ECU) sia le RP che le WP ed è possibile scaricare all'interno di una ECU nuovi software che è necessario testare/validare (operazione di ECU flashing). Senza entrare troppo nel funzionamento del SW, si riporta in *figura 4.2* un esempio di Experiment, ossia l'interfaccia di INCA da cui è possibile controllare Measurements e Calibrations. Dalla stessa figura si nota come le Calibrations possano essere di diversa forma e dimensione; si hanno infatti calibrazioni booleane (1), scalari (2), vettoriali (3) e matriciali (4), mentre le Measurements (5) sono in genere scalari.

All’interno di INCA la Reference Page e la WorkingPage vengono emulate e possono essere modificate off/on-line e poi scaricate all’interno della ETK. È anche possibile copiare la RP e WP dalle ETK. Si tenga conto che INCA permette l’accesso alla memoria della ECU in formato binario e per poter interpretare i dati grezzi che riceve ha bisogno di una descrizione di come questi dati sono organizzati all’interno del SW (nome, indirizzo di memoria, formattazione, formule di calcolo, ecc.). La completa descrizione della ECU è contenuta nel *project description file* (.a2l) che deve essere caricato dal calibratore prima di iniziare la comunicazione.

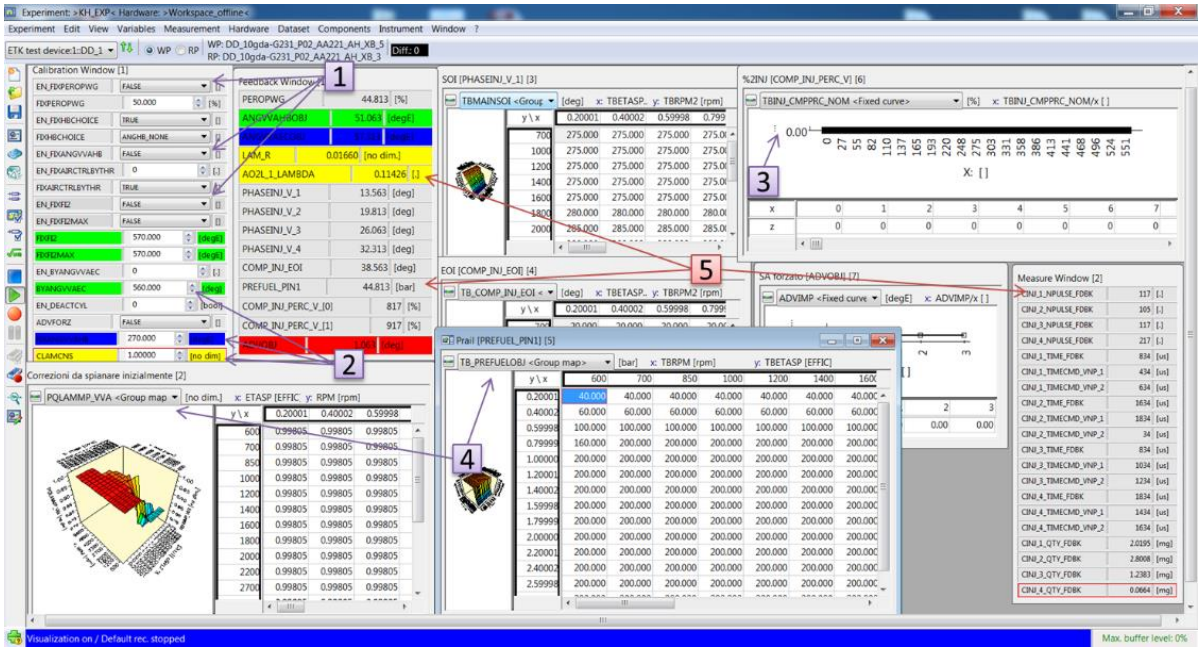


Figura 4.2: Interfaccia dell’Experiment: si riconoscono le Calibrations dalle Measurements perché le prime hanno lo sfondo bianco e le seconde grigio.

Contemporaneamente a questo file ne deve essere caricato un altro, il file .hex, in cui sono contenuti il software vero e proprio presente in centralina, con tutte le logiche di controllo implementate dal costruttore, e i dataset, ovvero le variabili di calibrazione con dei valori di default o impostati dal calibratore nelle fasi precedenti considerando che la completa descrizione della ECU è contenuta nel project description file (.a2l) descritto in precedenza.

In particolare, Maserati ha scelto di utilizzare per questo progetto due ECU con modulo xETK: questa tecnologia non si differenzia dall’ETK per la funzione, che rimane esattamente la stessa, ma per il protocollo di comunicazione utilizzato. In particolare gli xETK utilizzano per il trasferimento dati il protocollo XCP, standard appartenente ad

ASAM (Association for Standardization of Automation and Measuring Systems), che li rende facilmente integrabili in un ambiente di programmazione come LV. Questi moduli sono in grado di elaborare fino a 4 canali di comunicazione contemporaneamente, cosa che permette un accesso rapido ed in parallelo tra diverse applicazioni.

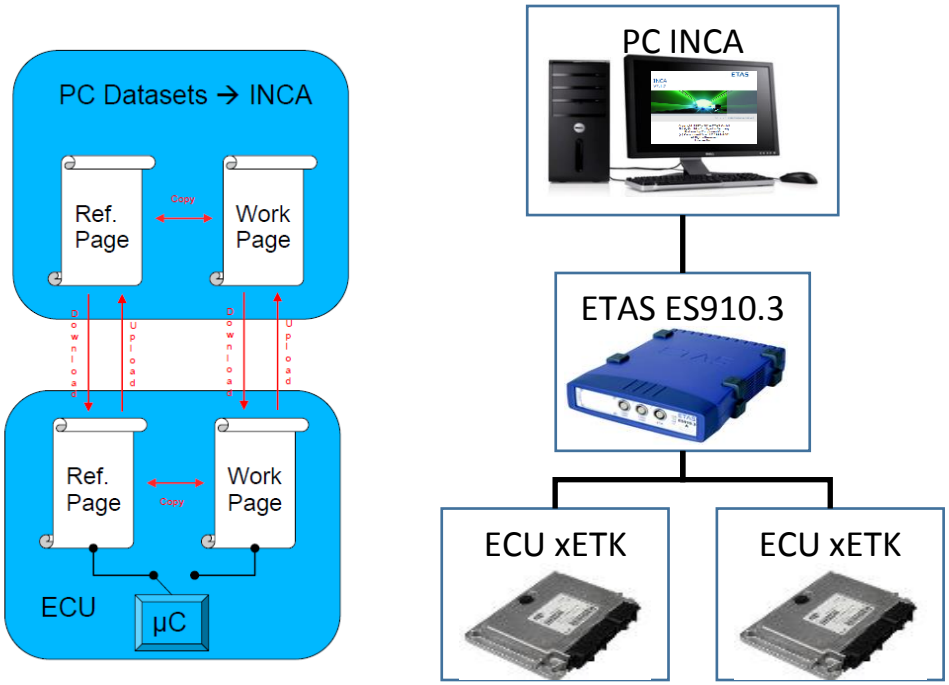


Figura 4.3: A sx, gestione del SW centralina da parte di INCA. A dx schema del layout presente in sala prove Maserati.

Le centraline con ETK/xETK non bastano, infatti per poter accedere alla memoria e per modificarla sono necessari dei moduli forniti da ETAS che permettono la connessione via ethernet del pc con INCA con la ECU. A seconda del tipo di protocollo di comunicazione, dell'applicazione, della centralina utilizzata esistono diversi moduli ETAS. Per il progetto esposto in questo elaborato è necessario il modulo ETAS ES910.3, un modulo di prototipazione rapida con elevate prestazioni computazionali che viene utilizzato per diverse applicazioni. In questo caso l'elevata capacità computazionale permetterà di trasferire su di esso alcune delle funzioni del SW INCA che potrà essere bypassato velocizzando la comunicazione e permettendo esecuzioni RT.

Nei paragrafi successevi, verranno brevemente mostrati i protocolli e le modalità di comunicazione del sistema di gestione della ECU con i sistemi di automazione. Il primo descritto è l'ASAP3 che rappresenta un po' lo standard fin qui utilizzato per la connessione fra sistemi differenti, successivamente verrà mostrato il protocollo ASAM XCP, utilizzato per la gestione delle interfacce delle applicazioni create e come base di

inca MCE e del protocollo iLinkRT che descriveremo in maniera approfondita come ultimo protocollo.

4.2 ASAM ASAP3

Questo protocollo è uno dei più utilizzati per la gestione dell'automazione in sala prova. Questo paragrafo contiene le informazioni di base sull' integrazione del motore con sistemi elettronici in un sistema banco di prova (Ausy - Sistema di automazione) tramite un sistema di taratura elettronica (MC System – Measurment Calibration) [6]. L'attuale integrazione avviene tramite una connessione dati di due tipologie:

- seriale RS232;
- ethernet con protocollo TCP / IP.

La *Figura 4.4* mostra la struttura hardware di base di questa integrazione. Esso dimostra chiaramente che il sistema MC viene utilizzato come ripetitore intelligente tra l'AuSy e il sistema di gestione dei parametri centralina (ECU - Electronic Control Unit). Le varie centraline possono essere azionate tramite il sistema MC, che può anche essere utilizzato in modo interattivo come dispositivo stand-alone. La comunicazione si basa sostanzialmente su un flusso di richieste da parte dell'Ausy del nome della grandezza e dimensioni in unità fisiche (esempio, rispettivamente flusso d'aria e m^3/h). Tutti i nomi o le etichette utilizzate nel sistema Ausy e MC sono definite in un file di descrizione dei dati conforme allo standard ECU ASAP2 (il file a2l che verrà ampiamente trattato successivamente). Il pacchetto software sul sistema MC che di conseguenza, è responsabile della preparazione e conversione in valori esadecimali dei dati per gli indirizzi fisici di memoria delle centraline.

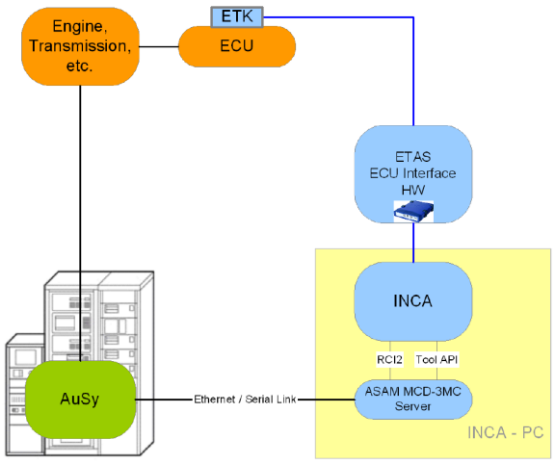


Figura 4.4: Layout dello standard ASAM ASAP3 con INCA

Quando si avvia l'interfaccia INCA ASAP3, si avvia la scrittura di comandi e le risposte nel dettaglio di tutti i parametri e si può monitorare la comunicazione tra il banco prova e INCA. È possibile impostare il grado di registrazione in modo da poter visualizzare specificamente eventi particolari, per i messaggi di errore di esempio. Visto il possibile verificarsi di problemi nella comunicazione tra il Ausy e il sistema MC (si veda la sequenza di comunicazione, *figura 4.4*) è possibile inserire una logica di richiesta e risposta in cui il sistema MC conferma la ricezione di una specifica richiesta a un comando dal Ausy, dopo aver ricevuto la richiesta correttamente. Questa implementazione da un lato è indispensabile per garantire il flusso di dati in sistemi come le ECU, dall'altro introduce delle latenze maggiori in quanto i tempi di implementazione di un comando raddoppiano.

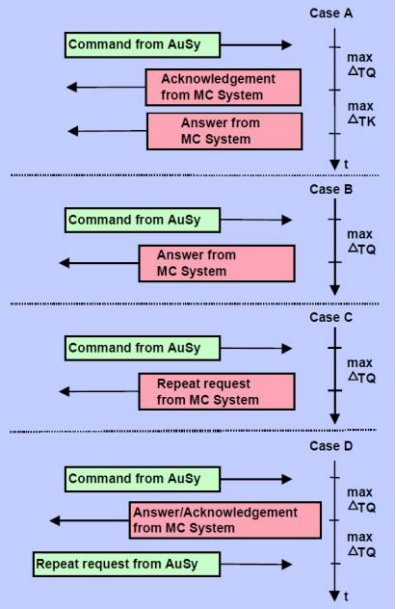


Figura 4.5: Possibili sequenza di comunicazione ASAP3

4.3 ASAM XCP

L'ASAM MCD-1 XCP è l'abbreviazione di "Universal Measurement and Calibration Protocol" ed è un protocollo standard redatto da ASAM che definisce un protocollo di comunicazione master-slave indipendente dal bus, per il collegamento della ECU con sistemi di calibrazione e di elaborazione di strategie di controllo [7]. Lo standard è stato redatto a partire dal 2003 e viene utilizzato sia dai costruttori che dai produttori di ECU. Lo standard trae origine dal precedente ASAM standard di MCD-1 CCP, che è un protocollo di misura e taratura specifico per il bus CAN. Lo scopo principale di questa

metodologia è quello di modificare i parametri (calibration) ed acquisire i valori istantanei delle variabili interne di una ECU (measure). La prima lettera X in XCP esprime il fatto che il protocollo è progettato per una varietà di sistemi di bus: CAN, FlexRay, Ethernet (UDP / IP e TCP / IP), collegamenti seriali (SPI e SIC) e USB. Nel dettaglio, L'ASAM MCD-1 XCP definisce l'accesso a parametri e variabili di misura che utilizzano indirizzi di memoria. Le proprietà e indirizzi di memoria di questi dati sono descritti nel formato A2L-file, che viene redatto tramite lo standard ASAM MCD-2 MC. Il file A2L contiene tutte le informazioni necessarie per accedere e interpretare correttamente i dati che vengono trasmessi tramite il protocollo XCP prevedendo pertanto l'accesso ad un parametro, senza la necessità di avere accesso al software applicativo ECU. In altre parole, la centralina contiene solo la generica risposta alle richieste di accesso alla memoria da parte del sistema di calibrazione. Diverse operazioni di calibrazione e di misurazione possono essere eseguite senza dover ricompilare e riprogrammare il codice dell'applicazione ECU.

4.3.1 Ambiti di utilizzo.

XCP può essere utilizzato in tutte le fasi di sviluppo ECU, come l'ottimizzazione di funzioni, la calibrazione e il collaudo delle ECU e permette di realizzare sistemi con elevate velocità di trasferimento dati e tempi di ciclo brevi (range di qualche microsecondo) risultando molto utile quando si analizza il comportamento dinamico di sistemi elettromeccanici, in maniera rilevante per il settore automotive (motore, trasmissione o azionamenti elettrici). Una funzionalità chiave di ASAM MCD-1 XCP è quella di consentire l'accesso in lettura e scrittura alla memoria della centralina:

- l'accesso in lettura, comunemente chiamato "misura", consente agli utenti di misurare una variabile interna. Le ECU sono sistemi con comportamento definibile a "tempo discreto", le cui variabili vengono aggiornate solo a intervalli di tempo specifici e definiti dall'applicatore; è quindi uno dei grandi punti di forza dello standard, quello di acquisire valori misurati dalla RAM periodicamente o in occasioni di eventi specifici. Questa periodicità permette la valutazione diretta del comportamento della ECU nel corso del tempo, durante l'acquisizione dei valori registrati da variabili interne e sensori esterni. L'accesso all'indirizzo è definito nel senso che la comunicazione tra master (cioè lo strumento di misurazione e taratura) e slave (cioè la centralina) fa riferimento a indirizzi in memoria. Così, la misurazione di una

variabile è essenzialmente implementata come una richiesta del master allo slave (ad esempio "Dammi il valore della locazione di memoria 0x1234").

- L'accesso in scrittura, chiamato nello standard "calibrazione", permette all'utente di ottimizzare i parametri degli algoritmi della ECU. La calibrazione di un parametro è essenzialmente implementata come una richiesta del master allo slave (ad esempio "Impostare il valore all'indirizzo 0x9876 a 5"). Le proprietà e gli indirizzi di memoria dei parametri sono descritti nel formato A2L-file (standard ASAM MCD-2 MC). Ciò significa che l'accesso ad un parametro specifico non deve essere codificato nell'applicazione ECU. In altre parole, la centralina contiene solo una generica implementazione del protocollo slave XCP, che risponde alle richieste del master di accedere alla specifica memoria del sistema di calibrazione.

Diverse operazioni di calibrazione e di misurazione possono essere eseguite dal sistema di calibrazione senza dover ricompilare e riprogrammare il software ECU.

4.3.2 Communication Model

Il protocollo XCP si basa sul principio master-slave in cui la ECU è lo slave e lo strumento di misurazione e calibrazione è il master. Uno slave può comunicare solo con un master in un dato momento mentre il master può comunicare simultaneamente con più slave. Il pacchetto XCP è incorporato in un frame (struttura) di un determinato transport layer (nel caso di XCP on Ethernet con UDP in un pacchetto UDP). Il telaio è costituito da tre parti: XCP header, XCP packet e la XCP tail. Nella *figura 4.6*, la parte di pacchetto XCP all'interno del messaggio è visualizzata in rosso. L'intestazione e la coda XCP dipendono dal protocollo di trasporto.

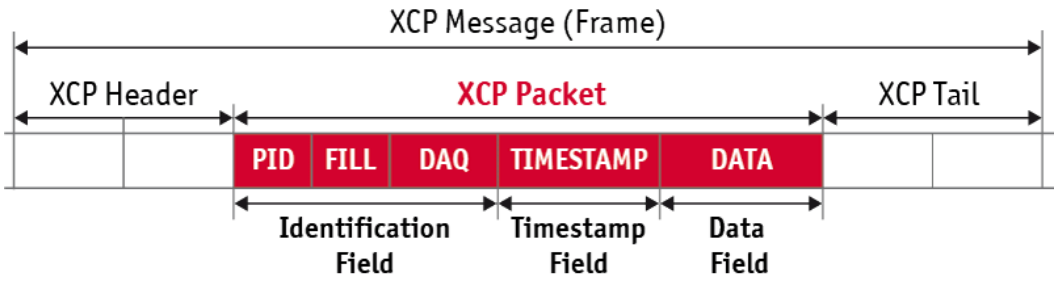


Figura 4.6: Composizione di un messaggio XCP (Frame)

Il pacchetto XCP è indipendente dal protocollo di trasporto utilizzato. Esso contiene sempre tre componenti: "Identification Field", "Timestamp Field" e "Data Field".

Il tipo di comunicazione tramite pacchetti XCP è suddiviso in una serie di comandi (CTO) e un intervallo per l'invio dei dati sincroni (DTO). I comandi vengono scambiati tramite CTO (Command Transfer Objects). Per esempio, il master inizia un contatto e lo slave deve rispondere a un CMD con RES o ERR. Gli altri messaggi CTO vengono inviati in modo asincrono. Gli oggetti di trasferimento dati (DTO) vengono utilizzate per trasferire i dati di misurazione e di stimolazione sincrona.

I comandi standard XCP permettono di avviare o interrompere la comunicazione, identificare l' ECU software (versione), sbloccare risorse protette, ottenere lo stato della sessione in corso e fornire accesso in lettura alle memoria della centralina. Al fine di velocizzare gli accessi di memoria, il sistema di calibrazione può memorizzare nella cache la memoria della calibrazione e ottimizzare i processi inviando solo le parti modificate di un determinato set di parametri. Ciò richiede che la cache e la memoria della centralina siano identiche. Per garantire questo, XCP specifica i comandi per il calcolo del checksum di memoria.

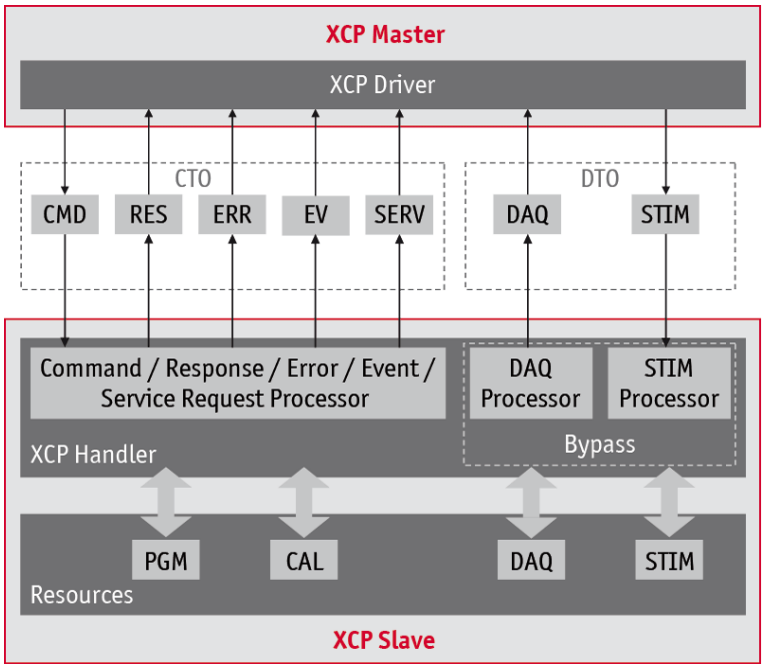


Figura 4.7: Struttura del processo di comunicazione XCP.

Di seguito verranno illustrati alcuni di questi comandi al fine di comprendere le logiche con cui è sviluppato il protocollo

4.3.3 Data Acquisition (Measurement):

Si tratta della parte di stimolazione da parte del master all'invio da parte dello slave delle measure. Due sono le modalità descritte di seguite, anche se la seconda è quella da preferire per aumentare l'efficienza della comunicazione.

1. Read-Service based data acquisition (Polling). Per leggere le locazioni di memoria richieste, il modo più semplice è quello di inviare richieste periodiche dal master XCP allo slave. Lo svantaggio di queste richieste periodiche è che almeno due messaggi vengono generati per ogni segnale (comando e risposta) e il tempo di acquisizione non è sincrono con il tempo di esecuzione di esecuzione degli algoritmi di calcolo della ECU.
2. Synchronous Data Acquisition (DAQ). Per ovviare agli inconvenienti della modalità polling, XCP offre la modalità di scambio dati sincrona. Nel flusso di controllo di un algoritmo ECU, eventi XCP (DAQ) possono essere richiamati allo slave XCP, che innesca il campionamento dei dati di misura. Quando lo slave è stimolato dal master da un evento di campionamento, l'XCP slave raccoglie i valori dei parametri di misura, li organizza in un buffer e li invia al master a condizione che lo slave sappia quali variabili competono a tale evento. Lo slave ottiene queste informazioni durante la fase di configurazione DAQ, in cui il master invia comandi specifici che definiscono le variabili di misura richieste ai loro eventi XCP associati. Dopo il completamento della fase di configurazione, master e slave instaurano la comunicazione delle measure, senza ulteriori richieste di lettura esplicite del master verso lo slave. Un evento non deve essere necessariamente ciclico e a tempo costante, nel caso di una ECU motore infatti, l'evento potrebbe essere base angolo rendendo l'intervallo di tempo tra due eventi dipendente dalla velocità angolare del motore. Oltre a questo aspetto, anche eventi singolari, quali attivazioni arbitrarie di un interruttore da parte del conducente, sono anch'esse non cicliche e non temporaneamente equidistanti. XCP fornisce la funzione di "timestamp" sul campione di dati di misura acquisiti presumendo che una ECU ha tipicamente un clock interno che può essere utilizzato per identificare nel tempo dati di misura.

4.3.4 Calibrazione (CAL)

Le "calibration" della ECU sono parametri costanti che vengono regolati e ottimizzati durante lo sviluppo della centralina. Questo è un processo iterativo, in cui il valore

ottimale di un parametro è ricavato attraverso la sperimentazione. Per rendere i parametri modificabili in fase di sviluppo di una ECU, è necessaria memoria RAM aggiuntiva. L'approccio più frequentemente usato per modificare i parametri in fase di esecuzione ("calibrazione on-line") è quello di modificare i parametri nella memoria RAM disponibile. Durante l'avvio della centralina, tutte le variabili RAM vengono inizializzate una volta con i loro valori di default dalla memoria flash e viene solitamente eseguita nel codice di avviamento del fabbricante (applicatore). Successivamente l'applicazione utilizza i valori dei parametri presenti nella RAM che possono essere modificati tramite normale accesso alla memoria XCP. Dal punto di vista del software ECU, i parametri di calibrazione nella RAM sono immutabili, vale a dire l'applicazione stessa non può autonomamente modificarli.

Dal punto di vista dello strumento di calibrazione, l'area RAM in cui si trovano i parametri viene indicato come RAM di calibrazione (memoria che può essere calibrata).

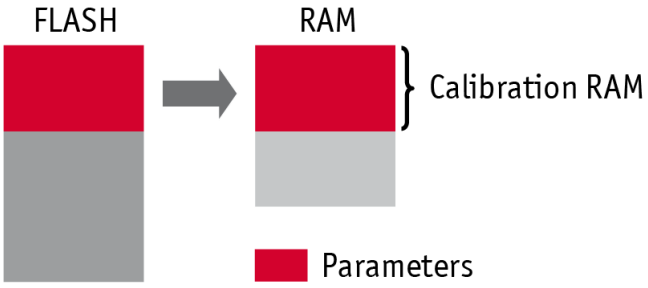


Figura 4.8: Schema a blocchi dell'utilizzo di memoria da parte per l'invio dei parametri

4.3.5 STIM Data Transmission, memory layout, page switching

- La funzione STIM è molto simile alla DAQ descritta in precedenza, ma funziona nella direzione opposta. Utilizzando STIM, lo strumento di calibrazione è in grado di sovrascrivere i valori di variabili dell'algoritmo di controllo in modo sincrono. Questa funzione può essere utilizzata per implementare un bypass, dove le parti di algoritmi di controllo della ECU sono calcolate al di fuori della centralina da un sistema di bypass esterno. Una configurazione STIM è impostata usando esattamente gli stessi comandi come una configurazione di misura DAQ.
- Il layout della memoria logica della ECU (slave XCP) è descritto da oggetti chiamati segmenti di memoria che hanno attributi che descrivono il contenuto e il tipo di accesso ai parametri, ad esempio DATA + RAM o CODICE + FLASH. Un segmento

è descritto utilizzando l'ASAM MCD-2 MC MEMORY_SEGMENT, che contiene informazioni come il nome, l'indirizzo e le dimensioni. Le informazioni specifiche XCP si trovano all'interno di una sezione IF_DATA, come ad esempio le definizioni di pagina e i parametri che possono essere regolati dal master devono essere situati in segmenti di memoria "fisicamente" situati nella RAM.

- Se questi servizi sono implementati, lo strumento di calibrazione è in grado di controllare la pagina attiva. Se si passa da un segmento di memoria Flash a una pagina RAM, un parametro situato in questo segmento di memoria può essere modificato durante l'esecuzione dell'algoritmo di controllo. Questo approccio è noto anche come calibrazione in linea.
- L'accesso da parte del tool di calibrazione all'algoritmo della ECU può essere modificato in modo indipendente e consente modifiche di parametri di grandi dimensioni (ad esempio mappe) senza compromettere la consistenza dei dati durante le operazioni di scrittura nella memoria della centralina. Tale coerenza è realizzata al primo accesso dello strumento di calibrazione alla pagina di RAM e, successivamente, si passa alla scrittura dei dati alla centralina, e infine al passaggio alla pagina di RAM per la ECU.

4.4 INCA MCE

Il modulo ES910.3 è il fulcro della comunicazione via iLinkRT che, come si vedrà nel prossimo capitolo, è quella che permette di interagire con la ECU in modo rapido dando la possibilità di rendere automatizzate alcune operazioni di calibrazione finora demandate all'operatore di sala. Questo tipo di comunicazione necessita di un'applicazione di INCA chiamata MCE (Measurement and Calibration Embedded): si tratta di una componente del SW descritto in precedenza, che permette di spostare alcune delle operazioni di calibrazione dal PC su HW specifici (il modulo ES910.3 appunto) in grado di comunicare ad una velocità elevatissima con sistemi di automazione. Il sistema di calibrazione utilizzato finora nelle sale prova Maserati è quello riportato in *figura 4.9*, in cui con un modulo ES592 si accede alle ECU con ETK attraverso INCA. INCA è a sua volta controllato dal sistema di controllo banco, ad esempio AVL PUMA Open, attraverso l'interfaccia ASAP3 che permette così di chiudere il loop con il motore. Con questa struttura la conversione dei dati, che sono trasmessi dalla ECU in formato esadecimale, avviene all'interno di INCA.

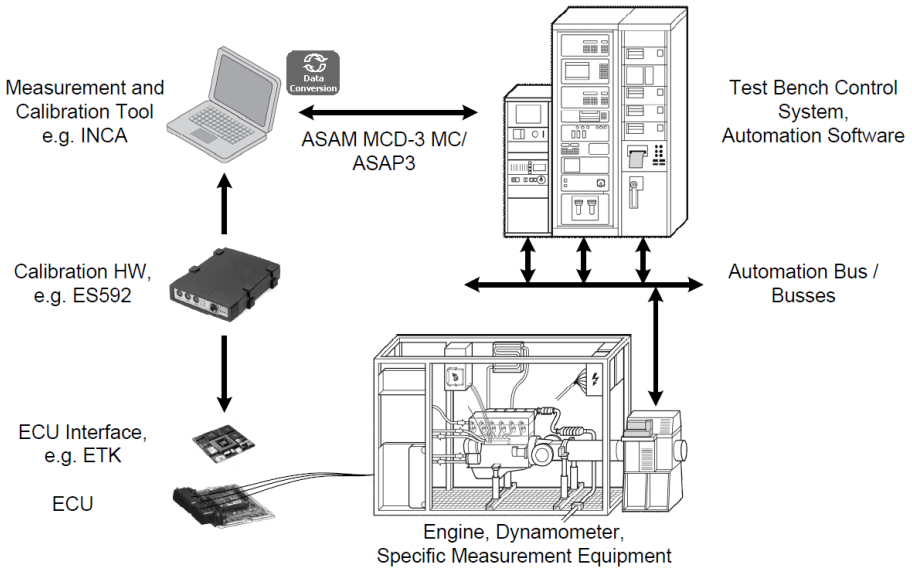


Figura 4.9: Struttura standard di comunicazione con la ECU, presente anche nelle sale prova di Maserati.

I limiti di questa configurazione fin qui utilizzata sono:

- tempi relativamente lunghi per effettuare un ciclo di calibrazione (dell'ordine dei 100 ms per caricare in centralina una mappa 16×16 di valori);
- tempi di calibrazione variabili a causa del fenomeno di jitter, ovvero i ritardi nell'elaborazione di un segnale dovuti a code che si creano su un sistema operativo come Windows, dove una percentuale non definita delle risorse viene dedicata ad altri processi;
- ASAP3 è un protocollo di comunicazione a blocchi, e quindi non è possibile acquisire misure durante la fase di calibrazione e viceversa;
- ASAP3 non supporta un'architettura multi-master, impedendo per esempio ad un SW di automazione e ad un SW di auto-calibrazione di accedere contemporaneamente alla ECU.

Oltre al superamento di questi limiti ad INCA-MCE è richiesta la completa compatibilità con i sistemi di controllo banco esistenti, bassi costi di migrazione da strutture precedenti e mantenimento della stessa accessibilità alle variabili già propria di INCA.
Ecco quindi che per soddisfare tutti questi requisiti, ETAS ha scelto di spostare la fase di decodifica dei dati su un HW dedicato, sviluppando contemporaneamente un protocollo di comunicazione veloce tra MCE e un qualsiasi SW di automazione (vedi *figura 4.10*).

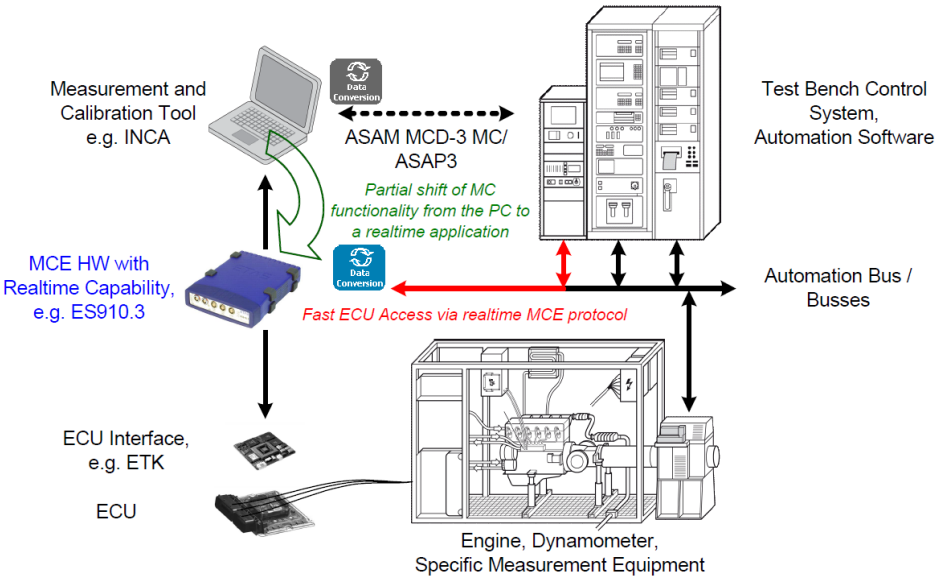


Figura 4.10: Struttura di sala utilizzando la soluzione MCE.

Per quanto riguarda il funzionamento di questi SW, lo si può dividere in due parti: per prima cosa è necessario configurare l'HW, scegliendo dall'ambiente INCA quali variabili, sia di misura che di calibrazione, si vogliono controllare; dopodiché si può passare alla fase operativa in cui un sistema di automazione può agire sulle variabili di calibrazione (durante questa fase INCA diventa "osservatore" e riporta su schermo i valori che assumono le calibrazioni).

Per ricapitolare, INCA MCE permette un controllo remoto del sistema di calibrazione, esattamente come l' interfaccia ASAP3, ma questo non implica che le due strutture non possano lavorare contemporaneamente; infatti si può scegliere di concentrare l'utilizzo di MCE per la valutazione RT delle variabili di calibrazione e per la connessione con sistemi di automazione, lasciando ai due protocolli ASAM il compito di interfacciarsi con INCA per tutte le altre operazioni (registrazioni, configurazione HW, flash di SW, ecc) come già in parte avviene.

La vera differenza tra i due tipi di protocolli è che, se ASAP agisce su interfaccia host per la comunicazione, INCA-MCE agisce direttamente sull'HW ES910.3, riducendo drasticamente i tempi di comunicazione e rendendola più stabile.

Infine, per una completa descrizione di questo SW, si riportano alcune delle possibili applicazioni in cui l'altissima velocità di trasferimento dati può fornire vantaggi notevoli.

Times are approximate values	ASAP3 (RS232)	ASAM-MCD3-MC	INCA-MCE
Single Parameter (2 byte)	~50 ms	~7 ms	~0.3 ms
Five Parameters (5 x 2 byte)	~250 ms	~35 ms	~0.7 ms
Single 16 x 16 Map (512 byte)	~330 ms	~110 ms	~0.7 ms
Five 16 x 16 Maps (5 x 512 byte)	~1550 ms	~550 ms	~2.6 ms

* Utilizing a parallel ETK - ETK 7.1 – Connection to the ECU, Source: ETAS

Figura 4.11: Confronto tra i tempi di implementazione valori di mappa tra diversi protocolli di comunicazione

In particolare, facendo riferimento alla tabella di *figura 4.11*, per il download di una mappa 16×16 si osserva come la comunicazione via INCA-MCE risulti fino a 400 volte più veloce di una comunicazione con interfaccia ASAP3 (e connessione RS232) e 160 volte rispetto ad ASAM MCD-3 MC via ethernet.

4.5 Protocollo iLinkRT.

Molte delle informazioni riportate nelle prossime righe sono state estratte da documentazione fornita da ETAS insieme al modulo ES910.3, con una descrizione schematica del protocollo iLinkRT. Come descritto in precedenza, il protocollo XCP è estremamente configurabile grazie alle molteplici variabili di controllo disponibili, ma solo alcune sono utilizzate dalla specifica iLinkRT. Di seguito si è voluto descrivere questa configurazione. Le caratteristiche che definiscono il protocollo sono le seguenti:

- Il valore del time-out t1 è fissato a 1 secondo; questo time-out è quello che il dispositivo slave attende dopo aver ricevuto un qualsiasi comando a cui deve rispondere negativamente (quindi con un errore).
- La convenzione per l'interpretazione dei byte è Motorola (BYTE_ORDER = 1). Questo parametro indica con che ordine sono memorizzati i bit costituenti un byte di informazione. In particolare per Motorola si intende una rappresentazione ordina a partire dal dato più significativo per finire con quello meno significativo (si contrappone alla memorizzazione Intel caratterizzata da un ordine opposto).
- Non è supportata la comunicazione a blocchi, ma solo quella standard e quella interleaved. Il modello di comunicazione standard di XCP prevede che ogni pacchetto

di richiesta sia seguito da un pacchetto di risposta. Il protocollo XCP però mette a disposizione altre due tipologie di comunicazione: l'interleaved mode, in cui per velocizzare il trasferimento di dati, il master può mandare il comando k+1 prima di aver ottenuto una risposta al comando k-simo, e il block mode, utilizzato per salvare o copiare dalla memoria grandi quantità di dati, permette ad alcuni comandi particolari di ottenere molteplici pacchetti di risposta a fronte di un solo comando mandato. Si ricorda che il protocollo XCP esclude già l'utilizzo di queste due modalità contemporaneamente sullo stesso device. Per iLinkRT si è abilitata solo l'interleaved mode, con un numero massimo di pacchetti mandati consecutivamente pari a 5 (QUEUE_SIZE = 4 è il parametro che indica quanti pacchetti in più oltre al primo, il dispositivo riesce a gestire).

- Sono supportate solo le DAQ list statiche e non quelle dinamiche, e non possono essere modificate direttamente dal SW; il significato di quanto appena affermato è che, con il protocollo iLinkRT, il dispositivo slave non ha la possibilità di configurare le DAQ list autonomamente, cioè non può gestire le dimensioni e l'ordine delle variabili al fine di ottimizzare il trasferimento dati, opzione invece disponibile con il protocollo XCP. Le DAQ list possono essere modificate solo in ambiente INCA.

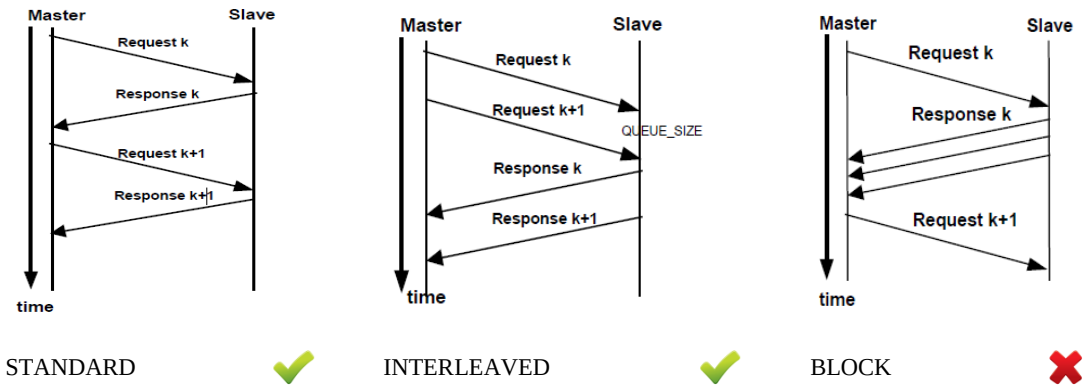


Figura 4.12: Strutture di comunicazione implementabili con XCP

- Esiste una relazione univoca tra DAQ list ed "eventi": infatti per ogni evento esiste una sola DAQ list; questa caratteristica è quella che effettivamente limita il numero di variabili della ECU che si possono leggere (le "measure" di INCA): in sé ogni DAQ

ist può contenere un massimo di 256 variabili e per ogni raster4 (questi sono gli eventi di cui parla il protocollo) esiste una sola DAQ list. Infatti in totale si possono osservare al massimo $(256 * n° \; di \; raster)$ variabili.

- L'Address Granularity (AG) è pari a 1; questo parametro indica in quale formato è espressa una variabile, in questo caso gli indirizzi di riferimento all'interno delle varie DAQ list. Per il protocollo XCP, AG può assumere 3 diversi valori che rappresentano il numero di byte utilizzati per rappresentare una variabile.

AG = 1	8 bit	BYTE
AG = 2	16 bit	WORD
AG = 4	32 bit	DWORD

Figura 4.13: Definizione dell'adress granularity.

- Per l'identificazione dei pacchetti dati, si utilizza il metodo "CTO Packet Code and absolute ODT number" sapendo che per ogni DAQ list esiste un solo ODT e che il timestamp ha dimensione 32 bit; questa informazione (che verrà approfondita più avanti) si riferisce a come è strutturata l'identificazione di ogni pacchetto scambiato tra master e slave. XCP mette a disposizione diverse possibilità, per la descrizione delle quali si rimanda alla documentazione ASAM.

Una particolarità dell'iLinkRT rispetto all'XCP standard prevede l'utilizzo di due porte di connessione UDP differenti, una per i comandi e le calibrazioni ed una dedicata completamente ed ininterrottamente all'acquisizione dati (DAQ list). I comandi implementati in questo protocollo possono essere divisi in due categorie: una selezione di alcuni comandi dello standard XCP e una serie di comandi User Defined, implementati da ETAS per ottimizzare la comunicazione. Alcuni comandi XCP standard (vedi tabella), sono stati implementati in questa trattazione, mentre altri non sono stati ritenuti fondamentali per il funzionamento del software.

[4] I raster di acquisizione sono dei "contenitori" all'interno dei quali vengono inserite, a scelta dell'utente, le variabili di centralina che si vogliono osservare; esistono diversi tipi di raster ognuno dei quali è caratterizzato da una differente frequenza di acquisizione dati.

COMANDO	PID	FUNZIONE
CONNECT	0xFF	Crea la connessione con il dispositivo slave
DISCONNECT	0xFE	Chiude la connessione con il dispositivo slave
GET_STATUS	0xFD	Ottiene dallo slave tutte le informazioni relative al suo stato
SYNCH	0xFC	Sincronizza l'esecuzione di un comando dopo un time-out
SHORT_UPLOAD	0xF4	Carica dei dati dallo slave al master (vers. ridotta)
SHORT_DOWNLOD	0xED	Scarica dal master verso lo slave (vers. ridotta)
SET_DAQ_PTR	0xE2	Gestisce il puntatore per operazioni acquisizione dati
GET_DAQ_LIST_MODE	0xDF	Informa sul modo di acquisizione in uso
START_STOP_DAQ_LIST	0xDE	Controlla l'acquisizione dati facendola partire o fermandola
START_STOP_SYNCH	0xDD	Controlla l'acquisizione dati in modo sincronizzato tra le DAQ list
GET_DAQ_EVENT_INFO	0xD7	Ottiene informazioni su uno specifico canale di eventi
GET_DAQ_RESOLUTION_INFO	0xD9	Ottiene informazioni generali sulla risoluzione di una DAQ list

Figura 4.14: Comandi standard di XCP utilizzabili anche su iLinkRT.

Per maggiori informazioni sui comandi appena elencati si rimanda alla descrizione del protocollo nel documento redatto da ASAM a cui proprio per questa applicazione l'Università di Bologna ha aderito all'associazione [7]. Per quanto riguarda i comandi implementati e utilizzati da ETAS essi sono riportati nella figura 4.15 e sono spiegati nel dettaglio di seguito. Si ricorda che per ogni User Defined Command il PID di riconoscimento stabilito dallo standard XCP è 0xF1, mentre varia il Subcommand da un comando all'altro secondo la tabella di *figura 4.15*.

COMANDO	SUB	FUNZIONE
GET_ALL_SERVER	01	Richiede allo slave alcuni dati utili relativi alla comunicazione
READ_DAQ_EXTENDED	02	Richiede alcune informazioni riguardo le variabili trasmesse nella DAQ list (Nome, Unità, ecc.)
CAL_DOWNLOAD_ADVANCED	04	Scarica dal master (PC) sullo slave (ECU) i valori di calibrazione specificati
READ_CAL_EXTENDED_V2	06	Richiede informazioni riguardo le variabili calibrabili (Nome, Unità di misura, Tipo, ecc.)
CAL_UPLOAD_ADVANCED_V2	07	Trasmette dallo slave (ECU) al master (PC) gli attuali valori di calibrazione di una determinata variabile specificata

Figura 4.15: User-Command specifici di iLinkRT.

Il comando GET_ALL_SERVER permette di ottenere la porta UDP da cui leggere le misurazioni, il nome del server e in generale le informazioni di comunicazione.

Di fondamentale importanza è invece il comando READ_DAQ_EXTENDED che permette al master di richiedere alcune informazioni addizionali riguardo le variabili trasmesse appartenenti a ogni DAQ list; ha lo stesso funzionamento del comando READ_DAQ (comando standard del protocollo XCP) e la struttura è quella nella tabella. Per poter funzionare correttamente questo comando deve essere preceduto sempre da un puntatore che indichi di quale valore si vogliono avere le informazioni. Questo puntatore viene regolato con il comando SET_DAQ_PTR, definito nello standard XCP. La risposta positiva al comando READ_DAQ_EXTENDED ha invece la seguente struttura:

Pos.	Type	Description
0	BYTE	PID = 0xFF (Risposta positiva)
1	BYTE	Address extension
2-5	4 BYTE	Address of DAQ element
6..n	STRING	Name of DAQ label
n+1..m	STRING	Unit of DAQ label
m+1..k	STRING	Description of DAQ label
k+1	BYTE	Data Type Body (DTY)5

Figura 4.16: Struttura della risposta al comando READ_DAQ_EXTENDED.

Il type STRING è utilizzato nel protocollo per comunicare testo attraverso array di byte ed è necessario per poter avere i valori un formato stringa un vi per trasformare i numeri in lettere utilizzando lo standard UTF-8.

Per poter avere informazioni riguardo le variabili di calibrazione su cui agire, esiste il comando READ_CAL_EXTENDED_V2 il cui funzionamento e la cui struttura sono molto simili a quelli del comando analogo, per la DAQ list, appena visto.

[5] Il Data Type Body (DTY) riporta il formato con cui è rappresentata una determinata variabile facendo riferimento alla legenda che segue:

DTY	Type	Description
0	Float32_IEEE	4-byte floating point number (IEEE format)
1	Float64_IEEE	8-byte floating point number (IEEE format)
2	UBYTE, BYTE	1-byte unsigned integer
3	SBYTE	1-byte signed integer
4	UWORD,WORD	2-byte unsigned integer
5	SWORD	2-byte signed integer
6	ULONG	4-byte unsigned integer
7	SLONG	4-byte signed integer

È necessario utilizzare almeno una volta questo comando prima di poter effettivamente leggere o modificare i valori delle variabili di calibrazione disponibili con i comandi che si vedranno più avanti. Si tenga conto inoltre, che il numero dell'elemento di cui si richiedono informazioni (byte 3 e 4), se non viene specificato si incrementa automaticamente ogni volta che viene dato il comando e si resetta quando il comando viene mandato verso un nuovo device; questo permette di leggere tutte le variabili presenti senza ricorrere ad un puntatore come si è visto per il comando READ_DAQ_EXTENDED. Per quanto riguarda il valore del Device ID, può essere lasciato pari a 0 quando si lavora con una sola centralina, ma quando si ha a che fare con due ECU (come nel caso di applicazioni trattate nell'elaborato) si può impostare a 0 o a 1 in funzione della ECU di cui si vogliono conoscere i parametri salvati su MCE.
Una volta ottenuta la lista delle variabili disponibili, si può procedere con la lettura del valore attuato in centralina utilizzando il comando CAL_UPLOAD_ADVANCED_V2 che permette di caricare un'area o la totalità della mappa, curva o scalare di calibrazione selezionati. Questo comando prevede due modalità: se si sceglie il "Mode 3" il pacchetto di dati trasmessi in risposta non contiene i valori degli assi di riferimento della mappa o curva, ma solo i valori della variabile, mentre per avere anche gli assi nel pacchetto dati di risposta si deve utilizzare il "Mode 4". Questa funzione ha lo scopo di alleggerire la comunicazione nel caso non siano necessari i valori delle ascisse ma, data l'elevata velocità di comunicazione disponibile si è scelto di utilizzare sempre il Mode 4. Per il resto il comando ha la struttura espressa nella tabella sotto:

Pos.	Type	Description	Valori ARRAY U8
0	BYTE	PID = 0xF1	241
1	BYTE	Subcommand 0x07	7
2	BYTE	Address extension	0
3-6	4 BYTE	Address	a a a a
7	BYTE	Mode	4
8,9	WORD	Start x (starts with 0)	x x
10,11	WORD	Start y (starts with 0)	y y
12,13	WORD	Size x	X X
14,15	WORD	Size y (1 for curves and 1-D arrays)	Y Y

Figura 4.17: Struttura del comando CAL_UPLOAD_ADVANCED_V2. Il numero di variabili nella colonna finale esprime come deve essere espresso il dato.

Per selezionare una determinata zona di una mappa si può agire sui valori di *Start x* e *Start y* per selezionare il punto della mappa da cui partire e *Size x* e *Size y* per scegliere la grandezza della zona; ovviamente per selezionare tutta la mappa basta porre *Start x* e

Start y pari a 0 e *Size x* e *Size y* pari alle dimensioni della mappa (valori ricavabili tra le informazioni ottenute con READ_CAL_EXTENDED_V2). Quando si ha a che fare con uno scalare basta impostare i valori di *Size* entrambi a 1.

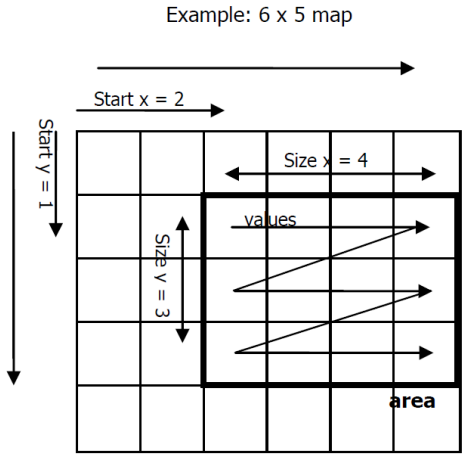


Figura 4.18: Esempio di lettura valori di una matrice, trasformata in array monodimensionale.

Si consideri inoltre che i dati in risposta sono trasmessi sotto forma di un vettore di byte anche nel caso si tratti di una matrice e sono orientati secondo le righe come rappresentato in *figura 4.18*. A seconda del Mode scelto si hanno due diverse strutture di risposta: rispettivamente per il Mode 3 si fa riferimento alla tabella di *figura 4.19*, mentre per il Mode 4 alla *figura 4.20*.

Pos.	Type	Description
0	BYTE	PID = 0xFF (Risposta positiva)
1..m	ELEMENTS	Z data elements (Size x * Size y values, row oriented)

Figura 4.19: Struttura di risposta al comando CAL_UPLOAD_ADVANCED_V2, secondo il MODE 3.

Pos.	Type	Description
0	BYTE	PID = 0xFF (Risposta positiva)
1..m	ELEMENTS	Z data elements (Size x * Size y values, row oriented)
m+1..n	ELEMENTS	X data elements (Size x)
n+1..o	ELEMENTS	Y data elements (Size y)

Figura 4.20: Tabella 1 Struttura di risposta al comando CAL_UPLOAD_ADVANCED_V2, secondo il MODE 4.

Come ripetuto più volte, con il protocollo iLinkRT i dati vengono trasmessi come array di byte e devono quindi essere convertiti. In particolare ogni valore numerico viene trasportato in 4 byte (32bit) secondo lo standard IEEE754 6.

Per poter scaricare in centralina dei valori di calibrazione scelti dall'utente, il protocollo iLinkRT prevede il comando CAL_DOWNLOAD_ADVACED (alternativo al comando SHORT_UPLOAD del protocollo XCP), per il quale sono supportati due metodi di funzionamento: il Flat Mode in cui tutti i punti di una variabile assumono il valore scelto (utile per gli scalari e per spianare una mappa o un vettore) e l'Area Mode, in cui si possono fissare valori diversi per i vari punti di una mappa o di una zona di essa. Nel caso dell'Area Mode i parametri da gestire sono analoghi a quelli del comando CAL_UPLOAD_ADVANCED_V2, infatti i valori da caricare sulla mappa sono riportati in un vettore, orientati per righe.

Si sarà notato che non sono stati descritti comandi in grado di acquisire misurazioni dalla DAQ list: questo aspetto infatti è demandato al comando START_STOP_DAQ_LIST del protocollo XCP standard che ha la doppia funzione di inizializzare la comunicazione dei valori di misura impostati nella DAQ list attraverso INCA-MCE e di fermarla nel momento in cui non è più necessaria. La struttura dei pacchetti di dati trasmessi su questo canale (si ricorda infatti che la capacità del modulo ES910.3 è quella di gestire contemporaneamente due flussi di informazioni attraverso due diverse porte UDP, una che scambia i comandi visti finora ed un'altra che trasmette ad una frequenza specifica e senza interruzioni i valori delle variabili della DAQ list) è diversa da quelli visti precedentemente ed è schematizzata in *figura 4.21*.

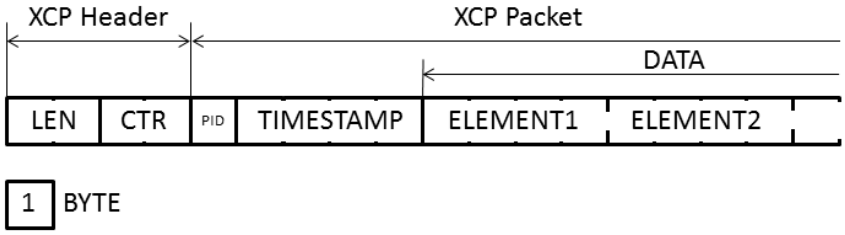


Figura 4.21: Rappresentazione di un pacchetto DAQ list.

[6] Lo standard IEEE per il calcolo in virgola mobile (IEEE 754) (ufficialmente: IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) o anche IEC 60559:1989, Binary floating-point arithmetic for microprocessor systems) è uno standard diffuso nel campo del calcolo automatico. Questo standard definisce il formato per la rappresentazione dei numeri in virgola mobile (compreso ±0 e i numeri denormalizzati; gli infiniti e i NaN, "not a number"), ed un set di operazioni effettuabili su questi.

Il pacchetto di dati è composto da una parte iniziale, *XCP Header*, che varia a seconda del layer di comunicazione che si utilizza: quella di *figura 4.21* è quella tipica della comunicazione via UDP/IP che è quella utilizzata effettivamente nel caso in questione. All'interno dell'header i primi due byte sono dedicati alla lunghezza del pacchetto XCP, mentre il terzo e quarto byte contengono un contatore dei pacchetti mandati fino a quel momento.

Come riportato inizialmente, per l'identificazione del pacchetto si utilizza il metodo "*CTO Packet Code and absolute ODT number*" con un timestamp di 32 bit: da *figura 4.21* si vede infatti che, per quanto riguarda *l'XCP Packet*, i primi 5 byte contengono un indicatore del tipo di informazione trasmessa (*PID*, Packet Identification Field) e un campo di 4 byte (*TIMESTAMP*) in cui è espresso il valore di clock del dispositivo slave, in questo caso l'ES910.3.

4.5.1 Implementazione dei comandi LV

Come anticipato, in questo paragrafo si vedrà come si è deciso di implementare in ambiente LV il protocollo descritto nel paragrafo precedente per permettere al SW di comunicare con la centralina. Per ogni comando si è deciso di creare due strutture: una per la formattazione del comando stesso, come richiesto dal protocollo, partendo dagli input scelti dall'utente e un'altra per l'interpretazione del pacchetto di risposta che il dispositivo slave manda al PC.

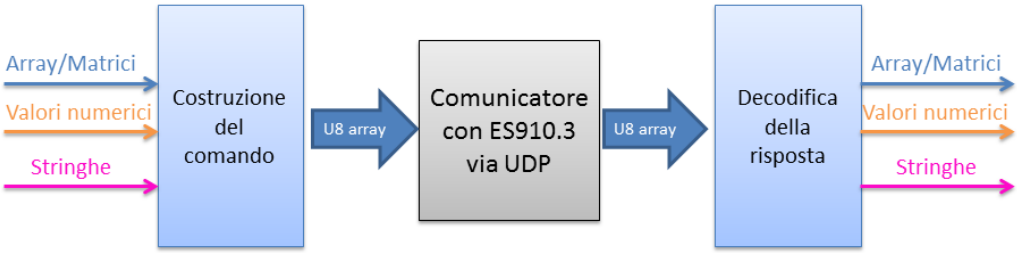


Figura 4.22: Struttura per l'implementazione di un comando.

Il cuore del sistema è un VI in cui è implementata la funzione di comunicazione via UDP che è descritto più avanti; prima si vedranno le strutture del comando e di decodifica della risposta (*figura 4.22*), che sono state denominate con il nome del comando che implementano, preceduto da una W per il vi che costruisce il comando e da una R per il vi che interpreta tale comando.

Utilizzando questa convenzione e seguendo l'ordine precedente, i primi VI che si analizzano sono W_READ_DAQ_EXTENDED e R_READ_DAQ_EXTENDED.

Come si è visto nel capitolo precedente, il comando READ_DAQ_EXTENDED non ha input controllabili dall'esterno quindi la struttura del relativo VI è piuttosto semplice ed è composta da due costanti: una che indica il comando ("F1") e una con il sottocomando ("2"). Il comando SET_DAQ_PTR è necessario per il funzionamento di READ_DAQ_EXTENDED. Esso è formattato esattamente come descritto nel protocollo XCP e in pratica permette di spostare il puntatore su una specifica variabile. Nel complesso la struttura che permette di ottenere informazioni sulle DAQ list presenti è quella di figura 4.23, dove i VI rappresentati sono quelli appena descritti.

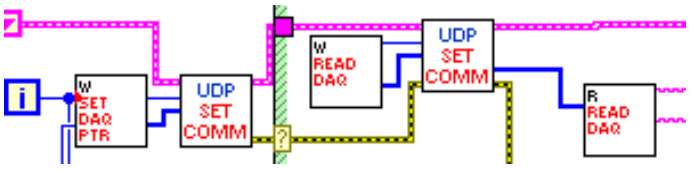


Figura 4.23: Richiesta informazioni sulle variabili nella DAQ; si noti la sequenza dei comandi SET_DAQ_PTR, poi la richiesta (W) e la risposta (R) di READ_DAQ_EXTENDED.

La logica di assemblaggio di un array di U8 applicata per il comando SET_DAQ_PTR si riscontra in praticamente tutti i VI di scrittura, ed è per questo che nel proseguo non si sono riportati quelli di tutti i comandi, ma solo i più significativi. Per esempio, in *figura 4.24* si osserva il comando READ_CAL_EXTENDED.

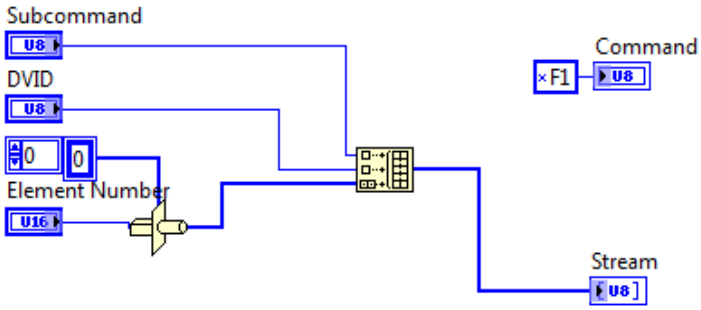


Figura 4.24: Struttura del VI R_READ_CAL_EXTENDED.

Una scelta importante nello sviluppo del SW è stata quella di inserire una gestione degli errori, per permettere sia allo sviluppatore sia all'utilizzatore finale di capire rapidamente in quale punto del codice si è generato un errore. In ogni VI di interpretazione di risposta si trova una case structure in cui viene generato un errore se la risposta del server alla richiesta del SW non è corretta.

Quanto appena detto vale anche per R_CAL_UPLOAD_ADVANCED; dove, oltre alla gestione dell'errore, vengono elaborati i dati della variabile di cui si sono richiesti i valori:

dopo che si è trasformato l'array di U8 in un array di double con l'apposito convertitore, si divide tale vettore separando i valori degli assi da quelli della variabile vera e propria. L'ultimo comando descritto è quello che permette di scaricare una calibrazione alla ECU, cioè CAL_DOWNLOAD_ADVANCED, ed in particolare il VI relativo alla generazione di un array di U8 contenente tutti i parametri di controllo disponibili.

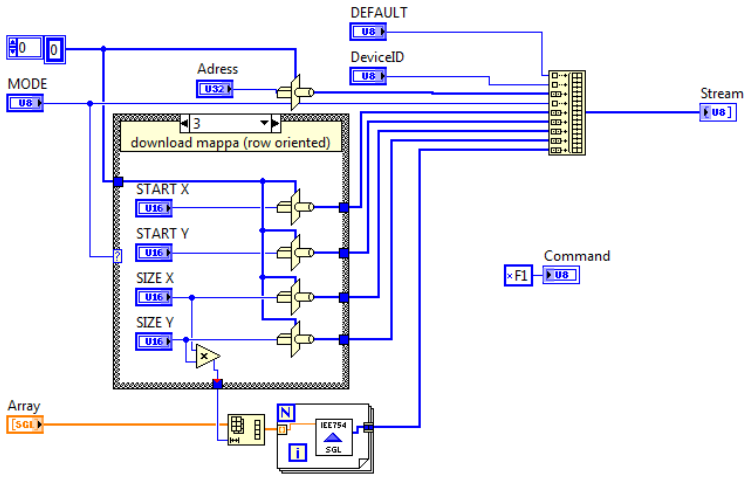


Figura 4.25:Struttura del VI W_CAL_DOWNLOAD_ADVANCED.

In *figura 4.25* è riportata in particolare l'implementazione del Mode 3 (quello che permette di variare una mappa punto per punto). La matrice di valori in ingresso subisce le operazioni inverse a quelle viste nel comando CAL_UPLOAD_ADVANCED: si ha prima un'operazione di reshape da matrice a vettore monodimensionale e poi i valori vengono convertiti da una rappresentazione decimale ad una rappresentazione in 4 byte di U8 nel VI appositamente creato.

4.5.2 Convertitore di stringe e di IEEE754

Data la struttura delle variabili STRING si è scelto di sviluppare un VI che permettesse una rapida conversione di un vettore di valori numerici in un vettore di lettere. Oltre alla semplice conversione numero-lettera secondo lo standard ASCII si è implementato un controllo sul numero di lettere tradotte: la traduzione della stringa viene effettuata finché non si trova il valore "\0" che indica la fine della parola e confrontando la lunghezza della stringa ottenuta con il valore riportato in un byte in testa al vettore di U8 si può verificare facilmente se la traduzione è stata effettuata correttamente.

Successivamente si vuole rapidamente spiegare lo standard IEEE754 [8] che il protocollo iLinkRT utilizza per rappresentare i valori numerici che trasmette. Si è deciso di creare due VI in grado di trasformare numeri dalla formattazione single-precision floating point

(SGL) ad una formattazione standard IEEE e viceversa, al fine di semplificare la gestione dei dati e le operazioni tra di essi. Lo standard IEEE 754 per numeri a precisione singola utilizza 32 bit per la rappresentazione, che vengono divisi in 3 parti (1 bit di Segno, 8 bit di Esponente e i rimanenti 23 bit di Mantissa) e ordinanti come raffigurato di seguito. Si tenga presente che per il protocollo iLinkRT/XCP un valore espresso in 32 bit viene trasmesso come un array di 4 BYTE.

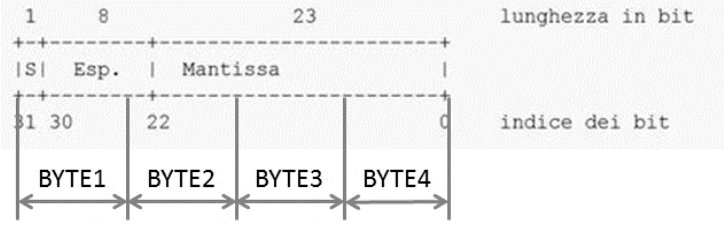


Figura 4.26: Protocollo IEEE754

I 3 elementi con cui viene rappresentato il numero sono riconducibili alla seguente formula:

$$(-1)^{s} \times 2^{E} \times M$$

Dove:

- S specifica il *Segno,* 0 per numeri positivi e 1 per i negativi.
- E rappresenta l'*Esponente* del numero; si consideri che negli 8 bit, per una rappresentazione più efficace di numeri molto grandi e molto piccoli, non viene immagazzinato però il valore E, ma un valore *e* ottenuto aggiungendo ad E un bias, cioè un valore costante, che rende sempre positivo il valore espresso. Il bias è pari a 127.

$$e = E + 127$$

- M è la Mantissa ed è un valore compreso tra 1 e 2, ma nel campo m dedicato a questa informazione, vengono inseriti solo i numeri dopo la virgola, omettendo il valore 1 che viene per questo motivo definito come bit nascosto. Il campo m che occupa gli ultimi 23 bit è dato dalla seguente formula:

$$M = 1, m$$

E nel caso il valore di m non occupi tutti i 23 bit disponibili, gli ultimi vengo riempiti di zeri. Nelle figure seguenti si può osservare come è stato implementato l'algoritmo appena spiegato.

4.5.3 Creazione della comunicazione UDP/IP in LV

Il protocollo UDP è una metodologia di comunicazione via ethernet. Risulta molto flessibile e non troppo specifica per quello che riguarda il controllo della comunicazione; non si ha infatti una verifica che i pacchetti inviati arrivino a destinazione e vi arrivino completi e ordinati. Questa caratteristica permette una grande flessibilità che caratterizza il protocollo UDP, che si esprime nella mancata necessità di avere un collegamento esplicito tra client e server per poter comunicare; infatti basta impostare l'indirizzo IP verso cui comunicare e la porta UDP (di chi sta comunicando) da cui partirà il messaggio. Per quanto riguarda il server (il modulo ES910.3), dall'interfaccia di INCA si può impostare l'indirizzo IP che assumerà la scheda di rete e la porta attraverso cui comunicherà (per il caso riportato si sono scelti l'indirizzo 169.254.1.100 e la porta *8091*). Il primo step per creare la comunicazione è l'invio della richiesta di connessione dal client (TAC) verso il server (ES910.3) utilizzando il comando CONNECT. Con questa operazione il server acquisisce l'indirizzo IP e la porta UDP del client ed è in grado di comunicare con esso, rispondendo affermativamente al comando di connessione. Prima di poter mandare il comando CONNECT, si è aperta una porta UDP generica (scelta automaticamente dal sistema tra quelle disponibili, oppure in *figura 4.27* si è scelta la 1111) con la funzione *Open UDP Port*.

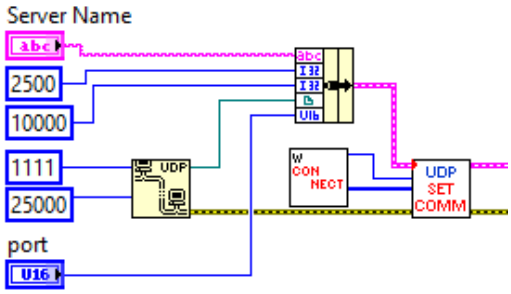


Figura 4.27: Parametri della connessione: Server Name riceve dall'utente l'indirizzo IP del Server.

2500 indica il timeout dato alla porta UDP per l'attesa di un comando, 10000 è la lunghezza massima in byte del pacchetto trasmesso o ricevuto, 1111 è il numero della porta UDP da cui si comunica, 25000 è un altro timeout ed infine attraverso port viene inserito dall'esterno il numero della porta da cui il server sta comunicando.

Il comando CONNECT, implementato come array di BYTE, entra nel subVI UDP_SET_COMMUNICATION, la cui struttura è riportata nella *figura 4.28*.

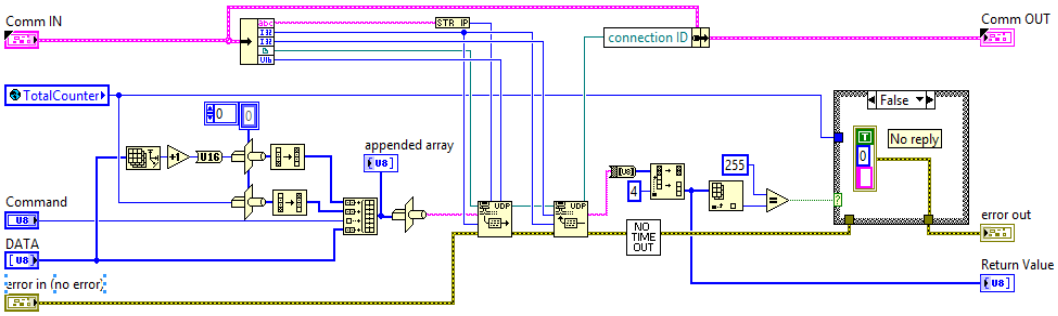


Figura 4.28: Logiche del SubVI UDP_SET_COMMUNICATION.

Soffermandosi rapidamente su questo subVI, si nota come, dopo aver assemblato un array ordinato con tutti i valori richiesti dal protocollo, esso viene mandato dal client verso il server attraverso il comando UDP Write; dopodiché con il comando UDP Read si possa leggere l'array di risposta del server. Si noti inoltre che viene eliminato, nel subVI NO_TIME_OUT, l'eventuale errore di time-out dovuto all'asincronia fra il tempo in cui viene comandata la lettura dalla rete e l'effettivo invio del pacchetto. Inoltre (come già si è visto in altri punti del SW) viene fatto un controllo sulla risposta, così che ogni volta che non si ha una risposta positiva (primo PID uguale a 255), dal subVI viene prodotto un errore a cui fare riferimento. Si è anche implementato un ulteriore controllo basato sul confronto dei counter dei messaggi mandati e ricevuti, che permette di verificare se qualche pacchetto di informazioni mandato dalla ECU verso il SW è stato perso: nel caso si rilevi un evento del genere viene riportato un errore dedicato.

5 Controllo retroazione anticipo:

5.1 Introduzione

L‘effetto dell’anticipo sulla prestazione (pressione media indicata PMI, pressione media effettiva PME) è rappresentabile attraverso l’ausilio della curva a ombrello: ogni punto sul piano PMI-SA è ottenuto come media di diversi (anche alcune centinaia) cicli, per svincolarsi il più possibile dall’effetto della dispersione ciclica. Nei motori ad accensione comandata, la leva di controllo più utilizzata è l’anticipo, anche se sarebbe possibile impiegare, ad esempio, lambda ed egr [9].

Un modo di valutare l’effetto di SA sulla combustione è dunque osservare gli andamenti di MFB10, MFB90, e il picco di rilascio calore al variare di SA e a parità di altre condizioni. Si tenga presente che l’assenza di determinismo nelle combustioni, porta a variazioni ‘statistiche’, cioè gli effetti vanno osservati sulle distribuzioni (media e deviazione standard dei parametri rilevanti). Per descrivere meglio questi concetti, si dà la definizione di funzione di Wiebe. Esso è un modello fenomenologico per rappresentare il processo di combustione: si descrive l’andamento della frazione di massa (di combustibile) bruciata, in funzione dell’angolo e di 4 ulteriori parametri.

$$x_b = 1 - e^{-a\left(\frac{\theta-\theta_{SOC}}{\Delta\theta}\right)^{m+1}}$$

Dove x_b è la percentuale di combustibile bruciato (Mass Fraction Burned MFB),a, la completezza della combustione, θ_{SOC} rappresenta la fase di combustione e $\Delta\theta$ la durata di combustione. A livello operativo, in fase di identificazione vengono fatti corrispondere a eventi riconoscibili, ad esempio il raggiungimento del 5% e del 95% dell’energia liberata (se a=3). Per quanto riguarda m, esso è legato alla velocità di combustione. Di solito xb viene considerato coincidente con il rapporto tra energia liberata e energia complessivamente introdotta col combustibile.

Sintetizzando, l’andamento dei parametri caratteristici della combustione può essere descritto attraverso l’MFB50, che quindi è un indicatore molto efficace della combustione nel suo complesso, col fatto che tiene conto della fase, della durata e della forma di essa. L’effetto della dispersione ciclica è evidente quando si analizzano I cicli nel piano PMI-MFB50. Tutte le dispersioni relative a un dato anticipo portano allo stesso andamento

della PMI in funzione dell'MFB50: segno che per quella fase il compromesso tra perdite di calore a parete (influenza su energia disponibile) e efficienza di trasformazione dell'energia termica in energia meccanica è ottimale. Spesso la fase ottimale viene indicata in 8°ATDC, in realtà può cambiare al variare delle condizioni operative.

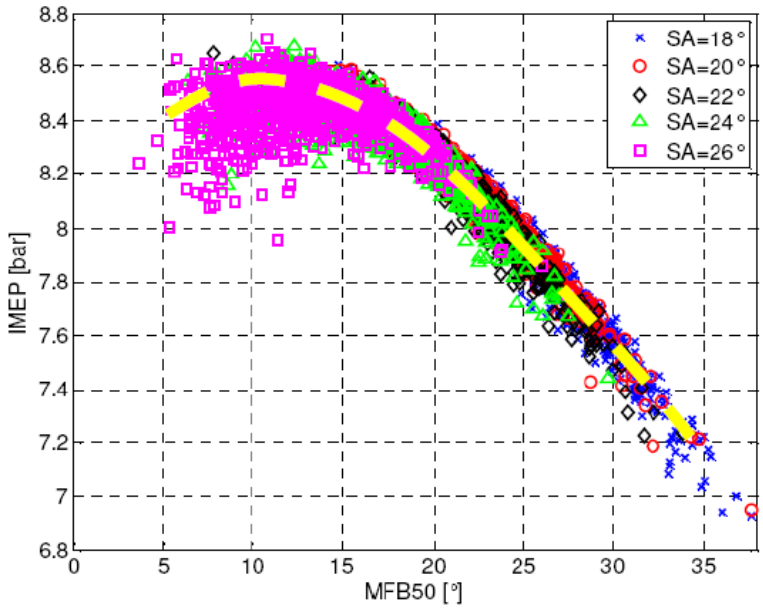


Figura 5.1: Esempio andamento PMI in funzione MFB50.

È evidente che la dispersione ciclica deteriora le prestazioni, impedendo il raggiungimento di una condizione ottimale sistematica, e consentendo solamente di ottenere la dispersione ottimale. Il deterioramento è legato alla impossibilità di ottenere una fase di combustione sistematica a fronte di un SA costante. La dispersione impone una calibrazione fatta su database che contengono molti cicli (di solito da 200 a 500), per poter stabilire con certezza I valori di SA ottimale (determinazione curva a ombrello).
Solitamente, I valori di SA ottimale vengono 'mappati' in ECU (scritti in EPROM), in funzione della condizione operativa (rpm, pcoll). A rigore, SA ottimale dipende anche da lambda e dall'EGR (oltre che da altri fattori come tipologia di combustibile, temperatura e umidità dell'aria, ecc.), quindi un controllo in retroazione sarebbe più efficace. Quest'approccio di controllo in retroazione, anche se difficile da applicare in vettura a causa del costo e della fragilità di sensori di pressione in camera, può essere sviluppato in sala prove. Ci sono diverse tipologie di approccio che sono applicabili. Nelle applicazioni sviluppate ci si è concentrati in particolare su due approcci.

1. Metodo a “denti di sega”: in pratica si tratta di un controllo della fase di combustione con limitazione knock. Il metodo garantisce un livello di detonazione mediamente accettabile, ma non ottimizza veramente la fase di combustione.

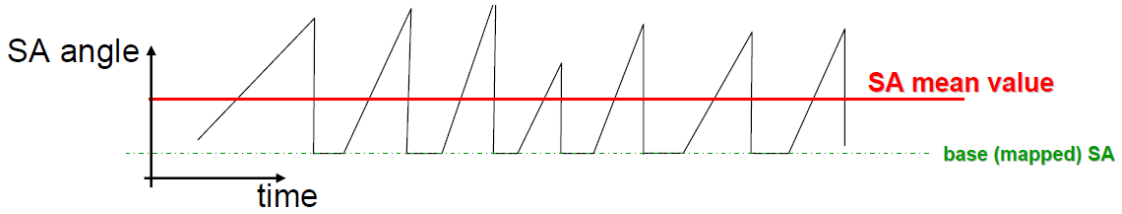


Figura 5.2: Andamento controllo a dente di sega

2. Un criterio possibile per controllare la fase di combustione in retroazione (con disponibile il segnale di pressione cilindro, o la stima della fase di combustione) è quello di fare in modo che, con l’anticipo considerato, la tangente alla distribuzione PMI-MFB50 sia orizzontale. Il controllore ottimizza il funzionamento del motore, portando stabilmente SA sul valore ottimale. Da notare che questo valore è diverso da cilindro a cilindro.

5.2 Controllo anticipo a limite di detonazione.

La parte descritta di seguito riguarda la definizione di strategie di ottimizzazione automatizzata delle prestazioni motore, mantenendo al contempo il motore all’interno di soglie accettabili di valori rilevati di detonazione.

L’obiettivo è quello di acquisire dati dai sistemi di analisi combustione (*figura 5.3* AVL indiset) via CAN, elaborare una strategia di controllo e comunicare le nuove calibrazioni alla ECU.

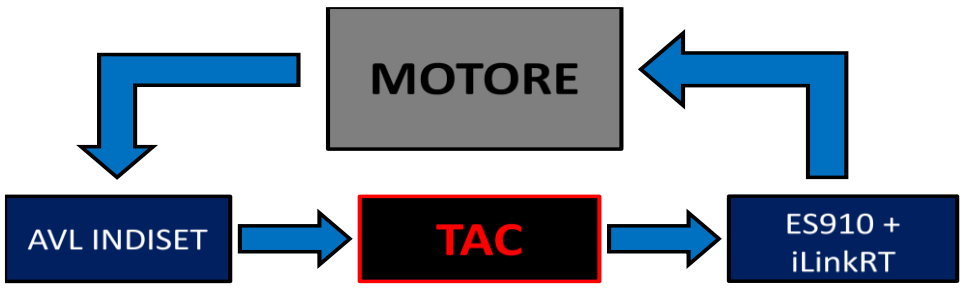


Figura 5.3: Diagramma a blocchi della strategia implementata.

Come indice di detonazione, fra quelli disponibili, si è deciso inizialmente di utilizzare il MAPO (Maximum Amplitude of Pressure Oscillations) creando un controllo che, se da un lato modifica i valori di anticipo, dall'altro è "pronto" a intervenire in caso di superamento delle soglie limite di tale indice. Un requisito fondamentale per un controllo efficace e sicuro dell'anticipo, che dipende dalla natura del fenomeno della detonazione, è infatti quello di avere dinamiche di intervento molto "rapide". Inizialmente l'algoritmo è stato sviluppato su un solo cilindro, per poi essere implementato su più cilindri (nel caso specifico fino a 6). Se i valori indicating di riferimento per ogni cilindro sono quindi il MAPO, l'MFB50 e la PMI, il nome della variabile di centralina che permette una variazione dell'anticipo è ZWINDn (n-esimo cilindro). Come riportato da specifica BOSCH relativa al software ECU, il controllo dell'anticipo è regolato da numerosissime variabili e funzioni, ma si è deciso di utilizzare i parametri ZWIND per due diverse ragioni: si è in grado di agire singolarmente su ogni cilindro e le modifiche intervengono nel calcolo a valle della maggior parte delle altre correzioni dovute principalmente al tipo di combustibile, alla modalità di iniezione e alla temperatura del liquido di raffreddamento, rendendo meno complessa la gestione del controllo. La calibrazione ZWINDn è una correzione di anticipo (quindi non rappresenta il valore effettivo che esso assumerà, ma un offset sul valore elaborato da altre funzioni), è in forma di mappa (cioè è funzione di due dati in ingresso, in questo caso il regime del motore, nmot, ed il carico, rappresentato dalla percentuale di aria relativa, rl) e la sua numerazione non è data dalla posizione geometrica, ma è riferita all'ordine di scoppio. Per quanto riguarda la struttura logica implementata, la si può riassumere con lo schema a blocchi riportato nella *figura 5.4*. Una volta definite soglie oltre cui la detonazione non è accettabile, si vanno a confrontare tali valori con quelli del MAPO letto via CAN e del MAPO percentile elaborato su un buffer di valori predefinito; è così possibile ottenere una differenza che se negativa (ovvero se il MAPO misurato è oltre la soglia) viene moltiplicata per un guadagno K_TH o K_%, che viene a sua volta ottenuto da una mappa impostata come funzione del regime motore e del cilindro in cui si è verificata la detonazione.

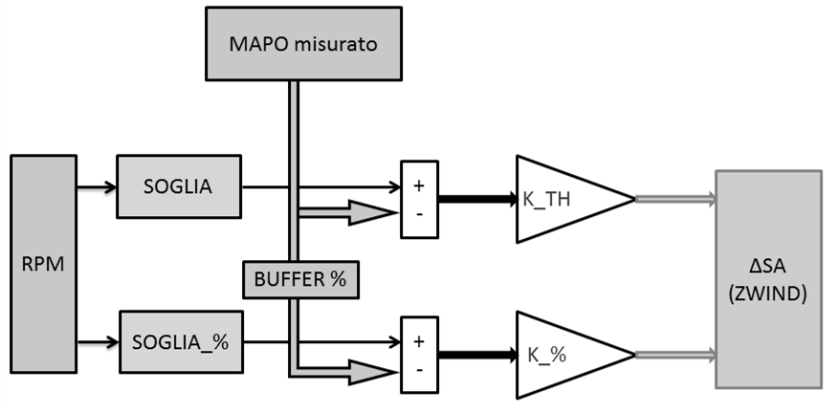


Figura 5.4: Logica di variazione anticipo.

L'output di questa logica di controllo è una variazione di anticipo di accensione del cilindro in cui si è rilevato un superamento dei limiti. Per le prime prove si è deciso di utilizzare dei guadagni costanti per gli eventi sopra-soglia, lasciando per una fase successiva la possibilità di rendere più accessibili all'utente questi valori: in particolare si sono scelti, dopo alcune prove, un valore di K_TH = 5 e un valore di K_% = 25.

Da un punto di vista della programmazione, la riduzione di anticipo calcolata precedentemente viene comandata alla ECU attraverso il subVI DOWNLOAD_CAL_BY_NAME, all'interno di cui si ritrovano i comandi di iLinkRT descritti in precedenza. Nella *figura 5.4* in particolare, viene 'spianata' tutta la mappa di ZWIND1 al valore di correzione in ingresso.

La funzione del comando UPLOAD_CAL_BY_NAME è quella di fornire l'indirizzo della variabile che si vuole modificare (ZWIND1) al comando di download; inoltre si può già osservare in questa figura come venga calcolata la variazione di anticipo per ciclo (dSA/Ciclo), valore che sarà utilizzato nella fase successiva di rientro post-correzione. Il valore che assume ZWIND1 in centralina è effettivamente quello richiesto dal SW; infatti, (come si vede nella *figura 5.5*) anche attraverso l'interfaccia (experiment) di INCA, la mappa di anticipo è completamente spianata al valore implementato dal TAC, in questo caso -9 gradi di anticipo.

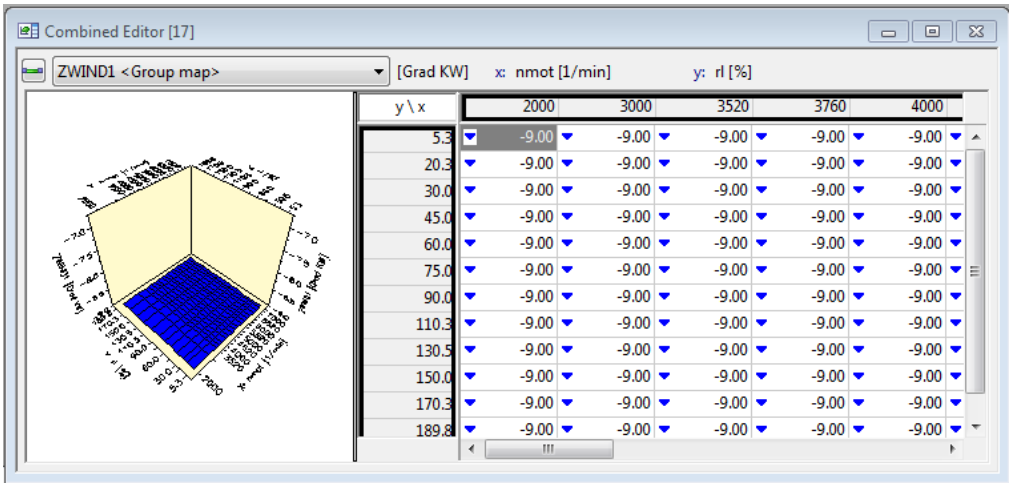


Figura 5.5: Mappa di ZWIND1 spianata dal controllo anticipo, così come è visualizzata in INCA.

Oltre alla possibilità di spianare la mappa ad un valore costante, è stata implementata anche una logica in grado di correggere con un offset la mappa di variazione anticipo ZWIND1, mantenendo quindi le differenze tra un punto operativo e l'altro. Per l'implementazione dell'offset si utilizza la funzione "area" del comando CAL_DOWNLOAD. Inoltre, per poter utilizzare questa modalità è necessario conoscere le dimensioni della mappa, che sono ottenute come risposta del comando UPLOAD_CAL_BY_NAME, da cui si ricava anche il valore delle calibrazioni implementate in centralina a cui viene sommato il valore di offset calcolato nelle fasi precedenti.

5.2.1 Logiche di rientro dell'anticipo attuato dopo correzione.

L'implementazione SW non si limita al controllo dell'anticipo al fine di mettere in sicurezza la fase di calibrazione da eventi di detonazione, ma prevede anche l'implementazione di logiche per il rientro della correzione una volta che questa fosse stata attuata.

L'obbiettivo infatti è di creare un sistema in grado di reagire in modo "intelligente" al fenomeno: una volta decurtato anticipo e portato il motore in una zona sicura, si vuole riportare tale valore a vicino a quello di prova limitando la correzione effettuata, ma rimanendo in sicurezza (ovvero continuando ad avere un livello di detonazione accettabile), dopo aver agito su altri parametri che influenzano il knock.

Come è noto infatti, molti sono i parametri che influenzano il fenomeno della detonazione oltre all'anticipo di accensione (pressione di boost, temperatura aria, lambda, raffreddamento motore, ecc.), ma solo controllando l'anticipo si riesce ad annullare la

detonazione già al ciclo motore successivo a quello in cui si è verificata, mentre le dinamiche motore che portano gli altri parametri a influenzare questo effetto rendono una eventuale modifica effettiva solo dopo alcuni cicli. Se inizialmente si era pensato di controllare questa fase lasciando scegliere all'utente il numero di cicli di rientro, cioè il numero di cicli motore al termine del quale il valore della riduzione ZWIND sarebbe dovuto essere a zero, in una seconda fase si è optato per utilizzare la pendenza del rientro (ovvero il rapporto tra la variazione di anticipo totale e il numero di cicli di rientro) per regolare la fase di recupero (vedi *figura 5.6*). In termini di utilizzo non esiste una grande differenza tra questi due parametri, infatti la pendenza è definibile nel modo seguente:

$$\text{Pendenza} = \frac{\text{Correzione tot}}{\text{N° cicli di rientro}}$$

si è scelto però questo parametro perché permette di svincolare la velocità di rientro dalla grandezza della correzione, rendendola quindi costante e regolabile.

Per rendere più efficiente questa procedura, ottimizzando il numero di pacchetti da inviare alla ECU, si è deciso di aggiornare il valore di anticipo solo quando necessario, considerando gli step minimi di variazione di ZWIND in centralina: questo valore è impostato nel database .a2l ed è leggibile nella descrizione della variabile su INCA.

Per esempio, considerando che la risoluzione di ZWIND è 0.75° di SA, tutti i valori compresi in un intorno di +/- 0.375 attorno a un valore non vengono recepiti: di conseguenza perde significato mandare più valori compresi all'interno dello stesso intervallo. In *figura 5.6* si vede come il SW richieda valori in modo continuo (si tratta ovviamente di un insieme discreto di valori ma con step di variazione talmente più piccoli rispetto a quelli di INCA da sembrare continuo), mentre in centralina i valori attuati hanno un andamento a gradini.

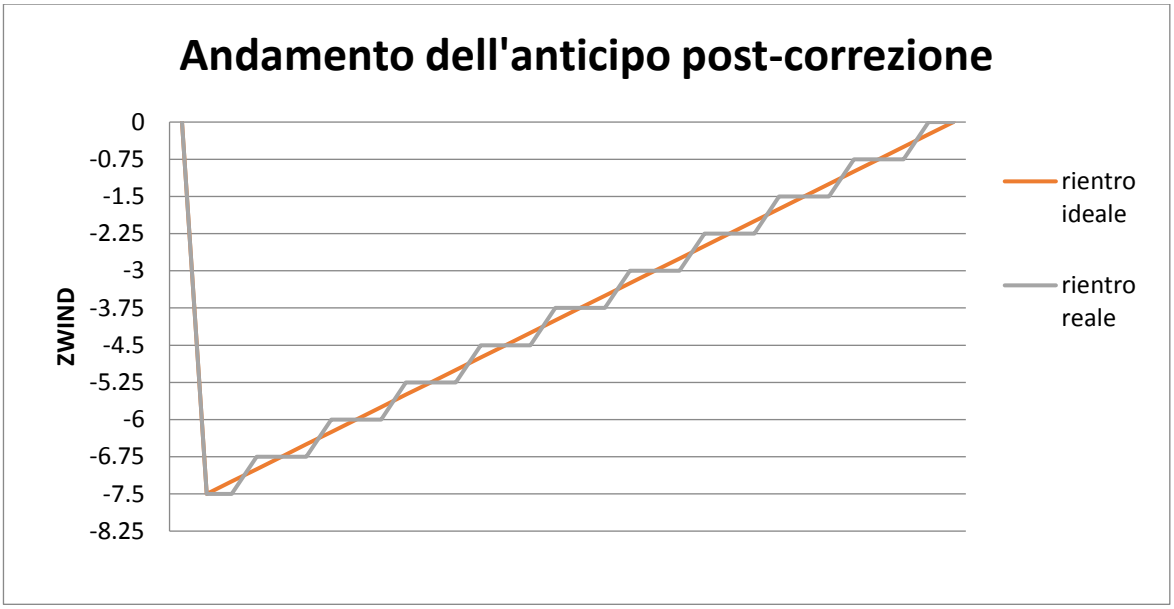


Figura 5.6: Andamento di ZWIND continuo e discreto; quello attuato in centralina è quello in verde con gradini di 0,75 gradi.

La logica che fa in modo che non vengano mandati comandi ridondanti alla centralina è stata implementata con una state machine. Per questa operazione, gli stadi principali sono 3:

- PrimoStep: in cui viene effettuata la decurtazione di anticipo con le logiche viste in precedenza.
- Wait: in cui il SW non manda alcun comando alla ECU e che viene ripetuto un numero di volte variabile in funzione della pendenza di rientro scelta. È il cuore dell'algoritmo in quanto valuta come rientrare da un evento detonante, infatti, l'esecuzione dei vari stati è funzione di un contatore utilizzato proprio per controllare la pendenza di rientro, da cui esso è effettivamente valutato.
- Execution: a sua volta diviso in una fase di lettura del valore attuato in centralina ed una fase di scrittura del nuovo valore da applicare.

Viene mostrato in figura il subVI COERCE FOR INCA, in cui entra la correzione proporzionale al valore di MAPO ottenuto, che raramente è un multiplo dello step minimo 0.75; in questa fase si arrotonda quindi la correzione al multiplo superiore dello step, in modo da richiedere un valore che è effettivamente quello che verrà applicato (prima di questa implementazione non si riusciva a sapere se il valore attuato fosse arrotondato per difetto o per eccesso e spesso il rientro non avveniva in modo completo).

Insieme al valore della correzione reale, in tale sottosistema vengono calcolati due parametri fondamentali per gestire la fase di rientro: uno (Ncicli) è il contatore già descritto che indica effettivamente il numero di cicli di rientro, l'altro (denominato NcicliACT) rappresenta il numero di cicli motore in cui si avrà un singolo step di rientro (per esempio, per una pendenza di 0.25 gradi/ciclo, la correzione minima di 0,75 dovrà avvenire ogni 3 cicli, quindi NcicliACT=3). Di seguito si osserverà il comportamento del SW per un singolo cilindro considerando che ogni operazione viene eseguita in parallelo su ogni cilindro.

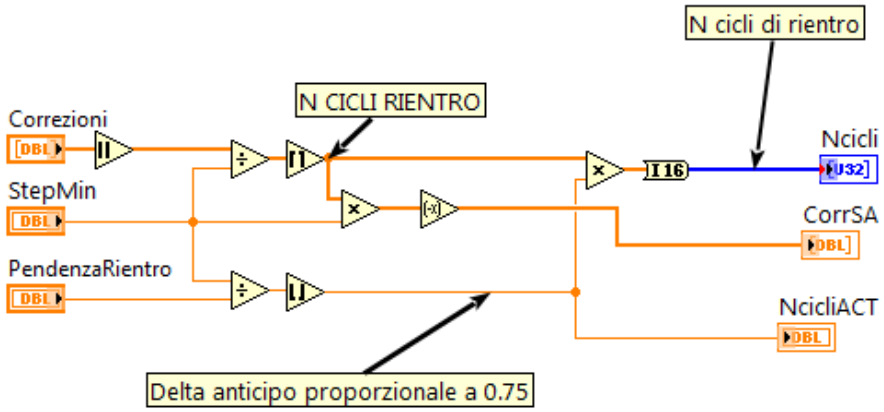


Figura 5.7: Logica del VI COERCE_FOR_INCA. In uscita si riconoscono i cicli di rientro, la correzione normalizzata e i cicli per step.

Durante il funzionamento, la state-machine si trova nello stato di WAIT, in cui non viene scambiata alcuna informazione con la centralina. Appena viene rilevato un evento di detonazione lo stato della struttura passa a PRIMOSTEP, a prescindere dallo stato in cui si trova; infatti la correzione dell'anticipo che avviene in questo stato ha una priorità maggiore di qualsiasi altra azione. In questo stato vengono eseguite diverse azioni: per prima cosa si leggono i valori attuati in centralina, poi viene applicato l'offset correttivo (calcolato come visto in precedenza) ed infine vengono scaricati nella ECU i valori corretti. In uscita da questo stato si osserva che viene richiesto di tornare nello stato di WAIT; il valore del contatore viene poi aggiornato al valore di Ncicli calcolato in COERCE. In realtà questo avviene solo quando la correzione è attuata su uno stato di "riposo" cioè ogni eventuale correzione precedente su tale cilindro è stata annullata; infatti, se l'evento di detonazione si ha durante la fase di rientro di una correzione precedente, è necessario tener memoria di quanto ancora non si è recuperato, per poter poi azzerare tutte le correzioni. Per fare ciò si è scelto di procedere sommando al numero

di cicli di rientro in ingresso il valore del contatore al ciclo precedente, che sarà maggiore di 0 se non si è ancora rientrati dalla correzione precedente.
Nel complesso quindi il nuovo valore del contatore in uscita dallo stato PRIMOSTEP è il seguente:

$$\text{Counter} = \begin{cases} \text{Ncicli}, & \text{riposo} \\ \text{Ncicli} + \text{Counter\_arr}, & \text{rientro da corr} \end{cases}$$

Dove Counter_arr rappresenta l'arrotondamento di Counter al primo multiplo di NcicliACT. Una volta eseguito PRIMOSTEP, lo stato torna su WAIT come richiesto: il controllo è lasciato nuovamente al valore assunto dal contatore; esso avrà un valore positivo, in cui sono possibili due ulteriori stati, anch'essi governati dal valore del contatore. Per prima cosa viene scalato il contatore, dopodiché si presentano due casi: se il valore ottenuto è un multiplo di NcicliACT, viene eseguito il comando di lettura dei valori di ZWIND in centralina, e viene richiesto lo stato EXECUTION per il ciclo successivo, altrimenti la variabile di stato rimane impostata su WAIT.
La presenza di questo stato di attesa è ciò che effettivamente regola la pendenza di rientro dell'anticipo, poiché al diminuire di essa aumenterà il numero di cicli motore in cui il SW si troverà in questo stato. Lo stato EXECUTION è quello in cui viene effettuato lo step di +0,75 gradi, infatti viene sommato tale offset al valore letto al ciclo precedente nello stato di WAIT. All'uscita di questo stato si torna poi al WAIT, in cui viene scalato ciclo per ciclo il contatore, finché non si arriva ad un altro multiplo di NcicliACT e si può procedere con un ulteriore step di rientro.
In particolare, la complessità nell'implementazione di questa logica consiste nella necessità di ottenere un risultato stabile per tutte le combinazioni di eventi. Come detto in precedenza, il controllo avviene contemporaneamente sia sulle soglie di MAPO istantaneo che su quelle di MAPO percentile. Questo vuole dire che sono diverse le combinazioni possibili. Il primo caso che ha richiesto un intervento da un punto di vista logico è stata la permanenza di valori sopra soglia di MAPO percentile: in questa occasione infatti si ha un superamento della soglia che dura per diversi cicli motore consecutivi (evento piuttosto frequente proprio per come è definito tale parametro) e, per come è stata impostata la macchina a stati appena spiegata, ogni volta che viene letta una correzione negativa si entra in PRIMOSTEP applicando tale correzione. Nel caso di MAPO percentile però si avrebbe una deriva dell'anticipo che andrebbe a ridursi sempre più. Per evitare che ciò accada si è inserito un confronto tra la correzione richiesta ad un

ciclo e quella richiesta al ciclo precedente: se tali correzioni sono identiche allora la seconda volta essa non verrà ripetuta.

Oltre a questo caso si sono fatte diverse prove con superamenti di soglia simultanei per entrambi i MAPO analizzati: nello specifico, si è provato sia un evento di MAPO istantaneo durante una serie di MAPO percentile sia un evento di MAPO percentile durante la fase di rientro da un MAPO istantaneo.

Nei grafici delle *figure 5.8 e 5.9* sono raffigurati questi due casi: in particolare in blu vediamo l'andamento del valore di ZWIND, mentre in rosso è riportato il valore del contatore. Si consideri che solo avendo ben chiaro l'obbiettivo rappresentato in questi grafici e con questi valori si è riusciti ad implementare una strategia flessibile e stabile come quella descritta poco sopra.

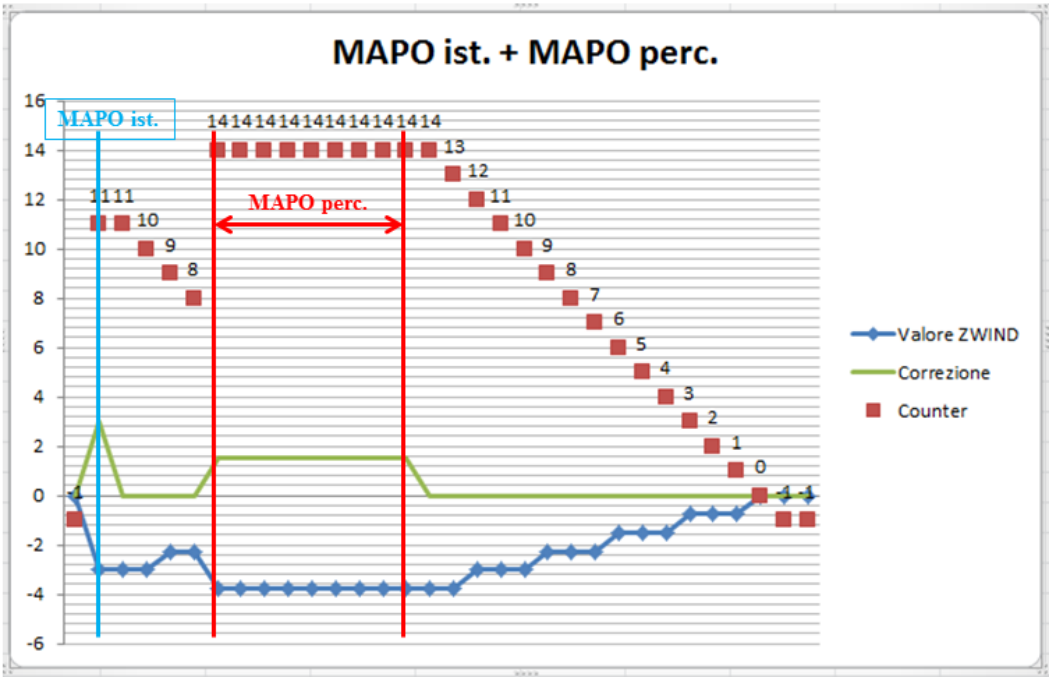


Figura 5.8: Evento di MAPO istantaneo seguito da un evento di MAPO percentile. Si osservi l'andamento del valore del counter che inizia a calare solo una volta che si è annullata la correzione del MAPO percentile.

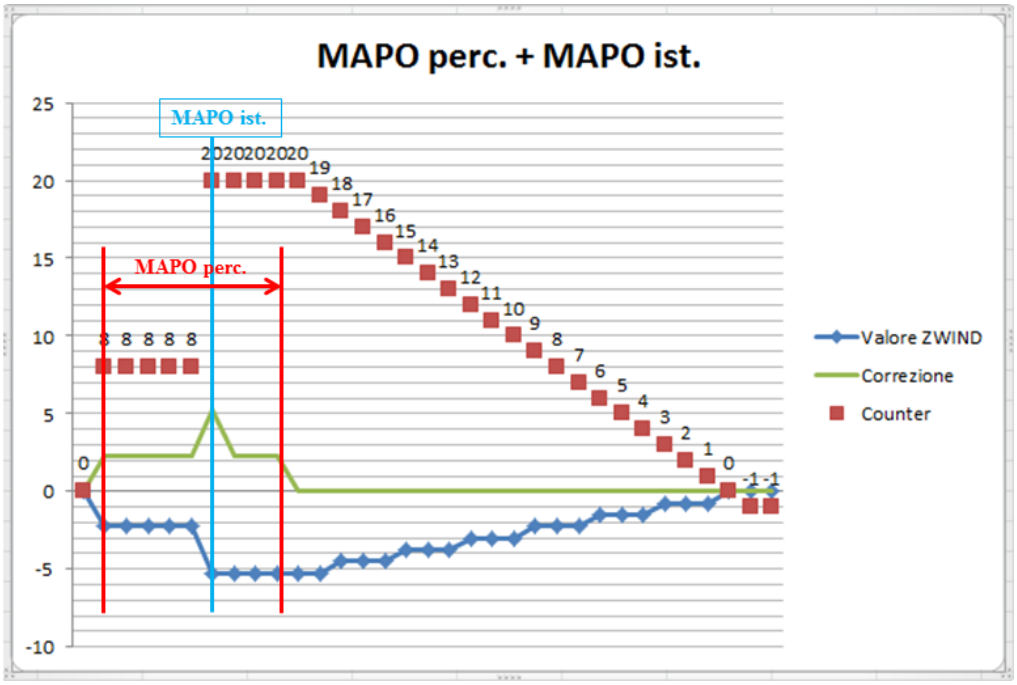


Figura 5.9: In questo caso viene rilevato un evento sopra-soglia di MAPO istantaneo durante una fase di segnale di MAPO percentile anch'esso sopra-soglia. per definizione la fase di rientro da entrambe le correzioni non può iniziare finché la correzione non ha valore nullo.

6 Automatizzazione test

6.1 Introduzione

Una delle applicazioni più importanti che sono state realizzate utilizzando le metodologie descritte e sviluppate in precedenza riguarda la realizzazione di un'applicazione in grado di automatizzare le prove in sala prova permettendo la comunicazione fra sistemi differenti. Per dare un'idea del sistema si rimanda allo schema di *figura 6.1*, dove si nota come il TAC al centro, comunichi in maniera integrata con gli strumenti presenti in sala prove.

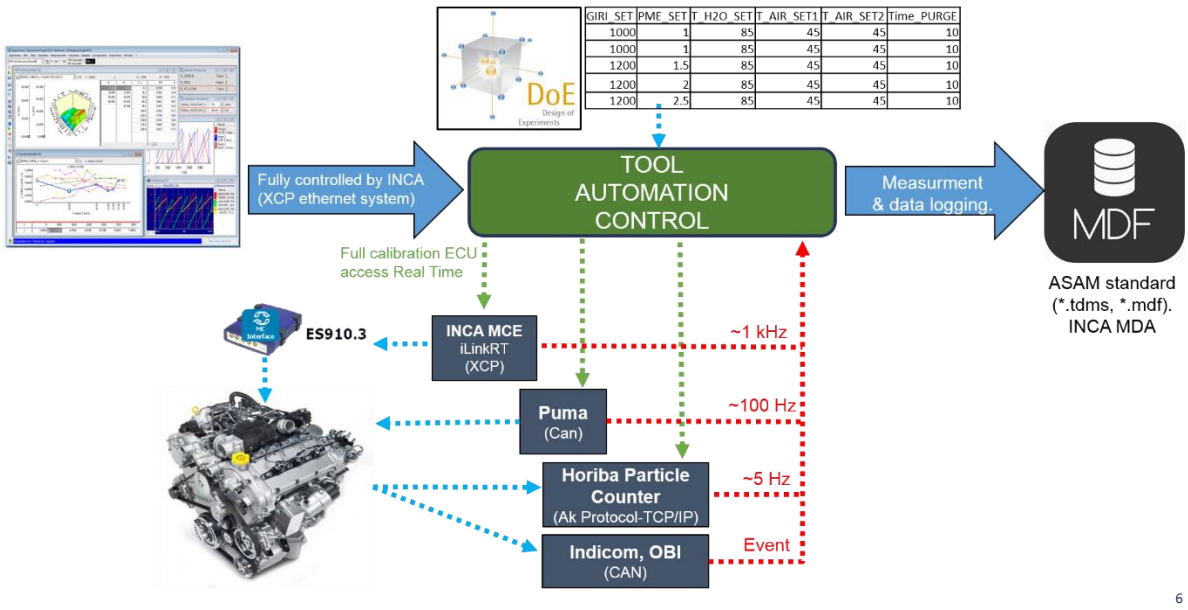


Figura 6.1: Schema dell'applicazione DOE sviluppata.

La *figura 6.1* mostra come l'input del sistema è l'output di sistemi di ottimizzazione di prestazione motore, tra cui (ETAS ASCMO).

6.2 ETAS ASCMO

Il software ETAS ASCMO (Advanced Simulation for Calibration, Modeling and Optimization) consente la modellazione del comportamento di un sistema fisico basato su leggi complesse e a volte non completamente note [10]. Un motore endotermico rappresenta un perfetto esempio: sono note le leggi fisiche che controllano i fenomeni che avvengono, ma è impossibile risolverle in modo analitico per valutare le influenze dei diversi parametri (per esempio spark advance, start of injection, fuel pressure, angolo

apertura valvola di aspirazione, angolo chiusura valvola aspirazione, waste gate, etc.) su tutti gli output (potenza, coppia, emissioni, consumi etc.) contemporaneamente.
ASCMO affronta questo problema generando un piano di test che determina efficacemente la natura delle risposte del motore ed è quindi utilizzato al fine di diminuire le ore di lavoro in sala ed i costi di tale attività nonché utile soprattutto ad ingegneri con poca esperienza.
Tale software supporta l'intero processo di calibrazione, dalla generazione di DoE (Design of Expriments), allo sviluppo di un modello numerico il cui comportamento sia equivalente al modello fisico in studio: il modello numerico viene "istruito" sul comportamento da seguire grazie ad un numero adeguato di misure sul sistema fisico (motore al banco), registrando input e output per ognuna di esse, e con i dati acquisiti si "allena" il modello numerico per poi eseguire un'ottimizzazione.

6.2.1 Design of Experiment

Uno degli approcci più comuni e semplici per lo studio di un sistema complesso e composto da molte di variabili è modificare un fattore per volta mantenendo tutti gli altri fissi. I risultati spesso forniscono una quantità limitata di informazioni con la richiesta di un eccessivo tempo di lavoro. Seguire questo procedimento implica un aumento esponenziale delle misure da effettuare. I *plans of experiment* possono essere molto diversi e dipendenti dalla complessità del modello in studio. I tre principali sono (vedi *figura 6.2)*:

- *grid* (gliglia), nel quale si eseguono prove con ogni combinazione possibile dei parametri da ottimizzare (il numero di prove può risultare molto elevato);
- *cross*, nel quale fissando un punto base, si fa variare un input per volta, ma non è adatto sistemi fisici complessi;
- *space-Filling*, è il metodo con il quale si ricopre in maniera uniforme l'intero spazio in prova, con un numero limitato di test;

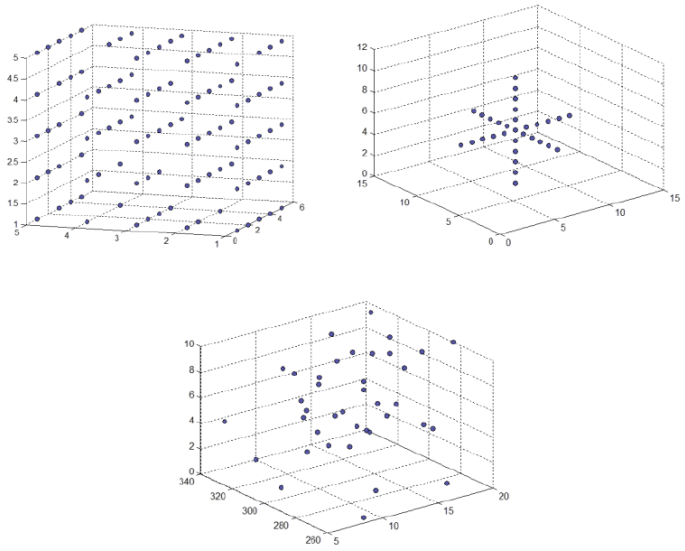


Figura 6.2: Tipologie di Plan of Experiment, Grid, Cross, Space-Filling

I metodi statistici coprono un ruolo importante nella pianificazione, esecuzione ed analisi dei risultati di un sistema incognito. Il Design of Experiment (DoE) è quindi un insieme di metodologie statistiche per la modellazione di un sistema non noto, capace di modellare il sistema avendo come base un numero limitato di misure [11]. Il DoE minimizza gli sforzi di misura a banco anche con la variazione di diversi input per volta. È importante notare come non tutti i fattori hanno lo stesso peso sulle prestazioni: alcuni possono avere elevate influenze, altre medie influenze, oppure nessuna influenza. Quindi un experiment deve essere pianificato con cura, in modo da capire quali fattori e con quale peso influenzano le prestazioni del sistema. Con un'opportuna applicazione del DoE si possono abbattere drasticamente i costi delle prove, ottenendo molte utili informazioni dai risultati.

Per ogni DoE vengono definite una serie di grandezza specifiche:

- **Fattori**: parametri su cui si intende agire ed oggetto di studio (controllabili) ;
- **Range**: valore minimo e massimo di variazione entro il quale i fattori verranno fatti variare. Il range può essere completo per un determinato fattore oppure limitato ad un intervallo di valori specifico se si è già a conoscenza del suo parziale effetto sulle misure;
- **Misure**: misurazioni effettuate sul sistema durante la variazione dei fattori e oggetto di ottimizzazione (non controllabili).

- **Matrice dell'experiment**: ha in colonna i fattori presi in considerazione, e in ogni riga saranno presenti i diversi valori (la singola riga sarà definita come punto DoE).
- **Condizioni di prova**: rappresentano le condizioni in cui saranno eseguiti i test (regime, carico, temperatura acqua motore, tempo di acquisizione, frequenza di acquisizione etc.).

L'esecuzione del DoE prevede la modifica in contemporanea dei fattori, al fine di osservare i corrispondenti cambiamenti. I principali vantaggi che comporta sono:

- maggiore affidabilità dei risultati;
- riduzione dei tempi di lavoro;
- riduzione dei costi;
- riduzione tempi di sviluppo;
- legame tra fattori e misure più intenso.

La variazione manuale di ogni fattore può comportare comunque un eccessivo spreco di tempo che farebbe perdere i vantaggi di velocità del DoE, soprattutto se il numero di fattori risulta elevato. Automatizzare questo processo sarà uno degli obiettivi del software di automazione in sviluppo (TAC). L'efficienza della sala motore con l'esecuzione del DoE aumenta sfruttando le potenzialità del tool che automatizza il processo, e con l'adeguata pianificazione del DoE si potranno ottenere risultati utili per il miglioramento delle prestazioni nel minor tempo possibile. La metodologia DoE è quindi divisa in tre parti:

1. scelta delle misure da ottimizzare. Pianificazione delle prove da effettuare, con selezione dei fattori ed il loro range di variazione. Scelta delle condizioni in cui operare;
2. esecuzione della prova al banco (con utilizzo del TAC);
3. analisi e verifica dei dati con generazione dei modelli matematici/empirici.

Una volta generato il modello è possibile ricavare i migliori valori dei fattori sia per l'ottimizzazione mono-obiettivo che multi-obiettivo. L'ottimizzazione mono-obiettivo fornirà i valori dei fattori che corrispondono ai migliori risultati con l'inserimento di opportuno peso ovvero l'importanza che il calibratore attribuisce ad ogni misura. Invece, l'ottimizzazione multi-obiettivo fornirà curve di Pareto delle misure, e i corrispondenti fattori che le generano. ASCMO è quindi molto utile per automatizzare il processo di calibrazione a banco dei nuovi motori. Lo schema concettuale è riportato in (*Figura 6.3).*

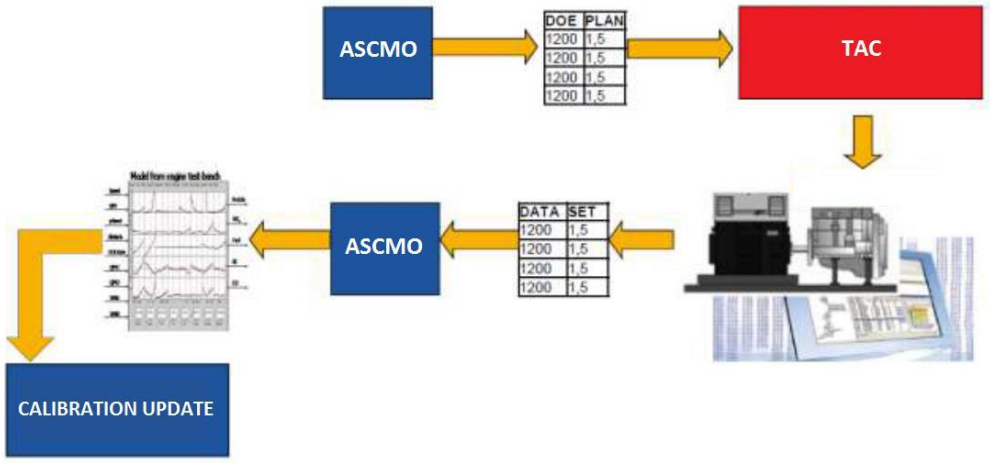


Figura 6.3: Schema automazione banco per attività DoE

6.3 Applicazione per l'esecuzione del DOE

6.3.1 Concetti preliminari.

L'applicazione prevede lo sviluppo di un sistema automatizzato per la realizzazione di prove in sala. La particolarità di questa applicazione che la differenzia da altre commerciali, è quella di semplificare i processi di comunicazione fra i vari strumenti, sfruttando le potenzialità in termini di velocità di comunicazione dei protocolli utilizzati. Oltre a questo c'è la possibilità di sviluppare strategie per l'esecuzione di algoritmi di controllo anche complessi in relazione a specifiche esigenze dello sperimentatore. L'applicazione, oltre ad avere grandi potenzialità per l'esecuzione di prove in automatico e una rilevante espandibilità, presenta anche dei lati puramente esecutivi legati alle richieste degli utilizzatori, in particolare l'esigenza di comandare l'applicazione con il software INCA e alcune scelte progettuali.

Gli input del sistema sono le misure dei vari strumenti e i passi da eseguire provengono del sistema di ottimizzazione (ad esempio ASCMO) che si presentano come un tabulato di punti contente 6 parametri banco e fino a 10 parametri centralina. La scelta del numero di variabili deve essere preventivamente definita per essere utilizzata da INCA via XCP (che prevede di assegnare una dimensione fissa a ogni variabile da controllare) ed è stata determinata in collaborazione con l'utente.

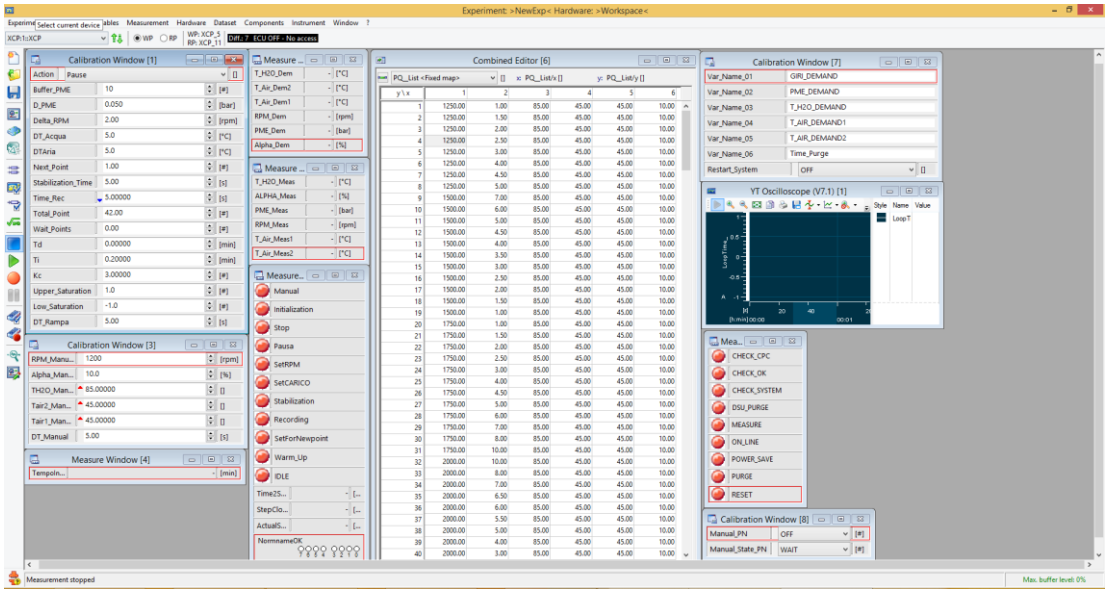


Figura 6.4: Interfaccia utente INCA dell'applicazione

La modalità di invio dei parametri al PUMA è quella descritta nei capitoli precedenti. L'utente avvia un ciclo automatico, che prevede una serie di operazione che il sistema di controllo banco deve eseguire per poter interagire con il TAC. Come precedentemente descritto, è caricato in entrambe le applicazioni un database CAN con le variabili che devono essere scambiate tra i due sistemi ed è il TAC che agendo, con le logiche che descriveremo nei prossimi paragrafi, sui setpoint del PUMA e di centralina gestisce ogni punto motore in maniera automatica.

L'applicazione prevede l'utilizzo di una logica di programmazione comune per l'esecuzione di processi automatizzati: la macchina a stati (state machine) di cui faremo un accenno nel prossimo paragrafo.

Dal punto di vista del codice, senza entrare troppo nel dettaglio, ma soffermandoci su una descrizione sommaria dell'approccio utilizzato, si è proceduto ottimizzando gli algoritmi al fine di sviluppare un'applicazione che avesse nella rapidità dello scambio di dati verso l'esterno il principale punto di forza, senza penalizzare la robustezza e la sicurezza dei processi. Semplificando, possiamo pensare l'applicazione come vari loop di controllo che girano in parallelo:

- State Machine: Questo loop definisce i passi dell'applicazione ed è quella che gestisce l'esecuzione degli altri loop. Questa parte di codice principale deve avere una frequenza fissata per permettere l'esecuzione deterministica dei vari setpoint al

sistema di controllo banco. Frequenze di esecuzione fino a 500Hz sono state validate, senza riscontrare ritardi nell'esecuzione dei vari step.

- Loop per la gestione degli strumenti di analisi gas (HORIBA SPCS, Horiba MEXA ONE, AVL Micro Soot Sensor): precedono l'implementazione del protocollo AK via ethernet. Questo protocollo era stato sviluppato per comunicazioni seriali (RS232), per cui non è possibile ottenere dati a frequenze maggiori di 10-15Hz. I loop di gestione non hanno una frequenza predefinita ed è lo strumento a regolarne la frequenza.
- ilinkRT. Come descritto in precedenza, questa implementazione prevede due flussi: quello di ricezione delle misure (non particolarmente rilevante per questa applicazione) che dipende dalla frequenza di invio delle variabili e quello di invio dei comandi. Frequenze di 1kHz possono essere facilmente raggiunte per l'invio dei dati.
- Gestione comunicazione CAN. Questo parte permette la comunicazione a frequenze dell'ordine 200Hz ed è gestito da un loop che prevede in parallelo la lettura e la scrittura dei Frames.

Di seguito trattiamo in modo particolare la gestione della state machine, in quanto rappresenta la parte principale del sistema.

6.3.2 State machine

La macchina a stati (o state machine) è una delle architetture fondamentali per gli sviluppatori software, utile per creare applicazioni in modo rapido e funzionale, impiegata nel caso di gestione di strategie di controllo formate da operazioni sequenziali. Il passaggio da uno stato all'altro è basato sull'input dell'utente o sui risultati ottenuti in seguito a specifiche elaborazioni dei dati. La macchina ruota attorno a tre concetti:

- lo stato;
- l'evento;
- l'azione.

Lo stato è la 'posizione' del programma in un dato momento (per esempio l'attesa di un comando); *gli eventi* sono avvenimenti che hanno un significato specifico per il programma stesso (come per esempio cliccare su un pulsante); *l'azione* è la risposta che il programma darà al verificarsi di un evento.

Molte applicazioni richiedono uno stato di "inizializzazione" dal quale è possibile poi eseguire molte azioni differenti. Come è spesso necessario ad uno stato di inizializzazione risulta utile avere anche uno stato di "Arresto".

Il compito della macchina a stati è di rendere automatico il processo. Le fasi saranno: lettura dei punti da un file, raggiungimento del punto di regime e carico richiesti, implementazione parametri in centralina, verifica della stabilità del sistema al fine di avviare la registrazione dei dati.

La Macchina a Stati creata per l'applicazione prevede diversi stati base:

- Inizializzazione
- Attesa
- Set RPM
- Set ECU
- Set Carico;
- Registrazione

A questi vanno aggiunti altre configurazioni, necessarie all'operatore per mettere in sicurezza il motore e per raggiungere condizioni particolari di funzionamento.

- Idle
- Warm Up
- Manual

Una particolarità importante di questo approccio prevede che durante l'esecuzione in automatico, sia possibile passare da uno stato all'altro.

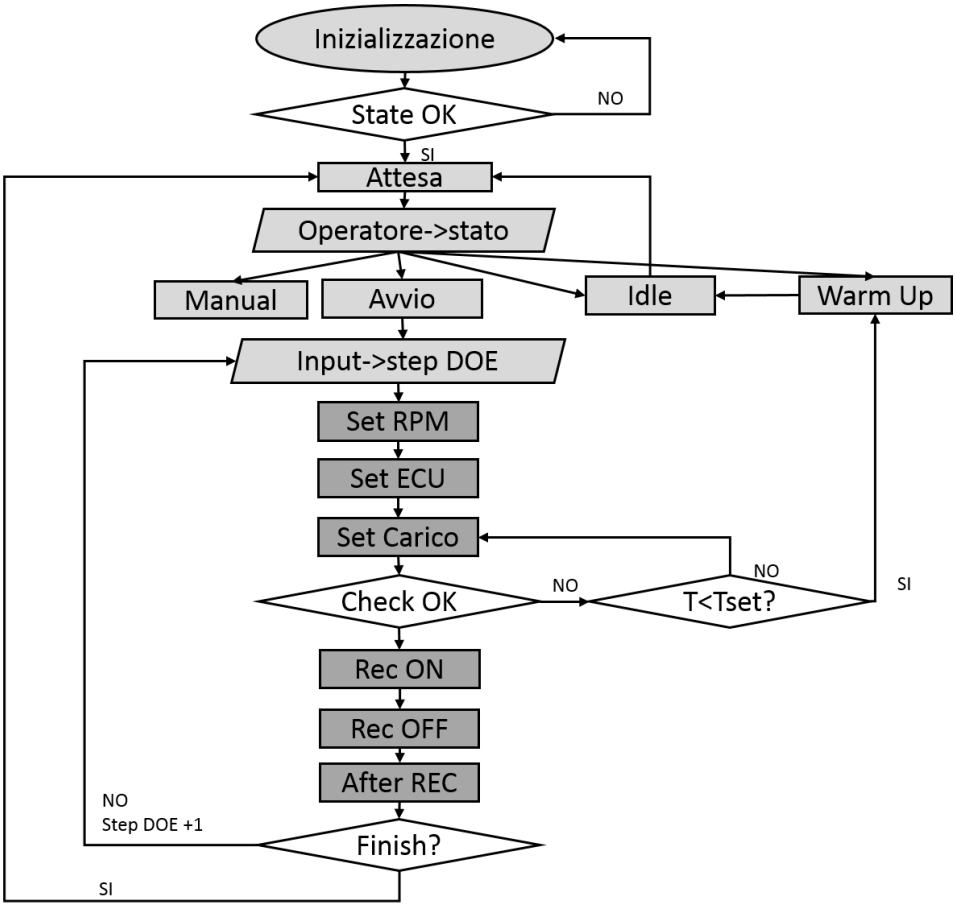


Figura 6.5: Schema a blocchi della macchina a stadi utilizzata per l'applicazione.

Inizializzazione. Lo stato di inizializzazione è il primo stato che viene effettuato dall'applicazione. Esso ha il compito di verificare che le condizioni iniziali per l'esecuzione di essa vengano verificate. In particolare, la presenza del file con la lista dei punti da eseguire, il corretto avviamento della comunicazione CAN con il PUMA, l'inizializzazione degli strumenti collegati (tra cui in particolari i vari analizzatori e misuratori di particelle inquinanti dei gas di scarico), l'avvenuta comunicazione con i parametri centralina (iLinkRT, modulo ES910.3) e il collegamento con l'interfaccia utente. Questo ultimo punto verifica che l'utente abbia correttamento aggiunto il TAC a INCA (che come abbiamo introdotto nei capitoli precedenti, viene utilizzato come interfaccia utente) e lo abbia inizializzato.

Attesa. Lo stato di attesa è uno stato transitorio in cui si attende che l'utente scelga lo step successivo e in cui ritorniamo dopo la fine di ogni registrazione. Durante l'esecuzione dell'applicazione ogni volta che si ritorna in questo stato vengono aggiornati

i valori di setpoint sul bus CAN ai valori misurati. Il rischio, infatti, è quello che, in particolare all'avvio, siano richiesti dei setpoint che possono differire molto dal punto di funzionamento in cui si trova il motore, da cui l'esigenza di aggiornare il valore dei setpoint a quello misurato, prima dell'avvio delle rampe per il raggiungimento del punto programmato. Per avere un ulteriore controllo sulla sequenza delle operazioni, al primo step di avvio è stato implementato un colloquio fra PUMA e il TAC: inizialmente il tool comunica al PUMA di essere pronto a partire e il PUMA risponde solo quando ha effettuato una serie di controlli sulle variabili di setpoint che gli sono comandate e finché questa procedura non da esisto positivo, l'utente non può avviare l'applicazione.

Esecuzione: Con esecuzione si comprendono tre stati:

- **Set rpm:** viene eseguita una rampa di giri in cui, in un tempo fisso, viene calcolata ed eseguita la pendenza necessaria per portare il motore al regime richiesta nello specifico punto:

$$RPM_n = RPM_{n-1} + \Delta RPM, \text{ definendo:}$$

$$\Delta RPM = \frac{(RPM_{Set} - RPM_{meas})}{60 \cdot T_{rampa}} \cdot F_{main}$$

Dove RPM_nè il valore attuato l'n-esimo passo di calcolo; ΔRPM è il valore di giri da decurtare ogni step; RPM_{Set} e RPM_{meas} sono rispettivamente la velocità target e la velocità misurata; T_{rampa} è il tempo di esecuzione del transitorio e F_{main} è la frequenza del loop principale di calcolo.

- **Set ECU:** in questa fase vi è l'implementazione dei parametri centralina. Questa parte è direttamente connessa con il loop di gestione del protocollo iLinkRT e ne gestisce le operazioni. Vengono letti e implementati i valori dei parametri di centralina. Per questa parte è fondamentale verificare l'effettiva variazione di tali parametri. Per questo viene implementato il comando di download della calibrazione dalla ECU come verifica dell'avvenuto upload.
- **Set Carico:** è uno degli step più elaborati dell'applicazione in cui è presente il controllo in retroazione con controllore PID della PME, agendo direttamente sull'attuatore pedale di PUMA (questa parte verrà descritta in seguito). In questa fase vengono inoltre valutati i limiti delle varie grandezze motore, prima di passare

all’implementazione dei parametri centralina. Viene valutato se le variabili di interesse (elencate in seguito) sono all’interno di un range predefinito dall’utente. Le variabili controllate sono: giri motore, temperatura liquido di raffreddamento, temperatura aria aspirata, PME.

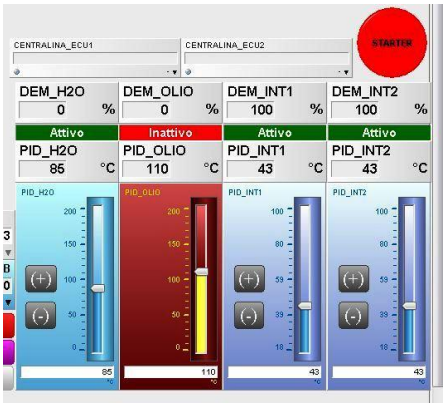


Figura 6.6: Controllori apertura elettrovalvole su interfaccia PUMA. Questi valori vengono implementati direttamente dal tool su PUMA.

Registrazione: viene inviato a PUMA il valore impostato come trigger della registrazione, PUMA esegue poi la registrazione per un tempo impostato dall’applicazione. Una volta terminate le varie registrazioni (in particolare su INDICOM, che misurando sempre un numero di cicli e non un tempo, può terminare in un momento differente da PUMA) viene incrementato un contatore che rappresenta il numero di registrazioni effettuate.

After REC: questo stato prevede l’avvio di quelle procedure necessarie al successivo punto motore e alla messa in sicurezza del motore. Prima di tutto vengono riportati i valori di centralina a quelli di riferimento e vengono lanciate le procedure di lavaggio per gli strumenti di analisi dei gas di scarico (in particolare l’Horiba SPCS). La Macchina a Stati creata per l’automatizzazione del PQ (Piano Quotato) e prevede diversi stati base:

- Inizializzazione
- Attesa
- Esecuzione
- Raggiungimento regime motore,
- Verifica limite PME,
- Implementazione parametri centralina;

- Verifica stabilità
- Registrazione

A questi vanno aggiunte altre configurazioni, necessarie all'operatore per mettere in sicurezza il motore e per raggiungere condizioni particolari di funzionamento. Come anticipato in precedenza, l'operatore può sempre passare a uno di questi tre stati interrompendo l'esecuzione della modalità automatica in qualsiasi momento.

- Idle: il motore viene messo, dopo aver effettuato un transitorio di giri e carico, al valore di rpm e carico considerato di minimo.
- Warm Up: in questa fase il motore viene portato in uno stato che ne permette il riscaldamento. Durante l'esecuzione in automatico se, nello stato set Carico le temperature sono più basse di quelle richieste, in automatico viene eseguito questo passo. Quest'ultima fase è composta da 3 stadi:

1. $T < 50°C$ Il motore avviato per la prima volta dovrà raggiungere una temperatura di 50°. Il carico e i regimi impostati devono tenere conto che il motore non ha ancora raggiunto una certa temperatura.
2. $T => 50°C$ Raggiunta la temperatura di 50°C, il successivo stadio è di arrivare ad ottenere la temperatura target richiesta (inseribile con il comando TEMP TARGET nell'interfaccia) per effettuare, ad esempio un Piano Quotato. In questo caso il carico e i regimi saranno più alti.
3. $T => T_{target}$ Raggiunta la Temperatura target il motore si passerà allo stato di idle descritto in precedenza.

- Manual, l'operatore ha la possibilità di gestire il motore in manuale e portarlo nelle condizioni definite. Questa operazione, può ovviamente essere implementata anche su PUMA, ma questo comporterebbe l'uscita dal ciclo automatico. Infatti si è pensato che la possibilità di riportare in manuale il PUMA, riguardi condizioni di emergenza,

6.3.3 Controllore PID

Il PID è un sistema in retroazione diffuso nei sistemi di controllo (vedi *figura 6.7*). Tale sistema, è in grado di reagire a un eventuale errore (positivo o negativo) tendendo verso il valore zero. La reazione all'errore può essere regolata rendendo il sistema molto versatile. Il PID acquisisce in ingresso un valore da un processo (*process variable)* e lo

confronta con un valore target (*set point*). La differenza tra i 2 valori, ovvero l'errore è impiegato per la determinazione della variabile di uscita cioè la variabile controllabile.

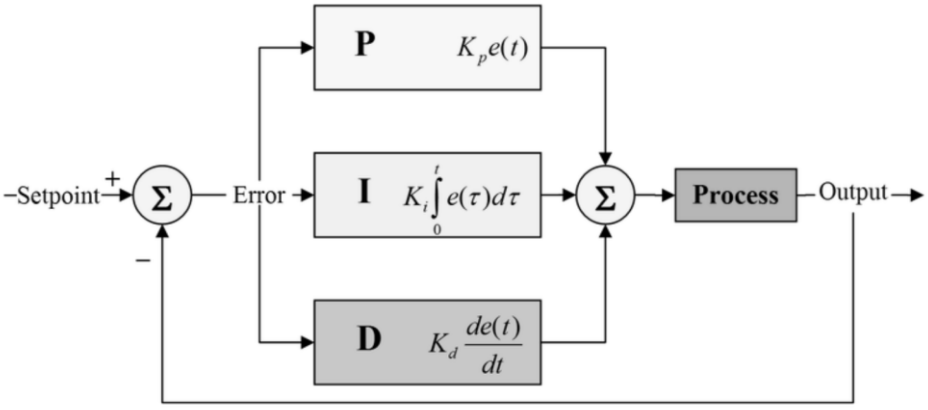


Figura 6.7: Schema sistema chiuso in retroazione

La risposta del PID è determinata attraverso tre azioni:

- Azione Proporzionale;
- Azione Integrale;
- Azione Derivativa.

L'azione Proporzionale up è ricavata moltiplicando il segnale di errore '***e***' per un costante calibrata opportunamente:

$$u_p = k_p \cdot \mathrm{e}$$

Con la sola azione proporzionale non è possibile garantire che l'errore raggiunga il valore 0, poiché l'azione *up* è possibile solo con l'errore diverso da zero.

L'azione integrale è proporzionale all'integrale nel tempo del segnale di errore "*e*":

$$u_i = K_i \int e\,(t)dt$$

Il PID così tiene in memoria i valori precedenti del segnale di errore, così da avere anche ui non nullo nel caso in cui l'errore sia zero. Questa azione permette il raggiungimento del target richiesto, superando il limite imposto dalla sola azione proporzionale. L'azione derivativa serve a migliorare le prestazioni del PID aggiungendo la componente proporzionale alla derivata dell'errore:

$$u_d = K_d \frac{de}{dt}$$

Questa evento serve a compensare rapidamente la variazione di errore nel caso si abbia un suo aumento, valutando la velocità di cambiamento. Spesso tale azione è tralasciata

nelle comuni applicazioni del PID, perché in particolari condizioni può portare all'instabilità del sistema.

È stata eseguita una taratura del PI (proporzionale-integrativo) ad anello chiuso, detto anche secondo metodo di ZIEGLER-NICHOLS, con le seguenti modalità:

- Controllore con il solo contributo proporzionale a basso Kp;
- Aumento progressivo del guadagno Kp fino al verificarsi di un'oscillazione ad ampiezza costante (vedi *figura 6.8*) a cui è assegnato il valore Ku;
- Misura il periodo Tu dell'oscillazione

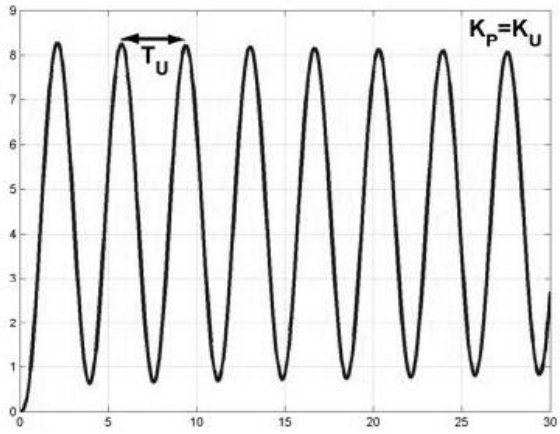


Figura 6.8: Grafico oscillazione ad ampiezza costante

- Una volta definito il Ku si seguono i valori tabellati in figura 6.9. Da qui si può partire con un tuning più raffinato a seconda del sistema controllato.

	K_P	T_I	T_D
P	$0.5\overline{K}_P$	–	–
PI	$0.45\overline{K}_P$	$\frac{\overline{T}}{1.2}$	–
PID	$0.6\overline{K}_P$	$\frac{\overline{T}}{2}$	$\frac{\overline{T}}{8}$

Figura 6.9: Costanti di ZIEGLER-NICHOLS

Procedendo con il metodo appena descritto, sono stati individuati i valori di Kp=0.6 e un tempo di oscillazione di 5 secondi. Utilizzando la tabella di ZIEGLER-NICHOLS relativa

ad un controllore PI si sono ricavati il Kp=0.3 e il Ti=0.07min. Dopo la fase finale di tuning, si è notato un miglioramento della stabilità del controllo e di riduzione dell'errore per un valore finale Ti=0.2 minuti.

6.4 Risultati Ottenuti

In questo capitolo verranno descritti i profondi cambiamenti che un approccio di questo tipo, può apportare all'interno di un ambiente industriale. I risultati rappresentati nei paragrafi successivi evidenzieranno dei vantaggi non solo in termini economici e di tempo ma anche di effettivo miglioramento della qualità dei test. Le sale prove Maserati sono provviste di un layout che prevede l'interfaccia in modalità semi-automatica con grande parte della cella motore.

6.4.1 Esempio di un Piano Quotato in Automatico

Il Piano Quotato utilizzato per realizzare la prova, contiene punti motore con regime compreso tra i 1000 e 5000 rpm e PME tra 1 e 10 bar. Inoltre, sono stati specificati anche i limiti relativi alla temperatura dell'acqua di raffreddamento in uscita e dell'aria in uscita dall'intercooler. Le caratteristiche scelte del PQ per tutti i punti sono:

- T_acqua = 95-100 °C;
- T_aria = 35-45 °C;
- T_REC= 20 s;
- Numero punti motore= 93;
- Max errore PME = -0,05 e +0,05 bar;
- Tempo Purge SPCS = 30 s (ogni 15 punti);
- Registrazione su PUMA dei canali di PUMA, INDICOM, HORIBA-SPCS, INCA;
- Time Wait Points = 5 s.
- Dt rampa = 3 s (per giri motore)

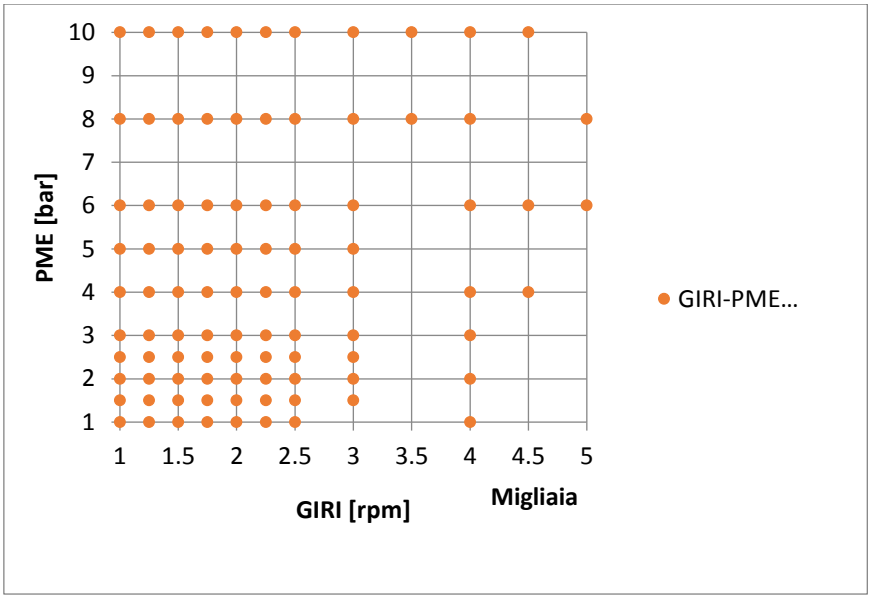


Figura 6.10: 93 punti motore caricati nel file .txt e inseriti nel PQ automatico.

6.4.2 Dati ottenuti e vantaggi riscontrati

Uno dei punti più importanti e difficile da rispettare è l'errore massimo della PME. Solitamente l'operatore in modo manuale deve modificare il valore manualmente provocando una perdita di tempo e una bassa precisione delle prove. Viste le difficoltà si accetta fino al 10% di errore sulla PME richiesta.

I risultati ottenuti relativamente al limite di errore richiesto per la PME sono (vedi *figura 6.11*):

- 83 punti con e<0,05 bar (azzurri);
- 8 punti con 0,09>e>0,05 bar (gialli);
- 2 punti con 0,09<e<0,30 bar (rossi).

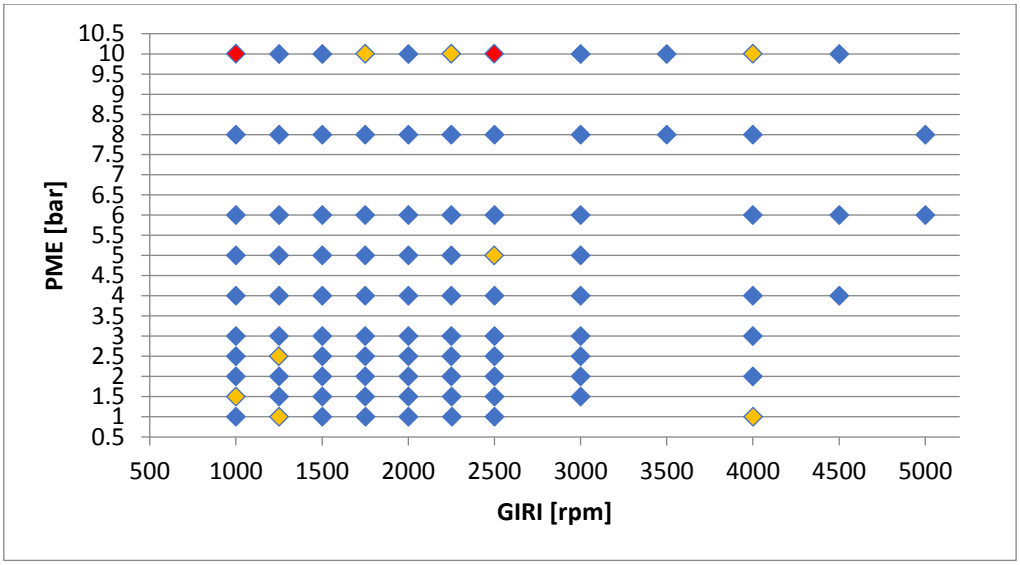


Figura 6.11: Punti Motore effettivamente raggiunti

La valutazione dei punti ottenuti è positiva perché si hanno 83 punti di sotto a un limite non sempre raggiungibile in manuale, ed 8 punti che non rientrano di pochissimo nel range. Nello specifico, si sono ottenuti 93 punti di cui due non accettabili (il motivo potrebbe essere legato alla non corretta calibrazione del motore in quei punti motore). Sono state inoltre verificate tutte le acquisizioni dei punti effettuati ed erano tutti presenti rispettando la durata inserita. Non si sono inoltre riscontrati valori della T_acqua e T_aria al di fuori di quelli inseriti nel file txt. Rilevata la precisione e l'affidabilità del processo rimane da conoscere la durata dell'intero processo. Generalmente un Piano Quotato come quello descritto avrebbe richiesto almeno dalle 5 alle 6 ore, mentre con il TAC ci sono voluti circa 160 minuti. Infatti con un operatore in modalità manuale, la velocità e l'affidabilità decrescono con l'aumentare del numero dei punti. Nell'esempio i punti erano 93, ma si deve immaginare che ci sono PQ che effettuano 500 punti, e alcuni di questi vengono limitati per un problema legato alla durata del processo. Questa nuova risorsa potrebbe essere opportunamente implementata per essere avviata nel periodo notturno, così che al mattino, si potrebbero avere i risultati delle prove e impiegare il resto della giornata per altre operazioni che richiedono, altresì, la presenza dell'ingegnere e dell'operatore di sala.

7 Sistemi RCP

7.1 Introduzione

Nell'ambito della definizione di strategie di controllo motore, a causa anche della crescente complessità dei propulsori, risulta sempre più importante la verifica preventiva delle potenzialità di soluzioni innovative in termini di prestazioni, consumi, emissioni, guidabilità, al fine di indirizzare lo sviluppo del motore. Nell'ambito appena descritto verranno mostrate le metodologie utilizzate in collaborazione con i gruppi R&D di Alfa Romeo-Maserati. Le applicazioni descritte prevedono l'interazione fra sistemi in grado di scambiare dati ed eseguire calcoli a frequenze elevate, permettendo l'esecuzione in tempo reale di strategie di controllo comparabili con gli algoritmi di calcolo dalla ECU.

7.1.1 Sistemi mild hybrid.

Al giorno d'oggi, come menzionato più volte è sempre più importante la diminuzione delle emissioni dei veicoli. Ad esempio per l'unione europea (UE) la legislazione sulle emissioni ha l'obiettivo ambizioso di ridurre le emissioni di CO2 media a 95 g/km per il nuovo parco auto, pari a il consumo di carburante di circa 4.11 / 100 km di benzina, o di 3.61 / 100 km di diesel. Non è facile raggiungere questo obiettivo migliorando solo l'efficienza del Motore a combustione interna (ICE) e di conseguenza l'elettrificazione del powertrain diventa un'opzione concreta per il recupero dell'energia in frenata per un miglioramento significativo dell'efficienza della powertrain [12].. I sistemi a 48V (mild hybrid) sono in grado di recuperare meno energia di frenatura rispetto a sistemi HV hybrid (High Voltage) a causa del ridotto picco del rate potenza/corrente. Nonostante questo, il costo di questi sistemi è, in maniera significativa, ridotto rispetto ai sistemi HV hybrid, il che rende i sistemi mild hybrid fra i migliori nel rapporto costi benefici.

Per dare un'idea dell'energia recuperabile, in figura 7.1 si riportano i diagrammi della potenza di frenata nei cicli di omologazione New European Driving Cycle (NEDC) e Worldwide harmonized Light vehicles Test Cycle (WLTC).

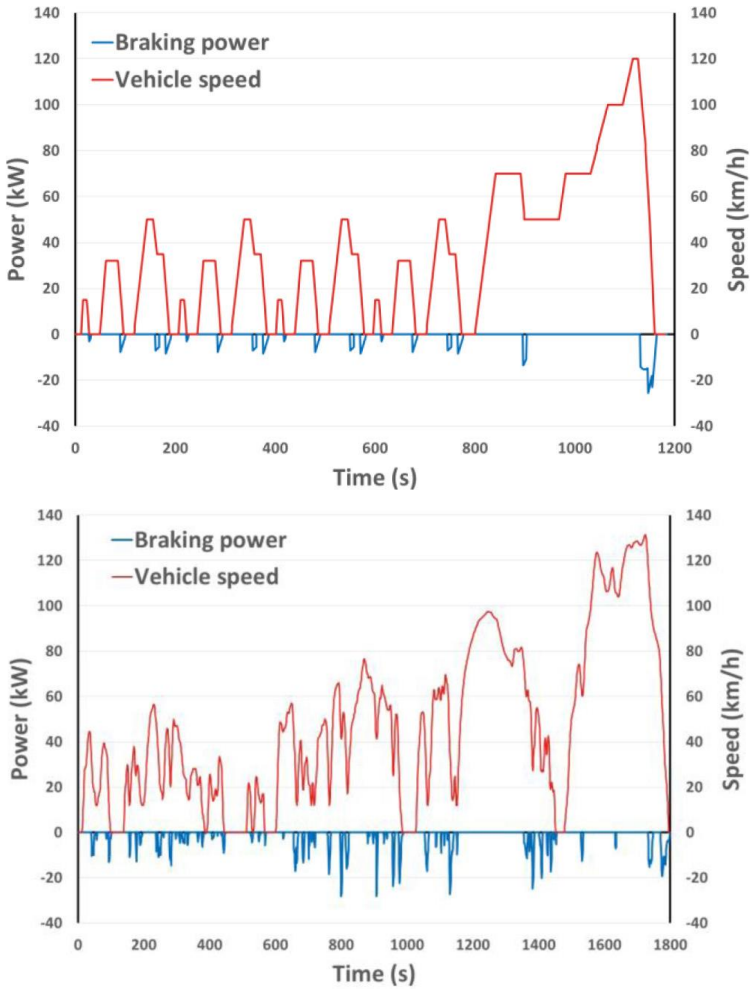


Figura 7.1: Velocità veicolo e potenza frenante dei cicli NEDC (sopra) e WLTC (sotto).

È importante, inoltre, chiarire che è presente una forte correlazione fra la potenza di frenata recuperabile e la taglia della e-machine. Dall'analisi della energia recuperabile attraverso la frenata nel ciclo WLTC si vede come nel passaggio da 10 a 30 kW, si ha un aumento della energia recuperata, al di sopra di tale taglia invece questa non incrementa. Questo fenomeno è dovuto principalmente alla mancanza di eventi frenanti che superano i 30 kW. Allo stesso modo anche la quantità di energia recuperabile tra i 20 e i 30 kW è piuttosto bassa, per cui in genere si utilizzano e-machines di taglia intorno ai 20kW.

Questi sistemi, possono prevedere diversi layout (differenti posizioni o in diverso numero e combinazione). Figura 7.2 rappresenta lo schema del sistema e delle possibili posizioni delle e-machines in un sistema 48 V mild HEV. Il veicolo usato è a trazione anteriore e il motore, frizioni, e cambio sono posti tra gli assali anteriore e posteriore. Da questa figura, si può vedere che ci sono cinque diverse posizioni per installare le e-macchines e quindi sono definibili cinque diversi layout:

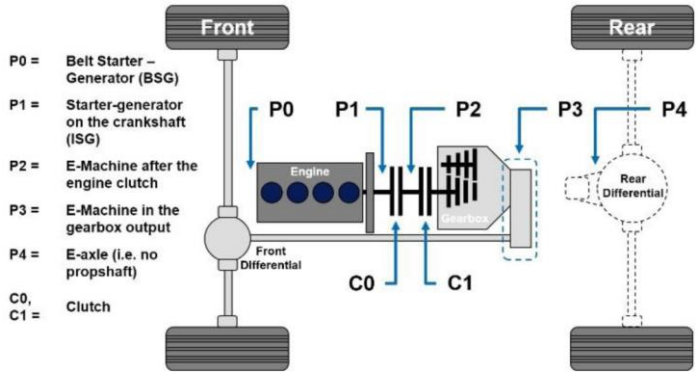


Figura 7.2: diagramma delle possibili posizioni in un sistema mild hybrid a 48V

- P0 – E-machine all'estremita anteriore: Belt Starter-Generator (BSG).
- P1 – E-machine sull'albero motore, tra il motore e frizione 1: Integrate Starter-Generator (ISG).
- P2 – E-machine all'ingresso del cambio.
- P3 – E-machine all'uscita del cambio.
- P4 – E-machine sull'asse che non ha alcun collegamento meccanico con la trasmissione.

Ci si è concentrati in maniera particolare sulla soluzione P0 in quanto è la configurazione che è stato possibile testare attraverso le metodologie descritte nell'elaborato. Il costo e le modifiche per installare l'e-machine P0 è minore perché il BSG può sostituire l'alternatore convenzionale. Il BSG necessita di un sistema di trasmissione a cinghia in grado di fornire la potenza per assistere il motore, o per recuperare energia cinetica in frenata e quindi prevede l'utilizzo di materiali ad alta resistenza per sopperire alle specifiche richieste. Lo svantaggio principale della disposizione P0 è la non alta efficienza (*figura 7.3*) comparata con altre soluzioni mild hybrid:

1. la trasmissione a cinghia non è così efficace come il motoriduttore utilizzato in altri layout;
2. quando il BSG è in modalità recupero e il motore è spento, la coppia di trascinamento del motore aumenta le perdite e riduce la quantità totale di energia di frenatura rigenerativa convertita in elettricità.

Inoltre, l'e-macchine avrebbe la capacità di supportare la modalità operativa puro veicolo elettrico (EV), ma l'efficienza sarebbe molto basso in quanto dovrebbe vincere la coppia di trascinamento del motore, nonché le perdite di trasmissione dei vari componenti.

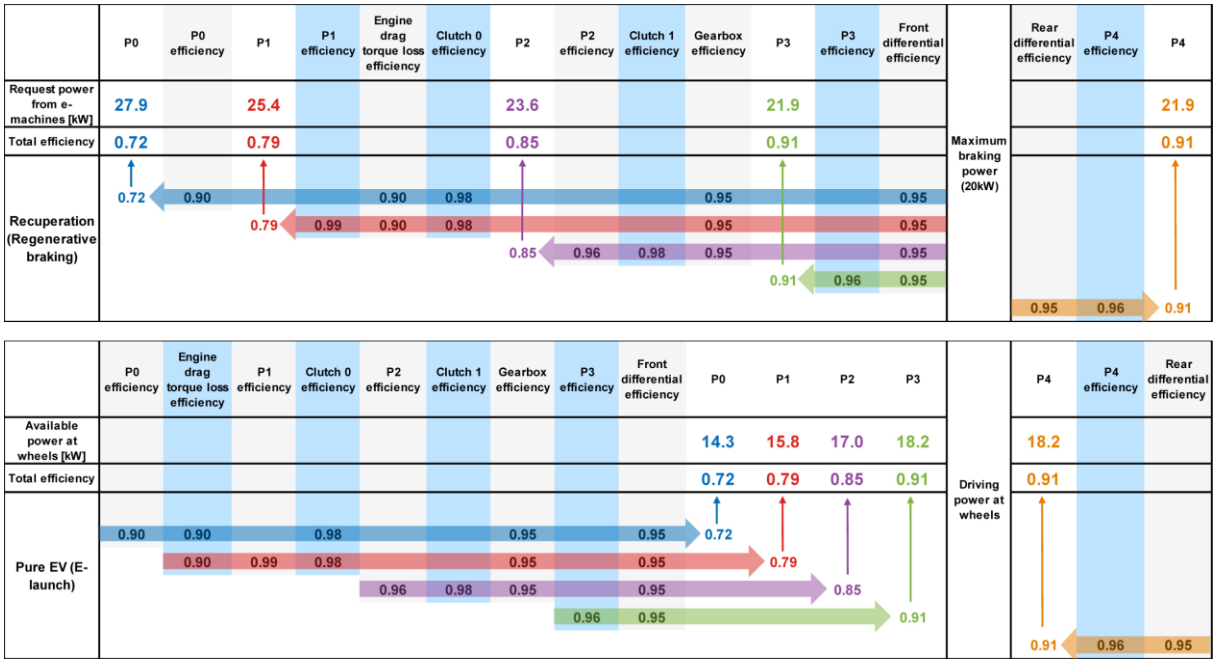


Figura 7.3: confronto tra i vari layout analizzati: modalità recupero (sopra) e E-launch (sotto).

7.2 Sistemi di Bypass funzionale.

Per bypass funzionale, si intende una metodologia che permette di testare strategie di controllo definite e modellate dal calibratore al fine di validare modelli che sostituiscono una funzione, o una sua parte, rispetto a quella implementata dall'applicatore e presente nella ECU. Questo permette, in tempi rapidi e con risultati immediati, di sviluppare strategie innovative svincolandosi (almeno nella parte iniziale) dall'applicatore SW con conseguente taglio di costi e riduzione dei tempi di implementazione. La parte di interazione con la ECU prevede l'acquisizione di misure (measures) dalla ECU e l'invio di valori di calibrazione (output del modello su calibrations della ECU), sfruttando le metodologie descritte in precedenza (via INCA MCE o ASAP3). La parte innovativa, che si è dimostrata vincente per questo tipo di applicazione, è rappresentata dalla possibilità di eseguire modelli anche complessi realizzati con software di modellazione standard nel settore automotive (in particolare Simulink della Mathworks) a frequenze elevate.

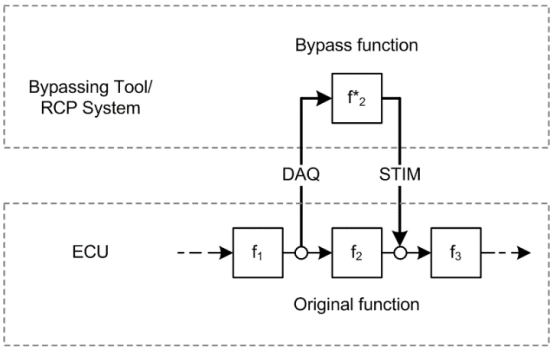


Figura 7.4: La figura mostra un esempio di una semplice bypass nella ECU. Gli ingressi della funzione bypass vengono campionati e inviati allo strumento di bypass come dati DAQ prima dell'esecuzione della funzione originale.

7.2.1 Implementazione strategie

Per l'applicazione si è utilizzata la libreria software MIT di Labview. Si è quindi sviluppato un ambiente che permette di caricare un modello compilato (attraverso tool di compilazione compatibili con Matlab) proveniente da Simulink.

In pratica si può ridurre il sistema all'esecuzione di un modello che può essere visto, una volta ottimizzato e compilato, come una "black box", in cui entrano input (nel caso specifico le measure della ECU) e dove gli output vanno associati ai vari parametri della centralina e sovrascrivono le calibrazioni motore definite per la verifica delle strategie. Oltre al flusso input/output è stata sviluppata la possibilità di modificare parametri del modello che non sono modificati in maniera ciclica, ma su cui può agire l'utente per condizionare il modello. Possiamo quindi riassumere il sistema con il diagramma a blocchi riportato in *figura 7.5.*

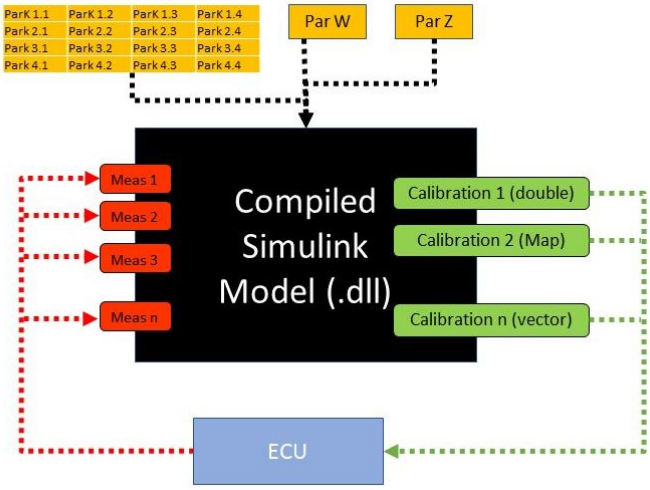


Figura 7.5: Diagrammi a blocchi del flusso dati dell'applicazione.

Possiamo distinguere:

- Measures: sono gli input del modello che nell'applicazione sono inviate dalla centralina attraverso i metodi descritti in precedenza.
- Calibrations: sono gli output del modello che vanno a sovrascrivere le calibration definite dall'utente sulla ECU. Le calibration devono essere di dimensione uguale al corrispettivo elemento scelto nella ECU. Come descritto nel capitolo relativo all'implementazione delle calibrazioni di centralina, è possibile agire sia sui vettori, che sulle mappe, "spianando" il valore della mappa a un unico valore o di una parte di essi.
- Parameters: sono quei valori definiti dall'utente nel modello che non partecipano al flusso di dati continuo, ma che partecipano al calcolo e servono per calibrare il comportamento del modello in caso di sperimentazione sul sistema fisico.

L'applicazione è stata implementata in modo che, al momento dell'importazione del compilato generato dal modello, riconosca in automatico il numero di input, degli output e dei parametri. Va sottolineato come, di quest'ultima categoria, riconosca anche i rispettivi valori di default.

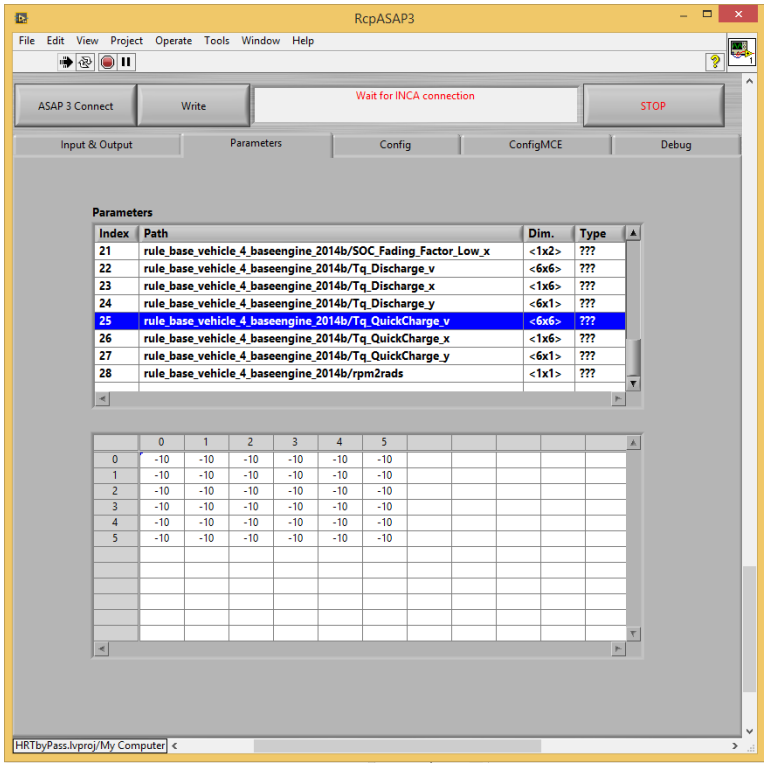


Figura 7.6: Interfaccia di modifica dei parametri del modello. È possibile agire sul singolo elemento della mappa o su tutta la mappa

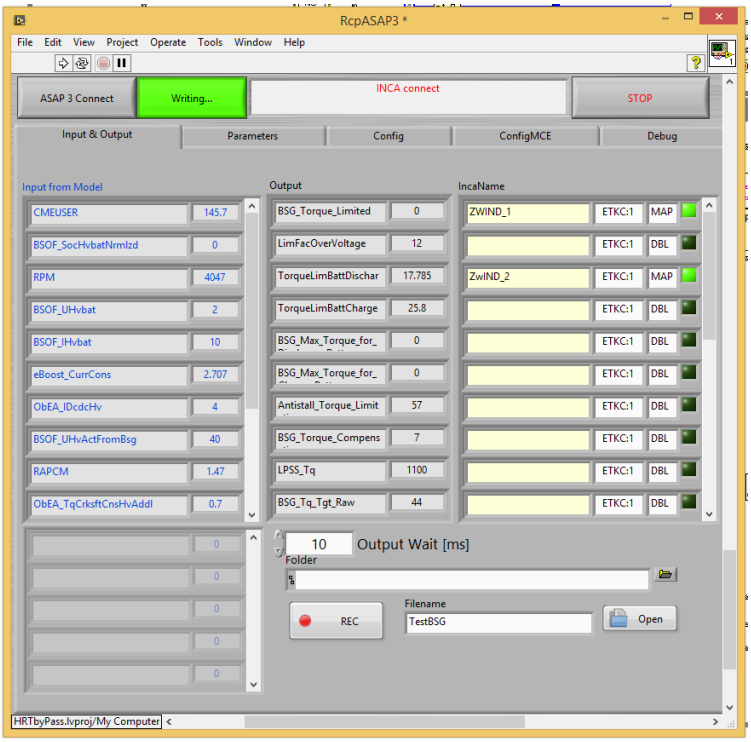


Figura 7.7: Interfaccia in cui si vede il valore degli input del modello con il valore delle measure e di output del modello. Nella lista IncaName è possibile scegliere il valore della centralina da sovrascrivere con i parametri del modello

Lo sviluppo dell'applicazione ha previsto due step:

1. Sviluppo dell'interazione fra la parte di modellazione Simulink e ambiente Labview.
2. L'implementazione di sistemi di comunicazione con la centralina attraverso l'utilizzo del protocollo ASAP3 (l'applicazione è definita SRT: slow real time) e INCA MCE e iLinkRT (definita con FRT, fast real time).

Come si evince dalla *figura 7.8*, la maggiore differenza tra i due approcci riguarda il tempo di chiusura del loop, da cui la differenza tra slow e fast. Le differenze fra le due soluzioni, sono riassumibili in alcuni aspetti:

- Il tipo di protocollo. I protocolli ASAP3 e iLinkRT sono già stati descritti in precedenza. Riassumendo sono due protocolli di comunicazione che se in parte hanno gli stessi obiettivi, dall'altra utilizzano metodologie differenti. Il primo si basa sulla richiesta di parametri al sistema attraverso un interlocutore (MC) che prevede poi la conversione dei dati da implementare in centralina, il secondo sulle richieste dirette a specifichi indirizzi di memoria direttamente al modulo di emulazione della ECU. Il secondo approccio, prevede, inoltre, un flusso di comandi ottimizzato per lo scambio dati di un sistema complesso come una ECU motore.

- L'HW utilizzato. Nel caso di applicazioni Host (SRT) il tempo di esecuzione può subire dei ritardi, in quanto un sistema operativo commerciale alloca degli slot di memoria per processi non necessariamente dedicati a uno specifico task. Utilizzando architetture HW-SW dedicate come Labview RT insieme a moduli di prototipazione rapida come ETAS ES910.3 (necessario per INCA MCE), i tempi di esecuzione dei loop di calcolo e di comunicazione sono ottimizzati e deterministici (FRT).

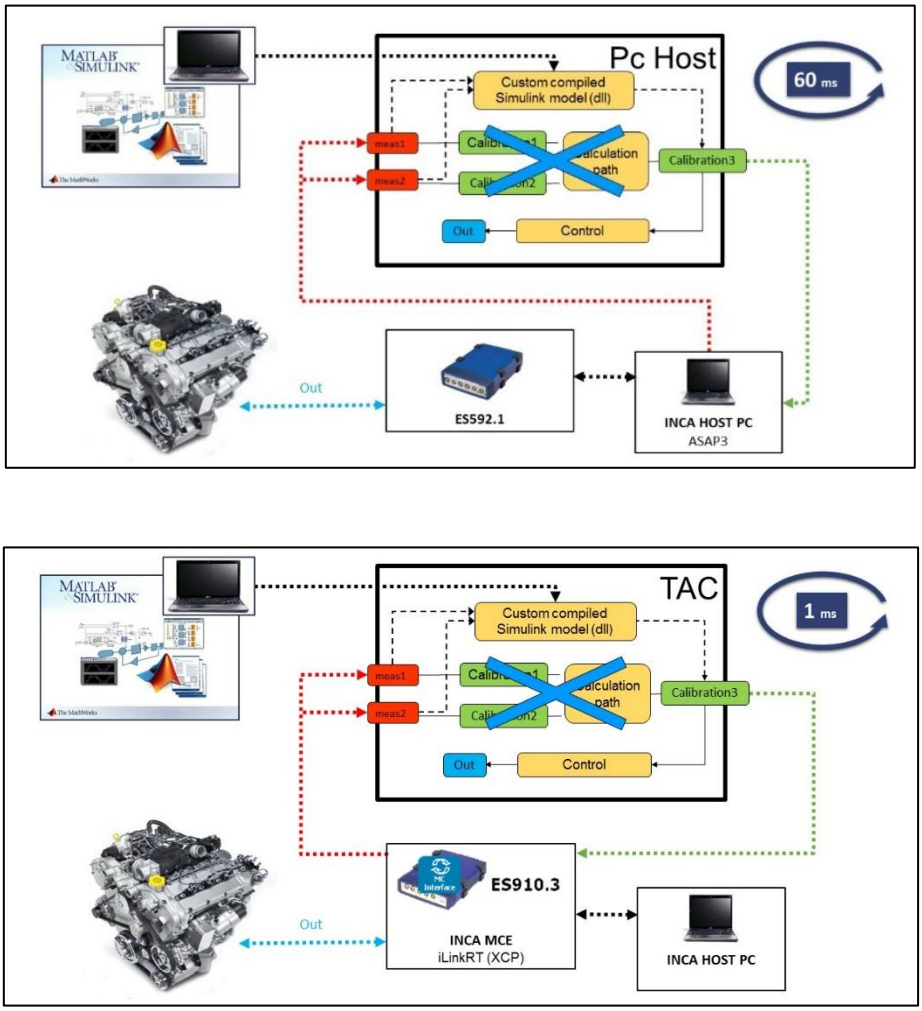


Figura 7.8: Confronto fra sistemi bypass definiti LRT sopra e HRT sotto

I vantaggi e gli svantaggi delle due metodologie possono essere così riassunti:

- Comunicazione con ASAP3. Frequenze di ricezione delle measure nell'ordine di 10 ms (dipendente dal numero di variabili) e di invio della calibrazione di 50 ms con una chiusura del loop di controllo con un tempo minimo di 60ms (escludendo il tempo di esecuzione del modello, che dipende dalla frequenza di calcolo imposta al modello stesso). L'applicazione essendo eseguita su pc Host, permette di essere eseguita senza

layout particolari rispetto a quelli tipicamente presenti in vettura e in sala prove, rendendola di fatto molto pratica in quanto si limita a una applicazione eseguibile, realizzata e compilata in LabVIEW, ed eseguita sullo stesso pc in cui è presente INCA.

- Comunicazione iLinkRT. Ricezione delle measure alle frequenze dei raster (frequenze specifiche definite nel file a2l per ogni tipologia di ECU a cui vengono rese disponibili le measure) presenti in centralina. Nel caso specifico i raster disponibili dipendono dall'applicatore SW e oltre al raster, sempre presente, che rende i dati disponibili ogni PMS (punto morto di lavoro di ogni cilindro), valori tipici possono essere 4, 8, 12, 100 e 500 ms (Marelli); 10, 100 ms (Bosch). La chiusura del loop può avvenire quindi all'interno del ciclo motore successivo, ma è richiesto l'utilizzo specifico del modulo ES910.3 e di un HW real time di National Instruments.

7.2.2 Applicazioni

Con queste metodologie è stato possibile lo sviluppo di strategie software in grado di regolare l'apporto del BSG al powertrain e verificarne l'effettivo beneficio in vettura in termini di emissioni ed in particolare di consumi su vetture prototipo. Si è preveduto a replicare un ciclo di omologazione alla ricerca dell'effettiva riduzione degli inquinanti. Lo stesso approccio di verifica dei benefici di una soluzione piuttosto che un'altra è stato utilizzato anche per altre tipologie di applicazioni, tra cui citiamo:

- Ottimizzazione della strategia di regolazione della temperatura del liquido di raffreddamento a seguito della modifica di un componente nel motore. Per questa prima applicazione si è utilizzato l'approccio con ASAP3. Di fatti, le dinamiche del sistema controllato sono compatibili con quelle descritte in precedenza.
- HEM (Half Engine Mode): nel caso di motori a due bancate, è possibile per particolari configurazioni motore, prevedere lo spegnimento dei cilindri di una bancata, compensando la coppia mancante con l'aumento di carico della bancata ancora attiva. Se queste strategie in fase di ciclo di omologazione possono portare a qualche noto beneficio dal punto di visti dei consumi, sono state testate anche per verificare gli effetti in fase di ottimizzazione del comportamento della vettura in particolari condizioni di funzionamento a regimi motori instabili.

7.3 Sistemi RCP per test layout motore.

In fase di definizione di strategie di sviluppo motore è fondamentale testare preliminarmente configurazioni innovative prima di prevederne l'integrazione vera e propria. In merito a questo, la collaborazione fra Alma Automotive e il gruppo R&D di Alfa Romeo-Maserati, ha riguardato lo sviluppo di sistemi RCP (rapid control prototyping) per eseguire studi di fattibilità di particolari configurazioni motore per testarne preliminarmente le caratteristiche. Nel corso del periodo di ricerca, sono state sviluppate diverse attività che, seppur riguardanti parti del motore e layout fra loro differenti, prevedono tutte caratteristiche comparabili.

- Per ogni architettura è previsto l'utilizzo di hardware National Instruments, le cui caratteristiche possono variare a seconda delle specifiche del sistema controllato. In particolare dal tipo di segnale da generare o dal bus di comunicazione richiesto per la comunicazione con l'esterno.
- Lo sviluppo di algoritmi di controllo più o meno complessi e adattati sulle caratteristiche di test sul motore. In particolare è necessario subordinare le strategie di controllo al sistema che si vuole testare e implementare strategie di controllo che presentano una forte interazione con la ECU.
- Vengono poi sviluppate le parti di comunicazione protocollata al fine di collegare l'architettura HW e SW sviluppata con i sistemi di controllo (come la ECU) e con le possibili interfacce di comunicazione.

Per rendere meglio l'idea del lavoro svolto, siccome l'elaborato tratta delle metodologie si supporto alla sperimentazione, più che alle attività di sperimentazione stessa, oltre al fatto che molte di esse non possono essere trattate nei dettagli per richiesta del gruppo Alfa Romeo-Maserati, si parlerà in particolare dell'applicazione Twin Spaek, che racchiude un po' tutte le problematiche affrontate in questo tipo di approccio.

7.4 Twin Spark

7.4.1 Effetti di combustioni multiple sull'efficienza di combustione.

Lo studio del comportamento e dell'evoluzione della combustione in camera in applicazioni con multi accensione, per applicazioni PFI e GDI fa riferimento a vari studi che dimostrano, come in particolare per motori a alta potenza specifica e a bassi carichi e regimi, l'utilizzo di accensioni multiple può rappresentare un importante approccio alla

riduzione dei consumi e alla diminuzione degli inquinanti . I maggiori benefici ottenibili dall'ottimizzazione di un sistema multi spark possono essere riassunti come:

- riduzione delle mancate o parziali combustioni,
- riduzione dell'instabilità di combustione,
- riduzione degli HC (dovuta alla riduzione del numero di mancate combustione),
- riduzione degli NOx (dovuto principalmente alla possibilità di aumentare la quantità di ricircolo del gas: EGR),
- significativa riduzione del consumo in particolare con bassi valori di PME (0.5-1 bar) e durante procedure di avviamento a freddo.

L'obiettivo dell'ottimizzazione di un approccio multi accensione è quello di prevedere l'inizializzazione del nucleo della combustione con la prima accensione, per poi aggiungere energia supplementare attraverso la successiva (o successive) accensione, evitando ritardi nella propagazione del fronte di fiamma che potrebbero causare una riduzione della pressione media indicata. Come anticipato in precedenza, le multi-accensioni hanno l'obiettivo di migliorare le prestazioni in condizioni di bassa stabilità di combustione e mancate combustioni, tipicamente a bassi carichi e bassi regimi motore. Nel caso invece di un aumento del carico non si riscontrano particolari benefici. Se si prendono ad esempio motori ad alte performance, l'analisi statistica dei parametri indicating potrebbe mostrare una maggiore tendenza alla detonazione in sistemi equipaggiati con due candele di accensione [13]. In particolare, la presenza di due candele può velocizzare la propagazione del fronte di fiamma, ma nel caso in cui non riesca a favorire il raggiungimento delle zone più lontane della camera di combustione, potrebbe anche portare ad un aumento della tendenza alla detonazione, a causa del più rapido incremento di pressione (e di temperatura) associato al doppio fronte di fiamma.

7.4.2 Caratteristiche del sistema.

L'obiettivo del sistema è stato lo sviluppo di un sistema RCP (Twin Spark) che permetta di comandare l'attuazione di candele addizionali inizialmente non previste nel layout motore, attraverso delle bobine prototipali con drive di potenza integrato. Si è quindi sviluppata un'applicazione prototipale per permettere di testare i benefici sull'andamento della combustione in camera, derivati dall'aggiunta di una seconda accensione comandata. Per il comando di tali bobine si è previsto il pilotaggio attraverso l'utilizzo di hardware esterno.

I requisiti del sistema erano infatti la generazione dei segnali di comando delle bobine aggiuntive fasati con la posizione dell'albero motore, partendo dal valore di anticipo di accensione (SA) delle bobine di serie dalla ECU. Riassumendo per punti i requisiti del sistema sono elencati sotto:

- Generazione di segnali di comando delle bobine slave, fasate con la posizione dell'albero motore (acquisizione del segnale motore SMOT e del segnale di fase SCAM).
- Il valore dell'angolo di attuazione delle bobine addizionali deve dipendere dal valore di anticipo (SA) delle bobine di serie (ECU master).
- Il valore di anticipo delle bobine slave deve essere recepito e attuato entro e non oltre il ciclo motore successivo.
- La calibrazione del valore utente di anticipo deve poter essere calibrabile con una mappa Rpm-Carico attraverso l'utilizzo del SW INCA.

Vari possibili modalità sono state valutate, in merito alla modalità con cui ottenere il valore di anticipo definito dalla ECU:

1. Replicare la catena di calcolo della centralina principale acquisendo i parametri necessari per il calcolo.
2. Spiare elettricamente le attuazioni di accensione, rilevarne la posizione angolare, compensando il ritardo di acquisizione, per poi replicarle, con lo scostamento definito dall'utente, sulle bobine secondarie al ciclo successivo.
3. Recepire il valore di anticipo attuato dalla centralina via bus (iLinkRT).

La soluzione scelta è stata l'ultima in quanto permette di avere un layout più semplice, senza problemi di affidabilità dovuti a replicare una funzione di controllo comunque complessa.

Il sistema nella sua complessità prevede lo sviluppo di tre parti software: l'algoritmo di fasatura e comando delle bobine; Il codice per la gestione del protocollo iLinkRT per la comunicazione con la ECU e il protocollo XCP per l'interfaccia esterna verso il software INCA. Di seguito le varie fasi del lavoro.

7.4.3 Definizione del layout del sistema

La prima parte ha riguardato la sintesi dei segnali necessari a soddisfare gli obiettivi descritti in precedenza (*figura 2.6*). L'architettura dell'applicazione, di per sé complessa, prevede infatti in contemporanea l'acquisizione e la generazione di segnali ad alta frequenza oltre che la necessità di avere la possibilità di poter utilizzare protocolli di

comunicazione implementabili via ethernet. La scelta dell'hardware è volta su Miracle, hardware fornito da Alma Automotive, in quanto oltre a essere programmabile in Labview con accesso alle FPGA (vedi capitolo 2). Questo modulo, compatto e imbarcabile permette la gestione di input/output mediante l'implementazione di strategie di controllo programmabili.

Segnale	Nome	Tipologia	Freq
Segnale Motore	SMOT	Digital Input	40MHz
Segnale Fase	SCAM	Digital Input	40MHz
Attuazione 1	DO1	2Digital Output	40MHz
Attuazione 2	DO2	2Digital Output	40MHz
Attuazione 3	DO3	2Digital Output	40MHz
Attuazione 4	DO4	2Digital Output	40MHz
Attuazione 5	DO5	2Digital Output	40MHz
Attuazione 6	DO6	2Digital Output	40MHz

Figura 7.9: Layout del sistema. Per semplificare il layout si è mostrato solo l'attuazione di una sola delle bobine slave

Una problematica che si è riscontrata è stata quella di realizzare un cablaggio diverso da quello standard per il comando delle bobine ausiliare. Questa problematica è dovuta alla combinazione di valori di impedenza tra il segnale di comando della bobina e l'output digitale di Miracle2 (DO) che abbassando il valore di tensione di output permetteva la commutazione delle bobine. Visto il numero di DO (16) di Miracle2 si è deciso di mettere in parallelo due uscite in modo che, raddoppiando l'impedenza, la tensione sia sufficiente alla commutazione.

7.4.4 Sviluppo algoritmi di controllo

L'algoritmo di controllo ha previsto diversi step: il primo è consistito nello sviluppo dell'algoritmo di fasatura del sistema con il motore, attraverso l'acquisizione dei segnali di ruota fonica e fase. A questa parte si è poi aggiunta la parte di gestione della posizione angolare delle attuazioni considerando che il valore angolare deve essere recepito il più a ridosso possibile dell'attuazione stessa in quanto dipende dal valore che viene ricevuto dalla ECU e non è calcolato direttamente dall'algoritmo di controllo. Una specifica del sistema infatti è quella di eseguire la seconda attuazione al massimo all'interno del ciclo motore successivo rispetto al valore implementato dalla centralina principale.

Il secondo step ha previsto l'integrazione delle parti di comunicazione Real Time, già precedentemente descritte, cioè l'utilizzo del modulo ES910.3 e del protocollo iLinkRT. L'ultima parte software è stata sviluppata con la collaborazione dell'ing. Paolo Bovicelli ed ha previsto l'inserimento parti di codice che permettono l'interazione di Miracle2 con INCA attraverso il protocollo XCP (standard ASAM, vedi capitoloi 4 e 8). In questo modo INCA riconosce il sistema come device connesso via XCP e può quindi interfacciarsi e essere gestita direttamente da INCA. Ogni configurazione, quindi, può essere direttamente impostata dal calibratore su INCA e inviata a Miracle2 che come HW slave recepisce i comandi demandati dall'utente. Questa funzionalità si è ressa necessaria per permettere a utenti di sala prove, di utilizzare l'applicazione e ha permesso di non appesantire il layout con l'aggiunta eventuale di un altro pc di gestione dedicato.

I segnali motore (SMOT e SCAM) e l'algoritmo di fasatura: i sensori di misura della posizione motore e dell'albero a camme, utilizzati nei motori su cui testare l'applicazione, sono ad effetto hall (*figura 7.10*).

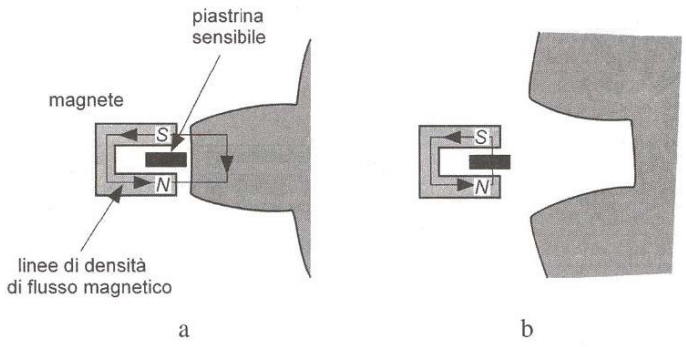


Figura 7.10: Sensore ad effetto Hall

Il sensore è costituito da un magnete permanente, che porta fra i poli l'elemento sensibile, disposto in prossimità di una ruota fonica, la quale rotazione modula il campo magnetico sviluppato dallo stesso magnete [14]. L'effetto Hall fu osservato per la prima volta da Eduard H. Hall nel 1879. Secondo questo fenomeno le proprietà elettriche di alcuni materiali dipendono dal campo magnetico al quale sono esposte. Si consideri a tal proposito una lamina, composta da un conduttore, o da un semiconduttore, attraversata da una corrente, i, immersa in un campo magnetico, generato dalla induzione magnetica B, come rappresentato in *figura 7.11*. La lamina ha una larghezza D, uno spessore S, e un'altezza L; la corrente, i, che l'attraversa, è generata da una forza elettromotrice $\overrightarrow{e_y}$.

Le cariche, q, attraversando il campo magnetico, $\overrightarrow{B_z}$, con una velocità $\overrightarrow{c_y}$,
sono soggette alla forza di Lorenz:

$$\overrightarrow{F_B} = q \cdot \overrightarrow{c_y} \wedge \overrightarrow{B_z}$$

Se i vettori e $\overrightarrow{c_y}$ e $\overrightarrow{B_z}$, sono fra loro ortogonali, la forza $\overrightarrow{F_B}$, per la regola della mano destra, ha la direzione dell'asse x (con riferimento alla *figura 7.11*) e il verso opposto, per cui le cariche elettriche positive si sano sulla faccia a della lamina, mentre quelle negative sulla faccia b, come mostrato in figura.

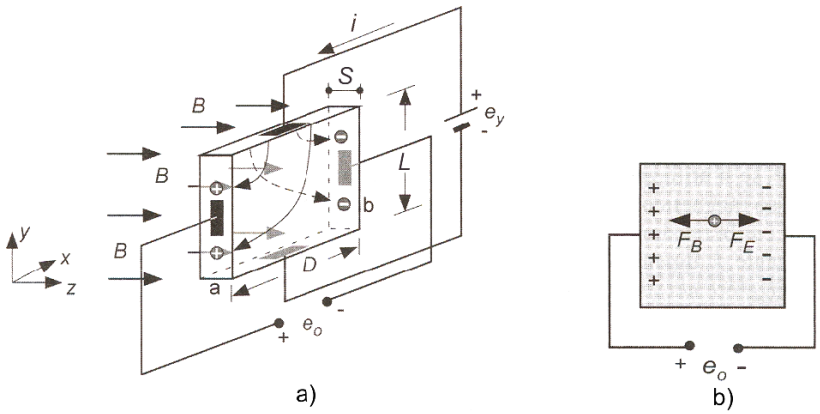


Figura 7.11: Schematizzazione di un sensore ad effetto Hall

Questo addensamento di cariche sulle due facce opposte genera all'interno della lamina un campo elettrico, $\overrightarrow{E_x}$, di intensità crescente, man mano che le cariche elettriche si accumulano. Questo campo elettrico esercita su queste cariche una forza, $\overrightarrow{F_E}$, che si oppone a quella prodotta dal campo magnetico. Il fenomeno dell'addensamento delle cariche termina, quando l'intensità della forza elettrica uguaglia quella magnetica, cioè:

$$\overrightarrow{F_E} = \overrightarrow{F_B} \Longrightarrow q \cdot E_x = q \cdot c_y \cdot B$$

Tenendo conto della relazione che lega la velocità, c_y, delle cariche alla mobilità, μ, delle cariche stesse:

$$\mu = \frac{c_y}{E_y}$$

e delle relazioni che legano il campo elettrico alle dimensioni della lamina:

$$E_y = \frac{e_y}{L} \quad E_x = \frac{e_o}{D}$$

Si ottiene:

$$e_0 = \left(\mu \cdot \frac{e_y}{L} \cdot D\right) \cdot B = k \cdot B$$

La costante, k, dipende dalla mobilità delle cariche elettriche e dato che questa grandezza è molto maggiore nei semiconduttori, rispetto ai conduttori, nei sensori ad effetto Hall si utilizzano semiconduttori. È un sensore che necessita di alimentazione e che, per sua intrinseca natura, si interfaccia molto bene con dispositivi elettronici. Il segnale in uscita infatti, ha forma d'onda quadra con tensione compatibile TTL.
Dopo la descrizione del sensore utilizzato si elencherà in maniera sintetica la parte di algoritmo di controllo. Questa parte del codice si occupa di acquisire il quadro segnali del motore e di ottenere la posizione angolare di ogni riferimento con la massima accuratezza possibile. Il ciclo di campionamento lavora alla massima frequenza possibile per la piattaforma hardware scelta, ovvero 40MHz, a cui corrisponde una risoluzione di 25ns. L'algoritmo consiste nell'incrementare un contatore finché non si verifica la condizione di inizio nuovo ciclo, a quel punto esso viene azzerato. Il riconoscimento del ciclo, si ottiene monitorando il valore del segnale di fase scelto (camma aspirazione o scarico) nel punto in cui viene riconosciuta la cava della ruota fonica. Inoltre, nel nostro caso, avendo a disposizione un layout che prevede una ruota fonica con denti mancanti, occorre ricostruire la posizione dei riferimenti mancanti in modo da non avere discontinuità nella risoluzione angolare. Le uscite del modulo di fasatura sono la posizione dei riferimenti angolari, gli istanti in cui l'albero motore raggiunge tali posizioni e il tempo impiegato dal motore a ruotare della distanza angolare tra gli ultimi due riferimenti.

Algoritmo di calcolo della posizione e del timing delle attuazioni. L'approccio classico per il calcolo del valore di anticipo in un motore a combustione interna è quello descritto in *figura 7.12*. Le bobine devono essere connesse alla piena tensione della batteria per un tempo di circa 4ms per avere sufficiente energia per l'accensione, ma se la bobina è connessa alla batteria per più di 10 ms potrebbe avere danni irreversibili [15].

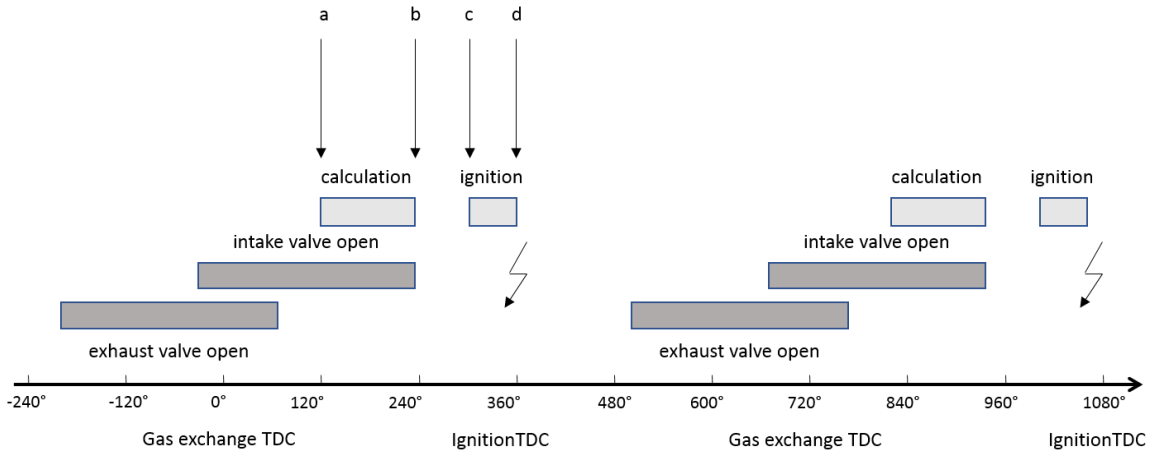


Figura 7.12: Diagramma temporale per l'accensione (ignition).

- Al punto a, le misure del motore necessarie alla determinazione del tempo di accensione (ignition time) sono inviate dalla ECU.
- I calcoli sono eseguiti fino al punto b, nel momento in cui i valori sono inviati alla unità di processo del timing dell'attuazione
- Successivamente, fino all'inizio dell'attuazione vera e propria, vi è un ulteriore delay che può essere calcolato, essendo τ la variabile tempo, come

$$\tau_{b)-c)} = \tau_{calculation} + \frac{1}{2} \cdot \tau_{sample}$$

  Al punto c, la bobina è connessa al terminale della batteria.
- In d, l'angolo di ignition (spark advance), il circuito è aperto provocando l'inizio dell'accensione. Il tempo tra c e d è l'angolo di attuazione. Ognuna delle variabili è scelta in modo che siano ottimizzate le condizioni termodinamiche della combustione.

Cambiare il tempo tra c e d è possibile con alcuni limiti. Il calcolo dell'anticipo deve tenere conto della velocità motore, la massa d'aria nel cilindro, l'insorgenza della detonazione nella combustione precedente, etc. Il caso di mancata accensione deve essere evitato perché tali misfire possono causare danni al TWC (Three-Way Catalyst). Allo stesso modo incrementare troppo l'anticipo di accensione può essere critico a causa di problemi di detonazione.

Tenendo conto dei metodi descritti in precedenza sul calcolo dell'anticipo nei sistemi classici la logica di controllo delle bobine ausiliarie è stata implementata tenendo conto delle specifiche e delle caratteristiche del sistema.

- Il valore di anticipo non è calcolato direttamente dal nostro algoritmo, ma è recepito dalla ECU a frequenze che possono essere a tempo costante (esempio 4ms per ECU Magneti Marelli e 10ms per ECU Bosch) o base evento (ogni PMS).
- Al valore recepito da centralina, deve essere applicato un offset impostabile da interfaccia dall' utente secondo una mappa giri e carico in un range di +/-30° con risoluzione angolare di 0.5°
- Il valore di angolo di attuazione della bobina slave, relativa a ogni cilindro e con l'aggiunta dell'offset definito dall'utente, deve essere recepito entro e non oltre il ciclo successivo rispetto al valore nativo inviato dalla ECU.

È chiaro che per un'implementazione precisa della strategia, in particolare in fase di transitori motore (in cui il valore di anticipo può cambiare in maniera marcata da ciclo a ciclo), la soluzione ottimale prevede che il valore attuato dalle bobine ausiliare sia quello del medesimo ciclo. Questa situazione è mostrata in *figura 7.13*.

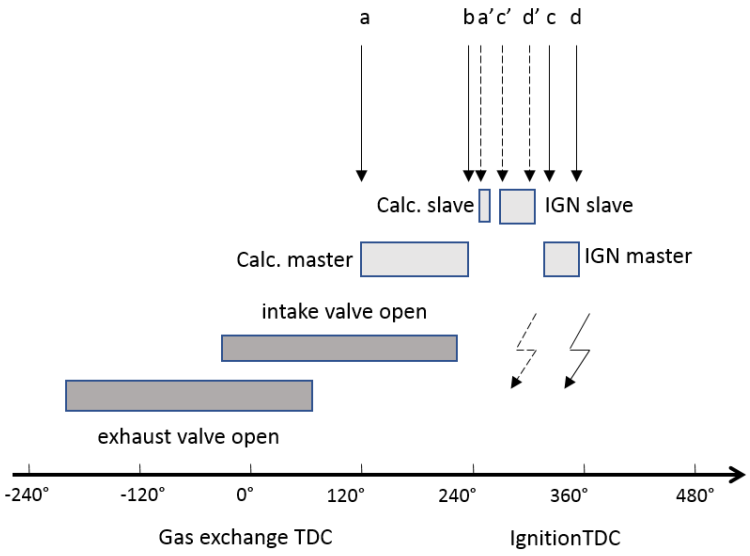


Figura 7.13: Schema che mostra l'esecuzione della seconda attuazione nel caso peggiore, cioè in cui la seconda accensione debba essere anticipato rispetto a quella principale.

Rispetto al layout di *figura 7.12* in *figura 7.13* abbiamo:

- Nel punto a' rileviamo il valore di anticipo presente su bus e proveniente da centralina. È chiaro che spostandosi, con la seconda attuazione, verso il TDC è più probabile che il valore sia quello aggiornato e relativo al ciclo in esame e non a quello precedente. La durata del calcolo è limitata (valori di circa 0.3 ms) in quanto prevede solamente l'associazione del campione angolare agli istanti di inizio e fine carica.

- Nel punto c'abbiamo l'inizio dell'attuazione di accensione.
- Come nel sistema base, d' è la posizione in cui il circuito è aperto provocando l'inizio dell'accensione.

Riassumendo, riusciamo a rimanere all'interno del ciclo di calcolo se riusciamo a rilevare il valore di attuazione della centralina master prima dell'inizio del loop di calcolo della delle seconde attuazioni (punto a'). In *figura 7.14* è mostrato nel dettaglio questo aspetto oltre che il calcolo del punto di inizio e fine carica bobina. Nel calcolo del punto di inizio c' è previsto un certo valore angolare (safety limit) per compensare il ritardo di attuazione, descritto in precedenza.

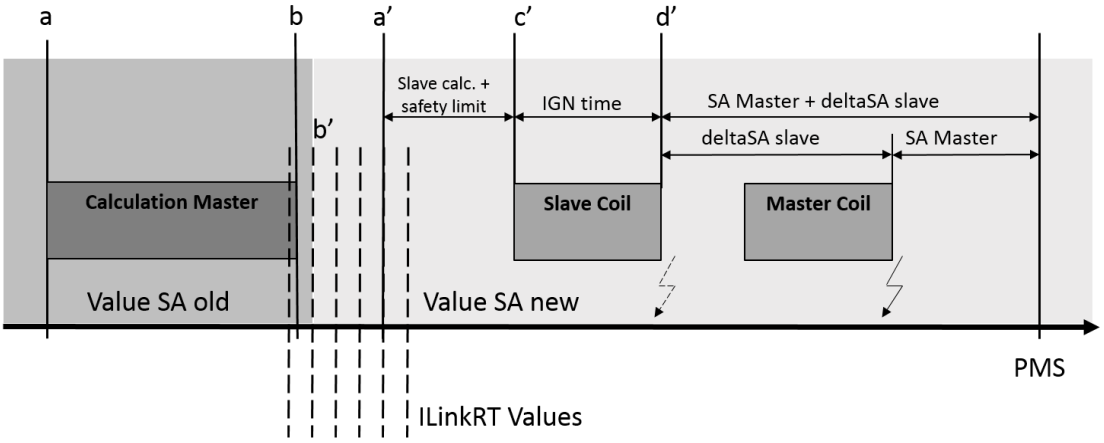


Figura 7.14: Layout del calcolo delle bobine slave. In aggiunta agli elementi di calcolo delle attuazioni, anche in valori (tratteggiati) recepiti via bus. Si identifica b' che il punto dal quale abbiamo l'update dei valori

Anticipando i risultati ottenuti si vede come il valore limite per "recepire" il valore di anticipo dipende dal regime, dal valore di anticipo della master e dall'offset della attuazione secondaria (deltaSA slave).

Il primo aspetto è dovuto all'asincronia fra un sistema evento (base angolo) e un sistema a tempo costante, nel nostro caso, il tempo di carica bobina (4ms) sommato al tempo di calcolo. Infatti a regimi motore elevati, con riferimento alla figura 7.14, il punto a' si sposta verso sinistra superando il limite b', che è quello limite per essere recepito dal sistema. Gli altri fattori che ci portano a superare il limite b' sono ovviamente i valori di SA master e deltaSA slave, che a parità delle altre condizione, di fatto avvicinano il valore di d' a b'. Un altro fattore che va considerato è che per regimi elevati l'anticipo di accensione, compatibilmente con il fenomeno della detonazione, tende ad aumentare e quindi gli effetti descritti in precedenza tendono a sommarsi.

8 Gestione interfaccia e comandi utente

Questo capitolo è trasversale a tutte le applicazioni descritte. In particolare si pone risalto sulla parte di gestione dei parametri per le varie applicazioni. Sistemi integrati di comunicazione fra vari attori presenti in un contesto come quello della sala prove, ma che prevedono la gestione da parte dell'utente di specifiche impostazioni, devono poter essere utilizzati senza prevedere particolari modifiche al layout già presente. Questa esigenza è rilevante, in particolare, per permettere di avere strumenti che nonostante la complessità degli algoritmi, siano semplici da utilizzare in ambiente industriali tipici di una sala prove motore. Durante le attività descritte in questo elaborato, lo sviluppo ha previsto la creazione di interfacce dedicate a ogni applicazione e sviluppate in LabVIEW. Questo approccio, se da un lato si è dimostrato "comodo" per l'implementazione di strategie dedicate alle esigenze di programmazione, dall'altro ha mostrato dei limiti per l'effettivo utilizzo quotidiano delle applicazioni per alcune ragioni:

- La necessità di avere un pc dedicato in cui sono installati tutti i pacchetti necessari all'esecuzione degli eseguibili creati.
- La necessità di istruire gli utilizzatori sulle nuove interfacce.

L'idea di fondo che sta alla base di questo nuovo approccio, è quella di utilizzare interfacce standard in sala prove, sviluppate in ambito commerciale per gestire gli Hardware adibiti all'esecuzione delle applicazioni sviluppate, senza l'aggiunta di ulteriori software, che, per quanto ottimizzati, rappresentano sempre un problema in ambito dell'esecuzione di test in sala prove.

Il software adibito alla gestione di queste istanza è INCA. Infatti INCA, ampiamente descritto in precedenza, prevede la gestione di misure e calibrazioni su ECU, ma in parallelo permette anche la gestione di altri dispositivi che scambiano dati con il PC Host con protocolli standard.

L'idea è quindi quella di far gestire ad INCA ogni applicazione (ad esempio Twin Spark o il DOE), attraverso il protocollo XCP standard, descritto in precedenza.

- Si aggiunge una comunicazione Ethernet XCP.
- L'utente carica i files a2l e hex specifici.
- Si verifica la comunicazione con l'HW.
- Si importa l'interfaccia utente predefinita.

8.1 Sviluppo protocollo di comunicazione.

Il protocollo utilizzato è l'ASAM XCP (descritto nei capitoli precedenti). In pratica si sono dovute sviluppare parti di codice per rispondere ai comandi che il master, INCA, invia allo slave (RCP, DOE, ecc), in modo che i due sistemi possano scambiarsi informazioni. Questa parte, sviluppata in collaborazione con il collega Paolo Bovicelli, ha prevista diversi step:

- Creazione di parti di codice che sono eseguite in parallelo ai vari algoritmi.
- Sviluppo di strumenti in grado di creare in maniera automatica a2l e hex file per l'importazione su INCA.
- Modifica delle logiche software dedicate per l'integrazione fra i due strumenti.

Il primo punto ha richiesto uno studio approfondito dello standard, per definire come interpretare e impacchettare i dati da master a slave. Il protocollo è sviluppato per utilizzare il protocollo udp/ip come vettore.
Il secondo punto è di fondamentale importanza. Nell'a2l devono essere implementati tutte le codifiche sul protocollo eseguito. Come per l'iLinkRT, che si base su questo standard, sono molteplici i parametri da impostare per l'utilizzo di questo protocollo (ad esempio: Address Granularity (AG), CTO Packet Code and absolute ODT number,ecc) specifici per l'implementazione scelta. La necessità di creare un hex file poi ha previsto l'esecuzione di codici che permettano di identificare le variabili definite come indirizzi fisici di memoria e salvarle in formato binario.
La terza parte ha previsto una parziale modifica dell'implementazione degli algoritmi, inserendo alcuni accorgimenti per rendere, i software sviluppati, compatibili con questo approccio. Per adattare la gestione dei software alle nuove esigenze e per la gestione di nuove esigenze, si elencano gli accorgimenti da tenere in considerazione per l o sviluppo di questo approccio:

- Dimensioni di vettori e mappe fisse. Essendo un protocollo implementato per la gestione di centraline motore, le dimensioni di ogni variabile devono essere fissate a priori, perché ad esse deve essere associato uno specifico e predefinito indirizzo di memoria.

- Il flusso di dati è staticamente definito, nel senso che le variabili che vanno da Slave a Master sono le measure, cioè paragonabili alle acquisizioni di centralina, mentre i parametri implementati da Master a Slave sono identificate come calibration. Questo, se da un lato, può apparire logico, dall'altro complica la struttura, in quanto le logiche di controllo sviluppate in precedenza, erano state sviluppate per una diversa interazione fra i due flussi.

- Gli stati di comando non possono essere transitori. Per semplificare la comprensione di questo concetto, si può immaginare i comandi booleani (governati da uno stato di spento 0 e acceso 1), come pulsanti fisici. In INCA, a differenza delle interfacce sviluppate Labview, i 'pulsanti' non sono gestiti con ritorno a molla, ma gli unici comandi implementabili sono selettori di stato.

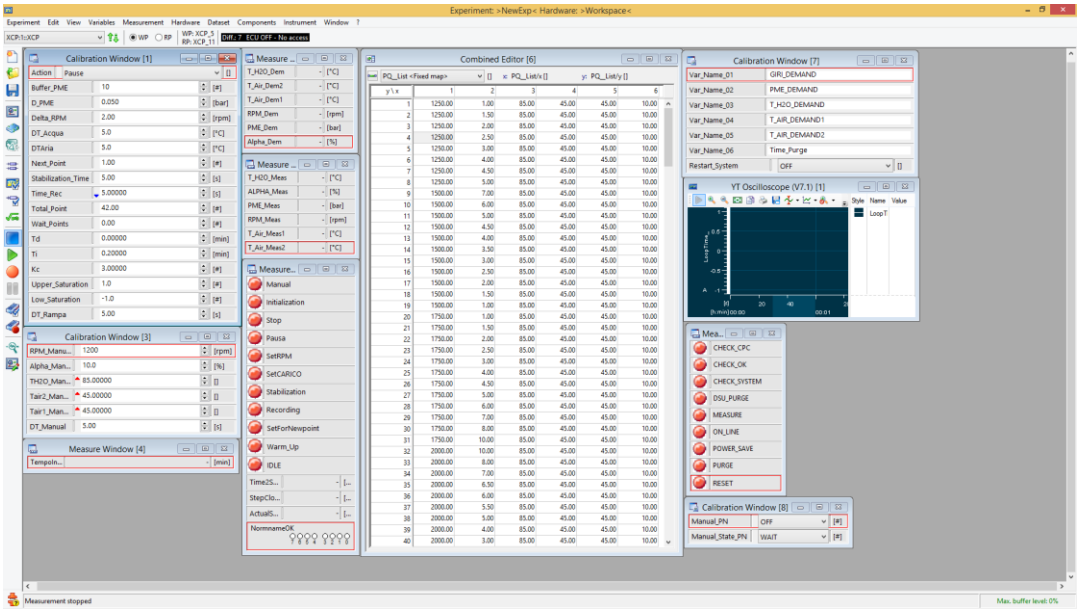


Figura 8.1: Esempio interfaccia gestione PQ.

A seguito di un lavoro di revisione e di adattamento delle parti di codice, tutte le applicazioni sviluppate, vengono ora gestite con questa modalità.

Altri vantaggi di questo approccio, riguardano la possibilità di registrare i dati senza lo sviluppo di parti di codice di salvataggio, direttamente sfruttando le logiche di registrazione dati da INCA. Si è verificato, in particolare per i sistemi RCP, che possono essere scambiati dati a frequenze elevate (ogni combustione motore), senza la perdita di pacchetti di dati.

9 Conclusioni.

L'attività svolta in questi anni di ricerca, ha portato alla creazione di un sistema integrato presente in tutte le sale prove motore di Maserati Alfa Romeo, che ha contribuito e richiesto lo sviluppo di queste metodologie. In generale, al di là di ogni specifica applicazione, il maggiore risultato è la creazione di un sistema in grado di gestire completamente la sala prove. Il valore aggiunto di questa attività è infatti quello di avere un sistema aperto, esterno rispetto alle toolchain per il controllo della strumentazione, dell'automazione di cella e del motore, in cui è possibile elaborare delle strategie di controllo complesse, lasciando le specifiche operazioni di sala alle caratteristiche di ogni dispositivo. Infatti, una volta sviluppate le parti di codice necessarie alla gestione di ogni singolo "interlocutore" presente in sala prove, è possibile considerare i dati da ogni fonte come parte di un flusso di dati motore disponibile alle massime frequenze possibili da ogni acquisitore. In questo modo si è potuto elaborare strategie a supporto della calibrazione e della sperimentazione con la possibilità di intervenire con azioni a frequenza comparabile a quella di ciclo motore. Analizzando nello specifico le applicazioni descritte, possiamo riassumere:

- Gestione anticipo e detonazione. Le parti di codice implementate hanno permesso la creazione di un loop per il controllo dell'anticipo in retroazione. Sono state sviluppate strategie per il mantenimento del sistema entro limiti di detonazione considerati accettabile, ma anche la gestione dell'anticipo per l'ottenimento di valori del baricentro della combustione definiti ottimali dal calibratore. Il risultato più importante ottenuto con queste metodologie è la possibilità di testare particolare condizioni motore, che possono influire sulla detonazione (titolo della miscela, pressione di sovralimentazione, ecc) avendo un supervisore del fenomeno in grado di mettere il motore in sicurezza senza portarlo in punti di funzionamento differenti da quelli da testare.
- Sistema automazione: partendo dalla necessità di gestire in automatico prove solitamente effettuate in manuale per risparmiare tempo dedicato alla sperimentazione e migliorare l'affidabilità dei risultati, si è arrivati a un'applicazione in grado di supervisionare nel suo complesso ogni azione motore. Il sistema è quindi in grado di recepire i punti da effettuare da un sistema di ottimizzatore delle prove (DOE) cambiando i parametri motore e centralina, gestendo le temperature del

comburente e del liquido refrigerante in maniera totalmente automatizzata. Il sistema di controllo banco, il sistema di gestione centralina, il sistema di analisi combustione e gli strumenti di analisi gas continuano ad effettuare le operazioni per cui sono state ottimizzati, ma sono subordinati all'azione di un master esterno che ne dirige le operazioni al fine di massimizzare il rendimento della sperimentazione. Oltre al considerevole risparmio di tempo, sono state vantaggiosamente implementate logiche per la valutazione dell'attendibilità delle prove e metodi per la gestione dei limiti in cui considerare la prova accettabile e far partire la registrazione dei dati. Nel sistema è stato sviluppato e calibrato un controllo in retroazione per la gestione dei valori di PME in modo che rientrino entro certe soglie di accettabilità. A questo va aggiunta la funzione di collettore di dati: ogni dato disponibile su TAC viene condiviso con tutti gli altri strumenti utilizzati, in modo che possano essere acquisiti e rapidamente verificati in fase di post analisi.

- Sistemi di Bypass funzionale e RCP: Con queste metodologie sono stati creati gli algoritmi di controllo prototipali, per la gestione di gran parte delle innovazioni presenti nei motopropulsori di ultima generazione Alfa Romeo Maserati. Gli algoritmi di controllo sviluppati hanno mostrato in particolare tre caratteristiche fondamentali per la prototipazione:

  1. La possibilità di gestire modelli e algoritmi a frequenze elevate, in grado di gestire frequenze di calcolo compatibili con le frequenze di calcolo della ECU, per la gestione di dispositivi (ad esempio il BSG) presenti nel motore e di cui ancora non si è sviluppato il controllo.
  2. L'integrabilità di questi sistemi con le applicazioni standard del mondo automotive. In particolare per i sistemi di bypass funzionale: la possibilità di modellare algoritmi customizzati dall'utente e poterli verificare direttamente in vettura, mentre per i sistemi di prototipazione la possibilità di gestire le leve di controllo nello stesso ambiente di gestione della ECU
  3. Il fattore di forma, cioè la possibilità di utilizzare architetture HW dedicate che permettessero l'esecuzione di strategie di controllo non solo al banco prova, ma anche in vettura, attraverso l'utilizzo di HW compatti, ma comunque in grado di prevedere lo sviluppo di algoritmi di controllo complessi.

Nel complesso il lavoro illustrato e svolto in questi tre anni, è utilizzato dal gruppo di calibrazione e ricerca e sviluppo delle sale prova motore Maserati Alfa Romee a supporto dell'innovazione e dello sviluppo dei propulsori. Il layout utilizzato, comune alla maggior parte degli ambienti di ricerca e sviluppo industriali e accademici, permette inoltre di utilizzare le metodologie sviluppate in ambiti diversi da quello specifici, illustrati nell'elaborato di tesi.

Definizioni e abbreviazioni.

ACS - Automatic Calibration System
ASAM - Association for Standardization of Automation and Measuring Systems
AuSy - Automation System
BSG - belt starter-generator
CAN - Controller Area Network
COV - Coefficient of variation
cRIO - Compact RIO
DOE - Design of experiment
ECU - Engine control unit
ETK - Emulator TastKopf
FPGA - Field Programmable Gate Array
HEV - Hybrid electric vehicle
HW - Hardware
LV - Labview
MAPO - Maximum Amplitude of Pressure Oscillations
MCI - Motori a combustione interna
MFB50 - Mass Fraction Burned 50%
PME - Pressione media effettiva
PMI - Pressione media indicata
PQ - Piano quotato
PS - Processing system
RCP - Rapid control prototyping
RT - Real time
SW - Software
TAC - Tool for automation and control
VI - Virtual Instruments

Riferimenti bibliografici

[1] Wikipedia, «Field Programmable Gate Array,» 18 01 2017. [Online]. Available: https://it.wikipedia.org/wiki/Field_Programmable_Gate_Array.

[2] National Instruments, «What is a Real-Time Operating System (RTOS)?,» 22 11 2013. [Online]. Available: http://www.ni.com/white-paper/3938/en/.

[3] Alma Automotive, «Products / Rapid Control Prototyping,» [Online]. Available: http://www.alma-automotive.it/en/products/rapid-control-prototyping/miracle2-013.

[4] Wikipedia, «Controller Area Network,» 22 01 2017. [Online]. Available: https://it.wikipedia.org/wiki/Controller_Area_Network.

[5] ETAS, «INCA Base Product,» ETAS, [Online]. Available: https://www.etas.com/en/products/inca-details.php.

[6] ASAM e V., «Overview,» in *ASAM ASAP3 Automation / Optimization Interface for ECU*, 1999, p. 64.

[7] ASAM e.V., «ASAM MCD-1 XCP,» 27 07 2015. [Online]. Available: https://wiki.asam.net/display/STANDARDS/ASAM+MCD-1+XCP.

[8] Wikipedia, «IEEE 754,» 20 03 2017. [Online]. Available: https://it.wikipedia.org/wiki/IEEE_754.

[9] E. Corti, *Dispense di controllo motori a combustione interna,* 2012.

[10] ETAS, «ETAS ASCMO – Data-based Modeling and Model-based Calibration,» 2017. [Online]. Available: https://www.etas.com/en/products/ascmo.php.

[11] D. C. Montgomery, Design and analysis of experiments 5th edition, John Wiley Sons, Inc..

[12] R. Bao, V. Avila e J. Baxter, «Effect of 48 V Mild Hybrid System Layout on Powertrain System Efficiency and Its Potential of Fuel Economy Improvement,» *SAE technical paper*, n. 2017-01-1175, p. 11, 03 2017.

[13] C. Forte, E. Corti, G. M. Bianchi e S. Falfari, «A RANS CFD 3D Methodology for the Evaluation of the Effects of Cycle By Cycle Variation

on Knock Tendency of a High Performance Spark Ignition Engine,» *SAE Technical paper*, n. 2014-01-1223, p. 10, 01 04 2014.

[14] P. M. Azzoni, Strumenti e misure per l'ingegneria meccanica, HOEPLI.

[15] L. Guzzella e C. H. Onder, Introducing to Modeling and Control of Internal Combustion Engine Systems, Springer.

[16] Wikipedia, «Orbix (software),» 22 08 2015. [Online]. Available: https://en.wikipedia.org/wiki/Orbix_(software).

[17] Xilinx, «Zynq-7000 All Programmable SoC First Generation Architecture,» 27 09 2016. [Online].

[18] R. Bosch, Gasoline Engine Management, Wiley.