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 del SOFTWARE di un SISTEMA per l'ANALISI in TEMPO REALE della COMBUSTIONE di MOTORI ENDOTERMICI ALTERNATIVI

Presentata da:

Paolo Bovicelli

Coordinatore Dottorato

Prof. Ing. Nicolò Cavina

Relatore

Prof. Ing. Enrico Corti

Esame finale anno 2017

ABSTRACT

Increasing restrictions concerning exhaust emission, higher fuel economy, higher performance and drivability, brings to the development of increasingly complex engine control algorithms. At the same time, the propulsion unit is becoming a more and more complex group of subsystems that have to work in unison. The engine calibration engineer is faced with a multitude of variables and algorithms to be calibrated and tested, and needs instruments helping him to analyse the engine behaviour and optimize his time by providing synthetic results easily collecting the amount of data. The development of the functionalities of a combustion analysis and diagnosis system has been carried out during this work, in order to generate a software that provides the best solutions for engine analysis, in terms of accuracy of data results, variety of solutions offered for the analyses, ease of use and configuration and integration with other systems by real time results sharing.

INDICE

ELENCO delle FIGURE

ABBREVIAZIONI

Sigla	Significato	Unità
ADC	Analog to Digital Converter	
APmax	Angolo di pressione massima	°CA
CHR	Comulative Heat Release	J
DAC	Digital to Analog Converter	
ECU	Engine Control Unit	
FPGA	Field-Programmable Gate Array	
HIL	Hardware In the Loop	
IMEP	Indicated Mean Effective Pressure	bar
IMEPH	Indicated Mean Effective Pressure High	bar
MAP	Manifold Air Pressure	mbar
MAPO	Maximum Amplitude of Pressure Oscillation	bar
MFB10	Angle of 10% of Mass Fraction Burned	°CA
MFB50	Angle of 50% of Mass Fraction Burned	°CA
MFB90	Angle of 90% of Mass Fraction Burned	°CA
Pmax	Pressione massima	bar
RC	Rapporto di Compressione	
RT	Real-Time	
sbRIO	Single Board RIO, National Instruments	
SOM	Sistem On Module, National Instruments	
TDC	Top Dead Center	°CA

L'evoluzione tecnica dei motori a combustione interna negli ultimi dieci anni è stata soggetta a regolamentazioni sulle emissioni inquinanti ogni anno più stringenti, che hanno portato allo sviluppo di algoritmi e strategie di controllo sempre più complicati. Di conseguenza, il processo di calibrazione del sistema di controllo motore diventa lungo e costoso, ed allo stesso tempo necessita dell'introduzione di un'automazione via via più importante, dato l'incremento delle variabili in gioco su cui è possibile agire. Si pensi ad esempio come l'evoluzione dei motori diesel abbia portato da sistemi a singola iniettata con turbine a geometria fissa fino a sistemi ad iniezioni multiple ad alta pressione con turbine a geometria variabile e sistema di ricircolo dei gas di scarico, o come un downsizing sempre più spinto abbia portato alla necessità di monitorare la detonazione a bordo veicolo e al controllo di più gruppi di sovralimentazione di diverse tipologie e sistemi di distribuzione valvole con fase e/o alzate variabili. Inoltre, visto il peso sempre più importante che viene dato al comfort di guida ed in seguito ai recenti sviluppi in materia di test di omologazione, diventa molto importante rendere possibile il test e la calibrazione a bordo veicolo.

È quindi necessario mettere a disposizione del calibratore strumenti che permettano di semplificare la diagnosi e l'analisi dell'andamento della combustione durante la sperimentazione, al banco ed on board, generando dati sintetici che descrivano il comportamento del motore in tempo reale, in maniera tale da abbattere i tempi di analisi dei dati offline, e che diano la possibilità di utilizzare questi dati durante la prova stessa come input di strategie dinamiche di controllo e calibrazione. Infatti risulta essere di grande interesse poter chiudere il loop di controllo in tempo reale utilizzando i risultati misurati e calcolati, in quanto questo permette di validare una strategia in maniera automatizzata, abbattendo i margini di errore e soprattutto il tempo speso al banco prova o in vettura.

Il lavoro svolto durante i tre anni di dottorato è stato guidato da questa necessità di strumenti di analisi della combustione sempre più sofisticati, facili da usare e che consentano all'utilizzatore di avere sotto controllo prove complesse: partendo da un sistema solido e robusto, sviluppato in precedenti lavori di tesi e dottorato ([1], [2], [8]), si è proseguita l'evoluzione software con l'obbiettivo di arrivare ad un sistema flessibile

e semplice da utilizzare, che metta a disposizione dell'utente gli strumenti necessari per ottimizzare i test sul motore.

Gli sforzi di sviluppo si sono concentrati sull'implementazione di nuove funzionalità in ottica di ampliare la flessibilità del sistema per renderlo facilmente adattabile ad ogni tipo di motore ed interfacciabile con gli strumenti tipici di una sala prove, migliorando la qualità e l'efficienza del calcolo in tempo reale, introducendo nuovi algoritmi di calcolo all'interno del software, e rendendolo infine facilmente utilizzabile da parte di qualsiasi utente, grazie ad un miglioramento dell'interfaccia grafica.

Una particolare attenzione è stata data alla possibilità di integrazione del sistema con il mondo esterno: infatti come già sottolineato, la necessità di calibrazione in tempo reale, e quindi lo sharing dei dati calcolati dal sistema di analisi combustione, diventa sempre più pressante. Inoltre, data la mole di dati che viene generata da una sala prove o da una vettura, diventa quasi indispensabile avere un collettore unico per tutti i dati, ed è quindi necessario che il sistema sviluppato si possa integrare con questi collettori tramite dei protocolli di comunicazione standard.

Inoltre la varietà di dati calcolati è stata migliorata ed ampliata con l'introduzione di algoritmi non strettamente dedicati all'analisi della combustione, ma sviluppati per l'analisi dei segnali che fanno parte del sistema di controllo del motore e che impattano sulla combustione stessa, fornendo all'utente un quadro più dettagliato e maggiori possibilità di testing.

La piattaforma software scelta nei precedenti lavori di dottorato, ha permesso di continuare lo sviluppo senza troppe difficoltà introdotte dall'evoluzione hardware, consentendo così di poter sfruttare un incremento delle prestazioni messe a disposizione da nuove piattaforme hardware senza compromettere la struttura del software, e inoltre ha consentito di concentrarsi sullo sviluppo degli algoritmi di analisi e non soltanto sulla loro integrazione.

Parte fondamentale del lavoro è stata la validazione del sistema, sia in hardware in the loop che al banco prova ed in vettura.

Il sistema sviluppato in questi ultimi tre anni è attualmente in uso a supporto dell'attività di ricerca, nelle sale prove universitarie di Bologna (laboratorio di Macchine di via Terracini) e di Forlì (laboratorio hangar di Macchine e Propulsione).

Nell'esposizione del lavoro svolto si daranno per scontate le nozioni di carattere generale sui motori a combustione interna e sulla combustione, cercando di approfondire solo gli argomenti specifici che si andranno a toccare.

1. LA PIATTAFORMA HARDWARE E SOFTWARE

1.1 Overview hardware

La scelta della piattaforma hardware è stata guidata dalle esigenze di utilizzo del sistema di analisi combustione. In particolare, considerando l'obbiettivo di sviluppare un sistema imbarcabile per l'analisi in tempo reale della combustione ciclo dopo ciclo, i fattori che hanno caratterizzato la scelta sono stati:

- Dimensioni ridotte, per consentire una facile "imbarcabilità" del sistema anche in veicoli con spazi ridotti (e.g. auto sportive o motoveicoli).
- Disporre di una potenza di calcolo sufficiente a svolgere i calcoli in tempo reale, campione dopo campione, il che equivale a dire un hardware che permetta frequenze di calcolo molto superiori a quella di combustione.
- Facilità di programmazione, senza una conoscenza approfondita dei linguaggi di basso livello, così da consentire alla stessa persona che sviluppa l'algoritmo di analisi di integrarlo in maniera relativamente facile all'interno del codice.

Questi fattori hanno portato alla scelta di hardware National Instruments che garantisce frequenze di calcolo elevate (FPGA e real time processor) ed allo stesso tempo un linguaggio di programmazione grafico ed intuitivo, univoco per lo sviluppo del codice FPGA, per la parte real time e per l'interfaccia utente.

La prima piattaforma scelta è stata una Single Board RIO 9606, di cui si riportano di seguito le caratteristiche:

Figura 1-1: Single Board RIO 9606

Riepilogo specifiche

Generale	
Form Factor	Single-Board RIO
Conformità RoHS	Si
I / O Digitale	
Canali bidirezionali	96
Livelli di logica	3,3 V
FPGA riconfigurabile	
FPGA	Spartan-6 LX45
Controller riconfigurabile	
Memoria nonvolatile	512 MB
Memoria di sistema	256 MB
Processore	PowerPC 400 MHz
Elettrico	
Intervallo di tensione in ingresso (sorgente esterna)	9 V - 30 V
Specifiche fisiche	
Lunghezza	10.3 cm
Larghezza	9.65 cm
Temperatura operativa	-40 °C - 85 °C

Figura 1-2: specifiche sbRIO 9606

Questa piattaforma mette a disposizione dello sviluppatore una FPGA ed un processore real time a cui è necessario interfacciare una scheda secondaria per il condizionamento dei segnali, in particolare per il campionamento in digitale di segnali analogici. La scheda secondaria, sviluppata da Alma Automotive, permette grazie ad un ADC esterno il campionamento di 8 segnali analogici a 16bit a 200kHz. L'accoppiamento di queste due schede elettroniche (sbRIO + Alma Automotive board), racchiuse in un case compatto e robusto, ha portato alla nascita del primo hardware per l'analisi indicating on board, OBI-1.

Figura 1-3: OBI-1

Analog input	
Number of channels	8 differential
ADC resolution	16 bit
ADC accuracy	0.1% calibrated
ADC temperature stability	< 15ppm/°C
Sampling Rate	200 kHz, simultaneous sampling
Antialiasing filter	hardware, Butterworth low pass 50kHz 3rd order
Input coupling	DC
Input range	± 10 V
Digital input	
Number of channels	2
VRS input	
Number of channels	2
Sampling Rate	10MHz
Overvoltage protection	200 V
Environmental	
Operating temperature	-40 to 85 °C
Connectivity	
Ethernet	100Mbit
Embedded CAN	
Single channel CAN	2.0 B interface
Termination	Software programmable

Figura 1-4: specifiche OBI-1

Su questa piattaforma è partito un primo lavoro di import degli algoritmi di analisi già sviluppati nei precedenti lavori di dottorato [1], [2], già implementati in ambiente LabVIEW e validati su hardware più ingombranti, dotati di sistema di acquisizione analogico digitale integrato (PXI e Compact RIO). In parallelo è iniziato un lavoro di sviluppo dei driver per la gestione degli ADC esterni e l'implementazione di nuovi algoritmi, nonché lo sviluppo dell'interfaccia software.

Questo ha portato ad ottenere un sistema di analisi combustione in tempo reale in grado di calcolare i parametri tipici dell'analisi combustione (quali IMEP, IMEPH, Pmax, CHR, MFB10, MFB50, MFB90), in grado di analizzare fino a 6 segnali di pressione cilindro contemporaneamente, ed allo stesso tempo di acquisire uno streaming di 8 canali analogici a 200kHz.

Il proseguimento del lavoro di sviluppo sulla scheda Alma Automotive ha portato alla nascita della board Miracle-1, nella quale oltre all'ADC sono stati introdotti DAC (per la generazione di segnali analogici) e memoria SD esterna per lo storage dei dati acquisiti (non ci si soffermerà sullo sviluppo delle board in quanto non riguardano il lavoro svolto durante il dottorato). Si passa così ad un hardware potenziato con un numero maggiore di AIO, racchiuso in un nuovo case, che ha come cuore sempre la sbRIO 9606.

Figura 1-5: Miracle-1

Hardware	
Dimensions	110x110x25mm
Weight	400g
Storage	512 MB on-board + 32 GB Flash
Power requirements	
Power supply range	9 to 30 Vdc
Recommended power supply	12 Vdc
Typical leakage current	~0.3 A
Power consumption	5W typ.
Analog input	
Number of channels	8 differential
ADC resolution	16 bit
ADC accuracy	0.1% calibrated
ADC temperature stability	< 15ppm/°C
Sampling Rate	200 kHz, simultaneous sampling

	Antialiasing filter	hardware, Butterworth low pass 80kHz 3rd order
	Input coupling	DC
	Input range	± 10 V
	Digital input	
	Number of channels	8
	Sampling Rate High Speed[1]	10MHz
	VRS input	
	Number of channels	2
	Sampling Rate	10MHz
	Overvoltage protection	200 V
	Environmental	
	Operating temperature	-40 to 85 °C
	Connectivity	
	Ethernet	100Mbit
	Embedded CAN	
	Dual channel CAN	2.0 B interface
	Termination	Software programmable

Figura 1-6: specifiche Miracle-1

La peculiarità principale di questa nuova versione hardware è la presenza di una memoria esterna da 32GB che permette lo storage di dati ad alta frequenza all'interno del dispositivo stesso.

L'ultimo step evolutivo hardware è stato fatto con l'uscita sul mercato della Single Board RIO 9651 SOM, di cui si riportano in seguito le caratteristiche:

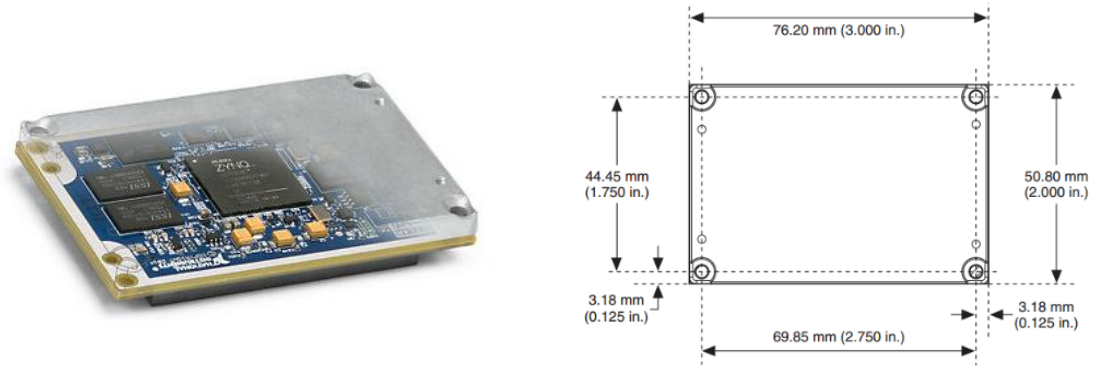


Figura 1-7: sbRIO 9651 SOM

Riepilogo specifiche

Generale	
Form Factor	Single-Board RIO
Operating System / Target	Linux
Conformità RoHS	Si
I / O Digitale	
Canali bidirezionali	160
Livelli di logica	1,8 V 2,5 V 3,3 V
FPGA riconfigurabile	
FPGA	Artix-7
Controller riconfigurabile	
CPU Clock Frequency	667 MHz
Memoria nonvolatile	512 MB
Memoria di sistema	512 MB
Specifiche fisiche	
Lunghezza	76.2 mm
Larghezza	50.8 mm
Temperatura operativa	-40 °C - 85 °C
Peso	100 gram

Figura 1-8: specifiche sbRIO 9651 SOM

Questo nuovo "cervello", dotato di una capacità di calcolo superiore alle precedenti versioni, sia lato real time che FPGA, ha portato ad un ulteriore sviluppo della board Miracle1 verso la board Miracle2, con un netto incremento prestazionale a livello di IO:

Hardware	
Dimensions	105x85x30mm
Weight	400g
Storage	512 MB on-board + 32 GB Flash
Power requirements	
Power supply range	6 to 26 Vdc
Recommended power supply	12 Vdc
Typical leakage current	~0.5 A
Power consumption	6W typ.
Analog input	

Number of channels	**24 differential (2 High Voltage)**
ADC resolution	16 bit
ADC accuracy	0.1% calibrated
ADC temperature stability	< 15ppm/°C
Sampling Rate	**400 kHz, simultaneous sampling**
Antialiasing filter	hardware, Butterworth low pass 100kHz 3rd order
Input coupling	DC
Input range	± 10 V (± 40 V High Voltage)
Overvoltage protection	± 50 V
Overvoltage protection High Voltage	Vin+ ±200 V; Vin- ±14 V
Digital input	
Number of channels	**16 (8 High Speed, 8 Low Speed)**
Sampling Rate High Speed	10MHz
Sampling Rate Low Speed	500kHz
Input logic levels, High Speed	low [0V ~ 0.8V]; high [2V ~ 5V]
Input logic level, Low Spee	low [-12V ~ 0.8V]; high [2V ~ 12V]
Overvoltage protection High Speed	-6 V; +12 V
Overvoltage protection Low Speed	±25 V
VRS input	
Number of channels	2
Sampling Rate	10MHz
Overvoltage protection	200 V
Environmental	
Operating temperature	-40 to 85 °C
Connectivity	
Ethernet	**Gigabit**
Embedded CAN	
Dual channel CAN	2.0 B interface
Termination	Software programmable
Auxiliary sensors	
Accelerometer, magnetometer, gyroscope	9 axis total

Figura 1-9: specifiche Miracle-2

L'incremento delle potenzialità di calcolo e del numero di canali acquisiti ad alta frequenza ha dato ampio margine per l'implementazione di nuove funzionalità e nuovi algoritmi di calcolo legati all'analisi del funzionamento del motore al banco prova o in vettura.

Il lavoro presentato in seguito si è basato su queste piattaforme hardware, cercando di adattarsi alle nuove potenzialità e sfruttarle al meglio per fornire all'utente finale strumenti sempre più evoluti e potenti per l'analisi delle prestazioni di un motore.

Figura 1-10: evoluzione dell'hardware

1.2 Overview software

Per quanto riguarda l'ambiente di sviluppo software, l'intero progetto, da FPGA ad interfaccia utente, è sviluppato in LabVIEW. Questo software consente un facile interfacciamento con hardware National Instrument (i.e. sbRIO) ed un facile sviluppo di codice per via grafica, raccogliendo tutti i livelli di sviluppo all'interno di un unico ambiente.

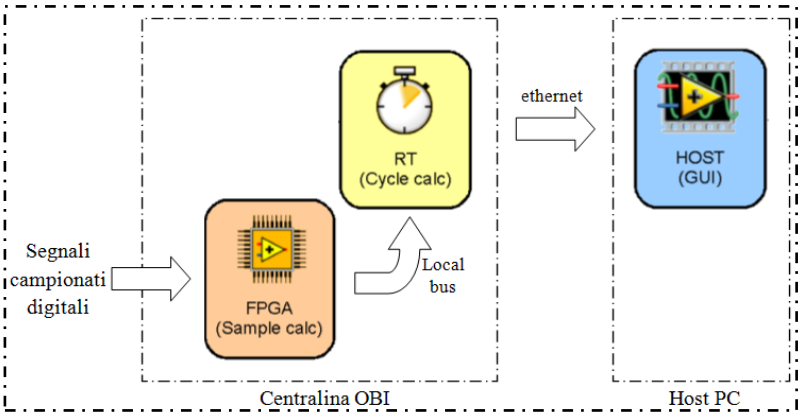


Figura 1-11: livelli di sviluppo in labview

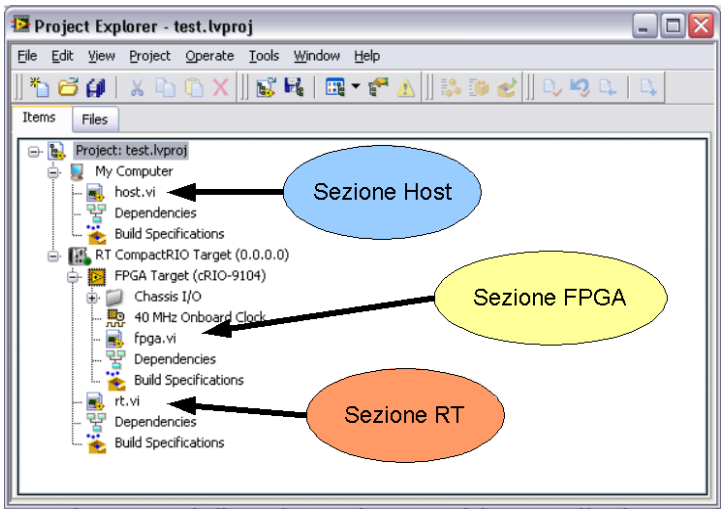


Figura 1-12: overview del progetto labview

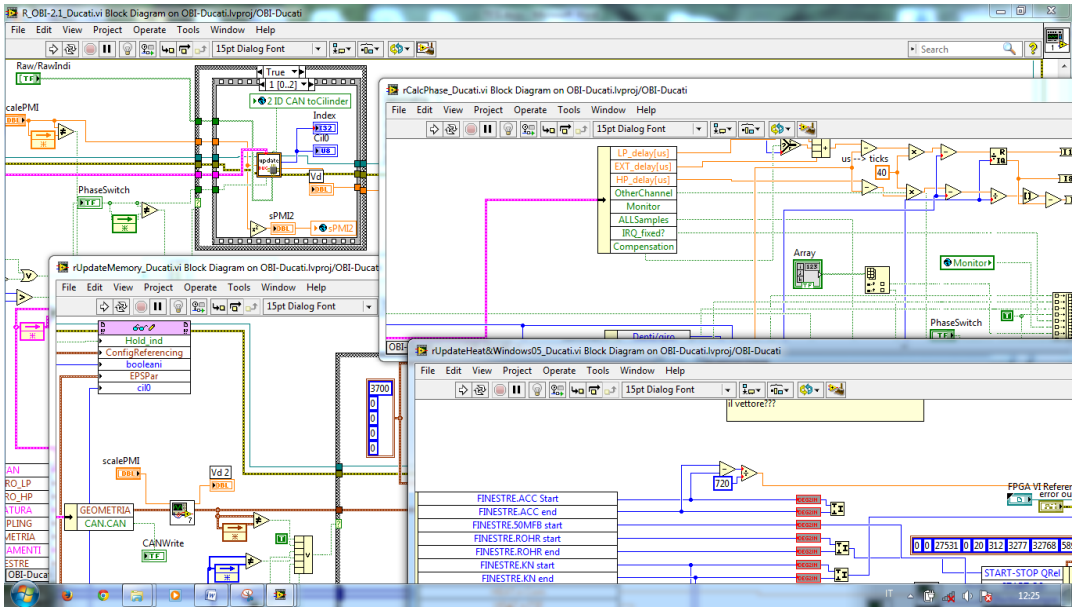


Figura 1-13: ambiente di sviluppo grafico

1.3 Evoluzione del software legata all'evoluzione hardware

L'evoluzione dell'hardware ha inevitabilmente portato ad implementare delle modifiche software per poter adattare il progetto alle nuove piattaforme. L'evoluzione più significativa, non solo a livello di AIO, ma anche a livello di codice è stata quella segnata dal passaggio da piattaforma Miracle-1 a piattaforma Miracle-2. Infatti come esposto in precedenza, i due hardware sono dotati di un sistema operativo real time differente: si passa da Pharlap a LinuxRT. Di conseguenza, nonostante l'ambiente di sviluppo sia rimasto lo stesso, si è dovuto procedere ad un adattamento del software: in taluni casi è

stata sufficiente la sostituzione di librerie software obsolete con librerie aggiornate mentre, in casi più sfortunati, è stato necessario sostituire alcune librerie con nuovi algoritmi creati ad hoc. Un secondo problema è stato la gestione dei file all'interno della memoria della sbRIO SOM e lo scambio di informazioni con l'interfaccia utente, in quanto anche questi sono caratterizzati da nuove modalità legate al cambio di sistema operativo.

Quindi la parte di evoluzione software legata all'hardware (e non allo sviluppo di nuovi algoritmi) non è stata del tutto indolore: questo ha costretto ad una nuova fase di validazione degli algoritmi già presenti e di quelli nuovi introdotti al posto degli obsoleti. Per verificarne il corretto funzionamento si è utilizzato un sistema Hardware In the Loop (HIL) che permette la generazione dei segnali tipici utilizzati per l'analisi combustione presenti al banco motore, quali segnale di ruota fonica e segnali di pressione cilindro. In questo modo si è proceduto al debug del software dopo ogni aggiornamento hardware, per arrivare a versioni stabili, con le stesse caratteristiche di quelle precedenti, ma su hardware aggiornato.

Il secondo step di ogni fase dell'evoluzione software, è stato quello di sfruttare le potenzialità messe a disposizione dall'evoluzione delle piattaforme. La più importante è stata l'introduzione di nuovi input analogici, passando da 8 a 24 AI. La prima complicazione è stata quella della gestione del traffico dei dati verso il PC host, per la visualizzazione e il salvataggio degli stessi. Infatti, aumentando il numero di canali aumenta il traffico di dati:

$$24\ canali \times 200000\ \frac{campioni}{s} \times 16 \frac{bit}{canale} = 24 \times 200000 \times {}^{16}/_{8} \frac{byte}{s} = 9.6 \frac{Mb}{s}$$

Contro i $3.2 \frac{Mb}{s}$ del vecchio sistema. Questo traffico è gestito in maniera ottimizzata e compatta, prima tramite una FIFO DMA da FPGA a real time, poi viene spedito al PC host tramite una connessione TCP/IP: è stato quindi necessario agire sull'ottimizzazione del codice di gestione dei dati per velocizzarne la ricezione e permettere al sistema di gestire il traffico triplicato.

L'aumento degli input, in parallelo con l'aumento della potenza di calcolo, ha permesso un'evoluzione anche a livello di numero di cilindri massimo su cui compiere l'analisi della combustione in tempo reale: si è passati da un massimo di 6 segnali di pressione analizzati ad un massimo di 12 cilindri analizzati in tempo reale. Questo è stato fatto ottimizzando gli algoritmi di calcolo per il recupero della componente media e per il calcolo di rilascio calore, rendendoli più efficienti in termini di tempo di esecuzione senza

comprometterne l'accuratezza (completando i calcoli in tempo utile, cioè entro il singolo intervallo angolare, per tutti e 12 i cilindri). Ciò è stato possibile grazie alla disponibilità di maggior memoria e all'incremento dei moltiplicatori a disposizione della nuova FPGA.
Un problema legato all'incremento degli AI, è stato quello del filtraggio digitale dei dati a livello FPGA: anche in questo caso si è lavorato sul codice già esistente, migliorando l'algoritmo di filtraggio digitale per far fronte ad un maggior numero di canali (i filtri già presenti erano dei Butterworth del 2° ordine passa basso, per filtrare i dati per l'analisi indicating, e passa alto, per analisi di detonazione). Per un calcolo del 2° ordine in FPGA sono necessari tre step[1] per canale, di conseguenza il tempo di calcolo cresce notevolmente col numero di canali. È stato quindi necessario spezzare la parte di filtraggio in due parti parallele per riuscire a completare l'operazione nel tempo necessario a consentire il campionamento alla frequenza corretta.
Le potenzialità del nuovo hardware hanno permesso di alzare la frequenza di campionamento dei dati da 200 a 400kHz. Questo permette di poter acquisire e quindi apprezzare fenomeni ad alta frequenza (fino alla frequenza di Nyquist, 200kHz in questo caso). Per quanto riguarda l'analisi indicating questo incremento non è rilevante (tale analisi viene fatta dopo un filtraggio passa basso a frequenza nell'intorno dei 3-5kHz), mentre a livello di analisi di detonazione permette di apprezzare fenomeni a frequenza più elevata. A livello pratico l'incremento della frequenza di campionamento ha posto il problema del dimezzamento delle tempistiche di calcolo tra due successivi campioni, in particolare per quello che riguarda il filtraggio e l'invio dei dati verso host. Inoltre a frequenza doppia, raddoppia anche il traffico dati da visualizzare e da salvare: al momento non è stato possibile gestire più di 12 segnali campionati a 400kHz. Di conseguenza il sistema si auto limiterà a 12 canali nel momento in cui l'utente chiede di visualizzare e salvare dati a 400kHz. A livello FPGA, per evitare aliasing, il campionamento è comunque mantenuto sempre a 400kHz, in conseguenza al fatto che l'hardware è dotato di un filtro analogico anti-aliasing con frequenza di taglio a 100kHz (-3db, -20db a 200kHz). Il segnale viene poi sottocampionato tramite media mobile a due campioni per portarlo a 200 kHz nel caso in cui non sia richiesta una frequenza maggiore dell'utente.
L'incremento del numero di canali e la possibilità di modificare la frequenza di streaming dei dati, ha introdotto la necessità di dover configurare e visualizzare i segnali acquisiti nelle nuove modalità: in particolare, l'utente deve poter scegliere che tipo di segnale sta

[1] Con step si intende l'esecuzione di un loop temporizzato alla frequenza base del clock FPGA (40 MHz). Ogni step avrà quindi durata 25ns.

acquisendo su un determinato input, decidere la frequenza di acquisizione, assegnargli un gain ed un offset per visualizzarlo nell'unità fisica corretta e salvarlo su file. Per far questo è stato necessario potenziare l'interfaccia di visualizzazione e l'interfaccia di configurazione (di cui si parlerà in seguito), per renderle idonee alla gestione di 24 canali invece di 8.

1.4 Sviluppo dell'interfaccia grafica

Per quanto riguarda lo sviluppo dell'interfaccia principale e di configurazione, ci si è basati su richieste funzionali provenienti da utilizzatori del software e su considerazioni di intuitività delle funzionalità che hanno portato ad un suo netto cambiamento come mostrato nelle figure seguenti. In particolare, l'obbiettivo è stato quello di rendere semplice ed intuitiva ogni funzionalità o modifica di parametri di configurazione, in maniera che l'utente possa interpretare i dati misurati in maniera semplice e allo stesso tempo perdere il minor tempo possibile per configurare la prova. Infatti, mettere a disposizione algoritmi di analisi avanzata senza fornire una facile interpretazione e configurazione dei parametri porta sicuramente all'inutilità del calcolo stesso; se l'utente non è in grado di interpretare in maniera semplice i risultati o, peggio ancora, se a causa di configurazioni erronee o assenti si ottengono dati errati senza esserne consapevoli, il sistema risulterà sicuramente poco utile.

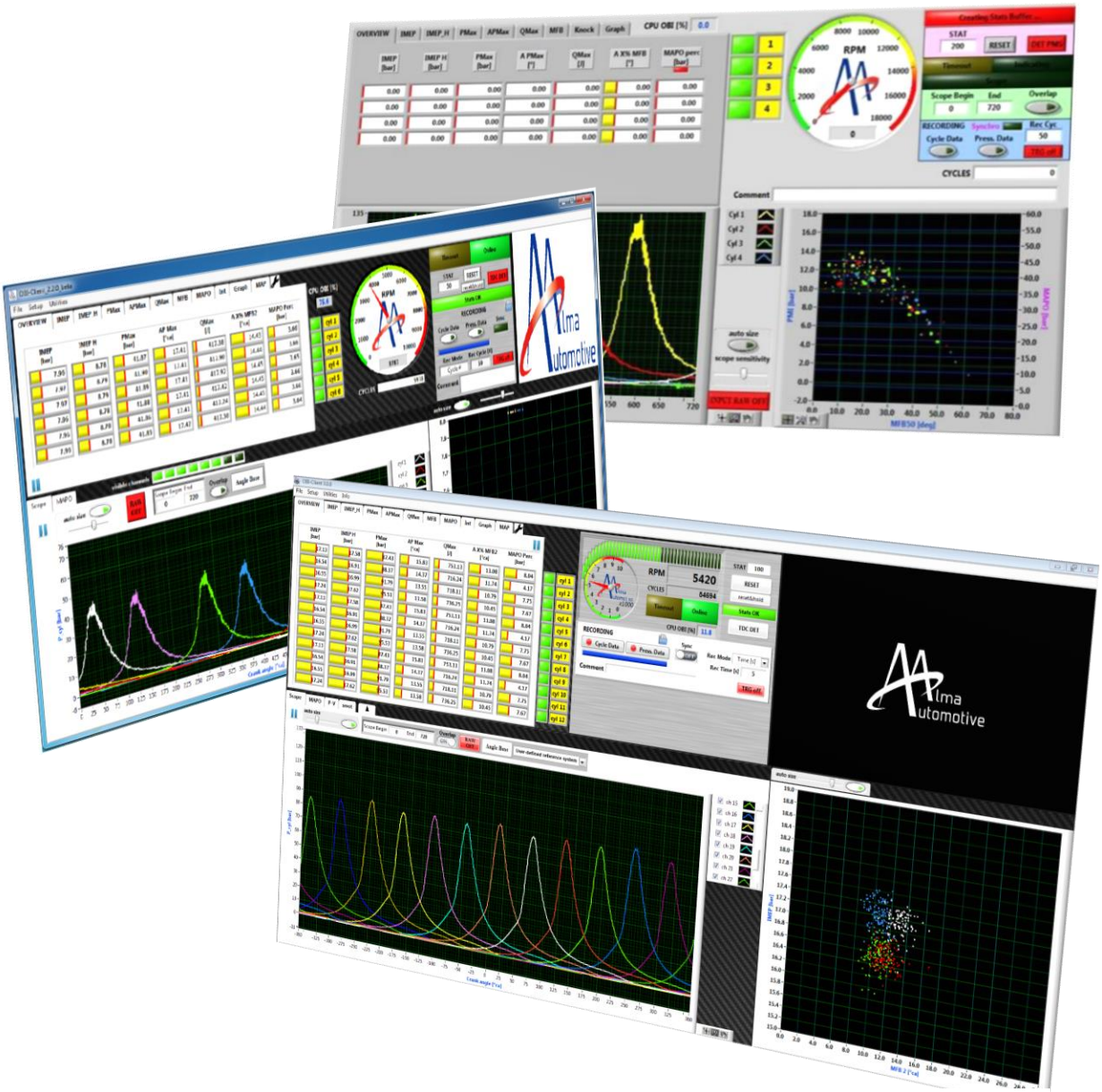


Figura 1-14: evoluzione dell'interfaccia grafica

Per una migliore leggibilità dei dati e per dare all'utente la possibilità di capire cosa sta succedendo con un "colpo d'occhio", è stata implementata la possibilità di aprire grafici in nuove finestre, in maniera da mettere a disposizione dell'utente varie informazioni facilmente interpretabili:

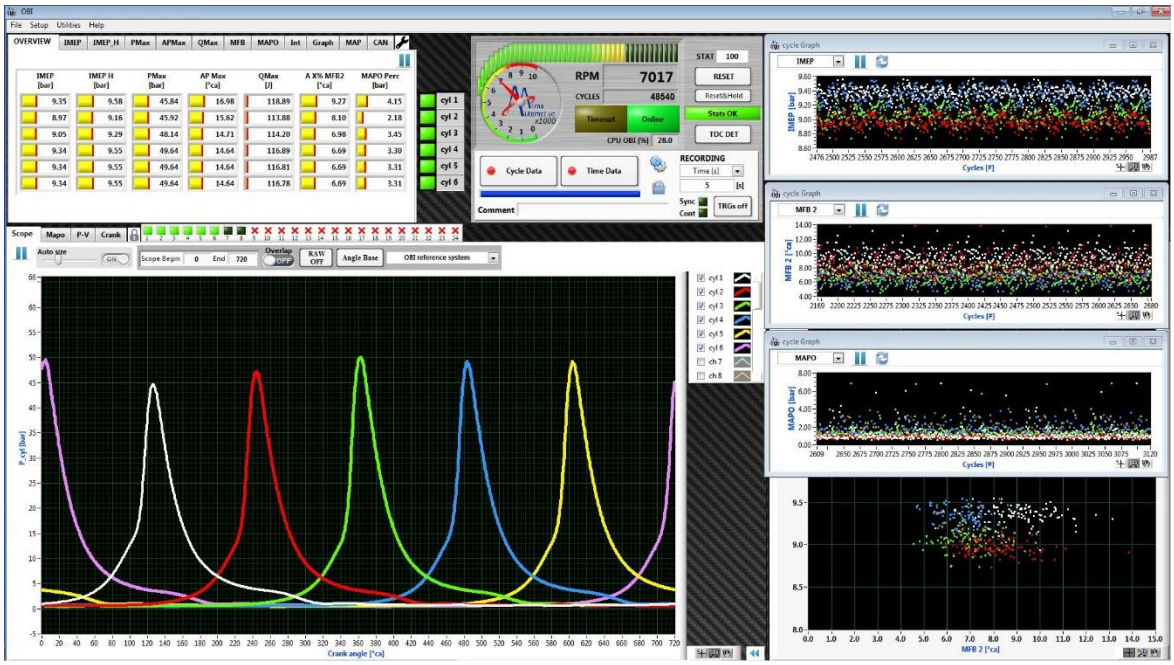


Figura 1-15: interfaccia OBI

Anche per quanto riguarda la configurazione l'interfaccia è stata migliorata di pari passo con l'aumento della flessibilità del software, permettendo all'utente di avere a disposizione sempre più variabili su cui poter intervenire e allo stesso tempo farlo in maniera più intuitiva.

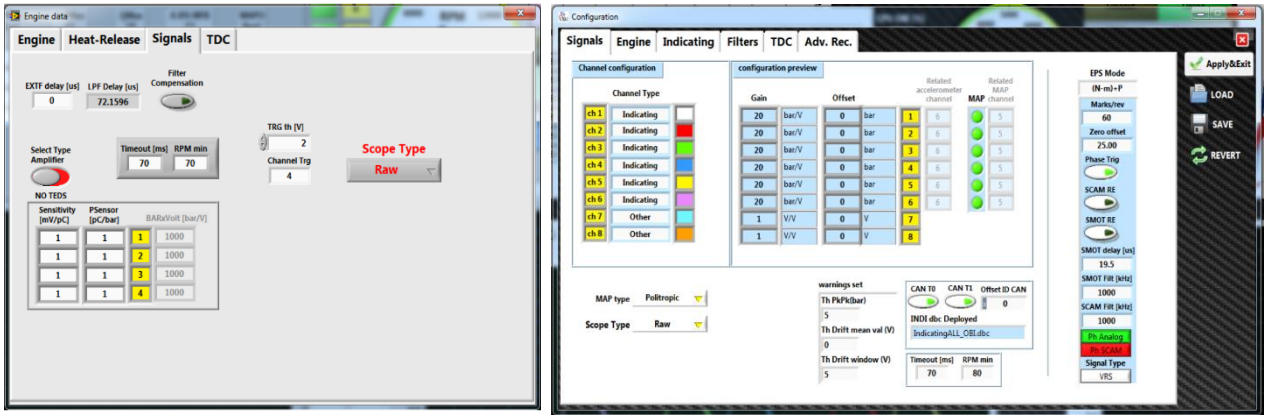


Figura 1-16: evoluzione interfaccia di configurazione segnali

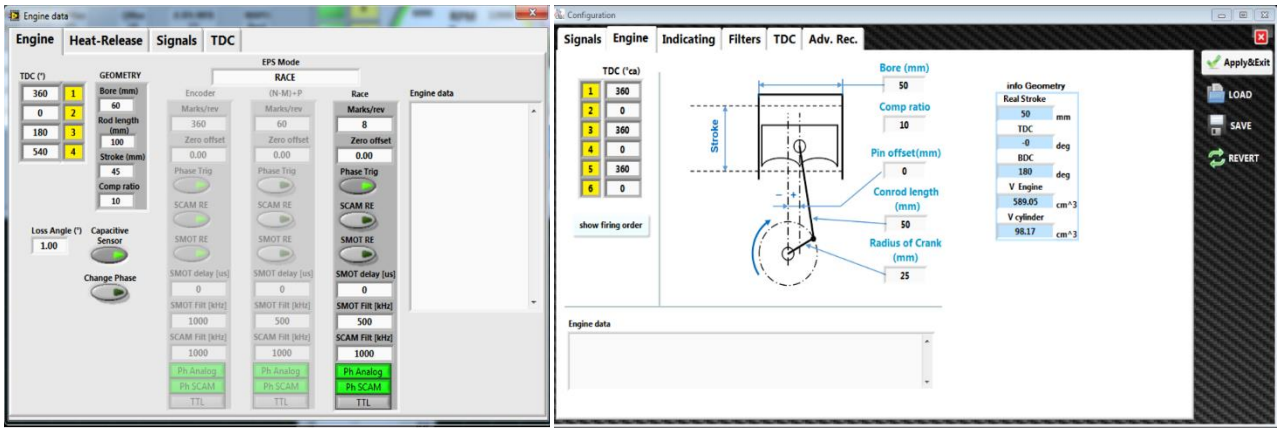


Figura 1-17: evoluzione interfaccia di configurazione parametri motore

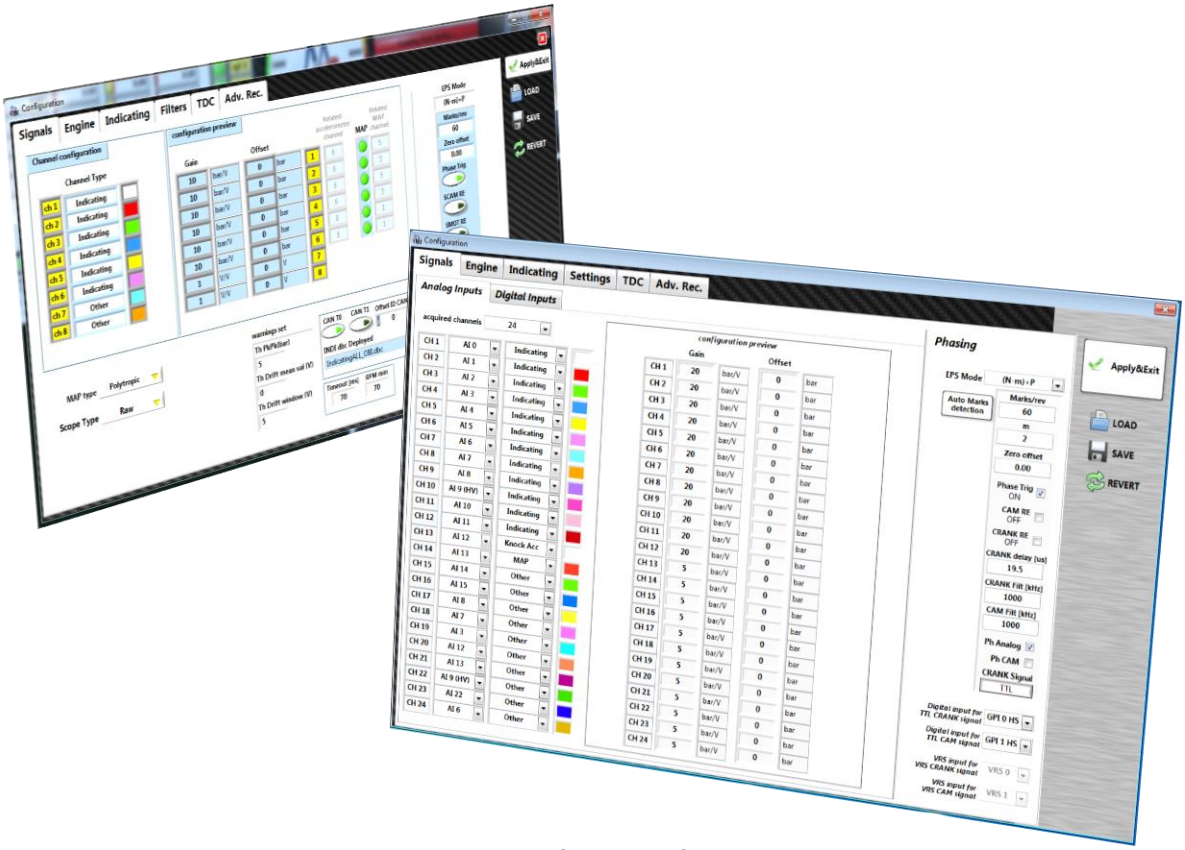


Figura 1-18: evoluzione interfaccia di configurazione

2. SINCRONIZZAZIONE E FASATURA

Il sistema di analisi necessita di essere sincronizzato e fasato con il motore da analizzare per associare ai campioni acquisiti su base tempo la rispettiva posizione angolare sul ciclo motore (compresa tra 0 e 720°ca). Per farlo si utilizza il segnale di posizione proveniente dalla ruota fonica o dall'encoder montato sull'albero motore, che permette l'identificazione di una posizione angolare del motore nel tempo. Il sistema di analisi combustione, per essere compatibile con le più svariate tipologie di motori, deve quindi essere in grado di adattarsi a ruote foniche differenti (con cava o senza cava) e ad encoder ottici, e deve essere in grado di adattarsi al numero di riferimenti per giro della ruota/encoder che si sta utilizzando (e.g. 60-2, piuttosto che 24-2, piuttosto che encoder a 360 tacche).

2.1 Acquisizione a passo angolare costante o a frequenza costante

Come noto, per poter procedere all'analisi dei dati indicati, è necessario conoscere la posizione angolare dei campioni di pressione acquisiti. Questo può essere fatto a costo zero "triggerando" l'acquisizione dei campioni nel momento dell'acquisizione di ogni marker angolare generato dal sensore di posizione angolare. Ad esempio, nel caso di utilizzo di un encoder a 720 tacche, si può lanciare l'acquisizione dei campioni ad ogni trigger ricevuto, ottenendo così una curva di pressione nel ciclo motore alla risoluzione di 0.5° di manovella. Questo tipo di acquisizione è detta a passo angolare costante, e restituisce una curva di pressione a risoluzione angolare determinata dal numero di marker generati. Spesso è necessaria una risoluzione ben superiore: risulta quindi indispensabile generare un numero maggiore di trigger di registrazione, andando a moltiplicare il numero di marker fisicamente presenti sull'encoder in maniera virtuale, specialmente nel caso in cui non si stia usando un encoder ma una ruota fonica con pochi denti per giro. La generazione delle tacche virtuali obbliga ad estrapolare la velocita dell'ultimo dente letto per poter generare i marker necessari per l'acquisizione fino al dente successivo, ipotizzando che la velocità non cambi in quell'arco angolare (o ipotizzando l'andamento fino al riferimento angolare successivo). Questo porta a

commettere un errore sull'attribuzione della posizione angolare del campione tanto maggiore quanto maggiore è la variazione di velocità tra due denti successivi. Per ovviare a questo problema si procede ad una acquisizione a frequenza costante dei campioni di pressione e parallelamente all'acquisizione dei marker angolari, procedendo all'attribuzione dell'angolo ai campioni tra due marker soltanto una volta noto il secondo marker, e quindi l'effettiva velocità (ipotizzata lineare) tra due marker successivi. Questo permette di sostituire l'estrapolazione (si prevede quello che potrebbe accadere, con alta probabilità di errore) con una interpolazione (si approssima linearmente un dato acquisito).

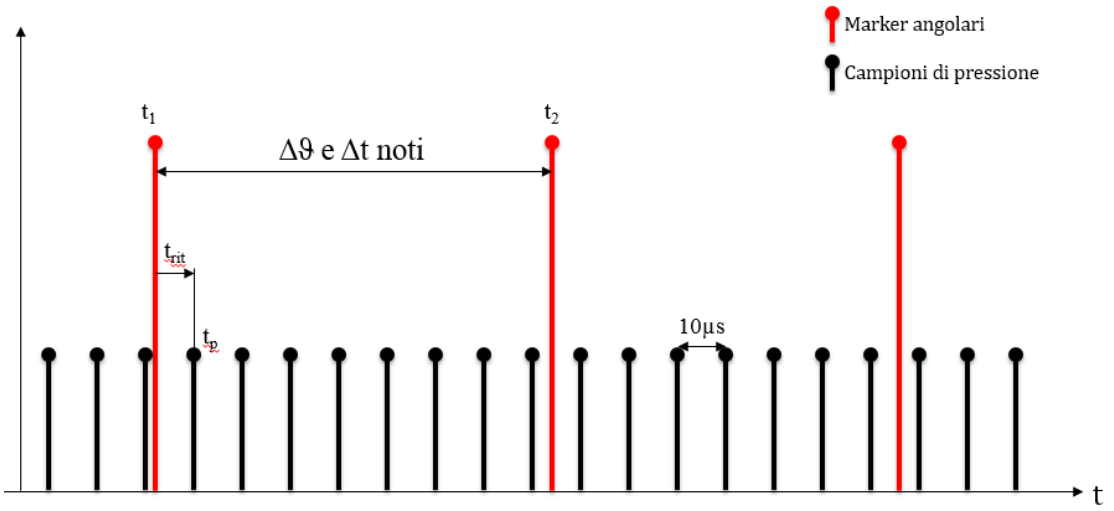


Figura 2-1: schematizzazione campionamento su base tempo

Nota la posizione angolare dei marker della ruota fonica e il loro tempo, e noto il tempo in cui si è acquisito il singolo campione di pressione tra due marker, si può determinare la posizione angolare ϑ_p del campione di pressione:

$$\frac{t_p - t_1}{t_2 - t_1} = \frac{\vartheta_p - \vartheta_1}{\vartheta_2 - \vartheta_1} \rightarrow \vartheta_p = \frac{t_{rit}\,\Delta\vartheta}{\Delta t} + \vartheta_1$$

Nel tempo di ritardo vanno tenuti in considerazione i ritardi della catena di misura (velocità angolare e pressione nel cilindro):

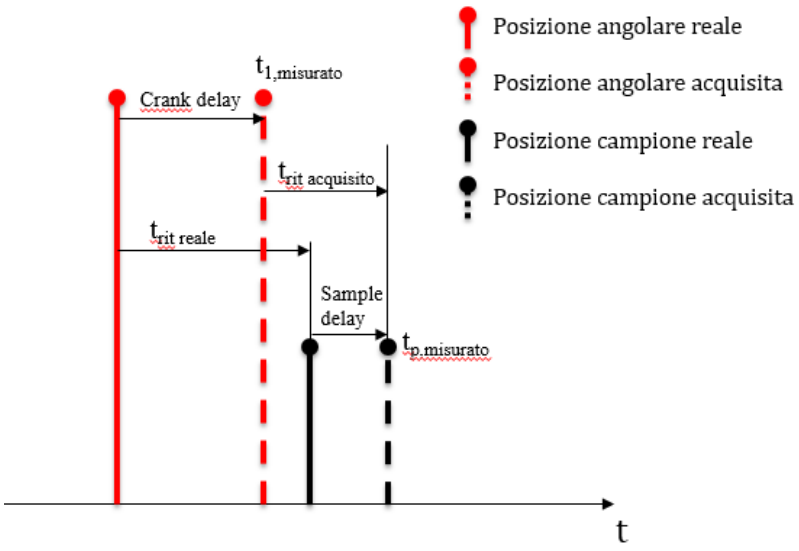


Figura 2-2: schematizzazione della compensazione dei ritardi

E quindi si ha che:

$$t_{rit,\,reale} = t_{rit,\,acquisito} + Crank\ delay - Sample\ delay$$

$$\vartheta_p = \frac{t_{rit,reale}\,\Delta\vartheta}{\Delta t} + \vartheta_1$$

In questo modo è possibile assegnare ad ogni campione acquisito a tempo costante il rispettivo angolo di manovella. La risoluzione angolare, essendo l'acquisizione a tempo costante, sarà dipendente dal regime motore, diminuendo all'aumentare della velocità di rotazione.

Per poter procedere a questo tipo di ricostruzione, è necessario che il sistema abbia a disposizione il corretto vettore angolo associato ai riferimenti della ruota fonica in uso, oltre a conoscere i ritardi introdotti dal filtraggio hardware anti-aliasing e da eventuali filtri digitali impostati dall'utente. Per associare il corretto valore angolare ad ogni marker, bisogna conoscere quante tacche per giro vengono acquisite, e tenerne traccia tramite un contatore. Inoltre, è necessario stabilire quale sia lo zero del sistema di riferimento: questo viene fatto in maniera differente in funzione della tipologia di ruota fonica utilizzata. In particolare, nel caso di presenza di cava, sarà questa a definire il dente zero. Per poter riconoscere la cava, un algoritmo dedicato monitora i tempi-dente: andando a riconoscere quando il tempo dente al passo i-esimo è superiore ad un multiplo del tempo dente precedente ($\Delta T_i > N \times \Delta T_{i-1}$) si può determinare la posizione della cava, e quindi azzerare il contatore. Il moltiplicatore N è tipicamente un 2, in quanto in condizioni di funzionamento normale non si hanno variazioni di velocità con fattore superiore a due da dente a dente (come vedremo in seguito, questo non sempre è corretto).

Nel caso di encoder (o ruota fonica senza cava) sarà invece necessario acquisire un secondo segnale (trigger di giro) per poter stabilire quale sia lo zero del sistema di riferimento.

2.2 Fasatura

Oltre alla sincronizzazione del sistema dente dopo dente, è anche necessario riconoscere la fase corretta del ciclo motore per poter assegnare correttamente la fase attiva del ciclo al rispettivo TDC attivo. Per farlo sono possibili due strade: il primo modo è utilizzare un secondo segnale di fase (fasatura con camma), con stato differente per cava attiva e cava passiva, oppure vista la presenza del segnale di pressione in camera, implementare un algoritmo che, in relazione al picco di pressione misurato e al TDC definito in configurazione, correli i due valori e stabilisca se effettivamente il picco cade nella parte di ciclo attiva, come atteso (fasatura analogica).

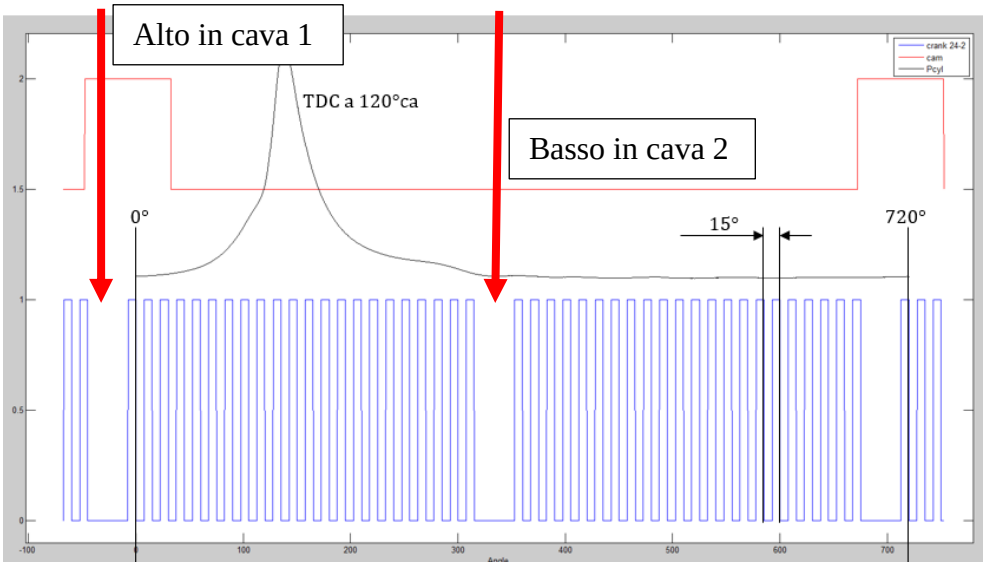


Figura 2-3: esempio di fasatura tramite segnale di camma

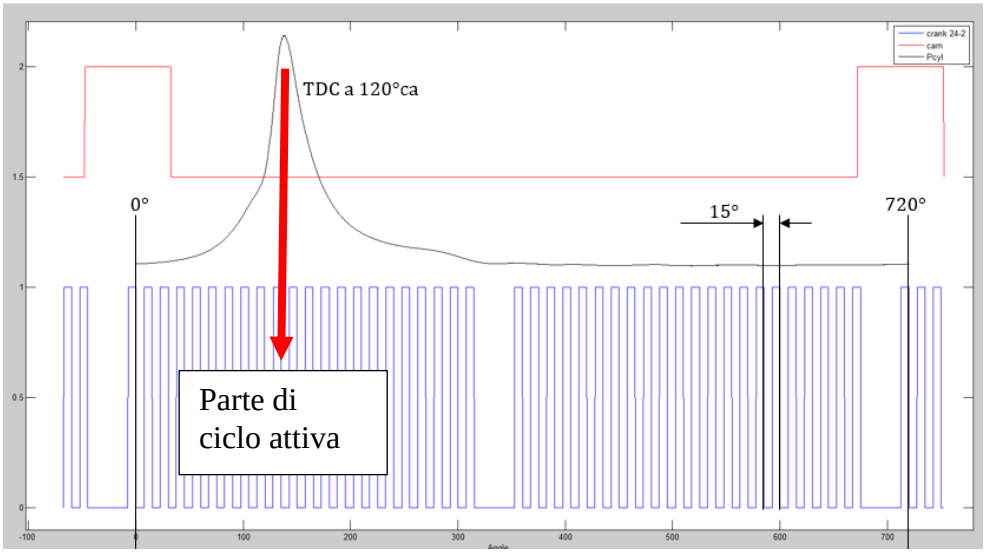


Figura 2-4: fasatura analogica, tramite picco di pressione

Questo algoritmo analogico è stato evoluto in termini di reattività: partendo da un'analisi della fase mediante picco di pressione controllata a intervalli di 1s (che equivale a molti cicli motore anche a basso regime durante la fase di avviamento) si è passati ad un controllo ciclico degli angoli misurati, permettendo così al sistema di monitorare ed eventualmente ripristinare la corretta fase in pochi cicli motore dopo l'accensione. Inoltre è stato introdotto un controllo sulla plausibilità del valore del picco di pressione misurato per evitare che problemi sul segnale (drift sensore, rottura cablaggio di un sensore, rumore) portino a determinare una fase sbagliata.

2.3 Sincronizzazione con ruota fonica di tipo n-m-m

La presenza sul mercato di una ruota fonica di tipo N-m-m (ruota fonica con due cave simmetriche con un solo dente mancante per cava) ha spinto ad evolvere il sistema di fasatura già presente per poterlo adattare a questa modalità. In particolare, per ruota con cava, come già visto, il sistema conta i riferimenti angolari e riconosce la cava azzerando il contatore al momento corretto. L'implementazione alla nuova modalità N-m-m dovrà riconoscere 4 cave per ciclo e contare $N/2 - m$ riferimenti tra una cava e la successiva, azzerando il contatore al momento corretto. È stato quindi necessario procedere alla modifica dei counter di fasatura in FPGA introducendo una nuova modalità che, a seconda dell'impostazione selezionata dall'utente, proceda all'opportuno conteggio dei

denti. Oltre a questo, anche l'algoritmo di fasatura è stato modificato per definire correttamente quale delle 4 parti in cui la doppia cava spezza il ciclo è quella effettivamente corretta per l'azzeramento del counter.
A questa fase è seguita una parte di validazione in HIL (Hardware In the Loop), generando un segnale TTL ideale che simula l'andamento di una 60-1-1, come riportato in figura:

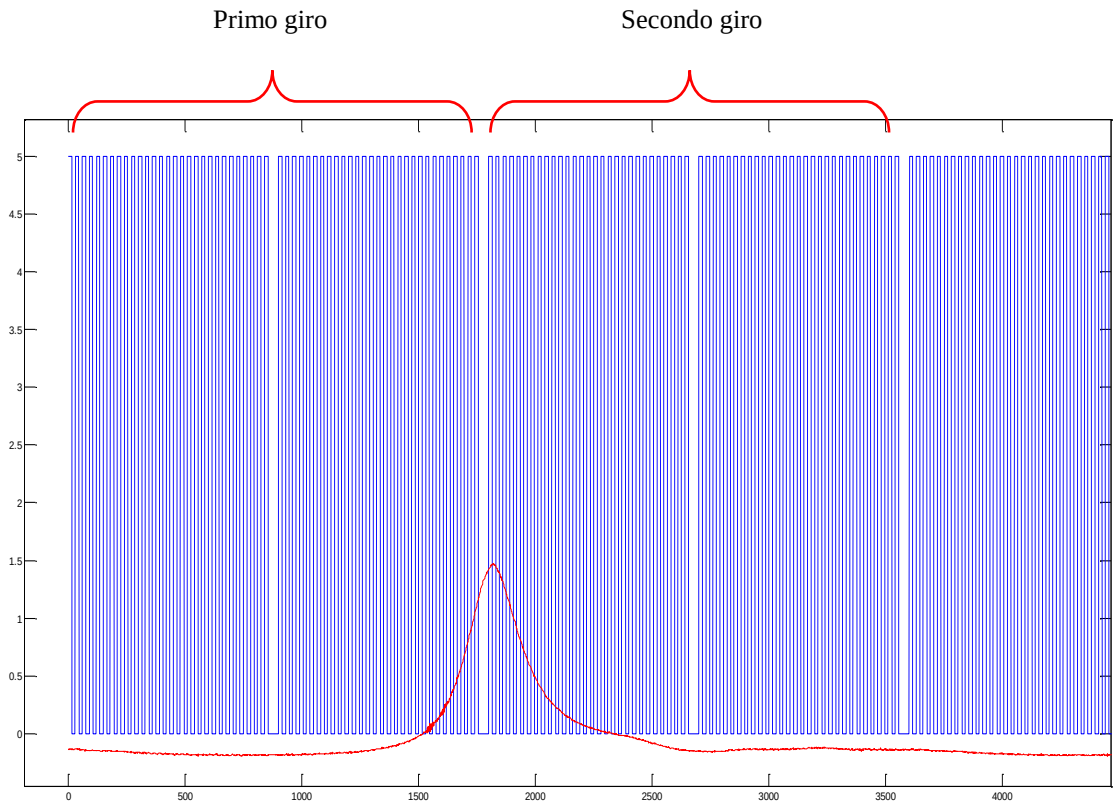


Figura 2-5: esempio di quadro segnale ideale con ruota fonica con cava simmetrica (60-1-1)

La validazione è stata fatta confrontando l'acquisizione real time del caso con doppia cava simmetrica e una successiva acquisizione dello stesso file ricostruendo il dente mancante della seconda cava e togliendo un dente nella prima (ripristinando cioè la cava classica 60-2). In questo modo si è verificato che i valori calcolati dal sistema sulla stessa campana di pressione fossero esattamente gli stessi, indipendentemente dal tipo di riferimento angolare adottato.
Infine si è proceduto ad una verifica al banco (motore FIAT multi jet 1.3, banco prova universitario, sede di Forlì). Non avendo a disposizione una ruota fonica con doppia cava simmetrica, si è simulato lo stesso tipo di sistema di riferimento angolare utilizzando un encoder da 180 tacche per giro, oscurandone due in maniera simmetrica, così da generare un quadro-segnali equivalente ad una ruota fonica con due cave contrapposte. Si è quindi

verificato che i risultati ottenuti fossero comparabili con quelli che si ottenevano in precedenza per prove analoghe. Ovviamente la dispersione ciclica non permette di fare prove di comparazione rigorose sui dati rilevati al banco come invece si può fare con la simulazione in hardware in the loop, ma d'altra parte l'oscillazione del regime motore che si ha al banco su un motore reale, non generabile in maniera estremamente fedele in HIL, permette di verificare che effettivamente il sistema riconosca in maniera ottimale le due cave e rimanga sincronizzato durante le varie fasi di test senza commettere errori.

2.4 Fasatura con cave di dimensione variabile

Tipicamente la cava di una ruota fonica è generata dalla mancanza di due denti. Questo non sempre è vero (come abbiamo visto per la ruota a doppia cava simmetrica). Per far fronte ai casi in cui siano utilizzate ruote foniche particolari, con cava di dimensione non standard (da 1 a 3 denti) è stato sviluppato un nuovo algoritmo di riconoscimento cava, flessibile in relazione al numero di denti mancanti impostato dall'utente. Questo comporta la definizione di una soglia di differenza tempo rispetto al dente precedente variabile a seconda della tipologia di cava utilizzata, per evitare di fare falsi o mancati riconoscimenti.

$$\Delta T_i > N \times \Delta T_{i-1}, con\ N\ variabile\ e\ configurabile$$

Oltre al riconoscimento della cava, è stato poi necessario rendere flessibile anche l'associazione campione-angolo in cava (vettore angolo associato ai marker), andando a battezzare intervalli angolari flessibili non solo in funzione del numero di denti per giro, ma anche del numero di denti mancanti per giro. In questo modo è stato reso possibile all'utente scegliere non solo la tipologia di ruota fonica e il numero di denti, ma anche il numero di denti mancanti in cava. Anche in questo caso il test di validazione ha seguito una prima fase in HIL ed una seconda fase al banco brova (sede universitaria di Forlì): le prove al banco sono state fatte creando cave con numero di denti voluto mediante l'oscurazione di tacche di un encoder, in maniera da poter cambiare il numero di riferimenti angolari senza dover effettivamente cambiare il sensore e il montaggio.

2.5 Riconoscimento automatico del tipo di ruota fonica

Non sempre si è a conoscenza del tipo di ruota fonica installata su un particolare motore, in particolare nel caso in cui si derivi il segnale dallo stesso sensore utilizzato dalla centralina, che è solitamente montato su una ruota calettata all'albero motore, coperta da un carter. In questo caso sarebbe necessaria un'acquisizione del segnale di ruota fonica ed una successiva analisi per determinarne il numero di denti. È stato quindi sviluppato un algoritmo che permette automaticamente la determinazione del numero di denti e del tipo di ruota fonica che si sta utilizzando. L'algoritmo impone il trascinamento (o lo stazionario) del motore ad un regime imposto noto, che l'utente deve inserire all'interno del software.

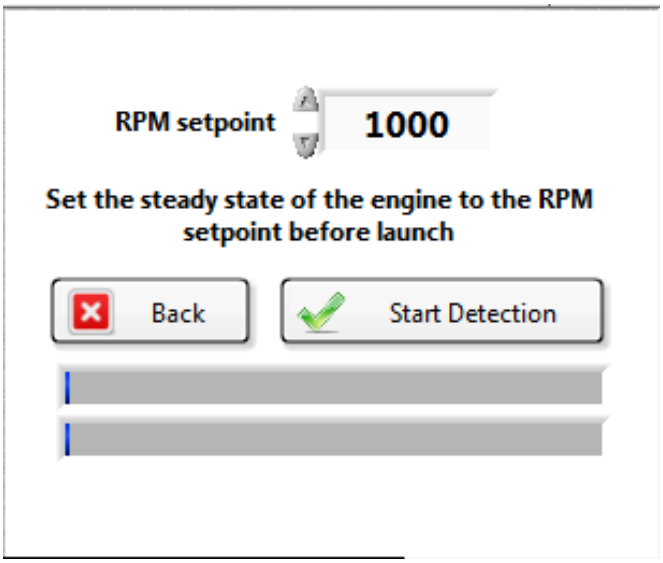


Figura 2-6: interfaccia di lancio dell'algoritmo di determinazione fonica

Una volta inserito il regime e lanciata la determinazione, l'algoritmo va ad analizzare i tempi-dente, trovandone la mediana, e ricava il numero di denti rapportando la velocità imposta con il singolo tempo dente (la mediana è necessaria per eliminare eventuali cave). A questo punto, rianalizzando i tempi-dente, si riconosce il massimo, si impone una soglia tra massimo e mediana, e si riconosce il numero di cave al giro e la loro estensione, esaminando i tempi dente maggiori della soglia imposta. Si riesce così a risalire alla tipologia di ruota fonica e alla dimensione della cava.

2.6 Problemi di sincronizzazione e fasatura

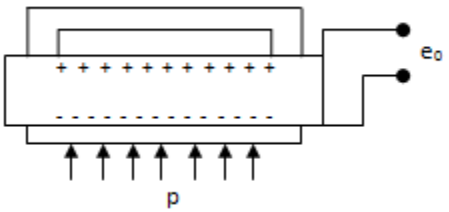
Molti dei problemi di funzionamento dei sistemi di analisi combustione in applicazioni in vettura ed al banco prova sono legati all'acquisizione del segnale di posizione. Una delle problematiche tipiche è quella legata a rumore elettrico sul segnale motore che si riscontra al banco prova nel momento in cui il freno deve esercitare coppie elevate (e genera quindi maggiore rumore elettrico) o nel momento in cui vengono azionati dei ventilatori per il raffreddamento dei radiatori posizionati in cella. In questi casi i disturbi sul segnale motore possono essere tali da generare dei picchi in tensione tali da far misurare al sistema una transizione logica interpretata come un marker angolare. Tipicamente, per risolvere questo problema si procede ad un filtraggio digitale sul segnale motore per evitare che uno spike venga riconosciuto come marker: in questo caso è necessario compensare correttamente il ritardo introdotto dal filtro digitale per evitare di commettere errori nell'assegnazione della posizione angolare dei campioni di pressione. Quando non è possibile risolvere il problema filtrando, è opportuno ricontrollare il cablaggio, cercando di schermarlo il più possibile da rumori elettrici.

Un altro problema riscontrato al banco prova è il falso riconoscimento della cava in condizioni di avviamento: questo fenomeno si ha quando il motore in analisi ha forti oscillazioni di velocità nella fase di trascinamento iniziale e nelle prime fasi di accensione dovute sia a cause legate al motore (basso numero di cilindri, RC elevato, fase di avviamento non calibrata) sia a cause esterne (motorino di avviamento non sufficientemente potente per il trascinamento omogeneo). In questo caso quello che si verifica è un'oscillazione istantanea della velocità molto elevata tra un dente e il successivo, superiore al fattore 2 canonico per cave di dimensione standard, e quindi si determina un falso riconoscimento della cava dove invece si ha una normale transizione di un dente. Il semplice aumento della soglia non sarebbe sufficiente, in quanto porterebbe a mancati riconoscimenti in condizione di funzionamento normale. È quindi necessario rendere dinamico il moltiplicatore per il riconoscimento della soglia, in relazione al punto di funzionamento del motore (questo algoritmo non è ancora stato introdotto all'interno del sistema di sincronizzazione nel codice, ma solo valutato offline).

3. IMPLEMENTAZIONE PRESSURE PEGGING IN TEMPO REALE

3.1 Perché è necessario il pegging

Come noto [10], l'utilizzo di sensori di pressione piezoelettrici per l'analisi della pressione in camera di combustione impone l'utilizzo di un amplificatore di carica per poter convertire la carica generata dal quarzo piezoelettrico in un segnale in tensione misurabile. In particolare, il quarzo, sottoposto ad una variazione di pressione, assumerà il comportamento di un condensatore, con carica Q proporzionale alla deformazione subita, e quindi alla pressione presente nell'ambiente in cui è affacciata la membrana. Sarà quindi possibile raccogliere tale carica sotto forma di tensione nota la capacità C del condensatore.



$$Q = -k_x\, x = -k_p\, p$$

$$e_0 = \frac{Q}{C} = \frac{-k_p\, p}{C}$$

Figura 3-1: schema sensore piezoelettrico

In particolare, a variazioni di pressione in camera corrisponderanno variazioni di carica, e quindi una corrente: $-k_p \frac{dp}{dt} = \frac{dQ}{dt} = i_{(t)}$

La tensione in uscita è molto bassa: è dunque necessario inserire nella catena di misura un amplificatore di carica. La catena di misura può quindi essere schematizzata in tre parti distinte, la prima riguardante il trasduttore, rappresentato come un generatore di corrente con una certa resistenza e capacità in parallelo, la seconda come una resistenza e capacità in parallelo, rappresentanti il cavo coassiale che collega trasduttore e amplificatore, ed una terza parte rappresentante un amplificatore operazionale:

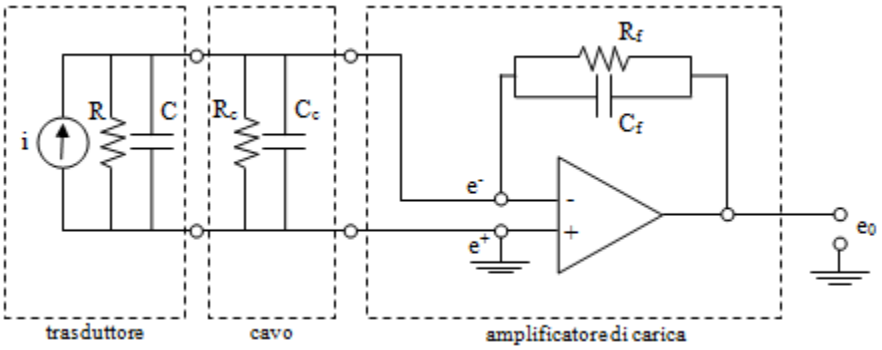
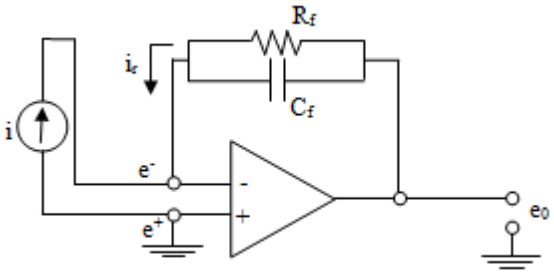


Figura 3-2: schematizzazione della catena di misura della pressione

L'amplificatore operazionale ha una costante di guadagno A idealmente infinita tale che: $e_0 = A(e^+ - e^-)$; quindi si ha $(e^+ - e^-) = \frac{e_0}{A} \cong 0$, cioè $e^+ = e^- = 0$, dato che e^+ è collegato a terra. Allora non si avrà circolazione di corrente nelle resistenze e capacità prima dell'amplificatore, e quindi il circuito può esser semplificato come segue:



$$i_a + i_f = i_a + i_R + i_C = 0$$

$$-K_p \frac{dp}{dt} - \frac{e_0}{R} - C\frac{de_0}{dt} = 0$$

$$\frac{dp}{dt} = \frac{e_0}{K_p\, R} + \frac{C}{K_p}\frac{de_0}{dt}$$

Figura 3-3: circuito di condizionamento del segnale semplificato

Avendo applicato la legge di Kirchoff al morsetto invertente e poi integrando si ottiene:

$$p(t) = \frac{1}{K_p\, R}\int_0^t e_0\, dt + \frac{C}{K_p} e_0 + \boldsymbol{p(0)}$$

Si nota quindi come l'amplificatore di carica non si limiti ad amplificare semplicemente la tensione, ma provoca un filtraggio passa alto, andando a tagliare le armoniche a bassa frequenza e quindi a causare anche la perdita della componente media, che deve essere recuperata per poter risalire ai valori effettivi dei picchi di pressione e per effettuare correttamente i calcoli riguardanti il rilascio di calore e le grandezze da esso dedotte.

L'algoritmo già implementato nelle precedenti versioni hardware rendeva possibile una sola modalità di recupero della componente media (pegging), basato sul confronto del segnale con un valore fisso imposto dall'utente in un particolare intervallo angolare (pegging a punto fisso). L'utente può selezionare una finestra angolare (tipicamente centrata sul punto morto inferiore in fase di aspirazione) ed un valore di pressione atteso:

il software calcola il valore medio del segnale nella finestra selezionata e definisce un Δp da applicare al segnale, calcolato come la differenza tra la media e il valore fisso impostato da utente.

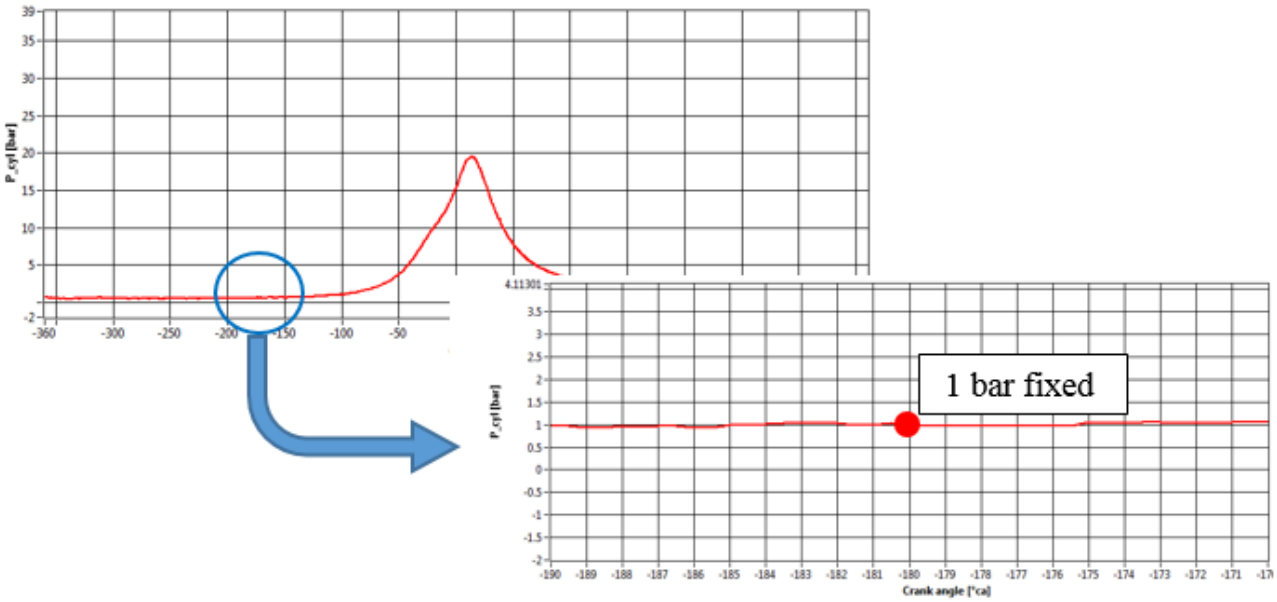


Figura 3-4:esempio di pegging a punto fisso

Questa modalità ovviamente porta a commettere un errore, specialmente in caso di funzionamento a carico parzializzato o in presenza di sovralimentazione, dove la pressione di aspirazione è fortemente differente da quella ambiente e variabile in maniera dinamica a seconda del punto di funzionamento (carico-regime). Per questo sono state introdotte delle nuove modalità di recupero della componente media:

- Recupero mediante sensore di pressione collettore assoluto (MAP)
- Metodo della politropica ad esponente forzato
- A punto fisso imposto dinamicamente da fonte esterna (via CAN)

3.2 Pegging con sensore MAP

Data la disponibilità di canali analogici ad alta frequenza, si è pensato di utilizzarne uno (o più di uno) per l'acquisizione di un sensore di pressione collettore assoluto (MAP: manifold air pressure) ed utilizzarlo come riferimento per recuperare la componente media del segnale di pressione. Infatti, nella finestra angolare di fine aspirazione, a valvole aperte, si può assumere che la pressione all'interno del cilindro sia coincidente

con quella nel condotto di aspirazione; quindi, confrontando i due segnali in tale finestra, è possibile risalire al Δp da applicare al segnale di pressione nel cilindro.

L'algoritmo implementato è piuttosto banale considerando che è sufficiente fare una media dei due segnali nella finestra angolare di interesse (selezionata dall'utente) e ricavarne la differenza. La parte complicata è l'implementazione in tempo reale del recupero, cioè riuscire a confrontare i due segnali acquisiti non solo entro il ciclo motore, ma prima dello svolgimento dei calcoli relativi al rilascio di calore, cioè prima dell'inizio della combustione. Infatti per il calcolo del calore rilasciato è necessario un valore del segnale di pressione già corretto tramite il recupero della componente media. Questo impone di sviluppare in FPGA una parte di codice che proceda al calcolo dell'offset da applicare alla curva di pressione, mediante il confronto tra pressione cilindro e pressione collettore misurate. Il codice FPGA non consente una facile conversione da bit acquisiti (unità elettrica) ad unità fisica (bar nel caso specifico): si è deciso quindi di mantenere i segnali in bit ed applicare gli opportuni guadagni ed offset per poterli confrontare (ricordiamo che ogni sensore ha la propria caratteristica $\frac{bar}{V}$, $\frac{V}{bit}$ e quindi $\frac{bar}{bit}$). Cioè:

$$MAP[bar] = MAP[bit] \times Gain_{MAP}\left[\frac{bar}{bit}\right] + Off_{MAP}[bar]$$

$$Pcyl[bar] = Pcyl[bit] \times Gain_{Pcyl}[\frac{bar}{bit}]$$

$$MAP_{comp}[bit] = MAP[bit] \times \underbrace{Gain_{MAP}\frac{bar}{bit} \times \frac{1}{Gain_{Pcyl}\left[\frac{bar}{bit}\right]}}_{GAIN} + \underbrace{Off_{MAP}[bar] \times \frac{1}{Gain_{Pcyl}\left[\frac{bar}{bit}\right]}}_{OFF}$$

Calcolando a livelli superiori, dove è possibile utilizzare la virgola mobile (Host o Real Time), i valori di *GAIN* ed *OFF* e spedendoli in FPGA, è possibile ricavare il valore in bit di pressione collettore confrontabile con quello della pressione cilindro e quindi ricavare il Δp da applicare al segnale:

$$MAP_{comp}[bit] = MAP[bit] \times GAIN + OFF$$

$$\Delta P = \overline{MAP}_{comp}[bit] - \overline{Pcyl}[bit]$$

Anche in questo caso si è proceduto alla validazione prima in HIL e poi al banco prova: nella figura seguente è mostrata un'acquisizione al banco in cui prima si acquisiscono un segnale di pressione cilindro ed un segnale di pressione collettore senza però attivare il recupero della componente media, per poi riattivare il recupero: notare come nella parte di aspirazione il segnale di pressione vada effettivamente a coincidere col segnale di pressione attivando il pegging, come imposto dall'algoritmo.

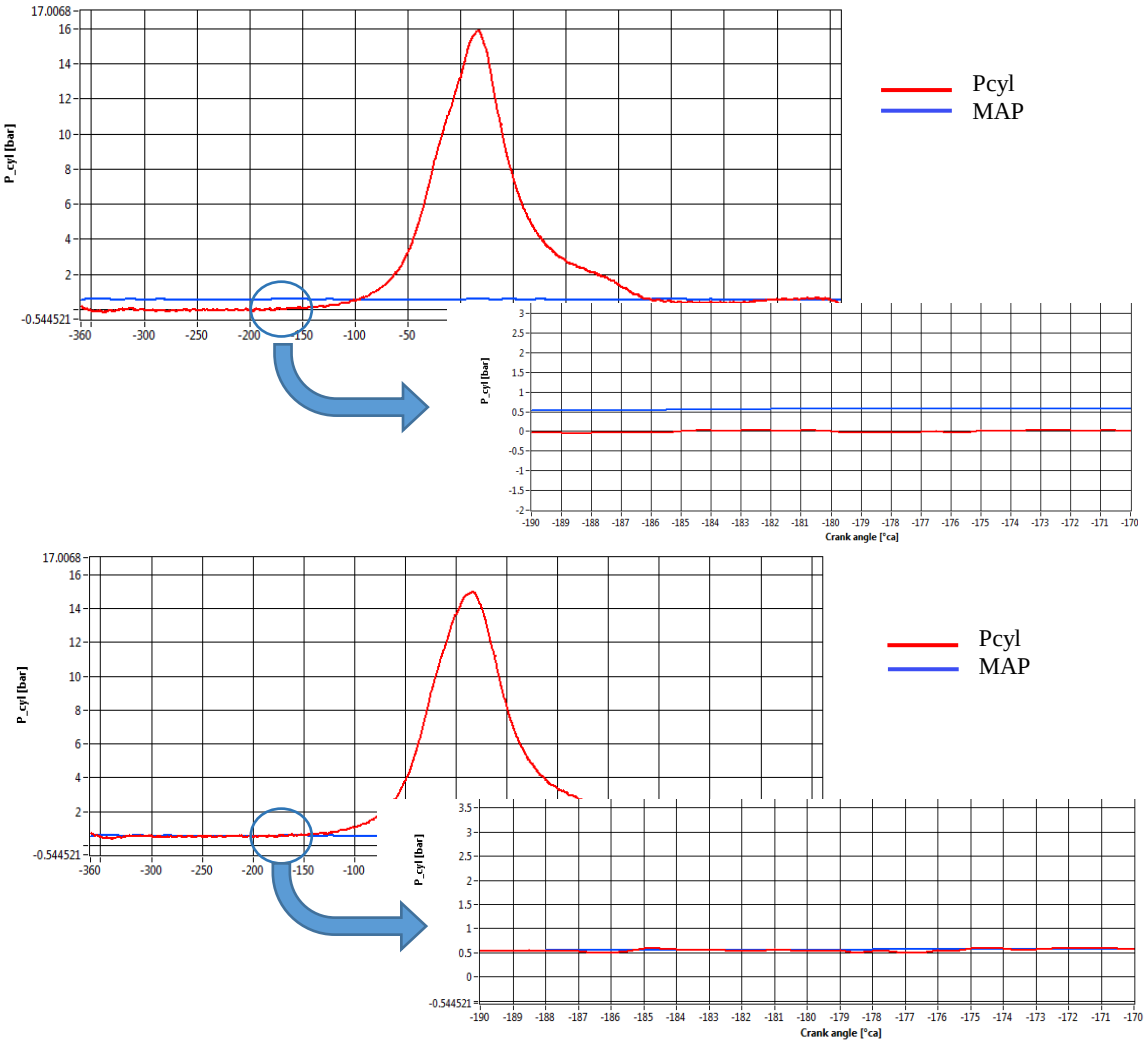


Figura 3-5: in alto, acquisizione in tempo reale con recupero della componente media disattivato. Sotto stessa acquisizione con metodo di pegging basato su MAP attivo.

Per i motori pluricilindrici diventa indispensabile il finestramento del segnale MAP. Infatti, avendo più cilindri che aspirano dallo stesso collettore è possibile utilizzare un unico segnale MAP finestrato in più sezioni, ognuna associata al corrispondente cilindro, in maniera da recuperare il corretto valore di pressione per ciascuna fase di apertura valvole al PMI di ogni cilindro. Inoltre è stata implementata la possibilità di utilizzare più

sensori di pressione collettore assoluta, permettendone l'associazione ad uno o più cilindri, in maniera da far fronte a motori con più di un collettore di aspirazione, o addirittura con sensori dedicati cilindro per cilindro (pressione runner di aspirazione).

3.3 Pegging con metodo della politropica

Visto che non sempre si ha a disposizione un sensore di pressione collettore, e considerato che avere una corretta valutazione della componente media è rilevante, specialmente nei motori sovralimentati, è stato introdotto il metodo della politropica ad esponente forzato, che permette una buona stima del valore medio della curva di pressione utilizzando solo il segnale di pressione acquisito per stimare la pressione assoluta.
Il metodo teorico consiste nello stabilire due punti angolari noti in fase di compressione (quando la curva può essere considerata una politropica) e memorizzare i valori rispettivi di pressione cilindro (acquisiti, privi di componente media) e volume (noto, conoscendo i parametri del motore e la funzione di manovellismo).
Per tali punti, 1 e 2, vale:

$$p_1 V_1^{\gamma} = p_2 V_2^{\gamma}$$

Allo stesso tempo vale:

$$p_{1reale} V_1^{\gamma} = p_{2reale} V_2^{\gamma}$$

Cioè:

$$(p_1 + \Delta p) V_1^{\gamma} = (p_2 + \Delta p) V_2^{\gamma}$$

Si può quindi ricavare il delta di pressione che deve essere applicato alla curva:

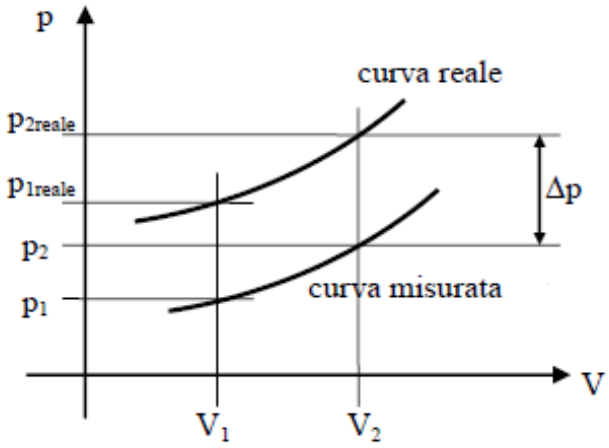


Figura 3-6: schematizzazione metodo della politropica

$$\Delta p = \frac{p_2 V_2^{\gamma} - p_1 V_1^{\gamma}}{V_1^{\gamma} - V_2^{\gamma}}$$

Dove l'esponente di politropica $\gamma = \frac{c_p}{c_v}$ deve essere valutato in base all'esperienza o a campagne di prove specifiche per il dato motore. In letteratura è possibile trovare dei valori consigliati: in particolare 1.32 per motori a carica omogenea e 1.27 per motori diesel (con corrispondenti valori angolari -110, -60°ca).

Nel caso sperimentale, a causa di rumori sul segnale, è necessario valutare il Δp per più coppie di punti e fare una media tra i vari valori ottenuti, al fine di rafforzare la validità e la correttezza del calcolo.

$$\Delta p = \frac{1}{n}\sum_{i=1}^{n}\frac{p_{2\_i}V_{2\_i}^{\gamma} - p_{1\_i}V_{1\_i}^{\gamma}}{V_{1\_i}^{\gamma} - V_{2\_i}^{\gamma}}$$

I campioni di pressione sono quelli acquisiti, mentre per quanto riguarda il volume, nota la posizione angolare, risulta noto come conseguenza della funzione di manovellismo, che si riporta di seguito per completezza:

$$l\sin(\varphi) = r\sin(\theta)$$

$$\sin(\varphi) = \frac{r}{l}\sin(\theta) = \lambda\sin(\theta)$$

$$\cos(\varphi) = \sqrt{1-\lambda^2\sin^2(\theta)}$$

$$x = r + l - (r\cos(\theta) + l\cos(\varphi))$$

$$x = r + l - r\left(\cos(\theta) + \frac{1}{\lambda}\sqrt{1-\lambda^2 sin^2(\theta)}\right)$$

$$\frac{dx}{d\theta} = r\left(sin(\theta) + \frac{\lambda\sin(2\theta)}{2\sqrt{1-\lambda^2 sin^2(\theta)}}\right) = r f_{m(\vartheta)}$$

$$dV = A\,dx = A\,r\,f_{m(\vartheta)}\,d\theta$$

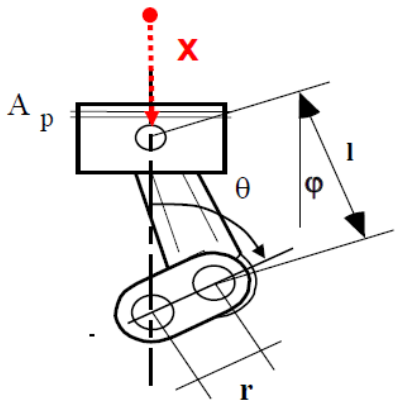


Figura 3-7: schema manovellismo di spinta

Anche in questo caso l'algoritmo non è particolarmente complicato, ma bisogna far fronte a tempistiche ristrette per il completamento del calcolo; come già sottolineato il valore di pressione assoluta deve essere disponibile prima dell'inizio del calcolo di rilascio calore, e poiché il metodo della politropica può essere applicato solo nella finestra angolare di compressione, quando le valvole sono già chiuse, sono disponibili solo pochi gradi di manovella per portare a termine le operazioni prima che inizi la combustione. Un altro problema è dato dal fatto che la disponibilità dei dati è progressiva con l'avanzamento del ciclo, ed è quindi necessario memorizzare temporaneamente tutti i primi valori che si vogliono utilizzare nell'equazione (in particolare tutti i punti p_{1_i} e V_{1_i}) fino a quando non sono disponibili i rispettivi valori p_{2_i} e V_{2_i}. In particolare, si è scelto di registrare una serie di campioni successivi variabile da 16 a 32 in modo da coprire un intervallo angolare circa costante al variare del regime: è quindi necessario salvare in una memoria un numero variabile di campioni e recuperarli in maniera ordinata non appena il

corrispettivo campione nella posizione 2 viene acquisito, in modo da completare il calcolo nel minor intervallo angolare possibile. Un'ulteriore difficoltà è data dal dovere lavorare con numeri interi in FPGA: questo impone, per velocizzare il calcolo, di pre-calcolare in real time i valori di V^{γ}e renderli disponibili in una memoria FPGA: in questo modo si riesce ad evitare il calcolo della funzione di manovellismo e l'elevamento a potenza a basso livello (entrambe richiederebbero un tempo eccessivo). Per quanto riguarda la divisione, invece, si è costretti ad eseguirla in FPGA per avere i dati disponibili al momento del calcolo di rilascio calore; questa è un'operazione lenta ed è l'anello debole dell'algoritmo. Si è verificato sperimentalmente che dallo start dell'acquisizione del secondo gruppo di campioni (cioè dallo start dei calcoli) sono necessari 15°ca per avere a disposizione il valore di Δp da utilizzare.

Nella seguente figura è riportato un esempio di recupero con politropica dove l'utente ha impostato come start di acquisizione del primo gruppo di campioni -110°ATDC e come start del secondo gruppo -65°ATDC. Come evidenziato in figura, non è possibile iniziare il calcolo del rilascio di calore prima di -65+15= -50°ATDC.

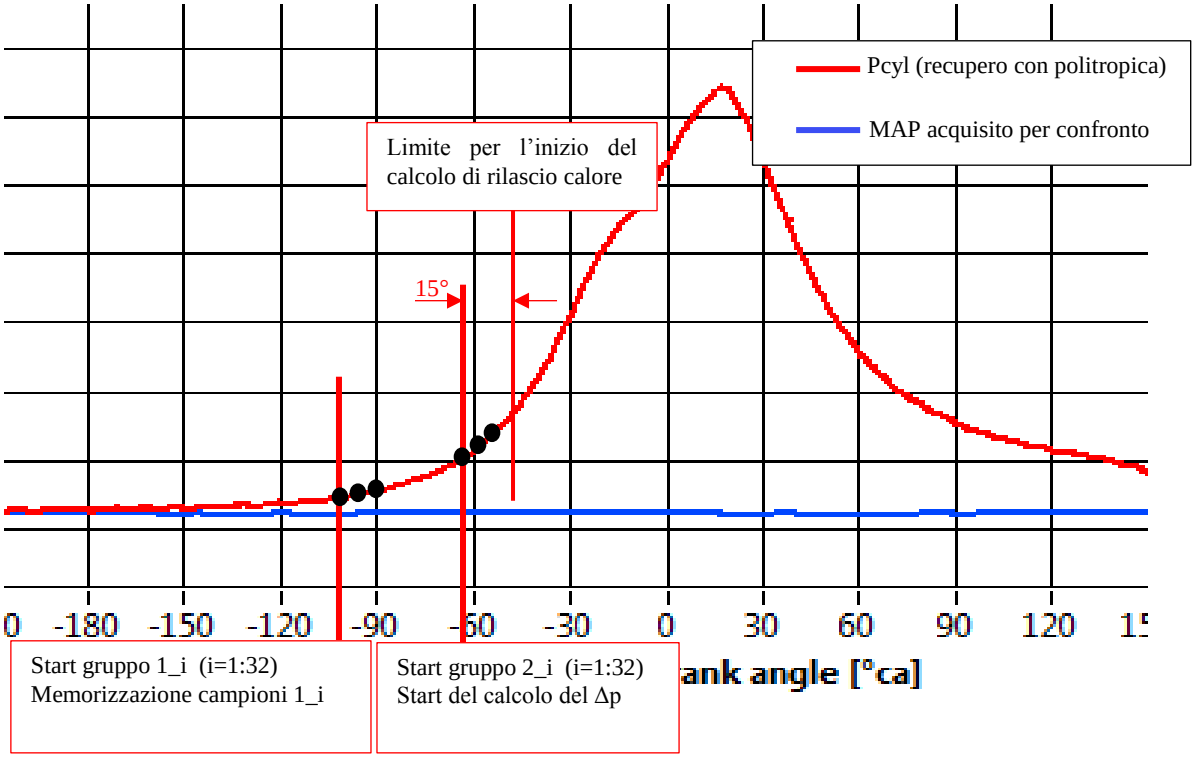


Figura 3-8:esempio di pegging con metodo della politropica

Nella figura seguente è riportata l'acquisizione di un test al banco prova per la validazione dell'algoritmo. È stato acquisito un segnale di pressione attivando il pegging con metodo

della politropica nell'intervallo [-110; -60] con esponente di politropica γ=1.32 (valori tipici in letteratura), ed in parallelo è stato acquisita la rispettiva pressione collettore mediante sensore MAP.

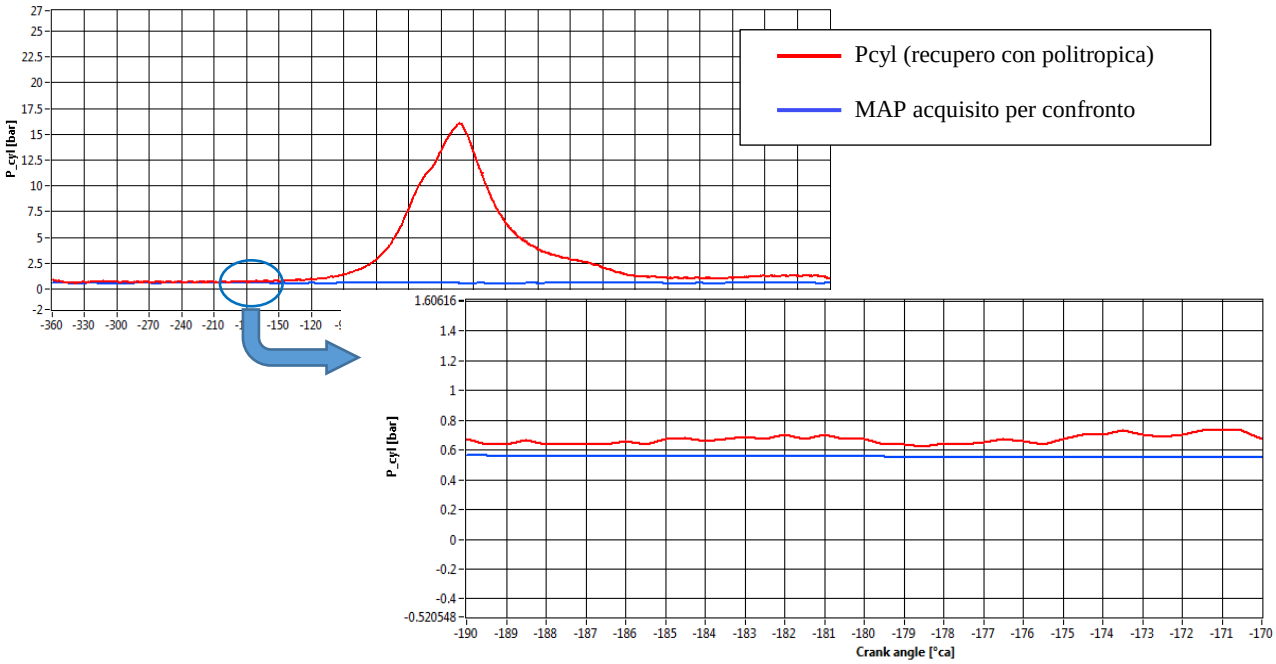


Figura 3-9: confronto segnale di pressione con componente media determinata tramite politropica e rispettiva pressione collettore acquisita con sensore MAP

Si nota come, nonostante non siano stati fatti studi sui valori dell'esponente di politropica e della posizione delle finestre di calcolo opportune per questo particolare motore, ma siano stati inseriti valori da letteratura, il recupero della componente media è compatibile con quello realmente misurato.

3.4 Ottimizzazione dell'algoritmo

Data la formula teorica per il calcolo del $\Delta p (= \frac{p_2 V_2^{\gamma} - p_1 V_1^{\gamma}}{V_1^{\gamma} - V_2^{\gamma}})$, si nota come i prodotti pV^{γ} possono essere numeri elevati nel caso in cui l'offset da recuperare sia elevato: ricordiamo infatti che p è un valore in bit non ancora recuperato in componente media, quindi potenzialmente prossimo al fondoscala dei 16bit assegnati al segnale in acquisizione. È quindi necessario dedicare un elevato numero di bit (64) alla variabile che contiene questo

prodotto. Questo obbliga di conseguenza ad una divisione a 64 bit che richiede più tempo e più risorse rispetto a divisioni con un numero di bit inferiore.
Si è pensato quindi di ottimizzare il calcolo del Δp modificando la formulazione dell'algoritmo con i seguenti passaggi:

mettendo a numeratore il termine $p_2 V_1^{\gamma} - p_2 V_1^{\gamma} = 0$

e raccogliendo, si ottiene:

$$\Delta p = \frac{-(p_1 - p_2)V_1^{\gamma} - p_2(V_1^{\gamma} - V_2^{\gamma})}{V_1^{\gamma} - V_2^{\gamma}} = \frac{(p_2 - p_1)V_1^{\gamma}}{V_1^{\gamma} - V_2^{\gamma}} - p_2$$

In questo modo, come si può notare, si ha a numeratore un prodotto di dimensioni sicuramente più piccole visto che la differenza tra le pressioni sarà sempre un numero relativamente piccolo anche per offset elevati, proprio perché i punti appartengono fisicamente alla stessa curva di compressione. Questo consente di memorizzare i dati in 32bit, e quindi procedere ad una divisione più leggera, senza rischio di overflow nel prodotto dei termini.

3.5 Pegging CAN

Il recupero della pressione media tramite dati ricevuti da linea CAN è un'evoluzione del recupero a punto fisso. È stato possibile introdurre questa funzionalità solo dopo l'introduzione della lettura di pacchetti CAN da una rete esistente (di cui si parlerà in seguito). Il metodo, analogamente al metodo a punto fisso, consiste nel calcolare il valore medio del segnale acquisito nell'intervallo di interesse (intorno del PMI) per poi confrontarlo con il valore di pressione collettore proveniente dal bus CAN e ricavare così il Δp. La pressione collettore aspirazione è infatti una informazione presente in centralina e tendenzialmente disponibile su linea CAN ECU. Una volta caricato il corretto database in lettura, è stata creata un'interfaccia che consente all'utente di selezionare quale tra i segnali a disposizione sulla linea utilizzare come riferimento per il recupero della componente media di pressione.

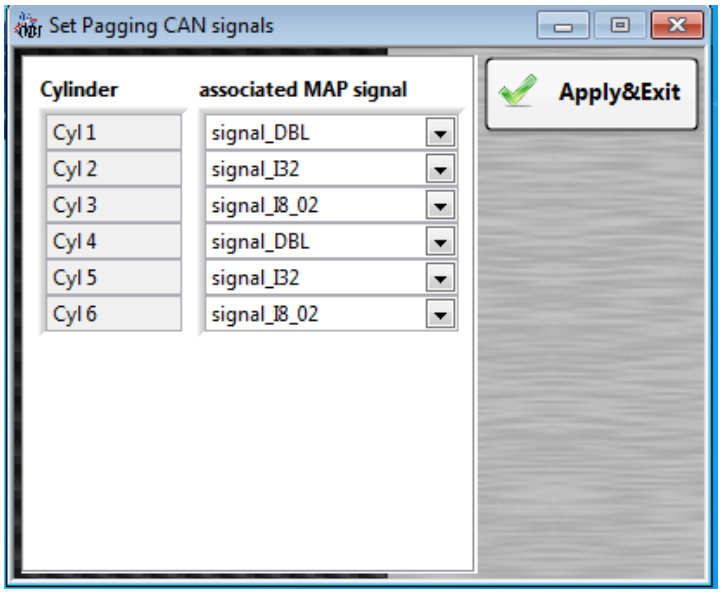


Figura 3-10: interfaccia di configurazione segnale CAN per riferimento pressione collettore

È stato reso possibile associare un differente segnale CAN per ogni cilindro, o viceversa utilizzare lo stesso valore per più cilindri, in relazione a ciò che si ha a disposizione sulla linea CAN banco o veicolo.

Per quanto riguarda l'accuratezza di questo metodo, il risultato dipende dalla frequenza di scrittura e lettura del parametro monitorato sulla linea CAN. Risulta ovvio che per procedere ad un recupero corretto ciclo dopo ciclo, il valore di riferimento deve essere aggiornato e disponibile sul bus CAN prima che il sistema inizi il calcolo del rilascio di calore. Il sistema è stato ottimizzato per monitorare aggiornamenti del dato di riferimento fino all'angolo di manovella (stabilito dall'utente) in cui si inizia il calcolo del rilascio di calore: nel caso in cui il valore su bus CAN non sia stato aggiornato, si utilizza quello del ciclo precedente. La frequenza di lettura dati da CAN dovrà essere impostata in maniera idonea al tipo di motore che si sta monitorando: ad esempio se il regime massimo è di 6000rpm, sarà necessario impostare la frequenza di lettura CAN ad un valore inferiore ai 20ms se si vuole un aggiornamento ciclico del dato.

3.6 Problematiche relative al pegging

Gli algoritmi di calcolo della componente media sono fortemente influenzati dalla qualità del segnale e dall'intervallo di calcolo impostato dall'utente. Ad esempio se la finestra

scelta nell'intorno del punto morto inferiore è poco ampia, ed il segnale presenta del rumore, la media calcolata non rispecchierà il valore reale di pressione: i valori tipici della finestra per evitare questo problema si aggirano intorno ai 20°ca di ampiezza.

Anche la scelta delle finestre di calcolo per il metodo della politropica ha un peso influente sul risultato, non solo a causa di eventuali rumori elettrici sul segnale, ma per la possibile presenza di rumore chiusura valvole o carica bobina: ricordiamo infatti che il metodo della politropica deve essere applicato nella fase di compressione dove la probabilità di trovarsi di fronte a queste tipologie di disturbi è elevata. In particolare, se si sceglie un finestramento troppo anticipato, si rischia di cadere nel tratto in cui il rumore di chiusura della valvola di aspirazione disturba il segnale: allo stesso modo, se si ritarda troppo il recupero, si rischia di entrare in una zona dove la combustione sta iniziando e quindi le ipotesi di curva politropica non sono più rispettate, e si commette quindi un errore nel calcolo.

Si noti nella seguente acquisizione fatta al banco su un motore caratterizzato da un forte rumore valvole, come la scelta di coppie di punti errate, causata dal rumore di chiusura valvole, influenzi pesantemente la ricostruzione della curva di compressione e quindi la determinazione del valore di offset.

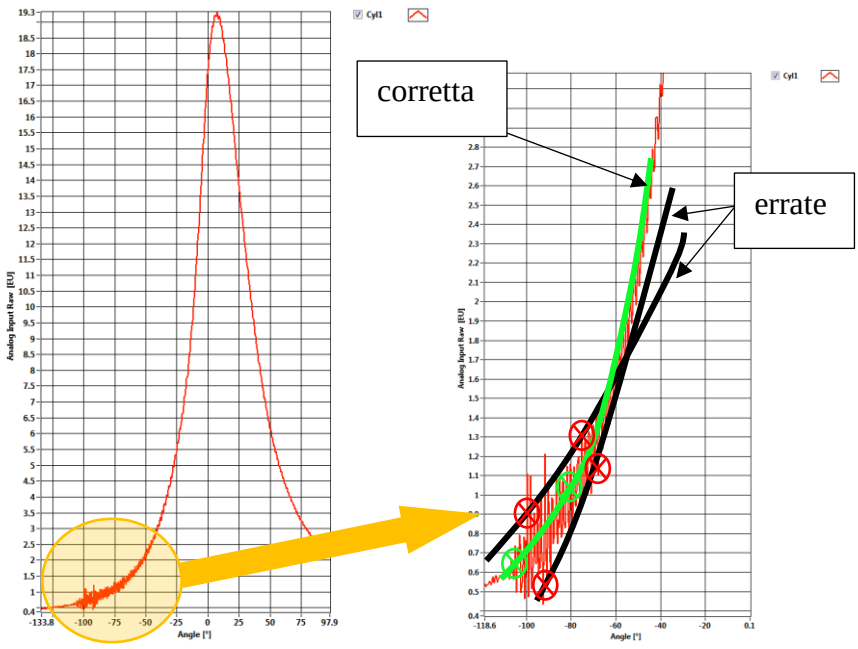


Figura 3-11: ricostruzione errata della fase di compressione a causa di rumore valvole

Anche un eventuale rumore di carica bobina (mostrato nella seguente acquisizione, fatta al banco prova di Bologna su un motore VW TSI 1.4) può portare ad errori sul calcolo se la finestra scelta è troppo ritardata.

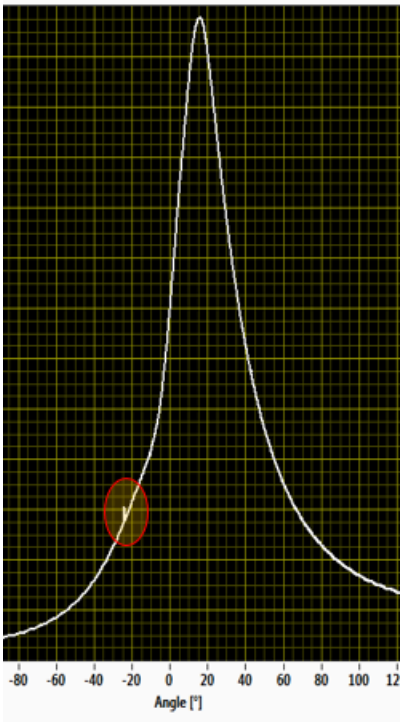


Figura 3-12: esempio di rumore dovuto a carica bobina

4. TRACKING ATTUAZIONI

I motori a combustione interna moderni sono caratterizzati da un sistema complesso per il controllo e la gestione ottimizzati della combustione, e quindi delle prestazioni, non solo in termini di performance ma anche di inquinanti e guidabilità. Per questo non solo è necessario l'analisi dell'andamento della pressione in camera di combustione, ma diventa di fondamentale importanza conoscere come si comporta il motore stesso in funzione delle azioni esterne di controllo, che devono quindi essere monitorate. Nasce così la necessità di inserire all'interno del sistema di analisi della combustione un algoritmo che permetta all'utente di tracciare all'interno del ciclo motore l'andamento dei segnali esterni di controllo, in termini di posizione angolare e durata temporale. In particolare, i segnali di maggiore interesse sono quelli di iniezione e carica bobina, ma allo stesso tempo può essere di interesse tracciare la posizione angolare di un segnale di comando VVT o di qualsiasi altra attuazione fasata con il ciclo motore (ad esempio comando valvola di regolazione pressione rail). Per questo si è deciso di sviluppare un algoritmo generico che possa essere applicato alla più svariata gamma di segnali.

4.1 Algoritmo implementato

Le informazioni che l'utente finale di un sistema di analisi combustione si aspetta di ottenere riguardo ad una attuazione, dipendono dal tipo di attuazione che si sta analizzando. Ad esempio, nel caso di iniezione, i parametri monitorati sono l'angolo di inizio e la durata, mentre per la carica bobina il momento più importante è l'angolo di fine carica (cioè l'anticipo di accensione effettivo). Quindi l'algoritmo di analisi sviluppato è stato pensato per fornire all'utente le informazioni di inizio e fine attuazione in termini angolari e la loro durata temporale. Inoltre vista la possibilità di attuazioni multiple all'interno del ciclo motore (un caso su tutti quello delle multi iniettate nei motori diesel) deve essere possibile individuare quante attuazioni ci sono state all'interno del ciclo ed analizzarle tutte.

L'algoritmo di analisi si basa sul riconoscimento dell'attraversamento di soglie definite dall'utente; in particolare:

- Si deve definire su quale canale ci si aspetta l'attuazione, ed a quale cilindro sarà relazionata l'attuazione misurata (ad esempio comando di iniezione relativo al cilindro 1)
- L'utente seleziona la posizione di tre soglie come percentuale dell'ampiezza (picco-picco) dell'attuazione da analizzare. Sulla base dell'attraversamento delle soglie fissate, il sistema calcolerà gli angoli di inizio e fine attuazione

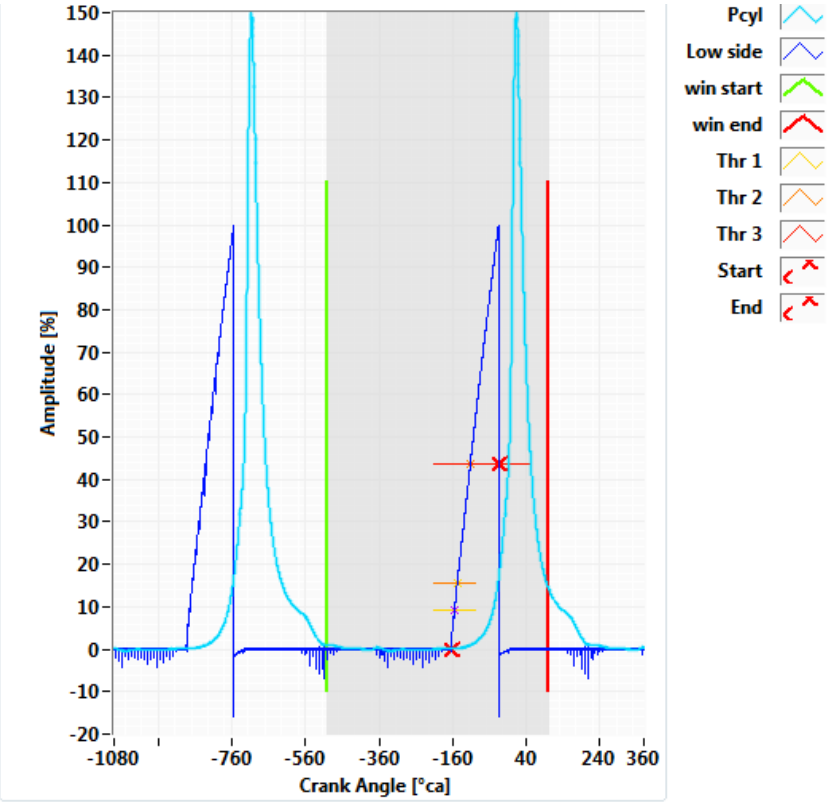


Figura 4-1: soglie e finestra di analisi fissate per un segnale di carica bobina

- È necessario definire una finestra di analisi all'interno della quale l'utente si aspetta di trovare l'attuazione, con riferimento relativo al TDC del singolo cilindro. Questo è necessario per poter utilizzare la funzione di multiplex: tipicamente un segnale di attuazione viene acquisito mediante pinza amperometrica, che permette di misurare la corrente assorbita da un'attuazione trasformandola in un segnale in tensione. Risulta comodo utilizzare la stessa pinza (per motivi sia di costo che di ingombro) contemporaneamente su più bobine/iniettori, avendo così sullo stesso canale analogico più attuazioni riferite a cilindri diversi. Il finestramento permette di associare i risultati correttamente ad ogni rispettivo cilindro, evitando di "incrociare" attuazioni e cilindri in maniera errata.

- Il sistema determina i punti di attraversamento delle soglie, memorizzando la posizione angolare e il valore in tensione misurato per ogni soglia attraversata in salita. Per quanto riguarda la transizione in discesa, vengono memorizzati posizione e valore solo per la soglia più alta. Questo perché in salita, vista la forma tipica dei segnali di attuazione (in particolare per la carica bobina), per ricostruire correttamente il punto di start, sarà necessario procedere ad una interpolazione, mentre per la discesa, essendo sempre un fronte ripido, questo non è necessario.

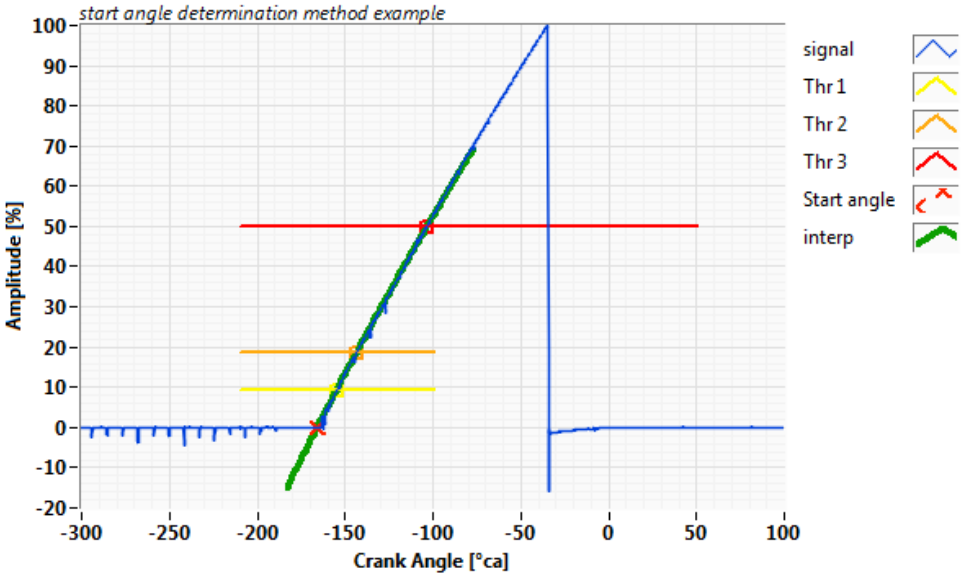


Figura 4-2: esempio di estrapolazione dei punti acquisiti per determinare l'effettivo start

- Il sistema interpola i tre valori di attraversamento in salita per ricostruire l'andamento della curva, per poi estrapolare il punto di start effettivo.
- Il punto di fine coincide con l'attraversamento in discesa della soglia più alta.
- Noti lo start e l'end angolari, il sistema calcola la durata temporale dell'attuazione, ipotizzando la velocita costante all'interno del ciclo motore.
- Riconosciuta la fine della prima attuazione, il sistema si mette in attesa per una seconda (aspettando un nuovo attraversamento delle soglie in salita) fino a quando rimane all'interno dell'intervallo angolare selezionato dall'utente.
- Terminato l'intervallo angolare di interesse, il sistema pubblica i valori calcolati verso il PC host, su bus CAN e su protocollo XCP [3] in relazione ai settings impostati dall'utente.

4.2 Posizionamento automatico delle soglie

Come detto, le soglie vengono posizionate come valore percentuale rispetto al picco del segnale misurato. Non sarà però possibile, dovendo fornire dei risultati in tempo reale, aspettare il picco effettivo del ciclo corrente per battezzare le soglie, in quanto le risorse FPGA non permettono di memorizzare tutti i valori della curva per 12 cilindri e determinare le transizioni a posteriori. Sarà quindi necessario determinare un valore atteso valido di picco rispetto al quale fissare le soglie, utilizzando uno storico dei valori picco-picco dell'attuazione misurata. Lo storico dovrà tenere conto del valor medio del picco, eliminando però eventuali spike dovuti a rumore elettrico e, soprattutto, mancate attuazioni, che porterebbero idealmente ad un valore nullo del picco e quindi all'abbattimento delle soglie verso lo zero, portando così a confondere l'individuazione delle transizioni con del rumore elettrico.

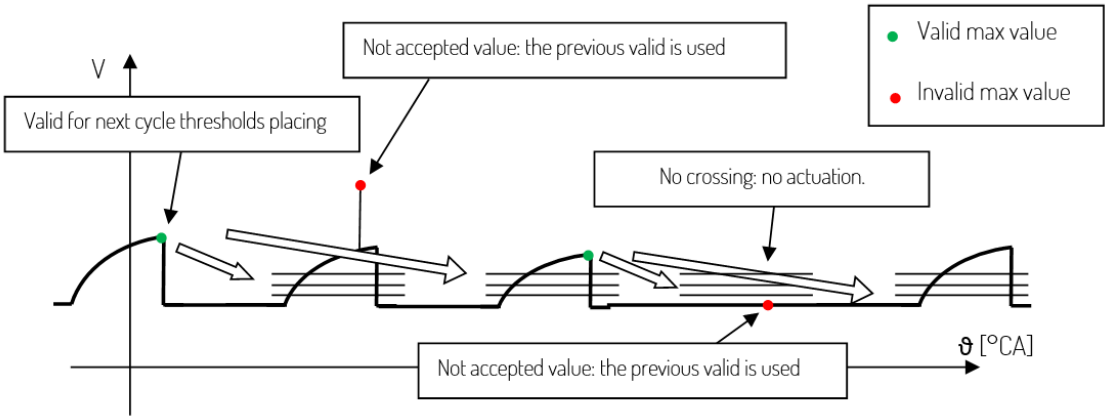


Figura 4-3: schematizzazione della logica di determinazione dei picchi validi per il piazzamento soglie

Come si vede dalla figura, i valori considerati validi vanno a contribuire al piazzamento delle soglie per il ciclo successivo, mentre i valori fuori range non sono considerati validi per il calcolo delle soglie. Se infatti si considerassero valori fuori range si andrebbe incontro a due differenti possibilità:

- innalzamento delle soglie: se vengono introdotti come validi i picchi superiori ad un certo limite, le soglie tenderebbero ad essere sempre più alte, portando così ad una mancata transizione attraverso la soglia stessa, cioè ad una segnalazione di falsa mancata attuazione

- appiattimento a zero delle soglie: se si considera come esempio un caso di monitoraggio carica bobina in cut-off durante un rilascio (dove l'attuazione è volutamente assente per un tempo elevato) si avrebbe come conseguenza una media del picco-picco del segnale tendente a zero (a meno del rumore), e quindi un setting delle soglie nell'intorno dello zero, causando così il riconoscimento di una transizione confusa con il rumore stesso del segnale, determinando quindi un numero molto elevato di false attuazioni.

4.3 Condizionamento del segnale

L'algoritmo di posizionamento delle soglie è stato descritto per un segnale ideale, cioè con media nulla e che si incrementa in senso positivo. Nella realtà, quando ci si trova a confronto con segnali realmente acquisiti al banco (o su veicolo), nessuna di queste due caratteristiche è necessariamente verificata. Sarà quindi indispensabile riportare il segnale misurato in condizioni ideali. Per prima cosa si procede al riposizionamento sullo zero, eseguendo un pegging a punto fisso, fissando il valore di referencing a zero. In particolare, si procede a fare una media su 16 campioni nell'intorno dello start della finestra angolare selezionato dall'utente, ricavando così l'offset da applicare a tutta la curva per imporla a zero in quell'intervallo. Per quanto riguarda il verso di incremento del segnale (positivo o negativo) dipende dal verso di scorrimento della corrente rispetto alla pinza amperometrica, cioè dal verso di montaggio della pinza. Spesso ci si ritrova con la pinza invertita una volta già chiuso il cablaggio: risulta quindi scomodo smontare e rimontare il tutto. Di conseguenza è opportuno che sia il software a invertire il segnale automaticamente o mediante azione dell'utente. In questo caso si è scelto di farlo automaticamente, utilizzando un valore assoluto dopo aver fatto il pegging a zero del segnale: questo consente di ottenere un segnale idoneo per essere interpretato dall'algoritmo di determinazione dei massimi validi.

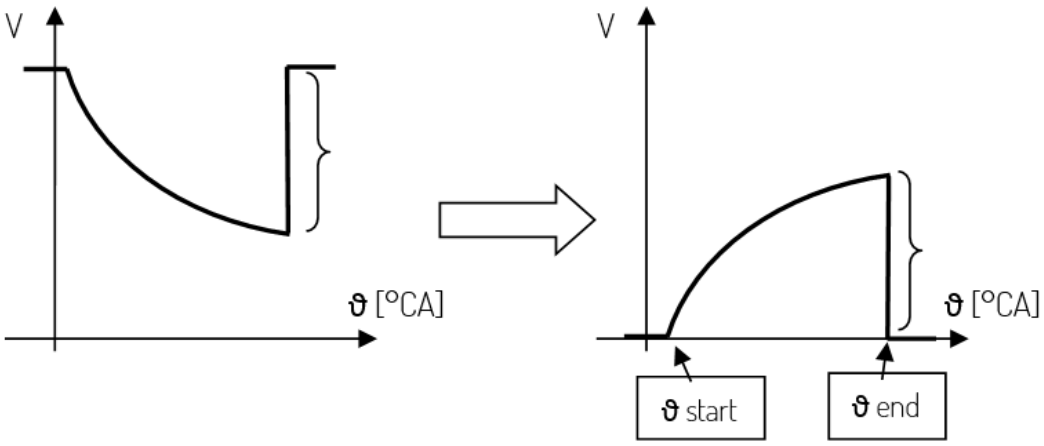


Figura 4-4: esempio di condizionamento del segnale di attuazione

4.4 Tipi di segnale ed estrapolazione dello start

Come già sottolineato, l'algoritmo sviluppato è stato pensato per permettere all'utilizzatore di caratterizzare una qualsiasi generica attuazione: si deve quindi prendere in considerazione una vasta gamma di possibili differenti segnali acquisibili davanti ai quali può trovarsi l'utilizzatore.

Uno dei segnali tipici è quello di carica bobina:

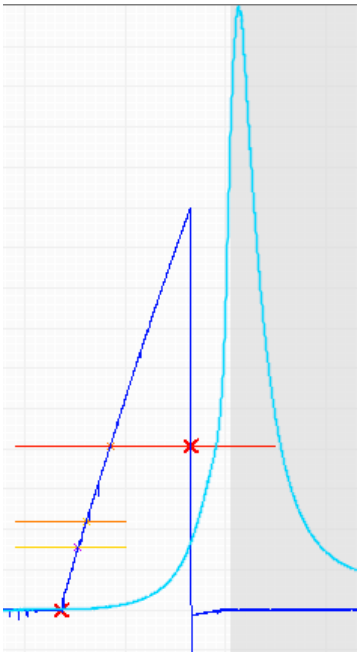


Figura 4-5: esempio di attuazione: carica bobina

Tipicamente i valori di interesse relativi a questo segnale sono la fine carica (che equivale alla scarica bobina, cioè all'anticipo di accensione) e la durata di carica, per stimare l'energia accumulata e quindi liberata con la scintilla. Per stabilire l'angolo di scarica, data la pendenza del fronte, è sufficiente determinare l'angolo di attraversamento della soglia 3 in discesa. Per quanto riguarda il punto di start invece, è necessario procedere ad un'interpolazione utilizzando le coordinate dei tre punti di attraversamento delle soglie, per poi estrapolare il punto di zero crossing.

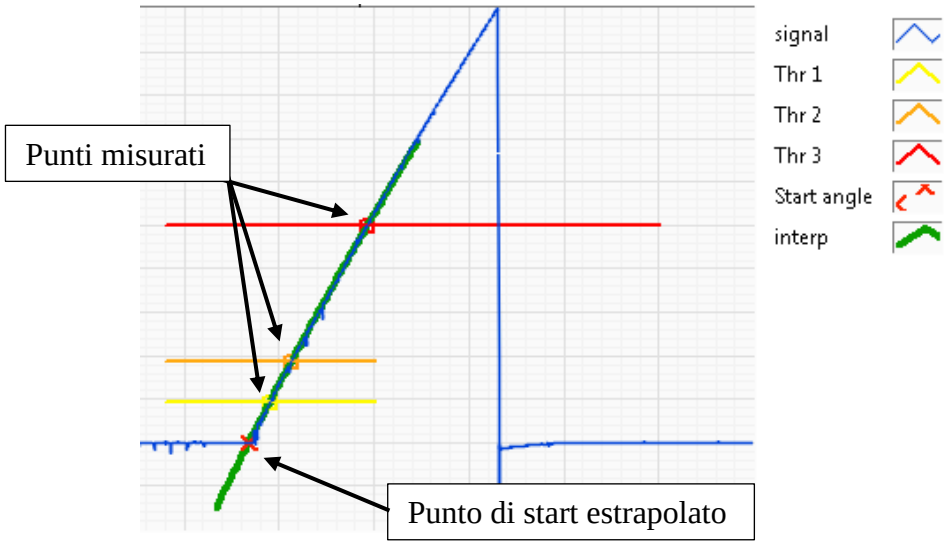


Figura 4-6: estrapolazione del punto di start per carica bobina

Il software consente all'utente di scegliere diversi tipi di interpolazione (parabolica, linear fit su 3 punti, lineare utilizzando solo due punti) oppure consente di evitare l'interpolazione ed utilizzare una sola soglia (a scelta tra le tre) anche per la salita. Si noti come l'utilizzo di una soglia per questo tipo di segnale porta ad errori considerevoli sul piazzamento dell'angolo di start.

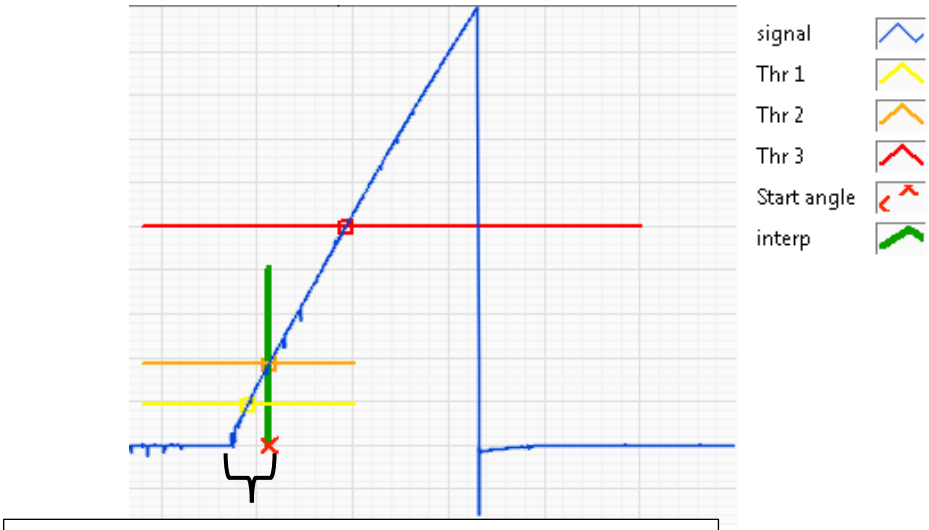


Figura 4-7: esempio di determinazione dello start utilizzando un unico valore di crossing

Una seconda tipologia di segnale molto diffuso è il peak and hold, tipico segnale di comando per gli iniettori.

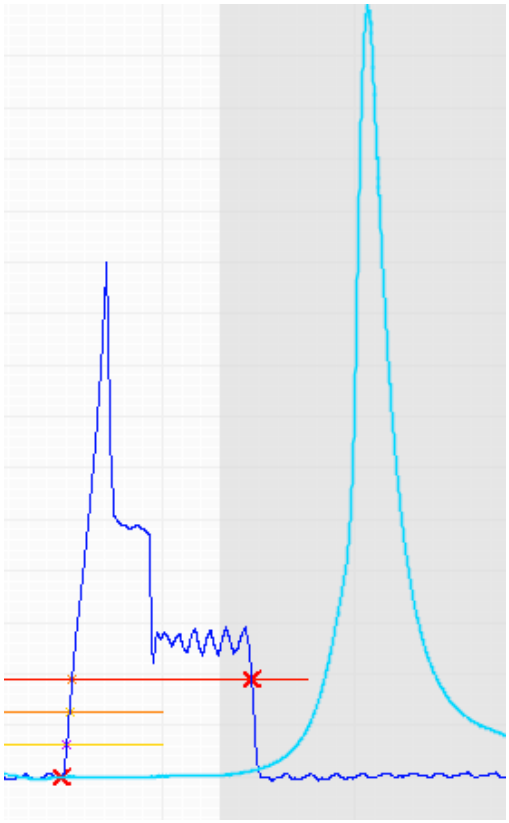


Figura 4-8: esempio di attuazione: pick and hold

In questo caso i valori di maggior interesse sono l'angolo di inizio iniezione e la durata dell'iniezione stessa. Per quanto riguarda l'angolo di start anche in questo caso è

opportuno utilizzare l'interpolazione o il fitting sui tre punti; l'errore commesso sarebbe comunque minore rispetto al caso di carica bobina, in quanto il fronte di salita risulta solitamente essere piuttosto ripido. Nella figura seguente è evidenziato (esagerando la pendenza del segnale) l'eventuale errore che si potrebbe commettere.

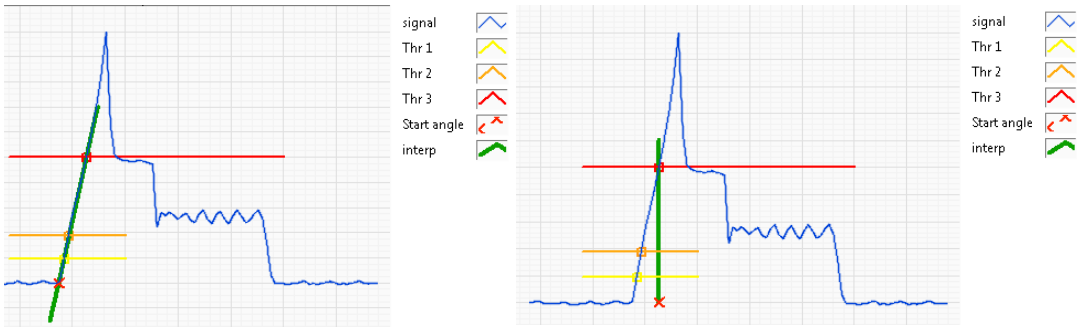


Figura 4-9: esempi di differenti estrapolazioni start per segnali pick and hold

Per quanto riguarda la fine dell'attuazione con questo tipo di segnale è possibile incorrere in un errore grossolano: infatti piazzando una soglia eccessivamente elevata, si rischia di determinare la fine dell'attuazione nel momento di discesa del peak, e non al termine effetivo della fase di hold.

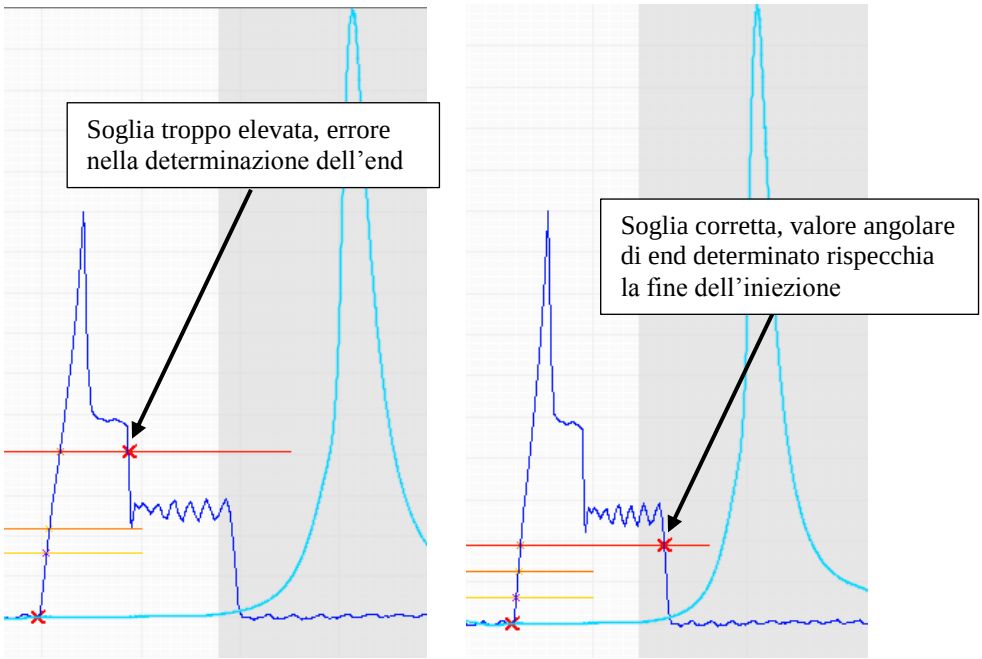


Figura 4-10: esempio di errato piazzamento della soglia in discesa

Sarà cura dell'utente prestare attenzione alla corretta selezione delle percentuali a cui piazzare le soglie in funzione del tipo di segnale peak and hold che sta acquisendo.

Una funzionalità interessante sarebbe la determinazione della durata della fase di peak e della fase di hold: questa funzione non è stata introdotta al momento perché richiederebbe una seconda soglia in discesa, non implementata.
Il segnale di attuazione più generico è il comando logico squadrato: questo segnale di comando può essere utilizzato sia come controllo per la carica bobina sia come controllo per l'apertura degli iniettori, utilizzando un TTL a pochi mA, prima delle rispettive parti di potenza.

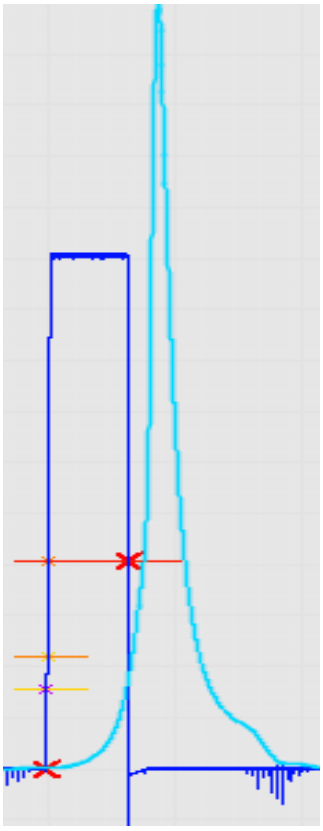


Figura 4-11: esempio di attuazione: comando logica squadrato

Questo tipo di segnale presenta due fronti molto ripidi, quindi si può utilizzare un'unica soglia anche per la determinazione dello start senza commettere errore. In ogni caso il software permette all'utente la scelta del metodo da utilizzare.

4.5 Iniettori piezoelettrici

Oltre alle tipologie di segnale descritte nel paragrafo precedente, è stata dedicata particolare cura alla possibilità di analizzare l'attuazione per iniettori di tipo piezoelettrico. Per questa tipologia di iniettori l'apertura e la chiusura vengono comandate

con due impulsi di segno opposto e non imponendo una certa corrente di mantenimento per far sì che lo spillo rimanga aperto.

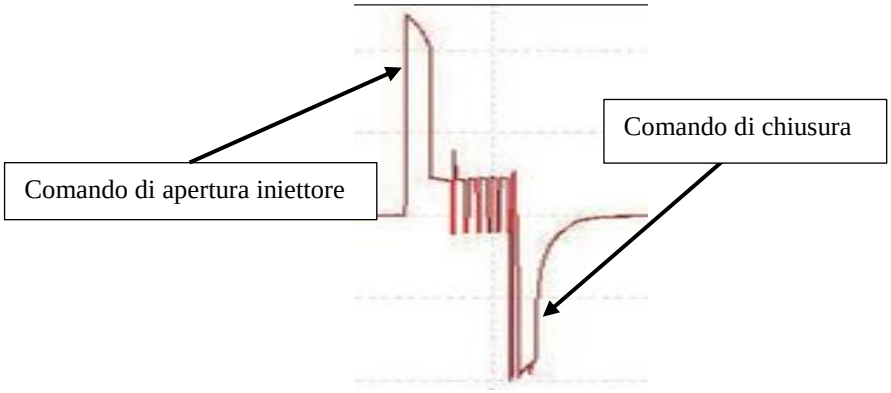


Figura 4-12 esempio di comando apertura e chiusura per iniettori piezoelettrici

Dando in pasto all'algoritmo descritto questo tipo di segnale, il risultato è che il sistema riconosce due attuazioni distinte di breve durata, non avendo a disposizione alcuna informazione su come differenziare una apertura da una chiusura. È stato quindi necessario sviluppare una parte di codice per caratterizzare il segnale di un iniettore piezoelettrico prima di poterlo analizzare con l'algoritmo generico descritto. In particolare, questo tipo di segnale viene squadrato in tempo reale prima di entrare in pasto all'algoritmo di tracking come un segnale logico squadrato.

Per poter generare un segnale squadrato partendo da un segnale di comando per iniettore piezoelettrico, è necessario stabilire due soglie, una di salita, positiva, ed una di nuovo di salita, ma per valori negativi. Quando la prima soglia è oltrepassata, si determina l'apertura dell'iniettore, quindi comando logico alto: quando anche la seconda soglia viene oltrepassata, si determina la fine dell'iniezione e quindi la chiusura del comando logico.

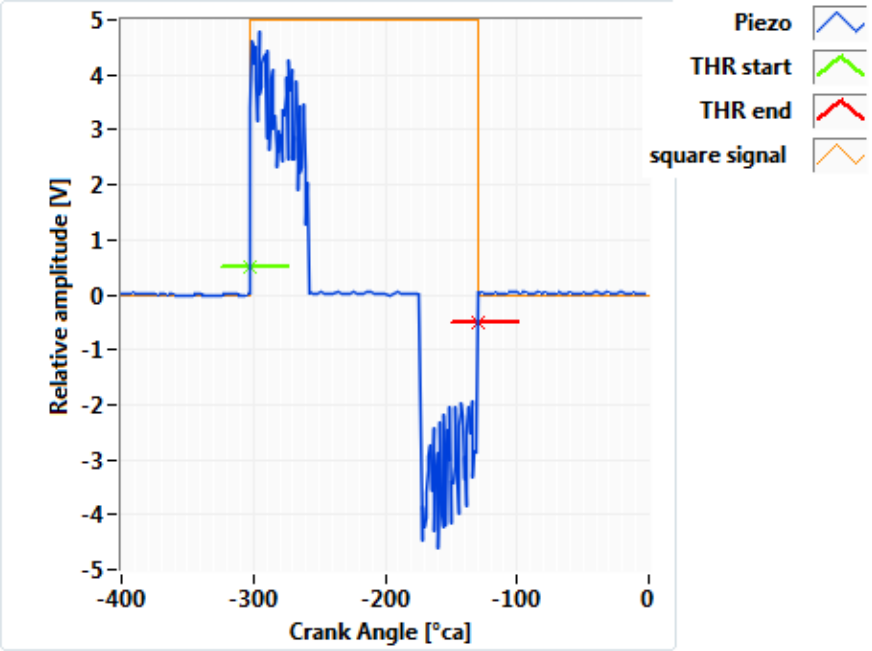


Figura 4-13: esempio di condizionamento del segnale per iniettori piezoelettrici

Una volta pre-trattato il segnale, questo viene analizzato dall'algoritmo di analisi attuazioni come un qualsiasi segnale squadrato.

4.6 Effetto del filtraggio sul fronte

Come visto, per quello che riguarda la determinazione della fine dell'attuazione, l'algoritmo si basa sull'attraversamento di un'unica soglia, soluzione scelta basandosi sull'ipotesi forte che il segnale in quella fase presenta sempre un fronte ripido. Questa ipotesi può venire meno, e quindi portare a commettere un errore, nel caso in cui si proceda all'analisi di un segnale filtrato. Dalle figure seguenti si può infatti notare come cambi il fronte di discesa nel caso di segnale filtrato o meno, e come nasca di conseguenza una imprecisione sulla posizione dell'angolo di fine attuazione in funzione della posizione percentuale della soglia

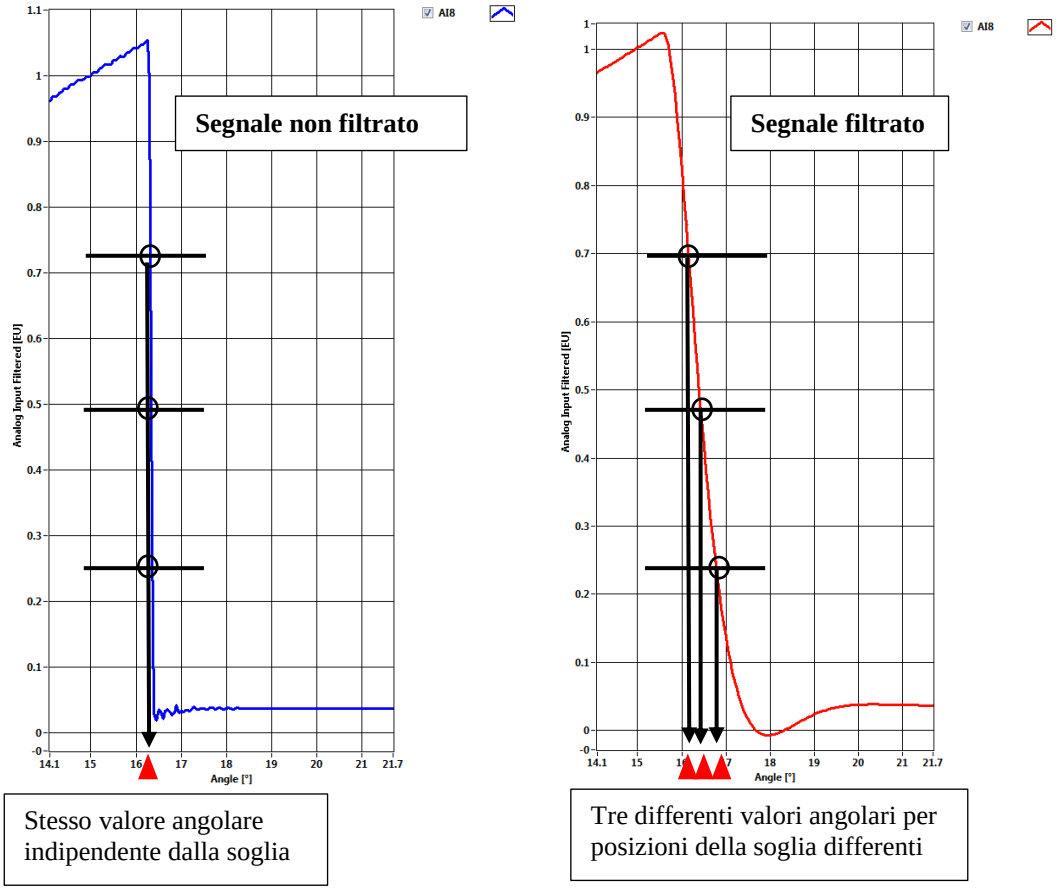


Figura 4-14: confronto tra fronte di discesa filtrato e non filtrato

Si nota quindi come sia opportuno non filtrare il comando di attuazione, a meno che non sia affetto da rumori elettrici importanti, ed anche in questo caso il filtraggio dovrà essere leggero, solo al fine di eliminare spike ad alte frequenze, senza distorcere eccessivamente la curva.

Dovendo associare i valori di inizio e fine attuazione ad una posizione angolare riferita ad un determinato cilindro, è necessario che i campioni del segnale di attuazione siano coerenti, in termini temporali, con quelli del segnale di pressione a cui viene associato un determinato angolo. Introducendo un filtraggio diverso (o meglio un non-filtraggio) per i segnali di attuazione rispetto ai campioni di pressione (che vengono filtrati passa basso) è necessario tenere conto del ritardo differente che hanno i due segnali. È stato quindi necessario introdurre una moria aggiuntiva per compensare il ritardo dei dati filtrati rispetto ai campioni dell'attuazione acquisita, mantenendo questi ultimi "in attesa" in una memoria FPGA dedicata.

4.7 Pubblicazione dei risultati

Per consentire all'utente di visualizzare in maniera efficace ad intuitiva i risultati calcolati, è stata sviluppata un'interfaccia che permette la visualizzazione i dati calcolati su base cilindro o su base evento[2]:

- Base cilindro: consente di visualizzare tutti gli eventi rilevati per un singolo cilindro
- Base evento: consente di visualizzare un determinato evento per tutti i cilindri abilitati

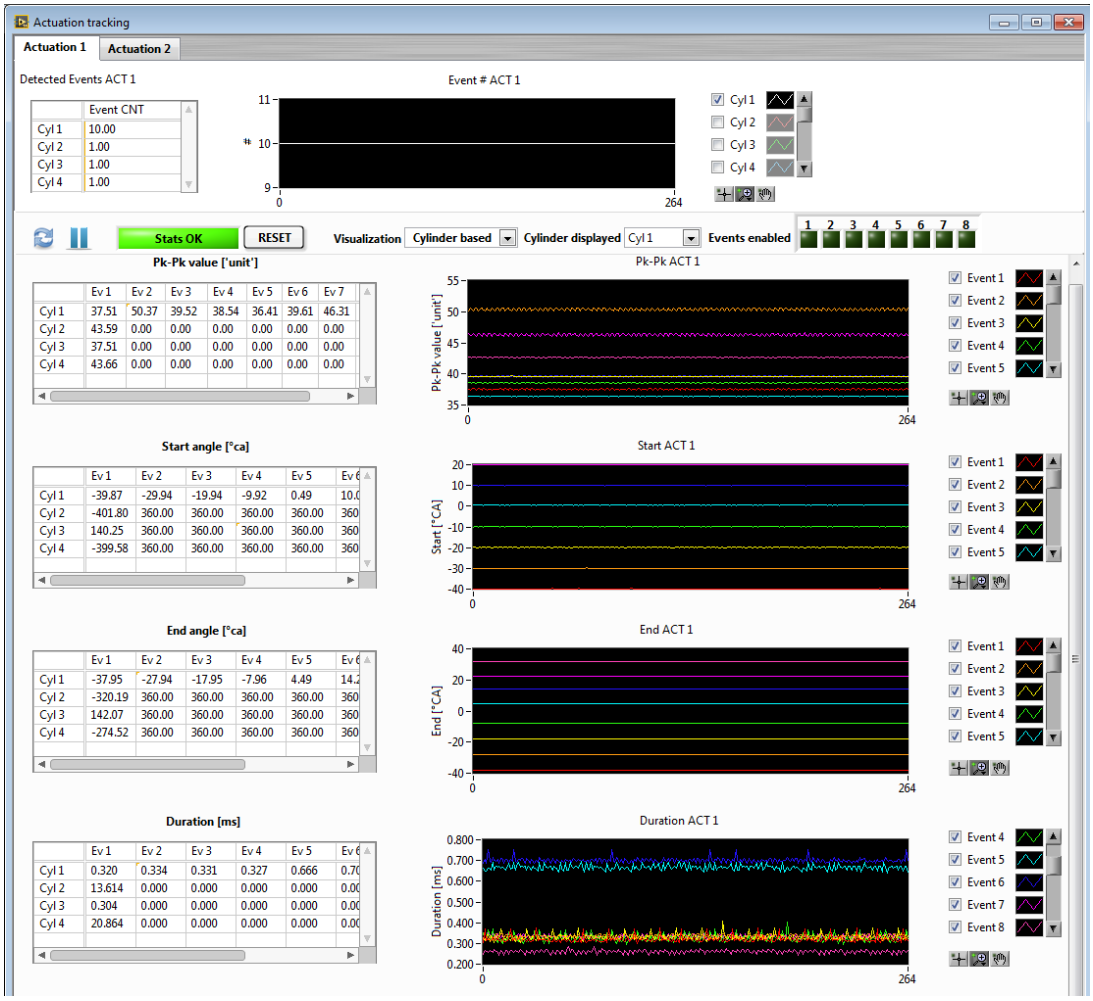


Figura 4-15: interfaccia di visualizzazione risultati tracking attuazioni

[2] Si definisce come **evento** ogni attuazione riconosciuta all'interno di un ciclo motore. Ad esempio, un pattern di iniezione formato da pre, main e post, viene riconosciuto come un'attuazione con tre eventi all'interno del ciclo motore, e quindi vengono calcolati tre valori di start, tre di end e tre durate distinti.

Oltre ad una visualizzazione in tempo reale, i dati calcolati vengono salvati all'interno del file sintetico che raccoglie tutti i risultati relativi all'analisi indicating ed allo stesso tempo, a livello real time, vengono pubblicati ciclo dopo ciclo su bus CAN e/o su protocollo XCP via ethernet (in seguito verranno descritte le potenzialità di questi tipi di comunicazione)

4.8 Problematiche e limitazioni di implementazione

La principale difficoltà di implementazione dell'algoritmo si è presentata al momento dell'integrazione in FPGA dello stesso. Infatti, la caratteristica di analisi in tempo reale richiede elevate risorse a livello di memorie e moltiplicatori. Questo ha obbligato ad un lavoro di ottimizzazione del codice, non solo di quello nuovo, necessario per il tracking, ma anche ad una revisione generale di tutto il codice già implementato, per cercare di ottimizzare al meglio lo sfruttamento delle risorse e delle memorie disponibili in FPGA. Al termine di questa fase si è riusciti ad ottenere la possibilità di registrare fino ad 8 eventi per cilindro per un massimo di 6 cilindri. Nel caso in cui invece il numero di cilindri del motore in analisi sia superiore a 6, il numero di eventi acquisibili per ciclo scende a 4. Questo compromesso è dovuto alla limitata memoria FPGA, ma è stato valutato come un compromesso accettabile, in quanto le multi-attuazioni sono tipiche di motori diesel che raramente hanno un numero di cilindri superiore a 6.

Per quanto riguarda il calcolo della durata temporale dell'attuazione ci si basa sulla velocità media nel ciclo, convertendo il delta angolare in un delta temporale basandosi su quel valore. Questo porta a commettere un errore tanto maggiore quanto maggiore è l'oscillazione di velocità rispetto al valore medio nel tratto angolare di interesse. Per evitare questo errore sarebbe necessario avere a disposizione oltre alla posizione angolare dell'attraversamento soglia, anche il momento temporale dell'attraversamento: questo non è possibile a causa della limitata memoria FPGA. Sarebbe infatti necessario salvare in memoria tre ulteriori valori per le soglie in salita per ogni evento, e fare una seconda interpolazione utilizzando i tempi al posto degli angoli, operazioni al momento non compatibili con le prestazioni dell'hardware (si ricorda che il sistema deve fornire i risultati in tempo reale entro il ciclo motore successivo per ogni cilindro).

4.9 Acquisizioni e test al banco

L'algoritmo è stato testato al banco prova su un motore FIAT multijet 1300, dove è stato acquisito un segnale di pressione cilindro e il relativo pattern di iniezione formato da due pre-iniezioni ed una main. Di seguito è riportata l'acquisizione del segnale ad alta frequenza (un solo ciclo esemplificativo in figura) ed i risultati sintetici forniti dal sistema lungo la prova (60 cicli in figura).

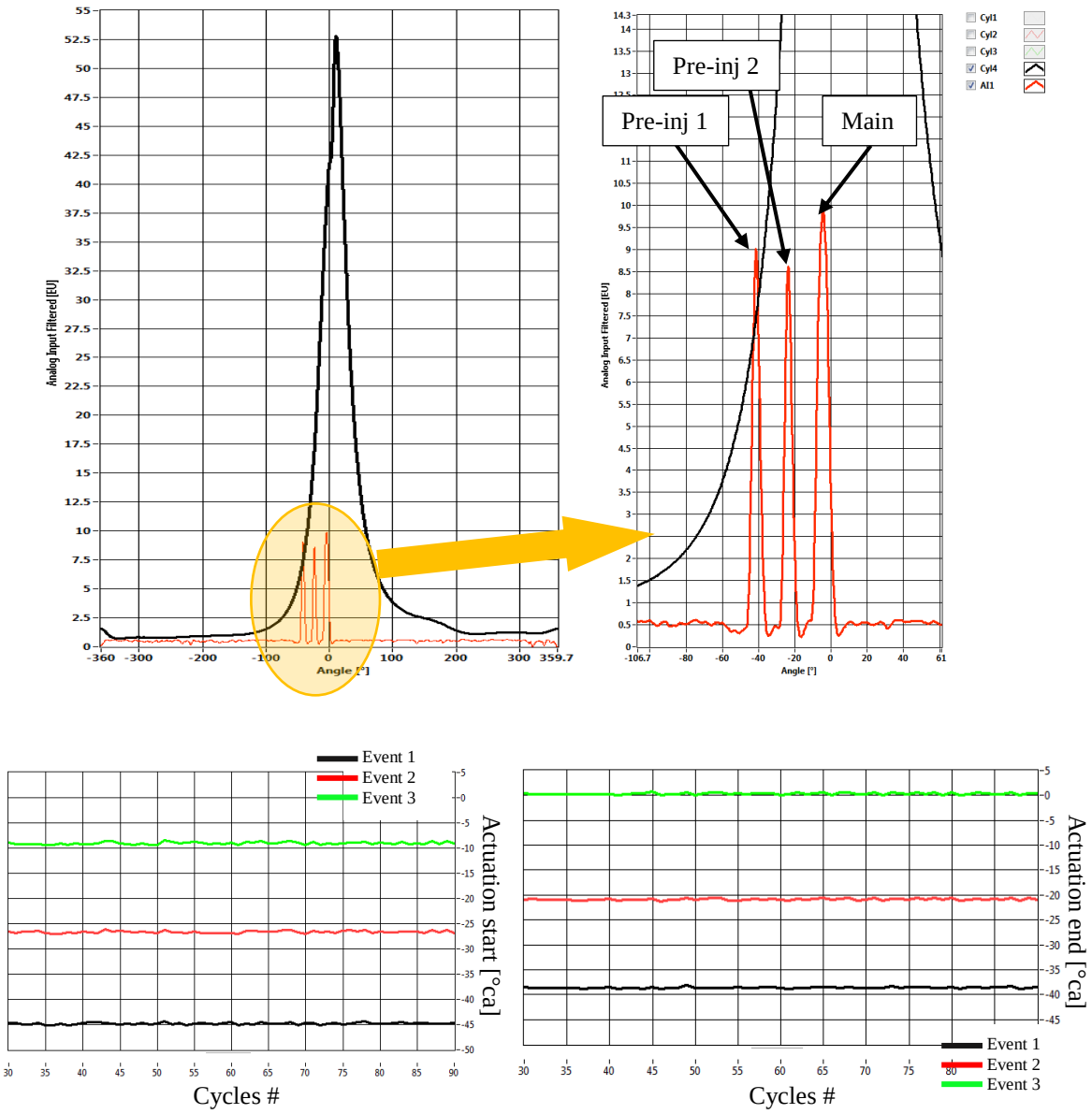


Figura 4-16: validazione algoritmo tracking su FIAT multijet

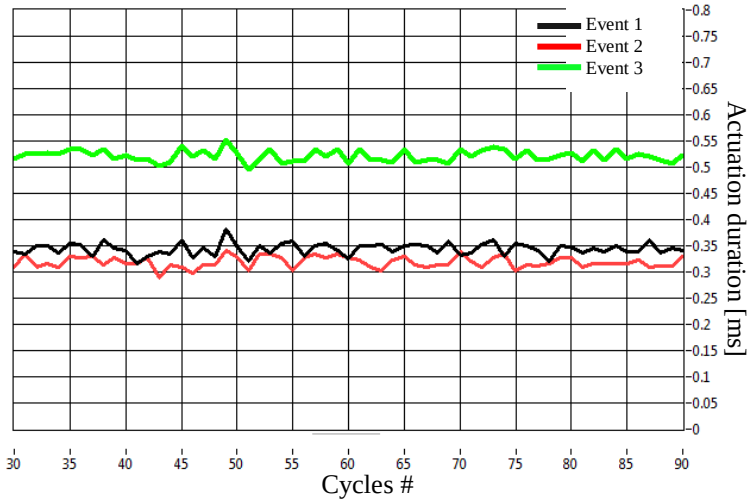


Figura 4-17: risultati relativi alla durata delle attuazioni

Inoltre sono stati fatti test anche su un motore Ducati 1198, per determinare il momento di scarica bobina modificando l'anticipo di accensione a gradino durante la prova. Si mostra il risultato sintetico del momento di scarica bobina calcolato in tempo reale e l'andamento del segnale grezzo acquisito durante la prova in alcuni punti di interesse, facendo notare come il valore calcolato dall'algoritmo real time sia fedele al segnale reale acquisito (si sottolinea che nei grafici seguenti i valori dei segnali sull'asse Y non sono quelli reali)

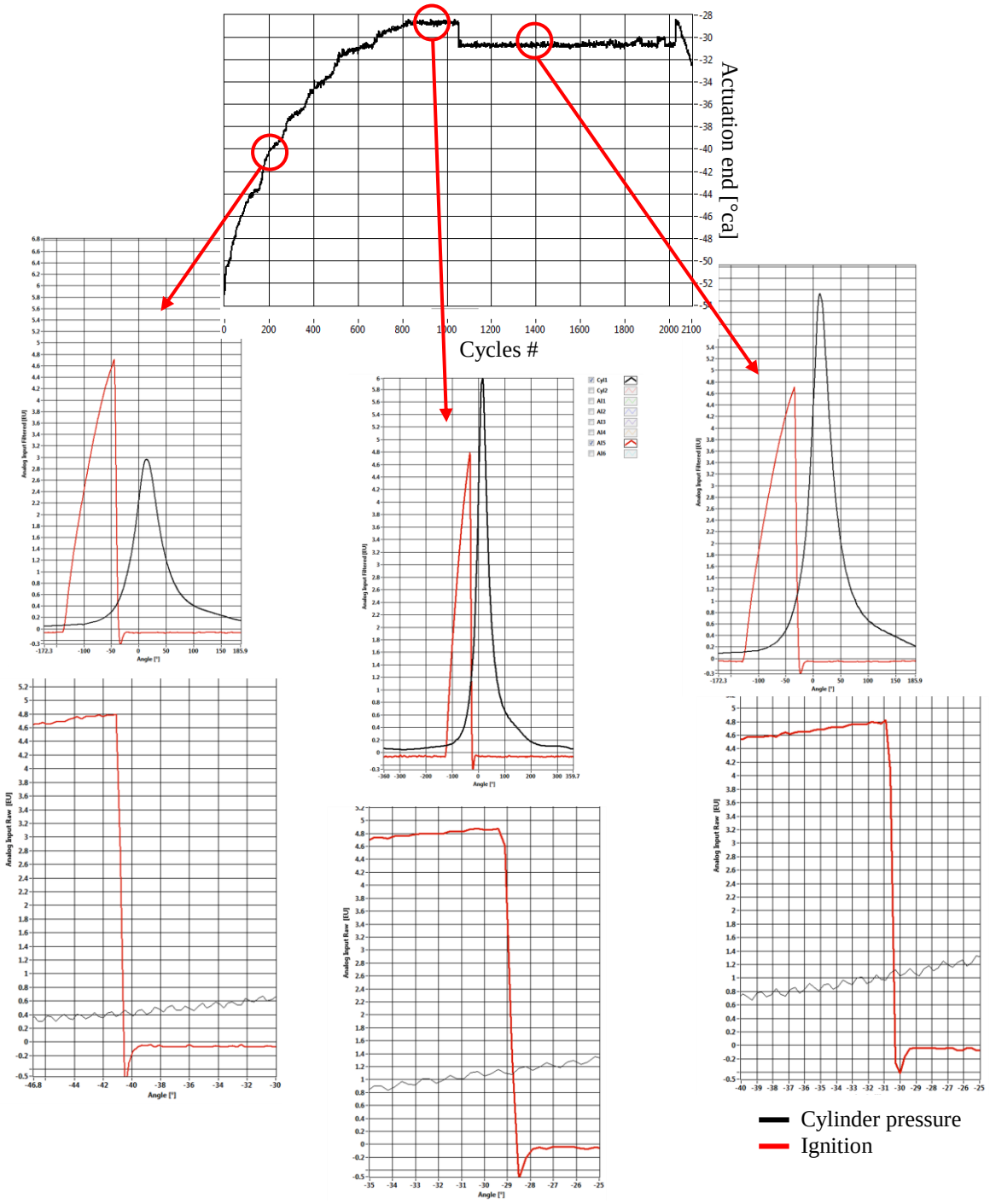


Figura 4-18: validazione algoritmo tracking su Ducati 1198

In entrambi i test effettuati, la validazione dell'algoritmo è stata fatta calcolando dall'acquisizione ad alta frequenza i valori di start, end e durata tramite analisi offline dei dati, e confrontando poi i risultati con quelli generati con il calcolo in tempo reale dal sistema, ottenendo un'ottima corrispondenza. Si ricorda che l'algoritmo è stato sviluppato utilizzando come input i segnali analogici acquisiti a 200kHz: questo implica che l'incertezza sul fronte di discesa acquisito è compresa nell'intorno dei 5µs, che porteranno

ad una incertezza angolare dipendente dal regime motore, tanto maggiore quando maggiore è la velocità di rotazione.

Una possibile evoluzione dell'algoritmo è quella di permettere l'utilizzo dei canali digitali per il tracciamento delle attuazioni, che porterebbe ad un netto incremento della risoluzione (la piattaforma Miracle-2 permette acquisizione di canali digitali a 10MHz). Per contro non sarebbe possibile la ricostruzione di alcuni tipi di segnale (carica bobina e peak and hold) utilizzando solo queste tipologie di porte.

5. COMUNICAZIONE SU BUS CAN

Come già sottolineato, una delle peculiarità del sistema è la modularità: uno dei principali obbiettivi è quello di permetterne l'integrazione all'interno di un sistema più complesso, quale può essere un sistema di controllo della sala prove, o un veicolo nel caso di installazione a bordo. Per questo è importante mettere a disposizione dell'utente la possibilità di comunicare i dati calcolati al mondo esterno mediante dei protocolli standard di scambio dati. Uno di questi è la comunicazione su bus CAN, tipica del mondo automotive, che permette al sistema di comunicare con altri hardware, che siano in ricezione o in scrittura, presenti all'interno della sala prove o a bordo veicolo.
La comunicazione CAN diventa una funzionalità indispensabile per comunicare con il sistema in modalità stand-alone, cioè quando il sistema è utilizzato senza essere connesso ad un PC host. Questa funzionalità diventa molto utile a bordo veicolo, dove l'utente può memorizzare i calcoli che il sistema effettua in tempo reale senza dover utilizzare un PC host, o addirittura sfruttare i risultati dell'analisi della combustione per agire in closed loop sul controllo del motore.

5.1 Il bus CAN

Il bus CAN (Controller Area Network) è uno standard seriale di tipo multicast tipico del mondo automotive in quanto molto robusto, poiché progettato per funzionare in ambienti con forte presenza di disturbi elettromagnetici. L'hardware di collegamento tra due nodi si basa su una coppia di fili, twistata per aumentare la resistenza a disturbi elettromagnetici. Il bus può contenere più nodi (ogni nodo è un dispositivo che può scrivere e leggere sulla rete); due sono le accortezze da seguire:

- il bus deve essere lineare e non a stella:

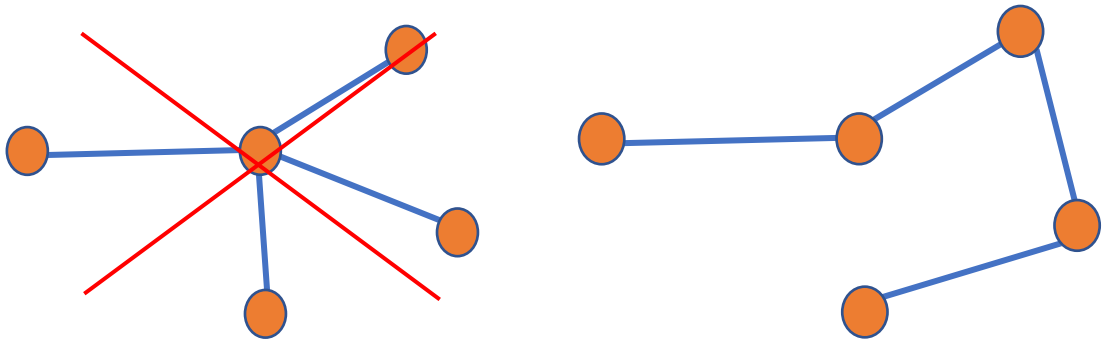


Figura 5-1: confronto tra bus a stella (errato) e bus lineare (corretto)

- i nodi estremi alla linea (e solo quelli) devono essere terminati con una resistenza di 120Ω

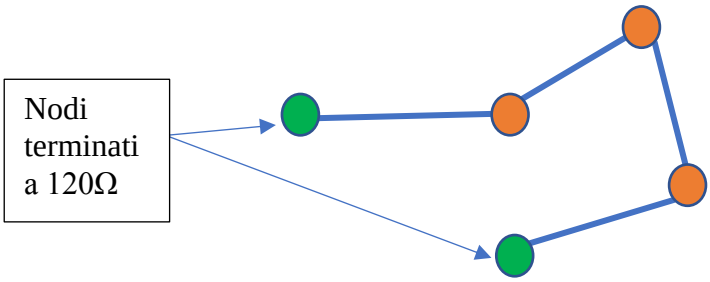


Figura 5-2: linea CAN terminata correttamente agli estremi

Seguendo queste accortezze si assicura una solidità della comunicazione anche in ambienti elettricamente rumorosi quali la sala prove motore, specialmente nei momenti in cui il motore si trova a carichi elevati e quindi freno e ventilatori sono in condizioni in cui generano forti disturbi.

Per quando riguarda la comunicazione dei pacchetti, driver appositi (presenti su ogni dispositivo che si aggancia alla rete) si occupano della gestione della spedizione e ricezione dei bit. Sarà cura dell'utente creare un database opportunamente formattato che identifica come devono essere interpretati i bit scambiati sul bus. Il database è composto da pacchetti, detti frame, caratterizzati da un indirizzo (ID), e contenenti un messaggio di 64byte. La formattazione di questi byte, cioè quali e quante grandezze sono contenute nel frame, e i guadagni di queste grandezze rappresentate su un intervallo in bit, sono definiti dall'utente al momento della creazione del database. Per la generazione dei database è possibile trovare software gratuiti che aiutano l'utente a creare e formattare opportunamente ogni frame (la stessa National Instruments mette a disposizione un tool per la creazione di database CAN chiamato database editor e contenuto nel pacchetto software XNET scaricabile gratuitamente dal sito National Instruments).

5.2 Gestione dei pacchetti indicating spediti sul bus CAN

Il codice di analisi della combustione già sviluppato nei precedenti lavori di dottorato [1], [2], era dotato di una parte di spedizione dei risultati su linea CAN. Questa parte di codice però non permetteva all'utente di selezionare quali dati si volevano spedire: il frame CAN generato era di formato fisso, contenente delle informazioni base relative alla combustione (IMEP, Pmax, MFB50, MAPO) senza possibilità di cambiarle, di assegnare loro un numero di bit differente (aumentarne cioè la risoluzione) e senza possibilità di cambiare gli indirizzi di spedizione dei frame. Il primo step è stato quindi quello di implementare un'interfaccia che permettesse all'utente di caricare un proprio database CAN e poi di associare ad ogni frame e ad ogni segnale una delle grandezze indicating calcolate in tempo reale, in modo tale da lasciare completa libertà all'utente di scegliere quali grandezze spedire ed in che modo. Questo, oltre a consentire una totale flessibilità rispetto ai dati inviati, permette di inserire i pacchetti spediti all'interno di un bus controllato da un database complesso (quale può essere quello di una sala prova che gestisce la comunicazione con la ECU motore, il banco ed il sistema indicating), senza il rischio di creare conflitti e quindi perdita di dati. Infatti, semplicemente caricando il database che gestisce la comunicazione dell'intera sala, ed associando ai campi desiderati i valori indicating che si vogliono pubblicare, avendo cura di mantenere liberi i frame utilizzati da altri dispositivi, si può procedere all'inserimento dei dati indicating nella propria rete CAN.

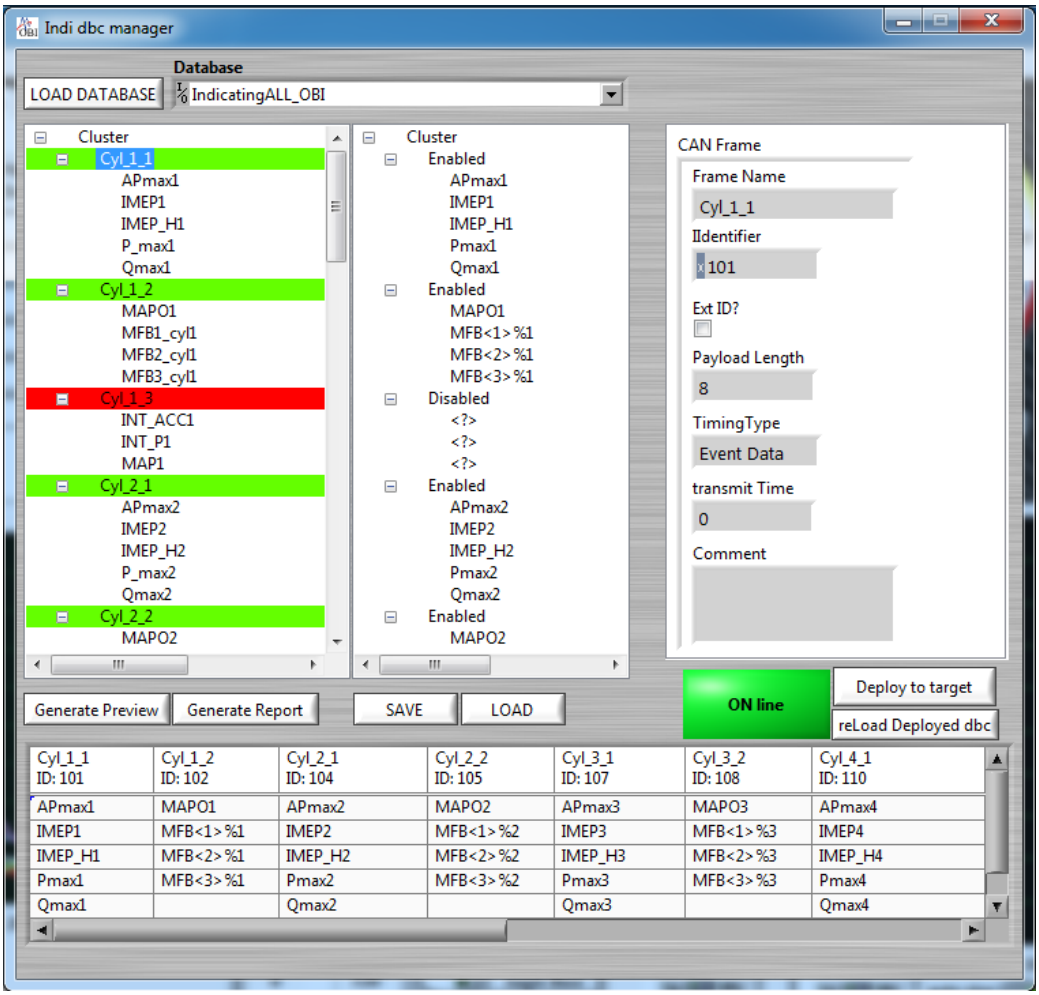


Figura 5-3: interfaccia di controllo dei dati indicating spediti via CAN

Tramite questa interfaccia host viene generata una mappa che associa il dato calcolato del singolo cilindro al rispettivo segnale CAN. Questa mappa viene scaricata all'interno del codice real time dove si ha cura di generare il frame da spedire, compilandolo opportunamente ad ogni ciclo con i dati aggiornati quindi spedendolo sul bus.

La pubblicazione di dati indicating su CAN in tempo reale, ciclo dopo ciclo, richiede delle risorse in termini di CPU real time tanto maggiori quanto maggiore è il numero di pacchetti da pubblicare (dipendente da quante grandezze si vogliono spedire per cilindro e dal numero di cilindri) e quanto maggiore è la frequenza a cui pubblicarli (cioè il regime motore dato che i messaggi sono spediti a frequenza di ciclo). Si è verificato sperimentalmente come la quantità di messaggi spediti influenzi la prestazione (il risultato seguente è relativo ad hardware Miracle-1: l'affermazione rimane valida su tutti gli hardware utilizzati, con occupazione totale di CPU differenti):

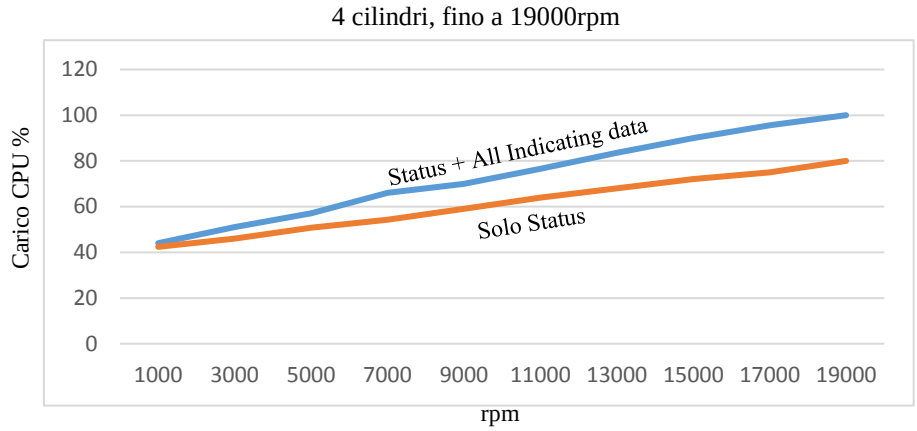


Figura 5-4:utilizzo CPU in funzione del regime motore e della pubblicazione di dati su CAN

Si noti come in condizioni limite (alti rpm, molti pacchetti spediti) si rischi la saturazione delle capacità e quindi la possibile perdita di dati. Sarà cura dell'utente monitorare le prestazioni del sistema in funzione delle impostazioni utilizzate e del punto motore analizzato, e diminuire quindi le richieste di dati in caso si raggiungano condizioni limite critiche (CPU media > 90%).

5.3 Invio dei pacchetti indicating ogni combustione

Il potenziale utilizzo dei dati relativi al calcolo indicating per il controllo in closed loop ha spinto lo sviluppo verso l'ottimizzazione delle tempistiche di pubblicazione dei dati. Infatti, per permettere un next cycle control su parametri di attuazione (un caso su tutti, la variazione dell'anticipo in caso di detonazione) è necessario avere il risultato dei calcoli del ciclo appena chiuso prima possibile. L'introduzione dell'hardware Miracle-2 ha consentito di lavorare in questa direzione, permettendo una spedizione dei risultati indicating dedicata al singolo cilindro, e quindi svincolando la spedizione dalla posizione angolare della fine del ciclo motore, legandola invece alla fine del ciclo del singolo cilindro.

Le nuove potenzialità hardware permettono di completare i calcoli indicating separatamente cilindro per cilindro, nel momento in cui finisce la fase di scarico del relativo cilindro, a condizione che lo sfasamento angolare tra due cilindri successivi sia superiore a 60°ca (altrimenti i calcoli di entrambi i cilindri vengono terminati contemporaneamente a fine scarico del secondo cilindro che va in combustione). Questo

significa che i dati indicating relativi ad ogni cilindro sono disponibili nel momento in cui il ciclo di quel cilindro termina, ed è quindi possibile pubblicarli in quell'esatto momento sul bus CAN. In questo modo le informazioni sulla combustione non vengono pubblicate tutte contemporaneamente, una volta per ciclo motore (riferito alla cava), ma vengono scritte sul bus CAN in momenti differenziati per ogni singolo cilindro, in funzione della sua posizione nel ciclo. Il beneficio di avere dati pubblicati singolarmente, e con il minimo ritardo rispetto alla fine del rispettivo ciclo, permette di avere maggiore tempo a disposizione prima della successiva combustione di quel cilindro, e quindi uno spazio di calcolo maggiore per implementare strategie di controllo retroazionato sui parametri della combustione già per il ciclo successivo. Nelle seguenti figure è riportata una rappresentazione grafica del vecchio metodo di pubblicazione e il nuovo metodo di scrittura dei pacchetti CAN: il metodo base ciclo è comunque mantenuto come alternativa, per ridurre il carico CPU del sistema nel caso in cui non sia necessaria una pubblicazione per combustione delle informazioni. Infatti, completare i calcoli in maniera dedicata al singolo cilindro richiede un maggiore utilizzo della CPU real time, inutile se non è richiesta una particolare velocità nella pubblicazione dei risultati.

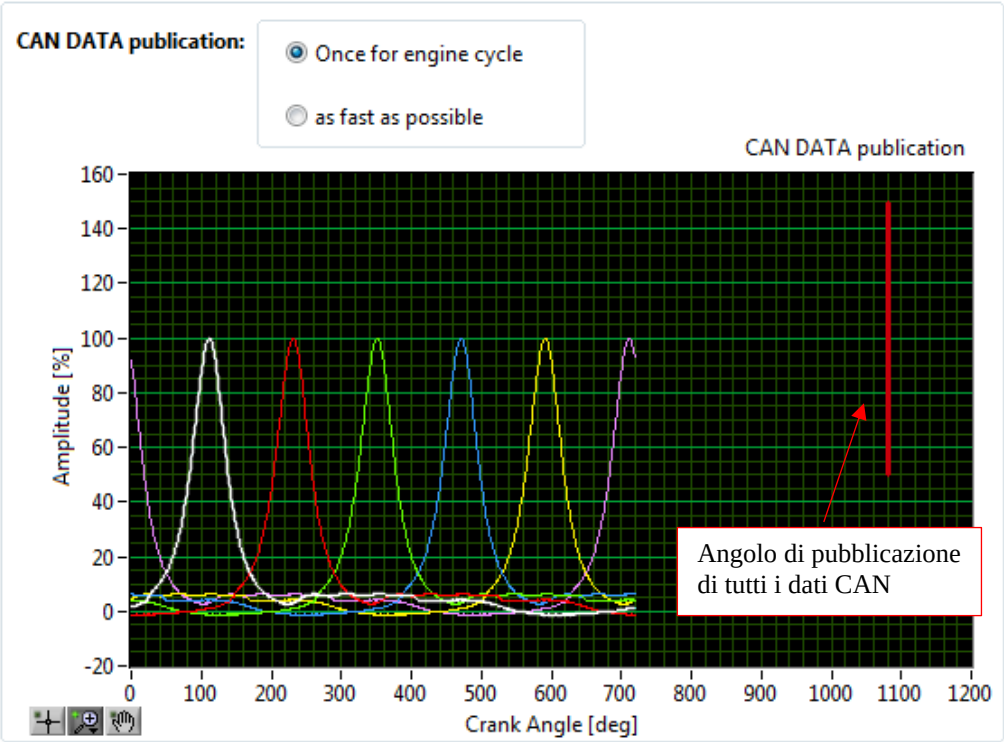


Figura 5-5: pubblicazione dati CAN una volta per ciclo motore

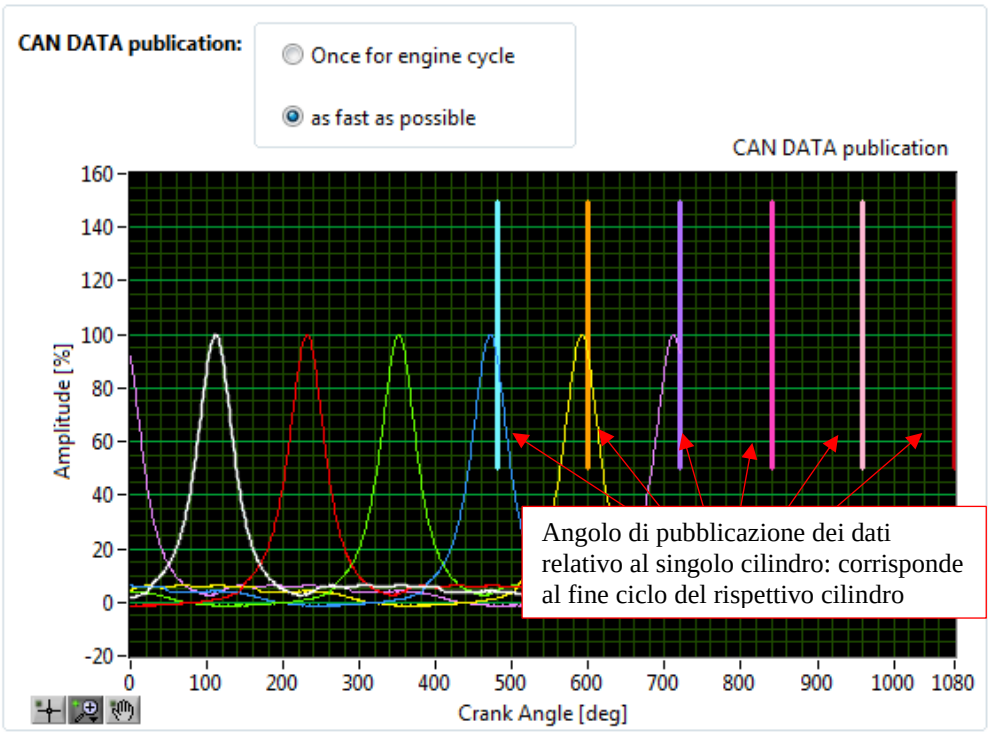


Figura 5-6: pubblicazione dati CAN combustione per combustione

5.4 Lettura di informazioni dal bus CAN

Proprio per la caratteristica di essere un bus a cui molti utenti possono accedere contemporaneamente in scrittura ed in lettura, si è pensato di introdurre la possibilità non solo di pubblicare dati, ma anche di leggere informazioni CAN dal mondo esterno, in maniera tale da rendere il sistema aperto a comandi provenienti dall'esterno, rendendolo anche un possibile collettore di dati da bus CAN.

L'hardware Miracle-2 mette a disposizione due linee CAN: si è quindi data possibilità all'utente di caricare un proprio database per ogni linea CAN (rendendo configurabile anche il baud rate di ogni linea) e di selezionare quali tra i pacchetti presenti nei database caricati leggere. Inoltre si è reso possibile configurare la frequenza di lettura dati da bus. Le informazioni vengono quindi acquisite a frequenza costante e salvate in formato TDMS (standard National Instruments, compatibile standard ASAM [4]) in base tempo. Parallelamente, le stesse informazioni vengono sincronizzate con i dati indicating base ciclo, e salvate, sempre in formato TDMS, insieme agli stessi parametri indicating. Questa funzionalità risulta essere molto utile per correlare i parametri riguardanti la combustione, calcolati dal sistema, con qualsiasi grandezza acquisita via CAN in sala prove proveniente

da hardware esterni (sistema di controllo banco, sistema di analisi fumi…), oppure con parametri di centralina, avendo al termine della registrazione tutte le grandezze di interesse all'interno dello stesso file, sincronizzate ciclo per ciclo. Bisogna precisare che la sincronizzazione non è fatta mediante timestamp dell'hardware che spedisce i pacchetti, ma viene associato al pacchetto un timestamp al momento della ricezione. Questo significa che i dati saranno sincronizzati a meno del ritardo della linea CAN e di un eventuale ritardo introdotto dall'hardware esterno che spedisce i dati. Questo problema può essere bypassato inserendo un ulteriore pacchetto contenente il timestamp del dato (se possibile lato writer): questo però richiede un'analisi offline per sincronizzare i dati, in quanto non è stata implementata in tempo reale la possibilità di gestire timestamp esterni provenienti da CAN per la sincronizzazione dei pacchetti. Solitamene la sincronizzazione fatta con timestamp di ricezione è sufficientemente precisa per le applicazioni standard di sala prove.

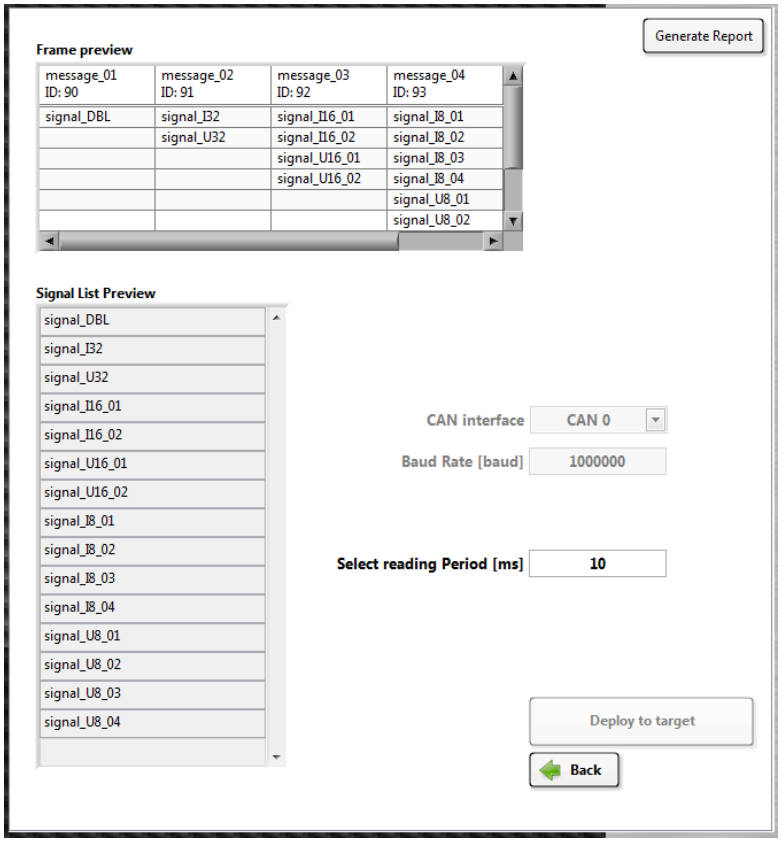


Figura 5-7: interfaccia di configurazione CAN in lettura

La possibilità di leggere pacchetti dalla linea CAN ha permesso, come già visto, di introdurre il recupero della componente media del segnale di pressione basandosi su dati di pressione collettore provenienti da hardware esterni (tipicamente da linea CAN ECU): inoltre questa funzionalità ha consentito di sviluppare il trigger della registrazione tramite

il monitoraggio di un parametro esterno presente su bus CAN, come vedremo nel capitolo seguente.

6. AUTOMAZIONE DELLA REGISTRAZIONE DEI DATI

Il primo obbiettivo di un sistema di analisi combustione è quello di raccogliere dati per poter analizzare il comportamento di un motore: non sempre durante una prova al banco l'operatore può curarsi di avviare manualmente una registrazione, ed inoltre, vista la mole di dati da controllare, anche se decide per una registrazione manuale, si potrebbe perdere un evento significativo che si verifica sul motore. Si aggiunga anche che il sistema può essere installato a bordo veicolo ed utilizzato in assenza di un PC host, vista la possibilità di salvare dati nella memoria interna di 32GB: diventa quindi di straordinaria importanza automatizzare l'inizio della registrazione dei dati.

Lo sviluppo delle funzionalità di trigger della registrazione è stato portato avanti parallelamente per la modalità con PC host e per la modalità stand-alone. Infatti risulta comodo remotare il sistema in entrambe le modalità, ed è conveniente farlo utilizzando gli stessi algoritmi, avendo cura che siano sviluppati in maniera idonea per essere utilizzati in real time: d'altra parte questo è un valore aggiunto al sistema in quanto garantisce (essendo l'algoritmo eseguito appunto in tempo reale) di avere un sistema molto reattivo ai trigger ricevuti, in maniera deterministica. Inoltre, ovviamente, mantenere un codice unico semplifica lo sviluppo e il debug.

6.1 Registrazione tramite segnale di trigger

Il salvataggio dei dati comandato da trigger si basa sull'acquisizione di un segnale esterno, che sia analogico, digitale o letto da bus CAN, ed il confronto di questo con una soglia impostata dall'utente.

La modalità di trigger tramite bus CAN pone le sue basi sulla parte di lettura CAN già esposta in precedenza. L'utente può caricare tramite interfaccia grafica un database dedicato al trigger, e scegliere quale tra i segnali presenti nel database monitorare. Il dato letto viene confrontato in tempo reale con una soglia (anche questa imposta dall'utente) e, al momento del superamento della soglia, viene avviata la registrazione.

La modalità di recording con trigger analogico si basa sull'utilizzo di uno dei canali analogici acquisiti per avviare la registrazione. L'utente può selezionare da interfaccia

host un canale qualsiasi tra i 24 analogici disponibili e stabilire una soglia in tensione: il sistema monitorerà tale canale ed inizierà la registrazione nel momento in cui la tensione misurata sarà superiore alla soglia imposta.
Per quello che riguarda il trigger digitale, anche in questo caso viene data all'utente la possibilità di selezionare quale canale monitorare, però la registrazione non sarà ovviamente avviata dall'attraversamento di una soglia, ma dal cambiamento di stato logico del canale digitale.
Tutte queste modalità funzionano indipendentemente in host o in stand alone mode, con una piccola differenza di durata ed interruzione della registrazione per la modalità host, in funzione della configurazione di registrazione scelta dall'utente; in particolare:

- Host mode:
  - Modalità *rec manuale*: il sistema inizia a registrare quando il trigger è sopra soglia, e continua la registrazione fino a quando il trigger rimane alto. La registrazione si ferma quando il trigger torna basso.
  - Modalità *rec cycle* o *rec time*: il sistema inizia a registrare quando il trigger supera la soglia. La registrazione continua per il numero di cicli o per il tempo impostato dall'utente e quindi si interrompe, indipendentemente dallo stato del trigger. Una seconda registrazione sarà possibile solo ad un nuovo attraversamento della soglia (se il trigger rimane costantemente alto dopo la fine registrazione, non si registrerà alcun nuovo file)
- Stand alone mode:
  - Unica modalità di recording: il sistema inizia la registrazione su memoria interna dei dati al superamento soglia. Un contatore di tempo si incrementa ed interrompe la registrazione quando raggiunge un valore in secondi impostato dall'utente. Se il trigger rimane costantemente alto dopo la fine registrazione, non si registrerà alcun nuovo file. Se si vede un nuovo attraversamento soglia mentre la registrazione è ancora attiva, il contatore tempo viene resettato.

In stand alone mode, per evitare di scrivere file eccessivamente grandi, il software genera un nuovo file ogni 3 minuti ininterrotti di generazione. Infine viene monitorato lo spazio disponibile per evitare di riempire la memoria interna (non si abilita la registrazione di nuovi file se non si ha spazio per salvarli).

Può accadere che il trigger di registrazione venga inviato da un sistema esterno nel momento in cui accade un determinato evento: risulta spesso di interesse per l'utilizzatore avere un'acquisizione che contenga anche dei dati antecedenti all'evento che ha innescato la registrazione. È stato quindi sviluppato un metodo basato su buffer di memoria mobile per creare uno storico che al momento del trigger consenta di salvare nel file registrato anche dati antecedenti all'evento. È stato necessario ottimizzare questo processo per evitare la perdita di dati ed il rallentamento dello streaming: si ricorda infatti che è necessario creare il pre-buffer per 24 canali, 16bit, a 200kHz. In modalità stand alone (cioè all'interno del processore real time) è stato implementato un pre-buffer di dimensione fissa che consente di avere due secondi di dati pre-evento. In modalità host invece, il buffer è stato reso configurabile a piacimento dall'utente, fino ad un massimo di 10 secondi.

6.2 Registrazione su evento

Data la disponibilità dei dati indicating calcolati in tempo reale, si è pensato di introdurre una modalità di registrazione stand-alone basata su evento indicating. Tramite un'interfaccia host, si permette all'utente di scegliere quali tra tutti i parametri indicating calcolati dal sistema si vogliono monitorare. Oltre a questo si permette la scelta del valore di soglia da monitorare e del verso (attivo per transizioni in salita o attivo in discesa).

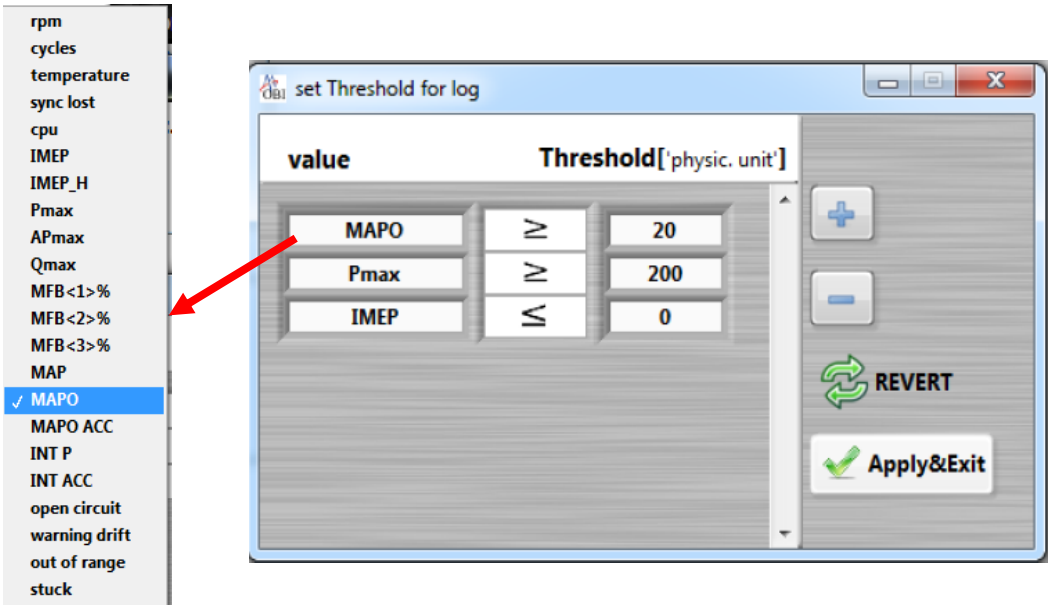


Figura 6-1:interfaccia di configurazione eventi e soglie da monitorare per la registrazione

Gli eventi sono monitorati tutti in OR logico. Al momento infatti non è stata implementata una modalità che permetta all'utente di usare in AND due eventi.
Anche in questo caso viene sfruttato il buffer mobile per poter registrare una porzione temporale antecedente rispetto al momento dell'evento che fa partire la registrazione: questo risulta indispensabile per questa tipologia di acquisizione. Si pensi ad esempio ad un evento di detonazione. Solo a fine ciclo avremo a disposizione un valore di MAPO che ci segnala il problema, quindi se non fosse possibile registrare uno storico, non saremmo in grado di vedere come è iniziata a svilupparsi la combustione anomala che ha portato a quel valore di MAPO, rendendo inutile la registrazione stessa.
Per facilitare il lavoro dell'utente nel riconoscimento dei file che contengono un certo evento registrato di interesse, viene generato un nome file che in automatico contiene al suo interno il nome della grandezza che ha scatenato quella registrazione e il valore assunto al superamento della soglia.
Le grandezze indicating sono disponibili solo dopo che il sistema si è correttamente sincronizzato e fasato con il riferimento angolare. In avviamento motore quindi non sarebbe possibile utilizzare questi eventi per triggerare la registrazione. Vista l'importanza degli avviamenti per l'impatto che hanno sugli inquinanti, specialmente per quanto riguarda il ciclo di omologazione, e il sempre maggiore interesse dei calibratori riguardo questa fase, è stata sviluppata una modalità di registrazione Start&Stop attivabile dall'utente, che permette proprio la registrazione dei dati in queste fasi anche se il sistema non è fasato e sincronizzato.

6.3 Recupero dei file da memoria interna

Il salvataggio dei dati in modalità stand alone all'interno della memoria del Miracle-2, pone il problema del recupero dei dati da parte dell'utente: è stata sviluppata un'interfaccia host che permette la visualizzazione dei file presenti sulla centralina e il loro trasferimento sul PC host tramite comunicazione WebDav. Inoltre, tramite questa interfaccia è possibile procedere alla cancellazione di tali file per liberare memoria per registrazioni future.

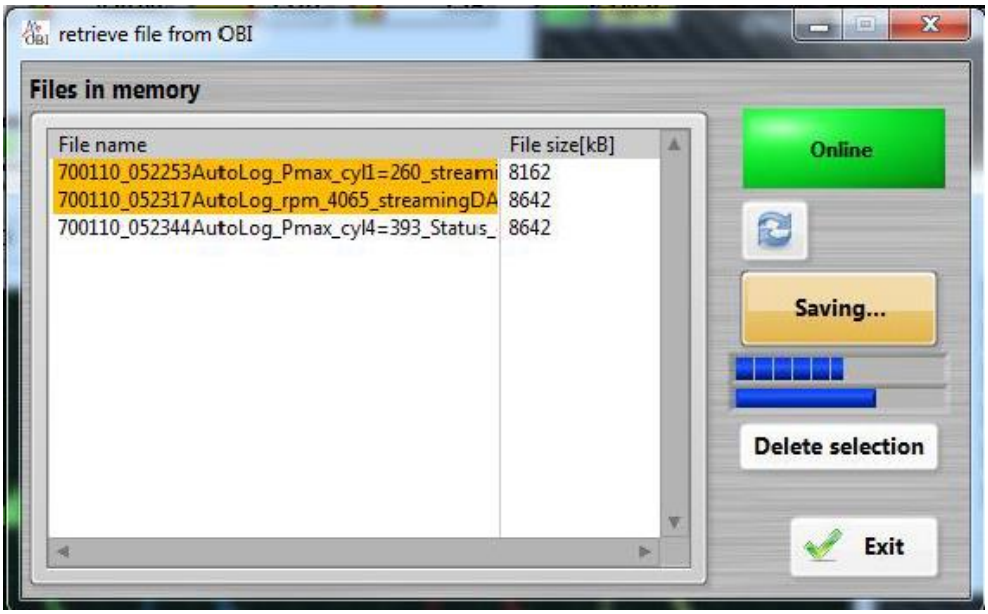


Figura 6-2: interfaccia per il recupero dei file da memoria interna

7. COMUNICAZIONE XCP

Il protocollo XCP è un protocollo di comunicazione standard ASAM [3] che permette di scambiare comandi tra due dispositivi, uno master ed uno slave. Data la forte diffusione all'interno del mondo automotive di software basati su protocollo XCP per la gestione della calibrazione motore, che permettono l'accesso e la modifica delle mappe scaricate in centralina, e che svolgono inoltre anche la funzione di collettore di dati (Etas INCA® in primis) si è reso necessario per rendere competitivo il sistema di analisi combustione sviluppato, introdurre la possibilità di essere utilizzato come device slave XCP.

7.1 Basi del protocollo XCP

Il protocollo XCP è un protocollo di comunicazione che può essere implementato su CAN, su ethernet via TCP/IP o via UDP/IP oppure può essere implementato su USB. Nel caso in questione si è scelto di implementare questo protocollo su ethernet, via UDP/IP. La comunicazione si basa su uno scambio di pacchetti ben definiti tra un master ed uno slave:

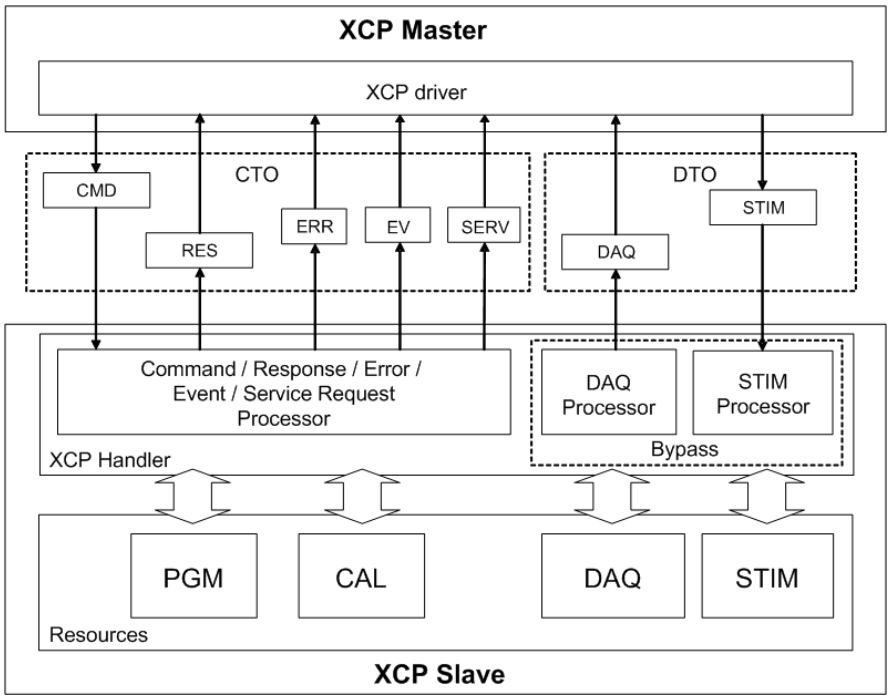


Figura 7-1: quadro generale comunicazione XCP

Questi pacchetti possono essere dei pacchetti CTO (command transfer object) che permettono lo scambio di comandi dal master allo slave (CMD) e delle eventuali risposte (RES, ERR,…) oppure possono essere dei pacchetti DTO (data transfer object) che consentono lo scambio di dati acquisiti (DAQ). La formattazione del singolo pacchetto è ben definita:

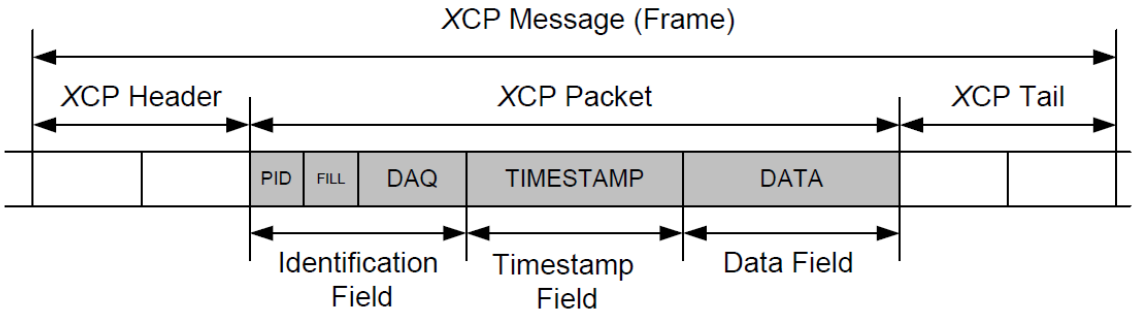


Figura 7-2: pacchetto XCP

Si noti come sia possibile introdurre un timestamp associato ai dati inviati: questo vedremo in seguito consentirà particolari manovre di sincronizzazione dei dati di combustione con dati provenienti da altri sistemi.

I vari comandi e risposte scambiati tra master e slave seguono una ben definita documentazione che descrive i possibili comandi che un master può richiedere e come devono essere formattate le risposte dello slave. Si evita di riportare la lista di comandi implementata per rendere il software un device XPC slave, ma si rimanda alla documentazione ASAM, a cui Unibo ha accesso [3].

Per poter scambiare informazioni tra master e slave è inoltre necessario creare una mappa della memoria dello slave e una mappa delle informazioni necessarie per accedere allo slave (indirizzi, porte, comandi implementati). Queste informazioni vengono contenute in un particolare file che il sistema master si aspetta di ricevere prima di aprire la comunicazione con lo slave: questo file si chiama A2L file, ed anche in questo caso la formattazione è definita da uno standard ASAM, a cui si fa riferimento [4]. In seguito verrà approfondita la creazione di questo file.

7.2 Specifiche del protocollo XCP implementato

Come detto, la comunicazione è stata sviluppata via UDP on ethernet, e sono stati introdotti 4 differenti raster di misura: 10ms, 100ms, 500ms, segment_synchronous (sincrono con la posizione di ogni TDC). Un raster di misura è un "acquisitore" di dati, a cui è possibile associare la misura da compiere, che nel caso del sistema indicating è uno dei valori indicating calcolati. Avendo quindi un raster di misura sincrono con il TDC di ogni cilindro, è possibile pubblicare i dati relativi all'analisi sintetica della combustione in maniera indipendente per ogni cilindro, ed associarli alla effettiva posizione temporale del TDC stesso. È possibile fare questo grazie all'utilizzo del timestamp XCP. Infatti come già detto riguardo alla comunicazione CAN dei dati indicating, il dato calcolato è disponibile solo a fine fase di espansione di ogni cilindro: mantenendo però in memoria il momento temporale assoluto in cui viene raggiunto il TDC effettivo del singolo cilindro, è possibile assegnare tale valore ai dati in spedizione, e, di conseguenza, sarà il device master a riposizionarli nel momento temporale opportuno in fase di salvataggio e successiva visualizzazione.

Data la flessibilità del sistema, che permette all'utente di scegliere come modificare il numero di canali acquisiti, ed il numero di cilindri, sarà necessario rendere dinamica la mappatura della memoria del dispositivo per poter permettere al device master di accedere correttamente ai dati qualsiasi sia la configurazione scelta. Questo impone la creazione di un file A2L dinamica e flessibile in relazione alla configurazione scaricata nel dispositivo. Questo è stato fatto sviluppando un codice host che consente la lettura della configurazione attuale e la generazione del file A2L rispettando il protocollo imposto dallo standard ASAM.

La comunicazione XCP consente anche la calibrazione del dispositivo, che nel caso di un sistema di analisi combustione corrisponde alla configurazione dei parametri. È stata quindi implementata all'interno del file A2L anche una mappatura automatica della memoria relativa alla configurazione e non solo alle misure, che consente all'utente di modificare la maggior parte dei parametri di impostazione del sistema senza utilizzare l'interfaccia host dedicata, ma accedendo tramite il master XCP.

Unico limite è l'impossibilità, al momento, di trasmettere lo streaming di dati ad alta frequenza tramite questo protocollo: questo impone l'utilizzo dell'interfaccia host

dedicata nel momento in cui l'utente vuole visualizzare in tempo reale lo streaming dei dati.

7.3 Perché è importante la comunicazione XCP

La comunicazione XCP diventa una caratteristica fondamentale per un sistema modulare che si deve integrare con un ambiente di sperimentazione motore, in quanto lo standard attuale per svolgere le prove di calibrazione si appoggia a collettori di dati basati su questo protocollo. Come già detto, il software più diffuso e maggiormente utilizzato per la calibrazione del motore e l'acquisizione di dati, in sala prove ed a bordo vettura, è Etas INCA (il 90% dei calibratori in Europa lo utilizza): questo software si basa appunto sul protocollo XCP, e consente di gestire tramite questo protocollo i dati provenienti da tutti i device slave collegati durante una prova, oltre a permettere il controllo e la calibrazione della centralina motore durante la prova. In questo modo il calibratore si troverà ad utilizzare un'unica interfaccia per gestire tutti gli strumenti di sala, avendo al termine della prova un unico file contenente tutte le informazioni di tutti i device connessi, tutte sincronizzate tra di essi.

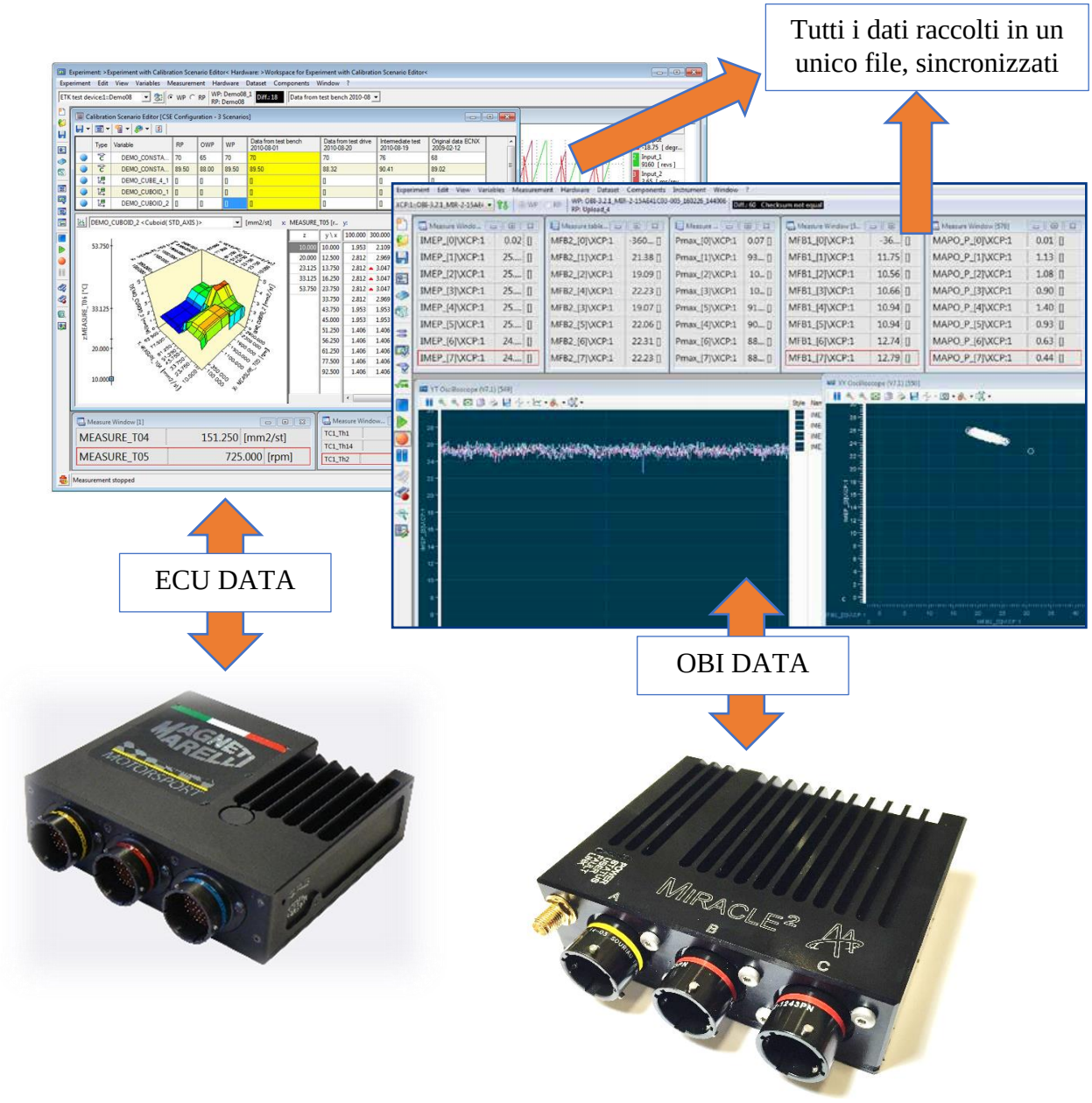


Figura 7-3: esempio di utilizzo in parallelo di più device slave mediante l'utilizzo di INCA

7.4 Compensazione di ritardi angolari costanti

Come visto, una caratteristica interessante del protocollo XCP è la possibilità di introdurre un timestamp associato ai dati che si stanno pubblicando: questo permette al collettore master di sincronizzare i dati tra diverse fonti (ad esempio ECU e sistema di analisi combustione). Il master compensa eventuali ritardi temporali tra i vari device basandosi sul timestamp che riceve; solitamente per questo tipo di misure (sincrone con la posizione

del TDC) il ritardo è costante su base angolo e non su base tempo. Questo può portare a ritardi differenziati tra dati combustione e dati ECU al variare del regime, e quindi ad una difficoltà da parte del calibratore per ri-sincronizzare i dati. Per ovviare a questo problema, si è deciso di correggere opportunamente il valore del timestamp associato ai dati spediti, in modo da muoverlo nel tempo in maniera angolarmente costante (imponendo quindi un ritardo temporale variabile in funzione degli rpm): questa funzionalità è stata implementata permettendo all'utente di scegliere le modalità di recupero, in termini di cicli di ritardo, combustioni di ritardo o angolo di ritardo, così da poter meglio adattare il ritardo da introdurre rispetto al caso specifico in considerazione.

8. HARDWARE IN THE LOOP

Lo sviluppo di nuovi algoritmi e il debug di ogni funzionalità inserita richiede la possibilità di testare il sistema in tutte le condizioni di funzionamento di fronte alle quali si può trovare. Data la flessibilità del sistema stesso, diventa impossibile avere a disposizione tutte le tipologie di motori che possono essere analizzate, oltre al fatto che prove su motori reali sono particolarmente costose. Si è quindi sviluppato un sistema di test in hardware in the loop che permette la generazione dei segnali tipici con cui un sistema di analisi combustione si trova ad interagire. Questo ha permesso di abbassare drasticamente i tempi di sviluppo, in quanto ogni modifica o aggiunta software può essere subito testata.

8.1 Hardware utilizzato

La parte hardware dell'HIL è costituita da un case contenente due schede National Instruments, una per l'I/O di segnali analogici (NI-PCI 6733) e l'altra per la comunicazione su bus CAN (NI-PCI 8512), installate su un bus PCI che può essere connesso al bus del PC-host tramite express card. Possono essere generati fino ad 8 segnali analogici di range ±10V a frequenza massima di 750KS/s, che vengono utilizzati per simulare il quadro segnali ed i segnali di pressione cilindro di uno specifico motore necessario per i test da compiere. La scheda CAN permette di effettuare il controllo sulla corretta spedizione dei dati CAN in uscita pubblicati dal sistema e parallelamente spedire pacchetti verso la centralina in prova.

8.2 Simulazione

Il sistema HIL è sviluppato anch'esso tramite Labview: si basa sulla generazione a frequenza variabile (selezionabile dall'utente) di un quadro segnali definito in tensione, caricato tramite file di testo. Il file in questione può contenere qualsiasi tipo di segnale si voglia rigenerare, da un'onda sinusoidale ad un segnale di pressione reale acquisito al

banco prova. Questo permette di creare segnali a piacimento e generarli in maniera sincrona insieme a segnali reali acquisiti al banco. In questo modo è possibile ad esempio assegnare ad un segnale di pressione acquisito utilizzando una ruota fonica 60-2, un differente tipo di riferimento angolare, che sia un encoder o una ruota fonica con un numero di riferimenti differente.

Oltre a questo, la possibilità di generare una campana di pressione nota e ripetibile all'infinito, permette di validare i calcoli indicating svolti in tempo reale dal sistema su quella campana, semplicemente avendo analizzato una singola volta offline i valori indicating che si ottengono da quel particolare segnale. Questo abbatte i tempi di analisi e validazione: infatti per validare una prova effettuata al banco, è necessario analizzare offline ogni volta la singola prova, in quanto la dispersione ciclica non consente di utilizzare i risultati di uno stesso punto motore per tutte le prove.

Figura 8-1: interfaccia di controllo della simulazione in HIL

Il software consente la riproduzione del file a punto fisso (si pensi alle problematiche che nascerebbero a mantenere un motore reale a regime massimo per ore in una prova al banco) oppure consente la riproduzione di un giro di pista, variando la frequenza di riproduzione del file in funzione di una traccia RPM definita.

Il sistema HIL permette inoltre di acquisire i dati pubblicati dal sistema via CAN e di assegnare loro una posizione temporale: questo ha consentito il debug e la validazione

della pubblicazione dei risultati indicating combustione per combustione, oltre al test in remoto senza interfaccia grafica e alla validazione dei trigger di registrazione.

Tipicamente i file vengono generati con una risoluzione angolare di 0.2 gradi (sufficiente per la validazione del funzionamento del sistema). Nel caso sia invece necessaria una risoluzione più spinta per validare un algoritmo (e.g. validazione tracking attuazioni) è possibile spingersi anche a risoluzioni più elevate.

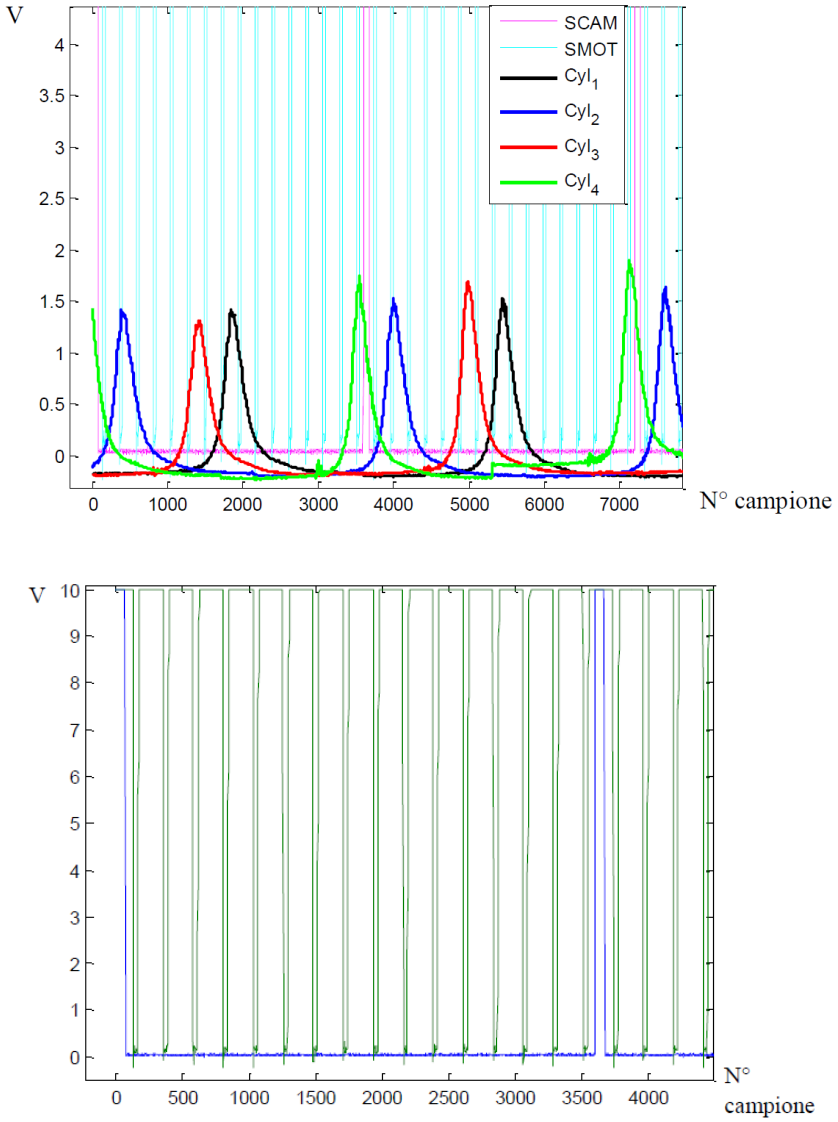


Figura 8-2: esempi di quadri segnale generabili in HIL

L'utilizzo di questo strumento ha consentito di testare il sistema in tutte le possibili condizioni di funzionamento, velocizzando notevolmente il tempo speso per il debug e abbattendo i costi di test e di sviluppo.

Un limite che obbliga a svolgere dei test di validazione finale al banco è la difficoltà nel riprodurre l'oscillazione ciclica della velocità motore e l'effetto di eventuali rumori

elettrici sui segnali. In realtà sarebbe possibile migliorare molto il test in hardware in the loop evolvendo il sistema in maniera da riprodurre fedelmente l'oscillazione di RPM nel ciclo basandosi su acquisizioni reali e sull'utilizzo di output digitali ad alta frequenza (10MHz) al posto di output analogico per la generazione del segnale di posizione angolare. Questo richiederebbe una fase di sincronizzazione aggiuntiva tra i segnali analogici e digitali generati per poter riprodurre correttamente il quadro segnali nel ciclo motore.

9. PROSPETTIVE DI EVOLUZIONE DEL SISTEMA

Gli sviluppi introdotti hanno portato ad un utilizzo di tutte le risorse disponibili a livello FPGA. Già allo stato attuale si è costretti a gestire dinamicamente l'utilizzo di queste risorse abilitando certe funzionalità a discapito di altre su richiesta dell'utente, non potendone garantirne il funzionamento contemporaneo. Non essendo ancora stata annunciata nessuna evoluzione hardware da National Instruments riguardo a nuove Single Board RIO, la parte di possibili sviluppi del codice FPGA si limita all'ottimizzazione di ciò che già esiste per poter introdurre nuove parti di codice, cercando di farle coesistere dinamicamente (cioè abilitando e disabilitando le funzioni non utilizzate) con il resto degli algoritmi. A livello real-time la situazione è migliore ma non ottimale: anche a questo livello le risorse occupate sono alte, ma il codice può essere ottimizzato e migliorato, con la prospettiva di poter introdurre non solo nuovi algoritmi definiti dal programmatore, ma anche calcoli personalizzabili da parte dell'utilizzatore, consentendo l'esecuzione di codici di analisi generati dall'utente secondo le sue esigenze. Inoltre l'ambiente Linux permetterà, interagendo direttamente da shell, l'introduzione di funzionalità aggiuntive in parallelo al codice Labview stesso.

La parte che ancora è maggiormente aperta a nuovi sviluppi è quella sviluppata su PC host. Questa sezione non potrà svolgere calcoli real time, ma potrà dedicarsi a calcoli on-line. La differenza è che i risultati non potranno essere utilizzati per un controllo in tempo reale ciclo per ciclo, ma l'utente avrà comunque a disposizione delle analisi avanzate mentre sta ancora svolgendo la prova, abbattendo così i tempi dell'analisi di post processing dei dati offline. Anche in questo caso l'obbiettivo finale deve essere quello di permettere all'utente di decidere quali calcoli vuole fare, lasciando la possibilità di integrare parti di codice, senza limiti stringenti di linguaggio di programmazione.

Un'altra parte fondamentale dello sviluppo del sistema dovrà essere quella dell'integrazione con il mondo esterno e dell'automatizzazione, continuando il percorso intrapreso di apertura verso i protocolli standard di comunicazione. In particolare deve avere un ruolo centrale l'interfacciamento con software esterni utilizzati in sala prove che svolgono la funzione di collettore dati, insieme al potenziamento delle funzioni di trigger di registrazione e condivisione dei dati.

Per quanto riguarda le potenzialità dell'hardware Miracle-2, non è ancora stata sfruttata la possibilità di generare segnali tramite DAC e DO: questo potrebbe aprire alla

generazione di controlli basati su eventi legati alla combustione, spaziando da allarmi ad azionamenti veri e propri. Lo sviluppo di questa parte non è però banale, in quanto richiede un lavoro a livello FPGA che, come detto, risulta essere quasi satura. Allo stesso tempo anche la piattaforma inerziale a 9 assi presente nel dispositivo non è sfruttata appieno: al momento viene solo acquisita e i dati vengono salvati senza nessuna analisi. Questo lascia spazio a possibili evoluzioni per l'acquisizione di dati di posizione e accelerazione veicolo, che potranno essere correlati ai dati dell'analisi combustione, pensando anche alla possibile introduzione di un segnale GPS tramite porta seriale ed all'integrazione di tutte queste informazioni.

Infine, nell'ottica di migliorare la facilità di utilizzo, è ancora tanto il lavoro da implementare, sia in termini di flessibilità dell'interfaccia, sia in termini di grafica volta a rendere più intuitivo il lavoro del calibratore, sia per quanto riguarda la configurazione che per quanto riguarda l'interpretazione rapida ed inequivocabile dei risultati calcolati.

Si lascia quindi aperta una strada ben avviata ma ancora lunga da proseguire per eventuali prossimi dottorati di ricerca, non solo in campo strettamente meccanico di analisi della combustione.

10. CONCLUSIONI

Il lavoro svolto ha portato alla realizzazione di un sistema di analisi combustione solido, affidabile e competitivo rispetto agli altri sistemi indicating presenti sul mercato. In particolare, il lavoro di debug e test al banco, unito alla semplificazione ed all'aumento di flessibilità e prestazioni del sistema, ha permesso di arrivare ad un prodotto in grado di rispondere alle esigenze tipiche di test e calibrazione su motori a combustione interna.
Le principali caratteristiche che emergono lato utente dopo il percorso di sviluppo seguito sono:

- Incremento del numero di canali acquisiti ad alta frequenza e del numero di segnali di pressione cilindro su cui compiere l'analisi della combustione in tempo reale.
- Possibilità di accedere a nuovi algoritmi di calcolo che migliorano la qualità dei dati acquisiti e che ampliano le possibili tipologie di sensori utilizzabili per i test
- Possibilità di monitorare eventi di controllo esterni che influenzano l'andamento della combustione correlandoli con l'effetto che hanno sulla combustione stessa
- Implementazione di trigger di registrazione evoluti, con possibilità di utilizzare il sistema senza la necessità dell'interazione dell'operatore per il salvataggio dei dati
- Accesso ai dati in tempo reale tramite protocolli di comunicazione standard (CAN, XCP), con la possibilità di utilizzare i risultati per un controllo in closed loop della combustione, riuscendo ad avere a disposizione i dati con sufficiente anticipo per poter sviluppare algoritmi che agiscono di ciclo in ciclo.
- Possibilità di utilizzare il sistema come collettore di dati CAN, sincronizzati con i risultati di analisi della combustione

Il punto di forza principale del sistema sviluppato è proprio la possibilità di interconnessione ed interfacciamento con il mondo esterno: infatti il sempre maggior numero di strumenti presenti in sala prove rende necessario poter minimizzare l'intervento dell'utente su ciascuno di questi, andando nella direzione dell'automatizzazione della prova e della calibrazione stessa, altrimenti si corre il rischio di fornire uno strumento che non verrà mai utilizzato perché il beneficio introdotto non è conveniente. Inoltre questa caratteristica, unita alle ridotte dimensioni dell'hardware, rende il sistema molto competitivo per l'analisi a bordo veicolo, attività che sta

diventando sempre più diffusa proprio a seguito degli ultimi sviluppi sull'omologazione dei veicoli.

L'intero sviluppo del nuovo codice Labview e la parte di ottimizzazione di quello già esistente, sono stati fatti avendo sempre cura di rendere il tutto flessibile e facilmente adattabile a successive modifiche ed implementazioni, in modo da semplificare la programmazione, anche in parallelo da parte di più sviluppatori, senza il rischio di compromettere il funzionamento generale. Infatti il codice preso in eredità all'inizio del percorso non era facilmente espandibile e rendeva critica ogni introduzione di nuovi algoritmi.

Il contributo dato ha quindi spaziato dall'ottimizzazione del software, alla definizione ed implementazione di nuovi algoritmi, al test e validazione in hardware in the loop e al banco prova. Il percorso ha portato a confrontarsi con le problematiche tipiche dell'acquisizione dei segnali e dell'utilizzo dei sensori di misura presenti in una sala prove, nonché con metodologie di test, debug e validazione, sia a livello di scrittura software sia a livello di algoritmi di analisi della combustione e validazione dei risultati indicating calcolati dal sistema, spaziando all'interno di tutto lo spettro di problematiche che nascono da entrambi i fronti.

Questo sistema di analisi della combustione è attualmente in uso nelle sale prova del Dipartimento di Ingegneria Industriale dell'Università di Bologna (laboratorio di via Terracini, Bologna, e laboratorio hangar di via Seganti, Forlì) ed è utilizzato a supporto delle attività di ricerca e didattica svolte al banco.

A. APPENDICE A: schema del codice LabVIEW

Si riporta in appendice la struttura generale del codice Labview implementato, con l'obbiettivo di fornire un quadro generale, evidenziando sia le parti alle quali sono state apportate modifiche sia nuove parti di codice introdotte.

Struttura del codice ad inizio lavoro

Il codice, come già sottolineato in precedenza, è sviluppato su tre livelli: FPGA, Real-Time ed host PC. Ogni livello si occupa della gestione di più task in parallelo. Il codice ereditato all'inizio del lavoro svolto, si sviluppava secondo i seguenti schemi a blocchi, riportati per i tre livelli di implementazione.

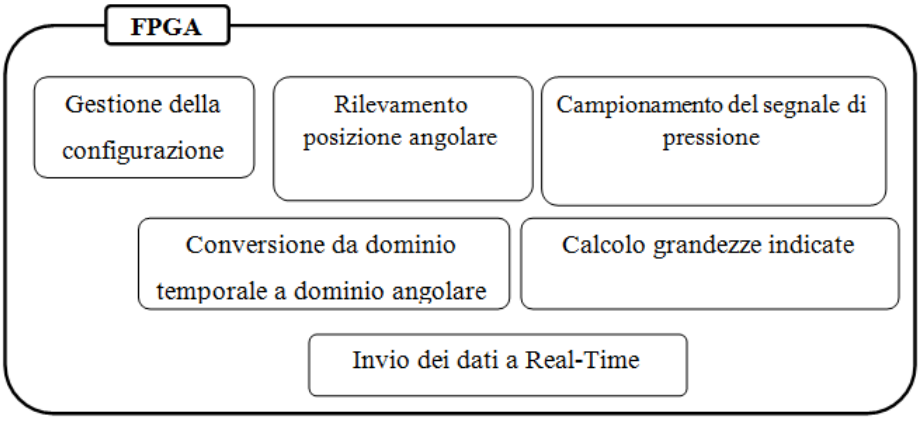


Figura A-1: schema a blocchi del codice FPGA

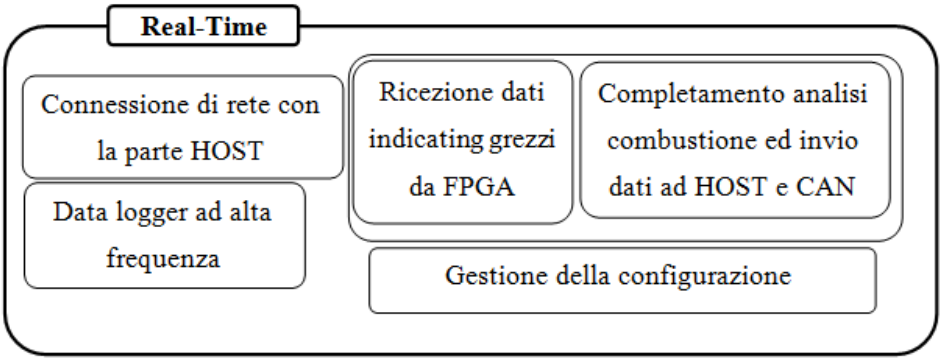


Figura A-2: schema a blocchi del codice Real-Time

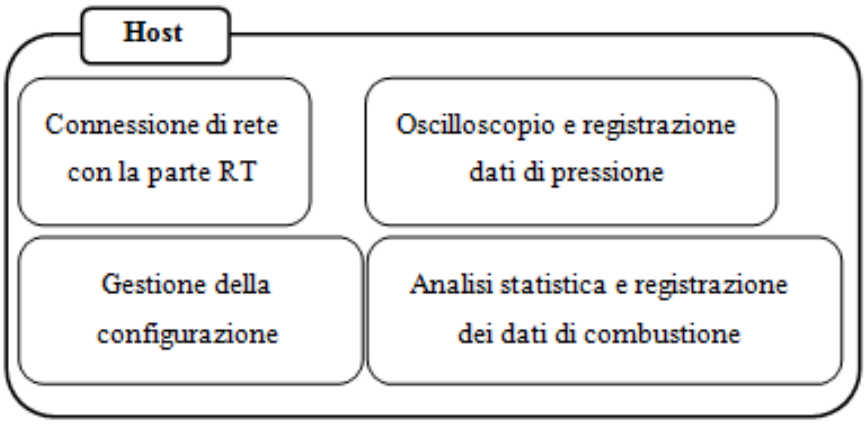
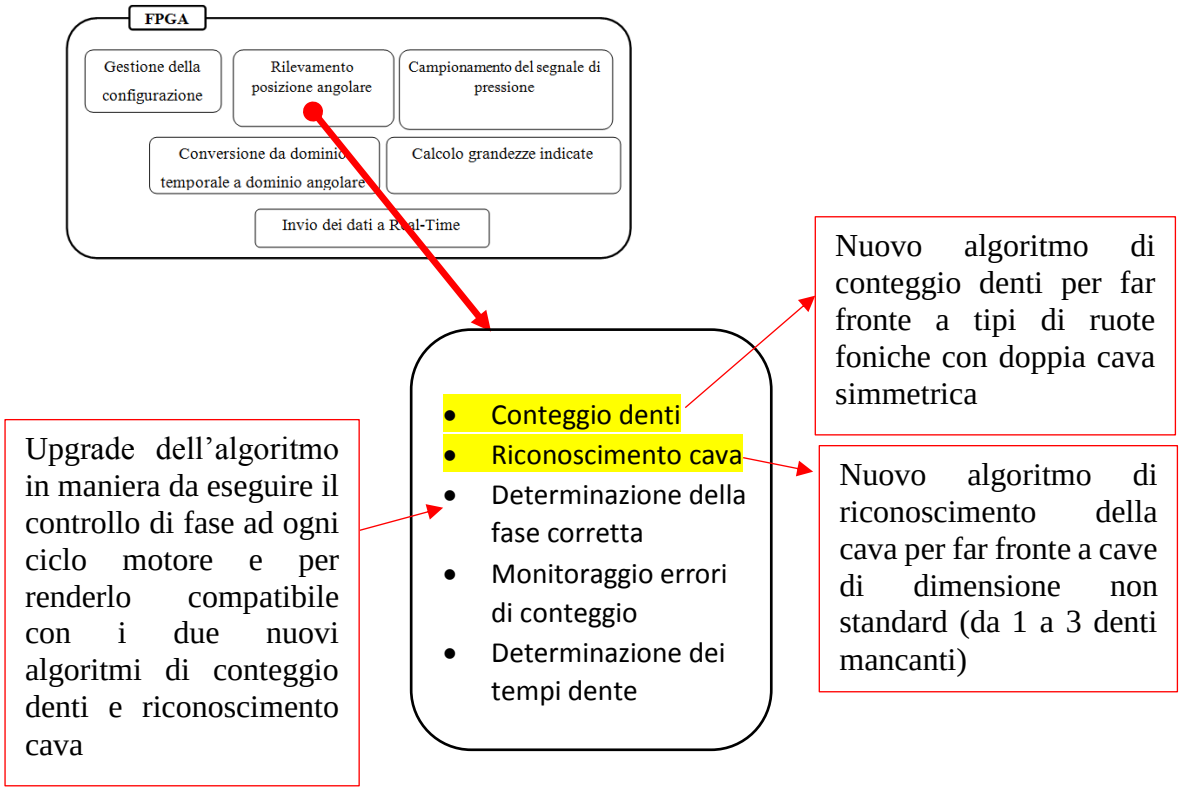


Figura A-3: schema a blocchi codice host PC

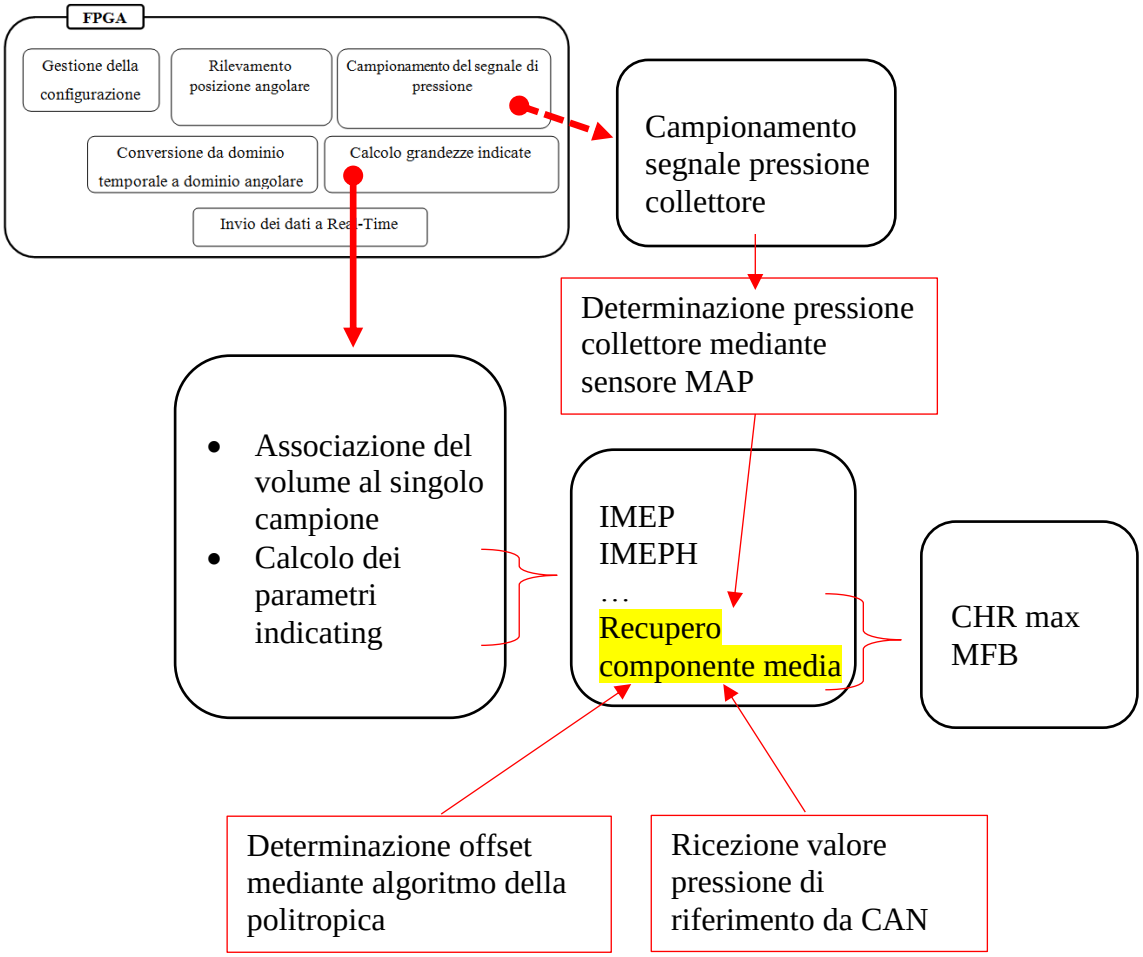
Per una descrizione più dettagliata dei singoli blocchi si rimanda ai lavori [1] ed [8].

Implementazione del codice a livello FPGA

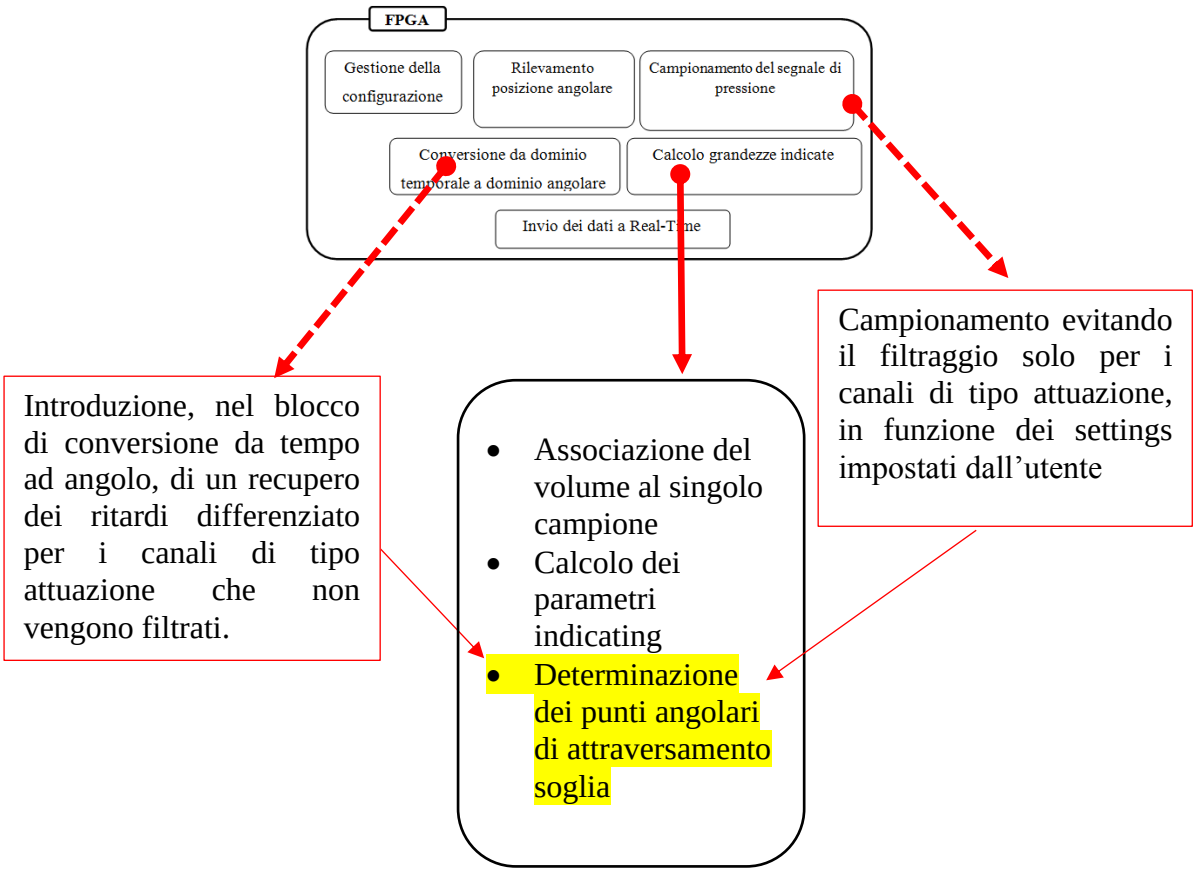
L'implementazione degli algoritmi sviluppati ha portato ad una rivisitazione di molte parti di codice FPGA. Innanzi tutto le evoluzioni riguardanti la fasatura (capitolo 2) hanno portato alla modifica della parte di rilevazione della posizione angolare:



Anche l'introduzione di nuove tipologie di pegging (capitolo 3) ha portato all'introduzione di nuove parti di codice a livello FPGA, all'interno della sezione di calcolo delle grandezze indicating:



Per quanto riguarda il tracking delle attuazioni (capitolo 4) è stato necessario introdurre una parte di codice totalmente nuova per l'esecuzione dell'algoritmo e allo stesso tempo modificare parti di codice per far fronte alle necessità di filtraggio differenziato tra i canali di pressione e i canali attuazioni. (vedi paragrafo 4.6).



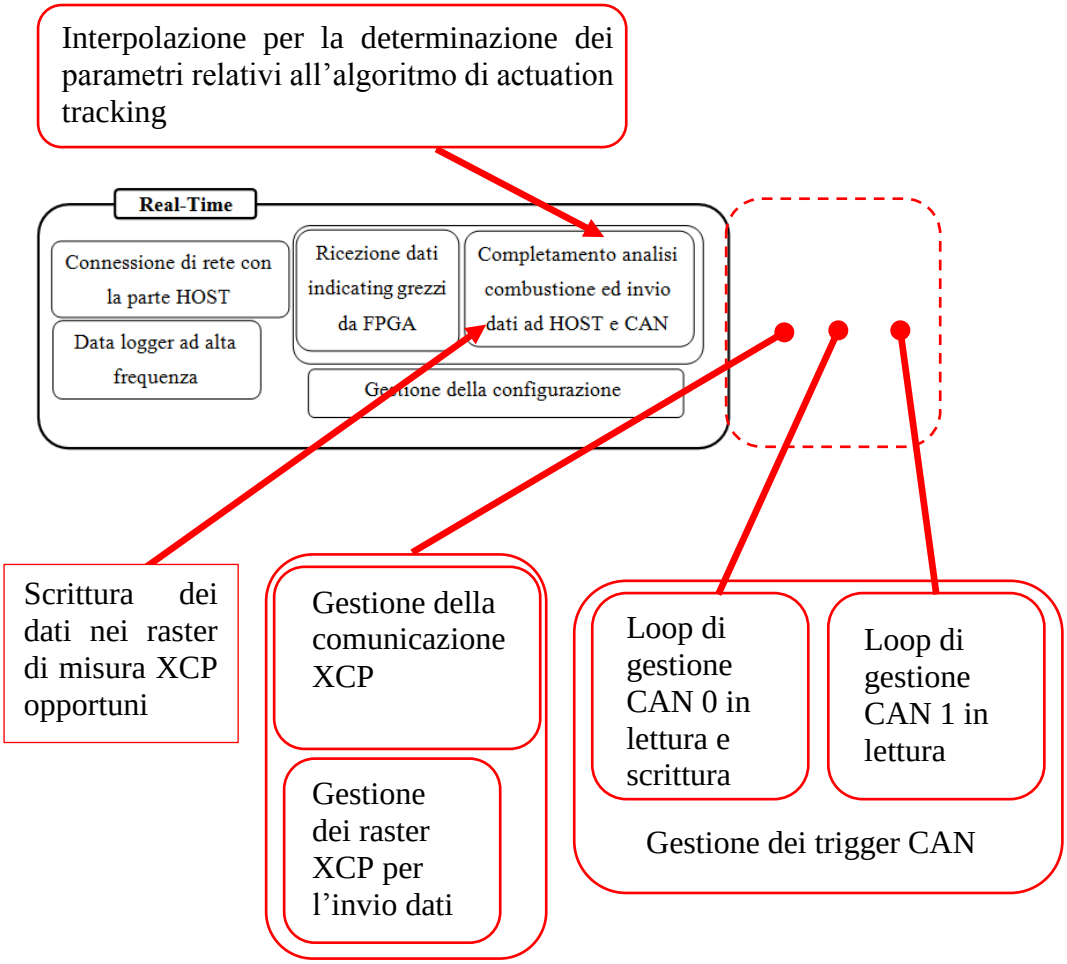
L'introduzione di nuovi parametri e di nuovi algoritmi ha portato obbligatoriamente alla modifica della sezione di invio dei dati a real-time e di gestione delle configurazioni, proprio per far fronte alla spedizione dei nuovi parametri calcolati e nuovi settings ricevuti. La sezione di invio dei dati a real-time ha subito forti cambiamenti anche a causa della necessità di spedire i risultati al livello real-time al termine di ogni singolo ciclo motore relativamente ad ogni TDC, per far fronte alla spedizione dei dati CAN combustione per combustione (vedi paragrafo 5.3)

Implementazione del codice a livello Real Time

Qualsiasi modifica venga introdotta nel codice a livello FPGA implica obbligatoriamente una revisione, implementazione od aggiornamento del codice real-time, in quanto i nuovi dati calcolati o le nuove configurazioni necessarie devono transitare attraverso il real-time per poter raggiungere l'FPGA. Si evita la descrizione delle singole modifiche introdotte per ogni aggiornamento di configurazione o di dati aggiunti in FPGA, nonostante abbiano

richiesto sforzi di sviluppo e di debug, evidenziando solo le modifiche macroscopiche più importanti introdotte durante il lavoro svolto.

Gli sviluppi più importanti sono quelli relativi alla comunicazione dei dati verso il mondo esterno mediante protocolli standard, cioè CAN e XCP (capitoli 5 e 7). Per quanto riguarda invece il tracking attuazioni, in real-time è stata introdotta una nuova sezione all'interno del blocco di completamento dell'analisi combustione che si occupa dell'interpolazione dei valori di attraversamento soglia (paragrafo 4.4).

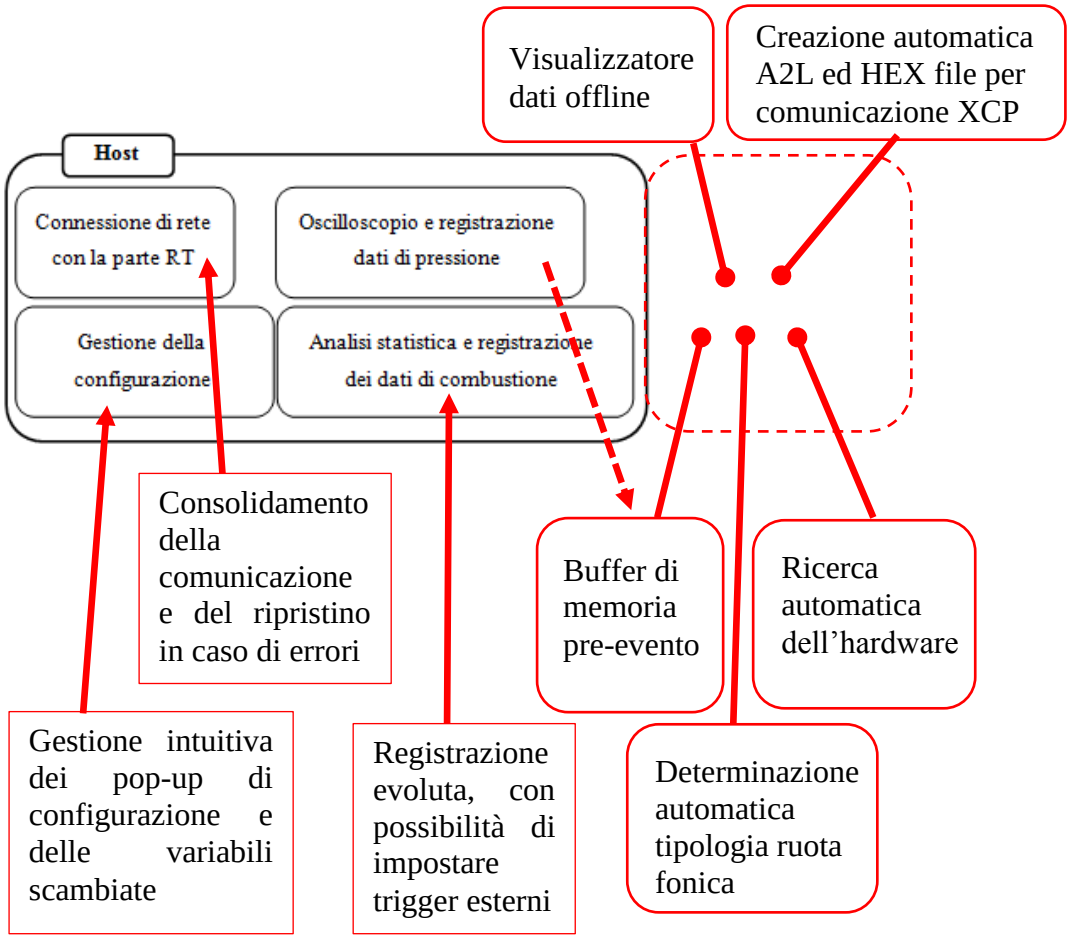


Implementazione del codice a livello host

Gran parte del lavoro svolto a livello host è stato scrivere codice che permettesse all'utente di interfacciarsi in maniera semplice ed intuitiva con le funzioni di visualizzazione dei dati e di setting dei parametri. Ovviamente ogni nuova funzionalità

introdotta ha richiesto l'introduzione di modifiche al codice in maniera da gestire e visualizzare i nuovi dati prodotti, permettendone anche il salvataggio. Questa parte è già stata messa in evidenza nel capitolo 1.4.

In aggiunta è stato introdotto codice per consolidare la comunicazione ethernet tra host e real-time (e il ripristino della comunicazione in caso di errori), e per la gestione del salvataggio dei dati, introducendo la possibilità di gestire un buffer di dati pre-evento (vedi paragrafo 6.2). Oltre a questo sono state introdotte funzionalità utili all'utente per semplificare l'inizializzazione e l'avvio delle prove riducendo il rischio di errori di configurazione o mancate connessioni.



Overview dei codici host e real-time

Per completezza si riportano i block diagram dei VI (Virtual Instrument file) del codice host e real time (per motivi di riservatezza non viene riportato quello FPGA). Si può

intuitivamente capire la struttura a blocchi costituita da vari loop temporizzati, all'interno dei quali sono presenti ulteriori sub VIs contenenti gli algoritmi sviluppati (non si entrerà all'interno dei singoli subVI, sempre per motivi di riservatezza). Il quadrato bianco con bordo rosso in figura rappresenta le dimensioni dello schermo del PC.

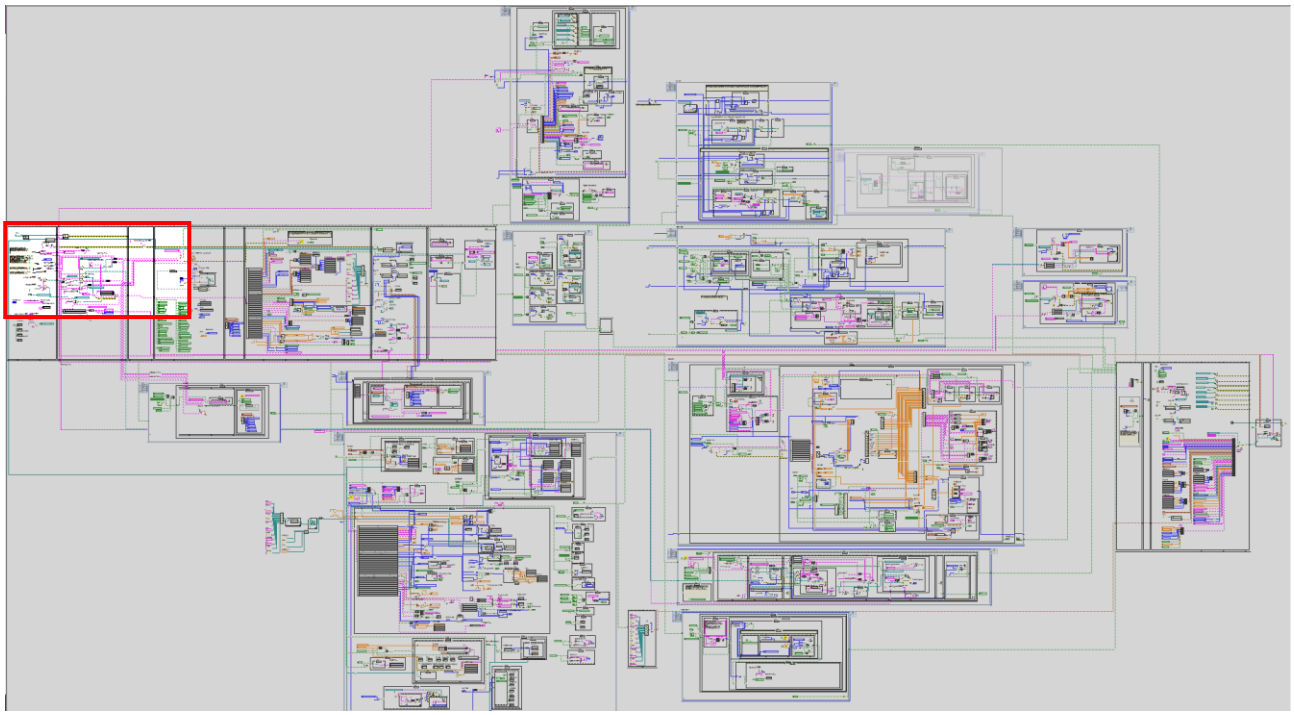


Figura A-4: block diagram host main VI

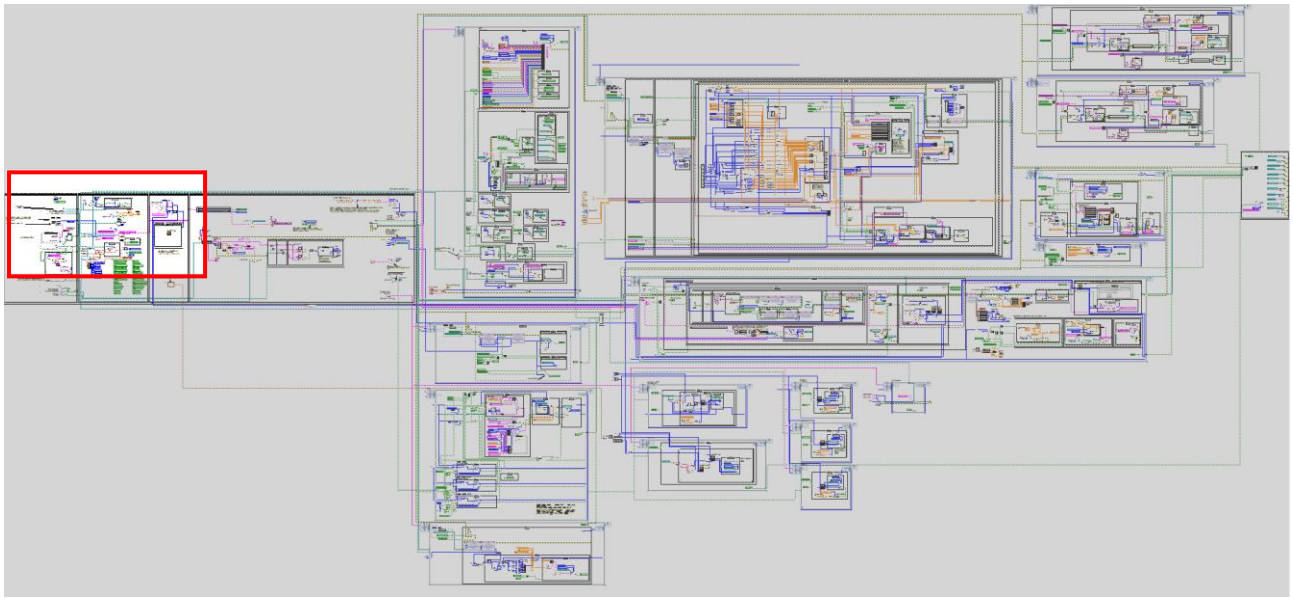


Figura A-5: block diagram real-time main VI

BIBLIOGRAFIA

[1] Solieri L., tesi di dottorato: *Sviluppo di algoritmi di analisi e diagnosi combustione in tempo reale per motori endotermici alternativi*, 2009

[2] Valbonetti M., tesi di dottorato: *Sviluppo di sistemi per l'analisi della combustione in tempo reale per motori endotermici alternativi*, 2013

[3] ASAM, *XCP Version 1.1*, 2008, ASAM e. V.

[4] ASAM, *ASAM MCD-2 MC (ASAP2 / A2L) Data Model for ECU Measurement and Calibration*, 2015, ASAM e. V.

[5] Heywood J. B., *Internal Combustion Engine Fundamentals*, 1989, McGraw Hill

[6] Ferrari G., *Motori a Combustione Interna*, 1992, Il Capitello

[7] Corti E., Moro D., Solieri L., 2008, *Measurement Errors in Real-Time IMEP and ROHR Evaluation*, SAE 2008-01-0980

[8] Bovicelli P., tesi di laurea: *Implementazione di una Piattaforma Software Modulare per l'Analisi della Combustione*, 2013

[9] Brown B. R., final year project: *Combustion Data Acquisition and Analysis*, Loughborough University

[10] Azzoni P. M., *Strumenti e Misure per l'Ingegneria Meccanica*, 2006, Hoepli