ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SECONDA FACOLTA' DI INGEGNERIA
CON SEDE A CESENA

CORSO DI LAUREA SPECIALISTICA
IN INGEGNERIA MECCANICA
Classe 36/S
Sede di Forlì

TESI DI LAUREA in

CONTROLLO DEI MOTORI A COMBUSTIONE INTERNA

DEFINIZIONE DELLE STRATEGIE DI ATTUAZIONE PER INIEZIONE E ACCENSIONE IN UN SISTEMA DI RAPID CONTROL PROTOTYPING REALIZZATO SU PIATTAFORMA NATIONAL INSTRUMENTS

CANDIDATO
Marco Cangini

RELATORE:
Chiar.mo Prof. Ing, Enrico Corti

CORRELATORE
Ing. Manuel Valbonetti

Anno Accademico [es.:2012/13]

Prima Sessione

INTRODUZIONE

Il lavoro svolto in questa tesi è mirato ad uno studio approfondito per lo sviluppo della centralina virtuale nata nei laboratori della Seconda Facoltà di Ingegneria di Forlì in ambito del progetto "VECU" (virtual engine control unit).

Gli obiettivi di questa trattazione sono in particolare 3:

1. Provvedere all'effettivo controllo del motore Maserati a banco cercando di risolvere tutte le problematiche che si sono riscontrate nelle precedenti versioni. In particolare per due aspetti:
   - la verifica delle strategie di attuazione implementate attraverso l'Host pc, come vedremo dettagliatamente, attraverso logica *FPGA*. Si è provveduto quindi a una prima fase, di verifica software della posizione angolare di ognuna delle attuazioni e una seconda fase di verifica hardware per avere un effettivo controllo di tutti i dispositivi in gioco.
   - In seconda battuta si è proceduto ad effettuare una serie di modifiche al software in modo da risolvere problemi legate alla fase di avviamento motore detta anche "*cranking"* dove la centralina cerca di incrociare i segnali di posizione angolare albero motore e di albero di distribuzione per iniziare in maniera attiva le attuazioni. In particolare si è svolto un lavoro di aggiornamento del software i fasatura finalizzato a migliorare la fase di fasatura del motore.

2. Rendere l'utilizzo del programma il più possibile semplificato per chi dovesse in futuro cimentarsi con la gestione della VECU. Infatti la potenzialità della piattaforma National

instruments utilizzata implica una certa esperienza con la realizzazione della parte Software, si è così reso necessario avere una versione VECU pronta a essere utilizzata anche da chi non ha piena esperienza con questo programma. Per far ciò è stato necessario selezionare le informazioni necessarie all' effettivo svolgimento delle attività e realizzare una configurazione dati di base che permettesse l'avvio dell'applicazione. Si è scelto un caricamento dati che in pratica fornisce all'utente sempre la ultima versione del programma, in modo da avere sempre sott'occhio l'ultima configurazione funzionante dei parametri e una modalità semplificata di caricamento e salvataggio dati.

3. Aumentare le potenzialità di controllo della VECU per rendere il motore a banco completamente indipendente dalla centralina di serie, in modo da poter gestire il motore nella sua totalità e spingerlo verso qualsiasi tipo di punto di funzionamento. Questo obiettivo si è perseguito lavorando su due dispositivi:
   - La pompa del carburante. Abbiamo creato un circuito di comando della pompa vero e proprio e separato dal vecchio comando di centralina di serie e abbiamo implementato una parte di software dedicata nella VECU con l'obiettivo di controllarla in caso di funzionamento, ma anche di spegnerla automaticamente per salvaguardare la batteria.
   - Il comando delle Pierburg che controllano le Waste Gate (implementando il lavoro svolto nella tesi di Maioli alla VECU) che permette di svincolare il controllo Waste Gate, cioè in pratica la sovralimentazione di boost, dalla centralina di serie, dando la possibilità di utilizzare il motore in qualsiasi tipo di condizione.

Tutto questo con l'obiettivo di mantenere lo sforzo computazionale nei limiti delle nostre potenzialità di calcolo tramite PXI (che è in pratica il computer industriale che fa da tramite tra software e sistema fisico).

Di seguito verranno esposti i concetti fondamentali riguardanti i capitoli che compongono questo elaborato:

- ✓ Il 1° capitolo descrive brevemente le motivazioni che hanno portato alla nascita del progetto VECU, gli obiettivi che si prefigge di centrare e i requisiti in termini di flessibilità e prestazioni che deve avere la centralina virtuale in sostituzione di quella originale. Vengono inoltre presentate le metodologie per la realizzazione e sperimentazione dell'unità di controllo, e i requisiti che deve avere il sistema per realizzare test in Rapid Control Prototyping, sia a livello di hardware che a livello di software.
- ✓ Il 2° capitolo presenta labVIEW a livello di software, nei suoi layer *Windows, Real Time,* e *FPGA* e l'hardware a disposizione, costituendo così gli istrumenti necessari all'implementazione della centralina.
- ✓ Il 3° capitolo presenta il sistema controllato montato a banco prova con i dettagli della centralina di serie. Di seguito vengono poi spiegate le caratteristiche da rispettare del controllore.
- ✓ Il 4° capitolo contiene la descrizione del programma di controllo in *FPGA*, in particolare per quello che riguarda la gestione delle attuazioni e l'acquisizione dei segnali di fonica e fase.
- ✓ Il 5° capitolo contiene la descrizione della parte *RT* del sistema e la descrizione dei calcoli fatti per la gestione delle attuazioni con l'acquisizione dei sensori.

- ✓ Il 6° capitolo invece mostra la parte di connessione fra la parte di controllo software e il controllo fisico del motore con l'insieme di tutte le connessioni fra sensori del motore e controllo VECU.
- ✓ Il 7° capitolo presenta le conclusioni del progetto di studio e il confronto fra gli obiettivi preposti e quelli effettivamente raggiunti. Infine vengono esposti i prossimi sviluppi del progetto VECU.

Capitolo 1

Progetto VECU e Rapid Control Prototyping

1.1 Obiettivi del progetto VECU

Il progetto VECU (Virtual Engine Control Unit) è nato con lo scopo di sviluppare un sistema di controllo in tempo reale di un motore a combustione interna, a ciclo Otto o Diesel, aspirato o sovralimentato, con un qualsiasi numero di cilindri, tramite la realizzazione di una centralina virtuale (da cui il nome), che si sostituisce a quella originale del motore, secondo una strategia detta full pass: si prevede l'utilizzo di hardware esterno su cui sono implementati via software tutti gli algoritmi di calcolo di controllo del motore. L'avere a disposizione hardware e software che bypassano completamente la centralina di serie, consente una gestione completa del motore, permettendo di gestire in modo flessibile le attuazioni di accensione e iniezione con elevata precisione e di sviluppare nuove strategie di controllo, in grado di facilitare enormemente la messa a punto e la sperimentazione dei motori a combustione interna, evitando le problematiche comportate dalle logiche di comando chiuse legate alla struttura hardware delle ECU di serie.
Il sistema, inoltre, consente di simulare malfunzionamenti mirati (tipo misfire e misfuel) o altre condizioni di funzionamento anomale così da osservarne la reazione e consentire la messa a punto di algoritmi diagnostici che individuino e pongano rimedio

agli errori o anomalie che possono manifestarsi durante il normale funzionamento del motore. Inoltre, il sistema deve prevedere la diagnosi in tempo reale della detonazione in ciascun cilindro, e consentire di agire sui parametri di funzionamento del motore in tempo reale.
Per fare tutto ciò occorre sostituire completamente l'apparato di gestione dell'accensione e dell'iniezione originale, e crearne uno completamente flessibile e facilmente personalizzabile in base alle possibili esigenze.

1.2 Requisiti del sistema VECU

Per l'estrema flessibilità, precisione e affidabilità che il sistema deve garantire, sono stati individuati requisiti specifici molto rigidi da rispettare:

- Deve poter gestire qualsiasi tipo di motore a combustione interna ad accensione comandata o spontanea (con specifiche modifiche agli stadi di potenza), aspirati o compressi, con frazionamenti anche elevati e ampio regime operativo;
- Il sistema di fasatura deve individuare la posizione angolare dell'albero motore all'interno del ciclo completo di 720° con un errore di posizione inferiore a 0.1°;
- Il sistema di gestione delle attuazioni deve eseguire i comandi stabiliti con una risoluzione di 1 µs sul tempo d'iniezione e di 0.1 ° sull'istante di scarica bobina;
- Non devono essere presenti vincoli sulla tipologia del comando da eseguire: in particolare, dovranno essere possibili iniezioni multiple per ogni cilindro, misfire e misfuel arbitrari, funzionamento full-group o multi-point degli iniettori;
- La capacità di calcolo deve essere sufficiente per eseguire l'algoritmo ogni ciclo per ogni cilindro, e i calcoli devono poter

essere anche complessi senza per questo compromettere la stabilità e tempistiche del sistema.

1.3 Metodologie utilizzate nella sperimentazione

Le metodologie utilizzate oggi nella progettazione di centraline in campo automobilistico consentono di realizzare prototipi in breve tempo e di testarli in modo immediato, svincolando la fase di sviluppo da quella produttiva.
Esse sono essenzialmente tre:

1. *Software In the Loop (SIL)*: è una tecnica di verifica della centralina basata su l'emulazione completa via software del sistema cui è destinata. In pratica, secondo la metodologia SIL il software di controllo da verificare è fatto interagire direttamente con l'emulazione del sistema cui è destinato per poterlo sollecitare come se fosse in condizioni operative. In questo modo è possibile eseguire una prima verifica già da quando sono disponibili solo modelli software funzionanti, senza attendere la disponibilità di prototipi fisici. Il vantaggio di quest'approccio è dato dalla possibilità di disporre di ambienti virtuali per il test utilizzando hardware standard (ad es. PC) evitando il ricorso a costosi sistemi di simulazione fisici (come i banchi necessari per l'Hardware In the Loop) o a prototipi funzionanti.

2. *Rapid Control Prototyping (RCP)*: il codice sviluppato è compilato sulla centralina con la possibilità di eseguire revisioni degli algoritmi di calcolo in tempi brevi e con costi ridotti.

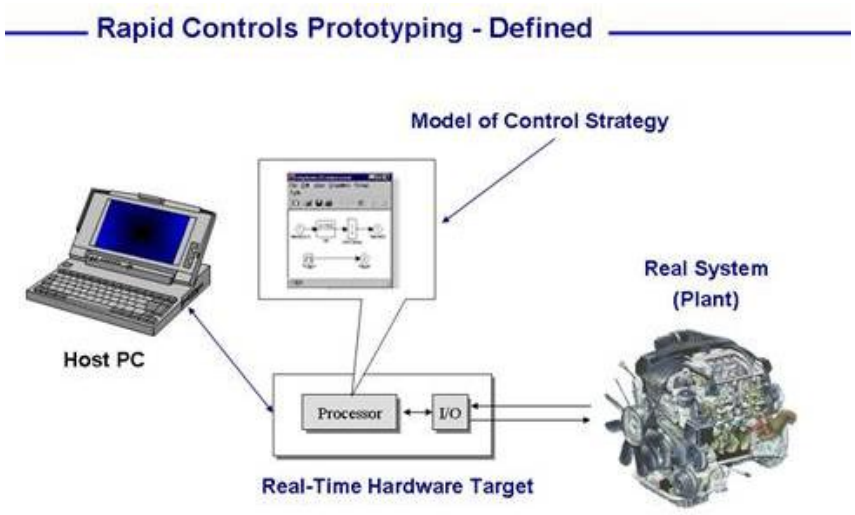


Figura 1.1: Logica del Rapid Control Prototyping

3. *Hardware In the Loop (HIL)*: il test si esegue collegando l'unità di controllo elettronico ad un apposito banco che riproduce, in modo più o meno completo, I' apparato elettrico ed elettronico del sistema a cui è destinata l'unità da verificare. Questo offre la possibilità di riprodurre svariate condizioni operative e di svolgere in modo automatizzato test di lunga durata senza attendere la disponibilità del prodotto finale cui la centralina è destinata.

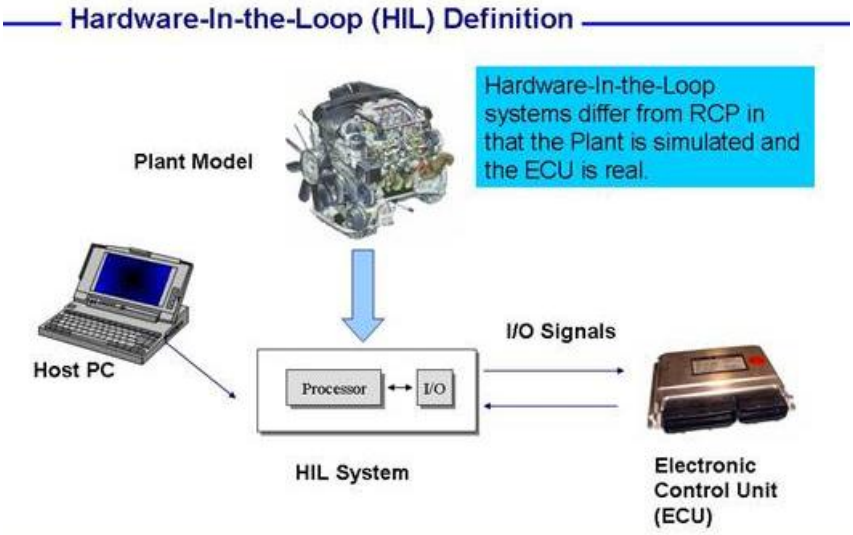


Figura 2.2 : Logica Hardware in the loop

L’insieme di queste tecniche consente di seguire un ciclo, durante la progettazione delle centraline, a partire dalle specifiche di sistema fino alla calibrazione dei parametri ottimali in cui vengono simulati sia il sistema di controllo, sia il sistema controllato. Questo percorso è denominato V-Cycle ed è illustrato nella figura 1.3. L'obiettivo di un rapido sviluppo è di rendere questa ciclo il più efficiente possibile riducendo al minimo le iterazioni richieste per un progetto. Se l'asse x del grafico è pensato come una misura del tempo, l'obiettivo è di chiudere la "V" per quanto possibile, avvicinando tra loro le due gambe del diagramma, in modo da ridurre i tempi di sviluppo.

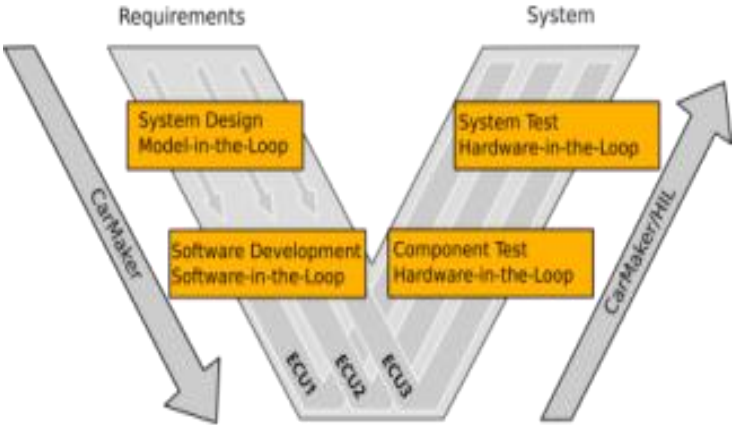


Figura 1.3: schema V-Cycle

1.4 Requisiti del sistema RCP

Attualmente sono disponibili sul mercato molti prodotti adatti ad applicazioni di tipo Rapid Control Prototyping (RCP). Si tratta, sostanzialmente, di sistemi dotati di un processore dedicato, che consentono lo svolgimento in tempo reale dei calcoli necessari a eseguire il loop di controllo, e di schede di Input/Output, che consentono l’elaborazione e lo scambio di dati con l’esterno, in forma analogica e digitale. La prestazione più qualificante è senz' altro la capacità di eseguire loop di calcolo a frequenza

molto elevata, che consente di controllare fenomeni rapidamente variabili nel tempo. Per questo i sistemi di controllo in tempo reale sono dotati di processori dedicati, basati su sistemi operativi "Real Time", le cui prestazioni sono in perenne e rapidissima ascesa. Da non sottovalutare è inoltre l'importanza del software di gestione del sistema che deve rendere semplice la programmazione, ovvero la prototipazione, per esempio, degli algoritmi di controllo, e facilmente controllabili i parametri dei programmi elaborati, i quali devono infine poter essere modificati in tempo reale, cioè senza interrompere lo svolgimento dei calcoli. Infine la possibilità di comunicazione con unità esterne, quali centraline elettroniche, calcolatori, strumenti di misura, secondo protocolli standard (CAN, seriale, ecc.), può svolgere un ruolo di fondamentale importanza; se da una parte, proprio per questo motivo, i produttori tendono ad offrire le medesime possibilità di accesso ad hardware esterno, dall'altra le modalità di impiego di tali funzioni possono essere più o meno semplificate. Volendo riassumere molto sinteticamente le caratteristiche salienti di un sistema di questo tipo, esse potrebbero essere così elencate:

- Potenza di calcolo in tempo reale (tipo e velocità del processore, memoria disponibile, sistema operativo);
- Funzioni di I/O analogico (frequenza di campionamento, numero di bit, range di tensione, velocità d'immagazzinamento dati);
- Funzioni di I/O digitale (frequenza di aggiornamento, possibilità di gestire protocolli di comunicazione, generazione pattern PWM o simili per il controllo di attuatori, interpretazione di segnali in ingresso, possibilità di gestire interrupt e di generare segnali su tale base);
- Software di programmazione;
- Software di gestione.

Capitolo 2

L'ambiente lab-VIEW

2.1 Introduzione

L'ambiente di sviluppo software scelto per l'implementazione della centralina virtuale è Lab VIEW for Windows con i due moduli aggiuntivi Lab VIEW FPGA module e Lab VIEW Real Time, che, uniti all'hardware realizzato dalla National Instruments, ha fornito gli strumenti ideali per progettare e realizzare in tempi relativamente brevi una VECU efficace e performante: infatti, non tutta la logica della centralina è gestita da un processore di un normale PC, poiché molte parti del programma vanno eseguite ciclicamente ad una frequenza che solo un hardware dedicato può non solo raggiungere, ma soprattutto garantire.
Per soddisfare i requisiti temporali imposti dalla necessità di fasare correttamente il motore e di generare attuazioni molto complesse, si è scelto di adottare un sistema di I\O basato su un chip tipo FPGA (Field programmable Gate Array, letteralmente 'matrice di porte logiche programmabile sul campo') ad alte prestazioni, le cui operazioni sono controllabili ad alto livello con il software LabVIEW for FPGA. L'onere di effettuare i calcoli dei parametri di funzionamento e dei comandi di attuazione è invece affidato a un sistema real time che gira su un computer esterno su cui è installato un sistema operativo di tipo deterministico pilotato in ambiente Lab VIEW Real Time, così da poter eventualmente dedicare molte delle risorse dell' FPGA all'implementazione delle parti di gestione del motore più raffinate e che richiedono la massima precisione, quali la fasatura e le attuazioni di iniezione e accensione del combustibile. La scheda

FPGA e il controller Real Time convivono in un unico chassis fornito dalla National Instruments che cura anche la comunicazione tra le due parti.
Complessivamente il sistema di controllo è quindi strutturato in due blocchi indipendenti: da un lato l'elaboratore Real-time (chassis, controller e scheda di I/O dall'altro una Human Machine Interface, che comunicano tra loro attraverso una connessione ethernet a 100 MBps, e in cui il funzionamento del primo prescinde dalla presenza del secondo (pertanto un eventuale malfunzionamento del dispositivo di interfaccia non ha conseguenze sul controllore). L'utente ha a disposizione una Graphical User Interface che fornisce l'accesso ai controlli necessari alla gestione del motore e permette di memorizzare su HD tutti i dati di funzionamento, per effettuare eventualmente un'elaborazione off-line. L'interfaccia è sviluppata con LabVIEW e può essere eseguita su una varietà di sistemi: da un semplice PC dotato di sistema operativo Windows o Linux, fino a dispositivi handheld, programmabili grazie al modulo Lab VIEW for PDA.

2.2 Hardware

Il cervello di tutto il sistema di controllo è rappresentato dall'elaboratore NI PXI-8186 dotato di sistema operativo Phar Lap che consente di utilizzare il modulo Real-Time di LabVIEW: tale componente è di fondamentale importanza perché è quello che consente alla VECU di eseguire calcoli in tempo reale. La macchina nel suo insieme è composta da diversi moduli che assolvono a specifiche funzioni tra cui l’acquisizione dati e la comunicazione con eventuali altri elaboratori; il tutto è racchiuso in uno chassis industriale.

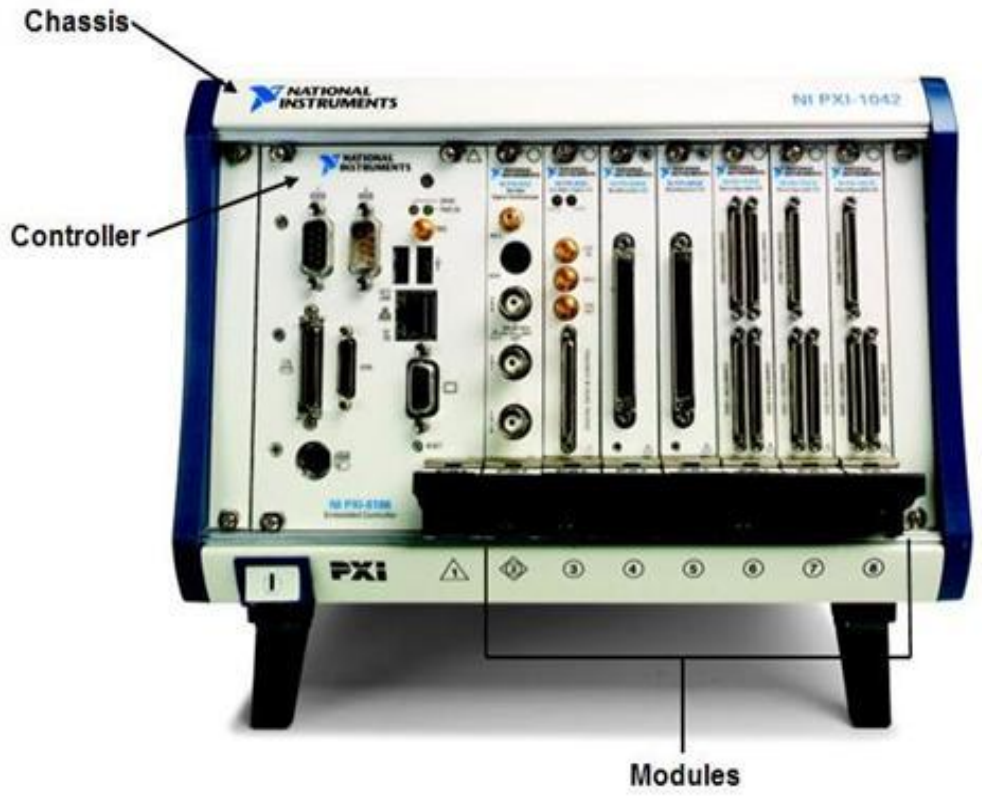


Figura 2.1: Macchina PXI racchiusa nello chassis completa di moduli e controller

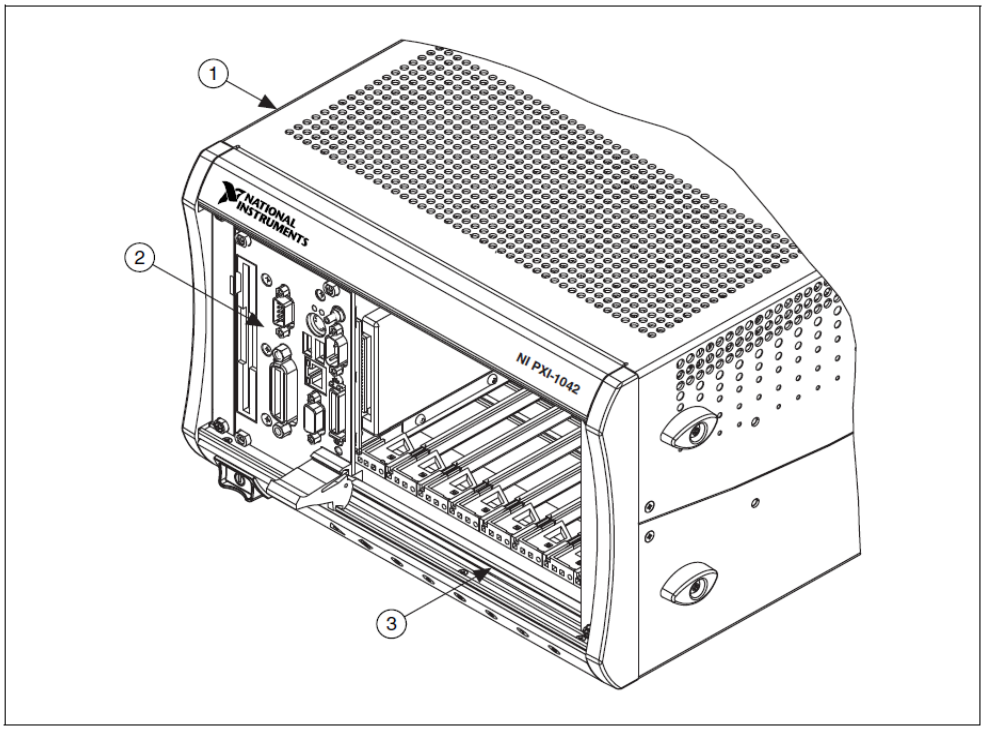
Il sistema implementato è così composto:

- Chassis NI PXI 1042
- Controller NI PXI 8186
- Scheda I/O riconfigurabile NI-PXI 7831R da 1 Mega-Gates
- Scheda I/O riconfigurabile NI-PXI 7833 R da 3 Mega-Gates
- Linea CAN NI PXI 8464
- Scheda di I/O temporizzata NI-PXI 6602
- Scheda di Multi-I/O NI-PXI 6133
- Scheda NI-PXI 6259 Multifunction DAQ
- Scheda di uscita analogica NI-PXI 6713

2.2.1 Chassis NI PXI-1042

Lo chassis PXI-1042 prodotto da National Instruments dispone di 8 slot adatti ad accomodare schede PXI o CompactPCI in formato 3U. È progettato per assolvere i compiti richiesti da una grande famiglia di applicazioni, dalla misura al controllo.
I bus PXI presentano uno standard maturo ed elevate prestazioni in campo sia scientifico che industriale, con la possibilità di scegliere tra due diversi clock, 1 KHz e 1 MHz. Questo consente di ottenere una risoluzione massima di 1 μs e di eseguire software con frequenza di ciclo fino 1 MHz. Inoltre, sono presenti 8 linee di interrupt hardware disponibili su tutte le schede collegate al bus, necessarie per sincronizzare tutte quelle attività che, come le acquisizioni e le attuazioni, richiedono l'intervento coordinato di più moduli, con sfasamenti inferiori al microsecondo. Infine, da non trascurarsi, i bus PXI garantiscono migliori prestazioni in ambiente Lab VIEW, nel quale è stato implementato il controllo motore.
Lo chassis è ottimizzato per fornire adeguato raffreddamento ai moduli in esso contenuti tramite due ventole di sistema che forniscono raffreddamento forzato e filtrato. Le ventole sono comandate secondo due schemi di funzionamento: HIGH o AUTO. In modalità HIGH le ventole vanno alla massima velocità operativa per fornire il massimo potenziale di raffreddamento mentre in posizione AUTO il sistema adatta dinamicamente il regime delle ventole per garantire un raffreddamento ottimale e, al contempo, un buon comfort acustico all'operatore.
Il clock generato ha una frequenza di riferimento di 10MHz e un'accuratezza di 25 ppm (parti per milione). Questo si traduce in una fluttuazione inferiore a 5ps e a uno sfasamento slot-to-slot inferiore a 250 ps. Il sistema di alimentazione è rimovibile dotato

di una robusta protezione contro le sovratensioni e funziona correttamente con le principali reti elettriche, a 110 o 220V, 50 o 60 Hz. La linea a 12V che alimenta le ventole è isolata per ridurre al minimo il rumore elettrico sul bus.
Lo chassis è dotato di connessioni BNC IN e OUT per il clock di riferimento. Quando il controller rileva la presenza di un segnale esterno, questo rimpiazza quello generato internamente. La connessione in uscita fornisce una copia, non TTL (transistor-transistor logic) del segnale di riferimento per la sincronizzazione di diversi moduli.



1 PXI-1042 Chassis 2 NI Controller 3 Injector/Ejector Rail

Figura 2.2: Chassis PXI 1042

Un sistema di monitoraggio remoto permette di tenere sotto controllo i valori delle tensioni e delle temperature tramite un connettore DB-9 presente sul retro. Un led di stato posto sull'interruttore di alimentazione indica un eventuale mal funzionamento del modulo di alimentazione.

CARATTERISTICHE			
Elettriche		**Raffreddamento**	
AC input		Num. ventole	2
Gamma ammessa	100-240 V	Capacità di	
Frequenza	50/60 Hz	raffreddamento per slot	25 W
Corrente	8 A	**Acustiche**	
Protezione Sovracorrente	10 A	*Pressione sonora*	
Regolazione		Modalità AUTO	50.5 dBA
3.3 V	<±0.2%	Modalità HIGH	58.7 dBA
5 V	<±0.1%	**Fisiche**	
±12 V	<±0.1%	Num. slots	1-7
Efficienza	70% tip.	Dimensioni cm	17x27x40
DC Output		Num. Unità Rack	4U
Massima corrente		Peso	8.4 Kg
3.3 V	12 A	MTBF	113000 h
5 V	25 A	**Ambiente operativo**	
12 V	3.5 A	Temperatura	0-55°C
-12 V	2 A	Umidità	0-90%
Regolazione del carico		**Bus 10 Mhz clock**	
3.3 V	<5%	Accuratezza	±25ppm
5 V	<5%	Oscillazione MAX	5 ps
12 V	<5%	Sfasamento MAX	250 ps
-12 V	<5%		

Tabella 2.3: Specifiche PXI 1042

2.2.2 Controller NI PXI-8186

Il controller PXI-8186 prodotto da National Instruments incorpora un processore Intel Pentium 4 a 2.2 GHz di frequenza, con 1 MB di DDR RAM e 40 GB di disco fisso e può essere usato in qualsiasi sistema PXI o CompactPCI. Esso è particolarmente indicato per applicazioni di analisi o per lo sviluppo di sistemi di

controllo su bus PXI in tempo reale poiché capitalizza al meglio le prestazioni del processore Intel ai benefici in termini di temporizzazione spinta e sincronizzazione derivanti dalla tecnologia PXI. Insieme allo chassis dedicato forma infatti una piattaforma compatta ad alte prestazioni per applicazioni modulari di acquisizione, analisi dati e controllo. Un'efficiente razionalizzazione dello spazio concesso al modulo dalle specifiche PXI ha permesso ai progettisti di inserire tutte le periferiche standard di un PC in una sola unità molto compatta. Questo alto livello di integrazione ha come effetto principale quello di lasciare liberi tutti gli slot dello chassis all'utente, ovviando in tal modo alla necessità di provvedere a complessi e costosi cablaggi.Il modulo incorpora tutte le periferiche di I/O standard: Ethernet 10/100 Base TX, tastiera, mouse e video. Inoltre dispone di una connessione GPIB (General Purpuse Inerface Bus, IEEE 488.2) che permette la connessione di strumentazione esterna. Due porte USB 2.0 permettono la connessione di lettori e masterizzatori CD/DVD, stampanti, dischi di memoria, ecc. ompletano la configurazione una porta parallela ECP/EPP (Extended Capabilities Port/Enhanced Parallel Port) e due porte seriali RS232. Il controller include una connessione 5MB esterna utilizzabile come input/output trigger o watchdog timer attraverso cui è possibile trasferire i segnali di interrupt presenti sul bus PXI all'esterno o fornire segnali di triggering al bus stesso. L'interfaccia video integrata è basata sul chipset Intel Extreme Graphics ed è capace di elevate prestazioni sia in 3D che in 2D: questa architettura, basata sulla condivisione della memoria di sistema, richiede il perfetto bilanciamento delle risorse, in termini soprattutto di RAM. La memoria volatile di sistema è del tipo double-data-rate (DDR) SDRAM, che offre un'ampiezza di banda doppia rispetto alle memorie convenzionali. Questa caratteristica rende il controller ideale per applicazioni di

analisi dati particolarmente intense. La configurazione base prevede 256Mb di RAM espandibili a 1024Mb. Il modulo è fornito con un set di software preinstallato e pronto all'uso:

- Microsoft Windows XP Professional o Windows 2000;
- Immagine hard drive per il ripristino del sistema;
- Drivers per NI-VISA e NI-488 (GPIB);
- Drivers per le periferiche installate.

Il controller può essere avviato con il comune sistema operativo Windows XP o in modalità Phar Lap per un'esecuzione molto più affidabile e prestante delle applicazioni real time, implementate con Lab VIEW Real Time e successivamente caricate sul controller tramite rete Ethernet.

CARATTERISTICHE		
Computazionali		
Processore	2.2 GHz Pentium 4-M	
Ethernet	10/100 TX. Connettore RJ45	
Video	Intel Extreme Graphics	
Porta seriale	2RS232	
Porta parallela	IEEE 1284 connttore tipo C	
GPIB	PCI-GPIB/TNT, connettore micro D25, IEEE488	
USB	2 versione 2.0	
Tastiera/Mouse	Connettori PS/2	
RAM	1 slot SO-DIMM, 256Mb espandibili a 1024Mb	
Hard drive	30GB minimo, 2.5", intrfaccia Ultra ATA100	
Assorbimento corrente elettrica		
Tensione (V)	Tipico (A)	Massimo (A)
+3.3	4.0	5.0
+5	6.5	8.0
+12	0.15	0.2
-12	0	0
Fisiche		
Dimensioni	PXI 3U, 8.1x13x21.6 cm	
Peso	1.0 Kg	
MTBF	170248 ore	
Ambiente operativo		
Temperatura	5-40°C	
Umidità relativa	10-90% non condensante	

Tabella 2.4: Specifiche controller NI PXI 8186

2.2.3 Scheda di I/O riconfigurabile NI PXI-7831/7833 R

Le schede NI PXI-7831R di National Instruments sono equipaggiate con una tecnologia che permette di disporre di hardware I/O riconfigurabile. Il cuore di ognuno dei moduli è un FPGA prodotto da Xylinx, le cui porte, slices e celle assumono una configurazione hardware determinata dal compilatore che traduce il codice prodotto con un modulo speciale di LabVIEW. È possibile personalizzare il comportamento del dispositivo sfruttando le seguenti caratteristiche:

Immagine 2.5: Scheda NI PXI-7833R

- controllo sulla temporizzazione e sincronizzazione con risoluzione di 25ns;
- esecuzione di codice con frequenze di ciclo di 40MHz;
- 96 linee di I/O digitali completamente configurabili come input, output, timer, PWM (Pulse Width Modulation), encoder input e protocolli definiti dall'utente;
- disponibilità di 8 ingressi analogici con risoluzione di 16bit e frequenza di campionamento di 200 Ksample/s ognuno;
- presenza di 8 uscite analogiche con risoluzione di 16bit e frequenza di aggiornamento di 1 Msample/s ciascuno.

Unendo queste potenzialità a quelle offerte dalla piattaforma Lab VIEW Real Time è possibile realizzare applicazioni come:

- PWM e protocolli di comunicazione digitali;
- Simulazione Hardware In the Loop con l'ausilio di altro hardware per simulare il plant;
- Rapid Control Prototyping;
- controllo in tempo reale in ambito analogico o digitale.

Ogni linea digitale è configurabile indipendentemente dalle altre e memorizzabile nella memoria flash del dispositivo per assicurare un ben definito stato all'avvio. Uno speciale modulo software, LabVIEW FPGA Module, permette la programmazione in ambiente grafico di tutte le funzionalità della scheda, con un approccio molto simile a quello del normale Lab VIEW per PC. Tuttavia non tutte le funzionalità di LabVIEW sono implementabili (ad esempio sono possibili solo calcoli con numeri interi). Le due schede differiscono sostanzialmente per il numero di elementi logici e RAM a disposizione, cioè in pratica sulla scheda 7833 possono essere implementati algoritmi più lunghi e complicati che sulla 7831.

CARATTERISTICHE			
Ingressi analogici		**I/O digitale**	
Num.Canali	8	Num.Canali	96
Modalità acquisizione	Diff/SE	Compatibilità	TTL
Risoluzione	16 bit	Corrente uscita Max.	±5 mA
Tempo conversione	4 µs	Tensione ammessa Max.	-0.5→7V
Freq. Max. campionamento	200 kS/s	**FPGA riconfigurabile**	
Impedenza ingresso	10 GΩ		
Tensione campionata	±10 V	Gates di sistema	1M
Accoppiamento	DC	Num. Slices logici	5120
Tensione Max. ammessa	±42 V	Num. Celle equivalenti	11520
		RAM disponibile	80 Kbytes
		Clock	40 MHz
Uscite analogiche		**Fisiche**	
Num.Canali	8	Dimensioni	16x10 cm
Risoluzione	16 bit	Connettori	3
Tempo aggiornamento	1 µs	Pin	68 VHDCI
Freq. Max. aggiornamento	1MS/s	**Ambientali**	
Tipo di DAC	Enh R-2R	Temperatura	0-55°C
Range	±10 V	Umidità relativa	10-90% n.c.
Accoppiamento	DC		
Impedenza di uscita	1.25 Ω		
Corrente Max.	±2.5 mA		

Tabella 2.6: Specifiche scheda NI PXI-7833R

2.2.4 Scheda di I/O non riconfigurabili

Le schede non ancora descritte sono tutte schede di I/O multifunzionali che si distinguono per alcune funzioni specifiche.
Le caratteristiche comuni a tutte le schede sono:

- Presenza di almeno 8 canali DIO con frequenza di trasferimento di 10 Mbyte/s;
- Presenza di almeno 2 contatori/timer
- Tensione di AI fino a ±10V e di DIO fmo a ±5V

Di seguito alcune caratteristiche peculiari di ogni scheda:

- NI PXI-6602 Timing I/O
  - 32 canali digitali;
  - Nessuna linea analogica;
  - 8 Timer.

Immagine 2.7: scheda NI PXI-6602

- NI PXI-6133 Multifunction I/O
  - 8 ingressi analogici con frequenza d'acquisizione fino a 2.5 MS7s.

Immagine 2.8: scheda NI PXI-6133

- NI PXI-6259 Multifunction DAQ
  - 48 canali digitali;
  - 8 ingressi analogici differrenziali o 16 singoli da 1.25 MS/s
  - 4 uscite digitali da 1.25 MS/s (max)

Immagine 2.9: scheda NI PXI-6259

- NI PXI-6713 Analog output
  - 8 uscite analogiche da 1 MS/s (max)

Immagine 2.10: scheda NI PXI-6713

2.2.5 Il compact RIO

Il CompactRIO è un avanzato sistema di controllo ed acquisizione potenziato dalla tecnologia di input ed output riconfigurabili (RIO) sviluppata da National Instruments. Per le ridotte dimensioni (180 x 88 x 88 mm3) peso (1.58 kg) e bassi consumi (per l'alimentazione è sufficiente una batteria da 9V) può essere utilizzato per eseguire svariate applicazioni come acquisizioni on board (ad esempio su di una macchina o di una moto) monitoraggio e controllo di macchine industriali, Rapid Control Prototyping ecc.

Esso è costituito dalle seguenti parti assemblate in un unico case:

- lo chassis NI cRIO-9103;
- il controller real-time NI cRIO-9012;
- i moduli di I/O.

Lo chassis NI cRIO-9103 è il cuore del sistema in quanto contiene il nucleo I/O riconfigurabile (RIO) da 3 milioni di porte logiche programmabili. Il RIO FPGA presenta una connessione con ogni modulo I/O ed è programmato con funzioni elementari che permettono di scrivere o leggere informazioni sui

Immagine 2.11: chassis Compact RIO

canali provenienti da ognuno di essi. Dal momento che non c'è un bus di comunicazione condiviso tra l' FPGA ed i moduli I/O, le operazioni di input ed output in ogni modulo devono essere sincronizzate con precisione sfruttando una risoluzione di 25 ns. Il nucleo RIO può elaborare dati in formato intero, prendere decisioni e trasferire direttamente un segnale da un modulo ad un altro ed è collegato al controller real-time tramite un bus locale di standard PCI. Il controller NI cRIO-9012 garantisce l'esecuzione on board di applicazioni real-time di tipo deterministico realizzate sempre tramite Lab VIEW. Esso include 64 IB di DRAM e 512 MB di memoria flash per la memorizzazione dei dati ed è realizzato per lavorare mantenendo la propria affidabilità anche in ambienti ostili con temperature comprese tra i -40 ed i 70°C. Un processore industriale Pentium da 400 MHz unisce ai bassi consumi la possibilità di analizzare ed elaborare funzioni reali eseguendo loop di controllo ad una frequenza anche superiore ad 1kHz. L'esecuzione di codice può essere sincronizzata sia attraverso l'invio di IRQ dall' FPGA sia con un clock real-time preciso al ms. In aggiunta alla comunicazione tramite i protocolli TCP/IP, UDP, Modbus/TCP, IrDA e seriali il controller include server per gli standard VISA, HTTP e FTP. I moduli I/O si collegano direttamente al RIO FPGA e costituiscono un sistema ad alte prestazioni che conferisce alle applicazioni di input/output realizzate tramite software le qualità e la flessibilità di un tradizionale circuito elettrico completamente dedicato a tali funzioni. Il CompactRIO fornisce direttamente l'accesso hardware ai circuiti di input/output di ognuno dei moduli utilizzando le funzioni I/O elementari di Lab VIEW FPGA. Ogni modulo, inoltre, contiene internamente applicazioni di condizionamento del segnale che consentono, in molti casi, il collegamento diretto a sensori ed attuatori facilitato dalla presenza di terminali ere a vite,

BNC o connettori D-Sub. I moduli I/O utilizzati nella nostra configurazione hardware sono i seguenti:

- scheda di IlO digitale NI 9401
- scheda di acquisizione analogica NI cRIO -9201
- scheda di acquisizione analogica simultanea NI 9215
- scheda di uscita analogica NI 9263

Immagine 2.12: esempio di applicazioni moduli Compact RIO

La NI 9401 è una periferica ad 8 canali digitali ad alta velocità (100 ns) e può assumere tre configurazioni: 8 ingressi, 8 uscite, 4 ingressi e 4 uscite. È compatibile con la logica TTL e tramite il CompactRIO può essere utilizzata per implementare contatori e temporizzatori ad alta velocità, protocolli di comunicazione, generatori di impulsi ecc. La NI cRIO-9201 include 8 ingressi multiplex analogici ed è in grado di raggiungere una velocità di campionamento di 500 kS/s. Presenta una risoluzione di 12 bit ed acquisisce segnali in tensione di intensità compresa tra -10 V e + 10 V oltre ad avere una protezione contro i transitori del

segnale superiore a 100V tra i canali in ingresso e la terra comune. La NI 9215 include 4 ingressi analogici per un'acquisizione simultanea a 100kS/s con una risoluzione di 16 bit. Infine la NI 9263 prevede 4 uscite analogiche simultanee ad una velocità di 100kS/s e presenta un protezione fino a ± 30 V di picco. Tutte le schede sopra descritte presentano una doppio isolamento tra ogni canale e la terra a scopo di sicurezza e di barriera contro eventuali rumori. Lo sviluppo di una applicazione per CompactRIO prevede tre tappe:

1. Determinazione del target dello chassis per stabilire automaticamente i moduli I/O e sviluppare l'applicazione sulla scheda RIO FPGA;
2. Compilazione dell'applicazione RIO per sintetizzare ed ottimizzare il circuito elettrico che realizza l'applicazione stessa;
3. Sviluppo dell'applicazione in real-time per aggiungere le parti che manipolano funzioni reali, elaborano segnali, memorizzano dati e comunicano con l'esterno.

2.3 Software

Il Software che si è scelto di utilizzare per l' implementazione della centralina virtuale è LabVIEW 11, un linguaggio di programmazione grafico sviluppato appositamente dalla National Instruments, studiato e ottimizzato per facilitare la realizzazione di applicazioni per acquisizione, analisi di segnali e controllo dati, compatibile con gran parte dell’ hardware oggi disponibile per questa tipologia di applicazioni. Lab VIEW è costituito da tre diversi moduli (Lab VIEW for Windows, Lab VIEW Real Time Module, Lab VIE W FPGA Module) che si differenziano tra loro per il supporto hardware a cui è destinata l'applicazione da realizzare (PC, Controller Real Time, Scheda FPGA) e hanno

caratteristiche differenti soprattutto per quanto riguarda le tempistiche con cui vengono eseguite le varie applicazioni e per la complessità delle librerie messe a disposizione.

2.3.1 LabVIEW for Windows

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

Un blocco di elaborazione non è altro che la rappresentazione di una serie di operazioni che vengono eseguite sui dati in ingresso a seconda della funzione del blocco stesso: può ad esempio essere una semplice operazione matematica o di un intero programma a sé stante, poiché, come nei linguaggi di programmazione tradizionali, un programma può richiamare una *sub-routine*. L'interfaccia con l'utente di ogni VI è il *Front Panel* in cui sono posizionati controlli e indicatori che rappresentano

rispettivamente gli input e gli output interattivi del VI. I controlli sono manopole, potenziometri, quadranti, pulsanti, interruttori, comandi scorrevoli, caselle numeriche o di testo e altri meccanismi di introduzione di dati, mentre gli indicatori sono grafici, LED, tabelle e altri componenti che consentano di visualizzare gli output acquisiti o generati dal *Block Diagram*. Dopo aver realizzato il pannello di controllo, è necessario implementare le funzionalità richieste dall'applicazione nello schema a blocchi, unendo ed elaborando le icone che rappresentano gli oggetti del pannello frontale. Quest'ultima parte è il corpo centrale di un VI in quanto contiene il codice grafico, sotto forma di diagramma a blocchi, che gestisce gli oggetti presenti nel pannello di controllo. Nella realizzazione del codice si possono utilizzare, oltre ai capisaldi ella programmazione classica come i cicli while, for e la struttura case, gli elementi messi a disposizione dalle librerie di Lab VIEW come le strutture temporizzate (*Timed Structures*) che permettono di stabilire una priorità tra le varie parti del programma o funzione già implementate, le cosiddette *Buit-in functions*, per l'acquisizione, l'analisi e l'esposizione di dati; una volta terminata anche questa operazione è possibile personalizzare l'icona del VI in alto a destra, attribuendo al file stesso il numero di connettori desiderato, per poi poterlo utilizzare in seguito come subVI. L'enorme flessibilità di LabVIEW risiede anche nella possibilità di stabilire una scala gerarchica e una priorità dei VI, consentendo questo modo di realizzare strutture anche molto complicate senza mai perdere di vista il quadro di insieme.

2.3.2 LabVIEW Real Time

L’applicazione realizzata con Lab VIEW Real Time differisce da una realizzata per Windows per il fatto che l'elaboratore RT comunica con il mondo esterno tramite un’ interfaccia di rete oltre

alle schede di input-output. Quindi, sfruttando questo canale di trasmissione dati, è possibile modificare tutti i parametri del sistema RT, con la necessità però di utilizzare un computer esterno, definito Host, che visualizza l'interfaccia con l'utente, quindi aggiorna i dati in tempo reale e invia al modulo real time le modifiche effettuate.

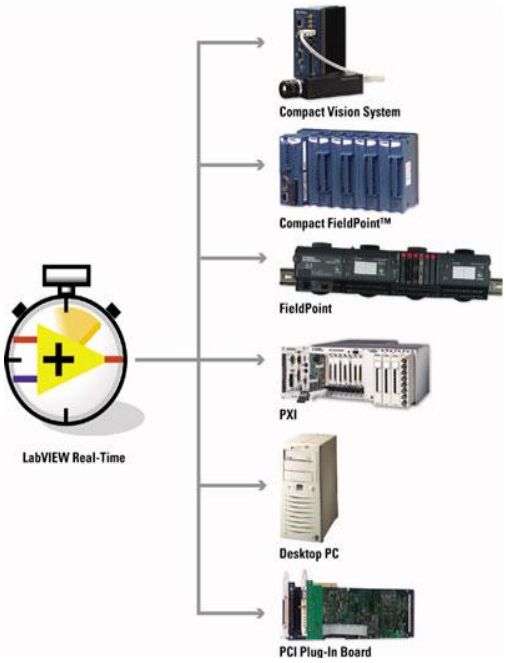


Immagine 2.13: Applicazioni Lab VIEW Real time

Questo espediente consente di al1eggerire il processore dal compito di visualizzare e aggiornare l'interfaccia, liberando più risorse di calcolo. In campo RT è possibile inoltre attribuire diverse priorità a ciascun VI a seconda della loro criticità temporale, cioè dell'intervallo di tempo entro i quali le applicazioni devono essere eseguite. In questo modo il processare, nel caso in cui siano richieste risorse maggiori rispetto a quelle disponibili, da precedenza ai VI definiti ad alta priorità, e quindi più critici.

2.3.3 Il modulo FPGA

Il modulo FPGA utilizza un approccio diverso rispetto al RT, definito hard real time, in cui il programmatore è celio, a priori, che l'esecuzione del programma avvenga entro tempi prestabiliti, in quanto l'algoritmo è implementato in hardware. La scheda utilizzata per questo compito è una NI PXI-7833R, la quale è dotata di un FPGA e nella quale viene codificato in hardware il modello realizzato In Lab VIEW, garantendo in questa maniera il determinismo della applicazione.

Immagine 2.14: applicazioni hardware FPGA (sito National Instruments).

- NI CompactRIO: sistema incorporato a moduli I/O intercambiabili, FPGA riconfigurabili in robusta e compatta armatura.

- NI Single-Board RIO: piattaforma ideale per applicazioni OEM, combinate con un processore Real Time, FPGA riconfigurabili e I/O analogiche e digitali incorporate.

- NI R series Multifunction RIO: questi Plug-in incorporano programmabili FPGA e I/O multifunzionali per essere utilizzate su PC o PXI per controllo e monitoraggio.

- NI FlexRIO: con questo modulo, potete creare un modulo I/O FPGA flessibile e incorporato per il monitoraggio e il controllo delle applicazioni PXI e PXI express

Il modulo FPGA mantiene la stessa facilità d'uso e intuitività dell'ambiente Lab VIEW, ma è dotato di un nuovo set di VI che consentono di realizzare applicazioni funzionanti sulla scheda utilizzata. Questo dispositivo è limitato però al calcolo intero a 64 bit, quindi non è possibile implementare VI complessi. Ciò è dovuto al fatto che, per rendere il più veloce possibile l'esecuzione delle applicazioni, si è reso necessario ridurre notevolmente la complessità dei calcoli che vengono effettuati. Inoltre altre limitazioni nascono dalla limitata disponibilità di RAM e di gales (elementi logici attivi).

La realizzazione del VI per il FPGA non è molto differente da quella abituale per uno di Lab VIEW in ambiente Windows, l'unica eccezione consiste nel dover compilare tale VI nel linguaggio del FPGA (binario) e poi caricarlo nella sua memoria. Questo processo può però richiedere anche alcune ore e deve essere eseguito ogni volta che si modifichi il flusso di dati all'interno del VI. Il software del sistema di controllo gestisce la comunicazione tra i vari VI. Una parte di questa viene eseguita sull' elaboratore real time, l'altra sulla scheda del FPGA. Inoltre è necessario un ulteriore VI, definito Host, che gira su di un altro elaboratore dotato di sistema operativo (ad esempio Windows) e di scheda di rete, che fornisce l' interfaccia utente per la gestione del sistema RT. Per una buona programmazione FPGA è necessario tenere a mente qualche regola pratica:

- formattare le variabili in modo da occupare meno memoria possibile (ad esempio in 8 bit con o senza segno);
- evitare divisioni dove possibile (FPGA accetta solo numeri interi);
- per divisioni o moltiplicazioni di potenze di due utilizzare uno shift logico di bit (ad esempio, per dividere per 4 shiftare tutti i bit della variabile a destra di due posizioni, per moltiplicare a sinistra);
- realizzare l'applicazione con il minor numero di operazioni possibile.

Capitolo 3

Il sistema controllato

3.1 Introduzione

Figura 3.1: Immagine motore Maserati AM 585.

Nonostante la VECU sia concepita per adattarsi a qualunque tipo di motore a combustione interna, il sistema controllato che sarà oggetto del nostro studio è il Maserati AM 585, che, per le sue caratteristiche tecniche e costruttive, costituisce un'ottima base di sperimentazione per il progetto: la doppia bancata, i due turbocompressori con waste gate a controllo elettronico, iniettori e bobine dedicati per ogni cilindro, controllo del titolo con una sonda lambda per bancata, controlli drive by wire e sensori per la diagnosi detonazione offrono parecchio e variegato materiale su

cui poter lavorare, e dal quale sarà poi più semplice adattarsi anche a motori meno sviluppati tecnicamente e meno complicati dal punto di vista gestionale.

3.2 Il motore allestito a banco pravo

Figura 3.2: foto del motore a banco.

Il motore Maserati AM585 ha equipaggiato molti modelli della casa del Tridente fino alla Coupé 3200 GT. Fedele alla tradizione che ha caratterizzato i modelli Maserati degli ultimi affili, questo motore è un 8 cilindri a V con un angolo di 90° tra le due bancate; con una cilindrata contenuta in 'soli' 3.2 litri, il motore quadro (alesaggio x corsa: 80 x 80 mm) è in grado di sviluppare una potenza massima di circa 370 CV e una coppia massima pari a 465 Nm; la buona potenza specifica è ottenuta grazie soprattutto alla presenza di due gruppi turbocompressori (uno per bancata) che lo sovralimentano e di altrettanti intercooler aria-aria (modificati in ana-acqua per esigenza della sala prove). La distribuzione è composta di due alberi a camme in testa per bancata comandati da cinghia dentata, i quali comandano 4

valvole per ogni cilindro. L'alimentazione avviene attraverso un sofisticato sistema di iniezione elettronica multipoint Magneti Marelli integrato con l'accensione, realizzata con 8 gruppi coil on plug (bobina sulla candela); entrambe sono comandate dalla centralina I.A.W.4CM C1 prodotta anch'essa dalla Marelli. Accoppiato al motore tramite un giunto a flange vi è il freno a correnti parassite prodotto dalla ditta Borghi&Saveri assieme al controller DCU 2000 che permette di regolare e visualizzare l'andamento di velocità, coppia e potenza.

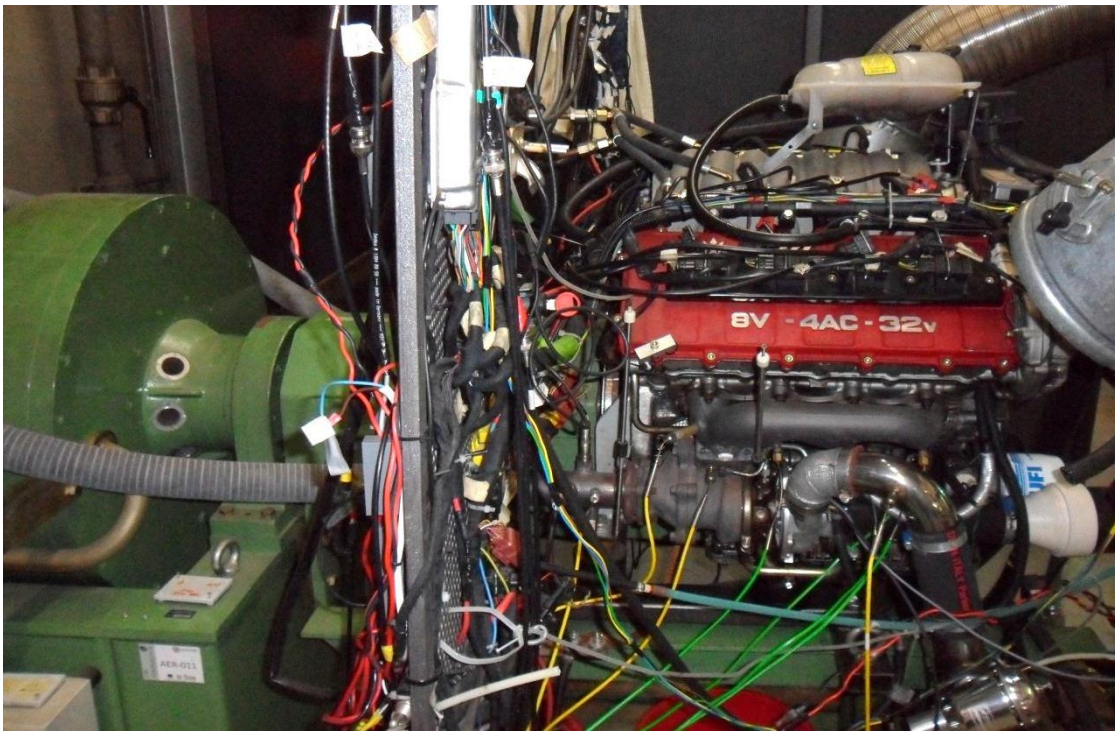


Figura 3.3: motore Maserati AM585 accoppiato al freno Borghi&Saveri.

A seguire la tabella 3.4 riporta in dettaglio le specifiche tecniche del motore.

CARATTERISTICHE

Motore		**Alimentazione**	
Tipo	AM 585	Tipo impianto	Marelli integrato
N° cilindri	8	Elettropompa combustibile	Bosch EKP 13.5
Alesaggio	80 mm		
Corsa	80 mm	Elettroiniettori	IW 058
Cilindrata totale	3217 cm^3	Regolatore pressione	40 RPM
Rapp. compressione	8	Sens. T Liquido refr.	Jaeger 402.183.01
		Sens, T Aria Aspirata	ATS 04
Potenza Max.	271kW 368.5CV	Sens. giri motore	CVM 01
		Centralina	I.A.W.4CM C1
Regime Potenza Max	6250 rpm	Filtro aria	JR 2 conici
Coppia Max	465Nm 47.4kgm	Sovralimentazione	2 Boost a gas di scarico
		Turbocompressori	IHI-RHF 5
Regime Coppia Max	4500 rpm	Intercooler	2 aria-acqua
Ordine accensione	1-8-4-5-7-3-6-2	Farfalla motorizzata	56 CFM 5
Regime Min	950 ± 50 rpm	Sonda λ	LSH 25
Lubrificazione motore		Bobine	Cooper BAE 01
Pressione nel circuito funzionamento motore a caldo		Elettrovalvola Booster	Pierburg VMI/122xx
a 1000 rpm	>2.5 bar	Sens. detonazione	NTK KNE
a 6000 rpm	4.5-5 bar	Elettrovalvola Canister	BOSCH TEV 2

Tabella 3.4: Specifiche Motore.

I turbocompressori sono del tipo a palettatura fissa e regolazione mediante valvola waste gate; essi sono costituiti essenzialmente da 2 giranti calettate su uno stesso albero. La girante della turbina è posta sul collettore di scarico ed è mossa dalla forza

cinetica e di pressione posseduta dai gas di scarico che vengono appositamente convogliati su di essa.

(a) (b) (c)

Figura 3.5: (a) Gruppo turbo compressione del Maserati
(b) dettaglio palette del compressore
(c) Ingresso in turbina dei gas di scarico

La rotazione della turbina investita dai gas pone in movimento alla stessa velocità la girante del compressore, posta sul condotto di aspirazione. Il compressore, grazie alla velocità di rotazione e alla forma particolare delle sue palette preleva l'aria esterna e la comprime nel collettore di aspirazione e quindi nei cilindri motore. Nella tabella seguente sono riportate le caratteristiche del turbo compressore.

CARATTERISTICHE Turbocompressore IHI-RHF 5	
Portata d'aria	1.4 – 1.8 m^3/min
Rapporto di compressione Max.	2.8
Velocità Max.	180000 rpm
Temperatura Max. gas di scarico	950° C
Applicazione nei motori benzina	54 - 153 kW
Applicazione nei motori diesel	40 – 113 kW
Peso (senza valvola waste gate)	3.2 kg

Il circuito di aspirazione e sovralimentazione è costituito da due circuiti indipendenti che confluiscono in un unico collettore di aspirazione. Il circuito è composto dai seguenti elementi:

1. due filtri aria con manicotto di collegamento al relativo turbo gruppo;
2. due valvole di corto circuito (valvola pop-off);
3. due turbo gruppi di sovralimentazione;
4. due intercooler;
5. corpo farfallato motorizzato Drive By Wire;
6. collettore di aspirazione;
7. elettrovalvola gestione sovralimentazione Pierburg;
8. valvola waste-gate;
9. precatalizzatore;
10. collettore di scarico.

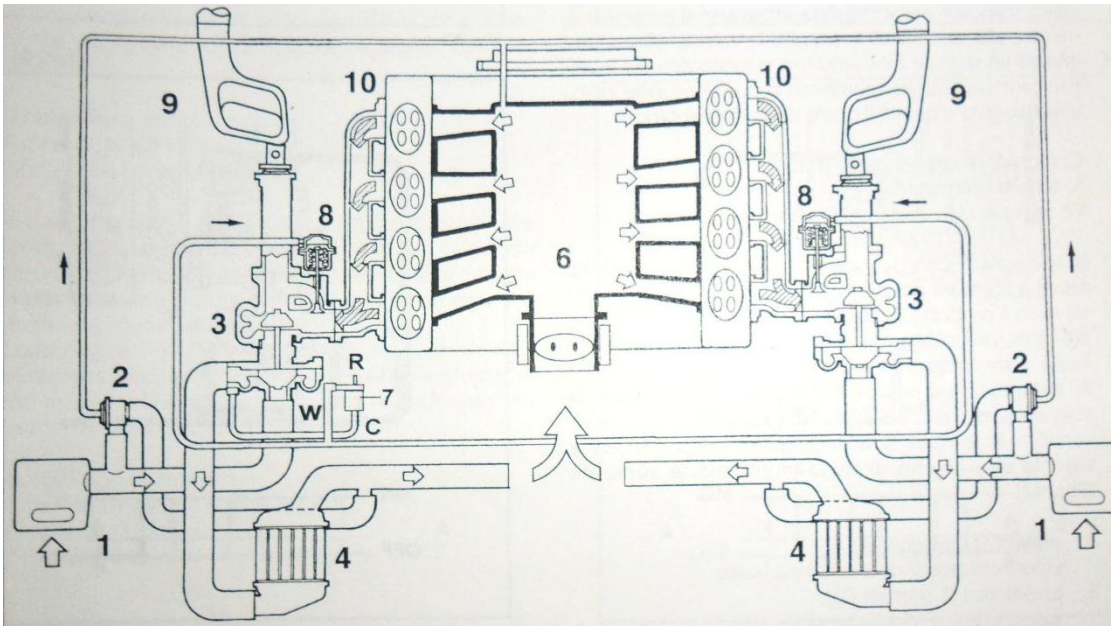


Figura 3.7: Schema circuito aspirazione e sovralimentazione.

La struttura di un motore turbo sovralimentato destinato a trazione stradale risulta abbastanza complicata perché il motore deve poter funzionare in un' ampia gamma di regimi, la curva della coppia deve essere decrescente con i giri favorendo così la guidabilità del mezzo ed avere una buona risposta nei transitori. Per soddisfare queste richieste in primo luogo i collettori di scarico devono avere volume relativamente piccolo, in modo da avere una buona efficienza nella trasmissione degli impulsi di energia alla turbina. In secondo luogo, la turbina e il compressore sono dimensionati in modo da fornire il più alto grado di sovralimentazione accettabile, già in corrispondenza del 40% - 50% della massima velocità del motore, per avere buone curve di coppia ai bassi regimi. Per evitare di ottenere pressioni di sovralimentazione troppo elevate agli alti regimi, si provvede a scaricare, in queste condizioni, parte dei gas combusti prima dell''ingresso in turbina attraverso una valvola di by-pass così detta valvola Waste Gate.

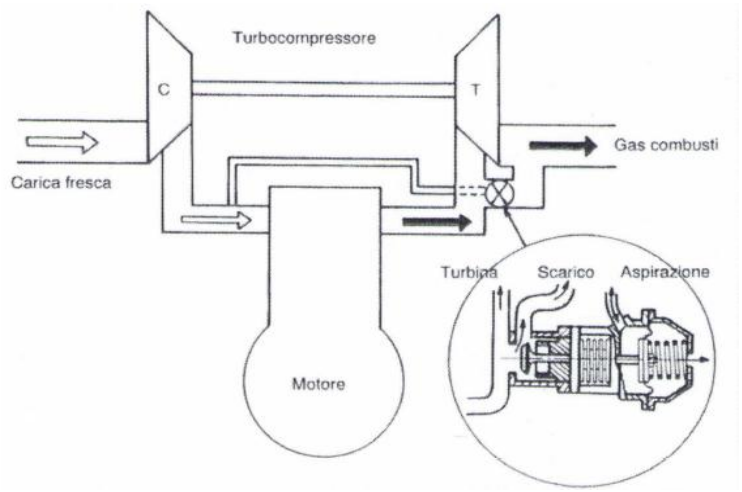


Figura 3.8: Schema valvola Waste-Gate

Lo scarico dei gas combusti ha anche l'effetto di diminuire la contropressione al motore nelle condizioni di pieno carico,

diminuendo così anche il lavoro del pistone nell' espulsione dei gas dal cilindro. Inoltre permette di utilizzare un turbocompressore con la turbina più piccola, con conseguente minor inerzia e maggior prontezza di risposta nei transitori (lo svantaggio è quello di aumentare il consumo specifico di combustibile agli alti regimi, a causa delle forti contropressioni allo scarico del motore dovute alla presenza di una turbina di piccola area). Questa soluzione presenta qualche problema costruttivo in più relativo alla valvola (specialmente per il suo gruppo di comando), dovuto all'alta temperatura dei gas di scarico e aggressività chimica. Normalmente, la valvola e il suo attuatore sono integrati con la turbina ed eventualmente raffreddati da un flusso d'aria fresca prelevata dalla linea di controllo della pressione di sovralimentazione.
La valvola pop-off (1), comandata dal segnale pneumatico (A) proveniente dal collettore di aspirazione, ha il compito di cortocircuitare l'aria (B) in uscita dall'intercooler (2) portandola a valle del filtro (C) quando viene bruscamente rilasciato l'acceleratore dopo una forte accelerata (cambio marcia con guida sportiva.

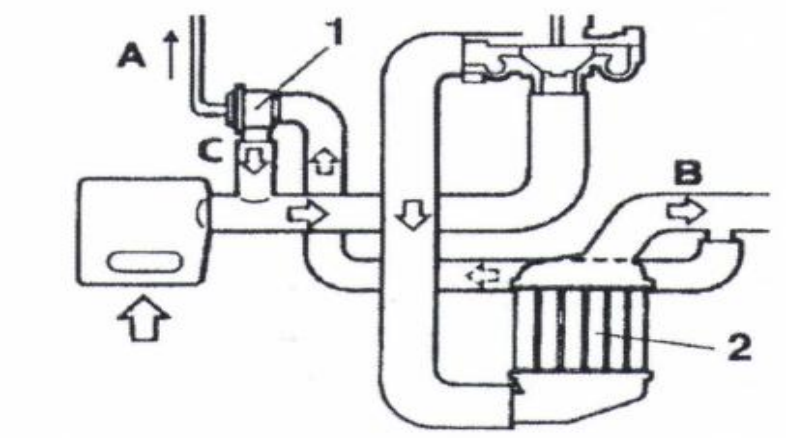


Figura 3.9: Valvola Pop-off e intecooler.

In tali condizioni infatti la depressione creatasi nel collettore, provoca l'apertura della valvola consentendo il bypass del turbo gruppo: in questo modo la girante del compressore non viene rallentata dalla presenza di una forte contropressione causata dalla farfalla chiusa, ed è quindi pronta a fornire la pressione alla successiva riapertura della farfalla.

Gli scambiatori di calore aria-aria (2) interposti tra il compressore e il collettore di aspirazione hanno il compito di raffreddare l'aria inviata ai cilindri che in seguito alla compressione ha aumentato notevolmente la sua temperatura. I vantaggi che si conseguono con l' inter-refrigerazione possono essere così sintetizzati:

Figura 3.10: scambiatore di calore (Intercooler)bancata destra.

- aumenta la massa di aria introdotta per ciclo nel cilindro e quindi la potenza fornita dal motore. Infatti, l'aumento della massa volumica all'uscita dal compressore è sempre minore dell'incremento di pressione e solo per una compressione isoterma i due andamenti sono uguali;
- si riducono tutti i livelli di temperatura raggiunti durante il ciclo, abbassando così i carichi termici sul motore e sulle palette della turbina;
- si migliora il rendimento organico (e quindi il consumo specifico di combustibile), perché si incrementa la potenza resa senza variare sensibilmente i livelli di pressione;
- si riduce il pericolo di detonazione che costituisce l' attuale limite per la sovralimentazione di questo tipo di motore.

3.3 Centralina controllo motore Magneti Marelli IAW 4CM

L'impianto Magneti Marelli IAW 4CM che equipaggia il motore 3200 V8 - 32v appartiene alla categoria degli impianti che integrano un sistema di accensione elettronica statica ad anticipo e distribuzione statica con un sistema di iniezione elettronica indiretta di combustibile di tipo intermittente multiplo fasato.

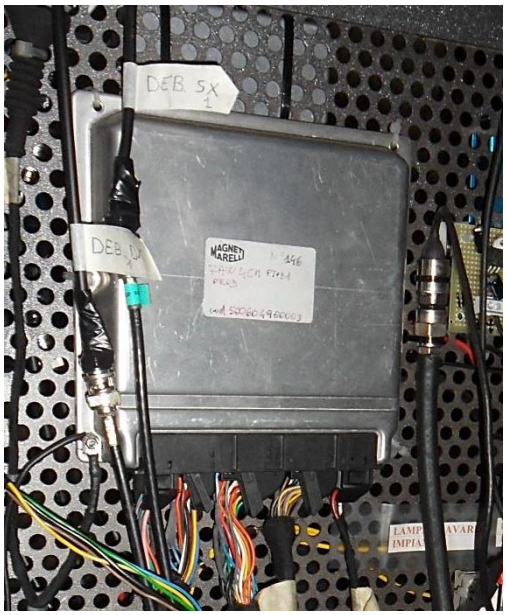


Figura 3.12: centralina motore Magneti Marelli IAW 4CM.

L'impianto può essere schematizzato nei seguenti sottosistemi:

- Circuito elettrico/elettronico
- Circuito aspirazione e alimentazione
- Circuito alimentazione combustibile
- Dispositivi per il controllo delle emissioni

La centralina di serie è in grado di acquisire i seguenti parametri:

- Il regime di rotazione del motore;
- La corretta sequenza del PMS di combustione dei cilindri (fasatura dell'iniezione);

- La pressione assoluta a monte della farfalla;
- La posizione pedale acceleratore;
- La posizione della farfalla;
- La temperatura aria aspirata;
- La temperatura del liquido refrigerante motore;
- La temperatura interna del precatalizzatore;
- Il titolo effettivo di miscela (sonda HEGO);
- L'eventuale presenza di detonazione;
- La velocità del veicolo;
- L'eventuale azionamento pedale freni;
- La tensione di batteria;
- L'eventuale inserimento del condizionatore;
- I segnali provenienti da altri sistemi linea CAN (cambio automatico CAE, sistema antipattinamento ASR, sistema antibloccaggio ABS).

Queste informazioni, quando sono di tipo analogico, vengono convertite in segnali digitali dai convertitori analogico/digitali (A/D) per poter essere utilizzate dalla centralina. All'interno della memoria della centralina è presente il programma software di gestione, composto da una serie di strategie, ciascuna delle quali gestisce una ben precisa funzione di controllo del sistema. Tramite l'utilizzo delle informazioni (input) prima elencate ciascuna strategia elabora una serie di parametri basandosi sulle mappe di dati memorizzati in apposite aree della centralina, e successivamente comanda gli attuatori (output) del sistema, che sono costituiti dai dispositivi che permettono al motore di funzionare, quali:

- Elettroiniettori;
- Bobine d'accensione;
- Attuatori ed elettrovalvole di attuazione;
- Teleruttori di comando;

- Interfaccia con il veicolo (spia diagnosi, spia temperatura catalizzatore, contagiri, strumento diagnosi);
- Interfaccia con altri sistemi (linea CAN).

3.3.1 Strategie di gestione dell'impianto

Un qualsiasi punto di funzionamento del motore viene individuato da due parametri (sistema Sped-density):

- Regime di rotazione, misurato direttamente dal relativo sensore (sensore ruota fonica);
- Carico motore, determinato in modo indiretto tramite il parametro densità ρ in funzione della pressione assoluta nel collettore di aspirazione e della temperatura dell' aria a monte della farfalla.

Noti tali parametri, attraverso opportune elaborazioni è possibile calcolare e successivamente attuare l'iniezione (quantità di combustibile erogato e relativa fasatura rispetto al PMS di scoppio), l'accensione (anticipo di accensione corretto), la sovralimentazione (valore della pressione di sovralimentazione) ed eventuali altre funzioni per ciascun punto di funzionamento del motore. In sede di sperimentazione vengono quindi realizzate delle mappature specifiche, nelle quali, per un certo numero di coppie di parametri regime-carico, vengono in pratica memorizzati i valori di tempo/fase di iniezione, di anticipo di accensione e di pressione di sovralimentazione necessari per il corretto funzionamento del motore. I valori del tempo di iniezione così determinati vengono inoltre corretti in funzione del segnale proveniente dalla sonda lambda la quale, in base a opportune strategie di funzionamento, determina una continua oscillazione del titolo di miscela intorno al valore stechiometrico. Il sistema è quindi definito del tipo "speed–density–lambda", in quanto il tempo di iniezione viene determinato fondamentalmente da

questi tre parametri. Tutte le situazioni di funzionamento che richiedono particolari adattamenti dei valori di tempo/fase di iniezione, anticipo di accensione e pressione di sovralimentazione calcolati vengono gestite dalla centralina controllo motore correggendo i valori di base calcolati tramite opportune strategie in funzione dei segnali provenienti dai vari sensori del sistema.

3.3.2 Gestione quadro segnali

All'atto dell' avviamento, la centralina procede al riconoscimento della fasatura dell'iniezione e dell'accensione che sono fondamentali per il successivo funzionamento di tutte le strategie. Tale riconoscimento viene effettuato in base all'interpretazione della successione di segnali provenienti dal sensore ruota fonica (a riluttanza variabile VRS), posta sull'albero motore, e dal sensore fase motore (ad effetto Hall), posto sull'albero di distribuzione. Sulla ruota fonica sono ricavati sessanta denti, due dei quali vengono asportati per creare una discontinuità (fonica tipo 60-2): l'angolo fra due denti consecutivi vale quindi 6°, mentre l'angolo relativo ai due denti mancanti vale 12° in più rispetto all'intervallo fra due denti consecutivi. Il segnale è di tipo sinusoidale, e deve essere trasformato in un segnale digitale tramite un convertitore A/D interno alla centralina.

(a) (b)

Figura 3.13:(a) ruota fonica. (b) dettaglio cava.

Sull'albero di distribuzione è posto un disco il cui profilo è ribassato per un arco pari a metà circonferenza. Il segnale proveniente dal sensore di fase è costituito da un'onda quadra direttamente interpretabile dalla centralina. Il PMS cilindro 1/7 corrisponde al passaggio del 5° dente dopo i due denti mancanti: se in tale situazione il segnale di fase è alto il cilindro 1 è in fase di combustione e il cilindro 7 in fase di incrocio, viceversa se il segnale di fase è basso. La successione dei cilindri è naturalmente quella relativa all'ordine di combustione del motore.

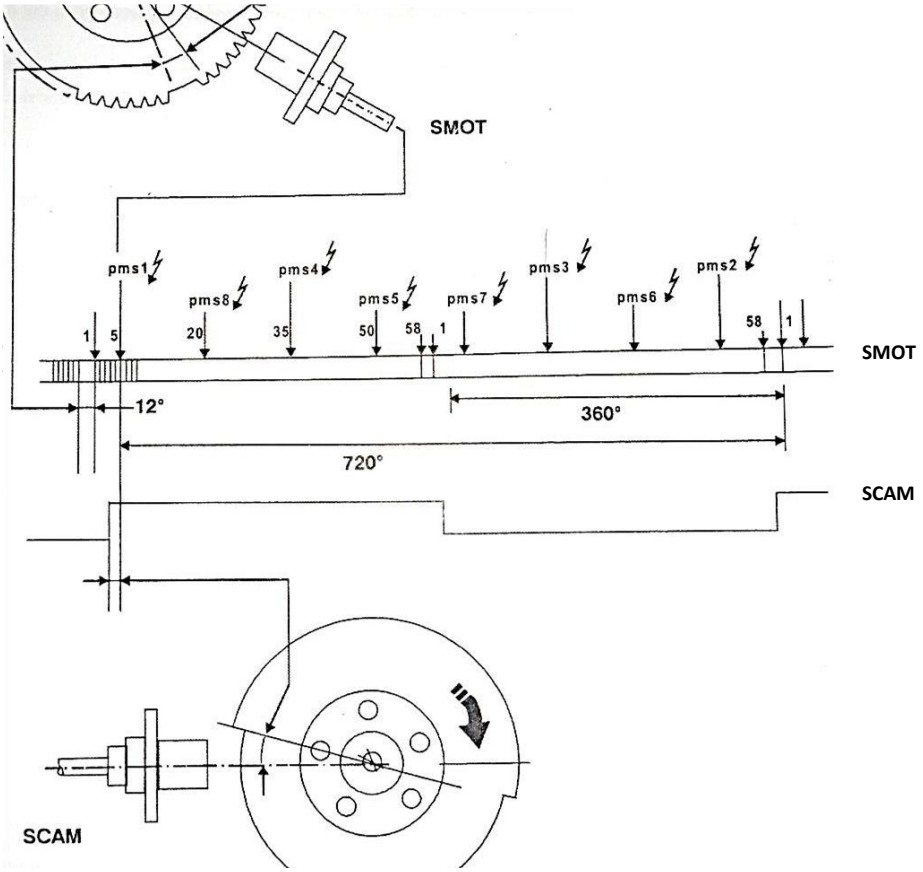


Figura 3.14: Schema fasatura dal manuale motore del Maserati. SMOT rappresenta il segnale dal sensore di giri albero motore (ruota fonica a 60 denti con cava); SCAM è il segnale dell'albero di distribuzione. Ordine di scoppio 1-8-4-5-7-3-6-2.

3.3.3 Gestione dell’iniezione e dell’accensione

Le strategie di gestione dell' iniezione hanno l'obiettivo di fornire al motore la quantità di combustibile corretta ed all'istante voluto in funzione delle condizioni di funzionamento del motore. La gestione dell'iniezione consiste essenzialmente nel calcolo del tempo di iniezione, nella determinazione della fase di iniezione e nella successiva attuazione tramite comando dell' iniettore. Il tempo di iniezione 'base' viene calcolato tramite interpolazione matematica della mappatura regime carico, e dipendono anche dalle caratteristiche dell' iniettore.. Essendo costante il differenziale di pressione tra interno ed esterno dell' iniettore tramite un apposito regolatore, la quantità di combustibile erogata a parità di tensione elettrica di alimentazione dipende solo dal tempo di apertura stabilito da centralina.
La gestione dell' accensione consiste sostanzialmente nella determinazione dell' anticipo di accensione voluto, in funzione delle condizioni di funzionamento del motore, e della sua attuazione tramite pilotaggio del transistor di potenza che si trova all'interno della centralina. Il primario di ciascuna bobina è alimentato dalla tensione di batteria tramite il teleruttore, ed è collegato al collettore del transistor di potenza incorporato nella centralina che invia la tensione di pilotaggio. Lo scopo dello stadio di potenza dell' accensione è di ottenere ai capi dell' avvolgimento secondario della bobina d'accensione una differenza di potenziale tale da generare un arco voltaico (la scintilla d'accensione) dotato di energia sufficiente ad accendere la miscela aria-benzina presente in camera di combustione. L'alta tensione è fornita da otto bobine singole montate direttamente sulle candele (bobine pencil-coil). L'anticipo ottimale di accensione viene calcolato dalla centralina in funzione del regime e del carico motore.

3.4 Proprietà del controllore

I requisiti prioritari che deve possedere il controllore sono naturalmente legati alle attuazioni che il sistema deve generare, in particolare al tempo di iniezione (Tj) e alla precisione angolare sull'anticipo dell'accensione. Nei moderni motori a ciclo Otto o Diesel, la pressione del carburante nel rail viene mantenuta pressoché costante, per cui l'unico parametro che influisce sulla quantità di combustibile iniettata è il tempo che l'iniettore rimane aperto. Solitamente la caratteristica dell'iniettore è lineare a meno di un offset iniziale, per cui, dati i mg di combustibile da iniettare, è facile determinare il Tj. La precisione sul tempo di iniezione diventa così fondamentale. Meno importante nei motori a ciclo Otto a iniezione nel collettore per l'iniezione risulta quindi la precisione angolare, che diventa invece fondamentale per i moderni motori Diesel (vedi multijet), dove le iniettate multiple vengono effettuate in precise posizioni angolari dell'albero motore. Nei motori ad accensione comandata e a iniezione indiretta l'iniettata dovrebbe essere effettuata il più tardi possibile per poter disporre dei dati aggiornati per il calcolo del Tj, ma anche abbastanza presto da favorire una migliore miscelazione tra aria e combustibile: il giusto compromesso viene scelto in base alle emissioni inquinanti ed alla prontezza di funzionamento, ma la precisione di posizionamento non è un parametro di elevata influenza. Per l'accensione la posizione angolare assume un' importanza fondamentale: l'anticipo viene calcolato in gradi rispetto al punto morto superiore del cilindro interessato dall'attuazione. La durata, circa 4 millisecondi, non è un parametro fondamentale in quanto le induttanze tendono naturalmente a saturarsi.

3.4.1 I segnali trattati

Il controllore deve poter leggere in ingresso i segnali provenienti dal motore quali:

- pressione aria collettore;
- pressione nel rail (collettore alimentazione combustibile);
- pressione in camera di combustione;
- temperatura aria collettore;
- temperatura acqua di raffreddamento;
- temperatura gas di scarico;
- tensione sonde λ (UEGO, Universal Exhaust Gas Oxygen sensor ed HEGO, Hysteresys Exhaust Gas Oxygen sensor);
- posizione valvola a farfalla;
- giri motore;
- accelerometri per diagnosi detonazione;
- ruota fonica;
- fase;
- tensione batteria;

e acquisire tutti i segnali necessari per l'implementazione delle strategie. Inoltre deve poter generare i comandi che gestiscono gli attuatori del motore, tra cui:

- comando iniettori (uno per cilindro);
- carica bobine di accensione (uno per bobina);
- controllo valvola ricircolo dei gas di scarico (E.G.R.);
- apertura valvola a farfalla (Drive by Wire);
- comando Waste Gate.

I segnali fondamentali su cui si costruisce una corretta gestione del motore sono il segnale di ruota fonica e quello di fase. Sulla ruota fonica, calettata direttamente sull'albero motore, è affacciato un pickup magnetico: al passaggio del dente, cambiando il traferro, cambia il campo magnetico concatenato

con le spire. La variazione genera un segnale circa sinusoidale la cui frequenza e ampiezza è proporzionale alla velocità di rotazione del motore. La mancanza di due denti sulla ruota crea una irregolarità nel segnale che consente alla centralina di riconoscere un riferimento ciclico e di azzerare il contatore denti. Per riconoscere la fase è presente invece un sensore ad effetto Hall affacciato alla ruota dentata che genera una onda quadra di periodo pari ad un ciclo motore e duty-cycle 50%. Entrambi i segnali vengono trattati prima di essere letti dal i tema: la fonica viene squadrata e, insieme alla fase, viene portata in un range di segnale compreso tra 0-5V.

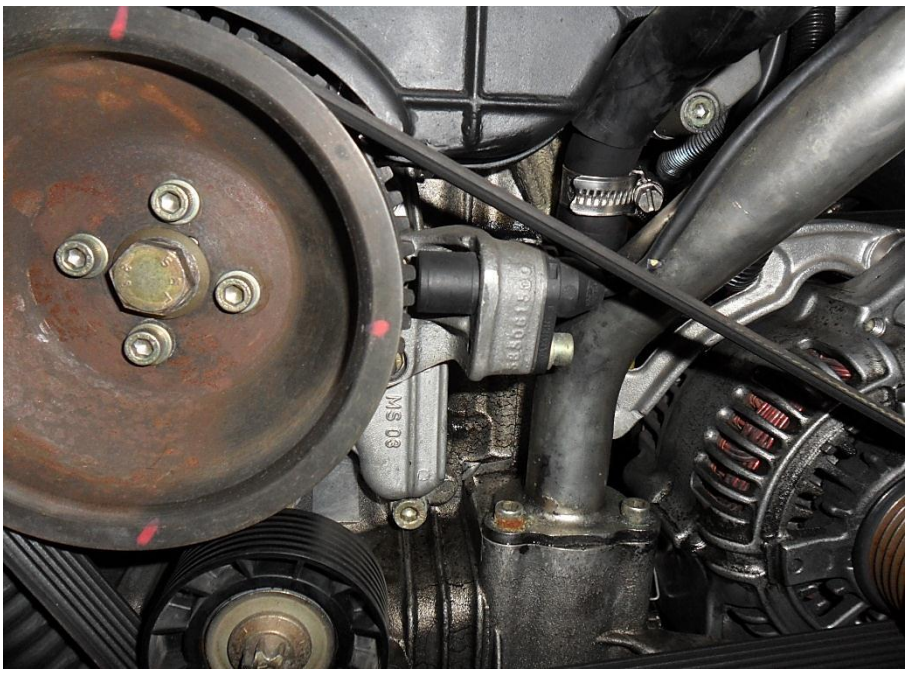


Figura 3.17: Ruota fonica con affacciato il sensore a induttanza magnetica.

3.4.2 Il comando di iniezione

Come detto in precedenza, la quantità di benzina iniettata è proporzionale al tempo di apertura dell'iniettore. Il comando,

elaborato dal calcolatore tramite un modello matematico (parte di alto livello), viene inviato alla parte di basso livello che si occupa di tradurlo e di generare un segnale all'istante desiderato e della durata voluta. Esso contiene le seguenti informazioni:

- Zj, dente di inizio iniezione;
- T0j, ritardo inizio iniezione;
- Tj, tempo di iniezione.

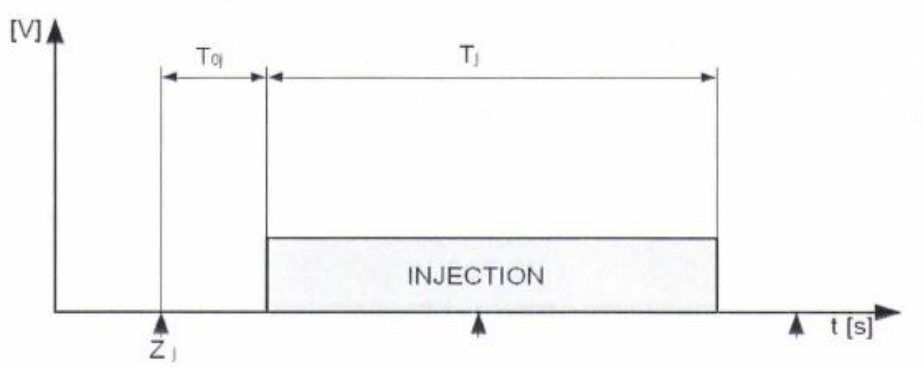


Figura 3.17: Il segnale di iniezione.

Il sistema in questo modo attende il passaggio del dente Zj, carica in un timer il tempo di attesa T0j, in un altro il tempo Tj che indica la durata dell' attuazione. In questo modo è possibile effettuare sia iniettate fasate che full-group. Il tempo di attesa T0j è stato introdotto per ottenere una precisione angolare sull'istante di inizio iniezione, altrimenti vincolata all'angolo compreso tra due denti (360/60 = 6°).

3.4.3 Il comando di accensione

A differenza dell'iniezione, per l'accensione la precisione angolare diventa molto importante, per via dell'anticipo, ovvero della posizione di scarica bobina calcolata rispetto al punto morto superiore attivo. Infatti, utilizzando un angolo, si aggira il problema dell'obsolescenza dei dati che nascerebbe qualora si utilizzasse un tempo, a causa del calcolo della velocità motore

che dovrebbe avvenire in precedenza. Nel caso di regime costante, l'errore non si verificherebbe, mentre nei transitori assumerebbe il valore anche di qualche grado. Passando invece all'FPGA un angolo, la conversione in tempo viene eseguita solo un istante prima dell'attuazione disponendo così di un dato di velocità aggiornato.

I parametri che così descrivono il comando sono i seguenti:

- Zic, dente di riferimento inizio carica;
- Pic, posizione angolare di inizio carica;
- Zfc, dente di riferimento fine carica;
- Pfc, posizione angolare di fine carica.

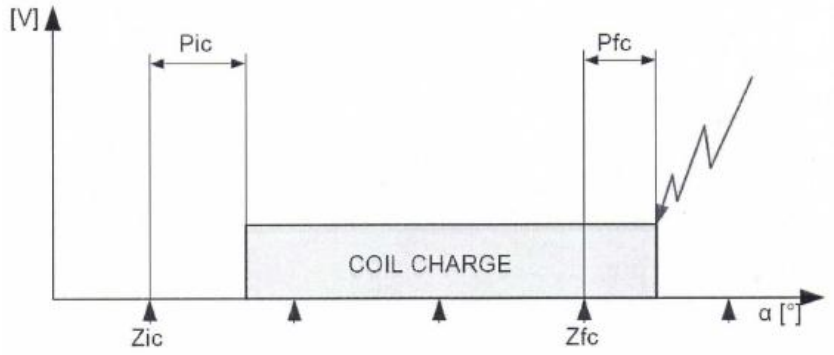


Figura 3.18: Il segnale di accensione (carica bobina).

Per una attuazione completa di accensione servono due comandi, uno di carica bobina, l'altro di scarica bobina, formalmente uguali. La carica comincia a Zic+Pic, ovvero si attende il dente di riferimento e, in seguito, si aspetta l'angolo corrispondente all' istante desiderato; la scarica avviene alla stessa maniera, l'unico parametro che si modifica è il comando di accensione, che da TRUE (inteso come 'ON') viene portato a FALSE (inteso come 'OFF'). La presenza di un blocchetto di accensione per ogni cilindro permette di far scoccare la scintilla in più cilindri anche contemporaneamente o in tempi molto

ravvicinati, come accade nei motori motociclistici da competizione con architettura denominata Big Bang (Ronda) o Twin Pulse (Ducati). L'angolo di attesa, Pic o Pfc, è espresso come frazione dell'arco tra due denti (6° nel caso di una ruota fonica a 60 denti): dato che si utilizzano 10 bit per tale parametro, si ottiene una precisione di 1/1024esimo di tale arco angolare, cioè di 0.00586°.

3.5 Controlli Drive by wire

Uno dei principali trend di sviluppo tecnologico nel campo dei veicoli civili e industriali è rappresentato dall'utilizzo di sistemi totalmente elettrici per la trasmissione e l'attuazione dei comandi di guida (di sterzatura, di frenatura, ecc.) in sostituzione dei tradizionali apparati meccanici e/o idraulici. Questi sistemi sono chiamati "drive-by-wire", letteralmente "guida con fili", poiché come principio prevedono l'ausilio di centraline elettroniche (e, dunque, di cavi) per rimuovere il collegamento meccanico e/o idraulico tra le parti che guidano il mezzo (volante, acceleratore, freno) e le parti che fisicamente eseguono il comando; è quindi un apparato composito in cui dispositivi elettrici eterogenei (trasduttori, attuatori elettromeccanici, unità elettroniche, reti di comunicazione) cooperano al fine di eseguire i comandi di guida impartiti dal conducente. L'introduzione di sistemi "drive-by-wire" nei veicoli permette di ottenere livelli di sicurezza di guida, efficienza di impiego e flessibilità di gestione mai raggiunti prima e rende operanti funzionalità innovative di gestione e di manutenzione dei veicoli. Oltre ad esercitare un notevole impatto sulla conduzione dei veicoli, i sistemi "drive-by-wire" sono destinati a rivoluzionare la progettazione strutturale e motoristica del settore powertrain, favorendo la progressiva transizione verso futuri sistemi di guida integrati con gli apparati di propulsione di

tipo ibrido e/o elettrico. Il drive by wire trova la sua prima applicazione automobilistica nel controllo della potenza erogata dal motore: innanzitutto non si controlla più direttamente la valvola a farfalla con un cavo in acciaio che la apre o chiude, ma attraverso un sistema indiretto che, collegato all'acceleratore, aziona un potenziometro; questo strumento a sua volta trasmette a una centralina elettronica l'informazione relativa alla richiesta di coppia (angolo di apertura del gas) veicolata dall'acceleratore, grazie a un calcolo di quanto il pedale è stato premuto. Questa informazione viene elaborata insieme a una serie di altri dati (quali per esempio la velocità relativa delle ruote, l'accelerazione trasversale e assiale cui è sottoposto il veicolo, l'angolo di sterzata, la temperatura esterna, i carichi sugli ammortizzatori, l'angolo di imbardata e rollio oltre a numerosi altri parametri) e ritrasmessa a un servomotore che provvede a far ruotare le farfalle del sistema di alimentazione in maniera tale da garantire il livello di coppia richiesto evitando perdite di aderenza dovute ad una eccessiva coppia applicata sulle mote motrici. In sostanza, la centralina elettronica risponde all'esigenza di un'erogazione di potenza ottimale, cercando di soddisfare la richiesta dell'utente attraverso l'acceleratore e i limiti fisici del veicolo nelle condizioni che si verificano in un determinato istante. Un'altra importante applicazione del drive by wire è quella relativa al controllo del turbocompressore attraverso la valvola waste gate, che in passato consisteva in una semplice valvola di massima pressione che si apriva oltre una certa soglia di pressione massima data dai gas di scarico in uscita dalla turbina, mentre nelle moderne vetture la sua apertura è gestita dalla centralina elettronica in funzione di determinati parametri, consentendo di regolare in modo continuo il grado di sovralimentazione del motore. Ovviamente, visto che il sistema rende sostanzialmente possibile intervenire direttamente sul comportamento dinamico del veicolo,

l'elettroattuazione dà la possibilità di gestire tutta una serie di variabili quali quelle del 'carattere' del veicolo stesso: su molte autovetture viene fornita di serie la possibilità di modificare le prestazioni del veicolo, variando per esempio la curva di coppia e potenza, la risposta elastica delle sospensioni, e la sensibilità dell'acceleratore, pur mantenendo la guida entro i margini di sicurezza.

3.5.1 Gestione del corpo farfallato motorizzato

Figura 3.19: Gruppo corpo farfallato.

Nel motore Maserati AM 585 è presente un corpo farfallato motorizzato 56 CFM 5. Consiste in una valvola che si apre e si chiude provocando una caduta di pressione nel condotto di alimentazione dell'aria e conseguentemente permette la regolazione della massa d'aria aspirata ad ogni ciclo e, quindi la quantità di combustibile utilizzato in quanto queste due quantità sono strettamente legate dal rapporto aria/combustibile A/F (in un motore benzina questo valore deve oscillare tra 12 e 15). Il servomotore che aziona la farfalla, un motorino passo-passo, è gestito dalla centralina attraverso comandi inviati in segnali tipo Pulse Width Modulation (PWM), una modulazione a larghezza di

impulso di tipo digitale in cui l'informazione è codificata sotto forma di durata nel tempo di ciascun impulso di un segnale.
La durata di ciascun impulso può essere espressa in rapporto al periodo tra due impulsi successivi, implicando il concetto di duty cycle: un duty cycle pari a 0% indica un impulso di durata nulla, quindi un'informazione di farfalla chiusa; un valore del 100% indica che l'impulso termina nel momento in cui inizia il successivo, quindi la farfalla sarà completamente aperta.

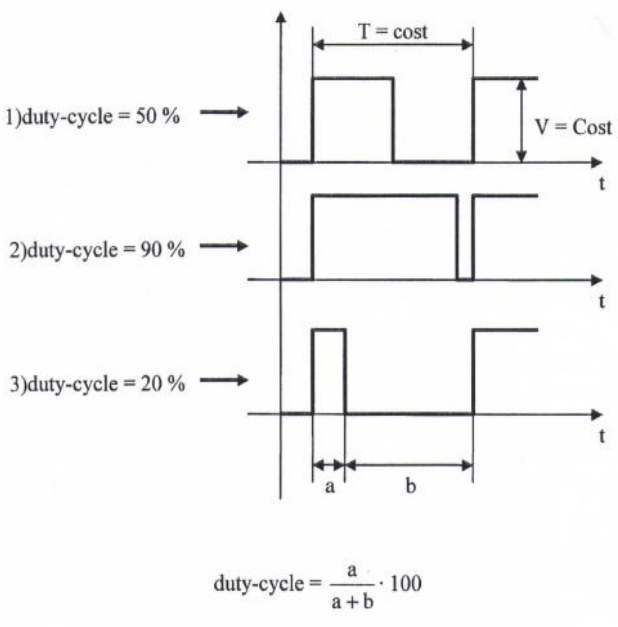


Figura 3.20: Esempi duty cycle.

L'utilizzo di due potenziometri per il controllo i retroazione della posizione dell' acceleratore, uno dei quali fornisce una tensione 0-5V in corrispondenza delle posizioni 0-100% di apertura rispettivamente, l'altro 5-0V in corrispondenza delle stesse posizioni, garantisce il funzionamento del propulsore anche in caso di guasto di uno dei due. L'utilizzo di un sistema di questo tipo consente di implementare un controllo in coppia del motore, svincolando la richiesta di potenza da parte del pilota dalle condizioni che si creano a valle del corpo farfallato. Nel

funzionamento con centralina di serie, l'informazione della posizione acceleratore viene interpretata dalla centralina stessa come una richiesta di coppia. In funzione di questa, la ECU ruota la farfalla di un angolo funzione della variazione di coppia richiesta, riuscendo così a stimare le condizioni di pressione che si creeranno nel collettore di aspirazione. In questo modo lo scostamento dal valore del titolo obiettivo sarà minore, con un conseguente miglioramento della gestione dei transitori.

Capitolo 4

Controllo attuazioni in FPGA

4.1 Organizzazione del programma di controllo

Nelle gestione di un motore a combustione interna vi sono diverse strategie che devono essere implementate, e non tutte presentano la stessa importanza o la stessa tempistica. Ad esempio, la parte di programma che deve rilevare la posizione angolare del motore ha una tempistica sicuramente più spinta di quella che, invece, deve elaborare le attuazioni di accensione e iniezione. Temporizzare tutto alla stessa maniera implicherebbe un notevole e inutile dispendio di risorse, sempre da centellinare in ottica di possibili applicazioni future. Spesso, inoltre, la flessibilità di una applicazione è inversamente proporzionale alla velocità con cui essa viene eseguita, per cui dovrebbero essere impiegate ulteriori risorse che verrebbero sottratte ad altre parti del programma. L'obiettivo nel programma di controllo è quindi stato prima di tutto stabilire quali applicazioni risultano tempisticamente critiche, poi, a seconda delle caratteristiche dei moduli di Lab VIEW (FPGA, Real Time), implementarle in una specifica parte del programma. Come spiegato nei precedenti capitoli, la VECU è composta essenzialmente di tre macro-parti di Virtual Instruments per la gestione che sono:

- FPGA VI;
- RT VI;
- Host VI;

ognuna corrispondente al relativo modulo di Lab VIEW. Le prime due sono strettamente necessarie al funzionamento del motore, mentre l'ultima fornisce l'interfaccia utente. Il RT VI è la parte del

programma che lavora associata al calcolatore in tempo reale della centralina e viene eseguita da un processore che può eseguire loop di calcolo alla frequenza di esecuzione di 1 MHz, ovvero è in grado di effettuare operazioni logiche semplici in 1 µs, anche se, come si vedrà, l'esecuzione dell' intero VI, che prevede plurimi cicli di calcolo non sempre elementari, occupa il processore per decine o centinaia di microsecondi, da considerarsi comunque un'ottima prestazione poiché decisamente lontana dai limiti imposti dal sistema di controllo per l'esecuzione della parte Real Time. Queste tempistiche sono invece insufficienti per la precisione richiesta dalla parte di attuazione: per questo motivo è stato necessario implementare parte della VECU in hardware, in modo che alcune funzioni della centralina possano essere eseguite in tempi notevolmente più brevi; di certo una parte hardware tradizionale sarebbe del tutto fuori luogo in una Virtual Engine Control Unit che si prefigge flessibilità e 'correggibilità' tra i principali obiettivi, per questo si utilizza una FPGA, cioè un circuito elettronico programmabile che unisce alla flessibilità di un software la velocità di un hardware. La procedura che permette di implementare di volta in volta sulla FPGA un diverso hardware di attuazione prende il nome di compilazione (attuata attraverso il *Compile Server* di *Lab VIEW FPGA Module*) che può durare anche più di un'ora, ma rimane comunque nettamente più rapida rispetto alla scrittura diretta in codice. Il *RT VI*, descritto dettagliatamente in seguito, costituisce la parte di alto livello del programma dove vengono eseguiti i calcoli che determinano le attuazioni per ogni cilindro per far scoccare al momento ottimale la scintilla nelle candele, caricando opportunamente le bobine ai tempi prestabiliti, e per far iniettare carburante nei tempi e nelle giuste quantità: infatti in questo VI sono implementate le mappe e le strategie di controllo (tra cui la retroazione delle sonde UEGO) che, in base a certi parametri,

permettono di ottenere gli output destinati agli attuatori. In pratica il RT VI è il cervello della VECU, la parte che si adatta alle varie condizioni di funzionamento prendendo di volta in volta adeguate misure in termini di quando e come effettuare le attuazioni poiché qui son contenuti tutti gli algoritmi che intervengono nel calcolo e soprattutto nella correzione delle attuazioni per far fronte a tutte le situazioni che si discostano da quella di motore a regime e carico costante: transitori di farfalla, avviamento a caldo e a freddo, compensazione di film fluido ecc. I calcoli vengono effettuati on demand, in base a degli *interrupt* hardware inviati dell' FPGA VI in punti predefiniti del ciclo motore che corrispondono a ben precise posizioni angolari in modo che i comandi generati dal RT VI siano inviati in tempo utile per essere eseguiti nel ciclo motore ma, soprattutto, basati su dati non troppo 'vecchi'. L' FPGA VI viene eseguito, appunto, su una scheda dotata di FPGA fisicamente contenuta nello chassis che incorpora l'elaboratore RT, e costituisce la parte a basso livello della VECU, nel senso che non esegue i calcoli che influiscono in modo diretto sul funzionamento del motore, ma monitora costantemente la posizione e la velocità angolare del motore con una elevata risoluzione, inviando opportunamente interrupt e le informazioni ricavate al RT VI, e assicurandosi che i comandi calcolati e ricevuti da quest'ultimo siano eseguiti in ordine e tempo utile. Il RT VI ignora totalmente la posizione angolare (quindi la posizione nel ciclo) del motore, e si limita a svolgere i calcoli per le attuazioni su richiesta inviando i comandi non appena son disponibili. L'HOST VI infine viene eseguito su un PC collegato sulla stessa rete su cui si trova l'elaboratore che esegue il RT VI e funge da interfaccia di comunicazione tra l'utente ed il sistema, fornendo informazioni sulle grandezze che possono essere interessanti e dando all'utente la possibilità di modificare in tempo reale certi parametri per le attuazioni sul motore. Allo stato attuale RT VI e HOST VI

sono integrati in un unico programma che va mandato in esecuzione da un PC Host sull' elaboratore Real Time tramite il modulo *Real Time* di *Lab VIEW* consentendo di visualizzare sull'Host le grandezze di monitoraggio.

4.2 I VI in FPGA

Il layout del programma in FPGA risulta complesso e la sua comprensione richiede una certa esperienza nell'applicazione di questo linguaggio. Si proverà a dare una spiegazione dell'architettura considerando che i vari blocchi sono a loro volta suddivisi in diversi subVI.
Le operazioni svolte dall'FPGA VI sono:

- determinare la posizione angolare del motore tramite il sensore di fase ruota fonica;
- determinare la fase sfruttando il segnale del sensore di fase;
- gestire le attuazioni di accensione come richiesto dal RT VI;
- gestire le attuazioni di iniezione come richiesto dal RT VI;
- comunicare al RT VI la posizione angolare motore;
- fornire al RT VI il regime istantaneo del motore;
- innescare nel RT VI in tempo utile il calcolo dei tempi e delle durate delle attuazioni;
- gestione allarme sicurezza bobine.

Il sistema deve così poter gestire un qualunque tipo di motore ad accensione comandata o spontanea, compresso o aspirato, con un numero qualsiasi di cilindri e a qualsiasi regime operativo con un errore di posizionamento inferiore a 0.1° angolari e con errori temporali inferiori al microsecondo. Devono inoltre essere possibili iniettate full-group o sequenziali, misfuel e misfire arbitrari.

4.2.1 Inizializzazione dei dati

Prima di poter elaborare tutte le applicazioni è necessario inizializzare alcuni dati e porte. Per questo motivo è stato implementato un blocco denominato *Init*, composto da un *Flat Sequence Structure* che viene eseguito prima di tutti gli altri ogni volta che si avvia il VI FPGA.

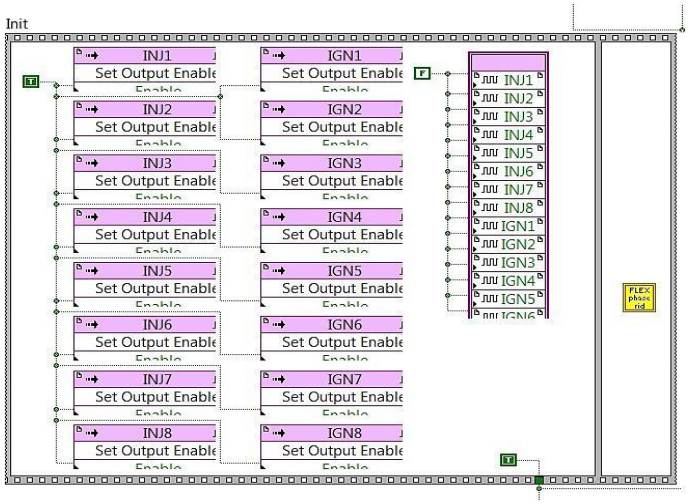


Figura 4.1: schema a blocchi *Edge Timer*.

Nel primo frame viene collegato il valore booleano *True* ad un *FPGA I/O Method Node* che permette di dedicare 8 linee digitali per l'iniezione e 8 per l'accensione alla scrittura di dati, vengono cioè impostate le linee in modo tale da poter essere utilizzate come comando. Nello stesso frame inoltre le linee digitali vengono impostate tutte a False per essere azzerate. Per essere sicuri che questo frame venga eseguito per primo colleghiamo una variabile booleana a tutti gli altri cicli *While* (che quindi attendono l'esecuzione del VI prima di partire). Nel blocco successivo invece viene dato avvio al blocco *Flex Phase 4* che è in pratica il blocco in cui vengono incrociati i segnali di fonica e fase in modo da fasare il sistema.

4.2.2 Il VI di fasatura: *Flex Phase*.vi

È stato cambiato il blocco di fasatura rispetto alla versione precedente, e parte del lavoro svolto è stato proprio quello di adattare tutti i moduli FPGA e RT al nuovo blocco di fasatura. Nel VI generale FPGA che chiameremo *Fmain* sono richiamate come *variabili globali* le variabili del blocco *Flex Phase* per essere successivamente implementate e riconosciute nel Real Time. Il richiamo di queste variabili è molto importante ed è stato dedicato parecchio tempo proprio all'implementazione di un sistema in grado di salvare questi parametri creando anche un file di caricamento dati predefinito, come vedremo in seguito. Ma entriamo nel dettaglio di questo VI.

Il blocco *Edge Timer* ha il compito di determinare la posizione angolare dell'albero motore ed è tempisticamente parlando il più critico dell'intera applicazione. Un ritardo o un errore nel rilevamento di questa grandezza non è accettabile in quanto comporterebbe un errato calcolo nei tempi delle attuazioni provocando un mal funzionamento del motore.

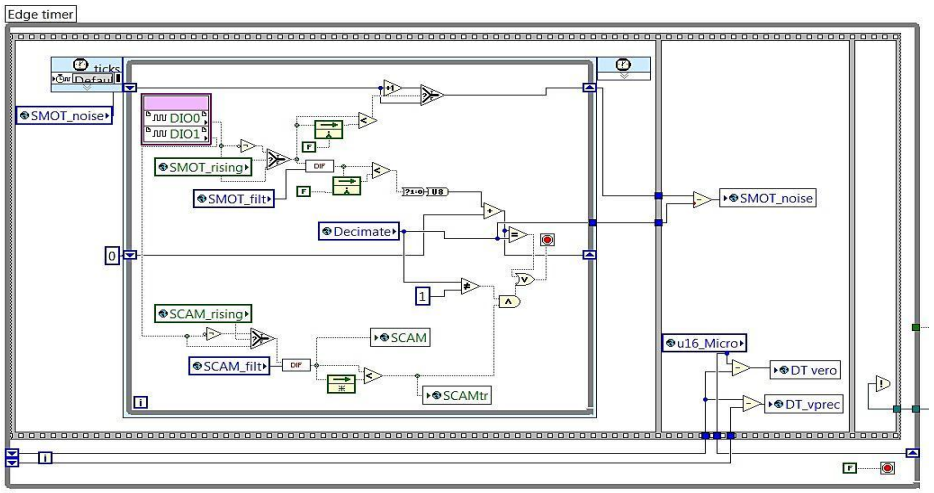


Figura 4.2: schema a blocchi *Edge Timer*.

All'interno di un *While loop* senza fine è posto un *Sequence loop* che nel primo *frame* un *Timed loop* che gira alla massima frequenza possibile in FPGA, ovvero 40 MHz in cui all'interno vengono monitorati sia il segnale di ruota fonica che il segnale dell'albero di distribuzione che sono caricati come i *DIO 0* e *DIO 1* rispettivamente attraverso il blocco rosa (che interfaccia il VI con il mondo esterno e i segnali in ingresso). I valori letti sono del tipo digitale e quindi del tipo vero o falso che identifichiamo come un ciclo alto-basso. La transizione del dente è riconosciuta dal passaggio da alto-basso oppure basso-alto a seconda del valore di *SMOT_rising* e *SCAM_rising* che identificano appunto quale delle due possibilità vengono riconosciute come passaggio dente e come cambio ciclo rispettivamente. Questo discorso vale in particolare per la ruota fonica a causa della presenza di uno squadratore che trasforma un segnale di andamento sinusoidale in un segnale digitale alto basso, che può portare a invertire il segnale.

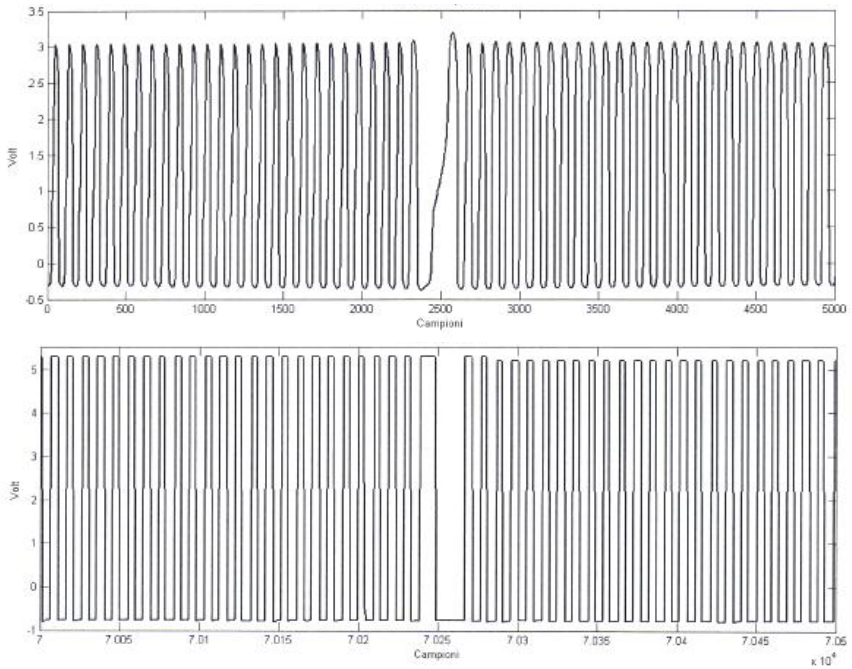


Figura 4.3: segnale di fonica prima e dopo la squadratura

Per evitare errori e false transazioni che possono essere provocate da disturbi del segnale in ingresso, il segnale passa all'interno di *subVI* chiamato *DIF* (acronimo di Digital Integral Filter) che verifica che il segnale monitorato (alto o basso) rimanga invariato per un certo numero di cicli (definiti dalla variabile *SMOT_filt* e *SCAM_filt* risettivamente) altrimenti viene riconosciuto come rumore e ignorato. La variabile *SMOT_noise* indica esattamente quante volte avviene una falsa transazione poi ignorata. Nel frame successivo si sfrutta la variabile globale che funge da contatore temporale in microsecondi *u32_Micro* e gli *Shift Register* (sono le frecce verso il basso che riportano il valore del ciclo n al ciclo n+1) per calcolare l'intervallo temporale tra il passaggio di due denti consecutivi, *DTvero*, e tenere in memoria il tempo passaggio dente precedente *DT_vprec.* Nel terzo frame della struttura *sequence* viene anche attivato un *occurence,* che manda un segnale che attiva parti del programma che sono in attesa a partire dal blocco *Main Counter.*
È il blocco che effettivamente aumenta il contatore denti ed estrapola i denti mancanti quando necessario. Ad ogni passaggio dente per prima cosa vengono aggiornate le variabili globali relative ai tempi dente e al contatore denti *SMOT_cnt* (frame 0).

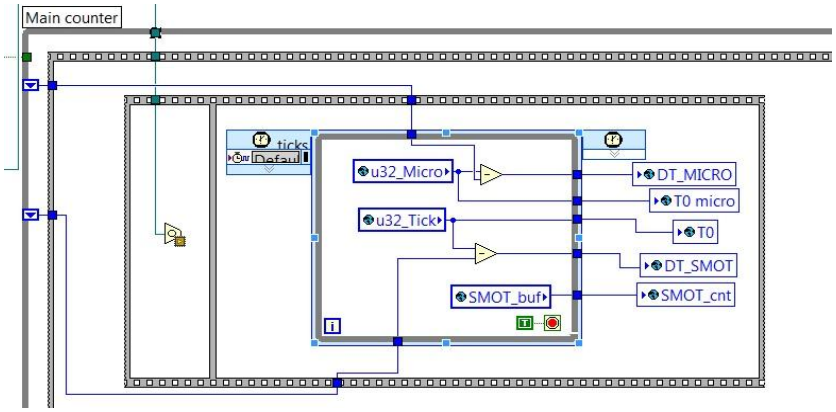


Figura 4.4: schema a blocchi *Main Counter frame 0*.

Nel frame successivo (*frame 1*) lo *SMOT_buf* viene confrontato con il numero di denti a cui iniziare l'estrapolazione *Z_extrap.* Quando non è necessario estrapolare i denti mancanti, il blocco si limita ad aumentare di uno il valore della variabile *SMOT_buf.* Si utilizza una sorta di *buffer* per evitare uno sfasamento poiché la variabile *SMOT_count* è presa come riferimento delle attuazioni e quindi non sarebbe accettabile.

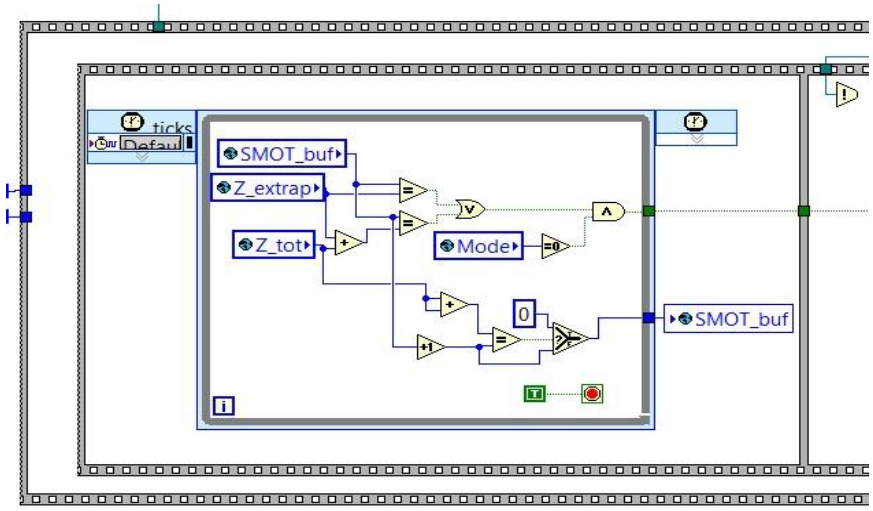


Figura 4.5: schema a blocchi *Main Counter frame 1*.

Nel caso in cui sia necessario estrapolare i denti mancanti (cioè quando *SMOT_buf* assume il valore definito da *Z_extrap*) si entra in un *For loop*, in cui per un numero uguale a *N_extrap*, che dipende dal motore e dal tipo di ruota fonica (una 60 denti con una cava di 2 ha *N_estrap*=3), si estrapola il tempo dente *DT_smot* e si aumenta di uno il contatore denti fino a quando arrivando a 119 (120° dente) viene azzerato. Cioè in pratica in questo blocco riusciamo a simulare, nonostante la presenza della cava, i due denti mancanti quando siamo, nel caso di ruota a 60 denti, al dente 56 e 116 (considerando il primo dente come il dente 0).

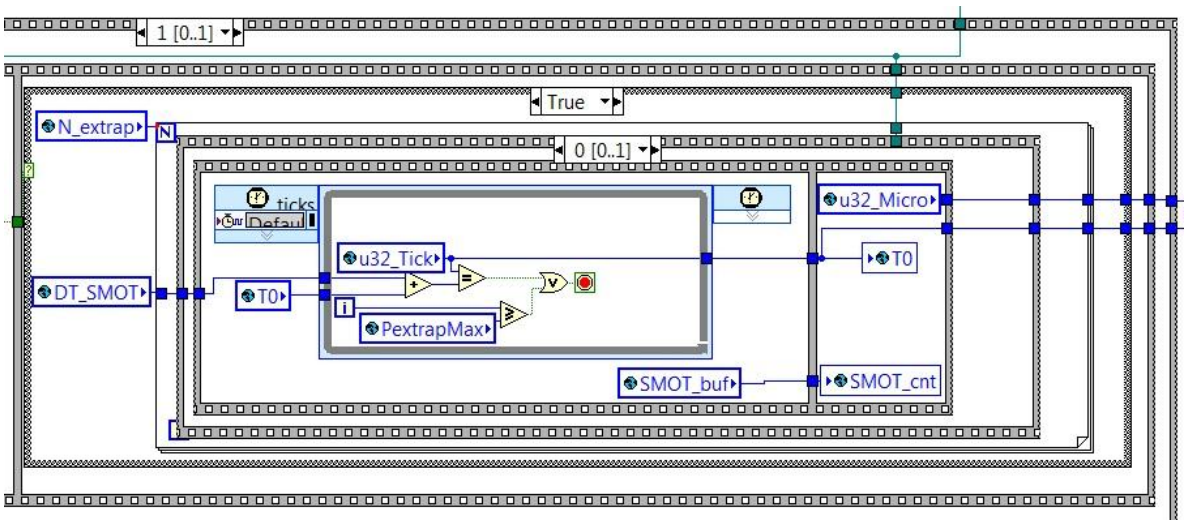


Figura 4.6: schema a blocchi *Main Counter frame 1.*

I blocchi per stabilire la fase del motore sono tre. Il blocco *Edge Timer* si occupa di leggere il segnale digitale dalla fonte *DIO 1* e di filtrarlo come descritto precedentemente e analogamente con il

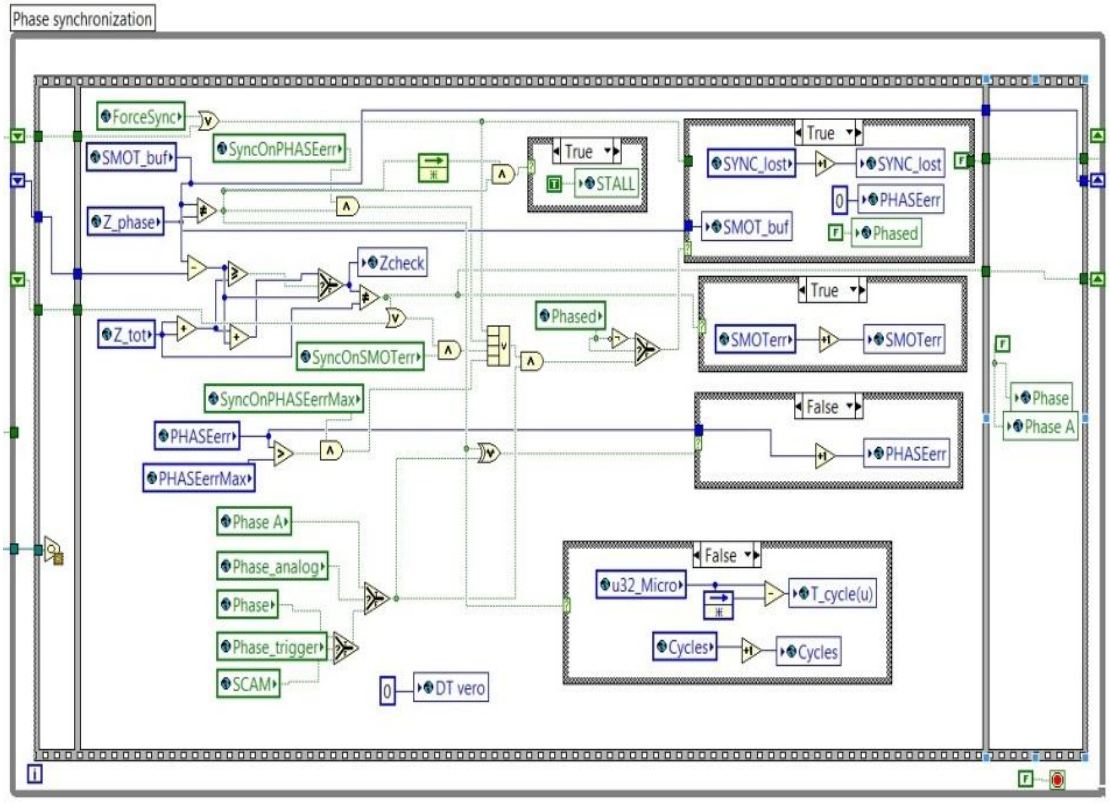


Figura 4.7: Phase Syncronization.

segnale di ruota fonica. Il blocco *Phase Syncronizazion* rileva invece la transizione della fase, processa l'avvenuta fasatura del motore nel momento in cui si ha contemporaneamente riconoscimento della cava e fase alta, e re-inizializza il contatore denti al valore *Z_phase* (solitamente uguale a 1). In seguito, l'accesso a questa parte del VI dovrebbe avvenire una volta per ciclo motore, ma non verrà alterato nulla in quanto *SMOT_buf* sarà già al valore 1. La variabile *Phase Trigger* consente di utilizzare come segnale di fase o il segnale proveniente dal sensore di fase *DIO 1*, oppure lo stesso segnale però filtrato *Phase*: solitamente si usa quest'ultimo per una maggiore sicurezza nell'evitare una falsa transazione di fase. Il blocco *Hole Control* ad ogni passaggio dente confronta il tempe dente precedente *DT_vprec* con l'ultimo dente misurato *DT vero*. Quando questo valore moltiplicato per 10 risulta maggiore del valore *DT_vprec* moltiplicato per la variabile *Hole_thershold* allora viene identificata la cava relativa ai due denti mancanti nella ruota fonica e viene riconosciuta la transazione di fase; il valore di soglia *Hole_Thresold* ha solitamente valore 20, la cava viene in pratica riconosciuta quando il tempo dente risulta essere il doppio di quello precedente.

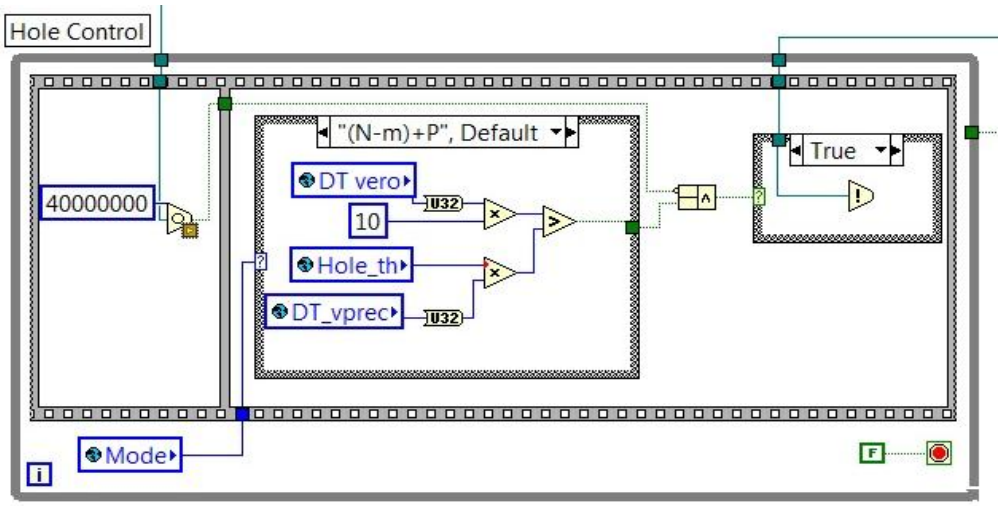


Figura 4.8: Hole control.

A questo punto viene lanciato un *occurence* al blocco *Phase syncronizazion*: nella condizione *False* del blocco non viene eseguita alcuna operazione, in quella *True* si pone *SMOT_buf* uguale a *Z_phase* e si azzera il *DT vero.* All'interno del loop viene riconosciuta l'avvenuta fasatura con la variabile *Phased* che si contrappone con la variabile *Stall* che indica la condizione di motore in stallo, inoltre vengono date informazioni riguardo al tempo di esecuzione del ciclo e al numero di cicli motore fino a quel momento effettuati.

4.2.3 La gestione degli interrupt di calcolo

Durante l'esecuzione del FPGA VI è necessario inviare dei flag al *RT VI* per avviare una sequenza di operazioni quali l'acquisizione delle grandezze del motore e il calcolo dei comandi di attuazione. L'esecuzione della routine di calcolo deve avvenire in funzione della posizione angolare dell'albero motore e senza eccessivi ritardi dal momento in cui viene effettuata la notifica, per cui non possono essere utilizzati dei semplici *soft IRQ* (variabile booleana che viene portata a vera quando necessario). Le librerie di Lab VIEW mettono a disposizione degli strumenti chiamati *IRQ (interrupt)* hardware, che possiedono una linea bus dedicata per la loro trasmissione. In tutto si hanno a disposizione 32 interrupt che sono trasmessi tramite un *array*. Ogni volta che si utilizza uno di questi, è necessario attribuire allo stesso un numero (variabile tra 0 e 31) per poi poter richiamare sul *RT VI* quello desiderato. Questi permettono di avere tempi di trasmissione del flag dell' ordine dei 50 μs, fornendo notevole affidabilità anche per quanto riguarda la sicurezza che questo sia ricevuto. All'interno del VI gli *IRQ* hardware vengono utilizzati per avviare la routine di calcolo delle attuazioni e acquisire le grandezze che forniscono informazioni sul funzionamento del propulsore. Nel primo caso

l'interrupt di calcolo deve essere inviato ogni qual volta deve essere effettuato il calcolo delle attuazioni per ciascun cilindro (quindi solitamente si impostano 8 interrupt di calcolo per un motore 8 cilindri, 4 per un 4 cilindri, ecc.). Nel secondo caso l'interrupt serve per avere dati sempre aggiornati sul valore delle grandezze, quali pressione collettore, temperatura aria aspirata, titolo (A/F) etc., e il numero di interrupt lanciati per effettuare l'aggiornamento parametri è uguale a quello degli interrupt per il calcolo.

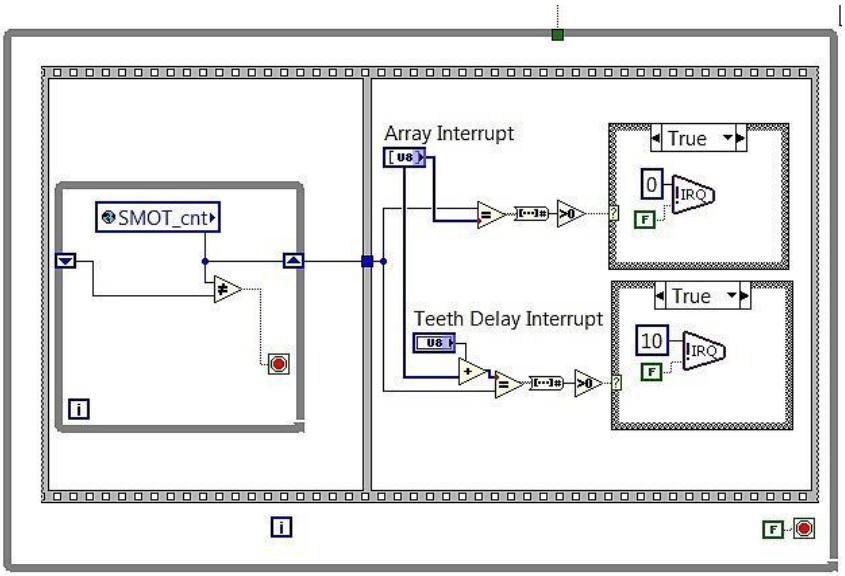


Figura 4.9: interrupt di calcolo.

Il blocco Interrupt consiste in un *While loop* senza fine al cui interno è contenuta una struttura *sequence* composta di due soli *frame*; nel primo, all'interno di un ulteriore *While loop*, viene verificato se il valore dello *SMOT_cnt* risulta diverso dal valore misurato al ciclo precedente, e solo quando ciò si verifica si passa al frame successivo; qui si va ad analizzare se il valore attuale di *SMOT_cnt* risulti uguale ad uno dei valori contenuti nell' *Array intenupt*, un vettore creato nel RT VI (più avanti verrà descritto come) composto da 8 elementi a cui corrispondono gli 8 denti (al massimo) a cui vanno lanciati gli *interrupt* di calcolo. La

stessa verifica viene fatta anche per gli *interrupt* da mandare per l'acquisizione e, quindi, l'aggiornamento dei parametri di controllo, con la differenza che il confronto viene fatto con lo stesso Array però aumentato di un valore *Teeth Delay Interrupt*, anch'esso imposto nel *RT VI*: in pratica si vogliono mandare gli Interrupt per l'aggiornamento parametri in numero uguale a quello degli *Interrupt* per il calcolo sfasandoli di una certa quantità di denti (magari a metà tra un interrupt di calcolo e il successivo) in modo da non far fare al processore contemporaneamente o quasi due loop, poiché ne sarebbero entrambi rallentati. Quando si verifica che il contatore denti è in corrispondenza di uno dei valori contenuti negli *Array*, in uscita dall' uguaglianza il vettore di booleani avrà certamente un *True* tra i suoi 8 elementi, quindi questo viene trasformato in una parola in formato binario che risulta essere maggiore di 0, condizione che fa entrare all'intero del *case structure True*, che consente l'invio dell'*interrupt*. All'icona che rappresenta l'*IRQ* è collegata una costante numerica che indica il numero dell'interrupt utilizzato (0 per quello di calcolo, 10 per quello di aggiornamento parametri) e una costante booleana che indica se il FPGA VI deve aspettare o meno la notifica di ricezione da parte del RT VI. Questa è posta a falsa in quanto questa operazione rallenta la trasmissione del flag, aspetto che può diventare critico agli alti regimi.

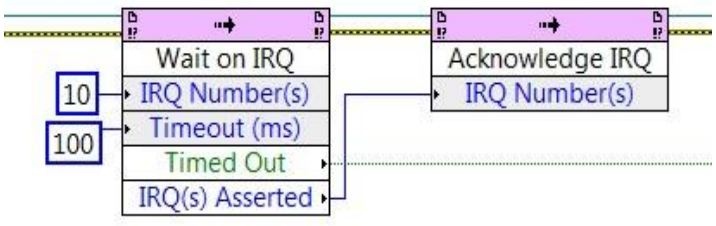


Figura 4.9: riconoscimento interrupt di calcolo in RT.

4.3 La gestione dell’accensione

Gran parte dell' FPGA VI è occupato visivamente, ma non solo, da una grande struttura sequence costituita da due soli frame in cui sono contenuti tutti i blocchi che servono a svolgere tutte le attuazioni di accensione e iniezione.
Il primo frame in realtà viene eseguito una sola volta quando viene avviata l'applicazione, e serve ad attendere, attraverso una variabile Ready, che siano pronte per essere eseguite le parole formulate ad alto livello; senza questo piccolo accorgimento, tutti i blocchi di attuazione contenuti nel *frame* successivo si troverebbero a leggere parole di comando composte di soli zeri, eseguendo le proprie funzioni di carica e scarica bobina o apertura e chiusura iniettore tutto al dente zero, in modo continuativo, senza sosta. I blocchi che hanno il compito di gestire l'accensione sono 8, cioè in numero pari al numero massimo di cilindri da gestire, e sono tutti uguali tra loro, ma ognuno di essi viene dedicato ad un solo cilindro: rispetto a versioni precedenti della VECU (dove il comando di accensione veniva eseguito da un solo blocco per tutti i cilindri). Questo implica certamente un’ occupazione maggiore della scheda FPGA, ma solo in tal modo è possibile gestire contemporaneamente l'accensione di più cilindri (prevista nei motori tipo Big Bang o Twin Pulse) con la garanzia di non aver ritardi e di non doversi più curare dei problemi che si verificavano quando i tempi in cui un'attuazione doveva essere effettuata erano già passati (e quindi il blocco che lo attendeva non procedeva perché rimaneva fermo in un *While loop* senza fine).

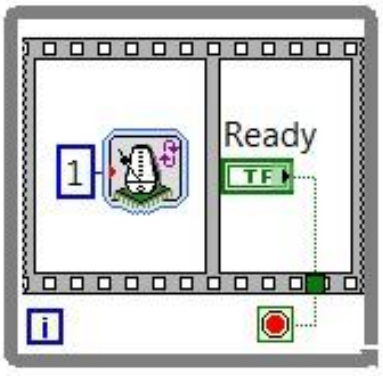


Figura 4.10: blocco ready.

Di seguito viene descritto il funzionamento dell' Igniton Timer 1, che però, come già detto, è uguale a quello di tutti gli altri 7 cilindri. Contenuto all'interno di un *While loop* senza fine, il blocco prevede 8 frame, da 0 a 7, in esecuzione ciclica. Ogni Frame è condizionato dalla variabile *Stall*, infatti in caso di stallo il sistema deve ripartire esattamente dove si è interrotto e non ripartire.

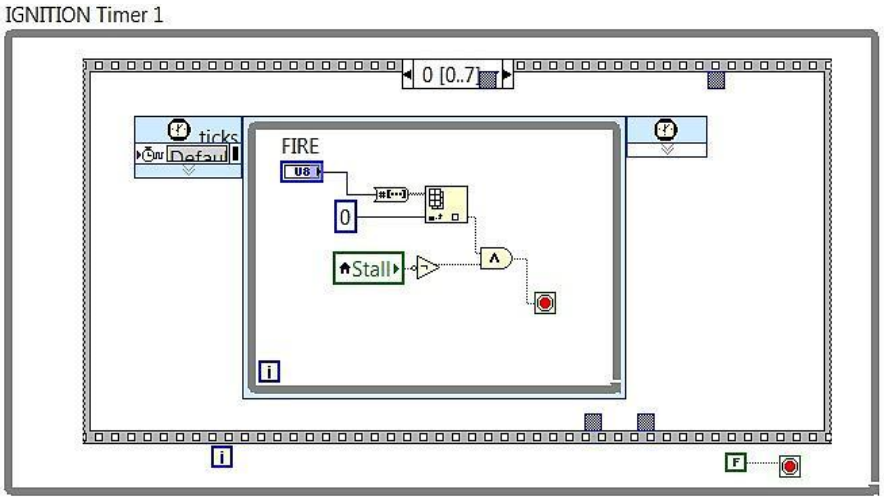


Figura 4.11: Ignition timer, frame 0.

Il frame 0 (figura 4.11) contiene un *Timed loop* temporizzato a 40MHz che serve a verificare se il blocco di accensione in esame relativo al particolare cilindro è destinato o meno ad essere utilizzato: infatti dal vettore di booleani FIRE creato in Real Time si ricava l'elemento relativo al cilindro (elemento 0 del vettore per il cilindro 1, elemento 1 per il cilindro 2, ecc.): se questo è True allora termina il While loop e il blocco procede al successivo frame, se è False rimane bloccato nel While loop. Questo permette all'utente di decidere quanti cilindri voler attuare: ad esempio, nel caso di un motore a 4 cilindri, il vettore di booleani FIRE sarà composto dai primi 4 valori ad alto livello e gli ultimi 4 a basso livello; oppure rende possibile creare strategie cicliche di Misfire (descritte in seguito nel RT VI) che possono servire per la diagnosi del sistema di controllo.

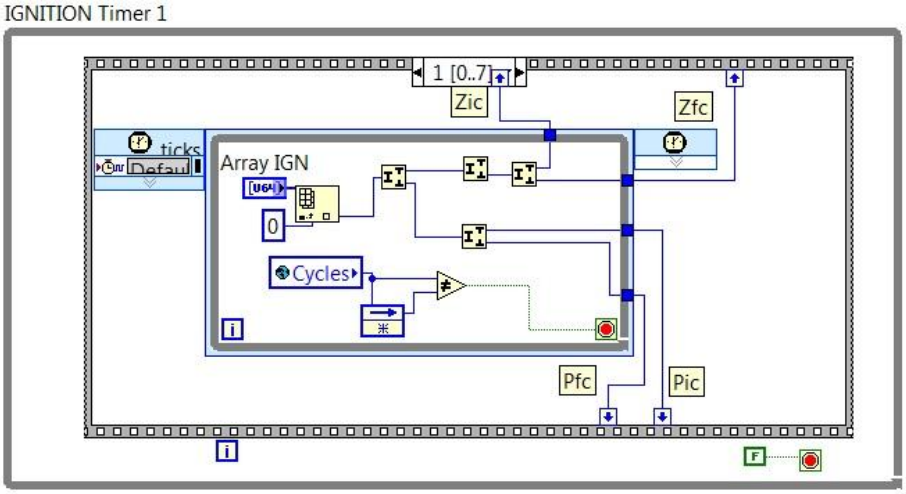


Figura 4.12: Ignition timer, frame 1.

Nel frame 1 (figura 4.12) ogni singola accensione va a leggere un certo elemento dell' *Array IGN,* vettore che è composto da 8 parole da 64 bit (1 per cilindro) contenenti le informazioni calcolate in *Real Time,* relative all'accensione. La parola da 64 bit è organizzata nel seguente modo:

- 8 bit rappresentano *Zic*, dente riferimento di inizio carica;
- 8 bit rappresentano *Zfc*, dente riferimento di fine carica;
- 16 bit rappresentano *Pic*, punto riferimento di inizio carica, inteso come ritardo in 1024 esimi di angolo dopo il dente di inizio carica;
- 16 bit rappresentano *Pfc*, punto riferimento di fine carica, rispetto al dente di fine carica.

Le informazioni ottenute vengono inviate nei frame successivi e sono subordinate dal termine del *Timed Loop* in cui sono contenute, questo ciclo è stato inserito per risolvere il problema riscontrato in fase di Crancking (avviamento) per cui se il termine dell'attuazione è contenuta all'interno del medesimo passaggio dente non avveniva l'attuazione. Si parla di regimi molto bassi (ordine dei 200 rpm) per cui è necessario forzare l'invio delle informazioni al cambio di ciclo attraverso il confronto della variabile *Cycles* rispetto a 2 cicli consecutivi.

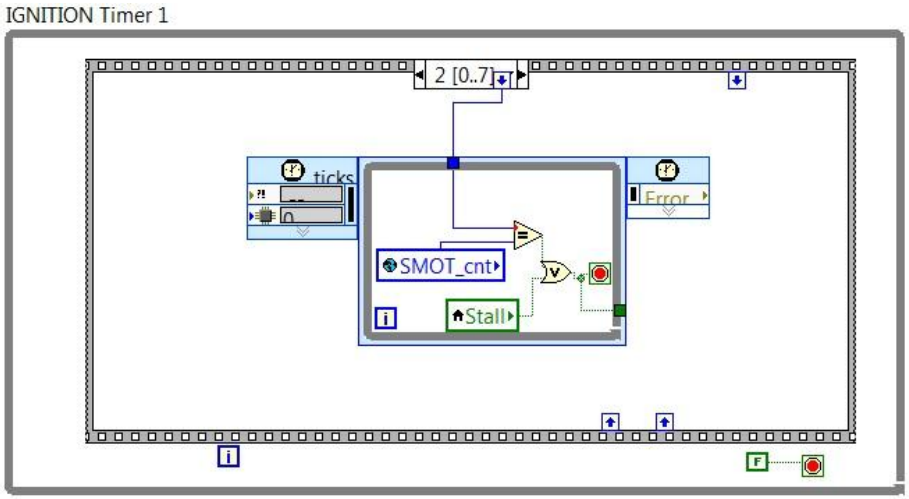


Figura 4.13: Ignition timer, frame 2.

Il *frame* 2 (figura 4.13) è composto da un *While loop* in cui riconosciuto il passaggio al dente di inizio iniezione si interrompe l'attesa e si passa al terzo frame, cioè quando il contatore è uguale a *Zic*.

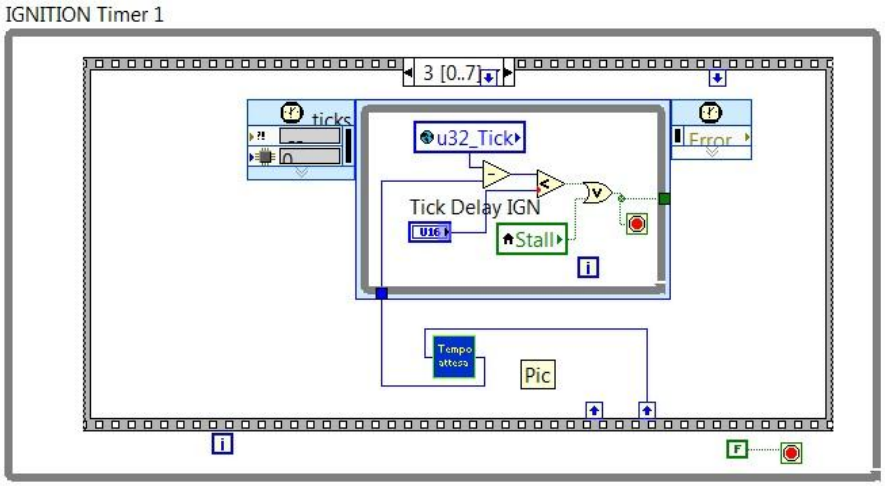


Figura 4.14: Ignition timer, frame 3.

Nel *Frame* 3 (figura 4.14) invece si converte le informazioni che entrano come frazioni di 1024 esimi di angolo in un tempo. È quindi necessario rapportare la frazione di angolo al tempo, contenuto nella variabile *DT_smot*, tenendo conto che l'FPGA

non è in grado di eseguire i calcoli con numeri decimali: questo sarebbe problematico poiché l'approssimazione apportata comporterebbe un errore percentuale troppo elevato per gli obbiettivi proposti.

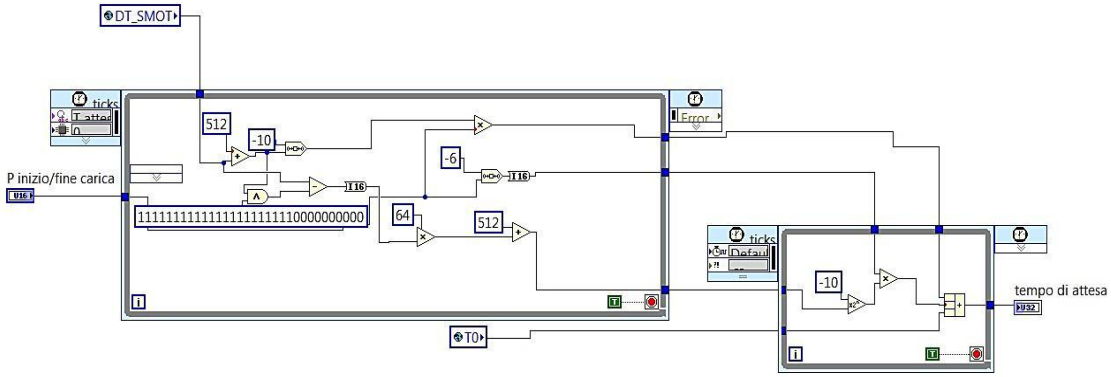


Figura 4.15: tempo di attesa Block Diagram.

Si utilizza un VI chiamato *tempo di attesa* (figura 4.15) che converte il *Pic* (oppure *Pfc*) in un tempo assoluto. Infatti il valore del passaggio dente è espresso in 32 bit mentre le informazioni dei punti di inizio carica in 1024 di angolo, a 16 bit. Questi due valori vengono associati in modo da eseguire una sorta di divisione fra il tempo passaggio dente e la durata angolare dopo il passaggio del dente stesso, combinando fra questi due valori parte dei loro BIT. A *DT_SMOT* viene sommato 512 che è l'equivalente di 10000000000 in codice binario che approssima il valore che verrà poi troncato al bit successivo, successivamente eliminiamo i primi 10 bit e moltiplichiamo i rimanenti 22 con il valore di *Pic* in modo da avere un primo tempo di attesa che considera i primi 22 BIT del valore di tempo passaggio dente. Successivamente si prendono i restanti 10 BIT di *DT_SMOT* e li si combinano con gli ultimi 6 bit di *Pic* in modo da rappresentare sommando questi due tempi con il tempo assoluto *T0* il tempo di attesa fra il dente e l'attuazione da effettuare. Il *While loop* è

impostato in modo da terminare e passare al frame successivo nel momento in cui il tempo totale, meno il tempo di inizio accensione, è minore di un tempo massimo di ritardo della carica bobina esprimibile tramite la variabile *Tick Delay IGN* impostabile in *RT.*

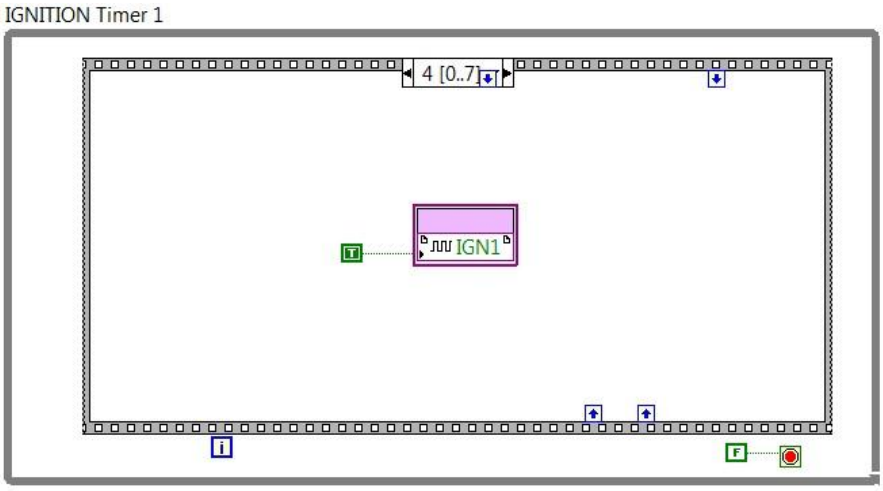


Figura 4.16: Ignition timer, frame 4.

Nel *frame 4* (figura 4.16) viene fisicamente inviato il segnale di inizio carica bobina collegando la variabile booleana *TRUE* alla linea digitale IGN corrispondente al cilindro in esame.

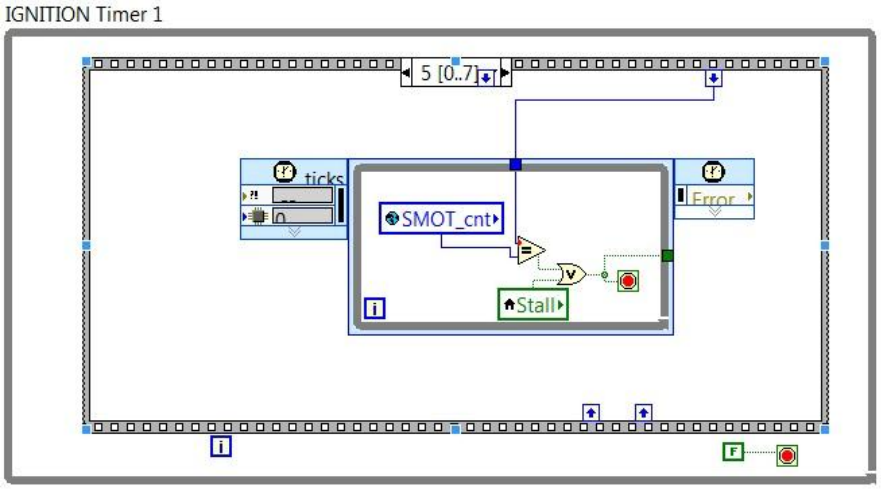


Figura 4.17: Ignition timer, frame 5.

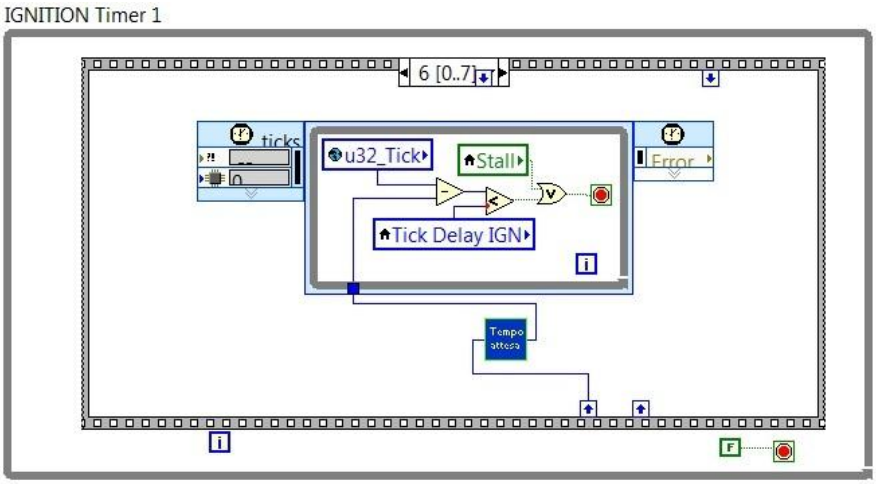


Figura 4.18: Ignition timer, frame 6.

I *frame 5* e *6* (figure 4.18 e 4.19) si occupano, in similitudine al *frame 2* e *3*, di attendere il passaggio del dente di fine carica per poi passare al successivo step che prevede l'identificazione del punto esatto di fine carica dopo il passaggio del dente.

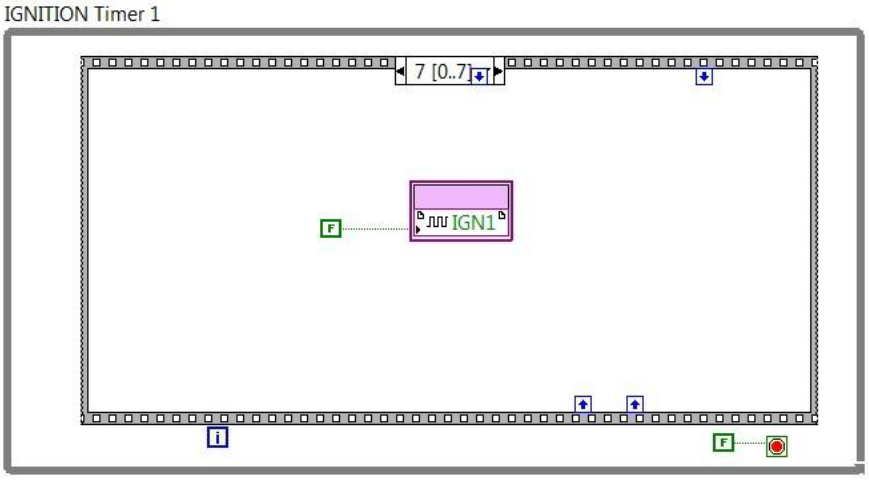


Figura 4.19: Ignition timer, frame 7.

Il *frame 7* (figura 4.19) invece fa terminare una volta esaurito il tempo di carica bobina collegando il booleano *False* alla linea digitale dedicata al cilindro.

A livello teorico, la distanza fra il tempo di inizio e fine carica bobina è pari al tempo di carica bobina imposto dall'utente (di solito un valore di 4 o 5 ms); ma in pratica c'è sempre un errore rispetto al tempo teorico che può essere dovuto a una variazione del tempo passaggio dente a causa del fatto che il VI calcola in anticipo questi tempi, oltre agli errori di approssimazione , seppur piccoli del blocco *Tempo attesa.*

4.3.1 Sicurezza bobine

Il sistema di protezione delle bobine è un apparato necessario della VECU che previene la rottura di questi elementi, nonché danni ai cablaggi del circuito elettrico derivanti dallo scoppio delle stesse.

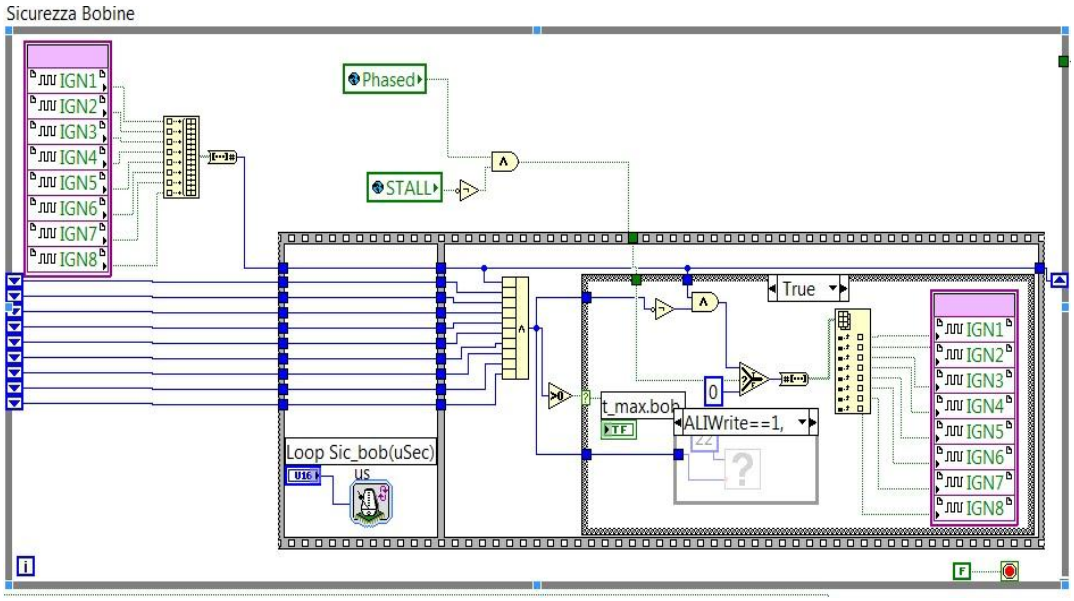


Figura 4.20: Sicurezza bobine.

Esso si basa sul controllo dello stato di una parola composta di 8 bit che viene creata andando a leggere lo stato delle linee digitali IGN per ogni cilindro, dove l'1 corrisponde al comando di carica (*True*) e lo 0 al comando di scarica (*False*); la parola in questione è aggiornata, insieme alle linee digitali, all'arrivo di ogni comando

ed è utilizzata per il controllo della *Sicurezza bobine*. Dove viene monitorata ogni 500 µs grazie al temporizzatore presente all'interno del ciclo *While* (comunque controllabile con la variabile di controllo *Loop Sic_bob*) e ad ogni ciclo viene eseguito un *And* tra le ultime 10 rilevazioni (attraverso l'uso di 9 *Shift Register*) per un totale di tempo pari a 10 volte quello impostato dal *Loop Sic_bob* (nel caso nostro impostato a 5 ms). Nel caso quest'ultima operazione dia un risultato maggiore di 0 e quindi si è verificato che il bit della porta è rimasto alto per più di 5 ms, si innesca la protezione: il comando della linea digitale corrispondente al cilindro dove si è verificato l'errore viene portato a 0 per comandare subito la scarica e viene notificato l'errore anche a livello *Real Time* traite la variabile *t_max.bob.*

4.4 La gestione dell'iniezione

L'iniezione viene gestita in modo analogo a quanto avviene per l'accensione, con la presenza di 8 blocchi, uno per cilindro, che consentono iniettate consecutive e sovrapposte. L'*Injection timer* si presenta come un blocco contenuto in un *While Loop* senza fine che contiene una struttura composta da 7 *Frame*. In tutti i vari frame oltre alla specifica funzione per cui si raggiunge il successivo step, tutto è subordinato dal valore della variabile STALL, in qualunque caso appena il motore va in stallo dobbiamo fermarci esattamente i quel punto.

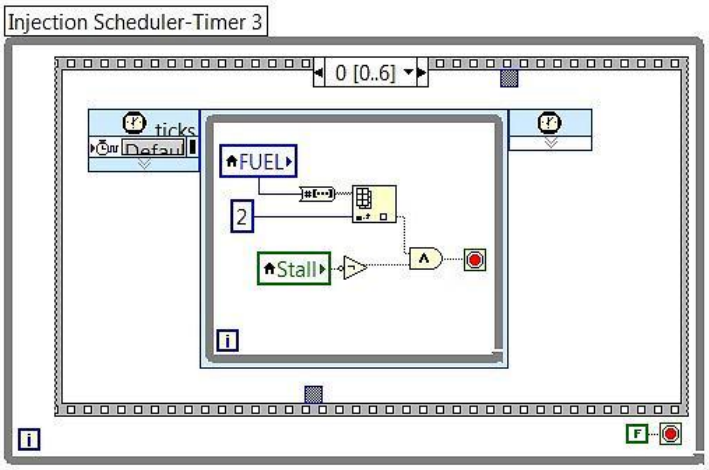


Figura 4.21: Injection timer, frame 0.

Il *Frame 0* (figura 4.21) contiene un *While Loop* temporizzato a 40 MHz che serve a verificare se il blocco di iniezione in esame, relativo al particolare cilindro, è destinato o meno ad essere utilizzato: infatti dal vettore a *FUEL* che rappresenta il numero dei cilindri in cui in quel momento è attiva l'iniezione estrapoliamo il valore relativo a quel cilindro selezionando il valore del cilindro corrispondente attraverso la funzione *Index Array* (nella figura il valore 2 corrisponde al cilindro 3) . il ciclo termina solo quando il valore per quel cilindro è *True* e possiamo passare al secondo *Frame.*

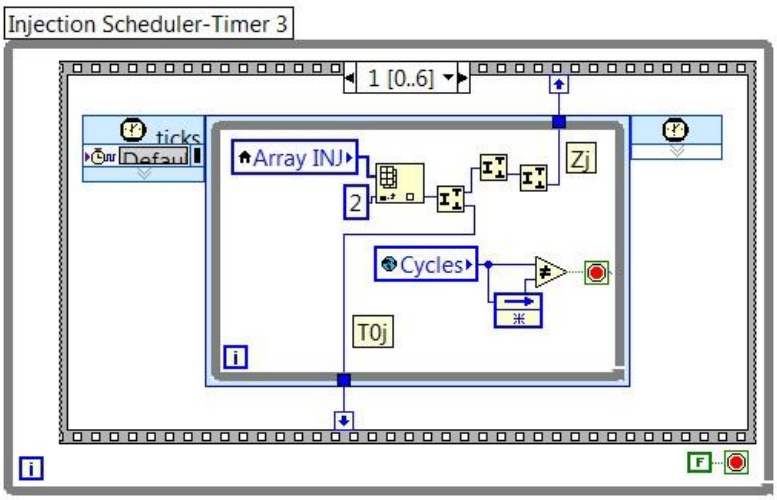


Figura 4.22: Injection timer, frame 1.

Nel frame 1 (figura 4.22) ogni blocco va a leggere la parola corrispondente al cilindro controllato nell' *Array INJ*, vettore composto da 8 parole di 64 bit ciascuna che contengono una serie di informazioni, calcolate nel *RT VI*. La parola è composta nel seguente modo:

- 8 bit rappresentano *Zj*, dente di riferimento di inizio iniezione;
- 32 bit rappresentano *T0j*, ritardo di attuazione rispetto al dente di riferimento espresso in ticks.

Le informazioni ottenute vengono poi richiamate nei successivi *Frame* in cui ci si interroga sulla posizione del sistema rispetto ai punti caratteristici in cui deve essere effettuata l'iniezione. Come per la gestione dell'accensione in questo blocco è stato inserito un *Timed Loop* che fa terminare il frame quando cambiamo ciclo in modo che nel funzionamento a bassi regimi riusciamo a effettuare le attuazioni anche se stiamo all'interno del medesimo dente (regimi bassi, quantità di benzina iniettata bassa, tempo di apertura iniettore basso).

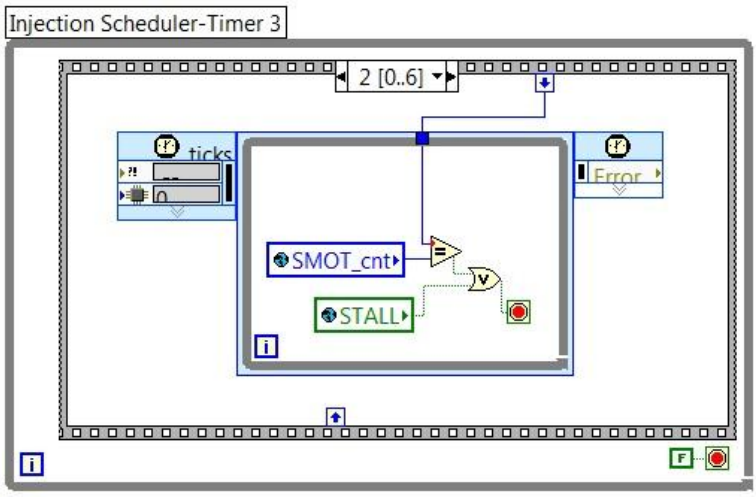


Figura 4.23: Injection timer, frame 2.

Nel *Frame 2* (figura 4.23) confrontiamo il dente di inizio iniezione estratto dalle parole del *Frame* precedente con la posizione

istantanea del motore. Una volta raggiunta la condizione il loop si ferma e possiamo passare al Frame successivo.

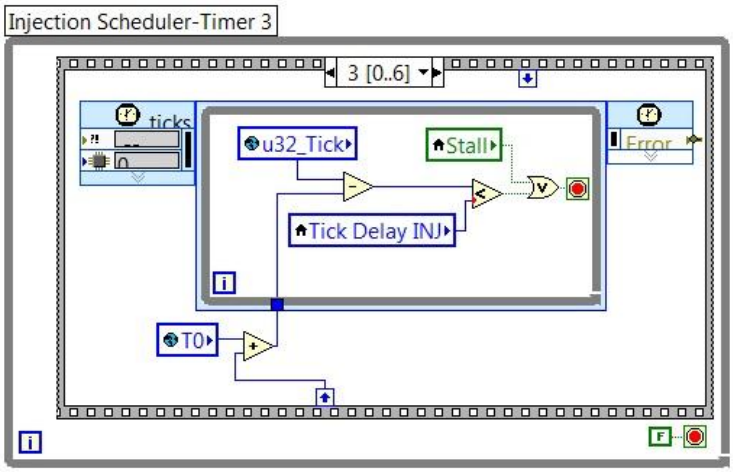


Figura 4.24: Injection timer, frame 3.

Nel *Frame 3* (figura 4.24) verifichiamo come il tempo globale meno il tempo *T0* sommato al *T0j*, cioè il tempo di attesa dell'iniezione dopo il passaggio dente, deve essere inferiore a un certo tempo di ritardo il *Tick Delay INJ* impostabile in *RT*.

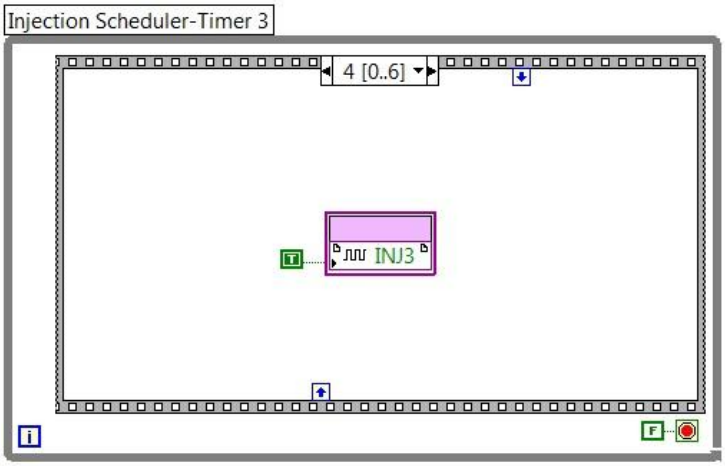


Figura 4.25: Injection timer, frame 4.

Nel *Frame 4* (figura 4.25) andiamo fisicamente ad attivare l'iniezione impostando a *True* la linea digitale la linea digitale dedicata all'attuazione.

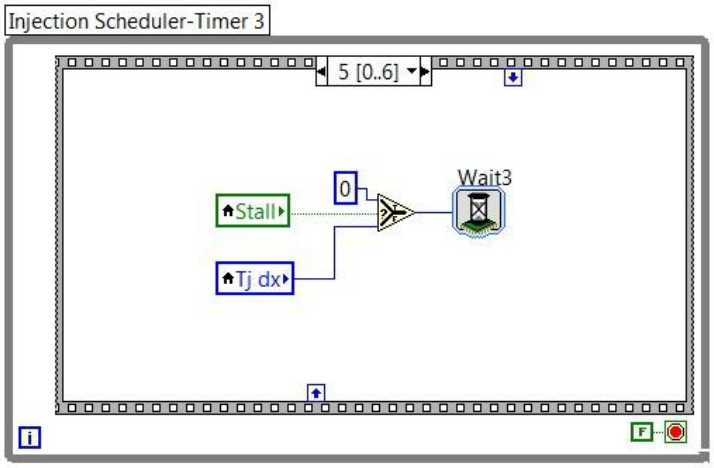


Figura 4.26: Injection timer, frame 5.

Nel Frame 5 (figura 4.26) viene letta la variabile Tj dx (per la bancata di destra cilindri 1-2-3-4, mentre Tj sx per la bancata sinistra, cioè i cilindri 5-6-7-8) influenzata dalla retroazione lambda e viene implementata la durata dell'iniezione collegando il valore letto in microsecondi a un comando *Wait:* solo dopo avere atteso il tempo *Tj* possiamo procedere al *Frame* successivo.

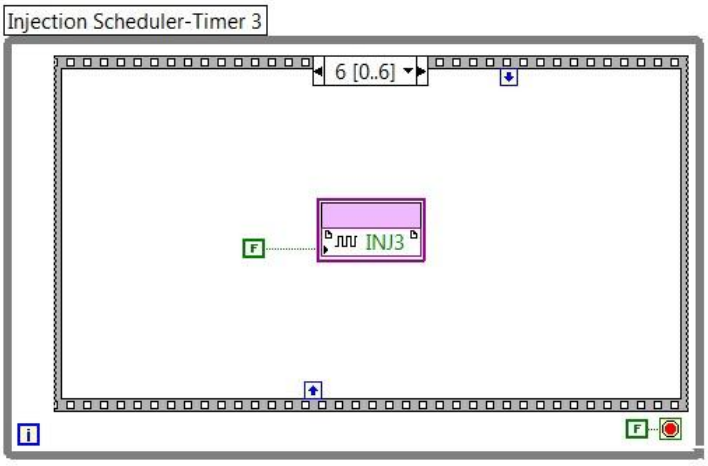


Figura 4.27: Injection timer, frame 6.

Nel *Frame 6* (figura 4.27) avviene, dopo un certo tempo di attesa, lo spegnimento dell'attuazione impostando a *False* la linea *INJ* dedicata all'attuazione.

4.4.1 Gestione degli errori di fasatura

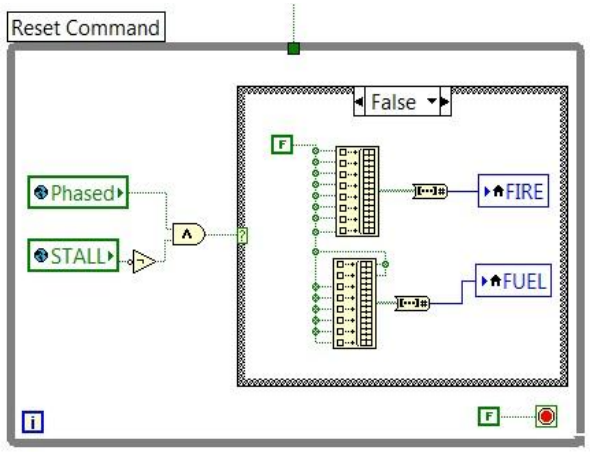


Figura 4.28: Injection timer, frame 6.

Il riquadro *Reset Command* (figura 4.28) ha il compito, semplice ma importante, di bloccare l'esecuzione delle attuazioni di accensione e di iniezione a seguito di errori che eventualmente si sono verificati durante la fasatura del motore.

All'interno di un *While Loop* senza fine, vengono continuamente monitorate le variabile *Phased* e *Stall* che indicano se il sistema di controllo è fasato con la ruota fonica e il sensore di fase: se questo non dovesse avvenire con questo blocco garantiamo che non avvengano esecuzioni di alcun tipo, collegando una costante *False* che si tramuta in una serie di 8 bit impostati a zero che vanno a modificare le variabili *Fuel* e *Fire* spegnendole di fatto tutte. Inoltre il blocco interviene nel caso di una perdita della fasatura.

4.5 Generazione Misfire e MIsfuel

Le nuove normative antiinquinamento hanno imposto controlli sempre severi in fatto di diagnosi di eventi quali misfuel e misfire, che possono causare un aumento considerevole delle emissioni nocive allo scarico. Per testare i sistemi di diagnosi, si è dovuto

così realizzare un controllo che consentisse l'induzione di uno di questi due fenomeni, in maniera manuale per ogni cilindro o automatica impostando dei loop che automaticamente vanno a spegnere l'una o l'altra attuazione in maniera ciclica attraverso il controllo.

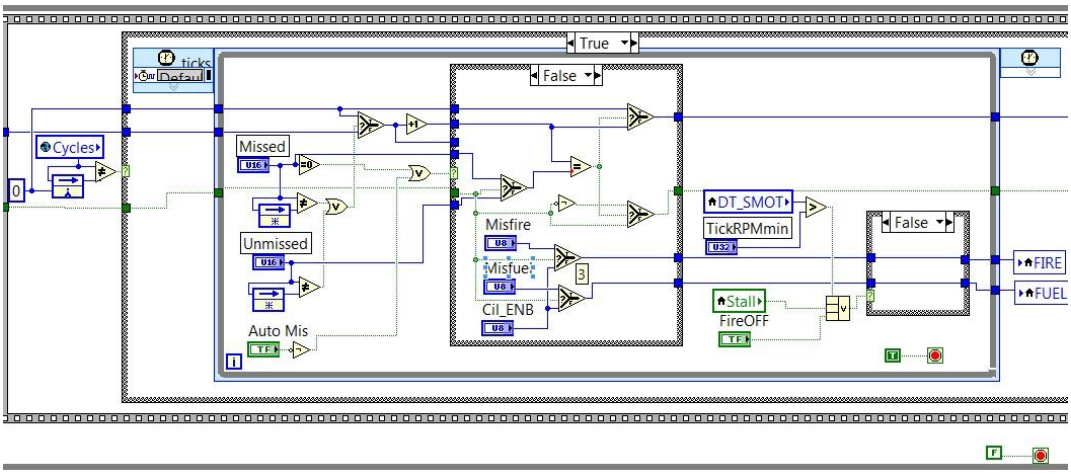


Figura 4.29: blocco Misfire e Misfuel.

Nella parte iniziale del *While Loop* (nel primo *Frame* del *Sequence*) abbiamo la funzione *Wait* che rende il blocco molto meno gravoso dal punto di vista computazionale perché permette l'esecuzione del secondo *Frame* ogni 200 millisecondi. Nel secondo frame (figura 4.29) invece c'è la parte principale del blocco che viene attivata solo ad ogni cambio ciclo imponendo che la variabile *Cycles* sia diversa rispetto al ciclo precedente entrando così in un *Timed Loop* dove se le variabili *Missed* e *Unmissed* sono diverse da 0, oppure siamo nella condizione *AUTO MIS* ON, entriamo nel *Case False* del *Case Structure* successivo. Qui il *Select* viene modificato ogni volta che abbiamo raggiunto il ciclo di *Missed* o *Unmissed* richiesti attraverso il confronto fra quanti cicli precedenti sono stati effettuati in una certa condizione, è infatti il blocco di confronto che riporta la variabile al ciclo successivo, dove viene incrementata di uno, o azzerata nel caso i cicli in una certa condizione siano stati

raggiunti. In questo modo andiamo a scrivere le matrici FIRE e FUEL che identificano quali attuazioni e in quali cilindri devono essere effettuate. Ovviamente i cilindri in cui spegnere le attuazioni sono scelti dall'utente nel RT VI (vedi prossimo paragrafi). In questo blocco a monte della scrittura delle variabili FIRE e FUEL c'è un *Case Structure* dove nel caso di stallo del motore (*Stall* a true), di un regime di rotazione troppo basso (impostato dall'utente di solito a 100 rpm attraverso la variabile *TickRPMmin*) o della variabile *FireOFF* impostata a true vengono spente tutte le attuazioni per tutti i cilindri.

4.6 Gestione del controllo *PWM* per corpo farfallato

L'introduzione di un controllo in Pulse With Modulation per il corpo farfallato, lo stesso utilizzato per il drive by wire di serie, è stato dettato sia da motivi pratici (in questo modo infatti si può stabilire il carico di funzionamento del motore direttamente dal pannello di controllo della VECU), sia dall'intenzione di avviare un progetto di interfaccia tra i controlli di freno e motore. Questo renderà possibile l'esecuzione in modo automatizzato di gradini, rampe o sequenze di punti di funzionamento prestabiliti dall' utente per lo svolgimento di test o calibrazioni. La variabile di controllo è rappresentata da *Duty Cycle* del PWM ossia il rapporto tra l'intervallo di tempo in cui l'onda quadra caratteristica del segnale, rimane alta ed il periodo totale dell'onda stessa che rimane circa costante, dato che, in genere, la frequenza del segnale è fissata. Il controllo attraverso software è stato aggiunto in parallelo a quello esistente dando la possibilità all'utente di scegliere con un interruttore manuale quale dei due utilizzare. Per il PWM realizzato con LabVIEW si è scelto di utilizzare una frequenza d'onda pari a 100 Hz ed un Duty Cycle regolabile tra lo 0% e il 100% attraverso il potenziometro virtuale indicato con

%*Farfalla*. La parte che si occupa dell'elaborazione del segnale è stata implementata sull'FPGA VI ed è a sua volta suddivisa in due blocchi contraddistinti, *PWM Read* e *PWM Generation*, entrambi delimitati da cicli *While* senza fine.

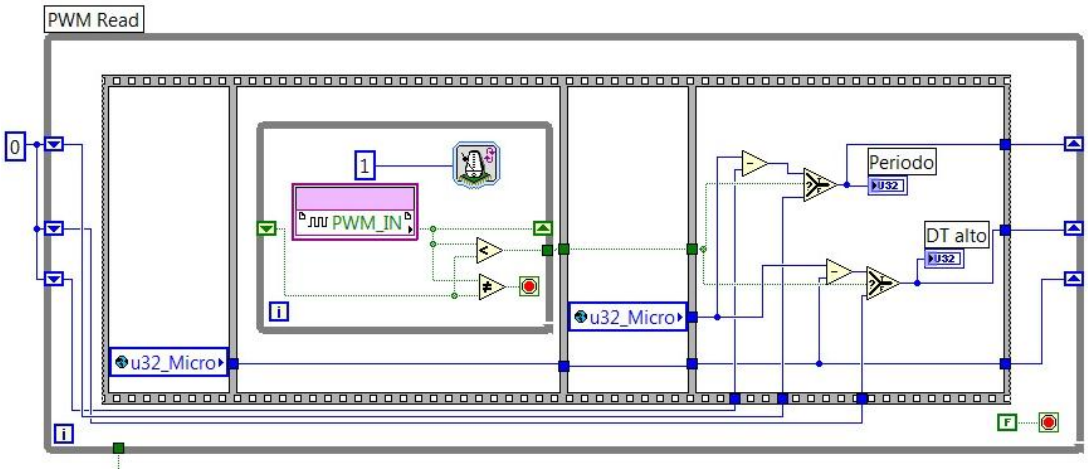


Figura 4.30: PWM Read.

Nella sequenza contenuta in *PWM Read* si trova prima di tutto una prima volta le variabili *u32_Micro* che ha il compito di registrare in quale istante inizia la sequenza; poi all'interno di un *While Loop* avviene periodicamente, con una frequenza pari a 1 millisecondo, la lettura del canale digitale *PWM_IN*, che interrompe il ciclo quando si verifica la transizione del segnale in ingresso da un valore basso (inferiore a 0.7V) ad uno alto (superiore ai 2.5 V) o viceversa. Nel terzo *Frame* richiamiamo nuovamente la variabile *u32_Micro* che memorizza l'istante in µs in cui è avvenuta la transizione. Nell'ultimo *Frame* vengono calcolati i valori di *Periodo, DT alto* e *DT basso*: se la transizione nel *While Loop* era stata del tipo alto-basso, il valore di *Periodo* viene calcolato coma la differenza tra i valori del terzo *Frame* della variabile *u32_Micro* e la stessa variabile al ciclo precedente, il valore di *DT alto* viene ricavato dalla differenza tra *u32_Micro* fra il terzo e il primo *Frame*, mentre *DT basso* viene ricavato per differenza tra periodo e *DT alto*. Se siamo invece nella condizione

che la transizione del *While Loop* era del tipo basso-alto, allora *Periodo, DT alto* vengono prelevati dai rispettivi *Shift register* e rimangono costanti al valore calcolato al ciclo precedente (e quindi tutte le variabili vengono aggiornate solo quando si ha la transizione di tipo alto-basso). Questi dati vengono poi elaborati, come visto in precedenza.

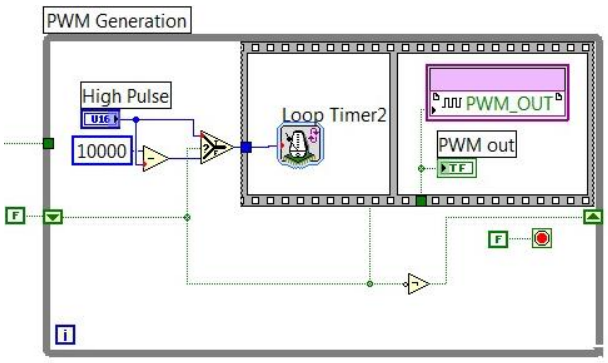


Figura 4.30: PWM Generation.

L'ultima parte dell'applicazione che genera l'onda quadra, con le caratteristiche determinate a livello di *Real time*, è il blocchetto *PWM Generation*. Per prima cosa viene letto dallo *shift register* di sinistra il valore opposto a quello in uscita dal canale digitale *PWM_OUT* e viene passato al timer, contenuto nella prima delle due sequenza interne, il tempo di attesa prima della scrittura sul canale. Quest'ultimo corrisponde al periodo dell'impulso alto, contenuto nella variabile *High Pulse*, se il segnale in uscita è alto, mentre corrisponde alla differenza tra 10000 e *High Pulse* stesso se il segnale è basso: 10000 rappresenta l'equivalente in µs del periodo corrispondente alla frequenza di 100 Hz. Terminata l'attesa del timer, si passa alla seconda sequenza dove viene scritto su *PWM_OUT* il valore precedentemente letto dallo *shift register* di sinistra e contemporaneamente su quello di destra viene messo a disposizione per l'iterazione successiva lo stesso valore negato. In questo modo si ottiene un'onda quadra con i

due intervalli di tempo (periodo alto e periodo basso) invertiti, poiché, prima di raggiungere la farfalla, tale segnale è condizionato da un piccolo circuito elettrico invertente che riporta l'onda alla forma desiderata in partenza.

4.7 Waste Gate e pompa benzina di bassa pressione

Si è reso necessario cercare di rendere la VECU assolutamente indipendente dall'utilizzo della centralina di serie comandando e rendendo gestibili dall'utente attraverso l'Host PC tutti i dispositivi che la centralina governa durante il suo normale funzionamento. In particolare è stato necessario implementare in FPGA la serie di comando e gestione della Waste Gate che come descritto nel capitolo 3 e uno degli strumenti utilizzati dal controllo per regolare il funzionamento del motore regolando, attraverso i dati rilevati dalla valvola Pierburg, il By-pass di gas di scarico in turbina. Questo lavoro di gestione è stato affrontato dalla tesi di Maioli e l'obiettivo è quello di implementare il suo lavoro all'interno della VECU 23. Stesso discorso vale per la pompa di bassa pressione della benzina in cui è stato necessario implementare un controllo in FPGA che rendesse possibile disattivare la pompa in condizione di non funzionamento motore (con l'obiettivo di risparmiare corrente in batteria), a anche come sicurezza, rendendola disattivabile e dando quindi la possibilità di tagliare l'alimentazione carburante al motore, provocandone quindi lo spegnimento in caso di malfunzionamento del motore.

4.7.1 Gestione Waste Gate

In *FPGA* la gestione delle Waste Gate avviene in modalità analoga alla generazione del corpo farfallato descritto in precedenza.

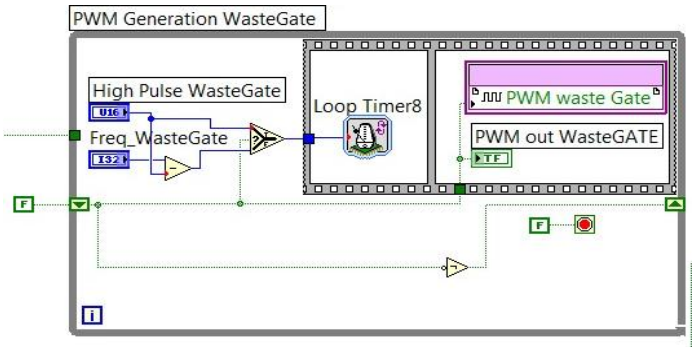


Figura 4.31: PWM Generation Waste Gate.

Le due variabili *High Pulse WasteGate* e *Freq_WasteGate* controllano infatti la durata temporale dei cicli di ON e OFF (intesi come Duty cycle) di controllo e di apertura delle valvole. La logica con cui effettuiamo questa transizione è la seguente: *High Pulse WasteGate* indica la transizione di alto (ON) della valvola e va a scrivere il valore del temporizzatore (tempo di attesa) *Loop timer 8* prima di andare a scrivere la variabile *True* nella linea digitale dedicata (la variabile è *True* perché è la medesima che governa l'operatore *Select*). Viceversa se la variabile che entra nel *Select* è *False* il tempo di attesa, prima di porre a *False* (OFF) la line digitale dedicata, è stabilita dalla differenza fra il tempo totale del ciclo della valvola meno la transizione di alto (in pratica determiniamo la transizione di basso). In pratica abbiamo la continua alternanza dei valori *True* (ON) e *False* (OFF) con i rispettivi tempi decisi dai tempi di Duty cycle.

4.7.2 Gestione pompa carburante

Il controllo di questo dispositivo si basa sostanzialmente sulla necessità di rendere la pompa di bassa spegnibile in ogni condizione dall'utente oltre che a salvaguardare batteria e funzionamento della stessa spegnendola in automatico nel caso il controllore sia acceso, ma non siamo a motore acceso. Tutto questo con l'esigenza però di non aggravare più del necessario la potenza di calcolo in *FPGA*.

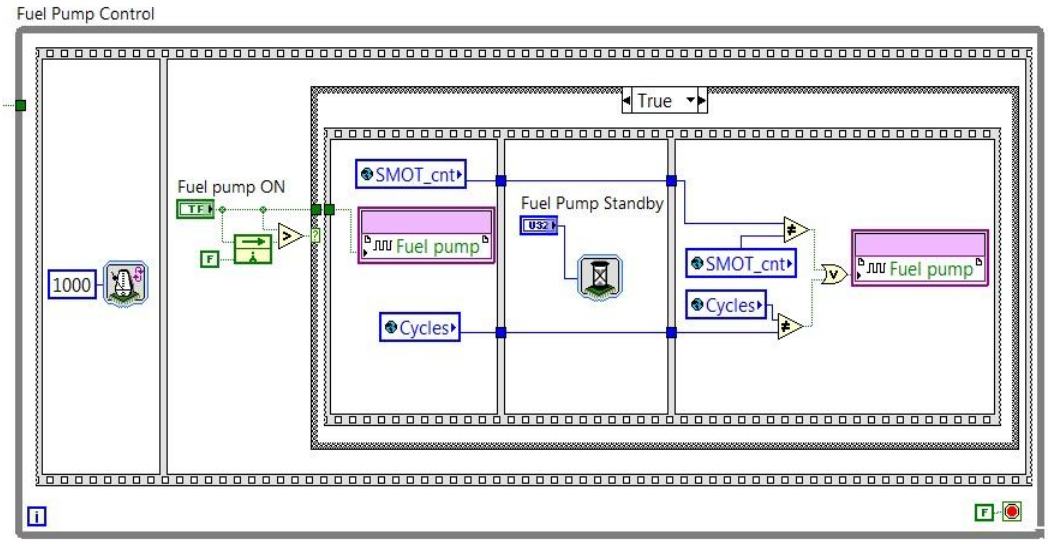


Figura 4.32: Fuel Pump control case True.

Per fare ciò si è impostato un *Sequence loop* con un temporizzatore di attesa nel primo *frame*, che in pratica svolge la funzione di interrogare la parte di codice successiva ogni secondo (sono impostati infatti 1000 ms) rendendo questo loop poco gravoso in termini di impegno computazionale. Nel secondo *frame* invece abbiamo la parte che si occupa dello spegnimento della pompa stessa. Il *Case Structure True* è governato dall'ipotesi che il valore del pulsante di controllo sia vero al primo ciclo oppure passi da falso a vero durante l'esecuzione dello stesso, mediante la condizione di maggiore fra due comandi booleani (in questo caso la condizione è vera solo nel caso True è maggiore di False) e verifica che con un' attesa massima imposta dal comando *Fuel pump standby* non ci siano variazioni nel passaggio dente o di ciclo. In questo caso, cioè in cui è *False* la condizione di diversità dei denti o del ciclo, impostiamo a OFF la linea dedicata al controllo pompa, il *DIO 17* denominato *Fuel Pump*.

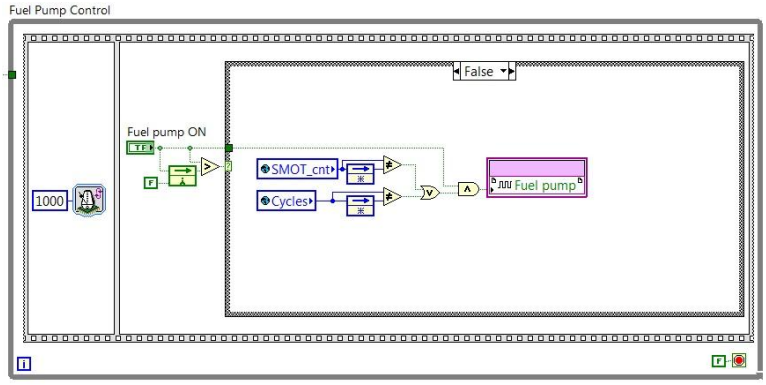


Figura 4.33: Fuel Pump control case False.

Nel *case structure false* che è attivo in tutte le altre situazioni è possibile spegnere o riaccendere la pompa durante il funzionamento stesso del motore anche in caso siamo in fase di normale funzionamento, oltre a essere presente la condizione per cui in caso di mancata accensione del motore in automatico si imposta a OFF la linea dedicata alla pompa.

4.8 Scrittura e lettura FIFO

La necessità di verificare il corretto funzionamento del sistema di controllo ha portato alle necessità di monitorare alcune grandezze provenienti dall'FPGA dall'esterno. Per questo motivo si utilizzano delle memorie esterne definite FIFO, che consentono di immagazzinare un numero considerevole di dati.
La memoria FIFO di cui si è trattato è la memoria che abbiamo chiamato come FIFO AI che da la possibilità di monitorare 8 canali di acquisizione analogici e 4 digitali rendendo possibile operazioni di post revisione, cioè si è implementato un blocco di salvataggio dati proveniente da FPGA per avere un monitor dei valori di questi canali. Vediamo nel dettaglio l'implementazione della scrittura dati per capire anche la logica con cui vanno utilizzati questi strumenti di analisi. L'applicazione consiste in un *Sequence Loop* a 4 *frame.*

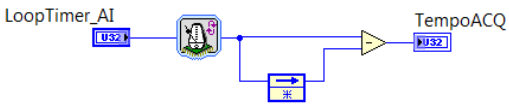


Figura 4.34: Frame 0 FIFO AI.

Nel *frame 0* abbiamo la definizione di un tempo di attesa prima dell'implementazione dei successivi, questo tempo è importante e non può essere scelto a caso, ma come vedremo deve essere scelto dall'utente secondo una certa logica che tiene conto dei limiti computazionali del' applicazione. A questo proposito l'indicatore *TempoACQ* ci permette di individuare qual è l'effettiva velocità di esecuzione dell'intero *Sequence*.

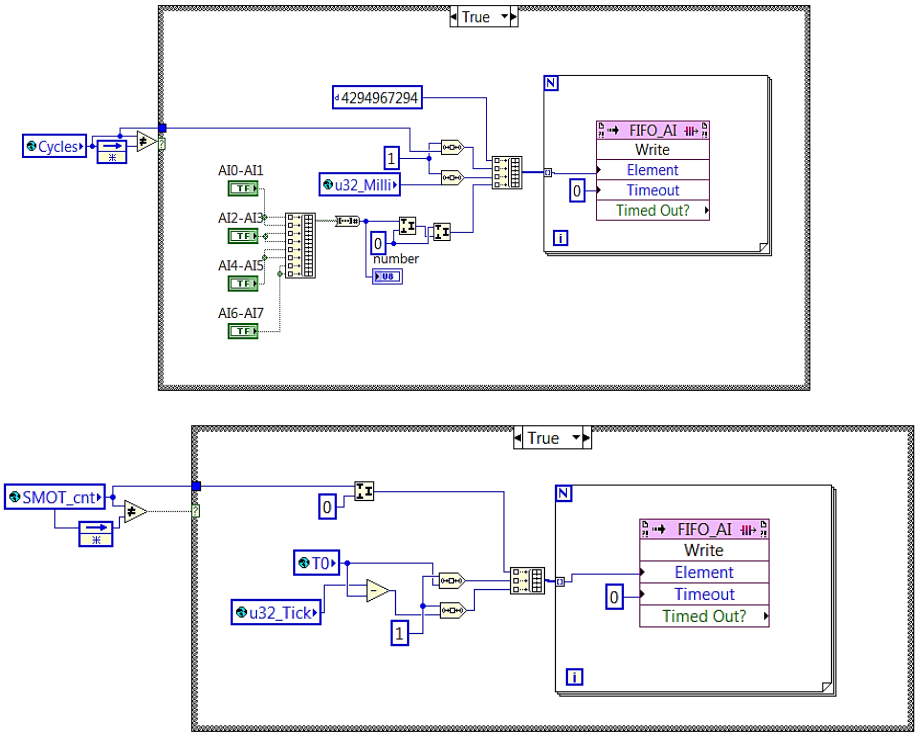


Figura 4.35: Frame 1 e 2 FIFO AI.

Nel *frame 1 e 2*, invece, iniziamo con la scrittura dati, infatti se siamo in corrispondenza rispettivamente di un cambiamento di ciclo o di dente entriamo, per entrambi, nel *Case True* dove andiamo a scrivere all'interno della *FIFO AI* alcuni valori che

imponiamo tutti aventi l'ultimo bit a zero per rendere più facile lo "spacchettamento" dei dati:

- Il primo, che si può definire come un marcatore è il numero 4296917294 corrispondente a 32 bit a 1 tranne l'ultimo;
- Il ciclo corrispondente variabile *Cycles*;
- Il tempo attuale di ciclo *u32_Milli*;
- i canali analogici attivati;
- Il dente corrispondente (primi 16 bit di un dato a 32 bit);
- Il tempo di passaggio dente *T0* e la differenza con il tempo totale.

Nel *frame 3* andiamo a scrivere tutti quei valori che sono in INPUT dai vari canali analogici e digitali. Gli 8 canali analogici sono raggruppati a gruppi di e attivabili attraverso 4 pulsanti governabili in *RT.*

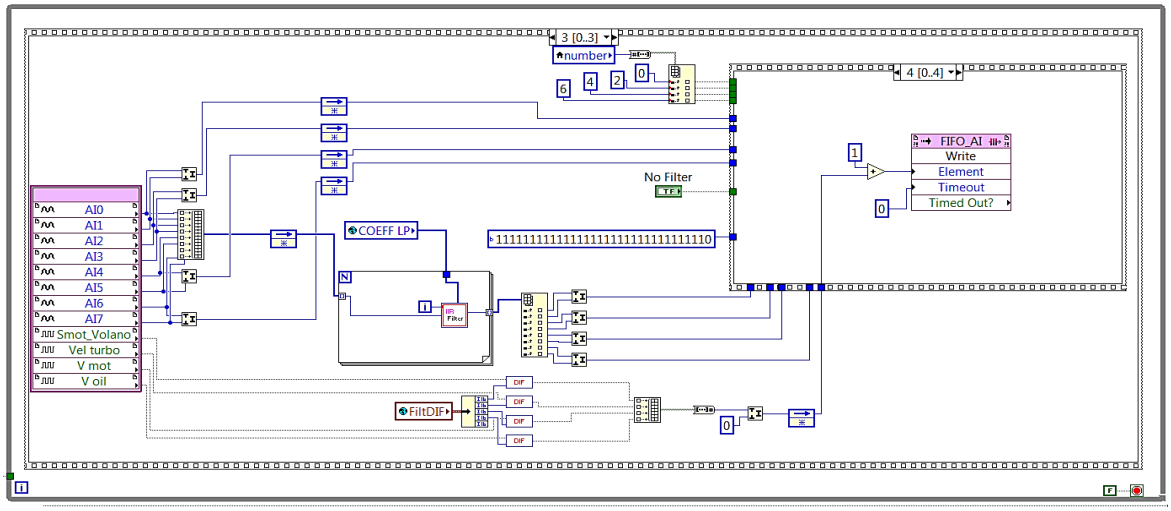


Figura 4.36: Frame 3 FIFO AI.

Tutti i valori sono richiamati al ciclo precedente per rendere ancora più veloce l'operazione di scrittura, infatti acquisire e avere a disposizione per la scrittura il medesimo dato all'interno dello stesso ciclo potrebbe essere leggermente penalizzante. Ogni canale analogico può essere scritto nella sua forma grezza oppure filtrato a seconda del comando *No filter* che se acceso

disattiva l’intervento del filtro sul canale. La logica della scrittura è quindi la seguente, andiamo ad acquisire e a scrivere su *FIFO* quei valori dei canali che ci interessano interponendo fra questo numero elevato di dati le informazioni su ciclo e dente del sistema ogni qual volta esso risulta diverso dal precedente.

Capitolo 5

Controllo attuazioni in RT

5.1 Il controllo in Real Time

In questo paragrafo verrà fatta una trattazione del *RT VI* chiamato *Control Panel* mirata ad evidenziare le parti strutturali del programma e le relative funzioni. Come già detto nei precedenti capitoli, il modulo *Real Time* possiede la caratteristica di essere un'applicazione deterministica, ma non a livello del modulo *FPGA.* Definito come *soft real time*, ha la possibilità di eseguire loop alla frequenza di 1 kHz in ambiente Windows, e fino a 1 MHz in ambiente Phar Lap. Questo modulo presenta però una notevole flessibilità, che consente l'implementazione di strategie di controllo complicate che non sarebbero possibili in ambiente *FPGA*. I compiti della parte ad alta priorità sono:

- interfaccia utente e visualizzazione regime motore;
- inizializzazione e supervisione del *FPGA VI;*
- esecuzione calcoli dei parametri di attuazione per accensione e iniezione;
- elaborazione comandi di iniezione e accensione;
- acquisizione parametri per il calcolo delle attuazioni (pressione e temperatura aria nel collettore, ecc.);
- generazione misfire e misfuel;
- gestione di tutti gli apparati del motore in sostituzione della centralina di serie;
- registrazione dati di telemetria.

Tutti questi dati sono implementati in un singolo Vi di cui nel paragrafo descriveremo nel dettaglio le caratteristiche.

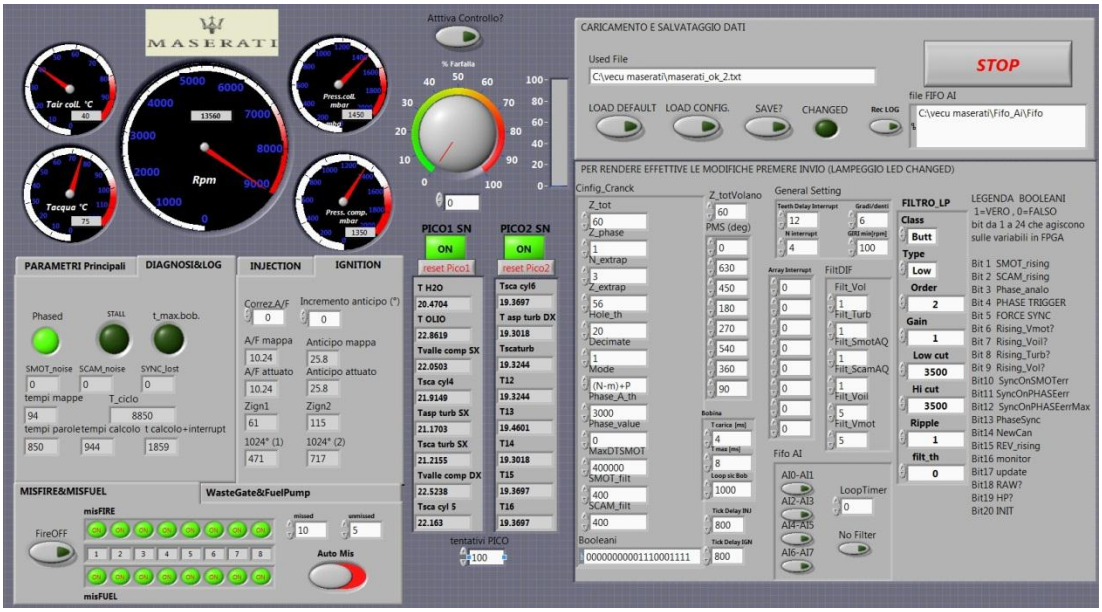


Figura 5.1: Front Panel del RT VI.

5.1.1 Introduzione al VI

Il *Front Panel* del VI è l'interfaccia grafica, così come è visualizzata nell'*Host PC*, della centralina virtuale e presenta indicatori per la visualizzazione delle grandezze monitorate dai sensori e controlli per l'iterazione dell'utente con gli algoritmi di calcolo implementati nel VI. In particolare, nella parte superiore si vedono gli indicatori analogici per:

- temperatura acqua di raffreddamento;
- temperatura aria nel collettore di aspirazione;
- regime di rotazione in rpm;
- pressione nel collettore di aspirazione;
- pressione di sovralimentazione.

È presente anche la parte di settings di tutti i parametri motore, la parte che si carica ad ogni avvio del VI e che permette il caricamento o il salvataggio di dati, oltre che una serie di tab che permettono il controllo di tutti i dispositivi del motore a banco. È inoltre presente il pulsante di *STOP* necessario a far cessare le attività dell'applicazione, oltre che il controllo dell'apertura della farfalla del motore. Entriamo nel dettaglio.

5.1.2 Tab di controllo del VI

Per tab si intendono delle finestre che hanno la funzione grafica di separare indicatori e pulsanti aventi fra loro differenti funzioni. In particolare sono tre, aventi ciascuno due finestre, concentrati nella parte inferiore del VI.

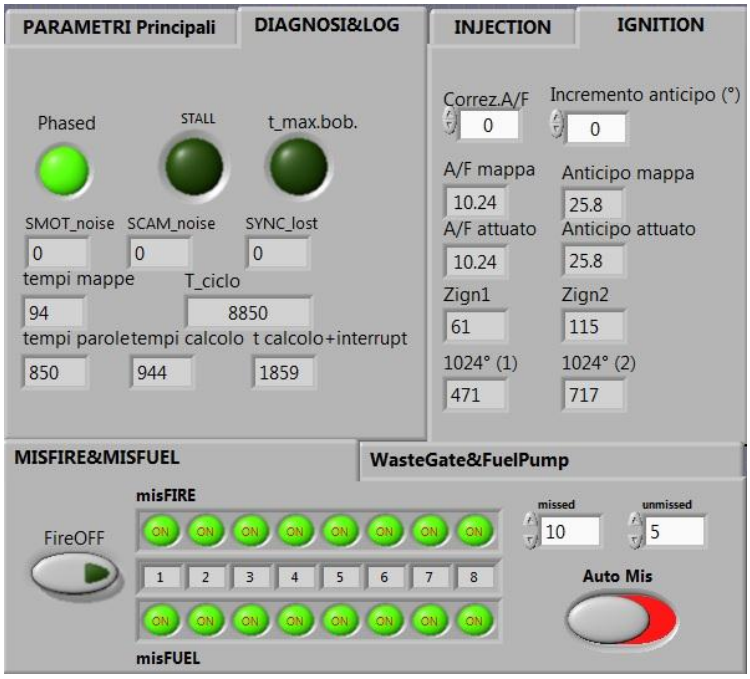


Figura 5.2: Tab nel RT VI.

Il tab *PARAMETRI Principali* fornisce delle informazioni riguardo il funzionamento del VI e hanno il compito di fornire all'utente le

adeguate informazioni riguardo l'interfaccia fra *FPGA* e il *RT VI* quali le variabili *FUEL* e *FIRE* il tempo di carica bobina, il numero di ciclo attuale, il tempo di passaggio dente oltre alla velocità effettiva di scrittura dei dati nelle memorie *FIFO* (argomento che tratteremo separatamente nei prossimi paragrafi).

Il tab *DIAGNOSI&LOG (figura 5.2)* permette invece di controllare i dati principali provenienti dall' *FPGA,* per monitorare la fasatura del motore (controllo dei led *Phased* e *Stall)*, eventuali superamenti dei tempi massimi di carica bobina, errori nell'elaborazione dei segnali di fonica e fase (indicatori *SMOT_noise*, *SCAM_noise* e *SYNC_lost*) oltre al monitoring dei tempi di calcolo del sistema per tutto quello che riguarda l'elaborazione parametri.

Il tab *INJECTION* (figura 5.3) contiene i parametri iniezione da cui è possibile monitorare il valore di lambda misurato dalle sonde lineari e si ha l'opportunità di correggere il tempo di iniezione calcolato da mappa indistintamente per le due bancate. Inoltre sono visualizzati i dati ricavati da mappa, in particolare il dente di riferimento di inizio iniezione, il tempo di attesa, il tempo di iniezione, e l'arricchimento miscela istantaneo del motore dopo l'intervento dell'utente, il quale può andare ad aumentare la quantità di iniettato tramite *Tj sx* e *Tj dx*, oltre che i valori delle costanti di retroazione lambda tramite *Ki* e *Kp*.

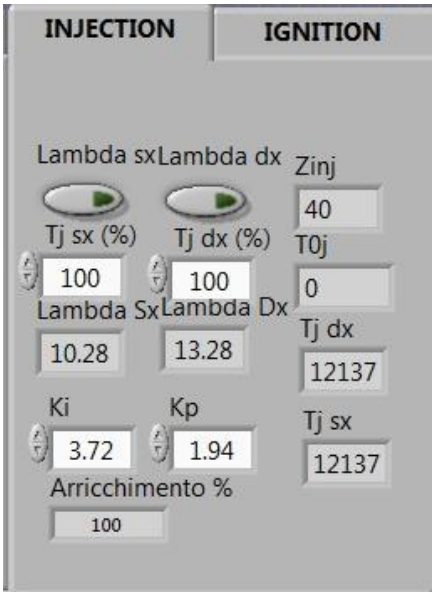


Figura 5.3: Tab INJECTION.

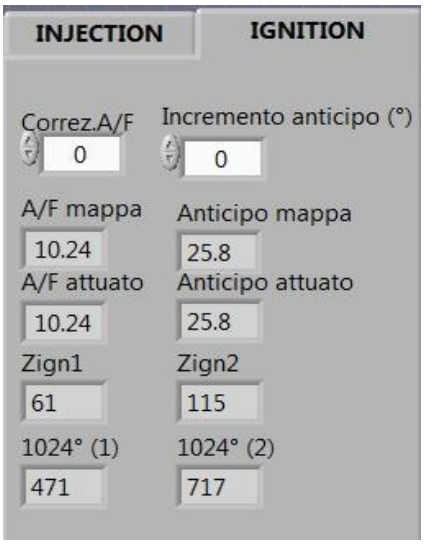


Figura 5.4: Tab IGNITON.

Il tab *IGNITON* (figura 5.4) contiene i parametri principali di accensione, cioè il titolo della miscela A/F con gli indicatori del titolo di mappa e del titolo effettivamente attuato dopo l'intervento utente (mediante il controllo *Correz.A/F*), l'anticipo di accensione in gradi eseguito dalla mappa e quello effettivamente comandato dopo la correzione dell'utente tramite il controllo *Incremento anticipo*. Sono inoltre riportati i valori dente di riferimento di inizio e fine accensione oltre al ritardo in 1024esimi di angolo (vedi capitolo precedente).

Il tab MISFIRE&MISFUEL (figura 5.2) permette invece la generazione di particolari condizioni operative del motore, in particolare per indurre misfuel e misfire (cioè rispettivamente mancate iniezioni e mancate accensioni) con la possibilità di sceglierne per ogni cilindro. È possibile anche eseguire cicli di spegnimento delle attuazioni attraverso il comando *Auto Mis* attraverso i controlli *Missed* e *Unmissed.*

5.2 Le parti del Real Time

Per comprendere meglio come tutti questi indicatori e controlli influiscono sul controllo del motore si analizzerà il diagramma a blocchi. Questo è composto da più parti distinte tra loro che sono:

1. Inizializzazione Vi e settings parametri motore (par. 5.2.1).
2. Inizializzazione dell'*FPGA* (par. 5.2.2).
3. Attesa e notifica degli interrupt (par. 5.2.3).

4. Acquisizione parametri (par. 5.2.4).
5. Calcolo ed invio parametri delle attuazioni (par. 5.2.5)
6. Stop (par. 5.2.6)

5.2.1 Inizializzazione VI e settings parametri motore

Si è reso necessario rendere il caricamento dei parametri del motore semplice e elementare per ogni utente che non abbia lavorato alla realizzazione del VI.

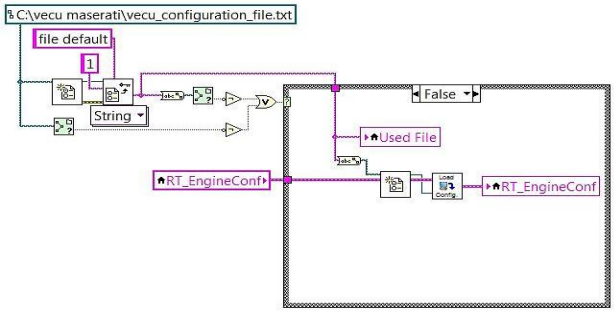


Figura 5.5: blocco inizializzazione case true.

Infatti le potenzialità di LabWIEV a volte si scontrano con la difficoltà di comprensione di certe parti del programma, in particolare certe funzionalità di codice possono essere raggiunte con molte tipologie di comandi. L'obiettivo era quindi quello, semplicemente avviando il VI, di avere una configurazione dei parametri in grado di avviare il motore, oltre che essere facilmente salvabili e caricabili.

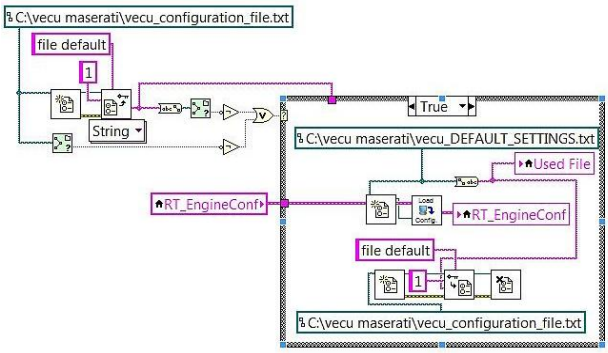


Figura 5.6: blocco inizializzazione case false.

Nel primo *frame* durante il caricamento del programma è presente un blocco che va a ricercare due files particolari all'interno della memoria del PXI: *Vecu_configuration_file.txt* che in pratica è il file in cui è scritto qual' è l'ultima configurazione utilizzata dall'utente e appunto il file in cui sono salvati tutti i valori. Attraverso la funzione *File Exists* verifichiamo se nella cartella sono presenti e andiamo all'interno del *Case False* a caricare i valori nel cluster *RT_EngineConf*.

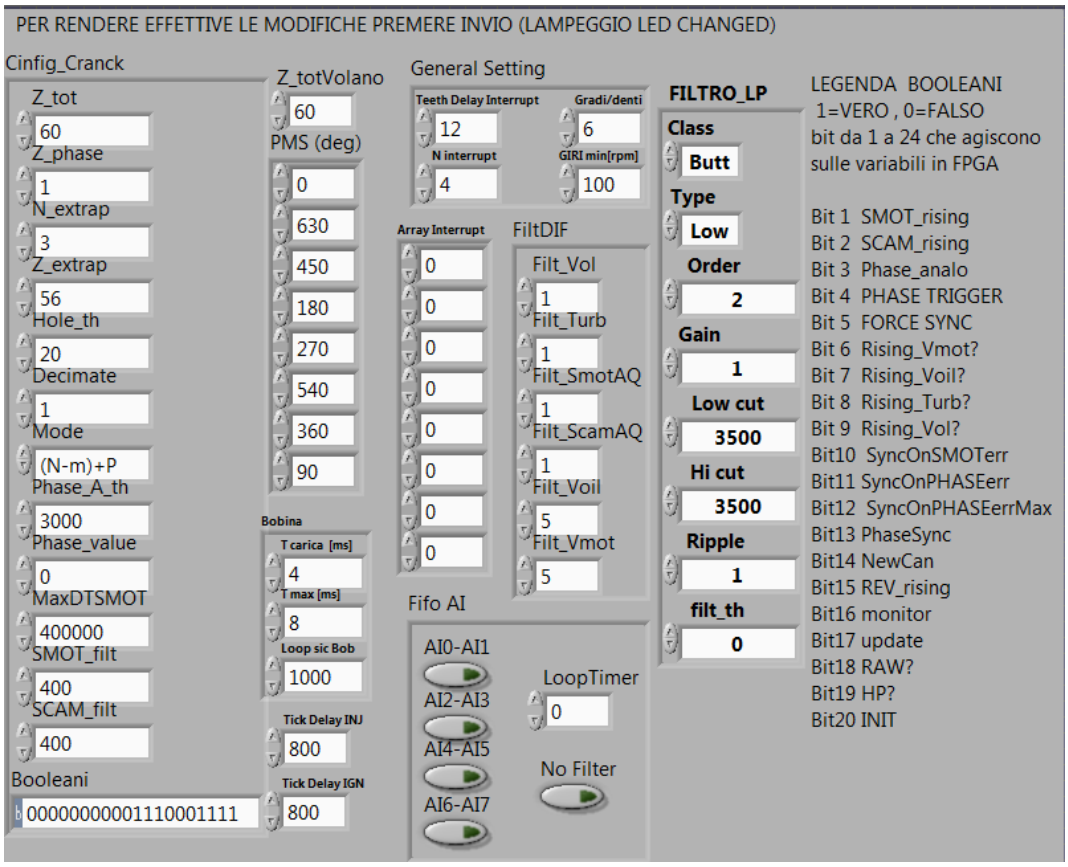


Figura 5.7: parametri del motore (cluster RT_EngineConf).

Se invece uno di questi due dovesse essere corrotto o non dovesse esistere nella cartella specificata, entrando nel *Case TRUE* andiamo a caricare le impostazioni dal file

vecu_DEFAULT_SETTINGS.txt che rappresentano una configurazione base del motore. Nel cluster di figura 5.7 sono presenti tutti i parametri motore fondamentali per l'avvio del motore stesso in particolare quelli che riguardano l'interfaccia e la fasatura dell'FPGA oltre che il parametro *Booleani* che racchiude tutti i booleani che si implementano in *FPGA* come spiegato nella legenda a fianco.

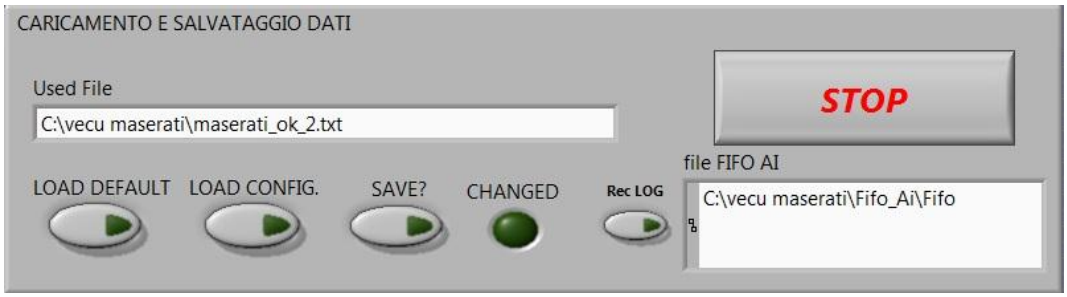


Figura 5.8: gestione dei file.

È inoltre possibile durante ogni istante dell'esecuzione del VI salvare o caricare impostazioni del cluster mediante i pannello di figura 5.8. Con i tasti *LOAD CONFIGURATION* e *SAVE?* si aprono automaticamente altri VI che permettono rispettivamente all'utente di caricare una configurazione fra i file presenti nella cartella dedicata e di salvarne uno con nome nella memoria del PXI, mentre con il tasto *LOAD DEFAULT* si richiama il file di configurazione base. Un'altra particolarità della gestione dei file è quella in pratica di andare ad aggiornare l'applicazione con l'ultimo file utilizzato ogni volta che si chiude il programma e da la possibilità all'utente una volta chiusa l'applicazione di aggiornare il file in caso di modifica di alcuni parametri. Il tipo di file scelto per la configurazione dei parametri è il file di testo *Configuration file* che da la possibilità di essere modificato anche senza entrare direttamente nel VI semplicemente cambiando i valori nel file di testo nella sezione corrispondente.

5.2.2 Inizializzazione FPGA

Le funzioni che vanno inizializzate sono il caricamento dell'*FPGA VI* su cui lo stesso deve poter operare, inizializzare le sue variabili e, successivamente, avviarlo. Il riferimento dell'*FPGA* va inserito in un blocco in cui si dichiara la scheda specifica (bel nostro caso una 7833R) che si intende utilizzare ed il VI da caricare sulla stessa. Segue la funzione *reset* del blocco *Invoke method*, che consente di impostare tutti gli indicatori, controllori e variabili locali e globali dell'*FPGA VI* al loro valore di default. Il grande blocco che segue si presenta come una flat sequence, ovvero une serie di *Frame* che vengono eseguiti una sola volta all'avvio del VI. Successivamente viene dato l'avvio all'*FPGA* attraverso il *run* del VI che invia il riferimento a tutti le altri parti del VI.

5.2.3 La ricezione degli interrupt

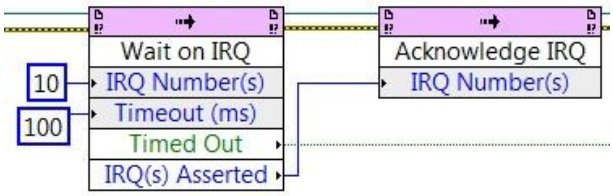


Figura 5.9: Richiamo IRQ.

Gli interrupt hardware costituiscono uno strumento molto potente, ma allo stesso tempo delicato poiché la loro scrittura impedisco il richiamo degli stessi in più punti: non è possibile utilizzare all'interno dello stesso VI più di un *Invoke Method* per l'attesa del medesimo IRQ pena la loro perdita. Sono presenti due ricezioni per l'IRQ, una per il numero 0 e una per il numero 10 entrambe seguite da un *acknowledge* che in pratica è lo strumento per comunicare all'*FPGA* l'avvenuto riconoscimento dell'interrupt e quindi abilita il resto del blocchetto ad eseguire tutto il resto delle operazioni. Gli IRQ sono di due tipi perché si ha la possibilità di avere uno sfasamento nei calcolo fra

FPGA e *RT* dato appunto dalla variabile *Teeth Delay Interrupt* che permette di avere sfasamento fra l'interrupt 0 e il 10. È possibile anche determinare i denti a cui far effettuare i calcoli attraverso la matrice *Array Interrupt*. L'interrupt 0 gestisce il calcolo di tutti i comandi del motore in particolare si entra nel blocco di elaborazione comandi, con tutti i dati rilevati dai sensori, andando ad individuare e a scrivere in FPGA tutti i parametri utili alle attuazioni. L'interrupt 10 invece va a determinare tutte quelle situazioni di aggiornamento dei valori di rilevamento di alcuni sensori acquisiti tramite multiplex compresi i sensori lambda lineari.

5.2.4 Acquisizione dei parametri motore

Come visto nei capitoli precedenti la parte dedicata all'acquisizione dei segnali di ruota fonica e del segnale di fase è demandata all'FPGA, infatti troppo strategica è la rilevazione della posizione angolare e la fasatura del motore per l'accensione e per l'invio di tutte le attuazioni per essere gestita in RT.

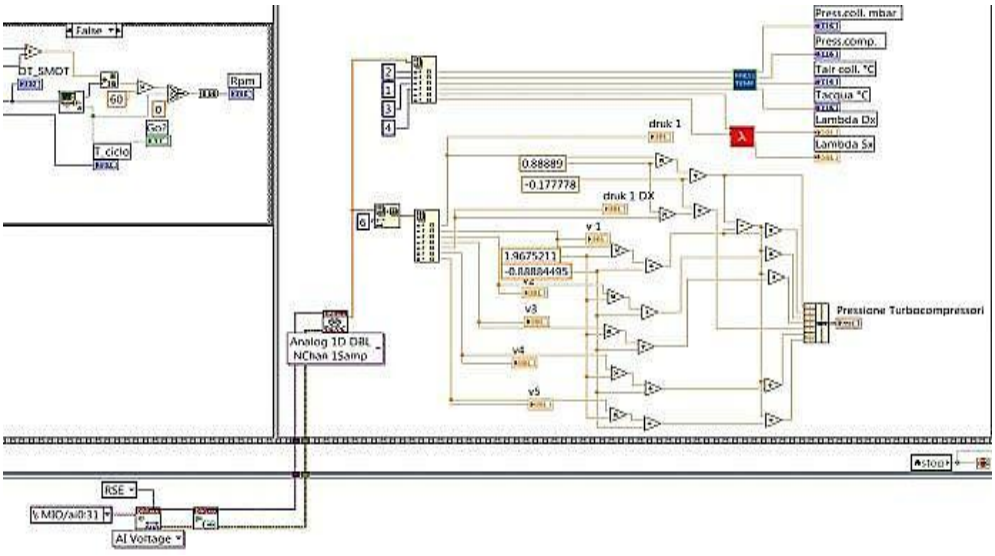


Figura 5.10: Acquisizione parametri.

Quindi la parte che riguarda questi sensori è implementata in *FPGA* mentre invece gli altri parametri necessari al calcolo delle varie tempistiche di attuazione (i cosiddetti parametri per la gestione SPEED-DENSITY-λ) sono rilevati in *RT* con appositi sensori. Le acquisizioni vengono fatte più volte all'interno del ciclo motore, di solito in numero pari al numero di cilindri ogni volta che viene generato un *interrupt* (il numero 10) fra *FPGA* e *RT*. Nella struttura sequence, prima delle acquisizioni c'è una struttura case in cui viene calcolato il numero dei giri RPM in media mobile in base al valore di *DT_SMOT* che viene letto insieme al *T_ciclo* dall'*FPGA*. Le grandezze che vengono monitorate nel ciclo successivo sono le seguenti:

- Pressione aria nel collettore di aspirazione;
- Pressione a valle del compressore;
- Temperatura aria collettore di aspirazione;
- Temperatura del liquido refrigerante;
- Sonde lambda lineari UEGO.

Pur essendo stata inizializzata, l'operazione di lettura dei segnali non inizia fino al blocco *DAQmax Read* contenuto nel *Frame* principale del *Timed Loop*, dove i segnali vengono scomposti, mediati e inviati ai *subVi Temp+ Press* e *letture sonde lambda_m* che contengono le caratteristiche dei sensori. Note le suddette caratteristiche le si implementano i modo da ottenere in uscita il valore desiderato, una volta portato in ingresso alla scheda di tensione misurata.

5.2.5 L'elaborazione delle attuazioni

Questa parte del RT VI ha il compito vero e proprio di controllare il funzionamento del propulsore. L'intera procedura di gestione è

inserita all'interno di un ciclo temporizzato a 1 MHz con dt di 1 µs e priorità 90, inferiore solo a quella del blocco di notifica.

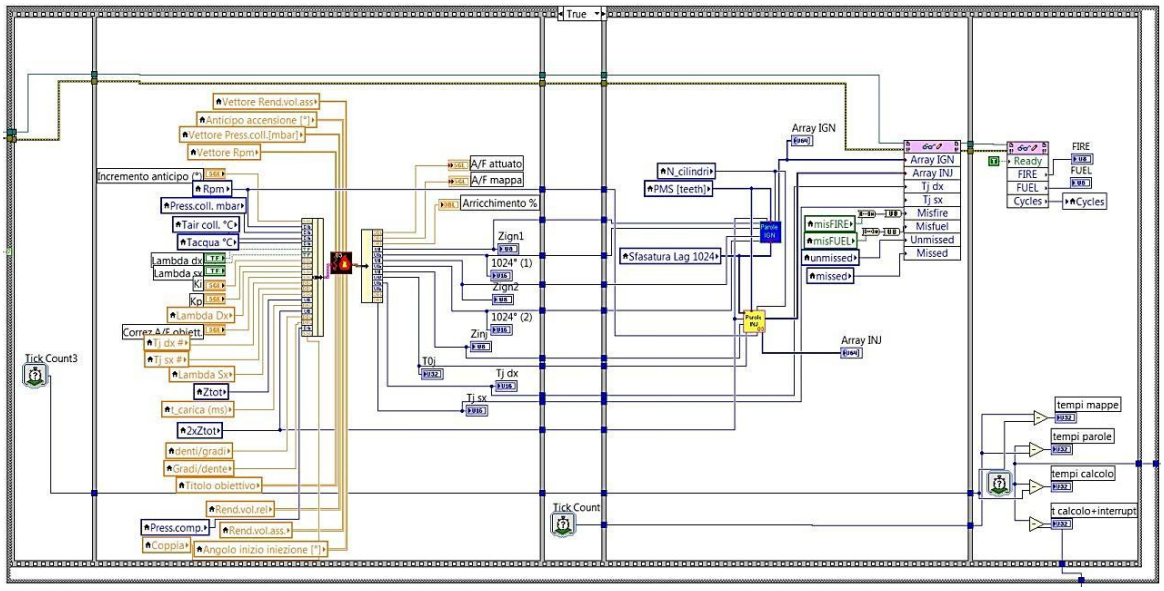


Figura 5.11: Elaborazione attuazioni.

Dopo l'arrivo del flag di notifica dell'*interrupt* 0 inviato dall'FPGA per il calcolo dei parametri di attuazione, vengono letti i dati riguardanti la pressione collettore, la temperatura aria nel collettore, la temperatura del liquido di raffreddamento, il numero di giri del propulsore e il valore di A/F bancata per bancata. Questi dati provenienti dal motore vengono portati in ingresso ad un sub VI *Elaboratore comandi ALL*, insieme ad altri parametri tipici del motore controllato, come *Z tot* e *t_carica bobina*, e a parametri di controllo che possono essere modificati da *Front Panel*, come *Incremento anticipo* (per modificare il valore d'anticipo d'accensione per tutti i cilindri), *Correzione A/F* (per modificare il valore di A/F obiettivo), *Tj%* (per impostare durate di iniezione maggiori o minori del 100% del valore teorico calcolato, bancata per bancata), *Lambda dx-sx* (per attivare la retro azione delle sonde).

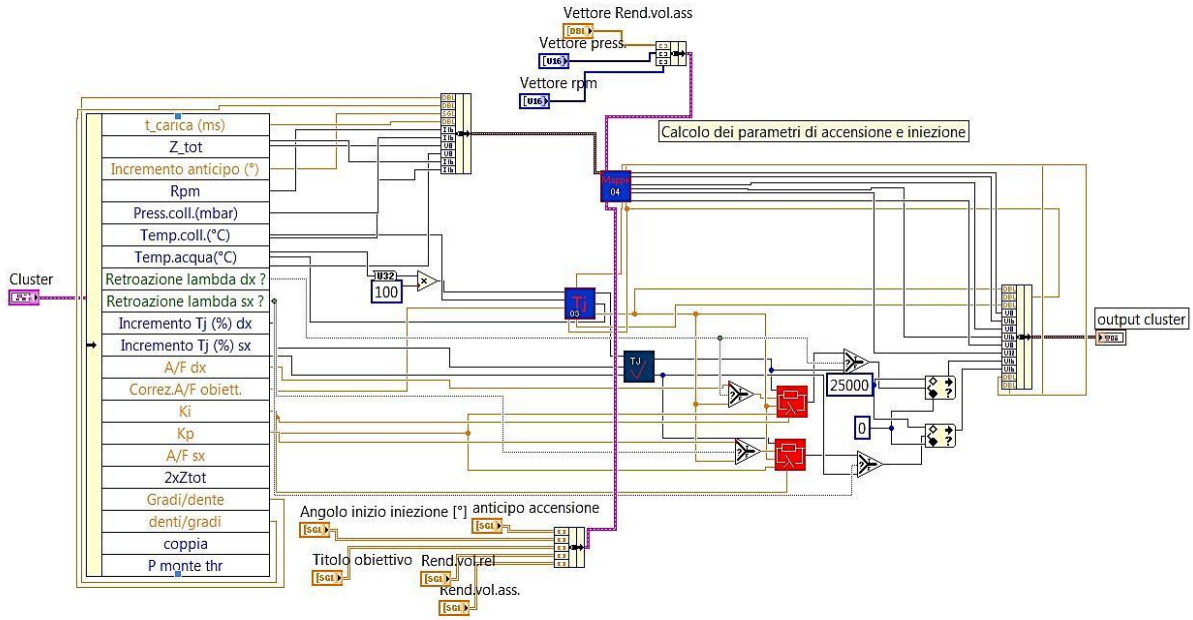


Figura 5.12: Elaboratori comandi all.

Il VI *Elaboratore comandi all* con i dati in ingresso visti fornisce in uscita i valori di riferimento per le attuazioni di accensione e iniezione oltre ai valori riguardanti la Waste Gate:

- *Zign1*: denti di posticipo di inizio carica bobina rispetto a PMS attivo;
- *1024esimi (1):* ritardo in termini angolari di inizio carica;
- *Zign2*: denti di posticipo di fine carica bobina rispetto a PMS attivo;
- *1024esimi (2):* ritardo in termini angolari di fine carica;
- *Zinj*: denti di posticipo inizio iniezione rispetto a PMS attivo;
- *T0j*: ritardo di inizio iniezione in termini di ticks;
- *Tj dx-sx*: tempi di iniezione calcolati per la bancata destra e sinistra;
- *T up e T down duty cycle*: tempo di ON e OFF pierburg;

Inoltre vengono anche ricavate le seguenti informazioni mostrate nel *Front Panel*:

- A/F mappa: valore di A/F calcolato dalla mappa;

- A/F attuato: valore di A/F attuato con le correzioni;
- Arricchimento %: indica l'arricchimento di combustibile quando il motore non è in temperatura.

All'interno del VI sono presenti altri *subVI*, con struttura gerarchica, che verranno descritti in seguito.

Nel primo di questi, Mappe, sono implementate appunto tutte le mappe del motore controllato, quali rendimento volumetrico assoluto e relativo, titolo obiettivo (inteso come $(A/F)_{obiettivo}/(A/F)_{stechiometrico}$), angolo anticipo accensione (riferito al PMS attivo), angolo inizio iniezione (riferito al PMS passivo) e i valori di *Duty Cycle* e apertura della Waste gate.

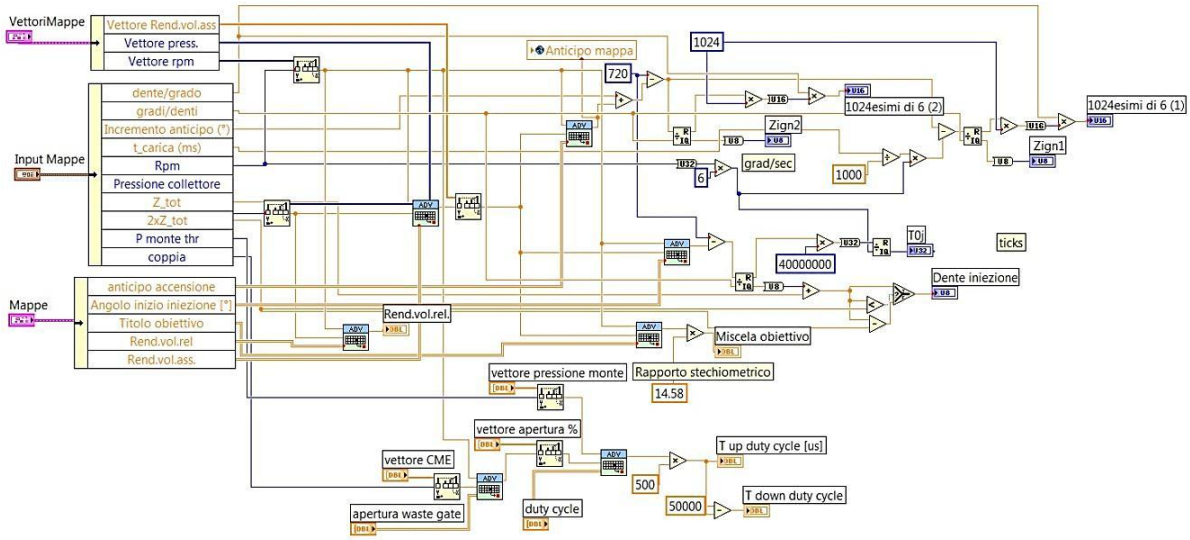


Figura 5.13: Mappe VI.

I dati in ingresso *RPM* e *P coll* vengono indicizzati nei blocchi *Index Array* secondo un vettore che corrisponde ai valori discreti utilizzati per la mappa in cui si deve entrare, così come i valori che, ricavati da una mappa, servono come ingresso in un'altra, come ad esempio il *Rendimento volumetrico assoluto*. Le mappe vengono fornite sotto forma di array bidimensionali: in ingresso ai blocchi *Look Up Table* con gli indici dei valori di ascissa e ordinata: viene restituito in uscita un valore della grandezza

desiderata ricavato come una interpolazione bilineare. Nella figura 5.13 viene mostrato il VI nella sua complessità, mentre in figura 5.14 viene mostrata la logica con cui incrociamo i valori di mappa per ottenere i risultati che ci servono per le varie attuazioni.

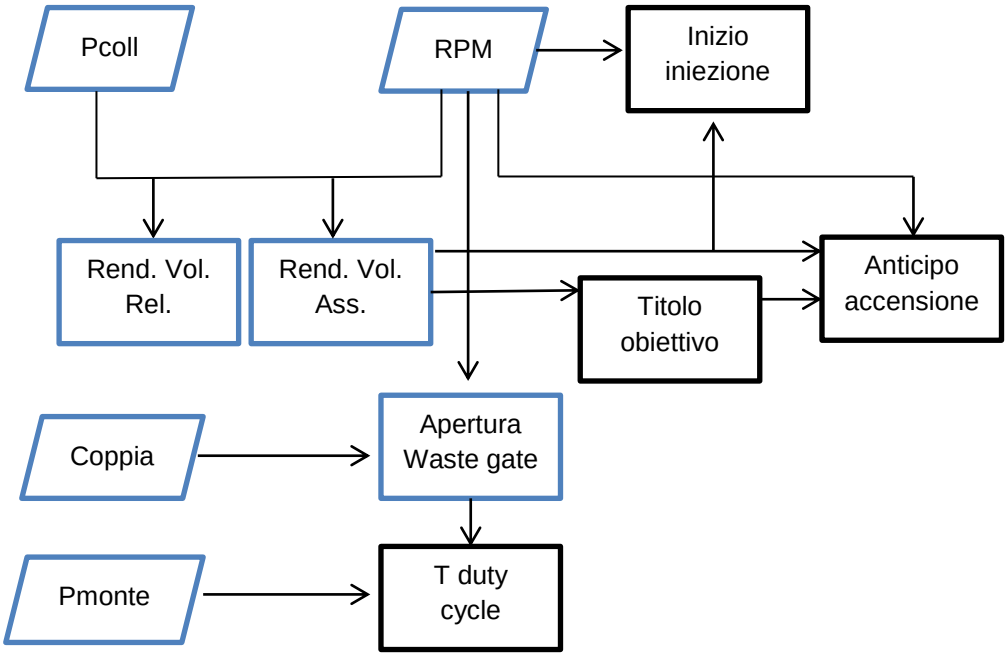


Figura 5.14: Schema calcolo parametri da mappa.

Con le grandezze *Pressione collettore*, *Temperatura aria collettore*, *Rendimento volumetrico relativo* e *Temperatura acqua* si entra in un altro *subVI Tempo iniezione*, all'interno del quale si vuole determinare appunto, nota la caratteristica dell'iniettore, il tempo d'iniezione. In questo VI viene poi corretto, a seconda del valore impostato, il valore di A/F obiettivo. Nel blocco *formula node* si calcola la densità dell'aria nel collettore d'aspirazione secondo la formula dei gas perfetti, nel secondo si passa alla massa d'aria aspirata in ogni cilindro tenendo conto del

rendimento volumetrico relativo. Infine dal rapporto A/F obiettivo, sommato al parametro Correzione A/F obiettivo, si calcola la massa di benzina da iniettare in un cilindro, moltiplicandola poi per un milione per convertirla in milligrammi; poi attraverso la caratteristica dell'iniettore si trova il tempo di iniezione in millisecondi necessario ad immettere la massa di benzina voluta. L'uscita dall' ultimo blocco viene poi moltiplicata per 1000, poiché il tempo di iniezione viene attuato in microsecondi, e per un coefficiente moltiplicativo che aumenta al massimo del 50% il tempo di iniezione a motore freddo (in base alla temperatura del liquido di raffreddamento).

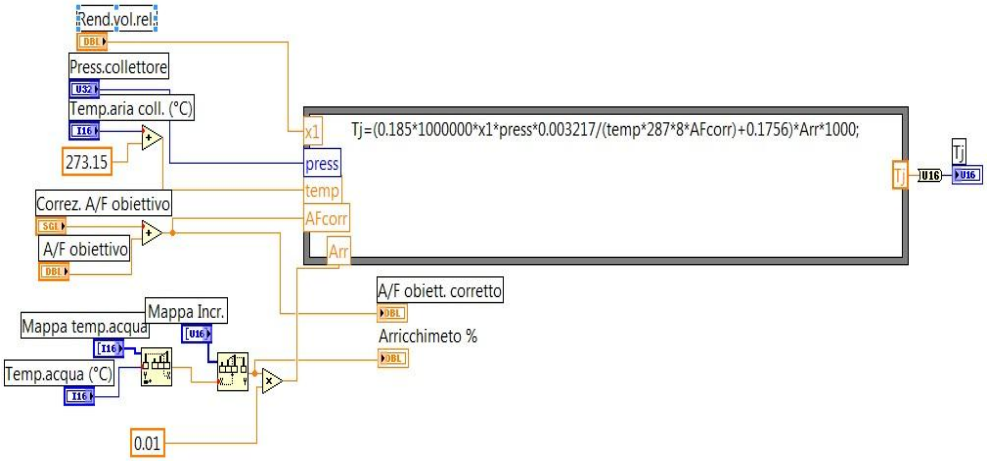


Figura 5.15: Tempo Iniezione.

Un VI dedicato*, Correzione manuale tempo iniezione*, incrementa o decrementa il tempo di iniezione, bancata per bancata, estrapolato da mappa, a seconda del valore impostato sul pannello di controllo. Il tempo di iniezione viene poi trattato dai VI *Retroazione lambda*, che, una volta abilitato il controllo, lo correggono in funzione dell'errore tra A/F obiettivo e A/F rilevato dalla sonde lineari applicate agli scarichi. Il controllore implementato è di tipo proporzionale - integrale (PI) con un blocco di saturazione che fissa l'incremento o decremento

percentuale massimo al 15%. Una volta ricavate tutte le grandezze calcolate, queste vanno sfasate per ogni cilindro in base alla posizione di attuazione relativa nel ciclo motore, che dipende appunto dal cilindro attuato e il corrispondente dente di riferimento per il punto morto superiore attivo (di accensione).

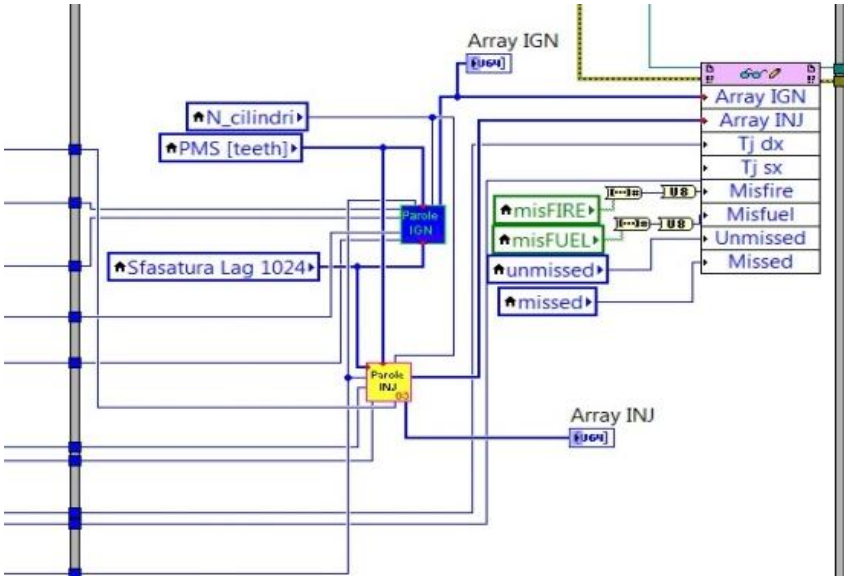


Figura 5.16: Terzo frame blocco calcolo.

In particolare attraverso due VI denominati *ZIGN&Parole IGN* e *Parole INJ_03* andiamo a riportare le attuazioni su ognuno dei cilindri attivi del motore e andiamo a determinare il valore delle matrici *Arary IGN* e *Array INJ* che forniscono le informazioni all'*FPGA* per la gestione e l'invio dei segnali di attuazione. Vedendo nel dettaglio i due VI abbiamo che per quanto riguarda *ZIGN&Parole IGN* (l'accensione sostanzialmente) viene eseguito un ciclo for per 8 volte (cioè il numero cilindri) in cui, in ogni esecuzione, viene elaborata per ogni cilindro la parola per l'esecuzione del carico bobina. In INPUT abbiamo il vettore dei PMS (riferito ai denti) che va a effettuare un offset su tutte le altre attuazioni. Scrivo in pratica la matrice accensione a 64 bit dove i primi 32 indicano: 16 bit il n° del cilindro e 16 bit il dente di inizio e

fine (rispettivamente i primi 8 bit e secondi 8 bit). Gli ultimi 32 bit le sfasature del primo e del secondo dente (rispettivamente i primi e i secondi 16 bit)

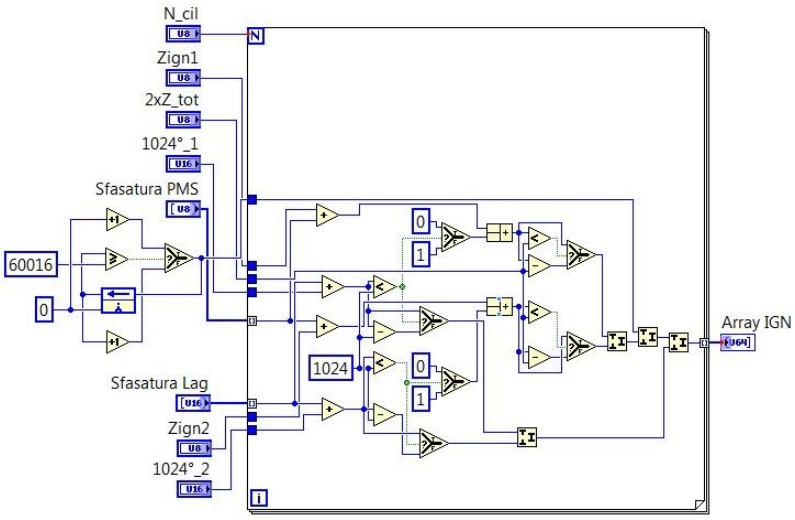


Figura 5.17: ZIGN&Parole IGN.

Per quanto riguarda il VI *Parole INJ* abbiamo che analogamente all'accensione viene eseguito un ciclo *FOR* 8 volte (numero di cilindri) in cui ogni ciclo viene aggiunto lo sfasamento angolare tipico di per ogni cilindro e composta la matrice con le parole necessarie all'*FPGA* per recepire l'insieme delle informazioni necessarie all'iniezione (*Array INJ*).

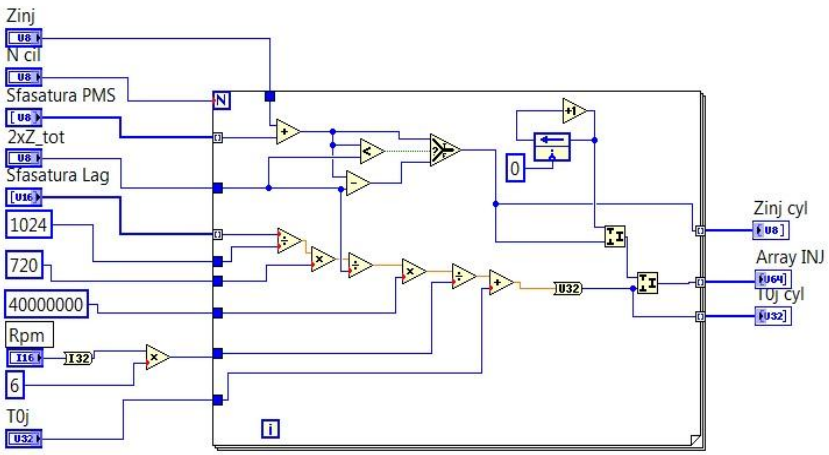


Figura 5.18: Parole IJN_03.

La parola viene implementata utilizzando 64 bit: i primi 32 indicano rispettivamente: 16 bit il cilindro in cui iniettare e 16 bit il dente di inizio iniezione; gli ultimi 32 bit invece indicano il tempo di durata iniezione.

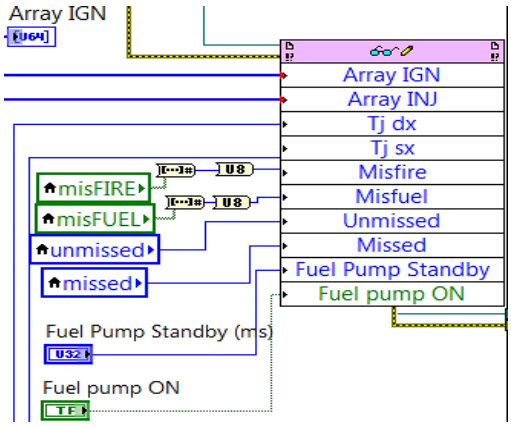


Figura 5.19: invio dati all'*FPGA*.

Tutte queste informazioni vengono poi successivamente inviate al blocco *FPGA* per l'esecuzione delle attuazioni, in particolare come mostrato da figura 5.19 oltre alle due parole create per le attuazioni, i tempi di correzione tramite sonda lambda e le informazioni elaborato dal controllo *missed* e *unmissed*. È inoltre qui presente il comando di controllo pompa e di tempo di attesa per lo spegnimento del dispositivo in caso di non funzionamento (vedi capitolo 4).

Nel quarto frame invece andiamo così a inizializzare l'applicazione e a verificare l'effettiva scrittura dei parametri attuati, ma soprattutto andiamo a calcolare i tempi necessari al calcolo e all'esecuzione del loop utilizzando i riferimenti temporali dei *tick count* interposti fra i vari *Frame* dell'applicazione. In particolare le variabili *tempi mappe, tempi parole, tempi calcolo e t calcolo+interrupt* ci forniscono in maniera rapida e istantanea la velocità con cui stiamo calcolando i vari parametri e ci forniscono un monitoring sulla velocità dell'applicazione.

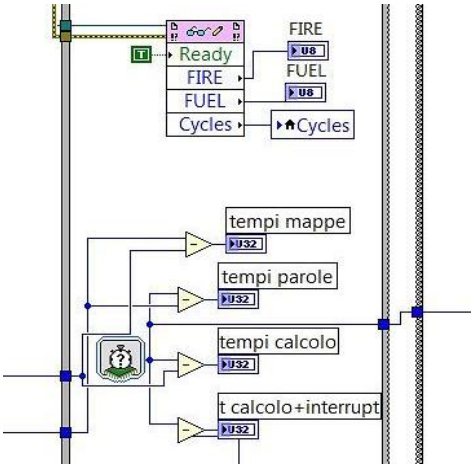


Figura 5.20: Tempi calcolo.

5.2.6 Stop applicazione

In ogni Timed loop del Vi è presente un comando di stop per interrompere l'esecuzione dell'applicazione. L'ultimo blocchetto, *Reset stop button*, ha il compito di chiudere l'*FPGA VI* e resetta il tasto di *STOP* a *false* per un avvio successivo solo dopo che tutti gli altri *Loop* hanno terminato la loro esecuzione. Inoltre richiamiamo la funzione *Reset* del blocco *Invoke Method* per riportare l'*FPGA VI* al suo stato di default, oltre a un blocco per verificare la presenza di eventuali errori.

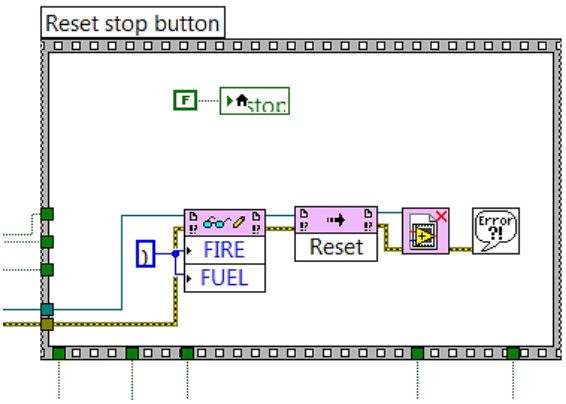


Figura 5.21: Reset stop button.

5.3 Lettura da FIFO

Si fa riferimento in pratica al paragrafo 4.8 in cui andiamo a salvare i dati nella FIFO in *FPGA.* In real time andiamo prima di tutto a richiamare la FIFO e a stabilirne la profondità attraverso l'assegnazione di una costante in ingresso al Depth.

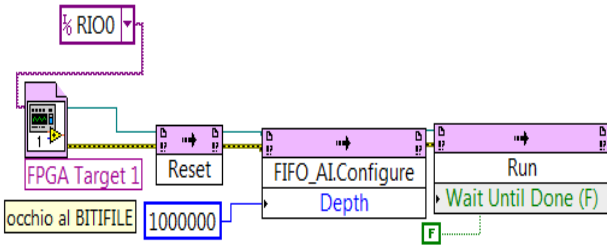


Figura 5.21: Richiamo Fifo.

Successivamente abbiamo implementato un blocco dati di lettura dalla FIFO in cui è possibile anche effettuare il salvataggio dei dati.

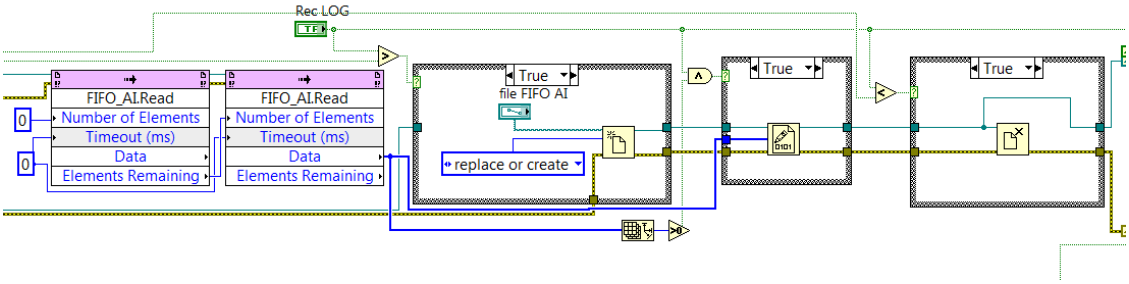


Figura 5.21: salvataggio dati FIFO.

Con il blocco *Invoke Method* FIFO AI *Read* andiamo a scaricare i dati, che vengono scaricati in un vettore della profondità massima definita dal *Depth* impostato in precedenza. Il salvataggio di questi dati è semplice, infatti all' attivare del pulsante booleano *rec Log* inizia la scrittura della FIFO che termina nell'istante in cui *rec Log* viene riportato a *False.*

Quando abbiamo controllato la funzionalità della scrittura dati , in particolare abbiamo verificato se effettivamente i valori presenti nel file erano quelli che ci si aspettava. Innanzitutto è importante la posizione del cosiddetto MARKER. In prima battuta i dati non corrispondevano poiché era presente un errore riguardo la velocità di scaricamento e di aggiornamento dei valori che provengono dall'FPGA. Per risolverlo si è agito su tre valori:

1. *Loop TIMER AI*: in FPGA influenza la velocità di attesa in ticks che si attendono prima di andare a scriver i miei valori che possono essere da 8 a 12 per ciclo. Impostando un valore di 400 ticks in pratica ogni (1/40000000) * 400= 0.00001 sec cioè 100 kHz vado a scrivere il mio pacco di valori (ovviamente uno per volta, siamo in FPGA)

2. *Depth*: In *RT*, indica quanto deve essere profondo il vettore dei dati FIFO che riesco a scaricare in real time (in pratica, l'FPGA va molto più veloce e scrive un dato alla volta, il real time va molto più piano, ma scarica più dati contemporaneamente) Abbiamo impostato un *Depth* di 1 milione.
3. La frequenza del *Timed loop* in cui ho inserito appunto il mio scaricamento dei dati dalla FIFO. In pratica indica ogni quanto scaricare un vettore profondo quanto il *Depth* dalla FIFO. Abbiamo impostato un tempo di 100000 che corrisponde (considerando un tempo globale del VI Rmain2 di 1MHz) a una frequenza di 1MHZ/100KHz=10Hz.

In pratica deve avvenire che ragionando come esempio su un arco temporale di 1 sec che:

1/*LoopTimerAi* (frequenza dati) * *N°dati*(consideriamo 8<10<12) = 100000Hz*10 = 1000000 (dati al secondo)

deve essere minore di:

10Hz(velocità del *Timed Loop)* * 1000000(depth) = 10000000 (dati al secondo).

Questo esempio dimostra come impostando il valore di 400 ticks come *Loop timer Ai* riusciamo a stare dentro i margini. Per verificare tuto ciò al momento del *Run* del VI bisogna verificare che il *TempoACQ* sia uguale a quello impostato in *Loop Timer.* Con le prove effettuate siamo riusciti a superare questo livello impostando un *Loop Timer* di 189 ms.

Capitolo 6

Interfaccia Motore VECU

6.1 Introduzione

Questo capitolo nasce dall'esigenza di fornire a chi si occuperà di sviluppare in futuro il controllo motore sul banco prova del Maserati, tutte quelle informazioni chiave per la connessione fra il sistema fisico e il controllo del motore descritto nei capitoli precedenti tramite l'ambiente LabVIEW. Infatti molto tempo è stato impiegato per svolgere tutte le operazioni di controllo dei sistemi che elaborano i segnali e li rendono disponibili per essere acquisiti tramite gli strumenti presenti in laboratorio. Tutti i vari sensori del motore prima di rendere le proprie informazioni utili al controllo motore devono subire un condizionamento del segnale per essere così utilizzabili per il nostro controllo. E con la medesima logica, tutte le attuazioni che dalla nostra scheda 7833R inviamo dal PXI al motore (da quelle di accensione e iniezione ai controlli pierburg e pompa benzina) devono essere elaborate in modo che il segnale 0-5 V in uscita dalla scheda diventi effettivamente un comando per i vari dispositivi considerati.

6.2 I sensori del motore

Di seguito daremo alcune caratteristiche dei sensori che dal Maserati vengono acquisiti fino alla scheda di acquisizioni dell'FPGA direttamente dal motore.

6.2.1 Sensore regime motore

Il sensore è fissato al basamento del motore lato puleggia comando accessori. Il sensore è costituito da un astuccio tubolare al cui interno si trova ·un magnete permanente ed un avvolgimento elettrico. Il flusso magnetico creato dal magnete subisce, a causa del passaggio dei denti (60 - 2) della ruota fonica, delle oscillazioni conseguenti alla variazione di traferro. Tali oscillazioni inducono una forza elettromotrice nell'avvolgimento ai cui capi si viene a trovare una tensione alternativamente positiva (dente affacciato al sensore) e negativa (cava affacciato al sensore). Il valore di picco della tensione in uscita dal sensore dipende, a parità di altri fattori, dalla distanza tra sensore e dente (traferro). La resistenza elettrica del sensore può essere misurata scollegando il connettore e collegando un ohmmetro.

Resistenza: 650 Q ± 10% a 20° C

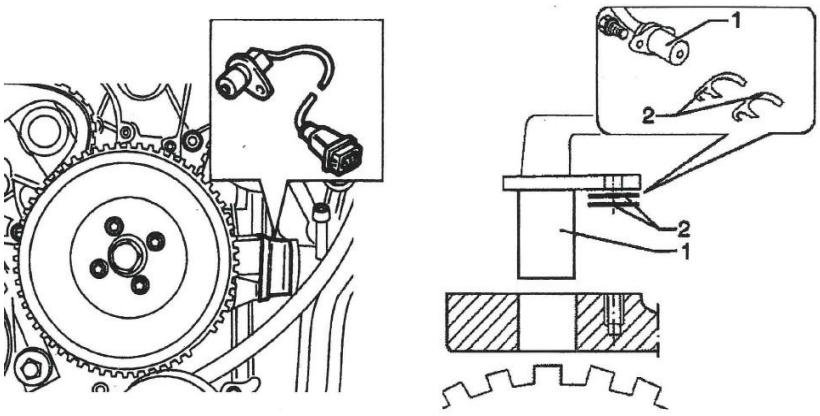


Figura 6.1: Sensore di giri.

6.2.2 Sensore fase del motore

Il segnale di fase motore è generato da un sensore ad effetto Hall, montato in corrispondenza della puleggia comando albero distribuzione bancata destra. Uno strato semiconduttore,

percorso da corrente immerso in un campo magnetico normale ad esso (linee di forza perpendicolari al verso della corrente), genera ai suoi capi una differenza di potenziale, nota come tensione di Hall. Se l'intensità della corrente rimane costante, la tensione generata dipende solo dall'intensità del campo magnetico. E' sufficiente quindi che l'intensità di campo vari periodicamente per ottenere un segnale elettrico modulato. Per ottenere questo cambiamento, il sensore viene fatto attraversare da un anello metallico (solidale con la parte interna della puleggia distribuzione) dotato di una apertura. Nel suo movimento, quando l'anello copre il sensore, blocca il campo magnetico ed il segnale rimane basso, mentre in corrispondenza dell' apertura, il campo si richiude ed il segnale diventa alto.

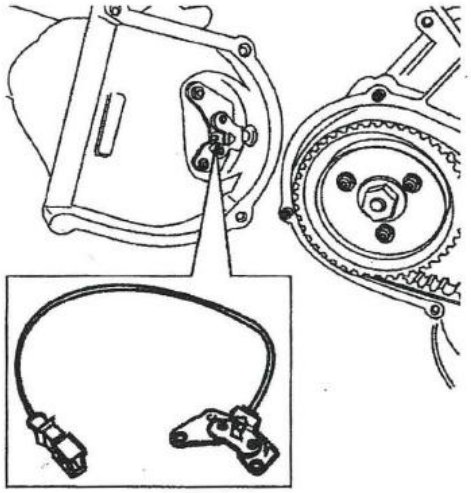


Figura 6.2: Sensore di fase.

6.2.3 Sensori pressione assoluta

I sensori di pressione assoluta sono posizionati rispettivamente:

- sotto i teleruttori impianto (sensore a monte farfalla);
- sulla paratia vano motore (sensore a valle farfalla).

Il sensore che incorpora un elemento sensibile, racchiuso in un contenitore in materiale plastico, è costituito da un ponte di resistenze (Wheatstone) serigrafate su una piastrina ceramica molto sottile (diaframma) montata sulla parte inferiore di un supporto a forma anulare. Il diaframma separa due camere: nella camera inferiore, sigillata, è stato creato il vuoto; mentre la

camera superiore, tramite una tubazione è in diretta comunicazione con il collettore di aspirazione. In funzionamento la pressione, che si genera nel punto di misura, produce un'azione meccanica sul diaframma del sensore, che flette facendo variare il valore delle resistenze. Essendo costante (+5 V) la tensione di alimentazione, la variazione delle resistenze provoca una variazione della tensione (V) di uscita, proporzionale al valore della pressione.

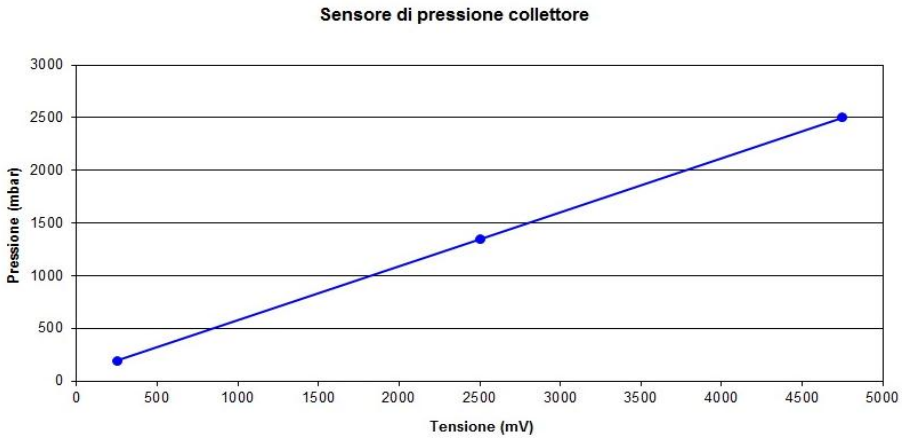


Figura 6.3: Caratteristica sensore di

Sensore a monte farfalla:

Alimentato dalla ECU, viene collegato pneumaticamente ai condotti di aspirazione a monte della farfalla OBW. Il segnale di risposta, permette alla ECU di conoscere:

- la pressione atmosferica (altitudine);
- la pressione istantanea dei turbocompressori.

E' possibile così attuare:

- la correzione del tempo iniezione in funzione della quota;
- il controllo della pressione di alimentazione in maniera ottimale.

Sensore a valle farfalla:

Alimentato dalla ECU, viene collegato pneumaticamente al collettore a valle farfalla. Il segnale associato a quello di temperatura aria, serve a calcolare la densità aria.

6.2.4 Sensori temperatura liquido refrigerante

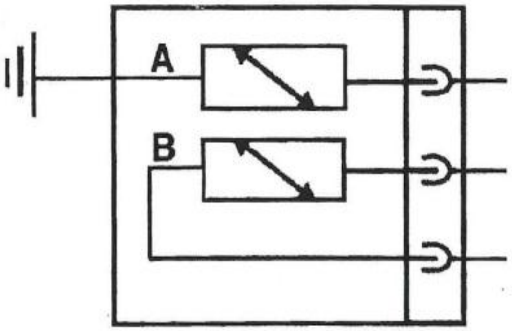


Figura 6.4: schema elettrico.

Il sensore è installato sul termostato. E' formato da un corpo in ottone che funge da protezione agli elementi resistivi veri e propri, costituiti da due termistori del tipo NTC (Coefficiente di Temperatura Negativo, la cui resistenza elettrica diminuisce con l'aumentare della temperatura). I due termistori sono distinti: il termistore (A) non è collegato, il termistore (B) fornisce l'informazione di temperatura alla centralina controllo motore. Per quest'ultimo, la tensione di riferimento è di 5 Volt, poiché il circuito di ingresso in centralina è progettato come divisore di tensione, la tensione di riferimento è ripartita tra una resistenza presente in centralina ed il sensore stesso. Ne consegue che la centralina è in grado di valutare le variazioni di resistenza del sensore attraverso i cambiamenti della tensione, ottenendo così l'informazione di temperatura.

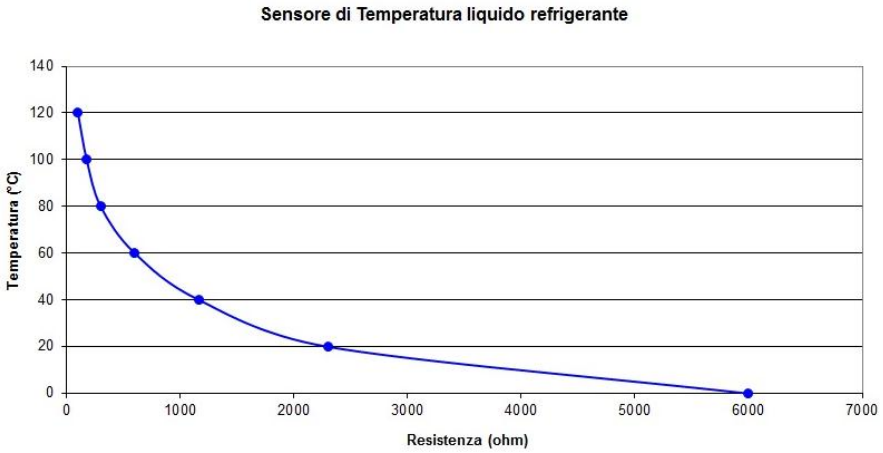


Figura 6.5: Caratteristica sensore temperatura liquido refrigerante.

6.2.5 Sensore temperatura aria aspirata

Il sensore è montato sul lato sinistro del collettore di aspirazione. Esso è formato da una gabbia in materiale sintetico che funge da protezione all'elemento resistivo vero e proprio, costituito da un termistore del tipo NTC (Coefficiente di Temperatura Negativo), il cui valore di resistenza elettrica diminuisce con l'aumentare della temperatura, e dal relativo corpo con connettore. Poiché il circuito di ingresso in centralina è progettato come divisore di tensione, la tensione di riferimento che è di 5 V, viene ripartita tra una resistenza presente in centralina ed il sensore stesso. Ne consegue che la centralina è in grado di valutare le variazioni di resistenza del sensore attraverso i cambiamenti della tensione, ottenendo così l'informazione di temperatura. Questa informazione, unitamente all'informazione di pressione assoluta**,** viene utilizzata dalla centralina elettronica per stabilire la "densità aria aspirata" che è un dato essenziale per poter risalire alla quantità di aria aspirata dal motore, in funzione della quale il calcolatore stesso dovrà elaborare il tempo di iniezione.

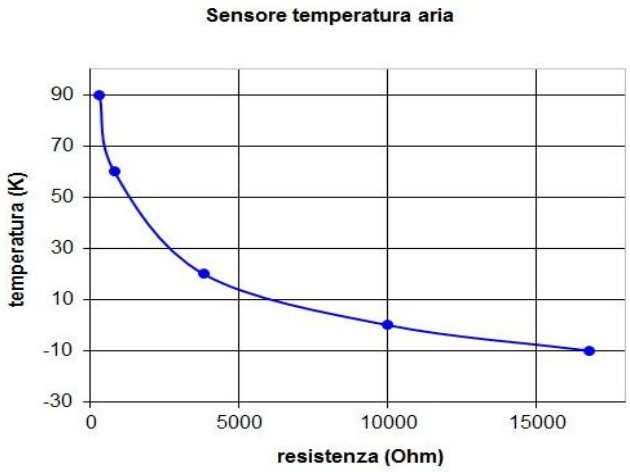


Figura 6.6: Caratteristica sensore temperatura aria.

6.2.6 Sonde lambda

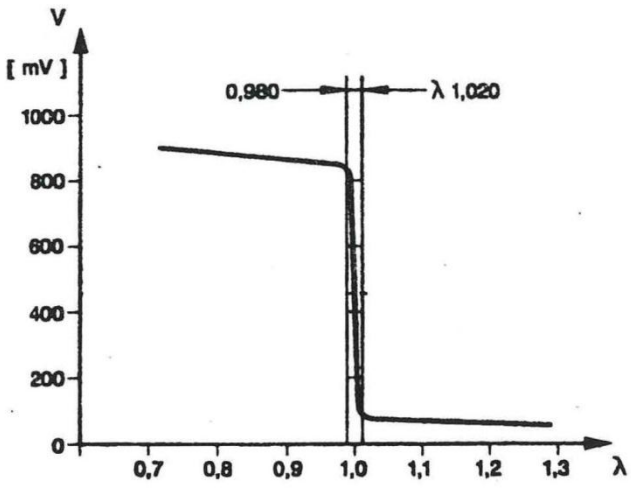


Figura 6.4: segnale di risposta.

Le sonde lambda misurano il contenuto di ossigeno nei gas di scarico. Viene montata una sonda su ciascun primo tratto della tubazione di scarico, immediatamente a monte del precatalizzatore. Il segnale di uscita del sensore viene inviato alla centralina per la correzione in retroazione (feedback) del titolo della miscela. Quando la sonda fornisce un segnale basso (tensione inferiore a 200 mV) la centralina riconosce un titolo magro ed incrementa il tempo di iniezione; successivamente quando il segnale della sonda è alto (tensione superiore a 800 mV), la centralina riconosce un titolo ricco e decrementa il tempo di iniezione. Questa sequenza di interventi si ripete con una frequenza dell'ordine di decine di Hertz, in modo che il motore funzioni con un titolo continuamente oscillante attorno a quello stechiometrico. Per temperature inferiori a 300°C il · materiale ceramico non è attivo, quindi la sonda non invia segnali attendibili. Per assicurare un rapido riscaldamento all'avviamento e mantenere la temperatura al minimo, la sonda è dotata di un riscaldatore a resistenza elettrica. La centralina controlla la corrente di alimentazione del riscaldatore di entrambe le sonde lambda in modo da portare rapidamente la sonda a regime e mantenerne la temperatura a livelli ottimali.

6.3 Dispositivi di esecuzione attuazioni

Sono quei dispositivi che tramutano i segnali in uscita dalla centralina di serie o nel nostro caso dalla VECU alle attuazioni che permettono il funzionamento del motore. Si parla quindi delle caratteristiche degli elementi che, comandati elettricamente, eseguono quelle operazioni che permettono al motore di funzionare. In particolare di iniettori, bobine, pierburg e pompa della benzina.

6.3.1 Bobina accensione singola

L'alta tensione è fornita da otto bobine singole montate direttamente sulle candele (bobine pencil-coil). Il circuito di accensione è del tipo statico a scarica induttiva. L'anticipo ottimale di accensione viene calcolato dalla centralina controllo motore in funzione del regime e del carico motore, e viene attuato sotto forma di tempo tra il PMS di scoppio e l'istante in cui interrompere l'alimentazione del circuito primario di ciascuna bobina. La bobina utilizzata è del tipo a circuito magnetico chiuso, con gli avvolgimenti posti in un contenitore in plastica ed immersi in resina epossidica. La bobina è collegata direttamente alla candela tramite una prolunga in materiale siliconico avente elevate caratteristiche dielettriche. Essendo isolata dalla testa cilindri il circuito secondario è collegato a massa con un apposito cavo.

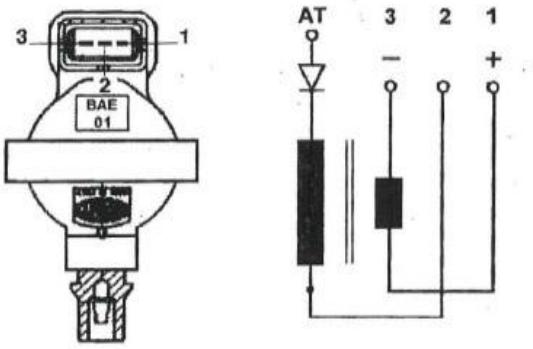


Figura 6.7: schema elettrico.

Figura 6.8: foto bobina.

6.3.2 Elettroiniettori (IW 058)

L' elettroiniettore ha il compito di erogare la quantità di combustibile, necessaria al funzionamento del motore, che viene iniettato nel condotto di aspirazione, immediatamente a monte delle valvole di aspirazione. L'iniettare è del tipo "top-feed", con alimentazione del combustibile dalla parte posteriore del corpo (1), dove è anche alloggiato l'avvolgimento elettrico (5) collegato al connettore (6). Quando l'avvolgimento viene percorso da corrente, il campo magnetico che si crea attira l'ancoretta (3) e di conseguenza l'otturatore (2) determinando l'apertura dell'iniettore ed il passaggio di combustibile. Al cessare della corrente, l'otturatore viene richiamato dalla molla (4).

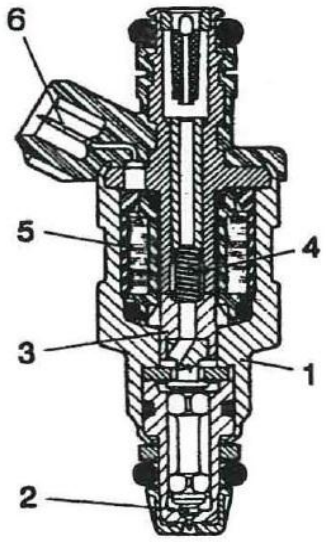


Figura 6.9: elettroiniettore.

Essendo costante il differenziale di pressione tra interno ed esterno dell'iniettore (grazie alla presenza del regolatore), la

quantità di combustibile erogata, a parità di tensione elettrica di alimentazione, dipende solo dal tempo di apertura, stabilito dalla centralina controllo motore.

Figura 6.10: foto elettroiniettore.

Resistenza: 14.5 Ω ± 10%
Portata: 0.43 l/min.

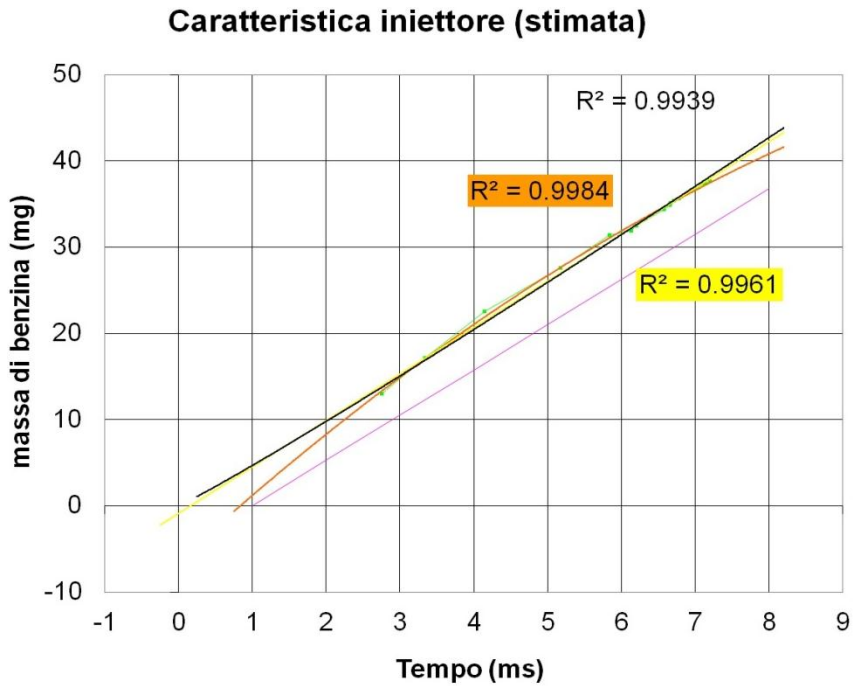


Figura 6.11: Caratteristica iniettore.

6.3.3 Elettropompa combustibile

Il filtro è inserito lungo la tubazione di mandata combustibile, esternamente al serbatoio. Esso è formato da un involucro in lamierino di alluminio e da un supporto interno in poliuretano sul quale è avvolto un elemento ad elevato potere filtrante. L'impianto è poi dotato di due elettropompe alloggiate all'interno del serbatoio combustibile in un apposito cestello dotato di prefiltro sull'aspirazione. La pompa è di tipo monostadio a flusso periferico ed è adatta a funzionare con combustibile senza piombo. Il rotore è mosso da un motore elettrico in corrente continua alimentato da un teleruttore, su comando della centralina controllo motore.

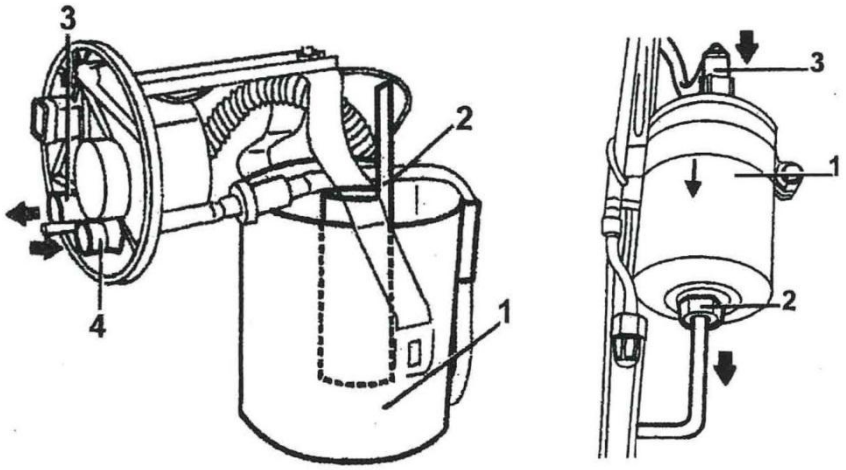


Figura 6.12: Pompa carburante e filtro combustibile.

La pompa è dotata di una valvola di sovrappressione, che cortocircuita la mandata con l'aspirazione nel caso la pressione del circuito di mandata dovesse superare i 5 bar, in modo da evitare il surriscaldamento del motore elettrico. Inoltre, una valvola di non ritorno inserita nella mandata impedisce lo svuotamento dell'intero circuito combustibile quando la pompa non è in funzione.

6.4 Condizionamenti dei segnali

Questo paragrafo tratta di tutte quelle operazioni che sono effettuate per interfacciare i segnali da e per la scheda 7833R, con il nostro Host Pc, con tutto il software impiegato nei capitoli precedenti e il motore che abbiamo il fine di controllare. La figura 6.13 infatti**,** fornisce una ricostruzione grafica di quelli che possono essere**,** in un sistema come il nostro, la serie di connessioni fra il Maserati e la nostra centralina virtuale.

6.4.1 Introduzione

La figura 6.14 (pag. 140) mostra chiaramente il tipo di interfaccia necessaria per la realizzazione del sistema di controllo del Maserati a banco. In particolare partendo dal nostro HOST PC (nella figura è rappresentato in un blocco unico ma dai capitoli 4 e 5 sappiamo che si divide nei moduli *RT* e *FPGA*) gli input/output digitali impostati dal progetto in labVIEW, arbitrariamente scelti dall'utente in fase di programmazione VECU, presentano una serie di ingressi e uscite specifiche di riferimento cablate attraverso il connettore SCB68. Questi ingressi e uscite fanno riferimento alla scheda 7833R. Dal connettore attraverso un cavo a 25 poli portiamo i segnali in uscita verso la scheda di controllo attuazioni (frecce rosse) e verso i circuiti di controllo Pierburg (che comandano le aperture della waste gate) e della pompa combustibile (frecce arancioni). I segnali in ingresso invece hanno due diverse strade per essere acquisiti:

- i segnali di fonica e fase attraverso il circuito di squadratura (alloggiato nella medesima scatola dei controlli Pierburg e pompa) passano attraverso il connettore SCB68 alla scheda 7833P e quindi in FPGA;

- i sensori invece di temperatura aria e acqua, pressione collettore di aspirazione e a valle del compressore, sonda lambda destra e sinistra sono acquisiti direttamente dal PXI senza passare dalla scheda 7833R e quindi da FPGA (freccia viola). Questo avviene tramite la scheda 6259 e il Multiplex DAQ MAX che consente di acquisire direttamente i valori dei vari sensori.

connettore	Colore cavo	pin scheda	attuazione	DI/O FPGA	PIN 7833
1	Marrone	IN GND	IGN GND	TERRA	1
14	Rosa	IN1	IGN1	DIO 8	43
2	Rosa marrone	IN2	IGN2	DIO 9	44
15	Bianco Blue	IN3	IGN3	DIO 10	45
3	Giallo Bianco	IN4	IGN4	DIO 11	46
16	Grigio marrone	IN5	IGN5	DIO 12	47
4	Verde Marrone	IN6	IGN6	DIO 13	48
17	Verde	IN7	IGN7	DIO 14	49
5	Rosso Blue	IN8	IGN8	DIO 15	50
18	Marrone Rosso	IN9	INJ1	DIO 0	35
6	Bianco Nero	IN10	INJ2	DIO 1	36
19	Viola	IN11	INJ3	DIO 2	37
7	Blue	IN12	INJ4	DIO 3	38
20	Giallo Marrone	IN13	INJ5	DIO 4	39
8	Bianco Rosso	IN14	INJ6	DIO 5	40
21	Grigio Bianco	IN15	INJ7	DIO 6	41
9	Blue marrone	IN16	INJ8	DIO 7	42

Tabella 6.13: schema del sistema di connessione per la VECU

La tabella 6.13 mostra i cablaggi fra le varie parti descritte in precedenza, in particolare sono presenti il numero di uscita digitale FPGA dal software, il corrispondente pin di uscita dalla scheda 7833R, il colore dei cavi, il riferimento al connettore, il pin della scheda di comando e il riferimento alle attuazioni.

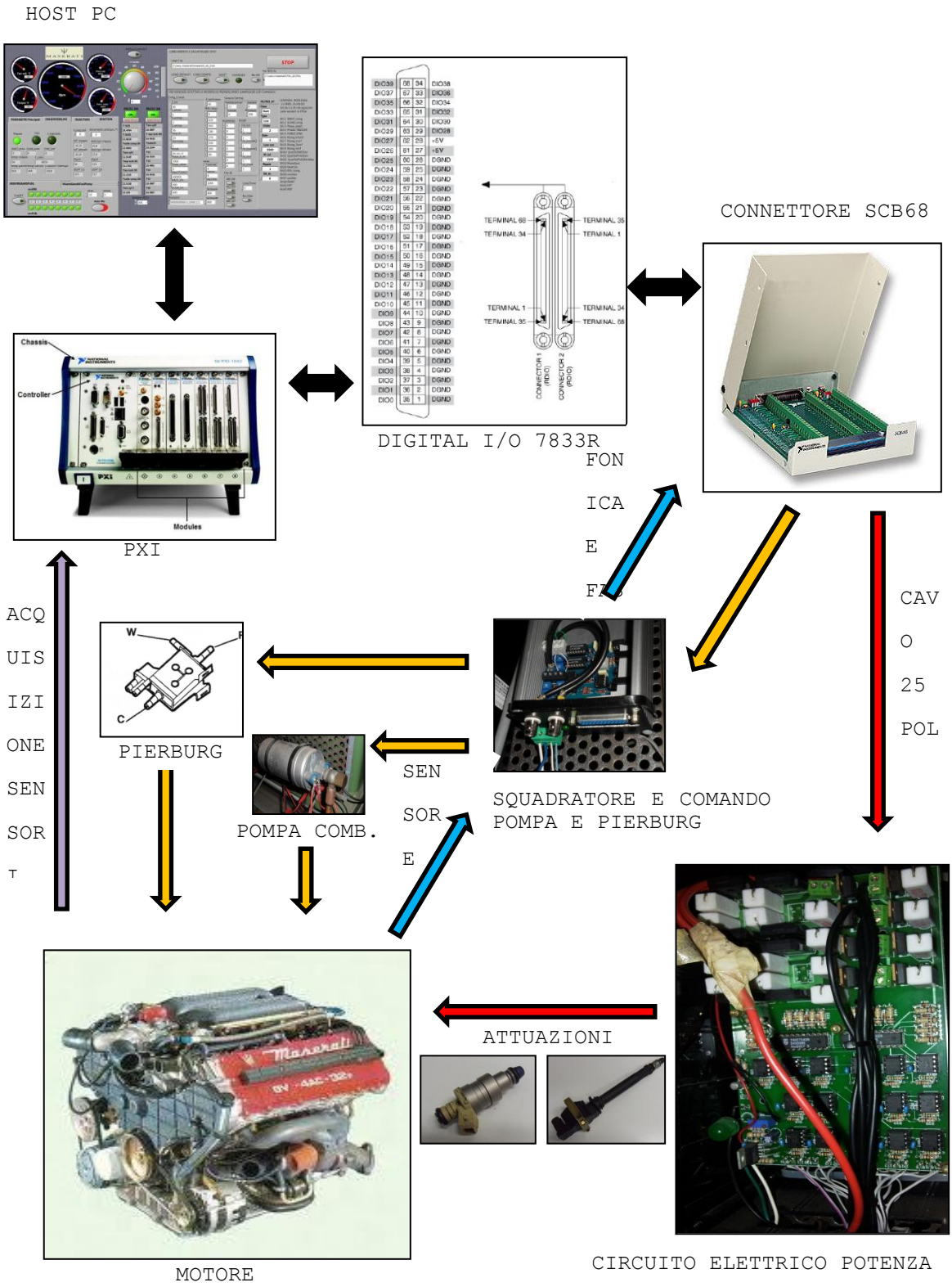


Figura 6.14: schema del sistema di connessione per la VECU

6.4.2 Circuito esterno attuazione VECU

È il circuito che, ricevendo i segnali che provengono dalla scheda 7833R e comandati dalla VECU, trasmette tutti gli input utili a eseguire tutte le attuazioni del motore.

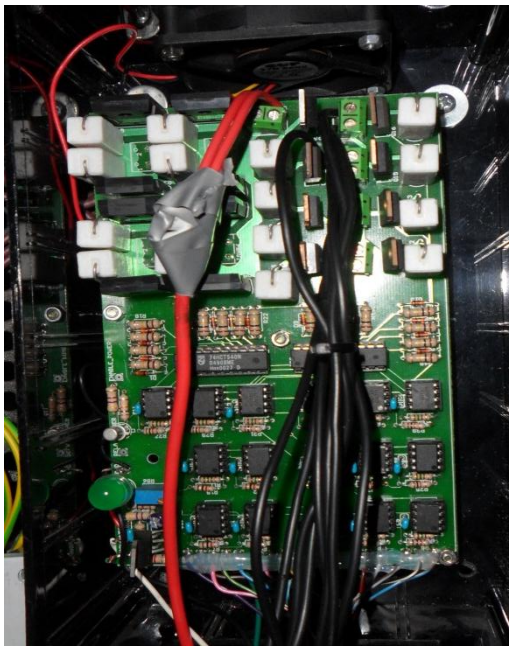


Figura 6.15: Circuito attuazioni

Il circuito in pratica è costituito da una serie di pin di ingresso che ricevono i segnali 0-5V da un cavo a 25 poli in uscita dalla scheda 7833R che identificano le varie attuazioni. Ma entriamo nel dettaglio delle varie connessioni partendo dal quadro segnali del progetto di labVIEW, passando al pin out della scheda 7833R (connettore 1) fino alle attuazioni dei vari cilindri. Sono presenti una serie di componenti elettronici che portano e vanno a condizionare il segnale di uscita dal PXI fino all'attuazione vera e propria. La scheda riceve 12 V come alimentazione da batteria. È presente un regolatore di tensione che va a regolare la tensione a 5 V che sarà poi necessaria per alimentare i componenti del circuito.

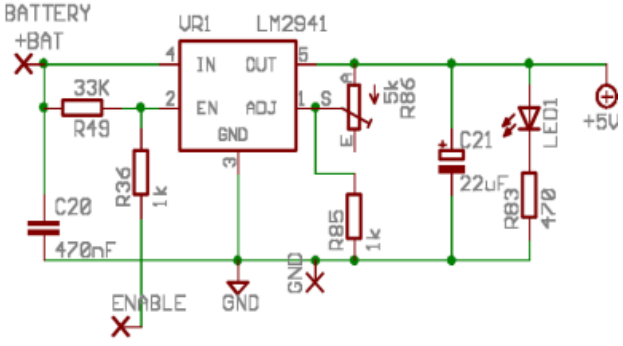


Figura 6.16: schema regolatore di tensione

Vediamo una linea per capirne meglio il funzionamento in particolare la linea IN1 che gestisce l'attuazione di accensione del cilindro 1. In pratica il segnale in uscita dal PXI a 5 V è in parallelo con la terra comune di tutto il sistema ed entra in un circuito opto-isolatore che quando riceve il segnale fornisce in uscita il segnale ribaltato e isolato. Successivamente attraverso un operatore NOT viene nuovamente ribaltato e viene inviato al MOSFET (transistor del tipo BJT) che appena riceve questo segnale ALTO come segnale di Gate abilita il passaggio della corrente tra il Drain e il Source che va effettivamente ad eccitare iniettore e bobina per l'esecuzione delle attuazioni.

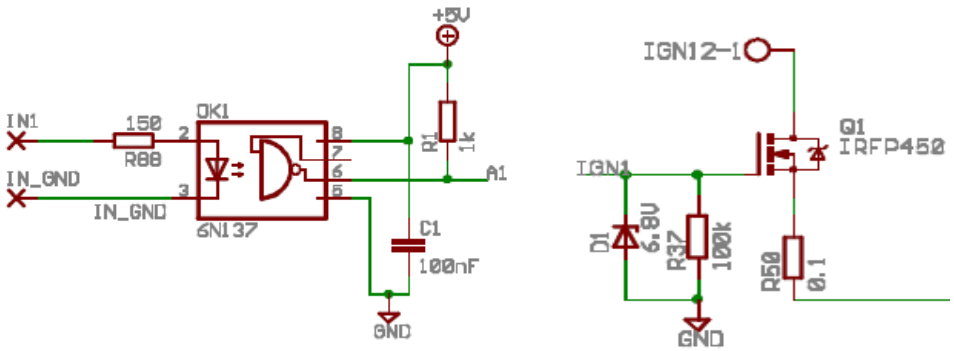


Figura 6.16: Opto-isolatore e MOSFET

6.4.3 Circuito di squadratura

Questi circuiti sono implementati all'interno della medesima scatola che si trova nel panno a fianco dei circuiti di controllo attuazione ed è formato da varie schede aventi ciascuna una specifica e differente funzione.
Iniziamo dal modulo definito "Scheda Madre" (MB) dove è presente l'ingresso dell'alimentazione di tutti i circuiti a 12 V dc e l'uscita di 5 V dc. E' inoltre presente la porta di interfaccia con il PXI che copia la connessione in ingresso tipica del CompactRio.

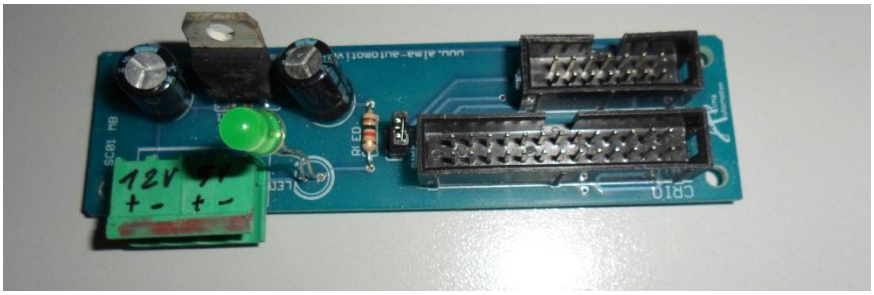


Figura 6.17: dettaglio scheda madre (MB)

PIN	SEGNALE	PIN	SEGNALE
1	GND	14	DIO0
2	NC	15	NC
3	GND	16	DIO1
4	GND	17	DIO2
5	NC	18	NC
6	GND	19	DIO3
7	GND	20	DIO4
8	NC	21	NC
9	GND	22	DIO5
10	GND	23	DIO6
11	NC	24	NC
12	GND	25	DIO7
13	GND		

Figura 6.18: dettaglio connessioni MB

Sulla MB è presente la porta MB (10 poli) che dovrà essere collegata ai circuiti di squadratura e al modulo LSD per le relative alimentazioni e la comunicazione dei segnali opportuni. È presente anche un Jumper GNDCON che mette in comune la terra del PXI con quella dell'alimentazione a 12 V dc.

Sulla scheda destinata alla squadratura dei segnali VRS (segnale ruota fonica) sono presenti due connessioni:

- la porta DB1 (10 poli), che verrà collegata alla MB con apposito cavo piatto, e che servirà ad alimentare il circuito e a portare verso MB (e da lì verso PXI) il segnale squadrato;

- 4 morsetti a vite per l'ingresso dei due segnali provenienti da sensore VRS (cablati poi con cavo BNC).

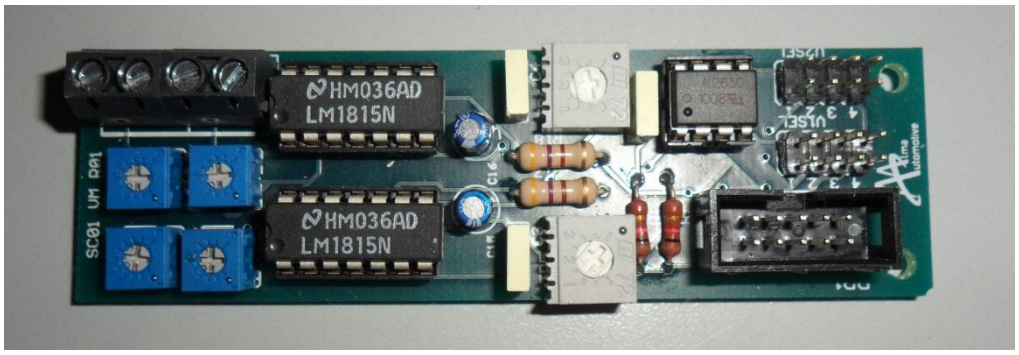


Figura 6.19: dettaglio scheda squadratore

Sono presenti 6 trimmer (condensatori e resistenze): TR1 e TR3 da 100k consentono la regolazione della tensione in ingresso al LM1815N. TR2 e TR4 da 100k sono posizionati sulle terre dei segnali e servono ad isolare, qualora ve ne fosse la necessità, la terra dei segnali da quella della centralina. TR5 e TR6 questi trimmer da 250k sono collegati al pin7 dell'amplificatore LM1815N e consentono di regolare la costante di tempo dell'amplificatore.

Sulla scheda destinata alla squadratura dei segnali effetto Hall (il nostro segnale di albero distribuzione) sono presenti due connessioni:

- la porta DB2 (10 poli), che verrà collegata alla MB con apposito cavo piatto, e che servirà ad alimentare il circuito e a portare verso MB (e da lì verso PXI) il segnale squadrato.
- 4 morsetti a vite per l'ingresso dei due segnali proveniente da sensore effetto Hall.

Sono presenti 4 trimmer: TR7 e TR8 da 10k consentono la regolazione della tensione in ingresso al circuito. TR9 e TR10 da 100k sono posizionati sulle terre dei segnali e servono ad isolare,

qualora ve ne fosse la necessità, le terra dei segnali da quella della centralina

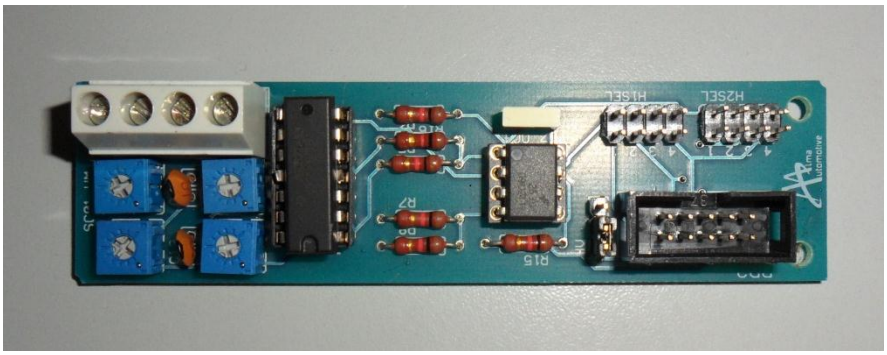


Figura 6.20: dettaglio squadratore effetto Hall

Queste due schede sono collegato alla MB e ricevono in ingresso i segnali da ruota fonica e da sensore di fase tramite cavo BNC e in uscita attraverso cavo piatto forniscono le informazioni di fonica e fase utilizzabili dal PXI attraverso FPGA per la fasatura del motore.

La scheda di comando controllo della pompa è denominata scheda di controllo dei LSDM (low side driver). In questa scheda sono presenti le seguenti connessioni:

- la porta DB3 (10 poli), che verrà collegata alla MB con apposito cavo piatto, e che servirà ad alimentare il circuito.
- 8 morsetti a vite per i quattro segnali comandati.

Ognuna di questa tre schede presenta la possibilità di essere alimentate con 5 o i 12 V dc. Le configurazioni standard l'alimentazione a 12 V dc da batteria (o nel nostro caso dal Kert).

Capitolo 7

Verifica attività e conclusioni

7.1 Introduzione

Il software VECU su piattaforma LabVIEW risulta molto difficile da essere compreso e presenta una mole di lavoro sviluppatasi a lungo nel tempo che ha richiesto molto tempo per la comprensione. Inoltre tutta la parte di connessione fra il sistema virtuale che si stava implementando in centralina e quello reale ha richiesto competenze che è stato necessario apprendere da chi ha esperienza nel campo. Il lavoro svolto rappresenta quindi una parte di sviluppo di un progetto che già aveva avuto negli anni i suoi risultati, che attinge dal lavoro di molte persone.

7.2 Attività principali svolte

La prima cosa è stata la comprensione del modo di operare e delle possibilità base di labVIEW. Successivamente il lavoro forse più lungo è stato quello di capire la logica di lavoro del progetto VECU, così come era stata pensata e immaginata dalle persone che ci avevano messo mano. Ci sono molti modi per ottenere risultati analoghi con LabVIEW e non è stato semplice capire come, chi aveva implementato parte di codice, aveva ragionato nel suo lavoro. Successivamente ci si è scontrati con la parte forse più complessa del lavoro: avendo a disposizione un software che gestisce un sistema fisico come quello motore, il problema era principalmente come andare a fare in modo che le due realtà si interfacciassero fra loro. Cioè, capire come una qualsiasi operazione eseguita su un HOST PC effettivamente si tramutata in una attuazione, oltre che comprendere come una

qualsiasi rilevazione dai sensori motore possa essere utilizzata dalla VECU.

7.2.1 Modifiche Software

Il lavoro di adattamento Software si è suddiviso in blocchi, con il compito di migliorare parti della versione precedente della VECU:

- Aggiornamento del blocco di fasatura: si e in pratica sostituito il VI precedente con una nuova versione capace di rispondere più prontamente alle esigenze di accensione del motore in fase di avviamento. In particolare l'adattamento dell'*FPGA* ha portato a una variazione di tutti i parametri di controllo in *RT*.
- Correzione strategie di avviamento motore dopo la verifica dell'incapacità del sistema di eseguire la fase di avviamento in hardware in the loop.
- Implementazione in *FPGA e RT* delle strategie di controllo della pompa combustibile e delle strategie di conrollo Waste Gate tramite Pierburg.
- Creazione di un blocco Caricamento e Salvataggio configurazione semplice, veloce e efficace per l'applicazione.
- Trattazione dei problemi di sovraccarico computazionale del PXI durante il funzionamento della VECU.

7.2.2 Verifica VECU Hardware in the Loop

Un'altra parte di lavoro molto importante è stata quella di verificare che tutto quello che si aveva in uscita dal PXI potesse essere sensato dal punto di vista del controllo e della gestione del motore. Sono state effettuate varie prove Hardware in the Loop, cioè in pratica si è simulato il segnale di ruota fonica e fase del motore (attraverso file di dati) per permetter al programma di effettuare il suo compito di fasatura e verificare che le attuazioni in uscita fossero nell' arco angolare prefissato.

In particolare sono state effettuate le seguenti prove: si è mantenuto costante il numero di giri e con l'ausilio di un oscilloscopio si sono tenute monitorate le attuazioni che si aveva interesse verificare con il riferimento dei segnali di ruota fonica e di fase. Si sono fatte quindi delle prove su:

- durate delle attuazioni di accensione (tempo carica bobina);
- posizione inizio carica e fine bobina nel ciclo (riferimento al dente);
- durata attuazione di iniezione (facendo riferimento al punto di funzionamento da mappa);
- inizio attuazione di iniezione;
- sequenza di attuazioni nei vari cilindri;
- capacità di generare misfuel e Misfire.

Dopo che queste prove hanno dato esito positivo si è passato alla fase più problematica del Cranking, cioè la fase di avviamento. La prima parte delle prove consisteva nel verificare la presenza di attuazioni anche con numero di giri inferiore ai 200 rpm. Questa fase ha dato da subito grossi problemi. il problema nasceva al raggiungimento di un numero di giri sui 400 rpm quando cioè le varie attuazioni davano calcoli di inizio e fine all'interno del medesimo dente di riferimento, il che rendeva impossibile al programma in *FPGA* la prosecuzione del *Loop* delle varie attuazioni (indistintamente di accensione o iniezione). È stato così necessario imporre una modifica imponendo il cambio ciclo (come trattato nel capitolo 4 sulle attuazioni in FPGA).

Una volta acceso il motore (dopo le operazioni di connessione fra sotware e sistema fisico di cui tratteremo nei prossimi paragrafi) a livello di software si è cercato di rendere il motore a banco completamente indipendente dai comandi della centralina di serie implementando il controllo pierburg e pompa di combustibile.

7.2.3 verifica attuazioni sul motore

Altra parte molto delicata del lavoro svolto è stata verificare l'effettiva presenza delle attuazioni nel motore. Cioè fisicamente si è verificato che ogni bobina e ogni iniettore venissero fisicamente eccitati. In pratica con un generatore di funzione si è simulato per ogni PIN della scheda di comando attuazioni (vedi capitolo 6) un segnale a onda quadra con frequenza tipica di un regime motore plausibile e si è verificato che effettivamente la bobina generasse la scintilla e che l'iniettore si muovesse. Dalla verifica di queste attuazioni si sono riscontrati dei problemi nelle iniezioni del cilindro 2 – 4 – 7.

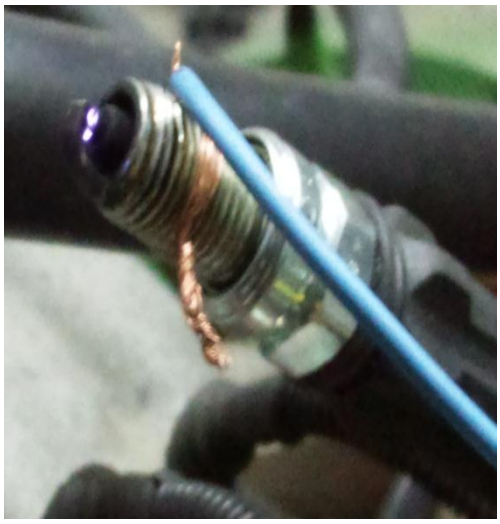


Figura 7.1: verifica carica bobina.

La seconda fase del lavoro è stata quella di cercare di capire quale componente elettronico fisicamente potesse essere guasto. Dopo varie prove si è proceduto a riconoscere per il cilindro 2 la rottura in un amplificatore operazionale, mentre per il 4 e il 7 nel Mosfet che manda il segnale all'iniettore. Sostituiti i componenti e verificate nuovamente tutte le attuazioni si è proceduto all'accensione del motore con la VECU.

7.3 Accensione del motore a banco

Una volta verificato la presenza delle attuazioni si è proceduto ad accendere il motore, in particolare si è effettuare lo scambio fra la gestione con la centralina di serie e la VECU. All'inizio si è messo in moto il motore con la centralina di serie e con la scheda delle attuazioni disabilitata per verificare se la VECU può fasarsi con il segnale di fonica e fase originali dal motore. Verificato così il funzionamento dei circuiti di squadratura si è proceduto a sostituire le varie attuazioni con quelle della VECU per verificare l'effettiva indipendenza del motore dalla centralina di serie. Si è testato che riusciamo effettivamente a sostituire la centralina di serie mentre il motore è in moto. Si è poi affrontata la fase delicata dell'accensione del motore tramite giro chiave del banco, quindi attraverso la fasatura a bassi giri imposta dal motorino di avviamento. Si è quindi verificato l'effettivo funzionamento del sistema studiato.

7.4 Sviluppi futuri

Gli sviluppi futuri della VECU riguardano in particolare lo sviluppo di un sistema in grado di effettuare una verifica dei dati memorizzati attraverso memoria FIFO e la possibilità di verificare e di avere la possibilità di implementare l'acquisizione di dati dai sensori attraverso gli 8 canali digitali e 4 analogici e la memoria attraverso l'FPGA della FIFO.
Un altro sviluppo futuro e l'applicazione delle strategie per portare il motore a una condizione di detonazione per far vedere a fini didattici quali elementi caratterizzano questo fenomeno.
Un' altra possibilità nasce da creare una versione della VECU in grado di poter esser implementata su CompactRIO. Oltre la possibilità ancora in fase di studio di creare un controllo motore

non solo influenzato dalle logiche classico del controllo di motori ad accensione comandata, ma anche controllo in coppia o in base al valore dei sensori ricavati

È poi auspicabile lo sviluppo di un sistema di controllo della detonazione basato sull'utilizzo di sensori accelerometrici, che risulta fondamentale per poter variare l'anticipo in tempo reale anche solo su alcuni cilindri interessati.

Per quanto riguarda il sistema non finalizzato al Rapid control Prototyping, verrà ulteriormente sviluppato il sistema hardware in the loop, con l'obiettivo di testare in maniera più completa il funzionamento delle centraline per il controllo motore, per poter risparmiare tempi e risorse.