Politecnico di Torino

dipartimento di ingegneria meccanica e aerospaziale

Corso di Laurea Magistrale in Ingegneria Meccanica

Tesi di Laurea Magistrale

Programmazione in ambiente LabVIEW del sistema di controllo e acquisizione dati di un banco prova per viti a ricircolo di sfere

Relatore:
Prof. Massimo Sorli

Correlatore:
Dott. Antonio Carlo Bertolino

Candidato:
Antonio Orgiu
Matricola S266199

Anno Accademico 2021-2022

Indice

Capitolo 0

Introduzione al banco prova

L'obiettivo del lavoro di tesi è stato la creazione, in ambiente LabVIEW, del codice atto all'acquisizione dati e al controllo di un banco prova per viti a ricircolo di sfere, locato presso il Laboratorio PEIC del Politecnico di Torino. L'attività è da considerarsi come proseguo di un tirocinio curriculare, della durata di 150 h*, svoltosi presso lo stesso laboratorio.*
In questo capitolo verrà fatta un'introduzione al banco prova, elencandone e descrivendone i componenti principali. Ci si concentrerà particolarmente sull'hardware CompactRIO, sul quale viene eseguito il programma realizzato. I capitoli successivi tratteranno esclusivamente del software LabVIEW e degli script realizzati, dei quali verranno descritte approfonditamente tutte le sezioni di codice.
Ringrazio Dr. Antonio Carlo Bertolino e Dr. Matteo Gaidano per l'essenziale supporto durante lo svolgimento di tutte le attività. Si ringraziano inoltre il Prof. Massimo Sorli, relatore ufficiale della tesi, e il Prof. Radu Bojoi, per aver permesso lo svolgimento dello stage.

0.1 Viti a ricircolo di sfere - Descrizione e utilizzo

Con "vite a ricircolo di sfere" ci si riferisce a un tipo particolare di vite, avente delle sfere di acciaio inserite negli spazi compresi tra vite e madrevite (anche detta "chiocciola"). Le sfere hanno il compito di trasformare l'attrito radente in attrito volvente. All'interno della chiocciola è ricavata una scanalatura elicoidale a sezione ad arco gotico, la quale migliora l'efficienza della vite, avente lo stesso passo della vite ma diametro medio superiore. Queste rientrano tra i metodi più diffusi per la trasformazione di moto rotatorio in traslatorio e viceversa.
Le viti a ricircolo di sfere ricoprono un vasto numero di campi d'applicazione: *Macchine a controllo numerico* (centri di lavoro, fresatrici, torni etc.), *Macchine di precisione* (trapani, piallatrici etc.), *Macchine industriali* (macchine per stampare, tessili etc.), *Macchine elettroniche* (robot, attuatori, manipolatori etc.) e altre diverse tipologie di macchine (avvolgitori, elevatori, letti per ospedali etc.).

Figura 1: *Vite a ricircolo di sfere con vista della madrevite in sezione.*

L'implementazione di questo tipo di viti comporta una serie di vantaggi:

- Un elevato rendimento meccanico, fino al 98%, che permette a questi dispositivi di essere pressoché reversibili (non autobloccanti).
- Un'elevata vita operativa grazie al basso attrito interno e ad una resistenza all'usura alquanto elevata.
- Bassa coppia di movimentazione e bassa potenza dissipata in calore.
- Significativa rigidità assiale.
- Elevate velocità e precisione di posizionamento.

0.1.1 Sistemi di ricircolo

Nella gamma di produzione delle viti a sfere, sono previsti diversi sistemi di ricircolo; i tipi maggiormente utilizzati sono due:

- **Ricircolo esterno** - Le sfere percorrono la pista di rotolamento nel suo intero sviluppo ed escono da un estremo, per poi venir fatte rientrare all'estremo opposto attraverso un percorso esterno alla pista stessa, caratterizzato da un tubetto deviatore in acciaio inox.
- **Ricircolo interno** - L'intera pista di rotolamento della chiocciola è suddivisa in tanti ricircoli, percorsi a ciclo chiuso dalle sfere. Al termine di ogni giro esse vengono riportate alla posizione iniziale da un tassello deviatore. Per questa ragione, le chiocciole di questo tipo hanno dimensioni più contenute. Per incrementarne la capacità di carico è necessario aumentare il numero di circuiti.

0.1.2 Precarico delle viti

Tramite il *precarico* è possibile eliminare il gioco assiale tra vite e chiocciola e aumentare la rigidezza stessa della vite, riducendo lo spostamento di deformazione dell'asse dovuto al carico assiale. Ne deriva un notevole incremento della precisione di posizionamento. Il precarico può essere realizzato in tre modi:

- **Precarico con sfere a diametro maggiorato** - Inserendo delle sfere con un diametro nominale maggiore rispetto a quello dello spazio di guida, si riesce ad ottenere un contatto su quattro punti.

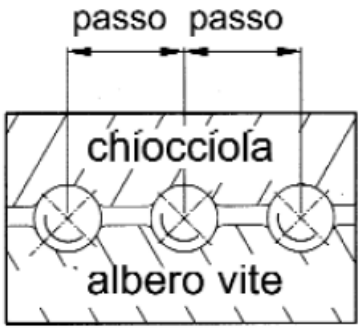

Figura 2: *Precarico con sfere maggiorate.*

- **Precarico tramite lavorazione della chiocciola** - La chiocciola viene lavorata in maniera tale da avere una compensazione di valore del passo centrale. Questa soluzione è ideale per chiocciole doppie ma non in applicazioni con carichi troppo pesanti; essi, infatti, causerebbero l'instaurarsi di sollecitazioni hertziane eccessive, tali da causare il cedimento a fatica del materiale.

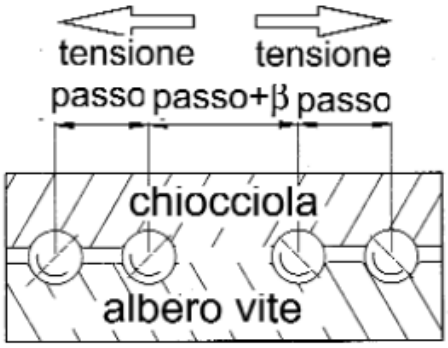

Figura 3: *Precarico con lavorazione della chiocciola.*

- **Precarico con distanziatori** - Nel caso di chiocciole doppie, tra le due viene inserito un opportuno spessore, facendole premere sulle sfere.

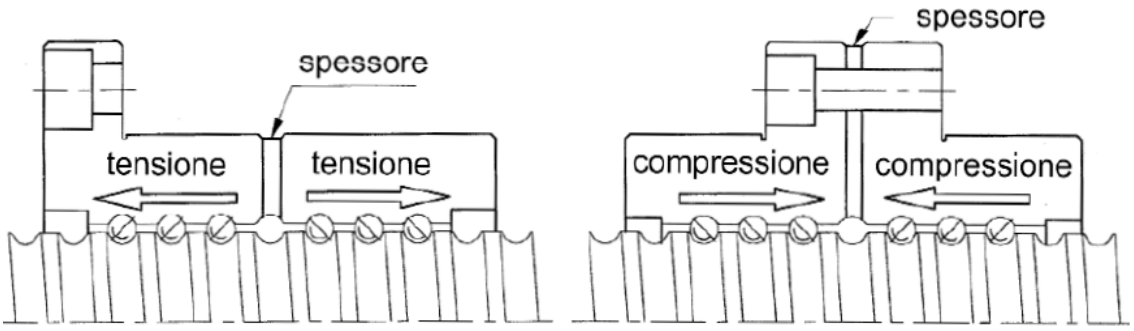

Figura 4: *Precarico con distanziatori.*

0.2 Banco prova - Finalità e layout generale

Lo scopo alla base della progettazione, e della successiva realizzazione, del banco prova in esame è stato quello di fare da supporto durante i test di viti a ricircolo di sfere. Queste sperimentazioni possono avere diverse finalità:

- Estrarre dei dati in condizioni nominali al fine di validare i risultati ottenuti dai modelli matematici.
- Testare diverse viti a ricircolo opportunamente degradate artificialmente.

In questo senso, il banco prova viene impiegato con lo scopo di identificare delle feature significative dei dati acquisiti, atti a identificare il difetto e, in ultima analisi, anche la vita residua del componente in prova.

Il banco prova è montato su un piano di appoggio autonomo, dotato di gambe e ruote di posizionamento. Esso è provvisto di un rack, per le alimentazioni elettriche e le sicurezze, e di uno ulteriore per il contenimento dell'unità di governo e acquisizione dati, dell'azionamento del motore elettrico e dei condizionatori dei trasduttori.
Il banco prova prevede un motoriduttore elettrico che movimenta la vite del componente in prova. Un sistema di cuscinetti reggispinta vincola la vite assialmente sul lato del motore. Un altro insieme di cuscinetti supporta l'altra estremità della vite solo radialmente, senza vincolarne lo spostamento assiale. Sulla vite sono montate due madreviti singole, che possono essere precaricate con configurazione ad *"O"* o ad *"X"* a seconda della prova. Le due madreviti e il sistema di precarico formano la slitta. Due guide lineari a basso attrito ne impediscono la rotazione, consentendone solamente la traslazione. Su una delle due madreviti (la *"Madrevite 1"* o *"Master Nut"*) è applicato il carico esterno tramite un cilindro pneumatico, la cui pressione è controllata da una valvola regolatrice di pressione di tipo proporzionale in flusso; questa permette il controllo della pressione all'interno di entrambe le camere. La valvola riceve un set di comando di pressione dall'hardware di acquisizione dati e controllo del banco. Nello specifico, si tratta del *CompactRIO-9047*, fornito dalla

National Instruments e oggetto del presente lavoro di tesi; esso comunica direttamente con il PC dell'operatore e invia anche il set di posizione o velocità al modulo di azionamento del motore elettrico.

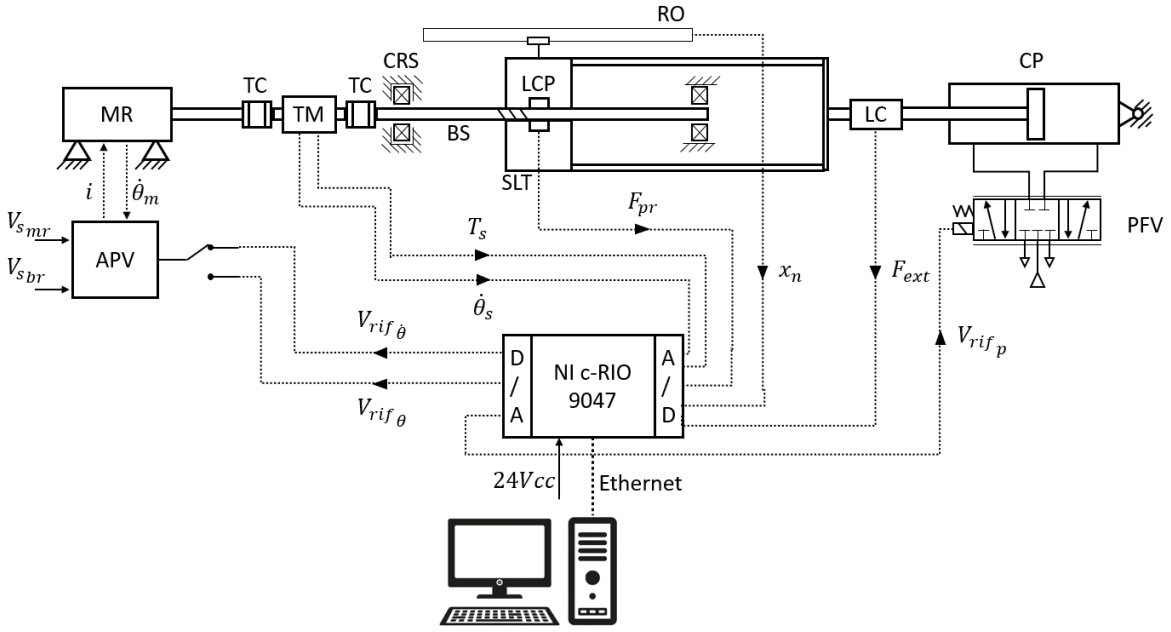

Figura 5: *Layout del banco prova.*

Tra il motoriduttore e la vite in prova è posizionato un torsiometro, interposto tra due giunti di coppia (*"Torque Coupling"*), per fornire i segnali di coppia e velocità angolare della vite. Una cella di carico è posizionata sulla slitta per leggere il livello di precarico. Una riga ottica chiude l'anello del controllo in *"Closed Loop"* e permette di misurare la posizione e la velocità lineare della slitta. Una cella di carico misura la forza assiale applicata alla Madrevite 1 dal cilindro pneumatico.

0.3 Banco prova - Componenti

Di seguito vengono riportati gli elementi principali costituenti il banco prova. Di essi verrà fatta una breve descrizione, nella quale verranno riportate le caratteristiche fondamentali.

0.3.1 Motoriduttore e modulo di azionamento

Il motoriduttore, rilasciato dalla *LENZE*, si compone fondamentalmente di due elementi: il motore elettrico e il riduttore.
Il motore elettrico (modello *LENZE MCS-09F38*) è di tipo sincrono a quattro coppie polari. Ha una velocità nominale di 3750 rpm e una coppia nominale di 3,1 Nm. Esso dispone di un freno di stazionamento a magneti permanenti incorporato; il suo scopo è il mantenimento di una posizione statica ma, in casi di emergenza, può essere utilizzato anche in condizioni

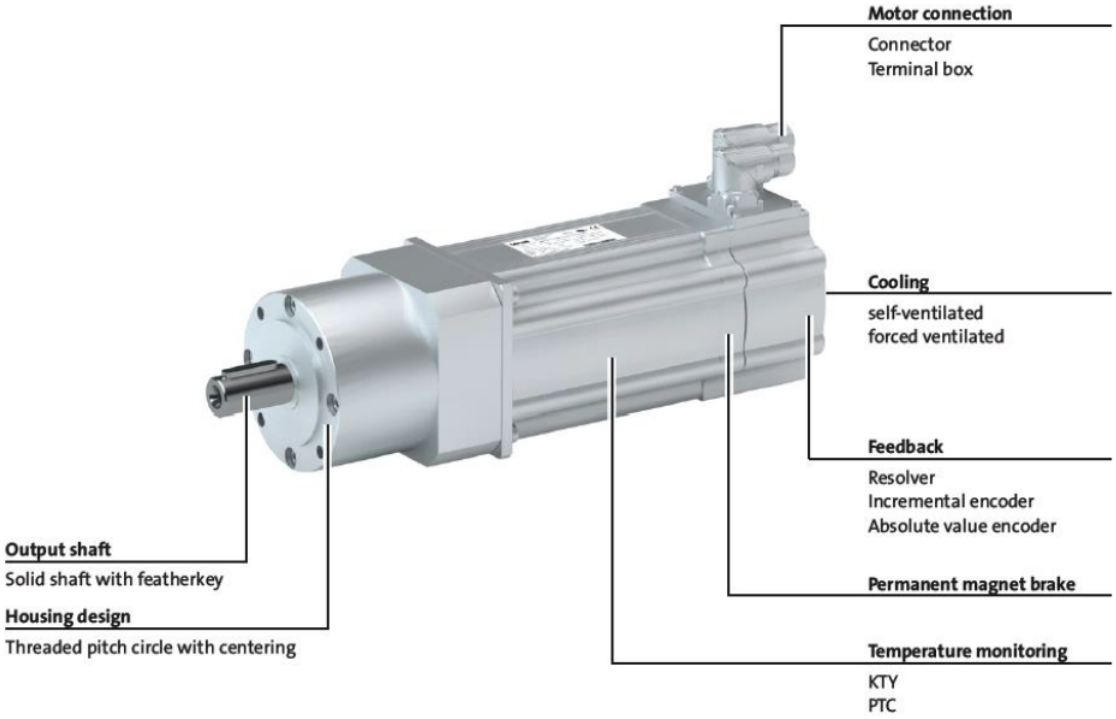

Figura 6: *Servomotore sincrono della LENZE.*

dinamiche, a discapito della coppia di frenatura che sarà sensibilmente inferiore a quella nominale. L'utilizzo non è quindi idoneo a casi di emergenza, almeno senza misure aggiuntive di sicurezza. Il freno è di tipo *normalmente chiuso*: agisce quando l'alimentazione viene disconnessa. Il comando del freno viene affidato a un modulo esterno al motore, implementato sul modulo di comando LENZE.
Il motore è dotato di un resolver, un sensore di posizione, incorporato e di tre sensori collegati in serie per il monitoraggio della temperatura. La temperatura del motore è quindi determinata con grande accuratezza nel campo di temperature ammissibili e, allo stesso tempo, il controllo della sovra-temperatura configurato nel controllore è effettuato in uno degli avvolgimenti.
Il riduttore (modello *LENZE g700-P44*) è di tipo *planetario* a singolo stadio. Esso garantisce un rapporto di riduzione pari a 4.
Per quanto riguarda l'azionamento, il motoriduttore è accoppiato con l'inverter *LENZE Servo Drives 9400 High Line modello E0044*. Questo riceve i comandi dal CompactRIO, comandando a sua volta il motoriduttore attraverso un segnale in corrente. Il driver viene programmato attraverso il programma *LENZE Engineer*, un software di programmazione a blocchi che permette la configurazione pressoché totale del dispositivo. Tra le varie funzioni, esso permette di gestire il tipo di controllo applicato sul servomotore (posizione, velocità, coppia etc.) e tutti gli input al driver provenienti dal controller CompactRIO. É inoltre possibile settare la *procedura di Homing* della slitta; ciò avviene con l'ausilio del segnale proveniente dal sensore di fine corsa e da quello di zero proveniente dalla riga ottica montati

sulla stessa. Questa applicazione permette quindi una completa implementazione del driver all'interno del banco prova.
Il driver è un dispositivo di tipo modulare, che è possibile dotare di una serie di accessori (e.g. il modulo di sbloccaggio del freno del motore elettrico).

0.3.2 Slitta e sistema di precarico

La slitta è costituita dalle due madreviti e dal sistema preposto al loro precarico, realizzabile nelle configurazioni ad *"X"* (le due madreviti sono spinte l'una contro l'altra) o ad *"O"* (le due madreviti sono allontanate). Per poter garantire entrambe le configurazioni, vengono utilizzate tre traverse: la prima e la terza sono solidali alla Madrevite 1, mentre la seconda è solidale alla Madrevite 2. É possibile anche lasciare le madreviti prive di precarico. Tutte le configurazioni si realizzano agendo sulla ghiera di precarico e su quella di anti-svitamento, di sinistra o di destra a seconda che si desideri la soluzione ad "X" o ad "O".

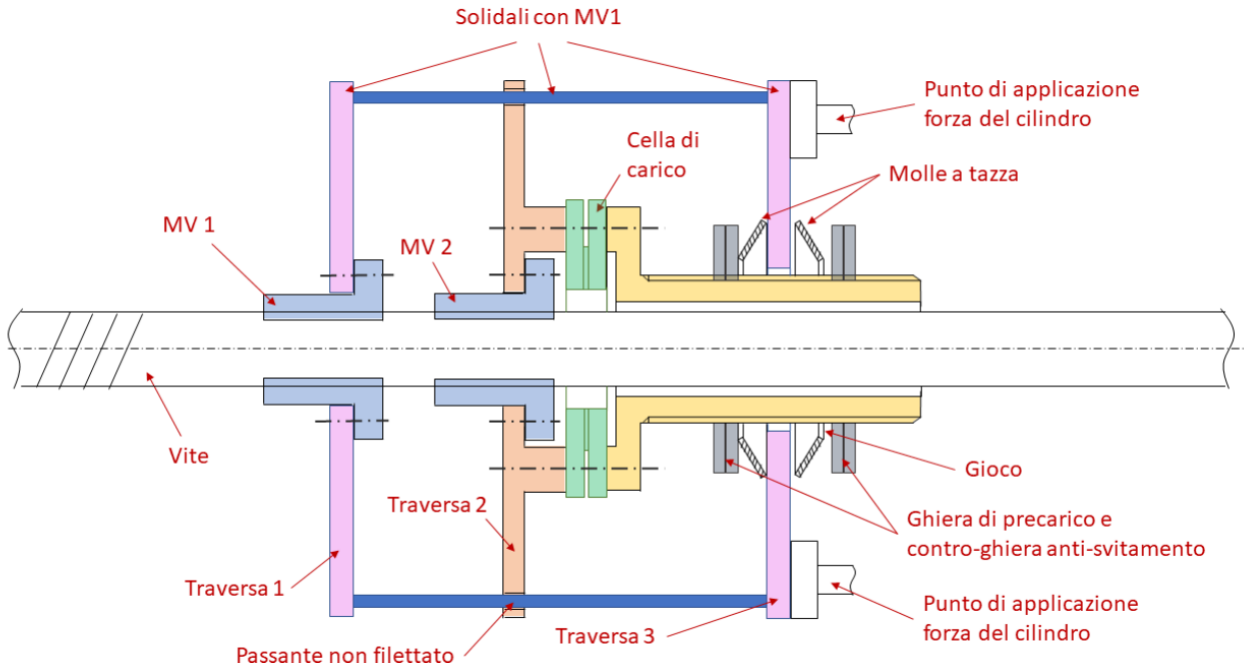

Figura 7: *Schema indicativo del sistema di precarico costituente la slitta.*

Lungo il percorso di precarico, solidale con la Madrevite 2, è installata una cella di carico che ne permette la misura. Al fine di applicare un precarico assiale sulle madreviti, evitando la generazione di momenti indesiderati, la cella di carico è di tipo toroidale. La presenza delle molle a tazza è utile a garantire una maggiore sensibilità sull'applicazione del precarico attraverso le ghiere. Sulla prima e terza traversa, e quindi sulla Madrevite 1, viene applicato il carico esterno dal cilindro pneumatico.

0.3.3 Torsiometro

Il torsiometro (modello *BURSTER 8661*) è di tipo dual range e permette, oltre la misura della coppia, quella della posizione e della velocità angolari.

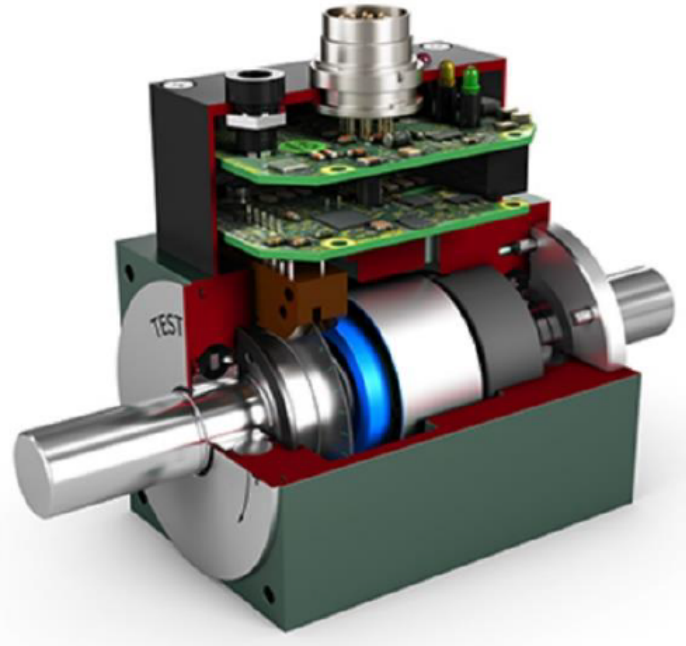

Figura 8: *Sezione del torsiometro BURSTER 8661.*

Questo sensore di coppia consiste essenzialmente in tre blocchi: il rotore, l'alloggiamento (contente lo statore) e l'elettronica di output. Il rotore è composto da diverse parti e contiene il dispositivo di misurazione reale, un elemento elastico; questo è progettato per deformarsi elasticamente sotto una coppia applicata. Ciò si traduce in torsione, che a sua volta produce una deformazione molto piccola nel materiale dell'elemento di misura. Entro certi limiti, questa deformazione è lineare e proporzionale alla coppia applicata; essa può essere misurata usando degli estensimetri collegati in un circuito a *ponte di Wheatstone*. Un microprocessore condiziona il segnale proveniente dal ponte e lo trasferisce allo statore. Il rotore è collegato allo statore tramite due cuscinetti a sfera e il trasferimento del segnale è senza contatto. Lo statore contiene l'elettronica necessaria per alimentare il rotore con la tensione operativa richiesta, attraverso mezzi induttivi e senza contatto. Nella direzione opposta, riceve il segnale di coppia trasmesso otticamente e digitalizzato, per poi indirizzarlo verso l'elettronica di uscita, in cui viene convertito in un segnale analogico $\pm 10\,\mathrm{V}$ e inviato in output tramite connettore.
Il sensore di coppia è dotato di un disco encoder incrementale per misurare la posizione e velocità angolari. Questo disco ha 2000 incrementi, permettendo risoluzioni angolari fino a $0{,}045\,^{\circ}$. Possono essere misurate velocità fino a 25,000 rpm, in base all'intervallo di misurazione selezionato e al disco encoder incrementale.
Tre LED indicano lo stato operativo del sensore, per una diagnostica più semplice e rapida.

0.3.4 Altri componenti

Al fine di evitare una semplice trascrizione delle schede tecniche dei prodotti, si riporta un breve elenco sommario degli altri componenti principali del banco prova. Alcuni di essi sono stati già citati in precedenza. All'hardware di controllo e acquisizione, il CompactRIO e i relativi moduli, verrà dedicato successivamente un intero paragrafo.

- **Vite a ricircolo di sfere**
- **Giunti di coppia**
- **Cella di carico per forza esterna**
- **Cella di carico per precarico**
- **Amplificatore**
- **Molle a tazza** - Devono essere tali da garantire una maggior precisione nell'impostazione del precarico iniziale e, contemporaneamente, non influire negativamente sulla rigidezza complessiva del sistema di precarico.
- **Cilindro pneumatico di carico** - Il cilindro pneumatico utilizzato per l'applicazione della forza esterna (modello *FESTO DSBG*) è a doppio effetto e ha un diametro di 160 mm. La sua corsa, di 350 mm, è maggiore di quella massima della slitta, dettata dai fine corsa meccanici, per evitare che esso si danneggi arrivando accidentalmente al suo fine corsa.
- **Riga ottica** - Della *Elcis Encoder S.r.l.*, modello *RV1846-L*, ha una corsa utile di 320 mm. Di default sono presenti due tacche di zero, all'inizio e alla fine della corsa utile. Tuttavia, essendo la corsa utile maggiore di quella effettiva della slitta, queste non vengono raggiunte. In sede di acquisto, è stata fatta aggiungere un'ulteriore tacca di zero in mezzeria, in modo da poter avviare la procedura di calibrazione all'inizio delle prove.
- **Cuscinetti reggispinta** - Devono essere scelti in accordo alle spinte sviluppate dal sistema di carico e dagli effetti inerziali del sistema.
- **Guide assiali anti-rotazione** - Le guide devono essere a basso attrito, realizzate con manicotti a sfera, atte a impedire la rotazione della slitta in entrambe le direzioni.
- **Fine corsa** - Per sicurezza, il banco deve prevedere dei finecorsa "software", seguiti da finecorsa con interruttore ad azionamento meccanico che disattivino l'alimentazione elettrica e pneumatica quando raggiunti; questi sono seguiti infine da finecorsa meccanici (*puffer*), in grado di assorbire l'energia cinetica di tutta la massa in movimento alla massima velocità e dissiparla senza causare danni ad altre parti del banco.

- **Pulsante a fungo per l'arresto di emergenza** - In caso di emergenza, l'azionamento del pulsante a fungo permette il cut istantaneo dell'alimentazione del motore elettrico, lasciando però alimentate le restanti parti del banco, in particolare tutto l'hardware di controllo e acquisizione.
- **Termocoppie**
- **Sensori di pressione**
- **Valvole pneumatiche di sicurezza**
- **Serbatoio pneumatico**
- **Servovalvola pneumatica in flusso**

0.4 Hardware di controllo e acquisizione

Il *National Instruments CompactRIO-9047* svolge il duplice compito di controllo e acquisizione dati. I componenti che necessitano di un controllo continuo sono il motoriduttore LENZE e l'attuatore pneumatico di carico. La generazione dei segnali di set verrà eseguita dallo stesso CompactRIO; tuttavia, le funzioni di SET, di posizione e velocità, vengono fornite al controller mediante PC. I segnali da acquisire dall'ambiente di prova sono quelli delle celle di carico, del torsiometro, degli encoder, dei sensori di pressione e delle termocoppie. Tutti i sistemi di alimentazione, continua e alternata, e lo stesso CompactRIO sono alloggiati nel vano dedicato alle apposite rack, citate precedentemente.

0.4.1 CompactRIO-9047

Il CompactRIO-9047, riportato in **Figura 9**, racchiude dei sistemi integrati che combinano un controller *RT* (*"Real-Time"*) e un processore *FPGA* (*"Field Programmable Gate Array"*) all'interno di uno chassis, per applicazioni di controllo industriali e di monitoraggio. Esso è costituito da uno chassis equipaggiabile con un massimo di 8 moduli I/O e racchiude al suo interno un processore dual-core *Intel Atom* da 1,6 GHz e un chip FPGA *Xilinx Kintex-7*.
Una memoria non volatile da 4 GB consente inoltre al CompactRIO di lavorare come un *sistema embedded*: il codice scritto in fase di programmazione viene caricato direttamente sulla memoria, consentendone il funzionamento autonomo e indipendente da una postazione PC esterna. Lo chassis è dotato di quattro porte ethernet, che consentono la connessione ad altri dispositivi. Tra questi si ha il *PC host*, avente il ruolo di monitoraggio, acquisizione dati e comando.
Uno dei punti di forza del sistema CompactRIO è sicuramente l'elevata modularità e configurabilità, assicurate dal circuito FPGA (indirettamente definibile dall'utente) e dalla vasta disponibilità di moduli I/O, forniti dalla NI stessa, equipaggiabili sullo chassis. I moduli

Figura 9: *Il CompactRIO-9047 montato sul banco prova.*

vengono scelti in base alle necessità della specifica applicazione e possono svolgere operazioni tra le più disparate: acquisizione e generazione di segnali analogici in tensione e corrente, acquisizione di segnali provenienti da termocoppie, accelerometri e celle di carico, moduli I/O digitali, contatori, temporizzatori e così via.

Durante la programmazione di un CompactRIO e, più in generale, di tutti i prodotti della serie NI RIO (= *"Reconfigurable Input/Output"*), bisogna considerare la presenza di tre elementi ben distinti. Questi elementi hardware trovano una corrispettiva sezione di codice all'interno del progetto LabVIEW; si tratta di:

- Interfaccia Uomo-Macchina (*HMI* = *"Human-Machine Interface"*)
- Processore Real-Time (*RT*)
- Field Programmable Gate Array (*FPGA*)

L'HMI, chiamata anche *"Host"*, fornisce all'utente un'interfaccia grafica (GUI) per il monitoraggio dello stato del sistema e il settaggio dei parametri operativi. Questa interfaccia può essere implementata su dispositivi di diverso tipo, come ad esempio computer Windows, tablet o Touch Panel Computer. L'uso di una HMI è opzionale, dal momento che i prodotti RIO, come già affermato, sono in grado di lavorare autonomamente come sistemi embedded. Il processore RT si occupa dell'esecuzione del principale programma omonimo, permettendo l'esecuzione di programmi che richiedono un timing ben preciso e una certa affidabilità. L'FPGA è un chip in silicio riprogrammabile ed è il pilastro di tutto il sistema embedded; dal punto di vista computazionale, è l'elemento più performante di tutto il controller.
L'FPGA connette direttamente gli I/O dello chassis al processore RT senza l'ausilio di un bus, il quale non permetterebbe un sistema di controllo della latenza accettabile, special-

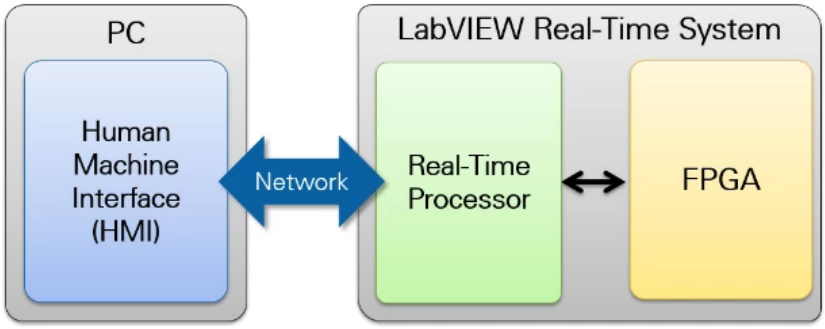

Figura 10: *Architettura di un sistema enbedded.*

mente se comparato con altre architetture. Grazie alla sua velocità e affidabilità, l'FPGA è utilizzata per tutte quelle operazioni che richiedono un costo computazionale basso, ma che devono essere eseguite istantaneamente ed in maniera assolutamente deterministica: I/O ad altissima velocità, loop di controllo a elevata frequenza, filtraggio di segnali e così via. Le corrispettive sezioni di codice di HMI, RT e FPGA sono anche locate in zone diverse del progetto LabVIEW dove sono programmate. La parte software del controller costituisce l'oggetto dell'elaborato e, pertanto, verrà ampiamente discussa successivamente.

Field Programmable Gate Array (FPGA)

Come accennato, la tecnologia FPGA concorre nell'implementare la riconfigurabilità del CompactRIO. Un FPGA è a tutti gli effetti un circuito elettronico programmabile la cui struttura, schematizzata in **Figura 11**, presenta una suddivisione reticolare che individua un numero finito di blocchi, detti *"celle logiche"*. Tra un blocco e l'altro esistono delle interconnessioni riconfigurabili in base alle necessità: dipendono dai dati che si scambiano i blocchi tra di loro. Esistono inoltre dei *blocchi I/O* che fungono, appunto, da input (acquisizione dei dati) e output (generazione di segnale) per il circuito.

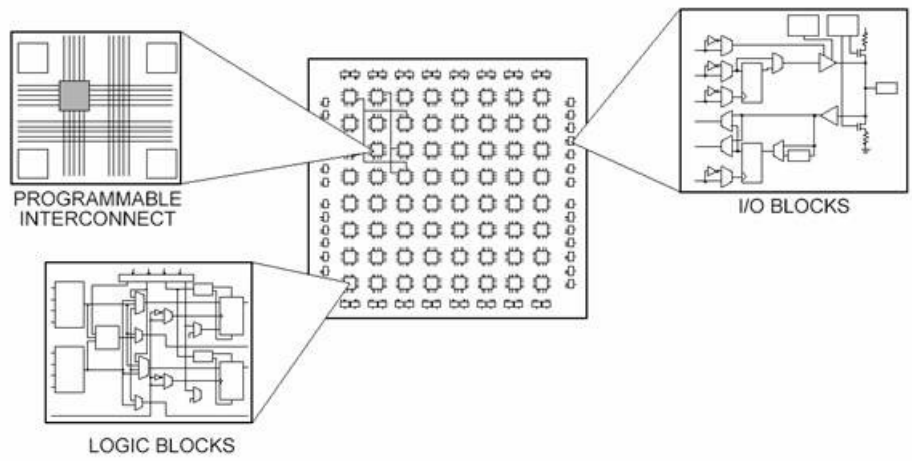

Figura 11: *Struttura di un FPGA.*

Ciascun blocco svolge una funzione specifica (e.g. somma, sottrazione etc.), assegnata in base al codice scritto dal programmatore: l'assegnazione delle funzioni ai vari blocchi viene gestita autonomamente dal software LabVIEW in fase di compilazione. Il programmatore può esclusivamente sviluppare il codice, attraverso le tipiche modalità previste dall'ambiente LabVIEW; il software prevede l'aggiunta di funzioni specifiche per l'FPGA, ottenibili con l'installazione dell'add-on *LabVIEW FPGA Module*. Ne risulta che, per come è definita, l'FPGA non può ospitare un codice che richieda un numero di celle logiche superiore a quello delle celle disponibili. Durante la fase di programmazione, sarebbe dunque opportuno prediligere architetture che richiedano un minor utilizzo di celle logiche, rispetto ad altre con un costo computazionale maggiore.
Il passaggio da un blocco logico a un altro è scandito dalla *frequenza di clock*: maggiore sarà la frequenza e minore sarà il tempo necessario all'esecuzione del programma, essendo quest'ultimo dato dall'insieme dei singoli blocchi logici interconnessi. Un notevole vantaggio dell'FPGA è la possibilità di lavorare a frequenze molto elevate, dell'ordine dei MHz. Nel CompactRIO-9047 si raggiungono i 40 MHz.

Real-Time (RT)

Un sistema computazionale "in real-time" è in grado di eseguire il programma in accordo a precisi requisiti di *timing*, caratteristica fondamentale per molte applicazioni ingegneristiche. Il componente chiave per la realizzazione di un sistema RT è il *Real-Time Operating System* (*RTOS*).
Per molte applicazioni ingegneristiche e di ricerca, l'esecuzione di un programma di acquisizione e/o controllo su un PC con un sistema operativo cosiddetto "general-purpose" (e.g. Windows) è inaccettabile. In qualsiasi istante, l'OS può infatti ritardare l'esecuzione di un programma per una svariata serie di ragioni: scansione antivirus, aggiornamenti del sistema o dei driver, esecuzione di altri programmi in background e così via. Per programmi che necessitano di un'esecuzione con un timing ben preciso, questo ritardo può causare la *failure* di tutto il sistema. Tuttavia, è opportuno precisare che questo comportamento è dovuto a come il sistema operativo stesso è stato concepito. Infatti, gli OS generici sono concepiti per eseguire molti processi e applicazioni alla volta e per fornire all'utente particolari feature, come ad esempio interfacce grafiche ricche e "pesanti" per l'hardware. Al contrario, un sistema operativo real-time è progettato per eseguire un solo programma con un timing molto preciso. Nello specifico, gli RTOS permettono di:

- Garantire che il ritardo "di worst-case" nello svolgimento delle attività rimanga contenuto in un certo intervallo di tempo.
- Dare priorità a determinate sezioni del codice.
- Eseguire loop con sostanzialmente lo stesso timing (tipicamente dell'ordine dei microsecondi).

- Rilevare se ci siano stati dei ritardi nell'esecuzione del loop.

Oltre al fornire un preciso timing di esecuzione, al sistema in real-time potrebbe essere richiesta una certa affidabilità, prolungata per giorni, settimane o addirittura mesi (nel caso di sistemi che lavorano ininterrottamente per archi di tempo molto estesi). Questo è importante non solo per sistemi ingegneristici che svolgono operazioni 24/7, ma anche per applicazioni nelle quali i tempi morti sono più costosi. In questo tipo di sistemi, è tipicamente inclusa anche una funzione di *"watchdog"* che permette, nel caso in cui il programma smetta di funzionare, di riavviare l'intero computer. Infine, l'hardware utilizzato nei sistemi real-time è spesso molto robusto e in grado di sostenere condizioni di funzionamento difficili per lunghi periodi di tempo.

Figura 12: *Elementi principali che costituiscono un sistema real-time.*

Sebbene l'elemento principale per la realizzazione di un sistema RT sia l'RTOS, sono necessari ulteriori componenti, sia di tipo software che hardware:

- **Tool per lo sviluppo del codice** - Un compilatore, un linker e un debugger che permettano di creare un codice compatibile con il sistema operativo real-time.
- **Drivers** - Per un sistema operativo real-time per comunicare con l'hardware e i moduli I/O.
- **Moduli I/O**

Tutti gli elementi costituenti un sistema RT e citati fino ad adesso sono riportati in **Figura 12**.

La National Instruments fornisce hardware e software in grado di lavorare sinergicamente, al fine di eseguire applicazioni in maniera affidabile, deterministica e con un timing ben preciso. La componente software è sicuramente rappresentata da LabVIEW e, nello specifico, dall'add-on *LabVIEW Real-Time Module*. Questo, in maniera analoga al modulo specifico per FPGA, fornisce un pacchetto di strumenti software completo per la realizzazione di un sistema real-time.
Come viene schematizzato in **Figura 13**, l'hardware Compact-RIO 9047 presenta un'architettura marcatamente eterogenea, combinando un processore real-time, un FPGA e dei moduli I/O dedicati. La frequenza di clock massima del processore RT è di 1 MHz.

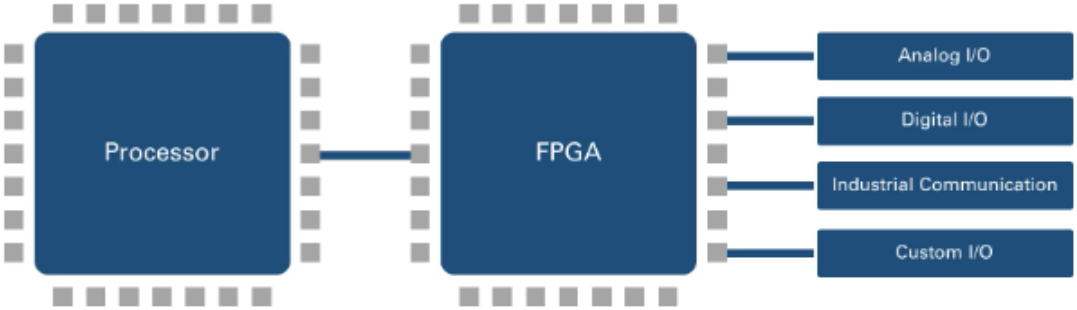

Figura 13: *Architettura hardware di un CompactRIO.*

0.4.2 Moduli I/O

Il Compact-RIO 9047 è un controller modulare con otto slot disponibili. Tutti i moduli utilizzati sul banco prova sono forniti dalla National Instruments. Facendo riferimento alla **Figura 9** e considerando i moduli da sinistra verso destra, se ne riporta una breve descrizione. L'ultimo modulo sulla destra, essendo lo slot inutilizzato, fa da copertura per i contatti che, altrimenti, rimarrebbero scoperti.

Modulo NI 9201 (2x) É un modulo di ingresso analogico per sistemi CompactDAQ e CompactRIO. Il NI 9201 fornisce otto canali di ingresso $\pm 10\,\mathrm{V}$ con frequenza di campionamento fino a $500\,\mathrm{kS/s}$. In questo modulo convergono i segnali analogici provenienti dal driver del motoriduttore, dal torsiometro, dai driver delle due celle di carico e dai sensori di pressione.

Modulo NI 9411 É un modulo *C Series* con sei ingressi digitali, aventi 500 ns massimi di *update rate*. Questo modulo funziona con livelli e segnali logici industriali, per il collegamento diretto a un'ampia gamma di interruttori, trasduttori e dispositivi industriali.

Modulo NI 9375 Si tratta di un modulo I/O digitale da $24\,\mathrm{V}$, con sedici canali di input e sedici di output. Questo modulo si presta particolarmente alla gestione dei segnali delle interfacce di sicurezza (interblocchi etc.), con un tempo minimo di aggiornamento di $7\,\mu\mathrm{s}$ in ingresso e $500\,\mu\mathrm{s}$ in uscita. Nel banco prova in esame, Il NI 9375 viene usato per la gestione dei segnali digitali da/verso i driver del motoriduttore e della cella di carico per la forza esterna.

Modulo NI 9401 É un modulo I/O digitale di tipo TTL (*"Transistor-transistor-logic"*), con otto canali, che lavora su range di tensione nell'intervallo da $0\,\mathrm{V}$ a $5\,\mathrm{V}$. Viene utilizzato per l'acquisizione dei segnali provenienti dall'encoder angolare integrato nel torsiometro; la massima frequenza di aggiornamento per ciascun canale è di $9\,\mathrm{MHz}$.

Modulo NI 9211 Il modulo NI 9211 è un dispositivo di ingresso specifico per le termocoppie, concepito per l'utilizzo con CompactDAQ e NI CompactRIO. Lo chassis include: un convertitore A/D ("analogico/digitale") sigma-delta a 24 bit, filtri anti-aliasing, rilevamento di termocoppie aperte e compensazione della giunzione fredda, per termocoppie ad alta precisione di misurazione.

Modulo NI 9269 É un modulo di output analogico a quattro canali, ciascuno dei quali è isolato dagli altri. Può raggiungere una frequenza di campionamento massima di 100 kS/s per canale, su qualsiasi chassis NI CompactRIO e NI CompactDAQ. Simile al modulo NI 9263, il NI 9269 aggiunge l'isolamento *channel-to-channel*, garantendo una maggiore sicurezza e una migliore qualità del segnale. É possibile aumentare la tensione di uscita nominale, "impilando" fino a quattro canali e ottenendo una tensione massima di ± 40 V. L'isolamento channel-to-channel è solitamente richiesto per per le applicazioni che hanno più sistemi elettrici, come test automobilistici o applicazioni industriali, soggetti a maggiore rumore e spesso contenenti più piani di messa a terra.

Capitolo 1

Software NI LabVIEW

Il codice LabVIEW, atto all'acquisizione dati e al controllo del banco prova, costituisce l'argomento centrale del lavoro di tesi. In questo capitolo, verrà fatta un'introduzione al software: verrà descritto l'ambiente di programmazione, le feature principali del software, per poi concentrarsi esclusivamente sui metodi di comunicazione tra le diverse parti del programma. La loro comprensione è, infatti, fondamentale per poter effettuare le scelte opportune durante la realizzazione del codice. Al contrario, non verranno trattati i principi della programmazione in ambiente LabVIEW, per i quali si rimanda all'opportuno materiale.
Si intende precisare che, essendo LabVIEW un programma estremamente vasto, il quale si presta a una svariata serie di scopi e dispositivi, ne verranno riportate solo le funzionalità effettivamente utilizzabili durante la realizzazione del programma.

1.1 Introduzione

LabVIEW (*"Laboratory Virtual Instruments Engineering Workbench"*) è un ambiente di sviluppo di applicazioni, principalmente orientate all'acquisizione dei dati, alla gestione di strumentazione elettronica e all'analisi ed elaborazione dei segnali. Esso fornisce un ambiente di programmazione di tipo *grafico ad oggetti*, denominato "*G-Language*", il quale consente di realizzare programmi in forma di diagrammi a blocchi.

Figura 1.1: *Logo del software LabVIEW.*

LabVIEW conserva comunque molte similitudini con gli ambienti di programmazione tradizionali:

- contiene tutti i tipi di dati, permettendo di crearne di nuovi attraverso la loro combinazione, e gli operatori di uso comune;
- permette di controllare l'esecuzione dei programmi, ricorrendo a strutture di controllo di flusso come, ad esempio, cicli e costrutti per l'esecuzione condizionale del codice.

LabVIEW mette a disposizione del programmatore una serie di librerie, richiamabili ed utilizzabili all'interno dei programmi; queste contengono funzioni di uso comune (aritmetiche e statistiche, per la manipolazione di stringhe etc.) e funzioni specializzate nell'acquisizione ed elaborazione dei segnali. Per specifiche applicazioni, può essere necessaria l'implementazione di nuovi driver e librerie. Assieme a LabVIEW, parte di essi viene fornita di default; tuttavia, è possibile ampliarne la disponibilità attraverso il tool *JKI - VI Package Manager*, che permette il download e l'installazione di librerie realizzate sia dalla NI che da enti di terze parti, e dal download di add-on direttamente dal sito della NI. Tra questi ultimi rientrano *LabVIEW FPGA Module*, *LabVIEW Real-Time Module* e *NI CompactRIO and Drivers*, installati appositamente per la programmazione del CompactRIO 9047 montato sul banco prova. Nuove funzioni e librerie possono essere realizzate dagli stessi utenti. La gestione dei pacchetti installati avviene tramite il tool *NI Package Manager* e la gestione degli hardware tramite *NI MAX*.
Infine, il programma consente il *debug* delle applicazioni realizzate in G-Language e fornisce un sistema di *error-handling* molto efficiente.

1.2 Virtual-Instrument (VI)

Il *Virtual Instrument*, o *VI*, è l'elemento fondamentale dei programmi scritti in G-Language. I VI permettono l'interazione tra PC e strumentazione, fornendo allo stesso tempo all'utente una GUI per poter interagire con gli stessi VI. Tentando un'analogia con altri linguaggi di programmazione, un VI è paragonabile a una funzione o a una *subroutine*. Il funzionamento di LabVIEW si basa proprio sull'interazione tra i diversi VI contenuti nel progetto. Il nome "Virtual Instrument" deriva dal fatto che le applicazioni LabVIEW sono progettate per simulare degli strumenti reali: al posto di essere strumenti fisici montati su un banco, essi sono virtuali che esistono solamente all'interno del software. I VI sono salvati come file "*.vi", formato proprietario della National Instruments.
L'organizzazione dei programmi tramite VI permette di dare al codice una struttura modulare, ovvero la suddivisione di programmi complessi in sottoprogrammi più e semplici e riutilizzabili. Essi prendono il nome di *"subVI"* e sono VI a tutti gli effetti (mantengono pure lo stesso formato); una subVI viene semplicemente inserita all'interno di una VI di livello gerarchicamente superiore. L'insieme delle VI, ma anche i componenti hardware reali, gli elementi di memoria, i controlli etc., vengono gestiti tramite il *Project Explorer*, nonché

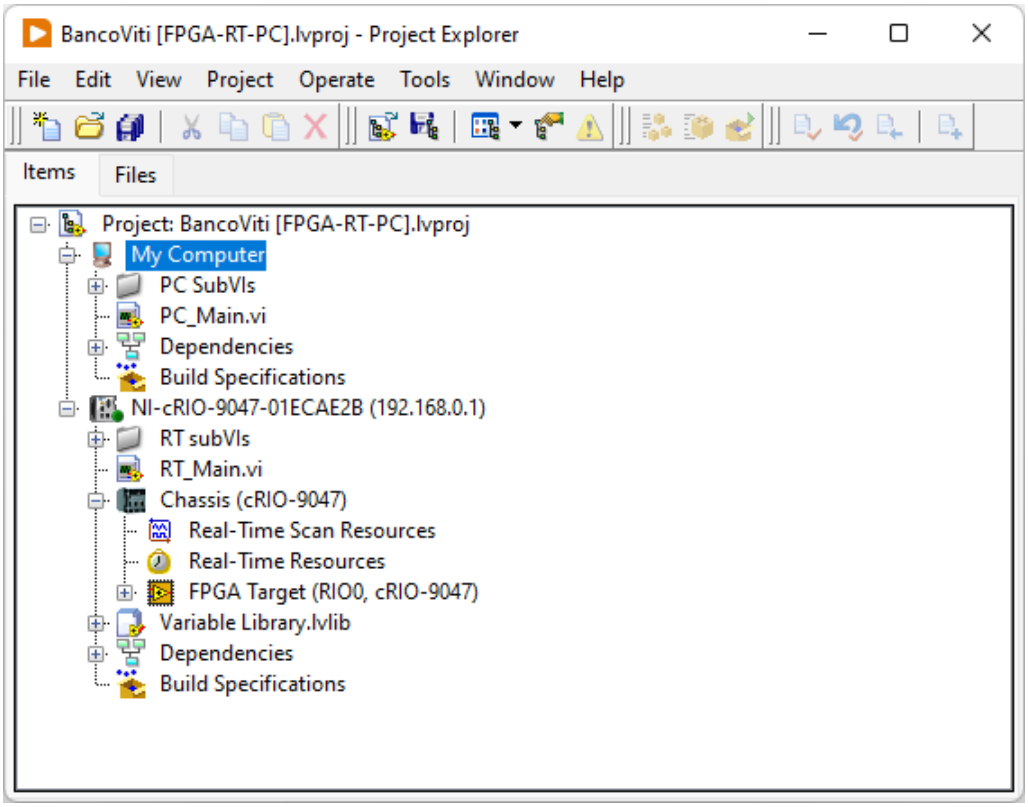

Figura 1.2: *Project Explorer del codice realizzato per il banco viti. Notare l'inserimento del CompactRIO 9047 come vero e proprio elemento del progetto e, a cascata, gerarchicamente sotto di esso, le VI correlate. Al fine di mantenere l'ordine all'interno dell'albero di progetto, le subVI vengono raccolte in cartelle, in ordine alfabetico.*

l'albero di progetto (**Figura 1.2**).

Un VI è costituito principalmente da quattro elementi:

- Pannello frontale (*Front Panel*)
- Diagramma funzionale a blocchi (*Block Diagram*)
- Insieme di Icona e *Connector Pane*

Front Panel Il *Front Panel* è la finestra che rappresenta l'interfaccia tra il programma e l'utilizzatore (GUI). Nel pannello frontale sono posizionati tutti i controlli e gli indicatori dello strumento virtuale. Con "controllo", ci si riferisce a una variabile in ingresso, che può essere modificata agendo sugli opportuni elementi del pannello frontale. L'utente può agire sui controlli in maniera diversa: tramite i più semplici menù a tendina e caselle in cui inserire il valore della variabile, ai pomelli e gli slider. Con "indicatore", invece, si intende una variabile in uscita, il cui valore può essere modificato dal programma e non direttamente dall'utente. Analogamente ai controlli, gli indicatori possono rappresentare le

variabili in diversi modi: indicatori a lancetta, semplici indicatori numerici/a stringa e così via. Un esempio di controlli e indicatori disponibili è riportato in **Figura 1.3**.

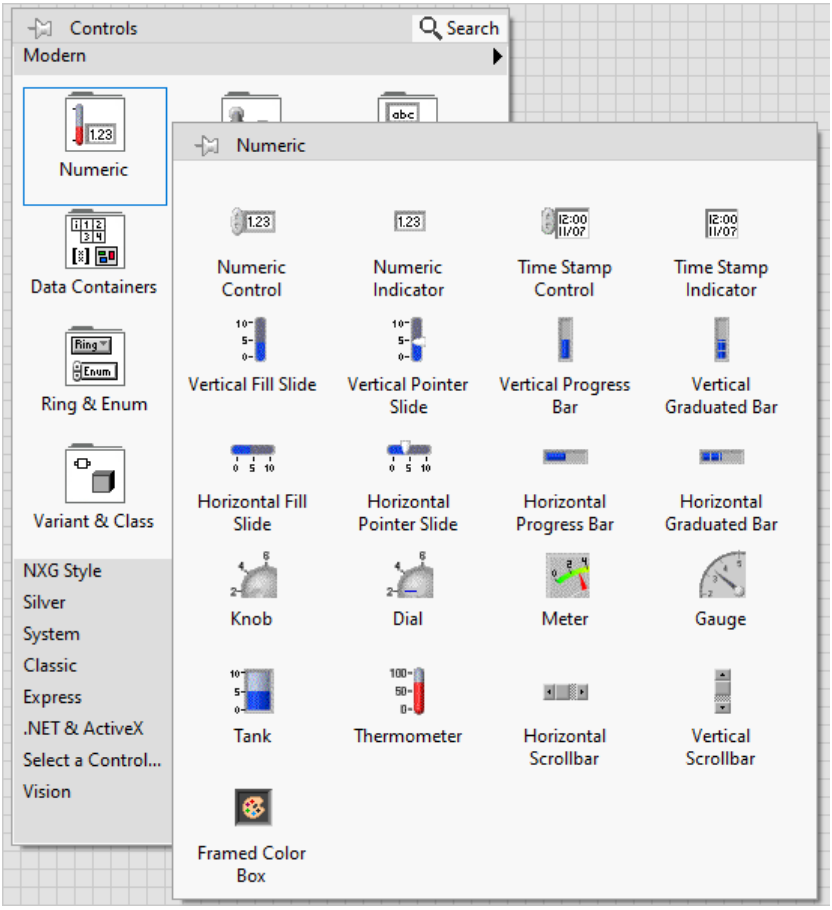

Figura 1.3: *Immagine esemplificativa che riporta la palette con i controlli e gli indicatori di tipo numerico, utilizzabili nel Front Panel.*

Block Diagram Il *Block Diagram* contiene il codice vero e proprio del programma. Gli elementi più importanti che lo compongono sono i blocchi delle funzioni, i quali svolgono tutte le operazioni richieste al programma. I blocchi possono essere analoghi a delle *black box* (si ha accesso agli input e agli output ma non alle operazioni interne ad essi) oppure essere usate come delle vere e proprie subVI. Queste possono essere presenti di default nel programma oppure essere realizzate dal programmatore; in entrambi i casi, è possibile modificarle. Oltre ai blocchi, sono presenti nel diagramma anche i terminali di input e output, ciascuno dei quali ha il suo corrispettivo controllo o indicatore nel Front Panel. Blocchi e terminali sono connessi tra di loro tramite i fili.
Il colore di fili e terminali varia in base alla tipologia di dato che essi trasportano.

Connector Pane e Icona L'insieme formato dall'icona e dal *Connector Pane* (*"pannello dei collegamenti"*) si trova in alto a destra del Front Panel ed è presente in tutte le VI.

L'icona è completamente personalizzabile dall'utente: tramite doppio click sull'icona stessa, è possibile accedere all'apposita finestra. Quando una VI viene utilizzata come subVI l'icona del suo blocco, nel Block Diagram della VI in cui viene inserita, è esattamente quella riportata nel Front Panel.

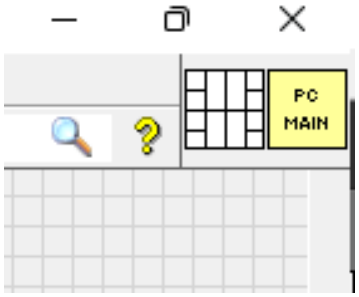

Figura 1.4: *Insieme di icona e Connector Pane della VI.*

Il Connector Pane riporta i collegamenti dei terminali della VI con l'esterno. Essendo i terminali delle variabili in ingresso o in uscita, i collegamenti non sono altro che gli ingressi o le uscite delle variabili della funzione realizzata all'interno del blocco. Il numero e la disposizione dei collegamenti possono essere scelti dall'apposito menù a tendina, all'interno del quale sono riportati tutti i pattern disponibili di default nel software. I collegamenti con i diversi terminali sono impostati manualmente dall'utente. I pattern disponibili sono riportati in **Figura 1.5**.

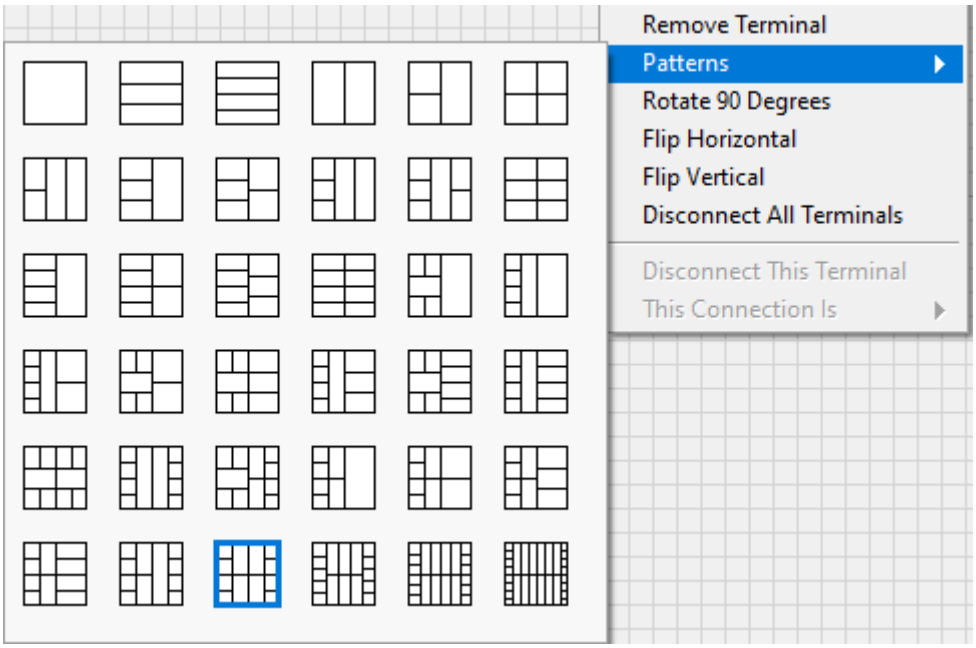

Figura 1.5: *Pattern disponibili per il collegamento della subVI con gli altri elementi del Block Diagram.*

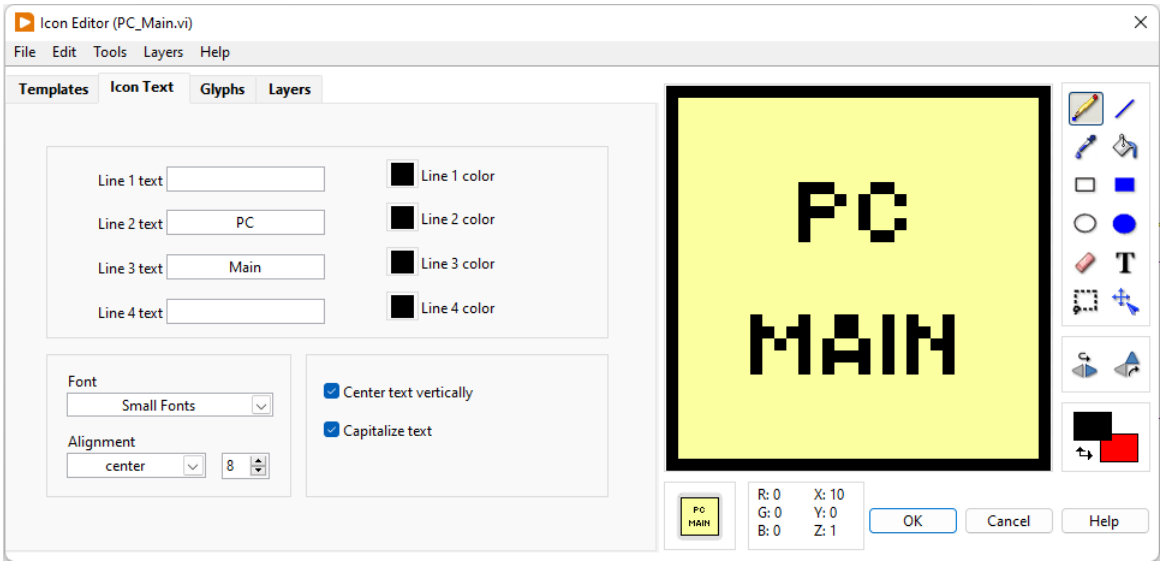

Figura 1.6: *Schermata per la personalizzazione dell'icona della VI, accessibile facendo doppio click sull'icona stessa.*

1.3 Metodi di comunicazione

I programmi realizzati su LabVIEW agiscono su più livelli, sia software (un singolo codice può comprendere più VI e subVI) che hardware (e.g. il CompactRIO, munito di processore ed FPGA e, nell'applicazione in esame, accoppiato al PC host). Vien da sé l'importanza di avere una visione chiara e totalizzante sul come i diversi sistemi comunichino tra loro.

1.3.1 Accesso agli I/O

Prima di addentrarsi nei differenti metodi di comunicazione, è importante capire come i dati sono trasportati dalle porte di input o output all'interno del dispositivo target RIO. Quando si usa un CompactRIO, l'accesso alle porte I/O può essere gestito in tre modi diversi. Una rappresentazione grafica del loro funzionamento viene riportata in **Figura 1.7**.

Modalità Real-Time (NI-DAQmx) - I dati vengono letti sul processore tramite le VI *NI-DAQmx* all'interno del VI Real-Time principale. Con questa modalità, l'utente accede agli I/O attraverso delle funzioni preimpostate e pronte all'uso.

Modalità Real-Time Scan (IO Variable) - I dati acquisiti vengono inviati dall'FPGA al processore Real-Time. A livello software, non è necessaria la programmazione del VI FPGA, ma solamente di quello RT. Qui, verranno inseriti gli I/O Nodes delle porte di interesse.

Modalità LabVIEW FPGA - A livello hardware, i dati seguono lo stesso percorso della modalità precedente. Tuttavia, si rende necessaria la programmazione totale del VI FPGA,

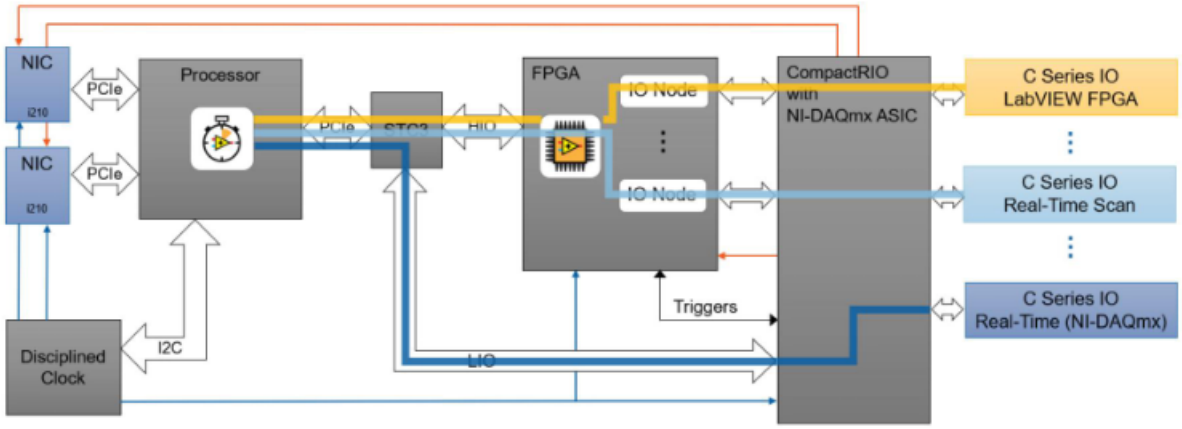

Figura 1.7: *Rappresentazione grafica dei metodi di accesso alle porte I/O.*

permettendo così all'utente di sfruttare a pieno le potenzialità del CompactRIO. I nodi I/O sono pertanto direttamente posizionati nel VI FPGA. Saranno poi necessarie delle soluzioni per l'invio dei dati acquisiti al VI Real-Time.

Il programma per il banco prova è realizzato con il metodo LabVIEW FPGA. Come è stato detto anche in precedenza, esso sarà quindi composto da tre VI principali: quella relativa all'FPGA (che quindi, tra le altre cose, si occuperà di gestire i dati da e verso le porte I/O), quella della parte Real-Time e quella che lavora sul PC host. Queste VI sono inserite nel Project Explorer con l'appellativo di *"MAIN"*, "principale" (*FPGA_Main*, *RT_Main* e *PC_Main*); tutte le altre sono subVI e assolvono a specifiche funzioni all'interno delle principali.

1.3.2 Tag, Stream e Message

Quando si programma un target RIO, è importante essere a conoscenza dei tre paradigmi di comunicazione presenti in LabVIEW:

- Tag;
- Stream;
- Message.

Il tipo *tag*, alle volte chiamato *"current value type"*, è utilizzato "per comunicare l'ultimo valore". Un esempio di tag è il SET di posizione fornito dall'operatore tramite slider sulla HMI. Il tag non ha bisogno di tutto lo storico dei valori del SET, ma solamente dell'ultimo valore in input fornito. Per questa sua peculiarità, la trasmissione del tag non è garantita; il tag fa quindi riferimento a una comunicazione di tipo *lossy*.

La comunicazione di tipo *stream*, diametralmente opposta al tag, prevede il trasferimento di tutti i valori assunti da una variabile. É usata in applicazioni dove il flusso di dati è più importante della latenza. Un esempio di comunicazione stream si ha quando si trasferiscono i dati ottenuti durante un'acquisizione al Real-Time, attraverso il modulo FPGA, per essere poi salvati all'interno di un file. In questo caso, lo storico dei dati è necessario, per cui deve essere garantito il trasferimento di tutti i dati; ci si riferisce quindi a una comunicazione di tipo *lossless*.

L'ultimo metodo di comunicazione, il *message*, è utilizzato quando l'invio dell'informazione deve essere garantito e con una latenza molto bassa. Un comando di stop di emergenza può essere un esempio di comunicazione di tipo message.

1.3.3 Comunicazione tra FPGA, RT e PC

Le effettive modalità di comunicazione tra le VI FPGA, RT e PC Host, *esclusivamente quelle che potrebbero essere usate durante la realizzazione del programma*, vengono riportate nelle tre tabelle di seguito. Nel caso del banco prova, il programma realizzato permette la comunicazione tra FPGA e RT e tra HMI e RT; non si ha comunicazione diretta tra HMI e FPGA. Le modalità vengono distinte in funzione del metodo per cui sono state utilizzate (tag, stream o message). Le tabelle vengono lette partendo dalla prima colonna di sinistra, nella quale viene preso un VI di riferimento. Nelle restanti due colonne, è riportato l'altro VI in comunicazione. Nel caso in cui il secondo elemento sia lo stesso VI di riferimento, si parla di *"Comunicazione interprocesso"* (trasporto dei dati all'interno dello stesso VI). Nella **Tabella 1.1** e nella **Tabella 1.2** vengono presi come riferimenti il VI PC e il VI FPGA, rispettivamente. Avendo già descritto la comunicazione con il VI FPGA nella **Tabella 1.2**, la tabella avente come riferimento il VI RT (**Tabella 1.3**) riguarda esclusivamente le comunicazioni interprocesso al Real-Time, tra loop deterministici e non.

da/per	**HMI (Comunicazione interprocesso)**	**RT**
HMI	Tag: • Variabile Locale Stream: • Channel Wires • Code	Tag: • Variabile Condivisa Stream: • Network Streams

Tabella 1.1: Modalità di comunicazione tra HMI e RT.

da/per	**FPGA (Comunicazione interprocesso)**	**RT**
FPGA	Tag: • Variabile Locale Stream: • FPGA Memory Items	Tag: • Controllo Read/Write Stream: • DMA FIFO

Tabella 1.2: Modalità di comunicazione tra FPGA e RT.

da/per	**da Loop Non-Deterministico a Loop Non-Deterministico**	**da Loop Deterministico a Loop Non-Deterministico**
RT (Comunicazione interprocesso)	Tag: • Variabile Locale Stream: • Code	Stream, Messaggi: • Funzioni RT FIFO

Tabella 1.3: Modalità di intercomunicazione utilizzate nel VI RT.

I metodi appena elencati verranno descritti singolarmente nei paragrafi successivi.

Variabili Locali, Globali e Condivise

Le *variabili locali* (*"Local Variables"*) sono usate per comunicazioni di tipo tag tra diversi oggetti posti nel Front Panel. Nello specifico, vengono utilizzate quando non si ha accesso diretto, o agevole, agli elementi da mettere in comunicazione oppure quando è necessaria la trasmissione tra nodi del Block Diagram.

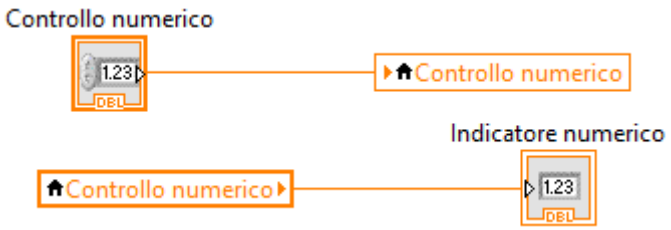

Figura 1.8: *Scrittura e lettura di una variabile locale.*

Con una variabile locale, è possibile leggere da un controllo o scrivere su un indicatore posti nel Front Panel; questo tipo di variabile è infatti utilizzabile sia in lettura che in scrittura. Come riportato in **Figura 1.8**, la scrittura e la lettura delle Local Variable risultano procedure abbastanza intuitive, simili al passaggio di dati tra due terminali qualsiasi. Nel Block Diagram, l'icona della variabile si modifica in funzione del tipo di dato trasportato dal filo a cui è connessa, assumendone la stessa colorazione.

Le *variabili globali* (*"Global Variables"*) sono utilizzate per l'accesso e lo scambio dei dati tra diversi VI, in esecuzione simultanea *sulla stessa macchina*. Alla creazione di una variabile globale, LabVIEW crea in automatico un speciale VI globale, dotato di Front Panel e non di Block Diagram. Qui è possibile aggiungere controlli e indicatori per definire il tipo di dato contenuto nella variabile. Infatti, il Front Panel costituisce l'accesso attraverso il quale i diversi VI accedono ai dati.
L'utilizzo delle *variabili condivise* (*"Shared Variables"*) è simile a quello delle locali e delle globali; tuttavia, esse permettono la condivisione dei dati attraverso una rete. Variabili globali e condivise sono raccolte in apposite librerie site nel Project Explorer.

Figura 1.9: *Esempio di variabile condivisa, vista all'interno del Block DIagram. A sinistra viene mostrato il blocco generico della Shared Variable, che deve essere ancora associato a una variabile specifica.*

Una variabile condivisa può essere di due tipi:

- **Single-Process Shared Variable** - Tipologia usata per mettere in comunicazione loop di processi deterministici e non. Risulta relativamente semplice da implementare nel codice e, nel suo funzionamento, è quella più simile a una variabile globale.
- **Network-Published Shared Variable** - Permette di leggere e scrivere variabili condivise attraverso la rete Ethernet. L'implementazione nella rete è gestita completamente dalla stessa variabile.

Una volta inizializzata una variabile di tipo Single-Process, è possibile convertirla in Network-Published e viceversa. Il programmatore può configurare completamente la variabile, attraverso l'apposita finestra delle proprietà raggiungibile dal Project Explorer (**Figura 1.10**).

Nel caso in cui l'intero processo venga svolto all'interno di una sola macchina, è fortemente consigliato l'uso delle variabili globali. Nel caso in cui, anche in un'ottica futura, si intenda ampliare il processo su un network, è conveniente l'utilizzo delle variabili condivise. L'utilizzo improprio delle variabili condivise, utilizzate come globali, comporta uno spreco dal punto di vista computazionale.

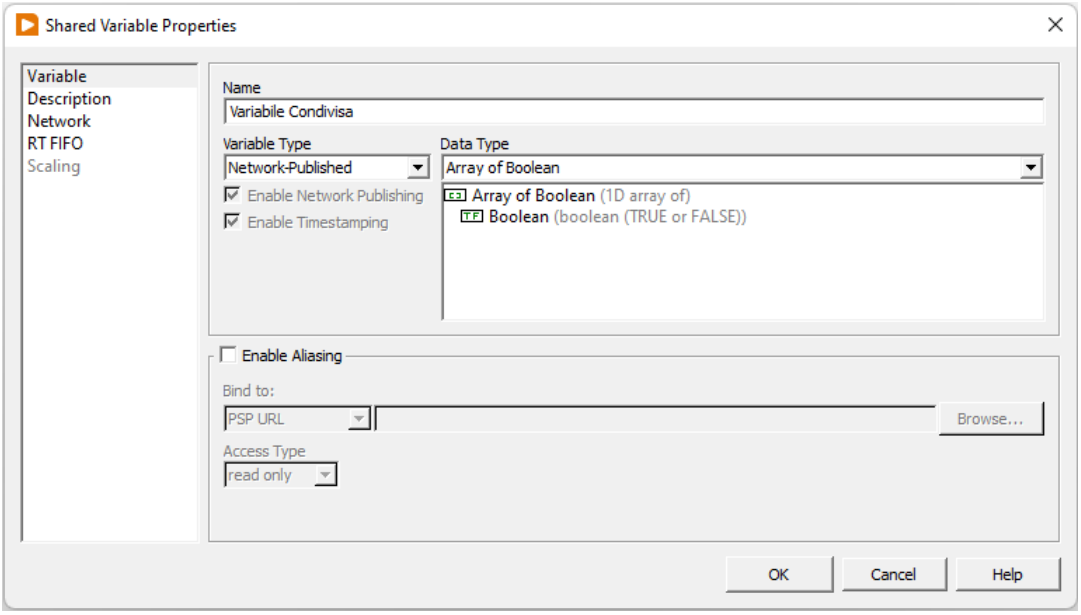

Figura 1.10: *Schermata per la configurazione delle Shared Variable.*

Code

Una *coda* (*"Queue"*) è un insieme bufferizzato di elementi che mantiene un ordine *first-in/first-out* (*FIFO*) dei dati. In LabVIEW, una coda può essere usata per mettere in comunicazione processi diversi all'interno di un programma. Essa può essere creata attraverso le *Queue Operations*, disponibili tra le funzioni del Block Diagram.

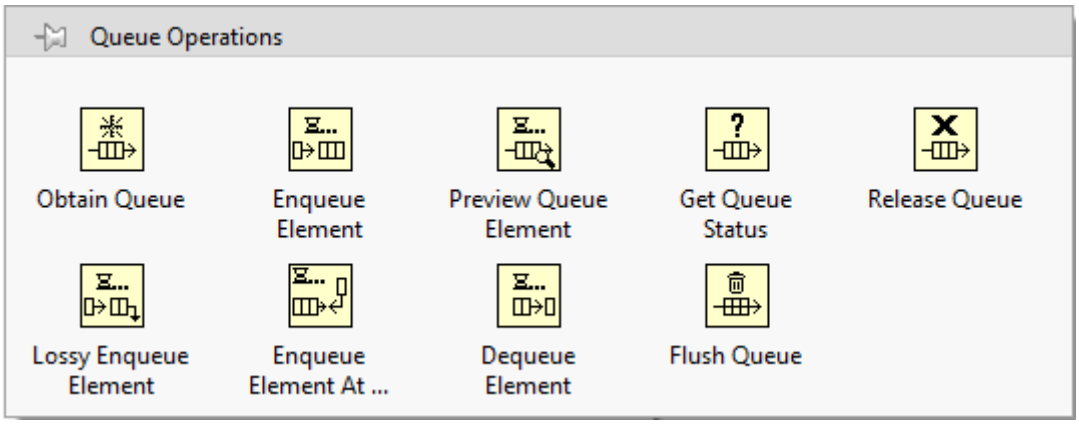

Figura 1.11: *Gamma di operazioni disponibili per le code.*

A differenza degli array, non è possibile accedere in maniera casuale agli elementi di una coda. Essa lavora come un vero e proprio buffer, permettendo esclusivamente l'inserimento (*"enqueue"*) o la rimozione (*"dequeue"*) dei dati dal suo inizio o dalla fine. L'unico modo per visualizzare tutti gli elementi di una coda è, quindi, quello di rimuoverli uno per uno. Vien da sé l'impossibilita di manipolare in qualsiasi modo gli elementi contenuti al suo interno.

Di seguito si riportata una breve descrizione delle operazioni possibili con le code code, rappresentate nella **Figura 1.11**:

Oggetto	**Descrizione**
Obtain Queue	Fornisce il riferimento a una coda.
Enqueue Element	Aggiunge un elemento al fondo della coda, dopo l'ultimo elemento inserito.
Preview Queue Element	Fornisce l'ultimo elemento inserito nella coda, senza eliminarlo da essa.
Get Queue Status	Fornisce informazioni riguardo lo stato attuale della coda, come ad esempio il numero corrente di elementi al suo interno.
Release Queue	Elimina il riferimento a una coda.
Lossy Enqueue Element	Aggiunge un elemento alla coda. Se essa è piena, questa funzione ne rimuove un elemento dall'inizio, scartandolo per creare spazio. A differenza della funzione *Enqueue Element*, questa non apetta che ci sia spazio disponibile nella coda. La dimensione massima della coda è impostata tramite il blocco *Obtain Queue Function*.
Enqueue Element At Opposite End	Aggiunge un elemento all'inizio della coda.
Dequeue Element	Rimuove un elemento dall'inizio della coda e lo restituisce.
Flush Queue	Rimuove tutti gli elementi dalla coda, restituendoli come array.

Tabella 1.4: Descrizione di tutte le Queue Operations.

Nella **Figura 1.12** viene riportato un esempio di utilizzo delle code all'interno del Block Diagram, per il trasferimento di dati tra i due loop.

Channel Wires

Un *Channel Wire* è un particolare filo all'interno di LabVIEW che esprime una comunicazione asincrona tra due sezioni parallele di codice, senza imporre un ordine di esecuzione. L'utilizzo che si fa dei Channel Wire è simile a quello delle code o di una variabile che viene scritta in un loop e letta nell'altro parallelo: si usano per il trasferimento di dati da un loop

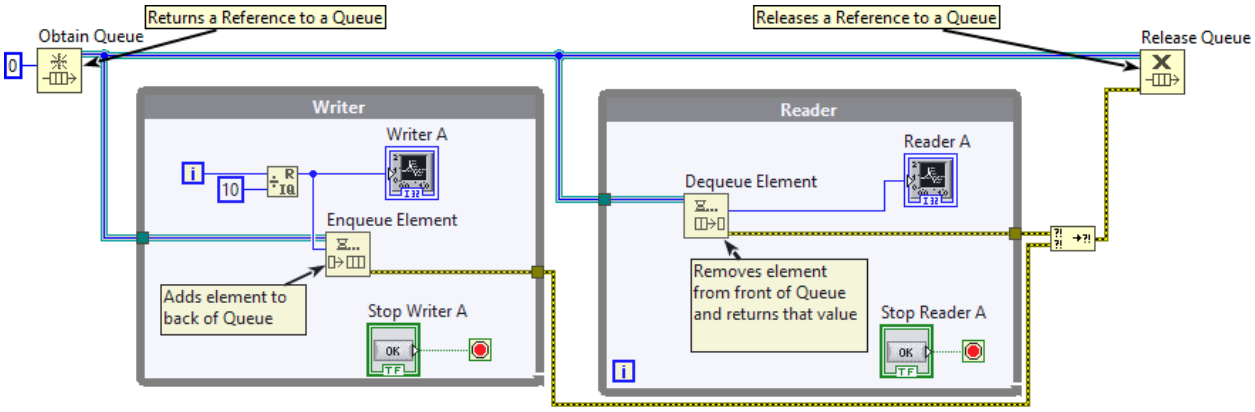

Figura 1.12: *Esempio di utilizzo delle code allìinterno del Block Diagram.*

all'altro, senza che questi vengano arrestati.
Come è possibile osservare dalla **Figura 1.13**, un Channel Wire ha necessità di due endpoint: uno scrittore e un lettore. A differenza delle code o delle variabili, questi canali riescono ad astrarre complessi pattern di trasferimento dei dati, permettendo che questi vengano rappresentati tramite un solo filo. L'utente è in grado di sviluppare un'applicazione funzionante in maniera più agevole, riducendo la chance di compiere errori durante la programmazione. La maggiore semplicità trova conferma nel confronto diretto tra la **Figura 1.13** e la **Figura 1.12**, nelle quali vengono utilizzati entrambi i metodi per trasportare lo stesso elemento.

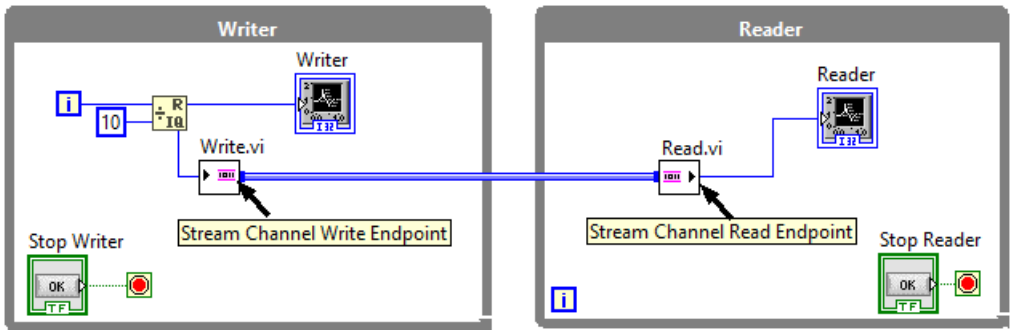

Figura 1.13: *Esempio di Channel Wire, utilizzato per un trasferimento "stream" di dati tra i due loop.*

I canali possono trasportare efficacemente **qualsiasi** tipo di dato presente su LabVIEW e sono utilizzabili con tutti metodi di comunicazione tag, stream e message. I canali vengono configurati tramite un'apposita schermata, rappresentata nella **Figura 1.14**. Al suo interno, oltre ai template di base, si nota la presenza di alcune varianti particolari (*"Accumulator Tag"*, *"Event messenger"* etc.) e di più opzioni per ciscuno di essi (*"Write"*, *"Write Multiple"* etc.). Per eventuali approfondimenti, si rimanda alla specifica documentazione online di LabVIEW [16].

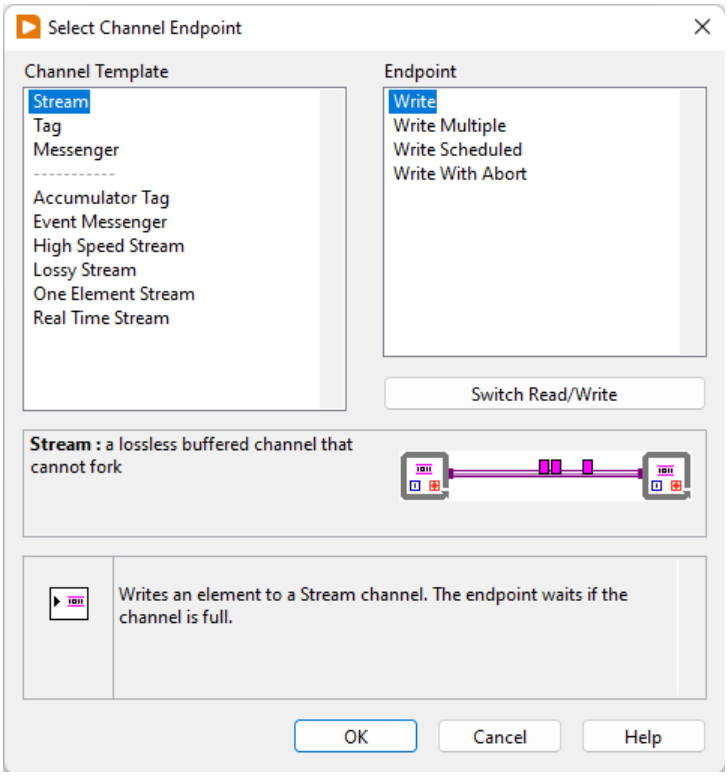

Figura 1.14: *Schermata di inizializzazione di un Channel Wire.*

RT FIFO

Le funzioni *Real-Time FIFO* forniscono un metodo di trasferimento di dati di tipo deterministico, che non aggiunge interferenze a un VI di stampo *time-critical*. Un RT FIFO è una forma di comunicazione di tipo lossy, che sovrascrive i dati meno recenti quando è pieno.
La configurazione che assumono all'interno del Block Diagram è molto simile a quella delle code, come visibile in **Figura 1.15**. Nonostante anche il loro scopo sia fondamentalmente lo stesso, RT FIFO e code presentano alcune differenze sostanziali:

- I RT FIFO funzionano deterministicamente in codici ti tipo time-critical, le code no. Ciò deriva dal fatto che le Queue Functions utilizzano le *blocking calls*[1]quando leggono/scrivono da/su una risorsa condivisa, mentre i FIFO le *non-blocking calls*.
- I RT FIFO hanno una dimensione fissata mentre le code crescono man mano che vengono loro aggiunti elementi.
- Le code funzionano con qualsiasi tipo di dato, mentre i dati che possono essere usati dagli RT FIFO sono limitati. Solitamente, ogni tipo di dato il cui utilizzo implica un'allocazione aggiuntiva di memoria non può essere usato con i FIFO Real-Time, per poterne preservare il determinismo.

[1]Si parla di *"blocking call"* quando la funzione non si avvia in assenza di input ed effettivamente "blocca" l'esecuzione del loop che la contiene. Intuitivamente, ciò non avviene nel caso delle *non-blocking call*.

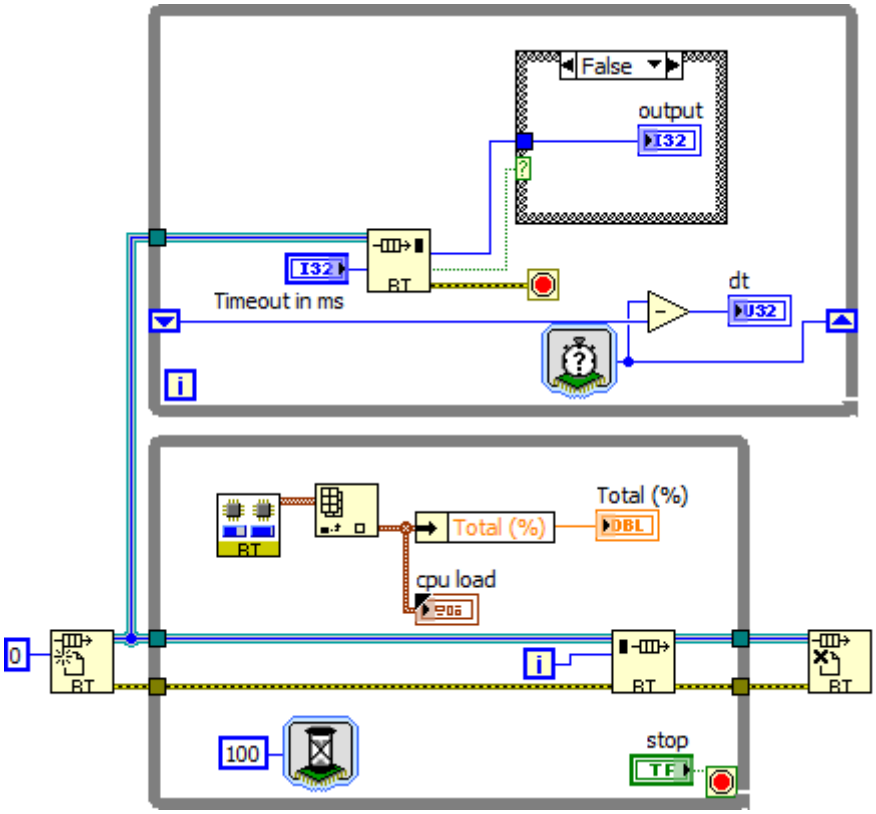

Figura 1.15: *Esempio di utilizzo degli RT FIFO.*

Network Streams

I Network Streams sono ottimizzati per comunicazioni lossless e *high-throughput*[2]; essi vengono utilizzati per il trasferimento di dati da un'applicazione all'altra. Per fare ciò, i Network Streams adottano un modello di comunicazione *one-way* e *point-to-point*; ciò significa che uno degli endpoint è l'elemento scrittore dei dati e l'altro il lettore. Inoltre, per ottenere una comunicazione bidirezionale, è necessario usare due stream separati, dove ogni applicazione contiene un lettore e uno scrittore, appaiati al corrispettivo scrittore/lettore nell'altra.

LabVIEW identifica ogni endpoint dello stream con un URL. Al fine di connettere due endpoint e creare uno stream valido, bisogna fornire l'URL di un endpoint remoto tramite le funzioni *Create Network Stream Endpoint Reader* o *Create Network Stream Endpoint Writer*.

Facendo riferimento alla **Figura 1.16**, quando il loop di lettura/scrittura si interrompe e lo stream non è più necessario, è possibile eliminarlo attraverso la funzione *Destroy Stream Endpoint*. Tuttavia, nel caso ci si volesse accertare che tutti i dati trasmessi dallo scrittore vengano ricevuti dal reader endpoint, è necessario richiamare la funzione *Flush Stream* prima di quella di eliminazione. La funzione di flush può essere richiamata solo dal lato scrittore. All'inizio, la funzione flush richiede che tutti i dati presenti nel buffer del writer endpoint

[2]Tipo di comunicazione nella quale viene trasportato un numero molto elevato di data point.

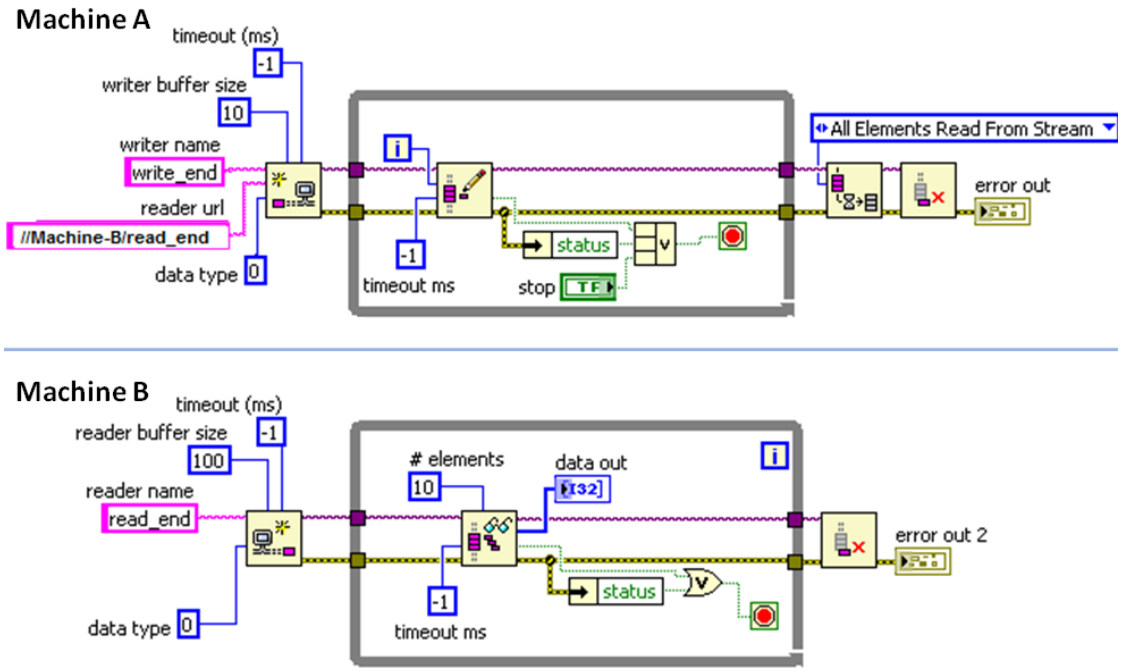

Figura 1.16: *Diagramma esemplificativo di come i Network Streams vengono configurati nel Block Diagram. La macchina A contiene lo scrittore e la B il lettore.*

vengano immediatamente trasferiti attraverso la rete. Successivamente, essa attende che la *wait condition* venga soddisfatta. É possibile scegliere tra due wait condition:

- *All Elements Read from Stream*;
- *All Elements Available for Reading.*

Quando si specifica "All Elements Read from Stream", la funzione attende che tutti i dati vengano trasferiti al reader endpoint e vengano letti da esso. L'opzione "All Elements Available for Reading" attenderà sempre che i dati vengano trasferiti al lettore, ma non che essi vengano letti dall'endpoint.
Un errore nella Flush Function prima dell'eliminazione dell'endpoint implica il rischio di perdere i dati residui nel buffer del writer endpoint e/o quelli in transito.
Una volta che la funzione destroy viene richiamata sul reader endpoint, ogni successivo richiamo dal writer endpoint provocherà un errore. Quando viene eliminato il writing endpoint, la funzione di lettura proseguirà fino a quando il buffer del reader endpoint non si svuota; da questa condizione, ogni richiamo ulteriore provocherà un errore.

FPGA Memory Items

Durante l'esecuzione di un'applicazione FPGA, gli elementi di memoria sono i mezzi principali di archiviazione dei dati nel dominio di un singolo clock. Il LabvIEW FPGA Module prevede due tipi di elementi di memoria:

- *VI-defined memory items*;
- *Target-scoped memory items.*

Gli elementi di memoria di tipo *target-scoped*, utilizzati per realizzare il programma del banco prova, sono stati scelti perché configurabili dal Project Explorer. I Memory Items presenti nel Block Diagram sono staticamente legati, tramite nome, a quelli corrispondenti nell'albero di progetto. Ciò implica che tutte le modifiche effettuate all'elemento nel Project Explorer si ripercuotono sugli elementi collocati nel Block Diagram. L'utilizzo dei *VI-defined memory items* sarebbe risultato sconveniente nella fase di debugging, in quanto si sarebbe dovuto intervenire sui singoli elementi del Block Diagram nel caso fossero state necessarie modifiche agli elementi di memoria.

Gli elementi di memoria sono configurabili dalla schermata *Memory Properties*, raggiungibile dal Project Explorere riportato in **Figura 1.17**. Al suo interno, si accede a un menù a tendina dal quale è possibile scegliere tra tre tipologie di implementazione dell'elemento: *Block Memory*, *Look-Up Table* e *DRAM*. A ciascun elemento di memoria corrisponde un elemento fisico sull'hardware FPGA, per cui il loro numero è limitato. Inoltre, non tutti i dispositivi *"FPGA-target"* sono dotati di DRAM, come il CompactRio 9047 utilizzato sul banco.

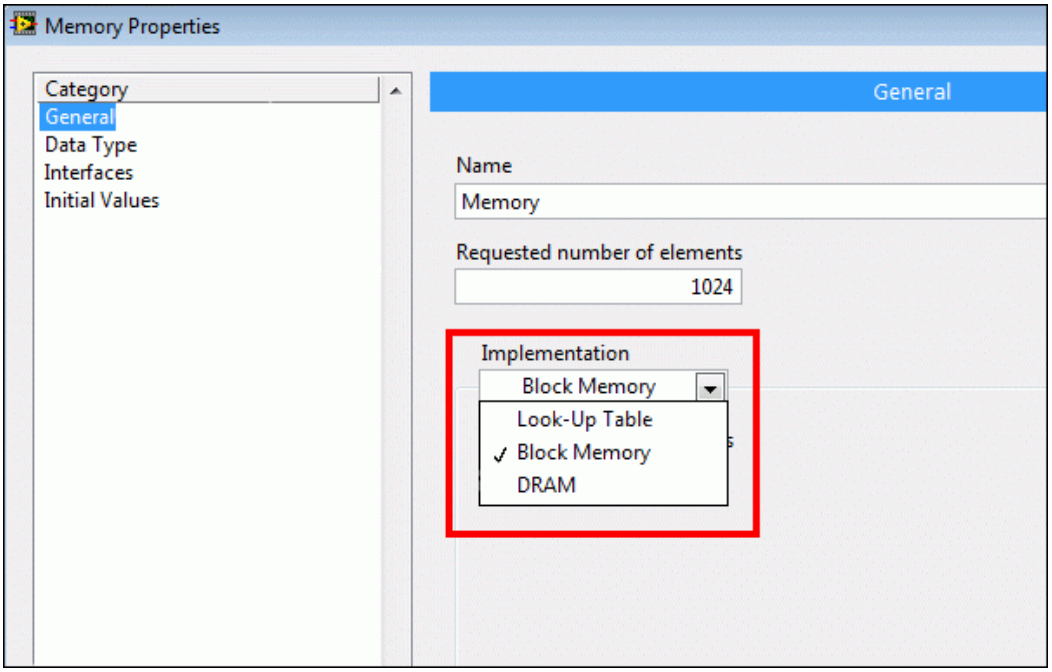

Figura 1.17: *Schermata Memory Properties. Il riquadro rosso evidenzia il menù a tendina "Implementation".*

- **Block Memory** Il *Block Memory* (anche conosciuto come *"block random access memory"*, *"block RAM"* o *"BRAM"*) è una risorsa interna al modulo FPGA per lo storage

di dati. I Memory Items che usano i blocchi di memoria permettono il loro inserimento all'interno di loop con frequenza clock più alta rispetto ad altri tipi di elementi. Con questa implementazione, è possibile usare un solo nodo di scrittura e uno di lettura per ciascuno degli elementi di memoria. Il Block Memory non consuma le risorse dell'FPGA.
Quando la memoria viene implementata tramite Block Memory, il numero di cicli richiesti alla funzione *Read Memory* per produrre un valore numerico valido è uguale al numero di cicli di *read latency*. Questo valore, di default impostato a 2, è altresì regolabile dalle Memory Properties; può assumere valori che vanno da 0 a 3. Dal Block Diagram, il numero di cicli di latenza è visibile nella parte superiore del blocco Read della memoria, come evidenziato in **Figura 1.18**.

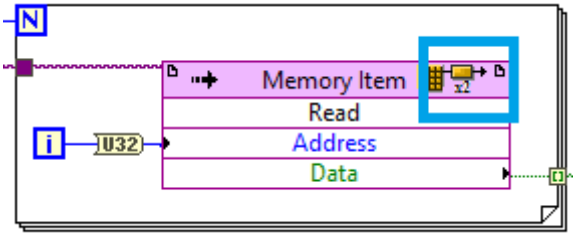

Figura 1.18: *Blocco Read della memoria, con evidenziato in azzurro il numero di cicli di latenza.*

- **Look-up Tables (LUTs)** Le *Look-up Tables*, conosciute anche come *didtributed RAM*, consistono in porte logiche strettamente legate all'FPGA. Le LUTs consumano risorse FPGA, dal momento che possono funzionare sia come FPGA che come memorie. Le Look-up Tables vengono utilizzate nei seguenti casi:
  - Quando si ha bisogno di accedere alla memoria in un *single-cycle Timed Loop* e di leggerne i dati durante lo stesso ciclo nel quale viene invocato il nodo.
  - Quando si hanno limitati Block Memory rimanenti.
- **Dynamic RAM (DRAM)** La *Dynamic RAM* è una sorta di memoria esterna presente su alcuni target FPGA; essa fornisce un grosso quantitativo di spazio di archiviazione. Tuttavia, dal momento che la DRAM è esterna all'FPGA, l'applicazione non può ricevere dati da essa in un ciclo a singolo clock. La DRAM richiede anche un accesso sequenziale, ovvero che solo un comando alla volta può accedere alla memoria. L'accesso sequenziale impedisce un timing di tipo deterministico e, a seconda di quanti comandi sono in attesa di accedere alla memoria, può aumentare il tempo di esecuzione.
  Quando la DRAM è disponibile, essa viene utilizzata per immagazzinare grandi quantità di dati non allocabili altrove all'interno dell'FPGA. Se la DRAM non è disponibile sul target, essa non comparirà elencata nel menù a tendina Implementation all'interno delle Memory Properties.

A meno che non si abbia specificata necessità dei vantaggi proposti dagli altri due tipi di implementazione, NI suggerisce l'utilizzo dei Block Memory. Non avendo esigenze particolari, le memorie all'interno del programma realizzato per il banco prova sono implementate in questo modo.

DMA FIFO

I *FIFO* vengono utilizzati per il trasferimento di dati tra differenti porzioni di un VI FPGA, più VI di un target FPGA oppure tra dispositivi diversi. Un FIFO è una struttura di dati che mantiene gli elementi nell'ordine in cui sono stati ricevuti e ne permette l'accesso secondo il criterio *"first-in, first-out"* (il termine "FIFO" è appunto l'acronimo ottenuto da questa espressione).
LabVIEW propone diverse tipologie di FIFO, a seconda delle esigenze. Per il codice del banco prova, vengono utilizzati quelli di tipo *Direct Memory Access (DMA)*. Questi FIFO vengono utilizzati per lo stream di dati tra un processore host e l'FPGA; nel caso specifico, sono usati per lo stream tra FPGA e processore RT. I DMA FIFO hanno la peculiarità di allocare memoria sia sull'host che sul target FPGA, col vantaggio di risparmiare risorse su entrambi. Essi permettono un trasferimento di dati di tipo lossless, high-throughtput e ad alta velocità.

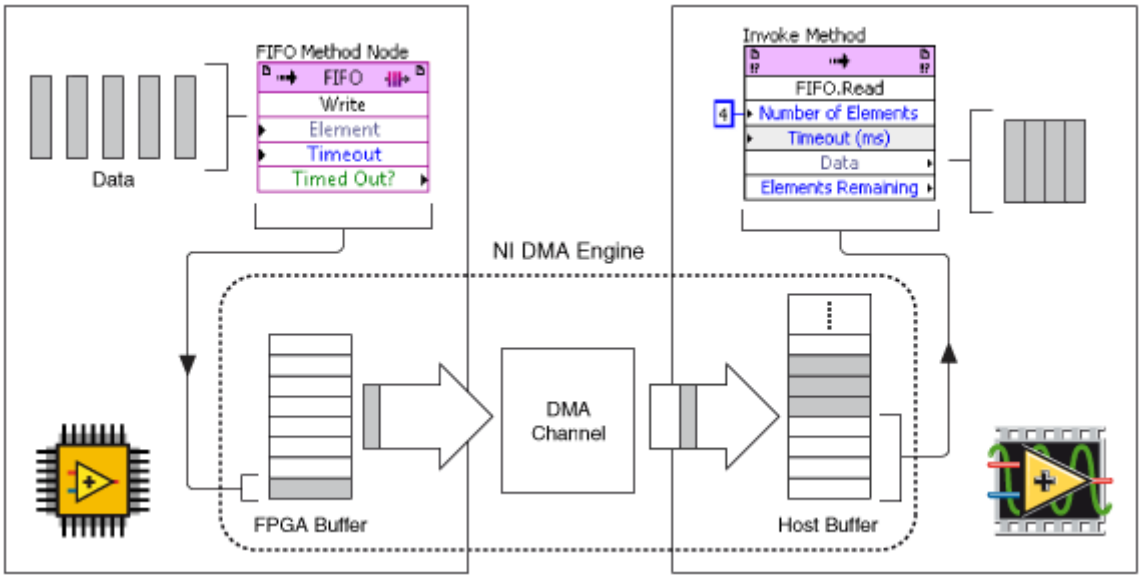

Figura 1.19: *Schema di funzionamento dei DMA FIFO. Si noti la presenza degli endpoint di scrittura e lettura, esattamente come appaiono all'interno del Block Diagram.*

Un canale DMA consiste in due buffer FIFO: uno sull'host e uno sul target FPGA. Una volta inizializzato un DMA FIFO, è necessario creare il codice nel Block Diagram per la scrittura dei dati al suo interno e, sull'altro fronte, per la loro lettura. La comunicazione DMA è unidirezionale: in caso si volesse invertire il verso della comunicazione (e.g. da FPGA verso host a host verso FPGA), è necessaria la creazione di un altro canale DMA.

In **Figura 1.19** è riportato, in forma esemplificativa, lo schema di un canale DMA tra target FPGA e Real-Time. Essendo il suo funzionamento basato sui FIFO, il trasferimento dei dati avverrà un elemento alla volta. Il primo elemento scritto nel primo buffer sarà il primo trasferito, e letto, sull'altro.

I DMA FIFO utilizzati nel progetto sono di tipo *"target-scoped"*. Ciò significa che essi sono visibili e configurabili dal Project Explorer, in maniera analoga a quanto visto con gli elementi di memoria. Inoltre, essi sono accessibili a tutti i VI che nel Prject Explorer stanno al di sotto del target FPGA di riferimento.

Capitolo 2

VI FPGA

Il VI FPGA si occupa principalmente di acquisire i dati dai moduli installati sul CompactRIO. Inoltre, essendo l'FPGA il dispositivo elettronico più performante, al VI dedicato vengono affidati i processi a più elevato carico computazionale.

2.1 Project Explorer e Front Panel

La **Figura 2.1** riporta la sezione di Project Explorer dedicata al VI FPGA e degli elementi correlati. Il file (*"FPGA_Main.vi"*) si trova ivi al di sotto degli elementi *"Chassis cRIO-9047"* e *"FPGA Target (RIO0, cRIO-9047)"*.

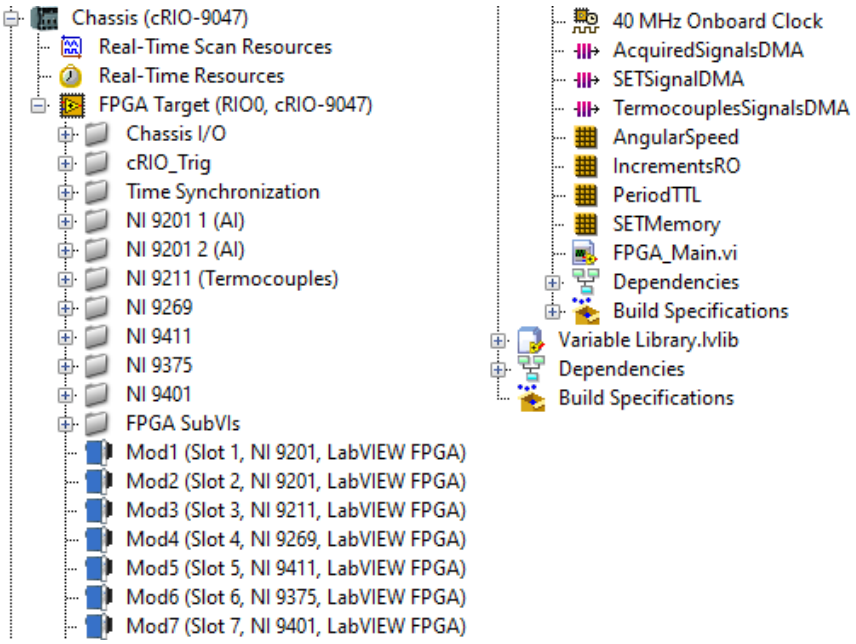

Figura 2.1: *Parte di Project Explorer nella quale è presente il VI FPGA, le memorie, i FIFO, i moduli installati sullo chassis e la cartella con i subVI.*

Allo stesso livello e assieme agli elementi di memoria, i DMA FIFO e la cartella contente i subVI utilizzati nel file principale, sono virtualmente allocati i moduli installati sul CompactRIO. Da qui è possibile rinominare i diversi canali e settarne le proprietà.

Il Front Panel del VI FPGA si divide in due sezioni: il *"Control Panel"* e il *"Security Module"*. Il Control Panel gestisce il funzionamento di tutto il VI e viene riportato nella **Figura 2.2**. Col riquadro in rosso, vengono evidenziati i controlli *"Control enable (RFR)"*, *"Displacement +"* e *"Displacement -"*. Il primo serve per l'attivazione del controllo del driver Lenze tramite CompactRIO. Gli altri due servono per far compiere alla slitta uno spostamento positivo o negativo, tramite una rotazione dell'albero del motore elettrico. **In accordo col sistema di riferimento del banco prova, uno spostamento è considerato positivo quando la slitta si allontana dal motore.**

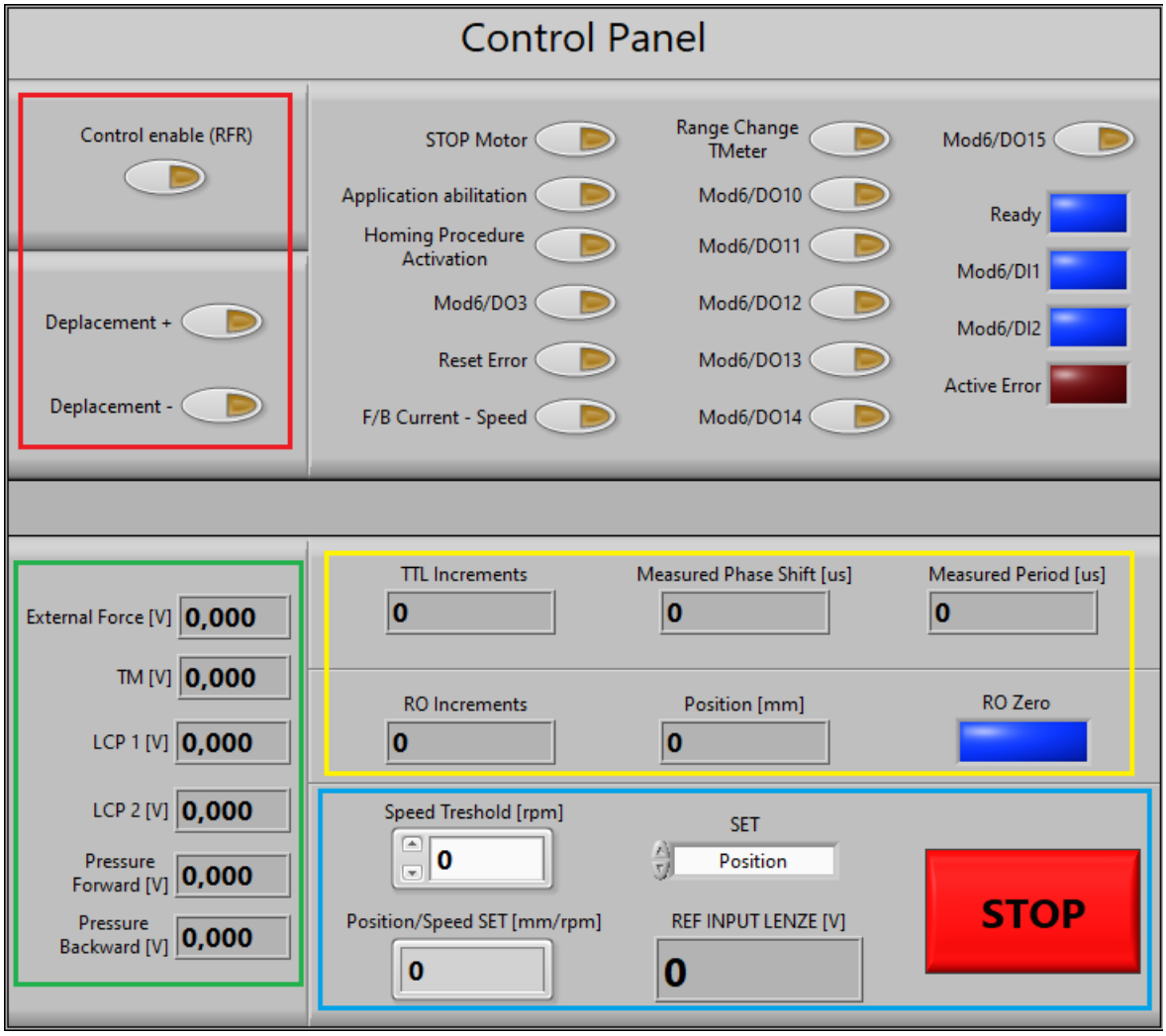

Figura 2.2: *Parte di Front Panel del VI FPGA riportante il "Control Panel".*

Nella restante parte superiore, è stata collocata una serie di controlli e indicatori di tipo digitale, atti al controllo del driver Lenze. Essi costituiscono gli input e gli output del modulo

NI 9375.
Passando alla parte inferiore del pannello, all'interno del riquadro verde sono contenuti degli indicatori numerici che riportano il valore delle tensioni misurate dai sensori analogici, facenti capo al modulo modulo NI 9201. Nel dettaglio, a esso sono collegati: la cella di carico che misura la forza esterna agente sulla slitta, i due sensori di pressione (*"Pressure Forward [V]"* e *"Pressure Backward [V]"*), le due celle di carico del sistema di precarico (*"LCP 1 [V]"* e *"LCP 2 [V]"*) e il torsiometro (solo il segnale analogico della coppia).
Nel riquadro in giallo, sono contenuti gli indicatori numerici degli incrementi, e delle grandezze da essi derivate, della riga ottica lineare e dell'encoder all'interno del torsiometro. Sono presenti inoltre gli indicatori della posizione della slitta, espressa in [mm], e del segnale di zero della riga ottica.
Per concludere, all'interno del riquadro azzurro sono presenti controlli e indicatori relativi al SET inviato al driver Lenze:

- *"Speed Treshold [rpm]"* - Controllo della velocità massima raggiungibile dal motore.
- *"SET"* - Tipo di SET impostato (posizione o velocità).
- *"Position/Speed SET [mm/rpm]"* - Valore istantaneo del SET desiderato.
- *"REF INPUT LENZE [V]"* - Tensione inviata dal modulo NI 9269 al driver.

Il pulsante *"STOP"* serve per arrestare l'esecuzione del VI e ne completa il pannello di controllo.

In **Figura 2.3** è riportato il "Security Module" del Front Panel. Esso contiene i quattro indicatori LED dei sistemi di sicurezza installati sul banco prova.

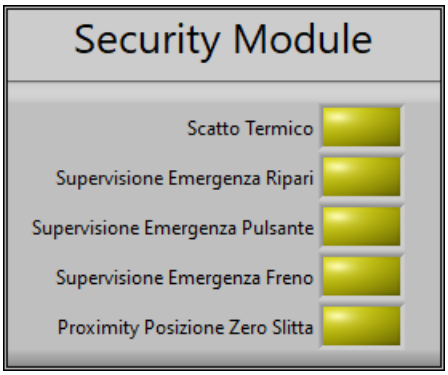

Figura 2.3: *"Security Module" del Front Panel del VI FPGA.*

2.2 Block Diagram

Il Block Diagram si articola in diversi loop, i quali possono essere suddivisi in base alla funzione che svolgono: acquisizione e invio di dati, calcolo degli incrementi e della velocità

angolare, gestione degli I/O digitali. Ciascun loop contiene una Flat Sequence Structure, il cui primo frame contiene il blocco per l'impostazione del timing del ciclo.

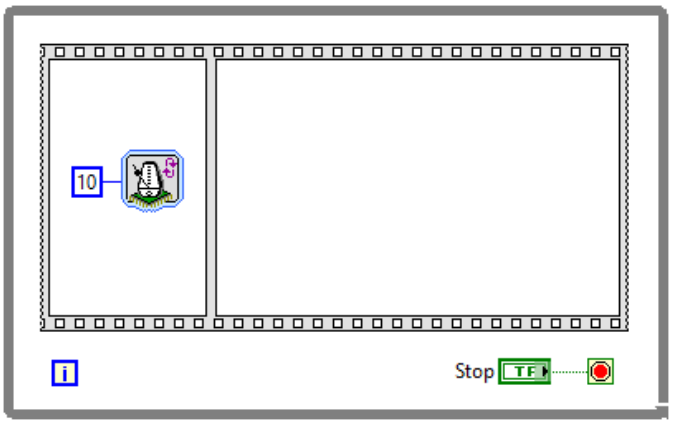

Figura 2.4: *Tecnica per l'impostazione del timing dei loop, specifica per VI di tipo FPGA.*

Questa tecnica (**Figura 2.4**) fa semplicemente parte delle regole di programmazione dei VI FPGA. Nei VI RT e PC, il blocco del timing è inserito direttamente all'interno del loop, senza bisogno della struttura a sequenza.

Come verrà ripetuto anche nei capitoli successivi, la totalità dei dati raccolti viene suddivisa in due acquisizioni separate. La prima, cosiddetta "principale", tiene conto delle tensioni provenienti dai sensori collegati al modulo NI 9201 (celle di carico, sensori di pressione etc.) e degli andamenti di posizione e velocità angolare (valutati indirettamente). L'altra raccoglie le tensioni provenienti dal solo modulo delle termocoppie, collegate al modulo 9211. Questa soluzione è stata necessaria, in quanto il modulo NI 9211 ha una frequenza di sample massima ammissibile molto inferiore rispetto a quella del NI 9201. Se si volesse acquisire da entrambi i moduli all'interno dello stesso loop, bisognerebbe ridurre drasticamente la frequenza di campionamento dal NI 9201, sotto-sfruttandone drasticamente le potenzialità. Inoltre, ciò avrebbe forti ripercussioni sulla qualità stessa dell'acquisizione principale. Le tensioni provenienti dal modulo 9201 vengono quindi campionate alla frequenza di 1 kHz, mentre quelle dal modulo NI 9211, provvisoriamente[1], a 2 Hz. Per mantenere l'andamento di posizione e velocità angolare in linea con l'acquisizione principale, il loop che si occupa di calcolare il numero degli incrementi è impostato ugualmente a 1 kHz.

All'interno del VI FPGA, vengono trattati dati prevalentemente di tipo *"Fixed-Point"* (*"FXP"*)(in italiano, "a virgola fissa"). Durante la realizzazione del codice si è scelto di operare con dati numerici con *word* di 32 bit, dei quali 20 bit dedicati alla parte intera. Tutte le conversioni presenti servono per uniformare la dimensione dei dati a questi valori.

[1]La parte relativa alla misurazione della temperatura, come si vedrà anche nella parte riguardante il VI Real-Time, è ancora in fase di sviluppo e non è stata conclusa in questa sede.

2.2.1 Acquisizione dati e segnale di SET

Main Acquisition Il loop riportato in **Figura 2.6** è dedicato alla creazione dell'array contenente i dati dell'acquisizione principale e della sua trasmissione al VI Real-Time. Esso comanda inoltre il segnale di tensione, corrispondente a quello di SET, in uscita dal CompactRIO e in ingresso al driver Lenze del motore elettrico.
Analizzando la figura, si nota la presenza di tre elementi di memoria: *"AngularSpeed"*, *"IncrementRO"* e *"SETMemory"*. Rispettivamente, essi contengono il valore istantaneo della velocità angolare, il numero di incrementi provenienti dalla riga ottica lineare e il valore istantaneo del SET. La velocità angolare viene calcolata in un altro ciclo While all'interno dello stesso VI. Gli incrementi compiuti dalla riga ottica costituiscono l'input di un subVI (*"IncrementToMM_RO.vi"*), al cui interno viene calcolata la posizione della slitta; ne viene riportato il Block Diagram in **Figura 2.5**. La posizione, espressa in [mm], viene ivi ottenuta moltiplicando il numero di incrementi per un fattore di conversione di $-0{,}0005$. Questo si ottiene partendo dalla risoluzione dell'encoder lineare, il cui valore sul data sheet è pari a $1\,\mu$m per incremento conteggiato. Questo valore si riferisce a ciascuna fase e considerando sia il fronte di salita che di discesa dell'onda quadra. Infatti, se si considerasse solo il fronte di salita del segnale, la risoluzione sarebbe di $2\,\mu$m, nonché lo spazio fisicamente presente tra una tacca e un vuoto sulla riga ottica. Data la presenza di due fasi, la risoluzione risulta dimezzata rispetto a quella presente nel data sheet, raggiungendo un valore di $0{,}5\,\mu$m ($0{,}0005$ mm) per incremento. La costante viene resa negativa per rendere coerenti la posizione misurata e il sistema di riferimento del banco prova (verso considerato positivo quando ci si allontana dal motore).

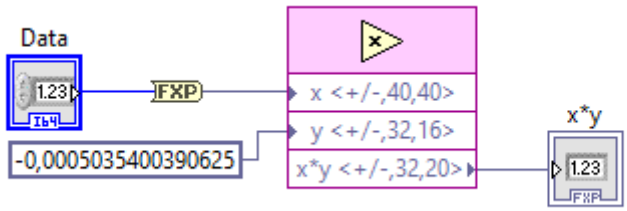

Figura 2.5: *Block Diagram del subVI "IncrementToMM_RO.vi".*

I valori di SET, velocità angolare della vite e posizione lineare della slitta rappresentano gli input del subVI *"SignalsAcquisition.vi"* (**Figura 2.7**). Assieme ai segnali acquisiti dai moduli, questi convergono in un blocco *"Build Array"*, creando il vettore di dati *"AcquiredSignals"*, formato da nove elementi e output del subVI. Al suo interno, sono posizionate le porte I/O che permettono l'acquisizione dei valori di tensione dai moduli del CompactRIO. Si ricorda che il loop in esame riguarda esclusivamente la sola acquisizione principale: i segnali proverranno esclusivamente dai sensori collegati al modulo NI 9201.
Dall'immagine si nota come i dati appena acquisiti, già di tipo FXP, passino per dei blocchi di conversione in FXP. Ciò serve a trasformare i numeri in virgola fissa dalla dimensione di

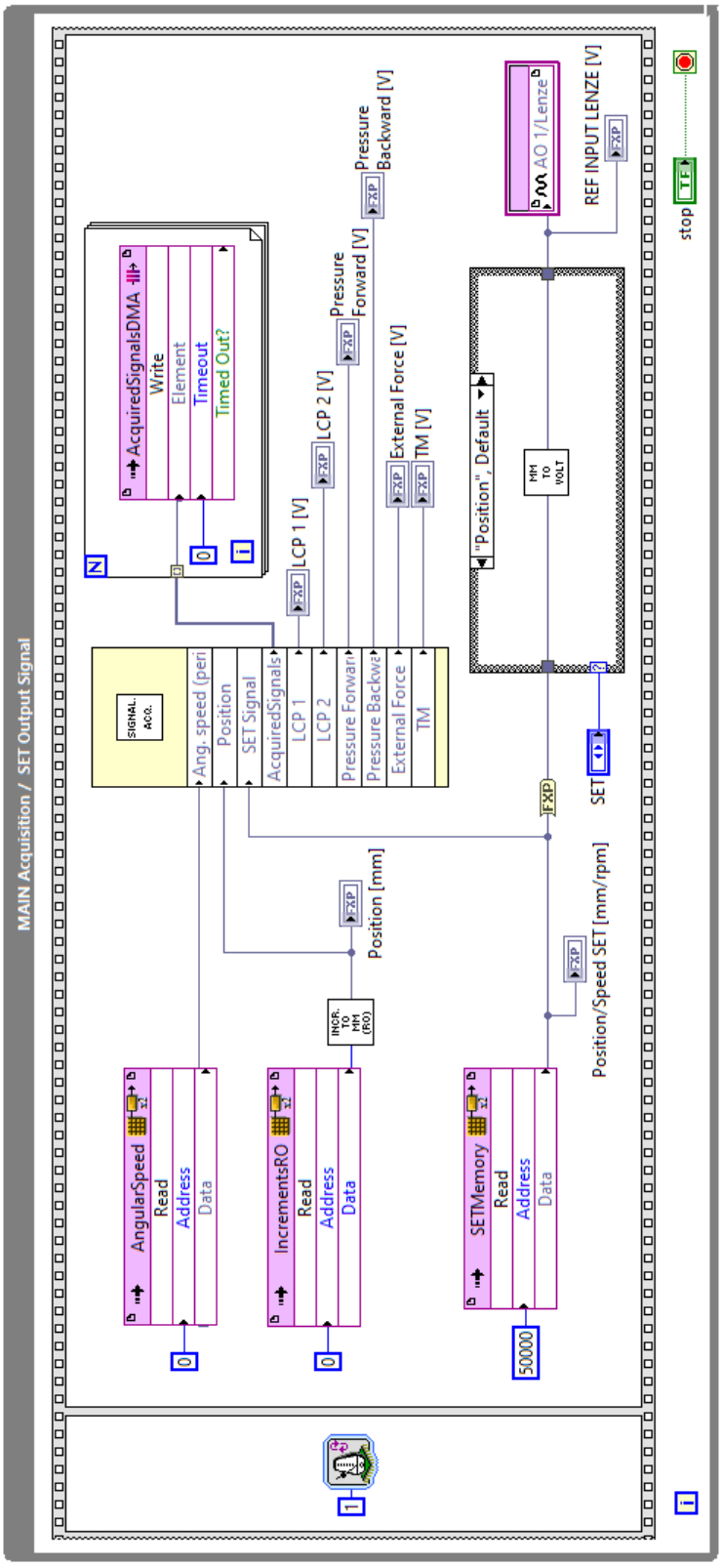

Figura 2.6: *Loop per l'acquisizione e l'invio dei dati dell'array principale al VI RT e l'invio del SET al driver del motore.*

16 bit (di cui 5 bit per la parte intera), nonché la dimensione assegnata automaticamente da LabVIEW ai valori delle tensioni acquisiti, ai 32 bit utilizzati nel resto del VI FPGA.

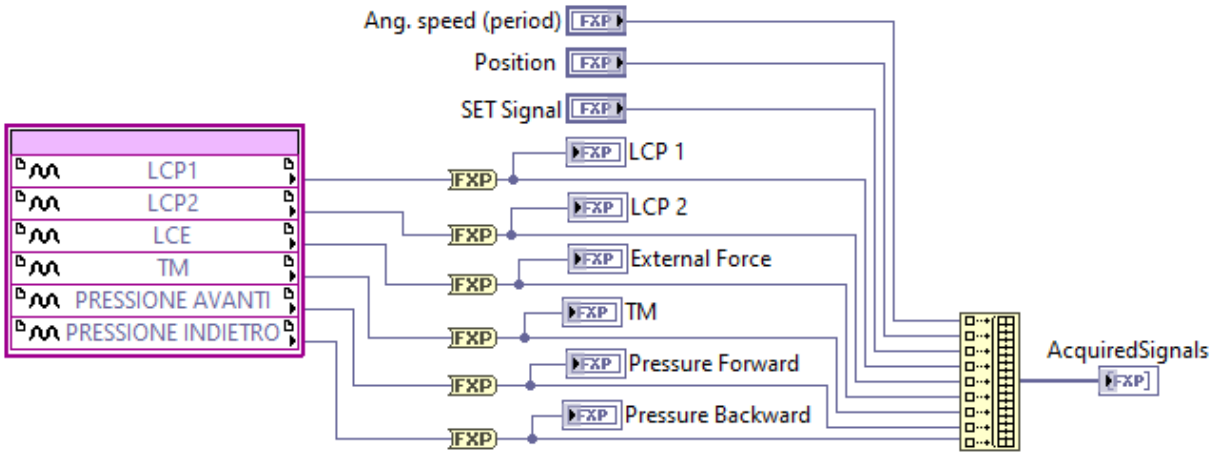

Figura 2.7: *Block Diagram del subVI "SignalsAcquisition.vi".*

L'array in uscita dal subVI appena descritto va in ingresso a un ciclo For (**Figura 2.8**) contenente il blocco per la scrittura dei dati nel DMA FIFO, i quali verranno letti nel VI Real-Time. Normalmente, il FIFO permette la scrittura di un solo elemento per volta. Grazie alle funzionalità di *Indexing* del ciclo For, questa procedura viene svolta in automatico, permettendo al FIFO di scrivere l'intero array a ogni iterazione.

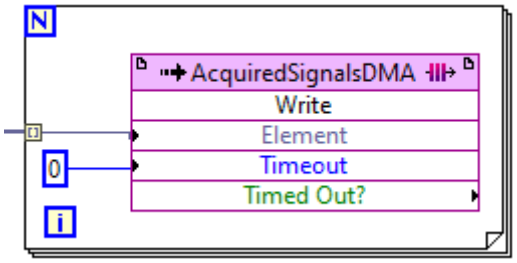

Figura 2.8: *Ciclo For per la scrittura dei dati nel FIFO dell'acquisizione principale.*

La parte del loop dedita all'invio del segnale di SET è riportata in **Figura 2.9**. Essa è costituita da una Case Structure, comandata da una costante di tipo "enum" e contenente due casi: *"Position"* e *"Speed"*. Per ciascun caso, è presente un subVI che converte il valore della grandezza nel corrispettivo in tensione. La strutturazione del Block Diagram non varia per i due case e ne viene riportato uno esemplificativo in **Figura 2.10**. Il valore di tensione rappresenta l'uscita della Case Structure e l'input del blocco I/O posto all'esterno della struttura.
Come già accennato, il loop dell'acquisizione principale contiene una memoria dalla quale viene letto il segnale di SET da inviare al driver. Il valore al suo interno viene scritto all'interno di un altro loop (*"SET Signal From RT"*, riportato in **Figura 2.11**) e proviene

da un DMA FIFO, a sua volta scritto nel VI RT, nella sezione dedicata alla generazione del segnale.

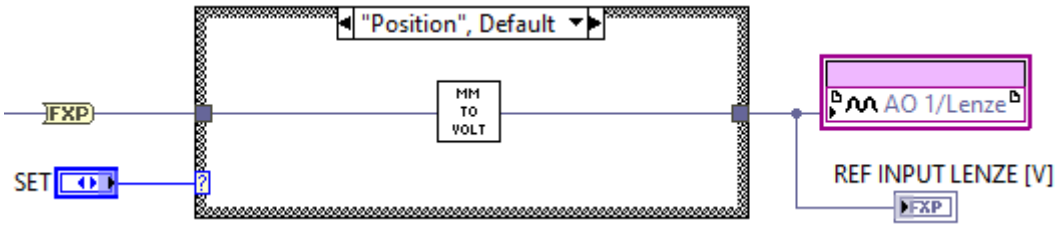

Figura 2.9: *Sezione di Block Diagram per l'invio del segnale di SET al driver Lenze.*

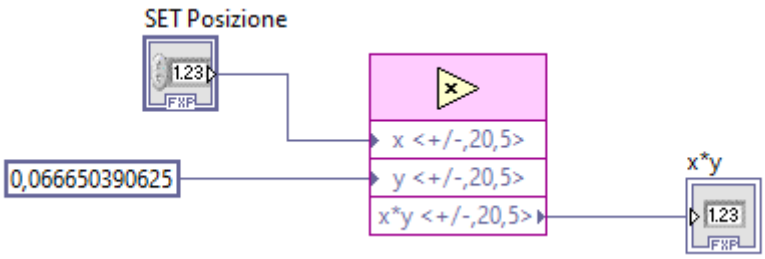

Figura 2.10: *SubVI per la conversione del segnale di SET di posizione, espresso in* [mm]*, nel corrispondente valore di tensione.*

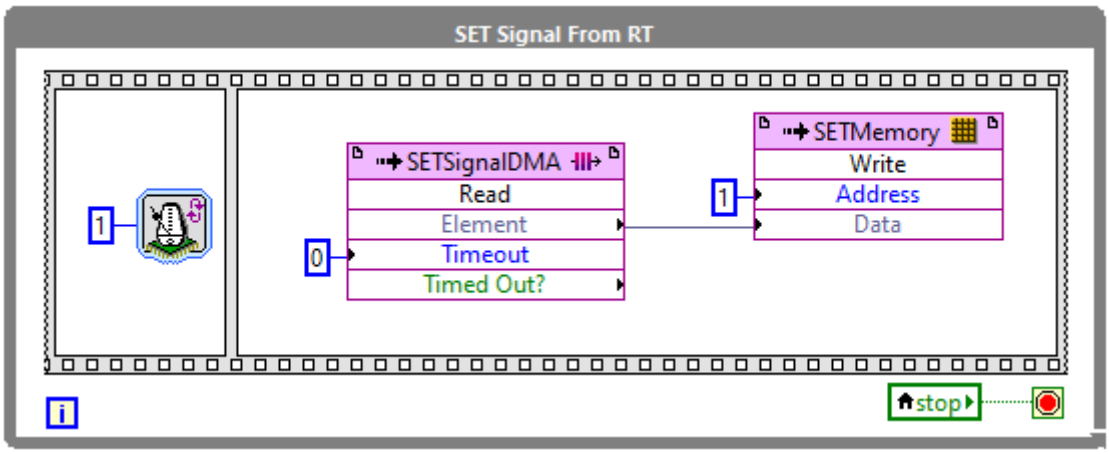

Figura 2.11: *Loop per la lettura del DMA FIFO contenente il segnale di SET.*

Termocouple Acquisition In **Figura 2.12** viene riportato il loop per l'acquisizione dei dati dal modulo NI 9211 delle termocoppie. Questo ciclo mantiene una struttura analoga a quella del loop dell'acquisizione principale: i valori di tensione vengono letti dai blocchi I/O, convertiti, raccolti in un unico array e, infine, inviati al VI Real-Time tramite un FIFO. L'array assemblato contiene un totale di 6 elementi. Anche in questo caso, vengono sfruttate le proprietà di indicizzazione del ciclo For per la scrittura dell'array nel FIFO.

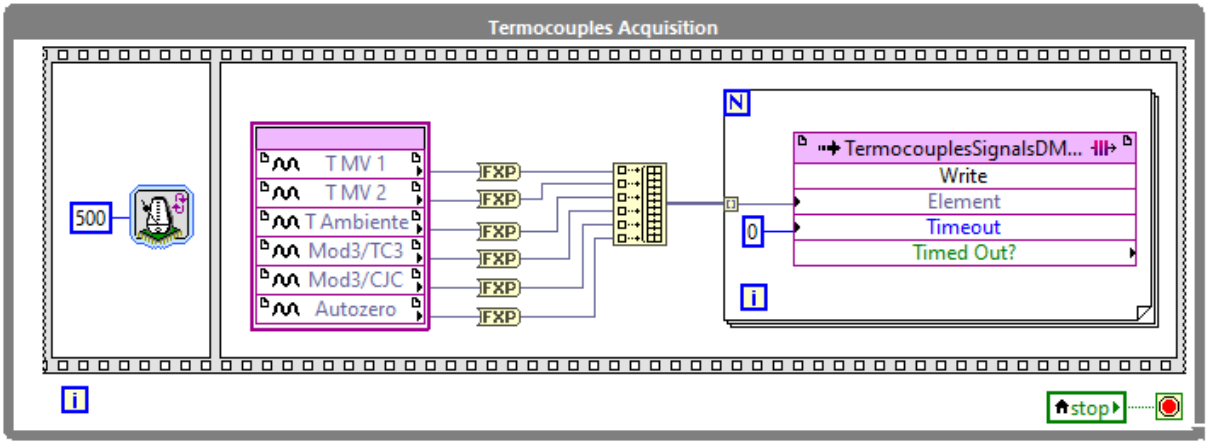

Figura 2.12: *Loop per l'acquisizione dei dati dal modulo termocoppie.*

2.2.2 Calcolo del numero di incrementi

Al fine di determinare la posizione della slitta lungo la guida e la velocità angolare della vite, è necessario il calcolo degli incrementi della riga ottica lineare e dall'encoder rotativo presente nel torsiometro. Per capire come è stata realizzata la relativa parte di codice, è necessario innanzitutto capire il funzionamento di questi dispositivi.
Nella sua definizione generale, un encoder è un trasduttore di posizione che genera un segnale elettrico, analogico o digitale, in funzione del movimento che si intende misurare. In base all'applicazione, può essere di tipo rotativo o lineare. Tra i due, il principio di funzionamento rimane invariato. Un encoder di tipo lineare può essere infatti considerato come uno rotativo virtualmente "srotolato" lungo una direzione retta. Ulteriori discriminanti, tipo di segnale di output e tecnologie utilizzate per la sensoristica, permettono una classificazione più articolata degli encoder. Una distinzione molto importante viene effettuata tra encoder *"assoluto"* e *"incrementale"*. I primi sono in grado di fornire valori di posizione univoci, senza il bisogno di conoscere le posizioni assunte in precedenza; al contrario, quelli di tipo incrementale esprimono la posizione rispetto a un punto iniziale preso come riferimento. In questo ultimo caso, sarà necessaria una *procedura di homing* che permetta di far coincidere il punto di zero misurato dall'encoder con quello fisico del sistema sul quale è installato.
Entrambi gli encoder presenti sul banco prova sono di tipo ottico incrementale. Questi sono dotati di due fasi e possono essere chiamati anche *"encoder a quadratura"*, per via dei segnali a onda quadra in uscita da ciascuna di esse. Le due onde quadre sono sfasate tra loro di 90 °; è proprio tale sfasamento che permette la valutazione del senso di rotazione o, in caso di encoder lineare, del verso della traslazione.
Lo schema di funzionamento di un encoder incrementale ottico di tipo rotativo è presentato in **Figura 2.13**. Si ricorda che le considerazioni presentate sono valide anche per uno di tipo lineare. Un disco trasparente, solitamente in vetro, è solidale all'albero. Sulla sua superficie, esso riporta una corona circolare realizzata come una sequenza periodica di tacche radiali trasparenti e opache. La corona ha il compito di modulare i fasci luminosi emessi da due

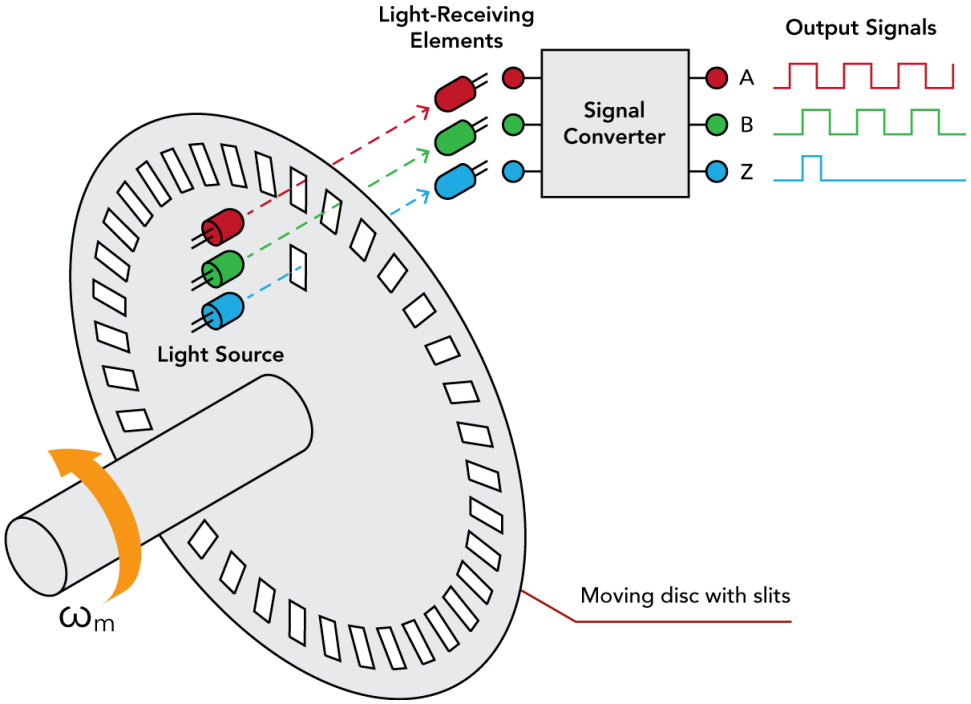

Figura 2.13: *Schema di funzionamento di un encoder incrementale rotativo.*

sorgenti luminose poste su un lato del disco sulla parte fissa dell'encoder (lo *statore*). Sul lato opposto, i fasci di luce modulati vengono intercettati da due sensori ottici. L'elettronica del trasduttore manda in uscita un segnale onda quadra, i cui fronti di salita e discesa dipendono dal fascio luminoso intercettato. Quando il fascio di luce attraversa una delle tacche trasparenti del disco, esso viene captato dal sensore e il segnale a onda quadra va a 1. Al contrario, quando il fascio luminoso colpisce una tacca opaca, il sensore non riceve più luce e l'onda va a 0. Essendoci due sorgenti luminose per due recettori, verranno generati due segnali a onda quadra. Questi, denominati *"A"* e *"B"*, corrispondono alle due fasi in uscita dall'encoder.
Un terzo fascio di luce è modulato da un'altra corona, avente una sola tacca trasparente. Il segnale che ne deriva, denominato *"Z"*, rappresenta *il segnale di zero* del trasduttore che consiste in un singolo impulso a ogni rotazione completa dell'albero. Considerando i trasduttori installati sul banco, l'encoder rotativo non è provvisto di segnale di zero mentre la riga ottica presenta la tacca di zero in mezzeria.
In **Figura 2.14** sono riportati gli andamenti dei due segnali in uscita dal trasduttore, in funzione del verso di rotazione (o di traslazione) dell'elemento mobile dell'encoder.
Durante il normale funzionamento dell'encoder, ciascun incremento viene conteggiato ogni qualvolta viene rilevata una variazione nell'andamento di una delle due fasi. Se la rotazione avviene in verso orario, la variazione del segnale della fase A verrà rilevata prima di quella nella fase B. In corrispondenza del fronte di salita o di discesa di A, il valore istantaneo corrispondente dell'onda B è a esso opposto (se A vale 1, B vale 0 e viceversa). Al contrario, durante la rotazione in senso antiorario, viene rilevata prima la variazione di B. In questo

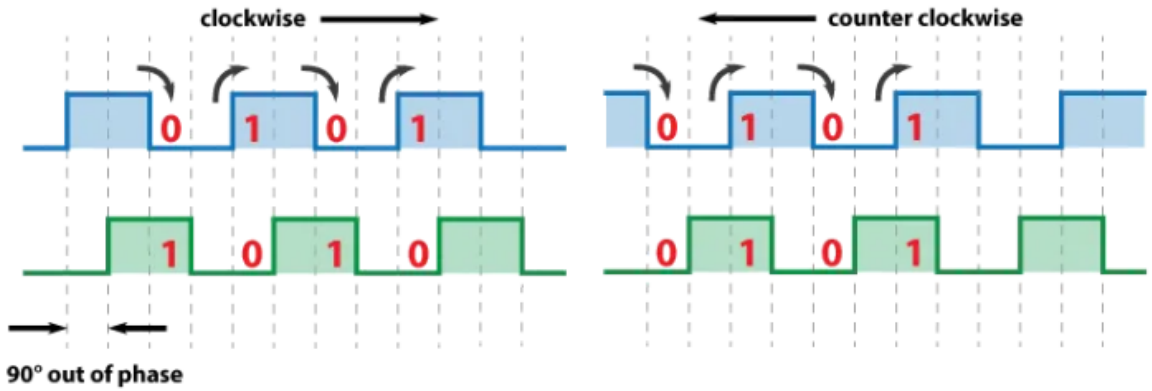

Figura 2.14: *Andamento delle due fasi in uscita dall'encoder, in funzione del verso di rotazione. In azzurro si ha la fase A, in verde la fase B.*

caso, i valori istantanei dei due segnali sono gli stessi. É quindi il confronto tra i valori delle due fasi che permette di stabilire il verso di rotazione dell'encoder.

Implementazione su LabVIEW Quanto spiegato fino ad adesso viene implementato su LabVIEW all'interno di un Timed Loop (il quale garantisce una frequenza di sampling di 40 MHz), al cui interno sono presenti due gruppi di Case Structure annidate: uno per il conteggio degli incrementi della riga ottica e l'altro per quelli dell'encoder rotativo.
La **Figura 2.15** riporta la parte del codice realizzata per il calcolo degli incrementi della riga ottica. Sono presenti tre Case Structure annidate aventi tutte solo due casi: "True" e "False". Lo switch di quella più esterna viene regolato dal segnale di zero che, come detto precedentemente, è un impulso. Quando ci si trova nel punto di zero della riga ottica, condizione "True", il numero di incrementi viene riportato a zero (**Figura 2.17**).
Quando la condizione è falsa, ovvero non ci si trova nella condizione di zero, operano le due restanti strutture. Le loro transizioni dipendono dalle due fasi A e B della riga ottica. Lo switch è comandato dalla funzione *"Not Equal?"*: dati due input, questa restituirà il valore *True* se sono diversi. Il confronto avviene tra il valore corrente del segnale della fase A o B con quello del ciclo precedente, rimandato in ingresso al ciclo tramite uno Shift Register. Si riportano di seguito le combinazioni di casi ottenibili[2]:

- *False/False/False* (**Figura 2.18**) - Le comparazioni dei valori delle fasi A e B producono entrambe "False". I valori correnti delle due fasi sono gli stessi del ciclo precedente: la slitta è ferma. Non essendoci stato alcun incremento, la struttura si limita a rimandare in uscita i valori ricevuti come input.

- *False/False/True* (**Figura 2.19**) - Il valore corrente della fase A è uguale a quello del ciclo precedente. Al contrario, varia quello di B. Questa configurazione può corrispondere a un'inversione del verso di percorrenza della slitta. All'interno della struttura più

[2]Trovandosi nel caso in cui non si è raggiunto il segnale di zero, il primo elemento della combinazione sarà sempre "False". Per evitare ambiguità, si è scelto comunque di specificarlo per ogni combinazione.

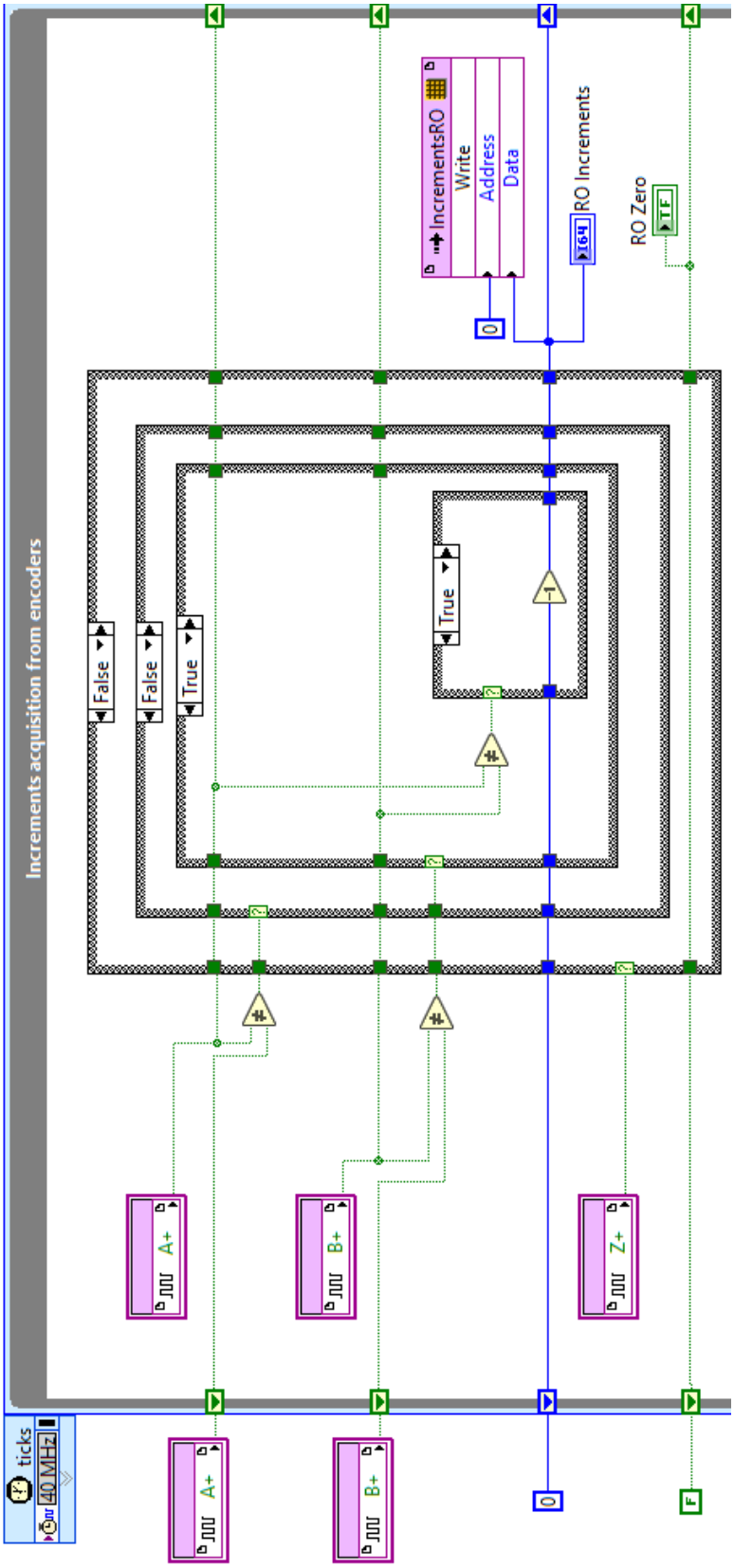

Figura 2.15: *Parte di Timed Loop per il calcolo degli incrementi della riga ottica.*

interna, se ne trova un'altra più piccola che viene comandata da un ulteriore blocco "Not Equal?". Questa ha il compito di aggiungere o sottrarre un incremento a quelli già conteggiati, in funzione del verso di percorrenza della slitta. Questo è determinato dal confronto dei valori correnti delle fasi A e B.

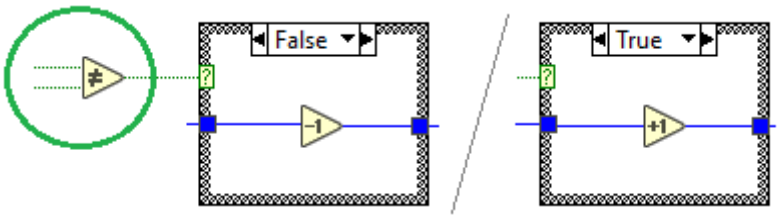

Figura 2.16: *Case Structure interna per il conteggio dei cicli in funzione del verso di traslazione della slitta. Il blocco "Not Equal?", cerchiato in verde, effettua il confronto tra i valori delle fasi A e B.*

- *False/True/*** (**Figura 2.20**) - Il valore della fase A varia, andando a rilevare univocamente uno spostamento della slitta. Infatti, non è necessaria l'aggiunta della terza Case Structure per la fase B. Anche in questo caso, è presente la struttura per l'aggiornamento del numero di incrementi.

Ritornando alla **Figura 2.15**, in uscita dal gruppo di Case Structure, il conteggio degli incrementi aggiornato di ± 1 viene scritto sulla memoria "IncrementsRO". Questo valore viene anche rinviato in ingresso al loop, tramite uno Shift Register. La memoria lavora con numeri interi a 64 bit; un numero di bit così alto è necessario per evitare che un numero di incrementi troppo elevato porti alla saturazione della variabile, generando errori nella lettura della posizione.

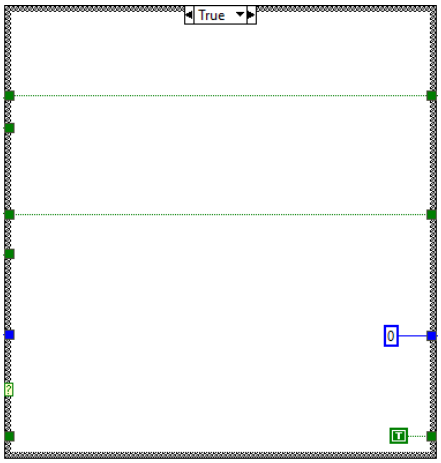

Figura 2.17: *Caso "True" della Case Structure controllata dal segnale Z.*

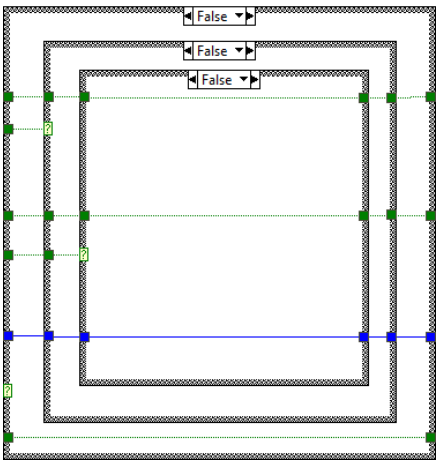

Figura 2.18: *Combinazione "False/False/False" delle Case Structure, sinonimo di slitta ferma.*

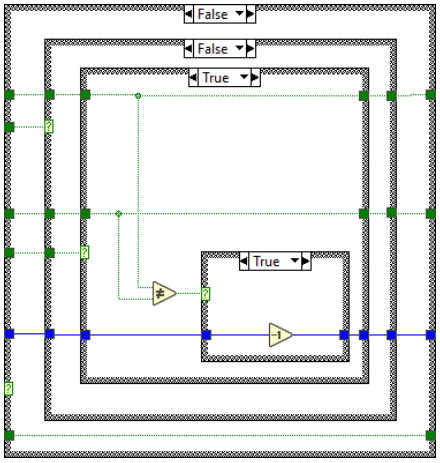

Figura 2.19: *Combinazione "False/False/True" delle Case Structure. Corrisponde a un'inversione del verso della traslazione della slitta.*

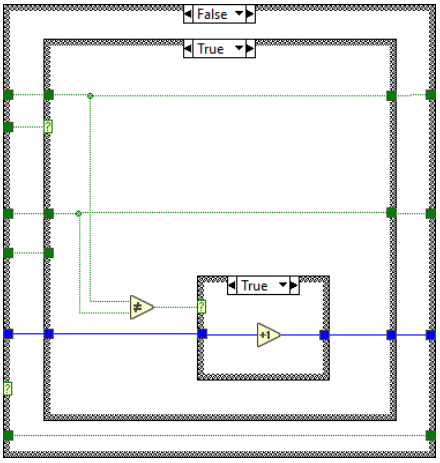

Figura 2.20: *Combinazione "False/True/**" delle Case Structure.*

Il secondo gruppo di Case Structure, per il conteggio degli incrementi dell'encoder rotativo, funziona esattamente allo stesso modo del precedente. L'unica differenza consiste nell'assenza del segnale di zero e, di conseguenza, della case structure più esterna. La specifica parte di Block Diagram viene riportata di seguito.

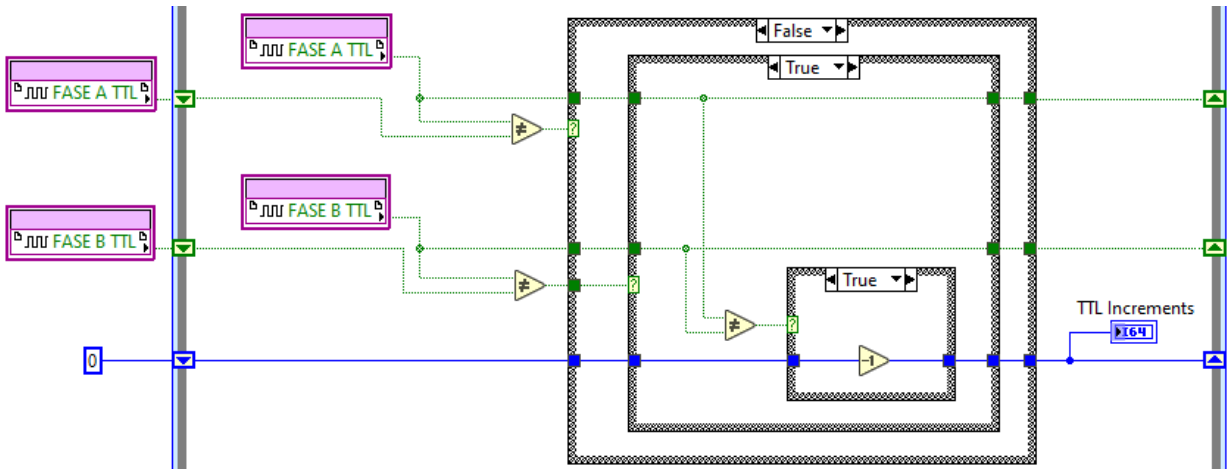

Figura 2.21: *Parte di codice per il calcolo degli incrementi compiuti dall'encoder rotativo.*

2.2.3 I/O Digitali

All'interno del Block Diagram sono stati realizzati due loop dedicati alle porte I/O digitali dei moduli del CompactRIO. In **Figura 2.22**, è riportato il loop per le porte del modulo NI 9375. Questo modulo si occupa della gestione degli I/O digitali da/per il driver del motore

Lenze. In particolare, esso contiene i controlli posizionati nella parte superiore del Front Panel, che comandano il driver (*"STOP Motor"*, *Application abilitation*, etc.), e quattro porte di input. Di queste, allo stato attuale ne sono utilizzate solamente due (*"Ready"* e *Active Error*). Gli input digitali provenienti dal modulo NI 9411, dedicato ai dispositivi di sicurezza installati sul banco, sono contenuti nel loop riportato in **Figura 2.23**.

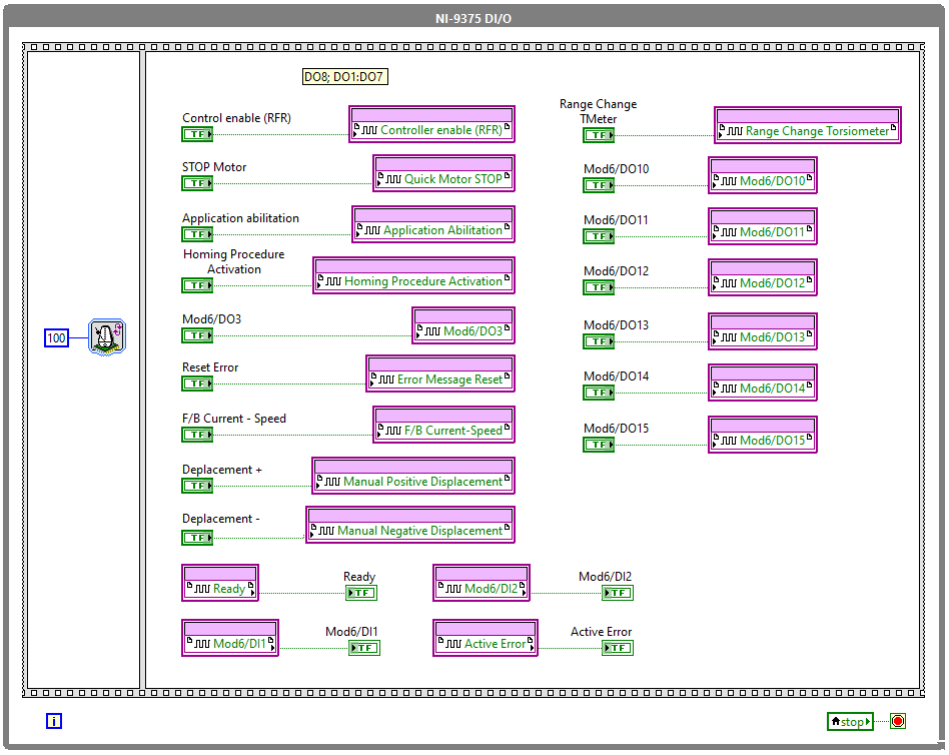

Figura 2.22: *Loop dedicato alle porte I/O digitali del modulo NI 9375.*

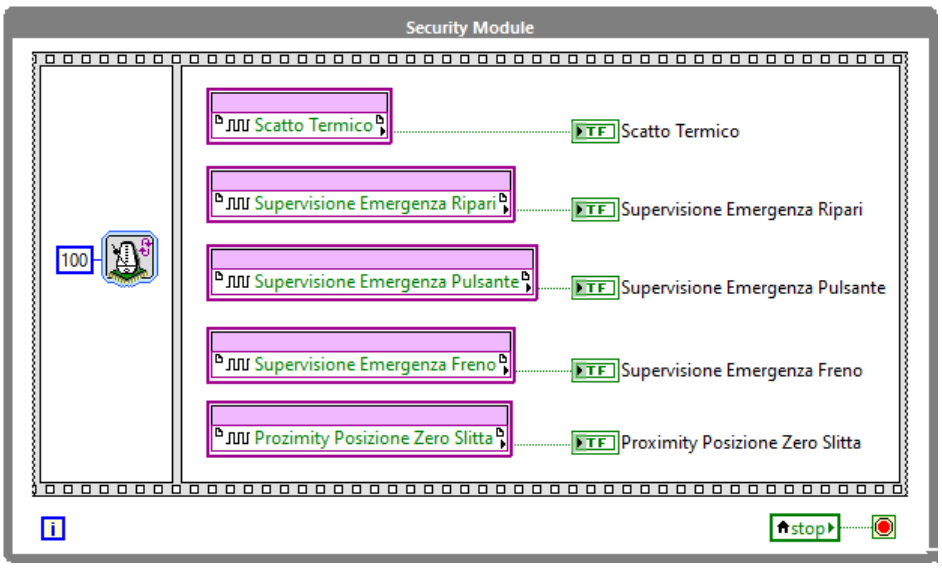

Figura 2.23: *Loop dedicato alle porte I/O digitali del modulo delle sicurezze.*

Capitolo 3

VI Real-Time

Il VI Real-Time è dedicato principalmente all'acquisizione dei dati dal VI FPGA e al loro invio al VI del PC Host. Esso si occupa, inoltre, dell'invio del segnale di SET al VI FPGA.

3.1 Project Explorer e Front Panel

All'interno del Project Explorer, il file principale del VI (*"RT_ Main.vi"*) si trova gerarchicamente al di sotto del target CompactRIO 9047. Come appare in **Figura 3.1**, la cartella contenente le subVI del RT, lo chassis del CompactRIO (dal quale si accede al VI FPGA) e la libreria delle variabili sono collocati allo stesso livello del file RT principale.

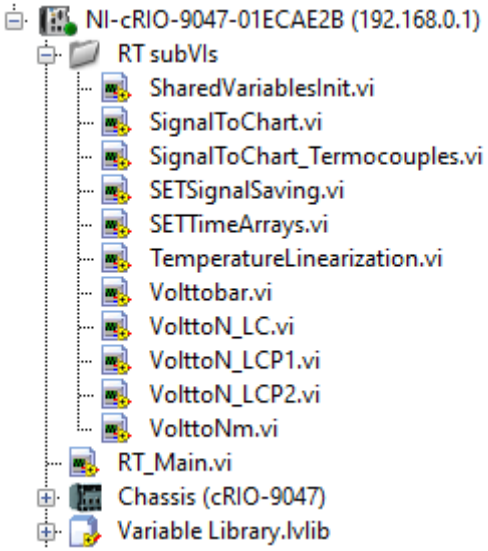

Figura 3.1: *Parte di Project Explorer dalla quale si accede al file "RT_ Main.vi" e alla cartella con i subVI.*

Il Front Panel del VI è riportato in **Figura 3.2**. Nel riquadro verde, in alto a sinistra, sono riportati i LED relativi ai Network Streams tra i VI RT e PC Host. Nello specifico, sono stati creati due canali: uno specifico per i dati provenienti dalle termocoppie e uno

per i segnali rimanenti (posizione, velocità, celle di precarico etc.); questi, rappresentando la maggioranza delle acquisizioni, sono riportati come *"MAIN"*. Per ogni stream sono presenti due LED di feedback: uno di avvenuta creazione del canale e uno di effettivo trasferimento tra le VI. La condizione affinché il LED *"Host PC Connected?"*, riquadrato in viola, sia acceso è che lo siano i due indicatori del solo canale principale.

Figura 3.2: *Front Panel del VI RT.*

Contornata in giallo, si individua la zona relativa al segnale di SET. I due LED in alto indicano se il segnale di SET è di posizione o velocità. L'indicatore *"Generation"* indica se la generazione del segnale di SET è *"Standard"* o *"Custom"*. Nel primo caso, il segnale di SET è un valore discreto fornito da uno slider presente nella VI PC; il valore fornito dal controllo è riportato nell'indicatore denominato *"Slider Value"*. Nel caso di segnale custom, il SET è una funzione, il cui andamento è fornito tramite file "*.txt" dall'utente. Quando il caricamento del file va a buon fine, si accende il LED *"SET Loaded"*; con il comando *"Generate Loaded SET"* il SET viene inviato al VI FPGA. I controlli e gli indicatori

all'interno di questo riquadro sono gestiti da variabili condivise scritte nel VI PC. Ne consegue che non è possibile comandare la VI FPGA dal VI RT, ma solo da quello del PC Host. La VI RT fa esclusivamente da collegamento tra le due.
In basso a sinistra, è posizionato il bottone rosso che blocca l'esecuzione del VI. Nella scheda *"Main Acquisition"*, sulla destra, sono riportati i grafici degli andamenti del segnale di SET, del feedback di posizione della slitta (in [mm]) e di quello della velocità angolare della vite (in [rpm]). Sebbene nell'asse delle ordinate del grafico del SET venga riportata l'ambivalente dicitura *"SET SIGNAL (Position / Angular Speed)"*, il tipo di controllo adottato è specificato dai LED sulla sinistra. Nella scheda *"Termocouples"*, vengono riportate le temperature lette dal modulo delle termocoppie. Al momento della scrittura, in attesa di sviluppi futuri, l'ultimo grafico in basso (*"Spare Termocouple"*) non riporta alcun andamento noto e/o di interesse.

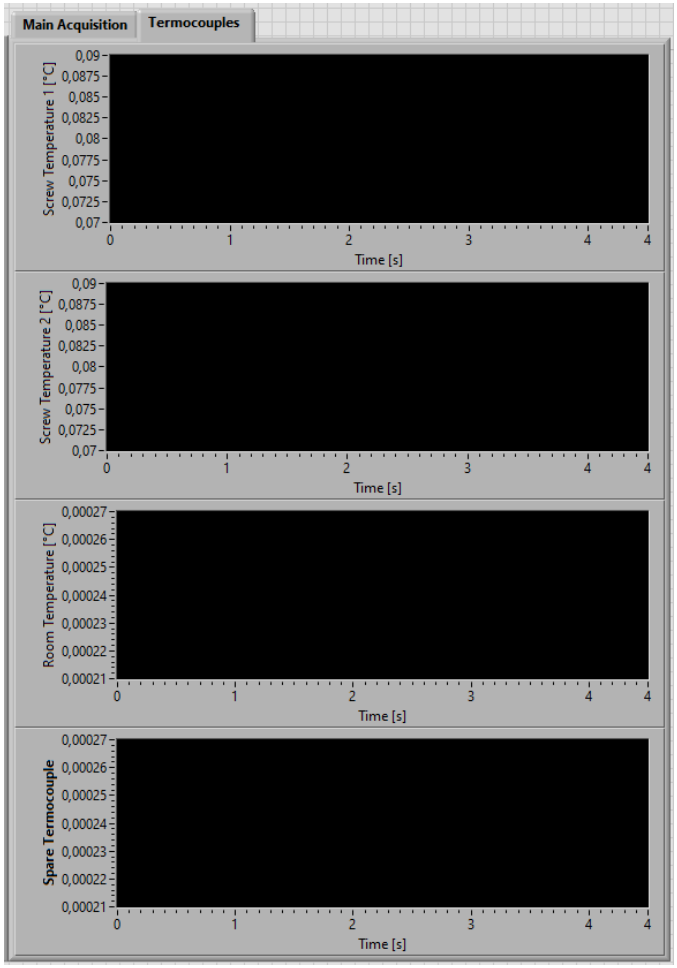

Figura 3.3: *Pagina del Front Panel contenente i grafici dei segnali provenienti dalle termocoppie.*

Nella parte destra del Front Panel, è posizionato l'indicatore *"Last Session"* (**Figura 3.4**). Esso riporta le informazioni relative all'ultima sessione di utilizzo del VI: data e ora

di avvio e di arresto e durata dell'ultima sessione (espressa in minuti). Il messaggio viene aggiornato automaticamente ad ogni avvio del VI.

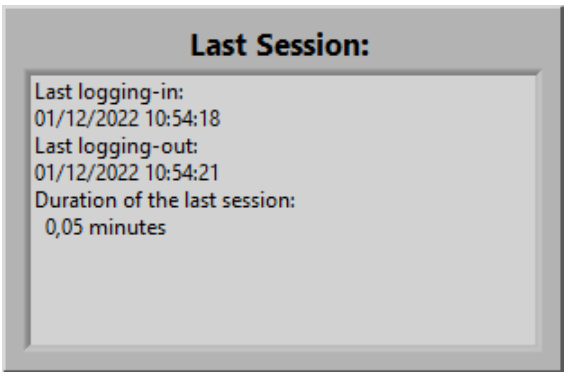

Figura 3.4: *Area del Front Panel riportante l'indicatore "Last Session".*

3.2 Block Diagram

Nei paragrafi successivi verrà descritta la parte di codice relativa al Block Diagram del VI RT. L'ordine di trattazione delle diverse parti cercherà di seguire il più possibile quello imposto dal flusso dei dati.
A differenza di quanto avviene nel VI FPGA, nel VI Real-Time sono permessi il trasporto dell'errore e l'utilizzo delle funzioni di *Error Handling*. I fili che trasportano i messaggi di errore sono quelli di colore giallo. In ingresso alle varie strutture nel Block Diagram, verrà alle volte collegata una costante *"No Error"*; questa permette un avvio delle strutture parallelo a quello dei loop che seguono il flusso dei dati (e.g. i loop di lettura dei dati dai FIFO), da cui risultano quindi svincolati, e l'imposizione di un errore nullo al loro ingresso, permettendo di circoscrivere più efficacemente gli errori nelle fasi di debugging.

3.2.1 Inizializzazione del VI

Nella **Figura 3.5**, è riportata la *Flat Sequence* che si occupa di inizializzare l'intero strumento virtuale.

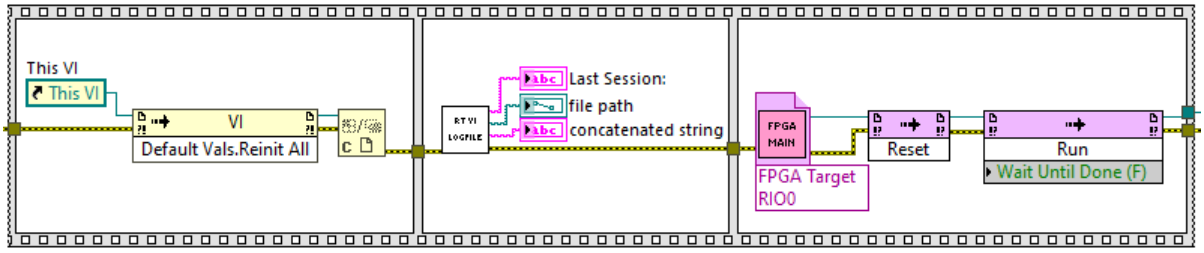

Figura 3.5: *Parte dedita all'inizializzazione del VI RT.*

Nel primo frame, è presente il blocco *Invoke Node*; esso permette il richiamo dei *metodi*[1] disponibili per la classe dell'oggetto, il VI corrente, il cui riferimento costituisce l'input al blocco (filo verde). Con questo blocco viene invocato il metodo *"Default Values: Reinitialize All To Default"*, il quale riporta tutti i controlli nel Front Panel al loro valore di default. Il blocco successivo interrompe il riferimento al VI.
Nella seconda sequenza è riportato il subVI che si occupa dell'aggiornamento dell'indicatore "Last Session". Dal momento che questa funzionalità si articola su due subVI differenti, le modalità di realizzazione saranno oggetto di un apposito paragrafo.
Nel terzo frame, con modalità analoghe a quelle del primo, viene aperto un riferimento al VI FPGA, ne viene bloccata l'eventuale esecuzione e ne vengono resettati controlli e indicatori (tramite il metodo *Reset*); infine, ne viene avviata l'esecuzione (metodo *Run*). L'avvio del solo VI RT implica anche quello del VI FPGA. Ai fini della sola acquisizione dati e imposizione del SET, l'avvio manuale di quest'ultimo è facoltativo. Il riferimento al VI FPGA non viene interrotto perché andrà in ingresso ai blocchi di Invoke Method che permettono la lettura dei dati dai FIFO dell'acquisizione principale e delle termocoppie.

3.2.2 Conversione e invio dei segnali acquisiti

Nella **Figura 3.6**, sono riportati i due While Loop che si occupano della conversione e dell'invio dei segnali provenienti dal FIFO dell'acquisizione principale. Tramite la funzione *"Wait Until Next ms Multiple"*, viene impostata la frequenza a 1 kHz, in linea con il loop di scrittura del FIFO nel VI FPGA. Il flusso di dati tra i due cicli è permesso da un FIFO RT, inizializzato dal blocco funzione *"RT FIFO Create"* posizionato al loro esterno ed evidenziato dal riquadro in arancione. Si ricorda che la configurazione dei FIFO RT all'interno del Block Diagram è analoga a quella delle code.
Considerando il primo loop, evidenziato dal riquadro in rosso si trova il blocco che invoca il metodo *"Read"* del FIFO. Esso riceve in ingresso il riferimento al VI FPGA, proveniente dalla sequenza di inizializzazione del VI. Dal FIFO, viene letto un array di nove elementi per ogni ciclo ci clock, lo stesso numero scritto per ogni ciclo nel VI FPGA. In uscita dal blocco, il filo che trasporta i dati letti si biforca, collegandosi al subVI per la conversione dei segnali (riquadrato in azzurro) e al blocco *"RT FIFO Write"*, per l'invio degli stessi al Network Stream con il VI PC.

[1]Nella programmazione orientata agli oggetti, l'elemento principale è costituito dalla *Classe*. Un *Oggetto* costituisce una *Istanza* della classe (ovvero, una sua rappresentazione). Una classe è costituita da *Attributi* (campi che specificano le caratteristiche o proprietà che tutti gli oggetti della classe devono avere, i cui valori in un certo istante determinano lo stato del singolo oggetto della classe) e da *Metodi* (funzioni che specificano le azioni o i comportamenti ammissibili che un oggetto della classe è in grado di compiere). I metodi costituiscono l'*Interfaccia* della classe, nonché gli unici strumenti grazie ai quali è possibile interagire con i suoi oggetti.

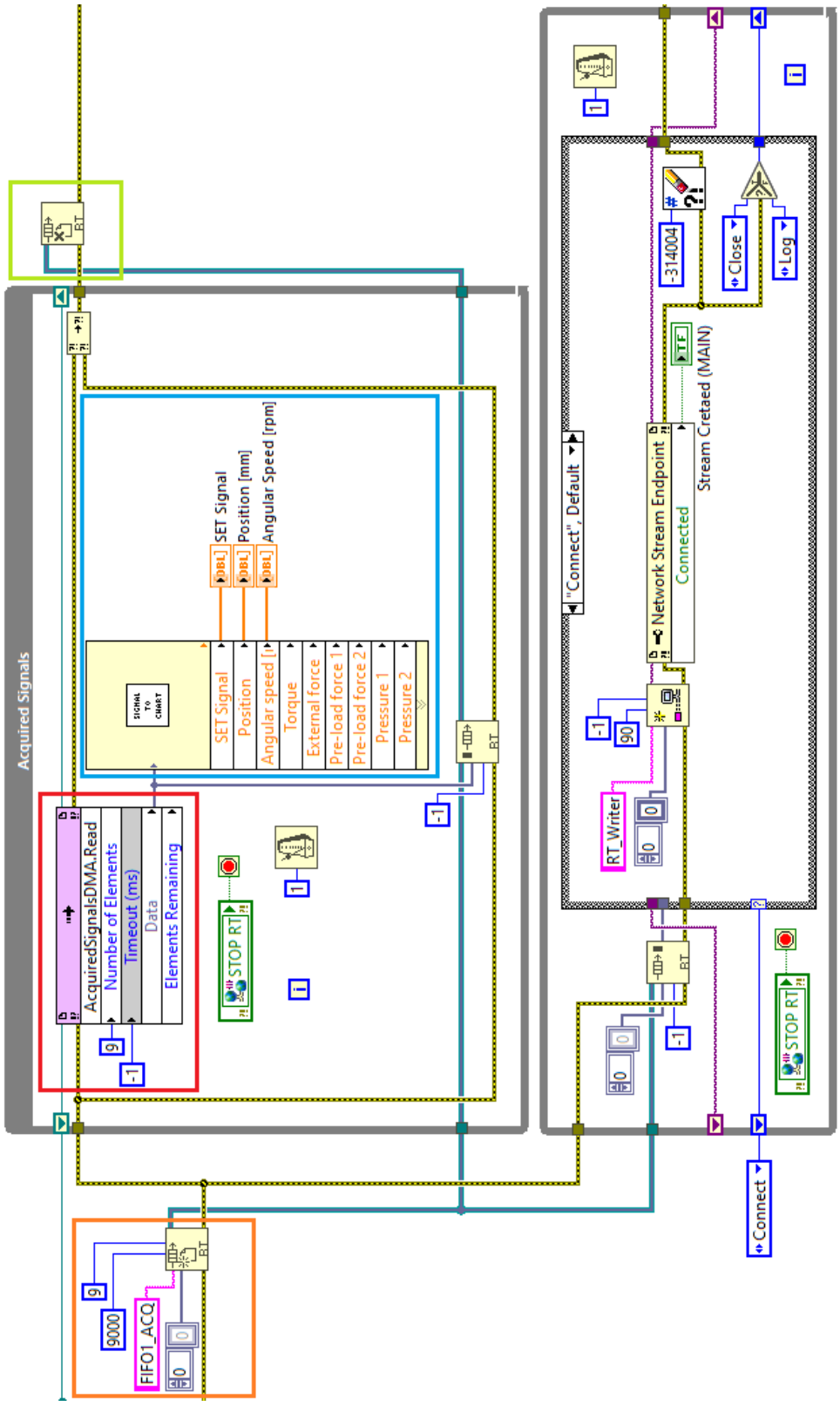

Figura 3.6: *Loop per la lettura dal FIFO, conversione e invio al VI PC dei dati provenienti dall'acquisizione principale.*

Ricordando che i segnali in uscita dal FIFO sono i valori delle tensioni lette dai moduli di acquisizione (eccetto i valori di posizione e velocità angolare) il blocco di conversione dei segnali si occupa di trasformarli nelle corrispettive grandezze fisiche, usufruibili dall'utente. Il Block Diagram di questo subVI, denominato *"Signal To Chart.vi"*, è riportato in **Figura 3.7**. L'array contenente i segnali viene ivi convertito in *DBL* (*"Double-precision floating-point"*) e, tramite la funzione *"Decimate 1D Array"*, viene scomposto nei suoi elementi, ciascuno dei quali attraversa un opportuno subVI dove se ne realizza l'effettiva conversione.

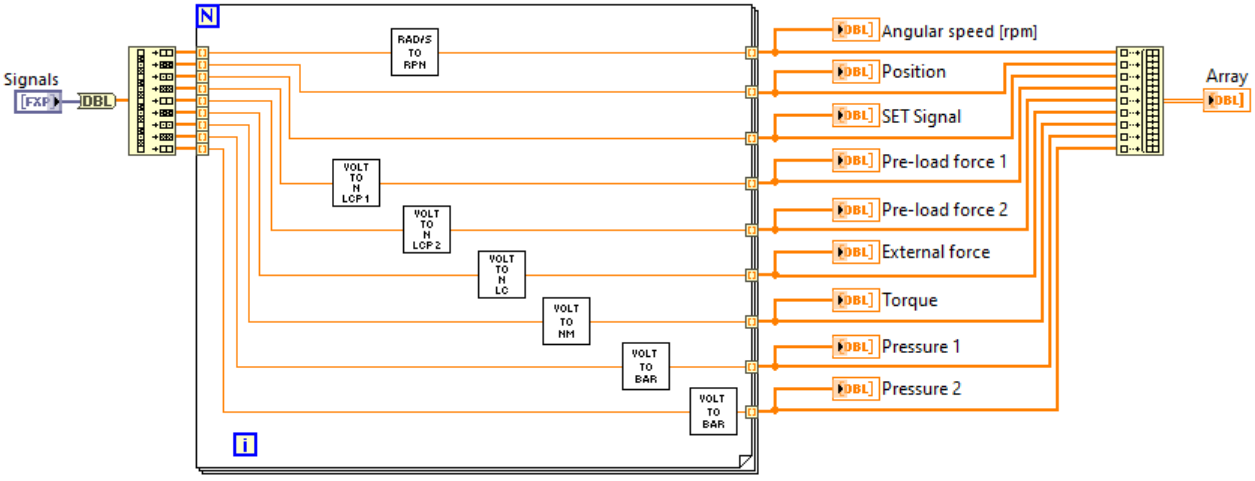

Figura 3.7: *Block Diagram del subVI atto alla conversione delle tensioni in grandezze fisiche.*

Il Block Diagram-tipo dei subVI di conversione è riportato in **Figura 3.8**: il valore di tensione in ingresso viene moltiplicato per la costante di conversione, per poi essere mandato in output. La costante di conversione è specifica per ogni sensore.

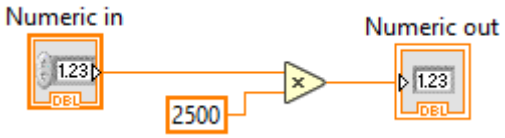

Figura 3.8: *Block Diagram blocco di conversione del segnale proveniente dalla cella di carico in un valore di forza.*

Uscito dalla subVI, il singolo segnale convertito viene raccolto insieme agli altri in un nuovo array (funzione *"Build Array"*), poi inviato in uscita dal subVI. Al fine di riportare le grandezze su dei grafici, per evitare di dover scomporre nuovamente l'array al di fuori del subVI, i segnali convertiti vengono mandati in output anche singolarmente. Questi costituiscono i dati in ingresso ai grafici della scheda "Main Acquisition", visibili nella **Figura 3.2**.
Facendo sempre riferimento alla **Figura 3.6**, in uscita dal Timed Loop e riquadrati in verde, si ha la funzione per l'eliminazione del riferimento al FIFO RT. Questa interviene una volta arrestato il ciclo.

Al di sotto del Timed Loop appena descritto, è posizionato quello realativo al Network Stream tra il VI RT e il VI PC Host. In input al ciclo si ha il riferimento al FIFO RT e, immediatamente al suo interno, la funzione *"RT FIFO Read"*; come suggerisce lo stesso nome, questo blocco si occupa di leggere i dati immessi nel FIFO nel loop descritto precedente.
I dati letti dal FIFO costituiscono uno degli input di una *Case Structure*, al cui interno sono presenti le funzioni del Network Stream. Questa si compone di tre *Casi*, selezionati tramite una costante di tipo *"enum"*[2] e riportati singolarmente in **Figura 3.9**.

- **Connect** - Rappresenta la schermata di default. Crea il collegamento tra i due VI.
- **Log** - Contiene le funzioni per il trasferimento dei dati.
- **Close** - Effettua il flush del canale e, infine, elimina l'endpoint.

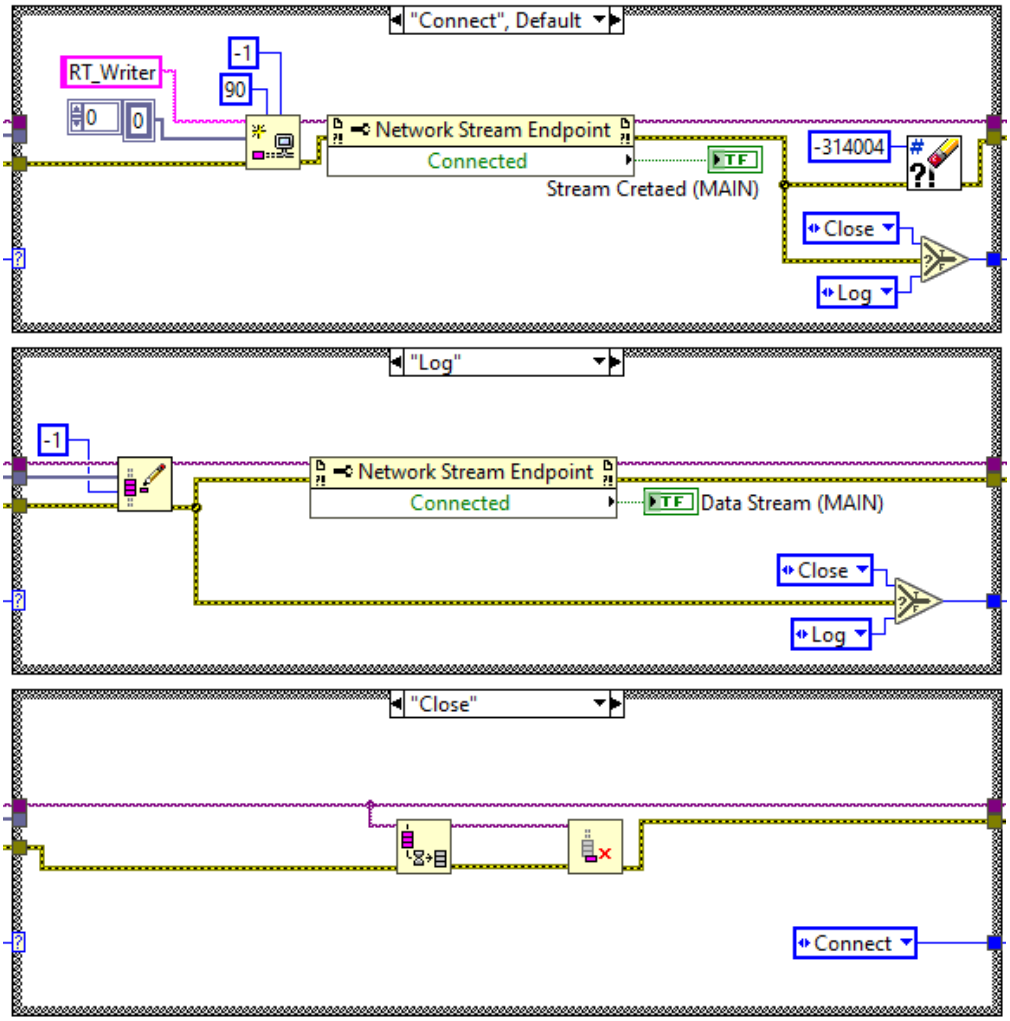

Figura 3.9: *Case Strucrure dedicata al Network Stream.*

[2]Un dato di tipo *"enumerated"* (abbreviato *"enum"*) è una lista di etichette di stringhe che vengono associate a dei numeri interi. All'interno di LabVIEW, viene trattato come un numero intero privo di segno.

Analizzando la **Figura 3.6** e la **Figura 3.9**, è possibile comprendere come viene realizzato il passaggio da un caso all'altro. A ogni ciclo di clock, la costante enum con il nome del caso voluto viene rimandata in ingresso al ciclo tramite uno *Shift Register*. La costante posta in ingresso al loop, riportante il caso *"Connect"*, vale solo per il primo ciclo e corrisponde al caso iniziale imposto alla Case Structure (esso è comunque settato come caso di default dalle opzioni della struttura stessa). Riferendosi ora alla sola **Figura 3.9**, per i primi due casi è stato posto un blocco *"if"* connesso al filo dell'errore. La presenza di un errore soddisferebbe la condizione *True*, portando al caso "Close" nel ciclo di clock successivo; in caso contrario, si verrebbe rimandati al caso di "Log" e lo stream tra i due VI procederebbe regolarmente. In ciascuna delle schermate Connect e Log, è inserito un *"Property Node"*[3] a cui è collegato uno dei LED di feedback posti nel Front Panel, riquadrati in verde nella **Figura 3.2**.

Per i dati provenienti dal modulo delle termocoppie, si mantiene la stessa esatta struttura. Sono presenti i due While Loop, interconnessi da un FIFO RT, nei quali vengono acquisiti i dati dal FIFO; allo stesso modo essi sono inviati al VI PC tramite Network Stream. L'unica differenza sostanziale sta nel subVI per la conversione delle tensioni in valori di temperatura. Infatti, allo stato attuale del lavoro, non si ha a disposizione un modello atto a tale scopo: il subVI si limita a inviare in output alcuni degli stessi valori di tensione che riceve in ingresso. Questa scelta è stata fatta con l'idea sì di impostare la struttura per la scomposizione e ricomposizione dell'array, ma lasciando a sviluppi futuri la realizzazione del modello per la conversione. Il Block Diagram del subVI viene riportato di seguito:

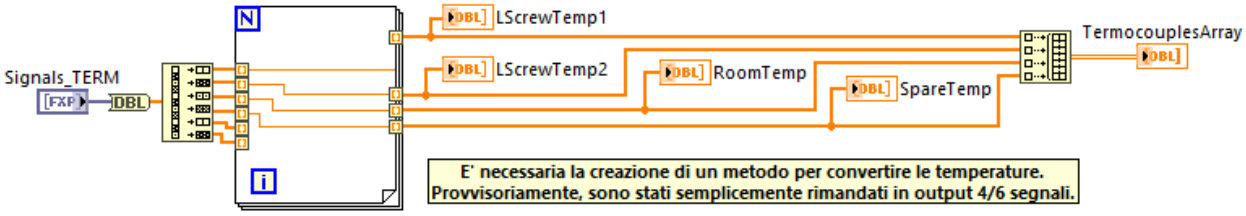

Figura 3.10: *Block Diagram del subVI che verrà dedicato alla conversione dei segnali di tensione provenienti dal modulo delle termocoppie.*

Come si nota dalla costante collegata al timer, la frequenza del ciclo è impostata a un valore inferiore rispetto a quello dell'acquisizione principale. Come visto per il VI FPGA, questo serve per rimanere nel range di frequenza di sample massima ammissibile dal modulo delle termocoppie.
Il loop relativo al Network Stream è lo stesso di quello visto per l'acquisizione principale.

[3]Il *Property Node* fornisce (legge) o impone (scrive) le proprietà di un elemento, il cui riferimento è posto in ingresso al blocco; esso viene quindi usato per la gestione di proprietà e metodi di istanze di applicazioni, VI e oggetti. Il blocco Property Node si adatta automaticamente al riferimento a cui viene collegato.

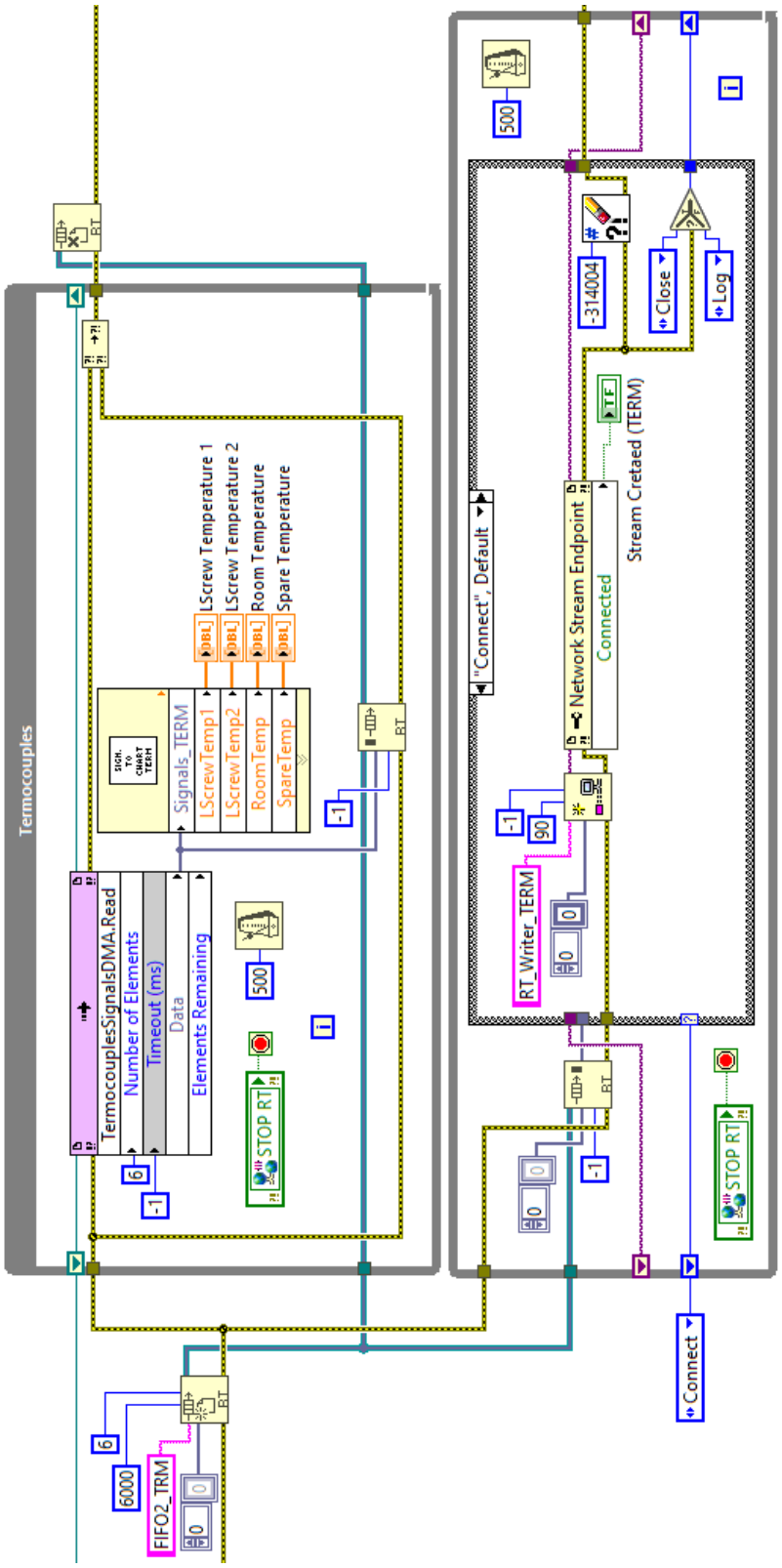

Figura 3.11: *Loop per la lettura dal FIFO e invio al VI PC dei dati ottenuti durante l'acquisizione dal modulo delle termocoppie.*

3.2.3 Attivazione dei controlli nel Front Panel

In **Figura 3.12** è riportato il loop che serve all'attivazione dei comandi posti nel Front Panel. Come input viene imposta una costante "No Error", non riportata in figura. Di default, esso contiene una Case Structure e due Property Node, aventi in output lo stato dei due LED di feedback del Network Stream principale. La struttura riporta due soli casi: *"True"* e *"False"*.

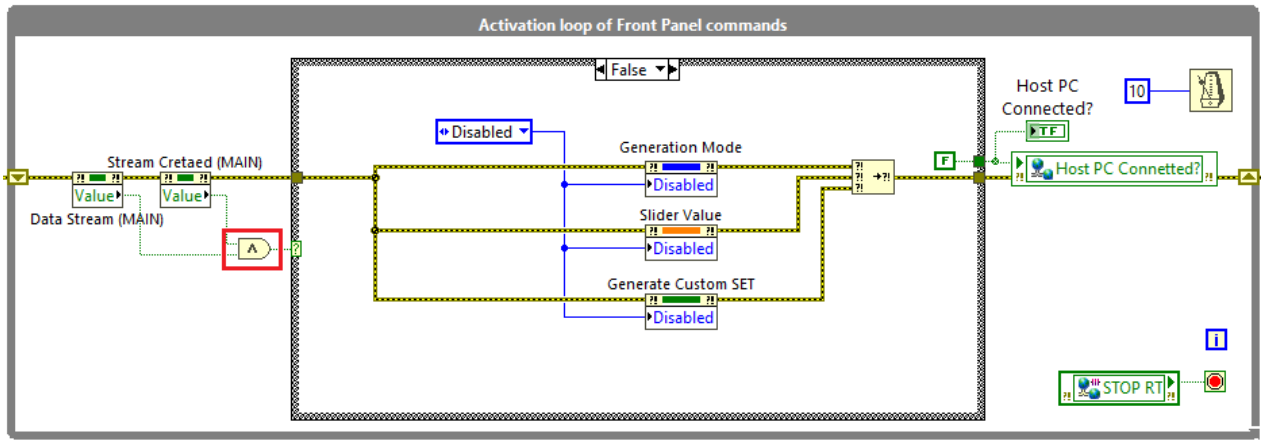

Figura 3.12: *Loop per l'attivazione dei comandi del Front Panel.*

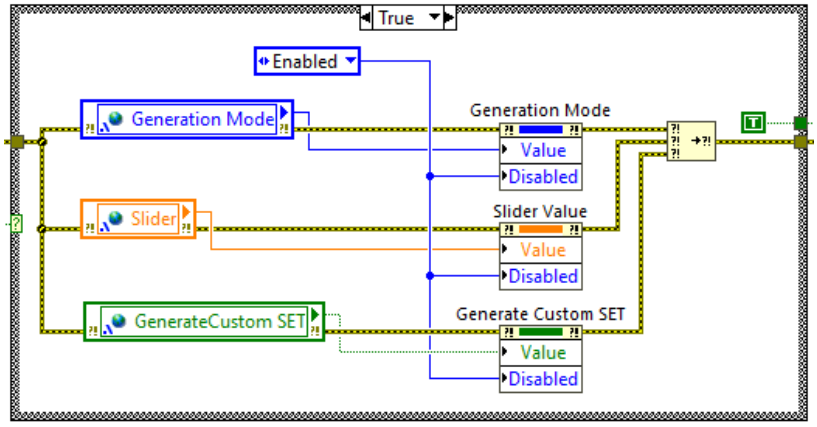

Figura 3.13: *Caso "True" della Case Structure all'interno del loop di attivazione dei comandi.*

Lo switch da uno all'altro viene gestito dall'operatore booleano AND (riquadrato in rosso nella **Figura 3.12**). Quando lo stream principale viene creato e avviene il trasferimento dei dati, entrambi i LED sono accesi, soddisfacendo la condizione dell'operatore AND. In tal caso, si passa al caso *True* della struttura, riportato in **Figura 3.13**. Qui, i comandi vengono attivati e viene loro conferito il valore dalle variabili condivise scritte all'interno del VI PC. Inoltre, viene acceso il LED *"Host PC Connected?"*, dando così all'utente la conferma di avvenuta connessione con il VI PC. Quando la condizione è falsa, tutti i comandi vengono disattivati. L'attivazione/disattivazione dei comandi e la scrittura del loro valore è realizzata con il solo ausilio dei Property Node.

3.2.4 Arresto del VI e impostazione del SET

Il loop riportato in **Figura 3.14** svolge due funzioni fondamentali: permette la procedura di arresto del VI e di impostare il tipo di SET desiderato (di posizione o velocità) nel VI FPGA. In ingresso e in uscita dal ciclo si hanno, col filo azzurro ma non riportati in figura, rispettivamente il blocco di riferimento al VI FPGA e quello della sua eliminazione.

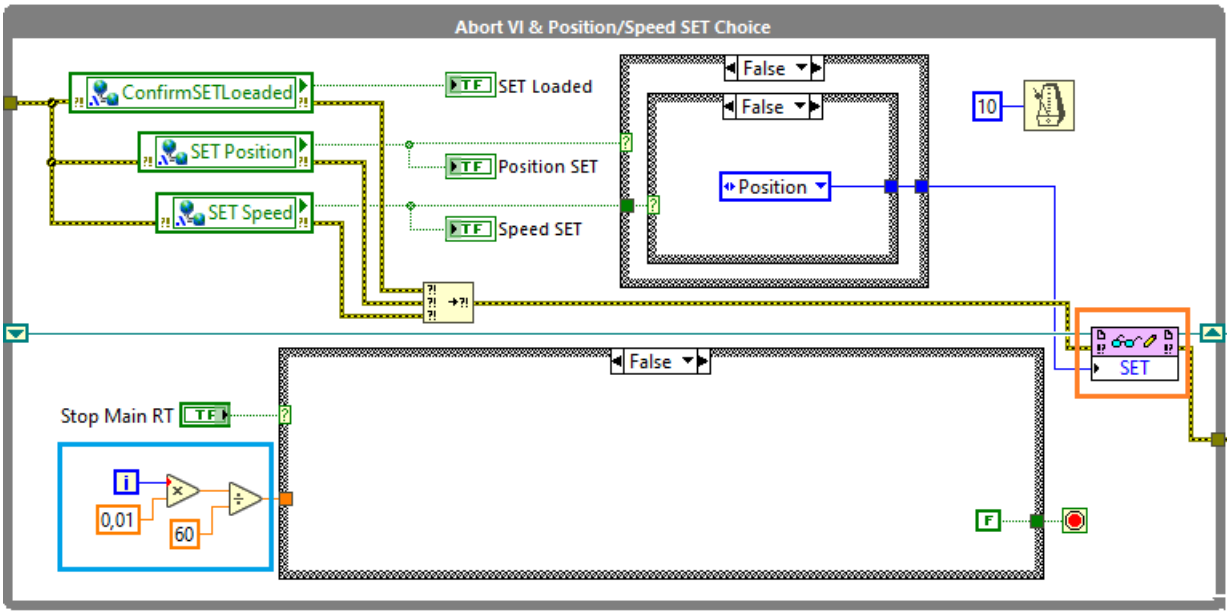

Figura 3.14: *Loop per l'arresto della VI e per l'impostazione del tipo di SET.*

L'operazione di arresto del VI viene realizzata tramite una Case Structure contenente due casi Vero/Falso. Lo switch tra i due avviene tramite un controllo di tipo booleano, denominato *"Stop Main RT"*, nonché il pulsante rosso visto nel Front Panel (**Figura 3.2**). Il caso "False" presenta una scheda completamente vuota, mentre quella del caso "True" contiene una Flat Sequence Structure (**Figura 3.15**).

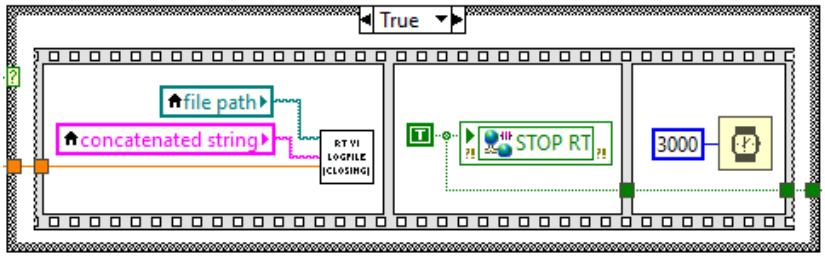

Figura 3.15: *Caso "True" della Case Structure per l'arresto del VI RT.*

Il primo frame contiene il secondo subVI per l'aggiornamento dell'indicatore "Last Session". Il secondo contiene la variabile condivisa *"STOP RT"*, alla quale viene conferito il valore "True"; in questo modo viene dato il comando di arresto a tutti i loop presenti nel VI. Il valore della costante booleana viene inviato in uscita dalla Case Structure, dove si collega

al blocco di arresto del ciclo. La funzione *"Wait (ms)"*, contenuta nella terza sequenza e impostata a 3 s, conferisce il tempo necessario a tutti i cicli di arrestarsi. All'uscita del loop in esame, sono presenti i blocchi per l'arresto del VI FPGA (si ricorda che il VI RT ne avvia automaticamente l'esecuzione) e dell'eliminazione del suo riferimento; questi sono riportati in **Figura 3.16**. Nel caso in cui il riferimento all'FPGA venisse eliminato mentre un loop lo sta ancora usando (e.g. uno dei loop contenenti il FIFO non si è ancora arrestato) si incorrerebbe nell'errore *"-63195"*, non risolvibile attraverso i blocchi di error handling e che causa un arresto critico del programma. É quindi fondamentale che questo loop sia l'ultimo ad arrestarsi.

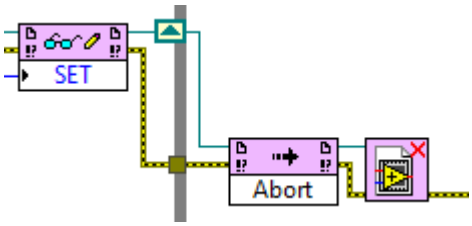

Figura 3.16: *Blocchi per l'arresto del VI FPGA e l'eliminazione del suo riferimento.*

In ingresso alla Case Structure, viene inviato il tempo di esecuzione del VI espresso in minuti. Esso viene calcolato moltiplicando il contatore del ciclo While per 0,01 s (10 ms, nonché l'unità di tempo imposta dal timer del ciclo) e dividendo il risultato per 60 s/min. Il minutaggio viene poi inviato al blocco dedicato all'indicatore "Last Session". Queste operazioni sono evidenziate in azzurro nella **Figura 3.14**.

L'impostazione del tipo di SET viene effettuata tramite Shared Variable scritte nel VI PC. Più in dettaglio, all'interno del loop vengono lette tre variabili di tipo booleano:

- *"ConfirmSETLoaded"* - Segnale di conferma dell'effettivo upload del segnale di SET di tipo "Custom";
- *"SETPosition"* - Set di posizione;
- *"SETSpeed"* - Set di velocità.

A esse sono collegati i corrispettivi indicatori LED, visibili nel Front Panel. Il controllo *"Read/Write"*, riquadrato in arancione nella **Figura 3.14**, si occupa di impostare il SET nel VI FPGA (del quale ne riceve in input il riferimento). Una costante di tipo "enum" costituisce il secondo ingresso alla funzione. Essa contiene due elementi, *"Position"* e *"Speed"*, che vengono gestiti da due Case Structure annidate. É opportuno evidenziare come, nel caso in cui le variabili booleane del SET siano entrambe false (nessun tipo di SET sia stato selezionato dall'utente), la Case Structure impone comunque il SET come di posizione; ne rappresenta, quindi, il valore di default. Questa scelta è stata una semplificazione durante la realizzazione del VI FPGA: l'aggiunta alla costante "enum" di un terzo elemento "neutro"

da gestire avrebbe complicato inutilmente la realizzazione dei VI. In ogni caso, per il corretto funzionamento dei VI RT e PC, è sempre richiesto all'utente di scegliere il SET desiderato. Come si vedrà nel paragrafo successivo, il programma è realizzato in modo tale che il mancato soddisfacimento di una delle due condizioni non permetta "lo sblocco" delle operazioni sul segnale di SET, sia esso "Standard" o "Custom".

3.2.5 Generazione del SET

Il ciclo While nel quale viene generato il segnale di SET si articola in diverse Flat Sequence e loop annidati. Essi occupano una sezione rilevante del Block Diagram, in maniera tale renderne impossibile l'impaginazione. Di seguito, ciascuna parte di codice verrà descritta singolarmente. Per la visione del complessivo, si suggerisce l'uso di LabVIEW.

Nonostante la struttura estremamente articolata, la presenza della variabile "STOP RT" in tutti i loop permette l'arresto del ciclo principale durante qualsiasi della generazione.

Gerarchicamente al di sotto del loop principale, è presente una Flat Sequence formata da due sequenze. Nella prima, è presente il loop riportato in **Figura 3.17**. Questo ciclo può essere definito come "di attesa", dal momento che non si arresta fino a che il Real-Time non stabilisce una connessione con il VI PC e l'utente non selezioni il tipo di controllo.

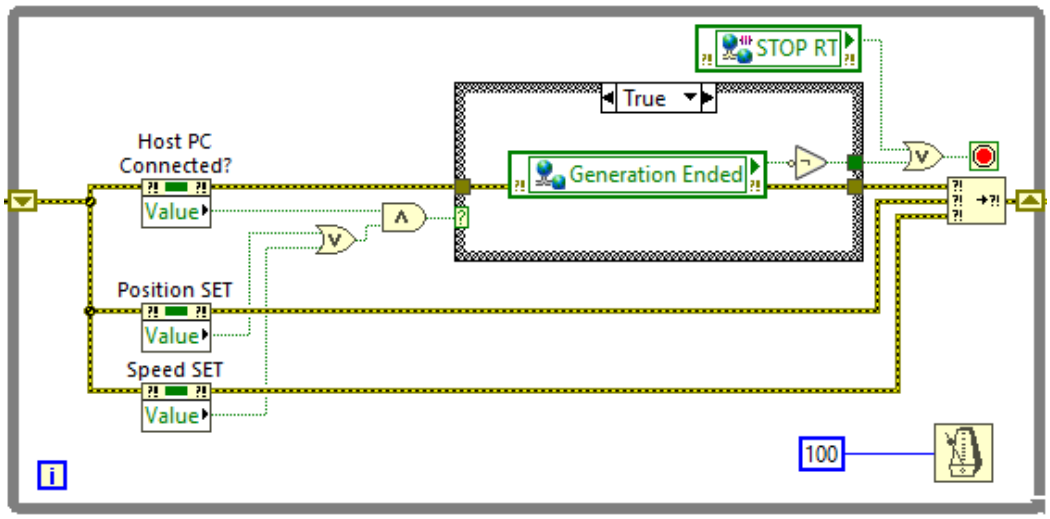

Figura 3.17: *Loop contenuto nel primo frame della sequenza di generazione del SET.*

L'indicatore "Host PC Connected?" è collegato alla funzione AND, la cui uscita è connessa a una Case Structure. Quando viene selezionato uno dei due SET (operatore OR), l'uscita dell'AND assume valore 1. La struttura passa la caso "True", dove viene letto il valore della variabile booleana *"Generation Ended"*. Quando è falsa, il programma è in grado di operare una nuova generazione del segnale di SET. Come si vedrà anche in seguito, alla fine di ogni generazione il suo valore viene portato a *True*, dando all'utente un avviso sul termine della

procedura. Nel ciclo *"Control to RT VI"* del VI PC, il valore viene riportato a *False*, permettendo un ripristino delle funzionalità del generatore di segnale. In tal caso, riferendosi nuovamente alla **Figura 3.17**, il segnale in uscita dal blocco della Shared Variable viene negato: il ciclo si arresta e si passa al frame successivo della sequenza.

Nel frame successivo della sequenza, si ha un loop con all'interno una Case Structure; essa è controllata dal comando "enum", che riporta i due casi "Standard" e "Custom".

SET Standard Il caso "Standard" contiene un Timed Loop, settato a una frequenza di 1 kHz. In ingresso, esso riceve il riferimento al VI FPGA che si collega al blocco per la scrittura del SET sul FIFO dedicato. Il dato che viene scritto è il valore discreto dello slider comandato dall'utente. Il valore corrente della variabile "enum" viene confrontato con quello iniziale: in caso si dovesse passare al caso "Custom", il ciclo si arresterebbe in automatico, per permettere lo switch della struttura.

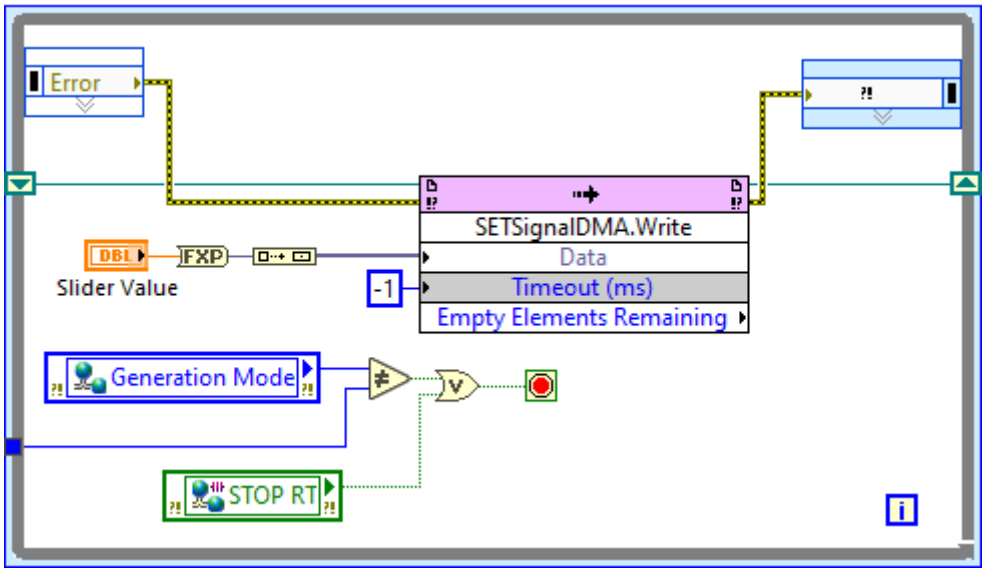

Figura 3.18: *Timed Loop per l'invio del segnale di SET "Standard".*

SET Custom Il caso "Custom" contiene a sua volta una Flat Sequence con quattro frame. I primi due sono riportati in **Figura 3.19**. Il primo di essi contiene un altro loop di attesa: l'arresto del ciclo e il passaggio alla sequenza successiva avvengono quando viene caricato il SET dal file, attivando il relativo indicatore. Nel frame successivo, viene letta la variabile contenente del cluster col segnale di SET, il quale viene scomposto dal blocco dal *"Unboundle By Name"*.
Il terzo frame contiene una Case Structure, il cui caso "True" si attiva col comando *"Generate Custom SET"*. Al suo interno, analogamente al caso "Standard", si ha una Flat Sequence contenente un Timed Loop con il blocco per la scrittura del SET sul FIFO (**Figura 3.20**).

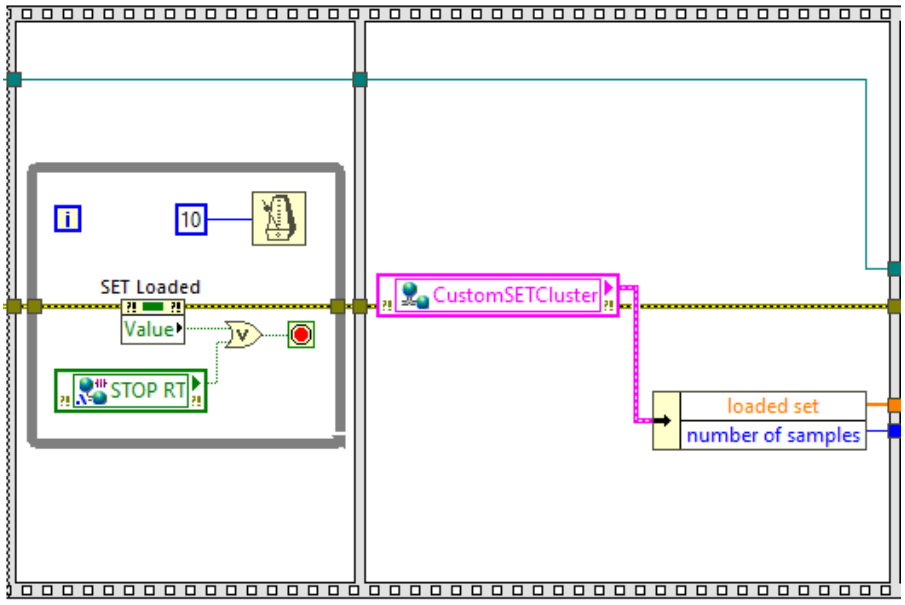

Figura 3.19: *Primi due frame della sequenza all'interno del caso "Custom".*

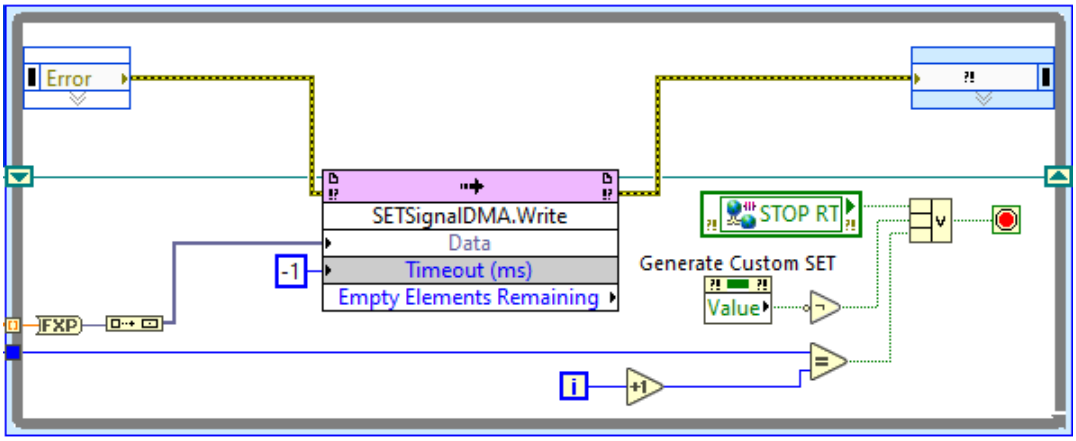

Figura 3.20: *Loop per l'invio del SET "Custom" al VI FPGA.*

Il loop si arresta automaticamente, quando vengono scritti tutti gli elementi del segnale, oppure manualmente, disattivando il comando "Generate Custom SET" (dal momento l'azione meccanica del comando è impostata a "Switch when pressed") o premento sul pulsante di stop. Nel frame successivo, la variabile *"Generation Ended"* viene portata a *True*; ci si ricollega al discorso del ripristino delle funzionalità del generatore di segnale. A seguito di diversi test, è comunque preferibile riavviare il VI a ogni nuova generazione che si desidera effettuare.

Che venga usata la modalità "Standard" o la "Custom", a ogni ciclo del loop principale viene letto il valore della variabile "Generation Mode" e riportato in ingresso con uno Shift Register. In questo modo, l'utente può passare da una modalità all'altra senza il bisogno di arrestare il VI.

3.2.6 File di Log - Ultima Sessione

Come già accennato in precedenza, l'indicatore "Last Session" funziona grazie all'impiego congiunto di due subVI. Esse si occupano delle operazioni di lettura/scrittura di un file di testo (in formato "*.txt"), salvato nella memoria interna del CompactRIO e nel quale sono contenute le informazioni riguardo l'ultima sessione in cui è stato usato il VI Real-Time.

Il primo subVI, denominato *"RTVI_logfile.vi"*, è quello contenuto nel secondo frame della sequenza di inizializzazione del VI Real-Time, riportata in **Figura 3.5**. Il suo scopo è quello di leggere il file di testo, visualizzarne a schermo il contenuto e registrare la data e l'ora di avvio del VI in una stringa. Collegati agli output del blocco, si hanno gli indicatori "Last Session", *"file path"* e *"concatenated string"*. Questi ultimi due sono in modalità "Hide Indicator", per cui non sono visibili sul Front Panel. Infatti, essi servono solamente per essere associati alle variabili locali per il trasferimento dei dati dal primo al secondo subVI.

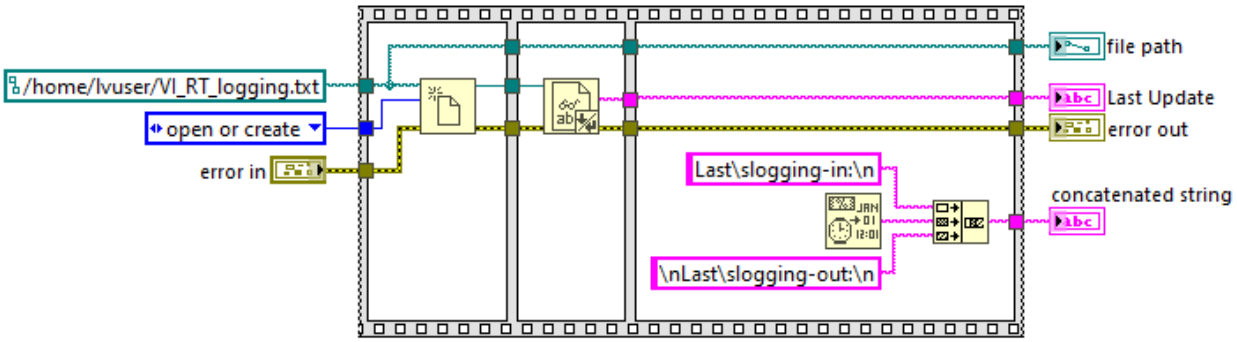

Figura 3.21: *Block Diagram del subVI ""RTVI_logfile.vi"".*

Il Block Diagram del subVI è riportato in **Figura 3.21**. La sequenza al suo interno si compone di tre frame. Nel primo si accede al file di testo. L'opzione *"open or create"* permette di creare il file, se questo non viene trovato nel percorso desiderato. Esso deve essere specificato, tramite una costante collegata all'ingresso del blocco, in una forma che varia in base al sistema operativo installato sul CompactRIO; in questo caso, si tratta di *"NI Linux Real-Time OS"*. Nella seconda sequenza viene letto il testo presente all'interno del file e viene inviato all'indicatore "Last Session". Nel terzo frame viene assemblata la stringa riportata di seguito:

> *"Last logging-in:*
> **data e ora di avvio**
> *Last logging-out:"*

Essa viene inviata all'indicatore "concatenated string", per la successiva lettura tramite variabile locale.
Il secondo subVI (*"RTVI_logfile_closing.vi"*) è quello visto nella sequenza di **Figura 3.15**. Esso agisce a seguito del comando di arresto del VI Real-Time. Al suo ingresso, si trovano

le sopracitate variabili locali ("file path" e "concatenated string") e la durata di esecuzione del VI Real-Time espressa in minuti. Il suo Block Diagram è riportato in **Figura 3.22**.

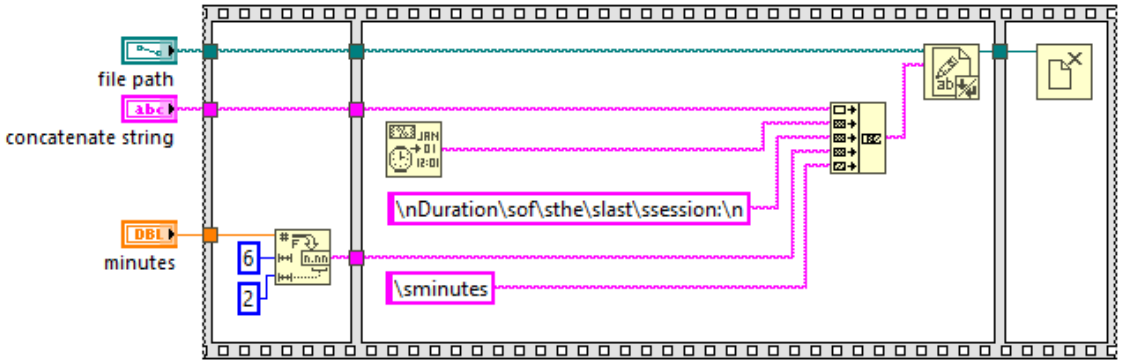

Figura 3.22: *Block Diagram del subVI ""RTVI_logfile_closing.vi"".*

Nel primo frame, il dato "DBL" dei minuti viene convertito in stringa; questa avrà una grandezza massima di sei cifre significative, di cui due decimali. La stringa "concatenated string" viene unita alla restante parte di testo, ottenendo

"Last logging-in:
data e ora di avvio
Last logging-out:"
data e ora di arresto
Duration of the last session:
**numero di minuti convertito in stringa* minutes*

Questa stringa viene scritta sul file di testo, il quale viene infine chiuso nell'ultimo frame della sequanza.

Capitolo 4

VI PC Host

Tra quelli realizzati, il VI PC è senza dubbio lo strumento virtuale più complesso. Il suo scopo principale è quello di fare da interfaccia tra l'operatore e il dispositivo di acquisizione e controllo. Da questo VI, l'utente comanda indirettamente i VI RT e FPGA.

4.1 Project Eplorer e Front Panel

Nel Project Explorer, il VI *"PC_ Main.vi"* si trova immediatamente al di sotto dell'oggetto *"MyComputer"*. Allo stesso livello, si trova la cartella contenente tutti i subVI utilizzati all'interno del principale.

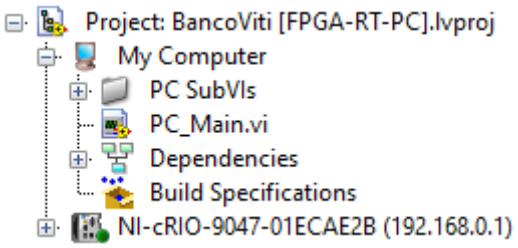

Figura 4.1: *Sezione di Project Explorer contenete il file "PC_ Main.vi" e la cartella dei subVI.*

4.1.1 Front Panel

Il Front Panel del VI PC si compone di diverse sezioni. Alcune di esse sono dedicate all'impostazione del SET, altre alla visualizzazione dei dati acquisiti. Per una migliore ottimizzazione degli spazi, sulla sinistra del pannello è stato posizionato un *Tab Control* contenente quattro schede. Una di esse, denominata *"Error"*, viene usata per la visualizzazione di eventuali messaggi di errore riscontrabili durante il funzionamento del VI.
Nella parte bassa del pannello, è presente una fascia grigia che contiene alcuni indicatori LED e, sulla destra, il pulsante di arresto del VI.

Impostazione del SET

All'interno del Tab Control è presente la scheda *"SET Choice"*, riportata in **Figura 4.2**.

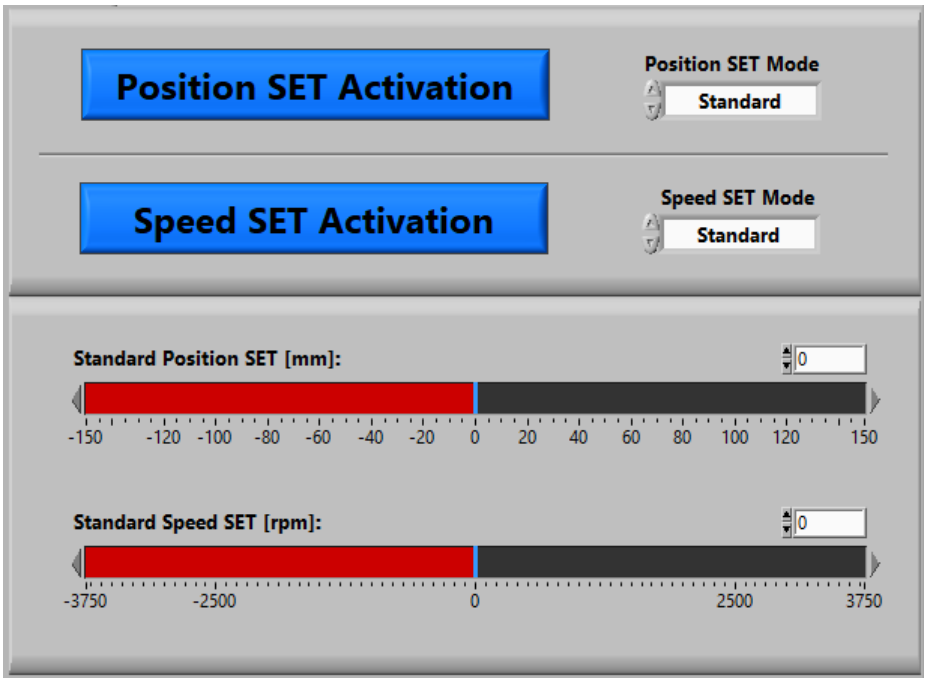

Figura 4.2: *Scheda "SET Choice" del Tab Control.*

In questa schermata è possibile distinguere due zone. Nella parte superiore, l'utente ha la possibilità di scegliere le modalità di impostazione del SET, con quattro combinazioni ottenibili: SET di posizione o velocità, "Standard" o "Custom". La parte inferiore contiene gli slider, uno per la posizione e uno per la velocità, per l'impostazione del valore discreto del SET di tipo "Standard".
Nella scheda *"Custom SET"* è possibile creare la funzione che andrà a costituire il segnale di SET. La schermata, riportata in **Figura 4.3**, contiene un grafico e diversi comandi. Il grafico serve a visualizzare su schermo la funzione che si sta assemblando. Per quanto riguarda i comandi, con il riquadro rosso viene evidenziata una serie di controlli "enum", atti alla parametrizzazione della funzione o, come si vedrà a breve, una parte di essa. Il controllo *"Signal Type [-]"* permette di scegliere, tra quelle disponibili, il tipo di funzione che si intende utilizzare: sinusoide, cosinusoide, rampa crescente, rampa decrescente e funzione a dente di sega, a onda quadra oppure triangolare. Scelto il tipo, è possibile impostare ampiezza, offset dello zero, frequenza etc. Il comando *"Duty Cycle [%]"* funziona esclusivamente per le funzioni a onda quadra. Il riquadro in verde evidenzia tre comandi di tipo booleano:

- *"+"* - Aggiunge un altro tratto di funzione a quello già presente.
- *"-"* - Dall'insieme di almeno due tratti di funzione di SET, rimuove l'ultimo aggiunto.
- *"END"* - Termina l'assemblaggio della funzione.

Il pulsante viola *"Save SET Signal"* permette di salvare la funzione creata in un file "*.txt", rinominabile e allocabile a piacimento.

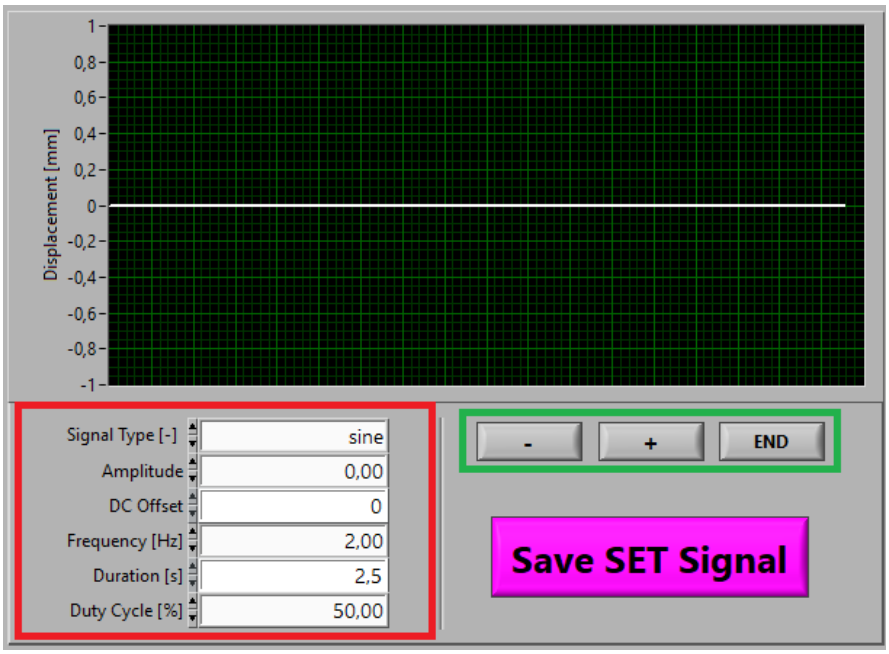

Figura 4.3: *Scheda "Custom SET" del Tab Control.*

La parte centrale del Front Panel viene riportata in **Figura 4.4**. Tramite riquadri colorati, vengono evidenziati due macro-settori.

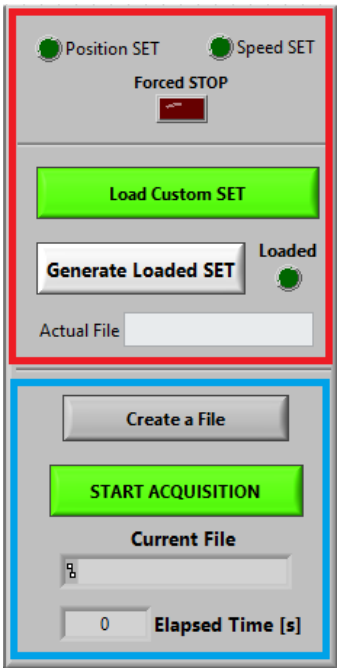

Figura 4.4: *Parte centrale del Front Panel per la gestione del segnale di SET e il salvataggio dei dati acquisiti.*

All'interno del riquadro rosso sono riportati tre LED: *"Position SET"*, *"Speed SET"* e *"Forced STOP"*. I primi due indicano il tipo di SET selezionato e sono i corrispettivi di quelli presenti nel VI RT. Durante le fasi di debugging, il VI ha dimostrato di non funzionare correttamente nel caso in cui vengano selezionati in contemporanea il SET di posizione e di velocità. Per evitare errori e malfunzionamenti, il VI è stato programmato per arrestarsi immediatamente al momento dell'attivazione contemporanea dei due controlli. L'indicatore "Forced STOP" si illumina a seguito dell'arresto forzato del VI. L'utente visualizzerà inoltre un pop-up che gli riferirà il motivo dell'arresto e che potrà procedere con un nuovo avvio del VI.
Sempre nello stesso riquadro, sono presenti i due comandi *"Load Custom SET"* e *"Generate Custom SET"*. Il primo serve per il caricamento del file "*.txt" contenente il segnale di SET customizzato. Il file deve soddisfare dei requisiti ben precisi: deve contenere due vettori colonna, uno riportante l'asse tempi e l'altro i valori assunti dalla funzione. Questo requisito viene specificato tramite un pop-up che compare alla pressione del comando, prima del caricamento. Il comando "Generate Loaded SET" avvia la generazione del segnale di SET: il CompactRIO invierà al driver del motore elettrico un segnale analogico di tensione corrispondente ai valori della funzione desiderata. L'indicatore *"Loaded"* dà conferma dell'avvenuto caricamento del file.
Il riquadro in azzurro contiene i comandi per il salvataggio dei dati acquisiti. Il comando *"Create a File"* permette la creazione del file di testo nel quale verranno salvati i dati, mentre *"START ACQUISITION"* avvia il salvataggio. Gli indicatori *"Current File"* e *"Elapsed Time [s]"* riportano, rispettivamente, il nome del file in cui verranno salvati i dati e il tempo trascorso dall'inizio della fase di salvataggio (espresso in secondi).

Visualizzazione dei dati e status delle acquisizioni

Parte dei dati ottenuti dall'acquisizione principale viene rappresentata su dei grafici posti sulla parte destra del Front Panel. Nello specifico, vengono riportati i grafici di posizione, velocità angolare, forza esterna e coppia. Essi sono riportati nella **Figura 4.5**.

La **Figura 4.6** riporta la scheda *"Termocouples"* del Tab Control. Essa contiene i grafici dei dati provenienti dall'acquisizione del modulo termocoppie.
Nella parte bassa della figura si possono notare tre indicatori LED che permettono all'utente di accertarsi del corretto funzionamento dei Network Streams. Questi indicatori trovano i loro corrispettivi nel VI Real-Time. *"Streams created"* indica se i Network Streams delle acquisizioni principale e dalle termocoppie sono stati creati. I LED *"Data Stream (MAIN)"* e *"Data Stream (TERM)"* indicano, per ciascun canale, se lo stream sta avvenendo correttamente.

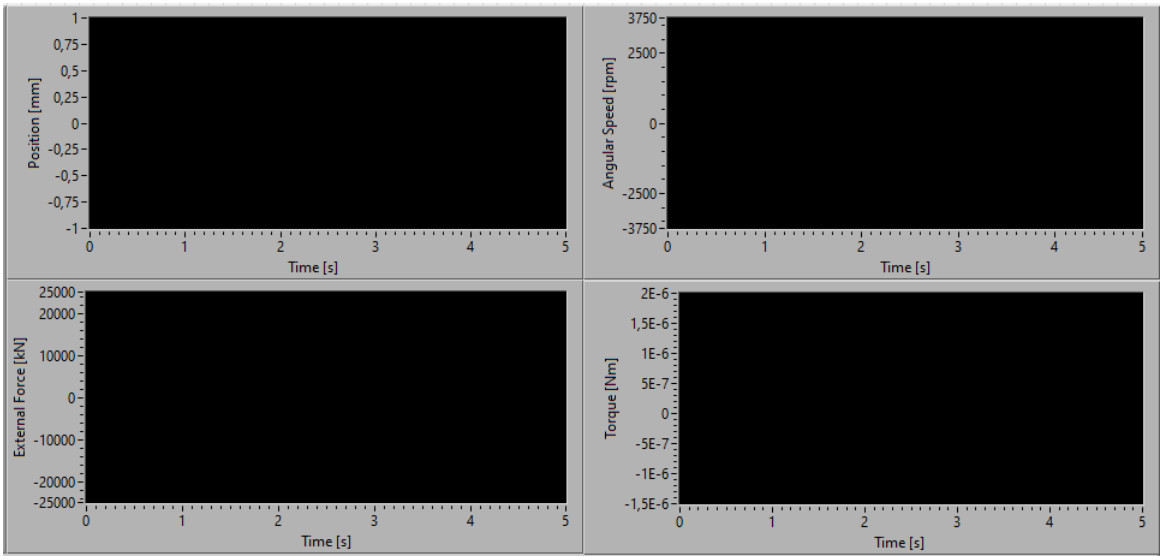

Figura 4.5: *Parte del Front Panel che contiene i grafici per la rappresentazione dei dati ottenuti dall'acquisizione principale.*

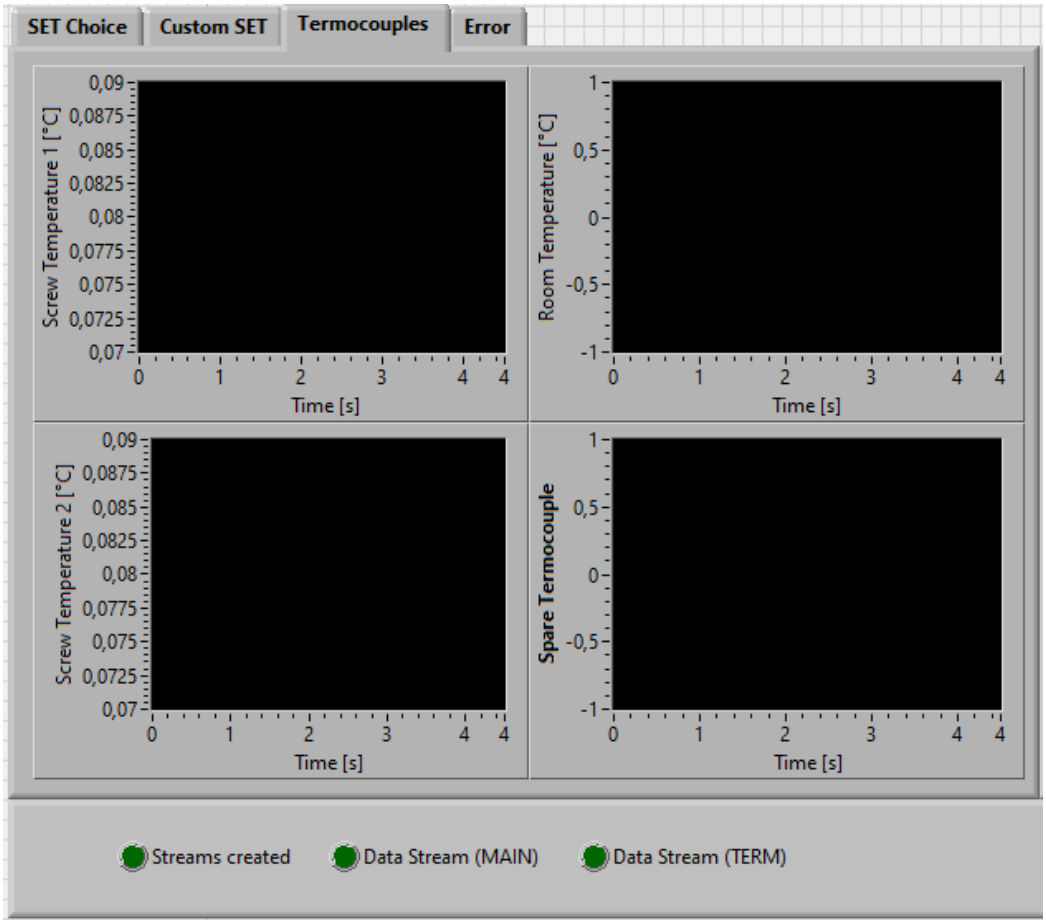

Figura 4.6: *Scheda del Tab Control contenente i grafici dei dati provenienti dal modulo delle termocoppie. Al di sotto, i LED per la verifica dello status delle acquisizioni.*

4.2 Block Diagram

In questa sezione verrà descritto il Block Diagram del VI PC.

4.2.1 Inizializzazione del VI e dei Network Streams

L'inizializzazione del VI è affidata a una Flat Sequence Structure, riportata in **Figura 4.7**.

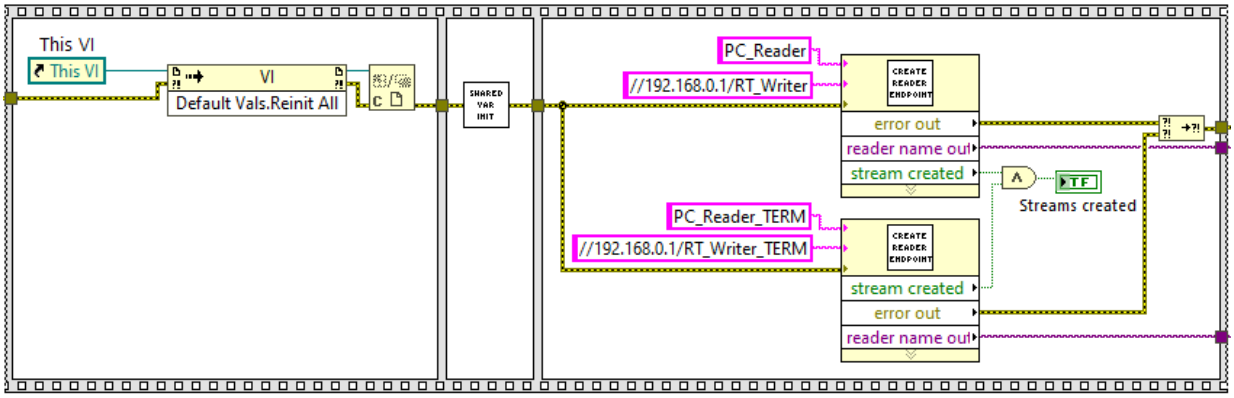

Figura 4.7: *Flat Sequence Structure per l'inizializzazione del VI.*

Il primo frame contiene un Invoke Node, per l'invocazione del metodo "Default Values: Reinitialize All To Default". Con questo, tutti i controlli presenti nel Front Panel vengono riportati al loro valore di default. Nel secondo, è presente un subVI nel quale vengono inizializzate le variabili condivise utilizzate nei VI PC e Real-Time principali. Il Block Diagram del subVI è riportato in **Figura 4.8**.

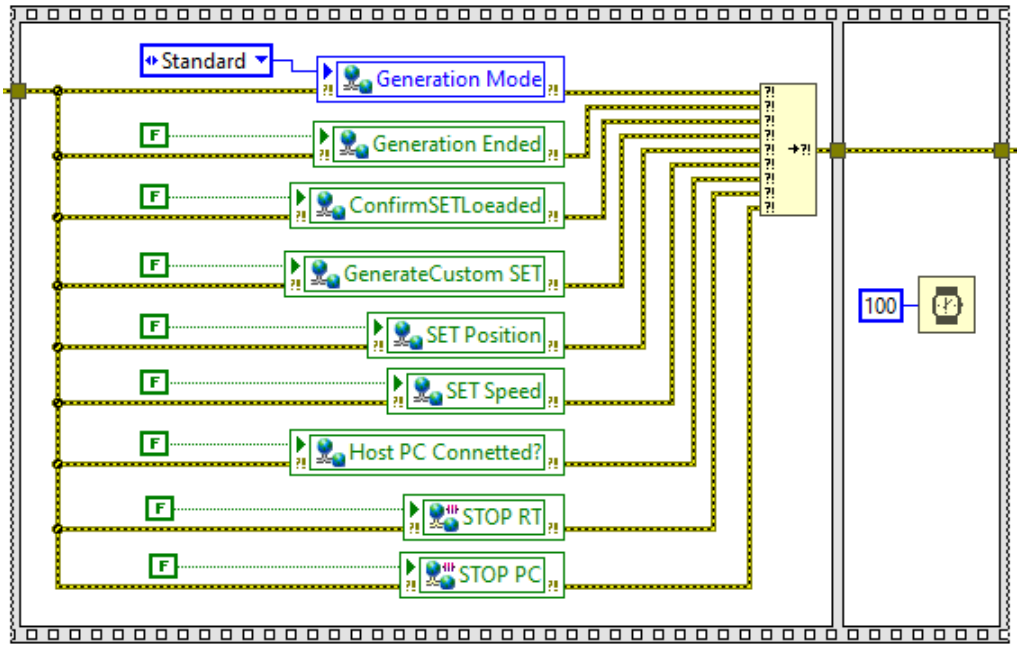

Figura 4.8: *Block Diagram del subVI per l'inizializzazione delle shared variable.*

L'inizializzazione consiste nell'attribuire, all'avvio del VI PC, un valore alle variabili condivise. Questo avviene nel primo frame della sequenza di **Figura 4.8**. Successivamente, viene utilizzata la funzione *"Wait (ms)"*, impostata a 100 ms, per fornire il tempo a tutte le variabili di inizializzarsi.
Tornando alla sequenza di inizializzazione del VI principale, il terzo e ultimo framae contiene il subVI *"CreateReaderEndpoint.vi"*. Come lo stesso nome suggerisce, esso serve per la creazione dell'endpoint *"reader"* del Network Stream tra VI RT e PC. Il blocco riceve in input due costanti di tipo stringa: il nome del lettore (e.g. *"PC_Reader"*) e l'URL per la connessione con lo scrittore. Affinché la connessione vada a buon fine, l'URL deve essere scritto nella forma che segue:

*//*indirizzo IP del target*/*nome del Writer**

Il subVI viene utilizzato due volte, una per il flusso di dati dall'acquisizione principale e una per l'acquisizione dal modulo termocoppie. A conferma dell'avvenuta creazione dell'endpoint, il blocco invia in output una variabile di tipo booleano; le due convergono in un operatore AND collegato all'indicatore LED "Streams created", visto nel Front Panel. Esso si illumina quando vengono creati entrambi gli endpoint. Le altre due uscite sono rappresentate dall'errore e dal riferimento del Network Stream. Quest'ultimo costituirà l'imput per il loop per la generazione dei grafici e il salvataggio dei dati. Il Block Diagram del subVI viene riportato in **Figura 4.9**.

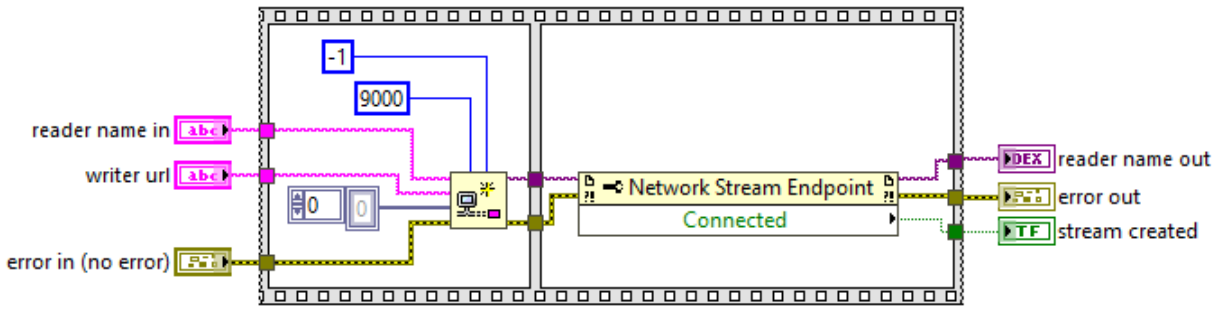

Figura 4.9: *Block Diagram del subVI "CreateReaderEndpoint.vi".*

4.2.2 Acquisizione principale - Rappresentazione e salvataggio dei dati

La **Figura 4.10** riporta il While Loop nel quale vengono processati i dati provenienti dal Network Stream dell'acquisizione principale. Essi vengono ivi letti, ne viene visualizzata una parte su dei grafici e, infine, vengono salvati su un file "*.txt". Il ciclo è impostato a una frequenza di 1 kHz.
In ingresso al ciclo si ha il riferimento al Network Stream, creato nella sequenza di inizializzazione e collegato al blocco per la lettura dei dati. All'uscita di quest'ultimo, i dati vanno in

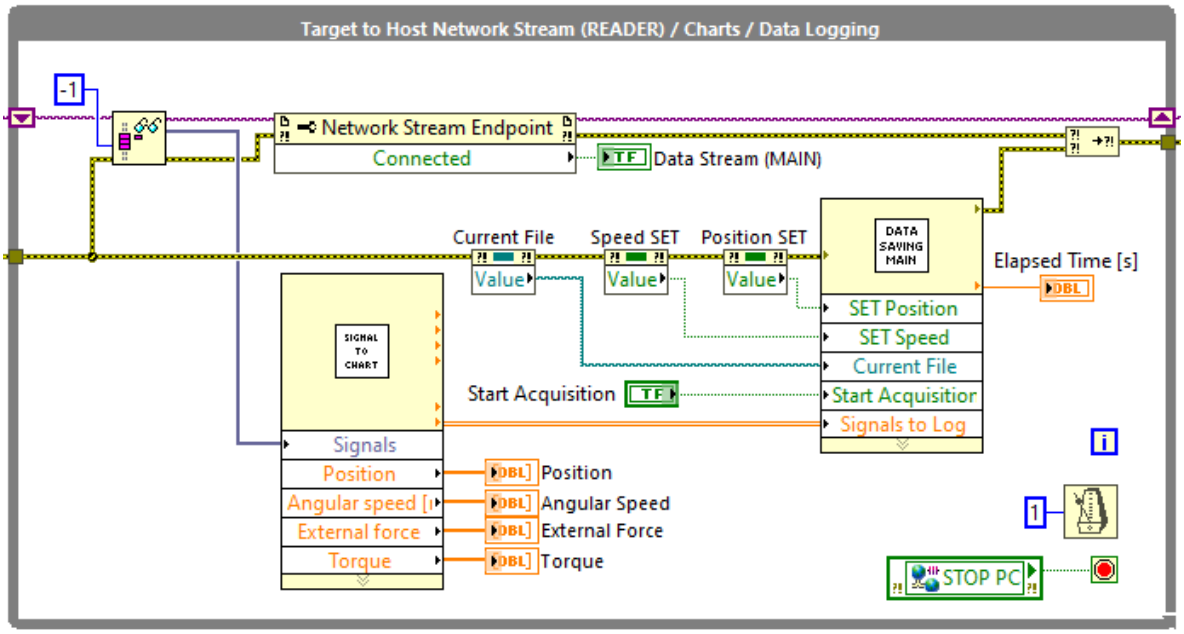

Figura 4.10: *Loop per la rappresentazione e il salvataggio dei dati dell'acquisizione principale.*

ingresso al subVI "SignalToChart.vi", già analizzato nel **Capitolo 3**. I valori di posizione, velocità angolare, forza esterna e coppia agente sul torsiometro vengono visualizzati su dei grafici. L'array contenente tutte le grandezze viene invece inviato in input al blocco per il logging dei dati. Gli altri ingressi sono costituiti dai comandi booleani *"Start Acquisition"*, *"Speed SET"* e *"Position SET"* e dal percorso, denominato *"Current File"*, del file "*.txt". Questo viene creato all'interno di un ciclo contenente una *"Event Structure"*, la quale verrà descritta in un paragrafo apposito.

Il Block Diagram del subVI per il salvataggio dei dati, denominato *"DataLogging_MAIN.vi"*, è riportato in **Figura 4.11**. Al suo interno, il comando booleano "Start Acquisition" regola lo switch di una Case Structure. Nel caso "False" ci si limita a rimandare in uscita il valore del contatore e del percorso del file di destinazione, senza ulteriori operazioni. Il caso "True" contiene invece una sequenza composta da tre frame. Nel primo, il valore corrente del contatore del ciclo While viene moltiplicato per l'unità di tempo del ciclo (0,001 s). Effettuando questo prodotto a ogni iterazione, viene costruito il vettore tempi che verrà affiancato a quelli dei dati dell'acquisizione. Nel secondo frame, è posizionata una Case Structure, comandata dalla funzione *"Equal To 0?"* collegata al contatore. In corrispondenza del primo ciclo, quando il contatore ha valore 0, si accede al caso "True" della struttura. Al suo interno, avviene la creazione e la scrittura dell'intestazione del file di log dei dati. Essa è rappresentata da un array di stringhe creato nel subVI *"ArraySTRHeader.vi"*; ne è riportato il Block Diagram in **Figura 4.12**. L'array viene poi inviato in input al blocco di scrittura sul file. L'intestazione cambia a seconda del SET selezionato; in caso non ne venga selezionato alcuno, l'intestazione della colonna del SET presenterà la dicitura "NO SET" (**Figura 4.13**).

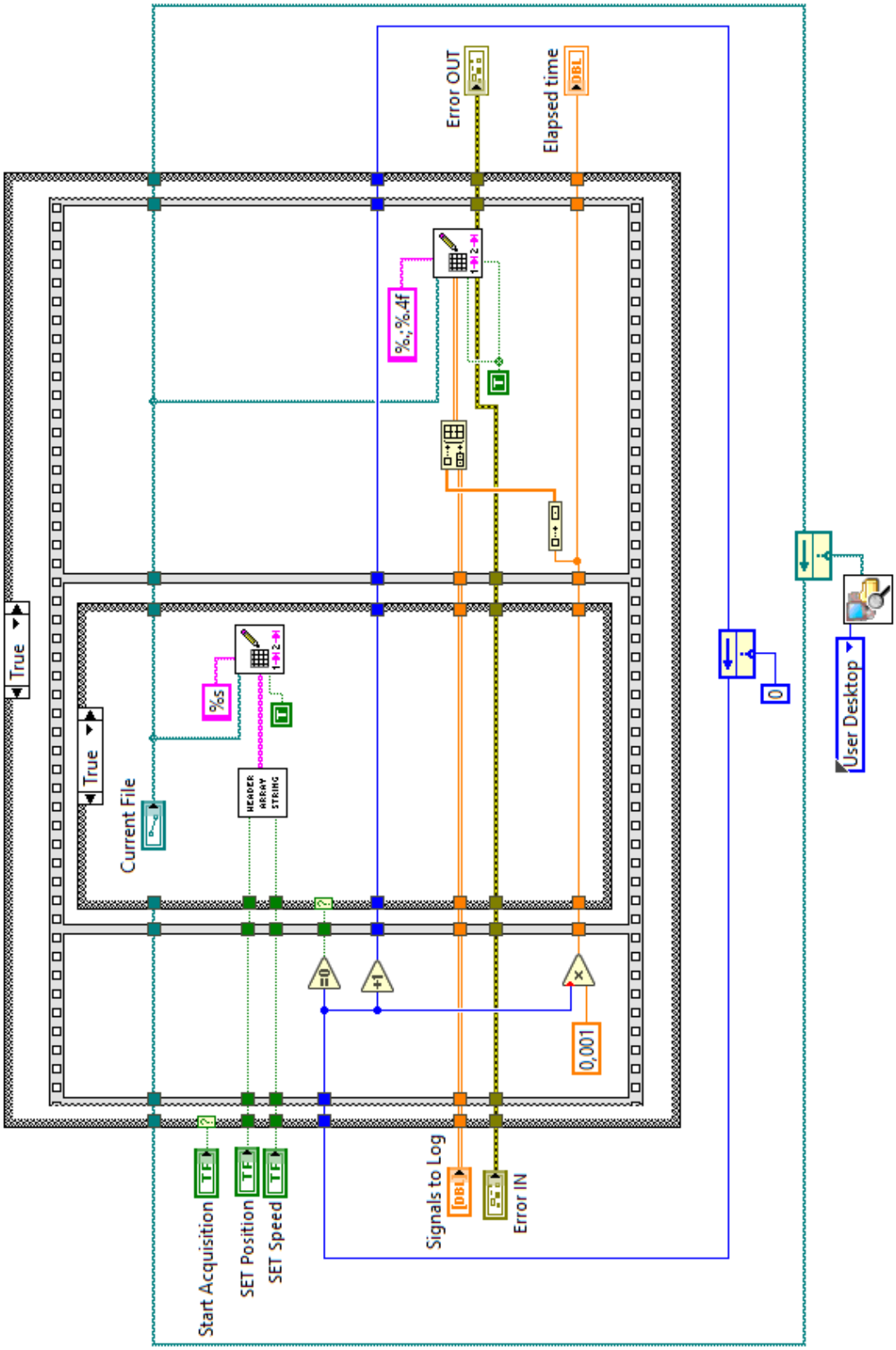

Figura 4.11: *Block Diagram del subVI "DataLogging_ MAIN.vi".*

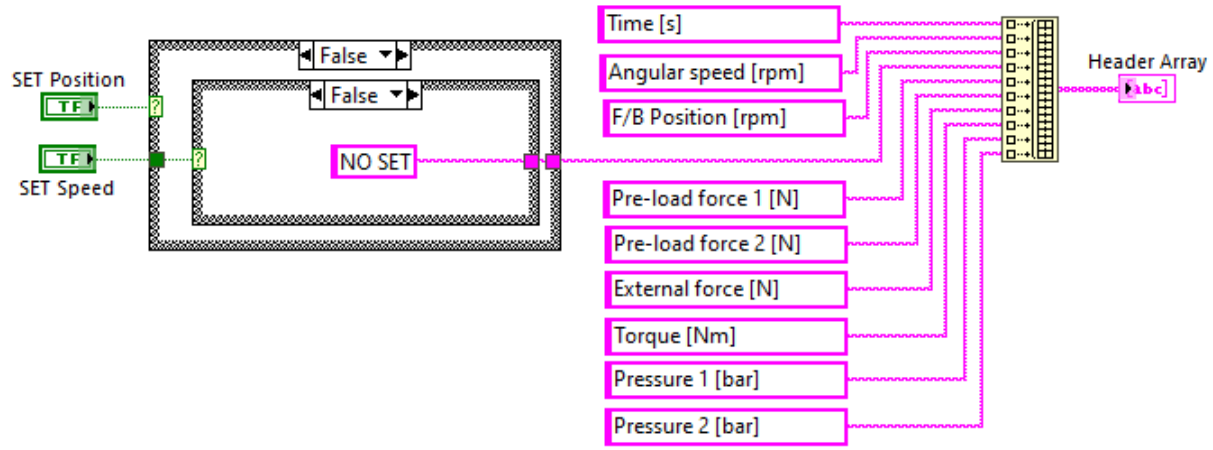

Figura 4.12: *Block Diagram del subVI "ArraySTRHeader.vi".*

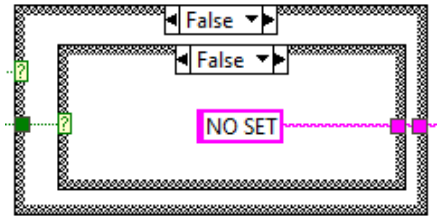

Figura 4.13: *Stringa "NO SET" che viene scritta nell'header del file nel caso non venga selezionato alcun tipo di SET.*

Il terzo e ultimo frame della sequenza contiene il blocco per l'unione del vettore dei tempi con il resto dei dati acquisiti e quello per il salvataggio dei dati sul file.
In uscita dal subVI si ha l'indicatore del tempo trascorso dall'inizio dell'acquisizione. Al di sotto della Case Structure principale, si hanno due *Feedback Node*: uno per il contatore e uno per il percorso del file. Questi hanno il compito di imporre il valore iniziale delle variabili e di rimandarne in input alla struttura il valore (eventualmente) aggiornato a ogni ciclo. Il loro utilizzo scongiura inutili complicazioni nel Block Diagram del VI principale. Non usandoli, bisognerebbe inizializzare le variabili al di fuori del Timed Loop e inviarle al VI "DataLogging_MAIN.vi". Da qui, esse verrebbero rinviate nel loop principale e, tramite Shift Register, di nuovo in ingresso al ciclo.

4.2.3 Acquisizione dal modulo termocoppie - Rappresentazione e salvataggio dei dati

Il loop realizzato per l'acquisizione dal modulo delle termocoppie è realizzato partendo da quello dell'acquisizione principale, visto precedentemente. La necessità di usare due loop e, come si vedrà, due file separati nasce dalle diverse frequenze di campionamento. Utilizzando un unico file, a fine acquisizione ci sarebbero stati due gruppi di vettori con un diverso numero finale di sample, complicando inutilmente le successive fasi di data processing. Per

ovviare a questo problema, si è scelto di creare un secondo file di log dei dati, unicamente dedicato alle termocoppie.

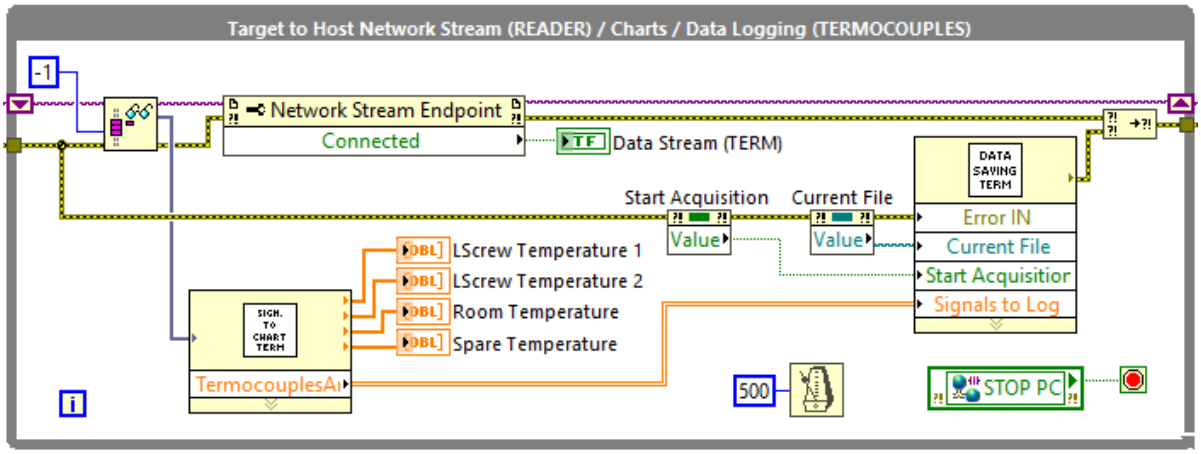

Figura 4.14: *Loop per la rappresentazione e il salvataggio dei dati dell'acquisizione dal modulo termocoppie.*

In **Figura 4.14** si riporta il ciclo While in cui vengono letti e convertiti i valori di tensione provenienti dal Network Stream. Il subVI per la conversione delle tensioni è lo stesso usato nel VI RT. Come già detto nel **Capitolo 3**, al suo interno non è ancora stato implementato un modello matematico che effettui la conversione delle tensioni in valori di temperatura. Allo stato attuale, esso si limita a mandare in uscita parte dei valori ricevuti in ingresso. Alle uscite di questo blocco vengono collegati quattro grafici per la visualizzazione dei dati nel Front Panel.
L'array contente i valori delle temperature va in ingresso al subVI per il salvataggio dei dati, denominato *"DataLogging_ TERMOCOUPLES.vi"*; ne viene riportato il Block Diagram in **Figura 4.16**. Anche in questo caso, esso è costituito da una Case Structure, il cui caso "True" contiene una Flat Sequence. Lo switch tra i casi è comandato dallo stesso comando "Start Acquisition" utilizzato per l'acquisizione principale.

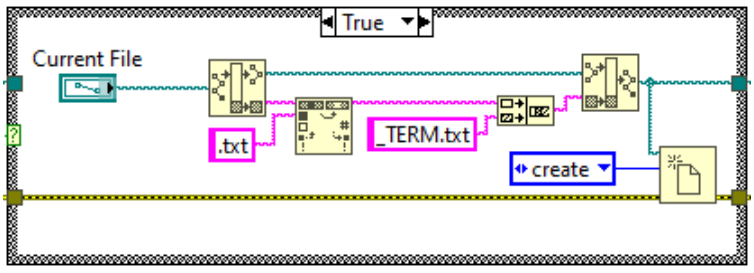

Figura 4.15: *Case Structure all'interno del subVI "DataLogging_ TERMOCOUPLES.vi" per la creazione del file in cui verranno salvati gli array dei valori di temperatura.*

Il primo frame della sequenza riporta esclusivamente l'operazione di aggiornamento del contatore e della sua moltiplicazione per l'unità di tempo. Nel secondo, è presente una Case

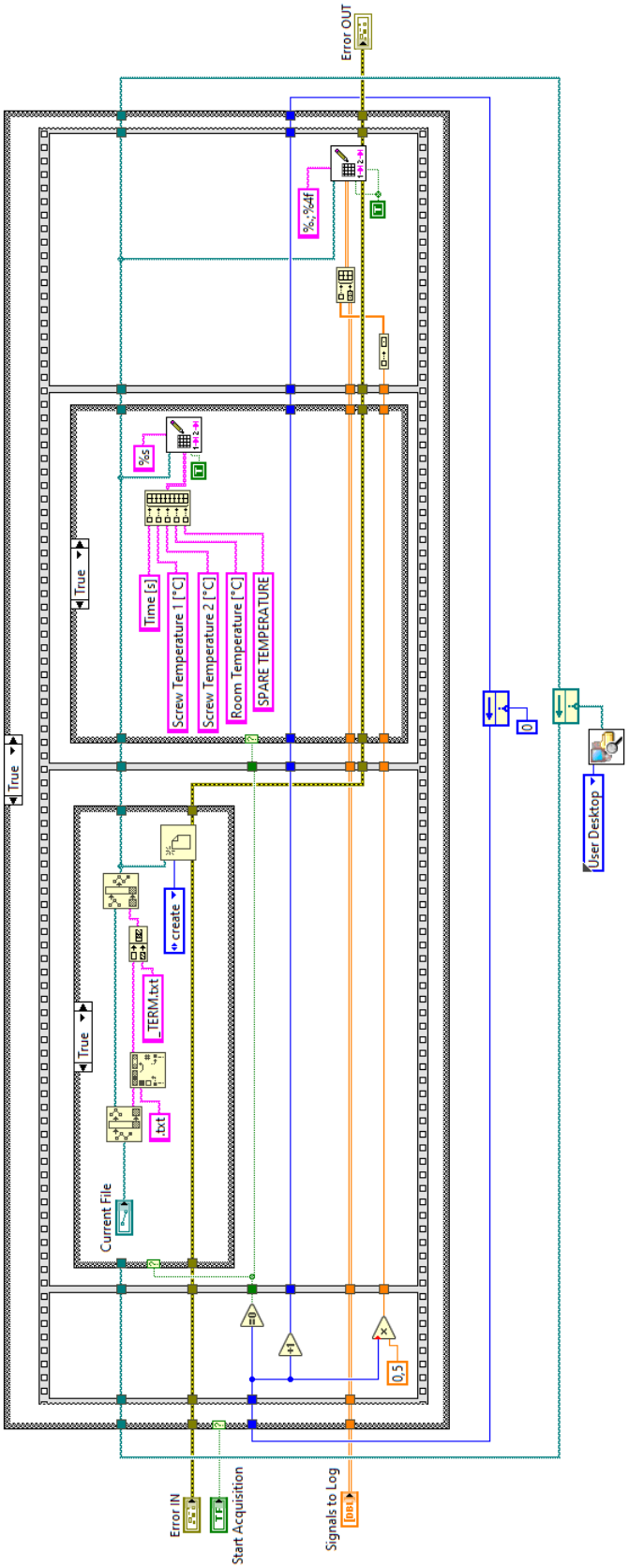

Figura 4.16: *Block Diagram del subVI "DataLogging_ TERMOCOUPLES.vi".*

Structure che al "ciclo zero" crea il file nel quale verranno scritti i dati (**Figura 4.15**).
Il file viene creato automaticamente. Partendo del file dell'acquisizione "MAIN", allocato e nominato manualmente dall'utente, il codice crea un file "*.txt" nella stessa cartella di destinazione e con lo stesso nome, con l'aggiunta del suffisso "*_TERM.txt" per distinguerlo dal primo.
Il terzo frame si occupa della generazione e scrittura dell'intestazione nel file di log dei dati, in corrispondenza del primo ciclo. Come la precedente, la Case Structure è comandata dalla funzione "Equal To 0?", collegata al contatore. Nel quarto e ultimo frame, il vettore tempi viene unito all'array dell'acquisizione e tutti i dati raccolti scritti sul file.

4.2.4 Creazione del segnale di SET

Il segnale di SET viene creato all'interno del loop riportato nella **Figura 4.18**. Il suo scopo è la creazione di un file "*.txt" contenente due vettori colonna: uno dell'asse tempi e l'altro dell'andamento del SET. Come già visto per il Front Panel, il programma permette all'utente di "assemblare" il segnale di SET, concatenando tratti di funzione scelti all'interno di un elenco preimpostato. Per ciascun tratto, è possibile settare ampiezza, offset dello zero e tutti gli altri parametri che ne definiscono l'andamento. A livello di Block Diagram, i valori di ciascun tratto di funzione vengono raccolti in un array. Questo viene a sua volta inserito in un cluster, assieme a un intero che ne riporta il numero di sample.

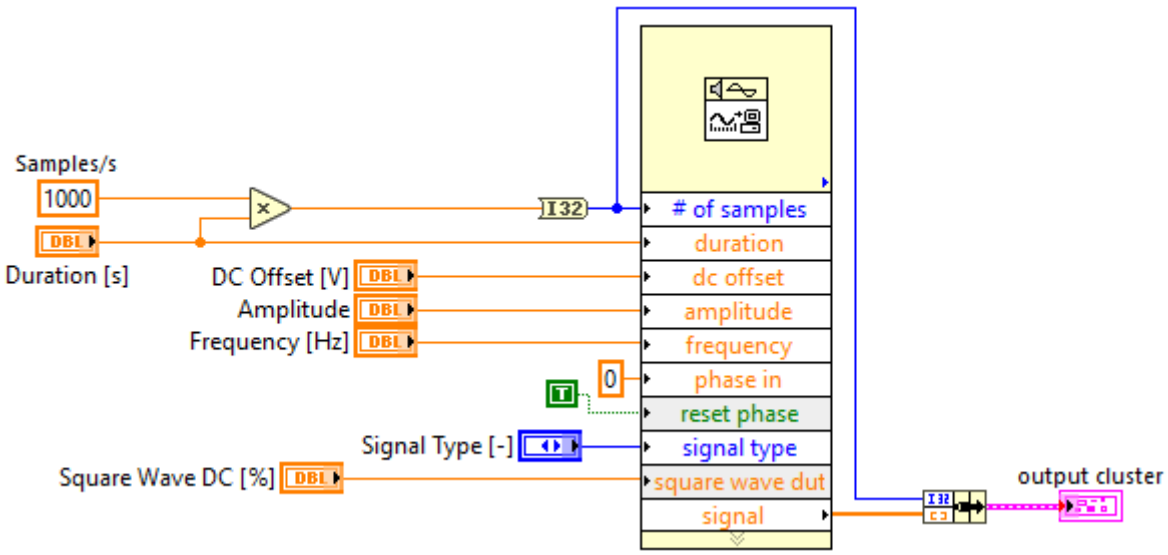

Figura 4.17: *Block Diagram del subVI "SETSignalCreator.vi".*

Due loop annidati nel principale scandiscono il susseguirsi dei cicli. Il loop più interno è quello contenente il subVI *"SETSignalCreator.vi"* (il Block Diagram è riportato in **Figura 4.17**), che contiene il blocco generatore di funzioni e si occupa della creazione del cluster. I parametri della funzione, in ingresso al subVI, sono inviati al blocco *"Signal Generator by Duration.vi"*. Si tratta di una funzione presente di default su LabVIEW dalla cui uscita

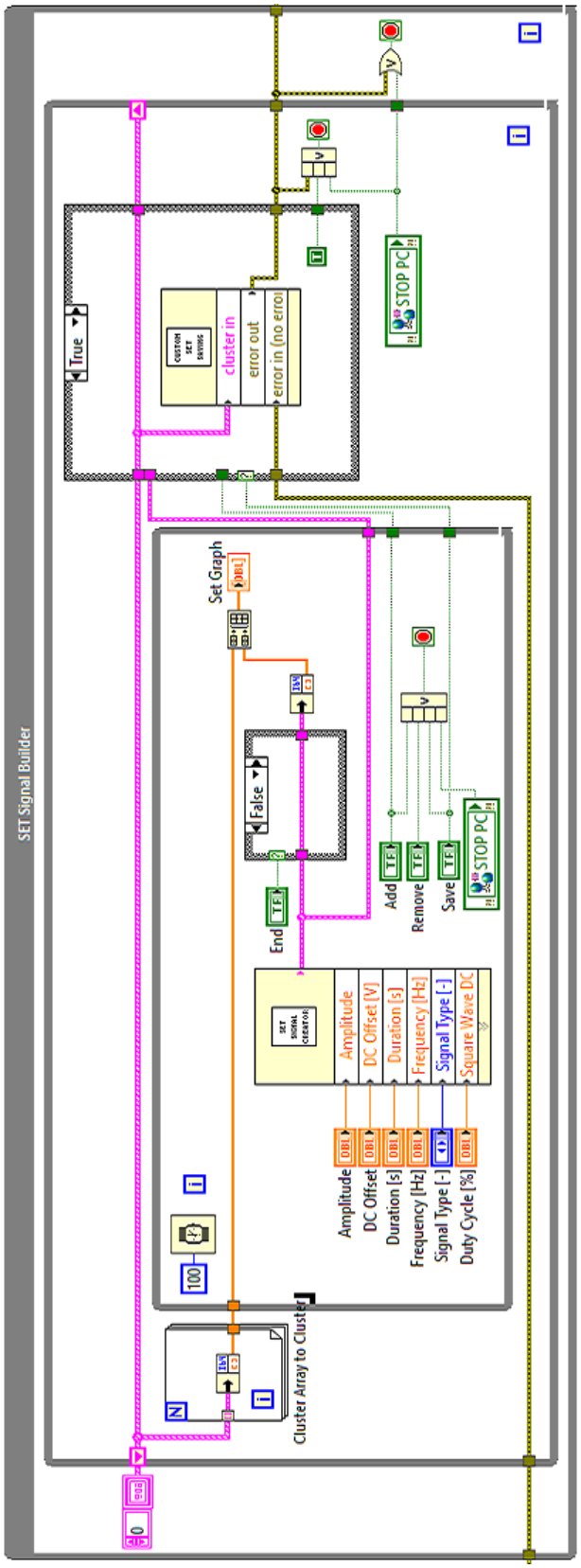

Figura 4.18: *Loop per la creazione del segnale di SET.*

proviene l'array contenente i valori del tratto di funzione. Il numero di sample della funzione viene ottenuto moltiplicando la durata, impostata dall'utente, per il numero di sample al secondo. Dovendo rendere compatibile il segnale di SET con un'acquisizione effettuata a 1 kHz, il valore è costante e pari a 1000 sample/s. Il cluster ottenuto è poi mandato in uscita dal subVI.
L'aggiunta o la rimozione di un tratto di funzione (i comandi "+" e "-" nel Front Panel), nonché il comando di salvataggio su file, determina l'arresto del loop più interno, permettendo l'inizio di un altra iterazione di quello immediatamente più esterno, nel quale è contenuto, e avviandone nuovo ciclo. I loop possono essere arrestati anche tramite l'azione del comando *"STOP PC"*. I cluster che a ogni ciclo abbandonano il subVI "SETSignalCreator.vi" vengono raccolti in un array. Si tratta di un *array di cluster* che viene inizializzato all'esterno del loop intermedio e rimandato continuamente in input tramite Shift Register.
Nella parte di Block Diagram in **Figura 4.19**, il filo del cluster prende due direzioni. Da un lato, esso si connette a un blocco *"Unboundle"*: viene estratto l'array dei valori del SET che, dopo essere stato unito ai precedenti tratti di funzione, viene riportato su un grafico. L'altro capo esce dal loop ed entra in una Case Structure che si occupa del salvataggio del segnale e dell'aggiunta, o rimozione, del nuovo tratto di funzione. Si noti anche la presenza dei comandi booleani "End", "Add", "Remove" e "Save". Gli ultimi tre sono connessi all'operatore OR che comanda lo stop del loop.

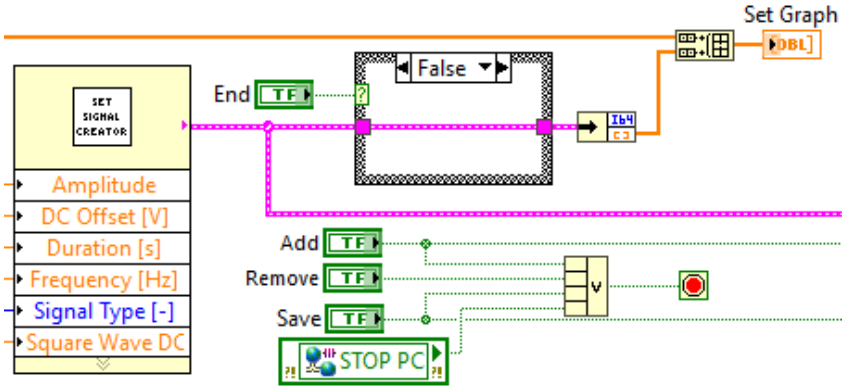

Figura 4.19: *Parte di Block Diagram immediatamente al di fuori del blocco "SETSignalCreator.vi".*

Come già accennato, all'esterno del loop interno è presente una Case Structure. Nel suo caso *"False"*, è presente una seconda Case Structure annidata. Gli switch sono gestiti dai comandi booleani "Add" e "Save". Si fa riferimento alla **Figura 4.20** nel caso in cui, all'arresto del ciclo interno, non sia stato selezionato il comando "Save" (che controlla la struttura più esterna).
Quando si preme il tasto "Remove", il ciclo si arresta (combinazione *"False/False"*) e ci si trova nella combinazione sulla sinistra della figura. In questo caso, il cluster contenente l'ultimo tratto di funzione viene rimosso dall'array di raccolta, tramite il blocco *"Delete From Array"*. Quando viene attivato il comando "Add", ci si trova nella combinazione *"False/True"* (lato destro della figura) e il cluster viene semplicemente unito all'array con la

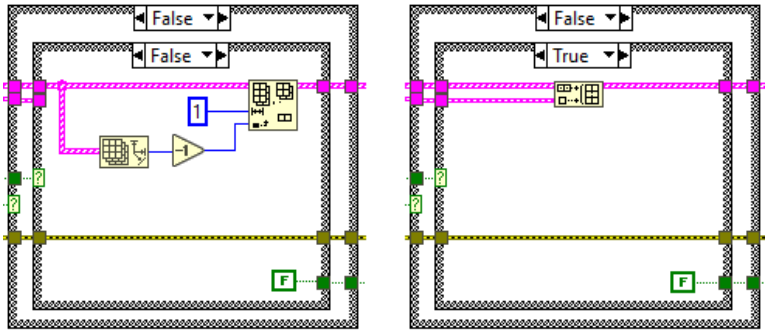

Figura 4.20: *Case Structure per la gestione del cluster contenente il segnale di SET. La struttura esterna è controllata dal comando "Save", quella interna dal comando "Add".*

funzione *"Build Array"*.
Nel caso in cui all'arresto del ciclo venga attivato il comando "Save", ci si trova nel caso riportato in **Figura 4.21**. L'array di cluster viene ivi inviato al subVI *"CustomSETSavings.vi"*. Nel suo Block Diagram (**Figura 4.22**) è presente una Flat Sequence composta da tre frame.

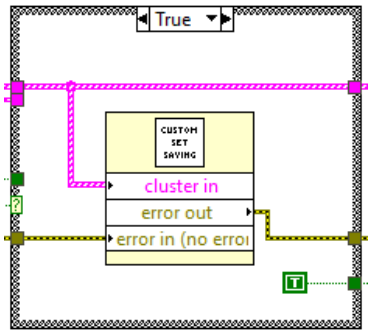

Figura 4.21: *Caso "True" della Case Structure per il salvataggio del segnale di SET.*

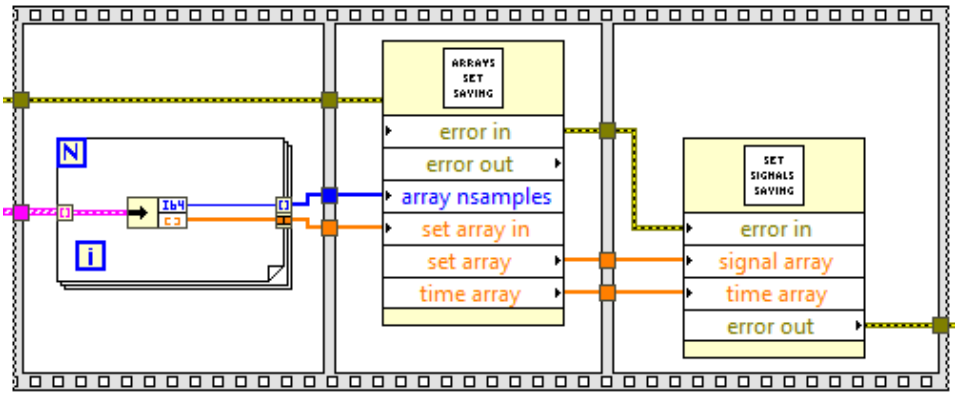

Figura 4.22: *Block Diagram del subVI "CustomSETSavings.vi".*

Nel primo frame, l'array di cluster viene scomposto sfruttando le proprietà di indicizzazione dei cicli For. Nello specifico, i numeri di sample dei diversi tratti di funzione vengono inseriti

in un array. I valori delle funzioni vengono invece concatenati in un unico vettore. Entrambi gli elementi vanno in ingresso al subVI *"SETTimearrays.vi"*, posizionato nel secondo frame e riportato in **Figura 4.23**. Esso è composto a sua volta da una struttura con due sequenze. Nella prima è presente un loop che somma i numeri di sample di tutti i tratti di funzione, ottenendone il totale. Il ciclo si arresta automaticamente all'esaurimento dell'array. La somma ottenuta passa alla seconda sequenza, nel quale viene collegata al terminale *"N"* di un ciclo For. All'interno di quest'ultimo, moltiplicando il contatore del ciclo per l'unità di tempo (0,001 s) e sfruttando l'indicizzazione automatica in uscita, si ottiene il vettore tempi del segnale di SET.

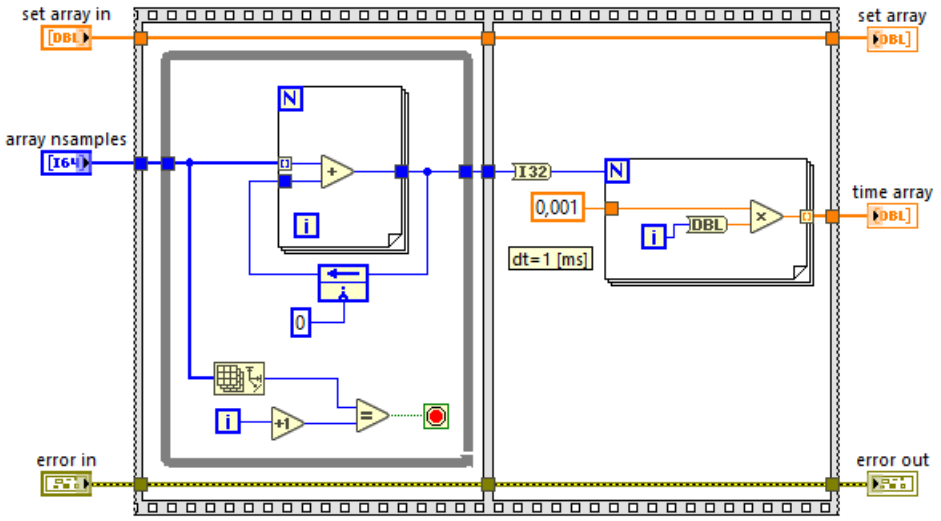

Figura 4.23: *Block Diagram del subVI "SETTimeArrays.vi".*

Nel terzo frame del subVI "CustomSETSavings.vi", i due array ottenuti vanno in ingresso al subVI *"SETSignalSavings.vi"* (**Figura 4.24**). Al suo interno è presente un'altra Flat Sequence. Nel primo riqadro è presente il blocco che, aprendo una schermata di dialogo con l'utente, permette di scegliere il nome e la destinazione del file di testo. Nel secondo, i dati vengono effettivamente salvati tramite l'apposita funzione.

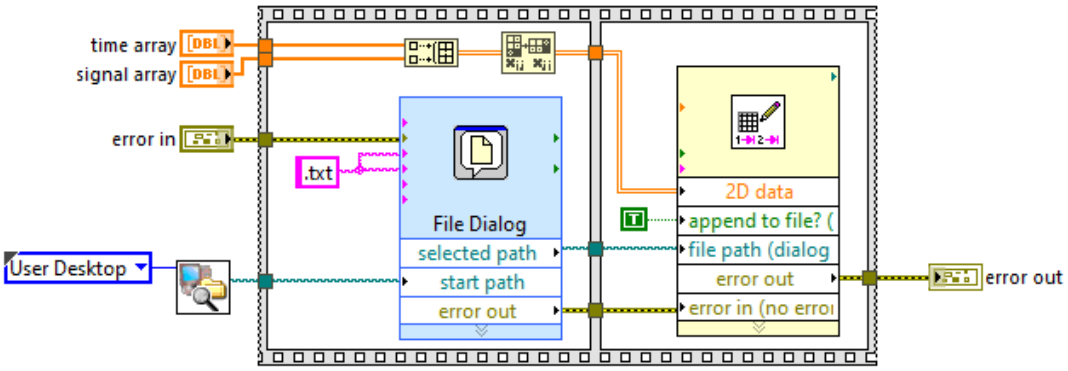

Figura 4.24: *Block Diagram del subVI "SETSignalSaving.vi".*

4.2.5 Events Loop

Nella parte alta del Block Diagram, è riportata un loop contenente una *"Event Structure"*. Essa è simile a una Case Structure, con la differenza che lo switch tra i diversi casi non è gestito da una variabile collegata alla stessa struttura, ma dall'occorrenza di determinati eventi, impostati durante la fase di programmazione. Esemplificativamente, un evento può essere l'attivazione di un comando booleano, il cambio di valore di una variabile e così via. A ogni evento corrisponde quindi un caso della struttura.
La struttura in esame contiene quattro casi. Il primo (**Figura 4.25**) si attiva tramite il comando *"Create a File"* e contiene una Flat Sequence per la creazione del file di log per i dati acquisiti dal CompactRIO. Il secondo caso (**Figura 4.26**) serve esclusivamente all'arresto del VI.

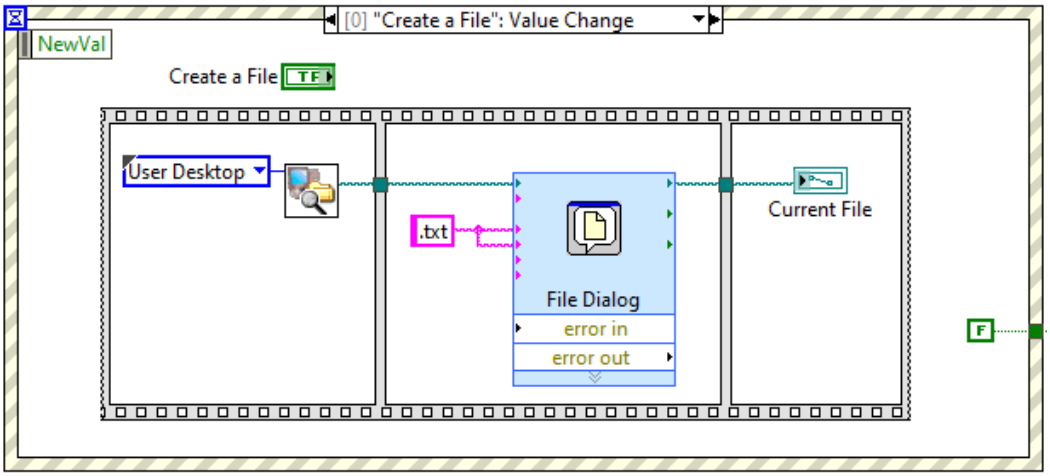

Figura 4.25: *Primo caso della Event Structure.*

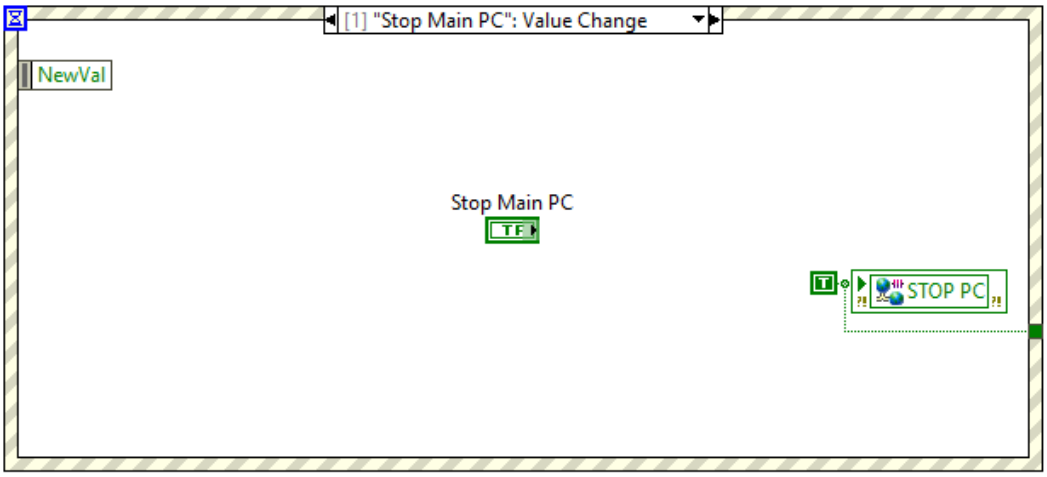

Figura 4.26: *Secondo caso della Event Structure.*

Il terzo caso, riportato in **Figura 4.27**, contiene la sequenza atta all'arresto forzato del VI, nel caso in cui vengano attivati contemporaneamente i SET di posizione e velocità.

Considerando la successione dei frame, inizialmente viene visualizzato un messaggio di errore che informa l'utente di quanto accaduto e lo invita a riavviare il VI. Successivamente, il LED "Forced STOP" sul Front Panel viene spento e i comandi "Position SET" e "Speed SET" riportati a "False". Infine, il VI viene arrestato.

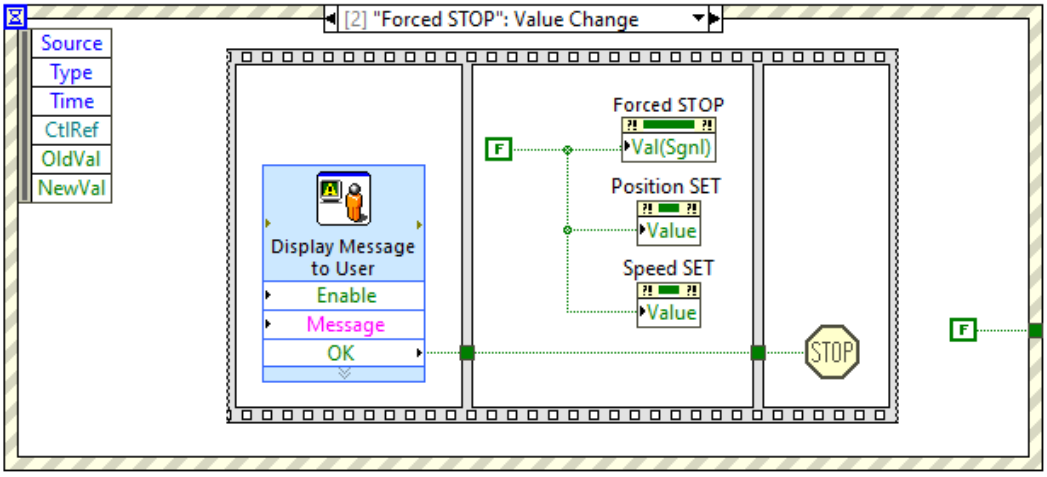

Figura 4.27: *Terzo caso della Event Structure.*

Nel quarto e ultimo caso (**Figura 4.28**) ci si occupa del caricamento del file ".txt" contenente il segnale di SET di "custom". Infatti, esso si attiva alla pressione del pulsante "Load Custom SET".

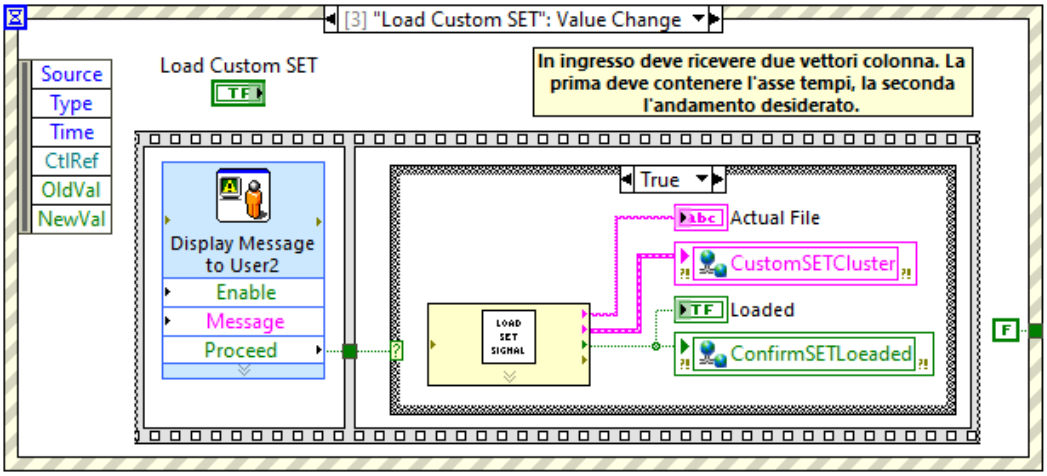

Figura 4.28: *Quarto caso della Event Structure.*

Al suo interno è presente una sequenza. Inizialmente, viene mostrato un messaggio all'utente, specificandogli che il file caricato deve contenere esclusivamente due vettori colonna (l'asse tempi sulla sinistra, il segnale sulla destra e senza intestazione) e chiedendogli se si voglia proseguire con l'upload. In caso affermativo, il booleano "Proceed" in uscita dal blocco attiva il caso "True" della Case Structure nel frame successivo. Essa contiene il subVI *"LoadSET.vi"*, per l'estrazione del cluster contenente il SET e della stringa col nome del file.

Questi dati vengono poi scritti su delle variabili globali, lette nel VI RT. Il Block Diagram del subVI "LoadSET.vi" è riportato in **Figura 4.29**

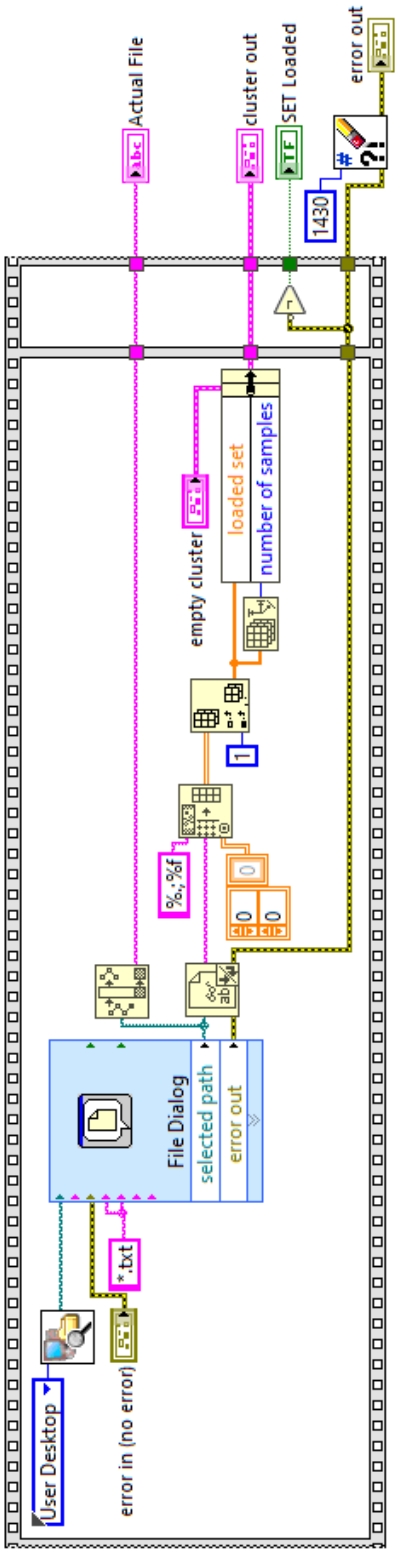

Figura 4.29: *Block Diagram del subVI "LoadSET.vi".*

4.2.6 Invio dei comandi al VI RT

Il loop in **Figura 4.32** completa il Block Diagram del VI PC. Esso svolge una serie di funzioni legate ai comandi posizionati nel Front Panel e all'invio del loro valore corrente al VI RT. Inoltre, è presente l'indicatore *"Generation Ended"*, il cui valore viene letto da una variabile condivisa scritta nel Real-Time. Il LED si illumina quando viene terminata la fase di generazione del segnale di SET "custom".
Il riquadro rosso evidenzia i due comandi "Position SET" e "Speed SET". Essi vengono collegati a un blocco AND che, a sua volta, comanda la Case Structure dove viene attivato il LED "Forced STOP". É proprio l'attivazione di questo indicatore che avvia l'evento correlato all'interno della Event Structure.

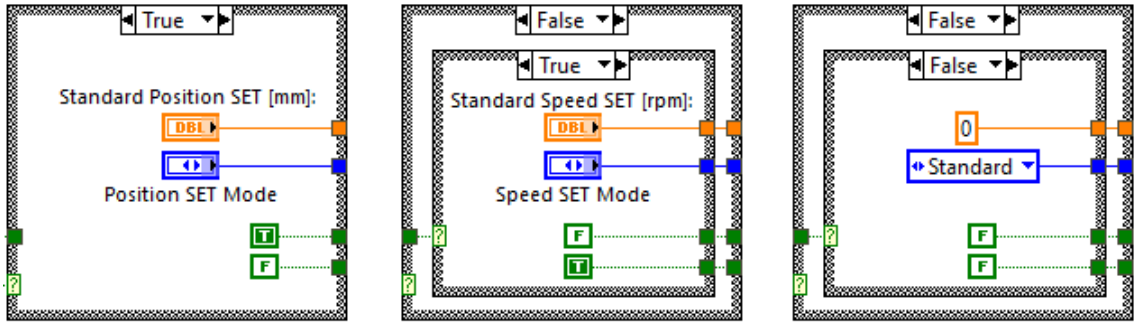

Figura 4.30: *Possibili combinazioni delle Case Structure regolate dai comandi "Position SET" e "Speed SET".*

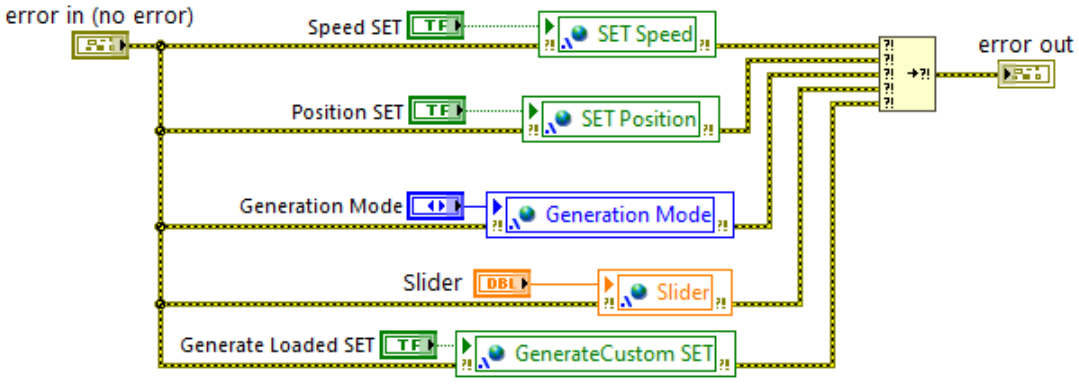

Figura 4.31: *Block Diagram del subVI "SharedVariableWriter.vi".*

I due controlli del SET comandano un'altra Case Structure, contenuta dal riquadro in azzurro. Nello specifico, essa viene comandata dal controllo di posizione. Il caso "False" contiene una seconda struttura, comandata da quello di velocità. Le possibili combinazioni dei casi ottenibili dalle due strutture sono riportati in **Figura 4.30**. In ciascuno dei casi "True", ovvero quando viene attivato uno dei due SET, gli output sono costituiti dal valore di posizionamento dello slider, dalla modalità d'impostazione del SET ("Standard" o "Custom") e dalle costanti booleane V/F che indicano il controllo selezionato. La combinazione "Fal-

se/False" è quella di default, quando nessun SET è stato ancora selezionato.
Assieme al comando *"Generate Loaded SET"*, queste grandezze costituiscono gli input al subVI *"SharedVariableWriter.vi"*; ne è riportato il Block Diagram in **Figura 4.31**. Al suo interno, le grandezze in ingresso vengono scritte su delle Shared Variable, poi lette nel VI RT. Se ne deduce che la parte Real-Time viene completamente comandata dall'utente attraverso il VI PC.

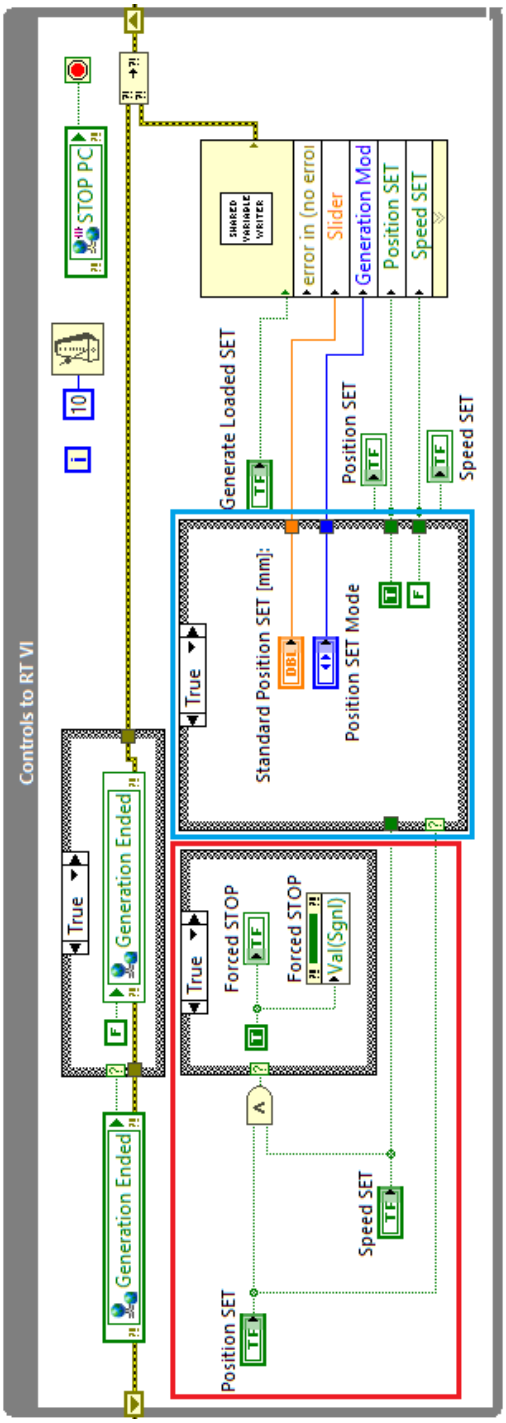

Figura 4.32: *Loop per l'invio dei comandi al VI Real-Time.*

Capitolo 5

Conclusioni e sviluppi futuri

Il lavoro ha riguardato la realizzazione di un codice per l'acquisizione dati e il controllo di un banco prova per viti a ricircolo di sfere. Il tutto è stato realizzato sul software LabVIEW, fornito dalla National Instruments. L'attività, dalla durata di quasi un anno, è iniziata con uno stage all'interno del laboratorio PEIC del Politecnico di Torino, per poi proseguire nei mesi successivi con l'attività di tesi.
Personalmente, reputo l'attività come un valido proseguo di quanto studiato durante la preparazione degli esami. Le mie conoscenze pregresse si sono rivelate una solida base da cui partire, grazie anche all'interdisciplinarità del corso di studi. Tuttavia, si è comunque rivelata necessaria un'intensa fase di formazione, per mettermi nelle condizioni di poter utilizzare il software e operare sul banco con efficacia e in tutta sicurezza. Dal punto di vista della postazione di lavoro, ho partecipato a diversi incontri preliminari, nei quali mi è stata fatta una descrizione e introduzione all'utilizzo del banco prova. Dal punto di vista del software, ho seguito dei corsi di formazione online erogati dalla stessa National Instruments. Nello specifico:

- *LabVIEW Core 1*
- *LabVIEW Core 2*
- *LabVIEW FPGA*
- *LabVIEW Real-Time*

La fase di formazione ha previsto inoltre la realizzazione di un programma per l'acquisizione dati da un CompactRIO 9074, simile a quello installato attualmente sul banco. Al termine di questa attività, ho potuto dedicarmi interamente al lavoro vero e proprio.

Per la realizzazione del codice, sono stati presi due punti di riferimento: un programma LabVIEW realizzato per un altro banco prova, sempre presente nei laboratori del Politecnico, e quello imbastito da un altro tesista, specifico per il banco prova in esame ma incompleto.

Inizialmente, ho preferito concentrarmi sullo sviluppo del sistema di acquisizione; in particolare, il calcolo della posizione partendo dagli input provenienti dalla riga ottica e, successivamente, la lettura dei valori di tensione dai moduli analogici. Una volta ottenuto un sistema d'acquisizione funzionante, mi sono potuto dedicare alla realizzazione delle parti che completano il codice fino a questo punto; ad esempio, la generazione del segnale di SET e il salvataggio dei dati.

Le criticità principali hanno riguardato la comprensione dei metodi di comunicazione tra le diverse parti del codice (VI FPGA, VI Real-Time e VI PC Host) e di come armonizzare la loro interconnessione. L'estrema versatilità del programma, dalla quale derivano numerosissime funzionalità e moduli software aggiuntivi, ha senza dubbio rallentato la scelta delle modalità di realizzazione del codice, nonché la scelta delle funzioni da implementare per le diverse funzionalità previste dal programma. Altre complicazioni sono sorte a seguito di alcuni interventi sull'impianto elettrico del banco, che hanno portato a una fase di stallo nelle fasi finali del lavoro.
Ciononostante, l'attività di tesi si è conclusa con la realizzazione, più che soddisfacente, del codice di controllo, gestione e acquisizione del banco. Allo stato attuale, esso è in grado di acquisire e salvare su file i valori della posizione e delle grandezze provenienti dai moduli analogici. Inoltre, è possibile l'impostazione del tipo di controllo (posizione o velocità) e l'assemblaggio di una funzione di SET da fornire al driver di pilotaggio del motore elettrico.

5.1 Sviluppi futuri

Lo sviluppo del programma di acquisizione e controllo del banco prova non è ancora terminato. Oltre agli interventi specifici per ogni VI principale, descritti di seguito, complessivamente è possibile espandere notevolmente le funzionalità del banco. Un esempio tra tutti, potrebbe essere quello di far lavorare il banco in controllo forza, in aggiunta al controllo posizione e velocità. Inoltre, è opportuno ricordare che il banco è dotato di un sistema pneumatico per l'applicazione del carico, il quale fino ad adesso è stato del tutto trascurato dal punto di vista software.
É fondamentale un'attenta revisione del codice di controllo del driver Lenze, realizzato sul software proprietario Lenze Engineer. É indispensabile che il codice sul Lenze venga aggiornato, nonché adattato al nuovo codice LabVIEW e alle modifiche eseguite recentemente all'impianto elettrico del banco.

VI FPGA All'interno del VI FPGA, è presente una parte di codice realizzata da un altro studente e atta al calcolo della velocità angolare della vite, partendo dal numero di incrementi dell'encoder rotativo sul torsiometro. Date le problematiche legate all'impianto elettrico del banco, non è stato possibile revisionare e rendere operativa questa sezione di codice.

VI Real-Time All'interno del VI Real-Time, ma anche del VI PC, è necessaria la creazione del modello matematico per la conversione delle tensioni, provenienti dal modulo delle termocoppie, in valori di temperatura. É necessario quindi il completamento del subVI *"SignalToChart_ Termocouples.vi"*, già impostato per ospitare il modello.

Un'idea molto valida è l'implementazione di un sistema a oggetti, nel quale ciascuno di essi è rappresentato da una vite. Ogni "oggetto vite" potrebbe essere rappresentato da un cluster di dati che contiene il nome della vite, i metri percorsi e la durata di utilizzo. In relazione a quest'ultima, si potrebbe implementare un sistema di notifica che, arrivati a un certo quantitativo di ore d'uso, ricordi all'operatore di lubrificare il dispositivo. Idealmente, l'utente deve essere in grado di aggiungere o eliminare gli elementi vite desiderati dal Front Panel del VI.

VI Pc Host Come già ampiamente argomentato all'interno della tesi, il VI PC fa da interfaccia grafica per l'utente comandando il VI Real-Time. Esso necessiterà di modifiche in funzione delle nuove funzionalità sviluppate codice.

Bibliografia

[1] Antonio Carlo Bertolino, *Banco Prova Viti a Ricircolo di Sfere - Capitolato Tecnico*, Dipartimento di Ingegneria Meccanica e Aerospaziale, Politecnico di Torino, 2019.

[2] Flaviana Calignano, *Macchine a Controllo Numerico - Struttura Meccanica*, Sistemi Integrati di Produzione, Slide del corso, Dipartimento di Ingegneria Gestionale e della Produzione, Politecnico di Torino, A.A. 2019/20.

[3] Emporio del cuscinetto S.r.l, *Viti a ricircolo di sfere e accessori*, Catalogo di prodotti, Padova, Italia.

[4] Edoardo Mercanti, *Banco prova per trasmissione vite/madrevite: progettazione del sistema di acquisizione dei dati.*, Tesi di Laurea Magistrale, Corso di Laurea Magistrale in Ingegneria Meccanica, Dipartimento di Ingegneria Meccanica e Aerospaziale, Politecnico di Torino, 2021.

[5] Nicola Ambrosino, *Implementazione di un servoattuatore elettromeccanico lineare a rulli planetari: progettazione dei software, modellazione e verifiche sperimentali.*, Tesi di Laurea Magistrale, Corso di Laurea Magistrale in Ingegneria Meccanica, Dipartimento di Ingegneria Meccanica e Aerospaziale, Politecnico di Torino, 2018.

[6] *Understanding Communication Options Between the Windows HMI, RT Processor, and FPGA*, Documentazione LabVIEW, National Instruments, 2022.

[7] *Do I Need a Real-Time System?*, Paper Online, National Instruments, 2022.

[8] *Building a Real-Time System With NI Hardware and Software*, Paper Online, National Instruments, 2022.

[9] *Virtual Instrument*, Enciclopedia Online, LabVIEW Wiki, 2020.

[10] *Local Variables*, Documentazione LabVIEW, National Instruments, 2022.

[11] *Global Variables*, Documentazione LabVIEW, National Instruments, 2022.

[12] *Using the LabVIEW Shared Variable*, Documentazione LabVIEW, National Instruments, 2022.

[13] *Single-process shared variable (SPSV)*, Guida Online, RIO Developer Essentials Guide for Academia, 2018.

[14] *What Is a Queue in LabVIEW?*, Supporto LabVIEW, National Instruments, 2022.

[15] *Queue Operations Functions*, Documentazione LabVIEW, National Instruments, 2022.

[16] *Communicating Data between Parallel Sections of Code Using Channel Wires*, Documentazione LabVIEW, National Instruments, 2022.

[17] *Introduction to LabVIEW Channel Wires*, Documentazione LabVIEW, National Instruments, 2022.

[18] *RT FIFO Functions*, Documentazione LabVIEW, National Instruments, 2022.

[19] *Real-Time FIFO Frequently Asked Questions*, Documentazione LabVIEW, National Instruments, 2022.

[20] *Lossless Communication with Network Streams: Components, Architecture, and Performance*, Paper Online, National Instruments, 2022.

[21] *FPGA Memory Items (FPGA Module)*, Documentazione LabVIEW, National Instruments, 2022.

[22] *Read (Memory Method)*, Documentazione LabVIEW, National Instruments, 2022.

[23] *Transferring Data between Devices or Structures Using FIFOs (FPGA Module)*, Documentazione LabVIEW, National Instruments, 2022.

[24] *Data Transfer Using FIFOs*, Documentazione LabVIEW, National Instruments, 2022.

[25] *How DMA Transfers Work (FPGA Module)*, Documentazione LabVIEW, National Instruments, 2022.

[26] *Elementi di OOP in C++*, Blog Personale, Claudio Maccherani, Perugia, 2016.

[27] *How Rotary Encoder Works and How To Use It with Arduino*, Tutorial Online, How To MECHATRONICS, 2020.

[28] *Working with File Paths on Real-Time Targets*, Tutorial Online, Support, National Instruments, 09/02/2021 (ultimo aggiornamento).


1859
Politecnico
di Torino