POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Meccanica

Tesi di Laurea Magistrale

Branco prova per trasmissione vite/madrevite: progettazione del sistema di acquisizione dati.

Relatori

Prof. Massimo SORLI

Ing. Antonio Carlo BERTOLINO

Ing. Matteo GAIDANO

Candidato

Edoardo MERCANTI

Dicembre 2021

Sommario

Negli ultimi anni, nel campo della ricerca aeronautica si è andati verso una direzione che coinvolgesse dei velivoli "più elettrici". In particolare, per quanto riguarda i sistemi di controllo volo, la volontà è quella di rimpiazzare le soluzioni tradizionali di tipo elettro-idrauliche con quelle di tipo elettro-meccanico. Tuttavia, a causa dei più alti tassi di malfunzionamento, quest'ultimo tipo di componenti ha faticato a diffondersi. Una possibile soluzione a cui si è pensato è quella di implementare delle robuste tecniche di PHM (Prognostic and Health Management), ovvero una serie di applicazioni specifiche dove un sistema o un componente viene supportato da un sistema di monitoraggio che ne controlla lo stato, la salute e fornisce dati utili sulla sua vite utile residua. Il seguente lavoro di tesi vuole descrivere il percorso intrapreso per progettare il sistema di acquisizione dati di un banco prova "hardware in the loop" (HiL) per l'analisi di un sistema vite-madrevite a ricircolo di sfere, allestito nel laboratorio di meccanica del Politecnico di Torino, in particolare lo studio del codice Labview che è stato scritto come parte centrale di questa tesi. L'idea che sta alla base delle tecniche HiL è quella di interfacciare al componente da analizzare, fisicamente disponibile e correttamente funzionante, un modello numerico, in grado di fornire gli input necessari al suo funzionamento. I comandi generati dal componente fisico sono invece forniti come input per il modello numerico in un collegamento ad anello chiuso. Un tipo di sistemi elettro-meccanici come quello preso in esame è uno dei più soggetti a malfunzionamenti, a causa della complessità che si ha nell'effettuare una sua modellazione accurata. In figura 1 viene riportato uno schema delle fasi di prognostica che si eseguono normalmente:

1. Fase offline: si utilizzano sia un modello ad alta fedeltà che il componente reale, su cui vengono inseriti dei difetti noti e , una volta misurati i dati, vengono estratti diversi indicatori. Successivamente si esegue una mappa del valore delle caratteristiche rispetto all'entità dei difetti.

2. Fase Online: il componente è funzionante e vengono estratte ed analizzate le caratteristiche precedentemente individuate nella fase offline, in modo da comprendere lo stato di salute del componente e la sua vita residua.

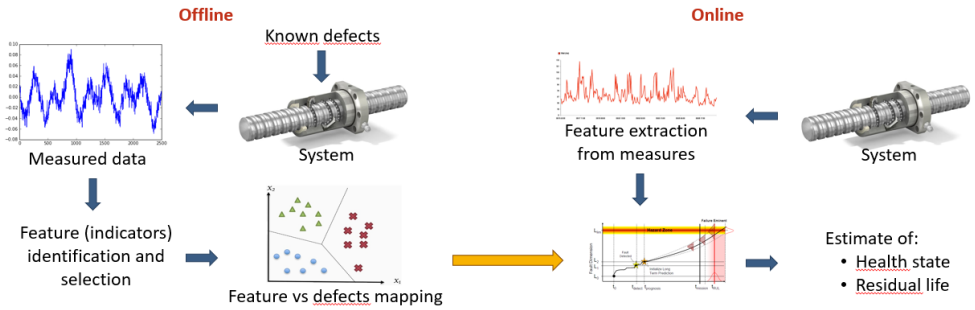


Figura 1: Schema illustrativo delle procedure di prognostica.

Nella prima parte di questa tesi viene presentato il banco prova e se ne descrivono i componenti, in particolare i sensori, che costituiscono la parte fondamentale per quanto riguarda lo sviluppo del codice Labview. Successivamente, si analizza il funzionamento del software utilizzato per acquisire i dati, ponendo l'attenzione sui meccanismi di trasmissione dei dati, sui vari hardware presenti e sul loro modo di comunicare.

Ringraziamenti

Porgo i miei più sentiti ringraziamenti alla mia famiglia, che mi ha sempre sostenuto nei percorsi di studio e non solo.

Ringrazio il Politecnico di Torino per avermi stimolato a tirare fuori sempre il meglio, il Professore Sorli per avermi dato la possibilità di intraprendere questo percorso di tesi, gli Ingegneri Bertolino e Gaidano per i loro preziosi consigli e per la loro disponibilità durante tutto il periodo di lavoro alla tesi.

Infine, un grazie agli amici che hanno alleggerito i miei anni universitari.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Descrizione del Banco Prova

1.1 Introduzione e Schema di Funzionamento

L'unità è mossa dal motoriduttore elettrico brushless (MR), che riceve gli input dal suo driver (APV). Un torsiometro è inserito nella connessione tra il MR albero di uscita e l'albero della vite a ricircolo di sfere (BS) a misurarne la coppia e l'angolo/velocità angolare. Questo sensore permette di separare gli effetti sul segnale

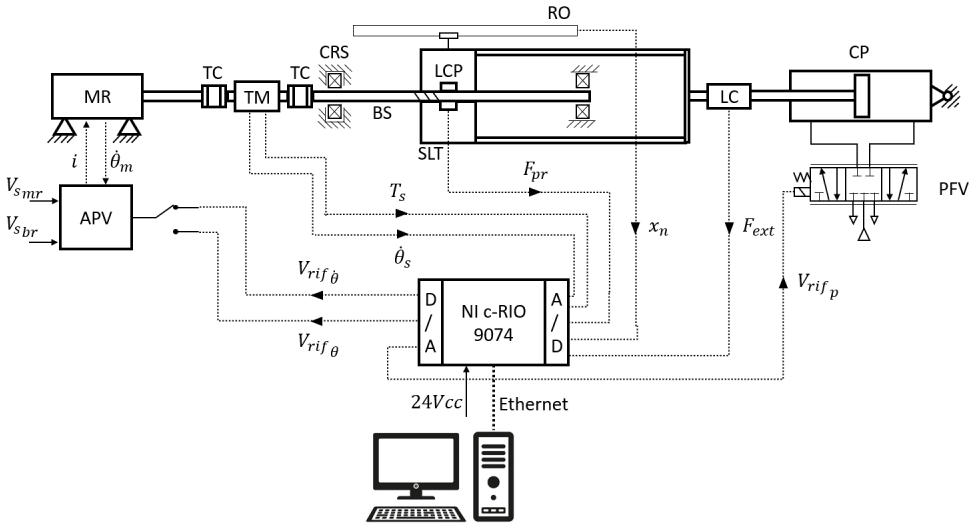


Figura 1.1: Schema Illustrativo Banco Prova

misurato causati dalla vite a ricircolo di sfere da quelle causate nei componenti a monte. Sono necessari giunti di coppia (TC) per compensare il possibile disallineamento per garantire una corretta misura della trasmissione coppia. L'albero della vite è supportato da due serie di cuscinetti in corrispondenza delle due estremità: quella lato motore (CRS) deve sopportare forze radiali e assiali quindi è composta da cuscinetti a sfera a scanalatura profonda. Per consentire deformazioni assiali della vite a ricircolo di sfere e per avere un sistema isostatico, dall'altra parte il cuscinetto non è vincolato assialmente ma deve sopportare solo carichi radiali. La rotazione delle chiocciole è impedita grazie a due cuscinetti paralleli a guide lineari. La slitta è composta da due dadi singoli e dal loro sistema di precarico, come schematicamente rappresentato in Fig. 3 Il sistema di precarico permette sia di precaricare le madreviti con una configurazione ad X (avvicinando i dadi) o con configurazione a O (spostando il dado ulteriormente) o lasciare i dadi senza precarico senza compensazione del gioco assiale interno. Questi due tipi di precarico si generano serrando i tappi di precarico, comprimendo l'apposito set di molle a disco. Ognuno dei due set di molle è composto da più molle Belleville in serie per avere maggiore precisione sulla generazione del precarico ottenuto mediante rotazione dei cappucci. Le molle creano una forza attrattiva/repulsiva tra il sottogruppo del dado 1, composto dalle traverse 1 e 3, e dal sottoassieme del dado 2. In quest'ultimo, tra le sorgenti e la traversa 2, viene inserita una cella di carico toroidale assiale per misurare dinamicamente il livello di precarico. Su una delle due madreviti (la MV1, o Master Nut) è applicato il carico esterno tramite un cilindro pneumatico, la cui

pressione è controllata da una valvola regolatrice di pressione proporzionale.

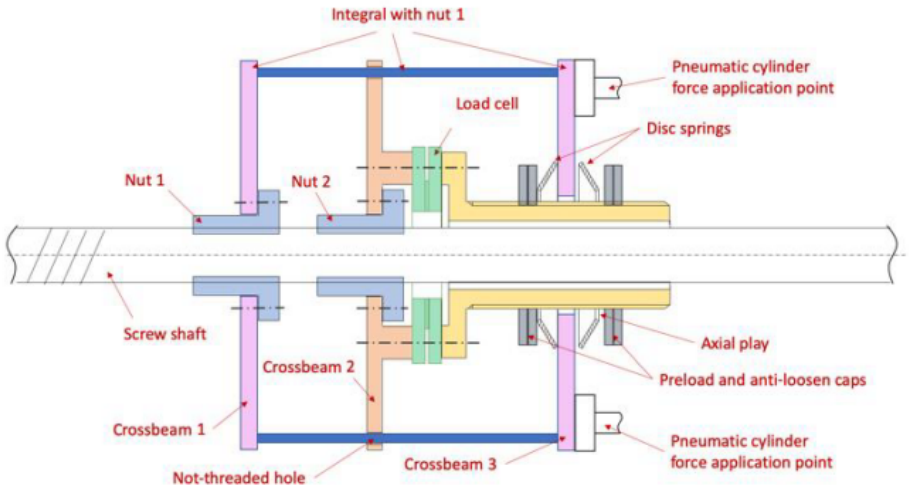


Figura 1.2: Slitta e sistema di Precarico

Questa valvola riceve un set di comando di pressione dall'hardware acquisizione dati e controllo. Quest'ultimo invia anche il set di posizione o velocità all'azionamento posizione e velocità del motoriduttore. Il sistema di acquisizione e controllo comunica con il PC. Fra motoriduttore e vite-madrevite è posizionato un torsiometro, interposto fra due giunti di coppia, per fornire i segnali di coppia, posizione e velocità angolari della vite. Una cella di carico (LCP) è posizionata sulla slitta per leggere il livello di precarico. Una riga ottica chiude l'anello chiuso di posizione e misura la posizione e velocità lineare della SLT. Una cella di carico misura la forza assiale applicata sulla MV1 dal cilindro pneumatico.

1.2 Componenti Meccanici

Si descrivono ora i principali componenti meccanici montati sul banco:

1.2.1 Motoriduttore

Si tratta di un motoriduttore della LENZE di tipo asincrono. Questo motoriduttore è di tipo epicicloidale sincrono, ha dalla sua una elevata precisione e dinamicità, presenta una buona uniformità di rotazione e permette la retroazione tramite resolver o encoder monocavo, per ottenere posizionamenti precisi. Il riduttore è di tipo epicicloidale ad uno stadio, ed è una soluzione adatta a numerose applicazioni servo. Le principali caratteristiche di questo motore sono una elevata precisione e flessibilità di utilizzo, un ingombro molto ridotto e una elevata accelerazione

angolare. In particolare, genera una coppia nominale all'uscita del riduttore di 12 Nm e una velocità nominale di 938 Nm, con un rapporto di riduzione pari a 4.

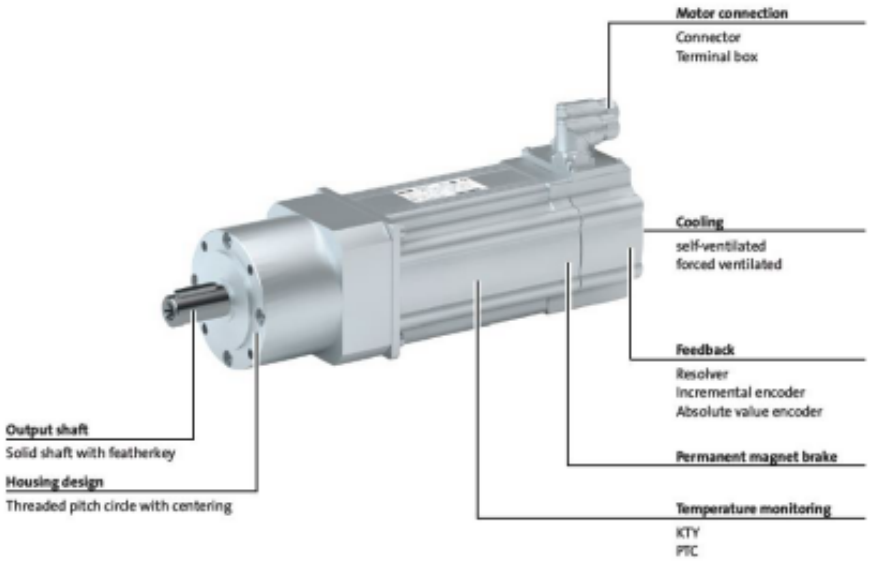


Figura 1.3: Motoriduttore della LENZE

1.2.2 Freno Integrato

Il freno è a magneti permanenti ed il suo scopo principale è di mantenimento di una posizione statica, ma, in casi di emergenza, può essere utilizzato anche in condizioni dinamiche, ma la coppia di frenatura è sensibilmente inferiore a quella nominale. Di conseguenza, non dovrebbe essere utilizzato come elemento di emergenza senza misure aggiuntive di sicurezza. Il freno è attivato quando l'alimentazione viene disconnessa. La velocità massima di commutazione è raggiunta con uno switching DC del voltaggio di alimentazione.

1.2.3 Vite a Ricircolo di Sfere

Si tratta di un tipo di vite, che viene così chiamata in quanto tra l'albero filettato e la madrevite, a filettatura concava, vengono inserite sfere di acciaio in un vano a spirale che si viene a formare, le quali hanno il compito di trasformare l'attrito radente in attrito volvente. La scanalatura elicoidale che viene ricavata ha una sezione emisferica avente lo stesso passo della vite ma diametro medio alquanto superiore. Quelle che si utilizzano in questo banco prova sono le S1N16-5M4 della UMBRAGROUP le quali hanno un diametro nominale di 16 mm, un passo di 5

Modello	MCS-09F38
Velocità nominale [rpm]	3750
Coppia di stallo [Nm]	4.2
Coppia nominale [Nm]	3.1
Coppia massima @ velocità > 75 rpm [Nm]	15
Potenza nominale [kW]	1.2
Corrente di stallo (standstill) [A]	3
Corrente nominale [A]	2.5
Corrente massima [A]	15
Voltaggio nominale [VAC]	330
Frequenza nominale [Hz]	250
Coppie polari	4
Rendimento percentuale	91
Momento d'inerzia (senza freno) [kgm^2]	$1.5 * 10^{-4}$
Costante di velocità @ 150°C [V/rad/s]	0.762
Costante di coppia @ 150°C [Nm/A]	1.4
Resistenza avvolgimenti a 20ř$C[\omega]$	5.2
Resistenza avvolgimenti a 150ř$C[\omega]$	7
Induttanza avvolgimenti [mH]	24.6
Velocità massima permissibile meccanica [rpm]	7000
Massa [kg]	5.20
Alimentazione [V]	3x400
Ventilazione	Naturale
Albero di uscita [mm]	14x30
Tolleranza albero	k7
Monitoraggio temperatura	1 x KTY 83-110 + 2 x PTC150°C

Tabella 1.1: Caratteristiche del solo Motore.

mm e un diametro delle sfere di 3.175 mm. Su ogni BS devono essere installate due MV singole non precaricate che verranno assemblate nella SLT per poter essere precaricate. Il loro funzionamento insieme a quello della slitta e a quello del sistema di precarico è stato spiegato in precedenza, nella parte introduttiva della descrizione del banco prova.

Tensione di alimentazione [V DC]	24
Coppia di frenatura @ 20°C [Nm]	8/12
Coppia di frenatura @ 120°C [Nm]	6/10
Corrente nominale [A]	0.65
Momento d'inerzia [kgm^2]	$1.07 * 10^{-4}$
Tempo di ingaggio [ms]	20
Tempo di disingaggio [ms]	40
Massa [kg]	0.8

Tabella 1.2: Caratteristiche del Freno Integrato.

Diametro nominale [mm]	16
Passo [mm]	5
Diametro sfere [mm]	3.175
Numero di circuiti	4
Diametro minimo [mm]	12.7
Dynamic Load Ca [kN]	14.1
Static Load Co [kN]	22.3

Tabella 1.3: Caratteristiche della Vite a Ricircolo di Sfere.

1.2.4 Giunti di Coppia

Sono dispositivi capaci di rendere solidali tra loro due estremità d'albero in modo tale che l'uno possa trasmettere un movimento torcente all'altro. Nel nostro caso i giunti consigliati dalla casa produttrice del torsiometro sono i BURSTER model 8690-5060-V4XX1, i quali accettano alberi con chiavetta secondo la DIN 6885. Questi giunti sono dotati di una rigidezza torsionale di 76000 Nm/rad e una coppia nominale di 60 Nm.

1.2.5 Molle a Tazza

Una molla a tazza è un tipo di molla a forma di rondella, la cui particolare forma tronco-conica dà infatti alla rondella le stesse caratteristiche di una molla. Devono essere tali da permetterci di ottenere la maggior precisione possibile nell'impostazione del precarico iniziale. Con riferimento alla Figura 1.2, si richiede che le molle a tazza abbiano una rigidezza tale da richiedere almeno due giri della relativa

Codice articolo	8690-5060-V4XX1
Diametro lato torsiometro [mm]	26
Coppia nominale [Nm]	60
Diametro altro lato [mm] 12 – 35	12-35
Coppia di serraggio [Nm]	40
Momento d'inerzia [kgm^2]	$3.2 * 10-4$
Rigidezza torsionale [Nm/rad]	76000
Maximum torque [Nm]	90

Tabella 1.4: Caratteristiche dei giunti di Coppia.

ghiera di manovra per garantire il raggiungimento del massimo del precarico sulla madrevite.

1.2.6 Cilindro Pneumatico di Carico

Il cilindro che si utilizza nel banco prova per applicare la forza esterna è della FESTO, modello DSBG a doppio effetto, con diametro di 160 mm. La sua corsa deve essere maggiore di quella della slitta, dettata dai fine corsa meccanici, per evitare che il cilindro pneumatico arrivi accidentalmente a fine corsa col pericolo di danneggiarsi sotto l'azione del resto del TB. La corsa è di 350 mm.

1.3 Sensori di Misura

[h] Sul banco prova sono montati molti sensori di diversa natura utilizzati per il controllo e la misura delle grandezze di interesse. Di seguito si riporta un elenco diviso per categoria.

1.3.1 Torsiometro

[h] Il sensore di coppia contactless scelto è della BURSTER modello 8661, e funziona secondo il principio dell'estensimetro. Grazie alla trasmissione induttiva e ottica dei segnali, il sensore è esente da manutenzione, i segnali sono digitalizzati direttamente sull'albero e resi disponibile dall'elettronica di valutazione come segnale di tensione o tramite USB. È inoltre possibile associare un encoder incrementale per la misura della velocità di rotazione, che verrà descritto in seguito. Le caratteristiche principali di questo sensore sono un range di coppia nominale misurabile di 50 Nm, un refresh rate di 1 kHz, un convertitore D/A interno a 16 bit e una tensione di output di

10 V. Il numero di incrementi è stato scelto in modo da poter rilevare il minimo gioco angolare definito in 2.3: usando un single-edge detection si ha una risoluzione di 0.18°, mentre usando un four-edge detection si arriva a 0.045°. Il sensore di coppia 8661 consiste essenzialmente di tre blocchi: il rotore, l'alloggiamento (contenente 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 elemento elastico 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. Può essere misurato usando degli estensimetri, che sono collegati in un circuito a ponte di Wheatstone. Un microprocessore condiziona il segnale proveniente dal ponte di Wheatstone 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 e lo indirizza all'elettronica di uscita, dove viene convertito in un segnale di uscita analogico 0 ... ± 10 V. Le caratteristiche complete sono riportate nella tabella seguente.

Codice articolo	8661-5050-V1402
Coppia nominale range 1 [Nm]	50
Coppia nominale range 2 [Nm]	5
Coppia massima [Nm]	75
Coppia di rottura [Nm]	150
Incrementi encoder impulsi a giro	2000
Tensione di alimentazione [V DC]	10-30
Tensione di output @ coppia nominale [V]	+/-10
Numero di bit convertitore D/A interno	16
Rigidezza [Nm/rad]	14000
Momento d'inerzia lato motore [$10-6kgm^2$]	85.7
Momento d'inerzia lato carico [$10-6kgm^2$]	33.3
Max carico assiale [N]	300
Max carico radiale [N]	125
Refresh rate [Hz]	1000

Tabella 1.5: Caratteristiche del Torsiometro.

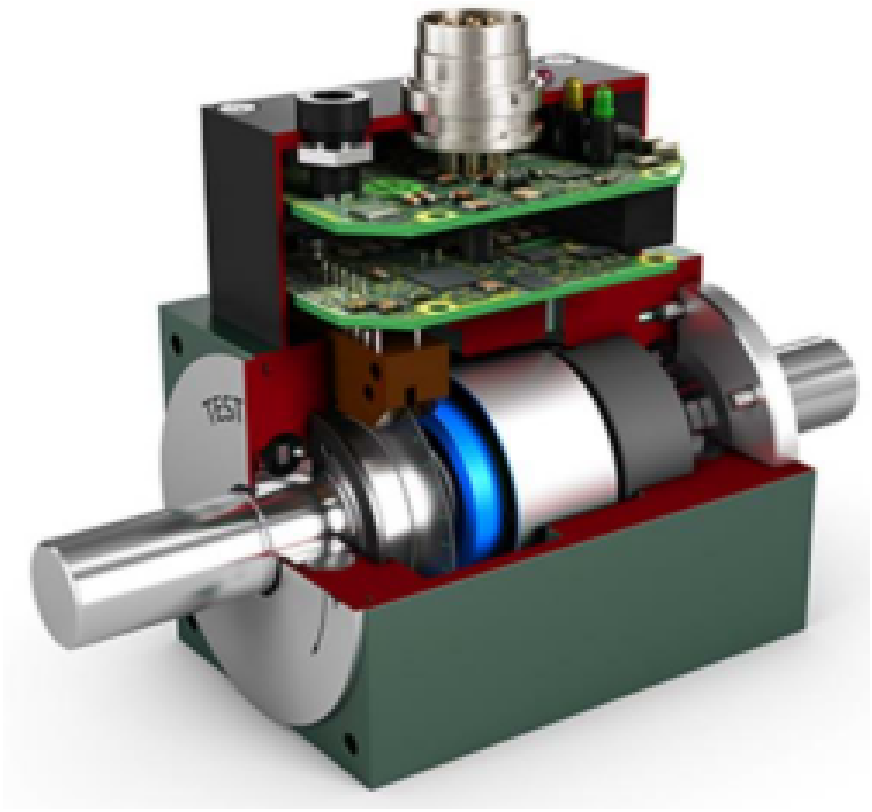


Figura 1.4: Sezione Torsiometro.

1.3.2 Cella di Carico per Forza Esterna

[h] Viene usata per misurare la forza scambiata tra la slitta e il cilindro pneumatico, è stata scelta una cella di carico trazione-compressione della HBM transducers, modello U9C. Il carico nominale è di 25 kN in quanto è il primo valore subito superiore al carico nominale applicato dal CP (12 kN).

1.3.3 Cella di Carico per Forza di Precarico

Il sensore che si è deciso di utilizzare in questo caso è la cella di carico K-1882 della Lorenz Messtechnik gmbh. È una cella di carico dual range per trazione e compressione, i due range di misura sono rispettivamente da 0 a 1 kN e da 0 a 10 kN. Le caratteristiche sono riportate in tab. 1.7

Codice articolo	K-U9C-20K0-03M0-Y-S
Carico nominale [kN]	20
Filettatura attacchi [kN]	M20x1.5
Sensibilità [mV/V]	1+/-1%*F.S.
Tensione di alimentazione massima [V]	5
Tensione di alimentazione consigliata [V]	10
Forza limite [% F_Nom]	>150
Forza di rottura [% F_Nom]	>400
Deflessione con carico nominale [mm]	0.09

Tabella 1.6: Caratteristiche della Cella di Carico per la Forza Esterna.

Codice articolo	104877
Forza nominale [kN]	1/10
Range di misura 1 [kN]	0-1
Range di misura 2 [kN]	1-10
Peso [kg]	3.9
Accuratezza [% F_nom]	0.2
Sensibilità [mV/V]	1.00+/-0.5%
Forza limite [kN]	15
Forza di rottura [kN]	>18
Deflessione con carico nominale [mm]	0.4

Tabella 1.7: Caratteristiche della Cella di Carico per la Forza di Precarico.

1.3.4 Riga Ottica

Si tratta di uno strumento molto utilizzato nell'ambito dell'automazione industriale come trasduttore digitale, si tratta di un sensore di posizione di tipo ottico che appartiene alla famiglia degli encoder. La riga ottica si compone di una parte fissa e di una parte mobile. La parte fissa è costituita da una riga di materiale trasparente (ad esempio vetro) che reca delle tacche serigrafate ad intervalli regolari. La parte mobile è invece formata da una sorgente luminosa, da una lente convergente, da una maschera forata di esplorazione e da un gruppo di fotocellule riceventi. La sorgente luminosa genera un fascio luminoso che passando attraverso la lente viene reso perpendicolare rispetto alla maschera forata. Parte della luce supera la maschera

(che ne aumenta il contrasto) e va a colpire la riga di vetro. Possono verificarsi due situazioni: se il fascio di luce colpisce la tacca serigrafata, la fotocellula non riceverà nulla (segnale 0); in caso contrario la fotocellula rileverà il fascio (segnale 1). L'insieme dei segnali delle fotocellule fornirà un'informazione circa la posizione esatta della parte mobile. È stata scelta una riga ottica della ELCIS encoder s.r.l., modello RV1846 famiglia L. La corsa utile è 320 mm e la risoluzione è di 1 µm. Di default sono presenti due tacche di zero all'inizio e alla fine della corsa utile, però essendo la corsa utile leggermente maggiore della corsa effettiva della SLT, queste non vengono raggiunte. È quindi opportuno chiedere l'inserimento aggiuntivo di una tacca di zero in mezzeria in modo da poter avviare la procedura di calibrazione all'inizio delle prove.

Codice articolo	L-RV1846-320-5-BZ-N-1
Tensione di Alimentazione [V]	5+/-5%
Risoluzione [μm]	1
Precisione [μm]	+/-5
Forza di avviamento [N]	<3
Frequenza massima [kHz]	50
Corsa utile [mm]	320
Lunghezza totale [mm]	425
Tolleranza tra fasi [°]	+/-45
Simmetria tra fasi [°]	+/-15

Tabella 1.8: Caratteristiche della Riga Ottica.

1.3.5 Trasduttori di pressione

Si è scelto di utilizzare il modello NAH 8253 della Trafag, questo tipo di sensori sono ampiamente utilizzati per la struttura piccola e per la loro versatilità, nel banco sono usati per monitorare l'andamento delle pressioni nelle camere del cilindro pneumatico.

1.3.6 Termocoppie

Sono strumenti di misurazione della temperatura estremamente utili e ampiamente utilizzati, comunemente utilizzati in un'ampia gamma di ambienti scientifici, industriali e ingegneristici. Le loro dimensioni ridotte e i tempi di risposta rapidi significano che possono essere messi al lavoro in tutti i tipi di ambienti pericolosi o

Figura 1.5: Sensore di Pressione.

impegnativi, pur fornendo la capacità di misurare rapidamente e accuratamente temperature estreme (ovunque nell'intervallo da 270 a 2.500 gradi Celsius, a seconda della loro configurazione specifica). Nonostante questa impressionante capacità, sono in realtà strumenti relativamente semplici che sono sia altamente robusti che estremamente convenienti. Diversi tipi di termocoppie, solitamente definite da lettere come J, K, L, N o T, offrono diversi estremi di queste caratteristiche chiave: alcune sono progettate utilizzando materiali particolari per resistere alle temperature più elevate e agli ambienti più difficili, mentre altre sono meno robuste , più economico da produrre e destinato all'uso in ambienti meno estremi. In questa guida, daremo un'occhiata più da vicino ai vari tipi di termocoppie disponibili sul mercato odierno e discuteremo alcuni dei loro potenziali usi. Nel nostro caso ne utilizziamo quattro di tipo K, per monitorare la temperatura delle madreviti, la temperatura ambiente all'interno del banco e la temperatura del motore.

1.4 Hardware di Controllo e Acquisizione

Di seguito viene riportato uno schema del sistema di acquisizione del banco:

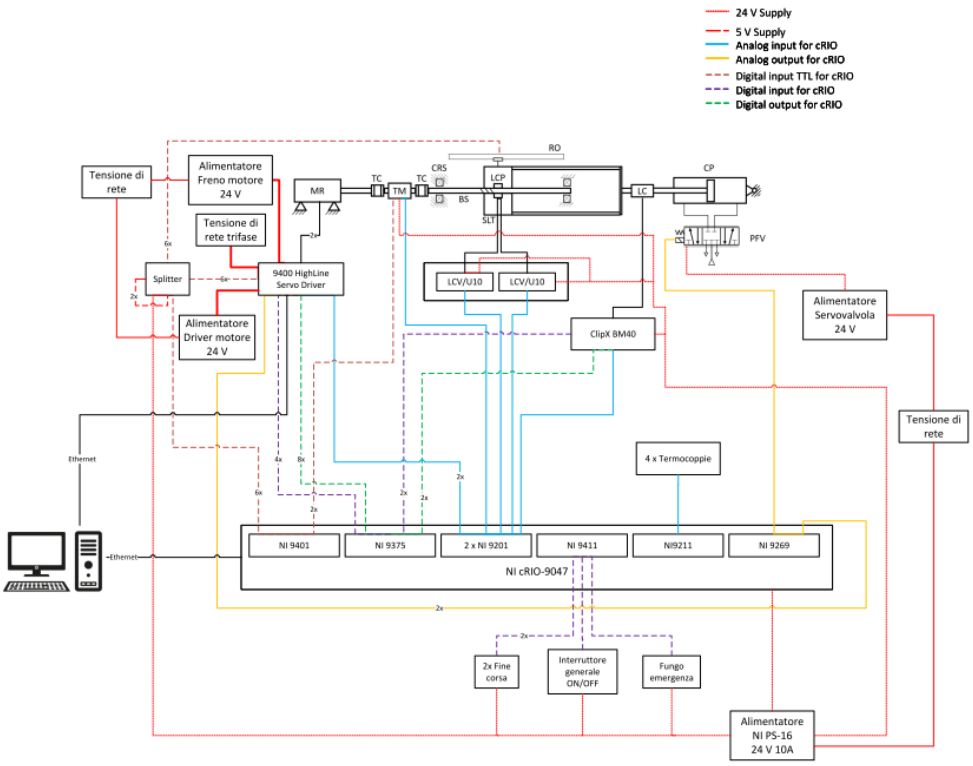


Figura 1.6: Schema di acquisizione del Banco.

1.4.1 National Instruments cRIO-9047

Il National Instruments c-RIO 9047 (fig. 1.7) svolge il duplice compito di controllo e acquisizione dati. I componenti da controllare in modo continuo sono il motoriduttore LENZE e l'attuatore pneumatico di carico. La generazione dei segnali di set verrà eseguita dallo stesso c- RIO; tuttavia, le storie temporali e si SET di posizione e di forza verranno fornite al c-RIO mediante un PC connesso ad esso con cavo Ethernet. I segnali da acquisire, provenienti dal campo, 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 c-RIO sono alloggiati al di sotto dei componenti meccanici, in una sorta di vano di acquisizione e controllo. Questo controller CompactRIO è dotato di un processore dual-core Intel Atom 1,60 GHz, un FPGA Xilinx Kintex-7 con memoria

DDR3 a 4GB e 8 slot per moduli I/O C Series per fornire un sistema di controllo e monitoraggio a prestazioni avanzate in un formato compatto, robusto, fanless, ideale anche per gli ambienti più difficili. Il controllore è dotato di sistema operativo NI Linux Real-Time che offre agli sviluppatori tutti i vantaggi del vasto ecosistema software di Linux. L'interfaccia utente embedded di NI Linux Real-Time permette l'implementazione di una HMI (human machine interface) locale per semplificare lo sviluppo delle applicazioni. È possibile utilizzare LabVIEW per la creazione, il debug e la distribuzione di logica su FPGA su scheda e sul processore NI Linux Real-Time OS. Un FPGA è, a tutti gli effetti, un circuito elettrico programmabile: la sua struttura, schematizzata in fig. 8, prevede una suddivisione reticolare che individua un numero ben definiti di blocchi, detti celle logiche (46080 nel caso del modello Spartan-3 2M). Tra un blocco e l'altro esistono delle interconnessioni riconfigurabili in base alle necessità (dipendono dai dati scambiati tra un blocco e l'altro). Esistono inoltre dei blocchi I/O che fungono da input per il circuito (es. acquisizione dati da un modulo) o da output (es. generazione di un segnale). Ciascun blocco logico svolge una funzione (es. somma, sottrazione, ecc.) assegnata in base al codice scritto dal programmatore: l'assegnazione di tale funzione e la scelta dei blocchi viene gestita autonomamente dal software LabVIEW in fase di "compilazione"; dunque, il programmatore può sviluppare il codice con le tipiche modalità previste dall'ambiente LabVIEW, con l'aggiunta di nuove funzioni relative esclusivamente all'FPGA (modulo LabVIEW FPGA). Tuttavia, per come è definito, un FPGA non può ospitare un codice che richieda un numero di celle logiche superiore a quelle disponibili; di conseguenza, occorre prestare attenzione durante lo sviluppo del codice scegliendo architetture che richiedano un minor uso di celle logiche rispetto ad altre. Un altro notevole vantaggio dell'FPGA è la possibilità di lavorare a frequenze notevoli (fino all'ordine dei MHz). Il passaggio da un blocco logico all'altro viene scandito dalla frequenza di clock: maggiore è la frequenza di clock, minore sarà il tempo richiesto per l'esecuzione del codice, essendo quest'ultimo dato dall'insieme dei singoli blocchi logici interconnessi.

1.4.2 Moduli I/O

Come si vede in fig. 1.7, al c-RIO sono collegati diversi moduli della National Instruments di seguito brevemente elencati:

- modulo NI 9375: è un modulo di I/O digitale da 24V con 16 canali di input e 16 canali di output. Tale modulo verrà utilizzato per gestire segnali di interfacce di sicurezza (interlock, etc.) con un tempo minimo di aggiornamento di 7 μs in ingresso e di 500 μs in uscita. Nel banco prova oggetto di questa tesi questo modulo viene usato per gestire i segnali digitali provenienti e diretti verso il driver del motoriduttore e verso il driver della cella di carico per la forza esterna.

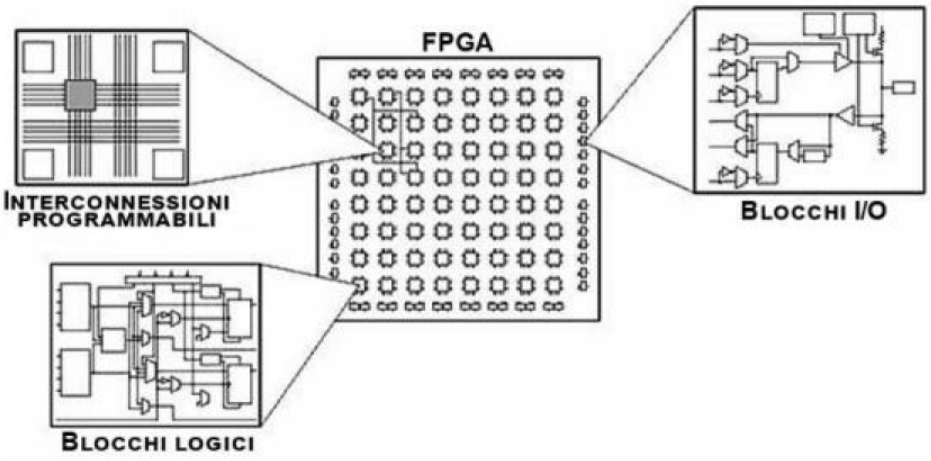


Figura 1.7: Circuito FPGA.

- modulo NI 9401: è un modulo di I/O digitale TTL (Transistor–transistor logic) da 8 canali che lavora su range di tensione nell'intorno di 0 V o 5 V. Viene utilizzato per l'acquisizione dei segnali provenienti dall'encoder angolare integrato nel torsiometro e la massima frequenza di aggiornamento per ciascun canale è di 9 MHz;
- modulo NI 9269 è un modulo isolato da canale a canale a quattro canali, 100 kS/s per canale per qualsiasi chassis NI CompactRIO e NI CompactDAQ. Simile al modulo NI 9263, NI 9269 aggiunge l'isolamento da canale a canale per una maggiore sicurezza, una migliore qualità del segnale e possibilità di impilare i canali per produrre fino a 40 V. Il supporto NI CompactDAQ è stato aggiunto a partire da NI-DAQmx versione 9.1 L'isolamento da canale a canale è comunemente necessario per le applicazioni che hanno più sistemi elettrici, come test automobilistici o applicazioni industriali che sono soggetti a maggiore rumore e spesso contengono più piani di terra.
- NI 9411: è un modulo C Series progettato per ingressi digitali a 6 canali, 500 ns differenziali/single-ended. Ciascun canale è compatibile con segnali da ± 5 V a 24 V. La NI 9411 funziona con livelli e segnali logici industriali per il collegamento diretto a un'ampia gamma di interruttori, trasduttori e dispositivi industriali. NI 9411 è un modulo digitale correlato, quindi può eseguire misurazioni, trigger e sincronizzazione correlati quando installato in uno chassis NI CompactDAQ.
- Il modulo di ingresso per termocoppia NI 9211 per l'utilizzo con CompactDAQ e NI CompactRIO lo chassis include un convertitore analogico-digitale

delta-sigma a 24 bit, filtri anti-aliasing, rilevamento di termocoppie aperte e compensazione della giunzione fredda per termocoppie ad alta precisione misurazioni. NI 9211 è dotato di calibrazione tracciabile NIST e messa a terra da canale a terra doppia barriera di isolamento per sicurezza, immunità ai disturbi e alta gamma di tensioni di modo comune.

- NI 9201 è un modulo di ingresso analogico per sistemi CompactDAQ e CompactRIO. Il NI 9201 fornisce otto canali di ingresso ±10 V con frequenza di campionamento di 500 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 (che non sono raffigurati nella figura sottostante).

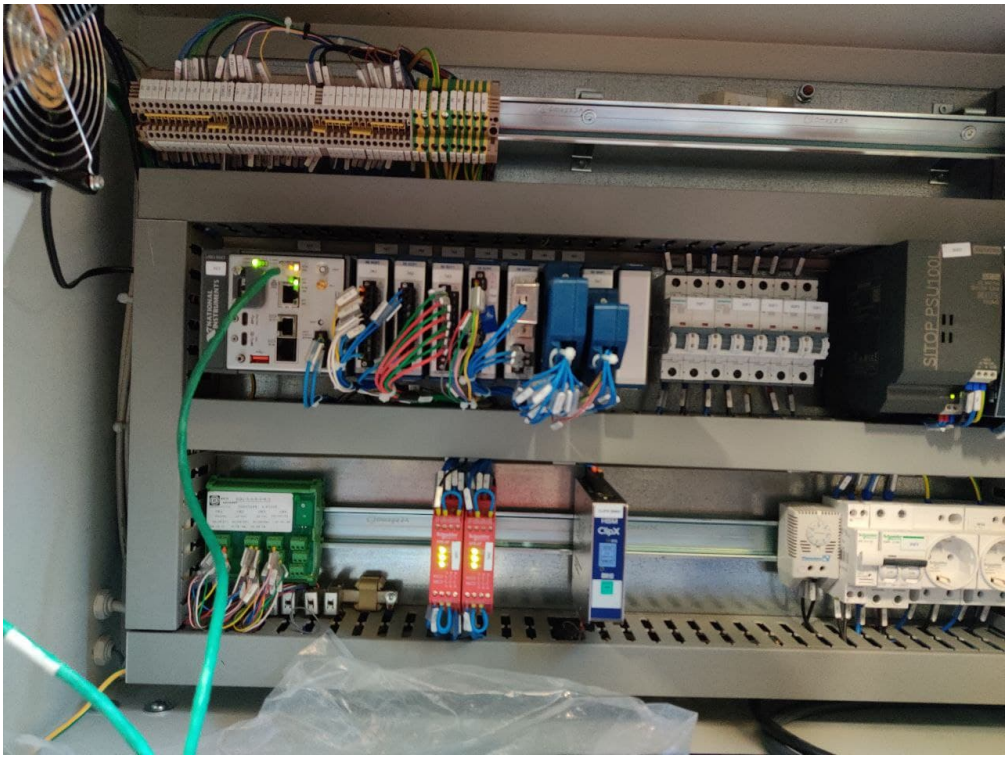


Figura 1.8: Foto del cRIO-9047, dello splitter della Riga Ottica e dei moduli di Sicurezza.

Capitolo 2

Grammatica del Codice Labview

2.1 Introduzione alla Logica di Programmazione

La parte centrale di questo lavoro di tesi è costituita dal codice scritto tramite il software NI Labview. LabVIEW component-oriented design (LCOD) è un approccio

proposto da Con-way e Watts che cerca di applicare idee dallo stile orientato agli oggetti al design delle applicazioni LabVIEW. Tuttavia, l'approccio è stato sviluppato prima di Lab-VIEW includeva tutte le funzionalità orientate agli oggetti. Tuttavia, riesce ancora incorporare queste idee nel design e può essere ampliato in un approccio orientato all'oggetto. Invece delle classi, gli elementi costitutivi principali di questo stile sono componenti che di fatto sono funzioni che elaborano alcuni argomenti di input per generare alcuni output, ma con alcuni requisiti aggiuntivi che saranno discusso in seguito. Prima di esplorare i dettagli di questo approccio progettuale, introduciamo brevemente LabVIEW e vediamo come si differenzia dagli altri linguaggi di programmazione. LabVIEW (Laboratory Virtual Instrument Engineering Workbench) è un ambiente di programmazione grafico costruito attorno al concetto di flusso di dati. I programmi in LabVIEW sono chiamati strumenti virtuali (VI) e consistono in un pannello frontale, lo schema a blocchi e il riquadro icone/connettori. Il pannello frontale rappresenta l'interfaccia utente e mostra i controlli e gli indicatori (input e output del programma) in cui alcuni indicatori possono visualizzare i dati in diversi modi come grafici e grafici. Il pannello dei blocchi costituisce il codice sorgente composto da controlli, indicatori, funzioni, strutture, subVI tutti collegati con fili che trasferiscono dati. L'icona/connettore viene utilizzato per modificare l'aspetto di un'icona VI e collegare i suoi terminali a controlli e indicatori. La Figura 10 mostra un esempio di VI con tutti i suoi elementi. A differenza dei linguaggi di programmazione tradizionali in cui ogni comando viene elaborato in sequenza, LabVIEW elaborerà più operazioni non appena ogni input a un blocco funzione o un subVI viene alimentato con i dati. Questo si presta naturalmente al principio del flusso di dati mentre seguiamo i dati da un blocco funzione all'altro. Con uno sforzo significativo potrebbe essere possibile deviare da questo principio, ma farlo è piuttosto impraticabile.

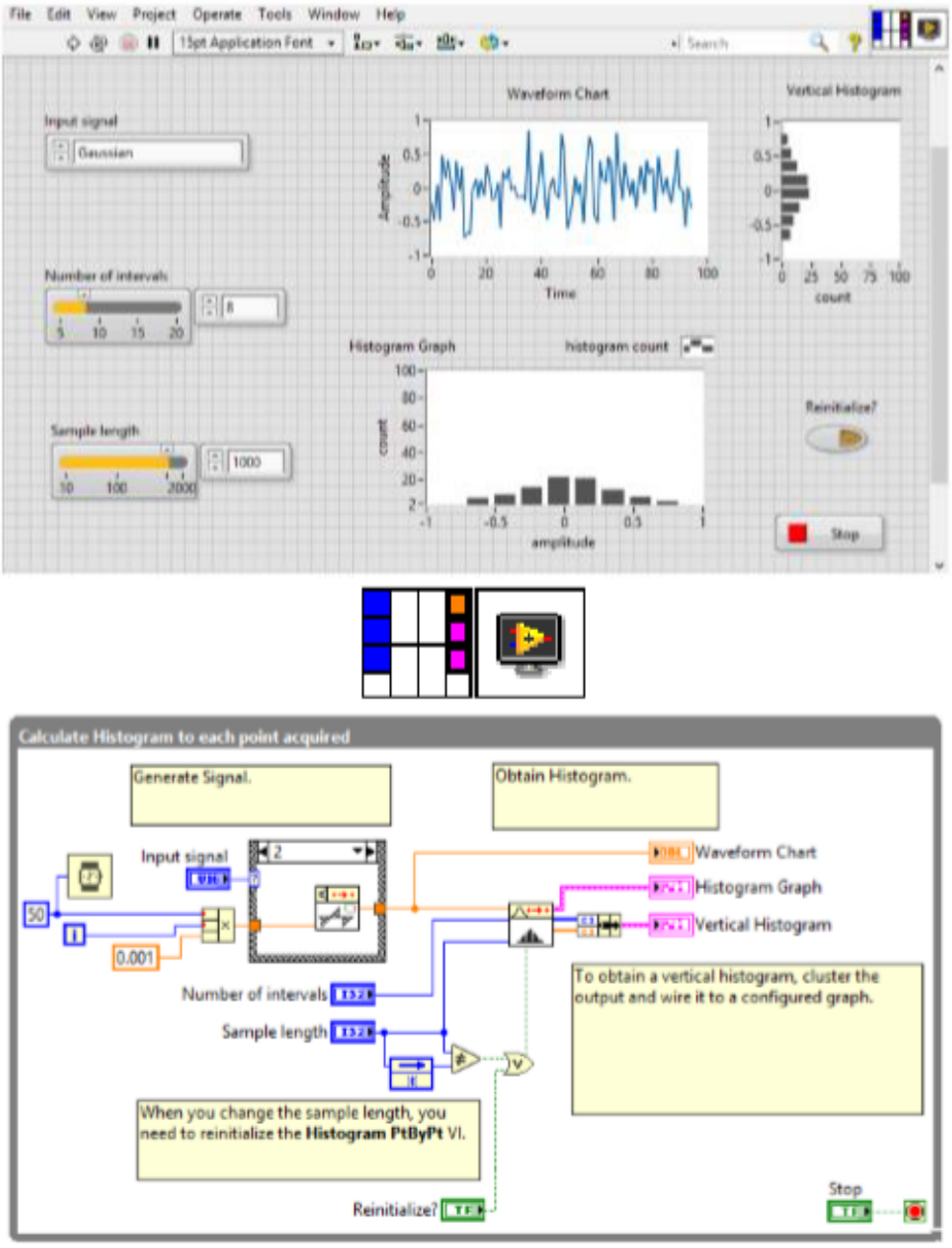


Figura 2.1: Front panel, Icon, connector pane e block diagram.

2.2 Caratteristiche del Codice

2.2.1 Accoppiamento e Coesione

L'accoppiamento e la coesione sono usati per descrivere l'effettiva modularità dei componenti e, come accennato in precedenza, miriamo a progettare componenti ad accoppiamento lasco con una forte coesione. I componenti strettamente accoppiati sono così strettamente intrecciati che diventa difficile distinguere dove finisce un componente e ne inizia un altro. Gran parte della funzionalità è distribuita attraverso lo stesso gruppo di componenti, rendendo difficile capire come viene realizzata una determinata funzione. Questo fenomeno è descritto con il termine spaghetti code dove i fili confusi delle operazioni assomigliano a una ciotola di spaghetti. Sebbene il termine preceda l'esistenza di LabVIEW, la sua natura grafica rende facile vedere quando un programma inizia a seguire quella strada mentre il programma diventa visivamente sempre più disordinato. Un altro buon indicatore di accoppiamento stretto è mostrato nella Figura 11, dove il numero di fili in entrata e in uscita dal VI è eccessivo. Quando un componente è fortemente coeso e nasconde una buona informazione, generalmente non ha bisogno di più di quattro o cinque input e una quantità simile di output. Un buon esempio di componente debolmente accoppiato è mostrato nella Figura 12. Il componente viene utilizzato per eseguire diverse misurazioni del sensore e ha un'interfaccia molto semplice.

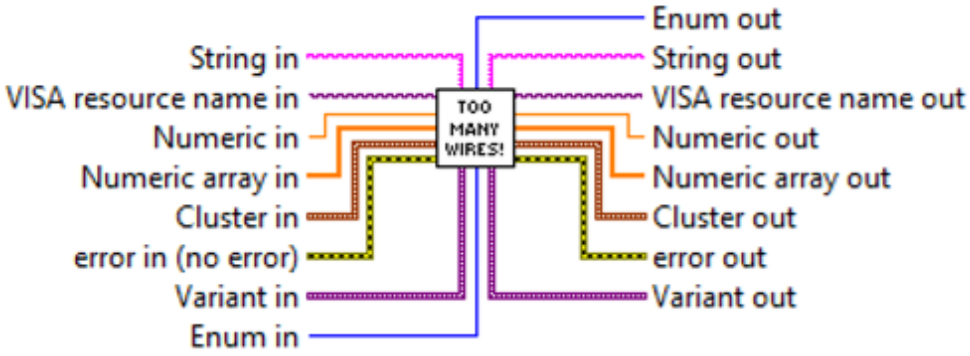


Figura 2.2: Esempio di VI con troppi input e output.

2.2.2 Nascondere e Incapsulare Informazioni

L'idea alla base dell'occultamento delle informazioni è quella di nascondere qualsiasi decisione di progettazione che potrebbe cambiare. Questo viene eseguito attraverso il processo di incapsulamento in cui queste decisioni sensibili vengono catturate come segreti del modulo o del componente. Per esplorare il concetto di occultamento

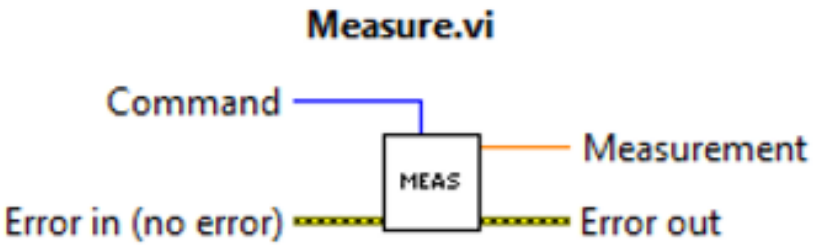


Figura 2.3: Esempio di VI illeggibile.

delle informazioni, consideriamo un esempio di un sistema di casa intelligente che realizza le sue funzionalità tramite un telecomando. Il controller è posizionato vicino a una posizione centrale vicino ai dispositivi desiderati ed è gestito da un computer centrale tramite una connessione TCP. Il controller ha un numero di pin di uscita digitale che possono essere utilizzati per accendere e spegnere i dispositivi. L'attuale implementazione utilizza questi pin per accendere un relè e spegnere una luce (per maggiore confusione la luce usa la logica invertita, TRUE = spento, FALSE = acceso). Il relè viene utilizzato per commutare l'alimentazione a un riscaldatore in una stanza, mentre la luce illumina quella stessa stanza. Per ottenere questa funzionalità un progettista potrebbe offrire una soluzione mostrata nella Figura 12 in cui ogni pin è controllato direttamente. A prima vista, questo non sembra male, ma moltiplicare questo problema per molte stanze con più dispositivi e l'implementazione diventa incredibilmente difficile da mantenere. Ogni volta che si verificano problemi o il layout dei pin viene aggiornato, lo sviluppatore deve esaminare il codice per applicare le modifiche appropriate. Inoltre, quando si apportano modifiche ai pin, si può facilmente commettere un errore poiché i dispositivi possono avere una logica diversa, alcune parti dell'applicazione con quei pin potrebbero essere perse e può essere difficile determinare a quale stanza e dispositivo corrisponde un determinato pin. Una soluzione molto migliore sarebbe

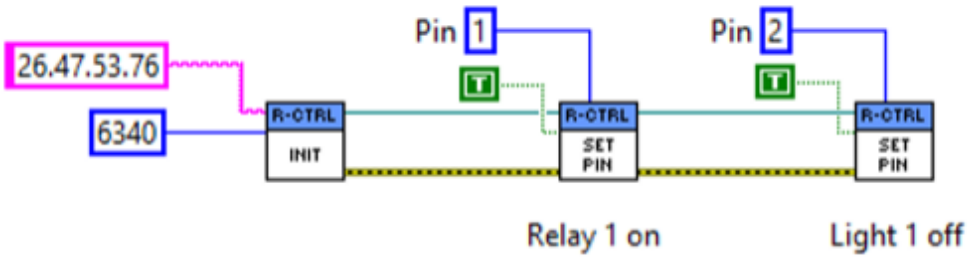


Figura 2.4: Esempio di Informazioni nascoste male.

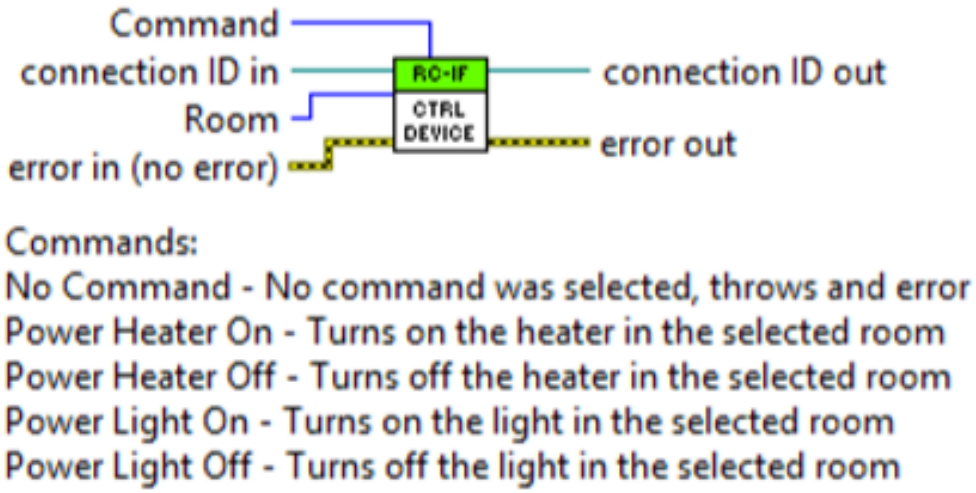


Figura 2.5: Componente per il controllo del dispositivo.

quella di creare un componente che fornisca tutte le funzionalità necessarie per controllare ciascun dispositivo, come mostrato nella Figura 14. Il componente stesso può determinare come gestisce l'inizializzazione e fornisce comandi per ogni dispositivo in uso in un'unica posizione centralizzata. In questo modo è facile capire a cosa è assegnato ciascun pin, aggiungere comandi per nuovi dispositivi, modificare vecchi comandi e persino modificare l'intera interfaccia o protocollo di comunicazione. Nell'implementazione riprogettata, mostrata nella Figura 2.5, i pin e la logica utilizzati sono incapsulati come un segreto del componente. Già la precedente implementazione aveva i dettagli della comunicazione incapsulati in quanto non sono rilevanti qui.

2.2.3 Astrazione

L'uso delle astrazioni, come accennato in precedenza, è uno dei modi principali di affrontare la complessità. Come Conway et al. nota, ci sono due tipi di astrazioni nella progettazione del software:

- Astrazione funzionale: un programma potrebbe essere progettato in termini di procedure, in cui le funzioni di alto livello contengono molte sottofunzioni e attraverso queste realizzano un comportamento più complesso. Questo tipo di astrazione è supportato in LabVIEW tramite l'uso di subVI.

- Astrazione dati: Attraverso questo tipo di astrazione un utente può creare il proprio tipo di dati e definirne il funzionamento. Un programma è quindi composto da molti oggetti appartenenti a questi tipi di dati la cui interazione

realizza il comportamento complesso del sistema. Questo tipo di astrazione viene utilizzato nella programmazione orientata agli oggetti ed è stato discusso in precedenza. Quando è stato sviluppato l'approccio orientato ai componenti, LabVIEW supportava solo l'astrazione dei dati tramite cluster che sono tipi di dati personalizzati che non contengono il comportamento. Da allora LabVIEW ha aggiunto il supporto nativo orientato agli oggetti, il cui impatto verrà discusso in una sezione successiva.

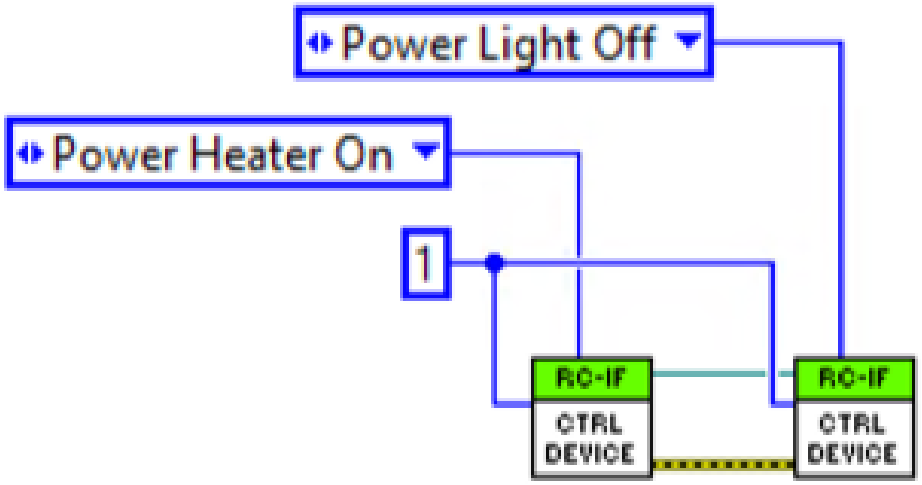


Figura 2.6: Informazioni ben Nascoste.

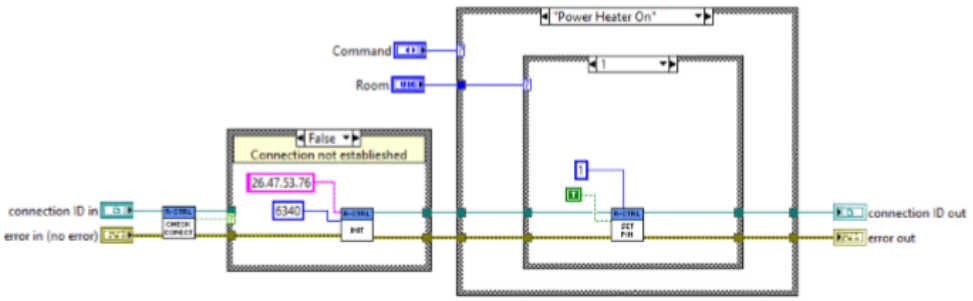


Figura 2.7: Implementazione del dispositivo di controllo.

Combinando i concetti di occultamento e incapsulamento dei dati con l'astrazione funzionale in LabVIEW, LCOD è stato in grado di portare le astrazioni a livelli simili di astrazioni orientate agli oggetti prima che il supporto nativo per le

funzionalità orientate agli oggetti fosse introdotto in LabVIEW. Il componente Control Device può essere utilizzato come esempio di un'astrazione di alto livello. La sua implementazione può essere realizzata tramite funzioni di livello inferiore come mostrato nella Figura 2.6.

2.2.4 Componente

I componenti sono gli elementi di base di LCOD e incorporano idee di coesione, accoppiamento e occultamento delle informazioni. Per definire cosa costituisce un componente Conway et al. fornire la seguente serie di requisiti.

- Un componente deve fornire una chiara specifica di quali servizi è pronto ad offrire. • L'unico modo in cui un componente può interagire con altri componenti e il resto dell'applicazione è attraverso le sue interfacce dichiarate. Deve incapsulare tutti i suoi dati e processi dietro questa interfaccia.
- Il componente dovrebbe essere sufficientemente indipendente da essere testato isolatamente.
- I componenti e il software che utilizzano un componente devono fare affidamento solo sulle interfacce definite e sulle operazioni specificate.

Un componente è costituito da VI che attraverso una semplice interfaccia fornisce un servizio o servizi. È caratterizzato da una forte coesione fornita dalla raccolta di VI, dall'occultamento delle informazioni nascondendo le funzioni dei suoi subVI dal mondo esterno e dall'accoppiamento lasco fornendo al resto del sistema un'interfaccia semplicistica per realizzare i suoi servizi. Un'interfaccia tipica di un componente è illustrata nella Figura 17. I messaggi vengono utilizzati per azionare il componente che può opzionalmente prendere ulteriori ingressi e fornire uscite.

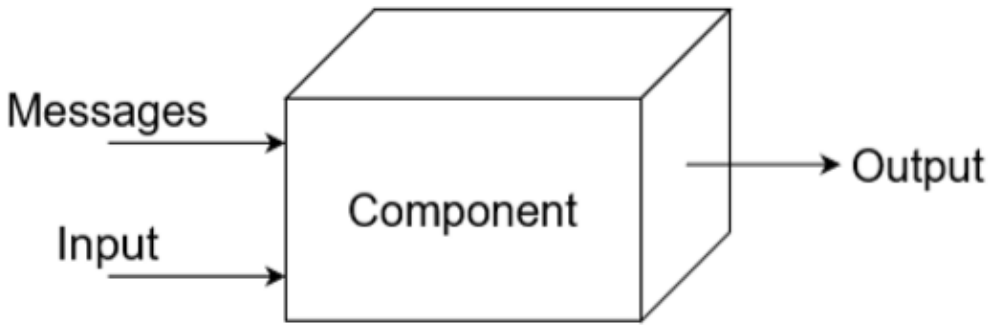


Figura 2.8: Esempio di Componente.

2.2.5 Implementazione

- Tutti i componenti, le funzioni pubbliche ei dati dovrebbero essere accessibili tramite un'interfaccia semplice.
- Dovremmo essere in grado di aggiungere/eliminare/modificare le azioni del nostro componente in modo semplice.
- Qualsiasi modifica dovrebbe avere scarso effetto sulla progettazione complessiva del software.
- Il componente memorizza il proprio stato localmente e in modo permanente.
- Il componente dovrebbe inizializzarsi.
- Gli errori vengono gestiti dal componente.
- Gli input e gli output dovrebbero essere verificati da soli.

2.2.6 Inivo di Messaggi

I componenti sono controllati tramite l'invio di messaggi, in cui un componente invia un messaggio che richiede l'esecuzione di un'operazione. È possibile utilizzare vari metodi per raggiungere l'obiettivo, ma i tipi enumerati sono i più utili. Consentono a un utente di inserire semplicemente un componente nel proprio programma, fare clic con il tasto destro del mouse sull'input e ottenere un riferimento completo delle possibili operazioni. Il tipo enumerato rappresenta un elenco di interi senza segno, ciascuno con un'etichetta allegata. I tipi enumerati sono presentati sul pannello frontale come una casella a discesa in cui l'utente può selezionare una delle opzioni testuali. Ciò offre il vantaggio di presentare chiaramente i comandi all'utente, mentre i confronti dietro le quinte sono facilitati e possono essere applicate operazioni aritmetiche. Le operazioni aritmetiche trattano semplicemente il tipo enumerato come un numero, mentre le funzioni Incremento e Decremento ruotano ulteriormente all'inizio e alla fine dell'elenco (ovvero, l'incremento dell'ultima enumerazione fornisce la prima enumerazione dell'elenco). Il risultato di un'operazione viene convertito nell'enumerazione più vicina e se un numero è fuori intervallo viene impostato sull'ultima enumerazione.

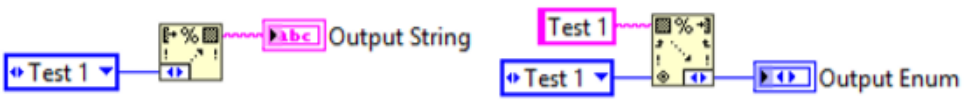


Figura 2.9: Lavorare con Enumeratori.

Oltre alle operazioni numeriche, l'etichetta di un tipo enumerato può essere estratta e utilizzata con operazioni sulle stringhe. L'estrazione dell'etichetta viene eseguita con il VI Format Into String mostrato nella Figura 2.9a e può essere utilizzata per memorizzare gli stati enumerati in un file o database. Questi stati possono essere recuperati in seguito scansionando la stringa usando il VI Scan From String come mostrato nella Figura 2.9b. Queste operazioni sono utili anche quando si inviano enumerazioni in remoto poiché le enumerazioni non possono essere inviate direttamente. Un altro strumento utile che può essere utilizzato in combinazione con i tipi enumerati è la case structure. Questa struttura, mostrata nella Figura 2.10, consente allo sviluppatore di implementare funzionalità diverse a seconda dell'enumerazione selezionata. Per i componenti, la case structure rende facile vedere dove viene elaborato ogni comando, migliorando così la leggibilità. La case structure può anche essere combinata con un ciclo while per creare una macchina a stati. Il registro a scorrimento del ciclo while memorizza lo stato precedente e la case structure reagisce allo stato in ogni iterazione. Un esempio di macchina a stati è mostrato nella Figura 2.11. Sebbene le case structure e le macchine a stati

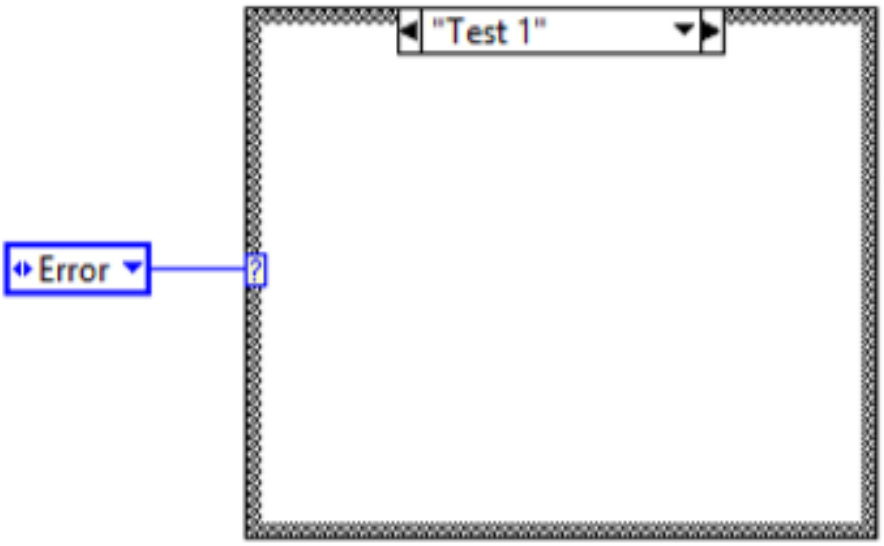


Figura 2.10: Case Structure.

siano molto utili, l'aggiunta di un nuovo comando al tipo enumerato interromperà la case structure poiché tratta questa modifica come un tipo completamente nuovo. Per evitare questo problema, è importante salvare il nostro controllo enumerato e utilizzare una definizione di tipo rigorosa. Una definizione del tipo rigorosa garantisce che qualsiasi modifica al tipo salvato si propaghi a tutti i componenti e le strutture che utilizzano quel tipo. Con l'uso del tipo rigoroso i componenti

sviluppati sono più robusti ed è facile applicarvi modifiche.

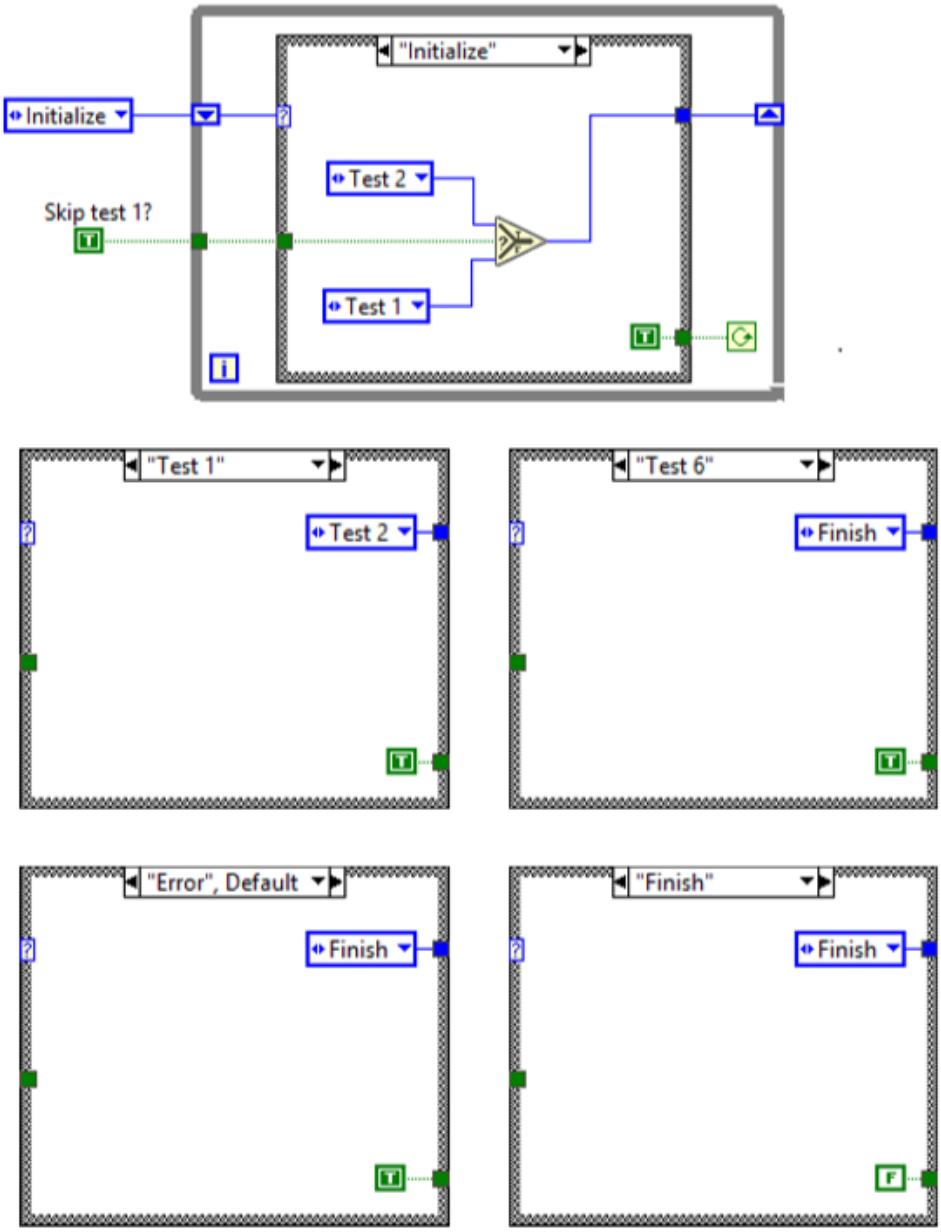


Figura 2.11: Esempio di Macchina a Stati.

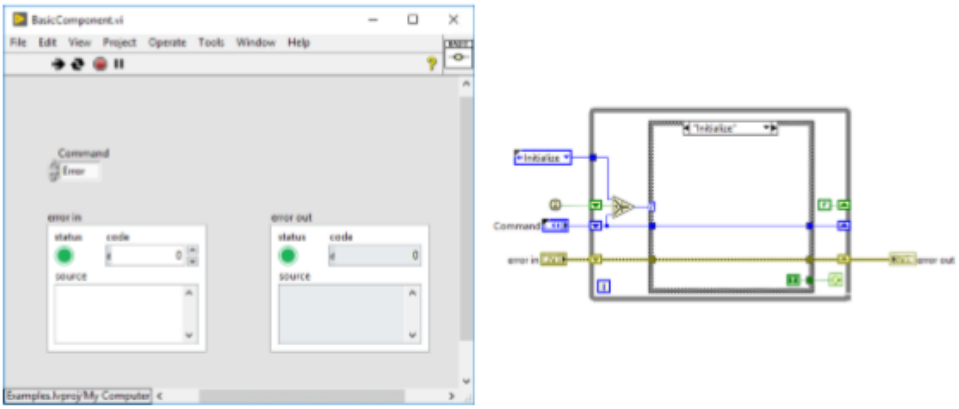


Figura 2.12: Componente di Base.

2.2.7 Archiviazione Locale Persistente

Un altro attributo utile per un componente sarebbe la capacità di conservare i dati internamente, anche dopo la conclusione di un'operazione. Con questa capacità un componente si avvicinerebbe all'idea di un oggetto poiché ora il componente memorizzerà il proprio stato oltre alla capacità di esibire il comportamento. Ciò può essere ottenuto in LabVIEW con l'uso di registri a scorrimento di un ciclo for/while. I registri a scorrimento sono variabili locali utilizzate principalmente per memorizzare i dati tra le iterazioni del ciclo, ma hanno anche un'importante caratteristica di persistenza. I dati detenuti all'interno di tali registri sono conservati anche a VI conclusa esecuzione. Pertanto, i registri a scorrimento danno indipendenza ai componenti, consentendo loro di contenere informazioni private nascoste all'esterno. Questo è essenziale per ottenere un accoppiamento lasco e nascondere le informazioni.

2.2.8 Struttura base di un Componente

Per dimostrare come dovrebbero essere realizzati i requisiti dei componenti menzionati all'inizio della Sezione di implementazione, diamo un'occhiata al tipo più elementare. Il componente più semplice, mostrato nella Figura 2.12, riceve un comando e lo elabora internamente senza bisogno di input aggiuntivi o di fornire output. Internamente è realizzato come una macchina a stati con capacità di autoinizializzazione.

- Auto inizializzazione: Per assicurarsi che l'inizializzazione venga eseguita solo una volta la Prima Chiamata? è possibile utilizzare la funzione. Viene visualizzato con una piccola icona circolare e quando il subVI viene chiamato

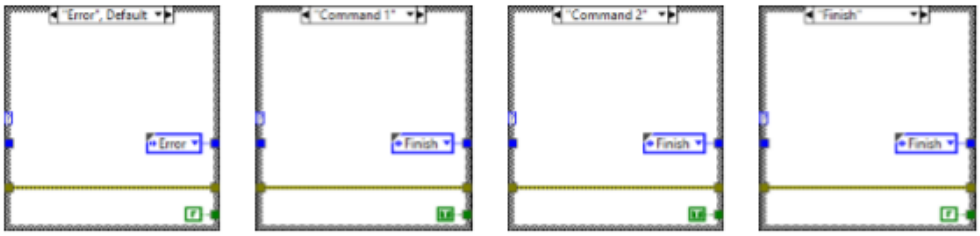


Figura 2.13: Stati di un Componente

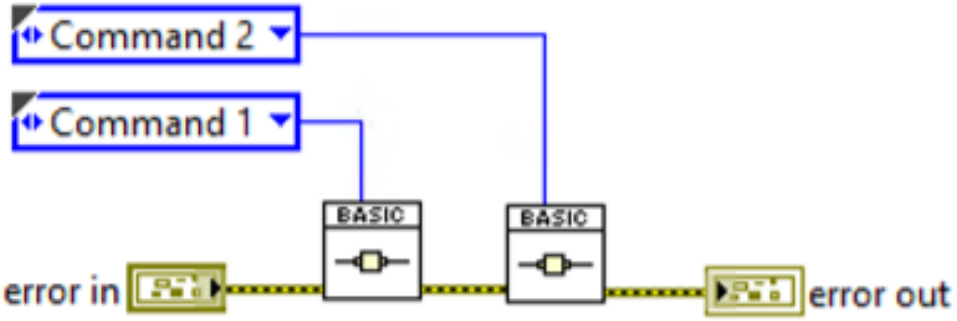


Figura 2.14: Interfaccia di un Componente di Base.

per la prima volta da qualsiasi altro VI emette TRUE, innescando così l'inizializzazione mostrata nella Figura 2.12. In ogni altra chiamata la funzione restituisce un output FALSE e solo il comando viene elaborato.

- Gestione degli errori e verifica input/output: Questo modello di base in tutti i casi finirà e uscirà o semplicemente uscirà come mostrato nella Figura 2.13. Se il componente per qualsiasi motivo ricevesse un comando errato o si verificasse un problema durante l'elaborazione, il componente dovrebbe generare un errore. Questo semplice esempio propaga semplicemente l'errore in avanti. Un componente stesso può essere utilizzato per gestire gli errori in un caso separato o come parte del caso Fine. Tuttavia, va notato che ci sono errori critici che devono essere propagati e interrompere l'esecuzione dell'applicazione. Il Fine viene utilizzato per il controllo post-condizione e deve assicurarsi che l'operazione sia stata gestita correttamente e che tutti i dati di output siano nel formato e nell'intervallo corretti. Il precondizionamento verifica la validità dei dati di input. Può essere gestito in un caso separato chiamato Start e fatto eseguire prima di ogni comando.
- Modifica delle azioni del componente e dell'effetto delle modifiche: L'aggiunta di una nuova funzionalità implica semplicemente l'aggiunta del nome della

funzione al tipo enumerato e l'aggiunta di un nuovo caso. Quindi la nuova implementazione può essere effettuata all'interno dello stato. La rimozione di una funzione richiede l'eliminazione del caso e quindi la rimozione della funzione dal tipo enumerato. Le modifiche vengono applicate modificando l'implementazione nel caso desiderato. Tutti questi cambiamenti hanno scarso effetto sul design generale poiché gli altri casi non dovrebbero essere interessati.

- Accessibilità attraverso una semplice interfaccia L'interfaccia pubblica del componente è mostrata nella Figura 2.14. L'interfaccia è molto semplice in quanto il componente non ha ingressi o uscite reali. Indipendentemente da ciò, è possibile accedere a tutte le funzioni semplicemente generando una costante enumerata rigorosamente tipizzata sull'ingresso di comando, visualizzando tutte le funzionalità disponibili.

- Archiviazione locale e persistente Poiché questo esempio è molto semplice, non è necessario archiviare alcuna variabile. Un componente che necessita di archiviazione avrebbe uno o più registri a scorrimento legati ai dati creati durante il processo di inizializzazione.

2.2.9 Error Handling

La gestione degli errori può essere uno degli aspetti più impegnativi, ma spesso è l'aspetto più ignorato della progettazione del software. Deve essere considerato in modo diverso durante due fasi di un progetto: la fase di sviluppo e il rilascio. Durante lo sviluppo l'applicazione dovrebbe lamentarsi ad alta voce nel momento in cui si verifica un errore. L'intento qui è quello di attirare l'attenzione dello sviluppatore in modo che l'errore possa essere trovato e risolto. Durante il rilascio, gli errori devono essere gestiti in modo molto più elegante. Ad esempio, è importante tenere un registro degli errori al momento del rilascio. In secondo luogo, lo sviluppatore deve considerare la propagazione e la gestione degli errori. LabVIEW utilizza il cluster di errore per connettere i VI in cui qualsiasi errore che si verifica viene passato a monte al VI successivo nella sequenza. Ciò solleva questioni di coesione poiché il prossimo VI potrebbe non aver bisogno di conoscere gli errori del suo predecessore e non dovrebbe essere responsabile della loro gestione. L'approccio standard alla gestione degli errori è attraverso il flusso di dati, come mostrato nella Figura 2.15. Un errore di immissione del VI numero due passerebbe senza che il VI completi la sua funzione. Lo stesso si ripete per i VI successivi (numero da 3 a 6). Quando un errore viene deliberatamente passato attraverso la linea di errore, l'idea è di interrompere l'operazione di qualsiasi VI attivo, perché si è verificato un errore critico. Tuttavia, se sono in esecuzione thread di elaborazione parallela, questo metodo non riuscirà a chiuderli. Pertanto, il metodo di gestione degli errori

del flusso di dati può portare a risultati imprevisti in applicazioni con elaborazione parallela.

Figura 2.15: Passaggio attraverso l'error handling.

Lo sforzo di ottenere un accoppiamento lasco tende a rendere i componenti più autonomi rispetto ai VI realizzati con la mentalità del flusso di dati standard. Anche la gestione degli errori dovrebbe riflettere questo ed essere contenuta il più possibile all'interno di un componente. Ad esempio, lo sviluppatore può implementare un componente Error Handler che può registrare un errore o aprire una finestra di dialogo. Questo componente può quindi essere utilizzato all'interno di altri componenti o alla fine di una sequenza con grande flessibilità. Un'altra cosa da considerare è che alcune aree di un'applicazione sono più soggette a guasti di altre. Queste aree ad alto rischio sono solitamente coinvolte in alcune interazioni del mondo reale. Concentrare gli sforzi di intrappolamento degli errori in queste aree può aiutare a catturare la maggior parte dei problemi. Pertanto, lo sviluppatore dovrebbe prestare particolare attenzione quando si tratta di parti del codice che coinvolgono: interazione con il database, interazione con file, interazione hardware (soprattutto per quanto riguarda il controllo), stampa e input dell'utente. Infine, occorre tenere conto della gravità dell'errore. Gli errori possono essere semplici avvisi, eccezioni che devono essere gestite o errori irreversibili in cui la chiusura dell'applicazione è l'unica opzione. Quando un componente raggiunge uno stato dal quale non può essere ripristinato, potrebbe dichiararsi in uno stato di errore. Questo stato non significa che l'applicazione deve essere abbattuta. Invece un altro componente di livello superiore potrebbe essere in grado di gestire quell'errore e riportare il componente di livello inferiore in uno stato di esecuzione normale. Pertanto, la gestione degli errori può essere gestita su più livelli dell'applicazione e lo sviluppatore deve valutare attentamente quale risposta merita ogni errore.

2.2.10 Stili e Standard

Il codice sorgente di un progetto comunica il design. Gli standard applicati aiutano a facilitare questa comunicazione e rendono il processo molto più semplice. Gli standard applicano una comprensione comune tra gli sviluppatori e aiutano a rendere il codice più semplice e leggibile, consentendo così una condivisione più naturale, una manutenzione più semplice e una migliore produttività. Pertanto, qualsiasi gruppo di sviluppatori deve scegliere uno stile, formalizzarlo e attenersi ad esso. Un buon punto di partenza per queste considerazioni sono le Linee guida per

lo sviluppo di LabVIEW, che si trovano come parte della documentazione online di LabVIEW. Questa sezione descriverà alcuni standard utili suggeriti da Conway et al. e mostrare alcune cattive pratiche. Cominciamo con l'orrore del diagramma mostrato nella Figura 2.16. Questo diagramma lascia un sacco di spazio vuoto per una ragione apparente e costringe chiunque lo usi a scorrere ovunque. Questo è il risultato di pura pigrizia e serve solo a rendere il processo di sviluppo molto più difficile. Pertanto, la prima regola da seguire è ridurre al minimo l'uso dello spazio bianco ove possibile. Il diagramma precedente è solitamente accompagnato da un pannello frontale simile alla Figura 25 Controlli e indicatori sono sparsi dappertutto quando ci sarebbe voluto meno di un minuto per allineare tutto correttamente. Al contrario, il diagramma nella Figura 2.17 mostra un'immagine molto affollata in cui è difficile vedere i singoli pezzi. Un grande vantaggio di LabVIEW è che un diagramma a blocchi aiuta a comunicare il progetto. Questo diagramma accumula così tante informazioni che non riesce a trasmettere molto di nulla.

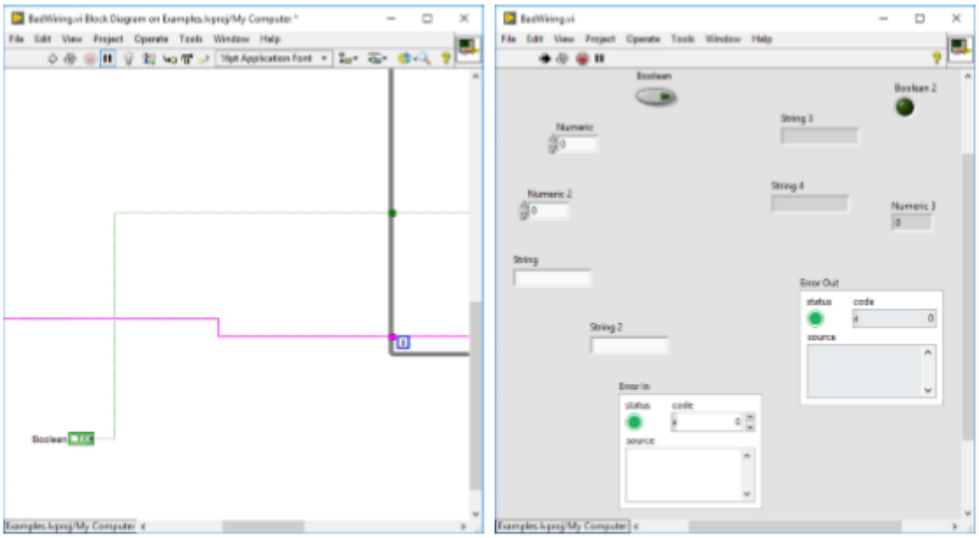


Figura 2.16: VI con spazi male utilizzati e molto dispersiva.

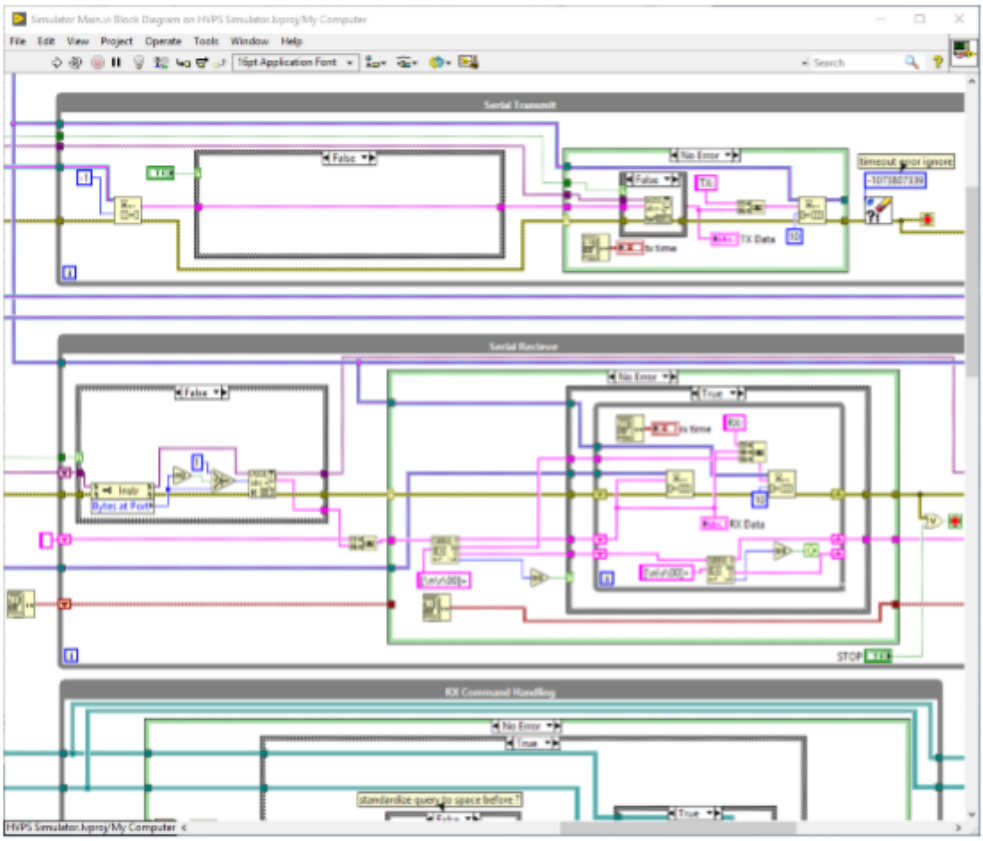


Figura 2.17: Esempio di VI troppo affollato.

2.2.11 Diagramma a Blocchi

Conway et al. ha proposto una serie di standard separati in categorie dedicate a ciascun aspetto dello sviluppo di LabVIEW. A partire dal diagramma a blocchi, uno sviluppatore dovrebbe attenersi ai seguenti standard per quanto riguarda il suo layout generale.

- Rendere i diagrammi il più compatti possibile pur avendo un flusso visibile.
- Non riempire lo schema a blocchi.
- Imponi sequenze. Non lasciare le strutture galleggianti. Oltre ad avere un aspetto negativo, può portare a incongruenze nel programma.
- Posizionare eventuali terminali privi di cablaggio nella prima struttura dello schema. Possono sembrare errori se lasciati fluttuare.

- Cerca di mantenere la dimensione complessiva del diagramma su una pagina dello schermo. Se è più grande, prova a mantenere 1 pagina larga o 1 pagina profonda in modo da dover scorrere solo in una direzione.

Una pagina può essere qualunque sia la risoluzione standard dello schermo. Le norme per il cablaggio dello schema a blocchi sono costituite dalle seguenti regole.

- Utilizzare il flusso di dati da sinistra a destra. Resisti alla tentazione di inserire i dati nelle strutture dall'alto o dal basso.
- I fili dovrebbero sovrapporsi solo se non ci sono alternative.
- Mantenere i fili il più diritti possibile.
- Non instradare mai i cavi attraverso un'icona a un terminale sull'altro lato dell'icona.
- Non instradare mai i cavi sotto strutture o icone.
- Non utilizzare variabili locali solo per evitare lunghi fili! Ogni variabile locale che legge i dati ne fa una copia.
- Ridurre il numero di zigzag allineando ingressi e uscite. Usa i tasti cursore per eliminare i nodi dai fili.
- Eliminare i fili in eccesso, come i loop.
- Cerca di mantenere i fili paralleli distanziati uniformemente anche intorno agli angoli. Gli standard per l'etichettatura all'interno dei diagrammi a blocchi includono le seguenti regole.
- Ogni frame in una struttura Sequence, Case o Loop dovrebbe avere un commento descrittivo. Questi dovrebbero essere rientrati se ci sono strutture incorporate.
- Utilizzare etichette libere su lunghi tratti di filo per etichettare i dati. Posiziona l'etichetta proprio sopra il filo con uno sfondo trasparente.
- I commenti per il cablaggio devono avere uno sfondo bianco senza bordi. Un commento con bordi potrebbe essere scambiato per un'icona.
- Quando un VI viene caricato su una piattaforma diversa, i font si adattano a quella nuova piattaforma. Sul pannello frontale, LabVIEW cerca di spostare le etichette in modo che non si sovrappongano ai controlli. Non lo fa sul diagramma. È meglio posizionare le etichette sotto gli oggetti che descrivono, così quando crescono e si restringono sul lato inferiore e destro, rimangono accanto all'oggetto. Se posizioni un'etichetta a sinistra del suo oggetto, giustificala a destra in modo che cresca a sinistra.

- Utilizzare le etichette per i casi booleani per documentare gli stati dei casi. Questo non solo migliora la leggibilità del tuo codice, ma aiuta anche con la logica di un programma. Scoprirai che il semplice atto di documentare un caso logico ti impedirà di essere legato agli OR, agli AND e alla logica negativa.
- Usa le etichette su While Loops e For Loops per documentare il loro scopo.
- Registri a scorrimento etichetta con il loro contenuto di dati.
- Per le etichette e le costanti, utilizzare le dimensioni per il testo ei ritorni a capo per ogni riga.
- Standardizzare su caratteri e stili.

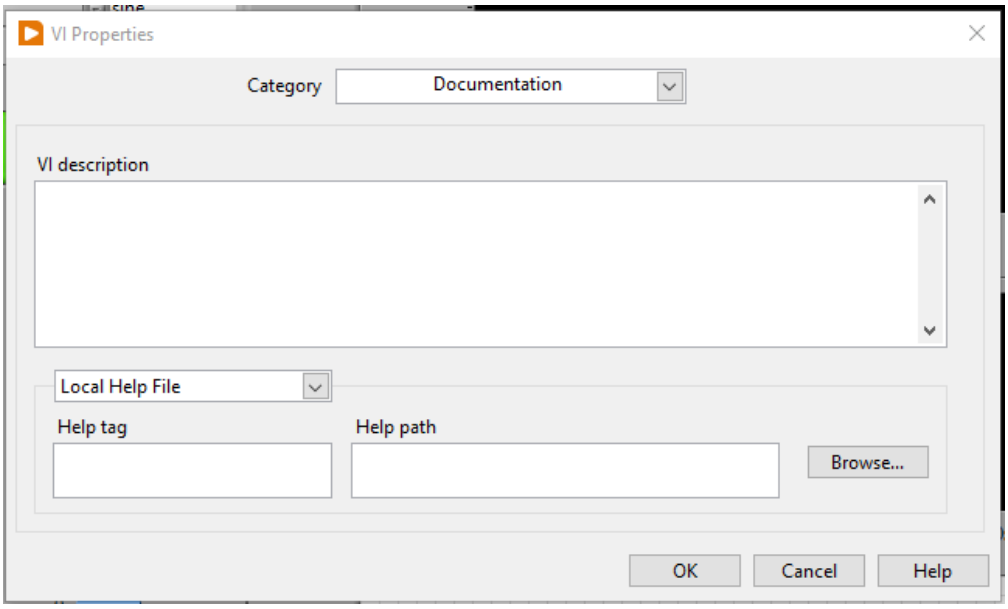


Figura 2.18: Proprietà dei VI.

2.2.12 Pannello Frontale

Per quanto riguarda il pannello frontale, Conway et al. introdurre prima una serie di norme generali.

- In generale, si dovrebbe usare un carattere standard sul pannello frontale invece di optare per qualcosa di stravagante. Una certa standardizzazione delle etichette è utile, ma non è una grande preoccupazione.
- Ogni VI dovrebbe avere una descrizione che contenga tutte le informazioni rilevanti disponibili nel menu contact help di LabVIEW. Anche avere un modello di descrizione standard per tutti i VI è una buona idea. La Figura 2.18 ne mostra un esempio. Per rendere pubblici i VI visibili all'utente, viene proposta la seguente serie di standard per il pannello frontale pubblico.
- Utilizzare definizioni di tipo rigorose per i comandi e gli attributi enumerati.
- Compila l'icona. Ci vuole solo un minuto e, poiché LabVIEW è un linguaggio iconico, migliorerà notevolmente la leggibilità del tuo programma.
- Una descrizione utente per ogni controllo o indicatore aiuterà nella documentazione dell'applicazione e renderà disponibili all'utente anche le funzionalità di aiuto di LabVIEW.
- Mantieni un aspetto comune, almeno per lo stesso cliente. Ciò migliorerà la familiarità e ridurrà la curva di apprendimento quando al cliente viene fornito un nuovo software.

Per creare subVI privati che non sono disponibili o visibili all'utente, viene proposta la seguente serie di standard per il pannello frontale privato.

- Controlli connettore a sinistra.
- Indicatori connettore a destra.
- Indicatori e controlli locali al centro o in basso.
- Allinearli e distanziarli uniformemente.
- Disponili nello stesso ordine in cui sono collegati
- Inquadrare le aree rilevanti Input, Output e Locale ove applicabile.
- Utilizzare ed etichettare la cornice da incasso dal menu delle decorazioni.
- Utilizzare il pennello con il primo piano impostato su trasparente per riordinare l'etichetta.

Infine, per quanto riguarda l'aspetto di un subVI, vengono proposti i seguenti standard di icone e connettori.

- Compila le tue icone.
- Un carattere leggibile e compatto è Arial alla dimensione 14.
- I caratteri piccoli di dimensione 11 funzionano bene come alternativa.
- Posiziona gli input a destra, gli output a sinistra e i comandi in alto al centro.
- Se possibile, per le icone a grandezza naturale, mantenere 4 in e 4 out come schema di connessione. Questo aiuta a mantenere il cablaggio in ordine.
- Raggruppamenti di errori di cablaggio lungo il connettore di ingresso e uscita inferiore.

Il connettore standard utilizzato da Conway et al. è mostrato nella Figura 2.19. Questo lavoro segue in gran parte questi standard con l'eccezione che il modello di connettore 4x2x2x4 viene utilizzato per impostazione predefinita.

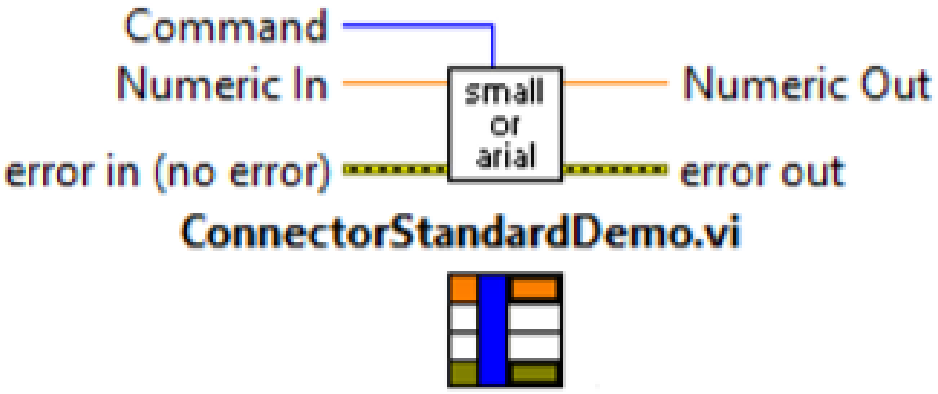


Figura 2.19: Icona del VI e connector pane.

2.3 Design orientato a oggetti in LABVIEW

Il supporto per la programmazione orientata agli oggetti è stato introdotto in LabVIEW nel 2006 con la versione 8.20. Da allora sono state aggiunte sempre più funzionalità orientate agli oggetti. Invece di utilizzare componenti, è ora possibile creare classi e oggetti che consentono di applicare più tecniche orientate agli oggetti. Prima di discutere come la progettazione orientata agli oggetti può essere applicata alle applicazioni LabVIEW, dobbiamo prima analizzare le differenze tra LabVIEW e altri linguaggi di programmazione per quanto riguarda l'orientamento agli oggetti. A differenza di altri linguaggi, LabVIEW supporta l'uso di:

1. Ereditarietà singola: l'ereditarietà multipla è severamente vietata (il supporto per le interfacce è stato aggiunto nella versione 2020. di LabVIEW).

2. Incapsulamento rigoroso: i dati della classe devono essere privati. I dati protetti e pubblici non sono supportati.

Queste e le altre differenze verranno ulteriormente discusse nella sezione seguente.

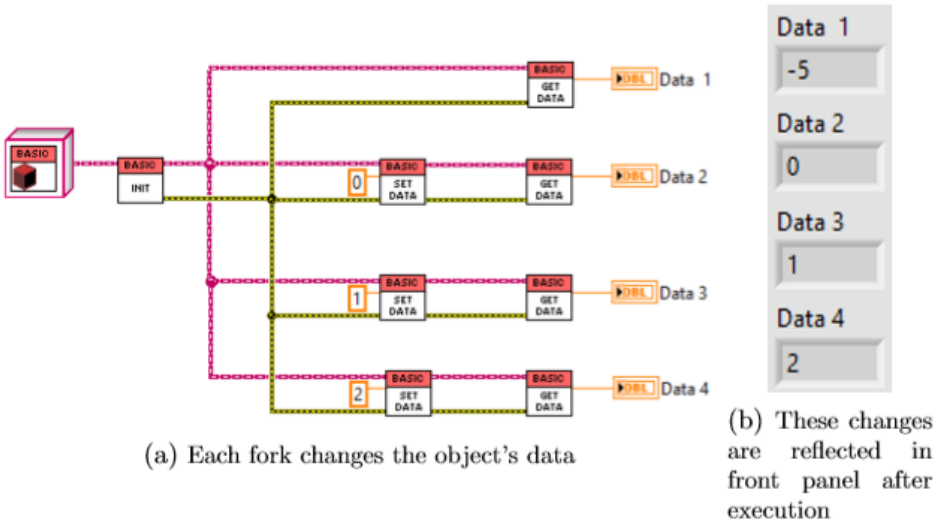
2.3.1 Funzionalità orientate agli oggetti

La sintassi del flusso di dati di LabVIEW ha avuto un grande impatto sulla progettazione e sull'implementazione di concetti orientati agli oggetti all'interno. Il funzionamento di classi e oggetti deve essere compatibile e comprensibile all'interno del framework del flusso di dati. La loro inclusione non dovrebbe infrangere alcuna concezione precedente che uno sviluppatore potrebbe avere, ma semplicemente estendere le proprie capacità di sviluppo. Pertanto, queste decisioni hanno determinato quali funzionalità sono state aggiunte e come sono state implementate.

2.3.2 Accesso per valore o per riferimento

In accordo con il principio del flusso di dati, gli oggetti in LabVIEW vengono trasportati attraverso i cavi come qualsiasi altro tipo di dati. Questo cavo contiene lo stato interno dell'oggetto e può essere gestito tramite i metodi della classe. I fili che tengono un oggetto sono coerenti con il principio del flusso di dati e trasportano i dati per valore. In effetti un oggetto funziona come un cluster di dati a cui è possibile accedere direttamente solo dai metodi della classe. La biforcazione del filo di un oggetto duplica l'oggetto o crea una "copia cartacea". Dopo un fork, i nuovi oggetti sono indipendenti dall'originale e le modifiche ai dati interessano solo ogni nuovo oggetto individualmente. Ciò è in contrasto con Java, dove una variabile oggetto contiene semplicemente il riferimento e la vera duplicazione viene ottenuta creando esplicitamente una "copia cartacea". Per dimostrare il principio

in LabVIEW, si consideri un oggetto di base che contiene solo un valore numerico chiamato Data inizializzato a -5. La Figura 2.20 mostra che la modifica del campo Data dopo un fork ha effetto solo sui nuovi oggetti indipendenti. Dal 2009 in poi è diventato possibile accedere agli oggetti per riferimento in LabVIEW. Le operazioni standard sono rimaste le stesse e gli oggetti wire hanno ancora applicato il principio dei dati per valore, ma LabVIEW ha introdotto un nuovo tipo di dati chiamato Data Value Reference (DVR). Questo tipo di dati può essere utilizzato per contenere un riferimento a qualsiasi altro dato in LabVIEW inclusi gli oggetti. Tuttavia, per impostazione predefinita, un riferimento a un oggetto può essere creato solo da un metodo dell'oggetto. L'intenzione è di limitare la creazione di riferimenti al progettista della classe e garantire una corretta gestione. Se necessario, questa opzione può essere disabilitata nella finestra di dialogo Proprietà classe.



(a) Each fork changes the object's data

(b) These changes are reflected in front panel after execution

Figura 2.20: Esempio di fork di un cavo di un oggetto.

2.3.3 Costruttori e Decostruttori

Non ci sono costruttori o distruttori in LabVIEW. In genere, i costruttori vengono utilizzati per configurare i valori iniziali di un oggetto impostandoli direttamente, calcolandoli da un insieme di parametri o calcolandoli da dati ambientali (come ottenere un timestamp al momento della creazione dell'oggetto). Un altro caso d'uso comune per un costruttore è la prenotazione di risorse di sistema (come memoria di sistema, dispositivi connessi, canali di comunicazione, ecc.). LabVIEW supporta la possibilità di impostare il valore di default dell'oggetto, configurando i default del private data typedef. Tuttavia, qualsiasi altro calcolo implicito non è supportato in quanto ciò potrebbe causare confusione per l'utente e va contro la filosofia del flusso di dati. Se è necessario eseguire calcoli o aprire canali di comunicazione, è possibile eseguirli esplicitamente tramite una funzione di inizializzazione. I calcoli basati sull'ambiente possono essere eseguiti da utenti più avanzati con l'input di XControl che può eseguire del codice in fase di esecuzione. I distruttori sono tipicamente usati per deallocare le risorse riservate dall'oggetto al momento della sua distruzione. Nei linguaggi di programmazione tradizionali la durata di un oggetto è sempre nota. Ad esempio, in C++ un oggetto cessa di esistere non appena il runtime lascia le parentesi graffe in cui è stato definito l'oggetto. In LabVIEW, tuttavia, la durata di un oggetto o di qualsiasi dato in generale è sconosciuta. La data si tiene finché è necessario. Un cavo dati memorizzerà il valore fino alla successiva esecuzione su quel cavo o fino alla fine del runtime. Qualsiasi dato collegato a un pannello frontale rimarrà lì anche dopo l'esecuzione. Pertanto, gli oggetti non hanno una durata definita e i distruttori non sarebbero in grado di rilasciare alcuna risorsa riservata. Risorse come canali di comunicazione o dispositivi possono essere riservate tramite i tipi di dati del numero di riferimento (refnum), che portano un numero che fa riferimento a un altro tipo di dati. Se un oggetto ha bisogno di riservare tale risorsa, deve essere fatto esplicitamente attraverso metodi di classe.

2.3.4 Dati Statici

Una caratteristica interessante di LabVIEW è la capacità di creare dati statici condivisi tra tutte le istanze della classe. In memoria esiste solo una copia di questi dati e chiunque abbia accesso sta lavorando con gli stessi dati. Questa non è una caratteristica apparente delle classi, ma può essere ottenuta utilizzando la stessa tecnica che ha permesso ai componenti di avere una memoria interna in LCOD. Un registro a scorrimento non inizializzato viene inserito all'interno di un metodo di una classe. Lo sviluppatore può quindi determinare quale tipo di dati deve contenere il registro a scorrimento e dare la possibilità di configurare e leggere questi dati con il metodo. L'ambito di questo metodo dovrebbe essere impostato su privato o protetto, poiché l'ambito pubblico dà troppo potere a fattori esterni e interrompe l'incapsulamento della classe. Qualsiasi numero di VI interni potrebbe dipendere da questi dati condivisi affinché un utente esterno possa apportare modifiche improprie ad esso.

2.3.5 Relazioni

Come accennato, LabVIEW supporta l'ereditarietà singola, mentre l'ereditarietà multipla non è consentita. Attraverso l'ereditarietà, una classe figlio ottiene l'accesso ai metodi protetti del genitore. Tutti i dati che il genitore potrebbe avere sono ancora nascosti al figlio poiché tutti i dati sono privati; il figlio può accedere solo ai propri campi dati. I metodi di una classe LabVIEW possono essere statici o dinamici. Il metodo statico funziona come un subVI standard. Un bambino può utilizzare qualsiasi metodo statico protetto o pubblico ereditato, ma non può modificarlo. Un metodo dinamico consente ai bambini di sovrascrivere e ridefinire la funzionalità del metodo (la funzione dinamica deve ancora essere nell'ambito protetto/pubblico affinché sia possibile eseguire l'override). Poiché il metodo dinamico può essere collegato a più subVI, la funzione che viene effettivamente chiamata viene determinata in fase di esecuzione. Sebbene l'ereditarietà da più classi non sia consentita, l'aggiunta di interfacce a LabVIEW ha apportato un interessante cambiamento alla dinamica. Un'interfaccia è effettivamente una classe che non ha dati, solo metodi. Tradizionalmente, queste sarebbero classi astratte che dichiarano solo le funzioni di una classe. Qualsiasi classe che utilizza l'interfaccia dovrebbe quindi implementare queste funzioni. Tuttavia, in LabVIEW anche i metodi nell'interfaccia hanno un'implementazione predefinita (anche altri linguaggi principali hanno iniziato ad aggiungere il supporto per l'implementazione predefinita). Ciò ha portato gli sviluppatori di LabVIEW a cambiare la terminologia quando si tratta di interfacce e ad adottare i familiari termini di ereditarietà per descrivere meglio la natura della relazione e rendere più facile la comprensione del concetto per i nuovi utenti. Se questo fosse tutto ciò che le interfacce hanno portato con sé, il cambiamento sarebbe piuttosto insignificante. Uno sviluppatore potrebbe già creare

una pseudo-interfaccia creando una classe senza dati che contenga solo metodi pubblici. Tuttavia, il grande cambiamento deriva dal fatto che "una classe può avere un genitore di classe e tanti genitori di interfaccia quanti ne vuole. Eredita da tutti loro. Le interfacce forniscono una forma di ereditarietà multipla". Questa aggiunta offre agli sviluppatori una maggiore flessibilità e aiuta a contenere e risolvere i problemi dell'ereditarietà multipla. In particolare, LabVIEW vieta la chiamata di metodi genitori di qualsiasi interfaccia, evitando così collisioni di nomi e problemi di ereditarietà ripetuta. Se un figlio eredita un metodo con lo stesso nome da più interfacce, deve sovrascrivere quel metodo. In genere, tutti i metodi ereditati da un'interfaccia devono essere sovrascritti per impostazione predefinita poiché di solito hanno funzionalità incomplete, ma l'utente può disabilitare questa impostazione per metodi specifici. L'aggregazione è un altro tipo di relazione che può essere formata in LabVIEW. All'interno di LabVIEW è possibile sia l'aggregazione per valore che con il rilascio dei DVR l'aggregazione per riferimento. Lo sviluppatore deve semplicemente aggiungere la classe aggregata o il DVR all'interno dei dati privati di una classe diversa. Tuttavia, si applicano ancora le regole standard dei DVR di classe in cui il DVR può essere creato solo se la classe ha VI specifici che lo gestiscono o se la classe disabilita l'opzione che limita la creazione di DVR ai metodi interni.

2.3.6 Amici

Alcuni linguaggi di programmazione danno alle classi la possibilità di dichiarare "amici" dando loro il permesso di accedere a tutte le loro parti private. LabVIEW implementa anche il concetto di amici, ma utilizza una versione molto più ristretta. Il ragionamento alla base dell'implementazione è che dare accesso a tutte le parti private apre troppe backdoor in cui i dati possono essere modificati evitando i VI destinati a tale scopo. In entrambi i casi, nella maggior parte dei casi non è necessario concedere l'accesso a tutte le funzioni private poiché gli amici potrebbero aver bisogno solo di alcune funzioni. Friendship in LabVIEW dà la possibilità a qualsiasi libreria (una classe in LabVIEW può essere definita libreria di classi) di dichiarare un elenco di VI o librerie amici. Questi amici ottengono quindi l'accesso ai VI dichiarati nell'ambito della comunità (tutti i dati sono privati per le classi). Ciò offre alle classi un maggiore controllo dei propri metodi, pur consentendo l'accesso a funzioni altrimenti private. Un'altra cosa da notare è l'interazione tra eredità e amicizia. Una classe figlio non ottiene automaticamente l'accesso ad alcun metodo nell'ambito della comunità di suo padre. Questo è di nuovo inteso per dare più controllo al progettista di classe.

2.3.7 Sovraccarico e sovrascrittura delle funzioni

Alcuni linguaggi orientati agli oggetti hanno la caratteristica di sovraccaricare le funzioni in cui due diverse funzioni possono avere lo stesso nome, ma accettare parametri di input diversi. Il compilatore determina quindi quale funzione utilizzare in base ai parametri forniti da ciascuna chiamata di funzione. Questa caratteristica non è presente in LabVIEW e non è possibile creare due VI con lo stesso nome all'interno della stessa libreria o classe. L'override delle funzioni è una caratteristica dell'ereditarietà che consente alle classi figlie di sovrascrivere le funzioni dei loro antenati. Questa funzionalità è supportata in LabVIEW per i metodi dinamici di una classe o interfaccia. Se un figlio tenta di sovrascrivere un metodo statico del genitore, l'implementazione del metodo del figlio diventa interrotta e non è eseguibile. Queste caratteristiche portano alla domanda su cosa succede nei casi in cui una classe eredita più interfacce che condividono metodi con lo stesso nome? Bene, ci sono due possibili risultati. Le interfacce potrebbero implementare metodi diversi con lo stesso nome, il che significa che questi metodi avrebbero input e output diversi. Questa situazione rientra nella categoria del sovraccarico di funzione che è vietato. Questo porta a una situazione rotta per la classe del bambino. La seconda situazione è quando le interfacce condividono gli stessi metodi con gli stessi ingressi/uscite. In questo caso, il figlio deve sovrascrivere tutti i metodi condivisi dalle interfacce padre. Altrimenti, il bambino è rotto. In futuro LabVIEW ha potenziali piani per introdurre l'aliasing del metodo e consentire al bambino di sovrascrivere più funzioni diverse che condividono lo stesso nome. Questo non sarebbe esattamente lo stesso dell'overload di funzioni poiché il bambino avrebbe nomi diversi per questi metodi. Tuttavia, ciò affronterebbe il primo scenario e consentirebbe al bambino di ereditare interfacce sviluppate in modo indipendente che scelgono lo stesso nome per alcuni metodi. Questo approccio ha il potenziale per confondere lo sviluppatore su quale VI verrà invocato, motivo per cui questa funzionalità è esclusa dalla versione corrente di LabVIEW.

2.3.8 Implementazione e Qualità

La progettazione orientata agli oggetti in LabVIEW dovrebbe utilizzare tutte le idee necessarie dalla progettazione orientata agli oggetti pura, pur utilizzando alcune tecniche e mantenendo gli utili standard di LCOD. L'uso di strutture di casi e macchine a stati in particolare può essere un'aggiunta molto utile. Tuttavia, non è più necessario combinare le macchine a stati in un unico componente che gestisce tutti i comandi. Gli oggetti stessi possono trasportare dati e i diversi aspetti della funzionalità possono essere realizzati attraverso metodi pubblici. Una classe di livello superiore può quindi racchiudere queste diverse funzioni in una struttura case e attivare comandi (metodi) diversi in base ai messaggi ricevuti da altre classi. È anche importante considerare i metodi di gestione degli errori discussi nella Sezione

apposita. Avere una classe di gestione degli errori che registri e gestisca correttamente qualsiasi problema relativo agli errori sarebbe di grande beneficio. Sarebbe utile sia durante lo sviluppo per trovare rapidamente i problemi e soprattutto durante la manutenzione mantenendo un registro dettagliato di qualsiasi problema che si è verificato. Quando si tratta di qualità del software, è determinata principalmente da due fattori: qualità del design e qualità dell'implementazione. L'aderenza agli standard LCOD, gioca un ruolo importante nella qualità dell'implementazione. Questi standard migliorano la leggibilità del codice, facilitano la manutenzione e aumentano la produttività. Alcuni set di standard dovrebbero essere applicati a qualsiasi tipo di codice LabVIEW, anche quando si utilizza un approccio di progettazione diverso. Il principale contributo alla qualità dell'approccio orientato agli oggetti sono gli attributi di qualità discussi nella Sezione dedicata in precedenza. Rappresentano la qualità del design, ma hanno anche un impatto sulla qualità dell'implementazione. Questi attributi di qualità sono: accoppiamento, coesione, sufficienza, completezza e primitività. Il design dovrebbe cercare di incorporare questi concetti il più possibile.

2.4 Modulo Real Time e modulo FPGA

Di seguito si riportano in dettaglio le parti che costituiscono il codice Labview, partendo dalla suddivisione fondamentale delle tre macroaree in cui il codice lavora: PC, Real Time target e FPGA. L'ambiente di sviluppo dell'applicazione LabVIEW Real-Time Module funge da strumento completo di sviluppo e debug. Le piattaforme di implementazione del modulo LabVIEW Real-Time si basano su ad architettura hardware e software comune. Ogni target hardware utilizza componenti di elaborazione come un microprocessore, RAM, memoria non volatile e un'interfaccia bus I/O. Il software integrato è costituito da un sistema operativo Real time, un software driver e una versione specializzata di LabVIEW Run-Time Engine. Un sistema in real time è costituito da componenti software e hardware. Il i componenti software includono LabVIEW, RT Engine e LabVIEW progetti e VI creati. I componenti hardware di un real-time il sistema include un computer host e un target RT. Le seguenti sezioni descrivono le diverse componenti di un sistema in tempo reale:

- Host computer: L'host computer è il computer su cui LabVIEW e LabVIEW Real-Time Module sono installati e su cui si sviluppano i VI per il sistema in tempo reale. Dopo aver sviluppato i VI di sistema in tempo reale, è possibile scaricare ed eseguire i VI sui target RT. Il computer host può eseguire VI che comunicare con i VI in esecuzione su target RT per fornire un'interfaccia utente.

- Labview: si sviluppano VI con LabVIEW sul computer host. Il modulo real time estende le capacità di LabVIEW con strumenti aggiuntivi per creazione, debug e distribuzione di VI deterministici.
- RT Engine: è una versione di LabVIEW che gira su target RT. Il RT Engine esegue i VI scaricati sui target RT. Il motore RT fornisce prestazioni deterministiche in tempo reale per i seguenti motivi:
  - Il motore RT funziona su un sistema operativo in tempo reale (RTOS), che assicura che il sistema di esecuzione LabVIEW e altri servizi aderiscano al funzionamento in tempo reale.
  - Il motore RT funziona su hardware della serie RT. I target RT sono progettati per eseguire solo i VI e i driver di dispositivo necessari per le applicazioni RT, che impedisce ad altre applicazioni di impedire l'esecuzione di RT VI.
  - I target RT non utilizzano la memoria virtuale, perché la memoria virtuale può causare prestazioni imprevedibili.
- RT target: Un target RT si riferisce all'hardware della serie RT che esegue il motore RT e i VI create utilizzando LabVIEW. Un dispositivo della serie RT in rete è una piattaforma hardware con un processore integrato e un funzionamento in tempo reale sistema che esegue RT Engine e LabVIEW VI. Puoi usare un separato computer host per comunicare e controllare i VI su una rete Dispositivo della serie RT tramite una connessione Ethernet.

In particolare, il cRIO è un sistema progettato per applicazioni che richiedono elevate prestazioni e affidabilità, e la parte che conferisce ancora più affidabilità del RT target è l'FPGA. Con il modulo FPGA si configura il comportamento del riconfigurabile FPGA per soddisfare i requisiti di una misura e un controllo specifici sistema. Utilizzare il modulo FPGA per scrivere VI FPGA. Quando scarichi l'FPGA VI all'FPGA, stai programmando la funzionalità dell'FPGA dispositivo. Ogni nuovo VI FPGA che crei e scarichi è una tempistica personalizzata, triggering e soluzione di I/O. Quando l'hardware standard non soddisfa i requisiti per uno specifico prima del modulo FPGA, dovevi creare un'applicazione personalizzata progettazione hardware utilizzando linguaggi di descrizione hardware di basso livello. Con il Modulo FPGA, non è necessario conoscere un linguaggio di descrizione dell'hardware per progettare una soluzione hardware specifica, è sufficiente LabVIEW. Con il Modulo FPGA, puoi progettare e sviluppare rapidamente componenti hardware con la potenza della programmazione grafica di LabVIEW.

Capitolo 3

Architettura del Codice Labview

3.1 FPGA

Di seguito è riportata la schermata “project explorer”, che viene usata per determinare le gerarchie dei vari VI. Come è possibile osservare nella sezione “My

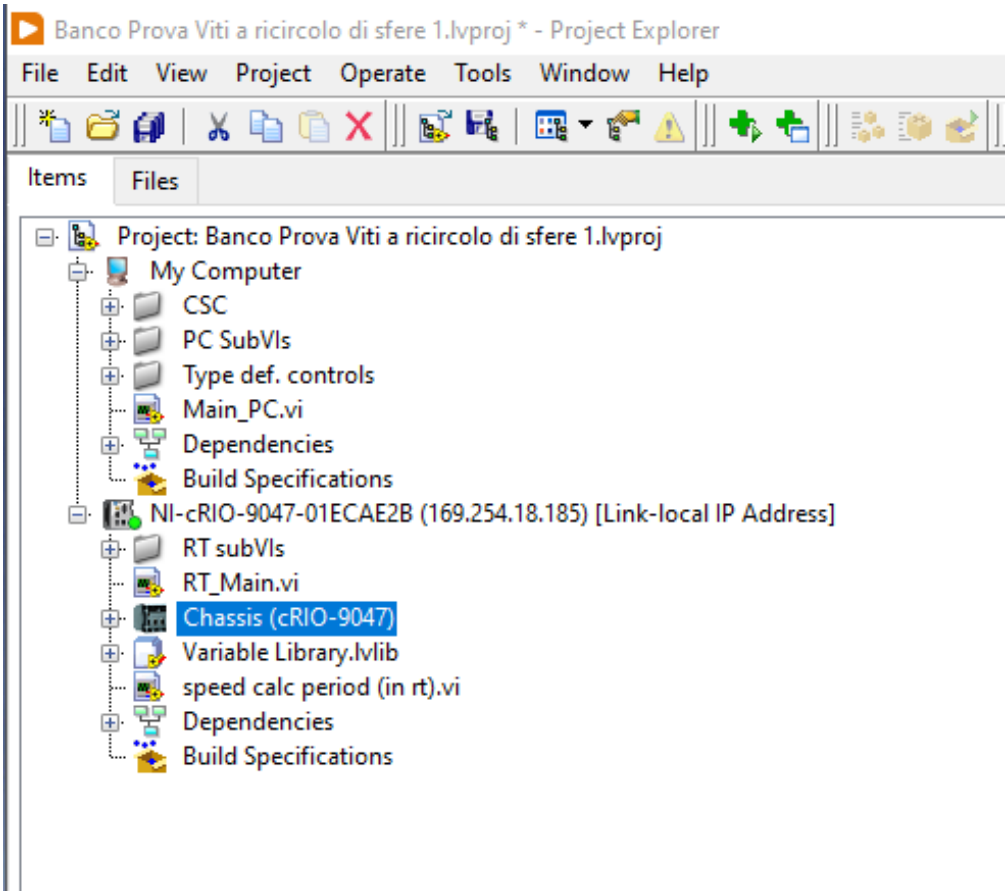


Figura 3.1: Screenshot del Project Explorer.

Computer" sono presenti i VI che vengono elaborati dal PC e che svolgono delle funzioni di supporto, post-processing e settaggio dei parametri di controllo come il SET di posizione o velocità. La sezione NI-cRIO-9047 contiene la parte Real Time e all'interno della sezione "Chassis" troviamo la parte FPGA, oggetto di questo capitolo. Di seguito viene riportata la parte dello chassis.

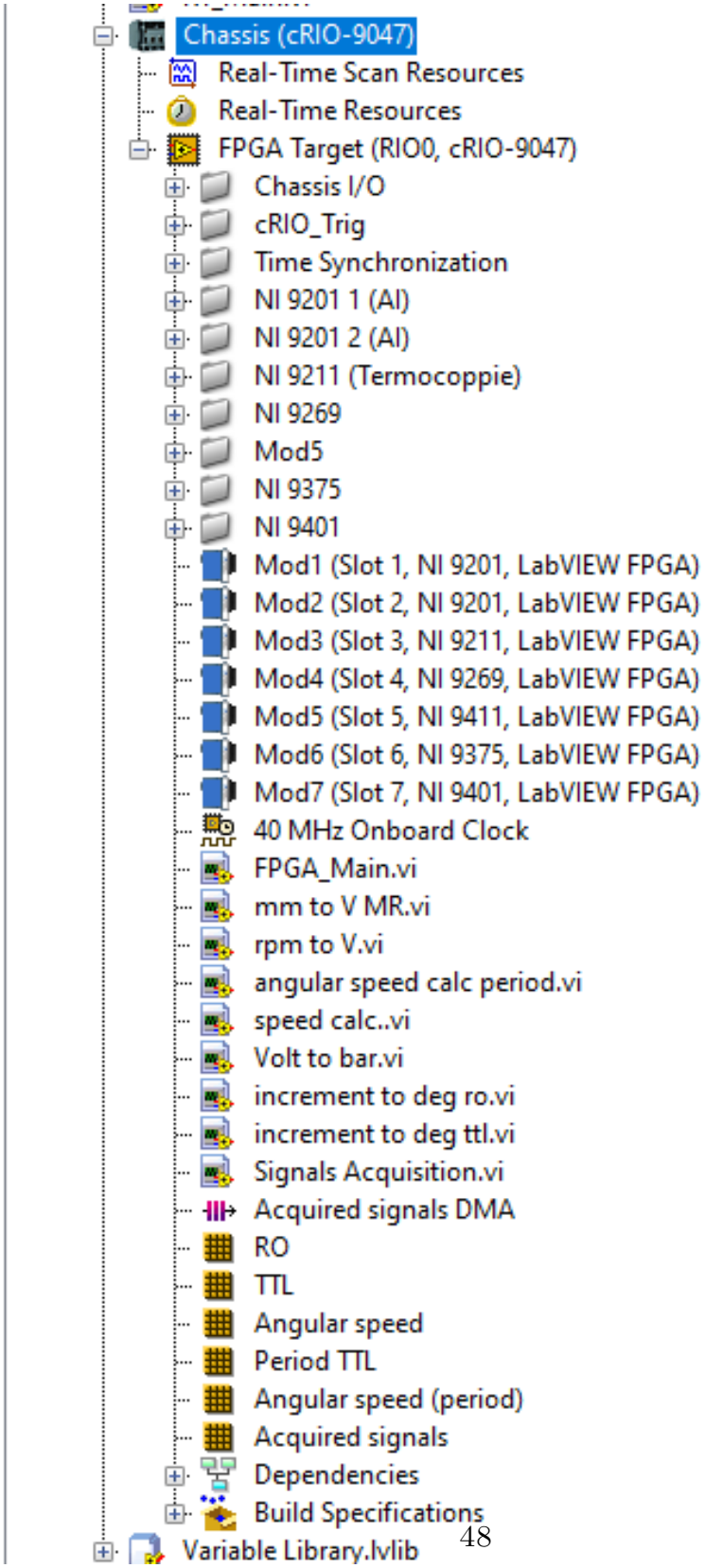


Figura 3.2: Screenshot del Project Explorer.

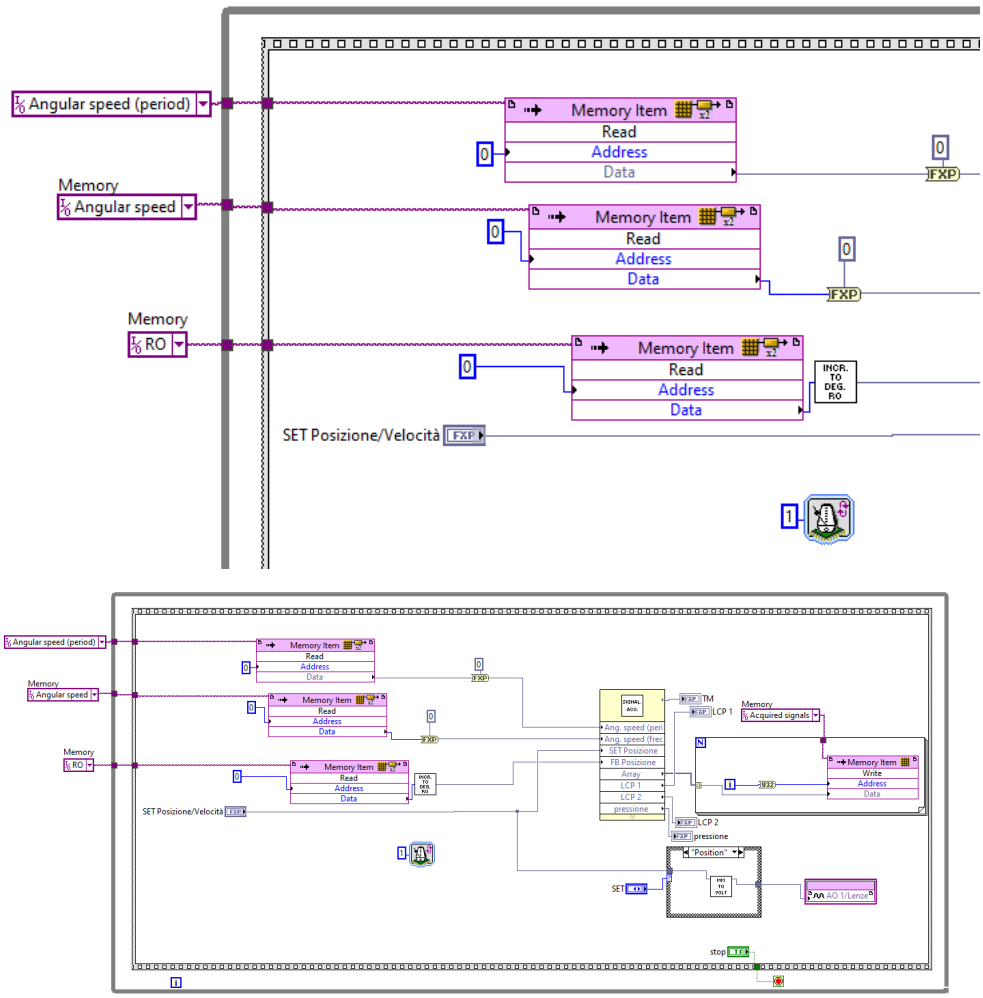


Figura 3.3: Loop di acquisizione dei segnali e Particolare sugli oggetti memoria.

In figura 3.2 si osservano le cartelle contenenti i canali presenti nei vari moduli. Mediante questi canali è possibile ricavare tramite il codice i segnali provenienti dai vari sensori. Lo schema elettrico che spiega a quali canali sono collegati i rispettivi sensori è riportato in fig. 1.6. Inoltre, notiamo i VI del gruppo FPGA, tra i quali "FPGA Main" è quello principale, gli altri sono tutti subVI contenuti in esso. Proseguendo sotto troviamo il clock presente nell'hardware di calcolo, di cui andremo a parlare in seguito. In basso sono presenti gli oggetti memoria e il DMA FIFO, che verranno discussi in seguito. Ora procediamo illustrando le varie parti che compongono il codice FPGA. In fig. 3.3 viene mostrato il loop che nel codice FPGA è dedicato ad acquisire i segnali provenienti dai moduli analogici, ad acquisire i dati provenienti dalle memorie, ovvero i segnali degli encoder trasformati

in segnali di posizione e velocità, ad inviare il seti di posizione o di velocità al motore Lenze e ad inviare il vettore contenente tutti i segnali al DMA FIFO. Nel particolare si vedono i memory items, che sono utilizzati per leggere dei dati che vengono elaborati in un'altra parte del codice FPGA. Il mezzo principale di memorizzazione dei dati in un singolo dominio di clock utilizzando un'applicazione FPGA è un elemento di memoria. Il modulo LabVIEW FPGA ha due tipi di elementi di memoria:

- Elementi di memoria definiti da VI: utilizzare gli elementi di memoria definiti da VI insieme ai controlli del nome della memoria per creare subVI rientranti ed evitare conflitti di risorse. Se si configura un elemento di memoria definito dal VI in un VI FPGA rientrante, LabVIEW crea una copia separata dell'elemento di memoria per ogni istanza del VI.

- Elementi di memoria con ambito di destinazione: utilizzare elementi di memoria con ambito di destinazione se si desidera che l'elemento di memoria sia visibile e configurabile dalla finestra Esplora progetti. Gli elementi di memoria nell'ambito di destinazione sono staticamente associati per nome a un elemento corrispondente nel progetto, il che significa che gli aggiornamenti dell'elemento del progetto influiscono su tutte le istanze della memoria nel diagramma a blocchi. Gli elementi di memoria con ambito target sono disponibili all'interno di qualsiasi VI FPGA sotto lo stesso target nella finestra Project Explorer. Se si utilizza un elemento di memoria con ambito target e si desidera inviare il VI FPGA a un altro utente, è necessario inviare l'intero progetto. In caso contrario, l'FPGA VI è rotto.

È possibile utilizzare la finestra di dialogo Memory Properties per specificare come LabVIEW implementa un elemento di memoria. È sufficiente espandere il menu a discesa Implementazione per visualizzare le opzioni di memoria disponibili, come mostrato nell'illustrazione seguente.

- La memoria a blocchi: nota anche come memoria ad accesso casuale a blocchi, RAM a blocchi o BRAM, è una risorsa FPGA interna per l'archiviazione dei dati. Gli elementi di memoria che utilizzano la memoria a blocchi vengono compilati a una frequenza di clock elevata rispetto ad altri tipi di elementi di memoria. È possibile configurare la memoria a blocchi per l'accesso in lettura-scrittura o l'accesso in lettura a doppia porta. È inoltre possibile utilizzare un elemento di memoria implementato utilizzando la memoria a blocchi per scrivere dati in un dominio di clock e leggere i dati da un dominio di clock diverso. In questa implementazione, puoi utilizzare solo un nodo writer e un nodo reader per ogni elemento di memoria. La memoria a blocchi non consuma risorse FPGA. Usa prima la memoria a blocchi, a meno che tu non abbia bisogno dei vantaggi di un diverso tipo di memoria.

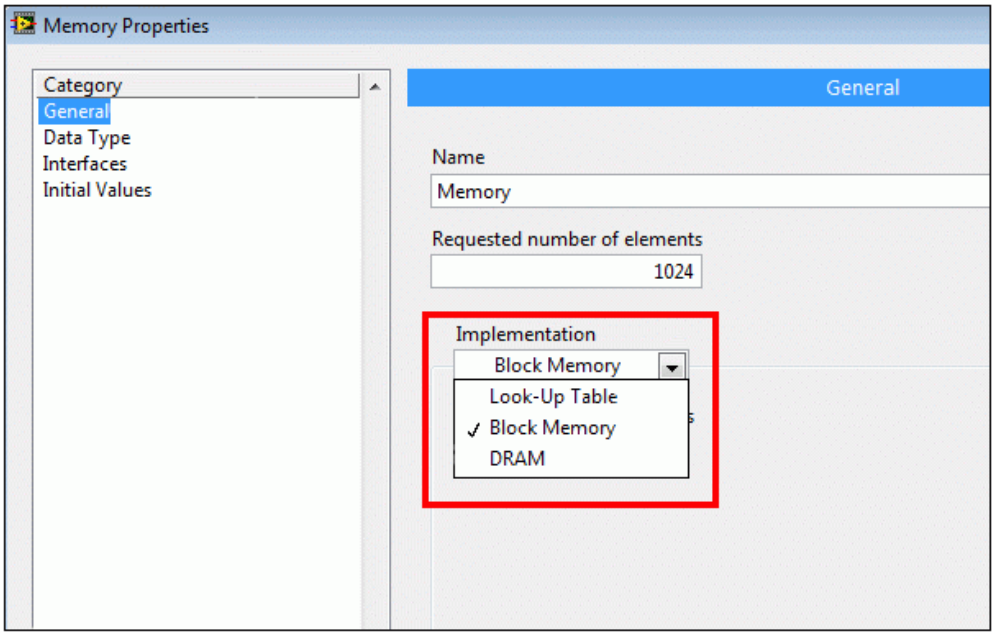


Figura 3.4: Impostazioni delle prorpietà delle memorie.

- Le tabelle di ricerca: note anche come RAM distribuita, sono costituite da porte logiche cablate sull'FPGA. Le LUT consumano risorse FPGA perché possono funzionare sia come risorse FPGA che come memoria. Utilizzare le tabelle di ricerca nelle seguenti situazioni:
  - Quando è necessario accedere a questa memoria in un Timed Loop a ciclo singolo e leggere i dati dall'elemento di memoria durante lo stesso ciclo di quello in cui si invoca il nodo
  - Quando hai una memoria di blocco rimanente limitata
- La DRAM è una forma di memoria esterna disponibile su alcuni target FPGA. La DRAM offre una grande quantità di spazio di archiviazione. Tuttavia, poiché la DRAM è esterna all'FPGA, l'applicazione non può ricevere dati dalla DRAM in un singolo ciclo di clock. La DRAM richiede anche l'accesso sequenziale, il che significa che solo un comando alla volta può accedere alla memoria. L'accesso sequenziale impedisce tempi deterministici e potrebbe aumentare il tempo di esecuzione, a seconda di quanti comandi sono in attesa di accedere alla DRAM.

Come si vede dal particolare di figura 3.3, queste memorie vengono qui utilizzate per leggere i dati che vi sono stati caricati in delle altre parti del codice, riuscendo a far lavorare tutte le varie sezioni in parallelo. Sempre nello stesso particolare, si

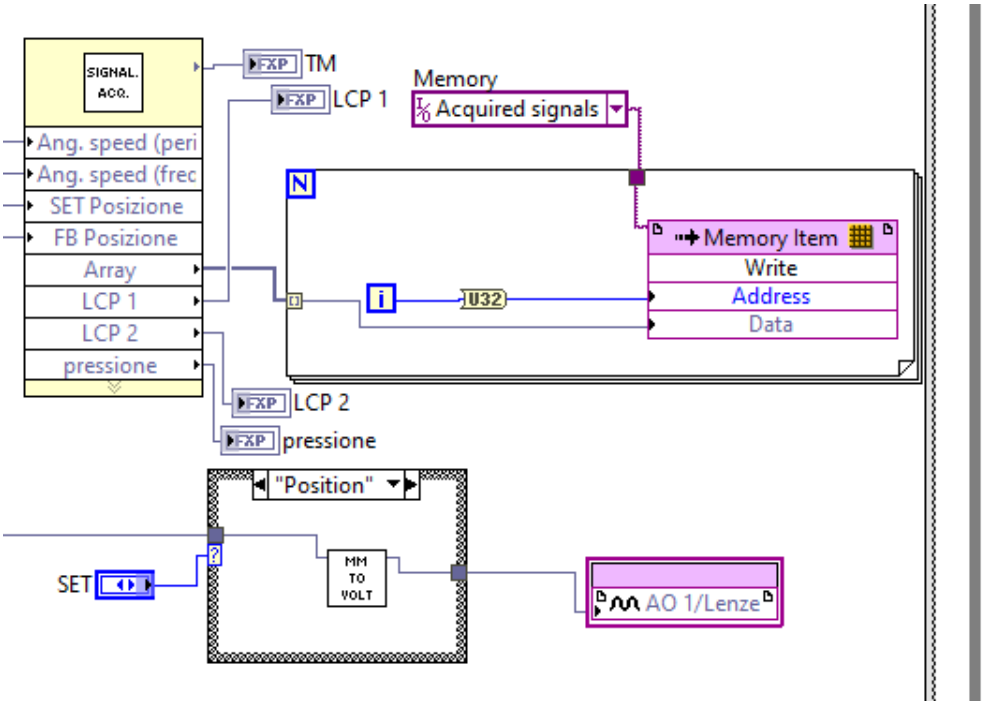


Figura 3.5: Particolare sulla VI di acquisizione segnali e sulla memoria dedicata ai FIFO.

nota la presenza di un "Loop Timer", ovvero una funzione che obbliga il ciclo while ad eseguire nel tempo stabilito dalla costante numerica assegnatagli, che in questo caso è di 1 ms, poiché vogliamo acquisire i segnali con una frequenza di 1kHz.

Nella figura 3.5 vediamo la seconda parte del ciclo while di acquisizione dati. È presente una subVI in modalità estesa, per evidenziarne meglio i pin, un ciclo for dove vengono salvati tutti i segnali in una memoria che verrà poi inviata al FIFO, e una case structure che serve a selezionare il tipo di SET che si vuole dare. Di seguito mostriamo cosa contiene la subVI di acquisizione.

Come si può notare, la subVi in fig. 3.6 utilizza vari FPGA I/O nodes e li unisce tutti in un unico vettore tramite la funzione "build array", che è molto utile quando, come in questo caso, si lavora con un numero elevato di segnali diversi. È importante osservare la presenza di molte funzioni per la conversione del tipo di variabile in fixed point. In Labview infatti ogni wire ha un preciso significato e un preciso colore, che ne determina il tipo di informazione che trasporta. Le macrocategorie di segnali sono quelle numeriche, le stringhe, i booleani (tra i quali sono presenti anche gli errori) e i cluster (ovvero dei gruppi di segnali di tipi diversi). Tutti questi tipi di segnali possono essere anche assemblati in vettori o matrici, quindi possono esistere vettori di numeri, vettori di stringhe, di booleani e anche

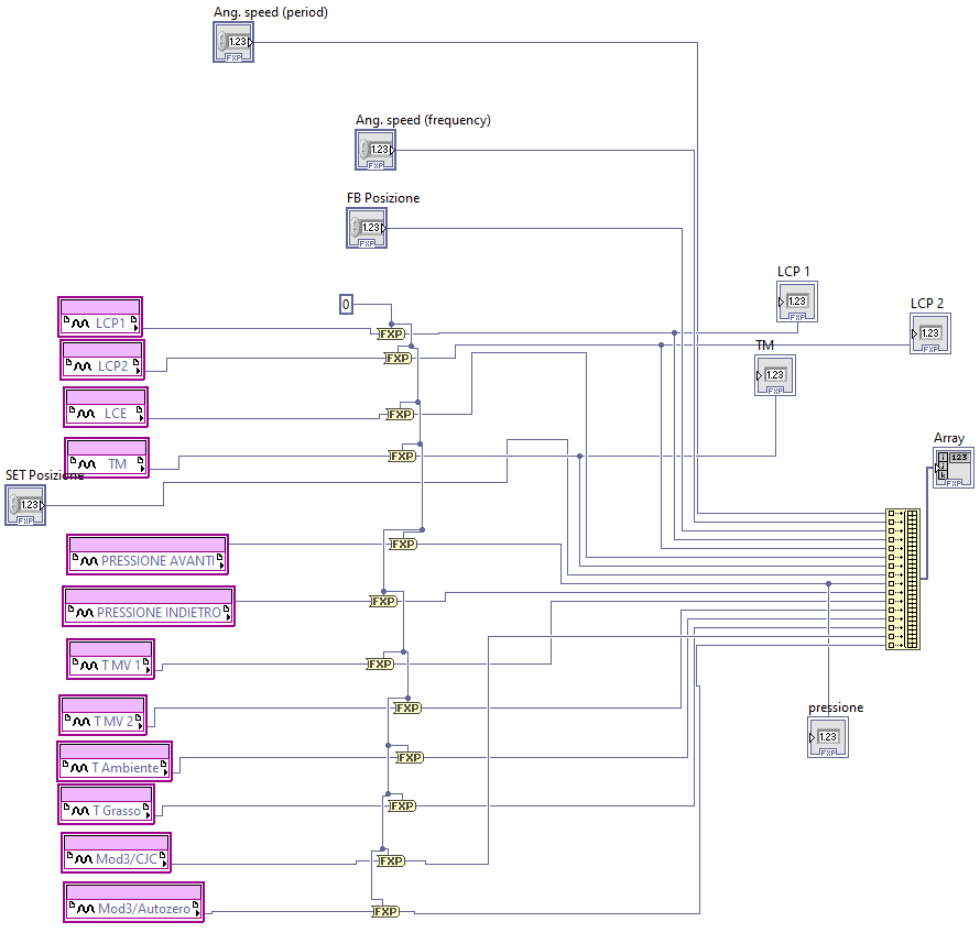


Figura 3.6: subVi di acquisizione segnali.

Data type	Details
Single-precision, floating-point numeric (SGL)	32 bit e 6 cifre significative
Double-precision, floating-point numeric (DBL)	64 bit e 15 cifre significative
Complex single-precision, floating-point numeric (CSG)	SGL with a real and an imaginary part.
Complex double-precision, floating-point numeric (CDB)	DBL with a real and an imaginary part.
8-bit signed integer numeric (I8)	A positive or negative integer stored using 8 bits, or a single byte.
16-bit signed integer numeric (I16)	A positive or negative integer stored using 16 bits, or two bytes.
32-bit signed integer numeric (I32)	A positive or negative integer stored using 32 bits, or four bytes.
64-bit signed integer numeric (I64)	A positive or negative integer stored using 64 bits, or eight bytes.
8-bit unsigned integer numeric (U8)	A positive integer stored using 8 bits, or a single byte.
16-bit unsigned integer numeric (U16)	A positive integer stored using 16 bits, or two bytes.
32-bit unsigned integer numeric (U32)	A positive integer stored using 32 bits, or four bytes.
64-bit unsigned integer numeric (U64)	A positive integer stored using 64 bits, or eight bytes.
timestamp	A time and date.
enumerated type	A list of items from which to select.
Boolean	A TRUE/FALSE value.
string	A series of text characters.
array	A collection of multiple pieces of data, each of which must be the same data type.
cluster	A collection of multiple pieces of data, each of which can be a different data type.
error cluster	A cluster of information that specifies error conditions returned by a node.
path	The location of a file or directory.
waveform	A cluster of information that specifies an array of data, the start time at which the data was collected, and the change in time between each measurement.
digital waveform	A cluster of information that specifies a digital table of data, the start time at which the data was collected, and the change in time between each measurement.
reference number (refnum)	A unique identifier for an object, such as a file or a task.
variant	A generic container for all other types of data.

Tabella 3.1: Caratteristiche dei tipi di Variabili in Labview.

vettori di cluster (i quali possono a loro volta essere costituiti da vettori o da variabili di varia natura). Di seguito è riportata una tabella contenente tutti i tipi di variabili, e i valori che possono assumere.

I fixed point sono un tipo di dati numerici razionali il cui intervallo e precisione vengono definiti configurando il numero di bit memorizzati e il valore assegnato a tali bit. I numeri a virgola fissa sono importanti per l'implementazione di programmi FPGA perché molti FPGA non supportano in modo efficiente i tipi di dati in virgola mobile e perché i numeri in virgola fissa consentono di utilizzare il numero minimo di bit per ottenere l'intervallo e la precisione richiesti dal proprio algoritmo. Un numero in virgola fissa è composto da un numero specificato di bit e da un punto binario. I numeri a virgola fissa possono essere firmati o non firmati. La posizione del punto binario rispetto al numero totale di bit memorizzati determina il valore di posizione assegnato a ciascun bit. Considera uno scenario che utilizza un insieme arbitrario di quattro bit. Il valore del set di bit 1001 cambia a seconda di dove si imposta il punto binario. Il valore del numero 10.01 differisce dal valore del numero 1.001 perché il punto binario determina il valore posizionale di ciascun bit. L'immagine seguente dimostra questo principio per un numero a virgola fissa senza segno che ha un numero positivo di bit memorizzati sia per la parte intera che per quella frazionaria.

I numeri a virgola fissa sono senza segno o con segno. I numeri a virgola fissa senza segno rappresentano solo valori positivi. I numeri a virgola fissa con segno possono rappresentare sia valori positivi che negativi. Per i numeri a virgola fissa con segno, il bit più a sinistra del set di bit memorizzato determina se il valore dei bit è positivo o negativo. Se il primo bit memorizzato è 1, il valore dei bit è negativo. Se il primo bit memorizzato è 0, il valore dei bit è positivo. I numeri a virgola fissa con segno vengono interpretati utilizzando una codifica chiamata

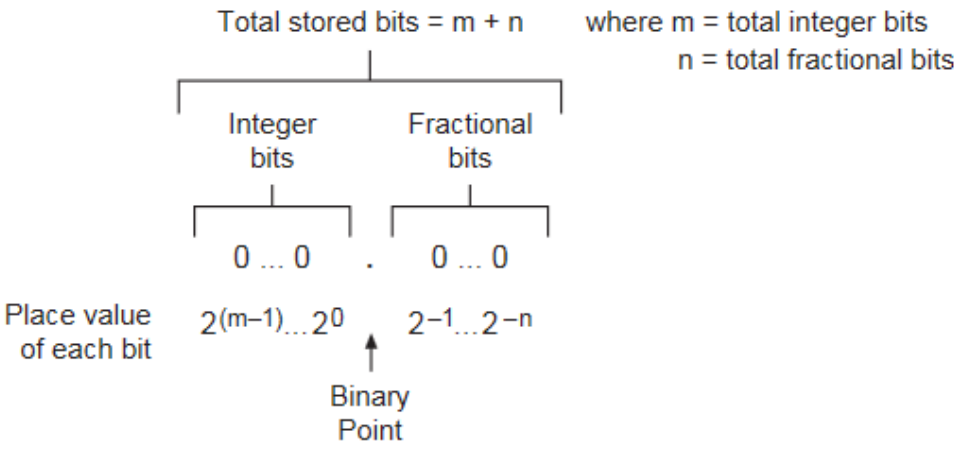


Figura 3.7: Schema sui fixed point e sul modo di impostarli.

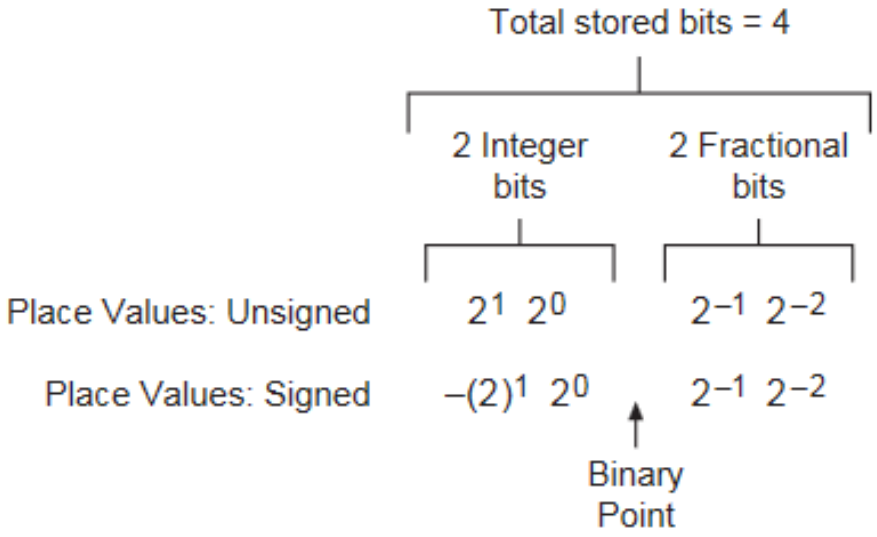


Figura 3.8: esempio fixed point signed ed unsigned.

complemento a due. L'immagine seguente mostra un numero a virgola fissa di esempio con due bit interi e due bit frazionari. Notare che gli esponenti di tutti i valori posizionali rimangono gli stessi tra i numeri con segno e senza segno, ma in un numero a virgola fissa con segno, il valore della posizione più a sinistra è negativo. Usando l'immagine precedente, considera uno scenario in cui entrambe le istanze con segno e senza segno del numero a virgola fissa contengono i bit 1011. Il risultato della moltiplicazione di ciascun bit per il suo valore posizionale e dell'aggiunta del valore risultante di ciascun bit insieme è il decimale valore per qualsiasi insieme di bit in un numero a virgola fissa. Per l'istanza senza segno di questo numero a virgola fissa, il valore decimale dei bit 1011 è uguale a 1 × 21 + 0 × 20 + 1 × 2-1 + 1 × 2-2 o 2,75. Per l'istanza con segno di questo numero a virgola fissa, il valore decimale dei bit 1011 è uguale a 1 × (-21) + 0 × 20 + 1 × 2-1 + 1 × 2-2 o -1,25.

Per qualsiasi numero a virgola fissa con segno e senza segno con lo stesso numero intero e frazionario di bit, il tipo senza segno è in grado di rappresentare valori positivi più grandi, mentre il tipo con segno può rappresentare sia valori positivi che negativi. Ad esempio, tra le istanze con segno e senza segno di un numero a virgola fissa a 4 bit con due interi e due bit frazionari, l'istanza senza segno ha un intervallo da 0 a 3,75, mentre l'istanza con segno ha un intervallo da -2 a 1,75. L'aumento del numero totale di bit memorizzati per qualsiasi numero a virgola fissa aumenta l'intervallo che il tipo di dati può rappresentare. L'aumento del numero di bit frazionari archiviati per qualsiasi numero a virgola fissa aumenta la precisione che il numero può memorizzare. L'intervallo di un numero a virgola fissa

è l'insieme di valori che un particolare tipo di dati a virgola fissa può rappresentare. Per determinare l'intervallo di un numero a virgola fissa, è necessario conoscere i valori massimo e minimo che il numero può rappresentare e l'incremento tra ciascun valore. L'incremento tra ciascun valore è anche chiamato precisione, o delta, del numero a virgola fissa. Considera un numero a virgola fissa senza segno a 2 bit che memorizza due bit interi e zero bit frazionari. Poiché ogni bit memorizzato può contenere uno 0 o 1, il numero a 2 bit può rappresentare quattro combinazioni univoche: 00, 01, 10 o 11. Il valore posizionale del bit sinistro in questo esempio è 21 e il valore posizionale del bit di destra è 20. I numeri a virgola fissa aiutano a controllare la spesa delle risorse sull'hardware eliminando i requisiti logici per gli spostamenti della gamma dinamica inerenti alle operazioni in virgola mobile. Molti FPGA non supportano l'aritmetica in virgola mobile o non possono elaborare le operazioni in virgola mobile in modo efficiente. Senza un'unità a virgola mobile (FPU) dedicata, l'aritmetica in virgola mobile richiede risorse logiche significative sull'hardware, aumenta il numero di cicli di clock richiesti per operazione e riduce l'efficienza di elaborazione del dispositivo. I numeri a virgola fissa offrono i seguenti vantaggi:

- Non è necessaria la logica per supportare gli spostamenti dinamici: le operazioni in virgola mobile richiedono uno spostamento dinamico, o float, dell'esponente utilizzato per ridimensionare il significante, o numero di base, in fase di esecuzione. Con i numeri a virgola fissa, il valore dell'esponente è definito dal tipo di dati a virgola fissa e non viene calcolato in fase di esecuzione. Sebbene i numeri a virgola fissa abbiano un intervallo più limitato rispetto ai numeri a virgola mobile con un numero equivalente di bit, i numeri a virgola fissa eliminano la necessità per la logica hardware di calcolare ed eseguire spostamenti dinamici.

- Maggiore controllo dei costi delle risorse FPGA: il tipo di dati a virgola fissa consente di esercitare un maggiore controllo delle risorse FPGA specificando dimensioni di bit non standard e opzioni di coercizione che possono conservare la memoria e le risorse logiche. Specificando le lunghezze delle parole e i comportamenti di coercizione per tutti i terminali e gli operatori nel codice progettato per l'esecuzione su hardware, ottimizzi le prestazioni del tuo dispositivo o sistema.

Per i motivi sopra riportati, nel modulo FPGA è comodo utilizzare i fixed point; infatti, tutti i segnali che acquisiamo dai moduli Ni e in generale tutto ciò che viene scritto sotto forma di fixed point viene convertito in un fixed point di tipo signed, con 24 integer bits e 13 fraction bits, poiché il modulo delle termocoppie richiede un numero di bit pari a 24 e perché il numero maggiore che si può rappresentare in questo modo è consono con quanto andremo a misurare o a inviare come SET.

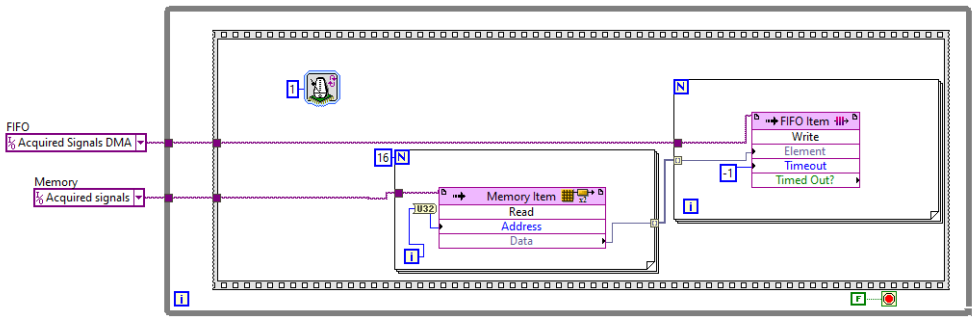


Figura 3.9: DMA FIFO.

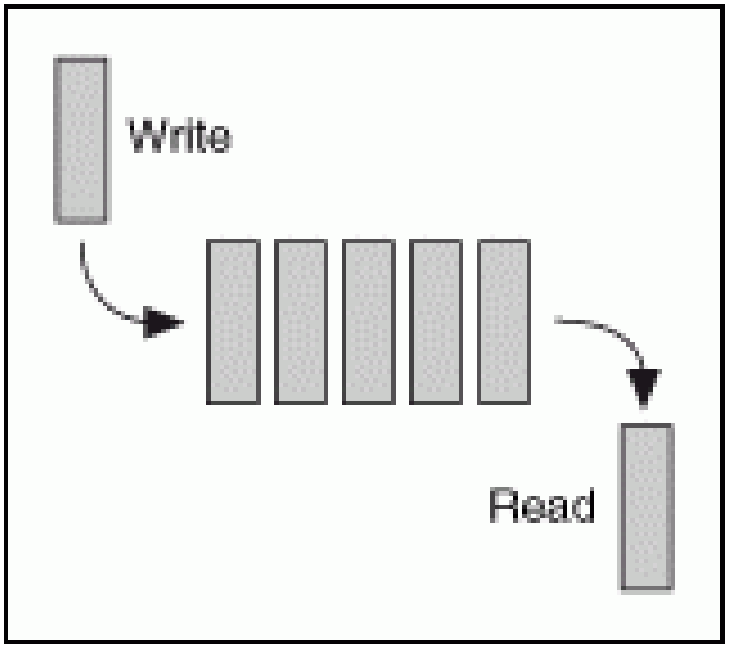


Figura 3.10: Modalità di scrittura e lettura FIFO.

Ora passiamo alla parte che costituisce la comunicazione con il sistema Real Time, ovvero i DMA FIFO. Anche qui è presente un ciclo while che esegue con una frequenza di 1 kHz, al suo interno sono presenti due cicli for, nei quali vengono prima letti i valori dei segnali acquisiti che abbiamo scritto nella parte precedente e poi vengono scritti tramite la funzione FIFO write. Andiamo ora a spiegare nel dettaglio che cosa sono i FIFO. Una FIFO è una struttura di dati che contiene gli elementi nell'ordine in cui vengono ricevuti e fornisce l'accesso a tali elementi utilizzando una base first-in, first-out. L'illustrazione seguente mostra il comportamento degli elementi che si muovono attraverso una FIFO.

Il tipo di FIFO che crei dipende dal modo in cui devi trasferire i dati.

- Tra i loop controllati dal clock (FIFO locale): Puoi accedere a FIFO locali creati tramite una raccolta di risorse (.grsc) su più documenti nel tuo progetto. Per condividere i documenti che fanno riferimento a tale FIFO con un altro utente, è necessario inviare la raccolta di risorse ei documenti. Per creare singoli VI FPGA all'interno di un singolo documento che è possibile inviare ad altri utenti, utilizzare il nodo Crea FIFO per creare un FIFO locale all'interno del VI.

- Tra il processore host e l'FPGA (DMA FIFO): Utilizza FIFO DMA per lo streaming di dati tra un processore host e l'FPGA. Un FIFO DMA alloca memoria sia sul computer host che sul target FPGA, ma agisce come un singolo FIFO per sfruttare le risorse di ciascun dispositivo.

- Tra due target FPGA o tra un target FPGA e un target non FPGA (FIFO peer-to-peer):i FIFO peer-to-peer scrivono dati su un FIFO writer peer-to-peer su un target e leggono dati da un lettore peer-to-peer su un altro obiettivo. Per definire FIFO peer-to-peer, utilizzare una raccolta di risorse. I FIFO di scrittura e lettura sono accoppiati nel VI host, ma l'effettivo trasferimento dei dati avviene direttamente da un target all'altro.

3.1.1 Acquisizione dei segnali digitali degli encoder

Di seguito viene riportata la spiegazione di come si passa dai segnali digitali che acquisiamo dagli encoder ad un segnale digitale di posizione o di velocità. L'encoder incrementale è un dispositivo che fornisce impulsi elettrici se il suo albero sta ruotando. Il numero degli impulsi generati è proporzionale alla posizione angolare dell'albero. L'encoder incrementale è uno dei trasduttori di posizione più utilizzati attualmente. Il principio di un encoder incrementale ottico è presentato in Fig.3.11. Insieme all'albero c'è una rotazione adisco rotore trasparente (solitamente in vetro) con una pista graduata circolare realizzata come una sequenza periodica di zone radiali trasparenti e non trasparenti che modula i fasci luminosi emessi da una sorgente luminosa posta su un lato del disco sulla parte fissa (statore) dell'encoder. Sul lato opposto i fasci di luce modulati vengono rilevati da due gruppi di sensori ottici ed elaborati da circuiti elettronici. Ciascuna delle due uscite dell'encoder (notate A e B) sarà generare un impulso quando l'albero ruota di un angolo pari al passo angolare di gradazione theta , cioè l'angolo secondo una successiva trasparente e non zona trasparente. Il numero di impulsi (contati solitamente da un'elettronica esterna contatori) è proporzionale alla posizione angolare dell'albero. Per il fatto che i fasci di luce sono posti spostati l'uno sull'altro con un angolo pari al quarto di passo angolare di graduazione theta/4, gli impulsi delle due

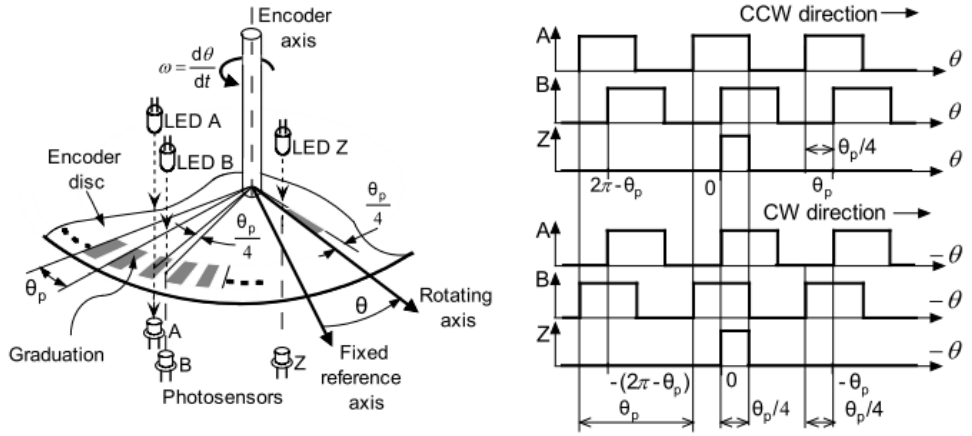


Figura 3.11: Schema di funzionamento di un encoder incrementale.

From CCW to CW Q=1 to Q=0 (Triggered by R)		From CW to CCW Q=1 to Q=0 (Triggered by S)	
Occurs if		Occurs if	
A	*B*	*A*	*B*
0	0→1	0→1	0
0→1	1	0	1→0
1→0	0	1	0→1
1	1→0	1→0	1

Figura 3.12: Schema di calcolo degli incrementi e del senso di rotazione dell'encoder.

uscite saranno anche spostati, rendendo possibile la determinazione del senso di rotazione. Un terzo fascio di luce è modulato da un altro binario con un'unica graduazione. Il segnale risultante (denominato Z) associato a questo terzo raggio fornisce un singolo impulso nel corso di una rotazione completa (360°). La posizione dell'albero corrispondente a questo l'impulso può essere considerata come posizione di riferimento. La Fig. 40 mostra gli impulsi di uscita dall'encoder. I due segnali di uscita p/4 sfasati dell'encoder contengono implicitamente anche l'informazione di direzione che può essere ottenuta in modi diversi. Tenendo conto di tutte e quattro le possibili combinazioni dei segnali A e B per le inversioni, è possibile rilevare il cambio di direzione in tutti i casi durante una rotazione del minimo incremento rilevabile dell'angolo di rotazione theta/4. Nella tabella sottostante sono riportate tutte le combinazioni di segnali che rilevano l'inversione del senso di rotazione.

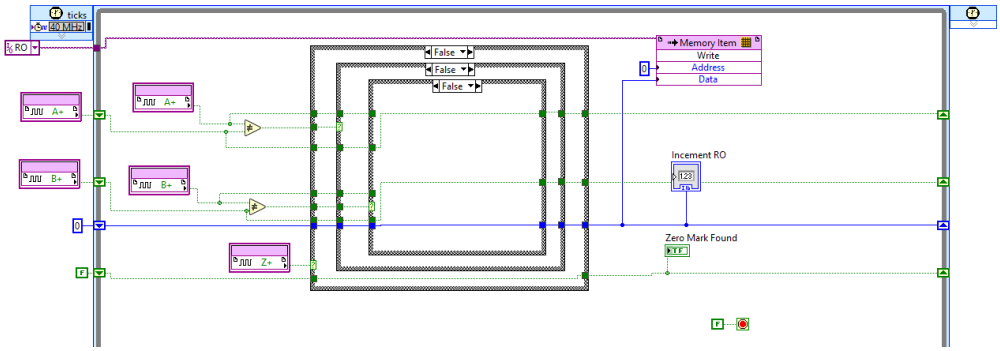


Figura 3.13: Sistema di rilevazione della posizione.

In figura 3.11 sono raffigurati i vari segnali che provengono dalla riga ottica, in particolare per determinare la posizione vengono utilizzati i segnali A+, B+ e Z+. Quanto descritto in precedenza e rappresentato nella fig. 3.12 è stato riportato in codice labview tramite un single cycle timed loop, al cui interno sono presenti varie case structures, le quali, in base a come i segnali booleani variano nel tempo. Questo tipo di ciclo è come un ciclo while, con la differenza fondamentale che esso è obbligato ad eseguire ripetutamente con la frequenza impostata dall'utente in fase di programmazione (che in questo caso è pari a 40 MHz). Tramite degli shift registers, tra un ciclo e l'altro vengono confrontati due valori successivi di un segnale booleano, e quando viene rilevato un cambiamento tramite una funzione "not equal", si aggiunge o sottrae una unità al contatore degli incrementi, in base alle relazioni riportate in tabella 9. Il tutto viene scritto in una memoria "RO", la quale veniva letta nella sezione precedente. Ora viene illustrato il metodo di calcolo del periodo di rotazione dell'encoder incrementale del torsiometro e il successivo calcolo della velocità. I segnali generati dall'encoder incrementale forniscono anche informazioni relative alla velocità di rotazione. Sono noti alcuni metodi di base che vengono discussi di seguito.

3.1.2 Identificazione della velocità basata sulla misurazione della frequenza

La frequenza degli impulsi dell'encoder è proporzionale alla velocità angolare. Il numero di impulsi Δ N viene contato durante un periodo di campionamento fisso noto Ts. La velocità angolare è determinata dall'espressione:

$$\omega = \frac{d\Theta}{dt} \cong \frac{d\Theta}{T_s} \cong \frac{2\pi\Delta N}{N_r T_s} = \frac{\Theta_p \Delta N}{T_s} \tag{3.1}$$

A causa della mancanza di sincronizzazione tra il periodo di campionamento e gli impulsi dell'encoder, si verifica un errore di quantizzazione. L'errore relativo della

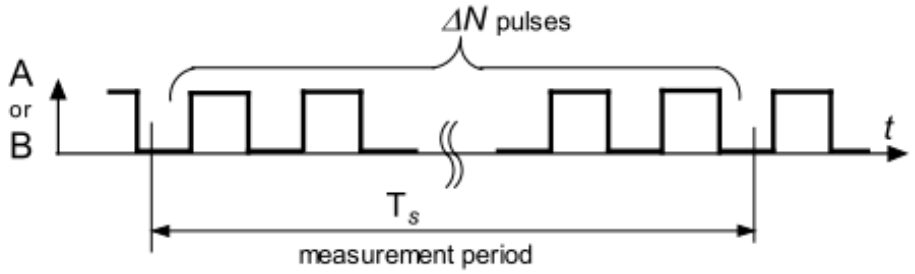


Figura 3.14: Metodo di calcolo della velocità basato sulla frequenza.

procedura è dato da

$$\epsilon_f = \frac{2\pi}{\omega N_r T_s} \tag{3.2}$$

e dipende dal reciproco della velocità, dall'intervallo di misura e dalla risoluzione dell'encoder. L'identificazione della velocità basata sulla misurazione della frequenza produce errori relativamente piccoli ad alta velocità perché il numero di impulsi dall'encoder nell'intervallo di tempo di misurazione è elevato.

3.1.3 Identificazione della velocità basata sulla misurazione del periodo

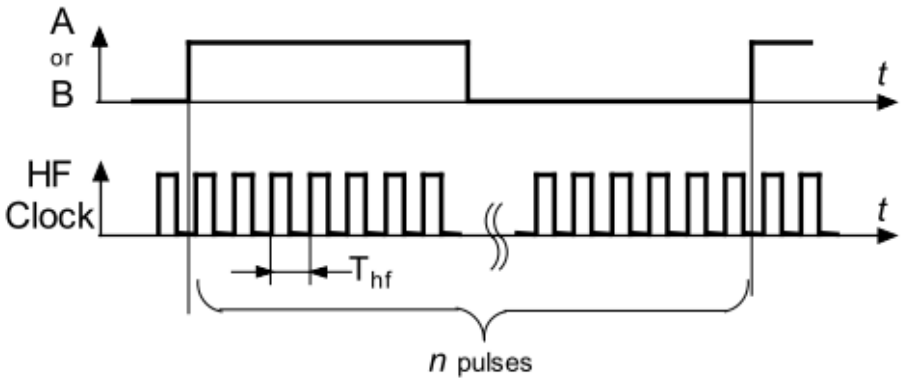


Figura 3.15: Metodo di calcolo della velocità basato sul periodo.

L'identificazione della velocità basata sulla misurazione della frequenza a bassa velocità non è più un'opzione. Una soluzione migliore è l'identificazione della velocità basata sulla misurazione del periodo. Il metodo consiste nel contare gli impulsi di un segnale di clock ad alta frequenza (avente periodo Thf) durante un periodo dell'encoder, come mostrato in fig.3.15. In questo caso l'espressione della velocità angolare è

$$\omega = \frac{d\Theta}{dt} \cong \frac{d\Theta}{nT_h f} \cong \frac{2\pi}{T_h f N_r n} \tag{3.3}$$

dove n rappresenta il numero contato degli impulsi ad alta frequenza. L'errore relativo aumenta con la frequenza di rotazione ed è dato da

$$\epsilon_f = \frac{\omega N_r T_h f}{2\pi} \tag{3.4}$$

Per l'encoder con cui acquisiamo i segnali in questo banco prova, il metodo di calcolo della velocità basato sul periodo risulta essere quello più accurato. Di seguito è

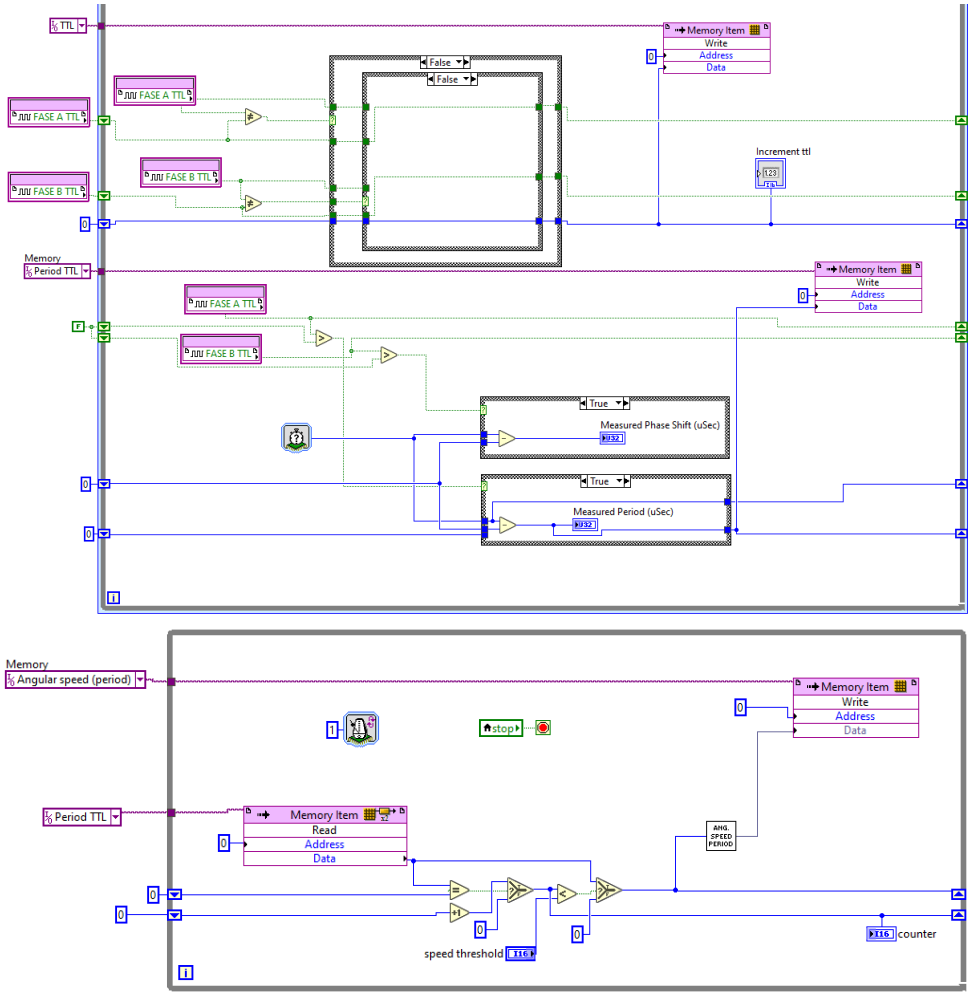


Figura 3.16: Calcolo della velocità basato sul periodo.

mostrato come viene implementato questo calcolo. Nella parte di codice all'interno del single cycle timed loop notiamo che è presente una parte dedicata al calcolo del periodo di cui si parla nelle formule precedenti (in particolare, si calcola il prodotto n*Thf), poi viene salvato in una memoria, viene letto nel ciclo while sottostante e viene calcolata la velocità tramite una subVI. Di seguito è riportata la struttura della subVI. In figura 3.16 si può notare anche una sezione dedicata all'azzeramento della velocità, che avviene quando viene rilevato un periodo molto grande. Tale operazione è gestita tramite dei selettori e tramite altre funzioni booleane. È inoltre possibile andare a controllare la soglia di velocità al di sotto della quale l'encoder azzera direttamente il valore di output. Nella figura sottostante viene mostrata l'ultima parte del diagramma a blocchi del modulo fpga, quella dove vengono

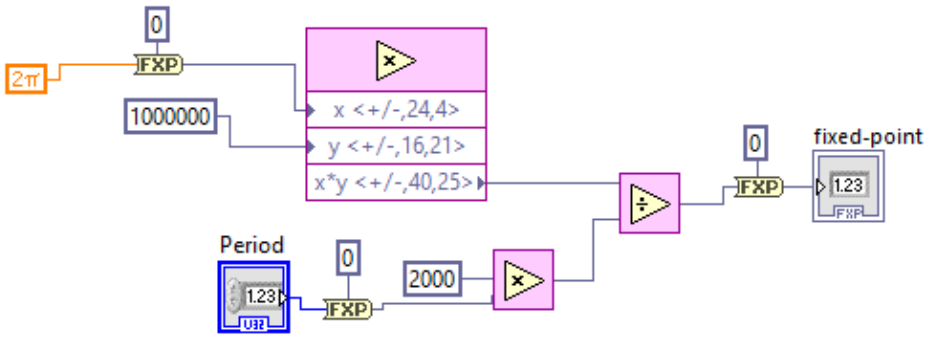


Figura 3.17: Calcolo della velocità.

inviati i segnali booleani al motore elettrico per il suo funzionamento. Come si può osservare, tale parte è all'interno di un ciclo while, e sono presenti vari controlli, gestibili tramite il pannello frontale, che vengono inviati ai canali di output digitale presenti nel modulo NI 9375. Il pannello frontale per la parte fpga viene usato principalmente per gestire il tipo di SET da inviare al motore Lenze e per i segnali digitali sopra citati. Di seguito è riportata una schermata del pannello frontale. In particolare, è possibile abilitare il controllo del motore tramite il bottone "Control enable (RFR)", è possibile attivare e abilitare la funzione di homing e volendo si può anche comandare manualmente il motore tramite i bottoni di spostamento manuale positivo e negativo.

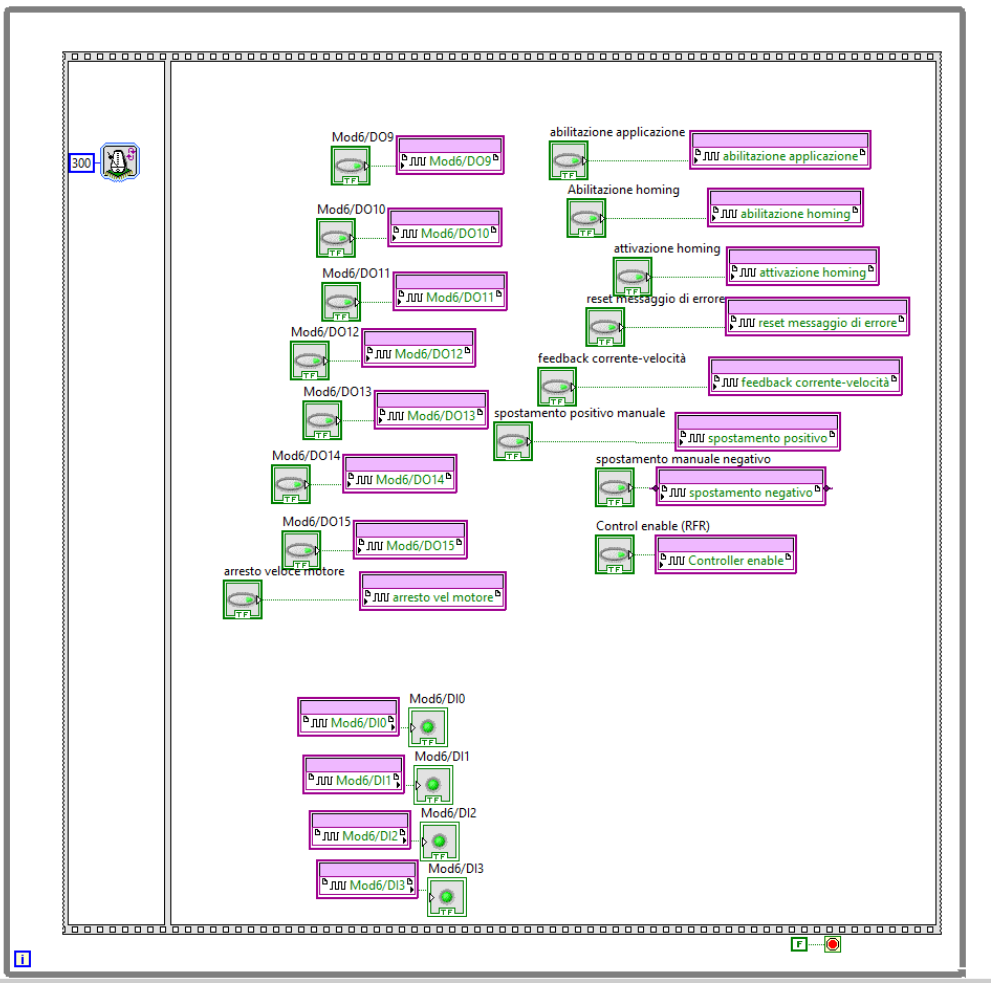


Figura 3.18: Sezione dedicata al controllo del motore elettrico.

3.2 Real Time Target

Andiamo ora ad illustrare le varie parti che costituiscono la parte di codice che risiede nel Real Time target. Come si può osservare dalla schermata del project explorer, è presente un file "Main RT", il quale contiene a sua volta varie subVI, utilizzate per semplificare la scrittura del codice, che ne risulterebbe altrimenti troppo confuso. Nella figura 3.20 è presente un ciclo while che ha il compito di inizializzare le variavili della VI e impostare il collegamento con il modulo FPGA tramite FIFO. Nella parte superiore dell'immagine vediamo che il primo step della case structure di inizializzazione utilizza un "property node", ovvero uno strumento molto utile per gestire i consensi e le proprietà di variabili sparse per il codice, senza ricorrere a wires o collegamenti troppo confusi. Questo oggetto

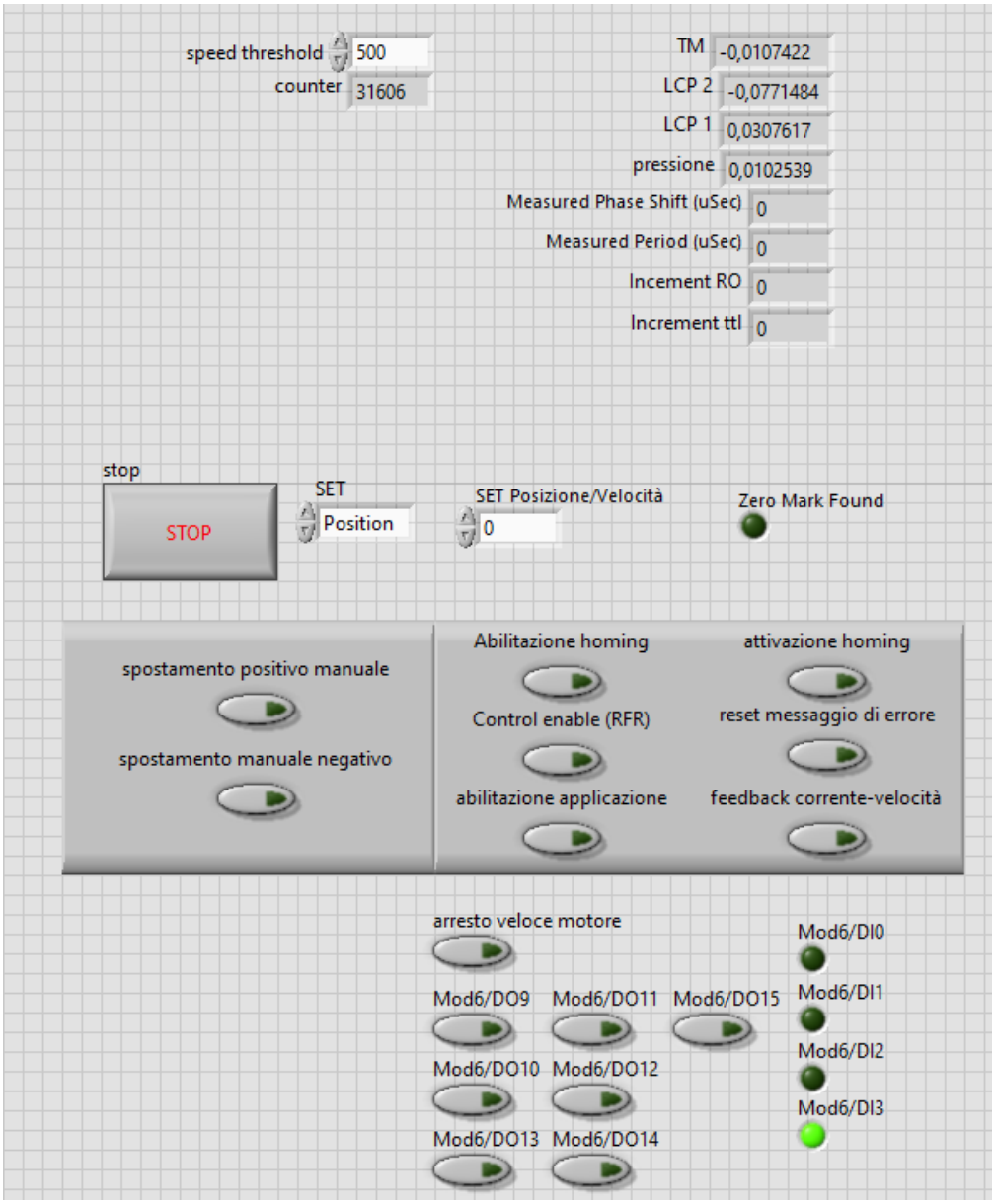


Figura 3.19: Pannello Frontale del file Main FPGA.

ottiene (legge) e/o imposta (scrive) le proprietà di un riferimento. È possibile utilizzare il property node per ottenere o impostare proprietà e metodi su istanze di applicazioni, VI e oggetti locali o remoti. È possibile anche usare il Property Node per accedere ai dati privati di una classe LabVIEW. Il Property Node si adatta automaticamente alla classe dell'oggetto a cui si fa riferimento. Per selezionare la classe su cui eseguire la proprietà, collegare il refnum all'ingresso di riferimento.

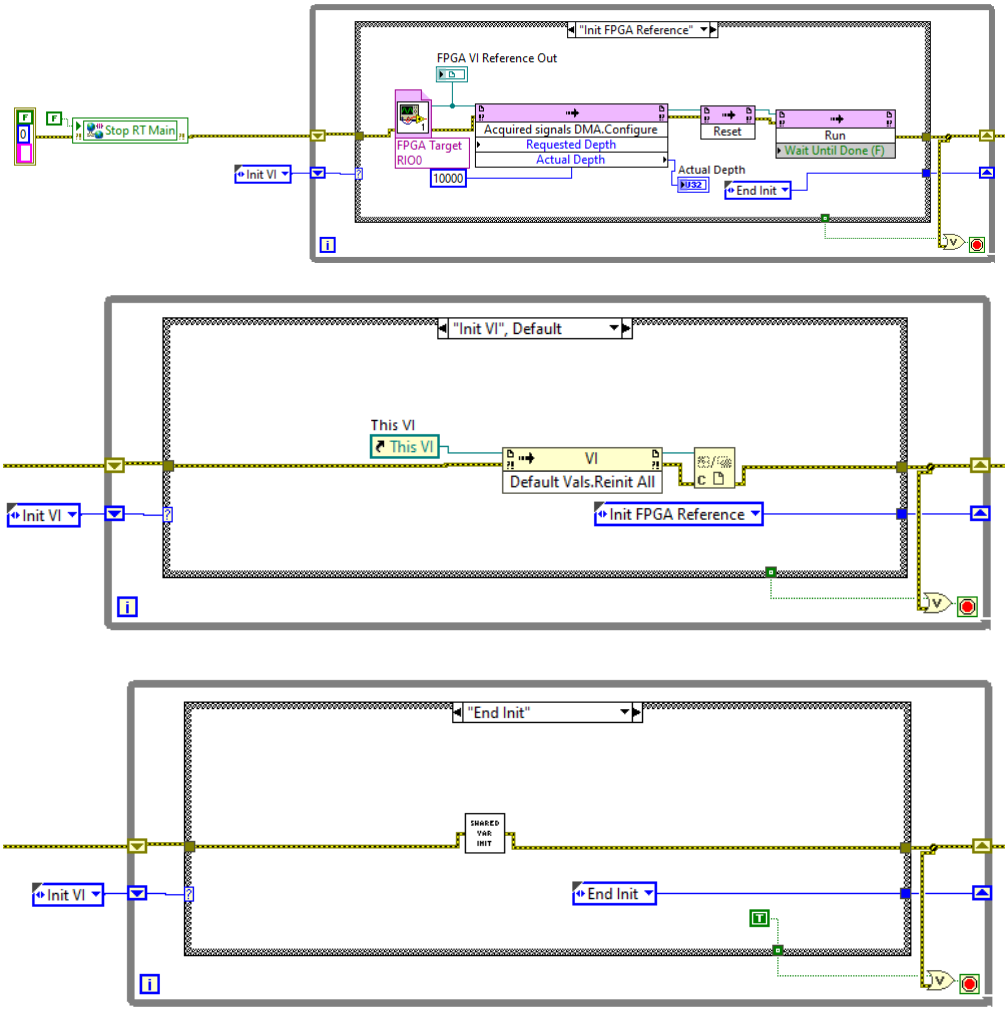


Figura 3.20: Ciclo di inizializzazione della VI.

Ad esempio, per selezionare la classe VI, Generic o Application, collegare il VI, l'oggetto VI o il riferimento dell'applicazione all'ingresso di riferimento. Il nodo si adatta automaticamente alla classe. Puoi anche fare clic con il pulsante destro del mouse sul nodo e selezionare una classe dal menu di scelta rapida. Puoi collegare una classe LabVIEW all'ingresso di riferimento di un Property Node. Se la classe LabVIEW dispone di VI accessor a cui è possibile accedere tramite un Property Node, è possibile leggere o scrivere sui VI accessor utilizzando un Property Node. È possibile visualizzare rapidamente l'implementazione di una proprietà della classe LabVIEW se è disponibile il diagramma a blocchi del VI accessor. Per visualizzare l'implementazione di una proprietà della classe LabVIEW, fare clic con il pulsante

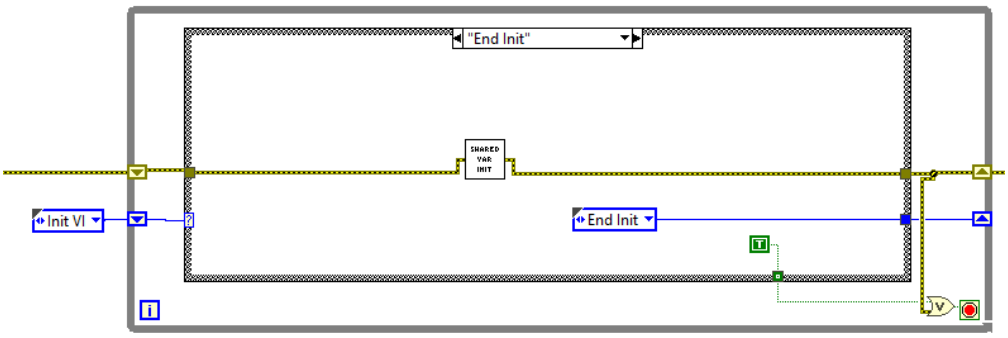


Figura 3.21: Property Node.

destro del mouse sulla proprietà e selezionare Open Accessor VI dal menu di scelta rapida. Se la proprietà è dinamica con più di un'implementazione, selezionando questa opzione viene visualizzata la finestra di dialogo Scegli implementazione. Utilizzare questa finestra di dialogo per visualizzare tutte le implementazioni della proprietà o del VI membro di invio dinamico e aprire una o più implementazioni. Si può spostare il cursore sui terminali nel Property Node per visualizzare ulteriori informazioni sulla proprietà nella finestra Context Help. È anche possibile fare clic con il pulsante destro del mouse su un terminale di proprietà e selezionare Guida per proprietà dal menu di scelta rapida, dove Proprietà è il nome della proprietà. Si possono leggere o scrivere più proprietà utilizzando un singolo nodo. Tuttavia, alcune proprietà non sono leggibili e altre non sono scrivibili. Utilizzare lo strumento Posizionamento per ridimensionare il Nodo Proprietà per aggiungere nuovi terminali. Una piccola freccia di direzione a destra della proprietà indica una proprietà che hai letto. Una piccola freccia di direzione a sinistra della proprietà indica una proprietà che scrivi. Fare clic con il pulsante destro del mouse sulla proprietà e selezionare Modifica in lettura o Modifica in scrittura dal menu di scelta rapida per modificare l'operazione della proprietà. Nel caso articolare di questa case structure il property node si usa per reinizializzare tutti i valori al loro stato di default, ma all'interno di questa Vi e di quella nel PC ce ne saranno molti altri con altri scopi. Lo step successivo della case structure è quello di creare un collegamento con la Vi dell'fpga e di impostare alcune proprietà di trasferimento dati tramite il FIFO, infine, tramite una subVI di inizializzazione delle shared variables, si danno i valori iniziali a queste variabili molto importanti nel codice. LabVIEW fornisce l'accesso a un'ampia varietà di tecnologie per la creazione di applicazioni distribuite. La variabile condivisa è un importante passo avanti nella semplificazione della programmazione necessaria per tali applicazioni. Usando la variabile condivisa, potete condividere dati tra loop su un singolo diagramma o tra VI attraverso la rete. A differenza di molti metodi di condivisione dei dati

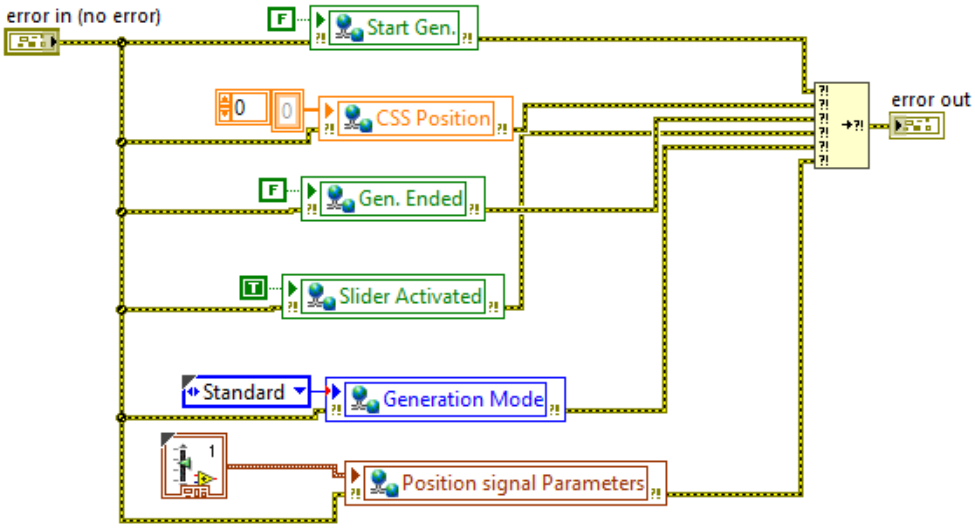


Figura 3.22: subVI di inizializzazione delle variabili.

esistenti in LabVIEW, come UDP/TCP, code di LabVIEW e FIFO in tempo reale, di solito si configura la variabile condivisa in fase di modifica utilizzando le finestre di dialogo delle proprietà e non è necessario includere il codice di configurazione nel proprio applicazione.

È possibile creare due tipi di variabili condivise: single-process e network-published. Se si fa clic con il pulsante destro del mouse su un target o su una cartella che non è all'interno di una libreria di progetto e si seleziona New»Variable dal menu di scelta rapida per creare una variabile condivisa, LabVIEW crea una nuova libreria di progetto e inserisce la variabile condivisa all'interno. Fare riferimento alla sezione Durata variabile condivisa per ulteriori informazioni su variabili e librerie. La figura 3.23 mostra la finestra di dialogo Proprietà variabile condivisa per una variabile condivisa a processo singolo. Il modulo LabVIEW Real-Time e il modulo LabVIEW Datalogging and Supervisory Control (DSC) forniscono funzionalità aggiuntive e proprietà configurabili alle variabili condivise. Sebbene in questo esempio siano installati sia LabVIEW Real-Time Module che LabVIEW DSC Module, è possibile utilizzare le funzionalità aggiunte dal LabVIEW DSC Module solo per le variabili condivise pubblicate in rete. È possibile selezionare da un gran numero di tipi di dati standard per una nuova variabile condivisa. Oltre a questi tipi di dati standard, è possibile specificare un tipo di dati personalizzato selezionando Personalizzato dall'elenco a discesa Tipo di dati e passando a un controllo personalizzato. Tuttavia, alcune funzionalità come il ridimensionamento e le FIFO in tempo reale non funzioneranno con alcuni tipi di dati personalizzati.

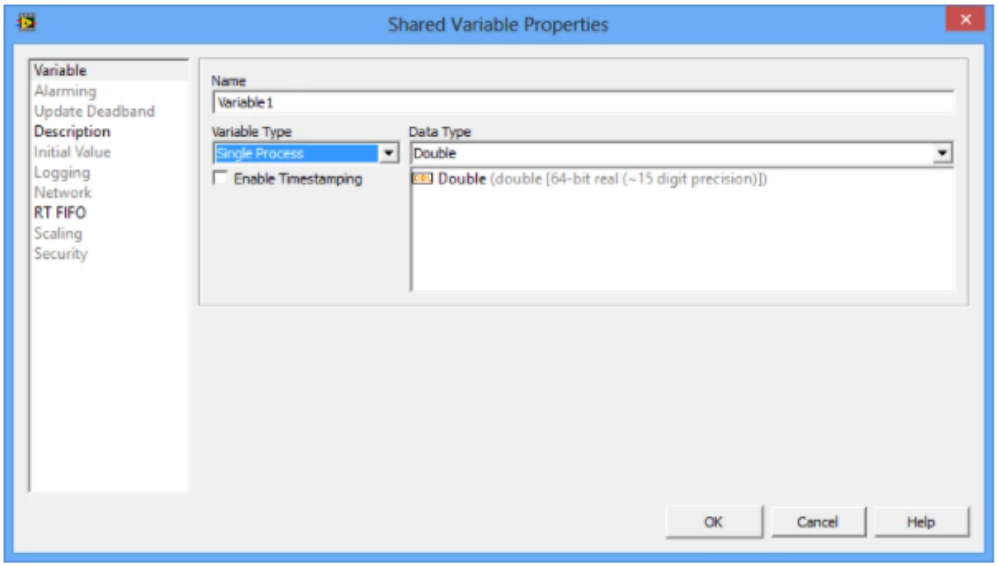


Figura 3.23: Proprietà delle network based shared variables.

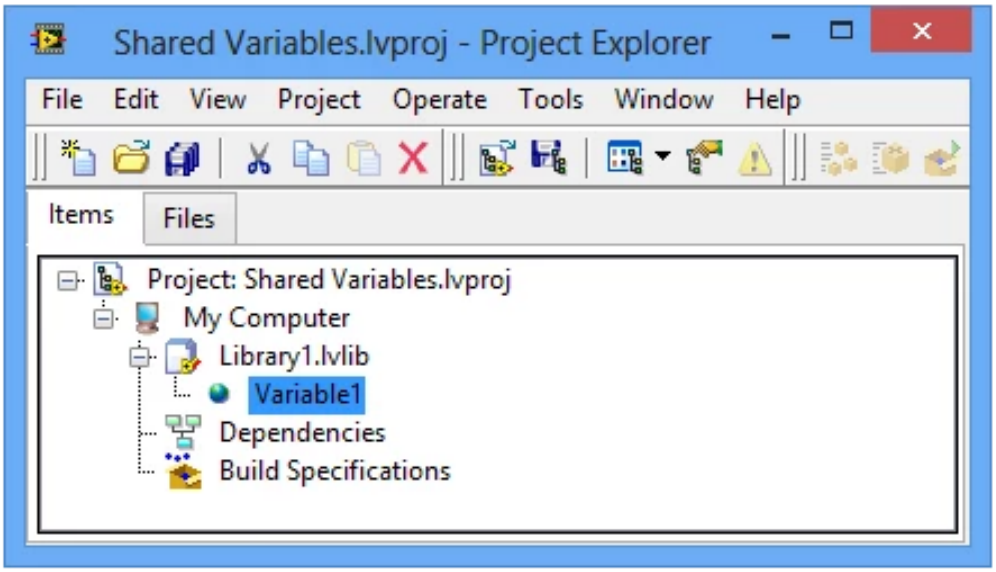


Figura 3.24: Individuare una variabile condivisa dal project explorer.

Dopo aver aggiunto una variabile condivisa a un progetto LabVIEW, è possibile trascinare la variabile condivisa nel diagramma a blocchi di un VI per leggere o scrivere la variabile condivisa. È possibile impostare un nodo Variabile condivisa come assoluto o relativo alla destinazione a seconda di come si desidera che il nodo si connetta alla variabile. Un nodo Variabile condivisa assoluta si connette alla

variabile condivisa sulla destinazione su cui è stata creata la variabile. Un nodo Shared Variable relativo al target si connette alla variabile condivisa sul target su cui si esegue il VI che contiene il nodo. Utilizzando variabili a processo singolo è possibile trasferire dati tra due diverse posizioni sullo stesso VI che non possono essere collegate tramite cavi, come loop paralleli sullo stesso VI o due VI diversi all'interno della stessa istanza dell'applicazione. L'implementazione sottostante della variabile condivisa a processo singolo è simile a quella della variabile globale LabVIEW. Il vantaggio principale delle variabili condivise a processo singolo rispetto alle variabili globali tradizionali è la capacità di convertire una variabile condivisa a processo singolo in una variabile condivisa pubblicata in rete a cui può accedere qualsiasi nodo su una rete. Per mantenere il determinismo, un'applicazione in tempo reale richiede l'uso di un meccanismo deterministico non bloccante per trasferire i dati da sezioni deterministiche del codice, come loop temporizzati con priorità più alta e VI con priorità time-critical, a sezioni non deterministiche del codice . Quando installi LabVIEW Real-Time Module, puoi configurare una variabile condivisa per utilizzare FIFO in tempo reale abilitando la funzione FIFO in tempo reale dalla finestra di dialogo Shared Variable Properties. National Instruments consiglia di utilizzare FIFO in tempo reale per trasferire i dati tra un loop time-critical e uno a bassa priorità. Potete evitare di utilizzare i VI FIFO in tempo reale di basso livello abilitando la FIFO in tempo reale su una variabile condivisa a processo singolo. Utilizzando la variabile condivisa pubblicata in rete, è possibile scrivere e leggere da variabili condivise su una rete Ethernet. L'implementazione della rete è gestita interamente dalla variabile pubblicata in rete.

Oltre a rendere disponibili i dati sulla rete, la variabile condivisa pubblicata in rete aggiunge molte funzionalità non disponibili con la variabile condivisa a processo singolo. Per fornire questa funzionalità aggiuntiva, l'implementazione interna della variabile condivisa pubblicata in rete è notevolmente più complessa di quella della variabile condivisa a processo singolo. Le prossime sezioni discutono gli aspetti di questa implementazione e offrono consigli per ottenere le migliori prestazioni dalla variabile condivisa pubblicata in rete. È possibile abilitare FIFO in tempo reale con una variabile condivisa pubblicata in rete, ma le variabili condivise pubblicate in rete abilitate per FIFO hanno un'importante differenza comportamentale rispetto alle variabili condivise a processo singolo in tempo reale, abilitate per FIFO. Ricordiamo che con la variabile condivisa a processo singolo, tutti gli autori e i lettori condividono un'unica FIFO in tempo reale; questo non è il caso della variabile condivisa pubblicata in rete. Ogni lettore di una variabile condivisa pubblicata in rete ottiene il proprio FIFO in tempo reale sia nel caso a elemento singolo che a elemento multiplo, come mostrato in seguito. Dopo il ciclo di inizializzazione, il codice si dirama in varie parti, le quali eseguono in parallelo funzioni di supporto e di trasmissione e postprocessing dei dati. La prima che andiamo ad analizzare è quella della figura sottostante, ovvero quella di

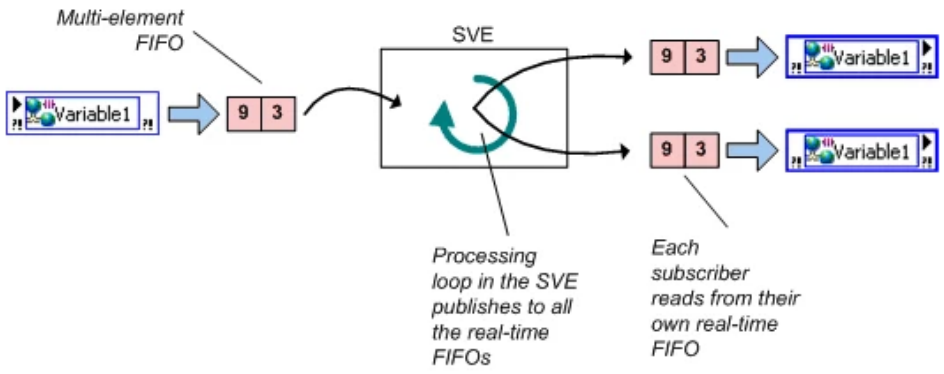


Figura 3.25: Real Time FIFO-Enabled Network Published variables.

supporto, che a seconda dello stato di connessione tra il dispositivo real Time e il PC, determina lo stato dei pulsanti sul pannello frontale. Queste parti sono tra di loro collegate anche tramite cavi di errore, in modo tale da poter ricevere sempre un messaggio di errore quando questo si verifica in qualsiasi parte del codice. In particolare, questo ciclo while utilizza un property node per ricevere informazioni sullo stato di connessione di PC e real time, e, a seconda della risposta del sistema, usa le variabili condivise per scrivere e aggiornare i valori degli indicatori sul pannello frontale tramite dei property nodes. Infine, scrive una costante booleana di vero o falso in un indicatore che comunica lo stato di connessione. La parte successiva è quella che viene mostrata in fig. 3.27, e si occupa di leggere i dati dal FIFO per poi inviarli in due direzioni, la prima è una subVI che sarà presente anche nel VI del PC, e serve a convertire i segnali dei sensori , quando necessario, nelle loro unità di misura appropriate per essere rappresentate graficamente. Di questa subVI vedremo un dettaglio in seguito, in quanto non viene utilizzata nella sua interezza in questa parte di codice. La seconda direzione è quella verso la funzione di "enqueue element", ovvero siutilizzano delle "code" per trasportare i dati da questa parte di codice ad una adiacente, la quale a sua volta li invierà al pc. Nella figura 56 vediamo come avviene il collegamento tra Real Time target e PC, per far si che i dati vengano mandati al PC si utilizzano i network stream.

I flussi di rete, introdotti in LabVIEW 2010, sono un metodo ideale per lo streaming di dati tra queste applicazioni. Utilizzando i flussi di rete, puoi condividere facilmente i dati attraverso la rete o sullo stesso computer. I flussi di rete sono un metodo di comunicazione dinamico, strettamente integrato e facile da configurare per il trasferimento di dati da un'applicazione a un'altra con caratteristiche di velocità e latenza paragonabili a TCP. Tuttavia, a differenza del TCP, i flussi di rete supportano direttamente la trasmissione di tipi di dati arbitrari senza la necessità di appiattire e smontare prima i dati in un tipo di dati intermedio. I flussi di rete

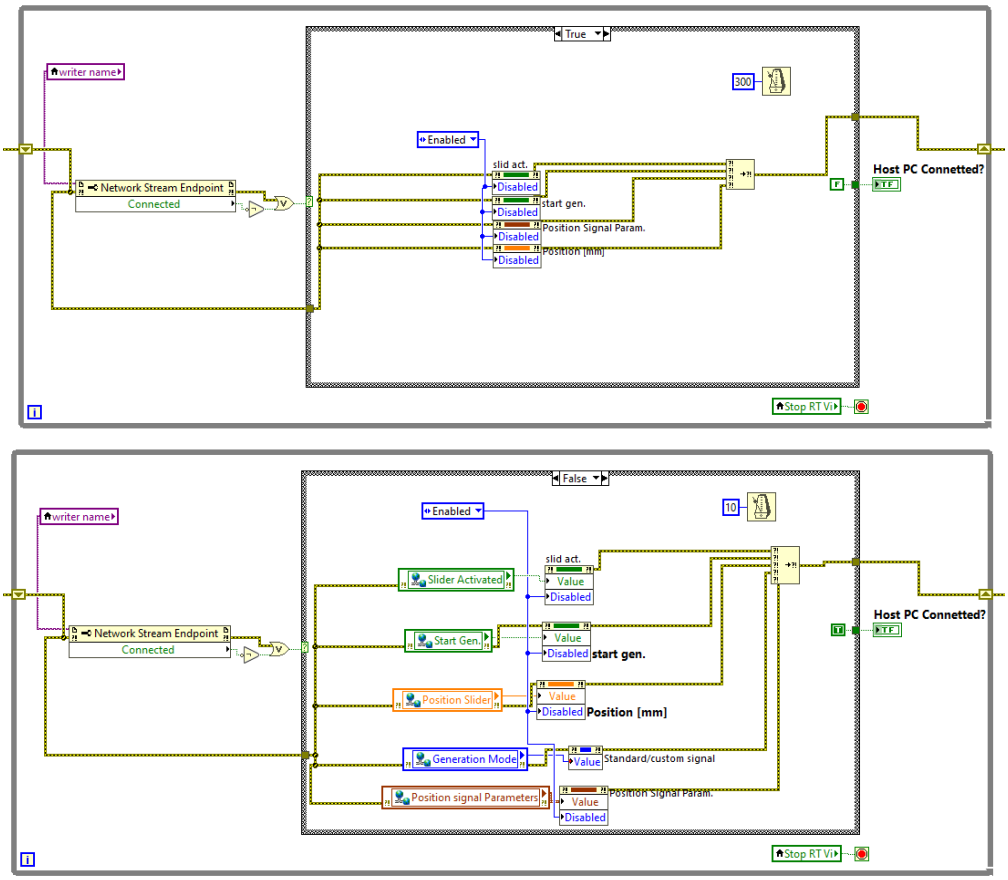


Figura 3.26: Ciclo while di aggiornamento parametri.

appiattiscono i dati in modo compatibile con le versioni precedenti, consentendo alle applicazioni che utilizzano diverse versioni del motore runtime di LabVIEW di comunicare tra loro in modo sicuro e corretto. I flussi di rete hanno anche una gestione della connessione avanzata che ripristina automaticamente la connettività di rete se si verifica una disconnessione a causa di un'interruzione della rete o di un altro errore di sistema. I flussi utilizzano una strategia di comunicazione bufferizzata e senza perdite che garantisce che i dati scritti nel flusso non vengano mai persi, anche in ambienti con connettività di rete intermittente. I flussi di rete sono stati progettati e ottimizzati per una comunicazione dati senza perdite e ad alto throughput. I flussi di rete utilizzano un modello di comunicazione bufferizzato punto-punto unidirezionale per trasmettere i dati tra le applicazioni. Ciò significa che uno degli endpoint è lo scrittore dei dati e l'altro è il lettore. È possibile realizzare una comunicazione bidirezionale utilizzando due flussi, in cui ogni computer contiene un lettore e un writer abbinati a un writer e un lettore sul

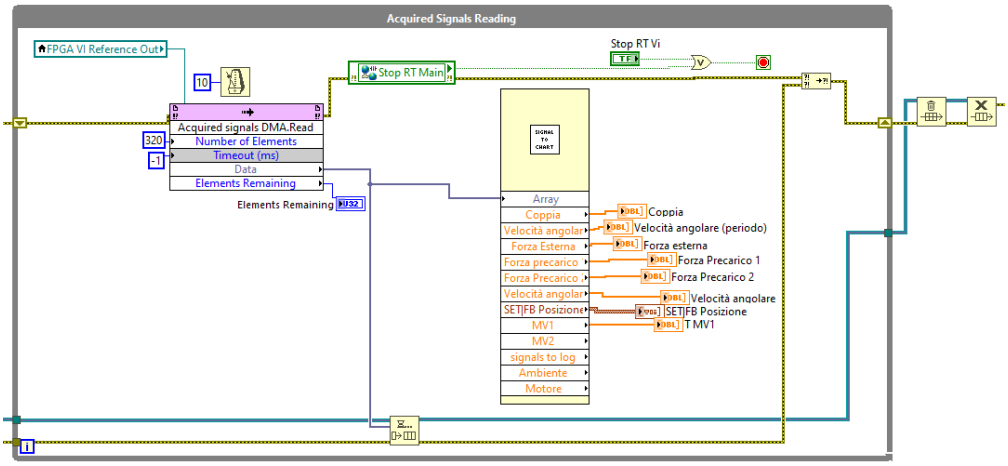


Figura 3.27: Ciclo di lettura del segnale dal FIFO.

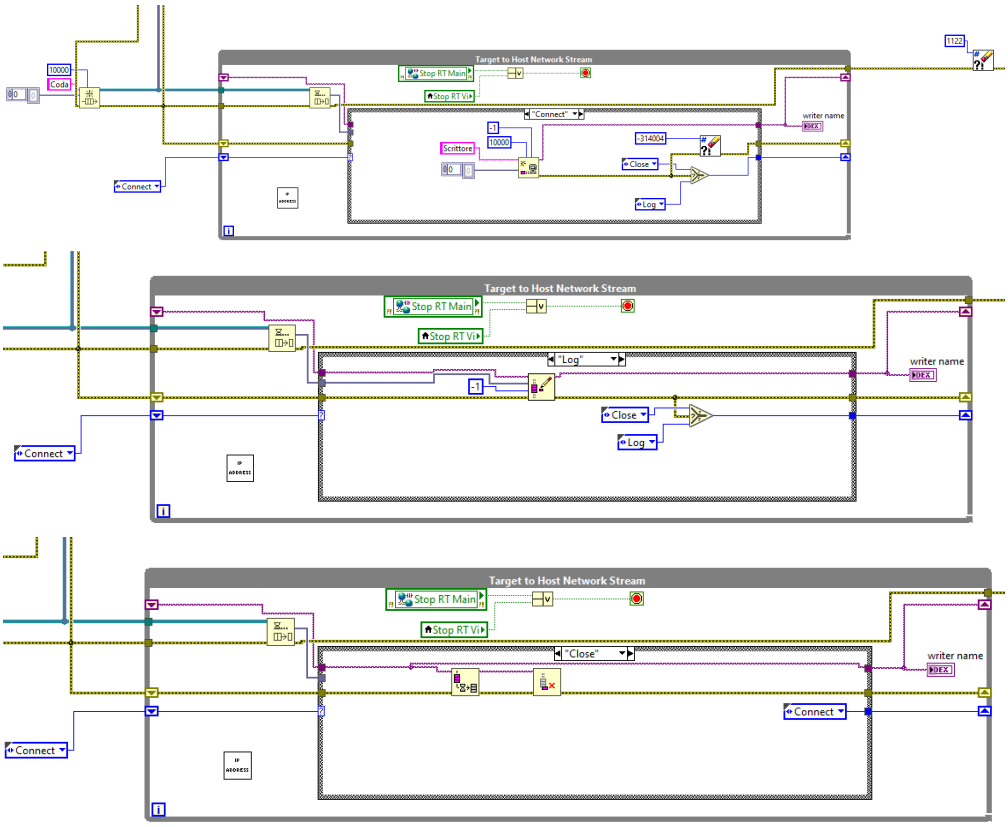


Figura 3.28: Ciclo while di Network stream.

computer opposto. Poiché i flussi sono stati creati con caratteristiche di throughput paragonabili a quelle del TCP grezzo, gli stream sono ideali per applicazioni ad alto throughput in cui il programmatore non vuole affrontare la complessità dell'utilizzo di TCP. I flussi possono essere utilizzati anche per comunicazioni senza perdita di dati a basso throughput come l'invio e la ricezione di comandi. Tuttavia, l'utilizzo di flussi per comunicazioni a bassa velocità effettiva può richiedere una gestione più esplicita quando i dati vengono trasmessi attraverso il flusso se si desidera la latenza più bassa assoluta. In figura 3.28 ciò viene implementato tramite una case

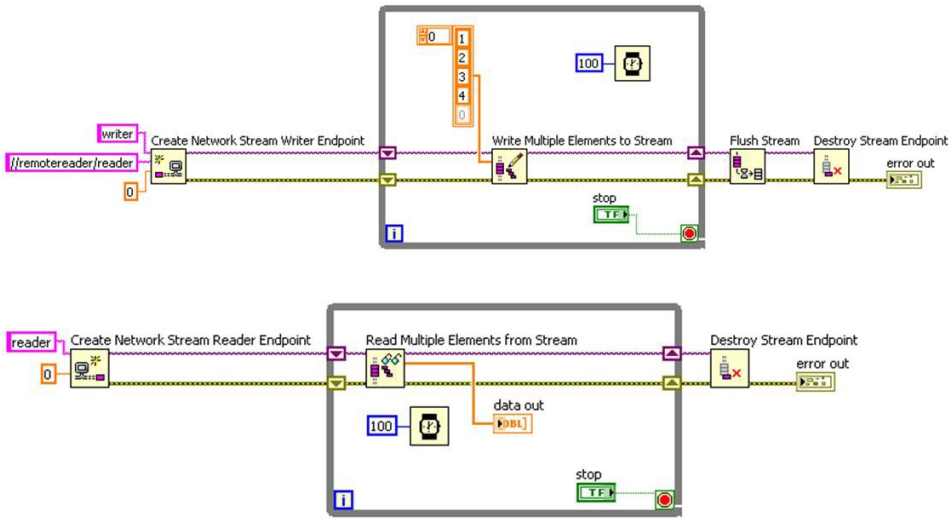


Figura 3.29: Esempio di network stream.

structure che prima verifica che ci sia la connessione tra gli hardware, e in seguito invia i dati che gli arrivano dalla coda tramite la funzione di "log". Un'altra parte importante di questa parte di codice è la subVI che identifica l'indirizzo IP e lo salva in una variabile condivisa, che poi verrà letta nel PC. L'indirizzo Ip serve per comporre l'URL che è necessario per far connettere i due target e streammare i dati. L'ultima parte del codice Real Time è quella del "Generation Loop", ovvero dove i segnali di SET generati tramite slider o customizzati vengono letti dal pc e vengono scritti sulle variabili di interesse dell'FPGA. Di seguito viene riportata un'immagine con dettaglio. Nella fig. 3.31 possiamo osservare come questo loop sia caratterizzato da un ciclo while esterno, una struttura sequenziale che serve a stabilire un ordine di esecuzione del ciclo, e diverse case structures, le quali vengono usate per gestire i consensi dei segnali booleani, delle variabili condivise e dei property nodes. In particolare, è presente inizialmente un property node che legge il valore dato dal bottone di controllo di PC connesso (quindi se il pc non è connesso, il ciclo non genera nessun segnale di set), poi viene letto il valore booleano di una variabile condivisa, la quale diventa vera quando la generazione è terminata.

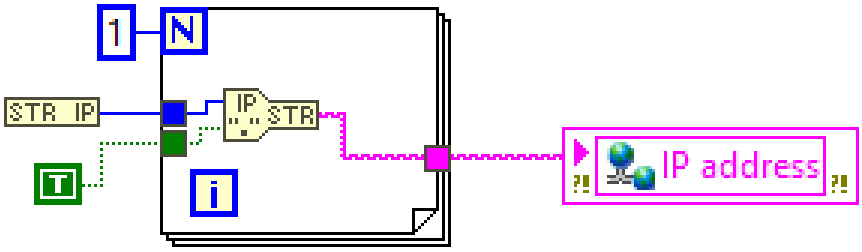


Figura 3.30: SubVI per identificare l'indirizzo IP.

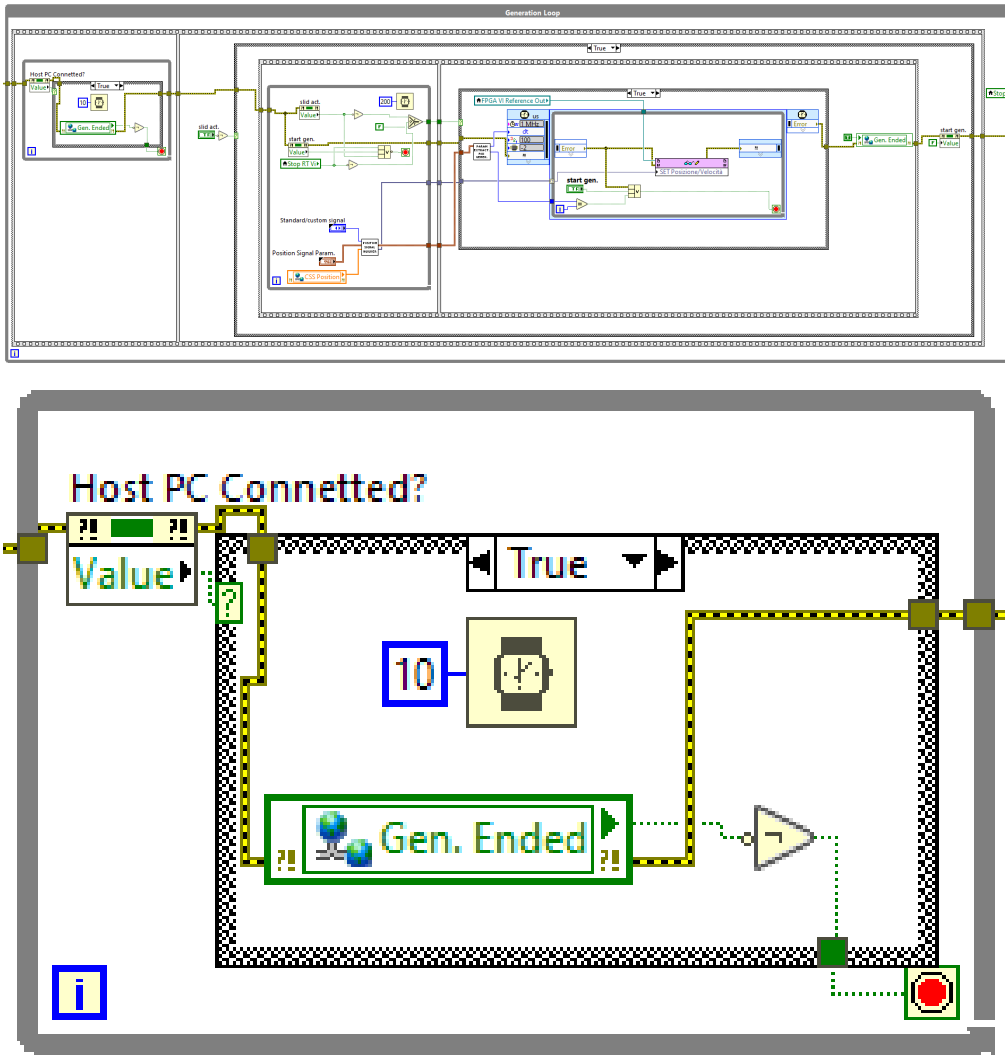


Figura 3.31: Generation Loop, caso segnale custom.

Se tale valore è falso, il ciclo si ferma e prosegue verso gli step successivi. Poi è presente un controllo booleano che permette all'utente di scegliere quale tipo di SET inviare, se tramite slider di posizione o tramite segnale customizzato. In base alla scelta dell'utente, il ciclo diventa quello di figura 3.31 o quello di fig.3.32. In fig.

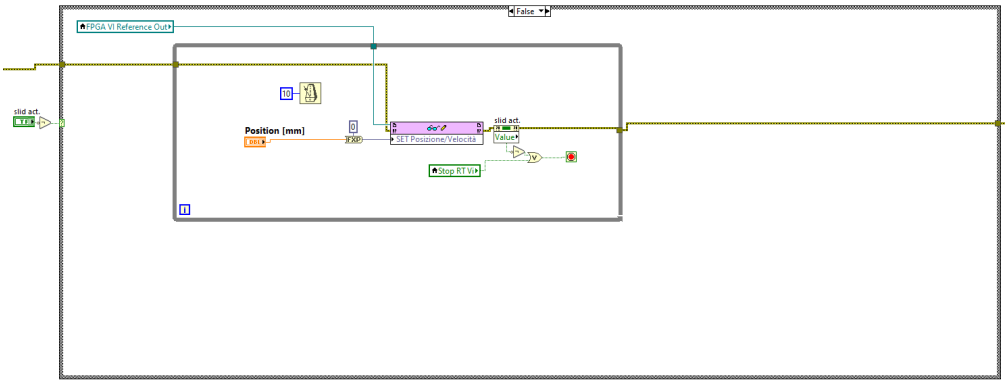


Figura 3.32: Generation Loop, caso SET tramite slider.

3.31 vediamo come vengano letti i dati provenienti dalla variabile condivisa dello slider di posizione e come si estraggano i parametri del segnale per utilizzarli nel timed loop di scrittura nel modulo FPGA. Le due subVI sono mostrate in fig.3.33 e 3.34 e sono rispettivamente delle VI di costruzione del segnale e di estrazione dei parametri per la generazione. Nella prima vediamo che è possibile scegliere due tipi di costruzione del segnale: standard o customizzata. In quella standard si prendono direttamente i parametri scelti quando si è costruito il segnale customizzato e si usano per stabilire le ampiezze, la frequenza di campionamento, la durata, la fase, il tipo di segnale, ecc., mentre in quello custom si inseriscono dei dati direttamente dal pannello frontale, ma è un tipo di generazione meno personalizzabile di quello standard. In fig. 62 si determinano i valori di dt e del numero di campioni in

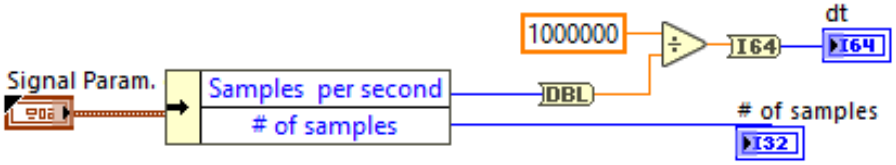


Figura 3.33: Estrazione dei parametri per la scrittura dei segnali.

base a ciò che è stato selezionato nei parametri scelti dall'utente. Il dt si ottiene

dividendo la frequenza (1 MHz) con cui il ciclo gira per il numero di campioni al secondo. Il metodo di generazione nel caso dello slider è più semplice, e consiste

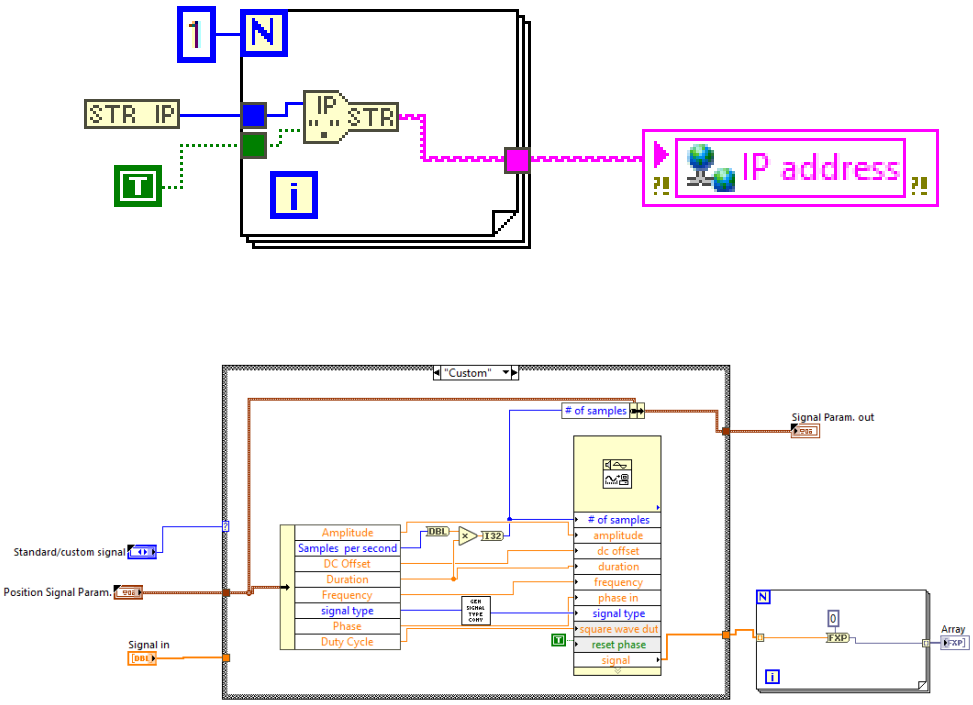


Figura 3.34: SubVI di costruzione del segnale di SET.

nel leggere il valore della variabile condivisa (che è stata scritta nel PC) e scriverlo tramite un'apposita funzione nella variabile "SET posizione/velocità". In ultimo, la VI principale del modulo RT viene chiusa andando a fermare il modulo FPGA e a unire tutti i segnali di errore in un Error handler comune. In fig. 3.36 è raffigurato il front panel del modulo RT. Viene utilizzato come supporto del PC, e in caso di necessità è possibile comandare in parte anche da qui la funzionalità base del modulo FPGA, ma si preferisce usare il PC.

3.3 Codice del PC

In quest'ultima sezione del codice Labview, si visualizzano i dati provenienti dal RT, si generano segnali di SET tramite slider o tramite una apposita interfaccia di personalizzazione del segnale. Inoltre, è possibile salvare i file acquisiti tramite un file .txt, che poi verrà postprocessato in matlab. La prima parte è quella

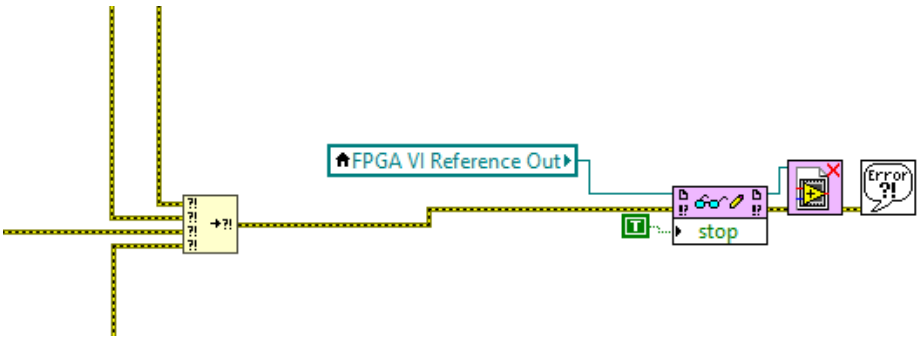


Figura 3.35: Chiusura della VI "Main RT".

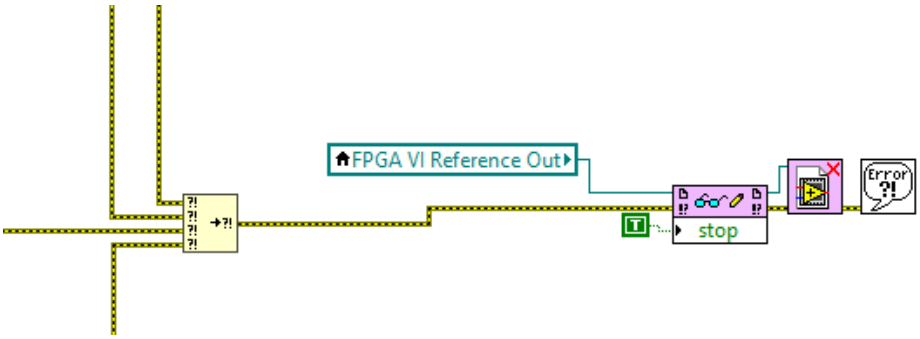


Figura 3.36: Pannello frontale del modulo RT.

di reinizializzazione delle variabili condivise, simile a quella presente nel modulo Real Time. La parte di creazione dell'endpoint di lettura per il network stream e necessaria per ricevere i segnali, ed è mostrata in dettaglio in fig.3.38.

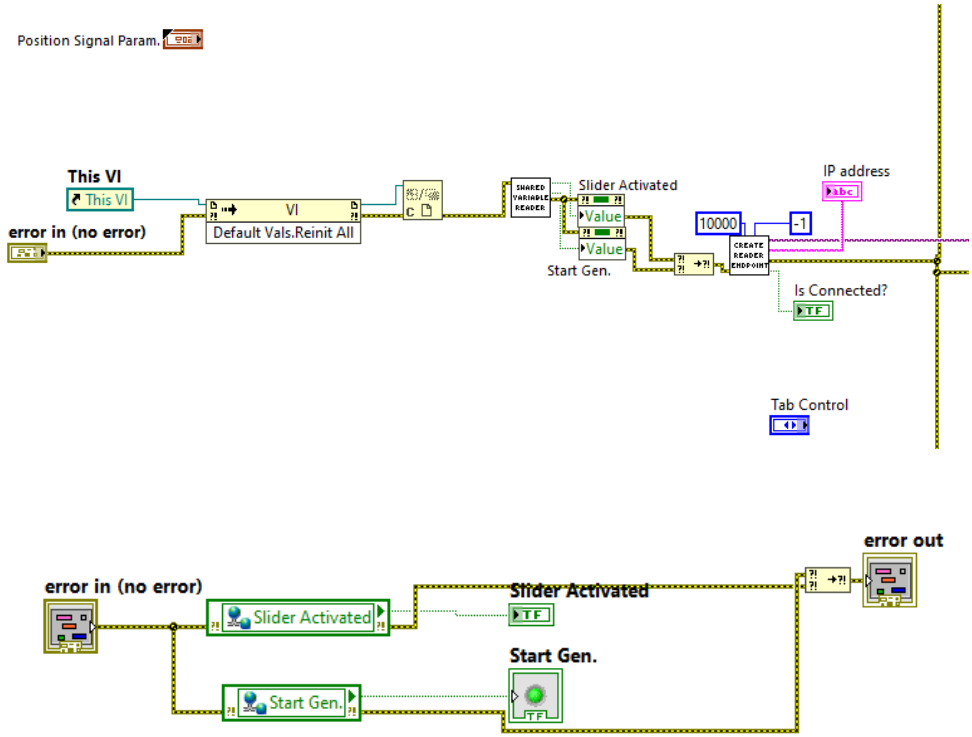


Figura 3.37: Parte iniziale del codice PC.

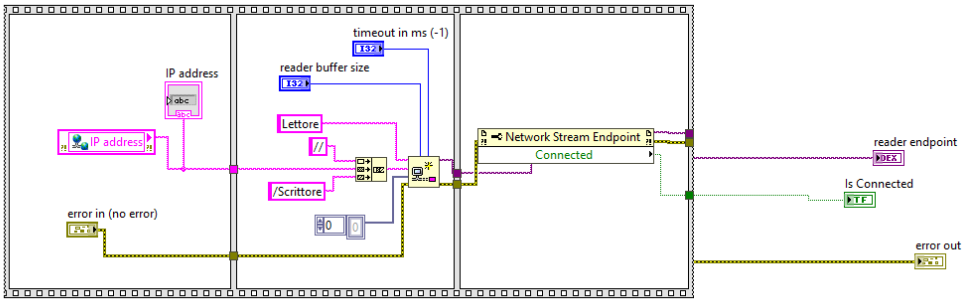


Figura 3.38: SubVI di identificazione dell'URL.

Passiamo ora a una delle parti più importanti del codice PC, ovvero quella dove

vengono letti i dati dal network stream e tramite delle code vengono spostati in un ciclo while che li converte in unnità di misura adatte e li raffigura tramite delle waveform charts.

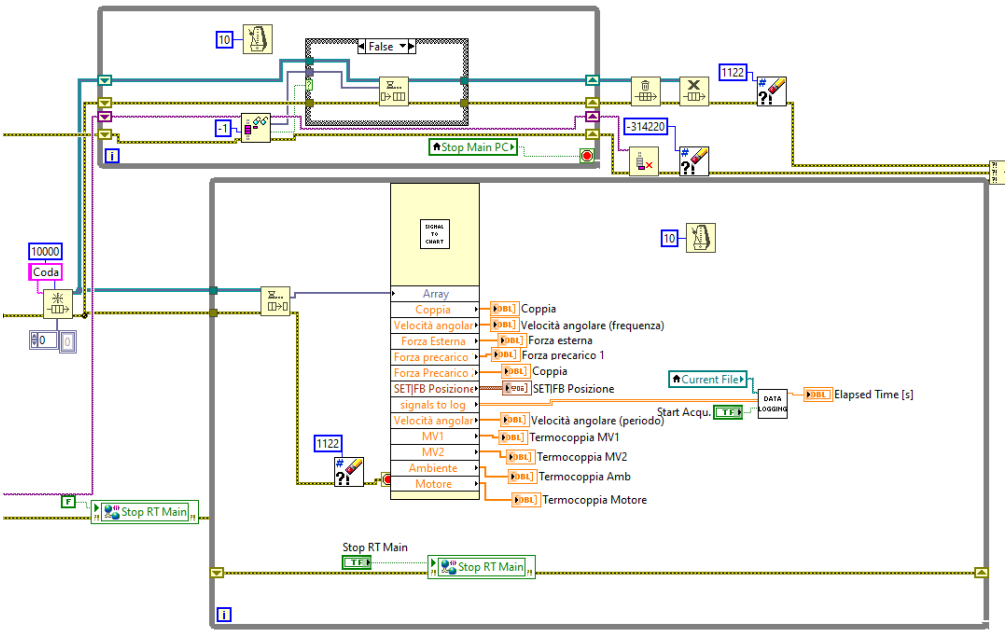


Figura 3.39: Lettura dei dati streammati e loro rappresentazione.

In fig. 3.39 possiamo osservare come il segnale proveniente dal modulo Real Time venga letto tramite una funzione apposita del gruppo dei network stream, per poi essere immesso in una coda e poi letto nel ciclo while sottostante. La subVI da cui fuoriescono tutti gli indicatori per i grafici è quella che ha il compito di convertire le unità di misura, dove necessario, e nel caso delle termocoppie è stato utilizzato un tipo di calibrazione già presente nell'archivio delle VI di Labview. Di seguito è mostrata la subVI appena descritta. I vettori contenenti i segnali acquisiti vengono prima convertiti tramite degli appositi guadagni contenuti nelle rispettive subVI e poi riuniti in un array 2D per poi essere salvati in una VI apposita di "Data Logging". Di seguito è mostrata nel dettaglio la subVI per la calibrazione delle termocoppie. Come si vede, tramite il modulo NI9211 noi acquisiamo sei segnali, di cui due servono in questa fase di calibrazione e sono interni al modulo stesso, e sono quelli relativi alla Cold Junction Conversion e al canale Autozero. Gli altri quattro sono quelli relativi alle quattro termocoppie installate nel banco prova. L'immagine 3.42 raffigura la subVI che imposta la creazione del file di salvataggio e "logga" i segnali preparati precedentemente in tale file. Inoltre, ci dà in output il tempo trascorso dall'inizio dell'acquisizione, così da avere un'idea di quanti dati stiamo acquisendo. Nella fig.3.43 c'è la parte in cui le variabili condivise vengono scritte,

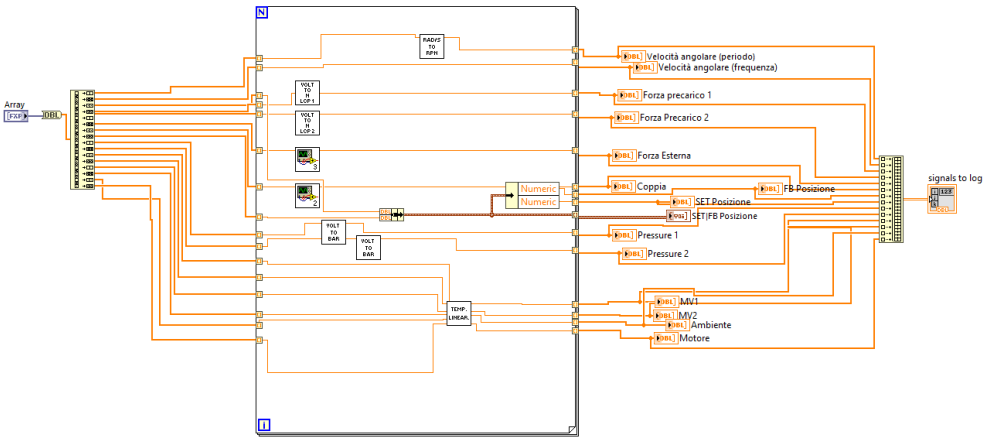


Figura 3.40: SubVI di conversione unità di misura e calibrazione.

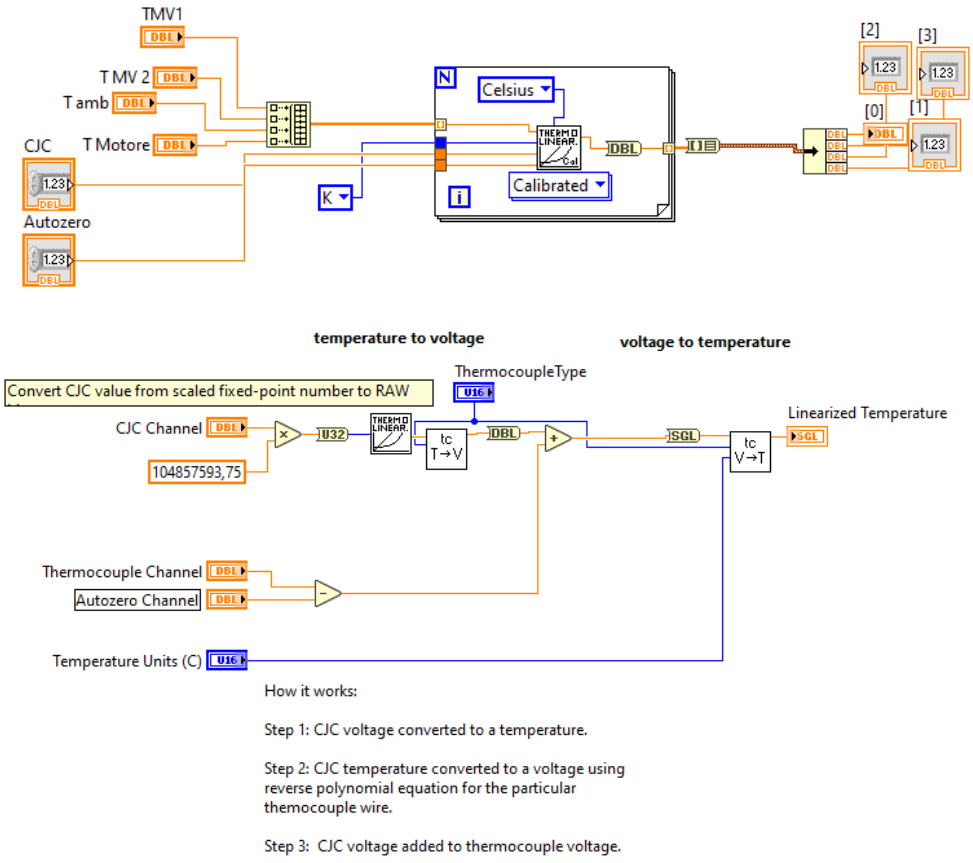


Figura 3.41: SubVI per la calibrazione delle termocoppie.

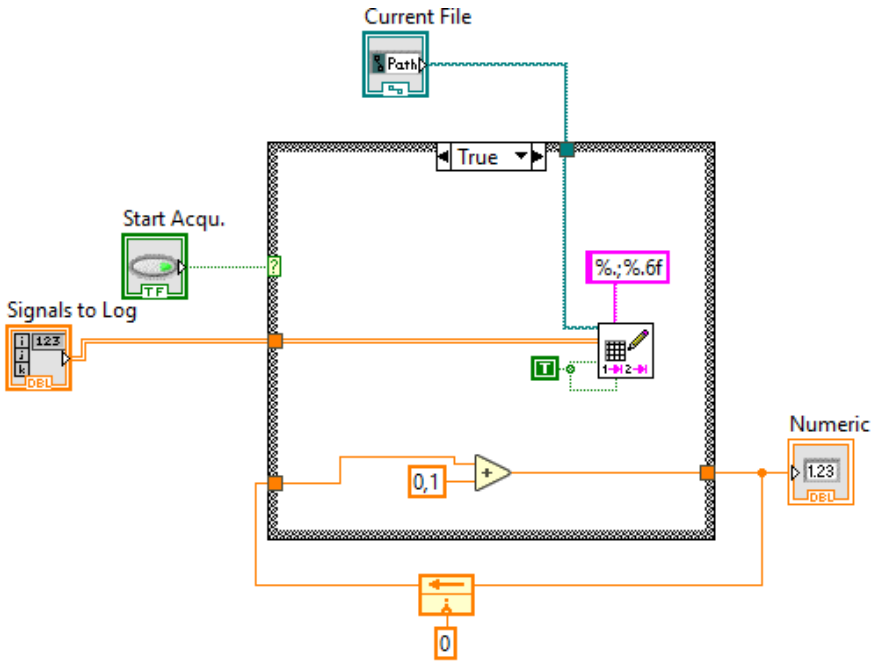


Figura 3.42: SubVI per salvare i dati acquisiti.

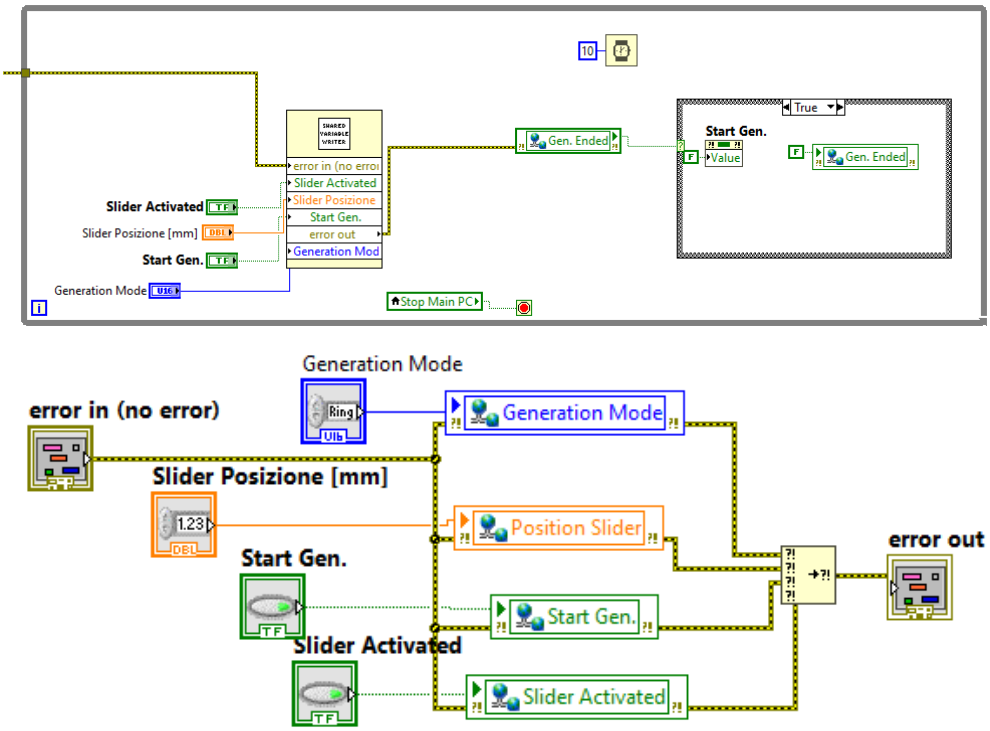


Figura 3.43: Scrittura delle variabili condivise.

in modo tale che queste informazioni possano essere lette dal target Real Time. Un'altra parte importante è quella della state machine, ovvero una sorta di case structure che a seguito dell'intervento dell'utente, è in grado di aprire determinate VI, le quali svolgono funzioni di supporto che andremo a spiegare. Come è possibile osservare dalla fig.3.44 , l'utente può eseguire varie azioni, come creare un file di salvataggio, aprire la VI di creazione del segnale di SET personalizzato e caricare il segnale di SET creato in precedenza. Tutto ciò viene gestito da questa parte di codice, all'interno della quale sono presenti delle subVI che ora vengono illustrate. Come si vede dalla fig. 3.45, l'interfaccia utente di questa subVI permette di scegliere il tipo di segnale, e di aggiungere o rimuovere più segnali di tipo diverso e con caratteristiche diverse. Di seguito viene riportata una figura che mostra il diagramma a blocchi di questa subVI, insieme a vari dettagli. In ultimo, viene riportata la schermata del pannello frontale del "PC MAIN", che l'utente utilizza per visualizzare i dati acquisiti e per gestire le varie funzioni del codice. Come si può osservare dalla figura 3.48, l'utente ha degli indicatori che segnalano lo stato di connessione tra PC e Real Time target, inoltre può accedere alla VI di creazione del segnale personalizzato, può salvarlo o caricarne uno già salvato, può scegliere il tipo di generazione e far partire la generazione del segnale appena creato. In alternativa, si può comandare il banco tramite uno slider, che in fase di avviamento dell'fpga viene scelto se eseguirà un SET di posizione o di velocità. Infine, è possibile salvare i dati che stiamo acquisendo tramite il box di controlli in basso.

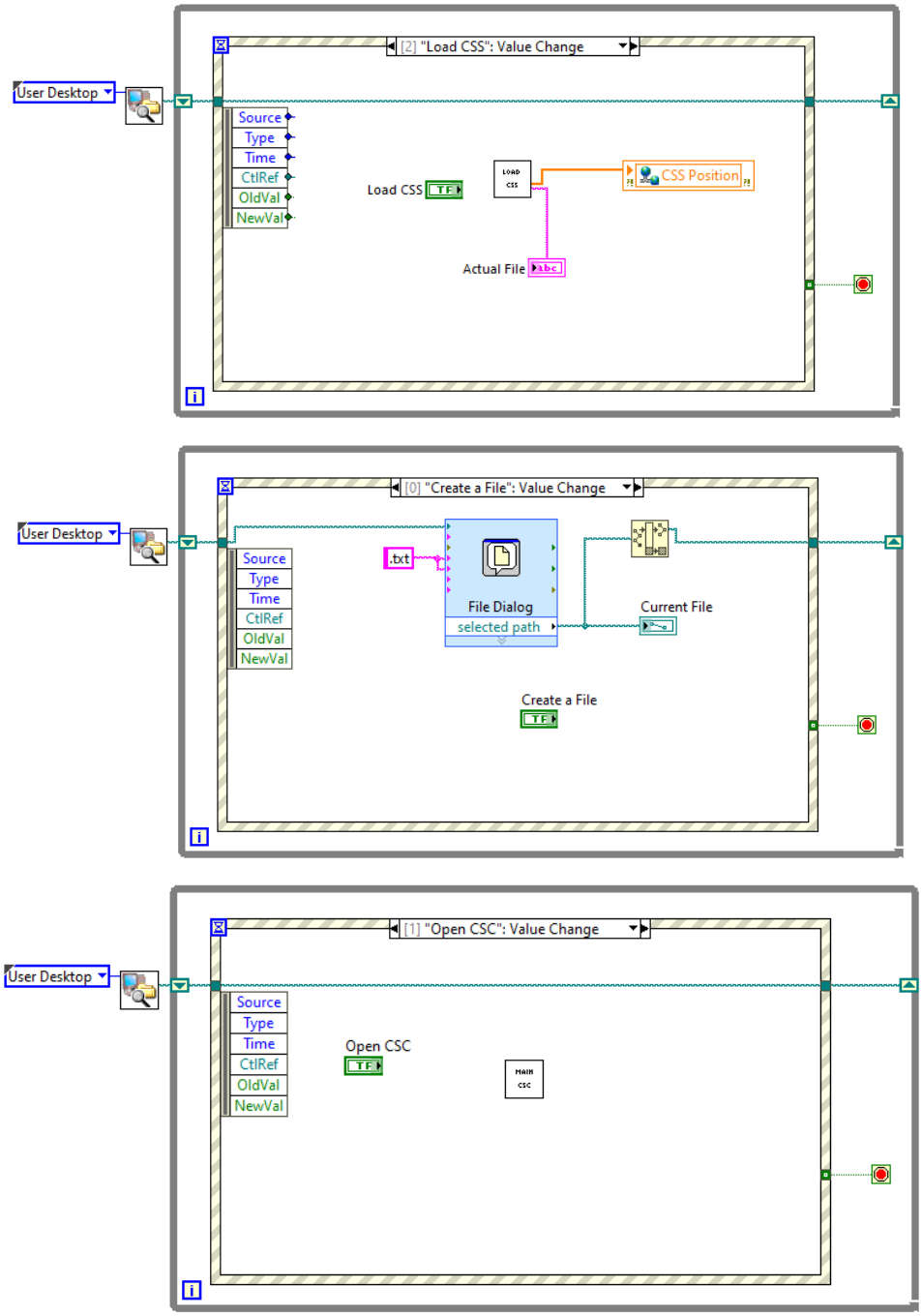


Figura 3.44: Struttura ad Eventi.

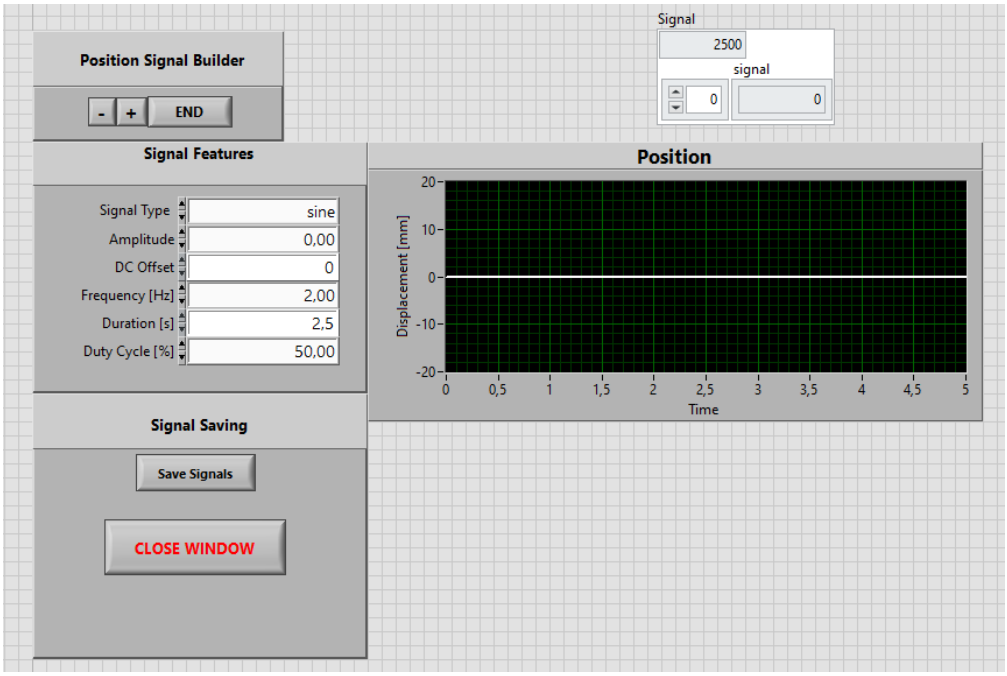


Figura 3.45: Pannello frontale della subVI di creazione del SET custom.

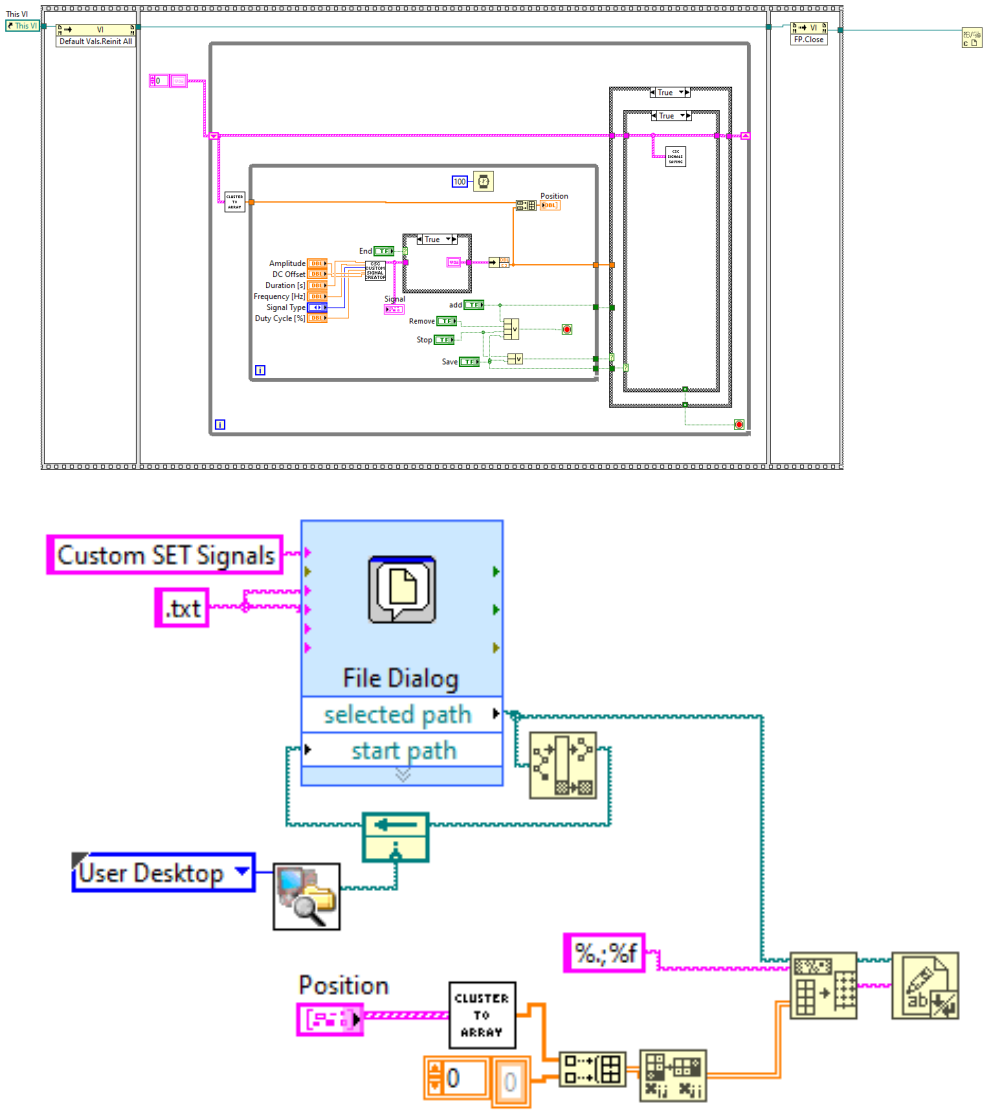


Figura 3.46: SubVI di creazione custom SET signal e subVI di salvataggio del segnale.

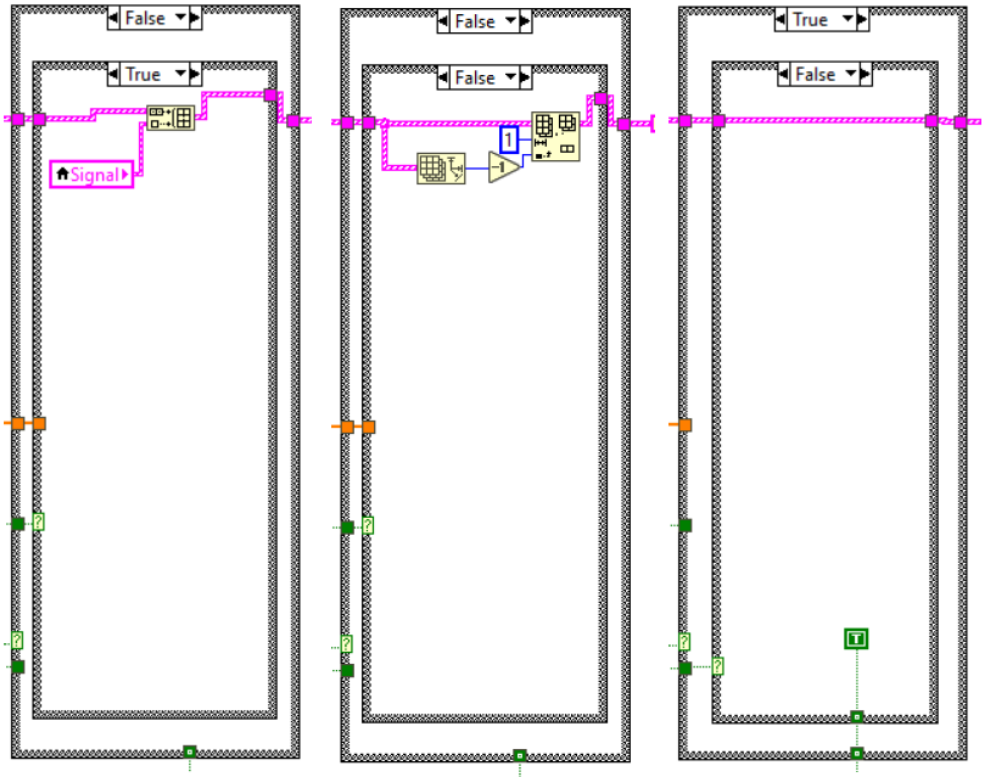


Figura 3.47: Case structure che gestisce i comandi dell'interfaccia utente.

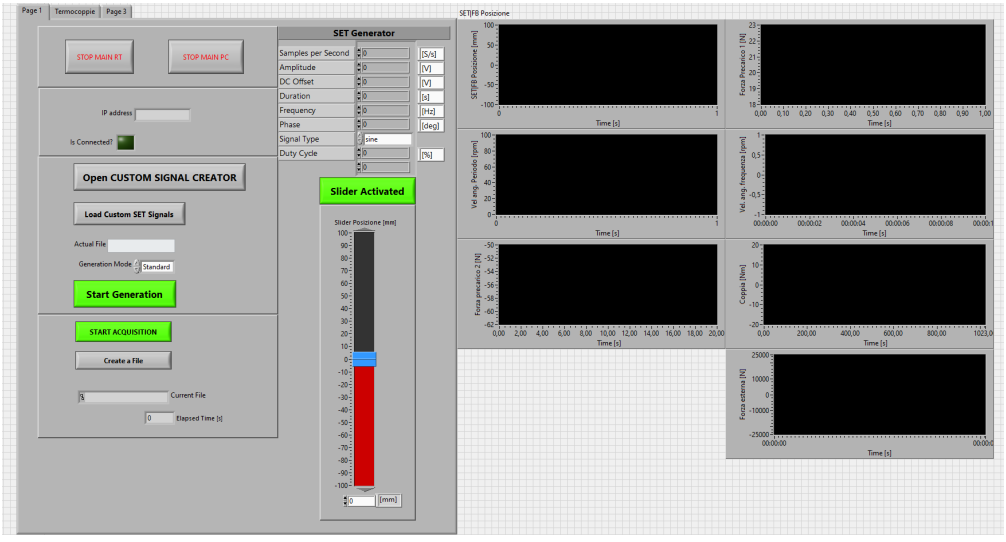


Figura 3.48: Schermata principale del pannello frontale.

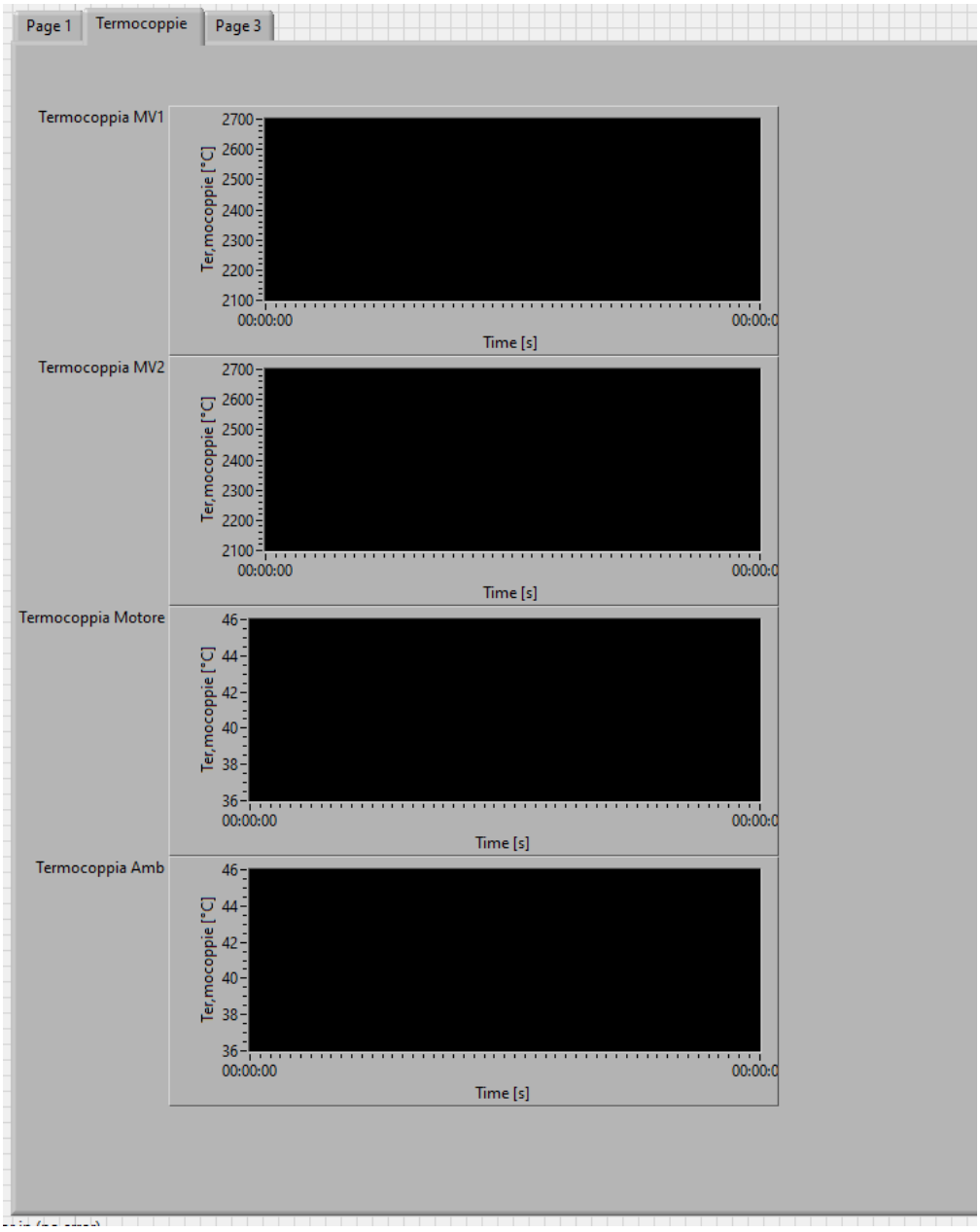


Figura 3.49: Visualizzazione delle termocoppie.

Capitolo 4

Conclusione

Il lavoro di tesi sin qui mostrato si è basato sullo studio del banco prova trasmissioni vite-madrevite e in particolare sulla scrittura del codice Labview che gestisce l'acquisizione dati e il controllo in posizione. Nel periodo dedicato al lavoro di tesi, è stato richiesto un notevole impegno dal punto di vista della multidisciplinarietà dei campi trattati e l'approfondimento di nuovi temi non trattati nel percorso di studi, come la progettazione di applicazioni di comando e di controllo e il funzionamento di hardware di acquisizione dati quali il cRIO-9047. Inoltre, sono stati frequentati vari corsi di formazione mediante il portale della didattica della National Instruments, tra cui quelli più importanti sono:

- Labview Core 1;
- Labview Core 2;
- Labview Core 3;
- Labview Real Time 1;
- Labview Real Time 2;
- Labview FPGA;
- Embedded Control and Monitoring Using LabVIEW.

La progettazione dei software ha richiesto un arduo studio preliminare della logica di programmazione e di implementazione degli algoritmi di trasmissione dati messi a disposizione dal software utilizzato. Gli obiettivi inizialmente prefissati per il progetto sono stati portati a termine: il codice di acquisizione dati è funzionante e le funzionalità necessarie per ora sono state raggiunte.

4.1 Sviluppi Futuri

In seguito allo studio svolto in questa tesi, si effettuerà il collegamento tra la vite e lo stelo del cilindro pneumatico, il quale potrà essere comandato tramite un segnale proveniente dal cRIO e diretto alla valvola proporzionale che lo controlla. Ciò richiederà un ampliamento del codice Labview, poiché andranno implementate tutte le fasi di generazione del segnale, così come è stato fatto per il comando del motore Lenze. A questo punto, si disporrà di un attuatore idraulico che, secondo delle leggi di forza in accordo ad un comando di Set, simula le forze aerodinamiche e che agisce come disturbo di forza sul servocomando di volo da testare, il quale verrà controllato in posizione durante una determinata condizione di volo.

Capitolo 5

Bibliografia

- National Instruments, 2014. LabVIEW Core 1 - Partecipant Guide. s.l.:National Instruments.

- National Instruments, 2014. LabVIEW Core 2 - Partecipant Guide. s.l.:National Instruments.

- National Instruments, 2014. Embedded Control and Monitoring Using LabVIEW - Partecipant Guide. s.l.:National Instruments.

- Sorli, M., 2010. Dispense del corso di meccatronica. Torino: Dipartimento di Ingegneria Meccanica e Aerospaziale.

- "Robust Design Of A Test Bench For Phm Study Of Ball Screw Drives", A.C. Bertolino, A. De Martin, G. Jacazio, S. Mauro, M. Sorli.

- "Speed Measurement Algorithms for Low-Resolution Incremental Encoder Equipped Drives: a Comparative Analysis", R. Petrella, M. Tursini, L. Petretti, M. Zigliotto.

- "Incremental Encoder Based Position and Speed Identification: Modeling and Simulation", I.I. Incze, A. Negrea, M. Imecs, C. Szabò.