ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
Sede di Forlì

Corso di Laurea in
INGEGNERIA MECCANICA
Classe LM-33

TESI DI LAUREA

in Controllo dei Motori a Combustione Interna LM

Sviluppo di un Sistema per l'effettuazione di Cicli di Guida per Motoveicoli sul Banco a Rulli

CANDIDATO
Fabio Ferrini

RELATORE
Prof. Ing. Enrico Corti

Anno Accademico 2014/2015
Sessione II

Dedicata ai miei genitori, a mio fratello, ai miei zii e a mia nonna

Ringraziamenti

Forlì, 30 Novembre 2015

Innanzi tutto vorrei ringraziare i miei genitori per avermi sempre dato la possibilità di scegliere cosa desideravo, permettendomi di inseguire le mie passioni.
Vorrei ringraziare il Prof. Ing. Enrico Corti per avermi dato la possibilità di svolgere la tesi in ambito automotive, per il quale nutro da sempre una grande passione. Lo ringrazio anche per tutti i suoi insegnamenti e soprattutto per la fiducia concessami.
Un doveroso ringraziamento all'Ing. Michele Taccioli per la collaborazione e il grande aiuto durante il lavoro di tesi.
Dopodiché vorrei ringraziare l'Ing. Michele Soli e l'Ing. Davide Galli di Alma Automotive per la loro disponibilità e la pazienza che hanno avuto all'inizio nel seguirmi, dandomi utili consigli e supporto tecnico ai fini della buona riuscita del lavoro.
Grazie anche all'Ing. Roberto Bertacin che è sempre stato molto disponibile a rispondere alle mie frequentissime domande.
Un grazie particolare anche a tutti i ragazzi dell'Hangar per il magnifico periodo trascorso insieme.

Grazie a tutti
Fabio

Indice

Introduzione

Il lavoro svolto in questa tesi fa parte di un progetto volto alla realizzazione di un sistema per svolgere test di guida di motoveicoli al banco a rulli, gestendo l'intera prova in remoto, ossia dall'esterno della cella.
Tale sistema, denominato RideIT, è stato progettato da Alma Automotive per conto della nota casa motociclistica MV Augusta.
Tramite il sistema RideIT, è possibile automatizzare la guida di motoveicoli sul banco a rulli, riproducendo via software, tramite un modello che governa degli attuatori, il comportamento del pilota per quel che riguarda gli azionamenti di acceleratore, frizione, cambio e freno durante un test.
I vantaggi di questa soluzione riguardano soprattutto la sicurezza degli addetti, che possono effettuare tutte le prove del caso rimanendo al di fuori della sala prove, oltre alla possibilità di incrementare la ripetibilità e la riproducibilità dei risultati dei test, grazie all'eliminazione del fattore di soggettività legato al guidatore.
Il sistema nasce con la specifica di poter eseguire test di omologazione al banco a rulli anche se, grazie alla flessibilità che è in grado di garantire, può essere perfettamente riconfigurato, anche in vista di futuri sviluppi del progetto.
L'architettura hardware del sistema RideIT è visibile nella seguente figura:

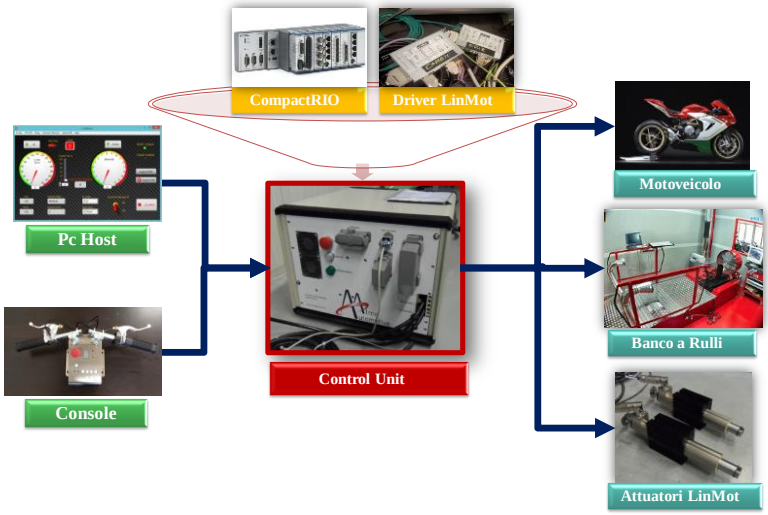


Figura .1: Sistema RideIT

Il banco a rulli e il motoveicolo sono le componenti con cui il sistema RideiIT deve interfacciarsi comunicando con:

- i sensori montati sulla moto e al banco
- la centralina del motore (ECU)
- (opzionalmente) il sistema di controllo sala.

Il controllo del sistema è realizzato su un hardware National Instruments CompactRIO con a bordo SO real-time, unità FPGA e schede per l'acquisizione e l'elaborazione dei segnali AI, AO, DIO, oltre ad avere interfacce CAN e CANopen rispettivamente per la comunicazione con la ECU e gli attuatori.
Per quanto riguarda le attuazioni si sono utilizzati dei motori lineari prodotti dalla dalla LinMot. Questi rappresentano una soluzione molto interessante in questa applicazione dal punto di vista del controllo e delle le loro caratteristiche fisiche e dinamiche.
Si ha un pc Host che fornisce l'interfaccia utente e offre la possibilità di modificare i vari parametri necessari a configurare correttamente il sistema per la moto che si vuole testare. In aggiunta, si ha un'interfaccia utente opzionale con cui è possibile controllare il sistema. Tale dispositivo è la Consolle. Essa emula fisicamente un manubrio motociclistico e include nella sua struttura tutti quei comandi utili per poter gestire la guida manuale in remoto, comprese le cambiate. Questa caratteristica la rende più comoda e agevole da utilizzare laddove sia necessario effettuare manovre di messa a punto, simili a quelle che effettuerebbe un pilota stando in sella alla moto.
Il lavoro svolto durante il periodo di tesi, ha riguardato una particolarizzazione del progetto RideIT, relativa appunto alla Consolle. Per essa è stata affrontato un lavoro di progettazione e realizzazione sia dal punto di vista hardware, che di quello software, rispettando le specifiche assegnate.
Dal punto di vista software, la programmazione della Consolle è stata realizzata in Simulink, ambiente grafico di sviluppo contenuto in Matlab®, mentre per l'implementazione del software stesso è stata usata la scheda di controllo STm32F4-Discovery prodotta dalla *STmicroelettronics* che sfrutta le potenzialità di un microcontrollore.

Per quanto riguarda altri lavori, inerenti le componenti HW/SW del sistema RideIT, sono stati portati avanti parallelamente presso Alma Automotive e presso il laboratorio dell'Università (Hangar).

Con riferimento alla struttura della tesi, abbiamo:

- Nel **Capitolo 1** la descrizione del sistema RideIT nel suo complesso con approfondimento sui segnali e le parti collegabili al sistema stesso. Viene fornita inoltre, una descrizione dettagliata di quelle che sono le componenti hardware utilizzate per realizzare la parte di controllo e azionamento.
- Nel **Capitolo 2** la presentazione delle interfacce utente (Host pc e Consolle) e di come queste possono essere utilizzate ai fini della controllabilità e configurabilità del sistema. Per quanto riguarda la Consolle vengono presentati alcuni aspetti tecnici e le scelte progettuali che sono state fatte per arrivare alla realizzazione del dispositivo.
- Nel **Capitolo 3** la descrizione passo a passo dei punti salienti della programmazione realizzata per la Consolle in ambiente Simulink.
- Nel **Capitolo 4** la descrizione del programma eseguito in real-time. In particolare si descrivono in modo approfondito gli schemi di controllo del modello Simulink contenenti la parte di calcolo che definisce come devono essere gestite le attuazioni sul motoveicolo al banco a rulli nei diversi stati di funzionamento possibili del sistema. Viene inoltre esposto come il modello si integra nell'ambiente di LabView quando viene eseguito sulla piattaforma NI CompactRIO.
- Nel **Capitolo 5** la descrizione delle prove sperimentali effettuate per valutare il corretto funzionamento degli strumenti hardware e software creati del sistema RideIT, cercando anche di porre le basi, in vista di uno sviluppo futuro, di una metodologia di test finalizzata a valutare in modo completo le prestazioni del sistema.

In **Appendice A** è riportata la tabella con le configurazioni dei collegamenti elettrici (*pinout*) sulla scheda di controllo STm32 di cui è equipaggiata la Consolle, mentre in **Appendice B** viene riportato il file di inizializzazione in cui si definiscono i parametri utilizzati nel modello Simulink eseguito sulla macchina

real-time, ed infine in **Appendice C** viene riportato lo schema elettrico seguito per la realizzazione dei cablaggi della Consolle.

Capitolo 1

Il Progetto RideIT

1.1 Obiettivi del progetto RideIT

Il progetto RideIT è nato con l'intento di creare un sistema, il cui scopo è consentire ad un operatore di svolgere test di ogni genere su motoveicoli al banco a rulli, gestendo l'intera prova in remoto, dall'esterno della cella.
Per fare ciò occorre sostituire il guidatore con un sistema automatico provvisto di attuatori che ne replichi fedelmente i movimenti sul motoveicolo e trasferire i comandi per la gestione della guida all'esterno della cella del banco a rulli. Da questa postazione l'utente è in grado di controllare il motoveicolo e il banco a distanza e in completa sicurezza.
Tale sistema deve essere inoltre completamente flessibile e facilmente personalizzabile in base alle possibili esigenze.
Come è stato citato nell'introduzione e mostrato in figura .1 si prevede l'utilizzo di hardware su cui implementare via software tutti gli algoritmi di calcolo e controllo del sistema. L'avere a disposizione hardware e software che comunicano con la centralina del motoveicolo, permette la gestione in modo flessibile e accurata del comando dell'acceleratore oltre a fare l'acquisizione di grandezze di interesse del motore con cui mettere in atto strategie di controllo che permettano di eseguire delle partenze e cicli di guida con cambio manuale o automatico sulla moto.
Si prevede anche di intervenire sui malfunzionamenti o altre condizioni di funzionamento anomale che possono manifestarsi durante il normale funzionamento in modo da permettere la messa in sicurezza del motoveicolo e dei vari componenti dell'impianto.
In aggiunta, si prevede di effettuare la diagnosi in tempo reale sugli attuatori, e di agire sui parametri di funzionamento del sistema.

Vediamo ora una di fare una panoramica su quelli che sono i componenti hardware necessari per il funzionamento di tutto il sistema, prestando particolare attenzione a come questi sono interconnessi.

1.2 Descrizione del sistema

Nello schema a blocchi di figura 1.1, abbiamo una rappresentazione del sistema dal punto di vista dei segnali importanti che lo riguardano e che può governare, partendo dalla colonna degli input che può ricevere, attraversando la parte intermedia rappresentata dall'unità di controllo che è il cuore del sistema, fino a raggiungere l'ultima colonna, quella degli output.

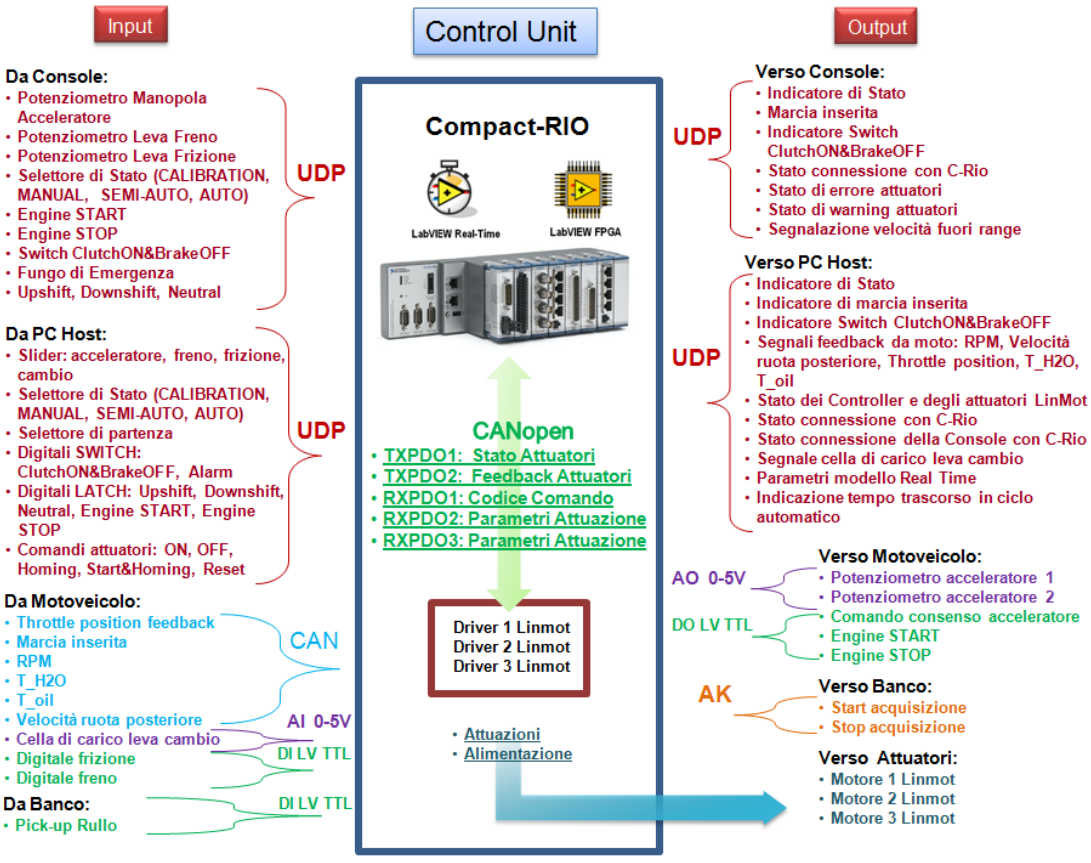


Figura 1.1: Schema generale comunicazioni

Come si può notare, alcune parti dello schema scambiano dati bidirezionalmente, sfruttando le capacità degli hardware di supportare questo tipo di comunicazione.

Dal punto di vista hardware, si ha l'unità di controllo, che ospita i servo controller degli attuatori e la piattaforma National Instrument CompactRIO compresa tutta la circuiteria di condizionamento e la strumentazione per il funzionamento in real-time del banco a rulli.
La parte Host e la Consolle forniscono l'interfaccia utente e danno la possibilità di modificare i vari parametri necessari per configurare correttamente il sistema per la moto che si vuole testare.
L'unità real-time si occupa della generazione e dell'acquisizione dei segnali analogici e digitali da e per il motoveicolo e il banco. Si occupa della comunicazione con la centralina di controllo motore del motoveicolo attraverso una rete seriale, utilizzando il protocollo CAN (Controller Area Network).
Per la comunicazione con i driver degli azionamenti si impiega il protocollo CANopen (secondo la modalità che verrà approfondita nel paragrafo 1.5), attraverso un'altra scheda CAN dedicata a tale funzione.
La comunicazione fra la parte real-time e i dispositivi di interfaccia (Host pc e Console), invece, avviene tramite un'interfaccia di rete standard a 100Mbps, utilizzando il protocollo UDP/IP (User Datagram Protocol / Internet Protocol). La scelta del protocollo UDP invece che del TCP (Transmission Control Protocol) è dovuta al fatto che questo protocollo è più rapido ed efficiente, quando lo si usa per applicazioni real-time che richiedono una frequenza minima di spedizione delle informazioni e non possono ritardare eccessivamente la trasmissione dei pacchetti, anche se possono tollerare qualche perdita di dati. Per contro però questo protocollo non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi.
Dallo schema di figura 1.1, si osserva inoltre che il sistema può essere predisposto anche per la comunicazione con banchi prova che sfruttano il protocollo AK.

1.3 Il CompactRIO

Il CompactRIO, riportato in figura 1.2 è lo strumento hardware dove si va a creare il programma che permette di comunicare con gli azionamenti e le interfacce utente.
Il CompactRIO è un avanzato sistema di controllo ed acquisizione potenziato dalla tecnologia di input e output riconfigurabili (RIO) sviluppata da National Instruments. Per le ridotte dimensioni (300 x 93 x 87 mm3), peso (1.164 kg) e bassi consumi (per

l'alimentazione è sufficiente una batteria da 9V) può essere utilizzato per eseguire svariate applicazioni di controllo e monitoraggio avanzate in ambito automotive, macchinari industriali, Rapid Control Prototyping ecc.

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

- Controller NI cRIO-9068;
- Moduli di I/O.

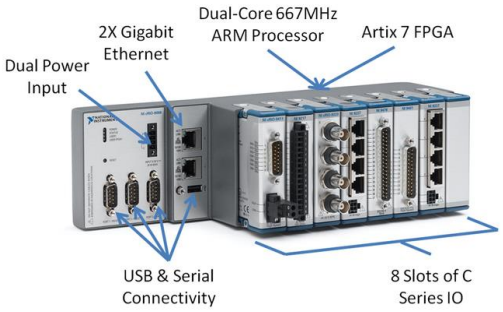


Figura 1.2: Piattaforma NI CompactRIO-9068

Il controller NI cRIO-9068 è il cuore del sistema in quanto integra un processore ARM Cortex-A9 dual-core da 667 MHz con NI Linux Real-Time OS, un chip Artix-7 FPGA e 8 slot per i moduli I/O. Il RIO FPGA presenta una connessione con ogni modulo I/O ed è programmato con funzioni elementari che permettono di scrivere o leggere informazioni sui canali provenienti da ognuno di essi.

Dal momento che non c'è un bus di comunicazione condiviso tra l'FPGA ed i moduli I/O, le operazioni di input e output in ogni modulo devono essere sincronizzate con precisione. A fronte di ciò, il dispositivo FPGA può elaborare i dati, prendere decisioni e trasferire direttamente un segnale da un modulo ad un altro e comunicare con il controller real-time tramite l'interfaccia DMA (Direct Memory Access), lavorando ad una frequenza di clock di 40MHz.

Il controller NI cRIO-9068 dispone di 512 MB di memoria RAM per operatività embedded, 1 GB di memoria non volatile per il data logging e una vasta gamma di opzioni di connettività come due porte Gigabit Ethernet, una USB Hi-Speed e tre porte seriali; inoltre è realizzato per lavorare mantenendo la propria affidabilità anche in ambienti ostili con temperature comprese tra i -40 ed i 70°C. Il processore ARM Cortex-A9 dual-core da 667 MHz unisce ai bassi consumi la possibilità di analizzare ed elaborare funzioni reali eseguendo loop di controllo ad una frequenza anche superiore ad 1kHz. L'esecuzione di codice può essere sincronizzata sia attraverso l'invio

di IRQ dall' FPGA sia con un clock real-time preciso al ms. In aggiunta alla comunicazione tramite i protocolli TCP/IP, UDP, Modbus/TCP, IrDA e seriali il controller include server per gli standard VISA, HTTP e FTP. I moduli I/O si collegano direttamente al RIO FPGA e costituiscono un sistema ad alte prestazioni che conferisce alle applicazioni di input/output realizzate tramite software le qualità e la flessibilità di un tradizionale circuito elettrico completamente dedicato a tali funzioni. Il CompactRIO fornisce direttamente l'accesso hardware ai circuiti di input/output di ognuno dei moduli utilizzando le funzioni I/O elementari di LabView FPGA. Ogni modulo, inoltre, contiene internamente applicazioni di condizionamento del segnale che consentono, in molti casi, il collegamento diretto a sensori e attuatori facilitato dalla presenza di terminali a vite, BNC o connettori D-Sub. I moduli I/O utilizzati nella nostra configurazione hardware sono i seguenti:

- Scheda NI 9862 High-Speed/FD CAN 1-port
- Scheda NI 9881 CANopen 1-port
- Scheda NI 9381 per I/O multifunzione da 0V a 5V
- Scheda 9375 di I/O digitali

La NI 9862 è la periferica utilizzata per la comunicazione CAN (Controller Area Network), attraverso cui si gestiscono i canali provenienti dal motoveicolo. Essa integra un controller in grado di manipolare centinaia di segnali e frame CAN ad alta velocità (fino a 8Mbit/s). La NI 9881 è un modulo CANopen ad 1 porta, utilizzato per trasmissione/ricezione di PDO (Process Data Objects) e SDO (Service Data Objects) ad alta velocità (fino a 1Mbit/s). Per usare le prime due schede, non è sufficiente andare a collegare un connettore CAN, ma è necessario portare l'alimentazione dall'esterno su entrambe le porte CAN in quanto, per motivi prestazionali, non è possibile portare l'alimentazione ad entrambe dallo chassis del CompactRIO.
La NI 9381 include 8 canali di uscita analogica (single-ended) a 12-bit da 0V a 5V e permette di raggiungere una velocità di campionamento di 20KS/s (in modalità multiplexata tra AI/AO). Dispone inoltre di 4 linee digitali con velocità fino a 1MHz e compatibili con la logica LVTTL. Infine la NI 9375 include 16 canali per ingressi digitali e 16 canali per uscite digitali, tutti compatibili con livelli logici 24V.
Lo sviluppo dell'applicazione per il CompactRIO prevede tre tappe:

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

Figura 1.3: Installazione del cRIO e dei moduli I/O

1.4 Gli azionamenti

All'interno di questa sezione viene fatta una panoramica su quelli che sono gli attuatori ed i sistemi per la gestione dei comandi sulla moto quali il cambio, il freno, la frizione e l'acceleratore.

1.4.1 Motori Lineari

Gli attuatori di cambio, frizione e freno sono dei motori elettrici lineari prodotti dall'azienda LinMot.

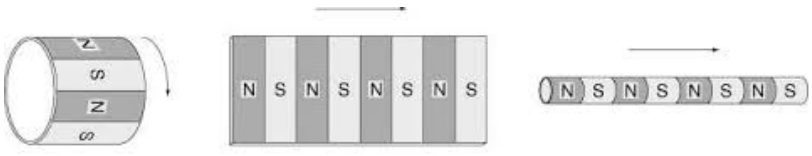


Figura 1.4: Campo magnetico nei motori rotativi (a), svolgimento planare del campo (b) e campo magnetico nei motori lineari (c).

In questi motori il movimento lineare è generato direttamente dal campo elettromagnetico senza l'intervento di alcuna parte meccanica. Come nei motori sincroni

rotativi, vi sono dei magneti permanenti nella parte mobile (slider) e degli avvolgimenti nella parte fissa (statore). In figura 1.4 viene evidenziata la disposizione del campo nei motori rotativi, il suo sviluppo planare e la successiva disposizione con direzione del campo parallela all'asse dello slider nei motori lineari.
Nei motori lineari l'avvolgimento statorico è a due fasi e non a tre come nei motori sincroni rotativi.
I modelli di motore lineari impiegati nel sistema RideIT sono del tipo PS01-48x240-c insieme ai servo-controller B1100-GP-XC, e alle power supply units S01-72/1000, tutti prodotti della LinMot. Integrati nel corpo dello statore vi sono gli avvolgimenti, il cuscinetto per il cursore, il sensore di posizione e un sensore di temperatura per il monitoraggio termico del motore. La stessa azienda propone anche diverse soluzioni per il raffreddamento degli attuatori. Nel nostro caso i tre motori sono stati dotati di flange di raffreddamento per cautelarsi da eventuali surriscaldamenti.

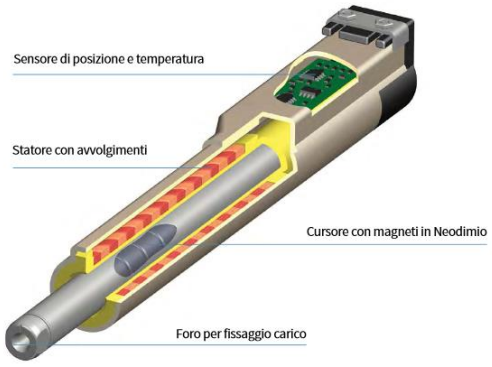


Figura 1.5: Schema del motore lineare Linmot

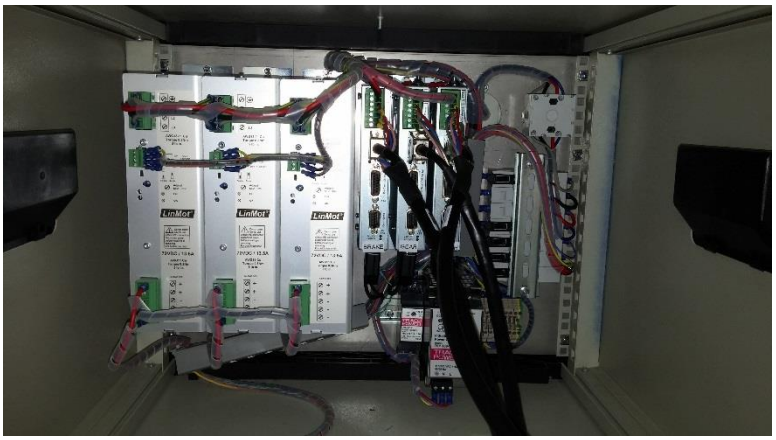


Figura 1.6: Fotografia servo-controller e power supply units Linmot

Il servo controller è un'unità digitale programmabile via porta seriale da PC tramite il programma LinMot-Talk. L'alimentatore di potenza S01-72/1000 è un alimentatore AC/DC che permette il passaggio dalla tensione 380 volt trifase ad una tensione DC di 72V per la generazione del segnale di potenza necessario alla movimentazione degli attuatori.
Le caratteristiche che hanno portato alla scelta dei motori elettrici lineari LinMot, riguardano soprattutto l'accuratezza di posizionamento (position repeatability) di 0,05mm, la flessibilità e l'estrema compattezza di tali sistemi rispetto ai motori elettrici rotativi o all'opzione degli attuatori pneumatici.
Dalle prove sperimentali effettuate su questi dispositivi e dai risultati di altre esperienze condotte in Alma Automotive si è notato come, entro i limiti delle applicazioni, la scelta di questi attuatori e dei loro servo controller sia più che accurata.
Alcuni dei dati caratteristici del motore lineare, ricavati dal *datasheet* del costruttore, sono riportati nella tabella 1.1.

Massima forza attuabile	$585N$
Massima forza attuabile continua	$145N$
Massima accelerazione	$400m/s2$
Massima velocità	$1.7m/s$
Massima corrente	15A

Tabella 1.1: Parametri caratteristici del motore lineare PS01-48x240-C con una tensione di alimentazione di 72V.

La massima forza di picco può essere attuata per pochi secondi dopo i quali il Servo controller toglie l'alimentazione ai motori per evitare danni da surriscaldamento.

1.4.2 Il montaggio degli attuatori

Per quanto riguarda l'installazione degli attuatori sul motoveicolo, è previsto che possa esser fatta senza ricorrere a grossi adattamenti meccanici alla moto. Questi attuatori, in quanto lineari, non sono adatti a sopportare dei carichi che non siano diretti secondo l'asse dello slider, perciò il montaggio di essi deve garantire che non si creino forze radiali durante le movimentazioni.
Per il montaggio dell'attuatore di frizione ad esempio è possibile adottare un supporto vincolato a terra, sul quale montare un semi manubrio con la leva frizione,

dove connettere lo slider del motore lineare (movimentazione in verticale). Un'altra possibilità consiste nell'azionare la leva frizione direttamente sulla moto, fissando l'attuatore ad un supporto solidale ad essa (movimentazione orizzontale).
Per quanto riguarda gli attuatori di freno e cambio, conviene vincolarli in maniera solidale alla moto, dal momento che questa, durante le prove al banco a rulli, anche se perfettamente vincolata, ha un certa escursione verticale dovuta all'azione del motore che comprime la molla della sospensione posteriore.
Per fare lavorare gli attuatori direttamente sui pedali di freno e frizione della moto, occorre realizzare degli accoppiamenti con un grado di libertà tale da garantire che non si creino forze radiali durante le movimentazioni per la cambiata e la frenata.

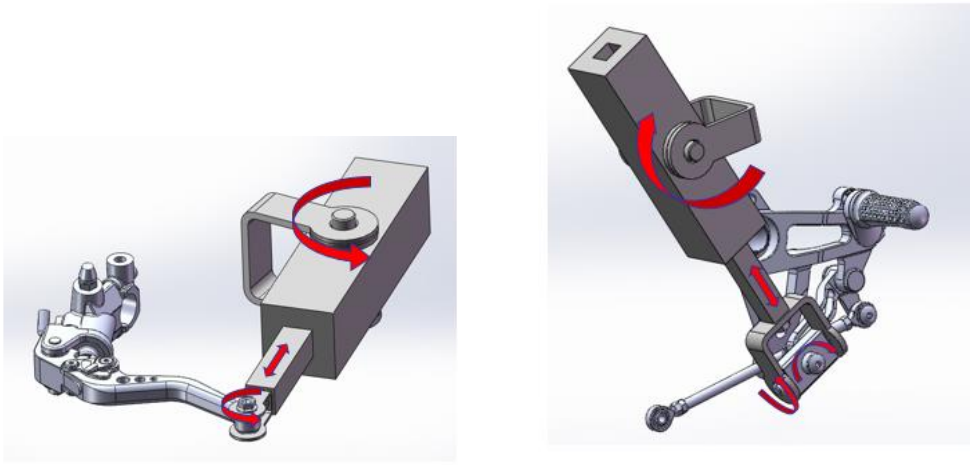


Figura 1.6: Esempi possibili di montaggio degli attuatori

1.4.3 Gestione dell'acceleratore

L'acceleratore nella moto è azionato dalla manopola posta sul semi manubrio destro. Questa manopola può in genere andare a tirare un cavo metallico che agisce su un potenziometro da cui viene generato il comando in tensione da inviare alla centralina del motore. Questo se il motoveicolo in questione è dotato del sistema *Ride by Wire*. La gestione del comando della farfalla infatti, nei moderni motoveicoli, è ormai completamente demandata alla centralina, rendendo così obsoleto il classico comando diretto con il cavo.

Come da specifica di MV Augusta il sistema deve consentire la possibilità di eseguire cicli di omologazione sul banco a rulli, per questo motivo è stato pensato principalmente per lavorare con motoveicoli di nuova generazione, dotati nella fattispecie di sistema ride by wire.
La gestione dell'acceleratore è effettuata perciò ricorrendo ad un segnale in tensione (0-5 volt) che va direttamente a sostituire il segnale proveniente dal reale potenziometro collegato al comando acceleratore del motoveicolo.
Tale soluzione ha il vantaggio di essere semplice da gestire, inoltre risulta poco ingombrante e permette di riprodurre con poca incertezza i dati inviati tramite il compactRIO. Nel sistema è consentito gestire manualmente tale segnale, grazie alle apposite interfacce messe a disposizione dell'utente. Si può infatti utilizzare il comando a manopola della Console, piuttosto che passare all'interfaccia Host in cui si impiega un apposito cursore grafico. Entrambi i dispositivi di interfaccia utente comunque, sono programmati per far sì che il segnale generato venga inviato al cRIO che lo elabora. Tramite la scheda NI 9381 che gestisce segnali analogici I/O (0, 5 volt), viene poi generato l'input analogico per la moto. Grazie all'interfaccia di guida del sistema, l'utente può effettuare la sua richiesta di coppia al motore, potendo anche visualizzare in Host il valore di apertura farfalla attuato. Il feedback di posizione farfalla viene anche utilizzato per eseguire procedure che prevedono controlli in retroazione quando si eseguono test in cui la gestione del motoveicolo è completamente automatica o semi automatica.

1.5 La Comunicazione CANopen degli attuatori

Gli attuatori della LinMot scelti per l'applicazione, si possono governare in vari modi. Si è scelto di comunicare con essi attraverso il protocollo CANopen, in quanto così è possibile accedere, gestire e modificare a piacere tutti i parametri di comando e configurazione del controllore.
Il CANopen è un protocollo di comunicazione standard non proprietario che sfrutta la rete CANbus. Dal punto di vista hardware, viene impiegato appositamente il modulo NI 9881 su CompactRIO per la comunicazione CANopen con i servocontroller degli attuatori. Dal punto di vista software la rete CANopen è gestita dal programma real-time sviluppato in LabView per il target CompactRIO. In sostanza

questo tipo di comunicazione si basa sullo scambio di PDO (Process Data Object), che sono messaggi per la trasmissione di dati in real-time tramite CANopen, con cui gli attuatori ricevono le movimentazioni impostegli da pc (PDO in ricezione o RXPDO) e trasmettono al pc stesso informazioni sul loro stato (PDO in trasmissione o TXPDO). Si può facilmente configurare la comunicazione con i controller dei motori, collegandosi con il pc all'ingresso seriale RS232, tramite il software del fornitore, LinMot-Talk. In particolare nella sezione *PDO Mapping* si può decidere quali informazioni codificare in ciascuna delle quattro word esadecimali contenute in ogni PDO.
I parametri di configurazione del CANopen sono i medesimi per i tre azionamenti e di seguito viene riportata la configurazione dei PDO adottata:

PDO type	*word 1*	*word 2*	*word 3*	*word 4*
TxPDO1	Status Word	State Var	Logged Error Code	Warn Word
TxPDO2	Motion Command Status	Actual Position 16 Bit	Demand Current	Demand Position 16 Bit
RxPDO1	Control Word			
RxPDO2	Motion Cmd Header + Par Byte 0..5 [4 W]			
RxPDO3	Cmd Slave Header + Par Byte 6..7 [2 W]		Maximal Current	

Figura 1.7: Parameters: PDO Mapping

La configurazione adottata prevede le seguenti impostazioni:

- il TXPDO1 riporta informazioni relative allo stato in cui si trova l'attuatore, ad eventuali errori e warnings che possono capitare durante una movimentazione;
- il TXPDO2 comunica informazioni relative al comando che il motore lineare sta mettendo in atto, alla posizione attuale, alla corrente istantaneamente assorbita, alla posizione richiesta;
- il RXPDO1 contiene la sola *Control Word* che permette all'utente di richiedere lo stato in cui l'attuatore deve portarsi (ON,OFF, Homing…) seguendo la logica di una precisa state machine;
- il RXPDO2 contiene nella prima word il *motion command header* dove si specifica, tramite opportuni ID, il tipo di comando che si vuol far realizzare ai motori

lineari. All'interno di tale istruzione è contenuto anche il *command count*, un contatore dei comandi inviati. Nelle rimanenti tre word esadecimali di questo PDO sono specificati i parametri relativi al tipo di comando indicato nella prima word. Nel nostro caso il comando che è stato selezionato per svolgere la movimentazione è il "16 bit VAI Go To Pos", il quale necessita che vengano indicati quattro parametri: posizione target, massima velocità, massima accelerazione, massima decelerazione. Essendo solo quattro le word per PDO, ed essendo la prima già utilizzata per specificare il tipo di comando, si è costretti ad utilizzare un ulteriore messaggio per la comunicazione della decelerazione.

- Il RXPDO3 contiene appunto nella seconda word il valore massimo di decelerazione, mentre la prima word è dedicata al *command slave header*, cioè alla stessa istruzione scritta nella prima word del RXPDO2 con anche il command count invariato, proprio perché, nonostante i messaggi siano due, l'istruzione è unica. Per evitare di stressare eccessivamente gli organi movimentati dai due motori lineari, per scongiurare surriscaldamenti eccessivi degli attuatori, ma soprattutto per avere la possibilità di differenziare la forza esercitata dagli attuatori per ogni movimentazione richiesta, si è scelto di impostare dei limiti massimi di forza erogabile dagli attuatori durante le movimentazioni che, come si intuisce dalla descrizione dei PDO, sono effettuate con controllo in posizione. Per ottenere ciò la terza word esadecimale del RXPDO3 era stata impostata manualmente, in modo da contenere l'informazione relativa alla corrente massima assorbibile dai due motori durante le movimentazioni. Infatti, impostando un limite massimo sulla corrente, si va indirettamente ad imporre un limite sulla forza con cui i due attuatori vanno a spingere sulle leve di cambio, frizione e freno. Cosa interessante: tale limite è un parametro "live", cioè alterabile real-time.

Capitolo 2

L’ interfaccia utente

Dopo aver fatto una panoramica di tutti gli strumenti e i segnali che caratterizzano il sistema, l’ambiente in cui si va a inserire il progetto, si trattano le interfacce utente tramite cui si gestisce l’autopilota sul banco a rulli.
Il controllo dell’intero sistema avviene in modo facile e intuitivo grazie alla presenza di due strumenti di interfaccia messi a disposizione dell’operatore. Da una parte abbiamo la **Consolle** che dispone solo di quei controlli e indicatori strettamente necessari alla guida della moto e alla selezione delle modalità operative del sistema, dall’altra abbiamo il **pc Host**, che invece offre un’interfaccia più ricca e di maggior supporto alla gestione del sistema stesso. Attraverso essa, non solo è possibile gestire la moto durante un test, ma anche configurare tutti i parametri delle logiche implementate nel sistema.
La Consolle è un’interfaccia più semplice e intuitiva dove tutti i comandi sono a portata di mano, pertanto può risultare anche più comoda e adatta quando si deve fare una messa a punto in preparazione di un test, o quando si devono svolgere test di guida in modalità manuale/semi-automatica, ad ogni modo valida anche nei casi di emergenza, quando si deve agire tempestivamente e richiedere un arresto forzato del sistema al fine di evitare un pericolo o ridurre l’entità di questo.
Tali interfacce possiedono delle funzioni che sono in comune, perciò è a discrezione dell’operatore decidere quale dispositivo utilizzare per soddisfare le proprie esigenze. È stato anche deciso di configurare il sistema prevedendo la possibilità di utilizzare contemporaneamente entrambe le interfacce ma imponendo la condizione che, connettendole entrambe, la Consolle abbia la priorità sul pc Host, vale a dire che la parte dei comandi in condivisione nelle due interfacce viene disabilitata a lato del pc Host, evitando così che siano effettuate delle richieste concomitanti.

2.1 Interfaccia Host

L'interfaccia Host sviluppata in LabView dall'Ing. Michele Taccioli [1], è un'interfaccia completa attraverso la quale non solo si può gestire la moto durante un test, ma è anche possibile configurare, tramite apposite finestre pop-up, tutti i parametri relativi a:

- Leggi di moto degli attuatori;
- Cambio automatico;
- Rete CAN;
- Linearizzazione dei segnali analogici Input/Output moto;
- Profilo da riprodurre;
- Modello di gestione stati e cambiata.

La finestra principale dell'interfaccia Host è la seguente:

Figure 2.1: Main Screen Host Interface

1. *Host Menu*	2. *Bike Controls and Feedbacks*
3. *STATUS fdbk*	4. *Reset Profile Button*
5. *Elapsed time*	6. *Start Type*
7. *N cycles*	8. *ClutchON & BrakeOFF*
9. *Alarm Button*	10. *START&STOP controls*
11. *Communication led*	

Sul pannello della schermata principale sono visualizzate informazioni relative allo stato delle connessioni e alla ricezione dei segnali di feedback dal motoveicolo, oltre ad essere presenti vari controlli che permettono la gestione della moto.
Come si può notare il pannello è suddiviso per funzionalità e sono le seguenti:

1) ***Host Menu:*** si tratta di un menu a tendina che viene attivato cliccando su uno dei bottoni della barra e consiste appunto di una tendina di elementi alternativi selezionabili.

Si riportano di seguito gli elementi della tendina corrispondenti ai pulsanti (in ordine da sinistra verso destra):

- ***Screen:***
  - **Main Screen**: mostra il pannello principale.
  - **Actuators Control Screen**: mostra il pannello per la gestione manuale degli azionamenti di cambio, freno e frizione.
  - **Close**: chiude l'interfaccia Host e tutte le finestre pop-up eventualmente aperte.
- ***STATUS:***
  - **CALIBRATION**: formula la richiesta di stato Calibrazione.
  - **MANUAL**: formula la richiesta di stato Manuale.
  - **SEMI-AUTO**: formula la richiesta di stato Semi-Automatico.
  - **AUTOMATIC**: formula la richiesta di stato Automatico.
- ***Setup:***
  - **Actuators Configuration**: apre la finestra pop-up relativa alla configurazione delle corse degli attuatori.
  - **Auto Gearshift Configuration**: apre la finestra pop-up relativa alla configurazione dei parametri per eseguire la cambiata in automatico.
  - **CAN configuration:** apre la finestra pop-up relativa alla configurazione dei parametri della rete CAN per far dialogare il sistema con la centralina del motoveicolo.
  - **Gear Configuration:** apre la finestra pop-up relativa alla configurazione dei parametri della trasmissione del motoveicolo.

  - **Model Parameters Configuration**: apre la finestra pop-up relativa alla configurazione dei parametri del modello di gestione degli stati e della cambiata.
  - **Net Configuration**: mostra il pannello per impostare i parametri di rete.
  - **Profile Configuration**: apre la finestra pop-up relativa alla configurazione del profilo da inseguire in stato automatico.
  - **Signal Linearization Configuration**: apre la finestra pop-up relativa alla configurazione delle linearizzazioni dei segnali analogici input/output moto.
  - **Start Configuration**: apre la finestra pop-up relativa alla configurazione della partenza da inseguire in stato automatico a moto ferma sul banco a rulli.

- ***Actuators Monitor:***
  - Apre la finestra pop-up contenente gli indicatori di stato degli attuatori e altre informazioni relative ad essi.

- ***Upload INI:***
  - Scarica il file di configurazione contenente i valori dei parametri di sistema all'interno dell'elaboratore real-time.

- ***Help:***
  - Mostra la documentazione relativa al sistema.

2) ***Bike Controls and Feedbacks:*** comandi per la guida manuale della moto (tasti *UP*, *DOWN* e *NEUTRAL* per eseguire la cambiata e lo slider *THROTTLE %* per il controllo dell'acceleratore) e i principali segnali di feedback derivanti da essa. Abbiamo infatti degli indicatori che riportano i valori letti direttamente dalla porta CAN della moto, di velocità del veicolo, giri motore, valore di apertura della farfalla, *GEAR* e le temperature di olio e acqua motore.

3) ***STATUS fdbk:*** indica lo stato in cui si trova il sistema nella condizione attuale.

4) ***Reset Profile:*** pulsante che permette di resettare il profilo di guida da eseguire in stato Automatico.

5) ***Elapsed time:*** indica il tempo trascorso durante la simulazione in stato Automatico.

6) ***Start Type:*** comando per impostare la partenza nell'ambito di un test di guida in stato automatico, ovvero il tipo di legge di rilascio per la frizione e di gestione dell'acceleratore.

7) ***N cycles:*** comando per impostare il numero di riproduzioni del profilo di guida in stato Automatico.

8) ***ClutchON & BrakeOFF:*** interruttore che permette allo stesso tempo di tirare la frizione e rilasciare il freno. Rappresenta un'ulteriore possibilità di controllare la frizione e può risultare particolarmente comodo quando si devono affrontare casi di emergenza di bassa rilevanza, in cui magari non si desidera provocare l'arresto degli attuatori e lo spegnimento del motoveicolo.

9) ***Alarm Button:*** pulsante per richiedere lo STOP di emergenza.

10) ***START&STOP controls:*** comandi per controllare l'accensione e lo spegnimento del motore del motoveicolo.

11) ***Communication led:*** indica se è attiva la connessione tra PC Host ed il C-Rio, oppure se è attiva quella tra la Consolle ed il C-Rio.

2.1.1 Controllo degli attuatori

In questa sezione viene mostrato il pannello della schermata principale che permette la gestione degli attuatori di cambio, frizione e freno.
Nel pannello infatti, si ha la possibilità di muovere manualmente gli sliders degli attuatori tramite dei controllori grafici di scorrimento come si può vedere dalla seguente figura:

Figure 2.2: Actuators Control Screen

Al di sopra dei cursori per la gestione manuale degli attuatori si trova la parte di pannello riguardante il controllo e la gestione degli azionamenti. Con l'opportuno selettore (in alto a sinistra), si seleziona l'attuatore **Gear, Clutch, o Brake** a cui inviare i comandi di:

- **ON**
- **OFF**
- **Homing**
- **Reset**

Il comando **OFF**, toglie la potenza agli attuatori, dopodiché si possono muovere anche a mano; non viene disattivata la parte logica e il sensore di posizione. Premendo **ON**, l'attuatore torna ad essere attivo. Questa sequenza si può utilizzare per trovare i riferimenti fisici per configurare le corse degli attuatori. **Homing** è il comando che consente di eseguire la ricerca della posizione di homing (ovvero definisce l'origine del sistema di riferimento). **Reset** in caso di errore va a resettare il singolo nodo. Una volta eseguito il reset è necessario, per riattivare l'azionamento, premere in sequenza Spegni e Accendi.

Nella parte in alto a destra del pannello, il tasto **ON (+homing)** va ad accendere la rete, eseguire la procedura di homing e porta nello stato “operation enabled” tutti e tre gli attuatori.

2.1.2 Configurazione attuatori

Un successivo pannello pop-up, è quello dove è possibile configurare le leggi di moto degli attuatori da mettere in atto nelle specifiche movimentazioni (figura 2.3). Ogni legge di moto prevede cinque parametri da impostare:

- Posizione in mm;
- Velocità massima in m/s;
- Accelerazione in m/s^2;
- Decelerazione in m/s^2;
- Forza massima in N.

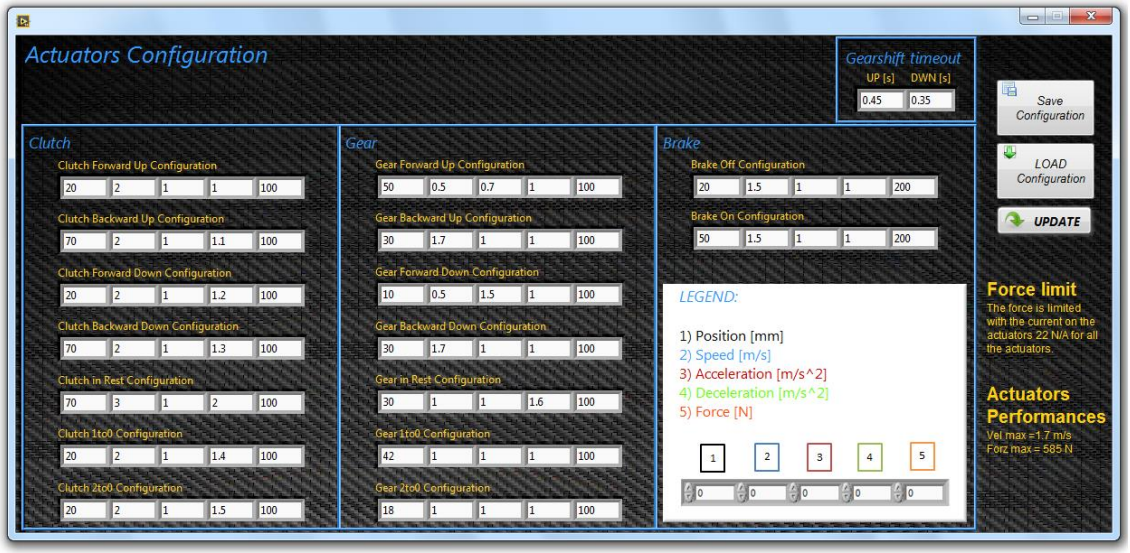


Figura 2.3: Particolare del pannello di configurazione delle corse degli attuatori

La schermata offre la possibilità all’utente di caricare (LOAD) una configurazione riportata su un file di testo, di aggiornare la configurazione attuale (UPDATE) e infine di aggiornare la configurazione attuale memorizzandola su file di testo (SAVE) in modo da renderla disponibile per un caricamento futuro.
L’attuatore del cambio per il suo funzionamento normale ha bisogno di definire, in una fase iniziale, i parametri a cui corrispondono le posizioni rilevanti della leva del cambio. Queste sono:

- **Gear Forward Up:** è la posizione da raggiungere premendo la leva del cambio verso l'alto per salire di marcia;
- **Gear Backward Up:** è la posizione di non attuazione, da raggiungere con la leva del cambio dopo che è stata inserita una marcia superiore. Tale configurazione, potrebbe essere utile per eseguire velocemente delle cambiate quando si deve salire di marcia; ad esempio per simulare una fase di accelerazione su un tratto di strada rettilineo. In sostanza, si potrebbe portare la leva del cambio in una posizione leggermente diversa da quella neutra (di riposo), simulando in questa maniera la pressione che eserciterebbe il piede del pilota sulla leva stessa prima dell'effettuazione della cambiata.
- **Gear Forward Down:** è la posizione verso il basso da raggiungere per effettuare una scalata di marcia;
- **Gear Backward Down:** è la posizione verso l'alto da raggiungere con la leva nella fase successiva all'attuazione della scalata;
- **Gear in rest:** è la posizione neutra da raggiungere quando ci si trova fra una cambiata e la successiva;
- **Gear1to0:** è la posizione da raggiungere per far entrare la folle quando ci si trova in prima;
- **Gear2to0:** è la posizione da raggiungere per innestare la folle quando si è in seconda.

Allo stesso modo, sono da configurare i posizionamenti ritenuti rilevanti per l'attuatore che agisce sulla leva della frizione. Questi corrispondono a:

- **Clutch Forward Up:** è la posizione che assume la leva quando ci si appresta a cambiare per salire di marcia;
- **Clutch Backward Up:** è la posizione da raggiungere per rilasciare la leva della frizione dopo che si è inserita una marcia superiore;
- **Clutch Forward Down:** è la posizione che assume la leva quando si esegue una scalata;
- **Clutch Backward Down:** è la posizione da raggiungere per rilasciare la leva frizione dopo che si è effettuata una scalata;

- **Clutch in rest:** è la posizione neutra a cui ci si va a settare la leva della frizione quando è completamente rilasciata;
- **Clutch1to0:** è la posizione che assume la leva quando la marcia inserita è una prima e ci si appresta ad inserire la folle;
- **Clutch2to0:** è la posizione che assume la leva quando la marcia inserita è una seconda e ci si appresta ad inserire la folle;

In ultimo**,** si hanno i posizionamenti caratteristici per l'attuatore che deve agire sul pedale del freno:

- **Brake ON:** è la posizione da raggiungere quando il pedale del freno viene premuto;
- **Brake OFF:** è la posizione da raggiungere quando il pedale del freno viene completamente rilasciato;

2.1.3 Configurazione trasmissione e cambio automatico

La finestra pop-up ***Auto Gearshift Configuration*** riportata in figura 2.4, serve a configurare il cambio automatico sul raggiungimento di una certa velocità ad una determinata marcia. Risulta comodo andare a mappare come limiti per le cambiate i giri motore. Sullo schermo vengono contemporaneamente visualizzate le corrispondenti velocità della ruota posteriore (*Rear_Speed*) calcolate partendo dai giri motore e conoscendo i rapporti di trasmissione della moto dall'albero principale, passando per il cambio, fino al mozzo della ruota. Conoscendo inoltre il diametro della ruota si hanno tutti gli elementi necessari per convertire un valore di *rpm* motore in velocità (km/h). Chiamando τ il rapporto di trasmissione complessivo di tutta la *driveline* si ha che:

$$Rear\_Speed\ [km/h] = \frac{rpm}{60} \cdot \tau \cdot \frac{Circonf[mm]}{1000} \cdot \frac{3600}{1000}$$

Il sisitema di controllo, per come è stato cinfigurato, utilizza il riferimento dei giri motore (letti via CAN) quando deve comandare un *Upshift*, mentre quando deve comandare un *Downshift* utilizza il riferimento relativo alle velocità della ruota posteriore (letta via CAN).

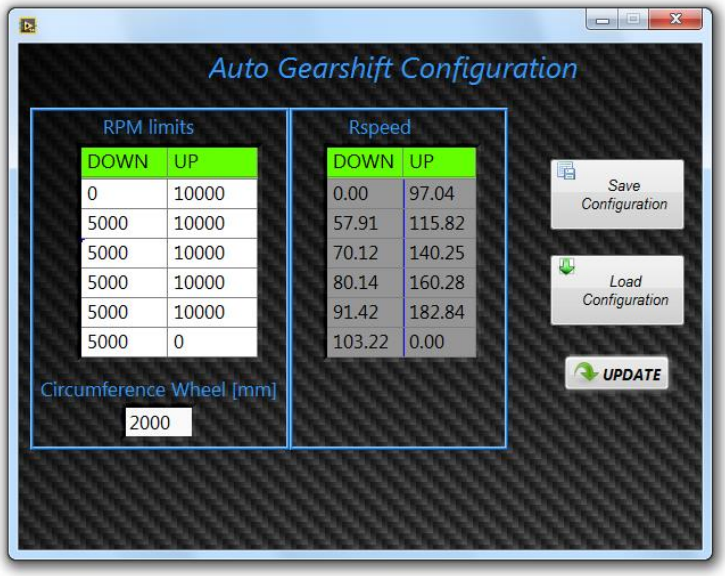


Figura 2.4: Particolare del pannello *Auto Gearshift Configuration*

In figura 2.5 è riportato il pannello pop-up per la configurazione della trasmissione. Nella sezione ***Clutch Bell Sprocket/Main Shaft*** occorre inserire il rapporto di trasmissione primario, ovvero quello fra campana frizione e la ruota primaria del cambio. In quella del ***Gearbox Ratios*** si devono impostare tutti i rapporti dei denti del cambio per le sei marce disponibili. L'ultimo rapporto di trasmissione da inserire è quello fra i denti del pignone in uscita dal cambio e la corona della ruota posteriore nella sezione ***Front Sprocket /Rear Sprocket***.

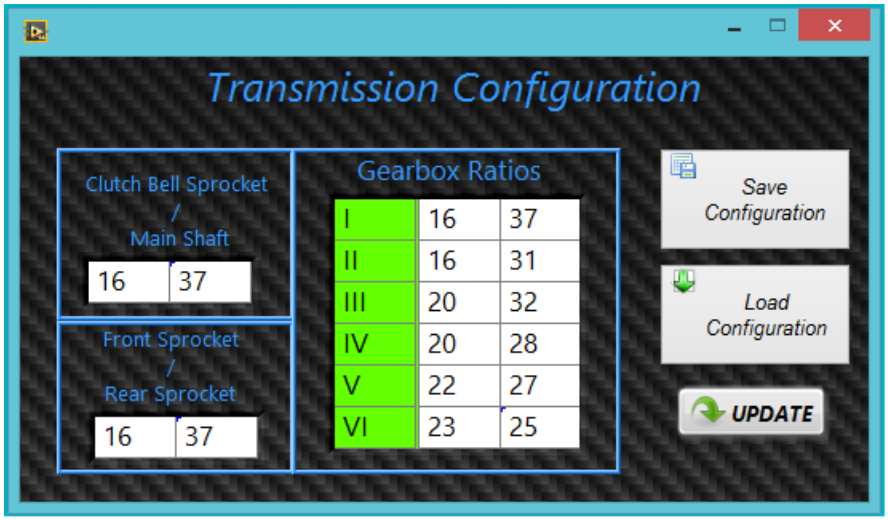


Figura 2.5: Particolare del pannello *Transmission Configuration*

Per evitare di dover scrivere tutte le volte i valori nei vari campi si può salvare e caricare la configurazione del pannello. Il salvataggio va a scrivere su di un file di testo un unico array che contiene i vari elementi del pannello, si aprirà quindi una finestra che chiederà di immettere un nome per il file. Per il caricamento dei dati invece viene chiesto di selezionare un file precedentemente salvato.

2.1.4 Configurazione dei parametri del modello

La seguente finestra pop-up, riportata in figura 2.6, consente di visualizzare e variare gli attuali valori dei parametri scalari del modello real-time che si occupa della gestione degli stati e delle cambiate (modello Simulink presentato nel capitolo 4). Nella finestra si trovano i seguenti pannelli:

1. **Inports**, dove si possono visualizzare gli input del modello;
2. **Outports**, dove si possono visualizzare gli output elaborati dal modello;
3. **Parameters**, dove si riportano quei parametri del modello che possono essere comodamente modificati *inline* dall'operatore che accede a tale pannello.

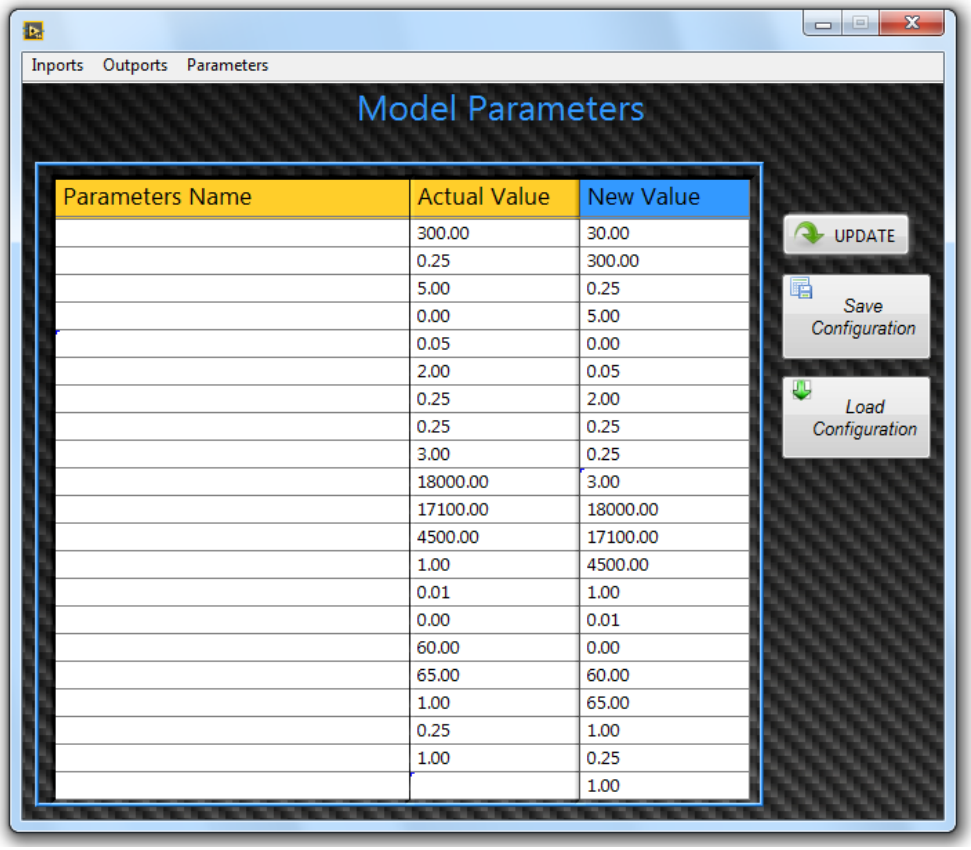


Figura 2.6: Particolare del pannello *Model Parameters Configuration*

2.1.5 Configurazione del profilo di guida

In questa sezione viene descritta la finestra pop-up in cui si va a configurare un profilo di guida da far riprodurre al motoveicolo quando il sistema si trova ad operare in stato automatico. La finestra ***Profile Configuration*** si compone di due pannelli:

1) Pannello **Screen** in cui si visualizza su un grafico base tempo, le grandezze target da inseguire durante la riproduzione del profilo di guida in stato automatico;

2) il pannello **Config**, che permette di caricare il file di testo in cui è memorizzato il profilo e impostare le tolleranze di velocità e tempo che devono essere rispettate durante la riproduzione del ciclo di guida. Il contenuto del file di testo viene mostrato in una tabella.

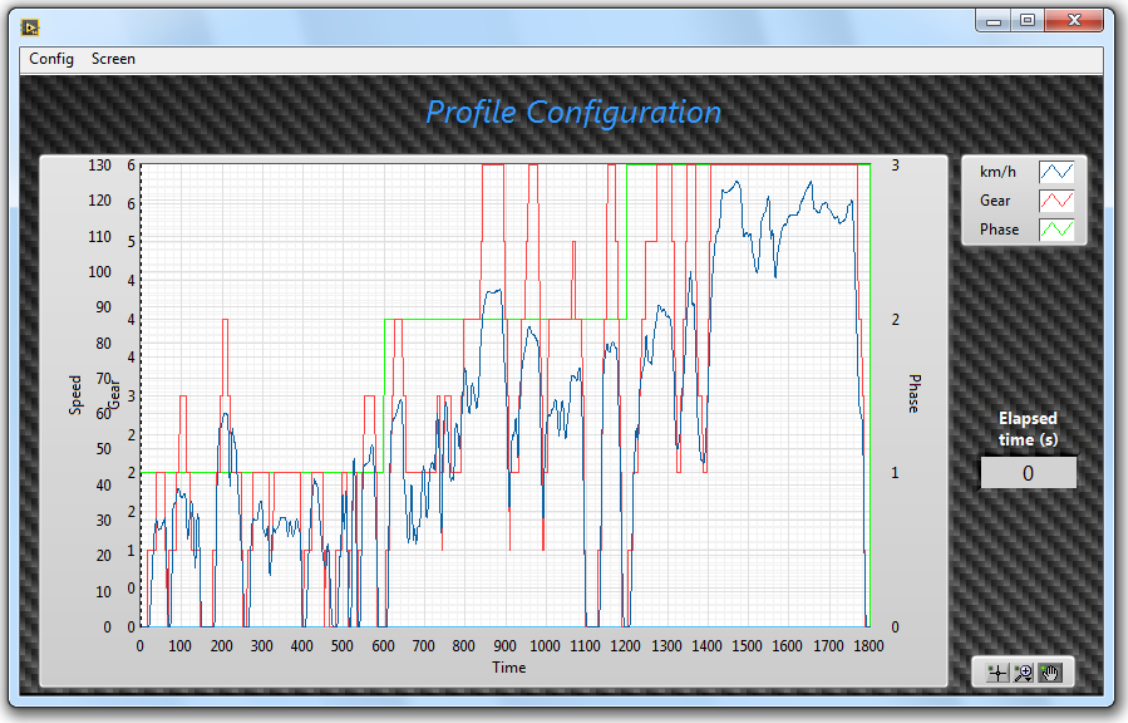


Figura 2.7: Particolare del pannello *Screen*

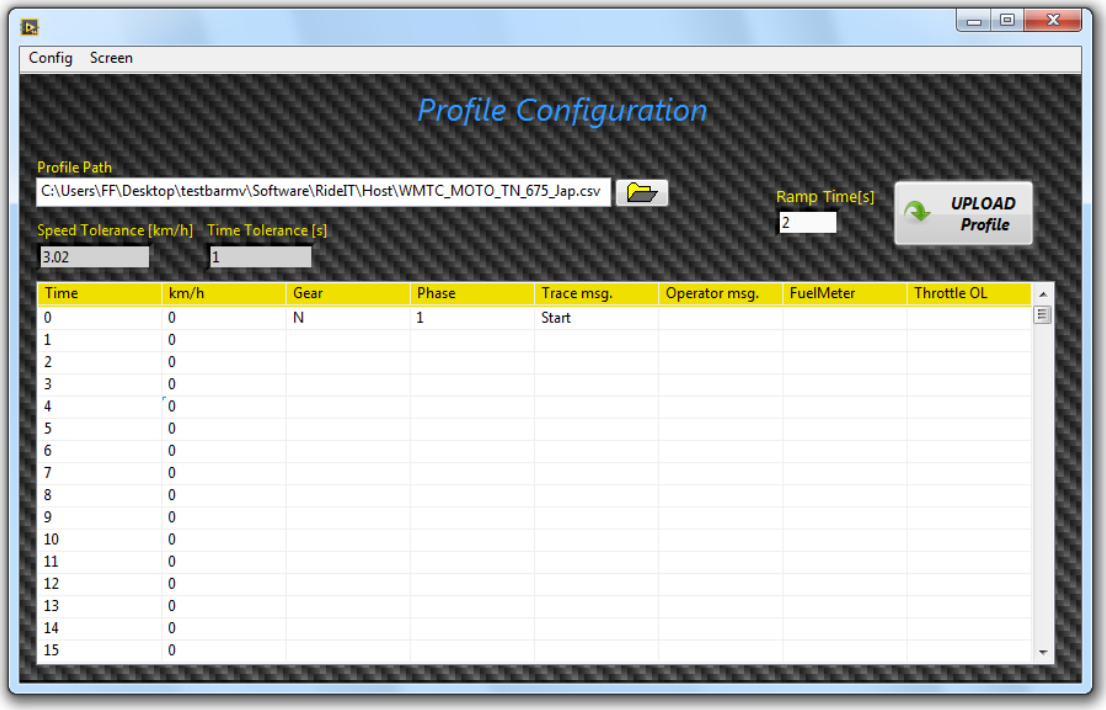


Figura 2.8: Particolare del pannello *Config*

Il *Ramp Time* è il tempo di rampa, ovvero il valore della durata della rampa di raccordo che si attiva ogni qual volta si rileva il passaggio allo stato automatico. Durante tale periodo occorre che il valore di velocità del rullo si porti al primo valore di velocità di setpoint impostata sul profilo del ciclo di guida automatico.
Per spiegare meglio il concetto di rampa e la funzione del pulsante ***Reset*** presente nel pannello *Main Screen* prendiamo in esame la parte di modello creata in ambiente LabView per il target real-time, riguardante il profilo automatico (figura 2.9).
Dalla logica del modello si può capire che, affinché venga eseguito il loop del profilo automatico, bisogna innanzi tutto portare il sistema in stato AUTOMATICO.
Nel momento in cui si entra in stato automatico si comincia a contare il tempo di rampa nell'arco del quale viene eseguito il primo loop relativo alla rampa stessa (condizione impostata a true sul loop di rampa). Il tempo di rampa (*ramp time*), ha un limite imposto dall'utente pari a 2 sec e fino a che, non si arriva al raggiungimento di quel limite, il profilo del ciclo automatico non viene eseguito. Successivamente, viene eseguito il profilo e, il primo istante che viene rilevato dal modello, risulta essere quello iniziale previsto dal ciclo se è la prima volta che viene avviato il sistema e si entra in stato automatico, oppure anche nel caso in cui è stato resettato in

precedenza il profilo premendo l'apposito tasto *reset* da interfaccia Host. In alternativa, è possibile che il ciclo cominci da un istante che risulta essere diverso da quello iniziale previsto, ma vorrebbe dire che in un test eseguito a monte, per un qualche motivo si è usciti anticipatamente dallo stato automatico ancor prima di aver terminato il profilo per poi rientrarci in una fase successiva senza premere il tasto reset. Allora l'istante iniziale che viene rilevato dal modello in quel caso, non corrisponde più al primo istante previsto per il profilo, ma corrisponde all'ultimo valore letto prima dell'uscita dal profilo stesso e a cui è associato un ben determinato valore di velocità del motoveicolo. Per far fronte a tutti questi casi, si è previsto che venga eseguita una rampa di raccordo non appena si entra in stato automatico, prima di eseguire il profilo (*Elapsed time* rimane fisso), al fine di raccordare la velocità del rullo (*Roll Speed*), ovvero del motoveicolo, con il primo valore target previsto dal profilo.

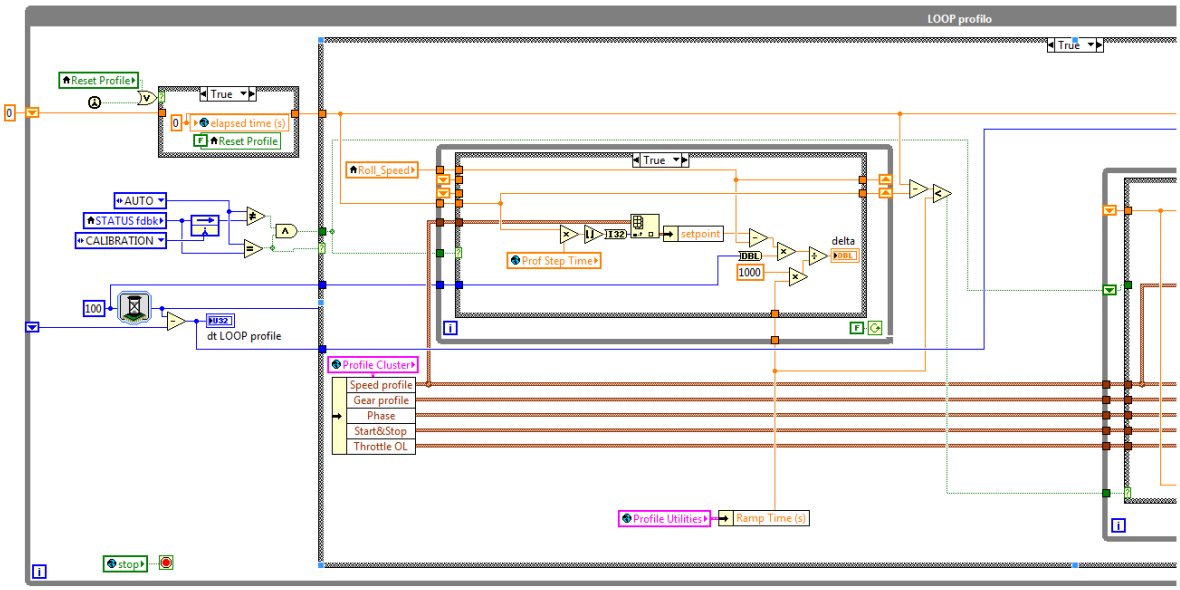


Figura 2.9: Loop del Profilo automatico (I parte)

Durante quei 2 sec previsti dal *ramp time* viene eseguito ad ogni *step time* del loop il seguente calcolo:

$$delta\ (\frac{km}{h}) = \frac{(Speed\_setpoint - Roll\_Speed)\ \cdot\ Loop\_step\_time(ms)}{Ramp\_time\ (ms)}$$

Si ricava in questo modo il gradino di velocità (delta) di cui deve essere incrementata o decrementata la velocità target, ad ogni step di calcolo del loop per raggiungere al termine della rampa un ben determinato setpoint di velocità del profilo.
La condizione di rampa, durante la quale il setpoint di velocità cambia continuamente, non viene imposto invece per gli altri setpoint delle grandezze interessate dal profilo quali ad esempio la marcia inserita, poiché per queste devono essere impostate correttamente dall'utente prima di entrare nello stato AUTO, altrimenti il modello fa in modo di non permettere l'accesso a tale stato.
La *speed setpoint* che ad ogni step di calcolo viene calcolata all'interno della rampa viene inviata alla parte di modello real-time (descritta nel capitolo 4) sempre integrata nel progetto LabView Real-Time, dove è implementato un controllore PID che decide sulla base dei target di velocità, come deve essere gestito l'acceleratore sul motoveicolo.
Al termine della rampa, il profilo di guida in stato automatico viene eseguito e questo viene visto nel modello real-time attraverso la *case structure* riportata in figura 2.10, quando viene portata a "False" la condizione sul *case*.

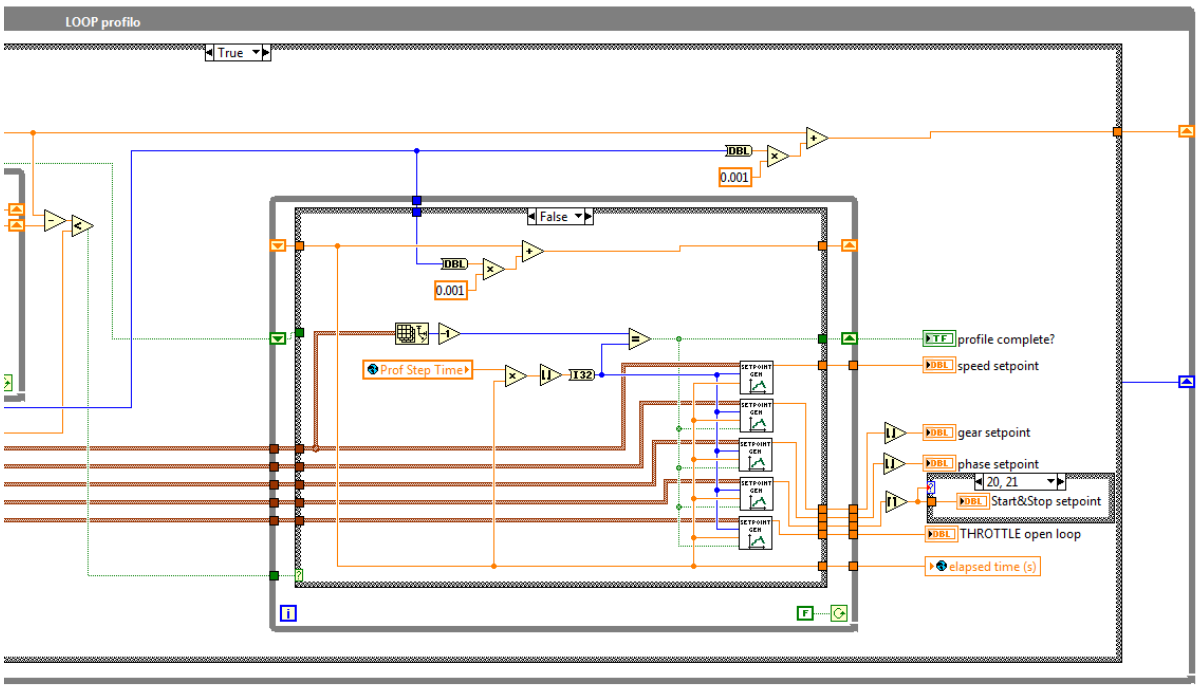


Figura 2.10 Loop del Profilo automatico (II parte)

Elapsed time in questa fase viene incrementato. Questo contatore oltre ad essere reso noto all'utente serve nel modello per selezionare, a ogni step di calcolo, gli indici corretti dai blocchetti generatori di set point, per tutti quei parametri del profilo (speed, gear, phase, Start&Stop, Throttle_OL) che devono essere inviati in input alla parte modello che si occupa di gestire le attuazioni (si veda sempre capitolo 4).

2.1.6 Configurazione della rete

Il pannello ***Net Configuration*** della schermata principale, permette di inserire gli indirizzi IP della Consolle, del pc Host e dell'elaboratore real-time (RT).

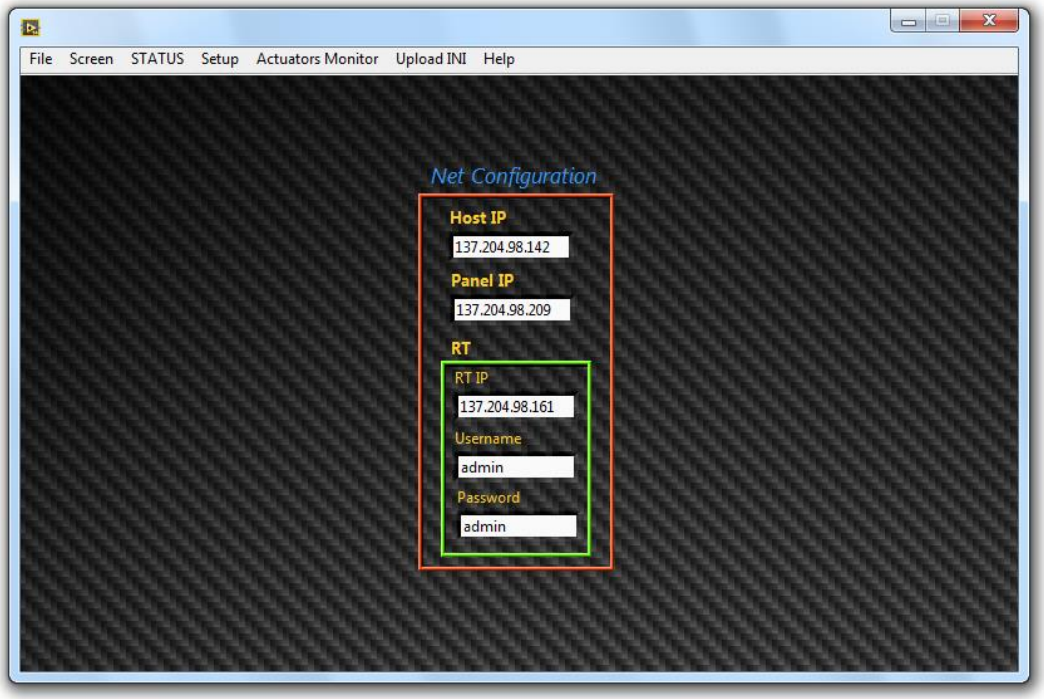


Figura 2.11: Particolare del pannello *Net Configuration*

2.1.7 Configurazioni dei segnali analogici

La finestra pop-up chiamata ***Signal Linearization Configuration*** offre la possibilità di effettuare la calibrazione dei segnali I/O analogici sulla scheda NI 9381, che sono:

- I due segnali dei potenziometri dell'acceleratore che devono essere inviati alla centralina del motoveicolo;
- Il segnale da acquisire relativo al sensore collegato alla leva di rinvio del cambio che ne rileva la forza (cella di carico).

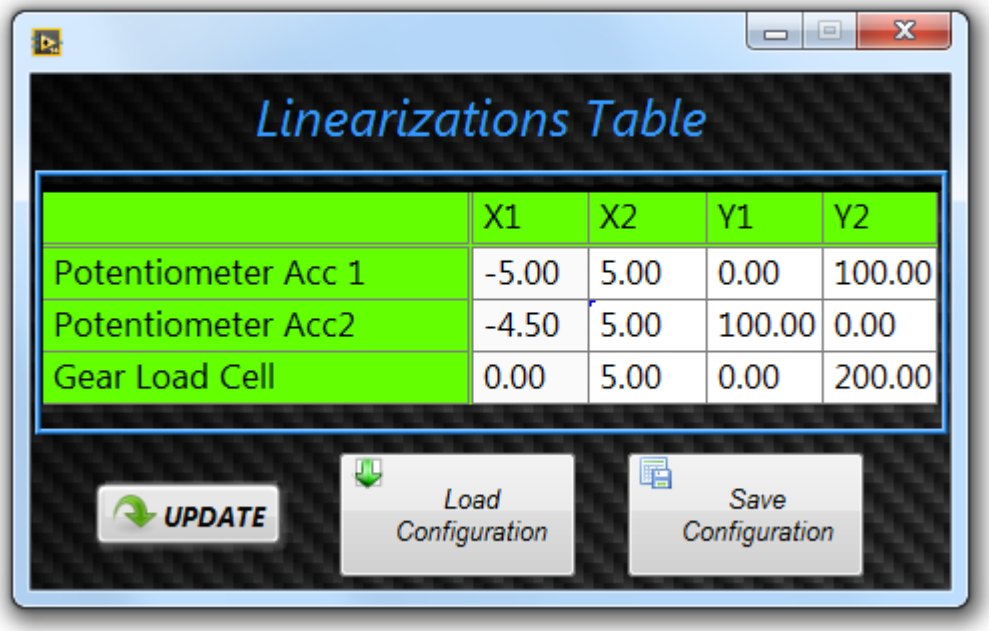


Figura 2.12: Particolare del pannello *Signal Linearization Configuration*

2.1.8 Configurazione della CAN

Questa finestra pop-up consente di configurare la rete CAN dalla quale andare a leggere le informazioni utili al modello real-time dalla centralina del motoveicolo (Gear, Throttle, T_oil, T_H2O, RPM e la Rear_Speed).

La finestra pop-up è costituita da tre pagine, poste una di seguito all'altra. Nella prima pagina, mostrata in figura 2.13, è possibile scegliere l'interfaccia CAN su cui lavorare. Nel nostro caso si ha la possibilità di scegliere solo un interfaccia, la **CAN1**, poiché il modulo NI 9862 è dotato di una sola porta CAN. Occorre inoltre, scegliere un database CAN tra quelli disponibili nell'apposito menu a tendina. Il database in questione può essere anche ricercato da un archivio interno al PC premendo l'apposito tasto *Load Database from file on PC*, oppure tramite il tasto *Use Deployed Database on RT*, può essere ricercato un database residente nel real-time (cRIO). Il database è un file *xml* o *dbc* (questi sono i formati tipici) che una volta selezionato viene caricato nella lista del menu a tendina. Il computer associa al file del database un **alias**, sostanzialmente un nome più abbreviato, quando lo carica nel programma. Questo fa in modo che il file venga effettivamente visto dal computer come un database CAN e non come un file di un altro tipo. Altro parametro fondamentale che deve essere definito quando si sceglie il database è il

Baud Rate (detto anche symbol rate o tasso di simbolo), ossia il numero di simboli trasmessi al secondo. È fondamentale che in una rete CAN, tutti i nodi, per poter comunicare, siano impostati allo stesso *baud rate.*

Una volta definito il database da utilizzare, con associato un alias e un *baud rate* corretto è possibile passare alla pagina successiva tramite il tasto Next.

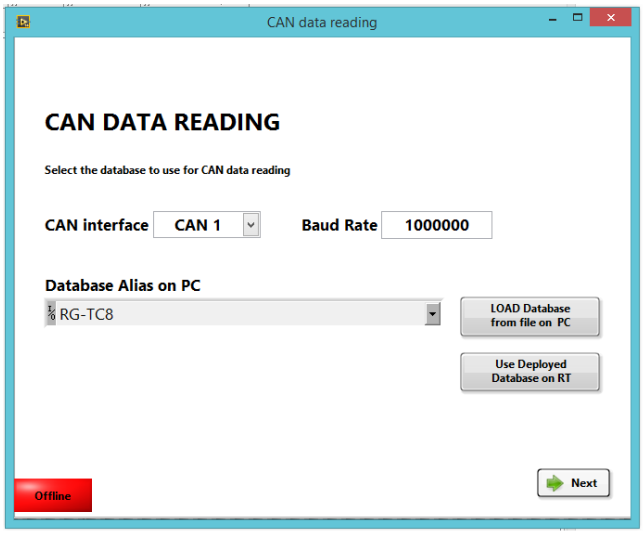


Figura 2.13: Particolare del pannello *CAN Configuration* – Pagina1

Nella seconda pagina (figura successiva 2.14), viene mostrato come è strutturato il database CAN e le informazioni relative al database stesso, sulla base dei livelli da cui è costituito.

Il database risulta infatti strutturato su 3 livelli che possono essere visualizzati selezionando gli elementi del riquadro *Database preview*. Nei vari livelli abbiamo:

1. Il **Cluster**: occupa il livello più elevato nella struttura gerarchica del database;
2. Il **Frame**: occupa una posizione intermedia;
3. Infine abbiamo i **Segnali** che rappresentano il livello più basso;

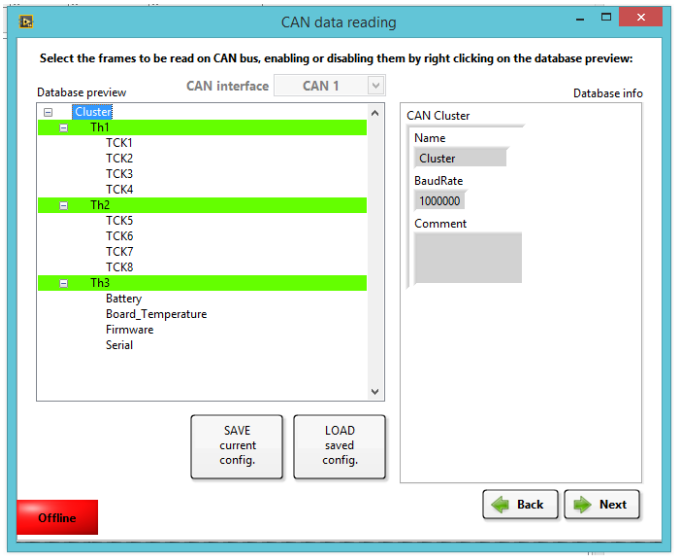


Figura 2.14: Particolare del pannello *CAN Configuration* – Pagina2 – Proprietà CAN Cluster

Il primo livello, rappresentato dai Cluster, riguarda l'hardware, ovvero le porte fisiche disponibili per la CAN (ad es: CAN1, CAN2, ecc..). Nel riquadro *Database info* posizionato sulla destra della pagina è possibile visualizzare le proprietà del Cluster, senza poter apportare alcuna modifica. All'interno di ogni cluster (nel nostro caso c'è solo 1 Cluster, avendo una sola porta CAN), ci sono i Frame, ossia i pacchetti di segnali. Selezionando un frame (ad es. uno tra Th1, Th2, Th3) sul riquadro *Database preview*, è possibile visualizzarne le proprietà sul riquadro di destra (figura 2.15). La funzione più importante in questa pagina, è quella di poter decidere quali Frame devono essere abilitati o meno all'interno di un database, senza però poter modificarne le proprietà del database stesso. È possibile infatti selezionare un Frame dal riquadro *Database preview* e decidere se abilitarlo o meno cliccando su uno dei tasti *enable* o *disable* in un apposito menu a comparsa. Nella pagina vengono in definitiva visualizzati i **frame abilitati**, ovvero quelli che devono essere letti dal target real-time, contraddistinti da una barra evidenziata in verde, poi ci sono i **frame disabilitati**, ovvero che non devono essere letti, contraddistinti da una barra evidenziata in rosso. È possibile salvare la configurazione corrente relativa ai frame abilitati e disabilitati nel database in cui si sta lavorando, oppure è possibile caricare un tipo di configurazione salvata nel PC, che sarà comunque relativa ai frame abilitati e disabilitati del medesimo database.

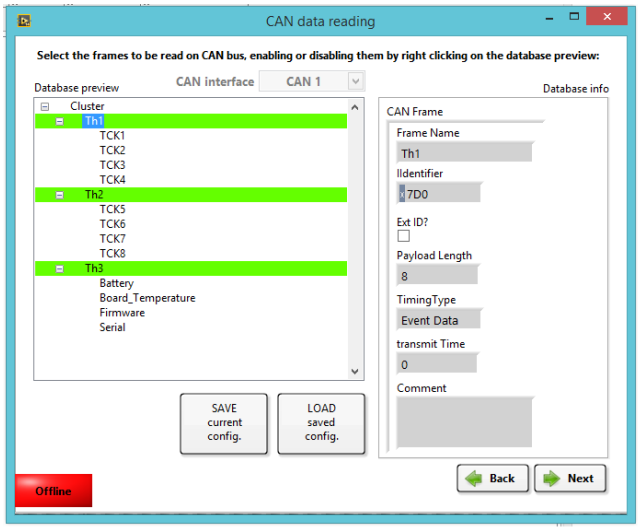


Figura 2.15: Particolare del pannello *CAN Configuration* – Pagina2 – Proprietà CAN Frame

Per ogni frame vengono mostrate le seguenti proprietà:

- **Nome** del frame;
- **ID**, ha la funzione di identificativo del pacchetto dati CAN. Viene espresso in formato esadecimale;
- **Payload**, ossia il contenuto dei dati nel frame, che può andare da un massimo di 8 bytes (64bit) ad un minimo di 0;
- **Timing Type** per decidere se la lettura o il trasferimento dei pacchetti nella rete CAN deve essere condotta in modo ciclico (*Cyclic*) o su base evento (*Event*);
- **Transmit Time** definisce il tempo intercorrente tra due cicli se il Timing Type è impostato su Cyclic;

Sotto ogni Frame, in *Database preview*, viene mostrato l'insieme dei segnali che compongono il frame stesso (es: TCK1, TCK2,ecc..).

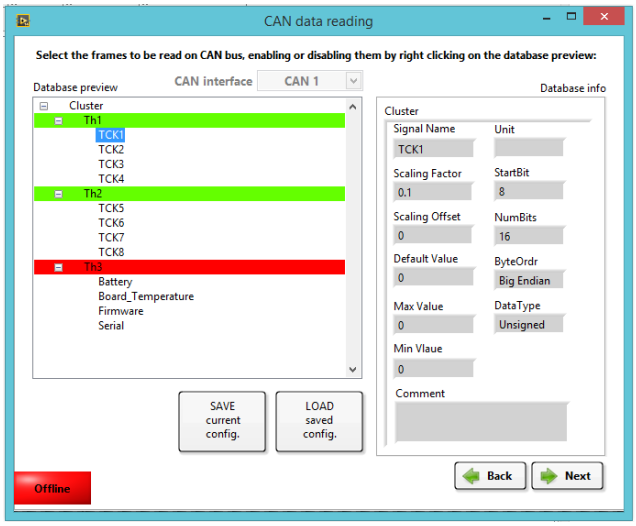


Figura 2.16: Particolare del pannello *CAN Configuration* – Pagina2 – Proprietà Segnali

Con riferimento alla figura 2.16, alcune delle proprietà dei segnali sono:

- **Start bit**: indica la posizione di inizio del segnale all'interno del frame.
- **Number of bits**: lunghezza del segnale all'interno del frame.
- **Data type**: indica il tipo di dato (*signed*, *unsigned*, o *float*).
- **Byte order**: *little* o *big endian*, sono modalità differenti per scrivere/leggere i segnali; in modalità *big endian* i bytes vengono inviati/letti dal più significativo al meno significativo poiché i bit più significativi del segnale sono collocati su indirizzi di byte inferiori all'interno del frame; l'opposto avviene per la modalità *little endian*.
- **Scaling factor** e **offset**: è utile alla conversione dei dati fisici e passare ad una rappresentazione in binario.

Una volta selezionati i frame desiderati all'interno del database e salvata la configurazione, si può passare alla terza pagina in cui vengono visualizzate le liste di tutti i segnali che si è deciso di leggere.

L'operazione da compiere arrivati a questo punto è quella di associare alla lista dei segnali disponibili, le grandezze feedback di interesse, stabilendo anche il periodo con cui andarle a leggere nella CAN. Per quanto riguarda i segnali a cui non vengono associate le grandezze, vengono lasciati in modalità *NOT USED*. Terminate le ultime impostazioni si fa il "deploy to target" che sostanzialmente consiste nel trasferire il database con le nuove configurazioni all'interno del target real-time.

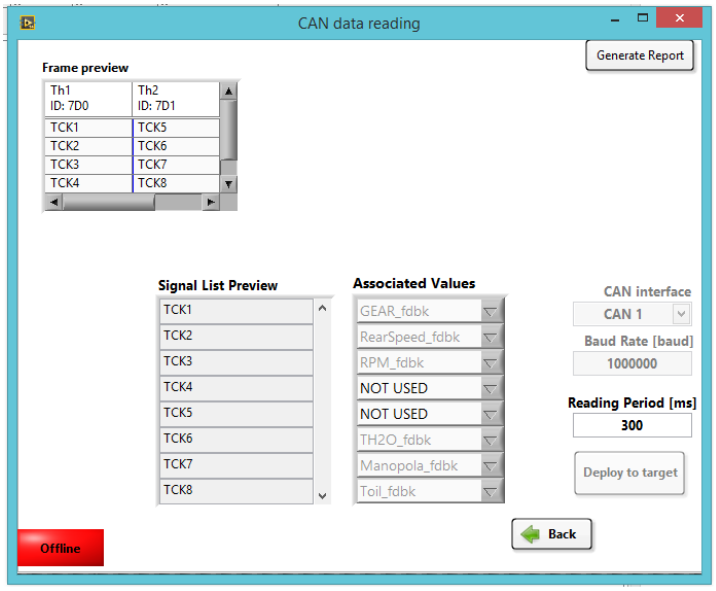


Figura 2.17: Particolare del pannello *CAN Configuration* – Pagina3

2.1.9 Configurazione della Partenza

Per quanto concerne il profilo di guida automatico è possibile configurare la procedura partenza tramite l'apposita finestra pop-up ***start configuration***.

Tale strumento, mostrato in figura 2.18, può essere utilizzato per creare, aggiornare, caricare e salvare facilmente una procedura per eseguire la partenza a moto ferma sul banco a rulli, quando si passa in stato di guida automatico.

Dal pc Host, ed in particolare dal *Main Screen*, è possibile impostare il tipo di procedura di partenza (*start type*) tra due opzioni disponibili. Entrambe, come è visibile sempre dalla figura 2.18, sono liberamente configurabili in termini di setpoint per le attuazioni di frizione e farfalla.

Occorre infatti andare ad inserire negli appositi spazi delle tabelle, i dati per costruire dei profili di corsa in funzione del tempo per la gestione dei comandi di farfalla e frizione, facendo attenzione a riempire i campi in modo corretto e sensato, in quanto il software non permette di scaricare dei parametri al modello real-time se riconosce che la configurazione non è valida. Non è dunque consentito fare un SAVE, piuttosto che un UPDATE, nel caso si sia creata una configurazione dove ad esempio, la colonna relativa agli istanti temporali non pareggia in numero di elementi quella relativa alle posizioni dell'attuatore, oppure nel caso in cui gli istanti temporali non siano inseriti esattamente in ordine crescente.

I parametri (vettori) delle tabelle una volta validati, vengono trasferiti al modello real-time; allo stesso tempo è possibile osservare i grafici dei valori mappati per le due tipologie di partenza dalla finestra di *start configuration* e quindi avere un quadro più chiaro sul tipo di procedura che si sta mettendo in atto.

In figura 2.18 sono riportati due esempi di procedure di partenza; una prevede un tipo di procedura veloce mentre nell'altro caso si ha una procedura più lenta.

Figura 2.18: Particolare del pannello *Start Configuration*

2.1.10 Actuators Monitor

Per monitorare lo stato dei servo controller e dei motori LinMot da interfaccia Host, si ha un pannello in cui si riportano le seguenti informazioni:

- Stato operativo dei controller
- Status word
- Warnings word
- Logged Error Code
- Motion Command Status
- Actual Position 16 bit [mm]
- Demand Current [A]
- Demand Position [mm]

Il pannello è visibile in figura 2.19. Come si vede è suddiviso in tre sezioni per distinguere le informazioni riguardanti gli attuatori di freno frizione e cambio.

Figura 2.19: ***Actuators Monitor***

2.2 La Consolle

Come già è stato accennato inizialmente, il controllo in remoto del motoveicolo durante i test di guida al banco a rulli può essere gestito anche tramite la Consolle, quando questa viene collegata al sistema.
Essa costituisce l'elemento principale dell'attività di tesi e rappresenta una periferica di controllo del sistema RideIT, creata con l'intento di rendere più diretta e confortevole l'esperienza di guida da parte dell'operatore, approssimativamente come se stesse in sella al motoveicolo. La consolle, offre appunto un'interfaccia intuitiva e accattivante in quanto replica fedelmente un manubrio vero di un motoveicolo, oltre a mettere a disposizione un pannello ricco di comandi a cui l'operatore può accedere quando deve interagire con il sistema. La maggior parte di questi comandi sono comunque comuni a quelli presenti sull'interfaccia Host, in quanto la consolle deve poter essere rimpiazzata pienamente dall'interfaccia Host quando viene disconnessa dal sistema.
La Consolle costituisce anche un'interfaccia di tipo programmabile, e dunque come l'interfaccia Host, dispone di un intelligenza propria capace di acquisire e processare i dati comunicando inoltre con il target real-time (CompactRIO) per decidere quali azioni possono essere o meno intraprese sul sistema. Questa flessibilità, opportunamente sfruttata dal software di gestione del dispositivo, permette di fare il controllo e la gestione di tutte le tipologie di comando installate a bordo, nonché, se necessario in una fase di messa a punto o installazione futura, offrire la possibilità di intervenire velocemente per modificare o al più integrare nel dispositivo funzionalità aggiuntive rispetto a quelle già implementate.

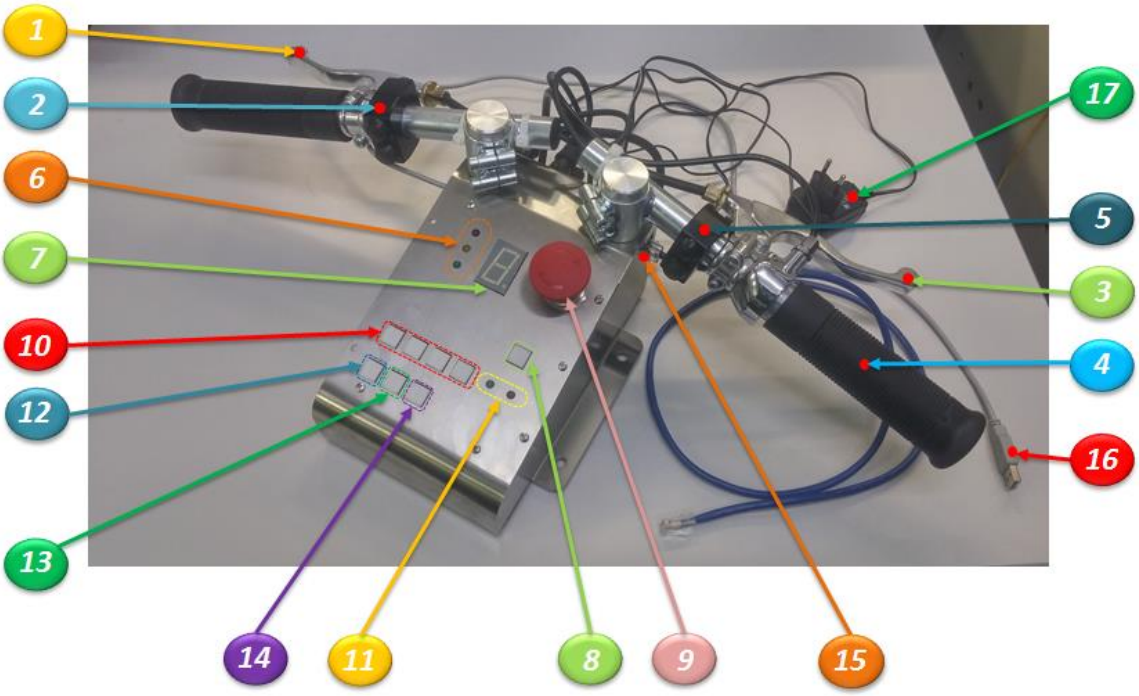


Figura 2.20: Fotografia della Consolle e illustrazione delle sue funzionalità

1. *Leva di controllo della frizione*
2. *Blocchetto elettrico con pulsanti Upshift, Downshift, Neutral*
3. *Leva di controllo del freno*
4. *Comando a manopola dell'acceleratore*
5. *Blocchetto elettrico con pulsanti di Start e Stop motore*
6. *Spie di diagnosi: Led Verde (connessione stabile), Led Arancione (warning velocità motoveicolo fuori range), Led Rosso (errore sugli attuatori)*
7. *Display indicatore di marcia inserita*
8. *Pulsante con Led di ClutchON&BrakeOFF*
9. *Pulsante "a fungo" di Alarm*
10. *Pulsanti con Led per la selezione dello stato (Calibration, Manual, Semi-Auto, Auto).*
11. *Led indicatori per gli stati di Idle e Alarm*
12. *Pulsante con Led di PanelCAL*
13. *Pulsante con Led di EditCAL*
14. *Pulsante con Led aggiuntivo*
15. *Porta Ethernet*
16. *Cavo USB per la programmazione rapida del microcontrollore*
17. *Alimentatore esterno (5V)*

Come si osserva dalla figura 2.20, tramite la Consolle è possibile gestire i tradizionali comandi di guida, quali:

- **Acceleratore**: è presente un classico comando a manopola con il quale l'operatore può richiedere la coppia da erogare al motore.
- **Frizione**: gestita grazie alla classica leva al manubrio riprodotta anche sulla Consolle
- **Freno**: la leva del freno riprodotta sulla consolle in questo caso fa riferimento all'azionamento del freno posteriore del motoveicolo.
- **Cambio**: per la gestione del cambio si hanno dei tasti +/- sul manubrio che permettono di effettuare dei cambi marcia a salire e a scalare sulla moto, inoltre c'è un tasto specifico per l'inserimento della folle.
- **Start/Stop**: classici pulsanti di Accensione/Spegnimento del motore presenti su un manubrio di un motoveicolo.

La Consolle è inoltre dotata di altri numerosi comandi, più caratteristici per il sistema, e sono installati sul pannello frontale. Questi sono:

- Il **tastierino degli stati** coi relativi indicatori per consentire di selezionare una modalità di impostazione desiderata del sistema durante un test (MANUAL, AUTOMATIC, SEMI-AUTOMATIC, CALIBRATION) e visualizzarne l'attivazione. Per ognuno dei pulsanti un indicatore a LED integrato informa l'utente su quale degli stati si è portato il sistema.
- **Indicatori a LED** per informare l'utente circa lo stato di connessione tra cRIO e Consolle e quelli dedicati allo stato di funzionamento degli attuatori.
- **Display a 7 segmenti** per informare l'utente su quale marcia viene innestata, o in caso di una calibrazione, per informare l'utente su quale tipo di comando sta eseguendo appunto la calibrazione.
- Il **pulsante a fungo**, per richiedere l'arresto in emergenza garantendo lo spegnimento rapido degli attuatori e della moto.
- Il pulsante di **ClutchON+BrakeOFF** (premendo il pulsante si tira la frizione e si rilascia il freno) con il relativo LED di segnalazione.

- I pulsanti di **Panel_CAL** e **Edit_CAL** impiegati per la procedura di calibrazione dei sensori delle leve di comando della Consolle.
- Il **pulsante Spare**, ovvero aggiuntivo (privo di funzione).

L'insieme di tutti questi comandi hanno portato alla definizione degli aspetti hardware, sia di quelli software (presentati nel capitolo 3), nonché di quelli operativi della Consolle.

2.2.1 Hardware della Consolle

La Consolle è stata pensata per essere utilizzata stando seduti comodamente alla scrivania garantendo anche una possibilità di fissaggio al piano della scrivania stessa, evitando di sbattere il dispositivo durante l'uso. La scatola che racchiude il circuito, i sensori e che costituisce un supporto per tutti i comandi e indicatori della Consolle è stata progettata tenendo in massima considerazione gli aspetti legati alla forma, alla funzione, all'ingombro, al layout dei componenti, e infine quelli di lavorazione e montaggio. Di seguito viene riportato il disegno 3D che è stato realizzato per mettere in opera il dispositivo.

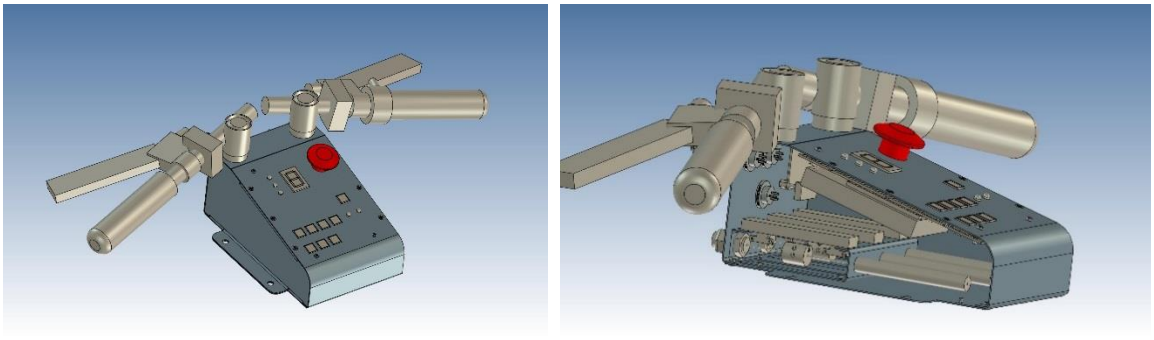


Figura 2.21 Disegno 3D della Consolle

Proprio per l'alto numero di controlli utilizzabili, si è ritenuto conveniente realizzare un software per la gestione della Consolle, in grado di effettuare la lettura di tutti i segnali input/output (I/O) digitali/analogici (D/O) e trasmetterli al cRIO sfruttando una comunicazione digitale per mezzo di un *bus protocol* piuttosto che ricorrere alla trasmissione per via diretta, che è di per sé meno robusta e comporta un maggiore impiego di cavi elettrici. Tramite la porta Ethernet quindi, la Consolle riceve i segnali inviati dal cRIO e allo stesso tempo invia tutti i segnali letti dai comandi presenti on-

board, in tale modo è possibile da una parte fare arrivare le richieste dell'operatore riguardo agli azionamenti del sistema, dall'altra monitorare l'andamento di tutte le variabili misurate per comprendere il comportamento del sistema stesso, comprese le problematiche che possono emergere durante un test o in una fase di messa a punto. Tra i dispositivi di comando presenti sulla Consolle ci sono quelli di tipo **analogico**, come il comando dell'acceleratore, la leva del freno e della frizione, e quelli di tipo **digitale** nel caso degli User Switch, User LED e del display indicatore di marcia. Come sensori utilizzati per la misura di posizione dei comandi analogici, si hanno dei potenziometri a guida lineare alloggiati nel vano interno dello chassis. Ciascuno di essi è collegato al corrispondente dispositivo di comando tramite un rinvio meccanico, ossia un filo d'acciaio che viene fatto scorrere all'interno di una guaina per poi andare a muovere il cursore del potenziometro. Al fine di garantire anche l'azione di richiamo su ciascun comando si è previsto di utilizzare delle molle da bloccare ad una estremità di ogni filo.

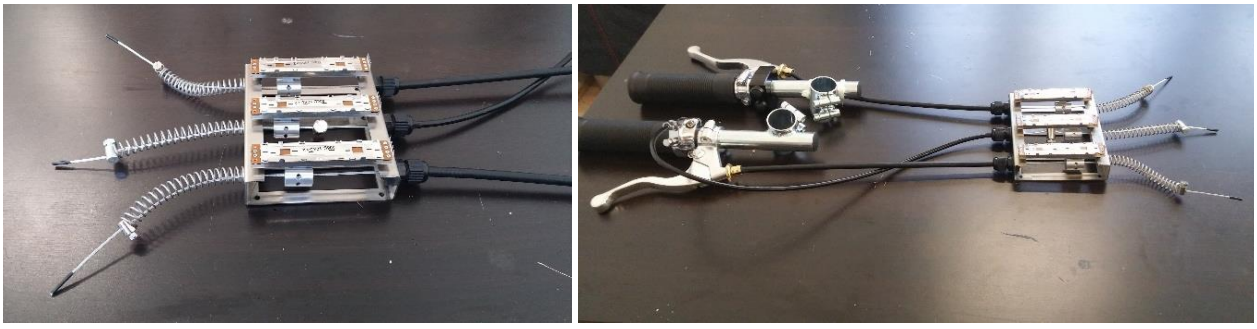


Figura 2.22: Fotografie dei sensori e del loro collegamento ai dispositivi di comando

Per l'implementazione del software di controllo della Consolle, si è utilizzata una scheda di sviluppo di costruzione *STMicroelectronics* basata su un microcontrollore *ARM Cortex M4F* da 32bit. Nella prossima sezione si andrà a presentare tale scheda con le sue funzionalità e le principali periferiche utilizzate.

2.2.2 Piattaforma MCU utilizzata

Il microcontrollore o MCU (*Micro Controller Unit*) è un dispositivo elettronico integrato su un singolo chip, chiamato per rispondere a requisiti quali:

- gestione di ingressi / uscite digitali;
- comunicazione con periferiche esterne;
- controllo di motori ed attuatori;
- gestione di programmi logico sequenziali;
- reazioni ad eventi;
- acquisizione dati da sensori.

È progettato per interagire direttamente con il mondo esterno tramite un programma residente nella propria memoria interna e mediante l'uso di *pin* specializzati o configurabili dal programmatore.
Nel caso della Consolle, considerando l'aspetto prototipale dell'applicazione si è scelto un dispositivo che avesse in dotazione, strumenti di sviluppo evoluti, in modo da facilitare la messa a punto ed il testing del software. Siccome la realizzazione del software è stata condotta con Matlab/Simulink, la scelta è ricaduta su un dispositivo il cui ambiente di sviluppo fosse facilmente integrabile con questo programma. A tal proposito, si è deciso di utilizzare la scheda *STM32F4 - Discovery* della *STMicroelectronics*.
La scheda STM32F4 include un microcontrollore (MCU) STM32F407VGT6 in un package LQFP, 100 pin multifunzione ed un debugger/programmer ST-LINK on board, che consente la programmazione della scheda tramite USB. Essa è provvista di una memoria flash da 1MB, 192 KB di RAM. È progettata per essere alimentata tramite porta USB, oppure da una sorgente di alimentazione esterna a 5V.
Il circuito della board mette a disposizione i *pin* del MCU su degli header facilmente accessibili, permettendo il collegamento I/O con altre *Board* compatibili o l'installazione della scheda su una *millefori* standard.
Volendo riassumere molto sinteticamente le caratteristiche salienti di un sistema di questo tipo, esse potrebbero essere così elencate:

- Potenza di calcolo in tempo reale (tipo e velocità del processore, memoria disponibile, sistema operativo)
- Disponibilità di una scheda a basso costo
- Funzioni di I/O analogico (frequenza di campionamento, numero di bit, range di tensione, velocità d'immagazzinamento dati)

- Funzioni di I/O digitale (frequenza di aggiornamento, possibilità di gestire protocolli di comunicazione, generazione pattern PWM o simili per il controllo di attuatori, interpretazione di segnali in ingresso, possibilità di gestire interrupt e di generare segnali su tale base).
- Possibilità di utilizzare Matlab/Simulink come Software di programmazione.

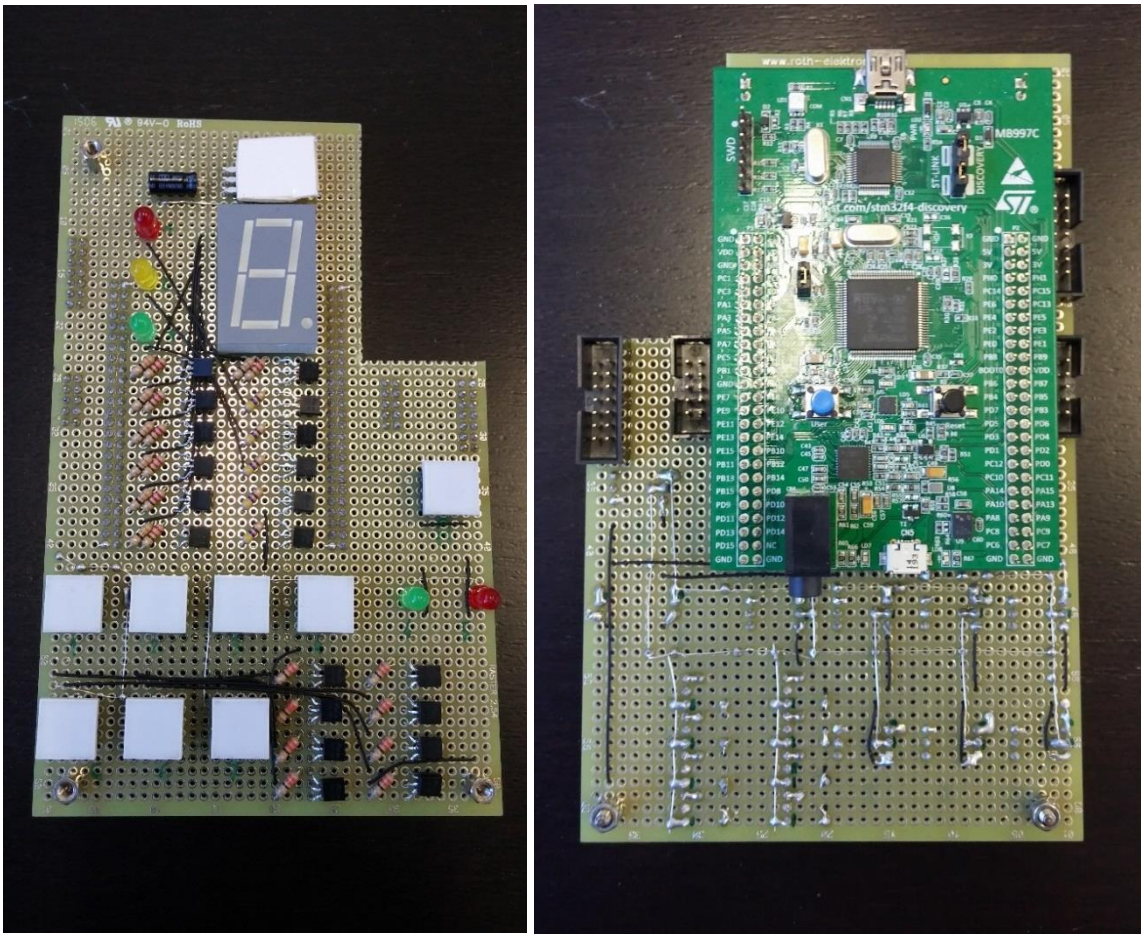


Figura 2.23: Fotografia della STM32F4 Discovery board e montaggio componenti su basetta millefori

Nel progetto sviluppato, le principali periferiche utilizzate della scheda STm32 sono:

- **GPIO**: la periferica GPIO è una delle più importanti periferiche del micro in quanto consente di configurare gli input/output della scheda (per il pinout relativo agli I/O della scheda STm32 si veda l'appendice A).
- **ADC**: la periferica ADC svolge la funzione di conversione di segnali analogici in segnali digitali interpretabili dal microcontrollore. Tale periferica viene usata per la lettura dei potenziometri. La scheda in questione presenta tre ADC a 12 bit da 2,4 Msps o 7,2 Msps in modalità interlacciata.
- **Ethernet**: tale periferica è utilizzata per lo scambio diretto di dati della Consolle con il ComapactRIO. La STm32 non è provvista di un connettore RJ45, ma è predisposta al collegamento di un modulo esterno dotato di una porta Ethernet, in modo da poterla connetterla alla rete locale (LAN). Il modulo Ethernet in questione e riportato nella seguente figura. Esso integra un chip che svolge la funzione di *transceiver Ethernet* ed è supportato dallo stesso ambiente di sviluppo impiegato nella programmazione STm32.

Features:

1. Texas Instrument DP83848C Ethernet Physical Layer Transceiver
2. Selectable MII (Default) / RMII (optional - need additional 50MHz clock source) Interface
3. Plug and Play with aMG F4Connect 2
4. Matlab / Simulink support with Waijung Blockset
5. RJ45 with Integrated Magnetics and status LED
6. Power LED

Figura 2.24: Modulo Ethernet

2.3 Stati del sistema di controllo

Arrivati a questo punto della trattazione è doveroso spiegare in cosa consistono gli **stati** del sistema già citati in più di un'occasione nel corso delle precedenti sezioni.
In sostanza, la logica di controllo del sistema RideIT si suddivide in sei differenti stati che rappresentano differenti modalità operative e dunque anche di gestire il motoveicolo durante un test.

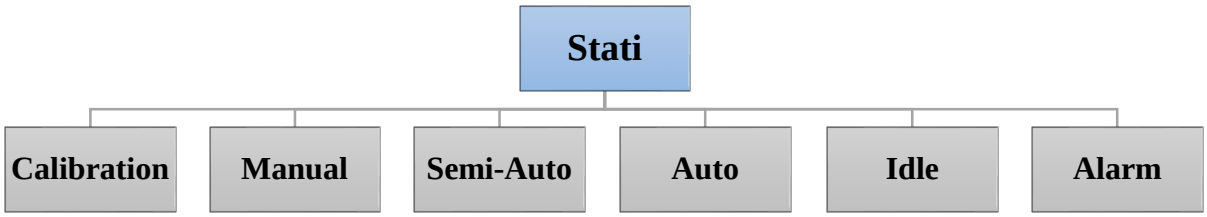


Figura 2.25: Stati del sistema di controllo

In base allo stato in cui il sistema si trova alcuni controlli delle interfacce utente saranno abilitati o meno. Nelle prossime sezioni saranno passati in rassegna, uno ad uno, i vari stati mettendo in particolare evidenza quali tipi di comandi sono abilitati e quali no per ciascuna delle interfacce del sistema.

2.3.1 Calibration

È lo stato in cui si possono eseguire le operazioni di calibrazione e regolazione delle corse degli attuatori a moto rigorosamente spenta. Dalla Consolle è possibile agire solo sulle leve di freno e frizione per movimentare i corrispondenti attuatori. In particolare, nella Consolle è anche possibile effettuare la calibrazione dei potenziometri che rivelano le posizioni e seguono gli spostamenti delle leve manovrate dall'utente quali: la manopola acceleratore e leve di freno e frizione. Da interfaccia Host è possibile agire, anche sulla movimentazione dell'attuatore del cambio, oltre che sugli altri due, tramite gli sliders. In tutti gli altri stati invece, il cambio viene gestito in maniera automatica.

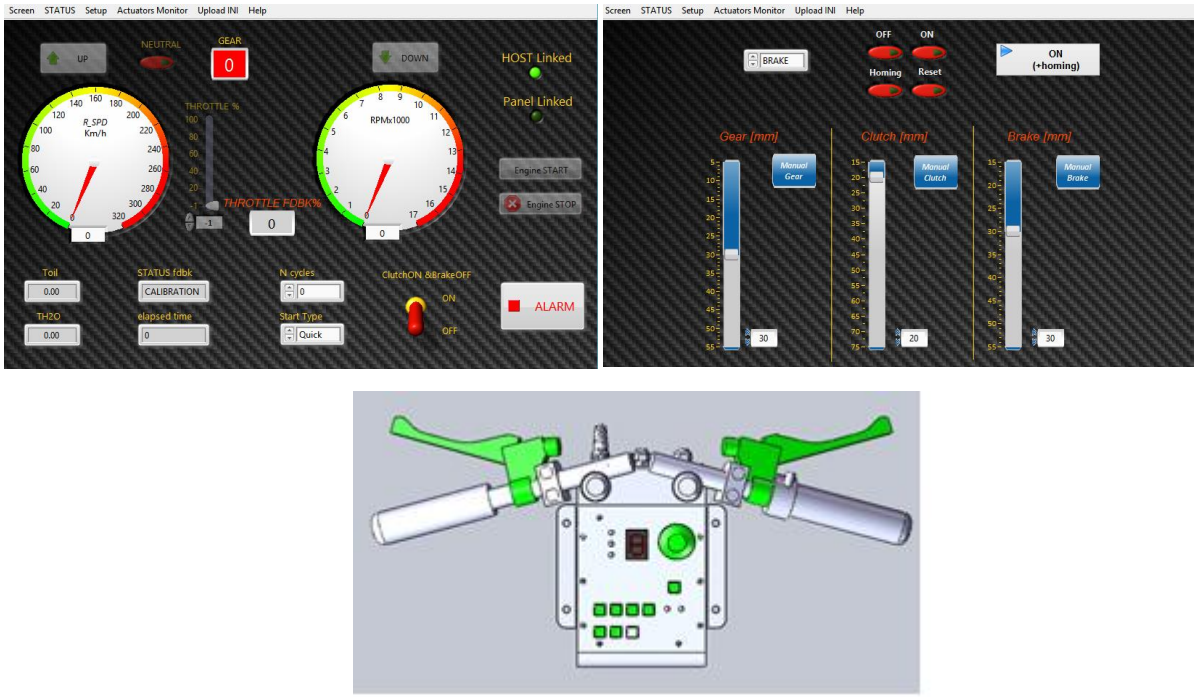


Figura 2.26: Screen delle interfacce di controllo, con evidenziati i comandi resi attivi in stato di *CALIBRATION*

Dalla figura 2.26 si può notare come alcuni dei comandi sull'Host siano oscurati rispetto ad altri. Ciò è stabilito dal software che abilita solo i comandi che possono essere selezionati dall'utente in quel determinato stato.

2.3.2 Manual

In *Manual*, sono abilitati tutti i comandi per condurre la guida in modalità manuale, pertanto è possibile replicare da remoto le stesse manovre che effettuerebbe un pilota in sella alla moto all'interno della cella, fatta eccezione per l'attuazione del cambio, movimentato in base alle richieste di *Upshift /Downshift* e *Neutral*, formulate dall'utente tramite gli appositi pulsanti sulle interfacce.

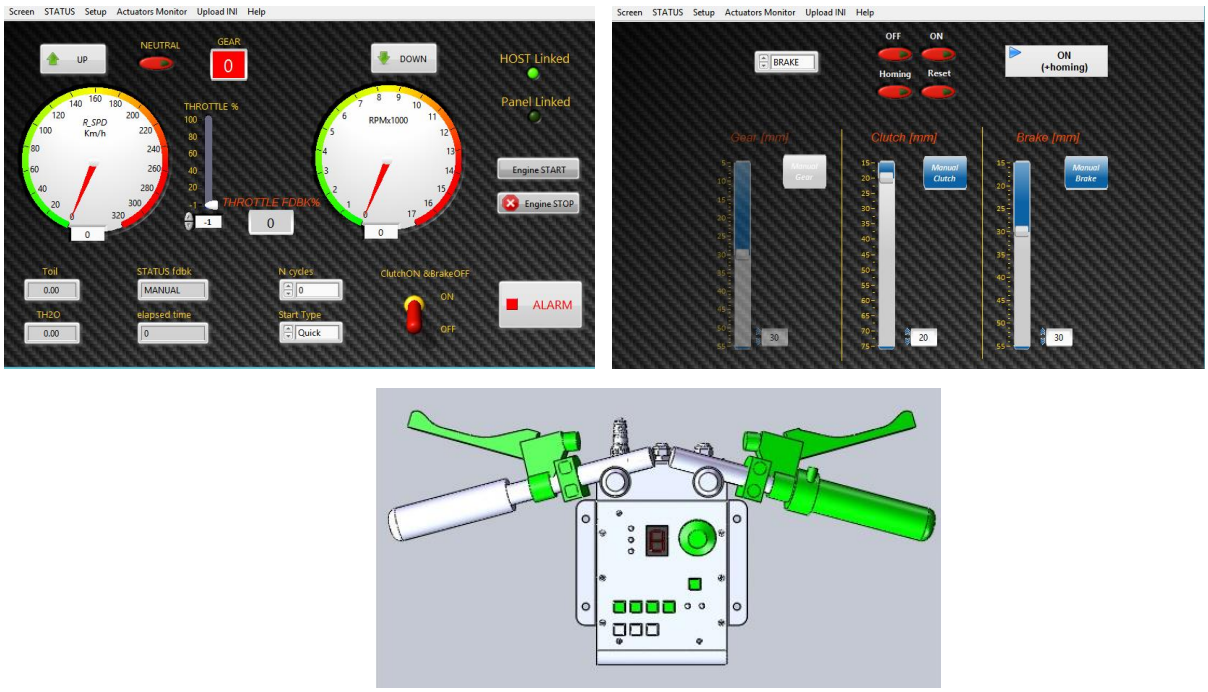


Figura 2.27: Screen delle interfacce di controllo, con evidenziati i comandi resi attivi in stato di *MANUAL*

2.3.3 Semi-Auto

In questo stato, la cambiata è gestita in modalità semiautomatica, ovvero il software interviene sugli attuatori di cambio e frizione in base alle condizioni di giri motore e richiesta acceleratore. Tuttavia è possibile richiedere una cambiata anche manualmente, agendo sugli appositi pulsanti che rimangono attivi in questo stato, così come gli altri comandi della guida manuale, ad eccezione della leva della frizione che viene disabilitata, in quanto viene gestita totalmente in automatico.

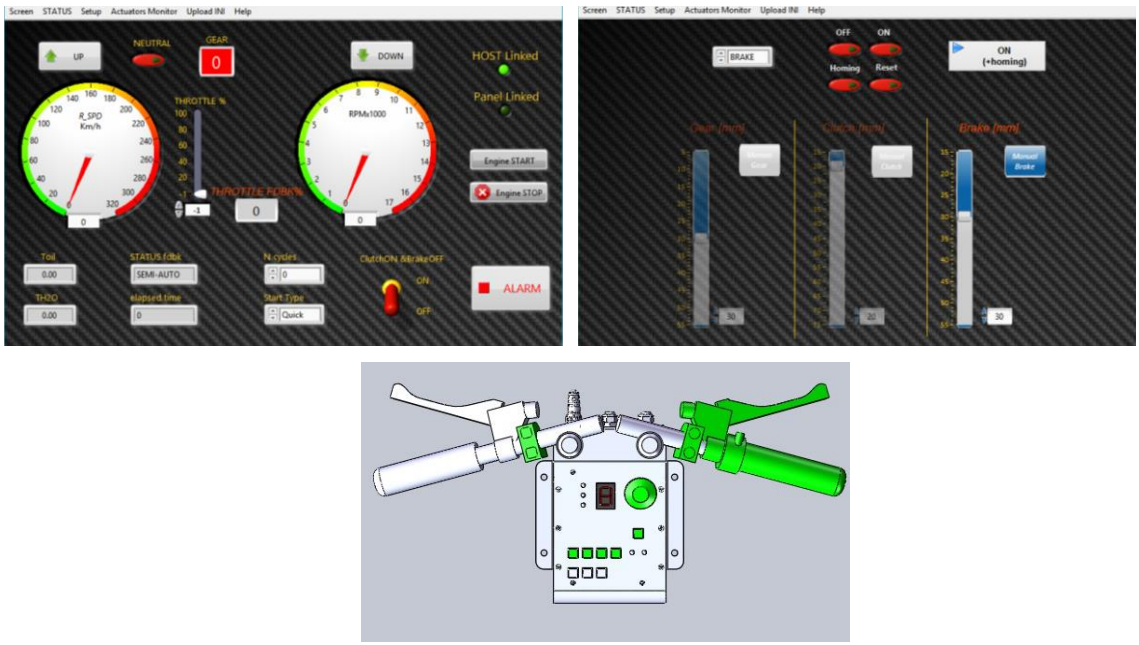


Figura 2.28: Screen delle interfacce di controllo, con evidenziati i comandi resi attivi in stato di *SEMI-AUTO*

2.3.4 Auto

Stato nel quale è possibile far replicare alla moto un ciclo di guida totalmente in automatico, inseguendo paramenti target riportati in un file Excel, precedentemente caricato sul sistema.

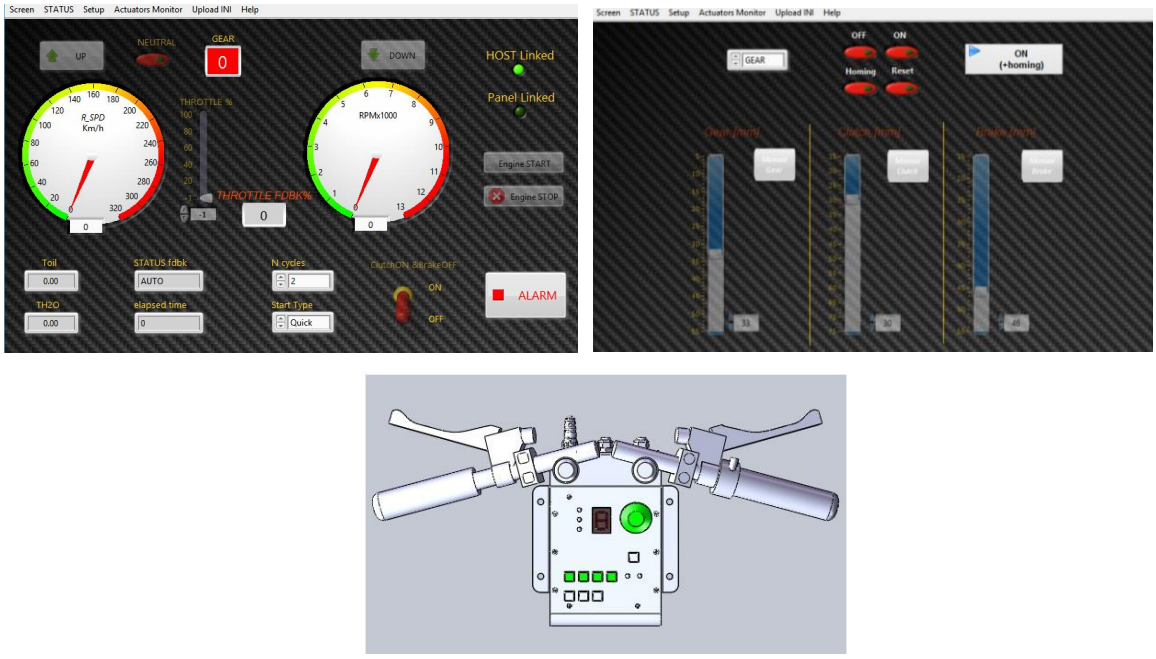


Figura 2.29: Screen delle interfacce di controllo, con evidenziati i comandi resi attivi in stato di *AUTO*

2.3.5 Alarm

È lo stato di Allarme che può essere richiamato automaticamente dal sistema, qualora venga rilevato un errore (errore sugli attuatori o disconoscimento della marcia inserita da parte della centralina), oppure richiamato dall'utente, ogni qualvolta egli imponga uno *STOP* d'urgenza tramite gli appositi pulsanti su Consolle o interfaccia Host. Il passaggio allo stato di *Alarm* comporta la messa in sicurezza automatica del banco e della moto, ovvero determina una chiusura repentina dell'acceleratore con conseguente azionamento della frizione, rilascio del freno e spegnimento del motore. Come si osserva in figura 2.30 tutti i comandi delle interfacce utente sono disabilitati.

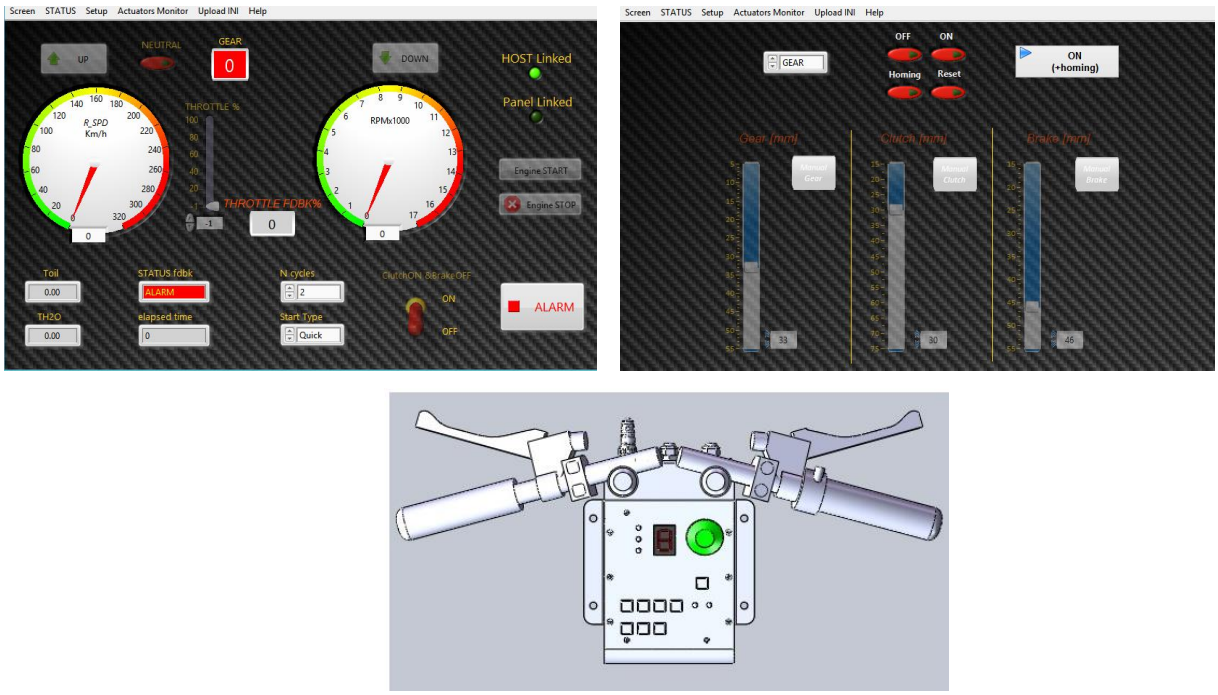


Figura 2.30: Screen delle interfacce di controllo, con evidenziati i comandi resi attivi in stato di *ALARM*

2.3.6 Idle

Stato in cui il sistema si porta automaticamente quando, a partire dallo stato di *Alarm*, il banco e la moto sono effettivamente in condizioni ritenute sicure, oppure a termine del ciclo di guida eseguito in automatico. In *Idle* la moto rimane con la frizione tirata, freno rilasciato e acceleratore chiuso, nessuna manovra è consentita all'utente, salvo quella di selezionare un nuovo stato in cui portare il sistema.

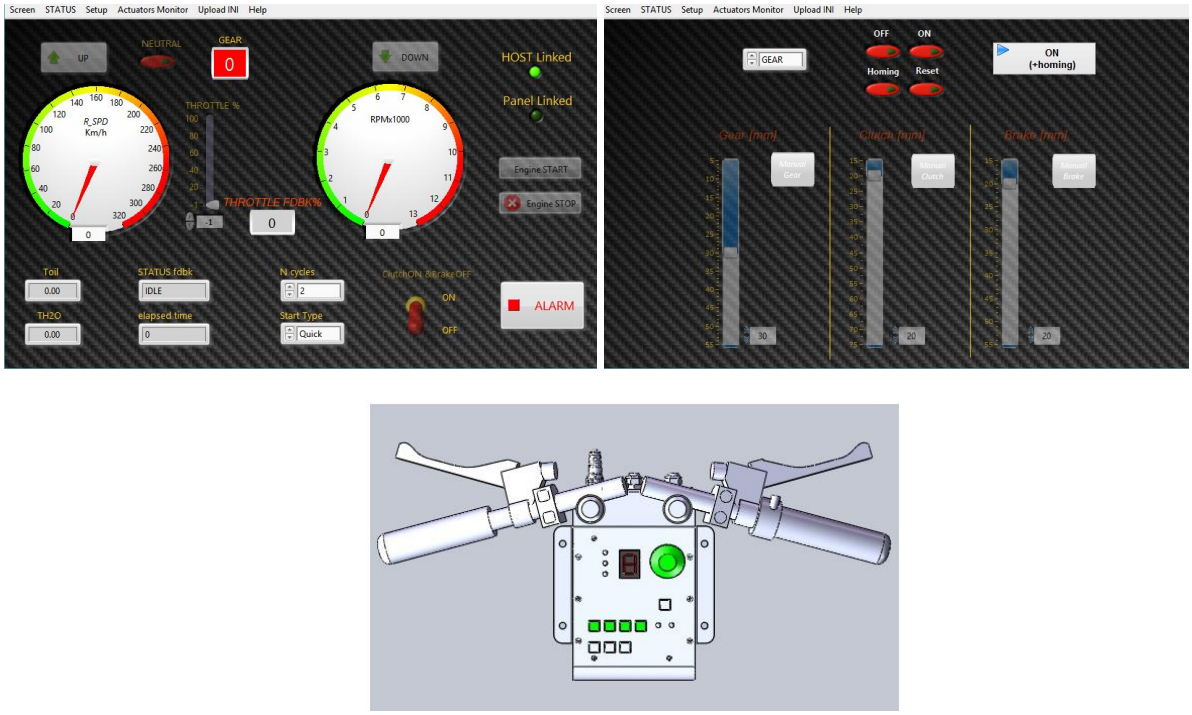


Figura 2.32: Screen delle interfacce di controllo, con evidenziati i comandi resi attivi in stato di *IDLE*

Capitolo 3

Software di controllo della Consolle

In questo capitolo della tesi si descrivono a fondo gli schemi a blocchi dell'ambiente Simulink che costituiscono il modello di controllo della Consolle.
In particolare con Simulink, si è utilizzata una toolchain di sviluppo per eseguire modelli su piattaforme target specifiche, come la STm32 nel caso in questione, denominata *Waijung Blockset* e che è scaricabile gratuitamente dal sito *aimagin.com*. La toolchain consente fondamentalmente di aggiungere strumenti alle librerie di Simulink, tali da permettere l'accesso e la configurabilità delle funzionalità hardware della scheda STm32, oltre a consentire di fare la compilazione e l'upload del file contenente il programma nella memoria flash dell'STm32, che poi lo esegue.
Si fa presente che, buona parte delle strutture logiche presenti nel modello e che verranno descritte in questo capitolo, sono state sviluppate nel corso dei test che hanno consentito di validare il funzionamento del dispositivo.

3.1 Il modello Simulink della Consolle

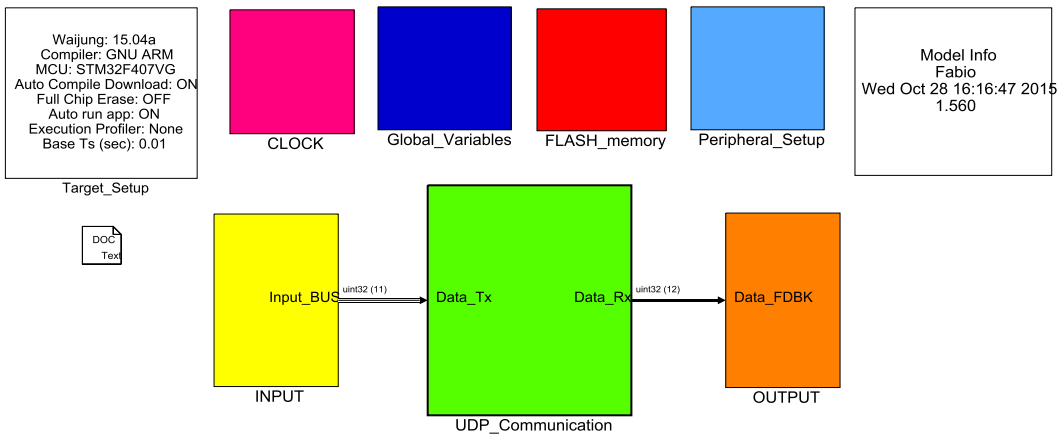


Figure 3.1: RideIT_Consolle_Model

Il modello, come riporta la figura 3.1, è suddiviso nelle sue parti funzionali. Nella parte riservata agli **INPUT**, si acquisiscono i segnali provenienti dai comandi di cui è munito il dispositivo e che vengono utilizzati dall'utente.
La parte costituita dal blocco **UDP_Communication**, contiene le istruzioni dedicate alla lettura/scrittura dei segnali scambiati fra la Consolle e il target real-time, ovvero il CompactRIO. Dentro al blocco inoltre si stabilisce se la connessione è stabilita in maniera corretta.
La parte degli **OUTPUT** è dedicata alla gestione e al controllo di tutti gli indicatori a LED presenti sulla consolle che informano l'utente circa il funzionamento del sistema.
In particolare, a questo livello gerarchico del modello si trova il blocco **Target_Setup** che è il blocco fondamentale in cui si impostano i parametri per la generazione del codice eseguibile a partire dal modello Simulink per il target STm32 una volta lanciato il processo di compilazione.
I restanti blocchi verranno esaminati più in dettaglio nelle prossime sezioni.

3.2 Peripheral Setup

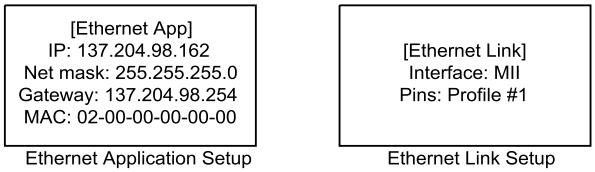


Figure 3.2: RideIT_Consolle_Model/Peripheral_Setup

In questo blocco sono contenuti i blocchi della libreria *Waijung* in cui si vanno a configurare i parametri per connettere in LAN la Consolle. In particolare, nel blocco *Ethernet Link* si è scelto un tipo di profilo (contrassegnato #1), che permette di utilizzare una certa sequenza di pin predefinita sulla scheda STm32 adatta al collegamento elettrico della *ethernet shield* scelta per l'applicazione. In alternativa occorreva configurare manualmente uno alla volta, i pin da utilizzare per il collegamento dell'interfaccia ethernet.

Il blocco *Ethernet Application Setup* contiene i parametri di configurazione dell'interfaccia di rete, ovvero l'*indirizzo IP* della Consolle, il *Subnet Mask*, il *Gateway*, il *MAC address* e il *TCP timer*.

3.3 CLOCK

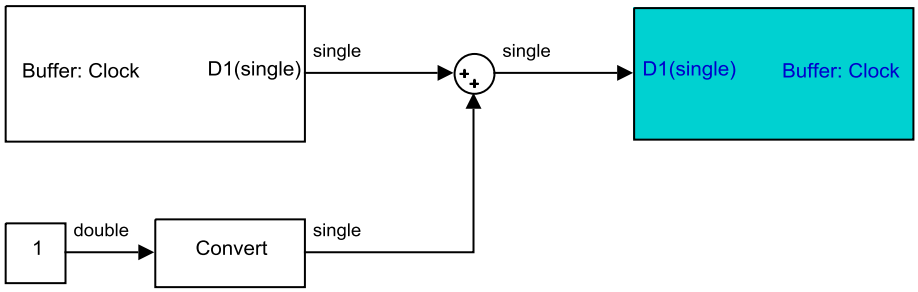


Figure 3.3: RideIT_Consolle_Model/CLOCK

Il blocco *CLOCK* produce il segnale di tempo della simulazione a partire da quando la consolle viene alimentata e quindi il modello inizia a girare. Esso incrementa di un'unità ad ogni passo di calcolo, che risulta impostato pari a 0.01 sec (100Hz). Durante il periodo di campionamento vengono mantenuti i valori della simulazione fino al successivo istante di campionamento. Il segnale di *CLOCK* viene richiamato in altri sottoblocchi all'interno del modello che necessitano del tempo si simulazione grazie alla cella di memoria.

3.4 Global Variables

Nel sottosistema sono raggruppati i blocchi di memoria archivio dati. In ciascuno di questi blocchi viene definito il nome, il formato e la condizione iniziale di una variabile che deve essere utilizzata nel modello e viene richiamata dai blocchi *Volatile Data Store Read* e *Volatile Data Store Write* che fanno parte della libreria Waijung.

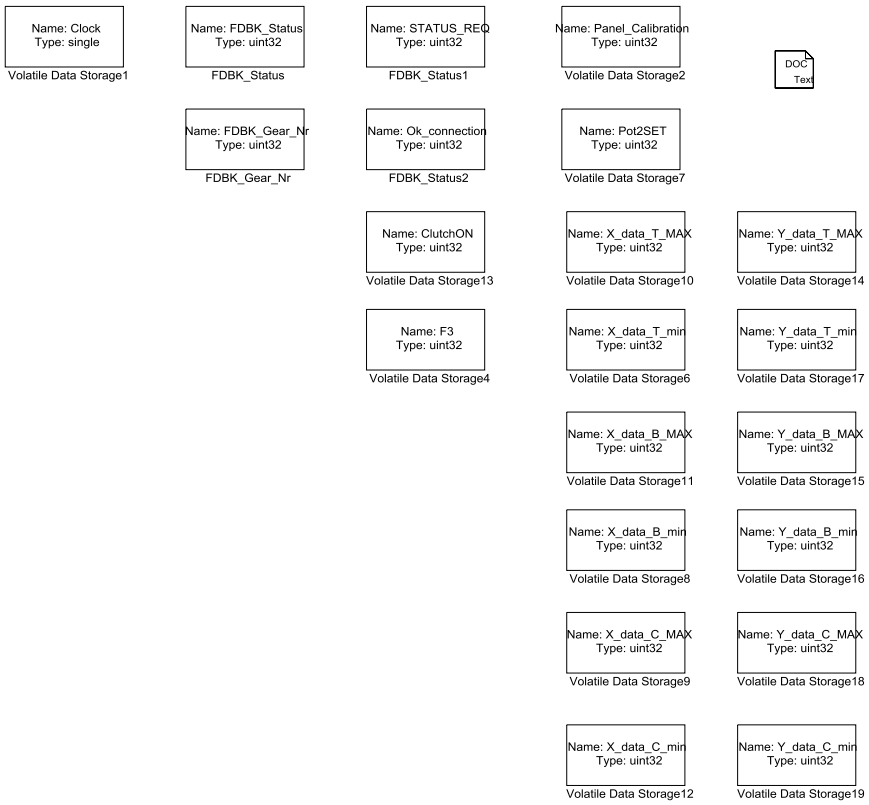


Figure 3.4: RideIT_Consolle_Model/Global_Variables

3.5 Flash Memory

Contiene i blocchi che consentono di impiegare la memoria flash (non volatile) interna del microcontrollore, allo scopo di conservare nel tempo i dati riguardanti la calibrazione dei potenziometri della consolle anche quando essa non viene alimentata, evitando dunque di dover ripetere la procedura di calibrazione ogni volta che si riavvia il dispositivo.

Le operazioni effettuate sulla memoria flash sono implementate nei seguenti sottosistemi:

- ***WriteParCal:*** permette la memorizzazione dei dati registrati in fase di calibrazione.
- ***ReadParCal:*** permette il recupero dei parametri memorizzati nella fase successiva alla calibrazione e all'avviamento del dispositivo.

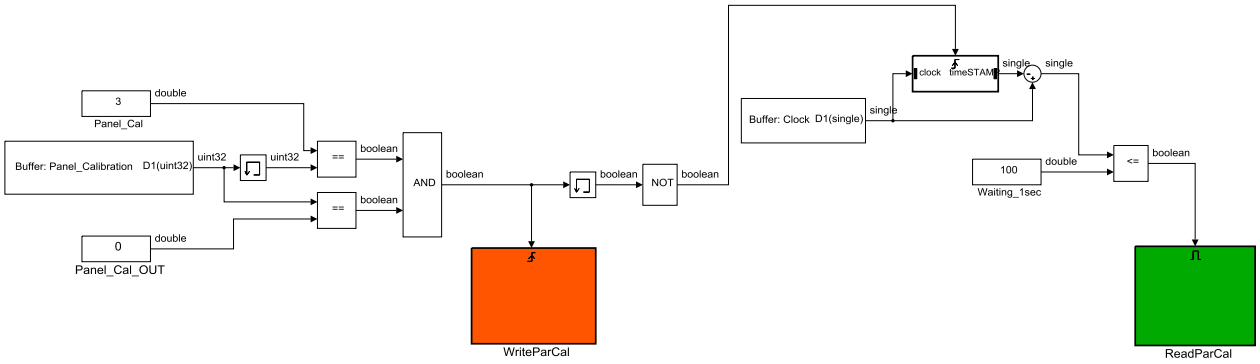


Figure 3.5: RideIT_Consolle_Model/Flash_Memory

3.5.1 Scrittura dati di calibrazione sulla memoria flash

Il sottoblocco *WriteParCal* viene eseguito una sola volta al momento in cui si è raggiunto lo stato "Panel_Cal_OUT" al termine della procedura di calibrazione.

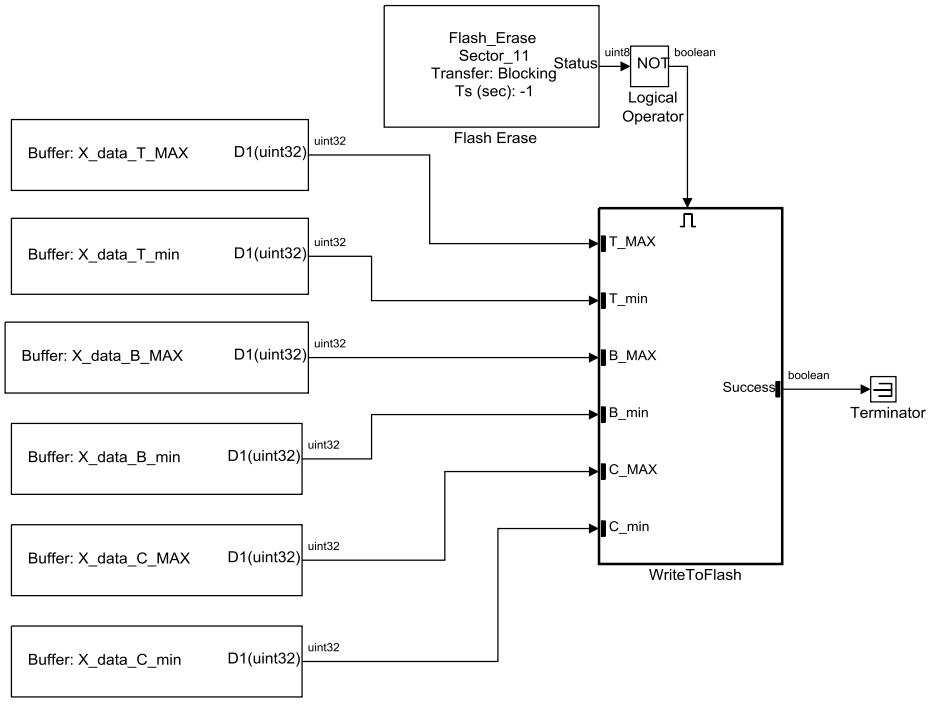


Figure 3.6: RideIT_Consolle_Model/Flash_Memory/WriteParCal

Il processo di scrittura consiste essenzialmente nella seguente sequenza di operazioni:

- All'istante in cui si è verificato il trigger viene eseguita la cancellazione dell'intero settore di memoria flash impiegato per la memorizzazione dei dati di calibrazione;
- Al completamento della cancellazione viene eseguito il blocco *WriteToFlash*, pertanto vengono inviati i nuovi dati di calibrazione verso gli indirizzi di memoria definiti per quel dato settore.
- Se la scrittura viene eseguita con successo allora il segnale "Success" risulta pari a true. Questo segnale non viene utilizzato per un qualche scopo all'interno del modello, viene perciò semplicemente terminato.

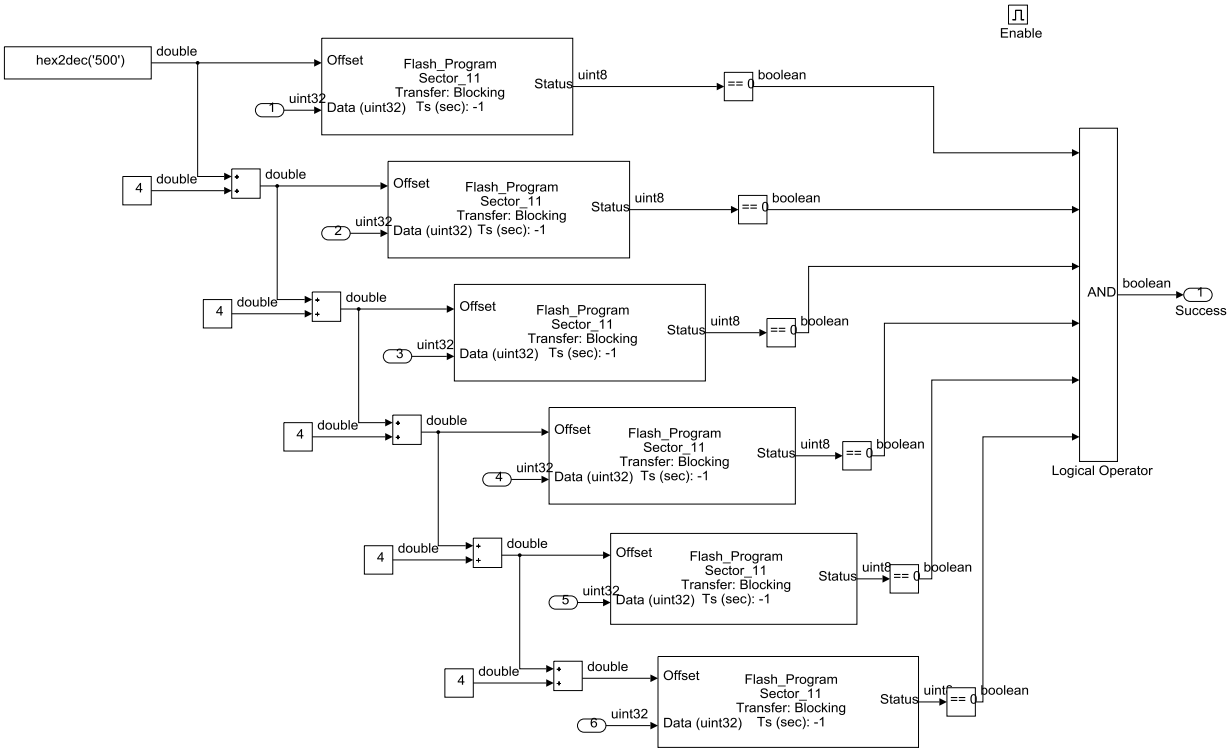


Figure 3.7: RideIT_Consolle_Model/FLASH_memory/WriteParCal/WriteToFlash

3.5.2 Lettura dati di calibrazione dalla memoria flash

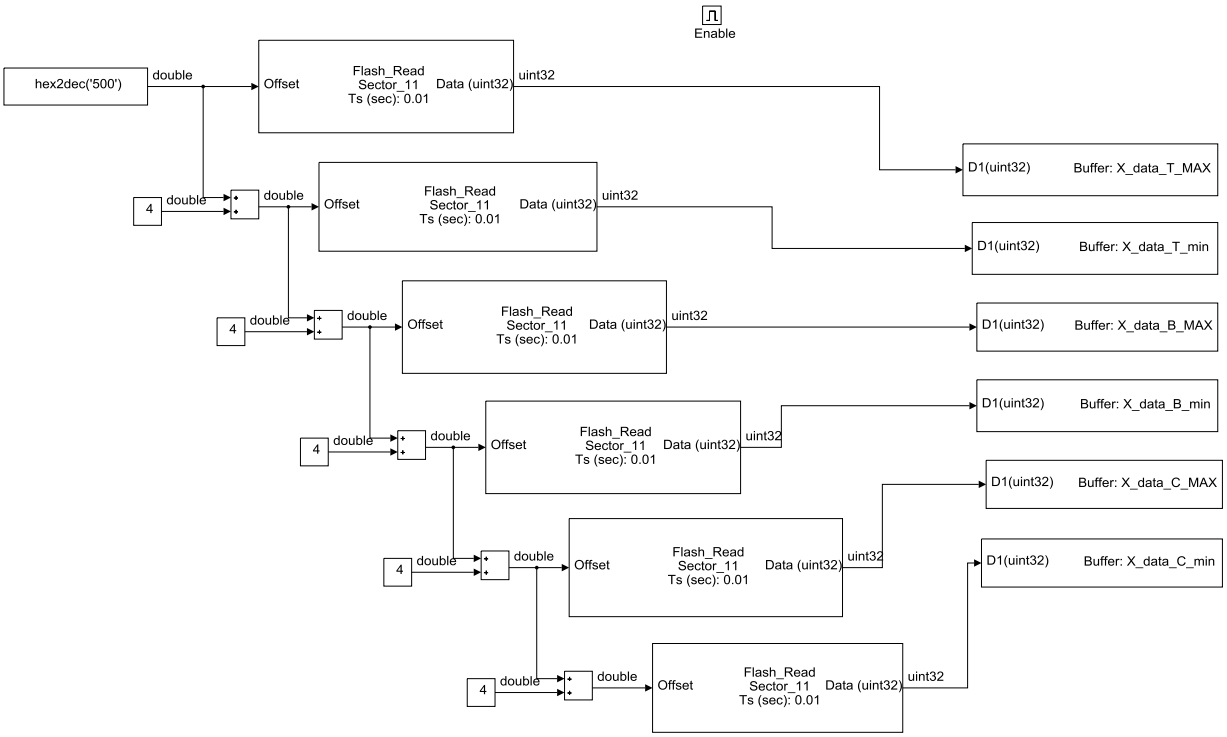


Figure 3.8: RideIT_Consolle_Model/FLASH_memory/ReadParCal

Il sottoblocco *ReadParCal* viene eseguito al passo di calcolo successivo rispetto a quello in cui si è verificato il trigger sul sottoblocco *WriteParCal* al termine della procedura di calibrazione.

Il processo di lettura consiste essenzialmente nel richiamare all'interno del modello i dati dalle celle di memoria flash aventi gli indirizzi uguali a quelli adottati nel processo di scrittura.

Per assicurare che non ci sia una mancata lettura dei valori dalle suddette celle, è stata creata una condizione di "enable" prolungata a un secondo sul sottoblocco *ReadParCa*l. Tale condizione si verifica anche quando il Clock si azzera, pertanto ogni qual volta la consolle viene avviata, nel primo secondo dopo l'accensione si esegue la lettura dagli indirizzi di memoria flash in cui risiedono i dati di calibrazione.

3.6 INPUT

Nel sottosistema INPUT viene creato un array di variabili, di formato *uint32*, che rappresentano le richieste dell'utente e devono essere trasmesse al CompactRIO via UDP.

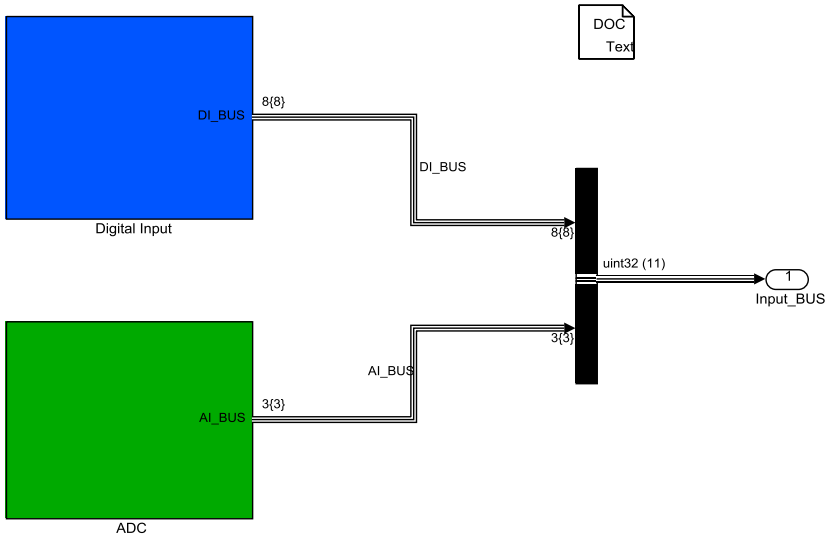


Figure 3.9: RideIT_Consolle_Model/INPUT

3.6.1 Digital Input

Nel sottosistema *Digital Input* si leggono i valori dei segnali digitali provenienti dai pulsanti della consolle e li si raggruppa in un vettore in modo da trasferirli più comodamente agli altri sottosistemi del modello.
I vari sottosistemi presenti all'interno del macro blocco *Digital Input*, riguardano la gestione e il controllo di ciascuna delle tipologie di pulsante presenti nella Consolle e sono suddivisi a seconda delle funzioni che devono adempiere. Vediamoli nel dettaglio:

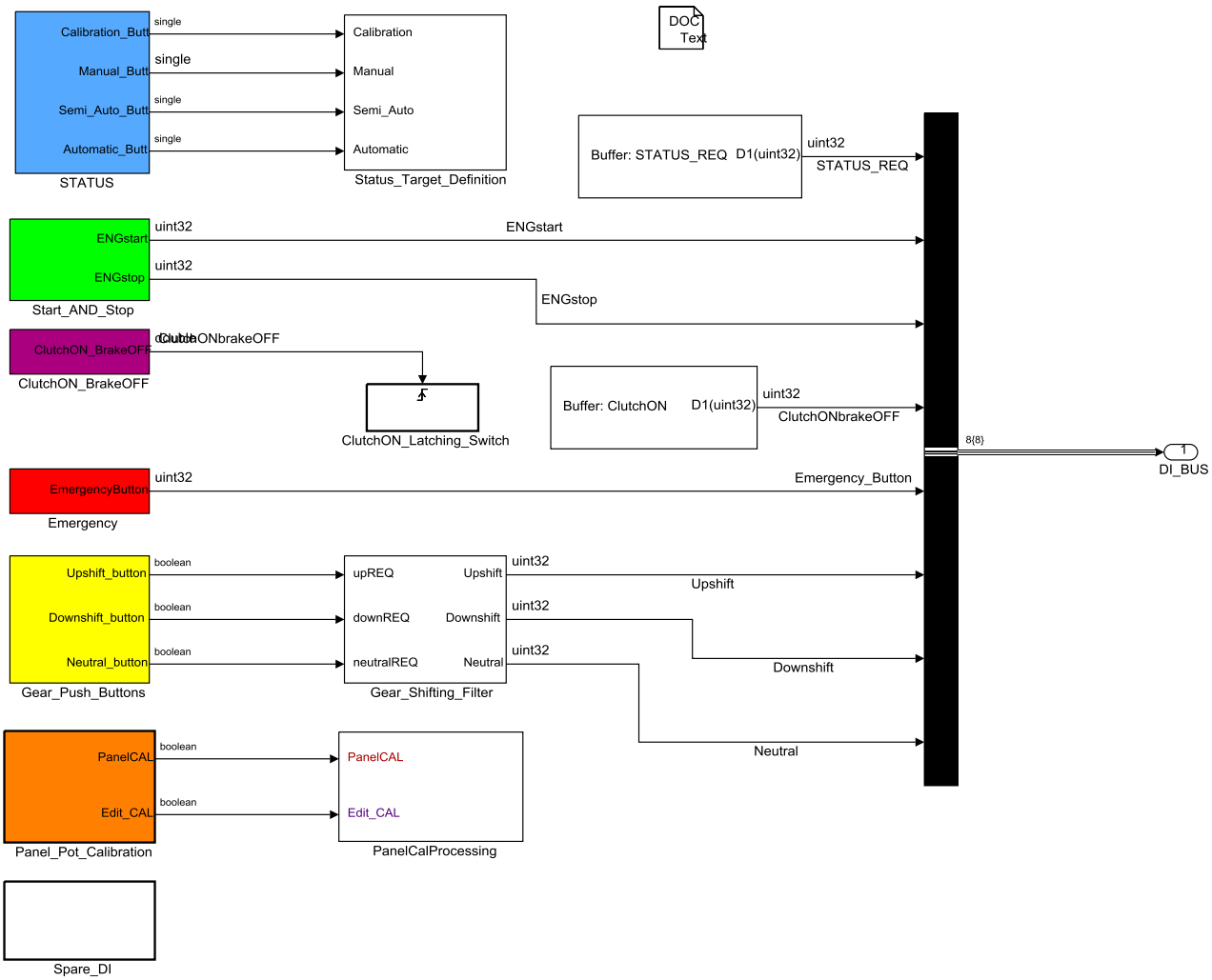


Figure 3.10: RideIT_Consolle_Model/INPUT/Digital Input

3.6.1.1 Pulsanti di STATUS

Il sottosistema **STATUS** permette di abilitare il microcontrollore a leggere i segnali provenienti dai pulsanti di selezione degli stati CALIBRATION, MANUAL, SEMI-AUTO e AUTOMATIC nel caso in cui siano verificate insieme le seguenti condizioni:

- Lo “Status_FDBK” è diverso da ALARM, oppure da AUTOMATIC.
- La consolle è connessa alla rete.

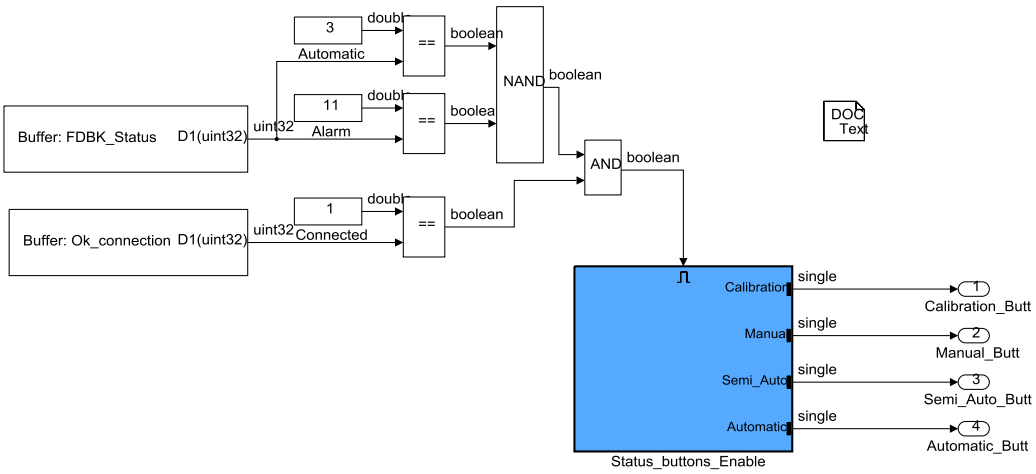


Figure 3.11: RideIT_Consolle_Model/INPUT/Digital Input/STATUS

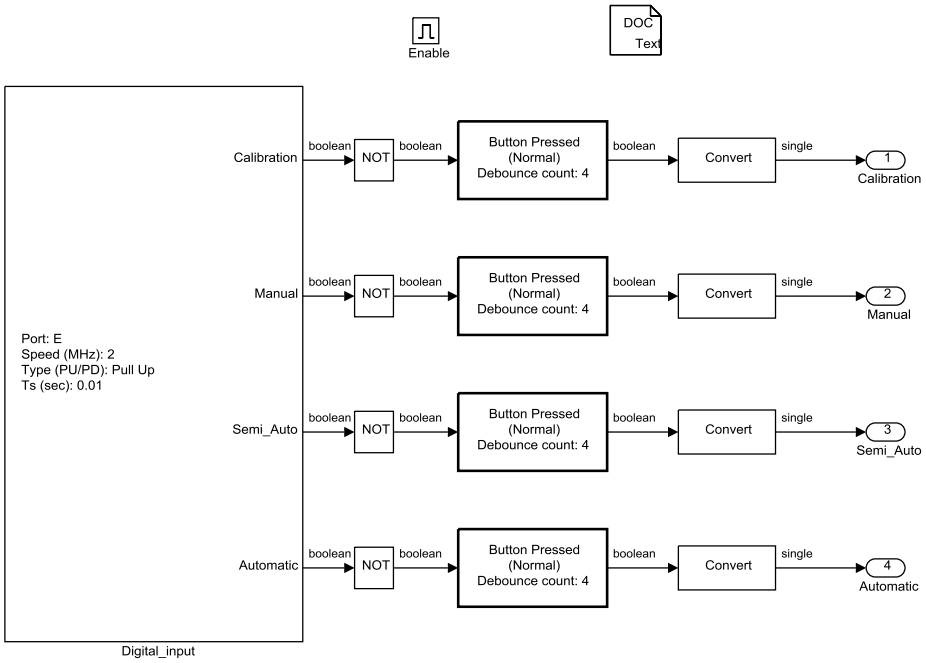


Figure 3.12: RideIT_Consolle_Model/INPUT/Digital Input/STATUS/Status_buttons_Enable

Status_buttons_Enable: il sottosistema imposta la funzione di lettura per gli ingressi digitali sui pin di programmazione della STm32 connessi ai pulsanti di selezione degli stati e restituisce i valori booleani prodotti dalle acquisizioni. In questo, è possibile

ricevere uno stato di tensione "HIGH" /"1", compreso tra i 2.0V e 3.3V, e uno stato "LOW"/ "0" fra 0.8 V e gli 0 Volt (TTL Technology).
I pin degli ingressi digitali sono configurati nel modello, attraverso il blocchetto dei DI (digital input) fornito dalla libreria Waijung. Per ogni *pin* riservato alla funzione di DI si è deciso di porre una resistenza interna di *pull-up*. La circuiteria che descrive il funzionamento di ciò che è stato implementato nella consolle per tutti i segnali digitali di input, è illustrata in figura 3.13. Un capo del pulsante è collegato alla massa (GND), l'altro capo arriva sia al pin della STm32, sia ad una resistenza collegata all'alimentazione positiva V_{DD} (3.3V); con il pulsante a riposo la lettura è "HIGH", poiché è presente la resistenza collegata all'alimentazione. Premendo il pulsante, il pin si viene a trovare a potenziale 0V, e si avrà di conseguenza la lettura LOW.

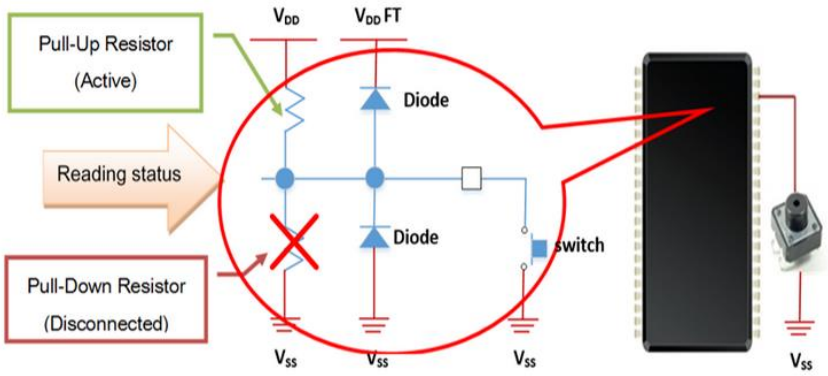


Figure 3.13: Pull-up circuit

La funzionalità della resistenza è duplice: se si rimuovesse del tutto si avrebbe una lettura instabile nelle condizioni a riposo, poiché un pin della STm32, lasciato scollegato, presenta un valore casuale dovuto a dei rumori di fondo e non un valore "LOW" come si potrebbe erroneamente pensare. Se invece si effettuasse un collegamento diretto a V_{DD}, premendo il bottone si avrebbe un cortocircuito. Per questo la resistenza deve avere un valore alto, tale da non provocare inutili correnti, ovvero dell'ordine dei 10kΩ.
Il chip integrato della STm32, per ogni pin di I/O dispone una resistenza integrata collegata a +3.3V (pull-up), oppure di una resistenza integrata collegata a GND (pull-

down). Essa può essere collegata internamente (questa possibilità si concretizza mediante un interruttore "interno" al chip). L'abilitazione viene fatta nel modello sul blocco dei DI, impostando i pin di input come *pull-up* piuttosto che come *pull-down,* o lasciarli privi di collegamento. Con questa scelta, il circuito esterno della consolle si è semplificato notevolmente, non essendoci stata più necessità di inserimento della resistenza esterna, inoltre l'adozione di una resistenza di *pull-up* ha comportato un abbassamento della corrente che attraversa il circuito e ha garantito un miglior comportamento dei pin di input che risultano meno soggetti a fluttuazioni.
L' inconveniente ha riguardato la modalità di funzionamento dei pulsanti a "logica invertita" rispetto alla convenzione usuale, per questo motivo sono stati inseriti dei NOT su ciascun segnale DI, in modo da ottenere nel modello uno stato "HIGH" quando il circuito viene chiuso anziché nel caso contrario.
I blocchi *Debounce* svolgono la funzione di eliminare l'effetto dei "rimbalzi" presenti in ogni interruttore o pulsante meccanico. Infatti quando un interruttore viene chiuso, tale chiusura non avviene in genere istantaneamente. A causa della flessibilità elastica della lamina metallica interna dell'interruttore, la chiusura produce infatti una serie di micro-rimbalzi (in inglese *bounce*), cioè in pratica una rapida sequenza di stati aperto/chiuso in successione. La seguente figura mostra i tipici rimbalzi che possono essere visualizzati sullo schermo di un oscilloscopio:

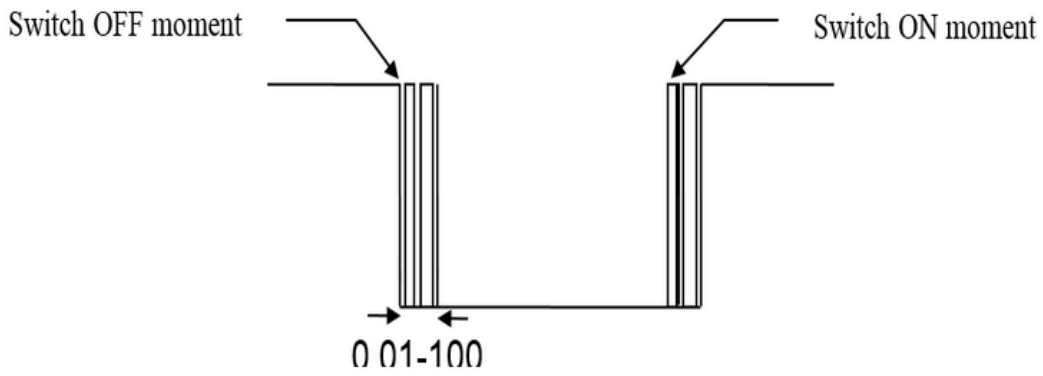


Figure 3.14: Bouncing effect

I rimbalzi non creano problemi se il circuito comandato dall'interruttore è un circuito "lento" (come ad esempio una lampadina), mentre possono essere molto dannosi nel caso di circuiti logici, in quanto i rimbalzi dell'interruttore potrebbero essere erroneamente interpretati dal circuito come stati logici alti e bassi validi.
Il blocco *Debounce* elimina i rimbalzi in quanto la commutazione dell'uscita avviene sulla prima chiusura (in pratica sul primo rimbalzo) dell'interruttore. A quel punto, per come è stato configurato il blocchetto, si ha che lo stato dell'uscita rimane memorizzato (stabile) per 4 step di calcolo e non varia anche in presenza di rimbalzi dell'ingresso.

3.6.1.2 Status Target Definition

Nel sottosistema vengono codificati quattro dei sei stati che l'utente può richiedere dalla consolle premendo gli appositi pulsanti.

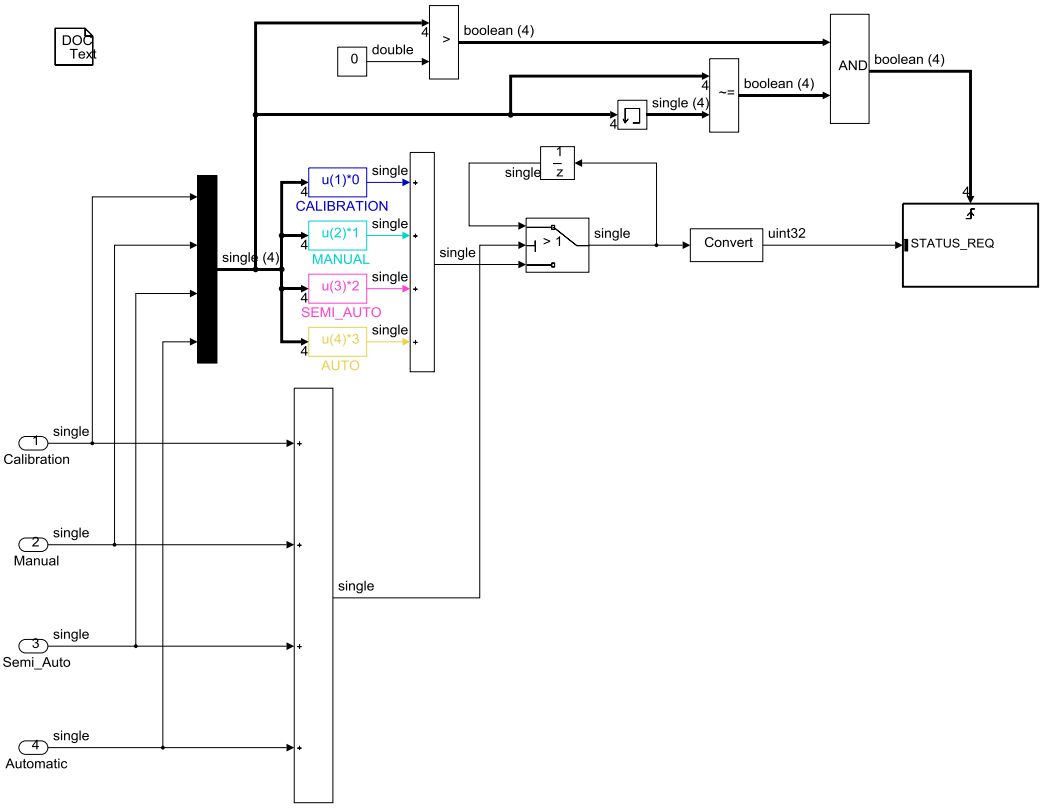


Figure 3.15: RideIT_Consolle_Model/INPUT/Digital Input/Status_Target_Definition

CALIBRATION	0
MANUAL	1
SEMI-AUTO	2
AUTOMATIC	3

Tabella 3.1: Nomenclatura Stati

La scrittura del segnale "STATUS_REQ" viene eseguita su una memoria racchiusa nel sotto blocco con trigger, eseguita su base evento. Questo è dato quando viene premuto uno dei pulsanti di selezione degli stati e dunque si verificano le condizioni in input al blocco AND, dal quale viene emesso il segnale booleano di trigger. Nel sottosistema è implementata inoltre una logica che consente di escludere la possibilità di richiedere uno stato e dunque modificare il segnale "STATUS_REQ" nel momento in cui due o più pulsanti vengono premuti contemporaneamente.

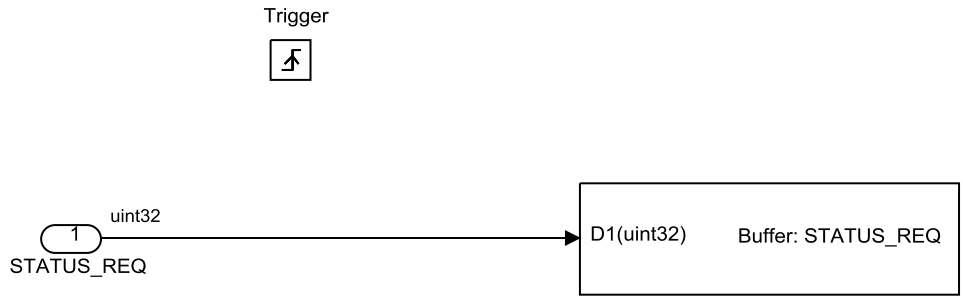


Figure 3.16: RideIT_Consolle_Model/INPUT/Digital Input/Status_Target_Definition/Subsystem

3.6.1.3 Pulsanti Start&Stop

Il sottosistema **Start_AND_Stop** permette di abilitare il microcontrollore alla lettura del segnale proveniente dal pulsante di "Engine_Start" nel caso in cui siano verificate contemporaneamente le seguenti condizioni:

- Lo "Status_FDBK" è impostato in MANUAL, oppure in SEMI_AUTO.
- La consolle è connessa alla rete.

Mentre viene abilitato il microcontrollore alla lettura del segnale proveniente dal pulsante di “Engine_Stop” nel caso in cui sono verificate contemporaneamente le seguenti condizioni:

- Lo “Status_FDBK” è posizionato in CALIBRATION, oppure in MANUAL, oppure in SEMI_AUTO
- La consolle è connessa alla rete

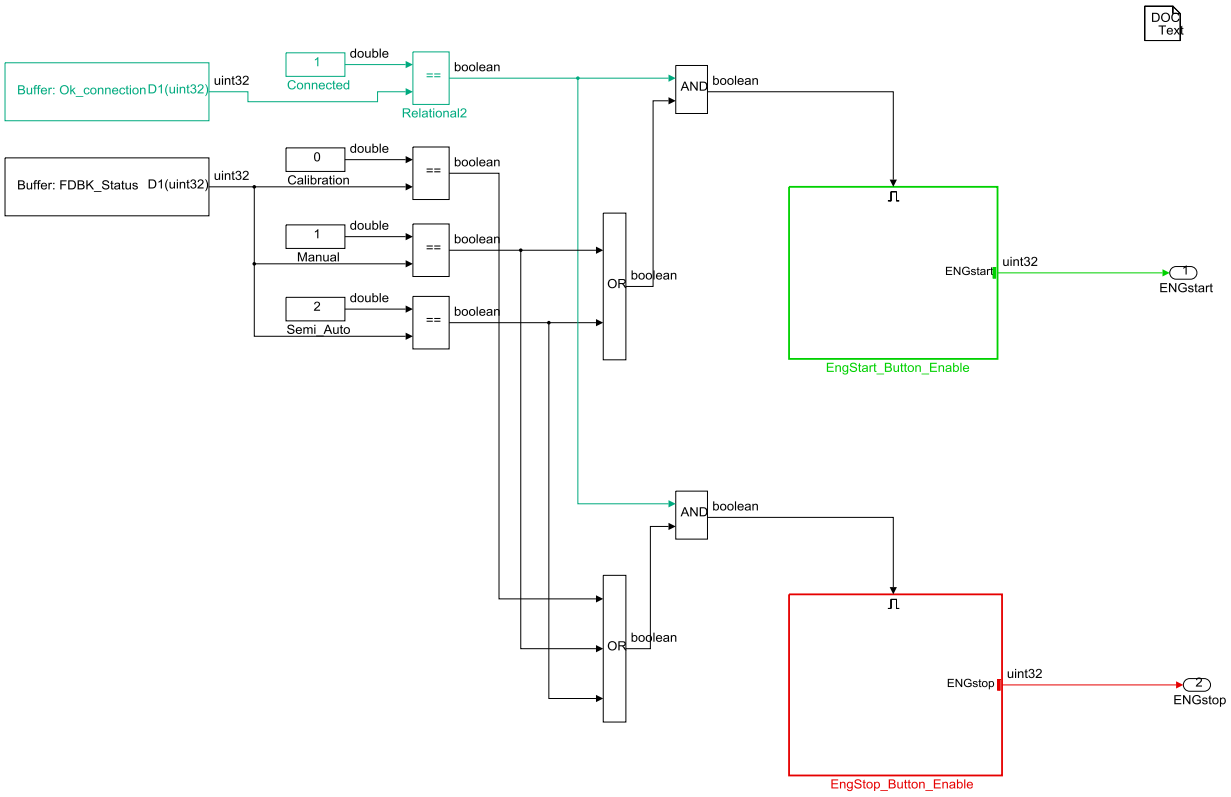


Figure 3.17: RideIT_Consolle_Model/INPUT/Digital_Input/Start_AND_Stop

EngStart_Button_Enable: Nel sottosistema, il blocco *digital input*, imposta la funzione di lettura del *pin* di programmazione scelto sulla STm32 e collegato al pulsante di “ENGINE_Start”, riservato alla funzione di accensione motore. Nel blocco si riportano le impostazioni adottate per la lettura di tale segnale e che sono relative al tipo di porta GPIO della scheda, alla frequenza di campionamento e al tipo di resistenza interna. L’inserimento della funzione NOT nel modello, permette di invertire la convenzione

degli stati del pulsante e quindi di leggere lo stato “LOW” con il pulsante a riposo, “HIGH” quando è premuto.

Il blocco *Debounce* serve a contrastare il comportamento non ideale di interruttori meccanici che possono creare più transizioni elettriche quando viene immesso un singolo input da parte dell'utente.

Infine il blocco *convert* viene inserito per cambiare il formato della variabile da boolean a uint32.

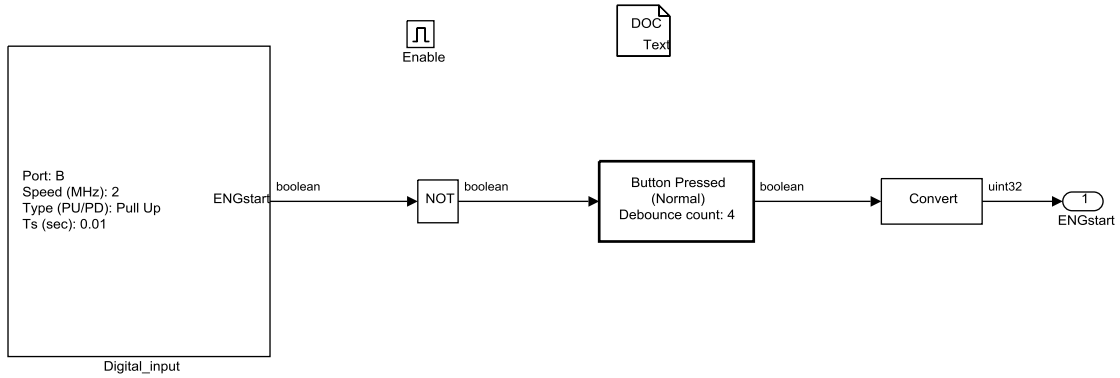


Figure 3.18:
RideIT_Consolle_Model/INPUT/Digital_Input/Start_AND_Stop/EngStart_Button_Enable

EngStop_Button_Enable: Identiche considerazioni a quelle viste per il comando “ENG_Start”, valgono per il comando “ENG_Stop” come si può osservare nella seguente figura:

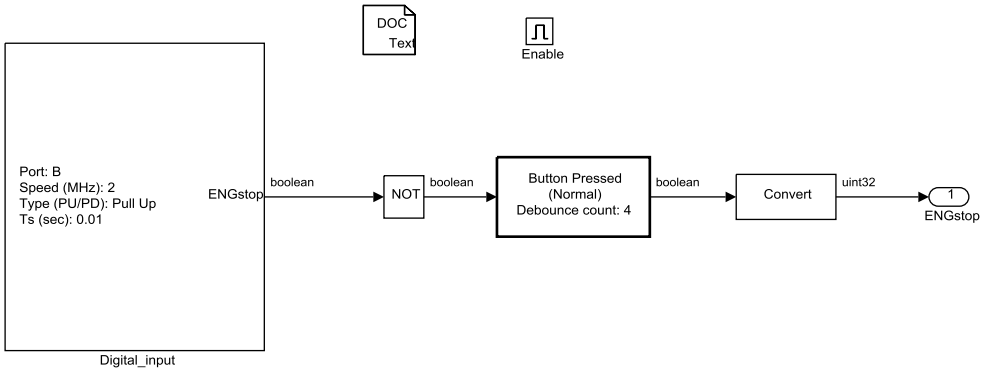


Figure 3.19:
RideIT_Consolle_Model/INPUT/Digital_Input/Start_AND_Stop/EngStop_Button_Enable

3.6.1.4 ClutchON&BrakeOFF

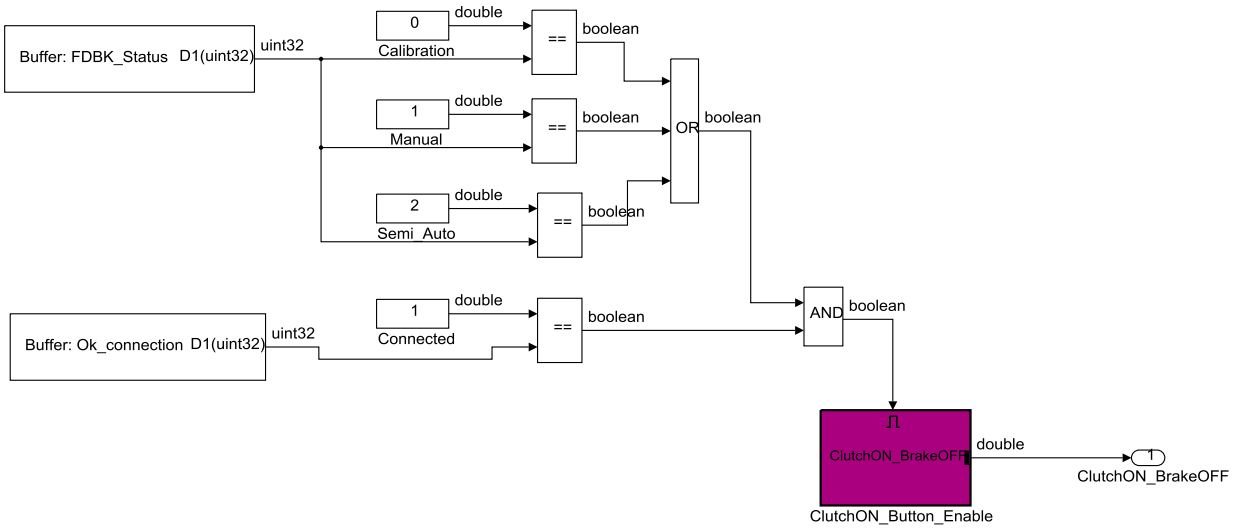


Figure 3.20: RideIT_Consolle_Model/INPUT/Digital_Input/ClutchON_BrakeOFF

Il sottosistema permette di abilitare il microcontrollore alla lettura del segnale proveniente dal pulsante di "ClutchON&BrakeOFF" nel caso in cui siano verificate contemporaneamente le seguenti condizioni:

- Lo "Status_FDBK" è impostato in CALIBRATION, oppure in MANUAL, oppure in SEMI_AUTO
- La consolle è connessa alla rete

ClutchON_Button_Enable: Il blocco *digital_input* in questo sottosistema, imposta la funzione di lettura del *pin* di programmazione scelto sulla STm32 per il collegamento elettrico del pulsante di "ClutchON_BrakeOFF". Il pin di ingresso digitale è configurato con resistenza interna di tipo *pull-up*. Proprio in ragione di questo, viene inserito un NOT, che inverte la logica di funzionamento, permettendo quindi di ottenere nel modello un segnale booleano pari a "HIGH" /"1", o "LOW"/ "0" a seconda del fatto che il pulsante sia rispettivamente premuto o meno.

Il blocco con la funzione di "debounce" (anti-rimbalzo) viene inserito per contrastare il comportamento non ideale di interruttori meccanici che possono creare più transizioni elettriche quando viene immesso un singolo input da parte dell'utente.

Infine il blocco *convert* viene inserito per cambiare il formato della variabile sulla base del formato richiesto da altri blocchi che vengono posti a valle e che eseguono operazioni su quel segnale.

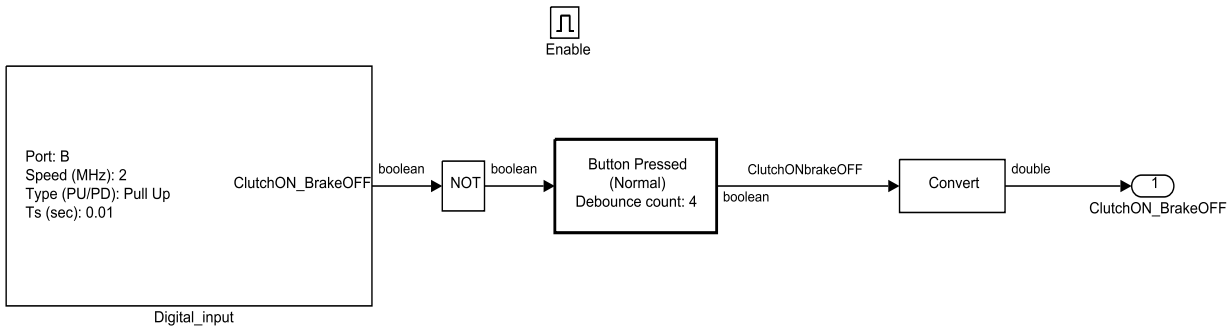


Figure 3.21: RideIT_Consolle_Model/INPUT/Digital_Input/ClutchON&BrakeOFF/ClutchON_Button_Enable

ClutchON_Latching_Switch: in questo sottosistema è stata implementata una logica per modificare il comportamento di un pulsante momentaneo, come quello adottato sulla consolle per il comando "ClutchON_BrakeOFF", in un pulsante di tipo latch. Normalmente quando viene premuto un pulsante momentaneo, il segnale rimane attivo alto fino a quando viene mantenuta la pressione sull'interruttore, mentre ritorna allo stato logico zero quando il pulsante viene rilasciato. Invece grazie ad un circuito latch set-reset con trigger, come quello rappresentato in figura 3.22, si memorizza il bit di informazione relativo all'ultima volta che è stato premuto il pulsante, pertanto viene mantenuto il segnale attivo alto anche quando il pulsante viene rilasciato. Analogamente, per disattivare il segnale riportandolo a zero bisogna premere di nuovo il pulsante.
Il segnale di "ClutchON_BrakeOFF" risulta in questo modo caratterizzato da due stati logici "0" e "1", che possono essere mantenuti stabili nel tempo.

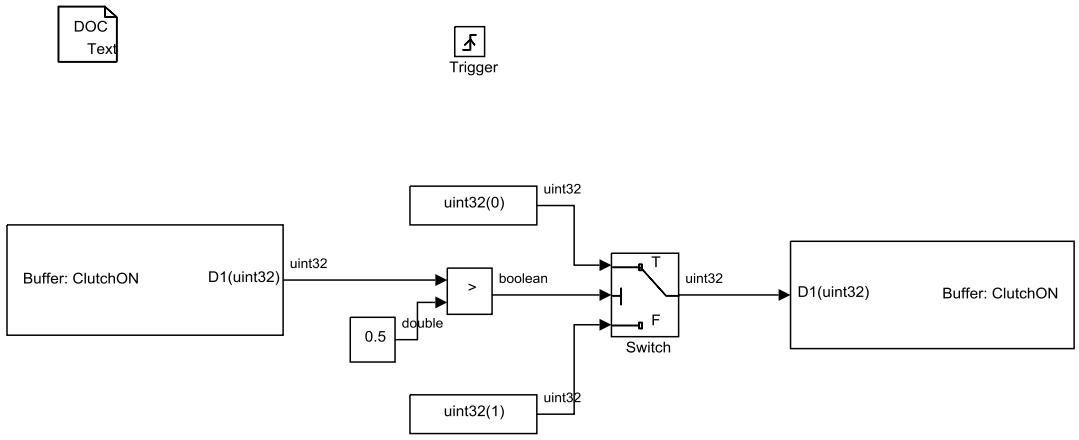


Figure 3.22: RideIT_Consolle_Model/INPUT/Digital Input/ClutchON_Latching_Switch

3.6.1.5 Pulsante di arresto in emergenza

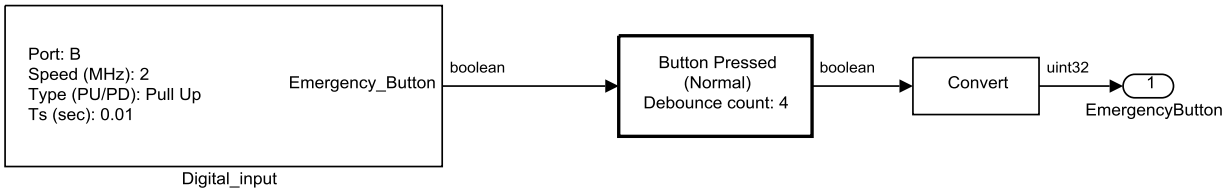


Figure 3.23: RideIT_Consolle_Model/INPUT/Digital Input/Emergency

All'interno del sottosistema **Emergency**, Il blocco "digital_input" imposta la funzione di lettura del *pin* di programmazione scelto sulla STm32 e collegato al pulsante di arresto di emergenza. Il pin di ingresso digitale è configurato con resistenza interna di tipo pull-up. Va notato che in questo caso non viene inserito un NOT per invertire la logica di funzionamento del DI, ed il motivo risiede nel fatto che il tipo di pulsante adottato per la funzione di arresto è NC (normalmente chiuso), ovvero il contatto è chiuso verso massa (*ground*) quando è in posizione di riposo, perciò sarà "0" il valore del segnale letto. Quando invece il pulsante viene premuto, allora il segnale passa a "1".
Il blocco con funzione di *debounce* viene inserito per contrastare il comportamento non ideale di interruttori meccanici che possono creare più transizioni elettriche quando viene immesso un singolo input da parte dell'utente.

Infine il blocco *convert* viene inserito per cambiare il formato della variabile in formato uint32.

3.6.1.6 Pulsanti della cambiata

Il sottosistema **Gear_Push_Buttons** permette di abilitare il microcontrollore alla lettura dei segnali provenienti dai pulsanti Up/Down e Neutral per effettuare la cambiata, nel caso in cui siano contemporaneamente verificate le seguenti condizioni:

- Lo "Status_FDBK" è impostato in MANUAL oppure in SEMI_AUTO
- La consolle è connessa alla rete

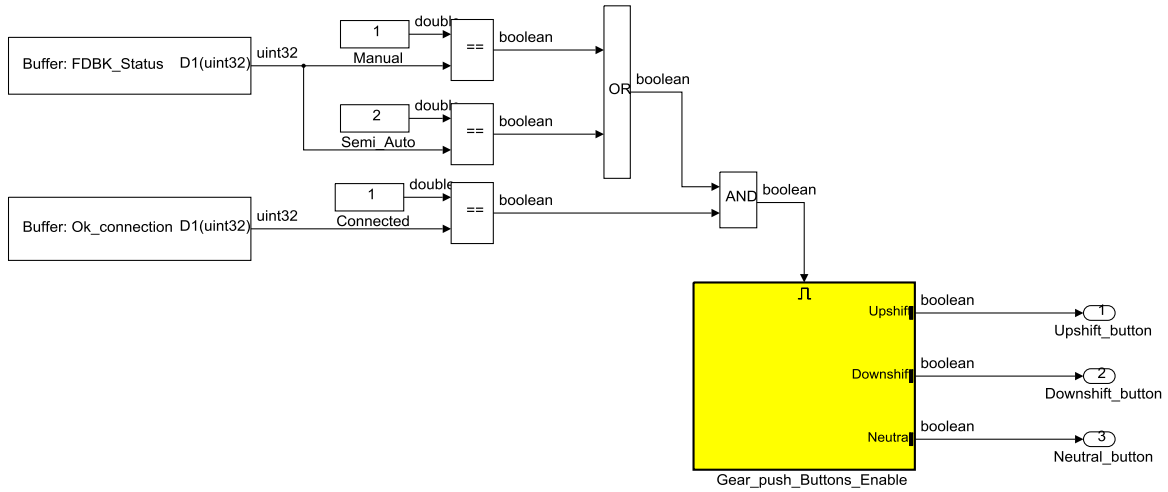


Figure 3.24: RideIT_Consolle_Model/INPUT/Digital Input/Gear_Push_Buttons

Gear_push_Buttons_Enable: I blocchi dei *digital_input* in questo sottosistema impostano le funzioni di lettura sui pin digitali della scheda STm32, impiegati per le acquisizioni dei segnali dei pulsanti del cambio.

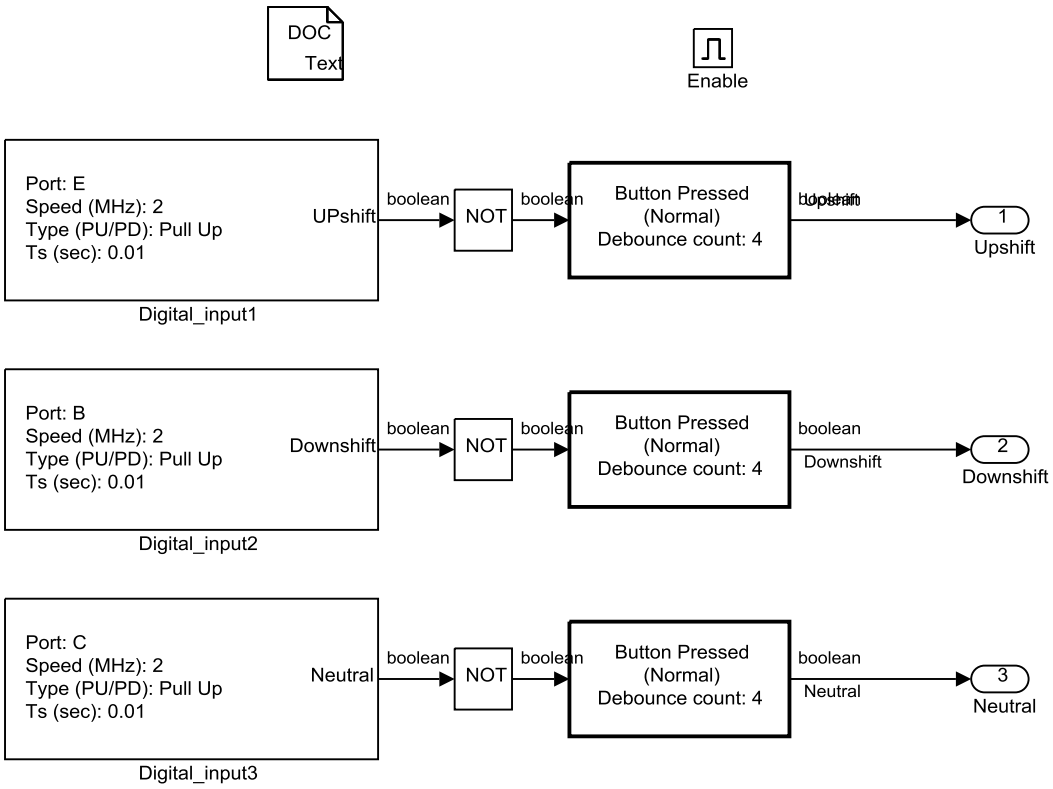


Figure 3.25: RideIT_Consolle_Model/INPUT/Digital Input/Gear_Push_Buttons/ Gear_push_Buttons_Enable

Gear_Shifting_Filter: Dentro a questo sottosistema si analizzano i segnali dei pulsanti con cui si gestiscono le cambiate sulla moto, in modo che dalla consolle possano essere impartite solo delle richieste di cambio marcia corrette. Guardando più in dettaglio, attraverso l'operatore logico XOR viene negata la possibilità di premere i due pulsanti di "Upshift e Downshift" contemporaneamente. Tale condizione viene posta a valle in AND con un'altra che abilita o meno la richiesta dell'utente ad effettuare un nuovo cambio di marcia. In quest'ultimo caso viene analizzata l'informazione relativa al numero di marcia inserito sulla moto, che è data dal segnale "FDBK_Gear_Nr", e sulla base di questa vengono rispettate le seguenti condizioni:

- Non è consentito richiedere una marcia superiore se quella inserita corrisponde ad una sesta (FDBK_Gear_Nr = 6);

- Non è consentito richiedere una scalata se la marcia inserita è una prima (FDBK_Gear_Nr = 1);
- È consentito richiedere una marcia tramite i pulsanti +/- nel caso in cui la marcia letta sul feedback corrisponde ad una folle (FDBK_Gear_Nr = 0);

La richiesta di inserire una folle, non viene in alcun modo condizionata nel sottosistema e di fatto viene semplicemente trascinata direttamente in output.
I blocchi *convert* modificano il formato delle variabili da booleane a uint32.

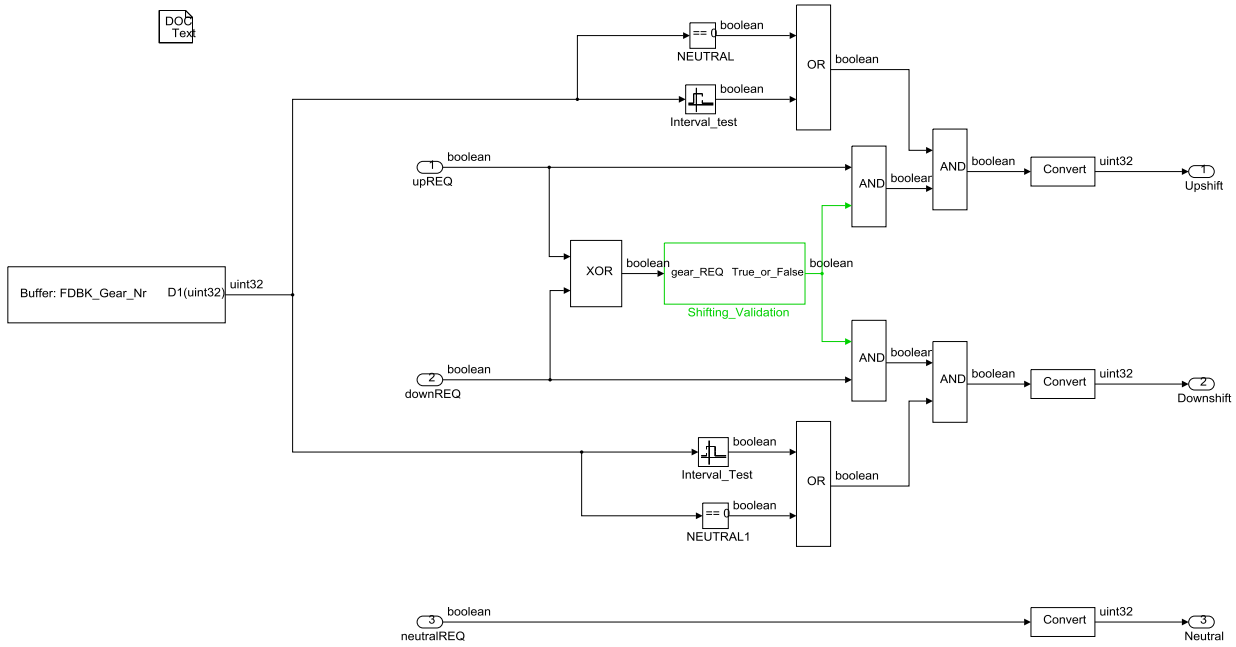


Figure 3.26: RideIT_Consolle_Model/INPUT/Digital Input/ Gear_Shifting_Filter

3.6.1.7 Calibrazione dei potenziometri

Per poter compensare le inevitabili discordanze tra il modello teorico e la consolle realmente costruita è stata definita una procedura di calibrazione che permette di acquisire i limiti dello spazio di lavoro dei comandi analogici e quindi calcolare i parametri necessari per far coincidere la corsa meccanica effettiva con la corsa teorica utilizzata per il comando. Nella sezione 3.6.2.1 vengono descritti gli schemi che

implementano la metodologia con cui vengono eseguite le letture e i calcoli dei parametri di calibrazione dei singoli potenziometri dei comandi della Consolle, mentre di seguito vengono descritti gli schemi che implementano la procedura di calibrazione in base a come questa può essere attivata/disattivata e di come devono essere gestiti i pulsanti specifici per eseguire determinate funzioni nell'ambito della procedura stessa.
Innanzi tutto partiamo dal sottosistema **Panel_Pot_Calibration** che permette di abilitare il microcontrollore alla lettura dei segnale provenienti dai pulsanti impiegati per eseguire la procedura di calibrazione dei potenziometri della consolle nel caso in cui lo stato del sistema, ovvero il segnale "FDBK_Status", sia impostato su CALIBRATION.

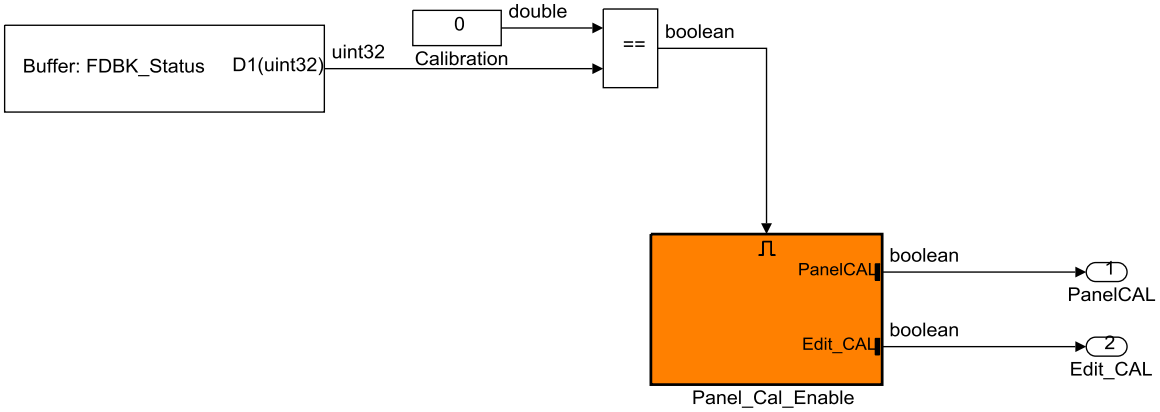


Figure 3.27: RideIT_Consolle_Model/INPUT/Digital Input/Panel_Pot_Calibration

PanelCalProcessing: Il sottosistema contiene le istruzioni per eseguire la calibrazione dei potenziometri della consolle. Come già è stato detto, questo tipo di procedura può essere eseguita solo quando il sistema è in stato CALIBRATION.
La logica di questa procedura segue una modellazione a "**macchina a stati**", del tutto simile a quella adottata per la gestione delle diverse modalità di guida della moto.
Si è definita la variabile "Panel_Calibration", per rappresentare l'evoluzione attraverso gli stati raggiungibili, durante la procedura di calibrazione. Gli stati sono nominati e contrassegnati da numeri interi che vanno da 0 a 4, come viene riportato nella figura 3.29, con particolare riferimento alla variabile "Panel_Calibration" durante le transizioni.

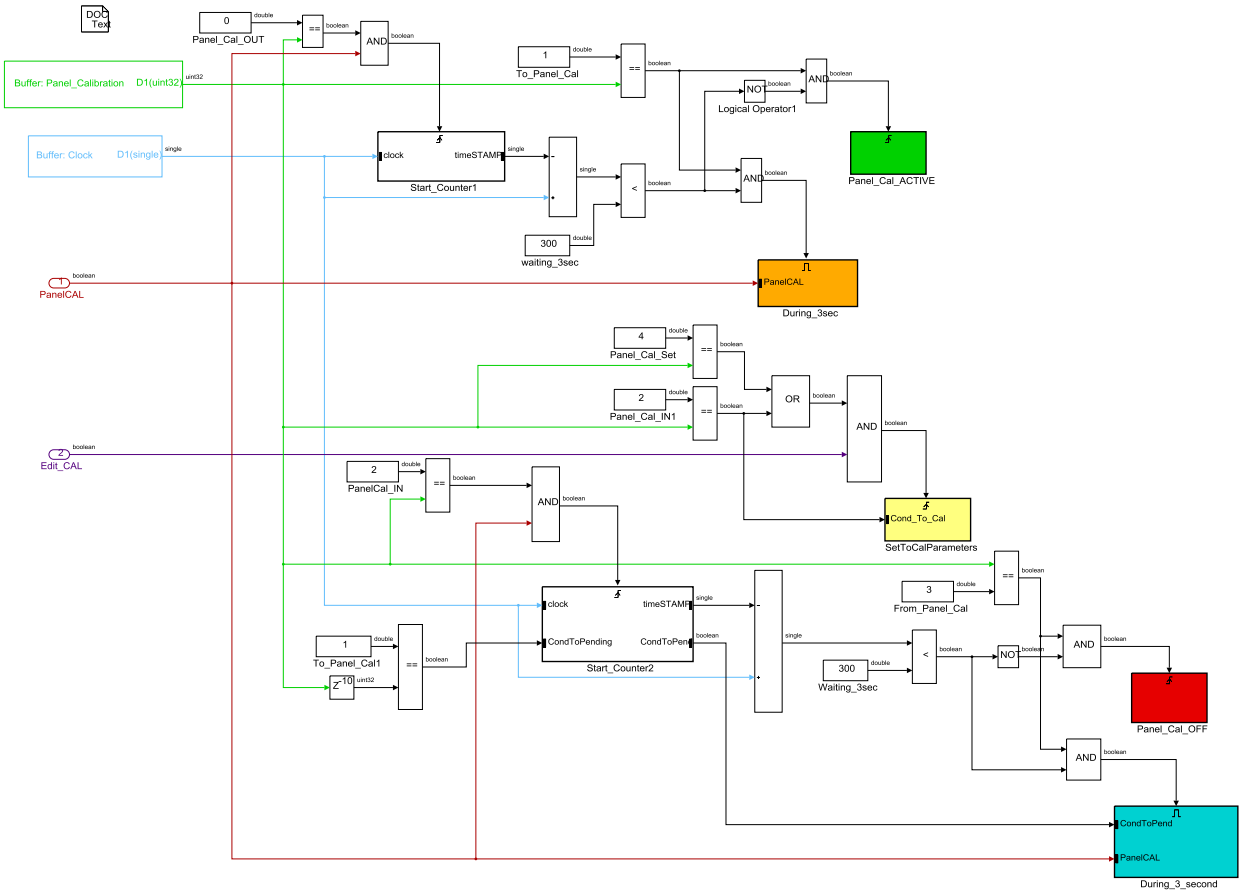


Figure 3.28: RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing

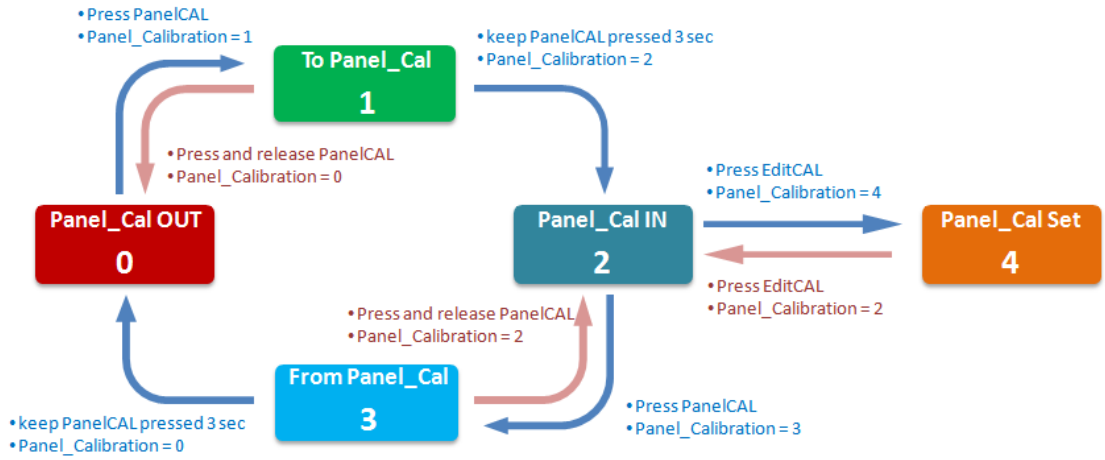


Figure 3.29: Macchina a stati che descrive la procedura di calibrazione dei potenziometri

Nella figura 3.29, viene rappresentato graficamente il circuito sequenziale degli stati raggiungibili, descrivendo in particolare le transizioni e le condizioni possibili per entrare o uscire da ciascuno di essi.
La condizione iniziale assegnata per "Panel_Calibration", viene inizializzata nel modello per mezzo della cella di memoria, ed è relativa allo stato "**0**", che appunto fa riferimento alla condizione **Panel_Cal_OUT**.
La condizione che permette di transitare da uno stato all'altro, e quindi far cambiare valore alla variabile "Panel_Calibration", è gestita dall'utente, in quanto ha la possibilità di agire sull'apposito pulsante della consolle che controlla il segnale digitale "PanelCAL" portato in ingresso al sottosistema.
Se tale pulsante viene premuto, allora "PanelCAL" passa a true; tale transizione permette di avviare la procedura di calibrazione, poiché viene eseguito fin da subito il sottoblocco *Start_Counter1*, all'interno del quale si va scrivere "**1**" nella cella di "Panel_Calibration".

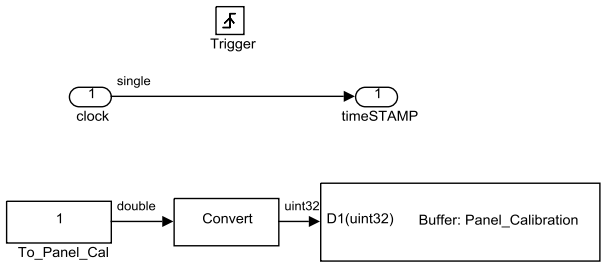


Figure 3.30: RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing/Start_Counter1

Sempre all'istante in cui si verifica il trigger, si azzera anche un *counter* che viene impiegato per definire un'altra condizione sulla variabile "Panel_Calibration", poiché arrivati a questo punto, si valuta se effettivamente c'è l'intenzione di voler realizzare la calibrazione dei potenziometri, andando a vedere che il pulsante "PanelCAL" venga mantenuto premuto per almeno 3 secondi. Se si verifica questo, ovvero "PanelCAL" resta a true per almeno 3 sec, allora viene eseguito il sottoblocco *Panel_Cal_ACTIVE*, all'interno del quale "Panel_Calibration" viene sovrascritta a "**2**".

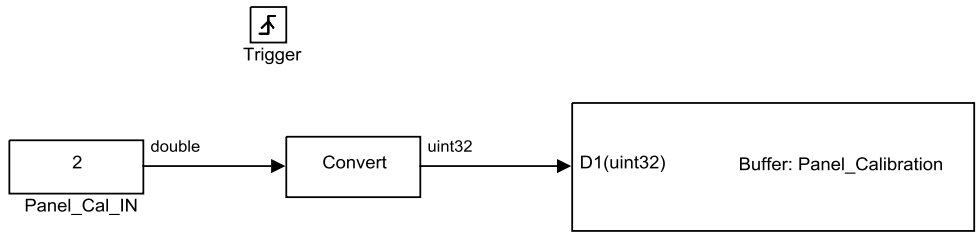


Figure 3.31:
RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing/Panel_Cal_ACTIVE

Invece, in caso di rilascio anticipato del pulsante "PanelCAL" prima che siano trascorsi i 3 secondi, si esce automaticamente dalla procedura di calibrazione. Questo nel modello è descritto all'interno del sottoblocco (evidenziato in arancio) *During_3sec*, in cui si va a scrivere "**0**" sulla cella di memoria di "Panel_Calibration" quando la condizione di enable viene soddisfatta.

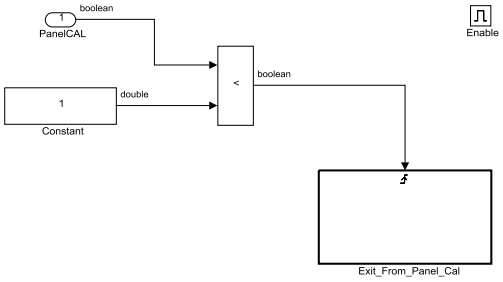


Figure 3.32:
RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing/During_3sec

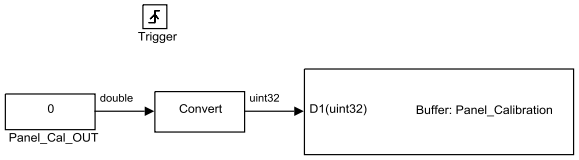


Figure 3.33:
RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing/During_3sec/Exit_From_Panel_Cal

Quando lo stato di "Panel_Calibration" è impostato a "**2**", il comando "PanelCAL" ricopre una funzione duplice all'interno del modello, a seconda del fatto che lo si rilasci subito dopo averlo premuto, o lo si mantenga premuto per un tempo sufficiente. In ogni caso, quando il pulsante "PanelCAL" viene premuto, si innesca il sottoblocco *Start_Counter2*, che permette di reimpostare un'altro contatore e impone la scrittura dello stato "**3**" sulla cella di memoria di "Panel_Calibration".

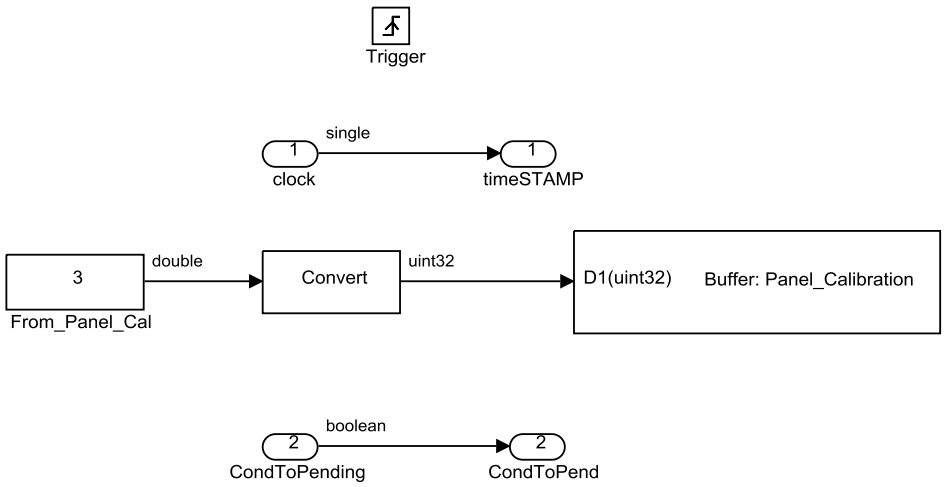


Figure 3.34:
RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing/Start_Counter2

Il ruolo del contatore in questo caso, è quello di distinguere le due funzioni assegnate al comando "PanelCAL" in modo da decidere quale delle due assecondare. In un caso, si avrà che se il segnale rimane a true oltre un tempo stabilito a 3 secondi, allora la funzione del comando è quella di uscire dalla procedura di calibrazione, dal momento che viene eseguito il sottoblocco 'triggerato' denominato *Panel_Cal_OFF*, in cui viene scritto "**0**" sulla cella di "Panel_Calibration".
Nell'altro caso si avrà che il comando "PanelCAL" viene premuto momentaneamente o comunque per un tempo inferiore a 3 secondi, allora ricopre il ruolo di selettore e serve a decidere su quale dei tre potenziometri presenti nella consolle, deve essere realizzata la calibrazione.
Quest'ultimo caso viene trattato all'interno del sottoblocco *During_3second* (evidenziato in azzurro), nel quale si li rileva l'istante di rilascio di "PanelCAL" per generare un'azione di trigger sul sottoblocco *Pot_Selector*.

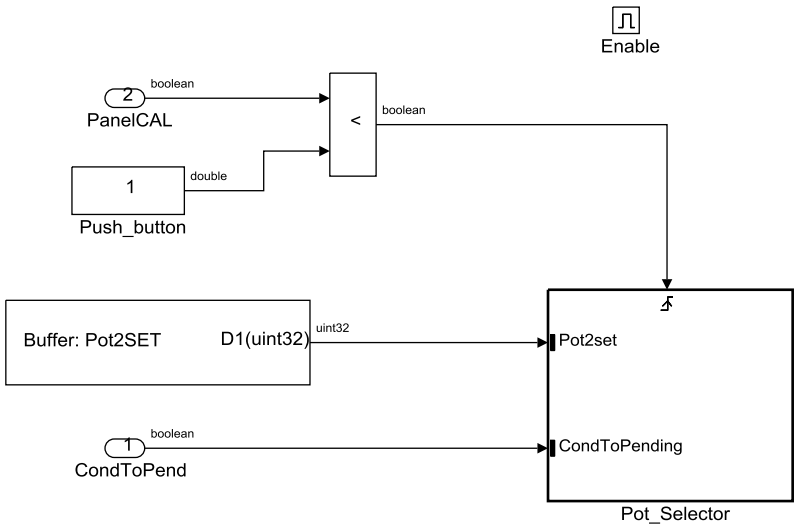


Figure 3.35:
RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing/During_3_second

La variabile "Pot2Set" presente in ingresso al sottoblocco e inizializzata nell'apposita cella di memoria serve appunto ad identificare il tipo di potenziometro da selezionare durante la fase di calibrazione della consolle. Nella figura seguente si rappresenta il circuito logico implementato nel modello per la selezione dei potenziometri, indicando le transizioni della variabile "Pot2Set" e le condizioni necessarie per cui esse si verificano:

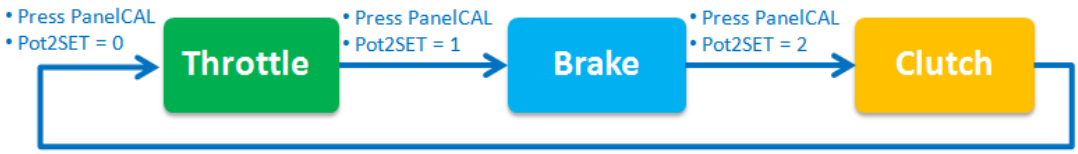


Figura 3.36: Schema a blocchi del processo di selezione dei potenziometri

Coerentemente al circuito riportato in figura 3.36, ogni volta che si preme "PanelCAL", si verifica la condizione di trigger sul sottoblocco *Pot_Selector*, pertanto viene eseguito un ciclo di somma in cui la variabile *Pot2Set* incrementa di

un’unità prima di raggiungere il limite superiore imposto a 2, dopodiché si riporta nuovamente a 0.

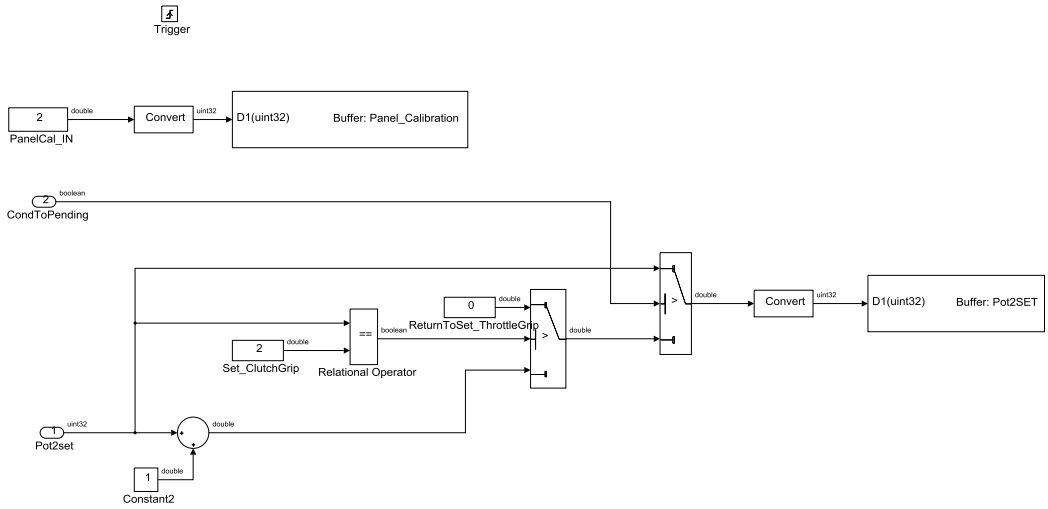


Figure 3.37:
RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing/During_3_second/Pot_Selector

Viene inoltre sovrascritto il valore” **2**” sulla cella di “Panel_Calibration” (in precedenza portata a “**3**”).

La condizione nominata “CondToPending” serve ad impedire l’esecuzione del calcolo di “Pot2Set” la prima volta che si entra nella procedura di calibrazione (passaggio di “Panel_Calibration” da “**1**” a “**2**”), così da annullare nel modello l’effetto dovuto all’azione di rilascio del pulsante “PanelCAL”.

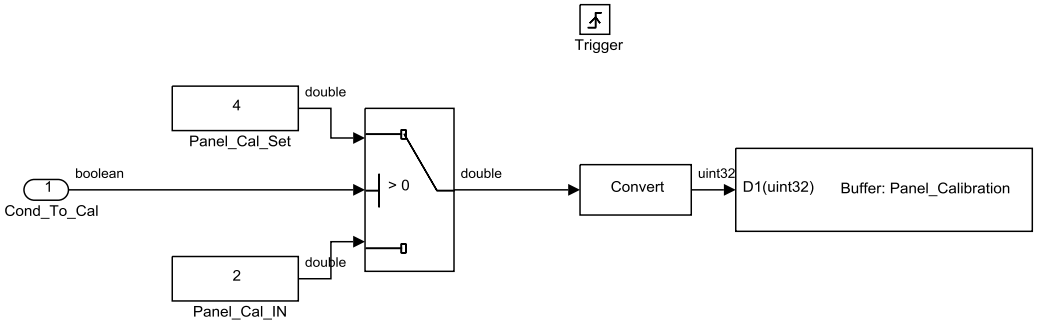


Figure 3.37: RideIT_Consolle_Model/INPUT/Digital_Input/PanelCalProcessing/SetToCalParameters

In ultima analisi abbiamo il comando digitale “Edit_CAL” sul quale può agire l’utente durante la fase di calibrazione della consolle per effettuare le nuove regolazioni sui potenziometri. Nel caso di “Panel_Calibration” impostata a “**2**”, tale comando, se premuto, genera un trigger per il sottoblocco *SetToCalParameters*, all’interno del quale avviene la transizione di stato di “Panel_Calibration”, che passa da **Panel_Cal_IN** a **Panel_Cal_Set**. Solamente quando si entra in questo stato è possibile registrare i nuovi valori (MAX e MIN) sulla corsa del potenziometro interessato dalla calibrazione.
Nel caso in cui, invece, viene premuto nuovamente il comando “Edit_CAL”, allora l’azione del trigger sul sottoblocco *SetToCalParameters* permette di riportare “Panel_Calibration” allo stato precedente, cioè a “**2**”, e quindi di congelare i valori che si sono registrati in calibrazione.

3.6.1.8 Pulsante Spare

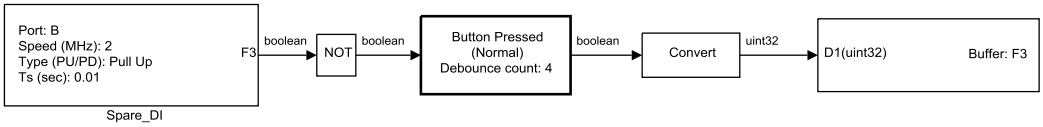


Figure 3.38: RideIT_Consolle_Model/INPUT/Digital Input/Spare_DI

Il sottosistema contiene il blocco che permette la lettura del segnale digitale sul *pin* di programmazione scelto sulla STm32 e che è connesso ad un pulsante di riserva presente sulla consolle. La scelta di dotare la consolle di un pulsante “spare*”* è giustificata dal fatto che nel corso dello sviluppo/debug di sistema, può accadere che ci sia la necessità di ricorrere all’utilizzo di un ulteriore input digitale da integrare nel circuito. Siccome il pulsante di riserva e il relativo collegamento elettrico alla scheda risultano già essere predisposti, sarà sufficiente, nel caso in cui occorra, lavorare solo sulla parte software per l’integrazione del nuovo segnale, anziché intervenire anche su quella hardware.

3.6.2 ADC

Il sottosistema contiene i tre macroblocchi dedicati alle funzioni di conversione anologico-digitale (ADC) e dai quali si effettua la lettura dei segnali provenienti dai tre potenziometri installati a bordo della consolle. Su ciascun blocco è presente una condizione di enable che governa anche uno switch posto a valle, pertanto la lettura da essi viene abilitata solo nel momento in cui siano soddisfatte le condizioni imposte dal segnale feedback di Status oltre che dalla variabile "Panel_Calibration", uscenti dalle rispettive cella di memoria. A tale proposito abbiamo che:

- La lettura dal potenziometro di manopola acceleratore viene abilitata quando il segnale "FDBK_Status" è impostato su MANUAL o SEMI_AUTO;
- La lettura dal potenziometro della leva del freno viene abilitata quando il segnale "FDBK_Status" è impostato su MANUAL o SEMI_AUTO o su CALIBRATION;
- La lettura dal potenziometro della leva di frizione viene abilitata quando il segnale "FDBK_Status" è impostato su MANUAL o CALIBRATION;
- La lettura su tutti i potenziometri è abilitata chiaramente durante la procedura di calibrazione prevista per la consolle, per consentire di ricavare l'ampiezza dello spazio in cui lavora il comando interessato dalla calibrazione. Questo è il caso in cui la variabile "Panel_Calibration" è impostata sullo stato 4, chiamato "Panel_Cal_Set";
- Per ciascun potenziometro, qualora le suddette condizioni non dovessero più essere verificate, interviene lo switch che interrompe di colpo la lettura lasciando passare in output dal sottosistema il valore fisso di fondo scala impostato per ciascuna attuazione.

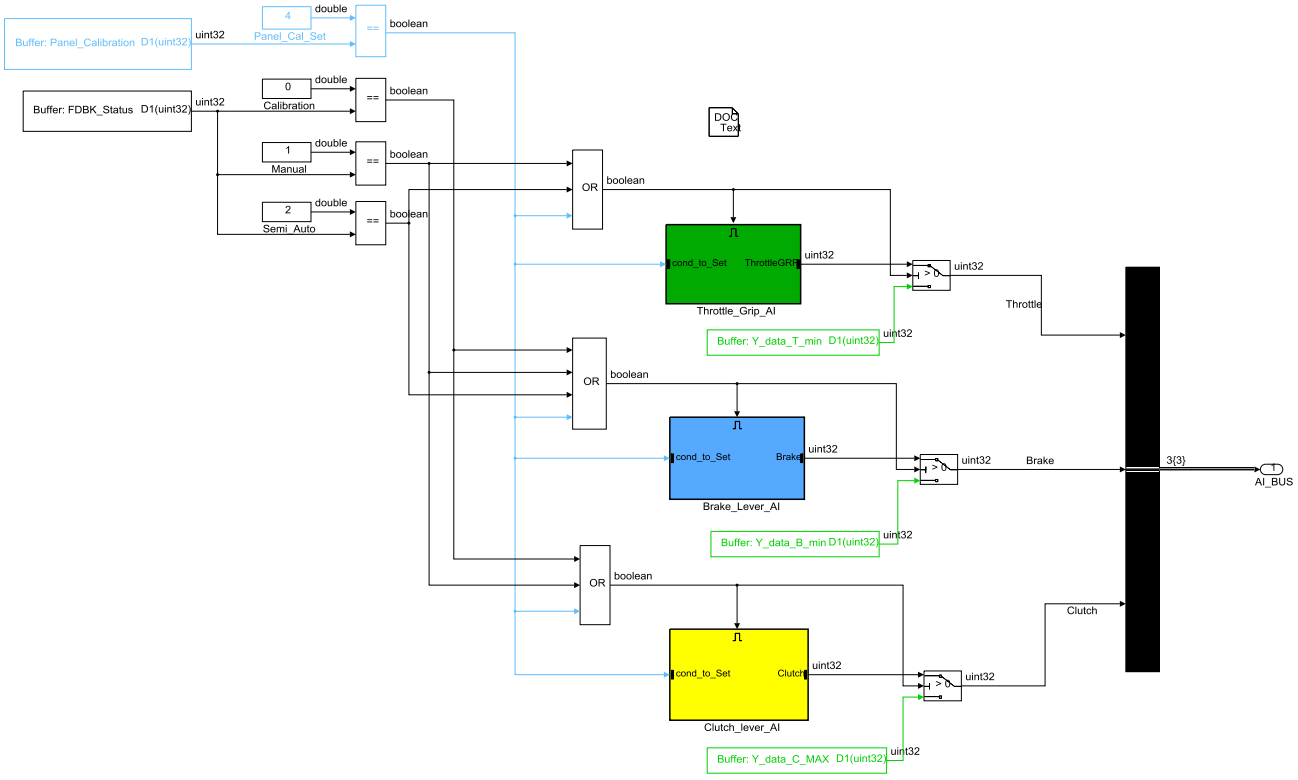


Figure 3.39: RideIT_Consolle_Model/INPUT /ADC

3.6.2.1 Lettura potenziometri

Throttle_Grip_AI: All'interno di questo sottosistema è presente il blocco ADC che imposta la funzione di conversione analogico-digitale sul pin di programmazione scelto sulla scheda STm32 per acquisire il segnale in tensione del potenziometro dell'acceleratore (compreso tra 0 V e 3.3V), e restituisce in uscita un valore intero leggibile dal microcontrollore.

Gli ADC della STm32 hanno una risoluzione di 12bit, questo significa che i valori letti da modello saranno compresi fra 0 e 4095, e che ogni singola unità corrisponde ad una variazione di tensione di circa 0.8mV (3.3V/4095).

Poiché i valori interi compresi nel range tra 0 a 4095 non forniscono informazioni sufficientemente significative, viene effettuata una interpolazione mono-dimensionale del segnale convertito attraverso una Look-Up-Table (LUT). Questo blocco genera in output un segnale corrispondente al grado di apertura della

manopola acceleratore (espresso in percentuale), in funzione della tensione acquisita dal rispettivo potenziometro.

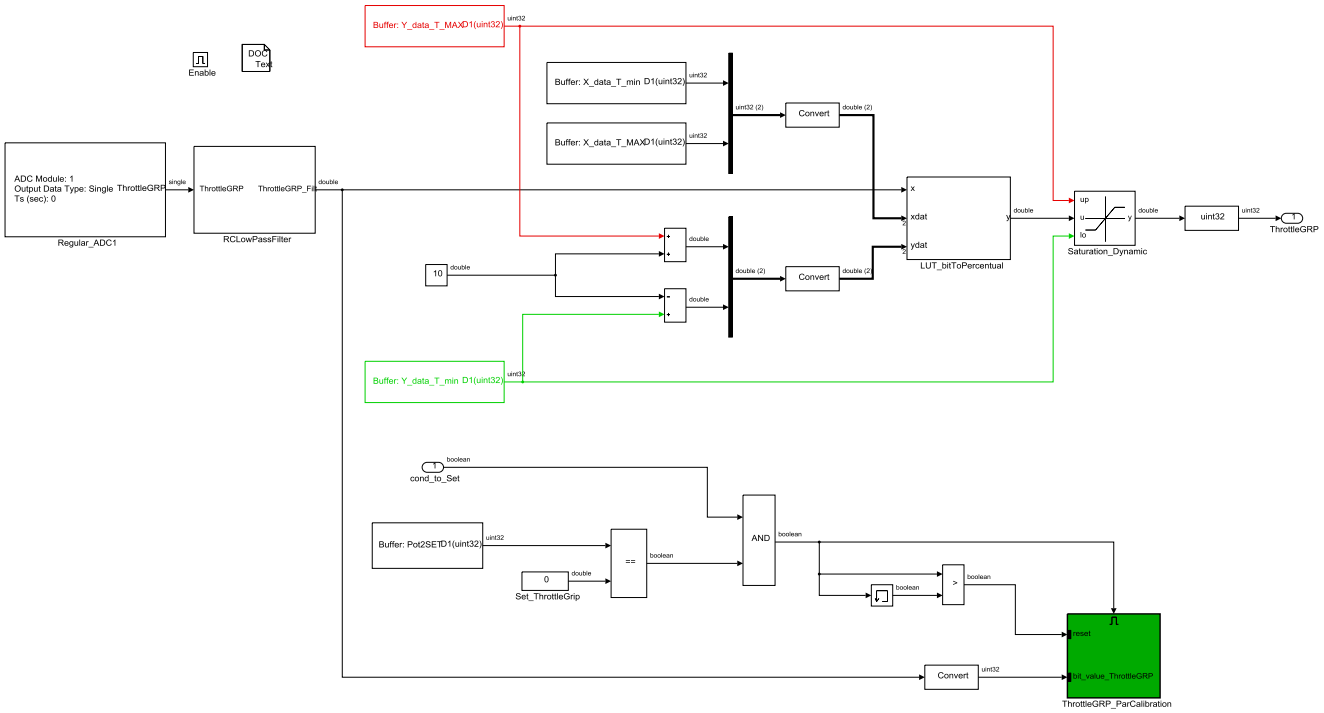


Figure 3.40: RideIT_Consolle_Model/INPUT/ADC/Throttle_Grip_AI

È possibile modificare nel corso della simulazione la caratteristica da interpolare agendo rispettivamente sui vettori Xdat e Ydat.
Il valore ottenuto in uscita dalla LUT viene fatto passare in un blocco di saturazione che mantiene il valore % compreso tra i limiti Min e Max stabiliti per il comando dell'acceleratore. Anche sul blocco saturazione vengono impostati i limiti Min e Max dall'esterno del blocco per consentire la possibilità di effettuare modifiche dei parametri nel corso dell'esecuzione del modello. Il blocco *Convert* viene inserito a valle per cambiare il formato della variabile da single a uint32.

Nel sottosistema **RCLowPassFilter** viene implementata la funzione di trasferimento relativa ad un filtro passa basso RC del primo ordine. Tale sottosistema è posto a valle del blocco ADC per fare in modo che sul segnale

acquisito vengano opportunamente eliminati i disturbi di frequenza superiore a 1.6 Hz. Una soluzione più adatta e sicuramente più elegante, in quanto permetteva di implementare il filtro in forma più compatta, sarebbe stata quella di utilizzare i blocchi specifici della libreria di Simulink dedicati alla funzione di filtraggio (in quel caso a partire dalla frequenza di taglio si devono determinare i coefficienti dai quali andare poi a definire la funzione di trasferimento).

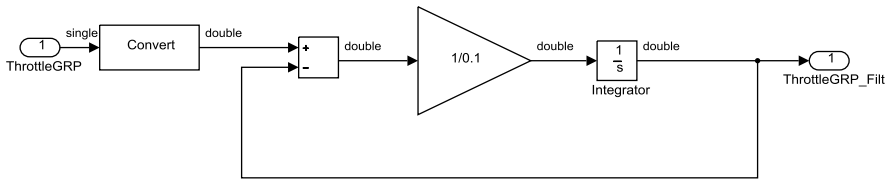


Figure 3.41: RideIT_Consolle_Model/INPUT /ADC/Throttle_Grip_AI/RCLowPassFilter

La logica che permette di acquisire i limiti dello spazio di lavoro della manopola acceleratore e quindi calcolare i parametri utili a far coincidere la corsa meccanica effettiva con la corsa teorica del comando in fase di calibrazione viene eseguita nel sottosistema **ThrottleGRP_ParCalibration.**

Selezionando il potenziometro acceleratore (Pot2SET = 0) in fase di calibrazione viene abilitato tale sottosistema; a quel punto se viene fatta ruotare la manopola dell'acceleratore, attraversando i limiti della corsa utile che si desidera imporre, nel sotto blocco si memorizzano i parametri di massimo e minimo tra quelli campionati durante la movimentazione del comando.

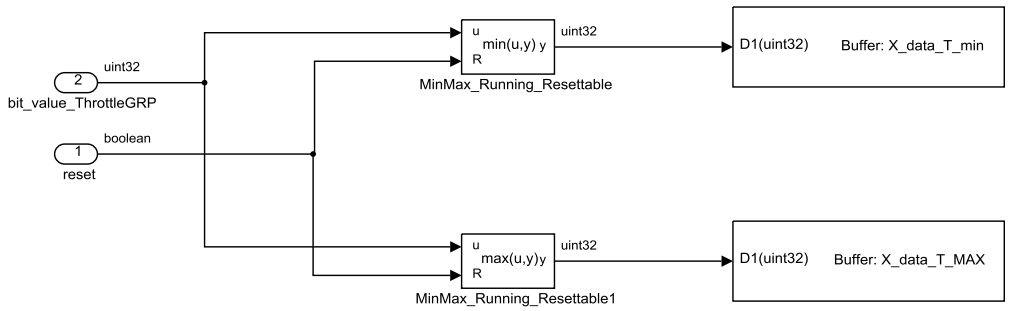


Figure 3.42:
RideIT_Consolle_Model/INPUT/ADC/Throttle_Grip_AI/ThrottleGRP_ParCalibration

I blocchi *MinMax_Running_Azzerable* ricevono entrambi un input che è il comando reset e consente appunto di reimpostare i valori di Max e Min all'avvio dell'acquisizione.

Brake_Lever_AI e Clutch_lever_AI: allo stesso modo, anche i segnali acquisiti dai potenziometri di freno e frizione vengono trattati in maniera completamente analoga nei rispettivi sottosistemi, rispetto a quanto visto per il potenziometro dell'acceleratore. Tutti i dati ricevuti dai sensori sono opportunamente filtrati per essere trasmessi e analizzati in modo più chiaro, mentre, per quanto riguarda le operazioni di conversione, gestite attraverso le Look-Up-Table, abbiamo in questo caso un'impostazione diversa per le unità di misura dei segnali, che vengono tradotti in termini di set point di posizione (millimetri) da richiedere agli attuatori di freno e frizione.

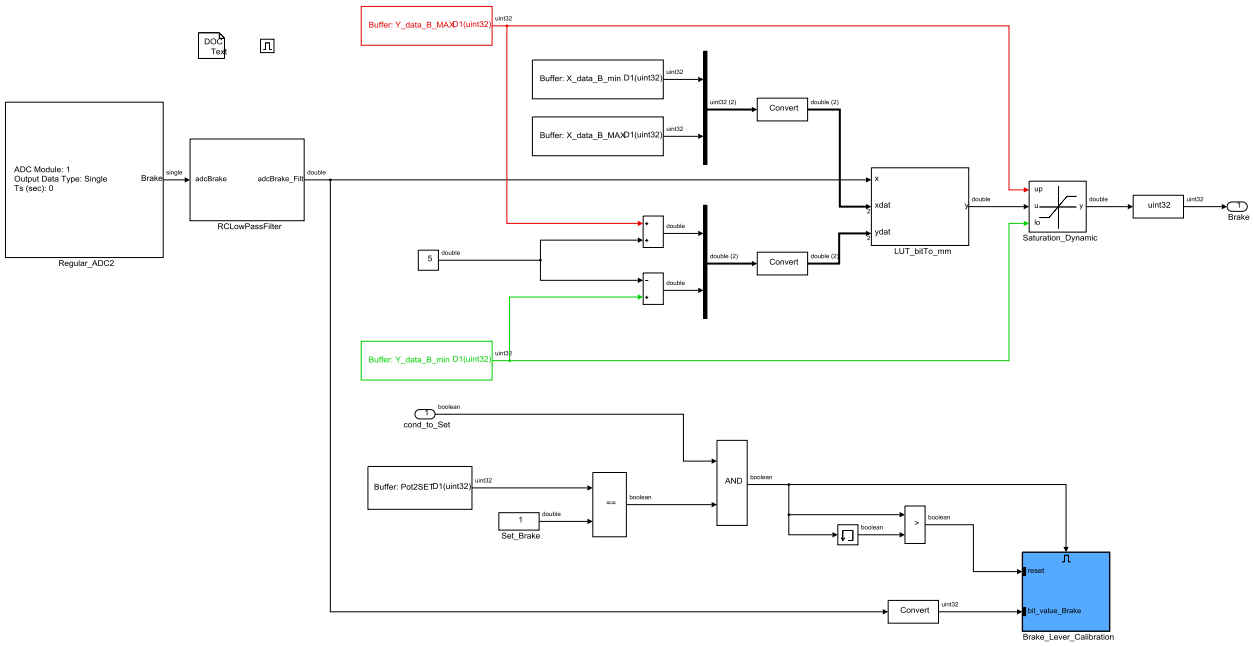


Figure 3.43: RideIT_Consolle_Model/INPUT/ADC/Brake_Lever_AI

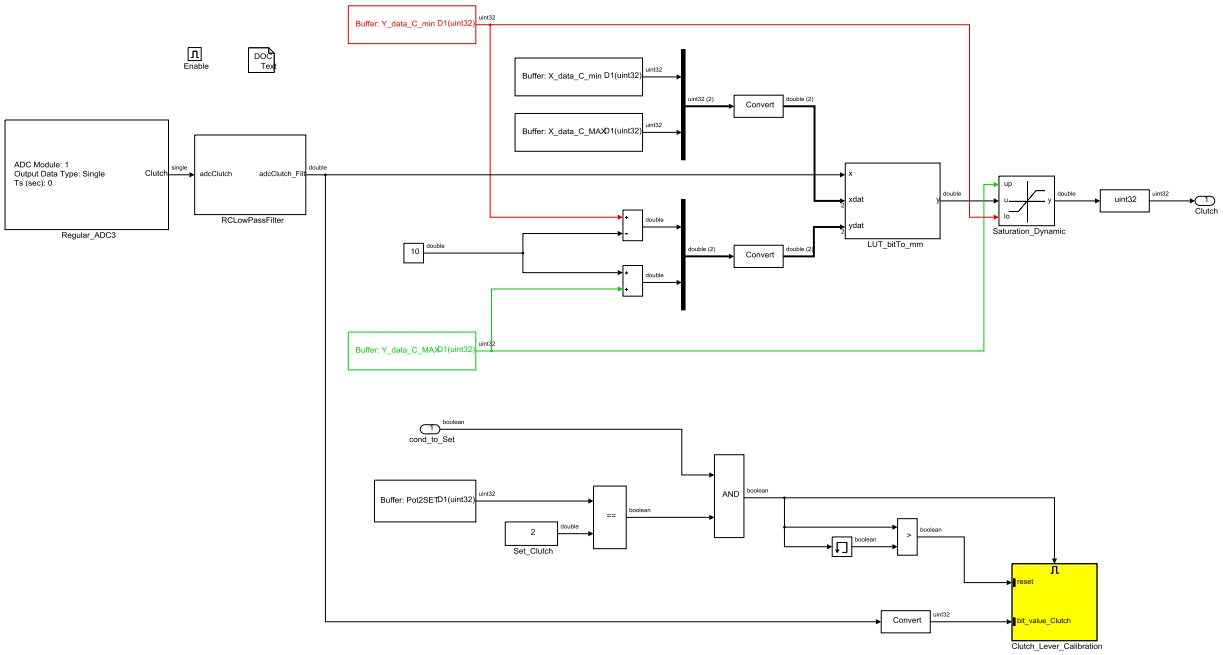


Figure 3.44: RideIT_Consolle_Model/INPUT/ADC/Clutch_lever_AI

3.7 OUTPUT

Nel sottosistema OUTPUT si ricevono i segnali di feedback trasmessi via UDP dal CompactRIO, i quali vengono prima manipolati nei vari sottosistemi, per poi essere inviati alle porte del microcontrollore riservate al comando delle spie a LED presenti sulla consolle. I pinout del MCU sono tutti configurati da software attraverso i blocchetti DO (digital output) della libreria Waijung e messi in mostra sul lato destro della figura 3.45. Le impostazioni dentro a questi blocchetti, riguardano in particolare le porte GPIO della STm32 e le resistenze interne, in questo caso fissate come push-pull per i DO.
Visto che i vari blocchetti DO sono tutti alimentati da segnali booleani, su ciascun LED si verificheranno le seguenti condizioni:

- Nei casi in cui il segnale che arriva in input al blocchetto DO si trova a livello "HIGH/ "1", la tensione sul pin di riferimento si troverà ad un livello alto e quindi il LED verrà acceso;

- Nei casi in cui il segnale che arriva in input al blocchetto DO si trova a livello "LOW"/ "0”, la tensione sul pin di riferimento si troverà ad un livello basso e quindi il led verrà spento.

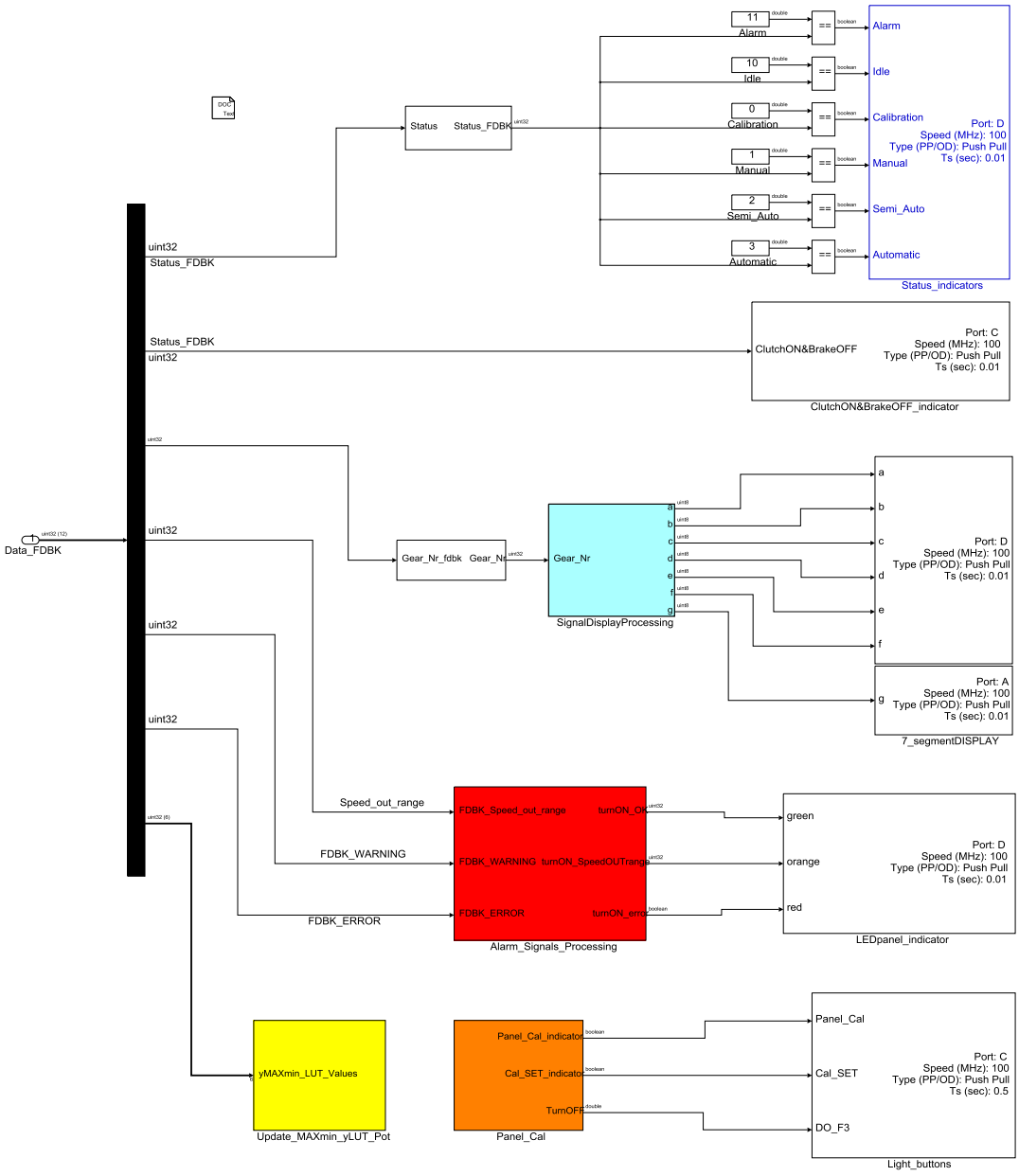


Figure 3.45: RideIT_Consolle_Model/OUTPUT

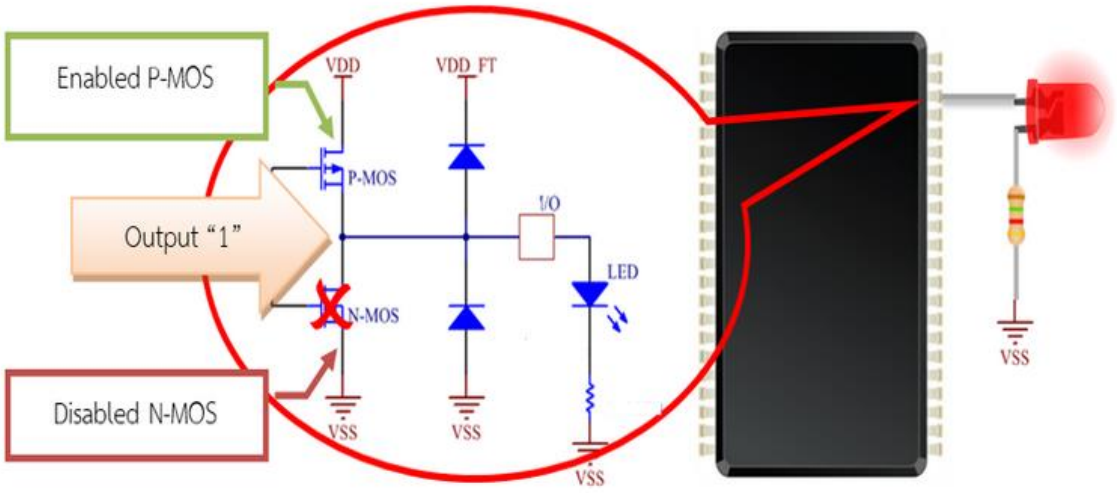


Figure 3.46: Push-Pull output circuit current for sourcing system

Come si osserva dalla figura 3.46, collegando il LED tra il pin di output della scheda STm32 e il ground, il LED viene acceso quando lo stato di output del segnale è uguale a "1", dunque in presenza della V_{DD} la corrente può fluire dal microcontrollore verso ground. Sono state inserite delle resistenze esterne nel circuito elettrico della consolle allo scopo di limitare la corrente massima che attraversa ciascun LED.

Verranno ora analizzati in maggiore dettaglio i singoli sottosistemi racchiusi all'interno del sottosistema degli OUTPUT.

3.7.1 Display a 7 segmenti

SignalDisplayProcessing è un sottosistema creato per pilotare la periferica display a 7 segmenti della consolle. Si convertono i valori di "Gear_Nr" e di "Pot2SET" in un set di segnali (a, b, c, d, e, f, g) con cui vengono alimentate le porte dei LED del display a 7 segmenti impiegato per il monitoraggio del numero di marcia innestato, o del sensore da calibrare nel caso in cui si stia effettuando una calibrazione su un potenziometro.

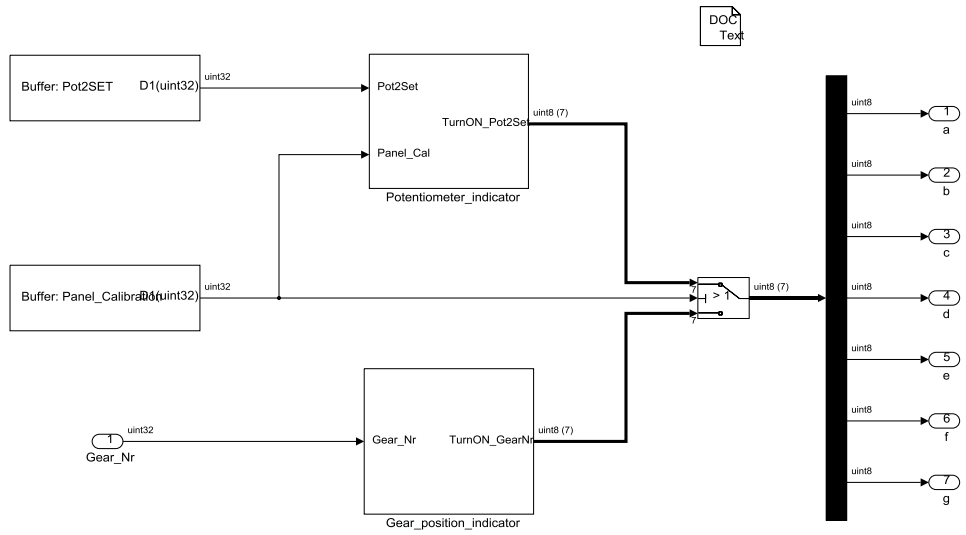


Figure 3.47: RideIT_Consolle_Model/OUTPUT/SignalDisplayProcessing

N° di marcia innestato	0 (Neutral),1,2,3,4,5,6
Potenziometro Acceleratore	A
Potenziometro Freno	B
Potenziometro Frizione	C

Tabella 3.2: significato delle diverse voci che compaiono sul display

Le conversioni di "Gear_Nr" e di "Pot2SET" vengono effettuate rispettivamente all'interno dei sottoblocchi *Gear_position_indicator* e *Potentiometer_indicator*. Siccome nella consolle è presente un'unica periferica display a 7 segmenti, i due segnali non possono essere visualizzati contemporaneamente, per questo motivo è presente uno switch controllato dalla variabile "Panel_Calibration" per contraddistinguere le due situazioni. Passa così in secondo piano l'indicazione relativa al numero di marcia durante la calibrazione dei potenziometri (Panel_Calibration > 1) per dare la possibilità all'utente di rendersi conto di ciò che sta accadendo sulla consolle, dal momento che vengono visualizzate tutte le informazioni utili per eseguire la calibrazione direttamente sul display a 7 segmenti.

Potentiometer_indicator: La calibrazione ha inizio nel momento in cui l'utente digita il comando "Edit_Cal" sulla consolle, e solo in quel momento il segnale "Panel_Cal" cambia di stato passando da "Panel_Cal_IN" a "Panel_Cal_Set ". Il verificarsi di questo evento viene visualizzato sul display facendo lampeggiare il simbolo relativo al potenziometro selezionato per la calibrazione (la frequenza di lampeggio è impostata a 2 Hz), poiché solo un potenziometro per volta può essere tarato. Una volta terminata la calibrazione sullo specifico potenziometro, il segnale sul display cessa di lampeggiare ritornando ad essere fisso.

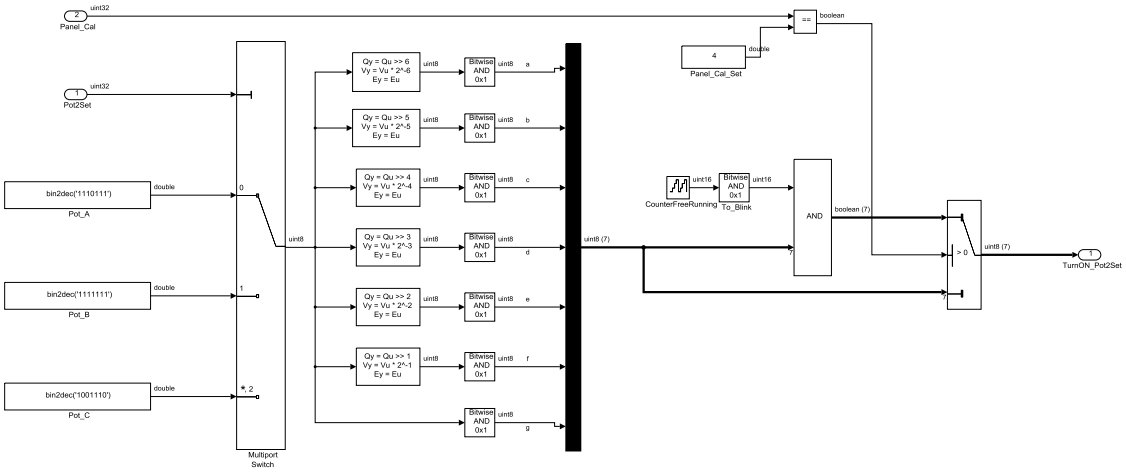


Figure 3.48:
RideIT_Consolle_Model/OUTPUT/SignalDisplayProcessing/Potentiometer_indicator

3.7.2 Spie a LED

Nel sottosistema **Alarm_Signals_Processing** si elaborano i segnali digitali che determinano il funzionamento delle spie luminose a LED presenti sulla consolle. La spia verde e quella arancione sono alimentate rispettivamente dai segnali "Connection_OK" e "FDBK_Speed_out_range", mentre la spia rossa può essere alimentata da due tipologie di segnali che arrivano dal sistema.
In quest'ultimo caso i due segnali vengono messi a confronto tra loro, dovendo necessariamente distinguere, mediante l'adozione di un unico LED, il caso in cui

ci sia da segnalare un WARNING del sistema, rispetto a quello in cui ci sia da segnalare un ERROR del sistema stesso.

Tale distinzione viene eseguita nel modello andando a definire sul canale di alimentazione del LED due modalità differenti di generare il segnale, ovvero:

- Nel caso in cui ci sia da segnalare un WARNING, il LED viene fatto lampeggiare alla frequenza di 2 Hz;
- Nel caso in cui ci sia da segnalare un ERROR, il LED viene acceso fisso.

Si stabilisce inoltre in quale ordine di priorità devono stare le richieste dei due segnali, e quindi decidere quale dei due segnali deve essere ignorato, nel caso essi siano entrambi attivi.

Chiaramente il segnale ERROR ha una priorità maggiore rispetto a quello di WARNING per questo viene lasciato passare per primo.

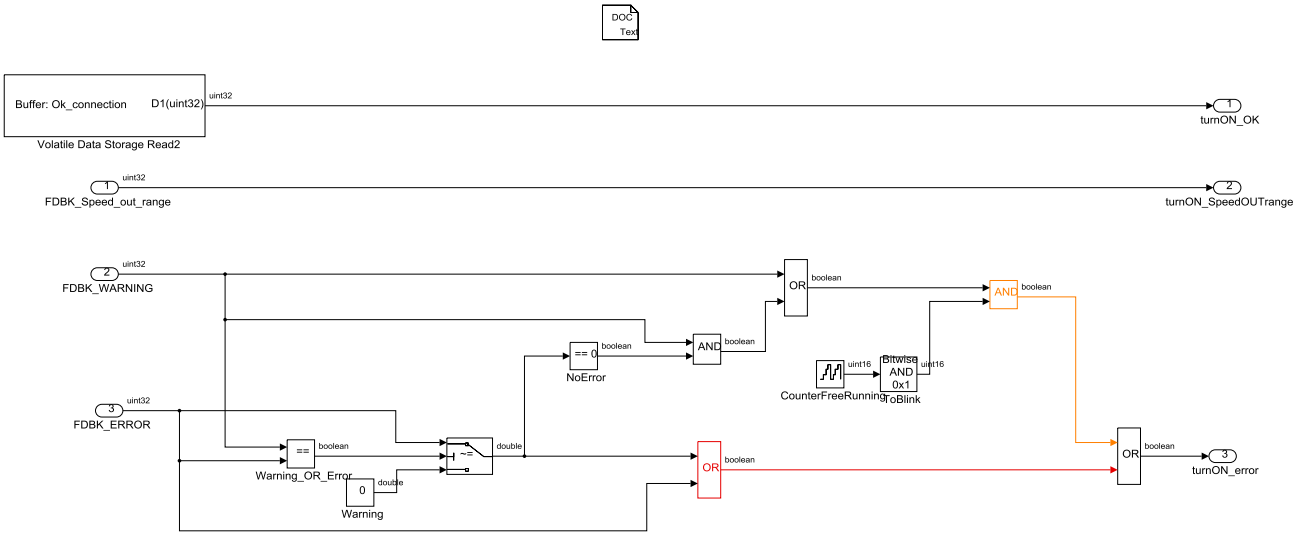


Figure 3.49: RideIT_Consolle_Model/OUTPUT/Alarm_Signals_Processing

3.7.3 Spie pulsanti calibrazione e spare

Nel sottosistema **Panel_Cal** si elaborano i segnali dei LED che si trovano integrati nei pulsanti “PanelCAL”, “Edit_CAL” e in quello di riserva.

Quest'ultimo in particolare, non svolgendo alcun ruolo nel sistema, nell'attesa che esso possa assolvere una qualche funzione in futuro, si è preferito lasciarlo spento e per questo motivo viene fissato un valore costante nullo sulla porta digitale.
Per quanto riguarda le altre due spie impiegate sui pulsanti dedicati alla calibrazione dei potenziometri, si ha che una volta entrati in questa procedura, il LED presente sul pulsante "PanelCAL" viene acceso e rimane tale per tutta la durata della calibrazione, poichè viene soddisfatta la disuguaglianza imposta alla variabile "Panel_Calibration". Mentre quando viene premuto il pulsante "Edit_CAL" si attiva lo stato **PanelCal_Set** , che permette di modificare i parametri di calibrazione su uno dei potenziometri presenti sulla consolle. Allora quando si verifica ciò, viene fatto lampeggiare il LED del pulsante "Edit_CAL", in modo che l'utente possa essere avvisato in quella fase, del fatto che sta eseguendo nuove regolazioni e quindi registrando nuove posizioni *Min* e *Max* della corsa del sensore. Il LED viene fatto lampeggiare fino a quando non viene premuto nuovamente il tasto "Edit_CAL", che fa in modo di congelare i nuovi parametri registrati.

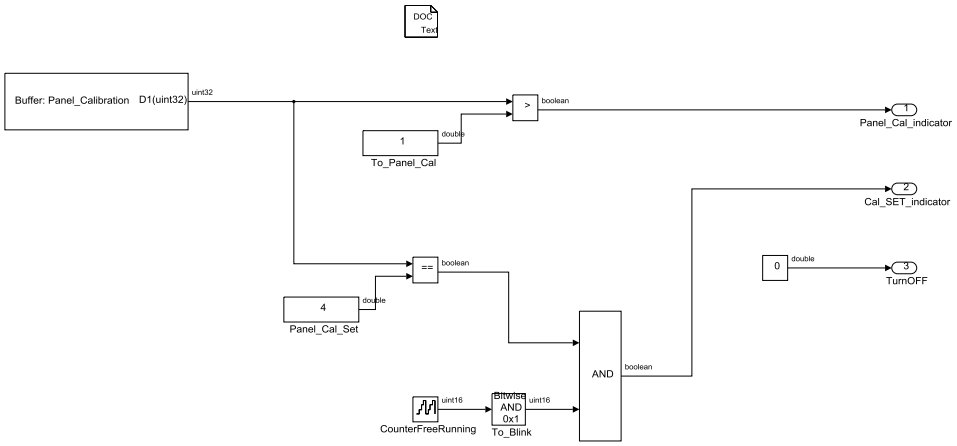


Figure 3.50: RideIT_Consolle_Model/OUTPUT/Panel_Cal

3.8 UDP_Communication

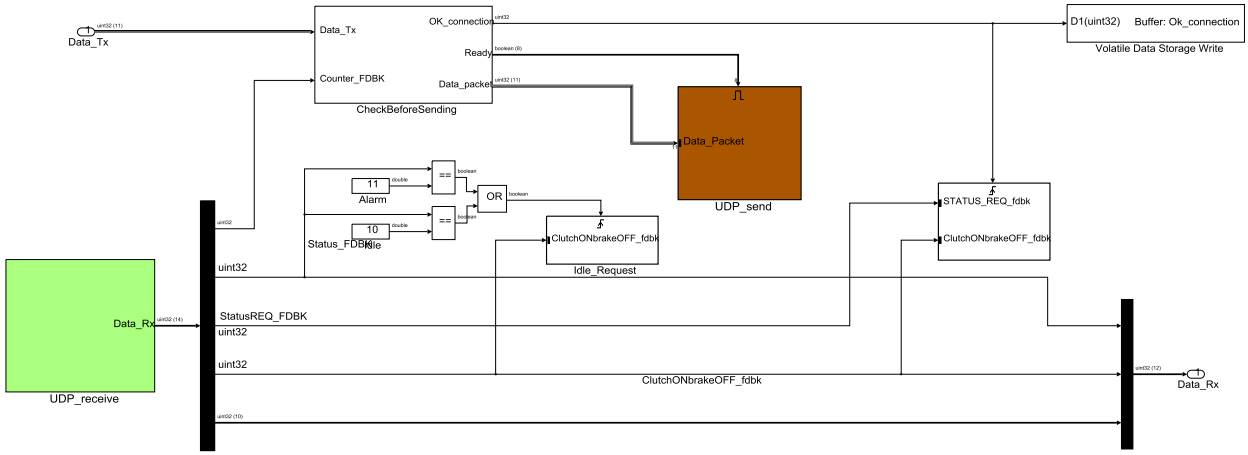


Figure 3.51: RideIT_Consolle_Model/UDP_Communication

In questo si trovano i blocchi che consentono di creare la connessione della consolle per ricevere/inviare pacchetti UDP sulla rete LAN. In output dal sottoblocco *UDP_receive* abbiamo un array "Data_Rx" contenente i dati di feedback che vengono ricevuti dal CompactRIO. Dall'array si estraggono i primi 4 elementi che vengono analizzati all'interno del sottosistema in appositi blocchi. I restanti elementi dell'array vengono invece spediti in output al sottosistema. Il primo elemento "Counter_FDBK" rappresenta un contatore che viene spedito all'interno del sottoblocco "CheckBeforeSending" allo scopo di verificare l'ordine dei pacchetti ricevuti e stabilire in base a ciò se la comunicazione tra i dispositivi sia stabile o meno.

Il segnale "OK_connection" contiene l'informazione relativa alla connessione tra il CompactRIO e la consolle e viene utilizzato per generare un trigger sul sottoblocco nel quale viene scritta l'informazione di "STATUS_REQ" e "ClutchON" sulle rispettive celle di memoria, relativa ai segnali di "StatusREQ_FDBK" e "ClutchONbrakeOFF_fdbk" che si rilevano nel primo istante in cui viene riconosciuta la connessione. Questo serve ad allineare i segnali trasmessi dalla consolle con quelli visti in lettura dal CompactRIO nell'istante

prima che la connessione tra i due dispositivi si attivi in tutti e due i sensi. Nel momento in cui si ricevono dati sulla porta di *UDP_receive*, la connessione si attiva, il booleano "OK_connection" passa a true; l'informazione viene portata anche al di fuori del sottosistema per comandare l'interfaccia a LED dedicata al monitoraggio della connessione. Il sottoblocco *UDP_Send* contiene il blocchetto di scrittura UDP dei dati da inviare al CompactRIO e viene eseguito solo quando è soddisfatta la condizione sulla porta Enable. I dati da trasmettere al cRIO sono raggruppati nell'array "Data_Packet" in formato di interi uint32.

Idle_Request: il segnale "StatusFDBK" viene comparato con i valori delle costanti che contrassegnano gli stati di *Alarm* e *Idle*. Al verificarsi di una delle due condizioni, viene 'triggerato' il sottoblocco *Idle_Request* andando a scrivere "10" sulla cella di memoria di "STATUS_REQ", in modo del tutto coerente alla macchina a stati associata al sistema. Inoltre, anche il segnale di "ClutchON&BrakeOFF" viene aggiornato al valore stabilito dal feedback che viene letto in quell'istante.

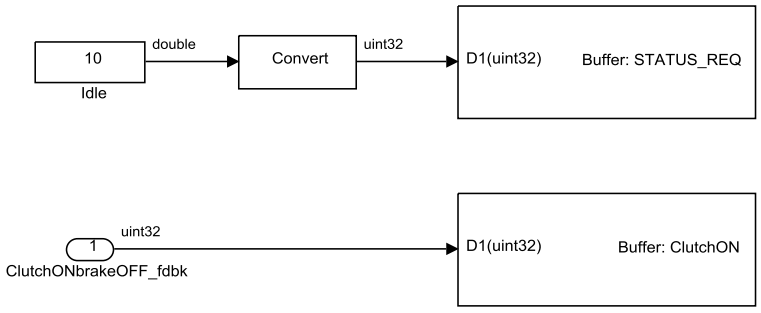


Figure 3.52: RideIT_Consolle_Model/UDP_Communication/Idle_Request

Tale accorgimento, che permette di aggiornare il valore di "ClutchON" e impostare lo "STATUS_REQ" su *Idle* nel caso in cui si verifichi un'emergenza sul sistema, fa in modo che l'utente, una volta che il sistema si ripristina portandosi in *Idle*, abbia la possibilità di effettuare nuovamente la richiesta dello stato in cui operava il sistema nella fase antecedente all'allarme, poiché altrimenti non

verrebbe riconosciuto l'evento a lato del modello real-time eseguito sulla piattaforma cRIO, dal momento che non verrebbe mai mutato il segnale "STATUS_REQ". D'altra parte invece viene 'triggerato' il sottoblocco *Idle_Request* quando viene rilevato uno "StatusFDBK" pari ad *Idle*, perché nel caso in cui venisse a mancare la connessione LAN sulla consolle e nel frattempo sul sistema venisse richiamato uno stato di *Idle*, e se poi in seguito a ciò venisse ripristinata la connessione, allora verrebbe negata all'utente la possibilità di richiedere lo stato attuato nella fase antecedente alla disconnessione poiché il segnale "STATUS_REQ" resterebbe comunque invariato. Aggiornando invece lo "STATUS_REQ" ad *Idle* si evita tale eventualità.

CheckBeforeSending: in questo sottosistema abbiamo il counter_FDBK, ovvero un numero che viene progressivamente incrementato dal cRIO e viene utilizzato per verificare l'ordine dei pacchetti UDP ricevuti dalla consolle.
In particolare, se il counter incrementa, viene eseguito il sottoblocco 'triggerato' in cui si memorizza il "timeSTAMP", rappresentante l'istante di tempo in cui un nuovo pacchetto di dati viene ricevuto dalla consolle. Successivamente viene messo a confronto tale segnale con il segnale di "CLOCK" del modello, che incrementa ad ogni passo di simulazione impostato a 0,01 secondi.
Dal confronto si stabilisce se la connesione è attiva, oppure se è interrotta: se la differenza è inferiore a 0.7 secondi, il modello reagisce portando a true il booleano "OK_connection", mentre se si supera tale intervallo di tempo si impone un'ulteriore condizione di "waiting", limitata a 2 secondi, prima di stabilire che la connessione si è interrotta (OK_connection = 0). Tale caratteristica determina una sorta di ciclo di isteresi che permette di reagire e segnalare l'evento con un certo ritardo, di fronte ad un problema di connessione che si può verificare nella rete in cui sono collegati i dispositivi.
Un'altra condizione introdotta dal "CLOCK" sul segnale "OK_connection" permette di passare alla modalità appena illustrata solo dopo un tempo di attesa, impostato a 2 sec, a partire dall'istante in cui si è fornita l'alimentazione sulla

consolle. In pratica ogni volta che la consolle viene accesa e il modello inizia a girare, c'è una condizione iniziale che impone un valore nullo sul booleano "OK_connection", ed è gestita dal segnale di "CLOCK" che logicamente si azzera ad ogni spegnimento. Questo consente di evitare il caso in cui durante il normale funzionamento del sistema, se per un qualche motivo venisse accidentalmente scollegata l'alimentazione sulla consolle e venisse poi in seguito ricollegata, il segnale "OK_connection" resti impostato all'ultimo valore precedente allo spegnimento: nel caso in cui questo si trovi impostato a true, all'istante di riaccensione si creerebbe infatti una discrepanza tra i segnali da inviare in input al cRIO e quelli che il cRIO stesso emette e invia come feedback alla consolle, in quanto durante la "messa in OFF" della consolle dal sistema, potrebbero variare i segnali del cRIO per effetto dell'interazione con la piattaforma Host, ma soprattutto si ha che il primo valore di "STATUS_REQ" emesso dalla consolle, è quello inizializzato dalla cella di memoria e corrisponde allo stato *Calibration*, pertanto potrebbe variare la richiesta di stato da attuare in maniera non controllata dall'utente. Se invece all'accensione si impone una condizione nulla sul segnale "OK_connection", nel momento in cui questo passa a "1", si aggiorna automaticamente il valore di "STATUS_REQ" e anche di "ClutchON" grazie ai dati di feedback inviati dal CompactRIO, impedendo in questo modo alla consolle di richiedere delle attuazioni non desiderate.

La trasmissione via UDP dei pacchetti di dati verso il CompactRIO viene gestita su base evento, attraverso la condizione nominata Ready utilizzata per eseguire il sottoblocco *UDP_send* e che viene generata all'interno di questo sottosistema. Sostanzialmente, il segnale Ready diventa true quando almeno uno degli elementi contenuti nell'array "Data_Tx", subisce una variazione, oppure in alternativa, almeno ogni 0.5 secondi, in quanto viene rilasciato un impulso da parte del "CLOCK" del modello.

In questo ultimo caso, il segnale di CLOCK che normalmente incrementa ogni 0.01 sec, viene trattato all'interno del blocco "floor", il quale riporta l'intero più vicino, inferiore o uguale, al numero ottenuto dividendo il segnale di "CLOCK"

stesso per 50. In tale maniera si è realizzato un diverso tipo di contatore che incrementa ogni 0.5 secondi. Con tale step time, viene rilevata la transizione di livello per generare l'evento che abilita la scrittura UDP.
Durante la calibrazione della consolle (Panel_Calibration > 1), si impone una condizione che fa in modo di lasciare invariati i segnali da trasmettere al cRIO, ai valori stabiliti dall'array "data_packetOLD" uscente dal sottoblocco innescato all'inizio della procedura.

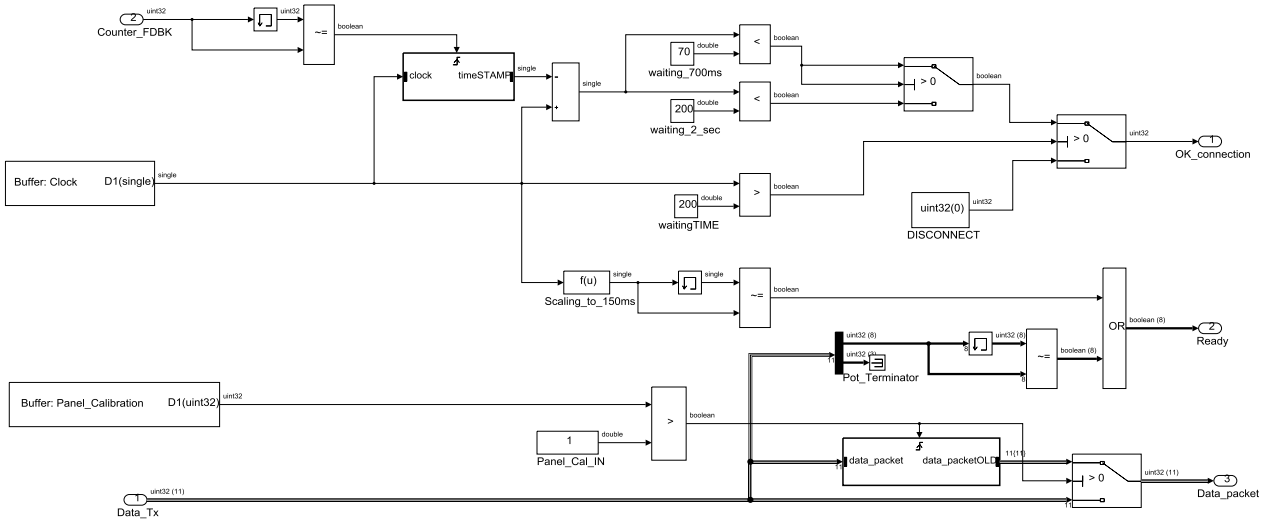


Figure 3.53: RideIT_Consolle_Model/UDP_Communication/CheckBeforeSending

UDP_receive: nel sottosistema è contenuto il blocco *Target_UDP_Receive* che permette di creare e configurare un'interfaccia di comunicazione per la lettura da un indirizzo remoto, utilizzando il protocollo UDP.
Durante l'esecuzione del modello un buffer "first in first out" (FIFO) riceve i dati. Questi vengono poi emessi sulla "Data port" ad ogni passo di simulazione.
Sono quattro le uscite configurabili su questo blocchetto, tre delle quali non vengono sfruttate all'interno del modello:

1) **Status**: è un segnale booleano che viene utilizzato per confermare il fatto che un nuovo pacchetto di dati sia stato ricevuto o meno.

2) **Remote Port**: indica il nome della porta utilizzata per la ricezione dei pacchetti di dati.
3) **Remote IP**: specifica l'indirizzo IP del dispositivo dal quale vengono spediti i pacchetti di dati.
4) **Data port**: specifica il formato e la dimensione dei pacchetti di dati che devono essere ricevuti ad ogni passo di simulazione.

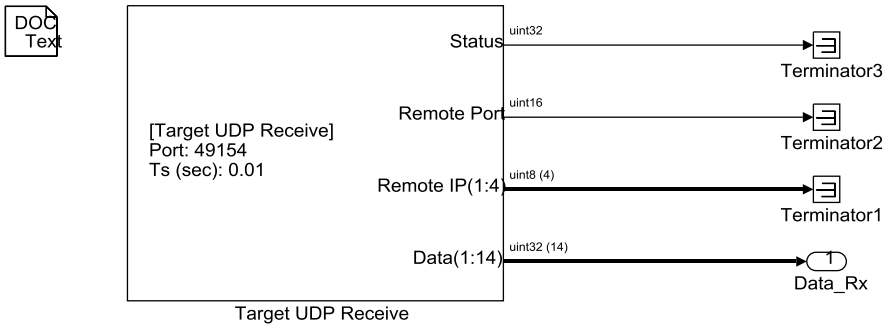


Figure 3.54: RideIT_Consolle_Model/UDP_Communication/UDP_receive

UDP_send: il blocco *Target_UDP_Send* impachetta i dati da inviare verso un indirizzo remoto (CompactRIO) utilizzando il protocollo UDP. Esso possiede tre porte di input:

1) **Port**: specifica il nome della porta utilizzata per la trasmissione dei pacchetti di dati.
2) **IP**: specifica l'indirizzo IP del dispositivo a cui si vuole spedire i pacchetti.
3) **Data**: specifica il formato e la dimensione dei pacchetti di dati che devono essere inviati.

Per quanto riguarda il segnale "Status" uscente dal blocchetto, si tratta di un booleano che riporta l'informazione relativa a quando un pacchetto di dati viene instradato nella rete con successo oppure viene perso; in questo caso però non viene impiegato nel modello.

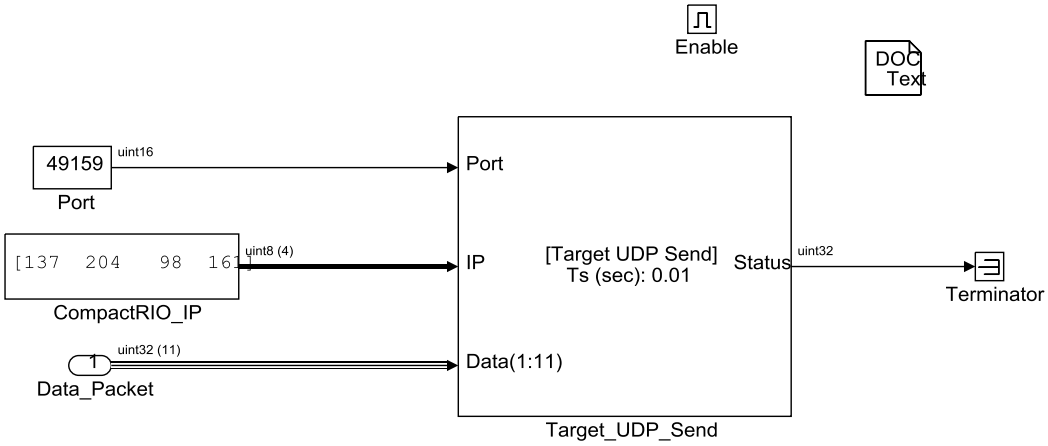


Figure 3.55: RideIT_Consolle_Model/UDP_Communication/UDP_send

Capitolo 4

Schemi di controllo real-time

In questo capitolo viene descritto il modello Simulink sviluppato dall'Ing. Michele [1] Taccioli per il controllo in tempo reale delle componenti principali del sistema RideIT. Esso infatti elabora gli input provenienti dalle interfacce utente o dalla centralina della moto per produrre delle variabili in output, alcune delle quali servono a gestire gli azionamenti della LinMot sulla moto, altre sono necessarie alla centralina, altre ancora sono informazioni che vengono semplicemente inviate all'interfaccia Host e alla Consolle perché utili all'utente.
Per la stesura di questa parte della tesi, in cui si descrivono a fondo gli schemi a blocchi dell'ambiente Simulink e in parte anche quelli di LabView che vanno a costituire il modello real-time, si è preso spunto da un lavoro precedentemente sviluppato da Michele Taccioli nell'ambito della sua tesi presso Ducati Corse, riguardo alla realizzazione di un software di controllo per la guida automatica nei test condotti al banco a rulli.
Per completezza e per evitare incomprensioni, alla fine del capitolo si descrive anche come il modello Simulink viene integrato nel progetto LabView Real-Time funzionante sulla piattaforma NI CompactRIO, riportando gli schemi utili del *VI* che permettono l'interazione tra i due modelli sviluppati in ambienti di programmazione differenti ma che girano insieme sul medesimo target.

4.1 Il modello Real-Time su Simulink

Nella figura 4.1 viene mostrato il livello a gerarchia più elevata del modello:

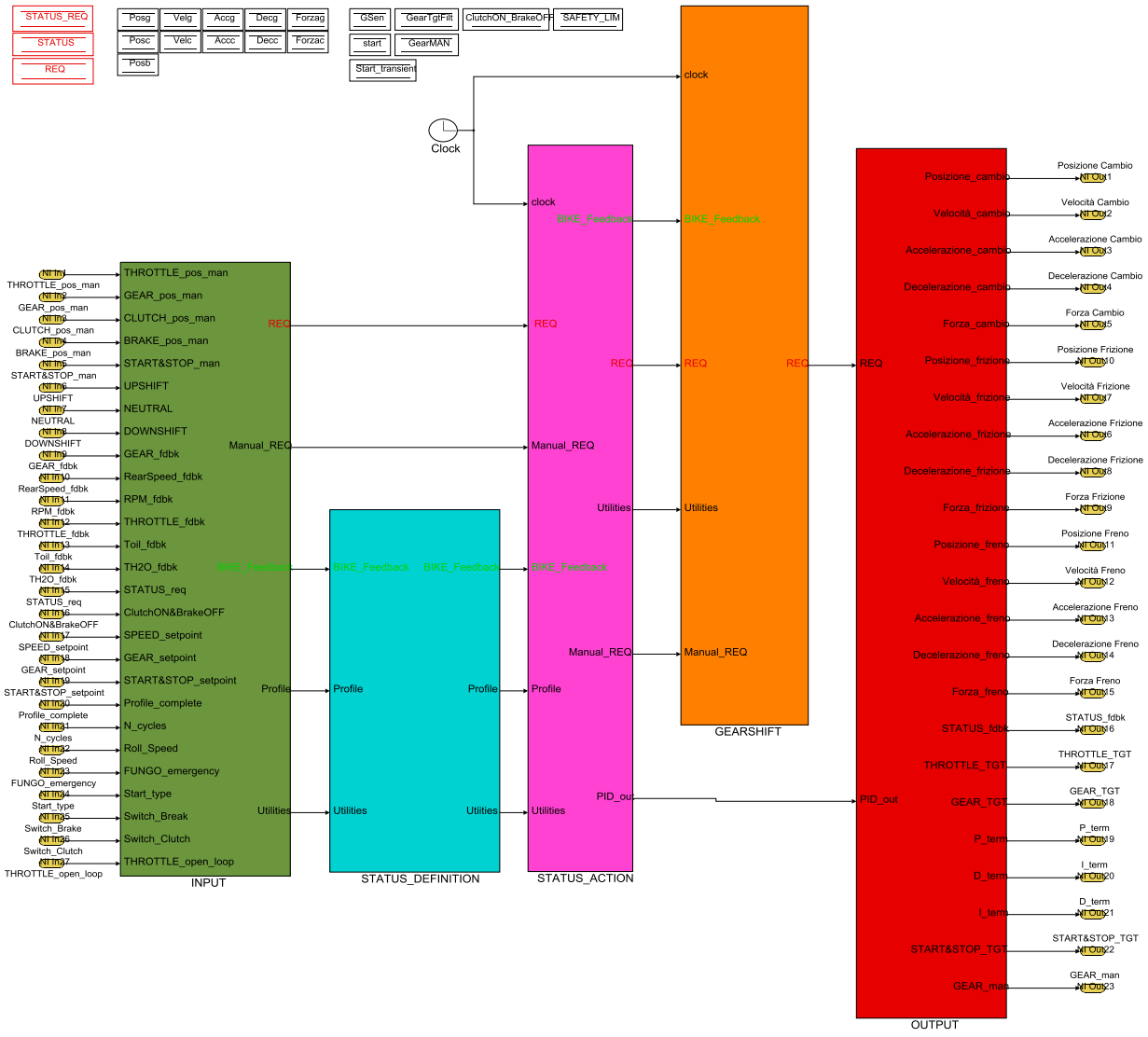


Figure 4.1. RideIT_RT_model

Si può osservare che è organizzato in sottosistemi, in ognuno dei quali sono racchiuse specifiche parti del modello.

- I blocchi *INPUT* e *OUTPUT* gestiscono rispettivamente le variabili in ingresso e in uscita al modello.
- Il blocco *STATUS_DEFINITION* serve per determinare in quale stato si trova il sistema.
- Il blocco *STATUS_ACTION* determina le azioni da compiere sulle variabili, secondo lo stato in cui ci si trova.
- Il blocco *GEARSHIFT* gestisce il cambio marcia.

In questo livello del modello viene inoltre generato il segnale di tempo “clock” e sono inoltre inizializzate le celle di memoria che sono utilizzate all’interno dei vari sottosistemi.

4.2 INPUT

All’interno del sottosistema *INPUT* sono importate nel modello determinate variabili e grandezze riguardanti il funzionamento della moto, provenienti dal progetto LabView Real-Time. Sono quindi costruiti dei vettori di grandezze e variabili che sono richiamati nelle restanti parti del modello.

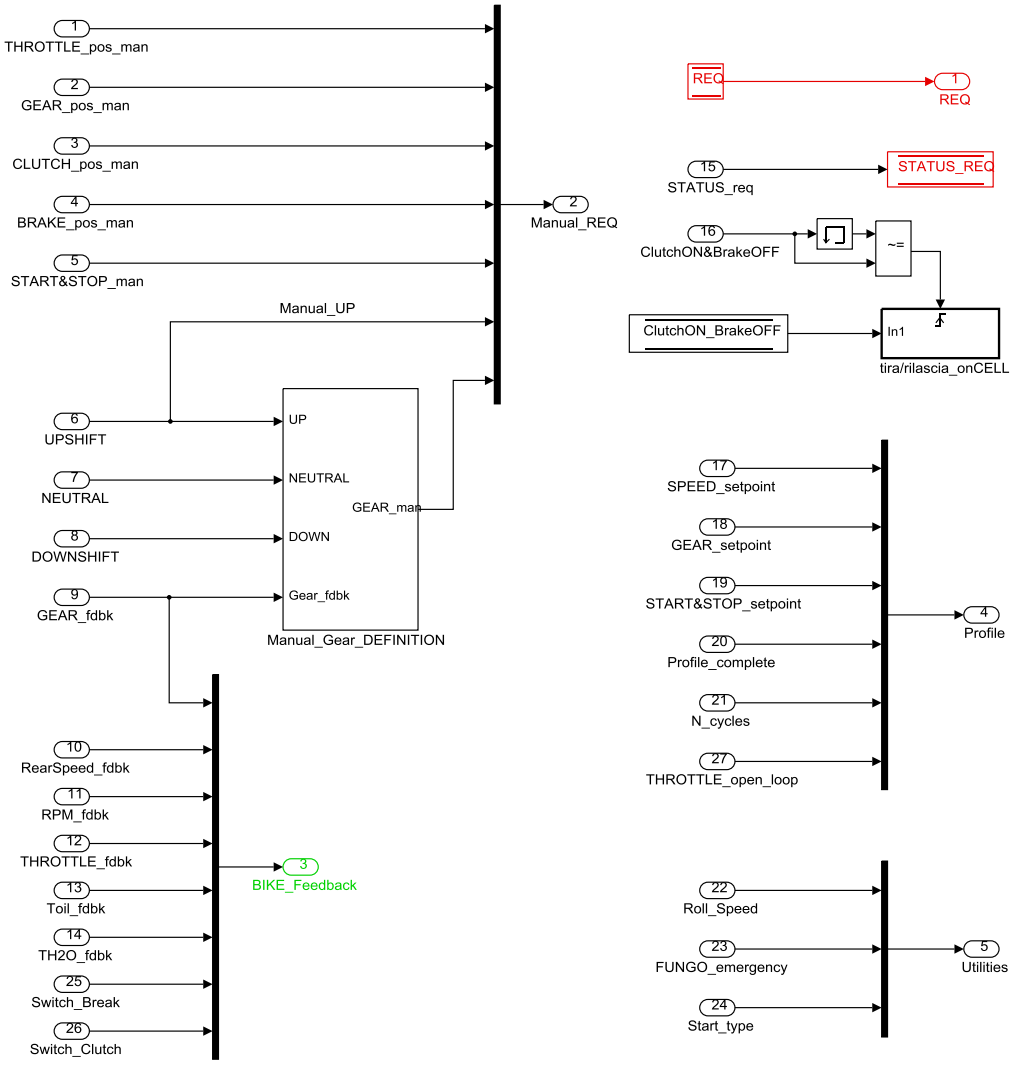


Figure 4.2. RideIT_RT_model /INPUT

- “Manual_REQ” è un vettore contenente le grandezze riguardanti il funzionamento della moto, ognuna delle quali può essere definita all’interno del progetto LabView Real-Time.
- “BIKE_Feedback” è anch’esso un vettore contenente informazioni in gran parte lette da centralina, quali marcia attuale, velocità del motore e della ruota posteriore, temperatura di olio e acqua, apertura acceleratore. Inoltre, nello stesso vettore sono contenute le informazioni che stabiliscono, in un caso se la frizione sia tirata o meno, nell’altro se il freno sia premuto o meno.
- “Profile” è un vettore contenete le informazioni utili, nonché i *set point* per eseguire il ciclo di guida in stato automatico.
- “REQ” (cella di memoria,) è un vettore che contiene i valori degli elementi richiesti per l’output; sono tutte grandezze che cambiano secondo lo stato di funzionamento e rappresentano i target.
- “STATUS_req” è una cella di memoria che rileva da LabView lo stato target, e lo mette a disposizione del modello.
- “Tira/rilascia_onCELL” (sottosistema) contiene la cella di memoria della variabile booleana “ClutchON_BrakeOFF”, il cui valore viene modificato ogniqualvolta viene rilevata la transizione sul segnale di ingresso proveniente da LabView.
- “Utilities” è un vettore che contiene informazioni che si riferiscono al banco a rulli, al tipo di partenza e se il pulsante di allarme (“fungo”) è premuto o meno.

Per quanto riguarda il sottosistema *Manual_Gear_DEFINITION*, questo serve a stabilire la marcia richiesta dall’operatore in stato di guida manuale.

4.3 Status Definition

In questa parte del modello lo scopo è quello di elaborare in quale stato si troverà il sistema, in conformità a principi logici che dipendono dallo stato attuale, dallo stato richiesto e da alcune variabili.

Come già è stato detto nel capitolo 2, si è scelto di definire 6 differenti stati, che rappresentano differenti condizioni della moto, ognuno dei quali è rappresentato nel modello da un numero:

Tabella 4.1. Rappresentazione numerica degli Stati

CALIBRATION	0
MANUAL	1
SEMI-AUTO	2
AUTOMATIC	3
IDLE	10
ALARM	11

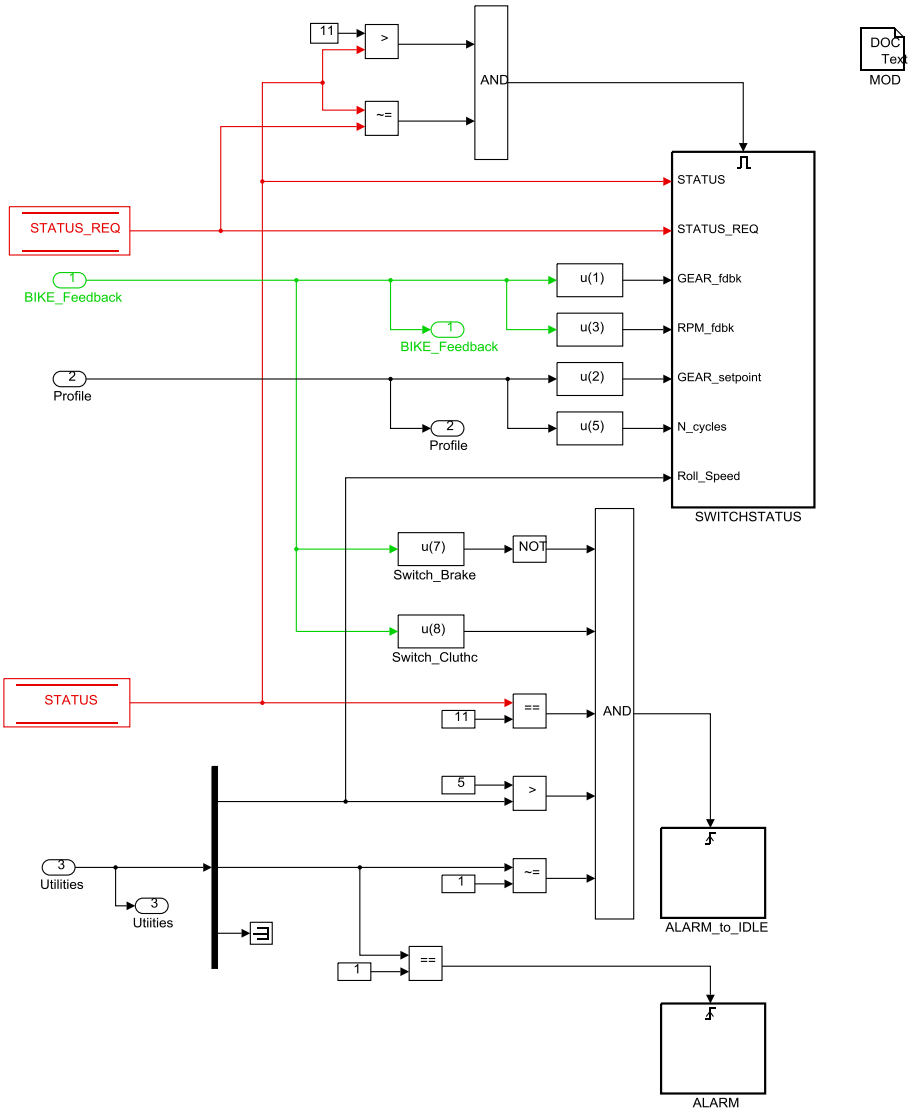


Figure 4.3 RideIT_RT_model/STATUS_DEFINITION

La determinazione dello stato è elaborata in Simulink nel modo seguente: viene letta la cella di memoria “STATUS”, che rappresenta lo stato in cui la moto si trova nello step di calcolo attuale; si richiamano dagli input lo stato richiesto, il vettore “BIKE_Feedback” che mette a disposizione la marcia inserita, la velocità di rotazione del motore e i segnali di switch di freno e frizione, inoltre viene letto il numero di riproduzioni del profilo di guida in stato automatico e il set point di marcia, elementi del vettore "BIKE_fdbk". In ultimo vengono estratti la velocità del rullo e il segnale dato dal pulsante di emergenza dal vettore “Utilities”.
Per acconsentire, o meno, lo switch fra i vari stati si tiene in considerazione che lo stato attuale sia diverso da quello di allarme. Il modello esegue poi determinati sotto blocchi in conformità alle differenti condizioni. Esaminiamoli nel dettaglio.

ALARM: questo sottosistema è contraddistinto dalla presenza di un “trigger”, perciò fa sì che l’istruzione contenuta dentro al blocco sia eseguita solo al primo step di calcolo in cui risulta essere verificata la condizione di attivazione, che in questo caso è relativa al segnale booleano “Fungo _emergency”, ossia la richiesta di allarme da parte dall’utente. Quando questo blocco viene eseguito, il sistema impone lo stato come “11”, cioè stato di allarme.

ALARM_to_IDLE: è anch’esso un sottosistema contraddistinto dalla presenza di un “trigger” e l’istruzione contenuta dentro al blocco viene eseguita solo quando risultano essere verificate contemporaneamente le condizioni poste in ingresso al blocco operatore AND. Infatti se, c’è uno stato di allarme, il pulsante di emergenza è rilasciato, la velocità del rullo trasformata in Km/h è inferiore a 5 e nel motoveicolo c’è la frizione tirata e il freno rilasciato, allora con tali requisiti è possibile uscire dallo stato di allarme, quindi, al verificarsi di tale situazione è consentito il passaggio allo stato IDLE.

SWITCHSTATUS: questa parte di modello è attiva quando lo stato non è di allarme e lo stato richiesto è diverso dallo stato attuale.
Un ciclo “if” determina lo stato in cui ci si può effettivamente portare e come verrà spiegato di seguito, ci sono molteplici le condizioni che devono essere rispettate nella transizione tra stati.

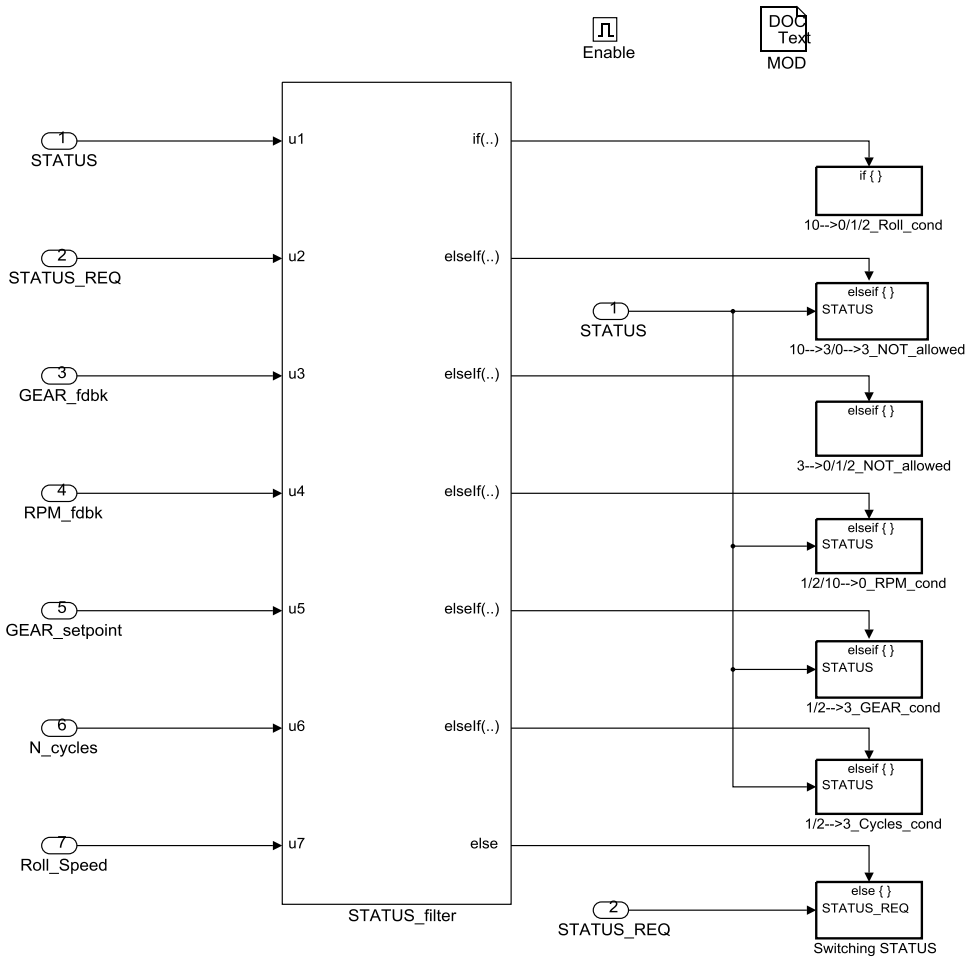


Figure 4.4 RideIT_RT_model/STATUS_DEFINITION /SWITCHSTATUS

- Se lo stato attuale è Idle (u1=10), e lo stato richiesto è uno tra Calibration (u2=0), o Manual (u2=1), o Semi-Auto (u2=2), e se il rullo è in movimento (u7 > 5*Km/h*), allora si continua a lasciare il sistema in Idle, in quanto il passaggio di stato è consentito solo avendo il rullo fermo. Il valore della cella "STATUS_REQ" diviene “10”, in tale modo non si acconsentono ulteriori switch tra stati, a meno che il sistema non venga riportato in condizioni sicure e venga rinnovata la richiesta da parte dell’utente.

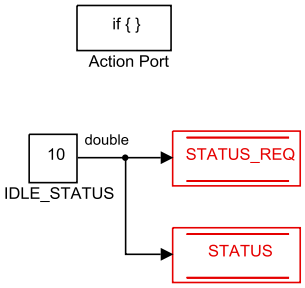


Figure 4.5 RideIT_RT_model/STATUS_DEFINITION/SWITCHSTATUS

10→ 0/1/2 Roll_cond

- Se lo stato attuale è Idle (u1=10), o Calibration (u1=0) e lo stato richiesto è automatico (u2=3), la richiesta non può essere soddisfatta, pertanto non vi è alcun cambiamento di stato.

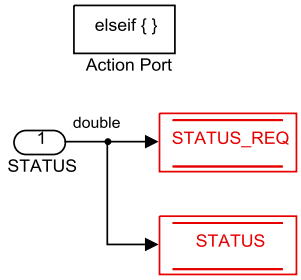


Figure 4.6 RideIT_RT_model/STATUS_DEFINITION/SWITCHSTATUS

10→3/ 0→3 NOT_allowed

- Se lo stato attuale è automatico (u1=3), e lo stato richiesto è uno tra Calibration (u2=0), o Manual (u2=1), o Semi-Auto (u2=2), la richiesta non può essere soddisfatta e s'impone lo stato di Idle non permettendo ulteriori cambiamenti di stato.

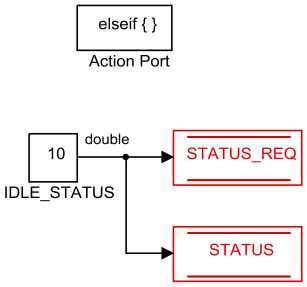


Figure 4.7 RideIT_RT_model/STATUS_DEFINITION/SWITCHSTATUS

3→ 0/1/2 NOT_allowed

- Se lo stato attuale è manuale (u1=1), o semiautomatico (u1=2), oppure è idle (u1=10) e lo stato richiesto è Calibration (u2=0) ma la moto non è spenta, la richiesta non può essere soddisfatta perciò si mantiene il sistema allo stato attuale.

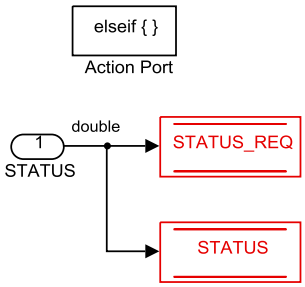


Figure 4.8 RideIT_RT_model/STATUS_DEFINITION/SWITCHSTATUS 1/2/10 → 0 RPM_cond

- Se nel passaggio dallo stato manuale (u1=1) ad automatico (u2=3), non viene rispettata la condizione di coincidenza tra la marcia letta dalla centralina e la prima richiesta di marcia del profilo di guida, il sistema permane in STATUS "1". Stessa condizione vale se si vuole passare dallo stato semiautomatico (u1=2) ad automatico. In quel caso il sistema permane in STATUS "2".

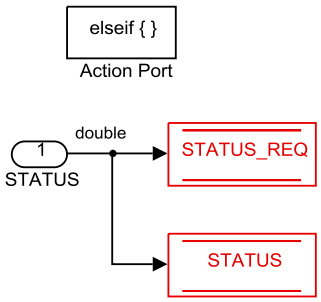


Figure 4.9 RideIT_RT_model/STATUS_DEFINITION/SWITCHSTATUS 1/2→ 3 GEAR_cond

- La transizione da manuale ad automatico, o da semi-auto ad automatico non è consentita neppure se l'operatore non ha stabilito il numero di ripetizioni del profilo da riprodurre (u6=0).

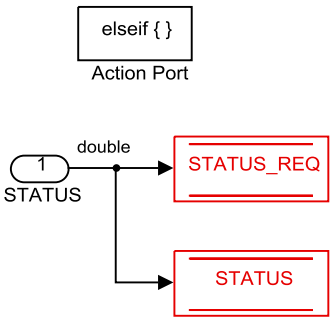


Figure 4.10 RideIT_RT_model/STATUS_DEFINITION/SWITCHSTATUS 1/2→3 Cycles_cond

- Nel caso in cui tutte le condizioni del blocco IF che regolano lo switch tra stati siano soddisfatte, le richieste possono essere acconsentite. Per evitare che si richiedano stati non previsti si satura l'input di stato richiesto tra 0 e 3.

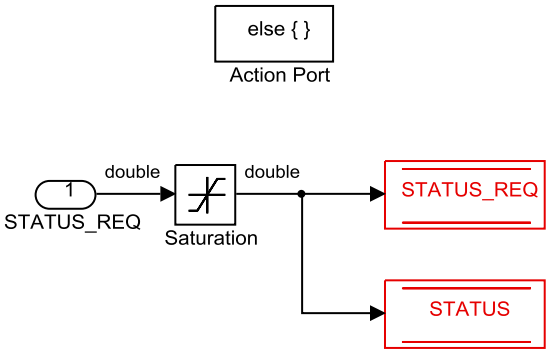


Figure 4.11 RideIT_RT_model/STATUS_DEFINITION/SWITCHSTATUS/Switching_STATUS

La logica di sistema appena descritta per il blocco *STATUS_DEFINITION*, che rappresenta le diverse modalità operative del sistema stesso, è rappresentabile anche attraverso il diagramma di flusso riportato in figura 4.12. Tale diagramma fa una panoramica di ciò che è stato implementato all'interno del blocco *STATUS_DEFINITION* e che si basa su un modello di **macchina a stati**. Come si può notare dalla figura 4.12, in primo luogo sono riportati gli stati e i passaggi in ciascuno di essi. Per quanto riguarda le transizioni di stato, vengono indicate solo quelle attive nonché consentite, riportando anche in dettaglio gli eventi e le condizioni necessarie affinché queste possano essere eseguite. Grazie al diagramma dunque, è più immediato visualizzare il comportamento del sistema durante una simulazione e aiuta a rendersi conto di quel che sta accadendo nel sistema se, ad

esempio, non viene soddisfatta una qualche particolare condizione e avviene magari un passaggio forzato di stato, oppure se viene effettuata una richiesta di cambiamento di stato che non è coerente con la logica implementata.

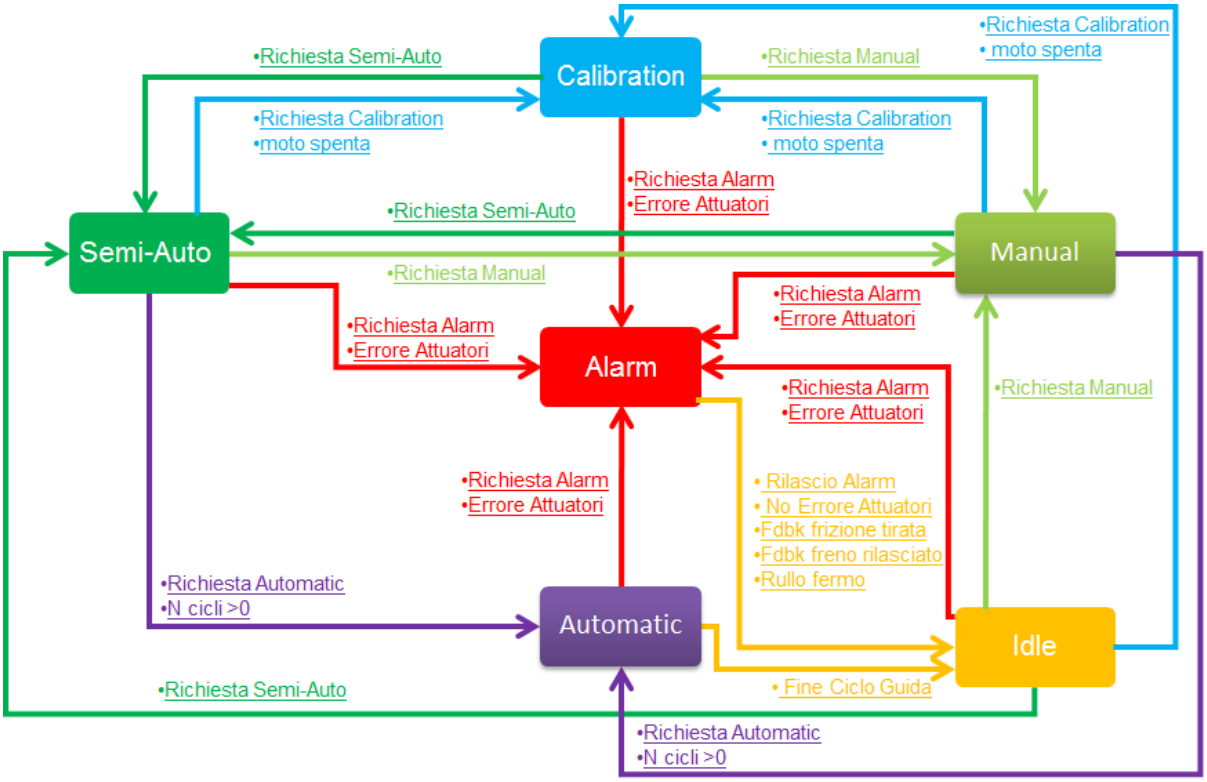


Figure 4.12 State Machine

4.4 Status Action

In questa parte di modello sono definite le azioni da compiere sulle variabili in input secondo lo stato in cui ci si trova, per poi scrivere le grandezze sulla cella di memoria "REQ" che rappresenta i target. Com'è logico immaginare, sono presenti sei sottosistemi ognuno dei quali è attivo quando ci si trova in una differente condizione.

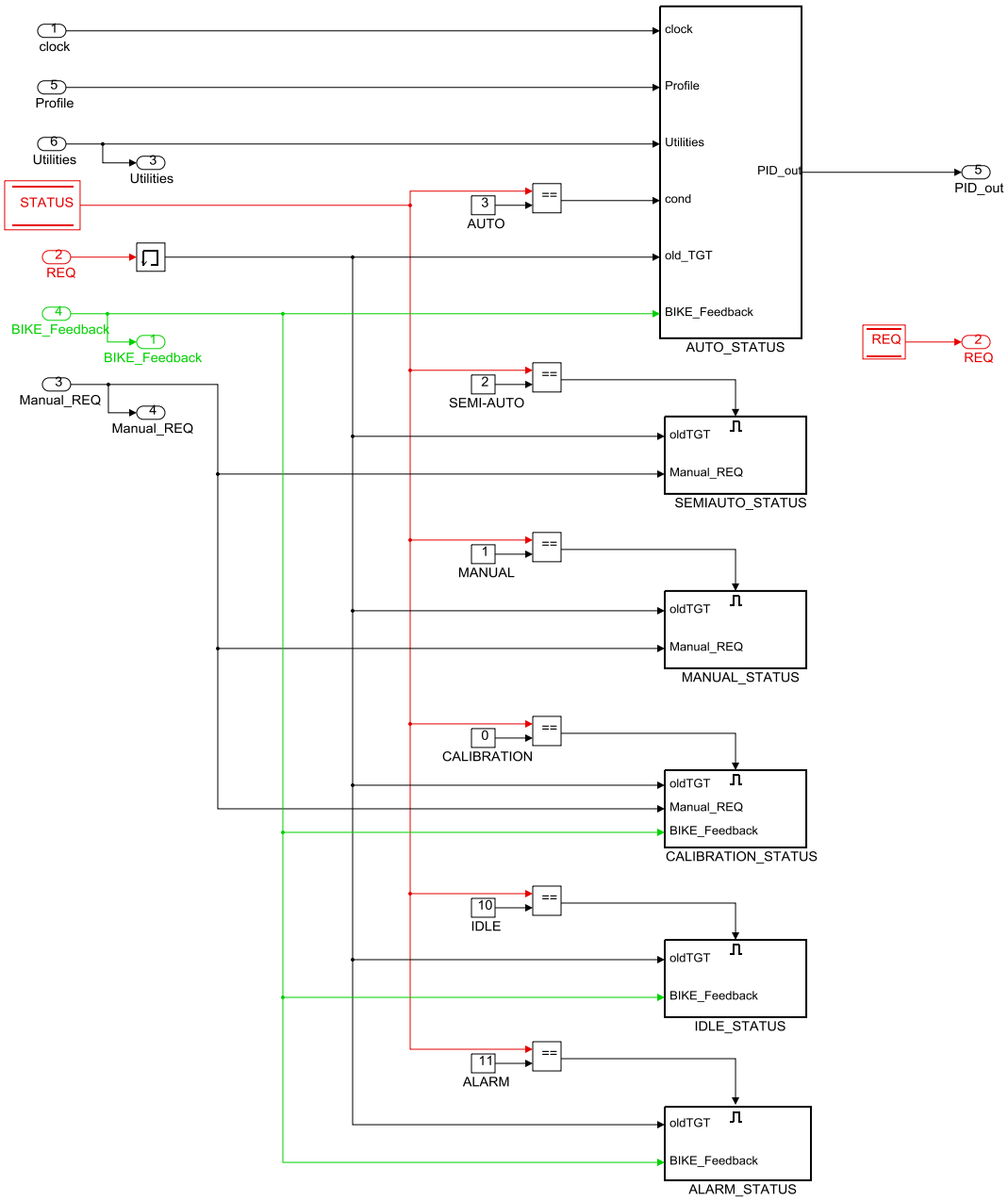


Figure 4.13 RideIT_RT_model/STATUS_ACTION

ALARM_STATUS: se lo stato è di allarme (11) il modello impone come "REQ" i valori target dello step precedente di calcolo con la differenza di imporre una chiusura completa dell'acceleratore (THROTTLE = -1) ed uno spegnimento del motore del motoveicolo (BIKE_turn_OFF). Non vi è alcun target di velocità da far rispettare al motoveicolo (Speed_TGT = 0), tanto meno una marcia target da inserire, perciò viene semplicemente scritto nel vettore "REQ" il valore della marcia attuale

rilevata dalla centralina. Inoltre, non appena viene rilevata la posizione di farfalla chiusa (THROTTLE_fdbk < 10%) s'impone di tirare completamente la frizione e rilasciare completamente il freno (ClutchON_BrakeOFF = 1).

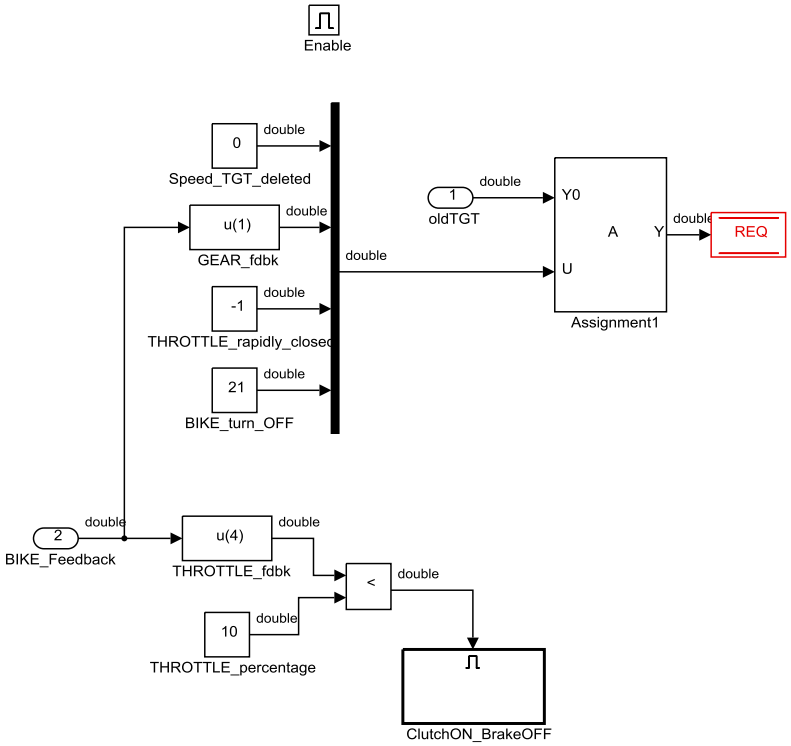


Figure 4.14 RideIT_RT_model/STATUS_ACTION/ALARM_STATUS

IDLE_STATUS: se lo stato è Idle (10) si scrivono in "REQ" i suoi valori allo step di calcolo precedente, imponendo la chiusura della manopola (THROTTLE=0) e sostituendo la marcia da inserire con quella rilevata dalla ECU. Inoltre non deve esserci alcuna richiesta di velocità per il motoveicolo (Speed_TGT=0).
Anche in questo caso è presente una logica per l'attivazione di "ClutchON_BrakeOFF" non appena viene rilevata la chiusura della farfalla (THROTTLE_fdbk < 10%).

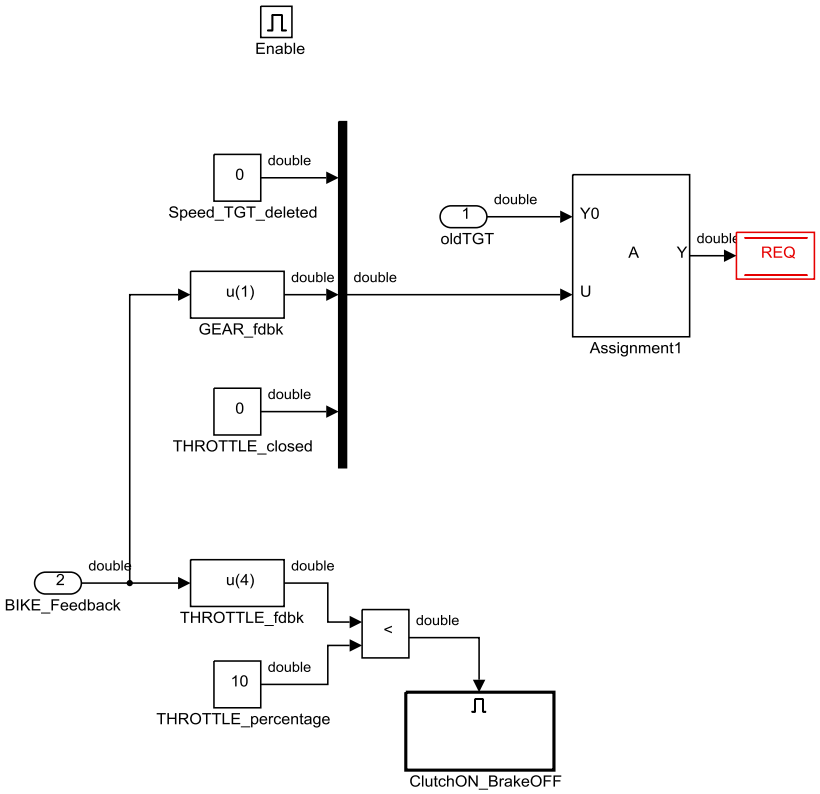


Figure 4.15 RideIT_RT_model/STATUS_ACTION/IDLE_STATUS

CALIBRATION_STATUS: quando lo stato è di Calibrazione (0) si scrivono su "REQ" i suoi valori allo step di calcolo precedente, imponendo un target di velocità nulla per il motoveicolo e un target di marcia invariato rispetto a quello rilevato dalla centralina. Inoltre viene imposta la rapida chiusura della manopola acceleratore e lo spegnimento del motore.

In stato di Calibrazione è possibile fare le regolazioni delle corse degli attuatori di freno, frizione e cambio, pertanto viene richiamato il vettore "Manual_REQ" attraverso il quale si aggiornano i valori di "GEAR_position", "CLUTCH_position" e "BRAKE_position".

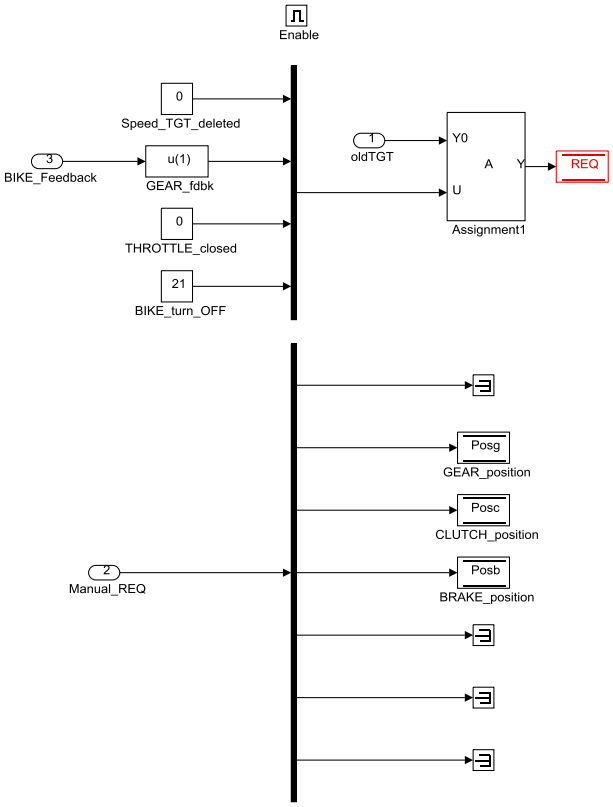


Figure 4.16 RideIT_RT_model/STATUS_ACTION/CALIBRATION_STATUS

MANUAL_STATUS: se lo stato è manuale (1), si gestiscono tutti i target con valori che possono essere modificati dagli input, ad eccezione del parametro di velocità a cui si vuol portare la ruota posteriore del motoveicolo, poiché non è un parametro da imporre via software quando si gestisce manualmente il comando dell'acceleratore (Speed_TGT = 0).

E' quindi richiamato il vettore "Manual_REQ", da cui si estraggono i contributi di "Gear", "Throttle", "Start&Stop" da imporre come target. Si aggiorna inoltre la posizione del freno.

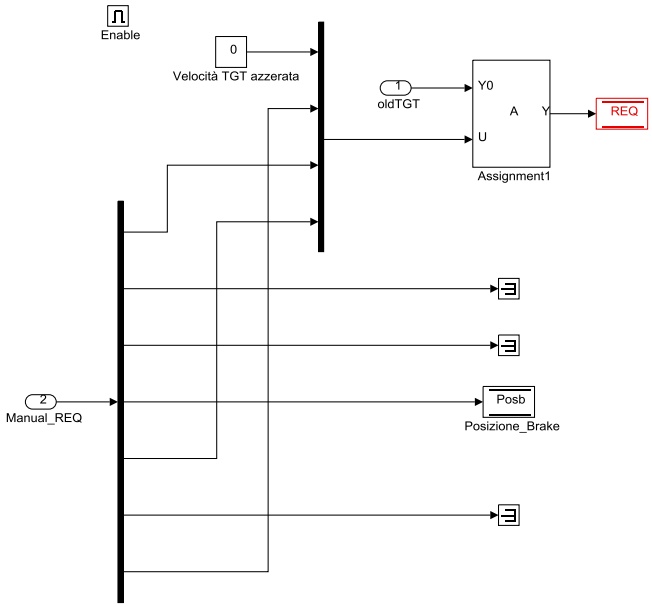


Figure 4.17 RideIT_RT_model/STATUS_ACTION/MANUAL_STATUS

SEMIAUTO_STATUS: se lo stato è semiautomatico (2), valgono le stesse considerazioni affrontate per lo stato manuale e così anche risulta essere identica la parte di codice contenuta dentro al sottosistema.

AUTO_STATUS: se lo stato è automatico (3), l'obiettivo è replicare al banco a rulli un ciclo di guida totalmente in automatico, inseguendo dei parametri target che possono essere impostati manualmente, oppure riportati da un file Excel, per poi essere caricati nel codice, avendo a disposizione una vasta gamma di strumenti nell'interfaccia Host. Utilizzando tutti i dati del profilo impostato sui comandi del pilota, non necessariamente si ottiene effettivamente il medesimo comportamento della moto, per molteplici fattori quali ad esempio una dinamica differente, differenti prestazioni del motore, diverso slittamento tra ruota e rullo rispetto a ruota e asfalto. All'interno di questo blocco sono presenti quattro sottosistemi: "End_profile" "Transient_Definition", "PID_THROTTLE" e "Write_on_REQ".

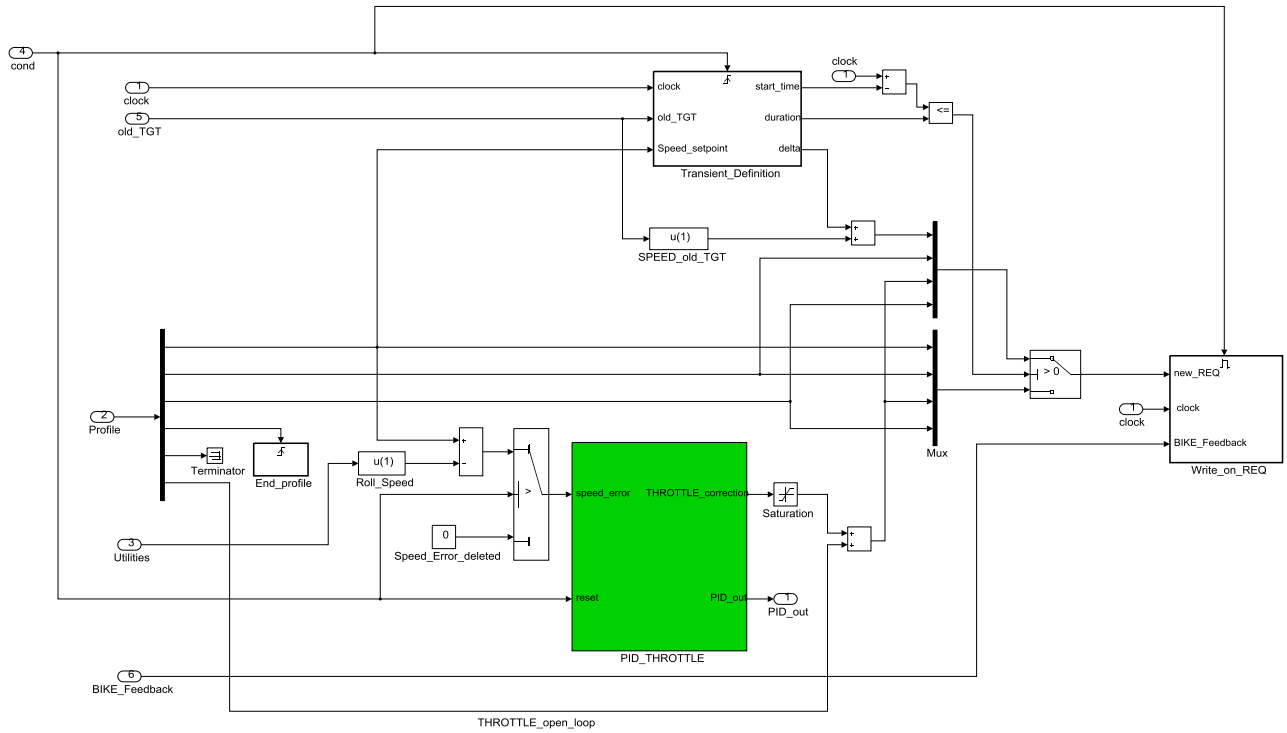


Figure 4.18 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS

End_profile: è all'interno di questo sottosistema che la condizione di fine riproduzione del profilo si concretizza in un'uscita dallo stato AUTO. Infatti, una volta che il digitale "Profile_complete" passa a "true", si va a sovrascrivere il valore "10" sulle celle di memoria di "STATUS" e "STATUS_REQ", decretando così il passaggio automatico in Idle.

Transient_Definition: all'interno di tale blocco, che si attiva ogni qual volta si rileva il passaggio allo stato automatico, oltre a memorizzare l'istante di tempo in cui avviene la transizione di stato ("start_time"), s'importa la velocità di set point del motoveicolo proveniente dal vettore delle grandezze che definisce tutti i set point del profilo di guida su base tempo (vettore "Profile"). Dal vettore "REQ" si richiama il valore di velocità target allo step di calcolo precedente (SPEED_old_TGT). Quest'ultima è una grandezza soggetta a transitorio durante la rampa antecedente la riproduzione del profilo. Inoltre dal file d'inizializzazione (si veda l'appendice B) si richiamano la "rampTOauto_duration" (durata della rampa) e lo "step_time". Attraverso semplici operazioni matematiche si arriva al "delta" ricercato, cioè al

valore di cui bisogna incrementare, ad ogni step di calcolo, la velocità target per portarla al primo valore richiesto dal profilo.

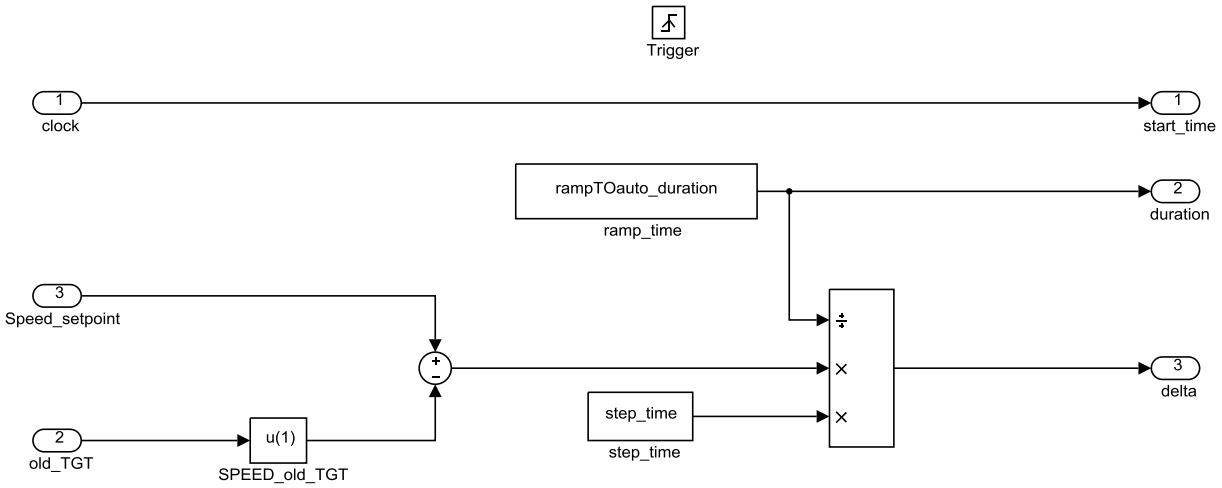


Figure 4.19 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Transient_Definition

PID_THROTTLE: uno degli input di questo blocco è lo "speed_error" calcolato come differenza tra la velocità del rullo e il relativo target stabilito dai campioni del ciclo di guida automatico.

Lo "speed_error", diverso da zero solo in stato AUTO ("cond" è unitaria sia in rampa, che in simulazione), è utilizzato all'interno di questo blocco in cui si ricerca la correzione da dare al segnale di "THROTTLE_open_loop" per far sì che l'errore sulla velocità sia minimo. La correzione di posizione acceleratore è calcolata con un controllore proporzionale integrale derivativo con coefficienti da testare al banco. Tale correzione, come si può vedere all'esterno del sottosistema, viene saturata per impedire che si superino le soglie, minima e massima, previste per l'acceleratore, per questioni di sicurezza.

Altro input di questo sottosistema è il segnale che alimenta lo switch dello "speed error" che funge anche da reset per l'integrale.

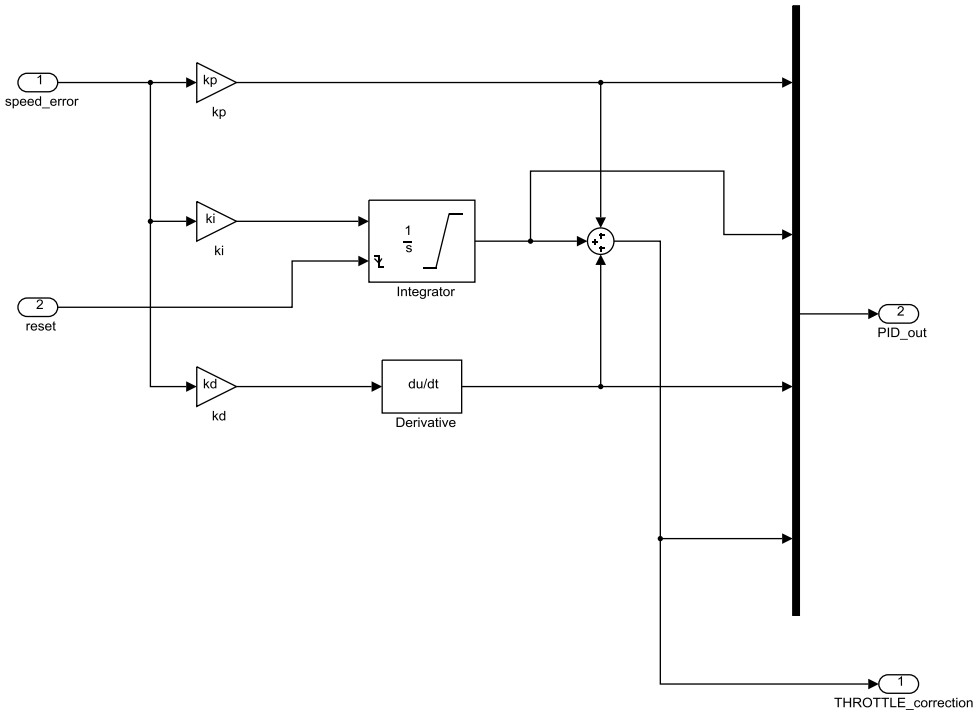


Figure 4.20 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/PID_THROTTLE

Write_on_REQ: all'interno di tale blocco, che si attiva in stato AUTO, si definisce il vettore dei target "REQ", a seconda che ci si trovi o meno nel transitorio iniziale. Infatti, prima di entrare nel sottosistema, l'operazione che viene eseguita a monte è quella di calcolare il tempo trascorso da quando la gestione della moto è passata ad automatica, sottraendo il valore "start_time" (che rappresenta appunto l'istante in cui è avvenuto il passaggio alla modalità automatica) al tempo attuale della simulazione ("clock"). Se tale differenza è minore o uguale alla durata del transitorio ("duration") sono utilizzati i valori di set point di "GEAR" e "Start&Stop" dati dal vettore "Profile", mentre per la posizione della manopola e la velocità, si usano i valori che variano linearmente durante il transitorio, a partire dal loro valore nell'istante in cui è attivata la modalità automatica, fino al primo valore che essi hanno nel profilo. Nell'arco del transitorio a ogni step time si determina il target di velocità sommando il "delta" al valore del target di velocità del passo di calcolo precedente ("SPEED_old_TGT"); ciò permette di portare l'apertura della manopola e di conseguenza la velocità ad avere il valore richiesto a inizio ciclo.

Raggiunta e superata la durata del transitorio, si continua a correggere, tramite il PID, la posizione della manopola acceleratore, al fine di inseguire un target di velocità imposto a ogni step time. La differenza rispetto al caso del transitorio di rampa, è che nel "REQ" il target di velocità in questo caso viene richiamato dal vettore "Profile", così come per gli altri set point.
Nel sottosistema *Write_on_REQ* sono inoltre presenti due blocchi che svolgono azioni ben determinate ai fini della sicurezza del sistema: *SAFETY_ENGINE* e *GEAR_fdbk_WRONG*. Esaminiamoli nel dettaglio.

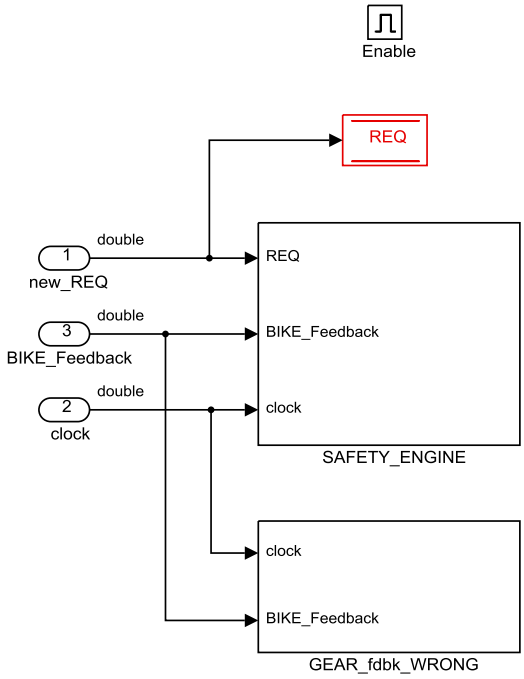


Figure 4.21 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Write_on_REQ

SAFETY_ENGINE: durante i test di simulazione del ciclo automatico può accadere di portare la moto a regimi tanto elevati da comportare l'entrata in funzione dei limitatori per tempi anche relativamente prolungati, minando l'integrità del motore. Per evitare di provocare danni si è pensato di elaborare una strategia, appunto di safety. La procedura, implementata in questo blocco, prevede che, nel caso in cui il motore raggiunga e permanga a limitatore per un certo lasso di tempo, si chiuda la farfalla di un valore riportato in "dec_THROTTLE" (scritto nel file di lancio) per portare ad un repentino calo dei giri motore.

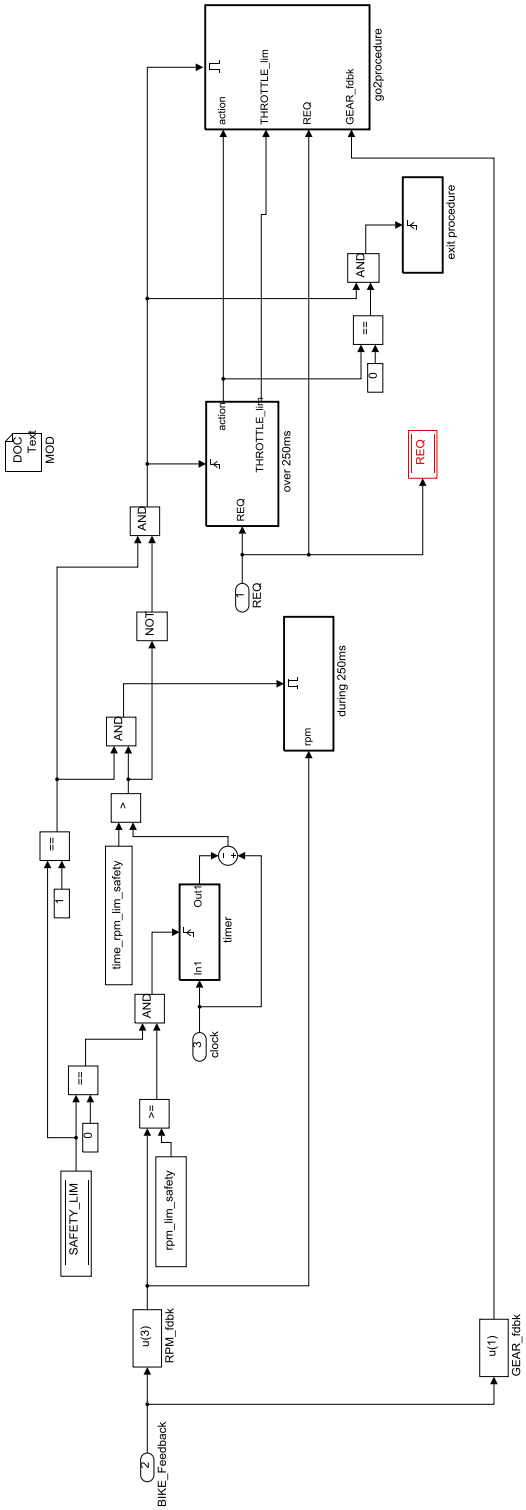


Figure 4.22 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Write_on_REQ/ SAFETY_ENGINE

Più nello specifico, la strategia di safety analizza i giri motore e se questi salgono sopra un valore di soglia, stabilito nel file di inizializzazione (“rpm_lim_safety”), viene innescato il blocco "timer" che memorizza il tempo in cui l'evento si è verificato e si arma la procedura di decremento dei giri, scrivendo “1” all'interno della cella di memoria "SAFETY_LIM".

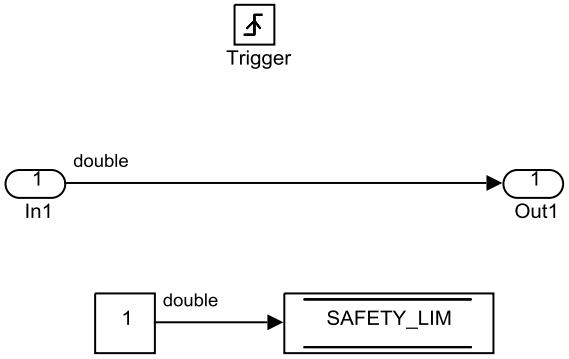


Figure 4.23 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Write_on_REQ/ SAFETY_ENGINE/timer

Dunque, nell’arco dei *250ms* a seguire, si continuano ad esaminare i giri nel blocco during_250ms e se in questo intervallo di tempo gli RPM calano al di sotto del valore di soglia decrementato di un certo “deltarpm” (riportato nel file di lancio), allora si disarma la procedura scrivendo “0” su "SAFETY_LIM".

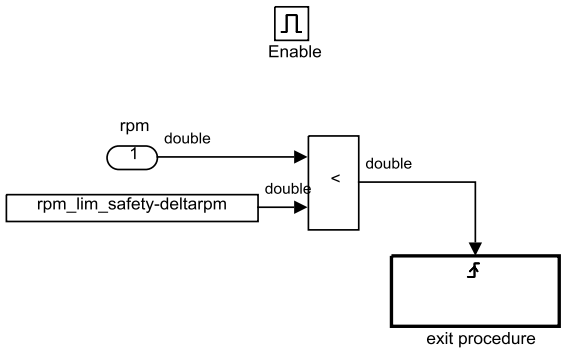


Figure 4.24 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Write_on_REQ/ SAFETY_ENGINE/during 250ms

Nel caso contrario, essendo trascorsi più di *250ms* con procedura ancora attiva, viene innescato il blocco "over_250ms" dove si decide se intervenire con la strategia di decremento giri.

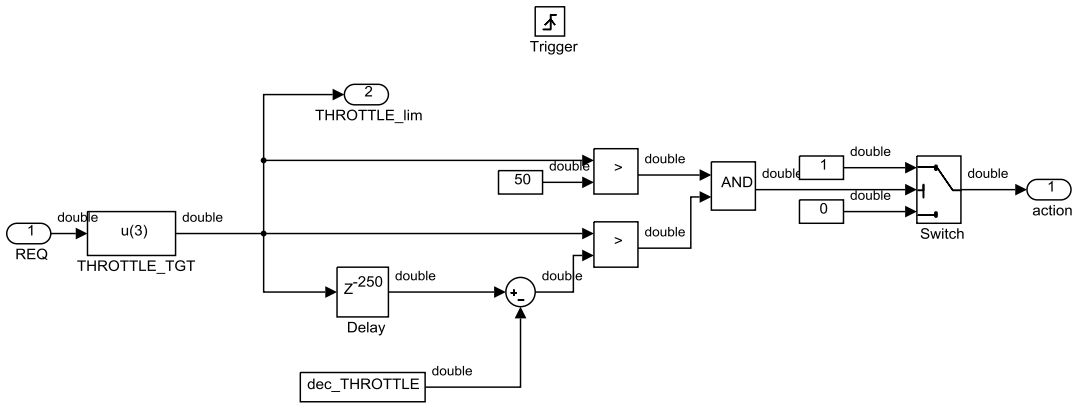


Figure 4.25 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Write_on_REQ/ SAFETY_ENGINE/over 250ms

All'interno di questo blocco viene richiamata la posizione di acceleratore richiesta, dal vettore "REQ".

Nel momento in cui questo blocco è stato attivato viene riscontrata una richiesta di far calare i giri il prima possibile, e per raggiungere tale scopo si provvede a decrementare la farfalla. Questa è la logica della strategia, però non è detto che i risultati prodotti da tale azione vengano applicati. Infatti, nel sottosistema si svolge l'analisi per capire se sia davvero necessario proseguire con la strategia di decremento dei giri.

Ad esempio, se la farfalla, *250ms* dopo che i giri hanno toccato i limitatori, è già stata ridotta da software (set point di profilo) di un valore superiore al decremento da attuare ("dec_THROTTLE" attualmente pari a 30%) rispetto al valore che essa aveva quando la procedura di safety è stata armata, è inutile portare avanti la procedura, perciò deve essere disarmata. La stessa cosa vale se la farfalla, *250ms* dopo che gli RPM sono finiti oltre la soglia, è divenuta inferiore al 50%.

Dunque, solo nel caso in cui si riscontra che è opportuno proseguire con la strategia, il valore "1" viene scritto nella variabile "action" e la procedura viene poi applicata in "go2procedure". Al contrario viene prodotto "action" = 0 e comporta un'uscita dalla procedura, che viene disarmata in "exit_procedure".

Vediamo nel dettaglio come si compone "go2procedures": il blocco è costituito dal sottosistema *TWGRD* relativo alla strategia di parziale chiusura della farfalla.

Vengono richiamati oltre alla variabile “action” e il vettore “REQ”, la richiesta di marcia da profilo e il valore richiesto di farfalla *250ms* dopo il superamento della soglia da parte dei giri (“THROTTLE_lim”).

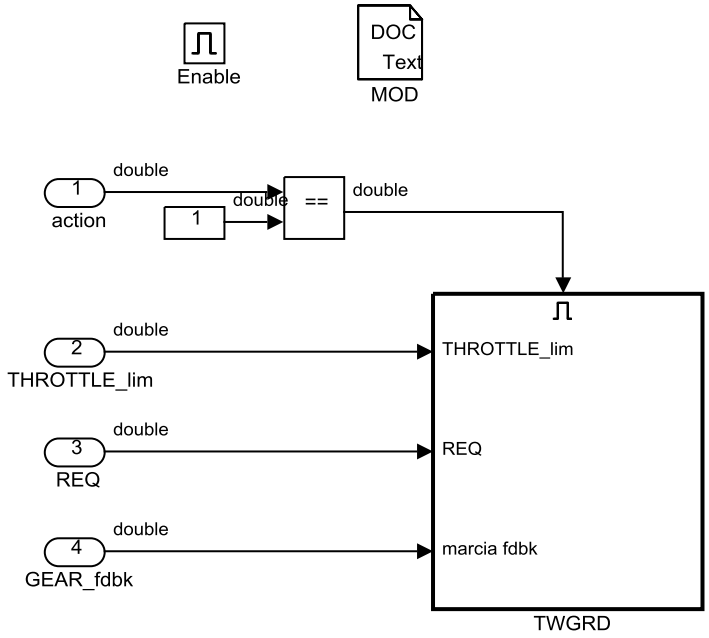


Figure 4.26 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Write_on_REQ/ SAFETY_ENGINE/go2procedure

La riduzione della farfalla viene attuata in “TWGRD”. Si formula una richiesta di farfalla decrementata di “dec_THROTTLE” rispetto al valore “THROTTLE_lim”. Si continua a sovrascrivere il target fintanto che il profilo non richiede un valore di manopola inferiore al valore di farfalla decrementata oppure fino a che non si verifica un cambio marcia. Al verificarsi di uno di questi casi si esce dalla procedura.

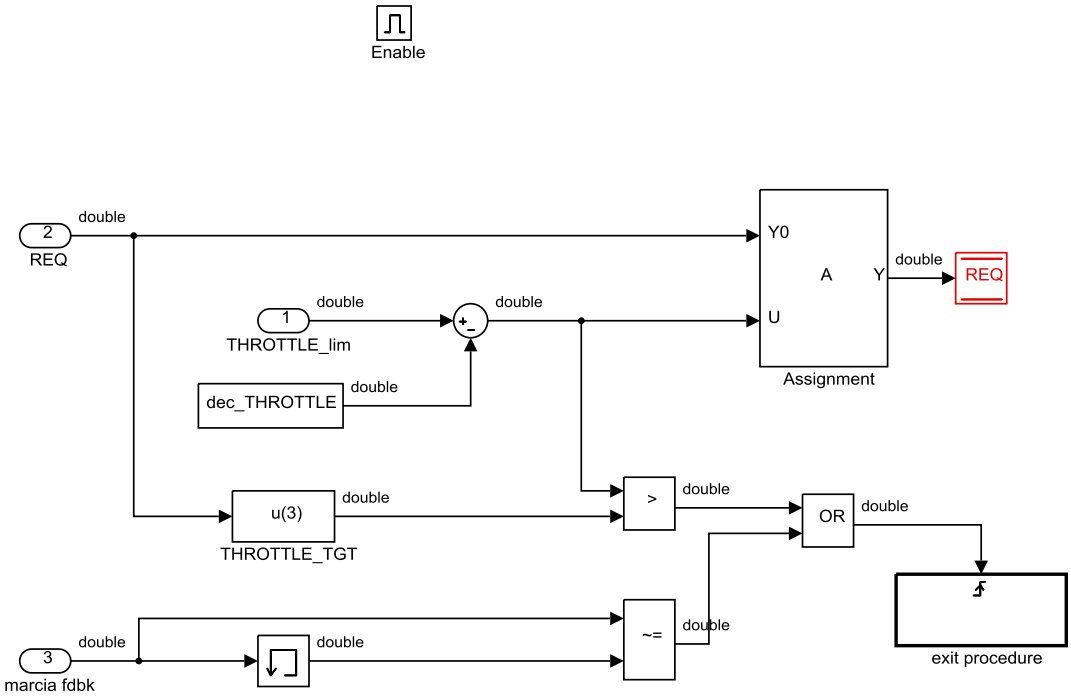


Figure 4.27 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Write_on_REQ/ SAFETY_ENGINE/go2procedure/TWGRD

GEAR_fdbk_WRONG: in questo blocco si forza il sistema ad una transizione verso lo stato ALARM, nel caso in cui si verifichi una condizione critica, ovvero l'incapacità della centralina a determinare la marcia inserita (feedback di marcia pari a -1), durante la simulazione del profilo automatico.

Non è raro che tale condizione si verifichi per un breve lasso di tempo durante la cambiata, poiché potrebbe capitare di inserire un cosiddetto "intermarcia" prima di ingranare la marcia effettivamente desiderata. La situazione diviene grave nel momento in cui tale condizione persiste per un tempo relativamente lungo. Ecco perché, trascorso un secondo dal verificarsi di marcia feedback = -1, si costringe il sistema ad uscire da AUTO, scrivendo "11" sulle celle di memoria di "STATUS" e "STATUS_REQ" all'interno del sottosistema *AUTOtoALARM* opportunamente innescato.

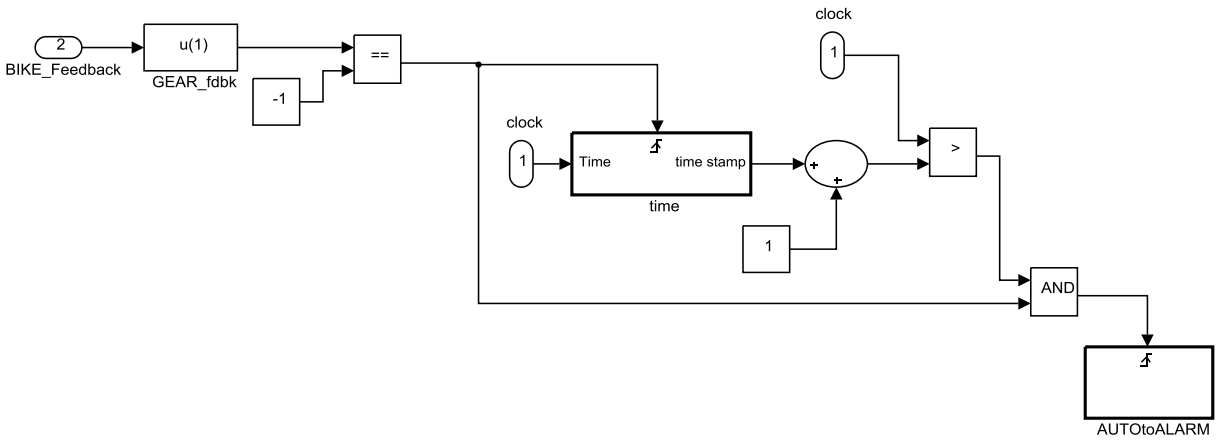


Figure 4.28 RideIT_RT_model/STATUS_ACTION/AUTO_STATUS/Write_on_REQ/ GEAR_fdbk_WRONG

4.5 GEARSHIFT

All'interno di questo blocco è gestita la cambiata.

Le grandezze in input principali sono i vettori “BIKE_Feedback”, “REQ”, “Utilities” e “Manual_REQ”. Il primo contiene informazioni relative a parametri di funzionamento della moto, il secondo è il vettore dei target, il terzo contiene l’informazione relativa al tipo di partenza (“Start_type”) e l’ultimo contiene la richiesta di Upshift manuale (“Manual_UP”). In seguito vengono descritti in dettaglio i sottosistemi costituenti *GEARSHIFT*.

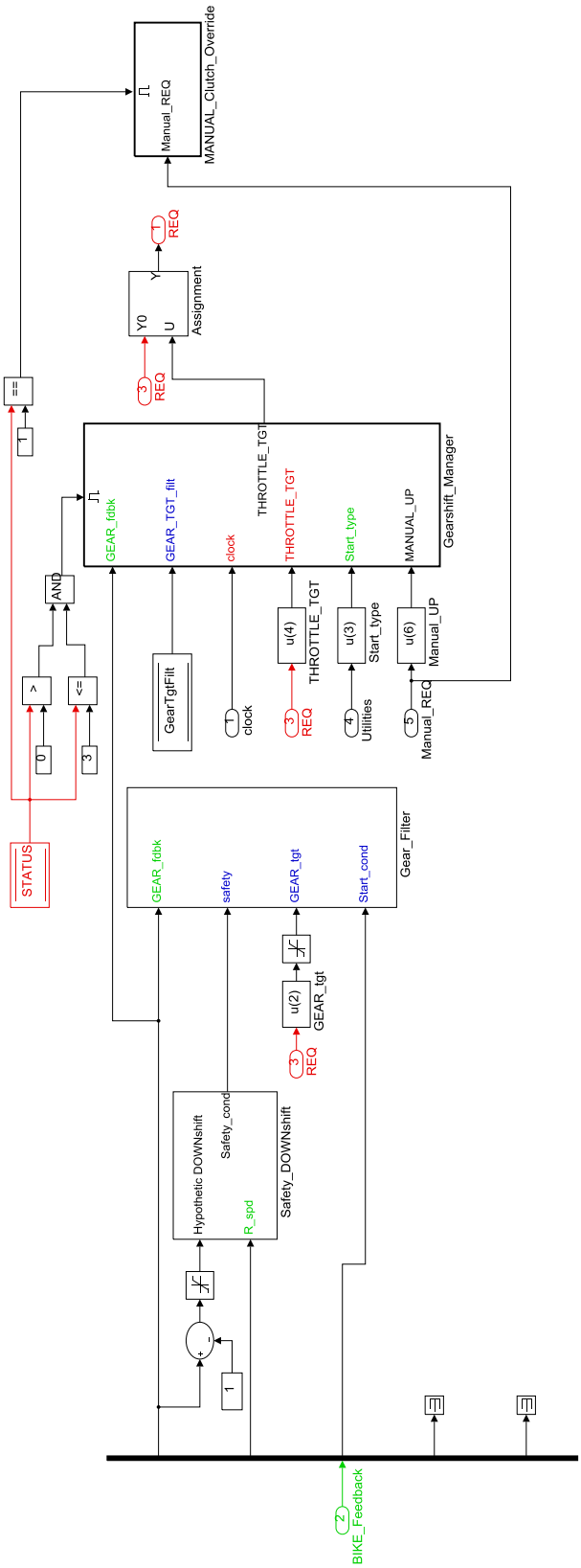


Figure 4.29 RideIT_RT_model/GEARSHIFT

Safety_DOWNshift: questo sottosistema valuta se in caso di scalata il motore rimane sotto una soglia di sicurezza espressa in giri per minuto. Attraverso un Multi-Port Switch è selezionato il rapporto d'ingranamento corrispondente alla marcia che s'inserirebbe con un'ipotetica scalata; questo lo si moltiplica poi per i rapporti di ingranamento costanti della moto (ovvero quello fra il pignone dell'albero motore e la ruota primaria del cambio e quello tra pignone in uscita dal cambio e la corona della ruota posteriore). Si divide poi la velocità della ruota posteriore (convertita in velocità angolare), per il fattore così trovato, in modo da riportarsi ai giri motore equivalenti alla velocità veicolo, nel nuovo rapporto di trasmissione. Se questo valore supera il limite imposto ("rpm_lim"), la scalata non sarà consentita.

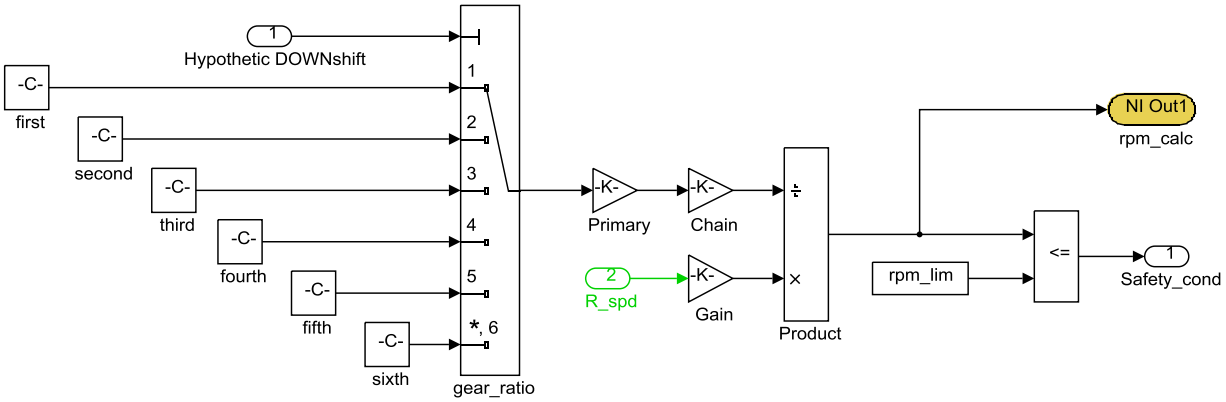


Figure 4.30. RideIT_RT_model/GEARSHIFT/Safety_DOWNshift

Gear_Filter: questo sottosistema ha il compito di valutare "GearTgtFilt", ovvero la marcia che è effettivamente richiesta sulla moto, infatti, perché non è detto che la marcia richiesta dai target possa effettivamente essere inserita dagli attuatori, ad esempio se non sono soddisfatte le condizioni di sicurezza o se la moto si trova in seconda ed è richiesta una quinta.

Questo sottosistema ha come input:

- "GEAR_fdbk", la marcia in cui si trova la moto.
- "GEAR_tgt" la marcia richiesta come obiettivo.
- "Safety", parametro di sicurezza che consente o meno la scalata, prodotto dal sottosistema *Safety_DOWNshift*.
- "Start_cond", vettore che contiene i valori provenienti dalla centralina della temperatura di olio e acqua, apertura manopola e giri per minuto del motore.

Le condizioni che consentono di modificare il valore “GearTgtFilt” sono: in un caso che sia soddisfatta la condizione di safety, nell’altro che la moto sia in prima, e nell’altro ancora, che non si voglia eseguire una scalata (GearTgt > GearFdbk). Nel caso in cui sia soddisfatta almeno una delle suddette condizioni il sistema può aggiornare il valore di “GearTgtFilt”.

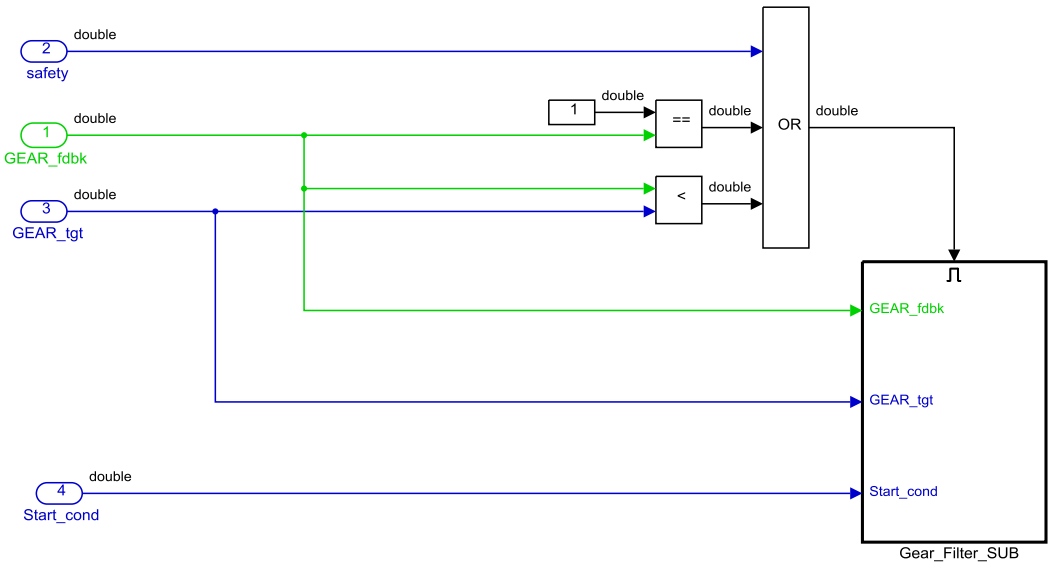


Figure 4.31. RideIT_RT_model/GEARSHIFT/Gear_Filter

In *Gear_Filter_SUB* è presente un sottosistema in cui un ciclo “if” gestisce la logica per la determinazione della marcia da richiedere.

Le variabili in ingresso sono la marcia rilevata dalla centralina (GEAR_fdbk = u1), la marcia richiesta (Gear_tgt = u2) e una variabile (u3) che è pari ad “1” in caso una serie di condizioni, da verificare quando si simula una partenza, siano vere.

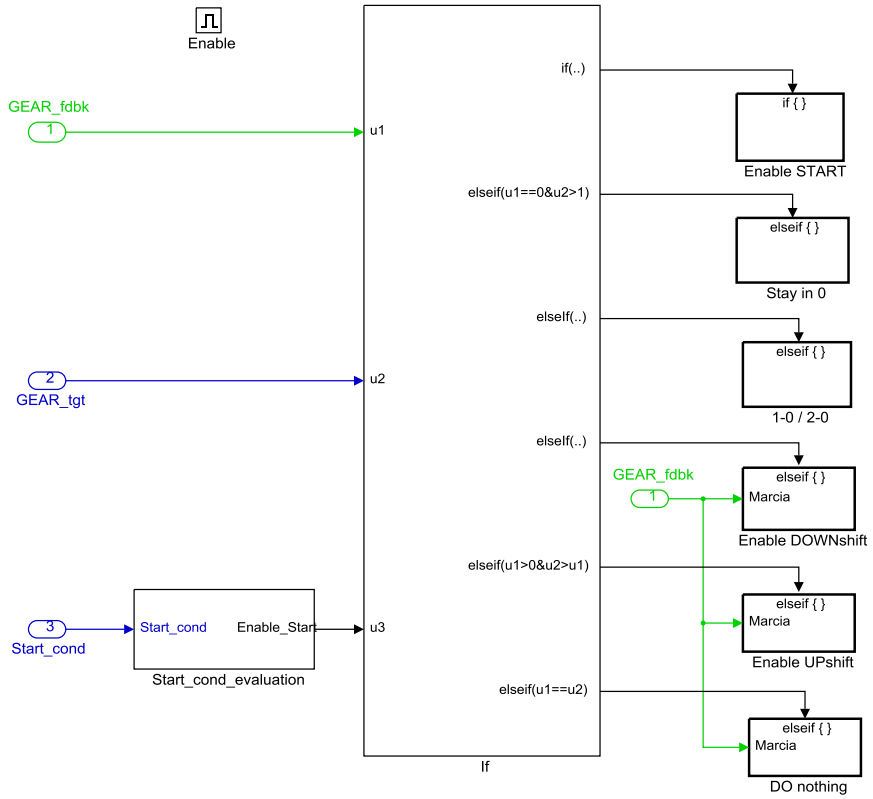


Figure 4.32 RideIT_RT_model/GEARSHIFT/Gear_Filter/Gear_Filter_SUB

In particolare, come riportato all'interno del blocco *Start_cond_evalutation* le temperature dell'olio e dell'acqua devono essere maggiori di determinati valori impostati nel file d'inizializzazione, l'apertura della manopola e i giri motore devono essere minori di valori di soglia massimi, e la condizione di "Start" deve essere attiva.

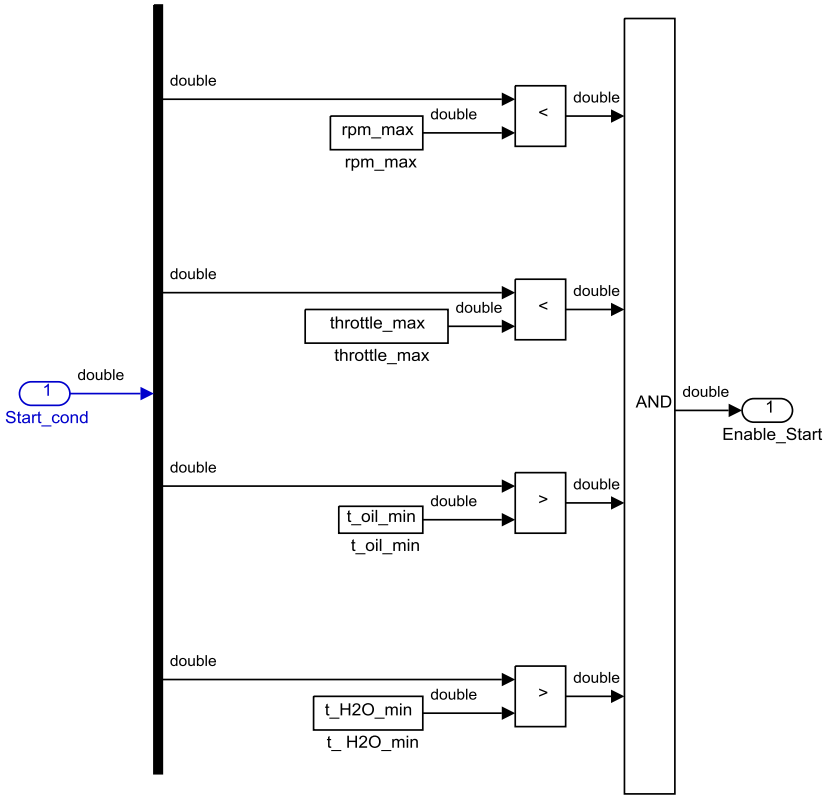


Figure 4.33 RideIT_RT_model/GEARSHIFT/Gear_Filter/Gear_Filter_SUB/ Start_cond_evaluation

In uscita dal ciclo “if” vi sono sei differenti situazioni:

- Se u3=1 significa che è richiesta una simulazione di partenza, e sono verificate le condizioni del blocco "Start_cond_evaluation". Per la partenza, se u1=0, quindi la moto è in folle, s’imposta “GearTgtFilt” in prima se è richiesta la prima (u2=1), altrimenti si resta in folle se è richiesta una marcia superiore alla prima (u2 >1).

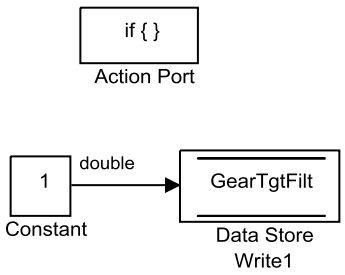


Figure 4.34 RideIT_RT_model/GEARSHIFT/Gear_Filter/Gear_Filter_SUB/ Enable START

- Se u2=0, quindi è richiesta la folle, e ci si trova in prima o in seconda s'imposta "GearTgtFilt" in folle.

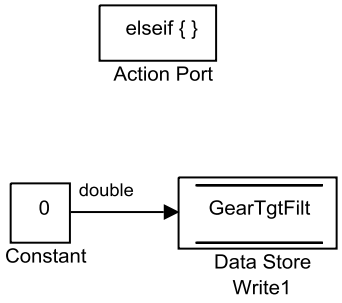


Figure 4.35 RideIT_model/GEARSHIFT/Gear_Filter/Gear_Filter_SUB/ 1→ 0 / 2→ 0

- Se u1>0, u2<u1 e u2>0 o se u2==0 e u1>2, che consiste nella condizione di scalata senza l'obiettivo consentito immediato di folle, s'imposta come "GearTgtFilt" la marcia inferiore rispetto alla marcia inserita.

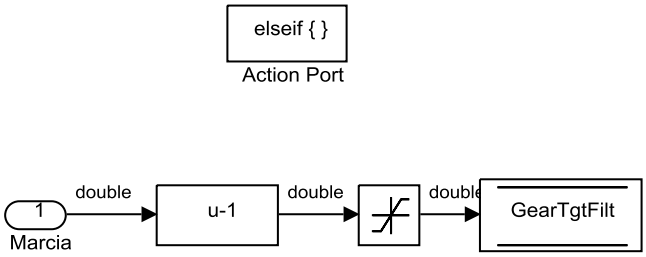


Figure 4.36 RideIT_RT_model/GEARSHIFT/Gear_Filter/Gear_Filter_SUB/Enable DOWNshift

- Se u1>0 e u2>u1, la richiesta è di un upshift, s'imposta quindi come "GearTgtFilt" la marcia superiore a quella inserita.

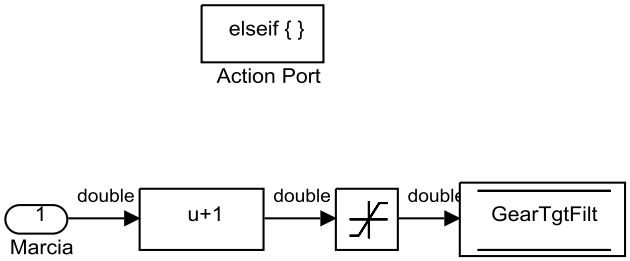


Figure 4.37 RideIT_RT_model/GEARSHIFT/Gear_Filter/Gear_Filter_SUB/Enable UPshift

- Se u1=u2, quindi la marcia richiesta è già inserita, “GearTgtFilt” è la marcia inserita.

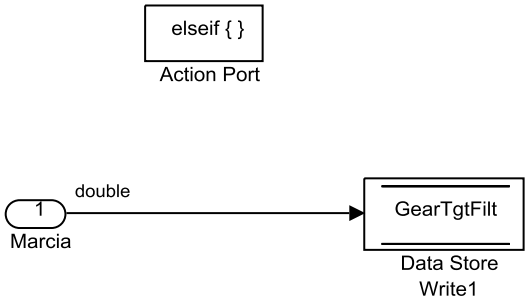


Figure 4.38 RideIT_RT_model/GEARSHIFT/Gear_Filter/Gear_Filter_SUB/DO nothing

Gearshift_Manager: una volta determinata la marcia da richiedere agli attuatori sulla base di condizioni logiche, si passa a questo blocco. Tale blocco si attiva solo in STATUS manuale, o semiautomatico, o automatico.
In input vi sono la marcia attuale (“GEAR_fdbk”), la marcia target consentita (“GearTgtFilt”), l’apertura della manopola richiesta (“THROTTLE_TGT”), il tipo di partenza (Start_type), il tempo di simulazione (“clock”) e una grandezza che assume valore unitario se si vuol richiedere un Upshift manuale nel caso in cui la centralina non riconosca la marcia in cui si trova la moto (“Manual_UP”).
All’interno di *Gearshift_Manager* vi sono una serie di sottosistemi, che consentono di gestire l’attuatore del cambio e della frizione nelle varie situazioni.
Da notare che in ingresso a tali sottosistemi, il cui scopo è quello di provare a far ingranare una marcia, vi è un parametro “GSen” (Gear Shift enable), ritardato di una quantità di tempo pari a “gs_ENdel” (gear shift ENABLE delay, il cui valore è assegnato nel file di lancio), tale che se il segnale ha valore unitario è consentita la movimentazione degli attuatori, altrimenti no. Tale parametro serve per impedire che durante una determinata cambiata ne sia richiesta una differente, mentre il ritardo serve per dare il tempo necessario agli attuatori di riposizionarsi dopo la cambiata.
Analizziamo nello specifico le singole parti di cui si compone *Gearshift_Manager*:

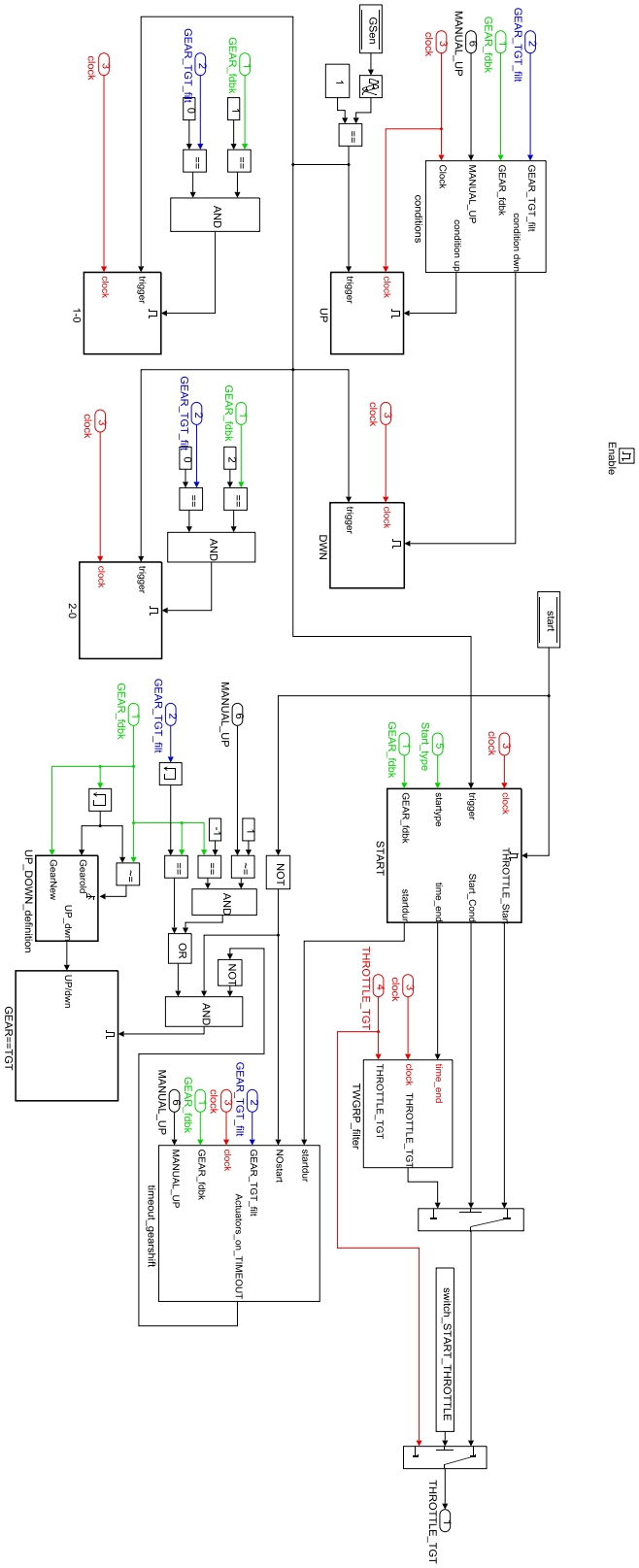


Figure 4.39 RideIT_RT_model/GEARSHIFT/Gearshift_Manager

conditions: all'interno di questo blocco viene eseguito il confronto tra la marcia inserita e la marcia da inserire. Si stabilisce quindi se è necessario un upshift o downshift inoltre, se è stata da poco richiesta una cambiata, s'impone un tempo di ritardo prima di poterne eseguire un'altra. Il calcolo del fattore che valuta la vicinanza o meno da un precedente cambio marcia, si esegue tramite il sotto blocco "Timer" che, ad ogni variazione della marcia, mette a disposizione del sistema l'istante di tempo in cui ciò è avvenuto. Dopo di che, si calcola il tempo trascorso dal cambio marcia e lo si confronta con "MaskUP" o "MaskDWN", che rappresentano il tempo da attendere prima di effettuare un'ulteriore cambiata in base al fatto che sia appena stato eseguito un upshift o un downshift. Se la disuguaglianza è verificata, si consente, tramite un fattore moltiplicativo l'opportuno cambio marcia, altrimenti no.
Nella condizione di upshift ("condition_up"), la marcia target è maggiore della marcia inserita, è richiesta una marcia superiore alla prima, e la marcia attuale è diversa da "-1" (condizione in cui la centralina non riesce a determinare la marcia).
Analogo ragionamento è svolto per determinare la condizione di downshift ("condition_down").
E' poi anche valutata la condizione "Start" che si attiva quando la moto è in folle e la marcia richiesta è la prima. In tal caso è attivato il blocco "Start_condition" che porta a valore unitario la specifica cella di memoria.
Infine, nel caso in cui la centralina riporti una marcia pari a -1 e, dando il consenso tramite l'input "MANUAL_UP", il sistema porta a "1" la cella di memoria "Gsen" (in modo da poter muovere gli attuatori) e viene comandato un upshift per uscire dall'intermarcia.

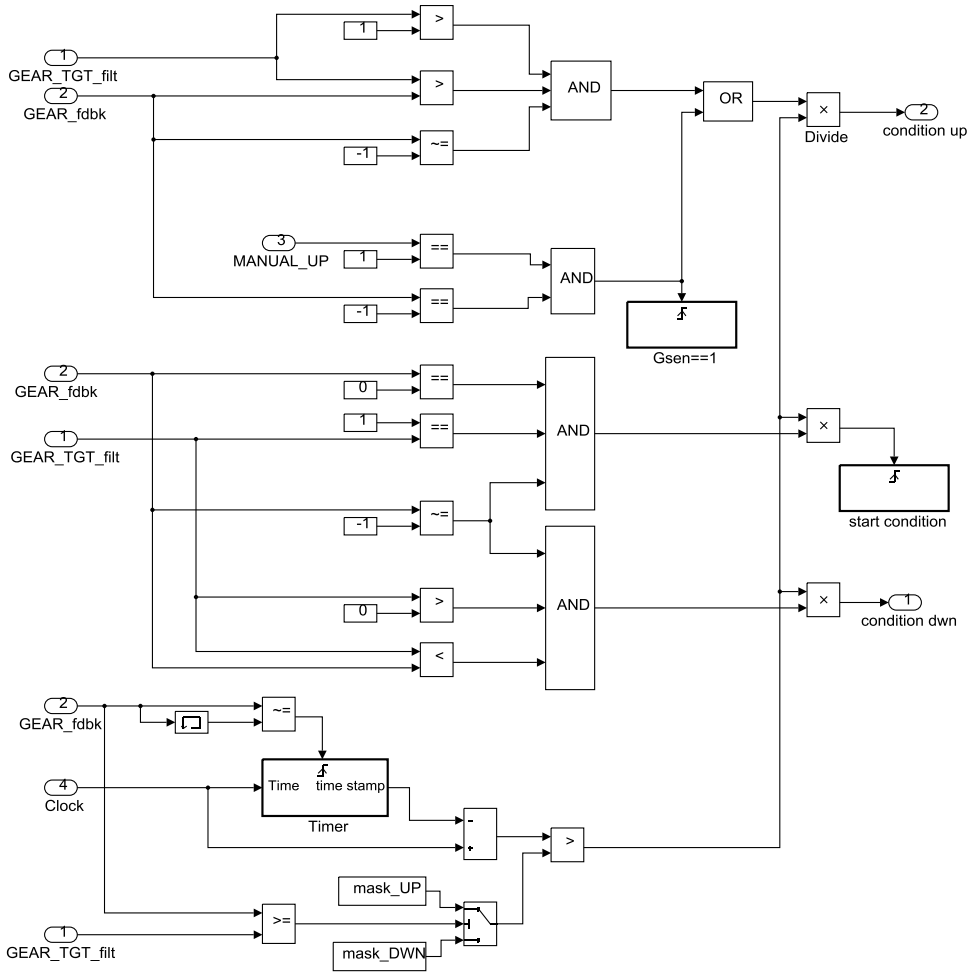


Figure 4.40 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/conditions

UP: quando sono soddisfatte le condizioni che richiedono l'upshift, viene abilitato questo sottosistema all'interno del quale sono contenute una serie di istruzioni da fornire agli attuatori per comandare le corrette movimentazioni.
Se Gsen è pari a 1, cioè il modello consente il moto degli attuatori, si ritarda la simulazione di un passo di calcolo e si attiva il blocco "Clutch_Forward" che comanda la movimentazione della frizione. Trascorso un ulteriore step di calcolo viene eseguito il blocco "Gear_Forward" che comanda la movimentazione del cambio per provare a fare l'upshift. All'interno di quest'ultimo vengono scritte nelle celle di memoria di "Posg", "Velg", "Accg", "Decg", "Forzag", i parametri che sono memorizzati nel vettore "Gfu" (Gear forward Up, inizializzato nel file di lancio) relativi alla posizione, velocità, accelerazione, decelerazione e forza da impartire

all'attuatore sul cambio durante un upshift. Viene inoltre messo a disposizione l'istante di tempo in cui inizia la procedura di cambiata. Analogamente si fa dentro al sottosistema "Clutch_Forward"per l'attuatore che gestisce la frizione. Nello stesso sottosistema, si porta "GSen" a zero per evitare che arrivino altri comandi agli attuatori mentre la movimentazione attuale è in atto.

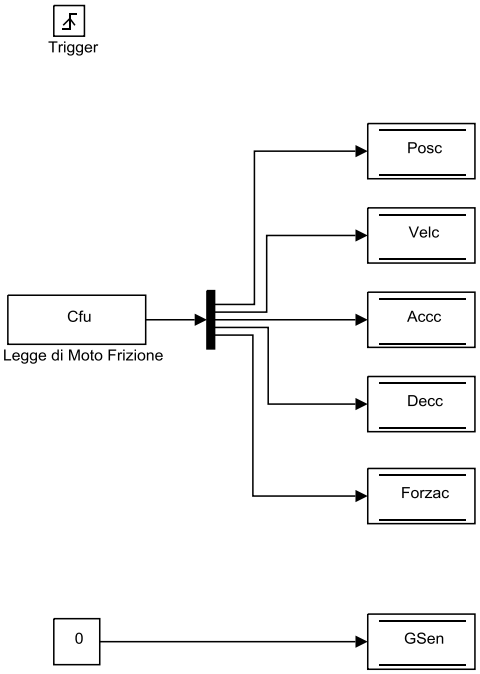


Figure 4.41
RideIT_RT_model/GEARSHIFT/
Gearshift_Manager/UP/Clutch_Forward

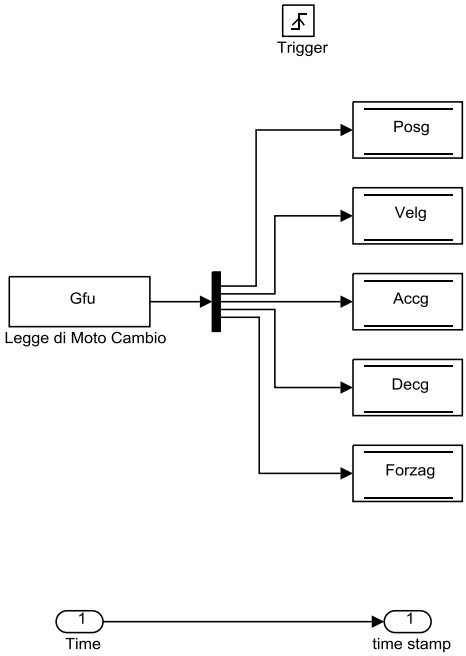


Figure 4.42
RideIT_RT_model/GEARSHIFT/
Gearshift_Manager/UP/Gear_Forward

Se dopo un certo intervallo di tempo pari a "DeltaTGear" ("DT(1)"), l'attuatore è ancora nella posizione di richiesta di upshift, il modello concepisce tale situazione come quella di un cambio marcia non andato a buon termine. Viene quindi attivato il blocco "Gear_Backward", dove si impone agli attuatori di tornare in posizione utile per riprovare la cambiata, scrivendo nelle suddette celle di memoria i parametri che sono memorizzati nel vettore "Gbu" (Gear backward Up, inizializzato anch'esso nel file di lancio). Inoltre s'imposta "Gsen" al valore unitario, proprio per consentire agli attuatori di riprovare la cambiata.

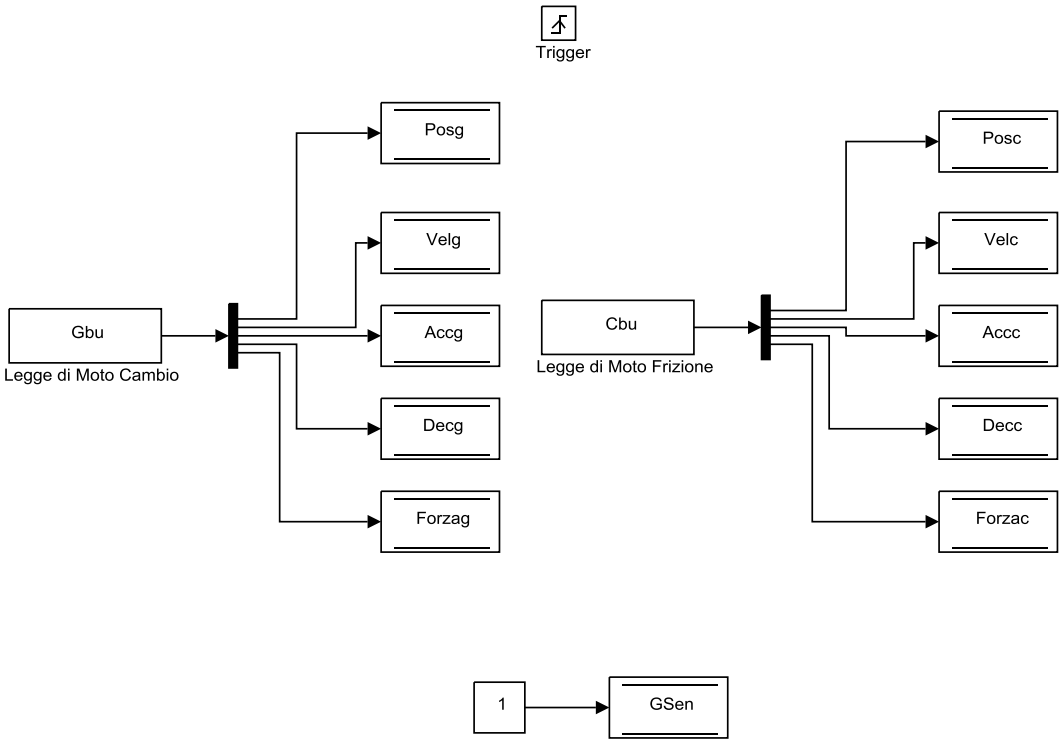


Figure 4.43 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/UP/Gear_Backward

DWN: la gestione del downshift è del tutto simile a quella dell'upshift, con la sola differenza nelle leggi di moto da imporre agli attuatori (per ogni situazione s'importa l'opportuna legge di moto dal file di lancio).

Anche il passaggio alla folle, messo in atto nei sottosistemi *1-0* (first to neutral) e *2-0* (second to neutral), è gestito in maniera analoga, ma impostando le opportune leggi di moto per il passaggio alla folle dalla prima o dalla seconda lette dal file di inizializzazione.

UP_DOWN_definition: questo blocco determina se l'ultimo cambio marcia comandato è stato un upshift o un downshift, un'informazione utile per il prossimo sottosistema che verrà analizzato.

GEAR=TGT: le istruzioni contenute all'interno di questo blocco riportano gli attuatori in posizione per compiere la cambiata successiva. Questo sottosistema si attiva quando la moto non è in simulazione di partenza (Start=0). In aggiunta deve essere verificato che la marcia inserita coincida con quella richiesta, o che la

centralina non riesca a determinare la marcia inserita mentre il consenso "MANUAL_UP" (che comanda un upshift per uscire da questa condizione di indeterminazione della marcia) non è attivo. Un'ultima condizione da rispettare affinché il blocco *GEAR=TGT* si attivi, è che non trascorra un lasso di tempo sufficientemente grande, stabilito all'interno del sottosistema *timeout_gearshift* (dell'ordine di qualche secondo) a partire dall'istante in cui è stato richiesto l'ultimo cambio marcia. Tale informazione, elaborata appunto nel sottosistema *timeout_gearshift* che vedremo meglio in seguito, risiede all'interno di un segnale booleano indicato con "Actuators_on_TIMEOUT".
In base al fatto che la moto provenga da un upshift o downshift (stabilito nel blocco *UP_DOWN_definition*), sono imposte differenti leggi di moto che riportano gli attuatori in posizione utile per compiere una successiva cambiata, secondo opportuni vettori definiti nel file di lancio. Gli attuatori, non vengono riportati in una posizione neutra (posizione di rilascio della leva di un comando sulla moto), ma si cerca di simulare il comportamento del pilota sulla moto, che può avere esigenza anche di eseguire cambiate in modo rapido, perciò si spostano gli attuatori solo del necessario. Si noti inoltre che "Gsen" viene riportato ad "1" per consentire cambiate future.

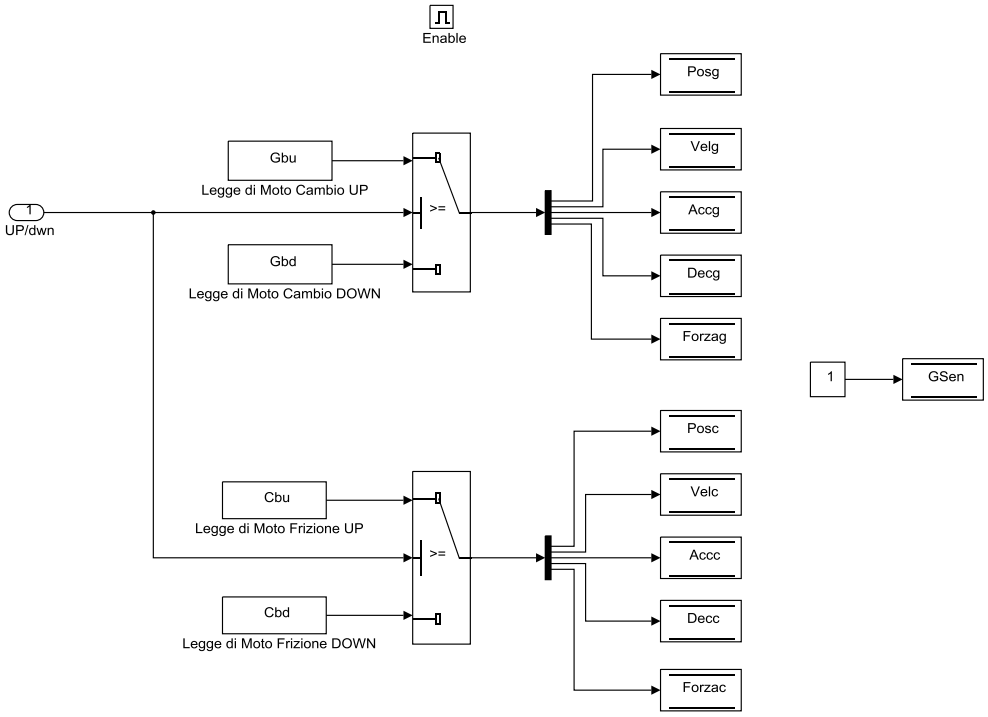


Figure 4.44 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/GEAR=TGT

timeout_gearshift: il ritorno alle condizioni neutre è regolato da questo blocco. Viene valutato l'istante di tempo in cui è stato richiesto l'ultimo cambio marcia ed è confrontato con il parametro "gs_wait" (dato dal file di lancio) nel caso in cui la moto non sia in simulazione di partenza (Start=0), altrimenti il confronto viene effettuato con il parametro "startdur", che tiene conto della durata della fase di partenza, sommato al contributo "gs_wait".
In tutti e due i casi di timeout, una volta superato il tempo di attesa, si riportano gli attuatori in posizione neutra, scrivendo le opportune leggi di moto sulle debite celle di memoria all'interno del sottosistema *timeout_actuators*. Un'altra condizione da verificarsi oltre al timeout è quella relativa alla marcia inserita che deve coincidere con quella richiesta. Questo è il caso in cui gli attuatori sono riportati alle condizioni neutre dopo che la cambiata è andata a buon fine.
Gli attuatori vengono riportati in posizione neutra anche nel "caso dubbio" in cui la centralina non si riesce a determinare la marcia in cui si trova la moto, pertanto il valore di "GEAR_fdbk" è impostato a "-1" e in aggiunta non viene neppure comandato un upshift per forzare un'uscita da tale condizione.
Nei casi in cui invece, la cambiata non riesca proprio e rimane la discrepanza tra la marcia inserita e quella richiesta, una volta trascorso il timeout, viene attivata la procedura di reset che consiste nell'andare a sovrascrivere il valore della marcia effettivamente inserita sulla variabile "GEARman" relativa alla marcia richiesta, in tal modo è possibile, sia riportare gli attuatori in posizione neutra ed eventualmente riformulare una nuova richiesta che consiste poi nell'andare a ripercorrere nuovamente la parte di modello, già descritta in precedenza, che si occupa di gestire le movimentazioni degli attuatori di cambio e frizione per l'esecuzione di un cambio marcia. In stato di guida automatico invece, nel caso in cui venga innescato il sottosistema *reset_condition*, si obbliga il passaggio allo stato di allarme.

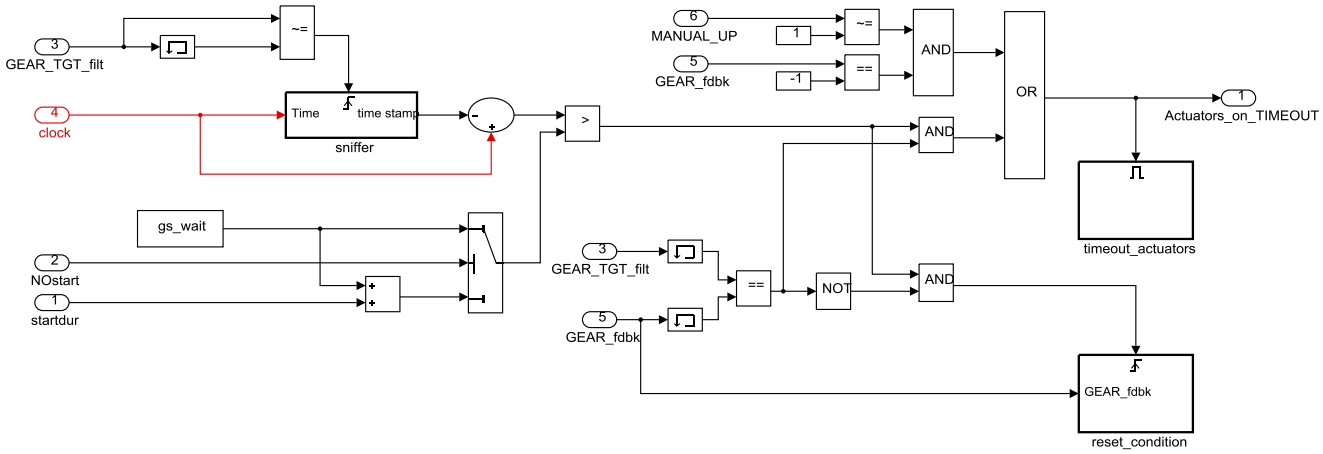


Figure 4.45 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/timeout_gearshift

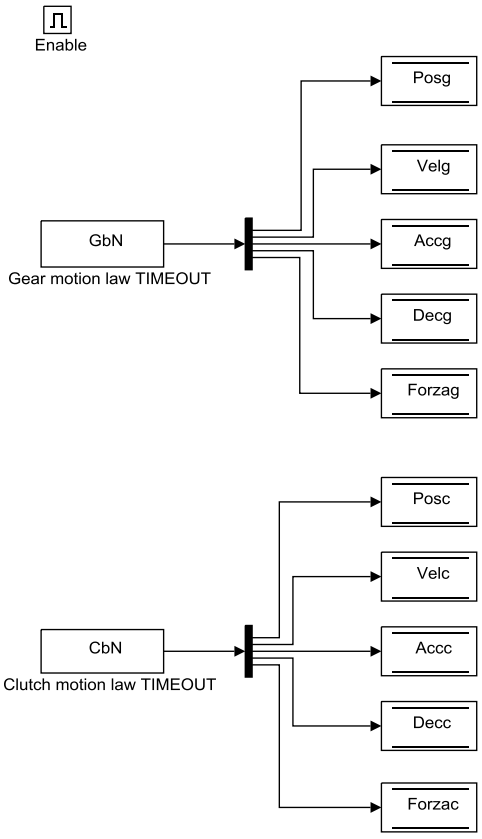


Figure 4.46 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/timeout_gearshift/ timeout_actuators

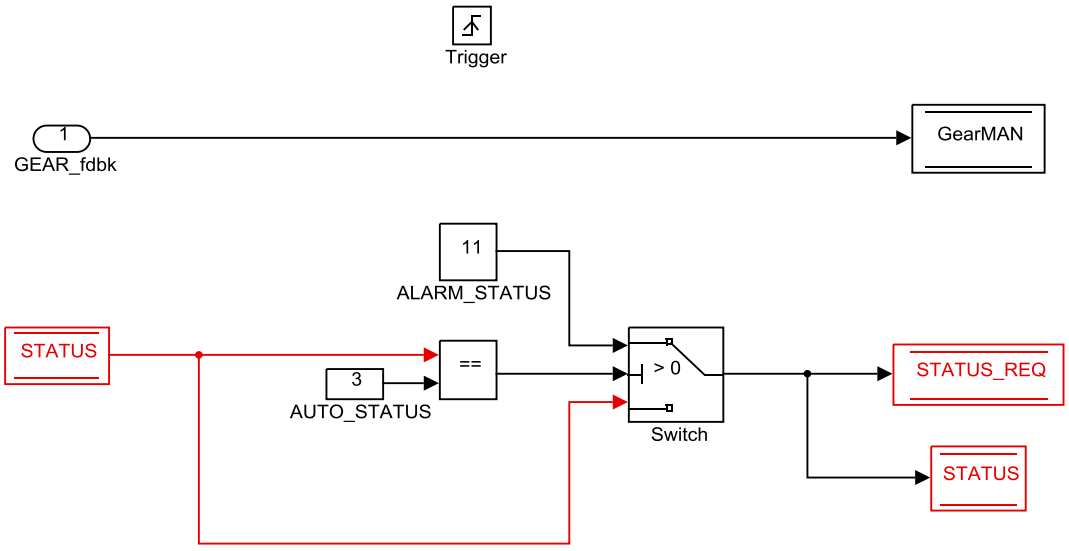


Figure 4.47 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/timeout_gearshift/ reset_condition

START: all'interno di questo blocco sono gestiti gli attuatori quando è richiesta una procedura di simulazione di partenza.

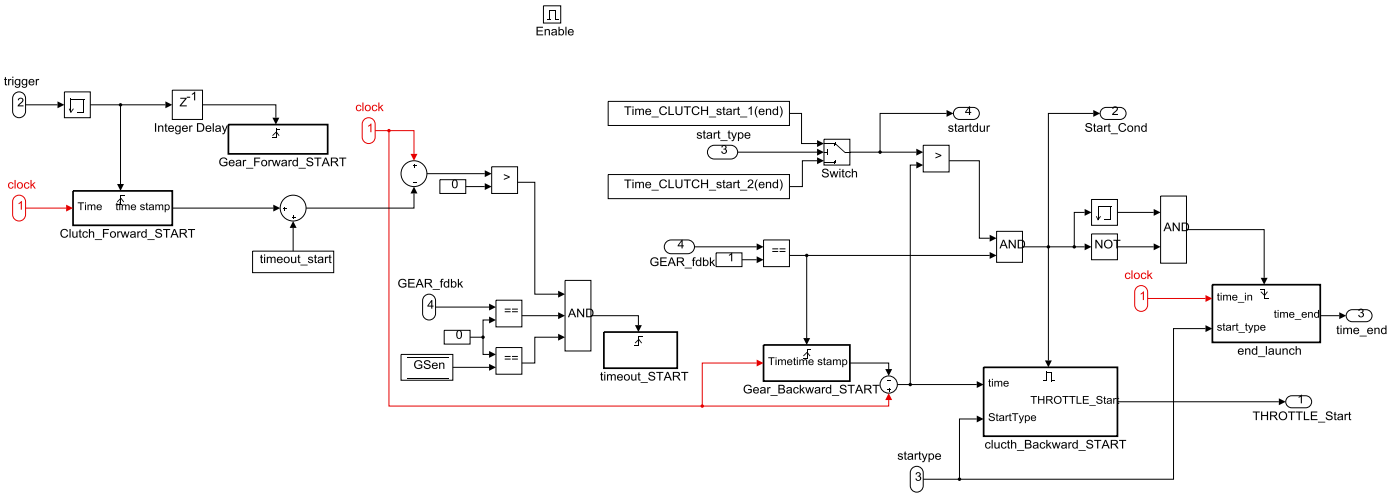


Figure 4.48 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/START

Durante tale procedura, si realizza il vincolo di muovere prima l'attuatore sulla frizione e dopo uno step di calcolo quello del cambio.

In *Clutch_Forward_START* è contenuta la legge di moto da imporre alla frizione, inoltre si porta a zero "GSen" e si mette a disposizione del modello l'istante di tempo in cui comincia la procedura di start.

In *Gear_Forward_START* è contenuta la legge di moto da imporre al cambio.

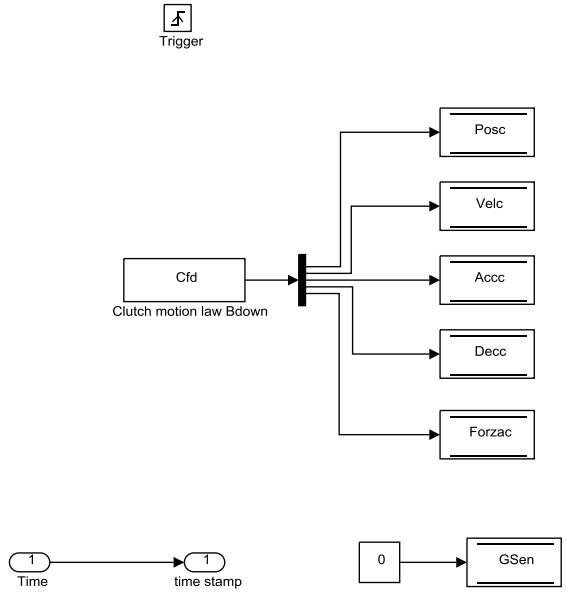


Figure 4.49
RideIT_RT_model/GEARSHIFT/Gearshift_ Manager/START/Clutch_Forward_START

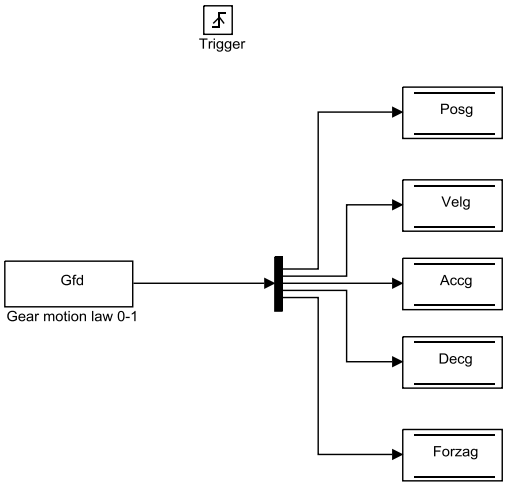


Figure 4.50
RideIT_RT_model/GEARSHIFT/Gearshift_ Manager/START/Gear_Forward_START

Dopo aver comandato la legge di moto agli attuatori, trascorso un lasso di tempo pari a "timeout_start", viene verificato se effettivamente è stata inserita la prima.
Se ciò non è vero, e quindi la marcia inserita è ancora la folle (GEAR_fdbk=0) e "GSen" è zero, viene imposta l'uscita dalla procedura di partenza, attraverso la parte di modello contenuta in *timeout_START*. In esso, oltre a riportare gli attuatori in posizioni utili per riprovare la cambiata, si va a scrivere "0" nella cella "START" e si riporta "Gsen" a valore unitario.

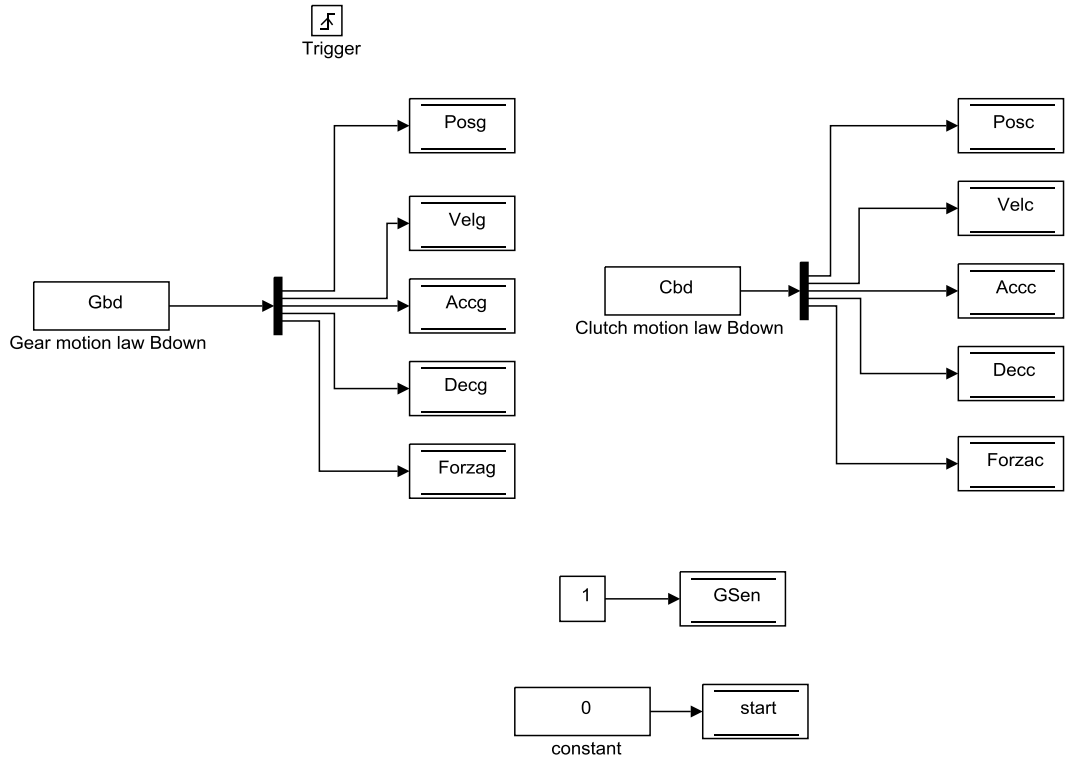


Figure 4.51 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/START/timeout_START

Nel caso in cui l'inserimento della prima marcia vada a buon fine, per prima cosa si riporta in posizione neutra l'attuatore del cambio, tramite il blocco *Gear_Backward_START*.

A partire da quando è stata inserita la prima, fintanto che ci si trova all'interno della fascia di tempo richiesta per la gestione della frizione in fase di partenza ("Time_CLUTCH_start_1(end)" o "Time_CLUTCH_start_2(end)" a seconda del tipo di partenza richiesto), il blocco *Clutch_Backward_START* riporta la frizione in posizione neutra e muove la manopola. Entrambe queste attuazioni seguono profili dettati da lookup tables, che associano ad ogni istante di tempo la posizione dell'attuatore sulla frizione e sulla manopola.

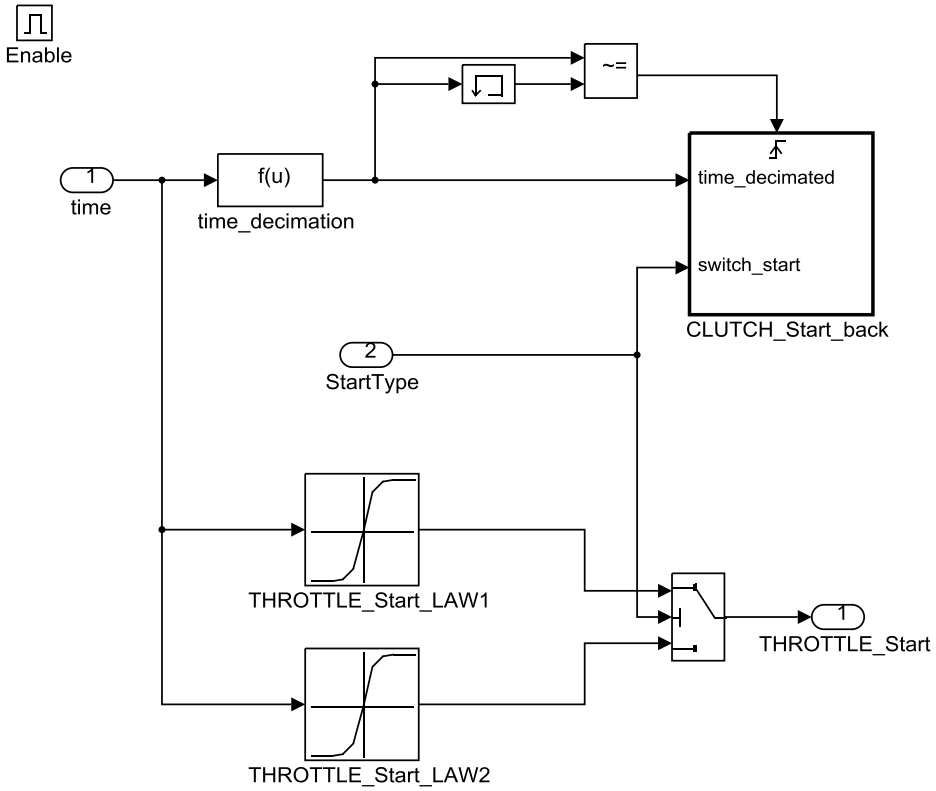


Figure 4.52 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/START/ clucth_Backward_START

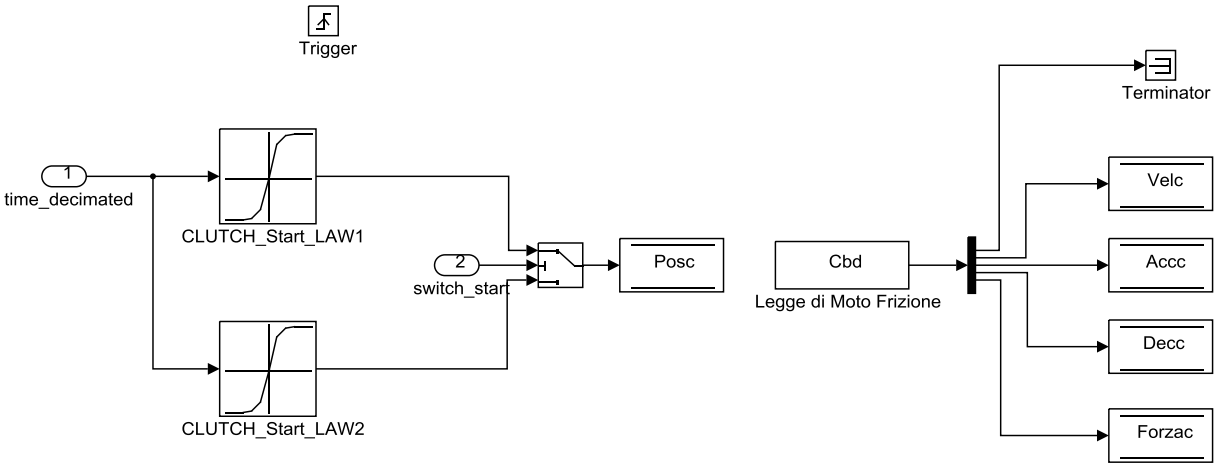


Figure 4.53 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/START/clucth_Backward_START/ CLUTCH_Start_back

Quando non è più verificata la condizione di partenza poiché è trascorso il tempo "Time_CLUTCH_start_(end)", si attiva il trigger relativo al sottosistema *end_launch*, che rileva il tempo e scrive nella cella di memoria "Start_transient" l'ultimo valore di apertura della farfalla in base a se è stata simulata una partenza di tipo 1 o 2.

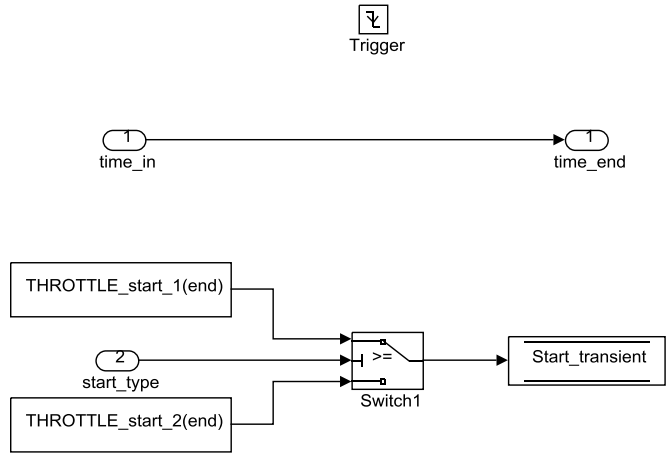


Figure 4.54 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/START/end_launch

TWGRP_filter: al termine della procedura di partenza è imposta una rampa di raccordo per portare il valore di apertura manopola al valore richiesto attraverso il blocco *TWGRP_filter*.

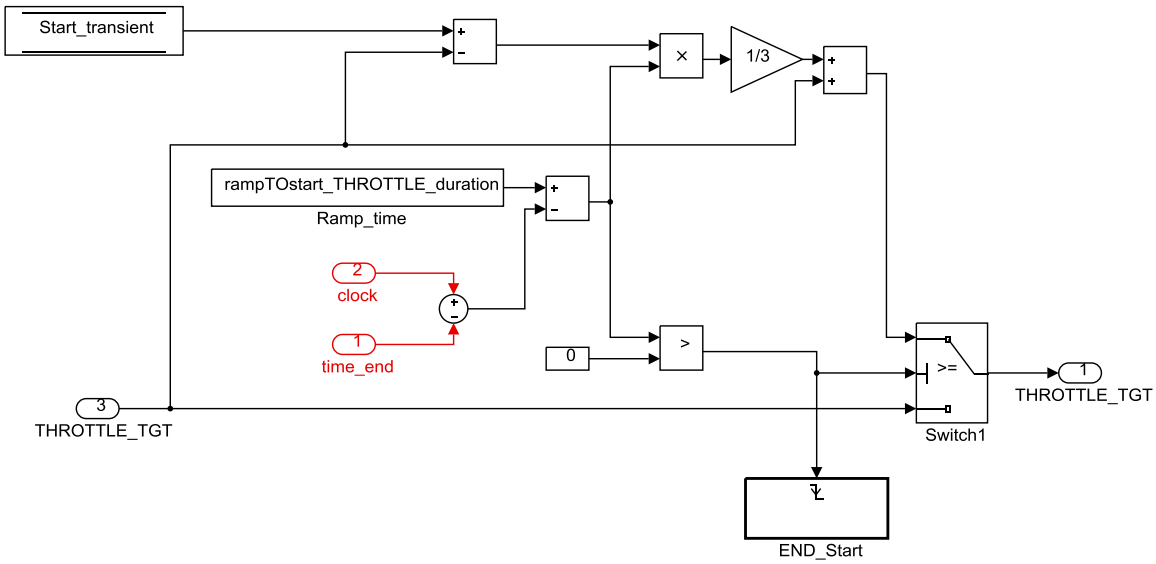


Figure 4.55 RideIT_RT_model/GEARSHIFT/Gearshift_Manager/TWGRP_filter

MANUAL_Clutch_Override: In stato di guida manual (STATUS 1), la frizione deve poter essere gestita manualmente, pertanto la legge di posizione per il rispettivo attuatore, in stato manuale, viene definita direttamente all'interno di questo sottosistema, che ha come input il vettore delle richieste manuali da cui andare a estrarre la variabile "CLUTCH_man" per sovrascriverla nella cella di memoria di "Posc" (Clutch Position).

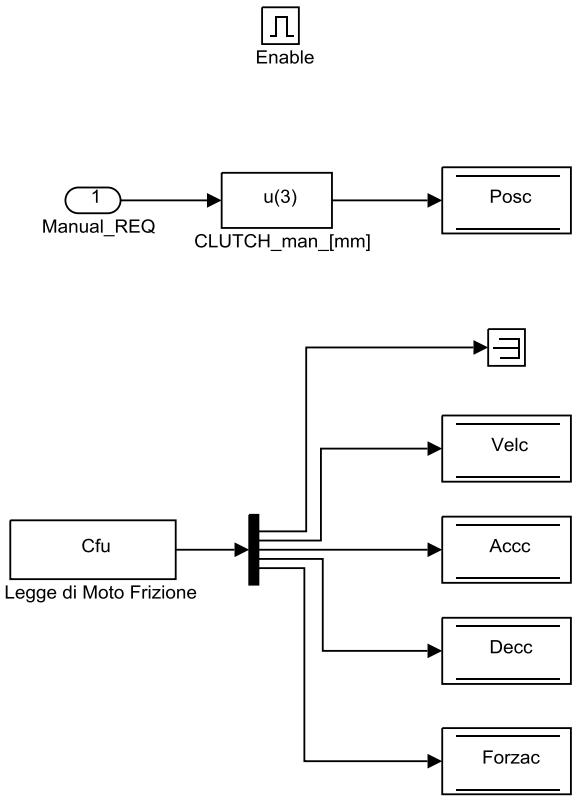


Figure 4.56 RideIT_RT_model/GEARSHIFT/MANUAL_Clutch_Override

4.6 OUTPUT

All'interno del blocco output, sono messe a disposizione del modello LabView Real-Time tutte le grandezze elaborate negli altri sottosistemi del modello, che potranno essere inviate agli attuatori, piuttosto che alle interfacce utente o alla centralina del motoveicolo. Alcune grandezze sono utili solo esclusivamente al modello stesso per cui, in questo blocco, vengono semplicemente terminate.

Come già è stato detto, il controllo degli attuatori Linmot avviene grazie allo scambio di informazioni tra il modello sviluppato per il target real-time e i relativi

servo-controller. In particolare vengono inviati a quest'ultimi tutti i parametri relativi alla legge di moto della movimentazione da eseguire.

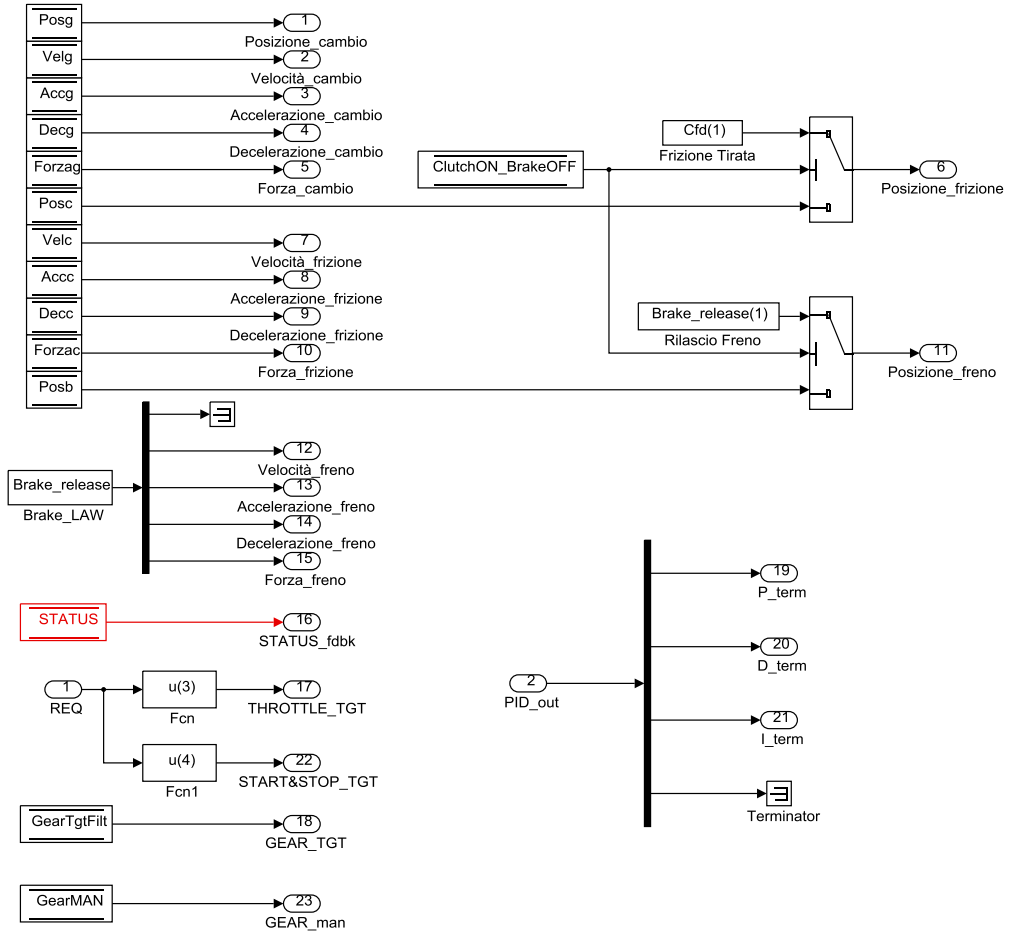


Figure 4.57 RideIT_RT_model/OUTPUT

4.7 Model Interface Toolkit

Per permettere la gestione degli attuatori da Host e da Consolle tramite segnali generati dal modello realizzato in Simulink e permettere l'esecuzione di questo sulla piattaforma NI CompactRIO con SO real-time, è stato necessario utilizzare un toolkit che la National mette a disposizione per comunicare con applicazioni sviluppate in ambienti differenti da LabView, nel caso in questione Matlab/Simulink.
Il Model Interface Toolkit (MIT), disponibile in LabView, è lo strumento che è stato utilizzato per importare il modello Simulink nell'**R_Main.vi** creato dall'Ing Michele Taccioli per essere eseguito sul target real-time. In particolare il MIT offre la

possibilità di collegare gli *input*, *output* e i *parameters* (scalari e vettori) del modello Simulink a controlli e indicatori del *VI*.

Per l'abilitazione al funzionamento in real-time, il modello Simulink deve essere compilato, in quanto il *VI*, grazie al MIT, attiva il modello ".dll", una volta che viene avviata la simulazione.

All'interno dell'**R_Main.vi** che racchiude l'intero progetto LabView Real-Time, la logica del MIT è la seguente:

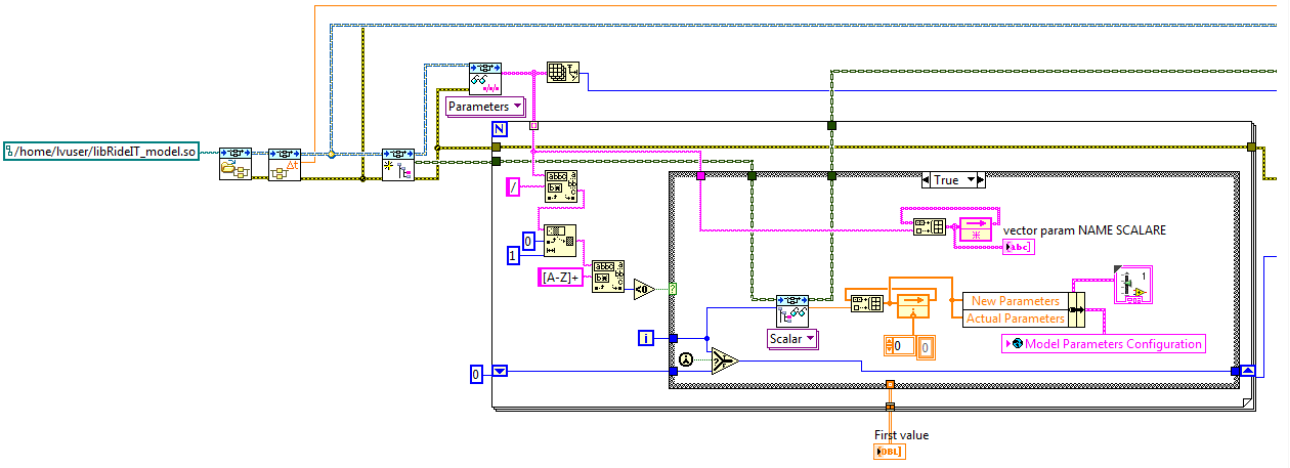


Figura 4.58 Diagramma a blocchi MIT e loop di controllo parametri

Con riferimento alla figura 4.58 si introduce la prima parte della struttura del MIT partendo dal **Load_Model.vi** che riporta la referenza del modello Simulink nel *VI* in base al percorso indicato in cui si trova il file *.dll* all'interno del disco. Tramite il **Get_Model_Period.vi** viene letto il segnale di tempo impostato in Simulink, per stabilire a quale frequenza di ciclo deve girare il loop principale del MIT che esegue il modello.

Successivamente c'è il **Create_Interface_Parameters.vi** che sostanzialmente crea le interfacce di collegamento degli elementi del modello che possono essere richiamati all'interno del *VI* e che riguardano: **Inports**, **Outports**, **Parameters**, **Signals**.

Il modello Simulink, oltre agli I/O, contiene al suo interno dei parametri che possono essere anche di diversa natura (matrici, vettori o scalari). Di base, questi parametri sono tutti impostati nel file di inizializzazione che viene lanciato prima della compilazione del modello. Ovviamente può succedere che, in base al tipo di test, o a seconda del modello di motoveicolo, ci siano esigenze differenti, tali da comportare

necessariamente la modifica dei parametri del modello (i rapporti di trasmissione del cambio tanto per fare un esempio). Per far fronte alle varie esigenze l'utente può comodamente riconfigurare i parametri del modello agendo dai singoli menù a pop-up dell'interfaccia Host, già descritti nel capitolo 2, senza quindi che ci sia bisogno di intervenire tutte le volte sul file di inizializzazione e ricompilare il modello. Il meccanismo di sovrascrittura dei parametri avviene nell' *R_Main.vi* e una parte della sua logica è riportata in figura 4.58 dove si può vedere il **Get_Paths.vi** che richiama tutti i parametri del modello raggruppandoli in forma di un unico array di stringhe contenente i nomi degli stessi e mandato all'interno di un ciclo "for" per essere processato.

Il ciclo for serve più che altro ad eseguire dei controlli dal front panel sul modello compilato e avere la certezza di ritrovare una lista completa dei parametri posizionati correttamente. All'interno è presente una *case structure* e s'indaga sui nomi dei parametri per capire quali di questi sono degli scalari (caso *true*) e quali sono dei vettori (caso *false*), dal momento che nel modello Simulink si è adottata la convenzione di nominare i parametri scalari con nome che comincia per lettera *minuscola* e i parametri vettore con nome che comincia per *maiuscola*.

Questa distinzione viene adoperata anche nel "while loop" successivo, mostrato in figura 4.59, dove effettivamente avviene la variazione dei parametri. Si tratta di un loop a bassa priorità e che viene fatto girare ad una frequenza relativamente bassa (step time impostato a 2 secondi).

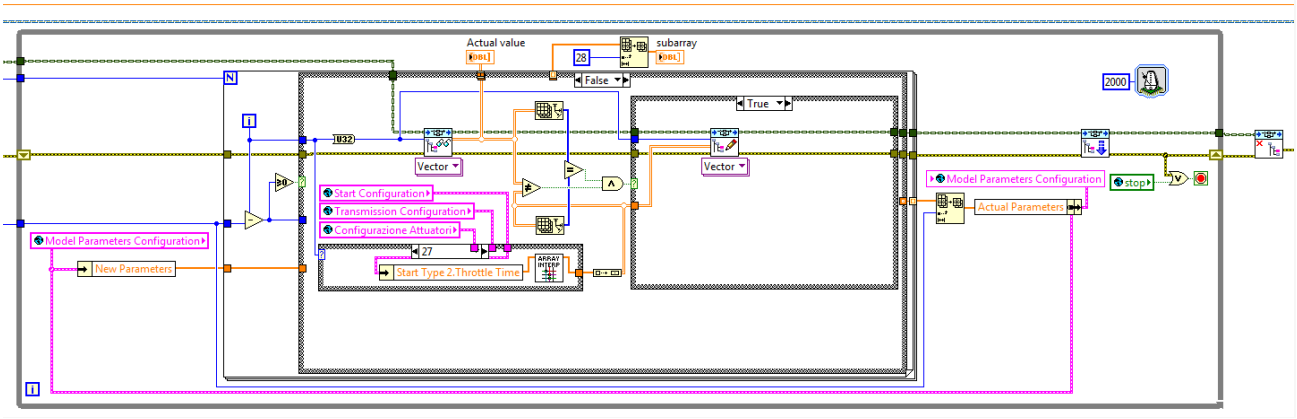


Figura 4.59 Loop di configurazione parametri vettore

I valori attuali dei vettori, facendo riferimento sempre alla figura 4.59, sono letti da modello tramite il **Get_Parameter.vi**. Si richiamano invece dall'Host i vettori

raggruppati dentro alle variabili globali: “Start Configuration”, “Trasmission Configuration” e “Configurazione attuatori”. Dal confronto si stabilisce se i valori dei vettori sono cambiati e quindi se si sta richiedendo una nuova configurazione. Si verifica inoltre, che le dimensioni dei vettori nella nuova configurazione siano compatibili con quelle relative ai vettori del modello in origine per poi stabilire se abilitare o meno la sovrascrittura dei parametri.

All’interno di una case structure si procede con la scrittura dei nuovi vettori tramite il **Set_Parameter.vi**. Tale scrittura consiste nella pratica in una sostituzione temporanea dei valori definiti dalle variabili globali con quelli dati dalla configurazione iniziale del modello compilato, senza che questi ultimi vengano eliminati.

Per gli scalari, si fa riferimento alla figura 4.60. La logica all’interno del while loop è simile sotto alcuni aspetti a quella del caso precedente. Anche per gli scalari si esegue un confronto fra i valori letti da modello con i “New parameters” definiti da Host tramite il Model Parameter Configuration e che nel *VI* sono richiamati attraverso l’omonima variabile globale.

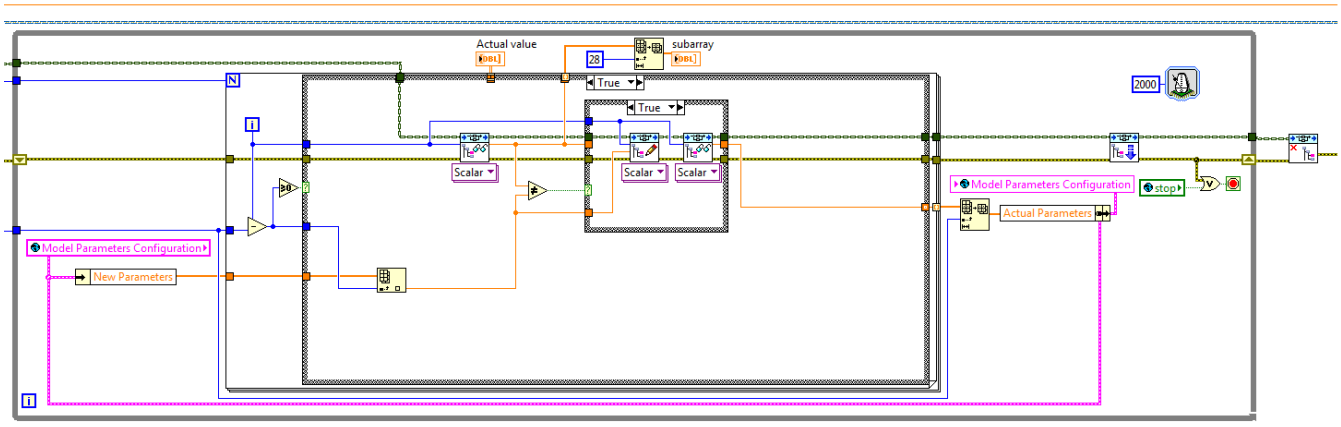


Figura 4.60 Loop di configurazione dei parametri scalari

I nuovi valori scalari vengono prima impostati nel modello se la condizione true sul case è rispettata, dopodiché vengono nuovamente riletti tramite il **Get_Parameter.vi** per assicurarsi che siano state applicate correttamente le modifiche.

Dopo aver esaminato i parametri ed eseguite le operazioni di lettura e scrittura per essi, per far sì che le modifiche siano effettivamente applicate al modello occorre eseguire una particolare istruzione, detta *commit* tramite il blocco **Commit**

Parameters.vi. Tali modifiche verranno poi mantenute all'interno del *VI* e riattivate ad ogni avvio del sistema.

Per quanto riguarda gli input e output, questi sono parametri che potenzialmente possono variare ad ogni ciclo di esecuzione del modello, pertanto non possono essere assolutamente modificati nel loop a bassa priorità. Dovendo aggiornarsi in tempo reale durante la simulazione, vengono trattati all'interno del loop principale del MIT che si occupa anche di eseguire il modello di Simulink sul target real time.

Si tratta in particolare di un "timed loop" (riportato in figura 4.61) e in quanto tale permette di effettuare cicli a frequenza costante; nel caso specifico la frequenza è pari a 1 KHz e viene stabilita sulla base del segnale di tempo letto dal modello, opportunamente moltiplicato per 1000 per convertirlo in *ms*.

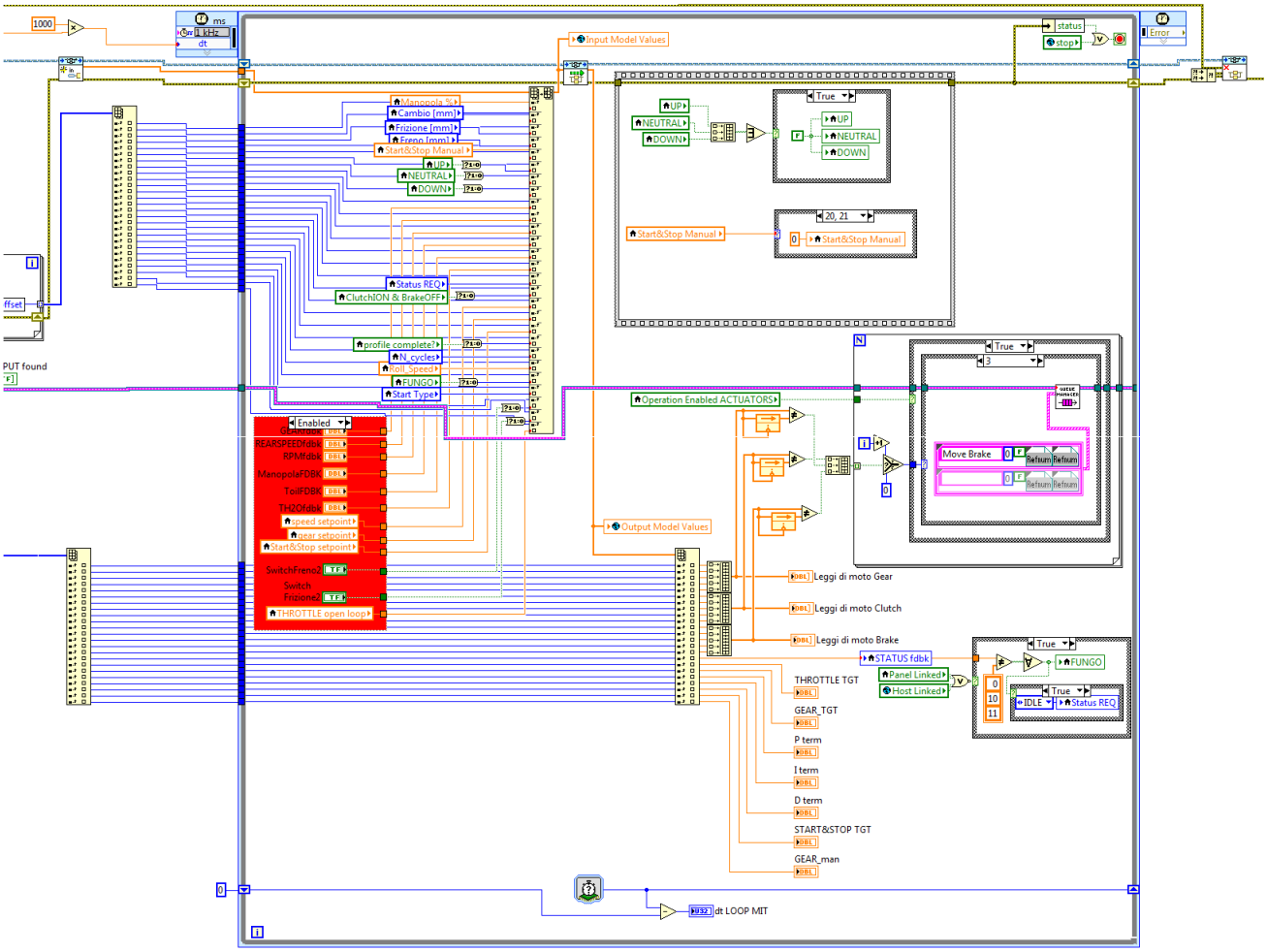


Figura 4.61 Timed Loop principale del MIT

Il blocco di lettura degli input è il **Create Inports Array.vi** che restituisce un array di dimensioni pari al numero di ingressi del modello Simulink. Il vettore viene mandato all'interno del timed loop per essere ridefinito sulla base dei valori attuali acquisiti dai comandi di interfaccia utente. Attraverso il blocco *Replace Array Subset* vengono gestiti i rimpiazzamenti per il vettore degli input per mezzo anche degli indici che arrivano in ingresso al blocco stesso e definiscono i campi opportuni dove andare a impostare i nuovi valori fissati dalle variabili locali e che andranno poi a comporre il vettore da inviare al modello.
A valle del suddetto blocco viene aggiornata la variabile globale "Input Model Values" con il vettore dei nuovi input prima di essere inviati al modello che all'interno del timed loop viene eseguito tramite il **Take Time Model Step.vi**. Tale blocco riceve in ingresso il vettore degli input aggiornati e restituisce il vettore degli output che prima va ad aggiornare la variabile globale "Output Model Values", poi viene suddiviso nei vari elementi di cui è costituito per mezzo del blocco *index Array*. Anche in questo caso occorre inviare un vettore che riporta tutti gli indici in ingresso al blocco *index array*, in modo da assegnare ad ogni grandezza l'output specifico.
La struttura logica del MIT viene completata con il blocco **Unload Model.vi** che interrompe la referenza del modello.
La creazione dei vettori degli indici è un aspetto importante della logica del MIT e viene affrontata in una parte del *VI* esterna al timed loop. Facendo riferimento alla figura 4.62 si possono notare le due liste di canali di input e output del modello Simulink riportate in formato stringa. In ciascuna delle stringhe viene indicato il percorso di un particolare input, o output, presente all'interno del modello. Come già è stato detto in precedenza, i vari input e output del modello Simulink son stati disposti nel livello a gerarchia più elevata, pertanto c'è corrispondenza sui nomi anche per gli elementi che compongono gli array di stringhe.
Per ciascuno dei due casi (I/O) si ha un array che viene processato dentro a un ciclo for. I blocchi significativi presenti all'interno del for sono il **Get information by Path.vi**, e l'*Unbundle*. Il primo ha come in ingresso la referenza del modello e il path relativo all'i-esimo elemento dell'array, mentre in uscita restituisce il cluster con le

informazioni del dato elemento. Il secondo va a selezionare dal cluster solo l'informazione di interesse, che riguarda l'offset nel caso specifico.

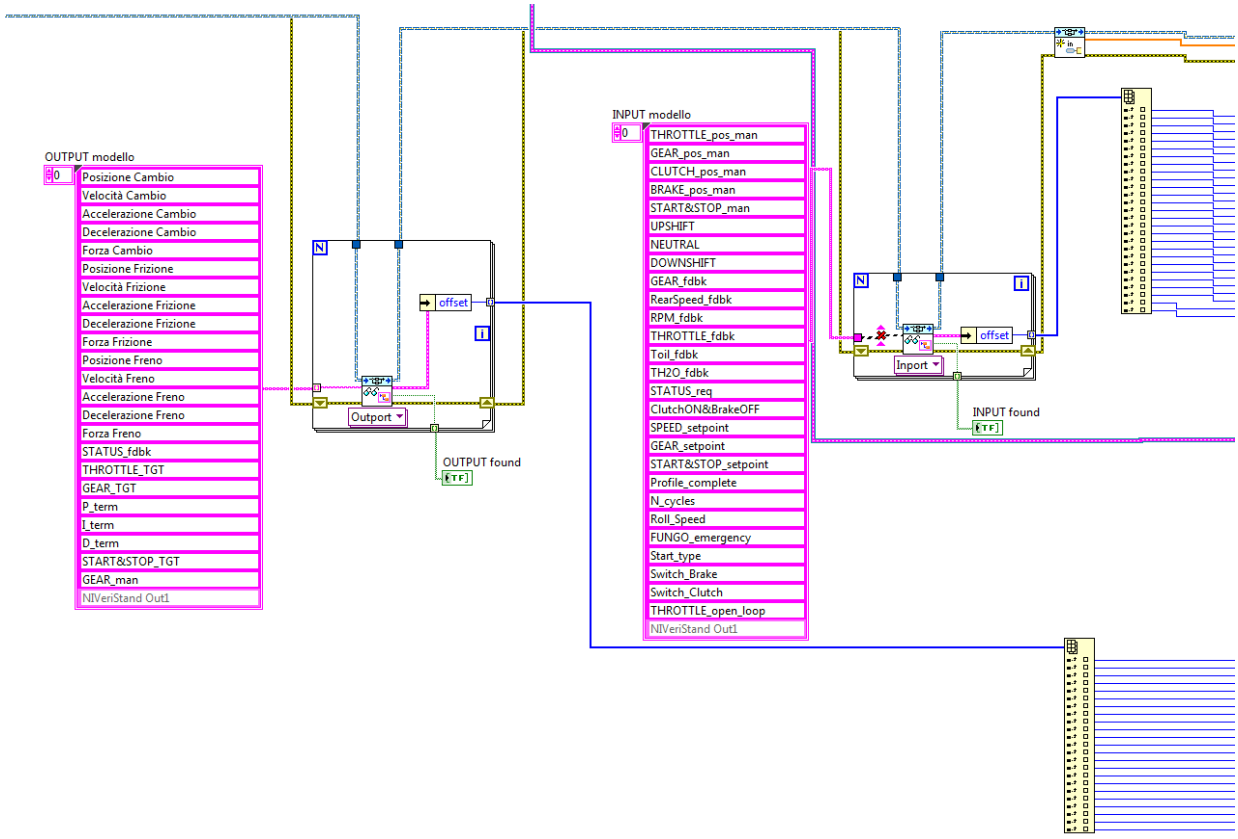


Figura 4.62 Costruzione dei vettori degli indici

Sempre con riferimento al blocco *Get information* viene effettuato anche un controllo sulla presenza o meno degli elementi del vettore I/O grazie al segnale booleano ("found"), che ad ogni iterazione viene passato ad un indicatore specifico connesso al front panel principale del *VI* utilizzato nella fase di sviluppo.
Attraverso i cicli for si ottengono in definitiva i due vettori di indici di dimensioni pari rispettivamente agli I/O del modello, che dovranno essere indirizzati ai blocchi di ricostruzione dei vettori all'interno del loop principale del MIT (descritto in precedenza), dopo averne compiuta la scomposizione nei singoli elementi dentro agli *index array*.

Capitolo 5

Test funzionali

Durante l'attività di tesi sono state eseguite diverse prove al fine di:

- Applicare e validare in ambiente simulato il modello di controllo della Consolle;
- Verificare sperimentalmente (attraverso specifici test condotti in laboratorio) il regolare funzionamento dei componenti del sistema, sotto diverse condizioni operative di funzionamento;
- Verificare sperimentalmente l'operatività dei modelli dal punto di vista delle comunicazioni e delle funzionalità volte alla messa in sicurezza dei componenti del sistema;

In quest'ultimo capitolo vengono presentati i test effettuati sul sistema e i risultati ottenuti mediante l'applicazione dei controlli precedentemente illustrati. Viene valutato infine, un possibile sviluppo futuro per il progetto RideIT.

5.1 Test sulla Consolle

5.1.1 Verifiche del Software

Questa sezione mostra in maniera pratica come si sono svolti i test di messa a punto del software della Consolle.
Il modello di controllo è stato implementato sulla piattaforma hardware STm32 utilizzando l'apposito tool adibito alla compilazione e facente parte della libreria *Waijung* installata in Simulink.
Il controllo e la verifica dell'applicazione è stato fatto fin dalle prime fasi dello sviluppo tramite un pc Host standard che permettesse di emulare la parte real-time del sistema RideIT cui doveva connettersi la Consolle.

In pratica, secondo tale metodologia il modello software da verificare veniva fatto interagire direttamente con la parte emulata del sistema RideIT, per poterlo far lavorare come se fosse in condizioni operative reali. In questo modo è stato possibile eseguire verifiche ogni volta che si arrivava a produrre un modello software funzionante, senza attendere la disponibilità del cRIO programmato.
II vantaggio di quest'approccio è legato alla possibilità di agevolare il debug del modello per il microcontrollore ad ogni passo evolutivo, disponendo di hardware standard, come il PC, per la creazione dell'ambiente virtuale del test, evitando il ricorso del cRIO. Procedendo un passo alla volta con lo sviluppo, quando ci si accorgeva di un errore o anomalia, il software veniva corretto e ricompilato per poi essere nuovamente verificato.
Per la verifica del software creato ed eseguito sul target STm32, ci si è avvalsi dell'ambiente di Simulink creando un modello che facesse appositamente dialogare la parte Host con quella target sfruttando il protocollo UDP. Il modello Simulink in questione impiegato per la simulazione è illustrato in figura 5.1. Si osserva che gli input e gli output, in questo modello, sono collegati agli appositi blocchetti che gestiscono la comunicazione UDP e che permettono anche di accedere e controllare le funzionalità della scheda di rete montata sull'Host. In questa configurazione, l'ambiente di Simulink funge da pannello di comando dell'applicazione in esecuzione sul target STm32.

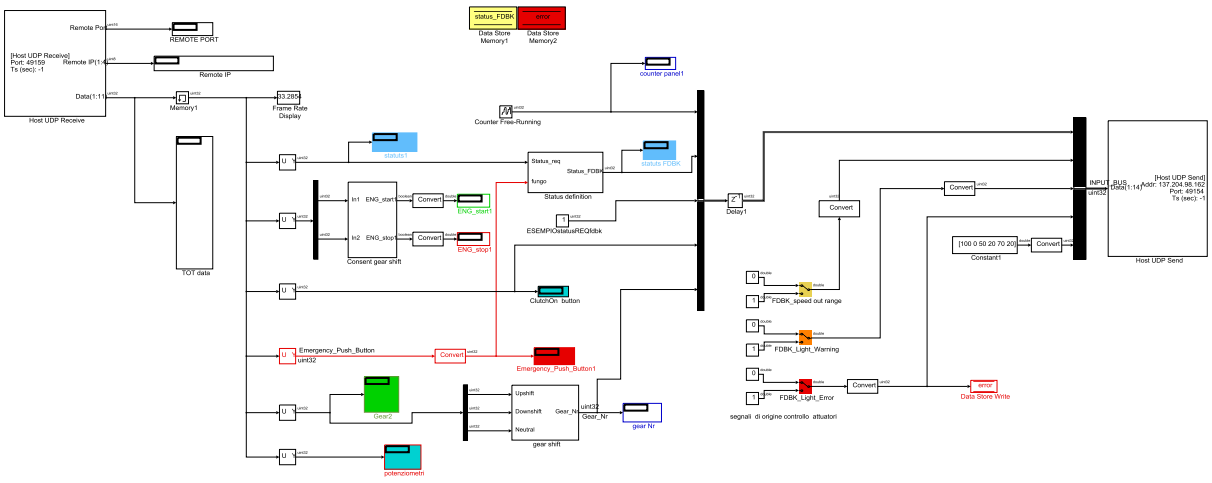


Fig.5.1: Screenshot del modello Host in SIMULINK utilizzato nei test

In Simulink sono stati introdotti vari *displa*y e *scope* per facilitare il controllo dell'applicazione a lato del pc Host, poiché essi permettono di acquisire e visualizzare on-line i segnali di interesse.
Quando l'applicazione è in esecuzione il target funge da controllore per il processo, l'Host permette il monitoraggio dei segnali che rappresentano i setpoint dati dai comandi della Consolle, mentre in output mette a disposizione i segnali di feedback che la Consolle stessa si aspetterebbe di ricevere. Tra le grandezze inviate alla Consolle ci sono anche dei parametri che possono essere modificati dall'Host, i cui valori condizionano il modo di funzionare dell'applicazione. Il cambiamento del valore di un parametro, sia questo derivante dal modello Simulink del sistema emulato (modello dei test), sia esso derivante dal modello real-time autentico del sistema reale, comporta la variazione dello stesso parametro nel codice eseguito sul target STm32.
Perciò, con riferimento allo schema a blocchi del modello della Consolle (capitolo 3), questo può voler dire poter cambiare le costanti numeriche che definiscono le mappe definite dalle Look-Up-Tables utilizzate per la caratterizzazione dei segnali letti dagli ADC.
In figura 5.2 è riportata un'immagine del circuito prova scattata durante un test:

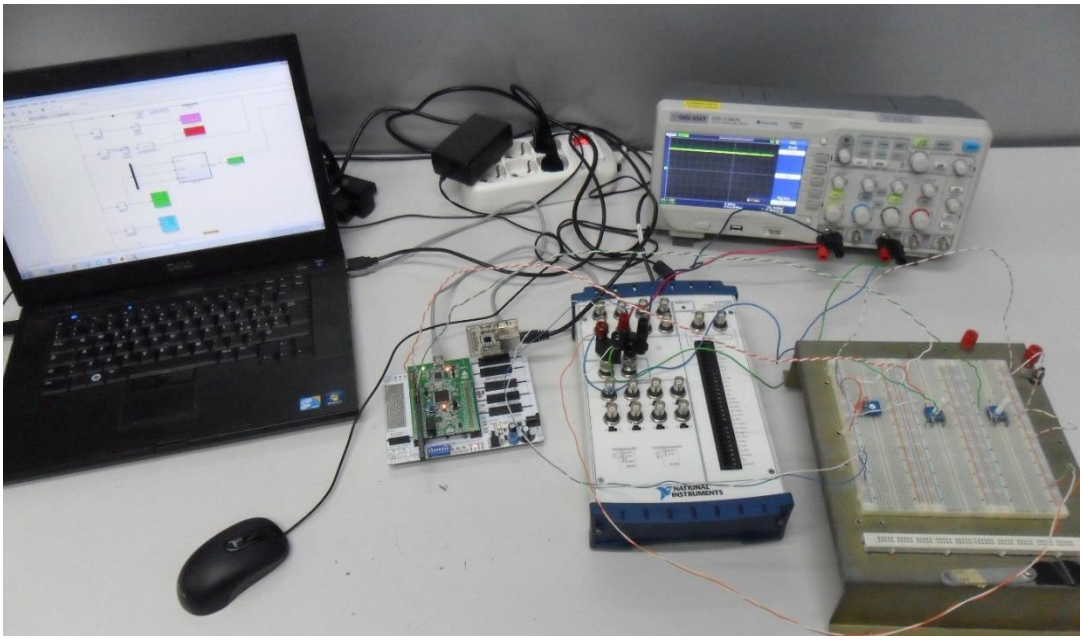


Fig.5.2: Fotografia del circuito realizzato in laboratorio per la verifica del componente STm32

Con riferimento alla figura 5.2, la STm32 viene installata su una particolare *expansion board* acquistata appositamente per eseguire i test. Questo componente permette, oltre alla possibilità di avere slot compatibili con gli header pins della STm32 e del modulo Ethernet, di utilizzare periferiche programmabili aggiuntive come pulsanti e indicatori LED. Ciò ha consentito di ovviare al problema dell'impiego e dell'affollamento di cavi elettrici da impiegare provvisoriamente nei test per la presenza di molti componenti da gestire e verificare.
E' stata utilizzata una scheda NI DAQ USB-6251 multifunzione configurata da pc con NI Measurement & Automation Explorer (MAX) per fare le acquisizioni, il data logging e portare l'alimentazione sui potenziometri. Infine, un oscilloscopio utilizzato a scopo di controverifica e monitoraggio dei segnali interessati.
La schermata dell'oscilloscopio riportata in figura 5.3 mostra la forma e il livello di tensione raggiunto nel caso di una misura effettuata su un pin della scheda STm32 sul quale è stato collegato un pulsante.

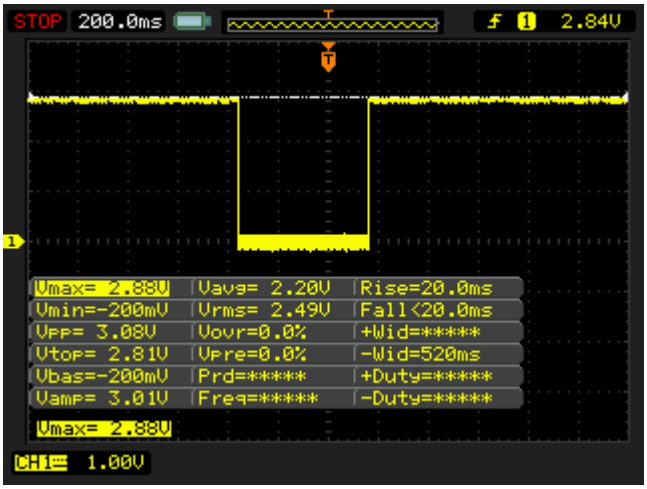


Fig.5.3: Misura della tensione con oscilloscopio per la verifica di un pulsante

Si osserva, che il segnale rimane a livello alto (~3V) quando il pulsante non viene premuto in quanto, come accennato in precedenza (capitolo 3), è invertita la logica rispetto alla convenzione usuale per tutti gli input digitali presenti nel circuito.
Test analoghi hanno consentito di verificare la correttezza dei segnali I/O gestiti dalla scheda di controllo STm32.

5.1.2 Verifica dei collegamenti elettrici

Questa fase ha riguardato la verifica dei collegamenti elettrici una volta che la Consolle è stata fisicamente realizzata e assemblata in ogni sua parte (per lo schema elettrico del circuito della console si veda l'appendice C). La prova è stata eseguita con la Consolle funzionante in modalità stand-alone avendo in questo caso scaricato nel target STm32 un modello semplificato che permettesse solo di collegare direttamente tutti gli input agli output. Il test è stato necessario al fine di valutare il corretto funzionamento di tutti i pulsanti e indicatori installati accertandosi del corretto collegamento elettrico.

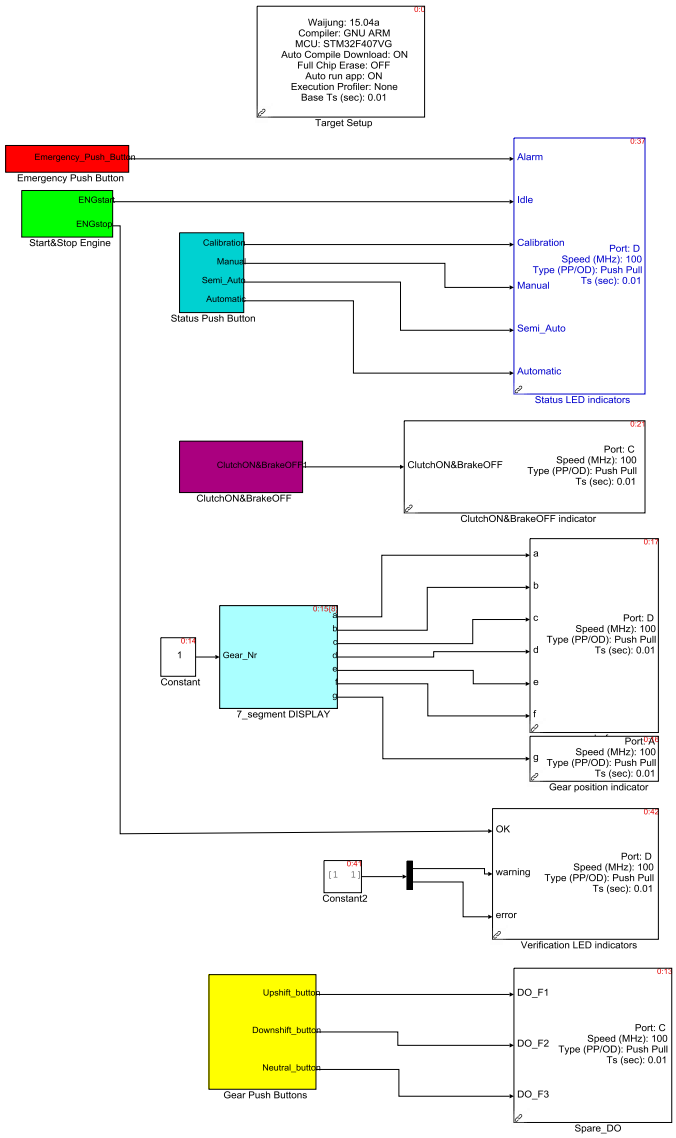
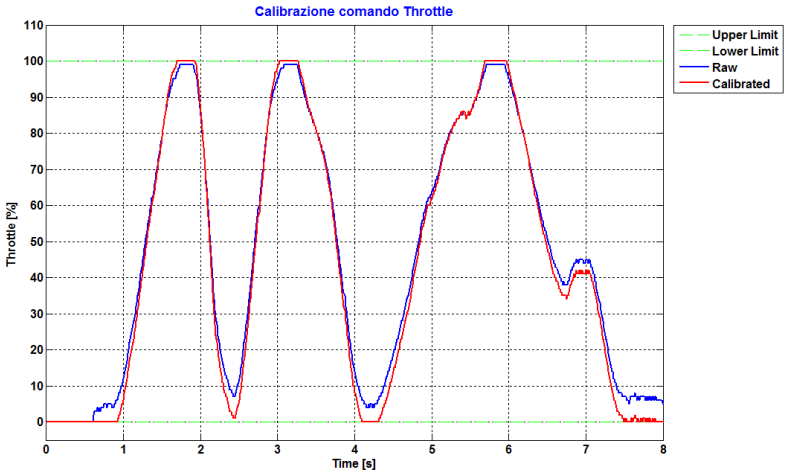


Figura 5.4: Screenshot del modello semplificato per la verifica del funzionamento dei comandi

5.1.3 Procedura di Calibrazione

L'obiettivo di questa calibrazione come già discusso nel capitolo 3, è stabilire dei riferimenti certi per i comandi della guida manuale. Per fare questo, una volta attivata tale procedura si trattano i segnali acquisiti dai comandi tenendo conto di come questi sono costruiti meccanicamente. Nella figura 5.5 vengono mostrati i grafici relativi ai risultati delle simulazioni, dove si riportano gli andamenti dei segnali acquisiti dai comandi della Consolle dimostrando in ciascun caso come la procedura di calibrazione implementata, riesca a far coincidere la corsa meccanica effettiva con quella teorica imposta da modello.

Nei grafici di figura 5.5 si può capire il comportamento del sistema nel caso in cui venga dato un setpoint fuori dallo spazio di lavoro. Si può notare in questo caso come intervengano le saturazioni che fanno in modo di limitare i valori dei setpoint entro lo spazio di lavoro consentito.



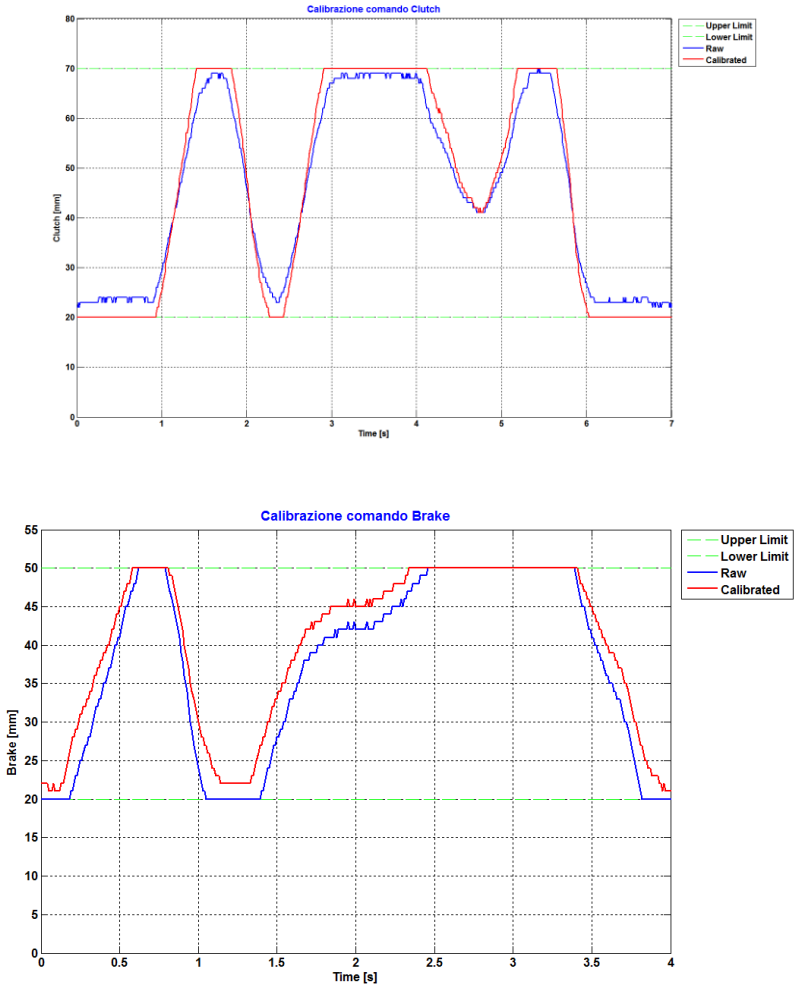


Figura 5.5: Risultati della calibrazione dei comandi della Consolle

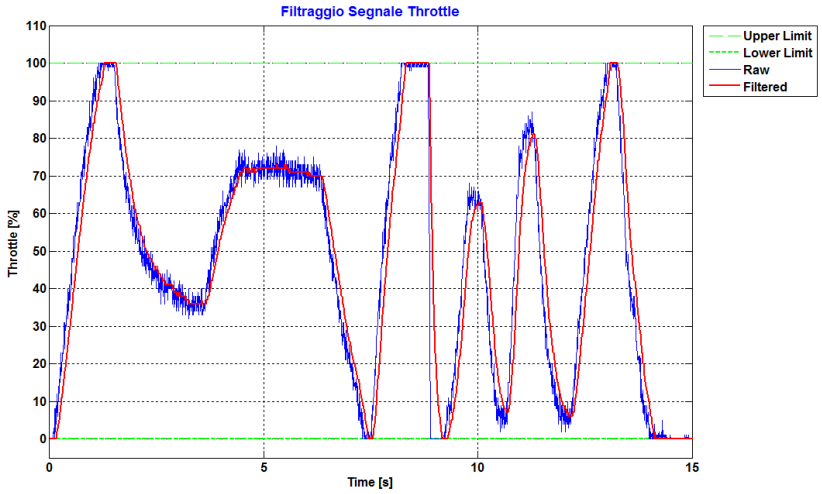
5.1.4 Verifiche del filtraggio

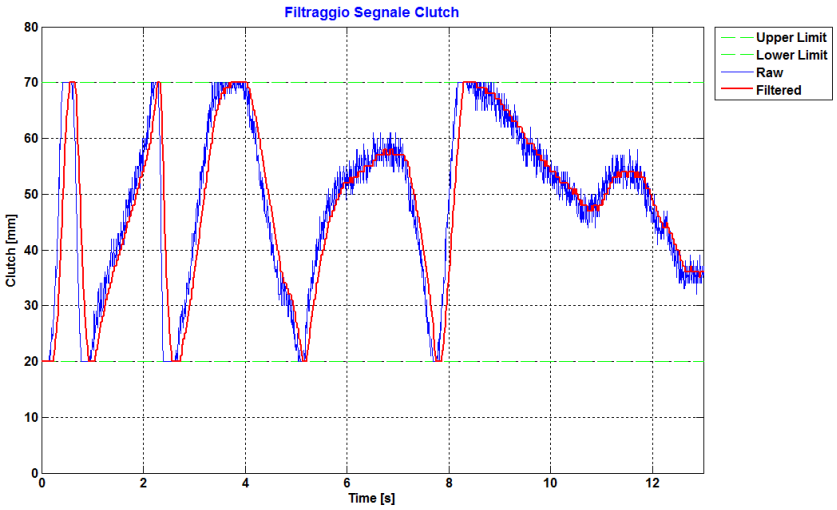
Durante le prove, si è visto che i valori di tensione forniti dai potenziometri e convertiti dalla periferica ADC della scheda STm32 erano parecchio instabili e ciò si rifletteva parecchio sul comportamento degli attuatori, in quanto la movimentazione per essi non cessava mai a causa di variazioni troppo rapide imposte dalla legge di controllo.

Per risolvere tale problema, sostanzialmente dovuto al consistente disturbo dei segnali dei potenziometri, si sono introdotti nel modello di controllo della Consolle dei filtri passa basso del primo ordine impostando la frequenza di taglio a 1,6 Hz.

Nei grafici successivi vengono mostrate le comparazioni tra i segnali originali e i segnali filtrati. Il segnale ottenuto con il filtro implementato risulta soddisfacente dal

punto di vista dell'attenuazione dei disturbi. Il fatto invece che ci sia un ritardo inevitabile su ogni componente filtrata non causa problemi in quanto tale ritardo risulta essere compatibile con la dinamica dell'applicazione.





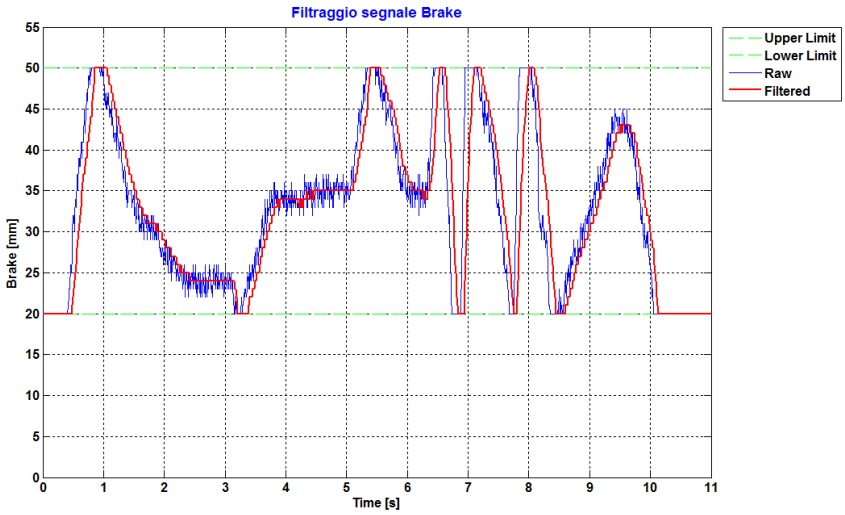


Figura 5.7: Confronto tra segnali filtrati e segnali originali

5.2 Verifica di funzionamento del sistema RideIT

Grazie alla collaborazione dell'Ing. Taccioli, al fine di valutare l'efficacia degli schemi di controllo descritti nel capitolo 4 attinenti al funzionamento del controllore real-time, è stata condotta una campagna di test che prevedevano sia l'impiego degli attuatori LinMot, sia quello delle interfacce Host pc e Consolle connesse all'unità di controllo centrale.

In queste prove ci si è concentrati molto sulla verifica dell'efficacia e della stabilità dei controlli implementati nonché sulla stabilità di connessione fra i vari dispositivi. In particolare si è intervenuti cercando di simulare dei malfunzionamenti mirati o altre condizioni di funzionamento anomale così da osservarne la reazione e consentire la messa a punto di procedure che individuino l'evento e pongano rimedio agli errori o anomalie che possono manifestarsi durante il normale funzionamento.

Il primo problema affrontato è stato quello di perfezionare la comunicazione tra i dispositivi Host-target RT e Consolle-target RT andando a rielaborare le logiche dei modelli relative alle operazioni di scrittura/lettura dei dati sulla rete LAN. In particolare è stata perfezionata una porzione specifica del software RT che si occupa di monitorare continuamente lo stato della connessione UDP tra i nodi e di prendere decisioni volte alla messa in sicurezza del sistema in caso di problemi.

Il passo successivo è stato quello verificare che una qualsiasi operazione eseguita da pc Host o da Consolle fosse effettivamente tramutata dal controllore real-time in

un’attuazione per i motori LinMot, verificando anche come i segnali generati dal controllore venissero elaborati e monitorati a lato delle interfacce utente.
E’ importante evidenziare che in questa fase di test, non avendo avuto a disposizione alcun segnale di centralina motore (ECU), tramite cui avere informazioni relative alla dinamica motore, si è dovuto ricreare tali segnali in modo fittizio, poiché da essi il sistema non può prescindere e la loro mancanza verrebbe diagnosticata come errore. In pratica si è implementato un generatore di segnali nella parte di controllo real-time, simulando in questa maniera qualsiasi lettura utile sulla porta CAN da parte del controllore real-time. La generazione di questi segnali è stata resa il più semplice possibile in modo da poter velocemente verificare le procedure implementate circa la gestione degli stati, dei cambi marcia e del ciclo di guida automatico.
Si è infine verificato che il tempo di esecuzione dell’algoritmo di controllo fosse adeguato affinché, da una parte non venisse sovraccaricato troppo a livello computazionale il lavoro del sistema di controllo real-time, dall’altra non ci fosse troppo ritardo fra le acquisizioni dei segnali e le attuazioni. Il tempo di esecuzione dell’algoritmo di controllo real-time è stato impostato pari a 10 ms.
L’ acquisizione dei dati dalle interfacce utente viene eseguita con un tempo di campionamento di 50 ms, mentre il tempo di esecuzione della scrittura dei dati di feedback verso le interfacce stesse è stato impostato pari a 150 ms.

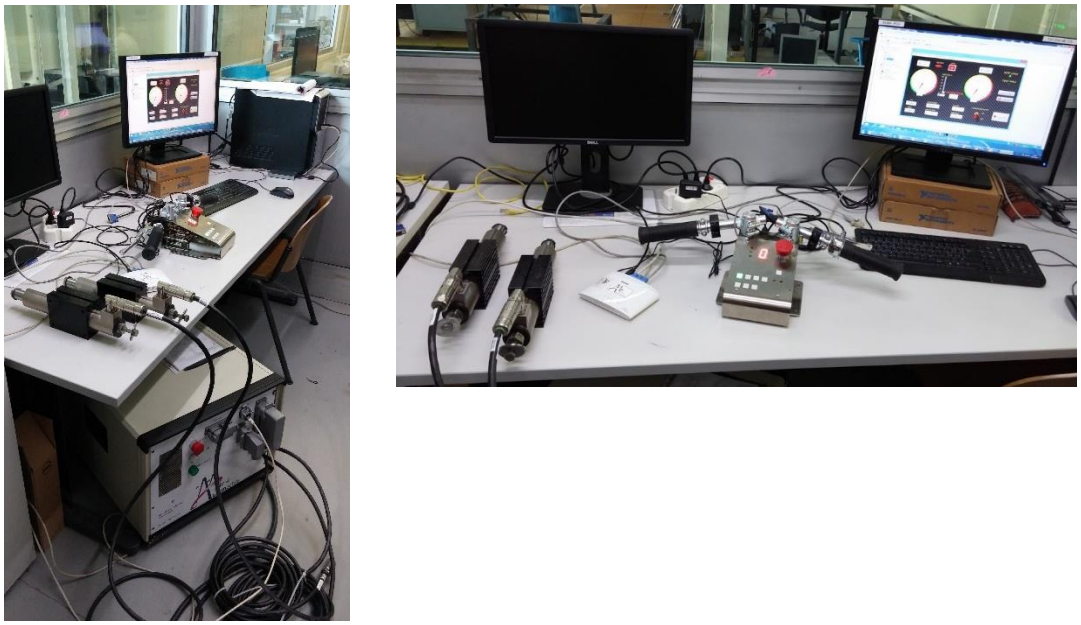


Figura 5.6: fotografie scattate in laboratorio durante lo svolgimento del test

5.3 Sviluppi futuri

Gli sviluppi futuri del sistema RideIT, prima di attendere la disponibilità del banco prova a rulli MV Augusta cui il sistema stesso è destinato, possono riguardare in particolare l'effettuazione di test in laboratorio gestiti in **Hardware in the Loop (HIL)**.

Nel nostro caso, avendo la disponibilità in laboratorio di un microHIL, l'idea potrebbe essere quella di configurarlo in modo da supportare il sistema RideIT nei test orientati a verificarne il funzionamento in maniera più completa e ripetibile.

Una soluzione di questo tipo prevede un'architettura hardware così composta (la figura 5.7 ne riporta una rappresentazione):

- Centralina di controllo motore (ECU);
- Il micro hardware in the loop vero e proprio, ossia un banco prova costituito da componenti hardware reali e componenti simulate via software. Nel nostro caso abbiamo un elaboratore real-time per l'esecuzione del modello della dinamica Motore e Veicolo e in grado di gestire la connessione di hardware fisici come sensori, attuatori, essendo dotato di moduli specifici per l'acquisizione e l'elaborazione di AI, AO, DI, DO e per la comunicazione via CAN;
- Un PC Host per il controllo e la gestione del sistema microHIL;
- Sistema di controllo RideIT con interfaccia utente per l'esecuzione del modello guidatore che gestisce gli azionamenti di acceleratore, frizione, cambio e freno.
- Attuatori LinMot configurati in modo tale da riuscire a leggere sulla rete CANopen la posizione istantanea o in alternativa dotata di sensori di posizione a parte con cui poter realizzare controlli in retroazione relativamente alla gestione della dinamica motoveicolo.

Utilizzando il sistema microHIL è possibile procedere con la validazione delle strategie implementate nel sistema RideIT di supervisione della guida del motoveicolo al banco a rulli consentendo una sperimentazione virtuale efficace anche senza la presenza di un banco a rulli e un motoveicolo.

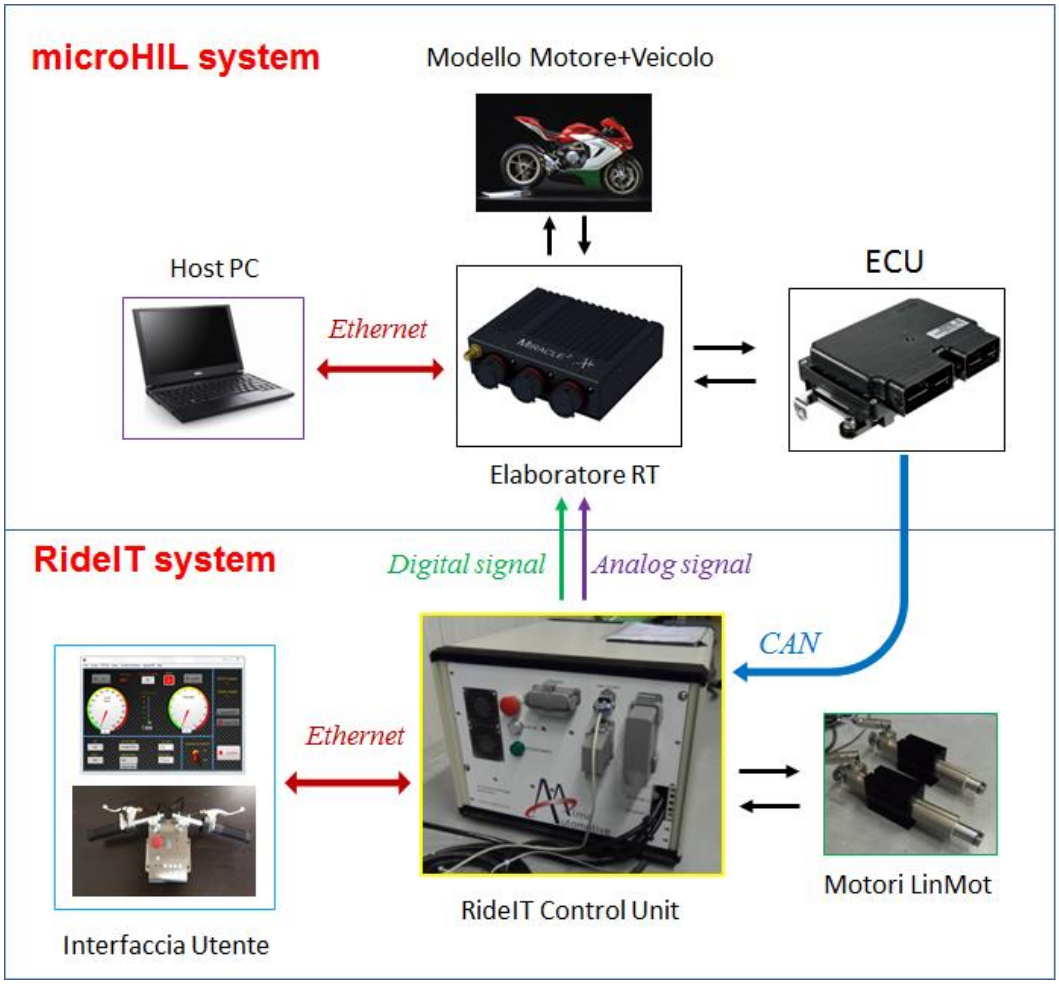


Figura 5.7: architettura del simulatore MHIL per la sperimentazione del prototipo RideIT

Appendice A

Tabella Assegnazione pin STM32F4 Discovery

Panel	Description	Assignment	Limit	HeaderPCB	Type
AI1	Potenziometro acceleratore	PA4	0-3.3V AI	P1-16	ADC 1
AI2	Potenziometro freno	PA5	0-3.3V AI	P1-15	ADC 1
AI3	Potenziometro frizione	PA6	0-3.3V AI	P1-18	ADC 1
DI0	Status CALIBRATION (0)	PE3	0-3.3V	P2-16	Pull Up
DI1	Status MANUAL (1)	PE4	0-3.3V	P2-13	Pull Up
DI2	Status SEMI-AUTO (2)	PE5	0-3.3V	P2-14	Pull Up
DI3	Status AUTOMATIC (3)	PE6	0-3.3V	P2-11	Pull Up
DI4	Pulsante Alarm	PB6	0-3.3V	P2-23	Pull Up
DI5	Upshift	PB2	0-3.3V	P1-24	Pull Up
DI6	Downshift	PB15	0-3.3V	P1-39	Pull Up
DI7	Neutral	PC10	0-3.3V	P2-37	Pull Up
DI8	ClutchOn&BrakeOff	PB7	0-3.3V	P2-24	Pull Up
DI9	ENGstart	PB4	0-3.3V	P2-25	Pull Up
DI10	ENGstop	PB5	0-3.3V	P2-26	Pull Up
DI11	Tasto Select Cal. Pot.	PC9	0-3.3V	P2-46	Pull Up
DI12	Tasto Edit/Confirm Cal. Pot.	PC11	0-3.3V	P2-38	Pull Up
DI13	Pulsante F3 (spare DI)	PB3	0-3.3V	P2-28	Pull Up
DO1	LED_Status ALARM (11)	PD0	0-3.3V	P2-36	Push pull
DO2	LED_Status IDLE (10)	PD1	0-3.3V	P2-33	Push pull
DO3	LED_status CALIBRATION (0)	PD12	0-3.3V	P1-44	Push pull
DO4	LED_Status MANUAL (1)	PD13	0-3.3V	P1-45	Push pull
DO5	LED_Status SEMI-AUTO (2)	PD14	0-3.3V	P1-46	Push pull
DO6	LED_Status AUTOMATIC (3)	PD15	0-3.3V	P1-47	Push pull
DO7	LED_ClutchOn&BrakeOff	PC8	0-3.3V	P2-45	Push pull
DO8	LED A_Display-7seg.	PD2	0-3.3V	P2-34	Push pull
DO9	LED B_Display-7seg.	PD3	0-3.3V	P2-31	Push pull
DO10	LED C_Display-7seg.	PD4	0-3.3V	P2-32	Push pull
DO11	LED D_Display-7seg.	PD5	0-3.3V	P2-29	Push pull
DO12	LED E_Display-7seg.	PD6	0-3.3V	P2-30	Push pull
DO13	LED F_Display-7seg.	PD7	0-3.3V	P2-27	Push pull
DO14	LED G_Display-7seg.	PD8	0-3.3V	P1-40	Push pull
DO15	LED Connection_OK	PD9	0-3.3V	P1-41	Push pull
DO16	LED Speed_out_range	PD10	0-3.3V	P1-42	Push pull
DO17	LED Warning/Error system	PD11	0-3.3V	P1-43	Push pull
DO18	LED tasto Select	PC6	0-3.3V	P2-47	Push pull

DO19	LED tasto Edit/Confirm	PC7	0-3.3V	P2-48	Push pull
DO20	LED F3 (spare DO)	PC12	0-3.3V	P2-35	Push pull
Ethernet_0		PA0			
Ethernet_1		PA1			
Ethernet_2		PA2			
Ethernet_3		PA3			
Ethernet_4		PA7			
Ethernet_5		PA8			
Ethernet_6		PB0			
Ethernet_7		PB1			
Ethernet_8		PB10			
Ethernet_9		PB11			
Ethernet_10		PB12			
Ethernet_11		PB13			
Ethernet_12		PB14			
Ethernet_13		PC1			
Ethernet_14		PC2			
Ethernet_15		PC3			
Ethernet_16		PC4			
Ethernet_17		PC5			
Ethernet_18		PE2			
Ethernet_19		NRST			

Appendice B

Il file di inizializzazione

Il file d'inizializzazione o di lancio, è uno script in "Matlab", attraverso il quale, oltre a importare nel modello Simulink i dati relativi al ciclo automatico, si definiscono determinati parametri del modello stesso.

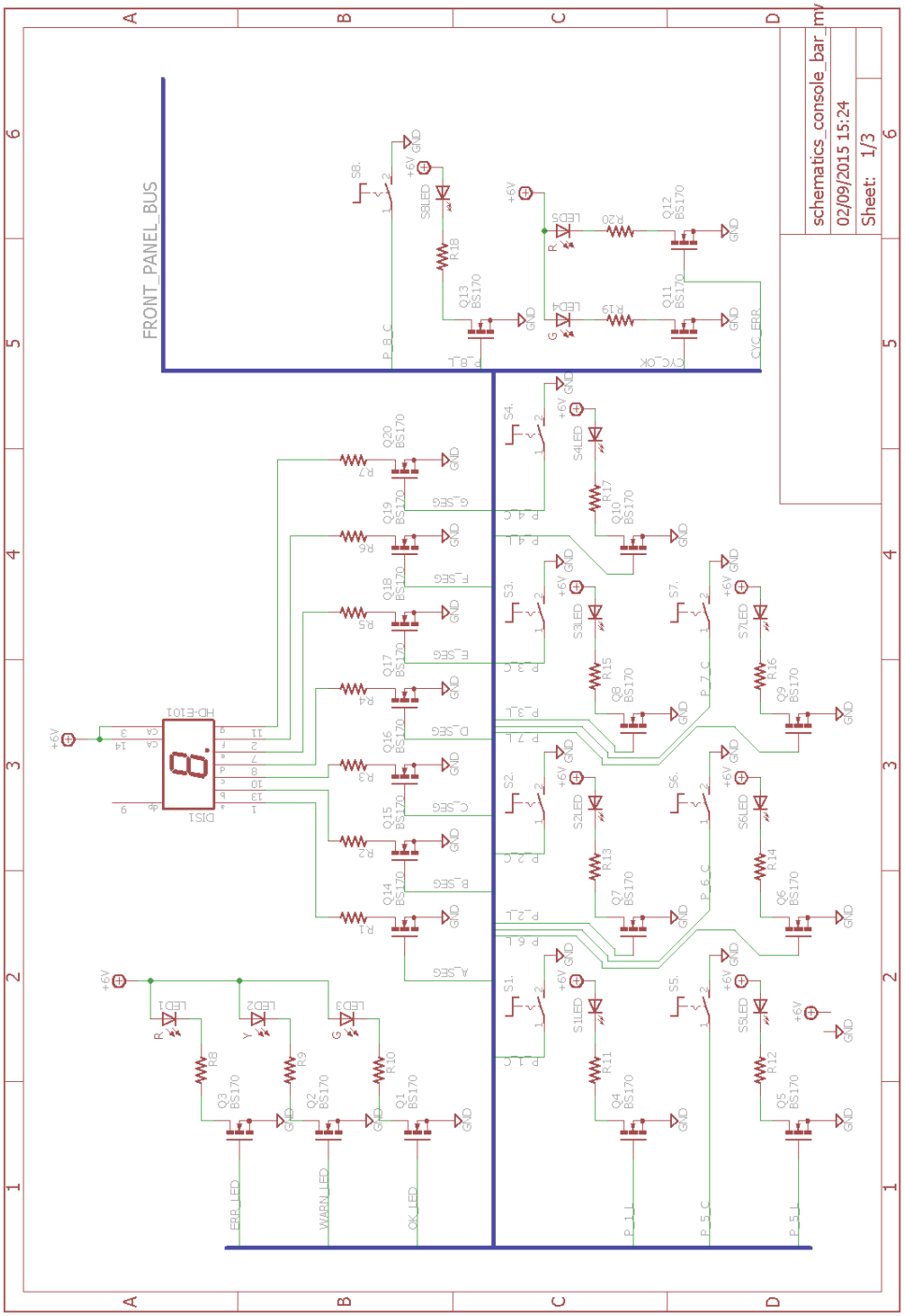
```
%Init_RideIT_Model.m

%Script di inizializzazione dei parametri di simulazione
clc, clear all, close all
warning('off')
step_time=1e-3;

%N.B. vige la regola che i parametri scalari hanno nome che comincia
per minuscola, mentre i parametri vettore hanno nome che comincia
per MAIUSCOLA. Ciò è necessario per una corretta modifica dei para-
metri inline in tempo reale

%DT è il vettore dove impostare tutti i timeout per le cambiate ed è
espresso in secondi: Tempo dopo il quale l'attuatore viene ritirato
in posizione di riposo, anche se la CAN non segnala l'avvenuta cam-
biata
DT(1:2)=[0.45 0.35];

%GSendel=tempo di attesa per il riarmo degli attuatori, prima di ri-
provare a effettuare la cambiata (per adesso uguale per UP/DWN)
gs_ENdel=0.25;

%tempo dopo il quale si porta l'attuatore in posizione neutra
gs_wait=2;

%MASK è il tempo di mascheramento tra una effettuazione di cambiata
e la richiesta di cambiata successiva: potrebbe essere posto uguale
a GSendel(tempo di riarmo), eventualmente tenuto diverso per UP/DWN
mask_UP=gs_ENdel;
mask_DWN=gs_ENdel;

%Gear_fwd, Gear_bckwd, Clutch_fwd, Clutch_bckwd, sono le matrici in
cui vanno impostati i parametri per la movimentazione degli attua-
tori di cambio e frizione nelle differenti situazioni (upshift e ri-
torno, downshift e ritorno, ingranare la folle a partire dalla prima
```

```
o dalla seconda, riportare gli attuatori in posizione neutra). Que-
sti sono: Posizione [mm]; velocità Massima [m/s]; Accelerzione
[m/s^2]; Decelerazione [m/s^2]; Forza Massima [N].

%Parametri del cambio
Gfu(1:5)=[10 0.5 0.7 0.7 50]; % mm, m/s, m/s^2, m/s^2, N
Gbu(1:5)=[30 2.5 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
Gfd(1:5)=[50 0.15 1.5 1.5 50]; % mm, m/s, m/s^2, m/s^2, N
Gbd(1:5)=[30 2.5 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
Gn1(1:5)=[18 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
Gn2(1:5)=[42 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
GbN(1:5)=[30 0.7 7 7 50]; % mm, m/s, m/s^2, m/s^2, N

%Parametri della frizione
Cfu(1:5)=[20 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
Cbu(1:5)=[70 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
Cfd(1:5)=[20 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
Cbd(1:5)=[70 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
Cn1(1:5)=[20 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
Cn2(1:5)=[20 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
CbN(1:5)=[70 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N

%Parametri del freno
Brake_release=[20 1 1 1 50]; % mm, m/s, m/s^2, m/s^2, N
%%%%%%%%%%INIZIALIZZAZIONE VETTORE DEI TARGET (REQ)
Default_values=zeros(1,4);

%Costanti per il Controllore PID che agisce sulla manopola in base
%all'errore di velocità
kp=2;
ki=0.05;
kd=0;

%PARAMETRI CAMBIO
SD_Primary=[27 54];
SD_Secondary=[15 39];
SD_Gear1(1:6)=[17 16 20 22 22 23];
SD_Gear2(1:6)=[37 31 33 32 28 26];
rpm_lim=18000; %RPM limitatore RATEO IN SCALATA

%%%%PARAMETRI PER GESTIONE START, ovvero i vettori tempo e posizione
tramite cui nel modello si costruiscono le lookup table per la ge-
stione di frizione e manopola, a fine partenza. Come già visto du-
rante l'analisi del modello, si può scegliere tra due generi di pro-
cedure per lo START, una più dolce, l'altra più aggressiva.
Time_CLUTCH_start_1=[0 0.5 1.6 1.8 3.1 3.5 4.5];%[0:step_time:4.5-
step_time]';
Time_CLUTCH_start_2=[0 1 3 3.5 8.5
8.7];%ime_clutch_start_2=[0:step_time:5.4-step_time]';
CLUTCH_start_1=[Cfd(1); Cfd(1);69; 69; 80; 80;94];
```

```
CLUTCH_start_2=[Cfd(1); Cfd(1);68; 68; 94;94];
Time_THROTTLE_start_1=[0;0.5;4.5];%[0:step_time:4.5-step_time]';
Time_THROTTLE_start_2=[0;1;3.5;5.8;8.7];%[0:step_time:5.4-
step_time]';
THROTTLE_start_1=[0;100;100];%[linspace(0,100,0.5/step_time)';
ones(4/step_time,1)*100];
THROTTLE_start_2=[0;10;15;30;30];%[linspace(0,15,1/step_time)';
ones(1.6/step_time,1)*15; linspace(15,30,1.3/step_time)';
ones(1.5/step_time,1)*30];
timeout_start=1;
rampTOstart_THROTTLE_duration=3;

%%%%PARAMETRI START CONDITION da soddisfare per simulare la partenza
(consenso a mettere la prima marcia)
t_oil_min=65;
t_H2O_min=60;
throttle_max=1;
rpm_max=4500;

%%%%PARAMETRI SAFETY ENGINE per la salvaguardia del motore.
rpm_lim_safety=17100; %limite rpm
deltarpm=300; % delta positivo sottratto a limite rpm
time_rpm_lim_safety=0.25; %tempo limitatore
dec_THROTTLE=30; %decremento farfalla

%%%PARAMETRO RAMPA di passaggio in STATUS AUTOMATICO, ovvero il va-
lore della durata della rampa di raccordo nel passaggio a stato au-
tomatico.

rampTOauto_duration=2; %in secondi

%%% SWITCH per simulazione procedura di partenza (1 YES / 0 NO)
switch_START_THROTTLE=0;

%%%PARAMETRO UTILE ALLA GESTIONE DELLA FRIZIONE IN PARTENZA
decimate=1;
```

Appendice C

Schema circuitale della Consolle



Bibliografia

[1] Michele Taccioli, *Evoluzione di un Sistema di Automazione della Guida di Motoveicoli per Test su Banco a Rulli*.

[2] Davide Galli, *Sviluppo hardware e software per automatizzare la guida di una motoGP su banco a rulli.*

Siti Internet consultati

- https://www.st.com/web/en/resource/technical/document/datasheet/DM00037051.pdf
- http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf
- https://www.aimagin.com/downloads/dl/file/id/104/amg_f4connect_datasheet.pdf
- http://aimagin.com/blog/waijung-tutorials
- http://www.ni.com/datasheet/pdf/en/ds-21
- http://sine.ni.com/nips/cds/view/p/lang/it/nid/211620
- https://www.aimagin.com/downloads/dl/file/id/131/dp83848c_datasheet.pdf
- http://it.rs-online.com
- https://it.wikipedia.org
- http://www.mathworks.com
- http://www.automotion.com.br/downloads/linmot-p01-48x240-e-v2-21.pdf
- http://www.linmot.cn/fileadmin/doc/InstallationGuides/ps_S01-72-1000_e_recent.pdf
- http://www.profilex.be/pdf/linmot/ServoControllers/B1100_Controllers_GE.pdf