FRT
FIRENZE RACE TEAM

FR-24
COMBUSTION &
DRIVERLESS

Sviluppo e Analisi Dettagliata del Software per la Main Control Unit nelle Applicazioni di Formula SAE

Candidati
Lorenzo Porcheddu
Emanuele Nencioni

Docente
Prof.ssa Laura Carnevali

UNIVERSITÀ
DEGLI STUDI
FIRENZE

DINFO
DIPARTIMENTO DI
INGEGNERIA DELL'INFORMAZIONE

Sommario

Introduzione

Il Firenze Race Team è la squadra ufficiale di Formula SAE e Formula Student dell'Università degli Studi di Firenze.

È un laboratorio didattico aperto agli studenti di tutte le facoltà dell'ateneo con lo scopo di progettare e realizzare una monoposto da competizione.

Visti i prossimi cambiamenti di regolamento delle competizioni di Formula Student e il futuro del mercato automobilistico, il Team ha iniziato a sviluppare il progetto di **auto a guida autonoma**.

L'obiettivo principale è quello di partecipare nel 2024 alla competizione di FS-East con la vettura a combustione, per poi scendere in pista nella categoria 1D di Formula ATA con la vettura *driverless*.

Canditati

Emanuele Nencioni: Head of Autonomous Department – Vision & System Integration
Lorenzo Porcheddu: Head of Electronics & Controls Department

Presentazione del Progetto e Obiettivi

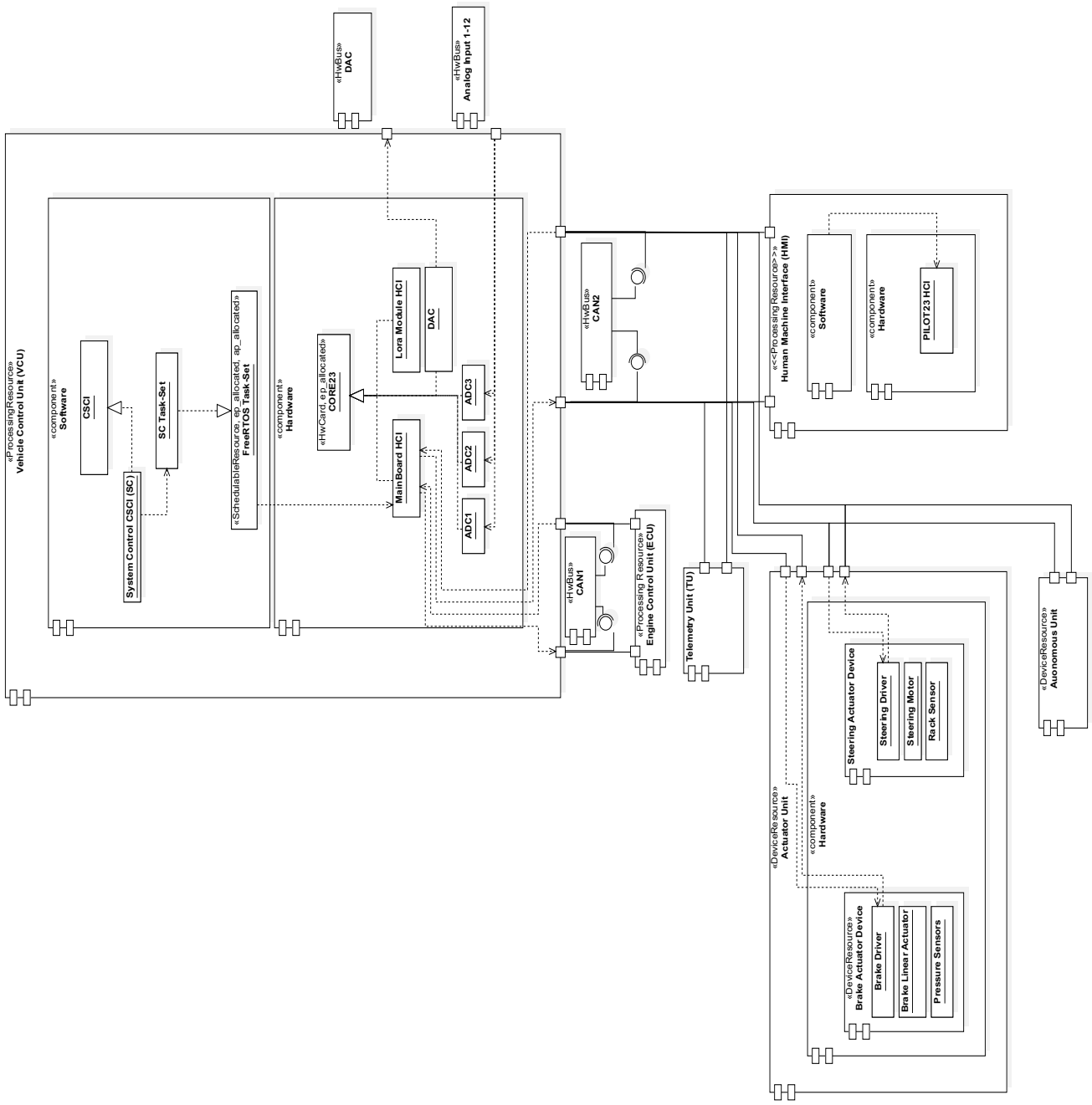
Nel nostro progetto per la Formula Student, stiamo lavorando a un software destinato a una scheda elettronica per auto da competizione. Quest'auto mescola elementi tradizionali con la guida autonoma. Il nostro obiettivo è semplice: sviluppare un sistema di controllo che gestisca motore, velocità, sterzo, freni, e non solo, per migliorare le prestazioni e assicurare la sicurezza.

La sfida più grande è fare in modo che l'auto possa navigare da sola, seguendo regole precise. Questo implica l'uso di algoritmi di intelligenza artificiale e sensori moderni per capire l'ambiente circostante e prendere decisioni di guida autonome. La nostra scheda elettronica deve fare da tramite tra il computer centrale dell'auto e i suoi componenti, come il sistema di alimentazione e i sensori, interpretando i dati raccolti e rispondendo prontamente per mantenere tutto efficiente e sicuro.

Anche l'interazione con il pilota è fondamentale. Il nostro software dovrà fornire aggiornamenti in tempo reale sullo stato dell'auto e permettere al pilota di intervenire se necessario. Ci stiamo concentrando su un'interfaccia utente che sia chiara e facile da usare.

In breve, stiamo cercando di creare un sistema che permetta a un'auto a combustione di guidarsi da sola in una competizione, tenendo sempre a mente l'importanza della sicurezza e delle prestazioni. È una sfida interessante che richiede di mettere insieme conoscenze di software, elettronica e intelligenza artificiale.

Architettura del Veicolo



Come si può vedere dal diagramma di alto livello, la nostra auto integra diverse centraline elettroniche. Sono presenti anche due linee CAN bus per interconnettere il tutto, una dedicata principalmente al motore e l'altra invece al sistema autonomo. Questa suddivisione si rende necessaria data la limitata larghezza di banda disponibile, pari a 1Mbit/s di massimo teorico per ogni linea. In realtà si cerca di non superare il 50% della banda in condizioni normali per evitare la saturazione del bus e il conseguente rallentamento di tutti i messaggi.

Di seguito andiamo ad elencare i componenti del nostro sistema:

- CORE23: si tratta della MCU (Main Control Unit), ovvero della scheda principale di controllo del veicolo. Tutte le altre centraline fanno sempre riferimento a lei per gestire le funzioni interconnesse. Possiede 40 tra ingressi e uscite, analogiche e digitali, che ci permettono di collegare una buona quantità di sensori. Avendo entrambe le linee CAN bus, può essere in grado di leggere una grande quantità di dati, anche dai sensori non collegati direttamente a lei. Per esempio, può leggere la temperatura del motore dalla ECU (Engine Control Unit) e inviarla alla scheda del volante, la PILOT23, per far sì che il pilota possa monitorarla. Viceversa, può leggere il segnale proveniente dal pedale dell'acceleratore e inviarlo alla ECU, che poi lo elaborerà secondo varie strategie per attuare la valvola a farfalla. Su questa scheda si concentrerà lo sviluppo del software, quindi per ora non ci dilunghiamo molto.
- PILOT23: si tratta della HMI del veicolo, infatti è posizionata dentro il volante e si occupa principalmente di gestire l'interazione con il pilota. Sul volante è posto un display, attraverso il quale si possono visualizzare varie informazioni come marcia inserita, velocità, giri motore, temperatura dell'acqua, tempo su giro, ecc. Inoltre, si occupa di leggere i pulsanti ma soprattutto i paddle e la leva della frizione, fondamentali per interagire con il cambio.
- MekTronic MKE7: si tratta della ECU (Engine Control Unit). Come suggerisce il nome, è la centralina che si occupa della gestione del motore a combustione interna; ad essa sono infatti collegati tutti i sensori del motore, come temperatura acqua, pressione turbo, ecc. Possiede una linea CAN per l'interfacciamento con il resto del veicolo, attraverso la quale invia tutta la telemetria del motore in tempo reale.
- Driver Freno: si tratta della scheda atta a gestire il motore del freno, è collegata alla linea CAN per ricevere istruzioni ed inviare dati sul suo funzionamento.
- Driver Sterzo: si tratta della scheda atta a gestire il motore dello sterzo, è collegata alla linea CAN per ricevere istruzioni ed inviare dati sul suo funzionamento.
- AIM EVO 5: si tratta della scheda che ha il compito di registrare e memorizzare tutte le informazioni del veicolo, in tempo reale. Memorizza tutti i dati inviati su entrambe le linee CAN, inoltre ha diversi sensori aggiuntivi collegati, i quali vengono letti e memorizzati. Alla bisogna è possibile inviare i dati di questi sensori sulle linee CAN.
- PC autonomo: si tratta di un PC industriale rugged, in grado di eseguire tantissime operazioni riguardanti il sistema autonomo. Attualmente esegue ROS1 su Ubuntu 20.04, ma si progetta un futuro update a ROS2 per migliorare prestazioni e compatibilità. Monta un Intel Core i7-9700 e una scheda video NVIDIA GTX1650.

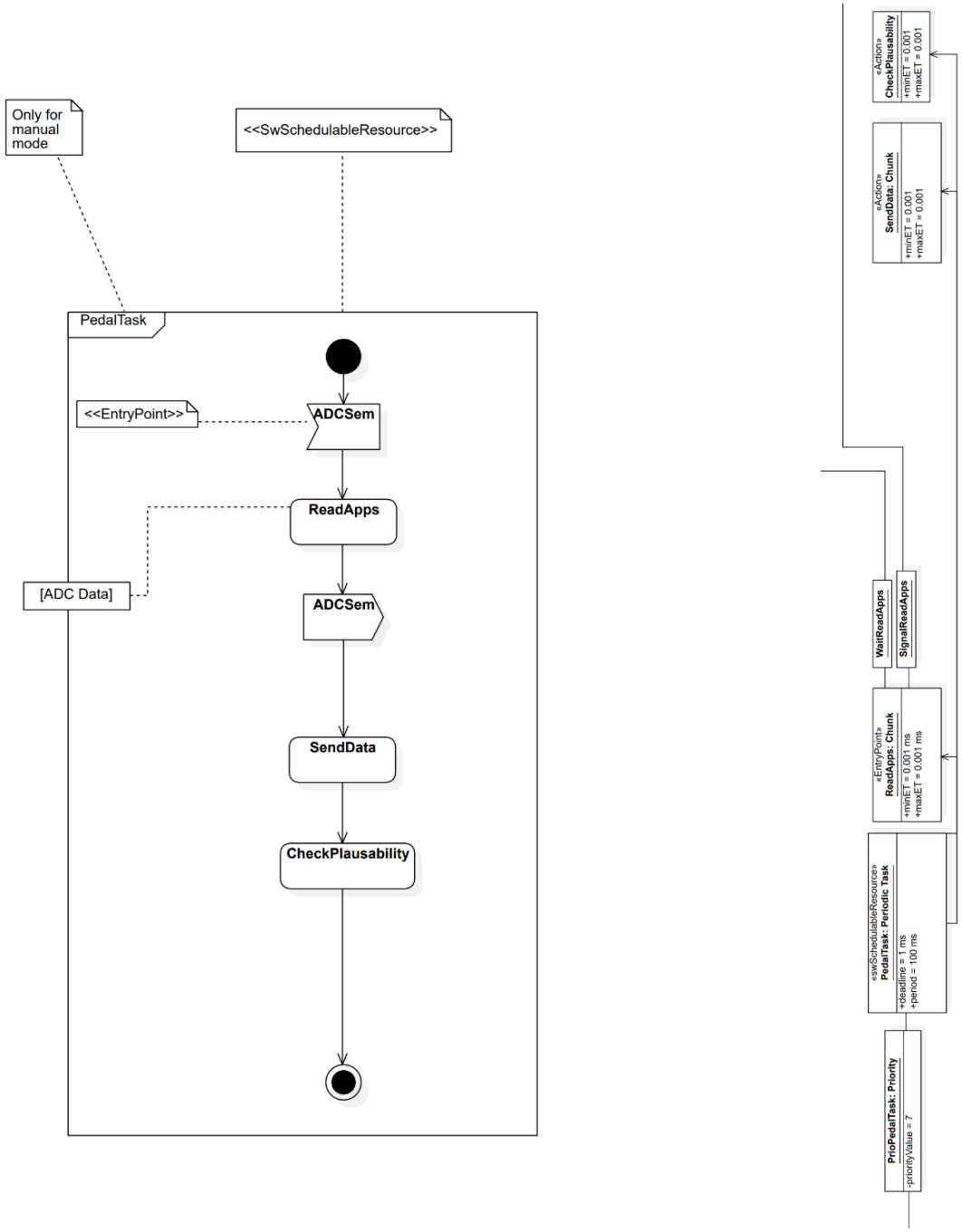
Task set e Activity Diagram

Abbiamo quindi redatto un Activity Diagram, usando il software StarUML.
Data la dimensione del diagramma stesso, lo alleghiamo al presente report.

Di seguito, andiamo a descrivere i singoli task e le relative funzionalità

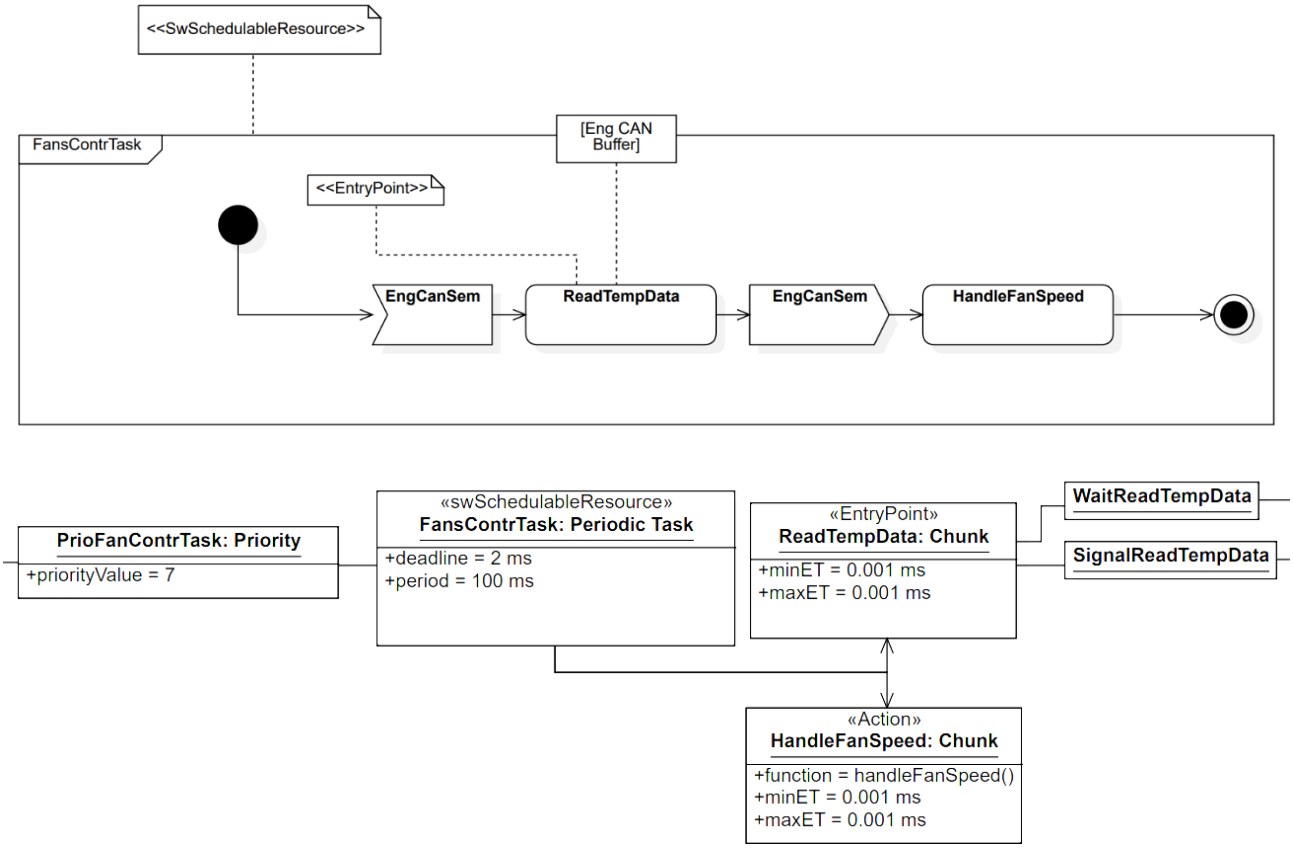
Pedal Task

Si tratta del task che si occupa della gestione del comando acceleratore, inviato poi al motore a combustione. Funziona solamente durante la modalità manuale, perché durante quella autonoma si utilizza un altro task. Si prende il semaforo sull'ADC per leggere il pedale dell'acceleratore, mandiamo quindi i dati alla centralina motore attraverso il DAC. L'ultima cosa che deve fare è un controllo sulla plausibilità della lettura, per motivi di sicurezza. Leggendo due valori, se vediamo un discostamento superiore al 10% si va in errore.



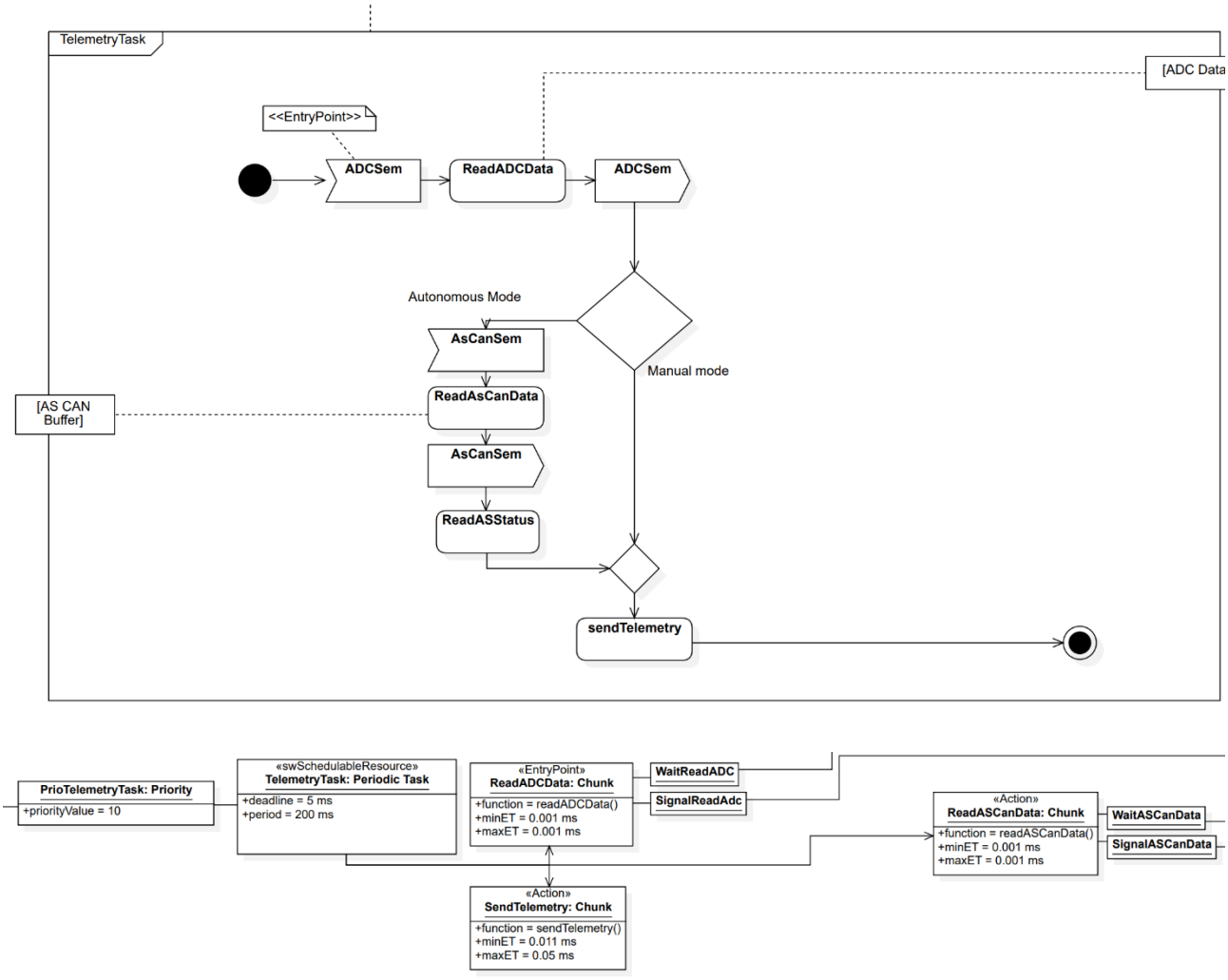
Fan Control Task

Questo task si occupa di gestire le ventole del radiatore e dell'intercooler, tramite modulazione PWM. Prendendo il semaforo della CAN motore leggiamo entrambe le temperature; seguendo quindi una strategia, come per esempio una rampa, andiamo a impostare il PWM in uscita. Il tutto dovrà essere gestito nel modo più efficiente possibile perché le ventole hanno un consumo energetico elevato che potrebbe portare all'esaurimento della batteria se utilizzate in maniera eccessiva.



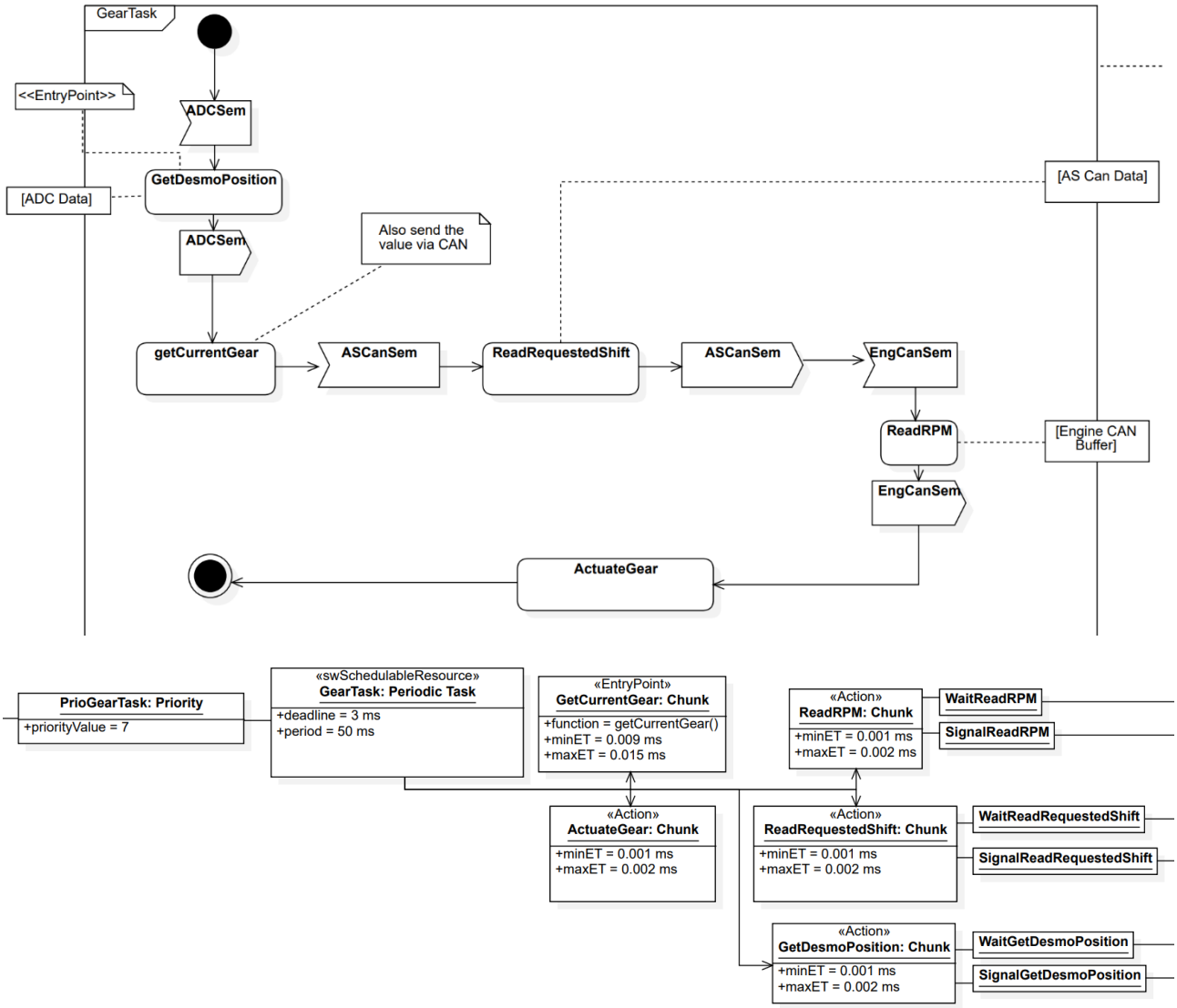
Telemetry Task

Si tratta del task che ci permette di salvare ed inviare tutti i dati di telemetria. Tramite protocollo LoRa è possibile inviare i dati ad una stazione di terra a distanza, quindi, può sapere come si sta comportando la vettura in tempo reale, riconoscendo eventuali problematiche attraverso la sensoristica. I dati vengono salvati anche nella vettura avendo a disposizione un logger AIM Evo 5.
La telemetria è composta principalmente dalle letture dei sensori, quindi degli ADC, ma anche i dati del sistema autonomo, nel caso di modalità driverless, tra i quali lo stato e altri valori di diagnostica. Se dovesse servire si potrebbero inviare anche altri dati riguardanti, per esempio, il motore a combustione oppure le sospensioni che non sono lette dalla CORE direttamente, ma collegate al nostro logger AIM Evo 5.



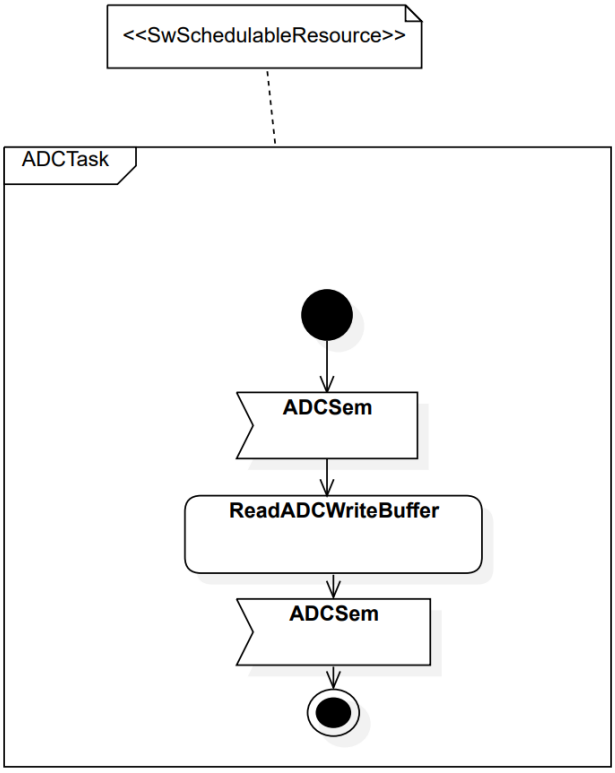
Gear Task

Si tratta del task che gestisce il cambio del motore a combustione interna. Per prima cosa andiamo a sempre leggere la marcia corrente, tramite lettura ADC e semaforo, che viene subito mandata al pilota a schermo. Controlliamo quindi se il pilota ha richiesto un cambio marcia, leggiamo il numero di giri e poi andiamo ad azionare gli attuatori per effettuare la cambiata. La lettura del numero di giri si rende necessaria ad ogni cambiata, perché ad un basso numero di giri non possiamo effettuare un'operazione di quick-shift, ovvero la cambiata rapida senza utilizzare la frizione. Sono quindi differenziate due modalità di cambiata, una con e una senza l'utilizzo della frizione.



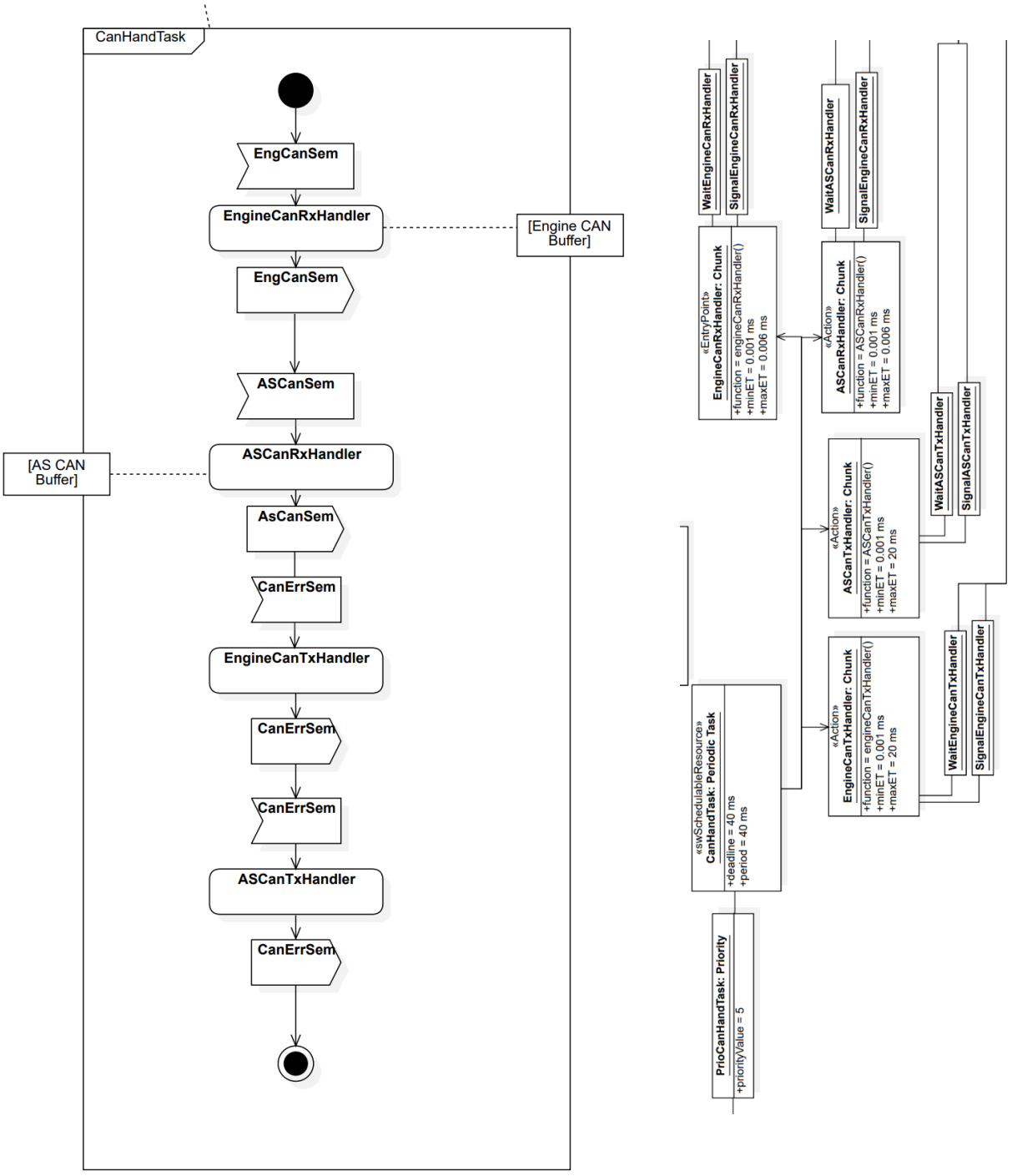
ADC Task

Questo task è tanto semplice quanto fondamentale, ci permette infatti di leggere gli ADC in ingresso al microcontrollore, eseguendo un polling sui dati letti in continuazione (Continuous Conversion Mode). L'ADC quindi converte sempre i dati in ingresso e li memorizza in un buffer, in modalità di funzionamento circolare. I dati vecchi vengono rimpiazzati continuamente da quelli più nuovi, contemporaneamente noi li leggiamo tramite polling. In questo modo, si crea un semplice buffer al quale tutti i task possono accedere tramite semaforo, senza creare problemi di lettura o di accesso ai dati concorrente. L'utilizzo di altre modalità quale il polling diretto, l'interrupt o il DMA con interrupt avrebbe rallentato notevolmente l'esecuzione del codice, o causato cambi di contesto superflui.



CAN Handler Task

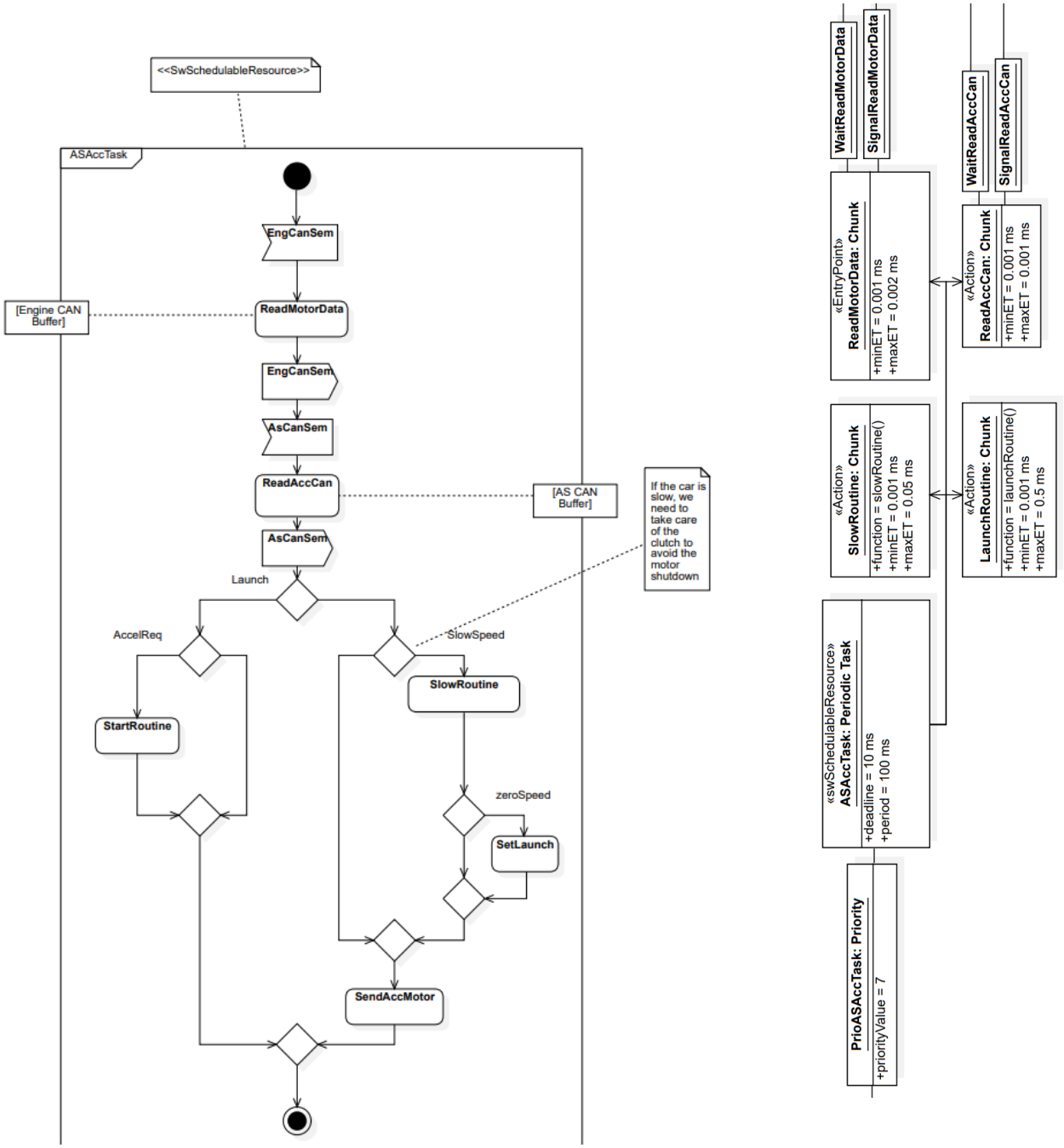
Si tratta del task che gestisce la comunicazione di entrambe le linee CAN. Anche in questo caso, si è scelto di non utilizzare l'interrupt per avere un codice più deterministico e per evitare anche un sovraccarico del processore, nel caso ci fossero una grande quantità di messaggi in arrivo. Il task gestisce in autonomia le code in ingresso e in uscita, tramite semafori, in modo tale da evitare situazioni di conflitto. Avendo tanti messaggi da leggere ed



inviare via CAN, abbiamo scelto un periodo di soli 40 ms per questo task. Data inoltre la criticità dei messaggi in arrivo dal CAN bus, soprattutto in modalità Driverless, la priorità di questo task è stata innalzata. Nella figura sottostante si possono vedere i vari messaggi di ingresso/uscita della scheda.

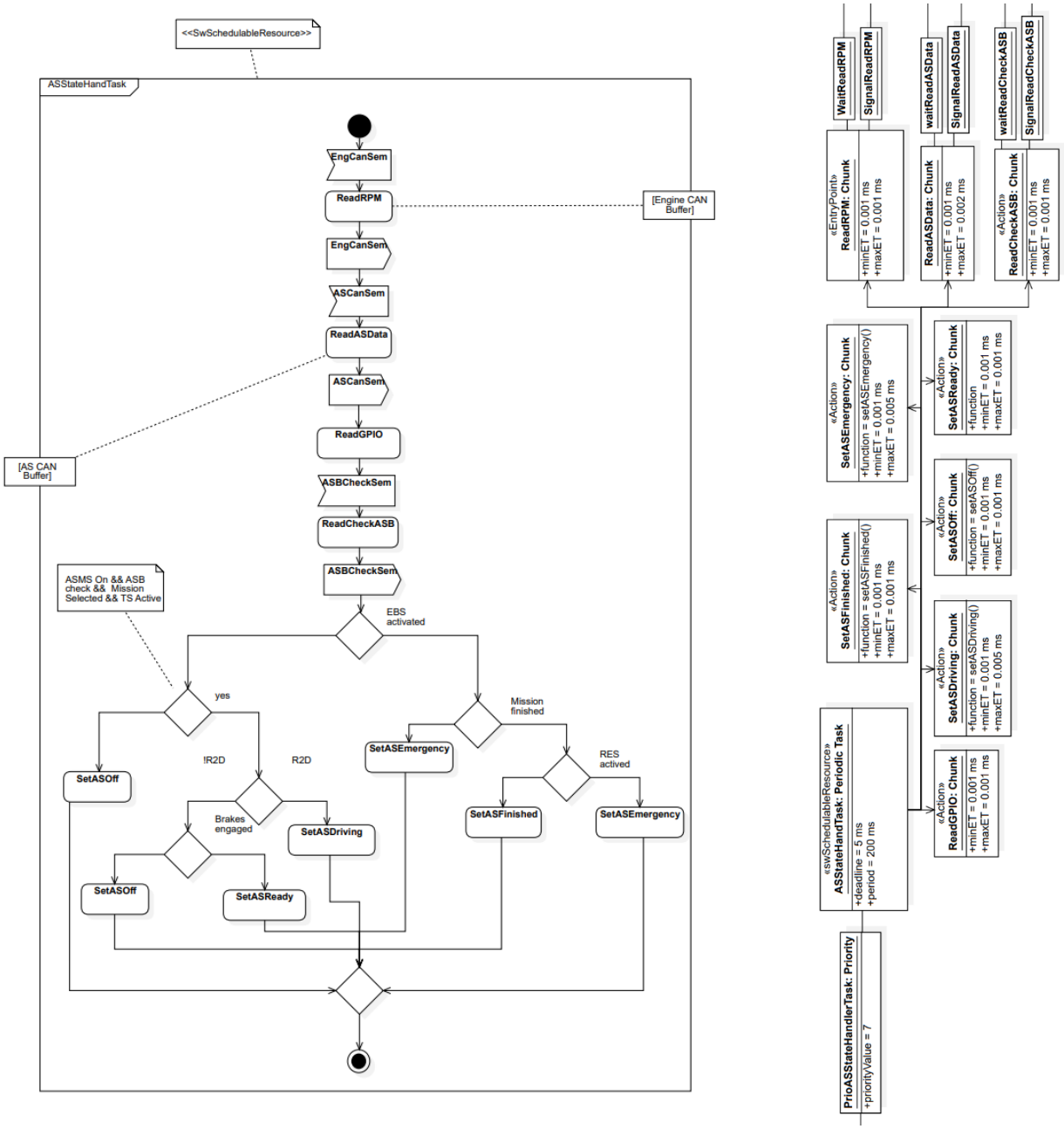
AS Accelerator Task

Si tratta del task che gestisce il comando acceleratore nel caso driverless, è infatti il PC autonomo in questo caso a fornirci l'indicazione riguardante la potenza da erogare. Il PC invia il dato via CAN e questo viene inviato alla centralina motore tramite DAC. Non essendoci il pilota umano, il task si occupa anche della gestione della frizione nelle partenze, ma anche quando la velocità e il numero di giri del motore sono bassi, per mantenere il motore acceso. I



dati di tempo richiesti dai vari chunk non sono purtroppo calcolabili in quanto non è possibile esegure e testare le varie procedure senza ancora avere la vettura a disposizione, ne è stata comunque data una stima all'interno delle timeline. Si suppone che il periodo di 100 ms sia più che sufficiente per garantire un corretto controllo dell'acceleratore.

AS State Handler Task



Si tratta del task che gestisce lo stato del sistema autonomo; infatti, durante la competizione driverless, la vettura deve rispettare una serie di stati ben definiti dal regolamento.
Dall'Activity Diagram si nota una divisione in due parti:

- **Lettura dati**, entrando nelle rispettive sezioni critiche, vengono letti e memorizzati in locale tutti i dati di interesse, quali stato del motore, stato del sistema frenante, etc.
- **Calcolo stato,** una volta raccolte tutte le informazioni, si calcola lo stato tramite una serie di if. Il diagramma comprende la totalità dei rami poiché fondamentale capire come ogni singolo stato viene calcolato e cosa ciò comporta.

Si vedano gli stati in particolare:

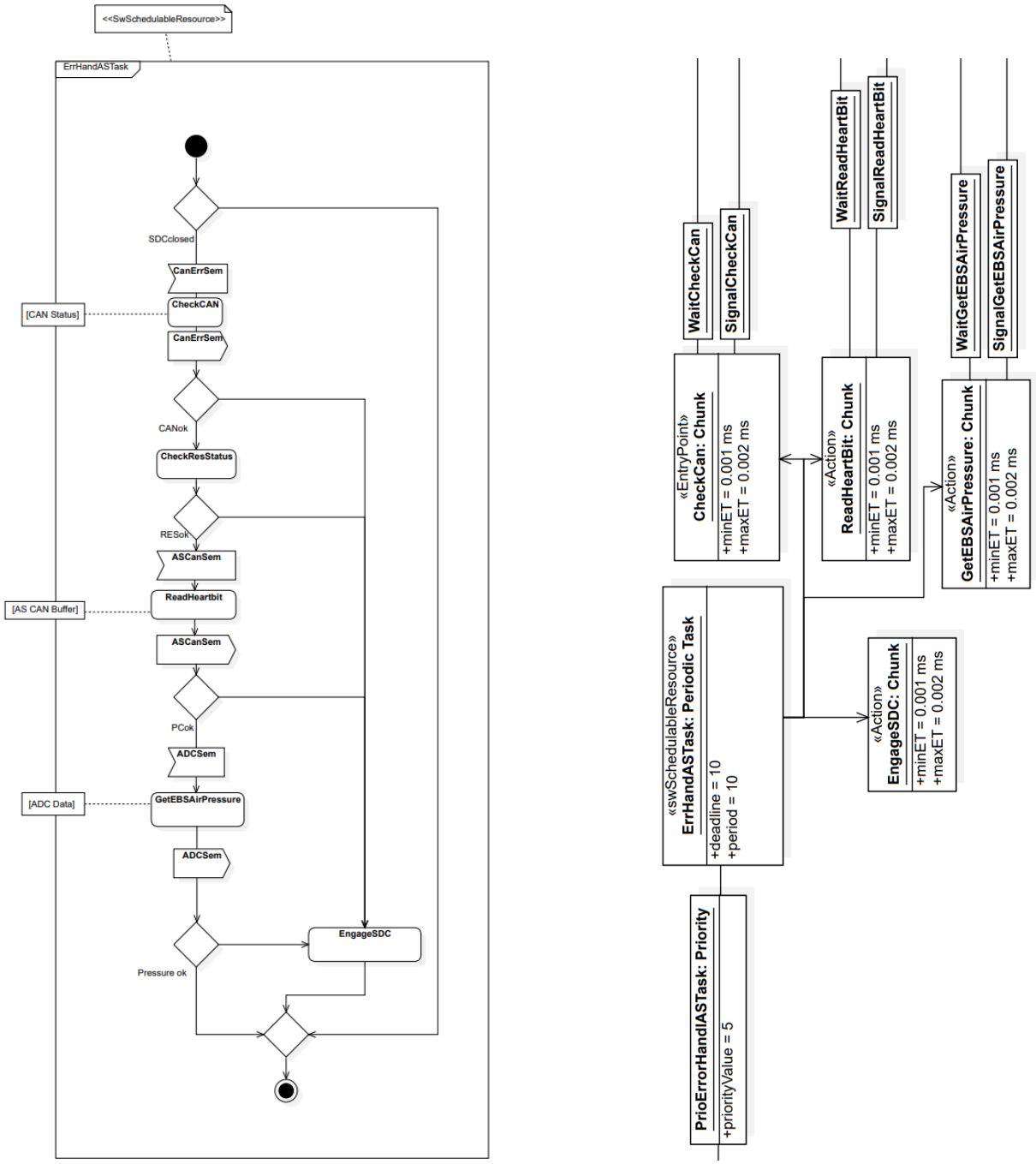
- **ASOff**, attualmente l'intero sistema autonomo non ha ancora eseguito uno dei seguenti passaggi o non sono state fatte delle operazioni manuali:

    - attivazione dell'Autonomous System Master Switch (ASMS), che permette di accendere tutti i sistemi di attuazione.
    - controllo dello stato dell'EBS (Emergency Brake System, ovvero il sistema di frenata di emergenza)
    - la missione non è stata selezionata
    - il motore non è stato acceso
- **ASReady**, tutte le operazioni precedentemente effettuate sono state attivate. Il task a questo punto precede ad aspettare l'arrivo del Signal GO da un addetto a distanza. Questo segnale viene inviato alla vettura tramite un sistema detto Remote Emergency System (RES). Il signal GO è ricevuto dalla scheda tramite un semplice GPIO. Una volta che il task rileva il signal GO , procede ad inviare all'Autonomous PC il messaggio che gli permette di prendere il controllo del veicolo.
- **ASDriving**, per essere in questo stato è sufficiente essere anche in ASReady con in più la marcia inserita, perciò è lo stato in cui si entra dopo che l'Autonomous PC prende il controllo della vettura e ci rimane fino alla fine della missione o una eventuale emergenza.
- **ASEmergency**, è stato rilevato un ingaggio del sistema di frenata di emergenza (EBS), dovuto a problemi riscontrati durante la navigazione del sistema autonomo oppure da un addetto che a distanza è capace di attivare l'EBS tramite il sistema RES. In questo stato deve essere inoltre attivata una Sirena di emergenza per 10 secondi.
- **ASFinished**, La missione è stata conclusa con successo.

Lo stato viene segnalato all'esterno della vettura tramite dei LED RGB posti in modo tale da essere visibili da ogni direzione. La particolare combinazione di flash o colore, definita dal regolamento della competizione della Formula SAE, indica lo stato in tempo reale di tutto il Sistema Autonomo.
Il task è periodico ciò permette di ricalcolare lo stato del sistema autonomo ogni 200 ms.

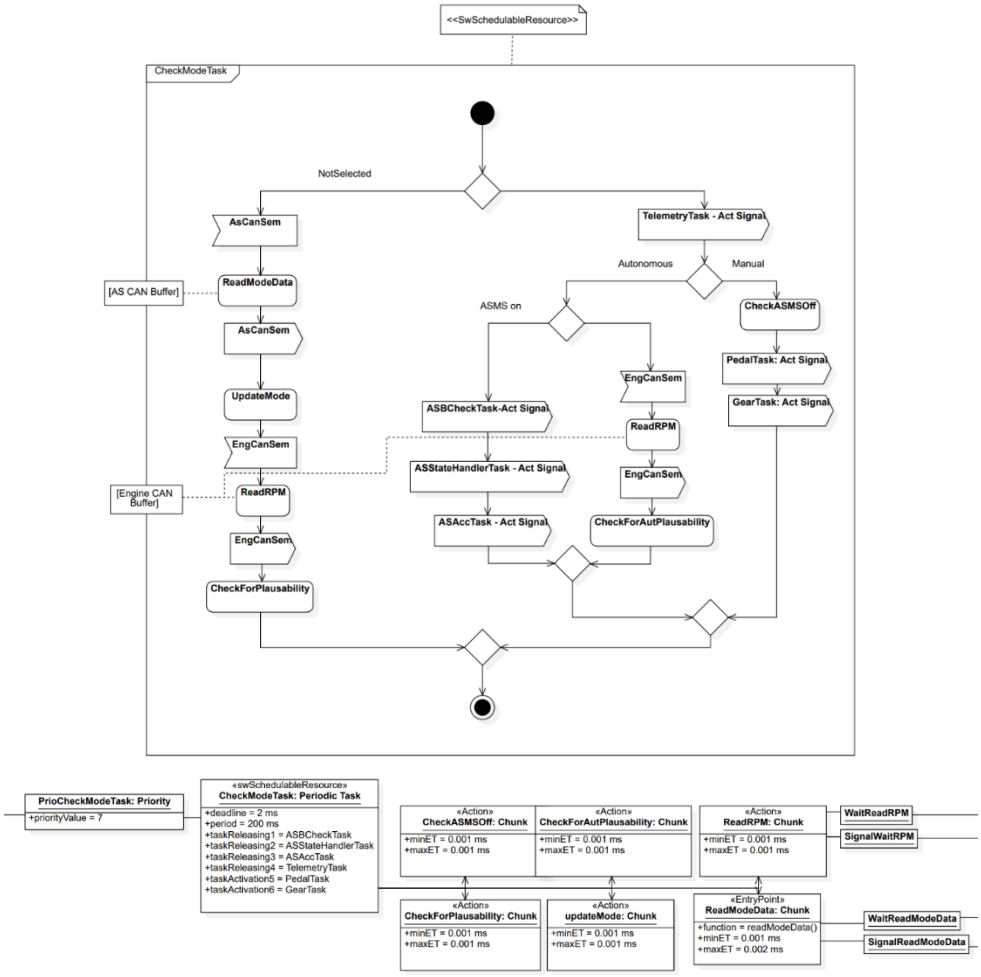
AS Error Handler Task

Si tratta del task di sicurezza della vettura, in grado di ingaggiare l'Emergency Brake System (EBS), ovvero il sistema di frenata di emergenza attuato tramite sistema pneumatico. Questo task controlla lo stato delle linee CAN, che ci sia pressione sufficiente per una frenata nel sistema EBS e che il sistema autonomo invii un Heartbit ogni 200 ms. Tutti controlli atti a garantire una risposta immediata in caso di fallimenti di una di queste parti fondamentali del veicolo. Ingaggiando l'EBS si spegne immediatamente anche il motore e la pompa della benzina collegate allo stesso circuito detto ShutDown Circuit (SDC). Task periodico con periodo molto basso per garantire minimo tempo di risposta in caso di fallimenti critici che potrebbero portare la vettura ad incidentarsi.



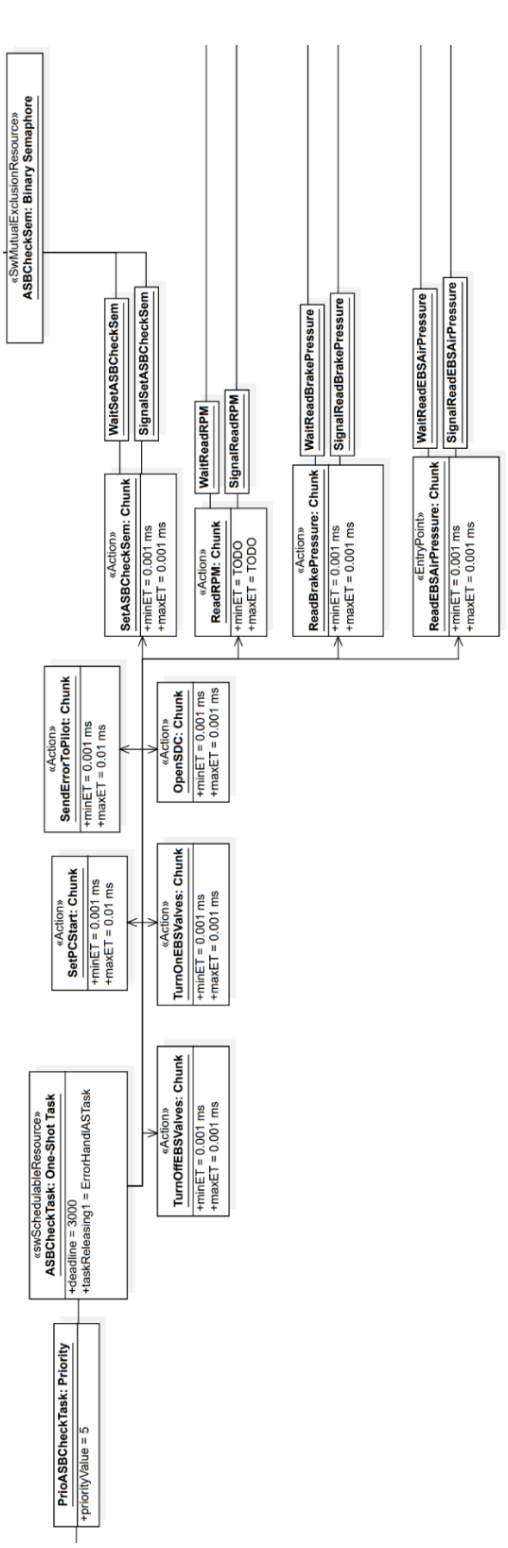
Check Mode Task

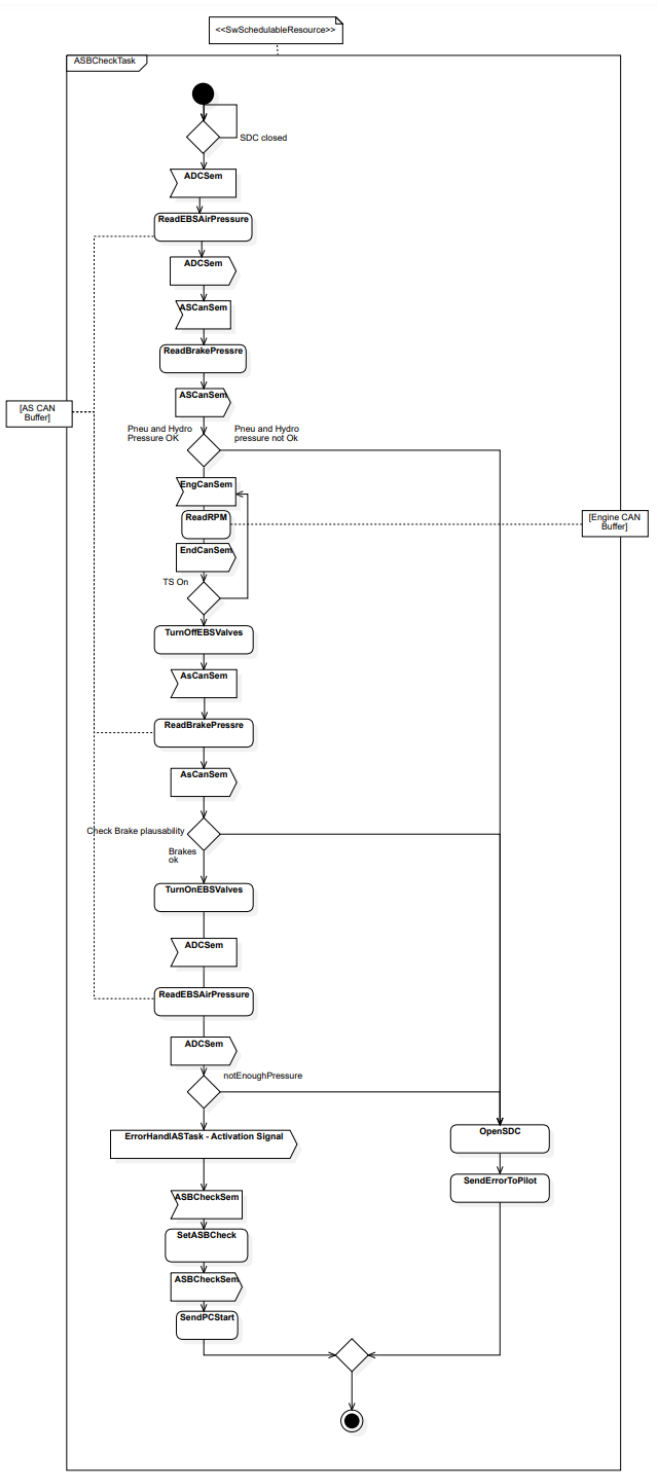
Dato che il veicolo deve comunque poter essere guidato da un pilota vero, è stato necessario costruire un task che permettesse di cambiare la modalità di funzionamento della vettura: guida manuale o driverless. Questo ha permesso di ridurre il numero di task attivi simultaneamente potendo disattivare i task della guida manuale quando in modalità driverless e viceversa, diminuendo così l'overhead. Una volta scelta la modalità e, se in Autonomous, anche la missione, questo task provvederà ad attivare tutti quelli necessari. La



modifica della modalità non viene più permessa, perciò non si potrà cambiare modalità di funzionamento o missione finché non sarà resettata la scheda. Questa scelta è stata fatta per semplificare il lavoro del pilota in guida manuale: in questo modo se occorre eseguire dei test che richiedono un ammontare di tempo non indifferente, non si avrà bisogno di selezionare una missione o una modalità finché non si ha un totale reset della scheda stessa. Il task prevede un periodo di 200 ms, non è fondamentale la sua continua esecuzione, soprattutto dopo aver selezionato la modalità. Una possibile futura implementazione potrebbe essere anche la totale disattivazione del task una volta selezionata la modalità di funzionamento. Nell'astrazione dell'Activity diagram non viene rappresentanto ma, ovviamente, i task attivati a seconda della modalità saranno attivati una sola volta pur essendo questo un task periodico.

ASB Check Task





Si tratta del task che si occupa di fare un check iniziale dell' Autonomous System Brake (ASB). Essendo l'impianto frenante un dispositivo di sicurezza, il regolamento ci impone un controllo sequenziale di tutti i sensori di pressione riguardanti il suddetto impianto. In caso non sia tutto ok, il task va in errore e spegne il motore, non permettendo di fatto che la vettura possa procedere con le operazioni. Inoltre, manda anche un messaggio di errore via CAN per essere visualizzato dal Display della PILOT23 e per essere salvato dal nostro logger. A procedura

finita, se non ci sono stati errori, il task provvede a settare una variabile a True per indicare all'ASStateHandler Task che il sistema frenante è pienamente in funzione (in sezione critica per gestire la concorrenza di accesso al dato), inoltre provvede ad inviare al pc il primo comando di start che prevede di accendere tutto il sistema di guida autonoma ed infine viene attivato il task periodico ErrorHandAS. Il task è one-shot quindi esegue la procedura descritta dall'Activity Diagram una sola volta. Se c'è stato errore comunque l'intero sistema ha bisogno di un reset e quindi il seguente task verrà rieseguito (in caso di modalità driverless).

Timelines e PTPN

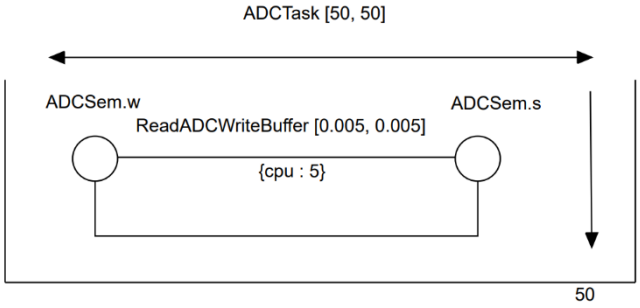
Dopo aver completato Activity Diagram e Object Diagram, è stato scritto il codice in C e sono stati misurati i tempi di esecuzione dei vari task. Questo è stato fatto usando un timer hardware per una precisione al decimo di microsecondo. I risultati ottenuti sono stati poi riportati a microsecondi.

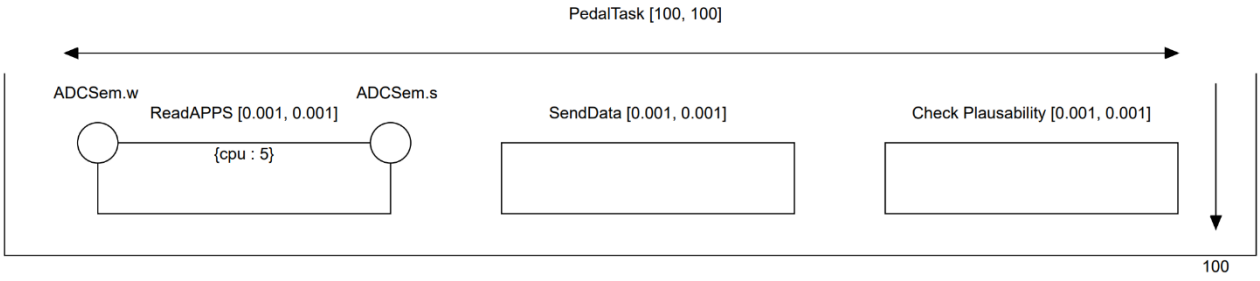
Ogni ET (Execution Time) è stato preso in modo indipendente per ogni chunk dei vari task e ognuno di quest'ultimi è stato eseguito in modo indipendente per evitare prelazioni; in questo modo si ottengono dei tempi di esecuzione più fedeli alla realtà. Inoltre, è stata fatta una media di 10000 elementi per avere un dato più accurato possibile. Sorprendentemente, i vari chunk hanno tempi di esecuzione di qualche microsecondo, data la frequenza di clock di 216MHz del nostro microcontrollore.

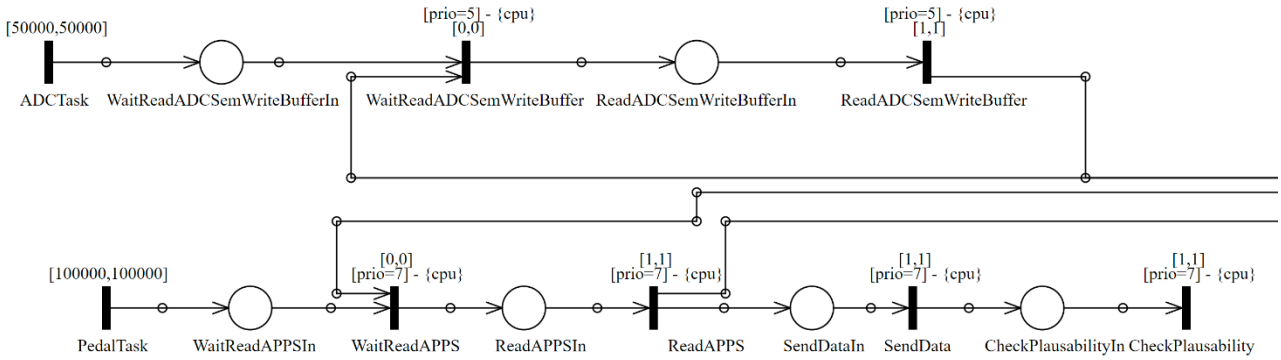
Da ciò si può già intuire la feasibility di questo taskset. I tempi delle primitive dei semafori come i tempi per eseguire le primitive per ottenere i tempi non sono stati presi in considerazione in quanto comunque risulterebbero, considerando i tempi dei vari chunk, trascurabili.

Ottenuti quindi i vari ET, sono state realizzate le timeline complete del taskset per ottenere così le PTPN. Entrambi i modelli sono allegati a questa documentazione.

Di seguito troviamo degli esempi, timelines e PTPN del Pedal Task e ADC Task







Conclusioni

Siamo arrivati alla conclusione del progetto, il codice quindi funziona in maniera ottimale, grazie anche alle analisi e alle ottimizzazioni effettuate in fase di progettazione.
Il capacity factor non è elevato, ma trattandosi di un sistema hard real-time non è uno svantaggio, anzi; è fondamentale, invece, che i tempi di esecuzione, per quanto comunque relativamente laschi rispetto alla velocità del processore, siano rispettati.

Volendo si potrebbe ridurre la frequenza di clock per diminuire il consumo energetico, anche se questo porta a una riorganizzazione completa di tutti i domini di clock, compresi quelli delle periferiche, ma anche un rallentamento generale dei task.

È stata anche sviluppata una nuova versione della CORE23, chiamata CORE24, in occasione del progetto di tesi del sottoscritto, che integra un microcontrollore STM32H7, con una frequenza di clock di 550MHz. In questo caso, andrebbe rifatta l'analisi oppure adeguata la frequenza di clock.