Facoltà di Ingegneria

CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA INFORMATICA

METODI E STRUMENTI DI COPROGETTAZIONE PER SISTEMI ELETTRONICI

Progetto di un computer di bordo per auto

Studenti:

Aulico Fabio (140907)

Giambruno Alessandro (140920)

Scherma Gaspare (141178)

Professore:

Reyneri Leonardo

Corso di laurea specialistica in Ingegneria Informatica 1
Metodi e strumenti di coprogettazione per sistemi elettronici 1
Introduzione 5
Struttura del Computer di bordo 5
Strumenti utilizzati 5
Casi d'uso 6
Attori 6
Diagramma alto livello 7
Tipi definiti 23
Diagramma delle Classi 24
Diagrammi Collaborazionali 27
Hardware e simulazioni 40
Descrizione fisica ..... **Error! Bookmark not defined.**
Rappresentazione logica e layout ..... **Error! Bookmark not defined.**
Messaggi di testo ..... **Error! Bookmark not defined.**
Modello logico ..... **Error! Bookmark not defined.**
Mappatura fisica ..... **Error! Bookmark not defined.**
Installazione del font ..... **Error! Bookmark not defined.**
Scrittura del messaggio ..... **Error! Bookmark not defined.**
Tachimetro ..... **Error! Bookmark not defined.**
Grafica (icone) ..... **Error! Bookmark not defined.**
Modello logico ..... **Error! Bookmark not defined.**
Mappatura fisica ..... **Error! Bookmark not defined.**
Considerazioni aggiuntive ..... **Error! Bookmark not defined.**
Partizionamento HARDWARE/SOFTWARE 55
Mappe di memoria 38
Codice Sorgente 55

AnabbagliantiHW.c 55
AnabbagliantiHW.h 57
CentralinaComandi.c 57
CentralinaComandi.h 63
CentralinaSensoriHW.h 65
CentralinaSensori.c 65
CentralinaSensori.h 67
ComandoCruscotto.c 69
ComputerDiBordo.c 69
ComputerDiBordo.h 70
Controllore.c 71
Controllore.h 88
Display.c 89
Display.h 93
DisplayHW.c 96
Display.h 106
FendiNebbiaAnterioriHW.c 112
FendiNebbiAnteriori.h 113
FendiNebbiaPOsterioriHW.c 114
FendiNebbiaPosterioriHW.c 115
global.h 115
LuciAbbagliantiHW.c 118
LuciAbbagliantiHW.h 119
LuciPosizioneHW.c 119
LuciPosizioneHW.h 121
PosizioneChiaveHW.c 121
PosizioneChiaveHW.h 123
Pulsantiera.c 123
Pulsantiera.h 124
PulsantieraHW.c 124

PulsantieraHW.h 125
SensoreLivCarburanteHW.c 126
SensoreLivCarburanteHW.h 127
SensoreLivOlioHW.c 127
SensoreLivOlioHW.h 129
SensoreOdometroHW.c 129
SensoreOdometroHW.h 130
SensoreStatoBatteriaHW.c 131
SensoreStatoBatteriaHW.h 132
SensoreStatoCintureHW.c 132
SensoreStatoCintureHW.h 134
SensoreStatoLuciDirezionaliHW.c 134
SensoreStatoLuciDirezionaliHW.h 135
SensoreTempAcquaHW.c 136
SensoreTempAcquaHW.h 137

INTRODUZIONE

Il seguente lavoro riguarda la progettazione di un computer di bordo per autoveicolo, che permetta la visualizzazione di informazioni e statistiche relative al funzionamento globale dell'autoveicolo stesso. Tali informazioni possono essere richieste dall'utente, o possono essere visualizzate in modo automatico dal sistema in caso di avarie o di livelli non nella norma. E' previsto un display 8 x 40 (240x64 pixel) ed una pulsantiera per il controllo delle funzionalità; sono previsti inoltre dei sensori per il controllo dei livelli di interesse. I servizi offerti sono pensati per un solo utilizzatore, che è l'utente che usufruisce del sistema.

STRUTTURA DEL COMPUTER DI BORDO

VENGONO ELENCATI DI SEGUITO I COMPONENTI CHE ANDRANNO A COMPORRE IL COMPUTER DI BORDO:

- Un *display 8 x 40*
- Una *pulsantiera* a 4 pulsanti
- Un *microcontrollore*
- Un *sensore odometrico* per il calcolo della velocità
- Un sensore per la misurazione della temperatura dell'acqua
- Un sensore per la misurazione del livello dell'olio
- Un sensore per la misurazione del livello del carburante
- Un sensori per il controllo dell'accensione delle luci
- Un sensore per il controllo dello stato di aggancio delle cinture di sicurezza
- Un sensore per il controllo della batteria

In seguito questi componenti verranno spiegati in dettaglio.

STRUMENTI UTILIZZATI

Gli strumenti che abbiamo utilizzato nella realizzazione del progetto sono:

· Eclipse C/C++ per la stesura degli oggetti relativi al distributore ed implementati in linguaggio C
· ArgoUML per la progettazione in linguaggio UML del progetto
· Microsoft Word per la stesura dei documenti
· Pdf Creator per la conversione in pdf
· CVS per la condivisione del codice tra i componenti del team, a causa della distanza fisica tra questi.

CASI D'USO

ATTORI

SENSORE_LUCIDIREZIONALI

Tale attore viene interrogato per ottenere lo stato delle luci direzionali

SENSORE_LUCI

Tale attore viene interrogato per ottenere lo stato delle Luci

SENSORE_CINTURE

Tale attore viene interrogato per ottenere lo stato delle cinture

SENSORE_BATTERIA

Tale attore viene interrogato per ottenere lo stato della batteria durante l'accensione dell'automobile

SENSORE_LIVELLOCARBURANTE

Tale attore viene interrogato per ottenere lo stato del livello carburante

SENSORE_LIVELLOOLIO

Tale attore viene interrogato per ottenere lo stato del livello dell'olio

SENSORE_TEMPERATURAMOTORE

Tale attore viene interrogato per ottenere lo stato della temperatura dell'acqua interna al motore

SENSORE_ODOMETRO

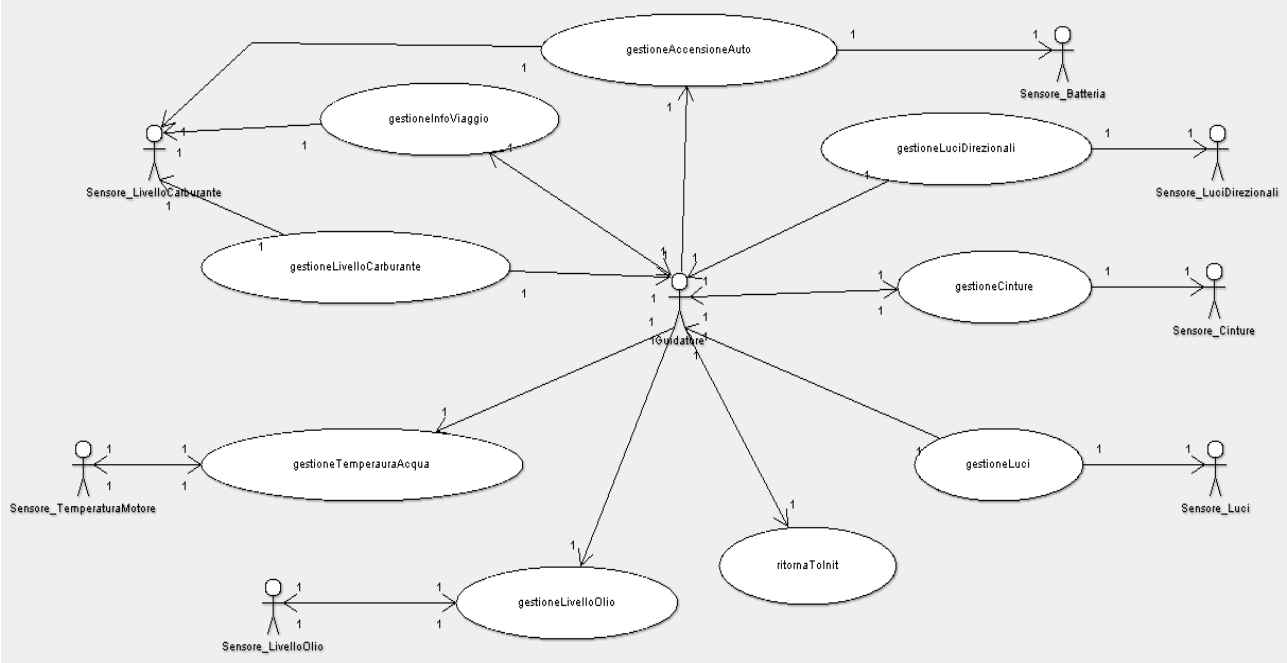
Tale attore invia un segnale al sistema indicando che un giro di ruota è stato effettuato

GUIDATORE

Attore principale che interagisce con il sistema ed utilizza le principali funzionalità

DIAGRAMMA ALTO LIVELLO

Tale diagramma dei casi d'uso descrive le funzionalità complessive offerte dal sistema, tali funzionalità vengono successivamente descritte nei diagrammi specifici. Effettuiamo una precisazione: quando il sistema richiede ad un sensore, tramite un messaggio specifico sulla porta RS232 dal controllore alla centralina, viene modellata la risposta del sensore ma non l'effettivo funzionamento del dispositivo (es: modelliamo il fatto che il sensore comunichi lo stato della luce, accesa o spenta, ma non il fatto che il sistema accenda la luce).



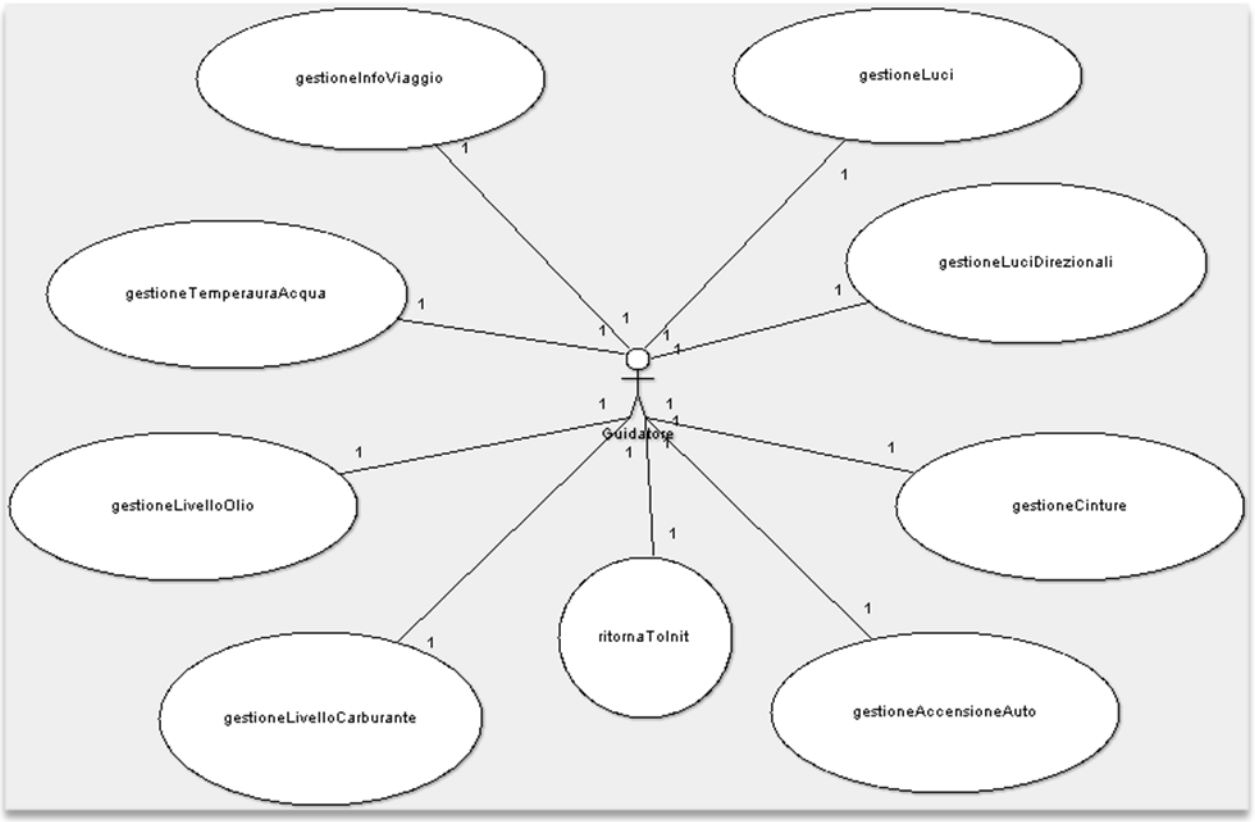
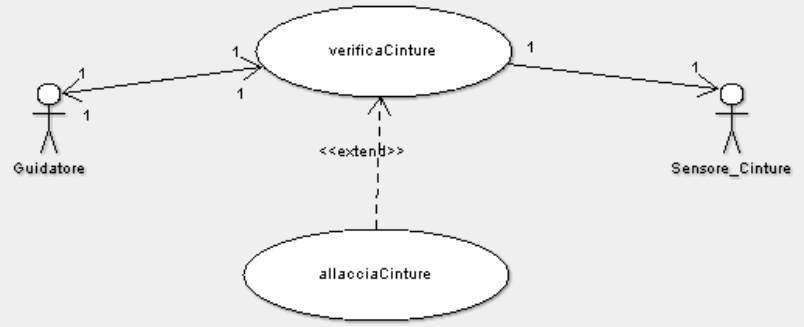


DIAGRAMMA GESTIONECINTURE



VERIFICACINTURE

Il caso d'uso gestisce il controllo del corretto utilizzo delle cinture di sicurezza.

- Il *sistema* si trova in stato ACCESO e chiede all'attore *Sensore_cintura* se la cintura è allacciata.
  - **Se** la cintura non è allacciata, si estende (<<extend>>) il caso d'uso allacciaCinture.
  - **Altrimenti** non viene visualizzato alcun messaggio.

ALLACCIACINTURE

Questo caso d'uso estende il caso d'uso verificaCinture e viene richiamato se il sensore indica che la cintura non è inserita.

- Il *sistema* visualizza sul Display la spia (`spia_cintura`) ed il messaggio "`Allacciare le cinture`", che richiede all'attore Guidatore di allacciare la cintura.
- L'attore *Guidatore* allaccia la cintura di sicurezza (lato guida)
- Il *sistema* disattiva la spia (`spia_cintura`) ed il messaggio dal display.

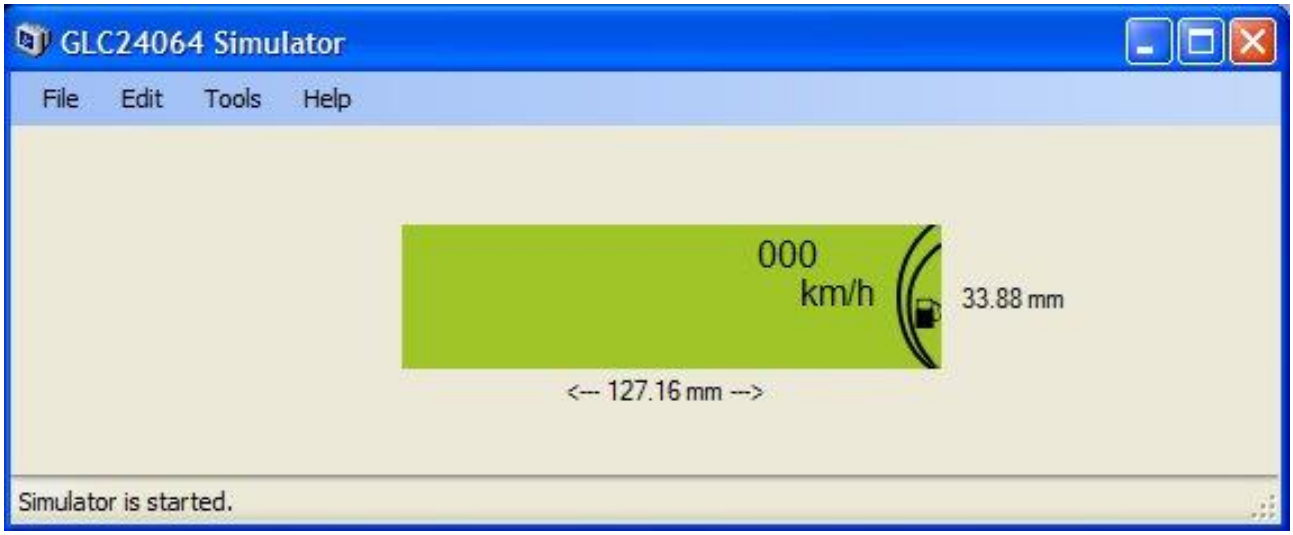
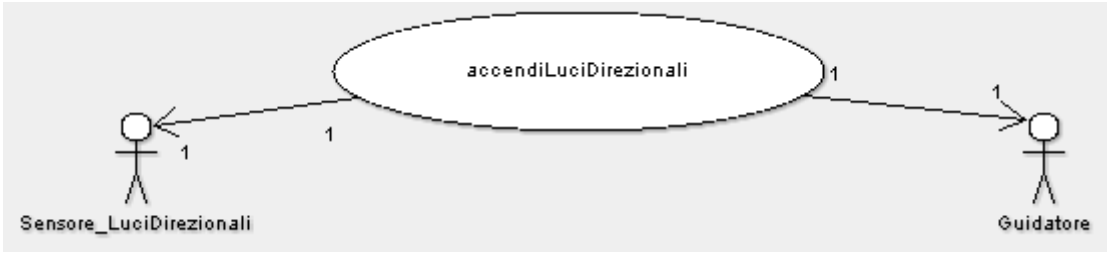


DIAGRAMMA GESTIONELUCIDIREZIONALI



ACCENDILUCIDIREZIONALI

Il caso d'uso gestisce l'accensione delle luci direzionali dell'auto.

- Il *sistema* richiede in polling lo stato delle luci direzionali all'attore *Sensore_LuciDirezionali*.
- il *sistema* ricevuto lo stato del sensore attiva sul display la spia (spia_frecce) nel caso in cui il suo stato sia DX SX e nulla in caso di WAIT.

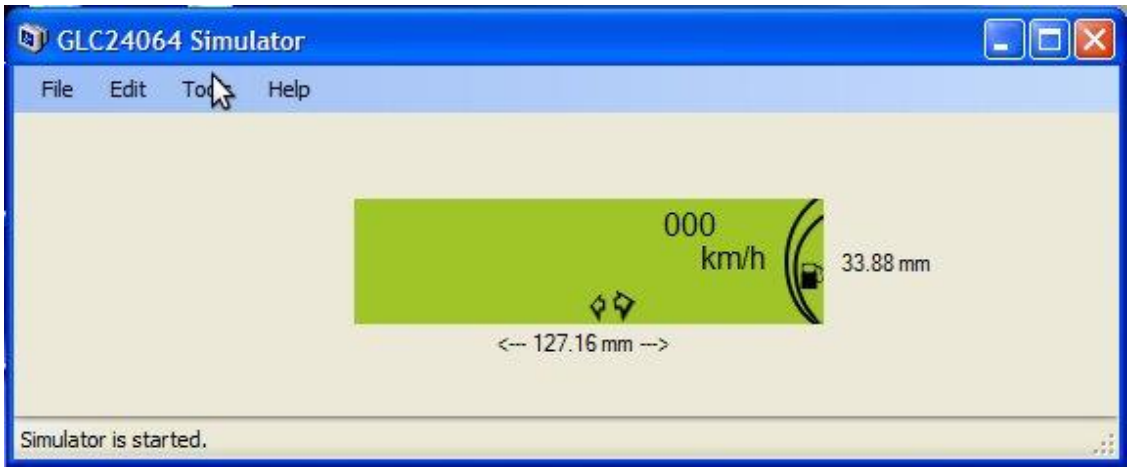
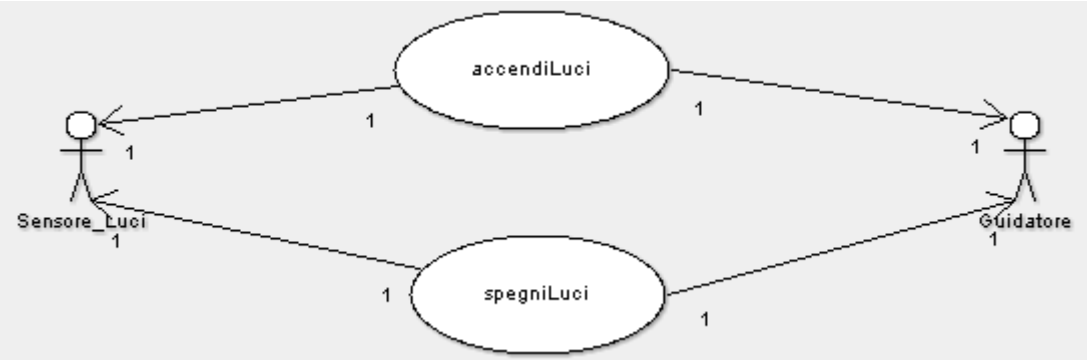


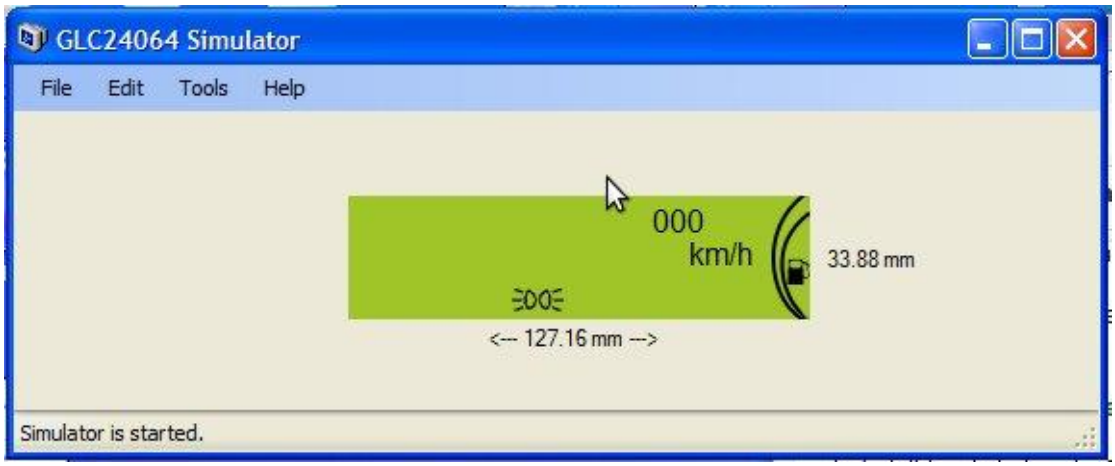
DIAGRAMMA GESTIONE LUCI



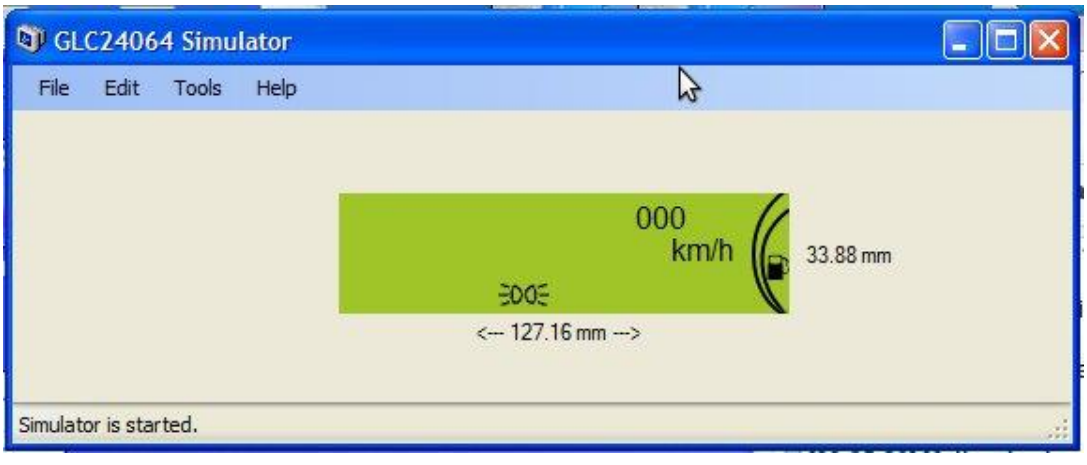
ACCENDILUCI

Il caso d'uso gestisce l'accensione delle luci dell'auto.

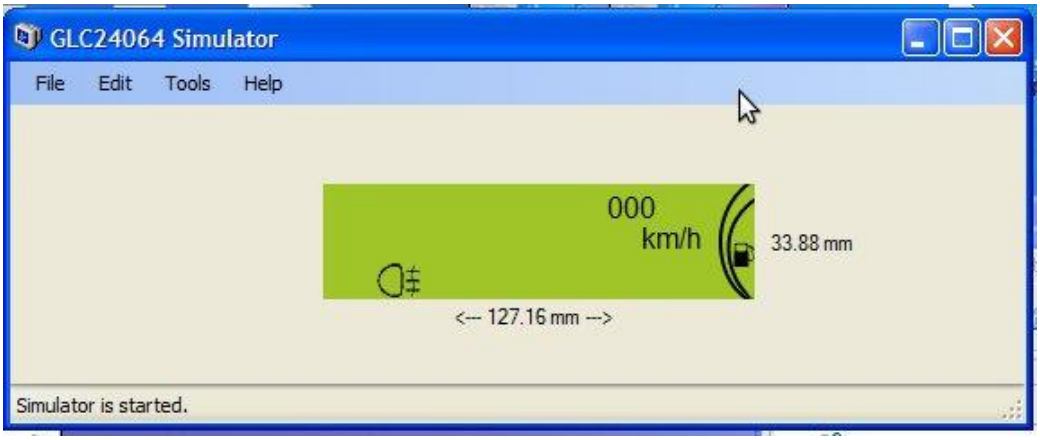
- Il *sistema* richiede in polling lo stato delle luci all'attore *Sensore_Luci* .
- il *sistema* ricevuto lo stato del sensore attiva sul display la spia luce (una tra: posizione, anabbaglianti, fendinebbia anteriori, fendinebbia posteriori, abbaglianti)



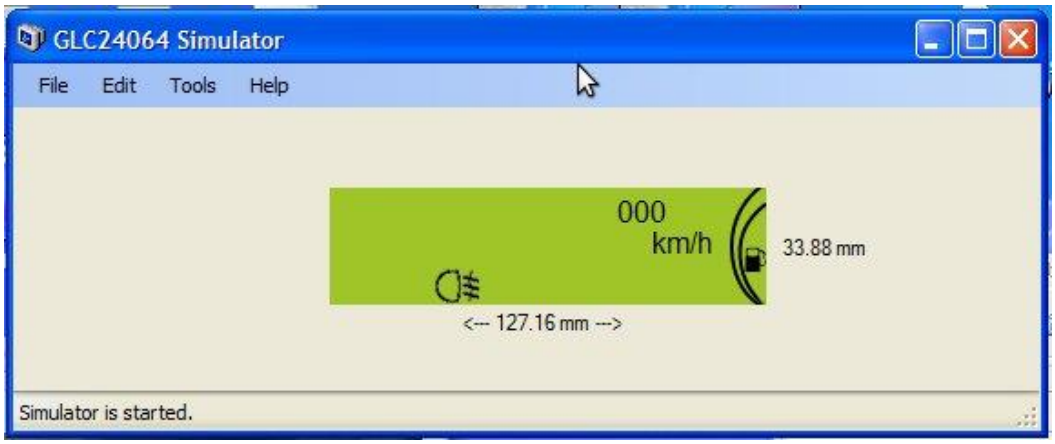
Screenshot per il caso d'uso riguardante le "luci posizione"



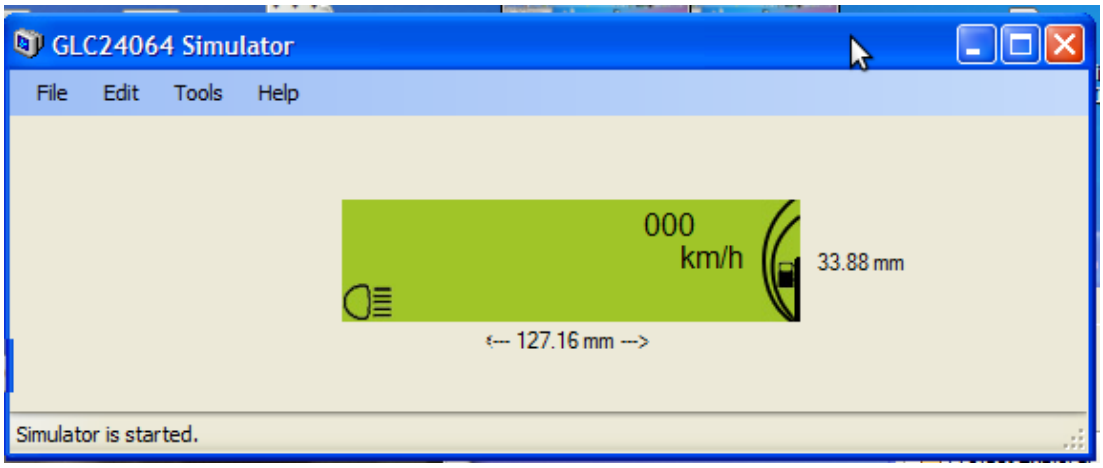
Screenshot per il caso d'uso riguardante le "luci anabbaglianti"



Screenshot per il caso d'uso riguardante le "fendinebbia"



Screenshot per il caso d'uso riguardante le "fendinebbia posteriore"



Screenshot per il caso d'uso riguardante le "luci abbaglianti"

SPEGNILUCI

Il caso d'uso gestisce lo spegnimento delle luci dell'auto.

- Il *sistema* richiede in polling lo stato delle luci all'attore *Sensore_Luci* .
- il *sistema* ricevuto lo stato del sensore disattiva sul display la spia luce precedentemente accesa (una tra: posizione, anabbaglianti, fendinebbia anteriori, fendinebbia posteriori, abbaglianti)

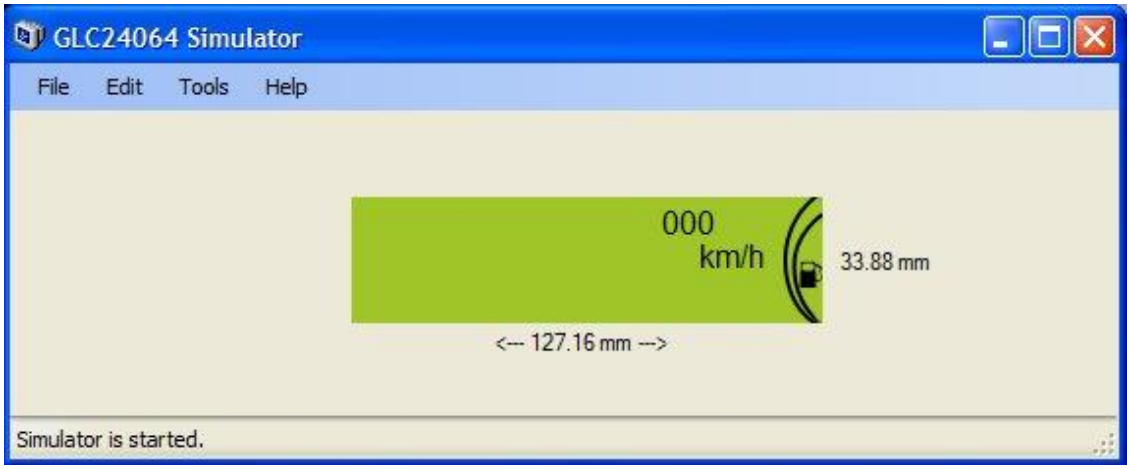
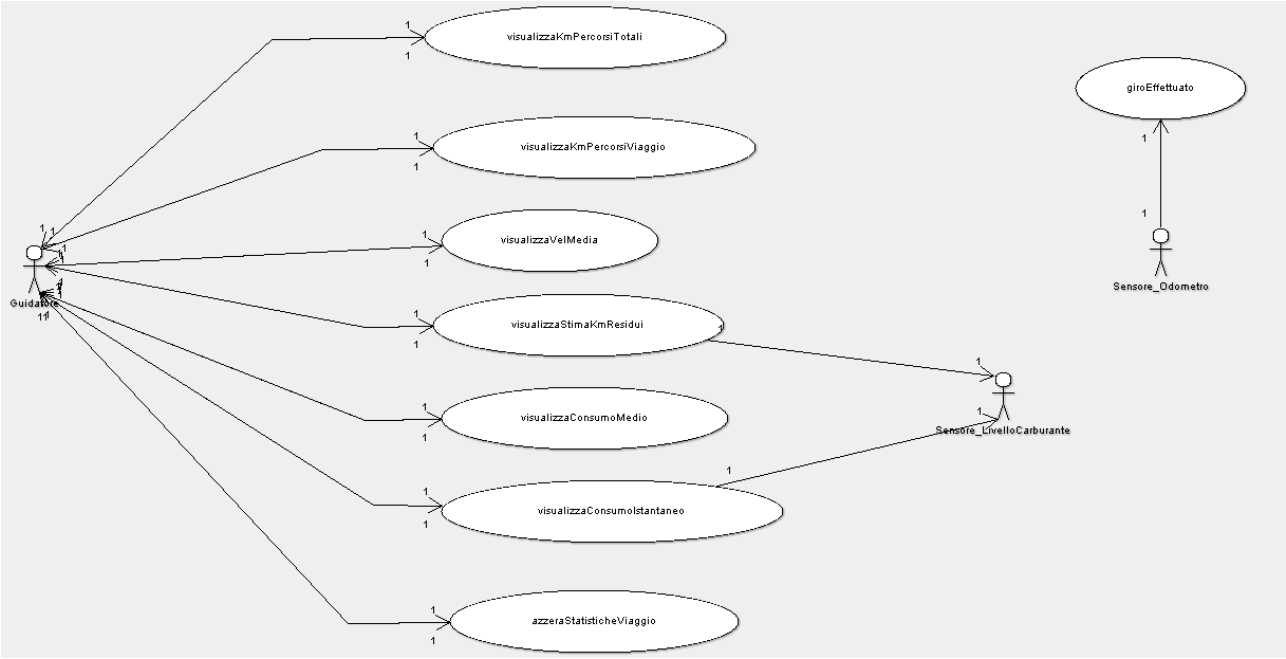


DIAGRAMMA GESTIONEINFOVIAGGIO



GIROEFFETTUATO

Tale caso d'uso viene richiamato dall'odometro per calcolare le statistiche di viaggio

- L'attore *Sensore_Odometro* invia l'informazione relativa al giro completato
- Il *sistema* riceve la comunicazione e memorizza all'interno di una variabile (`time_stamp`) l'istante di tempo in cui è stata ricevuta.
- Il *sistema* aggiorna lo stato di viaggio dall'automobile incrementando gli opportuni contatori della velocità calcolata sulle basi dei giri della ruota e la circonferenza di questa. (`count_viaggio, count_tot`).

VISUALIZZAKMPERCORSITOTALI

Il caso d'uso gestisce la visualizzazione della distanza totale percorsa dall'auto.

- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato INIT.
- L'attore *Guidatore* preme il pulsante P1.
- Il *sistema* visualizza sul display la distanza totale percorsa dall'automobile (`km_tot`) e lo stato del display passa a VISUALIZZA_TOT_KM.

VISUALIZZAKMPERCORSIVIAGGIO

Il caso d'uso gestisce la visualizzazione della distanza parziale percorsa dall'auto.

- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato VISUALIZZA_TOT_KM.
- L'attore *Guidatore* preme il pulsante P1.
- Il *sistema* visualizza sul display la distanza parziale percorsa dall'automobile (`km_viaggio`) e passa nello stato VISUALIZZA_PARZ_KM.

VISUALIZZAVELOCITÀMEDIA

Il caso d'uso gestisce la visualizzazione della velocità media.

- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato VISUALIZZA_PARZ_KM.
- L'attore *Guidatore* preme il pulsante P1.
- Il *sistema* visualizza sul display la velocità media dell'automobile (`vel_media`) e passa nello stato VISUALIZZA_VEL_MEDIA.

AZZERASTATISTICHEVIAGGIO

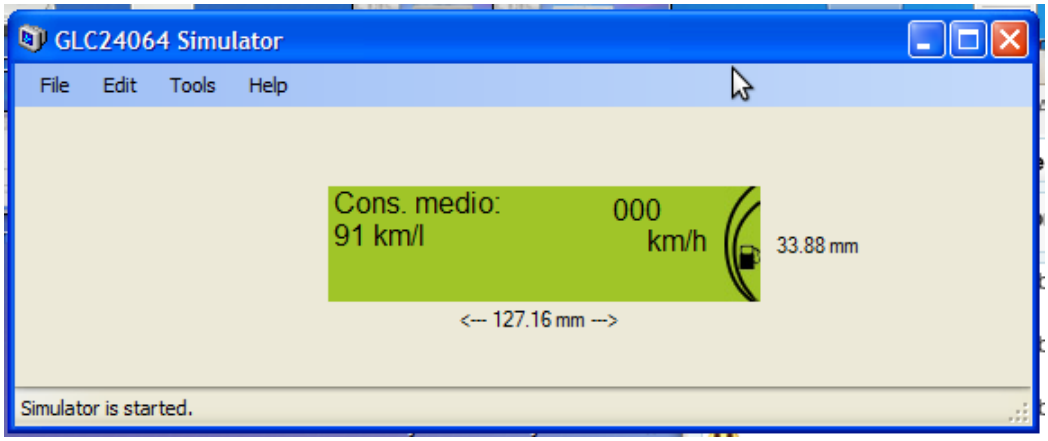
Il caso d'uso gestisce l'azzeramento della distanza parziale percorsa dall'auto.

- Il *sistema* si trova in stato STANDBY o ACCESO.
- L'attore *Guidatore* preme il pulsante P4.
- Il *sistema* cancella tutte le statistiche e mostra sul display un messaggio "`Statistiche azzerate`" e l'icona di avvertimento .

VISUALIZZACONSUMOMEDIO

Il caso d'uso gestisce la visualizzazione del consumo medio di carburante.

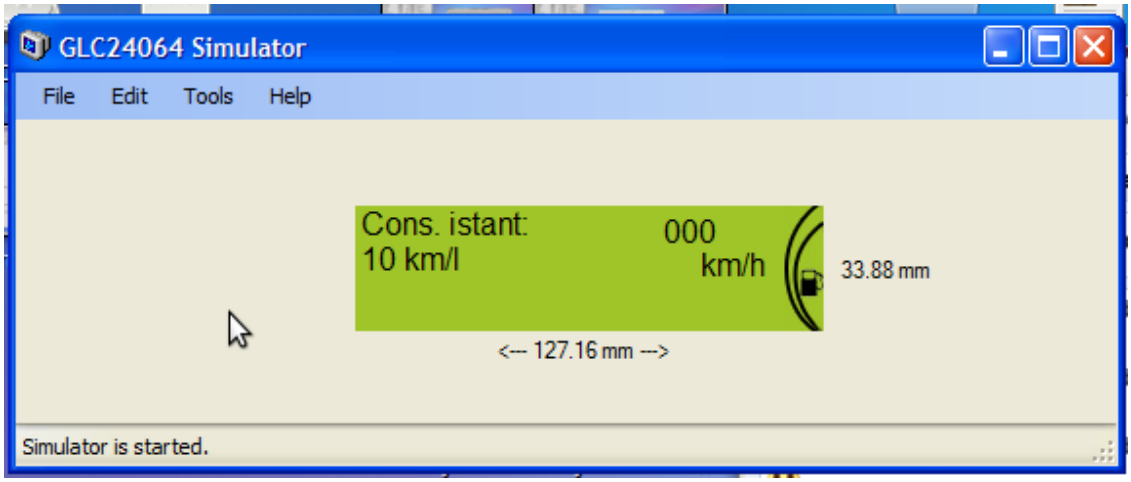
- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato INIT.
- L'attore Guidatore preme il pulsante P2.
- Il *sistema* visualizza sul display il consumo medio di carburante (`consumo_medio`) e passa nello stato VISUALIZZA_CONSUMO_MEDIO



VISUALIZZACONSUMOISTANTANEO

Il caso d'uso gestisce la visualizzazione del consumo istantaneo di carburante.

- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato VISUALIZZA_CONSUMO_MEDIO.
- L'attore *Guidatore* preme il pulsante P2.
- Il *sistema* calcola il valore prelevando la misura interrogata in polling precedentemente
- Il *sistema* visualizza sul display il consumo istantaneo di carburante (`consumo_istantaneo`) e passa nello stato VISUALIZZA_CONSUMO_ISTANTANEO



VISUALIZZASTIMAKMRESIDUI

Il caso d'uso gestisce la visualizzazione dei chilometri residui che il carburante permette di effettuare.

- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato VISUALIZZA_CONSUMO_ISTANTANEO.
- L'attore Guidatore preme il pulsante P2.
- Il *sistema* visualizza sul display la stima dei chilometri residui (`km_left`) che è possibile effettuare con il carburante presente nel serbatoio e passa allo stato VISUALIZZA_STIMA_KM_RESIDUI

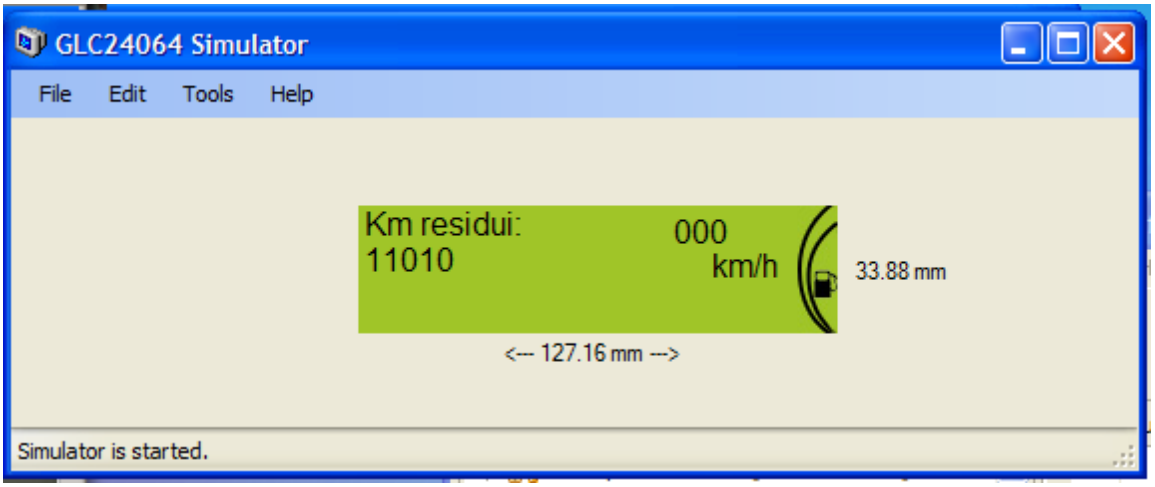
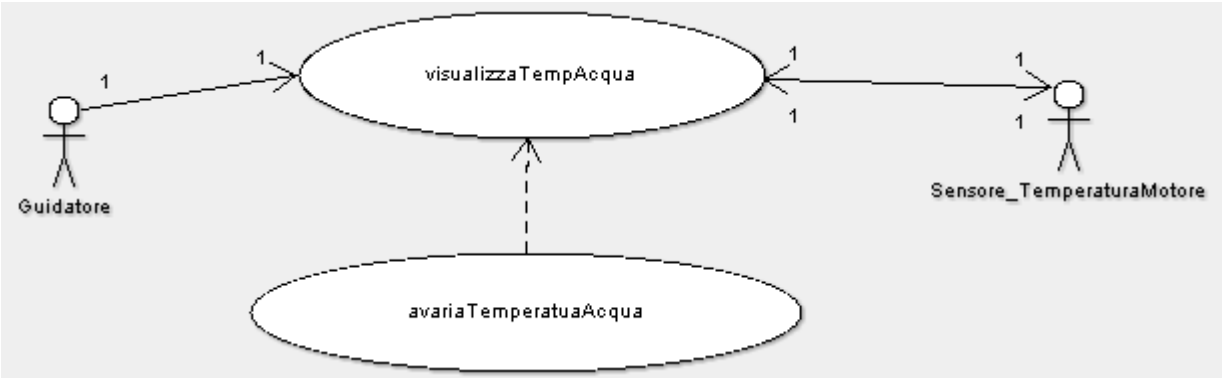


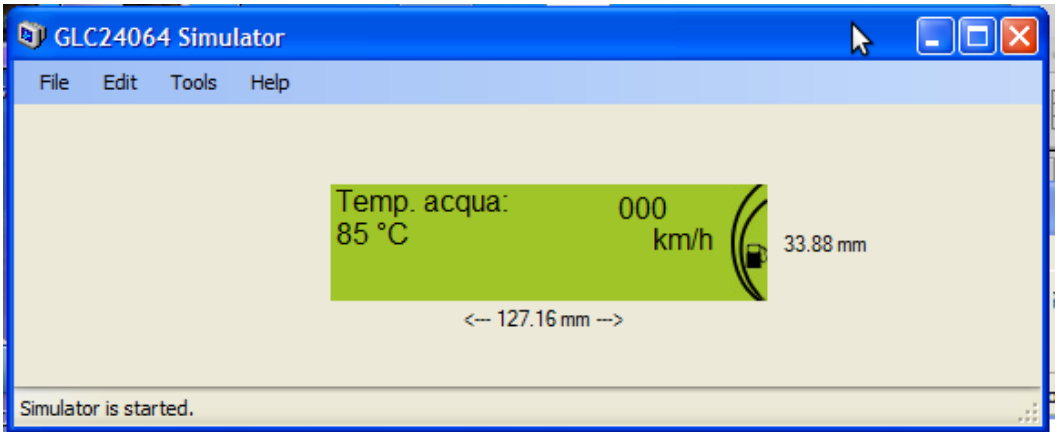
DIAGRAMMA GESTIONETEMPERATURAACQUA



VISUALIZZATEMPACQUA

Il caso d'uso gestisce la visualizzazione della temperatura dell'acqua

- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato INIT, e richiede in polling lo stato sulla temperatura dell'acqua all'attore *Sensore_TemperaturaAcqua.*
- L'attore Guidatore preme il pulsante P3.
- Il *sistema* visualizza sul display la temperatura dell'acqua (`temp_acqua`) e passa nello stato VISUALIZZA_TEMP_ACQUA. Se le temperatura risulta maggiore di 90 gradi centigradi, <<Extend>> il caso d'uso "AvariaTemperaturaAcqua"



AVARIATEMPERATURAACQUA

Il caso d'uso gestisce la situazione anomala derivante da un livello di temperatura dell'acqua del sistema di raffreddamento non nella norma.

- Il *sistema* visualizza sul display l'icona relativa all'avaria temperatura acqua ed il messaggio "`Temperatura elevata`"

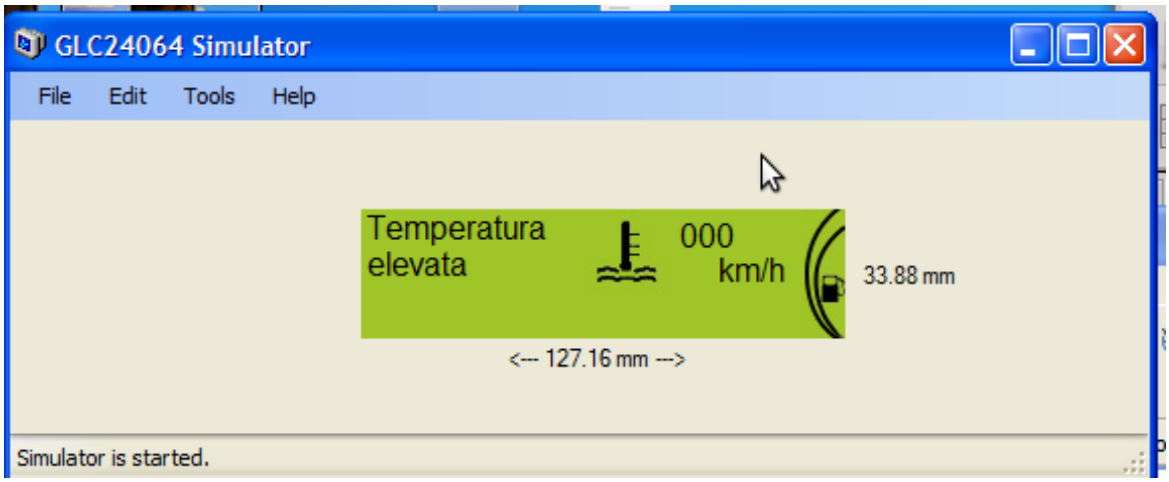
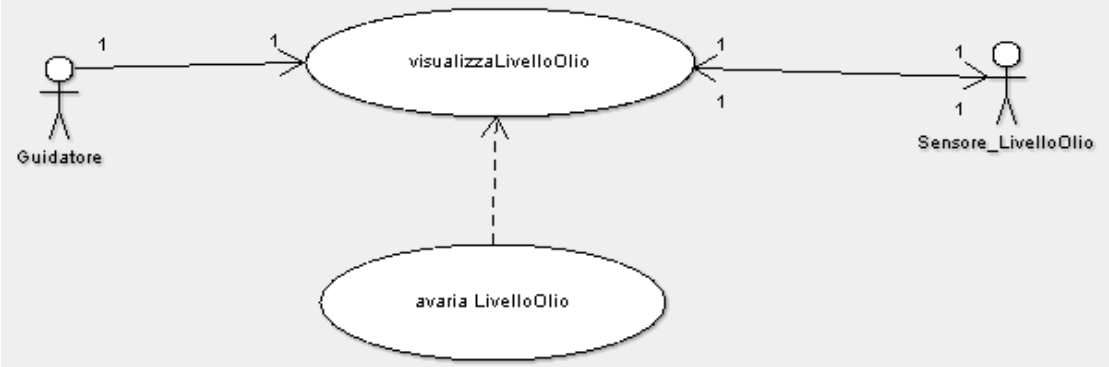


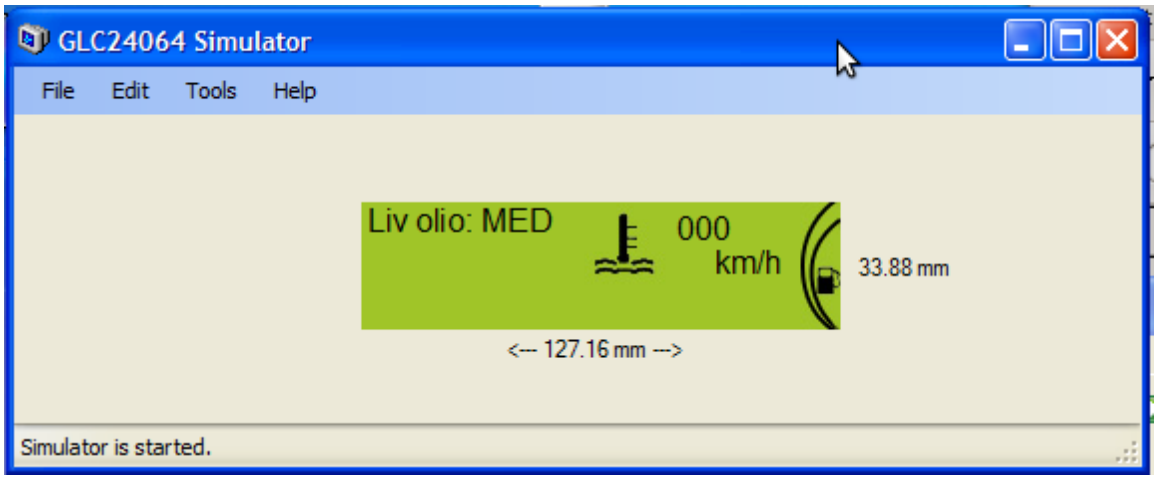
DIAGRAMMA GESTIONELIVELLOOLIO



VISUALIZZALIVELLOOLIO

Il caso d'uso gestisce la visualizzazione della temperatura dell'olio

- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato VISUALIZZA_TEMP_ACQUA. e richiede in polling lo stato sul livello dell'olio all'attore *Sensore_LivelloOlio.*
- L'attore *Guidatore* preme il pulsante P3.
- Il *sistema* riceve la comunicazione e visualizza sul display lo stato del livello dell'olio (`livello_olio`) che può assumere lo stato `MIN, MED, MAX;` passa nello stato VISUALIZZA_LIVELLO_OLIO. Se il livello è al di sotto del livello minimo, <<Extend>> il caso d'uso "AvariaLivelloOlio"



AVARIALIVELLOOLIO

Il caso d'uso gestisce la situazione anomala derivante da un livello dell'olio basso.

- *Il sistema* visualizza l'icona relativa all'avaria del livello olio ed il messaggio "`Livello olio insufficiente`"

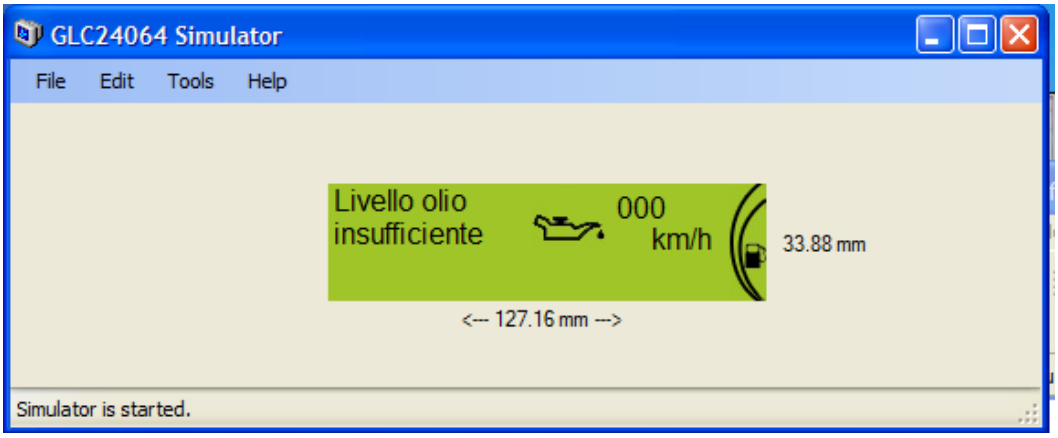
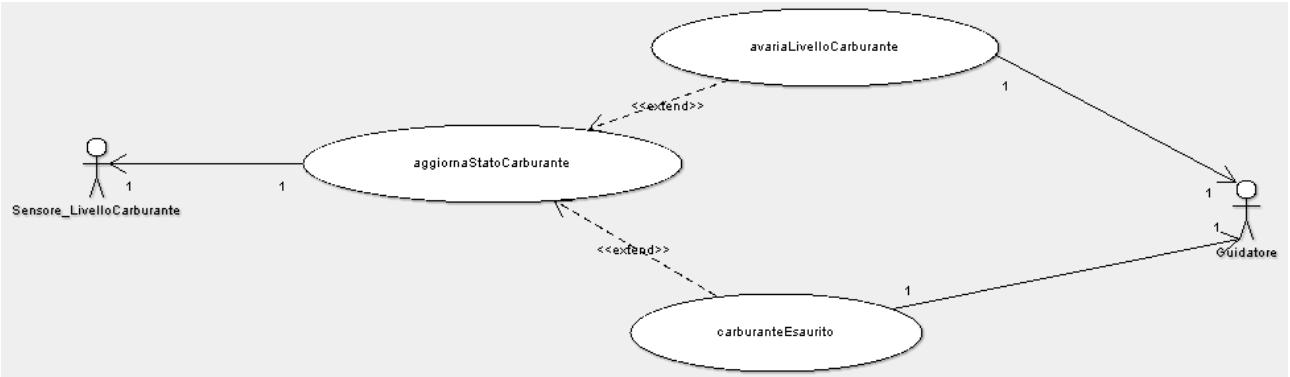


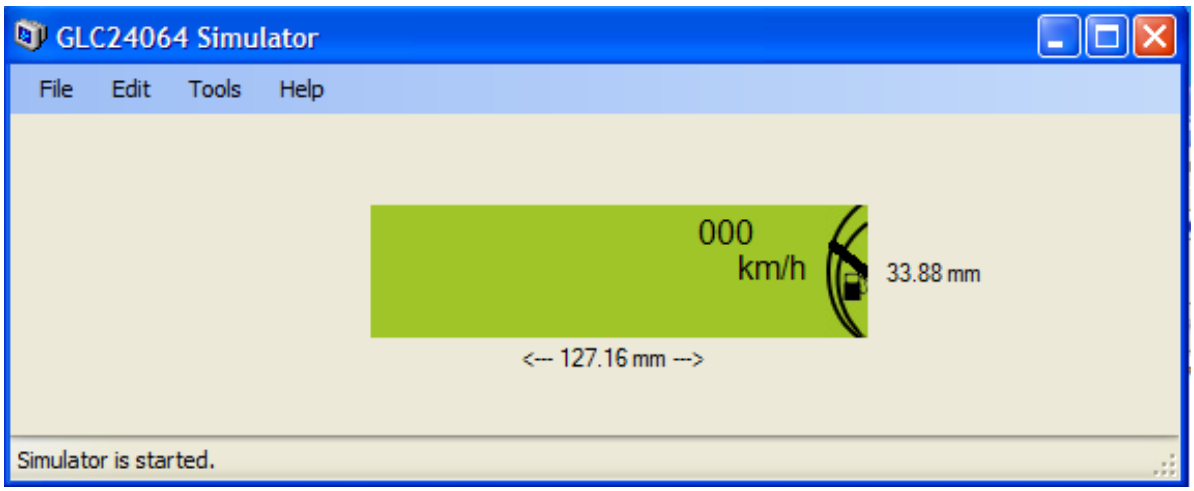
DIAGRAMMA GESTIONELIVELLOCARBURANTE



AGGIORNASTATOCARBURANTE

Il caso d'uso gestisce il controllo del livello carburante.

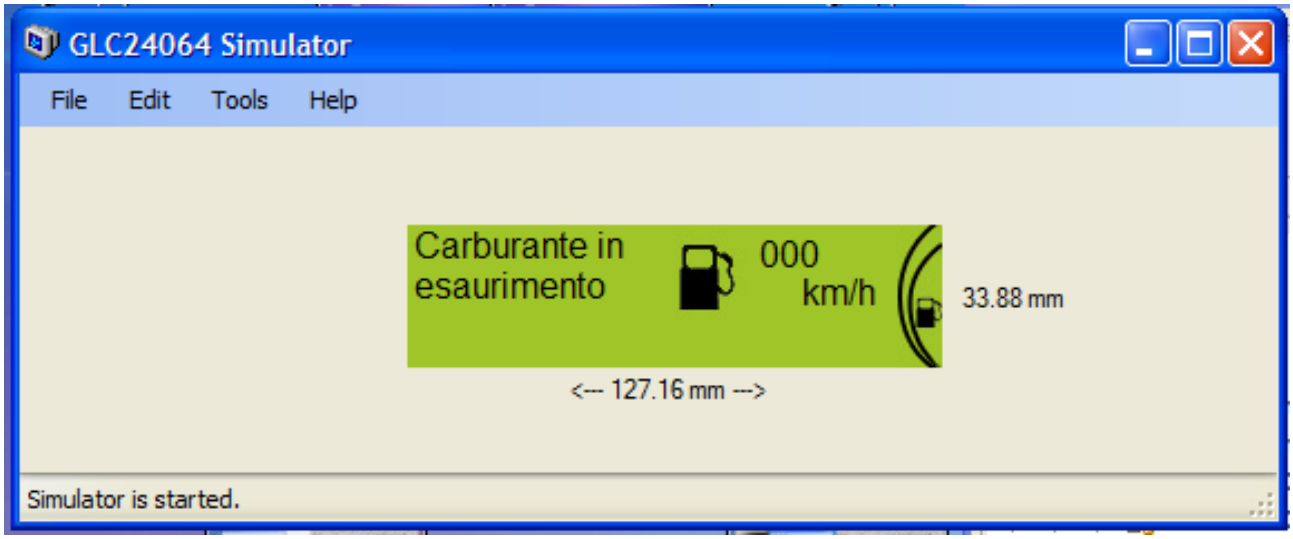
- *Il sistema* richiede in polling il livello del carburante all'attore *Sensore_LivelloCarburante* e lo visualizza in modo grafico tramite l'opportuno indicatore. Se il livello è al di sotto di una soglia minima (10% del totale), <<extend>> il caso d'uso "AvariaLivelloCarburante". Se il livello è al di sotto del 1%, <<Extend>> il caso d'uso "CarburanteEsaurito"



AVARIALIVELLOCARBURANTE

Il caso gestisce la situazione anomala derivante da un livello di carburante basso.

- Il *sistema* visualizza sul display l'icona relativa all'avaria del carbirante ed il messaggio `"Carburante in esaurimento"`.



CARBURANTEESAURITO

Il caso d'uso carburanteEsaurito si verifica quando il livello del carburante è insufficiente per l'accensione del motore o durante il moto dell'automobile. (L'icona viene spenta nel momento in cui il sistema, sempre in polling, si accorge che il livello del carburante è maggiore dell'1%)

- Il *sistema* riceve la comunicazione visualizza sul display la l'icona relativa al carburante esaurito ed il messaggio "`Carburante Esaurito`".

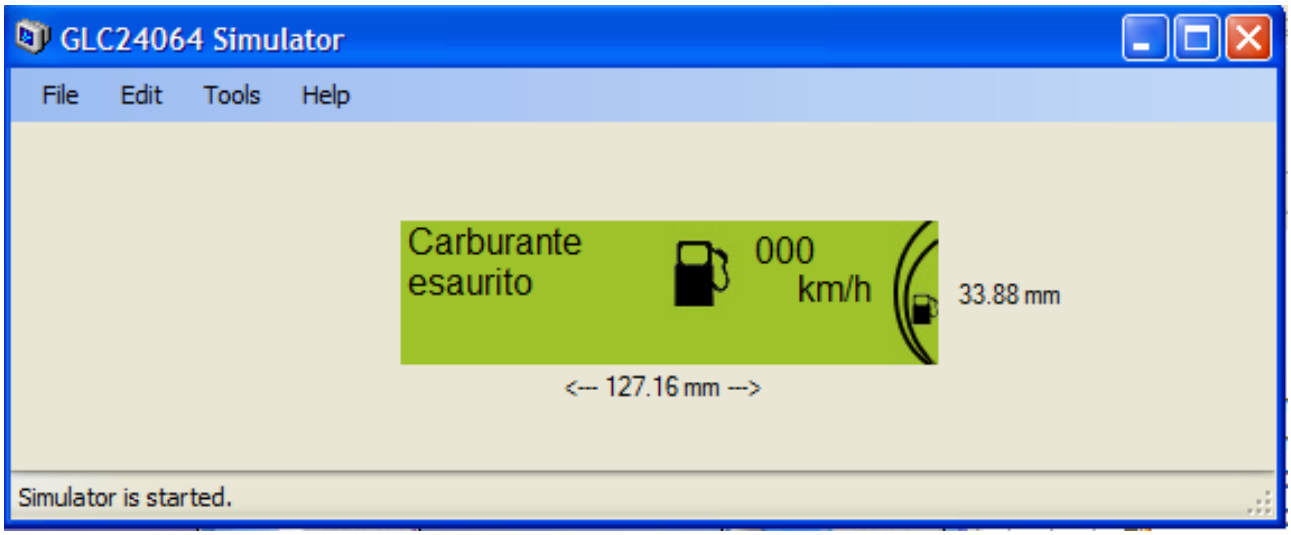


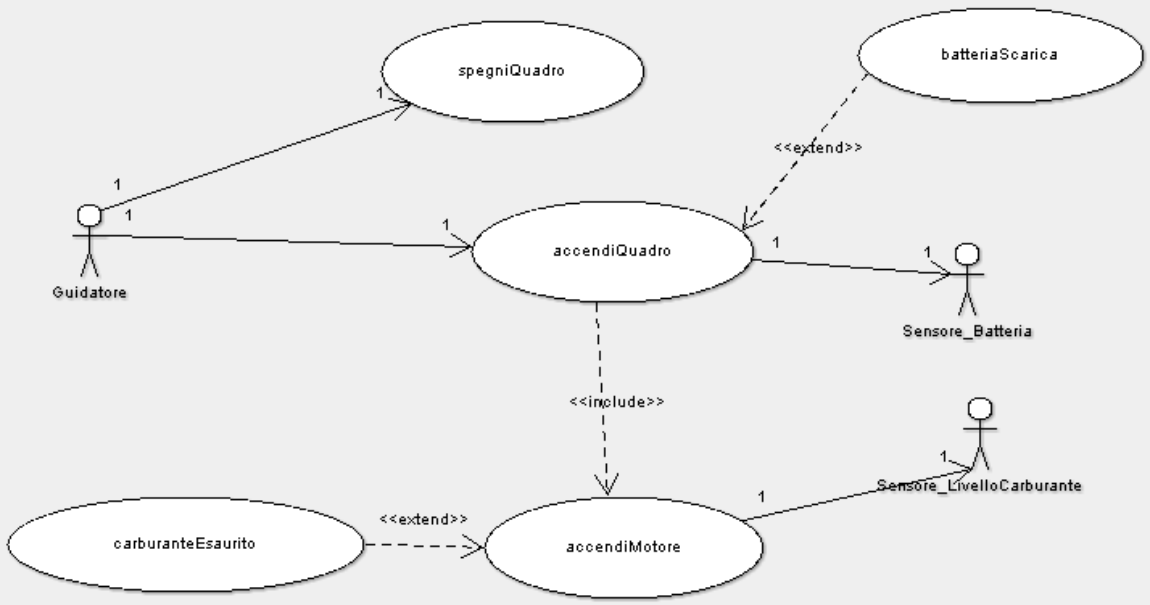
DIAGRAMMA RITORNATOINIT

RITORNATOINIT

Il caso d'uso gestisce la visualizzazione dello stato di INIT nel display

- Il *sistema* si trova in stato STANDBY o ACCESO, e lo stato del display è in stato VISUALIZZA_VEL_MEDIA, VISUALIZZA_STIMA_KM_RESIDUI o VISUALIZZA_LIVELLO_OLIO.
- L'attore *Guidatore* preme rispettivamente il pulsante P1,P2 o P3 per gli stati del display citati al punto precedente.
- Il *sistema* visualizza sul display la schermata di init e passa nello stato di INIT.

DIAGRAMMA GESTIONEACCENSIONEAUTO



ACCENDIQUADRO

Il caso d'uso accendiQuadro gestisce l'accensione e l'inizializzazione del sistema.

1. L'attore *Guidatore* ruota la chiave in senso orario, ponendola in posizione "I" (1 step).
2. Il *sistema* che si trova stato SPENTO, se il livello della batteria è sufficiente, se il livello dell'olio è sufficiente, se il livello del carburante è > del 10% e se la temperatura dell'acqua è < 90 gradi centigradi, passa alla modalità STANDBY e mostra sul display la spia della batteria (`spia_batteria`), il tachimetro con velocità "`0`", l'indicatore del livello del carburante e un messaggio di corretto funzionamento del sistema che indica che tutti i valori richiesti ai sensori sono nella norma;
    - Se il sistema funziona correttamente, <<Extend> il caso d'uso accendiMotore.

ACCENDIMOTORE

Il caso d'uso accendiMotore permette al Guidatore di avviare il motore dell'auto. Il caso d'uso viene richiamato se il caso d'uso accendiQuadro viene correttamente eseguito.

- L'attore *Guidatore* ruota la chiave in senso orario, ponendola in posizione "A" (1 step).
- Il *sistema* che si trova in modalità STANDBY, se il livello del carburante è sufficiente (superiore al 10%), passa alla modalità ACCESO, e disattiva sul display la spia della batteria (`spia_batteria`) e il messaggio "`Check OK`", dopo l'avviamento del motore. Altrimenti se <<Extend>> il caso d'uso "AvariaLivelloCarburante"",

SPEGNIQUADRO

Il caso d'uso spegniQuadro permette al Guidatore di spegnere l'auto e di conseguenza di disattivare il sistema.

- L'attore *Guidatore* ruota la chiave in senso antiorario, ponendola in posizione "O" (2 step).

- Il *sistema* che si trova in modalità ACCESO, passa in modalità SPENTO e disattiva il display.

BATTERIASCARICA

Il caso d'uso batteriaScarica estende il caso d'uso accendiQuadro e si verifica quando il livello della batteria è insufficiente per l'accensione del quadro.

- Se il livello della batteria è sufficiente, il quadro si accende e mostra la spia (`spia_avaria_batteria`) un messaggio di errore "`Avaria batteria`", altrimenti il sistema non viene avviato ed il display resta spento.

TIPI DEFINITI

Per la realizzazione del programma, oltre ai tipi standard messi a disposizione dal linguaggio

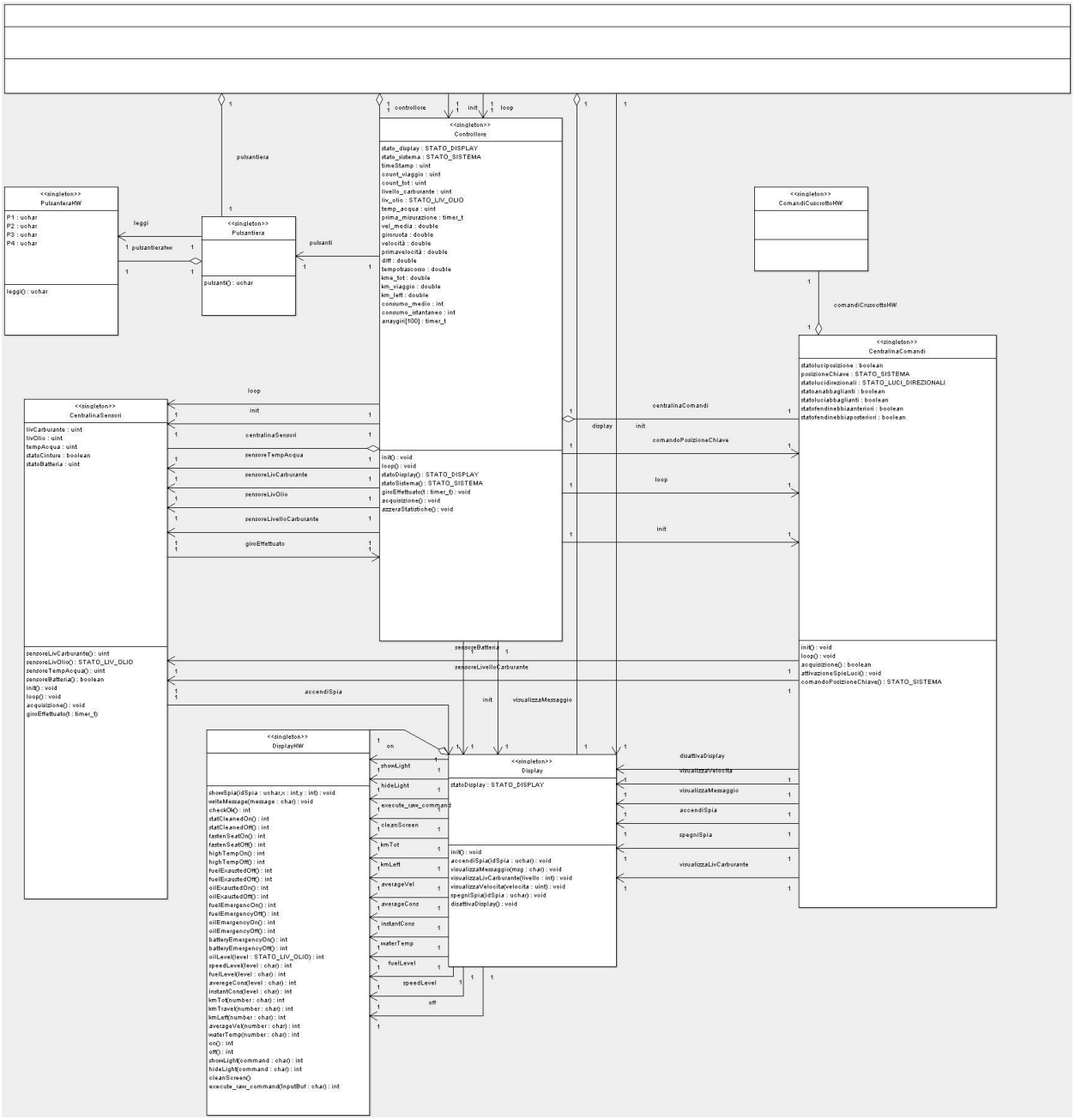
utilizzato, abbiamo definito:

- STATO_SISTEMA è una enum che indica lo stato del distributore e può assumere i seguenti valori:
  - SPENTO: Il sistema è spento; il display non è acceso e premendo i pulsanti non accade nulla.
  - STANDBY: Il sistema ed il display sono funzionanti. Vengono inizialmente visualizzate la spia della batteria, il livello del carburante, un messaggio di tipo "Check ok" ed il tachimetro con velocità nulla.
  - ACCESO: Il sistema ed il display sono funzionanti. Vengono inizialmente visualizzate il livello del carburante ed il tachimetro con velocità nulla.

DIAGRAMMA DELLE CLASSI

In figura è mostrata la visione completa delle classi che costituiscono il sistema. La figura successiva rappresenta il primo livello del diagramma delle classi;

Figura Classi completo



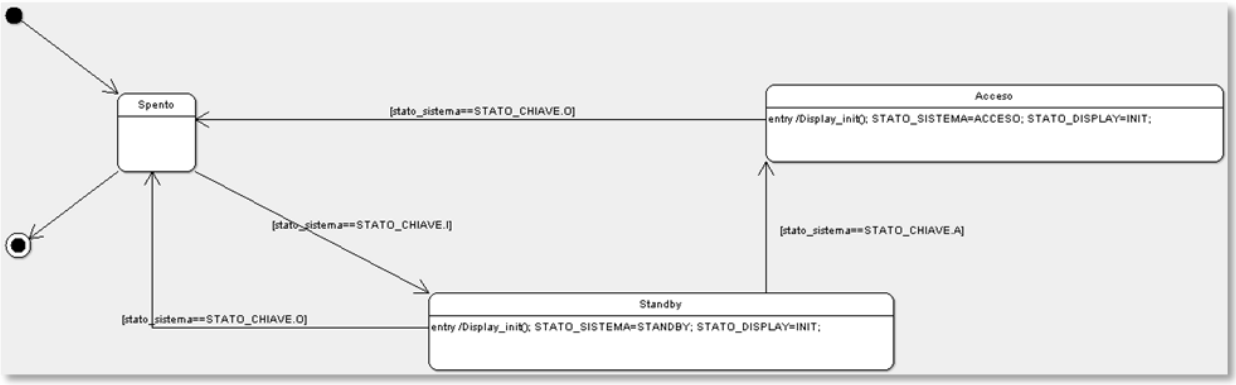
DIAGRAMMI A STATI

Tale diagrammi rappresentano come il sistema fa fronte alle richieste e risponde agli eventi che provengono dalla pulsantiera determinando così la visualizzazione a sul display delle informazioni definite durante la pianificazione dei casi d'uso.

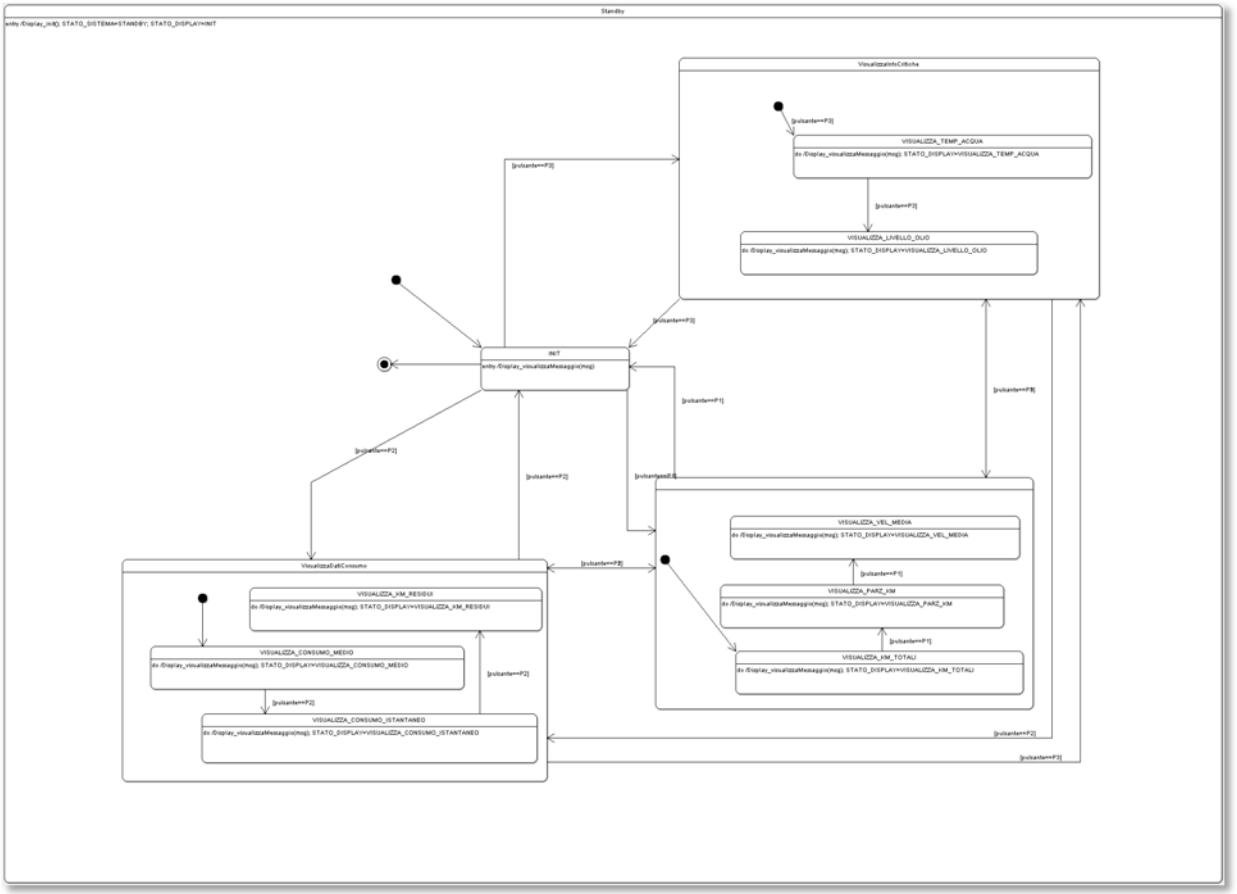
Si sono sviluppati principalmente due automi a stati, il primo si riferisce al sistema e in base alla condizione di questo definito dalla variabile globale STATO_SISTEMA reagisce ed interagisce come definito nei casi d'uso.

Viene fornita anche una spiegazione testuale dei passaggi di stato descritti nei diagrammi.

- STATO_DISPLAY è una enum che indica lo stato del display e può assumere i seguenti valori:
    - INIT:Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) e nessun tasto è stato premuto oppure se viene premuto il tasto P1 dallo stato VISUALIZZA_VEL_MEDIA, il tasto P2 dallo stato STIMA_KM_RESIDUI, il tasto P3 dallo stato VISUALIZZA LIVELLO OLIO. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio ed il livello del carburante.
    - VISUALIZZA_TOT_KM: Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) ed è stato premuto una volta il tasto P1 dallo stato INIT. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio e le informazioni relative ai chilometri totali percorsi dall'automobile.
    - VISUALIZZA_PARZ_KM: Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) ed è stato premuto una volta il tasto P1 dallo stato VISUALIZZA_TOT_KM. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio e le informazioni relative ai chilometri parziali percorsi dall'automobile.
    - VISUALIZZA_VEL_MEDIA: Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) ed è stato premuto una volta il tasto P1 dallo stato VISUALIZZA_PARZ_KM. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio e le informazioni relative alla velocità media.
    - VISUALIZZA_CONSUMO_MEDIO : Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) ed è stato premuto una volta il tasto P2 dallo stato INIT. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio e le informazioni relative al consumo medio dell'automobile.
    - VISUALIZZA_CONSUMO_ISTANTANEO:Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) ed è stato premuto una volta il tasto P2 dallo stato VISUALIZZA_CONSUMO_MEDIO. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio e le informazioni relative al consumo istantaneo dell'automobile.
    - STIMA_KM_RESIDUI: Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) ed è stato premuto una volta il tasto P2 dallo stato VISUALIZZA_CONSUMO_ISTANTANEO. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio e le informazioni relative ai chilometri residui che l'automobile può percorrere.
    - VISUALIZZA_TEMP_ACQUA: Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) ed è stato premuto una volta il tasto P3 dallo stato INIT. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio e le informazioni relative alla temperature dell'acqua del sistema di raffreddamento del motore dell'automobile.
    - VISUALIZZA_LIVELLO_OLIO: Il display si trova in questo stato quando il sistema è stato avviato (STATO_SISTEMA=STANDBY o ACCESO) ed è stato premuto una volta il tasto P3 dallo VISUALIZZA_TEMP_ACQUA. In questo stato il display mostra il tachimetro, le eventuali spie di luci, cinture di sicurezza e olio e le informazioni relative livello dell'olio dell'automobile.



Il secondo automa si riferisce alla gestione dello stato del display, e definisce il comportamento ad alto livello del display per la visualizzazione definite nei casi d'uso e le interazioni con l'utente altraverso la pulsantiera.



Quest'ultimo viene implementato all'interno degli stati ACCESO e STANDBY dell'automa ad alto livello.

DIAGRAMMI COLLABORAZIONALI

L'analisi del sistema procede attraverso lo studio dettagliato degli scambi di messaggi tra gli oggetti presenti. Per descrivere queste comunicazioni sono utilizzati i diagrammi collaborazionali, che riportano gli oggetti delle classi, le interazioni e le collaborazioni (messaggi scambiati). E' in questo modo possibile accorgersi di eventuali classi, metodi o attributi mancanti. Le collaborazioni sono formate da una sequenza di messaggi caratterizzati da un numero d'ordine che permette di ricostruire il percorso delle comunicazioni tra gli oggetti. Per costruire i diagrammi collaborazionali in maniera corretta è necessario descrivere le comunicazioni seguendo le operazioni riportate nei corsi d'azione base del sistema. Inoltre ogni oggetto, quando manda un messaggio ad un altro oggetto, deve necessariamente utilizzare uno dei metodi di interfaccia definito nella classe cui l'oggetto destinatario fornisce. Per questo motivo seguendo i corsi d'azione base ed utilizzando i metodi delle classi si riesce potenzialmente a verificare il corretto funzionamento del sistema pur non avendolo ancora realizzato. Di seguito si riportano i diagrammi collaborazionali per i casi d'uso più significativi. Si è fatto riferimento alla simulazione tramite file e quindi non sono presenti le classi dei componenti HW.

DIAGRAMMA ACCENDILUCI

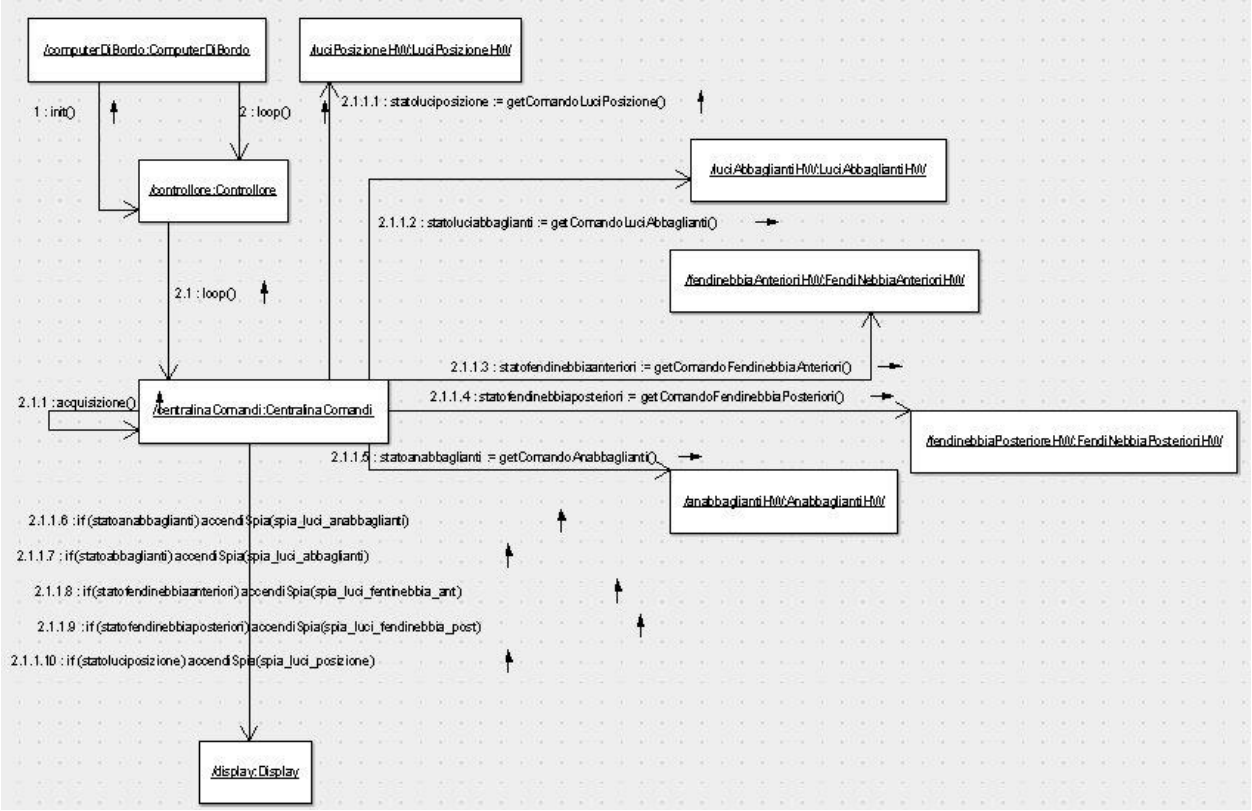


DIAGRAMMA ACCENDILUCIDIREZIONALI

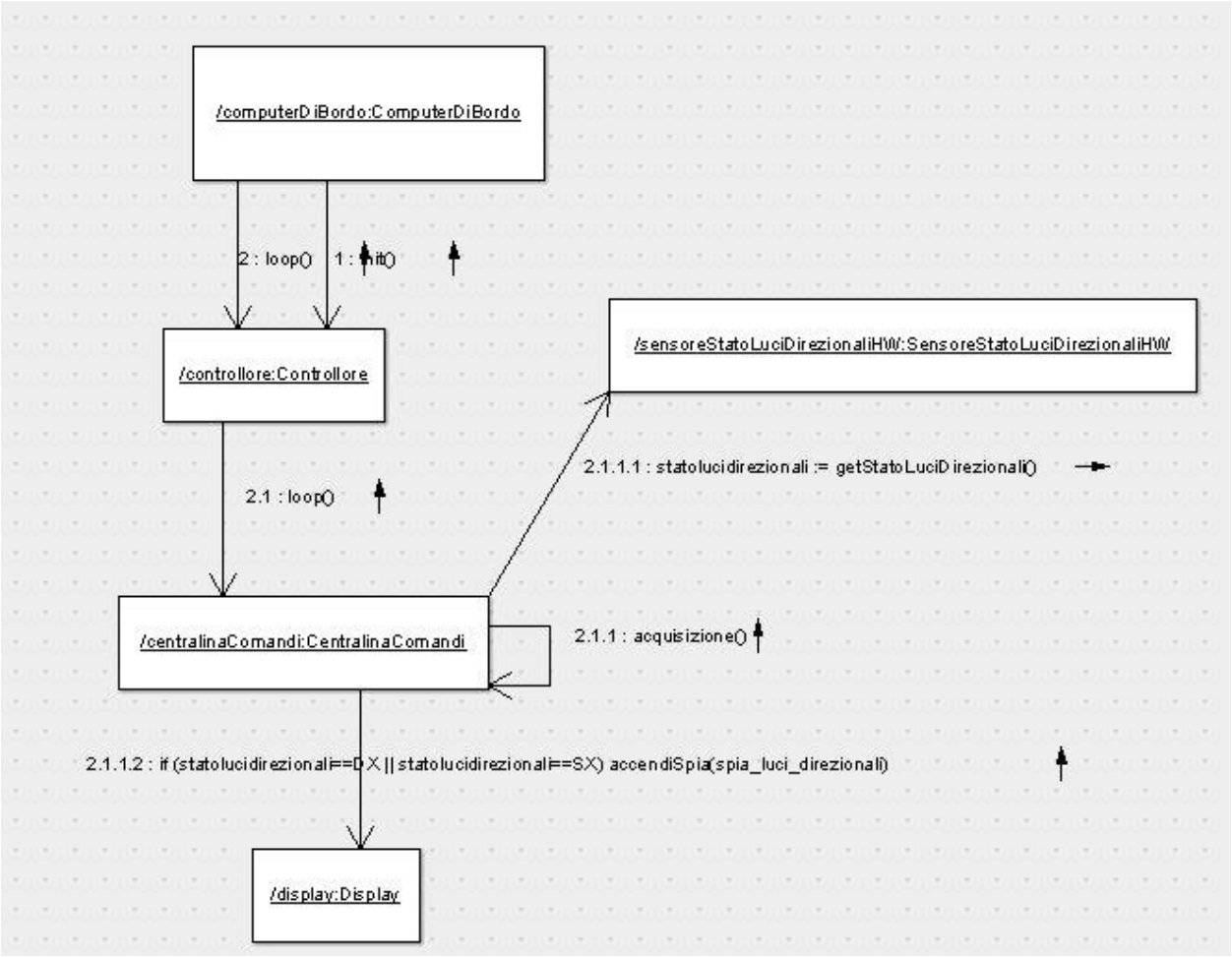


DIAGRAMMA ACCENDIMOTORI

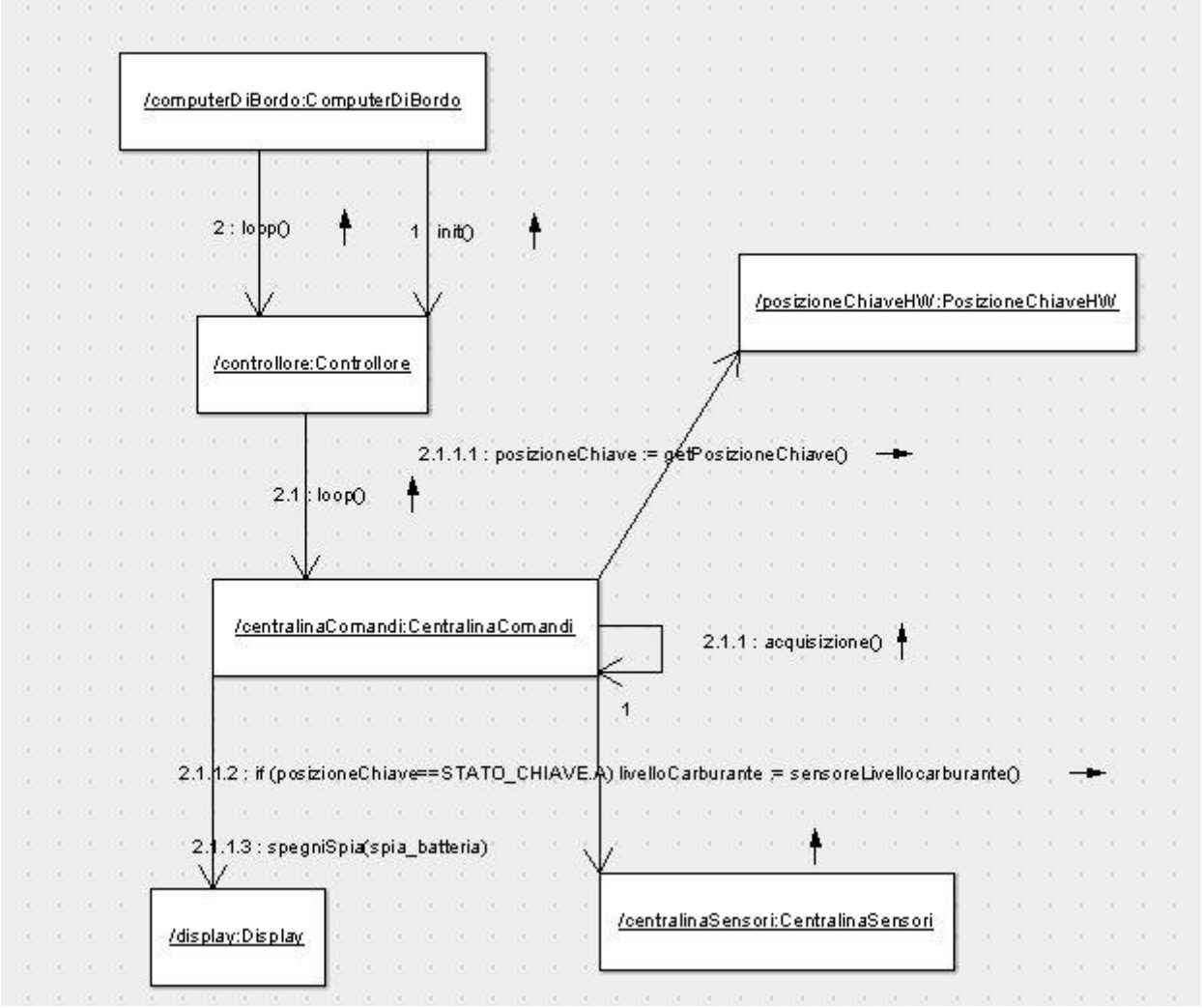


DIAGRAMMA ACCENDIQUADRO

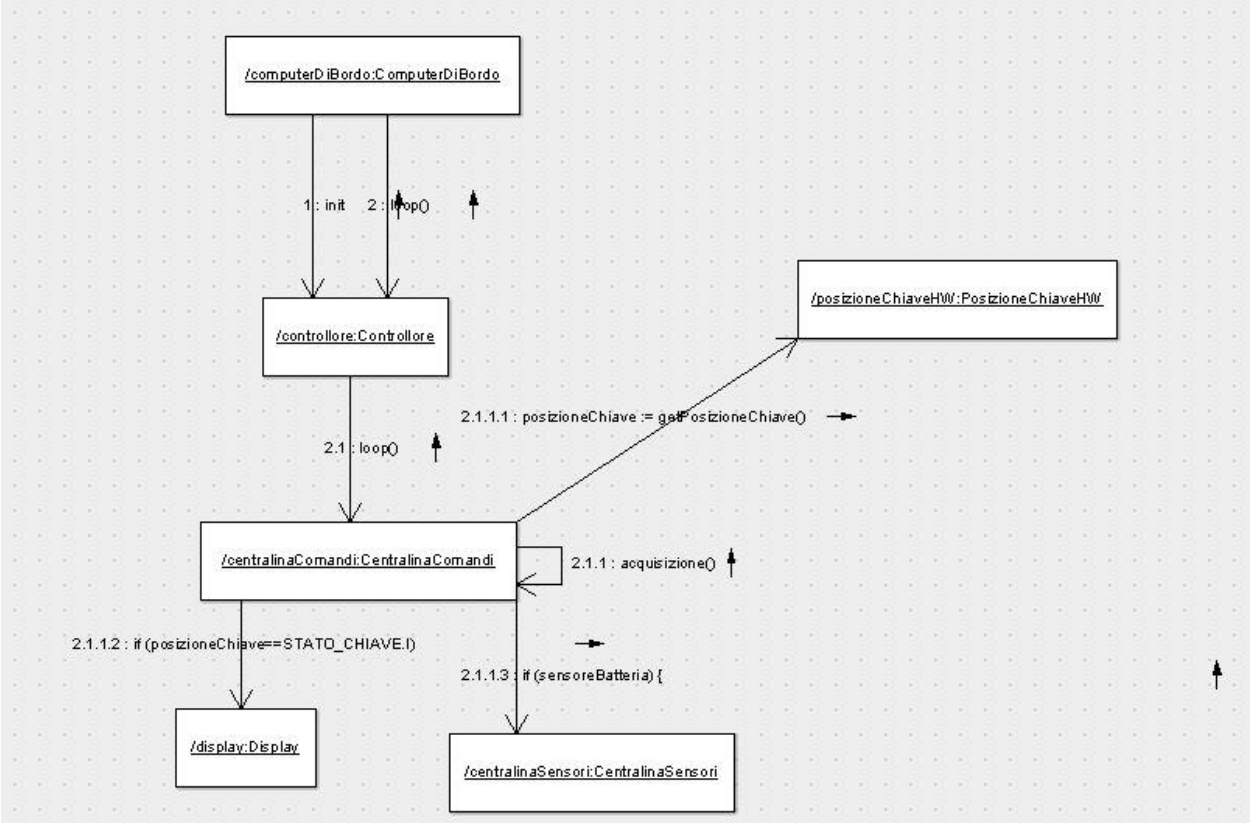
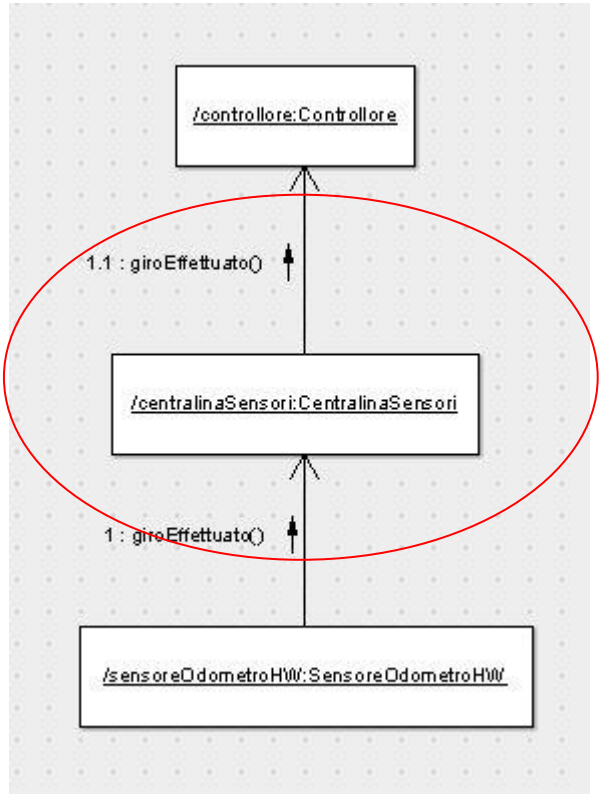


DIAGRAMMA GIROEFFETTUATO



Un interrupt??

DIAGRAMMA RETURNTOINIT

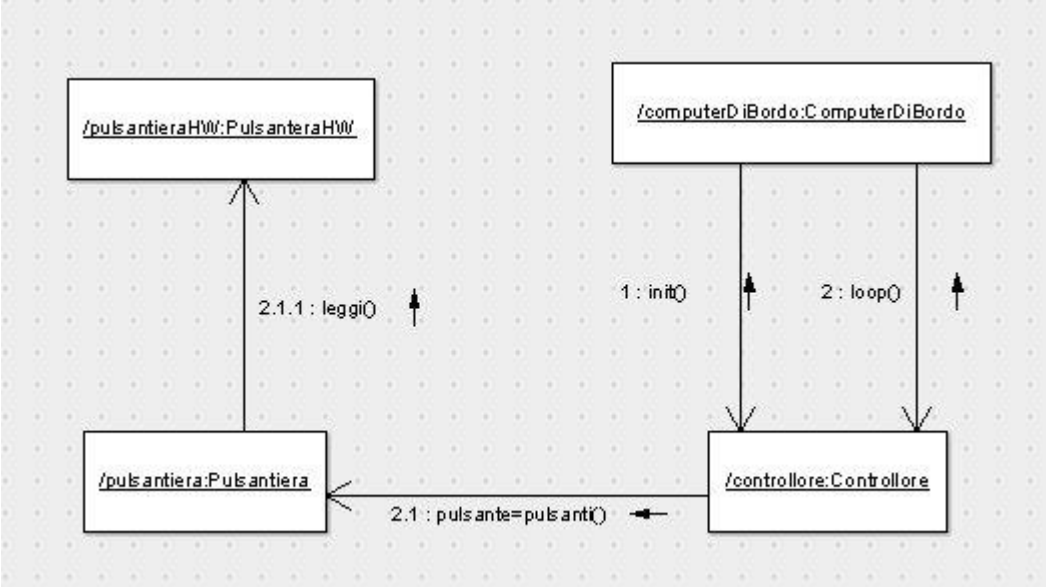


DIAGRAMMA SPEGNIQUADRO

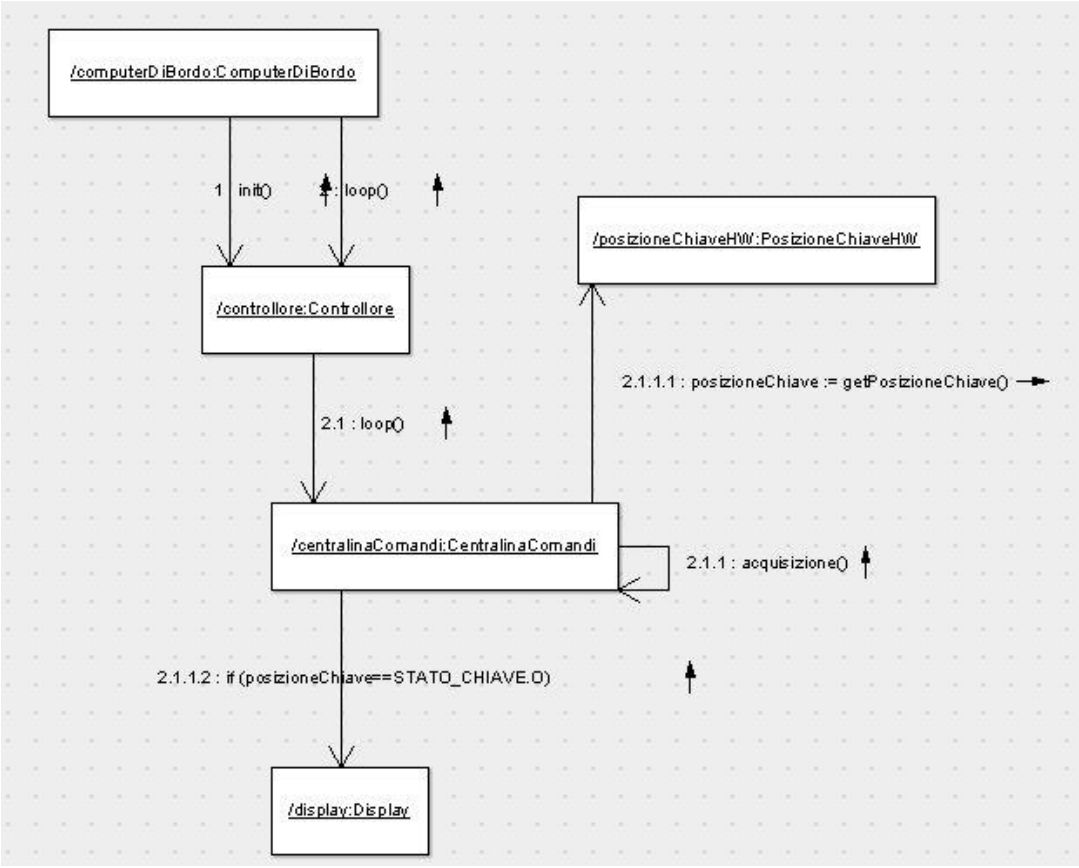


DIAGRAMMA VERIFICACINTURE

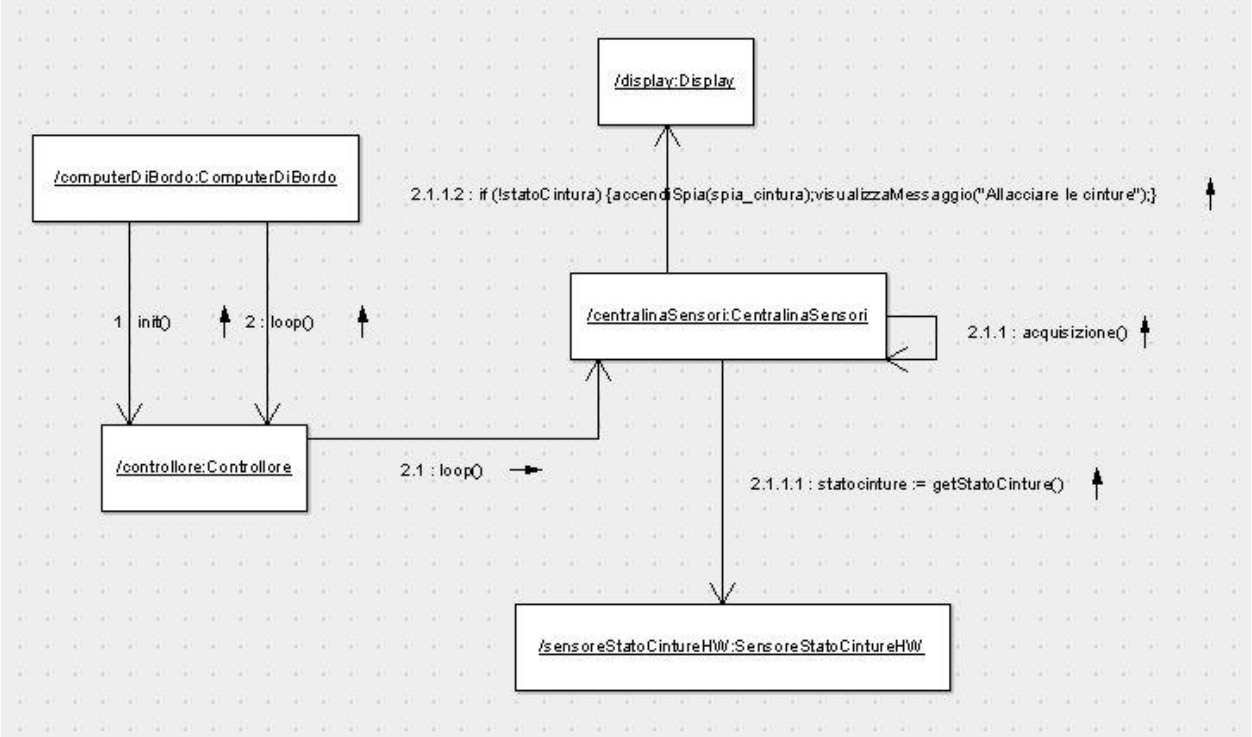


DIAGRAMMA VISUALIZZACONSUMOINSTANTANEO

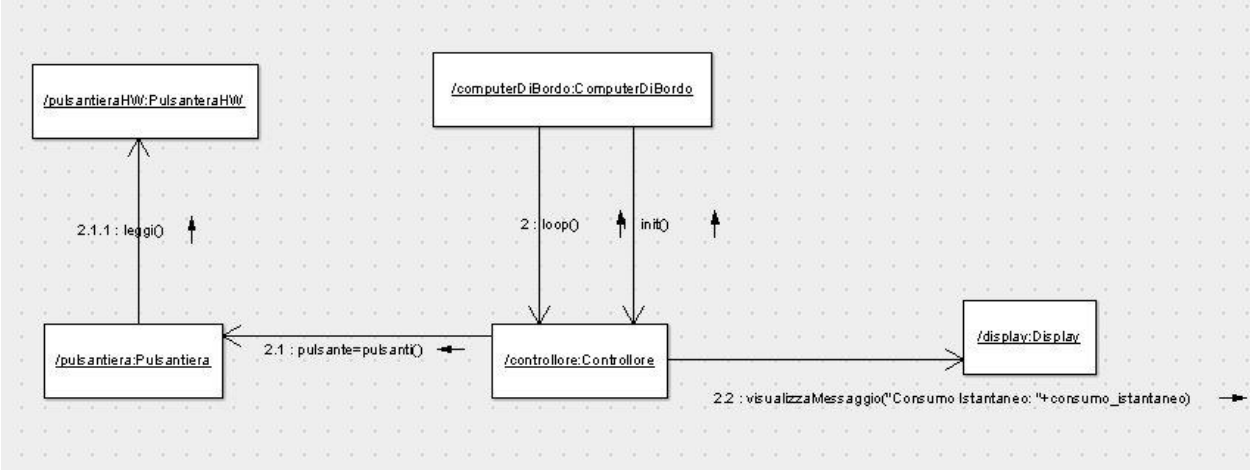


DIAGRAMMA VISUALIZZACONSUMOMEDIO

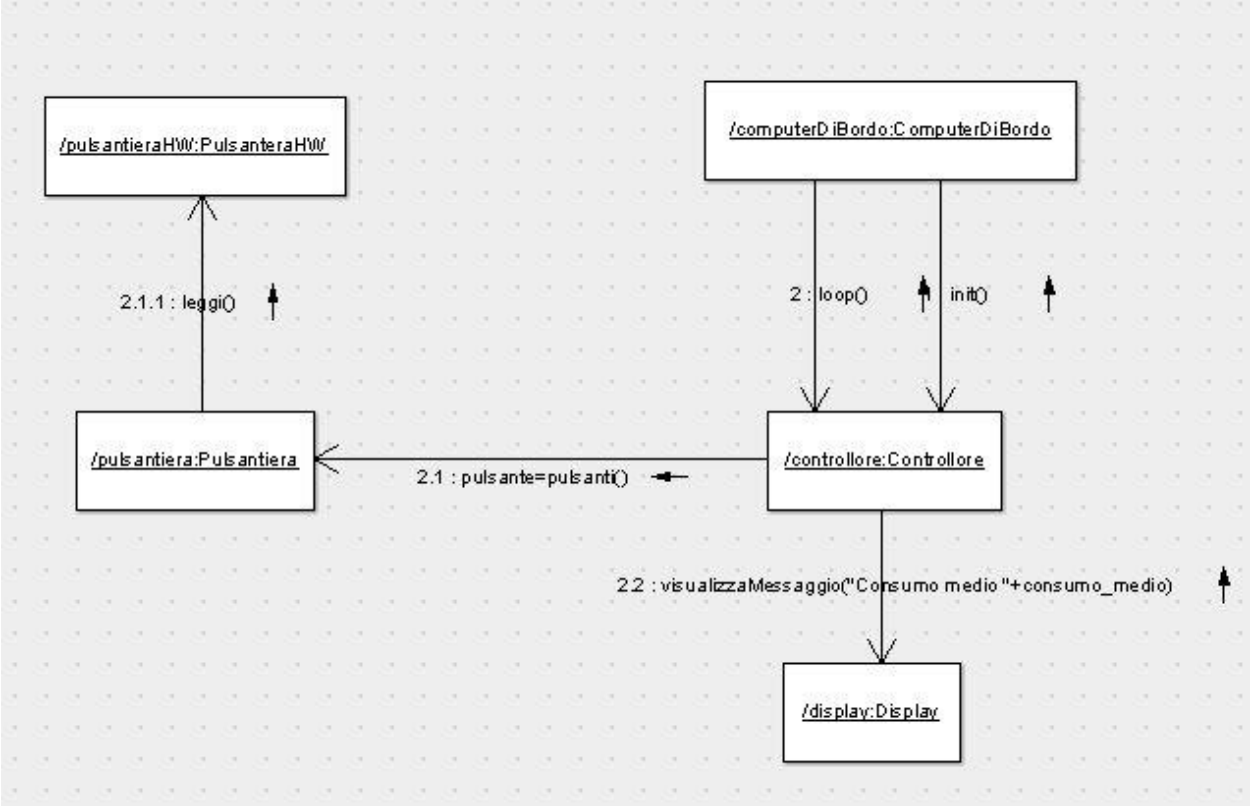


DIAGRAMMA VISUALIZZAKMPERCORSITOTALI

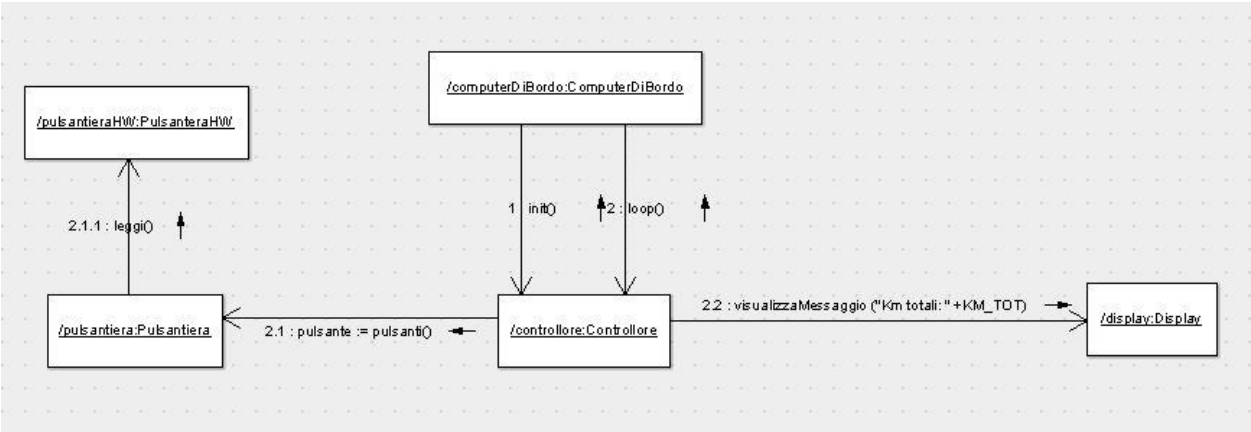


DIAGRAMMA VISUALIZZAKMPERCORSIVIAGGIO

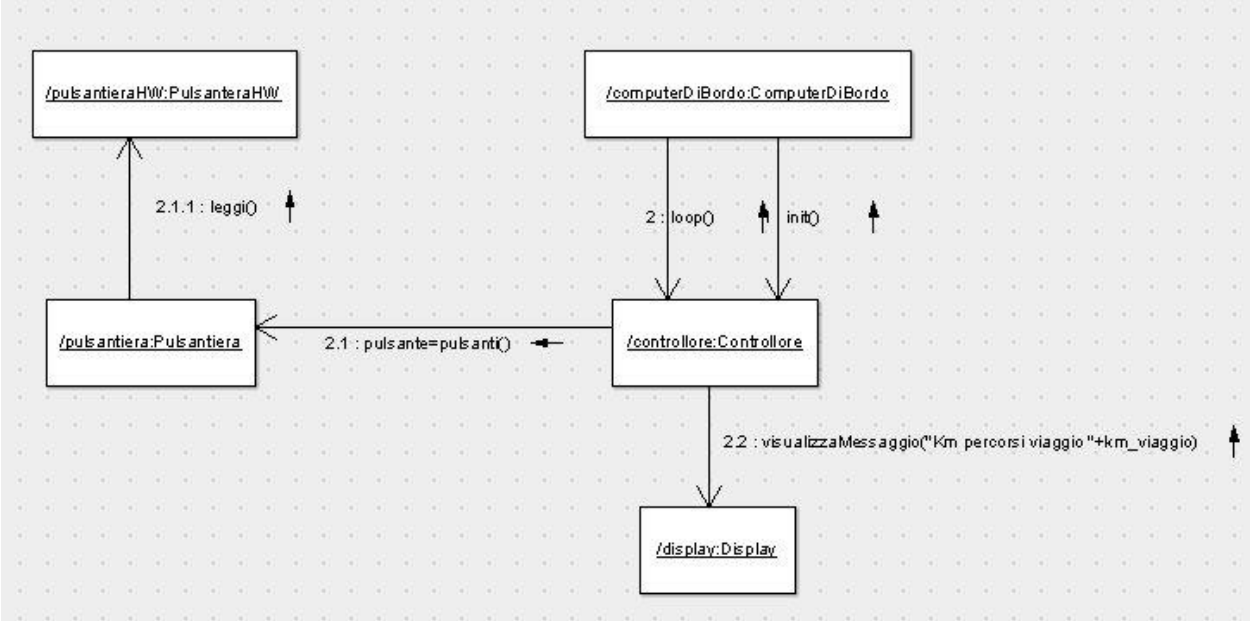


DIAGRAMMA VISUALIZZALIVELLOOLIO

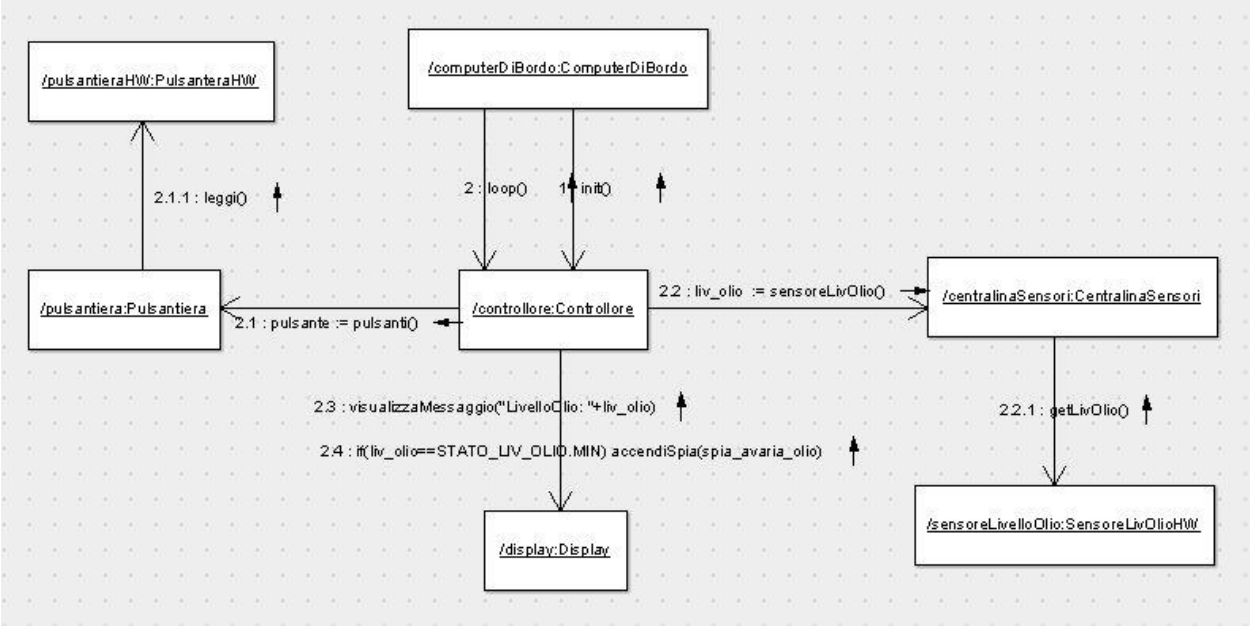


DIAGRAMMA VISUALIZZASTIMAKMRESIDUI

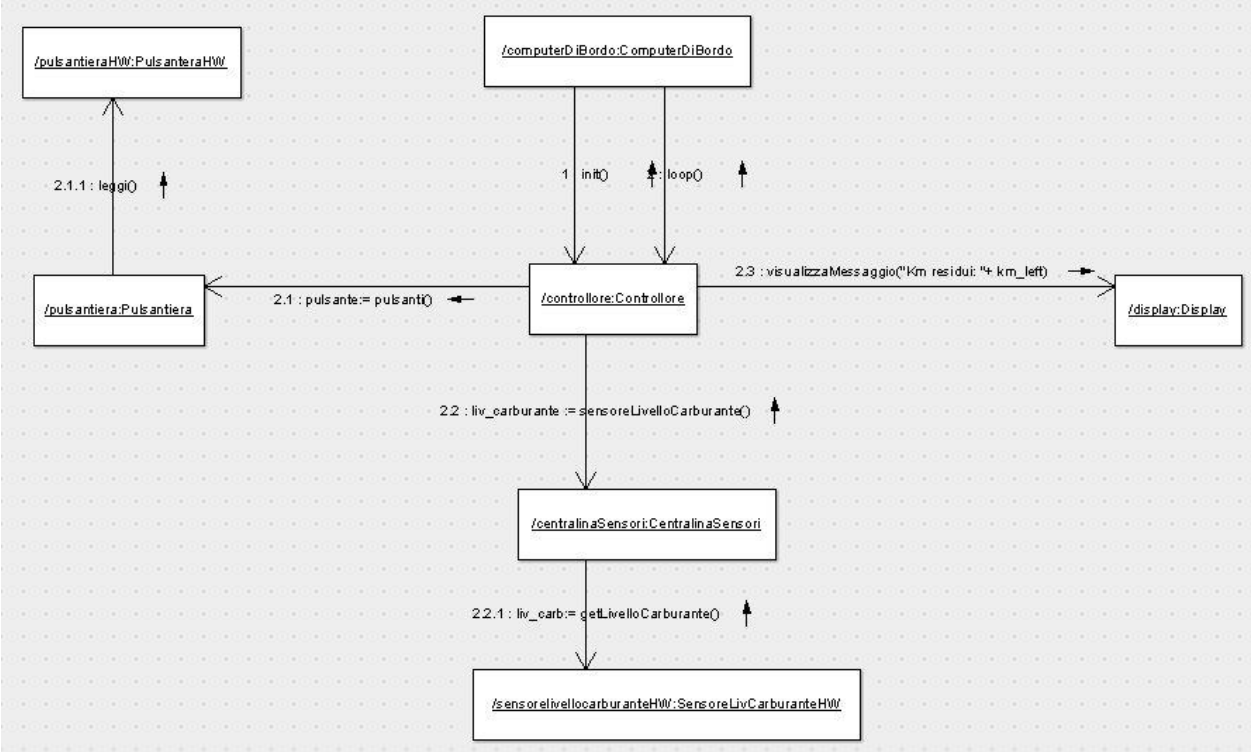


DIAGRAMMA VISUALIZZATEMPERATURAACQUA

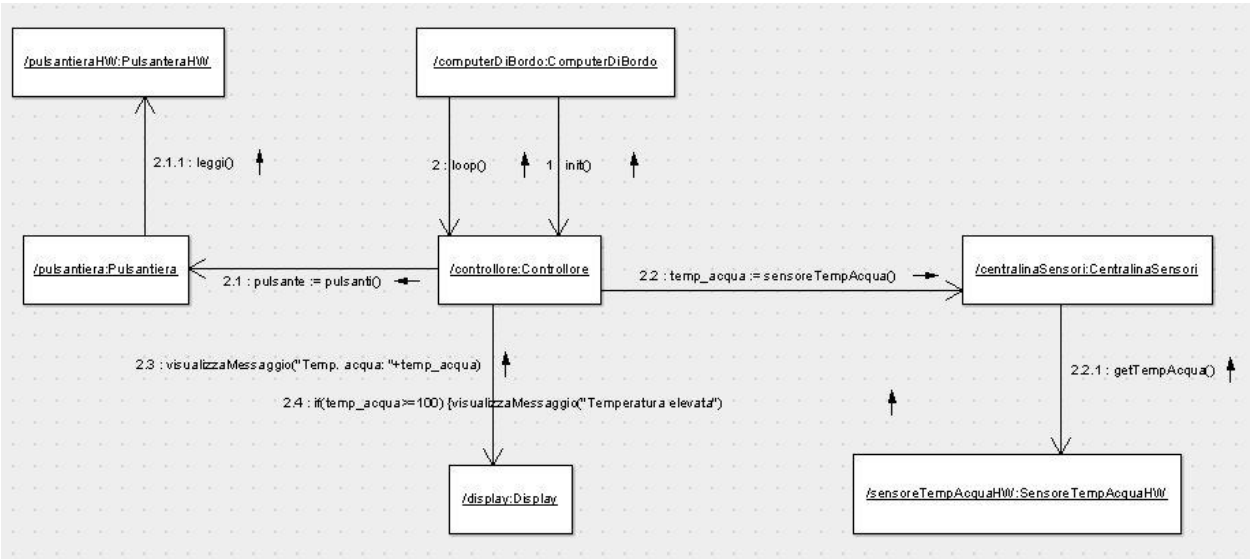


DIAGRAMMA VISUALIZZAVELMEDIA

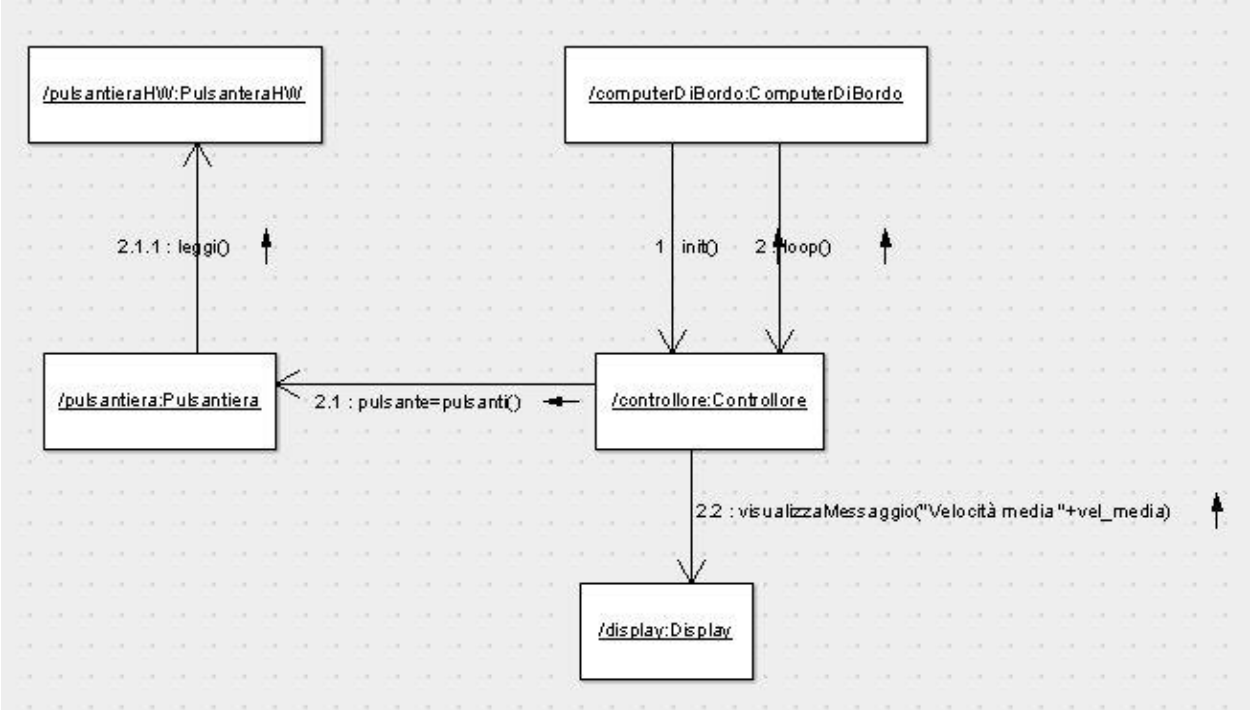
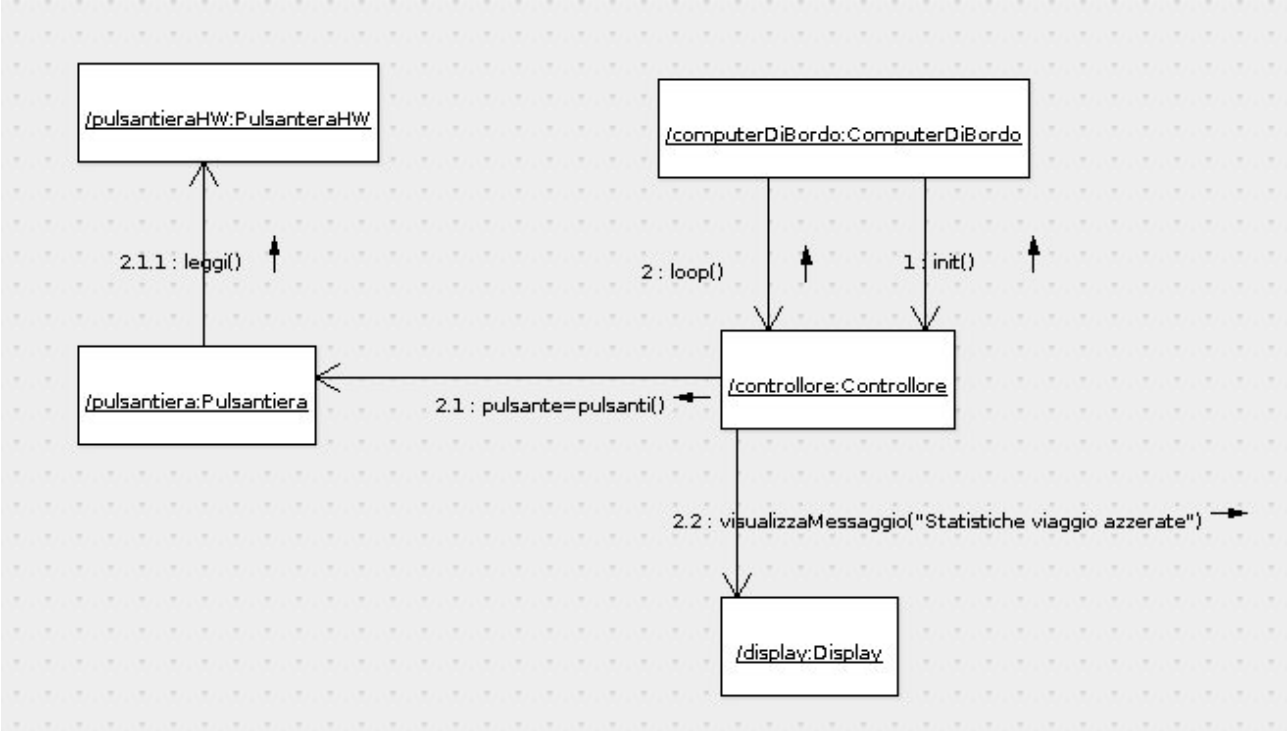


DIAGRAMMA AZZERASTATISTICHEVIAGGIO



MAPPE DI MEMORIA

I vari oggetti per poter comunicare tra di loro devono essere mappati in memoria in scrittura/lettura. Il collegamento tra il controllore e la centralina avviene tramite connessione con una porta seriale. Pertanto, per permettere la lettura dei dati, questi vengono scambiati tramite dei messaggi compliant allo standard RS232. Il formato dei messaggi in questione è il seguente:

[1 bit][8 bit (1 carattere)][8 bit (opzionali)]

Dove il primo bit sta ad indicare se l'operazione che si richiede è un'operazione di lettura (0) o di scrittura (1), gli 8 bit successivi servono ad indirizzare il sensore dal quale si vogliono ricevere le informazioni e i successivi 8 bit servono a portare eventuali dati. Nel progetto in questione, la lettura dei sensori avviene spesso in polling. Questo significa che il controllore manda alla centralina dei messaggi di lettura riguardanti tutti i sensori per i quali interessano i dati in questione, e la centralina risponde adeguatamente con un messaggio completo del dato da leggere.

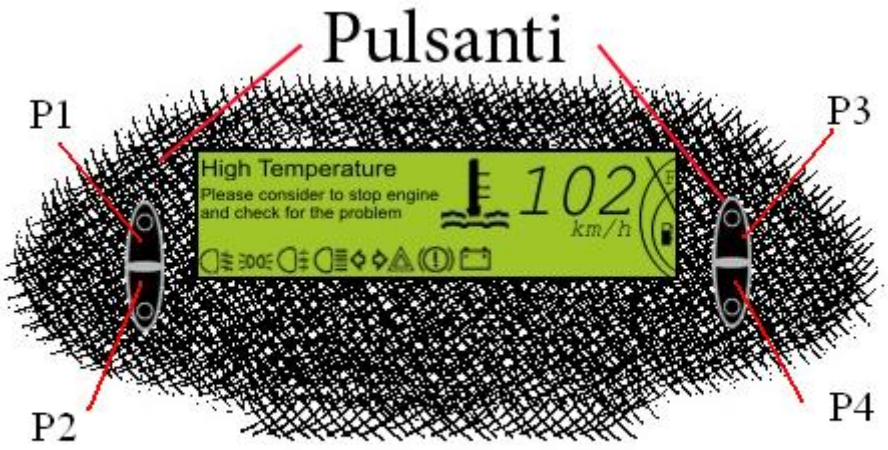
Descrizione	Indirizzo o messaggio RS232	Lettura	Scrittura
PulsantieraHW_leggi() **il byte restituito contiene l'identificativo del tasto premuto.**	0x03 (AD_LEGGI_PULS)	1 byte per il valore di ritorno	
CentralinaHW_accensione() **Il byte restituito contiene l'informazione sullo stato della chiave**	0a	0a+1byte per il valore.	
SensoreTempAcquaHW_getTempAcqua() **Il byte restituito contiene l'informazione sulla temperatura dell'acqua.**	0b	0b+1byte per il valore	
SensoreLivOlioHW_getLivOlio() **Il byte restituito contiene l'informazione sul livello dell'olio**	0c	0c+1 byte per il valore.	
CentralinaSensori_giroEffettuato() **Tale metodo si occupa di chiamare un metodo del controllore ogni volta che viene effettuato un giro completo della ruota.**			1d (in questo caso non serve alcun valore di ritorno)

SensoreLivCarburanteHW_getLivelloCarburante() **Il byte restituito contiene l'informazione sul livello del carburante**	0e	0e+1 byte per Il valore.
FendiNebbiaAnterioriHW **Il bit restituito contiene l'informazione sull'accensione dei fendinebbia anteriori**	0f	0f+1 bit per Il valore.
FendiNebbiaPosterioriHW **Il bit restituito contiene l'informazione sull'accensione dei fendinebbia posteriori**	0g	0g+1 bit per Il valore.
AnabbagliantiHW_getComandoAnabbaglianti() **Il bit restituito contiene l'informazione sull'accensione degli anabbaglianti**	0h	0h+1 bit per Il valore.
AbbagliantiHW_getComandoAbbaglianti() **Il bit restituito contiene l'informazione sull'accensione degli abbaglianti**	0i	0i+1 bit per il valore.
SensoreStatoLuciDirezionaliHW_getStatoLuciDirezionali() **Il byte restituito contiene le informazioni sull'accensione delle luci direzionali**	0j	0j+1 byte per il valore
SensoreStatoCintureHW_getStatoCinture() **Il bit restituito contiene le informazioni sullo stato delle cinture di sicurezza**	0k	0k+1 bit per il valore
SensoreStatoBatteriaHW_getStatoBatteria() **Il bit restituito contiene le informazioni sullo stato della batteria**	0l	0l+1 bit per il valore.
AnabbagliantiHW_getComandoAnabbaglianti() **Il bit restituito contiene le informazioni sulle luci di posizione**	0m	0m+1 bit per il valore.

HARDWARE E SIMULAZIONI

PULSANTIERA, PULSANTIERAHW E SIMULAZIONI

La pulsantieraHW è composta da 4 pulsanti, che vanno integrati nella struttura che comprende la pulsantiera ed il display.



Mock-up del computer di bordo con il display al centro e i pulsanti a lato

Un possibile esempio di componente utilizzabile il Keypads GFX Series - Customizable Keypads a 4 tasti della Storm Interface.

http://www.storm-interface.com/product_attachments/GFXSeries.pdf

La pulsantiera è stata simulata attraverso la tastiera del PC secondo la seguente corrispondenza:

· P1 ->'1'

· P2 ->'2'

· P3 ->'3'

· P4 ->'4'

Il metodo Pulsantiera_Pulsanti() richiama il metodo PulsantieraHW_leggi().

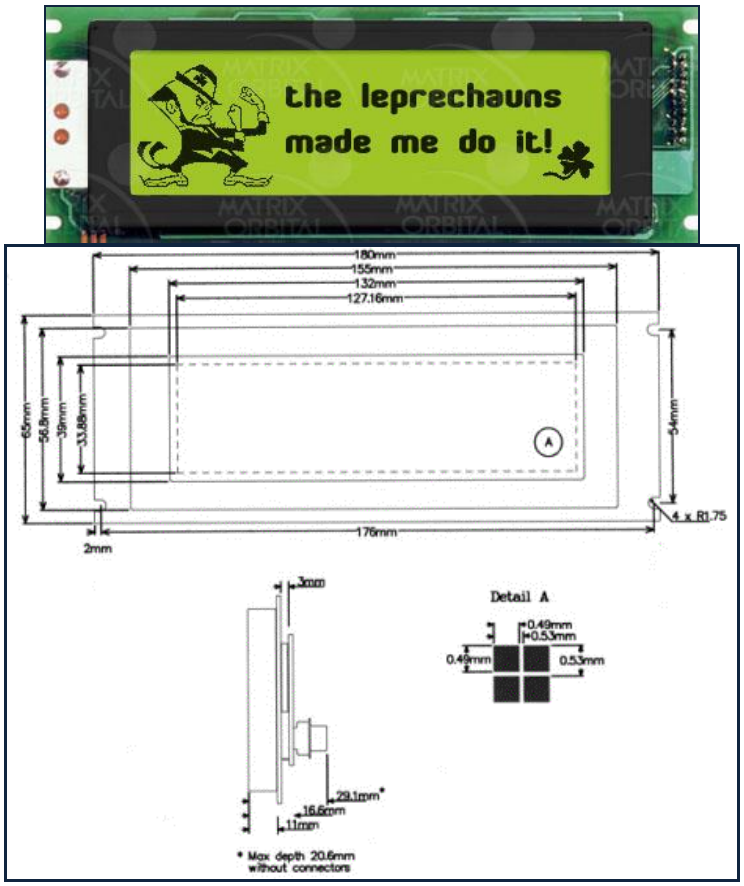
Il metodo PulsantieraHW_leggi(), nel caso di simulazione sofware, verifica la pressione di un tasto della tastiera del PC e, se questo è uno dei tasti sopra indicati, restituisce il codice ASCII corrispondente altrimenti restituisce 0.

Nel caso hw viene restituito il byte letto all'indirizzo AD_LEGGI_PULS

DISPLAY, DISPLAYHW E SIMULAZIONI

DESCRIZIONE FISICA

Per il progetto in esame è stato adottato un display LCD della *Matrix orbital*, il modello **GLC24064**.



Di seguito le caratteristiche tecniche:

- 240 x 64 pixel graphics display;
- Text display using built in or user supplied fonts;
- Adjustable contrast;
- Backlighting;
- RS-232 or I2C communications;
- Supply Voltage: 4.75 - 5.25 Vdc (Optional 7 - 30 Vdc);
- Supply Current: 31 mA typical;
- Supply Backlight Current: 160 mA typical;
- Pixel Layout 240 x 64 pixels XxY;
- Number of Characters 320 (maximum 40 characters x 8 Lines with 5x7 font);

- Display Area 127.16 x 33.88mm XxY;
- Dot Size 0.49 x 0.49mm (XxY);
- Dot Pitch 0.53 x 0.53mm (XxY);
- LED / CCFL Backlight Life 100, 000 hours typical;
- Color of Illumination Yellow Green (LED), Light Blue (CCFL);

Il suddetto LCD può comunicare con un host/controllore attraverso la porta seriale RS-232.

INSTALLAZIONE INIZIALE

Prima di utilizzare effettivamente il dispositivo e di installarlo sul cruscotto, è necessario configurare opportunamente la memoria interna del dispositivo stesso ed effettuale il setting di alcune caratteristiche fondamentali, quali ad esempio il set di font da utilizzare e le icone che verranno richiamate durante il suo l'utilizzo.

Sono richiesti, per l'installazione e la configurazione iniziale, i seguenti oggetti:

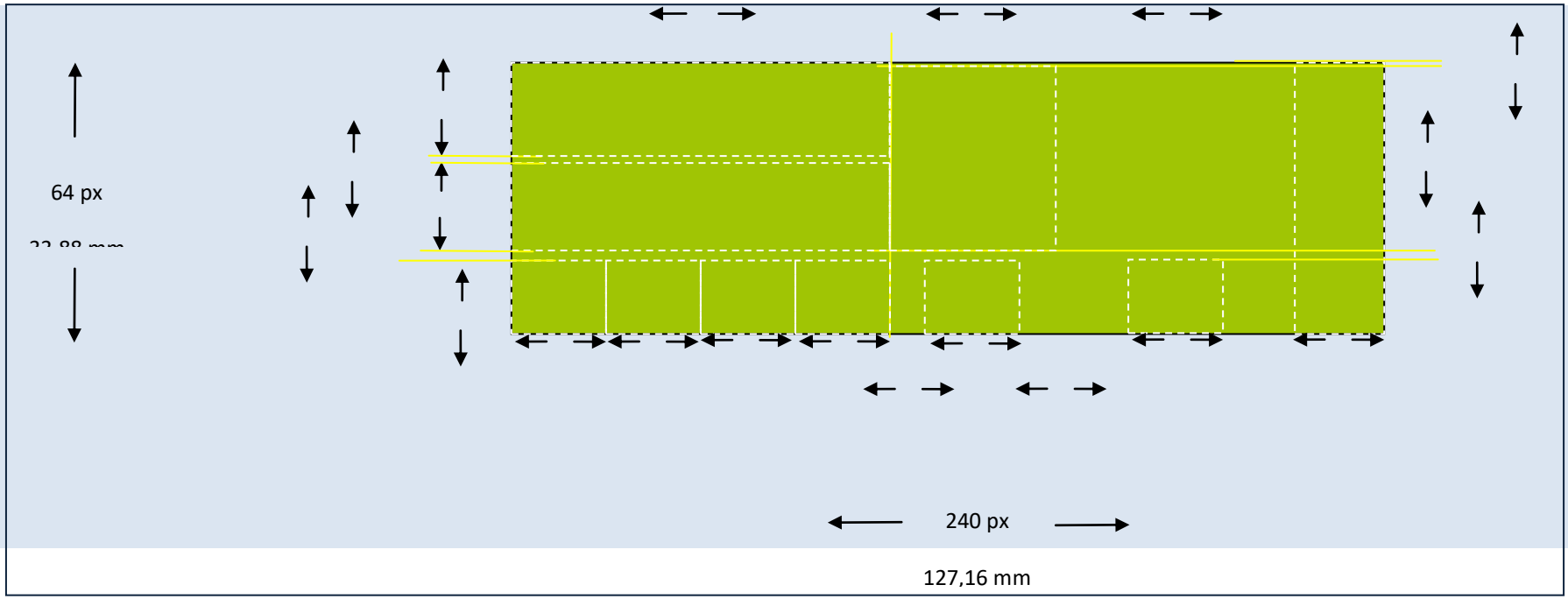
- Un alimentatore 5V (modelli da 8 a 30 VDC per "Efficient switching supply" - VPT);
- Un connectore del tipo usato per i floppy 3.5";
- Un PC con una porta RS-232 (COM1 o COM2);
- Il programma mogd.exe installato sull'host;
- Un cavo seriale RS-232 a 9 - 25 pin. Nel caso dell'utilizzo di un cavo 25 pin, è anche necessario un adattatore da 25 a 9 pin.

Si sottolinea che i seguenti oggetti NON sono necessari durante l'utilizzo del dispositivo, ma soltanto durante la sua configurazione iniziale. Ciò significa che non si deve pensare ad una loro improbabile collocazione nel cruscotto.

Matrix Orbital ha sviluppato un programma, **mogd.exe**, che si interfaccia con il dispositivo LCD e all'occorrenza configura opportunamente quest'ultimo (per quanto riguarda tutte le sue caratteristiche configurabili, quali installazione/download di set di caratteri e/o icone e/o grafica e/o settaggio del contrasto). **Anche se tale programma NON può essere utilizzato ovviamente una volta che il dispositivo è installato correttamente sul cruscotto (a progetto concluso),** esso comunque risulta uno strumento particolarmente interessante per quanto riguarda la preinstallazione di tutto il materiale grafico richiesto dal progetto stesso.

RAPPRESENTAZIONE LOGICA E LAYOUT

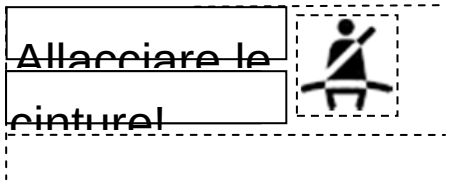
Di seguito viene rappresentato il layout logico del display con i dettagli di come viene rappresentata l'informazione al suo interno.



MESSAGGI DI TESTO

MODELLO LOGICO

I messaggi di teso nell'LCD sono organizzati su due righe allineate tra loro sulla parte in alto a sinistra. La dimensione disponibile riservata per le due righe è di 44 pixel in verticale e di 120 pixel in orizzontale. La dimensione del testo massima è pensata di 20 pixel in altezza e 8 il larghezza. Le due righe, dunque, vengono separate da 2 pixel (viene adottata una interlinea di 22 pixel) e il numero massimo di caratteri per riga è di 15.



MAPPATURA FISICA

L'LCD GLC24064 è corredato di un carattere di *default* di dimensione 5x7 pixel, con identificativo `0x01`.

Esso è contenuto nel file system del prodotto all'interno della memoria flash del prodotto. Il prodotto **GLC24064** dispone di memoria flash onboarde di una virtualizzazione delle risorse memorizzabili sotto forma file system.

Al fine di permettere il download sulla memoria flash di font e bitmap è stato sviluppato dalla Matrix Orbital un opportuno protocollo che supporta la comunicazione RS232. Questo viene fatto, come già accennato in precedenza, prima dell'installazione del dispositivo sul cruscotto, durante la sua prima inizializzazione. E' possibile, tramite la comunicazioen di opportuni comandi, gestire tale memoria flash onboard, come ad esempio effettuare la pulizia completa del file system con:

Wipe Filesystem

```
Syntax Hexadecimal 0xFE 0x21 0x59 0x21

Decimal 254 33 89 33

ASCII 254 "!" "Y" "!"
```

Oppure cancellare un file con:

Deleting a File

```
Syntax Hexadecimal 0xFE 0xAD [type] [refID]

Decimal 254 173 [type] [refID]

Parameters Parameter Length Description

type 1 Type of file (0:Font, 1:Bitmap)

refID 1 Reference ID of the file to delete.
```

Successivamente saranno descritte le funzioni del dispositivo inerenti alla installazione dei contenuti nel nostro caso. E' dunque possibile aggiungere, specificandolo, un nuovo set di caratteri che sia aderente alle proprie necessità. Come detto prima per il progetto in esame è stato pensato un carattere di altezza assoluta pari a 20 pixel. Un font generico viene definito completamente da un file di font, composto nel seguente modo:

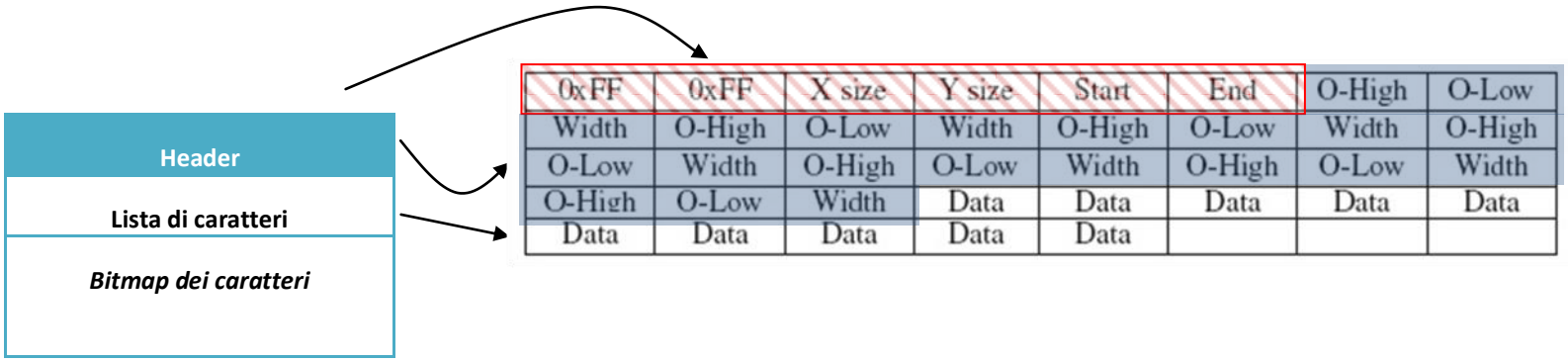


Figura 2 – Un file di font generico, a destra l'header rappresentato in rosso tratteggiato

L'header (in rosso tratteggiato) consta dei seguenti elementi:

- Un segnaposto di due byte che indica la fine del file;
- La larghezza nominale del carattere (1 byte);
- L'altezza assoluta del carattere (1 byte);
- Il valore ASCII del primo carattere (1 byte);
- Il valore ASCII dell'ultimo carattere (1 byte);

La lista di caratteri (in grigio) è, invece, è formata da gruppi di tre byte così composta:

- Offset nella bitmap dei caratteri (2 bytes);
- Larghezza del carattere corrente (1 byte);

Nel caso specifico occorrerebbe prima di tutto pensare alla creazione di un nuovo set di caratteri personalizzato o utilizzare un set di caratteri specifico che non risulti vincolato da copyright. Dopo aver progettato il set di caratteri a livello grafico occorre tradurre i vari caratteri in mappe di bit da inserire opportunamente nel dispositivo facendo l'upload di una tabella come in Figura 2 con i dati opportunamente specificati secondo il formato appena descritto.

Nel progetto corrente è stato deciso di rappresentare i seguenti 67 caratteri con una dimensione di **20x8** pixel:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**X**	**Y**	**Z**
a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**x**	**y**	**w**	**z**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**.**	**,**	**!**	**-**	**/**											

Per ognuno di essi deve essere presente una bitmap opportunamente progettata. Supponendo ad esempio di definire il carattere H minuscolo, dovremmo specificare una mappa di bit come in Figura 3.

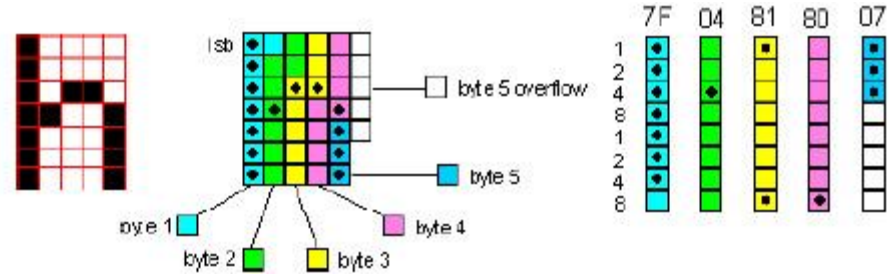


Figura 3 – Bitmap del carattere H minuscolo, i pallini rappresentano il valore 1, i vuoti lo 0

Il file del font finora descritto comporta una dimensione in memoria di **1547** byte, in quanto occorrono **6** byte per l'header, **201** byte per la lista di caratteri (3*67) e **1340** byte per la bitmap dei caratteri composti da 20x8 bit (in quanto rappresentabili da 20 byte e senza padding).

INSTALLAZIONE DEL FONT

Alla prima installazione del dispositivo occorre, dunque, fare l'upload in memoria della mappa del carattere da 20 pixel in memoria, attraverso il comando [`0xFE 0x24 [ref] [file size] [file data]`], dove `ref` è l'identificativo del font, nel nostro caso `0x02`, la dimensione del file è `1547` e i dati sono i 1547 byte successivi (cioè *0xFF 0xFF 0x0F 0x14 …*).

Qualora si verifichino malfunzionamenti al dispositivo (smemorizzazione della memoria flash) occorre rieffettuare la procedura di installazione iniziale. Questa è una condizione eccezionale (richiesta di l'intervento di assistenza tecnica) non trattata ivi esplicitamente.

Occorre successivamente effettuare la selezione del carattere interessato utilizzando il comando "`Set Current Font`", ovvero `[254 49 [font ID]]` che nel caso specifico sarebbe `254 49 02`.

SCRITTURA DEL MESSAGGIO

La scrittura del messaggio viene effettuata carattere per carattere rich8iamando l'opportuno metodo/funzione offerto dal set di istruzioni del dispositivo stesso, le più importanti delle quali sono descritte nella tabella che segue. Nella colonna ***semantica*** è descritta la sintassi del comando prima in esadecimale, poi in decimale e infine in decimale con codice ASCII.

Tipo di funzione	Sintassi	Descrizione
Set Current Font	FE 31 [font id] 254 49 254 '1'	Setta il tipo di carattere in base a quelli precaricati in memoria o installati dall'utente (come precedentemente accennato).
Auto scroll on	FE 51 254 81 254 'Q'	Avvia lo scroll automatico al fondo dello schermo. Lo scorrimento comporta la possibilità di ottenere spazio per una nuova linea di testo.
Auto scroll off	FE 52 254 82 254 'R'	Disattiva lo scorrimento automatico
Set text insertion point	FE 47 [col] [row] 254 71 [col] [row] 254 'G' [col] [row]	Setta il punto di inserimento di testo usando la dimensione di base del carattere corrente
Set text insertion point to top left	FE 48	Sposta il cursore in alto a destra (basato sulla metrica del carattere corrente)

	254 72 254 'H'	
Set text insertion point using pixel values	FE 79 [x][y] 254 121 [x][y] 254 'y' [x][y]	Setta il cursore nella posizione (x, y coordinate dello schermo in pixel partendo dall'alto a sinistra).
Set font metrics	FE 32 [metrics] 254 50 [metrics] 254 '2' [metrics]	Setta la metrica per il carattere corrente

TACHIMETRO

Per il tachimetro sono invece disponibili 60 pixel, quindi virtualmente uno spazio di 7 caratteri per riga, che permettono un layout come quello rappresentato in figura:

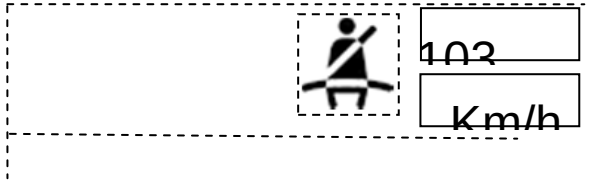


Figura 4 - Rappresentazione logica del tachimetro

Per i font vale lo stesso discorso affrontato per i messaggi a video.

GRAFICA (ICONE)

Il sistema progettato interagisce tramite l'utente con dei messaggi di testo corredati da icone che convogliano meglio le segnalazioni e lo stato del sistema. Il display **GLC24064** permette l'utilizzo di bitmap B/N attraverso alcune funzioni: le più importanti delle quali sono descritte nella tabella che segue. Nella colonna ***semantica*** è descritta la sintassi del comando prima in esadecimale, poi in decimale e infine in decimale con codice ASCII.

Tipo di funzione	Sintassi	Descrizione
Uploading a Bitmap File	0xFE 0x5E [refID] [size] [data] 254 94 [refID] [size][data] 254 "^" [refID] [size] [data]	Il GLC24064 è in grado di immagazzinare fino a sessanta-quattro file di font e bitmap combinati. Per caricare una bitmap sul GLC24064 è necessario prima avviare il comando caricare file di font (0xFE 0x5E), si deve poi passare un numero di identificazione di riferimento, che deve essere univoco per ogni font sul display modulo. L'ultimo parte del caricamento di una bitmap è la trasmissione del file bitmap dati (per istruzioni dettagliate su come caricare un file per vedere il manuale di riferimento del GLC24064 a pagina 39).
Drawing a Bitmap from Memory	0xFE 0x62 [refID] [X] [Y] 254 98 [refID] [X] [Y] ASCII 254 "b" [refID] [X] [Y]	Questo comando mostra una bitmap che si trova nella memoria onboard. La bitmap è referenziata dal bitmap di riferimento numero di identificazione,che è stabilito quando la bitmap viene caricato il modulo di visualizzazione. La bitmap verrà stilata all'inizio in alto a sinistra, da specificate coordinate X, Y.

MODELLO LOGICO

Sono stati previsti due tipologie di icone, le icone di dimensione grande (40x40 pixel). e quelle di dimensione piccola (28x20 pixel).

CONSIDERAZIONI AGGIUNTIVE

Il display in oggetto offre una molteplicità di primitive che rendono semplificata la rappresentazione di informazioni nello stesso.

MICROCONTROLLORE

Il microcontrollore utilizzato è SX ARM Linux (http://www.areasx.com/index.php?D=1&id=8174).

"La SX ARM Linux è una scheda di controllo basata su microprocessore STR9104 (ARM9-core) a 180MHz con preinstallato un sistema operativo Linux (kernel 2.6), programmabile in C/C++ tramite ARM cross-compiler ed in basic grazie all'interprete BBC Basic V.
Dotata di due interfacce ethernet 10/100, due porte USB 2.0 e due porte seriali DB9 maschio viene fornita già inscatolata in un pratico e robusto case di metallo con alette di fissaggio."

Microcontrollore SX ARM Linux

Di seguito vengono riportate le specifiche tecniche hardware e software:

Specifiche Hardware
CPU: Star STR9104 (ARM9-core)
Flash: 8MB
RAM: 32MB SDRAM
EEPROM: 16Kb
RTC: si
Buzzer: si
Ethernet: 2 porte RJ45 10/100 mbps
Serial: 2 porte COM1 e COM2 RS-232/422(4 wire)/485 (configurabili via software)
USB: 2 porte USB 2.0 host
H/W Reset SW: hardware reset
DIP S/W1 x2: la posizione 1 e 2 sono connesse al PIO e gestite via programma
Alimentazione: +9VDC~48VDC (~300mA@12V)
Temperatura operativa: da 0 a 50 °C
LAN LED: Stato della LAN Link/Activity
Ready LED: Controllabili via software

Specifiche Software
Sistema operativo : Linux, kernel 2.6.x
Boot Loader: U-Boot 1.1.2
File Systems: JFFS2, ETX2/ETX3, VFAT/FAT, NFS

- Protocolli supportati

IPv4, ICMP, ARP, DHCP, NTP, TCP, UDP, FTP,
Telnet, HTTP, PPP, PPPoE, CHAP, PAP, SMTP, SNMP V1/V3, SSL, SSH 1.0/2.0

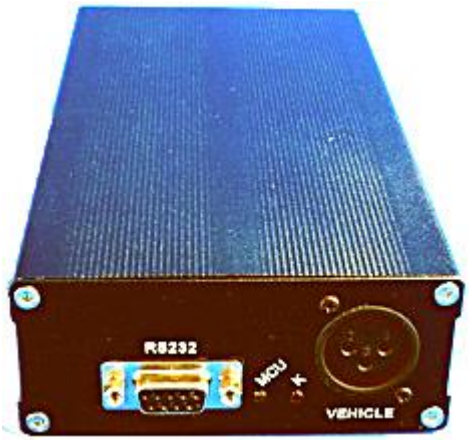
- Utility preinstallate
bash: shell command
tinylogin: login e user manager utility
telnet: Telnet client
busybox: Linux utility
ftp: FTP client
wi-fi: WiFi Tools

- Demoni attivi
pppd: Dial In/out su porta seriale & PPPoE
snmpd: SNMP agent
telnetd: server Telnet
inetd: server TCP
ftpd: server FTP
boa: Web server
sshd: secured shell server
iptables: Firewall
exmd: Expert manager daemon

- Drivers disponibili
SD/MMC (su usb), UART, Real Time Clock, Buzzer, Digital I/O, Ethernet, Watchdog Timer
- USB Host Drivers
Flash disk, WiFi (supporta adattatori USB con chipset RT2570), e convertitori USB - RS-232

CENTRALINA SENSORI, HW E SIMULAZIONI

La centralina sensori viene collegata ad i vari sensori e serve per leggerne i valori ad inviarli al controllore. La centralina invia tali dati tramite un cavo RS232. Per tali scopi, abbiamo deciso di utilizzare il componente commerciale **Mercedes Benz Carsoft 7,4 ,** in vendita presso :
http://carsoft.en.busytrade.com/selling_leads/info/383717/MB_Interface_for_Carsoft_7_4.html



Tale centralina possiede le seguenti caratteristiche:

7,4 MB Carsoft interfaccia Multiplexer
• OBD2 connessione via cavo (tutti i K-Lines cablata)
• 38 MB di pin di connessione via cavo (tutti i K-Lines cablata)
• MB Sprinter connessione via cavo (tutti i K-Lines cablata)
• MB 3 pin cavo di banane
• RS232 cavo di prolunga

Multiplexer contiene interfaccia per la connessione a tutti i moduli diagnosticable completamente automatico.

• LED di stato per il potere, MCU e K-Line status
• telaio in alluminio
• DB37 connessione al veicolo
• Comprende tutti i cavi tra cui anche Sprinter via cavo
• Tutti i cavi con una lunghezza di 1,5 metri

I sensori vengono collegati a un concentratore, il quale viene attaccato alla centralina. Questicome descritto in precedenza, essendo esterni al sistema vengono modellati come attori. Per tale motivo non vengono descritti in dettaglio e non vengono specificate le funzionalità per ognuno di essi. La lettura dei valori relativi a tali sensori da parte del controllore, viene simulata tramite la lettura di file di testo opportunamente creati. Per maggiori dettagli rimandiamo al paragrafo relativo alla suddivisione tra hardware e software. Elenchiamo di seguito quali sono i sensori di interesse per il nostro sistema, e le informazioni che questi ritornano:

- ***Sensore per il controllo dell'accensione dell'auto(chiave):*** Tale sensore viene simulato attraverso il file "PosizioneChiave.txt", e serve a controllare la modalità di accensione dell'automobile ove il computer di bordo è installato, ovvero "standby" o "motore acceso". Tale informazione è ottenuta in polling sull'apposito hardware. Il metodo *CentralinaHW_accensione()*, nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file. Questo rappresenta la lettura che la centralina deve effettuare dall'apposito hw di rilevazione dell'accensione dell'auto. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- **Sensore per la misurazione della temperatura dell'acqua**: Tale sensore viene simulato attraverso il file "tempacqua.txt", e serve a controllare la temperatura dell'acqua del radiatore. Il metodo SensoreTempAcquaHW_getTempAcqua() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file e ne legge i valori che sono presenti all'interno. Tali valori vengono letti in maniera randomica e rappresentano i valori della temperatura dell'acqua per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- **Sensore per la misurazione del livello dell'olio**: Tale sensore viene simulato attraverso il file "livelloolio.txt", e serve a controllare il livello dell'olio presente nel serbatoio. Il metodo SensoreLivOlioHW_getLivOlio() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file e ne legge i valori che sono presenti all'interno. Tali valori vengono letti in maniera randomica e rappresentano i valori del livello dell'olio per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- **Sensore per la misurazione della velocità (Odometro)**: Tale sensore viene simulato attraverso il file "odometro.txt" e serve a controllare l'odometro. All'interno di tale file è presente, nel caso di simulazione software, la velocità attuale del sistema. La classe SensoreOdometroHW, si occupa di chiamare il metodo CentralinaSensori_giroEffettuato() ogni volta che la ruota effettua un giro, ed è compito della centralina di calcolare la velocità effettiva a partire dal numero di giri effettuati in un certo lasso di tempo. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale al controllore (vedi mappa di memoria).

- **Sensore per la misurazione del livello del carburante**: Tale sensore viene simulato attraverso il file "livellocarburante.txt", e serve a controllare il livello del carburante presente nel serbatoio. Il metodo SensoreLivCarburanteHW_getLivelloCarburante() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file e ne legge i valori che sono presenti all'interno. Tali valori vengono letti in maniera randomica e rappresentano i valori dei possibili livelli di carburante per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- **Sensore per il controllo delle luci fendinebbia anteriori**: Tale sensore viene simulato attraverso il file "lucifendinebbiaanteriori.txt", e serve a controllare l'accensione delle luci fendinebbia anteriori. Il metodo AnabbagliantiHW_getComandoAnabbaglianti() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file, la cui presenza rappresenta l'accensione delle luci fendinebbia anteriori per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- **Sensore per il controllo delle luci fendinebbia posteriori**: Tale sensore viene simulato attraverso il file "lucifendinebbiaposteriori.txt", e serve a controllare l'accensione delle luci fendinebbia posteriori. Il metodo AnabbagliantiHW_getComandoAnabbaglianti() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file, la cui presenza rappresenta l'accensione delle luci fendinebbia posteriori per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- **Sensore per il controllo delle luci anabbaglianti:** Tale sensore viene simulato attraverso il file "lucianabbaglianti.txt", e serve a controllare lo stato delle luci anabbaglianti. Il metodo AnabbagliantiHW_getComandoAnabbaglianti() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file, che rappresenta l'accensione delle luci anabbaglianti per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- **Sensore per il controllo delle luci abbaglianti:** Tale sensore viene simulato attraverso il file "luciabbaglianti.txt", e serve a controllare lo stato delle luci abbaglianti. Il metodo LuciAbbagliantiHW_getComandoAbbaglianti() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file, che rappresenta l'accensione delle luci abbaglianti per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- **Sensore per il controllo delle quattro frecce:** Tale sensore viene simulato attraverso il file "quattrofrecce.txt", e serve a controllare lo stato delle quattrofrecce. Il metodo QuattroFrecceHW_getComandoQuattroFrecce() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file, che rappresenta l'accensione delle quattro frecce per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- Sensore per il controllo delle luci direzionali: Tale sensore viene simulato attraverso il file "lucidirezionali.txt", e serve a controllare lo stato delle luci direzionali. Il metodo SensoreStatoLuciDirezionaliHW_getStatoLuciDirezionali() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file, e ne legge il contenuto in modo randomico. Tale contenuto rappresenta lo stato delle luci direzionali per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- Sensore per il controllo dello stato di aggancio delle cinture di sicurezza: Tale sensore viene simulato attraverso il file "sensorecinture.txt" e serve a controllare se le cinture di sicurezza sono allacciate correttamente. Il metodo SensoreStatoCintureHW_getStatoCinture() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file. La presenza o no del file rappresenta l'informazione sulle cinture per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

- Sensore per il controllo della batteria: Tale sensore viene simulato attraverso il file "controllobatteria.txt", e serve a controllare il corretto funzionamento della batteria. Il metodo SensoreStatoBatteriaHW_getStatoBatteria() nel caso di simulazione sw, verifica la presenza del contenuto del suddetto file e ne legge i valori che sono presenti all'interno. Tali valori vengono letti in maniera randomica e rappresentano i valori dello stato della batteria per un determinato momento. Nel caso hw, viene viene inviato un messaggio opportuno tramite la porta seriale alla centralina (vedi mappa di memoria).

PARTIZIONAMENTO HARDWARE/SOFTWARE

CONTROLLORE

È un oggetto software che si interfaccia con tutti gli oggetti hardware ad eccezione dei sensori. I sensori infatti si collegano alla centralina. Questo oggetto è il nucleo del sistema poiché tutti i flussi dati vengono processati da esso, come si può notare dai diagrammi collaborazionali.

CENTRALINA

E' un oggetto software che si collega con il controllore e con i sensori. In pratica serve da wrapper tra questi due, in quanto permette la lettura dei valori presenti nei sensori.

DISPLAY E DISPLAYHW

L'oggetto DisplayHW è un oggetto hardware, si interfaccia con il controllore tramite l'oggetto software Display. Il DisplayHW è strutturato da 4 righe e 40 colonne, in ogni cella sarà visualizzato un carattere.

Il DisplayHW è stato simulato utilizzando lo schermo del PC.

PULSANTIERA E PULSANTIERAHW

La PulsantieraHW è un oggetto hardware, si interfaccia con il controllore tramite l'oggetto software Pulsantiera.

La PulsantieraHW è stata simulata utilizzando la tastiera del PC con la seguente corrispondenza:

- P1 ->'1'
- P2 ->'2'
- P3 ->'3'
- P4 ->'4'

CODICE SORGENTE I FILE .H VANNO MESSI PRIMA DEI .C

ANABBAGLIANTIHW.C

```
#include "AnabbagliantiHW.h"



boolean AnabbagliantiHW_getComandoAnabbaglianti()

{

                #ifdef MICRO//caso HW

                //legge il valore all'indirizzo AD_LUCIANABBAGLIANTI_GETT (0x11)

                uchar val;

                val=*AD_LEGGI_PULS;

                return val;

                #endif


                #ifdef PC

                        boolean ret=FALSE;

                        char buffer[BUFMAX]={0};

                        FILE *anabbaglianti;


                        uint letto = 0;


                        if (!(anabbaglianti=fopen("./interfacce/Anabbaglianti.txt", "rw" ))){

                                printf("File not found - interfacce/Anabbaglianti.txt\n");

                                fflush(stdout);

                                return 0;

                        }


                        fgets(buffer, BUFMAX, anabbaglianti);

                        fclose(anabbaglianti);


                        sscanf(buffer, "%d", &letto);


                        printf("[INFO] - Anabbaglianti = '%d'\n", letto);

                        fflush(stdout);
```

Non era lettura tramite RS232 dalla centralina??

```
                        if(letto==1){
                                ret=TRUE;
                        }else if (letto==0){
                                ret=FALSE;
                        }
                        return ret;
                #endif
}
```

ANABBAGLIANTIHW.H

```
#ifndef AnabbagliantiHW_h
#define AnabbagliantiHW_h

#include "CentralinaComandi.h"

boolean AnabbagliantiHW_getComandoAnabbaglianti();

#endif
```

CENTRALINACOMANDI.C

```
#include "global.h"
#include "CentralinaComandi.h"
#include "CentralinaSensori.h"
#include "Display.h"
#include "LuciPosizioneHW.h"
#include "LuciAbbagliantiHW.h"
#include "FendiNebbiaAnterioriHW.h"
```

```c
#include "FendiNebbiaPosterioriHW.h"

#include "SensoreStatoLuciDirezionaliHW.h"

CentralinaComandi centralinaComandi;

STATO_SISTEMA CentralinaComandi_comandoPosizioneChiave()
{
	STATO_CHIAVE statochiave = PosizioneChiaveHW_getPosizioneChiave();
	STATO_SISTEMA ret;

	switch(statochiave){
		case I:
			ret=STANDBY;
			break;
		case O:
			ret=SPENTO;
			break;
		case A:
			ret=ACCESO;
			break;
		default:
			printf("Error, PosizioneChiaveHW_getPosizioneChiave() returned an
unexpected value '%c'\n", statochiave);
			fflush(stdout);
			break;
	}
	return ret;
}

void CentralinaComandi_init()
{
	msgstate=0;
}
```

```
void CentralinaComandi_loop()

{

        CentralinaComandi_acquisizione();

}


boolean CentralinaComandi_acquisizione()

{

        char strNumb[6];

        STATO_CHIAVE posizioneChiave = PosizioneChiaveHW_getPosizioneChiave();

        uint livelloCarburante=CentralinaSensori_sensoreLivCarburante();

        boolean statoluciposizione = LuciPosizioneHW_getComandoLuciPosizione();

        boolean statoluciabbaglianti = LuciAbbagliantiHW_getComandoLuciAbbaglianti();

        boolean statofendinebbiaanteriori =
FendiNebbiaAnterioriHW_getComandoFendinebbiaAnteriori();

    boolean statofendinebbiaposteriori =
FendiNebbiaPosterioriHW_getComandoFendinebbiaPosteriori();

    boolean statoanabbaglianti = AnabbagliantiHW_getComandoAnabbaglianti();



    STATO_LUCI_DIREZIONALI statolucidirezionali =
SensoreStatoLuciDirezionaliHW_getStatoLuciDirezionali();



        switch(posizioneChiave){

                case I:

                        if(CentralinaSensori_sensoreBatteria()==TRUE){


                                if(msgstate!=1){

                                        Display_accendiSpia(COMANDO_SPIA_BATTERIA);

                                        Display_visualizzaVelocita("000");

                                        Display_visualizzaMessaggio(VOIDS,
COMANDO_MESSAGGIO_CHECK_OK);

                                        msgstate=1;

                                }

                        }else{


                                if(msgstate!=2){
```

```
                                        Display_accendiSpia(COMANDO_SPIA_BATTERIA);
                                        Display_visualizzaMessaggio(VOIDS,
COMANDO_MESSAGGIO_AVARIA_BATTERIA);
                                        msgstate=2;
                                }
                        }

                        CentralinaComandi_attivazioneSpieLuci(statoluciposizione,
statoluciabbaglianti,
statofendinebbiaanteriori,
statofendinebbiaposteriori,
statoanabbaglianti,
statolucidirezionali);

                        break;
                case O:
                        Display_disattivaDisplay();
                        break;
                case A:
                        if(livelloCarburante>=10){


                                if(msgstate!=3){
                                        Display_spegniSpia(COMANDO_SPIA_BATTERIA);
                                        sprintf(strNumb, "%f",livelloCarburante);
                                        Display_visualizzaLivCarburante(livelloCarburante);
                                        Display_visualizzaMessaggio(VOIDS,
COMANDO_MESSAGGIO_CHECK_OK);
                                        msgstate=3;
                                }

                        }else{
```

```
                                if(msgstate!=4){
                                        Display_spegniSpia(COMANDO_SPIA_BATTERIA);
                                        Display_accendiSpia(COMANDO_SPIA_AVARIA_CARBURANTE);
                                        sprintf(strNumb, "%f",livelloCarburante);
                                        Display_visualizzaLivCarburante(livelloCarburante);
                                        Display_visualizzaMessaggio(VOIDS,
COMANDO_MESSAGGIO_AVARIA_CARBURANTE);
                                        msgstate=4;
                                };

                        }

                        CentralinaComandi_attivazioneSpieLuci(statoluciposizione,
statoluciabbaglianti,
statofendinebbiaanteriori,
statofendinebbiaposteriori,
statoanabbaglianti,
statolucidirezionali);
                        break;
                default:
                        printf("Error, PosizioneChiaveHW_getPosizioneChiave() returned an
unexpected value '%c'\n", posizioneChiave);
                        fflush(stdout);
                        return FALSE;
                        break;
        }
        return TRUE;
}

void CentralinaComandi_attivazioneSpieLuci(boolean statoluciposizione,
                                                                        boolean
statoluciabbaglianti,
                                                                        boolean
statofendinebbiaanteriori,
```

```
                                                                        boolean
statofendinebbiaposteriori,
                                                                        boolean
statoanabbaglianti,
STATO_LUCI_DIREZIONALI statolucidirezionali)
{

        if (statoluciposizione==TRUE){
                Display_accendiSpia(COMANDO_SPIA_POSIZIONE);
        }
        else {
                Display_spegniSpia(COMANDO_SPIA_POSIZIONE);
        }

        if (statoluciabbaglianti==TRUE){
                Display_accendiSpia(COMANDO_SPIA_BEAM);
        }
        else {
                Display_spegniSpia(COMANDO_SPIA_BEAM);
        }

        if (statofendinebbiaanteriori==TRUE){
                Display_accendiSpia(COMANDO_SPIA_FENDINEBBIA);
        }
        else {
                Display_spegniSpia(COMANDO_SPIA_FENDINEBBIA);
        }

        if (statofendinebbiaposteriori==TRUE){
                Display_accendiSpia(COMANDO_SPIA_FENDINEBBIA_POST);
        }
        else {
                Display_spegniSpia(COMANDO_SPIA_FENDINEBBIA_POST);
        }
```

```
        if (statoanabbaglianti==TRUE){
                Display_accendiSpia(COMANDO_SPIA_ANABBAGLIANTI);
        }
        else {
                Display_spegniSpia(COMANDO_SPIA_ANABBAGLIANTI);
        }

        switch(statolucidirezionali){
                case WAIT:
                        Display_spegniSpia(COMANDO_SPIA_DIREZIONALI);
                        break;
                case SX:
                        Display_accendiSpia(COMANDO_SPIA_DIREZIONALI);
                        break;
                case DX:
                        Display_accendiSpia(COMANDO_SPIA_DIREZIONALI);
                        break;
                default:
                        printf("Error, statolucidirezionali contains an unexpected value \n");
                        fflush(stdout);
                        break;
        }
}
```

CENTRALINACOMANDI.H

```
#ifndef CentralinaComandi_h
#define CentralinaComandi_h

#include "global.h"
#include "Controllore.h"
#include "AnabbagliantiHW.h"
```

```
#include "ComandiCuscrottoHW.h"
#include "FendiNebbiaAnterioriHW.h"
#include "FendiNebbiaPosterioriHW.h"
#include "LuciAbbagliantiHW.h"
#include "LuciPosizioneHW.h"
#include "PosizioneChiaveHW.h"
#include "SensoreStatoLuciDirezionaliHW.h"

typedef struct {
    boolean statoluciposizione;
    boolean quattroFrecce;
    STATO_SISTEMA posizioneChiave;
    STATO_LUCI_DIREZIONALI statolucidirezionali;
    boolean statoanabbaglianti;
    boolean statoluciabbaglianti;
    boolean statofendinebbiaanteriori;
    boolean statofendinebbiaposteriori;
} CentralinaComandi;

extern CentralinaComandi centralinaComandi;

int msgstate;

void CentralinaComandi_init();

void CentralinaComandi_loop();

boolean CentralinaComandi_acquisizione();

void CentralinaComandi_attivazioneSpieLuci(boolean statoluciposizione,
                                                                    boolean
statoluciabbaglianti,
                                                                    boolean
statofendinebbiaanteriori,
                                                                    boolean
statofendinebbiaposteriori,
```

```
                                                                    boolean
statoanabbaglianti,

STATO_LUCI_DIREZIONALI statolucidirezionali);

STATO_SISTEMA CentralinaComandi_comandoPosizioneChiave();

#endif
```

CENTRALINASENSORIHW.H

```
#ifndef CentralinaHW_h
#define CentralinaHW_h

#include "global.h"
#include "CentralinaSensori.h"

#endif
```

CENTRALINASENSORI.C

```
#include "CentralinaSensori.h"
#include "SensoreLivCarburanteHW.h"
#include "SensoreStatoBatteriaHW.h"
#include "Controllore.h"

CentralinaSensori centralinaSensori;

uint CentralinaSensori_sensoreLivCarburante()
{
        return SensoreLivCarburanteHW_getLivelloCarburante();
}
```

```
STATO_LIV_OLIO CentralinaSensori_sensoreLivOlio()
{
        return SensoreLivOlioHW_getLivOlio();
}

uint CentralinaSensori_sensoreTempAcqua()
{
        return SensoreTempAcquaHW_getTempAcqua();
}

boolean CentralinaSensori_sensoreBatteria()
{
        return SensoreStatoBatteriaHW_getStatoBatteria();
}

void CentralinaSensori_init()
{
        msgstate=0;
}

void CentralinaSensori_loop()
{
        CentralinaSensori_acquisizione();
}

void CentralinaSensori_acquisizione()
{
        boolean statocinture = SensoreStatoCintureHW_getStatoCinture();

        if(statocinture==FALSE){

                if(msgstate!=1){
```

```
                Display_visualizzaMessaggio(VOIDS, COMANDO_MESSAGGIO_CINTURA);
                msgstate=1;
                Display_accendiSpia(COMANDO_SPIA_CINTURA);
            }

            printf("Cintura OK - SLACCIATA\n");
            fflush(stdout);

        }else{
            if(msgstate!=2){
                Display_spegniSpia(COMANDO_SPIA_CINTURA);
                Display_visualizzaMessaggio(VOIDS, COMANDO_MESSAGGIO_CLEAR);
                msgstate=2;

            }
            printf("Cintura OK - ALLACCIATA\n");
            fflush(stdout);
        }
}

boolean CentralinaSensori_SensoreBatteria()
{

}

void CentralinaSensori_giroEffettuato(time_t time)
{
        Controllore_giroEffettuato(time);

}
```

CENTRALINASENSORI.H

```
#ifndef CentralinaSensori_h
#define CentralinaSensori_h

#include "global.h"
#include "Controllore.h"
#include "CentralinaHW.h"
#include "SensoreLivCarburanteHW.h"
#include "SensoreLivOlioHW.h"
#include "SensoreOdometroHW.h"
#include "SensoreStatoBatteriaHW.h"
#include "SensoreStatoCintureHW.h"
#include "SensoreTempAcquaHW.h"

typedef struct {
    uint livCarburante;
    uint livOlio;
    uint tempAcqua;
    boolean statoCinture;
    uint statoBatteria;
} CentralinaSensori;

extern CentralinaSensori centralinaSensori;

int msgstate;

uint CentralinaSensori_sensoreLivCarburante();

STATO_LIV_OLIO CentralinaSensori_sensoreLivOlio();

uint CentralinaSensori_sensoreTempAcqua();

boolean CentralinaSensori_sensoreBatteria();

void CentralinaSensori_init();
```

```
void CentralinaSensori_loop();

void CentralinaSensori_acquisizione();

void CentralinaSensori_giroEffettuato(time_t time);


#endif
```

COMANDOCRUSCOTTO.C

```
#ifndef ComandiCuscrottoHW_h
#define ComandiCuscrottoHW_h

#include "global.h"
#include "CentralinaComandi.h"

#endif
```

COMPUTERDIBORDO.C

```
#include "global.h"
#include "ComputerDiBordo.h"
#include "SensoreOdometroHW.h"


void ComputerDiBordo_init()
{

        //Inizializzazioni
```

```
        Display_init();
        CentralinaComandi_init();
        CentralinaSensori_init();



        Controllore_init();

}

int main(){
        Controllore_init();

        int count=0;

        while(1){
                count++;

                Sleep(10);

                if(count==100){
                        Controllore_loop();
                        count=0;
                        printf("count = 100\n");
                        fflush(stdout);
                }
                if(controllore.stato_sistema==ACCESO){
                        SensoreOdometroHW_impulso();
                }
        }
}
```

COMPUTERDIBORDO.H

```
#ifndef ComputerDiBordo_h
#define ComputerDiBordo_h

#include "global.h"
#include "Controllore.h"
#include "Display.h"
#include "Pulsantiera.h"


void ComputerDiBordo_init();


#endif
```

CONTROLLORE.C

```
#include "Controllore.h"
#include "global.h"
#include "CentralinaComandi.h"
#include "Display.h"

#define IMPULSIPERCLICK 20
#define CAPACITASERBATOIO 100
#define CHILOMETRIPERLITRO 15


Controllore controllore;
uchar pulsante;
int i = 0;

STATO_DISPLAY stato_display;
STATO_SISTEMA stato_sistema;
uint timeStamp;
```

```
uint count_viaggio;
uint count_tot;
uint livello_carburante;
STATO_LIV_OLIO liv_olio;
uint temp_acqua;

time_t prima_misurazione;
double vel_media = 0;
double const giroruota = 1.5;
double velocita,primavelocita = 0;
double diff = 0;
double tempotrascorso = 0;
double km_tot = 0;
double km_viaggio = 0;
double km_left = 0;
STATO_LIV_OLIO liv_olio;
int consumo_medio=0;
uint temp_acqua=0;
int consumo_istantaneo=0;
time_t arraygiri[100];

void Controllore_init()
{
        controllore.stato_display=INIT;
        km_left = CAPACITASERBATOIO*CHILOMETRIPERLITRO;
        msgstate=0;
}

void Controllore_loop()
{
        char strNumb[6];
        pulsante=Pulsantiera_pulsanti();

        controllore.stato_sistema = Controllore_statoSistema();
```

```
printf("Info, stato sistema = '%c'\n", controllore.stato_sistema);
fflush(stdout);



switch(controllore.stato_sistema){
                case SPENTO:
                        printf("[INFO] System down\n");
                        fflush(stdout);
                        break;
                case STANDBY:


                        Display_init();


                        printf("[INFO] System standby\n");
                        fflush(stdout);
                        CentralinaComandi_loop();
                        CentralinaSensori_loop();
                        Controllore_acquisizione();


switch(controllore.stato_display){
        case INIT:
                switch(pulsante){
                case '1':       sprintf(strNumb, "%f",km_tot);
                                Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                                controllore.stato_display = VISUALIZZA_TOT_KM;
                break;
                case '2':       sprintf(strNumb, "%d",consumo_medio);
                        Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
                                controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
                break;
                case '3':       temp_acqua=CentralinaSensori_sensoreTempAcqua();
                                sprintf(strNumb, "%d",temp_acqua);
                        Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
```

```
                controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
break;
case '4':       Controllore_azzeraStatistiche();
break;
default:
break;
}
break;
case VISUALIZZA_TOT_KM:
        switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_viaggio);
                Display_visualizzaMessaggio(VISUALIZZA_PARZ_KM, strNumb);
                        controllore.stato_display = VISUALIZZA_PARZ_KM;
        break;
        case '2':       sprintf(strNumb, "%d",consumo_medio);
        Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
        controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
        break;
        case '3':       temp_acqua=CentralinaSensori_sensoreTempAcqua();
                        sprintf(strNumb, "%d",temp_acqua);
        Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
                        controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
        break;
        case '4':       Controllore_azzeraStatistiche();
        break;
        default:
        break;
        }
        break;
        case VISUALIZZA_PARZ_KM:
        switch(pulsante){
        case '1':       sprintf(strNumb, "%f",vel_media);
                Display_visualizzaMessaggio(VISUALIZZA_VEL_MEDIA, strNumb);
                        controllore.stato_display = VISUALIZZA_VEL_MEDIA;
```

```
            break;
            case '2':       sprintf(strNumb, "%d",consumo_medio);
            Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
            controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
            break;
            case '3':       temp_acqua=CentralinaSensori_sensoreTempAcqua();
                            sprintf(strNumb, "%d",temp_acqua);
            Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
                            controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
            break;
            case '4':       Controllore_azzeraStatistiche();
            break;
            default:


            break;


            }
            break;
    case VISUALIZZA_VEL_MEDIA:
    switch(pulsante){
            case '1':       Display_visualizzaMessaggio(INIT, "");
                            controllore.stato_display = INIT;
            break;
            case '2':       sprintf(strNumb, "%d",consumo_medio);
            Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
                    controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
            break;
            case '3':       temp_acqua=CentralinaSensori_sensoreTempAcqua();
                            sprintf(strNumb, "%d",temp_acqua);
            Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
                            controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
            break;
            case '4':       Controllore_azzeraStatistiche();
            break;
```

```
        default:


        break;


        }
        break;
case VISUALIZZA_CONSUMO_MEDIO:
switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_tot);
                Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                        controllore.stato_display = VISUALIZZA_TOT_KM;
        break;
        case '2':       sprintf(strNumb, "%d",consumo_istantaneo);
        Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_ISTANTANEO,strNumb);
                controllore.stato_display = VISUALIZZA_CONSUMO_ISTANTANEO;
        break;
        case '3':       temp_acqua=CentralinaSensori_sensoreTempAcqua();
                        sprintf(strNumb, "%d",temp_acqua);
        Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
        controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
        break;
        case '4':       Controllore_azzeraStatistiche();
        break;
        default:


        break;


        }
        break;
case VISUALIZZA_CONSUMO_ISTANTANEO:
switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_tot);
                Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                        controllore.stato_display = VISUALIZZA_TOT_KM;
```

```
		break;
		case '2':	sprintf(strNumb, "%f",km_left);
		Display_visualizzaMessaggio(VISUALIZZA_STIMA_KM_RESIDUI, trNumb);
			controllore.stato_display = VISUALIZZA_STIMA_KM_RESIDUI;
		break;
		case '3':	temp_acqua=CentralinaSensori_sensoreTempAcqua();
				sprintf(strNumb, "%d",temp_acqua);
		Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
				controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
		break;
		case '4':	Controllore_azzeraStatistiche();
		break;
		default:


		break;


		}
		break;
	case VISUALIZZA_STIMA_KM_RESIDUI:
	switch(pulsante){
		case '1':	sprintf(strNumb, "%f",km_tot);
			Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
			controllore.stato_display = VISUALIZZA_TOT_KM;
		break;
		case '2':	Display_visualizzaMessaggio(INIT, "");
				controllore.stato_display = INIT;
		break;
		case '3':	temp_acqua=CentralinaSensori_sensoreTempAcqua();
				sprintf(strNumb, "%d",temp_acqua);
		Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
				controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
		break;
		case '4':	Controllore_azzeraStatistiche();
		break;
```

```
        default:


        break;


}
break;
case VISUALIZZA_TEMP_ACQUA:
switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_tot);
        Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                        controllore.stato_display = VISUALIZZA_TOT_KM;
        break;
        case '2':       sprintf(strNumb, "%d",consumo_medio);
        Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
                controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
        break;
        case '3':       liv_olio=CentralinaSensori_sensoreLivOlio();
                        DisplayHW_oilLevel(liv_olio);
                        controllore.stato_display = VISUALIZZA_LIVELLO_OLIO;
        break;
        case '4':       Controllore_azzeraStatistiche();
        break;
        default:


        break;


        }
        break;
        case VISUALIZZA_LIVELLO_OLIO:
        switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_tot);
                Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                        controllore.stato_display = VISUALIZZA_TOT_KM;
        break;
```

```
                case '2':       sprintf(strNumb, "%d",consumo_medio);
                Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
                        controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
                break;
                case '3':       Display_visualizzaMessaggio(INIT, "");
                                controllore.stato_display = INIT;
                break;
                case '4':       Controllore_azzeraStatistiche();
                break;
                default:

                break;

                }
        break;
        default:
        break;
}



break;
case ACCESO:

printf("[INFO] System ON\n");
fflush(stdout);
CentralinaComandi_loop();
CentralinaSensori_loop();
Controllore_acquisizione();

switch(controllore.stato_display){
case INIT:
        switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_tot);
                        Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
```

```
                controllore.stato_display = VISUALIZZA_TOT_KM;
                break;
case '2':       sprintf(strNumb, "%d",consumo_medio);
        Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
                controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
break;
case '3':       sprintf(strNumb, "%d",temp_acqua);
        Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
                controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
break;
case '4':       Controllore_azzeraStatistiche();
break;
default:
break;
}
break;
case VISUALIZZA_TOT_KM:
switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_viaggio);
                Display_visualizzaMessaggio(VISUALIZZA_PARZ_KM, strNumb);
                        controllore.stato_display = VISUALIZZA_PARZ_KM;
        break;
        case '2':       sprintf(strNumb, "%d",consumo_medio);
        Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
        controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
        break;
        case '3':       sprintf(strNumb, "%d",temp_acqua);
        Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
                        controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
        break;
        case '4':       Controllore_azzeraStatistiche();
        break;
        default:
```

```
        break;
}
break;
case VISUALIZZA_PARZ_KM:
        switch(pulsante){
        case '1':       sprintf(strNumb, "%f",vel_media);
        Display_visualizzaMessaggio(VISUALIZZA_VEL_MEDIA, strNumb);
                controllore.stato_display = VISUALIZZA_VEL_MEDIA;
        break;
        case '2':       sprintf(strNumb, "%d",consumo_medio);
        Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
                controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
        break;
        case '3':       sprintf(strNumb, "%d",temp_acqua);
        Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
                controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
        break;
        case '4':       Controllore_azzeraStatistiche();
        break;
        default:
        break;
        }
        break;

        case VISUALIZZA_VEL_MEDIA:
        switch(pulsante){
        case '1':       Display_visualizzaMessaggio(INIT, "");
                controllore.stato_display = INIT;
        break;
        case '2':       sprintf(strNumb, "%d",consumo_medio);
        Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
                controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
        break;
        case '3':       sprintf(strNumb, "%d",temp_acqua);
```

```
Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
                controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
        break;
        case '4':       Controllore_azzeraStatistiche();
        break;
        default:
        break;
        }
        break;
        case VISUALIZZA_CONSUMO_MEDIO:
        switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_tot);
                Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                        controllore.stato_display = VISUALIZZA_TOT_KM;
        break;
        case '2':       sprintf(strNumb, "%d",consumo_istantaneo);
Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_ISTANTANEO,strNumb);
                controllore.stato_display = VISUALIZZA_CONSUMO_ISTANTANEO;
        break;
        case '3':       sprintf(strNumb, "%d",temp_acqua);
        Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
                controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
        break;
        case '4':       Controllore_azzeraStatistiche();
        break;
        default:
        break;
        }
        break;
        case VISUALIZZA_CONSUMO_ISTANTANEO:
        switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_tot);
                Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                        controllore.stato_display = VISUALIZZA_TOT_KM;
```

```
break;
case '2':      sprintf(strNumb, "%f",km_left);
Display_visualizzaMessaggio(VISUALIZZA_STIMA_KM_RESIDUI, strNumb);
       controllore.stato_display = VISUALIZZA_STIMA_KM_RESIDUI;
break;
case '3':      sprintf(strNumb, "%d",temp_acqua);
Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
       controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
break;
case '4':      Controllore_azzeraStatistiche();
break;
default:
break;
}
break;
case VISUALIZZA_STIMA_KM_RESIDUI:
switch(pulsante){
case '1':      sprintf(strNumb, "%f",km_tot);
       Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
       controllore.stato_display = VISUALIZZA_TOT_KM;
break;
case '2':      Display_visualizzaMessaggio(INIT, "");
               controllore.stato_display = INIT;
break;
case '3':      sprintf(strNumb, "%d",temp_acqua);
Display_visualizzaMessaggio(VISUALIZZA_TEMP_ACQUA, strNumb);
       controllore.stato_display = VISUALIZZA_TEMP_ACQUA;
break;
case '4':      Controllore_azzeraStatistiche();
break;
default:
break;
}
break;
```

```
case VISUALIZZA_TEMP_ACQUA:
switch(pulsante){
        case '1':       sprintf(strNumb, "%f",km_tot);
        Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                controllore.stato_display = VISUALIZZA_TOT_KM;
        break;
        case '2':       sprintf(strNumb, "%d",consumo_medio);
Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
        controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
        break;
        case '3':       DisplayHW_oilLevel(liv_olio);
                controllore.stato_display = VISUALIZZA_LIVELLO_OLIO;
        break;
        case '4':       Controllore_azzeraStatistiche();
        break;
        default:


        break;


        }
        break;
case VISUALIZZA_LIVELLO_OLIO:
switch(pulsante){
case '1':       sprintf(strNumb, "%f",km_tot);
        Display_visualizzaMessaggio(VISUALIZZA_TOT_KM, strNumb);
                controllore.stato_display = VISUALIZZA_TOT_KM;
break;
case '2':       sprintf(strNumb, "%d",consumo_medio);
Display_visualizzaMessaggio(VISUALIZZA_CONSUMO_MEDIO,strNumb);
        controllore.stato_display = VISUALIZZA_CONSUMO_MEDIO;
break;
case '3':       Display_visualizzaMessaggio(INIT, "");
                controllore.stato_display = INIT;
break;
```

```
                                case '4':       Controllore_azzeraStatistiche();
                                                break;
                                default:
                                break;
                                }
                                break;
                                default:
                        break;
                        }

                        break;
                        default:
printf("Error, Controllore_statoSistema() returned an unexpected system state '%c'\n",
controllore.stato_sistema);
                        fflush(stdout);
                        break;

                }



}

STATO_DISPLAY Controllore_statoDisplay()
{

}

STATO_SISTEMA Controllore_statoSistema()
{
        //Controllo dello stato della chiave
        return CentralinaComandi_comandoPosizioneChiave();

}

void Controllore_giroEffettuato(time_t time)
```

```
{

        double km_temp = 0;


        arraygiri[i]=time;
        int molt = 0;
                if((i%2)==0)
                {
                        molt = IMPULSIPERCLICK+2;
                }
                else
                {
                        molt = IMPULSIPERCLICK-4;
                }


        if(i==0)
        {
                prima_misurazione = time;


        }


        if(i>0)
        {
                diff = difftime(arraygiri[i],arraygiri[i-1]);
                velocita = ((60/diff)*giroruota*molt*60)/1000;
                tempotrascorso = difftime(time,prima_misurazione);
                km_tot += (double)((velocita*1000)/3600000.0)*diff;
                km_viaggio += (double)((velocita*1000)/3600000.0)*diff;
                km_temp = km_tot-km_temp;
                if(velocita>100)
                        km_left -= (double)km_temp/(CHILOMETRIPERLITRO-5);
                else
                        km_left -= (double)km_temp/(CHILOMETRIPERLITRO+5);
        }
```

```
        if(i==2)
        {
                vel_media = (primavelocita+velocita)/2;

        }
        if(i>2)
        {
                vel_media = (vel_media+velocita)/2;

        }
        char buffer [10];
        sprintf(buffer, "%f",velocita);
        Display_visualizzaVelocita(buffer);
        i++;

}

void Controllore_acquisizione()
{
        char strNumb[6];
        uint liv_carburante = CentralinaSensori_sensoreLivCarburante();

        sprintf(strNumb, "%d",liv_carburante);
        Display_visualizzaLivCarburante(liv_carburante);



        if((liv_carburante<10)&&(liv_carburante>0))  {

                if(msgstate!=1){
                        Display_accendiSpia(COMANDO_SPIA_AVARIA_CARBURANTE);
                        Display_visualizzaMessaggio(VOIDS, COMANDO_MESSAGGIO_CARBURANTE_ESAURITO);
                        msgstate=1;
                }
        }
```

```
        if(liv_carburante==0)  {

                if(msgstate!=2){
                        Display_spegniSpia(COMANDO_SPIA_AVARIA_CARBURANTE);
                        Display_visualizzaMessaggio(VOIDS, COMANDO_MESSAGGIO_AVARIA_CARBURANTE);
                        msgstate=2;
                }
        }
}

void Controllore_azzeraStatistiche()
{
        controllore.stato_display = INIT;
}
```

CONTROLLORE.H

```
#ifndef Controllore_h
#define Controllore_h

#include "global.h"
#include "ComputerDiBordo.h"
#include "CentralinaComandi.h"
#include "CentralinaSensori.h"


typedef struct {

} Controllore;


extern Controllore controllore;
```

```
int msgstate;

void Controllore_init();

void Controllore_loop();

STATO_DISPLAY Controllore_statoDisplay();

STATO_SISTEMA Controllore_statoSistema();

void Controllore_giroEffettuato(time_t time);

void Controllore_acquisizione();

void Controllore_azzeraStatistiche();

#endif
```

DISPLAY.C

```
#include "Display.h"
#include "DisplayHW.h"

Display display;



void Display_init()
{
        DisplayHW_on();
}

void Display_accendiSpia(uchar *idSpia)
```

```
{
        DisplayHW_showLight(idSpia);
}

void Display_spegniSpia(uchar *idSpia)
{
        DisplayHW_hideLight(idSpia);
}

void Display_visualizzaMessaggio(STATO_DISPLAY tipoMsg, uchar *msg)
{
        if (tipoMsg==VOIDS){
                execute_raw_command(msg);
        }else if (tipoMsg==INIT)
                DisplayHW_cleanScreen();
        else if (tipoMsg==VISUALIZZA_TOT_KM)
                DisplayHW_kmTot(msg);
        else if (tipoMsg==VISUALIZZA_PARZ_KM)
                DisplayHW_kmLeft(msg);
        else if (tipoMsg==VISUALIZZA_VEL_MEDIA)
                DisplayHW_averageVel(msg);
        else if (tipoMsg==VISUALIZZA_CONSUMO_MEDIO)
                DisplayHW_averageCons(msg);
        else if (tipoMsg==VISUALIZZA_CONSUMO_ISTANTANEO)
                DisplayHW_instantCons(msg);
        else if (tipoMsg==VISUALIZZA_STIMA_KM_RESIDUI)
                DisplayHW_kmLeft(msg);
        else if (tipoMsg==VISUALIZZA_TEMP_ACQUA)
                DisplayHW_waterTemp(msg);
}

/**
 * Use like:
 *                      Display_visualizzaLivCarburante('1' '0' '0' ); [100]
```

```
 *                      Display_visualizzaLivCarburante('0' '5' '0' ); [50]
 *                      Display_visualizzaLivCarburante('0' '0' '0' ); [0]
 */
void Display_visualizzaLivCarburante(uchar *livello)  /**/
{
        DisplayHW_fuelLevel(livello);
}


/**
 * Use like:
 *                      Display_visualizzaVelocita('0' '5' '5' );
 */
void Display_visualizzaVelocita(uchar *velocita)
{
        DisplayHW_speedLevel(velocita);
}


void Display_disattivaDisplay()
{
        DisplayHW_off();
}


/**
 * Millisec should be 1000.
 */
void demo(int millisec)
{
        srand(time(NULL));


        Display_init();


        int i=0,j=0;
        while(i++<20){
```

```
        char buffer[3]={'0'};

        for( j = 1; j < 3; j++ ) {

                buffer[j] = '0'+(rand() % 10);
        }

        Display_visualizzaLivCarburante(buffer);

        Sleep(millisec/10);
}



i=0,j=0;
while(i++<100){

        char buffer[3]={0};

        buffer[0] = '0'+(rand() % 2);

        for( j = 1; j < 3; j++ ) {

                buffer[j] = '0'+(rand() % 10);
        }

        Display_visualizzaVelocita(buffer);

        Sleep(millisec/10);
}

Display_accendiSpia(COMANDO_SPIA_BEAM);Sleep(millisec);
Display_accendiSpia(COMANDO_SPIA_POSIZIONE);Sleep(millisec);
Display_accendiSpia(COMANDO_SPIA_FENDINEBBIA_POST);Sleep(millisec);
Display_accendiSpia(COMANDO_SPIA_FENDINEBBIA);Sleep(millisec);
```

```
        Display_accendiSpia(COMANDO_SPIA_DIREZIONALI);Sleep(millisec);
        Display_accendiSpia(COMANDO_SPIA_ANABBAGLIANTI);Sleep(millisec);

        Display_accendiSpia(COMANDO_SPIA_EMERGENZA);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_EMERGENZA);Sleep(millisec);
        Display_accendiSpia(COMANDO_SPIA_CINTURA);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_CINTURA);Sleep(millisec);
        Display_accendiSpia(COMANDO_SPIA_BATTERIA);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_BATTERIA);Sleep(millisec);
        Display_accendiSpia(COMANDO_SPIA_AVARIA_OLIO);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_AVARIA_OLIO);Sleep(millisec);
        Display_accendiSpia(COMANDO_SPIA_TEMP_ALTA);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_TEMP_ALTA);Sleep(millisec);
        Display_accendiSpia(COMANDO_SPIA_AVARIA_CARBURANTE);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_AVARIA_CARBURANTE);Sleep(millisec);
        Display_accendiSpia(COMANDO_SPIA_POSIZIONE);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_POSIZIONE);Sleep(millisec);

        Display_spegniSpia(COMANDO_SPIA_ANABBAGLIANTI);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_DIREZIONALI);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_FENDINEBBIA);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_FENDINEBBIA_POST);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_POSIZIONE);Sleep(millisec);
        Display_spegniSpia(COMANDO_SPIA_BEAM);Sleep(millisec);

        Display_disattivaDisplay();

}
```

DISPLAY.H

```
#ifndef Display_h
#define Display_h
```

```c
#include "global.h"
#include "ComputerDiBordo.h"
#include "DisplayHW.h"


typedef struct {
    STATO_DISPLAY statoDisplay;
} Display;

extern Display display;


/**
 * Will switch on the display. Take care the
 * simulator is ON
 */
void Display_init();

/**
 * It switches on a particular light.
 * PLEASE NOTE: If you want to switch on an icon with
 * a message you can use facilities in DisplayHW. Please read the
 * Display_visualizzaMessaggio() notes.
 */
void Display_accendiSpia(uchar *idSpia);

/**
 * Views a particular message.
 * PLEASE NOTE: There are some facilities to show a
 * message and an icon together. You can directly use these
 * functions in DisplayHW to avoid manual
 * synchronization. Skip the usage of this method,
 * keep it only logically.
```

```
 */

void Display_visualizzaMessaggio(STATO_DISPLAY tipoMsg, uchar *msg);

/**
 * View fuel level.
 *
 * Use like:
 *                      Display_visualizzaLivCarburante('1' '0' '0' );  [100]
 *                      Display_visualizzaLivCarburante('0' '5' '0' );   [50]
 *                      Display_visualizzaLivCarburante('0' '0' '0' );    [0]
 */
 void Display_visualizzaLivCarburante(uchar *livello);

/**
 * View speed level.
 * Use like:
 *             Display_visualizzaVelocita('0' '5' '5' );
 */
void Display_visualizzaVelocita(uchar *velocita);

/**
 * It switches off a particular light
 */
void Display_spegniSpia(uchar *idSpia);

/**
 * Disables the display
 */
void Display_disattivaDisplay();

#endif
```

DISPLAYHW.C

```
#include "DisplayHW.h"
#include <winsock2.h>
#include <stdio.h>

/**
 * Accepts commands in the following format, depending on the
 * command genre:
 *
 *              COMMAND
 *              COMMAND [0-9] [0-9] [0-9]
 *              COMMAND [0-9] [0-9] [0-9] [0-9] [0-9]
 */
int execute_raw_command(char *InputBuf){
        WSADATA wsaData;

        SOCKADDR_IN RecvAddr;

        int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
        if (iResult != NO_ERROR){
                WSACleanup();
                printf("Error while calling WSAStartup socket\n");
                printf("Type <RETURN>\n");
                fflush(stdout);
        }

        SOCKET SendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        if (SendSocket==INVALID_SOCKET)
        {
                closesocket(SendSocket);
                printf("Error while creating socket\n");
                printf("Type <RETURN>\n");
```

```
                fflush(stdout);

        }


        RecvAddr.sin_family = AF_INET;

        RecvAddr.sin_port = htons(CLIENT_MULTICAST_PORT);

        RecvAddr.sin_addr.s_addr = inet_addr(CLIENT_MULTICAST_ADDRESS);


        if (InputBuf[0]=='K' || InputBuf[0]=='C' || InputBuf[0]=='H' ){  // C, H, K

                printf("Info, sending
%c%c%c%c%c%c%c\n",InputBuf[0],InputBuf[1],InputBuf[2],InputBuf[3],InputBuf[4],InputBuf[5],InputBu
f[6]);

                fflush(stdout);

                return sendto(SendSocket, InputBuf, 7, 0,(SOCKADDR *) &RecvAddr,
sizeof(RecvAddr));



        }else{

                printf("Info, sending
%c%c%c%c\n",InputBuf[0],InputBuf[1],InputBuf[2],InputBuf[3]);

                fflush(stdout);

                return sendto(SendSocket, InputBuf, 4, 0,(SOCKADDR *) &RecvAddr,
sizeof(RecvAddr));



        }


        closesocket(SendSocket);

        printf("Type <RETURN> to exit\n");

        fflush(stdout);

        return 0;


}




/************************************************************

 * ************************************************************

 * MAPPING SCENARIO with HW FUNCTIONS

 */
```

```
int DisplayHW_checkOk(){
        return execute_raw_command(COMANDO_MESSAGGIO_CHECK_OK);
}

int DisplayHW_statCleanedOn(){
        return execute_raw_command(WARN_STAT_PULITE);
}

int DisplayHW_statCleanedOff(){
        return execute_raw_command(WARN_STAT_PULITE_OFF);
}

int DisplayHW_fastenSeatOn(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CINTURA);
        int b = DisplayHW_showLight(COMANDO_SPIA_CINTURA);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_fastenSeatOff(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CLEAR);
        int b = DisplayHW_hideLight(COMANDO_SPIA_CINTURA);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_highTempOn(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_TEMP_ALTA);
        int b = DisplayHW_showLight(COMANDO_SPIA_TEMP_ALTA);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_highTempOff(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CLEAR);
        int b = DisplayHW_hideLight(COMANDO_SPIA_TEMP_ALTA);
        return (a==-1 || b==-1)? -1 : (a+b);
```

```
}

int DisplayHW_fuelExaustedOn(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CARBURANTE_ESAURITO);
        int b = DisplayHW_showLight(COMANDO_SPIA_AVARIA_CARBURANTE);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_fuelExaustedOff(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CLEAR);
        int b = DisplayHW_hideLight(COMANDO_SPIA_AVARIA_CARBURANTE);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_oilExaustedOn(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_EMERG_OLIO);
        int b = DisplayHW_showLight(COMANDO_SPIA_AVARIA_OLIO);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_oilExaustedOff(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CLEAR);
        int b = DisplayHW_hideLight(COMANDO_SPIA_AVARIA_OLIO);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_fuelEmergencyOn(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_AVARIA_CARBURANTE);
        int b = DisplayHW_showLight(COMANDO_SPIA_AVARIA_CARBURANTE);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_fuelEmergencyOff(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CLEAR);
```

```
        int b = DisplayHW_hideLight(COMANDO_SPIA_AVARIA_CARBURANTE);
        return (a==-1 || b==-1)? -1 : (a+b);
}


int DisplayHW_oilEmergencyOn(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_EMERG_OLIO);
        int b = DisplayHW_showLight(COMANDO_SPIA_AVARIA_OLIO);
        return (a==-1 || b==-1)? -1 : (a+b);
}


int DisplayHW_oilEmergencyOff(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CLEAR);
        int b = DisplayHW_hideLight(COMANDO_SPIA_AVARIA_OLIO);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_batteryEmergencyOn(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_AVARIA_BATTERIA);
        int b = DisplayHW_showLight(COMANDO_SPIA_BATTERIA);
        return (a==-1 || b==-1)? -1 : (a+b);
}

int DisplayHW_batteryEmergencyOff(){
        int a = execute_raw_command(COMANDO_MESSAGGIO_CLEAR);
        int b = DisplayHW_hideLight(COMANDO_SPIA_BATTERIA);
        return (a==-1 || b==-1)? -1 : (a+b);
}

/**
 * level STATO_LIV_OLIO.MIN | STATO_LIV_OLIO.MED | STATO_LIV_OLIO.MED
 *
 * returns the number of byte sent, if the input is valid and the
```

```
 * command is executed successfully. Otherwise it returns -1.
 *
 */
int DisplayHW_oilLevel(STATO_LIV_OLIO level){
        if (level == MAX)
                return execute_raw_command(COMANDO_MESSAGGIO_OLIO_MAX);
        else if (level == MED)
                return execute_raw_command(COMANDO_MESSAGGIO_OLIO_MED);
        else if(level == MIN)
                return execute_raw_command(COMANDO_MESSAGGIO_OLIO_MIN);
        else return -1;
}

/**
 * level [0-9] [0-9] [0-9]
 */
int DisplayHW_speedLevel(char *level){
        char buffer[4+1] = {0};
        char *command = COMM_SPEED;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), level, 3);

        return execute_raw_command(buffer);
}
/**
 * level [0-9] [0-9] [0-9]
 */
int DisplayHW_fuelLevel(char *level){
        char buffer[4+1] = {0};
        char *command = COMM_FUEL;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), level, 3);
```

sprintf ("%c %s", CRR_FUEL, level);

```
        return execute_raw_command(buffer);
}

/**
 * level [0-9] [0-9] [0-9]
 */
int DisplayHW_averageCons(char *level){
        char buffer[4+1] = {0};
        char *command = COMM_AVERAGE_CONSUMPTION;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), level, 3);

        return execute_raw_command(buffer);
}

/**
 * level [0-9] [0-9] [0-9]
 */
int DisplayHW_instantCons(char *level){
        char buffer[4+1] = {0};
        char *command = COMM_INSTANT_CONSUMPTION;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), level, 3);

        return execute_raw_command(buffer);
}

/**
 * number [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]
 */
int DisplayHW_kmTot(char *number){
```

```
        char buffer[7+1] = {0};
        char *command = COMM_KM_TOT;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), number, 6);

        return execute_raw_command(buffer);
}

/**
 * number [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]
 */
int DisplayHW_kmTravel(char *number){
        char buffer[7+1] = {0};
        char *command = COMM_KM_TRAVEL;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), number, 6);

        return execute_raw_command(buffer);
}

/**
 * number [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]
 */
int DisplayHW_kmLeft(char *number){
        char buffer[7+1] = {0};
        char *command = COMM_KM_LEFT;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), number, 6);

        return execute_raw_command(buffer);
}
```

```
/**
 * number [0-9] [0-9] [0-9]
 */
int DisplayHW_averageVel(char *number){
        char buffer[4+1] = {0};
        char *command = COMM_AVERAGE_SPEED;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), number, 3);

        return execute_raw_command(buffer);
}

/**
 * number [0-9] [0-9] [0-9]
 */
int DisplayHW_waterTemp(char *number){
        char buffer[4+1] = {0};
        char *command = COMM_WATER_TEMP;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), number, 3);

        return execute_raw_command(buffer);
}

/**
 * Handles any light
 * command COMM_LIGHT_BEAM_ON | COMM_LIGHT_BEAM_OFF | COMM_LIGHT_*
 */
int DisplayHW_showLight(char *lightId){
        char buffer[4+1] = {0};
        char *command = COMANDO_SPIA_ON;
```

```
        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), lightId, 3);

        return execute_raw_command(buffer);
}

int DisplayHW_hideLight(char *lightId){
        char buffer[4+1] = {0};
        char *command = COMANDO_SPIA_OFF;

        memcpy(buffer, command, 1);
        memcpy(&(buffer[1]), lightId, 3);

        return execute_raw_command(buffer);
}

int DisplayHW_on(){
        char buffer[4+1] = {0, 'x', 'x', 'x', 0};
                char *command = COMANDO_DISPLAY_ON;

                memcpy(buffer, command, 1);

                return execute_raw_command(buffer);
}

int DisplayHW_off(){
        char buffer[4+1] = {0, 'x', 'x', 'x', 0};
        char *command = COMANDO_DISPLAY_OFF;

        memcpy(buffer, command, 1);

        return execute_raw_command(buffer);
}
```

```
void DisplayHW_cleanScreen(){
        DisplayHW_speedLevel("000");
        DisplayHW_hideLight(COMANDO_SPIA_EMERGENZA);
        DisplayHW_hideLight(COMANDO_SPIA_CINTURA);
        DisplayHW_hideLight(COMANDO_SPIA_BATTERIA);
        DisplayHW_hideLight(COMANDO_SPIA_AVARIA_OLIO);
        DisplayHW_hideLight(COMANDO_SPIA_TEMP_ALTA);
        DisplayHW_hideLight(COMANDO_SPIA_AVARIA_CARBURANTE);
        DisplayHW_hideLight(COMANDO_SPIA_POSIZIONE);
        DisplayHW_hideLight(COMANDO_SPIA_ANABBAGLIANTI);
        DisplayHW_hideLight(COMANDO_SPIA_DIREZIONALI);
        DisplayHW_hideLight(COMANDO_SPIA_FENDINEBBIA);
        DisplayHW_hideLight(COMANDO_SPIA_FENDINEBBIA_POST);
        DisplayHW_hideLight(COMANDO_SPIA_POSIZIONE);
        DisplayHW_hideLight(COMANDO_SPIA_BEAM);
}
```

DISPLAY.H

```
#ifndef DisplayHW_h
#define DisplayHW_h

#include "global.h"
#include "Display.h"

/************************************************************
 * ************************************************************
 * MAPPING SCENARIO with HW FUNCTIONS
 */

int DisplayHW_checkOk();
```

```
int DisplayHW_statCleanedOn();
int DisplayHW_statCleanedOff();



/**
 * Returns -1 if error occurs while sending the command,
 * that is, if one (or more) of the sub-functions fails.
 * Otherwise returns the amount of byte sent.
 */
int DisplayHW_fastenSeatOn();
int DisplayHW_fastenSeatOff();

int DisplayHW_highTempOn();
int DisplayHW_highTempOff();

int DisplayHW_fuelExaustedOn();
int DisplayHW_fuelExaustedOff();

int DisplayHW_oilExaustedOn();
int DisplayHW_oilExaustedOff();

int DisplayHW_fuelEmergencyOn();
int DisplayHW_fuelEmergencyOff();

int DisplayHW_oilEmergencyOn();
int DisplayHW_oilEmergencyOff();

int DisplayHW_batteryEmergencyOn();
int DisplayHW_batteryEmergencyOff();

/**
 * level STATO_LIV_OLIO.MIN | STATO_LIV_OLIO.MED | STATO_LIV_OLIO.MED
```

```
 *
 * returns the number of byte sent, if the input is valid and the
 * command is executed successfully. Otherwise it returns -1.
 *
 */
int DisplayHW_oilLevel(STATO_LIV_OLIO level);

/**
 * level [0-9] [0-9] [0-9]
 */
int DisplayHW_speedLevel(char *level);

/**
 * level [0-9] [0-9] [0-9]
 */
int DisplayHW_fuelLevel(char *level);

/**
 * level [0-9] [0-9] [0-9]
 */
int DisplayHW_averageCons(char *level);

/**
 * level [0-9] [0-9] [0-9]
 */
int DisplayHW_instantCons(char *level);

/**
 * number [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]
 */
int DisplayHW_kmTot(char *number);

/**
 * number [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]
```

```
 */
int DisplayHW_kmTravel(char *number);

/**
 * number [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]
 */
int DisplayHW_kmLeft(char *number);

/**
 * number [0-9] [0-9] [0-9]
 */
int DisplayHW_averageVel(char *number);

/**
 * number [0-9] [0-9] [0-9]
 */
int DisplayHW_waterTemp(char *number);



int DisplayHW_on();



int DisplayHW_off();

/**
 * Show any light
 * command COMM_LIGHT_BEAM | COMM_LIGHT_*
 */
int DisplayHW_showLight(char *command);

/**
 * Hide any light
 * command COMM_LIGHT_BEAM_ON | COMM_LIGHT_BEAM_OFF | COMM_LIGHT_*
 */
```

```
int DisplayHW_hideLight(char *command);

/**
 * Tries to clean the screen
 */
void DisplayHW_cleanScreen();

/**
 * END OF MAPPING HW FUNCTIONS
 ************************************************************/




/************************************************************
 * ************************************************************
 * LOW LEVEL FUNCTIONS
 */

/**
 * Accepts commands in the following format, depending on the
 * command genre:
 *
 *              COMMAND
 *              COMMAND [0-9] [0-9] [0-9]
 *              COMMAND [0-9] [0-9] [0-9] [0-9] [0-9]
 */
int execute_raw_command(char *InputBuf);

#define CLIENT_MULTICAST_ADDRESS             "239.255.255.250"
#define CLIENT_MULTICAST_PORT                        1600
#define CLIENT_INPUT_BUF_LEN                         1024

#define WARN_STAT_PULITE                             "E002"
```

```
#define WARN_STAT_PULITE_OFF                        "E000"

#define COMANDO_MESSAGGIO_CLEAR                     "M000"
#define COMANDO_MESSAGGIO_CHECK_OK                  "M001"
#define COMANDO_MESSAGGIO_CINTURA                   "M002"
#define COMANDO_MESSAGGIO_TEMP_ALTA                 "M010"
#define COMANDO_MESSAGGIO_AVARIA_CARBURANTE         "M016"
#define COMANDO_MESSAGGIO_CARBURANTE_ESAURITO       "M017"
#define COMANDO_MESSAGGIO_EMERG_OLIO                "M014"
#define COMANDO_MESSAGGIO_AVARIA_BATTERIA           "M015"
#define COMANDO_MESSAGGIO_OLIO_MIN                  "M013"
#define COMANDO_MESSAGGIO_OLIO_MED                  "M012"
#define COMANDO_MESSAGGIO_OLIO_MAX                  "M011"

#define COMANDO_DISPLAY_ON                          "P"
#define COMANDO_DISPLAY_OFF                         "Z"

#define COMANDO_SPIA_ON                             "A"
#define COMANDO_SPIA_OFF                            "N"


#define COMANDO_SPIA_BEAM                           "001"
#define COMANDO_SPIA_POSIZIONE                      "002"
#define COMANDO_SPIA_FENDINEBBIA_POST               "003"
#define COMANDO_SPIA_FENDINEBBIA                    "004"
#define COMANDO_SPIA_DIREZIONALI                    "005"
#define COMANDO_SPIA_EMERGENZA                      "006"
#define COMANDO_SPIA_ANABBAGLIANTI                  "007"
#define COMANDO_SPIA_CINTURA                        "008"
#define COMANDO_SPIA_BATTERIA                       "009"
#define COMANDO_SPIA_AVARIA_OLIO                    "010"
#define COMANDO_SPIA_TEMP_ALTA                      "011"
#define COMANDO_SPIA_AVARIA_CARBURANTE              "012"
```

```
#define COMM_FUEL                                       "F" /* F0xx */
#define COMM_SPEED                                      "T" /* Txxx */
#define COMM_AVERAGE_CONSUMPTION                        "J" /* Jxxx */
#define COMM_INSTANT_CONSUMPTION                        "W" /* Wxxx */

#define COMM_KM_TOT                                     "K" /* Kxxxxxx */
#define COMM_KM_TRAVEL                                  "C" /* Cxxxxxx */
#define COMM_KM_LEFT                                    "H" /* Hxxxxxx */

#define COMM_AVERAGE_SPEED                              "V" /* Vxxx */
#define COMM_WATER_TEMP                                 "Q" /* Qxxx */

/**
 * END OF LOW LEVEL FUNCTIONS
 ************************************************************/

#endif
```

FENDINEBBIAANTERIORIHW.C

```
#include "FendiNebbiaAnterioriHW.h"


boolean FendiNebbiaAnterioriHW_getComandoFendinebbiaAnteriori()
{
        #ifdef MICRO//caso HW
        //legge il valore all'indirizzo AD_LUCIFENDINEBBIAANT_GETT (0x09)
        uchar val;
        val=*AD_LEGGI_PULS;
        return val;
        #endif
```

```
        #ifdef PC
        char buffer[BUFMAX]={0};
        FILE *fenidinebbiaanteriori;

        uint letto = 0;

        if (!(fenidinebbiaanteriori=fopen("./interfacce/FendiNebbiaAnteriori.txt", "rw" ))){
                printf("File not found - interfacce/FendiNebbiaAnteriori.txt\n");
                fflush(stdout);
                return 0;
        }

        fgets(buffer, BUFMAX, fenidinebbiaanteriori);
        fclose(fenidinebbiaanteriori);

        sscanf(buffer, "%d", &letto);

        printf("[INFO] - Fendinebbia Anteriori = '%d'\n", letto);
        fflush(stdout);

        if(letto==1){
                return TRUE;
        }else if (letto==0){
                return FALSE;
        }
        #endif
}
```

FENDINEBBIANTERIORI.H

```
#ifndef FendiNebbiaAnterioriHW_h
#define FendiNebbiaAnterioriHW_h
```

```
#include "global.h"
#include "CentralinaComandi.h"

boolean FendiNebbiaAnterioriHW_getComandoFendinebbiaAnteriori();

#endif
```

FENDINEBBIAPOSTERIORIHW.C

```
#include "FendiNebbiaPosterioriHW.h"


boolean FendiNebbiaPosterioriHW_getComandoFendinebbiaPosteriori()
{
	#ifdef MICRO//caso HW
	//legge il valore all'indirizzo AD_LUCIFENDINEBBIAPOST_GETT (0x10)
	uchar val;
	val=*AD_LEGGI_PULS;
	return val;
	#endif

	#ifdef PC
	char buffer[BUFMAX]={0};
	FILE *fenidinebbiaposteriori;

	uint letto = 0;

	if (!(fenidinebbiaposteriori=fopen("./interfacce/FendiNebbiaPosteriori.txt", "rw" ))){
		printf("File not found - interfacce/FendiNebbiaPosteriori.txt\n");
		fflush(stdout);
		return 0;
	}
```

```
        fgets(buffer, BUFMAX, fenidinebbiaposteriori);
        fclose(fenidinebbiaposteriori);

        sscanf(buffer, "%d", &letto);

        printf("[INFO] - Fendinebbia Posteriori = '%d'\n", letto);
        fflush(stdout);

        if(letto==1){
                return TRUE;
        }else if (letto==0){
                return FALSE;
        }
        #endif
}
```

FENDINEBBIAPOSTERIORIHW.C

```
#ifndef FendiNebbiaPosterioriHW_h
#define FendiNebbiaPosterioriHW_h

#include "global.h"
#include "CentralinaComandi.h"

boolean FendiNebbiaPosterioriHW_getComandoFendinebbiaPosteriori();

#endif
```

GLOBAL.H

```
#ifndef global_h
#define global_h
```

```
#define AD_LEGGI_PULS 0x03
#define AD_ACCENSIONE_GETT 0x04
#define AD_TEMPACQUA_GETT 0x05
#define AD_LIVOLIO_GETT 0x06
#define AD_ODOMETRO_SCRIVI 0x07
#define AD_LIVCARBURANTE_GETT 0x08
#define AD_LUCIFENDINEBBIAANT_GETT 0x09
#define AD_LUCIFENDINEBBIAPOST_GETT 0x10
#define AD_LUCIANABBAGLIANTI_GETT 0x11
#define AD_LUCIABBAGLIANTI_GETT 0x12
#define AD_LUCIDIREZIONALI_GETT 0x14
#define AD_STATOCINTURE_GETT 0x15
#define AD_STATOBATTERIA_GETT 0x16
#define AD_LUCIPOSIZIONE_GETT 0x17



#include <windows.h>
#include <stdio.h>
#include <ctype.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <conio.h>

#include <time.h>

typedef unsigned char uchar;

typedef unsigned int uint;
```

```
typedef enum {
    INIT,
    VISUALIZZA_TOT_KM,
    VISUALIZZA_PARZ_KM,
    VISUALIZZA_VEL_MEDIA,
    VISUALIZZA_CONSUMO_MEDIO,
    VISUALIZZA_CONSUMO_ISTANTANEO,
    VISUALIZZA_STIMA_KM_RESIDUI,
    VISUALIZZA_TEMP_ACQUA,
    VISUALIZZA_LIVELLO_OLIO,
    INIT_AVARIA,
    VOIDS
} STATO_DISPLAY;

typedef enum {
    SPENTO,
    STANDBY,
    ACCESO
} STATO_SISTEMA;

typedef enum {
    WAIT,
    SX,
    DX
} STATO_LUCI_DIREZIONALI;

typedef enum {
    I,
    A,
    O
} STATO_CHIAVE;



typedef enum {
```

```
    MAX,
    MED,
    MIN
} STATO_LIV_OLIO;


#endif
```

LUCIABBAGLIANTIHW.C

```
#include "LuciAbbagliantiHW.h"


boolean LuciAbbagliantiHW_getComandoLuciAbbaglianti()
{
        #ifdef MICRO//caso HW
        //legge il valore all'indirizzo AD_LUCIABBAGLIANTI_GETT(0x12)
        uchar val;
        val=*AD_LEGGI_PULS;
        return val;
        #endif

        #ifdef PC
        char buffer[BUFMAX]={0};
        FILE *lucipabbaglianti;

        uint letto = 0;

        if (!(lucipabbaglianti=fopen("./interfacce/LuciAbbaglianti.txt", "rw" ))){
                printf("File not found - interfacce/LuciAbbaglianti.txt\n");
                fflush(stdout);
                return 0;
        }
```

```
        fgets(buffer, BUFMAX, lucipabbaglianti);
        fclose(lucipabbaglianti);

        sscanf(buffer, "%d", &letto);

        printf("[INFO] - Luci abbaglianti = '%d'\n", letto);
        fflush(stdout);

        if(letto==1){
                return TRUE;
        }else if (letto==0){
                return FALSE;
        }
        #endif
}
```

LUCIABBAGLIANTIHW.H

```
#ifndef LuciAbbagliantiHW_h
#define LuciAbbagliantiHW_h

#include "global.h"
#include "CentralinaComandi.h"

boolean LuciAbbagliantiHW_getComandoLuciAbbaglianti();

#endif
```

LUCIPOSIZIONEHW.C

```
#include "LuciPosizioneHW.h"
```

```
boolean LuciPosizioneHW_getComandoLuciPosizione()
{
	#ifdef MICRO//caso HW
	//legge il valore all'indirizzo AD_LUCIPOSIZIONE_GETT (0x17)
	uchar val;
	val=*AD_LEGGI_PULS;
	return val;
	#endif

	#ifdef PC
	char buffer[BUFMAX]={0};
	FILE *luciposizione;

	uint letto = 0;

	if (!(luciposizione=fopen("./interfacce/LuciPosizione.txt", "rw" ))){
		printf("File not found - interfacce/LuciPosizione.txt\n");
		fflush(stdout);
		return 0;
	}

	fgets(buffer, BUFMAX, luciposizione);
	fclose(luciposizione);

	sscanf(buffer, "%d", &letto);

	printf("[INFO] - Luci posizione = '%d'\n", letto);
	fflush(stdout);

	if(letto==1){
		return TRUE;
	}else if (letto==0){
```

```
                return FALSE;
        }
        #endif
}
```

LUCIPOSIZIONEHW.H

```
#ifndef LuciPosizioneHW_h
#define LuciPosizioneHW_h

#include "global.h"
#include "CentralinaComandi.h"

boolean LuciPosizioneHW_getComandoLuciPosizione();

#endif
```

POSIZIONECHIAVEHW.C

```
#include "global.h"
#include "PosizioneChiaveHW.h"

STATO_CHIAVE PosizioneChiaveHW_getPosizioneChiave()
{
        #ifdef MICRO//caso HW
        //legge il valore all'indirizzo AD_ACCENSIONE_GETT(0x04)
        uchar val;
        val=*AD_LEGGI_PULS;
        return val;
        #endif

        #ifdef PC
```

```
char buffer[BUFMAX]={0};
FILE *chiave;

char letto = 0;

if (!(chiave=fopen("./interfacce/PosizioneChiave.txt", "rw" ))){
        printf("File not found - interfacce/PosizioneChiave.txt\n");
        fflush(stdout);
        return 0;
}

fgets(buffer, BUFMAX, chiave);
fclose(chiave);

sscanf(buffer, "%c", &letto);

printf("[INFO] - Stato chiave = '%c'\n", letto);
fflush(stdout);

switch(letto){
        case 'I':
                return I;
                break;
        case 'O':
                return 0;
                break;
        case 'A':
                return A;
                break;
        default:
                printf("Error, file contains an unexpected value '%c'\n", letto);
                fflush(stdout);
                break;
}
```

```
        #endif

}
```

POSIZIONECHIAVEHW.H

```
#ifndef PosizioneChiaveHW_h
#define PosizioneChiaveHW_h

#include "global.h"
#include "CentralinaComandi.h"

#define BUFMAX 255

STATO_CHIAVE PosizioneChiaveHW_getPosizioneChiave();

void PosizioneChiaveHW_getComandoChiave();

#endif
```

PULSANTIERA.C

```
#include "Pulsantiera.h"


uchar Pulsantiera_pulsanti()
{
        return PulsantieraHW_leggi();
}
```

PULSANTIERA.H

```
#ifndef Pulsantiera_h
#define Pulsantiera_h

#define Pulsante1 '1'
#define Pulsante2 '2'
#define Pulsante3 '3'
#define Pulsante4 '4'


#include "global.h"
#include "ComputerDiBordo.h"
#include "PulsantieraHW.h"


uchar Pulsantiera_pulsanti();


#endif
```

PULSANTIERAHW.C

```
#include "PulsantieraHW.h"


PulsantieraHW pulsantierahw;


uchar PulsantieraHW_leggi()
```

```
{
	#ifdef MICRO//caso HW
	//legge il valore all'indirizzo AD_LEGGI_PULS (0x03)
	uchar val;
	val=*AD_LEGGI_PULS;
	return val;
	#endif

	#ifdef PC
	char msg [1024];

	//simulo la pulsantiera con la tastiera
	Sleep(200);
	if(kbhit())
	{

	char buf = getch();
	if (((buf=='1')|| (buf=='2') || (buf == '3') || (buf == '4')))
	{
		return (uchar) buf;
	}
	else
		return 0;
	}
	#endif
}
```

PULSANTIERAHW.H

```
#ifndef PosizioneChiaveHW_h
#define PosizioneChiaveHW_h

#include "global.h"
```

```
#include "CentralinaComandi.h"

#define BUFMAX 255

STATO_CHIAVE PosizioneChiaveHW_getPosizioneChiave();

void PosizioneChiaveHW_getComandoChiave();

#endif
```

SENSORELIVCARBURANTEHW.C

```
#include "SensoreLivCarburanteHW.h"


uint SensoreLivCarburanteHW_getLivelloCarburante()
{
        #ifdef MICRO//caso HW
        //legge il valore all'indirizzo AD_LIVCARBURANTE_GETT(0x08)
        uchar val;
        val=*AD_LEGGI_PULS;
        return val;
        #endif

        #ifdef PC
        char buffer[BUFMAX]={0};
        FILE *carburante;

        uint letto = 0;

        if (!(carburante=fopen("./interfacce/LivCarburante.txt", "rw" ))){
                printf("File not found - interfacce/LivCarburante.txt\n");
                fflush(stdout);
```

```
                return 0;
        }

        fgets(buffer, BUFMAX, carburante);
        fclose(carburante);

        sscanf(buffer, "%d", &letto);

        printf("[INFO] - Livello Carburante = '%d'\n", letto);
        fflush(stdout);

        return letto;
        #endif
}
```

SENSORELIVCARBURANTEHW.H

```
#ifndef SensoreLivCarburanteHW_h
#define SensoreLivCarburanteHW_h

#include "global.h"
#include "CentralinaSensori.h"

uint SensoreLivCarburanteHW_getLivelloCarburante();

#endif
```

SENSORELIVOLIOHW.C

```
#include "SensoreLivOlioHW.h"
```

```
STATO_LIV_OLIO SensoreLivOlioHW_getLivOlio()
{
		#ifdef MICRO//caso HW
		//legge il valore all'indirizzo AD_LIVOLIO_GETT(0x06)
		uchar val;
		val=*AD_LEGGI_PULS;
		return val;
		#endif

		#ifdef PC

		STATO_LIV_OLIO ret;
		char buffer[BUFMAX]={0};
		FILE *livOlio;

		uint letto = 0;

		if (!(livOlio=fopen("./interfacce/LivOlio.txt", "rw" ))){
			printf("File not found - interfacce/LivOlio.txt\n");
			fflush(stdout);
			return 0;
		}

		fgets(buffer, BUFMAX, livOlio);
		fclose(livOlio);

		sscanf(buffer, "%d", &letto);

		printf("[INFO] - Livello Olio = '%d'\n", letto);
		fflush(stdout);

		if(letto==0){
				ret=MIN;
			}else if (letto==1){
```

```
                        ret=MED;
                }else if (letto==2){
                        ret=MAX;
                }
        return ret;
        #endif
}
```

SENSORELIVOLIOHW.H

```
#ifndef SensoreLivOlioHW_h
#define SensoreLivOlioHW_h

#include "global.h"
#include "CentralinaSensori.h"

STATO_LIV_OLIO SensoreLivOlioHW_getLivOlio();

#endif
```

SENSOREODOMETROHW.C

```
#include "SensoreOdometroHW.h"
#include "CentralinaSensori.h"
#include "global.h"

void SensoreOdometroHW_impulso(){

        #ifdef MICRO//caso HW
        //legge il valore all'indirizzo AD_ODOMETRO_SCRIVI(0x07)
```

```
                uchar val;
                val=*AD_LEGGI_PULS;
                return val;
                #endif

                #ifdef PC

        int ch = 0;
        if((ch = kbhit()) == 1)
        {

                getch();
                time_t o;
                time_t t = time(&o);
                printf ("%s\n",ctime(&t));
                CentralinaSensori_giroEffettuato(t);



        }
                #endif

}
```

SENSOREODOMETROHW.H

```
#ifndef SensoreOdometroHW_h
#define SensoreOdometroHW_h

#include "global.h"
#include "CentralinaSensori.h"

void SensoreOdometroHW_impulso();
```

```
#endif
```

SENSORESTATOBATTERIAHW.C

```
#include "SensoreStatoBatteriaHW.h"



boolean SensoreStatoBatteriaHW_getStatoBatteria()
{

        #ifdef MICRO//caso HW
        //legge il valore all'indirizzo AD_STATOBATTERIA_GETT(0x16)
        uchar val;
        val=*AD_LEGGI_PULS;
        return val;
        #endif


        #ifdef PC


        char buffer[BUFMAX]={0};
        FILE *batteria;


        uint letto = 0;


        if (!(batteria=fopen("./interfacce/StatoBatteria.txt", "rw" ))){
                printf("File not found - interfacce/StatoBatteria.txt\n");
                fflush(stdout);
                return 0;
        }


        fgets(buffer, BUFMAX, batteria);
        fclose(batteria);
```

```
        sscanf(buffer, "%d", &letto);


        printf("[INFO] - Stato Batteria = '%d'\n", letto);
        fflush(stdout);


        if(letto==1){
                return TRUE;
        }else if (letto==0){
                return FALSE;
        }
        #endif
}
```

SENSORESTATOBATTERIAHW.H

```
#ifndef SensoreStatoBatteriaHW_h
#define SensoreStatoBatteriaHW_h

#include "global.h"
#include "CentralinaSensori.h"



boolean SensoreStatoBatteriaHW_getStatoBatteria();



#endif
```

SENSORESTATOCINTUREHW.C

```
#include "SensoreStatoCintureHW.h"
```

```
boolean SensoreStatoCintureHW_getStatoCinture()
{
        #ifdef MICRO//caso HW
        //legge il valore all'indirizzo AD_STATOCINTURE_GETT(0x15)
        uchar val;
        val=*AD_LEGGI_PULS;
        return val;
        #endif

        #ifdef PC

        char buffer[BUFMAX]={0};
        FILE *batteria;

        uint letto = 0;

        if (!(batteria=fopen("./interfacce/StatoCinture.txt", "rw" ))){
                printf("File not found - interfacce/StatoCinture.txt\n");
                fflush(stdout);
                return 0;
        }

        fgets(buffer, BUFMAX, batteria);
        fclose(batteria);

        sscanf(buffer, "%d", &letto);

        printf("[INFO] - Stato Cinture = '%d'\n", letto);
        fflush(stdout);

        if(letto==1){
                return TRUE;
        }else if (letto==0){
                return FALSE;
```

```
        }
        #endif
}
```

SENSORESTATOCINTUREHW.H

```
#ifndef SensoreStatoCintureHW_h
#define SensoreStatoCintureHW_h

#include "global.h"
#include "CentralinaSensori.h"

boolean SensoreStatoCintureHW_getStatoCinture();

#endif
```

SENSORESTATOLUCIDIREZIONALIHW.C

```
#include "SensoreStatoLuciDirezionaliHW.h"

STATO_LUCI_DIREZIONALI SensoreStatoLuciDirezionaliHW_getStatoLuciDirezionali()
{
        #ifdef MICRO//caso HW
        //legge il valore all'indirizzo AD_LUCIDIREZIONALI_GETT(0x12)
        uchar val;
        val=*AD_LEGGI_PULS;
        return val;
        #endif
```

```
        #ifdef PC
        char buffer[BUFMAX]={0};
        FILE *anabbaglianti;

        uint letto = 0;

        if (!(anabbaglianti=fopen("./interfacce/LuciDirezionali.txt", "rw" ))){
                printf("File not found - interfacce/LuciDirezionali.txt\n");
                fflush(stdout);
                return 0;
        }

        fgets(buffer, BUFMAX, anabbaglianti);
        fclose(anabbaglianti);

        sscanf(buffer, "%d", &letto);

        printf("[INFO] - Luci Direzionali = '%d'\n", letto);
        fflush(stdout);

        if(letto==0){
                return WAIT;
        }else if (letto==1){
                return SX;
        }else if (letto==2){
                return DX;
        }
        #endif
}
```

SENSORESTATOLUCIDIREZIONALIHW.H

```
#ifndef SensoreStatoLuciDirezionaliHW_h
```

```
#define SensoreStatoLuciDirezionaliHW_h


#include "global.h"
#include "CentralinaComandi.h"


STATO_LUCI_DIREZIONALI SensoreStatoLuciDirezionaliHW_getStatoLuciDirezionali();


#endif
```

SENSORETEMPACQUAHW.C

```
#include "SensoreTempAcquaHW.h"



uint SensoreTempAcquaHW_getTempAcqua()
{
		#ifdef MICRO//caso HW
		//legge il valore all'indirizzo AD_TEMPACQUA_GETT(0x05)
		uchar val;
		val=*AD_LEGGI_PULS;
		return val;
		#endif

		#ifdef PC
		char buffer[BUFMAX]={0};
		FILE *tempacqua;

		uint letto = 0;

		if (!(tempacqua=fopen("./interfacce/TempAcqua.txt", "rw" ))){
			printf("File not found - interfacce/TempAcqua.txt\n");
			fflush(stdout);
			return 0;
		}
```

```
                fgets(buffer, BUFMAX, tempacqua);
                fclose(tempacqua);

                sscanf(buffer, "%d", &letto);

                printf("[INFO] - Temperatura Acqua = '%d'\n", letto);
                fflush(stdout);

                return letto;
                #endif

}
```

SENSORETEMPACQUAHW.H

```
#ifndef SensoreTempAcquaHW_h
#define SensoreTempAcquaHW_h

#include "global.h"
#include "CentralinaSensori.h"

uint SensoreTempAcquaHW_getTempAcqua();

#endif
```

MACANO LE SIMULAZIONI

il codice non coincide con il progetto (o almeno con quel poco che si capisce)