5.PROGRAMACION DE LA ECU

La ECU ha sido programada en lenguaje C en el entorno de programacion QNX Momentics Integrated Development Environment (IDE), que añade soporte para el sistema operativo en tiempo real QNX, tal y como se menciona en el capítulo 4.2.

En base a todo lo mencionado en el anterior capítulo, y sobre todo en el apartado de diseño (4.5), se ha desarrollado toda la programación de la Unidad de Control Electrónico para lograr un sistema estable, fiable, y autónomo capaz de supervisar el funcionamiento de los sistemas, actuando sobre ellos, y gestionando el uso de la potencia en el vehículo.

5.1. Jerarquía de programación

La programación se ha desarrollado de la manera más **modulada** posible, facilitando la depuración del sistema y el funcionamiento paralelo de diferentes procesos necesarios en el sistema (**programación concurrente**).

Teniendo en cuenta que la ECU tiene que realizar varias tareas de forma simultánea se han definido distintos procesos capaces de manejar todo de forma conjunta, comunicándose la información por medio de variables globales (de las que se hablarán en el siguiente apartado) y colas de mensajes. Los procesos y sus tareas son los siguientes:

- **Proceso principal**: inicializa el sistema realizando las primeras comprobaciones posteriores al encendido, y se encarga del bucle principal de funcionamiento de la ECU, manejando las siguientes tareas de forma secuencial:
  - **Cálculo de parámetros**: así se resuelve el estado de carga de las baterías, la velocidad lineal del vehículo o la carga de hidrógeno que resta en los depósitos.
  - **Control de las secuencias**: tal y como se describe en el apartado 4.5.3., para realizar la parada, el arranque o la parada de emergencia de los componentes del Leon.
  - **Gestión de la potencia**: se calculan constantemente las consignas de potencia en base al pedal del acelerador y los algoritmos de gestión de potencia.
- **Proceso de errores**: se encarga de comprobar continuamente si existen errores y qué tipo de actuación es necesaria realizar en cada momento.
- **Proceso de traducción de señales**: toma la informacion que llega a la ECU desde las señales analógicas y digitales, calculando datos importantes referentes al estado de los depósitos de hidrógeno y el pedal del acelerador, y enviando también señales digitales para el control de las válvulas de los depósitos.
- **Proceso de CAN**: maneja la tarjeta de bus CAN enviando y recibiendo mensajes, decidiendo cuándo y cómo, y adquiriendo los datos que llegan a través de las tramas CAN.
- **Proceso de adquisión de datos**: maneja la tarjeta de adquisión de datos, enviando y recibiendo la información pertinente por medio de las entradas y salidas analógicas y digitales, y comunicándolas con el proceso de traducción de señales por medio de colas de mensajes. Este proceso es igual al utilizado en el Sistema de Supervisión, y ha sido creado por otro miembro de AICIA-Automática, por lo que no ha sido incluido en este proyecto.

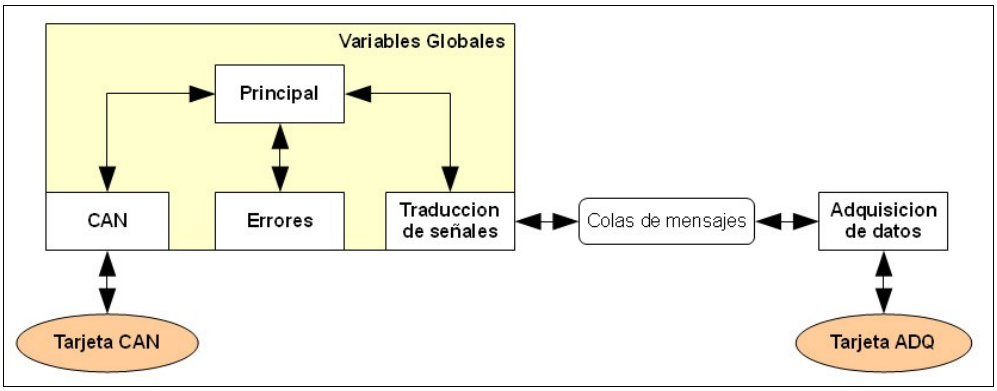

Imagen 63: jerarquía de procesos

Como se ve en la imagen 64, el proceso de adquisición de datos no pertenece a la jerarquía del proceso principal, en la cual, el de CAN, el de Errores y el de Traducción de Señales, dependerán directamente del Principal: éstos serán hilos distintos dentro del proceso principal, siguiendo los estándares POSIX.

Entre los procesos se pasarán la información de dos formas diferentes:

- **Variables globales**: listas de variables a las que tendrán acceso los procesos para intercambiarse información entre ellos (apartado 5.2.)
- **Colas de mensajes**: servirá para intercambiar datos de forma más particular entre dos procesos diferentes, siguiendo los estándares POSIX (apartado 5.4.1.)

5.2. Variables globales

Las variables globales utilizadas están organizadas en **estructuras** para facilitar su manejo y las tareas de exclusión mutua en los accesos a dichas variables, ya que habrá que evitar conflictos durante la programación concurrente, puesto que manejaremos hasta cuatro hilos distintos dentro del proceso principal y muchos parámetros.

Los tipos de variables son, casi todas, short y unsigned short ya que así tenemos palabras de 16 bits, que serán fáciles de manejar para realizar conversiones a las palabras de 8 bits necesarias para los mensajes CAN.

Para las tareas de exclusión mutua se han utilizado "**mutex**" en cada uno de los usos de estas estructuras. Estos mutex se han inicializado en el proceso principal y se han declarado externos para que todos los hilos pudiesen usarlos. Se ha creado un mutex por estructura, con su mismo nombre precedido por el prefijo "**mut_**".

Las **estructuras** creadas incluyen variables de control y datos de mensajes CAN y de adquisición, una para cada uno de los siguientes apartados del León:

- **Potencias** (*potencias*): incluye todas las consignas referentes a la gestión de potencia, y todas manejan unidades de 10 W, para evitar la conversión previa y posterior al envío de consignas, ya que éstas también manejan 10 W.
- **Convertidores** (*convertidores*): las variables de control incluyen flags para avisar del envío de mensajes, o de posibles errores como el de falta de Watchdog.
- **Pila de Combustible** (*fuelcell*): las variables de control incluyen flags para avisar del envío de mensajes concretos, o de posibles errores como el de falta de Watchdog.
- **Baterías** (*baterias*): incluye toda la información referente al estado de carga.
- **Variador y Motor** (*vmotor*): las variables de control incluyen flags para avisar del envío de ciertos mensajes, o de posibles errores.
- **Vehículo** (*coche*): incluye parámetros genéricos del coche y del motor, como velocidades, el par, etc. Además incorpora las variables referidas a la llave de arranque del mismo.
- **Hidrógeno** (*hidrogeno*): incluye todas los parámetros medidos de los depósitos de hidrógeno y de los sensores para detectar fugas.
- **Errores y alarmas** (*errores*): incluye todas los flags de alarmas y los códigos de errores: leves, graves y críticos.
- **Colas de mensajes** (*dig_rd, dig_wr, analog_in, analog_out*): son **4 estructuras** que se utilizarán para adquirir los datos de las colas de mensajes que se intercambiarán con el proceso de manejo de la tarjeta de adquisición de datos.

5.3. Proceso principal

El proceso principal (**main**) carga con los cálculos y las gestiones más importantes de toda la ECU: el control de los estados de funcionamiento, secuencias de cambios de estado, gestiones, calculos de variables, etc, así como de la inicialización y cierre de toda la programación de la misma.

Las primeras **fases** de proceso son:

- Externamente al programa se crean las estructuras de las **variables globales** que luego se enlazarán externamente desde los otros procesos. También se inicializan los **mutex** para el control de la exclusión mutua entre dichas variables.
- Ya dentro del programa se **inicializan las variables globales**, todas al valor cero.
- Se **crean** los hilos que dependerán de este proceso:
  - **thr_can**: hilo de de CAN.
  - **thr_adq**: hilo de traducción de señales de la adquisición de datos.
  - **thr_errores**: hilo de errores.
- Se llama a la función **check_comm**, cuyo objetivo se detalla más adelante.
- Se fjian las variables que controlarán las **válvulas** de los depósitos de hidrógeno, para mantenerlas cerradas.
- Se activa la **frenada regenerativa**.
- Se inicia el **bucle principal** de funcionamiento.
- Si se termina el bucle anterior, se **destruyen** los hilos creados y **se cierra** el programa.

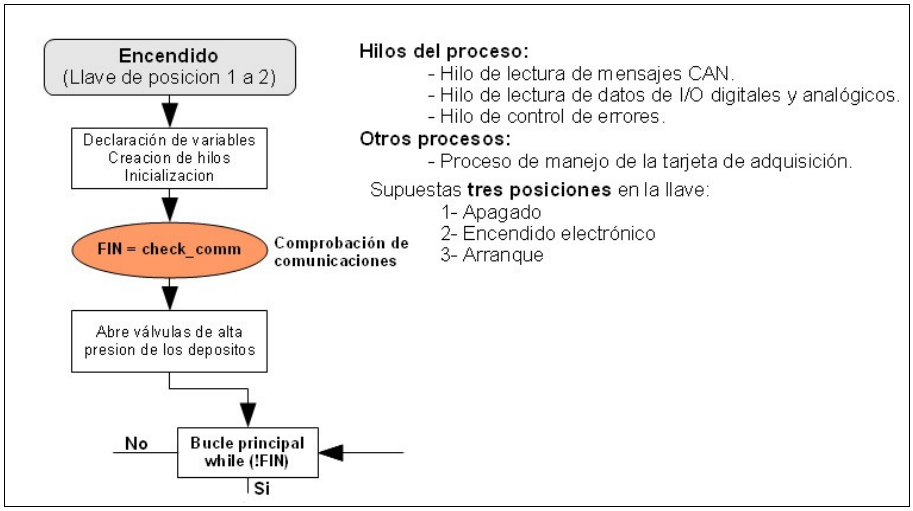

Imagen 64: diagrama previo al bucle principal

Antes de ver cómo opera el bucle principal, que es el centro de funcionamiento de toda la ECU, vamos a ver todas las funciones que son llamadas desde el proceso principal, para luego poder entender mejor dicho bucle.

5.3.1. Función check_comm

Esta función comprueba que no haya errores de comunicaciones de ningún tipo, ni de encendido de las tarjetas de comunicaciones (CAN o adquisicion). Para ello lo primero que hace es esperar 2 segundos para dar tiempo al resto de hilos para que funcionen durante el tiempo suficiente para comprobar si hay o no errores de cualquier clase. Una vez hecho ésto, se comprueba una variable que almacena TRUE o FALSE si ha habido o no falta de mensajes de Watchdog (*errores.er_watchdog*). Esto nos permitirá saber si todos los componentes están operando y están comunicando, además de que nos permitirá conocer si nuestra tarjeta CAN está funcionando. Para terminar se comprueba una variable que nos envía el proceso de manejo de la tarjeta de adquisición para saber si está operativa (*dig_rd.adq_error*).

En el caso de que haya errores de Watchdog o la tarjeta de ADQ no esté funcionando, se devuelve un valor que avisará al proceso para no arrancar el bucle principal de la ECU.

5.3.2. Función calculo_velocidad

Este módulo se encarga de calcular la velocidad lineal del vehículo en base a la velocidad angular del motor y los datos físicos del coche, siguiendo la siguiente fórmula:

$$V_{lineal} = V_{angular} \cdot (Radio_{rueda} / Coeficiente_{transmision})$$

Sabiendo que según las especificaciones del motor, una velocidad lineal de 150 km/h se corresponde con aproximadamente 6000 rpm, el coeficiente de transmisicón es de 5,27. El radio del neumático es de 0,35 metros.

5.3.3. Función calculo_H2

Esta función calcula la carga de hidrógeno en los depósitos y en el total, en %, según las medidas de los sensores de alta presión en los mismos. Para ello considera el 100% en la máxima medida que pueden dar dichos sensores (448 bares).

5.3.4. Función calculo_bateria

Calcula el estado de carga de las baterías siguiendo el método establecido por GreenPower para evitar discrepancias entre las medidas realizadas en el control del convertidor de las baterías y en la ECU. Dicho método se ciñe a medir el nivel de tensión en las mismas, y en base a él, calcula el nivel de carga que resta.

El nivel máximo de tensión en las baterías es de 153,6 V y el mínimo es de 120 V, aunque se deben mantener las baterías funcionando siempre entre el 5% y el 90%.

5.3.5. Función secuencia_arranque

Esta función realiza todo el proceso de arranque de los dispositivos de potencia del Leon, es decir, la pila, los convertidores, y el variador del motor. Para ello envía las órdenes pertinentes de cambio de estado mediante mensajes CAN, comprobando que dichos estados se alcanzan sin errores, y siguiendo los diagramas de estado de cada uno de ellos.

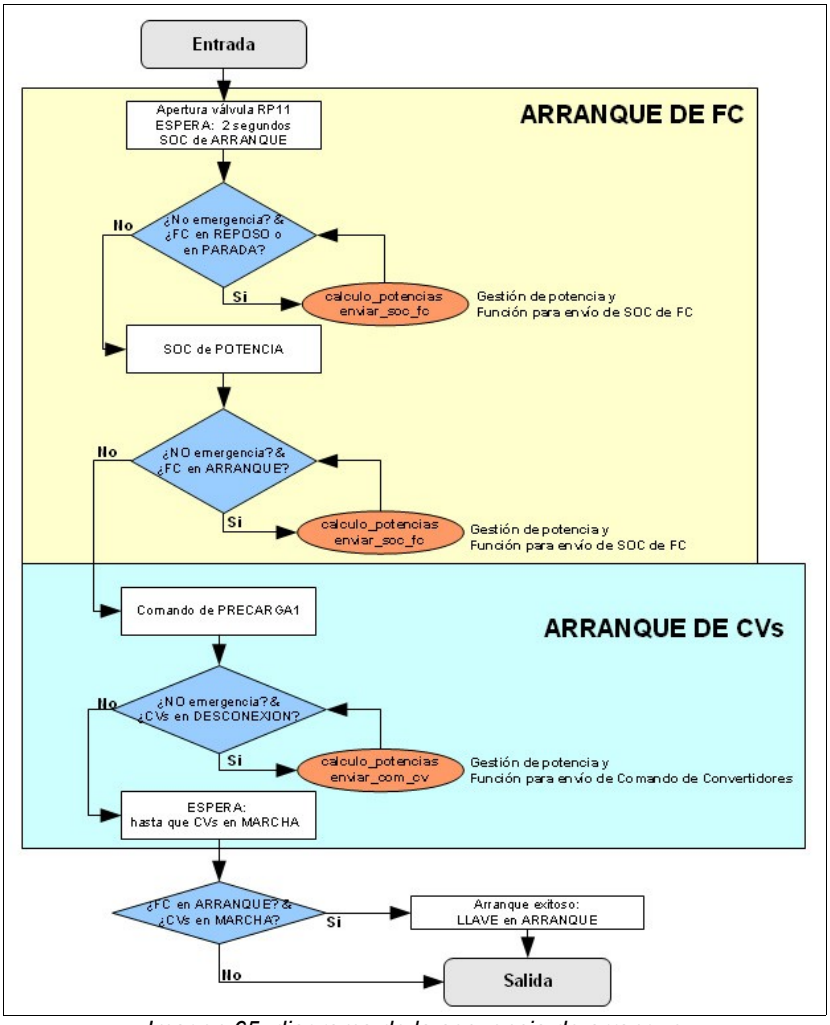

Imagen 65: diagrama de la secuencia de arranque

Si en cualquier instante ocurre un error que exiga parada de emergencia, la secuencia se suspende llegando al final de la función sin más trámites, y sin declarar el arranque concluido.

5.3.6. Función secuencia_parada

Esta función realiza todo el proceso de parada controlada de los dispositivos de potencia del Leon, es decir, la pila, los convertidores, y el variador del motor. Para ello envía las órdenes pertinentes de cambio de estado mediante mensajes CAN, comprobando que dichos estados se alcanzan sin errores, y siguiendo los diagramas de estado de cada uno de ellos.

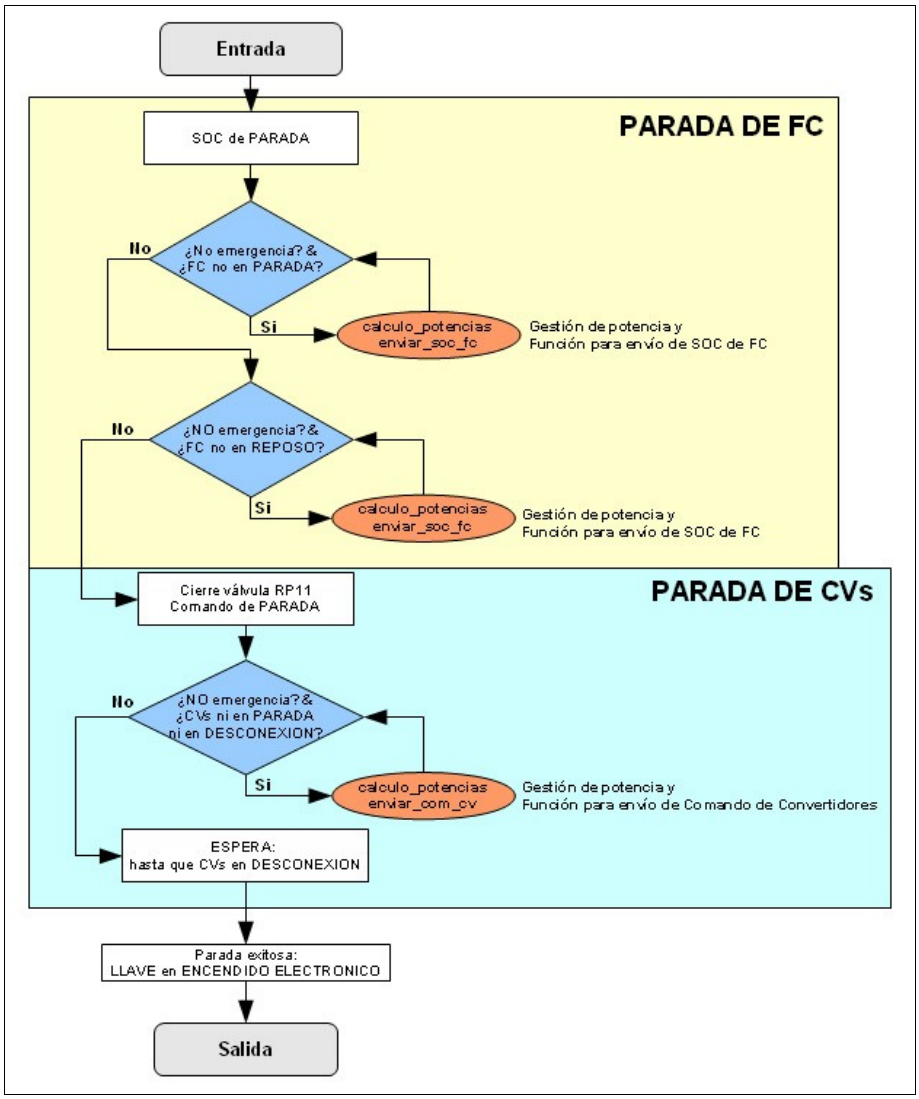

Imagen 66: diagrama de la secuencia de parada

Si en cualquier instante ocurre un error que exiga parada de emergencia, la secuencia se suspende llegando al final de la función sin más trámites, y sin declarar la parada concluida.

Es importante que se realice la gestión de la potencia a la misma vez para poder controlar los valores de las consignas durante los procesos de parada, ya que algunos exigen medidas concretas. Para ello se utilizan unas variables de control que gestionan la potencia de acuerdo a la necesidad de la secuencia, especialmente la de la pila de combustible. Dichas variables de control no aparecen en el diagrama (*fc_parada_normal y fc_desconectar*).

5.3.7. Función parada_emergencia

Esta función lleva a cabo la parada de emergencia ante cualquier situación que requiera una respuesta rápida que implique la necesidad de detener los dispositivos y llevarlos a un estado de seguridad.

Exige que se lleve a cabo a la misma vez la gestión de la potencia para poder aplicar las limitaciones necesarias a las consignas durante el proceso de parada de emergencia.

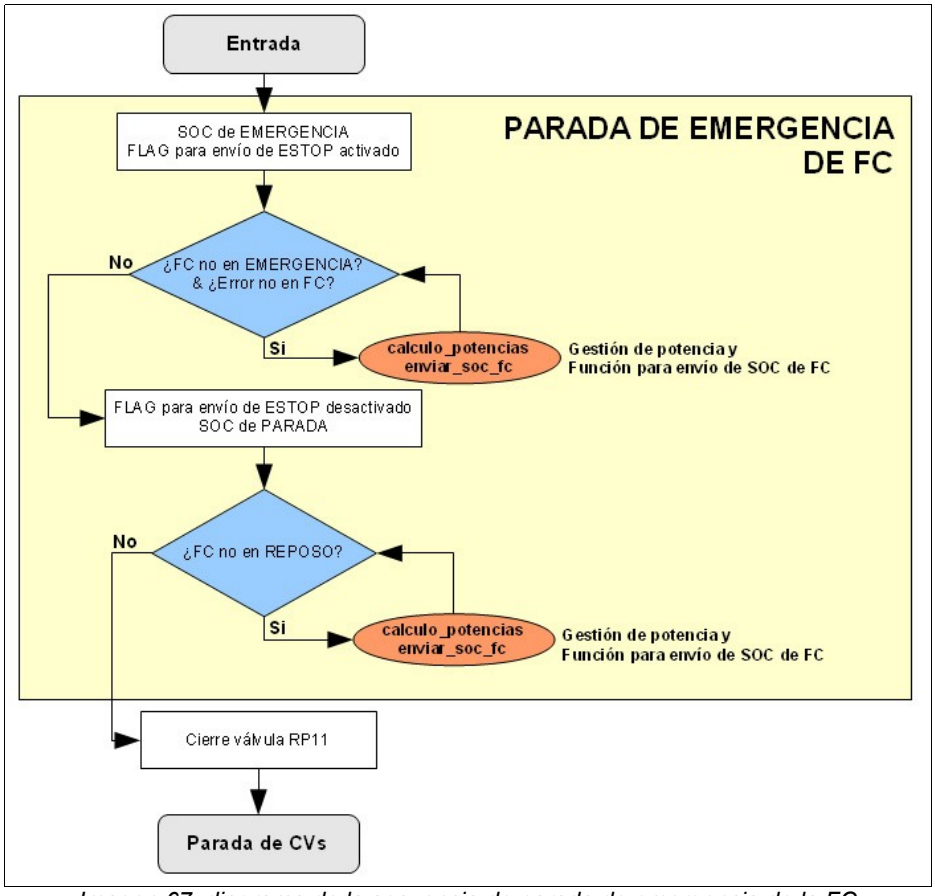

Imagen 67: diagrama de la secuencia de parada de emergencia de la FC

Esta es la parte más sensible, ya que exige controlar adecuadamente las consignas de las potencias. Por ello, existen unas variables de control (que no aparecen en el

diagrama, y que son: *fc_parada_normal y fc_desconectar*) que se utilizan para manejar la gestión de la potencia adecuadamente durante las secuencias.

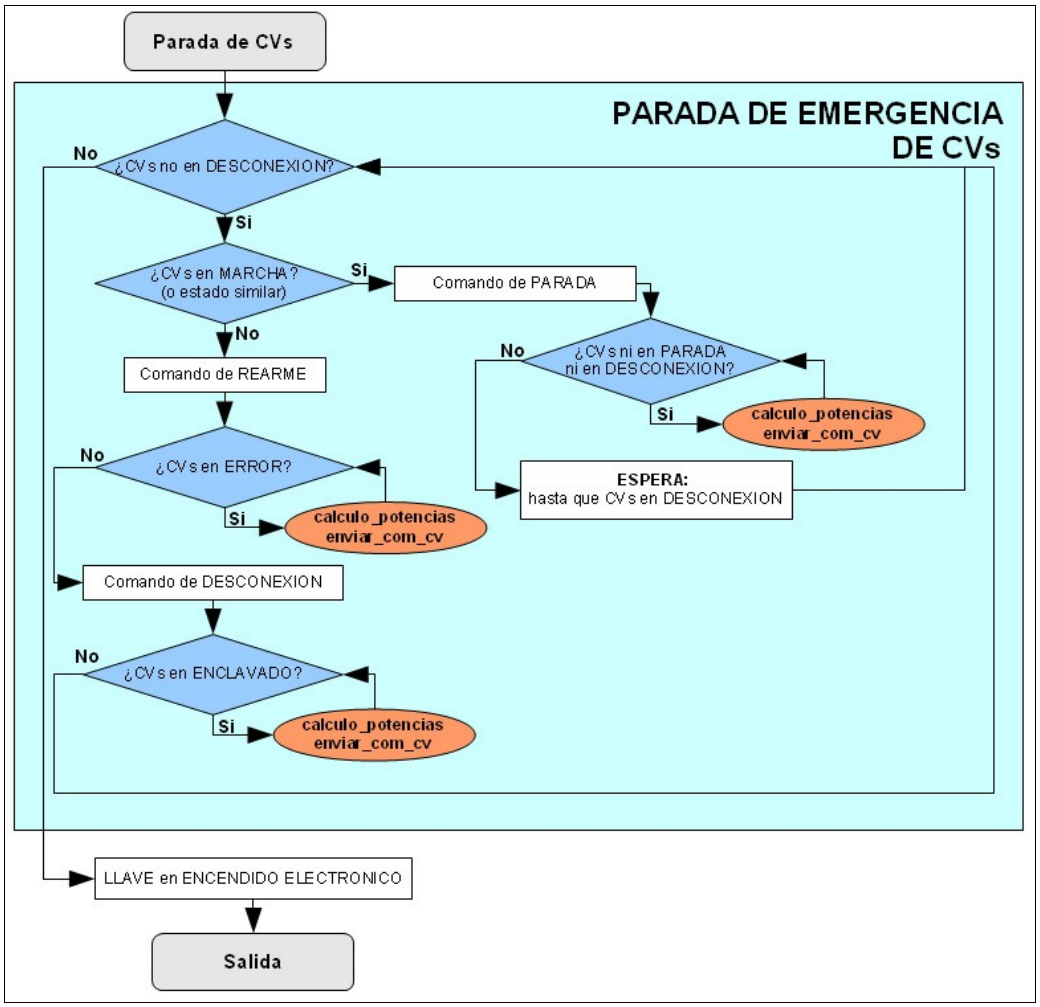

Imagen 68: diagrama de la secuencia de parada de emergencia de los convertidores

La secuencia de parada de emergencia de los Convertidores es más complicada ya que puede seguir dos caminos muy diferentes hasta detenerse, y uno de ellos puede llevar a los mismos a volver a ponerse en marcha, cuando lo primordial en esta secuencia es que se detengan totalmente. Por ello es importante responder si los mismos vuelven a activarse. Habría que llevarlos a desconexión en cualquier caso (ver anexo para diagrama de estados).

La secuencia completa es similar a los dos anteriores de arranque y de parada, pero no comprueba el estado de emergencia, porque se presupone que existe. Sin embargo, si ocurre un retraso en la respuesta de alguno de los componentes, termina dicha parte de la secuencia dando el aviso del error correspondiente, con lo que puede llegar a darse el caso en el que no pueda terminarse la parada en alguno de los componentes debido a que no respondan a cambios de estado. En esta situación se

alcanza el error más grave, el de “descontrol”, que implicará apagar el vehículo totalmente, para resetear todos los controles, ya que se ha perdido la capacidad de controlar los estados de los mismos.

5.3.8. Funciones para enviar comandos

Se trata de varias funciones que siguiendo la misma metodología pero específica para los siguientes componentes: pila de combustible, convertidores y variador; realiza el control del envío de los mensajes de Comando para llevar a cabo los cambios de estado en dichos dispositivos, avisando de los errores que puedan ocurrir, y abortando las secuencias.

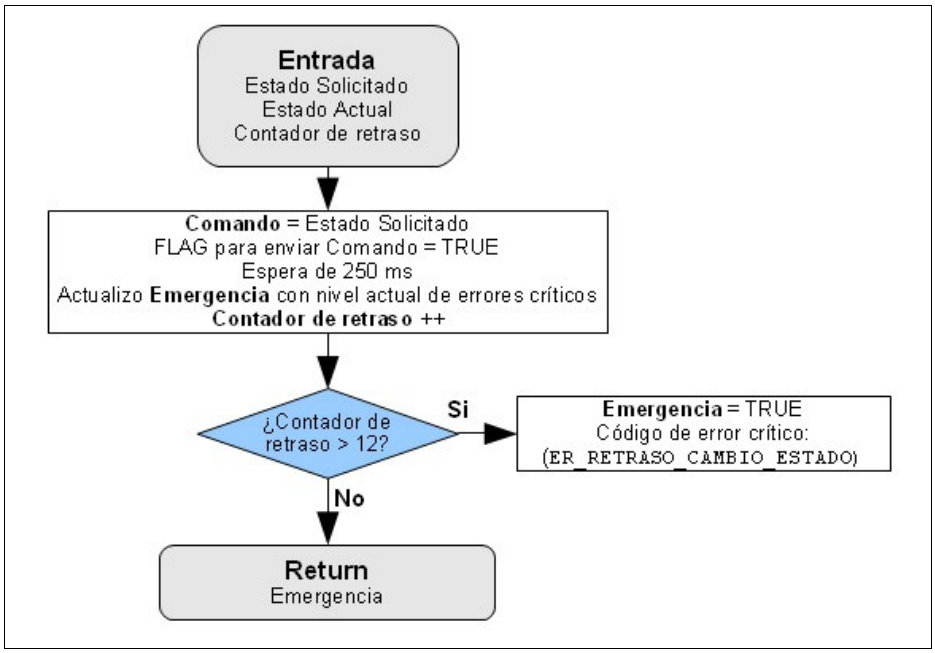

Imagen 69: diagrama de las funciones de envío de comandos

Los nombres de dichas funciones son: *enviar_soc_fc*, *enviar_com_cv* y *enviar_com_vm*. Se les pasa unos parámetros y devuelven el flag de emergencia. El objetivo es activar el flag para el envío del mensaje de comando con el estado solicitado, durante 12 ciclos de 250 ms, es decir, un total de 3 segundos. Si tras este tiempo no se han cumplido ciertas condiciones externas a la función llegará un instante en el que se superen esos 3 segundos, activando el error por retraso de cambio de estado, lo que también pondrá “true” en el flag de emergencia. Cualquier otro error externo también podría activar dicho flag durante uno de los ciclos ya que se actualiza la variable Emergencia con el nivel de error crítico.

El envío del mensaje de comando se realiza en el hilo de CAN, que comprueba periódicamente el valor del flag para enviar el comando. Tras cada envío se desactiva

dicho flag. Así se cumple la tríada “activar flag, enviar comando, desactivar flag”, que evitará que se envíen mensajes innecesarios.

5.3.9. Función calculo_potencias

Esta es la función que calcula las consignas de potencia en base a la posición del pedal, variables de control y los estados de los dispositivos. Es por tanto el núcleo de la gestión de la potencia.

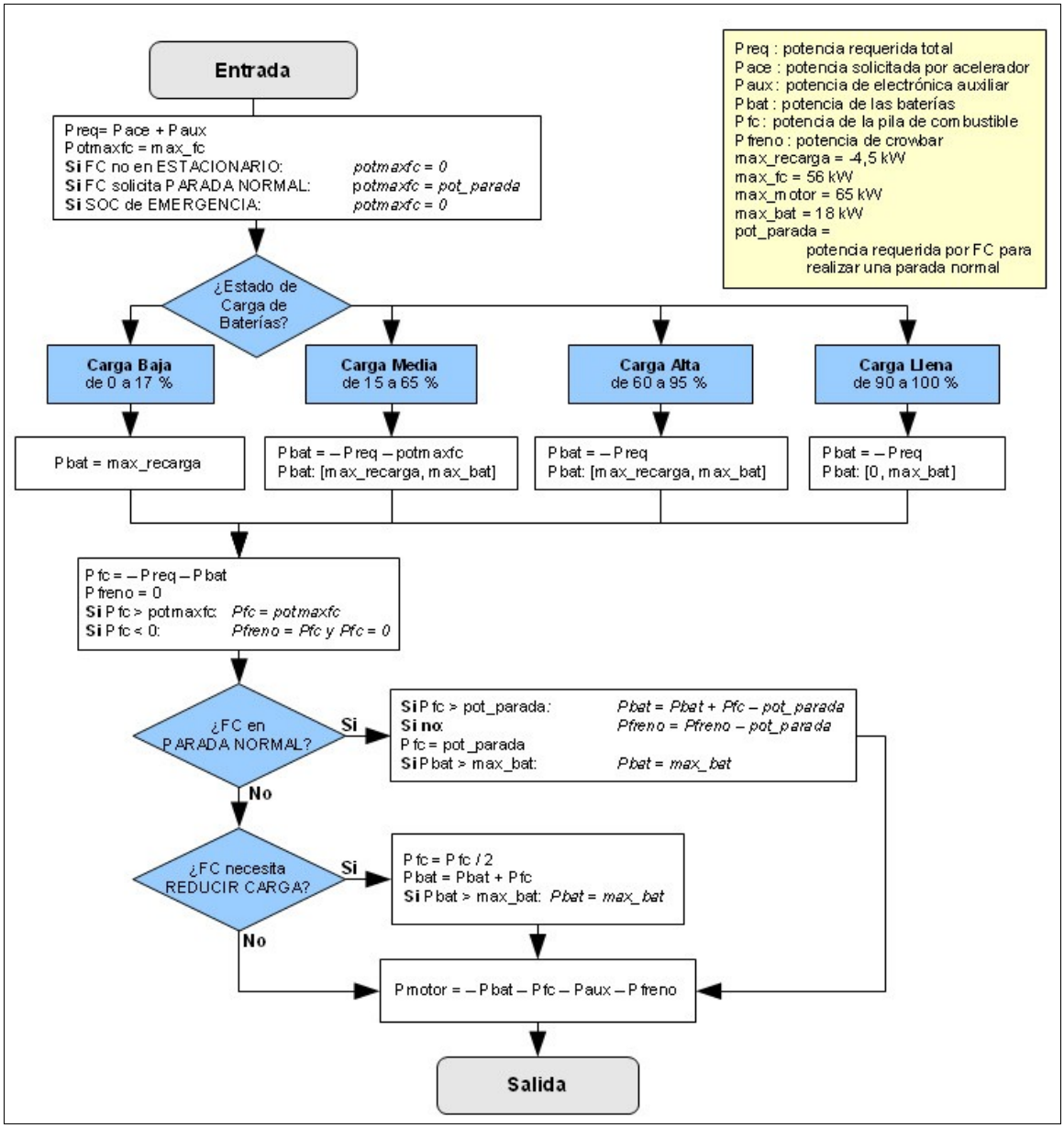

Imagen 70: diagrama de la gestión de la potencia

Todos los cálculos han sido realizados siguiendo las siguientes premisas:

- Los valores de las potencias están en unidades de **10 W**, ya que es el valor en el que las consignas se envían a los dispositivos.
- Una potencia **positiva** significa energía hacia el bus de potencia, y una potencia **negativa** es energía absorbida del bus de potencia.

La premisa de los signos nos permitirá tener un nodo en el bus de potencia para poder utilizar las leyes de Kirchhoff durante los cálculos, y así puedan servirnos en caso de que se requiera potencia para el motor, o se extraiga potencia del mismo. Las fórmulas serán iguales ya que se conserva la regla de signos.

Aunque no aparece en el diagrama, lo primero que hace esta función es comprobar que los convertidores están funcionando. Si es ese el caso, realiza todo los cálculos, si no, fija las consignas a 0 y termina.

Como se puede observar hace una serie de comprobaciones previas en base a las variables de control para manejar los estados en los que la FC pueda necesitar reducir la potencia, o que repentinamente se entre en un estado de emergencia en la misma. A continuación revisa el estado de carga de las baterías para fijar cuánta potencia se le pedirá a éstas en base al mismo (ver apartado 4.5.3.1). Después calcula la potencia de la FC y comprueba los límites físicos de las consignas, por ejemplo, que la potencia de la FC no sea negativa, ya que ésta solo genera. Así también se deduce el valor de la potencia en caso de resistencia de frenado (crowbar).

Finalmente se comprueban las variables de control para reducir potencias en caso de ser necesario, siempre manteniendo el balance de potencias en el bus, y se calcula la potencia destinada al motor.

5.3.10. Bucle principal

Para terminar este apartado solo resta analizar el bucle principal que es el que soporta los procedimientos más importantes de la ECU.

Se trata de un bucle infinito en el que se entra si la función *check_comm* no encontró errores de comunicaciones. Una vez dentro realiza cálculos y comprobaciones secuencialmente, mientras el resto de hilos está trabajando paralelamente. Este procesamiento concurrente es la base del funcionamiento de la ECU.

Lo primero que hace el bucle es calcular la velocidad lineal del vehículo, los niveles de carga de hidrógeno en los depósitos , y el estado de carga de las baterías. Luego lleva a cabo las comprobaciones pertinentes por si es necesario realizar alguna de las secuencias de funcionamiento de la ECU: arranque, parada y parada de emergencia.

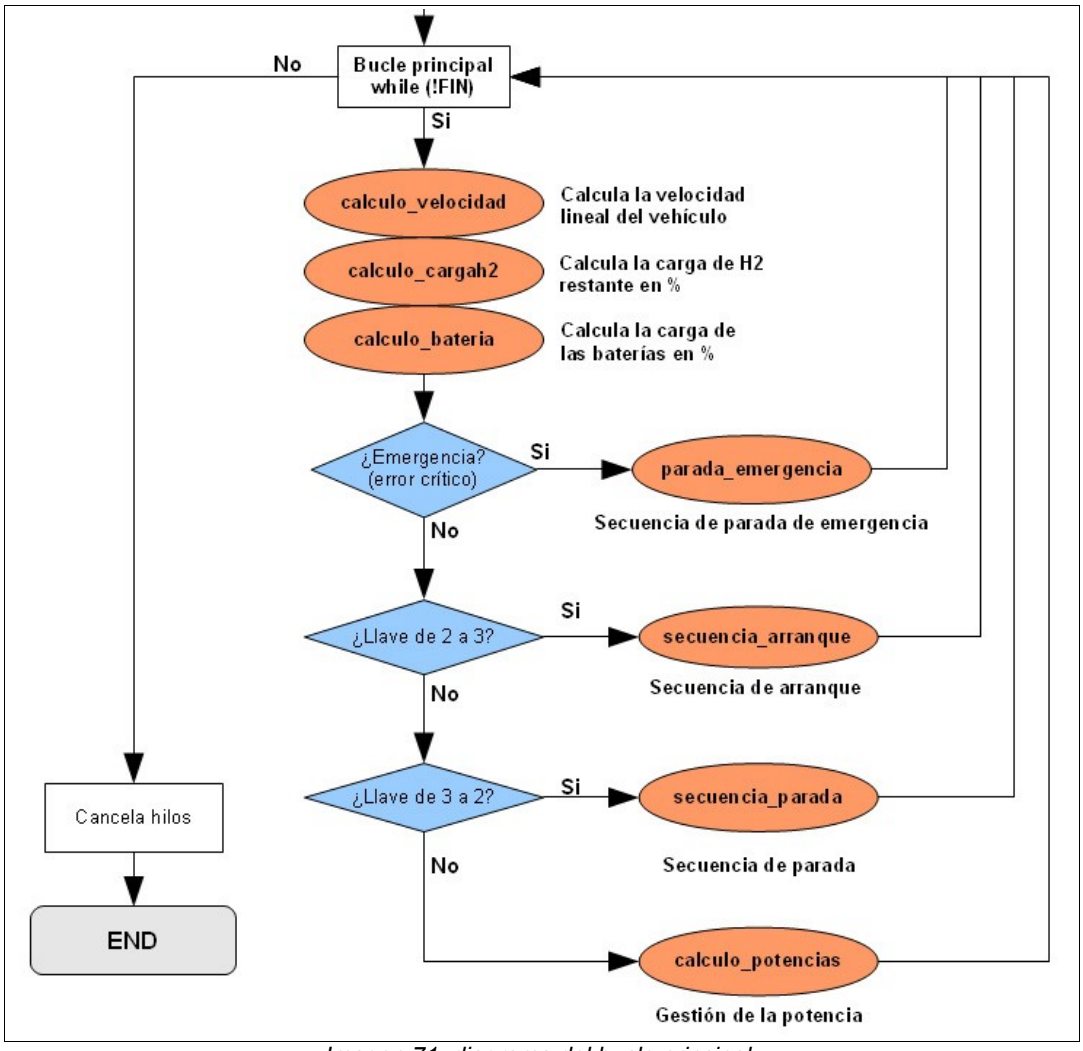

Imagen 71: diagrama del bucle principal

Finalmente, si no ha sido necesario realizar ninguna secuencia, lleva a cabo la gestión de la potencia. Este camino dentro del bucle principal será el común en el funcionamiento del coche, con la finalidad de calcular las consignas de potencia que serán tramitadas en los otros hilos.

5.4. Hilo de traducción de señales

El hilo de traducción de señales se encarga de intercambiar información por medio de las colas de mensaje con el proceso de manejo de la tarjeta de adquisición de datos. La información intercambiada se rige a cuatro estructuras de variables globales, dos para señales digitales, y dos para analógicas, y de cada dos, una son entradas y otra salidas:

- **Señales digitales de entrada** (*dig_rd*): estructura de variables globales para lectura de señales digitales, es decir, entradas.
- **Señales digitales de salida** (*dig_wr*): estructura de variables globales para escritura de datos para señales digitales de salida.
- **Señales analógicas de entrada** (*analog_in*): estructura de variables globales para lectura de las señales analógicas, es decir, entradas.
- **Señales analógicas de salida** (*analog_out*): estructura de variables globales para escritura de datos para señales analógicas de salida.

El intercambio según las colas de mensajes con el proceso de manejo de la tarjeta se puede ver en el siguiente diagrama:

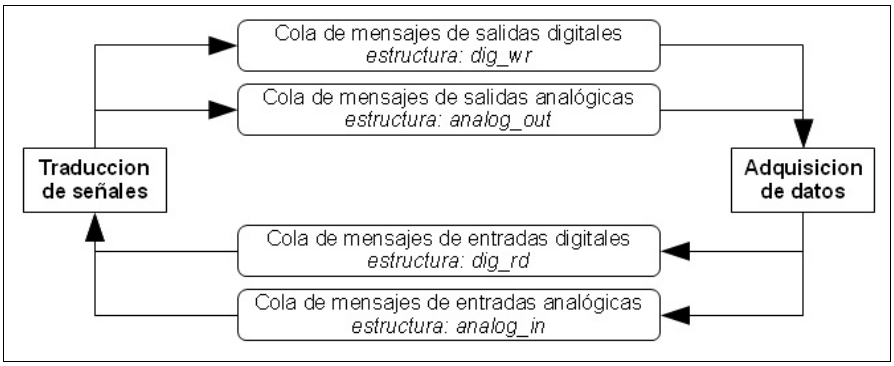

Imagen 72: colas de mensajes

El programa central sigue la misma metología que el proceso principal, es decir, incialización de variables y colas, y luego un bucle principal. En este caso el bucle llama a dos funciones concretas cada 200 ms: *digital* y *analog*.

5.4.1. Función digital

Este módulo recibe por referencia las colas de mensajes referidas a señales digitales y almacena los datos de las entradas que llegan por la cola de mensajes en la estructura destinada para éllo, dig_rd, y envía los datos de la estructura *dig_wr* para las salidas digitales por otra cola de mensajes.

Todo este proceso es secuencial y sin esperas.

5.4.2. Función analog

Esta función es similar a la anterior. Recibe por referencia las colas de mensajes de señales analógicas para almacenar los datos de las entradas en *analog_in* y enviar los datos de las salidas, guardados en *analog_out*.

También es un proceso secuencial y sin esperas. La diferencia estriba en que al final llama a otras dos funciones para que analicen los datos recibidos de la señales analógicas: *calculo_acelerador* y *calculo_presionH2*.

5.4.3. Función calculo_acelerador

Este módulo calcula la potencia que se solicita según la posición del pedal del acelerador. Dicha posición se suministra por medio de dos entradas analógicas redundantes para asegurar que se toma un valor fiable (ver apartado 4.5.4).

En base a dicha posición se calcula la potencia solicitada siguiendo la relación mostrada en la siguiente imagen. Se recuerda que se dispone de frenada regenerativa.

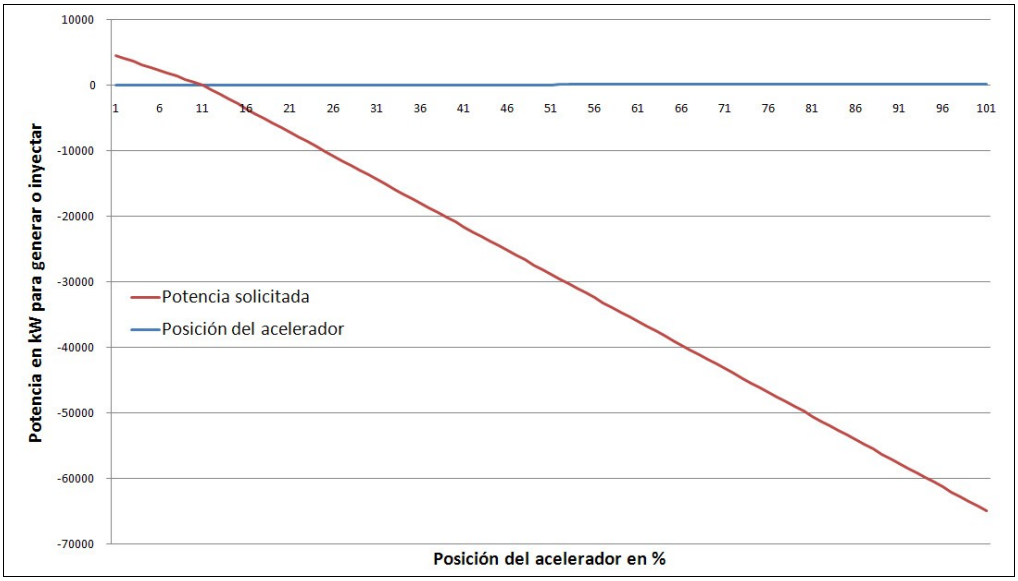

Imagen 73: relación potencia/posición del pedal del acelerador

Al final de la función se comprueban algunas variables de control:

- Puede ser necesario limitar la potencia solicitada al motor ante ciertos avisos (errores) que ocurren en él. Ésto se realiza limitando la potencia que se solicita desde el acelerador, y así se evita que siga subiendo la consigna final del motor.
- Si el coche no está en estado de arranque, la potencia solicitada desde el acelerador se anula.

5.4.4. Función calculo_presionH2

Esta función calcula las presiones y las temperaturas de los depósitos según los datos de las señales analógicas que provienen de los sensores. Son en total tres medidas de alta presión, una por depósito, una medida de baja presión, que representa la presión en el suministro hacia la FC, y tres medidas de temperatura, una por depósito.

5.5. Hilo de comunicación CAN

El hilo de comunicaciones CAN dependiente del proceso principal realiza varias tareas:

- **Manejo de la tarjeta CAN**: la función *can_main* controla el hardware de la comunicación CAN y gestiona la recepción y la transmisión de los mensajes de la siguiente manera
  - **Recepción**: se asocia la recepción de tramas con la recepción de un pulso durante el procesamiento de un bucle infinito que lo comprueba todo el tiempo, de tal manera que cada vez que se da éste, se realiza la lectura del mensaje CAN que acaba de llegar al buffer de recepción de la tarjeta, llamando a la función *can_in_datos*.
  - **Transmisión**: se asocia una señal a un temporizador de 50 ms que llama a un manejador (*manejador_contador_ciclos*) con cada aviso. En este manejador se controlan diferentes períodos multiplos de 50, para el envío de los mensajes siguiendo estos tiempos: 50 ms, 100 ms, 250 ms, 400 ms, 500ms, 750 ms.
- **Lectura** de los **datos** de los mensajes recibidos.
- **Envío** de los **mensajes** CAN siguiendo la regla temporal de cada uno.
- Controla la **dinámica** de las **consignas** de potencia realizando unos cálculos antes del envío de las mismas.

5.5.1. Función can_in_datos

Este módulo tiene un funcionamiento muy sencillo. De entrada se le pasa la estructura del mensaje CAN con los datos de la trama recien recibida. Recordemos que se llama a esta función cada vez que se recibe un mensaje. A continuación se comprueba su identificador y según éste se recorre una estructura switch para realizar el desglose de la trama según qué mensaje haya sido.

La única dificultad de esta tarea reside en convertir los datos de bytes a variables short o unsigned short. Para ello se ha utilizado cast y una función, *bytetosignedshort*, que convierte un byte a una variable signed short conservando el signo, considerando los valores negativos en complemento A2. La variable byte es de tipo unsigned char.

Para algunos mensajes (errores y watchdogs) se han utilizado variables de control para realizar comprobaciones en la función *manejador_contador_ciclos*, ya que en élla podemos considerar períodos.

5.5.2. Funcion manejador_contador_ciclos

Esta función es muy importante para la ECU, ya que es el único sitio en el que se controlan períodos, lo que nos permitirá hacer comprobaciones de recepción de ciertos mensajes además de enviar otros. Se realizan las siguientes tareas en cada período:

- **Períodos de 100 ms**: en el siguiente orden:
  - Llama a la funcion *dinamica_potencia.*
  - Envío del SOC de la FC si el flag de envío del mismo está activado. Al final desactiva el flag.
  - Envío de Emergency Stop a la FC si el flag de envío del mismo está activado. Dicho flag se desactiva en la secuencia de parada de emergencia, que es donde se maneja el susodicho.
  - Envío del Comando de los Convertidores si el flag de envío del mismo está activado. Al final desactiva el flag.
  - Envío del Comando del Variador y el Motor si el flag de envío del mismo está activado. Al final se desactiva el flag.
- **Períodos de 250 ms**:
  - Se envía el SOC de la FC.
  - Se envía la Consigna de potencia para el convertidor de la FC.
  - Se envía la Consigna de potencia para el convertidor de baterías.
  - Se envía la Consigna de potencia para el convertidor del crowbar.
  - Se envía la Consigna de potencia para el Variador y el Motor.
  - Comprueba si ha llegado un mensaje de error de FC durante este período mirando el valor de una variable de control (*errores*.*er_fc_msg*) que se activa cuando recibe dicho mensaje. Si pasa un tiempo determinado sin recibir mensajes de este tipo, se ponen a 0 todos sus errores.
  - Comprueba si ha llegado uno de los mensajes de error de los Convertidores durante este período mirando el valor de una variable de control (*errores*.*er_cv_msg*) que se activa cuando recibe uno de dichos mensajes. Si pasa un tiempo determinado sin recibir mensajes de este tipo, se ponen a 0 todos sus errores.
  - Comprueba si ha llegado un mensaje de error del Variador y Motor durante este período mirando el valor de una variable de control (*errores*.*er_vm_msg*) que se activa cuando recibe dicho mensaje. Si pasa un tiempo determinado sin recibir mensajes de este tipo, se ponen a 0 todos sus errores.

- **Períodos de 400 ms**:
  - Envío del Heartbeat.
- **Períodos de 500 ms**:
  - Se comprueba si ha llegado el Watchdog de la FC durante este período mirando el valor de una variable de control (*fuelcell.fc_watchdog*) que se activa cuando recibe dicho mensaje. Si pasa un tiempo determinado sin recibirlo, da error de falta de watchdog.
  - Se comprueba si ha llegado el Watchdog de los Convertidores durante este período mirando el valor de una variable de control (*convertidores.conv_watchdog*) que se activa cuando recibe dicho mensaje. Si pasa un tiempo determinado sin recibirlo, da error de falta de watchdog.
  - Se comprueba si ha llegado el Watchdog de la FC durante este período mirando el valor de una variable de control (*vmotor.vm_watchdog*) que se activa cuando recibe dicho mensaje. Si pasa un tiempo determinado sin recibirlo, da error de falta de watchdog.
- **Períodos de 750 ms**:
  - Envío de todos los mensajes Info from ECU.

Volvemos a tener la misma dificultad de convertir entre variables, para lo que usaremos cast y una función llamada *signedshorttobyte* que nos permitirá convertir de signed short a byte conservando el signo, de nuevo considerando los valores negativos en complemento A2. La variable byte es de tipo unsigned char.

5.5.3. Funcion dinamica_potencia

Realiza el control de la dinámica de las consignas de potencia. Utiliza las consignas calculadas en la función *calculo_potencia* como referencias y ajusta el aumento de los setpoints finales a una regla simple de 100 W cada 100 ms. La periodicidad se mantiene gracias a que se llama a esta función cada 100 ms desde el módulo anteriormente explicado: *manejador_contador_ciclos*.

5.5.4. Función bytetosignedshort

Convierte una variable de 8 bits con signo, byte (unsigned char), formateada con complemento A2, a una variable de 16 bits con signo. La regla a seguir es muy sencilla:

- Un número positivo tendrá el primer bit a 0, y al aumentar el número de bits, los

8 nuevos son también 0.

➢ Un número negativo tendrá el primer bit a 1 (complemento A2), y al aumentar el número de bits, para mantener el valor exactamente basta añadir 8 bits a valor 1.

Por ejemplo:

Valor en decimal	Byte	Signed shot
0	(binario) 00000000	(binario) 00000000/00000000
127	(binario) 01111111	(binario) 00000000/01111111
-70	(binario) 10111010	(binario) 11111111/10111010
-128	(binario) 10000000	(binario) 11111111/10000000
-1	(binario) 11111111	(binario) 11111111/11111111

Tabla 55: conversión byte a signed short

5.5.5. Función signedshorttobyte

Convierte una variable signed short de 16 bits a una tamaño byte (unsigned char) para introducir la variable en una trama de datos de CAN. La regla seguida es la misma que en el caso de *bytetosignedshort,* solo que en vez de añadir bits, se quitan los 8 de mayor peso, conservando siempre el último bit de los 8 de menor peso, según si era positivo o negativo:

➢ Si es un número positivo se hace un AND con 0x0007 para asegurarse que el octavo bit es 0 y los demás conservan el valor original.

➢ Si es un número negativo se hace un OR con 0xFFE0 para asegurarse que el octavo bit es 1 y los demás conservan su valor original.

Finalmente basta con hacer un cast a tipo byte (unsigned char).

5.6. Hilo de supervisión de errores

Este hilo de proceso, dependiente del principal, realiza toda la supervisión de los errores por medio de las siguientes tareas, que se llevan a cabo cada 200 ms dentro de un bucle infinito:

- Comprobación de los **estados de errores**, activando o desactivando bits en los códigos de errores, que son variables short (16 bits) dentro de la estructura *errores*.
- Comprobación del **nivel de error** actual: leve, grave o crítico.
- Protocolo de **actuación**.
- **Reseteo** de algunos estados de errores pasados un tiempo definido.

5.6.1. Función h2_errores

Esta función toma todas las medidas referidas a los depósitos de hidrógeno y los sensores de detección de fugas, y una por una compara sus valores con los límites para las alarmas y errores. Va activando y desactivando (poniendo a 1 o 0) los bits en los códigos de error correspondiente, sean leves, graves o críticos.

5.6.2. Función fc_errores

Este módulo realiza una tarea similar, pero a partir de los códigos de error que transmite la FC a la ECU. Por tanto lo que hace es desglosar uno por uno los códigos y guardando los valores en los códigos globales de la estructura *errores,* que es la que almacena todas las alarmas del León. De esta manera activa y desactiva los errores según los códigos que transmite la FC, aunque existe otra manera de poner a cero todos los errores de la susodicha, y es cuando, como ya se ha comentado anteriormente, se dejan de recibir mensajes de errores de la pila, lo que implica que no hay alarmas.

5.6.3. Función conv_errores

Este módulo realiza la misma tarea que el anterior, pero a partir de los códigos de error que transmite el controlador de los Convertidores a la ECU. Por tanto lo que hace es desglosar uno por uno los códigos y guardando los valores en los códigos globales de la estructura *errores,* que es la que almacena todas las alarmas del León. De esta manera activa y desactiva los errores según los códigos que transmite dicho controlador, aunque existe otra manera de poner a cero todos los errores de los susodichos, y es cuando, como ya se ha comentado anteriormente, se dejan de recibir mensajes de

errores de los convertidores, lo que implica que no hay alarmas en ellos. En este caso se disponen de 3 mensajes de errores diferentes, pero solo se limpian todos cuando pasa un tiempo sin recibir ninguno de ellos.

5.6.4. Función vm_errores

Este módulo realiza la misma tarea que los anteriores, pero a partir de los códigos de error que transmite el Variador (y motor) a la ECU. Por tanto lo que hace es desglosar uno por uno los códigos y guardando los valores en los códigos globales de la estructura *errores,* que es la que almacena todas las alarmas del León. De esta manera activa y desactiva los errores según los códigos que transmite el Variador-Motor, aunque existe otra manera de poner a cero todos los errores del susodicho, y es cuando, como ya se ha comentado anteriormente, se dejan de recibir mensajes de errores del mismo, lo que implica que no hay alarmas en ellos.

5.6.5. Función actuacion_critica

Habiendo comprobado el nivel de gravedad actual, si se está en situación crítica se accede a esta función, que simplemente comprueba si el error crítico en cuestión, o los errores, requieren alguna actuación especial.

En este caso solo se comprueba si el error fue originado en la FC, para tenerlo en cuenta durante la secuencia de parada de emergencia.

5.6.6. Función actuacion_grave

Habiendo comprobado el nivel de gravedad actual, si se está en situación grave se accede a esta función, que simplemente comprueba si el error grave en cuestión, o los errores, requieren alguna actuación especial.

En este caso se realizan las dos comprobaciones siguientes:

- En el caso de ciertos errores de la FC, se hace necesario llevar a cabo una parada normal de la misma. Si es el caso, se activa un flag de control para avisar al gestor de potencia.
- En el caso de ciertos errores en el Variador y Motor, se activa un flag de control que avisará a la función que calcula la posición del acelerador para no seguir subiendo la potencia requerida. La razón es que dichos errores requieren limitar la potencia solicitada al motor.

5.6.7. Función actuacion_leve

Habiendo comprobado el nivel de gravedad actual, si se está en situación leve se accede a esta función, que simplemente comprueba si el error leve en cuestión, o los errores, requieren alguna actuación especial.

En este caso se realizan las dos comprobaciones siguientes:

- En el caso de ciertos errores de la FC, se hace necesario reducir a la mitad la potencia que se le solicita. Para ello se activa un flag de control que avisará al gestor de potencia.
- En el caso de ciertos errores en el Variador y Motor, se activa un flag de control que avisará a la función que calcula la posición del acelerador para no seguir subiendo la potencia requerida. La razón es que dichos errores requieren limitar la potencia solicitada al motor.

5.7. Relación de archivos de código

Todo el software de la ECU incluye varios archivos de librerías y de código, que se puede relacionar con lo expuesto en los apartados anteriores de la siguiente manera:

- Archivo **ecu_pc104.c**: incluye todo el código referido al **proceso principal** según el siguiente esquema:
  - Inclusión de librerías.
  - Declaración de las estructuras de las variables globales.
  - Declaración e inicialización de los mutex.
  - Función main.
  - Función check_comm.
  - Función secuencia_arranque.
  - Función secuencia_parada.
  - Función parada_emergencia.
  - Función calculo_potencias.
  - Función calculo_bateria.
  - Función calculo_velocidad.
  - Función calculo_H2.
  - Función enviar_soc_fc.
  - Función enviar_com_cv.
  - Función enviar_com_vm.
- Archivo **com_ecu104.c**: incluye todo el código referido al **hilo de traducción de señales** según el siguiente esquema:
  - Inclusión de librerías.
  - Declaración externa de las estructuras de las variables globales.
  - Declaración externa de los mutex.
  - Declaración de variable global de control, propia de este hilo.
  - Función adq_main.
  - Función digital.
  - Función analog.
  - Función calculo_acelerador.

- Función calculo_presionH2.

➢ Archivo **can_ecu104.c**: incluye todo el código referido al **hilo de comunicación CAN** según el siguiente esquema:

- Inclusión de librerías.
- Declaración externa de las estructuras de las variables globales.
- Declaración externa de los mutex.
- Declaración de variables globales de control, propias de este hilo.
- Función can_main.
- Función can_in_datos.
- Función manejador_contador_ciclos.
- Función bytetosignedshort.
- Función signedshorttobyte.
- Función dinamica_potencia.

➢ Archivo **errores_ecu104.c**: incluye todo el código referido al **hilo de supervisión de errores** según el siguiente esquema:

- Inclusión de librerías.
- Declaración externa de las estructuras de las variables globales.
- Declaración externa de los mutex.
- Declaración de variable global de control, propia de este hilo.
- Función errores_main.
- Función actuacion_critica.
- Función actuacion_grave.
- Función actuación_leve.
- Función h2_errores.
- Función fc_errores.
- Función conv_errores.
- Función vm_errores.

➢ Librería **ecu_pc104.h**: es la librería más importante y la que tiene todas las declaraciones y definiciones de la ECU, salvando las propias de las tarjetas. Se sigue el siguiente esquema:

- Definiciones de limites, rangos, errores, canales de comunicaciones,

identificadores de CAN, etc.

- Definiciones de los tipos de estructuras que se usan para las variables globales de la ECU.
- Declaración de funciones del proceso principal y los tres hilos dependientes.

➢ Librería **colas.h**: es una librería común entre los procesos propios de la ECU y el de manejo de la tarjeta de adquisición de datos. Sirve para permitir el intercambio de colas de mensajes, y tiene definido todo lo necesario para ello: tamaños, estructuras (incluyendo las cuatro para las colas) y nombres.

➢ Otras librerías que son necesarias para poder manejar la tarjeta de comunicación CAN son: **candef.h**, **canglob.h**, **canstr.h**. Éstas son proveídas por el fabricante de la tarjeta CAN.