4 Programación de la ECU

La programación se ha realizado en lenguaje C sobre el sistema operativo QNX. Se ha hecho siguiendo unos criterios de modularidad para facilitar su depuración, y de paralelismo mediante el uso de subprocesos o hilos. El concepto de hilo permite la ejecución al mismo tiempo de numerosas tareas de forma pseudoparalela que comparten zonas de memoria comunes, como las variables globales. Esto requiere de estrategias de programación concurrente, por lo que se sigue el estándar POSIX orientado a sistemas en tiempo real con alta portabilidad del código.

4.1 Variables globales

Estas variables son accesibles por varios hilos a la vez, por lo que se requiere del uso de semáforos binarios o mutex asociados a cada una de ellas. La mayoría se han declarado en forma de estructuras, lo que permite asociarlas por su naturaleza.

- Variables de adquisición de datos: entrada_cad, salida_cda, entrada_dig, salida_dig. Contienen los valores de entrada y salida de la tarjeta de adquisición analógica/digital.
- Variables de las medidas de los convertidores: conv_bat, conv_fc1, conv_fc2, pb. Contienen los valores de tensión, corriente y estado del bus de potencia mandados por los convertidores.
- Comandos de la GUI: comandos_pccar. Contiene los comandos seleccionados por el usuario en la pantalla táctil.
- Control de las pilas: fc1, fc2. Contienen valores relevantes del estado de las pilas.
- Gestión de potencia: gest_pot. Contiene los valores relevantes de la gestión de potencia.
- Datos del CVM: datos_cvm_fc1, datos_cvm_fc2. Contienen los valores de las medidas de las tensiones de pares de celdas de ambas pilas.
- Errores: errores_cond, errores. Contienen condiciones de activación de errores y los errores que se han activado.

4.2 Librerías

Es necesaria la inclusión de las librerías de los controladores de la tarjeta de adquisición y la tarjeta CAN en el enlazado del programa: libdscud5.a y can.a respectivamente.

4.3 Cabeceras

Los archivos de cabecera tienen extensión .h y en ellos se declaran las librerías estándares utilizadas, así como las macros que definen los valores constantes (límites de las variables, condiciones de error, etc.).

4.4 Organización del programa

El programa consiste en un proceso principal que lanza y gestiona los diferentes hilos. Asimismo, arranca un proceso hijo para realizar las medidas del CVM y recibe los datos del mismo a través de una cola de mensajes.

La estructura de todos los hilos es similar. En primer lugar tienen una etapa de inicialización donde el hilo se prepara para realizar su tarea, consistente en reservar memoria, iniciar controladores, inicializar valores, etc. Una vez está listo lo indica mediante un flag que el proceso principal puede leer y espera la señal de mandato de inicio general desde el mismo. A partir de ese momento, ejecuta sin cesar un bucle donde realiza su tarea, del que sale únicamente si el flag correspondiente ha sido cambiado por el proceso principal o el hilo de errores, debido a un fallo o a la terminación normal del programa. Finalmente, al salir del bucle, se liberan las reservas y se finaliza el hilo.

Para controlar que ningún hilo se queda en un estado bloqueado, se hace uso de un "watchdog", que consiste en el uso de un temporizador que pone la bandera correspondiente de cada hilo a 1. Los hilos, en cada iteración de su bucle de operación, deben poner esta bandera a 0. Si después de un número determinado de interrupciones del temporizador esa bandera no ha cambiado, será indicativo de que el hilo está bloqueado.

4.5 El proceso principal

- Inicialización:
  1. Arranca el proceso hijo del CVM y espera un cierto tiempo a que se inicie.
  2. Inicializa el temporizador del *watchdog*.
  3. Da valores iniciales a las variables globales e inicializa sus mutex.
  4. Inicia los hilos de adquisición de datos, de comunicación CAN y de control de purgas.
  5. Espera a que estos hilos estén listos y les da orden de operar.
  6. Abre la cola de recepción de mensajes creada por el proceso CVM.
  7. Inicia los hilos de errores, de gestión de potencia y de control de la pila 1 y 2.
  8. Espera a que estos hilos estén listos y les da orden de operar.

9. Arranca el temporizador del *watchdog*.

- Bucle:
  1. Llama a la función recoge_cola para leer los datos del CVM.
  2. Pone a 0 la bandera del *watchdog*.
  3. Comprueba si la variable de orden de fin desde la GUI se ha activado: si es así, sale del bucle.
- Finalización:
  1. Activa la variable de asentimiento a la orden de fin para la GUI y espera un tiempo a que aquélla inicie su cierre.
  2. Desactiva la bandera que indica que los hilos están listos, para que salgan de sus bucles y finalicen. Espera un tiempo determinado.
  3. Cancela el temporizador del *watchdog*.
  4. Cancela los hilos por si acaso alguno no ha sido capaz de finalizar.
  5. Mata al proceso hijo del CVM.
  6. Borra la cola de mensajes y sale.

4.5.1 Función recoge_cola

Llamada por el proceso principal, comprueba en los atributos de la cola de mensajes si hay alguno. Si es así, los lee y actualiza las variables correspondientes: datos_cvm_fc1 y datos_cvm_fc2.

4.5.2 Manejador del temporizador del *watchdog*

Al producirse la interrupción por expiración del temporizador, se lanza esta función que comprueba si la bandera global del *watchdog* que ella misma escribe con valor 0xFF, permanece así con respecto a la interrupción anterior. Si es así, actualiza la variable de error correspondiente.

4.5.3 Funciones de impresión por pantalla

Son funciones auxiliares usadas durante el desarrollo del programa para imprimir en una pantalla los valores de las variables.

4.6 El hilo de adquisición de datos

Su función es actualizar las variables globales de medidas analógicas y digitales de entrada, entrada_cad y entrada_dig, así como escribir las de salida, salida_cda y salida_dig.

- Inicialización: llama a la función inicia_tadq para configurar e iniciar el controlador de la tarjeta de adquisición.
- Bucle:
  1. Recoge la entrada digital usando la función dig_rd.

2. Escribe la salida digital usando la función dig_wr.
3. Para cada canal de la entrada analógica, lee el valor sucesivamente haciendo uso de la función cad_rd.
4. Interpreta las lecturas convirtiéndolas a las unidades de trabajo y las guarda en la variable global correspondiente. Asimismo, prepara las de salida, todo ello llamando a la función interpreta_analog.
5. Escribe las salidas analógicas en cada canal sucesivamente.

4.6.1 Función inicia_tadq

Mediante llamadas a la librería del controlador de la tarjeta de adquisición, la inicia y establece sus parámetros de configuración: dirección base de la memoria, rango, ganancia...

4.6.2 Función calibra_adq

Función auxiliar que realiza una calibración de la tarjeta en el inicio si es necesario, siguiendo las indicaciones del fabricante.

4.6.3 Función cda_wr

Convierte el valor de tensión deseado para una salida analógica al valor tratable por el convertidor D/A como se indica en 4.1. Seguidamente escribe este valor en el canal.

$$Code_{out} = V_{out} \frac{4096}{Range_{CD/A}} \tag{4.1}$$

4.6.4 Función cad_rd

Lee un canal de la entrada analógica y convierte su valor a voltios. Si la entrada es unipolar:

$$V_{in} = \frac{Code_{in} + 32768}{65536} Range_{CA/D} \tag{4.2}$$

Si la entrada es bipolar:

$$V_{in} = \frac{Code_{in}}{32768} Range_{CA/D} \tag{4.3}$$

4.6.5 Función dig_wr

Escribe el octeto indicado en la salida digital.

4.6.6 Función dig_rd

Lee el octeto de la entrada digital. Al ser negada, el valor recogido es $255 - Byte$.

4.6.7 Función interpreta_analog

Convierte los valores en voltios recogidos de la entrada analógica a las unidades correspondientes para el guardado en la variable global. Hace el proceso inverso para las salidas analógicas.

4.6.8 Función termistores

Contiene la operación de conversión para los termistores.

4.7 El hilo de comunicación CAN

Este hilo inicializa el controlador para el puerto CAN usando la librería de funciones del fabricante y lleva la carga de las interrupciones producidas por la llegada de mensajes.

- Inicialización: crea el canal, la señal y el evento asociados al puerto CAN 1, mediante llamadas al controlador.
- Bucle:
  1. Se encuentra en estado de espera hasta que le llega la señal de mensaje en el *buffer*.
  2. Lee el mensaje y discrimina a qué hilo le corresponde según el identificador.
  3. Si es una respuesta de los convertidores, lo guarda en una variable y avisa con una señal de condición.
  4. Si no, lo guarda en un vector apilado para que el hilo de comunicación con la GUI lo trate.

4.8 El hilo de comunicación con los convertidores

Este hilo consiste en un bucle que en cada iteración forma el mensaje CAN de monitorización siguiendo lo indicado en la Sección 3.8, para cada uno de los tres convertidores. Lo manda usando la función manda_comando_conv. Si ésta devuelve error, acumula un error de comunicación si no, lo decrementa.

4.8.1 Función manda_comando_cv

Esta función puede ser llamada por cualquier hilo y realiza en un paso todo el procedimiento de mandar un mensaje al bus CAN para los convertidores y esperar respuesta/asentimiento. Ésta última la espera aguardando la señal condicional del hilo de comunicación CAN con un límite de tiempo. Si el límite se supera o el mensaje de respuesta es incorrecto, devuelve error. Si la respuesta contiene información relevante (de monitorización), la almacena en la variable correspondiente.

4.8.2 Función bytestofloat y floattobytes

Son dos funciones que facilitan la conversión de un dato flotante que está expresado en un vector de octetos y viceversa para que ésta se haga en un solo paso, ya que es una conversión muy utilizada para el tratamiento y formación de los mensajes CAN.

4.9 El hilo de comunicación con la GUI

Este hilo inicializa el temporizador y activa el temporizador que manda el lote de mensajes al PC-Car, cargando con sus interrupciones. Mientras tanto, también comprueba si hay mensajes en la variable de recepción para el PC-Car y los interpreta con la función rx_pccar.

4.9.1 Función rx_pccar

Interpreta los mensajes mandados desde la GUI a la ECU. Leyendo el identificador los discrimina y actualiza la variable correspondiente que contiene el mensaje, esto es, si es un *heartbeat* con el valor inicial del SOC y la pila dominante, si es el comando de salida del programa o si es un comando destinado a los convertidores o a las pilas.

4.9.2 Función tx_pccar

Esta función adapta las variables que se van a mandar al PC-Car y almacena todo un lote de mensajes CAN en un vector.

4.9.3 Manejador del temporizador de transmisión a la GUI

Cuando salta la interrupción del temporizador, esta función llama a tx_pccar y manda los mensajes CAN almacenados en el susodicho vector uno a uno, sin esperar respuesta. A su vez, incrementa el valor del *heartbeat*, que se resetea cuando llega algún mensaje desde la GUI. Si este valor alcanza cierto nivel, indica el error.

4.9.4 Función adapta_var

Esta función es llamada por tx_pccar para convertir cada variable que se va a mandar al PC-Car en el rango y resolución adecuados.

4.10 El hilo de control de la pila de hidrógeno

Hay dos hilos de control, uno para cada pila. Su estructura es la misma, si bien las variables son las correspondientes a cada una. Este hilo implementa la máquina de estados definida para la pila, haciendo uso de la variable que indica su estado en cada momento y cambiando el mismo a través de las funciones siguientes.

- Inicialización:
  1. Lee el estado de la pila en la variable fc{i}.
  2. Si el estado es de ERROR, no da aviso al proceso principal de que está listo, por lo que queda bloqueado en espera de alguna interrupción.
  3. Si el estado no es de ERROR, llama a reposo_fc para asegurar que la pila está en estado de REPOSO, lista para funcionar.
- Bucle:
  1. Si hay comando de cambio de estado de la GUI para la pila, mira cuál es el estado pedido.
  2. Si el estado actual es REPOSO y se pide ARRANQUE, se llama a start_fc.

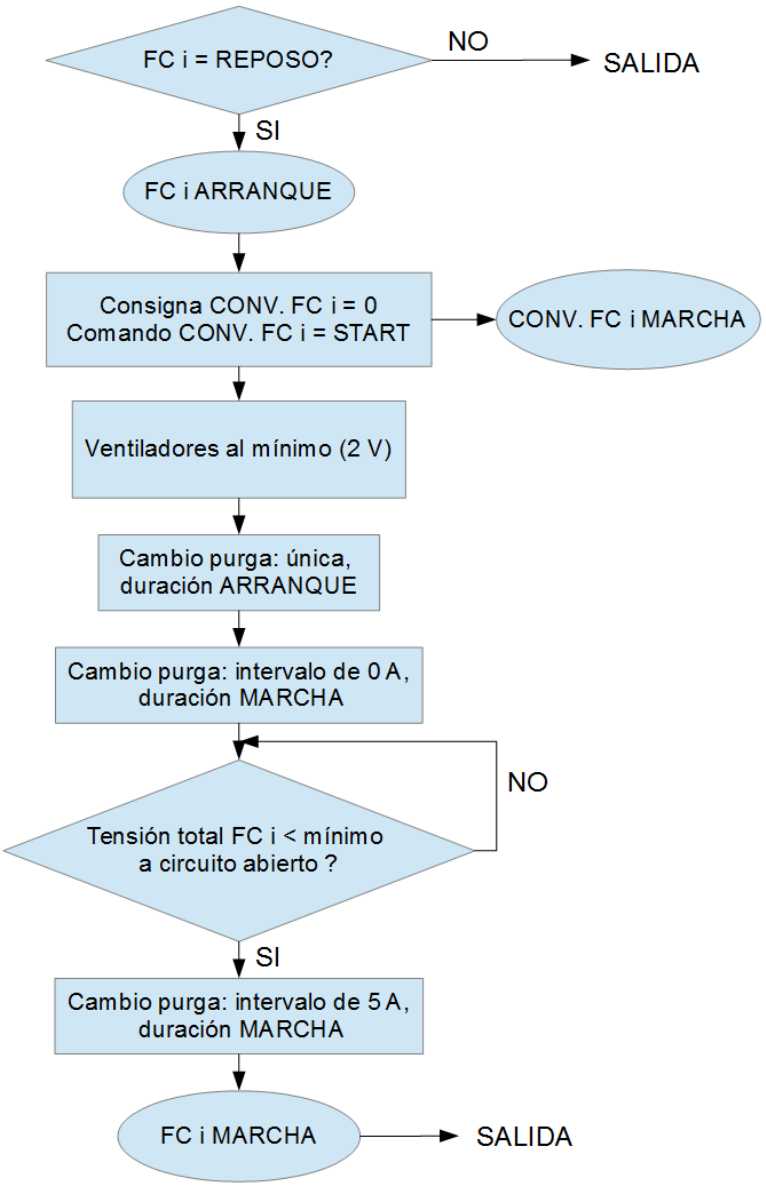

Figura 4.1 Función start_fc.

3. Si el estado actual es MARCHA o TRANSITORIO y se pide PARADA, se llama a stop_fc.
4. Si el estado actual es ERROR y se pide REPOSO, se llama a reposo_fc.
5. En todo caso, si el estado actual es MARCHA o TRANSITORIO, se llama a control_marcha.

4.10.1 Función start_fc

Pasa del estado de REPOSO al de MARCHA, realizando la secuencia de ARRANQUE. En la Figura 4.1 se incluye el diagrama de flujo.

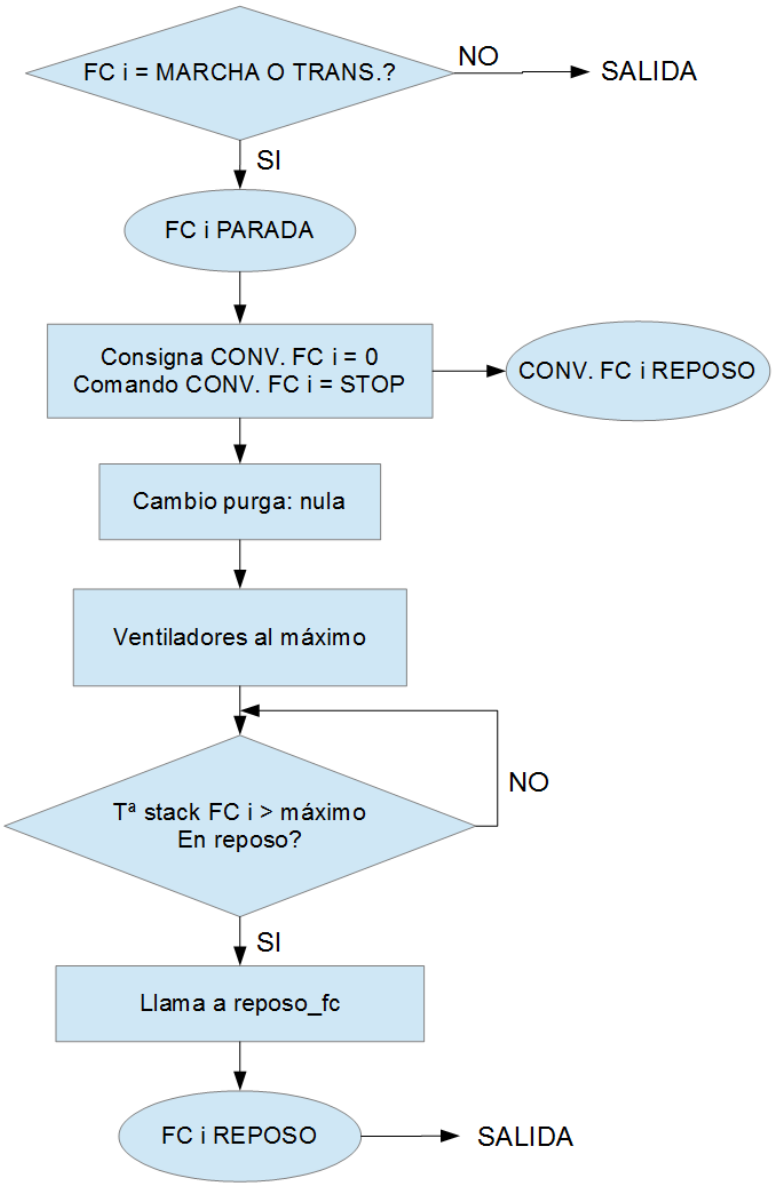

Figura 4.2 Función stop_fc.

4.10.2 Función stop_fc

Pasa del estado de MARCHA o TRANSITORIO al de REPOSO, mediante una PARADA NORMAL. En la Figura 4.2 se incluye el diagrama de flujo.

4.10.3 Función stop_emerg_fc

Realiza una PARADA DE EMERGENCIA poniendo a la pila en estado de ERROR. Esta función, al contrario que las demás, es llamada desde algún hilo distinto de los hilos de control de pilas como puede ser el proceso principal o el hilo de errores ante situaciones de error. En la Figura 4.3 se incluye el diagrama de flujo.

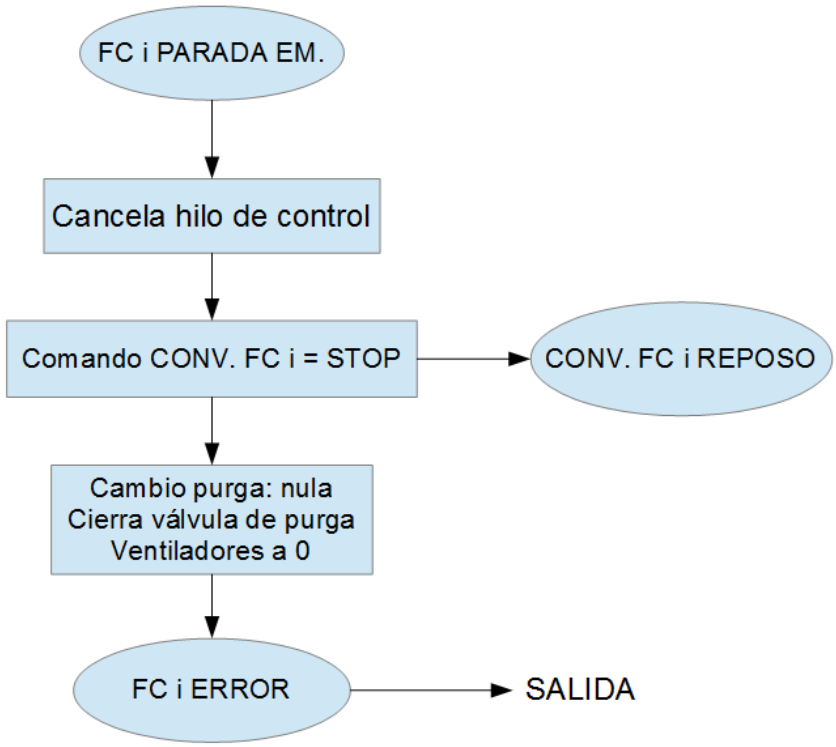

Figura 4.3 Función stop_emerg_fc.

4.10.4 Función reposo_fc

Pasa al estado de REPOSO con condiciones inciales tras un inicio o un estado previo de ERROR. En la Figura 4.4 se incluye el diagrama de flujo.

4.10.5 Función control_marcha

Realiza, para una iteración del hilo, el control sobre los ventiladores y las válvulas de purga en función de la corriente que en ese momento está aportando la pila. Cambia el estado a TRANSITORIO si se dan las condiciones para ello. En la Figura 4.5 se incluye el diagrama de flujo.

4.10.6 Función cambia_purga

Esta función es llamada por las funciones de cambio de estado o de control de las pilas para cambiar los parámetros de la purga de una pila.

4.11 El hilo control_purgas

Este hilo soporta las interrupciones producidas por los temporizadores que abren las válvulas de purga y comprueba en cada ciclo si hay peticiones de cambio en el intervalo de purga.

- Inicialización:

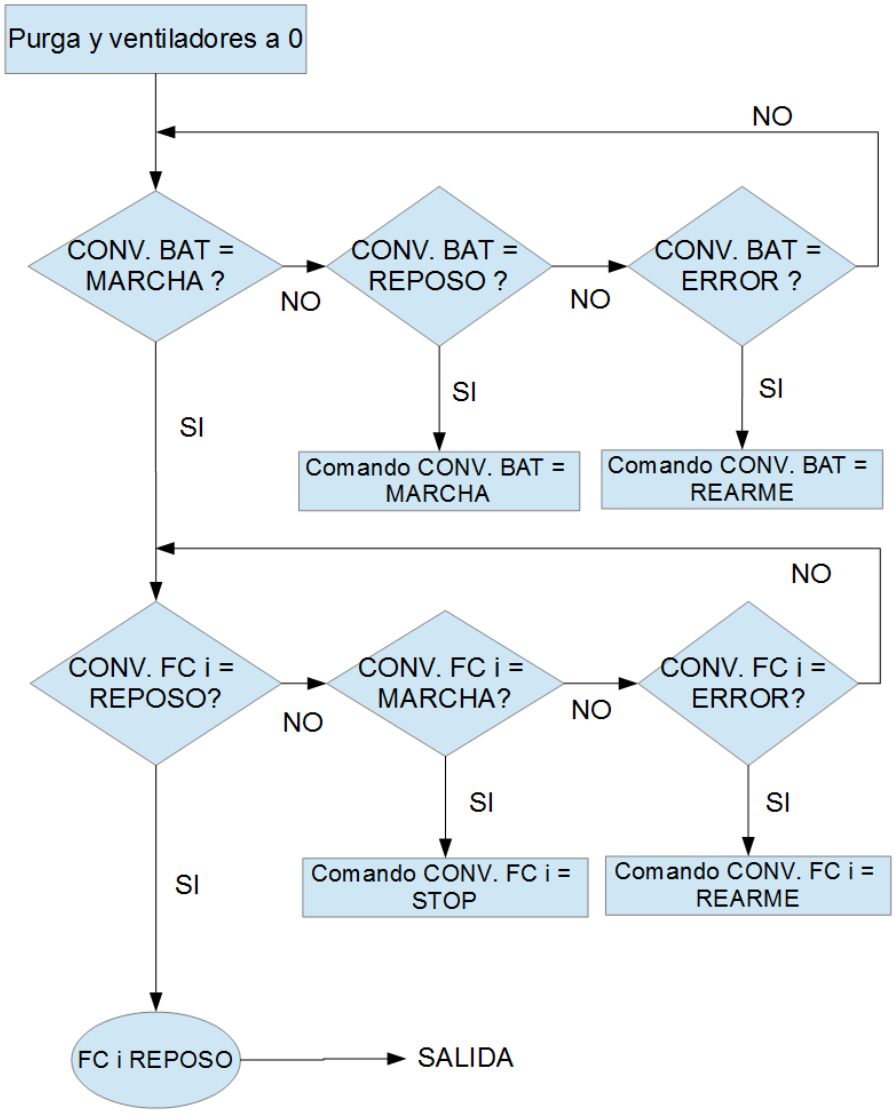

Figura 4.4 Función reposo_fc.

1. Define y arma los dos temporizadores de intervalo de purgas.
2. Abre la válvula solenoide del depósito.
3. Activa los temporizadores.

- Bucle:

1. Para cada una de las dos variables que almacenan los valores de las purgas, comprueba el flag de cambio.
2. Si se ha cambiado el flag pidiendo cambio en la purga actual, se recoge el valor transcurrido del temporizador y se compara con el valor inicial para el intervalo pedido.
3. Si el nuevo valor es mayor o igual que el transcurrido, se aplica al temporizador como valor inicial la diferencia.

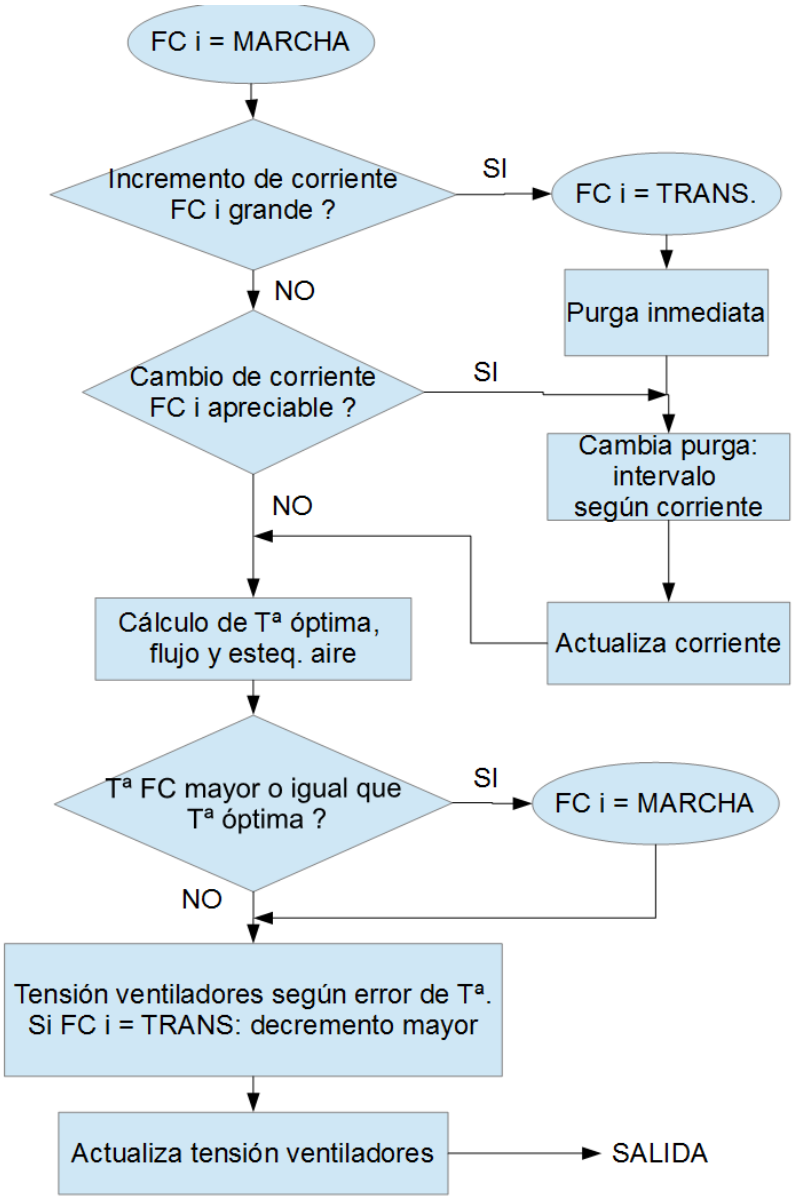

Figura 4.5 Función control_marcha.

4. Si es lo contrario, el nuevo valor será 0 (purga inmediata).
5. Se aplica al temporizador el intervalo deseado.

4.11.1 Manejador del temporizador de purgas

Es la función llamada cuando expira el intervalo de purga. Su cometido es abrir la válvula de purga correspondiente durante el tiempo definido en la variable y cerrarla, añadiendo luego una espera de seguridad antes de devolver el control al hillo de purgas.

4.12 El hilo de la gestión de potencia

Este hilo hace uso del estado de las pilas, de las medidas de los convertidores y de la señal del acelerador para implementar la estrategia de gestión de potencia. La condición primaria

es que el convertidor de baterías esté en estado de MARCHA. Si es así, entra en el bucle que consiste en los siguientes pasos:

1. Recoge los valores de tensión y corriente de las pilas, baterías y motor.
2. Calcula en función de ellos la potencia actual.
3. Llama a la función calcula_SOC para hallar el estado de carga actual de las baterías.
4. Llama a la función balance_pot para calcular las potencias previstas para la siguiente iteración.
5. Recoge de los comandos de la GUI cuál es la pila dominante.
6. Llama a la función actua_pilas para cambiar la consigna de corriente de las mismas.

4.12.1 Función calcula_SOC

Devuelve el valor porcentual de carga de las baterías en función del valor inicial almacenado por la GUI, de la tensión de las baterías y de la integración de la corriente en un tiempo determinado.

4.12.2 Función balance_pot

1. Recoge el valor del acelerador.
2. Lo compara al valor anterior almacenado.
3. En función de esta diferencia, realiza un incremento o decremento sobre el valor de la potencia demandada por el motor actualmente para predecir el valor siguiente.
4. Define la región de potencias posibles para las pilas alrededor del punto de potencias actuales.
5. Según el nivel de carga de la batería definido por intervalos de BAJA, MEDIA, ALTA o COMPLETA, decide el punto de potencias para las pilas previsto para la siguiente iteración.

4.12.3 Función actua_pilas

En función del valor de potencias devuelto por balance_pot, decide la corriente que debe aportar la pila dominante y la pila recesiva y manda los valores en forma de consigna a los convertidores de las mismas.

4.13 El hilo de errores

En cada iteración del bucle lee todas las variables globales. Llama a las funciones de comprobación de errores donde se comparan con respecto a los niveles definidos como límites. Si alguna devuelve un error de cierto nivel, llama a las funciones de actuación según corresponda.

4.13.1 Función h2_errores

Compara las variables de las medidas del circuito de hidrógeno.

4.13.2 Función conv_errores

Compara las variables de las medidas de los convertidores.

4.13.3 Función cvm_errores

Compara las variables de las medidas del CVM.

4.13.4 Función aire_errores

Compara las variables de las medidas de parámetros del aire de entrada a las pilas.

4.13.5 Función otros_errores

Compara otras variables como la activación de la seta, el *watchdog* o los contadores de errores de recepción en las comunicaciones.

4.13.6 Función actuacion_leve

Realiza la actuación ante errores leves, que consiste simplemente en indicar los errores para que se reflejen en la GUI.

4.13.7 Función actuacion_grave

Ejecuta los comandos necesarios para actuar ante errores graves en las pilas, como disminuir la corriente, aumentar la fuerza del ventilador o incrementar las purgas.

4.13.8 Función actuacion_critica

Realiza paradas de emergencia a las pilas y cancela hilos, como corresponde a la actuación definida ante errores críticos.

4.14 El proceso CVM

La comunicación con los módulos de adquisición del CVM se ha programado como proceso aparte debido a una limitación de la versión del sistema operativo QNX usada. Esta consiste en que es necesario la creación de un canal de notificaciones por cada puerto CAN y solo se acepta crear uno por proceso. Por tanto, el proceso CVM es lanzado como hijo por el proceso principal, comunicándose ambos mediante una cola de mensajes.

4.14.1 Hilo principal

- Inicialización:
  1. Crea la cola para mandar mensajes al proceso padre.
  2. Prepara el formato de mensaje CAN que va a solicitar las medidas a los módulos.
  3. Prepara lo necesario para la apertura del puerto CAN 2.
  4. Arranca el hilo trata_cvm para que trabaje paralelamente.
- Bucle:

1. Para cada canal de cada módulo escribe el mensaje de solicitud de medida y espera la respuesta. Una vez que la recibe comprueba que el formato es correcto y extrae los 4 caracteres con el dato, convirtiéndolo en una medida de tensión almacenada en un vector.
2. Una vez que tiene el vector de medidas completo manda a la cola de mensajes la estructura con las medidas y las variables resultantes de su análisis de ambas pilas.

4.14.2 Hilo trata_cvm

De forma paralela, para no sobrecargar el hilo principal que está pendiente de la comunicación CAN, este hilo toma las medidas almacenadas y obtiene resultados a partir de ellas. Produce la suma total de tensiones de celda, la media, la desviación típica, el máximo, el mínimo y la desviación máxima respecto a la media, así como qué pareja produce cada extremo. También, por parejas de celdas, su desviación respecto a la media.

4.15 Relación de archivos del código

- ecu.c: declaración de variables globales, main, recoge_cola, tempo_watchdog_manejador.
- cvm.c: main del CVM, trata_cvm.
- adq_datos.c: adq_datos_main, inicia_tadq, calibra_adq, cda_wr, cad_rd, dig_wr, dig_rd,interpreta_analog, termistores.
- com_can.c: can_main, com_conv, com_pccar, rx_pccar, tx_pccar, tempo_txpccar_manejador, adapta_var, manda_comando_cv, floattobytes, bytestofloat.
- control_fc.c: control_fc1_main, control_fc2_main, start_fc, stop_fc, stop_emerg_fc, reposo_fc, control_marcha, cambia_purga, control_purgas, tempo_interv_purga_manejador.
- gest_pot.c: gest_pot_main, calcula_SOC, balance_pot, actua_pilas, SOCinicial, interpolar.
- errores.c: errores_main, otros_errores, h2_errores, conv_errores, cvm_errores, aire_errores, actuacion_leve, actuacion_grave, actuacion_critica.
- Directorio include: comun.h, ecu.h, cvm.h, dscud.h, saj1000.h, canglob.h, canstr.h, canext.h, candef.h, com_can.h, adq_datos.h, control_fc.h, gest_pot.h, curvas_descarga.h, errores.h.
- Directorio lib: can.a, libdscud5.a.