DISEÑO Y PROGRAMACIÓN DEL PROTOCOLO DE COMUNICACIONES ENTRE UN ROBOT HEXÁPODO Y UN PC A TRAVÉS DE LÍNEA SERIE

David Lora Bartolomé
Josep M Porta Pleite
Enric Celaya Llover

ÍNDICE

1. INTRODUCCIÓN 1

1.1. ORGANIZACIÓN DE LA MEMORIA 1
1.2. PRESENTACIÓN DEL *GENGHIS II* 1

2. MOTIVACIÓN DEL PROYECTO 3

2.1. SITUACIÓN PREVIA Y PROYECTOS PARA EL GENGHIS II 3
2.2. LIMITACIONES 3
2.3. POSIBLES SOLUCIONES 3
2.3.1. Ampliación de memoria interna 3
2.3.2. Host externo 4
2.4. SOLUCIÓN ELEGIDA: ORIGEN DEL PROYECTO 4
2.5. DESCRIPCIÓN DEL PROYECTO 5

3. ESTUDIO PRELIMINAR 7

3.1. ESTRUCTURA EXTERNA E INTERNA DEL *GENGHIS II* 7
3.1.1. Actuadores 7
3.1.2. Sensores 7
3.1.3. Interruptores, jumpers y conectores externos 10
3.1.4. Alimentación de electrónica y motores 10
3.2. ELECTRÓNICA DEL *GENGHIS II* 10
3.2.1. Descripción 11
3.2.2. Microcomputador SERVO 12
3.2.3. Microcomputador STATUS 13
3.2.4. Microcomputador IR 14
3.2.5. Microcomputador MAIN 14
3.3. ENTORNO DE PROGRAMACIÓN DEL *GENGHIS II* 15
3.3.1. Descripción 15
3.3.2. Proceso de programación 15
3.3.3. Selección del micro a programar 15
3.3.4. Medio de comunicación Genghis II - Macintosh 16

4. OBJETIVOS Y RESTRICCIONES DE DISEÑO 17

4.1. FILOSOFÍA DE DISEÑO 17
4.2. OBJETIVOS 18
4.2.1. Medio Físico 18
4.2.2. Nivel Físico 18
4.2.3. Nivel de Enlace 19
4.3. REQUERIMIENTOS Y RESTRICCIONES 19
4.3.1. Requerimientos 19
4.3.2. Restricciones 19
4.4. PLATAFORMAS DE IMPLEMENTACIÓN 19

5. MEDIO FÍSICO 21

5.1. ESTUDIO DETALLADO DE LA SITUACIÓN ANTERIOR 21
5.2. SOLUCIÓN 22
5.3. CONCLUSIONES SOBRE EL MEDIO FÍSICO 23

6. NIVEL FÍSICO 25

6.1. PROTOCOLO DE NIVEL FÍSICO 25
6.2. ENTIDAD DE NIVEL FÍSICO DEL *GENGHIS II* 25
6.2.1. Implementación del protocolo 25
6.2.2. Estructura de funcionamiento 25
6.2.3. Interfície 27

6.3. ENTIDAD DE NIVEL FÍSICO DEL *PC* 29
6.3.1. Implementación del protocolo 29
6.3.2. Estructura y funcionamiento 29
6.3.3. Interficie 30
6.4. COMPATIBILIDAD ENTRE IMPLEMENTACIONES 32
6.4.1. Velocidad de funcionamiento 32
6.4.2. Paridad, bits de stop y número de bits por carácter. 33
6.5. PRUEBAS REALIZADAS Y CONCLUSIONES 33

7. NIVEL DE ENLACE 35

7.1. PROTOCOLO DE NIVEL DE ENLACE 35
7.1.1. Diseño de tramas 35
7.1.2. Diseño del protocolo 38
7.1.3. Limitaciones sobre Nivel Físico y independencia de niveles. 40
7.2. ENTIDAD DE NIVEL DE ENLACE EN EL GENGHIS II 40
7.2.1. Implementación del protocolo 40
7.2.2. Estructura y funcionamiento 42
7.2.3. Interficie 45
7.3. ENTIDAD DE NIVEL DE ENLACE EN EL PC 45
7.3.1. Implementación del protocolo 45
7.3.2. Estructura y funcionamiento 46
7.3.3. Interficie 47
7.4. PRUEBAS REALIZADAS Y CONCLUSIONES SOBRE NIVEL DE ENLACE 50

8. APLICACIÓN 53

9. ESTUDIO ECONÓMICO 57

10. CONCLUSIONES 59

BIBLIOGRAFÍA 61

DIRECCIONES DE INTERNET 61

ANEXO 1. ESTÁNDAR RS-232 63

ANEXO 2. MOTOROLA MC68HC11A2FN 65

Descripción 65
Puertos 66
Serial Peripheral Interface (SPI) 66
Conversor Analógico-Digital (A/D) 68
Interrupciones 69
Línea Serie 70

ANEXO 3. MODOS DE DIRECCIONAMIENTO Y CONJUNTO DE INSTRUCCIONES 73

ANEXO 4. ENSAMBLADOR DE LA FAMILIA *MC68HC11* EN EL ENTORNO *LISP* 81

ANEXO 5. CÓDIGO FUENTE DEL MICRO MAIN 83

ANEXO 6. CÓDIGO FUENTE DEL PC 95

ANEXO 7. CÓDIGO FUENTE DE LA APLICACIÓN 121

ANEXO 8. PROBLEMAS CON EL SERVO 139

ANEXO 9. INDICACIONES DE USO DEL PROTOCOLO 149

1. INTRODUCCIÓN

1.1. ORGANIZACIÓN DE LA MEMORIA

La memoria se ha estructurado por secciones que reflejan el orden en que se ha ido realizando el proyecto. Se empieza por una presentación del robot hexápodo y sus orígenes. Sigue una descripción de la situación previa y la motivación del proyecto, para después pasar a un estudio preliminar de los elementos disponibles. A continuación se concretan los objetivos que se marcaron inicialmente para la realización del proyecto, junto con los requerimientos y restricciones que se presentaban. Seguidamente se describe el proceso de diseño y las diferentes etapas realizadas, para finalizar con un estudio económico y conclusiones sobre el trabajo realizado y el resultado obtenido.

Se ha optado por incluir los apartados o cuestiones muy técnicas y listados de los programas como anexos, de manera que una persona con conocimientos mínimos del tema o que sólo pretenda obtener una idea global del proyecto no se vea obligada a pasar por estos apartados si no lo desea.

1.2. PRESENTACIÓN DEL *GENGHIS II*

El *Genghis II*, que podemos ver en la figura 1.1, es un robot hexápodo que fue diseñado en el *Massachusetts Institute of Technology (MIT)* a finales de la década de los 80. En concreto fue desarrollado por el *Mobot Group (Mobile Robots Group)*, grupo perteneciente al *MIT Artificial Intelligence Laboratory (AIL)* y que realizó el diseño bajo la dirección del profesor *Rodney Brooks*, director del *AIL* e importante personalidad dentro del mundo de la inteligencia artificial y la robótica.

Este robot, como se puede deducir de su nombre, es la evolución de un primer diseño: el *Genghis*. Aquel primer diseño fue utilizado en diversos experimentos y estudios que lo hicieron muy conocido en el mundo de la robótica y la inteligencia artificial. Por otro lado fue el primer paso para el desarrollo de otros modelos (*Attila*, *Hermes*,…) que intentaron mejorar la estructura y la movilidad. Actualmente el *Genghis*, que podemos ver en la figura 1.2, se puede ver en el *Smithsonian Air and Space Museum* en *Washington*.

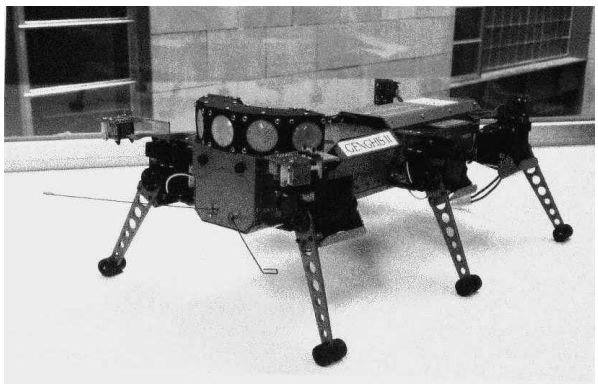

Figura 1.1*. Genghis II en el IRI.*

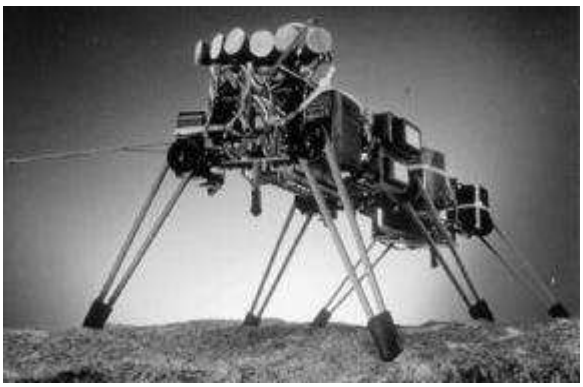

Figura 1.2. *Genghis* (predecesor del *Genghis II*).

Varias personas del *Mobot Group* crearon o impulsaron empresas relacionadas con el trabajo que habían realizado en el *MIT*. Entre estas empresas encontramos a *IS Robotics*, que se encarga, entre otras cosas, de comercializar robots aplicables a diferentes ámbitos como puedan ser la industria o la investigación. Uno de los fundadores y miembros actuales de *IS Robotics* es el propio *Rodney Brooks*.

La unidad del *Genghis II* objeto del proyecto pertenece al *Instituto de Robótica e Informática Industrial (IRI)* y fue adquirida a *IS Robotics* en el año 1994. Se comercializaron varias unidades más de este robot (dentro de ese mismo año éste fue la sexta unidad que se vendió) aunque actualmente parece ser que se ha retirado del mercado. En *España* sólo tenemos noticias de la existencia de otra unidad del *Genghis II*, además de la del *IRI*, en la *Universidad del País Vasco (San Sebastián)*.

El *IRI* es un centro perteneciente a la *UPC* [1] y al *CSIC* [2] especializado en robótica e informática aplicada a la industria en el que se realiza investigación básica y aplicada y es el lugar donde se ha desarrollado el proyecto durante los meses de marzo a junio de 1999.

Para más información se pueden consultar las direcciones sobre el *MIT*, *IS Robotics* y el *IRI* indicadas en la bibliografía al final de esta memoria.

---

[1] Universidad Politécnica de Cataluña
[2] Consejo Superior de Investigaciones Científicas

2. MOTIVACIÓN DEL PROYECTO

2.1. SITUACIÓN PREVIA Y PROYECTOS PARA EL GENGHIS II

Originalmente el *Genghis II* era un robot autónomo (utilizaba baterías tanto para la electrónica como para los motores) y programable mediante un lenguaje denominado *BL* [3]. El programa de usuario en *BL* era compilado y enviado al robot bajo un entorno *LISP* para *Macintosh*.

El robot fue adquirido con el objetivo de investigar y seguir avanzando en el campo de la inteligencia artificial. Principalmente se desea realizar proyectos de aprendizaje e investigación sobre arquitecturas de control reactivo. Hasta el momento se han desarrollado programas para hacer andar al robot sobre terreno abrupto y responder ante las situaciones que pueda encontrar, como por ejemplo obstáculos, precipicios, etc…

2.2. LIMITACIONES

Las tareas que se quieren llevar a cabo con el *Genghis II* (especialmente las de aprendizaje) requerirán de gran capacidad de memoria y cálculo para un funcionamiento óptimo. Esto, con la situación previa al proyecto, era un problema ya que el robot estaba limitado en cuanto a memoria (sólo permitía 32 Kb. de memoria para el programa de usuario) y en cuanto a capacidad de cálculo (los *micros* [4] del robot funcionan a una frecuencia de reloj de tan solo 2 MHz.). La situación originaria del robot imposibilitaba sobretodo el desarrollo de tareas de aprendizaje y por tanto era necesario buscar una solución.

2.3. POSIBLES SOLUCIONES

Las soluciones que se plantearon fueron dos, encaminadas en direcciones totalmente opuestas:

- Mantener la filosofía de programación y funcionamiento de robot como sistema autónomo donde se ejecutan los programas de usuario y simplemente aumentar la memoria interna.

- Abandonar la manera de funcionamiento y de programar el robot y convertirlo en un sistema periférico conectado a un *host* externo en el que se ejecutarían los programas en *BL*.

2.3.1. Ampliación de memoria interna

Esta solución consistiría en ampliar la memoria interna del robot comprando uno o varios módulos de expansión y mantener el resto de elementos y filosofía de funcionamiento.

Entre las ventajas de esta solución tenemos:

- Se puede mantener el mismo entorno de programación.

- Se pueden aprovechar los programas realizados hasta el momento.

- Se mantiene la autonomía del robot.

Como inconvenientes tenemos:

- No se aumenta la potencia de cálculo (sólo la capacidad de memoria).

[3] Behavior Language, diseñado por el profesor *Rodney Brooks*.
[4] Abreviatura de **microcomputadores**, que utilizaremos a partir de este momento en la memoria.

- Dificultad de ampliar la memoria del robot debido a que es un diseño cerrado.

- El espacio de memoria de los microcomputadores del robot es como máximo de 64K, con lo que no podremos ampliar la memoria a más de 64K para programas de usuario.

- Problema de escala: Incrementar la memoria o potencia de cálculo en el robot aumentaría el consumo, lo que podría necesitar de mayores baterías. Esto podría incrementar el peso en el robot de tal manera que fueran necesarios mayores motores, que incrementarían de nuevo el consumo requiriendo más baterías, etc….

2.3.2. *Host* externo

Esta solución consiste en conectar el robot a un *host* externo que se encargue de la ejecución de los programas de usuario. De esta manera el robot se comportaría como un periférico que recibiría órdenes del *host* externo y al cual enviaría el estado de sus sensores.

Las ventajas de esta solución son:

- Posibilidad de gran potencia de cálculo y capacidad de memoria dependiendo del *host* externo elegido (por ejemplo posibilidad de cálculos en coma flotante).

- Posibilidad de definir un lenguaje propio de programación diferente del *BL*.

Los inconvenientes son:

- Perdida de la autonomía del robot.

- Necesidad de desarrollar nuevas herramientas (comunicación entre robot y *host* externo, compilador para programas de usuario en el *host* externo, ...).

2.4. SOLUCIÓN ELEGIDA: ORIGEN DEL PROYECTO

Ante las dos soluciones que se plantearon la que pareció más adecuada para las tareas que se quieren realizar con el *Genghis II* fue la del *host* externo, aunque el gran inconveniente de esta opción era la pérdida de autonomía del robot.

Para llevarla a cabo se requería, por una parte, del diseño e implementación de un mecanismo de comunicación entre el robot y el *host* externo, y por otra, la creación de un nuevo compilador de *BL* o la creación de un nuevo lenguaje de programación (y compilador) para los programas de usuario. La primera parte es el origen y motivación de este proyecto de final de carrera y se puede considerar finalizada. La segunda parte esta por resolver y queda como posible proyecto de fin de carrera para un futuro (actualmente existe un primer prototipo de compilador de *BL*).

Pero como primer paso era necesario definir cual iba a ser el *host* externo encargado de la ejecución de los programas. Las posibilidades eran muy amplias pero interesaba un sistema con fuerte implantación en el mercado y sobretodo gran capacidad de cálculo.

Hoy día las máquinas con mayor implantación en el mercado de la informática en general son, sin duda, los *PC's*. Su precio es muy asequible en comparación con otro tipo de máquinas y la cantidad de *software* que se desarrolla para estos es inmensa.

La gran capacidad de cálculo en relación a su coste y la posibilidad de aprovechar los *PC's* disponibles en el IRI fue lo que decidió a elegir esta plataforma.

Como sistema operativo se eligió *DOS* debido a que permite sin problemas una programación directa del *PC* a bajo nivel (algo que no permite *W95* o *W98*), a su carácter genérico y especialmente a que permite un buen funcionamiento de aplicaciones en tiempo real.

El mecanismo elegido para la comunicación entre el robot y el *PC* fue la línea serie (estándar *RS-232*), debido a que el robot ya presentaba una conexión de este tipo.

2.5. DESCRIPCIÓN DEL PROYECTO

Este proyecto de final de carrera para la Ingeniería Técnica en Informática de Sistemas, consiste en el diseño e implementación de un protocolo de comunicaciones entre el robot hexápodo de la serie *Genghis II* y un *PC*.

Inicialmente se pide que el protocolo sea *full-duplex*, que permita la lectura de sensores desde el *PC* cada vez que haya nuevos datos disponibles en el robot; control en tiempo real mediante el envío de órdenes a los actuadores; ejecución en paralelo de un programa de usuario en el *PC*; que además sea configurable, de manera que se permita leer sólo los sensores del robot que se deseen; rápido y transparente al usuario.

También se desea la implementación de una interficie de adquisición de datos de los sensores del robot y que permita el control de sus actuadores en tiempo real, y cómo no, la documentación de todo el trabajo realizado.

3. ESTUDIO PRELIMINAR

En este apartado se realiza un estudio de cuál era la situación del *Genghis II* previa al inicio del proyecto, su estructura interna y externa, funcionamiento y el entorno de programación que se utilizaba. Con esto se pretende dar una base sólida que aporte el conocimiento suficiente sobre el funcionamiento del robot, para una comprensión correcta de los siguientes apartados que tratan sobre el diseño del protocolo de comunicaciones.

3.1. ESTRUCTURA EXTERNA E INTERNA DEL *GENGHIS II*

3.1.1. Actuadores

El *Genghis II* presenta tres tipos de actuadores: motores, *leds* y *buzzer*.

- Motores: Hay un total de doce motores, dos por pata, que dotan a cada extremidad de dos grados de libertad. Los motores que llamaremos *alfa* permiten el movimiento horizontal y los motores que llamaremos *beta* permiten el movimiento vertical de la pata. Los motores se pueden encontrar en la articulación de cada pata. Cada motor puede ir desde la posición 0 a la posición 63 (estos valores se traducen internamente a pulsos de unos y ceros) pero hay que tener cuidado con las posiciones extremas ya que en éstas algunos motores chocan con la carcasa del robot. Los motores están montados de forma diferente de manera que hay casos en que la posición más alta para una pata (controlada por un motor *beta*) es la 0 y en otros casos es la 63 (igual para motores *alfa*). Internamente la orden a un motor determinado se indica mediante un byte con un valor dentro del rango indicado.

- *Leds*: Hay un total de 16 *leds* situados en los laterales de la parte anterior del *Genghis II* (8 a cada lado), entre las patas delanteras y las intermedias. No tienen ninguna función específica y se pueden utilizar para lo que se desee (por ejemplo para indicar el nivel de baterías). El valor de los *leds* puede ir desde 0 (todos apagados) hasta 255 (todos encendidos) con lo que el valor que se indica en binario determina cuales se encienden (bits a uno) y cuales se apagan (bits a cero). Internamente la orden a los *leds* se indica mediante un byte.

- *Buzzer*: El *buzzer* es un zumbador que permite al robot emitir sonidos en un rango de frecuencias determinado. El valor a enviar al *buzzer* puede ir de 0 (buzzer desactivado) a 255. Internamente la orden a este actuador se indica mediante un byte.

3.1.2. Sensores

El *Genghis II* presenta un total de 9 tipos de información de sensores: fuerza de motores, posición de tacto, fuerza de tacto, infrarrojos, *piros*, inclinómetro, *whiskers* (antenas táctiles), nivel de batería de electrónica y nivel de batería de motores.

A continuación se detalla la función de los sensores y la información que proporcionan.

- Fuerza de motores: Cada uno de los doce motores proporciona un valor indicativo de la fuerza que está realizando dicho motor, de manera que se dispone de un total de doce valores de fuerza de motores. Este valor de fuerza corresponde en realidad al consumo que realiza el motor y los valores pueden ir de 0 (no realiza fuerza) a 14 (fuerza máxima), internamente el valor de fuerza de un motor ocupa un byte.

- Posición de tacto: El robot presenta en su parte inferior (lo que podríamos llamar "vientre") tres sensores de posición que proporcionan un valor determinado dependiendo del punto del sensor en el que se realiza presión. Cada sensor proporciona valores independientes del resto, por tanto hay un total de 3 valores de posición de tacto. Estos valores pueden ir de 0 (no se detecta presión), pasando por 1 (presión en uno de los extremos del sensor), hasta 15 (presión en el extremo contrario). Internamente el valor de posición de tacto de cada sensor ocupa medio byte. Estos sensores no están montados con la misma orientación en todos los casos con lo que

efectuar presión en el extremo anterior de un sensor de fuerza, puede dar un valor de 1 en unos casos y 15 en otros.

- Fuerza de tacto: Los mismos sensores de posición de tacto también proporcionan un valor que indica la fuerza de la presión que están detectando. Hay que tener las mismas consideraciones en cuanto a la orientación y número de sensores comentadas anteriormente. Los valores de cada sensor son independientes del resto y por tanto hay un total de tres valores de fuerza de tacto que pueden ir de 0 (no hay fuerza) a 14 (fuerza máxima). Internamente el valor de fuerza de tacto de cada sensor ocupa medio byte.

- Infrarrojos: El *Genghis II* dispone de un total de cuatro sensores de infrarrojos activos (emisores y receptores): dos delanteros (uno sobre cada uno de los dos motores delanteros) y dos traseros (uno sobre cada uno de los dos motores traseros). Internamente hay un único valor de infrarrojos de cuatro bits en el que se combina la información de los cuatro sensores (un bit para cada uno) con lo que los valores pueden ir de 0 a 15. A continuación se indica la correspondencia de bits:

  - Bit 3: sensor trasero izquierdo.
  - Bit 2: sensor trasero derecho.
  - Bit 1: sensor delantero izquierdo.
  - Bit 0: sensor delantero derecho.

- *Piros*: Los *piros* son sensores de infrarrojos pasivos (sólo receptores) que sirven para detección de calor y que están situados en la parte delantera del robot, encima de la carcasa. Tiene un total de 5 sensores de este tipo, cada uno con una orientación diferente. Cada sensor proporciona un valor independiente del resto, que pueden ir de 0 a 200 aproximadamente. Internamente el valor de cada sensor de este tipo ocupa un byte.

- Inclinómetro: El inclinómetro es un sensor interno que indica la inclinación longitudinal que presenta el robot en un momento determinado (no hay sensor de inclinación lateral). Los valores que proporciona pueden ir de 0 a 200 (el máximo normalmente no pasa de 170) e internamente ocupa un byte.

- *Whiskers*: Los *whiskers* son dos pequeños alambres que tiene el robot en la parte delantera a modo de antenas que le permiten detectar si choca con algo. Cada una de las antenas se activa, independientemente de la otra, por contacto (hay que hacer un poco de fuerza). Internamente el valor de cada sensor de este tipo ocupa un bit que en reposo está a uno y cuando se presiona la antena se pone a cero.

- Batería de motores: Indica el nivel de la batería de motores, puede ir de 0 a un valor máximo entorno a 100 cuando el robot está en reposo. Cuando el robot está en movimiento y debido al consumo realizado por los motores este valor baja. Este valor internamente ocupa un byte.

- Batería de electrónica: Indica el nivel de la batería de electrónica, puede ir de 0 a un valor máximo entorno a 80. Internamente ocupa un byte.

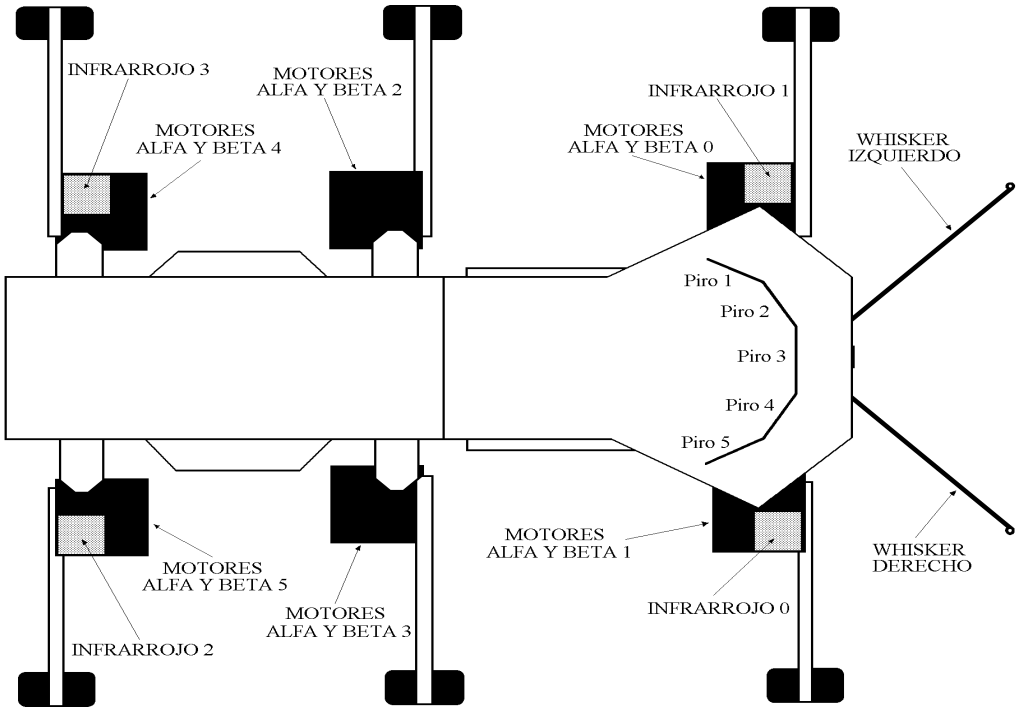

Figura 3.1. *Vista superior de sensores y actuadores del Genghis II.*

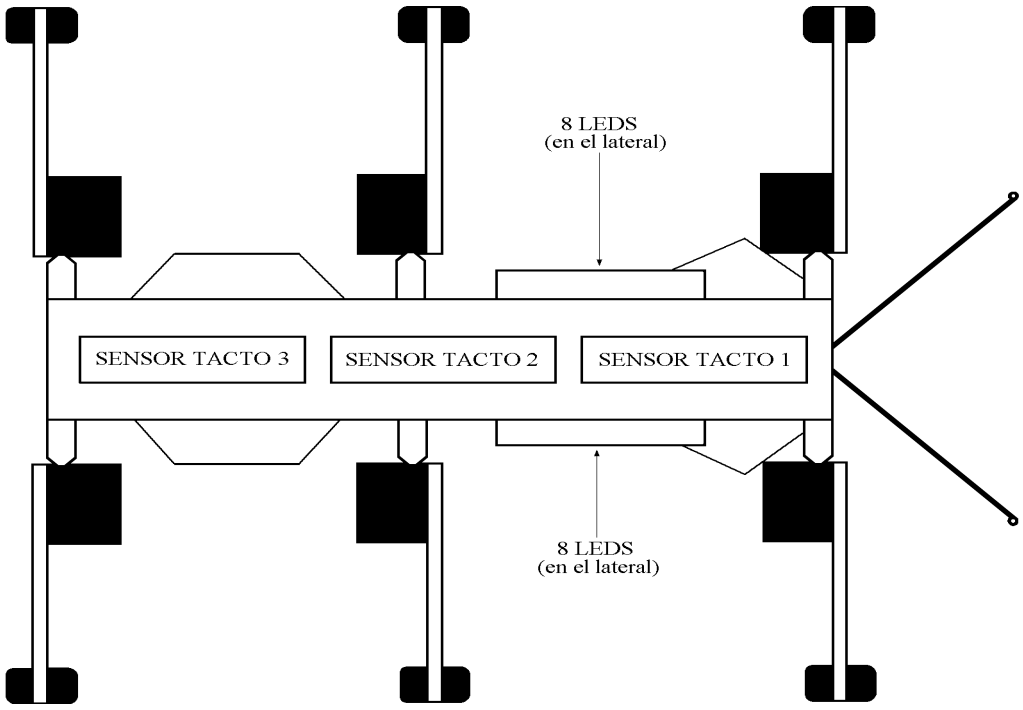

Figura 3.2. *Vista inferior de sensores y actuadores del Genghis II.*

3.1.3. Interruptores, *jumpers* y conectores externos

El robot presenta una serie de interruptores, *jumpers* y conectores externos que a continuación se describen con detalle.

- Interruptor de electrónica: Este interruptor permite activar la electrónica del robot o desactivarla.

- Interruptor de motores: Este interruptor permite activar los motores o desactivarlos.

- Interruptor de *reset*: Este pulsador permite hacer un *reset* del robot. Al pulsarlo se produce un *reset* de la electrónica.

- *Jumpers* de selección de *micro*: Permiten seleccionar qué *micro* de los cuatro posibles estará activo para programarlo (ver 3.3 para más información sobre estos *jumpers*).

- *Jumpers* de selección de *mode*: Permiten indicar "situaciones" al robot. Internamente se puede leer el estado de estos jumpers (a través de un registro) y por ejemplo dependiendo de su valor se puede actuar de una manera u otra en el programa de usuario.

- Conector de carga baterías: Permite recargar las baterías de motores que incorporaba originariamente el robot. Actualmente no se utiliza.

- Conector de línea serie: Es un conector de teléfono de 6 hilos. Tres de estos hilos son específicamente de línea serie: masa (GND), transmisión (TxD) y recepción (RxD). Dos más controlan el modo de funcionamiento del robot, *RUN* (ejecución normal de programas) o *DOWNLOAD* (carga de un programa en el *micro* que esté seleccionado mediante los *jumpers* de selección de *micro*). El sexto hilo sirve para hacer un *reset* al robot de igual modo que se puede hacer con el pulsador que presenta en la carcasa y que ya se ha comentado (ver 5.1. para más información sobre la función de estos hilos).

3.1.4. Alimentación de electrónica y motores

Originariamente la alimentación de la electrónica del *Genghis II* se realizaba mediante una pila alcalina de 9 voltios. Esta pila fue sustituida en el *IRI* por un transformador de 9 voltios, ya que la pila se gastaba muy a menudo y para cambiarla había que abrir el robot y manipular en su interior.

De igual modo, la alimentación de los motores inicialmente se realizaba mediante baterías de 4.8 voltios que iban en el interior del robot y que se podían recargar a través de un conector sin necesidad de abrirlo. Aun así, debido al hecho de tener que recargar las baterías a menudo (se gastan con facilidad), la alimentación de motores se sustituyó por una fuente de alimentación de 5 voltios.

A pesar de estos cambios el robot es capaz de funcionar en su estado original con pila para la electrónica y baterías para los motores, ya que se han mantenido los conectores que existían.

3.2. ELECTRÓNICA DEL *GENGHIS II*

A continuación se hace una descripción de la electrónica del *Genghis II*. Para cada uno de los microcomputadores que incorpora se indican primero su funcionalidad (software) y a continuación a qué puertos están conectados cada uno de los sensores que controla (hardware). Para más información se pueden consultar el anexo 2.

3.2.1. Descripción

La electrónica del *Genghis II* consta de 4 microcomputadores *MOTOROLA MC68HC11A2FN* (ver anexo 1 para más información sobre este *micro*) que denominaremos *SERVO*, *MAIN*, *IR* y *STATUS*. Estos *micros* están interconectados mediante un *bus* de *SPI* [5], de manera que colaboran entre sí para llevar a cabo las tareas de recogida de datos de los sensores y control de los actuadores.
A continuación se indica el esquema de interconexión de los *micros*, de manera que el *SERVO* actúa como *master*, dirigiendo la comunicación con el resto de *micros* (ver anexo 1 para funcionamiento de la *SPI*).

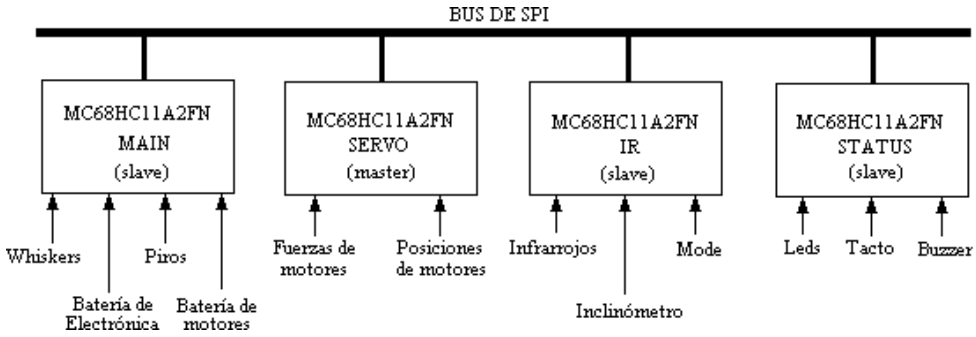

Figura 3.3. *Diagrama de los microcomputadores del Genghis II.*

Los cuatro microcomputadores presentan el mismo espacio de memoria a excepción del *MAIN* que incorpora un banco de memoria adicional de 32K de *EEPROM* [6]. A continuación se muestra un diagrama del espacio de memoria con el banco adicional del *MAIN* indicado en rayado. No obstante la distribución del espacio de memoria se puede modificar mediante la programación de ciertos registros internos de cada *micro*.

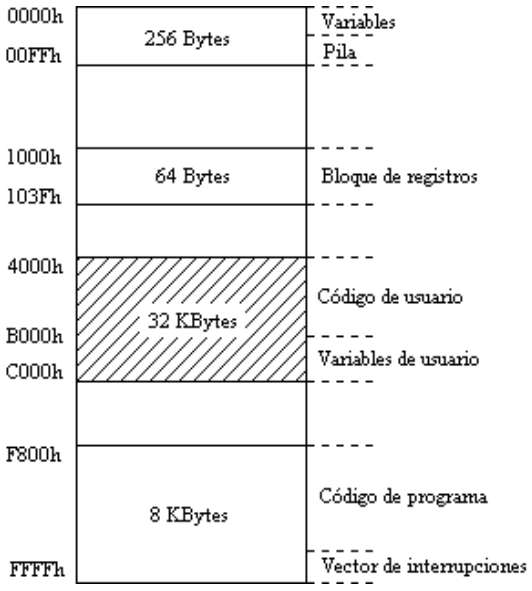

Figura 3.4. *Espacio de memoria de los micros del Genghis II.*

[5] Serial Peripheral Interface
[6] Electric Erasable Programable Read Only Memory

3.2.2. Microcomputador *SERVO*

Es el *micro* principal y controla el *timming*, la comunicación con el resto y los motores. En cuanto al *timming* controla que su trabajo, así como la comunicación con el resto, se realice en ciclos de 20 milisegundos. En cada ciclo envía órdenes a los motores (sólo si el nivel de baterías es suficiente), se comunica con el resto de *micros*, genera los pulsos de los motores para el siguiente ciclo a partir de las órdenes que recibe del *MAIN* y lee el valor de fuerza de éstos.

Este *micro* actúa como *master* en la comunicación con los otros, de manera que decide cuándo y con quién se comunica. El resto sólo actúan como *slaves* (esclavos) y contestan a las peticiones de comunicación del *SERVO*. A continuación se muestra el diagrama de funcionamiento del *SERVO*.

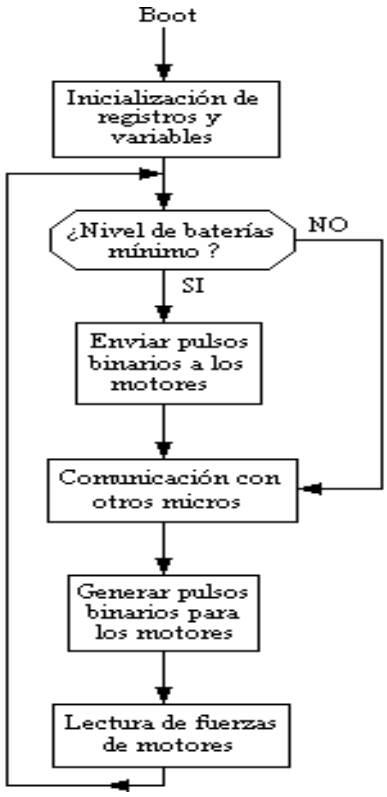

***Figura 3.5**. Diagrama de flujo del SERVO.*

En cada ciclo, al comunicarse con *STATUS* e *IR*, recibe los valores de los sensores controlados por esos *micros* y al comunicarse con el *MAIN* recibe órdenes procedentes de éste para los actuadores y a la vez le envía el estado de todos los sensores del robot controlados por él mismo o procedentes de *STATUS* o *IR*. En la figura 3.6 se indica el orden de estos datos en la comunicación entre *SERVO* y *MAIN* (como se puede ver se intercambian más bytes de los estrictamente necesarios).

SERVO → MAIN

0	Fuerza del motor alfa2	
1	Fuerza del motor alfa4	
2	Fuerza del motor beta4	
3	Fuerza del motor beta2	
4	Fuerza del motor beta3	
5	Fuerza del motor alfa5	
6	Fuerza del motor beta5	
7	Fuerza del motor alfa3	
8	Fuerza del motor alfa1	
9	Fuerza del motor beta1	
10	Fuerza del motor alfa0	
11	Fuerza del motor beta0	
12		
13		
14		
15		
16	*Mode*	Infrarrojos
17	Inclinómetro	
18	Posición de tacto2	Posición de tacto1
19	Posición de tacto3	Fuerza de tacto2
20	Fuerza de tacto1	Fuerza de tacto3

MAIN → SERVO

0	2 bits motores *on/off*	Orden motor alfa5
1		
2		
3		
4		
5	Orden al motor beta4	
6	Orden al motor alfa4	
7	Orden al motor beta5	
8	Orden al motor alfa1	
9	Orden al motor beta1	
10	Orden al motor alfa0	
11	Orden al motor beta0	
12	Orden al motor alfa3	
13	Orden al motor beta3	
14	Orden al motor alfa2	
15	Orden al motor beta2	
16		
17		
18	Orden a los *leds*	
19	Orden al *buzzer*	
20		

Figura 3.6.Orden del intercambio de datos entre SERVO y MAIN.

A continuación se detalla cómo están conectados los actuadores y sensores controlados por el SERVO a los puertos de este microcomputador:

- Puerto A: Bits 0-3: Entradas de fuerza de los motores.
  Bits 4-7: Selección de *slave* para comunicación por *SPI*.
- Puerto B: Bits 0-7: Salidas de control de los motores.
- Puerto C: Bits 0-7: Salidas de control de los motores.
- Puerto E: Bits 0-7: Entradas de fuerza de los motores.

3.2.3. Microcomputador *STATUS*

Este *micro* realiza el control de los *leds* y el *buzzer* a partir de las órdenes para estos sensores que recibe del *SERVO*. Por otro lado le envía la información de los sensores de posición y fuerza de tacto cuando éste lo requiere.

A continuación se detalla cómo están conectados los actuadores y sensores controlados por el *STATUS* a los puertos de este microcomputador:

- Puerto A: Bit 4: Salida para control del *buzzer*.
- Puerto B: Bit 0: Salida para control *led* delantero (el de ambos lados).
  Bits 1-6: Salida para control de *leds* intermedios (los de ambos lados).
  Bit 7: Salida para control de *led* trasero (el de ambos lados).
- Puerto C: Bit 0: Salida para seleccionar sensores de fuerza o de posición de tacto como entradas para el *A/D* [7] (conectado al puerto E). Con un cero las entradas del *A/D* corresponderán a valores de fuerza de tacto y con un uno a posiciones de tacto.
- Puerto E: Bit 1 (canal 1 *A/D*): Entrada analógica del sensor de tacto intermedio.

[7] Abreviatura de conversor Analógico-Digital

Bit 2 (canal 2 *A/D*): Entrada analógica del sensor de tacto delantero.
Bit 3 (canal 3 *A/D*): Entrada analógica del sensor de tacto trasero.

3.2.4. Microcomputador *IR*

El microcomputador *IR* no controla ningún actuador y simplemente envía al *SERVO* información del inclinómetro, los infrarrojos (controla en nivel de potencia con el que emiten) y el *mode* cuando éste lo requiere.

A continuación se detalla cómo están conectados los sensores controlados por el *IR* a los puertos de este microcomputador:

- Puerto A:
  Bit 3: Salida para selección de potencia de infrarrojos, máxima potencia.
  Bit 4: Salida para selección de potencia de infrarrojos, potencia intermedia.
  Bit 5: Salida para selección de potencia de infrarrojos, potencia intermedia.
  Bit 6: Salida para selección de potencia de infrarrojos, mínima potencia.

- Puerto B:
  Bit 0: Canal 0 de control de emisores de infrarrojos, delantero derecho.
  Bit 1: Canal 1 de control de emisores de infrarrojos, delantero izquierdo.
  Bit 2: Canal 2 de control de emisores de infrarrojos, trasero derecho.
  Bit 3: Canal 3 de control de emisores de infrarrojos, trasero izquierdo.

- Puerto C:
  Bit 0: Canal 3 de entrada de receptores de infrarrojos, trasero izquierdo.
  Bit 1: Canal 2 de entrada de receptores de infrarrojos, trasero derecho.
  Bit 2: Canal 1 de entrada de receptores de infrarrojos, delantero izquierdo.
  Bit 3: Canal 0 de entrada de receptores de infrarrojos, delantero derecho.

- Puerto E:
  Bit 0: Entrada de selector de modo, *jumper* 1.
  Bit 1: Entrada de selector de modo, *jumper* 2.
  Bit 2: Entrada de selector de modo, *jumper* 3.
  Bit 3: Entrada de selector de modo, *jumper* 4.
  Bit 5 (canal 5 *A/D*): Entrada analógica de inclinómetro.

3.2.5. Microcomputador *MAIN*

El microcomputador *MAIN* se encarga de contener el programa de usuario de manera que tras realizar las inicializaciones de registros necesarias salta a ejecutar dicho código.

Este *micro* no controla ningún actuador y accede directamente a los valores de los sensores de las baterías (electrónica y motores), los *piros* y los *whiskers* y por tanto ya dispone del valor de estos sensores sin necesidad de comunicarse con el *SERVO*. Como ya comentamos se comunica con dicho *micro* cada vez que aquel lo requiere (cada 20 milisegundos) y simultáneamente le envía las órdenes para el ciclo siguiente y recibe el estado de los sensores controlados por *SERVO*, *IR* y *STATUS* (se puede consultar la figura 3.4 para ver el orden del intercambio de datos en esta comunicación).

Otra de las funciones del *MAIN* es el control de la línea serie. Se le pueden enviar órdenes por línea serie para que ejecute ciertas acciones y se le puede pedir que envíe el valor de ciertas variables de usuario por línea serie.

A continuación se detalla cómo se conectan los sensores controlados por el *MAIN* a los puertos de este microcomputador:

- Puerto A:
  Bit 0: Entrada para el *whisker* izquierdo.
  Bit 1: Entrada para el *whisker* derecho.

- Puerto E:
  Bit 1 (canal 1 *A/D*): Entrada analógica para el sensor *piro* 2.
  Bit 2 (canal 2 *A/D*): Entrada analógica para el sensor *piro* 3.
  Bit 3 (canal 3 *A/D*): Entrada analógica para nivel de batería electrónica.
  Bit 4 (canal 4 *A/D*): Entrada analógica para el sensor *piro* 4.

Bit 5 (canal 5 *A/D*): Entrada analógica para el sensor *piro* 5.
Bit 6 (canal 6 *A/D*): Entrada analógica para el sensor *piro* 1.
Bit 7 (canal 7 *A/D*): Entrada analógica para el nivel de batería de motores.

3.3. ENTORNO DE PROGRAMACIÓN DEL *GENGHIS II*

3.3.1. Descripción

Para la programación del *Genghis II* se dispone de un entorno *LISP* para *Macintosh* que compila tanto programas de usuario en *BL* como programas en ensamblador del *MC68HC11*.

El Macintosh se puede conectar a través de la línea serie al robot mediante un cable que incorpora para dicho propósito y que incluye una caja que permite poner al robot en modo *RUN* (ejecución de programas) o modo *DOWNLOAD* (carga de programas). Mas tarde se hablará en detalle de este medio de comunicación entre el Macintosh y el robot.

3.3.2. Proceso de programación

Los pasos para la compilación de un determinado programa y envío de este programa al robot son los siguientes:

1- Cargar en el Macintosh el compilador *LISP*. A partir de este momento entraremos dentro del entorno de programación de *LISP* que nos permitirá cargar programas en memoria para compilarlos y enviarlos por línea serie al robot.
2- Cargar el entorno para compilación de programas para el *MOTOROLA MC68HC11*.
3- Cargar el fichero en *BL* o el fichero en *LISP* con instrucciones del *MC68HC11* dependiendo de si se quiere enviar un programa de usuario o un programa para el código de uno de los *micros*.
4- Poner la caja del cable de comunicación en modo *DOWNLOAD*.
5- Elegir en el robot, mediante los *jumpers* disponibles, el *micro* que se desea programar.
6- Ejecutar el comando correspondiente que se encarga de compilar y/o ensamblar el programa en *BL* o en *LISP* con instrucciones del *MC68HC11*.
7- Ejecutar el comando correspondiente para enviar por línea serie el código del programa correspondiente. Justo antes de ejecutar este comando hay que pulsar el botón de *reset* de la caja del cable de comunicación para que el robot y el *Macintosh* se sincronicen.
8- Esperar a que se indique por pantalla que la transferencia de datos se ha realizado correctamente. Es lenta ya que se realiza a 1200 baudios.

Si se han seguido correctamente los pasos anteriores ya tendremos el programa en la memoria del *micro* elegido.

3.3.3. Selección del *micro* a programar

Como ya comentamos anteriormente existen unos *jumpers* que nos permiten seleccionar uno de los *micros* para programarlo con el *Macintosh* a través de la línea serie. Hay un total de 5 *switchs* de los que solo se utilizan 4. A continuación se indica el número de *switch* y el *micro* al que corresponde:

1- *SERVO*
2- *MAIN*
3- *IR*
4- *STATUS*

Para seleccionar uno de los micros el *switch* correspondiente debe estar en posición de *ON* (posición alta) y el resto de *switchs* en posición de *OFF* (posición baja).

3.3.4. Medio de comunicación *Genghis II - Macintosh*

Como se ha comentado anteriormente para conectar el *Genghis II* y el *Macintosh* se dispone de un cable de línea serie especial. En primer lugar hay una caja, que permite controlar el modo de funcionamiento del robot y hacer un *reset*, de la que sale un cable de 6 hilos que va al robot y que en su extremo lleva un conector de teléfono de 6 vías. Además la caja incorpora un conector de línea serie de 25 *pins* donde se conecta otro cable que por un extremo lleva un conector de ese mismo tipo y por el otro lleva un conector de línea serie de Macintosh.

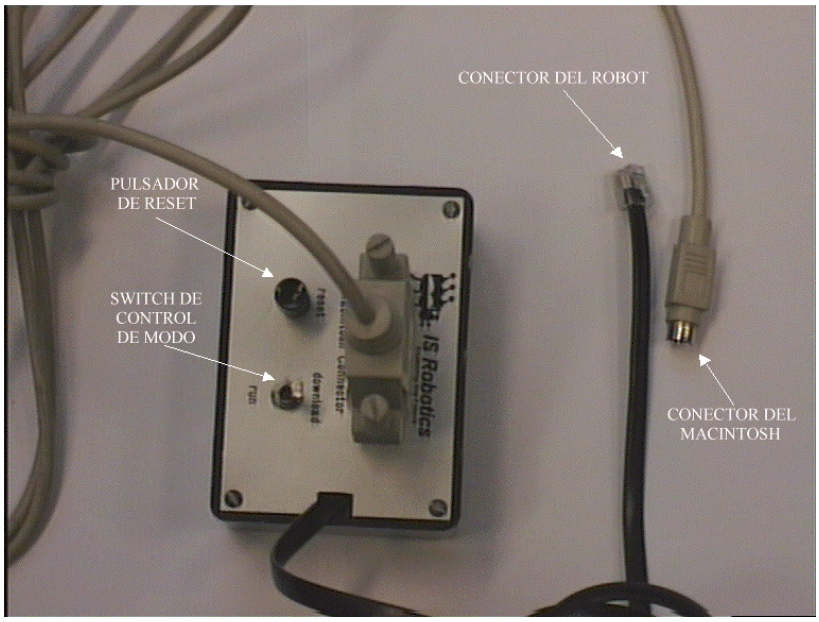

Figura 3.7. *Medio de comunicación Genghis II - Macintosh*

La caja consta de un *switch* que controla el modo de funcionamiento del robot a través de dos de los seis hilos y un pulsador que permite hacer un *reset* del robot a través de otro de los hilos. Los otros tres hilos son específicos de línea serie y no son modificados al pasar a través de la caja.

Cuando el *switch* está en posición de *RUN* el robot funciona ejecutando los programas de cada *micro* y cuando está en posición de *DOWNLOAD* espera a que se le envíe un programa por línea serie para cargarlo en la memoria del *micro* que esté seleccionado en ese momento.

4. OBJETIVOS Y RESTRICCIONES DE DISEÑO

Después del estudio de la situación previa y conocer el funcionamiento del robot y su entorno, es el momento de definir nuestros objetivos y estudiar que restricciones se nos presentan. No olvidemos que el objetivo final del proyecto es diseñar e implementar un protocolo de comunicaciones entre el *Genghis II* y un *PC* bajo *DOS*.

4.1. FILOSOFÍA DE DISEÑO

Para el diseño e implementación del protocolo se ha decidido seguir una filosofía *OSI* [8]. Esto quiere decir que estructuraremos la comunicación entre el robot y el *PC* en diferentes niveles independientes entre sí, de manera que cada nivel accederá al nivel inmediatamente inferior a través de la interficie ofrecida por éste y añadiendo nuevos servicios a los ofrecidos por el nivel inferior, que se ofrecerán al nivel inmediatamente superior a través de una determinada interficie de acceso. Esto ofrece la ventaja de que por la independencia de niveles podremos cambiar un nivel determinado sin necesidad de retocar los otros, siempre que mantengamos la misma interficie de acceso. En la figura 4.1 se puede observar la estructuración por niveles de la Torre OSI:

Figura 4.1. *Torre Osi*

En nuestro caso necesitaremos solo 3 niveles:

Un primer nivel que llamaremos *Medio Físico* que será el sistema físico encargado de interconectar los dos sistemas remotos (en nuestro caso robot y *PC*) y que será necesario construir.

Un segundo nivel que llamaremos *Nivel Físico* que será el encargado del intercambio de caracteres entre el robot y el *PC*. Los caracteres que se intercambian no tienen ningún significado específico a este nivel. Corresponde más o menos al nivel 1 de la torre *OSI*.

[8] Open Systems Interconection, define un modelo de referencia estructurado por niveles independientes para la implementación sistemas de comunicaciones (Torre OSI).

Un tercer nivel que llamaremos *Nivel de Enlace* que será el encargado del intercambio de tramas entre el robot y el *PC*. Las tramas a este nivel son conjuntos de caracteres que tienen un significado determinado por su valor o su posición respecto a otros. Corresponde más o menos al nivel 2 de la torre *OSI*.

Estos tres niveles son los que constituirán el sistema de comunicación entre los dos sistemas remotos y ofrecerán una determinada interficie de acceso a lo que podríamos considerar un nivel superior que denominaremos *Nivel de Aplicación*. Este nivel estará formado por el programa o programas equivalentes a los que antes del proyecto se denominaba "de usuario" en el robot y que implementarán una cierta funcionalidad del robot (que sea capaz mover una pata, caminar, etc…).

Según el modelo OSI un protocolo es, para que se entienda, el conjunto de reglas que hay que seguir para el intercambio de información entre dos entidades de un mismo nivel. Por lo tanto tendremos un protocolo de *Nivel Físico* y un protocolo *de Nivel de Enlace*. Para que no haya confusiones cuando queramos referirnos a un protocolo de un nivel concreto haremos mención a dicho nivel y si por el contrario hablamos solamente de "protocolo" sin hacer mención a ningún nivel, estaremos refiriéndonos al sistema de comunicaciones entre robot y *PC* en general.

En la figura 4.2 podemos ver un esquema de la estructura que se desea.

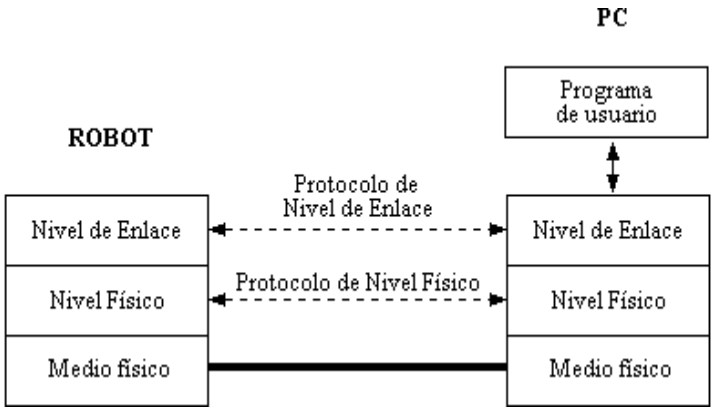

***Figura 4.2**. Estructura de comunicaciones deseada.*

4.2. OBJETIVOS

4.2.1. Medio Físico

En cuanto al *Medio Físico* se desea la construcción de un sistema físico que permita el intercambio de información entre el robot y el *PC*. Deberá respetar el tipo de conexión existente en el robot (tipo teléfono) y mantener (si es posible) la funcionalidad que presentaba el medio físico que existía previamente y que conectaba el robot al *Macintosh*.

4.2.2. Nivel Físico

Para el *Nivel Físico* básicamente nos interesa que sea capaz de intercambiar caracteres de manera fiable entre los dos sistemas remotos y ofrezca una interficie para enviar caracteres y poder consultar los caracteres recibidos, así como para consultar los posibles errores que se puedan producir. A este nivel se hace necesaria la construcción de *buffers* de entrada y salida que amortigüen las posibles diferencias de velocidad entre el flujo de caracteres entre el M*edio Físico* y la entidad de *Nivel Físico* y el flujo de datos entre la entidad de *Nivel Físico* y el nivel superior.

4.2.3. Nivel de Enlace

Deseamos, principalmente, un *Nivel de Enlace* capaz de intercambiar tramas que puedan llevar información de los sensores del robot y órdenes para sus actuadores de manera fiable. Que sepa recuperarse de posibles errores sucedidos en niveles inferiores o a su mismo nivel y que ofrezca una interficie sencilla para el envío de órdenes o consulta del valor de los sensores, entre otras posibilidades. Debería cumplir con los requerimientos que se piden (configurable, ágil para control del robot en tiempo real,…) además de otros que puedan surgir y adaptarse a las restricciones que puedan aparecer.

4.3. REQUERIMIENTOS Y RESTRICCIONES

A continuación haremos una enumeración de los requerimientos (cosas que se desean en cuanto a su funcionamiento) y restricciones (limitaciones impuestas por las características específicas del hardware o software del robot o del *PC*, o por el funcionamiento de cualquiera de estos dos sistemas) con que debemos contar. Estos requerimientos y restricciones pueden afectar a uno o varios de los niveles.

4.3.1. Requerimientos

1- Capacidad de recuperarse por *resets* en el robot o no existencia de éste al otro lado.
2- Respuesta tras una determinada orden con el nuevo estado de los sensores en un tiempo constante y mínimo.
3- Posibilidad de configuración para obtener sólo los sensores que se desean y hacer más ágil la comunicación.
4- Capacidad de permitir la ejecución de un programa de usuario en paralelo.
5- Detección de errores pero sin corregirlos ya que en el robot se generan datos de sensores suficientemente rápido.
6- Que no sea limitativo: poder disponer en cada ciclo de todos los sensores y todas las órdenes.

4.3.2. Restricciones

1- Los datos de los sensores se generan cada 20 milisegundos.
2- Las órdenes se envían a los actuadores cada 20 milisegundos, por lo que como máximo podremos enviar una orden por ciclo.
3- Hay un mínimo de 9 tipos de información de sensores y tres tipos de actuadores.
4- Rango de velocidades compatibles entre MOTOROLA MC68HC11 y línea serie del PC muy restringido (esto se estudiará en el apartado 6.4.1 de *Nivel Físico*).

4.4. PLATAFORMAS DE IMPLEMENTACIÓN

En cuanto a la implementación del protocolo en el *PC* se ha decidido utilizar el lenguaje de programación *C*, ya que es un lenguaje de alto nivel muy extendido que también permite programación a bastante bajo nivel. Se ha preferido éste y no otro debido a que las aplicaciones que utilizarán el protocolo se implementarán en este mismo lenguaje y así se facilitará la integración. En concreto se ha utilizado el compilador *Turbo C++* de *Borland* en su versión 3.0.

La implementación final del protocolo se desea en forma de una librería con sus correspondientes ficheros de cabecera. Se quiere así por facilidad de uso y eficiencia (en una librería el código ya está compilado).

Para la implementación en el robot ser utilizará el entorno de programación *LISP* para *Macintosh* ya existente. El lenguaje de programación será el ensamblador del *MOTOROLA MC68HC11* bajo *LISP*.

5. MEDIO FÍSICO

5.1. ESTUDIO DETALLADO DE LA SITUACIÓN ANTERIOR

Como ya se ha comentado en apartados anteriores, para comunicar el robot y el Macintosh existía un cable especial de seis hilos que incorporaba una caja. Esta caja permitía mediante un *switch* el control del modo de funcionamiento del robot (*RUN* o *DOWNLOAD*) y también se ofrecía la posibilidad de hacer un *reset* del robot mediante un pulsador.

No se dispone de ningún manual sobre esta caja y no hemos encontrado ningún tipo de información sobre su funcionamiento (es un diseño propio no público de *IS Robotics*). Ante esta situación se ha optado por deducir su funcionamiento interno observando la circuitería y comprobando las conexiones existentes al modificar la posición del *switch* y del pulsador.

El *switch* controlaba dos de los 6 hilos que llegaban al robot y el pulsador controlaba otro de los hilos. Los otros tres hilos del cable corresponden exclusivamente a línea serie y pasaban a través de la caja sin ser modificados. En la figura 5.1 se indica el esquema de conexión de los hilos dentro de la caja.

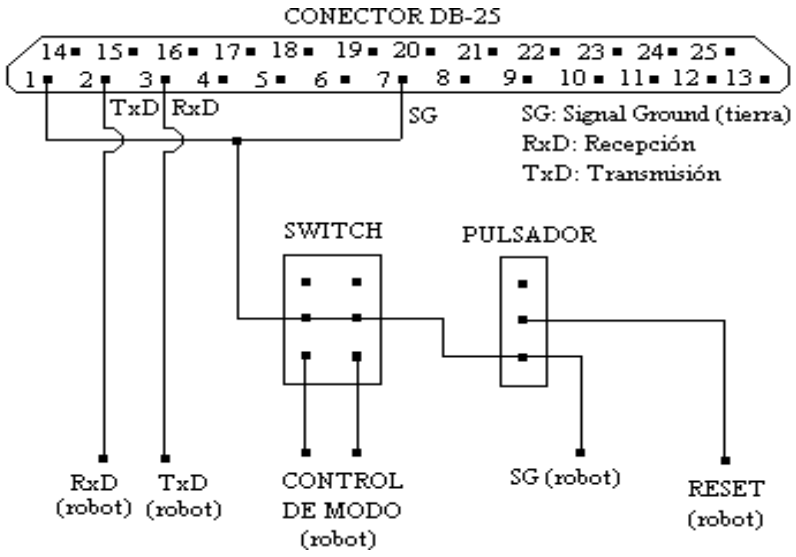

Figura 5.1. *Diagrama de conexiones de la caja.*

Cuando el *switch* está en posición *DOWNLOAD* en los dos hilos de control de modo de funcionamiento del robot se fuerzan cero voltios (se conectan al cable de masa de la línea serie procedente del *Macintosh*) y cuando está en modo *RUN* no se fuerza ningún valor concreto. En el caso del pulsador, si éste está activado, en el hilo que controla se fuerzan 0 voltios (se conecta al cable de masa igual que los hilos de modo) y cuando está en posición de reposo en el hilo no se fuerza ningún valor concreto.

A continuación se muestra un esquema de los estados del *switch* y el pulsador:

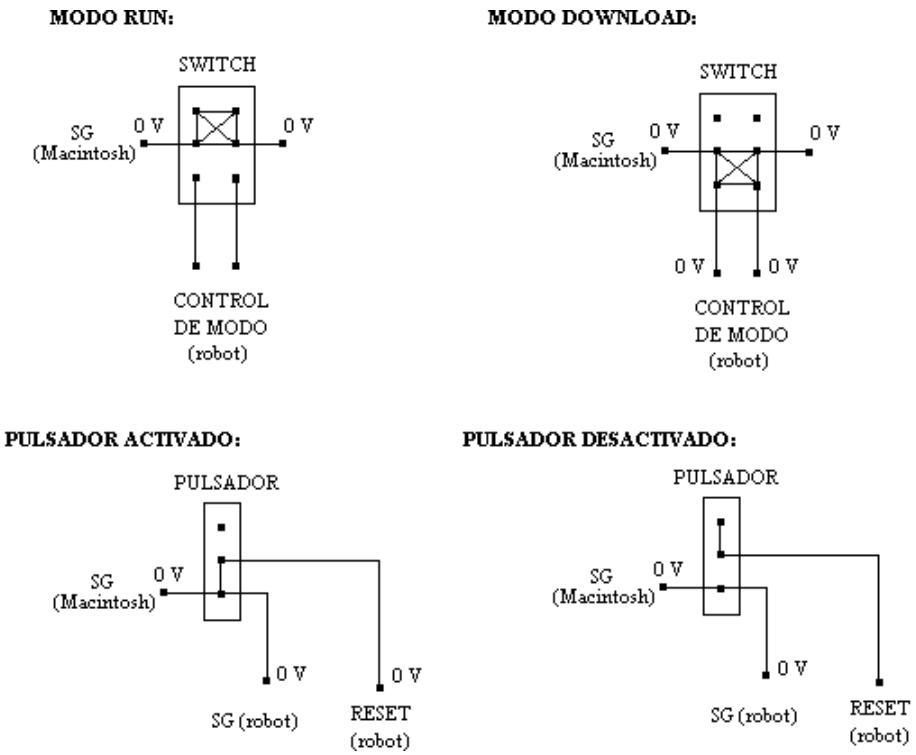

***Figura 5.2**. Estado de los hilos según la posición del switch y del pulsador.*

5.2. SOLUCIÓN

Para conectar el *Genghis II* y el *PC* a través de la línea serie queremos construir un nuevo medio físico que mantenga al máximo (si es posible) la funcionalidad que previamente ofrecía la caja. Podríamos haber mantenido el sistema físico existente, simplemente construyendo un pequeño cable con un conector de línea serie de 9 pins en un extremo, para conectarlo al PC, y un conector de 25 *pins* en el otro para conectarlo a la caja. En cambio se ha optado por construir el cable entero de nuevo para incorporar la funcionalidad de la caja por software, evitar posibles ruidos introducidos por ésta y ofrecer una mejor operatividad y que el robot no se vea obligado a arrastrar la caja al moverse.

Como medio físico hemos elegido un cable de 6 hilos por los siguientes motivos:

- Es suficiente para conectar los dos sistemas ya que no se espera demasiado ruido en la línea.
- Es barato.
- Es lo más parecido a lo que existía anteriormente por lo que no deberemos hacer demasiadas modificaciones.

Se podría haber elegido otros sistemas (como por ejemplo comunicación por radio, interesante porqué el robot hubiera mantenido parte de su autonomía, o fibra óptica, si el medio fuera especialmente ruidoso) pero hubieran requerido muchas modificaciones y serían mucho más caros.

Para mantener la funcionalidad de la caja se ha optado por conectar los hilos que controlaban el modo del robot y el que permitía el *reset*, a *pins* de la línea serie del *PC* que no se iban a utilizar y cuyo valor es controlable por

software. En concreto, los dos hilos de control del modo se han conectado al *pin RTS* [9] y el hilo de *reset* se ha conectado al *pin DTR* [10] (ver *pins* del registro de *control de módem (MCR)* en el anexo sobre la línea serie del PC). Así podemos poner un uno o cero lógico en los bits del registro *MCR* correspondientes a dichos *pins*, que se traducen en éstos a un voltaje positivo o negativo respectivamente. Este funcionamiento no es idéntico al de la caja ya que donde antes se forzaban cero voltios ahora se pone un valor negativo de voltaje. De todos modos la electrónica del robot interpreta un valor negativo de voltaje igual que cero voltios y, por tanto, así se consigue emular el comportamiento de la caja por software. Tras realizar las pruebas en los *PC's* disponibles en el *IRI* podemos asegurar que esto funciona correctamente para *chips* de línea serie que trabajen a 12.5 voltios y -12.5 voltios aproximadamente (como es el caso de los chips que montan los *PC's* actualmente).

En definitiva, se ha construido un cable que consta de 6 hilos, tres de los cuales son exclusivos de línea serie para emisión (*TxD*), recepción (*RxD*) y masa (*GND*) y van directamente del *PC* al robot. Los dos los hilos de control del modo de funcionamiento del robot se han conectado al *pin RTS* de la línea serie del *PC* de manera que cuando se activa (uno lógico) el robot se pone en modo RUN y cuando se desactiva (cero lógico) el robot se pone en modo *DOWNLOAD*. El hilo de control del *reset* se ha conectado al *pin* de *DTR* de la línea serie del *PC* de manera que cuando se desactiva (cero lógico) y se vuelve a activar (uno lógico) se produce el mismo efecto que con el pulsador de la caja y se produce un *reset* del robot.

El cable, en el extremo del robot, presenta un conector de teléfono de 6 vías y en el otro extremo tiene un conector de línea serie estándar tipo DB-9 (9 *pins*) hembra. En caso de que el conector del PC fuera de 25 *pins* se puede utilizar un adaptador que pase de 9 a 25 *pins*. En la figura 5.3 se muestra el diagrama de conexión de los hilos a cada uno de los conectores.

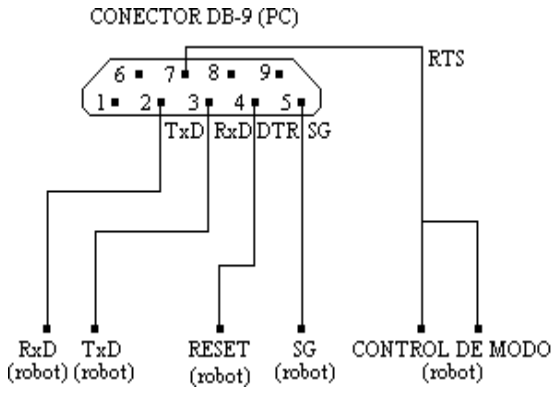

***Figura 5.3**. Esquema de conexión del nuevo medio físico.*

5.3. CONCLUSIONES SOBRE EL MEDIO FÍSICO

Hemos construido un cable de línea serie que comunica al *Genghis II* y al *PC* y mantiene la misma funcionalidad, pero por software, que ofrecía la caja del cable que conectaba anteriormente al robot y al *Macintosh*.

Se han realizado pruebas con *PC's* que incorporaban diferentes chips de línea serie (familias 8250 y 16550) y hemos comprobado que el intercambio de caracteres entre el robot y el PC se realiza de manera correcta en ambos sentidos en todos los casos. De igual modo, tanto el *reset* como el control de modo del robot por software mediante los *pins* de *control de módem* de la línea serie, funcionan correctamente.

[9] Request To Send
[10] Data Terminal Ready

6. NIVEL FÍSICO

6.1. PROTOCOLO DE NIVEL FÍSICO

El protocolo de *Nivel Físico* corresponde al estándar *RS-232* que ya viene parcialmente implementado en un *chip* especial en el *PC* e internamente en el *MOTOROLA MC68HC11*. El estándar, que es bastante amplio y está pensado para la comunicación con un módem, no sólo define el protocolo sino que también especifica los niveles eléctricos para su implementación. Estos chips, en cambio, no implementan todas las funcionalidades descritas en el estándar y no son 100 % compatibles entre sí.

Como breve descripción decir que se trata de una comunicación asíncrona, ya que no hay ningún tipo de señal de reloj para la sincronización. En la transmisión, el carácter se "serializa" y se envía como un bit de *start*, entre 5 y 8 bits de datos, un bit de paridad opcional y uno o varios bits de *stop*. Los sistemas que se desea comunicar deben estar configurados a la misma velocidad, número de bits de datos, paridad y número de bits de *stop* para que la comunicación sea posible.

El receptor, cuando detecta un bit de *start*, muestrea los bits de datos y paridad (si existe) a una velocidad determinada, hasta que detecta el bit de *stop*. Luego comprueba que el bit de paridad es correcto para los datos recibidos y acepta el carácter como válido o erróneo según el caso.

Para un estudio más detallado del protocolo *RS-232* ver el anexo 1.

6.2. ENTIDAD DE NIVEL FÍSICO DEL *GENGHIS II*

A continuación detallaremos cómo se implementa el protocolo de *Nivel Físico* en el *Genghis II* y cómo se ha diseñado e implementado.

6.2.1. Implementación del protocolo

Como vimos anteriormente, en el robot había cuatro microcomputadores *MOTOROLA MC68HC11A2FN* cada uno con una misión determinada. Nuestro objetivo es pasar los programas de usuario del robot a un *host* externo, por lo que el *micro MAIN* (era el que contenía el programa de usuario) queda sin ocupación. Por tanto la implementación del *Nivel Físico* se realizará en ese micro de manera que no se espera que haya que modificar ninguno de los demás.

El *MOTOROLA MC68HC11A2FN* ya incorpora un sistema de línea serie denominado *Serial Communications Interface (SCI)* que implementa un subconjunto de las funcionalidades definidas en el estándar *RS-232*.

Como breve descripción diremos que es un sistema asíncrono de comunicación que utiliza solo dos *pins* para emisión y recepción, *full-duplex*, con posibilidades de 8 o 9 bits de datos solamente, un único bit de *stop*, sin paridad y con la posibilidad de funcionamiento por interrupciones (para un estudio más detallado ver el anexo 2).

6.2.2. Estructura de funcionamiento

Como hemos dicho la implementación del *Nivel Físico* en el *Genghis II* se realizará en el microcomputador que denominamos *MAIN*. Este *micro* era el encargado de contener el programa de usuario y enviar órdenes al *SERVO* para el control de actuadores y recibir los datos de los sensores. Esta comunicación se realizaba cada vez que el *SERVO* lo requería (cada 20 milisegundos).

Para implementar el *Nivel Físico* hemos de tener en cuenta que el comportamiento descrito del *MAIN* respecto al *SERVO* se debe mantener, de manera que este micro además de ser capaz de controlar la línea serie para el

intercambio de caracteres con el *PC*, también debe ser capaz de comunicarse con el *SERVO* cada vez que éste lo requiera para realizar el intercambio de órdenes y valor de los sensores. También se espera que el micro deba hacer trabajo correspondiente a la implementación de *Nivel de Enlace* o lectura de los sensores que controla.

Todo esto nos lleva a la necesidad de una implementación por interrupciones para que el intercambio de caracteres por la línea serie no acapare el funcionamiento de la *CPU* y por tanto imposibilite hacer más trabajo. No entraremos en la explicación del funcionamiento del mecanismo de interrupciones en este punto, para más información sobre esto consultar [3].

La estructura que se ha diseñado consta de un *buffer* de entrada que contendrá los caracteres que vayan llegando por línea serie y otro de salida que contendrá los caracteres que se deseen enviar. Estos *buffers* sirven para amortiguar las posibles diferencias de velocidad del flujo de datos que pueda haber entre llegada de caracteres por línea serie y lectura de éstos o envío de caracteres y salida de éstos por línea serie.

Por las restricciones de comunicación con el *SERVO* cada 0,02 segundos (ver restricciones 1 y 2 del punto 4.3.2) no vamos a tener más de un conjunto de sensores u órdenes por ciclo. A la mayor velocidad posible a la que se podrá funcionar en el robot (10417 baudios) no será posible recibir o enviar más de 20 caracteres por ciclo. Por lo tanto se ha decidido implementar un tamaño de *buffer* de 64 bytes por ser un valor múltiplo de dos, por tanto más fácil de tratar a nivel de programación en ensamblador, y en principio suficientemente grande. De todas maneras este valor es fácilmente modificable hasta un máximo de 256 (se utilizan variables de tamaño byte para direccionar a los *buffers*), para tamaños de buffer mayores sería más complicado.

Con la estructura diseñada, cuando se quiere enviar un carácter por línea serie lo que se hace es escribirlo en el *buffer* de salida y cuando se quiere recibir un carácter de línea serie se lee del *buffer* de entrada (si hay algún carácter disponible). El mecanismo de interrupciones será el encargado de pasar los caracteres que llegan por línea serie al *buffer* de entrada y de coger caracteres del *buffer* de salida y pasarlos a la línea serie para ser enviados. En la figura 6.1 se muestra un esquema con la estructura del *Nivel Físico*.

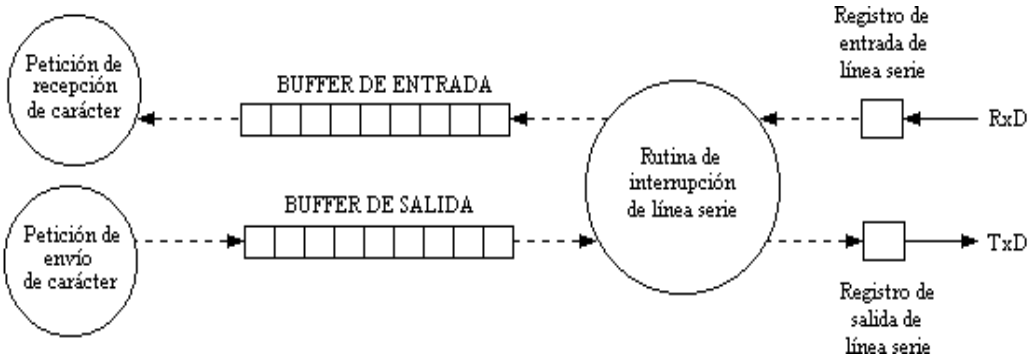

Figura 6.1*. Estructura del Nivel Físico del Genghis II.*

Aunque el tamaño de *buffer* debería ser suficiente para que los *buffers* no se colapsaran, podría ser que por diseño erróneo del programa que se encargue de enviar y recibir por línea serie (debido a que el programa escribe demasiado rápido en el buffer de salida o lee demasiado despacio del buffer de entrada) los *buffers* se llenaran. Por tanto se debe diseñar correctamente el programa que utilice los servicios ofrecidos por el *Nivel Físico* de manera que sea lo suficientemente ágil para leer caracteres recibidos en el *buffer* de entrada y no escriba demasiado rápido en el de salida. Ante una situación de colapso de un *buffer* el resultado es la pérdida del carácter recibido o que se deseaba enviar.

La parte principal de este nivel es el mecanismo de interrupciones, que se encarga de recibir caracteres por línea serie y guardarlos en el *buffer* de entrada y de leer caracteres del *buffer* de salida y enviarlos por línea serie. Las interrupciones se producen cada vez que se acaba de enviar un carácter por línea serie o cada vez que se recibe un nuevo carácter. Las interrupciones por recepción están siempre activas y las de transmisión sólo se activan cuando hay caracteres por enviar y se desactivan cuando ya no quedan caracteres en el *buffer* de salida. Ambos tipos de interrupciones comparten la misma rutina de interrupción cuyo diagrama de flujo se muestra en la figura 6.2. Cada vez que se activa comprueba si se ha producido error de recepción (se ignora el carácter recibido) o no (se lee el carácter recibido y se guarda en el *buffer* de entrada) y si se ha acabado de enviar un carácter (caso en el que se lee un nuevo carácter del *buffer* de salida y se envía por línea serie).

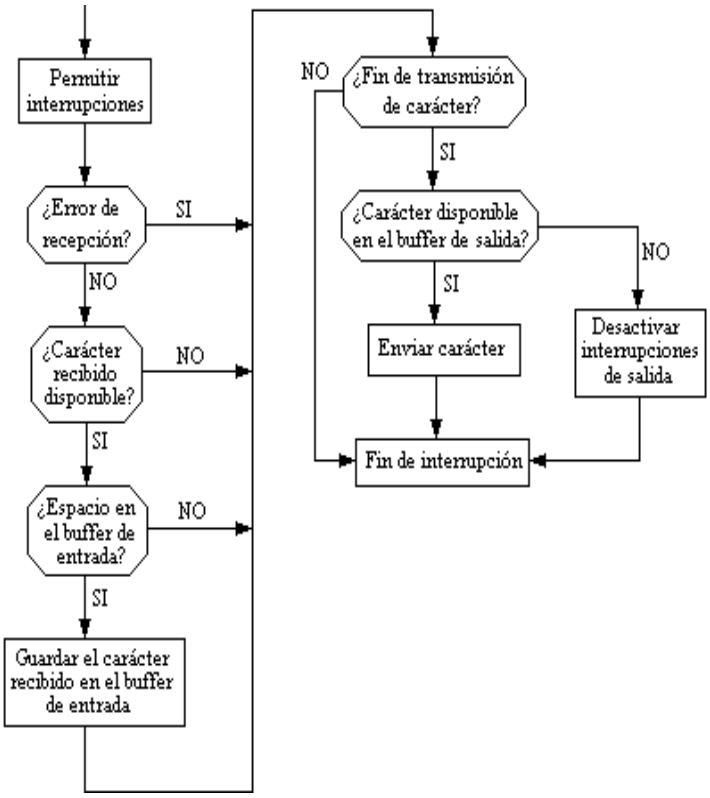

***Figura 6.2**. Diagrama de flujo de la rutina de interrupción en el robot.*

6.2.3. Interfície

La interficie de *Nivel Físico* en el *Genghis II* está compuesta principalmente por dos funciones: *EnviaCar* (permite escribir un carácter en el *buffer* de salida para ser enviado por línea serie) y *RecibeCar* (que lee un carácter del buffer de entrada en caso de que haya alguno disponible). También se puede consultar el tamaño de cualquiera de los dos *buffers* mediante unas variables: *InBuffer_Count* para el *buffer* de entrada (*InBuffer*) y *OutBuffer_Count* para el de salida (*OutBuffer*).

EnviaCar se encarga de escribir un carácter en el *buffer* de salida para ser enviado por línea serie. Hay que pasarle en el registro *acumulador A* (ver anexo 1 para información sobre los registros del *MC68HC11A2FN*) el carácter que se quiere enviar. Como resultado indica en la variable *CarEnv*, con un uno que el carácter se ha escrito correctamente en el *buffer* de salida y con un cero que el carácter no se ha podido escribir en el *buffer* de salida debido a que estaba lleno (por lo tanto no se puede enviar). En la figura 6.3 se muestra el diagrama de flujo de esta función.

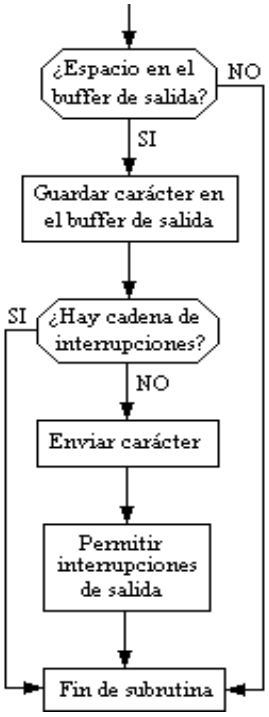

Figura 6.3. *Diagrama de flujo de EnviaCar*

RecibeCar se encarga de leer un carácter, procedente de línea serie, del *buffer* de entrada (en caso de que haya alguno disponible). Indica en la variable *CarEnv*, con un uno que se ha leído correctamente un carácter del *buffer* de entrada y con un cero que no había ningún carácter disponible en dicho *buffer*. En caso de haber leído algún carácter del *buffer* de entrada (*CarEnv*=1) el carácter se deja en el registro *acumulador A*, desde el que es accesible por el programa. En la figura 6.4 se muestra el diagrama de flujo de esta función.

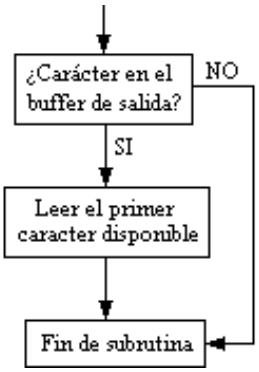

Figura 6.3. *Diagrama de flujo de RecibeCar*

6.3. ENTIDAD DE NIVEL FÍSICO DEL *PC*

6.3.1. Implementación del protocolo

En el PC la implementación del protocolo del standard *RS-232* se realiza mediante un chip especial. Dependiendo del tipo de ordenador puede pertenecer a la familia *i8250* (normalmente en *PC's* antiguos hasta el *i468*) o a la familia *i16550* (habitualmente en *PC's* nuevos a partir del *Pentium*). Para un estudio detallado de las características de estos *chips* consultar [3] o las direcciones de internet indicadas en el apartado correspondiente..

Como breve descripción, diremos que ambas familias son sistemas de comunicación asíncronos que utiliza tres *pins* para emisión y recepción. Incorporan *pins* para *control de flujo* activo, son *full-duplex* con posibilidad de 5 a 8 bits por carácter, bit de paridad seleccionable, uno o varios bits de *stop* y posibilidad de funcionamiento por interrupciones. Las dos familias se diferencian principalmente en que la familia *i16550* incorpora *buffers* internos para transmisión y recepción y la familia *i8250* no.

6.3.2. Estructura y funcionamiento

Uno de los requerimientos que teníamos que cumplir, en concreto el número cuatro del apartado 4.3.1, era que el protocolo (al menos en el *PC*) debía permitir la ejecución en paralelo de un programa de usuario. Por tanto es totalmente necesario implementar el *Nivel Físico* del *PC* por interrupciones, ya que sino no se podría cumplir este requerimiento.

La estructura del *Nivel Físico* en el *PC* es casi idéntica a la diseñada en el robot. Consta de un *buffer* de entrada para los caracteres que vayan llegando por línea serie y otro de salida que contendrá los caracteres que se deseen enviar. La función de estos *buffers* es la misma que ya se comentó para el *Nivel Físico* del robot.

La mayor velocidad de proceso que presenta el *PC* respecto al robot y limitaciones procedentes del *Nivel de Enlace* nos han llevado a elegir un tamaño de *buffer* mucho más grande que para el robot. En concreto en el *PC* tendremos *buffers* de 1024 bytes pero este parámetro es fácilmente modificable si se desea otro tamaño diferente.

Igual que en el robot, cuando se quiere enviar un carácter por línea serie lo que se hace es escribirlo en el *buffer* de salida y cuando se quiere recibir un carácter de línea serie se lee del *buffer* de entrada (si hay algún carácter disponible). El mecanismo de interrupciones es el encargado de pasar los caracteres que llegan por línea serie al *buffer* de entrada y de coger caracteres del *buffer* de salida y pasarlos a la línea serie para ser enviados. El esquema es idéntico al que ya vimos en la figura 6.1.

Aquí también podría suceder que los *buffers* se colapsaran por una mal diseño del programa que envía y recibe datos por la línea serie a través de este nivel y, de igual manera, ante una situación de colapso de un *buffer* el resultado es la pérdida del carácter recibido o que se deseaba enviar.

Las interrupciones de línea serie, encargadas de transferir caracteres entre los *buffers* de salida o entrada y la línea serie, se producen cada vez que se acaba de enviar un carácter o cada vez que se recibe uno nuevo. En el *PC* el mecanismo de interrupciones funciona de manera diferente que en el *MOTOROLA* y está gestionado por un *chip* especial. También tenemos una rutina de interrupción común para transmisión y recepción, cuyo diagrama de flujo se puede ver en la figura 6.4, pero al contrario que antes las interrupciones de línea serie están siempre activas (en el robot había que desactivar las de transmisión cuando no había más caracteres por enviar).

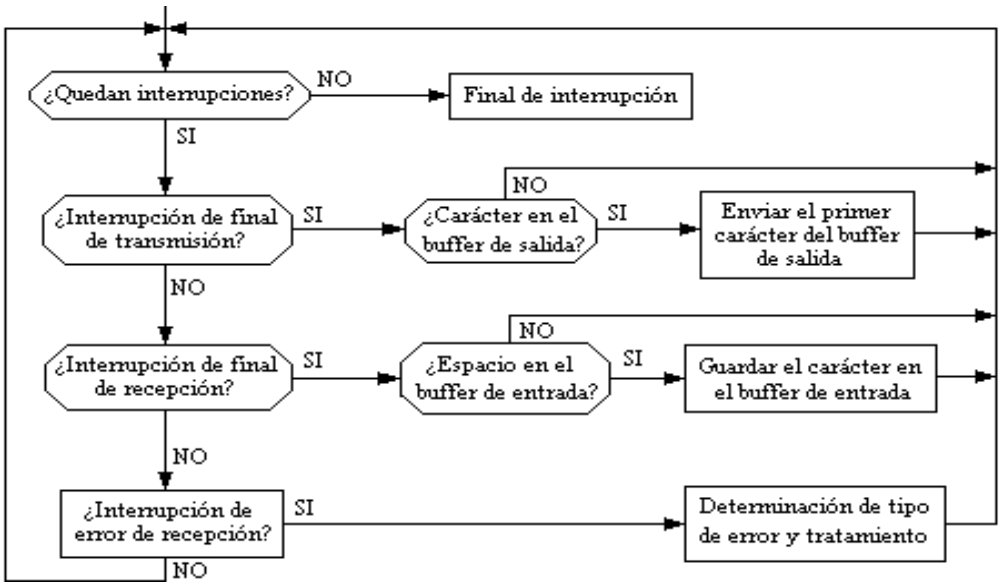

Figura 6.4. *Diagrama de flujo de la rutina de interrupción en el PC.*

Aprovechando las ventajas que nos ofrece la programación con un lenguaje de alto nivel hemos estructurado la implementación del *Nivel Físico* básicamente en dos módulos. Un primer módulo para implementación de *buffers* y funciones para el manejo de éstos, y un segundo módulo que utiliza al anterior que sería la implementación del *Nivel Físico* en sí, ya que es donde se programa la línea serie del PC según la estructura explicada.

Como ya hemos visto en el apartado 5, se construyó un *Medio Físico* que permita el control por software del modo del robot y hacer un *reset* de éste. Para ello, a este nivel, disponemos del *registro de control del módem (MCR)*. El bit de *RTS* (bit 1) controla el modo de funcionamiento del robot, de manera que si se pone a uno el robot está en modo *RUN* y si se pone un cero el robot queda en modo *DOWNLOAD*. Para realizar un *reset* del robot disponemos del bit de *DTR* (bit 0) del mismo registro, de manera que si se pone a uno y a continuación se pone a cero, se produce un *reset* del robot. Para que el robot funcione normalmente estos dos bits deben estar a uno.

También se dispone de unas variables de estado de *Nivel Físico* del *PC* que sirven para consultar los errores que se han producido en el *Nivel Físico*, como por ejemplo el número de errores de *overrun*, etc…

6.3.3. Interficie

La *interficie* de acceso al *Nivel Físico* del *PC* es más completa que la del robot ya que la implementación en el *PC* se ha realizado de manera más genérica que en el robot. Disponemos básicamente de 5 funciones: *F_Open* (permite abrir el *Nivel Físico*), *F_Close* (permite cerrar el *Nivel Físico*), *F_Read* (permite leer caracteres, procedentes de línea serie, del *buffer* de entrada) y *F_Write* (permite enviar caracteres por línea serie escribiéndolos en el buffer de salida), *F_State* (permite consultar las variables de estado del *Nivel Físico*). Disponemos también de dos funciones más que no son genéricas, sino que se han implementado ante una necesidad del *Nivel de Enlace*. Estas funciones son: *F_Env* (indica si el nivel físico está enviando caracteres) y *F_ResetRob* (permite hacer un *reset* del robot por software). Para un uso correcto del *Nivel Físico* primero se debe abrir con *F_Open*, ejecutar las funciones que se deseen para enviar o recibir caracteres, etc… y finalmente cerrarlo con *F_Close*. Si no se abre el *Nivel Físico* cualquier función diferente de *F_Open* devolverá error.

En este caso no indicaremos los diagramas de flujo de cada una de las funciones ya que no aportarían nada sustancial para la comprensión general del *Nivel Físico* implementado, simplemente haremos una descripción general del comportamiento de las funciones. Para conocer con detalle como se han implementado estas funciones o cuales son sus parámetros de entrada y salida se puede consultar el anexo con el código de la implementación de *Nivel Físico* en el *PC* (especialmente el fichero "fisico.h").

Todas la funciones de *Nivel Físico* que se describen a continuación devuelven como parámetro un indicador de que no ha habido error o un indicador con el tipo de error sucedido en la ejecución. Las constantes que indican los posibles errores son las siguientes:

F_NO_ERROR	(No hay error)
F_ERROR_COM_INCORRECTO	(puerto de comunicaciones (*COM*) incorrecto, no entre 0-4)
F_ERROR_COM_NO_DETECTADO	(puerto de comunicaciones indicado no detectado)
F_ERROR_NO_OPEN	(*Nivel Físico* no abierto)
F_ERROR_NO_CLOSE	(*Nivel Físico* no cerrado)
F_ERROR_PARAM_DESCONOCIDO	(Parámetro estado desconocido)
F_ERROR_ACCESO_BUFFER	(*Buffer* sin espacio o vacío)

F_Open permite abrir el *Nivel Físico*. Inicializa los parámetros de este nivel, como pueden ser la velocidad, el número de bits por carácter, tipo de paridad, bits de *stop*, puerto de comunicaciones, vector de interrupciones, etc… y el estado del *Nivel Físico*. Esta función sólo permite modificar la velocidad de transferencia y el puerto de comunicaciones que se desea utilizar ya que para resto de parámetros, por razones que ya veremos, sólo hay una configuración válida posible. Después de haber ejecutado *F_Open* es necesario cerrar el *Nivel Físico* mediante *F_Close* antes de poder volver a abrirlo nuevamente.

F_Close permite cerrar el *Nivel Físico*. Restaura los parámetros de línea serie que había antes de ejecutar *F_Open* excepto para el *registro de control de módem (MCR)*. Como se ha comentado, el bit *RTS* de este registro controla el modo de funcionamiento del robot y el bit *DTR* permite hacerle un *reset* (al inicializar el *PC* suelen estar a cero). Cada vez que se cerraba el *Nivel Físico*, al restaurar el registro *MCR*, el robot quedaba en modo *DOWNLOAD*. En este modo suele hacer un pequeño ruido y se suelen encender los *leds*, y lo más importante es que los motores se desconectan. Por tanto restaurar este registro con el valor anterior es peligroso ya que si el robot queda en modo *DOWNLOAD* los motores se desconectarán con el peligro de que el robot se caiga o se golpee. Ante este problema se decidió no restaurar este registro, de manera que cuando se cierra el *Nivel Físico* el robot continúa en modo *RUN* manteniendo la ultima posición en que ha quedado y, por tanto, evitando el riesgo de posibles caidas o golpes.

F_Read permite leer caracteres, recibidos por línea serie, del *buffer* de entrada. En concreto lee una cadena de caracteres de la longitud que se le indica del *buffer* de entrada. La longitud de la cadena que devuelve puede ser menor o igual que la longitud deseada dependiendo de si en el *buffer* de entrada hay o no suficientes disponibles.

F_Write permite escribir caracteres para ser enviados por línea serie en el buffer de salida. Escribe una cadena de caracteres de la longitud que se le indica en el buffer de salida sólo si hay espacio suficiente en dicho buffer. Si no hay espacio suficiente para escribir la cadena no se enviará nada por línea serie. También comprueba si la cadena de interrupciones de transmisión está activa y si no es así la activa enviando el primer carácter disponible directamente a la línea serie.

F_State consulta el valor de la variable de estado que se le indica. Hay que pasarle un parámetro indicando qué variable de estado se desea consultar y una variable para guardar dicho valor. Mediante esta función se puede consultar el número de errores de cada tipo posible que se han producido en el *Nivel Físico*: número de errores de *overrun*, *breakline*, paridad, trama y *overrun* de *buffer*.

F_Env simplemente indica en el *Nivel Físico* si se están enviando caracteres en el momento en que se ejecuta, en concreto comprueba si la cadena de transmisión de caracteres está activa.

F_ResetRob permite realizar un *reset* del robot por software de la manera indicada en el apartado anterior (a través el bit *DTR* del registro *MCR*).

6.4. COMPATIBILIDAD ENTRE IMPLEMENTACIONES

6.4.1. Velocidad de funcionamiento

Para comunicar los dos *Niveles Físicos* y en concreto los dos *chips* de línea serie necesitamos que éstos trabajen a la misma velocidad. Como se ha visto el la línea serie del robot tenemos un *MOTOROLA MC68HC11E2FN* que incorpora un sistema de línea serie y en el *PC* tenemos un *chip* específico que puede ser la familia *i8250* o *i16550*. El problema para comunicar los dos sistemas viene de que funcionan a diferentes frecuencias de reloj. Mientras que el *MOTOROLA* funciona a una frecuencia de 2 MHz. el *chip* de línea serie del PC funciona normalmente a una frecuencia de 1,8432 MHz.

En el *MOTOROLA MC68HC11* para obtener la velocidad de funcionamiento del sistema de línea serie se aplican dos divisores, cuyo valor se puede modificar, a la frecuencia de 2 MHz. (ver el anexo sobre la línea serie del *MC68HC11* para más información).

La manera de obtener la velocidad de funcionamiento en el *PC* es dividir los 1,8432 MHz. por 16 internamente, lo que nos da un valor máximo de 115200 baudios y luego se le aplica otro divisor cuyo valor es programable.

Esto nos ofrece un determinado rango de velocidades tanto en el robot como en el PC que en ningún caso son completamente iguales entre sí (siempre difieren aunque sea por poco). Sólo podremos utilizar aquellos pares de velocidades cuya diferencia esté por debajo de un valor máximo (errores de hasta el 3% o 5% pueden ser aceptables). Por otra parte, nos interesará la mayor velocidad de funcionamiento posible ya que así podremos enviar un mayor número de bytes por ciclo.

En la figura 6.5. podemos ver una tabla con las posibles parejas de velocidades a partir de 9600 (no nos interesan velocidades inferiores) y el tanto por ciento de error que presentan. Como se puede observar sólo disponemos de dos parejas de velocidades con error por debajo del 3% y un pareja con un error del 5% (aproximadamente), es resto presenta errores mucho mayores.

MOTOROLA MC68HC11			PC		
DIVISOR SCP	DIVISOR SCRC	VELOCIDAD	DIVISOR DL	VELOCIDAD	% ERROR
1	1	125.000	1	115.200	7,84 %
1	2	62.500	2	57.600	7,84 %
3	1	41.667	3	38.400	7,84 %
1	4	31.250	4	28.800	7,84 %
3	2	20.833	5	23.040	9,58 %
			6	19.200	7,84 %
1	8	15.625	7	16.457	5,06 %
			8	14.400	7,84 %
3	4	10.417	11	10.473	0,53 %
13	1	9.615	12	9.600	0,16 %

Figura 6.5. *Tabla comparativa de velocidades a partir de 9600.*

Tras un estudio de la situación y realizar las pruebas pertinentes se ha descartado la pareja de velocidades de 15625-16457 porque la diferencia entre ellas (error del 5%) es demasiado grande. Las otras dos parejas de velocidades son las únicas válidas y por tanto sólo tendremos como posibilidades de funcionamiento para el *Nivel Físico* 9615-9600 y 10417-10473. Esto podría limitar el diseño del *Nivel de Enlace* en cuanto a la información que podamos enviar dentro de cada ciclo (podría ser que no pudiéramos enviar todo lo necesario dentro de un ciclo), pero esto ya se verá más adelante. Por el momento no nos decantamos por ninguna de las dos posibilidades ya que por un lado nos interesaría la mayor pero por otro lado 9600 es una velocidad más usual. Por tanto dejamos esta decisión para más tarde y cuando se diseñe el *Nivel de Enlace* veremos cual nos interesa más.

6.4.2. Paridad, bits de *stop* y número de bits por carácter.

Como se ha visto para las implementaciones del protocolo de *Nivel Físico* se utilizan chips diferentes de manera que no son totalmente compatibles entre sí. Esto nos va a limitar en cuanto a las posibilidades de transmisión de los caracteres por línea serie.

En la parte del robot, el *MOTOROLA MC68HC11A2FN* presenta un sistema de línea serie interno, y en la parte del robot, se utiliza un chip específico que puede ser de la familia *i8250* o *i16550*. Las limitaciones principalmente van a venir del sistema de línea serie del *MOTOROLA* ya que es menos flexible y general que el sistema del *PC*.

En la figura 6.6 se muestra una tabla comparativa en cuanto a ciertas características de la transmisión por línea serie en que quedan claras las diferencias entre estos sistemas. Como se puede observar no podremos tener bit de paridad (el *MOTOROLA* no presenta esta opción) y el número de bits de *stop* sólo puede ser 1 (única posibilidad para el *MOTOROLA*). En cuanto al número de bits por carácter sólo puede ser de 8 (ya nos va bien) por ser el único valor en común entre los dos sistemas.

	ROBOT	PC
Numero de bits por carácter	8 ó 9	5, 6, 7 ó 8
Número de bits de paridad	Ninguno	1 (paridad par o impar) o ninguno
Número de bits de stop	1	1, 1.5 ó 2

Figura 6.6. *Tabla comparativa de las características de línea serie en el MOTOROLA MC68HC11 y el PC.*

6.5. PRUEBAS REALIZADAS Y CONCLUSIONES

Para probar el *Nivel Físico* se han realizado diferentes programas que se han probado a las dos velocidades de funcionamiento posibles. Los primeros programas eran muy sencillos de manera que por ejemplo el robot enviaba un patrón de bytes cíclicamente y en el *PC* simplemente se recibía por línea serie indicando el número de caracteres recibidos, el número de veces que el carácter recibido no coincidía con el esperado, errores de línea serie producidos, etc… Más adelante se pasó a una comunicación *full-duplex* de manera que tanto el robot como el PC enviaban y recibían caracteres, etc…

Uno de los programas implementados en el robot hacía que éste enviara por línea serie todo lo que recibía (como si hiciera un eco de todo lo que se le enviaba). Con este programa se detectó que la diferencia de velocidades de funcionamiento de la línea serie del robot y del *PC* puede producir algún problema.

Como se ha comentado las velocidades de funcionamiento de la línea serie del robot y el *PC* no son exactamente iguales. Estas pequeñas diferencias no son problema bajo un funcionamiento normal en que envío y recepción por línea serie son totalmente independientes. En el caso del programa del robot que hacia el eco de caracteres y a la mayor de las velocidades posibles, el *PC* enviaba algo más rápido de lo que podía enviar el robot (el *PC* enviaba a 10473 baudios y el robot sólo podía enviar a 10417 baudios, por lo que el flujo de entrada era algo mayor que el de salida) y esto hacía que después de cierto tiempo el buffer de entrada del robot se colapsara. Lo mismo, pero al contrario, sucedía con cuando el robot funcionaba a 9615 baudios y el *PC* a 9600 baudios) ya que en este caso el que envía más rápido es el robot y el que se colapsaba era el buffer de entrada del *PC* (después de más tiempo, por ser una velocidad menor y el buffer del *PC* más grande y porque la diferencia de velocidades también es menor).

De todos modos, como hemos comentado, esto sólo sucede cuando la recepción y la transmisión van ligadas de cierta manera como en el caso comentado. Normalmente los flujos de recepción y de transmisión son independientes y no son continuos, por tanto no se esperan problemas de este tipo.

En cuanto al resto no se han detectado ningún tipo de problemas con los programas implementados y pruebas realizadas. En todos los casos funcionando a las dos velocidades posibles, los resultados han sido totalmente

satisfactorios de manera que no se ha detectado ningún tipo de error. Por ejemplo se ha tenido el programa más completo de los implementados funcionando durante más de una hora y no se ha producido ningún tipo de error de recepción o transmisión ni en el robot ni en el *PC*.

Por tanto podemos concluir que se ha implementado un *Nivel Físico* que permite el intercambio de caracteres entre el robot y el *PC* de manera fiable. A este nivel se han cumplido la totalidad de objetivos que se marcaron inicialmente.

Por último comentar que en el *PC* este nivel se ha implementado de la forma más genérica posible por lo que se podría utilizar sin ningún problema para otro tipo de aplicaciones que necesitaran de unos servicios como los que ofrece este nivel (envío y recepción de caracteres a través de línea serie).

7. NIVEL DE ENLACE

7.1. PROTOCOLO DE NIVEL DE ENLACE

7.1.1. Diseño de tramas

Tramas del robot al *PC*

Como primer paso para el diseño del protocolo debemos especificar cómo serán las tramas que se enviarán del *PC* al robot y viceversa. Las necesidades y características en cada sentido son diferentes, ya que en un sentido se enviará el estado de los sensores y en el otro sentido se enviarán órdenes y peticiones de datos sobre qué sensores se quieren recibir.

En general para el diseño de las tramas necesitaremos como mínimo un valor que indique el inicio de una trama y un valor de *CRC* o *Checksum* para detectar posibles errores en su transmisión.

Del robot al PC se desea enviar el estado de los sensores. Como se vio en el apartado 3.1.1 hay un total de 9 grupos diferentes de información de sensores. Estos valores, en varios casos, internamente ocupan más espacio en memoria del realmente imprescindible: por ejemplo los motores sólo toman valores de fuerza de 0 a 14 pero internamente se guardan en un byte. Por lo tanto para la construcción de la trama habrá que eliminar todo el espacio sobrante y ocupar sólo el estrictamente necesario para el rango de valores que pueda alcanzar cada sensor. No tendría sentido enviar bytes (o bits) que no lleven información si no hay un motivo que lo justifique ya que se incrementaría el tráfico de información innecesariamente.

Por otra parte uno de los requerimientos (en concreto el número 3 del apartado 4.3.1) era que el protocolo fuera configurable, de manera que solo se enviara al *PC* información de los sensores que se deseara consultar. Esto requiere que las tramas puedan ser variables y que en función de la información que se desee puedan llevar información de sensores diferentes. Ante esta situación inicialmente se pensó que en el robot se guardara información sobre qué sensores se deseaban en el *PC*, información que cambiaría dinámicamente con nuevas peticiones sobre sensores por parte del *PC*. El problema que se presentaba era que desde el *PC* se podría hacer una petición de nuevos sensores a recibir, que podría perderse por el camino sin que el robot la recibiera. Entonces el *PC* recibiría tramas con un tipo de información de sensores antiguo, pensando que se reciben tramas con información del nuevo tipo deseado, y por tanto se produciría un comportamiento erróneo.

Por tanto para un buen funcionamiento del protocolo es necesario algo más de información en las tramas. La solución que se ha adoptado es incluir en cada trama de sensores información sobre los datos que lleva la trama, así el *PC* puede identificar siempre lo que recibe y comprobar si es lo solicitado. Como tenemos nueve grupos, necesitaremos como mínimo nueve bits para esta información, de manera que el bit a uno indicará que en la trama hay información sobre ese tipo de sensor y el bit a cero indicará que en la trama no existe información de ese tipo.

Otro factor a tener en cuenta es que hay sensores cuyo valor se actualiza cada dos ciclos, por lo que sólo será necesario enviar la información sobre el estado de estos sensores cuando se hayan actualizado. En concreto se trata de los *piros* y las baterías, de modo que en el robot en los ciclos pares se actualizan los *piros* 2 y 3 y la batería de electrónica, y los ciclos impares se actualizan los *piros* 1, 4 y 5 y la batería de motores. Ante esta situación se ha optado por tener dos tipos de trama (0 y 1) de manera que la trama 0 lleva información de *piros* y baterías correspondiente a los ciclos pares y la trama 1 lleva información de *piros* y baterías correspondiente a los ciclos impares (siempre que se desee recibir *piros* o baterías).

Después de estudiar diferentes opciones hemos optado por un formato de trama como el que se indica en la figura 7.1, donde podemos ver los dos posibles tipos de trama. Se indican las tramas con su tamaño máximo (caso en que se desee recibir todos los sensores) pero según lo que se desee recibir puede haber bytes que no aparezcan (por ejemplo si no se desean motores alfa los bytes del 4 al 6 no aparecerían, el byte número 4 sería el correspondiente a motores beta 0 y 1).

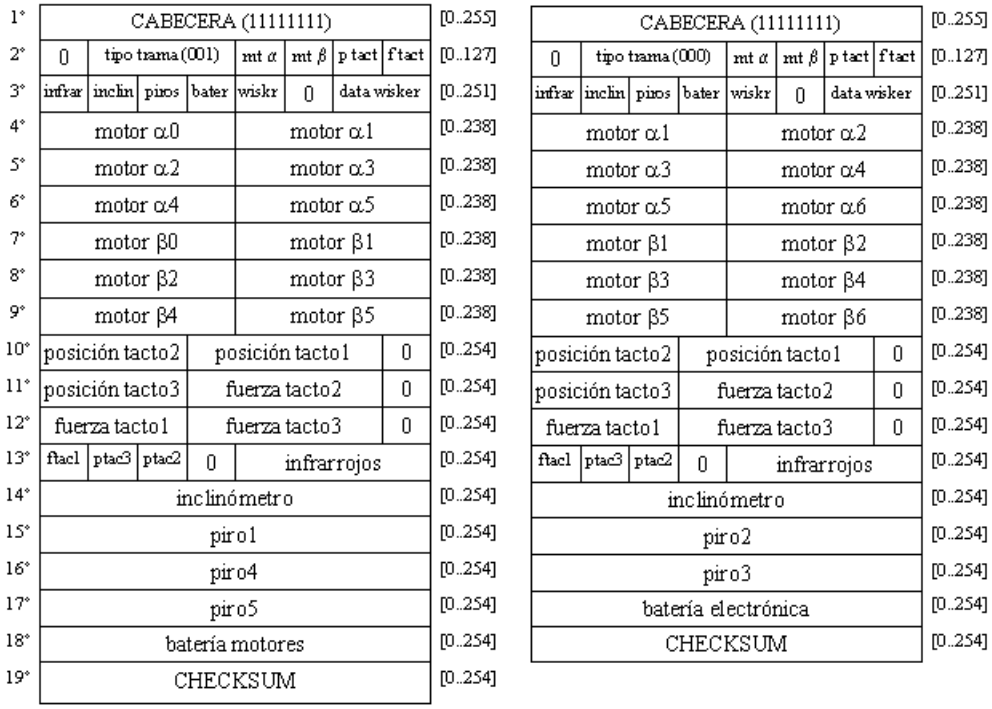

Figura 7.1. *Tramas de sensores del robot al PC.*

Los tres primeros bytes son comunes para ambos tipos de trama y siempre están presentes. Como valor para la cabecera se ha elegido el 255 ya que es un valor extremo y el valor con menor probabilidad de aparecer tanto en los valores de los sensores como al compactar esos valores en la trama.

El byte 2 consta de un primer bit a cero para evitar que se alcance el valor 255 y se pueda confundir con una cabecera. A continuación tenemos tres bits para el tipo de trama, que puede ser 0 ó 1, por lo que sobran bits por si en un futuro se desea ampliar el protocolo. Los cuatro bits restantes indican si la trama contiene información de motores alfa, motores beta, posiciones de tacto y fuerzas de tacto respectivamente.

En el byte 3 encontramos cinco bits para indicar si la trama contiene información de infrarrojos, inclinómetro, piros, baterías y whiskers respectivamente. Sigue un bit a cero para evitar que el byte llegue al valor 255 y se pueda confundir con una cabecera, y finalmente tenemos dos bits que corresponden a los valores de los whiskers (el derecho en el bit 1 y el izquierdo en el bit 0). En caso de que no se desee información sobre estos sensores estos bits quedan a cero.

Los siguientes bytes pueden o no aparecer (según los 9 grupos sensoriales indicados en la descripción de los bytes 2 y 3). En primer lugar tenemos tres bytes de fuerza de motores alfa, seguidos de otros tres de fuerza de motores beta. A continuación tenemos un byte y medio de posiciones de tacto y un byte y medio de fuerzas de tacto. Como al compactar las posiciones y fuerzas de tacto los bytes podrían llegar a 255, se ha optado por coger el bit de más peso de cada uno de los bytes de tacto (en caso de existir) y pasarlos a la mitad alta del siguiente byte. Entonces se desplaza el byte de tacto una posición a la izquierda y en el bit de menos peso se pone un cero, de manera que ya no es posible que alcance un valor de 255 (al recibir la trama hay que recuperar los valores correctamente).

El byte que sigue a los datos de tacto contiene tres bits correspondientes a los bits más significativos de los bytes de tacto, un bit a cero para evitar que llegue a 255 y se confunda con una cabecera y cuatro bits (los de menor peso) correspondientes a la información de sensores de infrarrojos. Este byte puede aparecer por dos motivos: que se desee información de tacto (ya sea posiciones, fuerza o ambas) o que se desee información de los infrarrojos.

El siguiente byte que puede aparecer corresponde al valor del inclinómetro, y a continuación vienen los *piros*, que según el tipo de trama (o de ciclo, según como se quiera ver) serán dos o tres. Por último tenemos el valor de la batería de electrónica o de motores según el ciclo (o tipo de trama). Ante la posibilidad de que alguno de los bytes de inclinómetro, *piros* o de batería pueda confundirse con una cabecera, si alguno llega a 255 (en principio no deberían llegar a estos valores pero preferimos asegurarnos) se le pone el último bit a cero de manera que valdrá 254 y ya no se podrá confundir.

El último byte corresponde al *checksum*, que contiene la suma (módulo 256) de todos los bytes de datos (todos menos la cabecera). Si llega a 255 se le pone el último bit a cero, igual que en los casos comentados anteriormente. Esto hay que tenerlo en cuenta al calcular el *checksum* para la comprobación de la integridad de la trama en la recepción.

Con esto conseguimos una trama con información dinámica (hay bytes que podrán aparecer o no), que por tanto cumple uno de los requerimientos, y que nos ofrece la seguridad en todo momento de saber cual es su contenido. La trama más larga será de 19 bytes (cuando se deseen todos los sensores) y la más corta será de 4 bytes (cuando no se desee ningún sensor).

Tramas del *PC* al robot

Ya sabemos cómo son las tramas que irán del robot al *PC* y ahora nos interesa saber cómo serán las tramas que circularán en sentido contrario, del *PC* al robot.

En este caso nos interesa poder enviar órdenes al robot y poder hacer peticiones sobre qué tipo de sensores queremos recibir. Podría suceder que en un momento concreto en que se quisiera realizar una petición de sensores no hubiera nuevas órdenes que enviar. Por lo tanto se ha optado por definir dos tipos de trama: la trama 0 será una trama de petición de datos (sin órdenes para los actuadores) y la trama de tipo 1 será una trama de órdenes para los actuadores del robot. Ante el requerimiento de que el protocolo sea capaz de recuperarse de *resets* (el número 1 del apartado 4.3.1) se ha optado, como veremos, por que el PC envíe a cada ciclo información de qué sensores desea recibir. Por lo tanto cuando se envíen órdenes también deberemos enviar qué datos se desean, por lo que no se contempla la posibilidad de una trama que sólo lleve información de órdenes para los actuadores. Por esto, lo que se ha hecho es que las tramas de órdenes (tipo 1) también incluyan un campo que indique qué datos desea recibir el PC, si se desea un nuevo tipo de datos se actualizan este campo y si no, se envía con el mismo valor que para la petición anterior.

Por lo tanto las tramas quedan como se puede ver en la figura 7.2. Los tres primeros bytes son prácticamente iguales para las dos tramas (sólo presentan una pequeña diferencia en los dos bits de menor peso del tercer byte) y también son prácticamente idénticos a los tres primeros bytes de las tramas procedentes del robot. Esto se ha hecho así porque de esta manera se obtiene una mayor uniformidad entre tramas que van en un sentido y en otro.

Para ambas tramas tenemos una cabecera de 255, un primer byte formado por un cero (para evitar un valor de 255) seguido de tres bits para tipo de trama (puede ser solo 0 ó 1 y vuelven a sobrar bits para ampliación del protocolo) y cuatro bits más que junto con los cinco primeros del byte siguiente indican si el *PC* desea recibir datos de motores alfa, motores beta, posiciones de tacto, fuerzas de tacto infrarrojos, inclinómetro, *piros*, baterías y *whiskers* respectivamente. Los dos bits de menor peso, en el caso de tramas de tipo 0 van a cero y en el caso de tramas de tipo 1 guardan el bit de mayor peso de la orden a los *leds* (bit 1) y el bit de mayor peso de la orden al *buzzer* (bit 0).

En la trama de tipo 1 además tenemos 6 bytes para órdenes a motores alfa y otros 6 para órdenes a los motores beta. Estos valores van sólo de 0 a 63 de manera que no necesitan un byte entero (con seis bits y uno más a cero para evitar un valor de 255 se podrían compactar) pero se ha preferido no compactarlos ya que sólo se ganaría algo más de un byte y dificultarían bastante el proceso de generación y decodificación de la trama. De esta manera se desperdicia un bit por cada orden de motor pero se mantiene la claridad en el diseño de la trama y la facilidad a la hora de construirla y realizar el paso inverso en la recepción. Los otros dos bytes corresponden a orden de leds y buzzer y ambos llevan el bit de mayor peso en el tercer byte de la trama, de manera que se han desplazado un bit a la izquierda y el último bit se ha puesto a cero para evitar el valor de 255 y confundirlos con una cabecera. Por tanto en la recepción se debe reconstruir estos valores correctamente.

El último byte de las dos tramas es un *chesksum*, que se calcula igual que para las tramas que van del robot al *PC*. Las dos son tramas de tamaño fijo, de modo que la trama de tipo 0 ocupa siempre 4 bytes y la de tipo 1 ocupa siempre 18 bytes.

1°	CABECERA							
2°	0	tipo trama (001)			mt α	mt β	p tact	f tact
3°	infrar	inclin	pinos	bater	wiskr	0	leds	buzzer
4°	orden motor α0							
5°	orden motor α1							
6°	orden motor α2							
7°	orden motor α3							
8°	orden motor α4							
9°	orden motor α5							
10°	orden motor β0							
11°	orden motor β1							
12°	orden motor β2							
13°	orden motor β3							
14°	orden motor β4							
15°	orden motor β5							
16°	orden leds							0
17°	orden buzzer							0
18°	CHECKSUM							

CABECERA							
0	tipo trama (000)			mt α	mt β	p tact	f tact
infrar	inclin	pinos	bater	wiskr	0	0	0
CHECKSUM							

Figura 7.2. *Tramas de órdenes del PC al robot.*

Finalmente comentar que se han diseñado unas tramas que cumplen los requerimientos planteados, fundamentalmente permiten que el protocolo sea configurable y no limitativo (ver requerimientos 3 y 6 del punto 4.3.1). Presentan una gran uniformidad entre los diferentes tipos y su estructura ayudará al buen funcionamiento del protocolo. En cuanto a su tamaño decir que durante el diseño se tuvo en cuenta en todo momento que a la menor de las velocidades posibles de *Nivel Físico* (ver 6.4.1) se pueden enviar o recibir como máximo 19 caracteres en un ciclo (a la velocidad máxima posible son 20 caracteres por ciclo). Por lo tanto las tramas van ajustadas al tamaño máximo pero por otra parte se prefiere así ya que de esta manera se puede disponer de todas las órdenes e información de todos los sensores en cada ciclo (el protocolo no es limitativo). En caso de que hubieran sido demasiado grandes y no hubiera sido posible, para ninguna de las posibilidades de velocidad de la línea serie, enviar una trama de órdenes y recibir una de sensores dentro de un ciclo, se hubieran tenido que acortar de alguna manera, posiblemente incumpliendo alguno de los requerimientos.

7.1.2. Diseño del protocolo

Antes del diseño definitivo de protocolo se realizó una primera aproximación con transmisión y recepción de tramas completamente independiente, que no cumplía la totalidad de los requerimientos. En concreto, la trama de sensores como respuesta a una determinada orden podía llegar al *PC* en un número de ciclos variable, incumpliendo el requerimiento número 2 del punto 4.3.1. La gran importancia de este requerimiento para el correcto funcionamiento de los programas de usuario que hagan uso del protocolo, llevó a un nuevo diseño que trabaja por ciclos, coordinados a partir del envío de datos por parte del robot.

El diseño final corresponde a un protocolo asimétrico para comunicar sólo dos sistemas remotos. Es asimétrico porqué uno de los dos sistemas remotos es el que dirige la comunicación (en nuestro caso el robot, ya que es donde se generan los datos cada 20 milisegundos) y el otro simplemente sigue el ritmo que marca el primero (en este caso el *PC* sigue el ritmo que marca el robot). Sólo permite la comunicación de dos entidades ya que no existe ningún tipo de identificador sobre el origen y destino de la trama.

El punto de partida del funcionamiento del protocolo se origina en el intercambio de datos que se realiza cada 20 milisegundos entre los micros *SERVO* y *MAIN* en el robot. A partir de este momento se genera una trama de sensores que se envía hacia el *PC* y se pasa a esperar una trama de órdenes procedente de éste. Cuando en el *PC* se detecta el inicio de una nueva trama de sensores se considera que se inicia un nuevo ciclo, que también será de 20 milisegundos, y es el momento en que se envían los datos de órdenes o peticiones al robot. Por tanto, para que puedan llegar órdenes al robot éste ha de enviar una trama de sensores a cada ciclo y, en el caso de que en el PC no se desee ningún sensor, lo que se hace es enviar una trama de sensores sin información, simplemente con cabecera, todos los bits de datos de la trama a cero y un *checksum*.

Cuando el robot recibe la trama del *PC* se actualiza la información sobre qué datos espera el *PC* y si es una trama de órdenes (tipo 1) guarda las órdenes para los actuadores, que se enviarán al *SERVO* en el ciclo siguiente. De esta manera se obtiene una trama de sensores como respuesta a una determinada trama de órdenes siempre cuatro ciclos después del ciclo en el que se ha enviado la trama de órdenes. Si el robot en un ciclo no recibe órdenes procedentes del *PC* lo que hace es enviar al *SERVO* las del ciclo anterior (el robot mantiene los actuadores con los valores que tienen).

En cuanto a la recepción en el PC de tramas procedentes del robot, el hecho de que se necesite el máximo tiempo posible para la ejecución de un programa de usuario, desaconsejaba que a cada ciclo se procesara la trama recibida en paralelo al programa usuario, ya que esto quitaría mucho tiempo a dicho programa. La solución por la que se ha optado es dejar los caracteres correspondientes a las tramas que llegan del robot en el *Nivel Físico* y solamente pasar a tratar estos bytes e intentar detectar una posible trama cuando el usuario indique que desea recibir datos (por ejemplo cuando el programa de usuario haya finalizado su trabajo). De esta manera el usuario de este nivel ha de ser el encargado de realizar lecturas de datos más o menos rápido ya que si no las tramas se irán acumulando en el *Nivel Físico* con el peligro de colapso de los *buffers*.

Por otra parte el requerimiento de que sea capaz de recuperarse de *resets* nos obliga a que a cada ciclo el *PC* envíe una trama de órdenes o de petición de datos con información de qué sensores desea recibir (en caso de que no haya nuevas órdenes se enviará una trama del segundo tipo). En caso de que el robot sufra un *reset*, después del primer envío de datos hacia el *PC* recibirá una trama, con información de qué datos desea el *PC* y a partir de ese momento será capaz de enviar tramas con el tipo de datos correcto. En caso contrario enviaría tramas con la información que tuviera definido enviar después del *reset*.

Cuando la trama de sensores recibida en el *PC* no corresponde al tipo esperado, aunque no sea la que se desea, se utiliza para actualizar la información de los sensores que incluye, ya que es información correcta y más actual que la que se tenía hasta ese momento. Por este motivo se ha optado por que el robot, por defecto después de un reset, envíe tramas con información de todos los sensores. Si el problema de la recepción de tramas con tipo de datos diferente del esperado persiste después de un cierto tiempo se indica de esta anomalía al usuario.

El hecho de que en el robot se generen datos de sensores y se envíen órdenes a los actuadores cada 20 milisegundos limita a los posibles programas de usuario de manera que sólo disponen de ese tiempo para leer nuevos datos de sensores, tratar dichos datos de la manera que se desee y generar posibles nuevas órdenes para el robot. Por lo tanto se espera que, para un funcionamiento óptimo (suponiendo que no se producen errores), un programa de usuario se coordine con el inicio de ciclo realizando lecturas hasta que reciba una trama de sensores y a continuación realice los cálculos o tratamientos de datos que desee en como máximo 20 milisegundos generando o no nuevas órdenes y volviendo a sincronizarse con el inicio de ciclo realizando nuevas lecturas de sensores hasta disponer de nuevos datos. Si un programa no sigue esta estructura el funcionamiento no será óptimo ya que o no se enviarán órdenes a cada ciclo o no se leerán tramas de sensores a cada ciclo.

Si el usuario de *Nivel de Enlace* no realiza lecturas cada ciclo (20 milisegundos) cuando quiera realizar una lectura tendremos más de una trama en el buffer de *Nivel Físico*, por lo que se ha optado por devolver al usuario la última de las tramas correctas detectadas, ya que es ésta la única que interesa por ser la que dispone de datos más actuales. En la figura 7.3 se puede ver un gráfico que indica el funcionamiento del protocolo tal y como se ha explicado.

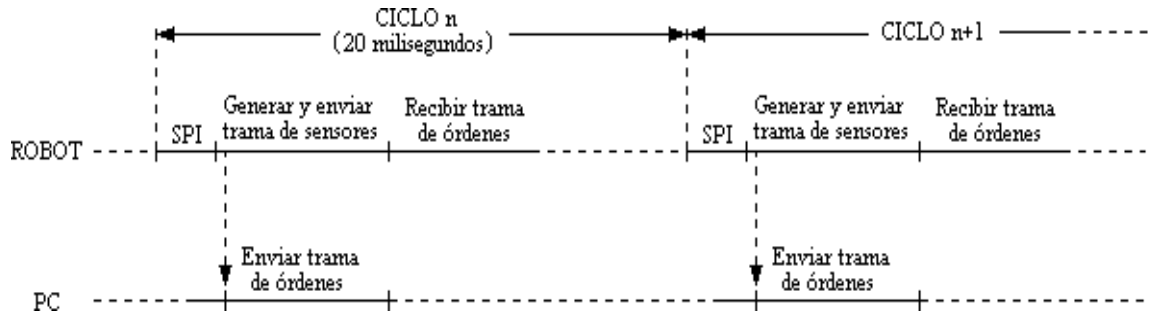

***Figura 7.3**. Esquema de funcionamiento del protocolo*

7.1.3. Limitaciones sobre Nivel Físico y independencia de niveles.

El protocolo de *Nivel de Enlace* por su funcionamiento y características produce unas limitaciones sobre el *Nivel Físico* en cuanto a la relación entre ambos niveles y el valor de ciertos parámetros.

En primer lugar, como hemos dicho, el encargado de realizar las lecturas de los datos procedentes del robot es el programa de usuario que estará por encima del *Nivel de Enlace*. Si este programa no realiza una lectura de sensores por ciclo las tramas se irán acumulando en el *Nivel Físico* con el riesgo de que el buffer se sature. Para que este riesgo sea menor o tarde más en producirse ante un mal diseño del programa de usuario, se decidió tener un buffer de entrada grande en el *Nivel Físico* del *PC*, de 1024 bytes, lo que nos permite tener un total de 53 tramas acumuladas (aproximadamente un segundo sin realizar lecturas de tramas).

Ante la necesidad de que el *PC* se tenga que coordinar con el inicio de una trama de sensores procedente del robot se ha decidido que, cada vez que llegue un carácter al *Nivel Físico* del PC éste informe al *Nivel de Enlace* de este hecho. Para mantener en cierto modo la independencia de niveles lo que se ha hecho es que además se le pase al *Nivel de Enlace* el carácter recibido y sea en este nivel donde se actúe de una manera u otra según su valor. Esto requiere que el *Nivel de Enlace* ofrezca al nivel inferior una función de interficie, para que éste le informe de la llegada de un carácter, y que el *Nivel Físico* para cada carácter recibido haga una llamada a esta función, lo que rompe en cierta manera con la filosofía *OSI*, en la cual, los servicios se ofrecen de un nivel al inmediatamente superior.

Como se verá en el punto 7.2.2 se ha tenido que descartar la velocidad de 9600 baudios para el *Nivel Físico*, ya que, por el tamaño de las tramas, limitación en cuanto al tiempo de ciclo disponible, y tiempo necesario para generar y recibir las tramas, en 20 milisegundos a 9600 baudios, en el robot no se dispone de suficiente tiempo para generar una trama de sensores y enviarla, recibir la de órdenes y realizar los cálculos necesarios.

7.2. ENTIDAD DE NIVEL DE ENLACE EN EL GENGHIS II

7.2.1. Implementación del protocolo

La implementación realizada en el *Genghis II* corresponde a la función de *master* (maestro) en la comunicación. Éste será el que marque el ritmo al PC y aquel simplemente actuará en consecuencia.

En el robot a cada ciclo se debe enviar la información con el estado de los sensores y recibir una posible trama de órdenes. Por tanto, para la implementación del protocolo sólo se ha necesitado de dos funciones, una que genere una trama de sensores y la envíe por línea serie a través de la interficie de *Nivel Físico*, y otra que implemente la máquina de estados para recibir una trama. A continuación se muestran las máquinas de estados para la generación y recepción de tramas en el robot.

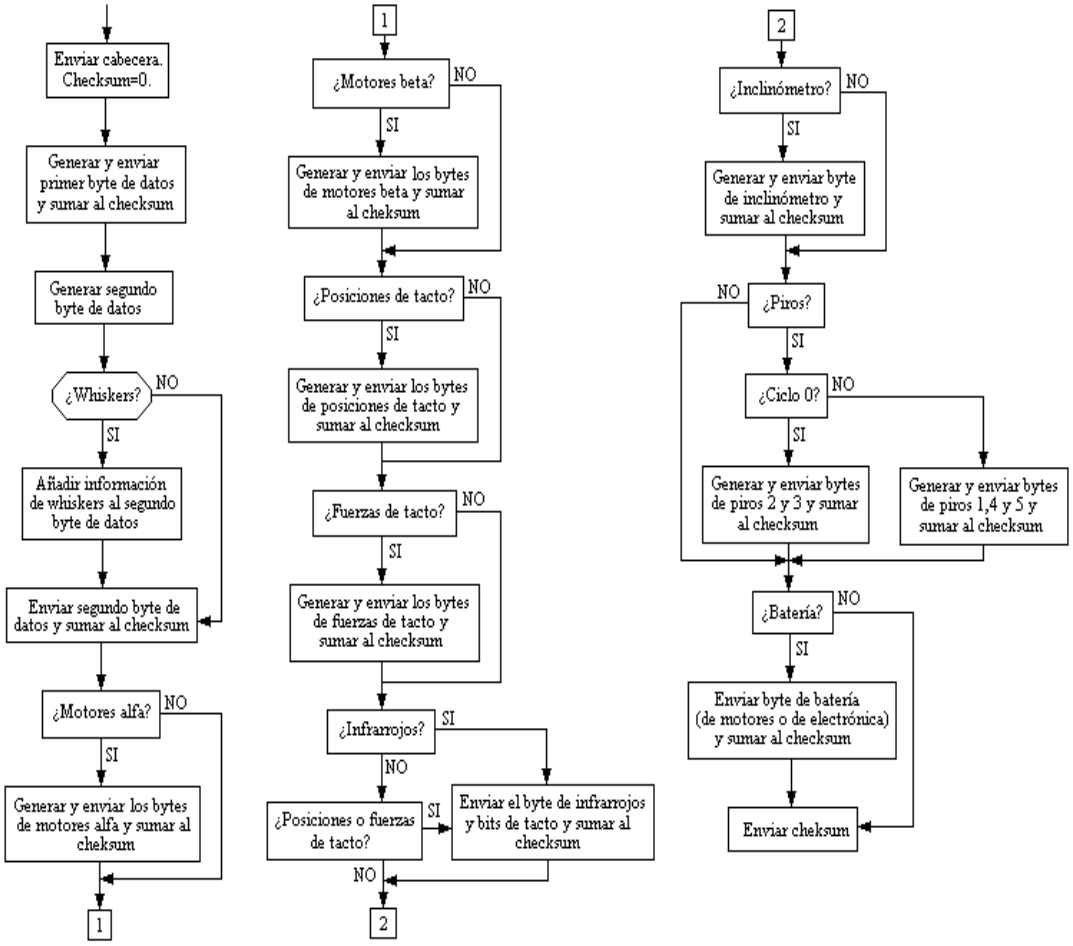

Figura 7.4. *Diagrama de flujo para generación de tramas en el robot*

El resultado de la generación es una trama dinámica que lleva un tipo de información u otro dependiendo de los sensores que se deseen recibir por parte del PC. Por defecto, después de un *reset* el robot envía tramas con información de todos los sensores hasta que recibe una petición que indique algo diferente.

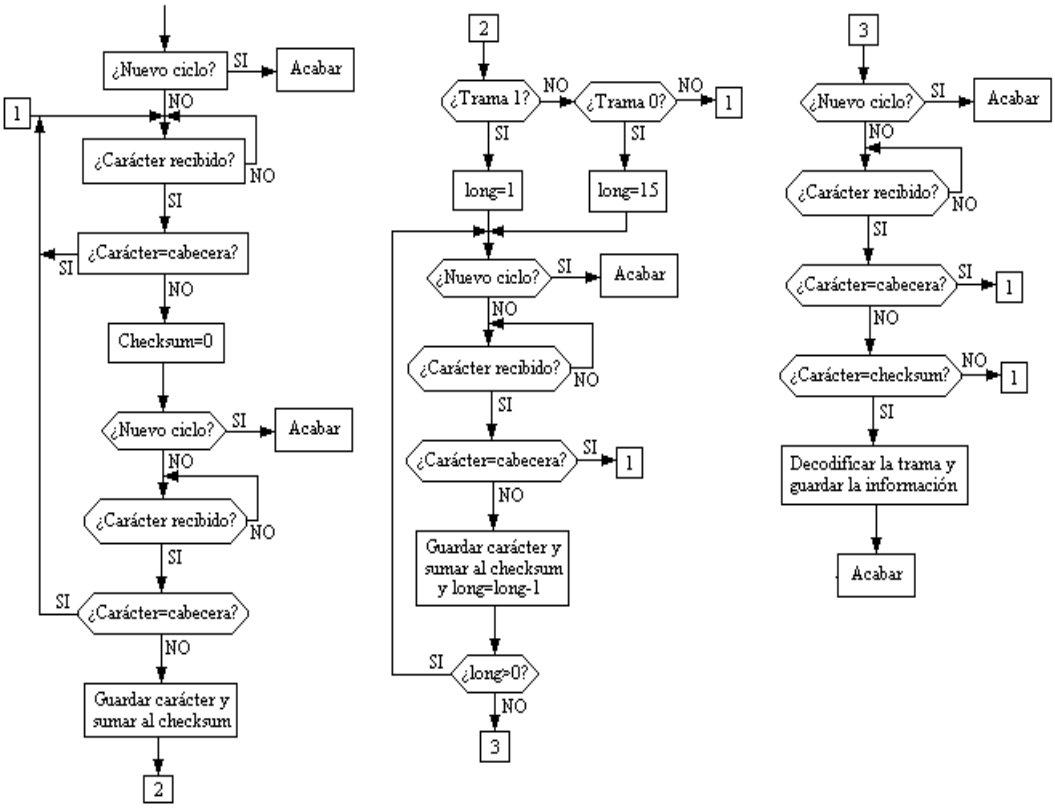

Figura 7.5. *Diagrama de flujo de recepción de tramas en el robot.*

En cuanto a la recepción de tramas, el robot (en concreto el micro *MAIN*) se queda recibiendo una posible trama hasta que detecta una correcta o hasta que se haya producido una comunicación con el *SERVO* (inicio de nuevo ciclo). Es importante destacar que en la recepción, la máquina de estados no guarda el estado de un ciclo a otro, ya que si al final de un ciclo no se ha recibido una trama o hay una a medias, esa información ya no interesa y lo que se ha de hacer es intentar detectar lo antes posible la trama siguiente (el resto de caracteres de la trama en recepción se ignorarán hasta detectar el inicio de trama siguiente).

Las restricciones en cuanto a tiempo en el robot son muy grandes, por lo que nos interesa enviar el inicio de una trama (la cabecera) lo antes posible para que así el *PC* comience a enviar la trama de órdenes también lo antes posible y la trama se pueda recibir en el robot dentro del ciclo. Por lo tanto lo que se hace es enviar el inicio de la cabecera justo después de la comunicación con el *SERVO* (en la interrupción de *SPI*) y más tarde se envía el resto de la trama. Por este mismo requisito el cálculo del *checksum* de la trama se realiza dinámicamente (a medida que se van recibiendo los bytes de la trama), así después de recibir el último carácter el *checksum* ya está calculado.

7.2.2. Estructura y funcionamiento

La estructura diseñada consta de dos *buffers* de salida para el envío de órdenes al *SERVO*, un buffer de entrada para el estado de sensores y una variable que indica qué información se desea en el *PC*. El contenido de estos *buffers* es el mismo que existía en la situación previa y que vimos en la figura 3.6.

La necesidad de dos *buffers* para envío de órdenes al *SERVO* viene de que hay que enviar unas órdenes concretas a cada ciclo. En el caso de que en un ciclo no haya nuevas órdenes procedentes del *PC* o que la trama recibida sea errónea, lo que habrá que hacer es enviar las órdenes del ciclo anterior de manera que los actuadores

mantendrán sus valores. Como la comunicación con el *SERVO* se produce por interrupciones mediante la *SPI* (ver anexo 1 para mayor información sobre la comunicación de *SPI*), podría suceder que, en mitad de la transferencia al *buffer* de salida de datos de una trama de órdenes recibida, llegara la interrupción de *SPI*. En este caso los datos a enviar serían erróneos ya que parte de las órdenes corresponderían a la nueva trama recibida y parte a la del ciclo anterior. La solución consiste en tener dos *buffers*, uno con las órdenes del ciclo anterior (*SpiOut*) y otro con las posibles nuevas órdenes (*SpiOutTmp*). En caso de disponer de nuevas órdenes procedentes de *PC* en *SpiOutTmp* al *SERVO* se le envía este buffer, y en caso contrario se le envía *SpiOut*. Esto requiere que si se dispone de nuevas órdenes, haya que realizar una copia de *OutSpiTmp* a *OutSpi*, para que en el ciclo siguiente, en caso de no disponer de nuevas órdenes, se envíen las mismas (ya que el buffer que se enviaría sería *OutSpi*).

Los ciclos en el *MAIN* se organizan en ciclos pares (tipo 0) e impares (tipo 1). Esto es necesario porque hay sensores, como ya vimos, que se leen cada dos ciclos, de manera que en los ciclos de tipo 0 se leen los *piros* 2 y 3 y la batería de electrónica, y en los ciclos de tipo 1 se leen los *piros* 1, 4 y 5 y la batería de motores.

Los pasos que se realizan en el *micro MAIN* según la estructura implementada son los siguientes: Inicialmente se espera hasta la detección del inicio de ciclo, que viene determinada por la comunicación con el *SERVO* a través de la interrupción de *SPI*. Cuando se ha detectado un inicio de ciclo lo primero que se hace es generar y enviar (a medida que se construye) una trama a partir de los datos de sensores que se han recibido del *SERVO* y la información de los sensores que desea recibir el *PC* (la cabecera de la trama ya se habrá enviado al final de la *SPI* que ha marcado el inicio de ciclo). Si en el ciclo anterior se recibieron órdenes procedentes del *PC* se hace una copia de *SpiOutTmp* a *SpiOut*. En caso de que sea un ciclo impar se actualiza el estado de la batería de motores y la indicación al *SERVO* de si este nivel es menor que el mínimo o no (dos bits altos del primer byte de órdenes para el *SERVO*). Este nivel de baterías mínimo es un límite de seguridad para que los motores no se dañen en caso de que en algún momento consuman demasiado. Por último se pasa a recibir la trama de órdenes hasta que se haya recibido una trama correcta o llegue una nueva *SPI* marcando el inicio del siguiente ciclo. En caso de recibir una nueva trama de órdenes, después de decodificarla, hay que pasar la información de los sensores a *OutSpiTmp* y actualizar la información de qué sensores desea recibir el *PC*. La figura 7.6 muestra un diagrama de flujo de la situación descrita.

El valor inicial para las órdenes a los actuadores en el robot corresponde a un valor de 30 para los motores (posiciones intermedias) y un valor de 0 para los *leds* y el *buzzer* (ambos desconectados). Por tanto después de un reset los actuadores toman los valores indicados.

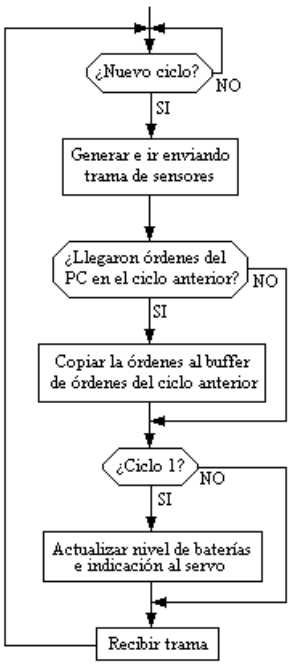

Figura 7.6. *Diagrama de flujo del MAIN.*

Ahora que ya se conoce como se ha implementado la estructura del Nivel Físico en el robot, es el momento de justificar porqué se ha descartado la velocidad de 9600 baudios.

Como la frecuencia de reloj de los micros del robot es de 2 Mhz, en 20 milisegundos, dispondremos de un total de 40.000 ciclos para la ejecución de instrucciones. Teniendo en cuenta que el tamaño de la trama más grande es de 19 caracteres, el tiempo máximo de cálculo que necesita el *MAIN* por ciclo, se puede determinar a partir de los tiempos de: ejecución de una interrupción de *SPI*, escribir el inicio de trama en el buffer de salida y enviarlo al registro de la línea serie (rutina *EnviaCar*), 18 interrupciones para pasar los caracteres del buffer de salida al registro de línea serie, enviar 19 caracteres por línea serie a la velocidad que corresponda y una ultima interrupción que no encuentra más caracteres en el buffer de salida (fin de envío de trama).

A continuación se detalla el número de ciclos para cada caso, según cada una de las dos velocidades del robot compatibles con las del *PC* (9615 ó 10417):

	9615 baudios	**10417 baudios**
Ejecución de interrupción de *SPI*	1147 ciclos	
Enviar inicio de trama (*EnviaCar*)	131 ciclos	
Interrupciones de línea serie	18 * 89 ciclos	
Envío de caracteres	19 * 2081 ciclos	19 * 1920 ciclos
Ultima interrupción de línea serie	83 ciclos	
TOTAL (ciclos)	42502 ciclos	39443 ciclos

Como vemos, a 9615 baudios el número de ciclos necesario supera el disponible y a 10417 baudios disponemos de suficientes ciclos para la ejecución de todo el código necesario . Por tanto la velocidad de la línea serie será de 10417 baudios para el robot y 10470 para el *PC*.

7.2.3. Interficie

En el caso del robot no podemos hablar claramente de una interficie de *Nivel de Enlace*, ya que por las características de programación y funcionamiento del robot, la implementación de los diferentes niveles no está tan clara como en el PC. Aun así consideraremos que hay dos funciones de interficie, una para generar y enviar una trama denominada *GeneraTrama* y otra para la recepción de tramas denominada *RecibeTrama*. Ambas implementan las máquinas de estados de la figura 7.3.

GeneraTrama se encarga de ir generando, a partir de la información recibida de los sensores, uno a uno los bytes de la trama excepto el de cabecera, que por las restricciones de tiempo se prefiere enviar antes (al final de la SPI). Si somos estrictos, esto hace que no sea en sí una función de interficie, ya que depende de que antes se haya enviado una cabecera y por sí sola no permite enviar una trama, pero a nivel abstracto la consideraremos como tal.

RecibeTrama se encarga de recibir una trama. Va leyendo caracteres procedentes de línea serie mientras no detecte una trama correcta o hasta que detecte que ha empezado un nuevo ciclo.

7.3. ENTIDAD DE NIVEL DE ENLACE EN EL PC

7.3.1. Implementación del protocolo

La implementación del protocolo realizada en el PC corresponde a la función de *slave* (esclavo). El *PC* se limita a seguir el ritmo que le marca el robot, de manera que sólo envía órdenes cuando recibe el inicio de una trama de sensores procedente del robot.

Como dijimos, en el *PC* la recepción de tramas va dirigida por el usuario, de manera que es éste el que indica cuándo quiere recibir datos de sensores y en ese momento, en caso de disponer de alguna trama, se le da la información de la última trama de sensores recibida.

En cuanto a la transmisión de órdenes también ha de ser el usuario quien diga cuándo hay nuevas órdenes para enviar al robot. Por tanto dependiendo de si hay nuevas órdenes o no deberemos enviar un tipo de trama u otro (internamente en el *PC* se envían datos cada ciclo, ya sea una trama de órdenes o de petición de datos nueva que desea enviar el usuario o la petición de datos del ciclo anterior, para no perder la sincronización con el robot en cuanto a qué tipo de información se desea recibir). Como estado inicial en el *PC* se pide al robot que no envíe ningún sensor (se podría haber optado por pedirlos todos, ya que el robot por defecto al principio los envía todos, pero se prefiere así).

La figura 7.7. muestra los diagramas de flujo para el envío y recepción de tramas.

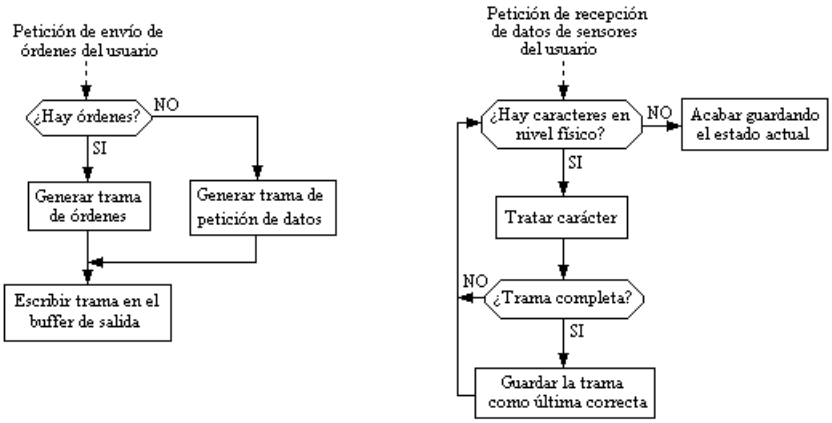

Figura 7.7. *Diagrama de flujo para la recepción y envío de tramas en el PC.*

7.3.2. Estructura y funcionamiento

El hecho de necesitar una interficie para la interacción con el usuario complica la estructura de este nivel. En primer lugar disponemos de un buffer de órdenes, uno de sensores y otro para indicar qué datos se desean del robot, todos ellos accesibles, para lectura o escritura, por el usuario a partir de funciones o macros. Además se dispone de *buffers* y variables de sólo lectura para consultar qué sensores se actualizaron tras la última recepción de información de sensores y para consultar los errores de *Nivel de Enlace* o del estado de la comunicación. Por otra parte disponemos de un buffer de salida donde se contiene la trama a enviar ya generada y que se actualiza cuando el usuario determina que hay nuevas órdenes a enviar.

El usuario escribe órdenes en el *buffer* de salida o modifica el tipo de datos que desea recibir con total libertad. Cuando esos datos son definitivos indica que desea enviar los datos al robot, caso en que, dependiendo de si hay sólo petición de datos o también órdenes, se genera una trama de petición de datos (tipo 0) o de órdenes (tipo 1) respectivamente y se pasa al buffer de salida.

Por otra parte, cuando se recibe el inicio de una trama de sensores procedente del robot, se pasa a enviar el contenido del buffer de salida que contiene la trama generada. En caso de que en ese momento se estuviera actualizando su contenido no se podrá enviar trama en el ciclo actual. Esta situación se indicará durante dicho ciclo como un error en la comunicación, activando el bit correspondiente en la variable de estado de la comunicación (el usuario debería enviar datos con suficiente antelación antes del fin de ciclo para un uso correcto). Si el buffer no se está actualizando (debería ser así siempre) se envía una trama de órdenes o petición de datos al robot según corresponda. En caso de que la trama no se pueda enviar por problemas de *Nivel Físico* se indica durante el ciclo actual como un error en la comunicación al enviar datos, activando el bit correspondiente en la variable de estado de la comunicación. En este caso no hubiera tenido sentido poner dos *buffers*, como se hacía en el robot para la comunicación con el *SERVO*, y enviar la trama de órdenes anterior, ya que esa trama ya está en el robot (excepto si hubiera llegado errónea).

En cuanto a la estructura para la recepción, cada vez que el usuario indica que desea recibir datos se pasa a analizar los caracteres disponibles en el *Nivel Físico* devolviendo, en el buffer de sensores, la información de la última de las tramas correctas detectadas en caso de que hubiera alguna e indicando al usuario si hay nueva información de sensores o no. En caso de que hubiera tardado demasiado en hacer una lectura (habría más de una trama en *Nivel Físico*, correcta o no) se le indica mediante la activación del bit correspondiente en la variable de estado de la comunicación (no es exactamente un error). El usuario es el encargado de comprobar esta y otras indicaciones sobre el estado de la comunicación después de cada lectura si lo desea. Si se ha recibido una trama correcta se decodifica, de manera que la información de sensores se pasa al buffer de sensores de usuario y en

otro buffer de usuario se indica qué grupos de sensores se actualizaron con información de la trama recibida. De esta manera el usuario puede consultar sólo la información actualizada a cada ciclo.

La figura 7.8. indica un diagrama de la estructura descrita.

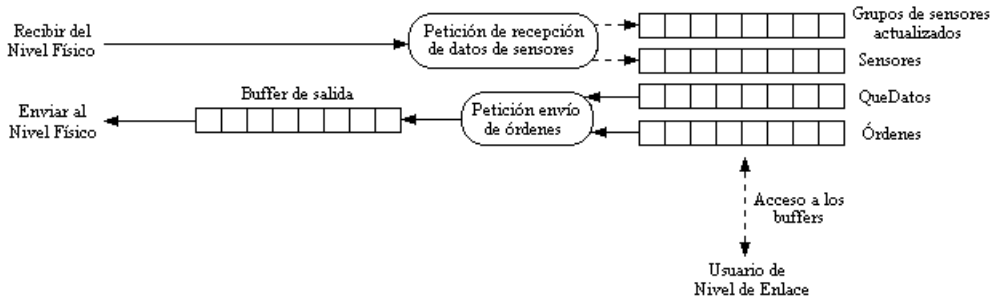

Figura 7.8. *Estructura del Nivel de Enlace en el PC.*

También se han implementado timeouts que permiten comprobar si el host remoto está presente o no al otro lado o si el número de datos de sensores que se reciben es o no óptimo (para esto se ha utilizado la interrupción de reloj del *PC*), y también un timeout por si tras una petición de nuevo tipo de datos de sensores no se recibe contestación con el tipo adecuado (después de cinco tramas correctas se debería tener contestación, normalmente antes).

Por último comentar que las órdenes iniciales para los actuadores corresponden a los valores iniciales por defecto en el robot: posición 30 para todos los motores (posición intermedia) y 0 para *leds* y *buzzer* (ambos desactivados).

7.3.3. Interficie

En el *Nivel de Enlace* del *PC* se ofrecen dos grupos de servicios: aquellos que nos permiten leer o escribir en los *buffers* de usuario (los denominaremos *servicios de usuario*) y aquellos que nos permiten controlar el intercambio de información con el robot o consultar el estado del *Nivel de Enlace* en general (los denominaremos *servicios de protocolo*).

En general el primer grupo se ha implementado como macros, ya que se espera que se utilicen muchas veces dentro de un mismo ciclo para consultar o modificar los valores de los *buffers* de usuario y así la ejecución será más eficiente (con una macro al compilar se hace una copia de código fuente y no es necesario salto a subrutinas ni paso de parámetros).

Las *servicios de protocolo*, en cambio, se han implementado como funciones y en todos los casos devuelven un valor indicando que no ha habido ningún error en su ejecución o el tipo de error sucedido en caso contrario. Los posibles errores son los siguientes:

E_NO_ERROR	(No hay error)
E_ERROR_NO_OPEN	(*Nivel de Enlace* no abierto)
E_ERROR_NO_CLOSE	(*Nivel de Enlace* no cerrado)
E_ERROR_PARAM_DESCONOCIDO	(Parámetro de variable de estado desconocido)
E_ERROR_OPEN_FISICO	(Error al abrir el *Nivel Físico*)

A continuación se describe la interficie para los *servicios de usuario*:

E_WriteMotor permite modificar el valor para un motor en el *buffer* de órdenes de usuario. Como parámetros se le indican, el motor (mediante una constante) y el valor que se quiere escribir, en el rango 0 a 63. Si el valor está dentro del rango y el motor indicado es válido retorna 1 y en caso contrario retorna 0 (el valor no queda modificado).
Las constantes para el parámetro de motor son las siguientes:

E_ORD_MA0	(motor alfa0)	*E_ORD_MB0*	(motor beta0)
E_ORD_MA1	(motor alfa1)	*E_ORD_MB1*	(motor beta1)
E_ORD_MA2	(motor alfa2)	*E_ORD_MB2*	(motor beta2)
E_ORD_MA3	(motor alfa3)	*E_ORD_MB3*	(motor beta3)
E_ORD_MA4	(motor alfa4)	*E_ORD_MB4*	(motor beta4)
E_ORD_MA5	(motor alfa5)	*E_ORD_MB5*	(motor beta5)

E_WriteLeds permite modificar el valor para los *leds* en el *buffer* de órdenes de usuario. Como parámetro se le pasa el nuevo valor, que sólo puede ir de 0 a 255. Si el valor es correcto retorna 1 y en caso contrario retorna 0.

E_WriteBuzzer permite modificar el valor para el *buzzer* en el *buffer* de órdenes de usuario. Como parámetro se le pasa el nuevo valor, que sólo puede ir de 0 a 255. Si el valor es correcto retorna 1 y en caso contrario 0.

E_ReadMotor permite consultar el valor actual de orden para un motor en el *buffer* de órdenes de usuario. Como parámetro se le indica el motor cuyo valor se desea consultar. Si el parámetro es correcto retorna dicho valor y en caso contrario retorna 255 (valor imposible para los motores). Las constantes para el parámetro del motor son las mismas que en el caso de *E_WriteMotor*.

E_ReadLeds permite consultar el valor actual de la orden a los *leds* en el *buffer* de órdenes de usuario. No tiene parámetros y retorna el valor correspondiente.

E_ReadBuzzer permite consultar el valor actual de la orden al *buzzer* en el *buffer* de órdenes de usuario. No tiene parámetros y retorna el valor correspondiente.

E_ReadSensor permite consultar el valor actual de un sensor del *buffer* de usuario de estado de los sensores. Como parámetro hay que indicarle una constante con el sensor cuyo valor se desea consultar. Si el parámetro es correcto retorna el valor correspondiente y en caso contrario retorna 255 (valor imposible para un sensor).
Las constantes para el parámetro de sensor son las siguientes:

E_SENS_MA0	(Motor alfa0)	*E_SENS_MA1*	(Motor alfa1)
E_SENS_MA2	(Motor alfa2)	*E_SENS_MA3*	(Motor alfa3)
E_SENS_MA4	(Motor alfa4)	*E_SENS_MA5*	(Motor alfa5)
E_SENS_MB0	(Motor beta0)	*E_SENS_MB1*	(Motor beta1)
E_SENS_MB2	(Motor beta2)	*E_SENS_MB3*	(Motor beta3)
E_SENS_MB4	(Motor beta4)	*E_SENS_MB5*	(Motor beta5)
E_SENS_PT1	(Posición tacto1)	*E_SENS_PT2*	(Posición tacto2)
E_SENS_PT3	(Posición tacto3)	*E_SENS_FT1*	(Fuerza tacto1)
E_SENS_FT2	(Fuerza tacto2)	*E_SENS_FT3*	(Fuerza tacto3)
E_SENS_IFR	(Infrarrojos)	*E_SENS_INC*	(Inclinómetro)
E_SENS_PR1	(*Piro*1)	*E_SENS_PR2*	(*Piro*2)
E_SENS_PR3	(*Piro*3)	*E_SENS_PR4*	(*Piro*4)
E_SENS_PR5	(*Piro*5)	*E_SENS_BTE*	(Batería de electrónica)
E_SENS_BTM	(Batería de motores)	*E_SENS_WKR*	(*Whisker* derecho)
E_SENS_WKL	(*Whisker* izquierdo)		

E_ActSensor permite consultar si un grupo determinado de sensores se ha actualizado tras la última lectura de sensores. Como parámetro hay que indicarle una constante con el grupo de sensores que se quiere consultar. Si el parámetro es correcto retorna un 1 si el grupo de sensores se actualizó tras la última lectura de sensores y retorna un 0 si el grupo de sensores no se actualizó. En caso de que el parámetro sea incorrecto retorna 255.

Las constantes para el parámetro de grupo son las siguientes:

E_ACT_MTALFA	(Motores alfa)	*E_ACT_MTBETA*	(Motores beta)
E_ACT_PTACTO	(Posiciones de tacto)	*E_ACT_FTACTO*	(Fuerzas de tacto)
E_ACT_INFRAR	(Infrarrojos)	*E_ACT_INCLIN*	(Inclinómetro)
E_ACT_PIR_23	(*Piros* 2 y 3)	*E_ACT_PIR145*	(*Piros* 1, 4 y 5)
E_ACT_BTELEC	(Batería de electrónica)	*E_ACT_BTMOTR*	(Batería de motores)
E_ACT_WHISKR	(Whiskers)		

E_WriteQueDatos permite escribir en el buffer de usuario de petición de tipos de sensores para indicar qué sensores se desea recibir. Como parámetros hay que indicarle el grupo o grupos de sensores que se desea modificar (permite indicar más de un grupo mediante OR lógicas de las constantes) y el valor. Un uno para un determinado grupo indica que se desea recibir información de esos sensores y un cero indica lo contrario. Como resultado devuelve un valor que indica que no ha habido error en su ejecución o un valor con el tipo de error correspondiente. Se ha implementado como una función en lugar de como macro ya que no se espera que se utilice demasiado.
Las constantes para el parámetro de grupo son las siguientes:

E_ALL_DAT	(Todos los grupos de sensores)	*E_DAT_WSK*	(*Whiskers*)
E_DAT_BTR	(Baterías: de motores y de electrónica)	*E_DAT_PIR*	(*Piros*)
E_DAT_INC	(Inclinómetro)	*E_DAT_INF*	(Infrarrojos)
E_DAT_FTC	(Fuerzas de tacto)	*E_DAT_PTC*	(Posiciones de tacto)
E_DAT_MTB	(Motores beta)	*E_DAT_MTA*	(Motores alfa)

E_ReadQueDatos permite consultar el buffer de usuario de petición de tipos de sensores para comprobar el estado de un grupo concreto (si se desea o no). Como parámetros hay que indicarle el grupo de sensores que se desea consultar y una variable donde devuelva el valor correspondiente. El valor de retorno tiene el mismo significado que para *E_WriteQueDatos*, y también se ha implementado como una función por el mismo motivo.

A continuación se describe la interficie para los *servicios de protocolo*:

E_Open permite abrir el *Nivel de Enlace*. Inicializa los parámetros de este nivel y de los niveles inferiores (en este caso *Nivel Físico*). No tiene parámetros y antes de volver a ejecutarla hay que cerrar el *Nivel de Enlace*. Las posiciones iniciales para el robot son las que ya se comentaron anteriormente en el punto 7.3.3 y no se recibe información de ningún sensor. Para poder indicar en qué puerto de comunicaciones del PC se desea mantener la comunicación con el robot se dispone de un fichero de configuración denominado *prtcgeng.cfg*. Es un fichero de texto (*ASCII*) y el carácter indica el puerto de comunicaciones deseado, que puede ser desde el 0 al 4. Si el fichero no existe por defecto se toma el puerto de comunicaciones número 1 (*COM* 1).

E_Close permite cerrar el *Nivel de Enlace*. No tiene parámetros.

E_CaracterIndication es la función de interficie que permite al nivel inferior (en este caso *Nivel Físico*) indicar al *Nivel de Enlace* que ha llegado un nuevo carácter y cuál es. Como parámetro hay que indicar dicho carácter, en caso de que sea un inicio de trama el *Nivel de Enlace* envía datos al robot.

E_Write permite dar la orden al *Nivel de Enlace* para que se envíen al robot los datos disponibles hasta el momento en los *buffers* de usuario. Si se han escrito órdenes en el *buffer* de usuario o se ha realizado una nueva petición de datos se enviarán al robot. No tiene parámetros.

E_Read permite realizar una lectura de sensores. Como parámetro hay que pasarle un entero donde devuelve un uno si se ha recibido nueva información de sensores o un cero en caso contrario. En caso de haber recibido nueva información de sensores (última trama correcta detectada) la información de los nuevos sensores queda actualizada en el buffer de usuario de sensores.

E_EstCom permite consultar ciertos indicadores booleanos que informan sobre el estado de la comunicación: si las órdenes no se han podido enviar dentro del ciclo porque el buffer de salida se estaba actualizando justo en el momento en que tocaba enviar (órdenes no enviadas dentro del ciclo), si se ha producido un error al enviar la trama de órdenes al *Nivel Físico*, si se ha detectado más de una trama en una lectura de sensores (*overrun* de sensores ya que sólo se coge la última), si se ha pedido enviar órdenes antes de haber podido enviar las anteriores (*overrun* de órdenes, solo se enviarán las últimas), si el número de tramas detectadas (número de cabeceras recibidas) no es el óptimo, si se ha producido un *timeout* (varios ciclos sin recibir tramas de sensores) o si se ha producido un *timeout* por petición de datos (no se ha recibido contestación con el tipo de sensores adecuado tras una petición de nuevos sensores). Hay que pasarle una constante que dice qué indicador se desea consultar (pueden ser todos o sólo uno concreto) y una variable para guardar el valor de dicho parámetro. En caso de que se le pida el valor de un único parámetro retornará un uno si el error o indicación se ha producido o cero en caso contrario. Si se le pide consultar todos los parámetros se devuelve un valor cuyos bits están a cero o a uno indicando el estado de todos los indicadores y se pueden consultar haciendo una máscara (una AND lógica) con las mismas constantes que sirven para decir qué indicador se desea consultar.
Las constantes para informar del indicador a consultar son las siguientes:

E_ALL_ERR_IND	(Consultar todos errores o indicaciones)
E_IND_ENV_CICLO	(Órdenes no enviadas dentro del ciclo)
E_ERR_ENV	(Error al enviar la trama de órdenes)
E_OVERR_SENSORES	(Varias tramas detectadas en una lectura)
E_OVERR_ORDENES	(Nueva petición de envío antes de poder enviar la orden anterior)
E_IND_NO_OPTIMO	(Número de tramas detectadas menor que el número óptimo)
E_ERR_TIMEOUT	(*Timeout* sin recibir tramas)
E_ERR_PET_DAT	(*Timeout* sin recibir tramas con los datos pedidos)

E_State permite consultar una variable de estado de *Nivel de Enlace* o de *Nivel Físico*. Como parámetro hay que indicar la variable que se desea consultar y un valor donde se guardará el valor de dicha variable. Las variables de estado de *Nivel Físico* ya se vieron en el punto 6.3.3 y en cuanto a las de *Nivel de Enlace* disponemos de: número de tramas detectadas, número de tramas correctas, número de tramas erróneas, número de errores por tipo de trama incorrecto, número de errores por *checksum* incorrecto, número de *timeouts* producidos y número de *timeouts* de petición de datos.
Las constantes para el parámetro que indica la variable de estado a consultar son las siguientes:

F_ERROR_OVERRUN	(Contador de errores de *overrun*)
F_ERROR_PARIDAD	(Contador de errores de paridad)
F_ERROR_TRAMA	(Contador de errores de trama)
F_ERROR_BREAKLINE	(Contador de errores de *breakline*)
F_ERROR_OVERRUN_BUFFER	(Contador de errores *overrun buffers*)
E_TRAMAS_DETECTADAS	(Contador de tramas detectadas)
E_TRAMAS_CORRECTAS	(Contador de tramas correctas)
E_TRAMAS_ERRONEAS	(Contador de tramas erróneas)
E_ERRORES_TIPO_TRAMA	(Contador de errores por tipo trama incorrecto)
E_ERRORES_CHECKSUM	(Contador de errores por *checksum* incorrecto)
E_NUM_TOUT	(Contador de timeouts)
E_NUM_TOUT_PET_DAT	(Contadorde timeouts de petición datos)

E_ResetRob permite hacer un *reset* del robot por software. Los actuadores van a su posición inicial (30 para los motores y 0 para el resto).

7.4. PRUEBAS REALIZADAS Y CONCLUSIONES SOBRE NIVEL DE ENLACE

En cuanto a las pruebas realizadas a este nivel se ha probado a desconectar el cable o provocar un *reset* en mitad de la comunicación, realizar menos de una lectura por ciclo (de manera que se detectara más de una trama por

lectura), hacer el *buffer* de salida de *Nivel Físico* en el *PC* muy pequeño para que diera errores al enviar la trama por falta de espacio, hacer una petición de datos con el cable desconectado de manera que el robot no se enterara, etc …

En todos los casos las indicaciones de errores o situaciones no esperadas han funcionado correctamente y el comportamiento del protocolo ha sido el esperado. Con todo esto podemos concluir que el resultado obtenido cumple los objetivos que se marcaron inicialmente y el protocolo permite una correcta comunicación entre el robot y el *PC*.

8. APLICACIÓN

Uno de los requerimientos del proyecto era la realización de una aplicación que implementara un entorno para la adquisición de datos de los sensores del robot y que permitiera el control remoto de sus actuadores.

La aplicación que se ha realizado permite consultar el estado de la comunicación para cada uno de los niveles (*Físico* y *Enlace*), consultar el estado de los sensores del robot pudiendo indicarle qué sensores se desea recibir en cada momento y controlar remotamente cada uno de sus actuadores por separado.

La interficie con el usuario se realiza a través de una única pantalla dividida en tres zonas, de manera que las dos primeras muestran el estado de la comunicación en los niveles *Físico* y *Enlace* respectivamente y la siguiente (es la única interactiva) muestra los datos de sensores y permite el control de los actuadores mediante el teclado.

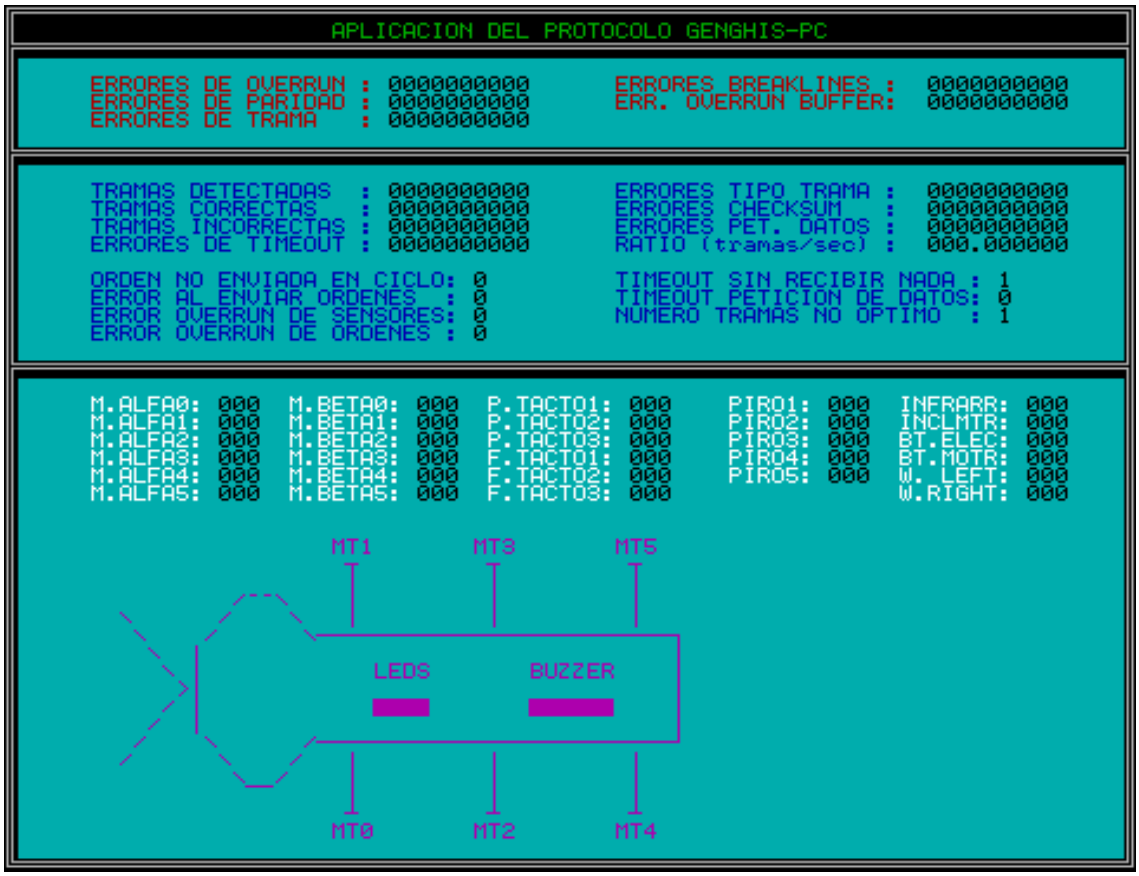

Figura 8.1. *Estado inicial de la aplicación.*

- Estado de *Nivel Físico*: Es la zona situada en la parte superior de la pantalla. En ella se pueden ver los contadores de errores pertenecientes al *Nivel Físico*: número de errores de *overrun*, paridad, trama, *breakline* y *overrun* de buffer (para más información sobre este tipo de errores se puede consultar el anexo 5). No es interactiva, sólo ofrece la información indicada.

- Estado de *Nivel de Enlace*: Es la zona situada bajo la anterior y ofrece información sobre las variables de estado de *Nivel de Enlace* y el estado de la comunicación. En la parte alta de esta zona se muestran los contadores de trama detectadas, tramas correctas, número de errores de timeout, errores por tipo de trama

incorrecto, errores por checksum incorrecto y el ratio de recepción de tramas. Este ratio indica el número de tramas detectadas (concretamente el número de cabeceras detectadas) desde el inicio de la aplicación, por lo que si en todo momento el flujo ha sido continuo y al ritmo correcto marcará 50 aproximadamente (una trama cada 20 milisegundos) y si no este valor normalmente será menor (en caso de que hubiera caracteres erróneos que se confundieran con el valor de la cabecera podría ser mayor). En cuanto a los indicadores sobre el estado de la comunicación se ponen a uno o a cero a cada ciclo dependiendo de la situación producida respecto al tipo de error y por tanto su variación en algunos casos puede ser demasiado rápida para verla con claridad por pantalla. No obstante se han incluido para que se pueda observar su funcionamiento (para más información sobre estas indicaciones sobre el estado de la comunicación ver el punto 7.3.2).

- Interficie de control: Es la zona situada en la parte baja de la pantalla y la que ocupa mayor espacio. Permite recibir información sobre el estado de los sensores y el control de los actuadores del robot.
  En la parte superior de la pantalla tenemos los indicadores del valor de cada uno de los sensores. Cuando se recibe información de un determinado grupo de sensores tras la lectura de una trama, su valor se actualiza y el nombre aparece destacado en amarillo, en caso de no disponer de información sobre ese sensor el valor no se actualiza y el nombre aparece en blanco (sin destacar). Inicialmente y por defecto se pide que no se envíe ningún tipo de información de sensores (el protocolo se inicializa así). Para la modificación de los datos a pedir disponemos de las teclas de control (*F1*-*F10*) de manera que se puede activar o desactivar la petición de un grupo concreto de sensores. De todos modos no se enviará la trama con la petición al robot hasta que el usuario lo indique pulsando *RETURN* después de haber realizado las modificaciones deseadas. La función de cada tecla es la que se describe a continuación:

F1:	Activa la petición de datos de fuerza de motores alfa.
SHIFT+F1:	Desactiva la petición de datos de fuerza de motores alfa.
F2:	Activa la petición de datos de fuerza de motores beta.
SHIFT+F2:	Desactiva la petición de datos de fuerza de motores beta.
F3:	Activa la petición de datos de posiciones de tacto.
SHIFT+F3:	Desactiva la petición de datos de posiciones de tacto.
F4:	Activa la petición de datos de fuerzas de tacto.
SHIFT+F4:	Desactiva la petición de datos de fuerzas de tacto.
F5:	Activa la petición de datos de *piros*.
SHIFT+F5:	Desactiva la petición de datos de *piros*.
F6:	Activa la petición de datos de infrarrojos.
SHIFT+F6:	Desactiva la petición de datos de infrarrojos.
F7:	Activa la petición de datos de *inclinómetro*.
SHIFT+F7:	Desactiva la petición de datos de *inclinómetro*.
F8:	Activa la petición de datos de baterías (de electrónica y de motores).
SHIFT+F8:	Desactiva la petición de datos de baterías.
F9:	Activa la petición de datos de *whiskers*.
SHIFT+F9:	Desactiva la petición de datos de *whiskers*.
F10:	Activa las peticiones de todos los grupos de sensores.
SHIFT+F10:	Desactiva las peticiones de todos los grupos de sensores.
RETURN:	Confirma que se desea realizar una petición de nuevo tipo de datos al robot.

Por ejemplo si se desea recibir información de sensores de motores beta, *piros* y *whiskers* pulsaríamos la siguiente secuencia de teclas: *F2 + F4 + F9 + RETURN*.

En cuanto al control del robot disponemos de 9 estados de control. Se puede cambiar de un estado a otro mediante la tecla TAB (tabulador).
En el primer estado (es el inicial) el control está desactivado y por tanto no se puede enviar órdenes a los actuadores. Los siguientes 6 estados permiten el control de los motores de cada una de las 6 patas, desde la pata 0 a la pata 5 respectivamente, y los dos estados restantes permiten el control de los *leds* y del *buzzer* en ese orden. En cada una de las posiciones se muestran los valores de órdenes de los actuadores correspondientes que hay en el *buffer* de órdenes de usuario.

En cada uno de los estados de control de las patas disponemos de dos opciones: indicar a cada uno de los dos motores un valor concreto o moverlos en tiempo real mediante los cursores.
Para indicar un valor concreto a los motores hay que introducir el valor que se desea, pulsando A para los motores alfa y B para los beta, dentro del rango 0-63. Si el valor indicado no está dentro de este rango se vuelve

a esperar un valor correcto y en caso de no querer indicar ningún valor, pulsando *RETURN* se aborta la introducción del valor de manera que se mantiene el que ya existía. Para que los valores modificados se traduzcan a una orden al robot hay que pulsar *RETURN* (del mismo modo que para las peticiones de datos).
En el caso del control en tiempo real, los cursores hacia arriba y abajo nos permiten mover el motor *beta* de la pata seleccionada y los cursores izquierdo y derecho nos permiten mover el motor *alfa*. Como los motores no están montados con la misma orientación el sentido del movimiento varía de algunos motores a otros. Lo que se hace es incrementar (cursor derecho o hacia arriba) o decrementar (cursor izquierdo o hacia abajo) el valor de la orden al motor correspondiente (en el *buffer* de órdenes) en una unidad y enviar una orden al robot sin necesidad de pulsar *RETURN*.

En los estados de control de los *leds* y *buzzer* disponemos también de la opción de indicar un valor determinado o de control en tiempo real. En este caso para indicar un valor hay que pulsar L en el caso de los *leds* y Z en el caso del *buzzer*. Los valores en ambos casos pueden ir de 0 a 255 y si el valor no está en este rango, igual que para los motores, se vuelve a esperar un nuevo valor hasta que se indique uno correcto o se pulse *RETURN* para abortar la modificación del valor. Para que los valores modificados se envíen al robot hay que pulsar *RETURN*.
En el caso de control en tiempo real sólo disponemos de los cursores hacia arriba y abajo que incrementan el valor en una unidad, el funcionamiento es el mismo que en el caso de los motores.

Por último comentar que el hecho de que las modificaciones de los valores de los actuadores no se envíen al robot hasta que se pulsa *RETURN* o alguno de los cursores (en un estado de control), nos permite modificar varios valores de los actuadores y de petición de datos y enviarlo todo de una sola vez.

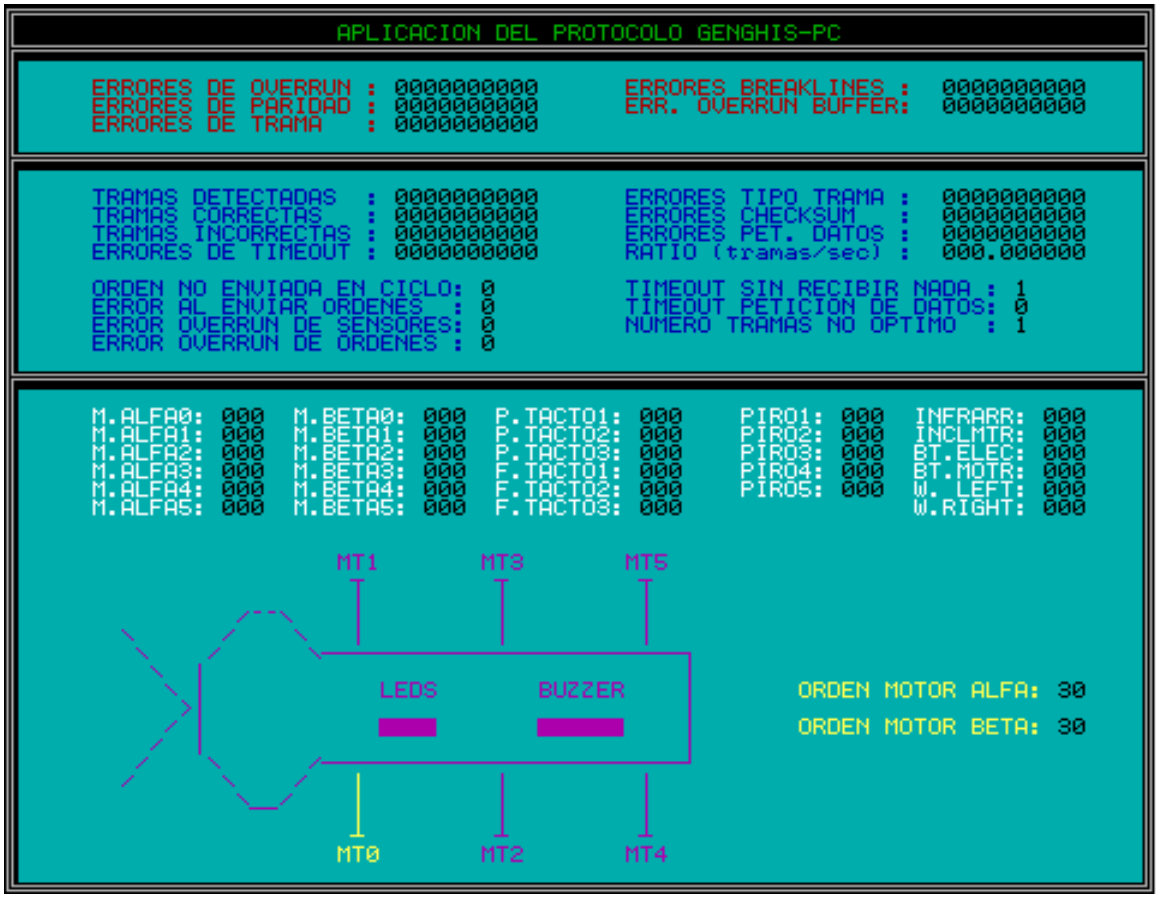

***Figura 1.1**. Estado de control de la pata cero.*

9. ESTUDIO ECONÓMICO

El proyecto no se ha orientado en ningún momento hacia una posible comercialización, más bien al contrario, ya que el diseño y la implementación se han dirigido en todo momento hacia las necesidades específicas que tendrán los programas de usuario que se quieren realizar en el *IRI*.

Su carácter específico lo aleja, en parte, de una posible comercialización pero, teniendo en cuenta que existen más unidades del *Genghis II* y que los propietarios pueden tener las mismas necesidades que motivaron este proyecto, la idea de comercialización puede tener sentido. De hecho el *Genghis* se diseñó como un robot didáctico para la investigación y en un primer momento tampoco se pensó en su comercialización.

Por tanto, nuestro estudio económico se centrará en un cálculo de costes de realización y un posible precio de venta para la comercialización del sistema de comunicaciones entre el *Genghis II* y un *PC*.

En cuanto al coste del proyecto, el único material que se ha necesitado ha sido el utilizado para la construcción del cable de comunicaciones. El resto de gastos corresponden a horas de diseño, programación y gastos diversos (electricidad, impresión, …) A continuación se muestra una tabla con la estimación aproximada de los gastos:

CONCEPTO	**CANTIDAD**
350 horas de trabajo de analista (2500 pts. Por hora)	875.000
170 horas de trabajo de programador (2000 pts. por hora)	340.000
5 metros de cable, conector de teléfono de 6 vías, conector de línea serie de 9 *pins*, material para soldadura,…	500
1 hora de trabajo para construcción y comprobación del cable	1.500
Gastos varios de laboratorio: Electricidad, impresión, …	28.000
TOTAL	**1.245.000**

Aunque no tenemos noticias del total de unidades del *Genghis* II que se han vendido, sabemos que en el mismo año en que se compró la unidad del robot objeto del proyecto, por lo menos, se vendieron otras cinco anteriores. Basándonos en este dato, en que el robot se empezó a comercializar a principios de los 90 y en que la unidad con la que se ha trabajado se adquirió en 1994, suponemos que podrían haberse vendido un número aproximado de unas 20 unidades como mínimo.

Si simplemente quisiéramos amortizar los gastos, como mínimo deberíamos vender el producto a un precio de 62.250 pesetas (más *IVA*). Si, por otra parte, se deseara obtener un beneficio económico con la venta del producto este precio debería ser mayor según el margen de beneficios que se deseara obtener. Suponiendo que deseáramos obtener un beneficio del 20% con cada una de las ventas, el precio de venta debería ser de 74.700 (más *IVA*) obteniendo un beneficio de 12.450 pesetas.

Como ya se ha indicado a estos precios habría que añadirles el tanto por ciento de *IVA* correspondiente y los posibles gastos de envío, de embalaje, etc… según el caso. El coste para el cliente podría variar según el estado del cambio de moneda si el posible cliente fuera extranjero.

Como conclusión, decir que se obtiene un posible precio de venta que puede ser razonable si se tiene en cuenta las enormes ventajas que ofrece poder conectar el robot a un *PC*. Con este precio podría ser un producto interesante para aquellas personas o instituciones que realicen gran cantidad de proyectos con este robot o que deseen realizarlos y actualmente se vean limitados igual que sucedía en nuestro caso. En cualquier caso, no es un precio elevado si se tiene en cuenta que el gasto realizado en la adquisición de un robot de este tipo ronda el millón y medio de pesetas.

10. CONCLUSIONES

Llegados a este punto solo nos queda realizar una valoración global del trabajo realizado y el resultado obtenido. Como hemos ido viendo a lo largo de esta memoria, los objetivos para cada nivel se han ido cumpliendo en su totalidad por lo que el resultado final se considera totalmente satisfactorio.

Se ha obtenido un protocolo robusto y ágil para la comunicación entre el *Genghis II* y el *PC* en tiempo real, que servirá como punto de partida para desarrollar nuevos proyectos con el robot. Los objetivos iniciales no solo se han cumplido sino que incluso se han superado, ya que se ha conseguido detectar y solucionar problemas internos que se desconocían anteriormente y conocer a fondo la manera en que funciona internamente el robot.

Concretamente los problemas detectados tienen que ver con fallos en el código del microcomputador *SERVO* del robot. En principio no se esperaba (y tampoco se deseaba) modificar el código de este micro, ya que no se consideraba parte del proyecto, pero tras comprobar que su funcionamiento no era el esperado fue necesario retocarlo. Esto ha retrasado la realización del proyector respecto a las previsiones iniciales pero era fundamental que el *SERVO* funcionara correctamente ya que de no ser así, y por la dependencia del resto de micros respecto a éste, el protocolo nunca habría funcionado perfectamente. Para más información sobre los problemas con dicho micro se puede consultar el anexo 9.

Afortunadamente se ha conseguido solucionar todos los problemas y se ha obtenido un protocolo que funciona correctamente. Todas las pruebas realizadas con la aplicación de ejemplo implementada han sido totalmente correctas. También se ha probado con programas que permiten al robot caminar tanto por terreno liso como por terreno accidentado. Los resultados obtenidos han sido igual de buenos y el protocolo responde perfectamente permitiendo un comportamiento correcto del robot.

Como ya se ha comentado el protocolo va a ser la base y pieza fundamental para poder desarrollar futuros proyectos con el robot. A partir de este momento, dada la gran capacidad de cálculo de los *PC's* y la gran cantidad de memoria de la que se puede llegar a disponer, se van a poder desarrollar aplicaciones mayores y más complicadas ya que se tendrá más memoria disponible y en el mismo tiempo que antes (20 milisegundos) se podrá ejecutar mucho más código. Por tanto, se van a poder desarrollar proyectos con el robot que antes era totalmente imposible plantearse por las limitaciones que presentaba.

Por último comentar que el protocolo se ha diseñado e implementado para funcionar directamente bajo *MS-DOS*. Aun así se han realizado pruebas sobre su comportamiento con *MS-DOS* bajo *WINDOWS95* y, aunque con limitaciones, se ha comprobado que también funciona.

BIBLIOGRAFÍA

[1] Motorola Inc. *"MC68HC11A8: HCMOS Single-Chip Microcomputer"*, Motorola Inc., 1987.

[2] Stallings W. *"Data and Computer Communications",* 4a edición, Ed. Mcmillan, New York, 1994.

[3] Tischerr M. "PC Interno 2.0: Programción del Sistema" Ed. Marcombo, 1995.

DIRECCIONES DE INTERNET

CK Corporation (información sobre switch y pulsador):
http.//www.ckcorp.com

Información sobre la línea serie del PC:
ftp://ftp.phil.uni-sb.de/pub/staff/chris/The_Serial_Port
http://www.geocities.com/SilliconValley/Bay/8302

IRI:
http://www-iri.upc.es

IS Robotics:
http://www.isr.com

MIT:
http://www.ai.mit.edu
http://www.ai.mit.edu/projects/genghis

ANEXO 1. ESTÁNDAR RS-232

El estándar *RS-232* es muy amplio y por lo tanto no se pretende realizar un estudio completo, sino una descripción de aquellas características que ayuden a una mejor compresión de funcionamiento de la comunicación asíncrona por línea serie. Para un estudio en profundidad o más detallado se recomienda consultar el estándar.

Este estándar fue presentado en 1962 por *Electronic Industries Asociation (EIA)* y fue rápidamente adoptado en la industria. La tercer revisión, *RS-232-C*, aparecida en 1969 alcanzó una gran popularidad y en 1987 se presentó una nueva revisión denominada *RS-232-D*. Además de este estándar aparecieron varios más que intentaron mejorar las limitaciones de éste en cuanto a velocidad de transmisión y distancia (*RS-448*, *EIA-530*, *RS-442*, *RS-423*, etc...), pero ninguno ha alcanzado la popularidad de éste.

En general, especifica parámetros mecánicos, eléctricos, funcionales y procedurales para la transmisión asíncrona de "caracteres". Que la transmisión sea asíncrona quiere decir que con los datos no se envía ninguna señal de reloj sino que el receptor se sincroniza mediante un bit de *start* y a partir de ese momento se realiza un muestreo del resto de bits al ritmo que marca un reloj interno.

Los "caracteres" se envían, según una codificación *NRZ-L* [11], como un conjunto de "marcas" (unos lógicos y valor de voltaje negativo) y "espacios" (ceros lógicos y valor de voltaje positivo), empezando por el bit de menos peso hasta el bit de mayor peso consecutivamente. Inicialmente se envía un bit de *start* (cero lógico) que indica el inicio de un "carácter" y que sirve para que el receptor se sincronice para muestrear el resto de bits. A continuación siguen entre 5 y 8 bits de datos y un bit de paridad opcional (paridad par, impar o sin paridad) para la detección de posibles errores. Finalmente hay un bit de *stop* (uno lógico de longitud variable correspondiente a 1, 1,5 o 2 bits) que sirve para separar del siguiente carácter que puede venir a continuación.

En la figura A1.1 se muestra el formato indicado. Como podemos observar la línea en estado de reposo (o *Idle*) tiene un valor correspondiente a un cero lógico (espacio).

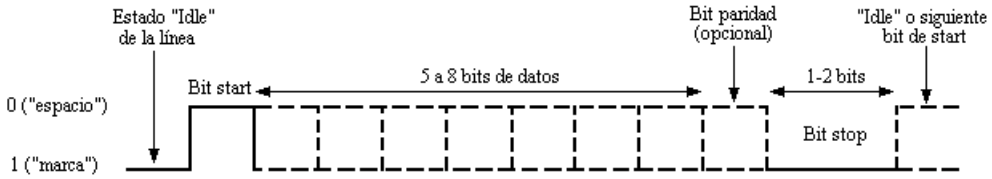

Figura A1.1. *Formato de carácter RS-232.*

Es muy importante que para un correcto muestreo de los bits en la recepción, las velocidades de transmisión y recepción sean compatibles. No han de ser estrictamente iguales, ya que pueden presentar pequeñas diferencias (en general no superiores al 3%) y obtener un funcionamiento correcto en cuanto al muestreo de los bits en la recepción. La figura A1.1 muestra un ejemplo de un muestreo erróneo en la recepción cuando las velocidades de funcionamiento son diferentes:

[11] Non Return To Zero - Level

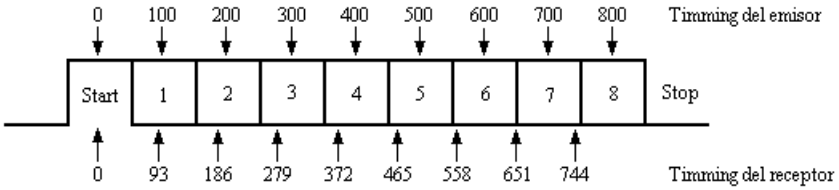

Figura A1.2. *Ejemplo de muestreo incorrecto por diferencia de velocidad.*

En cuanto a los niveles eléctricos el estándar define los siguientes rangos:

- Un uno lógico o "marca" se traduce a un valor de voltaje negativo que puede ir de -3 a -25 voltios.
- Un cero lógico o "espacio" se traduce a un valor de voltaje positivo que puede ir de +3 a +25 voltios.
- La zona entre -3 y +3 voltios queda indefinida.

ANEXO 2. MOTOROLA MC68HC11A2FN

A continuación se realiza un estudio de los aspectos del *MOTOROLA MC68HC11A2FN* que se han utilizado para la implementación del proyecto y que son importantes para una buena comprensión de los programas desarrollados para estos *micros*. No se pretende entrar a fondo en cada uno de los aspectos descritos, por lo tanto, para un estudio detallado será necesario referirse al manual del *micro* [1].

Descripción

El *MOTOROLA MC68HC11E2FN* es un microcomputador de 8 bits basado en tecnología *HCMOS* [12]. Puede direccionar un total de 64K de memoria e internamente incluye 256 bytes de memoria RAM, ampliables externamente [13] (hasta un máximo que complete todo el espacio direccionable), y 2K de *EEPROM* [14]. También incorpora un sistema de *timmer* de 16 bits, un sistema de comunicación asíncrono y *full-duplex* basado en el estándar *NRZ* [15] denominado *Serial Communication Interface (SCI)* (que habitualmente conocemos por línea serie) y un sistema de comunicación síncrona *full-duplex* denominado *Serial Periferal Interface (SPI)* que permite comunicar *micros* (que posean esta interficie) entre sí. Dispone de un sistema de conversión Analógico-Digital (*A/D*) de 8 bits y 8 canales, circuitería para interrupciones en tiempo real y sistema de *Watchdog* [16]. A continuación se muestra un diagrama de bloques genérico para la familia *MC68HC11*:

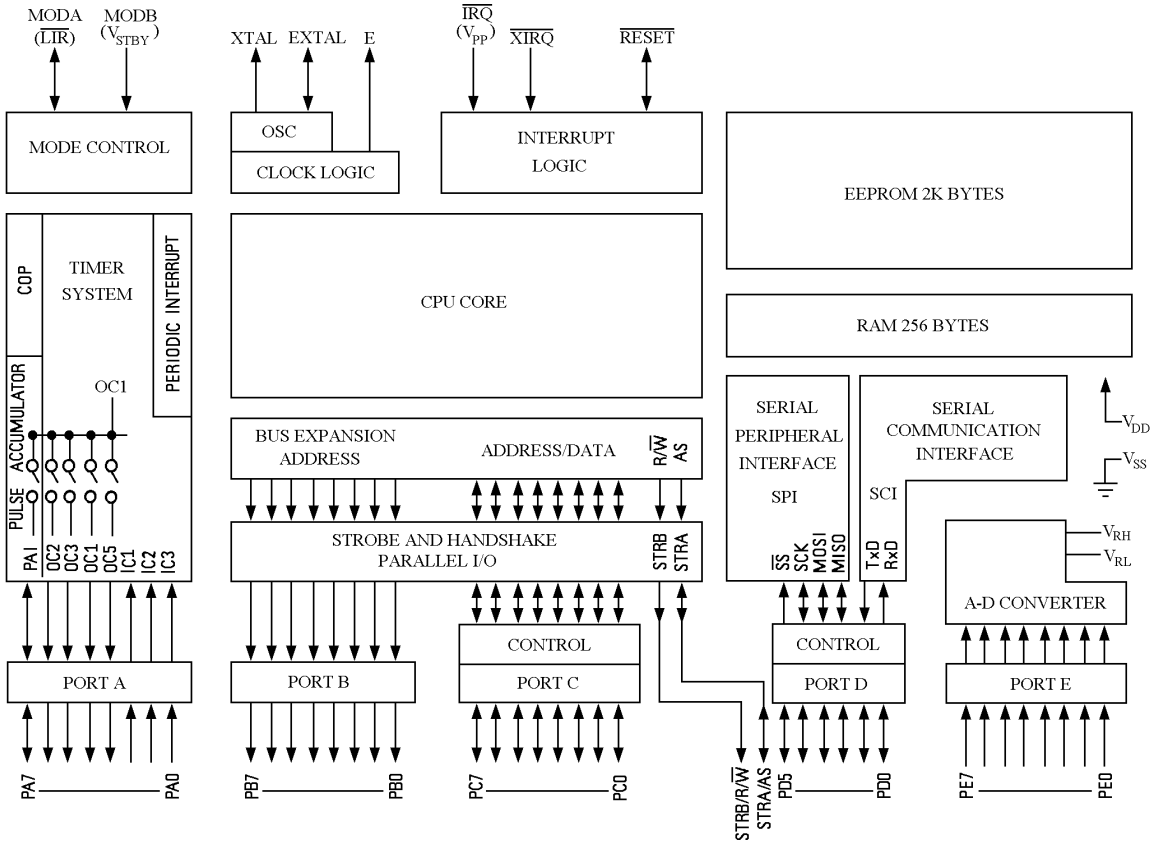

***Figura A2.1**. Diagrama de bloques del MOTOROLA MC68HC11A2FN*

[12] *High-Density CMOS tecnology.*
[13] El procesador *MAIN* tiene 32K de *SRAM* no volátil para programas de usuario.
[14] *Electric Erasable Programmable Read Only Memory.*
[15] *Non Return to Zero.*
[16] Sistema de protección contra errores de software.

Para la programación del *micro* disponemos de un total de 7 registros. Dos de propósito general, denominados *acumuladores A* y *B*, de 8 bits que se pueden utilizar como uno solo de 16 bits denominado *doble acumulador D*. Dos registros de índice de 16 bits denominados *registro de índice X (IX)* y *registro de índice Y (IY)*, un registro de pila denominado *SP (Stack Pointer)*, un registro que contiene la dirección de la próxima instrucción a ejecutar denominado *PC (Program Counter)* y un registro de *flags* denominado *Condition Code Register (CCR)*.

El *micro* dispone de cuatro modos de funcionamiento diferentes: *Single-chip*, *Expanded Multiplexed*, *Special Bootstrap* y *Special Test*. El modo de funcionamiento se determina automáticamente después de un *reset* dependiendo del valor de los *pins MODA* y *MODB* (ver el diagrama de bloques de la figura A2.1).

- En el modo *Single Chip* funciona como un *chip* aislado sin direcciones ni *buses* de datos externos.
- En el modo *Expanded Multiplexed* es capaz de direccionar un total de 64K entre direcciones internas y externas, y por tanto permite conectarle dispositivos periféricos o memorias externas.
- El modo *Boostrap* sirve para cargar un determinado programa en la *EEPROM* del *chip* a través de la *SCI*.
- El modo Test es un modo de "testeo" del chip, pensado originariamente para pruebas de fábrica, en el que las limitaciones de protección del *micro* son muy bajas. Es similar al *Expanded Multiplexed* y no debería utilizarse nunca debido a la reducida seguridad del sistema.

Puertos

El *MC68HC11E2FN* dispone de un total de cinco puertos (del *A* al *E*) para la comunicación con el exterior. La función de los puertos *B* y *C* depende del modo de funcionamiento del micro, mientras que la de los puertos *A*, *D* y *E* es independiente. Los *pins* de los puertos *C* y *D* se pueden configurar independientemente como entradas o salidas según se desee, en cambio los *pins* del resto de puertos son fijos (ya sean entradas o salidas).

- El puerto *A* se puede configurar para tres funciones de entrada, cuatro de salida y una de entrada o salida, relacionadas con el *timmer*. Aquí no entraremos en el funcionamiento de este sistema, por lo que si se desea más información habrá que consultar el manual [1].
- El puerto *B* en modo *Single Chip* proporciona 8 salidas de carácter general y en modo *Expanded Multiplexed* proporciona 8 bits de direcciones (los 8 de mayor peso).
- El puerto *C* en modo *Single Chip* proporciona 8 entradas/salidas de carácter general y en modo *Expanded Multiplexed* sirve para la multiplexación de los 8 bits de direcciones de menor peso y de los 8 bits de datos.
- Los *pins* del puerto D se pueden utilizar para funciones generales de entrada/salida pero también se utilizan para los sistemas *SCI* y *SPI* cuando éstos están activos. En el diagrama de bloques de la figura A2.1 se puede ver la disposición de los *pins*.
- El puerto *E* se puede configurar para funciones de entrada de propósito general y/o como canales del conversor *A/D*.

Serial Peripheral Interface (SPI)

La *Serial Peripheral Interface* es una interficie síncrona que permite conectar entre sí microcomputadores o periféricos que dispongan de dicha interficie. Se caracteriza por ser *full-duplex*, permitir dos modos de funcionamiento para la comunicación (*MASTER* o *SLAVE*), proporcionar varias posibilidades en cuanto a frecuencia de funcionamiento y ofrecer la posibilidad de interrupción por fin de transmisión entre otras características.

Hay cuatro señales básicas: *MISO*, *MOSI*, *SCK* Y */SS* conectadas al puerto D, de manera que el registro de control de dirección de dicho puerto debe tener los bits correspondientes a los *pins* de dichas señales configurados como salidas (con un uno).

La señal *Master Input Slave Output (MISO)* es una de las dos líneas para transmisión de datos. Envía datos en uno de los dos posibles sentidos, empezando por el bit más significativo.
La señal *Master Output Slave Input (MOSI)* es la otra de las dos líneas de transmisión de datos existente.

La señal *Serial Clock (SCK)* se utiliza para sincronizar el intercambio de datos entre *MASTER* y *SLAVE*. La genera el *MASTER* (en el *SLAVE* es una señal de entrada) y se puede configurar entre cuatro posibilidades diferentes en cuanto a polaridad y fase.
La señal *Slave Select (/SS)* permite la selección por parte de un *MASTER* de un dispositivo *SLAVE* determinado. Cuando esta señal está a cero el dispositivo *SLAVE* está seleccionado para la comunicación.

El dispositivo MASTER es el que dirige la comunicación con el resto, de manera que él es el que decide cuándo se comunica y con quién. Cuando un dispositivo *MASTER* transmite datos a un dispositivo *SLAVE* (a través la línea *MOSI*) el *SLAVE* contesta enviando datos por la línea *MISO* automáticamente. Entonces el byte enviado por el *MASTER*, escrito en su registro de datos de *SPI*, se reemplaza por el byte recibido procedente del *SLAVE*. De igual modo el contenido del registro de datos del *SLAVE* (donde está el carácter que había enviado) se reemplaza con el byte enviado por el *MASTER*. A continuación se indica un esquema sobre el intercambio de datos:

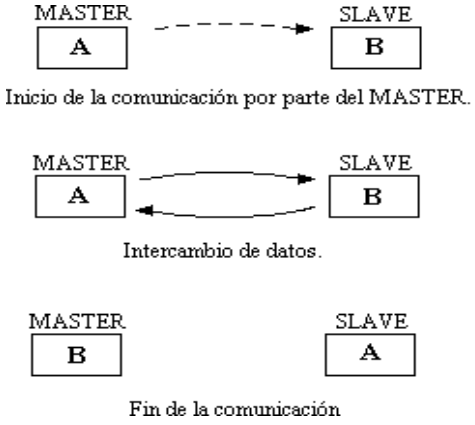

Figura a2.2. *Mecanismo de comunicación por SPI.*

Se dispone de un total de 3 registros para el control (*SPCR*), status (*SPSR*) y datos (*SPDR*) de SPI, que se describen a continuación.

El registro *Serial Peripheral Data I/O Register (SPDR)* se utiliza para la transmisión y recepción de datos a través del bus de *SPI*. Una escritura en este registro iniciará una transmisión y recepción de un nuevo byte, pero esto solo ocurre en un dispositivo *MASTER*. Cuando la transferencia a finalizado el bit *SPIF* del *SPSR* se activa tanto en el *MASTER* como en el *SLAVE*. Si se recibe un nuevo byte antes de haber leido el anterior del registro *SPDR* se producirá un overrun, de manera que el byte que causa esta situación se perderá.

SERIAL PERIPHERAL CONTROL REGISTER (SPCR)			
BIT	**NOMBRE**	**VALOR**	**DESCRIPCIÓN**
0	*SPR0 (SPI Clock Rate Selects bit 0)*	00	Reloj interno del sistema dividido por 2.
1	*SPR1 (SPI Clock Rate Selects bit 1)*	01	Reloj interno del sistema dividido por 4.
		10	Reloj interno del sistema dividido por 16.
		11	Reloj interno del sistema dividido por 32.
2	*CPHA (Clock Phase)*	0	Junto al bit *CPOL* permiten controlar la relación entre
		1	datos y reloj entre *MASTER* y *SLAVE* entre cuatro posibilidades (ver el manual para más detalles).
3	*CPOL (Clock Polarity)*	0	Estado de reposo del pin *SCK* = nivel bajo.
		1	Estado de reposo del pin *SCK* = nivel alto.
4	*MSTR (Master Mode Select)*	0	Modo *SLAVE*
		1	Modo *MASTER*
5	*DWOM (Port D Wire-Or Mode Option)*		Afecta a los 6 pins del puerto *D* a la vez.
		0	Salidas del puerto *D* como salidas *CMOS* normales.
		1	Salidas del puerto *D* como salidas tipo *Open-Drain*.
6	*SPE (Serial Peripheral System Enable)*	0	Interrupciones de *SPI* desinhibidas.
		1	Interrupciones de *SPI* inhibidas.
7	*SPIE (Serial Peripheral System Enable)*	0	Sistema de *SPI* desactivado.
		1	Sistema de *SPI* activado.

SERIAL PERIPHERAL STATUS REGISTER (SPSR)			
BIT	**NOMBRE**	**VALOR**	**DESCRIPCIÓN**
0	-	-	-
1	-	-	-
2	-	-	-
3	-	-	-
4	*MODF (Mode Fault)*	0 1	No hay conflicto por más de un master. Conflicto por la existencia de más de un master. Para ponerlo a cero hay que leer el registro *SPSR* seguido de una escritura del registro *SPCR*.
5	-	-	-
6	*WCOL (Write Collision)*	0 1	No hay colisión. Colisión: escritura en el *SPDR* mientras está teniendo lugar una transferencia. Se pone a cero leyendo el *SPSR* seguido de un acceso al *SPDR*.
7	*SPIF (SPI Transfer Complete Flag)*	0 1	Tranasferencia de datos no completa. Transferencia de datos completa. Si se pone a uno y el bit *SPIE* del *SPCR* está activado, se genera una interrupción de SPI. Para ponerlo a cero hay que leer el *SPSR* seguido de un acceso al *SPDR*. El acceso al registro *SPDR* está inhibido mientras el bit *SPIF* está activado y no se haya leido el registro *SPSR*.

Conversor Analógico-Digital (A/D)

El *MC68HC11E2FN* internamente incluye un conversor *A/D* de 8 bits y ocho canales, accesibles como dos bancos de cuatro canales (sólo se puede acceder a un banco en un determinado momento). Consta de dos *pins*, V_{RH} y V_{RL} , para los voltajes de referencia, de manera que un voltaje igual a V_{RH} se traduce a un valor de FFh y un voltaje igual a V_{RL} se traduce a un valor de 00h.

Presenta dos modos de funcionamiento, *Single Channel* y *Multiple Channel*, con dos variaciones posibles para cada uno (dependiendo del valor del bit *SCAN* del registro de control *ADCTL*).

Single Channel: En la primera variación (*SCAN*=0) el canal seleccionado se convierte cuatro veces guardando los resultados en los registros *ADR1* a *ADR4* respectivamente. Después de la cuarta conversión se espera hasta que se escribe un nuevo comando en el registro *ADCTL*. En la segunda variación (*SCAN*=1) se realizan conversiones de canal seleccionado indefinidamente y los resultados se van guardando de *ADR1* a *ADR4* respectivamente, de manera que después de las primeras cuatro conversiones se vuelve a empezar por *ADR1*, *ADR2*, etc…

Multiple Channel: En la primera variación (*SCAN*=0) los cuatro canales del banco seleccionado se convierten una única vez, guardando los resultados en los registros *ADR1* a *ADR4*. En la segunda variación (*SCAN*=1) se realizan conversiones indefinidamente, guardando los valores de los registros *ADR1* a *ADR4*, sobreescribiendo los valores anteriores después de cada nueva conversión.

Disponemos de un total de cinco registros, uno de control del A/D (*ADCTL*) y cuatro para los resultados de las conversiones (*ADR1*, *ADR2*, *ADR3* y *ADR4*). Además existen dos bits del registro *OPTION* para la activación o no del *A/D* y selección del clock de muestreo. A continuación se indica la asignación de bits de estos registros.

A/D CONTROL/STATUS REGISTER (ADCTL)			
BIT	**NOMBRE**	**VALOR**	**DESCRIPCIÓN**
0	*CA (Channel Select A)*	0000 0001 0010 0011	Canal AN0 - Resultado si MULT=1 en ADR1. Canal AN1 - Resultado si MULT=1 en ADR2. Canal AN2 - Resultado si MULT=1 en ADR3. Canal AN3 - Resultado si MULT=1 en ADR4.
1	*CB (Channel Select B)*	0100 0101 0110 0111	Canal AN4 - Resultado si MULT=1 en ADR1. Canal AN5 - Resultado si MULT=1 en ADR2. Canal AN6 - Resultado si MULT=1 en ADR3. Canal AN7 - Resultado si *MULT=1* en *ADR4*.
2	*CC (Channel Select C)*	1000 1001 1010 1011	Reservado - Resultado si *MULT=1* en *ADR1*. Reservado - Resultado si *MULT=1* en *ADR2*. Reservado - Resultado si *MULT=1* en *ADR3*. Reservado - Resultado si *MULT=1* en *ADR4*.
3	*CD (Channel Select D)*	1100 1101 1110 1111	Pin V_{RH} - Resultado si *MULT=1* en *ADR1*. Pin V_{RL} - Resultado si *MULT=1*en *ADR2*. $(V_{RH})/2$ - Resultado si *MULT=1*en *ADR3*. Reservado - Resultado si *MULT=1* en *ADR4*.
4	*MULT (Multiple-Channel/Single Channel Control)*	0 1	Solo se realizan cuatro conversiones de un único canal seleccionado por los bits *CD* a *CA*. Se realiza una conversión de cada uno de los cuatro canales del banco seleccionado, donde cada uno de los registros de resultado corresponde a uno de los canales.
5	*SCAN (Continuous Scan Control)*	0 1	Solo se realizan cuatro conversiones de *ADR1* a *ADR4*. Se realizan conversiones indefinidamente guardando los valores de *ADR1* a *ADR4* a medida que va habiendo resultados validos.
6	-	-	-
7	*CCF (Conversions Complete Flag)*	0 1	Todavía no hay resultados validos de conversiones. Los cuatro registros de resultado de conversión contienen resultados de conversiones validos. Este bit se pone a cero cada vez que se escribe un nuevo comando en el registro *ADCTL*.

CONFIGURATION OPTIONS REGISTER (OPTION)			
BIT	**NOMBRE**	**VALOR**	**DESCRIPCIÓN**
6	*ADPU (A/D Powerup)*	0 1	Conversor A/D desactivado. Conversor A/D activado.
7	*CSEL (A/D/EE Charge Pump Clock Source Select)*	0 1	Se utiliza el clock del sistema. Se utiliza un clock R-C interno.

Interrupciones

En cuanto a las interrupciones sólo haremos unos comentarios en cuanto a las cosas importantes que hay que saber para entender un programa que las utilice. Para un estudio detallado de los pasos que se realizan en el *micro* cuando se produce una interrupción o del número y tipo de interrupciones consultar el manual del *micro* [1].

Cuando se produce una petición de interrupción, ya sea interna o externa, ésta no se sirve hasta que la instrucción actual ha terminado de ejecutarse. Después de la ejecución de la instrucción en curso se sirven las interrupciones que no estén enmascaradas de acuerdo con un circuito interno de prioridad. Esta prioridad se puede cambiar mediante los cuatro bits de menos peso del registro *HPRIO* (denominados *PSEL3* a *PSEL0*), que permiten elevar una única interrupción hasta el grado de prioridad más elevado.

Algunas interrupciones se pueden inhibir en conjunto mediante una instrucción, mientras que hay otras que no se pueden inhibir nunca. Además de esta manera de enmascarar todas las interrupciones, se pueden inhibir cada una por separado mediante determinados bits de control en algunos de los registros.

Cuando se sirve una interrupción, por defecto y si no se indica lo contrario, todas las interrupciones que se pueden enmascarar están inhibidas.

Al produce una interrupción, en la pila se guarda la dirección de retorno y el estado de los registros como se indica a continuación (el *SP* toma un nuevo valor que corresponde al anterior menos 9):

	PILA
SP	
SP+1	CONDITION CODE REGISTER
SP+2	ACUMULADOR A
SP+3	A7CUMULADOR B
SP+4	REGISTRO DE ÍNDICE X (X_H)
SP+5	REGISTRO DE ÍNDICE X (X_L)
SP+6	REGISTRO DE ÍNDICE Y (Y_H)
SP+7	REGISTRO DE ÍNDICE Y (Y_L)
SP+8	DIRECCIÓN DE RETORNO (RTN_H)
→ SP+9	DIRECCIÓN DE RETORNO (RTN_L)

→ Stack Pointer (SP) antes y después de la ejecución de la interrupción

Línea Serie

El microcomputador *MOTOROLA MC68HC11* incorpora un sistema de línea serie denominado *Serial Communications Interface (SCI)* que sigue el formato NRZ estándar (marca/espacio).
Para la transmisión y recepción se dispone de dos únicos *pins*: *TxD* y *RxD* respectivamente, de manera que transmisión y recepción son independientes aunque van a la misma velocidad, seleccionable de entre un total de 32 posibilidades.
Cada carácter se transmite como: un bit de *start*, 8 ó 9 bits de datos y un bit de *stop* (ver anexo sobre el estándar *RS-232*).

Se dispone de un único registro de datos compartido para recepción y transmisión denominado *SCDR*. La transmisión de datos comienza con una escritura en el registro *SCDR*, cuyo contenido se transfiere, cuando la transmisión está activada, a un segundo registro de transmisión de datos. Esta transferencia activa el bit *TDRE* del registro de estado de la *SCI* (*SCSR*) y en caso de que las interrupciones estén activadas generará una interrupción. Desde este registro el carácter se envía por línea serie empezando por el bit menos significativo. Cuando la transmisión se ha completado el bit *TC* del registro *SCSR* se activa para indicar dicha situación y se genera una interrupción en caso de que las interrupciones de este tipo estén permitidas. Si la transmisión está desactivada o se desactiva en mitad de la transmisión de un carácter, éste se acaba de enviar completamente y el bit también se activa generando la interrupción en caso de estar permitida.
Si la recepción está activada, cuando se lee el registro *SCDR* se obtiene el último carácter recibido. El bit *RDRF* del registro *SCSR* se activa cuando, después de una recepción completa, se transfiere un carácter del registro de recepción de datos al registro *SCDR* y puede producir una interrupción en caso de que las interrupciones de recepción estén permitidas. En caso de error en la recepción se activan los bits de registro *SCSR* correspondientes: *OR* (*overrun*), *NF* (ruido en la línea), *FE* (error de trama).
Disponemos de un total de 5 registros: *SCDR*, *SCCR1*, *SCCR2*, *SCSR* y *BAUD* cuyos bits se detallan a continuación.
El registro *Serial Communication Data Register (SCDR)* actúa como el registro de datos de transmisión (cuando se escribe) y como el registro de datos de recepción (cuando se lee).

SERIAL COMMUNICATIONS CONTROL REGISTER 1 (SCCR1)			
BIT	**NOMBRE**	**VALOR**	**DESCRIPCIÓN**
0	-	-	No implementado.
1	-	-	No implementado.
2	-	-	No implementado.
3	*WAKE (Wake Up Method Select)*	0 1	Despertar si la línea está en estado *Idle*. Despertar si la línea en estado *Addres Mark*.
4	*M (SCI Character Length)*	0 1	1 bit de start, 8 bits de datos, 1 bit de stop. 1 bit de start, 9 bits de datos, 1 bit de stop.
5	-	-	No implementado.
6	*T8 (Transmit Data Bit 8)*		Contiene el noveno bit del carácter a enviar si *M*=1.
7	*R8 (Receive Data Bit 8)*	0 1	Contiene el noveno bit del carácter recibido si *M*=1.

SERIAL COMMUNICATIONS CONTROL REGISTER 2 (SCCR2)			
BIT	**NOMBRE**	**VALOR**	**DESCRIPCIÓN**
0	*SBK*	0 1	No se envía *break* Enviar *break* (grupo de 10 (*M*=0) o 11 (*M*=1) ceros consecutivos).
1	*RWU*	0 1	Desactivar modo *wake up*. Activar modo *wake up*.
2	*RE*	0 1	Recepción desactivada. Recepción activada.
3	*TE*	0 1	Transmisión desactivada. Tranasmisión activada.
4	*ILIE (Idle Line Interrupt Enable)*	0 1	Interrupciones por *IDLE* desabilitadas. Interrupción de *SCI* si *IDLE*=1.
5	*RIE (Receive Interrupt Enable)*	0 1	Interrupciones por *RDRF* y *OR* desabilitadas. Interrupción de *SCI* si *RDRF*=1 o *OR*=1.
6	*TCIE (Transmit Complete Interrupt Enable)*	0 1	Interrupciones por *TC* desabilitadas. Interrupción de *SCI* si *TC*=1.
7	*TIE (Transmit Interrupt Enable)*	0 1	Interrupciones por *TDRE* desabilitadas. Interrupción de *SCI* si *TDRE*=1.

SERIAL COMMUNICATIONS STATUS REGISTER (SCSR)			
BIT	**NOMBRE**	**VALOR**	**DESCRIPCIÓN**
0	-	-	No implementado.
1	*FE (Framing Error)*	0 1	No hay error de trama. Error de trama (bit de stop no detectado en la recepción). Se pone a cero leyendo el registro *SCSR* seguido de una lectura del registro *SCDR*.
2	*NF (Noise Flag)*	0 1	No hay ruido en los bits del carácter recibido. Se detectó ruido en los bits del carácter recibido. Se pone a cero leyendo el registro *SCSR* seguido de una lectura del registro *SCDR*.
3	*OR (Overrun Error)*	0 1	No hay error de *overrun*. Hay error de *overrun* (ha llegado un carácter al *SCDR* antes de leer el anterior, se pierden los datos que causan el *overrun*). Se pone a cero leyendo el registro *SCSR* seguido de una lectura del registro *SCDR*.
4	*IDLE (Idle Line Detect)*	0 1	Estado *IDLE* no detectado. Línea en estado *IDLE*. Se pone a cero leyendo el registro *SCSR* seguido de una lectura del registro *SCDR*.
5	*RDRF (Receive Data Register Full)*	0 1	No hay datos recibidos en el registro *SCDR*. Hay datos recibidos en el registro *SCDR*. Se pone a cero leyendo el registro *SCSR* seguido de una lectura del registro *SCDR*.
6	*TC (Transmit Complete)*	0 1	Transmisión no completa. Transmisión completa. Se pone a cero leyendo el registro *SCSR* seguido de una escritura en el registro *SCDR*.
7	*TDRE (Transmit Data Register Empty)*	0 1	Registro *TDRE* contiene datos para enviar. Registro *TDRE* está vacio. Se pone a cero leyendo el registro *SCSR* seguido de una escritura en el registro *SCDR*.

BAUD RATE REGISTER (BAUD)			
BIT	**NOMBRE**	**VALOR**	**DESCRIPCIÓN**
0	*SCR0 (SCI Baud Rate Selects)*		Valor del segundo divisor aplicado al reloj del sistema para obtener la velocidad de funcionamiento de la línea serie. División por:
		000	1
1	*SCR1 (SCI Baud Rate Selects)*	001	2
		010	4
		011	8
2	*SCR2 (SCI Baud Rate Selects)*	100	16
		101	32
		110	64
		111	128
3	*RCKB (SCI Baud Rate Clock Check (Test))*	-	Solo utilizado para test de fábrica (en modo *test*).
4	*SCP0 (SCI Baud Rate Prescaler Selects)*		Valor del primer divisor aplicado al reloj del sistema para obtener la velocidad de funcionamiento de la línea serie. División por:
		00	1
5	*SCP1 (SCI Baud Rate Prescaler Selects)*	01	3
		10	4
		11	13
6	-	-	No implementado.
7	*TCLR (Clear Baud Rate Counters (Test))*	-	Solo utilizado para test de fábrica (en modo *test*)

ANEXO 3. MODOS DE DIRECCIONAMIENTO Y CONJUNTO DE INSTRUCCIONES

A continuación se describen los modos de direccionamiento y el conjunto de instrucciones para la familia *MOTOROLA MC68HC11*. En primer lugar realizaremos una descripción del *Condition Code Register (CCR)* por la relación de este registro con la ejecución de ciertas instrucciones.

Este registro nos informa de cuales han sido los resultados de la ejecución de una instrucción. Cada uno de los bits, también denominados *flags*, tiene un significado concreto dependiendo de si está activo (a uno) o no (a cero).

Condition Code Register

- Bit de *Carry/Borrow* (C): Se activa si se produjo un *carry* o *borrow* en la *ALU* [17] durante la última operación aritmética. También se ve afectado por instrucciones de desplazamiento o rotación de bits.
- Bit de *Overflow* (V): Se activa cuando se produce un *overflow* como resultado de una operación.
- Bit *Zero* (Z): Se activa si el resultado de la última operación aritmética, lógica o de manipulación de datos fue cero.
- Bit *Negative* (N): Se activa si el resultado de la última operación aritmética, lógica o de manipulación de datos fue negativo. Un resultado se considera negativo si su bit más significativo es un uno.
- Bit *Interrupt Mask* (I): Se activa por *hardware* o mediante instrucciones para deshabilitar todas las interrupciones mascarables. Si está a uno, todas las interrupciones mascarables quedan inhibidas, y si está a cero, las interrupciones quedan habilitadas.
- Bit *Half Carry* (H): Se activa cuando se produce un *carry* entre los bits 3 y 4 en la *ALU* durante la ejecución de las instrucciones *ADD*, *ABA* o *ADC*.
- Bit *X Interrupt Mask* (X): Sólo se activa por hardware (pins */RESET* o */XIRQ*) y solo se desactiva mediante las instrucciones *TAP* o *RTI*.
- Bit *Stop Disable* (S): Se activa para deshabilitar la instrucción de *STOP* y se pone a cero para permitir esta instrucción. Este bit es controlable por programa. La instrucción *STOP* se trata como "no operación" (*NOP*) si el bit S está activo.

La familia *MOTOROLA MC68HC11* dispone de un total de 6 modos de direccionamiento: *Inmediato*, *Directo*, *Extendido*, *Indexado* (con uno de los dos posibles registros de índice de 16 bits y un *offset* de 8 bits), *Inherente* y *Relativo*.

- Direccionamiento *Inmediato*: En este modo el valor del operando está contenido en el/los byte/s que siguen a continuación del código de la instrucción, donde el número de bytes coincide con el tamaño del registro.
- Direccionamiento *Directo*: Aquí el byte menos significativo de la dirección del operando está contenido en un byte a continuación del código de operación, y el byte de mayor peso de la dirección del operando se considera 00h. Esto permite acceder a las direcciones desde 0000h a 00FFh utilizando una instrucción de tan solo dos bytes, lo que reduce el tiempo de ejecución por eliminar accesos adicionales a memoria.

[17] *Arithmetic Logic Unit*, unidad artimético-lógica.

- Direccionamiento *Extendido*: En este modo los dos bytes a continuación del código de operación contienen la dirección absoluta del operando.

- Direccionamiento *Indexado*: Utiliza uno de los dos registros de índice (X o Y) para calcular la dirección absoluta del operando. En este caso la dirección es variable y depende de dos factores: a) El contenido actual del registro de índice utilizado, y b) el *offset* positivo de 8 bits contenido en la instrucción. Este modo permite direccionar a cualquier posición del espacio de 64 K disponible.

- Direccionamiento *Inherente*: Aquí toda la información está contenida en el código de instrucción (*opcode*). Los operandos (si hay) son registros y, por lo tanto, no es necesario referenciar a memoria.

- Direccionamiento *Relativo*: Se utiliza para instrucciones de salto. Si la condición de salto se cumple el contenido del byte (con signo) a continuación del código de instrucción se suma al contenido del registro *PC*[18] para obtener la dirección absoluta del salto. En caso contrario se continua con la siguiente instrucción.

La *CPU* [19] en la familia *MC68HC11* es, básicamente, una extensión de la *CPU MC6801* y, por tanto, permite la ejecución de la totalidad de las instrucciones *M6800* y *M6801*. Pero, además, cuenta con nuevas instrucciones gracias a la incorporación de un byte adicional (denominado "prebyte") en el código de operación de ciertas instrucciones.

A continuación se muestra una tabla con la totalidad de las instrucciones disponibles. Para cada una se indica, además del nombre, su expresión booleana, modos de direccionamiento posibles, número de ciclos que tarda en ejecutarse y *flags* del *Condition Code Register* que modifica.

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Cycles	Condition Codes S	X	H	I	N	Z	V	C
ABA	Add Accumulators	A + B → A	INH	2	-	-	↑↓	-	↑↓	↑↓	↑↓	↑↓
ABX	Add B to X	IX + 00:B → IX	INH	3	-	-	-	-	-	-	-	-
ABY	Add B to Y	IY + 00:B → IY	INH	4	-	-	-	-	-	-	-	-
ADCA (opr)	Add with Carry to A	A + M + C → A	A IMM A DIR A EXT A IND,X A IND,Y	2 3 4 4 5	-	-	↑↓	-	↑↓	↑↓	↑↓	↑↓
ADCB (opr)	Add with Carry to B	B + M + C → B	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	↑↓	-	↑↓	↑↓	↑↓	↑↓
ADDA (opr)	Add Memory to A	A + M → A	A IMM A DIR A EXT A IND,X A IND,Y	2 3 4 4 5	-	-	↑↓	-	↑↓	↑↓	↑↓	↑↓
ADDB (opr)	Add Memory to B	B + M → B	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	↑↓	-	↑↓	↑↓	↑↓	↑↓
ADDD (opr)	Add 16-Bit to D	D + M:M + 1 → D	IMM DIR EXT IND,X IND, Y	4 5 6 6 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
ANDA (opr)	AND A with Memory	A • M → A	A IMM	2	-	-	-	-	↑↓	↑↓	0	-

[18] *Program Counter*, registro con la dirección se la siguiente instrucción a ejecutar.
[19] *Central Process Unit*, unidad central de proceso.

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Cycles	Condition Codes S	X	H	I	N	Z	V	C
			A DIR A EXT A IND,X A IND,Y	3 4 4 5								
ANDB (opr)	AND B with Memory	B • M → B	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
ASL (opr)	Arithmetic Shift Left	C ← 8 bits ← 0	EXT IND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
ASLA			A INH	2								
ASLB			B INH	2								
ASLD	Arithmetic Shift Left Double	C ← 16 bits ← 0	INH	3	-	-	-	-	↑↓	↑↓	↑↓	↑↓
ASR (opr)	Arithmetic Shift Right	bit7 → 8 bits → C	EXT IND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
ASRA			A INH	2								
ASRB			B INH	2								
BCC (rel)	Branch if Carry Clear	? C = 0	REL	3	-	-	-	-	-	-	-	-
BCLR (opr) (msk)	Clear Bit(s)	M • (/mm) → M	DIR IND,X IND,Y	6 7 8	-	-	-	-	↑↓	↑↓	0	-
BCS (rel)	Branch if Carry Set	? C = 1	REL	3	-	-	-	-	-	-	-	-
BEQ (rel)	Branch if = 0	? Z = 1	REL	3	-	-	-	-	-	-	-	-
BGE (rel)	Branch if ≥ 0	? N ⊕ V = 0	REL	3	-	-	-	-	-	-	-	-
BGT (rel)	Branch if >0	? Z + (N ⊕ V) = 0	REL	3	-	-	-	-	-	-	-	-
BHI (rel)	Branch if Higher	? C + Z = 0	REL	3	-	-	-	-	-	-	-	-
BHS (rel)	Branch if Higher or Same	? C = 0	REL	3	-	-	-	-	-	-	-	-
BIT A (opr)	Bit(s) Test A with Memory	A • M	A IMM A DIR A EXT A IND,X A IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
BIT B (opr)	Bit(s) Test B with Memory	B • M	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
BLE (rel)	Branch if ≤ 0	? Z + (N ⊕ V) = 1	REL	3	-	-	-	-	-	-	-	-
BLO (rel)	Branch if Lower	? C = 1	REL	3	-	-	-	-	-	-	-	-
BLS (rel)	Branch if Lower or Same	? C + Z = 1	REL	3	-	-	-	-	-	-	-	-
BLT (rel)	Branch if < 0	? N ⊕ V = 1	REL	3	-	-	-	-	-	-	-	-
BMI (rel)	Branch if Minus	? N = 1	REL	3	-	-	-	-	-	-	-	-
BNE (rel)	Branch if Not = Zero	? Z = 0	REL	3	-	-	-	-	-	-	-	-
BPL (rel)	Branch if Plus	? N = 0	REL	3	-	-	-	-	-	-	-	-
BRA (rel)	Branch Always	? 1 = 1	REL	3	-	-	-	-	-	-	-	-
BRCLR (opr) (msk) (rel)	Branch if Bit(s) Clear	? M • mm = 0	DIR IND,X IND,Y	6 7 8	-	-	-	-	-	-	-	-
BRN (rel)	Branch Never	? 1 = 0	REL	3	-	-	-	-	-	-	-	-
BRSET (opr) (msk) (rel)	Branch if Bit(s) Set	? (/M) • mm = 0	DIR IND,X IND,Y	6 7 8	-	-	-	-	-	-	-	-
BSET (opr) (msk)		M + mm → M	DIR IND,X IND,Y	6 7 8	-	-	-	-	↑↓	↑↓	0	-
BSR (rel)	Branch to Subrutine		REL	6	-	-	-	-	-	-	-	-
BVC (rel)	Branch if Overflow Clear	? V = 0	REL	3	-	-	-	-	-	-	-	-

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Cycles	Condition Codes S	X	H	I	N	Z	V	C
BVS (rel)	Branch if Overflow Set	? V = 1	REL	3	-	-	-	-	-	-	-	-
CBA	Compare A to B	A - B	INH	2	-	-	-	-	↑↓	↑↓	↑↓	↑↓
CLC	Clear Carry Bit	0 → C	INH	2	-	-	-	-	-	-	-	0
CLI	Clear Interrupt Mask	0 → I	INH	2	-	-	-	0	-	-	-	-
CLR (opr)	Clear Memory Byte	0 → M	EXT IND,X IND,Y	6 6 7	-	-	-	-	0	1	0	0
CLRA	Clear Accumulator A	0 → A	A INH	2	-	-	-	-	0	1	0	0
CLRB	Clear Accumulator B	0 → B	B INH	2	-	-	-	-	0	1	0	0
CLV	Clear Overflow Flag	0 → V	INH	2	-	-	-	-	-	-	0	-
CMPA (opr)	Compare A to Memory	A - M	A IMM A DIR A EXT A IND,X A IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	↑↓	↑↓
CMPB (opr)	Compare B to Memory	B - M	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	↑↓	↑↓
COM (opr)	1's Complement Memory Byte	$FF - M → M	EXT IND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	0	1
COMA	1's Complement A	$FF - A → A	A INH	2	-	-	-	-	↑↓	↑↓	0	1
COMB	1's Complement B	$FF - B → B	B INH	2	-	-	-	-	↑↓	↑↓	0	1
CPD (opr)	Compare D to Memory 16-Bit	D - M:M + 1	IMM DIR EXT IND,X IND,Y	5 6 7 7 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
CPX (opr)	Compare X to Memory 16-Bit	IX - M:M + 1	IMM DIR EXT IND,X IND,Y	4 5 6 6 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
CPY (opr)	Compare Y to Memory 16-Bit	IY - M:M + 1	IMM DIR EXT IND,X IND,Y	5 6 7 7 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
DAA	Decimal Adjust A	Adjust Sum to BCD	INH	2	-	-	-	-	↑↓	↑↓	↑↓	↑↓
DEC (opr)	Decrement Memory Byte	M - 1 → M	EXT IND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	↑↓	-
DECA	Decrement Accumulator A	A - 1 → A	A INH	2	-	-	-	-	↑↓	↑↓	↑↓	-
DECB	Decrement Accumulator B	B - 1 → B	B INH	2	-	-	-	-	↑↓	↑↓	↑↓	-
DES	Decrement Stack Pointer	SP - 1 → SP	INH	3	-	-	-	-	-	-	-	-
DEX	Decrement Index Register X	IX - 1 → IX	INH	3	-	-	-	-	-	↑↓	-	-
DEY	Decrement Index Register Y	IY - 1 → IY	INH	4	-	-	-	-	-	↑↓	-	-
EORA (opr)	Exclusive OR A with Memory	A ⊕ M → A	A IMM A DIR A EXT A IND,X A IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
EORB (opr)	Exclusive OR B with Memory	B ⊕ M → B	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
FDIV	Fractional Divide 16 by 16	D / IX → IX; r → D	INH	41	-	-	-	-	-	↑↓	↑↓	↑↓

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Cycles	Condition Codes S	X	H	I	N	Z	V	C
IDIV	Integer Divide 16 by 16	D / IX → IX; r → D	INH	41	-	-	-	-	-	↑↓	0	↑↓
INC (opr)	Increment Memory Byte	M + 1 → M	EXT ND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	↑↓	-
INCA	Increment Accumulator A	A + 1 → A	A INH	2	-	-	-	-	↑↓	↑↓	↑↓	-
INCB	Increment Accumulator B	B + 1 → B	B INH	2	-	-	-	-	↑↓	↑↓	↑↓	-
INS	Increment Stack Pointer	SP + 1 → SP	INH	3	-	-	-	-	-	-	-	-
INX	Increment Index Register X	IX + 1 → IX	INH	3	-	-	-	-	-	↑↓	-	-
INY	Increment Index Register Y	IY + 1 → IY	INH	4	-	-	-	-	-	↑↓	-	-
JMP (opr)	Jump		EXT IND,X IND,Y	3 3 4	-	-	-	-	-	-	-	-
JSR (opr)	Jump to Subroutine		DIR EXT IND,X IND,Y	5 6 6 7	-	-	-	-	-	-	-	-
LDAA (opr)	Load Accumulator A	M → A	A IMM A DIR A EXT A IND,X A IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
LDAB (opr)	Load Accumulator B	M → B	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
LDD (opr)	Load Double Accumulator D	M → A, M + 1 → B	IMM DIR EXT IND,X IND,Y	3 4 5 5 6	-	-	-	-	↑↓	↑↓	0	-
LDS (opr)	Load Stack Pointer	M:M + 1 → SP	IMM DIR EXT IND,X IND,Y	3 4 4 5 6	-	-	-	-	↑↓	↑↓	0	-
LDX (opr)	Load Index Register X	M:M + 1 → IX	IMM DIR EXT IND,X IND,Y	3 4 5 5 6	-	-	-	-	↑↓	↑↓	0	-
LDY (opr)	Load Index Register Y	M:M + 1 → IY	IMM DIR EXT IND,X IND,Y	4 5 6 6 6	-	-	-	-	↑↓	↑↓	0	-
LSL (opr)	Logical Shift Left	C ← 8 bits ← 0	EXT IND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
LSLA			A INH	2								
LSLB			B INH	2								
LSLD	Logical Shift Left Double	C ← 16 bits ← 0	INH	3	-	-	-	-	↑↓	↑↓	↑↓	↑↓
LSR (opr)	Logical Shift Right	0 → 8 bits → C	EXT IND,X IND,Y	6 6 7	-	-	-	-	0	↑↓	↑↓	↑↓
LSRA			A INH	2								
LSRB			B INH	2								
LSRD	Logical Shift Right Double	0 → 16 bits → C	INH	3	-	-	-	-	0	↑↓	↑↓	↑↓
MUL	Multiply 8 by 8	A x B → D	INH	10	-	-	-	-	-	-	-	↑↓
NEG (opr)	2's Complement Memory Byte	0 - M → M	EXT	6	-	-	-	-	↑↓	↑↓	↑↓	↑↓

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Cycles	Condition Codes S	X	H	I	N	Z	V	C
			IND,X IND,Y	6 7								
NEGA	2's Complement A	0 - A → A	A INH	2	-	-	-	-	↑↓	↑↓	↑↓	↑↓
NEGB	2's Complement B	0 - B → B	B INH	2	-	-	-	-	↑↓	↑↓	↑↓	↑↓
NOP	No Operation	No operation	INH	2	-	-	-	-	-	-	-	-
ORAA (opr)	OR Accumulator A (Inclusive)	A + M → A	A IMM A DIR A EXT A IND,X A IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
ORAB (opr)	OR Accumulator B (Inclusive)	B + M → B	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
PSHA	Push A onto Stack	A → Stk, SP=SP-1	A INH	3	-	-	-	-	-	-	-	-
PSHB	Push B onto Stack	B → Stk, SP=SP-1	B INH	3	-	-	-	-	-	-	-	-
PSHX	Push X onto Stack (Lo First)	IX → Stk, SP=SP-2	INH	4	-	-	-	-	-	-	-	-
PSHY	Push Y onto Stack (Lo First)	IY → Stk, SP=SP-2	INH	5	-	-	-	-	-	-	-	-
PULA	Pull A from Stack	SP=SP+1, A ← Stk	A INH	4	-	-	-	-	-	-	-	-
PULB	Pull B from Stack	SP=SP+1, B ← Stk	B INH	4	-	-	-	-	-	-	-	-
PULX	Pull X from Stack (Hi First)	SP=SP+2, IX ← Stk	INH	5	-	-	-	-	-	-	-	-
PULY	Pull Y from Stack (Hi First)	SP=SP+2, IY ← Stk	INH	6	-	-	-	-	-	-	-	-
ROL (opr)	Rotate Left		EXT IND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
ROLA			A INH	2								
ROLB			B INH	2								
ROR (opr)	Rotate Right		EXT IND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
RORA			A INH	2								
RORB			B INH	2								
RTI	Return from Interrupt		INH	12	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓
RTS	Return from Subrutine		INH	5	-	-	-	-	-	-	-	-
SBA	Substract B from A	A - B → A	INH	2	-	-	-	-	↑↓	↑↓	↑↓	↑↓
SBCA (opr)	Substract with Carry from A	A - M - C → A	A IMM A DIR A EXT A IND,X A IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	↑↓	↑↓
SBCB (opr)	Substract with Carry from B	A - M - C → B	B IMM B DIR B EXT B IND,X B IND,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	↑↓	↑↓
SEC	Set Carry	1 → C	INH	2	-	-	-	-	-	-	-	1
SEI	Set Interrupt Mask	1 → I	INH	2	-	-	-	1	-	-	-	-
SEV	Set Overflow Flag	1 → V	INH	2	-	-	-	-	-	-	1	-
STAA (opr)	Store Accumulator A	A → M	A DIR A EXT A IND,X A IND,Y	3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
STAB (opr)	Store Accumulator B	B → M	B DIR B EXT B IND,X B IND,Y	3 4 4 5	-	-	-	-	↑↓	↑↓	0	-
STD (opr)	Store Accumulator D	A → M, B → M + 1	DIR EXT IND,X	4 5 5	-	-	-	-	↑↓	↑↓	0	-

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Cycles	Condition Codes S	X	H	I	N	Z	V	C
			INX,Y	6								
STOP	Stop Internal Clocks		INH	2	-	-	-	-	-	-	-	-
STS (opr)	Store Stack Pointer	SP → M:M + 1	DIR EXT IND,X INX,Y	4 5 5 6	-	-	-	-	↑↓	↑↓	0	-
STX (opr)	Store Index Register X	IX → M:M + 1	DIR EXT IND,X INX,Y	4 5 5 6	-	-	-	-	↑↓	↑↓	0	-
STY (opr)	Store Index Register Y	IY → M:M + 1	DIR EXT IND,X INX,Y	5 6 6 6	-	-	-	-	↑↓	↑↓	0	-
SUBA (opr)	Substract Memory from A	A - M → A	A IMM A DIR A EXT A IND,X A INX,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	↑↓	↑↓
SUBB (opr)	Substract Memory from B	B - M → B	B IMM B DIR B EXT B IND,X B INX,Y	2 3 4 4 5	-	-	-	-	↑↓	↑↓	↑↓	↑↓
SUBD (opr)	Substract Memory from D	D - M:M + 1 → D	IMM DIR EXT IND,X INX,Y	4 5 6 6 7	-	-	-	-	↑↓	↑↓	↑↓	↑↓
SWI	Software interrupt		INH	14	-	-	-	1	-	-	-	-
TAB	Transfer A to B	A → B	INH	2	-	-	-	-	↑↓	↑↓	0	-
TAP	Transfer A to CC Register	A → CCR	INH	2	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓
TBA	Transfer B to A	B → A	INH	2	-	-	-	-	↑↓	↑↓	0	-
TEST	TEST (Only in Test Modes)	Address Bus Counts	INH	-	-	-	-	-	-	-	-	-
TPA	Transfer CC Register to A	CCR → A	INH	2	-	-	-	-	-	-	-	-
TST (opr)	Test for Zero or Minus	M - 0	EXT IND,X IND,Y	6 6 7	-	-	-	-	↑↓	↑↓	0	0
TSTA		A - 0	A INH	2	-	-	-	-	↑↓	↑↓	0	0
TSTB		B - 0	B INH	2	-	-	-	-	↑↓	↑↓	0	0
TSX	Transfer Stack Pointer to X	SP + 1 → IX	INH	3	-	-	-	-	-	-	-	-
TSY	Transfer Stack Pointer to Y	SP + 1 → IY	INH	4	-	-	-	-	-	-	-	-
TXS	Transfer X to Stack Pointer	IX - 1 → SP	INH	3	-	-	-	-	-	-	-	-
TYS	Transfer Y to Stack Pointer	IY - 1 → SP	INH	4	-	-	-	-	-	-	-	-
WAI	Wait for Interrupt	Stack Regs & WAIT	INH	-	-	-	-	-	-	-	-	-
XGDX	Exchange D with X	IX → D; D → IX	INH	3	-	-	-	-	-	-	-	-
XGDY	Exchange D with Y	IY → D; D → IY	INH	4	-	-	-	-	-	-	-	-

ANEXO 4. ENSAMBLADOR DE LA FAMILIA *MC68HC11* EN EL ENTORNO *LISP*

Como se explica en el punto 3.3, los *micros* del robot se programan bajo un entorno de programación *LISP* para Macintosh. El lenguaje de programación es el ensamblador del *MOTOROLA MC68HC11* pero bajo *LISP*, que lo dota de ciertas características específicas para adecuarlo al compilador de *LISP*, que se describirán a continuación.

En primer lugar hay que saber que todas las instrucciones excepto las etiquetas deben ir entre paréntesis (característica de *LISP*).

Para indicar un número en hexadecimal, por ejemplo FFh, se ha de poner: *#xFF*. En el caso de un número decimal se pone tal cual. Un número binario, por ejemplo 01010101, se expresa como: *#b01010101*.

Para declarar una constante o una variable hay que indicar entre paréntesis, como siempre, primero *=v* en el caso de variables o *=c* en el caso de constantes. A continuación hay que indicar cuál será el nombre de la variable o constante. Finalmente, para las variables hay que indicar cuál será la dirección de memoria dónde estará y para las constantes cuál será su valor. Ejemplos:

(=v var #x30) → Variable llamada *var* en la posición 48 (decimal) de memoria.
(=v var2 #xB000) → Variable llamada *var2* en la posición B000h de memoria.
(=c cons #x05) → Constante llamada *cons* de valor 5 (decimal).

Los modos de direccionamiento los explicaremos con ejemplos basados en las variables y constantes del ejemplo anterior para una mejor comprensión.

- Modo *Inmediato*: Como operando se indica un valor precedido del signo "!". Por ejemplo:

(addd (! #x0010)) → Suma 16 al registro D.
(ldaa (! var)) → Carga en el registro A la dirección de *var*, un 48 (decimal).
(ldaa (! cons)) → Carga en el registro A el valor de *cons*, un 5 (decimal).

- Modo *Indexado*: Como operando se indica un offset (como mucho hasta 255 ya que solo puede ser un byte) precedido de "&" o "&x" para indexar con el registro X, o "&y" para indexar con el registro Y. Ejemplos:

(ldaa (&x 3)) → Guarda en el registro A el valor que haya en la dirección (X+3).
(ldaa (& 3)) → Guarda en el registro A el valor que haya en la dirección (X+3)
(ldaa (&y 4)) → Guarda en el registro A el valor que haya en la dirección (Y+3).
(ldaa (&y cons)) → Guarda en el registro A el valor que haya en la dirección (Y+*cons*)=(Y+5).

- Para el resto de modos de direccionamiento no hay que indicar nada especial, solo pasar el valor de una dirección que podrá ser: un valor absoluto, una variable (realmente su dirección) o una etiqueta. El compilador detecta según el tipo de operación o el valor de la dirección, el modo de direccionamiento. En concreto, para una instrucción de salto asumirá modo relativo, para una instrucción en que solo se utilice registros modo inherente, si el valor de la dirección que se le pasa (o la dirección donde esta declarada la variable que se le pasa) está entre 0 y 255 (en decimal) asumirá modo directo y si está por encima de 255 modo extendido. Ejemplos:

(bra etiqueta) → Modo relativo, hace un salto a *etiqueta.*
(ldaa var) → Modo directo, carga en el registro A el valor de *var* (contenido de la dirección 48).
(ldaa var2) → Modo extendido, carga en A el valor de *var2* (contenido de la dirección B000h).

Finalmente, comentar que se pueden utilizar funciones nativas de *LISP* (siempre que su valor se pueda determinar en tiempo de compilación), por ejemplo:

(+ var 5) → Suma la dirección de *var* y 5.
(+ 3 7 8) → Suma 3, 7 y 8, el resultado sería 18.
(- var 2) → Resta 2 a la dirección de *var* (primer parámetro).
(- 9 2 1 4) → Resta 2, 1 y 4 a 9 (primer parámetro), el resultado sería 2.

y para iniciar un programa hay que poner:

```
(defprog id_prog
        : machine m68hc11
        : start #xF800
        : code ( .... código del programa ...
                )
)
```

dónde *id_prog* es un identificador para el código que permitirá compilarlo más tarde, *mc68hc11* indica el tipo de *CPU* para la que se programa el código y *#xF800* indica la dirección de memoria de inicio del código.

ANEXO 5. CÓDIGO FUENTE DEL MICRO MAIN

A continuación se adjunta el código fuente de microcomputador MAIN que implementa el protocolo en la parte del robot. Para cada instrucción se indica el número de ciclos que tarda en ejecutarse. Para cada subrutina o interrupción al final de ésta se indica el número de ciclos de el total de instrucciones (no todas tienen porqué ejecutarse siempre) y el número de ciclos que tarda en ejecutarse en el caso peor.

```
(defprog genghis
   :machine m68hc11
   :start #xF800
   :code(

(comment "************************* DIRECCIONES DE REGISTROS *************************")
   (6811-register-addresses)


(comment "************************* DECLARACION DE CONSTANTES ************************")
   (=c STACK #xFF          (comment "Stack Pointer"))
   (=c TAM_BUF_LS #x40     (comment "Longitud de los buffers de linea serie, 64 bytes"))
   (=c MOD_TAM_BUF_LS #x3F(comment "Mascara para hacer modulo longitud buffer linea serie"))
   (=c TAM_BUF_SPI #x15    (comment "Longitud de los buffers de spi, 21 bytes"))
   (=c 2TAM_BUF_SPI (+ TAM_BUF_SPI TAM_BUF_SPI) (comment "Doble de la longitud buffer spi"))
   (=c TAM_TRAMA_IN #x11   (comment "Tama-o buffer trama en recepcion, 17 bytes"))
   (=c SOH #xFF            (comment "Cabecera para las tramas"))
   (=c MOTOR_VOLT_MIN 80   (comment "Voltaje minimo de la bateria de motores"))

(comment "************************* DECLARACION DE VARIABLES *************************")
   (=v ValScsr #x00        (comment "Variable para consultar SCSR en rutina int_ls"))
   (=v ValSccr2 #x01       (comment "Variable para guardar valor del SCCR2 en rutina int_ls"))
   (=v CarEnv #x02         (comment "Indica si caracter recibido al llamar a RecibeCar"))
   (=v CarReb #x03         (comment "Indica si caracter enviado al llamar a EnviaCar"))
   (=v TamInfoSpi #x04     (comment "Indica el tama-o de los datos a recibir por SPI"))
   (=v QueDatos1 #x05      (comment "Primer byte que indica que datos enviar al PC"))
   (=v QueDatos2 #x06      (comment "Segundo byte que indica que datos enviar al PC"))
   (=v NuevoCiclo #x07     (comment "Indica nuevo ciclo envio/recepcion de datos"))
   (=v TipoCiclo #x08      (comment "Indica el tipo de ciclo actual de los dos posibles"))
   (=v Checksum #x09       (comment "Variable temporal para calculo del checksum"))
   (=v NuevasOrdenes #x0A (comment "Indica a spi si dispone de nuevas ordenes recibidas"))
   (=v GenTramaTmp #x0B    (comment "Variable temporal para generar la trama"))
   (=v TipoTrama #x0C      (comment "Guarda el tipo de trama en recepcion"))
   (=v LonTramaRecep #x0D (comment "Longitud que queda por recibir de la trama en recepcion"))

   (=v InBuffer #xB000     (comment "Buffer de entrada de la linea serie de TAMBUF bytes"))
   (=v InBuffer_In (+ InBuffer TAM_BUF_LS)
                           (comment "Offset del siguiente byte libre en InBuffer"))
   (=v InBuffer_Out (+ InBuffer TAM_BUF_LS 1)
                           (comment "Offset del siguiente byte disponible en InBuffer"))
   (=v InBuffer_Count (+ InBuffer TAM_BUF_LS 2)
                           (comment "Contador de bytes ocupados en InBuffer"))
   (=v OutBuffer (+ InBuffer TAM_BUF_LS 3)
                           (comment "Buffer de salida de la linea serie de TAM_BUF_LS bytes"))
   (=v OutBuffer_In (+ OutBuffer TAM_BUF_LS)
                           (comment "Offset del siguiente byte libre en OutBuffer"))
   (=v OutBuffer_Out (+ OutBuffer TAM_BUF_LS 1)
                           (comment "Siguiente byte disponible en OutBuffer"))
   (=v OutBuffer_Count (+ OutBuffer TAM_BUF_LS 2)
                           (comment "Contador de bytes ocupados en OutBuffer"))
   (=v InSpi (+ OutBuffer TAM_BUF_LS 3)
                           (comment "Buffer de entrada de la spi, 21 bytes: datos del SERVO"))
   (=v OutSpi (+ InSpi TAM_BUF_SPI)
                           (comment "Buffer de salida de la spi, 21 bytes: datos al SERVO"))
   (=v OutSpiTmp (+ OutSpi TAM_BUF_SPI)
                           (comment "Buffer de salida de la spi con nuevas ordenes, 21 bytes"))
   (=v TramaIn (+ OutSpiTmp TAM_BUF_SPI)
                           (comment "Buffer que contiene la trama en recepcion: 17 bytes"))


(comment "***************************** INICIALIZACIONES *****************************")
inicializar
   (comment "Inicializacion sp,registro base y registro D")
   (lds (! STACK)          (comment "Inicializacion del stack pointer"))
   (ldx (! XBASE)          (comment "Inicializacion del registro indice X"))
   (ldaa (! #x06))
   (staa (& DDRD)          (comment "Direccion pins reg D, out bits: MISO (spi slave) y TxD"))
```

```
   (comment "Inicializacion de spi")
   (ldaa (! #xCC))
   (staa (& SPCR)          (comment "Spi activa, interrupcion de spi activada"))
   (ldaa (! #x03))
   (staa (& HPRIO)         (comment "Spi interrupcion de maxima prioridad"))

   (comment "Inicializacion linea serie")
   (ldaa (! #x00))
   (staa (& SCCR1)         (comment "8 bits de datos"))
   (ldaa (! #x2C))
   (staa (& SCCR2)         (comment "Habilitamos transmision, recepcion e int. recepcion"))
   (ldaa (! #x12))
   (staa (& BAUD)          (comment "Baud rate a 10417"))

   (comment "Inicializacion del conversor A/D")
   (ldaa (! #x80))
   (staa (& OPTION)        (comment "Conversor A/D activo a frecuencia de E clock"))
   (ldaa (! #x30))
   (staa (& ADCTL)         (comment "Multiple chanel, continuous scan,chanel select=0000"))

   (comment "Inicializacion de variables")
   (ldaa (! #x00))
   (staa InBuffer_In       (comment "InBuffer_In=0"))
   (staa InBuffer_Out      (comment "InBuffer_Out=0"))
   (staa InBuffer_Count    (comment "InBuffer_Count=0"))
   (staa OutBuffer_In      (comment "OutBuffer_In=0"))
   (staa OutBuffer_Out     (comment "OutBuffer_Out=0"))
   (staa OutBuffer_Count   (comment "OutBuffer_Count=0"))
   (staa NuevoCiclo        (comment "NuevoCiclo=0"))
   (staa TipoCiclo         (comment "TipoCiclo=0: bateria motor y piros 1,3,5"))
   (staa (+ OutSpi 18)     (comment "OutSpi[18]=0: leds apagados"))
   (staa (+ OutSpi 19)     (comment "OutSpi[19]=0: buzzer parado"))
   (staa NuevasOrdenes     (comment "NuevasOrdenes=0, no hay nuevas ordenes recibidas"))
   (ldaa (! #x5E)          (comment "Dos bits mas peso OutSpi[0] y OutSpiTmp[0] a '01'"))
   (staa OutSpi            (comment "indican al servo motores on, nivel de baterias"))
   (staa OutSpiTmp         (comment "minimo y resto de bits valor motor alfa5 a 30"))
   (ldaa (! #x1E)          (comment "Valor inicial ordenes a motores: 30, pos. intermedia"))
   (staa (+ OutSpi 10)     (comment "Orden motor alfa0"))
   (staa (+ OutSpi 8)      (comment "Orden motor alfa1"))
   (staa (+ OutSpi 14)     (comment "Orden motor alfa2"))
   (staa (+ OutSpi 12)     (comment "Orden motor alfa3"))
   (staa (+ OutSpi 6)      (comment "Orden motor alfa4"))
   (staa (+ OutSpi 11)     (comment "Orden motor beta0"))
   (staa (+ OutSpi 9)      (comment "Orden motor beta1"))
   (staa (+ OutSpi 15)     (comment "Orden motor beta2"))
   (staa (+ OutSpi 13)     (comment "Orden motor beta3"))
   (staa (+ OutSpi 5)      (comment "Orden motor beta4"))
   (staa (+ OutSpi 7)      (comment "Orden motor beta5"))
   (ldaa (! #x0F))
   (staa QueDatos1         (comment "QueDatos1=#x0F, se envian datos de todos los sensores"))
   (ldaa (! #xF8))
   (staa QueDatos2         (comment "QueDatos2=#xF8, se envian datos de todos los sensores"))

   (cli                    (comment "Habilitamos las interrupciones"))


(comment "***************************** PROGRAMA PRINCIPAL **************************")
bucle
   (ldaa NuevoCiclo        (comment "Consultamos si inicio nuevo ciclo: fin de spi"))    ;[3]
   (beq bucle              (comment "Si no, esperamos"))                                 ;[3]
   (clr NuevoCiclo         (comment "NuevoCiclo a cero para detectar llegada SPI"))      ;[6]
   (jsr GeneraTrama        (comment "Generamos nueva trama a enviar por linea serie"));[6+2912]

   (brclr NuevasOrdenes #xFF no_ord_nuevas
                           (comment "Si ha habido ordenes nuevas se guardan"))           ;[6]
   (clr NuevasOrdenes      (comment "Reseteamos el indicador de si hay nuevas ordenes")) ;[6]
   (ldaa OutSpiTmp         (comment "Obtenemos valor de motor alfa5 y nivel baterias"))  ;[4]
   (staa OutSpi            (comment "Guardamos el valor en OutSpi[0]"))                  ;[4]
   (ldaa (+ OutSpiTmp 5)   (comment "Obtenemos valor de motor beta4"))                   ;[4]
   (staa (+ OutSpi 5)      (comment "Guardamos el valor en OutSpi[5]"))                  ;[4]
   (ldaa (+ OutSpiTmp 6)   (comment "Obtenemos valor de motor alfa4"))                   ;[4]
   (staa (+ OutSpi 6)      (comment "Guardamos el valor en OutSpi[6]"))                  ;[4]
   (ldaa (+ OutSpiTmp 7)   (comment "Obtenemos valor de motor beta5"))                   ;[4]
   (staa (+ OutSpi 7)      (comment "Guardamos el valor en OutSpi[7]"))                  ;[4]
   (ldaa (+ OutSpiTmp 8)   (comment "Obtenemos valor de motor alfa1"))                   ;[4]
   (staa (+ OutSpi 8)      (comment "Guardamos el valor en OutSpi[8]"))                  ;[4]
   (ldaa (+ OutSpiTmp 9)   (comment "Obtenemos valor de motor beta1"))                   ;[4]
   (staa (+ OutSpi 9)      (comment "Guardamos el valor en OutSpi[9]"))                  ;[4]
   (ldaa (+ OutSpiTmp 10)  (comment "Obtenemos valor de motor alfa0"))                   ;[4]
   (staa (+ OutSpi 10)     (comment "Guardamos el valor en OutSpi[10]"))                 ;[4]
   (ldaa (+ OutSpiTmp 11)  (comment "Obtenemos valor de motor beta0"))                   ;[4]
   (staa (+ OutSpi 11)     (comment "Guardamos el valor en OutSpi[11]"))                 ;[4]
```

```
   (ldaa (+ OutSpiTmp 12) (comment "Obtenemos valor de motor alfa3"))                  ;[4]
   (staa (+ OutSpi 12)    (comment "Guardamos el valor en OutSpi[12]"))                ;[4]
   (ldaa (+ OutSpiTmp 13) (comment "Obtenemos valor de motor beta3"))                  ;[4]
   (staa (+ OutSpi 13)    (comment "Guardamos el valor en OutSpi[13]"))                ;[4]
   (ldaa (+ OutSpiTmp 14) (comment "Obtenemos valor de motor alfa2"))                  ;[4]
   (staa (+ OutSpi 14)    (comment "Guardamos el valor en OutSpi[14]"))                ;[4]
   (ldaa (+ OutSpiTmp 15) (comment "Obtenemos valor de motor beta2"))                  ;[4]
   (staa (+ OutSpi 15)    (comment "Guardamos el valor en OutSpi[15]"))                ;[4]
   (ldaa (+ OutSpiTmp 18) (comment "Obtenemos valor de los leds"))                     ;[4]
   (staa (+ OutSpi 18)    (comment "Guardamos el valor en OutSpi[18]"))                ;[4]
   (ldaa (+ OutSpiTmp 19) (comment "Obtenemos valor del buzzer"))                      ;[4]
   (staa (+ OutSpi 19)    (comment "Guardamos el valor en OutSpi[19]"))                ;[4]

no_ord_nuevas
   (ldab TipoCiclo        (comment "Consultamos TipoCiclo para refresco de nivel de")) ;[3]
   (beq no_level_bat      (comment "bateria motores, solo en ciclos de tipo 1"))       ;[3]
   (ldaa (& ADR4)         (comment "Actualizamos bits nivel baterias para el servo"))  ;[4]
   (cmpa (! MOTOR_VOLT_MIN) (comment "Comparamos si el motor recibe el voltaje minimo"));[2]
   (bgt motor_on)                                                                      ;[3]
   (ldaa OutSpi           (comment "Obtenemos el valor de OutSpi[0]"))                 ;[4]
   (anda (! #xBF)         (comment "No voltaje minimo, motores off. Lo indicamos al")) ;[2]
   (oraa (! #x80)         (comment "servo con '10' en dos bits mayor peso de OutSpi")) ;[2]
   (staa OutSpi           (comment "Restauramos nuevo valor de OutSpi[0]"))            ;[4]
   (ldaa OutSpiTmp        (comment "Motores off. Lo indicamos al servo con '10' en"))  ;[2]
   (anda (! #xBF)         (comment "No voltaje minimo, motores off. Lo indicamos al")) ;[2]
   (oraa (! #x80)         (comment "servo con '10' en dos bits mayor peso de OutSpi")) ;[2]
   (staa OutSpiTmp        (comment "dos bits mayor peso OutSpiTmp"))                   ;[4]
   (bra no_level_bat      (comment "Saltamos a fin actualizacion nivel baterias"))     ;[3]
motor_on
   (ldaa OutSpi           (comment "Obtenemos el valor de OutSpi[0]"))                 ;[4]
   (oraa (! #x40)         (comment "Hay voltaje minimo, motores on. Lo indicamos al")) ;[2]
   (anda (! #x7F)         (comment "servo con '01' en dos bits mayor peso de OutSpi")) ;[2]
   (staa OutSpi           (comment "Restauramos nuevo valor de OutSpi[0]"))            ;[4]
   (ldaa OutSpiTmp        (comment "Motores on. Lo indicamos al servo con '01' en"))   ;[2]
   (oraa (! #x40)         (comment "Hay voltaje minimo, motores on. Lo indicamos al")) ;[2]
   (anda (! #x7F)         (comment "servo con '01' en dos bits mayor peso de OutSpi")) ;[2]
   (staa OutSpiTmp        (comment "dos bits mayor peso OutSpiTmp y el resto a cero")) ;[4]
no_level_bat

   (jsr RecibeTrama       (comment "Rutina que se encarga de recibir una trama"))

   (bra bucle             (comment "Bucle infinito"))                                 ;[3]


(comment "***************************** RUTINA EnviaCar *****************************")
;Rutina que envia un caracter del acumulador A por linea serie. Si hay espacio en el buffer de
;linea serie se escribe el caracter del acumulador A que luego se enviara por la linea y se
;indica en la variable CarEnv con un uno, en caso de no haber espacio disponible CarEnv se
;pone a cero y el caracter no se envia.

EnviaCar
   (psha                  (comment "Salvamos el valor del registro A en la pila"))    ;[3]
   (pshb                  (comment "Salvamos el valor del registro B en la pila"))    ;[3]
   (pshy                  (comment "Salvamos el valor del registro Y en la pila"))    ;[5]

   (ldab OutBuffer_Count  (comment "Consultamos numero caracteres buffer salida"))    ;[4]
   (cmpb (! TAM_BUF_LS)   (comment "Consultamos si el buffer esta lleno"))            ;[2]
   (bne guarda_car)                                                                   ;[3]
   (ldab (! #x00)         (comment "Si el buffer esta lleno NO enviamos el caracter")) ;[2]
   (stab CarEnv           (comment "y lo indicamos en variable CarEnv con un cero"))  ;[3]
   (bra ver_cad_int       (comment "Comprobamos si existe cadena de interrupciones")) ;[3]
guarda_car
   (ldab (! #x01)         (comment "Si el buffer no esta lleno enviamos el caracter")) ;[2]
   (stab CarEnv           (comment "y lo indicamos en la variable CarEnv con un uno")) ;[3]
   (ldy (! OutBuffer)     (comment "Guardamos en Y el puntero al inicio de OutBuffer")) ;[4]
   (ldab OutBuffer_In     (comment "Guardamos en B offset a siguiente posicion libre")) ;[4]
   (aby                   (comment "Obtenemos el puntero a siguiente posicion libre")) ;[4]
   (staa (&y 0)           (comment "Guardamos el caracter en A en la posicion libre")) ;[5]
   (incb                  (comment "Incrementamos offset a siguiente posicion libre")) ;[2]
   (andb (! MOD_TAM_BUF_LS)       (comment "Offset=Offset MOD TAM_BUF_LS"))           ;[2]
   (stab OutBuffer_In     (comment "Guardamos el nuevo offset en OutBuffer_In"))      ;[4]
   (inc OutBuffer_Count   (comment "Incrementamos el numero de bytes de OutBuffer"))  ;[6]
ver_cad_int
   (brset (& SCCR2) #x80 fin_EnviaCar (comment "Si hay cadena de interrupc., acabamos"));[7]
   (brclr (& SCSR) #x80 fin_EnviaCar  (comment "Si no SCDR vacio, acabamos"))         ;[7]
   (ldy (! OutBuffer)     (comment "Si no, guardamos en Y puntero a inicio OutBuffer")) ;[4]
   (ldab OutBuffer_Out    (comment "Guardamos en B offset primera posic. disponible")) ;[4]
   (aby                   (comment "Obtenemos puntero a primera posicion disponible")) ;[4]
   (ldaa (&y 0)           (comment "Guardamos en A siguiente caracter a enviar"))     ;[5]
   (incb                  (comment "Increment. offset a primera posicion disponible")) ;[2]
   (andb (! MOD_TAM_BUF_LS)   (comment "Hacemos el modulo al offset"))                ;[2]
   (stab OutBuffer_Out    (comment "Guardamos el nuevo offset en OutBuffer_Out"))     ;[4]
```

```
   (dec OutBuffer_Count   (comment "Decrementamos el numero de bytes en Outbuffer"))    ;[6]
   (staa (& SCDR)         (comment "Enviamos caracter por linea serie, registro SCDR")) ;[4]
   (bset (& SCCR2) #x80   (comment "Permitimos interrupciones transmision caracter"))   ;[7]

fin_EnviaCar
   (puly                  (comment "Restauramos el valor del registro Y de la pila"))   ;[6]
   (pulb                  (comment "Restauramos el valor del registro B de la pila"))   ;[4]
   (pula                  (comment "Restauramos el valor del registro A de la pila"))   ;[4]
   (rts                   (comment "Fin de la subrutina y retorno"))                    ;[5]
;Total ciclos de la subrutina EnviaCar [139], peor caso maximo [131].


(comment "****************************** RUTINA RecibeCar ****************************")
;Rutina que recibe un caracter de la linea serie en el acumulador A. En caso de haber algun
;caracter disponible en el buffer de linea serie se lee el caracter en el acumulador A y se
;indica en la variable CarReb con un uno, en caso de no disponer de ningun caracter se indica
;con un cero en CarReb y el acumulador A se deja igual.

RecibeCar
   (pshb                  (comment "Salvamos el valor del registro B en la pila"))      ;[3]
   (pshy                  (comment "Salvamos el valor del registro Y en la pila"))      ;[5]

   (ldab InBuffer_Count   (comment "Consultamos numero caracteres del buffer entrada")) ;[4]
   (bne recibe_car        (comment "Si buffer vacio, no hay caracter que recibir"))     ;[3]
   (stab CarReb           (comment "lo indicamos en la variable CarReb con un cero"))   ;[3]
   (puly                  (comment "Restauramos el valor del registro Y de la pila"))   ;[6]
   (pulb                  (comment "Restauramos el valor del registro B de la pila"))   ;[4]
   (rts                   (comment "y acabamos la subrutina"))                          ;[5]
recibe_car
   (ldab (! #x01)         (comment "Si el buffer no esta vacio"))                       ;[2]
   (stab CarReb           (comment "lo indicamos en la variable CarReb con un uno"))    ;[3]
   (ldy (! InBuffer)      (comment "Guardamos en Y el puntero al inicio de InBuffer"))  ;[4]
   (ldab InBuffer_Out     (comment "Guardamos en B offset primera posic. disponible"))  ;[4]
   (aby                   (comment "Obtenemos puntero a primera posicion disponible"))  ;[4]
   (ldaa (&y 0)           (comment "Guardamos en A caracter disponible en InBuffer"))   ;[5]
   (incb                    (comment "Increment. offset a primera posicion disponible"));[2]
   (andb (! MOD_TAM_BUF_LS) (comment "Offset=Offset MOD TAM_BUF_LS"))                   ;[2]
   (stab InBuffer_Out     (comment "Guardamos el nuevo offset en InBuffer_Out"))        ;[4]
   (dec InBuffer_Count    (comment "Decrementamos el numero de bytes en InBuffer"))     ;[6]

   (puly                  (comment "Restauramos el valor del registro Y de la pila"))   ;[6]
   (pulb                  (comment "Restauramos el valor del registro B de la pila"))   ;[4]
   (rts                   (comment "Acabamos la subrutina"))                            ;[5]
;Total ciclos de la subrutina RecibeCar [84], peor caso maximo [66]


(comment "***************************** RUTINA GeneraTrama ***************************")
;Rutina que genera una trama caracter a caracter a partir de los datos que hay en el buffer de
;entrada de la spi. Va enviando los caracteres de la trama por linea serie a medida que se
;construyen y a la vez va calculando el checksum. La cabecera de la trama ya se habra enviado
;al final de la spi anterior que ha marcado el inicio del nuevo ciclo. El tipo de trama
;generada depende del valor de las variables QueDatos1 y QueDatos2 que indican que datos de
;que sensores se quieren recibir en el PC y del tipo de ciclo en el que se esta.

GeneraTrama
   (psha                  (comment "Salvamos el valor del registro A en la pila"))      ;[3]
   (pshb                  (comment "Salvamos el valor del registro B en la pila"))      ;[3]

   (clr Checksum          (comment "Ponemos a cero variable para calculo checksum"))    ;[6]
   (ldaa QueDatos1        (comment "Obtenemos primer byte datos que llevara la trama")) ;[3]
   (ldab TipoCiclo        (comment "Consultamos el tipo de ciclo: valor 0 o 1"))        ;[3]
   (beq trama0            (comment "Si TipoCiclo=0 entonces tipo trama tambien 0"))     ;[3]
   (oraa (! #x10)         (comment "si no indicamos que es una trama de tipo 1"))       ;[2]
;[3]
trama0
   (jsr EnviaCar          (comment "Enviamos el primer byte de datos"))               ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))               ;[3]

   (clra                  (comment "Ponemos a cero el acumulador A"))                   ;[2]
   (brclr QueDatos2 #x08 fin_wiskers    (comment "Miramos si se desean los wiskers"))   ;[6]
   (ldaa (& PORTA)        (comment "Lectura valor de wiskers, dos bits menor peso"))    ;[4]
   (anda (! #x03)         (comment "Ponemos a cero el resto de bits de mas peso"))      ;[2]
fin_wiskers
   (oraa QueDatos2        (comment "A-adimos bits de segundo byte de datos"))           ;[3]
   (jsr EnviaCar          (comment "Enviamos el segundo byte de datos"))              ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))               ;[3]

   (brclr QueDatos1 #x08 fin_mot_alfa (comment "Miramos si se desean los motores alfa"));[6]
   (ldaa (+ InSpi 10)     (comment "Guardamos en B el valor del motor alfa0"))          ;[4]
   (asla                  (comment "Desplazamos el acumulador B 4 bits izquierda"))     ;[2]
```

```
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (oraa (+ InSpi 8)      (comment "Guardamos valor motor alfa1 en 4 bits menos peso")) ;[4]
   (jsr EnviaCar          (comment "Enviamos el primer byte de motores"))            ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))       ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))            ;[3]

   (ldaa (+ InSpi 0)      (comment "Guardamos en B el valor del motor alfa2"))       ;[4]
   (asla                  (comment "Desplazamos el acumulador B 4 bits izquierda"))  ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (oraa (+ InSpi 7)      (comment "Guardamos valor motor alfa3 en 4 bits menos peso")) ;[4]
   (jsr EnviaCar          (comment "Enviamos el segundo byte de motores"))           ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))       ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))            ;[3]

   (ldaa (+ InSpi 1)      (comment "Guardamos en B el valor del motor alfa4"))       ;[4]
   (asla                  (comment "Desplazamos el acumulador B 4 bits izquierda"))  ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (oraa (+ InSpi 5)      (comment "Guardamos valor motor alfa5 en 4 bits menos peso")) ;[4]
   (jsr EnviaCar          (comment "Enviamos el tercer byte de motores"))            ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))       ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))            ;[3]
fin_mot_alfa

   (brclr QueDatos1 #x04 fin_mot_beta (comment "Miramos si se desean los motores beta"));[6]
   (ldaa (+ InSpi 11)     (comment "Guardamos en B el valor del motor beta0"))       ;[4]
   (asla                  (comment "Desplazamos el acumulador A 4 bits izquierda"))  ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (oraa (+ InSpi 9)      (comment "Guardamos valor motor beta1 en 4 bits menos peso")) ;[4]
   (jsr EnviaCar          (comment "Enviamos el cuarto byte de motores"))            ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))       ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))            ;[3]
   (ldaa (+ InSpi 3)      (comment "Guardamos en B el valor del motor beta2"))       ;[4]
   (asla                  (comment "Desplazamos el acumulador B 4 bits izquierda"))  ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (oraa (+ InSpi 4)      (comment "Guardamos valor motor beta3 en 4 bits menos peso")) ;[4]
   (jsr EnviaCar          (comment "Enviamos el quinto byte de motores"))            ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))       ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))            ;[3]

   (ldaa (+ InSpi 2)      (comment "Guardamos en A el valor del motor beta4"))       ;[4]
   (asla                  (comment "Desplazamos el acumulador A 4 bits izquierda"))  ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (asla)                                                                    ;[2]
   (oraa (+ InSpi 6)      (comment "Guardamos valor motor beta5 en 4 bits menos peso")) ;[4]
   (jsr EnviaCar          (comment "Enviamos el sexto byte de motores"))             ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))       ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))            ;[3]
fin_mot_beta

   (clra                  (comment "Indicamos que no se guardan posiciones tacto"))  ;[2]
   (clrb                  (comment "Acumulador B a cero para guardar bits tacto"))   ;[2]

   (brclr QueDatos1 #x02 fin_pos_tact (comment "Miramos si se desean posiciones tacto"));[6]
   (ldaa (+ InSpi 18)     (comment "Obtenemos en A primer byte posiciones de tacto")) ;[4]
   (asla                  (comment "Desplazamos A 1 a izquierda, msb en flag C"))    ;[2]
   (rorb                  (comment "Bit mayor peso lo pasamos temporalmente a B"))   ;[2]
   (jsr EnviaCar          (comment "Enviamos el primer byte de posiciones de tacto")) ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))       ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))            ;[3]

   (ldaa (+ InSpi 19)     (comment "Obtenemos en A segundo byte posiciones de tacto"))  ;[4]
   (asla                  (comment "Desplazamos A 1 a izquierda, msb en flag C"))    ;[2]
   (rorb                  (comment "Bit mayor peso lo pasamos temporalmente a B"))   ;[2]
   (jsr EnviaCar          (comment "Enviamos segundo byte de posiciones de tacto")) ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))       ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))            ;[3]

   (ldaa (! #x01)         (comment "Indicamos que se han guardado posiciones tacto")) ;[2]
fin_pos_tact

   (brclr QueDatos1 #x01 no_frz_tact (comment "Miramos si se desean fuerzas de tacto")) ;[6]
```

```
   (cmpa (! #x01)         (comment "Miramos si ya insertado el valor fuerza tacto1"))   ;[2]
   (beq ya_frz_tacto1     (comment "Si ya insertado, A tendra valor 1 cargado antes"))  ;[3]
   (ldaa (+ InSpi 19)     (comment "Obtenemos en A segundo byte posiciones de tacto"))  ;[4]
   (asla                  (comment "Desplazamos A 1 a izquierda, msb en flag C"))       ;[2]
   (rorb                  (comment "Bit mayor peso lo pasamos temporalmente a B"))      ;[2]
   (jsr EnviaCar          (comment "Enviamos el primer byte de fuerzas de tacto"))    ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))               ;[3]
ya_frz_tacto1

   (ldaa (+ InSpi 20)     (comment "Obtenemos en A byte con fuerzas de tacto"))         ;[4]
   (asla                  (comment "Desplazamos A 1 a la izquierda, msb en flag C"))    ;[2]
   (rorb                  (comment "Bit mayor peso lo pasamos temporalmente a B"))      ;[2]
   (jsr EnviaCar          (comment "Enviamos el segundo byte fuerzas de tacto"))      ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))               ;[3]

   (ldaa (! #x01)         (comment "Indicamos que se han guardado fuerzas de tacto"))   ;[2]
   (bra fin_frz_tact      (comment "Saltamos al final de las fuerzas de tacto"))        ;[3]
no_frz_tact
   (lsrb)                 (comment "Para que los bits de tacto esten en su lugar")      ;[2]
fin_frz_tact

   (stab GenTramaTmp      (comment "Bits de tacto en GenTramaTmp, el resto a cero"))    ;[3]


   (brclr QueDatos2 #x80 fin_infrarr  (comment "Miramos si se desean los infrarrojos")) ;[6]
   (ldaa (+ InSpi 16)     (comment "Obtenemos en B byte con valor infrarrojos y modo")) ;[4]
   (anda (! #x0F)         (comment "Eliminamos los cuatro bits de mayor peso"))         ;[2]
   (oraa GenTramaTmp      (comment "Juntamos datos infrarrojos a bits tacto"))          ;[3]
   (staa GenTramaTmp      (comment "Datos infrarrojos y tacto en GenTramaTmp"))         ;[3]
   (ldaa (! #x01)         (comment "Indicamos que se han guardado infrarrojos"))        ;[2]
fin_infrarr

   (cmpa (! #x00)         (comment "Si no p. tact., f. tact. ni infrarr. A vale cero")) ;[2]
   (beq no_byte_infrarr)                                                               ;[3]
   (ldaa GenTramaTmp      (comment "Obtenemos byte de infrarrojos y bits de tacto"))    ;[3]
   (jsr EnviaCar          (comment "Enviamos byte de infrarrojos y bits de tacto"))   ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))               ;[3]
no_byte_infrarr

   (brclr QueDatos2 #x40 fin_inclin  (comment "Miramos si se desea valor deinclinom.")) ;[6]
   (ldaa (+ InSpi 17)     (comment "Obtenemos en B byte con valor del inclinometro"))   ;[4]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")       ;[2]
   (bne no_masc_inc       (comment "Si no es un 255 se envia directamente"))            ;[3]
   (anda (! #xFE)         (comment "Si es 255, valor cabecera, cortamos a 254"))        ;[2]
no_masc_inc
   (jsr EnviaCar          (comment "Enviamos byte de inclinometro"))                  ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))               ;[3]
fin_inclin

   (brclr QueDatos2 #x20 fin_piros  (comment "Miramos si se desea valor de los piros")) ;[6]
   (ldaa TipoCiclo        (comment "Consultamos en que tipo de ciclo estamos"))         ;[4]
   (bne trama1)                                                                         ;[3]
   (ldaa (& ADR2)         (comment "Leemos en A valor de ADR2=piro2"))                  ;[4]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")       ;[2]
   (bne no_masc_piro2     (comment "Si no es un 255 se envia directamente"))            ;[3]
   (anda (! #xFE)         (comment "Aseguramos que valor de piro2 no llega a 255"))     ;[2]
no_masc_piro2
   (jsr EnviaCar          (comment "Enviamos el byte de piro2"))                      ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))               ;[3]

   (ldaa (& ADR3)         (comment "Leemos en B valor de ADR3=piro3"))                  ;[4]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")       ;[2]
   (bne no_masc_piro3     (comment "Si no es un 255 se envia directamente"))            ;[3]
   (anda (! #xFE)         (comment "Aseguramos que valor de piro3 no llega a 255"))     ;[2]
no_masc_piro3
   (jsr EnviaCar          (comment "Enviamos el byte de piro3"))                      ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))               ;[3]
   (bra fin_piros)                                                                      ;[3]
trama1
   (ldaa (& ADR3)         (comment "Leemos en A valor de ADR3=piro1"))                  ;[4]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")       ;[2]
   (bne no_masc_piro1     (comment "Si no es un 255 se envia directamente"))            ;[3]
   (anda (! #xFE)         (comment "Aseguramos que valor de piro1 no llega a 255"))     ;[2]
no_masc_piro1
   (jsr EnviaCar          (comment "Enviamos el byte de piro1"))                      ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))          ;[3]
```

```
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))              ;[3]

   (ldaa (& ADR1)         (comment "Leemos en A valor de ADR1=piro4"))                 ;[4]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")      ;[2]
   (bne no_masc_piro4     (comment "Si no es un 255 se envia directamente"))           ;[3]
   (anda (! #xFE)         (comment "Aseguramos que valor de piro4 no llega a 255"))    ;[2]
no_masc_piro4
   (jsr EnviaCar          (comment "Enviamos el byte de piro4"))                     ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))         ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))              ;[3]

   (ldaa (& ADR2)         (comment "Leemos en D valor de ADR2=piro5"))                 ;[4]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")      ;[2]
   (bne no_masc_piro5     (comment "Si no es un 255 se envia directamente"))           ;[3]
   (anda (! #xFE)         (comment "Aseguramos que valor de piro5 no llega a 255"))    ;[2]
no_masc_piro5
   (jsr EnviaCar          (comment "Enviamos el byte de piro5"))                     ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))         ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))              ;[3]
fin_piros

   (brclr QueDatos2 #x10 fin_bat  (comment "Miramos si se desea el valor de baterias")) ;[6]
   (ldaa (& ADR4)         (comment "Leemos en A valor de ADR4=bateria elect. o motor")) ;[4]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")      ;[2]
   (bne no_masc_bat       (comment "Si no es un 255 se envia directamente"))           ;[3]
   (anda (! #xFE)         (comment "Aseguramos que valor de bateria no llega a 255"))  ;[2]
no_masc_bat
   (jsr EnviaCar          (comment "Enviamos el byte de baterias"))                  ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))         ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))              ;[3]
fin_bat

   (ldab TipoCiclo        (comment "Consultamos TipoCiclo para programacion del A/D")) ;[3]
   (beq ciclo0            (comment "y refresco nivel bateria motores.Si TipoCiclo==1")) ;[3]
   (ldaa (! #x30)         (comment "Proximo ciclo A/D: piros 2,3 y bat. electronica")) ;[2]
   (bra fin_set_AD)                                                                    ;[3]
ciclo0                    (comment "Si TipoCiclo==0")
   (ldaa (! #x34)         (comment "Proximo ciclo A/D: piros 1,4,5 y bateria motores")) ;[2]
fin_set_AD
   (staa (& ADCTL)        (comment "Programamos A/D para el proximo ciclo"))           ;[4]

   (ldaa Checksum         (comment "Obtenemos el valor del checksum calculado"))       ;[3]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")      ;[2]
   (bne no_masc_chksm1    (comment "Si no es un 255 se envia directamente"))           ;[3]
   (anda (! #xFE)         (comment "Mascara para evitar que el checksum llegue a 255")) ;[2]
no_masc_chksm1
   (jsr EnviaCar          (comment "Enviamos el byte de checksum"))                  ;[6+131]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))         ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))              ;[3]

   (ldaa TipoCiclo        (comment "Consultamos el tipo de ciclo para cambiarlo"))     ;[3]
   (inca                  (comment "Incrementamos el tipo de ciclo"))                  ;[2]
   (anda (! #x01)         (comment "Mascara para que el tipo de ciclo sea 0 o 1"))     ;[2]
   (staa TipoCiclo        (comment "Guardamos el nuevo tipo de ciclo"))                ;[3]

   (pulb                  (comment "Restauramos el valor del registro B de la pila"))  ;[4]
   (pula                  (comment "Restauramos el valor del registro A de la pila"))  ;[4]
   (rts                   (comment "Acabamos la subrutina"))                           ;[5]
;Total ciclos de la subrutina GeneraTrama [3396], peor caso maximo [2912]


(comment "***************************** Rutina RecibeTrama **************************")
;Rutina que implementa la maquina de estados que se encarga de recibir tramas. La subrutina se
;queda ejecutando la maquina de estados infinitamente hasta que llega una spi y pone la
;variable NuevoCiclo a uno indicando el inicio de un nuevo ciclo. En cuanto se detecta esta
;situacion se sale de la subrutina ignorando por completo el estado en que estabamos y la
;trama que se estaba recibiendo ya que en principio solo deberiamos recibir una trama por
;ciclo.

RecibeTrama
   (psha                  (comment "Salvamos el valor del registro A en la pila"))     ;[3]
   (pshb                  (comment "Salvamos el valor del registro B en la pila"))     ;[3]
   (pshy                  (comment "Salvamos el valor del registro Y en la pila"))     ;[5]

espera_SOH
   (brset NuevoCiclo #x01 fin_tr1    (comment "Acabamos si nueva spi, nuevo ciclo=1"))  ;[6]
   (jsr RecibeCar         (comment "Intentamos leer un caracter"))                   ;[6+66]
   (ldab CarReb           (comment "Consultamos si hay caracter disponible en A"))     ;[3]
   (beq espera_SOH        (comment "Si no hemos leido ningun caracter esperamos"))     ;[3]
   (cmpa (! SOH)          (comment "Comprobamos si es un inicio de trama"))            ;[2]
   (bne espera_SOH        (comment "Si no inicio trama esperamos siguiente caracter")) ;[3]
tipo_trama
```

```
   (clr Checksum          (comment "Ponemos a cero variable para calculo checksum"))    ;[6]
   (brset NuevoCiclo #x01 fin_tr1    (comment "Acabamos si nueva spi, nuevo ciclo "))   ;[6]
   (jsr RecibeCar         (comment "Intentamos leer un caracter"))                     ;[6+66]
   (ldab CarReb           (comment "Consultamos si hay caracter disponible en A"))     ;[3]
   (beq tipo_trama        (comment "Si no hemos leido ningun caracter esperamos"))     ;[3]
   (cmpa (! SOH)          (comment "Comprobamos si es un inicio de trama"))            ;[2]
   (beq tipo_trama        (comment "Si inicio de trama esperamos en el mismo estado")) ;[3]
   (ldy (! TramaIn)       (comment "Cargamos en Y puntero buffer para guardar trama ")) ;[4]
   (staa (&y 0)           (comment "Guardamos el primer caracter"))                    ;[5]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))         ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))              ;[3]
   (iny                   (comment "Incrementamos puntero a buffer"))                  ;[4]
   (lsra                  (comment "Desplazamos 4 bits a la derecha para comprobar"))  ;[2]
   (lsra                  (comment "el tipo de trama"))                                ;[2]
   (lsra)                                                                              ;[2]
   (lsra)                                                                              ;[2]
   (staa TipoTrama        (comment "Guardamos de que tipo de trama se trata"))         ;[3]
   (cmpa (! #x01)         (comment "Comprobamos si trama de tipo1, la mas habitual"))  ;[2]
   (beq trama_recep1)                                                                  ;[3]
   (cmpa (! #x00)         (comment "Comprobamos si trama de tipo0"))                   ;[2]
   (bne espera_SOH        (comment "Tipo trama incorrecto, ignoramos hasta nuevo SOH")) ;[3]

   (ldaa (! #x01)         (comment "Longitud trama tipo0: 1 bytes datos mas"))         ;[2]
   (staa LonTramaRecep    (comment "Guardamos la longitud que queda por recibir"))     ;[3]
   (bra seg_datos)                                                                     ;[3]
trama_recep1
   (ldaa (! #x0F)         (comment "Longitud trama tipo1: 15 bytes datos mas"))        ;[2]
   (staa LonTramaRecep    (comment "Guardamos la longitud que queda por recibir"))     ;[3]

seg_datos
   (brset NuevoCiclo #x01 fin_tr1    (comment "Acabamos si nueva spi, nuevo ciclo "))   ;[6]
   (jsr RecibeCar         (comment "Intentamos leer un caracter"))                     ;[6+66]
   (ldab CarReb           (comment "Consultamos si hay caracter disponible en A"))     ;[3]
   (beq seg_datos         (comment "Si no hemos leido ningun caracter esperamos"))     ;[3]
   (cmpa (! SOH)          (comment "Comprobamos si es un inicio de trama"))            ;[2]
   (beq tipo_trama        (comment "Si inicio de trama vamos a estado tipo_trama"))    ;[3]
   (staa (&y 0)           (comment "Guardamos el caracter recibido"))                  ;[5]
   (adda Checksum         (comment "Sumamos nuevo byte a valor del checksum"))         ;[3]
   (staa Checksum         (comment "Guardamos nuevo valor del checksum"))              ;[3]
   (iny                   (comment "Incrementamos puntero a buffer"))                  ;[4]
   (dec LonTramaRecep     (comment "Decrementamos numero bytes trama por recibir"))    ;[2]
   (bne seg_datos         (comment "Seguimos leyendo mientras trama no completa"))     ;[3]
   (bra recibe_checksum)                                                               ;[3]

fin_tr1
   (puly                  (comment "Restauramos el valor del registro Y de la pila"))  ;[6]
   (pulb                  (comment "Restauramos el valor del registro B de la pila"))  ;[4]
   (pula                  (comment "Restauramos el valor del registro A de la pila"))  ;[4]
   (rts                   (comment "Acabamos la subrutina"))                           ;[5]

recibe_checksum
   (brset NuevoCiclo #x01 fin_tr1    (comment "Acabamos si nueva spi, nuevo ciclo "))   ;[6]
   (jsr RecibeCar         (comment "Intentamos leer un caracter"))                     ;[6+66]
   (ldab CarReb           (comment "Consultamos si hay caracter disponible en A"))     ;[3]
   (beq recibe_checksum   (comment "Si no hemos leido ningun caracter esperamos"))     ;[3]
   (cmpa (! SOH)          (comment "Comprobamos si es un inicio de trama"))            ;[2]
   (beq tipo_trama        (comment "Si inicio de trama vamos a estado tipo_trama"))    ;[3]
   (staa (&y 0)           (comment "Guardamos el checksum recibido"))                  ;[5]
   (ldaa Checksum         (comment "Obtenemos el checksum calculado"))                 ;[3]
   (cmpa (! SOH))         (comment "Comprobamos si es un 255, valor de cabecera")      ;[2]
   (bne no_masc_chksm2    (comment "Si no es un 255 se envia directamente"))           ;[3]
   (anda (! #xFE)         (comment "Mascara para evitar que el checksum llegue a 255")) ;[2]
no_masc_chksm2
   (cmpa (&y 0)           (comment "Comparamos checksum calculado con el de la Trama")) ;[5]
   (bne espera_SOH        (comment "Checksum incorrecto, ignoramos la trama"))         ;[3]

trama_correcta
   (comment "Trama correcta con checksum correcto")
   (brset NuevoCiclo #x01 fin_tr2    (comment "Acabamos si  nueva spi, nuevo ciclo "))  ;[6]
   (ldaa TramaIn          (comment "Obtenemos primer byte de que datos espera host"))  ;[4]
   (anda (! #x0F)         (comment "Ponemos a cero los bits que no importan"))         ;[2]
   (staa QueDatos1        (comment "Guardamos en QueDatos1"))                          ;[3]
   (ldaa (+ TramaIn 1)    (comment "Obtenemos segundo byte de que datos espera host")) ;[4]
   (anda (! #xF8)         (comment "Ponemos a cero los bits que no importan"))         ;[2]
   (staa QueDatos2        (comment "Guardamos en QueDatos2"))                          ;[3]
   (ldaa TipoTrama        (comment "Consultamos el tipo de trama recibida"))           ;[3]
   (beq fin_tr2           (comment "Si trama tipo0, no hay ordenes para el servo"))    ;[3]
   (ldaa (+ TramaIn 1)    (comment "Recuperamos bit de buzzer del segundo byte trama")) ;[4]
   (anda (! #x01)         (comment "Ignoramos el resto de bits"))                      ;[2]
   (oraa (+ TramaIn 15)   (comment "A-adimos el bit al resto del byte"))               ;[4]
   (staa (+ OutSpiTmp 19) (comment "Guardamos orden buzzer en OutSpiTmp[19]"))         ;[4]
   (ldaa (+ TramaIn 1)    (comment "Recuperamos bit de leds del segundo byte trama"))  ;[4]
```

```
   (lsra                  (comment "Desplazamos un bit a la derecha"))                  ;[2]
   (anda (! #x01)         (comment "Ignoramos el resto de bits"))                       ;[2]
   (oraa (+ TramaIn 14)   (comment "Añadimos el bit al resto del byte"))                ;[4]
   (staa (+ OutSpiTmp 18) (comment "Guardamos orden de leds en OutSpiTmp[18]"))         ;[4]
   (ldaa (+ TramaIn 2)    (comment "Recuperamos orden a motor alfa0"))                  ;[4]
   (staa (+ OutSpiTmp 10) (comment "Guardamos orden a motor alfa0 en OutSpiTmp[10]"))   ;[4]
   (ldaa (+ TramaIn 3)    (comment "Recuperamos orden a motor alfa1"))                  ;[4]
   (staa (+ OutSpiTmp 8)  (comment "Guardamos orden a motor alfa1 en OutSpiTmp[8]"))    ;[4]
   (ldaa (+ TramaIn 4)    (comment "Recuperamos orden a motor alfa2"))                  ;[4]
   (staa (+ OutSpiTmp 14) (comment "Guardamos orden a motor alfa2 en OutSpiTmp[14]"))   ;[4]
   (ldaa (+ TramaIn 5)    (comment "Recuperamos orden a motor alfa3"))                  ;[4]
   (staa (+ OutSpiTmp 12) (comment "Guardamos orden a motor alfa3 en OutSpiTmp[12]"))   ;[4]
   (ldaa (+ TramaIn 6)    (comment "Recuperamos orden a motor alfa4"))                  ;[4]
   (staa (+ OutSpiTmp 6)  (comment "Guardamos orden a motor alfa4 en OutSpiTmp[6]"))    ;[4]
   (ldaa OutSpiTmp        (comment "Obtenemos el valor de nivel de baterias"))          ;[4]
   (anda (! #xC0)         (comment "Conservamos dos bits mayor peso, resto a cero"))    ;[2]
   (oraa (+ TramaIn 7)    (comment "Añadimos orden alfa5 a bits de nivel baterias"))    ;[4]
   (staa OutSpiTmp        (comment "Orden mot. alfa5 y nivel bat. en OutSpiTmp[0]"))    ;[4]
   (ldaa (+ TramaIn 8)    (comment "Recuperamos orden a motor beta0"))                  ;[4]
   (staa (+ OutSpiTmp 11) (comment "Guardamos orden a motor beta0 en OutSpiTmp[11]"))   ;[4]
   (ldaa (+ TramaIn 9)    (comment "Recuperamos orden a motor beta1"))                  ;[4]
   (staa (+ OutSpiTmp 9)  (comment "Guardamos orden a motor beta1 en OutSpiTmp[9]"))    ;[4]
   (ldaa (+ TramaIn 10)   (comment "Recuperamos orden a motor beta2"))                  ;[4]
   (staa (+ OutSpiTmp 15) (comment "Guardamos orden a motor beta2 en OutSpiTmp[15]"))   ;[4]
   (ldaa (+ TramaIn 11)   (comment "Recuperamos orden a motor beta3"))                  ;[4]
   (staa (+ OutSpiTmp 13) (comment "Guardamos orden a motor beta3 en OutSpiTmp[13]"))   ;[4]
   (ldaa (+ TramaIn 12)   (comment "Recuperamos orden a motor beta4"))                  ;[4]
   (staa (+ OutSpiTmp 5)  (comment "Guardamos orden a motor beta4 en OutSpiTmp[5]"))    ;[4]
   (ldaa (+ TramaIn 13)   (comment "Recuperamos orden a motor beta5"))                  ;[4]
   (staa (+ OutSpiTmp 7)  (comment "Guardamos orden a motor beta5 en OutSpiTmp[7]"))    ;[4]
   (ldaa (! #x01))                                                                     ;[2]
   (staa NuevasOrdenes    (comment "Indicamos que hay nuevas ordenes disponibles"))    ;[3]

fin_tr2
   (puly                  (comment "Restauramos el valor del registro Y de la pila"))   ;[6]
   (pulb                  (comment "Restauramos el valor del registro B de la pila"))   ;[4]
   (pula                  (comment "Restauramos el valor del registro A de la pila"))   ;[4]
   (rts                   (comment "Acabamos la subrutina"))                            ;[5]
;Total ciclos del codigo RecibeTrama [677], peor caso maximo variable.


(comment "************************ INTERRUPCION DE LINEA SERIE ****************************")
;Interrupcion de linea serie. Se encarga de coger los caracteres que llegan por linea serie y
;guardarlos en el buffer de entrada "InBuffer" y de coger los caracteres que se quieren enviar
;por linea serie disponibles en el buffer "OutBuffer" y enviarlos por la linea serie.
;En caso de error de recepcion se ignora el caracter recibido.

int_ls
   (ldaa (& SCCR2)        (comment "Guardamos el valor del registro SCCR2"))             ;[4]
   (ldab (! #x0C)         (comment "Nuevo valor para el registro SCCR2"))                ;[2]
   (stab (& SCCR2)        (comment "Inhabilitamos las interrupciones de linea serie"))   ;[4]
   (cli                   (comment "Permitimos interrupciones por si llega una de SPI")) ;[2]
   (staa ValSccr2         (comment "Salvamos el valor del SCCR2 en una variable"))       ;[3]
   (ldaa (& SCSR)         (comment "Consulta del valor del SCSR, estatus linea serie"))  ;[4]
   (staa ValScsr          (comment "Guardamos el valor del SCSR en variable ValScsr"))   ;[3]

   (brclr ValScsr #x0E si_car_reb (comment "Si no error recep. mirar si car. recibido")) ;[6]
   (ldaa (& SCDR)         (comment "Leemos SCDR para limpiar bits de error del SCSR"))   ;[4]
   (bra si_car_env        (comment "No interesa caracter recibido, puede que erroneo"))  ;[3]

si_car_reb
   (brclr ValScsr #x20 si_car_env (comment "Si no nuevo caract. miramos si fin trans.")) ;[6]
   (ldaa (& SCDR)         (comment "Consultamos el caracter recibido"))                  ;[4]
   (ldab InBuffer_Count   (comment "Consultamos numero caracteres buffer de salida"))    ;[4]
   (cmpb (! TAM_BUF_LS)   (comment "Si buffer salida esta lleno se pierde el caracter")) ;[2]
   (beq si_car_env        (comment "error overrun buffer, esperamos proximo caracter"))  ;[3]
   (ldy (! InBuffer)      (comment "Guardamos en Y el puntero al inicio de InBuffer"))   ;[4]
   (ldab InBuffer_In      (comment "Guardamos en B offset a primera posicion libre"))    ;[4]
   (aby                   (comment "Obtenemos el puntero a  primera posicion libre"))    ;[4]
   (staa (&y 0)           (comment "Guardamos caracter recibido en la posicion libre"))  ;[5]
   (incb                  (comment "Incrementamos offset a siguiente posicion libre"))   ;[2]
   (andb (! MOD_TAM_BUF_LS)    (comment "Hacemos el modulo TAM_BUF_LS al nuevo offset")) ;[2]
   (stab InBuffer_In      (comment "Guardamos el nuevo offset en InBuffer_In"))          ;[4]
   (inc InBuffer_Count    (comment "Incrementamos el numero de bytes en OutBuffer"))     ;[6]

si_car_env
   (brclr ValSccr2 #x80 fin_ls   (comment "Si no interrupcion de transmision, acabamos"));[7]
   (brclr ValScsr #x80 fin_ls    (comment "Si TDRE=0 acabamos, no fin de transmision"))  ;[6]
   (ldab OutBuffer_Count  (comment "Si no hay caracteres recibidos disponibles"))       ;[4]
   (bne hay_cars          (comment "acabamos"))                                         ;[3]
   (bclr ValSccr2 #x80    (comment "Inhibimos interrupciones transmision (al acabar)")) ;[7]
   (bra fin_ls            (comment "y acabamos"))                                       ;[3]
```

```
hay_cars
   (ldy (! OutBuffer)     (comment "Guardamos en Y el puntero al inicio de OutBuffer"))  ;[4]
   (ldab OutBuffer_Out    (comment "Guardamos en B offset primera posicion disponible")) ;[4]
   (aby                   (comment "Obtenemos puntero a primera posicion disponible"))   ;[4]
   (ldaa (&y 0)           (comment "Guardamos en A el siguiente caracter a enviar"))     ;[5]
   (incb                  (comment "Increment. offset a primera posicion disponible"))   ;[2]
   (andb (! MOD_TAM_BUF_LS) (comment "Offset=Offset MOD TAM_BUF_LS"))                    ;[2]
   (stab OutBuffer_Out    (comment "Guardamos el nuevo offset en OutBuffer_Out"))        ;[4]
   (dec OutBuffer_Count   (comment "Decrementamos el numero de bytes en OutBuffer"))     ;[6]
   (staa (& SCDR)         (comment "Enviamos caracter en A por linea serie"))            ;[4]

fin_ls
   (ldaa ValSccr2         (comment "Recuperamos el valor del SCCR2"))                    ;[3]
   (staa (& SCCR2)        (comment "Restauramos el valor del SCCR2"))                    ;[4]
   (comment "A partir de este momento pueden llegar otras interrupciones de linea serie y")
   (comment "podria suceder que estas se encadenaran infinitamente y se desbordara la pila.")
   (comment "Esto no sucede debido a que el tiempo de enviar o recibir un caracter es mucho")
   (comment "mayor que la duracion de esta rutina.")
   (rti                   (comment "Fin de la rutina de interrupcion de linea serie"))   ;[12]
;Total ciclos interrupcion int_ls [169], peor caso maximo [152]


(comment "*************************** INTERRUPCION DE SPI **************************")
;Rutina de interrupcion de Serial Periferal Interface que se encarga de recibir los datos
;procedentes del SERVO y envia las ordenes procedentes de la linea serie. Si no tenemos nuevas
;ordenes se cogen los datos disponibles en OutSpi correspondientes a ordenes de ciclos
;anteriores o ordenes iniciales inicializadas al principio del programa. En caso de disponer
;de nuevas ordenes procedentes de la linea serie se obtienen del buffer OutSpiTmp y despues de
;enviarlas al SERVO se pasan al buffer OutSpiTmp (los datos no se pasan al buffer OutSpi a la
;vez que se envian al SERVO ya que en caso de fallo de la comunicacion tendriamos datos
;inconsistentes en OutSpi).

int_spi
   (ldaa (! #x4C))                                                                    ;[2]
   (staa (& SPCR)         (comment "Inhibimos las interrupciones de spi"))            ;[4]
   (ldaa (& SPSR)         (comment "Leemos SPSR para limpiar el bit SPIF"))           ;[4]
   (ldaa (& SPDR)         (comment "Obtenemos numero de bytes que nos van a enviar")) ;[4]
   (cmpa (! 21)           (comment "Comprobamos que el master nos envia 21 bytes"))   ;[2]
   (bne fin_spi           (comment "Si no nos envian 21 bytes: error y acabamos"))    ;[3]
   (staa TamInfoSpi       (comment "Guardamos el numero de bytes a recibir"))         ;[3]
   (ldy (! InSpi)         (comment "Direccion del buffer de entrada de la spi"))      ;[4]
   (ldaa NuevasOrdenes    (comment "Miramos si hay nuevas ordenes para enviar SERVO")) ;[3]
   (bne ordenes_nuevas    (comment "Si nuevas ordenes buffer salida es SpiOutTmp"))   ;[3]
   (ldaa (&y TAM_BUF_SPI) (comment "Guardamos en A el primer byte a enviar al SERVO")) ;[5]
   (staa (& SPDR)         (comment "Enviamos el primer valor por la spi"))            ;[4]
quedan_bytes1
   (ldab (! 100)          (comment "Activamos un timeout de 100"))                    ;[2]
timeout1
   (decb                  (comment "Decrementamos el timeout"))                       ;[2]
   (beq fin_spi           (comment "Timeout superado, acabamos"))                     ;[3]
   (brclr (& SPSR) #x80 timeout1 (comment "Mientras no nuevo byte, decrementar timeout"));[7]
   (ldaa (& SPDR)         (comment "Leemos el nuevo byte recibido"))                  ;[4]
   (staa (&y 0)           (comment "Guardamos el byte recibido en buffer de entrada")) ;[5]
   (iny                   (comment "Incrementamos offset buffer de entrada y salida")) ;[4]
   (ldaa (&y TAM_BUF_SPI) (comment "Obtenemos en A el proximo byte a enviar"))        ;[5]
   (staa (& SPDR)         (comment "Enviamos el nuevo byte por la spi al SERVO"))     ;[4]
   (dec TamInfoSpi        (comment "Decrementamos el numero de bytes a recibir"))     ;[6]
   (bne quedan_bytes1     (comment "Si quedan bytes por recibir los esperamos"))      ;[3]
   (bra fin_spi           (comment "Fin de la comunicacion con el SERVO"))            ;[3]

ordenes_nuevas
   (ldaa (&y 2TAM_BUF_SPI)(comment "Guardamos en A el primer byte a enviar al SERVO")) ;[5]
   (staa (& SPDR)         (comment "Enviamos el primer valor por la spi"))            ;[4]
quedan_bytes2
   (ldab (! 100)          (comment "Activamos un timeout de 100"))                    ;[2]
timeout2
   (decb                  (comment "Decrementamos el timeout"))                       ;[2]
   (beq fin_spi           (comment "Timeout superado, acabamos"))                     ;[3]
   (brclr (& SPSR) #x80 timeout2 (comment "Mientras no nuevo byte, decrementar timeout"));[7]
   (ldaa (& SPDR)         (comment "Leemos el nuevo byte recibido"))                  ;[4]
   (staa (&y 0)           (comment "Guardamos el byte recibido en buffer de entrada")) ;[5]
   (iny                   (comment "Incrementamos offset buffer de entrada y salida")) ;[4]
   (ldaa (&y 2TAM_BUF_SPI)(comment "Obtenemos en A el proximo byte a enviar"))        ;[5]
   (staa (& SPDR)         (comment "Enviamos el nuevo byte por la spi al SERVO"))     ;[4]
   (dec TamInfoSpi        (comment "Decrementamos el numero de bytes a recibir"))     ;[6]
   (bne quedan_bytes2     (comment "Si quedan bytes por recibir los esperamos"))      ;[3]

fin_spi
   (ldaa (! SOH))                                                                     ;[2]
   (jsr EnviaCar          (comment "Enviamos la cabecera de la trama"))               ;[6+131]
   (ldaa (! #x01)         (comment "Valor para indicar nuevo ciclo"))                 ;[2]
   (staa NuevoCiclo       (comment "Activamos indicador nuevo ciclo: valor mayor cero"));[3]
```

```
   (ldaa (! #xCC))                                                              ;[2]
   (staa (& SPCR)         (comment "Habilitamos las interrupciones por spi"))   ;[4]
   (rti                   (comment "Fin de la rutina de interrupcion de spi"))  ;[12]
;Total ciclos interrupcion int_spi [301], peor caso maximo [1147]


(comment "****************** RUTINA PARA INTERRUPCIONES NO DESEADAS ******************")
bad_int                   (comment "No hace nada")
   (rti                   (comment "Fin de la rutina de interrupcion")) ;[12]


(comment "******************** INICIALIZACION VECTOR INTERRUPCIONES ******************")
   (= #xFFD6)
   (!16 int_ls            (comment "Interrupcion de SCI Serial System"))
   (!16 int_spi           (comment "Interrupcion de SPI Serial Transfer Complete"))
   (!16 bad_int           (comment "Interrupcion de Pulse Accumulator Input Edge"))
   (!16 bad_int           (comment "Interrupcion de Pulse Accumulator Overflow"))
   (!16 bad_int           (comment "Interrupcion de Timer Overflow"))
   (!16 bad_int           (comment "Interrupcion de Timer Output Compare 5"))
   (!16 bad_int           (comment "Interrupcion de Timer Output Compare 4"))
   (!16 bad_int           (comment "Interrupcion de Timer Output Compare 3"))
   (!16 bad_int           (comment "Interrupcion de Timer Output Compare 2"))
   (!16 bad_int           (comment "Interrupcion de Timer Output Compare 1"))
   (!16 bad_int           (comment "Interrupcion de Timer Input Capture 3"))
   (!16 bad_int           (comment "Interrupcion de Timer Input Capture 2"))
   (!16 bad_int           (comment "Interrupcion de Timer Input Capture 1"))
   (!16 bad_int           (comment "Interrupcion de Real Time Interrupt"))
   (!16 bad_int           (comment "Interrupcion de IRQ"))
   (!16 bad_int           (comment "Interrupcion de XIRQ"))
   (!16 bad_int           (comment "Interrupcion de SWI"))
   (!16 inicializar       (comment "Interrupcion de Illegal Opcode Trap"))
   (!16 bad_int           (comment "Interrupcion de COP Failure"))
   (!16 bad_int           (comment "Interrupcion de COP Clock Monitor Fail"))
   (!16 inicializar       (comment "Interrupcion de RESET"))
   ))
```

ANEXO 6. CÓDIGO FUENTE DEL PC

A continuación se adjunta el código fuente del PC correspondiente a cada uno de los módulos implementados. En cada caso se indica el nombre del fichero al que corresponde. Para cada módulo se tiene un fichero de cabecera (o definición) con extensión ".h" y un fichero de implementación con extensión ".h".

Fichero "buffers.h"

```
#ifndef _BUFFERS
#define _BUFFERS

#define TAM_BUFFER 1024

typedef struct
{  unsigned char mem[TAM_BUFFER];    /* Buffer de 1024 caracteres */
   int in,out;                       /* Punteros de entrada y salida del buffer */
   int count;                        /* Numero de caracteres contenidos por el buffer*/
} buffer;

/* Funcion que inicializa el buffer b. En caso de contener algo se pierde
   su contenido. En caso de puntero nulo se aborta la ejecucion del programa
   en curso */
void IniBuffer(buffer *b);

/* Funcion que inserta un caracter en el buffer b. Si se inserta correcta-
   mente retorna 1 y si no hay espacio retorna 0. En caso de puntero nulo
   se aborta la ejecucion del programa en curso */
int WriteCharBuffer(buffer *b,unsigned char c);

/* Funcion que saca un caracter del buffer b. Si todo va bien retorna 1 y
   si no hay caracteres disponibles retorna 0. En caso de puntero nulo
   se aborta la ejecucion del programa en curso */
int ReadCharBuffer(buffer *b,unsigned char *c);

/* Funcion que inserta una cadena de caracteres en el buffer b, como
   parametro hay que indicar la longitud de la cadena. Si se puede
   insertar toda la cadena y todo va bien retorna 1, en caso de que
   no haya espacio para TODA la cadena retorna 0. En caso de que algun
   puntero sea nulo se aborta la ejecucion del programa en curso */
int WriteBuffer(buffer *b,unsigned char *str,unsigned int longitud);

/* Funcion que saca una cadena de caracteres del buffer b, como parametro
   hay que indicar el numero de caracteres que se quiere leer. Si hay
   caracteres disponibles y todo va bien retorna 1 y en "longitud" el
   indica el numero de caracteres leidos (menor o igual que el numero
   requerido) y si no hay caracteres disponibles retorna 0. En caso de que
   algun puntero sea nulo se aborta la ejecucion del programa en curso */
int ReadBuffer(buffer *b,unsigned char *str,unsigned int *longitud);

#endif
```

Fichero "buffers.c"

```
#include <process.h>
#include "buffers.h"

/* ********************************************************************** */
/*                              IniBuffer                                 */
/* ********************************************************************** */

void IniBuffer(buffer *b)
{  if(!b)             /* Error, puntero nulo */
      exit(-1);       /* Se aborta el programa */

   b->in=0;           /* Inicializacion del puntero de entrada del buffer */
   b->out=0;          /* Inicializacion del puntero de salida del buffer */
```

```
   b->count=0;        /* Inicializacion del contador del buffer */
}


/* ********************************************************************** */
/*                            NumCharsBuffer                              */
/* ********************************************************************** */
int NumCharsBuffer(buffer *b)
{  if(!b)                     /* Error, puntero nulo */
      exit(-1);               /* Se aborta el programa */
   return(b->count);          /* Tamaño del contenido del buffer */
}


/* ********************************************************************** */
/*                          WriteCharBuffer                               */
/* ********************************************************************** */

int WriteCharBuffer(buffer *b,unsigned char c)
{  if(!b)                     /* Error, puntero nulo */
      exit(-1);               /* Se aborta el programa */

   if(b->count<TAM_BUFFER)    /* Si hay espacio libre en el buffer */
   {  b->mem[b->in]=c;        /* Guardamos el caracter c en el buffer */
      b->in++;                /* Incrementamos puntero a primera posicion libre */
      b->in%=TAM_BUFFER;      /* Controlamos de no superar el tama¤o del buffer */
      b->count++;             /* Incrementamos el numero de caracteres del buffer */
      return(1);              /* Caracter encolado correctamente en el buffer */
   }
   /* No hay espacio en la cola */
   return(0);                 /* Caracter no encolado en el buffer */
}


/* ********************************************************************** */
/*                           ReadCharBuffer                               */
/* ********************************************************************** */

int ReadCharBuffer(buffer *b,unsigned char *c)
{  if(!b)                     /* Error, puntero nulo */
      exit(-1);               /* Se aborta el programa */

   if(b->count>0)             /* Si hay caracteres en el buffer */
   {  *c=b->mem[b->out];      /* Leemos en c el primer caracter disponible */
      b->out++;               /* Incrementamos puntero a primer caracter disponible */
      b->out%=TAM_BUFFER;     /* Controlamos de no superar el tama¤o del buffer */
      b->count--;             /* Decrementamos el numero de caracteres del buffer */
      return(1);              /* Caracter desencolado correctamente del buffer */
   }
   /* No hay caracteres en el buffer */
   return(0);                 /* No hay caracter desencolado del buffer */
}


/* ********************************************************************** */
/*                            WriteBuffer                                 */
/* ********************************************************************** */

int WriteBuffer(buffer *b,unsigned char *str,unsigned int longitud)
{  int i;

   if(!b || !str)                   /* Error, puntero nulo */
      exit(-1);                     /* Se aborta el programa */

   if(longitud<=TAM_BUFFER-(b->count))      /* Si cabe todo el string en el buffer */
   {  for(i=0;i<longitud;i++)       /* Se encolan los caracteres uno por uno */
      {  b->mem[b->in]=str[i];
         b->in++;
         b->in%=TAM_BUFFER;
      }
      b->count+=longitud;           /* Sumamos la longitud del string al contador */
      return(1);                    /* String encolado correctamente */
   }
   /* No hay espacio en el buffer */
   return(0);                       /* No hay espacio para encolar el string */
}


/* ********************************************************************** */
/*                             ReadBuffer                                 */
/* ********************************************************************** */
```

```
int ReadBuffer(buffer *b,unsigned char *str,unsigned int *longitud)
{  unsigned int num_cars,i;

   if(!b || !str)                    /* Error, puntero nulo */
      exit(-1);                      /* Se aborta el programa */

   if(b->count>0)                    /* Si hay caracteres en el buffer */
   {  if(*longitud>b->count)         /* Si nos piden mas de los que tenemos */
         num_cars=b->count;          /* desencolamos solo los disponibles */
      else                           /* sino */
         num_cars=*longitud;         /* desencolamos el total de los que se piden */

      *longitud=num_cars;            /* Longitud de los caracteres a desencolar */
      for(i=0;num_cars>0;num_cars--) /* Desencolamos caracteres uno a uno */
      {  str[i]=b->mem[b->out];
         b->out++;
         b->out%=TAM_BUFFER;
         b->count--;
         i++;
      }
      return(1);                     /* Caracteres desencolados del buffer */
   }
   /* No hay caracteres en el buffer */
   return(0);                        /* No hay caracteres desencolados del buffer */
}
```

Fichero "clock.h"

```
#ifndef _CLOCK
#define _CLOCK

/* Rutina que inicializa el clock del sistema con la rutina de clock cuyo
   puntero le pasamos en RutClk de manera que sera nuestra rutina la que se
   ejecute a cada interrupcion de clock */
void OpenClk(void interrupt (*IntClk)(void));

/* Rutina que restaura la interrupcion de clock del sistema */
void CloseClk(void);

#endif
```

Fichero "clock.c"

```
#include "clock.h"
#include <dos.h>

#define NIntClk 0x1C  /* Numero de interrupcion de int. de clock */

/* Puntero a la rutina de clock anterior */
void interrupt (*OldIntClk)();

/* ********************************************************************** */
/*                              OpenClk                                   */
/* ********************************************************************** */

void OpenClk(void interrupt (*IntClk)())
{  OldIntClk=getvect(NIntClk);       /* Guardamos la rutina de clock anterior */
   setvect(NIntClk,IntClk);          /* Inicializamos con la nueva rutina de clock */
}


/* ********************************************************************** */
/*                              CloseClk                                  */
/* ********************************************************************** */

void CloseClk(void)
{  setvect(NIntClk,OldIntClk);       /* Restauramos con la rutina de clock anterior */
}
```

Fichero "fisico.h"

```
#ifndef _FISICO
#define _FISICO

/* Definicion de tipos de error */
#define F_NO_ERROR                 0        /* No hay error */
#define F_ERROR_COM_INCORRECTO    -1        /* COM incorrecto, no entre 0-4 */
#define F_ERROR_COM_NO_DETECTADO  -2        /* COM indicado no detectado */
#define F_ERROR_NO_OPEN           -3        /* Nivel fisico no abierto */
#define F_ERROR_NO_CLOSE          -4        /* Nivel fisico no cerrado */
#define F_ERROR_PARAM_DESCONOCIDO -5        /* Parametro estado desconocido */
#define F_ERROR_ACCESO_BUFFER     -6        /* Buffer sin espacio o vacio */


/* Parametros para consultar el estado del nivel fisico. Todos los
   parametros para consultar el estado del nivel fisico comienzan por
   cero en hexadecimal. Esto permite a niveles superiores detectar que el
   parametro es de nivel fisico */
#define F_ERROR_OVERRUN        0x0001       /* Contador errores de overrun */
#define F_ERROR_PARIDAD        0x0002       /* Contador errores de paridad */
#define F_ERROR_TRAMA          0x0003       /* Contador errores de trama */
#define F_ERROR_BREAKLINE      0x0004       /* Contador errores de breakline */
#define F_ERROR_OVERRUN_BUFFER 0x0005       /* Contador errores overrun buffers */


/* Funcion que inicializa el nivel fisico para el COM y divisor de velocidad
   indicados como parametros. "ComId" puede ir de 1 a 4 y "DivVel" de 1 a
   65535. La velocidad de la linea serie sera 115200/DivVel. En caso de COM
   incorrecto o no detectado o que ya se hubiera abierto el nivel fisico
   retorna el error correspondiente, si todo va bien retorna F_NO_ERROR. */
int F_Open(unsigned char ComId,unsigned int DivVel);

/* Funcion que restaura los parametros de la linea serie excepto el estado
   del registro de control de modem (en nuestro caso control del modo y
   reset del robot). Solo se debe ejecutar si antes se ha llamado a F_Open,
   si no es asi retorna el error   F_NO_OPEN. Si la llamada se produce tras
   haber ejecutado F_Open y todo va bien retorna F_NO_ERROR. */
int F_Close(void);

/* Funcion que sirve de interficie para consultar los parametros del
   nivel fisico, principalmente contadores de errores. Se le pasa el
   parametro de nivel fisico que se quiere consultar y un puntero a la
   variable tipo long donde se quiere obtener el valor. En caso de que el
   nivel fisico no este inicializado retorna F_ERROR_NO_OPEN y si el
   parametro no corresponde a nivel   fisico o es desconocido se retorna el
   error F_ERROR_PARAM_DESCONOCIDO y en caso contrario si todo va bien
   retorna F_NO_ERROR. */
int F_State(unsigned int param,unsigned long *valor);

/* Funcion que sirve de interficie para obtener una cadena de caracteres
   del nivel fisico, almacenados en el buffer de entrada de ese nivel. Como
   parametros se indican el puntero al buffer de destino "str" y un puntero
   a la longitud (numero de caracteres) que se desean obtener. Si hay
   caracteres disponibles en el nivel fisico se retorna F_NO_ERROR y   en
   "longitud" se indica el numero de caracteres disponibles en "str" que
   sera menor o igual que el numero pedido en "longitud" en la llamada, si
   no hay ningun caracter disponible retorna F_ERROR_ACCESO_BUFFER. En caso
   de que el nivel fisico no este inicializado retorna F_ERROR_NO_OPEN. */
int F_Read(unsigned char *str,unsigned int *longitud);

/* Funcion que sirve de interficie para enviar una cadena de caracteres
   al nivel fisico, almacenandolos en el buffer de salida de ese nivel.
   El puntero "str" indica los datos que se quieren enviar y en "longitud"
   indicaremos la longitud (numero de caracteres a enviar). En   caso de que
   haya espacio para toda la cadena retorna F_NO_ERROR y en caso contrario
   retorna F_ERROR_ACCESO_BUFFER indicando que no hay espacio en el nivel
   fisico para toda la cadena y no guarda ningun caracter. En caso de que el
   nivel fisico no este inicializado retorna F_ERROR_NO_OPEN. */
int F_Write(unsigned char *str,unsigned int longitud);

/* Funcion que indica si en nivel fisico se esta enviando o no. En caso de
   que el nivel fisico no este inicializado retorna F_ERROR_NO_OPEN y en
   caso contrario retorna F_NO_ERROR y en vel se indica si se esta enviando
   actualmente (env=1) o si no (env=0). */
int F_Env(int *env);

/* Funcion que hace un reset del robot por software. Si todo va bien
```

```
   retorna F_NO_ERROR y si el nivel fisico no esta inicializado retorna
   F_NO_OPEN. */
int F_ResetRob(void);

#endif
```

Fichero "fisico.c"

```
#include <dos.h>
#include "fisico.h"
#include "buffers.h"
#include "enlace.h"

/* Registros de la linea serie */
unsigned int RBR,THR,IER,DLHB,DLLB,IIR,FCR,LCR,MCR,LSR,MSR;

/* Struct para guardar el estado del nivel fisico: valor de los registros
   de linea serie, interrupcion antigua, contadores de error ... */
struct
{  unsigned char dllb;                      /* Valor anterior DLLB */
   unsigned char dlhb;                      /* Valor anterior DLHB */
   unsigned char lcr;                       /* Valor anterior LCR */
//   unsigned char mcr;                     /* Valor anterior MCR */
   unsigned char ier;                       /* Valor anterior IER */
   unsigned char imr;                       /* Valor anterior IMR */
   void interrupt (*OldIntLs)();            /* Interrupcion linea serie anterior */
   unsigned char NInt;                      /* Numero interrupcion linea serie */

   unsigned long ErrorOverrun;              /* Contador errores overrun */
   unsigned long ErrorParidad;              /* Contador errores paridad */
   unsigned long ErrorTrama;                /* Contador errores trama */
   unsigned long BreakLine;                 /* Contador de breaklines */
   unsigned long ErrorOverrunBuffer;        /* Contador errores overrun buffers */
} EstadoFisico;


/* Variable para indicar si el nivel fisico esta inicializado o no */
int F_Init=0;

/* buffers de lectura y escritura para la linea serie */
buffer BufferIn,BufferOut;


/* ********************************************************************** */
/*                                 IntLs                                  */
/* ********************************************************************** */
/* Rutina de interrupcion de la linea serie. En caso de errores de linea
   serie modifica las variables de estado del nivel fisico. */

void interrupt IntLs()
{  unsigned int RegIir,RegLsr;
   unsigned char c;

   disable();                                   /* Inhibimos las interrupciones */
   RegIir=inportb(IIR);
   while(!(RegIir&0x01))
   {  switch(RegIir&0x06)
      {  // case 0x00:   Interrupcion cambio estado del modem inabilitada

         case 0x02:                             /* Final de transmision de caracter */
            if(ReadCharBuffer(&BufferOut,&c))   /* Lee caracter del buffer salida */
               outportb(THR,c);                 /* Enviamos caracter */
            break;

         case 0x04:                             /* Caracter valido */
            c=inportb(RBR);                     /* Leemos caracter recibido */
            E_CaracterIndication(c);            /* Indicamos a nivel superior */
            if(!WriteCharBuffer(&BufferIn,c))   /* Guardamos en el buffer entrada */
               EstadoFisico.ErrorOverrunBuffer++;   /* Error, buffer lleno */
            break;

         case 0x06:                             /* Error de recepcion de caracter */
            RegLsr=inportb(LSR);
            switch(RegLsr)
            {  case 0x02:                       /* Error de overrun de linea serie */
                  EstadoFisico.ErrorOverrun++;
```

```
                  break;
               case 0x04:                          /* Error de paridad de linea serie */
                  EstadoFisico.ErrorParidad++;
                  break;
               case 0x08:                          /* Error de trama de linea serie */
                  EstadoFisico.ErrorTrama++;
                  break;
               case 0x10:                          /* Error de breakline de linea serie */
                  EstadoFisico.BreakLine++;
                  break;
            }
            inportb(RBR);                          /* Limpiamos registro recepcion */
            break;
      }
      RegIir=inportb(IIR);                         /* Consultamos si interrupciones
pendientes */
   }
   outportb(0x20,0x20);                            /* EOI: final de interrupcion */
   enable();                                       /* Habilitamos las interrupciones */
}


/* ********************************************************************** */
/*                            DetectCOM                                   */
/* ********************************************************************** */
/* Rutina que se encarga de detectar la direccion base del COM indicado como
   parametro (0-4). Si este parametro es incorrecto (no 0-4) o si no se
   detecta el COM indicado   en la BIOS, se retorna el tipo de error
   correspondiente. Si el COM indicado se detecta correctamente se devuelve
   que no ha habido error y AddrCom contendra la direccion base. */

int DetectCOM(unsigned char ComId, unsigned int *AddrCom)
{  unsigned long BiosAddr;

   /* Si el puerto indicado no es correcto (0-4), acabar */
   if(ComId<1 || ComId>4)
      return(F_ERROR_COM_INCORRECTO);

   /* Base Address para el COM indicado en "com_id" */
   BiosAddr=(0x00000400+2*(ComId-1));
   /* Deteccion de la direccion base del COM indicado en com_id */
   *AddrCom=*(unsigned int far *)BiosAddr;
   if(*AddrCom==0)   /* COM no detectado, acabar */
      return(F_ERROR_COM_NO_DETECTADO);

   /* COM correcto y detectado sin problemas */
   return(F_NO_ERROR);
}


/* ********************************************************************** */
/*                            F_Open                                      */
/* ********************************************************************** */

int F_Open(unsigned char ComId,unsigned int DivVel)
{  unsigned int AddrCom;
   unsigned char Mask;
   int aux;

   if(F_Init==1)                               /* Si nivel fisico ya abierto (inicializado) */
      return(F_ERROR_NO_CLOSE);                /* Error primero hay que cerrarlo */

   /* Deteccion del COM indicado en ComId y su direccion base */
   if((aux=DetectCOM(ComId,&AddrCom))!=F_NO_ERROR)  /* Si error */
      return(aux);                             /* Devolvemos el tipo de error sucedido */

   /* Inicializacion de direcciones de los registros de linea serie */
   RBR=AddrCom;                                /* Receive Buffer Register (DLAB=0) */
   THR=AddrCom;                                /* Transmitter Holding Register (DLAB=0) */
   DLLB=AddrCom;                               /* Divisor Latch Low Byte (DLAB=1) */
   DLHB=AddrCom+1;                             /* Divisor Latch High Byte (DLAB=1) */
   IER=AddrCom+1;                              /* Interrupt Enable Register */
   IIR=AddrCom+2;                              /* Interrupt Identification Register */
   FCR=AddrCom+2;                    /* FIFO Control Register, solo 16550 y superiores */
   LCR=AddrCom+3;                              /* Line Control Register */
   MCR=AddrCom+4;                              /* Modem Control Register */
   LSR=AddrCom+5;                              /* Line Status Register */
   MSR=AddrCom+6;                              /* Modem Status Register */

   /* Salvamos el estado de la linea serie y la inicializamos */
   disable();   /* Inhibimos las interrupciones */
```

```
   EstadoFisico.lcr=inportb(LCR);           /* Salvamos LCR */
   outportb(LCR,EstadoFisico.lcr|0x80);     /* DLAB=1 en el LCR, acceso al DL */
   EstadoFisico.dllb=inportb(DLLB);         /* Salvamos DLLB */
   EstadoFisico.dlhb=inportb(DLHB);         /* Salvamos DLHB */
   outportb(DLLB,DivVel&0x00FF);
   outportb(DLHB,DivVel>>8);                /* Inicializamos la velocidad */
   outportb(LCR,0x03);                     /* DLAB=0,no parity,1 stop,8 data */
/* En nuestro caso no salvamos MCR ya que luego no lo restauraremos, en
   otro caso habria que salvarlo para dejar la linea serie en su estado
   inicial */
//   EstadoFisico.mcr=inportb(MCR);         /* Salvamos MCR */
   EstadoFisico.ier=inportb(IER);           /* Salvamos IER */
   outportb(IER,0x07);                      /* Enables RLSI,THREI,RDAI */
   outportb(MCR,0x0B);                      /* RTS=1 (modo run),DTR=1 (no reset) */
   /* Los bits de control de modem RTS y DTR controlan el modo de
      funcionamiento del "Genghis II" y el reset respectivamente */
   outportb(FCR,0x07);                      /* Buffers activados (si hay), tama¤o 1 byte */
   if(ComId%2==0)                           /* Interrupcion para COM2 o COM4, IRQ3 */
   {   EstadoFisico.NInt=0x0B;              /* Direccion de IRQ3 en vector interrup. */
      Mask=0xF7;                            /* Mascara para permitir interrupcion IRQ3 */
   }
   else                                     /* Interrupcion para COM1 o COM3, IRQ4 */
   {  EstadoFisico.NInt=0x0C;               /* Direccion de IRQ4 en vector interrup. */
      Mask=0xEF;                            /* Mascara para permitir interrupcion IRQ4 */
   }

   /* Salvamos el vector de interrupciones, inicializamos con nuestra
      interrupcion y activamos las interrupciones del puerto serie para el
      COM indicado */
   EstadoFisico.OldIntLs=getvect(EstadoFisico.NInt);
   setvect(EstadoFisico.NInt,IntLs);
   EstadoFisico.imr=inportb(0x21);          /* Salvamos IMR del 8259 (mascara int.) */
   outportb(0x21,EstadoFisico.imr&Mask);    /* Enable interrupcion linea serie */

   enable();                                /* Habilitamos las interrupciones */

   /* Eliminamos las interrupciones pendientes */
   inportb(MSR);                            /* Limpia estado del Modem Status Register */
   inportb(LSR);                            /* Limpia estado del Line Status Register */
   inportb(IIR);             /* Limpia estado del Interrupt Indentification Register */
   inportb(RBR);                            /* Limpia el registro RBR */
   inportb(RBR);                     /* Limpia posible valor que viniera del RSR al RBR */

   /* Inicializamos el estado del nivel fisico */
   EstadoFisico.ErrorOverrun=0;
   EstadoFisico.ErrorParidad=0;
   EstadoFisico.ErrorTrama=0;
   EstadoFisico.BreakLine=0;
   F_Init=1;   /* Indicamos que nivel fisico inicializado */

   /* Inicializamos los buffers de nivel fisico */
   IniBuffer(&BufferIn);
   IniBuffer(&BufferOut);

   return(F_NO_ERROR);
}


/* ********************************************************************** */
/*                              F_Close                                   */
/* ********************************************************************** */

int F_Close(void)
{  if(F_Init==0)                            /* Si nivel fisico no abierto (no inicializado) */
      return(F_ERROR_NO_OPEN);              /* Error primero hay que abrirlo */

   /* Restauramos el estado de la linea serie */
   disable();                               /* Inhibimos las interrupciones */
   outportb(LCR,0x80);                      /* DLAB=1 (acceso al DL),no importa resto */
   outportb(DLLB,EstadoFisico.dllb);        /* Restauramos DLLB */
   outportb(DLHB,EstadoFisico.dlhb);        /* Restauramos DLHB */
   outportb(LCR,0x7F);                      /* DLAB=0, no importa el resto */
   /* No restauramos valor anterior del MCR para mantener el valor actual
      que mantiene al robot en modo RUN y no reset */
//   outportb(MCR,EstadoFisico.mcr);        /* Restauramos MCR */
   outportb(IER,EstadoFisico.ier);          /* Restauramos IER */
   outportb(LCR,EstadoFisico.lcr);          /* Restauramos LCR */
   outportb(0x21,EstadoFisico.imr);         /* Restaurar IMR 8259, mascara int. */

   /* Restauramos el vector de interrupciones */
   setvect(EstadoFisico.NInt,EstadoFisico.OldIntLs);
   enable();                                /* Habilitamos las interrupciones */
```

```
   return(F_NO_ERROR);
}


/* ********************************************************************** */
/*                              F_State                                   */
/* ********************************************************************** */

int F_State(unsigned int param,unsigned long *valor)
{  if(F_Init==0)                    /* Si nivel fisico no abierto (no inicializado ) */
      return(F_ERROR_NO_OPEN);             /* Error primero hay que abrirlo */

   if((param&0xF000)==0x0000)              /* Si el parametro es de nivel fisico */
   {  switch(param)
      {   case F_ERROR_OVERRUN:
            *valor=EstadoFisico.ErrorOverrun;
            break;

         case F_ERROR_PARIDAD:
            *valor=EstadoFisico.ErrorParidad;
            break;

         case F_ERROR_TRAMA:
            *valor=EstadoFisico.ErrorTrama;
            break;

         case F_ERROR_BREAKLINE:
            *valor=EstadoFisico.BreakLine;
            break;

         case F_ERROR_OVERRUN_BUFFER:
            *valor=EstadoFisico.ErrorOverrunBuffer;
            break;

         default:
            return(F_ERROR_PARAM_DESCONOCIDO);   /* Parametro desconocido */
      }
      return(F_NO_ERROR);                  /* Lectura de estado correcta */
   }
   /* Si el parametro no es de nivel fisico se ignora ya que no hay niveles
      inferiores y indicamos que el parametro es desconocido */
   return(F_ERROR_PARAM_DESCONOCIDO);
}


/* ********************************************************************** */
/*                               F_Read                                   */
/* ********************************************************************** */

int F_Read(unsigned char *str,unsigned int *longitud)
{  if(F_Init==0)                    /* Si nivel fisico no abierto (no inicializado ) */
      return(F_ERROR_NO_OPEN);       /* Error primero hay que abrirlo */

   if(ReadBuffer(&BufferIn,str,longitud)==0)
      return(F_ERROR_ACCESO_BUFFER);
   else
      return(F_NO_ERROR);
}


/* ********************************************************************** */
/*                              F_Write                                   */
/* ********************************************************************** */

int F_Write(unsigned char *str,unsigned int longitud)
{  unsigned char c;
   int ValRet;

   if(F_Init==0)                       /* Si nivel fisico no abierto (no inicializado ) */
      return(F_ERROR_NO_OPEN);       /* Error primero hay que abrirlo */

   ValRet=WriteBuffer(&BufferOut,str,longitud);
   /* ValRet=1 si se ha podido escribir y 0 en caso contrario */

   /* Comprobamos si existe la cadena de interrupciones */
   disable();
   if((inportb(LSR)&0x60)==0x60)             /* No existe cadena de interrupciones */
   {  if(ReadCharBuffer(&BufferOut,&c))      /* Si hay caracteres en el buffer */
         outportb(THR,c);                    /* Iniciamos la cadena de interrupciones */
   }
   enable();
```

```
   if(ValRet==0)
      return(F_ERROR_ACCESO_BUFFER);
   else
      return(F_NO_ERROR);
}


/* ********************************************************************** */
/*                                F_Env                                   */
/* ********************************************************************** */

int F_Env(int *env)
{   if(F_Init==0)                    /* Si nivel fisico no abierto (no inicializado ) */
      return(F_ERROR_NO_OPEN);              /* Error primero hay que abrirlo */
   /* Comprobamos si hay cadena de interrupciones, si la hay env=1 (se
      esta enviando) y en caso contrario env=0 (no se esta enviando) */
   *env=!((inportb(LSR)&0x60)==0x60);       /* No existe cadena de interrupciones */
      return(F_NO_ERROR);
}


/* ********************************************************************** */
/*                                F_ResetRob                              */
/* ********************************************************************** */

int F_ResetRob(void)
{  if(F_Init==0)                     /* Si nivel fisico no abierto (no inicializado ) */
      return(F_ERROR_NO_OPEN);       /* Error primero hay que abrirlo */

   outportb(MCR,0x0A);               /* RTS=1 (modo run), DTR=0 (reset) */
   outportb(MCR,0x0B);               /* RTS=0 (modo run), DTR=1 (no reset) */

   return(F_NO_ERROR);
}
```

Fichero "enlace.h"

```
#ifndef _ENLACE
#define _ENLACE

#include "fisico.h"

/* Definicion de tipos de error */
#define E_NO_ERROR                 0        /* No hay error */
#define E_ERROR_NO_OPEN           -3        /* Nivel enlace no abierto */
#define E_ERROR_NO_CLOSE          -4        /* Nivel enlace no cerrado */
#define E_ERROR_PARAM_DESCONOCIDO -5        /* Parametro estado desconocido */
#define E_ERROR_OPEN_FISICO       -6        /* Error al abrir nivel fisico */

/* Parametros para consultar el estado del nivel de enlace. Todos los
   parametros para consultar el estado del nivel de enlace comienzan por
   1 en hexadecimal. Esto permite a niveles superiores detectar que el
   parametro es de nivel de enlace o al nivel de enlace detectar si
   es un parametro para el o para el nivel fisico inferior */
#define E_TRAMAS_DETECTADAS  0x1001  /* Contador SOH detectados */
#define E_TRAMAS_CORRECTAS   0x1002 /* Contador tramas correctas */
#define E_TRAMAS_ERRONEAS    0x1003 /* Contador tramas erroneas */
#define E_ERRORES_TIPO_TRAMA 0x1004 /* Contador errores por tipo trama */
#define E_ERRORES_CHECKSUM   0x1005 /* Contador errores checksum */
#define E_NUM_TOUT           0x1006 /* Contador numero timeouts */
#define E_NUM_TOUT_PET_DAT   0x1007 /* Contador timeouts peticion datos */


/* Definiciones para acceso a los datos de los sensores. Parametros de
   "ReadSensor". */
#define E_SENS_MA0 0x00      /* Motor alfa0 */
#define E_SENS_MA1 0x01      /* Motor alfa1 */
#define E_SENS_MA2 0x02      /* Motor alfa2 */
#define E_SENS_MA3 0x03      /* Motor alfa3 */
#define E_SENS_MA4 0x04      /* Motor alfa4 */
#define E_SENS_MA5 0x05      /* Motor alfa5 */
#define E_SENS_MB0 0x06      /* Motor beta0 */
#define E_SENS_MB1 0x07      /* Motor beta1 */
#define E_SENS_MB2 0x08      /* Motor beta2 */
#define E_SENS_MB3 0x09      /* Motor beta3 */
#define E_SENS_MB4 0x0A      /* Motor beta4 */
#define E_SENS_MB5 0x0B      /* Motor beta5 */
```

```
#define E_SENS_PT1 0x0C      /* Posicion tacto1 */
#define E_SENS_PT2 0x0D      /* Posicion tacto2 */
#define E_SENS_PT3 0x0E      /* Posicion tacto3 */
#define E_SENS_FT1 0x0F      /* Fuerza tacto1 */
#define E_SENS_FT2 0x10      /* Fuerza tacto2 */
#define E_SENS_FT3 0x11      /* Fuerza tacto3 */
#define E_SENS_IFR 0x12      /* Infrarrojos */
#define E_SENS_INC 0x13      /* Inclinometro */
#define E_SENS_PR1 0x14      /* Piro1 */
#define E_SENS_PR2 0x15      /* Piro2 */
#define E_SENS_PR3 0x16      /* Piro3 */
#define E_SENS_PR4 0x17      /* Piro4 */
#define E_SENS_PR5 0x18      /* Piro5 */
#define E_SENS_BTE 0x19      /* Bateria electronica */
#define E_SENS_BTM 0x1A      /* Bateria de motores */
#define E_SENS_WKR 0x1B      /* Whisker derecho */
#define E_SENS_WKL 0x1C      /* Whisker izquierdo */

/* Definiciones para acceso a los datos de las ordenes. Parametros de
   "WriteMotor" y "ReadMotor". */
#define E_ORD_MA0 0x00       /* Motor alfa0 */
#define E_ORD_MA1 0x01       /* Motor alfa1 */
#define E_ORD_MA2 0x02       /* Motor alfa2 */
#define E_ORD_MA3 0x03       /* Motor alfa3 */
#define E_ORD_MA4 0x04       /* Motor alfa4 */
#define E_ORD_MA5 0x05       /* Motor alfa5 */
#define E_ORD_MB0 0x06       /* Motor beta0 */
#define E_ORD_MB1 0x07       /* Motor beta1 */
#define E_ORD_MB2 0x08       /* Motor beta2 */
#define E_ORD_MB3 0x09       /* Motor beta3 */
#define E_ORD_MB4 0x0A       /* Motor beta4 */
#define E_ORD_MB5 0x0B       /* Motor beta5 */


/* Definiciones para acceso a que grupo de sensores se ha actualizado
   despues de una lectura de sensores. Parametros para "ActSensor". */
#define E_ACT_MTALFA 0x00           /* Motores alfa */
#define E_ACT_MTBETA 0x01           /* Motores beta */
#define E_ACT_PTACTO 0x02           /* Posiciones de tacto */
#define E_ACT_FTACTO 0x03           /* Fuerzas de tacto */
#define E_ACT_INFRAR 0x04           /* Infrarrojos */
#define E_ACT_INCLIN 0x05           /* Inclinometro */
#define E_ACT_PIR_23 0x06           /* Piros 2 y 3 */
#define E_ACT_PIR145 0x07           /* Piros 1, 4 y 5 */
#define E_ACT_BTELEC 0x08           /* Bateria electronica */
#define E_ACT_BTMOTR 0x09           /* Bateria de motores */
#define E_ACT_WHISKR 0x0A           /* Whiskers */


/* Definiciones para indicar o consultar que grupos de sensores se desean
   recibir. Los sensores de cara al usuario siempre se consideran agrupados
   en una variable de tipo entero de manera que cada bit corresponde a uno
   de los grupos de sensores desde el bit de menos peso (bit 0) hasta el
   bit numero 8, en total nueve grupos. Parametros para "ReadQueDatos" y
   "WriteQueDatos". */
#define E_ALL_DAT 0x01FF     /* Todos los grupos de sensores */
#define E_DAT_WSK 0x0001     /* Whiskers */
#define E_DAT_BTR 0x0002     /* Baterias: de motores y electronica */
#define E_DAT_PIR 0x0004     /* Piros */
#define E_DAT_INC 0x0008     /* Inclinometro */
#define E_DAT_INF 0x0010     /* Infrarrojos */
#define E_DAT_FTC 0x0020     /* Fuerzas de tacto */
#define E_DAT_PTC 0x0040     /* Posiciones de tacto */
#define E_DAT_MTB 0x0080     /* Motores beta */
#define E_DAT_MTA 0x0100     /* Motores alfa */


/* Tipos de errores o indicaciones sobre el estado de la comunicacion.
   Parametros para la funcion "E_EstCom" */
#define E_ALL_ERR_IND    0x007F     /* Consultar todos errores o indicaciones */
#define E_IND_ENV_CICLO  0x0001     /* Ordenes no enviadas dentro del ciclo */
#define E_ERR_ENV        0x0002     /* Error al enviar la trama de ordenes */
#define E_OVERR_SENSORES 0x0004     /* Varias tramas detectadas en lectura */
#define E_OVERR_ORDENES  0x0008     /* Nueva orden antes de enviar la anterior */
#define E_IND_NO_OPTIMO  0x0010     /* SOH detectados menor numero optimo */
#define E_ERR_TIMEOUT    0x0020     /* Timeout sin recibir nada */
#define E_ERR_PET_DAT  0x0040       /* Timeout sin recibir datos pedidos */



/* Macros de interficie para escribir ordenes a los actuadores en el buffer
   de usuario. En caso de que el valor que se quiere escribir sea correcto
```

```
   para cada tipo de sensor la macro evalua 1 y en caso de que el valor a
   escribir para el sensor sea incorrecto evalua 0 y no se escribe el valor.
   Si el valor es correcto se actualiza el flag indicando que hay nuevas
   ordenes. Estos datos no se envian al host remoto hasta que se ejecuta
   E_Write. */
/* Macro para escribir una orden a uno de los motores */
#define E_WriteMotor(motor,valor)\
((motor<E_ORD_MA0 || motor>E_ORD_MB5 || valor>63 ||\
valor<0 )?0:(valor==(Ordenes[motor]=valor))&&(HayOrdenes=1)==1)
/* Macro para escribir una orden a los leds */
#define E_WriteLeds(valor)\
((valor>255 || valor<0)?0:(valor==(Ordenes[12]=valor))&&(HayOrdenes=1)==1)
/* Macro para escribir una orden al buzzer */
#define E_WriteBuzzer(valor)\
((valor>255 || valor<0)?0:(valor==(Ordenes[13]=valor))&&(HayOrdenes=1)==1)


/* Macros de interficie para consultar el valor de las ordenes a los
   actuadores en el buffer de usuario */
/* Macro para consultar el valor de la orden de uno de los motores. Si el
   parametro es correcto devuelve el valor del motor correspondiente y si
   no devuelve 255 (valor imposible para un motor). */
#define E_ReadMotor(motor)\
((motor>=E_ORD_MA0 && motor<=E_ORD_MB5)?Ordenes[motor]:255)
/* Macro para consultar el valor de la orden de los leds */
#define E_ReadLeds() (Ordenes[12])
/* Macro para consultar el valor de la orden del buzzer */
#define E_ReadBuzzer() (Ordenes[13])


/* Macro de interficie para consultar el valor de los sensores en el buffer
   de usuario. Si el parametro es correcto retorna el valor del sensor
   correspondiente, si no retorna 255 (valor imposible para un sensor). */
#define E_ReadSensor(tipo)\
((tipo>=E_SENS_MA0 && tipo<=E_SENS_WKL)?Sensores[tipo]:255)


/* Macro de interficie para consultar si un grupo de sensores ha sido
   actualizado tras la ultima lectura de sensores. Si el parametro es
   correcto retorna el valor de si el grupo de sensores ha sido actualizado
   (1) o no (0) y en caso contrario retorna 255 (valor imposible). */
#define E_ActSensor(grupo)\
((grupo>=E_ACT_MTALFA && grupo<=E_ACT_WHISKR)?SensAct[grupo]:255)


/* Funcion que inicializa el nivel de enlace. No tiene parametros. Si todo ha
   ido bien retorna E_NO_ERROR y en caso contrario retorna el error
   correspondiente. */
int E_Open(void);


/* Funcion que cierra el nivel de enlace. Si todo ha ido bien retorna
   E_NO_ERROR y en caso contrario retorna el error correspondiente. */
int E_Close(void);


/* Funcion con la que se puede indicar al nivel de enlace la llegada de un
   nuevo caracter para coordinar el envio de ordenes (niveles inferiores).
   El nivel de enlace consulta dicho caracter que se le pasa como parametro y
   en caso de detectar un inicio de trama inicia el envio de ordenes al nivel
   fisico. No retorna parametros pero en caso de que el nivel de enlace no
   este inicializado no hace nada. */
void E_CaracterIndication(unsigned char c);


/* Funcion que indica que se pueden enviar los datos escritos en los buffers
   de usuario "Ordenes" con ordenes a los motores y "QueDatos" donde se
   indica que datos de que sensores se quieren recibir. La funcion copia
   los datos en el buffer "TramaOrd" calculando la trama, que sera enviada
   en el momento en que se reciba una nuevo inicio de trama procedente del
   host remoto. En caso de que todo vaya bien retorna E_NO_ERROR y en caso
   contrario retorna el error correspondiente. */
int E_Write(void);


/* Funcion que permite obtener el valor de los sensores de la ultima trama
   correcta recibida en el vector "Sensores". En caso de que todo vaya bien
   retorna E_NO_ERROR y si hay algun problema retorna el error
   correspondiente. Si todo va bien en "TramaLeida" se pone un 1 en caso de
   disponer de informacion de una nueva trama o 0 en caso contrario. En el
   vector "SensAct" se indica con un 1 los grupos de sensores actualizados y
   con un 0 los que no han sido actualizados con la lectura de la trama. En
```

```
   caso de que hubiera mas de una trama acumulada en al buffer desde la
   anterior lectura se activa la indicacion IND_REC_CICLO. */
int E_Read(int *TramaLeida);


/* Funcion que sirve de interficie para consultar los parametros del
   nivel de enlace o del nivel fisico, principalmente contadores de errores
   o tramas. Se le pasa el parametro de nivel de enlace o fisico
   correspondiente a la variable de estado que se quiere consultar y un
   puntero a la variable de tipo long donde se quiere obtener el valor.
   Si el parametro es de nivel fisico (inferior) se realiza una llamada
   a F_State con el mismo parametro. En caso de que el nivel de enlace no
   este inicializado retorna E_NO_OPEN y si el parametro no corresponde a
   nivel de enlace ni fisico se retorna E_ERROR_PARAM_DESCONOCIDO. Si todo
   va bien retorna F_NO_ERROR. */
int E_State(unsigned int param,unsigned long *valor);


/* Funcion de interficie para consultar el valor del indicador o error sobre
   el estado de la comunicacion que se le pide. Si todo va bien retorna
   E_NO_ERROR y en caso contrario retorna el error correspondiente. En "tipo"
   se le indica si se quiere consultar todos los indicadores del estado de la
   comunicacion o solo uno concreto. Si todo va bien, en "valor" se indica si
   el error o indicador esta activo (1) o si no (0) en el caso de un tipo
   concreto o se indica un valor entero de manera que cada uno de los bits
   corresponde a uno de los posibles errores o indicaciones, de manera que se
   pueden consultar con las mascaras que los identifican y que sirven como
   parametro "tipo". */
int E_EstCom(int tipo,int *valor);


/* Funcion que permite escribir en el buffer de usuario "QueDatos" para
   indicar la informacion de que sensores se quiere recibir del host
   remoto. Se puede pedir un grupo concreto o todos a la vez con las
   mascaras definidas anteriormente. Esta informacion solo se envia al
   host remoto cuando se ejecuta E_Write. Si todo va bien retorna E_NO_ERROR
   y en caso contrario retorna el error correspondiente. */
int E_WriteQueDatos(int GrpSens,int valor);

/* Funcion que permite consultar del buffer de usuario "QueDatos" que
   sensores se habia indicado recibir hasta el momento o si un grupo
   concreto de sensores se habia pedido o no anteriormente. Si todo va
   bien retorna E_NO_ERROR y en caso contrario retorna el error
   correspondiente. */
int E_ReadQueDatos(int GrpSens,int *valor);

/* Funcion inicializa el buffer de ordenes a su estado inicial (posiciones
   intermedias para los motores y cero para leds y buzzer) y hace un reset
   por software del robot */
int E_ResetRob(void);

/* ******************************  PRIVADO ****************************** */
/* Variables privadas de nivel de enlace, solo accesibles por el usuario
   a traves de las macros o funciones de interficie. Nunca se deberian
   acceder directamente. Aqui solo para que sean visibles al usuario
   del nivel de enlace. */

/* Definicion de tama¤os para los buffers accesibles por el usuario */
#define TAM_SENSORES 29
#define TAM_SENSACT 11
#define TAM_ORDENES 14
#define TAM_QUEDATOS 2

/* Buffers visibles por el usuario del nivel de Enlace con Ordenes,
   Tipo de datos que se quieren recibir, valor de Sensores y indicador de
   que sensores se han actualizado tras una lectura */
extern unsigned char Ordenes[TAM_ORDENES];  /* 12 motores + led + buzzer */
extern int QueDatos;  /* Solo se utilizan 9 bits */
extern unsigned char Sensores[TAM_SENSORES];/* 28 sensores diferentes */
extern unsigned char SensAct[TAM_SENSACT];  /* 11 grupos sensores */
extern int HayOrdenes;
extern int HayQueDatos;

/* ******************************  PRIVADO ****************************** */

#endif
```

Fichero "enlace.c"

```
#include <io.h>
#include <fcntl.h>
#include <sys/stat.h>
#include "enlace.h"
#include "clock.h"

/* Tamaños de vectores de nivel de enlace */
#define TAM_TRAMAORD 18
#define TAM_TRAMASENS 18
/* El resto en el fichero de cabecera */

/* Cabecera de la trama */
#define SOH 0xFF

/* Estados de deteccion de una trama */
#define DETECTA_CABECERA 0x01
#define LEE_BYTE_DATOS1 0x02
#define LEE_BYTE_DATOS2 0x03
#define LEE_LONGIT_TRAMA 0x04

/* Valor de Timeouts */
/* Respuesta a una peticion de datos como maximo despues de cinco tramas
   detectadas correctas, si no error */
#define E_MAX_TIMEOUT_PET_DAT 0x05
/* Cada interrupcion de reloj deberiamos haber detectado dos inicios trama */
#define E_MIN_SOH_CICLO_TIMEOUT 0x02

/* Macros para actualizar posibles errores o indicaciones de comunicacion */
#define ActivaEstCom(tipo) EstadoEnlace.EstCom|=tipo
#define DesactivaEstCom(tipo) EstadoEnlace.EstCom&=(~tipo)


/* Buffer con la trama de ordenes a enviar */
unsigned char TramaOrd[TAM_TRAMAORD];              /* 14 datos + SOH + checksum */
/* Buffer de recepcion de la ultima trama correcta */
unsigned char TramaSens[TAM_TRAMASENS];     /* 17 datos + checksum (sin SOH) */
/* Buffer temporal de recepcion de tramas */
unsigned char TramaSensTmp[TAM_TRAMASENS];/* 17 datos + checksum (sin SOH) */

/* Buffers de interficie con el usuario del nivel de enlace */
/* Buffer con los datos de los sensores recibidos hasta el momento */
unsigned char Sensores[TAM_SENSORES];       /* 28 sensores diferentes */
/* Buffer indica que sensores son actualizados con ultima trama recibida */
unsigned char SensAct[TAM_SENSACT];         /* 11 grupos diferentes de sensores */
/* Buffer de ordenes que quiere enviar el usuario */
unsigned char Ordenes[TAM_ORDENES];  /* 12 motores + led + buzzer */
int QueDatos;  /* Se utilizan 9 bits */

/* Variables que indican la existencia o no de nuevas ordenes */
int HayTramaOrd=0;    /* Indica si nueva trama ordenes en "TramaOrd" */
int OrdenesATramaOrd=0;       /* Indica si pasando ordenes a "TramaOrd" */
int HayOrdenes=0;     /* Indica si nuevas ordenes en "Ordenes" */
int HayQueDatos=0;    /* Indica si nueva peticion de datos en "QueDatos" */

/* Situacion de la maquina de estados para recibir tramas */
unsigned char EstadoRecep=DETECTA_CABECERA;
/* Indicador de si hay trama o no en estado de recepcion */
unsigned char TramaEnRecep=0;
/* Offset para guardar la trama en recepcion en TramaTmp */
int offset;
/* Longitud que queda por leer de la trama en estado de recepcion */
unsigned char LonTramaRecep;
/* Tipo de trama en estado de recepcion */
unsigned char TipoTramaRecep;

/* Timeout por peticion de datos */
int TimeoutPetDat=0;
/* Timeouts por recepcion de caracteres y SOH */
int NumSOHTimeout=0;  /* Para controlar si protocolo funciona modo optimo */
int RecepTimeout=0;   /* Para controlar si host remoto no existe */

/* Estado del nivel de Enlace */
struct
{  unsigned long TramasDetectadas;   /* Contador de SOH detectados */
   unsigned long TramasCorrectas;    /* Contador de tramas correctas */
   unsigned long TramasErroneas;     /* Contador de tramas erroneas */
   unsigned long ErrorTipoTrama;     /* Contador errores por tipo de trama */
   unsigned long ErrorChecksum;      /* Contador errores por checksum */
   unsigned long NumToutPetDat;      /* Contador timeouts peticion datos */
```

```
   unsigned long NumTout;            /* Contador timeouts (no recepcion) */
   int EstCom;                       /* Estado comunicacion: indicaciones o errores */
} EstadoEnlace;

/* Variable para indicar si el nivel de enlace esta inicializado o no */
int E_Init=0;


/* ********************************************************************** */
/*                              IntClk                                    */
/* ********************************************************************** */
/* Interrupcion de clock. Se genera una interrupcion cada 18.2 segundos. Se
   utiliza para detectar si no se recibe ningun dato del host remoto
   (timeout) o si el numero de tramas que se reciben es menor al numero
   optimo esperado */

void interrupt Timeouts(void)
{  if(NumSOHTimeout<E_MIN_SOH_CICLO_TIMEOUT)
   {  /* No se ha detectado el numero minimo de inicios de trama, llegan
      menos de los esperados para un funcionamiento optimo */
      ActivaEstCom(E_IND_NO_OPTIMO);
   }
   else
      DesactivaEstCom(E_IND_NO_OPTIMO);
   NumSOHTimeout=0;

   if(RecepTimeout==0)
   {  /* No se ha recibido ningun caracter, host remoto no esta presente */
      ActivaEstCom(E_ERR_TIMEOUT);
      EstadoEnlace.NumTout++;
   }
   else
      DesactivaEstCom(E_ERR_TIMEOUT);
   RecepTimeout=0;
   /* No necesita EOI */
}


/* ********************************************************************** */
/*                               E_Open                                   */
/* ********************************************************************** */
int E_Open(void)
{  int i,fd,c=0,aux;                 /* c=0 por que al leer solo se actualiza byte bajo */

   if(E_Init==1)                     /* Si nivel enlace ya abierto (inicializado) */
      return(E_ERROR_NO_CLOSE);      /* Error primero hay que cerrarlo */

   /* Inicializacion de nivel fisico */
   if((fd=open("prtcgeng.cfg",O_RDONLY))<0)  /* Si no existe fichero configuracion */
   {  if(F_Open(1,0x000B)!=F_NO_ERROR)       /* COM1 por defecto */
         return(E_ERROR_OPEN_FISICO);        /* Si no disponible error */
   }
   else                              /* Comprobamos el valor del fichero de configuracion */
   {  if((aux=read(fd,&c,1))>0)      /* Leemos valor del fichero de config. */
      {  close(fd);                  /* Cerramos el fichero de configuracion */
         switch(c)
         {  case '1':
               aux=F_Open(c-'0',0x000B);
               break;
            case '2':
               aux=F_Open(c-'0',0x000B);
               break;
            case '3':
               aux=F_Open(c-'0',0x000B);
               break;
            case '4':
               aux=F_Open(c-'0',0x000B);
               break;
            default:
               return(E_ERROR_OPEN_FISICO);
         }
         if(aux!=F_NO_ERROR)         /* Error inicializacion nivel fisico */
            return(E_ERROR_OPEN_FISICO);
      }
      else                           /* Si error de lectura o vacio, acabamos */
      {   close(fd);                 /* Cerramos el fichero de configuracion */
         return(E_ERROR_OPEN_FISICO);       /* Error abriendo el nivel fisico */
      }
   }

   /* Ningun sensor actualizado */
   for(i=0;i<TAM_SENSACT;i++)
```

```
      SensAct[i]=0;

   /* Ninguna orden ni trama de ordenes */
   HayTramaOrd=0;
   OrdenesATramaOrd=0;
   HayOrdenes=0;
   HayQueDatos=0;

   /* Estado inicial de Enlace y de la comunicacion */
   EstadoEnlace.TramasDetectadas=0;
   EstadoEnlace.TramasCorrectas=0;
   EstadoEnlace.TramasErroneas=0;
   EstadoEnlace.ErrorTipoTrama=0;
   EstadoEnlace.ErrorChecksum=0;
   EstadoEnlace.NumToutPetDat=0;
   EstadoEnlace.NumTout=0;
   EstadoEnlace.EstCom=0;
   E_Init=1;                        /* Indicamos que nivel de enlace inicializado */

   /* Estado inicial del buffer de ordenes de usuario */
   for(i=0;i<TAM_ORDENES-2;i++)
      Ordenes[i]=30;  /* Ordenes iniciales a motores: 30, pos. intermedia */
      Ordenes[i]=0;           /* Orden inicial a los leds: 0, apagados */
      Ordenes[i+1]=0;         /* Orden inicial al buzzer: 0, desactivado */

   /* Estado inicial del buffer de sensores de usuario */
      for(i=0;i<TAM_SENSORES;i++)
         Sensores[i]=0;

   /* Estado inicial de los datos que se esperan del host remoto */
      TramaOrd[0]=(unsigned char)SOH;
      TramaOrd[1]=0;          /* Ningun dato de motores ni tacto */
      TramaOrd[2]=0;          /* Ningun dato infrarr,inclinom,piros,baterias y wiskers */
      QueDatos=0x0000;        /* Ningun dato */

       /* Inicializacion de timeouts */
       TimeoutPetDat=0;
       NumSOHTimeout=0;
       RecepTimeout=0;

       /* Inicializacion rutina para timeouts */
       OpenClk(Timeouts);

       return(F_NO_ERROR);
}


/* ********************************************************************** */
/*                              E_Close                                   */
/* ********************************************************************** */
int E_Close(void)
{  if(E_Init==0)                    /* Si nivel enlace no abierto (no inicializado) */
      return(E_ERROR_NO_OPEN);      /* Error primero hay que abrirlo */

   /* Restauramos (cerramos) interrupcion de clock */
   CloseClk();
   /* Cerramos nivel fisico, seguro no error */
   F_Close();

   return(E_NO_ERROR);
}


/* ********************************************************************** */
/*                         E_CaracterIndication                           */
/* ********************************************************************** */
void E_CaracterIndication(unsigned char c)
{  int Env;

   if(E_Init==0)                    /* Si nivel enlace no abierto (no inicializado) */
      return;                       /* no hacemos nada */

   RecepTimeout++;                  /* Nuevo caracter recibido, nunca puede desbordar */
   if(c==(unsigned char)SOH)        /* Si es inicio de trama intentamos enviar ordenes si hay
*/
   {  NumSOHTimeout++;              /* Inc. contador SOH timeout, no puede desbordar */
      /* Comprobamos si estan pasando nuevas ordenes a la trama de ordenes */
      if(OrdenesATramaOrd)
         ActivaEstCom(E_IND_ENV_CICLO);
      else
      {  DesactivaEstCom(E_IND_ENV_CICLO);
         /* Comprobamos que ya no se esta enviando en el nivel fisico, si
```

```
            se esta enviando la cabecera es erronea o la linea serie no
            va suficientemente rapido y no hay que enviar para evitar un
            colapso. */
         F_Env(&Env);
         if(Env==1)
         {  ActivaEstCom(E_ERR_ENV);               /* Error enviando ordenes */
            /* Se intentaran enviar en proximo ciclo si nadie pasa nuevas
               ordenes al buffer de ordenes */
            return;
         }

         if(HayTramaOrd>0)                         /* Si hay nueva trama de ordenes enviamos
*/
         {  if(F_Write(TramaOrd,HayTramaOrd)==F_NO_ERROR)
            {  DesactivaEstCom(E_ERR_ENV);         /* No error enviando ordenes */
               HayTramaOrd=0;                      /* No nuevas ordenes */
            }
            else
               ActivaEstCom(E_ERR_ENV);            /* Indicamos error enviando ordenes */
               /* Se intentaran enviar en proximo ciclo si nadie pasa nuevas
                  ordenes al buffer de ordenes */
         }
         else                                 /* Enviamos peticion de ordenes de trama anterior
*/
         {  TramaOrd[1]=(TramaOrd[1]&0x0F);         /* Indicamos trama tipo 0 */
            TramaOrd[2]=(TramaOrd[2]&0xF8);
            TramaOrd[3]=TramaOrd[1];                /* Calculo del checksum */
            TramaOrd[3]+=TramaOrd[2];               /* Calculo del checksum */
            if(TramaOrd[3]==(unsigned char)SOH)     /* Si checksum==255 */
               TramaOrd[3]&=0xFE;                   /* cortamos a 254 */
               if(F_Write(TramaOrd,4)==F_NO_ERROR)
                  DesactivaEstCom(E_ERR_ENV);       /* No error enviando ordenes */
               else
                  ActivaEstCom(E_ERR_ENV);          /* Error enviando ordenes */
         }
      }
   }
}


/* ********************************************************************** */
/*                              E_Write                                   */
/* ********************************************************************** */
int E_Write(void)
{  unsigned char Checksum=0;
      int i=0,j;

   if(E_Init==0)      /* Si nivel enlace no abierto (no inicializado) */
      return(E_ERROR_NO_OPEN);       /* Error primero hay que abrirlo */

   if(HayOrdenes || HayQueDatos)
   {  OrdenesATramaOrd=1;                           /* Pasando datos a "TramaOrd" */

      if(HayTramaOrd!=0)      /* Trama anterior no enviada todavia */
         ActivaEstCom(E_OVERR_ORDENES);
      else
         DesactivaEstCom(E_OVERR_ORDENES);

      TramaOrd[0]=(unsigned char)SOH;               /* Inicializamos la cabecera */
      TramaOrd[1]=((QueDatos>>5)&0x000F);           /* Primer byte tipo datos */
      TramaOrd[2]=((QueDatos<<3)&0x00F8);           /* Segundo byte tipo datos */
      if(HayOrdenes)
      {  TramaOrd[1]|=0x10;                         /* Indicamos trama tipo 1 */
         for(i=0;i<TAM_ORDENES;i++)                 /* Copiamos ordenes a los motores */
            TramaOrd[i+3]=Ordenes[i];               /* a su lugar en "TramaOrd" */
         i+=3;                                      /* Apuntamos a ultimo byte trama */
         TramaOrd[2]|=((TramaOrd[i-2]&0x01)<<1);    /* LSB leds en 3§ byte */
         TramaOrd[i-2]&=0xFE;                       /* LSB leds a cero */
         TramaOrd[2]|=(TramaOrd[i-1]&0x01);         /* LSB buzzer en 3§ byte */
         TramaOrd[i-1]&=0xFE;                       /* LSB buzzer a cero */
         HayOrdenes=0;                              /* No hay nuevas ordenes */
         HayQueDatos=0;
      }
      else
      {  /* Ya queda indicado que es trama de tipo 0 */
         HayQueDatos=0;                             /* No se piden nuevos datos */
         i=3;                                       /* Apuntamos a ultimo byte trama */
      }

      for(j=1;j<i;j++)                              /* Calculo de checksum para todos los */
         Checksum+=TramaOrd[j];                     /* bytes excepto cabecera y ultimo */
      if(Checksum==(unsigned char)SOH)              /* Si checksum==cabecera (255) se */
```

```
         Checksum&=0xFE;                              /* corta ultimo bit para obtener 254 */
      TramaOrd[i]=Checksum;                           /* Checksum en ultima posicion trama */
      i++;                                            /* Longitud trama a enviar */

      OrdenesATramaOrd=0;                             /* Fin de pasar datos a "TramaOrd" */
      HayTramaOrd=i;                                  /* Nueva trama a enviar y longitud */
   }

   return(E_NO_ERROR);
}


/* ********************************************************************** */
/*                        LeeUltimaTramaCorrecta                          */
/* ********************************************************************** */
/* Funcion que trata todos los caracteres recibidos hasta el momento en el
   nivel fisico y devuelve en el buffer "Trama" la ultima trama correcta
   detectada. Cuando no hay mas caracteres en el nivel fisico se retorna
   quedando en un estado o otro dependiendo de los caracteres tratados.
   Primero se detecta el inicio de trama, a continuacion se identifica el
   tipo de trama con el siguiente byte y por ultimo se leen el resto de
   bytes de la trama segun el tipo de datos que contiene y que se deducen
   a partir de los dos bytes siguientes al SOH. Siempre que se detecta un
   inicio de trama (SOH) se comienza desde el principio. Si la trama
   detectada es correcta se envia al buffer definitivo "Trama" como ultima
   trama correcta detectada y en caso contrario (error checksum o tipo de
   trama) se ignora. Si se ha detectado alguna trama correcta retorna 1 y en
   caso contrario retorna 0. En "Trama" se retorna toda la trama excepto la
   cabecera de manera que el primer byte corresponde al siguiente al SOH. */

int LeeUltimaTramaCorrecta(void)
{  unsigned char c,checksum,ptact,ftact,infr;
   unsigned int lon;
   int TramaLeida=0,i;
   char Fin=0;

   while(!Fin)                                 /* Mientras haya caracteres en el nivel fisico */
   {  switch(EstadoRecep)
      {  case DETECTA_CABECERA:                          /* Estado de deteccion de inicio de trama
*/
            lon=1;                                       /* Indicamos que queremos leer un byte */
            while((Fin=F_Read(&c,&lon))==0)              /* Leemos caracter nivel fisico */
            {  if(c==(unsigned char)SOH)                       /* Si inicio de trama */
               {  EstadoRecep=LEE_BYTE_DATOS1;           /* cambio de estado */
                  TramaEnRecep=1;                        /* Indicamos hay trama en recepcion */
                  offset=0;
                  EstadoEnlace.TramasDetectadas++;
                  break;                                 /* Salimos del bucle */
               }
               else
                  lon=1;
            }
            break;

         case LEE_BYTE_DATOS1:                           /* Estado de indentificacion tipo trama */
            lon=1;                                       /* Indicamos que queremos leer un byte */
            if((Fin=F_Read(&c,&lon))==0)                 /* Leemos caracter nivel fisico */
            {  if(c==(unsigned char)SOH)
               {  EstadoEnlace.TramasDetectadas++;       /* Inicio trama detectado */
                  offset=0;
                  break;                                 /* Fin del estado, continuamos en el mismo
*/
               }
               TramaSensTmp[offset]=c;                   /* Guardamos primer byte trama */
               c>>=4;                  /* Desplazamos para consultar el tipo de trama */
               if(c!=0 && c!=1)                                /* Si tipo de trama no 0 ni 1 */
               {  EstadoEnlace.ErrorTipoTrama++;         /* Tipo trama incorrecto */
                  EstadoEnlace.TramasErroneas++;         /* Error de trama */
                  EstadoRecep=DETECTA_CABECERA;                /* Ignoramos la trama */
                  TramaEnRecep=0;                        /* Indicamos no hay trama en recepcion */
               }
               else
               {  TipoTramaRecep=c;                      /* Guardamos el tipo de trama */
                  offset++;
                  EstadoRecep=LEE_BYTE_DATOS2;
               }
            }
            break;

          case LEE_BYTE_DATOS2:                          /* Estado de indentificacion tipo trama */
            lon=1;                                       /* Indicamos que queremos leer un byte */
            if((Fin=F_Read(&c,&lon))==0)                 /* Leemos caracter nivel fisico */
```

```
            {  if(c==(unsigned char)SOH)
               {  EstadoRecep=LEE_BYTE_DATOS1;      /* Inicio trama detectado */
                  offset=0;
                  EstadoEnlace.TramasDetectadas++;
                  break;                    /* Fin del estado, esperamos 1§ byte nueva trama
*/
               }
               TramaSensTmp[offset]=c;              /* Guardamos segundo byte trama */
               c>>=4;
               LonTramaRecep=(c&0x01);
               c>>=1;
               LonTramaRecep+=((c&0x01)*(2+TipoTramaRecep));
               c>>=1;
               LonTramaRecep+=(c&0x01);
               infr=c>>1;                           /* Indicador de si hay byte infrarrojos */
               c=TramaSensTmp[offset-1];            /* Obtenemos byte anterior */
               ftact=(c&0x01);
               c>>=1;
               ptact=(c&0x01);
               c>>=1;
               LonTramaRecep+=(c&0x01)*3;
               c>>=1;
               LonTramaRecep+=(c&0x01)*3;
               if(ptact==1)
                  LonTramaRecep+=3;
               if(ftact==1)
               {  if(ptact==1)
                     LonTramaRecep++;
                  else
                     LonTramaRecep+=3;
               }
               if((infr==1)&&(ptact==0)&&(ftact==0))
                  LonTramaRecep++;
               LonTramaRecep++;                     /* Sumamos longitud trama byte checksum */
               offset++;                            /* Incrementamos el offset */
               EstadoRecep=LEE_LONGIT_TRAMA;
            }
            break;

         case LEE_LONGIT_TRAMA:
            lon=1;                                  /* Indicamos que queremos leer un byte */
            while((Fin=F_Read(&c,&lon))==0)
            {  if(c==(unsigned char)SOH)
               {  EstadoRecep=LEE_BYTE_DATOS1;      /* Inicio trama detectado */
                  offset=0;
                  EstadoEnlace.TramasDetectadas++;
                  break;                    /* Salimos del bucle */
               }
               TramaSensTmp[offset]=c;              /* Guardamos el byte recibido */
               offset++;                            /* Incrementamos offset */
               LonTramaRecep--;                     /* Decrementamos longitud que falta */
               if(LonTramaRecep==0)         /* Si la trama esta completa */
                  break;                            /* Salimos del bucle */
            }
            if((!Fin)&&(c!=(unsigned char)SOH))     /* Si fin de bucle por trama completa */
            {  offset--;                            /* Decrementamos offset */
               checksum=0;                          /* inicializamos valor checksum a cero */
               for(i=0;i<offset;i++)        /* Calculamos checksum para todos */
                  checksum+=TramaSensTmp[i];        /* excepto el ultimo byte */
               if(checksum==(unsigned char)SOH)     /* Si checksum == 255 */
                  checksum&=0xFE;                   /* Eliminamos el bit menor peso */
               if(checksum==TramaSensTmp[offset])  /* Si checksum correcto */
               {  /* Comprobamos si es la primera trama del buffer o
                     si por el contrario habia mas de una en el mismo ciclo */
                  if(TramaLeida==1)
                     ActivaEstCom(E_OVERR_SENSORES);
                  else
                     DesactivaEstCom(E_OVERR_SENSORES);
                  TramaLeida=1;                     /* Indicamos que hay trama leida */
                  EstadoEnlace.TramasCorrectas++;
                  for(i=0;i<=offset;i++)            /* Pasamos trama al buffer de */
                     TramaSens[i]=TramaSensTmp[i];  /* ultima trama leida */
                  /* Comprobamos si los datos que vienen en la trama
                     corresponden con los esperados que se pidieron en la
                     ultima trama enviada */
                  if((TramaSens[0]&0x0F)!=(TramaOrd[1]&0x0F) ||
                    (TramaSens[1]&0xF8)!=(TramaOrd[2]&0xF8))
                  {  TimeoutPetDat++;
                     if(TimeoutPetDat>E_MAX_TIMEOUT_PET_DAT)
                     {  ActivaEstCom(E_ERR_PET_DAT);
                        EstadoEnlace.NumToutPetDat++;
                     }
```

```
                }
                else
                {  TimeoutPetDat=0;
                   DesactivaEstCom(E_ERR_PET_DAT);
                }
             }
             else
             {  EstadoEnlace.TramasErroneas++;
                EstadoEnlace.ErrorChecksum++;
             }
             EstadoRecep=DETECTA_CABECERA;        /* Estado detec. inicio trama */
             TramaEnRecep=0;                      /* Indicamos no hay trama en recepcion */
          }
          break;
      }
   }
   return(TramaLeida);
}


/* ********************************************************************** */
/*                          RecuperaDatosTrama                            */
/* ********************************************************************** */
/* Funcion que a partir de una trama sin cabecera cuyo puntero se le pasa
   como parametro, recupera la informacion de los sensores que contiene
   dicha trama y la deposita en el buffer de datos de sensores "Sensores" */

void RecuperaDatosTrama(unsigned char *Trama)
{  unsigned char TipoTrama;
   int offset=2;                                  /* Offset al primer byte con datos */

    TipoTrama=(Trama[0]>>4);                      /* Obtenemos el identificador de trama */

   if(Trama[0]&0x08)                          /* Si hay datos de motores alfa */
   {  Sensores[E_SENS_MA0]=(Trama[offset]>>4);
      Sensores[E_SENS_MA1]=(Trama[offset]&0x0F);
      offset++;
      Sensores[E_SENS_MA2]=(Trama[offset]>>4);
      Sensores[E_SENS_MA3]=(Trama[offset]&0x0F);
      offset++;
      Sensores[E_SENS_MA4]=(Trama[offset]>>4);
      Sensores[E_SENS_MA5]=(Trama[offset]&0x0F);
      offset++;
      SensAct[E_ACT_MTALFA]=1;                    /* Hay nuevos datos de motores alfa */
   }
   else
      SensAct[E_ACT_MTALFA]=0;                    /* No hay nuevos datos de motores alfa */

   if(Trama[0]&0x04)                          /* Si hay datos de motores beta */
   {  Sensores[E_SENS_MB0]=(Trama[offset]>>4);
      Sensores[E_SENS_MB1]=(Trama[offset]&0x0F);
      offset++;
      Sensores[E_SENS_MB2]=(Trama[offset]>>4);
      Sensores[E_SENS_MB3]=(Trama[offset]&0x0F);
      offset++;
      Sensores[E_SENS_MB4]=(Trama[offset]>>4);
      Sensores[E_SENS_MB5]=(Trama[offset]&0x0F);
      offset++;
      SensAct[E_ACT_MTBETA]=1;                    /* Hay nuevos datos de motores beta */
   }
   else
      SensAct[E_ACT_MTBETA]=0;                    /* Hay nuevos datos de motores beta */

   if(Trama[0]&0x02)                          /* Si hay datos de posiciones de tacto */
   {  Sensores[E_SENS_PT1]=((Trama[offset]>>1)&0x0F);
      Sensores[E_SENS_PT2]=(Trama[offset]>>5);      /* Falta bit mas peso */
      offset++;
      Sensores[E_SENS_PT3]=(Trama[offset]>>5);      /* Falta bit mas peso */
      SensAct[E_ACT_PTACTO]=1;                 /* Hay nuevos datos de posiciones de tacto */
   }
   else
      SensAct[E_ACT_PTACTO]=0;                 /* No hay nuevos datos de posiciones de tacto */

   if(Trama[0]&0x01)                          /* Si hay datos de fuerzas de tacto */
   {  Sensores[E_SENS_FT2]=((Trama[offset]>>1)&0x0F);
      offset++;
      Sensores[E_SENS_FT1]=(Trama[offset]>>5);      /* Falta bit mas peso */
      Sensores[E_SENS_FT3]=((Trama[offset]>>1)&0x0F);
      offset++;
      SensAct[E_ACT_FTACTO]=1;                    /* Hay nuevos datos de fuerzas de tacto */
   }
   else
```

```
   {  if(SensAct[E_ACT_PTACTO])                    /* Si habia posiciones de tacto */
         offset++;                              /* hay que incrementar el offset */
      SensAct[E_ACT_FTACTO]=0;                  /* No hay nuevos datos de fuerzas de tacto */
   }

   /* Recuperamos los bits de posiciones y fuerzas que faltan */
   if(SensAct[E_ACT_PTACTO])          /* Si se han actualizado posiciones tacto */
   {  Sensores[E_SENS_PT2]|=((Trama[offset]>>2)&0x08);     /* A¤adimos bit a PT2 */
      Sensores[E_SENS_PT3]|=((Trama[offset]>>3)&0x08); /* A¤adimos bit a PT3 */
   }
   if(SensAct[E_ACT_FTACTO])                  /* Si se han actualizado fuerzas de tacto */
      Sensores[E_SENS_FT1]|=((Trama[offset]>>4)&0x08); /* A¤adimos bit a FT1 */
   if(Trama[1]&0x80)                          /* Si hay datos de infrarrojos */
   {  Sensores[E_SENS_IFR]=(Trama[offset]&0x0F);
      SensAct[E_ACT_INFRAR]=1;                      /* Hay nuevos datos de infrarrojos */
   }
   else
      SensAct[E_ACT_INFRAR]=0;                      /* No hay nuevos datos de infrarrojos */

   /* Si habia datos de p. tacto, f. tacto o infrarr. incrementar offset */
   if(SensAct[E_ACT_PTACTO]||SensAct[E_ACT_FTACTO]||SensAct[E_ACT_INFRAR])
      offset++;

   if(Trama[1]&0x40)                          /* Si hay datos de inclinometro */
   {  Sensores[E_SENS_INC]=Trama[offset];
      offset++;
      SensAct[E_ACT_INCLIN]=1;                      /* Hay nuevos datos de inclinometro */
   }
   else
      SensAct[E_ACT_INCLIN]=0;                      /* No hay nuevos datos de inclinometro */

   if(Trama[1]&0x20)                                /* Si hay datos de piros */
   {  if(TipoTrama==0)
      {  Sensores[E_SENS_PR2]=Trama[offset];
         offset++;
         Sensores[E_SENS_PR3]=Trama[offset];
         offset++;
         SensAct[E_ACT_PIR_23]=1;                   /* Hay nuevos datos de piros 2,3 */
         SensAct[E_ACT_PIR145]=0;                   /* No hay nuevos datos de piros 1,4,5 */
      }
      else
      {  Sensores[E_SENS_PR1]=Trama[offset];
         offset++;
         Sensores[E_SENS_PR4]=Trama[offset];
         offset++;
         Sensores[E_SENS_PR5]=Trama[offset];
         offset++;
         SensAct[E_ACT_PIR_23]=0;                   /* No hay nuevos datos de piros 2,3 */
         SensAct[E_ACT_PIR145]=1;                   /* Hay nuevos datos de piros 1,4,5 */
      }
   }
   else
   {  SensAct[E_ACT_PIR_23]=0;                      /* No hay nuevos datos de piros 2,3 */
      SensAct[E_ACT_PIR145]=0;                      /* No hay nuevos datos de piros 1,4,5 */
   }

   if(Trama[1]&0x10)                          /* Si hay datos de bateria */
   {  if(TipoTrama==0)
      {  Sensores[E_SENS_BTE]=Trama[offset];
         SensAct[E_ACT_BTELEC]=1;                   /* Hay nuevos datos de piros 2,3 */
         SensAct[E_ACT_BTMOTR]=0;                   /* No hay nuevos datos de piros 1,4,5 */
      }
      else
      {  Sensores[E_SENS_BTM]=Trama[offset];
         SensAct[E_ACT_BTELEC]=0;                   /* No hay nuevos datos de piros 2,3 */
         SensAct[E_ACT_BTMOTR]=1;                   /* Hay nuevos datos de piros 1,4,5 */
      }
   }
   else
   {  SensAct[E_ACT_BTELEC]=0;       /* No hay nuevos datos de piros 2,3 */
      SensAct[E_ACT_BTMOTR]=0;       /* No hay nuevos datos de piros 1,4,5 */
   }

   if(Trama[1]&0x08)                          /* Si hay datos de wiskers */
   {  Sensores[E_SENS_WKR]=((Trama[1]>>1)&0x01);
      Sensores[E_SENS_WKL]=(Trama[1]&0x01);
      SensAct[E_ACT_WHISKR]=1;                  /* Hay nuevos datos de wiskers */
   }
   else
      SensAct[E_ACT_WHISKR]=0;                  /* No hay nuevos datos de wiskers */
}
```

```
/* ********************************************************************** */
/*                              E_Read                                    */
/* ********************************************************************** */
int E_Read(int *TramaLeida)
{  if(E_Init==0)                          /* Si nivel enlace no abierto (no inicializado) */
      return(E_ERROR_NO_OPEN);            /* Error primero hay que abrirlo */

   *TramaLeida=LeeUltimaTramaCorrecta();
   if(*TramaLeida)
      RecuperaDatosTrama(TramaSens);

   return(E_NO_ERROR);
}


/* ********************************************************************** */
/*                              E_State                                   */
/* ********************************************************************** */
int E_State(unsigned int param,unsigned long *valor)
{  if(E_Init==0)                          /* Si nivel enlace no abierto (no inicializado) */
      return(E_ERROR_NO_OPEN);            /* Error primero hay que abrirlo */

   if((param&0xF000)==0x1000)
   {  switch(param)
      {  case E_TRAMAS_DETECTADAS:
            *valor=EstadoEnlace.TramasDetectadas;
            break;

         case E_TRAMAS_CORRECTAS:
            *valor=EstadoEnlace.TramasCorrectas;
            break;

         case E_TRAMAS_ERRONEAS:
            *valor=EstadoEnlace.TramasErroneas;
            break;

         case E_ERRORES_TIPO_TRAMA:
            *valor=EstadoEnlace.ErrorTipoTrama;
            break;

         case E_ERRORES_CHECKSUM:
            *valor=EstadoEnlace.ErrorChecksum;
            break;

         case E_NUM_TOUT:
            *valor=EstadoEnlace.NumTout;
            break;

         case E_NUM_TOUT_PET_DAT:
            *valor=EstadoEnlace.NumToutPetDat;
            break;
      }
   }
   else
   {  if(F_State(param,valor)==F_NO_ERROR)
         return(E_NO_ERROR);
      else
         return(E_ERROR_PARAM_DESCONOCIDO);
   }

   return(E_NO_ERROR);
}


/* ********************************************************************** */
/*                              EstCom                                    */
/* ********************************************************************** */
int E_EstCom(int tipo,int *valor)
{  int aux;

   if(E_Init==0)                          /* Si nivel enlace no abierto (no inicializado) */
      return(E_ERROR_NO_OPEN);            /* Error primero hay que abrirlo */

   *valor=(EstadoEnlace.EstCom & tipo);
   if(tipo!=E_ALL_ERR_IND)                /* Si se quiere un error especifico */
   {  if(tipo!=0)                         /* Si el parametro es correcto */
      {  aux=tipo;
         while(aux!=0x01)
         {  aux>>=1;                      /* Desplazamos el numero de bits necesario */
            (*valor)>>=1;                 /* para devolver 0 o 1 */
         }
```

```
      }
      else
         return(E_ERROR_PARAM_DESCONOCIDO);
   }
   return(E_NO_ERROR);
}


/* ********************************************************************** */
/*                              WriteQueDatos                             */
/* ********************************************************************** */
int E_WriteQueDatos(int GrpSens,int valor)
{  if(E_Init==0)                        /* Si nivel enlace no abierto (no inicializado) */
   return(E_ERROR_NO_OPEN);             /* Error primero hay que abrirlo */

   QueDatos&=~GrpSens;                  /* Ponemos a cero el grupo o grupos indicados */
   if(valor!=0)                         /* Si se quieren activar  (valor != 0) */
      QueDatos|=GrpSens;                /* Grupo o grupos de sensores a uno */
   /* else; si se quieren desactivar se dejan a cero */
   HayQueDatos=1;
   return(E_NO_ERROR);
}


/* ********************************************************************** */
/*                              ReadQueDatos                              */
/* ********************************************************************** */
int E_ReadQueDatos(int GrpSens,int *valor)
{  if(E_Init==0)      /* Si nivel enlace no abierto (no inicializado) */
      return(E_ERROR_NO_OPEN);       /* Error primero hay que abrirlo */

   *valor=(QueDatos & GrpSens);
   return(E_NO_ERROR);
}

/* ********************************************************************** */
/*                                E_ResetRob                              */
/* ********************************************************************** */
int E_ResetRob(void)
{  int i;

   if(E_Init==0)      /* Si nivel enlace no abierto (no inicializado) */
      return(E_ERROR_NO_OPEN);       /* Error primero hay que abrirlo */

   /* Ordenes al robot a su estado inicial, posiciones intermedias */
   for(i=0;i<TAM_ORDENES-2;i++)
      Ordenes[i]=30;  /* Ordenes iniciales a motores: 30, pos. intermedia */
   Ordenes[i]=0;             /* Orden inicial a los leds: 0, apagados */
   Ordenes[i+1]=0;    /* Orden inicial al buzzer: 0, desactivado */
   F_ResetRob();

   return E_NO_ERROR;
}
```

Fichero "prtcgeng.h" (fichero de cabecera de la librería que implementa el protocolo)

```
#ifndef _PRTCGENG
#define _PRTCGENG

/* Parametros para consultar el estado del nivel fisico. Todos los
   parametros para consultar el estado del nivel fisico comienzan por
   cero en hexadecimal. Esto permite a niveles superiores detectar que el
   parametro es de nivel fisico */
#define F_ERROR_OVERRUN        0x0001       /* Contador errores de overrun */
#define F_ERROR_PARIDAD        0x0002       /* Contador errores de paridad */
#define F_ERROR_TRAMA          0x0003       /* Contador errores de trama */
#define F_ERROR_BREAKLINE      0x0004       /* Contador errores de breakline */
#define F_ERROR_OVERRUN_BUFFER 0x0005       /* Contador errores overrun buffers */


/* Definicion de tipos de error */
#define E_NO_ERROR                0         /* No hay error */
#define E_ERROR_NO_OPEN          -3         /* Nivel enlace no abierto */
#define E_ERROR_NO_CLOSE         -4         /* Nivel enlace no cerrado */
#define E_ERROR_PARAM_DESCONOCIDO -5        /* Parametro estado desconocido */
#define E_ERROR_OPEN_FISICO       -6        /* Error al abrir nivel fisico */
```

```
/* Parametros para consultar el estado del nivel de enlace. Todos los
   parametros para consultar el estado del nivel de enlace comienzan por
   1 en hexadecimal. Esto permite a niveles superiores detectar que el
   parametro es de nivel de enlace o al nivel de enlace detectar si
   es un parametro para el o para el nivel fisico inferior */
#define E_TRAMAS_DETECTADAS  0x1001         /* Contador SOH detectados */
#define E_TRAMAS_CORRECTAS   0x1002         /* Contador tramas correctas */
#define E_TRAMAS_ERRONEAS    0x1003         /* Contador tramas erroneas */
#define E_ERRORES_TIPO_TRAMA 0x1004         /* Contador errores por tipo trama */
#define E_ERRORES_CHECKSUM   0x1005         /* Contador errores checksum */
#define E_NUM_TOUT           0x1006         /* Contador numero timeouts */
#define E_NUM_TOUT_PET_DAT   0x1007         /* Contador timeouts peticion datos */


/* Definiciones para acceso a los datos de los sensores. Parametros de
   "ReadSensor". */
#define E_SENS_MA0 0x00       /* Motor alfa0 */
#define E_SENS_MA1 0x01       /* Motor alfa1 */
#define E_SENS_MA2 0x02       /* Motor alfa2 */
#define E_SENS_MA3 0x03       /* Motor alfa3 */
#define E_SENS_MA4 0x04       /* Motor alfa4 */
#define E_SENS_MA5 0x05       /* Motor alfa5 */
#define E_SENS_MB0 0x06       /* Motor beta0 */
#define E_SENS_MB1 0x07       /* Motor beta1 */
#define E_SENS_MB2 0x08       /* Motor beta2 */
#define E_SENS_MB3 0x09       /* Motor beta3 */
#define E_SENS_MB4 0x0A       /* Motor beta4 */
#define E_SENS_MB5 0x0B       /* Motor beta5 */
#define E_SENS_PT1 0x0C       /* Posicion tacto1 */
#define E_SENS_PT2 0x0D       /* Posicion tacto2 */
#define E_SENS_PT3 0x0E       /* Posicion tacto3 */
#define E_SENS_FT1 0x0F       /* Fuerza tacto1 */
#define E_SENS_FT2 0x10       /* Fuerza tacto2 */
#define E_SENS_FT3 0x11       /* Fuerza tacto3 */
#define E_SENS_IFR 0x12       /* Infrarrojos */
#define E_SENS_INC 0x13       /* Inclinometro */
#define E_SENS_PR1 0x14       /* Piro1 */
#define E_SENS_PR2 0x15       /* Piro2 */
#define E_SENS_PR3 0x16       /* Piro3 */
#define E_SENS_PR4 0x17       /* Piro4 */
#define E_SENS_PR5 0x18       /* Piro5 */
#define E_SENS_BTE 0x19       /* Bateria electronica */
#define E_SENS_BTM 0x1A       /* Bateria de motores */
#define E_SENS_WKR 0x1B       /* Whisker derecho */
#define E_SENS_WKL 0x1C       /* Whisker izquierdo */

/* Definiciones para acceso a los datos de las ordenes. Parametros de
   "WriteMotor" y "ReadMotor". */
#define E_ORD_MA0 0x00        /* Motor alfa0 */
#define E_ORD_MA1 0x01        /* Motor alfa1 */
#define E_ORD_MA2 0x02        /* Motor alfa2 */
#define E_ORD_MA3 0x03        /* Motor alfa3 */
#define E_ORD_MA4 0x04        /* Motor alfa4 */
#define E_ORD_MA5 0x05        /* Motor alfa5 */
#define E_ORD_MB0 0x06        /* Motor beta0 */
#define E_ORD_MB1 0x07        /* Motor beta1 */
#define E_ORD_MB2 0x08        /* Motor beta2 */
#define E_ORD_MB3 0x09        /* Motor beta3 */
#define E_ORD_MB4 0x0A        /* Motor beta4 */
#define E_ORD_MB5 0x0B        /* Motor beta5 */


/* Definiciones para acceso a que grupo de sensores se ha actualizado
   despues de una lectura de sensores. Parametros para "ActSensor". */
#define E_ACT_MTALFA 0x00     /* Motores alfa */
#define E_ACT_MTBETA 0x01     /* Motores beta */
#define E_ACT_PTACTO 0x02     /* Posiciones de tacto */
#define E_ACT_FTACTO 0x03     /* Fuerzas de tacto */
#define E_ACT_INFRAR 0x04     /* Infrarrojos */
#define E_ACT_INCLIN 0x05     /* Inclinometro */
#define E_ACT_PIR_23 0x06     /* Piros 2 y 3 */
#define E_ACT_PIR145 0x07     /* Piros 1, 4 y 5 */
#define E_ACT_BTELEC 0x08     /* Bateria electronica */
#define E_ACT_BTMOTR 0x09     /* Bateria de motores */
#define E_ACT_WHISKR 0x0A     /* Whiskers */


/* Definiciones para indicar o consultar que grupos de sensores se desean
   recibir. Los sensores de cara al usuario siempre se consideran agrupados
   en una variable de tipo entero de manera que cada bit corresponde a uno
   de los grupos de sensores desde el bit de menos peso (bit 0) hasta el
```

```
   bit numero 8, en total nueve grupos. Parametros para "ReadQueDatos" y
   "WriteQueDatos". */
#define E_ALL_DAT 0x01FF      /* Todos los grupos de sensores */
#define E_DAT_WSK 0x0001      /* Whiskers */
#define E_DAT_BTR 0x0002      /* Baterias: de motores y electronica */
#define E_DAT_PIR 0x0004      /* Piros */
#define E_DAT_INC 0x0008      /* Inclinometro */
#define E_DAT_INF 0x0010      /* Infrarrojos */
#define E_DAT_FTC 0x0020      /* Fuerzas de tacto */
#define E_DAT_PTC 0x0040      /* Posiciones de tacto */
#define E_DAT_MTB 0x0080      /* Motores beta */
#define E_DAT_MTA 0x0100      /* Motores alfa */


/* Tipos de errores o indicaciones sobre el estado de la comunicacion.
   Parametros para la funcion "E_EstCom" */
#define E_ALL_ERR_IND    0x007F      /* Consultar todos errores o indicaciones */
#define E_IND_ENV_CICLO  0x0001      /* Ordenes no enviadas dentro del ciclo */
#define E_ERR_ENV        0x0002      /* Error al enviar la trama de ordenes */
#define E_OVERR_SENSORES 0x0004      /* Varias tramas detectadas en lectura */
#define E_OVERR_ORDENES  0x0008      /* Nueva orden antes de enviar la anterior */
#define E_IND_NO_OPTIMO  0x0010      /* SOH detectados menor numero optimo */
#define E_ERR_TIMEOUT    0x0020      /* Timeout sin recibir nada */
#define E_ERR_PET_DAT    0x0040      /* Timeout sin recibir datos pedidos */



/* Macros de interficie para escribir ordenes a los actuadores en el buffer
   de usuario. En   caso de que el valor que se quiere escribir sea correcto
   para cada tipo   de sensor la macro evalua 1 y en caso de que el valor a
   escribir para el sensor sea incorrecto evalua 0 y no se escribe el valor.
   Si el valor es correcto se actualiza el flag indicando que hay nuevas
   ordenes. Estos datos no se envian al host remoto hasta que se ejecuta
   E_Write. */
/* Macro para escribir una orden a uno de los motores */
#define E_WriteMotor(motor,valor)\
((motor<E_ORD_MA0 || motor>E_ORD_MB5 || valor>63 ||\
valor<0 )?0:(valor==(Ordenes[motor]=valor))&&(HayOrdenes=1)==1)
/* Macro para escribir una orden a los leds */
#define E_WriteLeds(valor)\
((valor>255 || valor<0)?0:(valor==(Ordenes[12]=valor))&&(HayOrdenes=1)==1)
/* Macro para escribir una orden al buzzer */
#define E_WriteBuzzer(valor)\
((valor>255 || valor<0)?0:(valor==(Ordenes[13]=valor))&&(HayOrdenes=1)==1)


/* Macros de interficie para consultar el valor de las ordenes a los
   actuadores en el buffer de usuario */
/* Macro para consultar el valor de la orden de uno de los motores. Si el
   parametro es correcto devuelve el valor del motor correspondiente y si
   no devuelve 255 (valor imposible para un motor). */
#define E_ReadMotor(motor)\
((motor>=E_ORD_MA0 && motor<=E_ORD_MB5)?Ordenes[motor]:255)
/* Macro para consultar el valor de la orden de los leds */
#define E_ReadLeds() (Ordenes[12])
/* Macro para consultar el valor de la orden del buzzer */
#define E_ReadBuzzer() (Ordenes[13])


/* Macro de interficie para consultar el valor de los sensores en el buffer
   de usuario. Si el parametro es correcto retorna el valor del sensor
   correspondiente, si no retorna 255 (valor imposible para un sensor). */
#define E_ReadSensor(tipo)\
((tipo>=E_SENS_MA0 && tipo<=E_SENS_WKL)?Sensores[tipo]:255)


/* Macro de interficie para consultar si un grupo de sensores ha sido
   actualizado   tras la ultima lectura de sensores. Si el parametro es
   correcto retorna el valor de si el grupo de sensores ha sido actualizado
   (1) o no (0) y en caso contrario retorna 255 (valor imposible). */
#define E_ActSensor(grupo)\
((grupo>=E_ACT_MTALFA && grupo<=E_ACT_WHISKR)?SensAct[grupo]:255)


/* Funcion que inicializa el nivel de enlace. No tiene parametros. Si todo ha
   ido bien retorna E_NO_ERROR y en caso contrario retorna el error
   correspondiente. */
int E_Open(void);


/* Funcion que cierra el nivel de enlace. Si todo ha ido bien retorna
   E_NO_ERROR y en caso contrario retorna el error correspondiente. */
```

```
int E_Close(void);


/* Funcion con la que se puede indicar al nivel de enlace la llegada de un
   nuevo caracter para coordinar el envio de ordenes (niveles inferiores).
   El nivel de enlace consulta dicho caracter que se le pasa como parametro y
   en caso de detectar un inicio de trama inicia el envio de ordenes al nivel
   fisico. No retorna parametros pero en caso de que el nivel de enlace no
   este inicializado no hace nada. */
void E_CaracterIndication(unsigned char c);


/* Funcion que indica que se pueden enviar los datos escritos en los buffers
   de usuario "Ordenes" con ordenes a los motores y "QueDatos" donde se
   indica que datos de que sensores se quieren recibir. La funcion copia
   los datos en el buffer "TramaOrd" calculando la trama, que sera enviada
   en el momento en que se reciba una nuevo inicio de trama procedente del
   host remoto. En caso de que todo vaya bien retorna E_NO_ERROR y en caso
   contrario retorna el error correspondiente. */
int E_Write(void);


/* Funcion que permite obtener el valor de los sensores de la ultima trama
   correcta recibida en el vector "Sensores". En caso de que todo vaya bien
   retorna E_NO_ERROR y si hay algun problema retorna el error
   correspondiente. Si todo va bien en "TramaLeida" se pone un 1 en caso de
   disponer de   informacion   de una nueva trama o 0 en caso contrario. En el
   vector "SensAct" se indica con un 1 los grupos de sensores actualizados y
   con un 0 los que no han sido actualizados con la lectura de la trama. En
   caso de que hubiera mas de una trama acumulada en al buffer desde la
   anterior   lectura se activa la   indicacion IND_REC_CICLO. */
int E_Read(int *TramaLeida);


/* Funcion que sirve de interficie para consultar los parametros del
   nivel de enlace o del nivel fisico, principalmente contadores de errores
   o tramas. Se le pasa el   parametro de nivel de enlace o fisico
   correspondiente a la variable de estado que se quiere consultar y un
   puntero a la variable de tipo long donde se quiere obtener el valor.
   Si el parametro es de nivel fisico (inferior) se realiza una llamada
   a F_State con el mismo parametro. En caso de que el nivel de enlace no
   este inicializado retorna E_NO_OPEN y si el parametro no corresponde a
   nivel de enlace ni fisico se retorna E_ERROR_PARAM_DESCONOCIDO. Si todo
   va bien retorna F_NO_ERROR. */
int E_State(unsigned int param,unsigned long *valor);


/* Funcion de interficie para consultar el valor del indicador o error sobre
   el   estado de la comunicacion que se le pide. Si todo va bien retorna
   E_NO_ERROR y en caso contrario retorna el error correspondiente. En "tipo"
   se le indica si se quiere consultar todos los indicadores del estado de la
   comunicacion o solo uno concreto. Si todo va bien, en "valor" se indica si
   el error o indicador esta activo   (1) o si no (0) en el caso de un tipo
   concreto o se indica un valor entero de manera que cada uno de los bits
   corresponde a uno de los posibles errores o indicaciones, de manera que se
   pueden consultar con las mascaras que los identifican y que sirven como
   parametro "tipo". */
int E_EstCom(int tipo,int *valor);


/* Funcion que permite escribir en el buffer de usuario "QueDatos" para
   indicar la informacion de que sensores se quiere recibir del host
   remoto. Se puede pedir un grupo concreto o todos a la vez con las
   mascaras definidas anteriormente. Esta informacion solo se envia al
   host remoto cuando se ejecuta E_Write. Si todo va bien retorna E_NO_ERROR
   y en caso contrario retorna el error correspondiente. */
int E_WriteQueDatos(int GrpSens,int valor);

/* Funcion que permite consultar del buffer de usuario "QueDatos" que
   sensores   se habia indicado recibir hasta el momento o si un grupo
   concreto de sensores se habia pedido o no anteriormente. Si todo va
   bien retorna E_NO_ERROR y en caso contrario retorna el error
   correspondiente. */
int E_ReadQueDatos(int GrpSens,int *valor);

/* Funcion inicializa el buffer de ordenes a su estado inicial (posiciones
   intermedias para los motores y cero para leds y buzzer) y hace un reset
   por software del robot */
int E_ResetRob(void);

/* ****************************  PRIVADO ***************************** */
/* Variables privadas de nivel de enlace, solo accesibles por el usuario
```

```
   a traves de las macros o funciones de interficie. Nunca se deberian
   acceder directamente. Aqui solo para que sean visibles al usuario
   del nivel de enlace. */

/* Definicion de tama¤os para los buffers accesibles por el usuario */
#define TAM_SENSORES 29
#define TAM_SENSACT 11
#define TAM_ORDENES 14
#define TAM_QUEDATOS 2

/* Buffers visibles por el usuario del nivel de Enlace con Ordenes,
   Tipo de datos que se quieren recibir, valor de Sensores y indicador de
   que sensores se han actualizado tras una lectura */
extern unsigned char Ordenes[TAM_ORDENES];   /* 12 motores + led + buzzer */
extern int QueDatos;   /* Solo se utilizan 9 bits */
extern unsigned char Sensores[TAM_SENSORES];   /* 28 sensores diferentes */
extern unsigned char SensAct[TAM_SENSACT];   /* 11 grupos sensores */
extern int HayOrdenes;
extern int HayQueDatos;

/* ****************************  PRIVADO ***************************** */

#endif
```

ANEXO 7. CÓDIGO FUENTE DE LA APLICACIÓN

A continuación se adjunta el código fuente de la aplicación. En cada caso se indica al fichero al que corresponde el código. Hay partes del código correspondientes a caracteres que se escriben por pantalla (por ejemplo al dibujar la silueta del robot) que al trasladarlas a este documento no corresponden con el fichero original.

Fichero "es.h"

```
#ifndef _ES
#define _ES


/* Zona de pantalla para leer el valor de las ordenes 1 y 2. La orden 1
   puede ser de motor alfa, led o buzzer dependiendo del "ActuadorActivo"
   y la orden 2 solo puede ser un motor beta */
#define X_ORDEN1 73
#define Y_ORDEN1 9
#define X_ORDEN2 73
#define Y_ORDEN2 11


/* Funcion que salva el estado de la pantalla y inicializa la pantalla*/
void OpenPantalla(void);

/* Funcion que restaura el estado anterior de la pantalla */
void ClosePantalla(void);

/* Funcion para leer de teclado teclas incluso de funcion. Una tecla de
   de funcion siempre viene precedida de un cero. En caso de tecla de
   funcion el valor de la tecla se pone el el byte alto del entero y
   el byte bajo queda a cero */
int getchExt(void);

/* Funcion que lee un natural de "n_dig" digitos (como maximo 4),
   mostrandolo por pantalla a partir de la posicion (x,y) y no lo acepta
   hasta que se pulsa RETURN, permitiendo hasta ese momento la correccion
   de alguna de sus cifras pulsando BACKSPACE. Si se ha leido algun natural
   retorna dicho valor y en caso contrario retorna -1. */

int LeerNat(int x,int y,int n_dig);

/* Funcion que dibuja el robot en pantalla marcando el actuador activo que
   se le pasa como parametro y mostrando el valor escrito para dicho
   actuador con anterioridad en el buffer de ordenes a enviar al robot */
void DibujarRobot(int ActuadorActivo);


/* Funcion que inicializa la pantalla con el dise¤o basico sobre el que
   se realizara el refresco de contadores, etc ... */
void IniPantalla(void);

/* Funcion que realiza el refresco de pantalla segun la situacion actual
   de contadores, etc... */
void RefrescoPantalla(int ActuadorActivo,int RefrescoRobot);

#endif
```

Fichero "es.c"

```
#include <conio.h>
#include <stdlib.h>
#include "es.h"
#include "enlace.h"
```

```
#define VENTANA_TITULO 1            /* Altura ventana titulo */
#define VENTANA_EST_FISICO 5        /* Altura ventana estado nivel fisico */
#define VENTANA_EST_ENLACE 11       /* Altura ventana estado nivel enlace */
#define VENTANA_SENSORES 8          /* Altura ventana de estado sensores */
#define VENTANA_ORDENES 19          /* Altura ventana de estado ordenes */


/* El dibujo del robot se puede mover a partir del recuadro virtual que
   ocupa definido por la esquina superior izquierda y el ancho y alto del
   dibujo */
#define LEFT_ROB 8           /* Posicion x esquina izquierda dibujo del robot */
#define UP_ROB 2             /* Posicion y esquina izquierda dibujo del robot */


/* Zona de memoria para salvar el estado de la pantalla */
unsigned char Pantalla[4000];
struct text_info TextInf;


/* ********************************************************************** */
/*                            OpenPantalla                                */
/* ********************************************************************** */

void OpenPantalla(void)
{  gettext(1,1,80,25,Pantalla);
   gettextinfo(&TextInf);
   textmode(C4350);
   _setcursortype(_NOCURSOR);
   clrscr();
}


/* ********************************************************************** */
/*                            ClosePantalla                               */
/* ********************************************************************** */

void ClosePantalla(void)
{  textmode(C80);
   puttext(1,1,80,25,Pantalla);
   textcolor(TextInf.attribute);
   gotoxy(TextInf.curx,TextInf.cury);
   _setcursortype(_NORMALCURSOR);
}


/* ********************************************************************** */
/*                            getchExt                                    */
/* ********************************************************************** */

int getchExt(void)
{  int c;

   c=getch();
   if(c==0)                  /* Indica que es una tecla de funcion */
   {  c=getch();             /* Obtenemos la tecla */
      c<<=8;                 /* Dejamos la tecla de funcion en el byte alto */
   }
   return c;
}


/* ********************************************************************** */
/*                           LeerNat                                      */
/* ********************************************************************** */

int LeerNat(int x,int y,int n_dig)
{  int i,lon,wx,fin=0;
   char c,str[5]="\0";

   if((x>80)||(x<1)||(y>25)||(y<1)||(n_dig<=0)||(n_dig>4))
      return(-1);

   lon=n_dig;
   gotoxy(x,y);
   textcolor(BLACK);
   _setcursortype(_NORMALCURSOR);

   while(!fin)
   {  do
      c=getch();
```

```
      while(((c<'0')||(c>'9'))&&(c!='\r')&&(c!='\b'));

      wx=wherex();
      if((c!='\r')&&(wx>x)&&(lon==n_dig))
      {  gotoxy(x,y);
         for(i=0;i<n_dig;i++)
         putch(' ');
         gotoxy(x,y);
         wx=wherex();
      }
      switch(c)
      {  case '\b':
            if(wx>x)
            {  gotoxy(wx-1,y);
               putch(' ');
               gotoxy(wx-1,y);
               lon++;
            }
            break;

         case '\r':
            fin=1;
            break;

         default:
            if(lon>0)
            {  putch(c);
               str[n_dig-lon]=c;
               lon--;
            }
      }
   }

   _setcursortype(_NOCURSOR);

   if(lon>0)
      str[n_dig-lon]='\0';
   else
      str[n_dig]='\0';

   if(str[0]!='\0')
      return(atoi(str));
   else
      return(-1);
}


/* ********************************************************************** */
/*                              DibujarRobot                              */
/* ********************************************************************** */

void DibujarRobot(int ActuadorActivo)
{  int ValorOrden1,ValorOrden2;

   textcolor(MAGENTA);

   gotoxy(LEFT_ROB,5);
   cputs("        /îî\\");
   gotoxy(LEFT_ROB,6);
   cputs("\\      /    \\");
   gotoxy(LEFT_ROB,7);
   cputs(" \\    /      \\ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿");
   gotoxy(LEFT_ROB,8);
   cputs("  \\  ³                                    ³");
   gotoxy(LEFT_ROB,9);
   cputs("   \\ ³                                    ³");
   gotoxy(LEFT_ROB,10);
   cputs("    >³                                   ³");
   gotoxy(LEFT_ROB,11);
   cputs("   / ³                                   ³");
   gotoxy(LEFT_ROB,12);
   cputs("  /  ³                                   ³");
   gotoxy(LEFT_ROB,13);
   cputs(" /    \\      /ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ");
   gotoxy(LEFT_ROB,14);
   cputs("/      \\    /");
   gotoxy(LEFT_ROB,15);
   cputs("        \\__/");

   if(ActuadorActivo==1)
      textcolor(YELLOW);
   gotoxy(LEFT_ROB+16,UP_ROB+12); putch('³');
```

```
gotoxy(LEFT_ROB+16,UP_ROB+13); putch('³');
gotoxy(LEFT_ROB+16,UP_ROB+14); putch('³');
gotoxy(LEFT_ROB+16,UP_ROB+15); putch('Á');
gotoxy(LEFT_ROB+15,UP_ROB+16); cputs("MT0");
textcolor(MAGENTA);

if(ActuadorActivo==2)
   textcolor(YELLOW);
gotoxy(LEFT_ROB+15,UP_ROB); cputs("MT1");
gotoxy(LEFT_ROB+16,UP_ROB+1); putch('Â');
gotoxy(LEFT_ROB+16,UP_ROB+2); putch('³');
gotoxy(LEFT_ROB+16,UP_ROB+3); putch('³');
gotoxy(LEFT_ROB+16,UP_ROB+4); putch('³');
textcolor(MAGENTA);

if(ActuadorActivo==3)
   textcolor(YELLOW);
gotoxy(LEFT_ROB+26,UP_ROB+12); putch('³');
gotoxy(LEFT_ROB+26,UP_ROB+13); putch('³');
gotoxy(LEFT_ROB+26,UP_ROB+14); putch('³');
gotoxy(LEFT_ROB+26,UP_ROB+15); putch('Á');
gotoxy(LEFT_ROB+25,UP_ROB+16); cputs("MT2");
textcolor(MAGENTA);

if(ActuadorActivo==4)
   textcolor(YELLOW);
gotoxy(LEFT_ROB+25,UP_ROB); cputs("MT3");
gotoxy(LEFT_ROB+26,UP_ROB+1); putch('Â');
gotoxy(LEFT_ROB+26,UP_ROB+2); putch('³');
gotoxy(LEFT_ROB+26,UP_ROB+3); putch('³');
gotoxy(LEFT_ROB+26,UP_ROB+4); putch('³');
textcolor(MAGENTA);

if(ActuadorActivo==5)
   textcolor(YELLOW);
gotoxy(LEFT_ROB+36,UP_ROB+12); putch('³');
gotoxy(LEFT_ROB+36,UP_ROB+13); putch('³');
gotoxy(LEFT_ROB+36,UP_ROB+14); putch('³');
gotoxy(LEFT_ROB+36,UP_ROB+15); putch('Á');
gotoxy(LEFT_ROB+35,UP_ROB+16); cputs("MT4");
textcolor(MAGENTA);

if(ActuadorActivo==6)
   textcolor(YELLOW);
gotoxy(LEFT_ROB+35,UP_ROB); cputs("MT5");
gotoxy(LEFT_ROB+36,UP_ROB+1); putch('Â');
gotoxy(LEFT_ROB+36,UP_ROB+2); putch('³');
gotoxy(LEFT_ROB+36,UP_ROB+3); putch('³');
gotoxy(LEFT_ROB+36,UP_ROB+4); putch('³');
textcolor(MAGENTA);

if(ActuadorActivo==7)
   textcolor(YELLOW);
gotoxy(LEFT_ROB+18,UP_ROB+7); cputs("LEDS");
gotoxy(LEFT_ROB+18,UP_ROB+9); cputs("ÛÛÛÛ");
textcolor(MAGENTA);

if(ActuadorActivo==8)
   textcolor(YELLOW);
gotoxy(LEFT_ROB+29,UP_ROB+7); cputs("BUZZER");
gotoxy(LEFT_ROB+29,UP_ROB+9); cputs("ÛÛÛÛÛÛÛ");


textcolor(BLACK);
gotoxy(X_ORDEN1-18,Y_ORDEN1);
cputs("                    ");
gotoxy(X_ORDEN2-18,Y_ORDEN2);
cputs("                    ");

textcolor(YELLOW);
if(ActuadorActivo>0 && ActuadorActivo<7)   /* Motores */
{  gotoxy(X_ORDEN1-18,Y_ORDEN1);
   cputs("ORDEN MOTOR ALFA:");
   gotoxy(X_ORDEN2-18,Y_ORDEN2);
   cputs("ORDEN MOTOR BETA:");

   switch(ActuadorActivo)
   {  case 1:
         ValorOrden1=E_ReadMotor(E_ORD_MA0);
         ValorOrden2=E_ReadMotor(E_ORD_MB0);
         break;
      case 2:
```

```
            ValorOrden1=E_ReadMotor(E_ORD_MA1);
            ValorOrden2=E_ReadMotor(E_ORD_MB1);
            break;
         case 3:
            ValorOrden1=E_ReadMotor(E_ORD_MA2);
            ValorOrden2=E_ReadMotor(E_ORD_MB2);
            break;
         case 4:
            ValorOrden1=E_ReadMotor(E_ORD_MA3);
            ValorOrden2=E_ReadMotor(E_ORD_MB3);
            break;
         case 5:
            ValorOrden1=E_ReadMotor(E_ORD_MA4);
            ValorOrden2=E_ReadMotor(E_ORD_MB4);
            break;
         case 6:
            ValorOrden1=E_ReadMotor(E_ORD_MA5);
            ValorOrden2=E_ReadMotor(E_ORD_MB5);
            break;
      }
      textcolor(BLACK);
      gotoxy(X_ORDEN1,Y_ORDEN1);
      cprintf("%d",ValorOrden1);
      gotoxy(X_ORDEN2,Y_ORDEN2);
      cprintf("%d",ValorOrden2);
   }
   else if(ActuadorActivo==7)   /* Leds */
   {  gotoxy(X_ORDEN1-12,Y_ORDEN1);
      cputs("ORDEN LEDS: ");
      ValorOrden1=E_ReadLeds();
      textcolor(BLACK);
      gotoxy(X_ORDEN1,Y_ORDEN1);
      cprintf("%d",ValorOrden1);
   }
   else if(ActuadorActivo==8)   /* Buzzer */
   {  gotoxy(X_ORDEN1-14,Y_ORDEN1);
      cputs("ORDEN BUZZER: ");
      ValorOrden1=E_ReadBuzzer();
      textcolor(BLACK);
      gotoxy(X_ORDEN1,Y_ORDEN1);
      cprintf("%d",ValorOrden1);
   }
}


/* ********************************************************************** */
/*                              IniPantalla                                */
/* ********************************************************************** */

void IniPantalla(void)
{  int i,j;

   lowvideo();
   textbackground(BLACK);
   i=1;

   gotoxy(1,i);
   cputs("ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ");
   cputs("ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»");
   i++;
   gotoxy(1,i);
   putch('°');
   gotoxy(80,i);
   putch('°');
   i++;
   gotoxy(1,i);
   cputs("ÌÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ");
   cputs("ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹");
   i++;
   for(j=0;j<VENTANA_EST_FISICO;j++)
   {  gotoxy(1,i);
      putch('°');
      gotoxy(80,i);
      putch('°');
      i++;
   }
   gotoxy(1,i);
   cputs("ÌÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ");
   cputs("ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹");
   i++;
   for(j=0;j<VENTANA_EST_ENLACE;j++)
   {   gotoxy(1,i);
```

```
   putch('°');
   gotoxy(80,i);
   putch('°');
   i++;
}
gotoxy(1,i);
cputs("ÌÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ");
cputs("ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹");
i++;
for(j=0;j<VENTANA_SENSORES;j++)
{   gotoxy(1,i);
   putch('°');
   gotoxy(80,i);
   putch('°');
   i++;
}

for(j=0;j<VENTANA_ORDENES;j++)
{   gotoxy(1,i);
   putch('°');
   gotoxy(80,i);
   putch('°');
   i++;
}
cputs("ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ");
cputs("ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼");

textbackground(BLACK);
i=1;

/* ********** Titulo ********** */
window(2,i+1,79,i+VENTANA_TITULO);
clrscr();
textcolor(GREEN);
gotoxy(23,1);
cputs("APLICACION DEL PROTOCOLO GENGHIS-PC");
i+=2;

textbackground(CYAN);

/* ********** Estado de nivel fisico ********** */

window(2,i+1,79,i+VENTANA_EST_FISICO);
clrscr();
textcolor(RED);
gotoxy(6,2);
cprintf("ERRORES DE OVERRUN :");
gotoxy(6,3);
cprintf("ERRORES DE PARIDAD :");
gotoxy(6,4);
cprintf("ERRORES DE TRAMA   :");
gotoxy(43,2);
cprintf("ERRORES BREAKLINES :");
gotoxy(43,3);
cprintf("ERR. OVERRUN BUFFER:");
i=i+VENTANA_EST_FISICO+1;


/* ********** Estado de nivel de enlace ********** */
window(2,i+1,79,i+VENTANA_EST_ENLACE);
clrscr();
textcolor(BLUE);
gotoxy(6,2);
cputs("TRAMAS DETECTADAS  :");
gotoxy(6,3);
cputs("TRAMAS CORRECTAS   :");
gotoxy(43,2);
cputs("ERRORES TIPO TRAMA :");
gotoxy(6,4);
cputs("TRAMAS INCORRECTAS :");
gotoxy(6,5);
cputs("ERRORES DE TIMEOUT :");
gotoxy(43,3);
cputs("ERRORES CHECKSUM   :");
gotoxy(43,4);
cputs("ERRORES PET. DATOS :");
gotoxy(43,5);
cputs("RATIO (tramas/sec) :");

gotoxy(6,7);
cputs("ORDEN NO ENVIADA EN CICLO:");
gotoxy(6,8);
```

```
cputs("ERROR AL ENVIAR ORDENES  :");
gotoxy(6,9);
cputs("ERROR OVERRUN DE SENSORES:");
gotoxy(6,10);
cputs("ERROR OVERRUN DE ORDENES :");
gotoxy(43,7);
cputs("TIMEOUT SIN RECIBIR NADA :");
gotoxy(43,8);
cputs("TIMEOUT PETICION DE DATOS:");
gotoxy(43,9);
cputs("NUMERO TRAMAS NO OPTIMO  :");
i=i+VENTANA_EST_ENLACE+1;


/* ********** Estado de los sensores ********** */
window(2,i+1,79,i+VENTANA_SENSORES);
clrscr();
textcolor(WHITE);
gotoxy(6,2);
cputs("M.ALFA0:");
gotoxy(6,3);
cputs("M.ALFA1:");
gotoxy(6,4);
cputs("M.ALFA2:");
gotoxy(6,5);
cputs("M.ALFA3:");
gotoxy(6,6);
cputs("M.ALFA4:");
gotoxy(6,7);
cputs("M.ALFA5:");

gotoxy(20,2);
cputs("M.BETA0:");
gotoxy(20,3);
cputs("M.BETA1:");
gotoxy(20,4);
cputs("M.BETA2:");
gotoxy(20,5);
cputs("M.BETA3:");
gotoxy(20,6);
cputs("M.BETA4:");
gotoxy(20,7);
cputs("M.BETA5:");

gotoxy(34,2);
cputs("P.TACTO1:");
gotoxy(34,3);
cputs("P.TACTO2:");
gotoxy(34,4);
cputs("P.TACTO3:");
gotoxy(34,5);
cputs("F.TACTO1:");
gotoxy(34,6);
cputs("F.TACTO2:");
gotoxy(34,7);
cputs("F.TACTO3:");

gotoxy(51,2);
cputs("PIRO1:");
gotoxy(51,3);
cputs("PIRO2:");
gotoxy(51,4);
cputs("PIRO3:");
gotoxy(51,5);
cputs("PIRO4:");
gotoxy(51,6);
cputs("PIRO5:");

gotoxy(63,2);
cputs("INFRARR:");
gotoxy(63,3);
cputs("INCLMTR:");
gotoxy(63,4);
cputs("BT.ELEC:");
gotoxy(63,5);
cputs("BT.MOTR:");
gotoxy(63,6);
cputs("W. LEFT:");
gotoxy(63,7);
cputs("W.RIGHT:");
i=i+VENTANA_SENSORES;
```

```
   /* ********** Estado de las ordenes ********** */
   window(2,i+1,79,i+VENTANA_ORDENES);
   clrscr();
   DibujarRobot(0);

   /* La ventana activa para escritura o lectura solo es la de ordenes */
}


/* ************************************************************************ */
/*                              RefrescoPantalla                            */
/* ************************************************************************ */

void RefrescoPantalla(int ActuadorActivo,int RefrescoRobot)
{  unsigned long valor;
   int valor2,i;
   extern float Ratio;

   textcolor(BLACK);
   i=1+VENTANA_TITULO+1;

   /* ********** Estado de nivel fisico ********** */
   window(2,i+1,79,i+VENTANA_EST_FISICO);
   gotoxy(27,2);
   E_State(F_ERROR_OVERRUN,&valor);
   cprintf("%010lu",valor);
   gotoxy(27,3);
   E_State(F_ERROR_PARIDAD,&valor);
   cprintf("%010lu",valor);
   gotoxy(27,4);
   E_State(F_ERROR_TRAMA,&valor);
   cprintf("%010lu",valor);
   gotoxy(65,2);
   E_State(F_ERROR_BREAKLINE,&valor);
   cprintf("%010lu",valor);
   gotoxy(65,3);
   E_State(F_ERROR_OVERRUN_BUFFER,&valor);
   cprintf("%010lu",valor);
   i=i+VENTANA_EST_FISICO+1;


   /* ********** Estado de nivel de enlace ********** */
   window(2,i+1,79,i+VENTANA_EST_ENLACE);
   gotoxy(27,2);
   E_State(E_TRAMAS_DETECTADAS,&valor);
   cprintf("%010lu",valor);
   gotoxy(27,3);
   E_State(E_TRAMAS_CORRECTAS,&valor);
   cprintf("%010lu",valor);
   gotoxy(27,4);
   E_State(E_TRAMAS_ERRONEAS,&valor);
   cprintf("%010lu",valor);
   gotoxy(27,5);
   E_State(E_NUM_TOUT,&valor);
   cprintf("%010lu",valor);
   gotoxy(65,2);
   E_State(E_ERRORES_TIPO_TRAMA,&valor);
   cprintf("%010lu",valor);
   gotoxy(65,3);
   E_State(E_ERRORES_CHECKSUM,&valor);
   cprintf("%010lu",valor);
   gotoxy(65,4);
   E_State(E_NUM_TOUT_PET_DAT,&valor);
   cprintf("%010lu",valor);
   gotoxy(65,5);
   cprintf("%010f",Ratio);

   gotoxy(33,7);
   E_EstCom(E_IND_ENV_CICLO,&valor2);
   cprintf("%ld",valor2);
   gotoxy(33,8);
   E_EstCom(E_ERR_ENV,&valor2);
   cprintf("%ld",valor2);
   gotoxy(33,9);
   E_EstCom(E_OVERR_SENSORES,&valor2);
   cprintf("%ld",valor2);
   gotoxy(33,10);
   E_EstCom(E_OVERR_ORDENES,&valor2);
   cprintf("%ld",valor2);
   gotoxy(70,7);
   E_EstCom(E_ERR_TIMEOUT,&valor2);
   cprintf("%ld",valor2);
```

```
gotoxy(70,8);
E_EstCom(E_ERR_PET_DAT,&valor2);
cprintf("%ld",valor2);
gotoxy(70,9);
E_EstCom(E_IND_NO_OPTIMO,&valor2);
cprintf("%ld",valor2);
i=i+VENTANA_EST_ENLACE+1;


/* ********** Estado de los sensores ********** */
window(2,i+1,79,i+VENTANA_SENSORES);
if(E_ActSensor(E_ACT_MTALFA))
   textcolor(YELLOW);
else
   textcolor(BLACK);
gotoxy(15,2);
cprintf("%03u",E_ReadSensor(E_SENS_MA0));
gotoxy(15,3);
cprintf("%03u",E_ReadSensor(E_SENS_MA1));
gotoxy(15,4);
cprintf("%03u",E_ReadSensor(E_SENS_MA2));
gotoxy(15,5);
cprintf("%03u",E_ReadSensor(E_SENS_MA3));
gotoxy(15,6);
cprintf("%03u",E_ReadSensor(E_SENS_MA4));
gotoxy(15,7);
cprintf("%03u",E_ReadSensor(E_SENS_MA5));

if(E_ActSensor(E_ACT_MTBETA))
   textcolor(YELLOW);
else
   textcolor(BLACK);
gotoxy(29,2);
cprintf("%03u",E_ReadSensor(E_SENS_MB0));
gotoxy(29,3);
cprintf("%03u",E_ReadSensor(E_SENS_MB1));
gotoxy(29,4);
cprintf("%03u",E_ReadSensor(E_SENS_MB2));
gotoxy(29,5);
cprintf("%03u",E_ReadSensor(E_SENS_MB3));
gotoxy(29,6);
cprintf("%03u",E_ReadSensor(E_SENS_MB4));
gotoxy(29,7);
cprintf("%03u",E_ReadSensor(E_SENS_MB5));

if(E_ActSensor(E_ACT_PTACTO))
   textcolor(YELLOW);
else
   textcolor(BLACK);
gotoxy(44,2);
cprintf("%03u",E_ReadSensor(E_SENS_PT1));
gotoxy(44,3);
cprintf("%03u",E_ReadSensor(E_SENS_PT2));
gotoxy(44,4);
cprintf("%03u",E_ReadSensor(E_SENS_PT3));

if(E_ActSensor(E_ACT_FTACTO))
   textcolor(YELLOW);
else
   textcolor(BLACK);
gotoxy(44,5)   ;
cprintf("%03u",E_ReadSensor(E_SENS_FT1));
gotoxy(44,6);
cprintf("%03u",E_ReadSensor(E_SENS_FT2));
gotoxy(44,7);
cprintf("%03u",E_ReadSensor(E_SENS_FT3));

if((E_ActSensor(E_ACT_PIR_23)) || (E_ActSensor(E_ACT_PIR145)))
   textcolor(YELLOW);
else
   textcolor(BLACK);
gotoxy(58,2);
cprintf("%03u",E_ReadSensor(E_SENS_PR1));
gotoxy(58,3);
cprintf("%03u",E_ReadSensor(E_SENS_PR2));
gotoxy(58,4);
cprintf("%03u",E_ReadSensor(E_SENS_PR3));
gotoxy(58,5);
cprintf("%03u",E_ReadSensor(E_SENS_PR4));
gotoxy(58,6);
cprintf("%03u",E_ReadSensor(E_SENS_PR5));
```

```
   if(E_ActSensor(E_ACT_INFRAR))
      textcolor(YELLOW);
   else
      textcolor(BLACK);
   gotoxy(72,2);
   cprintf("%03u",E_ReadSensor(E_SENS_IFR));

   if(E_ActSensor(E_ACT_INCLIN))
      textcolor(YELLOW);
   else
      textcolor(BLACK);
   gotoxy(72,3);
   cprintf("%03u",E_ReadSensor(E_SENS_INC));

   if((E_ActSensor(E_ACT_BTELEC)) || (E_ActSensor(E_ACT_BTMOTR)))
      textcolor(YELLOW);
   else
      textcolor(BLACK);
   gotoxy(72,4);
   cprintf("%03u",E_ReadSensor(E_SENS_BTE));
   gotoxy(72,5);
   cprintf("%03u",E_ReadSensor(E_SENS_BTM));

   if(E_ActSensor(E_ACT_WHISKR))
      textcolor(YELLOW);
   else
      textcolor(BLACK);
   gotoxy(72,6);
   cprintf("%03u",E_ReadSensor(E_SENS_WKL));
   gotoxy(72,7);
   cprintf("%03u",E_ReadSensor(E_SENS_WKR));
   i=i+VENTANA_SENSORES;


   /* ********** Estado de ordenes ********** */
   window(2,i+1,79,i+VENTANA_ORDENES);
   if(RefrescoRobot)
      DibujarRobot(ActuadorActivo);

   /* La ventana activa para escritura o lectura solo es la de ordenes */
}
```

Fichero "main.c"

```
#include <conio.h>
#include <time.h>
#include "es.h"
#include "enlace.h"

/* Definicion de codigos de teclas */
#define TECLA_F1  0x3B00
#define SHIFT_F1  0x5400
#define TECLA_F2  0x3C00
#define SHIFT_F2  0x5500
#define TECLA_F3  0x3D00
#define SHIFT_F3  0x5600
#define TECLA_F4  0x3E00
#define SHIFT_F4  0x5700
#define TECLA_F5  0x3F00
#define SHIFT_F5  0x5800
#define TECLA_F6  0x4000
#define SHIFT_F6  0x5900
#define TECLA_F7  0x4100
#define SHIFT_F7  0x5A00
#define TECLA_F8  0x4200
#define SHIFT_F8  0x5B00
#define TECLA_F9  0x4300
#define SHIFT_F9  0x5C00
#define TECLA_F10 0x4400
#define SHIFT_F10 0x5D00
#define TECLA_F11 0x4500
#define SHIFT_F11 0x5E00
#define TECLA_F12 0x4600
#define SHIFT_F12 0x5F00
#define CURSOR_ARRIBA    0x4800
#define CURSOR_ABAJO     0x5000
```

```
#define CURSOR_DERECHA   0x4D00
#define CURSOR_IZQUIERDA 0x4B00
#define TECLA_ESC 27
#define RETURN    13
#define TABULADOR 9

#define ACTUADORES 14

float Ratio=0.0;



/************************ Programa principal ***************************/

void main()
{  int fin=0,c,primero=1,NuevosDatos,ActuadorActivo=0,RefrescoRobot=0,Orden;
   int OrdenCursor[ACTUADORES],i,Enviar=0;
   time_t tinicial,tfinal;
   unsigned long NumTramas;

   OpenPantalla();
   if(E_Open()==F_NO_ERROR)
   {  IniPantalla();

      for(i=0;i<ACTUADORES;i++)
         OrdenCursor[i]=0;

      while(!fin)
      {  if(kbhit())
         {  c=getchExt();
            switch(c)
            {  case TECLA_ESC:
                  fin=1;
                  break;
               case RETURN:
                  E_Write();
                  break;
               case TECLA_F10:
                  E_WriteQueDatos(E_ALL_DAT,1);
                  break;
               case TECLA_F1:
                  E_WriteQueDatos(E_DAT_MTA,1);
                  break;
               case TECLA_F2:
                  E_WriteQueDatos(E_DAT_MTB,1);
                  break;
               case TECLA_F3:
                  E_WriteQueDatos(E_DAT_PTC,1);
                  break;
               case TECLA_F4:
                  E_WriteQueDatos(E_DAT_FTC,1);
                  break;
               case TECLA_F5:
                  E_WriteQueDatos(E_DAT_PIR,1);
                  break;
               case TECLA_F6:
                  E_WriteQueDatos(E_DAT_INF,1);
                  break;
               case TECLA_F7:
                  E_WriteQueDatos(E_DAT_INC,1);
                  break;
               case TECLA_F8:
                  E_WriteQueDatos(E_DAT_BTR,1);
                  break;
               case TECLA_F9:
                  E_WriteQueDatos(E_DAT_WSK,1);
                  break;
               case SHIFT_F10:
                  E_WriteQueDatos(E_ALL_DAT,0);
                  break;
               case SHIFT_F1:
                  E_WriteQueDatos(E_DAT_MTA,0);
                  break;
               case SHIFT_F2:
                  E_WriteQueDatos(E_DAT_MTB,0);
                  break;
               case SHIFT_F3:
                  E_WriteQueDatos(E_DAT_PTC,0);
                  break;
               case SHIFT_F4:
                  E_WriteQueDatos(E_DAT_FTC,0);
                  break;
```

```
case SHIFT_F5:
   E_WriteQueDatos(E_DAT_PIR,0);
   break;
case SHIFT_F6:
   E_WriteQueDatos(E_DAT_INF,0);
   break;
case SHIFT_F7:
   E_WriteQueDatos(E_DAT_INC,0);
   break;
case SHIFT_F8:
   E_WriteQueDatos(E_DAT_BTR,0);
   break;
case SHIFT_F9:
   E_WriteQueDatos(E_DAT_WSK,0);
   break;
case TABULADOR:
   ActuadorActivo++;
   ActuadorActivo%=9;
   RefrescoRobot=1;
   break;
case 'R':
case 'r':
   E_ResetRob();
   RefrescoRobot=1;
   break;
case 'A':
case 'a':
   if(ActuadorActivo>0 && ActuadorActivo<7)   /* Motores */
   {  do
      {   Orden=LeerNat(X_ORDEN1,Y_ORDEN1,2);
      }
      while(Orden>63);
      /* Si Orden==-1 no hay natural leido */
      if(Orden>=0)
      {  switch(ActuadorActivo)
         {  case 1:
               Orden=E_WriteMotor(E_ORD_MA0,Orden);
               break;
            case 2:
               Orden=E_WriteMotor(E_ORD_MA1,Orden);
               break;
            case 3:
               Orden=E_WriteMotor(E_ORD_MA2,Orden);
               break;
            case 4:
               Orden=E_WriteMotor(E_ORD_MA3,Orden);
               break;
            case 5:
               Orden=E_WriteMotor(E_ORD_MA4,Orden);
               break;
            case 6:
               Orden=E_WriteMotor(E_ORD_MA5,Orden);
            break;
         }
      }
      RefrescoRobot=1;
   }
   break;
case 'B':
case 'b':
   if(ActuadorActivo>0 && ActuadorActivo<7)   /* Motores */
   {  do
      {  Orden=LeerNat(X_ORDEN2,Y_ORDEN2,2);
      }
      while(Orden>63);
      /* Si Orden==-1 no hay natural leido */
      if(Orden>=0)
      {  switch(ActuadorActivo)
         {  case 1:
               Orden=E_WriteMotor(E_ORD_MB0,Orden);
               break;
            case 2:
               Orden=E_WriteMotor(E_ORD_MB1,Orden);
               break;
            case 3:
               Orden=E_WriteMotor(E_ORD_MB2,Orden);
               break;
            case 4:
               Orden=E_WriteMotor(E_ORD_MB3,Orden);
               break;
            case 5:
               Orden=E_WriteMotor(E_ORD_MB4,Orden);
```

```
                  break;
               case 6:
                  Orden=E_WriteMotor(E_ORD_MB5,Orden);
               break;
            }
         }
         RefrescoRobot=1;
      }
      break;
   case 'L':
   case 'l':
      if(ActuadorActivo==7)   /* Leds */
      {  do
         {  Orden=LeerNat(X_ORDEN1,Y_ORDEN1,3);
         }
         while(Orden>255);
         /* Si Orden==-1 no hay natural leido */
         if(Orden>=0)
            Orden=E_WriteLeds(Orden);
         RefrescoRobot=1;
      }
      break;
   case 'Z':
   case 'z':
      if(ActuadorActivo==8)   /* Buzzer */
      {  do
         {  Orden=LeerNat(X_ORDEN1,Y_ORDEN1,3);
         }
         while(Orden>255);
         /* Si Orden==-1 no hay natural leido */
         if(Orden>=0)
            Orden=E_WriteBuzzer(Orden);
         RefrescoRobot=1;
      }
      break;
   case CURSOR_DERECHA:
      if(ActuadorActivo>0)
      {  switch(ActuadorActivo)
         {  case 1:
               if(OrdenCursor[0]==0)
               {  Orden=E_ReadMotor(E_ORD_MA0);
                  if(Orden<63)
                  {  Orden++;
                     Orden=E_WriteMotor(E_ORD_MA0,Orden);
                     OrdenCursor[0]=1;
                  }
               }
               break;
            case 2:
               if(OrdenCursor[1]==0)
               {  Orden=E_ReadMotor(E_ORD_MA1);
                  if(Orden<63)
                  {  Orden++;
                     Orden=E_WriteMotor(E_ORD_MA1,Orden);
                     OrdenCursor[1]=1;
                  }
               }
               break;
            case 3:
               if(OrdenCursor[2]==0)
               {  Orden=E_ReadMotor(E_ORD_MA2);
                  if(Orden<63)
                  {  Orden++;
                     Orden=E_WriteMotor(E_ORD_MA2,Orden);
                     OrdenCursor[2]=1;
                  }
               }
               break;
            case 4:
               if(OrdenCursor[3]==0)
               {  Orden=E_ReadMotor(E_ORD_MA3);
                  if(Orden<63)
                  {  Orden++;
                     Orden=E_WriteMotor(E_ORD_MA3,Orden);
                     OrdenCursor[3]=1;
                  }
               }
               break;
            case 5:
               if(OrdenCursor[4]==0)
               {  Orden=E_ReadMotor(E_ORD_MA4);
                  if(Orden<63)
```

```
                {  Orden++;
                   Orden=E_WriteMotor(E_ORD_MA4,Orden);
                   OrdenCursor[4]=1;
                }
             }
             break;
          case 6:
             if(OrdenCursor[5]==0)
             {  Orden=E_ReadMotor(E_ORD_MA5);
                if(Orden<63)
                {  Orden++;
                   Orden=E_WriteMotor(E_ORD_MA5,Orden);
                   OrdenCursor[5]=1;
                }
             }
             break;
          case 7:
             if(OrdenCursor[12]==0)
             {  Orden=E_ReadLeds();
                if(Orden<255)
                {  Orden++;
                   Orden=E_WriteLeds(Orden);
                   OrdenCursor[12]=1;
                }
             }
             break;
          case 8:
             if(OrdenCursor[13]==0)
             {  Orden=E_ReadBuzzer();
                if(Orden<255)
                {  Orden++;
                   Orden=E_WriteBuzzer(Orden);
                   OrdenCursor[13]=1;
                }
             }
             break;
       }
    }
    break;
 case CURSOR_IZQUIERDA:
    if(ActuadorActivo>0)
    {   switch(ActuadorActivo)
       {  case 1:
             if(OrdenCursor[0]==0)
             {  Orden=E_ReadMotor(E_ORD_MA0);
                if(Orden>0)
                {  Orden--;
                   Orden=E_WriteMotor(E_ORD_MA0,Orden);
                   OrdenCursor[0]=1;
                }
             }
             break;
          case 2:
             if(OrdenCursor[1]==0)
             {  Orden=E_ReadMotor(E_ORD_MA1);
                if(Orden>0)
                {  Orden--;
                   Orden=E_WriteMotor(E_ORD_MA1,Orden);
                   OrdenCursor[1]=1;
                }
             }
             break;
          case 3:
             if(OrdenCursor[2]==0)
             {  Orden=E_ReadMotor(E_ORD_MA2);
                if(Orden>0)
                {  Orden--;
                   Orden=E_WriteMotor(E_ORD_MA2,Orden);
                   OrdenCursor[2]=1;
                }
             }
             break;
          case 4:
             if(OrdenCursor[3]==0)
             {  Orden=E_ReadMotor(E_ORD_MA3);
                if(Orden>0)
                {  Orden--;
                   Orden=E_WriteMotor(E_ORD_MA3,Orden);
                   OrdenCursor[3]=1;
                }
             }
             break;
```

```
            case 5:
               if(OrdenCursor[4]==0)
               {  Orden=E_ReadMotor(E_ORD_MA4);
                  if(Orden>0)
                  {  Orden--;
                     Orden=E_WriteMotor(E_ORD_MA4,Orden);
                     OrdenCursor[4]=1;
                  }
               }
               break;
            case 6:
               if(OrdenCursor[5]==0)
               {  Orden=E_ReadMotor(E_ORD_MA5);
                  if(Orden>0)
                  {  Orden--;
                     Orden=E_WriteMotor(E_ORD_MA5,Orden);
                     OrdenCursor[5]=1;
                  }
               }
               break;
            case 7:
               if(OrdenCursor[12]==0)
               {  Orden=E_ReadLeds();
                  if(Orden>0)
                  {  Orden--;
                     Orden=E_WriteLeds(Orden);
                     OrdenCursor[12]=1;
                  }
               }
               break;
            case 8:
               if(OrdenCursor[13]==0)
               {  Orden=E_ReadBuzzer();
                  if(Orden>0)
                  {  Orden--;
                     Orden=E_WriteBuzzer(Orden);
                     OrdenCursor[13]=1;
                  }
               }
               break;
         }
      }
      break;
   case CURSOR_ARRIBA:
      if(ActuadorActivo>0 && ActuadorActivo<7)
      {  switch(ActuadorActivo)
         {  case 1:
               if(OrdenCursor[6]==0)
               {  Orden=E_ReadMotor(E_ORD_MB0);
                  if(Orden<63)
                  {  Orden++;
                     Orden=E_WriteMotor(E_ORD_MB0,Orden);
                     OrdenCursor[6]=1;
                  }
               }
               break;
            case 2:
               if(OrdenCursor[7]==0)
               {  Orden=E_ReadMotor(E_ORD_MB1);
                  if(Orden<63)
                  {  Orden++;
                     Orden=E_WriteMotor(E_ORD_MB1,Orden);
                     OrdenCursor[7]=1;
                  }
               }
               break;
            case 3:
               if(OrdenCursor[8]==0)
               {  Orden=E_ReadMotor(E_ORD_MB2);
                  if(Orden<63)
                  {  Orden++;
                     Orden=E_WriteMotor(E_ORD_MB2,Orden);
                     OrdenCursor[8]=1;
                  }
               }
               break;
            case 4:
               if(OrdenCursor[9]==0)
               {  Orden=E_ReadMotor(E_ORD_MB3);
                  if(Orden<63)
                  {  Orden++;
                     Orden=E_WriteMotor(E_ORD_MB3,Orden);
```

```
                  OrdenCursor[9]=1;
               }
            }
            break;
         case 5:
            if(OrdenCursor[10]==0)
            {  Orden=E_ReadMotor(E_ORD_MB4);
               if(Orden<63)
               {  Orden++;
                  Orden=E_WriteMotor(E_ORD_MB4,Orden);
                  OrdenCursor[10]=1;
               }
            }
            break;
         case 6:
            if(OrdenCursor[11]==0)
            {  Orden=E_ReadMotor(E_ORD_MB5);
               if(Orden<63)
               {  Orden++;
                  Orden=E_WriteMotor(E_ORD_MB5,Orden);
                  OrdenCursor[11]=1;
               }
            }
            break;
      }
   }
   break;
case CURSOR_ABAJO:
   if(ActuadorActivo>0 && ActuadorActivo<7)
   {  switch(ActuadorActivo)
      {  case 1:
            if(OrdenCursor[6]==0)
            {  Orden=E_ReadMotor(E_ORD_MB0);
               if(Orden>0)
               {  Orden--;
                  Orden=E_WriteMotor(E_ORD_MB0,Orden);
                  OrdenCursor[6]=1;
               }
            }
            break;
         case 2:
            if(OrdenCursor[7]==0)
            {  Orden=E_ReadMotor(E_ORD_MB1);
               if(Orden>0)
               {  Orden--;
                  Orden=E_WriteMotor(E_ORD_MB1,Orden);
                  OrdenCursor[7]=1;
               }
            }
            break;
         case 3:
            if(OrdenCursor[8]==0)
            {  Orden=E_ReadMotor(E_ORD_MB2);
               if(Orden>0)
               {  Orden--;
                  Orden=E_WriteMotor(E_ORD_MB2,Orden);
                  OrdenCursor[8]=1;
               }
            }
            break;
         case 4:
            if(OrdenCursor[9]==0)
            {  Orden=E_ReadMotor(E_ORD_MB3);
               if(Orden>0)
               {  Orden--;
                  Orden=E_WriteMotor(E_ORD_MB3,Orden);
                  OrdenCursor[9]=1;
               }
            }
            break;
         case 5:
            if(OrdenCursor[10]==0)
            {  Orden=E_ReadMotor(E_ORD_MB4);
               if(Orden>0)
               {  Orden--;
                  Orden=E_WriteMotor(E_ORD_MB4,Orden);
                  OrdenCursor[10]=1;
               }
            }
            break;
         case 6:
            if(OrdenCursor[11]==0)
```

```
                        {  Orden=E_ReadMotor(E_ORD_MB5);
                           if(Orden>0)
                           {  Orden--;
                              Orden=E_WriteMotor(E_ORD_MB5,Orden);
                              OrdenCursor[11]=1;
                           }
                        }
                        break;
                     }
                  }
                  break;
            }
         }
         if(fin)
            continue;

         E_Read(&NuevosDatos);
         if(NuevosDatos)
         {  for(i=0;i<ACTUADORES;i++)
            {  if(OrdenCursor[i]==1)
                  Enviar=1;
               OrdenCursor[i]=0;
            }

            if(Enviar)
            {  E_Write();
               Enviar=0;
               RefrescoRobot=1;
            }

            if(primero==1)
            {  time(&tinicial);
               primero=0;
            }
            time(&tfinal);
            if((tfinal-tinicial)>0)
            {  E_State(E_TRAMAS_DETECTADAS,&NumTramas);
               Ratio=(float)NumTramas/(tfinal-tinicial);
            }
         }
         RefrescoPantalla(ActuadorActivo,RefrescoRobot);
         RefrescoRobot=0;
      }
      E_Close();
   }
   else
   {  clrscr();
      cprintf("Error Inicializando el protocolo");
      getch();
   }
   ClosePantalla();
}
```

ANEXO 8. PROBLEMAS CON EL SERVO

Como se explica en el punto 3.2.2, el microcomputador *SERVO* es el encargado, entre otras funciones, del *timming* (marca la duración de cada ciclo) y de la comunicación con el resto de *micros*.

Inicialmente no se pensó en que fuera necesario modificar el código de este *micro* y se confió en el buen funcionamiento de éste. Desafortunadamente, investigando el origen de comportamientos extraños del código que se estaba implementado en el *MAIN*, se comprobó que las causas de los problemas eran atribuibles a que el *SERVO* no presentaba el comportamiento esperado.

Principalmente se han detectado dos problemas, uno en cuanto a la comunicación con el *micro MAIN* y otro en cuanto al *timming*. El primero se detectó al principio de la implementación del *Nivel de Enlace* en el robot, justo al añadir al *MAIN* el código para la comunicación con el *SERVO*. El segundo se detectó mientras se realizaban pruebas del correcto funcionamiento del protocolo, ya habiendo finalizado las implementaciones en el robot y el *PC*.

Como se ha visto la comunicación entre *SERVO* y *MAIN* se realiza mediante el sistema de *SPI*, de manera que el *SERVO* actúa como maestro y el *MAIN* como esclavo. Cuando el *SERVO* desea comunicarse y envía el primer carácter, en el *MAIN* se genera una interrupción de *SPI*.

El problema de comunicación con el *MAIN* consistía en que, en ciertas ocasiones, el *SERVO* al querer comunicarse con el *MAIN* no recibía respuesta de éste y abortaba la comunicación.
En un primer momento se pensó que el origen de este problema era que la interrupción de *SPI* tardaba demasiado en despertarse porqué coincidía con momentos en que se estaba ejecutando una interrupción de línea serie y por lo tanto no podía ejecutarse el código de la *SPI* hasta el final de la rutina de línea serie (por defecto cuando se despierta una interrupción las interrupciones están inhibidas si no se han permitido expresamente mediante la instrucción correspondiente).
Por lo tanto, la primera solución que se adoptó fue permitir las interrupciones dentro de la interrupción de línea serie lo antes posible y durante la mayor zona de código posible. Esto provocó una mejora en cuanto a que el número de errores descendió pero no solucionó el problema por completo.
Tras un estudio más detallado del código y varias pruebas, se llegó a la conclusión de que el *SERVO* no se esperaba suficiente tiempo después de enviar el primer carácter. Así en ciertas ocasiones en que la interrupción de *SPI* tardaba (por el motivo que fuera) un poco más en despertarse, la comunicación fallaba porqué el *SERVO* no recibía respuesta antes del tiempo esperado.
La solución fue incrementar un poco el tiempo de espera del *SERVO* después de enviar el primer carácter, concretamente 25 ciclos (aunque se vio que se puede aumentar hasta un máximo de 1165 ciclos). Con esto se corrige el problema y se consigue el comportamiento esperado (después de pruebas dejando que los *micros* se comunicaran durante bastante tiempo no se ha producido ningún error de este tipo).

Como vimos, el micro *SERVO* realizaba diferentes tareas durante un ciclo (20 milisegundos). Concretamente, divide el ciclo en un total de 17 estados (numerados de 0 a 16) de una duración determinada cada uno. El cambio de un estado a otro se controla mediante la interrupción de *timmer*.
Inicialmente el programa principal del *SERVO* está en un bucle esperando la actualización de un *flag* (*run-motors-flag*), de manera que cuando en el estado 0 se activa ese *flag*, se pasa a ejecutar el código para enviar las órdenes a los motores y se entra en otro bucle en espera de la activación de un segundo *flag* (*communicate-flag*). Este último se activa en el estado 1 permitiendo la ejecución del código para la comunicación con el resto de *micros*.

El comportamiento anómalo que se detectó era que, en ocasiones, la orden enviada al robot no llegaba completa (al *MAIN*) dentro del ciclo (en un ciclo la trama debería llegar completa). En cambio, como era un error poco frecuente y el MAIN se ha programado para que en esos casos se envíe la orden del ciclo anterior, a simple vista esto no se notaba y hasta ese momento daba la impresión de que todo funcionaba correctamente (el problema se detectó tras una modificación del código).
Tras descartar varias hipótesis y no encontrar una explicación satisfactoria, se decidió modificar los programas para calcular el tiempo que transcurría en el *MAIN* entre dos comunicaciones del *SERVO* y enviar ese valor al PC para poder visualizarlo por pantalla.

Con esto se comprobó que el *SERVO* se comunicaba casi siempre casa 20 milisegundos pero que en ciertas ocasiones, y de manera acíclica, se comunicaba mucho antes de lo debido. Esto, evidentemente, provocaba que la trama de órdenes no pudiera llegar al robot dentro del ciclo.
Después de un largo estudio del código y realización de pruebas y modificaciones buscando la posible causa, se detectó el origen del problema. El código está pensado para que se empiece en el estado 0, en el cual se activa el *run-motors-flag* que permite enviar órdenes a los motores, y a continuación, después de 4 milisegundos, se pase al estado 1 (en el que se activa el *communicate-flag*) permitiendo la comunicación con el resto de *micros*. El problema residía en que una inicialización errónea según el funcionamiento de la interrupción de timmer (que controla el cambio de estado), provocaba que el código correspondiente al estado 1 para la comunicación con el resto de *micros* se ejecutara antes de tiempo la primera vez y no se respetara el tiempo de duración del estado 0. Como desde el principio ya no se respetaban los tiempos de los estados a partir de este momento todo funcionaba sin una correcta sincronización, de manera que a veces se comunicaba correctamente pero otras lo volvía a hacer antes de tiempo.
La solución fue corregir la inicialización de manera que el código del primer estado en ejecutarse fuera el del estado 1 y los tiempos de los estados se respetaran correctamente, manteniendo la sincronización. Los cambios efectuados solucionan el problema y el funcionamiento del *SERVO* tras las modificaciones parece totalmente correcto.

La modificación del código realizada para solucionar el segundo de los problemas tiene un efecto colateral ya que el valor de las lecturas del nivel de baterías cambia. Esto se debe a que la lectura del nivel de baterías de motores (que varía considerablemente cuando estos se mueven) se realiza en el *MAIN* un tiempo determinado después de que se haya realizado la comunicación con el *SERVO*. Antes de las modificaciones, como el *SERVO* no esperaba los 4 milisegundos que debe durar el estado 0 y rápidamente pasaba a comunicarse, el muestreo del nivel de batería de motores se realizaba en un momento determinado y no constante por los problemas de sincronización, pero como después de las modificaciones la espera en el estado 0 se realiza correctamente el momento del muestreo del valor de la batería de motores se desplaza y se obtienen pequeñas diferencias en los valores obtenidos.
El efecto de esto fue que se alcanzaba el límite del valor de batería de motores (hay un tope mínimo de seguridad, de manera que si el valor de la batería de motores es menor que este valor por un consumo excesivo, los motores se desconectan para evitar dañarlos) y los motores se desconectaban de vez en cuando imposibilitando que el robot caminara correctamente. La solución para este problema ha sido modificar ligeramente este valor mínimo del nivel de baterías, bajándolo en dos unidades (sobre un valor normal de 100) para permitir un límite de consumo mayor de los motores y que no se desconecten ante las nuevas lecturas que se obtienen. Esta pequeña modificación no será ningún problema en cuanto a la seguridad de los motores.

A continuación se incluye el código fuente del *SERVO* tras las modificaciones realizadas:

```
;;; Genghis servo code for the motors, force sensors, and communications
;;; Copyright, Artificial Creatures, Inc., 1992
;;; Copyright, IS Robotics, Inc., 1993,1994
;;; Copyrignt, J.M.Porta Pleite & David Lora Bartolome, 1999

;;; ***History***
;;;
;;; 22-Jun-92 23:26:04  Created.
;;; 21-Sep-92 22:59:35  Modified for Genghis.
;;; 11-Nov-92 11:21:25  Belgian changes
;;; 11-Nov-92 12:32:00  Added serial debugger support - jlj
;;; 15-Jan-93 11:31:43  Added early support for `status' processor - jlj
;;; 21-Jan-93 15:07:59  Fixed error is force measurement.
;;; 31-Jan-93 14:22:27  Added battery down/don't run motors - jlj
;;; 25-Feb-93 21:44:48  Bullet proofed above fix
;;; 12-May-93 21:03:09  Made statdummies copy over positions 1 & 2.
;;; 29-Jun-93 15:06:50  jlj: Made statdumies copy to irdummies
;;; 13-Oct-93 11:17:10  cjr: cleaned up and improved fsr interface
;;; 28-May-99 13:00:00  dlb: detected and corrected important bug in initial state and cleaned
up.
;;;
;;; ***End history***

;;; there are potentially 16 motors to servo and 12 force readings to be had.
;;;
;;; genhis only receives positions for 12 motors from the BL processor
;;;
;;;

;;; port assignments:
;;;     port B  0-7     servo outputs
```

```
;;;     port C  0-7     servo outputs
;;;     port E  0-7     force inputs
;;;     port A  0-3     force inputs
;;;             4       slave #1 = main
;;;             5       slave #2 = IR
;;;             6       combit #1
;;;             7       combit #2

;;; Genghis servo processor memory map:

;;; Servo sends these 21 values to the main proc each com cycle:
;;;
;;; 8C: Forces      [ 01 ] [ 02 ] \
;;; 8E              [ 03 ] [ 04 ]  \
;;; 90              [ 05 ] [ 06 ]   \  12 Force
;;; 92              [ 07 ] [ 08 ]   /   Measurements
;;; 94              [ 09 ] [ 10 ]  /
;;; 96              [ 11 ] [ 12 ] /
;;; 98: Not Used    [ 13 ] [ 14 ] \ 4 bytes unused
;;; 9A             [ 15 ] [ 16 ] /
;;; 9C: Infrareds   [ 17 ] [ 18 ] > 2 Bytes from IR processor
;;; 9E: Status      [ 19 ] [ 20 ] \
;;; A0             [ 21 ]        / 3 Bytes from Status processor
;;;
;;;
;;;
;;; Servo receives these 21 values from the main proc each com cycle:
;;;
;;; AC: Positions   [ 01 ] [ 02 ] \
;;; AE              [ 03 ] [ 04 ]  \
;;; B0              [ 05 ] [ 06 ]   \
;;; B2              [ 07 ] [ 08 ]    \  16 position
;;; B4              [ 09 ] [ 10 ]    /   Commands
;;; B6              [ 11 ] [ 12 ]   /
;;; B8              [ 13 ] [ 14 ]  /
;;; BA              [ 15 ] [ 16 ] /
;;; BC: IR-comms    [ 17 ] [ 18 ] > 2 bytes to IR processor
;;; BE: Stat-comms  [ 19 ] [ 20 ] \
;;; C0              [ 21 ]        / 3 bytes to Status processor


;;; This arrangement means that the command:
;;;   (master-communicate main-addr 21 forces positions)
;;; exchanges all the data with the main processor each com cycle.

;;; The servo processor must also acuqire the data from the IR and Status
;;; processors, and provide storage for it.
;;;
;;;                       Bus addr    # bytes  Inbuf addr      Outbuf addr
;;;                      --------    -------  ----------      -----------
;;;  (master-communicate ir-addr       2       ir-commands     infrareds)
;;;  (master-communicate stat-addr     3       stat-commands   status)


(defprog gengservo
  :machine 6811
  :start #xf800
  :code (
  (comment)
  (comment "*************************************************")
  (comment "*************************************************")
  (comment)
  (comment "This is the servo processor code for Genghis.  In this")
  (comment "version the servo processor is the master processor for")
  (comment "communications.  It transfers 16 byte packets back and forth")
  (comment "to the slave processor (at 50Hz!!), which is the one running")
  (comment "the behavior language.")
  (comment)
  (comment "The basic cycle is 20ms long, in 17 different time slices.")
  (comment "Timing is controlled by output compare matches using output")
  (comment "compare register number one.  No actual use of output or input")
  (comment "signals happens on PORTA---we just use the OC1 interrupt.")
  (comment)


  (6811-register-addresses)
  (comment)
  (comment "constants")
  (=c stack #xFF (comment "stack pointer"))
  (comment)
  (comment "network")
  (comment "no interrupts, SPI enabled, CMOS outputs, master mdoe")
```

```
  (comment "clock idles high, clock phase high, (2) 1 MHz rate")
  (=c master-spcr #b01011100)
  (=c spi-idle #b11111000 (comment "select no slave"))
  (=c ddrd-master #b00011010)
  (=c ddrc-upper4 #b11111111 (comment "all 8 bits are outputs"))
  (=c tmsk1-flag #b10000000 (comment "enable oc1i"))
  (=c spi-int-wait 1 (comment "SPI constants"))
  (=c spi-byte-wait 1)
  (=c 4ms 8000 (comment "E clock cycles in 4 ms"))
  (=c 1ms 2000 (comment "E clock cycles in 2 ms"))
  (=c numslices    17  (comment "total number of time slices"))
  (=c main-addr    #xe0 (comment "address of main processor"))
  (=c ir-addr      #xd0 (comment "address of IR processor"))

                       (comment "status processor slave board is set to 4")
                       (comment " but bits are inverted and the two MSB's are flipped")
                       (comment " the two LSB's are ignored")
                       (comment " this works as follows: 0100 -> 1011 -> 0111 = x78")
  (=c stat-addr  #x78   (comment " address of status processor"))

;;; Note: On R-2 #xB8 => slave jumper 4, #x78 => slave jumper 5
;;;   on Genghis #xB8 => slave jumper 5, #x78 => slave jumper 4

  (comment)
  (comment)
  (comment "***************************************************")
  (comment "***************************************************")
  (comment)
  (comment "memory map")
  (comment)
  (=v table           #x00    (comment "control table"))
  (=v tempforces      #x80    (comment "temporary forces -- 12 readings"))

                              (comment "SPI buffers")
  (=v forces          #x8c    (comment "slave #1 out buffer -- 12 forces"))
  (=v funused         #x98    (comment "slave #1 out buffer --  4 unused bytes"))
  (=v infrareds       #x9c    (comment "slave #1 out buffer -- 2 infrareds, also slave #2 in"))
  (=v status          #x9e    (comment "slave #1 out buffer --  3 status, also status in"))
  (=v positions       #xac    (comment "slave #1 in  buffer -- 16 motor commands"))
  (=v ir-commands     #xbc    (comment "slave #1 in buffer -- 2 infrareds, also slave #2 out"))
  (=v stat-commands   #xbe    (comment "slave #1 in  buffer --  3 status, also status out"))

                              (comment "general variable locations")
  (=v timeslice       #xcc    (comment "current time slice"))
  (=v netbytecount    #xcd    (comment "network byte count"))
  (=v2 slavebuffaddr  #xce    (comment "slave buffer address"))
  (=v2 masterbuffaddr #xd0    (comment "master buffer address"))
  (=v run-motors-flag #xd2    (comment "flag to run the motors"))
  (=v communicate-flag#xd3    (comment "flag for communication"))


  (comment)
  (comment)
  (comment "***************************************************")
  (comment "***************************************************")
  (comment)
  (comment "start up code")
  (comment)
start
  (lds (! stack) (comment "initialize stack pointer"))
  (comment "first initialize SPI, then serial port")
  (ldx (! xbase) (comment "leave indx here forever"))
  (ldaa (! #x80))
  (staa (& pactl) (comment "make pa7 be an output, pa3 be an input"))
  (ldaa (! spi-idle))
  (staa (& porta))
  (ldaa (! master-spcr) (comment "initialize SPI network"))
  (staa (& spcr))
  (ldaa (! ddrd-master) (comment "configure port D"))
  (staa (& ddrd))
  (ldaa (! ddrc-upper4) (comment "configure port C"))
  (staa (& ddrc))
  (ldaa (! tmsk1-flag) (comment "enable OC1I"))
  (staa (& tmsk1))
  (clr run-motors-flag (comment "initialize the flags"))
  (clr communicate-flag)
  (clr timeslice)
  (ldd (& tcnt) (comment "pick up the clock"))
  (subd (! 1) (comment "maximize time before first interrupt"))
  (std (& toc1))

  (ldaa (! #x0B))
```

```
  (staa (& HPRIO)                (comment "Spi interrupcion de maxima prioridad"))

  (comment "Inicializacion de buffers inicial para comenzar por el estado en que el servo
            se comunica con el resto de micros.")
  (ldaa (! #x00))
  (iterate ((i 21)) (staa (+ forces i)))

  (cli                (comment "Global interrupt enable"))
  (comment "Salto al primer estado que sera el estado de comunicarse con los otros micros.
           Este debe ser el estado inicial ya que es el primer estado por el que empieza la
           interrupcion de reloj y el primero en el que se consultan valores de fuerzas.")
  (jmp waitcommunicate)


  (comment)
  (comment)
  (comment)
  (comment "***************************************************")
  (comment "***************************************************")
  (comment)
  (comment "processor main loop")
  (comment)

mainloop
  (tst run-motors-flag (comment "see if time to run the motors"))
  (beq mainloop)
  (clr run-motors-flag (comment "reset flag"))

  (comment "Don't run servos if battery power is off")
  (ldaa positions)
  (anda (! #b11000000))       ;Get highest order bits from position slot 0
  (cmpa (! #b01000000))       ;Motor on code is 01xxxxxx
  (bne waitcommunicate)       ;Skip writing to the motors if battery power is off
  (jsr output-table (comment "go run the motors"))

waitcommunicate
  (tst communicate-flag (comment "see if time to communicate"))
  (beq waitcommunicate)
  (clr communicate-flag (comment "reset flag"))

  (comment     "Get data from status processor")
  (master-communicate stat-addr 3 stat-commands status)

  (comment     "pick up IR values")
  (master-communicate ir-addr 2 ir-commands infrareds)

  (comment     "pass them on with forces, plus get positions")
  ;(master-communicate main-addr 21 forces positions)
  (jsr _communicate2)
  (!8 main-addr)
  (!8 21)
  (!16 forces)
  (!16 positions)

  (jsr compute-table   (comment "compute new table"))
  (clr run-motors-flag (comment "reset flag - BULLET PROOF"))
  (bra mainloop)

  (comment)
  (comment)
  (comment)
  (comment "***************************************************")
  (comment "***************************************************")
  (comment)
  (comment "compute the motor table")
  (comment)
  (comment "total of 1565 cycles = 783ms")
compute-table
  (comment "clear table")
  (clra)                                      ;2
  (clrb)                                      ;2
  (iterate ((v 64))                           ;512
           (std (+ table (ash v 1))))         ;4
  (comment "compute table beginnings")
  (iterate ((v 16)                            ;528
            (i (#x8000 #x4000 #x2000 #x1000
                #x0800 #x0400 #x0200 #x0100
                #x0080 #x0040 #x0020 #x0010
                #x0008 #x0004 #x0002 #x0001)))
           (ldab (+ positions v))             ;2
           (andb (! #x3f))                    ;2
           (aslb)                             ;2
```

```
          (ldy (! table))                     ;4
          (aby)                               ;4
          (ldd (! i))                         ;3
          (oraa (&y 0))                       ;5
          (orab (&y 1))                       ;5
          (std (&y 0)))                       ;6
  (comment "fill table")
  (clra)                                      ;2
  (clrb)                                      ;2
  (iterate ((v 64))                           ;512
          (oraa (+ table (ash v 1)))          ;2
          (orab (+ table (1+ (ash v 1))))     ;2
          (std (+ table (ash v 1))))          ;4
  (rts)                                       ;5

  (comment)
  (comment)
  (comment "***************************************************")
  (comment "***************************************************")
  (comment)
  (comment "output the motor table to the control ports")
  (comment)
  (comment "we step through the 64 entry table in 1ms (992ms),")
  (comment "then add an additional 1.008s, so that totally we are in")
  (comment "the range [1.008, 2.000] ms.")
  (comment)
  (comment "this means that the first loop should take 16 microseconds,")
  (comment "or 32 cycles.")
  (comment)
output-table
  (ldab (! 64))
  (ldy (! table))
m-lp
  (ldaa (&y 0))         ;5
  (staa (& portb))      ;4
  (ldaa (&y 1))         ;5
  (staa (& portc))      ;4
  (iny)                 ;4
  (iny)                 ;4
  (decb)                ;2
  (bne m-lp)            ;3
  (comment)
  (comment "offset loop -- should be 1.008 microseconds = 2016 = 7*288 cycles")
  (ldaa (! 144))
o-lp
  (nop)                 ;2
  (deca)                ;2
  (bne o-lp)            ;3
  (ldaa (! 144))
o-lp2
  (nop)                 ;2
  (deca)                ;2
  (bne o-lp2)           ;3
  (staa (& portb))
  (staa (& portc))
  (rts)

  (comment)
  (comment)
  (comment "***************************************************")
  (comment "***************************************************")
  (comment)
  (comment "communications interface")
  (comment)
_communicate
  (puly (comment "get return address = data block"))
  (ldaa (&y 0) (comment "pick up slave address"))
  (staa (& porta) (comment "select the appropriate slave"))
  (ldaa (& spsr) (comment "clear out the SPSR"))
  (ldaa (&y 1) (comment "pick up byte count"))
  (staa (& spdr))
  (staa netbytecount)
  (ldd (&y 2) (comment "pick up master buffer address"))
  (std masterbuffaddr)
  (ldd (&y 4) (comment "pick up slave buffer address"))
  (std slavebuffaddr)
  (ldab (! 6) (comment "fix up return address"))
  (aby)
  (pshy)
  (ldab (! spi-int-wait) (comment "wait for the slave to interrupt and wake up"))
  (bsr waiter)
  (ldd masterbuffaddr)
```

```
  (xgdy)
comloop
  (ldab (& spsr) (comment "clear out the SPSR"))
  (ldab (&y 0))
  (stab (& spdr))
  (iny)
  (ldd slavebuffaddr (comment "get slave destination"))
  (xgdy)
  (std masterbuffaddr)
  (ldab (! spi-byte-wait) (comment "wait for subsequent bytes"))
  (bsr waiter)
  (ldab (& spdr))
  (stab (&y 0))
  (iny)
  (ldd masterbuffaddr)
  (xgdy)
  (std slavebuffaddr)
  (dec netbytecount (comment "check for enough bytes"))
  (bne comloop)
fora
  (ldaa (! spi-idle))
  (staa (& porta) (comment "switch off the slave"))
  (rts)
  (comment)
  (comment "a routine to wait some amount of time")
  (comment "a call (ldab (! <n>)) (jsr waiter) takes 2+7+2n+3n+5 = 14+5n cycles")
waiter
  (decb)
  (bne waiter)
  (rts)

_communicate2
  (puly (comment "get return address = data block"))
  (ldaa (&y 0) (comment "pick up slave address"))
  (staa (& porta) (comment "select the appropriate slave"))
  (ldaa (& spsr) (comment "clear out the SPSR"))
  (ldaa (&y 1) (comment "pick up byte count"))
  (staa (& spdr))
  (staa netbytecount)
  (ldd (&y 2) (comment "pick up master buffer address"))
  (std masterbuffaddr)
  (ldd (&y 4) (comment "pick up slave buffer address"))
  (std slavebuffaddr)
  (ldab (! 6) (comment "fix up return address"))
  (aby)
  (pshy)
  (ldab (! (+ spi-int-wait 5)) (comment "wait for the slave to interrupt and wake up"))
  ;; Espera con cinco vueltas al bucle por si el main no esta disponible
  (bsr waiter)
  (ldd masterbuffaddr)
  (xgdy)
;bucle
  (ldab (& spsr) (comment "clear out the SPSR"))
  (ldab (&y 0))
  (stab (& spdr))
  (iny)
  (ldd slavebuffaddr (comment "get slave destination"))
  (xgdy)
  (std masterbuffaddr)
  (ldab (! spi-byte-wait) (comment "wait for subsequent bytes"))
  (bsr waiter)
  (ldab (& spdr)) (comment "get the first byte from the main processor")
  (cmpb (! 21))
  (bne continua)
  (inc funused)
  (bra fora)
continua
  (stab (&y 0))
  (iny)
  (ldd masterbuffaddr)
  (xgdy)
  (std slavebuffaddr)
  (dec netbytecount (comment "check for enough bytes"))
  (jmp comloop)



  (comment)
  (comment)
  (comment "************************************************")
  (comment "************************************************")
```

```
  (comment)
  (comment "timer overflow compare 1 interrupt")
  (comment)
toc1i
  (ldaa (! tmsk1-flag) (comment "pick up the right bit pattern"))
  (staa (& tflg1) (comment "and switch off the interrupt"))
  (ldab timeslice (comment "pick up the time slice"))
  (incb (comment "and increment"))
  (cmpb (! numslices) (comment "and check for overflow"))
  (bne no-overflow)
  (clrb (comment "reset to slice zero"))
no-overflow
  (stab timeslice (comment "store new value"))
  (aslb (comment "multiply by two for two byte indexing"))
  (ldy (! time-slices) (comment "point to time slice table"))
  (aby (comment "compute index"))
  (ldd (&y 0) (comment "pick up time slice"))
  (addd (& toc1) (comment "add in time of last interrupt"))
  (std (& toc1) (comment "and store again"))
  (ldy (&y (- dispatch-table time-slices)) (comment "pick up dispatch address"))
  (jmp (&y 0) (comment "and jump there"))
  (comment)
  (comment "Time slice table (17 entries)")
  (comment)
time-slices
  (!16 4ms)
  (iterate ((i 16)) (!16 1ms))
  (comment)
  (comment "Dispatch table (17 entries)")
  (comment)
dispatch-table
  (!16 zero-state)
  (!16 first-state)
  (iterate ((i 14)) (!16 normal-state))
  (!16 terminal-state)
  (comment)
  (comment "all the different interrupt handlers for 17 possible states")
  (comment)
  (comment)
  (comment "zero state --- signal ok to turn on the motors")
  (comment)
zero-state
  (inc run-motors-flag (comment "make the flag non zero"))
  (rti (comment "and dismiss the interrupt"))
  (comment)
  (comment "first state --- start force measurements and communicate")
  (comment)
first-state
  (inc communicate-flag (comment "make the flag non-zero"))
  (ldy (! tempforces) (comment "load up temporary table and clear"))
  (iterate ((i 12)) (clr (&y i)))
  (comment "now fall into normal state")
  (comment)
  (comment "normal state --- update force measurements")
  (comment)
;;; TIMESLICE tells which of 17 different slices this one is.  Each time the
;;; motor current is ON (the corresponding bit of portE or portA is low) then
;;; TIMESLICE is stored in the correct TEMPFORCE slot for that motor. -jlj
normal-state
  (ldab timeslice (comment "pick up current time slice"))
  (decb (comment "back up from update"))
  (ldaa (& porte) (comment "get first eight bits"))
  (coma)
  (iterate ((label (end01 end02 end03 end04 end05 end06 end07 end08))
            (place (0 1 2 3 4 5 6 7)))
           (tsta)
           (bge label)
           (stab (+ place tempforces))
           label
           (asla))
  (ldaa (& porta) (comment "get last four bits"))
  (coma)
  (asla (comment "get bit leftmost"))
  (asla)
  (asla)
  (asla)
  (iterate ((label (end09 end10 end11 end12))
            (place (8 9 10 11)))
           (tsta)
           (bge label)
           (stab (+ place tempforces))
           label
```

```
          (asla))
  (rti (comment "and dismiss the interrupt"))
  (comment)
  (comment "terminal state --- finish up force measurements")
  (comment)
terminal-state
  (ldy (! tempforces) (comment "load up temporary table and transfer"))
  (iterate ((i (0 2 4 6 8 10)))
          (ldd (&y i))
          (std (&y (+ i (- forces tempforces)))))
  (rti (comment "and dismiss the interrupt"))



 (comment "******************** RUTINA PARA INTERRUPCIONES NO DESEADAS *********************")
bad_int                          (comment "No hace nada")
   (rti                          (comment "Fin de la rutina de interrupcion"))


(comment "*********************** INICIALIZACION VECTOR INTERRUPCIONES *********************")
   (= #xFFD6)
   (!16 bad_int                  (comment "Interrupcion de SCI Serial System"))
   (!16 bad_int                  (comment "Interrupcion de SPI Serial Transfer Complete"))
   (!16 bad_int                  (comment "Interrupcion de Pulse Accumulator Input Edge"))
   (!16 bad_int                  (comment "Interrupcion de Pulse Accumulator Overflow"))
   (!16 bad_int                  (comment "Interrupcion de Timer Overflow"))
   (!16 bad_int                  (comment "Interrupcion de Timer Output Compare 5"))
   (!16 bad_int                  (comment "Interrupcion de Timer Output Compare 4"))
   (!16 bad_int                  (comment "Interrupcion de Timer Output Compare 3"))
   (!16 bad_int                  (comment "Interrupcion de Timer Output Compare 2"))
   (!16 tocli                    (comment "Interrupcion de Timer Output Compare 1"))
   (!16 bad_int                  (comment "Interrupcion de Timer Input Capture 3"))
   (!16 bad_int                  (comment "Interrupcion de Timer Input Capture 2"))
   (!16 bad_int                  (comment "Interrupcion de Timer Input Capture 1"))
   (!16 bad_int                  (comment "Interrupcion de Real Time Interrupt"))
   (!16 bad_int                  (comment "Interrupcion de IRQ"))
   (!16 bad_int                  (comment "Interrupcion de XIRQ"))
   (!16 bad_int                  (comment "Interrupcion de SWI"))
   (!16 start                    (comment "Interrupcion de Illegal Opcode Trap"))
   (!16 bad_int                  (comment "Interrupcion de COP Failure"))
   (!16 bad_int                  (comment "Interrupcion de COP Clock Monitor Fail"))
   (!16 start                    (comment "Interrupcion de RESET"))
   ))
```

ANEXO 9. INDICACIONES DE USO DEL PROTOCOLO

El protocolo se presenta como una librería (*prtcgeng.lib*) con un fichero de cabecera con la definición de las macros y las funciones (*prtcgeng.h*). Esto permite optimizar el proceso de compilación de los programas que lo utilicen. Además se puede crear un fichero para indicar el puerto de comunicaciones (*COM*) que se desea utilizar para la comunicación. Este fichero debe llamarse *prtcgeng.cfg* y el primer carácter de este fichero debe ser uno entre "1" y "4" e indicará el *COM* a utilizar, el resto de caracteres no importan.

La manera de *compilar* y *linkar* los programas de usuario con la librería del protocolo depende del entorno elegido, pero en general lo que se ha de hacer es incluir el fichero de cabecera *prtcgeng.h* en el que implemente el código de usuario y, después de la *compilación* de este fichero o ficheros de usuario, *linkar* los ficheros objeto junto con la librería *prtcgeng.lib*.

En cuanto a los programas de usuario no es preciso que tengan una estructura determinada, pero lo que sí se debe tener en cuenta es que el protocolo se ha diseñado para un funcionamiento óptimo con un tipo de programas de usuario determinados. En caso de otro tipo de programas de usuario el funcionamiento del protocolo puede no ser óptimo.

El programa de usuario para el que se ha diseñado el protocolo inicialmente realiza lecturas de sensores continuamente hasta que detecta el inicio de ciclo por la llegada de nuevos datos. En ese momento se recomienda hacer una consulta de los indicadores sobre el estado de la comunicación por si se han producido situaciones anómalas o no esperadas. Después de la llegada de información de sensores el programa dispone de 20 milisegundos para realizar los cálculos que desee a partir de los datos recibidos o otros, y si se desea, generar nuevas órdenes para los actuadores e indicar que se desea enviarlas al robot. Después de este tiempo se deberán realizar nuevas lecturas de sensores para sincronizarse de nuevo con el inicio del nuevo ciclo y volver a repetir todo el proceso.

Para saber exactamente cuál es la interficie disponible en cuanto a los parámetros, valores de retorno, etc... de las funciones se puede consultar el punto 7.3.3 o el propio fichero de cabecera *prtcgeng.lib*.