ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN
Universidad de Málaga

PROYECTO FIN DE CARRERA:

ENLACE RADIO BIDIRECCIONAL PC-MICROBOT

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

SISTEMAS ELECTRÓNICOS

Málaga, 2001 Marcos Enrique Rodríguez Regidor

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN
Universidad de Málaga

ENLACE RADIO BIDIRECCIONAL PC-MICROBOT

REALIZADO POR:

Marcos Enrique Rodríguez Regidor

DIRIGIDO POR:

Cristina Urdiales García

DEPARTAMENTO DE: Tecnología Electrónica

TITULACIÓN: Ingeniería Técnica de Telecomunicación
Sistemas Electrónicos

Palabras Clave: Microbot, enlace radio, SRD, encoder, Labwindows.

RESÚMEN:

El presente proyecto consiste en el diseño e implementación de un sistema que permite realizar la comunicación entre un ordenador personal (PC) y un robot autónomo *(Microbot)*. El enlace radio diseñado para permitir dicha comunicación es bidireccional, es decir tanto el PC como el Microbot poseen un transmisor y un receptor radio. Dicho enlace permite tratar en el PC la información procedente de los sensores del Microbot, ampliando así la escasa capacidad de proceso de este último. El desarrollo del Software incluido en este proyecto se ha llevado a cabo utilizando los entornos *TurboC* de *Borland* y *Labwindows* para el PC y el Ensamblador propio del Microcontrolador 68HC11 de *Motorola* para el Microbot.

Málaga, Noviembre de 2001

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN

1 Presentación y objetivos

El objetivo de este proyecto es el diseño e implementación física de un sistema que permita realizar la comunicación entre un ordenador personal (PC) y distintos robots autónomos (Microbots), utilizando una interfaz inalámbrica. Una vez posibilitada esta comunicación, quedará constituido un sistema de control local donde el PC actuará como la base y los Microbots como los móviles. El enlace radio que permitirá realizar dicha comunicación deberá ser b*idireccional*, es decir tanto el PC como los Microbots deberán disponer de un transmisor y de un receptor, con lo que podrán tanto transmitir como recibir datos, posibilitando tanto el flujo de información PC → Microbot, como Microbot → PC e incluso Microbot → Microbot.

Este proyecto se plantea como evolución del enlace radio unidireccional PC→ Microbot diseñado en un proyecto anterior realizado en el seno del Departamento de Tecnología Electrónica de la Escuela Técnica Superior de Ingenieros de Telecomunicación en la Universidad de Málaga *[PFC1]*, con la idea de utilizar el máximo numero de componentes comunes con aquel. En dicho proyecto se implementó un sistema en el que el PC poseía un transmisor radio y en el Microbot un receptor, de modo que el primero enviaba constantemente comandos de movimiento al segundo, con lo que no se podía realizar en el PC tratamiento alguno de la información de los sensores del Microbot, puesto que ni éste poseía un transmisor, ni el PC poseía un receptor. Esta capacidad de proceso externo al Microbot es lo que se pretende conseguir con el diseño del sistema que nos ocupa.

Como resultado final del presente proyecto se dispondrá de un prototipo constituido por dos placas de circuito impreso, una conectada al PC y otra integrada en el Microbot, ambas con capacidad para recibir y transmitir, las cuales permitirán probar los resultados de los trabajos realizados.

Para estructurar de forma clara la presentación del trabajo realizado, la memoria se estructura en seis capítulos:

- En el primer capítulo se lleva a cabo una introducción en los objetivos de este proyecto, así como en los conceptos que permitirán la comprensión de los restantes apartados.
- El segundo capítulo trata del sistema Microbot y en el se muestran conceptos teóricos básicos relativos este sistema, necesarios para una buena compresión del trabajo realizado.
- En el tercer capítulo es el de Definición y Análisis. En la Definición del sistema partiendo de unos objetivos se llegará a unas especificaciones. El Análisis se concretará en la división del sistema en bloques funcionales.
- El diseño completo de estos bloques se detalla en el capítulo cuarto.
- En el quinto capitulo se describen las pruebas realizadas con los prototipos, así como los resultados que se desprenden de ellas.
- En el sexto capitulo se deja constancia de las conclusiones del proyecto y de las posibles líneas futuras de desarrollo del sistema.

Por último se describen los materiales empleados en la realización del prototipo, constituyendo un presupuesto con el coste final del mismo. Al final de la memoria se incluye la bibliografía empleada, así como tres anexos. El primero trata del diseño de las Placas de Circuito Impreso de los prototipos, el segundo del montaje de encoders en las ruedas del Microbot y en el tercero se describen las funciones y subrutinas que se pueden encontrar en el software desarrollado.

2 Introducción a los Microbots

Los Microbots son una familia de robots autónomos de reducidas dimensiones normalmente basados en microcontroladores. Los Microbots pueden actuar independientemente o realizando una tarea en conjunto, es decir a cada Microbot se le asigna una "subtarea" que debe cumplir y además debe ser capaz de realizar la tarea de otro compañero que por alguna razón deja de funcionar. Así el conjunto de Microbots debe tener la capacidad de suplir las posibles bajas, de tal manera que los Microbots restantes vayan cumpliendo sus propias "subtareas" y las de sus compañeros "estáticos". En el mejor de los casos el sistema "cae" cuando todas las unidades lo hacen simultáneamente.

Estos pequeños robots autónomos han ido tomando diferentes nombres desde su aparición *(Mobile Robots, Microbots, etc*) pero lo que no ha variado es su concepción.

El interés que estos sistemas ha despertado, ha llevado a la construcción de granjas (habitaciones inteligentes que forman el universo de los Microbots, donde pueden ser programados y altamente controlados). Este campo ha llamado la atención de profesionales que ven un alto grado de paralelismo entre las granjas y sus estudios. *[web1]*

2.1 Microbot Clónico

Es el *Microbot* objeto de este proyecto. Su estructura es metálica, lo que la hace muy robusta. Asimismo, para su movimiento esta dotado de orugas, lo que le permite desplazarse por terrenos más difíciles que los que pueden superar por la mayoría de sus compañeros de granja. Está pensado para crecer hacia arriba fácilmente con nuevas placas de circuito impreso que le añadan nuevas funciones.

Incorpora diversos sensores, como son los de infrarrojos de corto alcance situados en su parte frontal, que le permiten por ejemplo seguir una línea negra. Incorpora además sensores de contacto (o *bumpers*) también situados en su parte delantera, para detectar la colisión contra obstáculos y un sensor de luz que le permite por ejemplo seguir una fuente luminosa.

A continuación se muestra un esquema de los actuadores (sensores y motores) del Microbot Clónico.

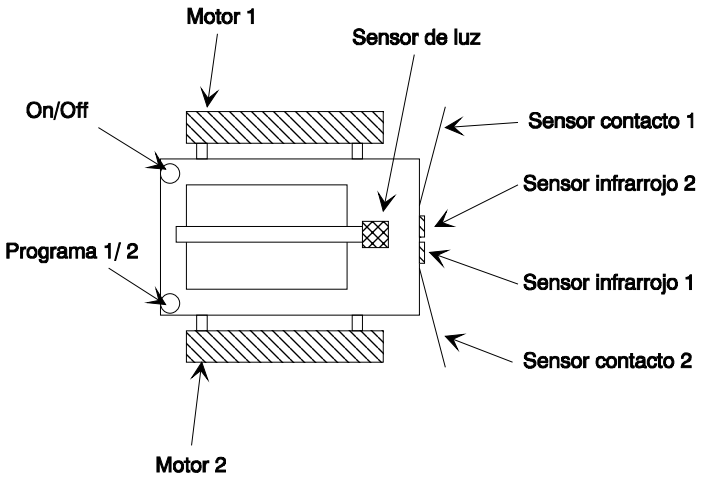


Fig. 1: Diagrama del Microbot Clónico.

El Microbot Clónico lleva como sistema de control la tarjeta CT6811 que está basada en el microcontrolador 68HC11 y la tarjeta CT293+ como *driver* de potencia para controlar dos servomotores.

Dadas las características de estas tarjetas puede funcionar en modo autónomo o bien conectado al PC a través del puerto serie.

2.2 Modelo Torre Bot

Una de las clasificaciones que normalmente se utiliza para analizar a los *Microbots* es la llamada *Torre Bot.* Se trata de un modelo en torre, donde cada uno de sus niveles representa un paso en la fabricación de un sistema *Microbótico.* Se pueden distinguir seis niveles: Nivel Físico, Nivel de Reacción, Nivel de Control, Nivel de Inteligencia, Nivel de Comunidad y Nivel de Cooperación. Se describen a continuación dos de los niveles más importantes:

*Nivel de Control: Incluye los circuitos más básicos que relacionan las salidas de los sensores con las restantes unidades. Partiendo de una simple lógica digital hasta potentes *microcontroladores*, se *bus*ca dotar al Microbot de la capacidad para procesar la información obtenida por los sensores, así como actuar de una manera controlada sobre unidades motoras.

*Nivel de Cooperación: Comprende los sistemas donde a partir de un nivel de comunidad (granjas), se planifican o programan los Microbots para que tengan conocimiento de la existencia de otros, tal que posean la capacidad de cooperar para el buen desarrollo de una tarea.

3 Necesidad de una interfaz inalámbrica

Al incorporar sistemas inalámbricos de transmisión de datos (infrarrojos, radio, ultrasonidos, etc) tanto a los Microbots, como a un ordenador central que los pudiera controlar, se consigue mejorar en el sistema estos dos niveles:

*En el Nivel de Control: debido a que los Microbots suelen tener pequeña cantidad de memoria, con lo que la capacidad de proceso de estos se podría mejorar realizando el tratamiento de la información procedente de los sensores en un ordenador central (PC) o incluso distribuyéndolo a través de una red de ordenadores. Para ello y debido a la movilidad de los Microbots es necesaria una comunicación inalámbrica.

*En el Nivel de Cooperación: ya que esta cooperación se basa en esencia en una comunicación entre los Microbots. Esto hace necesaria la existencia de una interfaz inalámbrica, como la que es el objeto de este proyecto.

4 Los sistemas de transmisión de datos inalámbricos

Los sistemas de comunicación de datos inalámbricos entre *Microbots*, son sistemas que no suelen requerir mucha potencia, a continuación se pasa a hacer un estudio más detallado de los dos sistemas más utilizados, los sistemas de transmisión de datos por ***infrarrojos*** y por ***radiofrecuencia*.**

Ambos sistemas constan de un emisor y un receptor, para la comunicación en cada sentido, según la Figura 2.

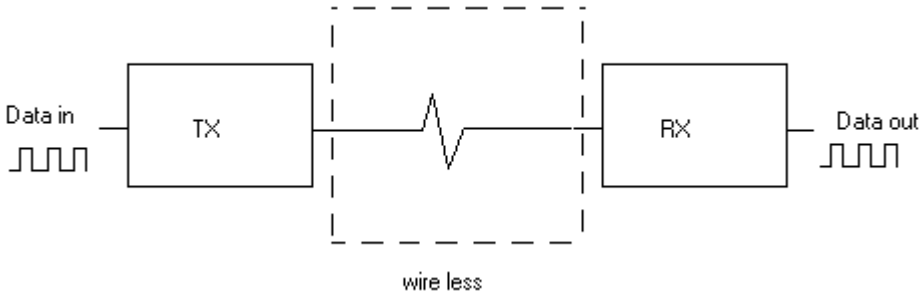


Fig. 2: Esquema de una transmisión de datos inalámbrica.

4.1 Sistemas de transmisión de datos por Infrarrojos

En los sistemas por infrarrojos el módulo emisor consta de un diodo LED emisor de infrarrojos, el cual mediante un oscilador se hace oscilar a una frecuencia (frecuencia de portadora). Esta modulación se producirá o no en función de los datos a enviar (moduladora). Por ejemplo, para los '1' transmite y no lo hace para los '0'. En el receptor un diodo sensible a infrarrojos recibirá los impulsos del emisor. Al producirse la sintonía con el emisor se detectará un '1' y en cualquier otro caso un '0'.

Este tipo de sistemas presenta dos grandes inconvenientes: Por un lado, su corto alcance y por el otro su directividad, esto es, el emisor y el receptor deben estar perfectamente encarados para la sintonización, además no pueden existir obstáculos entre ellos. Como ventaja es de destacar su precio y la facilidad de montaje de estos módulos, ya que requieren poca cantidad de componentes y están muy difundidos.

4.2 Sistemas de transmisión de datos por Radiofrecuencia

Los sistemas de radiofrecuencia emiten ondas electromagnéticas a través de un medio natural. Si se emplean estos sistemas en la transmisión de datos se obtiene un sistema de comunicación más fiable y robusto que en el caso de los infrarrojos.

Estos módulos de radiofrecuencia independientemente de su complejidad o técnica de modulación (AM, FM, FSK, etc), subsanan los inconvenientes de los infrarrojos, esto es, se consiguen sistemas de transmisión mucho menos directivos, lo que permite una movilidad tanto del emisor como del receptor sin peligro a que se pierda la comunicación. Además se permite la existencia de obstáculos entre los sistemas de TX (transmisión) y RX (recepción). Existiendo módulos que incorporan empaquetados juntos transmisor y receptor, de aquí la forma de llamarlos en ingles: '*Transceivers'*. Utilizando transmisión por radiofrecuencia se consigue también un aumento del alcance, estos módulos pese a que sean de baja potencia consiguen unas distancias del orden de decenas de metros.

El precio que hay que pagar para obtener estas ventajas es el coste de estos módulos (bastante mayor que los infrarrojos), así como un aumento de la complejidad y de los ajustes (siempre en comparación con los sistemas infrarrojos).

Por todo lo comentado y atendiendo a las necesidades que exige la comunicación entre *Microbots*, se ha optado por la segunda opción (los módulos de transmisión de RF) para la realización del presente proyecto. Concretamente se emplearán dispositivos de baja potencia (LPDs), también llamados dispositivos de bajo alcance (SRDs). Estos sistemas se utilizan para la transmisión de datos y de control remoto sin licencia. Para ello utilizan alguna de las bandas ICM (antiguas ISM - Industrial, Scientific and Medical), o cualquier otra banda en la cual no se necesite licencia para transmitir (por ejemplo la de 27 MHZ, frecuencia pública para radioaficionados).

Estos módulos tienen muchas aplicaciones, como por ejemplo, comunicaciones digitales entre un PC y una impresora, actuando como radio enlace efectivo sin necesidad de un largo cable de RS-232 *[Elek.98]*.

CAPÍTULO 2. EL SISTEMA MICROBOT

1 LA TARJETA CT6811

1.1 Introducción a la CT6811

La CT6811 es una tarjeta para el desarrollo de aplicaciones hardware y software basadas en el microcontrolador 68HC11. No sólo sirve para el desarrollo de prototipos, sino también está pensada para funcionar en **sistemas terminados**. Se trata sobre todo de una tarjeta muy versátil, que se puede emplear como:

- **Tarjeta entrenadora:** Conectándose a ordenadores PC a través del puerto serie. Es posible enviar programas desde el PC a la CT6811 para que los ejecute. De esta manera, la tarjeta es muy útil para la depuración de programas en fase de pruebas y para aprender a programar el 68HC11 desde un punto de vista práctico: todos los programas creados sobre el "papel" se pueden ejecutar en un 68HC11 "de verdad".
- **Tarjeta autónoma de control:** La CT6811 funciona en modo autónomo, es decir, sin conectarse al PC. Cada vez que se encienda la tarjeta, se ejecutará el programa que previamente se habrá grabado en la memoria EEPROM del 68HC11. Conectando periféricos a través de los puertos de expansión, se consiguen desarrollar sistemas inteligentes de control: control de las luces de una casa, control de la maqueta de un tren, manejo de *Microbots,* alarmas, etc.
- **Periférico inteligente del PC:** También es posible utilizar la tarjeta para comunicar el PC con el mundo exterior. La CT6811 actuaría como un periférico conectado al PC a través del puerto serie. Este periférico "inteligente" puede recibir órdenes del PC y actuar en consecuencia. Se pueden realizar aplicaciones del tipo: digitalización de señales y presentación en el PC, diseño de joysticks no convencionales, control de luces de la casa a través del PC, control de maquetas de tren visualizando en el PC la situación de los trenes, semáforos, etc.

1.2 Características de la CT6811

- **Microcontrolador 68HC11**, incluyendo por tanto todas las características de este:
  - 512 Bytes de EEPROM en los modelos A1 y A8.
  - 256 Bytes de memoria RAM interna.
  - Temporizador de 16 bits.
  - 3 capturadores de entrada.
  - 5 comparadores con salida hardware.
  - Un acumulador de pulsos.
  - Comunicaciones serie asíncronas.

  - Comunicaciones serie síncronas.
  - 8 canales de conversión analógico digital.
  - Interrupciones en tiempo real.
  - 4 puertos de E/S.

- ***Bus* de expansión:** *Bus* de expansión dividido en 6 puertos de 10 bits cada uno. Desde estos puertos se tiene acceso a todas las patas (*pines*) del micro.
- ***Switches* de configuración:** 2 *switches* de configuración del modo de arranque de la placa (*bootstrap*, *singlechip*, *expanded*, *special* test*)* y 2 *switches* disponibles para aplicaciones del usuario.
- ***Led* de pruebas conectado al bit 6 del puerto A.** Este *led* se puede conectar y desconectar por medio de un *jumper.*
- **Pulsador de *reset*/pruebas IRQ:** Pulsador para inicializar la tarjeta. Este pulsador se puede utilizar también, cambiando un *jumper,* como un pulsador de propósito general conectado a la entrada de interrupciones IRQ.
- **Niveles VRH y VRL** configurables a VCC y masa respectivamente mediante 2 *jumpers.*
- **Modos de funcionamiento** entrenador/autónomo configurables mediante un *jumper.*
- **Alimentación** a través de clemas o conector tipo *jack.*
- **Conexión directa al PC** por medio de un cable tipo teléfono.
- ***Reset* software y hardware** de la placa.

1.3 Diagrama de la tarjeta CT6811

La tarjeta CT6811 presenta el siguiente aspecto:

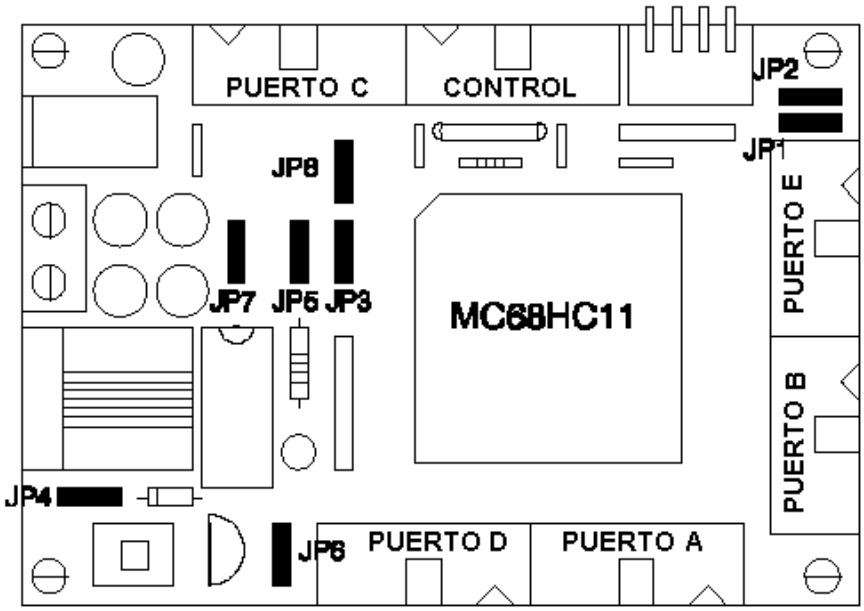


Fig. 3: Diagrama de la tarjeta CT6811.

La tarjeta se puede dividir en 8 bloques: el corazón de la placa (68HC11), el circuito de reloj, el circuito de inicialización o *reset*, el circuito de pruebas, el circuito de comunicaciones con el PC, la configuración del sistema, los puertos de expansión y la alimentación del sistema completo. A continuación se explica cada bloque:

- **Microcontrolador 68HC11A1:** Se trata de un microcontrolador de 8 bits. Además de una CPU que le permite ejecutar instrucciones y realizar operaciones aritmético lógicas, dispone en el propio chip de memorias ROM, RAM y EEPROM, así como una serie de periféricos integrados que hacen de este micro una herramienta muy potente.
- **Circuito de reloj:** Este circuito permite que el micro obtenga la señal de reloj necesaria para funcionar. Está constituido por un cristal de 8 MHz, que es el valor máximo que permite el 68HC11. Además, con este valor, el micro es capaz de comunicarse con el PC a las velocidades de 1200,2400 ó 9600 baudios.
- **Circuito de *reset*:** El circuito de *reset* permite la correcta inicialización del 68HC11. Se ha diseñado de tal forma que es posible realizar un *reset* por *hardware* apretando un pulsador, o bien realizar un *reset* por *software* desde el PC, cuando la CT6811 está funcionando en modo entrenador. El *reset* software es muy útil cuando se están desarrollando programas para *Microbot,* pues cada vez que hay que cargar un programa nuevo en el *Microbot,* no es necesario desplazarse hasta él y apretar el botón de *reset*, sino que directamente desde el PC se puede llevar a cabo.
- **Configuración:** La parte de configuración de la CT6811 está constituida por 6 *jumpers* y 4 *switches*. Dos de los *switches* permiten configurar el 68HC11 para trabajar en cualquiera de los 4 modos para los que está diseñado: *bootstrap*, *single* chip, *expanded* y *special*. Los otros 2 *switches* están a disposición del usuario a través de los puertos de expansión para configurar tarjetas periféricas conectadas a la CT6811. Los 6 *jumpers* permiten configurar la CT6811 para realizar diferentes funciones, que se comentarán más adelante.
- **Circuito de pruebas:** La CT6811 dispone de un *led* conectado al bit 6 del puerto A del 68HC11. Este *led*, que se puede conectar y desconectar a través de un *jumper*, es muy útil para la realización de software de pruebas. El pulsador de la tarjeta, se puede configurar mediante un *jumper* para actuar bien como pulsador de *reset* o bien como un pulsador normal conectado a la entrada de interrupción IRQ del 68HC11. De este forma, es posible realizar pequeños programas de prueba que lean el pulsador y actúen en consecuencia.
- **Comunicaciones con el PC:** Esta es una de las partes más importante. A través del circuito de comunicaciones es posible comunicarse con el PC para intercambiar información. Las comunicaciones se realizan a través del puerto serie del PC, mediante la norma RS-232. La velocidad máxima compatible con el PC es de 9600 baudios, aunque el 68HC11 es capaz de transmitir a mucha más velocidad.

- **Puertos de expansión:** La CT6811 dispone de 6 puertos de expansión donde es posible conectar los diferentes periféricos. Todas las señales del 68HC11, salvo las correspondientes al reloj (EXTAL y XTAL), son accesibles desde los puertos de expansión. Cinco de los puertos coinciden con los 5 puertos del 68HC11: A, B, C, D y E. El sexto puerto está destinado a llevar las señales de control del 68HC11.
- **Alimentación:** La tensión de alimentación de la CT6811 oscila entre 4.5 y 5.5 voltios. La tarjeta dispone de un conector hembra de tipo *jack* cilíndrico para la conexión de un transformador, y dos bornas ajustables mediante tornillos para la conexión de cables de alimentación, provenientes por ejemplo de pilas o baterías. *[Elek.94]*

1.4 Configuración de la tarjeta

La tarjeta CT6811 dispone de *switches* y *jumpers* para que el usuario la configure según sus necesidades. Existen 4 *switches* y 8 *jumpers*. De los 4 *switches*, dos se utilizan para configurar los modos de funcionamiento del 68HC11 y los otros dos quedan disponibles para aplicaciones del usuario.

1.4.1 Configuración de los modos del micro

La configuración del modo de funcionamiento del microcontrolador se realiza a través de los switches presentes en la tarjeta, según se indica en el siguiente cuadro.

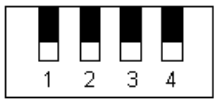
SWITCHES 1 Y 2	MODA	MODB	MODO	DESCRIPCION
	0	0	Bootstrap	No acceso a memoria externa Ejecución programa BOOTSTRAP
	0	1	Single Chip	No acceso a memoria externa Vectores interrupción en ROM
	1	0	Special test	Acceso a memoria externa Modo especial Vectores interrupción en ROM
	1	1	Expanded	Acceso a memoria externa Vectores interrupción en ROM o RAM

Fig. 4: Configuración de los modos mediante los *switches* de la tarjeta CT6811.

1.4.2 Configuración de los *jumpers*

La tarjeta CT6811, dispone de una serie de jumpers, mediante los cuales se pueden configurar diversos aspectos de su funcionamiento. La función de dichos jumpers se detalla en la figura 5.

JUMPER	FUNCIÓN	CONECTADO	QUITADO
JP1	Actuar sobre VRH	VRH = VCC	VRH accesible desde el bus de expansión
JP2	Actuar sobre VRL	VRL = GND	VRL accesible desde el bus de expansión
JP3	Actuar sobre el LED	LED conectado a PA6	LED desconectado PA6 liberado
JP4	Seleccionar usos del pulsador: IRQ/RESET	El jumper en la izquierda activa la IRQ	El jumper en la derecha activa el pulsador de reset
JP5	Seleccionar modo entrenador o autónomo	Modo autónomo	Modo entrenador.
JP6	Actuar sobre el LVI	LVI conectado	LVI desconectado
JP7	Seleccionar reset software ON / OFF	El jumper abajo activa el reset software	El jumper arriba desactiva el reset software
JP8	Actuar sobre XIRQ	Modo normal	Sólo en caso de programar la EPROM del 68HC11E9

Fig. 5: Configuración de los *jumpers* de la CT6811.

1.5 Puertos de expansión

La tarjeta CT6811 dispone de 6 puertos de expansión. Cinco de estos puertos coinciden con los cinco puertos del 68HC11, y se han denominado igual: puerto A, puerto B, puerto C, puerto D y puerto E. El sexto puerto contiene otras señales de interés del 68HC11.

Puerto A: Este puerto está constituido por los 8 bits del puerto A del 68HC11 más un *pin* de VCC y otro *pin* de GND ambos para poder llevar alimentación a los periféricos de una forma cómoda.

Puerto B: A este puerto de expansión se lleva el puerto B del 68HC11. Similar al puerto A, con la excepción de que el puerto B del 68HC11 sirve también como parte alta del *bus* de direcciones en caso de ampliación del micro con memoria externa. Cuando funciona en modo no expandido, el bit PB0 se corresponde con A8, PB1 con A9,..., PB7 con A15.

Puerto C: A este puerto de expansión se lleva el puerto C del 68HC11. Este puerto funciona como un puerto normal de E/S cuando no hay conectada memoria externa. Cuando se conecta memoria externa, este puerto lleva el byte bajo del *bus* de direcciones *multiplexado* con el *bus* de datos. El bit PC0 se corresponde con AD0, PC1 con AD1,..., PC7 con AD7.

Puerto D: Este puerto está constituido por el puerto D del 68HC11. Esta formado por 6 bits que se pueden configurar como entradas o como salidas, pudiendo ser estas normales o en colector abierto. Asimismo estos pines están compartidos por el SCI y el SPI (Unidades de comunicaciones asíncrona y síncrona).

Puerto E: Constituido por el puerto E del 68HC11. Este puerto en el 68HC11 tiene una doble función: puerto de entrada de 8 bits ó 8 canales de conversión A/D. Las señales VRL y VRH se utilizan para introducir las señales de referencia alta y baja del conversor. Si los *jumpers* JP1 y JP2 están colocados, por estos podemos obtener la alimentación: VRH=VCC, VRL=GND. Si no están colocados estos *jumpers*, podremos introducir las tensiones de referencia necesarias para nuestra aplicación.

Control: Por este puerto se han "recopilado" una serie de señales del 68HC11 para el control. Por los *pines* SW3 y SW4 se sacan las señales correspondientes al estado de los *switches* de usuario 3 y 4.

En la siguiente figura se presentan las señales asociadas a cada puerto.

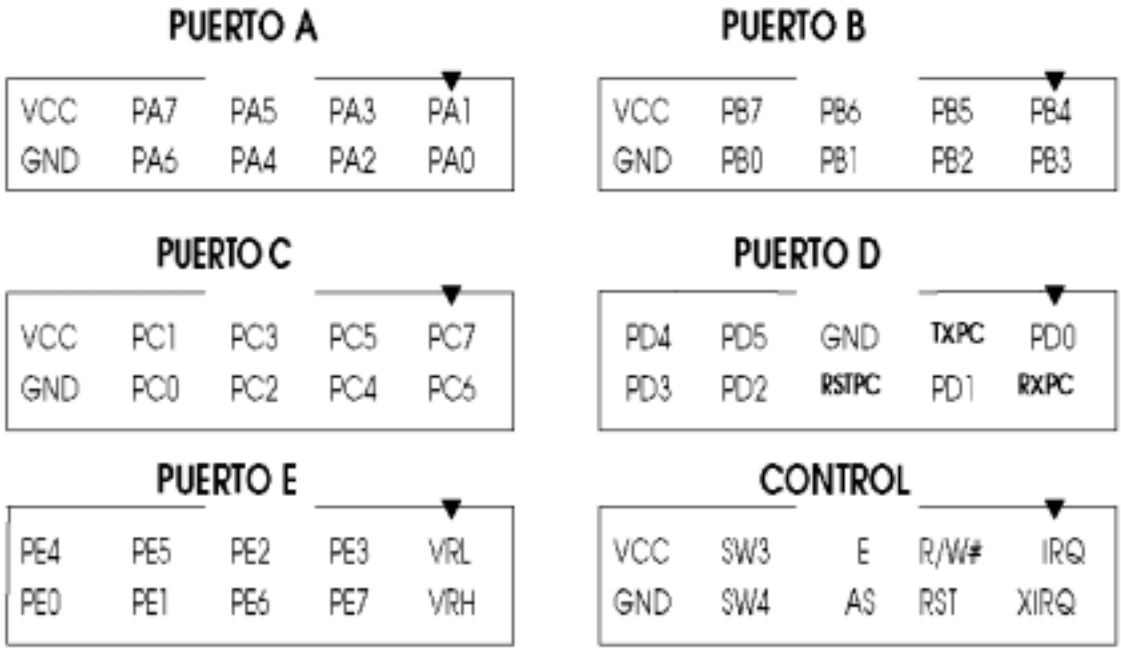


Fig. 6: Señales de los puertos de expansión

1.6 Desarrollo de programas para la CT6811

1.6.1 Filosofía de trabajo

La filosofía empleada para el manejo de la CT6811 es la siguiente. En el PC se programan las aplicaciones para la CT6811 en ensamblador del 68HC11. Estos programas fuente tienen extensión. ASM. Utilizando el ensamblador FREEWARE AS11 de Motorola, los programas se compilan, obteniéndose archivos ejecutables con extensión. S19. Estos son los archivos que se envían a la CT6811.

Para enviar programas a la CT6811 se pueden utilizar varios programas. En los ejemplos de más adelante se utiliza el programa DOWNMCU v.10. Una vez que se ha enviado el programa, el 68HC11 comienza su ejecución. Se puede cargar programas en la CT6811 tantas veces como se quiera. Los programas así cargados son almacenados en la RAM interna del 68HC11.

Una vez que se tiene la aplicación depurada, se graba en la memoria EEPROM del 68HC11 para que se quede permanentemente. Ahora la CT6811 no necesita del PC. Funciona en modo autónomo. Para grabar programas en la EEPROM interna se pueden utilizar varios programas: PC-BUG, CTDIALOG... En esta sección se utilizará el programa CTDIALOG. Este programa, además de permitir grabar la EEPROM, permite ver el contenido de la memoria del 68HC11, cambiar valores de la memoria, desensamblar, etc.

1.6.2 Un ejemplo completo

En esta sección se va a presentar un programa fuente que se va a compilar para después ser cargado en la CT6811 y finalmente grabado en la EEPROM.

El programa en cuestión, hace parpadear el *led* de la CT6811. El código fuente se muestra a continuación.

```
; Simplemente se enciende y se apaga el led de la tarjeta CT6811.

        ORG $000

comienzo
        LDAA $1000
        EORA #$40               ; Cambiar de estado el bit PA6
        STAA $1000

        LDY #$FFFF              ; Realizar una pausa
dec     DEY
        CPY #0
        BNE dec

        BRA comienzo            ; Repetir el proceso
        END
```

Listado del programa LEDP.ASM

El siguiente paso es compilar el programa para obtener el archivo LEDP.S19, que es el fichero ejecutable por el microcontrolador Motorola 68HC11. Para ello se utiliza el ensamblador AS11 v.1.0.3 suministrado por Motorola. A continuación se irán mostrando, los comandos que hay que teclear desde el shell de MS-DOS y los resultados obtenidos.

En la figura 7 primero se muestran todos los ficheros necesarios para la compilación del programa y la comunicación con el Microbot, y a continuación se compila el programa *ledp.asm*. Tras la compilación se obtiene el archivo ledp.s19, que se trata de un archivo de texto y por tanto se puede editar (¡Pero no se debe modificar!).

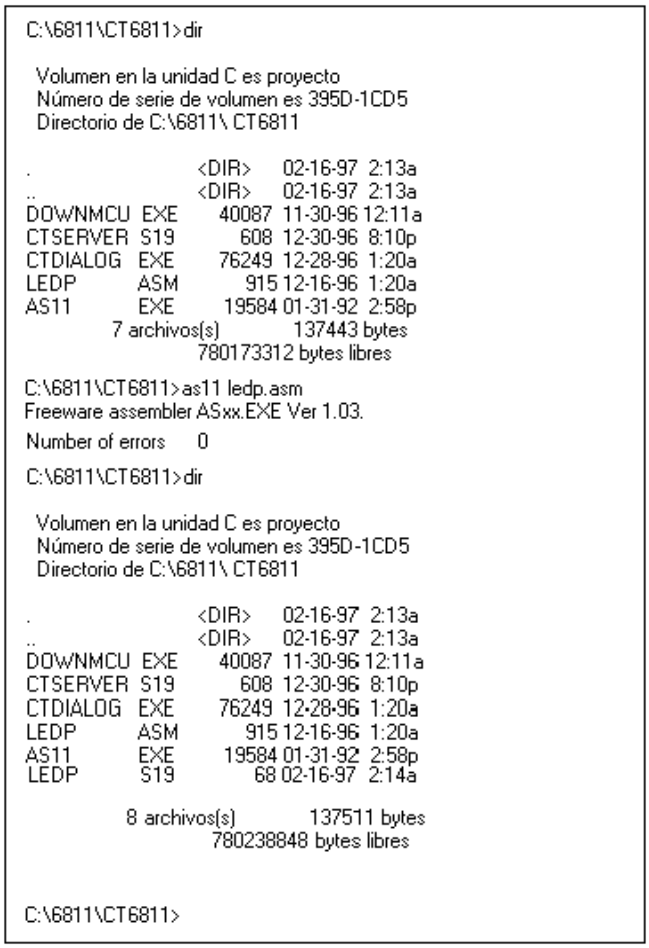


Fig. 7

Ahora hay que enviar el programa LEDP.S19 a la CT6811. En la figura 8 se muestran los comandos. Se ha supuesto que la CT6811 se encuentra conectada en al puerto serie COM2 del PC. Si se quiere utilizar el puerto COM1 basta con cambiar el parámetro –com2 por –com1 (¡en minúsculas!). En cuanto se recibe el mensaje de "ENVIO CORRECTO" el programa ya se estará ejecutando en el 68HC11 y se podrá ver como parpadea el *led*.

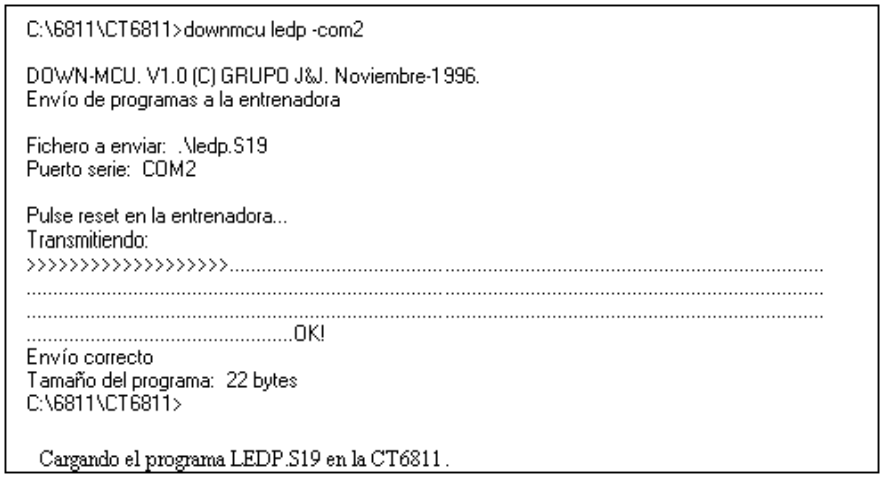


Fig. 8

Los programas así cargados en el 68HC11 se almacenan en la RAM interna del micro. Se pueden cargar los programas todas las veces que se quiera, sin ninguna restricción. Una vez que se tiene el programa depurado, se graba en la EEPROM. En el ejemplo, se trata de un programa muy sencillo y su depuración es muy fácil. A continuación se indica como grabar el programa LEDP en la EEPROM.

Antes de nada hay que modificar el programa fuente para indicar que lo queremos situar en la memoria EEPROM. Si el programa a grabar en la *eeprom* utilizase variables, habría que situar las variables en la RAM interna y dejar sólo el código en la EEPROM. Como el programa *ledp* no tiene variables, no hay que preocuparse. El único cambio que ha habido que hacer es en la directiva ORG, que indica el comienzo del programa. En vez de comenzar en la dirección $0000, ahora deberá comenzar en la dirección $b600, que es el comienzo de la memoria EEPROM en el microcontrolador 68HC11A1. Si utilizamos otro modelo se deberá averiguar esta posición de inicio. El nuevo programa creado se denomina *ledpe.asm*. La "e" del final se ha puesto para indicar que se trata de un programa para la *eeprom*. Una vez compilado este programa, de forma similar a como se ha hecho con el programa *ledp.asm*, obtenemos el fichero ledpe.s19. Este será el fichero que utilicemos para grabar en la EEPROM. A continuación se muestra el listado del programa *ledp.asm* modificado para ser grabado en la memoria EEPROM del 68HC11.

```
; Simplemente se enciende y se apaga el led de la tarjeta CT6811.

        ORG $B600               ; ¡ Memoria EEPROM !

comienzo
        LDAA $1000
        EORA #$40               ; Cambiar de estado el bit PA6
        STAA $1000

        LDY #$FFFF              ; Realizar una pausa
dec     DEY
        CPY #0
        BNE dec

        BRA comienzo            ; Repetir el proceso
        END
```

Programa ledp preparado para ser grabado en la EEPROM.

El programa que se va a emplear es el CTDIALOG. En la figura 9 se muestra como se carga este programa. Primero hay que enviar a la CT6811 el programa CTSERVER.S19, tal y como se ha visto anteriormente, para que éste quede grabado en la memoria RAM, y a continuación se ejecuta el programa CTDIALOG.EXE. Además de permitir grabar la EEPROM del 68HC11, con el CTDIALOG podremos ver el contenido de la memoria, modificarla, desensamblar, ejecutar programas, etc.

Si se ejecuta el programa CTDIALOG y el servidor CTSERVER no se ha cargado previamente, o se ha cargado mal, aparecerá en el *prompt* del CTDIALOG un asterisco, indicando que no hay conexión con la tarjeta.

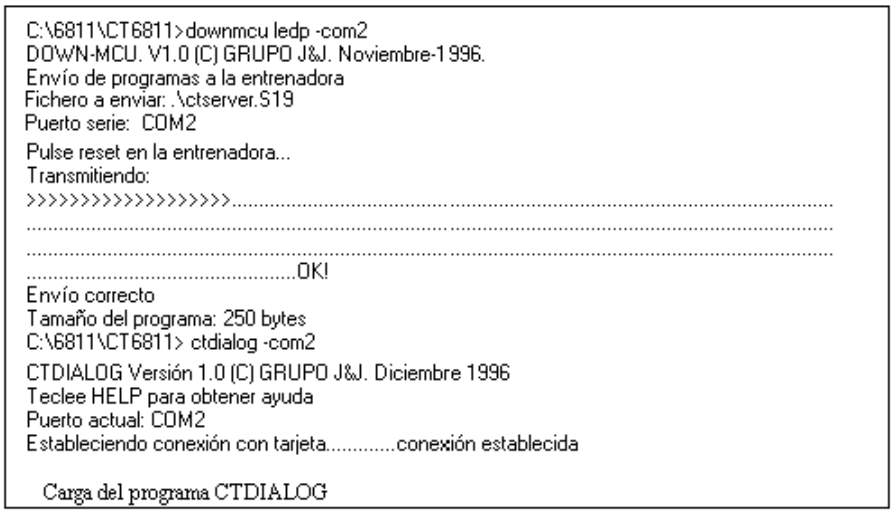


Fig. 9

Para grabar el programa ledpe.s19 en la EEPROM hay que utilizar el comando EEPROM seguido del nombre del fichero. Los pasos a seguir se encuentran explicados en la figura 10: primero se graba el programa en la EEPROM y después se desensambla el contenido de la *eeprom* para asegurarse de que se ha grabado correctamente. Para probar el programa, se puede configurar la CT6811 en modo autónomo (conectando el *jumper* JP5) y hacer un *reset*. También es posible realizar un "salto" a la EEPROM desde CTDIALOG, como se ha hecho en el ejemplo de la figura 10 (el comando de salto a la EEPROM es '*g b600*'). Al realizar el salto, el 68HC11 comienza a ejecutar el programa almacenado en la EEPROM. El servidor que estaba en la RAM interna se deja de ejecutar y por tanto se pierde la conexión del CTDIALOG con la CT6811. Para volver a ejecutar el CTDIALOG habrá que salir utilizando el comando '*quit*', volver a cargar el servidor y volver a ejecutar el CTDIALOG.

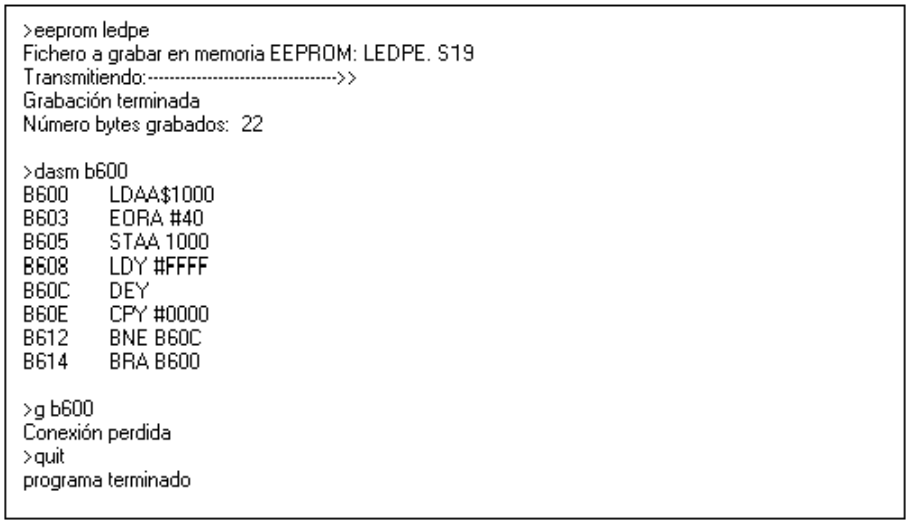


Fig. 10

Para que la tarjeta arranque en modo autónomo, es decir ejecutando el programa almacenado en la EEPROM, hay que colocar el jumper JP5 antes de alimentar la tarjeta o de pulsar el botón de reset.

2 LA TARJETA CT293+

2.1 Introducción a la CT293+

La CT293+ es una tarjeta que proporciona al sistema *tower* la posibilidad de controlar motores y sensores. Esta diseñada para adaptarse perfectamente a la CT6811 y poder controlarla sin ninguna variación tanto en *Bootstrap*, como en *Single* Chip. La tarjeta también puede controlarse desde otro sistema por ejemplo el puerto paralelo de un PC. Esta tarjeta es la versión mejorada de la CT293, pero se ha respetado la compatibilidad. Los programas de su antecesora valen también para la nueva, lo único que hay que verificar es la situación de los sensores y motores.

La CT293+ es el soporte principal de los dos primeros niveles de la torre BOT, con ella se proporciona el movimiento a los sistemas de control y la capacidad de analizar el entorno. Con esta placa y con la CT6811 se puede obtener la plataforma de un *Microbot.*

En la Figura 11 se representa el diagrama de bloques del hardware de un *Microbot*, la tarjeta CT6811 es el sistema microcontrolador usado para la programación y la tarjeta CT293+ es la interfaz entre la tarjeta anterior y los recursos externos (motores, sensores, entradas digitales y entradas analógicas).

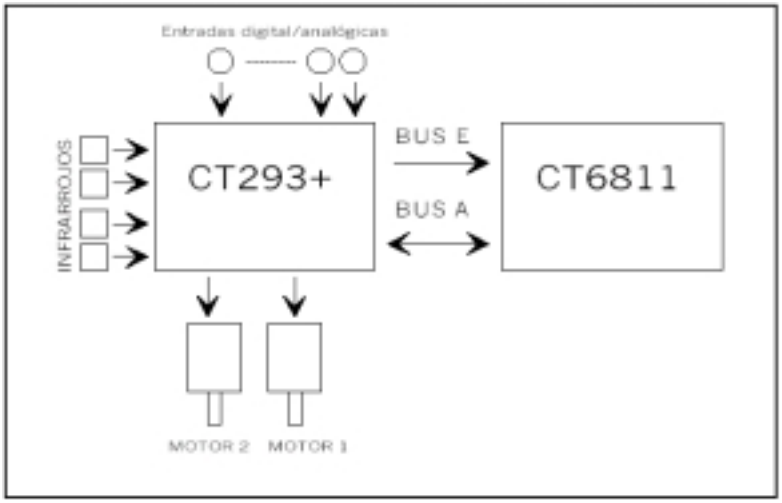


Fig. 11: Conexiones del la CT293+

Las características de la tarjeta CT293+ se pueden resumir en:

1. Posibilidad de control de dos motores de continua o uno paso a paso.
2. Capacidad para leer cuatro sensores de infrarrojos, pudiendo ser estos optoacopladores.
3. Disponibilidad de 8 entradas digitales de propósito general con la posibilidad de usarlas como entradas analógicas.
4. Alimentación de los motores externa o interna.

2.2 Descripción de los elementos de la CT293+

En la figura 12 se puede apreciar la disposición de los componentes en la placa. Básicamente tiene dos bloques independientes. El primero de ellos (BLOQUE A) se encarga de los motores y de los sensores de infrarrojos, mientras que el segundo (BLOQUE E) controla las entradas digitales/analógicas. Hay un *bus* de control para cada bloque, llamados PUERTO A y PUERTO E. El usuario que este familiarizado con la CT6811 se dará cuenta del motivo de esta nomenclatura. El *bus* A (Puerto A) es el que maneja el bloque A, es decir motores e infrarrojos, el *bus* E (Puerto E) maneja el bloque E, que tiene las entradas analógico/digitales.

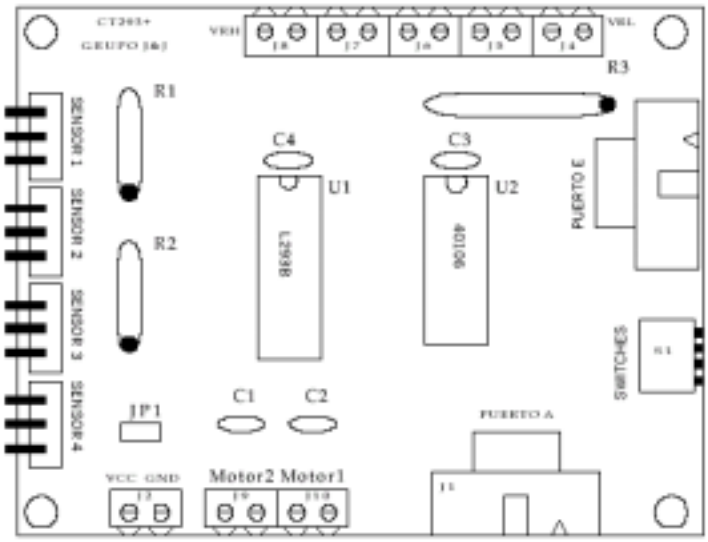


Fig. 12: Situación de los componentes de la CT293+

El bloque A esta formado por el chip L293B (aunque también se puede poner el L293D, que incluye diodos) que gestiona la potencia de los motores y el chip 40106 que tiene dos funciones. La primera consiste en realizar una pequeña lógica para facilitar el uso de los motores, la segunda en realizar una conversión de niveles en la lectura de los infrarrojos. A este bloque también pertenecen los arrays de resistencias de 220 Ω y 47 KΩ que se usan para polarizar los infrarrojos.

El bloque E esta formado por un *array* de ocho resistencias de valor 4K7. Sirve de *pull-up* para las entradas digitales. Más adelante se explica esto con más detalle.

Descripción de los elementos de la CT293+:

- C1, C2: Condensadores en paralelo con los motores. Sirven para eliminar ruido. El valor del condensador dependerá del motor utilizado. No poner condensadores electrolíticos.
- C3, C4: Condensadores de eliminación de ruido para las pastillas integradas.
- S1: *Array* de cuatro *switches*. Acodado. Sirve para anular las entradas de los sensores.
- U2: Pastilla 40106 CMOS inversora. Adapta niveles de entrada de los sensores.
- U1: *driver* de potencia L293B para control de motores.

- PUERTO A, E: Conectores tipo *bus* acodado (5+5 líneas, Macho). Conexión al sistema de control.
- R1: *Array* de resistencias de 4+1, polarización de infrarrojos.
- R2: *Array* de resistencias de 4+1, polarización de infrarrojos.
- R3: *Array* de resistencias de 8+1, *pull-up* de las entradas digitales.
- J2: Clema doble para la alimentación de los motores.
- J4-J8: Clemas dobles para las entradas digitales.
- Motor1, motor2: Clemas dobles donde se conectan los motores.
- JP1: *Jumper* para conexión interna de los motores.
- Sensor1-sensor 4: Conexiones para los sensores de infrarrojos CNY70.

2.3 Instalación de los sensores y motores

La CT293+ puede incorporar bastantes tipos de sensores, también puede manejar motores paso a paso y motores de continua.

2.3.1 Conexión de los motores de continua a la CT293+

La tarjeta esta preparada para poder controlar dos motores de continua simultáneamente. Se realiza por medio del circuito integrado L293B que contiene dos circuitos internos con el "puente en H" necesario para controlar motores. El L239B es un integrado que permite alimentaciones para los motores de hasta +36 v y una corriente de salida de 1 Amperio y que se adapta perfectamente al sistema, ofreciendo una etapa de control y de potencia en un mínimo espacio.

La conexión de los motores a la tarjeta se realiza a través de las clemas dobles llamadas "motor1" y "motor 2". Estas clemas están situadas en el centro de la parte inferior de la tarjeta (ver figura 13). Los motores se pueden situar lejos de la placa, aunque es aconsejable que el cable de unión no supere los 20 cm, por cuestiones de ruido eléctrico, y pérdida de potencia.

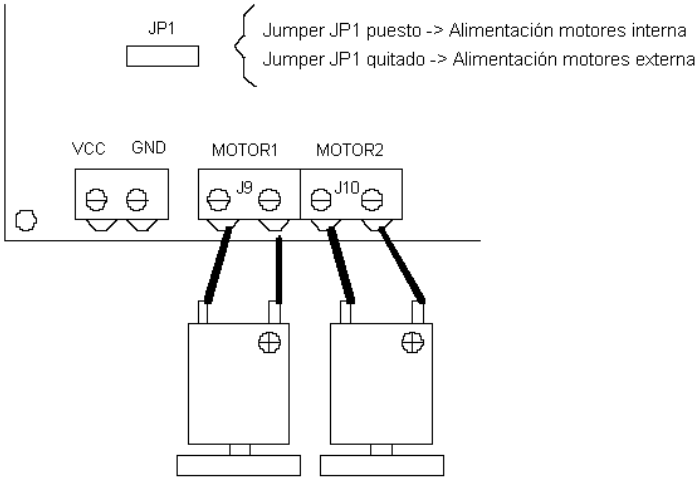


Fig. 13: Conexión de los motores de continua.

La gestión de los motores corresponde al bloque A. Los bits sombreados (del byte de control) son salidas y se encargan del funcionamiento de los motores. Los bits marcados con Dirección seleccionan el sentido de giro. Los bits marcados con ON/OFF indican si se conecta o no el motor (Fig. 14).

Switch 2 Sensor 3	Motor 2 Dirección	Motor 1 Dirección	Motor 2 ON/OFF	Motor 1 ON / OFF	Switch 1 sensor 4	Switch 3 sensor 2	Switch 4 sensor 1	PuertoA S1000
Bit 7-in	Bit 6-out	Bit 5-out	Bit 4-out	Bit 3-out	Bit 2-in	Bit 1-in	Bit 0- in	

Fig. 14: Utilidad de los bits del Puerto A

Siguiendo la forma de conexión anteriormente descrita y mirando los motores de frente se obtiene la siguiente tabla (Fig. 15).

	MOTOR 1	MOTOR 2
DIRECCIÓN	BIT 5 ON (1) - DERECHA OFF (0) - IZQUIERDA	BIT 6 ON (1) - IZQUIERDA OFF (0) - DERECHA
ESTADO	BIT 3 ON (1) - motor ON OFF (0) - motor OFF	BIT 4 ON (1) - motor ON OFF (0) - motor OFF

Fig. 15: Tabla de control de los motores

Se explica ahora la función del *jumper* JP1. Este es el encargado de seleccionar entre la alimentación interna o externa de los motores. Cuando se conecta el *jumper*, la tensión TTL (+5v) se conecta con la entrada de alimentación de los motores. La ventaja es que con una sola fuente de tensión se puede dar energía a todo el sistema. El inconveniente esta en el ruido que los motores introducen en el sistema. Sobretodo aparecen caídas bruscas de tensión que pueden desprogramar la EEPROM interna del 68HC11. Para evitar esto esta el *jumper* JP6 en la CT6811 (LVI), este *jumper* protege a la *eeprom* haciendo un *reset* de la placa siempre que la tensión de alimentación este por debajo de 4.5 v. Esto supone un compromiso. Si se protege la *eeprom* el sistema puede desconectarse automáticamente en las caídas bruscas de tensión, si se quita el *jumper* JP6 de la CT6811 no se produce esa desconexión (salvo caídas muy grandes < 2.5) pero se corre el riesgo de que se desprograme la *eeprom*.

Cuando el *jumper* JP1 de la CT293+ esta quitado significa que se ha seleccionado la alimentación de motores externa. La desventaja es la necesidad de utilizar dos fuentes de alimentación, mientras que la ventaja es el poder poner más tensión a los motores. Por ejemplo se puede utilizar 12v,9v,1v,... sin interferir con la alimentación TTL. Otra ventaja es que no se produce tanto ruido en la línea y el LVI puede ser compatible con los motores.

2.3.2 Conexión de los sensores de infrarrojos a la CT293+

Otro de los elementos que maneja la CT293+ son los infrarrojos CNY70. En concreto esta tarjeta tiene capacidad para controlar directamente cuatro. Si se recurre a las entradas digitales extras se puede llegar a controlar hasta un número de doce. Cada CNY70 incorpora un emisor y un receptor en el mismo encapsulado.

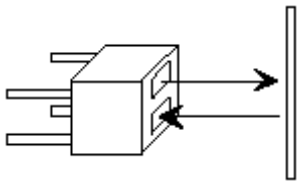


Fig. 16: Sensor de infrarrojos CNY70

Estos infrarrojos son de corta distancia, son capaces de distinguir el negro de otro color mientras la distancia del infrarrojo a la superficie no supere unos pocos centímetros (ver Figura 17). Lo más normal es situarlos a medio centímetro del suelo para evitar interferencias con otras fuentes de luz, ya sea solar o artificial.

La **interpretación** del bit de un sensor es la siguiente:

bit a **1** significa detectado **Negro**
bit a **0** significa detectado **Blanco (No negro)**

Fig. 17: Salida de los sensores de infrarrojos

Se remite al lector al manual de usuario de la tarjeta CT293+ *[Micro1]*, para una información detallada acerca del montaje de los sensores infrarrojos.

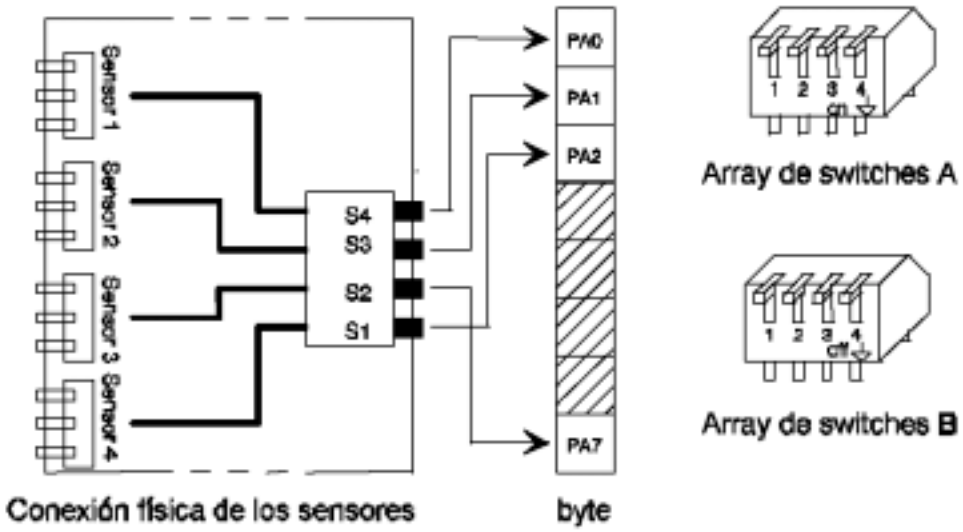


Fig. 18: Conexión de los sensores al Puerto A

Observando el diagrama anterior (Fig. 18) se tiene que los bits en blanco son de entrada y proporcionan el estado de los infrarrojos. Para que la lectura de estos bits sea correcta es necesario activar las salidas de los sensores de infrarrojo correspondientes. Eso se hace con los interruptores que hay en la placa.

Los interruptores (*switches*) que se hayan en la placa tienen la utilidad de anular y dejar libres los bits que emplean los infrarrojos. Si el sistema sólo va a manejar motores se pueden utilizar los bits restantes como entradas sin entrar en conflicto con los sensores. Esto es importante cuando se trabaja con el Puerto A de la CT6811 ya que esos bits son muy útiles y no conviene malgastarlos. Como norma básica se aconseja que los interruptores estén a ON cuando se vayan a manejar los sensores, en caso contrario ponerlos a OFF. La correspondencia entre la localización física del infrarrojo y su interruptor se representa en la figura 19.

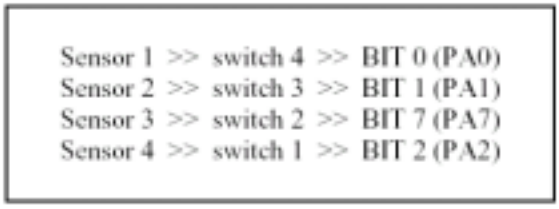


Fig. 19: Correspondencia entre sensores y switches.

2.3.3 Conexión con las entradas digitales

Las entradas digitales se introducen por las clemas de conexión situadas en la parte superior de la tarjeta. Hay un total de 10 entradas pero sólo 8 son entradas, las otras dos se usan para establecer los niveles de referencia VRH y VRL. En modo digital estos niveles serán VRL=0 y VRH=5. Si se trabaja con la CT6811 y se tienen los *jumpers* JP1 y JP2 puestos los niveles se sitúan automáticamente en VRL=GND="0" y VRH=VCC="5".

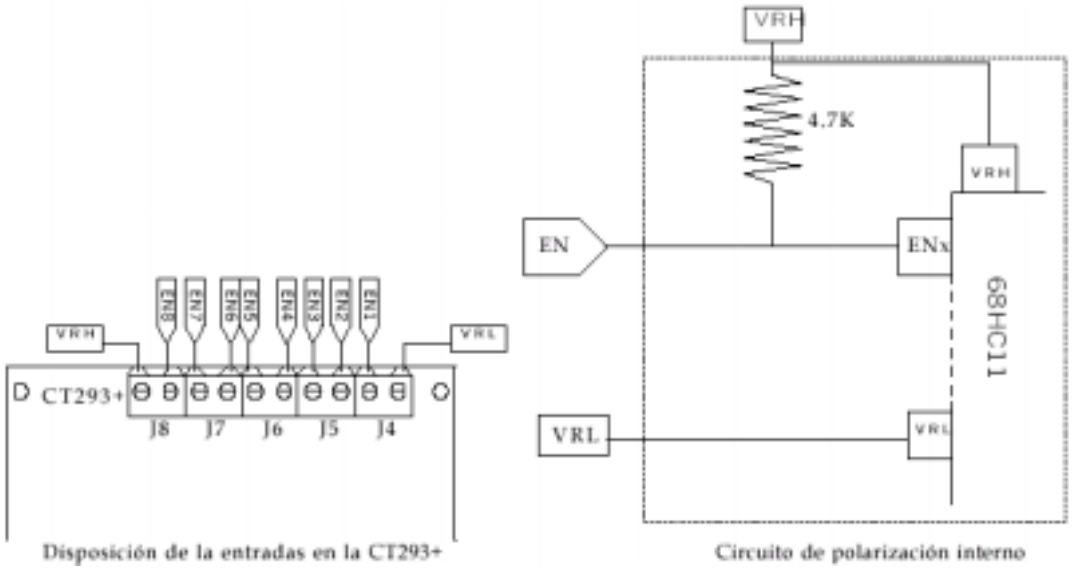


Fig. 20

En la figura 20 se presenta el circuito de polarización desarrollado en la tarjeta CT293+ y la disposición de las entradas. Según el circuito de polarización la lectura que se obtiene cuando no se conecta nada en las entradas, estando los niveles de referencia a VCC y GND, es prácticamente VRH, es decir VCC. Lo que significa nivel alto "1". Este comportamiento se debe a la resistencia de 4.7 K, que se denomina resistencia de *pull-up*.

2.3.4 Conexión con las entradas analógicas

Las entradas digitales que se explican en el apartado anterior pueden configurarse desde la CT6811 como entradas analógicas. Ahora se indica como utilizar correctamente las entradas y su circuito de polarización. Para utilizar este modo es necesario disponer la CT6811 y además configurarla para la utilización de los conversores analógico-digitales.

El Puerto E del 68hc11 puede funcionar de dos formas distintas. La primera se corresponde con el funcionamiento en modo digital explicado en el apartado anterior. La segunda es el modo analógico que se explica aquí. Los dos utilizan el mismo circuito de conexión para sus entradas y lo que era una ventaja en modo digital ahora se convierte en un inconveniente. La resistencia de *pull-up* que antes facilitaba la conexión de ciertos dispositivos, ahora hace que la señal analógica sufra una distorsión no deseable.

Las entradas VRH y VRL son los niveles de referencia que necesita el conversor analógico digital del 68HC11 para su correcto funcionamiento. Lo normal es utilizar VRH=VCC y VRL=GND, por eso esta opción se encuentra disponible en la CT6811 con tan solo conectar los *jumpers* JP1 y JP2 (situados en la esquina superior derecha). Si se desconectan hay que introducir los niveles por las entradas correspondientes. El valor de estos debe estar comprendido entre (0V – 6V). Si se superan los 6 V las conversiones pueden ser erróneas, y si se introduce una tensión por debajo de cero, negativa, puede dañar permanentemente la entrada de nivel analógica del microcontrolador 68HC11. Además por las entradas analógicas no hay que introducir nunca tensiones negativas, pues probablemente se rompería dicha entrada. En el siguiente cuadro se incluyen los aspectos más importantes a tener en cuenta al utilizar los conversores analógico-digitales.

Los mayores errores cometidos en aplicaciones con los conversores se deben:

1) Impedancia de carga a la entrada analógica superior a 10kΩ, produce una medida errónea de la conversión. Se evita no superando los 10kΩ.

2) Impedancia de carga a la entrada demasiado pequeña que produce una corriente de entrada superior a los 25mA. Puede dañar permanentemente la entrada utilizada del 68HC11. Se evita colocando una resistencia que haga que la corriente de entrada sea inferior a los 25mA.

3) Introducir una tensión negativa por la entrada cuando hay una impedancia de carga de bajo valor conectada a ella. Puede dañar permanentemente la entrada utilizada del 68hc11. La mejor forma de evitar esto es no introducir nunca tensiones negativas. Tampoco en VRL.

2.4 Programación de la tarjeta CT293+

La tarjeta CT293+ esta pensada para conectarse a la tarjeta CT6811. Aunque se puede conectar a otros tipos de controladores. El rendimiento máximo de la tarjeta se saca cuando se conecta el puerto A y el puerto E de la CT6811. El control de la CT293+ se realiza a través de dos bytes. Un byte controla los motores y sensores de infrarrojos y el otro las entradas digitales/analógicas.

Se divide este apartado en dos partes, en la primera se explica como controlar el bloque A de la CT293+ (motores e infrarrojos) y en la segunda se explica como usar el bloque E (entradas digitales/analógicas).

2.4.1 Programación del bloque A. Motores y sensores de infrarrojos.

Lo primero es unir el bloque A de la CT293+ con la CT6811 utilizando el *bus* A., para ello conectar un cable de *bus* entre el Puerto A de la CT293+ (conector J1) y el Puerto A de la CT6811 (conector J1). Aquellas personas que no poseen la CT6811 tienen que conectar el Puerto A de la CT293+ con el puerto correspondiente en su sistema de control.

Automáticamente al colocar dicho cable se proporcionará la alimentación TTL a la CT293+. Ahora hay que elegir una de las dos opciones siguientes. La primera consiste en utilizar esa alimentación TTL para alimentar los motores, para ello hay que colocar en su sitio el *jumper* JP1 de la CT293+. La segunda consiste en alimentar los motores con otra fuente de alimentación, para ello hay que desconectar el *jumper* JP1 e introducir la alimentación externa por la clema J2. Tener cuidado con la polaridad.

Al utilizar la alimentación externa tener mucho cuidado con no olvidar desconectar el *jumper* JP1 y con la polaridad de la tensión de alimentación de los motores. Si se utiliza alimentación única antes de colocar el *jumper* JP1 desconectar la alimentación externa de los motores (clema J2).

Una vez realizado lo anterior el control de la CT293+ es muy sencillo. Tan solo se utiliza un byte de control, que se corresponde con el Puerto A de la CT6811, posición de memoria $1000 del microcontrolador 68HC11. El significado de los bits de dicho byte se detalla a continuación, donde los bits en blanco corresponden con entradas y los sombreados con salidas.

Switch 2 Sensor 3	Motor 2 Dirección	Motor 1 Dirección	Motor 2 ON/OFF	Motor 1 ON / OFF	Switch 1 sensor 4	Switch 3 sensor 2	Switch 4 sensor 1	PuertoA $1000
Bit 7-in	Bit 6-out	Bit 5-out	Bit 4-out	Bit 3-out	Bit 2-in	Bit 1-in	Bit 0- in	

Fig. 21: Utilidad de los bits del Puerto A

2.4.2 Programación del Bloque E: Entradas digitales/analógicas

Lo primero que se tiene que hacer es unir el Bloque E de la CT293+ con la CT6811 utilizando el *bus* E. Se conecta un cable de *bus* entre el Puerto E de la CT293+ (conector J3) y el Puerto E de la CT6811 (conector J3). El byte de control de este bloque esta situado en la posición de memoria $100A, que se corresponde con el puerto E.

Este cable de *bus* tiene diez hilos, ocho de ellos se corresponden con bits de entrada y los otros dos se utilizan cuando se trabaje con las entradas analógicas. La principal función del bloque E es la de proporcionar ocho entradas con *pull-up*. Si no hay nada conectado en ellas se lee en los bits correspondientes un nivel alto "1". Cuando se conecte alguna señal en la entrada dependiendo de su valor se obtiene un nivel alto o bajo.

Cuando se utiliza la CT6811 el Puerto E se puede utilizar de dos formas diferentes. La primera se corresponde con lo anterior, utilizar el puerto E como entradas digitales con *pull-up*. La segunda es configurar dicho puerto para leer canales analógicos. Aunque el bloque E no se pensó para usarlo de forma analógica las pruebas realizadas con diferentes sensores de este tipo dieron buenos resultados. El inconveniente es que debido a las resistencias de *pull-up* hay una distorsión en la medida. Distorsión por otro lado conocida y por tanto corregible por hardware o por software.

A continuación se detallan dos ejemplos, correspondientes a la utilización del Puerto E en modo digital y en modo analógico respectivamente. Estos ejemplos se han considerado interesantes, porque en ellos se ilustra la utilización de los bumpers y del sensor de luz que podemos encontrar en el Microbot Clónico, que es el utilizado en este proyecto.

2.4.2.1 Ejemplo de utilización en modo digital (leyendo los sensores de contacto):

El bloque E esta especialmente diseñado para poder conectar pulsadores sin ningún esfuerzo en la tarjeta. Un tipo de pulsador puede ser un *bumper* o interruptor de fin de carrera. En la Figura 20 se recuerda su estructura interna y el circuito de polarización al conectarse a la CT293+. Viendo dicho circuito se aprecia que cuando no se activa el *bumper*, la lectura que se obtiene en la entrada digital es nivel alto "1". Cuando se activa el *bumper* la entrada se cortocircuita con VRL, lo que implica que ahora la lectura sea nivel bajo "0". Se recuerda también que los niveles de referencia VRL y VRH se tienen que definir salvo que se este utilizando la CT6811 con los *jumpers* JP1 y JP2 conectados. En este caso por defecto se tiene VRH=VCC y VRL=GND.

Si no se conectan los *jumpers* se tienen que establecer los niveles externamente. El rango de tensiones admisible es de 0 V a 5 V. Utilizando las entradas en modo digital estos valores son VRL=0 V y VRH=5 V. En modo analógico es cuando más se suelen utilizar otras referencias distintas.

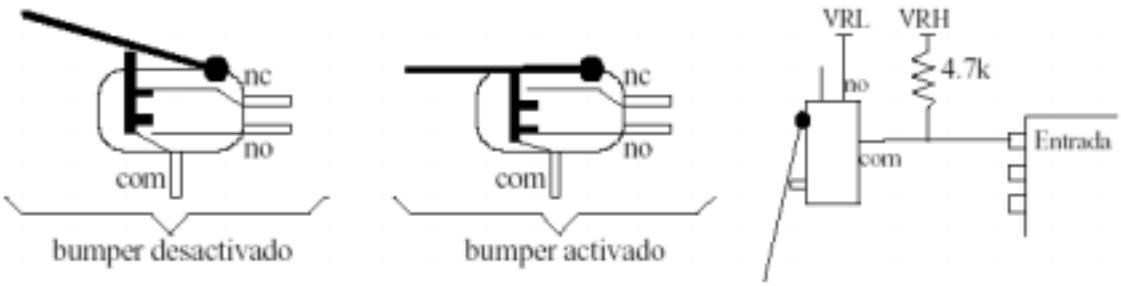


Fig. 22: Funcionamiento de los jumpers y circuito de polarización.

En el circuito de polarización propuesto (Fig. 22) cuando el *bumper* esta inactivo se lee nivel alto a la entrada debido a la resistencia de *pull-up*. Cuando se activa se lee nivel bajo pues la entrada se ha cortocircuitado con VRL dentro del *bumper*. Este circuito no solo es válido para *bumpers* sino que también se puede conectar cualquier circuito exterior al cual no le afecte tener una resistencia de *pull-up* a la salida.La entrada 8 se corresponde con el bit ocho del byte de control E situado en la posición de memoria $100A del microcontrolador 68hc11. Al apretar la palanca en la entrada se lee nivel bajo "0", si no se aprieta se lee nivel alto "1".

El programa que se muestra a continuación como ejemplo, comienza haciendo avanzar al microbot, y en cuanto se pulsa el sensor de contacto 1 el microbot se para, siendo necesario pulsar el sensor de contacto 2 para volver a ponerlo en marcha. Lo importante es saber que cuando un sensor de contacto esta activado, es decir se esta apretando, el bit que refleja su estado estará a nivel alto '1', mientras que si no esta activado estará a nivel bajo '0'. El switch 'programa ½' funciona como un sensor de contacto. Se supone que la conexión es la misma que la figura de arriba con los niveles VRH=VCC y VRL=GND.

```
; ejemplo 1: El microbot se para cuando pulsemos el  sensor de contacto 2,y
; se pone en marcha cuando pulsemos el otro.
; el sensor de contacto_1 se lee en el PUERTO E, bit PE0
; el sensor de contacto_2 se lee en el PUERTO E, bit PE1

PORTA   EQU $0
PORTE   EQU $A

        ORG $0       ; programa para la RAM interna

        LDX  #$1000  ; cargamos en el Registro X el valor $1000

avanza LDAA #$18
       STAA PORTA,X  ; PORTA + X = $0 + $1000 = $1000 = PUERTO A
       LDAA PORTE,X  ; guardo en A el contenido del PUERTO E
       ANDA #$01     ; Me quedo con el valor del bit PE0,sensor contacto 1
       CMPA #$1      ; ¿Esta a nivel alto? = ¿ Esta pulsado el sensor?
       BNE  avanza   ; la comparación no es cierta -> no se para

para  CLRA           ; la comparación es cierta, se ha pulsado el sensor
      STAA PORTA,X   ; paramos el microbot desactivando los bits del PUERTO A
      LDAA PORTE,X   ; guardo en A el contenido del PUERTO E
      ANDA #$02      ; Me quedo con el valor del bit PE1,sensor_contacto2
      CMPA #$2       ; ¿Esta a nivel alto? = ¿ Esta pulsado el sensor?
      BNE  para      ; la comparación no es cierta ->no se activa microbot
      BRA  avanza    ; la comparación es cierta, se activa el microbot

      END
```

2.4.2.2 Ejemplo de utilización en modo analógico (lectura del sensor de luz):

En este ejemplo el Microbot avanzará mientras reciba la suficiente intensidad de luz por su sensor de luz. Para poder leer el sensor de luz será necesario programar el conversor analógico-digital que posee el MC68hc11 en su interior. Para ello conviene leer la página 31 del manual de usuario de la CT293+ *[Micro1]*.

Para desarrollar este ejemplo tan sólo necesitaremos saber la entrada del sensor luz, que como ya se ha indicado corresponde al bit PE3, del puerto E.

```
; Ejemplo 2: El microbot avanza cuando recibe luz por el sensor LDR.
; El sensor estar  conectado a la entrada PE3 (canal 4).

OPTION  EQU $39
ADCTL   EQU $30
PORTA   EQU $0
ADR4    EQU $34  ; canal 4

        ORG $0
inicio
        LDX  #$1000
        LDAA #$80
        STAA OPTION,X             ; encender el conversor A/D
        LDAA #$23                 ; configuración del conversor
        STAA ADCTL,X              ; SCAN -> activo
                                  ; MULT -> inactivo
                                  ; ADR4 -> seleccionar canal 4
espera  BRCLR ADCTL,X $80 espera  ; espera a que termine la conversión
        LDAA  ADR4,X              ; leer el resultado de la conversión
        CMPA  #$E0                ; comparar un el umbral
        BLO   avanza              ; si rebaso el umbral avanza
        CLRA
        STAA  PORTA,X             ; no rebaso umbral: parado
        BRA   espera              ; vuelve a realizar conversión

avanza  LDAA  #$18                ; supero umbral: avanzamos
        STAA  PORTA,X
        BRA   espera              ; vuelve a realizar conversión

        END
```

3 Localización lógica de los actuadores del Microbot Clónico

A continuación se muestra una tabla donde se puede encontrar para cada actuador (motor o sensor) del Microbot Clónico, su situación en el mapa de memoria del 68HC11 y la función que realiza.

Actuador	Localización	Dirección memoria	Descripción
Motor 1 On/Off	Puerto A: bit PA3	$1000 : $08	Activa motor 1: mueve oruga 1
Motor 1 Dirección	Puerto A: bit PA5	$1000 : $20	Cambia el sentido de giro motor_1
Motor 2 On/Off	Puerto A: bit PA4	$1000 : $10	Activa motor 2: mueve oruga 2
Motor 2 Dirección	Puerto A: bit PA6	$1000 : $40	Cambia el sentido de giro motor_2
Sensor Infrarrojo 1	Puerto A: bit PA7	$1000 : $80	Distingue blanco/negro
Sensor Infrarrojo 2	Puerto A: bit PA1	$1000 : $02	Distingue blanco/negro
Sensor Contacto 1	Puerto E: bit PE0	$100A : $01	Detecta contacto con obstáculo
Sensor Contacto 2	Puerto E: bit PE1	$100A : $02	Detecta contacto con obstáculo
Sensor de Luz	Puerto E: bit PE3	$100A : $08	Detecta intensidad de luz
Switch Programa ½	Puerto E: bit PE2	$100A : $04	Permite conmutar programa (solo para el programa original incluido el microbot)

Fig. 23: Actuadores del Microbot Clónico.

La situación en el Microbot de todos estos actuadores se encuentra representada en la figura 1, en el capítulo anterior.

CAPÍTULO 3. DEFINICIÓN Y ANÁLISIS DEL SISTEMA

1 DEFINICIÓN

1.1 Objetivos

Se desea diseñar un sistema que permita realizar el control desde un ordenador personal (PC) de uno o más Microbots mediante radiofrecuencia. La comunicación que permitirá realizar dicho control será bidireccional entre el PC y cada Microbot, según se indica en la figura 24.

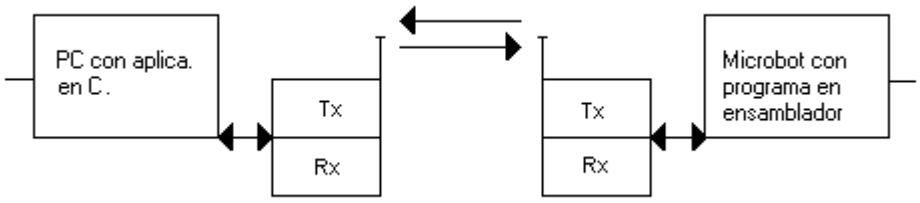


Fig. 24: Esquema general de la comunicación entre el PC y el Microbot.

A continuación se indican las partes que compondrán el sistema en sus dos extremos, PC y Microbot.

- EXTREMO PC:

  - Parte Software: Una aplicación que "correrá" bajo el PC programada en lenguaje C y que deberá proporcionar una interfaz visual al usuario de forma que cualquier persona podrá utilizar el sistema y controlar los Microbots, enviándoles comandos y recibiendo la información procedente de estos, utilizando un protocolo de comunicación a diseñar.

  - Parte Hardware: Un módulo Transmisor/Receptor estático conectado al PC, y que será el encargado de transmitir los comandos y recibir la información necesaria para controlar los Microbots. Este deberá tener un alcance efectivo de al menos una decena de metros, básicamente lo suficiente para efectuar el control de los Microbots en espacios reducidos, normalmente cerrados y con distancias cortas, pero donde no es adecuada la utilización de cables.

- EXTREMO MICROBOT:

  - Parte Hardware: Un módulo Transmisor/Receptor a incluir en el sistema móvil Microbot que reciba los mensajes procedentes del PC y transmita los datos generados por el Microbot. Su estructura mecánica debe ser la adecuada para que se adapte el Microbot *Clónico*, esto es, crecimiento hacia arriba.

- Parte Software: Una aplicación que se ejecute en el Microbot, la cual soporte el protocolo de comunicación a diseñar y que deberá ser capaz de interpretar y procesar los comandos e información procedentes del PC y de enviarle a éste los datos solicitados. Esta aplicación será programada en el lenguaje ensamblador propio del microcontrolador 68HC11 y será almacenada en la memoria no volátil del Microbot, para permitir el funcionamiento autónomo de éste.

1.2 Condiciones de entorno

1.2.1 Características mecánicas

La característica mecánica que se exigirá al sistema estará determinada por la estructura del Microbot. Es decir, debido a la estructura de crecimiento hacia arriba, en la que se colocan las placas de circuito impreso unas encima de las otras separadas por tornillos separadores, las dimensiones de la PCI a diseñar vienen determinadas por los soportes de las placas de circuito impreso incluidas en éste, además aunque esta pueda sobresalir en alguna dimensión con respecto a las demás, no es deseable que sobrepase los límites físicos de la estructura del robot, quedando asimismo esta posibilidad limitada por la situación de los conectores incluidos en el sistema y que interconectan los distintos elementos de este. Así pues, para el Microbot *Clónico*, las dimensiones de la PCI a diseñar vendrán limitadas por la superficie de las demás placas (la limitada por los tornillos separadores) y por la longitud total del robot, pues la solución más sencilla de tener que ampliar las dimensiones de la PCI, es de hacerla crecer hacia la parte frontal del Microbot.

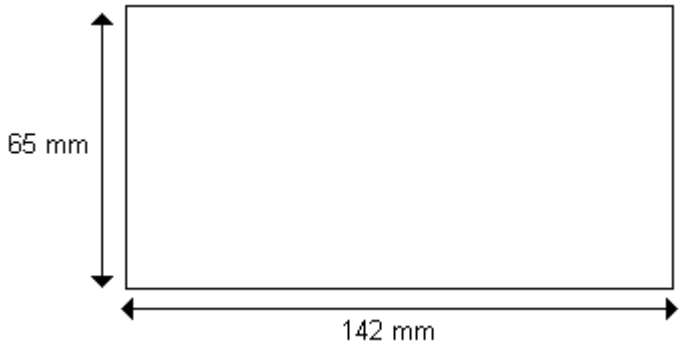


Fig. 25: Dimensiones máximas de la PCI del Microbot.

1.2.2 Compatibilidad electromagnética

Debe tenerse en cuenta que el sistema debe ser capaz de funcionar cumpliendo las normas referentes a interferencias electromagnéticas (EMI). De utilizar para la transmisión una de las bandas ICM, habrá que considerar que la potencia radiada máxima en estas bandas en Europa es de 100 mW .

1.3 Especificaciones

Como conclusión de los apartados anteriores, se indican a continuación las especificaciones de las distintas partes que compondrán el sistema y que han de permitir realizar el control de los Microbots desde un PC mediante una interfaz inalámbrica:

- Aplicación:

  Interfaz de Panel para el usuario, desde donde éste podrá enviar comandos de movimiento, así como solicitar información acerca de sus sensores y de otros tipos al Microbot. Asimismo, el usuario podrá visualizar en pantalla una representación de la ruta seguida por el robot. Además este SW debe permitir controlar simultáneamente varios Microbots.

- Conexión del módulo Transmisor/Receptor al PC:

  Conexión a uno de los puertos exteriores del PC, debido al interés en tener la posibilidad de conectar este módulo fácilmente a distintos equipos (el puerto del PC debe ser bidireccional para permitir la comunicación en ambos sentidos).

- Transmisión de datos:

  Transmisión bidireccional de datos digitales por radiofrecuencia utilizando una banda sin licencia. Con un alcance de al menos una decena de metros y cumpliendo con la normativa vigente relativa a compatibilidad electromagnética.

- Protocolo de comunicación:

  Protocolo que asegure la transmisión y recepción correcta de los comandos y de la información en uno y otro sentido.

- Dimensiones máximas de la PCI a montar en el Microbot:

  Estas serán de 65 mm x 142 mm, quedando determinadas por la separación entre los soportes de las placas y por la longitud total del Microbot.

- Entrada y salida de datos al Microbot:

  A través de alguno de los puertos bidireccionales (es decir configurables como entradas/salidas) del microcontrolador 68HC11, requiriéndose tanto líneas o bits para datos, como líneas de control para el transmisor/receptor.

- Tamaño máximo del programa ensamblador:

  Como máximo 512 Bytes (Tamaño de la memoria EEPROM del Microbot, única memoria no volátil que podemos utilizar sin recurrir a ampliaciones de memoria).

- Sistema modular y fácilmente ampliable y configurable.

2 ANÁLISIS

2.1 Diagrama general del sistema

En la siguiente figura se muestra un diagrama general del sistema, que constituye un sistema de control local, donde el PC actúa como la base y el Microbot como uno de los móviles.

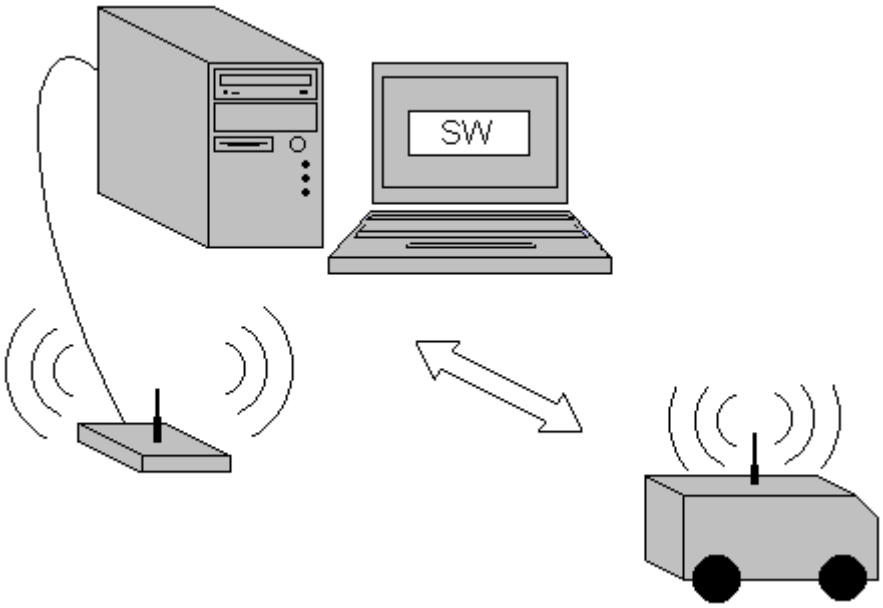


Fig. 26: Diagrama general del sistema.

2.2 Diagrama de bloques de primer nivel

En este diagrama se observa como el sistema recibe comandos a través del PC y como mediante la utilización de un enlace radio, dicho sistema genera una serie de acciones en el Microbot y devuelve una cierta información.

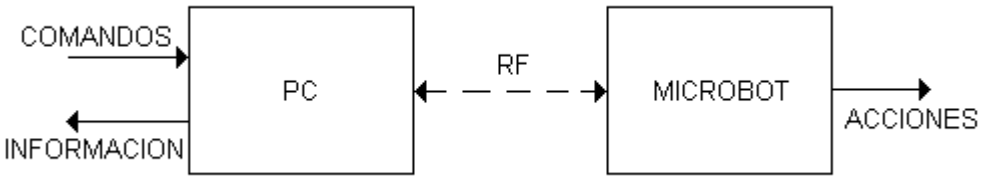


Fig. 27: Diagrama de bloques de primer nivel.

2.3 Diagrama de Entrada/Salida del sistema

En el siguiente diagrama se observa como al sistema le llegan como entradas los comandos de movimiento y peticiones de información dirigidos al Microbot, junto con la alimentación, las interferencias electromagnéticas (ruido) y demás condiciones ambientales. Generando el sistema como salidas el movimiento de los motores del Microbot, así como la información procedente de éste (sensores y estado) y las indicaciones luminosas indicando que se ha producido la transmisión o recepción de un dato.

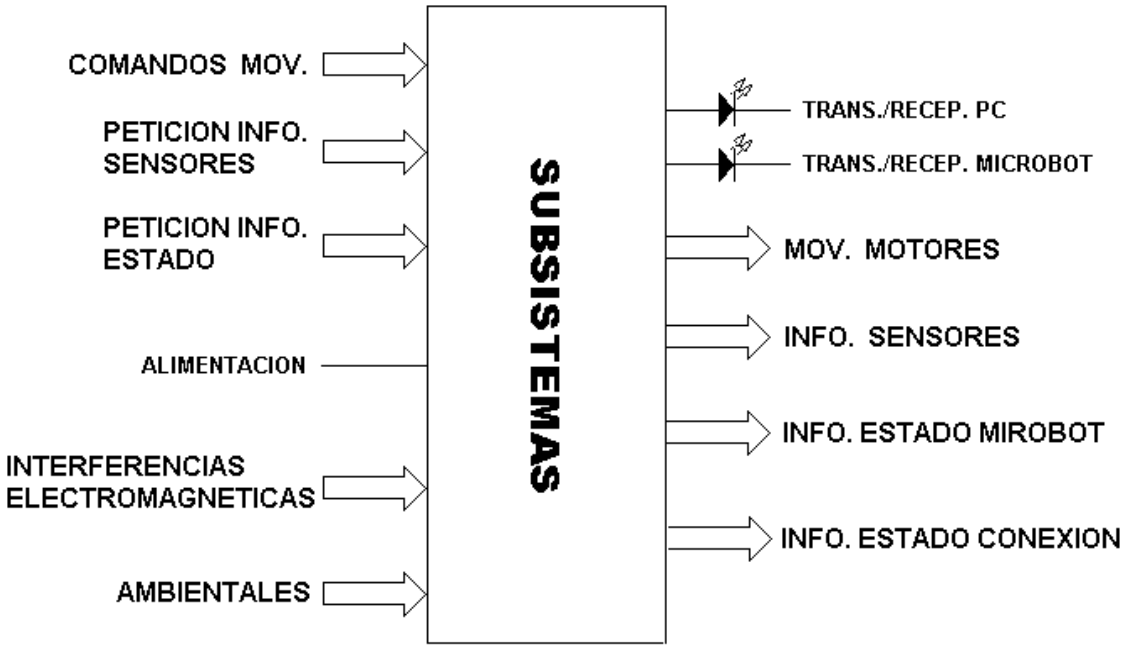


Fig. 28: Diagrama de E/S del sistema.

2.4 Diagrama de bloques de segundo nivel

En la figura 29 se muestra un diagrama del sistema dividido en bloques de segundo nivel. En él se puede observar como los datos generados en las comunicaciones viajan entre los transmisores y receptores conectados al PC y al Microbot .

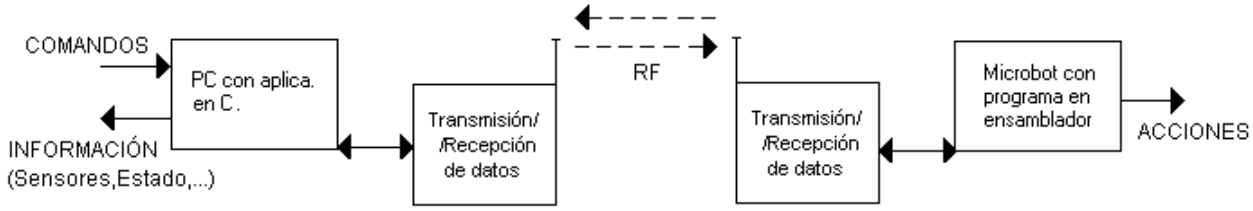


Fig. 29: Diagrama de bloques de segundo nivel.

2.5 Alternativas de diseño

A continuación se muestran las diferentes opciones consideradas para la realización de los distintos bloques que constituirán el sistema, junto con la justificación de la opción elegida en cada caso.

- ALIMENTACIÓN TRANSMISOR/RECEPTOR CONECTADO AL PC:

a. - Fuente de alimentación interna incluida en la PCI de 5 V (TRF + puente rectificador + regulador + condensadores de filtrado), con alimentación externa a 220 V A.C.

b. – Alimentación mediante baterías o pilas tipo comercial de 1.5 V, con o sin entrada de alimentación externa de 5V D.C.

c. – Toma de alimentación de 5V D.C. a suministrar desde el exterior.

d. – Alimentación a través del puerto de comunicaciones del PC utilizado (solo es posible con algunos tipos de puerto).

Respecto a la alimentación de la tarjeta conectada al PC, la opción elegida entre las anteriores ha sido la de una **toma de alimentación externa de 5V D.C**, suministrables desde el exterior por una fuente de alimentación o baterías, para no complicar el diseño de la PCI y utilizar un esquema de alimentación similar al utilizado en los robots de Microbotica . La tarjeta a conectar al robot tomará por su parte la alimentación del propio sistema Microbot.

- CONEXIÓN AL PC

a- Puerto paralelo (IEE1284)

b- Puerto serie (RS232)

c- Puerto USB (Universal Serial Bus)

Para la conexión del sistema al PC se utilizará el **Puerto Paralelo (IEEE1284)**, en uno de sus modos bidireccionales, el llamado modo Nibble, el cual permite una entrada de 4 bits de datos, junto con una línea de interrupción, una salida de 8 bits de datos y algunos bits más de control. Se utilizará este puerto en dicho modo por su acoplamiento directo con el codificador y decodificador, sin necesidad de lógica alguna de interfaz.

- CONEXIÓN AL MICROBOT (AL MICROCONTROLADOR 68HC11)

    a- Puerto B (8 bits bidireccionales)

    b- Puerto C (8 bits bidireccionales)

    c- Puerto D (6 bits bidireccionales)

Entre las opciones anteriores se decide se utilizar el **PUERTO D del 68HC11**, pues sus 6 bits se pueden configurar como entradas o salidas independientemente, así parte de estos bits se utilizarán como señales de control y el resto como entrada/salida de datos, con lo que se necesitará de una lógica adicional de interfaz y control. Además con la utilización del PUERTO D se dejan libres los Puertos B y C, necesarios ambos para poder utilizar memoria externa al microcontrolador. Se requerirá además del Puerto 'CONTROL' proporcionado por la tarjeta CT6811, que contiene pines de alimentación y masa, así como otros pines útiles del 68HC11 como son las entradas de interrupción externa.

- FAMILIA LOGICA DE LOS CHIPS DIGITALES

    a- Familia TTL (S, LS, ALS, FAST,…).

    b- Familia CMOS (HC,HCT,..).

    c- Combinación de las anteriores, con posible necesidad de adaptación de niveles.

Entre las distintas familias lógicas se ha decidido utilizar la **HCMOS**, quedando la posibilidad de utilizar HCTMOS allí donde se requiera compatibilidad con niveles lógicos TTL. Se empleará esta tecnología sobre todo por su menor consumo respecto a las familias TTL, así como por su difusión, por sus prestaciones más que suficientes para la aplicación deseada y también por ser la utilizada en el Microbot (el microcontrolador 68HC11 está fabricado en esta tecnología) y en los codificadores y decodificadores elegidos.

- CODIFICADORES Y DECODIFICADORES DE DATOS

    a- Codificador de datos MOTOROLA MC145026 y decodificador MC145027, especiales para aplicaciones de control remoto.

    b- Codificación propia de la norma RS232.

Para la codificación y decodificación de los datos se utilizará el juego de codificador–decodificador **MC145026-MC145027** de Motorola, por la sencillez de su utilización, disponibilidad y su especialización en aplicaciones de control remoto, así como por la experiencia en su utilización en otros

proyectos en el departamento de Tecnología Electrónica de la Facultad de Ing. de Telecomunicación de la Universidad de Málaga.

- BANDA DE TRANSMISIÓN DE RF

a- 27 MHZ (frecuencia publica para radioaficionados)

b- 433.05 – 434.87 MHZ (frec ICM)

c- 868 – 870 MHZ (frec ICM)

d- 902 – 928 MHZ (frec ICM)

e- 2400 – 2483.5 MHZ (frec ICM)

f- 5725 – 5850 MHZ (frec ICM)

Para la transmisión de la señal de RF se utilizará la banda ICM de **433.05 a 434.87 MHZ**, ya que a pesar de ser una de las de ancho de banda estrecho dentro de las frecuencias ICM, esté es más que suficiente para la aplicación deseada, existiendo dispositivos comerciales disponibles para transmitir en esta banda y no existiendo en ella la saturación de uso de otras de las bandas como es la de 2.4 GHz (Bluetooth, IEEE 802.11b, HomeRF, etc).

- TRANSMISIÓN - MODULACIÓN RADIO

a- Transmisor AM marca AUREL y receptor misma marca, ambos en la banda de 433 MHZ, especiales para la transmision de datos digitales, con modulación AM On/Off.

b- Transmisor FM marca AUREL y receptor superheterodino misma marca, ambos en la banda de 433 MHZ.

c- Transmisor/Receptor marca NORDIC con modulación digital FSK hasta 76.8 Kbits/s y receptor misma marca, en distintas bandas ICM.

d- Chips Bluetooth con ancho de Banda máximo teórico de 1Mbits/s (usual máximo de 720 Kbits/s (Banda de 2.4 MHz). Con alcance mínimo de 10 metros y posibilidad de aumento hasta 100 metros.

e- Chips standard DECT para transmisión digital.

f- Transmisor y receptor realizados a base de componentes discretos especiales para RF, usando modulaciones analógicas (AM o FM) y diseñados para la banda deseada de frecuencias.

g- Transmisor y receptor diseñados específicamente, utilizando algún tipo de modulación digital (FSK, PSK, etc).

Se utilizarán **Transmisores AM marca AUREL y receptores de la misma marca, ambos en la frecuencia 433,92 MHZ, con modulación AM On/Off**. Se emplearán estos módulos por su sencillez para incluirlos en el diseño, especialización para la transmisión de datos digitales, disponibilidad y sobre todo por haber sido los utilizados en un proyecto anterior realizado en el seno del Departamento de Tecnología Electrónica de la Facultad de Ing. de Telecomunicación de la Univ. de Málaga, en el que se implementaba un sistema de transmisión unidireccional de datos vía radio desde un PC hacia un Microbot *[PFC1]*. Aunque con un ancho de banda muy reducido, estos módulos son suficientes para la consecución del objetivo de este proyecto, el cual es la implementación de un enlace radio bidireccional, dejándose para proyectos posteriores el aumento en la velocidad de transmisión del enlace. Su utilización determinará que la ***comunicación*** será ***simplex bidireccional***, ya que se utiliza la misma frecuencia (canal) para transmitir en ambos sentidos.

- ANTENAS

Los posibles tipos de antenas considerados han sido:

a- Antena unipolo, enrollada o no en espiral.

b- Antena tipo dipolo.

c- Antena diferencial en bucle mediante pistas de Circuito Impreso.

Asimismo, respecto a la conmutación de la antena entre transmisión y recepción se han considerado las siguientes alternativas:

a- Relé de antena tipo Reed con tensión de control de 5 Voltios.

b- Switch analógico construido con tecnología CMOS con tensión de control de 5 Voltios.

c- Conmutador de antena marca AUREL, especial para la banda de 433 MHz, compatible con cualquier transmisor y receptor de la línea TÓTEM de AUREL y con tensión de control entre 5 y 12 Voltios.

d- Duplexor para permitir la transmisión y recepción simultanea por la misma antena.

e- 2 antenas separadas, una para Transmisión y otra para Recepción (no es posible para todos los tipos de modulación).

Se ha decidido utilizar **antenas de tipo unipolar de ¼ de onda (17 cm para 433 MHz),** por su sencillez y bajo coste, además de ser requerida una antena unipolar por el receptor y transmisor elegidos. Asimismo debido a estas mismas razones y al permitirlo el tipo de modulación a utilizar (AM On-Off), se emplearán **antenas separadas para recepción y transmisión** para no complicar el diseño con un duplexor, o tener que utilizar relés (retardos, consumo y vida limitada) o switches analógicos CMOS (atenuación de la señal de radiofrecuencia).

- SOPORTE FISICO DEL CIRCUITO

  a- Placa agujereada para conectar los componentes a través de hilos de wrapping. Con componentes insertados.

  b- Placa de circuito impreso a una sola cara. Con componentes insertados.

  c- Placa de circuito impreso a dos caras. Con componentes insertados y/o de montaje superficial.

  d- Placa de circuito impreso con más de dos capas, incluyendo planos de masa y alimentación. Con componentes insertados y/o de montaje superficial.

El circuito se montará en una **Placa de Circuito Impreso a dos caras** y con componentes insertados, por la mayor o exclusiva disponibilidad de los componentes a utilizar en tecnología PTH, además ésta es suficiente para el nivel de integración que se desea. Se descarta el montarlo en una PCI de una sola capa por la dificultad de las conexiones en cuanto aumenta el número de estas, y las de más capas por su complejidad y por tener que recurrir a empresas externas a la Universidad para su fabricación, con el aumento de costes que esto conlleva, pudiéndose realizar PCIs a dos caras en los propios laboratorios del Departamento de Tecnología Electrónica de la Facultad de Ing. de Telecomunicación de la Universidad de Málaga.

2.6 Diagrama de bloques tercer nivel

Una vez elegidos los elementos que formarán los distintos bloques en los que se puede dividir el sistema completo se llega al diagrama de bloques de tercer nivel que se muestra a continuación.

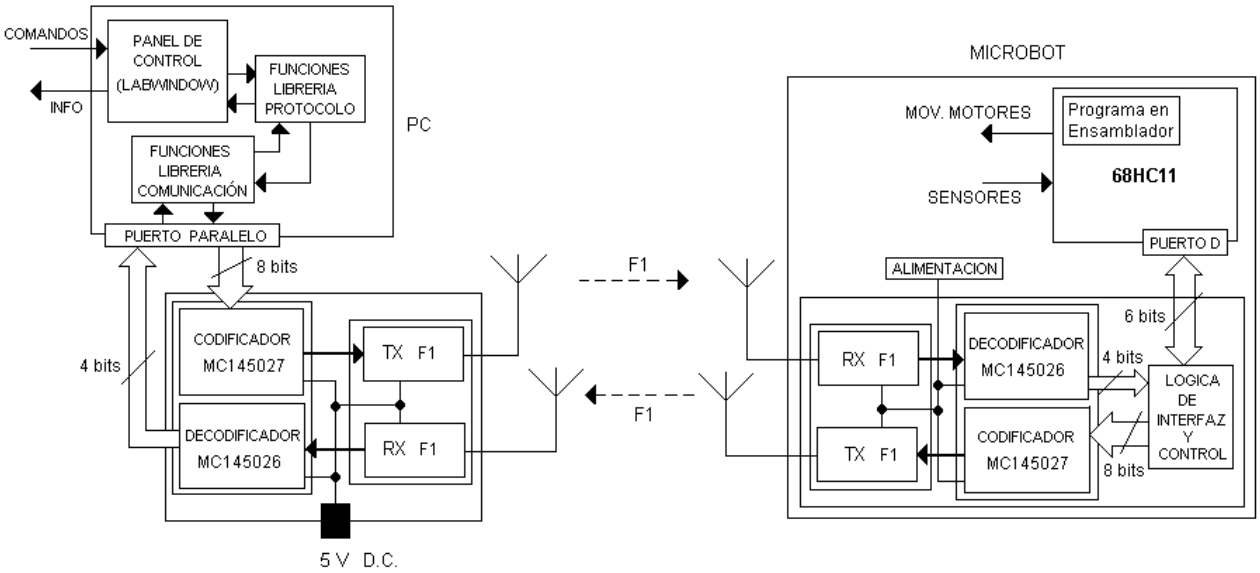


Fig. 30: Diagrama de bloques de tercer nivel.

2.7 Descripción de los bloques

- PANEL DE CONTROL

Una aplicación programada en C (LABWINDOWS) mostrará en pantalla un panel con los elementos interactivos (botones, indicadores,...), necesarios para que el usuario envíe al robot los comandos y visualice a información generada.

- LIBRERÍA PROTOCOLO

Librería en C que contendrá las funciones que implementen el protocolo de comunicación a diseñar, permitiendo el envío de comandos y la recepción de información a cualquier aplicación.

- LIBRERÍA COMUNICACIÓN

Librería en C que contendrá las funciones de bajo nivel que permitan la transmisión y recepción de datos a través del Transmisor/Receptor radio conectado al PC a través del Puerto Paralelo.

- CODIFICADOR MC145026

Chip que codifica los datos a transmitir junto con la dirección, del receptor, generando una salida serie. Tanto el dato a transmitir es de 4 bits, mientras que la dirección destino es de 5 bits, con lo que el codificador permite hasta 32 direcciones destino.

- TRANSMISOR DATOS RF

Transmite en la frecuencia 433,92 MHz la salida serie del codificador utilizando una modulación AM On/Off.

- RECEPTOR DATOS RF

Ajustado a la misma frecuencia que el transmisor, recibe la señal transmitida por éste, la demodula y se la pasa al decodificador.

- DECODIFICADOR MC145027

Chip que de decodifica los datos procedentes del receptor de RF y codificados por el MC14526, colocando el dato de 4 bits recibido a su salida, cuando la dirección destino del mensaje coincida con la que se coloca a su entrada de dirección.

- LOGICA DE INTERFAZ Y CONTROL

Lógica que permitirá realizar la entrada y salida de los datos recibidos (4 bits) en recepción y de los datos a enviar (4 bits) junto con la dirección destino en transmisión, a través del PUERTO D del Microbot. Así como permitirá realizar el control del modulo Transmisor/Receptor también a través de dicho puerto.

- PROGRAMA EN EL MICROBOT

Programa hecho en ensamblador que se ejecutará en el Microbot (por el 68HC11) y que deberá ser capaz de interpretar y procesar los comandos e información procedentes del PC y de enviarle a éste los datos solicitados utilizando el protocolo de comunicación. Esta aplicación será almacenada en la memoria EEPROM del 68HC11.

2.8 Interacción entre bloques

Al realizar una acción sobre el panel de control que implique una comunicación, el programa en ejecución llamará a alguna función de la librería PROTOLOCO, esta función a su vez utilizará funciones de la librería COMUNICACIÓN para transmitir o recibir datos a través del transmisor/receptor conectado al PC por el Puerto Paralelo.

En caso de una transmisión del PC, al codificador llegará a través del Puerto Paralelo un paquete formado por un campo *dirección* y un campo *dato*, el cual codificará y pasará al transmisor para que lo envié. Este dato llegará entonces al receptor del robot, que se lo pasará al decodificador, que de reconocer que es el destinatario de la transmisión activará una señal para que el microcontrolador lea el dato recibido a través del PUERTO D.

Cuando el Microbot quiera enviar un dato, se lo pasará al sistema junto con la dirección del destinatario, también a través del PUERTO D, generándose una secuencia similar a la anteriormente descrita.

En caso de producirse la recepción de un dato, ambas plataformas (PC y Microbot) lo leerán y lo interpretarán según la situación en la que se encuentre la comunicación en ese momento, respondiendo con una transmisión si así se requiere. Si el PC ha solicitado alguna información al robot (acerca de sus sensores, estado, etc) cuando la haya recibido correctamente, la mostrará por pantalla.

CAPÍTULO 4. DISEÑO DEL SISTEMA

1 DISEÑO HARDWARE

1.1 Introducción

Al final del capítulo anterior se llegó a unos bloques funcionales perfectamente definidos en cuanto a entradas, salidas y funcionalidad. A continuación se describe, para la parte *Hardware* del diseño, cómo se ha realizado la implementación de cada uno de éstos bloques, presentándose a continuación la conjunción de todos ellos para formar el sistema completo.

1.2 Interfaz con el PC: El PUERTO PARALELO (IEE1284)

El puerto paralelo del PC se compone de 4 líneas de control, 5 líneas de estado y 8 líneas de datos. Se presenta normalmente en los PCs y compatibles en forma de conector DB25 hembra. La utilización normal de éste es como puerto *Centronics* para impresora, donde se hace uso de las 8 líneas de datos unidireccionales y de un número de señales de control.

Los nuevos puertos paralelos están estandarizados bajo la norma IEEE 1284 aparecida en 1994. Este estándar define 5 modos de operación, los cuales se detallan a continuación.

1. Compatibility Mode.
2. Nibble Mode.
3. Byte Mode.
4. EPP Mode *(Enhanced Parallel Port).*
5. ECP Mode *(Extended Capabilities Port).*

Los modos Compatibility, Nibble y Byte usan el mismo hardware presente en los puertos paralelos originales, por lo que son completamente compatibles con estos (definidos por el estándar SPP - Standard Parallel Port). Los modos EPP y ECP requieren sin embargo de hardware adicional para realizar el control de las transferencias (‘handshake’), con lo que consiguen mayores velocidades de transmisión que los modos anteriores.

El modo Compatibility o modo “Centronics”, como es conocido normalmente, solo puede enviar datos en una dirección a una velocidad de hasta 150 Kbytes/s. Para poder leer datos ha de utilizarse el modo Nibble, o el modo Byte. El modo Nibble permite la entrada de hasta 5 bits, utilizando los pines correspondientes al registro de estado. El modo Byte utiliza bidireccionalidad en las líneas de datos (8 bits) para realizar la lectura de un Byte (no es posible en todos los PCs).

El puerto paralelo 1 (LPT1) tiene normalmente asociada la dirección base 378h en la memoria de E/S del PC, mientras que el puerto 2 (LPT2) tiene asociada normalmente la 278h (la ‘h’ indica

hexadecimal). Sin embargo esto puede no ser siempre así, cambiando estas direcciones de unas maquinas a otras. Para conocer las direcciones base de los Puertos Paralelos instalados en cada maquina se puede mirar en una tabla de 'lookup' que proporciona la BIOS. En la figura 31 se muestra donde se encuentra almacenada para cada Puerto Paralelo su dirección base.

Start Address	Function
0000:0408	LPT1's Base Address
0000:040A	LPT2's Base Address
0000:040C	LPT3's Base Address
0000:040E	LPT4's Base Address (Note 1)

Fig. 31: Posiciones de memoria donde encontrar las direcciones de los PP.

A continuación se muestran las señales correspondientes a cada pin del Puerto Paralelo en los conectores DB25 y Centronics.

Pin No (D-Type 25)	Pin No (Centronics)	SPP Signal	Direction In/out	Register	Hardware Inverted
1	1	nStrobe	In/Out	Control	Yes
2	2	Data 0	Out	Data	
3	3	Data 1	Out	Data	
4	4	Data 2	Out	Data	
5	5	Data 3	Out	Data	
6	6	Data 4	Out	Data	
7	7	Data 5	Out	Data	
8	8	Data 6	Out	Data	
9	9	Data 7	Out	Data	
10	10	nAck	In	Status	
11	11	Busy	In	Status	Yes
12	12	Paper-Out PaperEnd	In	Status	
13	13	Select	In	Status	
14	14	nAuto-Linefeed	In/Out	Control	Yes
15	32	nError / nFault	In	Status	
16	31	nInitialize	In/Out	Control	
17	36	nSelect-Printer nSelect-In	In/Out	Control	Yes
18 - 25	19-30	Ground	Gnd		

Fig. 32: Señales del Puerto Paralelo

La tabla anterior utiliza la 'n' delante del nombre de la señal para indicar que esta es activa a nivel bajo (ej: nError). Las señales marcadas como 'Hardware inverted' están invertidas internamente por el hardware del puerto (por ejemplo la señal Busy perteneciente al registro STATUS).

Los niveles de salida de los puertos paralelos son normalmente niveles TTL. Asimismo, en teoría las líneas de datos tienen salida en colector abierto, lo que no es siempre cierto puesto que pueden tener resistencias internas de pull-up, o ser salidas normales (Totem-pole). Por esta razón además de para

adaptar los niveles a los exigidos por el Chip codificador MC145026 que requiere niveles de entrada HCMOS, ha sido necesario en el sistema diseñado colocar a las salidas resistencias de *pull-up*. Estas resistencias tienen un valor de 2,2 KΩ, valor calculado para que garanticen los niveles de tensión requeridos a las entradas del codificador (HCMOS). Se ha elegido este valor después de calcular los valores máximo y mínimo posibles, los cuales se desprenden de las siguientes expresiones.

$$R_{p\max} = \frac{Vcc_{\min} - V_{OH\min}}{I_{OH} + I_{IH}} = \frac{4{,}5 - 3{,}9}{2 \cdot 110 \cdot 10^{-6}} = 2727\Omega$$

$$R_{p\min} = \frac{Vcc_{\max} - V_{OL\max}}{I_{OL\max} - I_{IL}} = \frac{5.5 - 0{,}5}{8 \cdot 10^{-3} - 0{,}1 \cdot 10^{-6}} = 588\Omega$$

Como se puede observar en las expresiones anteriores, se permitirá para el sistema una tensión de alimentación de entre 4,5 y 5,5 voltios (tanto en la tarjeta conectada al Microbot como en la conectada al PC).

En las siguientes figuras se muestra la función de cada uno de los bits de los registros de DATOS, CONTROL y ESTADO del puerto paralelo.

Offset	Name	Read/Write	Bit No.	Properties
Base + 0	Data Port	Write (Note-1)	Bit 7	Data 7 (Pin 9)
			Bit 6	Data 6 (Pin 8)
			Bit 5	Data 5 (Pin 7)
			Bit 4	Data 4 (Pin 6)
			Bit 3	Data 3 (Pin 5)
			Bit 2	Data 2 (Pin 4)
			Bit 1	Data 1 (Pin 3)
			Bit 0	Data 0 (Pin 2)

Fig. 33: Puerto de DATOS

Base + 1	Status Port	Read Only	Bit 7	Busy
			Bit 6	Ack
			Bit 5	Paper Out
			Bit 4	Select In
			Bit 3	Error
			Bit 2	IRQ (Not)
			Bit 1	Reserved
			Bit 0	Reserved

Fig. 34: Puerto de ESTADO

Base + 2	Control Port	Read/Write	Bit 7	Unused
			Bit 6	Unused
			Bit 5	Enable bi-directional Port
			Bit 4	Enable IRQ Via Ack Line
			Bit 3	Select Printer
			Bit 2	Initialize Printer (Reset)
			Bit 1	Auto Linefeed
			Bit 0	Strobe

Fig. 35: Puerto de CONTROL

En el presente proyecto se ha utilizado el llamado modo Nibble, donde los 8 bits del registro de *datos* se utilizan para la salida de las direcciones (4 bits) y de los datos a transmitir (4 bits), mientras que se utilizan 4 bits del registro de *estado* para la entrada de los datos recibidos (bits 3,4,5 y 7), empleándose un quinto bit de este registro como entrada de interrupción (bit 6). Del registro de *control* se utilizarán dos bits para las señales de control /TE y nR/W (bits 0 y 1 respectivamente). Para que el puerto paralelo genere una interrupción por flanco de subida en la señal Ack (bit 6 del registro de estado) el bit 4 del registro de control debe estar a uno (interrupción habilitada).

1.3 Bloque CODIFICADOR - DECODIFICADOR

Para la codificación y decodificación de los datos se han utilizado los circuitos integrados MC145026 y MC145027 de Motorola, especializados en aplicaciones de control remoto. En la figura 36 se puede observar el *pin out* de estos Chips.

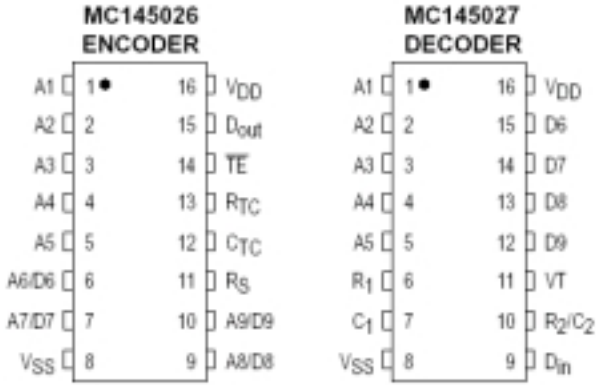


Fig. 36: Chips MC145026 y MC145027

El MC145026 codifica los datos definidos por el estado de las 5 lineas trinarias de dirección (A1-A5) y las 4 líneas binarias de datos (D6-D9), y los transmite de forma serie a través del pin Dout cuando se activa la señal de transmisión (/TE). Mientras que el MC145027 decodifica la señal que le llega por el pin Din, y si detecta que la dirección del dato transmitido coincide con la que tiene en sus 5 pines de dirección (A1-A5), activa la señal VT y coloca el dato recibido en sus salidas de dato (D6-D9).

Para las tensiones de alimentación que se van a utilizar en el sistema diseñado (5 voltios), los niveles de tensión a la entrada y a la salida de estos circuitos integrados son similares a los de la familia HCMOS. Para más información acerca de las características eléctricas de estos Chips se remite al lector a su libro de características *[Moto1]*.

1.3.1 Codificador MC145026

El C.I. Motorola MC145026 en cada secuencia de transmisión envía la codificación de un paquete formado por un campo *dirección* (A1-A5) y un campo *dato* (D6-D9), con redundancia doble (dos veces) para incrementar la seguridad.

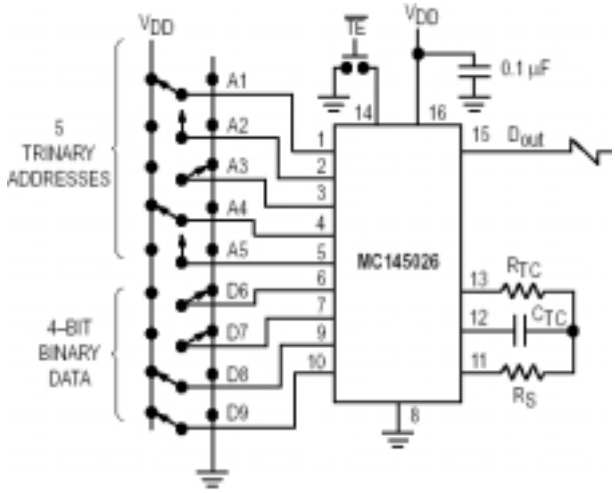


Fig. 37: Diagrama eléctrico del Codificador

Los *pines* de direcciones A1-A5 son trinarios porque pueden estar en uno de los tres estados (bajo, alto, o alta impedancia), permitiendo la existencia de 243 direcciones destino distintas.

Cada dígito transmitido es codificado mediante dos pulsos (ver Figura 38). Un '0' lógico (bajo) esta codificado como dos pulsos cortos consecutivos, un '1' lógico (alto) como dos pulsos largos consecutivos, alta impedancia se codifica con un pulso largo seguido de uno corto.

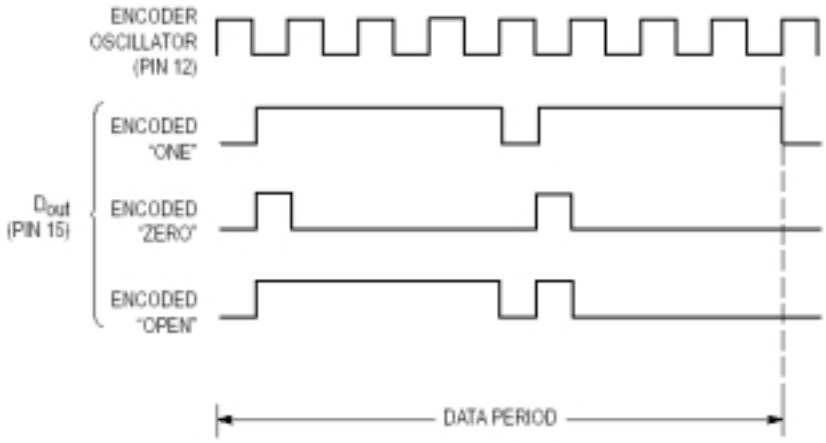


Fig. 38: Codificación de cada estado.

La secuencia de transmisión de un dato se inicia cuando se pone a cero la señal de entrada /TE (activa a nivel bajo), además si para cuando se termina de transmitir un dato la señal /TE sigue activa, el dispositivo vuelve a realizar la transmisión del dato que haya a su entrada. Así el MC145026 puede estar transmitiendo continuamente tanto tiempo como la señal /TE permanezca a nivel bajo, dejando de transmitir datos cuando al final de la transmisión de uno de estos la señal /TE está a uno. Entre el final de la transmisión de un dato y el comienzo de la del siguiente no se envía señal en tres períodos del reloj de transmisión (ver Figura 39).

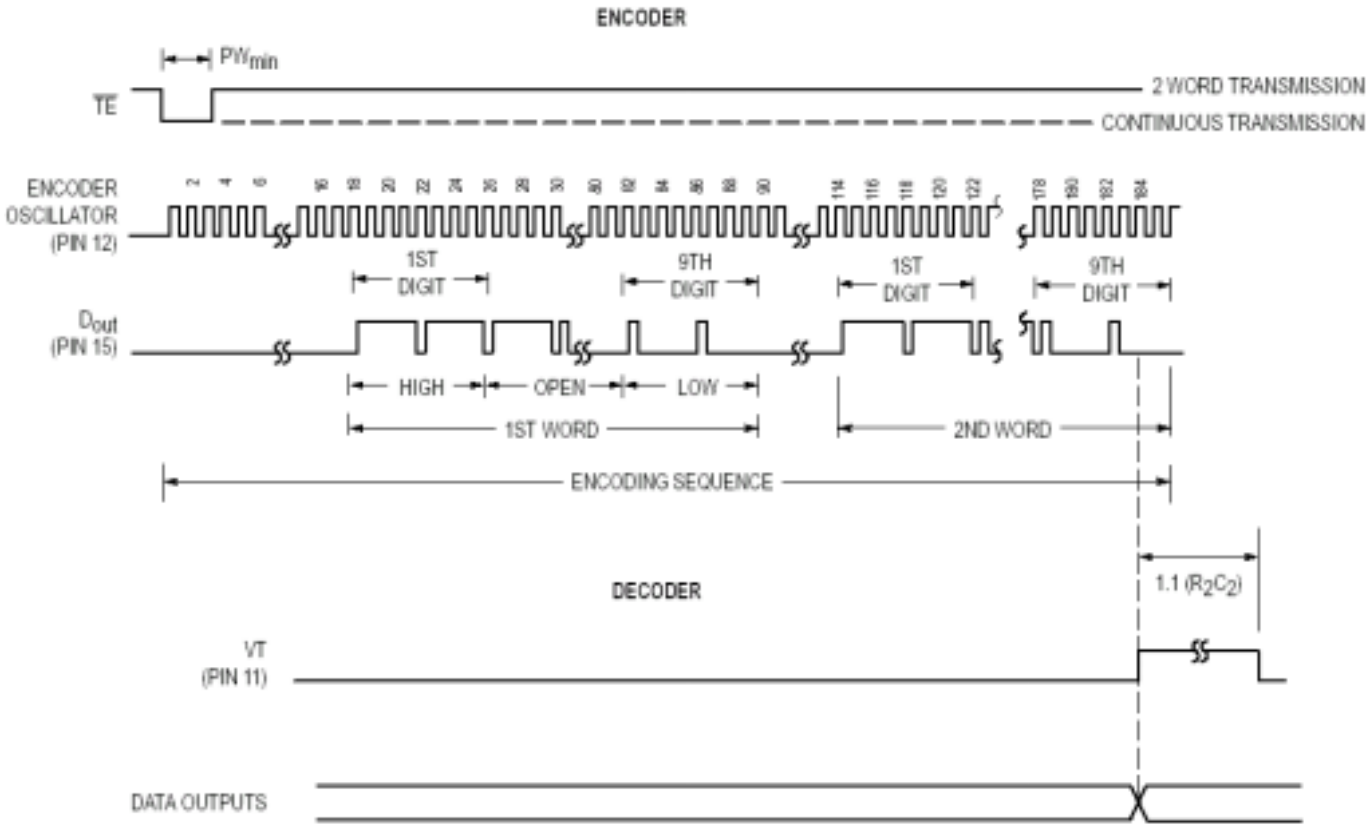


Fig. 39: Diagrama de tiempos.

Mientras /TE esté a nivel alto y el dato ya haya sido transmitido el codificador estará completamente deshabilitado, el oscilador inhibido y el consumo de corriente será mínimo. Cuando se lleva /TE a nivel bajo el oscilador arranca y la secuencia de transmisión comienza. Las entradas son secuencialmente seleccionadas, y las acciones a llevar a cabo son determinadas por los estados lógicos de éstas. Esta información se transmite en serie vía el *pin Dout.* Hay que destacar que la entrada /TE tiene internamente un dispositivo de *pull-up*, por lo que no será necesario colocar en este pin ninguna resistencia de pull-up externa.

A continuación se detalla la función de cada uno de los pines del MC145026:

A1-A5 → Entradas de Direcciones (*Pines* 1-5)

Estos pines son entradas trinarias de direcciones, que son codificadas junto con los datos y enviados vía serie por el pin *Dout.*

D6-D9 → Entradas de Datos (*Pines* 6,7, 9 y 10)

Estos pines son entradas binarias de datos, que son codificados junto con las direcciones y enviados vía serie por el pin *Dout.*

Rs, Ctc, Rtc (*Pines* 11,12, y 13)

Estos *pins* son parte de la sección del oscilador.

Si una señal externa se usa como fuente osciladora, se debe conectar a la entrada Rs y la Rtc y la Ctc deben estar abiertas.

/TE → Transmit Enable (*Pin* 14)

Esta entrada de habilitación de transmisión inicia la transmisión cuando se fuerza a nivel bajo. Un dispositivo interno de *pull-up* mantiene esta entrada normalmente a 1 lógico.

Dout → Data Out (*Pin* 15)

Esta es la salida del codificador que de forma serie presenta la palabra de datos.

Vss → Negative Power Supply (*Pin 8)*

El potencial de alimentación más negativo. Este *pin* está normalmente a masa.

Vdd → Positive Power Supply (*Pin 16)*

El *pin* más positivo de alimentación.

Los valores de Rtc, Ctc, y Rs determinan la frecuencia del oscilador del codificador. Para poder calcular el valor de estos componentes según la frecuencia de oscilación deseada, el fabricante proporciona las formulas o expresiones que se indican a continuación (validas para 1 KHz ≤ f_{osc} ≤ 400 KHz).

$$f_{osc} = \frac{1}{2.3\,R_{TC}C_{TC'}} \qquad C_{TC'} = C_{TC} + C_{layout} + 12\ pF$$

$$100\ pF \le C_{TC} \le 15\ \mu F$$

$$R_{TC} \ge 10\ k\Omega;\ R_S \approx 2\,R_{TC}$$

Puesto que la máxima frecuencia que soporta el receptor de radiofrecuencia es de 2 KHz (aunque experimentalmente se ha comprobado como está es sensiblemente menor) para el presente proyecto se ha utilizado la frecuencia de oscilación de 1,65 KHz, con lo que se obtienen los siguientes valores para Rtc, Ctc y Rs.

Rtc = 47 KΩ Ctc = 5,6 nF Rs = 100 KΩ

La transmisión completa de un dato realizada individualmente requiere de 217 ciclos de reloj, 185 ciclos para la transmisión y 32 ciclos de espera para que se desactive la señal que indica '*transmisión valida*' en el decodificador (VT). Como la frecuencia del reloj es de 1650 ciclos/segundo se obtiene una velocidad de transmisión aproximada de 7,6 Nibbles por segundo (en cada transmisión se transmite un dato de 4 bits ≡ 1 Nibble).

1650/217 ≈ 7,6 Nibbles por segundo.

Lo cual es una velocidad suficiente para demostrar la funcionalidad del sistema.

1.3.2 Decodificador MC145027

Los datos que van saliendo en serie del receptor necesitan la respectiva decodificación para obtener el dato transmitido. Para ello se ha utilizado el módulo dual al MC145026, esto es, el MC145027 que es un decodificador.

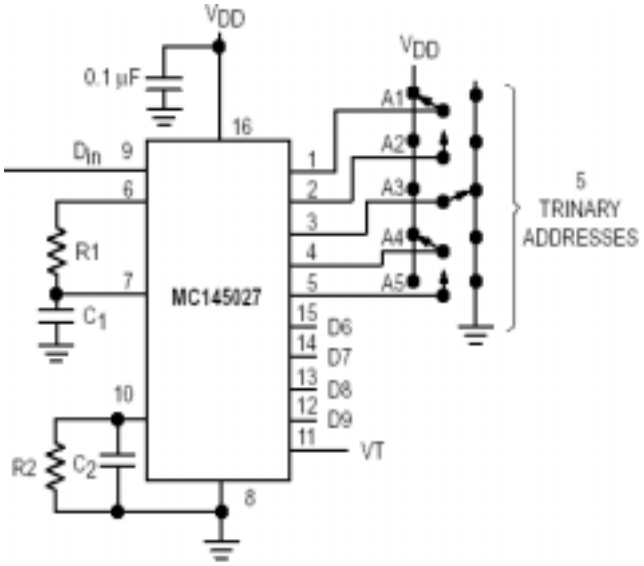


Fig. 40: Diagrama eléctrico del Decodificador

El MC145027 recibe la trama serie e interpreta cinco de los dígitos trinarios como un código de dirección. Entonces 243 direcciones son posibles. Si se usan datos binarios en el codificador son posibles 32 direcciones. La información serie restante se interpreta como cuatro bits de datos binarios. La salida VT (transmisión válida) se activa poniéndose a nivel alto cuando se cumplen dos condiciones. La primera, dos direcciones deben ser recibidas consecutivamente (en una secuencia de codificación), las cuales deben ser iguales a la dirección local seleccionada en el decodificador. La segunda, los 4 bits de datos (al igual que las direcciones) deben ser iguales a los últimos datos válidos recibidos. La activación de "VT" indica que la información en la salida de datos ha sido actualizada.

Los datos transmitidos consisten de dos secuencias idénticas, examinadas bit a bit durante la recepción. Los primeros cinco dígitos trinarios son asumidos como dirección. Si la dirección recibida coincide con la dirección local los siguientes cuatros bits (datos) son almacenados internamente, pero no son transferidos al *latch* de datos de salida. Cuando la segunda palabra codificada se recibe la dirección debe ser otra vez igual a la local. Si esto ocurre, los nuevos bits de datos son chequeados con los previos almacenados. Si los dos *Nibbles* de datos (4 bits cada uno) son iguales los datos son transferidos al *latch* de datos de salida y se mantienen en éste hasta que unos nuevos datos lo reemplacen. A su vez la salida VT se lleva a nivel alto y permanece así hasta que se recibe un error o hasta que no se recibe señal de entrada en cuatro períodos de datos o lo que es lo mismo 32 ciclos del relog del codificador (8 ciclos por cada dato).

A continuación se detalla la función de cada uno de los pines del MC145027:

A1-A5 → Entradas de dirección (*Pins* 1-5)

Estas son las entradas de dirección local. Los estados de estos *pins* deben ser iguales a la entrada del codificador apropiado para que el *pin* VT suba a nivel alto. La dirección local puede ser codificada con datos trinarios o binarios.

D6-D9 → Salida de Datos(*Pins* 15,14,13,12)

Estas salidas presentan la información binaria que se codificó en las entradas D6 a D9 del MC145026. Sólo son reconocidos en estas salidas datos binarios, una línea en alta impedancia en el codificador MC145026 se decodifica como un nivel alto (1 lógico).

Din → Data In (*Pin 9)*

Este *pin* es la entrada serie al decodificador. El voltaje de entrada debe ser con niveles HCMOS.

R1, C1 → Resistor 1, Capacitor 1 (*Pins* 6,7)

Ver Figura 40.

R2/C2 → Resistor 2/Capacitor 2 (*Pin* 10)

Ver Figura 40.

VT → Salida Transmisión Válida (*Pin* 11)

Esta salida de transmisión válida se pone en alto después de la segunda palabra de una secuencia de decodificación cuando se satisfacen las siguientes condiciones:

1. La dirección recibida de ambas palabras iguala a la dirección local del decodificador.
2. Los bits de datos de ambas palabras se igualan.

La señal VT permanece en alto hasta que no se produce igualdad entre dos palabras consecutivas o no se recibe señal en cuatro períodos de datos.

Vss → Negative Power Supply (*Pin 8)*

El potencial más negativo. Este *pin* esta normalmente a masa.

Vdd → Positive Power Supply (*Pin* 16)

El potencial más positivo.

Los valores de resistencias R1, R2 y los condensadores C1 y C2 determinan la frecuencia a la que se espera recibir los datos. Así, para que el decodificador esté en perfecta sincronía con el codificador, el valor de estos componentes hay que calcularlo utilizando las siguientes formulas, donde es necesario conocer el valor de " Rtc x Ctc ".

$$R_1C_1 = 3.95\,R_{TC}C_{TC} \qquad R_1 \geq 10\ k\Omega,\ C_1 \geq 400\ pF$$

$$R_2C_2 = 77\,R_{TC}C_{TC} \qquad R_2 \geq 100\ k\Omega,\ C_2 \geq 700\ pF$$

Con lo que para un valor de Rtc de 47 KΩ y de Ctc 5,6 nF, calculados para una frecuencia de 1,65 KHz (ver codificador), se obtienen unos valores para R1, R2, C1 y C2 de:

R1 = 47 KΩ C1 = 22 nF R2 = 200 KΩ C2 = 100 nF

Al final de una recepción en la salida de datos se tiene el dato original (enviado por el codificador), validado por la señal VT.

1.4 Bloque TRANSMISOR – RECEPTOR radio

La trama serie que llega desde el codificador necesita ser enviada por radio frecuencia, para esto se ha utilizado un módulo transmisor de corto alcance (SRD) fabricado por AUREL y exento de licencia al transmitir en la banda de 433MHz (banda ICM). Este transmisor es especifico para datos, utilizando una Modulación AM On/Off. Asimismo, para recibir la señal se ha utilizado un receptor del mismo fabricante y especialmente diseñado para demodular la señal emitida por el transmisor. En la figura 41 se puede observar el aspecto exterior de ambos módulos.

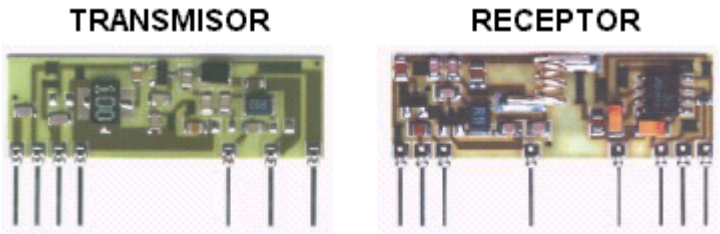


Fig. 41: Módulos transmisor y receptor.

La antena recomendado para utilizar con estos módulos es unipolar de ¼ de onda (17 cm) y de este tipo son las utilizadas en la realización de este proyecto, las cuales se han implementado utilizando hilos de cobre de 1 mm de diámetro cortados a la longitud indicada.

1.4.1 Módulo transmisor de RF

Se trata de un transmisor con oscilador SAW y con antena unipolar exterior, ideal para aplicaciones donde se quiera modular On-Off una portadora RF con datos digitales. El aspecto exterior es el de un módulo híbrido con encapsulado SIL (*Single In Line*), cuyo diagrama de bloques se muestra en la figura 42.

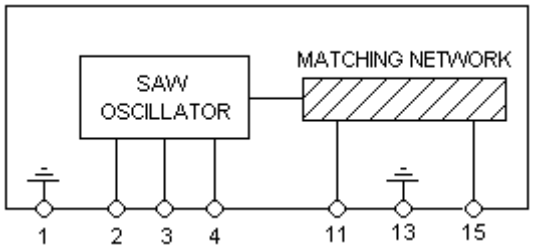


Fig.42: Diagrama de bloques del transmisor RF

Las características de este transmisor RF son las siguientes:

- Circuito híbrido de elevada miniaturización tipo SIL (*Single In Line*).
- Frecuencia de trabajo: 433,92 MHz, obtenida mediante un resonador SAW.
- Potencia de salida RF sobre una carga de 50 Ω, *pin* 11 (ver figura 14).
- Espúreos: -50 dB respecto a la fundamental.
- Frecuencia de modulación: 4 kHz max. (ver figura 14).
- Antena externa unipolar de ¼ de onda (17 cm).
- Formato: in *line*, paso 2,54 mm.
- Dimensiones : 38,1 x 13,2 x 3 mm.
- Conforme con la normativa europea ETS 300220 y ETS300683 (Compatibilidad Electromagnética). Los límites de estabilidad de la Normativa ETS 300220 son respetados sólo para Vcc ≤ 5V (ver figura 14).

El conexionado de los *pines* del módulo transmisor se corresponde con el indicado en la figura 43.

Conexionado de los pines:	
1	Masa
2	Entrada Modulación (ver figura 14)
3	Entrada Modulación (ver figura 14)
4	Masa
11	Antena
13	Masa
15	Positivo (ver figura 14)

Fig. 33: Conexionado de los pines del transmisor RF.

La conexión de los *pines*, como se puede apreciar, es sencilla salvo para la "*ENTRADA DE MODULACIÓN*", para la que habrá que seguir las indicaciones de la tabla que se muestra a continuación (figura 44). En dicha tabla se observa como para cada situación cambian las prestaciones del sistema en cuanto a ancho de banda, potencia emitida y consumo.

Alimentación Vcc	**Pin2** V	**Pin3** V	**Frecuencia modulación** KHz	**Potencia emitida** Dbm	**Consumo** MA
3 – 5	O.C. (0 a Vcc)	N.C.	3	3,5 – 8	3,5 – 7,5
5 – 8	N.C.	O.C. (0 a 5)	4	7,5 – 10,5	3,5 – 4
8 - 12	O.C. (0 a 5)	N.C.	4	12 - 15	7,5 – 9,5

Fig. 44: Modos de operación del transmisor.

En el sistema diseñado se ha utilizado un *jumper* para que el usuario pueda seleccionar entre el pin 2 y el pin 3 para introducir al transmisor la señal a modular, con objeto de hacer al sistema lo más versátil posible y conseguir que éste se adapte a distintas necesidades y requerimientos en cuanto a ancho de banda, consumo y potencia emitida. Aunque se recomienda, sobre todo en el Microbot, utilizar la entrada de modulación correspondiente al pin 3, por oscilar la tensión de alimentación en éste entre 5 y 6 voltios normalmente y ser además ésta entrada la que implica un menor consumo, aspecto éste muy importante en el Microbot al estar éste normalmente alimentado por baterías.

1.4.2 Módulo receptor de RF

Los datos transmitidos en la banda de 433 MHz son necesarios trasladarlos a banda base para su recepción. Para ello se necesita un receptor adecuado que trabaje en la misma frecuencia (433,92 MHz). Se ha utilizado para esto, un receptor del mismo fabricante (AUREL) y especialmente diseñado para demodular la señal emitida por el transmisor utilizado. Este módulo es un receptor de baja absorción, alta inmunidad al ruido de alimentación y radiación en antena.

Al igual que el módulo emisor, éste se presenta como un módulo híbrido con encapsulado SIL (*Single In Line*), mostrándose a continuación su diagrama de bloques (figura 45).

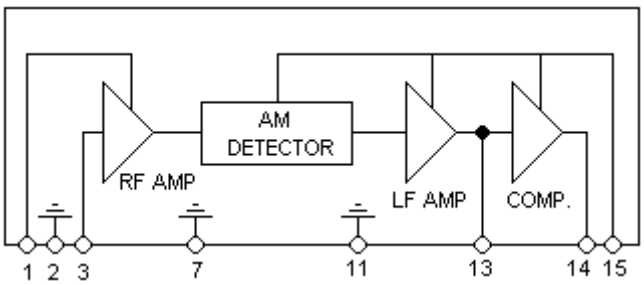


Fig. 45: Diagrama de bloques del receptor RF.

Las características de este receptor RF son las siguientes:

- Circuito híbrido de elevada miniaturización tipo SIL (*Single In Line*).
- Frecuencia de recepción: 433,92 MHz.
- Recepción de señal con modulación OOK (*On-Off Keying*).
- Sensibilidad RF, medida con señal On-Off en la entrada: mejor de 3μV (-97 dBm) a centro de banda.
- Banda pasante RF a –3 dB: típica 1,2 MHz.
- Alimentación con filtro RC para eliminar parásitos debidos a circuitos tipo sirena de alarma.
- Antena externa unipolar de ¼ de onda (17 cm).
- Salida con onda cuadrada, frecuencia máxima de 2 kHz.
- Nivel lógico de salida: bajo en ausencia de señal RF.
- Alimentación: 5 V, consumo máximo 3mA (típico 2,7 mA).
- Radiación en antena <-60 dBm (analizador 50 Ω, filtro FI 100 kHz).
- Tiempo de subida < 2 s
- Formato: *in line,* paso 2,54 mm
- Dimensiones: 38,1 x 13,7 x 5,5 mm.
- Conforme con la normativa europea ETS 300220

El conexionado de los *pines* del módulo receptor se corresponde con el indicado en la figura 46.

Conexionado de los pines:	
1	Positivo 5V
2	Masa
3	Antena
7	Masa
11	Masa
13	Punto de prueba
14	Salida
15	Positivo 5V

Fig. 46: Conexionado de los pines del receptor RF

1.5 Interfaz con el MICROBOT

1.5.1 El PUERTO D del 68HC11

El puerto que se va a utilizar para realizar el control, la entrada, y la salida de datos a la tarjeta conectada al Microbot es el PUERTO D del microcontrolador Motorola 68HC11. Se trata de un puerto bidireccional, en el que se pueden configurar sus bits independientemente como entradas o como salidas.

El registro de configuración **DDRD** se encuentra en la dirección $1009. Los 'unos' en los bits de este registro se corresponden con salidas para los bits del Puerto D y los 'ceros' con entradas. El Puerto D está mapeado en la dirección $1008 de la memoria del 68HC11. Este puerto es de sólo 6 bits, estando algunos de sus bits compartidos con el SPI y el SCI (unidades de comunicación sincronía y asíncrona).

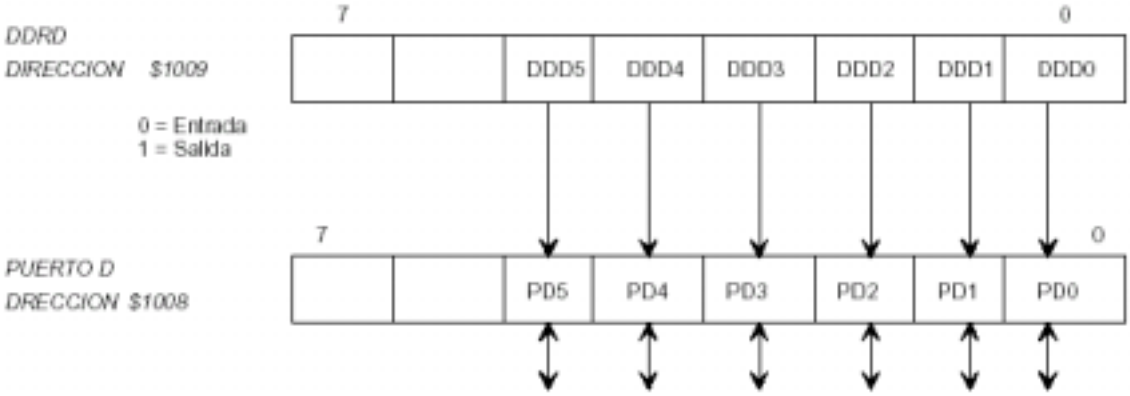


Fig. 47: Puerto D y su configuración

Los bits 0 y 1 están compartidos con el puerto serie. Por ello, para poder ser utilizados es preciso que el SCI esté desactivado. Los bits 2,3,4 y 5 están compartidos por el SPI (ver figura 48), con lo que para que funcionen es necesario asegurarse que el SPI está desactivado.

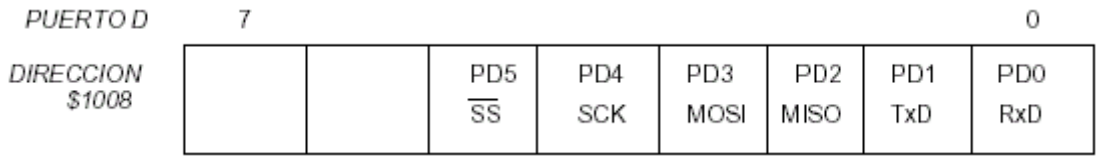


Fig. 48: Otros usos del Puerto D

Una de las características del puerto D es que sus salidas pueden ser normales o en colector abierto. Esto se configura con el bit 5 del registro SPCR ($1028). Si este bit está a '0' las salidas serán normales. Si está a '1' las salidas estarán en colector abierto. Por defecto están en colector abierto. Para la realización de este proyecto se ha debido seleccionar el modo con salidas en colector abierto, puesto que en la tarjeta CT6811 existen resistencias de pull-up colocadas en algunas salidas (bit PD0). No debiéndose configurar como salidas normales. En la tarjeta que se conectará al Microbot se han colocado resistencias de *pull-up* en los pines del puerto donde no están colocadas éstas internamente en la CT6811

(bits PD1 a PD5). El valor de estas resistencias de *pull-up* es de 4,7 KΩ, garantizando éstas los niveles de tensión necesarios para la tecnología de los chips utilizados (HCMOS).

Como se ha dicho, el PUERTO D tiene solo seis bits, de éstos, dos se utilizaran para el control de la tarjeta transmisora/receptora, mientras que los otros 4 bits se utilizarán para realizar la entrada y salida de datos a la tarjeta. Como para realizar una transmisión va a ser necesario pasarle al bloque codificador una dirección destino (4 bits) y una dato a transmitir (4 bits), será necesaria la utilización de una lógica de interfaz que realice una multiplexación en el tiempo de las direcciones y de los datos a través del Puerto D. El diseño de dicha lógica adicional se detalla más adelante en el apartado dedicado al diseño del sistema completo.

1.5.2 El puerto CONTROL de la CT6811

En el llamado puerto CONTROL de la tarjeta CT6811 se han recopilado una serie de señales significativas del 68HC11. En el sistema que se ha diseñado se han utilizado por una parte los pines de alimentación y masa contenidos en este puerto, y por otra la entrada de interrupción externa IRQ. A través de esta ultima señal se puede generar una interrupción enmascarable, bien por un nivel bajo o bien por un flanco de bajada. Como se verá más adelante, en el sistema diseñado se ha configurado el microcontrolador 68HC11 para que dicha interrupción se produzca por un flanco de bajada en la señal IRQ.

1.6 Desarrollo del sistema completo

En los apartados anteriores se han descrito los bloques que van a formar el sistema. Una vez en este punto y con capacidad de comprender el funcionamiento y estructura de dichos bloques, se va a presentar el diseño del sistema completo donde se podrá observar la interacción entres los distintos bloques que lo constituyen.

La estructura física del sistema se divide en dos partes, por un lado la tarjeta que se conectará al PC a través del Puerto Paralelo, y por otro la tarjeta que se conectará al MICROBOT. Así, la presentación del diseño del sistema completo se hará separando en dos apartados el diseño de cada una de estas dos partes.

La tarjeta conectada al PC esta unida a éste mediante un cable de extensión del Puerto Paralelo con conectores DB25 macho y hembra en sus dos extremos y está alimentada mediante una fuente de alimentación externa de 5 voltios D.C.

La tarjeta conectada la PC se conecta al MICROBOT a través de los puertos D y CONTROL de la CT6811, tomando la alimentación de la propio Microbot, el cual puede ser alimentado a través de una fuente de alimentación externa, pilas o baterías.

1.6.1 Tarjeta conectada al PC

El diagrama eléctrico del sistema conectado al PC a través del Puerto Paralelo se muestra en la siguiente figura:

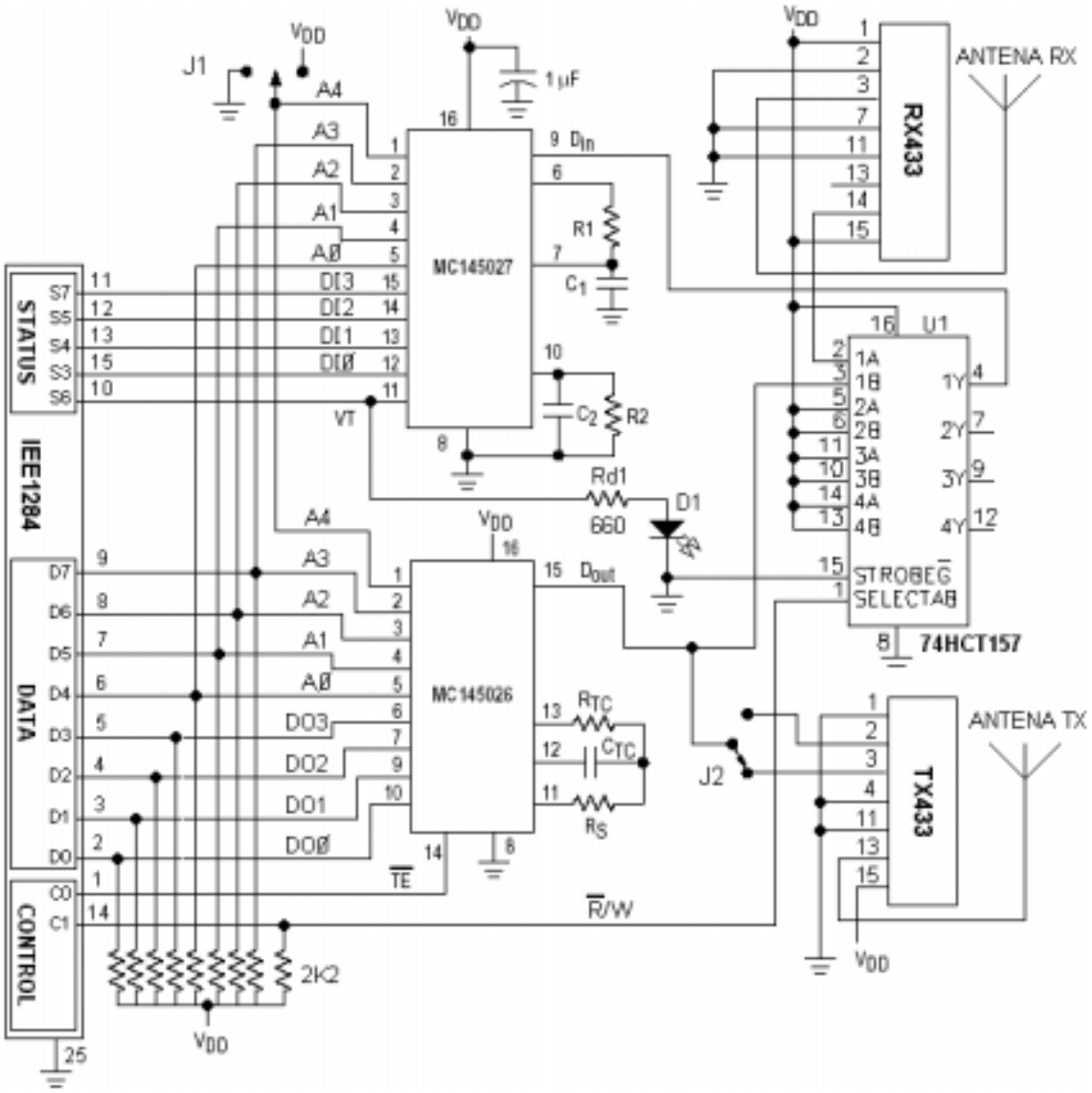


Figura 49: Diagrama eléctrico de la tarjeta conectada al PC.

Se observa en el diagrama eléctrico como la interfaz entre el Puerto Paralelo, el codificador y el decodificador es directa. A continuación se comentan las señales que posibilitan el funcionamiento del sistema:

- La señal **nR\W** (bit 1 del registro de CONTROL del PP) se utiliza para conmutar el estado de la tarjeta entre transmisión y recepción. Un cero lógico en esta señal indica recepción y un uno lógico transmisión. Esta señal sería además la encargada de gobernar el conmutador de antena, en caso de utilizarse éste junto con una misma antena para Transmisión y Recepción.

- La señal **/TE** o *'transmit enable'* (bit 0 del registro de CONTROL del PP) es activa a nivel bajo, y al ponerse a cero hace que comience la secuencia de transmisión de un dato.
- Los pines **DO0-DO3** (bits 0 a 3 del registro de DATOS del PP) constituyen el dato de salida en una transmisión.
- Los pines **A0-A3** (bits 4 a 7 del registro de DATOS del PP) constituyen la dirección que se coloca tanto en el codificador como en el de codificador. Durante una transmisión se deberá poner en estos pines la dirección destino, mientras que cuando el sistema esté en recepción en estos pines deberá colocarse la dirección local del PC (todo mediante SW). Al ser 4 bits están permitidas 16 direcciones posibles.
- Los pines **DI0-DI3** (bits 3,4,5 y 7 del registro de ESTADO del PP) son los pines a través de los cuales el PC lee los datos recibidos.
- El pin **A4** posibilita la existencia de tres grupos distintos de direcciones al poderse fijar su valor mediante el jumper J1 en 1 lógico, 0 lógico, o alta impedancia. Para que una tarjeta pueda recibir datos de otra tarjeta ambas tendrán que tener el mismo valor en la señal A4 (misma situación del jumper J1).
- La señal **VT** (bit 6 del registro de ESTADO del PP) es la que avisa que se ha recibido un dato, generando una interrupción en el PC por flanco de subida.
- El pin **Dout** es a través del cual el codificador pasa la trama serie codificada al emisor de RF para que la transmita. Se ha utilizado el jumper J2 con el objeto de poder seleccionar entre las dos entradas que proporciona el transmisor para la entrada de la señal a modular.
- El pin **Din** es a través del cual el decodificador recibe la señal del receptor de RF.

Durante una transmisión se ha hecho que las direcciones presentes en el codificador y el decodificador coincidan y que la señal recibida por el decodificador sea la transmitida por el codificador (mediante el multiplexor 74HCT157), con lo que al terminar una transmisión la señal VT se activa provocando una interrupción. Así se consigue saber cuando termina una transmisión, aunque esto requiere de un control Software que sepa cuando se estaba transmitiendo y cuando no. Este mecanismo obliga a que no se puedan recibir datos mientras se está transmitiendo, lo que en el sistema en cuestión no limita en absoluto puesto que la comunicación es *simplex* (un solo canal para transmitir y recibir), pero imposibilita el que esta misma disposición hardware se utilice para realizar una comunicación *full duplex* (frecuencias distintas para transmitir y recibir, y capacidad de transmitir y recibir datos simultáneamente).

Añadiendo el multiplexor 74HCT157 se consigue un segundo objetivo, cómo es el garantizar que a la entrada Din del decodificador llegue una señal con los niveles de tensión requeridos por éste. Esto es necesario puesto que el nivel de tensión que proporciona el receptor de RF está muy próximo al mínimo requerido por el decodificador (3,5 voltios), lo que se consigue subsanar intercalando entre ellos un chip con tecnología HCTMOS, que acepta niveles de entrada TTL y proporciona salidas compatibles con la familia HCMOS.

En la figura 50 se muestra el ciclo de una transmisión realizada por el PC, para a continuación explicarlo detenidamente.

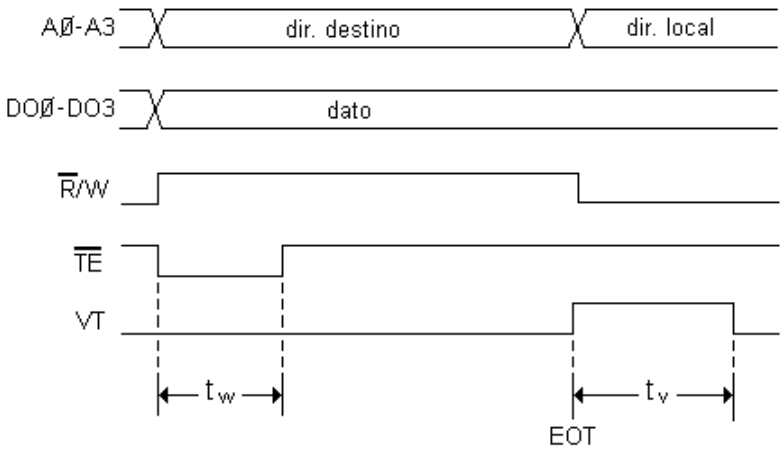


Fig. 50: Ciclo de una transmisión del PC.

Para realizar una transmisión desde el PC primero habrá que poner en los bits 0 a 3 del registro de DATOS el Nibble a enviar y en los bits 4 a 7 del mismo registro la dirección destino. A continuación habrá que conmutar el estado de la tarjeta a transmisión poniendo la señal nR/W a '1' y activar la señal /TE, la cual deberá permanecer a '0' un tiempo mínimo $\mathbf{t_W}$. Una vez se active la señal VT, cuando se produzca el final de la transmisión (EOT- End of Transmision), se generará una interrupción en el PC que pondrá de nuevo la tarjeta en recepción y colocará en los pines de direcciones (A0-A3) su dirección local.

El tiempo $\mathbf{t_W}$ mínimo está determinado por la anchura de pulso mínima requerida por el codificador en el pin /TE, siendo esta de 65 ns (para Vdd = 5 V).

El tiempo $\mathbf{t_V}$ es el que la señal VT permanece activa tras la recepción de un dato. Este tiempo es el correspondiente a 4 periodos de dato, es decir 32 ciclos del oscilador del codificador, lo que para una frecuencia de 1,65 KHz equivale a aprox. 20 ms (este tiempo también puede aproximarse como $1.1 \cdot (R2 \cdot C2)$). Mientras la señal VT esté activa no se debe comenzar ninguna transmisión, puesto que entonces puede que ésta no se desactive (ver sección del descodificador). Así, antes de comenzar una transmisión el software deberá comprobar siempre que la señal VT no está activa.

En la figura 51 se muestra el ciclo de la recepción de un dato por el PC, para a continuación explicarlo detenidamente.

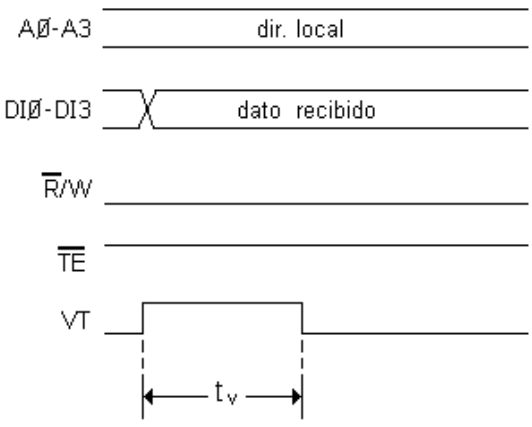


Fig. 51: Ciclo de una recepción del PC.

Para que se produzca la recepción de un dato será necesario que la señal nR/W esté a '0' (tarjeta en recepción), con lo que la entrada de datos del decodificador estará conectada con la salida del receptor de RF y la dirección local del PC presente en los pines de dirección (A0-A3). Por lo demás, cuando se produzca un flanco de subida en la señal VT se generará una interrupción en el PC indicando que se ha recibido un dato, con lo que éste solo tendrá que leerlo a través de la entrada de datos (pines DI0 – DI3).

1.6.2 Tarjeta conectada al MICROBOT

El diagrama eléctrico del sistema conectado al MICROBOT a través del Puerto D y el Puerto CONTROL se muestra en la siguiente figura:

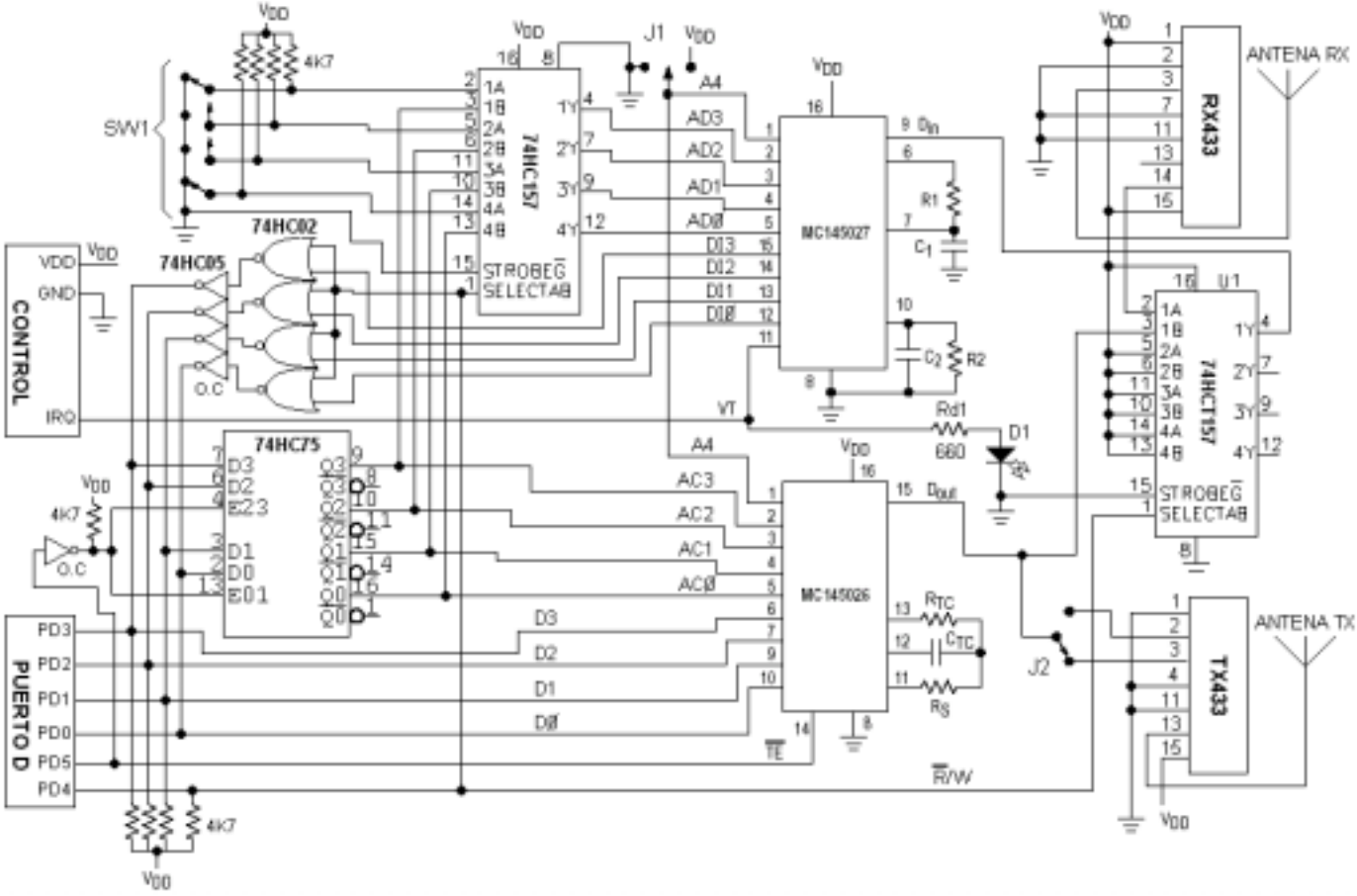


Figura 52: Diagrama eléctrico de la tarjeta conectada al MICROBOT.

En el diagrama eléctrico se puede observar la lógica adicional que ha sido necesario añadir con respecto a la tarjeta conectada al PC.

- Debido a que solo se disponen de 4 bits para realizar la salida de direcciones y datos (4 bits), ha sido necesario añadir un *lacth* (74HC75) para almacenar la dirección destino durante la transmisión. Con lo que, como se verá más adelante, será necesario realizar una multiplexación de las direcciones y los datos que salen a través del PUERTO D. El *enable* del *latch* se consigue invirtiendo la señal /TE.

- Asimismo se han debido añadido unas puertas NOR y unos inversores en colector abierto en las salidas de datos del decodificador, esto se ha hecho con el fin de que estos datos solo lleguen a los pines del PUERTO D cuando la tarjeta esté en recepción (nR/W a 0), en otro caso (nR/W a 1) la salida de los inversores estará en alta impedancia, permitiendo la salida de datos a través de dichos pines.
- También se ha añadido un multiplexor (74HC157), controlado por la señal nR/W, para conseguir que, mientras la tarjeta esté en recepción, en los pines de dirección del decodificador se encuentre la dirección local del Microbot.

A continuación se comentan las señales que posibilitan el funcionamiento del sistema:

- La señal **nR\W** (bit 4 del PUERTO D) se utiliza para conmutar el estado de la tarjeta entre transmisión y recepción. Un cero lógico en esta señal indica recepción y un uno lógico transmisión. Esta señal sería además la encargada de gobernar el conmutador de antena, en caso de utilizarse éste junto con una misma antena para Transmisión y Recepción.
- La señal **/TE** o *'transmit enable'* (bit 5 del PUERTO D) es activa a nivel bajo y al ponerse a cero hace que comience la secuencia de transmisión de un dato.
- Los pines **D0-D3** (bits 0 a 3 del PUERTO D) constituyen las líneas de entrada y salida de datos a la tarjeta. Durante una transmisión en ellos se colocará primero la dirección destino, que será almacenada en el latch (74HC75) y a continuación el dato a transmitir, que habrá de permanecer en ellos hasta que finalice la transmisión. Mientras la tarjeta este en recepción en estos pines estará el ultimo dato recibido.
- Las señales **AC0-AC3** forman la dirección destino de cada transmisión, se almacenan en el latch (74HC75) y se colocan formando parte de las entradas de dirección del codificador.
- Las señales **AD0-AD3** forman parte de las entradas de dirección del decodificador. Durante una transmisión se pondrá en estos pines la dirección destino, mientras que cuando el sistema esté en recepción en estos pines estará la dirección local del Microbot (la cual se fija mediante el switch SW1).
- Los pines **DI0-DI3** son los pines donde el decodificador coloca los datos recibidos.
- El pin **A4** posibilita la existencia de tres grupos distintos de direcciones, al poderse fijar su valor mediante el jumper J1 en 1 lógico, 0 lógico, o alta impedancia. Para que una tarjeta pueda recibir datos de otra tarjeta ambas tendrán que tener el mismo valor en la señal A4 (misma situación del jumper J1).
- La señal **VT** (entrada IRQ del puerto CONTROL) es la que avisa que se ha recibido un dato, generando una interrupción en el 68HC11 por flanco de bajada.
- El pin **Dout** es a través del cual el codificador pasa la trama serie codificada al emisor de RF para que la transmita. Se ha utilizado el jumper J2 con el objeto de poder seleccionar entre las dos entradas que proporciona el transmisor para la entrada de la señal a modular.
- El pin **Din** es a través del cual el decodificador recibe la señal del receptor de RF.
- El switch **SW1** se utiliza para fijar la dirección local del Microbot (16 direcciones posibles).

Como se puede observar, al igual que en la tarjeta conectada al PC, se ha hecho que durante una transmisión el decodificador reciba la señal que emite el codificador y que las direcciones presentes en uno y otro coincidan, para conseguir el mismo objetivo que en aquella.

En la figura 53 se muestra el ciclo de una transmisión realizada por el Microbot, para a continuación explicarlo detenidamente.

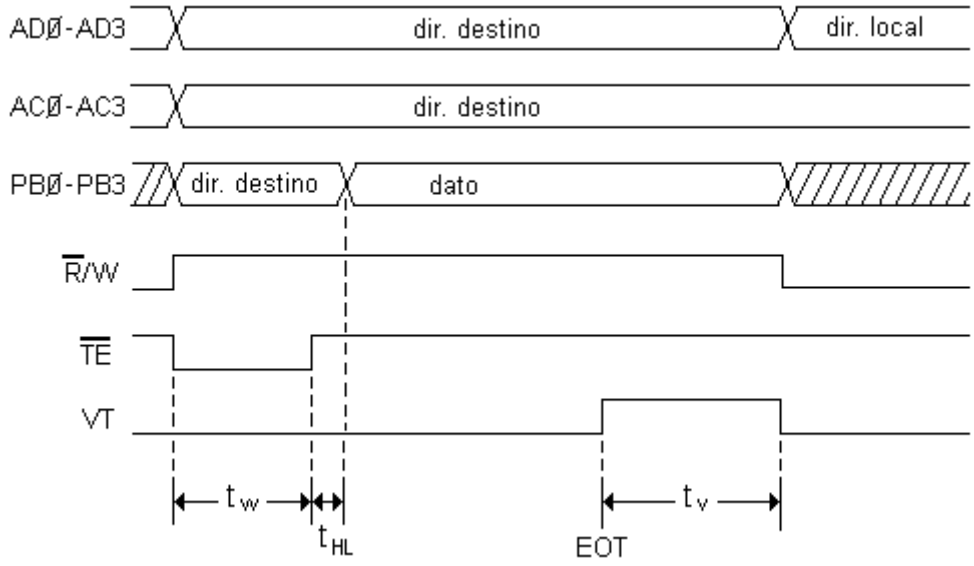


Fig. 53: Ciclo de una transmisión del Microbot.

Para realizar una transmisión desde el Microbot primero habrá que escribir en los bits 0 a 3 del PUERTO D la dirección destino, poner la tarjeta en transmisión (nR/W a 1) y activar la señal /TE. A continuación habrá que desactivar la señal /TE, para después de un tiempo $\mathbf{t_{HL}}$ colocar el dato a transmitir en los bits 0 a 3 del PUERTO D. Una vez finalice la transmisión y se produzca un flanco de bajada en la señal VT se generará una interrupción en el Microbot que pondrá de nuevo la tarjeta en recepción.

El tiempo $\mathbf{t_W}$ mínimo será el mayor de los siguientes tiempos:

- La anchura de pulso mínima requerida por el codificador en el pin /TE (65 ns para Vdd = 5V).
- La anchura de pulso mínima requerida para el *enable* del latch (24 ns para Vdd = 4.5 V).
-El tiempo de *set-up* del latch, (18 ns para Vdd = 4,5 V).

Por lo tanto el tiempo $\mathbf{t_W}$ mínimo será de 65 ns.

La suma de los tiempos $\mathbf{t_W}$ y $\mathbf{t_{HL}}$ quedará limitada como máximo al tiempo determinado por el número de periodos del oscilador necesarios para transmitir los 5 bits de dirección en la primera trama de la codificación, pues para cuando estos transcurran, el dato a transmitir deberá estar estable en los bits PD0-PD3.

$$(17 \text{ periodos} + (5 \text{ bits} \times 8 \text{ periodos/bit})) \times (1/1650 \text{ Hz}) = 34{,}5 \text{ ms}$$

Con lo que:

$$\mathbf{t_W} + \mathbf{t_{HL}} < 34{,}5 \text{ ms}$$

Siguiendo las normas del diseño en el caso peor se tiene que el tiempo $\mathbf{t_{HL}}$ mínimo será el resultado de sumar el retardo máximo del inversor al pasar de '1' a '0' (t_{PHL} = 21 ns) al tiempo de *hold* del lacth (t_H = 3 ns para Vdd = 4,5 V):

$$t_{HL} = t_{PHL} + t_H = 3 \text{ ns} + 21 \text{ ns} = 24 \text{ ns}$$

El tiempo $\mathbf{t_V}$ es similar al que se ha calculado para la tarjeta conectada al PC. Sin embargo aquí no habrá que preocuparse de no comenzar una transmisión antes de que se desactive VT, puesto que el microcontrolador 68HC11 detecta que la transmisión ha finalizado mediante una interrupción que se produce por flanco de bajada en la señal VT (cuando ésta se desactiva).

En la figura 54 se muestra el ciclo de la recepción de un dato por el Microbot, para a continuación explicarlo detenidamente.

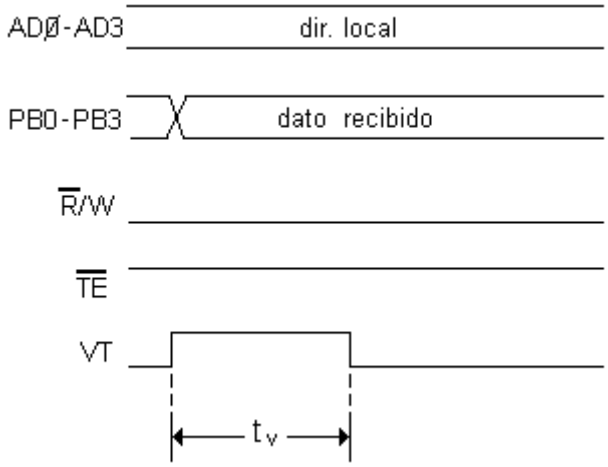


Fig. 54: Ciclo de una recepción del Microbot

Para que se produzca la recepción de un dato será necesario que la señal nR/W esté a '0' (tarjeta en recepción), con lo que la dirección local del Microbot estará en la entrada de dirección del decodificador y la entrada de datos de éste estará conectada con el receptor de RF. Cuando se produzca un flanco de bajada en la señal VT, se generará una interrupción en el Microbot, el cual detectará que ha recibido un dato (si no estaba transmitiendo), procediendo a leer dicho dato después de configurar los pines 0 a 3 del PUERTO D como entradas.

2 DISEÑO SOFTWARE

2.1 Introducción

El diseño del *Software* que estará presente en el PC se ha estructurado en tres niveles o bloques. El primero y más cercano a la máquina está representado por la librería COMUNICACIÓN (*com.h*), donde se incluyen las funciones que permiten utilizar el Hardware diseñado. En un segundo nivel se encuentra la librería PROTOCOLO (*protocol.h*), la cual aporta las funciones que constituyen el protocolo de comunicación diseñado. El tercer y último nivel esta formado por el programa *interfaz.c*, que implementa el panel de control desde donde el usuario podrá enviar comandos al Microbot y visualizar la información procedente de éste.

A continuación se muestra detenidamente el desarrollo de dichos bloques partiendo desde el más cercano al Hardware hasta llegar al más próximo al usuario, así como el del programa en ensamblador diseñado para ejecutarse en el Microbot.

2.2 Desarrollo de la Librería COMUNICACIÓN

Esta librería ('*com.h*') es la que trabaja directamente con el Hardware, proporcionando funciones que son utilizadas por los programas que se verán más adelante, liberando a estos de trabajar a bajo nivel al proporcionar una interfaz transparente al usuario.

En esta librería se encuentran las funciones que permiten:

- El envío de datos desde el PC, con y sin temporización.
- La lectura de los datos recibidos, con y sin temporización.
- La configuración de la dirección local del PC.
- La configuración de la interrupción del Puerto Paralelo

El código de esta librería es el que se muestra a continuación:

```
//com.h
/* Libreria que contiene las funciones basicas para realizar
una comunicacion via radio entre el PC y distintos Microbots */

/*----------------------Librerias incluidas--------------------*/

#include <time.h>
#include <dos.h>

/*----------------------Definicion de constantes---------------*/

#define TRUE    1
#define FALSE   0
#define PIC     0x20    /* Programmable Interrupt Controller (PIC) Base Address */
#define MASK    PIC+1
#define IRQ     7       /* interrupcion del Puerto Paralelo */
#define CLK_TCK 18.2
#define TETX    0.17    /* Tiempo de espera en transmision */
#define TERX    0.25    /* Tiempo de espera en recepcion */

/*----------------------Variables globales----------------------*/

unsigned short int transmitiendo;      /* Variable que indica si se esta transmitiendo */
unsigned short int dato_recibido;      /* variable que indica que se ha recibido un dato */
unsigned short int dir_local = 0;      /* direccion local del PC */
unsigned int DATA = 0x378;             /* Inicializacion de las dir. del PP con los valores standard */
unsigned int STATUS = 0X379;
unsigned int CONTROL = 0X37A;
```

```
unsigned short int intlev = IRQ+0x08;  /* 0x0F */
void interrupt far (*oldhandler)();

/*----------------------Funciones-------------------------------*/

unsigned short int enviar(unsigned short int dir,unsigned short int dato);
unsigned short int recibir();
unsigned short int datoListo();
void inicializar();
void terminar();
void setDirLocal(unsigned short int dir);
unsigned short int getDirLocal();
unsigned short int enviarNibble(unsigned short int dir, unsigned short int comando);
unsigned short int recibirNibble(unsigned short int *dato);
void interrupt far intserv();     /* Interrupt Service Routine (ISR) */

/*----------------------Procedimiento enviar---------------------*/

unsigned short int enviar(unsigned short int dir, unsigned short int dato)
{
    clock_t t1,t2;
    t1=clock();
    while ((transmitiendo==TRUE) || (inportb(STATUS) & 0x40))
    {
          t2=clock();
          if ( ((t2-t1)/CLK_TCK) > TETX ) {
                    return(FALSE);  // Retorna que no se ha podido comenzar la transmision
          }
    }
    transmitiendo=TRUE;                       // indica que se esta transmitiendo
    outportb(DATA,(dir<<4) | (dato & 0x0F));  // configura el modem con la direccion local del PC
    outportb(CONTROL,0x11);     // pone el modem en TX e inicia la transmision activando /TE
    delay(0.0001);              // espera tw
    outportb(CONTROL,0x10);     // desactiva /TE
    outportb(MASK,inportb(MASK) & ~0x80);  // habilita la interrupcion del PP
    return(TRUE);               // Retorna que ha comenzado la transmision correctamente
}

/*----------------------Funcion recibir---------------------------*/

unsigned short int recibir()    // Lee el dato recibido
{
    unsigned short int dato,temp;
    dato=inportb(STATUS);                 // Lee el dato del registro STATUS del PP
    dato=dato >> 3;
    temp=~(dato | ~0x10);
    temp=temp >> 1;
    dato=(dato & 0x07);
    dato=(dato | temp);
    dato_recibido=FALSE;                 // Indica que se ha leido el dato recibido
    outportb(CONTROL,0x12);              // Configura el modem en Recepcion
    outportb(MASK,inportb(MASK) & ~0x80);  // habilita la interrupcion del PP
    return(dato);                         // Retorna el dato leido
}

/*----------------------Funcion dato listo-------------------------*/

unsigned short int datoListo()
{
          return(dato_recibido);  // Retorna si se ha recibido algun dato
}

/*----------------------Procedimiento inicializar------------------*/

void inicializar()   // Inicializa el Modem
{
    unsigned int far *xptr;
    xptr=(unsigned int far *)0x00000408;
    DATA=*xptr;                           // Establece las direcciones del PP para
    STATUS=DATA+1;                        // para la maquina en particular
    CONTROL=DATA+2;                       // a partir de la tabla de Look Up que proporciona la BIOS
    dato_recibido=FALSE;
    transmitiendo=FALSE;
    outportb(CONTROL,0x12);              // habilita la interrupcion del PP por ACK y pone el modem en RX
    outportb( DATA , dir_local<<4 );
    oldhandler = getvect(intlev);         // salva el anterior vector de interrupcion
    setvect(intlev, intserv);                     // direcciona la rutina de atencion a la interrupcion
    outportb(MASK,inportb(MASK) & ~0x80);  // habilita la interrupcion del PP
}

/*----------------------Procedimiento terminar-----------------------*/

void terminar()   // restaura el sistema a sus valores iniciales
{
    setvect(intlev, oldhandler);                 // restaura el anterior vector de interrupcion
    outportb(MASK,inportb(MASK) | 0x80);        // deshabilita la interrupcion del PP
    outportb(CONTROL,0x02);                     // deshabilita la interrupcion del PP por ACK
}

/*----------------Procedimiento cambiar direccion local--------------*/

void setDirLocal(unsigned short int dir)
{
          dir_local=dir;      // Establece una nueva direccion local del PC
          outportb( DATA , (inportb(DATA) & 0x0F) | (dir_local<<4) );
}

/*----------------Procedimiento obtener direccion local--------------*/

unsigned short int getDirLocal()
{
          return(dir_local);   // Retorna la actual direccion local del PC
```

```
}

/*----------------------Funcion enviar nibble----------------------*/

unsigned short int enviarNibble(unsigned short int dir, unsigned short int comando)
{                  // Funcion que transmite un dato asegurandose de que la transmision se reliza correctamente
    clock_t t1,t2;
    if ( enviar(dir,comando)==FALSE ) {
          transmitiendo=FALSE;
          dato_recibido=FALSE;
          outportb( DATA , (inportb(DATA) & 0x0F) | (dir_local<<4) ); // Configura el modem con la direccion local del
PC
          outportb(CONTROL,0x12);                                    // Configura el modem en recepcion
          return(FALSE);    // Retorna que no se ha completado la transmision satisfactoriamente
    }
    t1=clock();
    t2=t1;
    while ( (transmitiendo==TRUE) && ( ((t2-t1)/CLK_TCK) < TETX ) )
    {
          t2=clock();
    }
    if (transmitiendo==TRUE) {
          transmitiendo=FALSE;
          dato_recibido=FALSE;
          outportb( DATA , (inportb(DATA) & 0x0F) | (dir_local<<4) ); // Configura el modem con la direccion local del
PC
          outportb(CONTROL,0x12);
          return(FALSE);    // Retorna que no se ha completado la transmision satisfactoriamente
    } else if ( comando==recibir() ) {  // Lee el dato del modem para comprobar que se haya transmitido correctamente
          return(TRUE);     // Retorna que se ha completado la transmision satisfactoriamente
    } else {
          return(FALSE);    // Retorna que no se ha completado la transmision satisfactoriamente
    }
}

/*----------------------Funcion recibir nibble----------------------*/

unsigned short int recibirNibble(unsigned short int *dato)
{                  // Funcion que espera un tiempo TERX al recepcion de un dato
    clock_t t1,t2;
    t1=clock();
    t2=t1;
    outportb(CONTROL,0x12);  // se asegura que el modem esta en Recepcion
    while ( (dato_recibido==FALSE) && ( ((t2-t1)/CLK_TCK) < TERX ) )
    {
          t2=clock();
    }
    if (dato_recibido==TRUE) {
          *dato = recibir();  // Lee el dato recibido
          return(TRUE);       // Retorna que se ha recibido un dato antes de transcurrido TERX
    } else {
          return(FALSE);      // Retorna que no se ha recibido un dato antes de transcurrido TERX
    }
}

/*----------------Rutina de atencion a la interrupcion del PP--------*/

void interrupt far intserv()
{
    outportb(MASK,inportb(MASK) | 0x80);         // deshabilita la interrupcion del PP
    outportb(PIC,0x20);                          // indica que la interrupcion ha sido procesada
    if (transmitiendo==TRUE) {                  // comprueba si se estaba o no transmitiendo
          transmitiendo=FALSE;                   // indica que se ha terminado de transmitir
          dato_recibido=FALSE;
          outportb( DATA , (inportb(DATA) & 0x0F) | (dir_local<<4) );  // Configura el modem con la direccion local
del PC
          outportb(CONTROL,0x12);                // configura el modem en recepcion
          outportb(MASK,inportb(MASK) & ~0x80);   // habilita la interrupcion del PP
    }
    else {
          outportb(CONTROL,0x10);                       // pone el puerto en TX para no machacar el dato recibido
          dato_recibido=TRUE;                    // indica que se ha recibido un dato
    }
}
```

En las funciones enviar y recibir se puede observar como se siguen los pasos descritos en los ciclos de transmisión y recepción definidos en la parte de diseño Hardware de la tarjeta conectada al PC.

En esta librería existen dos constantes que es necesario comentar: TETX y TERX.

- TETX es el tiempo máximo (en segundos) que espera la función '*enviarNibble*' a que la transmisión termine. Si dicha función, después de comenzada una transmisión, no detecta el final de ésta antes de transcurridos TETX segundos, devolverá un error indicando que la conexión con el sistema transmisor no es correcta.
- TERX es el tiempo máximo (en segundos) que espera la función '*recibirNibble*' (utilizada para esperar la recepción de un dato), antes de devolver un error indicando que no se ha recibido un dato cuando se esperaba.

Como puede observarse en el código, la detección del final de las transmisiones y de las recepciones se hace mediante el uso de interrupciones. Cuando esto no sea posible también podría realizarse mediante *poolling*, testeando el software el bit 6 del registro STATUS del Puerto Paralelo (señal VT).

La descripción detallada de las funciones contenidas en esta librería se puede encontrar en el anexo C de este proyecto.

2.3 Desarrollo de la Librería PROTOCOLO

En esta librería ('*protocol.h*') se encuentran las funciones que configuran el protocolo de comunicación diseñado. Este protocolo se basa en comandos de movimiento o peticiones de información que el PC envía a cada Microbot. Estas comunicaciones se dividen en varias transmisiones en cada sentido, siendo siempre el PC el que comienza una comunicación. Así a cada dato que envía el PC al Microbot, éste responde con la transmisión de otro dato hacia el PC. Este proceso se repetirá tantas veces como sea necesario hasta completar la comunicación en cuestión (cada dato transmitido se corresponde con un Nibble).

Para poder controlar la comunicación, el bit más significativo de cada dato transmitido por el PC será un bit de control. Cuando tras la transmisión de un dato el PC reciba una respuesta satisfactoria del Microbot, el valor de este bit cambiará para la siguiente transmisión. Mediante este mecanismo el Microbot sabrá cuando el PC ha recibido su respuesta correctamente, lo que le permitirá interpretar el dato recibido como uno nuevo o como una repetición del anterior. Así, si el bit de control no cambia con respecto al dato recibido anteriormente, el Microbot interpretará que el PC no ha recibido su respuesta y que está transmitiendo de nuevo el mismo dato, con lo que repetirá la respuesta anterior. El uso de un bit de control limita a 3 bits los utilizables en las transmisiones que realiza el PC, lo que no ocurre en las realizadas por el Microbot. A continuación se muestran los diagramas de flujo simplificados del protocolo en el PC y en el Microbot.

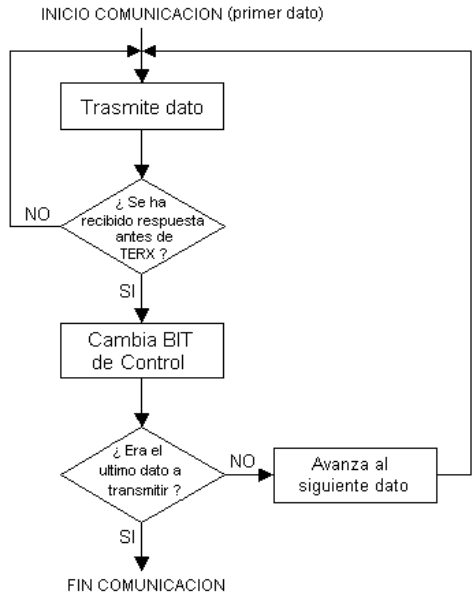


Fig. 55: Diagrama de flujo del protocolo en el PC.

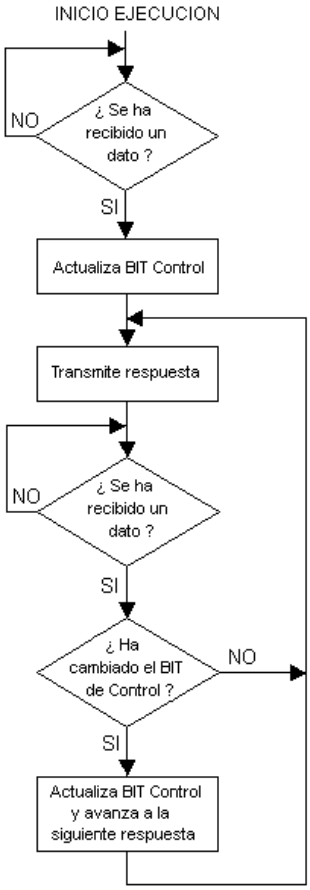


Fig. 56: Diagrama de flujo del protocolo en el Microbot.

En el siguiente diagrama se detalla como ejemplo el flujo de información que se da entre el PC y un Microbot para un comando de movimiento, el cual implica la transmisión de tres Nibbles en cada sentido, en el caso de que no se pierda ningún dato y por lo tanto el PC no tenga que repetir ninguna transmisión.

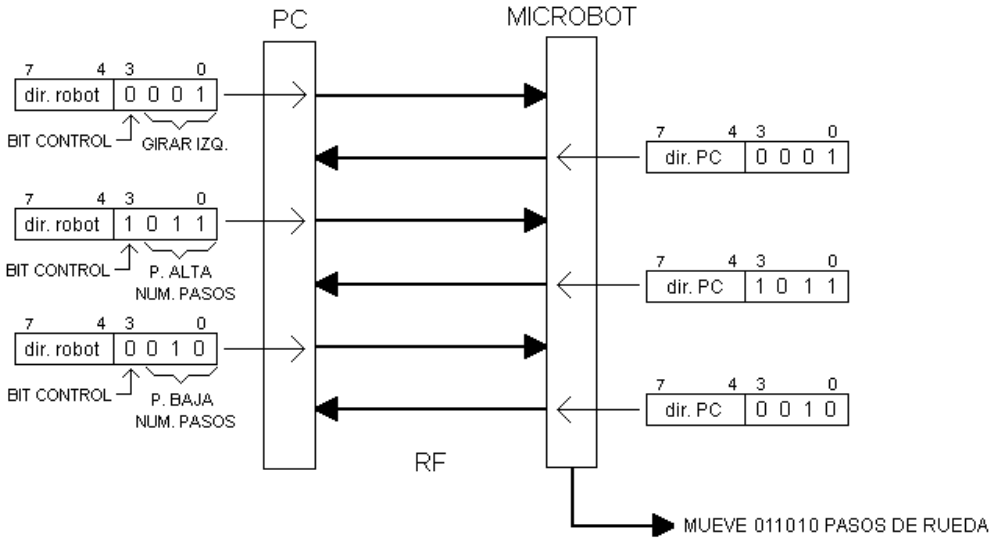


Fig. 57: Proceso de una comunicación.

En la figura 57 se observa como el PC comienza la comunicación enviando un Nibble al Microbot que indica comando de movimiento, a continuación le envía otro Nibble donde los 3 bits menos significativos constituyen la parte alta del numero de pasos que han de girar las ruedas, por ultimo le envía la parte baja del numero de pasos de rueda (otros 3 bits). Para cada Nibble enviado por el PC el Microbot responde con un eco, ejecutando el movimiento una vez finalizada la comunicación correctamente (de este diagrama se deduce que el número máximo de pasos de rueda que se pueden ordenar cada vez es de: $2^6 - 1 = 63$ pasos, ya que se dispone de 6 bits y '000000' indica movimiento indefinido).

La librería PROTOCOLO contiene funciones que permiten:

- Enviar al Microbot comandos de movimiento, ya sean tanto desplazamientos hacia delante o hacia atrás como giros a izquierda o derecha. Para cada movimiento se le indicará al Microbot si éste debe ser indefinido o no. En caso de no serlo se le pasará el número de pasos que han de girar las ruedas.
- Saber cuantos pasos de rueda le restan al Microbot para terminar el último movimiento ordenado desde el PC.
- Leer los puertos del 68HC11 (a estos puertos estarán conectados los sensores en el microbot).
- Leer el estado de sensores conectados a las entradas analógicas del Microbot, utilizando los conversores analógico-digitales presentes en el 68HC11.

El código de esta librería es el que se muestra a continuación.

```
// protocol.h
/* Libreria que incluye las funciones que definen el
protocolo de comunicacion entre el PC y el microbot */

/*-----------------Librerias incluidas-------------------------------------*/

#include "com.h"

/*-----------------Definicion de Constantes--------------------------------*/

#define NUMREP  100  // Numero de veces que repite la transmision de un dato hasta obtener respuesta
#define ADELANTE  0
#define IZQUIERDA 1
#define DERECHA   2
#define ATRAS     3
#define PARO      4
#define PORTA     0
#define PORTB     1
#define PORTC     2
#define PORTD     3
#define PORTE     4

/*--------------------Variables globales---------------------------------*/

unsigned short int mask[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};  // Mascaras que determinan el actual bit de control
para cada Microbot

/*-----------------Declaracion Funciones -----------------------------------*/

unsigned short int avanzar(unsigned short int robot, unsigned short int n);
unsigned short int retroceder(unsigned short int robot, unsigned short int n);
unsigned short int girarDerecha(unsigned short int robot, unsigned short int n);
unsigned short int girarIzquierda(unsigned short int robot, unsigned short int n);
unsigned short int parar(unsigned short int robot);
unsigned short int pasos(unsigned short int robot, unsigned short int *dato);
unsigned short int puerto(unsigned short int robot, unsigned short int puerto, unsigned short int *dato);
unsigned short int canalAnalog(unsigned short int robot, unsigned short int canal, unsigned short int *dato);

unsigned short int comandoMov(unsigned short int robot, unsigned short int pasos, unsigned short int dir);
unsigned short int peticion(unsigned short int robot, unsigned short int option, unsigned short int num, unsigned short
int *dato);
unsigned short int changeMask(unsigned short int robot);

/*----------------Implementacion Funciones-------------------------------------------*/

/*-- Funciones para ordenar movimientos al robot (llaman a la funcion ComandoMov) --*/

unsigned short int avanzar(unsigned short int robot, unsigned short int n)
{
```

```
 return(comandoMov(robot,n,ADELANTE));
}
unsigned short int retroceder(unsigned short int robot, unsigned short int n)
{
 return(comandoMov(robot,n,ATRAS));
}
unsigned short int girarDerecha(unsigned short int robot, unsigned short int n)
{
 return(comandoMov(robot,n,DERECHA));
}
unsigned short int girarIzquierda(unsigned short int robot, unsigned short int n)
{
 return(comandoMov(robot,n,IZQUIERDA));
}

/*-- Funcion para solicitar al Microbot la lectura de un puerto --*/

unsigned short int puerto(unsigned short int robot, unsigned short int puerto, unsigned short int *dato)
{
         return(peticion(robot,0,puerto,dato));
}

/*-- Funcion para solicitar al Microbot una conversion A/D --*/

unsigned short int canalAnalog(unsigned short int robot, unsigned short int canal, unsigned short int *dato)
{
         return(peticion(robot,1,canal,dato));
}

/*-- Funcion para cambiar el BIT de Control para las tarnsmisiones a un Microbot --*/

unsigned short changeMask(unsigned short int robot)
{
          if (mask[robot]==0x00) {
              mask[robot]=0x08;
          } else {
              mask[robot]=0x00;
          }
          return(TRUE);
}

/*-- Funcion para ordenar a un Microbot que se pare --*/

unsigned short int parar(unsigned short int robot)
{
          unsigned short int acierto,aciertoModem,i,d;
          acierto=FALSE;
          i=0;
          do {
                    aciertoModem=enviarNibble(robot,mask[robot] | 0x4);                 // envia dato
                    if (aciertoModem==FALSE) { break; }
                    if ( (recibirNibble(&d)==TRUE) ) { //&& (d==mask[robot] | 0x4) ) {
                              acierto=TRUE;
                    }
                    i++;
          } while ( (acierto==FALSE) && (i<NUMREP) );
          changeMask(robot);                        // Cambia BIT de control
          if (aciertoModem==FALSE) { return(2); }  // Retorna que ha fallado la conexion con el modem
          else if (acierto==FALSE) { return(1); }  // Retorna que no se ha podido completar la comunicacion con el
robot
          else { return(0); }                       // Retorna que la comunicacion se ha completado satisfactorimente
}

/*-- Funcion para ordenar un movimiento a un Microbot --*/

unsigned short int comandoMov(unsigned short int robot, unsigned short int pasos, unsigned short int dir)
{
          unsigned short int d,acierto,aciertoModem,aciertoParar,i,j;
          unsigned short int nibble[3];
          if (dir==PARO) {
              return(parar(robot));
          }
          nibble[0]=dir;
          nibble[2]=(pasos & 0x07);
          nibble[1]=( (pasos>>3) & 0x07 );
          i=0;
          aciertoParar=0;
          do {
                    j=0;
                    do {
                              aciertoModem=enviarNibble( robot , mask[robot] | nibble[i] );
                              if (aciertoModem==FALSE) { break; }
                              if ( (recibirNibble(&d)==TRUE) ) { //& (d==mask[robot] | nibble[i]) ) {
                                        acierto=TRUE;
                              } else {
                                        acierto=FALSE;
                              }
                              j++;
                    } while ( (acierto==FALSE) && (j<NUMREP) );
                    changeMask(robot);
                    i++;
          } while ( (acierto==TRUE) && (i<3) && (aciertoModem==TRUE) );
          if ((aciertoModem==FALSE)||(aciertoParar==2)) { return(2);
          } else if (acierto==FALSE) { return(1);
          } else { return(0); }
}

/*-- Funcion para solicitar al robot el numero de pasos restantes para finalizar el ultimo movimiento --*/

unsigned short int pasos(unsigned short int robot, unsigned short int *dato)
{
          unsigned short int i,j,acierto,d,aciertoModem;
```

```
          unsigned short int nibbleOut[2];
          unsigned short int nibbleIn[2];
          nibbleOut[0]=0x05;
          nibbleOut[1]=0x00;
          i=0;
          do {
                    j=0;
                    do {
                              aciertoModem=enviarNibble( robot , mask[robot] | nibbleOut[i] );
                              if (aciertoModem==FALSE) { break; }
                              if (recibirNibble(&d)==FALSE) {
                                        acierto=FALSE;
                              } else {
                                        acierto=TRUE;
                                        if(i==0) { nibbleIn[1]=d;            // Almacena el Nibble alto
                                        } else if (i==1) { nibbleIn[0]=d; }  // Almacena el Nibble bajo
                              }
                              j++;
                    } while ( (acierto==FALSE) && (j<NUMREP) );
                    changeMask(robot);
                    i++;
          } while ( (acierto==TRUE) && (i<2) && (aciertoModem==TRUE) );
          if (aciertoModem==FALSE) { return(2);
          } else if (acierto==FALSE) {
                    return(1);
          } else {
                    *dato=( (nibbleIn[1]<<4) | (nibbleIn[0]) );
                    return(0);
          }
}

/*--- Funcion para solicitar al robot la lectura de un puerto o una conversion A/D --*/

unsigned short int peticion(unsigned short int robot, unsigned short int option, unsigned short int num, unsigned short
int *dato)
{
          unsigned short int i,j,acierto,d,aciertoModem;
          unsigned short int nibbleOut[3];
          unsigned short int nibbleIn[2];
          nibbleOut[0]=0x06+option;              // Puerto o conversion A/D
          nibbleOut[1]=num;
          nibbleOut[2]=0x00;
          i=0;
          do {
                    j=0;
                    do {
                              aciertoModem=enviarNibble( robot , mask[robot] | nibbleOut[i] );
                              if (aciertoModem==FALSE) { break; }
                              if ( (recibirNibble(&d)==FALSE) ) {  //| ( (i==0) && (d!=mask[robot] | nibbleOut[i]) ) )
{
                                        acierto=FALSE;
                              } else {
                                        acierto=TRUE;
                                        if(i==1) { nibbleIn[1]=d;             // Almacena el Nibble alto
                                        } else if (i==2) { nibbleIn[0]=d; }   // Almacena el Nibble bajo
                              }
                              j++;
                    } while ( (acierto==FALSE) && (j<NUMREP) );
                    changeMask(robot);
                    i++;
          } while ( (acierto==TRUE) && (i<3) && (aciertoModem==TRUE) );
          if (aciertoModem==FALSE) { return(2);
          } else if (acierto==FALSE) {
                    return(1);
          } else {
                    *dato=( (nibbleIn[1]<<4) | (nibbleIn[0]) );
                    return(0);
          }
}
```

Como se puede observar, esta librería utiliza a su vez las funciones que le proporciona la librería '*com.h*'. Sobre todo utiliza las funciones '*enviarNibble*' y '*recibirNibble*', la primera usada para enviar datos y la segunda para esperar a la recepción de un dato.

La constante NUMREP indica el número de veces que el PC repetirá una transmisión hasta obtener una respuesta por parte del Microbot.

La descripción detallada de las funciones contenidas en esta librería se puede encontrar en el anexo C de este proyecto.

2.4 Desarrollo del PANEL DE CONTROL utilizando *LABWINDOWS*

En este apartado se presenta el programa realizado para demostrar como se pueden controlar los Microbots utlizando las funciones contenidas en las librerias desarrolladas. Este programa se compilará y linkará en *TurboC* pero utilizará librerias de la *User Interface Library* de *Labwindows.*

Labwindows es un sistema de desarrollo que proporciona un entorno interactivo para la generación y depuración de programas *[García.B.96]*. Este entorno de desarrollo utiliza los lenguajes *QuickBASIC* o C, a elección del usuario, no siendo éstos normalizados sino específicos de éste. Dispone además de multitud de librerías de funciones para control de instrumentación (GPIB, VXI) y manejo de tarjetas de adquisición, lo que permite una gran versatilidad y facilidad en el intercambio de información ordenador-instrumentación.

Pero la contribución más importante de *Labwindows* a este proyecto es el "*User Interface Editor"*, con el que se pueden realizar paneles gráficos de control (mandos e indicadores), cuadros de dialogo, menús desplegables, etc., que serán sobre los que opere el usuario de programa. Estos objetos que se generan y editan con el *User Interface Editor,* se manejan desde programa con un conjunto de funciones específicas que se agrupan bajo la "*User Interface Library".*

Para la realizar la programación en lenguje C necesaria en este proyecto se ha utilizado el entorno *TurboC 3.0* de *Borland*, por utilizar el lenguaje C normalizado o ANSI C. Pero a la hora de crear el panel de control que se mostrará al usuario se ha utilizado el *User Interface Editor* proporcionado por *Labwindows*, ya que facilita mucho la creación de paneles graficos, no encontrandose nada similar en el entorno TurboC.

Los objetos generados por el *User Interface Editor* se almacenan en ficheros que tienen la extensión por defecto *fichero.*uir. Para que estos objetos se muestren en pantalla, el programa debe recurrir a las funciones de la *User Interface Library* que cargan y muestran el objeto. Las intervenciones posteriores del usuario o del programa sobre el objeto también se procesan mediante funciones de la citada librería.

El flujo general de proceso cuando se emplea un panel de control es el siguiente:

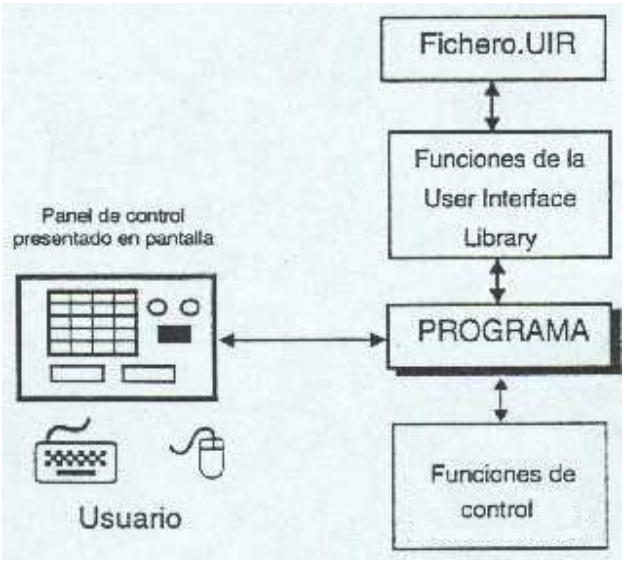


Fig. 58: Flujo de información en un panel

Los paneles de control creados con el *User Interface Editor* se almacenan en ficheros con extensión 'uir'. Un *fichero.uir* puede contener más de un panel, que se diferencian a la hora de invocarlos por el nombre introducido en el campo *Constant prefix.*

Ha de tenerse en cuenta que cada control definido en el panel (pulsador, *display*, etc.) tiene asociado el nombre de una constante. Por ejemplo, un interruptor de encendido puede tener asociada una constante llamada *power*, un control para elegir un número de ciclos puede tener asociada una constante llamada *num_cycles* y así sucesivamente. El nombre total de la constante lo confecciona el editor en automático anteponiendo el del panel. Así, la constante asociada al interruptor de encendido, si se ha rellenado el campo *Constant prefix* con "P", se llama *P_power*, la del control del número de ciclos *P_num_cycles*, etc. El editor asigna valores secuenciales a estas constantes en un fichero *fichero.H* que se genera automáticamente y que es necesario incluir en el programa que utilice el panel. Estas constantes, que identifican el control, serán el "lazo" de unión del panel con el programa.

Las acciones que el usuario realiza sobre los controles presentes en el panel generan eventos. Un evento es una información que manda el control al programa indicando que ha sido manipulado. No siempre que se manipula un control se produce un evento, solo cuando el control ha sido configurado para ello. Cuando el programa detecta un evento debe decidir que acción tomar.

La "*User Interface Library"* es muy extensa y potente. Las dos funciones más significativas de esta librería son '*GetUserEvent*' y '*SetCtrlVal*', usadas respectivamente para detectar eventos en los controles y para fijar el valor de los indicadores. Ambas funciones son usadas en el programa creado para gestionar el panel (*interfaz.c)*. El código correspondiente a dicho programa se muestra a continuación, debidamente comentado para su fácil comprensión.

```
//interfaz.c
/* Programa que muestra un panel desde donde controlar los Microbots */

/*--------------------Librerias incluidas----------------*/

#include <math.h>
#include "protocol.h"
#include "panel.h"
#include "userint.h"

/*--------------------Definicion de Constantes-----------*/

#define DIMGRAPH 200        //Dimension de la grafica
#define MODULO   1          //Distancia a avanzar en la grafica por cada paso de rueda
#define PI       3.14
#define P2PI     32         //Pasos de rueda necesarios para girar 360 grados
#define ALPHA    2*PI/P2PI  // angulo de giro por cada paso en radianes
#define ALPHAG   360.0/P2PI // angulo de giro por cada paso en grados

/*--------------------Variables globales------------------*/

int panel_control;
double x = DIMGRAPH/2.0;    // coordenada x incial del robot en la grafica
double y = DIMGRAPH/2.0;    // coordenada y incial del robot en la grafica
short int dir = 0;          // rumbo inicial del microbot

/*--------------------Declaracion de funciones------------*/

void dibujarPunto(short int sentido, unsigned short int pasos);
void girarPunto(short int sentido, unsigned short int pasos);

/*--------------------Programa principal------------------*/

void main()
{
   int controla;
   unsigned short int robot = 15;
   unsigned short int numPasos = 0;
   unsigned short int temp,temp2,temp3,dato;

   panel_control = LoadPanel("panel.uir",MAIN);  /* Cargamos el panel */
   DisplayPanel(panel_control);                  /* Mostramos el panel */
   inicializar();                                /* Inicializamos el Modem */
   SetCtrlVal(panel_control, MAIN_DIRLOCAL, getDirLocal()); /* Inicializamos indicador de Dir Local */

   DeletePlots (panel_control, MAIN_GRAPH);      /* Limpiamos la grafica */
   PlotPoint (panel_control, MAIN_GRAPH, x, y, 0, 1);  /* Dibujamos posicion del robot en la grafica */
```

```
   SetCtrlVal(panel_control, MAIN_RUMBO, ALPHAG*dir);  /* Se inicializa el rumbo del robot */

   MessagePopup("Pulse SALIR para terminar");

   while(TRUE)
       {
       while(GetUserEvent(0, &panel_control, &controla) == 0) {  /*Esperamos a que se utilce un control en el panel
principal*/
               if(datoListo()==TRUE) {
                           recibir();
                           SetCtrlVal(panel_control, MAIN_CONSOLA, "Dato recibido.");
               }
       }
       GetCtrlVal(panel_control, MAIN_ROBOT, &robot);
       switch(controla) {
          case MAIN_SALIR:       /* el usuario ha pulsado salir */
            terminar();
            UnloadPanel (panel_control);
            return;
          case MAIN_DELANTE:     /* avanzar */
            GetCtrlVal(panel_control, MAIN_NUMPASOS, &numPasos);
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Enviando comando AVANZAR.");
            temp=avanzar(robot,numPasos);
            if (temp==2) {
                 MessagePopup("Fallo en la conexion con el modem.");
            } else if (temp==1) {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Imposible conectar con el robot.");
            } else {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Avanzando.");
                 dibujarPunto(1,numPasos);
            }
            break;
          case MAIN_ATRAS:      /* retroceder */
            GetCtrlVal(panel_control, MAIN_NUMPASOS, &numPasos);
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Enviando comando RETROCEDER.");
            temp=retroceder(robot,numPasos);
            if (temp==2) {
                 MessagePopup("Fallo en la conexion con el modem.");
            } else if (temp==1) {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Imposible conectar con el robot.");
            } else {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Retrocediendo.");
                 dibujarPunto(-1,numPasos);
            }
            break;
          case MAIN_DERECHA:     /* girar a la derecha */
            GetCtrlVal(panel_control, MAIN_NUMPASOS, &numPasos);
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Enviando comando GIRAR DERECHA.");
            temp=girarDerecha(robot,numPasos);
            if (temp==2) {
                 MessagePopup("Fallo en la conexion con el modem.");
            } else if (temp==1) {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Imposible conectar con el robot.");
            } else {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Girando a la derecha.");
                 girarPunto(1,numPasos);
            }
            break;
          case MAIN_IZQUIERDA:    /* girar a la izquierda */
            GetCtrlVal(panel_control, MAIN_NUMPASOS, &numPasos);
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Enviando comando GIRAR IZQUIERDA.");
            temp=girarIzquierda(robot,numPasos);
            if (temp==2) {
                 MessagePopup("Fallo en la conexion con el modem.");
            } else if (temp==1) {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Imposible conectar con el robot.");
            } else {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Girando a la izquierda.");
                 girarPunto(-1,numPasos);
            }
            break;
          case MAIN_STOP:         /* parar */
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Enviando comando PARAR.");
            temp=parar(robot);
            if (temp==2) {
                 MessagePopup("Fallo en la conexion con el modem.");
            } else if (temp==1) {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Imposible conectar con el robot.");
            } else {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Parado.");
            }
            break;
          case MAIN_DIRLOCAL:     /* establecer nueva direccion local de PC */
            GetCtrlVal(panel_control, MAIN_DIRLOCAL, &temp);
            setDirLocal(temp);
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Establecida nueva direccion local del PC.");
            break;
          case MAIN_PASOS:        /* solicitar numero de pasos restantes para finalizar el movimiento */
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Solicitando numero de pasos restantes.");
            temp=pasos(robot,&numPasos);
            if (temp==2) {
                 MessagePopup("Fallo en la conexion con el modem.");
            } else if (temp==1) {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Imposible conectar con el robot.");
            } else {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Recibido numero de pasos.");
                 SetCtrlVal(panel_control, MAIN_NUMPASOS, numPasos);
            }
            break;
      case MAIN_CONVER:        /* solicitar conversion A/D */
            GetCtrlVal(panel_control, MAIN_CANAL, &temp);
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Solicitando conversion A/D.");
```

```
            temp2=canalAnalog(robot,temp,&dato);
            if (temp2==2) {
                 MessagePopup("Fallo en la conexion con el modem.");
            } else if (temp2==1) {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Imposible conectar con el robot.");
            } else {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Recibida conversion A/D.");
                 SetCtrlVal(panel_control, MAIN_VALOR, dato);
            }
            break;
      case MAIN_PORT:          /* solicitar lectura de un puerto del Microbot */
            GetCtrlVal(panel_control, MAIN_PUERTO, &temp);
            SetCtrlVal(panel_control, MAIN_CONSOLA, "Solicitando lectura del PUERTO.");
            temp2=puerto(robot,temp,&dato);
            if (temp2==2) {
                 MessagePopup("Fallo en la conexion con el modem.");
            } else if (temp2==1) {
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Imposible conectar con el robot.");
            } else {          // se actualizan los indicadores
                 SetCtrlVal(panel_control, MAIN_LECTURA, dato);
                 SetCtrlVal(panel_control, MAIN_BIT0,dato&0X01);
                 SetCtrlVal(panel_control, MAIN_BIT1,dato&0X02);
                 SetCtrlVal(panel_control, MAIN_BIT2,dato&0X04);
                 SetCtrlVal(panel_control, MAIN_BIT3,dato&0X08);
                 SetCtrlVal(panel_control, MAIN_BIT4,dato&0X10);
                 SetCtrlVal(panel_control, MAIN_BIT5,dato&0X20);
                 SetCtrlVal(panel_control, MAIN_BIT6,dato&0X40);
                 SetCtrlVal(panel_control, MAIN_BIT7,dato&0X80);
                 SetCtrlVal(panel_control, MAIN_CONSOLA, "Dato recibido satisfactoriamente.");
            }
            break;
       }
   }
}

/*----------------------Funciones---------------------------*/

void dibujarPunto(short int sentido, unsigned short int pasos)
{                       //Funcion para desplazar la posicion del robot en la grafica tras un avance o retroceso
   unsigned short int i;
   short int mod;
   double angulo;
   angulo = ALPHA * dir;      // 1-->Adelante; -1-->Atras
   mod = MODULO * sentido;
   for (i=0;i<pasos;i++) {      // Se dibuja la estela del movimiento del robot
     PlotPoint (panel_control, MAIN_GRAPH, x, y, 0, 2);
     x = x + ( mod * sin(angulo) );
     y = y + ( mod * cos(angulo) );
     if (x >= DIMGRAPH) {          // Detectamos si se superan los limites de la grafica
           x=0.0;
           DeletePlots (panel_control, MAIN_GRAPH);
     }else if (x<=0.0) {
           x=DIMGRAPH;
           DeletePlots (panel_control, MAIN_GRAPH);
     }
     if (y>=DIMGRAPH) {
           y=0.0;
           DeletePlots (panel_control, MAIN_GRAPH);
     }else if (y<=0.0) {
           y=DIMGRAPH;
           DeletePlots (panel_control, MAIN_GRAPH);
     }
     PlotPoint (panel_control, MAIN_GRAPH, x, y, 0, 1);  // Se pinta la posicion final del robot
   }
}

void girarPunto(short int sentido, unsigned short int pasos)  //Funcion para establecer el nuevo rumbo del robot tras
un giro
{                       //Funcion para establecer el nuevo rumbo del robot tras un giro
   unsigned short int i;
   double angulo;
   for (i=0;i<pasos;i++) {
          dir = dir+sentido;       // 1-->Derecha; -1-->Izquierda
          if (dir==-1) {                // dir = 0 --> hacia arriba
              dir=P2PI-1;
          }else if (dir==P2PI) {   // 360 grados / 11.25 grados = 32 direcciones posibles
              dir=0;
          }
          SetCtrlVal(panel_control, MAIN_RUMBO, ALPHAG*dir);    // Se actualiza el rumbo
   }
}
```

Se puede ver en el código, como es necesario incluir la librería '*panel.h*', donde se definen las constantes con las que identifican a los distintos elementos del panel. También se observa como para cargar el panel en memoria es necesario pasarle a la función '*LoadPanel*' el nombre del fichero donde se encuentra el panel creado ("*panel.uir*") junto con el identificador del panel principal (MAIN), devolviendo esta función un manejador de panel (*panel_control*), que se le pasa a la función '*DisplayPanel*', para que lo muestre en pantalla (las funciones '*LoadPanel*' y '*DisplayPanel*' están ambas incluidas en la '*User Interface Library*').

Para detectar los eventos en los controles se establece un bucle *while* infinito, donde se espera a que ocurra un evento para a continuación tratarlo. Solo se saldrá de este bucle infinito cuando se pulse el botón salir, entonces se descargará el panel de la memoria y el programa terminará.

Cuando se ejecute una acción sobre el panel que implique una comunicación con algún robot, el programa utilizará las funciones contenidas en la librería '*protocol.h*'. Al final de una comunicación el PC informará al usuario si esta se ha podido completar de forma satisfactoria, y en caso de una petición de información al Microbot mostrará por pantalla la información obtenida.

Para poder utilizar en *TurboC* funciones de la '*User Interface Library*' ha sido necesario crear un proyecto de *TurboC*, que incluyera el programa '*interfaz.c*' y las librerías '*userint.lib*' y '*lwc.lib*' (proporcionadas por Labwindows). Este proyecto se ha nombrado como '*proyecto.prj*', con lo se ha generado un fichero ejecutable llamado '***proyecto.exe***'. El panel que se presenta al ejecutar este archivo es el que se muestra en la figura 59.

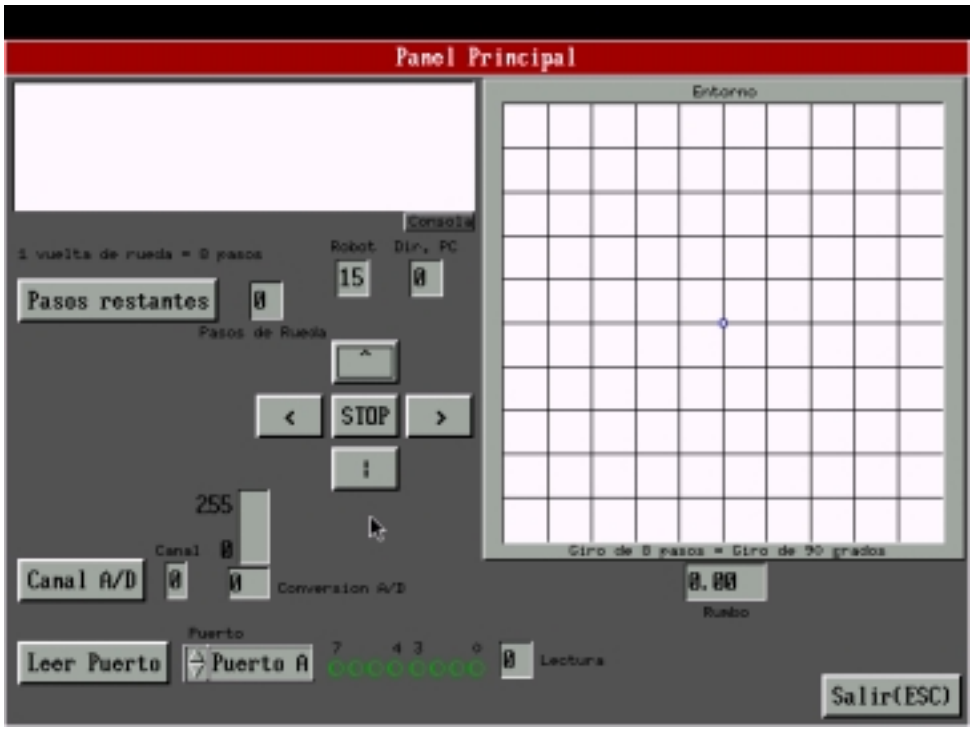


Fig. 59: Panel de control.

En el panel se pueden distinguir los siguientes controles e indicadores:

- Casilla para introducir la dirección local del PC y la dirección del Microbot al que se dirigen las transmisiones.
- Controles de movimiento (adelante, atrás, izquierda, derecha y stop).
- Casilla para introducir y leer el número de pasos de ruedas a enviar o recibidos.
- Botón para solicitar al Microbot el número de pasos que le restan para terminar un movimiento.
- Control para seleccionar el puerto del Microbot a leer.
- Botón para leer uno de los puertos del PC.

- Casilla para seleccionar el canal analógico en caso de una conversión A/D.
- Botón para solicitar el resultado de la conversión A/D de uno de los canales analógicos del Microbot.
- Indicadores de la lectura de los puertos y de los canales analógicos.
- Consola que informa de los resultados de las comunicaciones.
- Grafica para visualizar la ruta seguida por el Microbot.
- Indicador de rumbo del Microbot.
- Botón para salir de la aplicación.

La descripción detallada de las funciones contenidas en este programa se puede encontrar en el anexo C de este proyecto.

2.5 Desarrollo del Programa en ENSAMBLADOR para el 68HC11

En este último bloque se detalla el programa que estará permanentemente en ejecución en el Microbot, y que se almacenará en la memoria EEPROM del microcontrolador Motorola 68HC11. Éste programa es capaz de interpretar las los mensajes que le envía el PC y de contestar según el protocolo diseñado hasta que se completa cada comunicación (comando o petición de información).

El código de este programa se detalla a continuación (debidamente comentado para su fácil compresión). En él se puede observar del flujo de ejecución con el que se consigue implementar el protocolo de comunicación (tal como se indica en la figura 56), así como las subrutinas que permiten enviar datos, recibir datos y llevar el control de la comunicación.

```
;PROYECTO.ASM
;Programa para el control de un microbot mediante un enlace radio bidireccional

OPTION  EQU $39
PIOC    EQU $02
PORTA   EQU $00
PORTB   EQU $04
PORTC   EQU $03
PORTD   EQU $08
DDRD    EQU $09
PORTE   EQU $0A
SCCR2   EQU $2D
SPCR    EQU $28
HPRIO   EQU $3C
TMSK1   EQU $22
TFLG1   EQU $23
TMSK2   EQU $24
TFLG2   EQU $25
PACTL   EQU $26
PACNT   EQU $27
TCTL1   EQU $20
TOC4    EQU $1C
TCNT    EQU $0E
ADCTL   EQU $30
ADR1    EQU $31
DIRPC   EQU $00
TIEMPO  EQU 65000  ; Numero de pulsos de reloj necesarios para generar un
                   ; retraso 32,5 ms. Cada tic de reloj son 0.5 us
                   ; 65000*0.5 = 32500 us = 32,5 ms.
;Variables

        ORG $0000        ; RAM

dato_recibido  RMB 1     ; Variables que no se inicializan, solo se reserva un Byte
dato           RMB 1
mask           RMB 1
temp           RMB 1

;Programa

        ORG $B600        ; EEPROM
```

```
config  LDX #$1000
        LDS #$00B4       ; DIR 180 Comienzo de al pila

        LDAA #$00
        STAA  SCCR2,X    ; SCI deshabilitado

        LDAA #$30
        STAA  SPCR,X     ; SPI deshabilitado y salidas OC
        STAA  DDRD,X     ; BITS 0 a 3 del puerto D entradas, BITS 4 y 5 salidas

        LDAA #$00
        STAA TMSK1,X     ; No permitir la interupci¢n de los comparadores
        STAA PACTL,X     ; Bit 7 Puerto A entrada, ac de pulsos desactivado
        STAA dato_recibido

        LDAA  #$20
        STAA  PORTD,X    ; BIT 4 a 0 y BIT 5 del puerto D a 1
        STAA OPTION,X    ; IRQ activa por flanco de bajada

        LDAA #$C6
        STAA HPRIO,X     ; Maxima prioridad IRQ

        LDAA #$7E        ; cargamos en el A el codigo de la instruccion JUMP
        STAA $EE         ; principio del vector de interrupcion de la IRQ
        LDD  #int_irq    ; D <- direccion de comienzo de la rutina interrup.
        STD  $EF

        LDAA #$7E        ; cargamos en el A el codigo de la instruccion JUMP
        STAA $CD         ; principio del vector de interrupcion del acumulador
        LDD  #int_ac     ; D <- direccion de comienzo de la rutina interrup.
        STD  $CE

        CLI              ; permite interrupciones

        JSR recibir      ; espera a recbir el primer dato
        JSR unmask       ; obtiene el bit de control del dato recibido
inicio  STAA mask        ; actualiza el bit de control
        CMPB #$04        ; evalua el comando recibido
        BNE c5
        JMP parar
c5      CMPB #$05        ; peticion numero de pasos restantes
        BNE c6
        JMP pasos
c6      CMPB #$06        ; peticion lectura de un puerto
        BNE c7
        JMP puerto
c7      CMPB #$07        ; peticion conversion A/D
        BNE mover        ; si no es ninguna de las anteriores --> Comando movimiento
        JMP analog

mover   ORAB mask        ; enmascara el dato a¤adiendole el bit de control
        JSR enviar       ; lo envia (es un eco)
        ;ANDB #$03        ; me quedo con los dos bits menos significativos (redundante)
        LSLB             ; Desplazo el dato recibido 5 bits a la izquierda
        LSLB
        LSLB
        LSLB
        LSLB
        STAB PORTA,X     ; Si Bits 6 y 5 = 00-->delante ; 11-->detras ; 10-->Derecha ; 01--> Izquierda
em1     JSR recibir      ; espera parte alta del numero de pasos
        JSR enviar       ; envia un eco
        JSR unmask
        CMPA mask        ; comprueba si ha variado el bit de control en el nuevo dato recibido
        BEQ em1          ; si no ha variado vuelve a la espera anterior
        STAA mask        ; actualiza bit de control

        LSLB             ; desplaza el dato 3 bits a la izquierda
        LSLB
        LSLB
        STAB dato        ; almacena los 3 bits mas significativos del numero de pasos
em2     JSR recibir      ; espera parte baja del numero de pasos
        JSR enviar
        JSR unmask
        CMPA mask        ; a¤ade al dato el bit de control
        BEQ em2
        STAA mask        ; actualiza bit de control

        ORAB dato
        STAB dato
        CMPB #$00
        BEQ sinfin       ; Si pasos==0 --> Movimiento sin fin
        NEGB
        ADDB #$FF
        STAB PACNT,X     ; Carga el numero de pasos en el acumulador de pulsos
        BSET TMSK2,X $20 ; Activar interrupcion de overflow del acumulador
        BSET PACTL,X $40 ; Activa el acumulador de pulsos
sinfin  BSET PORTA,X $18 ; Enciende los motores
em3     JSR recibir      ; espera una nueva recepcion
        JSR unmask
        CMPA mask
        BNE fin_mov      ; si es un nuevo comando termina el movimiento
        ORAB mask
        JSR enviar       ; si no envia un eco
        BRA em3          ; y vuelve atras
fin_mov JMP inicio

parar   JSR detener      ; para el Microbot
        ORAB mask        ; a¤ade bit control
        JSR enviar       ; envia eco
        JSR recibir      ; espera una nueva recepcion
        JSR unmask
        CMPA mask
```

```
        BEQ parar        ; si no es un nuevo comando vuelve atras
        JMP inicio

pasos   LDAB #$00
        BRCLR PACTL,X $40 store ; si el robot esta parado devuelve 0
        LDAB #$FF
        SUBB PACNT,X            ; si no devuelve el numero de pasos restantes
store   STAB dato
        JSR des4der
        JSR enviar              ; Envia el nibble alto
        JSR recibir             ; Espero al siguiente dato
        JSR unmask
        CMPA mask
        BEQ pasos               ; si no cambia el bit de control vuelve atras
        STAA mask
ev2     LDAB dato
        JSR enviar              ; Envia el nibble bajo
        JSR recibir             ; espera una nueva recepcion
        JSR unmask
        CMPA mask
        BEQ ev2                 ; si no es un nuevo comando vuelve atras
        JMP inicio

puerto  ORAB mask               ; a¤ade el bit de control
        JSR enviar              ; hace eco
        JSR recibir             ; espera a recibir el puerto
        JSR unmask
        CMPA mask
        BEQ puerto              ; si el bit de control no cambia vuelve atras
        STAA mask
ep2     CMPB #$00               ; lee el puerto correspondiente
        BNE cp1
        LDAB PORTA,X
        BRA fc
cp1     CMPB #$01
        BNE cp2
        LDAB PORTB,X
        BRA fc
cp2     CMPB #$02
        BNE cp3
        LDAB PORTC,X
        BRA fc
cp3     CMPB #$03
        BNE cp4
        LDAB PORTD,X
        BRA fc
cp4     BCLR OPTION,X $80       ;apaga el conversor A/D
        LDAB PORTE,X
fc      STAB dato
        JSR des4der
        JSR enviar              ; envia el nibble alto
        JSR recibir             ; espero a la siguiente recepcion
        JSR unmask
        CMPA mask
        BEQ ep2                 ; si el bit de control no cambia vuelve atras
        STAA mask
          BRA e3

analog  BSET OPTION,X $80       ; encender el conversor A/D
        ORAB mask
        JSR enviar
        JSR recibir             ; espera a recibir el canal
        JSR unmask
        CMPA mask
        BEQ analog              ; si el bit de control no cambia vuelve atras
        STAA mask
        ;ANDB #$07               ; nos quedamos con el numero de canal (redundante)
        STAB ADCTL,X            ; SCAN -> inactivo ; MULT -> inactivo ; ADR1 -> seleccionar primer canal
sigue   BRCLR ADCTL,X $80 sigue ; espera a que termine conversion
ea2     LDAB ADR1,X             ; lee el resultado de la conversion
        STAB dato
        JSR des4der
        JSR enviar              ; Envia el nibble alto
        JSR recibir             ; Espero al siguiente nibble
        JSR unmask
        CMPA mask
        BEQ ea2                 ; si el bit de control no cambia vuelve atras
        STAA mask

e3      LDAB dato
        JSR enviar              ; envia el nibble bajo
        JSR recibir             ; espera una nueva recepcion
        JSR unmask
        CMPA mask
        BEQ e3                  ; si el bit de control no cambia vuelve atras
        JMP inicio

;funciones

enviar  BRSET PORTD,X $10 enviar  ; Subrutina que transmite el dato en el acum B

        STAB temp

        BSR delay                 ; Espera 32,5 ms

        BSET DDRD,X $0F           ; Bits 0 a 3 del PUERTO D salidas

        LDAA #DIRPC
        ;ANDA #$0F
        ORAA #$10
        STAA PORTD,X              ; Escribe la direccion destino e inicia la transmision
```

```
        BSET PORTD,X $20
        BSET PORTD,X $20          ; Se repite para que transcurra el tiempo thl
        BSET PORTD,X $20
        BSET PORTD,X $20

        LDAB temp
        ORAB #$30
        STAB PORTD,X              ; Escribe el dato a transmitir

        ;ANDA #$0F
        ANDB #$0F
esp_tx  BRSET PORTD,X $10 esp_tx  ; Espera a que finalice la transmision
        RTS

recibir LDAB dato_recibido        ; Subrutina que espera a que se reciba un dato para leerlo
        CMPB #$00
        BEQ recibir               ; Espera a que se reciba un dato
;espera  BRSET PORTD,X $10 espera ; Espera a que finalice una posible transmision
        LDAB #$00
        STAB dato_recibido        ; indica que se ha leido el dato
        LDAB PORTD,X              ; lee el dato y lo deja en el acum B
        ANDB #$0F
        RTS

delay   LDD TCNT,X                ; Subrutina que espera 32 ms
        ADDD #TIEMPO              ; Esperar 32 ms
        STD TOC4,X              ; Escribe en el comparador 4
        BSET TFLG1,X $10          ; Poner a cero flag del comparador 4
buclees BRCLR TFLG1,X $10 buclees  ; Comprobar si a terminado la cuenta
        RTS

unmask  TBA           ; Subrutina que separa el BIT de control del dato recibido
        ANDB #$07     ; dato en acum B
        ANDA #$08     ; mascara para el BIT de control en el acum A
        RTS

des4der LSRB          ; Subrutina que desplaza el dato en el acum B 4 bits a la derecha
        LSRB
        LSRB
        LSRB
        RTS

detener BCLR PORTA,X $18     ; Parar motores
        BCLR TMSK2,X $20     ; Desactiva interrupcion de overflow del acumulador
        BCLR PACTL,X $40     ; Desactiva el acumulador de pulsos
        RTS

;Rutina de atencion a la interrupcion externa IRQ

int_irq BRSET PORTD,X $10 fin_tr ; Si se estaba transmitiendo salta a finalizar la transmision
        LDAA #$01
        STAA dato_recibido       ; Si no se estaba transmitiendo activa dato_recibido
        RTI
fin_tr  BCLR DDRD,X $0F          ; Bits 0 a 3 del PUERTO D entradas
        BCLR PORTD,X $10         ; Pone la tarjeta en recepcion
        RTI

;Rutina de atencion a la interrupcion de overflow del acumulador de pulsos

int_ac  BSET TFLG2,X $20 ; Poner a cero flag de overflow del ac
        BSR detener  ; Para el Microbot
        RTI

          END
```

En las subrutinas '*enviar*' y '*recibir*' se observa como se implementan los ciclos de transmisión y recepción descritos en la parte de diseño Hardware de la tarjeta conectada al Microbot.

La descripción detallada de las subrutinas contenidas en este programa se puede encontrar en el anexo C de este proyecto.

CAPÍTULO 5. PRUEBAS

1 Introducción

Durante el desarrollo del sistema se han llevado acabo diversas pruebas hasta la finalización del diseño y puesta en funcionamiento del sistema. A continuación se describen las pruebas más significativas de las realizadas.

2 Pruebas relativas al Codificador y al Decodificador

Las primeras y más obvias pruebas realizadas con estos Chips han sido las de transmitir datos con el codificador y ver como los recibía el decodificador. Esto se ha realizado tanto conectando directamente la salida de datos del codificador a la entrada de datos del decodificador, como intercalando entre ellos el transmisor y el receptor radio, sirviendo además esto último para comprobar el alcance efectivo de los módulos de radiofrecuencia. Inicialmente se fijaban las direcciones y los datos a transmitir mediante switches, para como se verá más adelante, posteriormente hacerlo mediante SW utilizando el puerto paralelo del PC.

El esquema seguido en estas pruebas ha sido el que se indica en la figura 60.

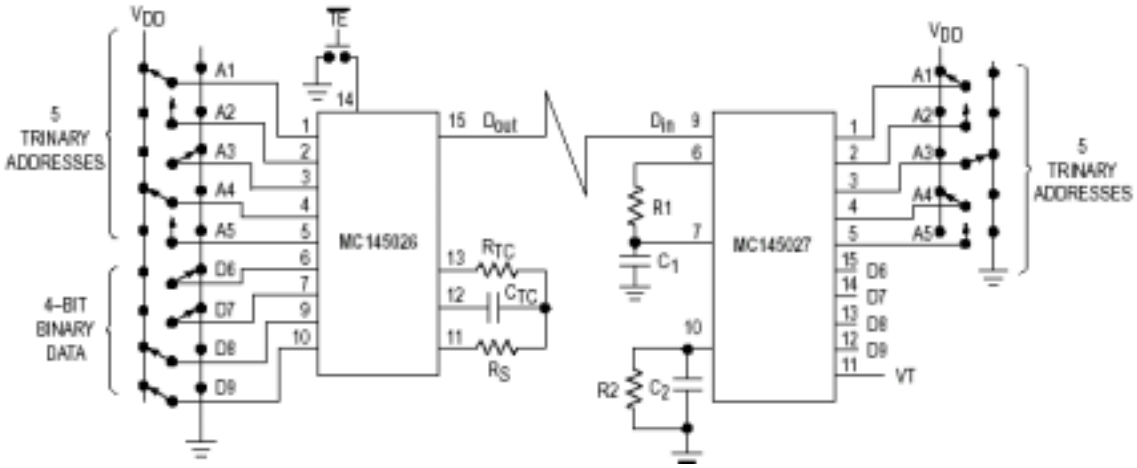


Fig. 60: Esquema de conexión para pruebas del MC145026 y MC145027.

Estas pruebas se han realizado para distintas velocidades de transmisión (fosc = 4 Khz, 2KHz, 1.65 KHz, 1KHz,...), comprobándose las limitaciones en ancho de banda del transmisor y receptor radio.

Además se han realizado al decodificador las pruebas recomendadas en el propio libro de características de los C.I.s MC145026 y MC145026 *[Moto1]*.

La primera de estas pruebas se basa en comprobar como para la señal en el condensador C1 (Pin 7), los puntos denominados como 'DOS' en la figura 61 (puntos en donde se determina si el dato en Din es un 0 o un 1) no se encuentran demasiado cercanos a los flancos de subida o bajada de la señal en el pin

Din. De estar estos puntos demasiado cercanos podrían darse intermitencias en la comunicación. Se ha comprobado en el montaje realizado como estos puntos quedan aproximadamente en la mitad del intervalo formado por los flancos en Din, no debiendo generarse por este motivo ningún problema en la comunicación.

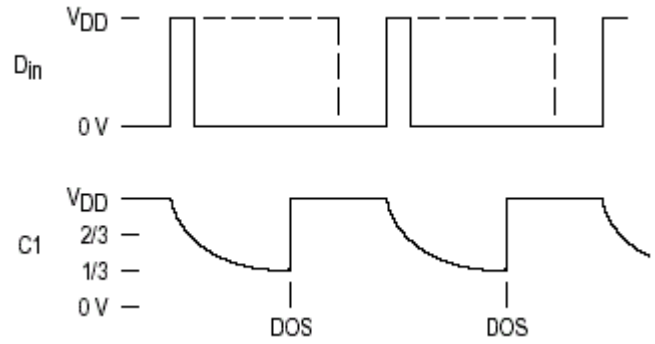


Fig. 61: Caída de tensión en el condensador C1 (pin 7).

La otra prueba recomendada es observar como, recibiendo el decodificador una transmisión continuada, la caida de tensión en el pin R2/C2 al final de la transmisión de cada dato se sitúa entre 2/3 y 1/3 de Vdd (ver figura 62). Esto también se ha comprobado en los circuitos montados, no detectándose ningún problema al encontrar la caída de tensión en R2/C2 aproximadamente en la mitad del rango permitido.

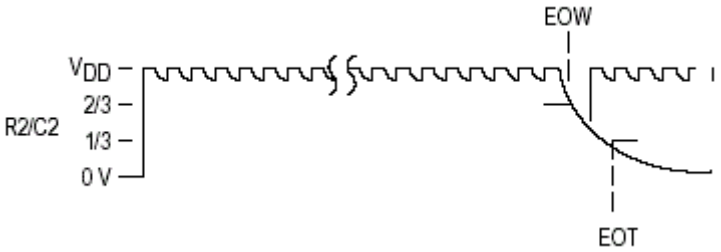


Fig. 62: Caída de tensión en R2/C2 (pin 10).

3 Pruebas con el Sistema Completo

Una vez montado el sistema completo, ya fuera sobre una placa de inserción o sobre las definitivas PCIs, además de probar el funcionamiento de los programas diseñados para el control de los Microbots (proyecto.exe y proyecto.s19), se han utilizado otros programas específicos para probar el funcionamiento del Hardware.

El más significativo de estos ha sido el programa cuyo código se encuentra en el fichero *prueba.c* y que se muestra a continuación. Este programa ha servido también para comprobar el alcance y la tasa de fallos del enlace radio.

```
//prueba.c
/* Programa para probar en funcionamiento del enlace radio bidireccional
diseñado para comunicar un PC con distintos Microbots */

/*--------------------- Librerias incluidas --------------------*/

#include "com.h"
#include <stdio.h>
#include <bios.h>
#include <dos.h>

/*--------------------- Funciones -----------------------------*/

void prueba_trans()
{       // Prueba para comprobar que el modem transmite bien todos los datos posibles
          unsigned short int i,a,error;
          unsigned int fallos,aciertos,total;
          double t1,t2,t3,t4,t;
          clrscr();
          fallos=0;
          aciertos=0;
          total=0;
          t3=clock();
          for (a = 0; a <= 15; a++) {  // tramsmite todos los datos posibles a todis las dirrecciones posibles
           for (i = 0; i <= 15; i++) {
                    error=FALSE;
                    t1=clock();
                    enviar(a,i);
                    while ( (transmitiendo==TRUE) && (error==FALSE) )  // Se espera a que ternmine la transmision
                    {
                              t2=clock();
                              t=(t2 - t1)/CLK_TCK;
                              if (t > TETX) {    // Se pone el modem en recepcion y se configura su direccion local
                                        transmitiendo=FALSE;
                                        dato_recibido=FALSE;
                                        outportb( DATA , (inportb(DATA) & 0x0F) | (dir_local<<4) );
                                        outportb(CONTROL,0x12);
                                        error=TRUE;   // No se detecto el final de la transmision
                              }
                    }
                    printf("dir:%d  dato:%2d   ",a,i);
                    if (error==TRUE) {   // No se ha detectado el fin de la transmision
                              printf("Tiempo sobrepasado.");
                              fallos++;
                    }
                    else if (recibir()==i) {   // La transmision se completo satisfactoriamente
                              printf("Prueba satisfactoria.");
                              aciertos++;
                    }
                    else {                    // No se transmitio el dato correctamente
                              printf("Prueba no satisfactoria.");
                              fallos++;
                    }
                    printf(" Duracion: %f milisegundos.\n",t*1000.0); // Se muestra la duracion de la transmision
           }
          }
          t4=(t2-t3)/CLK_TCK;
          total=fallos+aciertos;
          printf("%d Transmisiones en total.\n",total);      // Se muestra una estadistica de los resultados
          printf("%f Segundos en total.\n",t4);
          t3=(t4*1000.0)/total;
          printf("%f Milisegundos de media cada transmision.\n",t3);
          t3=1000/t3;
          printf("%f Transmisiones por segundo.\n",t3);
          printf("%1d fallos, %2d aciertos.\n",fallos,aciertos);
          printf("%1f por ciento de fallos, %2f de aciertos.\n",(fallos*100.0)/total,(100.0*aciertos)/total);
          printf("Pulse una tecla para continuar.");
          while (kbhit()==FALSE) {
          }
}

void prueba_recep()    // Prueba que muestran todos los datos que se reciben
{
          unsigned short int dato,dir;
          double t1,t2;
        char ch;
          ch='0';
          t1=0;
        t2=0;
          clrscr();
          printf("Introduzca la direccion local del PC:  ");
          scanf("%u",&dir);
          printf("\n");
          setDirLocal(dir);  // Se configura el modem con la direccion local indicada por el usuario
          outportb(CONTROL,0x12); // Se configura el modem en Recepcion
          printf("Prueba de recepcion. Pulsar ESC para salir \n");
          do {
                    while ((datoListo()==FALSE) & (ch !=27))
                    {
                              if (kbhit()){
                              ch = getch();
                              }
                    }
                    if (datoListo()==TRUE) {   // Se ha recibido un dato
                              t2=clock();
                              dato=recibir();    // Se lee el dato
                              printf("dato recibido: %d   ",dato);  // Se muestra el dato recibido
                              printf("tiempo desde la ultima recepcion: %f \n",(t2-t1)/CLK_TCK);
                              t1=t2;
                  }
          } while (ch !=27); // Se sale cuando se pulsa ESC (ASCII 27)
```

```
}

void prueba_eco()
{       // Para probar la bidireccionalidad del enlace. Para cada dato transmitido se espera un eco
          unsigned short int i,j,numit,dir,dato,error;
          unsigned int fallos,aciertos,total;
          double t1,t2,t3,t4;
          clrscr();
          fallos=0;
          aciertos=0;
          total=0;
          printf("Introduzca el numero de repeticiones de la prueba:  ");
          scanf("%u",&numit);
          printf("\n");
          printf("Introduzca la direccion local del PC:  ");
          scanf("%u",&dir);
          printf("\n");
          setDirLocal(dir); // Se configura el modem con la direccion local indicada por el usuario
          printf("Introduzca la direccion del robot:  "); // Se pide al usuario la direccion destino
          scanf("%u",&dir);
          printf("\n");
          t3=clock();
          for (j = 1; j <= numit; j++) { // Se envia ciclicamente del 0 al 15 esperando un eco para cada uno
          for (i = 0; i <= 15; i++) {
                    printf("Enviando: dir:%d  dato:%2d   ",dir,i);
                    enviarNibble(dir,i);    // Se envia el dato
                    error=FALSE;
                    t1=clock();
                    while ( (datoListo()==FALSE) && (error==FALSE) )   // Se espera a recibir el eco
                    {
                              t2=clock();
                              if ( ((t2-t1)/CLK_TCK) > TERX) {
                                        error=TRUE;    // No se recibio el eco
                              }
                    }
                    if (error==TRUE) {         // No se recibio el eco
                              printf("Tiempo para el eco sobrepasado.\n");
                              fallos++;
                    }
                    else {
                              dato=recibir();
                              printf("Eco recibido:%d  ",dato);
                              if (dato==i) {      // Se recibio un eco correcto
                                        printf("==> Eco satisfactorio.\n");
                                        aciertos++;
                              }
                              else {              // No se recibio un eco correcto
                                        printf("==> Eco no satisfactorio.\n");
                                        fallos++;
                              }
                    }
          }
          }
          t4=clock();
          t4=(t4-t3)/CLK_TCK;
          total=fallos+aciertos;
          printf("%d Transmisiones en total.\n",total);     // Se muestra una estadistica
          printf("%f Segundos en total.\n",t4);
          t3=(t4*1000.0)/total;
          printf("%f Milisegundos cada transmision y su eco.\n",t3);
          t3=1000/t3;
          printf("%f Transmisiones y ecos por segundo.\n",t3);
          printf("%1d fallos, %2d aciertos.\n",fallos,aciertos);
          printf("%1f por ciento de fallos, %2f de aciertos.\n",(fallos*100.0)/total,(100.0*aciertos)/total);
          printf("Pulse una tecla para continuar.");
          while (kbhit()==FALSE) {
          }
}

void main()
{
          char *elec;
          *elec='0';
          inicializar();          // Se inicializa el Modem
          while (*elec!='4') {
                    clrscr();       // Se muestra el menu
                    printf("1. Prueba de transmision. \n");
                    printf("2. Prueba de recepcion. \n");
                    printf("3. Prueba de eco. \n");
                    printf("4. Salir. \n");
                    printf("Elija un opcion: ");
                    scanf("%c",elec);
                    if (*elec=='1') {
                              prueba_trans();       // Prueba de transmision
                    }
                    else if (*elec=='2') {
                              prueba_recep();       // Prueba de recepcion
                    }
                    else if (*elec=='3') {
                              prueba_eco();         // Prueba de eco
                    }
          }
          terminar();
}
```

Al ejecutar este programa se presenta en pantalla el siguiente menú:

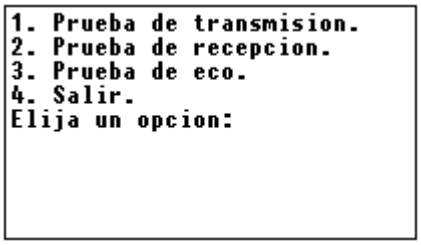


Fig. 63: Menú mostrado por pantalla.

Este menú da la posibilidad de realizar tres pruebas distintas en el PC:

- Prueba de transmisión: Se realiza desde el PC una transmisión para cada combinación posible de dirección y dato, mostrándose para cada una si ésta se ha producido satisfactoriamente. Al final de la prueba se muestra un computo de las transmisiones satisfactorias y las no satisfactorias, así como la duración media de cada transmisión y el número de transmisiones por segundo.
- Prueba de recepción: Se configura la tarjeta conectada al PC en recepción y se muestra cada dato recibido, así como el tiempo transcurrido desde la ultima recepción. Esta prueba se ha utilizado junto con el programa '*prue_tx.asm*' cargado en el Microbot, el cual transmitía cíclicamente todos los datos posibles hacia el PC para comprobar la recepción de éstos por parte de la tarjeta. Esta prueba ha servido para comprobar el alcance del enlace radio transmitiendo únicamente desde el Microbot hacia el PC.
- Prueba de eco: Actuando junto con el programa '*eco.asm*' cargado en el Microbot, en esta prueba se envían por orden hacia el robot seleccionado todos los datos posibles, esperando para cada uno de ellos un eco procedente del Microbot, para a continuación mostrar si se recibió o no el eco satisfactoriamente. Como en la prueba de transmisión al final se muestra una estadística de los fallos y los aciertos obtenidos. Esta prueba ha servido para comprobar el alcance del enlace radio transmitiendo datos en las dos direcciones alternativamente.

Con estas pruebas se ha conseguido comprobar el alcance efectivo del enlace radio, que resulta ser superior a una decena de metros en espacios cerrados para la prueba de recepción (con el Microbot transmitiendo datos permanentemente), y ligeramente inferior para la prueba de eco (como era previsible ya que intervienen en la comunicación ambos segmentos transmisores).

Como se ha visto, dos de estas pruebas requieren que en el Microbot se carguen unos programas que ejerzan la función complementaria a dichas pruebas. A continuación se muestra el código de estos programas (prue_tx.asm y eco.asm), que son cargados en la memoria RAM del 68HC11.

* prue_tx.asm: En este programa el Microbot transmite cíclicamente hacía el PC datos del 1 al 15.

```
;PRUE_TX.ASM
;Programa que transmite ciclicamente datos de 0 al 15 hacia el PC

OPTION  EQU $39
PIOC    EQU $02
PORTA   EQU $00
PORTD   EQU $08
DDRD    EQU $09
PORTE   EQU $0A
SCCR2   EQU $2D
SPCR    EQU $28
HPRIO   EQU $3C
TMSK1   EQU $22
TMSK2   EQU $24
TFLG1   EQU $23
TCTL1   EQU $20
DIRPC   EQU $00

        ORG $0000   ;RAM

        BRA config

;variables

dato_recibido  DB 0
dato           DB 0

config    LDX #$1000
        LDS #$00B4              ; Pila en DIR 180


        LDAA #$00
        STAA  SCCR2,X           ; SCI deshabilitado

        LDAA #$30
        STAA  SPCR,X            ; SPI deshabilitado y salidas OC

        LDAA  #$30              ; BITS 0 a 3 del puerto D entradas,
        STAA  DDRD,X            ; BITS 4 y 5 del puerto D salidas


        LDAA  #$20
        STAA  PORTD,X           ; BIT 4 a 0 y BIT 5 del puerto D a 1

        BSET OPTION,X $20       ; IRQ activa por flanco de bajada

        LDAA #$C6
        STAA HPRIO,X            ;Maxima prioridad IRQ

        LDAA #$00
        STAA TMSK1,X            ; No permitir la interupci¢n de los comparadores

        LDAA #$00
        STAA dato_recibido

        LDY #$FFFF         ; Realizar una pausa
dec     DEY
        CPY #0
        BNE dec

        CLI                ; permite interrupciones

        LDAA #DIRPC
inicio  LDAB #00
        STAB dato
bucle   BSR enviar         ; envia el dato
        LDAA #DIRPC
        INC dato
        LDAB dato
        CMPB #16
        BEQ inicio
        BRA bucle

;Funcion enviar: Transmite el dato en B a la direccion en A

enviar  BRSET PORTD,X $10 enviar ;espera a que se termine la anterior transmision

        BSET DDRD,X $0F          ;BITS 0 a 3 del puerto D salidas

        ANDA #$0F
        ORAA #$10
        STAA PORTD,X             ;Escribe la direccion destino e inicia la transmision

        BSET PORTD,X $20
        BSET PORTD,X $20
        BSET PORTD,X $20
        BSET PORTD,X $20         ;Se repite para que transcurra thl

        ORAB #$30
        STAB PORTD,X             ;Escribe el dato a transmitir

        ANDA #$0F

          RTS

;Rutina de atencion a la interrupcion externa IRQ
```

```
int_irq BRSET PORTD,X $10 fin_tr  ;Si se estaba transmitiendo salta a finalizar la transmision
        LDAA #$01
        STAA dato_recibido        ;Si no se estaba transmitiendo activa dato_recibido
        RTI
fin_tr  BCLR DDRD,X $0F           ;BITS 0 a 3 del puerto D entradas
        BCLR PORTD,X $10          ;Pone la tarjeta en recepcion
        RTI

        ORG $00EE
        JMP int_irq

          END
```

* eco.asm: En este programa para cada dato recibido el Microbot responde con la transmisión de otro dato igual al que le es transmitido desde el PC.

```
;ECO.ASM
;Programa que hace ECO del dato recibido

OPTION  EQU $39
PIOC    EQU $02
PORTA   EQU $00
PORTD   EQU $08
DDRD    EQU $09
PORTE   EQU $0A
SCCR2   EQU $2D
SPCR    EQU $28
HPRIO   EQU $3C
TMSK1   EQU $22
TMSK2   EQU $24
TFLG1   EQU $23
TCTL1   EQU $20
DIRPC   EQU $00
TOC4    EQU $1C
TCNT    EQU $0E
TIEMPO  EQU 65000  ; Numero de pulsos de reloj necesarios para generar un
                   ; retraso 32,5 ms. Cada tic de reloj son 0.5 us
                   ; 65000*8 = 32500 us = 32,5 ms.

        ORG $0000   ; RAM

        BRA config

;variables

dato_recibido  DB 0
dato           DB 0
dir            DB 0
cuenta         DB 0

config  LDX #$1000
        LDS #$00B4              ; PILA en DIR 180

        LDAA #$00
        STAA  SCCR2,X           ; SCI deshabilitado

        LDAA #$30
        STAA  SPCR,X            ; SPI deshabilitado y salidas OC

        LDAA  #$30              ; BITS 0 a 3 del puerto D entradas,
        STAA  DDRD,X            ; BITS 4 y 5 del puerto D salidas

        LDAA  #$20
        STAA  PORTD,X           ; BIT 4 a 0 y BIT 5 del puerto D a 1

        BSET OPTION,X $20       ; IRQ activa por flanco de bajada

        LDAA #$C6
        STAA HPRIO,X            ;Maxima prioridad IRQ

        LDAA #$00
        STAA TMSK1,X            ; No permitir la interupci¢n de los comparadores

        LDAA #$00
        STAA dato_recibido

        CLI                     ; permite interrupciones

        LDAA #DIRPC
bucle   LDAB dato_recibido      ;espera a recibir un dato
        CMPB #$00
        BEQ bucle
        BSR recibir             ;lee el dato recibido
        BSR enviar              ;envia el dato recibido
        BRA bucle

;Funcion enviar: Transmite el dato en B a la direccion en A

enviar  STAA dir
        STAB dato
ex      BRSET PORTD,X $10 ex ;espera a que se termine la anterior transmision

        BSR delay            ;Espera 32,5 ms

        BSET DDRD,X $0F      ;BITS 0 a 3 del puerto D salidas
```

```
        LDAA dir
        ANDA #$0F
        ORAA #$10
        STAA PORTD,X         ;Escribe la direccion destino e inicia la transmision

        BSET PORTD,X $20
        BSET PORTD,X $20
        BSET PORTD,X $20
        BSET PORTD,X $20     ;Se repite para que transcurra thl

        LDAB dato
        ORAB #$30
        STAB PORTD,X         ;Escribe el dato a transmitir

        ANDA #$0F

        RTS

;Funcion recibir: lee de la tarjeta el dato recibido

recibir BRSET PORTD,X $10 recibir  ;espera a que no se este transmitiendo
        LDAB #$00
        STAB dato_recibido         ;indica que se ha leido el dato
        ;BCLR DDRD,X $0F
        LDAB PORTD,X               ;lee el dato y lo deja en el acum B
        ANDB #$0F
        RTS

;Funcion delay: genera un retardo de 32,5 ms

delay   LDD TCNT,X                 ;Subrutina que espera 32,5 ms
        ADDD #TIEMPO               ;Esperar 32,5 ms
        STD TOC4,X                 ;Escribe en el comparador
        BSET TFLG1,X $10           ;Pone a cero flag del comparador 4
buclees BRCLR TFLG1,X $10 buclees  ;Comprobar si a terminado la cuenta
        RTS

;Rutina de atencion a la interrupcion externa IRQ

int_irq BRSET PORTD,X $10 fin_tr  ;Si se estaba transmitiendo salta a finalizar la transmision
        LDAA #$01
        STAA dato_recibido        ;Si no se estaba transmitiendo activa dato_recibido
        RTI
fin_tr  BCLR DDRD,X $0F           ;BITS 0 a 3 del puerto D entradas
        BCLR PORTD,X $10          ;Pone la tarjeta en recepcion
        RTI

        ORG $00EE
        JMP int_irq

        END
```

Tras repetidas pruebas y una intensa tarea de depuración se ha comprobado, con la ejecución de este programa, cómo para que el PC reciba bien los datos transmitidos por el Microbot, es necesario esperar un tiempo después de una recepción para comenzar la transmisión del eco. Esto se hace para dejar que el sistema se recupere de las interferencias que se inducen entre el transmisor y el receptor de una misma tarjeta. Dicho retraso puede observarse incluido en la subrutina '*enviar'* en el código anterior, así como también en la misma subrutina del programa *proyecto.asm*.

Otros programas utilizados para probar el sistema han sido *simple.c* y *simple2.c*. En estos programas se utiliza en enlace radio en la dirección PC → Microbot, funcionando el PC simplemente como un mando a distancia, pudiéndose ordenar al robot que se desplace hacia delante o hacía atrás, que gire a derecha o izquierda o que se pare (el Microbot continuará ejecutando el ultimo comando recibido hasta que no reciba uno nuevo). Ambos programas utilizan el mismo panel de control diseñado para el programa *proyecto.exe* y actúan enviándole constantemente al Microbot comandos de movimiento, variando únicamente la forma de hacerlo en cada uno de ellos.

En *simple.c* se deja permanentemente activa la señal /TE (*Transmit Enable*) del codificador, con lo que éste está transmitiendo permanentemente el dato presente en sus entradas de dato. En *simple2.c* el programa espera a que termine la transmisión de un dato para comenzar la transmisión del siguiente. A continuación se muestra el código de ambos programas:

* simple.c:

```
//simple.c
/* Programa para mandar comandos de movimiento al microbot
con interfaz de panel para el usuario */

/*---------------------- Librerias incluidas -------------------------*/

#include <dos.h>
#include "userint.h"
#include "panel.h"

/*---------------------- Definicion de constantes --------------------*/

#define TRUE 1
#define FALSE 0
#define DELANTE 0x01
#define ATRAS 0x02
#define IZQUIERDA 0x04
#define DERECHA 0x08
#define PARO 0x0F
#define DATA 0x378
#define STATUS 0X379
#define CONTROL 0X37A

/*----------------------Funciones----------------------------*/

void enviar(unsigned short int dir_robot,int comando);

/*----------------------Programa principal--------------------*/

main()
{
    int p,ctrl;
    unsigned short int robot,temp;
    outportb(CONTROL,0x00);        // configura el Modem en transmision con la interrupcion del PP deshabilitada
    p=LoadPanel("panel.uir",MAIN); // Carga el panel
    DisplayPanel(p);               // Visualiza el panel
    while(TRUE){
          GetUserEvent(1,&p,&ctrl);  // Espera un evento del panel
          GetCtrlVal(p, MAIN_ROBOT, &robot);
          switch(ctrl){
              case MAIN_STOP:enviar(robot,PARO);
                    break;
              case MAIN_DELANTE:enviar(robot,DELANTE);
                    break;
              case MAIN_ATRAS:enviar(robot,ATRAS);
                    break;
              case MAIN_IZQUIERDA:enviar(robot,IZQUIERDA);
                    break;
              case MAIN_DERECHA:enviar(robot,DERECHA);
                    break;
              case MAIN_SALIR:outportb(CONTROL,0x00);
                    return;
                    break;
              case MAIN_PASOS:     // Se ha pulsado el boton 'Pasos restantes'
                    GetCtrlVal(p, MAIN_NUMPASOS, &temp);
                    enviar(robot,temp);  // Envia el dato que hubiera en la casilla 'Numero de pasos'
                    break;
              }
     }
}
/*--------------------------Funcion enviar-----------------------*/
void enviar(unsigned short int dir_robot,int comando)
{
     outportb(DATA,(dir_robot<<4) | (comando & 0x0F)); // Escribe el comando en el modem junto con la dir destino
     outportb(CONTROL,0x01);   // Activa la transmision
}
```

* simple2.c:

```
//simple2.c
/* Programa para mandar comandos de movimiento al microbot
con interfaz de panel para el usuario */

/*---------------------- Librerias incluidas -------------------------*/
#include "userint.h"
#include "com.h"
#include "panel.h"

/*---------------------- Definicion de constantes --------------------*/

#define TRUE 1
#define FALSE 0
#define DELANTE 0x01
#define ATRAS 0x02
#define IZQUIERDA 0x04
#define DERECHA 0x08
#define PARO 0x0F
#define DATA 0x378
#define STATUS 0X379
#define CONTROL 0X37A
```

```
/*----------------------Programa principal--------------------*/

main() {
    int p,ctrl;
    unsigned short int robot,comando;
    inicializar();
    p=LoadPanel("panel.uir",MAIN);    // Carga el panel
    DisplayPanel(p);                  // Visualiza el panel
    comando=PARO;
    while(TRUE) {
          if (GetUserEvent(0, &p, &ctrl) != 0) {
            GetCtrlVal(p, MAIN_ROBOT, &robot);
            switch(ctrl){
              case MAIN_STOP:comando=PARO;
                     break;
              case MAIN_DELANTE:comando=DELANTE;
                     break;
              case MAIN_ATRAS:comando=ATRAS;
                     break;
              case MAIN_IZQUIERDA:comando=IZQUIERDA;
                     break;
              case MAIN_DERECHA:comando=DERECHA;
                     break;
              case MAIN_SALIR:terminar();
                     UnloadPanel (p);
                     return;
                     break;
              case MAIN_PASOS:      // Se ha pulsado el boton 'Pasos restantes'
                     GetCtrlVal(p, MAIN_NUMPASOS, &comando); // Envia el dato que hubiera en la casilla 'Numero de
pasos'
                     break;
              }
          }
          enviarNibble(robot,comando);   // Envia el comando al robot
     }
}
```

Estos programas necesitan de un programa complementario ejecutándose en el Microbot. El código de este programa se encuentra en el archivo *simple.asm*, el cual se muestra listado a continuación.

```
SIMPLE.ASM
;programa que mueve el robot segun el comando que este recibiendo del PC

        ORG $0
PORTA   equ $0
PORTD   equ $8
DDRD    EQU $09
SCCR2   EQU $2D
SPCR    EQU $28

        LDX #$1000
        LDS #$00B4          ; PILA en DIR 180

        SEI                 ; interrupciones deshabilitadas

        LDAA #$00
        STAA  SCCR2,X       ; SCI deshabilitado

        LDAA #$30
        STAA  SPCR,X        ; SPI deshabilitado y salidas OC
        STAA  DDRD,X        ; BITS 0 a 3 del puerto D entradas, BITS 4 y 5 salidas

        LDAA  #$20
        STAA  PORTD,X       ; BIT 4 a 0 y BIT 5 del puerto D a 1 (recepcion)

        LDAA #$00

inicio  STAA PORTA,X        ;contenido del acum A al puerto A

comando BRSET PORTD,X $0F para
        BRSET PORTD,X $01 delante
        BRSET PORTD,X $02 detras
        BRSET PORTD,X $04 izquier
        BRSET PORTD,X $08 derecha
        BRA inicio

delante LDAA #$18   ;avanza
        BRA inicio

detras  LDAA #$78   ;retrocede
        BRA inicio

izquier LDAA #$38   ;gira a a la izquierda
        BRA inicio

derecha LDAA #$58   ;gira a a la izquierda
        BRA inicio

para    LDAA #$00   ;se para
        BRA inicio

        END
```

Se puede observar como en este programa lo único que hace el Microbot es leer constantemente el comando recibido para activar los motores según el movimiento que indique dicho comando.

Estos programas (*simple.c, simple2.c y simple.asm*) han servido para comprobar como el enlace radio transmitiendo únicamente desde el PC hacía el Microbot posee un alcance superior a 10 metros en espacios cerrados y con multitud de obstáculos, lo cual era el alcance previsto para el enlace radio.

4 Resultado de las pruebas realizadas

Como resultado de las pruebas realizadas a los prototipos, donde se incluyen las descritas anteriormente, se desprenden los siguientes resultados:

- Frecuencia máxima de una onda cuadrada a la salida del receptor de radiofrecuencia de entorno a un 1 KHz (en el peor de los receptores probados), en cualquier caso inferior a la indicada por el fabricante (2 KHz). Pese a esto se ha podido utilizar una frecuencia del oscilador del codificador y del decodificador superior (1,65 KHz), debido a que como se indica en las pruebas realizadas al decodificador, aunque los pulsos más cortos en la señal transmitidas son ensanchados por el receptor, éstos permanecen dentro de los márgenes permitidos.
- Velocidad de transmisión del enlace de aproximadamente 7 Nibbles/s, o lo que es lo mismo 28 Bits/s.
- Alcance del enlace radio funcionando únicamente en una sola dirección (PC → Microbot o Microbot → PC) y transmitiendo datos continuamente superior a 10 metros en espacios cerrados.
- Alcance máximo del enlace utilizando la bidireccionalidad, de tal forma que para cada dato transmitido por el PC el Microbot responde con la transmisión de otro dato, de aproximadamente 10 metros en espacios cerrados (prueba de eco en *prueba.exe* y ejecución de *proyecto.exe*).

Se ha comprobado como, cuando el Microbot está en movimiento, la comunicación se hace más difícil debido al ruido que introducen los motores en el sistema Microbot. Este efecto se puede paliar utilizando una fuente de alimentación independiente para los motores de continua, tal y como permite la tarjeta CT293+ (ver capítulo 2).

Hay que decir que el alcance del enlace radio podría aumentarse mejorando la eficiencia en potencia transmitida por el emisor de RF. Puesto que para dicho transmisor se recomienda utilizar una antena de 50 Ω de impedancia característica (al ser su impedancia de salida de 50 Ω), de utilizar ésta se produciría la máxima transferencia de potencia a la antena con lo que la potencia radiada sería máxima. Esto podría realizarse utilizando una antena comercial con una impedancia característica de 50 Ω o bien utilizando para cada antena una red adaptadora de impedancia (red pasiva LC) para conseguir que el transmisor 'vea' una antena de dicha impedancia. Por supuesto también podría aumentarse el alcance utilizando transmisores más potentes o usando amplificadores (Boosters) a la salida de éstos.

Otro aspecto a destacar es que al utilizar una antena para recepción y otra para transmisión, separadas por escasos centímetros, más que una antena individual para cada caso se tiene un array de antenas, lo que puede hacer variar la direccionalidad de la transmisión y de la recepción. Este hecho hace

que sea más recomendable el uso de una misma antena para transmisión y recepción, junto con el correspondiente conmutador de antena.

Asimismo, se podría aumentar fácilmente la velocidad de transmisión (puesto que el codificador y el decodificador lo permiten) sustituyendo los módulos transmisor y receptor por otros con mayor ancho de banda.

Estas mejoras no se han llevado a cabo en el transcurso de este proyecto por considerarse completado el objetivo del mismo, el cual era la implementación y adaptación a ambas plataformas (PC y Microbot) del enlace radio, pese a conocer de antemano las pobres características del los módulos Transmisor y Receptor radio que se iban a utilizar. Se deja por tanto para proyectos posteriores la mejora de dicho enlace en cuanto a alcance y ancho de banda.

Se ha comprobado durante las pruebas del sistema el correcto funcionamiento del protocolo de comunicación diseño. Verificándose que no se producen interpretaciones erróneas de las comunicaciones, ni perdidas de control de los Microbots mientras se pudiera establecer una comunicación entre el PC y estos, garantizándose además que la información que viaja en una y otra dirección es correcta.

Asimismo, de las medidas de consumo de corriente realizadas a los prototipos se desprenden los siguientes valores:

- Consumo máximo de la tarjeta conectada al PC en recepción inferior a 23 mA.
- Consumo máximo de la tarjeta conectada al PC en transmisión inferior a 26 mA.
- Consumo máximo de la tarjeta conectada al Microbot en recepción inferior a 28 mA.
- Consumo máximo de la tarjeta conectada al Microbot en transmisión inferior a 30 mA.

CAPÍTULO 6. CONCLUSIONES Y LÍNEAS FUTURAS

1 CONCLUSIONES

Como conclusión, se puede decir que se han conseguido culminar con éxito las especificaciones requeridas para el sistema al comienzo del proyecto. Asimismo, se ha logrado crear un sistema físico perfectamente acondicionado para ser utilizado, que cumple con los objetivos requeridos a priori en cuanto a prestaciones (alcance, autonomía, fiabilidad, etc), diseño físico (sistema compacto y robusto) y adaptación y correcto funcionamiento sobre las plataformas a las que está destinado (PC y Microbot).

Pese a ser un proyecto laborioso por incluirse en él tanto diseño hardware como software, el esfuerzo ha sido gratificante ya que en la realización del mismo se han aplicado gran variedad de conocimientos teóricos y prácticos previamente adquiridos en el período de formación académica. A su vez se han ampliado estos conocimientos en las siguientes áreas:

Diseño electrónico: Diseño físico de los distintos bloques funcionales (codificadores, transmisores, etc.), así como la interacción entre ellos y con sistemas ya implementados.

Diseño de Placas de Circuito Impreso: Se ha realizado el diseño e implementación física de las PCIs utilizadas para montar el sistema, lo que ha permitido ahondar en la utilización de herramientas SW para el diseño de éstas, así como en el conocimiento de las técnicas necesarias para su fabricación.

Comunicación inalámbrica: Se han estudiado los diferentes sistemas de transmisión inalámbricos (infrarrojos, AM, FM, etc.), así como también se ha realizado una introducción en el diseño de protocolos de comunicación.

Microbótica: Se ha hecho un estudio exhaustivo de los *Microbots* (familias, hardware, software, etc.), así como de la programación en ensamblador del microcontrolador 68HC11 de *Motorola.*

Programación en Lenguaje C: Se han asentado los conocimientos para la programación en C, utilizando dos programas distintos, TurboC y Labwindows. También se han adquirido conocimientos relativos a la utilización del puerto paralelo del PC en sus distintos modos de funcionamiento.

2 LÍNEAS FUTURAS

Una de las metas del presente proyecto, como se indicó en el apartado de objetivos era que el sistema fuese modular y abierto, y efectivamente se ha conseguido, debido a que con una simple mejora en alguno de los componentes se conseguiría mejorar las prestaciones del sistema. Se pueden proponer las siguientes líneas futuras:

a) **Aumentar la velocidad de transmisión del enlace radio.** Sería, la mejora más sencilla e inmediata a realizar en el sistema diseñado, pues simplemente cambiando los módulos transmisor-receptor por otros con mayor ancho de banda, se podría aumentar mucho la velocidad de transmisión. Pudiéndose para esto, utilizar alguno de los módulos de radiofrecuencia considerados como alternativas de diseño en el capítulo 3.

b) **Reprogramación de los Microbots utilizando el enlace radio.** Conseguir poder enviar un programa al Microbot para que éste lo ejecute en modo autónomo en vez de limitarse únicamente a obedecer las instrucciones recibidas desde el PC (cuando se disponga de memoria suficiente en el Microbot).

c) **Comunicación *full-duplex* entre el PC y los *Microbot*.** Con este tipo de comunicación el PC transmitiría en una frecuencia y el robot en otra, esto es, se utilizarían dos canales distintos, uno para la comunicación en cada sentido, pudiéndose estas producirse simultáneamente.

d) **Granjas de *Microbots***. Crear una granja de Microbots donde estos se comuniquen con el ordenador central y entre ellos utilizando el enlace radio diseñado. Granja donde estos Microbots cooperen en la realización conjunta de tareas.

PRESUPUESTO

A continuación se detallan los materiales utilizados en la realización del prototipo, indicando el precio unitario de cada componente y el coste final del sistema:

DESCRIPCIÓN	Cantidad	Precio U.	TOTAL
Placa de circuito impreso fotosensible de dos caras	2	1000	2000
C.I. 145026	2	250	500
C.I. 145027	2	250	500
C.I.s varios	6	150	900
Conector DB25 macho para PCI	1	250	250
Zocalos para C.I.	7	25	175
Tiras de pines para jumpers y conectores	2	200	400
Resistencias varias de ¼ de W.	13	5	65
Arrays de resistencias	3	25	75
Condensadores varios	7	20	140
Módulo transmisor radio	2	2490	4980
Módulo receptor radio	2	1400	2800
Tornillos, diodos, switches, cables, etc.		500	500
Cable extensión del puerto paralelo	1	500	500
		TOTAL	13.785 ptas.

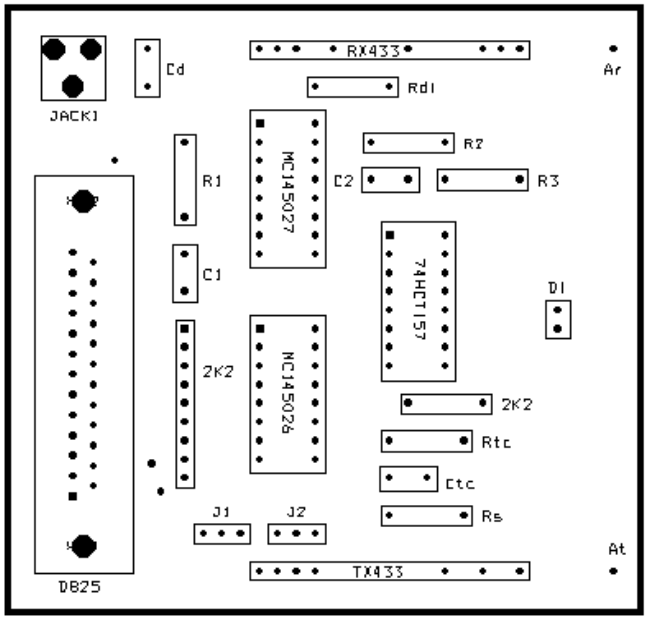
Este presupuesto corresponde al prototipo realizado. El coste de producción unitario de cada componente, para una posible producción en serie del sistema, sería sensiblemente más barato al adquirirse en mayores cantidades.

ANEXO A: Diseño de las Placas de Circuito Impreso

El diseño de la Placas de circuito impreso (PCI) se ha realizado utilizando la herramienta *Layout Plus*, incluida dentro del paquete *Orcad 9.0*. Este diseño se ha realizado con componentes insertados y con pistas en las dos caras de la placa, tanto para la PCI conectada al PC, como para la conectada al Microbot. A continuación se muestra el rutado de las pistas para ambas placas.

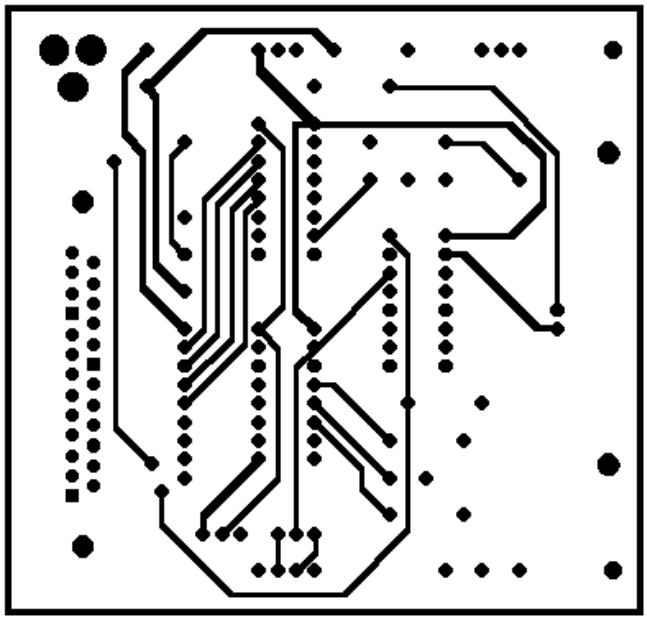
- La PCI conectada al PC se ha realizado con unas dimensiones de 82 x 85 mm y cuenta con un conector DB25 (25 pines) macho para su conexión con el Puerto Paralelo del PC (a través de un cable con conectores DB25 macho-hembra).

- Situación de los componentes y de los taladros:

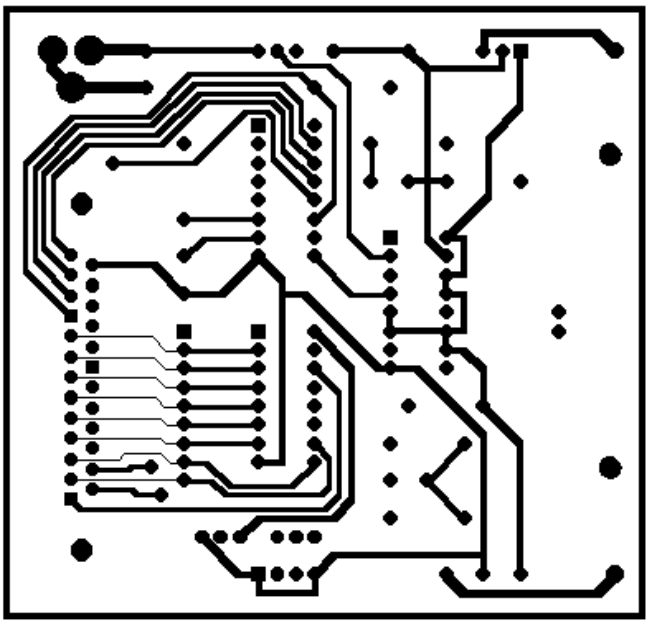


La resistencia R2 que se puede ver en los diagramas eléctricos en el capítulo 4, equivale en los diseños de las PCIs a las resistencias R2 y R3 conectadas en serie. Esto se ha hecho así debido a la imposibilidad de encontrar una resistencia del valor requerido, con lo que se ha optado por conseguir este valor colocando dos resistencias en serie.

- Rutado de las pistas en la cara superior:

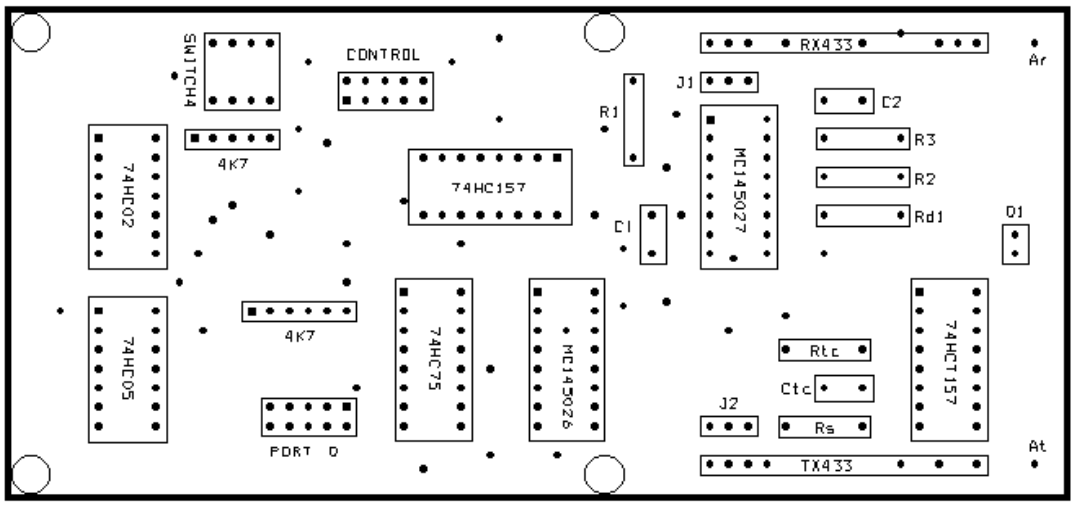


- Rutado de las pistas en la cara inferior:

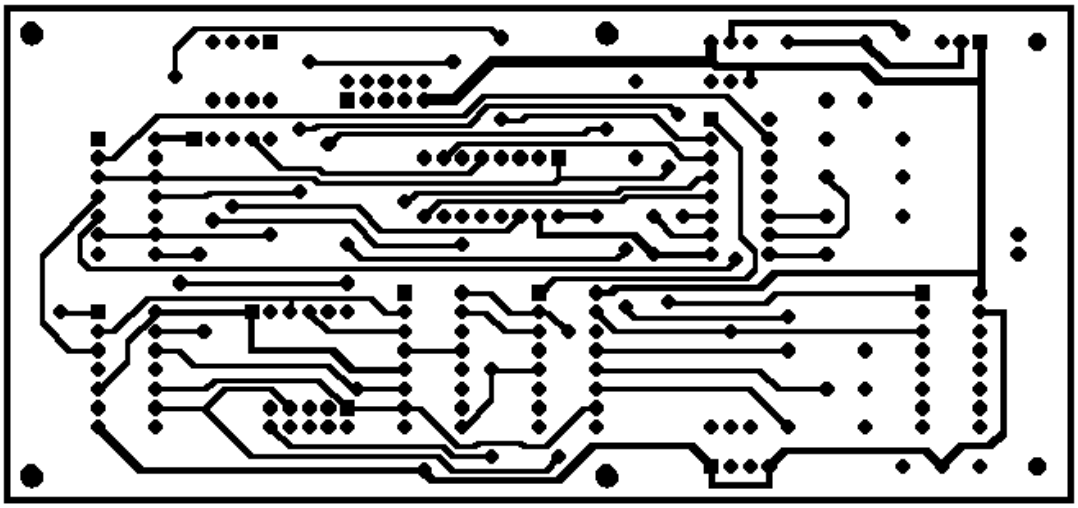


- La PCI conectada al Microbot se ha realizado con unas dimensiones de 65 x 142 mm, para adaptarse perfectamente a la estructura del Microbot Clónico, de ahí los orificios para los tornillos separadores que la sujetarán en la estructura de crecimiento hacia arriba.

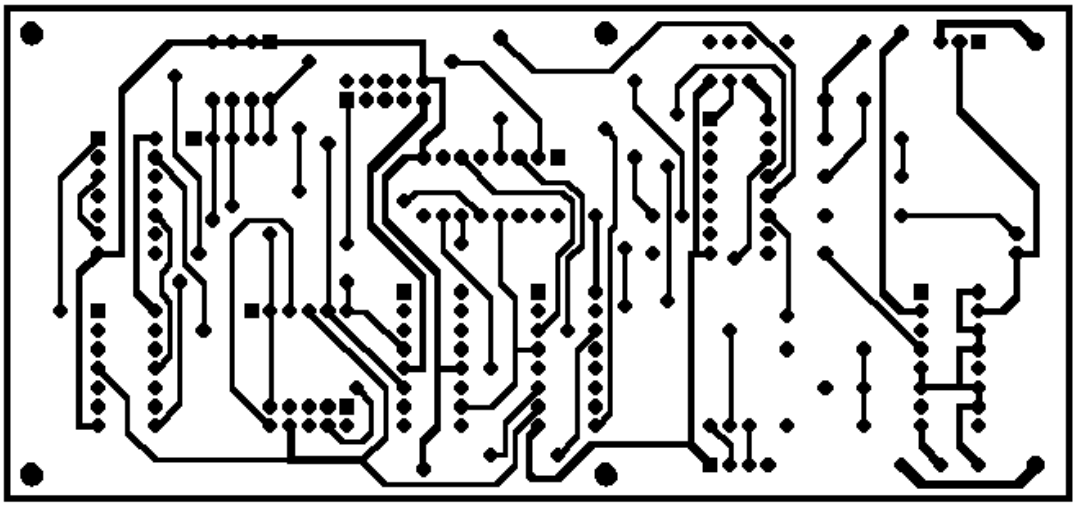
- Situación de los componentes y de los taladros:



- Rutado de las pistas en la cara superior:

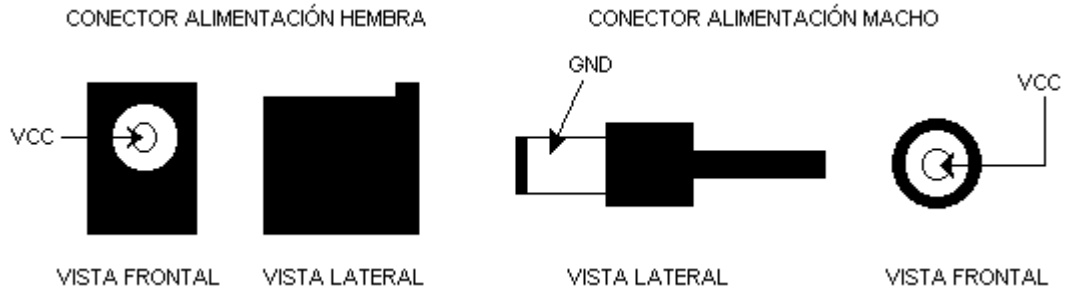


- Rutado de las pistas en la cara inferior:



Los diseños se pueden encontrar incluidos en este proyecto con los nombres **MODEM.MAX**[1] (PCI conectada al PC) y **ROBOT.MAX** (PCI conectada al Microbot), en formato del programa *Layout Plus* del *Orcad 9.0*.

A continuación se encuentra representado el conector hembra de alimentación utilizado en la placa a conectar al PC, junto con el jack macho cilíndrico con el que se debe conectar a la fuente de alimentación. Es importante hacer notar que **la parte exterior del jack macho debe ser GND**, mientras que **la parte interior debe ser VCC**.

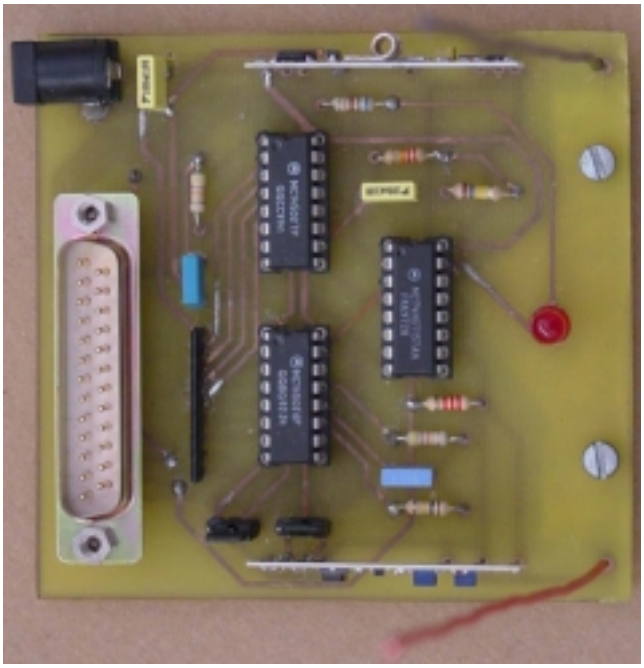


[1] *Nota*: En la realización del prototipo de la PCI conectada al PC, para el dibujo de las pistas en la cara superior, se ha utilizado el rutado correspondiente a la cara inferior del diseño en el fichero MODEM.MAX, utilizándose el de la cara superior para el dibujo de las pistas en la cara inferior del prototipo. Esto se ha hecho por considerarse más conveniente por la situación y tipo de componentes a insertar.

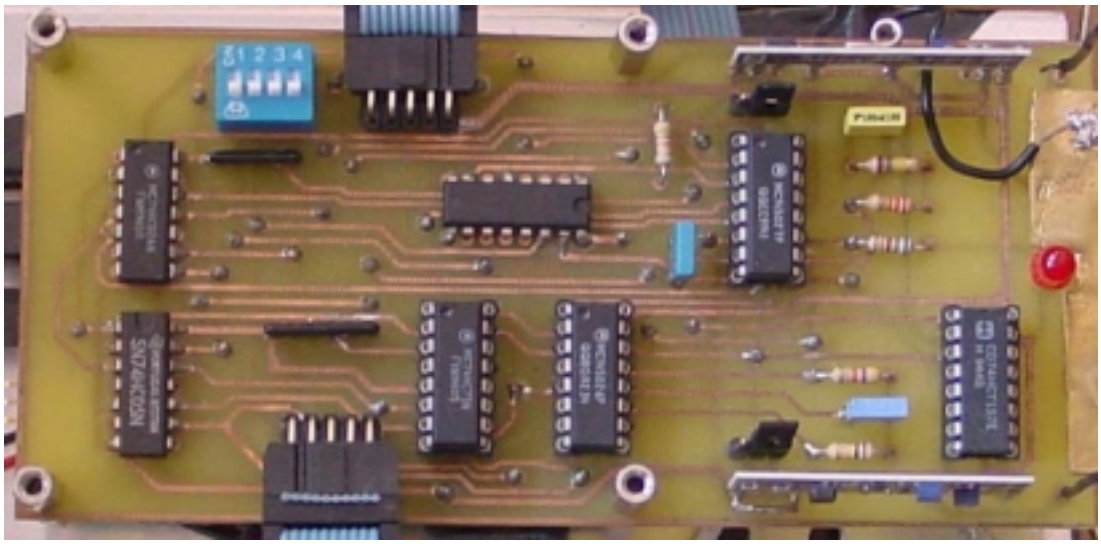
Para la PCI conectada al Microbot, las caras superior e inferior si corresponden con las del diseño en el fichero ROBOT.MAX.

El aspecto exterior de las placas realizadas para los prototipos es el siguiente:

- PCI conectada al PC:



- PCI conectada al Microbot:



Para realizar el montaje de los C.I.s más significativos o de más valor se han utilizado zócalos en vez de soldar los Chips directamente a la placa. Asimismo también se han utilizado zócalos para el montaje de las resistencias que sirven para configurar la velocidad de transmisión en el codificador y en el decodificador. Se puede también observar como se han sujetado en la parte frontal de las placas, junto a las antenas, unas laminas de cobre conectadas a masa, esto se ha hecho para aumentar el plano de masa de éstas y mejorar así sus características.

ANEXO B. Construcción y montaje de los encoders

1 Introducción

En el presente documento se describe como construir encoders para el Microbot objeto de este proyecto, es decir el Microbot Clónico, basado en la placa CT6811 de Microbótica.

Los usos de este tipo de dispositivos son diversos: hacer que el robot avance una serie de pasos y no un tiempo determinado (simulando el funcionamiento de un motor paso a paso), controlar la velocidad del robot, detectar posibles situaciones en las que el robot quede detenido ante un obstáculo, etc. En cualquier caso, siempre se debe tener en cuenta que en robótica móvil se trabaja con una gran incertidumbre introducida por muchos factores (rugosidad de la superficie, peso del robot, características de los sensores, incluso errores en las lecturas) que pueden hacer que se pierda la cuenta de los pasos. Por tanto, este mecanismo no es demasiado apropiado para definir movimientos absolutos del robot en el entorno, ya que estos errores implicarían necesariamente imprecisiones en la estimación de su posición.

En el proyecto al que acompaña este documento se utilizarán con el objetivo de definir movimientos relativos, esto es, contando vueltas de rueda en movimientos a ejecutar desde una posición previamente conocida, ya sean movimientos de avance y retroceso, como giros a derecha e izquierda.

En este anexo se trata acerca de cómo construir y montar encoders utilizando las ruedas de los Microbots, sin embargo es de destacar que para Microbots con orugas, como el utilizado en este proyecto (Microbot Clónico) también es posible construir encoders pintando las orugas con franjas blancas y negras y utilizando por lo demás los mismos sensores de infrarrojos y los mismos recursos de programación para su control.

2 Los sensores

Para la construcción de los encoders vamos a utilizar sensores de infrarrojos CNY70 y discos con franjas blancas y negras alternas. Colocaremos el par sensor-disco en una de las ruedas del robot. De esta forma dividiremos (con más o menos exactitud dependiendo del número de franjas) una vuelta de rueda en una serie de intervalos o pasos.

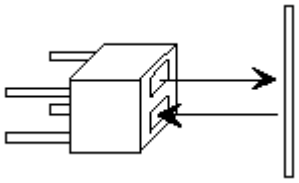


Figura 1: Sensor de infrarrojos CNY70

Se han seleccionado los sensores de infrarrojos CNY70 porque estos sensores son ideales en cuanto a relación calidad-precio para detectar cambios de blanco a negro en una distancia de milímetros. Como para el encoder utilizaremos un disco con franjas de estos colores, este tipo de sensores resultan

apropiados. En la Fig. 2 se muestra la salida del sensor según se encuentre enfrentado a una franja blanca o negra.

La **interpretación** del bit de un sensor es la siguiente:

bit a **1** significa detectado **Negro**
bit a **0** significa detectado **Blanco (No negro)**

Figura 2: Salida de los sensores de infrarrojos

Así, contando los pulsos generados a la salida del sensor, será posible contar las vueltas que da la rueda.

Por otro lado, la tarjeta CT293+ de Microbótica puede controlar directamente cuatro de estos dispositivos. Como inicialmente los robots vienen equipados con dos sensores para el seguimiento de líneas, quedan otras dos entradas libres para controlar otros dos sensores que se pueden utilizar para la realización de encoders.

Para montare estos sensores basta con conectar el emisor, el cátodo y conjuntamente el ánodo y el colector con las tres patas de uno de los conectores acodados que quede libre en la tarjeta. En la figura 3, se muestra como realizar dichas conexiones. Para una información más detallada se remite al lector al manual de usuario de la tarjeta CT293+ *[Micro1]*.

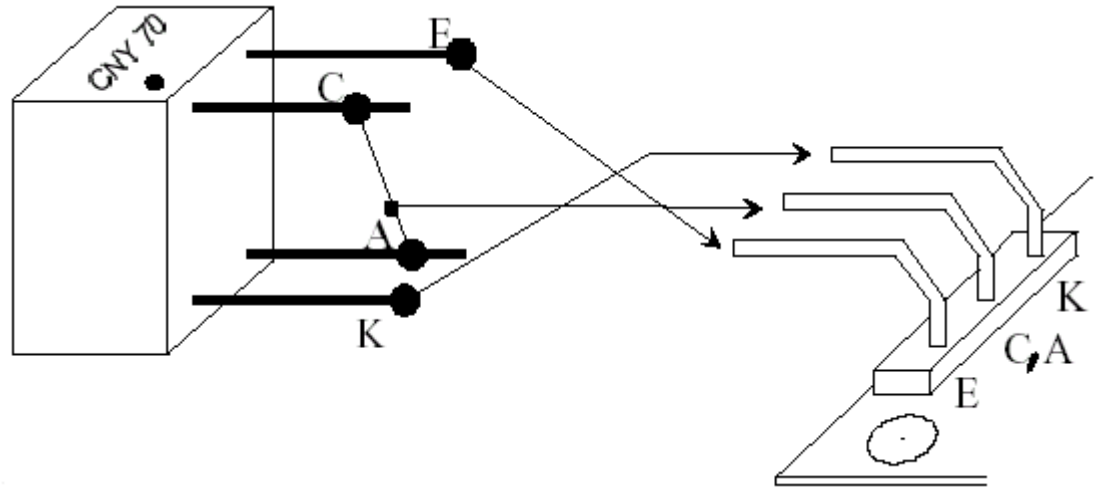


Figura 3: Conexión de los sensores de infrarrojos a la CT293+

3 Los discos

Los discos se han construido imprimiendo en un papel ligeramente grueso (el de 80 gramos que se utiliza normalmente para fotocopiadoras e impresoras es suficiente, aunque también se podría utilizar una cartulina) la plantilla mostrada en la Figura 3, utilizando una impresora láser, ya que las de chorro de tinta o matriciales pueden no cubrir bien toda la zona negra o provocar que se tenga problemas en la detección del color negro con los sensores.

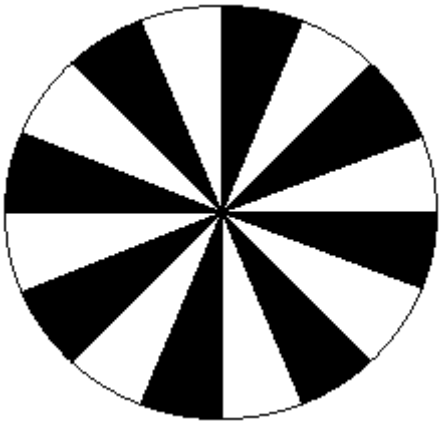


Figura 3: Disco para los encoders de 16 segmentos

El disco utilizado tiene 8 franjas blancas y 8 negras, aunque normalmente no se habla del número de franjas negras sino de las totales (negras y blancas). Por tanto el borde o perímetro del disco estará dividido en 16 segmentos.

También se podrían utilizar discos con un mayor número de segmentos pero eso depende del campo de visión de los sensores utilizados. En el caso de los CNY70, éste es muy amplio y dado el diámetro de los discos puede que no se detecten bien los cambios. El de 24 segmentos es el mayor que se puede utilizar en estas condiciones sin que se produzcan pérdidas en las detecciones.

Una vez impreso el disco se comprueba si se detectan bien los cambios de negro a blanco y viceversa, y en caso de necesidad se repasa el disco con un rotulador negro.

El diámetro del disco de la imagen es aproximadamente de unos 5.5 centímetros. Si este diámetro es demasiado pequeño para las ruedas con las que se está trabajando, pueden construirse otros discos mayores con cualquier utilidad o programa gráfico.

4 Montaje

Una vez cortados los discos, se deben pegar por la parte interna de las ruedas intentando que sus centros coincidan con los ejes de las ruedas. Para ello se recomienda desmontar la rueda y no cortar el disco hasta que no se tenga pegado a ésta.

Los sensores se sujetan a la estructura del robot, bien con cinta aislante, bien de una forma más robusta. Como este paso es dependiente de la plataforma que se esté utilizando, sólo se indica como recomendación que se aproxime lo máximo posible al disco sin que entre en contacto con él para mejorar la detección durante el uso.

Las conexiones de los sensores se podrán hacer en cualquier entrada que quede libre en la tarjeta CT293+, que están indicadas en la placa como Sensor 1, Sensor 2, Sensor 3 y Sensor 4. La conexión a una entrada en particular hace que se conecte a un BIT del puerto A del microcontrolador MC68HC11. Normalmente (verifíquese este punto en cada robot) las entradas de la CT293+ están asociadas con los bits del puerto A que se especifican en la siguiente tabla.

Sensor i	*Bit del Puerto A*
1	0
2	1
3	7
4	3

Fig. 4: Entradas sensoriales de la CT293+

Para la conexión, lo único que debemos tener en cuenta es que los bits 0, 1 y 2 del puerto A del microcontrolador MC68HC11 están asociados con tres capturadores de entrada, que nos podrán ser de gran utilidad a la hora de capturar las señales que llegan de los sensores. El BIT 7 del puerto A también está asociado con otro recurso interno del microcontrolador: el acumulador de pulsos.

5 Programación

Como ya se ha comentado, para la programación del los encoders se pueden utilizar, bien los capturadores de entradas, bien el acumulador de pulsos[1]. Este ultimo recurso es muy interesante para aplicaciones como la que estamos considerando. Como un acumulador de pulsos tiene asociado un contador que se va incrementando automáticamente con la llegada de nuevos pulsos, la cuenta de los pasos en el giro de las ruedas se hará de forma casi automática. Desafortunadamente el MC68HC11 tiene un solo acumulador de pulsos.

La programación del acumulador de pulsos es muy sencilla. El modo en el que se utilizará el acumulador de pulsos junto con los encoders, es aquel en el cual la señal de reloj que incrementa el contador se toma del exterior a través del pin asociado al BIT 7 del Puerto A (salida del sensor del encoder). Cada vez que se detecta un flanco activo (de subida o bajada según se configure) se incrementa el contador en una unidad y se produce una interrupción, si ésta está permitida. Cuando el contador llega a $FF se produce una interrupción de overflow que permite extender el rango del contador y vuelve a comenzar desde cero.

Un capturador de entrada genera una interrupción cada vez que detecta un flanco en la señal que tiene asociada. El flanco que genera la interrupción es configurable, ésta se puede producir por un flanco de subida, de bajada, o por ambos. Con lo que permiten al programa en ejecución realizar la cuenta de los pasos que dan las ruedas.

Para obtener información detallada acerca de la programación de los capturadores y del acumulador de pulsos se remite al lector al libro del Microcontrolador 68HC11 *[Pdf1]*, disponible en la página web de Microbótica *[web1]*.

[1]**Nota:** Para la realización del presente proyecto se ha utilizado el acumulador de pulsos para programar el encoder, puesto que así, aunque solo se controla el giro de una rueda, esto se considera suficiente para comprobar la utilidad del radio-enlace en los movimientos controlados del Microbot, además para el Microbot Clónico, el giro de las ruedas debe similar para todas, de producirse éste correctamente.

ANEXO C. Descripción de funciones y subrutinas

1 Introducción

En el presente documento se describen las funciones y procedimientos más significativos de los presentes en los programas en C presentados en al parte de Diseño SW del Capítulo 4 (*interfaz.c*, *protocol.h*, y *com.h*). Asimismo, también se detallan las subrutinas utilizadas en el programa en ensamblador que ejecuta el Microbot (*proyecto.asm*).

2 Funciones utilizadas en el PC

A continuación se detalla para las principales funciones y procedimientos implicados en los programas realizados: su función, módulo al que pertenecen, algoritmo, entradas, salidas, funciones a las que llaman, y desde donde son llamadas.

Hay que recordar que a diferencia de otros lenguajes de programación, en C no existe el tipo *Booleano*, con lo que para indicar FALSE se utiliza el '0' y para indicar TRUE cualquier valor distinto de '0'. En los programas realizados se ha definido la constante FALSE como '0' y la constante TRUE como '1'.

2.1 enviar.

unsigned short int enviar(unsigned short int dir,unsigned short int dato)

Función

Transmitir un dato a la dirección especificada.

Modulo al que pertenece

com.h

Algoritmo

- Espera que la señal que indica que se acaba de recibir un dato se desactive, si esta no lo hace en un tiempo máximo la función termina y retorna un '0'.
- Inicia la transmisión de un dato según el ciclo de transmisión descrito en el diseño HW de la tarjeta conectada al PC en el capítulo 4.

Entradas

- unsigned short int dir: Dirección destino de la transmisión (4 bits).
- unsigned short int dato: Dato a transmitir (4 bits).

Salidas

- Retorna '1' si la transmisión se ha podido realizar satisfactoriamente y '0' en caso contrario.

Llamada por

enviarNibble en com.h

2.3 recibir.

unsigned short int recibir()

Función

Leer el último dato recibido.

Modulo al que pertenece

com.h

Algoritmo

- Lee un dato de la tarjeta conectada al PC según se describe en el diseño HW de dicha tarjeta en el capítulo 4.
- Pone la variable global dato_recibido a '0'.

Salidas

Retorna el dato leído.

Llamada por

enviarNibble en com.h

recibirNibble en com.h

2.2 datoListo.

unsigned short int datoListo();

Función

Conocer si se ha recibido algún dato que esté pendiente de leerse.

Modulo al que pertenece

com.h

Algoritmo

Devuelve el estado de la variable global 'dato_recibido'.

Salidas

Retorna 0 si el valor de 'dato_recibido' es '0', y '1' en caso contrario.

Llamada por

recibirNibble en com.h

2.4 enviarNibble.

unsigned short int enviarNibble(unsigned short int dir, unsigned short int comando);

Función

Transmitir un dato controlando que la transmisión finalice correctamente.

Modulo al que pertenece

com.h

Algoritmo

- Comienza la transmisión del dato.

- Espera a detecta el fin de la transmisión. Si transcurre el tiempo determinado por la constante TETX y no se ha detectado aún el final de la transmisión termina la función retornando un ‘0’. En caso de que la transmisión finalice correctamente comprueba que el dato a la salida del codificador coincide con el transmitido, si es así retorna un ‘1’, si no retorna un ‘0’.

Entradas

- unsigned short int dir: Dirección destino de la transmisión (4 bits).

- unsigned short int dato: Dato a transmitir (4 bits).

Salidas

- Retorna ‘1’ si la transmisión finalizado satisfactoriamente y ‘0’ en caso contrario.

Llama a

enviar() en com.h

Llamada por

comandoMov en protocol.h

parar en protocol.h

peticion en protocol.h

pasos en protocol.h

2.5 recibirNibble.

unsigned short int recibirNibble(unsigned short int *dato);

Función

Esperar a que se produzca la recepción de un dato, y leerlo en caso de que éste se reciba en un tiempo inferior al definido por la constante TERX.

Modulo al que pertenece

com.h

Algoritmo

Se espera a recibir un dato, y si éste se recibe antes de transcurrido el tiempo límite (TERX) se almacena donde apunta el puntero ‘ *dato ‘ y se retorna un ‘1’. En caso de transcurrir dicho tiempo y no haberse recibido ningún dato se retorna un ‘0’.

Entradas

- unsigned short int *dato: puntero a la posición de memoria donde se ha de almacenar el dato recibido.

Salidas

- Retorna ‘1’ si se recibe un dato antes de transcurrido el tiempo determinado por la constante TERX, y ‘0’ en caso contrario.

Llama a

r*ecibir* en com.h

datoListo en com.h

Llamada por

comandoMov en protocol.h

parar en protocol.h

peticion en protocol.h

pasos en protocol.h

2.6 getDirLocal.

unsigned short int getDirLocal();

Función

Obtener la actual dirección local del PC.

Modulo al que pertenece

com.h

Algoritmo

Retorna el valor de la variable global 'dir_local'.

Salidas

Retorna el valor de 'dir_local'

Llamada por

main() en interfaz.c

2.7 setDirLocal.

void setDirLocal(unsigned short int dir);

Función

Establecer una nueva dirección local para el PC.

Modulo al que pertenece

com.h

Algoritmo

Cambia el valor de la variable global 'dir_local' al indicado por el parametro 'dir'.

Entradas

- unsigned short int dir: nueva dirección local del PC.

Llamada por

main() en interfaz.c

2.8 inicializar.

void inicializar();

Función

Inicializar el PC y la tarjeta transmisora/receptora conectada a éste para poder empezar a utilizar esta última para transmitir y recibir datos.

Modulo al que pertenece

com.h

Algoritmo

- Busca la dirección en memoria de los registros del Puerto Paralelo (DATA, CONTROL y STATUS).
- Inicializa las variables globales 'transmitiendo' y 'dato_recibido'.
- Configura al PP para que a través del pin Ack se genere una interrupción y coloca a la tarjeta en recepción.
- Configura la tarjeta con la dirección local del PC.
- Salva el anterior vector de interrupción correspondiente al PP.
- Establece el nuevo vector de interrupción del PP, apuntando a la rutina de tratamiento de la interrupción diseñada.
- Habilita la interrupción del PP.

Llamada por

main() en interfaz.c

2.9 terminar.

void terminar();

Función

Restaura para el PC (incluyendo el Puerto Paralelo) el estado anterior a la utilización del sistema de comunicación diseñado.

Modulo al que pertenece

com.h

Algoritmo

- Restaura el anterior vector de interrupción del PP
- Deshabilita la interrupción del PP.
- Deshabilita la interrupción por 'Ack' en el PP.

Llamada por

main() en interfaz.c

2.10 comandoMov.

unsigned short int comandoMov(unsigned short int robot, unsigned short int pasos, unsigned short int dir);

Función

Ordenar al robot que realice un movimiento. Este movimiento puede ser bien un desplazamiento hacia delante o hacia atrás, o bien un giro a derecha o izquierda. Asimismo, también se puede determinar que éste sea indefinido o el correspondiente a un numero concreto de pasos de rueda.

Modulo al que pertenece

protocol.h

Algoritmo

Intenta completar la comunicación donde le indica al robot el tipo de movimiento y el número de pasos que han de girar las ruedas. Esta función trata de obtener una respuesta para cada dato a transmitir al Microbot un número máximo de veces determinado por la constante NUMREP.

Entradas

- unsigned short int robot: dirección del robot.

- unsigned short int pasos: Numero de pasos que han de girar las ruedas (0 para indicar movimiento indefinido). Para el montaje de los encoders que se ha hecho en el prototipo (16 segmentos) 8 pasos equivalen a una vuelta de rueda. En los giros una vuelta de rueda equivale a un giro del Microbot de ± 90º.

- unsigned short int dir: Dirección de movimiento:

0 → ADELANTE

1 → IZQUIERDA

2 → DERECHA

3 → ATRÁS

4 → PARO

Salidas

Esta función retorna:

0 → si la comunicación se ha podido completar satisfactoriamente.

1 → si la comunicación no se ha podido realizar satisfactoriamente.

2 → si la tarjeta transmisora/receptora conectada al puerto paralelo no responde (posible fallo en la conexión).

Llama a

enviarNibble en com.h

recibirNibble en com.h

parar en protocol.h

Llamada por

avanzar en protocol.h

retroceder en protocol.h

girarDerecha en protocol.h

girarIzquierda en protocol.h

2.11 parar.

unsigned short int parar(unsigned short int robot);

Función

Ordena al robot que se detenga.

Modulo al que pertenece

protocol.h

Algoritmo

Intenta completar la comunicación donde le indica al robot que se pare. Esto lo hace un máximo de veces determinado por la constante NUMREP.

Entradas

- unsigned short int robot: dirección del robot.

Salidas

Esta función retorna:

0 → si la comunicación se ha podido completar satisfactoriamente.

1 → si la comunicación no se ha podido realizar satisfactoriamente.

2 → si la tarjeta transmisora/receptora conectada al puerto paralelo no responde (posible fallo en la conexión).

Llama a

enviarNibble en com.h

recibirNibble en com.h

Llamada por

comandoMov en protocol.h

main() en interfaz.c

2.12 peticion.

unsigned short int peticion(unsigned short int robot, unsigned short int option, unsigned short int num, unsigned short int *dato);

Función

Solicitar a un Microbot bien el estado de uno de sus puertos, o bien una conversión Analógico/Digital.

Modulo al que pertenece

protocol.h

Algoritmo

Intenta completar la comunicación donde le solicita al robot el estado de un de sus puertos o el resultado de una conversión A/D. Esta función trata de obtener una respuesta para cada dato a transmitir al Microbot un número máximo de veces determinado por la constante NUMREP.

Entradas

- unsigned short int robot: dirección del robot.
- unsigned short int option: 0 → Lectura de un puerto

  1 → Conversión A/D
- unsigned short int num: número del puerto o del canal A/D.
- unsigned short int *dato: puntero a la posición de memoria donde se ha de almacenar el dato recibido.

Salidas

Esta función retorna:

0 → si la comunicación se ha podido completar satisfactoriamente.

1 → si la comunicación no se ha podido realizar satisfactoriamente.

2 → si la tarjeta transmisora/receptora conectada al puerto paralelo no responde (posible fallo en la conexión).

Llama a

enviarNibble en com.h

recibirNibble en com.h

Llamada por

canalAnalog en protocol.h

puerto en protocol.h

2.13 puerto.

unsigned short int puerto(unsigned short int robot, unsigned short int puerto, unsigned short int *dato);

Función

Solicitar a un robot el estado de los bits del puerto seleccionado.

Modulo al que pertenece

protocol.h

Algoritmo

Solicita al Microbot la lectura de uno de sus puertos a través de la función *peticion*.

Entradas

- unsigned short int robot: dirección del robot.
- unsigned short int puerto: puerto del Microbot.

0 → Puerto A

1 → Puerto B

2 → Puerto C

3 → Puerto D

4 → Puerto E

- unsigned short int *dato: puntero a la posición de memoria donde se ha de almacenar el dato recibido.

Salidas

Esta función retorna:

0 → si la comunicación se ha podido completar satisfactoriamente.

1 → si la comunicación no se ha podido realizar satisfactoriamente.

2 → si la tarjeta transmisora/receptora conectada al puerto paralelo no responde (posible fallo en la conexión).

Llama a

petición en protocol.h

Llamada por

main() en interfaz.c

2.14 canalAnalog.

unsigned short int canalAnalog(unsigned short int robot, unsigned short int canal, unsigned short int *dato);

Función

Solicitar a un robot el valor de la conversión analógica/digital del canal especificado.

Modulo al que pertenece

protocol.h

Algoritmo

Solicita el valor de la conversión analógica/digital de un canal a través de la función *petición.*

Entradas

- unsigned short int robot: dirección del robot.
- unsigned short int canal: canal analógico del Microbot (entre 0 y 7).
- unsigned short int *dato: puntero a la posición de memoria donde se ha de almacenar el dato recibido.

Salidas

Esta función retorna:

0 → si la comunicación se ha podido completar satisfactoriamente.

1 → si la comunicación no se ha podido realizar satisfactoriamente.

2 → si la tarjeta transmisora/receptora conectada al puerto paralelo no responde (posible fallo en la conexión).

Llama a

enviarNibble en com.h

recibirNibble en com.h

Llamada por

petición en protocol.h

main() en interfaz.c

2.15 pasos.

unsigned short int pasos(unsigned short int robot, unsigned short int *dato);

Función

Conocer el número de pasos de rueda que le restan para finalizar el último movimiento ordenado desde el PC, con objeto de saber el número de vueltas que han dado las ruedas.

Modulo al que pertenece

protocol.h

Algoritmo

Intenta completar la comunicación donde le solicita al robot el numero de pasos de rueda restantes para finalizar un movimiento. Esta función trata de obtener una respuesta para cada dato a transmitir al Microbot un número máximo de veces determinado por la constante NUMREP.

Entradas

- unsigned short int robot: dirección del robot.

- unsigned short int *dato: puntero a la posición de memoria donde se ha de almacenar el número de pasos recibido.

Salidas

Esta función retorna:

0 → si la comunicación se ha podido completar satisfactoriamente.

1 → si la comunicación no se ha podido realizar satisfactoriamente.

2 → si la tarjeta transmisora/receptora conectada al puerto paralelo no responde (posible fallo en la conexión).

Llama a

enviarNibble en com.h

recibirNibble en com.h

Llamada por

main() en interfaz.c

2.16 changeMask.

unsigned short int changeMask(unsigned short int robot);

Función

Cambiar el valor de la mascara que determina el bit de control de la siguiente transmisión hacia el Microbot correspondiente. Esto se hace después de transmitir un dato a un robot y recibir respuesta por parte de éste, para así poder avanzar en la comunicación.

Modulo al que pertenece

protocol.h

Algoritmo

Si la mascara es ‘1000’ (bit control = ‘1’) → mascara = ‘0000’ (bit control = ‘0’)

Si la mascara es ‘0000’ (bit control = ‘0’) → mascara = ‘1000’ (bit control = ‘1’)

Entradas

- unsigned short int robot: robot con el que se ha completado la transmisión de un dato.

Llamada por

c*omandoMov* en protocol.h

petición en protocol.h

parar en protocol.h

pasos en protocol.h

2.17 dibujarPunto.

void dibujarPunto(short int sentido, unsigned short int pasos);

Función

Dibujar la nueva posición del Microbot en la gráfica tras un avance o retroceso.

Modulo al que pertenece

interfaz.c

Algoritmo

- Para cada paso avanzado dibuja un punto en la dirección que ha seguido el robot.
- Si el robot sobrepasa los limites de grafica desplaza la grafica para seguir viendo la posición del robot.
- Se marca el punto donde esta actualmente el robot con un color distinto del de los puntos que indican la ruta seguida.

Entradas

- short int sentido: avance o retroceso.

1 → Adelante

-1 → Atrás

- unsigned short int pasos: número de pasos de rueda.

Llamada por

main() en interfaz.c

2.18 girarPunto.

void girarPunto(short int sentido, unsigned short int pasos);

Función

Establecer el nuevo rumbo del Microbot en la gráfica tras un giro.

Modulo al que pertenece

interfaz.c

Algoritmo

Actualiza para cada paso la variable global 'dir', que indica el rumbo actual del robot.

Entradas

- short int sentido: sentido del giro.

1 → Derecha

-1 → Izquierda

- unsigned short int pasos: numero de pasos de rueda de giro.

Llamada por

main() en interfaz.c

3 Subrutinas utilizadas en el MICROBOT.

3.1 enviar.

Función

Transmitir un dato al PC.

Modulo al que pertenece

proyecto.asm

Algoritmo

-Llama a la subrutina delay (que genera un retardo).

- Transmite el dato según el ciclo de transmisión descrito en el diseño HW de la tarjeta conectada al Microbot en el capítulo 4.

Entradas

- **Acumulador B**: Dato a transmitir (4 bits).

Llama a

delay

Llamada por

proyecto.asm

3.2 recibir.

Función

Espera a que se reciba un dato, leyéndolo de la tarjeta cuando esto se produce.

Modulo al que pertenece

proyecto.asm

Algoritmo

- Espera a que se reciba un dato.
- Lee un dato de la tarjeta conectada al Microbot según se describe en el diseño HW de dicha tarjeta en el capítulo 4.
- Pone la variable dato_recibido a 0.

Salidas

Retorna en el **Acumulador B** el dato recibido (4 bits).

Llamada por

proyecto.asm

3.3 delay.

Función

Genera el retardo necesario antes de comenzar una transmisión (esto solo es necesario realizarlo cuando se va a transmitir justo después de una recepción).

Modulo al que pertenece

proyecto.asm

Algoritmo

Genera un retardo de 32,5 ms.

Llamada por

enviar

3.4 unmask.

Función

Para un dato recibido, separa su parte significativa (3 bits) del bit de control incluido en él.

Modulo al que pertenece

proyecto.asm

Algoritmo

- Transfiere el dato en el acumulador B al A.
- Se queda en B solo con los 3 bits menos significativos.
- Se queda en A solo con el cuarto bit, que representa el bit de control (pone los demás a 0).

Entradas

- **Acumulador B**: Dato recibido (4 bits).

Salidas

- **Acumulador B**: Parte significativa del dato recibido (3 bits).
- **Acumulador A**: Mascara que determina el bit de control incluido en dato recibido.

Llamada por

proyecto.asm

REFERENCIAS BIBLIOGRÁFICAS

[web1]. "www.microbotica.es".

[web2] "www.beyondlogic.org/spp/parallel.htm".

[web3] "www.beyondlogic.org/interrupts/interupt.htm".

[Micro1] "Manual de Microbótica", Microbótica S.L., 1997.

[Elek.98] 418/433 MHz comunicación de pequeño alcance, Revista Elektor Octubre 98.

[Elek.94] Placa para microprocesador 68HC11, Revista Elektor Abril de 1994.

[HER] José Mª. Hernando Rábanos, "Transmisión Por Radio", E.T.S. Ingenieros de Telecomunicación, Madrid, 1987.

[García.B.96] Carmen García Verdones, "Tutorial de Labwindows", *Una guía para el laboratorio de instrumentación*, DTE 2/96.

[HID] Juan Hidalgo Crestemayer, "Curso Basico de C", Revista Microsoft para programadores Junio 1993.

[PFC1] Manuel González Martín, "Sistema de Comunicación Vía Radio Entre PC y Microbot".

[Moto1] "MC145026,MC145027,MC145028: Encoder and Decoder Pair", MOTOROLA, *Semiconductor technical data book.*

[Moto2] MC68HC11A8 Programming Reference Guide, MOTOROLA.

[Moto3]. *M68HC11E Family - Technical Data*, MOTOROLA.

[Pdf1] (**libro6811.pdf**) "MICROCONTROLADOR 68HC11: Fundamentos, recursos y programación", Microbótica S.L.

[Pdf2] (**encoders.pdf**) Javier de Lope Asiaín, "ENCODERS: Construcción, Montaje y Programación", Departamento de Sistemas Inteligentes Aplicados. Universidad Politécnica de Madrid. 15 de noviembre de 2000.